aws

使用者指南

Amazon Aurora DSQL

Amazon Aurora DSQL: 使用者指南

Table of Contents

什麼是 Amazon Aurora DSQL？ 1
使用情況 1
主要功能 1
AWS 區域 可用性 2
多區域叢集 3
定價 4
後續步驟？ 4
開始使用 5
先決條件 5
存取 Aurora DSQL 6
主控台存取 6
SQL 用戶端 6
PostgreSQL 通訊協定 9
建立單一區域叢集 10
連接至叢集 11
執行 SQL 命令 12
建立多區域叢集 13
身分驗證和授權 17
管理您的叢集 17
連線至您的叢集 17
PostgreSQL 和 IAM 角色 18
搭配 Aurora DSQL 使用 IAM 政策動作 19
使用 IAM 政策動作連線至叢集 19
使用 IAM 政策動作來管理叢集 19
使用 IAM 和 PostgreSQL 撤銷授權 20
產生身分驗證字符 21
主控台 21
AWS CloudShell 22
AWS CLI 23
Aurora DSQL SDKs 24
資料庫角色和 IAM 身分驗證 33
IAM 角色 33
IAM 使用者 33
連接 33

Query 33
撤銷 34
Aurora DSQL 和 PostgreSQL 35
相容性重點 35
關鍵架構差異 36
SQL 相容性 36
支援的資料類型 36
支援的 SQL 功能 41
SQL 命令支援的子集 44
不支援的 PostgreSQL 功能 54
並行控制 57
交易衝突 57
最佳化交易效能的指導方針 58
DDL 和分散式交易 58
主索引鍵 59
資料結構和儲存 59
選擇主索引鍵的準則 60
非同步索引 60
語法 61
參數 61
使用須知 62
建立索引 62
查詢索引 63
唯一索引建置失敗 64
唯一性違規 65
系統資料表和命令 67
系統表 67
ANALYZE 命令 76
管理 Aurora DSQL 叢集 78
單一區域叢集 78
建立叢集 78
描述叢集 79
更新叢集 79
刪除叢集 80
列出叢集 81
多區域叢集 81

連線至多區域叢集 81
建立多區域叢集 82
刪除多區域叢集 85
AWS CloudFormation 87
使用 Aurora DSQL 進行程式設計 90
..... 90
AWS SDKs、驅動程式和範本程式碼 90
轉接器和方言 90
範例 91
AWS CLI 94
CreateCluster 94
GetCluster 94
UpdateCluster 95
DeleteCluster 96
ListClusters 96
多區域叢集上的 GetCluster 97
建立、讀取、更新、刪除叢集 98
建立叢集 98
取得叢集 130
更新 叢集 138
刪除叢集 147
教學課程 171
AWS Lambda 教學課程 171
備份和還原 177
入門 AWS Backup 177
還原備份 177
監控和合規 178
其他資源 178
監控和記錄 180
檢視叢集狀態 180
叢集狀態 180
檢視叢集狀態 181
使用 CloudWatch 進行監控 182
可觀測性 182
用量 183
使用 CloudTrail 進行記錄 185

管理事件 185
資料事件 186
安全 188
AWS 受管政策 189
AmazonAuroraDSQLFullAccess 189
AmazonAuroraDSQLReadOnlyAccess 190
AmazonAuroraDSQLConsoleFullAccess 190
AuroraDSQLServiceRolePolicy 191
政策更新 191
資料保護 195
資料加密 195
SSL/TLS 憑證 196
資料加密 195
KMS 金鑰類型 202
靜態加密 203
使用 KMS 和資料金鑰 204
授權您的 KMS 金鑰 206
加密內容 207
監控 AWS KMS 208
建立加密的叢集 210
移除或更新金鑰 212
考量事項 214
身分與存取管理 215
目標對象 215
使用身分驗證 216
使用政策管理存取權 218
Aurora DSQL 如何與 IAM 搭配使用 220
身分型政策範例 226
故障診斷 228
使用服務連結角色 230
Aurora DSQL 的服務連結角色許可 230
建立服務連結角色 231
編輯服務連結角色 231
刪除服務連結角色 231
Aurora DSQL 服務連結角色支援的區域 231
使用 IAM 條件金鑰 232

在特定區域中建立叢集 232
在特定區域中建立多區域叢集 232
建立具有特定見證區域的多區域叢集 233
事件回應 234
法規遵循驗證 234
恢復能力 235
備份和還原 235
複寫 236
高可用性 236
基礎設施安全性 236
使用 管理叢集 AWS PrivateLink 237
組態與漏洞分析 245
預防跨服務混淆代理人 246
安全最佳實務 247
偵測性安全最佳實務 247
預防性安全最佳實務 248
標記 資源 250
Name tag (名稱標籤) 250
標記需求 250
標記用量備註 250
考量事項 252
配額和限制 253
叢集配額 253
資料庫限制 254
API 參考 257
故障診斷 258
連線錯誤 258
身分驗證錯誤 258
授權錯誤 259
SQL 錯誤 260
OCC 錯誤 260
SSL/TLS 連線 261
文件歷史紀錄 262
..... cclxvi

什麼是 Amazon Aurora DSQL？

Amazon Aurora DSQL 是針對交易工作負載最佳化的無伺服器分散式關聯式資料庫服務。Aurora DSQL 提供幾乎無限制的擴展，不需要您管理基礎設施。主動-主動高可用性架構提供 99.99% 的單一區域和 99.999% 的多區域可用性。

何時使用 Aurora DSQL

Aurora DSQL 已針對受益於 ACID 交易和關聯式資料模型的交易工作負載進行最佳化。由於它是無伺服器，Aurora DSQL 非常適合微型服務、無伺服器和事件驅動架構的應用程式模式。Aurora DSQL 與 PostgreSQL 相容，因此您可以使用熟悉的驅動程式、物件關聯式映射 (ORMs)、架構和 SQL 功能。

Aurora DSQL 會自動管理系統基礎設施，並根據工作負載擴展運算、I/O 和儲存。由於您沒有要佈建或管理的伺服器，因此您不需要擔心與佈建、修補或基礎設施升級相關的維護停機時間。

Aurora DSQL 可協助您建置和維護隨時可在任何規模使用的企業應用程式。主動-主動無伺服器設計可自動化故障復原，因此您不需要擔心傳統的資料庫容錯移轉。您的應用程式受益於異地同步備份和多區域可用性，您不需要擔心最終一致性或與容錯移轉相關的遺失資料。

Aurora DSQL 中的主要功能

下列重要功能可協助您建立無伺服器分散式資料庫，以支援您的高可用性應用程式：

分散式架構

Aurora DSQL 由下列多租用戶元件組成：

- 轉送和連線
- 運算和資料庫
- 交易日誌、並行控制和隔離
- 儲存

控制平面會協調上述元件。每個元件提供跨三個可用區域 AZs) 的備援，可在元件故障時自動叢集擴展和自我修復。若要進一步了解此架構如何支援高可用性，請參閱 Amazon Aurora DSQL 中的彈性。

單一區域和多區域叢集

Aurora DSQL 叢集提供下列優點：

- 同步資料複寫
- 一致的讀取操作
- 自動故障復原
- 跨多個AZs或區域的資料一致性

如果基礎設施元件故障，Aurora DSQL 會自動將請求路由至運作狀態良好的基礎設施，而無需手動介入。Aurora DSQL 提供原子、一致性、隔離和耐久性 (ACID) 交易，具有強大的一致性、快照隔離、原子性，以及跨可用區域和跨區域耐久性。

多區域對等叢集提供與單一區域叢集相同的彈性和連線能力。但是，它們透過提供兩個區域端點來提高可用性，每個對等叢集區域中各一個端點。對等叢集的兩個端點都呈現單一邏輯資料庫。它們可用於並行讀取和寫入操作，並提供強大的資料一致性。您可以建置同時在多個區域中執行的應用程式，以獲得效能和彈性，並知道讀者永遠會看到相同的資料。

與 PostgreSQL 資料庫的相容性

Aurora DSQL 中的分散式資料庫層 （運算） 是以 PostgreSQL 的目前主要版本為基礎。您可以使用熟悉的 PostgreSQL 驅動程式和工具連線至 Aurora DSQL，例如 `psql`。Aurora DSQL 目前與 PostgreSQL 第 16 版相容，並支援 PostgreSQL 功能、表達式和資料類型的子集。如需支援的 SQL 功能的詳細資訊，請參閱 Aurora DSQL 中的 SQL 功能相容性。

Aurora DSQL 的區域可用性

使用 Amazon Aurora DSQL，您可以在多個 之間部署資料庫執行個體 AWS 區域 ，以支援全域應用程式並滿足資料駐留需求。區域可用性決定您可以在何處建立和管理 Aurora DSQL 資料庫叢集。需要設計高可用性、全球分散式資料庫系統的資料庫管理員和應用程式架構師，通常需要了解區域對其工作負載的支援。常見的使用案例包括設定跨區域災難復原、從地理位置更接近的資料庫執行個體為使用者提供服務以降低延遲，以及在特定位置維護資料副本以確保合規性。

下表顯示目前可使用 Aurora DSQL AWS 區域 的 ，以及每個 DSQL 的端點 AWS 區域。

支援的 AWS 區域 和 端點

區域名稱	區域	端點	通訊協定
美國東部 (維吉尼亞北部)	us-east-1	dsql.us-east-1.api.aws	HTTPS
美國東部 (俄亥俄)	us-east-2	dsql.us-east-2.api.aws	HTTPS
美國西部 (奧勒岡)	us-west-2	dsql.us-west-2.api.aws	HTTPS

區域名稱	區域	端點	通訊協定
歐洲 (愛爾蘭)	eu-west-1	dsql.eu-west-1.api.aws	HTTPS
歐洲 (倫敦)	eu-west-2	dsql.eu-west-2.api.aws	HTTPS
Europe (Paris)	eu-west-3	dsql.eu-west-3.api.aws	HTTPS
亞太區域 (東京)	ap-northeast-1	dsql.ap-northeast-1.api.aws	HTTPS
亞太區域 (首爾)	ap-northeast-2	dsql.ap-northeast-2.api.aws	HTTPS
亞太區域 (大阪)	ap-northeast-3	dsql.ap-northeast-3.api.aws	HTTPS

Aurora DSQL 的多區域叢集可用性

您可以在特定 AWS 區域集中建立 Aurora DSQL 多區域叢集。每個區域集都會將可在多區域叢集中一起運作的地理相關區域分組。

美國區域

- 美國東部 (維吉尼亞北部)
- 美國東部 (俄亥俄)
- 美國西部 (奧勒岡)

亞太區域

- 亞太區域 (大阪)
- 亞太區域 (首爾)
- 亞太區域 (東京)

歐洲區域

- 歐洲 (愛爾蘭)

- 歐洲 (倫敦)
- Europe (Paris)

重要限制

多區域叢集必須在單一區域集中建立。例如，您無法建立同時包含美國東部（維吉尼亞北部）和歐洲（愛爾蘭）區域的叢集。

> **Important**
>
> Aurora DSQL 目前不支援跨內容多區域叢集。

Aurora DSQL 定價

如需成本資訊，請參閱 Aurora DSQL 定價。

後續步驟？

如需有關 Aurora DSQL 中核心元件的資訊，以及開始使用服務的資訊，請參閱下列內容：

- Aurora DSQL 入門
- Aurora DSQL 中的 SQL 功能相容性
- 存取 Aurora DSQL
- Aurora DSQL 和 PostgreSQL

Aurora DSQL 入門

Amazon Aurora DSQL 是針對交易工作負載最佳化的無伺服器分散式關聯式資料庫。在下列各節中，您將了解如何建立單一區域和多區域 Aurora DSQL 叢集、與其連線，以及建立和載入範例結構描述。您將使用 存取叢集 AWS Management Console ，並使用 psql公用程式與資料庫互動。最後，您將設定並準備好用於測試或生產工作負載的工作 Aurora DSQL 叢集。

主題

- 先決條件
- 存取 Aurora DSQL
- 步驟 1：建立 Aurora DSQL 單一區域叢集
- 步驟 2：連線至您的 Aurora DSQL 叢集
- 步驟 3：在 Aurora DSQL 中執行範例 SQL 命令
- 步驟 4：建立多區域叢集

先決條件

在開始使用 Aurora DSQL 之前，請確定您符合下列先決條件：

- 您的 IAM 身分必須具有登入 的 AWS Management Console許可。
- 您的 IAM 身分必須符合下列任一條件：
  - 存取 以對 中的任何資源執行任何動作 AWS 帳戶
  - IAM 許可iam:CreateServiceLinkedRole和存取 IAM 政策動作的能力 dsql:*
- 如果您在類似 AWS CLI Unix 的環境中使用 ，請確定已安裝 Python 3.8+ 版和 14+ psql版。若要檢查您的應用程式版本，請執行下列命令。

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

  如果您在 AWS CLI 不同的環境中使用 ，請務必手動設定 Python 3.8+ 版和 14+ psql版。
- 如果您打算使用 存取 Aurora DSQL AWS CloudShell，Python 3.8+ 版和 14+ psql版無需額外設定。如需 的詳細資訊 AWS CloudShell，請參閱什麼是 AWS CloudShell？。
- 如果您想要使用 GUI 存取 Aurora DSQL，請使用 DBeaver 或 JetBrains DataGrip。如需詳細資訊，請參閱使用 DBeaver 存取 Aurora DSQL及使用 JetBrains DataGrip 存取 Aurora DSQL。

存取 Aurora DSQL

您可以透過下列技術存取 Aurora DSQL。若要了解如何使用 CLI、APIs和 SDKs，請參閱 存取 Aurora DSQL。

主題

- 透過 存取 Aurora DSQL AWS Management Console
- 使用 SQL 用戶端存取 Aurora DSQL
- 搭配 Aurora DSQL 使用 PostgreSQL 通訊協定

透過 存取 Aurora DSQL AWS Management Console

您可以在 存取 AWS Management Console 適用於 Aurora DSQL 的 https://console.aws.amazon.com/dsql。

使用 SQL 用戶端存取 Aurora DSQL

Aurora DSQL 使用 PostgreSQL 通訊協定。在連線至叢集時，提供已簽署的 IAM 身分驗證字符做為密碼，以使用您偏好的互動式用戶端。身分驗證字符是 Aurora DSQL 使用 AWS Signature 第 4 版動態產生的唯一字元字串。

Aurora DSQL 僅使用權杖進行身分驗證。字符在建立之後不會影響連線。如果您嘗試使用過期的字符重新連線，連線請求會遭拒。如需詳細資訊，請參閱在 Amazon Aurora DSQL 中產生身分驗證字符。

主題

- 使用 psql 存取 Aurora DSQL (PostgreSQL 互動式終端機)
- 使用 DBeaver 存取 Aurora DSQL
- 使用 JetBrains DataGrip 存取 Aurora DSQL

使用 psql 存取 Aurora DSQL (PostgreSQL 互動式終端機)

`psql` 公用程式是 PostgreSQL 的終端型前端。它可讓您以互動方式輸入查詢、將查詢發佈至 PostgreSQL，以及查看查詢結果。若要改善查詢回應時間，請使用 PostgreSQL 第 17 版用戶端。

從 PostgreSQL 下載頁面下載作業系統的安裝程式。如需 的詳細資訊`psql`，請參閱 https：//https://www.postgresql.org/docs/current/app-psql.htm。

如果您已 AWS CLI 安裝 ，請使用下列範例來連接至您的叢集。您可以使用psql預先安裝的 AWS CloudShell，也可以psql直接安裝 。

```
# 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
```

使用 DBeaver 存取 Aurora DSQL

DBeaver 是一種以 GUI 為基礎的開放原始碼資料庫工具。您可以使用它來連線至並管理您的資料庫。若要下載 DBeaver，請參閱 DBeaver 社群網站上的下載頁面。下列步驟說明如何使用 DBeaver 連線至您的叢集。

在 DBeaver 中設定新的 Aurora DSQL 連線

1. 選擇新的資料庫連線。
2. 在新資料庫連線視窗中，選擇 PostgreSQL。
3. 在連線設定/主要索引標籤中，選擇依：主機連線，然後輸入下列資訊。
    - 主機 – 使用您的叢集端點。

      資料庫 – 輸入 `postgres`

      身分驗證 – 選擇 `Database Native`

      使用者名稱 – 輸入 `admin`

密碼 – 產生身分驗證字符。複製產生的字符並將其用作您的密碼。

4. 忽略任何警告，並將您的身分驗證字符貼到 DBeaver 密碼欄位中。

> **Note**
>
> 您必須在用戶端連線中設定 SSL 模式。Aurora DSQL 支援 `SSLMODE=require`。Aurora DSQL 在伺服器端強制執行 SSL 通訊，並拒絕非 SSL 連線。

5. 您應該連接到叢集，並且可以開始執行 SQL 陳述式。

> **Important**
>
> DBeaver 為 PostgreSQL 資料庫 （例如 Session Manager 和 Lock Manager) 提供的管理功能不適用於資料庫，因為它是唯一的架構。可存取時，這些畫面不會提供有關資料庫運作狀態或狀態的可靠資訊。

DBeaver 的身分驗證憑證過期

建立的工作階段會保持身分驗證最多 1 小時，或直到 DBeaver 中斷連線或逾時為止。若要建立新的連線，請在連線設定的密碼欄位中提供有效的身分驗證字符。嘗試開啟新的工作階段 （例如，列出新的資料表或新的 SQL 主控台） 會強制新的身分驗證嘗試。如果連線設定中設定的身分驗證字符不再有效，則新工作階段會失敗，且 DBeaver 會讓先前開啟的所有工作階段失效。使用 `expires-in`選項選擇 IAM 身分驗證字符的持續時間時，請記住這一點。

使用 JetBrains DataGrip 存取 Aurora DSQL

JetBrains DataGrip 是跨平台 IDE，可用於 SQL 和資料庫，包括 PostgreSQL。DataGrip 包含具有智慧型 SQL 編輯器的強大 GUI。若要下載 DataGrip，請前往 JetBrains 網站上的下載頁面。

在 JetBrains DataGrip 中設定新的 Aurora DSQL 連線

1. 選擇新資料來源，然後選擇 PostgreSQL。
2. 在資料來源/一般索引標籤中，輸入下列資訊：

   - 主機 – 使用您的叢集端點。

     連接埠 – Aurora DSQL 使用 PostgreSQL 預設： `5432`

資料庫 – Aurora DSQL 使用 的 PostgreSQL 預設值 `postgres`

身分驗證 – 選擇 `User & Password` 。

使用者名稱 – 輸入 `admin`。

密碼 – 產生字符並貼到此欄位。

URL – 請勿修改此欄位。它將根據其他欄位自動填入。

3. 密碼 – 透過產生身分驗證字符來提供此功能。複製權杖產生器產生的輸出，並將其貼到密碼欄位中。

> **Note**
>
> 您必須在用戶端連線中設定 SSL 模式。Aurora DSQL 支援 `PGSSLMODE=require`。Aurora DSQL 會在伺服器端強制執行 SSL 通訊，並會拒絕非 SSL 連線。

4. 您應該連接到叢集，並且可以開始執行 SQL 陳述式：

> **Important**
>
> DataGrip 為 PostgreSQL 資料庫 （例如工作階段） 提供的某些檢視因其唯一的架構而不適用於資料庫。可存取時，這些畫面不會提供有關連線到資料庫之實際工作階段的可靠資訊。

身分驗證憑證過期

建立的工作階段會保持身分驗證最多 1 小時，或直到發生明確中斷連線或用戶端逾時為止。如果需要建立新的連線，則必須在資料來源屬性的密碼欄位中產生並提供新的身分驗證字符。嘗試開啟新的工作階段 （例如，列出新的資料表或新的 SQL 主控台） 會強制新的身分驗證嘗試。如果連線設定中設定的身分驗證字符不再有效，則該新工作階段將會失敗，且所有先前開啟的工作階段都會失效。

搭配 Aurora DSQL 使用 PostgreSQL 通訊協定

PostgreSQL 使用訊息型通訊協定，在用戶端和伺服器之間進行通訊。透過 TCP/IP 和 Unix 網域通訊端支援通訊協定。下表顯示 Aurora DSQL 如何支援 PostgreSQL 通訊協定。

PostgreSQL	Aurora DSQL	備註
角色（也稱為使用者或群組）	資料庫角色	Aurora DSQL 會為您建立名為 的角色`admin`。建立自訂資料庫角色時，您必須使用該`admin`角色將它們與 IAM 角色建立關聯，以便在連線至叢集時進行驗證。如需詳細資訊，請參閱設定自訂資料庫角色。
主機（也稱為主機名稱或 hostspec)	叢集端點	Aurora DSQL 單一區域叢集提供單一受管端點，並在區域內無法使用時自動重新導向流量。
連線埠	不適用 – 使用預設值 `5432`	這是 PostgreSQL 預設值。
資料庫 (dbname)	使用 `postgres`	當您建立叢集時，Aurora DSQL 會為您建立此資料庫。
SSL 模式	SSL 一律啟用伺服器端	在 Aurora DSQL 中，Aurora DSQL 支援 `require` SSL 模式。沒有 SSL 的連線會遭到 Aurora DSQL 拒絕。
密碼	身分驗證字符	Aurora DSQL 需要暫時身分驗證字符，而不是長期密碼。如需詳細資訊，請參閱 在 Amazon Aurora DSQL 中產生身分驗證字符。

步驟 1：建立 Aurora DSQL 單一區域叢集

Aurora DSQL 的基本單位是 叢集，這是您存放資料的位置。在此任務中，您會在單一 中建立叢集 AWS 區域。

在 Aurora DSQL 中建立單一區域叢集

1. 登入 AWS Management Console ，並在 開啟 Aurora DSQL 主控台https://console.aws.amazon.com/dsql。
2. 選擇建立叢集，然後選擇單一區域。
3. （選用）在叢集設定中，選取下列任一選項：

- 選取自訂加密設定（進階）以選擇或建立 AWS KMS key。
- 選取啟用刪除保護，以防止刪除操作移除您的叢集。根據預設，會選取刪除保護。

4. （選用）在標籤中，選擇或輸入此叢集的標籤。
5. 選擇 建立叢集。

步驟 2：連線至您的 Aurora DSQL 叢集

當您根據叢集 ID 和區域建立 Aurora DSQL 叢集時，會自動產生叢集端點。命名格式為 *clusterid*.dsql.*region*.on.aws。用戶端使用端點來建立叢集的網路連線。

身分驗證是使用 IAM 進行管理，因此您不需要將登入資料存放在資料庫中。身分驗證字符是動態產生的唯一字元字串。字符僅用於身分驗證，建立後不會影響連線。在嘗試連線之前，請確定您的 IAM 身分具有 dsql:DbConnectAdmin許可，如中所述先決條件。

> **Note**
>
> 若要最佳化資料庫連線速度，請使用 PostgreSQL 第 17 版用戶端，並將 PGSSLNEGOTIATION設定為 Direct：PGSSLNEGOTIATION=direct。

使用身分驗證字符連線至您的叢集

1. 在 Aurora DSQL 主控台中，選擇您要連線的叢集。
2. 選擇連線。
3. 從端點（主機）複製端點。
4. 請確定已在身分驗證字符（密碼）區段中選擇您以管理員身分連線。
5. 複製產生的身分驗證字符。此字符的有效期為 15 分鐘。
6. 在作業系統命令列上，使用下列命令啟動psql並連線至您的叢集。*your_cluster_endpoint* 將 取代為您先前複製的叢集端點。

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

當系統提示您輸入密碼時，請輸入您先前複製的身分驗證字符。如果您嘗試使用過期的字符重新連線，連線請求會遭拒。如需詳細資訊，請參閱在 Amazon Aurora DSQL 中產生身分驗證字符。

7. 按 Enter。您應該會看到 PostgreSQL 提示。

```
postgres=>
```

如果您收到存取遭拒錯誤，請確定您的 IAM 身分具有 `dsql:DbConnectAdmin`許可。如果您擁有 許可並繼續取得存取拒絕錯誤，請參閱疑難排解 IAM 和如何使用 IAM 政策對存取遭拒或未經授權的操作錯誤進行故障診斷？。

步驟 3：在 Aurora DSQL 中執行範例 SQL 命令

執行 SQL 陳述式來測試您的 Aurora DSQL 叢集。下列範例陳述式需要名為 `department-insert-multirow.sql`和 的資料檔案`invoice.csv`，您可以從 GitHub 上的 aws-samples/aurora-dsql-samples 儲存庫下載。

在 Aurora DSQL 中執行範例 SQL 命令

1. 建立名為 的結構描述`example`。

```
CREATE SCHEMA example;
```

2. 建立使用自動產生的 UUID 做為主索引鍵的發票資料表。

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

3. 建立使用空白資料表的次要索引。

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

4. 建立部門資料表。

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

5. 使用 命令`psql \include`載入您從 GitHub 上的 aws-samples/aurora-dsql-samples 儲存庫下載`department-insert-multirow.sql`的名為 的檔案。將 *my-path* 取代為本機複本的路徑。

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

6. 使用 命令`psql \copy`載入您從 GitHub 上的 aws-samples/aurora-dsql-samples 儲存庫下載`invoice.csv`的名為 的檔案。將 *my-path* 取代為本機複本的路徑。

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

7. 查詢部門並依其總銷售額排序。

```
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;
```

下列範例輸出顯示 Department Three 的銷售額最多。

```
           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)
```

步驟 4：建立多區域叢集

當您建立多區域叢集時，您可以指定下列區域：

遠端區域

這是您在其中建立第二個叢集的區域。您在此區域中建立第二個叢集，並將其與初始叢集對等。Aurora DSQL 會將初始叢集上的所有寫入複寫到遠端叢集。您可以在任何叢集上讀取和寫入 。

見證區域

此區域會接收寫入多區域叢集的所有資料。不過，見證區域不會託管用戶端端點，也不會提供使用者資料存取。加密交易日誌的有限時段會保留在見證區域中。此日誌有助於復原，並在區域無法使用時支援交易仲裁。

下列範例示範如何建立初始叢集、在不同區域中建立第二個叢集，然後對等兩個叢集以建立多區域叢集。它還示範了跨區域寫入複寫以及來自兩個區域端點的一致讀取。

建立多區域叢集

1. 登入 AWS Management Console ，並在 開啟 Aurora DSQL 主控台https://console.aws.amazon.com/dsql。
2. 在導覽窗格中，選擇叢集。
3. 選擇建立叢集，然後選擇多區域。
4. （選用）在叢集設定中，為您的初始叢集選取下列任一選項：
   - 選取自訂加密設定（進階）以選擇或建立 AWS KMS key。
   - 選取啟用刪除保護，以防止刪除操作移除您的叢集。根據預設，會選取刪除保護。
5. 在多區域設定中，為您的初始叢集選擇下列選項：
   - 在見證區域中，選擇區域。目前，多區域叢集中的見證區域僅支援美國區域。
   - （選用）在遠端區域叢集 ARN 中，輸入另一個區域中現有叢集的 ARN。如果沒有叢集可做為多區域叢集中的第二個叢集，請在建立初始叢集後完成設定。
6. （選用）選擇初始叢集的標籤。
7. 選擇建立叢集以建立您的初始叢集。如果您在上一個步驟中未輸入 ARN，主控台會顯示叢集設定待定通知。
8. 在叢集設定等待通知中，選擇完成多區域叢集設定。此動作會啟動在另一個區域中建立第二個叢集。
9. 為您的第二個叢集選擇下列其中一個選項：

- 新增遠端區域叢集 ARN – 如果叢集存在，且您希望它成為多區域叢集中的第二個叢集，請選擇此選項。
- 在另一個區域中建立叢集 – 選擇此選項以建立第二個叢集。在遠端區域中，選擇第二個叢集的區域。

10. 選擇***your-second-region***建立叢集，其中*your-second-region*是第二個叢集的位置。主控台會在您的第二個區域中開啟。
11. （選用）選擇第二個叢集的叢集設定。例如，您可以選擇 AWS KMS key。
12. 選擇建立叢集以建立第二個叢集。
13. 在 ***initial-cluster-region*** 中選擇對等，其中 *initial-cluster-region* 是託管您建立的第一個叢集的區域。
14. 出現提示時，選擇確認。此步驟會完成多區域叢集的建立。

連線至您的第二個叢集

1. 開啟 Aurora DSQL 主控台，並選擇第二個叢集的區域。
2. 選擇 Clusters (叢集)。
3. 選取多區域叢集中第二個叢集的資料列。
4. 在動作中，選擇在 CloudShell 中開啟。
5. 選擇以管理員身分連線。
6. 選擇啟動 CloudShell。
7. 選擇執行。
8. 依照中的步驟建立範例結構描述步驟 3：在 Aurora DSQL 中執行範例 SQL 命令。

   交易範例

   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. 使用 `psql`的`copy`和 `include`命令載入範例資料。如需詳細資訊，請參閱步驟 3：在 Aurora DSQL 中執行範例 SQL 命令。

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

從託管初始叢集的 區域查詢第二個叢集中的資料

1. 在 Aurora DSQL 主控台中，選擇初始叢集的區域。
2. 選擇 Clusters (叢集)。
3. 選取多區域叢集中第二個叢集的資料列。
4. 在動作中，選擇在 CloudShell 中開啟。
5. 選擇 Connect as admin。
6. 選擇啟動 CloudShell。
7. 選擇執行。
8. 查詢您插入第二個叢集的資料。

