aws

Aurora 使用者指南

Amazon Aurora

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

Amazon Aurora: Aurora 使用者指南

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

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能隸屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

Table of Contents

什麼是 Aurora？ 1
Amazon RDS 共同的責任模型 2
Amazon Aurora 如何搭配 Amazon RDS 運作 2
Aurora 資料庫叢集 3
Aurora 版本 5
支援的資料庫引擎 5
指定 Aurora 版本 5
Aurora 版本控制 6
Aurora 資料庫叢集升級 11
Aurora 版本支援 13
區域與可用區域 15
AWS 區域 16
可用區域 23
資料庫叢集的本機時區 24
依區域和引擎支援的 Aurora 功能 30
資料表慣例 31
藍/綠部署 31
Aurora 叢集組態 31
資料庫活動串流 32
將叢集資料匯出至 Amazon S3 39
將快照資料匯出至 Amazon S3 41
Aurora 全球資料庫 41
IAM 資料庫身分驗證 50
Kerberos 身分驗證 50
Aurora Machine Learning 60
Performance Insights 68
零 ETL 整合 78
RDS Proxy 84
Secrets Manager 整合 93
Aurora Serverless v2 94
Aurora Serverless v1 100
RDS 資料 API 104
零停機時間修補 (ZDP) 114
Aurora PostgreSQL Limitless Database 114

引擎原生功能 115
端點連線 115
Aurora 端點的類型 116
檢視端點 117
端點和高可用性 118
叢集端點 118
讀取器端點 119
執行個體端點 120
自訂端點 121
資料庫執行個體類別 139
資料庫執行個體類別的類型 139
支援的資料庫引擎 142
在 中判斷資料庫執行個體類別支援 AWS 區域 153
硬體規格 157
儲存 168
Aurora 儲存體的概觀 168
叢集磁碟區內容 169
Aurora 叢集儲存組態 169
儲存體如何調整大小 170
資料計費 171
可靠性 171
儲存體自動修復 171
可存活的頁面快取 172
從非計劃的重新啟動中復原 172
Aurora 安全性 173
將 SSL 與 Aurora 資料庫叢集搭配使用 174
高可用性 174
Aurora 資料的高可用性 175
Aurora 資料庫執行個體的高可用性 175
使用 Aurora 全域資料庫跨 AWS 區域提供高可用性 176
容錯能力 176
與 Amazon RDS Proxy 的高可用性 178
複寫 178
Aurora 複本 178
Aurora MySQL 180
Aurora PostgreSQL 181

Aurora 資料庫執行個體計費 181
隨需資料庫執行個體 183
預留資料庫執行個體 185
AWS 免費方案上的 Aurora 198
設定您的環境 200
註冊 AWS 帳戶 200
建立具有管理存取權的使用者 200
授與程式設計存取權 202
判定需求 203
提供對資料庫叢集的存取 204
入門 207
建立 Aurora MySQL 資料庫叢集並與之連線 207
先決條件 209
步驟 1：建立 EC2 執行個體 209
步驟 2：建立 Aurora MySQL 資料庫叢集 214
（選用）使用 建立 VPC、EC2 執行個體和 Aurora MySQL 叢集 CloudFormation 218
步驟 3：連線至 Aurora MySQL 資料庫叢集 220
步驟 4：刪除 EC2 執行個體和資料庫叢集 223
(選用) 刪除使用 CloudFormation 建立的 EC2 執行個體和資料庫叢集 224
(選用) 將資料庫叢集連線至 Lambda 函數 224
建立 Aurora PostgreSQL 資料庫叢集並與之連線 224
使用完整組態建立 225
使用快速組態建立 241
教學：建立 Web 伺服器和 Amazon Aurora 資料庫叢集 259
啟動 EC2 執行個體以與資料庫叢集連線 260
建立資料庫叢集 266
安裝 Web 伺服器 277
教學課程和範本程式碼 289
本指南中的教學課程 289
其他 AWS 指南中的教學課程 290
GitHub 中的教學課程和範本程式碼 291
AWS 資料庫技術指南 291
AWS 研討會 - Amazon Aurora PostgreSQL 292
AWS 研討會 - Amazon Aurora MySQL 293
使用 AWS SDKs 294
以程式設計方式存取 Amazon Aurora 296

Console-to-Code 297
設定 Aurora 資料庫叢集 298
建立資料庫叢集 299
先決條件 300
建立資料庫叢集 305
可用設定 315
Aurora 資料庫叢集不適用的設定 331
Aurora 資料庫執行個體不適用的設定 332
使用 建立資源 AWS CloudFormation 335
Aurora和 CloudFormation 範本 335
進一步了解 CloudFormation 335
連接至資料庫叢集 336
使用 AWS 驅動程式連線至 Aurora 資料庫叢集 337
連接到 Aurora MySQL 338
連接到 Aurora PostgreSQL 345
對連線進行故障診斷 346
參數群組 348
參數群組概觀 348
資料庫叢集參數群組 351
資料庫參數群組 368
比較資料庫參數群組 384
指定資料庫參數 384
將資料遷移至資料庫叢集 389
Aurora MySQL 389
Aurora PostgreSQL 389
從 Amazon RDS 建立 ElastiCache 快取 390
使用 Aurora 資料庫叢集設定建立 ElastiCache 快取的概觀 390
使用來自 Aurora 資料庫叢集的設定建立 ElastiCache 快取 391
自動遷移 EC2 資料庫 394
概觀 394
先決條件 395
限制 396
建立 IAM 資源 396
設定資料遷移 404
管理遷移 405
監控 407

教學課程：使用自訂參數群組建立 MySQL DB 叢集 ... 409
先決條件 ... 409
建立自訂參數群組 ... 410
修改參數值 ... 410
建立您的資料庫資料庫叢集 ... 411
管理 Aurora 資料庫叢集 ... 412
停用和啟動叢集 ... 413
停用和啟動叢集的概觀 ... 413
限制 ... 414
停用資料庫叢集 ... 414
停用資料庫叢集時 ... 415
啟動資料庫叢集 ... 416
連線至 EC2 執行個體 ... 418
概觀 ... 419
連線至 EC2 執行個體 ... 422
檢視已連線的運算資源 ... 425
連線至執行特定資料庫引擎的資料庫執行個體 ... 426
連線 Lambda 函數 ... 427
概觀 ... 429
連線 Lambda 函數 ... 437
檢視已連線的運算資源 ... 439
修改 Aurora 資料庫叢集 ... 440
使用主控台、CLI 和 API 修改資料庫叢集 ... 440
修改資料庫叢集中的資料庫執行個體 ... 442
變更主要使用者的密碼 ... 444
可用設定 ... 446
Aurora 資料庫叢集不適用的設定 ... 476
Aurora 資料庫執行個體不適用的設定 ... 477
新增 Aurora 複本 ... 479
Aurora 複本的 Auto Scaling ... 485
新增自動擴展政策 ... 490
編輯自動擴展政策 ... 501
刪除自動擴展政策 ... 503
管理效能和擴展 ... 506
儲存體擴展 ... 506
執行個體擴展 ... 512

讀取擴展 513
管理連線 513
管理查詢執行計劃 513
複製 Aurora 資料庫叢集的一個磁碟區 514
Aurora 複製的概觀 514
Aurora 複製的限制 515
Aurora 複製的運作方法 516
建立 Aurora 複製 519
跨 VPC 複製 529
跨帳戶複製 545
與 AWS 服務整合 561
Aurora MySQL 561
Aurora PostgreSQL 561
維持 Aurora 資料庫叢集 563
資料庫叢集維護更新概觀 563
檢視待處理的維護 564
選擇 Aurora MySQL 維護更新的頻率 566
Maintenance window (維護時段) 567
套用更新 572
Aurora 資料庫叢集的自動次要版本升級 574
作業系統更新 578
AWS Organizations 升級推展 585
重新啟動 Aurora 資料庫叢集或執行個體 589
在 Aurora 叢集中重新啟動資料庫執行個體 590
使用讀取可用性功能重新啟動 Aurora 叢集 591
在無讀取可用性的情況下重新啟動 Aurora 叢集 592
檢查 Aurora 叢集和執行個體的執行時間 593
Aurora 重新啟動操作的範例 596
容錯移轉 Aurora 資料庫叢集 612
刪除 Aurora 叢集和執行個體 614
刪除 Aurora 資料庫叢集 614
Aurora 叢集的刪除保護 621
刪除已停止的 Aurora 叢集 622
刪除僅供讀取複本的 Aurora MySQL 叢集 622
刪除叢集時的最終快照 622
從 Aurora 個體資料庫叢集刪除資料庫執行個體 622

標記 Aurora 和 RDS 資源 625
為什麼要使用 RDS 標籤？ 626
RDS 標籤的運作方式 627
最佳實務 629
複製標籤到資料庫叢集快照 630
標記自動備份 631
在 Amazon RDS 中新增和刪除標籤 631
教學：使用標籤指定要停止哪些 Aurora 資料庫叢集 635
Amazon RDS 中的 ARN 639
建構 ARN 639
取得現有的 ARN 646
Aurora 更新 650
識別您的 Amazon Aurora 版本 650
RDS 延長支援 652
RDS 延伸支援概觀 653
RDS 延長支援費用 653
避免 RDS 延長支援的費用 654
具有 RDS 延伸支援的版本 654
RDS 延伸支援的責任 654
Amazon Aurora 責任 655
您的責任 655
建立Aurora 資料庫叢集或全域叢集 655
RDS 延伸支援行為 656
RDS 延伸支援的考量事項 656
使用 RDS 延伸支援建立Aurora 資料庫叢集或全域叢集 657
檢視 RDS 延長支援註冊 658
檢視支援日期 660
還原Aurora 資料庫叢集或全域叢集 662
RDS 延伸支援行為 663
RDS 延伸支援的考量事項 663
使用 RDS 延伸支援還原Aurora 資料庫叢集資料庫叢集或全域叢集 664
使用藍/綠部署進行資料庫更新 666
藍/綠部署的概觀 667
區域和版本可用性 667
優勢 668
工作流程 668

授權存取 ... 673
限制及考量 ... 675
最佳實務 ... 681
建立藍/綠部署 ... 685
準備進行藍/綠部署 ... 685
指定變更 ... 687
建立藍/綠部署 ... 687
可用設定 ... 689
檢視藍/綠部署 ... 691
切換藍/綠部署 ... 694
切換逾時 ... 694
切換防護機制 ... 694
切換動作 ... 695
切換最佳實務 ... 696
在切換前驗證 CloudWatch 指標 ... 697
轉換前監控複本延遲 ... 698
切換藍/綠部署 ... 698
切換後 ... 700
刪除藍/綠部署 ... 702
備份與還原 Aurora 資料庫叢集 ... 705
備份與還原概觀 ... 706
備份 ... 706
備份時段 ... 707
還原資料 ... 709
資料庫複製 ... 710
恢復 ... 710
保留自動備份 ... 711
保留期間 ... 711
保留成本 ... 712
防止自動刪除備份 ... 712
限制 ... 712
檢視保留的自動備份 ... 713
刪除保留的自動備份 ... 713
備份儲存體 ... 715
自動備份儲存 ... 715
快照儲存 ... 715

用於備份儲存的 CloudWatch 指標 715
計算備份儲存用量 716
FAQs 717
建立資料庫叢集快照 720
判斷快照是否可用 722
從資料庫叢集快照還原 723
參數群組 723
Security groups (安全群組) 724
Aurora 考量 724
從快照還原 724
資料庫叢集快照複製 727
使用 AWS 管理主控台 複製資料庫叢集快照 727
限制 728
考量事項 729
複製未加密資料庫叢集快照 731
複製加密資料庫叢集快照 733
跨帳戶複製資料庫叢集快照 736
共享資料庫叢集快照 741
共用快照 742
共用公有快照 745
共用加密快照 746
停止快照共用 750
將資料庫叢集資料匯出至 Amazon S3 752
考量事項 753
設定對 S3 儲存貯體的存取權 755
建立資料庫叢集匯出任務 759
監控資料庫叢集匯出 762
取消資料庫叢集匯出 764
故障診斷 765
將資料庫叢集快照資料匯出至 Amazon S3 768
考量事項 770
設定對 S3 儲存貯體的存取權 780
建立快照匯出任務 786
監控快照匯出 789
取消快照匯出 791
Aurora MySQL 中的匯出效能 793

疑難排解 793
時間點復原 796
將資料庫叢集還原至時間點 797
從保留的自動備份的時間點復原 799
使用 AWS Backup 進行時間點復原 802
刪除資料庫叢集快照 808
刪除資料庫叢集快照 808
教學：從快照還原資料庫執行個體叢集 810
使用主控台還原資料庫叢集 810
使用 AWS CLI 還原資料庫叢集 814
在 Aurora 資料庫叢集中監控指標 821
監控計畫 821
效能基準 822
效能指導方針 822
監控工具 824
自動化監控工具 824
手動監控工具 826
檢視叢集狀態 827
檢視資料庫叢集 828
檢視資料庫叢集狀態 834
在 Aurora 叢集中檢視 資料庫執行個體狀態 838
Amazon Aurora 的建議 844
檢視 建議 846
套用建議 852
關閉建議 856
將已關閉的建議修改為作用中 857
建議參考 859
在 Amazon RDS 主控台中檢視指標 871
檢視 Performance Insights 儀表板 872
在監控索引標籤中選擇新的監控檢視 873
選擇具有 Performance Insights 頁面的新監控檢視 874
建立自訂儀表板 876
選擇預先設定的儀表板 879
用 CloudWatch 監控 Aurora 881
Amazon Aurora 和 Amazon CloudWatch 的概觀 882
檢視 CloudWatch 指標 883

將 Performance Insights 指標匯出至 CloudWatch 889
建立 CloudWatch 警示 894
使用 Database Insights 進行監控 896
定價 896
引擎、區域和執行個體類別支援 896
開啟進階模式 900
開啟標準模式 904
監控慢速查詢 909
考量事項 910
使用 Performance Insights 來監控資料庫負載 912
績效詳情概觀 913
開啟和關閉 Performance Insights 922
Aurora MySQL 的效能結構描述 928
績效詳情政策 933
使用績效詳情儀表板來分析指標 947
檢視 Performance Insights 主動建議 979
使用績效詳情 API 來擷取指標 981
使用 AWS CloudTrail 記錄績效詳情呼叫 1005
VPC 端點 (AWS PrivateLink) 1007
使用 DevOps Guru for RDS 分析效能 1010
DevOps Guru for RDS 的好處 1010
DevOps Guru for RDS 的運作方式 1011
設定 DevOps Guru for RDS 1012
使用增強型監控來監控作業系統 1020
增強型監視概觀 1020
設定並啟用增強型監控 1021
在 RDS 主控台中檢視作業系統指標 1028
使用 CloudWatch Logs 檢視作業系統指標 1030
Aurora 指標參考 1031
Aurora 的 CloudWatch 指標 1031
Aurora 的 CloudWatch 維度 1059
Amazon RDS 主控台中 Aurora 指標的可用性 1060
Performance Insights 的 CloudWatch 指標 1064
Performance Insights 的計數器指標 1066
績效詳情的 SQL 統計數字 1092
增強型監控中的作業系統指標 1100

監控事件、日誌和資料庫活動串流 1107
在 Amazon RDS 主控台中檢視日誌、事件和串流 1108
監控 Aurora 事件 1112
Aurora 的事件概觀 1112
檢視 Amazon RDS 事件 1114
使用 Amazon RDS 事件通知 1118
建立由 Amazon Aurora 事件觸發的規則 1143
Aurora 的 Amazon RDS 事件類別和事件訊息 1147
監控 Aurora 日誌 1171
檢視並列出資料庫日誌檔案 1171
下載資料庫日誌檔案 1173
查看資料庫日誌檔案 1174
發佈至 CloudWatch Logs 1175
使用 REST 讀取日誌檔案內容 1178
MySQL 資料庫日誌檔案 1179
PostgreSQL 資料庫日誌檔案 1188
在 CloudTrail 中監控 Aurora API 呼叫 1197
CloudTrail 與 Amazon Aurora 整合 1197
Amazon Aurora 日誌檔案項目 1198
使用資料庫活動串流來監控 Aurora 1202
概觀 1202
Aurora MySQL 網路必要條件 1205
開始資料庫活動串流 1207
取得活動串流狀態 1210
停用資料庫活動串流 1212
監控活動串流 1213
活動串流的 IAM 政策範例 1247
使用 GuardDuty RDS Protection 監控威脅 1250
使用 Aurora MySQL 1251
Aurora MySQL 概觀 1251
Amazon Aurora MySQL 效能增強功能 1252
Aurora MySQL 和空間資料 1253
Aurora MySQL 第 3 版與 MySQL 8.0 相容 1254
Aurora MySQL 第 2 版與 MySQL 5.7 相容 1284
Aurora MySQL 的安全性 1287
Aurora MySQL 的主要使用者權限 1288

TLS 連線 1289
更新應用程式以取得新的 TLS 憑證 1296
判斷任何應用程式是否使用 TLS 連線至 Aurora MySQL 資料庫叢集 1296
判斷用戶端是否需要驗證憑證才能連線 1297
更新應用程式信任存放區 1298
建立 TLS 連線的 Java 程式碼範例 1299
針對 Aurora MySQL 使用 Kerberos 身分驗證 1301
Aurora MySQL 的 Kerberos 身分驗證概觀 1302
限制 1303
為 Aurora MySQL 設定 Kerberos 身分驗證 1304
使用 Kerberos 身分驗證連線至 Aurora MySQL 1313
管理網域中的資料庫叢集 1316
將資料遷移至 Aurora MySQL 1318
從外部 MySQL 資料庫遷移至 Aurora MySQL 1323
從 MySQL 資料庫執行個體遷移至 Aurora MySQL 1347
管理 Aurora MySQL 1370
管理 Amazon Aurora MySQL 的效能和擴展 1370
恢復資料庫叢集 1380
使用錯誤注入查詢測試 Amazon Aurora MySQL 1400
使用快速 DDL 更改 Amazon Aurora 中的資料表 1404
顯示 Aurora 資料庫叢集的磁碟區狀態 1410
調校 Aurora MySQL 1412
Aurora MySQL 調校的基本概念 1412
使用等待事件調校 Aurora MySQL 1415
使用執行緒狀態調校 Aurora MySQL 1466
使用 Amazon DevOps Guru 主動洞察，調校 Aurora MySQL 1473
Aurora MySQL 的平行查詢 1478
平行查詢的概觀 1478
建立平行查詢叢集 1481
開啟和關閉平行查詢 1485
最佳化平行查詢 1488
驗證平行查詢使用情形 1492
監控平行查詢 1496
平行查詢的 SQL 建構 1500
進階稽核搭配 Aurora MySQL 1519
啟用進階稽核 1519

