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 ?

Amazon Aurora DSQL est un service de base de données relationnelle distribué sans serveur optimisé pour les charges de travail transactionnelles. Aurora DSQL offre une évolutivité 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 de 99,999 % dans plusieurs régions.

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. Parce qu'il est sans serveur, Aurora DSQL est idéal pour les modèles d'applications d'architectures de microservices, sans serveur et pilotées par des événements. Aurora DSQL est compatible avec PostgreSQL. Vous pouvez donc utiliser des pilotes, des mappages relationnels objets (), des frameworks et des fonctionnalités SQL courants. ORMs

Aurora DSQL gère automatiquement l'infrastructure du système et adapte le calcul, les E/S et le stockage en fonction de votre charge de travail. Comme vous n'avez aucun serveur à provisionner ou à gérer, vous n'avez pas à vous soucier des interruptions de maintenance liées au provisionnement, à l'application de 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 actif-actif automatise la reprise en cas de panne, vous n'avez donc pas à vous soucier du basculement de base de données traditionnel. Vos applications bénéficient d'une disponibilité multi-AZ et multirégionale, et vous n'avez pas à vous soucier de la cohérence éventuelle ou des données manquantes 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

- Informatique 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 en savoir plus sur la manière dont cette architecture prend en charge la haute disponibilité, consultezRésilience dans Amazon Aurora DSQL.

Clusters monorégionaux et multirégionaux

Les clusters Aurora DSQL offrent les avantages suivants :

- Réplication synchrone des données
- Opérations de lecture cohérentes
- Reprise automatique en cas de panne
- Cohérence des données entre plusieurs régions AZs ou 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'isolation et de durabilité (ACID) avec une cohérence, une isolation des instantanés, une atomicité et une durabilité entre les AZ et entre les régions.

Les clusters homologues multirégionaux offrent la même résilience et la même connectivité que les clusters mono-régionaux. Mais ils améliorent la disponibilité en proposant deux points de terminaison régionaux, un dans chaque région de cluster apparenté. Les deux points de terminaison d'un cluster pair 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 simultanément dans plusieurs régions pour des raisons de performance et de résilience, tout en sachant que les lecteurs voient toujours les mêmes données.

Compatibilité avec les bases de données 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 sous-ensemble de fonctionnalités, d'expressions et

de types de données PostgreSQL. Pour plus d'informations sur les fonctionnalités SQL prises en charge, consultezCompatibilité 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 ont besoin de concevoir des systèmes de base de données hautement disponibles et distribués dans le monde entier doivent souvent comprendre le soutien apporté par les régions à leurs charges de travail. Les cas d'utilisation courants incluent la configuration de la reprise après sinistre entre 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.

Points de terminaison Régions AWS et terminaux pris en charge

Nom de la région	Région	Point de terminaison	Protocole
USA Est (Virginie du Nord)	us-east-1	dsql.us-east-1.api.aws	HTTPS
USA Est (Ohio)	us-east-2	dsql.us-east-2.api.aws	HTTPS
USA Ouest (Oregon)	us-west-2	dsql.us-west-2.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
Asie-Pacifique (Tokyo)	ap-northeast-1	dsql.ap-northeast-1.api.aws	HTTPS
Asie-Pacifique (Séoul)	ap-northeast-2	dsql.ap-northeast-2.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

Disponibilité des clusters multirégionaux 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 multirégional.

Régions des États-Unis

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

Régions Asie-Pacifique

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

Régions européennes

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

Limitations importantes

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

> **Important**
>
> Aurora DSQL ne prend actuellement pas en charge les clusters multirégionaux intercontinentaux.

Tarification d'Aurora DSQL

Pour plus d'informations sur les coûts, consultez la section 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 :

- Commencer à utiliser Aurora DSQL
- Compatibilité des fonctionnalités SQL dans Aurora DSQL
- Accès à Aurora DSQL
- Aurora DSQL et PostgreSQL

Commencer à utiliser Aurora DSQL

Amazon Aurora DSQL est une base de données relationnelle distribuée 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 avec votre base de données AWS Management Console et interagirez avec celle-ci à l'aide de `psql` cet utilitaire. À la fin, vous disposerez d'un cluster Aurora DSQL fonctionnel configuré et prêt à être utilisé pour les charges de travail de test ou de production.

Rubriques

Prérequis

Avant de commencer à utiliser Aurora DSQL, assurez-vous de remplir les conditions préalables suivantes :

- Votre identité IAM doit être autorisée à vous connecter au AWS Management Console.
- Votre identité IAM doit répondre à l'un des critères suivants :
  - Accès pour effectuer n'importe quelle action sur n'importe quelle ressource de votre Compte AWS
  - L'autorisation IAM `iam:CreateServiceLinkedRole` et la possibilité d'accéder à l'action de politique IAM `dsql:*`
- Si vous l'utilisez AWS CLI dans un environnement de type Unix, assurez-vous que les versions 3.8+ et `psql` 14+ de Python sont installées. Pour vérifier les versions de votre application, exécutez les commandes suivantes.

```
python3 --version
```

```
psql --version
```

Si vous l'utilisez AWS CLI dans un autre environnement, assurez-vous de configurer manuellement les versions 3.8+ et `psql` 14+ de Python.

- Si vous avez l'intention d'accéder à Aurora DSQL en utilisant AWS CloudShell les versions 3.8+ et `psql` 14+ de Python sont fournies sans configuration supplémentaire. Pour plus d'informations AWS CloudShell, voir Qu'est-ce que c'est AWS CloudShell ? .
- Si vous avez l'intention d'accéder à Aurora DSQL à l'aide d'une interface graphique, utilisez DBeaver ou JetBrains DataGrip. Pour plus d'informations, consultez Accès à Aurora DSQL avec DBeaver et Accès à Aurora DSQL avec JetBrains DataGrip.

Accès à Aurora DSQL

Vous pouvez accéder à Aurora DSQL à l'aide des techniques suivantes. Pour savoir comment utiliser la CLI APIs, et SDKs, voirAccès à Aurora DSQL.

Rubriques

- Accès à Aurora DSQL via AWS Management Console
- Accès à Aurora DSQL à l'aide de clients SQL
- Utilisation du protocole PostgreSQL avec Aurora DSQL

Accès à Aurora DSQL via AWS Management Console

Vous pouvez accéder au AWS Management Console pour Aurora DSQL à l'https://console.aws.amazon.com/dsqladresse.

Accès à Aurora DSQL à l'aide de clients SQL

Aurora DSQL utilise le protocole PostgreSQL. Utilisez votre client interactif préféré en fournissant un jeton d'authentification IAM signé comme mot de passe lors de la connexion à votre cluster. Un jeton d'authentification est une chaîne de caractères unique qu'Aurora DSQL génère dynamiquement à l'aide de AWS Signature Version 4.

Aurora DSQL utilise le jeton uniquement pour l'authentification. Le jeton n'affecte pas la connexion une fois celle-ci établie. Si vous essayez de vous reconnecter à l'aide d'un jeton expiré, la demande

de connexion est refusée. Pour de plus amples informations, veuillez consulter Génération d'un jeton d'authentification dans Amazon Aurora DSQL.

Rubriques

- Accès à Aurora DSQL avec psql (terminal interactif PostgreSQL)
- Accès à Aurora DSQL avec DBeaver
- Accès à Aurora DSQL avec JetBrains DataGrip

Accès à Aurora DSQL avec psql (terminal interactif PostgreSQL)

L'`psql`utilitaire est une interface basée sur un terminal pour PostgreSQL. Il vous permet de saisir des requêtes de manière interactive, de les envoyer à PostgreSQL et de voir les résultats des requêtes. Pour améliorer les temps de réponse aux requêtes, utilisez le client PostgreSQL version 17.

Téléchargez le programme d'installation de votre système d'exploitation depuis la page des téléchargements de PostgreSQL. Pour plus d'informations sur`psql`, consultez https://www.postgresql. org/docs/current/app-psql.htm.

Si vous l'avez déjà AWS CLI installé, utilisez l'exemple suivant pour vous connecter à votre cluster. Vous pouvez soit utiliser AWS CloudShell, ce qui est fourni avec le `psql` préinstallé, soit installer `psql` directement.

```
# 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
```

Accès à Aurora DSQL avec DBeaver

DBeaver est un outil de base de données open source basé sur une interface graphique. Vous pouvez l'utiliser pour vous connecter à votre base de données et la gérer. Pour le télécharger DBeaver, consultez la page de téléchargement sur le site Web de la DBeaver communauté. Les étapes suivantes expliquent comment vous connecter à votre cluster à l'aide de DBeaver.

Pour configurer une nouvelle connexion Aurora DSQL dans DBeaver

1. Choisissez Nouvelle connexion à la base de données.
2. Dans la fenêtre Nouvelle connexion à la base de données, sélectionnez PostgreSQL.
3. Dans l'onglet Paramètres de connexion/Main, choisissez Connect by : Host et entrez les informations suivantes.

   - Hôte : utilisez le point de terminaison de votre cluster.

     Base de données — Entrez `postgres`

     Authentification — Choisissez `Database Native`

     Nom d'utilisateur — Entrez `admin`

     Mot de passe — Générez un jeton d'authentification. Copiez le jeton généré et utilisez-le comme mot de passe.
4. Ignorez les avertissements et collez votre jeton d'authentification dans le champ DBeaverMot de passe.

   > **Note**
   >
   > Vous devez définir le mode SSL dans les connexions client. Aurora DSQL prend en charge`SSLMODE=require`. Aurora DSQL applique la communication SSL côté serveur et rejette les connexions non SSL.
5. Vous devez être connecté à votre cluster et pouvez commencer à exécuter des instructions SQL.

> **Important**
>
> Les fonctionnalités administratives fournies par DBeaver les bases de données PostgreSQL (telles que Session Manager et Lock Manager) ne s'appliquent pas à une base de données

en raison de son architecture unique. Bien qu'accessibles, ces écrans ne fournissent pas d'informations fiables sur l'état ou l'état de la base de données.

Expiration des informations d'authentification pour DBeaver

Les sessions établies restent authentifiées pendant une heure maximum ou jusqu'à ce que les DBeaver connexions soient déconnectées ou expirent. Pour établir de nouvelles connexions, fournissez un jeton d'authentification valide dans le champ Mot de passe des paramètres de connexion. Toute tentative d'ouverture d'une nouvelle session (par exemple, pour répertorier de nouvelles tables ou 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, la nouvelle session échoue et DBeaver invalide toutes les sessions ouvertes précédemment. Gardez cela à l'esprit lorsque vous choisissez la durée de votre jeton d'authentification IAM avec l'`expires-in`option.

Accès à Aurora DSQL avec JetBrains DataGrip

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 JetBrains site Web.

Pour configurer une nouvelle connexion Aurora DSQL dans JetBrains DataGrip

1. Choisissez Nouvelle source de données, puis PostgreSQL.
2. Dans l' Sources/General onglet Données, 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 — Fournissez-le en générant 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 chargePGSSLMODE=require. Aurora DSQL applique la communication SSL côté serveur et rejette les connexions non SSL.

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 à une base de données en raison de son architecture unique. Bien qu'accessibles, ces écrans ne fournissent pas d'informations fiables sur les sessions réellement connectées à la base de données.

Expiration des informations d'authentification

Les sessions établies restent authentifiées pendant une heure maximum ou jusqu'à ce qu'une déconnexion explicite ou un délai d'attente côté client se produise. Si de nouvelles connexions doivent être établies, un nouveau jeton d'authentification doit être généré et fourni dans le champ Mot de passe des propriétés de la source de données. Toute tentative d'ouverture d'une nouvelle session (par exemple, pour répertorier de nouvelles tables ou 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.

Utilisation du protocole PostgreSQL avec Aurora DSQL

PostgreSQL utilise un protocole basé sur des messages pour la communication entre les clients et les serveurs. Le protocole est pris en charge sur TCP/IP et également sur les sockets du domaine Unix. Le tableau suivant montre comment Aurora DSQL prend en charge le protocole PostgreSQL.

PostgreSQL	Aurora SQL	Remarques
Rôle (également appelé utilisateur ou groupe)	Rôle de base de données	Aurora DSQL crée pour vous un rôle nomméadmin. Lorsque vous créez des rôles de base de données personnalisés, vous devez les admin utiliser pour les associer à des rôles IAM afin de vous authentifier lors de la connexion à votre cluster. Pour plus d'informations, voir Configurer des rôles de base de données personnalisés.
Hôte (également appelé hostname ou hostspec)	Point de terminaison de cluster	Les clusters à région unique 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.
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 require SSL. 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 Génération d'un

PostgreSQL	Aurora SQL	Remarques
		jeton d'authentification dans Amazon Aurora DSQL.

Étape 1 : Création d'un cluster Aurora DSQL à région unique

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

Pour créer un cluster à région unique 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 Create cluster, puis Single-Region.
3. (Facultatif) Dans les paramètres du cluster, sélectionnez l'une des options suivantes :
   - Sélectionnez Personnaliser les paramètres de chiffrement (avancés) pour choisir ou créer un AWS KMS key.
   - Sélectionnez Activer la protection contre la suppression pour empêcher une opération de suppression de votre cluster. Par défaut, la protection contre la suppression est sélectionnée.
4. (Facultatif) Dans Balises, choisissez ou entrez une balise pour ce cluster.
5. Choisissez Créer un cluster.

Étape 2 : Connectez-vous à votre cluster Aurora DSQL

Un point de terminaison de cluster est automatiquement généré lorsque vous créez un cluster Aurora DSQL en fonction de son ID de cluster et de sa région. Le format de dénomination est`clusterid`.`dsql`.`region`.`on`.`aws`. Un client utilise le point de terminaison pour créer une connexion réseau avec votre cluster.

L'authentification est gérée via IAM, vous n'avez donc pas besoin de stocker les informations d'identification dans la base de données. Un jeton d'authentification est une chaîne de caractères unique générée dynamiquement. Le jeton est uniquement utilisé pour l'authentification et n'affecte pas la connexion une fois celle-ci établie. Avant de tenter de vous connecter, assurez-vous que votre identité IAM dispose de l'`dsql:DbConnectAdmin`autorisation requise, comme décrit dansPrérequis.

> **Note**
>
> Pour optimiser la vitesse de connexion à la base de données, utilisez le client PostgreSQL version 17 et `PGSSLNEGOTIATION` configurez-le sur direct :. `PGSSLNEGOTIATION=direct`

Pour vous connecter à votre cluster à l'aide d'un jeton d'authentification

1. Dans la console Aurora DSQL, choisissez le cluster auquel vous souhaitez vous connecter.
2. Choisissez Se connecter.
3. Copiez le point de terminaison depuis le point de terminaison (hôte).
4. Assurez-vous que vous vous connectez en tant qu'administrateur est sélectionné dans la section Jeton d'authentification (mot de passe).
5. Copiez le jeton d'authentification généré. Ce jeton est valide pendant 15 minutes.
6. Sur la ligne de commande du système d'exploitation, utilisez la commande suivante pour démarrer `psql` et vous connecter à votre cluster. *`your_cluster_endpoint`*Remplacez-le par le point de terminaison du cluster que vous avez copié précédemment.

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

   Lorsque vous êtes invité à saisir un mot de passe, entrez le jeton d'authentification que vous avez copié précédemment. Si vous essayez de vous reconnecter à l'aide d'un jeton expiré, la demande de connexion est refusée. Pour de plus amples informations, veuillez consulter Génération d'un jeton d'authentification dans Amazon Aurora DSQL.
7. Appuyez sur Entrée. Vous devriez voir une invite PostgreSQL s'afficher.

   ```
   postgres=>
   ```

   Si vous recevez un message d'erreur de refus d'accès, assurez-vous que votre identité IAM dispose de l'`dsql:DbConnectAdmin`autorisation requise. Si vous êtes autorisé et que vous continuez à recevoir des erreurs de refus d'accès, consultez Résoudre les problèmes liés à l'IAM et Comment puis-je résoudre les problèmes liés aux refus d'accès ou aux erreurs de fonctionnement non autorisées avec une politique IAM ? .

Étape 3 : Exécuter des exemples de commandes SQL dans Aurora DSQL

Testez votre cluster Aurora DSQL en exécutant des instructions SQL. Les exemples d'instructions suivants nécessitent les fichiers de données nommés `department-insert-multirow.sql` et`invoice.csv`, que vous pouvez télécharger depuis le référentiel aurora-dsql-samplesaws-samples/ sur. GitHub

Pour exécuter des exemples de commandes SQL dans Aurora DSQL

1. Créez un schéma nommé`example`.

```
CREATE SCHEMA example;
```

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

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

3. Créez un index secondaire qui utilise la table vide.

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

4. Créez un tableau des départements.

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

5. Utilisez la commande `psql \include` pour charger le fichier nommé `department-insert-multirow.sql` que vous avez téléchargé depuis le dépôt aurora-dsql-samplesaws-samples/ sur. GitHub Remplacez *`my-path`* par le chemin d'accès à votre copie locale.

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

6. Utilisez la commande `psql \copy` pour charger le fichier nommé `invoice.csv` que vous avez téléchargé depuis le dépôt aurora-dsql-samplesaws-samples/ sur. GitHub Remplacez *`my-path`* par le chemin d'accès à votre copie locale.

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

7. Interrogez les départements et triez-les en fonction de leurs ventes totales.

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

L'exemple de résultat suivant montre que le département 3 enregistre le plus de ventes.

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

Étape 4 : Création d'un cluster multirégional

Lorsque vous créez un cluster multirégional, vous spécifiez les régions suivantes :

Région éloignée

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 multirégional. 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.

L'exemple suivant montre comment 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 régions et les lectures cohérentes à partir des deux points de terminaison régionaux.

Pour créer un cluster multirégional

1. Connectez-vous à la console Aurora AWS Management Console DSQL et ouvrez-la à https://console.aws.amazon.com/dsql l'adresse.
2. Dans le panneau de navigation, choisissez Clusters.
3. Choisissez Créer un cluster, puis Multi-région.
4. (Facultatif) Dans les paramètres du cluster, sélectionnez l'une des options suivantes pour votre cluster initial :
   - Sélectionnez Personnaliser les paramètres de chiffrement (avancés) pour choisir ou créer un AWS KMS key.
   - Sélectionnez Activer la protection contre la suppression pour empêcher une opération de suppression de votre cluster. Par défaut, la protection contre la suppression est sélectionnée.
5. Dans les paramètres multirégionaux, choisissez les options suivantes pour votre cluster initial :
   - Dans Witness Region, choisissez une région. Actuellement, seules les régions basées aux États-Unis sont prises en charge pour les régions témoins dans les clusters multirégionaux.
   - (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 multirégional, terminez la configuration après avoir créé le cluster initial.
6. (Facultatif) Choisissez des balises pour votre cluster initial.
7. Choisissez Create cluster pour créer votre cluster initial. Si vous n'avez pas saisi d'ARN à l'étape précédente, la console affiche la notification de configuration du cluster en attente.
8. Dans la notification Configuration du cluster en attente, choisissez Compléter la configuration du cluster multirégional. Cette action initie la création d'un deuxième cluster dans une autre région.
9. Choisissez l'une des options suivantes pour votre deuxième cluster :

- Ajouter un ARN de cluster de régions distantes : choisissez cette option s'il existe un cluster et que vous souhaitez qu'il s'agisse du deuxième cluster de votre cluster multirégional.
- 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.

10. 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.
11. (Facultatif) Choisissez les paramètres de cluster pour votre deuxième cluster. Par exemple, vous pouvez choisir un AWS KMS key.
12. Choisissez Create cluster pour créer votre deuxième cluster.
13. Choisissez Peer in ***initial-cluster-region***, où se *initial-cluster-region* trouve la région qui héberge le premier cluster que vous avez créé.
14. Lorsque vous y êtes invité, choisissez Confirmer. Cette étape termine la création de votre cluster multirégional.