   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;
```

Aurora DSQL 的身分驗證和授權

Aurora DSQL 使用 IAM 角色和政策進行叢集授權。您可以將 IAM 角色與 PostgreSQL 資料庫角色建立關聯，以進行資料庫授權。此方法結合了 IAM 的優勢與 PostgreSQL 權限。Aurora DSQL 使用這些功能為您的叢集、資料庫和資料提供全面的授權和存取政策。

使用 IAM 管理您的叢集

若要管理您的叢集，請使用 IAM 進行身分驗證和授權：

IAM 身分驗證

若要在管理 Aurora DSQL 叢集時驗證 IAM 身分，您必須使用 IAM。您可以使用 AWS Management Console、 AWS CLI或 AWS SDK 提供身分驗證。

IAM 授權

若要管理 Aurora DSQL 叢集，請使用 Aurora DSQL 的 IAM 動作授予授權。例如，若要建立叢集，請確定您的 IAM 身分具有 IAM 動作 的許可`dsql:CreateCluster`，如下列範例政策動作所示。

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

如需詳細資訊，請參閱使用 IAM 政策動作來管理叢集。

使用 IAM 連線至您的叢集

若要連線至叢集，請使用 IAM 進行身分驗證和授權：

IAM 身分驗證

使用具有連接到叢集之授權的 IAM 身分產生臨時身分驗證字符。如需詳細資訊，請參閱 在 Amazon Aurora DSQL 中產生身分驗證字符。

IAM 授權

將下列 IAM 政策動作授予您用來建立叢集端點連線的 IAM 身分：

- `dsql:DbConnectAdmin` 如果您使用的是 admin角色，請使用 。Aurora DSQL 會為您建立和管理此角色。下列範例 IAM 政策動作允許 admin 連線到 *my-cluster*。

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

- `dsql:DbConnect` 如果您使用的是自訂資料庫角色，請使用 。您可以在資料庫中使用 SQL 命令來建立和管理此角色。下列範例 IAM 政策動作允許自訂資料庫角色連接至 *my-cluster* 長達一小時。

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

建立連線後，您的角色會獲得最多一小時的連線授權。

使用 PostgreSQL 資料庫角色和 IAM 角色與資料庫互動

PostgreSQL 使用 角色的概念來管理資料庫存取許可。根據角色的設定方式，可以將角色視為資料庫使用者或資料庫使用者群組。您可以使用 SQL 命令建立 PostgreSQL 角色。若要管理資料庫層級授權，請將 PostgreSQL 許可授予 PostgreSQL 資料庫角色。

Aurora DSQL 支援兩種類型的資料庫角色： admin角色和自訂角色。Aurora DSQL 會在 Aurora DSQL 叢集中自動為您建立預先定義admin的角色。您無法修改admin角色。當您以 身分連線至資料庫時admin，您可以發出 SQL 來建立新的資料庫層級角色，以與您的 IAM 角色建立關聯。若要讓 IAM 角色連線至您的資料庫，請將您的自訂資料庫角色與您的 IAM 角色建立關聯。

身分驗證

使用 admin角色連線到您的叢集。連接資料庫後，請使用 命令AWS IAM GRANT將自訂資料庫角色與授權連線至叢集的 IAM 身分建立關聯，如下列範例所示。

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

如需詳細資訊，請參閱 授權資料庫角色連線至您的叢集。

授權

使用 admin角色連線到您的叢集。執行 SQL 命令來設定自訂資料庫角色並授予許可。若要進一步了解，請參閱 PostgreSQL 文件中的 PostgreSQL 資料庫角色和 PostgreSQL 權限 PostgreSQL。PostgreSQL

搭配 Aurora DSQL 使用 IAM 政策動作

您使用的 IAM 政策動作取決於您用來連線至叢集的角色： admin或自訂資料庫角色。此政策也取決於此角色所需的 IAM 動作。

使用 IAM 政策動作連線至叢集

當您使用預設資料庫角色 連線到叢集時admin，請使用具有 授權的 IAM 身分來執行下列 IAM 政策動作。

```
"dsql:DbConnectAdmin"
```

當您使用自訂資料庫角色連線至叢集時，請先將 IAM 角色與資料庫角色建立關聯。您用來連線至叢集的 IAM 身分必須具有執行下列 IAM 政策動作的授權。

```
"dsql:DbConnect"
```

若要進一步了解自訂資料庫角色，請參閱 使用資料庫角色和 IAM 身分驗證。

使用 IAM 政策動作來管理叢集

管理 Aurora DSQL 叢集時，請僅為您的角色需要執行的動作指定政策動作。例如，如果您的角色只需要取得叢集資訊，您可以將角色許可限制為僅 GetCluster和 ListClusters許可，如下列範例政策所示

JSON

```
{
"Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
```

```
        "dsql:GetCluster",
        "dsql:ListClusters"
      ],
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/my-cluster"
    }
  ]
}
```

下列範例政策顯示用於管理叢集的所有可用 IAM 政策動作。

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" : "*"
    }
  ]
}
```

使用 IAM 和 PostgreSQL 撤銷授權

您可以撤銷 IAM 角色存取資料庫層級角色的許可：

撤銷連接至叢集的管理員授權

若要撤銷使用 admin角色連線至叢集的授權，請撤銷 IAM 身分對 的存取權dsql:DbConnectAdmin。編輯 IAM 政策或從身分分離政策。

從 IAM 身分撤銷連線授權後，Aurora DSQL 會拒絕該 IAM 身分的所有新連線嘗試。任何使用 IAM 身分的作用中連線，在連線期間都可能保持授權狀態。如需連線持續時間的詳細資訊，請參閱配額和限制。

撤銷自訂角色授權以連線至叢集

若要撤銷對 以外資料庫角色的存取權`admin`，請撤銷 IAM 身分對 的存取權`dsql:DbConnect`。編輯 IAM 政策或從身分分離政策。

您也可以使用資料庫中的 命令，移除資料庫角色與 IAM 之間的關聯`AWS IAM REVOKE`。若要進一步了解如何從資料庫角色撤銷存取權，請參閱 從 IAM 角色撤銷資料庫授權。

您無法管理預先定義`admin`資料庫角色的許可。若要了解如何管理自訂資料庫角色的許可，請參閱 PostgreSQL 權限。在 Aurora DSQL 成功遞交修改交易之後，權限的修改會在下一個交易上生效。

在 Amazon Aurora DSQL 中產生身分驗證字符

若要使用 SQL 用戶端連線至 Amazon Aurora DSQL，請產生身分驗證字符以用作密碼。此字符僅用於驗證連線。建立連線後，即使身分驗證字符過期，連線仍然有效。

如果您使用 AWS 主控台建立身分驗證字符，字符預設會在一小時內自動過期。如果您使用 AWS CLI 或 SDKs來建立權杖，則預設值為 15 分鐘。持續時間上限為 604，800 秒，也就是一週。若要從用戶端再次連線至 Aurora DSQL，您可以在尚未過期時使用相同的身分驗證字符，也可以產生新的字符。

若要開始使用產生權杖，請在 Aurora DSQL 中建立 IAM 政策和叢集。然後使用 AWS 主控台 AWS CLI、 或 AWS SDKs來產生字符。

視您用來連線的資料庫角色而定使用 IAM 連線至您的叢集，您至少必須擁有 中列出的 IAM 許可。

主題

- 使用 AWS 主控台在 Aurora DSQL 中產生身分驗證字符
- 使用 在 Aurora DSQL 中 AWS CloudShell 產生身分驗證字符
- 使用 AWS CLI 在 Aurora DSQL 中產生身分驗證字符
- 使用 SDKs 在 Aurora DSQL 中產生權杖

使用 AWS 主控台在 Aurora DSQL 中產生身分驗證字符

Aurora DSQL 會使用字符而非密碼來驗證使用者。您可以從 主控台產生字符。

產生身分驗證字符

1. 登入 AWS Management Console ，並在 開啟 Aurora DSQL 主控台https://console.aws.amazon.com/dsql。
2. 選擇您要為其產生身分驗證字符之叢集的叢集 ID。如果您尚未建立叢集，請遵循 步驟 1：建立 Aurora DSQL 單一區域叢集或 中的步驟步驟 4：建立多區域叢集。
3. 選擇連線，然後選取取得權杖。
4. 選擇您要以 admin或 身分連接自訂資料庫角色。
5. 複製產生的身分驗證字符並將其用於 使用 SQL 用戶端存取 Aurora DSQL。

若要進一步了解 Aurora DSQL 中的自訂資料庫角色和 IAM，請參閱 身分驗證和授權。

使用 在 Aurora DSQL 中 AWS CloudShell 產生身分驗證字符

在使用 產生身分驗證字符之前 AWS CloudShell，請務必執行下列動作：

- 建立 Aurora DSQL 叢集。
- 新增執行 Amazon S3 操作的許可get-object，以從組織 AWS 帳戶 外部的 擷取物件。如需詳細資訊，請參閱 Amazon S3 使用者指南。

使用 產生身分驗證字符 AWS CloudShell

1. 登入 AWS Management Console ，並在 開啟 Aurora DSQL 主控台https://console.aws.amazon.com/dsql。
2. 在 AWS 主控台的左下角，選擇 AWS CloudShell。
3. 依照安裝或更新 的最新版本來 AWS CLI安裝 AWS CLI。

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

4. 執行下列命令來產生admin角色的身分驗證字符。將 *us-east-1* 取代為您的區域，並將 *your_cluster_endpoint* 取代為您叢集的端點。

> **Note**
>
> 如果您未以 身分連線admin，請generate-db-connect-auth-token改用 。

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

如果您遇到問題，請參閱疑難排解 IAM 和如何使用 IAM 政策對存取遭拒或未經授權的操作錯誤進行疑難排解？。

5. 使用以下命令來使用 psql 啟動與叢集的連線。

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

6. 您應該會看到提示以提供密碼。複製您產生的字符，並確保不包含任何額外的空格或字元。從 將它貼到下列提示中psql。

```
Password for user admin:
```

7. 按 Enter。您應該會看到 PostgreSQL 提示。

```
postgres=>
```

如果您收到存取遭拒錯誤，請確定您的 IAM 身分具有 dsql:DbConnectAdmin許可。如果您擁有 許可並繼續取得存取拒絕錯誤，請參閱疑難排解 IAM 和如何使用 IAM 政策對存取遭拒或未經授權的操作錯誤進行故障診斷？。

若要進一步了解 Aurora DSQL 中的自訂資料庫角色和 IAM，請參閱 身分驗證和授權。

使用 AWS CLI 在 Aurora DSQL 中產生身分驗證字符

當您的叢集為 時ACTIVE，您可以使用 aws dsql命令在 CLI 上產生身分驗證字符。使用下列任一技術：

- 如果您要與 admin角色連線，請使用 generate-db-connect-admin-auth-token選項。
- 如果您要與自訂資料庫角色連線，請使用 generate-db-connect-auth-token選項。

下列範例使用下列屬性來產生admin角色的身分驗證字符。

- *your_cluster_endpoint* – 叢集的端點。它遵循格式 *your_cluster_identifier*.dsql.*region*.on.aws，如範例 所示01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws。
- *region* – AWS 區域，例如 us-east-2或 us-east-1。

下列範例設定權杖在 3600 秒 (1 小時） 後過期的過期時間。

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
```

使用 SDKs 在 Aurora DSQL 中產生權杖

您可以在叢集處於 ACTIVE 狀態時產生身分驗證字符。SDK 範例使用以下屬性來產生admin角色的身分驗證字符：

- *your_cluster_endpoint* （或 *yourClusterEndpoint*) – Aurora DSQL 叢集的端點。命名格式為 *your_cluster_identifier*.dsql.*region*.on.aws，如範例 所示01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws。
- *region* （或 *RegionEndpoint*) – AWS 區域 叢集所在的 ，例如 us-east-2或 us-east-1。

Python SDK

您可以透過下列方式產生字符：

- 如果您要與 admin角色連線，請使用 generate_db_connect_admin_auth_token。

- 如果您要與自訂資料庫角色連線，請使用 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

您可以透過下列方式產生字符：

- 如果您要與 admin角色連線，請使用 GenerateDBConnectAdminAuthToken。
- 如果您要與自訂資料庫角色連線，請使用 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

您可以透過下列方式產生字符：

- 如果您要與 admin角色連線，請使用 getDbConnectAdminAuthToken。
- 如果您要與自訂資料庫角色連線，請使用 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

您可以透過下列方式產生字符：

- 如果您要與 admin角色連線，請使用 generateDbConnectAdminAuthToken。
- 如果您要與自訂資料庫角色連線，請使用 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

您可以透過下列方式產生字符：

- 如果您要與 admin角色連線，請使用 db_connect_admin_auth_token。
- 如果您要與自訂資料庫角色連線，請使用 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

您可以透過下列方式產生字符：

- 如果您要與 admin角色連線，請使用 generate_db_connect_admin_auth_token。
- 如果您要與自訂資料庫角色連線，請使用 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**
>
> 適用於 .NET 的官方 SDK 不包含內建 API 呼叫，可產生 Aurora DSQL 的身分驗證字符。反之，您必須使用 DSQLAuthTokenGenerator，這是公用程式類別。下列程式碼範例示範如何產生 .NET 的身分驗證字符。

您可以透過下列方式產生字符：

- 如果您要與 admin角色連線，請使用 DbConnectAdmin。
- 如果您要與自訂資料庫角色連線，請使用 DbConnect。

下列範例使用 DSQLAuthTokenGenerator 公用程式類別，為具有 admin角色的使用者產生身分驗證字符。將 *insert-dsql-cluster-endpoint* 取代為您的叢集端點。

```
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**
>
> Golang SDK 不提供產生預先簽署字符的內建方法。您必須手動建構已簽署的請求，如下列程式碼範例所示。

在下列程式碼範例中，`action`根據 PostgreSQL 使用者指定 ：

- 如果您要與 `admin`角色連線，請使用 `DbConnectAdmin`動作。
- 如果您要與自訂資料庫角色連線，請使用 `DbConnect`動作。

除了*`yourClusterEndpoint`* 和*`##`*之外，下列範例使用 *`##`*。根據 PostgreSQL 使用者指定*`##`*。

```
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
}
```

使用資料庫角色和 IAM 身分驗證

Aurora DSQL 支援使用 IAM 角色和 IAM 使用者進行身分驗證。您可以使用任一種方法來驗證和存取 Aurora DSQL 資料庫。

IAM 角色

IAM 角色是 中的身分 AWS 帳戶 ，具有特定許可，但未與特定人員建立關聯。使用 IAM 角色可提供臨時安全登入資料。您可以透過多種方式暫時擔任 IAM 角色：

- 透過在 中切換角色 AWS Management Console
- 透過呼叫 AWS CLI 或 AWS API 操作
- 使用自訂 URL

擔任角色之後，您可以使用角色的暫時登入資料來存取 Aurora DSQL。如需使用角色方法的詳細資訊，請參閱《IAM 使用者指南》中的 IAM 身分。

IAM 使用者

IAM 使用者是 中的身分 AWS 帳戶 ，具有特定許可，並與單一人員或應用程式相關聯。IAM 使用者具有長期登入資料，例如可用於存取 Aurora DSQL 的密碼和存取金鑰。

> **Note**
>
> 若要使用 IAM 身分驗證執行 SQL 命令，您可以在以下範例中使用 IAM 角色 ARNs 或 IAM 使用者 ARNs。

授權資料庫角色連線至您的叢集

建立 IAM 角色，並使用 IAM 政策動作授予連線授權：`dsql:DbConnect`。

IAM 政策也必須授予存取叢集資源的許可。使用萬用字元 (*) 或遵循搭配使用 IAM 條件金鑰與 Amazon Aurora DSQL 中的指示。

授權資料庫角色在您的資料庫中使用 SQL

您必須使用具有 授權的 IAM 角色來連線至叢集。

1. 使用 SQL 公用程式連線至 Aurora DSQL 叢集。

   使用 admin 資料庫角色搭配 IAM 身分，該身分已獲授權讓 IAM 動作dsql:DbConnectAdmin連線至您的叢集。

2. 建立新的資料庫角色，請務必指定 WITH LOGIN選項。

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

3. 將資料庫角色與 IAM 角色 ARN 建立關聯。

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

4. 將資料庫層級許可授予資料庫角色

   下列範例使用 GRANT命令在資料庫中提供授權。

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

如需詳細資訊，請參閱 PostgreSQL GRANT 文件中的 PostgreSQL 和 PostgreSQL PostgreSQL 權限。

從 IAM 角色撤銷資料庫授權

若要撤銷資料庫授權，請使用 AWS IAM REVOKE操作。

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

若要進一步了解撤銷授權，請參閱 使用 IAM 和 PostgreSQL 撤銷授權。

Aurora DSQL 和 PostgreSQL

Aurora DSQL 是 PostgreSQL 相容的分散式關聯式資料庫，專為交易工作負載而設計。Aurora DSQL 使用核心 PostgreSQL 元件，例如剖析器、規劃器、最佳化器和類型系統。

Aurora DSQL 設計確保所有支援的 PostgreSQL 語法提供相容的行為，並產生相同的查詢結果。例如，Aurora DSQL 提供類型轉換、算術運算，以及與 PostgreSQL 相同的數值精確度和比例。會記錄任何偏差。

Aurora DSQL 也引進進階功能，例如樂觀並行控制和分散式結構描述管理。透過這些功能，您可以使用熟悉的 PostgreSQL 工具，同時受益於現代、雲端原生、分散式應用程式所需的效能和可擴展性。

PostgreSQL 相容性重點

Aurora DSQL 目前是以 PostgreSQL 第 16 版為基礎。金鑰相容性包括下列項目：

線路通訊協定

Aurora DSQL 使用標準 PostgreSQL v3 線路通訊協定。這可讓 與標準 PostgreSQL 用戶端、驅動程式和工具整合。例如，Aurora DSQL 與 `psql`、 `pgjdbc`和 相容`psycopg`。

SQL 相容性

Aurora DSQL 支援交易工作負載中常用的各種標準 PostgreSQL 表達式和函數。支援的 SQL 表達式會產生與 PostgreSQL 相同的結果，包括下列項目：

- 處理 null
- 排序順序行為
- 數值操作的規模和精確度
- 字串操作的等效性

如需詳細資訊，請參閱Aurora DSQL 中的 SQL 功能相容性。

交易管理

Aurora DSQL 會保留 PostgreSQL 的主要特性，例如 ACID 交易和相當於 PostgreSQL 可重複讀取的隔離層級。如需詳細資訊，請參閱Aurora DSQL 中的並行控制。

關鍵架構差異

Aurora DSQL 的分散式共用物件設計會產生與傳統 PostgreSQL 的一些基本差異。這些差異是 Aurora DSQL 架構不可或缺的，可提供許多效能和可擴展性優勢。主要差異包括下列項目：

樂觀並行控制 (OCC)

Aurora DSQL 使用樂觀並行控制模型。這種無鎖定方法可防止交易彼此封鎖、消除死結，並啟用高輸送量平行執行。這些功能使得 Aurora DSQL 對於需要大規模一致效能的應用程式特別有價值。如需更多範例，請參閱 Aurora DSQL 中的並行控制。

非同步 DDL 操作

Aurora DSQL 會以非同步方式執行 DDL 操作，在結構描述變更期間允許不間斷的讀取和寫入。其分散式架構可讓 Aurora DSQL 執行下列動作：

- 將 DDL 操作作為背景任務執行，將中斷降至最低。
- 協調目錄會隨著高度一致的分散式交易而變更。這可確保所有節點的原子可見性，即使在故障或並行操作期間也是如此。
- 使用解耦運算和儲存層，在多個可用區域中以完全分散式、無領導的方式操作。

如需詳細資訊，請參閱Aurora DSQL 中的 DDL 和分散式交易。

Aurora DSQL 中的 SQL 功能相容性

Aurora DSQL 和 PostgreSQL 會傳回所有 SQL 查詢的相同結果。請注意，Aurora DSQL 與沒有 `ORDER BY`子句的 PostgreSQL 不同。在下列各節中，了解 Aurora DSQL 對 PostgreSQL 資料類型和 SQL 命令的支援。

主題

- Aurora DSQL 中支援的資料類型
- 支援的 SQL for Aurora DSQL
- Aurora DSQL 中支援的 SQL 命令子集
- Aurora DSQL 中不支援的 PostgreSQL 功能

Aurora DSQL 中支援的資料類型

Aurora DSQL 支援一部分常見的 PostgreSQL 類型。

主題

- 數值資料類型
- 字元資料類型
- 日期和時間資料類型
- 其他資料類型
- 查詢執行時間資料類型

數值資料類型

Aurora DSQL 支援下列 PostgreSQL 數值資料類型。

名稱	Aliases	範圍和精確度	儲存體大小	索引支援
smallint	int2	-32768 到 +32767	2 位元組	是
integer	int, int4	-2147483648 到 +2147483647	4 位元組	是
bigint	int8	-9223372036854775808 至 +9223372036854775807	8 位元組	是
real	float4	6 小數位精確度	4 位元組	是
double precision	float8	15 小數位精確度	8 位元組	是
numeric 【(*p*、*s*)】	decimal 【(*p*、*s*)】 dec【(*p* , *s*)】	可選取精確度的確切數值。精確度上限為 38，比例上限為 37.[1]。預設值為 numeric(18,6)。	8 個位元組 + 每個精確度位數 2 個位元組。大小上限為 27 個位元組。	否

1 – 如果您在執行 CREATE TABLE或 時未明確指定大小ALTER TABLE ADD COLUMN，Aurora DSQL 會強制執行預設值。當您執行 INSERT或 UPDATE陳述式時，Aurora DSQL 會套用限制。

字元資料類型

Aurora DSQL 支援下列 PostgreSQL 字元資料類型。

名稱	Aliases	描述	Aurora DSQL 限制	儲存體大小	索引支援
`character【(n)】`	`char【(n)】`	固定長度的字元字串	4096 位元組[1]	最多 4100 個位元組的變數	是
`character varying【(n)】`	`varchar【(n)】`	可變長度字元字串	65535 位元組[1]	最多 65539 個位元組的變數	是
`bpchar【(n)】`		如果是固定長度，這是 的別名`char`。如果是可變長度，這是 的別名`varchar`，其中尾端空格在語義上不重要。	4096 位元組[1]	最多 4100 個位元組的變數	是
`text`		可變長度字元字串	1 MiB[1]	最多 1 MiB 的變數	是

[1] – 如果您在執行 `CREATE TABLE`或 時未明確指定大小`ALTER TABLE ADD COLUMN`，則 Aurora DSQL 會強制執行預設值。當您執行 `INSERT`或 `UPDATE`陳述式時，Aurora DSQL 會套用限制。

日期和時間資料類型

Aurora DSQL 支援下列 PostgreSQL 日期和時間資料類型。

名稱	Aliase	描述	範圍	Resolution	儲存體大小	索引支援
date		日曆日期 (年、月、日)	4713 BC – 5874897 AD	1 天	4 位元組	是
time 【(*p*)】 【without time zone】	time	一天中的時間，沒有時區	0 – 1	1 毫秒	8 位元組	是
time 【(*p*)】 with time zone	timet	一天中的時間，包括時區	00：00：00+1559 – 24：00：00 –1559	1 毫秒	12 個位元組	否
timestamp 【(*p*)】 【without time zone】		沒有時區的日期和時間	4713 BC – 294276 AD	1 毫秒	8 位元組	是
timestamp 【(*p*)】 with time zone	timestz	日期和時間，包括時區	4713 BC – 294276 AD	1 毫秒	8 位元組	是
interval 【fields】 【(*p*)】		時間範圍	-178000000 年 – 178000000 年	1 毫秒	16 個位元組	否

其他資料類型

Aurora DSQL 支援下列其他 PostgreSQL 資料類型。

名稱	Aliases	描述	Aurora DSQL 限制	儲存體大小	索引支援
boolean	bool	邏輯布林值 (true/false)		1 位元組	是
bytea		二進位資料 (「位元組陣列」)	1 MiB[1]	變數上限為 1 MiB	否
UUID		全域唯一識別符		16 個位元組	是

[1] – 如果您在執行 CREATE TABLE或 時未明確指定大小ALTER TABLE ADD COLUMN，則 Aurora DSQL 會強制執行預設值。當您執行 INSERT或 UPDATE陳述式時，Aurora DSQL 會套用限制。

查詢執行時間資料類型

查詢執行時間資料類型是在查詢執行時間使用的內部資料類型。這些類型與您在結構描述中定義的 PostgreSQL 相容類型不同integer，例如 varchar和 。反之，這些類型是 Aurora DSQL 在處理查詢時使用的執行時間表示法。

僅在查詢執行時間支援下列資料類型：

陣列類型

Aurora DSQL 支援支援的資料類型陣列。例如，您可以有整數陣列。該函數會將字串string_to_array分割為具有逗號分隔符號 (,) 的 PostgreSQL 樣式陣列，如下列範例所示。在查詢執行期間，您可以在表達式、函數輸出或暫時運算中使用陣列。

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

函數會傳回類似以下的回應：

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

inet 類型

資料類型代表 IPv4, IPv6 主機地址及其子網路。此類型在剖析日誌、篩選 IP 子網路或在查詢中進行網路計算時非常有用。如需詳細資訊，請參閱 PostgreSQL 文件中的 inet。

支援的 SQL for Aurora DSQL

Aurora DSQL 支援各種核心 PostgreSQL SQL 功能。在下列各節中，您可以了解一般 PostgreSQL 表達式支援。此清單並不詳盡。

SELECT 命令

Aurora DSQL 支援 SELECT命令的下列子句。

主要子句	支援的子句
FROM	
GROUP BY	ALL, DISTINCT
ORDER BY	ASC, DESC, NULLS
LIMIT	
DISTINCT	
HAVING	
USING	
WITH（常見資料表表達式）	
INNER JOIN	ON
OUTER JOIN	LEFT, RIGHT, FULL, ON
CROSS JOIN	ON
UNION	ALL
INTERSECT	ALL

主要子句	支援的子句
EXCEPT	ALL
OVER	RANK (), PARTITION BY
FOR UPDATE	

資料定義語言 (DDL)

Aurora DSQL 支援下列 PostgreSQL DDL 命令。

Command	主要子句	支援的子句
CREATE	TABLE	如需 CREATE TABLE命令支援語法的相關資訊，請參閱 CREATE TABLE。
ALTER	TABLE	如需 ALTER TABLE命令支援語法的相關資訊，請參閱 ALTER TABLE。
DROP	TABLE	
CREATE	[UNIQUE] INDEX ASYNC	您可以搭配下列參數使用此命令：ON、NULLS FIRST、NULLS LAST。 如需 CREATE INDEX ASYNC命令支援語法的相關資訊，請參閱 Aurora DSQL 中的非同步索引。
DROP	INDEX	
CREATE	VIEW	如需 CREATE VIEW命令支援語法的詳細資訊，請參閱 CREATE VIEW。
ALTER	VIEW	如需 ALTER VIEW命令支援語法的相關資訊，請參閱 ALTER VIEW。
DROP	VIEW	如需 DROP VIEW命令支援語法的相關資訊，請參閱 DROP VIEW。

Command	主要子句	支援的子句
CREATE	ROLE, WITH	
CREATE	FUNCTION	LANGUAGE SQL
CREATE	DOMAIN	

資料處理語言 (DML)

Aurora DSQL 支援下列 PostgreSQL DML 命令。

Command	主要子句	支援的子句
INSERT	INTO	VALUES SELECT
UPDATE	SET	WHERE (SELECT) FROM, WITH
DELETE	FROM	USING, WHERE

資料控制語言 (DCL)

Aurora DSQL 支援下列 PostgreSQL DCL 命令。

Command	支援的子句
GRANT	ON, TO
REVOKE	ON, FROM, CASCADE, RESTRICT

交易控制語言 (TCL)

Aurora DSQL 支援下列 PostgreSQL TCL 命令。

Command	支援的子句
COMMIT	
BEGIN	[WORK | TRANSACTION] [READ ONLY | READ WRITE]

公用程式命令

Aurora DSQL 支援下列 PostgreSQL 公用程式命令：

- EXPLAIN
- ANALYZE （僅限關係名稱）

Aurora DSQL 中支援的 SQL 命令子集

此 PostgreSQL 區段提供有關支援表達式的詳細資訊，著重於具有大量參數集和子命令的命令。例如，PostgreSQL 中的 CREATE TABLE 提供許多子句和參數。本節說明 Aurora DSQL 針對這些命令支援的所有 PostgreSQL 語法元素。

主題

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

CREATE TABLE

CREATE TABLE 會定義新的資料表。

```
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 TABLE 會變更資料表的定義。

```
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 VIEW 會定義新的持久性檢視。Aurora DSQL 不支援暫時檢視；僅支援永久檢視。