檢視稽核日誌 1522
稽核日誌詳細資訊 1522
以 Aurora MySQL 進行複寫 1524
Aurora 複本 1524
複寫選項 1525
複寫效能 1526
複寫篩選條件 1527
監控複寫 1533
跨區域複寫 1534
二進位日誌 (binlog) 複寫 1547
GTID 式複寫 1586
本機寫入轉送 1592
啟用本機寫入轉送 1593
檢查資料庫叢集是否已啟用寫入轉送 1594
應用程式和 SQL 與寫入轉送的相容性 1595
寫入轉送的隔離層級 1597
讀取一致性 1597
使用寫入轉送執行多部分陳述式 1601
具有寫入轉送的交易 1601
寫入轉送的組態參數 1601
寫入轉送的指標 1603
識別轉送的交易和查詢 1606
將 Aurora MySQL 與 AWS 服務整合 1608
授權 Aurora MySQL 存取 AWS 服務 1608
從 Amazon S3 中的文字檔案載入資料 1625
將資料儲存至 Amazon S3 中的文字檔案 1638
從 Aurora MySQL 叫用 Lambda 函式 1648
將 Aurora MySQL 日誌發佈至 CloudWatch Logs 1659
Aurora MySQL 實驗室模式 1664
Aurora 實驗室模式功能 1664
Aurora MySQL 的最佳實務 1665
判斷您連接的資料庫執行個體 1666
效能和擴展的最佳實務 1666
高可用性的最佳實務 1674
對 MySQL 功能的建議 1676
使用 Amazon CloudWatch 指標評估 Aurora MySQL 的資料庫執行個體用量 1682

故障診斷 Aurora MySQL 效能 1684
AWS 監控選項 1684
資料庫效能問題的最常見原因 1685
針對工作負載問題進行故障診斷 1686
記錄 Aurora MySQL 1713
針對資料庫連線問題進行疑難排解 1715
針對查詢效能進行故障診斷 1728
Aurora MySQL 參考 1733
組態參數 1733
全域狀態變數 1790
等待事件 1803
執行緒狀態 1808
隔離層級 1813
提示 1818
預存程序參考 1821
information_schema 資料表 1865
Aurora MySQL 更新 1872
檢查版本編號 1872
長期支援和 Beta 版 1874
為 Aurora MySQL 第 2 版結束生命週期做好準備 1876
為 Aurora MySQL 第 1 版結束生命週期做好準備 1880
升級 Amazon Aurora MySQL 資料庫叢集 1883
Amazon Aurora MySQL 的資料庫引擎更新和修正 2001
使用 Aurora PostgreSQL 2002
資料庫預覽環境 2003
支援的資料庫執行個體類別類型 2004
預覽環境中不支援的功能 2005
在預覽環境中建立新的資料庫叢集 2005
資料庫預覽環境中的 PostgreSQL 第 17 版 2007
Aurora PostgreSQL 的安全性 2008
了解 PostgreSQL 角色和許可 2009
使用 SSL/TLS 保護 Aurora PostgreSQL 資料的安全 2023
Aurora PostgreSQL 動態遮罩 2036
將應用程式更新為使用新的 SSL/TLS 憑證 2068
判斷任何應用程式是否使用 SSL 連線至 Aurora PostgreSQL 資料庫叢集 2069
判斷用戶端是否需要驗證憑證才能連線 2069

更新應用程式信任存放區 2070
針對不同類型的應用程式使用 SSL/TLS 連線 2070
使用 Kerberos 身分驗證 2071
區域和版本可用性 2072
Kerberos 身分驗證概觀 2073
設定 2074
在 Active Directory 網域中管理 Aurora PostgreSQL 資料庫叢集 2087
使用 Kerberos 身分驗證進行連線 2088
使用 AD 安全群組進行 Aurora PostgreSQL 存取控制 2091
將資料遷移至 Aurora PostgreSQL 2102
使用快照遷移 RDS for PostgreSQL 資料庫執行個體 2103
使用 Aurora 僅供讀取複本遷移 RDS for PostgreSQL 資料庫執行個體 2109
在 Aurora PostgreSQL 中最佳化查詢效能 2121
使用 Aurora Optimized Reads 改善查詢效能 2121
最佳化 Aurora PostgreSQL 中的相互關聯子查詢 2127
使用調適型聯結改善查詢效能 2133
使用共用計劃快取 2134
在 Aurora PostgreSQL 中使用未記錄的資料表 2136
建立未記錄的資料表 2137
在遷移期間處理未記錄的資料表 2137
將未記錄的資料表轉換為已記錄的資料表 2138
未記錄的資料表和邏輯複寫 2138
使用 PostgreSQL 自動資料清理 2139
配置自動資料清理的記憶體 2140
降低交易 ID 包圍的可能性 2141
判斷資料庫中的資料表是否需要清理 2142
判斷哪些資料表目前適合進行自動資料清理 2143
判斷自動資料清理目前是否執行中且執行多久時間 2144
執行手動清理凍結 2145
在自動資料清理執行時重新為資料表建立索引 2147
使用大型索引管理自動清空 2148
影響自動資料清理的其他參數 2151
設定自動資料清理參數資料表層級 2152
記錄清理和自動資料清理活動 2152
了解具有無效資料庫的自動清空行為 2153
識別清空封鎖程式 2156

管理 TOAST OID 爭用 2176
了解 TOAST 操作 2176
識別效能挑戰 2176
建議 2178
監控 2179
使用 Babelfish for Aurora PostgreSQL 2184
Babelfish 限制 2186
了解 Babelfish 架構和組態 2187
建立 Babelfish for Aurora PostgreSQL DB 叢集 2225
將 SQL Server 資料庫遷移到 Babelfish 2234
使用 Babelfish for Aurora PostgreSQL 進行資料庫身分驗證 2243
連線至 Babelfish 資料庫叢集 2261
使用 Babelfish 2274
Babelfish 疑難排解 2344
停用 Babelfish 2345
管理 Babelfish 版本更新 2346
Babelfish 參照 2367
Aurora PostgreSQL 的效能和擴展 2421
擴展 Aurora PostgreSQL 資料庫執行個體 2422
連線數上限 2422
暫存空間限制 2423
Aurora PostgreSQL 的巨型分頁 2428
使用錯誤注入查詢測試 Amazon Aurora PostgreSQL 2428
顯示 Aurora 資料庫叢集的磁碟區狀態 2433
指定 stats_temp_directory 的 RAM 磁碟 2434
使用 PostgreSQL 管理暫存檔案 2435
調校 Aurora PostgreSQL 的等待事件 2441
Aurora PostgreSQL 調校的基本概念 2442
Aurora PostgreSQL 等待事件 2446
Client:ClientRead 2448
Client:ClientWrite 2451
CPU 2453
IO:BufFileRead 和 IO:BufFileWrite 2458
IO:DataFileRead 2465
IO:XactSync 2475
IPC:DamRecordTxAck 2477

IPC:parallel 等待事件 2478
IPC:ProcArrayGroupUpdate 2484
Lock:advisory 2486
Lock:extend 2488
Lock:Relation 2491
Lock:transactionid 2497
Lock:tuple 2500
LWLock:buffer_content (BufferContent) 2504
LWLock:buffer_mapping 2507
LWLock:BufferIO (IPC:BufferIO) 2509
LWLock:lock_manager 2511
LWLock:MultiXact 2515
LWLock:pg_stat_statements 2519
Timeout:PgSleep 2523
使用 Amazon DevOps Guru 主動洞察，調校 Aurora PostgreSQL 2523
資料庫在交易連線中長時間閒置 2524
Aurora PostgreSQL 的最佳實務 2527
避免 Aurora PostgreSQL 資料庫執行個體效能變慢、自動重新啟動和容錯移轉 2527
診斷資料表和索引膨脹 2528
在 Amazon Aurora PostgreSQL 中管理高物件計數 2531
Aurora PostgreSQL 中的記憶體管理已改善 2537
快速容錯移轉 2539
容錯移轉後快速復原 2549
管理連線流失 2555
PostgreSQL 中的無效連線處理 2562
調整 Aurora PostgreSQL 的記憶體參數 2565
使用 CloudWatch 指標分析資源用量 2572
使用邏輯複寫進行主要版本升級 2576
管理自訂轉換 2584
Aurora PostgreSQL 中平行查詢的最佳實務 2588
針對 Aurora PostgreSQL 中的儲存問題進行故障診斷 2592
以 Aurora PostgreSQL 進行複寫 2593
Aurora 複本 2593
改善 Aurora 複本的讀取可用性 2594
監控複寫 2596
邏輯複寫概觀 2596

設定邏輯複寫 2597
關閉邏輯複寫 2599
監控全部寫入快取和邏輯槽 2600
範例：使用邏輯複寫 2602
範例：使用 Aurora PostgreSQL 和 進行邏輯複寫 AWS DMS 2604
設定邏輯複寫連線的 IAM 身分驗證 2606
在 Aurora PostgreSQL 中使用本機寫入轉送 2608
Aurora PostgreSQL 中本機寫入轉送的限制和考量 2609
設定 Aurora PostgreSQL 以進行本機寫入轉送 2610
使用 Aurora PostgreSQL 的本機寫入轉送 2613
在 Aurora PostgreSQL 中監控本機寫入轉送 2616
使用 Aurora PostgreSQL 作為 Amazon Bedrock 的知識庫 2622
先決條件 2622
準備將 Aurora PostgreSQL 作為知識庫 2623
在 Bedrock 主控台中建立知識庫 2626
快速建立 Aurora PostgreSQL Amazon Bedrock 知識庫 2627
將 Aurora PostgreSQL 與 AWS 服務整合 2629
將資料從 Amazon S3 匯入 Aurora PostgreSQL 2630
將 PostgreSQL 資料匯出至 Amazon S3 2648
從 Aurora PostgreSQL 叫用 Lambda 函數 2663
將 Aurora PostgreSQL 日誌發佈至 CloudWatch Logs 2677
監控 Aurora PostgreSQL 的查詢執行計劃和尖峰記憶體 2689
使用 Aurora 函數存取查詢執行計劃和尖峰記憶體 2689
Aurora PostgreSQL 查詢執行計劃的參數參考 2690
管理 Aurora PostgreSQL 的查詢執行計劃 2694
Aurora PostgreSQL 查詢計劃管理的概觀 2694
Aurora PostgreSQL 查詢計劃管理的最佳實務 2701
查詢計劃管理 2704
擷取 Aurora PostgreSQL 執行計畫 2705
使用 Aurora PostgreSQL 受管計劃 2708
在 dba_plans 檢視中檢查 Aurora PostgreSQL 查詢計劃 2712
改善計劃 2713
刪除計劃 2717
匯出和匯入受管計劃 2719
參數參考 2720
函數參考 2725

apg_plan_mgmt.dba_plans 檢視的參考 2735
查詢計畫管理中的進階功能 2739
使用擴充功能和外部資料包裝函式 2752
使用 PostgreSQL 的 Amazon Aurora 委派的延伸模組支援 2753
使用 lo 模組更有效率地管理大型物件 2765
使用 PostGIS 管理空間資料 2768
使用 pg_partman 擴充功能來管理分割區 2777
使用 pg_cron 擴充功能排程維護 2782
使用 PgAudit 記錄資料庫活動 2791
使用 pglogical 跨執行個體同步資料 2803
Amazon Aurora PostgreSQL 中支援的外部資料包圍函式 2815
使用適用於 PostgreSQL 的受信任語言延伸模組 2829
術語 2830
使用受信任語言延伸模組的需求 2830
設定受信任語言延伸模組 2833
受信任語言延伸模組概觀 2837
建立 TLE 延伸模組 2838
從資料庫中捨棄您的 TLE 延伸模組 2843
解除安裝受信任語言延伸模組 2844
搭配您的延伸模組使用 PostgreSQL 掛鉤 2845
受信任語言延伸模組的函數參考 2850
受信任語言延伸模組的掛鉤參考 2863
Aurora PostgreSQL 參考 2866
適用於 EBCDIC 和其他大型主機遷移的 Aurora PostgreSQL 定序 2866
Aurora PostgreSQL 中支援定序 2868
Aurora PostgreSQL 函數參考 2868
Aurora PostgreSQL 參數 2923
Aurora PostgreSQL 等待事件 2976
Aurora PostgreSQL 更新 3002
識別 Amazon Aurora PostgreSQL 版本 3002
Aurora PostgreSQL 版本 3004
Aurora PostgreSQL 的擴充功能版本 3005
升級 Amazon Aurora PostgreSQL 資料庫叢集 3005
使用長期支援 (LTS) 版本 3029
使用 Aurora PostgreSQL Limitless Database 3031
Limitless Database 架構 3032

重要用語 3034
資料表類型 3035
帳單 3035
Limitless Database 入門 3036
Aurora PostgreSQL Limitless Database 要求和考量事項 3037
要求 3037
考量事項 3038
不支援的功能 3040
使用 Limitless Database 的先決條件 3041
啟用資料庫碎片群組操作 3041
建立資料庫叢集，此叢集會使用 Limitless Database 3042
將容量上限與路由器和碎片建立關聯 3042
建立資料庫叢集 3045
使用資料庫碎片群組 3054
連線至 Aurora PostgreSQL Limitless Database 資料庫叢集 3055
尋找資料庫碎片群組中的路由器和碎片數量 3056
描述資料庫碎片群組 3056
重新啟動資料庫碎片群組 3057
變更資料庫碎片群組的容量 3058
分割碎片 3060
新增路由器 3065
刪除資料庫碎片群組 3069
將資料庫碎片群組新增至現有的 Limitless Database 資料庫叢集 3070
建立 Limitless Database 資料表 3075
使用變數建立無限資料表 3076
將標準資料表轉換為無限資料表 3081
範例結構描述 3085
將資料載入 Limitless Database 3086
使用 COPY 命令搭配 Limitless Database 3088
使用 Limitless Database 資料載入公用程式 3090
查詢 Limitless Database 3116
單一碎片查詢 3116
分散式查詢 3126
分散式查詢追蹤 3127
分散式死鎖 3135
管理 Limitless Database 3137

資料庫和資料表大小考量事項 3137
透過清空回收儲存空間 3138
監控 Limitless Database 3142
使用 CloudWatch 監控 Limitless Database 3143
使用 Database Insights 監控 Limitless Database 3147
使用 CloudWatch Logs 監控 Limitless Database 3158
使用增強型監控來監控 Limitless Database 3158
使用 Performance Insights 監控 Limitless Database 3160
使用 GuardDuty RDS 保護監控 Limitless Database 3168
Limitless Database 的函數和檢視 3169
Limitless Database 的等待事件 3193
使用 函數建置以提高效率 3204
執行個體監控 3211
概觀 3211
執行個體架構基本概念 3211
監控執行個體效能 3212
路由器監控和故障診斷 3213
碎片監控和疑難排解 3214
限制 3215
備份和還原 Limitless Database 3217
備份資料庫叢集，此叢集會使用 Aurora PostgreSQL Limitless Database 3217
還原資料庫叢集，此叢集會使用 Aurora PostgreSQL Limitless Database 3219
不支援 PostgreSQL 備份和還原公用程式 3222
升級 Limitless Database 3223
升級使用 Aurora PostgreSQL Limitless Database 的資料庫叢集 3223
Limitless Database 參考 3225
DDL 命令 3225
DDL 限制和資訊 3234
DML 命令 3270
DML 限制和資訊 3273
Variables 3277
資料庫叢集參數 3278
使用 Aurora 全球資料庫 3279
Aurora 全球資料庫的概觀 3279
Amazon Aurora 全球資料庫的優點 3281
區域和版本可用性 3281

Aurora 全球資料庫的限制 3281
Aurora 全球資料庫入門 3283
組態需求 3284
建立全球資料庫 3285
新增次要叢集 3297
建立無周邊次要叢集 3302
從快照建立全球資料庫 3305
管理 Aurora 全域資料庫 3306
修改 Aurora 全域資料庫 3307
修改全域資料庫參數 3308
從 Aurora 全域資料庫中移除叢集 3308
刪除 Aurora 全域資料庫 3311
Aurora 全球資料庫的標記 3313
連線至 Aurora 全域資料庫 3317
選擇符合您應用程式需求的端點 3317
檢視全域資料庫的端點 3318
使用全域寫入器端點的考量 3321
在 Aurora 全域資料庫中使用寫入轉送 3322
在 Aurora MySQL 中使用寫入轉送 3322
在 Aurora PostgreSQL 中使用寫入轉送 3340
Aurora 全球資料庫的切換或容錯移轉 3353
規劃 BCDR 3353
執行切換 3354
從意外中斷復原 3360
管理 RPO (Aurora PostgreSQL) 3368
次要叢集的跨區域彈性 3373
監控 Aurora 全域資料庫 3374
利用績效詳情來監控 Aurora 全域資料庫 3375
使用資料庫活動串流來監控 Aurora 全域資料庫 3376
監控 Aurora MySQL 型全球資料庫 3376
監控 Aurora PostgreSQL 型全球資料庫 3379
使用 Aurora Global Database 的藍/綠部署 3382
搭配 Aurora Global Database 使用藍/綠部署的優勢 3382
藍/綠部署如何與 Aurora Global Database 搭配使用 3382
將 Aurora 全域資料庫與其他 AWS 服務搭配使用 3385
搭配 Aurora Global Database 使用 Amazon Application Recovery Controller (ARC) 3386