Pour vous connecter 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 multirégional.
4. Dans Actions, sélectionnez Ouvrir dans CloudShell.
5. Choisissez Connect en tant qu'administrateur.
6. Choisissez Lancer CloudShell.
7. Cliquez sur Exécuter.
8. Créez un exemple de schéma en suivant les étapes décrites dansÉtape 3 : Exécuter des exemples de commandes SQL dans Aurora DSQL.

   Exemples de transactions

Example

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

9. Utilisez `include` les commandes `psql` `copy` et pour charger des exemples de données. Pour de plus amples informations, veuillez consulter Étape 3 : Exécuter des exemples de commandes SQL dans Aurora DSQL.

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

Pour interroger les données du deuxième cluster depuis 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 multirégional.
4. Dans Actions, sélectionnez Ouvrir dans CloudShell.
5. Choisissez Connect en tant qu'administrateur.
6. Choisissez Lancer CloudShell.
7. Cliquez sur Exécuter.
8. Recherchez les données que vous avez insérées dans le deuxième cluster.

Example

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

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 de l'IAM avec les privilèges de PostgreSQL. Aurora DSQL utilise ces fonctionnalités pour fournir une politique 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 du AWS Management ConsoleAWS CLI, ou du 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 créer un cluster, assurez-vous que votre identité IAM dispose d'autorisations pour l'action IAM`dsql:CreateCluster`, comme dans l'exemple d'action de politique suivant.

```
{
  "Effect": "Allow",
  "Action": "dsql:CreateCluster",
  "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 vous connecter à 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 autorisée à vous connecter à votre cluster. Pour en savoir plus, veuillez consulter la section Génération 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 :

- À utiliser `dsql:DbConnectAdmin` si vous utilisez le `admin` rôle. 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"
}
```

- À utiliser `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 `admin` rôle et des rôles personnalisés. Aurora DSQL crée automatiquement un `admin` rôle prédéfini pour vous dans votre cluster Aurora DSQL. Vous ne pouvez pas modifier le `admin` rôle. Lorsque vous vous connectez à votre base de données en tant que`admin`, vous pouvez émettre du code 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 `admin` rôle 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, veuillez consulter la section Autoriser les rôles de base de données à se connecter à votre cluster.

Autorisation

Utilisez le `admin` rôle pour vous connecter à votre cluster. Exécutez des commandes SQL pour configurer des rôles de base de données personnalisés et octroyer des autorisations. Pour en savoir plus, consultez les rôles de base de données PostgreSQL et les 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 de base de données personnalisé `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 `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 en savoir plus sur les rôles de base de données personnalisés, consultezUtilisation 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 stratégie uniquement pour les actions que votre rôle doit effectuer. Par exemple, si votre rôle a uniquement besoin d'obtenir des informations sur le cluster, vous pouvez limiter les autorisations de rôle aux seules `ListClusters` autorisations `GetCluster` et, 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 stratégie suivant montre toutes les actions de stratégie 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 `admin` rôle, révoquez l'accès de l'identité IAM à. `dsql:DbConnectAdmin` Modifiez la stratégie IAM ou détachez-la 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 la section Quotas et limites.

Révocation de l'autorisation de rôle personnalisée pour se connecter aux clusters

Pour révoquer l'accès aux rôles de base de données autrement que`admin`, révoquez l'accès de l'identité IAM à. `dsql:DbConnect` Modifiez la stratégie IAM ou détachez-la 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 en savoir plus sur la révocation de l'accès à des rôles de base de données, consultezRévocation de l'autorisation de base de données associée à un rôle IAM.

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

Génération 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 authentifier la connexion. Une fois la connexion établie, elle reste valide même si le jeton d'authentification expire.

Si vous créez un jeton d'authentification à l'aide de la AWS console, le jeton expire automatiquement dans une heure par défaut. Si vous utilisez le AWS CLI ou SDKs pour créer le jeton, la valeur par défaut est de 15 minutes. 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 au minimum disposer des autorisations IAM répertoriées dansConnexion 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 depuis la console.

Pour générer 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 Aurora DSQL à région unique ouÉtape 4 : Création d'un cluster multirégional.
3. Choisissez Connect, puis sélectionnez Get Token.
4. Choisissez si vous souhaitez vous connecter en tant que rôle de base de données personnalisé `admin` ou avec un rôle de base de données personnalisé.
5. Copiez le jeton d'authentification généré et utilisez-le pourAccès à Aurora DSQL à l'aide de clients SQL.

Pour en savoir plus sur les rôles de base de données personnalisés et l'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 d'effectuer les opérations suivantes :

- Créez un cluster SQL Aurora.
- Ajoutez une autorisation pour exécuter l'opération Amazon S3 `get-object` afin de récupérer des objets depuis un Compte AWS extérieur de votre organisation. Pour plus d'informations, consultez le guide de l'utilisateur Amazon S3.

Pour générer un jeton d'authentification en utilisant 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. Suivez la section Installation ou mise à jour vers la dernière version du AWS CLI pour installer le AWS CLI.

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

4. Exécutez la commande suivante pour générer un jeton d'authentification pour le `admin` rôle. *`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-le à la `generate-db-connect-auth-token` place.

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

Si vous rencontrez des problèmes, consultez Résoudre les problèmes liés à l'IAM et Comment puis-je résoudre les problèmes liés au refus d'accès ou aux erreurs de fonctionnement non autorisées avec une politique IAM ? .

5. Utilisez la commande suivante pour `psql` établir une connexion à votre cluster.

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

6. 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 ou de caractères supplémentaires. Collez-le dans le formulaire d'invite suivant`psql`.

```
Password for user admin:
```

7. Appuyez sur Entrée. Vous devriez voir une invite PostgreSQL s'afficher.

```
postgres=>
```

Si vous recevez un message d'erreur de refus d'accès, assurez-vous que votre identité IAM dispose de l'`dsql:DbConnectAdmin`autorisation requise. Si vous êtes autorisé et que vous continuez à recevoir des erreurs de refus d'accès, consultez Résoudre les problèmes liés à l'IAM et Comment puis-je résoudre les problèmes liés aux refus d'accès ou aux erreurs de fonctionnement non autorisées avec une politique IAM ? .

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

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

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

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

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

- *`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 en `ACTIVE` état. Les exemples de SDK utilisent les attributs suivants pour générer un jeton d'authentification pour le `admin` rôle :

- *`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

Vous pouvez générer le jeton de différentes manières :

- Si vous vous connectez au `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`.

```
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 de différentes manières :

- Si vous vous connectez au `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) {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQLClient client{clientConfig};
    std::string token = "";

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

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

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

JavaScript SDK

Vous pouvez générer le jeton de différentes manières :

- Si vous vous connectez au `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

Vous pouvez générer le jeton de différentes manières :

- Si vous vous connectez au `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.create())
                .build();

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

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

Rust SDK

Vous pouvez générer le jeton de différentes manières :

- Si vous vous connectez au `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

Vous pouvez générer le jeton de différentes manières :

- Si vous vous connectez au `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::SharedCredentials.new()

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

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

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

.NET

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

Vous pouvez générer le jeton de différentes manières :

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

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

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

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

AWSCredentials credentials = FallbackCredentialsFactory.GetCredentials();

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

Console.WriteLine(token);
```

Golang

> **Note**
>
> Le SDK Golang ne fournit pas de méthode intégrée pour générer un jeton pré-signé. Vous devez créer manuellement la demande signée, comme indiqué dans l'exemple de code suivant.

Dans l'exemple de code suivant, spécifiez le `action` en fonction de l'utilisateur PostgreSQL :

- Si vous vous connectez au `admin` rôle, utilisez l'`DbConnectAdmin`action.
- Si vous vous connectez avec un rôle de base de données personnalisé, utilisez l'`DbConnect`action.

En plus de *`yourClusterEndpoint`* et*`region`*, l'exemple suivant utilise*`action`*. Spécifiez le *`action`* en fonction de l'utilisateur PostgreSQL.

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

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

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

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

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

 return url, nil
}
```

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 méthode pour authentifier et accéder aux 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 assumer temporairement un rôle IAM de plusieurs manières :

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

Après avoir assumé 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 la section Identités IAM dans le guide de l'utilisateur IAM.

Utilisateurs IAM

Un utilisateur IAM est une identité au sein de vous Compte AWS dotée d'autorisations spécifiques et 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 utilisées 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 de la section Utilisation des clés de condition IAM avec Amazon Aurora DSQL.

Autoriser les rôles de 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 `admin` de base de données avec une identité IAM autorisée à effectuer une action IAM afin de vous connecter `dsql:DbConnectAdmin` à votre cluster.

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

   ```
   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. Accorder des autorisations au niveau de la base de données au rôle de base de données

   Les exemples suivants utilisent la `GRANT` commande 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 la section Privilèges de PostgreSQL et de `GRANT` PostgreSQL dans la documentation de PostgreSQL.

Révocation de l'autorisation de base de données associée à un rôle IAM

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

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

Pour en savoir plus sur la révocation de l'autorisation, consultezRévocation 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 requises pour les applications distribuées modernes, natives dans le cloud.

Points forts de la compatibilité avec PostgreSQL

Aurora DSQL est actuellement basé sur la version 16 de PostgreSQL. Les principales compatibilités sont les suivantes :

Protocole filaire

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

Compatibilité avec 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 les suivants :

- Gestion des valeurs nulles
- Comportement des ordres de tri
- Échelle et précision pour les opérations numériques
- Équivalence pour les opérations sur les chaînes

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'isolation équivalent à PostgreSQL Repeatable Read. Pour de plus amples informations, veuillez consulter Contrôle de simultanéité dans Aurora DSQL.

Principales différences architecturales

La conception distribuée et sans partage d'Aurora DSQL présente quelques différences fondamentales par rapport à PostgreSQL traditionnel. Ces différences font partie intégrante de l'architecture Aurora DSQL et offrent de nombreux avantages en termes de performances et d'évolutivité. Les principales différences 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 plus d'exemples, voirContrôle 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 :

- Exécutez les opérations DDL en tant que tâches de fond, afin de minimiser les perturbations.
- Coordonnez 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 sans leader sur plusieurs zones de disponibilité grâce à des couches de calcul et de stockage découplées.

Pour de plus amples informations, veuillez consulter DDL et transactions distribuées dans Aurora DSQL.

Compatibilité des fonctionnalités SQL dans Aurora DSQL

Aurora DSQL et PostgreSQL renvoient des résultats identiques pour toutes les requêtes SQL. Notez qu'Aurora DSQL diffère de PostgreSQL sans clause. `ORDER BY` 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

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

Rubriques

Types de données numériques

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

Nom	Alias	Portée et précision	Taille de stockage	Support de l'index
`smallint`	`int2`	-32768 à +32767	2 bytes	Oui
`integer`	`int`, `int4`	-2147483648 à +2147483647	4 bytes	Oui

Nom	Alias	Portée et précision	Taille de stockage	Support de l'index
`bigint`	`int8`	-9223372036854775808 au +9223372036854775807	8 bytes	Oui
`real`	`float4`	Précision à 6 chiffres décimaux	4 bytes	Oui
`double precision`	`float8`	Précision de 15 chiffres décimaux	8 bytes	Oui

`numeric [ (p, s) ]`	`decimal [ (p, s) ]`
	
	`dec[ (p,s)]`
		Chiffre exact avec une précision sélectionnable. La précision maximale est de 38 et l'échelle maximale 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.	Non

[1]— Si vous ne spécifiez pas explicitement de 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 `INSERT` des `UPDATE` instructions.

Types de données de personnages

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

Nom	Alias	Description	Limite SQL d'Aurora	Taille de stockage	Support de l'index
`character [ (n) ]`	`char [ (n) ]`	Chaîne de caractères de longueur fixe	4096 octets [1]	Variable jusqu'à 4 100 octets	Oui

Nom	Alias	Description	Limite SQL d'Aurora	Taille de stockage	Support de l'index
`character varying [ (n) ]`	`varchar [ (n) ]`	Chaîne de caractères de longueur variable	65535 octets 1	Variable jusqu'à 65539 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 sémantiqu ement insignifiants.	4096 octets 1	Variable jusqu'à 4 100 octets	Oui
`text`		Chaîne de caractères de longueur variable	1 MiB 1	Variable jusqu'à 1 MiB	Oui

1— Si vous ne spécifiez pas explicitement de 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 `INSERT` des `UPDATE` instructions.

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	Alias	Description	Range	Résolution	Taille de stocka	Support de l'index
`date`		Date calendair	4713 AVANT JC — 5874897 AD	1 jour	4 bytes	Oui

Nom	Alias	Description	Range	Résolution	Taille de stocka	Support de l'index
		e (année, mois, jour)				
`time[(p)][without time zone]`	`time`	Heure de la journée, sans fuseau horaire	0 — 1	1 microseconde	8 bytes	Oui
`time[(p)] with time zone`	`timet`	heure de la journée, y compris le fuseau horaire	00:00:00 +1559 — 24:00:00 —1559	1 microseconde	12 octets	Non
`timestamp [(p)][ without time zone]`		Date et heure, sans fuseau horaire	4713 AVANT JC — 294276 ANNONCE	1 microseconde	8 bytes	Oui
`timestamp [(p)]with time zone`	`timestz`	Date et heure, y compris le fuseau horaire	4713 AVANT JC — 294276 ANNONCE	1 microseconde	8 bytes	Oui
`interval [fields][(p)]`		Échelle de temps	-178000000 ans — 178000000 ans	1 microseconde	16 octets	Non

Types de données divers

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

Nom	Alias	Description	Limite SQL d'Aurora	Taille de stockage	Support de l'index
`boolean`	`bool`	Booléen logique (true/false)		1 octet	Oui
`bytea`		Données binaires (« tableau d'octets »)	1 MiB [1]	Limite variable jusqu'à 1 MiB	Non
`UUID`		Identifiant unique universel		16 octets	Oui

[1]— Si vous ne spécifiez pas explicitement de 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 `INSERT` des `UPDATE` instructions.

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 `integer` que `varchar` et 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 lors de l'exécution des requêtes :

Type de matrice

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 le séparateur de virgules (`,`), comme indiqué dans l'exemple suivant. Vous pouvez utiliser des tableaux dans les expressions, les sorties de fonctions ou les calculs temporaires lors de l'exécution de requêtes.

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

La fonction renvoie une réponse similaire à la suivante :

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

type d'entrée

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

SQL pris en charge pour Aurora DSQL

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

SELECT commande

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

Clause principale	Clauses soutenues
`FROM`	
`GROUP BY`	`ALL`, `DISTINCT`
`ORDER BY`	`ASC`, `DESC`, `NULLS`
`LIMIT`	
`DISTINCT`	
`HAVING`	
`USING`	

Clause principale	Clauses soutenues
`WITH`(expressions de table courantes)	
`INNER JOIN`	`ON`
`OUTER JOIN`	`LEFT`, `RIGHT`, `FULL`, `ON`
`CROSS JOIN`	`ON`
`UNION`	`ALL`
`INTERSECT`	`ALL`
`EXCEPT`	`ALL`
`OVER`	`RANK ()`, `PARTITION BY`
`FOR UPDATE`	

Langage de définition des données (DDL)

Aurora DSQL prend en charge les commandes DDL PostgreSQL suivantes.

Command	Clause principale	Clauses compatibles
`CREATE`	`TABLE`	Pour plus d'informations sur la syntaxe prise en charge de la `CREATE TABLE` commande, consultez`CREATE TABLE`.
`ALTER`	`TABLE`	Pour plus d'informations sur la syntaxe prise en charge de la `ALTER TABLE` commande, consultez`ALTER TABLE`.
`DROP`	`TABLE`	

Command	Clause principale	Clauses compatibles
`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 prise en charge de la `CREATE INDEX ASYNC` commande, consultezIndex asynchrones dans Aurora DSQL.
`DROP`	`INDEX`	
`CREATE`	`VIEW`	Pour plus d'informations sur la syntaxe prise en charge de la `CREATE VIEW` commande, consultez`CREATE VIEW`.
`ALTER`	`VIEW`	Pour plus d'informations sur la syntaxe prise en charge de la `ALTER VIEW` commande, consultezALTER VIEW.
`DROP`	`VIEW`	Pour plus d'informations sur la syntaxe prise en charge de la `DROP VIEW` commande, consultez`DROP VIEW`.
`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.

Command	Clause principale	Clauses soutenues
`INSERT`	`INTO`	`VALUES`

Command	Clause principale	Clauses soutenues
		SELECT
UPDATE	SET	WHERE (SELECT) FROM, WITH
DELETE	FROM	USING, WHERE

Langage de contrôle des données (DCL)

Aurora DSQL prend en charge les commandes PostgreSQL DCL suivantes.

Command	Clauses soutenues
GRANT	ON, TO
REVOKE	ON, FROM, CASCADE, RESTRICT

Langage de contrôle des transactions (TCL)

Aurora DSQL prend en charge les commandes TCL PostgreSQL suivantes.

Command	Clauses soutenues
COMMIT	
BEGIN	[WORK | TRANSACTION] [READ ONLY | READ WRITE]

Commandes utilitaires

Aurora DSQL prend en charge les commandes utilitaires PostgreSQL suivantes :

- EXPLAIN
- ANALYZE(nom de relation uniquement)

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

Cette section PostgreSQL fournit des informations détaillées sur les expressions prises en charge, en se concentrant sur les commandes comportant des ensembles de paramètres et des sous-commandes étendus. Par exemple, CREATE TABLE dans PostgreSQL propose de nombreuses clauses et paramètres. Cette section décrit tous les éléments de syntaxe PostgreSQL pris en charge par Aurora DSQL pour ces commandes.

Rubriques

`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 |
  UNIQUE [ NULLS [ NOT ] DISTINCT ] index_parameters |
  PRIMARY KEY index_parameters |

and table_constraint is:
```

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

and like_option is:

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

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

ALTER TABLE

ALTER TABLEmodifie 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
    OWNER TO { new_owner | CURRENT_ROLE | CURRENT_USER | SESSION_USER }
```

CREATE VIEW

CREATE VIEWdéfinit 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 donnant lieu aux colonnes de sortie peuvent être différents.

Si un nom de schéma est donné `CREATE VIEW myschema.myview ...` (tel que), la vue est créée dans le schéma spécifié. Dans le cas contraire, il est créé dans le schéma actuel.

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

Paramètres

`CREATE VIEW`prend en charge divers paramètres pour contrôler le comportement des vues actualisables 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`

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

`column_name`

Liste facultative de noms à utiliser pour les colonnes de la vue. Si ce n'est pas le cas, les noms des 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 l'`local`un ou `cascaded` l'autre et équivaut à spécifier`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. Consultez les notes ci-dessous pour plus de détails.

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, `INSERT` les `UPDATE` commandes 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 le n'`CHECK OPTION`est pas spécifié, `INSERT` les `UPDATE` commandes de la vue sont autorisées à créer des lignes qui ne sont pas visibles dans la vue. Les options de vérification suivantes sont prises en charge.
- `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 le 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 le `CHECK OPTION` est spécifié et que ni l'un `LOCAL` ni l'autre ne `CASCADED` sont spécifiés, alors `CASCADED` c'est supposé.

> **Note**
>
> Ils ne `CHECK OPTION` peuvent pas être utilisés avec des `RECURSIVE` vues. Le n'`CHECK OPTIO`Nest pris en charge que sur les vues automatiquement actualisables.

Remarques

Utilisez la `DROP VIEW` déclaration 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 est par défaut`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 `security_invoker` propriété de la vue est définie sur true, l'accès aux relations de base sous-jacentes est déterminé par les autorisations de l'utilisateur exécutant la requête, plutôt que par le propriétaire de la vue. Ainsi, l'utilisateur d'une vue invoquant la 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'invocateur de sécurité, elle sera traitée comme si elle avait été accessible directement depuis la requête d'origine. Ainsi, une vue invoquant la 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 cette `security_invoker` propriété.
- 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`. Par exemple, un appel `CURRENT_USER` direct dans une vue renverra toujours l'utilisateur appelant,

et non le propriétaire de la vue. Cela n'est pas affecté par le `security_invoker` réglage de la vue, et une vue `security_invoker` définie sur false n'est donc pas équivalente à une `SECURITY DEFINER` fonction.