支援的語法

```
CREATE [ OR REPLACE ] [ RECURSIVE ] VIEW name [ ( column_name [, ...] ) ]
    [ WITH ( view_option_name [= view_option_value] [, ... ] ) ]
    AS query
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

描述

CREATE VIEW 定義查詢的檢視。檢視不會實際具體化。相反地，每次在查詢中參考檢視時都會執行查詢。

CREATE or REPLACE VIEW 類似，但如果已存在相同名稱的檢視，則會予以取代。新查詢必須產生由現有檢視查詢產生的相同資料欄（亦即，相同資料欄名稱的順序和資料類型相同），但可能會將其他資料欄新增至清單結尾。產生輸出資料欄的計算可能不同。

如果指定結構描述名稱，例如 CREATE VIEW myschema.myview ...)，則會在指定的結構描述中建立檢視。否則，它會在目前的結構描述中建立。

檢視的名稱必須與相同結構描述中任何其他關係（資料表、索引、檢視）的名稱不同。

參數

CREATE VIEW 支援各種參數，以控制自動更新檢視的行為。

RECURSIVE

建立遞迴檢視。語法： CREATE RECURSIVE VIEW [schema .] view_name (column_names) AS SELECT ...; 等同於 CREATE VIEW [schema .] view_name AS WITH RECURSIVE view_name (column_names) AS (SELECT ...) SELECT column_names FROM view_name;。

必須為遞迴檢視指定檢視欄名稱清單。

name

要建立的檢視名稱，可能選擇性符合結構描述資格。必須為遞迴檢視指定資料欄名稱清單。

column_name

用於檢視資料欄的選用名稱清單。如果未指定，則會從查詢中刪除資料欄名稱。

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

此子句指定檢視的選用參數；支援下列參數。

- check_option (enum) — 此參數可以是 local或 cascaded，且等同於指定 WITH [CASCADED | LOCAL] CHECK OPTION。
- security_barrier (boolean)- 如果檢視旨在提供資料列層級安全性，則應使用此檢視。Aurora DSQL 目前不支援資料列層級安全性，但此選項仍會強制首先評估檢視WHERE的條件（以及任何使用標記為 之運算子的條件LEAKPROOF)。
- security_invoker (boolean)- 此選項會導致根據檢視使用者的權限檢查基礎基礎關係，而不是檢視擁有者。如需完整詳細資訊，請參閱以下備註。

您可以使用 在現有檢視上變更上述所有選項ALTER VIEW。

query

提供檢視資料欄和資料列的 SELECT或VALUES 命令。

- WITH [CASCADED | LOCAL] CHECK OPTION— 此選項可控制自動更新檢視的行為。指定此選項時，將檢查檢視上的INSERTUPDATE命令，以確保新資料列符合檢視定義條件（也就是說，會檢查新資料列，以確保它們可透過檢視顯示）。如果不是，則會拒絕更新。如果CHECK OPTION未指定 ，則允許檢視上的 UPDATE INSERT命令建立無法透過檢視看見的資料列。支援下列檢查選項。
- LOCAL- 新資料列只會針對檢視本身直接定義的條件進行檢查。不會檢查在基礎基礎基本檢視上定義的任何條件（除非它們也指定 CHECK OPTION)。
- CASCADED- 根據檢視和所有基礎基礎檢視的條件檢查新資料列。如果CHECK OPTION已指定 ，而且未指定 LOCAL 或 CASCADED ，則會假設 CASCADED 。

> **Note**
>
> CHECK OPTION 可能無法與RECURSIVE檢視搭配使用。CHECK OPTION 僅支援自動更新的檢視。

備註

使用 DROP VIEW陳述式捨棄檢視。

應仔細考慮檢視資料欄的名稱和資料類型。例如，CREATE VIEW vista AS SELECT 'Hello World'；不建議使用，因為資料欄名稱預設為 ?column?;。此外，資料欄資料類型預設為 text，這可能不是您想要的。

更好的方法是明確指定資料欄名稱和資料類型，例如：CREATE VIEW vista AS SELECT text 'Hello World' AS hello;。

根據預設，檢視中參考的基礎關係存取權取決於檢視擁有者的許可。在某些情況下，這可用來提供對基礎資料表的安全但限制存取。不過，並非所有檢視都安全，不會遭到竄改。

- 如果檢視的 security_invoker 屬性設為 true，則基礎基礎基礎關係的存取權取決於執行查詢的使用者許可，而不是檢視擁有者。因此，安全調用者檢視的使用者必須具有檢視及其基礎關係的相關許可。
- 如果任何基礎基礎關係是安全調用者檢視，它將被視為直接從原始查詢存取。因此，安全調用者檢視一律會使用目前使用者的許可檢查其基礎關係，即使它是從沒有 security_invoker 屬性的檢視存取。
- 檢視中呼叫的函數會被視為與使用檢視直接從查詢呼叫的函數相同。因此，檢視的使用者必須具有呼叫檢視使用之所有函數的許可。檢視中的函數會以執行查詢的使用者或函數擁有者的權限執行，視函數是否定義為 SECURITY INVOKER或 而定SECURITY DEFINER。例如，在檢視中CURRENT_USER直接呼叫 一律會傳回叫用使用者，而不是檢視擁有者。這不受檢視security_invoker的設定影響，因此將 security_invoker設為 false 的檢視不等同於SECURITY DEFINER函數。
- 建立或取代檢視的使用者必須具有檢視查詢中提及的任何結構描述USAGE的權限，才能在這些結構描述中查詢參考的物件。不過請注意，此查詢只會在建立或取代檢視時發生。因此，檢視的使用者只需要包含檢視的結構描述上的權限，而不需要檢視查詢中參考的結構描述上的USAGE權限，即使是安全呼叫者檢視也一樣。
- 在現有檢視上使用 CREATE OR REPLACE VIEW 時，只會CHECK OPTION變更檢視的定義SELECT規則，以及任何WITH (...)參數及其參數。其他檢視屬性，包括擁有權、許可和非 SELECT 規則，保持不變。您必須擁有檢視才能取代它（這包括成為擁有角色的成員）。

可更新的檢視

簡單檢視會自動更新：系統會允許在檢視上使用 UPDATE、 INSERT和 DELETE陳述式，方式與一般資料表相同。如果檢視符合下列所有條件，則會自動更新檢視：

- 檢視在其FROM清單中必須只有一個項目，該項目必須是資料表或其他可更新的檢視。
- 檢視定義不得包含頂層的 WITH、DISTINCT、LIMIT、、 GROUP BY HAVING或 OFFSET子句。
- 檢視定義不得在頂層包含集合操作 (INTERSECT、 UNION或 EXCEPT)。
- 檢視的選取清單不得包含任何彙總、視窗函數或集合傳回函數。

自動可更新檢視可能包含可更新和不可更新資料欄的混合。如果資料欄是基礎基礎關係的可更新資料欄的簡單參考，則可進行更新。否則，資料欄為唯讀，如果 INSERT或 UPDATE陳述式嘗試為其指派值，則會發生錯誤。

對於自動可更新的檢視，系統會將檢視上的任何 UPDATE、 INSERT或 DELETE陳述式轉換為基礎基礎關係上的對應陳述式。 INSERT陳述式與 ON CONFLICT UPDATE子句完全支援。

如果自動可更新檢視包含WHERE條件，則條件會限制基本關係的哪些資料列可供檢視上的 UPDATE和 DELETE陳述式修改。不過， UPDATE可以變更資料列，使其不再滿足WHERE條件，使其無法透過檢視看見。同樣地， INSERT命令可能會插入不符合WHERE條件的基礎關係資料列，使它們無法透過檢視看見。 ON CONFLICT UPDATE可能會類似地影響無法透過檢視看見的現有資料列。

您可以使用 CHECK OPTION來防止 INSERT和 UPDATE 命令建立無法透過檢視看到的資料列。

如果自動可更新檢視以 security_barrier 屬性標記，則在新增檢視使用者的任何條件之前，一律會評估所有檢視WHERE的條件 （以及任何使用標記為 運算子的條件LEAKPROOF)。請注意，由於此原因，最終未傳回的資料列 （因為未通過使用者WHERE的條件） 可能仍會遭到鎖定。您可以使用 EXPLAIN 來查看在關聯層級套用哪些條件 （因此不會鎖定資料列），以及哪些條件不會。

根據預設，不符合所有這些條件的更複雜檢視是唯讀的：系統不允許在檢視上插入、更新或刪除。

> **Note**
>
> 在檢視上執行插入、更新或刪除的使用者，必須在檢視上具有對應的插入、更新或刪除權限。根據預設，檢視的擁有者必須擁有基礎基礎關係的相關權限，而執行更新的使用者則不需要基礎基礎關係的任何許可。不過，如果檢視的 security_invoker 設定為 true，則執行更新的使用者，而不是檢視擁有者，必須具有基礎基礎關係的相關權限。

範例

建立包含所有喜劇電影的檢視。

```
CREATE VIEW comedies AS
```

```
    SELECT *
    FROM films
    WHERE kind = 'Comedy';
```

這會建立檢視，其中包含建立檢視時film資料表中的資料欄。雖然 * 用來建立檢視，但稍後新增至資料表的資料欄不會成為檢視的一部分。

使用 建立檢視LOCAL CHECK OPTION。

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

這會建立同時檢查新資料列之 kind和 classification的檢視。

建立混合可更新和不可更新資料欄的檢視。

```
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';
```

此檢視將支援 INSERT、 UPDATE和 DELETE。影片資料表中的所有資料欄都是可更新的，而運算的資料欄country和 avg_rating將是唯讀的。

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

> **Note**
>
> 雖然遞迴檢視的名稱在此 中符合結構描述資格CREATE，但其內部自我參考不符合結構描述資格。這是因為隱含建立的通用資料表表達式 (CTE) 名稱不能符合結構描述資格。

相容性

CREATE OR REPLACE VIEW 是 PostgreSQL 語言延伸模組。WITH (...) 子句也是 延伸模組，也是安全障礙檢視和安全呼叫者檢視。Aurora DSQL 支援這些語言擴充功能。

ALTER VIEW

ALTER VIEW 陳述式允許變更現有檢視的各種屬性，而 Aurora DSQL 支援此命令的所有 PostgreSQL 語法。

支援的語法

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

描述

ALTER VIEW會變更檢視的各種輔助屬性。（如果您想要修改檢視的定義查詢，請使用 CREATE OR REPLACE VIEW。）您必須擁有檢視才能使用 ALTER VIEW。若要變更檢視的結構描述，您還必須具有新結構描述CREATE的權限。若要變更擁有者，您必須能夠SET ROLE使用新的擁有角色，且該角色必須具有檢視結構描述CREATE的權限。這些限制會強制變更擁有者不會執行任何您無法透過捨棄並重新建立檢視執行的動作。）

參數

ALTER VIEW 參數

name

現有檢視的名稱（選擇性符合結構描述資格）。

column_name

現有資料欄的新名稱。

IF EXISTS

如果檢視不存在，請勿擲回錯誤。在此情況下會發出通知。

SET/DROP DEFAULT

這些表單會設定或移除欄的預設值。檢視欄的預設值會替換為目標為檢視的任何 INSERT或 UPDATE命令。檢視的預設值將優先於基礎關係中的任何預設值。

new_owner

檢視新擁有者的使用者名稱。

new_name

檢視的新名稱。

new_schema

檢視的新結構描述。

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

設定或重設檢視選項。以下是支援的選項。

- check_option (enum) - 變更檢視的檢查選項。值必須為 local 或 cascaded。
- security_barrier (boolean) - 變更檢視的安全障礙屬性。此值必須是布林值，例如 true或 false。
- security_invoker (boolean) - 變更檢視的安全障礙屬性。此值必須是布林值，例如 true或 false。

備註

基於歷史 PostgreSQL 原因， 也可以與檢視ALTER TABLE搭配使用；但唯一允許與檢視搭配使用ALTER TABLE的變體等同於先前顯示的變體。

範例

將檢視重新命名foo為 bar。

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

將預設資料欄值連接至可更新的檢視。

```
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
```

相容性

ALTER VIEW 是 Aurora DSQL 支援的 SQL 標準的 PostgreSQL 擴充功能。

DROP VIEW

DROP VIEW 陳述式會移除現有的檢視。Aurora DSQL 支援此命令的完整 PostgreSQL 語法。

支援的語法

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

描述

DROP VIEW 會捨棄現有的檢視。若要執行此命令，您必須是檢視的擁有者。

參數

IF EXISTS

如果檢視不存在，請勿擲回錯誤。在此情況下會發出通知。

name

要移除之檢視的名稱（選擇性符合結構描述資格）。

CASCADE

自動捨棄依賴檢視的物件（例如其他檢視），然後捨棄依賴這些物件的所有物件。

RESTRICT

如果有任何物件相依於檢視，請拒絕捨棄檢視。此為預設值。

範例

```
DROP VIEW kinds;
```

相容性

此命令符合 SQL 標準，但標準只允許每個命令捨棄一個檢視，除了 `IF EXISTS`選項之外，這是 Aurora DSQL 支援的 PostgreSQL 擴充功能。

Aurora DSQL 中不支援的 PostgreSQL 功能

Aurora DSQL 與 PostgreSQL 相容。這表示 Aurora DSQL 支援核心關聯式功能，例如 ACID 交易、次要索引、聯結、插入和更新。如需支援的 SQL 功能概觀，請參閱支援的 SQL 表達式。

下列各節重點說明 Aurora DSQL 目前不支援哪些 PostgreSQL 功能。

不支援的物件

Aurora DSQL 不支援的物件包括下列項目：

- 單一 Aurora DSQL 叢集上的多個資料庫
- 暫時資料表
- 觸發
- 類型（部分支援）
- 資料表空間
- 以 SQL 以外語言撰寫的函數
- 序列
- 資料分割

不支援的限制條件

- 外部索引鍵
- 排除限制

不支援的命令

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

- VACUUM

> **Note**
>
> Aurora DSQL 不需要清空。系統會維護統計資料並自動管理儲存最佳化，無需手動清空命令。

不支援的延伸模組

Aurora DSQL 不支援 PostgreSQL 擴充功能。下表顯示不支援的擴充功能：

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

不支援的 SQL 表達式

下表說明 Aurora DSQL 中不支援的 子句。

類別	主要子句	不支援的子句
CREATE	INDEX ASYNC	ASC | DESC
CREATE	INDEX [1]	
TRUNCATE		
ALTER	SYSTEM	所有ALTER SYSTEM命令都會遭到封鎖。
CREATE	TABLE	COLLATE, AS SELECT, INHERITS, PARTITION

類別	主要子句	不支援的子句
CREATE	FUNCTION	LANGUAGE *non-sql-lang* ，其中 *non-sql-lang* 是 SQL
CREATE	TEMPORARY	TABLES
CREATE	EXTENSION	
CREATE	SEQUENCE	
CREATE	MATERIALIZED	VIEW
CREATE	TABLESPACE	
CREATE	TRIGGER	
CREATE	TYPE	
CREATE	DATABASE	您無法建立其他資料庫。

[1] 若要在指定資料表的資料欄上Aurora DSQL 中的非同步索引建立索引，請參閱 。

PostgreSQL 相容性的 Aurora DSQL 考量事項

使用 Aurora DSQL 時，請考慮下列相容性限制。如需一般考量，請參閱 使用 Amazon Aurora DSQL 的考量事項。如需配額和限制，請參閱 Amazon Aurora DSQL 中的叢集配額和資料庫限制。

- Aurora DSQL 使用名為 的單一內建資料庫postgres。您無法建立其他資料庫，或重新命名或捨棄postgres資料庫。
- postgres 資料庫使用 UTF-8 字元編碼。您無法變更編碼。
- 資料庫只會使用C定序。
- Aurora DSQL 使用 UTC做為系統時區。您不能使用參數或 SQL 陳述式修改時區，例如 SET TIMEZONE。
- 交易隔離層級在 PostgreSQL 固定Repeatable Read。
- 交易有下列限制：
  - 交易無法混合 DDL 和 DML 操作

  - 交易只能包含 1 個 DDL 陳述式
  - 無論次要索引的數量為何，交易最多可以修改 3，000 個資料列
  - 3，000 列限制適用於所有 DML 陳述式 (INSERT、UPDATE、DELETE)
- 資料庫連線會在 1 小時後逾時。
- Aurora DSQL 目前不允許您執行 `GRANT [permission] ON DATABASE`。如果您嘗試執行該陳述式，Aurora DSQL 會傳回錯誤訊息 `ERROR: unsupported object type in GRANT`。
- Aurora DSQL 不會讓非管理員使用者角色執行 `CREATE SCHEMA`命令。您無法執行 `GRANT [permission] on DATABASE`命令並授予資料庫的`CREATE`許可。如果非管理員使用者角色嘗試建立結構描述，Aurora DSQL 會傳回錯誤訊息 `ERROR: permission denied for database postgres`。
- 非管理員使用者無法在公有結構描述中建立物件。只有管理員使用者可以在公有結構描述中建立物件。管理員使用者角色具有許可，可將這些物件的讀取、寫入和修改存取權授予非管理員使用者，但無法授予公有結構描述本身的`CREATE`許可。非管理員使用者必須使用不同的使用者建立結構描述來建立物件。
- Aurora DSQL 不支援命令 `ALTER ROLE [] CONNECTION LIMIT`。如果您需要提高連線限制，請聯絡 AWS 支援。
- Aurora DSQL 不支援非同步 PostgreSQL 資料庫驅動程式，適用於 Python。

Aurora DSQL 中的並行控制

並行允許多個工作階段同時存取和修改資料，而不會影響資料完整性和一致性。Aurora DSQL 提供 PostgreSQL 相容性，同時實作現代、無鎖定的並行控制機制。它透過快照隔離來維持完整的 ACID 合規，以確保資料一致性和可靠性。

Aurora DSQL 的關鍵優勢是其無鎖定架構，可消除常見的資料庫效能瓶頸。Aurora DSQL 可防止緩慢的交易封鎖其他操作，並消除死結的風險。這種方法讓 Aurora DSQL 特別適用於效能和可擴展性至關重要的高輸送量應用程式。

交易衝突

Aurora DSQL 使用樂觀並行控制 (OCC)，其運作方式與傳統的鎖定型系統不同。OCC 不會使用鎖定，而是評估遞交時的衝突。當多個交易在更新相同資料列時發生衝突時，Aurora DSQL 會管理交易，如下所示：

- 具有最早遞交時間的交易由 Aurora DSQL 處理。

- 衝突的交易會收到 PostgreSQL 序列化錯誤，表示需要重試。

設計您的應用程式以實作重試邏輯來處理衝突。理想的設計模式是等冪的，盡可能讓交易重試成為第一個方法。建議的邏輯類似於標準 PostgreSQL 鎖定逾時或死鎖情況下的中止和重試邏輯。不過，OCC 需要您的應用程式更頻繁地執行此邏輯。

最佳化交易效能的指導方針

若要最佳化效能，請將單一金鑰或小型金鑰範圍上的高度爭用降至最低。若要達成此目標，請使用下列準則設計您的結構描述，將更新分散至叢集金鑰範圍：

- 為您的資料表選擇隨機主索引鍵。
- 避免增加單一金鑰爭用率的模式。即使交易量增加，此方法也能確保最佳效能。

Aurora DSQL 中的 DDL 和分散式交易

資料定義語言 (DDL) 在來自 PostgreSQL 的 Aurora DSQL 中的行為不同。Aurora DSQL 具有以多租用戶運算和儲存機群為基礎建置的異地同步備份分散式和共用無資料庫層。由於不存在單一主要資料庫節點或領導者，因此會分佈資料庫目錄。因此，Aurora DSQL 會以分散式交易的形式管理 DDL 結構描述變更。

具體而言，DDL 在 Aurora DSQL 中的行為不同，如下所示：

並行控制錯誤

如果您在另一個交易更新資源時執行一個交易，Aurora DSQL 會傳回並行控制違規錯誤。例如，請考慮下列動作順序：

1. 在工作階段 1 中，使用者將資料欄新增至資料表 `mytable`。
2. 在工作階段 2 中，使用者嘗試將資料列插入 `mytable`。

   Aurora DSQL 傳回錯誤 `SQL Error [40001]: ERROR: schema has been updated by another transaction, please retry: (OC001).`

相同交易中的 DDL 和 DML

Aurora DSQL 中的交易只能包含一個 DDL 陳述式，且不能同時包含 DDL 和 DML 陳述式。此限制表示您無法在相同交易中建立資料表並將資料插入相同的資料表。例如，Aurora DSQL 支援下列循序交易。

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

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

Aurora DSQL 不支援下列交易，其中包括 CREATE和 INSERT陳述式。

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

非同步 DDL

在標準 PostgreSQL 中，DDL 操作，例如CREATE INDEX鎖定受影響的資料表，使其無法供其他工作階段的讀取和寫入使用。在 Aurora DSQL 中，這些 DDL 陳述式會使用背景管理員以非同步方式執行。不會封鎖對受影響資料表的存取。因此，大型資料表上的 DDL 可以在不停機或效能影響的情況下執行。如需 Aurora DSQL 中非同步任務管理員的詳細資訊，請參閱 Aurora DSQL 中的非同步索引。

Aurora DSQL 中的主要金鑰

在 Aurora DSQL 中，主索引鍵是實際組織資料表資料的功能。它類似於 PostgreSQL 中的 CLUSTER操作或其他資料庫中的叢集索引。當您定義主索引鍵時，Aurora DSQL 會建立包含資料表中所有資料欄的索引。Aurora DSQL 中的主要金鑰結構可確保有效的資料存取和管理。

資料結構和儲存

當您定義主索引鍵時，Aurora DSQL 會以主索引鍵順序存放資料表資料。此索引組織結構允許主索引鍵查詢直接擷取所有資料欄值，而不是像傳統 B 樹索引一樣遵循資料指標。與 PostgreSQL 中僅重新

組織一次的資料CLUSTER操作不同，Aurora DSQL 會自動且持續地維護此順序。此方法可改善依賴主金鑰存取的查詢效能。

Aurora DSQL 也會使用主索引鍵，為資料表和索引中的每一列產生整個叢集的唯一索引鍵。此唯一金鑰也支援分散式資料管理。它可自動分割多個節點的資料，支援可擴展的儲存和高並行。因此，主索引鍵結構可協助 Aurora DSQL 自動擴展並有效率地管理並行工作負載。

選擇主索引鍵的準則

在 Aurora DSQL 中選擇和使用主索引鍵時，請考慮下列準則：

- 建立資料表時定義主索引鍵。您稍後無法變更此金鑰或新增新的主金鑰。主索引鍵會成為整個叢集的索引鍵的一部分，用於資料分割和寫入輸送量的自動擴展。如果您未指定主索引鍵，Aurora DSQL 會指派合成隱藏 ID。
- 對於具有高寫入磁碟區的資料表，請避免使用單調增加整數做為主索引鍵。這可能會導致效能問題，方法是將所有新的插入導向單一分割區。反之，請使用具有隨機分佈的主索引鍵，以確保跨儲存分割區平均分佈寫入。
- 對於不常變更或唯讀的資料表，您可以使用遞增索引鍵。遞增索引鍵的範例為時間戳記或序號。密集金鑰有許多緊密間隔或重複的值。即使寫入效能較不重要，您也可以使用遞增金鑰，即使金鑰密集也一樣。
- 如果完整資料表掃描不符合您的效能需求，請選擇更有效率的存取方法。在大多數情況下，這表示使用符合您在查詢中最常見聯結和查詢金鑰的主索引鍵。
- 主索引鍵中的資料欄合併大小上限為 1 KB。如需詳細資訊，請參閱 Aurora DSQL 中的資料庫限制和 Aurora DSQL 中支援的資料類型。
- 您可以在主索引鍵或輔助索引中包含最多 8 個資料欄。如需詳細資訊，請參閱 Aurora DSQL 中的資料庫限制和 Aurora DSQL 中支援的資料類型。

Aurora DSQL 中的非同步索引

`CREATE INDEX ASYNC` 命令會在指定資料表的一或多個資料欄上建立索引。此命令是一種非同步 DDL 操作，不會封鎖其他交易。當您執行 時`CREATE INDEX ASYNC`，Aurora DSQL 會立即傳回 `job_id`。

您可以使用`sys.jobs`系統檢視來監控此非同步任務的狀態。當索引建立任務正在進行時，您可以使用下列程序和命令：

sys.wait_for_job(job_id)*'your_index_creation_job_id'*

封鎖目前的工作階段，直到指定的任務完成或失敗為止。傳回布林值，指出成功或失敗。

DROP INDEX

取消進行中索引建置任務。

當非同步索引建立完成時，Aurora DSQL 會更新系統目錄，將索引標記為作用中。

> **Note**
>
> 請注意，在此更新期間存取相同命名空間中物件的並行交易可能會遇到並行錯誤。

當 Aurora DSQL 完成非同步索引任務時，它會更新系統目錄，以顯示索引處於作用中狀態。如果其他交易目前參考相同命名空間中的物件，您可能會看到並行錯誤。

語法

CREATE INDEX ASYNC 使用以下語法。

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

參數

UNIQUE

指示 Aurora DSQL 在每次新增資料時，檢查資料表中是否有重複的值。如果您指定此參數，插入和更新會導致重複項目的操作會產生錯誤。

IF NOT EXISTS

如果具有相同名稱的索引已存在，表示 Aurora DSQL 不應擲回例外狀況。在這種情況下，Aurora DSQL 不會建立新的索引。請注意，您嘗試建立的索引可能與已存在的索引具有非常不同的結構。如果您指定此參數，則需要索引名稱。

name

索引的名稱。您無法在此參數中包含結構描述的名稱。

Aurora DSQL 會在與其父資料表相同的結構描述中建立索引。索引的名稱必須與結構描述中任何其他物件的名稱不同，例如資料表或索引。

如果您未指定名稱，Aurora DSQL 會根據父資料表和索引資料欄的名稱自動產生名稱。例如，如果您執行 `CREATE INDEX ASYNC on table1 (col1, col2)`，Aurora DSQL 會自動命名索引 `table1_col1_col2_idx`。

`NULLS FIRST | LAST`

Null 和非 Null 資料欄的排序順序。 `FIRST`表示 Aurora DSQL 應該在非 Null 資料欄之前排序 Null 資料欄。 `LAST`表示 Aurora DSQL 應該在非 Null 資料欄之後排序 Null 資料欄。

`INCLUDE`

要作為非索引鍵資料欄包含在索引中的資料欄清單。您無法在索引掃描搜尋資格中使用非索引鍵資料欄。Aurora DSQL 會根據索引的唯一性忽略資料欄。

`NULLS DISTINCT | NULLS NOT DISTINCT`

指定 Aurora DSQL 是否應將 null 值視為唯一索引中的不同值。預設值為 `DISTINCT`，表示唯一索引可以包含資料欄中的多個 null 值。 `NOT DISTINCT`表示索引不能包含資料欄中的多個 null 值。

使用須知

請考量下列準則：

- `CREATE INDEX ASYNC` 命令不會引入鎖定。它也不會影響 Aurora DSQL 用來建立索引的基本資料表。
- 在結構描述遷移操作期間， `sys.wait_for_job(job_id)`*`'your_index_creation_job_id'`* 程序很有用。它可確保後續的 DDL 和 DML 操作以新建立的索引為目標。
- 每次 Aurora DSQL 執行新的非同步任務時，都會檢查`sys.jobs`檢視並刪除狀態為 `completed`或 `failed`超過 30 分鐘的任務。因此， `sys.jobs`主要會顯示進行中的任務，且不包含舊任務的相關資訊。
- 如果 Aurora DSQL 無法建立非同步索引，則索引會保留 `INVALID`。對於唯一索引，DML 操作會受到唯一性限制，直到您捨棄索引為止。我們建議您捨棄無效索引並重新建立索引。

建立索引：範例

下列範例示範如何建立結構描述、資料表，以及索引。

1. 建立名為 的資料表`test.departments`。

```
CREATE SCHEMA test;

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

2. 將資料列插入資料表。

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

3. 建立非同步索引。

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

   `CREATE INDEX` 命令會傳回任務 ID，如下所示。

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

   `job_id` 表示 Aurora DSQL 已提交新任務來建立索引。您可以使用 程序`sys.wait_for_job(job_id)`*`'your_index_creation_job_id'`*來封鎖工作階段的其他工作，直到工作完成或逾時為止。

查詢索引建立的狀態：範例

查詢`sys.jobs`系統檢視以檢查索引的建立狀態，如下列範例所示。

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

Aurora DSQL 會傳回類似以下的回應。

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

狀態欄可以是下列其中一個值。

submitted	processing	failed	completed
任務已提交，但 Aurora DSQL 尚未開始處理。	Aurora DSQL 正在處理任務。	任務失敗。如需詳細資訊，請參閱詳細資訊欄。如果 Aurora DSQL 無法建置索引，Aurora DSQL 不會自動移除索引定義。您必須使用 DROP INDEX命令手動移除索引。	Aurora DSQL

您也可以透過目錄資料表pg_index和 查詢索引的狀態pg_class。具體而言， 屬性 indisvalid 和 indisimmediate可以告訴您索引的狀態。當 Aurora DSQL 建立您的索引時，其初始狀態為 INVALID。索引的 indisvalid旗標會傳回 FALSE或 f，表示索引無效。如果旗標傳回 TRUE或 t，表示索引已就緒。

```
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)
```

唯一索引建置失敗

如果您的非同步唯一索引建置任務顯示失敗狀態與詳細資訊 Found duplicate key while validating index for UCVs，這表示由於唯一性限制違規而無法建置唯一索引。

解決唯一的索引建置失敗

1. 針對唯一次要索引中指定的索引鍵，移除主要資料表中具有重複項目的任何資料列。
2. 捨棄失敗的索引。
3. 發出新的建立索引命令。

在主資料表中偵測唯一性違規

下列 SQL 查詢可協助您識別資料表指定欄中的重複值。當您需要在目前未設定為主索引鍵或沒有唯一限制的資料欄上強制執行唯一性時，此功能特別有用，例如使用者資料表中的電子郵件地址。

以下範例示範如何建立範例使用者資料表、將包含已知複本的測試資料填入其中，然後執行偵測查詢。

定義資料表結構描述

```
-- 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
);
```

插入包含一組重複電子郵件地址的範例資料

```
-- 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');
```

執行重複的偵測查詢

```
-- 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;
```

檢視具有重複電子郵件地址的所有記錄

```
 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)
```

如果我們現在嘗試索引建立陳述式，將會失敗：

```
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)
```

Aurora DSQL 中的系統資料表和命令

請參閱下列各節，以了解 Aurora DSQL 中支援的系統資料表和目錄。

系統表

Aurora DSQL 與 PostgreSQL 相容，因此來自 PostgreSQL 的許多系統目錄資料表和檢視也存在於 Aurora DSQL 中。

重要的 PostgreSQL 目錄資料表和檢視

下表說明您可以在 Aurora DSQL 中使用的最常見資料表和檢視。

名稱	描述
pg_namespace	所有結構描述的資訊
pg_tables	所有資料表的資訊
pg_attribute	所有屬性的資訊
pg_views	（預先）定義檢視的資訊
pg_class	描述所有資料表、資料欄、索引和類似的物件
pg_stats	規劃器統計資料的檢視
pg_user	使用者的相關資訊
pg_roles	使用者和群組的相關資訊

名稱	描述
pg_indexes	列出所有索引
pg_constraint	列出資料表的限制

支援和不支援的目錄資料表

下表指出 Aurora DSQL 中支援和不支援哪些資料表。

名稱	適用於 Aurora DSQL
pg_aggregate	否
pg_am	是
pg_amop	否
pg_amproc	否
pg_attrdef	是
pg_attribute	是
pg_authid	否 （使用 pg_roles）
pg_auth_members	是
pg_cast	是
pg_class	是
pg_collation	是
pg_constraint	是
pg_conversion	否
pg_database	否

名稱	適用於 Aurora DSQL
pg_db_role_setting	是
pg_default_acl	是
pg_depend	是
pg_description	是
pg_enum	否
pg_event_trigger	否
pg_extension	否
pg_foreign_data_wrapper	否
pg_foreign_server	否
pg_foreign_table	否
pg_index	是
pg_inherits	是
pg_init_privs	否
pg_language	否
pg_largeobject	否
pg_largeobject_metadata	是
pg_namespace	是
pg_opclass	否
pg_operator	是
pg_opfamily	否

名稱	適用於 Aurora DSQL
pg_parameter_acl	是
pg_partitioned_table	否
pg_policy	否
pg_proc	否
pg_publication	否
pg_publication_namespace	否
pg_publication_rel	否
pg_range	是
pg_replication_origin	否
pg_rewrite	否
pg_seclabel	否
pg_sequence	否
pg_shdepend	是
pg_shdescription	是
pg_shseclabel	否
pg_statistic	是
pg_statistic_ext	否
pg_statistic_ext_data	否
pg_subscription	否
pg_subscription_rel	否