升級 Amazon Aurora 全域資料庫 3386
主要版本升級 3386
次要版本升級 3387
Amazon RDS Proxy 3391
區域和版本可用性 3392
限制和配額 3392
MySQL 限制 3394
PostgreSQL 限制 3395
規劃在哪裡使用 RDS Proxy 3396
RDS Proxy 概念和術語 3397
RDS Proxy 概念概觀 3398
連線集區 3398
安全性 3399
容錯移轉 3401
交易 3402
RDS Proxy 入門 3402
設定代理網路 3403
設定資料庫憑證 3407
設定 IAM 身分驗證 3411
建立代理 3417
檢視代理 3428
透過 RDS Proxy 連線 3429
管理 RDS Proxy 3433
修改 RDS Proxy 3433
新增資料庫使用者 3440
移至end-to-end IAM 身分驗證 3441
RDS Proxy 連線考量事項 3444
避免鎖定 RDS Proxy 3448
刪除 RDS Proxy 3452
使用 RDS Proxy 端點 3453
代理端點概觀 3454
代理端點的限制 3454
將讀取器端點與 Aurora 叢集搭配使用 3455
跨 VPC 存取 Aurora 資料庫 3459
建立代理端點 3460
檢視代理端點 3463

修改代理端點 3464
刪除代理端點 3465
使用 CloudWatch 監控 RDS Proxy 3466
使用 RDS Proxy 事件 3472
RDS Proxy 事件 3472
RDS Proxy 範例 3474
對 RDS Proxy 進行故障診斷 3477
驗證代理的連線能力 3477
常見問題與解決方案 3479
針對 RDS for MySQL 進行疑難排解 3480
針對 RDS for PostgreSQL 進行疑難排解 3482
將 RDS Proxy 與 AWS CloudFormation 搭配使用 3487
搭配 Aurora 全域資料庫使用 RDS Proxy 3488
RDS Proxy 搭配全域資料庫的限制 3488
RDS Proxy 端點如何使用全域資料庫 3488
零 ETL 整合 3490
優勢 3492
重要概念 3492
限制 3493
一般限制 3493
Aurora MySQL 限制 3494
Aurora PostgreSQL 限制 3494
Amazon Redshift 限制 3494
Amazon SageMaker AI Lakehouse 限制 3495
配額 3495
支援的區域 3495
開始使用零 ETL 整合 3495
步驟 1：建立自訂資料庫叢集參數群組。 3496
步驟 2：選取或建立來源資料庫叢集 3497
步驟 3a：建立目標資料倉儲 3498
使用 AWS SDKs 設定整合 3499
步驟 3b：建立 Amazon SageMaker Lakehouse 零 ETL 整合的 AWS Glue 目錄 3509
後續步驟 3512
建立與 Amazon Redshift 的零 ETL 整合 3512
先決條件 3512
所需的許可 3513

建立零 ETL 整合 3516
加密整合 3519
後續步驟 3521
建立與 Amazon SageMaker 資料湖倉的零 ETL 整合 3521
先決條件 3521
所需的許可 3522
建立與 Amazon SageMaker 資料湖倉的零 ETL 整合 3524
加密整合 3529
後續步驟 3531
零 ETL 整合的資料篩選 3531
資料篩選的格式 3532
篩選條件邏輯 3534
篩選條件優先順序 3534
Aurora MySQL 範例 3535
Aurora PostgreSQL 範例 3535
新增資料篩選條件 3536
移除資料篩選條件 3538
新增和查詢資料 3538
建立目標資料庫 3539
將資料新增至來源資料庫叢集 3539
在 Amazon Redshift 中查詢您的 Aurora 資料。 3540
資料類型差異 3541
Aurora PostgreSQL 的 DDL 操作 3551
檢視和監控零 ETL 整合 3553
檢視整合 3554
使用系統資料表進行監視 3555
使用 EventBridge 進行監控 3556
修改零 ETL 整合 3556
刪除零 ETL 整合 3557
對零 ETL 整合進行疑難排解 3558
我無法建立零 ETL 整合 3559
我的整合停滯在 Syncing 狀態 3559
我的資料表未複寫至 Amazon Redshift 3559
我的一個或多個 Amazon Redshift 資料表需要重新同步 3560
Amazon SageMaker AI Lakehouse 零 ETL 整合的整合失敗問題 3563
未完成的 DDL 操作會顯示在 Amazon Redshift 中 3563

使用 Aurora Serverless v2 3564
Aurora Serverless v2 使用案例 3564
轉換已佈建工作負載 3566
Aurora Serverless v2 的優點 3566
Aurora Serverless v2 的運作方式 3567
概觀 3568
叢集組態 3569
Capacity 3569
擴展 3572
高可用性 3573
儲存 3574
組態參數 3575
Aurora Serverless v2 的要求與限制 3575
區域和版本可用性 3575
使用 Aurora Serverless v2 的叢集必須指定容量範圍 3576
Aurora Serverless v2 中不支援某些佈建功能 3576
某些 Aurora Serverless v2 方面與 Aurora Serverless v1 不同 3577
建立 Aurora Serverless v2 資料庫叢集 3577
設定 3577
建立 Aurora Serverless v2 資料庫叢集 3578
建立 Aurora Serverless v2 寫入器 3582
管理 Aurora Serverless v2 3583
設定叢集的 Aurora Serverless v2 容量範圍 3584
檢查 Aurora Serverless v2 容量範圍 3590
新增 Aurora Serverless v2 讀取器 3591
從已佈建轉換為 Aurora Serverless v2 3592
從 Aurora Serverless v2 轉換為已佈建 3593
暫停 Aurora Serverless v2 資料庫執行個體 3594
選擇 Aurora Serverless v2 讀取器的提升層 3594
搭配 Aurora Serverless v2 使用 TLS/SSL 3595
檢視 Aurora Serverless v2 寫入器和讀取器 3596
為 Aurora Serverless v2 記錄日誌。 3597
Aurora Serverless v2 的效能和擴展 3603
選擇容量範圍 3604
使用 Aurora Serverless v2 的參數群組 3616
避免記憶體不足錯誤 3620

重要的 CloudWatch 指標 3621
使用績效詳情監控 Aurora Serverless v2 效能 3625
針對 Aurora Serverless v2 容量問題進行疑難排解 3626
使用自動暫停和繼續擴展至 0 ACU 3628
自動暫停概觀 3628
先決條件和限制 3629
開啟和關閉自動暫停 3630
自動暫停的運作方式 3633
自動暫停和 Aurora 叢集組態 3637
監控自動暫停 3639
自動暫停的故障診斷 3642
自動暫停的應用程式設計 3644
遷移至 Aurora Serverless v2 3644
Aurora Serverless v2 與現有叢集一起使用 3645
從已佈建叢集切換 3648
比較 Aurora Serverless v2 和 Aurora Serverless v1 3653
從 Aurora Serverless v1 升級至 Aurora Serverless v2 3661
從內部部署資料庫遷移至 Aurora Serverless v2 3664
使用 Aurora Serverless v1 3665
Aurora Serverless v1 的區域和版本可用性 3666
Aurora Serverless v1 的優點 3666
Aurora Serverless v1 的應用案例 3666
Aurora Serverless v1 的限制 3667
Aurora Serverless v1 的組態需求 3669
搭配 Aurora Serverless v1 使用 TLS/SSL 3670
適用於 Aurora Serverless v1 資料庫叢集連線的受支援密碼套件 3672
Aurora Serverless v1 的運作方式 3673
Aurora Serverless v1 架構 3673
自動擴展 3674
逾時動作 3675
暫停和繼續 3677
決定 max_connections 3678
參數群組 3680
日誌 3683
Maintenance (維護) 3686
容錯移轉 3687

快照 3687
建立 Aurora Serverless v1 資料庫叢集 3688
還原 Aurora Serverless v1 資料庫叢集 3691
修改 Aurora Serverless v1 資料庫叢集 3694
修改擴展組態 3695
升級主要版本 3697
從 Aurora Serverless v1 轉換為已佈建 3699
從 Aurora Serverless v1 資料庫叢集轉換為已佈建叢集時的考量 3700
手動擴展 Aurora Serverless v1 資料庫叢集容量 3702
檢視 Aurora Serverless v1 資料庫叢集 3704
使用 CloudWatch 監控 Aurora Serverless v1 資料庫叢集 3707
刪除 Aurora Serverless v1 資料庫叢集 3708
Aurora Serverless v1 和 Aurora 資料庫引擎版本 3711
Aurora MySQL Serverless 3712
Aurora PostgreSQL Serverless 3712
自動次要版本升級 3712
使用 Amazon RDS 資料 API 3713
Amazon RDS 資料 API 的區域和版本可用性 3714
IPv6 支援 3714
IPv6 端點支援 3714
使用 IPv6 端點 3715
AWS PrivateLink 搭配 IPv6 使用 3718
遷移考量事項 3718
針對 IPv6 連線進行疑難排解 3719
限制 3719
與 Aurora Serverless v1 相比具有 Aurora Serverless v2 的資料 API 3720
每秒請求數上限 3720
在現有資料庫啟用或停用 Amazon RDS 資料 API 3720
CloudTrail 事件 3721
多陳述式支援 3721
相同交易 ID 的並行請求 3722
BatchExecuteStatement 行為 3723
ExecuteSQL 行為 3723
ExecuteStatement 行為 3724
結構描述參數行為 3724
身分驗證和授權 3725

標籤型授權 3726
在 Secrets Manager 秘密中存放憑證 3728
啟用資料 API 3729
建立資料庫時啟用 RDS 資料 API 3729
在現有資料庫啟用 RDS 資料 API 3730
建立 Amazon VPC 端點 (PrivateLink) 3733
呼叫資料 API 3736
操作參考 3736
從 AWS CLI 呼叫 3741
從 Python 應用程式呼叫 3751
從 Java 應用程式呼叫 3755
控制逾時行為 3759
Java 用戶端程式庫 3760
下載適用於資料 API 的 Java 用戶端程式庫 3761
Java 用戶端程式庫範例 3761
以 JSON 處理查詢結果 3763
以 JSON 格式擷取查詢結果 3763
資料類型映射 3764
故障診斷 3764
範例 3765
針對資料 API 進行疑難排解 3769
找不到交易 <transaction_ID> 3770
查詢封包過大 3770
資料庫回應超過大小上限 3770
叢集 <cluster_ID> 並未啟用 HttpEndpoint 3770
DatabaseErrorException：交易仍在執行查詢 3771
不支援的結果例外狀況 3771
不支援多陳述式 3771
不支援結構描述參數 3772
IPv6 連線問題 3772
記錄資料 API 呼叫 3772
使用 CloudTrail 中的資料 API 資訊 3773
包含 CloudTrail 線索中的資料 API 事件，和將資料 API 事件從 CloudTrail 線索中排除 3774
了解資料 API 日誌檔案項目 3776
監控資料 API 查詢 3778
Performance Insights 中 RDS 資料 API 查詢的呈現方式 3778

使用查詢編輯器 3779
查詢編輯器的可用性 3779
授權存取 3779
執行查詢 3781
DBQMS API 參考 3785
CreateFavoriteQuery 3786
CreateQueryHistory 3786
CreateTab 3786
DeleteFavoriteQueries 3786
DeleteQueryHistory 3786
DeleteTab 3786
DescribeFavoriteQueries 3786
DescribeQueryHistory 3786
DescribeTabs 3786
GetQueryString 3787
UpdateFavoriteQuery 3787
UpdateQueryHistory 3787
UpdateTab 3787
使用 Aurora 機器學習 3788
將 Aurora Machine Learning 與 Aurora MySQL 搭配使用 3789
使用 Aurora Machine Learning 的建議 3790
區域和版本可用性 3791
支援的功能和限制 3791
為 Aurora Machine Learning 設定您的 Aurora 叢集 3792
搭配 Aurora MySQL 資料庫叢集使用 Amazon Bedrock 3804
搭配 Aurora MySQL 資料庫叢集使用 Amazon Comprehend 3806
搭配 Aurora MySQL 資料庫叢集使用 SageMaker AI 3808
效能考量 3812
監控 3813
將 Aurora Machine Learning 與 Aurora PostgreSQL 搭配使用 3814
使用 Aurora Machine Learning 的建議 3815
支援的功能和限制 3816
設定 Aurora 資料庫叢集來使用 Aurora Machine Learning 3816
搭配 Aurora PostgreSQL 資料庫叢集使用 Amazon Bedrock 3829
搭配 Aurora PostgreSQL 資料庫叢集使用 Amazon Comprehend 3831
搭配 Aurora PostgreSQL 資料庫叢集使用 SageMaker AI 3833

將資料匯出至 Amazon S3 進行 SageMaker AI 模型訓練 (進階) 3837
效能考量 3837
監控 3842
程式碼範例 3843
基本概念 3844
Hello Aurora 3844
了解基本概念 3855
動作 4024
案例 4235
建立出借圖書館 REST API 4235
建立 Aurora 無伺服器工作項目追蹤器 4236
Aurora 的最佳實務 4241
Amazon Aurora 的基本操作準則 4241
資料庫執行個體 RAM 建議 4242
AWS 資料庫驅動程式 4243
監控 Amazon Aurora 4243
使用資料庫參數群組和資料庫叢集參數群組 4243
Amazon Aurora 最佳實務影片 4243
執行 Aurora 概念驗證 4244
Aurora 概念驗證概觀 4244
1. 確認您的目標 4244
2. 了解您的工作負載特性 4245
3. 透過主控台或 CLI 來練習 4246
透過主控台來練習 4246
透過 AWS CLI來練習 4247
4. 建立 Aurora 叢集 4247
5. 設定您的結構描述 4248
6. 匯入資料 4249
7. 移植 SQL 程式碼 4249
8. 指定組態設定 4250
9. 連線到 Aurora 4251
10. 執行工作負載 4252
11. 測量效能 4252
12. 演練 Aurora 高可用性 4255
13. 後續作業 4256
安全 4258

資料庫身分驗證 4260
密碼身分驗證 4261
IAM 資料庫身分驗證 4261
Kerberos 身分驗證 4261
使用 Aurora 和 Secrets Manager 進行密碼管理 4263
區域和版本可用性 4263
限制 4263
概觀 4264
優勢 4264
許可 4265
強制執行 Aurora 管理 4266
管理資料庫叢集的密碼 4267
為資料庫叢集輪換秘密 4270
檢視資料庫叢集的秘密詳細資訊 4272
資料保護 4275
資料加密 4276
網際網路流量隱私權 4303
身分與存取管理 4304
目標對象 4304
使用身分來驗證 4305
使用政策管理存取權 4307
Amazon Aurora 如何搭配 IAM 運作 4308
身分型政策範例 4314
AWS 受管政策 4335
政策更新 4341
預防跨服務混淆代理人 4349
IAM 資料庫身分驗證 4351
故障診斷 4397
日誌記錄和監控 4399
法規遵循驗證 4401
彈性 4402
備份和還原 4402
複寫 4402
容錯移轉 4403
基礎設施安全 4404
Security groups (安全群組) 4404

Public accessibility (公開存取性) 4404
VPC 端點 (AWS PrivateLink) 4405
考量 1007
可用性 1007
建立介面 VPC 端點 1007
建立 VPC 端點政策 1008
安全最佳實務 4408
使用安全群組控制存取 4409
VPC 安全群組概觀 4409
安全群組案例 4410
建立 VPC 安全群組 4412
與資料庫叢集建立關聯 4412
主要使用者帳戶權限 4412
服務連結角色 4415
Amazon Aurora 的服務連結角色許可 4415
Amazon RDS Beta 的服務連結角色許可 4418
Amazon RDS 預覽的服務連結角色 4419
搭配使用 Amazon Aurora 與 Amazon VPC 4420
在 VPC 中使用資料庫叢集 4420
在 VPC 中存取資料庫叢集的案例 4435
教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4) 4441
教學課程：建立要與資料庫叢集搭配使用的 (VPC)(雙堆疊模式) 4448
配額和條件限制 4458
Amazon Aurora 中的配額 4458
Amazon Aurora 中的命名限制 4463
Amazon Aurora 大小限制 4463
疑難排解 4465
無法連線至資料庫執行個體 4465
測試資料庫執行個體連線 4467
對連線身分驗證進行故障診斷 4468
安全問題 4468
錯誤訊息「無法擷取帳戶屬性，某些主控台功能可能受損。」 4468
重新設定資料庫執行個體擁有者密碼 4468
資料庫執行個體停機或重新開機 4469
參數變更未生效 4469
Aurora 可用記憶體問題 4470

Aurora MySQL 複寫問題 4471
診斷和解決僅供讀取複本之間的延遲 4471
診斷和解決 MySQL 讀取複寫失敗 4473
複寫已停止錯誤 4474
僅供讀取複本複寫無法初始化中繼資料結構 4474
Amazon RDS API 參考 4476
使用查詢 API 4476
查詢參數 4476
查詢請求身分驗證 4476
對應用程式進行故障診斷 4477
擷取錯誤 4477
對秘訣進行故障診斷 4477
文件歷史記錄 4479
AWS 詞彙表 4550
..... mmmmdli

什麼是 Amazon Aurora？

Amazon Aurora (Aurora) 為全受管關聯式資料庫引擎，可與 MySQL 和 PostgreSQL 相容。您已知道 MySQL 和 PostgreSQL 如何將高階商用資料庫的速度和可靠性，與開源資料庫的簡易性和成本效益結合在一起。您目前搭配現有 MySQL 和 PostgreSQL 資料庫使用的程式碼、工具和應用程式，可用來配合 Aurora 使用。透過一些工作負載，Aurora 可提供 MySQL 最多五倍的輸送量和 PostgreSQL 最多三倍的輸送量，而不需變更您的多數現有應用程式。