- L'utilisateur qui crée ou remplace une vue doit disposer de `USAGE` privilèges sur tous les schémas auxquels il est fait référence dans la requête de vue, afin de pouvoir rechercher les objets référencés dans ces schémas. Notez toutefois que cette recherche n'a lieu que lorsque la vue est créée ou remplacée. Par conséquent, l'utilisateur de la vue n'a besoin du `USAGE` privilège que sur le schéma contenant la vue, et non sur les schémas auxquels il est fait référence dans la requête de vue, même pour une vue d'appel de sécurité.
- Lorsqu'elle `CREATE OR REPLACE VIEW` est utilisée sur une vue existante, seule la `SELECT` règle de définition de la vue, ainsi que tous `WITH ( ... )` les paramètres et ses paramètres, `CHECK OPTION` sont modifiés. Les autres propriétés d'affichage, notamment la propriété, les autorisations et les règles de non-sélection, restent inchangées. Vous devez être propriétaire de la vue pour la remplacer (cela inclut le fait d'être membre du rôle propriétaire).

Vues actualisables

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

- La vue doit comporter exactement une entrée dans sa `FROM` liste, qui doit être une table ou une autre vue modifiable.
- La définition de la vue ne doit pas contenir de clauses `WITH` `DISTINCT` `GROUP BYHAVING`,`LIMIT`,,, ni de `OFFSET` clauses au niveau supérieur.
- La définition de la vue ne doit pas contenir d'opérations définies (`UNIONINTERSECT`,, ou`EXCEPT`) au niveau supérieur.
- La liste de sélection de la vue ne doit pas contenir d'agrégats, de fonctions de fenêtre ou de fonctions renvoyant des ensembles.

Une vue actualisable automatiquement peut contenir un mélange de colonnes actualisables et non actualisables. Une colonne est modifiable s'il s'agit d'une simple référence à une colonne modifiable de la relation de base sous-jacente. Dans le cas contraire, la colonne est en lecture seule et une erreur se produit si une `UPDATE` instruction `INSERT` or tente de lui attribuer une valeur.

Pour les vues actualisables automatiquement, le système convertit n'importe quelle `INSERT` `DELETE` instruction de la vue en instruction correspondante sur la relation de base sous-jacente. `UPDATE` `INSERT`les déclarations comportant une `ON CONFLICT UPDATE` clause sont entièrement prises en charge.

Si une vue pouvant être mise à jour automatiquement contient une `WHERE` condition, celle-ci limite les lignes de la relation de base qui peuvent être modifiées par la vue `UPDATE` et les `DELETE` instructions qui y figurent. Cependant, un `UPDATE` utilisateur peut modifier une ligne afin qu'elle ne réponde plus à la `WHERE` condition, la rendant invisible dans la vue. De même, une `INSERT` commande peut potentiellement insérer des lignes de relation de base qui ne satisfont pas à la `WHERE` condition, les rendant ainsi invisibles dans la vue. `ON CONFLICT UPDATE`peut affecter de la même manière une ligne existante qui n'est pas visible dans la vue.

Vous pouvez utiliser `INSERT` les `UPDATE` commandes `CHECK OPTION` pour empêcher la création de lignes invisibles dans la vue.

Si une vue pouvant être mise à jour automatiquement est marquée par la propriété security_barrier, toutes les `WHERE` conditions de la vue (et toutes les conditions utilisant des opérateurs marqués comme`LEAKPROOF`) sont toujours évaluées avant toutes les conditions ajoutées par un utilisateur de la vue. Notez que de ce fait, les lignes qui ne sont finalement pas renvoyées (parce qu'elles ne répondent pas aux `WHERE` conditions de l'utilisateur) peuvent tout de même être verrouillées. Vous pouvez l'utiliser `EXPLAIN` pour voir quelles conditions sont appliquées au niveau de la relation (et donc ne verrouillez pas les lignes) et lesquelles ne le sont pas.

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 ou 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 correspondant sur la vue. Par défaut, le propriétaire de la vue doit disposer des privilèges appropriés sur les relations de base sous-jacentes, tandis que l'utilisateur effectuant 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

Pour créer une vue comprenant tous les films comiques.

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

Cela créera une vue contenant les colonnes présentes dans le `film` tableau au moment de la création de la vue. Bien qu'elles aient `*` été utilisées pour créer la vue, les colonnes ajoutées ultérieurement au tableau ne feront pas partie de la vue.

Créez une vue avec`LOCAL CHECK OPTION`.

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

Cela créera une vue qui vérifie à la fois les nouvelles lignes `kind` et `classification` les nouvelles lignes.

Créez une vue avec un mélange de colonnes actualisables et non actualisables.

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

Ce point de vue soutiendra `INSERT``UPDATE`, et`DELETE`. Toutes les colonnes du tableau des films pourront être mises à jour, tandis que les colonnes calculées `avg_rating` seront `country` en lecture seule.

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

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

> **Note**
>
> Bien que le nom de la vue récursive y soit qualifié de schémaCREATE, son autoréférence interne n'est pas qualifiée de schéma. Cela est dû au fait que le nom de l'expression de table commune (CTE) créée implicitement ne peut pas être qualifié de schéma.

Compatibilité

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

ALTER VIEW

L'`ALTER VIEW`instruction 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 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 vue. 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 vue.)

Paramètres

`ALTER VIEW` paramètres

`name`

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

`column_name`

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 par toute `UPDATE` commande `INSERT` ou dont la cible est la vue. La valeur par défaut de la vue aura priorité sur toutes les valeurs par défaut des relations sous-jacentes.

nouveau_propriétaire

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

nouveau_nom

Le nouveau nom de la vue.

nouvel_schéma

Le nouveau schéma de la vue.

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

Définit ou réinitialise 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. La valeur doit être une valeur booléenne, telle que `true` ou. `false`
- `security_invoker (boolean)`- Modifie la propriété de barrière de sécurité de la vue. La valeur doit être une valeur booléenne, telle que `true` ou. `false`

Remarques

Pour des raisons historiques`ALTER TABLE`, PostgreSQL peut également être utilisé avec des vues ; mais les seules variantes `ALTER TABLE` autorisées avec les vues sont équivalentes à celles présentées précédemment.

Exemples

Renommer la vue `foo` en. `bar`

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

Attacher une valeur de colonne par défaut à une vue actualisable.

```
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'`DROP VIEW`instruction 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 le propriétaire de la vue.

Paramètres

`IF EXISTS`

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

`name`

Le nom (éventuellement qualifié par le schéma) de la vue à supprimer.

`CASCADE`

Supprimez automatiquement les objets qui dépendent de la vue (tels que les autres vues), puis tous les objets qui dépendent de ces objets.

`RESTRICT`

Refusez 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 d'une seule vue par commande, et à part l'`IF EXISTS`option, qui est une extension PostgreSQL prise en charge par Aurora DSQL.

Fonctionnalités PostgreSQL non prises en charge dans Aurora DSQL

Aurora DSQL est compatible avec PostgreSQL. Cela signifie qu'Aurora DSQL prend en charge les fonctionnalités relationnelles de base telles que les transactions ACID, les index secondaires, les jointures, les insertions et les mises à jour. Pour un aperçu des fonctionnalités SQL prises en charge, voir Expressions SQL prises en charge.

Les sections suivantes mettent en évidence les fonctionnalités de PostgreSQL qui ne sont actuellement pas prises en charge dans Aurora DSQL.

Objets non pris en charge

Les objets non pris en charge par Aurora DSQL sont les suivants :

- Plusieurs bases de données sur un seul cluster Aurora DSQL
- Tables temporaires
- Déclencheurs
- Types (support partiel)
- Espaces de table
- Fonctions écrites dans des langages autres que SQL
- Séquences
- Partitions

Contraintes non prises en charge

- Clés étrangères
- Contraintes d'exclusion

Commandes non prises en charge

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

> **Note**
>
> Aurora DSQL ne nécessite pas de mise sous vide. Le système gère les statistiques et gère automatiquement l'optimisation du stockage sans commandes manuelles d'aspiration.

Extensions non prises en charge

Aurora DSQL ne prend pas en charge les extensions PostgreSQL. Le tableau suivant indique les extensions qui ne sont pas prises en charge :

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

Expressions SQL non prises en charge

Le tableau suivant décrit les clauses qui ne sont pas prises en charge dans Aurora DSQL.

Catégorie	Clause principale	Clause non étayée
CREATE	INDEX ASYNC	ASC | DESC
CREATE	INDEX[1]	
TRUNCATE		
ALTER	SYSTEM	Toutes les ALTER SYSTEM commandes sont bloquées.
CREATE	TABLE	COLLATE, AS SELECT, INHERITS, PARTITION
CREATE	FUNCTION	LANGUAGE *non-sql-lang* , où se *non-sql-lang* trouve une langue autre que SQL
CREATE	TEMPORARY	TABLES
CREATE	EXTENSION	
CREATE	SEQUENCE	
CREATE	MATERIALIZED	VIEW

Catégorie	Clause principale	Clause non étayée
CREATE	TABLESPACE	
CREATE	TRIGGER	
CREATE	TYPE	
CREATE	DATABASE	Vous ne pouvez pas créer de bases de données supplémentaires.

[1] Voir Index asynchrones dans Aurora DSQL pour créer un index sur une colonne d'une table spécifiée.

Considérations relatives à Aurora DSQL relatives à la compatibilité avec PostgreSQL

Tenez compte des limites de compatibilité suivantes lors de l'utilisation d'Aurora DSQL. Pour des considérations générales, voirConsidérations relatives à l'utilisation d'Amazon Aurora DSQL. Pour les quotas et les limites, voirQuotas de cluster et limites de base de données dans Amazon Aurora DSQL.

- Aurora DSQL utilise une seule base de données intégrée nomméepostgres. Vous ne pouvez pas créer de bases de données supplémentaires, renommer ou supprimer la `postgres` base de données.
- La `postgres` base de données utilise le codage de caractères UTF-8. Vous ne pouvez pas modifier le codage.
- La base de données utilise uniquement le `C` classement.
- Aurora DSQL utilise `UTC` comme fuseau horaire du système. Vous ne pouvez pas modifier le fuseau horaire à l'aide de paramètres ou d'instructions SQL tels que`SET TIMEZONE`.
- Le niveau d'isolation des transactions est fixé dans PostgreSQL`Repeatable Read`.
- Les transactions sont soumises aux contraintes suivantes :
  - Une transaction ne peut pas mélanger les opérations DDL et DML
  - Une transaction ne peut inclure qu'un seul relevé 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 ne vous permet pas d'exécuter `GRANT [permission] ON DATABASE` actuellement. Si vous tentez d'exécuter cette instruction, Aurora DSQL renvoie le message `ERROR: unsupported object type in GRANT` d'erreur.
- Aurora DSQL n'autorise pas les rôles d'utilisateur non administrateurs à exécuter la `CREATE SCHEMA` commande. Vous ne pouvez pas exécuter la `GRANT [permission] on DATABASE` commande et accorder `CREATE` des autorisations sur la base de données. Si un rôle d'utilisateur non administrateur tente de créer un schéma, Aurora DSQL renvoie le message d'erreur. `ERROR: permission denied for database postgres`
- Les utilisateurs non administrateurs ne peuvent pas créer d'objets dans le schéma public. Seuls les utilisateurs administrateurs peuvent créer des objets dans le schéma public. Le rôle d'utilisateur administrateur est autorisé à accorder l'accès en lecture, en écriture et en modification à ces objets à des utilisateurs non administrateurs, mais il ne peut pas accorder d'`CREATE`autorisations au schéma public lui-même. Les utilisateurs non administrateurs doivent utiliser des schémas différents créés par l'utilisateur pour créer des objets.
- Aurora DSQL ne prend pas en charge cette commande`ALTER ROLE [] CONNECTION LIMIT`. Contactez le AWS support si vous avez besoin d'une augmentation de la limite de connexion.
- Aurora DSQL ne prend pas en charge asyncpg, le pilote de base de données PostgreSQL asynchrone pour Python.

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. Il assure une conformité totale à l'ACID grâce à l'isolation 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 l'évolutivité sont essentielles.

Conflits de transactions

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, 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 implémenter une logique de nouvelle tentative afin de 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 contention élevés sur des touches uniques ou sur de petites plages de touches. 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 la contention sur les touches individuelles. Cette approche garantit des performances optimales même lorsque le volume des transactions augmente.

DDL et transactions distribuées dans Aurora DSQL

Le langage de définition de données (DDL) se comporte différemment dans Aurora DSQL par rapport à PostgreSQL. Aurora DSQL comporte une couche de base de données distribuée et partagée multi-AZ construite sur des flottes de calcul et de stockage mutualisées. Comme il n'existe aucun nœud ou leader de base de données principal, le catalogue de base de données est distribué. Ainsi, Aurora DSQL gère les modifications du schéma DDL sous forme de transactions distribuées.

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

Erreurs de contrôle de la 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 `INSERT` instructions `CREATE` et.

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

DDL asynchrone

Dans PostgreSQL standard, les opérations DDL `CREATE INDEX` telles que le verrouillage de la table affectée, 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, le DDL sur de grandes tables peut fonctionner sans interruption ni impact sur les performances. Pour plus d'informations sur le 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'`CLUSTER`opération dans PostgreSQL ou à un index clusterisé 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 principale 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 B-tree traditionnel. Contrairement à l'`CLUSTER`opération de PostgreSQL, qui ne réorganise les données qu'une seule fois, Aurora DSQL maintient cet ordre automatiquement et en continu. Cette approche améliore les performances des requêtes qui reposent sur un 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 sous-tend également la gestion distribuée des données. Il 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 le choix d'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 ou 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 attribue un identifiant masqué synthétique.
- 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 d'une clé primaire est de 1 kibioctet. Pour plus d'informations, voir 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, voir Limites de base de données dans Aurora DSQL et Types de données pris en charge dans Aurora DSQL.

Index asynchrones dans Aurora DSQL

La `CREATE INDEX ASYNC` commande 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 `sys.jobs` système. 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 le succès 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 s'afficher.

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 ]
```

Paramètres

`UNIQUE`

Indique à Aurora DSQL de vérifier la présence de 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 aucun nom, Aurora DSQL en génère automatiquement un 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 de colonne non clé dans une qualification de recherche par analyse d'index. Aurora DSQL ignore la colonne en termes d'unicité d'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 `CREATE INDEX ASYNC` commande 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 `sys.wait_for_job(job_id)` *`'your_index_creation_job_id'`* procédure 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 `sys.jobs` vue et supprime les tâches dont le statut est supérieur `completed` ou égal à 30 `failed` minutes. Ainsi, affiche `sys.jobs` 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 est conservé. `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 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 le tableau.

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

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

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

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

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

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

Aurora DSQL renvoie une réponse similaire à la suivante.

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

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

submitted	**processing**	**failed**	**completed**
La tâche est envoyée, mais Aurora DSQL n'a pas encore commencé à la traiter.	Aurora DSQL est en train de traiter la tâche.	La tâche a échoué. Consultez la colonne de détails pour plus d'informations. Si Aurora DSQL ne parvient pas à créer l'index, Aurora DSQL ne supprime pas	Aurora SQL

submitted	processing	failed	completed
		automatiquement la définition de l'index. Vous devez supprimer manuellement l'index à l'aide de la DROP INDEX commande.	

Vous pouvez également demander l'état de l'index via les tables du catalogue pg_index etpg_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 un statut initial deINVALID. L'indisvalidindicateur de l'index renvoie FALSE ouf, ce qui indique que l'index n'est pas valide. Si le drapeau revient TRUE out, 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 détailléFound 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.

Pour résoudre les échecs 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.

Détecter les 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 d'utilisateurs, le 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.

Tables système

Aurora DSQL étant compatible avec PostgreSQL, de nombreuses tables de catalogue système et 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.

Name (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 (prédéfinies)
`pg_class`	Décrit toutes les tables, colonnes, index et objets similaires
`pg_stats`	Vue des statistiques du planificateur
`pg_user`	Informations sur les utilisateurs

Name (Nom)	Description
`pg_roles`	Informations sur les utilisateurs et les groupes
`pg_indexes`	Liste 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 (utilisation`pg_roles`)
`pg_auth_members`	Oui
`pg_cast`	Oui
`pg_class`	Oui
`pg_collation`	Oui
`pg_constraint`	Oui
`pg_conversion`	Non

Nom	Applicable à Aurora DSQL
pg_database	Non
pg_db_role_setting	Oui
pg_default_acl	Oui
pg_depend	Oui
pg_description	Oui
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

Nom	Applicable à Aurora DSQL
pg_opfamily	Non
pg_parameter_acl	Oui
pg_partitioned_table	Non
pg_policy	Non
pg_proc	Non
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

Nom	Applicable à Aurora DSQL
pg_subscription_rel	Non
pg_tablespace	Non
pg_transform	Non
pg_trigger	Non
pg_ts_config	Oui
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 du 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

Nom	Applicable à Aurora DSQL
pg_file_settings	Non
pg_group	Oui
pg_hba_file_rules	Non
pg_ident_file_mappings	Non
pg_indexes	Oui
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

Nom	Applicable à Aurora DSQL
pg_stats	Oui
pg_stats_ext	Non
pg_stats_ext_exprs	Non
pg_tables	Oui
pg_timezone_abbrevs	Oui
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

Nom	Applicable à Aurora DSQL
pg_stat_bgwriter	Non
pg_stat_wal	Non
pg_stat_database	Non
pg_stat_database_conflicts	Non
pg_stat_all_tables	Non
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

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

Les vues sys.jobs et sys.iam_pg_role_mappings

Aurora DSQL prend en charge les vues système suivantes :

sys.jobs

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

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

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

sys.iam_pg_role_mappings

La vue `sys.iam_pg_role_mappings` fournit des informations sur les autorisations accordées aux utilisateurs IAM. Par exemple, s'il s'`DQSLDBConnect`agit d'un rôle IAM qui donne accès

à Aurora DSQL à des non-administrateurs et qu'un utilisateur nommé `testuser` reçoit le `DQSLDBConnect` rôle et les autorisations correspondantes, vous pouvez interroger la `sys.iam_pg_role_mappings` vue pour voir quels utilisateurs ont obtenu quelles autorisations.

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

La table pg_class

La `pg_class` table stocke les métadonnées relatives aux objets de base de données. Pour obtenir le nombre approximatif de lignes d'un tableau, exécutez la commande suivante.

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

La commande renvoie un résultat semblable à ce qui suit.

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

La **ANALYZE** commande

La `ANALYZE` commande collecte des statistiques sur le contenu des tables de la base de données et stocke les résultats dans la vue `pg_stats` système. 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 `ANALYZE` commande 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 s'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**
>
> ANALYZELes tâches sont facturées comme les autres tâches asynchrones dans Aurora DSQL. Lorsque vous modifiez un tableau, cela peut déclencher indirectement une tâche automatique de collecte de statistiques en arrière-plan, ce qui peut entraîner des frais de mesure en raison de l'activité associée au niveau du système.

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

Gestion des clusters SQL Aurora

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

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 au fur et à mesure de leur croissance et de leur évolution.

Configuration de clusters à région unique

Création d'un cluster

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

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

Example Command

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

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

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"
    }
}
```

Décrire un cluster

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

Example Command

```
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 update-cluster commande.

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

Example Command

```
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 delete-cluster commande.

> **Note**
>
> Vous ne pouvez supprimer que les clusters dont la protection contre la suppression est désactivée. Par défaut, la protection contre la suppression est activée lorsque vous créez de nouveaux clusters.

Example Command

```
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"
}
```

Liste des clusters

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

Example Command

```
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 multirégionaux

Ce chapitre explique comment configurer et gérer des clusters sur plusieurs Régions AWS.