名稱	適用於 Aurora DSQL
pg_tablespace	否
pg_transform	否
pg_trigger	否
pg_ts_config	是
pg_ts_config_map	是
pg_ts_dict	是
pg_ts_parser	是
pg_ts_template	是
pg_type	是
pg_user_mapping	否

支援和不支援的系統檢視

下表指出 Aurora DSQL 支援和不支援哪些檢視。

名稱	適用於 Aurora DSQL
pg_available_extensions	否
pg_available_extension_versions	否
pg_backend_memory_contexts	是
pg_config	否
pg_cursors	否
pg_file_settings	否

名稱	適用於 Aurora DSQL
pg_group	是
pg_hba_file_rules	否
pg_ident_file_mappings	否
pg_indexes	是
pg_locks	否
pg_matviews	否
pg_policies	否
pg_prepared_statements	否
pg_prepared_xacts	否
pg_publication_tables	否
pg_replication_origin_status	否
pg_replication_slots	否
pg_roles	是
pg_rules	否
pg_seclabels	否
pg_sequences	否
pg_settings	是
pg_shadow	是
pg_shmem_allocations	是
pg_stats	是

名稱	適用於 Aurora DSQL
pg_stats_ext	否
pg_stats_ext_exprs	否
pg_tables	是
pg_timezone_abbrevs	是
pg_timezone_names	是
pg_user	是
pg_user_mappings	否
pg_views	是
pg_stat_activity	否
pg_stat_replication	否
pg_stat_replication_slots	否
pg_stat_wal_receiver	否
pg_stat_recovery_prefetch	否
pg_stat_subscription	否
pg_stat_subscription_stats	否
pg_stat_ssl	是
pg_stat_gssapi	否
pg_stat_archiver	否
pg_stat_io	否
pg_stat_bgwriter	否

名稱	適用於 Aurora DSQL
pg_stat_wal	否
pg_stat_database	否
pg_stat_database_conflicts	否
pg_stat_all_tables	否
pg_stat_all_indexes	否
pg_statio_all_tables	否
pg_statio_all_indexes	否
pg_statio_all_sequences	否
pg_stat_slru	否
pg_statio_user_tables	否
pg_statio_user_sequences	否
pg_stat_user_functions	否
pg_stat_user_indexes	否
pg_stat_progress_analyze	否
pg_stat_progress_basebackup	否
pg_stat_progress_cluster	否
pg_stat_progress_create_index	否
pg_stat_progress_vacuum	否
pg_stat_sys_indexes	否
pg_stat_sys_tables	否

名稱	適用於 Aurora DSQL
pg_stat_xact_all_tables	否
pg_stat_xact_sys_tables	否
pg_stat_xact_user_functions	否
pg_stat_xact_user_tables	否
pg_statio_sys_indexes	否
pg_statio_sys_sequences	否
pg_statio_sys_tables	否
pg_statio_user_indexes	否

sys.jobs 和 sys.iam_pg_role_mappings 檢視

Aurora DSQL 支援下列系統檢視：

sys.jobs

sys.jobs 提供非同步任務的狀態資訊。例如，在您建立非同步索引之後，Aurora DSQL 會傳回 job_uuid。您可以job_uuid搭配 使用此項目sys.jobs來查詢任務的狀態。

```
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

檢視sys.iam_pg_role_mappings提供授予 IAM 使用者之許可的相關資訊。例如，如果 DQSLDBConnect 是授予 Aurora DSQL 非管理員存取權的 IAM 角色，且名為 的使用者testuser獲得該DQSLDBConnect角色和對應的許可，您可以查詢sys.iam_pg_role_mappings檢視以查看哪些使用者獲得哪些許可。

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

pg_class 資料表

pg_class 資料表會儲存資料庫物件的中繼資料。若要取得資料表中有多少資料列的大致計數，請執行下列命令。

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

此命令會傳回如下輸出：

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

ANALYZE 命令

ANALYZE 命令會收集資料庫中資料表內容的統計資料，並將結果存放在pg_stats系統檢視中。之後，查詢規劃器會使用這些統計資料來協助判斷最有效率的查詢執行計畫。

在 Aurora DSQL 中，您無法在明確交易內執行 ANALYZE命令。 ANALYZE 不受資料庫交易逾時限制的約束。

為了減少手動介入的需求並使統計資料保持最新狀態，Aurora DSQL 會自動ANALYZE執行為背景程序。此背景任務會根據資料表中觀察到的變更率自動觸發。它連結到自上次分析以來已插入、更新或刪除的資料列（元組）數目。

ANALYZE 會在背景以非同步方式執行，其活動可透過下列查詢在系統檢視 sys.jobs 中進行監控：

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

重要考量事項

> **Note**
>
> ANALYZE 任務會像 Aurora DSQL 中的其他非同步任務一樣計費。當您修改資料表時，這可能會間接觸發自動背景統計資料收集任務，這可能會因為相關聯的系統層級活動而產生計量費用。

背景ANALYZE任務會自動觸發，收集與手動相同的統計資料類型，ANALYZE並依預設套用到使用者資料表。此自動化程序會排除系統和目錄資料表。

管理 Aurora DSQL 叢集

Aurora DSQL 提供多種組態選項，協助您建立符合您需求的正確資料庫基礎設施。若要設定 Aurora DSQL 叢集基礎設施，請檢閱下列各節。

主題

- 設定單一區域叢集
- 設定多區域叢集

本指南中討論的特徵和功能可確保您的 Aurora DSQL 環境更具彈性、回應能力，並且能夠在應用程式成長和發展時支援應用程式。

設定單一區域叢集

建立叢集

使用 create-cluster命令建立叢集。

> **Note**
>
> 叢集建立是非同步操作。呼叫 GetCluster API，直到狀態變更為 為止ACTIVE。您可以在叢集變成作用中後連線到叢集。

Example Command

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

> **Note**
>
> 若要在建立期間停用刪除保護，請包含 `--no-deletion-protection-enabled`旗標。

Example 回應

```
{
```

```
    "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"
    }
}
```

描述叢集

使用 get-cluster命令取得叢集的相關資訊。

Example Command

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

Example 回應

```
{
    "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"
    }
}
```

更新叢集

使用 update-cluster命令更新現有的叢集。

> **Note**
>
> 更新是非同步操作。呼叫 GetCluster API，直到狀態變更為 ACTIVE為止，查看您的變更。

Example Command

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

Example 回應

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

刪除叢集

使用 delete-cluster命令刪除現有的叢集。

> **Note**
>
> 您只能刪除已停用刪除保護的叢集。根據預設，當您建立新的叢集時，會啟用刪除保護。

Example Command

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

Example 回應

```
{
```

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

列出叢集

使用 list-clusters命令列出您的叢集。

Example Command

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

Example 回應

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

設定多區域叢集

本章說明如何設定和管理跨多個 的叢集 AWS 區域。

連線至多區域叢集

多區域對等叢集提供兩個區域端點，每個對等叢集各一個 AWS 區域。這兩個端點都提供單一邏輯資料庫，支援具有強大資料一致性的並行讀取和寫入操作。除了對等叢集之外，多區域叢集還具有見證區域，可存放有限時段的加密交易日誌，用於改善多區域耐用性和可用性。多區域見證區域沒有端點。

建立多區域叢集

若要建立多區域叢集，請先建立具有見證區域的叢集。然後，將此叢集與第二個叢集對等，第二個叢集與第一個叢集共用相同的見證區域。下列範例說明如何在美國東部（維吉尼亞北部）和美國東部（俄亥俄）中建立叢集，並以美國西部（奧勒岡）做為見證區域。

步驟 1：在美國東部（維吉尼亞北部）建立一個叢集

若要在美國東部（維吉尼亞北部）AWS 區域 使用多區域屬性建立叢集，請使用下列命令。

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

Example 回應：

```
{
    "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**
>
> 當 API 操作成功時，叢集會進入 PENDING_SETUP 狀態。叢集建立會保留在中，PENDING_SETUP直到您使用其對等叢集的 ARN 更新叢集為止。

步驟 2：在美國東部（俄亥俄）建立叢集 2

若要在美國東部（俄亥俄）AWS 區域 使用多區域屬性建立叢集，請使用下列命令。

```
aws dsql create-cluster \
--region us-east-2 \
```

```
--multi-region-properties '{"witnessRegion":"us-west-2"}'
```

Example 回應：

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

當 API 操作成功時，叢集會轉換為 PENDING_SETUP 狀態。在您使用另一個叢集的 ARN 進行對等互連之前，叢集建立會保持 PENDING_SETUP 狀態。

步驟 3：美國東部（維吉尼亞北部）與美國東部（俄亥俄）的對等叢集

若要將美國東部（維吉尼亞北部）叢集與美國東部（俄亥俄）叢集對等，請使用 update-cluster命令。使用美國東部（俄亥俄）叢集的 ARN，指定您的美國東部（維吉尼亞北部）叢集名稱和 JSON 字串。

```
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 回應

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

步驟 4：美國東部（俄亥俄）與美國東部（維吉尼亞北部）的對等叢集

若要將您的美國東部（俄亥俄）叢集與美國東部（維吉尼亞北部）叢集對等，請使用 update-cluster命令。使用美國東部（維吉尼亞北部）叢集的 ARN，指定您的美國東部（俄亥俄）叢集名稱和 JSON 字串。

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 回應

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

> **Note**
>
> 成功對等互連後，兩個叢集都會從「PENDING_SETUP」轉換為「CREATING」，並在準備好使用時最終轉換為「ACTIVE」狀態。

檢視多區域叢集屬性

當您描述叢集時，您可以檢視不同 中叢集的多區域屬性 AWS 區域。

Example

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

Example 回應

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

建立期間的對等叢集

您可以在叢集建立期間包含對等資訊，以減少步驟數量。在美國東部（維吉尼亞北部）建立第一個叢集（步驟 1) 之後，您可以在美國東部（俄亥俄）建立第二個叢集，同時透過包含第一個叢集的 ARN 來啟動對等互連程序。

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"]}'
```

這結合了步驟 2 和 4，但您仍然需要完成步驟 3（使用第二個叢集的 ARN 更新第一個叢集）來建立互連關係。完成所有步驟後，兩個叢集都會轉換到與標準程序中相同的狀態：從 PENDING_SETUP 到 CREATING，最後在準備就緒時轉換為 ACTIVE。

刪除多區域叢集

若要刪除多區域叢集，您需要完成兩個步驟。

1. 關閉每個叢集的刪除保護。
2. 在各自的 中分別刪除每個對等叢集 AWS 區域

在美國東部（維吉尼亞北部）更新和刪除叢集

1. 使用 update-cluster命令關閉刪除保護。

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

2. 使用 delete-cluster命令刪除叢集。

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

該命令會傳回下列回應。

```
{
    "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**
>
> 叢集會轉換為 PENDING_DELETE 狀態。在您刪除美國東部（俄亥俄）的對等叢集之前，刪除不會完成。

在美國東部（俄亥俄）更新和刪除叢集

1. 使用 update-cluster命令關閉刪除保護。

```
aws dsql update-cluster \
```

```
--region us-east-2 \
--identifier 'foo0bar1baz2quux3quux4quuux' \
--no-deletion-protection-enabled
```

2. 使用 delete-cluster命令刪除叢集。

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

命令會傳回下列回應：

```
{
    "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**
>
> 叢集會轉換為 PENDING_DELETE 狀態。幾秒鐘後，系統會在驗證後自動將兩個對等叢集轉換為 DELETING 狀態。

使用 設定多區域叢集 AWS CloudFormation

您可以使用相同的 AWS CloudFormation 資源AWS::DSQL::Cluster來部署和管理單一區域和多區域 Aurora DSQL 叢集。

如需如何使用 資源建立、修改和管理叢集的詳細資訊，請參閱 Amazon Aurora DSQL 資源類型參考。AWS::DSQL::Cluster

建立初始叢集組態

首先，建立 AWS CloudFormation 範本以定義多區域叢集：

```
---
Resources:
```

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

使用下列 CLI AWS 命令在兩個區域中建立堆疊：

```
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
```

尋找叢集識別符

擷取叢集的實體資源 IDs：

```
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"
]
```

更新叢集組態

更新您的 AWS CloudFormation 範本以包含兩個叢集 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
```

將更新的組態套用至兩個區域：

```
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
```

使用 Aurora DSQL 進行程式設計

Aurora DSQL 為您提供下列工具，以程式設計方式管理您的 Aurora DSQL 資源。

AWS Command Line Interface (AWS CLI)

您可以使用命令列 shell AWS CLI 中的 來建立和管理 資源。 AWS CLI 可讓您直接存取 APIs AWS 服務，例如 Aurora DSQL。如需 Aurora DSQL 命令的語法和範例，請參閱《 AWS CLI 命令參考》中的 dsql。

AWS 軟體開發套件 SDKs)

AWS 為許多熱門技術和程式設計語言提供SDKs。它們可讓您更輕鬆地 AWS 服務 從應用程式內以該語言或技術呼叫 。如需這些 SDKs的詳細資訊，請參閱開發和管理應用程式的工具 AWS。

Aurora DSQL API

此 API 是 Aurora DSQL 的另一個程式設計界面。使用此 API 時，您必須正確格式化每個 HTTPS 請求，並為每個請求新增有效的數位簽章。如需詳細資訊，請參閱API 參考。

AWS CloudFormation

AWS::DSQL::Cluster 是一項 AWS CloudFormation 資源，可讓您建立和管理 Aurora DSQL 叢集做為基礎設施的一部分做為程式碼。 AWS CloudFormation 可協助您在程式碼中定義整個 AWS 環境，讓您更輕鬆地以一致且可靠的方式佈建、更新和複寫基礎設施。

當您在 AWS CloudFormation 範本中使用 AWS::DSQL::Cluster 資源時，您可以與其他雲端資源一起宣告佈建 Aurora DSQL 叢集。這有助於確保您的資料基礎設施與應用程式堆疊的其餘部分一起部署和管理。

Amazon Aurora DSQL SDKs、驅動程式和範本程式碼

AWS 軟體開發套件 SDKs) 適用於許多熱門的程式設計語言。每個 SDK 都提供 API、程式碼範例和說明文件，讓開發人員能夠更輕鬆地以偏好的語言建置應用程式。

轉接器和方言

下表顯示 Aurora DSQL 可用的 ORM 轉接器和資料庫方言。

程式設計語言	架構	儲存庫連結
Java	Hibernate	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/

程式碼範例

使用 AWS SDK 進行叢集管理

下表顯示使用 AWS SDKs 的不同程式設計語言的叢集管理程式碼範例。

程式設計語言	範例儲存庫連結
C++	https://github.com/aws-samples/aurora-dsql-samples/tree/main/cpp/cluster_management
C# (.NET)	https://github.com/aws-samples/aurora-dsql-samples/tree/main/dotnet/cluster_management
Go	https://github.com/aws-samples/aurora-dsql-samples/tree/main/go/cluster_management
Java	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/cluster_management
JavaScript	https://github.com/aws-samples/aurora-dsql-samples/tree/main/javascript/cluster_management
Python	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/cluster_management

程式設計語言	範例儲存庫連結
Ruby	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/cluster_management
Rust	https://github.com/aws-samples/aurora-dsql-samples/tree/main/rust/cluster_management

驅動程式和物件相關映射 (ORMs範例

下表顯示不同程式設計語言的資料庫驅動程式和 ORM 架構程式碼範例。

程式設計語言	驅動程式或架構	範例儲存庫連結
C++	libpq	https://github.com/aws-samples/aurora-dsql-samples/tree/main/cpp/libpq
C# (.NET)	Npgsql	https://github.com/aws-samples/aurora-dsql-samples/tree/main/dotnet/npgsql
Go	pgx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/go/pgx
Java	Hibernate	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

程式設計語言	驅動程式或架構	範例儲存庫連結
JavaScript	Postgres.js	https://github.com/aws-samples/aurora-dsql-samples/tree/main/javascript/postgres-js
Python	Psycopg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/psycopg
Python	Psycopg2	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
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	Sequelize	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/sequelize
TypeScript	TypeORM	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/type-orm

使用 的 Aurora DSQL AWS CLI

請參閱下列各節，了解如何使用 管理您的叢集 AWS CLI。

CreateCluster

若要建立叢集，請使用 `create-cluster`命令。

> **Note**
>
> 叢集建立會以非同步方式進行。呼叫 `GetCluster` API，直到狀態為 為止`ACTIVE`。一旦叢集變成 ，您就可以連線到叢集`ACTIVE`。

範例命令

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

> **Note**
>
> 如果您想要在建立時停用刪除保護，請包含 `--no-deletion-protection-enabled`旗標。

回應範例

```
{
    "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

若要描述 叢集，請使用 `get-cluster`命令。

範例命令

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

回應範例

```
{
    "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

若要更新現有的叢集，請使用 `update-cluster`命令。

> **Note**
>
> 更新會以非同步方式進行。呼叫 `GetCluster` API，直到狀態為 `ACTIVE`為止，您將會看到變更。

範例命令

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

回應範例

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

DeleteCluster

若要刪除現有的叢集，請使用 `delete-cluster`命令。

> **Note**
>
> 您只能刪除已停用刪除保護的叢集。建立新叢集時，預設會啟用刪除保護。

範例命令

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

回應範例

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

ListClusters

若要取得叢集的 ，請使用 `list-clusters`命令。

範例命令

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

回應範例

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

多區域叢集上的 GetCluster

若要取得多區域叢集的相關資訊，請使用 `get-cluster`命令。對於多區域叢集，回應將包含連結 ARNs。

範例命令

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

回應範例

```
{
    "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"
    }
}
```

建立、讀取、更新、刪除 Aurora DSQL 叢集

建立、讀取、更新、刪除 (CRUD) 範例適用於單一區域和多區域部署。包含為每個程式設計語言的專用`cluster_management`區段，示範這些金鑰管理任務。

單一區域部署非常適合為特定地理區域的使用者提供服務的應用程式，提供簡化的管理和更低的延遲。多區域部署透過將資料庫分散到多個 ，協助您實現更高的可用性和災難復原功能 AWS 區域。

選擇符合您應用程式對可用性、效能和地理分佈需求的部署類型。

主題

- 建立叢集
- 取得叢集
- 更新 叢集
- 刪除叢集

建立叢集

請參閱以下資訊，了解如何在 Aurora DSQL 中建立單一區域和多區域叢集。

Python

若要在單一 中建立叢集 AWS 區域，請使用下列範例。

```
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()
```

若要建立多區域叢集，請使用下列範例。建立多區域叢集可能需要一些時間。

```
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++

下列範例可讓您在單一 中建立叢集 AWS 區域。

```
#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;
}
```

若要建立多區域叢集，請使用下列範例。建立多區域叢集可能需要一些時間。

```
#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

若要在單一 中建立叢集 AWS 區域，請使用下列範例。

```
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();
```

若要建立多區域叢集，請使用下列範例。建立多區域叢集可能需要一些時間。

```
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

使用下列範例在單一 中建立叢集 AWS 區域。

```
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);
        }
    }
}
```

若要建立多區域叢集，請使用下列範例。建立多區域叢集可能需要一些時間。

```
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

使用下列範例在單一 中建立叢集 AWS 區域。

```
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(())
}
```

若要建立多區域叢集，請使用下列範例。建立多區域叢集可能需要一些時間。

```
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

使用下列範例在單一 中建立叢集 AWS 區域。

```
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__
```

若要建立多區域叢集，請使用下列範例。建立多區域叢集可能需要一些時間。

```
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

使用下列範例在單一 中建立叢集 AWS 區域。

```
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);
        }
    }
}
```

若要建立多區域叢集，請使用下列範例。建立多區域叢集可能需要一些時間。

```
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

使用下列範例在單一 中建立叢集 AWS 區域。

```
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)
 }
}
```

若要建立多區域叢集，請使用下列範例。建立多區域叢集可能需要一些時間。

```
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)
 }

}
```

取得叢集

請參閱以下資訊，了解如何在 Aurora DSQL 中傳回叢集的資訊。

Python

若要取得單一或多區域叢集的相關資訊，請使用下列範例。

```
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++

使用下列範例取得單一或多區域叢集的相關資訊。

```
#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

若要取得單一或多區域叢集的相關資訊，請使用下列範例。

```
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

下列範例可讓您取得單一或多區域叢集的相關資訊。

```
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

下列範例可讓您取得單一或多區域叢集的相關資訊。

```
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

下列範例可讓您取得單一或多區域叢集的相關資訊。

```
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

下列範例可讓您取得單一或多區域叢集的相關資訊。

```
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

下列範例可讓您取得單一或多區域叢集的相關資訊。

```
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)
 }
}
```

更新 叢集

請參閱以下資訊，了解如何在 Aurora DSQL 中更新叢集。更新叢集可能需要一兩分鐘的時間。建議您等待一段時間，然後執行取得叢集以取得叢集的狀態。

Python

若要更新單一或多區域叢集，請使用下列範例。

```
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++

使用下列範例來更新單一或多區域叢集。

```
#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

若要更新單一或多區域叢集，請使用下列範例。

```
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

使用下列範例來更新單一或多區域叢集。

```
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

使用下列範例來更新單一或多區域叢集。

```
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

使用下列範例來更新單一或多區域叢集。

```
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

使用下列範例來更新單一或多區域叢集。

```
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

使用下列範例來更新單一或多區域叢集。

```
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)
 }
}
```

刪除叢集

請參閱以下資訊，了解如何在 Aurora DSQL 中刪除叢集。

Python

若要刪除單一叢集 AWS 區域，請使用下列範例。

```
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()
```

若要刪除多區域叢集，請使用下列範例。

```
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++

若要刪除單一叢集 AWS 區域，請使用下列範例。

```
#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;
}
```

若要刪除多區域叢集，請使用下列範例。刪除多區域叢集可能需要一些時間。

```
#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

若要刪除單一叢集 AWS 區域，請使用下列範例。

```
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();
```

若要刪除多區域叢集，請使用下列範例。刪除多區域叢集可能需要一些時間。

```
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

若要刪除單一叢集 AWS 區域，請使用下列範例。

```
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);
        }
    }
}
```

若要刪除多區域叢集，請使用下列範例。刪除多區域叢集可能需要一些時間。

```
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

若要刪除單一叢集 AWS 區域，請使用下列範例。

```
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(())
}
```

若要刪除多區域叢集，請使用下列範例。刪除多區域叢集可能需要一些時間。

```
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

若要刪除單一叢集 AWS 區域，請使用下列範例。

```
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__
```

若要刪除多區域叢集，請使用下列範例。刪除多區域叢集可能需要一些時間。

```
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

若要刪除單一叢集 AWS 區域，請使用下列範例。

```
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);
        }
    }
}
```

若要刪除多區域叢集，請使用下列範例。刪除多區域叢集可能需要一些時間。

```
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

若要刪除單一叢集 AWS 區域，請使用下列範例。

```
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)
 }
```

```
}
```

若要刪除多區域叢集，請使用下列範例。刪除多區域叢集可能需要一些時間。

```
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)
 }
}
```

教學課程

GitHub 上的下列教學課程和範例程式碼可協助您在 Aurora DSQL 中執行常見任務。

- 將 Benchbase 與 Aurora DSQL 搭配使用 – Benchbase 開放原始碼基準測試公用程式的一個分支，經過驗證可與 Aurora DSQL 搭配使用。
- Aurora DSQL 載入器 – 此開放原始碼 Python 指令碼可讓您針對使用案例更輕鬆地將資料載入 Aurora DSQL，例如填入資料表以進行測試或將資料傳輸至 Aurora DSQL。
- Aurora DSQL 範例 – GitHub 上的`aws-samples/aurora-dsql-samples`儲存庫包含程式碼範例，說明如何使用 AWS SDKs、物件關聯式映射器 (ORMs) 和 Web 架構，以各種程式設計語言連接和使用 Aurora DSQL。這些範例示範如何執行常見任務，例如安裝用戶端、處理身分驗證，以及執行 CRUD 操作。

AWS Lambda 搭配 Amazon Aurora DSQL 使用

下列教學課程說明如何將 Lambda 與 Aurora DSQL 搭配使用

先決條件

- 建立 Lambda 函數的授權。如需詳細資訊，請參閱 Lambda 入門。
- 建立或修改 Lambda 建立之 IAM 政策的授權。您需要許可 `iam:CreatePolicy`和 `iam:AttachRolePolicy`。如需詳細資訊，請參閱 IAM 的動作、資源和條件索引鍵。
- 您必須已安裝 npm v8.5.3 或更高版本。
- 您必須已安裝 zip v3.0 或更新版本。

在 中建立新的 函數 AWS Lambda。

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/lambda/ 開啟 AWS Lambda 主控台。
2. 選擇 Create function (建立函數)。
3. 提供名稱，例如 `dsql-sample`。
4. 請勿編輯預設設定，以確保 Lambda 建立具有基本 Lambda 許可的新角色。
5. 選擇 Create function (建立函數)。

授權您的 Lambda 執行角色連線到您的叢集

1. 在 Lambda 函數中，選擇組態 > 許可。
2. 選擇角色名稱以在 IAM 主控台中開啟執行角色。
3. 選擇新增許可 > 建立內嵌政策，然後使用 JSON 編輯器。
4. 在動作中貼上下列動作，以授權您的 IAM 身分使用 管理員資料庫角色進行連線。

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

> **Note**
>
> 我們使用管理員角色，將入門的先決條件步驟降至最低。您不應該為生產應用程式使用管理員資料庫角色。請參閱 以使用資料庫角色和 IAM 身分驗證了解如何建立具有最低資料庫許可的授權的自訂資料庫角色。

5. 在資源中，新增叢集的 Amazon Resource Name (ARN)。您也可以使用萬用字元。

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

6. 選擇下一步。
7. 輸入政策的名稱，例如 dsql-sample-dbconnect。
8. 選擇建立政策。

建立要上傳至 Lambda 的套件。

1. 建立名為 的資料夾myfunction。
2. 在 資料夾中，package.json使用下列內容建立名為 的新檔案。

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

3. 在 資料夾中，使用下列內容index.mjs在 目錄中建立名為 的檔案。

```
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. 使用下列命令來建立套件。

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

上傳程式碼套件並測試您的 Lambda 函數

1. 在 Lambda 函數的程式碼索引標籤中，選擇從 > .zip 檔案上傳
2. 上傳pkg.zip您建立的 。如需詳細資訊，請參閱使用 .zip 檔案封存部署 Node.js Lambda 函數。
3. 在 Lambda 函數的測試索引標籤中，貼上下列 JSON 承載，然後修改它以使用您的叢集 ID。
4. 在 Lambda 函數的測試索引標籤中，使用以下修改的事件 JSON 來指定叢集的端點。

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

5. 輸入事件名稱，例如 dsql-sample-test。選擇儲存。
6. 選擇測試。
7. 選擇詳細資訊以展開執行回應和日誌輸出。
8. 如果成功，Lambda 函數執行回應應傳回 200 狀態碼。

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

如果資料庫傳回錯誤，或者如果與資料庫的連線失敗，Lambda 函數執行回應會傳回 500 狀態碼。

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

Amazon Aurora DSQL 的備份和還原

Amazon Aurora DSQL 整合全受管的資料保護服務 AWS Backup，可讓您輕鬆地集中和自動化跨 AWS 服務、雲端和內部部署的備份，進而協助您符合法規合規和業務持續性需求。此服務可簡化單一區域和多區域 Aurora DSQL 叢集的備份建立、管理和還原。

重要功能如下所示：

- 透過 、 SDK AWS Management Console或 進行集中式備份管理 AWS CLI
- 完整叢集備份
- 自動化備份排程和保留政策
- 跨區域和跨帳戶功能
- 您存放的所有備份的 WORM （ 一次寫入、多讀 ） 組態

如需保存庫鎖定功能的詳細資訊，以及 Aurora DSQL AWS Backup 的廣泛可用 AWS Backup 功能清單，請參閱《 AWS Backup 開發人員指南》中的保存庫鎖定優點和AWS Backup 功能可用性。

入門 AWS Backup

AWS Backup 會建立 Aurora DSQL 叢集的完整副本。您可以依照入門 AWS Backup中的步驟開始使用 AWS Backup for Aurora DSQL：

1. 建立隨需備份以立即保護。
2. 建立自動化排程備份的備份計劃。
3. 設定保留期間和跨區域複製。
4. 設定備份活動的監控和通知。

還原備份

當您還原 Aurora DSQL 叢集時， AWS Backup 一律會建立新的叢集以保留您的來源資料。若要還原 Aurora DSQL 單一區域叢集，請使用 https://console.aws.amazon.com/backup 或 CLI 選取要還原的復原點 （ 備份 ）。設定將從備份建立之新叢集的設定。

僅支援透過 還原 Aurora DSQL 多區域叢集 AWS CLI。若要還原 Aurora DSQL 多區域叢集，您需要同時使用 AWS Backup 和 Aurora DSQL CLI。

還原 Aurora DSQL 多區域叢集。

1. 選取多區域叢集的復原點。
2. 將復原點複製到另一個 AWS 區域 支援多區域叢集的叢集。

> **Note**
>
> 不支援多區域叢集的區域會導致還原操作失敗。

3. 使用 CLI AWS Backup 為每個叢集啟動還原任務。
4. 使用 設定多區域叢集 文件對新建立的 Aurora DSQL 叢集進行對等互連。

如需這些步驟的詳細說明，請參閱 Amazon Aurora DSQL 還原文件。

若要還原至多區域 Aurora DSQL 叢集，您可以使用在單一 中擷取的備份 AWS 區域。不過，在啟動還原程序之前，您必須先將備份複製到 AWS 區域 支援多區域叢集的另一個備份。此步驟可確保還原操作可以成功完成。我們建議在美國東部（維吉尼亞北部）、美國東部（俄亥俄）或美國西部（奧勒岡）AWS 區域 等金鑰中建立備份複本，以啟用強大的災難復原選項並符合合規要求。

監控和合規

AWS Backup 使用下列資源提供備份和還原操作的完整可見性。

- 用於追蹤備份和還原任務的集中式儀表板
- 與 CloudWatch 和 CloudTrail 整合。
- 合規報告和稽核的 AWS Backup Audit Manager。

如需使用 記錄 Aurora DSQL 操作 AWS CloudTrail進一步了解如何使用 Aurora DSQL AWS 服務 記錄使用者、角色或 所採取動作的記錄。