Aurora 包括高效能的儲存子系統。其 MySQL 和 PostgreSQL 相容的資料庫引擎會自訂為善用該快速分散式儲存。基礎儲存體會視需要自動成長。Aurora 叢集磁碟區的大小最多可增長至 128 tebibytes (TiB)。Aurora 也會自動化並標準化資料庫叢集和複寫，這通常是資料庫設定和管理中最具挑戰性的層面。

Aurora 是受管資料庫服務 Amazon Relational Database Service (Amazon RDS) 的一部分。Amazon RDS 可以讓雲端中關聯式資料庫的設定、操作和擴展更加簡單。如果您尚未熟悉 Amazon RDS，請參閱《Amazon Relational Database Service 使用者指南》。若要進一步了解 Amazon Web Services 上可用的各種資料庫選項，請參閱在 AWS上選擇適合您組織的資料庫。

主題

- Amazon RDS 共同的責任模型
- Amazon Aurora 如何搭配 Amazon RDS 運作
- Amazon Aurora 資料庫叢集
- Amazon Aurora 版本
- 區域和可用區域
- AWS 區域 和 Aurora 資料庫引擎在 Amazon Aurora 中支援的功能
- Amazon Aurora 端點連線
- Amazon Aurora 資料庫執行個體類別
- Amazon Aurora 儲存體
- Amazon Aurora 可靠性
- Amazon Aurora 安全性
- Amazon Aurora 的高可用性
- 以 Amazon Aurora 進行複寫
- Aurora 資料庫執行個體計費
- AWS 免費方案上的 Amazon Aurora

Amazon RDS 共同的責任模型

Amazon RDS 負責託管資料庫執行個體和資料庫叢集的軟體元件和基礎設施。您則負責查詢調校，這是調整 SQL 查詢以提高效能的程序。查詢效能大幅取決於資料庫設計、資料大小、資料分佈、應用程式工作負載和查詢模式，而這些模式可能差異很大。監控和調校是您針對 RDS 資料庫的高度個人化程序。您可以使用 Amazon RDS Performance Insights 和其他工具識別出有問題的查詢。

Amazon Aurora 如何搭配 Amazon RDS 運作

下列幾點說明 Amazon Aurora 如何與 Amazon RDS 中提供的標準 MySQL 和 PostgreSQL 引擎相關：

- 透過 Amazon RDS 設定新的資料庫伺服器時，請選擇 Aurora MySQL 或 Aurora PostgreSQL 做為資料庫引擎選項。
- Aurora 利用熟悉的 Amazon Relational Database Service (Amazon RDS) 功能來進行管理。Aurora 使用 Amazon RDS AWS 管理主控台 介面、 AWS CLI 命令和 API 操作來處理例行資料庫任務，例如佈建、修補、備份、復原、故障偵測和修復。
- Aurora 管理操作通常涉及資料庫伺服器的整個叢集，而透過複寫同步化的是這些叢集，而非個別資料庫執行個體。自動叢集、複寫和儲存配置使得設定、操作及擴展最大的 MySQL 和 PostgreSQL 部署既簡單且經濟實惠。
- 您可以將 Amazon RDS for MySQL 和 Amazon RDS for PostgreSQL 的資料帶入至 Aurora，方法為建立並還原快照，或設定單向複寫。您可以使用按鈕遷移工具，以將現有的 RDS for MySQL 和 RDS for PostgreSQL 應用程式轉換為 Aurora。

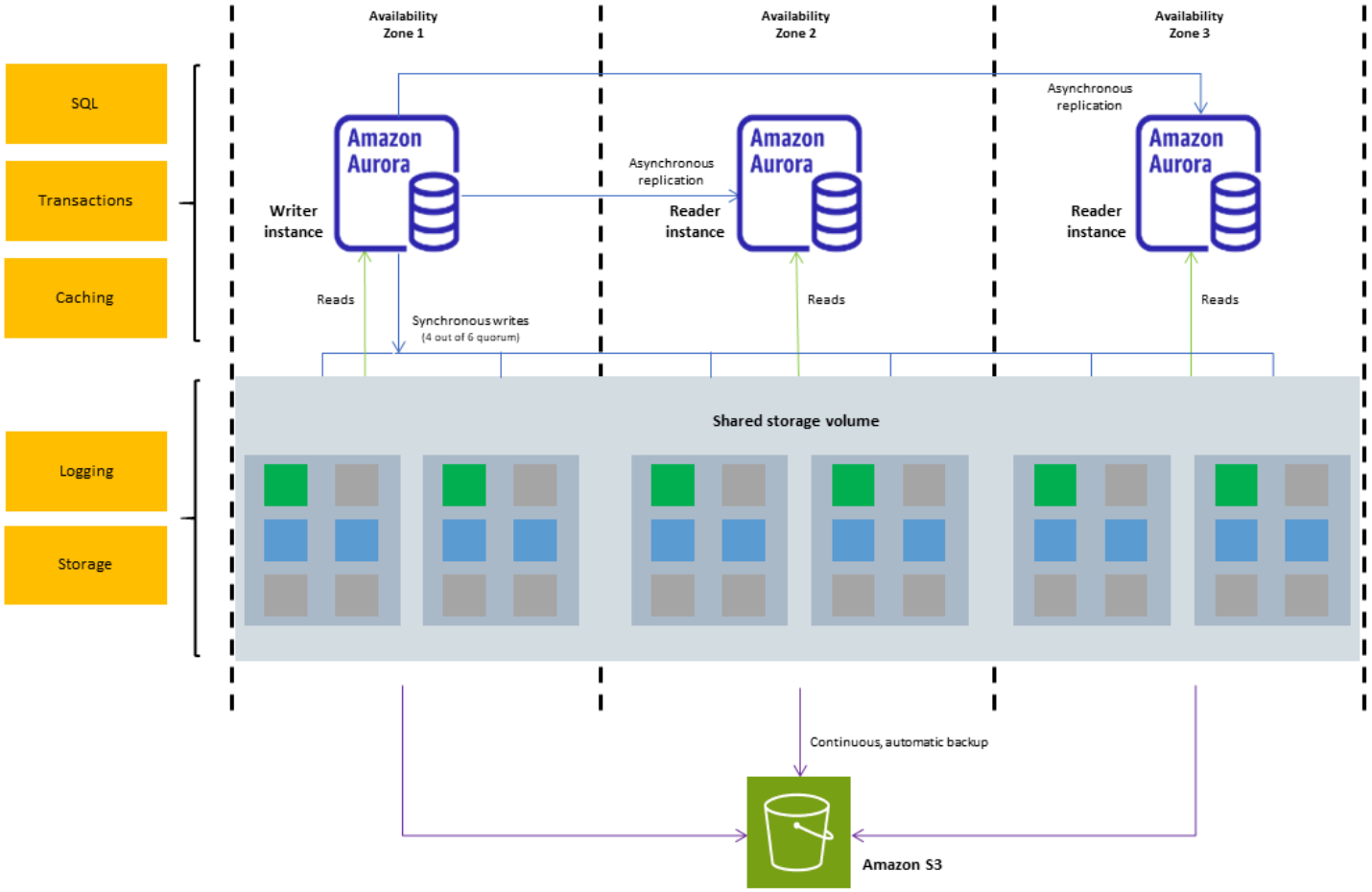
使用 Amazon Aurora 之前，請完成設定您的 Amazon Aurora 環境中的步驟，然後在 Amazon Aurora 資料庫叢集中檢閱 Aurora 的概念和功能。

Amazon Aurora 資料庫叢集

Amazon Aurora 資料庫叢集包含一或多個資料庫執行個體，以及一個為這些資料庫執行個體管理資料的叢集磁碟區。Aurora 叢集磁碟區是虛擬資料庫儲存體磁碟區，可跨多個可用性區域，並且每個可用性區域具有資料庫叢集資料的複本。Aurora 資料庫叢集是由兩個類型的資料庫執行個體組成：

- 主要資料庫 (寫入器) 執行個體 – 支援讀寫操作，並對叢集磁碟區執行所有資料修改。每個 Aurora 資料庫叢集具有一個主要資料庫執行個體。
- Aurora 複本 (讀取器資料庫執行個體) – 連接到與主要資料庫執行個體相同的儲存磁碟區，但僅支援讀取操作。除了主要資料庫執行個體，每個 Aurora 資料庫叢集最多可具有 15 個 Aurora 複本。透過將 Aurora 複本置放在不同的可用區域內，來維持高可用性。若主要資料庫執行個體無法使用，Aurora 會自動容錯移轉至 Aurora 複本。您可以指定 Aurora 複本的容錯移轉優先順序。Aurora 複本也可以從主要資料庫執行個體卸載讀取工作負載。

下圖說明叢集磁碟區、寫入器資料庫執行個體和 Aurora 資料庫叢集中讀取器資料庫執行個體之間的關係。



> **Note**
>
> 上述資訊適用於所有 Aurora 資料庫叢集 - 佈建、平行查詢、Aurora 全球資料庫、Aurora Serverless、Aurora MySQL 相容和 Aurora PostgreSQL 相容。

Aurora 資料庫叢集分離了運算容量與儲存體。例如，只有單一資料庫執行個體的 Aurora 組態仍然是一個叢集，因為基礎儲存體磁碟區涉及多個分散在多個可用區域 (AZ) 中的儲存體節點。

Aurora 資料庫叢集中的輸入/輸出 (I/O) 操作以相同的方式計數，無論它們位於寫入器還是讀取器資料庫執行個體上。如需更多詳細資訊，請參閱 Amazon Aurora 資料庫叢集的儲存組態。

Amazon Aurora 版本

使用 Amazon Aurora，您可以選擇最符合您應用程式需求的支援關聯式資料庫引擎，同時保持與基礎引擎的相容性。Aurora 會從支援的引擎重複使用資料庫引擎程式碼，因此您可以利用這些引擎的現有技能、工具和程式庫。建立叢集時，您可以指定要使用的 Amazon Aurora 資料庫引擎版本。您選擇的版本會決定相容性和可用功能。

本文件說明 Amazon Aurora 與對應資料庫引擎之間的共同點和差異。透過此資訊，您可以判斷應選取的軟體版本，以及如何確認每個版本中可用的功能和錯誤修正。您也可以使用此參考來判斷適當的升級節奏並規劃升級。

Amazon Aurora 資料庫叢集支援的資料庫引擎

Amazon Aurora 提供下列關聯式資料庫。Aurora 會重複使用程式碼，並維持與基礎資料庫引擎的相容性。然而，Aurora 有唯一的版本號碼、發行週期和版本棄用的時間表。每個新的 Aurora 版本都隨附版本備註，其列出適用於每個版本的新功能、修正，以及其他變更和增強功能。

Aurora 資料庫	使用者指南	可用版本	版本備註
Amazon Aurora MySQL-Compatible Edition	使用 Amazon Aurora MySQL	Amazon Aurora MySQL 的資料庫引擎更新	Aurora MySQL 版本備註
Amazon Aurora PostgreSQL-Compatible Edition	使用 Amazon Aurora PostgreSQL	Amazon Aurora PostgreSQL 的資料庫引擎更新	Aurora PostgreSQL 版本備註

為您的資料庫叢集指定 Amazon Aurora 資料庫版本

當您使用 AWS 管理主控台 或 中的建立資料庫操作 AWS CLI，或使用 `CreateDBCluster` API 操作建立新的資料庫叢集時，您可以指定任何目前可用的 Aurora 資料庫版本 （主要或次要）。

若要了解如何建立 Aurora 叢集，請參閱 建立 Amazon Aurora 資料庫叢集。若要了解如何變更現有 Aurora 叢集的版本，請參閱 修改 Amazon Aurora 資料庫叢集。

> **Note**
>
> 並非每個 Aurora 資料庫版本在每個 AWS 區域都可供使用。若要進一步了解 區域和每個 中的可用版本 AWS 區域，請參閱 區域和可用區域和 Aurora 全球資料庫的支援區域和資料庫引擎。

Amazon Aurora 版本控制

Amazon Aurora 版本與其相容的上游社群資料庫不同。為了協助您維持應用程式相容性並利用最新的資料庫引擎功能，下列各節說明 Aurora 版本控制慣例，以及 Aurora 版本如何映射至其各自的社群資料庫。

如需 Amazon Aurora 上可用的關聯式資料庫清單，請參閱 Amazon Aurora 資料庫叢集支援的資料庫引擎。

社群資料庫和 Aurora 之間的版本號碼差異

每個 Amazon Aurora 版本都與其對應社群資料庫的特定版本相容。您可以使用 `version` 函數找到資料庫的社群版本和使用 `aurora_version` 函數找到 Aurora 版本。

下列範例示範如何尋找 Aurora MySQL 和 Aurora PostgreSQL 資料庫的社群版本。

Aurora MySQL

`version` 函數會傳回 Aurora MySQL 資料庫的社群版本。

```
mysql> select version();
```

輸出範例：

```
+-------------------+
|   version()       |
+-------------------+
|  8.0.32           |
+-------------------+
```

而 `aurora_version` 函數會傳回 Aurora 版本：

```
mysql> select aurora_version(), @@aurora_version;
```

輸出範例：

```
+------------------+------------------+
| aurora_version() | @@aurora_version |
+------------------+------------------+
| 3.05.2           | 3.05.2           |
+------------------+------------------+
```

Aurora PostgreSQL

`version` 函數會傳回 Aurora PostgreSQL 資料庫的社群版本。

```
postgres=> select version();
```

輸出範例：

```
-------------------------------------------------------------------------------
PostgreSQL 11.7 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.9.3, 64-bit
(1 row)
```

而 `aurora_version` 函數會傳回 Aurora 版本：

```
postgres=> select aurora_version();
```

輸出範例：

```
aurora_version
----------------
3.2.2
```

如需詳細資訊，請參閱使用 SQL 檢查 Aurora MySQL 版本及識別 Amazon Aurora PostgreSQL 版本。

預設 Amazon Aurora 版本

預設版本是當您不手動指定目標引擎版本時，Aurora 自動為資料庫建立或升級選擇的版本。例如，下列命令顯示 Aurora PostgreSQL 的預設引擎版本 (包含範例輸出)。

```
aws rds describe-db-engine-versions \
    --engine aurora-postgresql \
    --default-only \
```

```
    --query 'DBEngineVersions[0].EngineVersion' \
    --output text

16.4
```

每個主要版本都有對應的預設次要版本。因此，Aurora PostgreSQL 16 的預設次要版本為 16.n，當 Aurora 發行新的預設次要版本時，版本編號 n 會變更。通常，Aurora 會針對每年每個主要版本發行兩個預設次要版本。下列 bash shell 指令碼顯示一組 Aurora PostgreSQL 主要版本的預設次要版本 (包含範例輸出)。

```
for major in 16 15 14 13 12 11; do
  echo -n "Default for Aurora PostgreSQL major version $major: "
  aws rds describe-db-engine-versions \
    --engine aurora-postgresql \
    --engine-version "$major" \
    --default-only \
    --query 'DBEngineVersions[0].EngineVersion' \
    --output text
done

Default for Aurora PostgreSQL major version 16: 16.4
Default for Aurora PostgreSQL major version 15: 15.8
Default for Aurora PostgreSQL major version 14: 14.13
Default for Aurora PostgreSQL major version 13: 13.16
Default for Aurora PostgreSQL major version 12: 12.20
Default for Aurora PostgreSQL major version 11: 11.21
```

如果您為 Aurora 資料庫叢集啟用自動次要版本升級，Aurora 會使用預設次要版本或指定主要版本的較新次要版本。例如，如果 Aurora PostgreSQL 15 的預設次要版本為 15.8，而且也提供較新的 15.10 版，則 Aurora 可以自動升級至 15.8 或 15.10。

Amazon Aurora 主要版本

Aurora 版本使用 *major*.*minor*.*patch* 結構描述。Aurora 主要版本是指與 Aurora 相容的 MySQL 或 PostgreSQL 社群主要版本。Aurora MySQL 和 Aurora PostgreSQL 主要版本在標準支援下，至少在對應的社群版本生命週期結束前會保持可用。您可以在 Aurora 標準支援結束日期之後繼續付費執行主要版本。如需詳細資訊，請參閱 使用 Amazon Aurora 的 Amazon RDS 延伸支援 和 Amazon Aurora 定價。

如需 Aurora MySQL 和 Aurora PostgreSQL 主要版本和發佈行事曆的詳細資訊，請參閱各版本備註中的下列頁面：

- Aurora MySQL 主要版本的發佈行事曆
- Aurora PostgreSQL 主要版本的發佈行事曆

您也可以執行 describe-db-major-engine-versions AWS CLI 命令或使用 DescribeDBMajorEngineVersions RDS API 操作，以檢視主要引擎版本支援日期的相關資訊。

> **Note**
>
> 針對 Aurora MySQL 第 2 版的 Amazon RDS 延長支援從 2024 年 11 月 1 日開始，但直到 2024 年 12 月 1 日才會收費。在 2024 年 11 月 1 日至 11 月 30 日期間，所有 Aurora MySQL 第 2 版資料庫叢集都涵蓋在 Amazon RDS 延長支援的範圍內。如需詳細資訊，請參閱所選 Aurora 版本的 Amazon RDS 延長支援。

Amazon Aurora 主要版本可用的時間會維持多久

Amazon Aurora 主要版本至少在對應的社群版本的生命週期結束前都會保持可用。您可以使用 Aurora 標準支援結束日期規劃您的測試和升級週期。這些日期表示可能需要升級到較新版本的最早日期。如需日期的詳細資訊，請參閱 Amazon Aurora 主要版本。

在 Aurora 要求您升級至更新的主要版本並協助您計劃之前，您會至少提前 12 個月前收到提醒。提醒會傳達下列有關升級程序的資訊。

- 特定里程碑的時間
- 對於資料庫叢集的影響
- 建議的動作

我們建議在將叢集升級至新的主要版本之前，先徹底對應用程式進行新資料庫版本的測試。

在主要版本達到 Aurora 結束標準支援後，任何仍在執行舊版的資料庫叢集都會在排定的維護時段期間自動升級至延長支援版本。可能需要支付延長支援費用。如需 Amazon RDS 延長支援的詳細資訊，請參閱使用 Amazon RDS 延長支援。