Connexion à votre cluster multirégional

Les clusters homologues multirégionaux fournissent deux points de terminaison régionaux, un dans chaque cluster apparenté. 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 homologues, un cluster multirégional 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 multirégionales n'ont pas de points de terminaison.

Création de clusters multirégionaux

Pour créer des clusters multirégionaux, 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 l'est des États-Unis (Virginie du Nord) et dans l'est des États-Unis (Ohio) avec l'ouest des États-Unis (Oregon) comme région témoin.

Étape 1 : Création du cluster 1 dans l'est des États-Unis (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'`PENDING_SETUP`état. La création du cluster se poursuit `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 l'est des États-Unis (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'`PENDING_SETUP`état. La création du cluster reste inchangée jusqu'à `PENDING_SETUP` ce que vous le mettiez à jour avec l'ARN d'un autre cluster pour le peering.

Étape 3 : Cluster de pairs dans l'est des États-Unis (Virginie du Nord) avec l'est des États-Unis (Ohio)

Pour associer votre cluster USA Est (Virginie du Nord) à votre cluster USA Est (Ohio), utilisez la `update-cluster` commande. Spécifiez le nom de votre cluster USA East (Virginie du Nord) et une chaîne JSON avec l'ARN du cluster USA East (Ohio).

```
aws dsql update-cluster \
--region us-east-1 \
--identifier 'foo0bar1baz2quux3quuxquux4' \
--multi-region-properties '{"witnessRegion": "us-west-2","clusters": ["arn:aws:dsql:us-
east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5"]}'
```

Example 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 : Cluster de pairs dans l'est des États-Unis (Ohio) avec l'est des États-Unis (Virginie du Nord)

Pour associer votre cluster US East (Ohio) à votre cluster US East (Virginie du Nord), utilisez la `update-cluster` commande. Spécifiez le nom de votre cluster USA East (Ohio) et une chaîne JSON avec l'ARN du cluster US East (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 un peering réussi, les deux clusters passent du statut « PENDING_SETUP » à « CREATING » et enfin au statut « ACTIF » lorsqu'ils sont prêts à être utilisés.

Affichage des propriétés d'un cluster multirégional

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"
       ]
```

```
    }
}
```

Clusters de pairs lors de la création

Vous pouvez réduire le nombre d'étapes en incluant des informations de peering lors de la création du cluster. Après avoir créé votre premier cluster dans l'est des États-Unis (Virginie du Nord) (étape 1), vous pouvez créer votre deuxième cluster dans l'est des États-Unis (Ohio) tout en lançant le processus de peering 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 second cluster) pour établir la relation d'appariement. 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 multirégionaux

Pour supprimer un cluster multirégional, 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

Mettre à jour et supprimer un cluster dans l'est des États-Unis (Virginie du Nord)

1. Désactivez la protection contre la suppression à l'aide de la `update-cluster` commande.

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

2. Supprimez le cluster à l'aide de la `delete-cluster` commande.

```
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 au `PENDING_DELETE` statut. La suppression n'est pas complète tant que vous n'avez pas supprimé le cluster pair dans l'est des États-Unis (Ohio).

Mettre à jour et supprimer un cluster dans l'est des États-Unis (Ohio)

1. Désactivez la protection contre la suppression à l'aide de la `update-cluster` commande.

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

2. Supprimez le cluster à l'aide de la `delete-cluster` commande.

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

La 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 au `PENDING_DELETE` statut. Après quelques secondes, le système fait passer automatiquement les deux clusters homologues à l'`DELETING`état après validation.

Configuration de clusters multirégionaux à l'aide de AWS CloudFormation

Vous pouvez utiliser la même AWS CloudFormation ressource `AWS::DSQL::Cluster` pour déployer et gérer des clusters Aurora DSQL à région unique ou multirégionale.

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 cette `AWS::DSQL::Cluster` ressource.

Création de la configuration initiale du cluster

Créez d'abord un AWS CloudFormation modèle pour définir votre cluster multirégional :

```
---
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 AWS CLI 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
```

Trouver des identifiants de cluster

Récupérez la ressource physique IDs 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 du cluster

Mettez à jour votre AWS CloudFormation modèle pour inclure les deux clusters ARNs :

```
---
Resources:
  MRCluster:
    Type: AWS::DSQL::Cluster
    Properties:
      DeletionProtectionEnabled: true
      MultiRegionProperties:
        WitnessRegion: us-west-2
        Clusters:
        - arn:aws:dsql:us-east-2:123456789012:cluster/auabudrks5jwh4mjt6o5xxhr4y
        - arn:aws:dsql:us-east-1:123456789012:cluster/imabudrfon4p2z3nv2jo4rlajm
```

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
```

Programmation avec Aurora DSQL

Aurora DSQL met à votre disposition les outils suivants pour gérer vos ressources Aurora DSQL par programmation.

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 le manuel AWS CLI Command Reference.

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 SQL Aurora

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.

AWS CloudFormation

AWS::DSQL::ClusterIl s'agit d'une AWS CloudFormation ressource qui vous permet de créer et de gérer des clusters Aurora DSQL dans le cadre de votre infrastructure en tant que code. AWS 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 AWS CloudFormation modèles, vous pouvez provisionner de manière déclarative des clusters Aurora DSQL parallèlement à vos autres ressources cloud. Cela permet de garantir que votre infrastructure de données est déployée et gérée parallèlement au reste de votre pile d'applications.

Amazon Aurora DSQL SDKs, pilotes et exemple de code

AWS des kits de développement logiciel (SDKs) sont disponibles pour de nombreux langages de programmation courants. Chaque SDK fournit une API, des exemples de code et de la documentation qui facilitent la création d'applications par les développeurs dans leur langage préféré.

Adaptateurs et dialectes

Le tableau suivant indique les adaptateurs ORM et les dialectes de base de données disponibles pour Aurora DSQL.

Langage de programmation	Framework	Lien vers le référentiel
Java	Mise en veille prolongée	https://github.com/awslabs/aurora-dsql-hibernate/
Python	Django	https://github.com/awslabs/aurora-dsql-django/
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-sqlalchemy/

Exemples de code

Gestion de clusters à l'aide du AWS SDK

Le tableau suivant présente des exemples de code de gestion de cluster pour les différents langages de programmation utilisés AWS SDKs.

Langage de programmation	Exemple de lien vers le référentiel
C++	https://github.com/aws-samples/aurora-dsql-samples/tree/main/cpp/cluster_gestion
C# (.NET)	https://github.com/aws-samples/aurora-dsql-samples/tree/main/dotnet/cluster_gestion

Langage de programmation	Exemple de lien vers le référentiel
Go	https://github.com/aws-samples/aurora-dsql-samples/tree/main/go/cluster_gestion
Java	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/cluster_gestion
JavaScript	https://github.com/aws-samples/aurora-dsql-samples/tree/main/javascript/cluster_gestion
Python	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/cluster_gestion
Ruby	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/cluster_gestion
Rust	https://github.com/aws-samples/aurora-dsql-samples/tree/main/rust/cluster_gestion

Exemples de pilotes et de mappage relationnel objet () ORMs

Le tableau suivant présente des exemples de code de pilote de base de données et de framework ORM pour différents langages de programmation.

Langage de programmation	Pilote ou cadre	Exemple de lien vers le 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

Langage de programmation	Pilote ou cadre	Exemple de lien vers le référentiel
Go	pgx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/go/pgx
Java	Mise en veille prolongée	https://github.com/awslabs/aurora-dsql-hibernate/tree/main/examples/pet-clinic-app
Java	PGJDBC	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/pgjdbc
JavaScript	node-postgres	https://github.com/aws-samples/aurora-dsql-samples/tree/main/javascript/node-postgres
JavaScript	Postgres.js	https://github.com/aws-samples/aurora-dsql-samples/tree/main/javascript/postgres-js
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
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-sqlalchemy/tree/main/examples/pet-clinic-app

Langage de programmation	Pilote ou cadre	Exemple de lien vers le référentiel
Ruby	Rails	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/rails
Ruby	pg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/ruby-pg
Rust	SQLx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/rust/sqlx
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

Aurora DSQL avec le AWS CLI

Consultez les sections suivantes pour savoir comment gérer vos clusters avec le AWS CLI.

CreateCluster

Pour créer un cluster, utilisez la `create-cluster` commande.

> **Note**
>
> La création de clusters s'effectue de manière asynchrone. Appelez l'`GetCluster`API jusqu'à ce que le statut soit rétabli`ACTIVE`. Vous pouvez vous connecter à un cluster une fois qu'il le devient`ACTIVE`.

Exemple de commande

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

> **Note**
>
> Si vous souhaitez désactiver la protection contre la suppression lors de la création, incluez le `--no-deletion-protection-enabled` drapeau.

Exemple de réponse

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

GetCluster

Pour décrire un cluster, utilisez la `get-cluster` commande.

Exemple de commande

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

Exemple de réponse

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

```
    "status": "ACTIVE",
    "creationTime": "2025-05-22T14:03:26.631000-07:00",
    "deletionProtectionEnabled": true,
    "tags": {},
    "encryptionDetails": {
        "encryptionType": "AWS_OWNED_KMS_KEY",
        "encryptionStatus": "ENABLED"
    }
}
```

UpdateCluster

Pour mettre à jour un cluster existant, utilisez la `update-cluster` commande.

> **Note**
>
> Les mises à jour se font de manière asynchrone. Appelez l'`GetCluster`API jusqu'à ce que le statut soit `ACTIVE` rétabli et que vous observiez les modifications.

Exemple de commande

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

Exemple de réponse

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

DeleteCluster

Pour supprimer un cluster existant, utilisez la `delete-cluster` commande.

> **Note**
>
> Vous ne pouvez supprimer qu'un cluster dont la protection contre la suppression est désactivée. La protection contre la suppression est activée par défaut lors de la création de nouveaux clusters.

Exemple de commande

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

Exemple de réponse

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

ListClusters

Pour obtenir le a des clusters, utilisez la `list-clusters` commande.

Exemple de commande

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

Exemple de réponse

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

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

GetCluster sur les clusters multirégionaux

Pour obtenir des informations sur un cluster multirégional, utilisez la `get-cluster` commande. Pour les clusters multirégionaux, la réponse inclura le cluster ARNs lié.

Exemple de commande

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

Exemple de réponse

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

Création, lecture, mise à jour et suppression de clusters Aurora DSQL

Des exemples de création, lecture, mise à jour, suppression (CRUD) sont fournis pour les déploiements mono-régionaux et multirégionaux. Une `cluster_management` section dédiée à chaque langage de programmation est incluse et décrit ces tâches de gestion clés.

Les déploiements dans une seule région sont idéaux pour les applications destinées aux utilisateurs d'une zone géographique spécifique, car ils simplifient la gestion et réduisent le temps de latence. Les déploiements multirégionaux vous aident à améliorer la disponibilité et les capacités de reprise après sinistre en répartissant votre base de données entre plusieurs régions. Régions AWS

Choisissez le type de déploiement qui correspond aux exigences de votre application en matière de disponibilité, de performances et de distribution géographique.

Rubriques

Créer un cluster

Consultez les informations suivantes pour savoir comment créer des clusters à région unique ou multirégionale dans Aurora DSQL.

Python

Pour créer un cluster en un seul 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()
```

Pour créer un cluster multirégional, utilisez l'exemple suivant. La création d'un cluster multirégional peut prendre un certain 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={
```

C++

L'exemple suivant vous permet de créer un cluster en un seul 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;
}
```

Pour créer un cluster multirégional, utilisez l'exemple suivant. La création d'un cluster multirégional peut prendre un certain 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);

    // We can only set the witness region for the first cluster
    std::cout << "Creating cluster in " << region1 << std::endl;

    CreateClusterRequest createClusterRequest1;
    createClusterRequest1.SetDeletionProtectionEnabled(true);

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

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

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

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

    // For the second cluster we can set witness region and designate cluster1 as a
peer
    std::cout << "Creating cluster in " << region2 << std::endl;

    CreateClusterRequest createClusterRequest2;
    createClusterRequest2.SetDeletionProtectionEnabled(true);

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

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

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

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

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

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

    MultiRegionProperties updatedProps;
    updatedProps.SetWitnessRegion(witnessRegion);

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

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

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

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

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

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

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

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

JavaScript

Pour créer un cluster en un seul 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();
```

Pour créer un cluster multirégional, utilisez l'exemple suivant. La création d'un cluster multirégional peut prendre un certain 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 updateClusterCommand1 = new UpdateClusterCommand(
            {
                identifier: response1.identifier,
                multiRegionProperties: {
                    witnessRegion: witnessRegion,
                    clusters: [response2.arn]
                }
            }
        );

        await client1.send(updateClusterCommand1);
        console.log(`Added ${response2.arn} as a peer of ${response1.arn}`);

        // Now that multiRegionProperties is fully defined for both clusters
        // they'll begin the transition to ACTIVE
        console.log(`Waiting for cluster 1 ${response1.identifier} to become
ACTIVE`);

        await waitUntilClusterActive(
            {
                client: client1,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response1.identifier
            }
        );
        console.log(`Cluster 1 is now active`);

        console.log(`Waiting for cluster 2 ${response2.identifier} to become
ACTIVE`);
        await waitUntilClusterActive(
            {
                client: client2,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response2.identifier
            }
        );
        console.log(`Cluster 2 is now active`);
```

```
        console.log("The multi region clusters are now active");
        return;
    } catch (error) {
        console.error("Failed to create cluster: ", error.message);
        throw error;
    }
}

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

    await createMultiRegionCluster(region1, region2, witnessRegion);
}

main();
```

Java

Utilisez l'exemple suivant pour créer un cluster en un seul 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);
        }
    }
}
```

Pour créer un cluster multirégional, utilisez l'exemple suivant. La création d'un cluster multirégional peut prendre un certain 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

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

```
use aws_config::load_defaults;
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::operation::get_cluster::GetClusterOutput;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};
use std::collections::HashMap;

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

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

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

    Client::from_conf(config)
}

/// Create a cluster without delete protection and a name
pub async fn create_cluster(region: &'static str) -> GetClusterOutput {
    let client = dsql_client(region).await;
    let tags = HashMap::from([(
        String::from("Name"),
        String::from("rust single region cluster"),
    )]);

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

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

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

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

Pour créer un cluster multirégional, utilisez l'exemple suivant. La création d'un cluster multirégional peut prendre un certain 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

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

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

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

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

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

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

main if $PROGRAM_NAME == __FILE__
```

Pour créer un cluster multirégional, utilisez l'exemple suivant. La création d'un cluster multirégional peut prendre un certain 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__
```

.NET

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

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

namespace DSQLExamples.examples
{
    public class CreateSingleRegionCluster
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
```

```
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
region)
        {
            var awsCredentials = await
DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Create a cluster without delete protection and a name.
        /// </summary>
        public static async Task<CreateClusterResponse> Create(RegionEndpoint
region)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var tags = new Dictionary<string, string>
                {
                    { "Name", "csharp single region cluster" }
                };

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

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

                return response;
            }
        }

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

            await Create(region);
```

```
        }
    }
}
```

Pour créer un cluster multirégional, utilisez l'exemple suivant. La création d'un cluster multirégional peut prendre un certain 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}");
        }
    }
}
```

Golang

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

```
package main

import (
```

```
 "context"
 "fmt"
 "log"
 "time"

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

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

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

 // Create a DSQL client
 client := dsql.NewFromConfig(cfg)

 deleteProtect := true

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

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

 if err != nil {
  return fmt.Errorf("error creating cluster: %w", err)
 }

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

 // Create the waiter with our custom options
 waiter := dsql.NewClusterActiveWaiter(client, func(o
 *dsql.ClusterActiveWaiterOptions) {
  o.MaxDelay = 30 * time.Second
  o.MinDelay = 10 * time.Second
  o.LogWaitAttempts = true
 })
```

```
 id := clusterProperties.Identifier

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

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

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

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

 region := "us-east-1"

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

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

Pour créer un cluster multirégional, utilisez l'exemple suivant. La création d'un cluster multirégional peut prendre un certain 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)
 }

}
```

Obtenir un cluster

Consultez les informations suivantes pour savoir comment renvoyer des informations à un cluster dans Aurora DSQL.

Python

Pour obtenir des informations sur un cluster unique ou multirégional, 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 région unique ou multirégionale.

```
#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 unique ou multirégional, 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 région unique ou multirégionale.

```
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 région unique ou multirégionale.

```
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 région unique ou multirégionale.

```
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 région unique ou multirégionale.

```
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 région unique ou multirégionale.

```
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)
 }
}
```

Mettre à jour un cluster

Consultez les informations suivantes pour savoir comment mettre à jour un cluster dans Aurora DSQL. La mise à jour d'un cluster peut prendre une minute ou deux. Nous vous recommandons d'attendre un certain temps, puis d'exécuter get cluster pour obtenir l'état du cluster.

Python

Pour mettre à jour un cluster unique ou multirégional, 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 à une ou plusieurs 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 unique ou multirégional, 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 à une ou plusieurs 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 à une ou plusieurs 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 à une ou plusieurs 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 à une ou plusieurs 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 à une ou plusieurs 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)
 }
```

```
}
```

Supprimer un cluster

Consultez les informations suivantes pour savoir comment supprimer un cluster dans Aurora DSQL.

Python

Pour supprimer un cluster en un seul 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()
```

Pour supprimer un cluster multirégional, utilisez l'exemple suivant.

```
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 en un seul 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;
}
```

Pour supprimer un cluster multirégional, utilisez l'exemple suivant. La suppression d'un cluster multirégional peut prendre un certain 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 en un seul 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();
```

Pour supprimer un cluster multirégional, utilisez l'exemple suivant. La suppression d'un cluster multirégional peut prendre un certain 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 en un seul 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);
        }
    }
}
```

Pour supprimer un cluster multirégional, utilisez l'exemple suivant. La suppression d'un cluster multirégional peut prendre un certain 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 en un seul 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(())
}
```

Pour supprimer un cluster multirégional, utilisez l'exemple suivant. La suppression d'un cluster multirégional peut prendre un certain 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 en un seul Région AWS, utilisez l'exemple suivant.

```
equire "aws-sdk-dsql"

def delete_cluster(region, identifier)
  client = Aws::DSQL::Client.new(region: region)
```

```
  cluster = client.delete_cluster(identifier: identifier)
  puts "Initiated delete of #{cluster.arn}"

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

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

main if $PROGRAM_NAME == __FILE__
```

Pour supprimer un cluster multirégional, utilisez l'exemple suivant. La suppression d'un cluster multirégional peut prendre un certain 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 en un seul 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);
        }
    }
}
```

Pour supprimer un cluster multirégional, utilisez l'exemple suivant. La suppression d'un cluster multirégional peut prendre un certain 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 en un seul 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 supprimer un cluster multirégional, utilisez l'exemple suivant. La suppression d'un cluster multirégional peut prendre un certain 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)
 }
}
```

Didacticiels

Les didacticiels et les exemples de code suivants vous GitHub aideront à effectuer des tâches courantes dans Aurora DSQL.

- Utilisation de Benchbase avec Aurora DSQL : une branche de l'utilitaire d'analyse comparative open source Benchbase dont la compatibilité avec Aurora DSQL a été vérifiée.
- Chargeur Aurora DSQL : ce script Python open source vous permet de charger plus facilement des données dans Aurora DSQL pour vos cas d'utilisation, tels que le remplissage de tables à des fins de test ou le transfert de données dans Aurora DSQL.
- Exemples d'Aurora DSQL : le `aws-samples/aurora-dsql-samples` référentiel GitHub contient des exemples de code expliquant comment connecter et utiliser Aurora DSQL dans différents langages de programmation à l'aide des frameworks Web AWS SDKs, des mappeurs relationnels objets () ORMs et des frameworks Web. Les exemples montrent comment effectuer des tâches courantes, telles que l'installation de clients, la gestion de l'authentification et les opérations CRUD.