其他資源

若要進一步了解 AWS Backup 功能，以及將其與 Aurora DSQL 搭配使用，請參閱下列資源：

- 的受管政策 AWS Backup
- Amazon Aurora DSQL 還原

- 支援的 服務 AWS 區域
- 中的備份加密 AWS Backup

透過使用 AWS Backup for Aurora DSQL，您可以實作強大、合規且自動化的備份策略，以保護關鍵資料庫資源，同時將管理開銷降至最低。無論您管理單一叢集或複雜的多區域部署， 都會 AWS Backup 提供所需的工具，以確保您的資料保持安全且可復原。

Aurora DSQL 的監控和記錄

監控和記錄是維護 Amazon Aurora DSQL 資源可靠性、可用性和效能的重要部分。您應該從 Aurora DSQL 資源的所有部分監控和收集記錄資料，以便輕鬆偵錯多點故障。

- Amazon CloudWatch AWS 會即時監控您的 AWS 資源和您在 上執行的應用程式。您可以收集和追蹤指標、建立自訂儀板表，以及設定警示，在特定指標達到您指定的閾值時通知您或採取動作。例如，您可以讓 CloudWatch 追蹤 CPU 使用量或其他 Amazon EC2 執行個體指標，並在需要時自動啟動新的執行個體。如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南。
- AWS CloudTrail 會擷取由 發出或代表發出的 API 呼叫和相關事件， AWS 帳戶 並將日誌檔案交付至您指定的 Amazon S3 儲存貯體。您可以識別呼叫的使用者和帳戶 AWS、進行呼叫的來源 IP 地址，以及呼叫的時間。如需詳細資訊，請參閱「AWS CloudTrail 使用者指南」。

檢視 Aurora DSQL 叢集狀態

Aurora DSQL 叢集狀態提供有關叢集運作狀態和連線能力的重要資訊。您可以使用 AWS CLI、 AWS Management Console或 Aurora DSQL API 檢視叢集和叢集執行個體的狀態。

Aurora DSQL 叢集狀態和定義

下表說明 Aurora DSQL 叢集的每個可能狀態，以及每個狀態的意義。

狀態	描述
正在建立	Aurora DSQL 正在嘗試建立或設定叢集的資源。當叢集處於此狀態時，任何連線嘗試都會失敗。
Active (作用中)	叢集可運作且隨時可供使用。
閒置	當叢集閒置的時間夠長，讓 Aurora DSQL 回收為其設定的資源時，就會變成閒置。當您連線到閒置叢集時，Aurora DSQL 會將叢集轉換回作用中狀態。
非作用中	當叢集上長時間沒有活動時，叢集會變成非作用中。當您嘗試連線到非作用中叢集時，Aurora DSQL 會自動將叢集轉換回作用中狀態。
更新中	當您變更叢集組態時，叢集會轉換為更新狀態。
正在刪除	當您提交刪除請求時，叢集會轉換為刪除狀態。

狀態	描述
Deleted (已刪除)	叢集已成功刪除。
失敗	Aurora DSQL 因為遇到錯誤而無法建立叢集。
待定設定	僅適用於多區域叢集。當您使用見證區域在第一個區域中建立多區域叢集時，多區域叢集會進入待定設定狀態。叢集建立會暫停，直到您在次要區域中建立另一個叢集，並將兩個叢集對等。
待刪除	僅適用於多區域叢集。當您從叢集中刪除叢集時，多區域叢集會進入待定刪除狀態。一旦您刪除最後一個對等叢集，叢集就會移至刪除狀態。

檢視您的 Aurora DSQL 叢集狀態

若要檢視叢集的狀態，請使用 AWS CLI AWS Management Console或 Aurora DSQL API。

主控台

請依照下列步驟，在 中檢視叢集狀態 AWS Management Console：

在主控台中檢視叢集狀態

1. 在 開啟 Aurora DSQL 主控台https://console.aws.amazon.com/dsql。
2. 在導覽窗格中，選擇 Clusters (叢集)。
3. 在儀表板中檢視每個叢集的狀態。

AWS CLI

使用下列 AWS CLI 命令來檢查單一叢集的狀態。

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

執行下列命令來列出所有叢集的狀態。

```
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
```

此範例輸出會顯示兩個作用中叢集，以及一個正在刪除的叢集。

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

使用 Amazon CloudWatch 監控 Aurora DSQL

使用 CloudWatch 監控 Aurora DSQL，它會收集原始資料並將其處理為可讀且近乎即時的指標。CloudWatch 會保留這些統計資料 15 個月，協助您更清楚 Web 應用程式或服務效能。設定警示以監控特定閾值，並在符合時傳送通知或採取動作。檢閱下列適用於 Aurora DSQL 的用量和可觀測性指標。

如需更多資訊，請參閱 Amazon CloudWatch 使用者指南。

可觀測性和效能

此資料表概述 Aurora DSQL 的可觀測性指標。它包含用於追蹤唯讀和總交易的指標，以提供整體工作負載特性。包括查詢逾時和 OCC 衝突率等可行指標，以協助識別效能問題和並行衝突。與工作階段相關的指標，包括作用中和總計，都提供對系統目前負載的洞見。

CloudWatch 指標名稱	指標	單位	描述
ReadOnlyTransactions	Read-only transactions	none	The number of read-only transactions
TotalTransactions	Total transactions	none	The total number of transactions executed on the system, including read-only transactions.
QueryTimeouts	Query timeouts	none	The number of queries which have

CloudWatch 指標名稱	指標	單位	描述
			timed out due to hitting the maximum transaction time
OccConflicts	OCC conflicts	none	The number of transactions aborted due to key level OCC
CommitLatency	Commit Latency	milliseconds	Time spent by commit phase of query execution (P50)
BytesWritten	Bytes Written	bytes	Bytes written to storage
BytesRead	Bytes Read	bytes	Bytes read from storage
ComputeTime	QP compute time	milliseconds	QP wall clock time
ClusterStorageSize	Cluster Storage Size	bytes	Cluster size

用量指標

Aurora DSQL 會使用稱為分散式處理單元 (DPU) 的單一標準化帳單單位，測量所有以請求為基礎的活動，例如查詢處理、讀取和寫入。

CloudWatch 指標名稱	指標	維度：ResourceId	單位	描述
WriteDPU	Write Units	<cluster-id>	DPU	Approximates the write active-use component of your Aurora DSQL cluster DPU usage.

CloudWatch 指標名稱	指標	維度：ResourceId	單位	描述
MultiRegionWriteDPU	Multi-Region Write Units	<cluster-id>	DPU	Applicable for Multi-Region clusters: Approximates the multi-Region write active-use component of your Aurora DSQL cluster DPU usage.
ReadDPU	Read Units	<cluster-id>	DPU	Approximates the read active-use component of your Aurora DSQL cluster DPU usage.
ComputeDPU	Compute Units	<cluster-id>	DPU	Approximates the compute active-use component of your Aurora DSQL cluster DPU usage.
TotalDPU	Total Units	<cluster-id>	DPU	Approximates the total active-use component of your Aurora DSQL cluster DPU usage.

使用 記錄 Aurora DSQL 操作 AWS CloudTrail

Amazon Aurora DSQL 已與 整合AWS CloudTrail，這項服務可提供使用者、角色或 所採取動作的記錄 AWS 服務。CloudTrail 中有兩種類型的事件：管理事件和資料事件。管理事件會發出以稽核 AWS 資源組態變更。資料事件通常會擷取服務資料平面中的 AWS 資源用量。

CloudTrail 會將 Aurora DSQL 的所有 API 呼叫擷取為事件。Aurora DSQL 會將主控台活動記錄為管理事件。它也會將對叢集的已驗證連線嘗試擷取為資料事件。

您可以使用 CloudTrail 所收集的資訊，判斷對 Aurora DSQL 提出的請求、提出請求的 IP 地址、提出請求的時間、提出請求的使用者身分，以及其他詳細資訊。

您建立帳戶 AWS 帳戶 時，預設會在 中啟用 CloudTrail，而且您可以存取 CloudTrail 事件歷史記錄。CloudTrail 事件歷史記錄為 AWS 區域中過去 90 天記錄的管理事件，提供可檢視、可搜尋、可下載且不可變的記錄。如需詳細資訊，請參閱「AWS CloudTrail 使用者指南」中的使用 CloudTrail 事件歷史記錄。記錄事件歷史記錄無需支付 CloudTrail 費用。

若要建立 AWS 帳戶中事件的持續記錄，包括 Aurora DSQL 的事件，請建立追蹤或 AWS CloudTrail Lake 事件資料存放區（AWS CloudTrail 事件的集中式儲存和分析解決方案）。如需建立追蹤的詳細資訊，請參閱使用 CloudTrail 追蹤。若要了解如何設定和管理事件資料存放區，請參閱 CloudTrail Lake 事件資料存放區。

CloudTrail 中的 Aurora DSQL 管理事件

CloudTrail 管理事件提供對 AWS 帳戶中資源執行之管理操作的相關資訊。這些也稱為控制平面操作。根據預設，CloudTrail 會擷取事件歷史記錄中的管理事件。

Amazon Aurora DSQL 會將所有 Aurora DSQL 控制平面操作記錄為管理事件。如需 Aurora DSQL 記錄到 CloudTrail 的 Amazon Aurora DSQL 控制平面操作清單，請參閱 Aurora DSQL API 參考。

控制平面日誌

Amazon Aurora DSQL 會將下列 Aurora DSQL 控制平面操作記錄到 CloudTrail 做為管理事件。

- CreateCluster
- DeleteCluster
- GetCluster
- GetVpcEndpointServiceName

- ListClusters
- ListTagsForResource
- TagResource
- UntagResource
- UpdateCluster

備份和還原日誌

Amazon Aurora DSQL 會將下列 Aurora DSQL 備份和還原操作記錄為管理事件至 CloudTrail。

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

如需使用 保護 Aurora DSQL 叢集的詳細資訊 AWS Backup，請參閱 Amazon Aurora DSQL 的備份和還原 。

AWS KMS 日誌

Amazon Aurora DSQL 會將下列 AWS KMS 操作記錄到 CloudTrail 做為管理事件。

- `GenerateDataKey`
- `Decrypt`

若要進一步了解 CloudTrail 日誌如何追蹤 Aurora DSQL AWS KMS 代表您傳送到 的請求，請參閱 監控 Aurora DSQL 與 的互動 AWS KMS。

CloudTrail 中的 Aurora DSQL 資料事件

CloudTrail 資料事件通常會提供有關在資源上執行或在資源中執行的資源操作的資訊。這些也會用來擷取服務的資料平面操作。資料事件通常是大量資料的活動。根據預設，CloudTrail 不會記錄資料事件。CloudTrail 事件歷史記錄不會記錄資料事件。

如需如何記錄資料事件的詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的使用 AWS Management Console記錄資料事件和使用 AWS Command Line Interface記錄資料事件。

資料事件需支付額外的費用。如需 CloudTrail 定價的詳細資訊，請參閱 AWS CloudTrail 定價。

對於 Aurora DSQL，CloudTrail 會將對 Aurora DSQL 叢集所做的任何連線嘗試擷取為資料事件。下表列出您可以記錄資料事件的 Aurora DSQL 資源類型。資源類型 （ 主控台 ） 欄顯示從 CloudTrail 主控台上的資源類型清單中選擇的值。resources.type 值欄會顯示值，您會在使用 AWS CLI 或 CloudTrail APIs 設定進階事件選取器時指定此`resources.type`值。記錄到 CloudTrail 的資料 API 資料行會針對資源類型顯示記錄到 CloudTrail 的 API 呼叫。

資源類型 （ 主控台 ）	resources.type 值	記錄到 CloudTrail 的資料 API
Amazon Aurora DSQL	`AWS::DSQL::Cluster`	• `DbConnect` • `DbConnectAdmin`

您可以設定進階事件選取器來篩選 `eventName`和 `resources.ARN` 欄位，以僅記錄已篩選的事件。如需這些欄位的詳細資訊，請參閱AWS CloudTrail API 參考中的AdvancedFieldSelector。

下列範例示範如何使用 AWS CLI 設定 `dsql-data-events-trail` 來接收 Aurora DSQL 的資料事件。

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

Amazon Aurora DSQL 的安全性

的雲端安全性 AWS 是最高優先順序。身為 AWS 客戶，您可以受益於資料中心和網路架構，這些架構是為了滿足最安全敏感組織的需求而建置。

安全性是 AWS 與您之間的共同責任。共同責任模型將其描述為雲端的安全性和雲端中的安全性：

- 雲端的安全性 – AWS 負責保護在 中執行 AWS 服務的基礎設施 AWS 雲端。 AWS 也為您提供可安全使用的服務。作為AWS 合規計劃的一部分，第三方稽核人員會定期測試和驗證我們安全的有效性。若要了解適用於 Amazon Aurora DSQL 的合規計劃，請參閱AWS 合規計劃的 服務範圍。
- 雲端的安全性 – 您的責任取決於您使用 AWS 的服務。您也必須對其他因素負責，包括資料的機密性、您公司的要求和適用法律和法規。

本文件可協助您了解如何在使用 Aurora DSQL 時套用共同責任模型。下列主題說明如何設定 Aurora DSQL 以符合您的安全與合規目標。您也會了解如何使用其他 AWS 服務來協助您監控和保護 Aurora DSQL 資源。

主題

- AWS Amazon Aurora DSQL 的 受管政策
- Amazon Aurora DSQL 中的資料保護
- Amazon Aurora DSQL 的資料加密
- Aurora DSQL 的身分和存取管理
- 在 Aurora DSQL 中使用服務連結角色
- 搭配 Amazon Aurora DSQL 使用 IAM 條件金鑰
- Amazon Aurora DSQL 中的事件回應
- Amazon Aurora DSQL 的合規驗證
- Amazon Aurora DSQL 中的彈性
- Amazon Aurora DSQL 中的基礎設施安全性
- Amazon Aurora DSQL 中的組態和漏洞分析
- 預防跨服務混淆代理人
- Aurora DSQL 的安全最佳實務

AWS Amazon Aurora DSQL 的 受管政策

AWS 受管政策是由 AWS AWS 受管政策建立和管理的獨立政策旨在為許多常用案例提供許可，以便您可以開始將許可指派給使用者、群組和角色。

請記住， AWS 受管政策可能不會授予特定使用案例的最低權限許可，因為這些許可可供所有 AWS 客戶使用。我們建議您定義使用案例專屬的客戶管理政策，以便進一步減少許可。

您無法變更 AWS 受管政策中定義的許可。如果 AWS 更新 AWS 受管政策中定義的許可，則更新會影響政策連接的所有主體身分（使用者、群組和角色）。當新的 啟動或新的 API 操作可供現有 服務使用時， AWS 最有可能更新 AWS 服務 AWS 受管政策。

如需詳細資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策。

AWS 受管政策：AmazonAuroraDSQLFullAccess

您可以`AmazonAuroraDSQLFullAccess`連接到您的使用者、群組和角色。

此政策會授予許可，以允許完整管理存取 Aurora DSQL。具有這些許可的主體可以建立、刪除和更新 Aurora DSQL 叢集，包括多區域叢集。他們可以從叢集新增和移除標籤。他們可以列出叢集並檢視個別叢集的相關資訊。他們可以查看連接到 Aurora DSQL 叢集的標籤。他們可以任何使用者身分連線到資料庫，包括管理員。他們可以執行 Aurora DSQL 叢集的備份和還原操作，包括啟動、停止和監控備份和還原任務。此政策包含允許對用於叢集加密的客戶受管金鑰進行操作的 AWS KMS 許可。他們可以在您的帳戶上查看來自 CloudWatch 的任何指標。他們也具有為`dsql.amazonaws.com`服務建立服務連結角色的許可，這是建立叢集所需的。

許可詳細資訊

此政策包含以下許可。

- `dsql`- 授予委託人對 Aurora DSQL 的完整存取權。
- `cloudwatch`- 授予將指標資料點發佈至 Amazon CloudWatch 的許可。
- `iam`- 授予建立服務連結角色的許可。

- backup and restore- 授予許可，以啟動、停止和監控 Aurora DSQL 叢集的備份和還原任務。
- kms- 在建立、更新或連線至叢集時，授予驗證存取用於 Aurora DSQL 叢集加密之客戶受管金鑰所需的許可。

您可以在 IAM 主控台和《 AWS 受管AmazonAuroraDSQLFullAccess政策參考指南》中的 AmazonAuroraDSQLFullAccess 中找到政策。

AWS 受管政策：AmazonAuroraDSQLReadOnlyAccess

您可以AmazonAuroraDSQLReadOnlyAccess連接到您的使用者、群組和角色。

允許讀取存取 Aurora DSQL。具有這些許可的主體可以列出叢集，並檢視個別叢集的相關資訊。他們可以查看連接到 Aurora DSQL 叢集的標籤。他們可以在您帳戶中從 CloudWatch 擷取和查看任何指標。

許可詳細資訊

此政策包含以下許可。

- dsql – 授予 Aurora DSQL 中所有資源的唯讀許可。
- cloudwatch – 准許擷取 CloudWatch 指標資料的批次量，並對擷取的資料執行指標數學

您可以在 IAM 主控台上找到AmazonAuroraDSQLReadOnlyAccess政策，並在 AWS 受管政策參考指南中找到 AmazonAuroraDSQLReadOnlyAccess。

AWS 受管政策：AmazonAuroraDSQLConsoleFullAccess

您可以AmazonAuroraDSQLConsoleFullAccess連接到您的使用者、群組和角色。

允許透過 對 Amazon Aurora DSQL 進行完整管理存取 AWS Management Console。具有這些許可的主體可以使用 主控台建立、刪除和更新 Aurora DSQL 叢集，包括多區域叢集。他們可以列出叢集並檢視個別叢集的相關資訊。他們可以查看您帳戶上任何資源的標籤。他們可以任何使用者身分連線到資料庫，包括管理員。他們可以執行 Aurora DSQL 叢集的備份和還原操作，包括啟動、停止和監控備份和還原任務。此政策包含允許對用於叢集加密的客戶受管金鑰進行操作的 AWS KMS 許可。他們可以 AWS CloudShell 從 啟動 AWS Management Console。他們可以在您的帳戶上查看來自 CloudWatch

的任何指標。他們也具有為dsql.amazonaws.com服務建立服務連結角色的許可，這是建立叢集所需的。

您可以在 IAM 主控台上找到AmazonAuroraDSQLConsoleFullAccess政策，並在 AWS 受管政策參考指南中找到 AmazonAuroraDSQLConsoleFullAccess。

許可詳細資訊

此政策包含以下許可。

- dsql- 透過 授予 Aurora DSQL 中所有資源的完整管理許可 AWS Management Console。
- cloudwatch- 准許擷取 CloudWatch 指標資料的批次量，並在擷取的資料上執行指標數學。
- tag- 授予許可，以傳回目前在 AWS 區域 為呼叫帳戶指定的 中使用的標籤索引鍵和值。
- backup and restore- 授予許可，以啟動、停止和監控 Aurora DSQL 叢集的備份和還原任務。
- kms- 在建立、更新或連線至叢集時，授予驗證存取用於 Aurora DSQL 叢集加密之客戶受管金鑰所需的許可。
- cloudshell- 授予啟動 AWS CloudShell 以與 Aurora DSQL 互動的許可。
- ec2- 准許檢視 Aurora DSQL 連線所需的 Amazon VPC 端點資訊。

您可以在 IAM 主控台上找到AmazonAuroraDSQLReadOnlyAccess政策，並在 AWS 受管政策參考指南中找到 AmazonAuroraDSQLReadOnlyAccess。

AWS 受管政策：AuroraDSQLServiceRolePolicy

您無法將 AuroraDSQLServiceRolePolicy 連接至 IAM 實體。此政策會連接到服務連結角色，允許 Aurora DSQL 存取帳戶資源。

您可以在 IAM 主控台和 AWS 受管AuroraDSQLServiceRolePolicy政策參考指南中的 AuroraDSQLServiceRolePolicy 中找到政策。

AWS 受管政策的 Aurora DSQL 更新

檢視自此服務開始追蹤這些變更以來，Aurora DSQL AWS 受管政策更新的詳細資訊。如需此頁面變更的自動提醒，請訂閱 Aurora DSQL 文件歷史記錄頁面上的 RSS 摘要。

變更	描述	日期
AmazonAuroraDSQLFullAccess 更新	新增為 Aurora DSQL 叢集執行備份和還原操作的功能，包括啟動、停止和監控任務。它還新增了使用客戶受管 KMS 金鑰進行叢集加密的功能。 如需詳細資訊，請參閱 AmazonAuroraDSQLFullAccess 和在 Aurora DSQL 中使用服務連結角色。	2025 年 5 月 21 日
AmazonAuroraDSQLConsoleFullAccess 更新	新增透過 執行 Aurora DSQL 叢集備份和還原操作的功能 AWS Console Home。這包括啟動、停止和監控任務。它還支援使用客戶管理的 KMS 金鑰進行叢集加密和啟動 AWS CloudShell。 如需詳細資訊，請參閱 Aurora DSQL 中的 AmazonAuroraDSQLConsoleFullAccess 和使用服務連結角色。	2025 年 5 月 21 日
AmazonAuroraDSQLFullAccess 更新	政策會新增四個新許可，以跨多個 建立和管理資料庫叢集 AWS 區域：`PutMultiRegionProperties` 、`AddPeerCluster` 、 `PutWitnessRegion` 和 `RemovePeerCluster` 。這些許可包括資源層級控制項和條件索引鍵，讓您可以控制可以修改哪些叢集使用者。	2025 年 5 月 13 日

變更	描述	日期
	此政策也會新增 `GetVpcEndpointServiceName` 許可，協助您透過 連線至 Aurora DSQL 叢集 AWS PrivateLink。 如需詳細資訊，請參閱 AmazonAuroraDSQLFullAccess 和在 Aurora DSQL 中使用服務連結角色。	
AmazonAuroraDSQLReadOnlyAccess 更新	包括透過 Aurora DSQL 連線至 Aurora DSQL 叢集時判斷正確 VPC AWS PrivateLink 端點服務名稱的能力，可為每個儲存格建立唯一的端點，因此此 API 有助於確保您可以識別叢集的正確端點，並避免連線錯誤。 如需詳細資訊，請參閱 Aurora DSQL 中的 AmazonAuroraDSQLReadOnlyAccess 和使用服務連結角色。	2025 年 5 月 13 日

變更	描述	日期
AmazonAuroraDSQLConsoleFullAccess 更新	將新許可新增至 Aurora DSQL，以支援多區域叢集管理和 VPC 端點連線。新的許可包括： `PutMultiRegionProperties` `PutWitnessRegion` `AddPeerCluster` `RemovePeerCluster` `GetVpcEndpointServiceName` 如需詳細資訊，請參閱 Aurora DSQL 中的 AmazonAuroraDSQLConsoleFullAccess 和使用服務連結角色。	2025 年 5 月 13 日
AuroraDsqlServiceLinkedRolePolicy 更新	新增將指標發佈至 的功能，`AWS/AuroraDSQL` 以及將 `AWS/Usage` CloudWatch 命名空間發佈至政策的功能。這可讓相關聯的服務或角色將更全面的用量和效能資料傳送到 CloudWatch 環境。 如需詳細資訊，請參閱 AuroraDsqlServiceLinkedRolePolicy 和在 Aurora DSQL 中使用服務連結角色。	2025 年 5 月 8 日
頁面已建立	開始追蹤與 Amazon Aurora DSQL 相關的 AWS 受管政策	2024 年 12 月 3 日

Amazon Aurora DSQL 中的資料保護

共同責任模型適用於 中的資料保護。如此模型所述， 負責保護執行所有 的 全球基礎設施 AWS 雲端。您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使用 的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱資料隱私權常見問答集。如需有關歐洲資料保護的相關資訊，請參閱 安全性部落格上的 共同的責任模型和 GDPR 部落格文章。

基於資料保護目的，我們建議您保護登入資料，並使用 AWS IAM Identity Center 或 設定個別使用者 AWS Identity and Access Management。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 資源通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用 設定 API 和使用者活動記錄 AWS CloudTrail。如需有關使用線索擷取活動的資訊，請參閱《 使用者指南》中的使用線索。
- 使用加密解決方案，以及 中的所有預設安全控制 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。

我們強烈建議您絕對不要將機密或敏感資訊，例如您的客戶電子郵件地址，放入標籤或任意格式的文字欄位中，例如名稱欄位。這包括當您使用 或使用 主控台、API、 AWS CLI或 AWS SDKs的其他 時。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

資料加密

Amazon Aurora DSQL 提供高耐用性的儲存基礎設施，專為關鍵任務和主要資料儲存而設計。資料會以備援方式存放在 Aurora DSQL 區域中跨多個設施的多個裝置上。

傳輸中加密

根據預設，會為您設定傳輸中加密。Aurora DSQL 使用 TLS 來加密 SQL 用戶端和 Aurora DSQL 之間的所有流量。

在 AWS CLI SDK 或 API 用戶端與 Aurora DSQL 端點之間加密和簽署傳輸中的資料：

- Aurora DSQL 提供 HTTPS 端點，用於加密傳輸中的資料。

- 為了保護對 Aurora DSQL 提出 API 請求的完整性，發起人必須簽署 API 呼叫。根據 Signature 第 4 版簽署程序 (Sigv4)，呼叫由 X.509 憑證或客戶的 AWS 私密存取金鑰進行簽署。如需詳細資訊，請參閱《AWS 一般參考》中的 Signature 第 4 版簽署程序。
- 使用 AWS CLI 或其中一個 AWS SDKs 向 提出請求 AWS。這些工具會自動使用您設定工具時指定的存取金鑰，替您簽署請求。

如需靜態加密，請參閱 Aurora DSQL 中的靜態加密。

網際網路流量隱私權

Aurora DSQL 和內部部署應用程式之間，以及 Aurora DSQL 和相同資源內其他 AWS 資源之間的連線都會受到保護 AWS 區域。

您的私有網路與 之間有兩個連線選項 AWS：

- AWS Site-to-Site連接。如需詳細資訊，請參閱什麼是 AWS Site-to-Site VPN？
- AWS Direct Connect 連線。如需詳細資訊，請參閱什麼是 AWS Direct Connect？

您可以使用 AWS發佈的 API 操作，透過網路存取 Aurora DSQL。使用者端必須支援下列專案：

- Transport Layer Security (TLS)。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 具備完美轉送私密(PFS)的密碼套件，例如 DHE (Ephemeral Diffie-Hellman)或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

為 Aurora DSQL 連線設定 SSL/TLS 憑證

Aurora DSQL 需要所有連線才能使用 Transport Layer Security (TLS) 加密。若要建立安全連線，您的用戶端系統必須信任 Amazon 根憑證授權機構 (Amazon Root CA 1)。此憑證已預先安裝在許多作業系統上。本節提供在各種作業系統上驗證預先安裝的 Amazon Root CA 1 憑證的說明，並引導您完成手動安裝憑證的程序。

建議使用 PostgreSQL 17 版。

> **Important**
>
> 對於生產環境，建議使用 `verify-full` SSL 模式以確保最高層級的連線安全性。此模式會驗證伺服器憑證是否由信任的憑證授權單位簽署，以及伺服器主機名稱是否與憑證相符。

驗證預先安裝的憑證

在大多數作業系統中，Amazon Root CA 1 已預先安裝。若要驗證這一點，您可以遵循下列步驟。

Linux (RedHat/CentOS/Fedora)

在終端機中執行下列命令：

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

如果已安裝憑證，您會看到下列輸出：

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

macOS

1. 開啟 Spotlight 搜尋 (命令 + 空間)
2. 搜尋 Keychain 存取
3. 選取系統金鑰鏈下的系統根目錄
4. 在憑證清單中尋找 Amazon Root CA 1

Windows

> **Note**
>
> 由於 psql Windows 用戶端的已知問題，使用系統根憑證 (`sslrootcert=system`) 可能會傳回下列錯誤：`SSL error: unregistered scheme`。您可以遵循 從 Windows 連線做為使用 SSL 連線至叢集的替代方法。

如果您的作業系統中未安裝 Amazon Root CA 1，請遵循下列步驟。

安裝憑證

如果您的作業系統未預先安裝`Amazon Root CA 1`憑證，您將需要手動安裝憑證，才能建立與 Aurora DSQL 叢集的安全連線。

Linux 憑證安裝

請依照下列步驟，在 Linux 系統上安裝 Amazon Root CA 憑證。

1. 下載根憑證：

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

2. 將憑證複製到信任存放區：

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

3. 更新 CA 信任存放區：

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

4. 驗證安裝：

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

macOS 憑證安裝

這些憑證安裝步驟是選用的。Linux 憑證安裝 也適用於 macOS。

1. 下載根憑證：

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

2. 將憑證新增至系統金鑰鏈：

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

3. 驗證安裝：

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

使用 SSL/TLS 驗證連線

設定 SSL/TLS 憑證以安全連線至 Aurora DSQL 叢集之前，請確定您有下列先決條件。

- 已安裝 PostgreSQL 第 17 版

- AWS CLI 使用適當的登入資料設定
- Aurora DSQL 叢集端點資訊

從 Linux 連線

1. 產生並設定身分驗證字符：

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

2. 使用系統憑證連線（如果已預先安裝）：

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

3. 或者，使用下載的憑證進行連線：

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

> **Note**
>
> 如需 PGSSLMODE 設定的詳細資訊，請參閱 PostgresQL 17 資料庫連線控制函數文件中的 sslmode。

從 macOS 連線

1. 產生並設定身分驗證字符：

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

2. 使用系統憑證進行連線（如果已預先安裝）：

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

3. 或者，下載根憑證並將其儲存為 root.pem（如果未預先安裝憑證）

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

4. 使用 psql 連線：

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

從 Windows 連線

使用命令提示字元

1. 產生身分驗證字符：

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

2. 設定密碼環境變數：

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

3. 設定 SSL 組態：

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

4. 連線至資料庫：

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

使用 PowerShell

1. 產生並設定身分驗證字符：

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

2. 設定 SSL 組態：

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

3. 連線至資料庫：

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

其他資源

- PostgreSQL SSL 文件
- Amazon Trust Services

Amazon Aurora DSQL 的資料加密

Amazon Aurora DSQL 會加密所有靜態使用者資料。為了增強安全性，此加密使用 AWS Key Management Service (AWS KMS)。此功能協助降低了保護敏感資料所涉及的操作負擔和複雜性。靜態加密可協助您：