Amazon Aurora 次要版本

Aurora 版本使用 *major*.*minor*.*patch* 結構描述。Aurora 次要版本為服務提供增量式社群和 Aurora 特定的改進，例如新功能和修正。

如需 Aurora MySQL 和 Aurora PostgreSQL 次要版本和發佈行事曆的詳細資訊，請參閱各版本備註中的下列頁面：

- Aurora MySQL 次要版本的發佈行事曆
- Aurora PostgreSQL 次要版本的發佈行事曆

下列各節說明您可以預期 Aurora 次要版本之節奏和生命週期的詳細資訊。

主題

- Amazon Aurora 次要版本發行的頻率
- Amazon Aurora 次要版本可用的時間會維持多久

Amazon Aurora 次要版本發行的頻率

一般來說，我們每季都會發佈 Amazon Aurora 次要版本。發行排程可能會因選擇其他功能或修正程式而有所不同。

Amazon Aurora 次要版本可用的時間會維持多久

通常，Amazon Aurora會讓特定主要版本的每個次要版本供應時間應持續至少 12 個月。在此期間結束時，Aurora 可能會將資料庫升級至預設次要版本或較新版本。Aurora 會在任何執行較舊次要版本之資料庫叢集的排定維護時段期間開始升級。

在某些情況下，Aurora 可能會比平常的 12 個月期間更早取代特定主要版本的次要版本。原因可能包括重大安全問題或主要版本的終止支援日期。

在開始對生命即將結束的次要版本進行自動升級之前，Aurora 通常會提前三個月發送提醒。Aurora 會詳細說明下列升級程序的相關資訊。

- 特定里程碑的時間
- 對於資料庫叢集的影響
- 建議的動作

三個月以內的通知會描述安全性問題等需要更快採取行動的重大事項。

如果已啟用自動次要版本升級設定，您會收到提醒，但不會收到 RDS 事件通知。在強制升級截止日期過後，Aurora 會在維護時段內升級您的資料庫。

如果未啟用自動次要版本升級設定，則會收到提醒和 Amazon RDS 資料庫叢集事件通知，其類別為 `maintenance` 且 ID 為 `RDS-EVENT-0156`。Aurora 會在下一個維護時段升級您的資料庫。

請注意，在次要版本達到 Aurora 結束標準支援後，將不會針對該次要版本發行其他修補程式版本。若要接收重大錯誤修正或 CVE，您必須升級至具有標準支援的次要版本。

如需自動次要版本升級的詳細資訊，請參閱 Aurora 資料庫叢集的自動次要版本升級。

Amazon Aurora 修補程式版本

Aurora 版本使用 *major*.*minor*.*patch* 結構描述。Aurora 修補程式版本包含在初始版本後新增至次要版本的重要修正 (例如，Aurora MySQL 3.04.0、3.04.1、...、3.04.3)。雖然每個新的次要版本都提供新的 Aurora 功能，但特定次要版本中的新修補程式版本主要是用來解決重要問題。

如需修補的詳細資訊，請參閱 維持 為 Amazon Aurora 資料庫叢集。

升級 Amazon Aurora 資料庫叢集

透過 Amazon Aurora，您可以控制和測試資料庫叢集的升級。Amazon Aurora 提供自動次要版本升級、手動升級控制、必要升級和預先升級測試的選項。您可以讓叢集保持在最新的次要版本、延遲非關鍵升級、強制嚴重問題的升級，以及在非生產環境中驗證升級行為。下列各節詳細說明如何使用這些功能管理和測試 Aurora 資料庫叢集升級。

Aurora 的自動次要版本升級

自動次要版本升級會定期將資料庫更新為最新的資料庫引擎版本。不過，升級不一定包含最新的資料庫引擎版本。如果您需要在特定時間將資料庫維持在特定版本上，建議您根據所需的排程手動升級至所需的資料庫版本。如果發生重大安全問題或版本達到終止支援的日期，Amazon Aurora 可能會套用次要版本升級，即使您尚未啟用自動次要版本升級選項，也是如此。如需詳細資訊，請參閱適用於您特定資料庫引擎的升級文件。

請參閱 升級 Aurora MySQL 資料庫叢集的次要版本或修補程式層級 和 執行次要版本升級。

您可以透過為 Aurora 叢集中的每個資料庫執行個體開啟 Auto minor version upgrade (自動次要版本升級)，來使 Aurora 次要版本保持在最新狀態。Aurora 僅在叢集中的所有資料庫執行個體皆已開啟此設定時才會執行自動升級。

如果資料庫叢集的自動次要版本升級為是，Aurora 會自動升級至預設次要版本或較新的次要版本。例如，如果 Aurora PostgreSQL 15 的預設次要版本為 15.8，且 15.10 版存在，則自動升級的目標可以是 15.8 或 15.10。

Aurora 通常會為已啟用自動次要版本更新的資料庫叢集排程一年自動升級兩次。這些升級會在您為叢集指定的維護時段期間發生。如需詳細資訊，請參閱Aurora 資料庫叢集的自動次要版本升級。

自動次要版本升級會透過 Amazon RDS 資料庫叢集事件預先傳達，該事件的類別為 `maintenance`，而其 ID 為 `RDS-EVENT-0156`。如需詳細資訊，請參閱Aurora 的 Amazon RDS 事件類別和事件訊息。

手動控制資料庫叢集升級至新版本

如果您已啟用自動次要版本升級設定，Aurora 會自動將您的資料庫叢集升級至預設次要版本或較新的次要版本。Aurora 通常會為已啟用自動次要版本更新設定的資料庫叢集排程一年自動升級兩次。這些升級會在客戶指定的維護時段期間啟動。

若要關閉自動次要版本升級，請在 Aurora 叢集內的任一資料庫執行個體上停用自動次要版本升級。只有在叢集中的所有資料庫執行個體都已啟用設定時，Aurora 才會執行自動次要版本升級。

> **Note**
>
> 對於次要版本生命週期結束等強制性升級，即使停用自動次要版本升級設定，Aurora 仍會升級資料庫叢集。您會收到提醒，但沒有 RDS 事件通知。在強制升級截止日期過後，Aurora 會在維護時段內升級您的叢集。

由於主要版本升級涉及一定的相容性風險，因此這些升級不會自行啟動。您必須起始這些操作，除非有主要版本棄用。我們建議在將叢集升級至主要版本之前，先徹底對應用程式進行新資料庫版本的測試。

如需將資料庫叢集升級至新 Aurora 主要版本的詳細資訊，請參閱 升級 Amazon Aurora MySQL 資料庫叢集 和 升級 Amazon Aurora PostgreSQL 資料庫叢集。

必要的 Amazon Aurora 升級

針對特定的關鍵修正，Aurora 可能會在相同的次要版本中執行達到某個修補層級的受管升級。在此情況下，即使已關閉自動次要版本升級，Aurora 仍會升級您的叢集。在這樣做之前，Aurora 會先說明詳細的升級程序。詳細資訊包括特定里程碑的時間、對資料庫叢集的影響，以及建議的動作。此受管升級在叢集維護時段內自動發生。

在升級之前，使用新的 Aurora 版本測試資料庫叢集

您可以測試升級程序，以及新版本如何與應用程式和工作負載搭配運作。使用下列其中一種方法：

- 使用 Amazon Aurora 快速資料庫複製功能複製叢集。在新叢集上執行升級和任何升級後測試。
- 從叢集快照進行還原，以建立新的 Aurora 叢集。您可以自行從現有 Aurora 叢集建立叢集快照。Aurora 也會自動為每個叢集建立定期快照。然後，您可以啟動新叢集的版本升級。在決定是否要升級原始叢集之前，您可以先對叢集的升級副本進行實驗。

如需進一步了解建立用於測試之新叢集的方法，請參閱 複製 Amazon Aurora 資料庫叢集的一個磁碟區 和 建立資料庫叢集快照。

Amazon Aurora 版本支援

如果您的 Amazon Aurora 資料庫對特定資料庫引擎行為有複雜的相依性，建議您在升級至較新的資料庫引擎版本之前進行廣泛的測試。提供長期支援選項，讓您可以在選取的 Aurora 引擎版本上維護資料庫叢集，即使已由較新版本取代。下列各節說明 Aurora 資料庫叢集的長期支援選項。

對選定的 Amazon Aurora 次要版本的長期支援

對於每個 Aurora 主要版本，某些次要版本被指定為長期支援 (LTS) 版本，並且提供時間至少為期三年。也就是說，每個主要版本至少提供一個次要版本，為期超過常見的 12 個月的時間。一般而言，Aurora 會在這段期間結束前六個月提醒您。Aurora 會傳達下列有關升級程序的資訊。

- 特定里程碑的時間
- 對於資料庫叢集的影響
- 建議的動作

六個月以內的通知會傳達安全性問題等必須更快採取行動的重大事項。

LTS 次要版本只包含關鍵修正 (透過修補程式版本)。LTS 版本不包含推出後發行的新功能。每年一次，在 LTS 次要版本上執行的資料庫叢集會修補至 LTS 版本的最新修補程式版本。Aurora 會修補您的叢集，讓您受益於累積的安全性和穩定性修正。如果有關鍵修復，Aurora 可能會更頻繁地修補 LTS 次要版本 (例如安全性)。

> **Note**
>
> 若要在 LTS 次要版本的生命週期內繼續使用，請務必停用資料庫執行個體的自動次要版本升級。為避免自動從 LTS 次要版本升級您的資料庫叢集，請清除 Aurora 叢集中所有資料庫執行個體上的啟用自動次要版本升級核取方塊。

如需所有 Aurora LTS 版本的版本編號，請參閱 Aurora MySQL 長期支援 (LTS) 版本 和 使用 Aurora PostgreSQL 長期支援 (LTS) 版本。

所選 Aurora 版本的 Amazon RDS 延長支援

使用 Amazon RDS 延長支援，就可以在 Aurora 標準支援結束日期過後，繼續在主要引擎版本上執行您的資料庫，但須另付費用。在 RDS 延長支援期間，Amazon RDS 將根據國家漏洞資料庫 (NVD) CVSS 嚴重性評級所定義的嚴重和高 CVE 提供修補程式。如需詳細資訊，請參閱使用 Amazon Aurora 的 Amazon RDS 延伸支援。

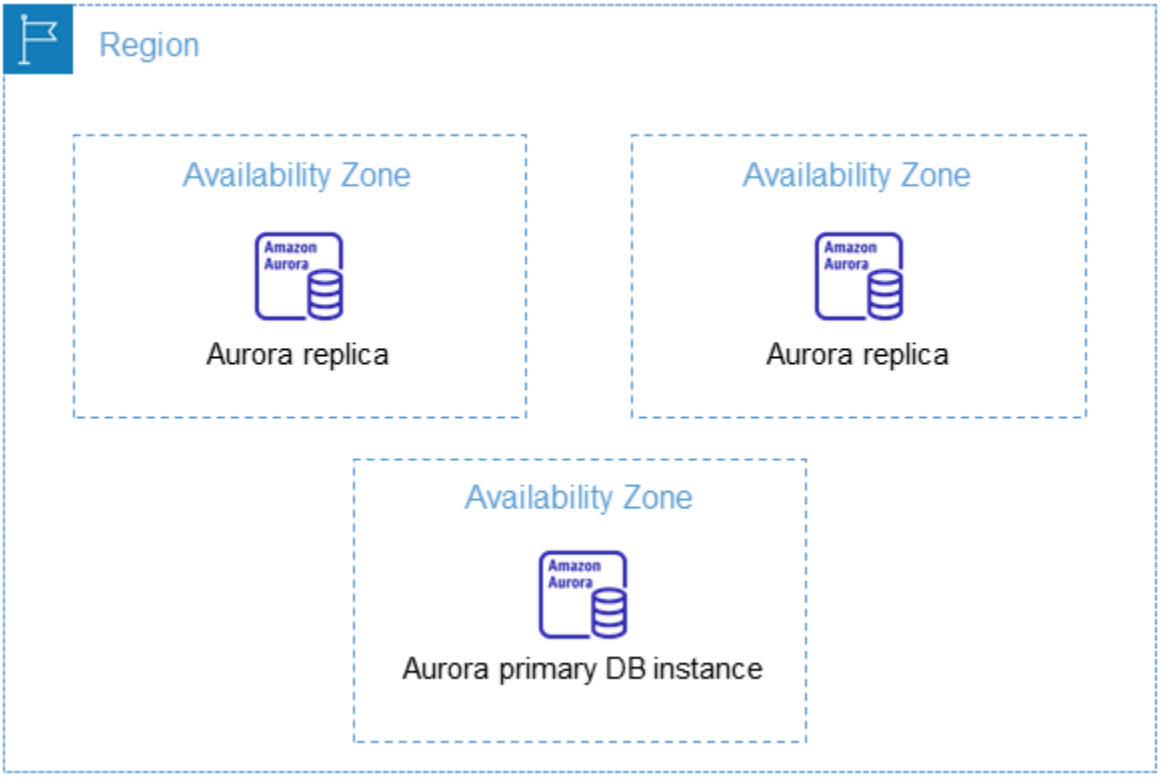
延長支援僅在特定 Aurora 版本上提供。如需詳細資訊，請參閱Amazon Aurora 主要版本。

區域和可用區域

Amazon 雲端運算資源託管於全球的多個地點。這些位置由 AWS 區域和可用區域組成。各個 AWS 區域為獨立的地理區域。每個 AWS 區域都有多個隔離的位置，稱為可用區域。

> **Note**
>
> 如需尋找 AWS 區域可用區域的詳細資訊，請參閱 Amazon EC2 文件中的描述您的可用區域。

Amazon 營運的尖端資料中心均為高度可用。儘管故障極為少見，但仍可能影響相同位置內資料庫執行個體的可用性。若您將所有資料庫執行個體都託管於某一位置，一旦該位置受故障影響，所有資料庫執行個體都將無法使用。



請務必記住，每個 AWS 區域都是完全獨立的。您啟動的任何 Amazon RDS 活動（例如，建立資料庫執行個體或列出可用的資料庫執行個體）只會在您目前的預設 AWS 區域中執行。您可以在 主控台中或透過設定`AWS_DEFAULT_REGION`環境變數來變更預設 AWS 區域。或者，您可以使用 `--region` 參數搭配 AWS Command Line Interface () 來覆寫它AWS CLI。如需詳細資訊，請參閱設定 AWS Command Line Interface，特別是有關環境變數和命令列選項的章節。

Amazon RDS 支援稱為 的特殊 AWS 區域 AWS GovCloud (US)。這些區域旨在讓美國政府機構和客戶將更敏感的工作負載移到雲端。 AWS GovCloud (US) 區域處理美國政府的特定法律與法規要求。如需詳細資訊，請參閱什麼是 AWS GovCloud (US)？

若要在特定區域中建立或使用 Amazon RDS 資料庫執行個體 AWS ，請使用對應的區域服務端點。

> **Note**
>
> Aurora 不支援「Local Zones」。

AWS 區域

每個 AWS 區域旨在與其他 AWS 區域隔離。此設計可達到最高的容錯能力與穩定性。

當您檢視資源時，只會看到與您指定的 AWS 區域繫結的資源。這是因為 AWS 區域彼此隔離，而且我們不會自動跨 AWS 區域複寫資源。

區域可用性

當您使用命令列界面或 API 操作處理 Aurora 資料庫叢集時，請務必指定其區域端點。

主題

- Aurora MySQL 區域可用性
- Aurora PostgreSQL 區域可用性

Aurora MySQL 區域可用性

下表顯示 Aurora MySQL 目前可用的 AWS 區域，以及每個區域的端點。

區域名稱	區域	端點	通訊協定	
美國東部 (俄亥俄)	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
美國東部 (維吉尼亞北部)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	

區域名稱	區域	端點	通訊協定	
美國西部 (加利佛尼亞北部)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS	
美國西部 (奧勒岡)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS	
Africa (Cape Town)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS	
亞太區域 (香港)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	
亞太區域 (海德拉巴)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS	
亞太區域 (雅加達)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS	
亞太地區 (馬來西亞)	ap-southeast-5	rds.ap-southeast-5.amazonaws.com	HTTPS	
亞太區域 (墨爾本)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com	HTTPS	
亞太區域 (孟買)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS	
亞太區域 (紐西蘭)	ap-southeast-6	rds.ap-southeast-6.amazonaws.com	HTTPS	

區域名稱	區域	端點	通訊協定	
亞太區域 (大阪)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS	
亞太區域 (首爾)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS	
亞太區域 (新加坡)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS	
亞太區域 (雪梨)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS	
亞太區域 (台北)	ap-east-2	rds.ap-east-2.amazonaws.com	HTTPS	
亞太區域 (泰國)	ap-southeast-7	rds.ap-southeast-7.amazonaws.com	HTTPS	
亞太區域 (東京)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS	
加拿大 (中部)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS	
加拿大西部 (卡加利)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS	
歐洲 (法蘭克福)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS	
歐洲 (愛爾蘭)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS	

區域名稱	區域	端點	通訊協定	
歐洲 (倫敦)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS	
歐洲 (米蘭)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS	
歐洲 (巴黎)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	
歐洲 (西班牙)	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS	
歐洲 (斯德哥爾摩)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS	
歐洲 (蘇黎世)	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS	
以色列 (特拉維夫)	il-central-1	rds.il-central-1.amazonaws.com	HTTPS	
墨西哥 (中部)	mx-central-1	rds.mx-central-1.amazonaws.com	HTTPS	
中東 (巴林)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS	
中東 (阿拉伯聯合大公國)	me-central-1	rds.me-central-1.amazonaws.com	HTTPS	
南美洲 (聖保羅)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS	

區域名稱	區域	端點	通訊協定	
AWS GovCloud（美國東部）	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS	
AWS GovCloud（美國西部）	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS	