Utilisation AWS Lambda avec Amazon Aurora DSQL

Le didacticiel suivant explique comment utiliser Lambda avec Aurora DSQL

Prérequis

- Autorisation de créer des fonctions Lambda. Pour plus d'informations, consultez Getting started with Lambda.
- Autorisation de créer ou de modifier la politique IAM créée par Lambda. Vous avez besoin d'autorisations `iam:CreatePolicy` et`iam:AttachRolePolicy`. Pour plus d'informations, consultez la section Actions, ressources et clés de condition pour IAM.
- Vous devez avoir installé npm v8.5.3 ou supérieur.

- Vous devez avoir installé zip v3.0 ou supérieur.

Créez une nouvelle fonction dans AWS Lambda.

1. Connectez-vous à la AWS Lambda console AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/lambda/.
2. Choisissez Créer une fonction.
3. Entrez un nom, tel que `dsql-sample`.
4. Ne modifiez pas les paramètres par défaut pour vous assurer que Lambda crée un nouveau rôle avec des autorisations Lambda de base.
5. Choisissez Créer une fonction.

Autorisez votre rôle d'exécution Lambda à se connecter à votre cluster

1. Dans votre fonction Lambda, choisissez Configuration > Autorisations.
2. Choisissez le nom du rôle pour ouvrir le rôle d'exécution dans la console IAM.
3. Choisissez Ajouter des autorisations > Créer une politique intégrée, puis utilisez l'éditeur JSON.
4. Dans Action, collez l'action suivante pour autoriser votre identité IAM à vous connecter à l'aide du rôle de base de données d'administrateur.

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

> **Note**
>
> Nous utilisons un rôle d'administrateur afin de minimiser les étapes préalables au démarrage. Vous ne devez pas utiliser de rôle d'administrateur de base de données pour vos applications de production. Consultez Utilisation des rôles de base de données et de l'authentification IAM pour savoir comment créer des rôles de base de données personnalisés avec l'autorisation qui dispose du moins d'autorisations sur votre base de données.

5. Dans Resource, ajoutez le nom de ressource Amazon (ARN) de votre cluster. Vous pouvez également utiliser un joker.

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

6. Choisissez Suivant.
7. Entrez un nom pour la politique, par exemple`dsql-sample-dbconnect`.
8. Choisissez Create Policy (Créer une politique).

Créez un package à télécharger sur Lambda.

1. Créez un dossier nommé`myfunction`.
2. Dans le dossier, créez un nouveau fichier dont le nom `package.json` est le suivant.

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

3. Dans le dossier, créez un fichier nommé `index.mjs` dans le répertoire avec le contenu suivant.

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


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

```
    client = new Client({
      host: clusterEndpoint,
      user: "admin",
      password: token,
      database: "postgres",
      port: 5432,
      // <https://node-postgres.com/announcements> for version 8.0
      ssl: true,
      rejectUnauthorized: false
    });

    // Connect
    await client.connect();

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

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

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

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

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

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

  const response = {
    statusCode: responseCode,
    endpoint: endpoint,
  };
  return response;
};
```

4. Utilisez les commandes suivantes pour créer un package.

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

Téléchargez le package de code et testez votre fonction Lambda

1. Dans l'onglet Code de votre fonction Lambda, choisissez Upload from > .zip
2. Téléchargez le fichier `pkg.zip` que vous avez créé. Pour plus d'informations, voir Déployer les fonctions Lambda de Node.js avec des archives de fichiers .zip.
3. Dans l'onglet Test de votre fonction Lambda, collez la charge utile JSON suivante et modifiez-la pour utiliser votre ID de cluster.
4. Dans l'onglet Test de votre fonction Lambda, utilisez le JSON d'événement modifié suivant pour spécifier le point de terminaison de votre cluster.

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

5. Entrez un nom d'événement, tel que`dsql-sample-test`. Choisissez Enregistrer.
6. Sélectionnez Tester).
7. Choisissez Détails pour développer la réponse d'exécution et la sortie du journal.
8. En cas de succès, la réponse d'exécution de la fonction Lambda doit renvoyer un code d'état 200.

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

Si la base de données renvoie une erreur ou si la connexion à la base de données échoue, la réponse d'exécution de la fonction Lambda renvoie un code d'état 500.

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

Backup et restauration pour Amazon Aurora DSQL

Amazon Aurora DSQL vous aide à répondre à vos exigences de conformité réglementaire et de continuité d'activité grâce à l'intégration à un service de protection des données entièrement géré qui facilite la centralisation et l'automatisation des sauvegardes entre les AWS services, dans le cloud et sur site. AWS Backup Le service rationalise la création, la gestion et la restauration des sauvegardes pour les clusters Aurora DSQL à région unique ou multirégionale.

Les principales fonctionnalités sont décrites ci-après.

- Gestion centralisée des sauvegardes via AWS Management Console le SDK ou AWS CLI
- Sauvegardes complètes du cluster
- Programmes de sauvegarde automatisés et politiques de conservation
- Capacités interrégionales et multicomptes
- Configuration WORM (écriture unique, lecture multiple) pour toutes les sauvegardes que vous stockez

Pour plus d'informations sur les fonctionnalités de AWS Backup Vault Lock et une liste complète des AWS Backup fonctionnalités disponibles pour Aurora DSQL, consultez la section Avantages de Vault Lock et disponibilité des AWS Backup fonctionnalités dans le guide du AWS Backup développeur.

Commencer 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 Getting started with 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 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 créez toujours de nouveaux clusters pour préserver vos données sources. Pour restaurer un cluster Aurora DSQL à région unique, utilisez https://console.aws.amazon.com/backup ou CLI pour sélectionner le point de restauration (sauvegarde) que vous souhaitez restaurer. Configurez les paramètres du nouveau cluster qui sera créé à partir de votre sauvegarde.

La restauration d'un cluster multirégional Aurora DSQL n'est prise en charge que via le. AWS CLI Pour restaurer un cluster multirégional Aurora DSQL, vous devez utiliser à la fois la CLI Aurora DSQL AWS Backup et la CLI Aurora.

Pour restaurer un cluster multirégional Aurora DSQL.

1. Sélectionnez le point de récupération de votre cluster multirégional.
2. Copiez le point de restauration vers un autre point Région AWS qui prend en charge les clusters multirégionaux.

   > **Note**
   >
   > Les régions qui ne prennent pas en charge les clusters multirégionaux entraîneront l'échec de l'opération de restauration.

3. Lancez une tâche de restauration pour chaque cluster à l'aide de la AWS Backup CLI.
4. Utilisez la Configuration de clusters multirégionaux documentation pour comparer les clusters Aurora DSQL nouvellement créés.

Pour obtenir des instructions détaillées sur ces étapes, consultez la documentation relative à la restauration SQL d'Amazon Aurora.

Pour effectuer une restauration dans un cluster Aurora DSQL multirégional, vous pouvez utiliser une sauvegarde effectuée en une seule. Région AWS Toutefois, avant de lancer le processus de restauration, vous devez d'abord copier la sauvegarde vers un autre Région AWS qui prend en charge les clusters multirégionaux. Cette étape garantit la réussite de l'opération de restauration. Nous vous recommandons de créer des copies de sauvegarde dans des clés Régions AWS telles que l'est des États-Unis (Virginie du Nord), l'est des États-Unis (Ohio) ou l'ouest des États-Unis (Oregon) afin de bénéficier d'options de reprise après sinistre robustes et de répondre aux exigences de conformité.

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 SQL Aurora à l'aide de AWS CloudTrail pour en savoir plus sur la journalisation des enregistrements des actions entreprises par un utilisateur, un rôle ou Service AWS lors de l'utilisation d'Aurora DSQL.

Ressources supplémentaires

Pour en savoir plus sur les AWS Backup fonctionnalités et leur utilisation en tandem avec Aurora DSQL, consultez les ressources suivantes :

- Politiques gérées pour AWS Backup
- Restauration SQL d'Amazon Aurora
- 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 multirégional complexe, il 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 jouent un rôle important dans le maintien de la fiabilité, de la disponibilité et des performances de vos ressources SQL Amazon Aurora. Vous devez surveiller et collecter les données de journalisation de toutes les parties de vos ressources Aurora DSQL afin de pouvoir facilement corriger une défaillance multipoint.

- 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.

Affichage 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.

Statuts et définitions des clusters Aurora DSQL

Le tableau suivant décrit chaque état possible pour un cluster Aurora DSQL et sa signification.

État	Description
Création	Aurora DSQL tente de créer ou de configurer des ressources pour le cluster. Toute tentative de connexion échouera tant qu'un cluster est dans cet état.
Actif	Le cluster est opérationnel et prêt à être utilisé.

État	Description
Inactif	Un cluster devient inactif lorsqu'il reste inactif suffisamment longtemps pour qu'Aurora DSQL puisse récupérer les ressources configurées pour celui-ci. Lorsque vous vous connectez à un cluster inactif, Aurora DSQL fait revenir le cluster à l'état actif.
Inactif	Un cluster devient inactif lorsqu'il n'y a pas eu d'activité sur le cluster pendant une période prolongée. Lorsque vous tentez de vous connecter à un cluster inactif, Aurora DSQL fait automatiquement revenir le cluster à l'état actif.
Mise à jour en cours	Un cluster passe à l'état de mise à jour lorsque vous modifiez la configuration du cluster.
Suppression en cours	Un cluster passe au statut Suppression lorsque vous soumettez une demande de suppression.
Supprimé	Le cluster a été supprimé avec succès.
Échec	Aurora DSQL n'a pas pu créer le cluster car il a rencontré une erreur.
En attente de configuration	Pour les clusters multirégionaux uniquement. Un cluster multirégional passe au statut En attente de configuration lorsque vous créez un cluster multirégi onal 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 examiniez les deux clusters ensemble.
En attente de suppression	Pour les clusters multirégionaux uniquement. Un cluster multirégional passe au statut 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.

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 AWS Management Console :