- 減少保護敏感資料的操作負擔
- 建置符合嚴格加密合規和法規要求的安全敏感應用程式
- 一律保護加密叢集中的資料，以新增多一層的資料保護
- 遵守組織政策、產業或政府法規，以及合規要求

使用 Aurora DSQL，您可以建置安全敏感的應用程式，以滿足嚴格的加密合規性和法規要求。下列各節說明如何為新的和現有的 Aurora DSQL 資料庫設定加密，並管理您的加密金鑰。

主題

- Aurora DSQL 的 KMS 金鑰類型
- Aurora DSQL 中的靜態加密
- 搭配 Aurora DSQL 使用 AWS KMS 和 資料金鑰
- 授權使用 AWS KMS key 適用於 Aurora DSQL 的
- Aurora DSQL 加密內容
- 監控 Aurora DSQL 與 的互動 AWS KMS
- 建立加密的 Aurora DSQL 叢集
- 移除或更新 Aurora DSQL 叢集的金鑰
- 使用 Aurora DSQL 加密的考量事項

Aurora DSQL 的 KMS 金鑰類型

Aurora DSQL 與 整合 AWS KMS ，以管理叢集的加密金鑰。若要進一步了解金鑰類型和狀態，請參閱《 AWS Key Management Service 開發人員指南》中的AWS Key Management Service 概念。建立新叢集時，您可以選擇下列 KMS 金鑰類型來加密叢集：

AWS 擁有的金鑰

預設加密類型。Aurora DSQL 擁有金鑰，您無需額外付費。當您存取加密的叢集時，Amazon Aurora DSQL 會透明地解密叢集資料。您不需要變更程式碼或應用程式來使用或管理加密的叢集，而且所有 Aurora DSQL 查詢都適用於您的加密資料。

客戶受管金鑰

您可以在 中建立、擁有和管理金鑰 AWS 帳戶。您可以完全控制 KMS 金鑰。需支付 AWS KMS 費用。

使用 AWS 擁有的金鑰 進行靜態加密無需額外費用。不過， AWS KMS 費用適用於客戶受管金鑰。如需詳細資訊，請參閱 AWS KMS 定價頁面。

您可以隨時在這些金鑰類型之間切換。如需金鑰類型的詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》AWS 擁有的金鑰中的客戶受管金鑰和 。

> **Note**
>
> Aurora DSQL 靜態加密可用於可使用 Aurora DSQL 的所有 AWS 區域。

Aurora DSQL 中的靜態加密

Amazon Aurora DSQL 使用 256 位元進階加密標準 (AES-256) 來加密靜態資料。此加密有助於保護您的資料免於未經授權存取基礎 storage。 會 AWS KMS 管理叢集的加密金鑰。您可以使用預設 AWS 擁有的金鑰，或選擇使用您自己的 AWS KMS 客戶受管金鑰。若要進一步了解如何指定和管理 Aurora DSQL 叢集的金鑰，請參閱 建立加密的 Aurora DSQL 叢集和 移除或更新 Aurora DSQL 叢集的金鑰。

主題

- AWS 擁有的金鑰
- 客戶受管金鑰

AWS 擁有的金鑰

Aurora DSQL 預設會使用 加密所有叢集 AWS 擁有的金鑰。這些金鑰每年可免費使用和輪換，以保護您的帳戶資源。您不需要檢視、管理、使用或稽核這些金鑰，因此資料保護不需要採取任何動作。如需的詳細資訊 AWS 擁有的金鑰，請參閱《 AWS Key Management Service 開發人員指南AWS 擁有的金鑰》中的 。

客戶受管金鑰

您可以在 中建立、擁有和管理客戶受管金鑰 AWS 帳戶。您可以完全控制這些 KMS 金鑰，包括其政策、加密資料、標籤和別名。如需管理許可的詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》中的客戶受管金鑰。

當您為叢集層級加密指定客戶受管金鑰時，Aurora DSQL 會使用該金鑰加密叢集及其所有區域資料。為了防止資料遺失和維護叢集存取，Aurora DSQL 需要存取您的加密金鑰。如果您停用客戶受管金鑰、排程刪除金鑰，或具有限制服務存取的政策，叢集的加密狀態會變更為

KMS_KEY_INACCESSIBLE。當 Aurora DSQL 無法存取金鑰時，使用者無法連線至叢集、叢集的加密狀態變更為 KMS_KEY_INACCESSIBLE，且服務無法存取叢集資料。

對於多區域叢集，客戶可以分別設定每個區域的 AWS KMS 加密金鑰，而每個區域叢集會使用自己的叢集層級加密金鑰。如果 Aurora DSQL 無法存取多區域叢集中對等的加密金鑰，則該對等的狀態KMS_KEY_INACCESSIBLE會變成無法使用，無法進行讀取和寫入操作。其他對等繼續正常操作。

> **Note**
>
> 如果 Aurora DSQL 無法存取您的客戶受管金鑰，您的叢集加密狀態會變更為 KMS_KEY_INACCESSIBLE。還原金鑰存取後，服務會在 15 分鐘內自動偵測還原。如需詳細資訊，請參閱叢集閒置。
> 對於多區域叢集，如果長時間遺失金鑰存取，叢集還原時間取決於無法存取金鑰時寫入的資料量。

搭配 Aurora DSQL 使用 AWS KMS 和 資料金鑰

Aurora DSQL 靜態加密功能使用 AWS KMS key 和資料金鑰階層來保護您的叢集資料。

建議您在 Aurora DSQL 中實作叢集之前規劃加密策略。如果您在 Aurora DSQL 中存放敏感或機密資料，請考慮在您的計劃中包含用戶端加密。如此一來，您就能盡量靠近資料來源進行加密，並確保資料在整個生命週期受到保護。

主題

- AWS KMS key搭配 Aurora DSQL 使用
- 搭配 Aurora DSQL 使用叢集金鑰
- 叢集金鑰快取

AWS KMS key搭配 Aurora DSQL 使用

靜態加密可在 下保護您的 Aurora DSQL 叢集 AWS KMS key。根據預設，Aurora DSQL 會使用 AWS 擁有的金鑰，這是在 Aurora DSQL 服務帳戶中建立和管理的多租用戶加密金鑰。但是，您可以在 中的客戶受管金鑰下加密 Aurora DSQL 叢集 AWS 帳戶。您可以為每個叢集選取不同的 KMS 金鑰，即使其參與多區域設定。

當您建立或更新叢集時，請選取叢集的 KMS 金鑰。您可以隨時在 Aurora DSQL 主控台或使用 UpdateCluster操作變更叢集的 KMS 金鑰。切換金鑰的程序不需要停機或降級服務。

> **Important**
>
> Aurora DSQL 僅支援對稱 KMS 金鑰。您無法使用非對稱 KMS 金鑰來加密 Aurora DSQL 叢集。

客戶受管金鑰提供下列優點。

- 您可以建立和管理 KMS 金鑰，包括設定金鑰政策和 IAM 政策來控制對 KMS 金鑰的存取。您可以啟用和停用 KMS 金鑰、啟用和停用自動金鑰輪換，以及於不再使用時刪除 KMS 金鑰。
- 您可以搭配匯入的金鑰材料使用客戶受管金鑰，或是使用位於您所擁有及管理自訂金鑰存放區中的客戶受管金鑰。
- 您可以在 AWS CloudTrail 日誌 AWS KMS 中檢查對 的 Aurora DSQL API 呼叫，以稽核 Aurora DSQL 叢集的加密和解密。

不過， AWS 擁有的金鑰 是免費的，其使用方式不會計入 AWS KMS 資源或請求配額。客戶受管金鑰會針對每個 API 呼叫產生費用，而 AWS KMS 配額會套用至這些金鑰。

搭配 Aurora DSQL 使用叢集金鑰

Aurora DSQL 使用叢集 AWS KMS key 的 來產生和加密叢集的唯一資料金鑰，稱為叢集金鑰。

叢集金鑰會用作金鑰加密金鑰。Aurora DSQL 使用此叢集金鑰來保護用於加密叢集資料的資料加密金鑰。Aurora DSQL 會為叢集中的每個基礎結構產生唯一的資料加密金鑰，但多個叢集項目可能受到相同的資料加密金鑰保護。

若要解密叢集金鑰，Aurora DSQL 會在您第一次存取加密的叢集 AWS KMS 時傳送請求至 。為了讓叢集保持可用，Aurora DSQL 會定期驗證對 KMS 金鑰的解密存取，即使您未主動存取叢集也一樣。

Aurora DSQL 會存放並使用 外部的叢集金鑰和資料加密金鑰 AWS KMS。它使用進階加密標準 (AES) 加密和 256 位元加密金鑰來保護所有金鑰。然後，它會將加密的金鑰與加密的資料一起存放，以便它們可以隨需解密叢集資料。

如果您變更叢集的 KMS 金鑰，Aurora DSQL 會使用新的 KMS 金鑰重新加密現有的叢集金鑰。

叢集金鑰快取

為了避免 AWS KMS 針對每個 Aurora DSQL 操作呼叫 ，Aurora DSQL 會快取記憶體中每個呼叫者的純文字叢集金鑰。如果 Aurora DSQL 在閒置 15 分鐘後收到快取叢集金鑰的請求，它會將新的請求

傳送至 AWS KMS 以解密叢集金鑰。在上次請求解密叢集金鑰之後，此呼叫將擷取對 AWS KMS 或 AWS Identity and Access Management (IAM) AWS KMS key 中 存取政策所做的任何變更。

授權使用 AWS KMS key 適用於 Aurora DSQL 的

如果您使用帳戶中的客戶受管金鑰來保護 Aurora DSQL 叢集，則該金鑰上的政策必須授予 Aurora DSQL 代表您使用它的許可。

您可以完全控制客戶受管金鑰上的政策。Aurora DSQL 不需要額外的授權，即可使用預設值 AWS 擁有的金鑰 來保護您 中的 Aurora DSQL 叢集 AWS 帳戶。

客戶受管金鑰的金鑰政策

當您選取客戶受管金鑰來保護 Aurora DSQL 叢集時，Aurora DSQL 需要 AWS KMS key 代表進行選取之主體使用 的許可。該委託人、使用者或角色必須擁有 AWS KMS key Aurora DSQL 所需的 許可。您可以在金鑰政策或 IAM 政策中提供這些許可。

Aurora DSQL 至少需要客戶受管金鑰的下列許可：

- `kms:Encrypt`
- `kms:Decrypt`
- `kms:ReEncrypt*`（適用於 kms:ReEncryptFrom 和 kms:ReEncryptTo)
- `kms:GenerateDataKey`
- `kms:DescribeKey`

例如，以下範例金鑰政策只會提供必要許可。政策具有下列效果：

- 允許 Aurora DSQL 在密碼編譯操作 AWS KMS key 中使用 ，但僅限於代表帳戶中有權使用 Aurora DSQL 的主體時。如果政策陳述式中指定的委託人沒有使用 Aurora DSQL 的許可，呼叫會失敗，即使它來自 Aurora DSQL 服務。
- `kms:ViaService` 條件金鑰只有在請求來自 Aurora DSQL 時，才允許 許可，代表政策陳述式中列出的委託人。這些委託人無法直接呼叫這些操作。
- 授予 AWS KMS key 管理員 （ 可擔任該`db-team`角色的使用者 ） 對 的唯讀存取權 AWS KMS key

使用範例金鑰政策之前，請將範例主體取代為來自您的實際主體 AWS 帳戶。

```
{
  "Sid": "Enable dsql IAM User Permissions",
```

```
  "Effect": "Allow",
  "Principal": {
    "Service": "dsql.amazonaws.com"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey",
    "kms:Encrypt",
    "kms:ReEncryptFrom",
    "kms:ReEncryptTo"
  ],
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "kms:EncryptionContext:aws:dsql:ClusterId": "w4abucpbwuxx",
      "aws:SourceArn": "arn:aws:dsql:us-east-2:111122223333:cluster/w4abucpbwuxx"
    }
  }
},
{
  "Sid": "Enable dsql IAM User Describe Permissions",
  "Effect": "Allow",
  "Principal": {
    "Service": "dsql.amazonaws.com"
  },
  "Action": "kms:DescribeKey",
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "aws:SourceArn": "arn:aws:dsql:us-east-2:111122223333:cluster/w4abucpbwuxx"
    }
  }
}
```

Aurora DSQL 加密內容

加密內容是一組金鑰/值對，其中包含任意非私密資料。當您在加密資料的請求中包含加密內容時， 會以 AWS KMS 加密方式將加密內容繫結至加密的資料。若要解密資料，您必須傳遞相同的加密內容。

Aurora DSQL 在所有 AWS KMS 密碼編譯操作中使用相同的加密內容。如果您使用客戶受管金鑰來保護 Aurora DSQL 叢集，您可以使用加密內容來識別在稽核記錄和日誌 AWS KMS key 中使用 。它也會以純文字顯示在日誌中，例如 中的日誌 AWS CloudTrail。

加密內容也可以用作政策中授權的條件。

在其對 的請求中 AWS KMS，Aurora DSQL 會使用具有金鑰/值對的加密內容：

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

鍵/值對可識別 Aurora DSQL 正在加密的叢集。金鑰為 `aws:dsql:ClusterId`。值是叢集的識別符。

監控 Aurora DSQL 與 的互動 AWS KMS

如果您使用客戶受管金鑰來保護 Aurora DSQL 叢集，您可以使用 AWS CloudTrail 日誌來追蹤 Aurora DSQL AWS KMS 代表您傳送到 的請求。

展開下列各節，以了解 Aurora DSQL 如何使用 AWS KMS 操作`GenerateDataKey`和 `Decrypt`。

`GenerateDataKey`

當您在叢集上啟用靜態加密時，Aurora DSQL 會建立唯一的叢集金鑰。它會傳送`GenerateDataKey`請求至 AWS KMS ，以指定叢集的 AWS KMS key 。

記錄 `GenerateDataKey` 操作的事件類似於以下範例事件。使用者是 Aurora DSQL 服務帳戶。這些參數包括 的 Amazon Resource Name (ARN) AWS KMS key、需要 256 位元金鑰的金鑰指標，以及識別叢集的加密內容。

```
{
    "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"
}
```

解密

當您存取加密的 Aurora DSQL 叢集時，Aurora DSQL 需要解密叢集金鑰，以便在階層中解密其下方的金鑰。然後，它會解密叢集中的資料。若要解密叢集金鑰，Aurora DSQL 會傳送`Decrypt`請求至 AWS KMS ，以指定叢集 AWS KMS key 的 。

記錄 `Decrypt` 操作的事件類似於以下範例事件。使用者是 中存取叢集的委託 AWS 帳戶 人。這些參數包括加密的叢集金鑰 （ 做為加密文字 Blob) 和可識別叢集的加密內容。 會從加密文字 AWS KMS 衍生 的 AWS KMS key ID。

```
{
  "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"
}
```


建立加密的 Aurora DSQL 叢集

所有 Aurora DSQL 叢集都會靜態加密。根據預設，叢集 AWS 擁有的金鑰 可免費使用 ，或者您可以指定自訂 AWS KMS 金鑰。請依照下列步驟，從 AWS Management Console 或 建立您的加密叢集 AWS CLI。

Console

在 中建立加密叢集 AWS Management Console

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/dsql/ 開啟 Aurora DSQL 主控台。
2. 在主控台左側的導覽窗格中，選擇叢集。
3. 選擇右上角的建立叢集，然後選取單一區域。
4. 在叢集加密設定中，選擇下列其中一個選項。
   - 接受預設設定以使用 AWS 擁有的金鑰 加密，無需額外費用。
   - 選取自訂加密設定（進階）以指定自訂 KMS 金鑰。然後，搜尋或輸入 KMS 金鑰的 ID 或別名。或者，選擇建立 AWS KMS 金鑰以在 AWS KMS 主控台中建立新的金鑰。
5. 選擇 建立叢集。

若要確認叢集的加密類型，請導覽至叢集頁面，然後選取叢集的 ID 以檢視叢集詳細資訊。檢閱叢集設定索引標籤，叢集 KMS 金鑰設定會顯示 Aurora DSQL 預設金鑰，適用於使用 AWS 擁有的金鑰或其他加密類型的金鑰 ID 的叢集。

> **Note**
>
> 如果您選擇擁有和管理自己的金鑰，請務必適當地設定 KMS 金鑰政策。如需範例和詳細資訊，請參閱 the section called “客戶受管金鑰的金鑰政策”。

CLI

建立使用預設 加密的叢集 AWS 擁有的金鑰

- 使用下列命令來建立 Aurora DSQL 叢集。

```
aws dsql create-cluster
```

如下列加密詳細資訊所示，叢集的加密狀態預設為啟用，預設加密類型為 AWS 擁有的金鑰。叢集現在已使用 Aurora DSQL 服務帳戶中的預設 AWS 擁有金鑰加密。

```
"encryptionDetails": {
```

```
  "encryptionType" : "AWS_OWNED_KMS_KEY",
  "encryptionStatus" : "ENABLED"
}
```

建立使用客戶受管金鑰加密的叢集

- 使用下列命令建立 Aurora DSQL 叢集，以客戶受管金鑰的 ID 取代紅色文字的金鑰 ID。

```
aws dsql create-cluster \
--kms-encryption-key d41d8cd98f00b204e9800998ecf8427e
```

如下列加密詳細資訊所示，叢集的加密狀態預設為啟用，加密類型為客戶受管 KMS 金鑰。叢集現在已使用您的 金鑰加密。

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:kms:us-east-1:111122223333:key/
d41d8cd98f00b204e9800998ecf8427e",
  "encryptionStatus" : "ENABLED"
}
```

移除或更新 Aurora DSQL 叢集的金鑰

您可以使用 AWS Management Console 或 AWS CLI 來更新或移除 Amazon Aurora DSQL 中現有叢集上的加密金鑰。如果您移除金鑰但未取代，Aurora DSQL 會使用預設的 AWS 擁有的金鑰。請依照下列步驟，從 Aurora DSQL 主控台或 更新現有叢集的加密金鑰 AWS CLI。

Console

在 中更新或移除加密金鑰 AWS Management Console

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/dsql/ 開啟 Aurora DSQL 主控台。
2. 在主控台左側的導覽窗格中，選擇叢集。
3. 從清單檢視中，尋找並選取您要更新的叢集資料列。
4. 選取動作功能表，然後選擇修改。
5. 在叢集加密設定中，選擇下列其中一個選項來修改加密設定。

- 如果您想要從自訂金鑰切換至 AWS 擁有的金鑰，請取消選取自訂加密設定 （進階） 選項。預設設定將 AWS 擁有的金鑰 免費套用和加密 叢集。
- 如果您想要從一個自訂 KMS 金鑰切換到另一個，或從 切換 AWS 擁有的金鑰 到 KMS 金鑰，如果尚未選取自訂加密設定 （進階） 選項，請選取此選項。然後，搜尋並選取您要使用的金鑰 ID 或別名。或者，選擇建立 AWS KMS 金鑰以在主控台中 AWS KMS 建立新的金鑰。

6. 選擇儲存。

CLI

下列範例示範如何使用 AWS CLI 更新加密的叢集。

使用預設值更新加密叢集 AWS 擁有的金鑰

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key "AWS_OWNED_KMS_KEY"
```

叢集描述EncryptionStatus的 設定為 ENABLED，而 EncryptionType為 AWS_OWNED_KMS_KEY。

```
"encryptionDetails": {
  "encryptionType" : "AWS_OWNED_KMS_KEY",
  "encryptionStatus" : "ENABLED"
}
```

此叢集現在使用 AWS 擁有的金鑰 Aurora DSQL 服務帳戶中的預設值加密。

使用 Aurora DSQL 的客戶受管金鑰更新加密叢集

更新加密叢集，如下列範例所示：

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
```

```
--kms-encryption-key arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-a123-
ab1234a1b234
```

叢集描述EncryptionStatus的 會轉換為 UPDATING，而 EncryptionType是 CUSTOMER_MANAGED_KMS_KEY。Aurora DSQL 透過平台完成傳播新金鑰後，加密狀態將轉換為 ENABLED

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:us-east-1:kms:key/abcd1234-abcd-1234-a123-ab1234a1b234",
  "encryptionStatus" : "ENABLED"
}
```

> **Note**
>
> 如果您選擇擁有和管理自己的金鑰，請務必適當地設定 KMS 金鑰政策。如需範例和詳細資訊，請參閱 the section called “客戶受管金鑰的金鑰政策”。

使用 Aurora DSQL 加密的考量事項

- Aurora DSQL 會加密所有靜態叢集資料。您無法停用此加密，或僅加密叢集中的某些項目。
- AWS Backup 會加密您的備份，以及從這些備份還原的任何叢集。您可以使用 AWS 擁有 AWS Backup 的金鑰或客戶受管金鑰，在 中加密備份資料。
- 已針對 Aurora DSQL 啟用下列資料保護狀態：
  - 靜態資料 - Aurora DSQL 會加密持久性儲存媒體上的所有靜態資料
  - 傳輸中資料 - Aurora DSQL 預設會使用 Transport Layer Security (TLS) 加密所有通訊
- 當您轉換到不同的金鑰時，我們建議您保持啟用原始金鑰，直到轉換完成為止。 AWS 需要原始金鑰來解密資料，才能使用新的金鑰加密您的資料。當叢集的 encryptionStatus 為 ENABLED且您看到新客戶受管金鑰kmsKeyArn的 時，程序即完成。
- 當您停用客戶受管金鑰或撤銷 Aurora DSQL 使用金鑰的存取權時，您的叢集將進入 IDLE 狀態。
- AWS Management Console 和 Amazon Aurora DSQL API 對加密類型使用不同的術語：
  - AWS 主控台 – 在主控台中，您會KMS在使用客戶受管金鑰和使用 DEFAULT時看到 AWS 擁有的金鑰。

  - API – Amazon Aurora DSQL API `CUSTOMER_MANAGED_KMS_KEY`用於客戶受管金鑰和 `AWS_OWNED_KMS_KEY` AWS 擁有的金鑰。
- 如果您在叢集建立期間未指定加密金鑰，Aurora DSQL 會使用 自動加密您的資料 AWS 擁有的金鑰。
- 您可以隨時在 AWS 擁有的金鑰 和客戶受管金鑰之間切換。使用 AWS Management Console AWS CLI或 Amazon Aurora DSQL API 進行此變更。

Aurora DSQL 的身分和存取管理

AWS Identity and Access Management (IAM) 是一種 AWS 服務 ，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可控制誰可以進行身分驗證 （登入） 和授權 （具有許可），以使用 Aurora DSQL 資源。IAM 是您可以免費使用 AWS 服務 的 。

主題

- 目標對象
- 使用身分驗證
- 使用政策管理存取權
- Amazon Aurora DSQL 如何與 IAM 搭配使用
- Amazon Aurora DSQL 的身分型政策範例
- 對 Amazon Aurora DSQL 身分和存取進行故障診斷

目標對象

使用方式 AWS Identity and Access Management (IAM) 會有所不同，取決於您在 Aurora DSQL 中執行的工作。

服務使用者 – 如果您使用 Aurora DSQL 服務來執行任務，您的管理員會為您提供所需的登入資料和許可。當您使用更多 Aurora DSQL 功能來執行工作時，您可能需要額外的許可。了解存取許可的管理方式可協助您向管理員請求正確的許可。如果您無法存取 Aurora DSQL 中的功能，請參閱 對 Amazon Aurora DSQL 身分和存取進行故障診斷。

服務管理員 – 如果您在公司負責 Aurora DSQL 資源，您可能擁有 Aurora DSQL 的完整存取權。您的任務是判斷服務使用者應存取的 Aurora DSQL 功能和資源。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解貴公司如何搭配 Aurora DSQL 使用 IAM，請參閱 Amazon Aurora DSQL 如何與 IAM 搭配使用。

IAM 管理員 – 如果您是 IAM 管理員，建議您進一步了解如何撰寫政策以管理對 Aurora DSQL 的存取。若要檢視您可以在 IAM 中使用的 Aurora DSQL 身分型政策範例，請參閱 Amazon Aurora DSQL 的身分型政策範例。

使用身分驗證

身分驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者身分、IAM 使用者身分或擔任 IAM 角色來驗證 （登入 AWS)。

您可以使用透過身分來源提供的憑證，以聯合身分 AWS 身分身分登入 。 AWS IAM Identity Center (IAM Identity Center) 使用者、您公司的單一登入身分驗證，以及您的 Google 或 Facebook 登入資料，都是聯合身分的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您使用聯合 AWS 身分存取 時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需登入的詳細資訊 AWS，請參閱AWS 登入 《 使用者指南》中的如何登入您的 AWS 帳戶 。

如果您以 AWS 程式設計方式存取 ， AWS 會提供軟體開發套件 (SDK) 和命令列界面 (CLI)，以使用您的登入資料以密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，則必須自行簽署請求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱《IAM 使用者指南》中的適用於 API 請求的AWS Signature 第 4 版。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如， AWS 建議您使用多重要素驗證 (MFA) 來提高帳戶的安全性。如需更多資訊，請參閱《AWS IAM Identity Center 使用者指南》中的多重要素驗證和《IAM 使用者指南》中的 IAM 中的AWS 多重要素驗證。

AWS 帳戶 根使用者

當您建立 時 AWS 帳戶，您會從一個登入身分開始，該身分可完整存取帳戶中的所有 AWS 服務 和資源。此身分稱為 AWS 帳戶 Theroot 使用者，可透過使用您用來建立帳戶的電子郵件地址和密碼登入來存取。強烈建議您不要以根使用者處理日常任務。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，請參閱 IAM 使用者指南中的需要根使用者憑證的任務。

聯合身分

最佳實務是， 要求人類使用者，包括需要管理員存取權的使用者，使用 聯合身分提供者 AWS 服務 來使用臨時憑證來存取 。

聯合身分是您企業使用者目錄、Web 身分提供者、 AWS Directory Service、Identity Center 目錄，或 AWS 服務 是透過身分來源提供的登入資料存取的任何使用者。當聯合身分存取時 AWS 帳戶，它們會擔任 角色，而角色會提供臨時登入資料。

對於集中式存取權管理，我們建議您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中建立使用者和群組，也可以連接並同步到您自己的身分來源中的一組使用者 AWS 帳戶 和群組，以便在所有 和應用程式中使用。如需 IAM Identity Center 的詳細資訊，請參閱 AWS IAM Identity Center 使用者指南中的什麼是 IAM Identity Center？。

IAM 使用者和群組

IAM 使用者是 中的身分 AWS 帳戶 ，具有單一人員或應用程式的特定許可。建議您盡可能依賴臨時憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需更多資訊，請參閱 IAM 使用者指南中的為需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是一種指定 IAM 使用者集合的身分。您無法以群組身分簽署。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的程序變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供臨時憑證。如需更多資訊，請參閱《IAM 使用者指南》中的 IAM 使用者的使用案例。

IAM 角色

IAM 角色是 中具有特定許可 AWS 帳戶 的身分。它類似 IAM 使用者，但不與特定的人員相關聯。若要暫時在 中擔任 IAM 角色 AWS Management Console，您可以從使用者切換至 IAM 角色（主控台）。您可以透過呼叫 AWS CLI 或 AWS API 操作或使用自訂 URL 來擔任角色。如需使用角色的方法詳細資訊，請參閱《IAM 使用者指南》中的擔任角色的方法。

使用臨時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需有關聯合角色的相關資訊，請參閱《IAM 使用者指南》中的為第三方身分提供者 (聯合) 建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的許可集。
- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。

- 跨帳戶存取權：您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的主體) 存取您帳戶的資源。角色是授予跨帳戶存取權的主要方式。不過，對於某些 AWS 服務，您可以將政策直接連接到資源（而不是使用角色做為代理）。如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。
- 跨服務存取 – 有些 AWS 服務 使用其他 中的功能 AWS 服務。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉送存取工作階段 (FAS) – 當您使用 IAM 使用者或角色在其中執行動作時 AWS，您會被視為委託人。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用呼叫 的委託人許可 AWS 服務，結合 AWS 服務 請求向下游服務提出請求。只有在服務收到需要與其他 AWS 服務 或 資源互動才能完成的請求時，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱《轉發存取工作階段》。
  - 服務角色 – 服務角色是服務擔任的 IAM 角色，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以委派許可權給 AWS 服務。
  - 服務連結角色 – 服務連結角色是一種連結至 的服務角色類型 AWS 服務。服務可以擔任代表您執行動作的角色。服務連結角色會出現在您的 中 AWS 帳戶 ，並由服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon EC2 上執行的應用程式 – 您可以使用 IAM 角色來管理在 EC2 執行個體上執行之應用程式的臨時登入資料，以及提出 AWS CLI 或 AWS API 請求。這是在 EC2 執行個體內儲存存取金鑰的較好方式。若要將 AWS 角色指派給 EC2 執行個體並將其提供給其所有應用程式，您可以建立連接至執行個體的執行個體描述檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得臨時憑證。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM 角色來授予許可權給 Amazon EC2 執行個體上執行的應用程式。

使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策是 中的物件，當與身分或資源建立關聯時， AWS 會定義其許可。當委託人（使用者、根使用者或角色工作階段）發出請求時， 會 AWS 評估這些政策。政策中的許可決定是否允許或拒絕請求。大多數政策會以 JSON 文件 AWS 形式存放在 中。如需 JSON 政策文件結構和內容的詳細資訊，請參閱 IAM 使用者指南中的 JSON 政策概觀。

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該政策的使用者可以從 AWS Management Console AWS CLI、 或 AWS API 取得角色資訊。

身分型政策

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管政策是獨立的政策，您可以連接到 中的多個使用者、群組和角色 AWS 帳戶。受管政策包括 AWS 受管政策和客戶受管政策。如需了解如何在受管政策及內嵌政策之間選擇，請參閱《IAM 使用者指南》中的在受管政策和內嵌政策間選擇。

資源型政策

資源型政策是連接到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。委託人可以包含帳戶、使用者、角色、聯合身分使用者或 AWS 服務。

資源型政策是位於該服務中的內嵌政策。您無法在以資源為基礎的政策中使用來自 IAM 的 AWS 受管政策。

存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3 AWS WAF和 Amazon VPC 是支援 ACLs的服務範例。如需進一步了解 ACL，請參閱 Amazon Simple Storage Service 開發人員指南中的存取控制清單 (ACL) 概觀。