Aurora PostgreSQL 區域可用性

下表顯示 Aurora PostgreSQL 目前可用的 AWS 區域，以及每個區域的端點。

區域名稱	區域	端點	通訊協定	
美國東部 (俄亥俄)	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
美國東部 (維吉尼亞北部)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	
美國西部 (加利佛尼亞北部)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS	
美國西部 (奧勒岡)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS	
Africa (Cape Town)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS	
亞太區域 (香港)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	

區域名稱	區域	端點	通訊協定	
亞太區域 (海德拉巴)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS	
亞太區域 (雅加達)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS	
亞太地區 (馬來西亞)	ap-southeast-5	rds.ap-southeast-5.amazonaws.com	HTTPS	
亞太區域 (墨爾本)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com	HTTPS	
亞太區域 (孟買)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS	
亞太區域 (紐西蘭)	ap-southeast-6	rds.ap-southeast-6.amazonaws.com	HTTPS	
亞太區域 (大阪)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS	
亞太區域 (首爾)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS	
亞太區域 (新加坡)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS	
亞太區域 (雪梨)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS	

區域名稱	區域	端點	通訊協定	
亞太區域 (台北)	ap-east-2	rds.ap-east-2.amazonaws.com	HTTPS	
亞太區域 (泰國)	ap-southeast-7	rds.ap-southeast-7.amazonaws.com	HTTPS	
亞太區域 (東京)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS	
加拿大 (中部)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS	
加拿大西部 (卡加利)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS	
歐洲 (法蘭克福)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS	
歐洲 (愛爾蘭)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS	
歐洲 (倫敦)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS	
歐洲 (米蘭)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS	
歐洲 (巴黎)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	
歐洲 (西班牙)	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS	
歐洲 (斯德哥爾摩)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS	

區域名稱	區域	端點	通訊協定	
歐洲 (蘇黎世)	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS	
以色列 (特拉維夫)	il-central-1	rds.il-central-1.amazonaws.com	HTTPS	
墨西哥 (中部)	mx-central-1	rds.mx-central-1.amazonaws.com	HTTPS	
中東 (巴林)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS	
中東 (阿拉伯聯合大公國)	me-central-1	rds.me-central-1.amazonaws.com	HTTPS	
南美洲 (聖保羅)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS	
AWS GovCloud （美國東部）	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS	
AWS GovCloud （美國西部）	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS	

可用區域

可用區域是指位於特定 AWS 區域區域中的隔離區域。每個區域都有多個可用區域 (AZ)，旨在為該區域提供高可用性。可用 AWS 區域代碼加上字母識別符 （例如，`us-east-1a`) 來識別 AZ。如果選擇建立 VPC 和子網路，而非使用預設 VPC，則您需要在特定可用區域中定義每個子網路。在您建立

Aurora 資料庫叢集時，Aurora 會在 VPC 資料庫子網路群組內的一個子網路中建立主要執行個體。因此其會將該執行個體與 Aurora 選擇的特定 AZ 建立關聯。

每個 Aurora 資料庫叢集都會在 Aurora 從資料庫子網路群組中的可用區域自動選取的三個不同可用區域中託管其儲存體的副本。叢集中的每個資料庫執行個體都必須位於這三個可用區域中的其中一個。

在叢集中建立資料庫執行個體時，如果您未指定可用區域，則 Aurora 會自動為該執行個體選擇適當的可用區域。

使用 describe-availability-zones Amazon EC2 命令，描述為您的帳戶啟用的指定區域內的可用區域，如下所示。

```
aws ec2 describe-availability-zones --region region-name
```

例如，若要描述美國東部 (維吉尼亞北部) 區域 (us-east-1) 內針對您帳戶啟用的可用區域，請執行下列命令：

```
aws ec2 describe-availability-zones --region us-east-1
```

如需了解如何在建立叢集或新增執行個體時指定可用區域，請參閱 設定資料庫叢集的網路。

Amazon Aurora 資料庫叢集的本機時區

Amazon Aurora 資料庫叢集的時區預設為國際標準時間 (UTC)。您可以將您的資料庫叢集中執行個體的時區改為設定成應用程式的本機時區。

若要設定資料庫叢集的本機時區，請將時區參數設為支援的值。您可以在叢集參數群組中針對資料庫叢集設定此參數。

- 針對 Aurora MySQL，此參數的名稱為 `time_zone`。有關設定 `time_zone` 參數的最佳實務，請參閱 最佳化時間戳記操作。
- 針對 Aurora PostgreSQL，此參數的名稱為 `timezone`。

設定資料庫叢集的時區參數時，資料庫叢集中的所有執行個體會變更為使用新的本機時區。在某些情況下，其他 Aurora 資料庫叢集可能正在使用相同的叢集參數群組。若是如此，這些資料庫叢集中的所有執行個體也會變更為使用新的本機時區。如需叢集層級參數的詳細資訊，請參閱 Amazon Aurora 資料庫叢集和資料庫執行個體參數。

設定本機時區後，所有資料庫的新連線都會反映此變更。在某些狀況下，您可能會在變更本機時區時，已開啟資料庫的連線。若是如此，您在關閉連線並開啟新的連線之後，才會看見本機時區更新。

如果您要跨 AWS 區域複寫，複寫來源資料庫叢集和複本會使用不同的參數群組。參數群組對 AWS 區域是唯一的。若要對每個執行個體使用相同本機時區，您務必在複寫來源叢集和複本的參數群組中設定時區參數。

從資料庫叢集快照還原資料庫叢集時，本機時區會設為 UTC。您也可在還原作業完成後，將時區更新為本機時區。在某些情況下，您可能會將資料庫叢集還原至時間點。若是如此，該還原資料庫叢集的本機時區會使用還原資料庫叢集參數群組中的時區設定。

下表列出部分值，您可以將本機時區設為其中一個值。若要列出所有可用的時區，您可以使用下列 SQL 查詢：

- Aurora MySQL：`select * from mysql.time_zone_name;`
- Aurora PostgreSQL：`select * from pg_timezone_names;`

> **Note**
>
> 針對某些時區，可能不正確地報告某些日期範圍的時間值，如資料表中所述。針對澳洲時區，傳回的時區縮寫是過時的值，如資料表中所述。

時區	備註
Africa/Harare	此時區設定可能傳回不正確的值，從 28 Feb 1903 21:49:40 GMT 到 28 Feb 1903 21:55:48 GMT。
Africa/Monrovia	
Africa/Nairobi	此時區設定可能傳回不正確的值，從 31 Dec 1939 21:30:00 GMT 到 31 Dec 1959 21:15:15 GMT。
Africa/Windhoek	
America/Bogota	此時區設定可能傳回不正確的值，從 23 Nov 1914 04:56:16 GMT 到 23 Nov 1914 04:56:20 GMT。
America/Caracas	

時區	備註
America/Chihuahua	
America/Cuiaba	
America/Denver	
America/Fortaleza	在某些情況下，對於南美洲 (聖保羅) 區域中的資料庫叢集，時間未針對最近變更的巴西時區正確地顯示。若是如此，請將資料庫叢集的時區參數重設為 America/Fortaleza 。
America/Guatemala	
America/Halifax	此時區設定可能傳回不正確的值，從 27 Oct 1918 05:00:00 GMT 到 31 Oct 1918 05:00:00 GMT。
America/Manaus	如果您的資料庫叢集位於南美洲 (古雅巴) 時區，而且預期的時間未正確顯示最近變更的巴西時區，請將資料庫叢集的時區參數重設為 America/Manaus 。
America/Matamoros	
America/Monterrey	
America/Montevideo	
America/Noronha	
America/Phoenix	
America/Tijuana	
Asia/Ashgabat	

時區	備註
Asia/Baghdad	
Asia/Baku	
Asia/Bangkok	
Asia/Beirut	
Asia/Calcutta	
Asia/Kabul	
Asia/Karachi	
Asia/Kathmandu	
Asia/Muscat	此時區設定可能傳回不正確的值，從 31 Dec 1919 20:05:36 GMT 到 31 Dec 1919 20:05:40 GMT。
Asia/Riyadh	此時區設定可能傳回不正確的值，從 13 Mar 1947 20:53:08 GMT 到 31 Dec 1949 20:53:08 GMT。
Asia/Seoul	此時區設定可能傳回不正確的值，從 30 Nov 1904 15:30:00 GMT 到 07 Sep 1945 15:00:00 GMT。
Asia/Shanghai	此時區設定可能傳回不正確的值，從 31 Dec 1927 15:54:08 GMT 到 02 Jun 1940 16:00:00 GMT。
Asia/Singapore	
Asia/Taipei	此時區設定可能傳回不正確的值，從 30 Sep 1937 16:00:00 GMT 到 29 Sep 1979 15:00:00 GMT。
Asia/Tehran	
Asia/Tokyo	此時區設定可能傳回不正確的值，從 30 Sep 1937 15:00:00 GMT 到 31 Dec 1937 15:00:00 GMT。

時區	備註
Asia/Ulaanbaatar	
Atlantic/Azores	此時區設定可能傳回不正確的值，從 24 May 1911 01:54:32 GMT 到 01 Jan 1912 01:54:32 GMT。
Australia/Adelaide	此時區的縮寫會傳回為 CST 而不是 ACDT/ACST。
Australia/Brisbane	此時區的縮寫會傳回為 EST 而不是 AEDT/AEST。
Australia/Darwin	此時區的縮寫會傳回為 CST 而不是 ACDT/ACST。
Australia/Hobart	此時區的縮寫會傳回為 EST 而不是 AEDT/AEST。
Australia/Perth	此時區的縮寫會傳回為 WST 而不是 AWDT/AWST。
Australia/Sydney	此時區的縮寫會傳回為 EST 而不是 AEDT/AEST。
Brazil/East	
Canada/Saskatchewan	此時區設定可能傳回不正確的值，從 27 Oct 1918 08:00:00 GMT 到 31 Oct 1918 08:00:00 GMT。
Europe/Amsterdam	
Europe/Athens	
Europe/Dublin	
Europe/Helsinki	此時區設定可能傳回不正確的值，從 30 Apr 1921 22:20:08 GMT 到 30 Apr 1921 22:20:11 GMT。

時區	備註
Europe/Paris	
Europe/Prague	
Europe/Sarajevo	
Pacific/Auckland	
Pacific/Guam	
Pacific/Honolulu	此時區設定可能傳回不正確的值，從 21 May 1933 11:30:00 GMT 到 30 Sep 1945 11:30:00 GMT。
Pacific/Samoa	此時區設定可能傳回不正確的值，從 01 Jan 1911 11:22:48 GMT 到 01 Jan 1950 11:30:00 GMT。
US/Alaska	
US/Central	
US/Eastern	
US/East-Indiana	
US/Pacific	
UTC	

AWS 區域 和 Aurora 資料庫引擎在 Amazon Aurora 中支援的功能

Aurora MySQL 和 PostgreSQL 相容的資料庫引擎支援幾個 Amazon Aurora 和 Amazon RDS 功能和選項。支援會因每個資料庫引擎的特定版本以及 AWS 區域而有所不同。若要識別特定功能中的 Aurora 資料庫引擎版本支援和可用性 AWS 區域，您可以使用下列章節。

其中一些功能僅為 Aurora 功能。例如，Aurora Serverless、Aurora 全域資料庫以及與 AWS 機器學習服務整合的支援不受 Amazon RDS 支援。Amazon RDS Proxy 等其他功能受 Amazon Aurora 和 Amazon RDS 支援。

支援的區域和資料庫引擎

- 資料表慣例
- 藍/綠部署支援的區域和 Aurora 資料庫引擎
- 叢集儲存體組態的支援區域和 Aurora 資料庫引擎
- 資料庫活動串流的支援區域和 Aurora 資料庫引擎
- 將叢集資料匯出至 Amazon S3 的支援區域和 Aurora 資料庫引擎
- 將快照資料匯出至 Amazon S3 的支援區域和 Aurora 資料庫引擎
- Aurora 全球資料庫的支援區域和資料庫引擎
- IAM 資料庫身分驗證的支援區域和 Aurora 資料庫引擎
- Kerberos 身分驗證的支援區域和 Aurora 資料庫引擎
- Aurora 機器學習的支援區域和資料庫引擎
- Performance Insights 的支援區域和 Aurora 資料庫引擎
- 零 ETL 整合的支援區域和 Aurora 資料庫引擎
- Amazon RDS Proxy 的支援區域和 Aurora 資料庫引擎
- Secrets Manager 整合的支援區域和 Aurora 資料庫引擎
- Aurora Serverless v2 支援的區域和 Aurora 資料庫引擎
- Aurora Serverless v1
- RDS 資料 API 的支援區域和 Aurora 資料庫引擎
- 支援零停機時間修補 (ZDP) 的支援區域和 Aurora 資料庫引擎
- 支援的區域 Aurora PostgreSQL Limitless Database
- Aurora 引擎原生功能的支援區域和資料庫引擎

資料表慣例

功能區段中的資料表會使用下列模式來指定版本號碼和支援層級：

- x.y 版 – 僅支援特定版本。
- x.y 版和更新版本 – 此特定版本及其主要版本所有更新的次要版本均受支援。例如，「10.11 版和更新版本」表示支援 10.11、10.11.1 和 10.12 版。

藍/綠部署支援的區域和 Aurora 資料庫引擎

藍/綠部署會在個別已同步的預備環境中複製生產資料庫環境。透過使用 Amazon RDS 藍/綠部署，您可以在預備環境中對資料庫進行變更，而不會影響生產環境。例如，您可以升級主要或次要資料庫引擎版本、變更資料庫參數，或在預備環境中進行結構描述變更。備妥後，您可以將預備環境提升為新的生產資料庫環境。如需詳細資訊，請參閱使用 Amazon Aurora 藍/綠部署進行資料庫更新。

透過 Aurora MySQL 進行藍/綠部署

藍/綠部署功能適用於所有版本的 Aurora MySQL AWS 區域，包括設定為 Aurora 全域資料庫的 Aurora MySQL 叢集。

透過 Aurora PostgreSQL 進行藍/綠部署

下列區域和引擎版本適用於使用 Aurora PostgreSQL 的藍/綠部署，包括設定為 Aurora Global Database 的 Aurora PostgreSQL 叢集。

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
全部 AWS 區域	17.4 版及更新版本	16.1 版及更新版本	15.4 版及更新版本	14.9 版及更新版本	13.12 版及更新版本	12.16 版及更新版本	11.21 版及更新版本

叢集儲存體組態的支援區域和 Aurora 資料庫引擎

Amazon Aurora 具有兩個資料庫叢集儲存組態，即 Aurora I/O-Optimized 和 Aurora Standard。如需詳細資訊，請參閱Amazon Aurora 資料庫叢集的儲存組態。

Aurora I/O-Optimized

Aurora I/O-Optimized 適用於下列 AWS 區域 Amazon Aurora 版本的所有 中：

- Aurora MySQL 3.03.1 版及更新版本
- Aurora PostgreSQL 17.4 版及更新版本、16.1 版及更新版本、15.2 版及更新版本、14.7 版及更新版本，和 13.10 版及更新版本

Aurora Standard

Aurora Standard 適用於所有 AWS 區域 Aurora MySQL 和 Aurora PostgreSQL 版本的 。

資料庫活動串流的支援區域和 Aurora 資料庫引擎

透過使用 Aurora 中的資料庫活動串流，您可以監控 Aurora 資料庫中的稽核活動並設定警示。如需詳細資訊，請參閱使用資料庫活動串流來監控 Amazon Aurora。

下列功能不支援資料庫活動串流：

- Aurora Serverless v1
- Aurora Serverless v2
- Babelfish for Aurora PostgreSQL

主題

- 搭配 Amazon Aurora 的資料庫活動串流
- 搭配 Amazon PostgreSQL 的資料庫活動串流

搭配 Amazon Aurora 的資料庫活動串流

下列區域和引擎版本可供搭配 Aurora MySQL 的資料庫活動串流使用。

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
美國東部 (維吉尼亞北部)	所有可用版本	Aurora 2.11 版和更高版本
美國東部 (俄亥俄)	所有可用版本	Aurora 2.11 版和更高版本
美國西部 (加利佛尼亞北部)	所有可用版本	Aurora 2.11 版和更高版本

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
美國西部 (奧勒岡)	所有可用版本	Aurora 2.11 版和更高版本
非洲 (開普敦)	所有可用版本	Aurora 2.11 版和更高版本
亞太地區 (香港)	所有可用版本	Aurora 2.11 版和更高版本
亞太地區 (海德拉巴)	所有可用版本	Aurora 2.11 版和更高版本
亞太地區 (雅加達)	所有可用版本	Aurora 2.11 版和更高版本
亞太地區 (馬來西亞)	不適用	不適用
亞太地區 (墨爾本)	不適用	不適用
亞太地區 (孟買)	所有可用版本	Aurora 2.11 版和更高版本
亞太區域 (紐西蘭)	不適用	不適用
亞太地區 (大阪)	所有可用版本	Aurora 2.11 版和更高版本
亞太地區 (首爾)	所有可用版本	Aurora 2.11 版和更高版本
亞太地區 (新加坡)	所有可用版本	Aurora 2.11 版和更高版本
亞太地區 (悉尼)	所有可用版本	Aurora 2.11 版和更高版本
亞太區域 (台北)	不適用	不適用
亞太區域 (泰國)	不適用	不適用
亞太地區 (東京)	所有可用版本	Aurora 2.11 版和更高版本
加拿大 (中部)	所有可用版本	Aurora 2.11 版和更高版本
加拿大西部 (卡加利)	不適用	不適用
中國 (北京)	不適用	不適用
中國 (寧夏)	不適用	不適用