Pour afficher l'état du cluster dans la console

1. Ouvrez la console Aurora DSQL à l'https://console.aws.amazon.com/dsqladresse.
2. Choisissez Clusters dans le volet de navigation.
3. Consultez le statut 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 suivante 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
```

Surveillance d'Aurora DSQL avec Amazon CloudWatch

Surveillez Aurora DSQL à l'aide CloudWatch de ce qui collecte les données brutes et les transforme en métriques lisibles en temps quasi réel. CloudWatch conserve ces statistiques pendant 15 mois, afin de vous aider à mieux comprendre les performances de votre application Web ou de votre service. Définissez des alarmes pour surveiller les seuils spécifiques et envoyer des notifications ou prendre des mesures lorsqu'ils sont atteints. Consultez les mesures d'utilisation et d'observabilité suivantes disponibles pour Aurora DSQL.

Pour plus d'informations, consultez le guide de CloudWatch l'utilisateur Amazon.

Observabilité et performance

Ce tableau présente les mesures d'observabilité pour Aurora DSQL. Il inclut des métriques pour le suivi des transactions en lecture seule et du total afin de fournir une caractérisation globale de la charge de travail. Des indicateurs exploitables tels 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 indicateurs liés aux sessions, à la fois actifs et totaux, donnent un aperçu de la charge actuelle du système.

CloudWatch Nom de la métrique	Mesure	Unit	Description
ReadOnlyTransactions	Transactions en lecture seule	none	Le nombre de transactions en lecture seule
TotalTransactions	Nombre total de transactions	none	Le nombre total de transactions exécutées sur le système, y compris les transactions en lecture seule.
QueryTimeouts	Délais d'expiration des requêtes	none	Le nombre de requêtes qui ont expiré en raison de l'atteinte de la durée maximale de transaction
OccConflicts	Conflits OCC	none	Le nombre de transactions abandonnées en raison d'un OCC au niveau clé

CloudWatch Nom de la métrique	Mesure	Unit	Description
CommitLatency	Latence de validation	millisecondes	Temps passé par la phase de validation de l'exécution de la requête (P50)
BytesWritten	Octets écrits	octets	Octets écrits dans le stockage
BytesRead	Octets lus	octets	Octets lus depuis le stockage
ComputeTime	Temps de calcul QP	millisecondes	Horloge murale QP Time
ClusterStorageSize	Taille de stockage en cluster	octets	Taille du cluster

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).

CloudWatch Nom de la métrique	Métrique	Dimension : ResourceId	Unit	Description
PU écrit	Unités d'écriture	<cluster-id>	DPU	Estime l'utilisation active en écriture du DPU de votre cluster Aurora DSQL.
MultiRegionWriteDPU	Unités d'écriture multirégionales	<cluster-id>	DPU	Applicable aux clusters multirégi

CloudWatch Nom de la métrique	Métrique	Dimension : ResourceId	Unit	Description
				onaux : indique approximativement le composant à utilisation active en écriture multirégionale de l'utilisation du DPU de votre cluster Aurora DSQL.
Lire le DPU	Unités de lecture	<cluster-id>	DPU	Indique approximativement l'utilisation active du composant en lecture du DPU de votre cluster Aurora DSQL.
PU informatique	Unités de calcul	<cluster-id>	DPU	Estimation de l'utilisation active du composant de calcul utilisé par le DPU de votre cluster Aurora DSQL.

CloudWatch Nom de la métrique	Métrique	Dimension : ResourceId	Unit	Description
DPU total	Nombre total d'unités	<cluster-id>	DPU	Estimation de l'utilisation totale du composant actif du DPU de votre cluster Aurora DSQL.

Journalisation des opérations SQL Aurora à l'aide de AWS CloudTrail

Amazon Aurora DSQL est intégré à AWS CloudTrailun service qui fournit un enregistrement des actions entreprises par un utilisateur, un rôle ou un Service AWS. Il existe deux types d'événements CloudTrail : les événements de gestion et les événements de données. Des événements de gestion sont émis pour auditer les modifications de configuration des AWS ressources. Les événements de données capturent l'utilisation AWS des ressources généralement dans le plan de données du service.

CloudTrail capture tous les appels d'API pour Aurora DSQL sous forme d'événements. Aurora DSQL enregistre l'activité de la console sous forme d'événements de gestion. Il capture également les tentatives de connexion authentifiées aux clusters sous forme d'événements de données.

À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande envoyée à Aurora DSQL, l'adresse IP à partir de laquelle la demande a été effectuée, la date à laquelle elle a été faite, l'identité de l'utilisateur à l'origine de la demande et des informations supplémentaires.

CloudTrail est activé par défaut dans votre compte Compte AWS lorsque vous créez le compte et que vous avez accès à l'historique des CloudTrail événements. L'historique des CloudTrail événements fournit un enregistrement consultable, consultable, téléchargeable et immuable des 90 derniers jours des événements de gestion enregistrés dans un. Région AWS Pour plus d'informations, consultez la section Utilisation de l'historique des CloudTrail événements dans le guide de AWS CloudTrail l'utilisateur. L'enregistrement de CloudTrail l'historique des événements est gratuit.

Pour créer un enregistrement continu des événements de votre AWS compte, 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 AWS CloudTrail événements). Pour plus d'informations sur la création de sentiers, consultez la section Utilisation des CloudTrail sentiers. Pour en savoir plus sur la configuration et la gestion des magasins de données d'événements, consultez la section Stockages de données d'événements CloudTrail Lake.

Événements de gestion Aurora DSQL dans CloudTrail

CloudTrail Les événements de gestion fournissent des informations sur les opérations de gestion effectuées sur les ressources de votre AWS compte. 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 enregistre toutes les opérations du plan de contrôle Aurora DSQL en tant qu'événements de gestion. Pour obtenir la liste des opérations du plan de contrôle Amazon Aurora DSQL auxquelles Aurora DSQL se connecte CloudTrail, consultez la référence de l'API Aurora DSQL.

Journaux du plan de contrôle

Amazon Aurora DSQL enregistre les opérations du plan de contrôle Aurora DSQL suivantes en CloudTrail tant qu'événements de gestion.

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

Logs de sauvegarde et de restauration

Amazon Aurora DSQL enregistre les opérations de sauvegarde et de restauration Aurora DSQL suivantes en CloudTrail tant qu'événements de gestion.

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

Pour en savoir plus sur la protection de vos clusters SQL Aurora à l'aide de AWS Backup, consultezBackup et restauration pour Amazon Aurora DSQL.

Journaux AWS KMS

Amazon Aurora DSQL enregistre les AWS KMS opérations suivantes en CloudTrail tant qu'événements de gestion.

- `GenerateDataKey`
- `Decrypt`

Pour en savoir plus sur la façon dont CloudTrail les journaux suivent les demandes qu'Aurora DSQL envoie AWS KMS en votre nom, consultezSurveillance de l'interaction SQL d'Aurora avec AWS KMS.

Événements de données Aurora DSQL dans CloudTrail

CloudTrail Les événements de données fournissent généralement des informations sur les opérations de ressources effectuées 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 n'enregistre pas les événements liés aux données. L'historique des CloudTrail événements n'enregistre pas les événements liés aux 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 plus d'informations sur la CloudTrail tarification, consultez la section AWS CloudTrail Tarification.

Pour Aurora DSQL, CloudTrail capture toute tentative de connexion effectuée à un cluster Aurora DSQL en tant qu'événement de données. Le tableau suivant répertorie les types de ressources Aurora DSQL pour lesquels vous pouvez enregistrer des événements de données. La colonne Type de ressource (console) indique la valeur à choisir dans la liste des types de ressources de la CloudTrail console. La colonne de valeur resources.type indique la **`resources.type`** valeur que vous devez spécifier lors de la configuration de sélecteurs d'événements avancés à l'aide du ou. AWS CLI CloudTrail APIs La CloudTrail colonne Données APIs enregistrées indique les appels d'API enregistrés CloudTrail pour le type de ressource.

Type de ressource (console)	valeur resources.type	Données APIs enregistrées sur CloudTrail
Amazon Aurora DSQL	`AWS::DSQL::Cluster`	• `DbConnect` • `DbConnectAdmin`

Vous pouvez configurer des sélecteurs d'événements avancés pour filtrer `eventName` et des `resources.ARN` champs pour enregistrer uniquement 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 la configuration AWS CLI pour `dsql-data-events-trail` 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 cela 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

- Résilience dans Amazon Aurora DSQL
- Sécurité de l'infrastructure dans Amazon Aurora DSQL
- Analyse de configuration et de vulnérabilité 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 vous associer `AmazonAuroraDSQLFullAccess` à vos utilisateurs, groupes et rôles.

Cette politique accorde des autorisations qui permettent un accès administratif 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 multirégionaux. Ils peuvent ajouter et supprimer des

balises dans les clusters. Ils peuvent répertorier les clusters et consulter les informations relatives à des clusters individuels. Ils peuvent voir les balises associées aux clusters Aurora DSQL. Ils peuvent se connecter à la base de données comme n'importe quel utilisateur, y compris en tant qu'administrateur. Ils peuvent effectuer des opérations de sauvegarde et de restauration pour les clusters Aurora DSQL, notamment démarrer, arrêter et surveiller les tâches de sauvegarde et de restauration. La politique inclut des AWS KMS autorisations qui autorisent les opérations sur les clés gérées par le client utilisées pour le chiffrement des clusters. Ils peuvent consulter tous les indicateurs CloudWatch de votre compte. Ils sont également autorisés à créer des rôles liés au service pour le `dsql.amazonaws.com` service, 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`: autorise la création d'un rôle lié à un service.
- `backup and restore`: accorde les autorisations nécessaires 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.

Vous trouverez la `AmazonAuroraDSQLFullAccess` politique sur la console IAM et AmazonAuroraDSQLFullAccess dans le AWS Managed Policy Reference Guide.

AWS politique gérée : AmazonAurora DSQLRead OnlyAccess

Vous pouvez vous 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. 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 pouvez trouver la `AmazonAuroraDSQLReadOnlyAccess` politique sur la console IAM et AmazonAuroraDSQLReadOnlyAccessdans le AWS Managed Policy Reference Guide.

AWS politique gérée : AmazonAurora DSQLConsole FullAccess

Vous pouvez vous associer `AmazonAuroraDSQLConsoleFullAccess` à vos utilisateurs, groupes et rôles.

Permet un accès administratif complet à Amazon Aurora DSQL via le AWS Management Console. Les principaux disposant de ces autorisations peuvent créer, supprimer et mettre à jour des clusters Aurora SQL, y compris des clusters multirégionaux, à l'aide de la console. Ils peuvent répertorier les clusters et consulter les informations relatives à des clusters individuels. Ils peuvent voir les tags de n'importe quelle ressource de votre compte. Ils peuvent se connecter à la base de données comme n'importe quel utilisateur, y compris l'administrateur. Ils peuvent effectuer des opérations de sauvegarde et de restauration pour les clusters Aurora DSQL, notamment démarrer, arrêter et surveiller les tâches de sauvegarde et de restauration. La politique inclut des AWS KMS autorisations qui autorisent les opérations sur les clés gérées par le client utilisées pour le chiffrement des clusters. Ils peuvent être lancés AWS CloudShell depuis le AWS Management Console. Ils peuvent consulter tous les indicateurs CloudWatch de votre compte. Ils sont également autorisés à créer des rôles liés au service pour le `dsql.amazonaws.com` service, 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 le. 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 les autorisations nécessaires 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 SQL Aurora.

Vous pouvez trouver la `AmazonAuroraDSQLReadOnlyAccess` politique sur la console IAM et AmazonAuroraDSQLReadOnlyAccessdans le AWS Managed Policy Reference Guide.

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 à un 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 d'historique du document Aurora DSQL.

Modification	Description	Date
AmazonAuroraDSQLFullMise à jour des accès	Permet d'effectuer des opérations de sauvegarde et de restauration pour les clusters SQL Aurora, notamment le démarrage, l'arrêt et la surveillance de tâches. Il 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	Ajoute la capacité d'effectuer des opérations de sauvegarde et de restauration pour les clusters SQL Aurora via le AWS Console Home. Cela inclut le démarrage, l'arrêt et le suivi des tâches. Il prend également en charge l'utilisation de clés KMS gérées par le client pour le chiffrement et le lancement du cluster. AWS CloudShell Pour plus d'informations, reportez-vous à la section	21 mai 2025

Modification	Description	Date
	Utilisation AmazonAuroraDSQLConsoleFullAccessde rôles liés à un service dans Aurora DSQL.	
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, voir Pour plus d'informations, voir AmazonAuroraDSQLFullAccès et utilisation des rôles liés à un service dans Aurora DSQL.	13 mai 2025

Modification	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

Modification	Description	Date
AmazonAuroraDSQLConsoleFullAccess update	Ajoute de nouvelles autorisations à Aurora DSQL pour prendre en charge la gestion de clusters multirégionaux et la connexion des points de terminaison 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

Modification	Description	Date
AuroraDsqlServiceLinkedRolePolicy update	Permet de publier des métriques `AWS/AuroraDSQL` et des `AWS/Usage CloudWatch` espaces de noms dans la politique. Cela permet au service ou au rôle associé de transmettre des données d'utilisation et de performance plus complètes à votre CloudWatch environnement. Pour plus d'informations, reportez-vous à la section Utilisation AuroraDsqlServiceLinkedRolePolicyde 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 de s'applique à la protection des données dans. Comme décrit dans ce modèle, est chargé de protéger l'infrastructure mondiale qui gère tous les AWS Cloud. 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 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 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

AWS Identity and Access Management. 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 la section Utilisation des sentiers dans le guide de l'utilisateur.
- Utilisez des solutions de chiffrement, ainsi que tous les contrôles de sécurité par défaut qu'ils contiennent 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 vivement 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 libres 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 fournit une infrastructure de stockage extrêmement durable conçue pour le stockage de données critiques et primaires. Les données sont stockées de manière redondante sur plusieurs appareils répartis dans plusieurs installations d'une 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 chiffrer les 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.

Pour le chiffrement au repos, voirChiffrement 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 AWS Direct Connect connexion. Pour plus d'informations, voir Qu'est-ce que c'est AWS Direct Connect ?

Vous pouvez accéder à Aurora DSQL via le réseau en utilisant des opérations d'API AWS publiées. 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.

Configuration des SSL/TLS certificats pour les connexions Aurora DSQL

Aurora DSQL exige que toutes les connexions utilisent le chiffrement TLS (Transport Layer Security). Pour établir des connexions sécurisées, votre système client doit faire confiance à l'autorité de certification racine Amazon (Amazon Root CA 1). 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 `verify-full` SSL 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 valider cela, vous pouvez suivre les étapes 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, le résultat suivant s'affiche :

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

macOS

1. Ouvrez Spotlight Search (Command + Space)
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 Connexion depuis Windows 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 `Amazon Root CA 1` certificat 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 de certificats 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. Copiez le certificat dans le 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. Ils fonctionnent Installation de certificats Linux 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 SQL Aurora

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 des 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. Connectez-vous 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 de l'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 du 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. Connectez-vous 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. Connectez-vous à la base de données :

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

Ressources supplémentaires

- Documentation SSL de PostgreSQL
- Amazon Trust Services

Chiffrement des données pour Amazon Aurora DSQL

Amazon Aurora DSQL chiffre toutes les données utilisateur 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 :

- Réduction de la charge opérationnelle liée à la protection des données sensibles
- Développez des applications sensibles à la sécurité qui répondent aux exigences réglementaires et de conformité strictes en matière de chiffrement
- Ajoutez une couche supplémentaire de protection des données en sécurisant toujours vos données dans un cluster crypté
- Respectez les politiques organisationnelles, les réglementations sectorielles ou gouvernementales et les exigences de conformité

Avec Aurora DSQL, vous pouvez créer des applications sensibles à la sécurité qui répondent aux exigences réglementaires et de conformité strictes en matière de chiffrement. 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

Aurora DSQL s'intègre AWS KMS pour gérer les clés de chiffrement de vos clusters. Pour en savoir plus sur les types et les états de clés, consultez AWS Key Management Service les concepts du Guide du AWS Key Management Service développeur. 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 pour vous. 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 ou 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, voir Clés gérées par le client et Clés détenues par AWSdans le Guide du AWS Key Management Service développeur.

> **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 de chiffrement avancée 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 la valeur par défaut Clés détenues par AWS ou choisir d'utiliser la vôtre 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 etSuppression 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 peuvent être utilisées gratuitement et sont renouvelées chaque année afin de protéger les ressources de votre compte. Vous n'avez pas besoin de consulter, de gérer, d'utiliser ou 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, détenez et gérez les clés gérées par les clients 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 la section Clés gérées par le client dans le Guide du AWS Key Management Service développeur.

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 multirégional, le statut de cet homologue devient `KMS_KEY_INACCESSIBLE` indisponible pour les opérations de lecture et d'écriture. Les autres pairs 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 la section Cluster idling.
> Pour les clusters multirégionaux, 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 plan. 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 de AWS KMS key s avec Aurora DSQL
- Utilisation de clés de cluster avec Aurora DSQL
- Mise en cache des clés du cluster

Utilisation de AWS KMS key s avec Aurora DSQL

Le chiffrement au repos protège votre cluster SQL Aurora sous un 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 clé KMS différente pour chaque cluster, même s'il participe à une configuration multirégionale.

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 utilisant l'`UpdateCluster`opération. Le processus de changement de clé ne nécessite pas de temps d'arrêt ni de dégradation du service.

> **Important**
>
> Aurora DSQL ne prend en charge que les clés KMS symétriques. Vous ne pouvez pas utiliser de clé KMS asymétrique pour chiffrer vos clusters Aurora DSQL.

Une clé gérée par le client offre les avantages suivants.

- Vous créez et gérez la clé KMS, notamment en définissant les politiques clés 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 AWS KMS des quotas s'appliquent à ces clés.

Utilisation de 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 comme clé de chiffrement. 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 de données unique pour chaque structure sous-jacente d'un cluster, mais plusieurs éléments du cluster peuvent être protégés par la même clé de chiffrement de 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. Il stocke ensuite les clés chiffrées avec les données chiffrées afin qu'elles soient disponibles pour déchiffrer les données du 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 du 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 AWS Identity and Access Management (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 relatives à cette clé doivent autoriser Aurora DSQL à l'utiliser en votre nom.

Vous avez un contrôle total sur les politiques relatives à 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 politique clé ou une politique IAM.

Aurora DSQL nécessite 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 politique 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 politique ne sont pas autorisés à utiliser Aurora DSQL, l'appel échoue, même s'il provient du service Aurora DSQL.
- La clé de `kms:ViaService` condition autorise les autorisations uniquement lorsque la demande provient d'Aurora DSQL pour le compte des principaux mentionnés dans la déclaration de politique. Ces principals ne peuvent pas appeler ces opérations directement.
- Donne aux AWS KMS key administrateurs (utilisateurs pouvant assumer le `db-team` rôle) un accès en lecture seule au AWS KMS key

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 SQL Aurora

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. Il apparaît également en texte clair dans des journaux tels que ceux contenus dans 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 qu'Aurora DSQL est en train de chiffrer. 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 un cluster Aurora DSQL chiffré, Aurora DSQL doit déchiffrer la clé du cluster afin de pouvoir déchiffrer les clés situées en 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 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 (avancés) pour spécifier 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 de définir la politique de clé KMS de manière appropriée. Pour des exemples et de plus amples informations, consultezthe 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 appartenant à AWS. Le cluster est désormais chiffré avec la clé appartenant à 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 une 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 par défaut Clé détenue par AWS. Procédez comme suit pour mettre à jour les clés de chiffrement d'un cluster existant à partir de la console Aurora DSQL ou du AWS CLI.

Console

Pour mettre à jour ou supprimer une clé de chiffrement 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. Dans la vue de liste, recherchez et sélectionnez la ligne du cluster que vous souhaitez mettre à jour.
4. Sélectionnez le menu Actions, puis choisissez Modifier.
5. Dans les paramètres de chiffrement du cluster, choisissez l'une des options suivantes pour modifier vos 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 clé KMS, sélectionnez l'option Personnaliser les paramètres de chiffrement (avancés) si elle n'est pas déjà sélectionnée. Ensuite, recherchez et sélectionnez 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"
```

La description `EncryptionStatus` du cluster est définie sur `ENABLED` et `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.