其他政策類型

AWS 支援其他較不常見的政策類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限 – 許可範圍是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 `Principal` 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可界限的詳細資訊，請參閱 IAM 使用者指南中的 IAM 實體許可界限。
- 服務控制政策 SCPs) – SCPs是 JSON 政策，可指定 中組織或組織單位 (OU) 的最大許可 AWS Organizations。 AWS Organizations 是一種服務，用於分組和集中管理您企業擁有 AWS 帳戶 的多個 。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 會限制成員帳戶中實體的許可，包括每個實體 AWS 帳戶根使用者。如需 Organizations 和 SCP 的詳細資訊，請參閱《AWS Organizations 使用者指南》中的服務控制政策。
- 資源控制政策 (RCP) - RCP 是 JSON 政策，可用來設定您帳戶中資源的可用許可上限，採取這種方式就不需要更新附加至您所擁有的每個資源的 IAM 政策。RCP 會限制成員帳戶中資源的許可，並可能影響身分的有效許可，包括 AWS 帳戶根使用者，無論它們是否屬於您的組織。如需 Organizations 和 RCPs的詳細資訊，包括 AWS 服務 支援 RCPs 清單，請參閱AWS Organizations《 使用者指南》中的資源控制政策 (RCPs)。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過撰寫程式的方式建立角色或聯合使用者的暫時工作階段時，做為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請參閱 IAM 使用者指南中的工作階段政策。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。若要了解如何 AWS 在涉及多種政策類型時決定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

Amazon Aurora DSQL 如何與 IAM 搭配使用

在您使用 IAM 管理對 Aurora DSQL 的存取之前，請先了解哪些 IAM 功能可與 Aurora DSQL 搭配使用。

您可以搭配 Amazon Aurora DSQL 使用的 IAM 功能

IAM 功能	Aurora DSQL 支援
身分型政策	是

IAM 功能	Aurora DSQL 支援
資源型政策	否
政策動作	是
政策資源	是
政策條件索引鍵	是
ACL	否
ABAC (政策中的標籤)	是
臨時憑證	是
主體許可	是
服務角色	是
服務連結角色	是

若要全面了解 Aurora DSQL 和其他 AWS 服務如何與大多數 IAM 功能搭配使用，請參閱《IAM 使用者指南》中的AWS 與 IAM 搭配使用的 服務。

Aurora DSQL 的身分型政策

支援身分型政策：是

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。您無法在身分型政策中指定主體，因為這會套用至連接的使用者或角色。如要了解您在 JSON 政策中使用的所有元素，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素參考。

Aurora DSQL 的身分型政策範例

若要檢視 Aurora DSQL 身分型政策的範例，請參閱 Amazon Aurora DSQL 的身分型政策範例。

Aurora DSQL 內的資源型政策

支援資源型政策：否

資源型政策是附加到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。委託人可以包含帳戶、使用者、角色、聯合身分使用者或 AWS 服務。

如需啟用跨帳戶存取權，您可以指定在其他帳戶內的所有帳戶或 IAM 實體，做為資源型政策的主體。新增跨帳戶主體至資源型政策，只是建立信任關係的一半。當委託人和資源位於不同位置時 AWS 帳戶，信任帳戶中的 IAM 管理員也必須授予委託人實體（使用者或角色）存取資源的許可。其透過將身分型政策連接到實體來授與許可。不過，如果資源型政策會為相同帳戶中的主體授予存取，這時就不需要額外的身分型政策。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的快帳戶資源存取。

Aurora DSQL 的政策動作

支援政策動作：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 `Action` 元素描述您可以用來允許或拒絕政策中存取的動作。政策動作通常具有與相關聯 AWS API 操作相同的名稱。有一些例外狀況，例如沒有相符的 API 操作的僅限許可動作。也有一些作業需要政策中的多個動作。這些額外的動作稱為相依動作。

政策會使用動作來授予執行相關聯動作的許可。

若要查看 Aurora DSQL 動作的清單，請參閱服務授權參考中的 Amazon Aurora DSQL 定義的動作。

Aurora DSQL 中的政策動作在動作之前使用以下字首：

```
dsql
```

若要在單一陳述式中指定多個動作，請用逗號分隔。

```
"Action": [
```

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

若要檢視 Aurora DSQL 身分型政策的範例，請參閱 Amazon Aurora DSQL 的身分型政策範例。

Aurora DSQL 的政策資源

支援政策資源：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

`Resource` JSON 政策元素可指定要套用動作的物件。陳述式必須包含 `Resource` 或 `NotResource` 元素。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。您可以針對支援特定資源類型的動作 (稱為資源層級許可) 來這麼做。

對於不支援資源層級許可的動作 (例如列出操作)，請使用萬用字元 (*) 來表示陳述式適用於所有資源。

```
"Resource": "*"
```

若要查看 Aurora DSQL 資源類型及其 ARNs，請參閱《服務授權參考》中的 Amazon Aurora DSQL 定義的資源。若要了解您可以使用哪些動作指定每個資源的 ARN，請參閱 Amazon Aurora DSQL 定義的動作。

若要檢視 Aurora DSQL 身分型政策的範例，請參閱 Amazon Aurora DSQL 的身分型政策範例。

Aurora DSQL 的政策條件索引鍵

支援服務特定政策條件金鑰：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

`Condition` 元素 (或 `Condition` 區塊) 可讓您指定使陳述式生效的條件。`Condition` 元素是選用項目。您可以建立使用條件運算子的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。

若您在陳述式中指定多個 Condition 元素，或是在單一 Condition 元素中指定多個索引鍵， AWS 會使用邏輯 AND 操作評估他們。如果您為單一條件索引鍵指定多個值， 會使用邏輯OR操作 AWS 評估條件。必須符合所有條件，才會授與陳述式的許可。

您也可以在指定條件時使用預留位置變數。例如，您可以只在使用者使用其 IAM 使用者名稱標記時，將存取資源的許可授予該 IAM 使用者。如需更多資訊，請參閱 IAM 使用者指南中的 IAM 政策元素：變數和標籤。

AWS 支援全域條件金鑰和服務特定的條件金鑰。若要查看所有 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的AWS 全域條件內容索引鍵。

若要查看 Aurora DSQL 條件索引鍵的清單，請參閱《服務授權參考》中的 Amazon Aurora DSQL 的條件索引鍵。若要了解您可以使用條件金鑰的動作和資源，請參閱 Amazon Aurora DSQL 定義的動作。

若要檢視 Aurora DSQL 身分型政策的範例，請參閱 Amazon Aurora DSQL 的身分型政策範例。

Aurora DSQL 中的 ACLs

支援 ACL：否

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

ABAC 搭配 Aurora DSQL

支援 ABAC (政策中的標籤)：是

屬性型存取控制 (ABAC) 是一種授權策略，可根據屬性來定義許可。在 中 AWS，這些屬性稱為標籤。您可以將標籤連接至 IAM 實體 （使用者或角色） 和許多 AWS 資源。為實體和資源加上標籤是 ABAC 的第一步。您接著要設計 ABAC 政策，允許在主體的標籤與其嘗試存取的資源標籤相符時操作。

ABAC 在成長快速的環境中相當有幫助，並能在政策管理變得繁瑣時提供協助。

如需根據標籤控制存取，請使用 aws:ResourceTag/*key-name*、aws:RequestTag/*key-name* 或 aws:TagKeys 條件索引鍵，在政策的條件元素中，提供標籤資訊。

如果服務支援每個資源類型的全部三個條件金鑰，則對該服務而言，值為 Yes。如果服務僅支援某些資源類型的全部三個條件金鑰，則值為 Partial。

如需 ABAC 的詳細資訊，請參閱《IAM 使用者指南》中的使用 ABAC 授權定義許可。如要查看含有設定 ABAC 步驟的教學課程，請參閱 IAM 使用者指南中的使用屬性型存取控制 (ABAC)。

搭配 Aurora DSQL 使用臨時登入資料

支援臨時憑證：是

當您使用臨時登入資料登入時，有些 AWS 服務 無法運作。如需詳細資訊，包括哪些 AWS 服務 使用臨時登入資料，請參閱《AWS 服務 IAM 使用者指南》中的使用 IAM 的 。

如果您 AWS Management Console 使用使用者名稱和密碼以外的任何方法登入 ，則會使用臨時登入資料。例如，當您 AWS 使用公司的單一登入 (SSO) 連結存取 時，該程序會自動建立臨時登入資料。當您以使用者身分登入主控台，然後切換角色時，也會自動建立臨時憑證。如需切換角色的詳細資訊，請參閱《IAM 使用者指南》中的從使用者切換至 IAM 角色 (主控台)。

您可以使用 AWS CLI 或 AWS API 手動建立臨時登入資料。然後，您可以使用這些臨時登入資料來存取 AWS。 AWS 建議您動態產生臨時登入資料，而不是使用長期存取金鑰。如需詳細資訊，請參閱 IAM 中的暫時性安全憑證。

Aurora DSQL 的跨服務主體許可

支援轉寄存取工作階段 (FAS)：是

當您使用 IAM 使用者或角色在 中執行動作時 AWS，您會被視為委託人。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用呼叫 的委託人許可 AWS 服務，結合 請求向下游服務 AWS 服務 提出請求。只有當服務收到需要與其他 AWS 服務 或 資源互動才能完成的請求時，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱轉發存取工作階段。

Aurora DSQL 的服務角色

支援服務角色：是

服務角色是服務擔任的 IAM 角色，可代您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以委派許可權給 AWS 服務。

> **Warning**
>
> 變更服務角色的許可可能會中斷 Aurora DSQL 功能。只有在 Aurora DSQL 提供指引時，才能編輯服務角色。

Aurora DSQL 的服務連結角色

支援服務連結角色：是

服務連結角色是連結至 的一種服務角色 AWS 服務。服務可以擔任代表您執行動作的角色。服務連結角色會出現在您的 中 AWS 帳戶 ，並由服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

如需為 Aurora DSQL 建立或管理服務連結角色的詳細資訊，請參閱 在 Aurora DSQL 中使用服務連結角色。

Amazon Aurora DSQL 的身分型政策範例

根據預設，使用者和角色沒有建立或修改 Aurora DSQL 資源的許可。他們也無法使用 AWS Management Console、 AWS Command Line Interface (AWS CLI) 或 AWS API 來執行任務。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

如需了解如何使用這些範例 JSON 政策文件建立 IAM 身分型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策 (主控台)。

如需 Aurora DSQL 定義的動作和資源類型的詳細資訊，包括每種資源類型的 ARNs 格式，請參閱《服務授權參考》中的 Amazon Aurora DSQL 的動作、資源和條件金鑰。

主題

- 政策最佳實務
- 使用 Aurora DSQL 主控台
- 允許使用者檢視他們自己的許可

政策最佳實務

身分型政策會判斷您帳戶中的某個人員是否可以建立、存取或刪除 Aurora DSQL 資源。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予您的使用者和工作負載，請使用將許可授予許多常見使用案例的 AWS 受管政策。它們可在您的 中使用 AWS 帳戶。我們建議您定義特定於使用案例 AWS 的客戶受管政策，以進一步減少許可。如需更多資訊，請參閱 IAM 使用者指南中的 AWS 受管政策或任務職能的AWS 受管政策。
- 套用最低權限許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱 IAM 使用者指南中的 IAM 中的政策和許可。

- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。如果透過特定 例如 使用服務動作 AWS 服務，您也可以使用條件來授予其存取權 AWS CloudFormation。如需詳細資訊，請參閱 IAM 使用者指南中的 IAM JSON 政策元素：條件。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您撰寫安全且實用的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM Access Analyzer 驗證政策。
- 需要多重要素驗證 (MFA) – 如果您的案例需要 IAM 使用者或 中的根使用者 AWS 帳戶，請開啟 MFA 以提高安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_configure-api-require.html中的透過 MFA 的安全 API 存取。

如需 IAM 中最佳實務的相關資訊，請參閱 IAM 使用者指南中的 IAM 安全最佳實務。

使用 Aurora DSQL 主控台

若要存取 Amazon Aurora DSQL 主控台，您必須擁有一組最低許可。這些許可必須允許您列出和檢視中 Aurora DSQL 資源的詳細資訊 AWS 帳戶。如果您建立比最基本必要許可更嚴格的身分型政策，則對於具有該政策的實體 (使用者或角色) 而言，主控台就無法如預期運作。

對於僅呼叫 AWS CLI 或 AWS API 的使用者，您不需要允許最低主控台許可。反之，只需允許存取符合他們嘗試執行之 API 操作的動作就可以了。

為了確保使用者和角色仍然可以使用 Aurora DSQL 主控台，也請將 Aurora DSQL `AmazonAuroraDSQLConsoleFullAccess`或 `AmazonAuroraDSQLReadOnlyAccess` AWS 受管政策連接到實體。如需詳細資訊，請參閱《IAM 使用者指南》中的新增許可到使用者。

允許使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視附加到他們使用者身分的內嵌及受管政策。此政策包含在主控台或使用 或 AWS CLI AWS API 以程式設計方式完成此動作的許可。

```
{
    "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": "*"
        }
    ]
}
```

對 Amazon Aurora DSQL 身分和存取進行故障診斷

使用以下資訊來協助您診斷和修正使用 Aurora DSQL 和 IAM 時可能遇到的常見問題。

主題

- 我無權在 Aurora DSQL 中執行動作
- 我未獲得執行 iam:PassRole 的授權
- 我想要允許 以外的人員 AWS 帳戶 存取我的 Aurora DSQL 資源

我無權在 Aurora DSQL 中執行動作

如果您收到錯誤，告知您未獲授權執行動作，您的政策必須更新，允許您執行動作。

當 mateojackson 嘗試使用主控台檢視*my-dsql-cluster*資源的詳細資訊，但沒有 *GetCluster*許可時，會發生下列範例錯誤。

```
User: iam:::user/mateojackson is not authorized to perform: GetCluster on resource: my-dsql-cluster
```

在此情況下，必須更新 mateojackson 使用者的政策，允許使用 *GetCluster* 動作存取 *my-dsql-cluster* 資源。

如需任何協助，請聯絡您的 管理員。您的管理員提供您的簽署憑證。

我未獲得執行 iam:PassRole 的授權

如果您收到錯誤，告知您無權執行 iam:PassRole動作，您的政策必須更新，以允許您將角色傳遞至 Aurora DSQL。

有些 AWS 服務 可讓您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

當名為 的 IAM marymajor 使用者嘗試使用主控台在 Aurora DSQL 中執行動作時，會發生下列範例錯誤。但是，動作請求服務具備服務角色授予的許可。Mary 沒有將角色傳遞至該服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 iam:PassRole 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我想要允許 以外的人員 AWS 帳戶 存取我的 Aurora DSQL 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

如需進一步了解，請參閱以下內容：

- 若要了解 Aurora DSQL 是否支援這些功能，請參閱 Amazon Aurora DSQL 如何與 IAM 搭配使用。
- 若要了解如何提供您擁有 AWS 帳戶 的資源存取權，請參閱《IAM 使用者指南》中的在您擁有 AWS 帳戶 的另一個 中為 IAM 使用者提供存取權。

- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的將存取權提供給第三方 AWS 帳戶 擁有。
- 如需了解如何透過聯合身分提供存取權，請參閱 IAM 使用者指南中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。

在 Aurora DSQL 中使用服務連結角色

Aurora DSQL 使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是直接連結至 Aurora DSQL 的唯一 IAM 角色類型。服務連結角色由 Aurora DSQL 預先定義，並包含服務 AWS 服務 代表 Aurora DSQL 叢集呼叫 所需的所有許可。

服務連結角色可讓您更輕鬆地設定程序，因為您不必手動新增使用 Aurora DSQL 的必要許可。當您建立叢集時，Aurora DSQL 會自動為您建立服務連結角色。只有在刪除所有叢集之後，您才能刪除服務連結角色。這可保護您的 Aurora DSQL 資源，因為您不會不小心移除存取資源所需的許可。

如需關於支援服務連結角色的其他服務資訊，請參閱可搭配 IAM 運作的AWS 服務，尋找 Service-Linked Role (服務連結角色) 欄中顯示為 Yes (是) 的服務。選擇具有連結的是，以檢視該服務的服務連結角色文件。

服務連結角色適用於所有支援的 Aurora DSQL 區域。

Aurora DSQL 的服務連結角色許可

Aurora DSQL 使用名為 的服務連結角色 `AWSServiceRoleForAuroraDsql` – 允許 Amazon Aurora DSQL 代表您建立和管理 AWS 資源。此服務連結角色會連接至下列受管政策：AuroraDsqlServiceLinkedRolePolicy。

> **Note**
>
> 您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。您可能會遇到下列錯誤訊息：`You don't have the permissions to create an Amazon Aurora DSQL service-linked role`。如果您看到此訊息，請確認您已啟用以下許可：
>
> ```
> {
> ```

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

如需詳細資訊，請參閱服務連結角色許可。

建立服務連結角色

您不需要手動建立 AuroraDSQLServiceLinkedRolePolicy 服務連結角色。Aurora DSQL 會為您建立服務連結角色。如果已從您的帳戶刪除 AuroraDSQLServiceLinkedRolePolicy 服務連結角色，Aurora DSQL 會在您建立新的 Aurora DSQL 叢集時建立角色。

編輯服務連結角色

Aurora DSQL 不允許您編輯 AuroraDSQLServiceLinkedRolePolicy 服務連結角色。因為可能有各種實體會參考服務連結角色，所以您無法在建立角色之後變更其名稱。不過，您可以使用 IAM 主控台、AWS Command Line Interface (AWS CLI) 或 IAM API 編輯角色的描述。

刪除服務連結角色

若您不再使用需要服務連結角色的功能或服務，我們建議您刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。

您必須先刪除帳戶中的任何叢集，才能刪除 帳戶的服務連結角色。

您可以使用 IAM 主控台 AWS CLI、 或 IAM API 來刪除服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的建立服務連結角色。

Aurora DSQL 服務連結角色支援的區域

Aurora DSQL 支援在提供服務的所有區域中使用服務連結角色。如需詳細資訊，請參閱AWS 區域與端點。

搭配 Amazon Aurora DSQL 使用 IAM 條件金鑰

當您在 Aurora DSQL 中授予許可時，您可以指定決定許可政策如何生效的條件。以下是如何在 Aurora DSQL 許可政策中使用條件金鑰的範例。

範例 1：授予在特定 中建立叢集的許可 AWS 區域

下列政策授予在美國東部（維吉尼亞北部）和美國東部（俄亥俄）區域中建立叢集的許可。此政策使用資源 ARN 來限制允許的區域，因此 Aurora DSQL 只能在政策的 `Resource`區段中指定該 ARN 時建立叢集。

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

範例 2：授予在特定 AWS 區域中建立多區域叢集的許可

下列政策授予在美國東部（維吉尼亞北部）和美國東部（俄亥俄）區域中建立多區域叢集的許可。此政策使用資源 ARN 來限制允許的區域，因此 Aurora DSQL 只有在政策的 `Resource`區段中指定此 ARN 時，才能建立多區域叢集。請注意，建立多區域叢集還需要每個指定區域中的 `PutWitnessRegion`、 `PutMultiRegionProperties`和 `AddPeerCluster`許可。

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/*"
        ]
      }
    ]
}
```

範例 3：授予許可，以建立具有特定見證區域的多區域叢集

下列政策使用 Aurora DSQL dsql:WitnessRegion條件金鑰，並允許使用者在美國西部（奧勒岡）使用見證區域建立多區域叢集。如果您未指定dsql:WitnessRegion條件，您可以使用任何區域做為見證區域。

```
{
    "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"
```

```
                    ]
                }

            }
        }
    ]
}
```

Amazon Aurora DSQL 中的事件回應

安全性是 的最高優先順序 AWS。作為 AWS Cloud 共同責任模型的一部分， 會 AWS 管理符合最安全敏感組織需求的資料中心、網路和軟體架構。 AWS 負責任何與 Amazon Aurora DSQL 服務本身相關的事件回應。此外，身為 AWS 客戶，您需共同負責維護雲端的安全性。這表示您可以從可存取的 AWS 工具和功能控制您選擇實作的安全性。此外，您有責任在共同責任模型中回應事件。

透過建立符合雲端中執行之應用程式目標的安全基準，您可以偵測可回應的偏差。為了協助您了解事件回應和您的選擇對您公司目標的影響，建議您檢閱下列資源：

- AWS 安全事件回應指南
- AWS 安全性、身分和合規的最佳實務
- AWS 雲端採用架構 (CAF) 的安全觀點白皮書

Amazon GuardDuty 是一種受管威脅偵測服務，會持續監控惡意或未經授權的行為，以協助客戶保護 AWS 帳戶 和工作負載，並在可疑活動可能升級到事件之前識別可疑活動。它會監控活動，例如異常 API 呼叫或潛在未經授權的部署，指出可能遭惡意行為者入侵或偵察帳戶或資源。例如，Amazon GuardDuty 能夠偵測 Amazon Aurora DSQL APIs 中的可疑活動，例如從新位置登入和建立新叢集的使用者。

Amazon Aurora DSQL 的合規驗證

若要了解 是否 AWS 服務 在特定合規計劃的範圍內，請參閱AWS 服務 合規計劃範圍內然後選擇您感興趣的合規計劃。如需一般資訊，請參閱 AWS Compliance Programs。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱在 中下載報告 AWS Artifact。

您使用 時的合規責任 AWS 服務 取決於資料的機密性、您公司的合規目標，以及適用的法律和法規。AWS 提供下列資源來協助合規：

- 安全合規與治理 - 這些解決方案實作指南內容討論了架構考量，並提供部署安全與合規功能的步驟。

- HIPAA 合格服務參考 – 列出 HIPAA 合格服務。並非所有 AWS 服務 都符合 HIPAA 資格。
- AWS 合規資源 – 此工作手冊和指南集合可能適用於您的產業和位置。
- AWS 客戶合規指南 – 透過合規的角度了解共同責任模型。本指南摘要說明跨多個架構 （包括國家標準技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準化組織 (ISO)) 保護 AWS 服務和映射指南至安全控制的最佳實務。
- 《AWS Config 開發人員指南》中的使用規則評估資源 – AWS Config 服務會評估資源組態符合內部實務、產業準則和法規的程度。
- AWS Security Hub – 這 AWS 服務 可讓您全面檢視其中的安全狀態 AWS。Security Hub 使用安全控制，可評估您的 AWS 資源並檢查您的法規遵循是否符合安全業界標準和最佳實務。如需支援的服務和控制清單，請參閱「Security Hub 控制參考」。
- Amazon GuardDuty – 這會監控您的環境是否有可疑和惡意活動，以 AWS 服務 偵測對您 AWS 帳戶、工作負載、容器和資料的潛在威脅。GuardDuty 可滿足特定合規架構所規定的入侵偵測需求，以協助您因應 PCI DSS 等各種不同的合規需求。
- AWS Audit Manager – 這 AWS 服務 可協助您持續稽核 AWS 用量，以簡化您管理風險和符合法規和業界標準的方式。

Amazon Aurora DSQL 中的彈性

AWS 全球基礎設施是以 AWS 區域 和可用區域 (AZ) 為基礎建置。 AWS 區域 提供多個實體隔離且隔離的可用區域，這些可用區域以低延遲、高輸送量和高度備援的網路連接。透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴展能力，均較單一或多個資料中心的傳統基礎設施還高。Aurora DSQL 的設計可讓您利用 AWS 區域基礎設施，同時提供最高的資料庫可用性。根據預設，Aurora DSQL 中的單一區域叢集具有多可用區域可用性，可容忍可能影響完整可用區域存取的主要元件故障和基礎設施中斷。多區域叢集提供多可用區域彈性的所有優點，同時仍提供高度一致的資料庫可用性，即使應用程式用戶端無法存取也 AWS 區域 一樣。

如需 AWS 區域 和可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

除了 AWS 全球基礎設施之外，Aurora DSQL 還提供數種功能，以協助支援您的資料彈性和備份需求。

備份和還原

Aurora DSQL 支援使用 備份和還原 AWS Backup 主控台。您可以為單一區域和多區域叢集執行完整備份和還原。如需詳細資訊，請參閱Amazon Aurora DSQL 的備份和還原。

複寫

根據設計，Aurora DSQL 會將所有寫入交易遞交至分散式交易日誌，並將所有遞交的日誌資料同步複寫至三個AZs的使用者儲存複本。多區域叢集可在讀取和寫入區域之間提供完整的跨區域複寫功能。

指定的見證區域支援僅限交易日誌寫入，且不會耗用儲存體。見證區域沒有端點。這表示見證區域只會儲存加密的交易日誌、不需要管理或組態，而且使用者無法存取。

Aurora DSQL 交易日誌和使用者儲存體會與呈現給 Aurora DSQL 查詢處理器的所有資料一起分發，做為單一邏輯磁碟區。Aurora DSQL 會根據資料庫主索引鍵範圍和存取模式自動分割、合併和複寫資料。Aurora DSQL 會根據讀取存取頻率自動擴展和縮減僅供讀取複本。

叢集儲存複本會分散在多租用戶儲存機群中。如果元件或可用區域受損，Aurora DSQL 會自動重新導向對存活元件的存取權，並以非同步方式修復缺少的複本。Aurora DSQL 修正受損的複本後，Aurora DSQL 會自動將複本新增回儲存規定人數，並將其提供給叢集。

高可用性

根據預設，Aurora DSQL 中的單一區域和多區域叢集為作用中，您不需要手動佈建、設定或重新設定任何叢集。Aurora DSQL 可完全自動化叢集復原，免除傳統主要次要容錯移轉操作的需求。複寫一律是同步的，並在多個可用AZs完成，因此在故障復原期間，不會因為複寫延遲或容錯移轉至非同步次要資料庫而導致資料遺失的風險。

單一區域叢集提供多可用區域備援端點，可自動啟用並行存取，並跨三個AZs提供強大的資料一致性。這表示這三個 AZs 中的任何一個上的使用者儲存複本都會將相同的結果傳回給一或多個讀取器，並且永遠可用於接收寫入。Aurora DSQL 多區域叢集的所有區域皆可使用這種強大的一致性和多可用區域彈性。這表示多區域叢集提供兩個高度一致的區域端點，因此用戶端可以無差別地讀取或寫入至遞交時沒有複寫延遲的任一區域。

Aurora DSQL 為單一區域叢集提供 99.99% 的可用性，為多區域叢集提供 99.999% 的可用性。

Amazon Aurora DSQL 中的基礎設施安全性

Amazon Aurora DSQL 是受管服務，受到安全性、身分和合規最佳實務中所述的 AWS 全球網路安全程序的保護。

您可以使用 AWS 發佈的 API 呼叫，透過網路存取 Aurora DSQL。用戶端必須支援 Transport Layer Security (TLS) 1.2 或更新版本。用戶端也必須支援具備完美轉送私密 (PFS) 的密碼套件，例如臨時 Diffie-Hellman (DHE) 或橢圓曲線臨時 Diffie-Hellman (ECDHE)。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

此外，請求必須使用存取金鑰 ID 和與 IAM 主體相關聯的私密存取金鑰來簽署。或者，您可以透過 AWS Security Token Service (AWS STS) 來產生暫時安全憑證來簽署請求。

使用 管理和連線至 Amazon Aurora DSQL 叢集 AWS PrivateLink

使用 AWS PrivateLink for Amazon Aurora DSQL，您可以在 Amazon Virtual Private Cloud 中佈建介面 Amazon VPC 端點（介面端點）。這些端點可直接透過 Amazon VPC 和 內部部署的應用程式存取 AWS Direct Connect，或在 AWS 區域 Amazon VPC 對等互連的不同 中存取。使用 AWS PrivateLink 和 介面端點，您可以簡化從應用程式到 Aurora DSQL 的私有網路連線。

Amazon VPC 內的應用程式可以使用 Amazon VPC 介面端點存取 Aurora DSQL，而不需要公有 IP 地址。

介面端點由一或多個彈性網路介面 (ENIs) 表示，這些介面會從 Amazon VPC 中的子網路指派私有 IP 地址。透過介面端點對 Aurora DSQL 的請求會保留在 AWS 網路上。如需如何將 Amazon VPC 連線至內部部署網路的詳細資訊，請參閱 AWS Direct Connect 使用者指南和 AWS Site-to-Site VPN VPN 使用者指南。

如需介面端點的一般資訊，請參閱AWS PrivateLink《 使用者指南》中的使用介面 Amazon VPC 端點存取 AWS 服務。

Aurora DSQL 的 Amazon VPC 端點類型

Aurora DSQL 需要兩種不同類型的 AWS PrivateLink 端點。

1. 管理端點 — 此端點用於管理操作，例如 `get`、`delete`、、 `create` `update`和 Aurora DSQL 叢集`list`上的 。請參閱 使用 管理 Aurora DSQL 叢集 AWS PrivateLink。
2. 連線端點 — 此端點用於透過 PostgreSQL 用戶端連線至 Aurora DSQL 叢集。請參閱 使用 連線至 Aurora DSQL 叢集 AWS PrivateLink。

使用 AWS PrivateLink for Aurora DSQL 時的考量事項

Amazon VPC 考量適用於 Aurora DSQL AWS PrivateLink 的 。如需詳細資訊，請參閱《 AWS PrivateLink 指南》中的使用介面 VPC 端點和配額存取 AWS 服務。 AWS PrivateLink

使用 管理 Aurora DSQL 叢集 AWS PrivateLink

您可以使用 AWS Command Line Interface 或 AWS 軟體開發套件 (SDKs) 透過 Aurora DSQL 介面端點管理 Aurora DSQL 叢集。

建立 Amazon VPC 端點

若要建立 Amazon VPC 介面端點，請參閱《 AWS PrivateLink 指南》中的建立 Amazon VPC 端點。

```
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 \
```

若要使用 Aurora DSQL API 請求的預設區域 DNS 名稱，請勿在建立 Aurora DSQL 介面端點時停用私有 DNS。啟用私有 DNS 時，從 Amazon VPC 內對 Aurora DSQL 服務的請求會自動解析為 Amazon VPC 端點的私有 IP 地址，而不是公有 DNS 名稱。啟用私有 DNS 時，Amazon VPC 內發出的 Aurora DSQL 請求會自動解析為您的 Amazon VPC 端點。

如果未啟用私有 DNS，請使用 `--region`和 `--endpoint-url` 參數搭配 AWS CLI 命令，透過 Aurora DSQL 介面端點管理 Aurora DSQL 叢集。

使用端點 URL 列出叢集

在下列範例中，將 和 Amazon VPC 端點 ID 的 DNS 名稱取代 AWS 區域 `us-east-1vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com`為您自己的資訊。