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
歐洲 (法蘭克福)	所有可用版本	Aurora 2.11 版和更高版本
歐洲 (愛爾蘭)	所有可用版本	Aurora 2.11 版和更高版本
歐洲 (倫敦)	所有可用版本	Aurora 2.11 版和更高版本
歐洲 (米蘭)	所有可用版本	Aurora 2.11 版和更高版本
Europe (Paris)	所有可用版本	Aurora 2.11 版和更高版本
歐洲 (西班牙)	所有可用版本	Aurora 2.11 版和更高版本
歐洲 (斯德哥爾摩)	所有可用版本	Aurora 2.11 版和更高版本
歐洲 (蘇黎世)	不適用	不適用
以色列 (特拉維夫)	不適用	不適用
Middle East (Bahrain)	所有可用版本	Aurora 2.11 版和更高版本
墨西哥 (中部)	不適用	不適用
中東 (阿拉伯聯合大公國)	所有可用版本	Aurora 2.11 版和更高版本
南美洲 (聖保羅)	所有可用版本	Aurora 2.11 版和更高版本

搭配 Amazon PostgreSQL 的資料庫活動串流

下列區域和引擎版本可供搭配 Aurora PostgreSQL 的資料庫活動串流使用。

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
美國東部 (維吉尼亞北部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
美國東部 (俄亥俄)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
美國西部 (加利佛尼亞北部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
美國西部 (奧勒岡)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
非洲 (開普敦)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
亞太地區 (香港)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
亞太地區 (海德拉巴)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
亞太地區 (雅加達)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亞太地區 (馬來西亞)	不適用	不適用	不適用	不適用	不適用	不適用	
亞太地區 (墨爾本)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
亞太地區 (孟買)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
亞太區域 (紐西蘭)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
亞太地區 (大阪)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
亞太地區 (首爾)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
亞太地區 (新加坡)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
亞太地區 (悉尼)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亞太區域 (台北)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
亞太區域 (泰國)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
亞太地區 (東京)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
加拿大 (中部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
加拿大西部 (卡加利)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
中國 (北京)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
中國 (寧夏)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
歐洲 (法蘭克福)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
歐洲 (愛爾蘭)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
歐洲 (倫敦)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
歐洲 (米蘭)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
Europe (Paris)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
歐洲 (西班牙)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
歐洲 (斯德哥爾摩)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
歐洲 (蘇黎世)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
以色列 (特拉維夫)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
墨西哥 (中部)	不適用	不適用	不適用	不適用	不適用	不適用	不適用

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Middle East (Bahrain)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
中東 (阿拉伯聯合大公國)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本
南美洲 (聖保羅)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	所有可用版本	所有可用版本	所有可用版本	11.9 版和 11.13 版以及更新版本

將叢集資料匯出至 Amazon S3 的支援區域和 Aurora 資料庫引擎

您可以將 Aurora 資料庫叢集資料匯出至 Amazon S3 儲存貯體。匯出資料後，您可以直接透過 Amazon Athena 或 Amazon Redshift Spectrum 等工具分析匯出後的資料。如需詳細資訊，請參閱將資料庫叢集資料匯出至 Amazon S3。

將叢集資料匯出至 S3 適用於下列 AWS 區域：

- 美國東部 (維吉尼亞北部)
- 美國東部 (俄亥俄)
- 美國西部 (加利佛尼亞北部)
- 美國西部 (奧勒岡)
- 亞太區域 (香港)
- 亞太區域 (海德拉巴)
- 亞太地區 (雅加達)
- 亞太地區 (墨爾本)
- 亞太地區 (孟買)

- 亞太區域 (紐西蘭)
- 亞太地區 (大阪)
- 亞太區域 (首爾)
- 亞太區域 (新加坡)
- 亞太地區 (雪梨)
- 亞太區域 (東京)
- 加拿大 (中部)
- 加拿大西部 (卡加利)
- 中國 (寧夏)
- 歐洲 (法蘭克福)
- 歐洲 (愛爾蘭)
- 歐洲 (倫敦)
- Europe (Paris)
- 歐洲 (西班牙)
- 歐洲 (斯德哥爾摩)
- 歐洲 (蘇黎世)
- 以色列 (特拉維夫)
- 中東 (阿拉伯聯合大公國)
- 南美洲 (聖保羅)

主題

- 使用 Aurora MySQL 將叢集資料匯出至 S3
- 使用 Aurora PostgreSQL 將叢集資料匯出至 S3

使用 Aurora MySQL 將叢集資料匯出至 S3

所有目前可用的 Aurora MySQL 引擎版本支援將資料庫叢集資料匯出至 Amazon S3。如需版本的詳細資訊，請參閱 Aurora MySQL 版本資訊。

使用 Aurora PostgreSQL 將叢集資料匯出至 S3

所有目前可用的 Aurora PostgreSQL 引擎版本支援將資料庫叢集資料匯出至 Amazon S3。如需版本的更多資訊，請參閱 Aurora PostgreSQL 版本備註。

將快照資料匯出至 Amazon S3 的支援區域和 Aurora 資料庫引擎

您可將 Aurora 資料庫叢集快照資料匯出至 Amazon S3 儲存貯體。您可以匯出手動快照和自動化系統快照。匯出資料後，您可以直接透過 Amazon Athena 或 Amazon Redshift Spectrum 等工具分析匯出後的資料。如需詳細資訊，請參閱將資料庫叢集快照資料匯出至 Amazon S3。

所有 都可以將快照匯出至 S3，但下列項目 AWS 區域 除外：

- 亞太地區 (馬來西亞)
- 亞太區域 (紐西蘭)
- 亞太區域 (台北)
- 亞太區域 (泰國)
- 墨西哥 (中部)

主題

- 使用 Aurora MySQL 將快照資料匯出至 S3
- 使用 Aurora PostgreSQL 將快照資料匯出至 S3

使用 Aurora MySQL 將快照資料匯出至 S3

所有目前可用的 Aurora MySQL 引擎版本支援將資料庫叢集快照資料匯出至 Amazon S3。如需版本的詳細資訊，請參閱 Aurora MySQL 版本資訊。

使用 Aurora PostgreSQL 將快照資料匯出至 S3

所有目前可用的 Aurora PostgreSQL 引擎版本支援將資料庫叢集快照資料匯出至 Amazon S3。如需版本的更多資訊，請參閱 Aurora PostgreSQL 版本備註。

Aurora 全球資料庫的支援區域和資料庫引擎

Aurora 全域資料庫是跨越多個 的單一資料庫 AWS 區域，可從任何區域中斷啟用低延遲全域讀取和災難復原。它為您的部署提供內建容錯能力，因為資料庫執行個體不依賴單一 AWS 區域，而是依賴多個區域和不同的可用區域。如需詳細資訊，請參閱使用 Amazon Aurora 全球資料庫。

主題

- 使用 Aurora MySQL 的 Aurora 全域資料庫

- 使用 Aurora PostgreSQL 的 Aurora 全域資料庫

使用 Aurora MySQL 的 Aurora 全域資料庫

下列區域和引擎版本可供搭配 Aurora MySQL 的 Aurora 全域資料庫使用。

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
美國東部 (維吉尼亞北部)	3.01.0 版及更新版本	2.07.0 版及更新版本
美國東部 (俄亥俄)	3.01.0 版及更新版本	2.07.0 版及更新版本
美國西部 (加利佛尼亞北部)	3.01.0 版及更新版本	2.07.0 版及更新版本
美國西部 (奧勒岡)	3.01.0 版及更新版本	2.07.0 版及更新版本
非洲 (開普敦)	3.01.0 版及更新版本	2.07.1 版及更新版本
亞太地區 (香港)	3.01.0 版及更新版本	2.07.1 版及更新版本
亞太地區 (海德拉巴)	3.02.0 版及更新版本	2.11.2 版及更新版本
亞太地區 (雅加達)	3.01.0 版及更新版本	2.07.6 版及更新版本
亞太地區 (馬來西亞)	3.04.0 版及更新版本	2.07.6 版及更新版本
亞太地區 (墨爾本)	3.03.0 版及更新版本	2.07.6 版及更新版本
亞太地區 (孟買)	3.01.0 版及更新版本	2.07.0 版及更新版本
亞太區域 (紐西蘭)	3.01.0 版及更新版本	2.07.0 版及更新版本
亞太地區 (大阪)	3.01.0 版及更新版本	2.07.3 版及更新版本
亞太地區 (首爾)	3.01.0 版及更新版本	2.07.0 版及更新版本
亞太地區 (新加坡)	3.01.0 版及更新版本	2.07.0 版及更新版本
亞太地區 (悉尼)	3.01.0 版及更新版本	2.07.0 版及更新版本
亞太區域 (台北)	3.01.0 版及更新版本	2.07.0 版及更新版本

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
亞太區域 (泰國)	3.01.0 版及更新版本	2.07.0 版及更新版本
亞太地區 (東京)	3.01.0 版及更新版本	2.07.0 版及更新版本
加拿大 (中部)	3.01.0 版及更新版本	2.07.0 版及更新版本
加拿大西部 (卡加利)	3.01.0 版及更新版本	2.07.0 版及更新版本
中國 (北京)	3.01.0 版及更新版本	2.07.2 版及更新版本
中國 (寧夏)	3.01.0 版及更新版本	2.07.2 版及更新版本
歐洲 (法蘭克福)	3.01.0 版及更新版本	2.07.0 版及更新版本
歐洲 (愛爾蘭)	3.01.0 版及更新版本	2.07.0 版及更新版本
歐洲 (倫敦)	3.01.0 版及更新版本	2.07.0 版及更新版本
歐洲 (米蘭)	3.01.0 版及更新版本	2.07.1 版及更新版本
Europe (Paris)	3.01.0 版及更新版本	2.07.0 版及更新版本
歐洲 (西班牙)	3.02.0 版及更新版本	2.07.6 版及更新版本
歐洲 (斯德哥爾摩)	3.01.0 版及更新版本	2.07.0 版及更新版本
歐洲 (蘇黎世)	3.02.0 版及更新版本	2.07.6 版及更新版本
以色列 (特拉維夫)	3.02.0 版及更新版本	2.07.6 版及更新版本
墨西哥 (中部)	3.02.0 版及更新版本	2.07.6 版及更新版本
Middle East (Bahrain)	3.01.0 版及更新版本	2.07.1 版及更新版本
中東 (阿拉伯聯合大公國)	3.02.0 版及更新版本	2.07.6 版及更新版本
南美洲 (聖保羅)	3.01.0 版及更新版本	2.07.1 版及更新版本
AWS GovCloud （美國東部）	3.01.0 版及更新版本	2.07.0 版及更新版本

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
AWS GovCloud （美國西部）	3.01.0 版及更新版本	2.07.0 版及更新版本

使用 Aurora PostgreSQL 的 Aurora 全域資料庫

下列區域和引擎版本可供搭配 Aurora PostgreSQL 的 Aurora 全域資料庫使用。

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
美國東部 (維吉尼亞北部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
美國東部 (俄亥俄)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
美國西部 (加利佛尼亞北部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
美國西部 (奧勒岡)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
非洲 (開普敦)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亞太地區 (香港)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (海德拉巴)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (雅加達)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (馬來西亞)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (墨爾本)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (孟買)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太區域 (紐西蘭)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亞太地區 (大阪)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (首爾)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (新加坡)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (悉尼)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (悉尼)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太區域 (台北)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太區域 (泰國)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亞太地區 (東京)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
加拿大 (中部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
加拿大西部 (卡加利)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
中國 (北京)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
中國 (寧夏)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
歐洲 (法蘭克福)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
歐洲 (愛爾蘭)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
歐洲 (倫敦)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
歐洲 (米蘭)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
Europe (Paris)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
歐洲 (西班牙)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
歐洲 (斯德哥爾摩)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
歐洲 (蘇黎世)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
以色列 (特拉維夫)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
墨西哥 (中部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
Middle East (Bahrain)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
中東 (阿拉伯聯合大公國)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
南美洲 (聖保羅)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
AWS GovCloud （美國東部）	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
AWS GovCloud （美國西部）	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本

IAM 資料庫身分驗證的支援區域和 Aurora 資料庫引擎

透過 Aurora 中的 IAM 資料庫身分驗證，您可以使用 AWS Identity and Access Management (IAM) 資料庫身分驗證對資料庫叢集進行身分驗證。透過此身分驗證方法，您連線至資料庫叢集時不需要使用密碼。而是改用身分驗證字符。如需詳細資訊，請參閱 IAM 資料庫身分驗證。

主題

- 使用 Aurora MySQL 進行 IAM 資料庫身分驗證
- 使用 Aurora PostgreSQL 進行 IAM 資料庫身分驗證

使用 Aurora MySQL 進行 IAM 資料庫身分驗證

若為下列版本，所有區域都可以使用 Aurora MySQL 進行 IAM 資料庫身分驗證：

- Aurora MySQL 3 – 所有可用版本
- Aurora MySQL 2 – 所有可用版本

使用 Aurora PostgreSQL 進行 IAM 資料庫身分驗證

若為下列引擎版本，所有區域都可以使用 Aurora PostgreSQL 進行 IAM 資料庫身分驗證：

- Aurora PostgreSQL 17 - 所有可用版本
- Aurora PostgreSQL 16 - 所有可用版本
- Aurora PostgreSQL 15 - 所有可用版本
- Aurora PostgreSQL 14 - 所有可用版本
- Aurora PostgreSQL 13 - 所有可用版本
- Aurora PostgreSQL 12 - 所有可用版本
- Aurora PostgreSQL 11 - 所有可用版本

Kerberos 身分驗證的支援區域和 Aurora 資料庫引擎

透過搭配 Aurora 使用 Kerberos 身分驗證，您可以支援 Kerberos 和 Microsoft Active Directory 之資料庫使用者的外部身分驗證。使用 Kerberos 和 Active Directory，提供了資料庫使用者的單一登入和集中式身分驗證優點。Kerberos 和 Active Directory 可與 AWS Directory Service for Microsoft Active Directory的功能搭配使用 Directory Service。如需詳細資訊，請參閱Kerberos 身分驗證。

主題

- 使用 Aurora MySQL 進行 Kerberos 身分驗證
- 使用 Aurora PostgreSQL 進行 Kerberos 身分驗證
- 使用 Aurora PostgreSQL 的 Active Directory(AD) 安全群組

使用 Aurora MySQL 進行 Kerberos 身分驗證

下列區域和引擎版本可與 Aurora MySQL 搭配使用進行 Kerberos 身分驗證。

區域	Aurora MySQL 第 3 版
美國東部 (維吉尼亞北部)	3.03.0 版及更新版本
美國東部 (俄亥俄)	3.03.0 版及更新版本
美國西部 (加利佛尼亞北部)	3.03.0 版及更新版本
美國西部 (奧勒岡)	3.03.0 版及更新版本
非洲 (開普敦)	不適用
亞太地區 (香港)	不適用
亞太地區 (雅加達)	不適用
亞太地區 (馬來西亞)	不適用
亞太地區 (墨爾本)	不適用
亞太地區 (孟買)	3.03.0 版及更新版本
亞太區域 (紐西蘭)	不適用
亞太地區 (大阪)	不適用
亞太地區 (首爾)	3.03.0 版及更新版本
亞太地區 (新加坡)	3.03.0 版及更新版本
亞太地區 (悉尼)	3.03.0 版及更新版本

區域	Aurora MySQL 第 3 版
亞太區域 (台北)	不適用
亞太區域 (泰國)	不適用
亞太地區 (東京)	3.03.0 版及更新版本
加拿大 (中部)	3.03.0 版及更新版本
加拿大西部 (卡加利)	不適用
中國 (北京)	3.03.0 版及更新版本
中國 (寧夏)	3.03.0 版及更新版本
歐洲 (法蘭克福)	3.03.0 版及更新版本
歐洲 (愛爾蘭)	3.03.0 版及更新版本
歐洲 (倫敦)	3.03.0 版及更新版本
歐洲 (米蘭)	不適用
Europe (Paris)	3.03.0 版及更新版本
歐洲 (西班牙)	不適用
歐洲 (斯德哥爾摩)	3.03.0 版及更新版本
歐洲 (蘇黎世)	不適用
以色列 (特拉維夫)	不適用
墨西哥 (中部)	不適用
Middle East (Bahrain)	不適用
中東 (阿拉伯聯合大公國)	不適用
南美洲 (聖保羅)	3.03.0 版及更新版本

區域	Aurora MySQL 第 3 版
AWS GovCloud（美國東部）	3.03.0 版及更新版本
AWS GovCloud（美國西部）	3.03.0 版及更新版本

使用 Aurora PostgreSQL 進行 Kerberos 身分驗證

下列區域和引擎版本可與 Aurora PostgreSQL 搭配使用進行 Kerberos 身分驗證。

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
美國東部 (維吉尼亞北部)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
美國東部 (俄亥俄)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
美國西部 (加利佛尼亞北部)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
美國西部 (奧勒岡)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
非洲 (開普敦)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (香港)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (海德拉巴)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亞太地區 (雅加達)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (馬來西亞)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
亞太地區 (墨爾本)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (孟買)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太區域 (紐西蘭)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
亞太地區 (大阪)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
亞太地區 (首爾)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (新加坡)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (悉尼)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太區域 (台北)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
亞太區域 (泰國)	不適用	不適用	不適用	不適用	不適用	不適用	不適用

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亞太地區 (東京)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
加拿大 (中部)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
加拿大西部 (卡加利)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
中國 (北京)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
中國 (寧夏)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (法蘭克福)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (愛爾蘭)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (倫敦)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (米蘭)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
Europe (Paris)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (西班牙)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
歐洲 (斯德哥爾摩)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (蘇黎世)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
以色列 (特拉維夫)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
墨西哥 (中部)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
Middle East (Bahrain)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
中東 (阿拉伯聯合大公國)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
南美洲 (聖保羅)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
AWS GovCloud （美國東部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
AWS GovCloud （美國西部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

使用 Aurora PostgreSQL 的 Active Directory(AD) 安全群組

下列區域和引擎版本可供搭配 Aurora PostgreSQL 的 ActiveDirectory 使用。

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
美國東部 (維吉尼亞北部)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
美國東部 (俄亥俄)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
美國西部 (加利佛尼亞北部)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
美國西部 (奧勒岡)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
非洲 (開普敦)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (香港)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (海德拉巴)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (雅加達)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (馬來西亞)	不適用	不適用	不適用	不適用	不適用	不適用	不適用