Pour mettre à jour 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
```

La description `EncryptionStatus` du cluster passe à `UPDATING` et `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 sera transféré 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 de définir la politique de clé KMS de manière appropriée. Pour des exemples et de plus amples informations, consultezthe 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 du cluster au repos. Vous ne pouvez pas désactiver ce chiffrement ou 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 le cluster l'`encryptionStatus`est `ENABLED` et que vous voyez `kmsKeyArn` 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 en `IDLE` état.
- 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 SQL Amazon Aurora est utilisée `CUSTOMER_MANAGED_KMS_KEY` pour les clés gérées par le client et `AWS_OWNED_KMS_KEY` pour 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

AWS Identity and Access Management (IAM) est un outil Service AWS qui permet à un administrateur de contrôler en toute sécurité l'accès aux AWS ressources. Les administrateurs IAM contrôlent qui peut être authentifié (connecté) et autorisé (autorisé) à utiliser les ressources Aurora DSQL. IAM est un Service AWS outil que vous pouvez utiliser sans frais supplémentaires.

Rubriques

Public ciblé

La façon dont vous utilisez AWS Identity and Access Management (IAM) varie en fonction du travail que vous effectuez dans Aurora DSQL.

Utilisateur du service : si vous utilisez le service Aurora DSQL pour effectuer votre travail, votre administrateur vous fournit les informations d'identification et les autorisations dont vous avez besoin. Au fur et à mesure que vous utilisez de plus en plus de fonctionnalités d'Aurora DSQL pour effectuer votre travail, vous aurez peut-être besoin d'autorisations supplémentaires. En comprenant bien la gestion des accès, vous saurez demander les autorisations appropriées à votre administrateur. Si vous ne pouvez pas accéder à une fonctionnalité dans Aurora DSQL, consultezRésolution des problèmes d'identité et d'accès Amazon Aurora DSQL.

Administrateur de service — Si vous êtes responsable des ressources Aurora DSQL dans votre entreprise, vous avez probablement un accès complet à Aurora DSQL. C'est à vous de déterminer les fonctionnalités et les ressources d'Aurora DSQL auxquelles les utilisateurs de votre service doivent accéder. Vous devez ensuite soumettre les demandes à votre administrateur IAM pour modifier les autorisations des utilisateurs de votre service. Consultez les informations sur cette page pour comprendre les concepts de base d'IAM. Pour en savoir plus sur la manière dont votre entreprise peut utiliser IAM avec Aurora DSQL, consultez. Comment Amazon Aurora DSQL fonctionne avec IAM

Administrateur IAM : si vous êtes administrateur IAM, vous souhaiterez peut-être en savoir plus sur la manière dont vous pouvez rédiger des politiques pour gérer l'accès à Aurora DSQL. Pour consulter des exemples de politiques basées sur l'identité Aurora DSQL que vous pouvez utiliser dans IAM, consultez. 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 (connecté à AWS) en tant qu'utilisateur IAM ou en assumant un rôle IAM. Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en AWS tant qu'identité fédérée en utilisant les informations d'identification fournies par le biais d'une source d'identité. AWS IAM Identity Center Les utilisateurs (IAM Identity Center), l'authentification unique de votre entreprise et vos informations d'identification Google ou Facebook sont des exemples d'identités fédérées. Lorsque vous vous connectez avec une identité fédérée, votre administrateur aura précédemment configuré une fédération d'identités avec

des rôles IAM. Lorsque vous accédez à AWS l'aide de la fédération, vous assumez indirectement un rôle.

Selon le type d'utilisateur que vous êtes, vous pouvez vous connecter au portail AWS Management Console ou au portail AWS d'accès. Pour plus d'informations sur la connexion à AWS, consultez la section Comment vous connecter à votre compte Compte AWS dans le guide de Connexion à AWS l'utilisateur.

Si vous y accédez AWS par programmation, AWS fournit un kit de développement logiciel (SDK) et une interface de ligne de commande (CLI) pour signer cryptographiquement vos demandes à l'aide de vos informations d'identification. Si vous n'utilisez pas d' AWS outils, vous devez signer vous-même les demandes. Pour plus d'informations sur l'utilisation de la méthode recommandée pour signer des demandes vous-même, consultez AWS Signature Version 4 pour les demandes d'API dans le Guide de l'utilisateur IAM.

Quelle que soit la méthode d'authentification que vous utilisez, vous devrez peut-être fournir des informations de sécurité supplémentaires. Par exemple, il vous AWS recommande d'utiliser l'authentification multifactorielle (MFA) pour renforcer la sécurité de votre compte. Pour plus d'informations, consultez Authentification multifactorielle dans le Guide de l'utilisateur AWS IAM Identity Center et Authentification multifactorielle AWS dans IAM dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une identité de connexion unique qui donne un accès complet à toutes Services AWS les ressources du compte. Cette identité est appelée utilisateur Compte AWS root et est accessible en vous connectant avec l'adresse e-mail et le mot de passe que vous avez utilisés pour créer le compte. Il est vivement recommandé de ne pas utiliser l'utilisateur racine pour vos tâches quotidiennes. Protégez vos informations d'identification d'utilisateur racine et utilisez-les pour effectuer les tâches que seul l'utilisateur racine peut effectuer. Pour obtenir la liste complète des tâches qui vous imposent de vous connecter en tant qu'utilisateur racine, consultez Tâches nécessitant des informations d'identification d'utilisateur racine dans le Guide de l'utilisateur IAM.

Identité fédérée

La meilleure pratique consiste à obliger les utilisateurs humains, y compris ceux qui ont besoin d'un accès administrateur, à utiliser la fédération avec un fournisseur d'identité pour accéder à l'aide Services AWS d'informations d'identification temporaires.

Une identité fédérée est un utilisateur de l'annuaire des utilisateurs de votre entreprise, d'un fournisseur d'identité Web AWS Directory Service, du répertoire Identity Center ou de tout utilisateur qui y accède à l'aide des informations d'identification fournies Services AWS par le biais d'une source d'identité. Lorsque des identités fédérées y accèdent Comptes AWS, elles 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. Vous pouvez créer des utilisateurs et des groupes dans IAM Identity Center, ou vous pouvez vous connecter et synchroniser avec un ensemble d'utilisateurs et de groupes dans votre propre source d'identité afin de les utiliser dans toutes vos applications Comptes AWS et applications. Pour obtenir des informations sur IAM Identity Center, 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 au sein de vous Compte AWS qui possède des autorisations spécifiques pour une seule personne ou application. Dans la mesure du possible, nous vous recommandons de vous appuyer sur des informations d'identification temporaires plutôt que de créer des utilisateurs IAM ayant des informations d'identification à long terme telles que des mots de passe et des clés d'accès. Toutefois, si certains cas d'utilisation spécifiques nécessitent des informations d'identification à long terme avec les utilisateurs IAM, nous vous recommandons d'effectuer une rotation des clés d'accès. Pour plus d'informations, consultez Rotation régulière des clés d'accès pour les cas d'utilisation nécessitant des informations d'identification dans le Guide de l'utilisateur IAM.

Un groupe IAM est une identité qui concerne un ensemble d'utilisateurs IAM. Vous ne pouvez pas vous connecter en tant que groupe. Vous pouvez utiliser les groupes pour spécifier des autorisations pour plusieurs utilisateurs à la fois. Les groupes permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Par exemple, vous pouvez nommer un groupe IAMAdminset lui donner les autorisations nécessaires pour administrer les ressources IAM.

Les utilisateurs sont différents des rôles. Un utilisateur est associé de manière unique à une personne ou une application, alors qu'un rôle est conçu pour être endossé par tout utilisateur qui en a besoin. Les utilisateurs disposent d'informations d'identification permanentes, mais les rôles fournissent des informations d'identification temporaires. 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 au sein de vous Compte AWS dotée d'autorisations spécifiques. Le concept ressemble à celui d'utilisateur IAM, mais le rôle IAM n'est pas associé à une personne en particulier. Pour assumer temporairement un rôle IAM dans le AWS Management Console, vous pouvez passer d'un rôle d'utilisateur à un rôle IAM (console). Vous pouvez assumer un rôle en appelant une opération d' AWS API AWS CLI ou en utilisant une URL personnalisée. Pour plus d'informations sur les méthodes d'utilisation des rôles, consultez Méthodes pour endosser un rôle dans le Guide de l'utilisateur IAM.

Les rôles IAM avec des informations d'identification temporaires sont utiles dans les cas suivants :

- Accès utilisateur fédéré : pour attribuer des autorisations à une identité fédérée, vous créez un rôle et définissez des autorisations pour le rôle. Quand une identité externe s'authentifie, l'identité est associée au rôle et reçoit les autorisations qui sont définies par celui-ci. Pour obtenir des informations sur les rôles pour la fédération, consultez Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM. Si vous utilisez IAM Identity Center, vous configurez un jeu d'autorisations. IAM Identity Center met en corrélation le jeu d'autorisations avec un rôle dans IAM afin de contrôler à quoi vos identités peuvent accéder après leur authentification. Pour plus d'informations sur les jeux d'autorisations, consultez Jeux d'autorisations dans le Guide de l'utilisateur AWS IAM Identity Center .
- Autorisations d'utilisateur IAM temporaires : un rôle ou un utilisateur IAM peut endosser un rôle IAM pour profiter temporairement d'autorisations différentes pour une tâche spécifique.
- Accès intercompte : vous pouvez utiliser un rôle IAM pour permettre à un utilisateur (principal de confiance) d'un compte différent d'accéder aux ressources de votre compte. Les rôles constituent le principal moyen d'accorder l'accès intercompte. Toutefois, dans certains Services AWS cas, vous pouvez associer une politique directement à une ressource (au lieu d'utiliser un rôle comme proxy). Pour en savoir plus sur la différence entre les rôles et les politiques basées sur les ressources pour l'accès intercompte, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.
- Accès multiservices — Certains Services AWS utilisent des fonctionnalités dans d'autres Services AWS. Par exemple, lorsque vous effectuez un appel dans un service, il est courant que ce service exécute des applications dans Amazon EC2 ou stocke des objets dans Amazon S3. Un service peut le faire en utilisant les autorisations d'appel du principal, un rôle de service ou un rôle lié au service.
  - Sessions d'accès direct (FAS) : lorsque vous utilisez un utilisateur ou un rôle IAM pour effectuer des actions AWS, vous êtes considéré comme un mandant. Lorsque vous utilisez certains

services, vous pouvez effectuer une action qui initie une autre action dans un autre service. FAS utilise les autorisations du principal appelant et Service AWS, associées Service AWS à la demande, pour adresser des demandes aux services en aval. Les demandes FAS ne sont effectuées que lorsqu'un service reçoit une demande qui nécessite des interactions avec d'autres personnes Services AWS ou des ressources pour être traitée. Dans ce cas, vous devez disposer d'autorisations nécessaires pour effectuer les deux actions. Pour plus de détails sur une politique lors de la formulation de demandes FAS, consultez Transmission des sessions d'accès.

- Rôle de service : il s'agit d'un rôle IAM attribué à un service afin de réaliser 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.
- Rôle lié à un service — 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.

- Applications exécutées sur Amazon EC2 : vous pouvez utiliser un rôle IAM pour gérer les informations d'identification temporaires pour les applications qui s'exécutent sur une EC2 instance et qui envoient des demandes AWS CLI d' AWS API. Cela est préférable au stockage des clés d'accès dans l' EC2 instance. Pour attribuer un AWS rôle à une EC2 instance et le rendre disponible pour toutes ses applications, vous devez créer un profil d'instance attaché à l'instance. Un profil d'instance contient le rôle et permet aux programmes exécutés sur l' EC2 instance d'obtenir des informations d'identification temporaires. Pour plus d'informations, consultez Utiliser un rôle IAM pour accorder des autorisations aux applications exécutées sur des EC2 instances Amazon dans le guide de l'utilisateur IAM.

Gestion des 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 est un objet AWS qui, lorsqu'il est associé à une identité ou à une ressource, définit leurs autorisations. AWS évalue ces politiques lorsqu'un principal (utilisateur, utilisateur root ou session de rôle) fait une demande. Les autorisations dans les politiques déterminent si la demande est autorisée ou refusée. La plupart des politiques sont stockées AWS sous forme de documents JSON. Pour plus d'informations sur la structure et le contenu des documents de politique JSON, consultez Vue d'ensemble des politiques JSON dans le Guide de l'utilisateur IAM.

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.

Par défaut, les utilisateurs et les rôles ne disposent d'aucune autorisation. 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. L'administrateur peut ensuite ajouter les politiques IAM aux rôles et les utilisateurs peuvent assumer les rôles.

Les politiques IAM définissent les autorisations d'une action, quelle que soit la méthode que vous utilisez pour exécuter l'opération. Par exemple, supposons que vous disposiez d'une politique qui autorise l'action `iam:GetRole`. Un utilisateur appliquant cette politique peut obtenir des informations sur le rôle à partir de AWS Management Console AWS CLI, de ou de l' AWS API.

Politiques basées sur l'identité

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.

Les politiques basées sur l'identité peuvent être classées comme des politiques en ligne ou des politiques gérées. Les politiques en ligne sont intégrées directement à un utilisateur, groupe ou rôle. Les politiques gérées sont des politiques autonomes que vous pouvez associer à plusieurs utilisateurs, groupes et rôles au sein de votre Compte AWS. Les politiques gérées incluent les politiques AWS gérées et les politiques gérées par le client. Pour découvrir comment choisir entre une politique gérée et une politique 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. 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 principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou. Services AWS

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.

Listes de contrôle d'accès (ACLs)

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.

Amazon S3 et AWS WAF Amazon VPC sont des exemples de services compatibles. ACLs Pour en savoir plus ACLs, consultez la présentation de la liste de contrôle d'accès (ACL) dans le guide du développeur Amazon Simple Storage Service.

Autres types de politique

AWS prend en charge d'autres types de politiques moins courants. Ces types de politiques peuvent définir le nombre maximum d'autorisations qui vous sont accordées par des types de politiques plus courants.

- Limite d'autorisations : une limite d'autorisations est une fonctionnalité avancée dans laquelle vous définissez le nombre maximal d'autorisations qu'une politique basée sur l'identité peut accorder à une entité IAM (utilisateur ou rôle IAM). Vous pouvez définir une limite d'autorisations pour une entité. Les autorisations en résultant représentent la combinaison des politiques basées sur l'identité d'une entité et de ses limites d'autorisation. Les politiques basées sur les ressources qui spécifient l'utilisateur ou le rôle dans le champ `Principal` ne sont pas limitées par les limites d'autorisations. Un refus explicite dans l'une de ces politiques annule l'autorisation. Pour plus d'informations sur les limites d'autorisations, consultez Limites d'autorisations pour des entités IAM dans le Guide de l'utilisateur IAM.
- Politiques de contrôle des services (SCPs) : SCPs politiques JSON qui spécifient les autorisations maximales pour une organisation ou une unité organisationnelle (UO) dans AWS Organizations. AWS Organizations est un service permettant de regrouper et de gérer de manière centralisée Comptes AWS les multiples propriétés de votre entreprise. Si vous activez toutes les fonctionnalités d'une organisation, vous pouvez appliquer des politiques de contrôle des services (SCPs) à l'un ou à l'ensemble de vos comptes. Le SCP limite les autorisations pour les entités

figurant dans les comptes des membres, y compris chacune Utilisateur racine d'un compte AWS d'entre elles. Pour plus d'informations sur les Organizations SCPs, voir Politiques de contrôle des services dans le Guide de AWS Organizations l'utilisateur.

- Politiques de contrôle des ressources (RCPs) : RCPs politiques JSON que vous pouvez utiliser pour définir le maximum d'autorisations disponibles pour les ressources de vos comptes sans mettre à jour les politiques IAM associées à chaque ressource que vous possédez. Le RCP limite les autorisations pour les ressources des comptes membres et peut avoir un impact sur les autorisations effectives pour les identités, y compris Utilisateur racine d'un compte AWS, qu'elles appartiennent ou non à votre organisation. Pour plus d'informations sur les Organizations RCPs, y compris une liste de ces Services AWS supports RCPs, consultez la section Resource control policies (RCPs) dans le guide de AWS Organizations l'utilisateur.
- Politiques de séance : les politiques de séance sont des politiques avancées que vous utilisez en tant que paramètre lorsque vous créez par programmation une séance temporaire pour un rôle ou un utilisateur fédéré. Les autorisations de séance en résultant sont une combinaison des politiques basées sur l'identité de l'utilisateur ou du rôle et des politiques de séance. Les autorisations peuvent également provenir d'une politique basée sur les ressources. Un refus explicite dans l'une de ces politiques annule l'autorisation. 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.

Comment Amazon Aurora DSQL fonctionne avec IAM

Avant d'utiliser IAM pour gérer l'accès à Aurora DSQL, découvrez quelles fonctionnalités IAM peuvent être utilisées avec Aurora DSQL.

Fonctionnalités IAM que vous pouvez utiliser avec Amazon Aurora DSQL

Fonctionnalité IAM	Prise en charge d'Aurora DSQL
Politiques basées sur l'identité	Oui

Fonctionnalité IAM	Prise en charge d'Aurora DSQL
Politiques basées sur les ressources	Non
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
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. Vous ne pouvez pas spécifier le principal dans une politique basée sur une identité,

car celle-ci s'applique à l'utilisateur ou au rôle auquel elle est attachée. 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 consulter des exemples de politiques basées sur l'identité Aurora DSQL, consultez. Exemples de politiques basées sur l'identité pour Amazon Aurora DSQL

Politiques basées sur les ressources dans Aurora DSQL

Prend en charge les politiques basées sur les ressources : non

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 principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou. Services AWS

Pour permettre un accès intercompte, vous pouvez spécifier un compte entier ou des entités IAM dans un autre compte en tant que principal dans une politique basée sur les ressources. L'ajout d'un principal intercompte à une politique basée sur les ressources ne représente qu'une partie de l'instauration de la relation d'approbation. Lorsque le principal et la ressource sont différents Comptes AWS, un administrateur IAM du compte sécurisé doit également accorder à l'entité principale (utilisateur ou rôle) l'autorisation d'accéder à la ressource. Pour ce faire, il attache une politique basée sur une identité à l'entité. Toutefois, si une politique basée sur des ressources accorde l'accès à un principal dans le même compte, aucune autre politique basée sur l'identité n'est requise. Pour plus d'informations, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Actions de stratégie 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. Les actions de stratégie portent généralement le même nom que l'opération AWS d'API associée. Il existe quelques exceptions, telles que les actions avec autorisations uniquement qui n'ont pas d'opération API correspondante. Certaines opérations nécessitent également plusieurs actions dans une politique. Ces actions supplémentaires sont nommées actions dépendantes.

Intégration d'actions dans une politique afin d'accorder l'autorisation d'exécuter les opérations associées.

Pour consulter la liste des actions Aurora DSQL, consultez la section Actions définies par Amazon Aurora DSQL dans le Service Authorization Reference.

Les actions de stratégie 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 consulter des exemples de politiques basées sur l'identité Aurora DSQL, consultez. Exemples de politiques basées sur l'identité pour Amazon Aurora DSQL

Ressources relatives aux politiques 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. Les instructions doivent inclure un élément `Resource` ou `NotResource`. Il est recommandé de définir une ressource à l'aide de son Amazon Resource Name (ARN). Vous pouvez le faire pour des actions qui prennent en charge un type de ressource spécifique, connu sous la dénomination autorisations de niveau ressource.

Pour les actions qui ne sont pas compatibles avec les autorisations de niveau ressource, telles que les opérations de liste, 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 savoir avec quelles actions vous pouvez spécifier l'ARN de chaque ressource, consultez Actions définies par Amazon Aurora DSQL.

Pour consulter des exemples de politiques basées sur l'identité Aurora DSQL, consultez. Exemples de politiques basées sur l'identité pour Amazon Aurora DSQL

Clés de conditions de 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` (ou le bloc `Condition`) vous permet de spécifier des conditions lorsqu'une instruction est appliquée. L'élément `Condition` est facultatif. 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.

Si vous spécifiez plusieurs éléments `Condition` dans une instruction, ou plusieurs clés dans un seul élément `Condition`, AWS les évalue à l'aide d'une opération `AND` logique. Si vous spécifiez plusieurs valeurs pour une seule clé de condition, AWS évalue la condition à l'aide d'une `OR` opération logique. Toutes les conditions doivent être remplies avant que les autorisations associées à l'instruction ne soient accordées.

Vous pouvez aussi utiliser des variables d'espace réservé quand vous spécifiez des conditions. Par exemple, vous pouvez accorder à un utilisateur IAM l'autorisation d'accéder à une ressource uniquement si elle est balisée avec son nom d'utilisateur IAM. Pour plus d'informations, consultez Éléments d'une politique IAM : variables et identifications dans le Guide de l'utilisateur IAM.

AWS prend en charge les clés de condition globales et les clés de condition spécifiques au service. 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 consulter la liste des clés de condition Aurora DSQL, consultez la section Clés de condition pour Amazon Aurora DSQL dans le manuel Service Authorization Reference. Pour savoir avec quelles actions et ressources vous pouvez utiliser une clé de condition, consultez Actions définies par Amazon Aurora DSQL.

Pour consulter des exemples de politiques basées sur l'identité Aurora DSQL, 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 des autorisations en fonction des attributs. Dans AWS, ces attributs sont appelés balises. Vous pouvez associer des balises aux entités IAM (utilisateurs ou rôles) et à de nombreuses AWS ressources. L'étiquetage des entités et des ressources est la première étape d'ABAC. Vous concevez ensuite des politiques ABAC pour autoriser des opérations quand l'identification du principal correspond à celle de la ressource à laquelle il tente d'accéder.

L'ABAC est utile dans les environnements qui connaissent une croissance rapide et pour les cas où la gestion des politiques devient fastidieuse.

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 d'informations d'identification temporaires avec Aurora DSQL

Prend en charge les informations d'identification temporaires : oui

Certains Services AWS ne fonctionnent pas lorsque vous vous connectez à l'aide d'informations d'identification temporaires. Pour plus d'informations, y compris celles qui Services AWS fonctionnent avec des informations d'identification temporaires, consultez Services AWS la section relative à l'utilisation d'IAM dans le guide de l'utilisateur d'IAM.

Vous utilisez des informations d'identification temporaires si vous vous connectez à l' AWS Management Console aide d'une méthode autre qu'un nom d'utilisateur et un mot de passe. Par exemple, lorsque vous accédez à AWS l'aide du lien d'authentification unique (SSO) de votre entreprise, ce processus crée automatiquement des informations d'identification temporaires. Vous créez également automatiquement des informations d'identification temporaires lorsque vous vous connectez à la console en tant qu'utilisateur, puis changez de rôle. Pour plus d'informations sur le changement de rôle, consultez Passage d'un rôle utilisateur à un rôle IAM (console) dans le Guide de l'utilisateur IAM.

Vous pouvez créer manuellement des informations d'identification temporaires à l'aide de l' AWS API AWS CLI or. Vous pouvez ensuite utiliser ces informations d'identification temporaires pour y accéder AWS. 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.

Autorisations principales interservices pour Aurora DSQL

Prend en charge les sessions d'accès direct (FAS) : oui

Lorsque vous utilisez un utilisateur ou un rôle IAM pour effectuer des actions AWS, vous êtes considéré comme un mandant. Lorsque vous utilisez certains services, vous pouvez effectuer une action qui initie une autre action dans un autre service. FAS utilise les autorisations du principal appelant et Service AWS, associées Service AWS à la demande, pour adresser des demandes aux services en aval. Les demandes FAS ne sont effectuées que lorsqu'un service reçoit une demande qui nécessite des interactions avec d'autres personnes Services AWS ou des ressources pour être traitée. Dans ce cas, vous devez disposer d'autorisations nécessaires pour effectuer les deux actions. Pour plus de détails sur une politique lors de la formulation de demandes FAS, consultez Transmission des sessions 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 associées à un rôle de service peut interrompre les fonctionnalités d'Aurora DSQL. Modifiez les rôles de service uniquement lorsque Aurora DSQL fournit des instructions à cet effet.

Rôles liés à un service pour Aurora DSQL

Prend en charge les rôles liés aux services : 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 de rôles liés à un service pour Aurora DSQL, consultez. Utilisation de rôles liés à un service dans 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 ou à modifier des ressources Aurora DSQL. Ils ne peuvent pas non plus effectuer de tâches à l'aide de l'API AWS Management Console, AWS Command Line Interface (AWS CLI) ou de AWS l'API. 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. L'administrateur peut ensuite ajouter les politiques IAM aux rôles et les utilisateurs peuvent assumer les rôles.

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 la référence d'autorisation de service.

Rubriques

- Bonnes pratiques en matière de politiques
- Utilisation de la console Aurora DSQL
- Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations

Bonnes pratiques en matière de politiques

Les politiques basées sur l'identité déterminent si quelqu'un peut créer, accéder 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 AWS 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 minimal 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": "*"
        }
    ]
}
```

Résolution des problèmes d'identité et d'accès Amazon Aurora DSQL

Utilisez les informations suivantes pour diagnostiquer et résoudre les problèmes courants que vous pouvez rencontrer lors de l'utilisation d'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` système essaie d'utiliser la console pour afficher les détails de la *`my-dsql-cluster`* ressource mais ne dispose pas des *`GetCluster`* autorisations 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 un message d'erreur indiquant que vous n'êtes pas autorisé à effectuer l'`iam:PassRole`action, vos politiques doivent être mises à jour pour 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, un utilisateur doit 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 effectuer une action dans Aurora DSQL. Toutefois, l'action nécessite que le service ait des autorisations accordées par un rôle de service. Mary ne dispose pas des autorisations nécessaires pour transférer 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, consultezComment Amazon Aurora DSQL fonctionne 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.

Utilisation de rôles liés à un service dans Aurora DSQL

Aurora DSQL utilise des rôles liés à un service AWS Identity and Access Management (IAM). Un rôle lié à un service est un type unique de rôle IAM directement lié 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.

Les rôles liés à un service facilitent le processus de configuration, car il n'est pas nécessaire d'ajouter manuellement les autorisations nécessaires pour utiliser Aurora DSQL. Lorsque vous créez un cluster, Aurora DSQL crée automatiquement un rôle lié à un service pour vous. Vous ne pouvez supprimer le rôle lié à un service qu'après avoir supprimé tous vos clusters. Cela protège vos ressources Aurora DSQL, car vous ne pouvez pas supprimer par inadvertance les autorisations nécessaires pour accéder aux ressources.

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 avec un Yes (Oui) dans la colonne Service-Linked Role (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 de rôle liées 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. Le message d'erreur suivant peut s'afficher : `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 :
>
> ```
> {
> "Sid" : "CreateDsqlServiceLinkedRole",
>     "Effect" : "Allow",
>     "Action" : "iam:CreateServiceLinkedRole",
>     "Resource" : "*",
>     "Condition" : {
>       "StringEquals" : {
>         "iam:AWSServiceName" : "dsql.amazonaws.com"
>       }
>     }
>   }
> ```
>
> Pour plus d'informations, consultez la section Autorisations de rôle liées 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 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 pas d'entité inutilisée qui n'est pas activement surveillée ou maintenue.

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 la section Créer un rôle lié à un service dans le guide de l'utilisateur IAM.

Régions prises en charge pour les rôles liés aux services Aurora DSQL

Aurora DSQL prend en charge l'utilisation de rôles liés à un service dans toutes les régions où le service est disponible. Pour de plus amples informations, veuillez consulter AWS Régions et points de terminaison.

Utilisation de clés de condition IAM avec Amazon Aurora DSQL

Lorsque vous accordez des autorisations dans Aurora DSQL, vous pouvez spécifier les conditions qui déterminent la manière dont une politique d'autorisations prend effet. Vous trouverez ci-dessous des exemples d'utilisation des clés de condition dans les politiques d'autorisation SQL d'Aurora.