```
aws dsql --region us-east-1 --endpoint-url https://vpce-1a2b3c4d-5e6f.dsql.us-
east-1.vpce.amazonaws.com list-clusters
```

API 操作

如需在 Aurora DSQL 中管理資源的文件，請參閱 Aurora DSQL API 參考。

管理端點政策

透過徹底測試和設定 Amazon VPC 端點政策，您可以協助確保您的 Aurora DSQL 叢集安全、合規，並符合組織的特定存取控制和控管要求。

範例：完整 Aurora DSQL 存取政策

下列政策會透過指定的 Amazon VPC 端點授予所有 Aurora DSQL 動作和資源的完整存取權。

```
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": "*"
        }
      ]
    }'
```

範例：受限制的 Aurora DSQL 存取政策

下列政策僅允許這些 Aurora DSQL 動作。

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

所有其他 Aurora DSQL 動作都會遭到拒絕。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "dsql:CreateCluster",
        "dsql:GetCluster",
        "dsql:ListClusters"
      ],
      "Resource": "*"
    }
```

```
  ]
}
```

使用 連線至 Aurora DSQL 叢集 AWS PrivateLink

設定 AWS PrivateLink 端點並處於作用中狀態後，您可以使用 PostgreSQL 用戶端連線至 Aurora DSQL 叢集。以下連線指示概述建構適當的主機名稱以透過 AWS PrivateLink 端點連線的步驟。

設定 AWS PrivateLink 連線端點

步驟 1：取得叢集的服務名稱

建立端點以 AWS PrivateLink 連線至叢集時，您必須先擷取叢集特定的服務名稱。

AWS CLI

```
aws dsql get-vpc-endpoint-service-name \
--region us-east-1 \
--identifier your-cluster-id
```

回應範例

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

服務名稱包含識別符，例如`dsql-fnh4`範例中的 。建構主機名稱以連線至叢集時，也需要此識別符。

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);
```

步驟 2：建立 Amazon VPC 端點

使用上一個步驟中取得的服務名稱，建立 Amazon VPC 端點。

> **Important**
>
> 以下連線指示僅適用於在啟用 DNS 私有時連線至叢集。建立端點時請勿使用 `--no-private-dns-enabled`旗標，因為這會使下列連線指示無法正常運作。如果您停用私有 DNS，則需要建立自己的萬用字元私有 DNS 記錄，以指向建立的端點。

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
```

回應範例

```
{
    "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

使用 Aurora DSQL APIs端點 URL

```
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);
```

使用連線 AWS PrivateLink 端點連線至 Aurora DSQL 叢集

設定 AWS PrivateLink 端點並處於作用中狀態（檢查 `State`是否為 `available`) 後，您可以使用 PostgreSQL 用戶端連線至 Aurora DSQL 叢集。如需有關使用 AWS SDKs的說明，您可以遵循使用 Aurora DSQL 進行程式設計中的指南。您必須變更叢集端點以符合主機名稱格式。

建構主機名稱

透過 連線的主機名稱與公有 DNS 主機名稱 AWS PrivateLink 不同。您需要使用下列元件來建構它。

1. `Your-cluster-id`
2. 來自服務名稱的服務識別符。例如：`dsql-fnh4`
3. 的 AWS 區域

使用下列格式：*`cluster-id`*.*`service-identifier`*.*`region`*.on.aws

範例：使用 PostgreSQL 的連線

```
# 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
```

使用 AWS PrivateLink來疑難排解問題

常見問題與解決方案

下表列出 AWS PrivateLink 與 Aurora DSQL 相關的常見問題和解決方案。

問題	可能的原因	解決方案
連線逾時	安全群組未正確設定	使用 Amazon VPC Reachability Analyzer 確保您的聯網設定允許連接埠 5432 上的流量。
DNS 解析失敗	未啟用私有 DNS	確認已在啟用私有 DNS 的情況下建立 Amazon VPC 端點。
身分驗證失敗	登入資料不正確或權杖過期	產生新的身分驗證字符並驗證使用者名稱。
找不到服務名稱	不正確的叢集 ID	擷取服務名稱 AWS 區域 時，請再次檢查您的叢集 ID 和 。

相關資源

如需詳細資訊，請參閱下列資源：

- Amazon Aurora DSQL 使用者指南
- AWS PrivateLink 文件
- 透過 存取 AWS 服務 AWS PrivateLink

Amazon Aurora DSQL 中的組態和漏洞分析

AWS 處理基本安全任務，例如訪客作業系統 (OS) 和資料庫修補、防火牆組態和災難復原。這些程序已由適當的第三方進行檢閱並認證。如需詳細資訊，請參閱以下 資源：

- 共同的責任模型
- Amazon Web Services：安全程序概觀 (白皮書)

預防跨服務混淆代理人

混淆代理人問題屬於安全性問題，其中沒有執行動作許可的實體可以強制具有更多許可的實體執行該動作。在 中 AWS，跨服務模擬可能會導致混淆代理人問題。在某個服務 (呼叫服務) 呼叫另一個服務 (被呼叫服務) 時，可能會發生跨服務模擬。可以操縱呼叫服務來使用其許可，以其不應有存取許可的方式對其他客戶的資源採取動作。為了預防這種情況， AWS 提供的工具可協助您保護所有服務的資料，而這些服務主體已獲得您帳戶中資源的存取權。

我們建議在資源政策中使用 aws:SourceArn和 aws:SourceAccount全域條件內容金鑰，以限制 Amazon Aurora DSQL 為資源提供其他服務的許可。如果您想要僅允許一個資源與跨服務存取相關聯，則請使用 aws:SourceArn。如果您想要允許該帳戶中的任何資源與跨服務使用相關聯，請使用 aws:SourceAccount。

防範混淆代理人問題的最有效方法是使用 aws:SourceArn 全域條件內容索引鍵，以及資源的完整 ARN。如果不知道資源的完整 ARN，或者如果您指定了多個資源，請使用 aws:SourceArn 全域內容條件索引鍵搭配萬用字元 (*) 來表示 ARN 的未知部分。例如 arn:aws:*servicename*:*:*123456789012*:*。

如果 aws:SourceArn 值不包含帳戶 ID (例如 Amazon S3 儲存貯體 ARN)，您必須使用這兩個全域條件內容索引鍵來限制許可。

aws:SourceArn 的值必須是 ResourceDescription。

下列範例示範如何在 Aurora DSQL 中使用 aws:SourceArn和 aws:SourceAccount全域條件內容金鑰，以防止混淆代理人問題。

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"
      }
    }
  }
}
```

Aurora DSQL 的安全最佳實務

Aurora DSQL 提供許多安全功能，供您在開發和實作自己的安全政策時考慮。以下最佳實務為一般準則，並不代表完整的安全解決方案。這些最佳實務可能不適用或無法滿足您的環境需求，因此請將其視為實用建議就好，而不要當作是指示。

主題

- Aurora DSQL 的 Detective 安全最佳實務
- Aurora DSQL 的預防性安全最佳實務

Aurora DSQL 的 Detective 安全最佳實務

除了以下安全使用 Aurora DSQL 的方式之外，請參閱 中的安全性 AWS Well-Architected Tool ，以了解雲端技術如何改善您的安全性。

Amazon CloudWatch 警示

使用 Amazon CloudWatch 警示，您可在自己指定的一段時間內監看單一指標。如果指標超過指定的閾值，則會傳送通知至 Amazon SNS 主題或 AWS Auto Scaling 政策。CloudWatch 警示不會因為處於特定狀態而叫用動作。必須是狀態已變更並維持了所指定的時間長度，才會呼叫動作。

標記您的 Aurora DSQL 資源以進行識別和自動化

您可以將中繼資料以標籤形式指派給 AWS 資源。每個標記都是由客戶定義金鑰和選用值組成的簡單標籤，能夠更輕鬆地管理、搜尋和篩選資源。

標記允許實現分組控制。雖然標籤不具固有類型，但能讓您依用途、擁有者、環境或其他條件分類資源。下列是一些範例：

- 安全性：用於確定加密等需求。

- 機密性：資源支援的特定資料機密等級識別符。
- 環境：用來區分開發、測試和生產基礎設施。

您可以將中繼資料以標籤形式指派給 AWS 資源。每個標記都是由客戶定義金鑰和選用值組成的簡單標籤，能夠更輕鬆地管理、搜尋和篩選資源。

標記允許實現分組控制。雖然標籤不具固有類型，但能讓您依用途、擁有者、環境或其他條件分類資源。下列是一些範例。

- 安全性 – 用來判斷加密等需求。
- 機密性 – 資源支援的特定資料機密性層級的識別符。
- 環境 – 用來區分開發、測試和生產基礎設施。

如需詳細資訊，請參閱標記 AWS 資源的最佳實務。

Aurora DSQL 的預防性安全最佳實務

除了以下安全使用 Aurora DSQL 的方式之外，請參閱 中的安全性 AWS Well-Architected Tool ，以了解雲端技術如何改善您的安全性。

使用 IAM 角色來驗證對 Aurora DSQL 的存取。

AWS 服務 存取 Aurora DSQL 的使用者、應用程式和其他 必須在 AWS API 和 AWS CLI 請求中包含有效的 AWS 登入資料。您不應將 AWS 登入資料直接存放在應用程式或 EC2 執行個體中。這些是不會自動輪換的長期登入資料。如果這些登入資料遭到入侵，會對業務產生重大影響。IAM 角色可讓您取得可用來存取 AWS 服務 和資源的臨時存取金鑰。

如需詳細資訊，請參閱Aurora DSQL 的身分驗證和授權。

針對 Aurora DSQL 基礎授權使用 IAM 政策。

當您授予許可時，您可以決定誰取得許可、他們取得許可的 Aurora DSQL API 操作，以及您想要在這些資源上允許的特定動作。對降低錯誤或惡意意圖所引起的安全風險和影響而言，實作最低權限是其中關鍵。

將許可政策連接至 IAM 角色，並授予在 Aurora DSQL 資源上執行操作的許可。也提供 IAM 實體的許可界限，可讓您設定身分型政策可授予 IAM 實體的最大許可。

與 的根使用者最佳實務 AWS 帳戶類似，請勿使用 Aurora DSQL 中的 admin角色來執行日常操作。反之，我們建議您建立自訂資料庫角色來管理和連線至您的叢集。如需詳細資訊，請參閱存取 Aurora DSQL 和了解 Aurora DSQL 的身分驗證和授權。

`verify-full` 在生產環境中使用 。

此設定會驗證伺服器憑證是否由信任的憑證授權單位簽署，以及伺服器主機名稱是否與憑證相符。

更新您的 PostgreSQL 用戶端

定期將 PostgreSQL 用戶端更新為最新版本，以受益於安全性改善。建議使用 PostgreSQL 第 17 版。

在 Aurora DSQL 中標記資源

在 中 AWS，標籤是您定義並與叢集等 Aurora DSQL 資源建立關聯的使用者定義鍵值對。標籤是選擇性的。如果您提供金鑰，則值為選用。

您可以使用 AWS Management Console AWS CLI、 或 AWS SDKs 來新增、列出和刪除 Aurora DSQL 叢集上的標籤。您可以使用 AWS 主控台在建立叢集期間和之後新增標籤。若要在使用 建立叢集後標記叢集， AWS CLI 請使用 `TagResource`操作。

使用名稱標記叢集

Aurora DSQL 會使用指派為 Amazon Resource Name (ARN) 的全域唯一識別符來建立叢集。如果您想要為叢集指派易於使用的名稱，建議您使用標籤。

如果您使用 Aurora DSQL 主控台建立主控台，Aurora DSQL 會自動建立標籤。此標籤的索引鍵為 Name，而自動產生的值代表叢集的名稱。此值是可設定的，因此您可以為叢集指派更易記的名稱。如果叢集具有具有關聯值的名稱標籤，您可以在 Aurora DSQL 主控台中看到該值。

標記需求

標籤均擁有以下要求：

- 索引鍵字首不能是 `aws:`。
- 索引鍵在標籤集內必須是唯一的。
- 索引鍵必須介於 1 到 128 個允許的字元之間。
- 值必須介於 0 到 256 個允許的字元之間。
- 值在每個標籤集中不需要是唯一的。
- 索引鍵和值允許的字元包括字母、數字、空格和下列任何符號：_ . : / = + - @。
- 金鑰和值會區分大小寫。

標記用量備註

在 Aurora DSQL 中使用標籤時，請考慮下列事項。

- 使用 AWS CLI 或 Aurora DSQL API 操作時，請務必為要使用的 Aurora DSQL 資源提供 Amazon Resource Name (ARN)。如需詳細資訊，請參閱 Aurora DSQL 資源的 Amazon Resource Name (ARNs) 格式。
- 每個資源皆有一個標籤集，此為指派給該資源之一或多個標籤的集合。
- 每個資源每個標籤集最多可擁有 50 個標籤。
- 如果您刪除資源，任何關聯的標籤也會遭到刪除。
- 您可以在建立資源時新增標籤，您可以使用下列 API 操作來檢視和修改標籤：`TagResource`、`UntagResource`和 `ListTagsForResource`。
- 您可以搭配 IAM 政策使用標籤。您可以使用它們來管理對 Aurora DSQL 叢集的存取，以及控制哪些動作可以套用至這些資源。若要進一步了解，請參閱使用標籤控制對 AWS 資源的存取。
- 您可以將標籤用於各種其他活動 AWS。若要進一步了解，請參閱常見標記策略。

使用 Amazon Aurora DSQL 的考量事項

當您使用 Amazon Aurora DSQL 時，請考慮下列行為。如需 PostgreSQL 相容性和支援的詳細資訊，請參閱 Aurora DSQL 中的 SQL 功能相容性。如需配額和限制，請參閱 Amazon Aurora DSQL 中的叢集配額和資料庫限制。

- Aurora DSQL 未在大型資料表的交易逾時之前完成`COUNT(*)`操作。若要從系統目錄擷取資料表列計數，請參閱在 Aurora DSQL 中使用系統資料表和命令。
- 呼叫 的驅動程式`PG_PREPARED_STATEMENTS`可能會為叢集提供快取預備陳述式的不一致檢視。對於相同的叢集和 IAM 角色，每個連線可能會看到超過預期的預備陳述式數量。Aurora DSQL 不會保留您準備的陳述式名稱。
- 在極少數的多區域連結叢集受損案例中，交易遞交可用性恢復可能需要比預期更長的時間。一般而言，自動化叢集復原操作可能會導致暫時性並行控制或連線錯誤。在大多數情況下，您只會看到工作負載百分比的效果。當您看到這些傳輸錯誤時，請重試交易或重新與您的用戶端連線。
- 有些 SQL 用戶端，例如 Datagrip，對系統中繼資料進行廣泛呼叫，以填入結構描述資訊。Aurora DSQL 不支援所有此資訊並傳回錯誤。此問題不會影響 SQL 查詢功能，但可能會影響結構描述顯示。
- 管理員角色具有一組與資料庫管理任務相關的許可。根據預設，這些許可不會延伸至其他使用者建立的物件。管理員角色無法將這些使用者建立的物件許可授予或撤銷給其他使用者。管理員使用者可以授予自己任何其他角色，以取得這些物件的必要許可。

Amazon Aurora DSQL 中的叢集配額和資料庫限制

下列各節說明 Aurora DSQL 的叢集配額和資料庫限制。

叢集配額

您的 在 Aurora DSQL 中 AWS 帳戶 具有下列叢集配額。若要請求增加特定內單一區域和多區域叢集的服務配額 AWS 區域，請使用 Service Quotas 主控台頁面。如需提高其他配額，請聯絡 AWS 支援。

描述	預設值限制	可設定？	Aurora DSQL 錯誤代碼
每個 的單一區域叢集上限 AWS 帳戶	20 個叢集	是	API 錯誤碼 ServiceQuotaExceededException :
每個 的多區域叢集上限 AWS 帳戶	5 個叢集	是	API 錯誤碼 ServiceQuotaExceededException :
每個叢集的最大儲存空間	10 TiB 預設限制，最多 128 TiB，並核准增加限制	是	DISK_FULL(53100)
每個叢集的最大連線數	10，000 個連線	是	TOO_MANY_CONNECTIONS(53300)
每個叢集的最大連線速率	每秒 100 個連線	否	CONFIGURED_LIMIT_EXCEEDED(53400)
每個叢集的最大連線高載容量	1，000 個連線	否	無錯誤碼
並行還原任務上限	4	否	無錯誤碼

描述	預設值限制	可設定？	Aurora DSQL 錯誤代碼
連線補充率	每秒 100 個連線	否	無錯誤碼

Aurora DSQL 中的資料庫限制

下表說明 Aurora DSQL 中的資料庫限制。

描述	預設值限制	可設定？	Aurora DSQL 錯誤代碼	錯誤訊息
主索引鍵中使用的資料欄合併大小上限	1 KiB	否	54000	ERROR: key size too large
次要索引中資料欄的合併大小上限	1 KiB	否	54000	ERROR: key size too large
資料表中資料列的大小上限	2 MiB	否	54000	ERROR: maximum row size excee
不屬於索引的資料欄大小上限	1 MiB	否	54000	ERROR: maximum column size ex
主索引鍵或輔助索引中的資料欄數目上限	8	否	54011	ERROR: more than 8 column key are not supported
資料表中的資料欄數目上限	255	否	54011	ERROR: tables can have at mos
資料表中的索引數目上限	24	否	54000	ERROR: more than 24 indexes p allowed

描述	預設值限制	可設定？	Aurora DSQL 錯誤代碼	錯誤訊息
寫入交易中修改的所有資料大小上限	10 MiB	否	54000	ERROR: transaction size limit DETAIL: Current transaction s 10mb
交易區塊中可變更的資料表和索引資料列數目上限	每筆交易 3，000 列。請參閱 PostgreSQL 相容性的 Aurora DSQL 考量事項。	否	54000	ERROR: transaction row limit
查詢操作可使用的記憶體基本數量上限	每筆交易 128 MiB	否	53200	ERROR: query requires too muc out of memory.
資料庫中定義的結構描述數目上限	10	否	54000	ERROR: more than 10 schemas n
資料庫中的資料表數目上限	1，000 個資料表	否	54000	ERROR: creating more than 100 allowed
叢集中的資料庫數目上限	1	否	無錯誤碼	ERROR: unsupported statement
交易時間上限	5 分鐘	否	54000	ERROR: transaction age limit exceeded
最大連線持續時間	60 分鐘	否	無錯誤碼	沒有錯誤訊息

描述	預設值限制	可設定？	Aurora DSQL 錯誤代碼	錯誤訊息
資料庫中的檢視數目上限	5,000	否	54000	ERROR: creating more than 500 allowed
檢視定義大小上限	2 MiB	否	54000	ERROR: view definition too la

如需 Aurora DSQL 特定的資料類型限制，請參閱 Aurora DSQL 中支援的資料類型。

Aurora DSQL API 參考

除了 AWS Management Console 和 AWS Command Line Interface (AWS CLI)，Aurora DSQL 還提供 API 界面。您可以使用 API 操作來管理 Aurora DSQL 中的資源。

如需依字母排序的 API 操作清單，請參閱動作。

如需依字母排序的資料類型清單，請參閱資料類型。

如需常用查詢參數的清單，請參閱常用參數。

如需錯誤碼的說明，請參閱常見錯誤。

如需 的詳細資訊 AWS CLI，請參閱 Aurora DSQL 的 AWS Command Line Interface 參考。

故障診斷 Aurora DSQL 中的問題

> **Note**
>
> 下列主題提供您在使用 Aurora DSQL 時可能遇到的錯誤和問題的疑難排解建議。如果您發現此處未列出的問題，請聯絡 Support AWS

主題

- 對連線錯誤進行故障診斷
- 對身分驗證錯誤進行故障診斷
- 對授權錯誤進行故障診斷
- 故障診斷 SQL 錯誤
- 對 OCC 錯誤進行故障診斷
- 故障診斷 SSL/TLS 連線

對連線錯誤進行故障診斷

錯誤：無法辨識的 SSL 錯誤代碼：6

原因：您可能使用的 psql 版本早於 14 版，不支援伺服器名稱指示 (SNI)。連線至 Aurora DSQL 時需要 SNI。

您可以使用 檢查您的用戶端版本`psql --version`。

錯誤：NetworkUnreachable

連線嘗試期間的`NetworkUnreachable`錯誤可能表示您的用戶端不支援 IPv6 連線，而不是發出實際網路問題的訊號。由於 PostgreSQL 用戶端如何處理雙堆疊連線，此錯誤通常發生在IPv4-only執行個體上。當伺服器支援雙堆疊模式時，這些用戶端會先將主機名稱解析為 IPv4 和 IPv6 地址。他們會先嘗試 IPv4 連線，然後在初始連線失敗時嘗試 IPv6。如果您的系統不支援 IPv6，您會看到一般`NetworkUnreachable`錯誤，而不是清晰的「不支援 IPv6」訊息。

對身分驗證錯誤進行故障診斷

使用者 "..." 的 IAM 身分驗證失敗

當您產生 Aurora DSQL IAM 身分驗證字符時，您可以設定的最長持續時間為 1 週。一週後，您無法使用該字符進行身分驗證。

此外，如果您擔任的角色已過期，Aurora DSQL 會拒絕您的連線請求。例如，如果您嘗試連接臨時 IAM 角色，即使您的身分驗證字符尚未過期，Aurora DSQL 將拒絕連接請求。

若要進一步了解 IAM 如何使用 Aurora DSQL，請參閱了解 Aurora DSQL 和 Aurora DSQL 中的身分驗證和授權。 AWS Identity and Access Management

呼叫 GetObject 操作時發生錯誤 (InvalidAccessKeyId)：您提供的 AWS 存取金鑰 ID 不存在於我們的記錄中

IAM 已拒絕您的請求。如需詳細資訊，請參閱為什麼要簽署請求。

IAM 角色 <role> 不存在

Aurora DSQL 找不到您的 IAM 角色。如需詳細資訊，請參閱 IAM 角色。

IAM 角色必須看起來像 IAM ARN

如需詳細資訊，請參閱 IAM 識別符 - IAM ARNs。

對授權錯誤進行故障診斷

不支援角色 <角色 >

Aurora DSQL 不支援 GRANT操作。請參閱 Aurora DSQL 中支援的 PostgreSQL 命令子集。

無法使用角色 <role> 建立信任

Aurora DSQL 不支援 GRANT操作。請參閱 Aurora DSQL 中支援的 PostgreSQL 命令子集。

角色 <role> 不存在

Aurora DSQL 找不到指定的資料庫使用者。請參閱授權自訂資料庫角色以連線至叢集。

錯誤：以角色 <role> 授予 IAM 信任的許可遭拒

若要授予資料庫角色的存取權，您必須使用 管理員角色連線到叢集。若要進一步了解，請參閱授權資料庫角色以在資料庫中使用 SQL。

錯誤：角色 <role> 必須具有 LOGIN 屬性

您建立的任何資料庫角色都必須具有 LOGIN許可。

若要解決此錯誤，請確定您已使用 LOGIN許可建立 PostgreSQL 角色。如需詳細資訊，請參閱 PostgreSQL 文件中的 CREATE ROLE 和 ALTER ROLE。

錯誤：無法捨棄角色 <role>，因為某些物件相依於它

如果您捨棄具有 IAM 關係的資料庫角色，Aurora DSQL 會傳回錯誤，直到您使用 撤銷關係為止AWS IAM REVOKE。若要進一步了解，請參閱撤銷授權。

故障診斷 SQL 錯誤

錯誤：不支援

Aurora DSQL 不支援所有 PostgreSQL 型方言。若要了解支援的內容，請參閱 Aurora DSQL 中支援的 PostgreSQL 功能。

錯誤：唯讀交易中的 SELECT FOR UPDATE 是無操作

您正在嘗試唯讀交易中不允許的操作。若要進一步了解，請參閱了解 Aurora DSQL 中的並行控制。

錯誤：請**CREATE INDEX ASYNC**改用

若要在具有現有資料列的資料表上建立索引，您必須使用 CREATE INDEX ASYNC命令。若要進一步了解，請參閱在 Aurora DSQL 中非同步建立索引。

對 OCC 錯誤進行故障診斷

OC000「ERROR：變動與另一個交易衝突，視需要重試」

OC001「ERROR：結構描述已由另一個交易更新，請視需要重試」

您的 PostgreSQL 工作階段具有結構描述目錄的快取副本。該快取複本在載入時有效。讓我們呼叫時間 T1 和版本 V1。

另一個交易會在 T2 時間更新目錄。讓我們呼叫此 V2。

當原始工作階段嘗試在 T2 時從儲存體讀取時，它仍然使用目錄版本 V1。Aurora DSQL 的儲存層拒絕請求，因為 T2 的最新目錄版本是 V2。

當您在原始工作階段的 T3 時間重試時，Aurora DSQL 會重新整理目錄快取。T3 的交易使用目錄 V2。只要自 T2 時間以來沒有其他目錄變更發生，Aurora DSQL 就會完成交易。

故障診斷 SSL/TLS 連線

SSL 錯誤：憑證驗證失敗

此錯誤表示用戶端無法驗證伺服器的憑證。請確定：

1. Amazon Root CA 1 憑證已正確安裝。如需如何驗證和安裝此憑證的說明為 Aurora DSQL 連線設定 SSL/TLS 憑證，請參閱 。
2. PGSSLROOTCERT 環境變數指向正確的憑證檔案。
3. 憑證檔案具有正確的許可。

無法辨識的 SSL 錯誤碼：6

低於版本 14 的 PostgreSQL 用戶端會發生此錯誤。將您的 PostgreSQL 用戶端升級至版本 17 以解決此問題。

SSL 錯誤：未註冊的配置 (Windows)

這是使用系統憑證時 Windows psql 用戶端的已知問題。使用 從 Windows 連線說明中所述的下載憑證檔案方法。

Amazon Aurora DSQL 使用者指南的文件歷史記錄

下表說明 Aurora DSQL 的文件版本。

變更	描述	日期
Amazon Aurora DSQL 的一般可用性 (GA)	Amazon Aurora DSQL 現在已全面推出，並新增對 CloudWatch 監控、增強型資料保護功能和 AWS Backup 整合的支援。如需詳細資訊，請參閱使用 CloudWatch 監控 Aurora DSQL、Amazon Aurora DSQL 的備份和還原，以及 Amazon Aurora DSQL 的資料加密。	2025 年 5 月 27 日
AmazonAuroraDSQLFullAccess 更新	新增為 Aurora DSQL 叢集執行備份和還原操作的功能，包括啟動、停止和監控任務。它還新增了使用客戶受管 KMS 金鑰進行叢集加密的功能。如需詳細資訊，請參閱 AmazonAuroraDSQLFullAccess 和在 Aurora DSQL 中使用服務連結角色。	2025 年 5 月 21 日
AmazonAuroraDSQLConsoleFullAccess 更新	新增透過 執行 Aurora DSQL 叢集備份和還原操作的功能 AWS Console Home。這包括啟動、停止和監控任務。它還支援使用客戶管理的 KMS 金鑰進行叢集加密和啟動 AWS CloudShell。如需詳細資訊，請參閱 AmazonAuroraDSQLConsoleFullAccess	2025 年 5 月 21 日

	和在 Aurora DSQL 中使用服務連結角色。	
AmazonAuroraDSQLReadOnlyAccess 更新	包括透過 Aurora DSQL 連線至 Aurora DSQL 叢集時判斷正確 VPC AWS PrivateLink 端點服務名稱的能力，可為每個儲存格建立唯一的端點，因此此 API 有助於確保您可以識別叢集的正確端點，並避免連線錯誤。如需詳細資訊，請參閱 Aurora DSQL 中的 AmazonAuroraDSQLReadOnlyAccess 和使用服務連結角色。	2025 年 5 月 13 日
AmazonAuroraDSQLFullAccess 更新	政策會新增四個新許可，以跨多個 建立和管理資料庫叢集 AWS 區域：`PutMultiRegionProperties` 、`AddPeerCluster` 、 `PutWitnessRegion` 和 `RemovePeerCluster` 。這些許可包括資源層級控制項和條件索引鍵，讓您可以控制可以修改哪些叢集使用者。此政策也會新增 `GetVpcEndpointServiceName` 許可，協助您透過 連線至 Aurora DSQL 叢集 AWS PrivateLink。如需詳細資訊，請參閱 AmazonAuroraDSQLConsoleFullAccess 和在 Aurora DSQL 中使用服務連結角色。	2025 年 5 月 13 日

AmazonAuroraDSQLConsoleFullAccess 更新	將新許可新增至 Aurora DSQL，以支援多區域叢集管理和 VPC 端點連線。新的許可包括：`PutMultiRegionProperties` `PutWitnessRegion` `AddPeerCluster` `RemovePeerCluster` `GetVpcEndpointServiceName` 。SeeAmazonAuroraDSQLConsoleFullAccess和在 Aurora DSQL 中使用服務連結角色。	2025 年 5 月 13 日
AuroraDsqlServiceLinkedRolePolicy 更新	新增將指標發佈至 的功能，`AWS/AuroraDSQL` 以及將 `AWS/Usage` `CloudWatch` 命名空間發佈至政策的功能。這可讓相關聯的服務或角色將更全面的用量和效能資料傳送到 CloudWatch 環境。如需詳細資訊，請參閱AuroraDsqlServiceLinkedRolePolicy 和在 Aurora DSQL 中使用服務連結角色。	2025 年 5 月 8 日

AWS PrivateLink for Amazon Aurora DSQL	Aurora DSQL 現在支援 AWS PrivateLink。透過 AWS PrivateLink，您可以使用界面 Amazon VPC 端點和私有 IP 地址，簡化虛擬私有雲端 (VPCs)、Aurora DSQL 和內部部署資料中心之間的私有網路連線。如需詳細資訊，請參閱使用 管理和連線至 Amazon Aurora DSQL 叢集 AWS PrivateLink。	2025 年 5 月 8 日
初始版本	Amazon Aurora DSQL 使用者指南的初始版本。	2024 年 12 月 3 日

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。