Exemple 1 : accorder l'autorisation de créer un cluster dans un environnement spécifique Région AWS

La politique suivante autorise la création de 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 `Resource` section de la stratégie.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            # Control where clusters can be created
```

```
            "Action": ["CreateCluster"],
            "Resource": [
                "arn:aws:dsql:us-east-1:*:cluster/*",
                "arn:aws:dsql:us-east-2:*:cluster/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Exemple 2 : accorder l'autorisation de créer un cluster multirégional dans des domaines spécifiques Région AWS

La politique suivante autorise la création de clusters multirégionaux dans les régions de l'est des États-Unis (Virginie du Nord) et de l'est des États-Unis (Ohio). Cette politique utilise l'ARN de la ressource pour limiter les régions autorisées. Aurora DSQL ne peut donc créer des clusters multirégionaux que si cet ARN est spécifié dans la `Resource` section de la stratégie. Notez que la création de clusters multirégionaux nécessite également les `AddPeerCluster` autorisations `PutMultiRegionPropertiesPutWitnessRegion`, et 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 multirégional avec une région témoin spécifique

La politique suivante utilise une clé de `dsql:WitnessRegion` condition Aurora DSQL et permet à un utilisateur de créer des clusters multirégionaux avec une région témoin dans l'ouest des États-Unis (Oregon). Si vous ne spécifiez pas la `dsql:WitnessRegion` condition, vous pouvez utiliser n'importe quelle région comme région témoin.

```
{
    "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

La sécurité est la priorité absolue chez AWS. 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 de votre côté 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 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. AWS fournit les ressources suivantes pour faciliter la mise en conformité :

- Conformité et gouvernance de la sécurité : ces guides de mise en œuvre de solutions traitent des considérations architecturales et fournissent les étapes à suivre afin de déployer des fonctionnalités de sécurité et de conformité.
- Référence des services éligibles HIPAA : liste les services éligibles HIPAA. Tous ne Services AWS sont pas éligibles à la loi HIPAA.
- AWS Ressources de https://aws.amazon.com/compliance/resources/ de conformité — Cette collection de classeurs et de guides peut s'appliquer à votre secteur d'activité et à votre région.
- AWS Guides de conformité destinés aux clients — Comprenez le modèle de responsabilité partagée sous l'angle de la conformité. Les guides résument les meilleures pratiques en matière de sécurisation Services AWS et décrivent les directives relatives aux contrôles de sécurité dans de nombreux cadres (notamment le National Institute of Standards and Technology (NIST), le Payment Card Industry Security Standards Council (PCI) et l'Organisation internationale de normalisation (ISO)).
- Évaluation des ressources à l'aide des règles du guide du AWS Config développeur : le AWS Config service évalue dans quelle mesure les configurations de vos ressources sont conformes aux pratiques internes, aux directives du secteur et aux réglementations.
- AWS Security Hub— Cela Service AWS fournit une vue complète de votre état de sécurité interne AWS. Security Hub utilise des contrôles de sécurité pour évaluer vos ressources AWS et vérifier votre conformité par rapport aux normes et aux bonnes pratiques du secteur de la sécurité. Pour obtenir la liste des services et des contrôles pris en charge, consultez Référence des contrôles Security Hub.
- Amazon GuardDuty — Cela Service AWS détecte les menaces potentielles qui pèsent sur vos charges de travail Comptes AWS, vos conteneurs et vos données en surveillant votre environnement pour détecter toute activité suspecte et malveillante. GuardDuty peut vous aider à répondre à diverses exigences de conformité, telles que la norme PCI DSS, en répondant aux exigences de détection des intrusions imposées par certains cadres de conformité.

- AWS Audit Manager— Cela vous Service AWS permet d'auditer en permanence votre AWS utilisation afin de simplifier la gestion des risques et la conformité aux réglementations et aux normes du secteur.

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 région unique 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 multirégionaux 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 les clients de l'application ne Région AWS sont pas accessibles.

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 région unique ou multirégionale. Pour de plus amples informations, veuillez consulter Backup 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 multirégionaux fournissent des fonctionnalités complètes de réplication entre 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 n'ont pas de 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.

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 redimensionne automatiquement les répliques en lecture, à la hausse comme à la baisse, en fonction de la fréquence d'accès en lecture.

Les répliques de stockage en cluster sont distribuées au sein d'un parc de stockage mutualisé. Si un composant ou une AZ est endommagé, Aurora DSQL redirige automatiquement l'accès aux composants survivants et répare de manière asynchrone les répliques manquantes. Une fois qu'Aurora DSQL a corrigé les répliques défectueuses, Aurora DSQL les ajoute automatiquement au quorum de stockage et les met à la disposition de votre cluster.

Haute disponibilité

Par défaut, les clusters mono-régionaux et multirégionaux dans Aurora DSQL sont actifs-actifs, et il n'est pas nécessaire de provisionner, configurer ou 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 multirégionaux Aurora DSQL. Cela signifie que les clusters multirégionaux 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 région unique et de 99,999 % pour les clusters multirégionaux.

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 AWS 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 local, consultez le guide de l'utilisateur et le guide de AWS Direct Connect l'utilisateur du AWS Site-to-Site VPN VPN.

Pour des informations générales sur les points de terminaison d'interface, consultez la section Accéder 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 `get`create, telles que`update`,`delete`,, et `list` sur les clusters Aurora SQL. 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.

Lister les clusters à l'aide d'une URL de point de

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.dynamodb.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 relatives aux terminaux

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, conforme et conforme aux exigences de gouvernance et de contrôle d'accès spécifiques de votre organisation.

Exemple : politique d'accès complète à Aurora DSQL

La politique suivante accorde un accès complet 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-xxxxxxxxxxxxxxxx \
    --region region \
    --policy-document '{
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": "*",
          "Action": "dsql:*",
          "Resource": "*"
        }
      ]
    }'
```

Exemple : politique d'accès restreint à Aurora DSQL

La politique 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'`--no-private-dns-enabled`indicateur 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);
```

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

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
```

Résolution des problèmes liés à AWS PrivateLink

Problèmes courants et solutions correspondantes

Le tableau suivant répertorie les problèmes courants et les solutions liés AWS PrivateLink à Aurora DSQL.

Problème	Cause possible	Solution
Délai de connexion	Le groupe de sécurité n'est pas correctem ent configuré	Utilisez Amazon VPC Reachability Analyzer pour vous assurer que votre configuration réseau autorise le trafic sur le port 5432.
Défaillance de résolution DNS	DNS privé non activé	Vérifiez que le point de terminaison Amazon VPC a été créé avec le DNS privé activé.
Échec de l'authent ification	Informations d'identif ication incorrectes ou jeton expiré	Générez un nouveau jeton d'authentification et vérifiez le nom d'utilisateur.
Nom du 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 d'Amazon Aurora DSQL
- Documentation AWS PrivateLink
- Accédez aux AWS services via AWS PrivateLink

Analyse de configuration et de vulnérabilité 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 la confusion des 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 d'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 recommandons d'utiliser les clés de contexte de condition `aws:SourceAccount`globale `aws:SourceArn`et les clés de contexte dans les politiques de ressources afin de limiter les autorisations qu'Amazon Aurora DSQL accorde à un autre service à la ressource. 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:`*`servicename`*`:*:`*`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 vous pouvez utiliser les clés de contexte de condition `aws:SourceAccount` globale `aws:SourceArn` et les clés de contexte dans Aurora DSQL pour éviter le problème de confusion des adjoints.

JSON

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

Bonnes pratiques de sécurité pour Aurora DSQL

Aurora DSQL fournit un certain nombre de fonctionnalités de sécurité à prendre en compte lors de l'élaboration et de la mise en œuvre de 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

Meilleures pratiques en matière de sécurité Detective 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.

Étiquetez vos ressources Aurora DSQL à des fins d'identification et d'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 les exigences telles que le chiffrement.
- Confidentialité : identifiant du niveau spécifique de confidentialité des données pris en charge par une ressource.
- Environnement : utilisé pour faire la distinction entre l'infrastructure 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.

Utilisez les 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 EC2 les instances. 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.

Utilisez les politiques IAM pour l'autorisation de base Aurora DSQL.

Lorsque vous accordez des autorisations, vous décidez qui les obtient, pour quelles opérations d'API Aurora DSQL elles obtiennent des autorisations 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 effectuer des opérations sur les ressources Aurora DSQL. Des limites d'autorisations sont également disponibles pour les entités IAM, qui vous permettent de définir le maximum d'autorisations 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, voir Accès à Aurora DSQL et Comprendre l'authentification et l'autorisation pour Aurora DSQL.

Utilisation **`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.

Mettez à jour 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 des ressources dans Aurora DSQL

Dans AWS, les balises sont des paires clé-valeur définies par l'utilisateur que vous définissez et associez aux ressources SQL Aurora telles que les clusters. Les balises sont facultatives. Si vous fournissez une clé, la valeur est facultative.

Vous pouvez utiliser les balises AWS Management Console AWS CLI, le ou AWS SDKs pour ajouter, répertorier et supprimer des balises sur les clusters Aurora SQL. Vous pouvez ajouter des balises pendant et après la création du cluster à l'aide de la AWS console. Pour étiqueter un cluster après sa création, AWS CLI utilisez l'`TagResource`opération.

Marquer les clusters avec un nom

Aurora DSQL crée des clusters avec un identifiant unique mondial attribué sous la forme d'Amazon Resource Name (ARN). Si vous souhaitez attribuer un nom convivial à votre cluster, nous vous recommandons d'utiliser un tag.

Si vous créez une console avec la console Aurora DSQL, Aurora DSQL crée automatiquement une balise. Cette balise possède une clé nommée Name 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 Name associée à une valeur, vous pouvez voir cette valeur dans l'ensemble de 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 blancs et les symboles suivants : _. :/= + - @.
- Les clés et les valeurs sont sensibles à la casse.

Marquage des notes d'utilisation

Lorsque vous utilisez des balises dans Aurora DSQL, tenez compte des points suivants.

- Lorsque vous utilisez les opérations de l'API DSQL AWS CLI ou Aurora, assurez-vous de fournir le nom de ressource Amazon (ARN) pour la ressource Aurora DSQL avec laquelle travailler. Pour plus d'informations, consultez le format Amazon Resource Name (ARNs) pour les ressources SQL Aurora.
- 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 les politiques IAM. Vous pouvez les utiliser pour gérer l'accès aux clusters Aurora DSQL et pour contrôler les actions qui peuvent être appliquées à ces ressources. Pour en savoir plus, consultez la section Contrôle de l'accès aux AWS ressources à l'aide de balises.
- Vous pouvez utiliser des tags pour diverses autres activités AWS. Pour en savoir plus, consultez la section Stratégies de balisage courantes.

Considérations relatives à l'utilisation d'Amazon Aurora DSQL

Tenez compte des comportements suivants lorsque vous travaillez avec Amazon Aurora DSQL. Pour plus d'informations sur la compatibilité et la prise en charge de PostgreSQL, consultez. Compatibilité des fonctionnalités SQL dans Aurora DSQL Pour les quotas et les limites, voirQuotas de cluster et limites de base de données dans Amazon Aurora DSQL.

- Aurora DSQL ne termine pas les `COUNT(*)` opérations avant l'expiration du délai d'expiration des transactions pour les grandes tables. Pour récupérer le nombre de lignes d'une table à partir du catalogue système, consultez la section Utilisation des tables et des commandes du système dans Aurora DSQL.
- Les pilotes qui appellent `PG_PREPARED_STATEMENTS` peuvent fournir une vue incohérente des instructions préparées mises en cache pour le cluster. Le nombre d'instructions préparées par connexion peut être supérieur au nombre attendu pour le même cluster et le même rôle IAM. Aurora DSQL ne conserve pas les noms des instructions que vous préparez.
- Dans de rares scénarios de détérioration de clusters liés multirégionaux, le rétablissement de la disponibilité des validations de transactions peut prendre plus de temps que prévu. En général, les opérations de restauration automatique du cluster peuvent entraîner un contrôle de simultanéité transitoire ou des erreurs de connexion. Dans la plupart des cas, vous ne constaterez les effets que pour un pourcentage de votre charge de travail. Lorsque ces erreurs de transit s'affichent, réessayez votre transaction ou reprenez contact avec votre client.
- Certains clients SQL, tels que Datagrip, font des appels étendus aux métadonnées du système pour renseigner les informations de schéma. Aurora DSQL ne prend pas en charge toutes ces informations et renvoie des erreurs. Ce problème n'affecte pas la fonctionnalité des requêtes SQL, mais il peut affecter l'affichage du schéma.
- Le rôle d'administrateur dispose d'un ensemble d'autorisations liées aux tâches de gestion de base de données. Par défaut, ces autorisations ne s'étendent pas aux objets créés par d'autres utilisateurs. Le rôle d'administrateur ne peut pas accorder ou révoquer des autorisations sur ces objets créés par les utilisateurs à d'autres utilisateurs. L'utilisateur administrateur peut s'octroyer n'importe quel autre rôle pour obtenir les autorisations nécessaires sur ces objets.

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 clusters

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 SQL Aurora
Nombre maximal de clusters à région unique par Compte AWS	20 clusters	Oui	Code d'erreur de l'API `ServiceQuotaExceededExc`
Nombre maximal de clusters multirégionaux par Compte AWS	5 clusters	Oui	Code d'erreur de l'API `ServiceQuotaExceededExc`
Stockage maximal par cluster	Limite par défaut de 10 TiB, jusqu'à 128 TiB avec augmentat	Oui	`DISK_FULL(53100)`

Description	Limite par défaut	Configurable ?	Code d'erreur SQL Aurora
	ion de limite approuvée		
Nombre maximum de connexions par cluster	10 000 connexions	Oui	`TOO_MANY_CONNECTIONS(53300)`
Débit de connexion maximal par cluster	100 connexions par seconde	Non	`CONFIGURED_LIMIT_EXCEEDED(53400)`
Capacité maximale de rafale de connexions 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 recharge des connexions	100 connexions par seconde	Non	Aucun code d'erreur

Limites de base de données dans Aurora DSQL

Le tableau suivant décrit les limites des bases de données dans Aurora DSQL.

Description	Limite par défaut	Configurable ?	Code d'erreur SQL Aurora	Message d'erreur
Taille combinée maximale 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 un tableau	2 Mo	Non	54000	ERROR: maximum row size excee
Taille maximale d'une colonne ne faisant 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 maximum de colonnes dans un tableau	255	Non	54011	ERROR: tables can have at mos

Description	Limite par défaut	Configurable ?	Code d'erreur SQL Aurora	Message d'erreur
Nombre maximum 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 et d'index pouvant être mutées dans un bloc de transacti on	3 000 lignes par transacti on. Consultez Considérations relatives à Aurora DSQL relatives à la compatibi lité avec PostgreSQL.	Non	54000	ERROR: transaction row limit
Quantité de mémoire de base maximale qu'une opération de requête peut utiliser	128 MiB par transaction	Non	53200	ERROR: query requires too muc out of memory.

Description	Limite par défaut	Configurable ?	Code d'erreur SQL Aurora	Message d'erreur
Nombre maximum 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 maximale de transaction	5 minutes	Non	54000	ERROR: transaction age limit exceeded
Durée maximale de connexion	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 définition de vue	2 Mo	Non	54000	ERROR: view definition too la

Pour connaître les limites de type de données spécifiques à Aurora DSQL, consultezTypes de données pris en charge dans Aurora DSQL.

Référence de l'API SQL Aurora

Outre le AWS Management Console et le AWS Command Line Interface (AWS CLI), Aurora DSQL fournit également une interface API. Vous pouvez utiliser les opérations d'API 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 la description des codes d'erreur, veuillez consulter la page Erreurs courantes.

Pour plus d'informations à ce sujet AWS CLI, consultez la AWS Command Line Interface référence pour Aurora DSQL.

Résolution des problèmes dans Aurora DSQL

> **Note**
>
> Les rubriques suivantes fournissent des conseils de résolution des erreurs et des problèmes que vous pouvez rencontrer lors de l'utilisation d'Aurora DSQL. Si vous trouvez un problème qui n'est pas répertorié ici, contactez le AWS support

Rubriques

Résolution des erreurs de connexion

erreur : code d'erreur SSL non reconnu : 6

Cause : 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 la version de votre 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 d'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 assumé 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 en savoir plus sur le fonctionnement d'IAM avec Aurora DSQL, voir Comprendre l'authentification et l'autorisation pour Aurora DSQL et dans AWS Identity and Access Management Aurora DSQL.

Une erreur s'est produite (InvalidAccessKeyId) lors 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, voir Pourquoi les demandes sont signées.

Le rôle IAM n'existe pas <role>

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.

Résolution des erreurs d'autorisation

Rôle non pris en charge <role>

Aurora DSQL ne prend pas en charge cette `GRANT` opération. Consultez la section Sous-ensembles de commandes PostgreSQL pris en charge dans Aurora DSQL.

Impossible d'établir la confiance avec le rôle <role>

Aurora DSQL ne prend pas en charge cette `GRANT` opération. Consultez la section Sous-ensembles de commandes PostgreSQL pris en charge dans Aurora DSQL.

Le rôle n'existe pas <role>

Aurora DSQL n'a pas pu trouver l'utilisateur de base de données spécifié. Voir Autoriser les rôles de base de données personnalisés pour se connecter à un cluster.

ERREUR : autorisation refusée pour accorder la confiance à 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 en savoir plus, voir Autoriser les rôles de base de données à utiliser le SQL dans une base de données.

ERREUR : le rôle doit avoir l'attribut LOGIN <role>

Tous les rôles de base de données que vous créez doivent disposer de `LOGIN` cette autorisation.

Pour corriger cette erreur, assurez-vous d'avoir créé le rôle PostgreSQL avec l'autorisation requise. `LOGIN` Pour plus d'informations, consultez CREATE ROLE et ALTER ROLE dans la documentation de PostgreSQL.

ERREUR : le rôle ne peut pas être supprimé car certains objets en dépendent <role>

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 en savoir plus, consultez la section 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 en savoir plus sur les fonctionnalités prises en charge, consultez la section Fonctionnalités PostgreSQL prises en charge dans Aurora DSQL.

Erreur : SELECT FOR UPDATE dans une transaction en lecture seule est une opération interdite

Vous tentez une opération qui n'est pas autorisée dans une transaction en lecture seule. Pour en savoir plus, voir Comprendre le contrôle de simultanéité dans Aurora DSQL.

Erreur : utiliser à la **`CREATE INDEX ASYNC`** place

Pour créer un index sur une table contenant des lignes existantes, vous devez utiliser la `CREATE INDEX ASYNC` commande. Pour en savoir plus, consultez la section Création d'index de manière asynchrone dans Aurora DSQL.

Résolution des erreurs OCC

OC000 « ERREUR : une mutation entre en conflit avec une autre transaction, réessayez si nécessaire »

OC001 « ERREUR : le schéma a été mis à jour par une autre transaction, réessayez si nécessaire »

Votre session PostgreSQL contenait une copie en cache du catalogue de schémas. Cette copie mise en cache était valide au moment du chargement. Appelons l'heure 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 depuis le stockage à l'instant 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 T2 est la V2.

Lorsque vous réessayez à l'heure T3 depuis la session d'origine, Aurora DSQL actualise le cache du catalogue. La transaction au T3 utilise le catalogue V2. Aurora DSQL terminera la transaction tant qu'aucune autre modification du catalogue n'est apportée depuis 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 les instructions relatives à la validation et à l'installation de ce certificat.
2. La variable d'`PGSSLROOTCERT`environnement pointe vers le bon fichier de certificat.
3. Le fichier de certificat possède les autorisations appropriées.

Code d'erreur SSL non reconnu : 6

Cette erreur se produit avec les clients PostgreSQL inférieurs 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 Connexion depuis Windows instructions.

Historique du document pour le guide de l'utilisateur Amazon Aurora DSQL

Le tableau suivant décrit les versions de documentation pour Aurora DSQL.

Modification	Description	Date
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	Ajoute la capacité d'effectuer des opérations de sauvegarde et de restauration pour les clusters SQL Aurora, notamment le démarrage, l'arrêt et la surveillance des tâches. Il permet également d'utiliser des clés KMS gérées par le client pour le chiffrement des clusters. Pour plus d'informations, voir AmazonAuroraDSQLFullAccès	21 mai 2025

	et utilisation des rôles liés à un service dans Aurora DSQL.	
AmazonAuroraDSQLConsoleFullAccess update	Ajoute la capacité d'effectuer des opérations de sauvegarde et de restauration pour les clusters SQL Aurora via le AWS Console Home. Cela inclut le démarrage, l'arrêt et le suivi des tâches. Il prend également en charge l'utilisation de clés KMS gérées par le client pour le chiffrement et le lancement du cluster. 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 mise à jour	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	Ajoute de nouvelles autorisations à Aurora DSQL pour prendre en charge la gestion de clusters multirégionaux et la connexion des points de terminaison VPC. Les nouvelles autorisations incluent : `PutMultiRegionProperties` `PutWitnessRegion` `AddPeerCluster` `RemovePeerCluster` `GetVpcEndpointServiceName` . Consultez AmazonAuroraDSQLConsoleFullAccessetUtilisation des rôles liés à un service dans Aurora DSQL.	13 mai 2025
AuroraDsqlServiceLinkedRolePolicy update	Permet de publier des métriques `AWS/AuroraDSQL` et des `AWS/Usage` `CloudWatch` espaces de noms 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 SQL	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 SQL Amazon Aurora à l'aide AWS PrivateLink de.	8 mai 2025
Première version	Première publication 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.