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亞太地區 (墨爾本)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (孟買)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太區域 (紐西蘭)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
亞太地區 (大阪)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
亞太地區 (首爾)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (新加坡)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (悉尼)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太區域 (台北)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
亞太區域 (泰國)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
亞太地區 (東京)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
加拿大 (中部)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
加拿大西部 (卡加利)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
中國 (北京)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
中國 (寧夏)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (法蘭克福)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (愛爾蘭)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (倫敦)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (米蘭)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
Europe (Paris)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (西班牙)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (斯德哥爾摩)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (蘇黎世)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
以色列 (特拉維夫)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
墨西哥 (中部)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
Middle East (Bahrain)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
中東 (阿拉伯聯合大公國)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
南美洲 (聖保羅)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
AWS GovCloud （美國東部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
AWS GovCloud （美國西部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

Aurora 機器學習的支援區域和資料庫引擎

透過使用 Amazon Aurora Machine Learning，您可以根據您的需求，將 Aurora 資料庫叢集與下列其中一項 AWS 機器學習服務整合。其各自支援特定的機器學習使用案例。

Amazon Bedrock 是一項全受管服務，可透過 API 提供來自 AI 公司的頂尖基礎模型，以及提供開發人員工具，以協助建置和擴展生成式 AI 應用程式。

Amazon Comprehend 是一種自然語言處理 (NLP) 服務，用來從文件擷取洞見。透過搭配 Amazon Comprehend 使用 Aurora Machine Learning，您可以判斷資料庫表格中文字的觀點。

SageMaker AI 是全功能的機器學習服務。資料科學家會使用 Amazon SageMaker AI，針對為各種推論任務 (例如詐騙偵測) 建置、訓練和測試機器學習模型。透過搭配 SageMaker AI 使用 Aurora 機器學習，資料庫開發人員可以在 SQL 程式碼中調用 SageMaker AI 功能。

並非所有 都 AWS 區域 支援所有機器學習服務。只有某些 AWS 區域 支援 Aurora 機器學習，因此可讓您從 Aurora 資料庫叢集存取這些服務。Aurora Machine Learning 的整合程序也因資料庫引擎而異。如需詳細資訊，請參閱使用 Amazon Aurora 機器學習。

主題

- 將機器學習與 Aurora MySQL 搭配使用
- 將機器學習與 Aurora PostgreSQL 搭配使用

將機器學習與 Aurora MySQL 搭配使用

僅在 Aurora MySQL 3.06 版及更新版本支援 Amazon Bedrock。如需 Amazon Bedrock 區域可用性的相關資訊，請參閱《Amazon Bedrock 使用者指南》中的依 AWS 區域的模型支援。

表格中 AWS 區域 列出的 Aurora MySQL 支援 Aurora Machine Learning 搭配 Amazon Comprehend 和 Amazon SageMaker AI。除了提供您的 Aurora MySQL 版本之外， 還 AWS 區域 必須支援您要使用的服務。如需 Amazon SageMaker AI 可用 AWS 區域 位置的清單，請參閱《》中的 Amazon SageMaker AI 端點和配額Amazon Web Services 一般參考。如需 Amazon Comprehend 可用 AWS 區域 位置的清單，請參閱《》中的 Amazon Comprehend 端點和配額Amazon Web Services 一般參考。

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
美國東部 (維吉尼亞北部)	3.01.0 版及更新版本	2.07 版及更新版本
美國東部 (俄亥俄)	3.01.0 版及更新版本	2.07 版及更新版本
美國西部 (加利佛尼亞北部)	3.01.0 版及更新版本	2.07 版及更新版本
美國西部 (奧勒岡)	3.01.0 版及更新版本	2.07 版及更新版本

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
非洲 (開普敦)	不適用	不適用
亞太地區 (香港)	3.01.0 版及更新版本	2.07 版及更新版本
亞太地區 (海德拉巴)	3.01.0 版及更新版本	2.07 版及更新版本
亞太地區 (雅加達)	3.01.0 版及更新版本	2.07 版及更新版本
亞太地區 (馬來西亞)	3.04.0 版及更新版本	不適用
亞太地區 (墨爾本)	3.01.0 版及更新版本	2.07 版及更新版本
亞太地區 (孟買)	3.01.0 版及更新版本	2.07 版及更新版本
亞太區域 (紐西蘭)	不適用	不適用
亞太地區 (大阪)	3.01.0 版及更新版本	2.07.3 版及更新版本
亞太地區 (首爾)	3.01.0 版及更新版本	2.07 版及更新版本
亞太地區 (新加坡)	3.01.0 版及更新版本	2.07 版及更新版本
亞太地區 (悉尼)	3.01.0 版及更新版本	2.07 版及更新版本
亞太區域 (台北)	不適用	不適用
亞太區域 (泰國)	不適用	不適用
亞太地區 (東京)	3.01.0 版及更新版本	2.07 版及更新版本
加拿大 (中部)	3.01.0 版及更新版本	2.07 版及更新版本
加拿大西部 (卡加利)	3.01.0 版及更新版本	2.07 版及更新版本
中國 (北京)	3.01.0 版及更新版本	2.07 版及更新版本
中國 (寧夏)	3.01.0 版及更新版本	2.07 版及更新版本
歐洲 (法蘭克福)	3.01.0 版及更新版本	2.07 版及更新版本

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
歐洲 (愛爾蘭)	3.01.0 版及更新版本	2.07 版及更新版本
歐洲 (倫敦)	3.01.0 版及更新版本	2.07 版及更新版本
歐洲 (米蘭)	不適用	不適用
Europe (Paris)	3.01.0 版及更新版本	2.07 版及更新版本
歐洲 (西班牙)	3.01.0 版及更新版本	2.07 版及更新版本
歐洲 (斯德哥爾摩)	3.01.0 版及更新版本	2.07 版及更新版本
歐洲 (蘇黎世)	3.01.0 版及更新版本	2.07 版及更新版本
以色列 (特拉維夫)	3.01.0 版及更新版本	2.07 版及更新版本
墨西哥 (中部)	不適用	不適用
Middle East (Bahrain)	3.01.0 版及更新版本	2.07 版及更新版本
中東 (阿拉伯聯合大公國)	3.01.0 版及更新版本	2.07 版及更新版本
南美洲 (聖保羅)	3.01.0 版及更新版本	2.07 版及更新版本
AWS GovCloud （美國東部）	3.01.0 版及更新版本	2.07 版及更新版本
AWS GovCloud （美國西部）	3.01.0 版及更新版本	2.07 版及更新版本

將機器學習與 Aurora PostgreSQL 搭配使用

如需 Aurora PostgreSQL 上 Amazon Bedrock 版本支援的相關資訊，請參閱使用 Aurora PostgreSQL 作為 Amazon Bedrock 的知識庫。

表格中 AWS 區域 列出的 Aurora PostgreSQL 支援 Aurora Machine Learning 搭配 Amazon Comprehend 和 Amazon SageMaker AI。除了提供您的 Aurora PostgreSQL 版本之外， 還 AWS 區域 必須支援您要使用的服務。如需 Amazon SageMaker AI 可用 AWS 區域 位置的清單，請參閱《》中的 Amazon SageMaker AI 端點和配額Amazon Web Services 一般參考。如需 Amazon Comprehend 可

用 AWS 區域 位置的清單，請參閱《》中的 Amazon Comprehend 端點和配額Amazon Web Services 一般參考。

下列區域和引擎版本可供搭配 Aurora PostgreSQL 的 Aurora machine learning 使用。

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
美國東部 (維吉尼亞北部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
美國東部 (俄亥俄)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
美國西部 (加利佛尼亞北部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
美國西部 (奧勒岡)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
非洲 (開普敦)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
亞太地區 (香港)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
亞太地區 (海德拉巴)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
亞太地區 (雅加達)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亞太地區 (馬來西亞)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
亞太地區 (墨爾本)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
亞太地區 (孟買)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
亞太區域 (紐西蘭)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
亞太地區 (大阪)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
亞太地區 (首爾)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
亞太地區 (新加坡)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
亞太地區 (悉尼)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
亞太區域 (台北)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亞太區域 (泰國)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
亞太地區 (東京)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
加拿大 (中部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
加拿大西部 (卡加利)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
中國 (北京)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
中國 (寧夏)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
歐洲 (法蘭克福)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
歐洲 (愛爾蘭)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
歐洲 (倫敦)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
歐洲 (米蘭)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
Europe (Paris)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
歐洲 (西班牙)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
歐洲 (斯德哥爾摩)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
歐洲 (蘇黎世)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
以色列 (特拉維夫)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
墨西哥 (中部)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
Middle East (Bahrain)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
中東 (阿拉伯聯合大公國)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
南美洲 (聖保羅)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
AWS GovCloud（美國東部）	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本
AWS GovCloud（美國西部）	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.3 版及更新版本	12.4 版及更新版本	11.9、11.12 及更高版本

Performance Insights 的支援區域和 Aurora 資料庫引擎

Important

AWS 已宣布績效詳情的end-of-life：2026 年 6 月 30 日。在此日期之後，Amazon RDS 將不再支援 Performance Insights 主控台體驗、彈性保留期間 (1-24 個月) 及其相關定價。Performance Insights API 將繼續存在，不會有任何定價變更。Performance Insights API 的成本會顯示在您的 AWS 帳單中，其中包含 CloudWatch Database Insights 的成本。

我們建議您在 2026 年 6 月 30 日之前，使用已付費的績效詳情方案，將任何資料庫叢集升級至資料庫詳情的進階模式。如需有關升級至 Database Insights 進階模式的資訊，請參閱 開啟適用於 Amazon Aurora 的 Database Insights 的進階模式。

如果您不採取任何動作，則使用 Performance Insights 的資料庫叢集依預設會使用 Database Insights 的標準模式。使用 Database Insights 的標準模式時，您可能會失去超過 7 天的效能資料歷史記錄存取權，而且可能無法在 Amazon RDS 主控台中使用執行計劃和隨需分析功能。2026 年 6 月 30 日之後，只有進階模式的 Database Insights 將支援執行計畫和隨需分析。

透過 CloudWatch Database Insights，您可以監控資料庫機群的資料庫負載，以及大規模分析效能和進行疑難排解。如需 Database Insights 的詳細資訊，請參閱 使用 CloudWatch

Database Insights 監控 Amazon Aurora 資料庫。如需定價資訊，請參閱 Amazon CloudWatch 定價。

Performance Insights 會擴展現有監控功能，以說明並協助您分析資料庫效能叢集效能。利用 Performance Insights 儀表板，您可將 Amazon RDS 資料庫執行個體負載上的資料庫負載視覺化，並依等候、SQL 陳述式、主機或使用者篩選負載。如需詳細資訊，請參閱Amazon Aurora 上的績效詳情概觀。

如需 Performance Insights 功能的區域、資料庫引擎和執行個體類別支援資訊，請參閱 Performance Insights 功能的 Amazon Aurora 資料庫引擎、區域和執行個體類別支援。

主題

- 搭配 Aurora MySQL 的 Performance Insights
- 搭配 Aurora PostgreSQL 的 Performance Insights
- 搭配 Aurora Serverless 的 Performance Insights

搭配 Aurora MySQL 的 Performance Insights

> **Note**
>
> 如果您已開啟平行查詢，則搭配 Aurora MySQL 的 Performance Insights 會有不同的引擎版本支援。如需平行查詢的詳細資訊，請參閱 Amazon Aurora MySQL 的平行查詢。

主題

- 搭配 Aurora MySQL 並已關閉平行查詢的 Performance Insights
- 搭配 Aurora MySQL 並已開啟平行查詢的 Performance Insights

搭配 Aurora MySQL 並已關閉平行查詢的 Performance Insights

下列區域和引擎版本可供搭配 Aurora MySQL 且關閉平行查詢的績效詳情使用。

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
美國東部 (維吉尼亞北部)	所有版本	所有版本

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
美國東部 (俄亥俄)	所有版本	所有版本
美國西部 (加利佛尼亞北部)	所有版本	所有版本
美國西部 (奧勒岡)	所有版本	所有版本
非洲 (開普敦)	所有版本	所有版本
亞太地區 (香港)	所有版本	所有版本
亞太地區 (海德拉巴)	所有版本	所有版本
亞太地區 (雅加達)	所有版本	所有版本
亞太地區 (馬來西亞)	所有版本	所有版本
亞太地區 (墨爾本)	所有版本	所有版本
亞太地區 (孟買)	所有版本	所有版本
亞太區域 (紐西蘭)	所有版本	所有版本
亞太地區 (大阪)	所有版本	所有版本
亞太地區 (首爾)	所有版本	所有版本
亞太地區 (新加坡)	所有版本	所有版本
亞太地區 (悉尼)	所有版本	所有版本
亞太區域 (台北)	所有版本	所有版本
亞太區域 (泰國)	所有版本	所有版本
亞太地區 (東京)	所有版本	所有版本
加拿大 (中部)	所有版本	所有版本
加拿大西部 (卡加利)	所有版本	所有版本

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
中國 (北京)	所有版本	所有版本
中國 (寧夏)	所有版本	所有版本
歐洲 (法蘭克福)	所有版本	所有版本
歐洲 (愛爾蘭)	所有版本	所有版本
歐洲 (倫敦)	所有版本	所有版本
歐洲 (米蘭)	所有版本	所有版本
Europe (Paris)	所有版本	所有版本
歐洲 (西班牙)	所有版本	所有版本
歐洲 (斯德哥爾摩)	所有版本	所有版本
歐洲 (蘇黎世)	所有版本	所有版本
以色列 (特拉維夫)	所有版本	所有版本
墨西哥 (中部)	所有版本	所有版本
Middle East (Bahrain)	所有版本	所有版本
中東 (阿拉伯聯合大公國)	所有版本	所有版本
南美洲 (聖保羅)	所有版本	所有版本
AWS GovCloud （美國東部）	所有版本	所有版本
AWS GovCloud （美國西部）	所有版本	所有版本

搭配 Aurora MySQL 並已開啟平行查詢的 Performance Insights

下列區域和引擎版本可供搭配 Aurora MySQL 且啟動平行查詢的績效詳情使用。

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
美國東部 (維吉尼亞北部)	不適用	2.09.0 版及更高版本
美國東部 (俄亥俄)	不適用	2.09.0 版及更高版本
美國西部 (加利佛尼亞北部)	不適用	2.09.0 版及更高版本
美國西部 (奧勒岡)	不適用	2.09.0 版及更高版本
非洲 (開普敦)	不適用	2.09.0 版及更高版本
亞太地區 (香港)	不適用	2.09.0 版及更高版本
亞太地區 (海德拉巴)	不適用	所有版本
亞太地區 (雅加達)	不適用	2.09.0 版及更高版本
亞太地區 (馬來西亞)	不適用	2.09.0 版及更高版本
亞太地區 (墨爾本)	不適用	2.09.0 版及更高版本
亞太地區 (孟買)	不適用	2.09.0 版及更高版本
亞太區域 (紐西蘭)	不適用	2.09.0 版及更高版本
亞太地區 (大阪)	不適用	2.09.0 版及更高版本
亞太地區 (首爾)	不適用	2.09.0 版及更高版本
亞太地區 (新加坡)	不適用	2.09.0 版及更高版本
亞太地區 (悉尼)	不適用	2.09.0 版及更高版本
亞太區域 (台北)	不適用	2.09.0 版及更高版本
亞太區域 (泰國)	不適用	2.09.0 版及更高版本
亞太地區 (東京)	不適用	2.09.0 版及更高版本
加拿大 (中部)	不適用	2.09.0 版及更高版本

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
加拿大西部 (卡加利)	不適用	2.09.0 版及更高版本
中國 (北京)	不適用	2.09.0 版及更高版本
中國 (寧夏)	不適用	2.09.0 版及更高版本
歐洲 (法蘭克福)	不適用	2.09.0 版及更高版本
歐洲 (愛爾蘭)	不適用	2.09.0 版及更高版本
歐洲 (倫敦)	不適用	2.09.0 版及更高版本
歐洲 (米蘭)	不適用	2.09.0 版及更高版本
Europe (Paris)	不適用	2.09.0 版及更高版本
歐洲 (西班牙)	不適用	2.09.0 版及更高版本
歐洲 (斯德哥爾摩)	不適用	2.09.0 版及更高版本
歐洲 (蘇黎世)	不適用	2.09.0 版及更高版本
以色列 (特拉維夫)	不適用	2.09.0 版及更高版本
墨西哥 (中部)	不適用	2.09.0 版及更高版本
Middle East (Bahrain)	不適用	2.09.0 版及更高版本
中東 (阿拉伯聯合大公國)	不適用	2.09.0 版及更高版本
南美洲 (聖保羅)	不適用	2.09.0 版及更高版本
AWS GovCloud （美國東部）	不適用	2.09.0 版及更高版本
AWS GovCloud （美國西部）	不適用	2.09.0 版及更高版本

搭配 Aurora PostgreSQL 的 Performance Insights

下列區域和引擎版本可供搭配 Aurora PostgreSQL 的績效詳情使用。

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
美國東部 (維吉尼亞北部)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
美國東部 (俄亥俄)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
美國西部 (加利佛尼亞北部)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
美國西部 (奧勒岡)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
非洲 (開普敦)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (香港)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (海德拉巴)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (雅加達)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
亞太地區 (馬來西亞)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (墨爾本)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (孟買)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太區域 (紐西蘭)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (大阪)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (首爾)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (新加坡)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (悉尼)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太區域 (台北)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
亞太區域 (泰國)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亞太地區 (東京)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
加拿大 (中部)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
加拿大西部 (卡加利)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
中國 (北京)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
中國 (寧夏)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (法蘭克福)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (愛爾蘭)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (倫敦)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (米蘭)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
Europe (Paris)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
歐洲 (西班牙)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (斯德哥爾摩)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
歐洲 (蘇黎世)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
以色列 (特拉維夫)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
墨西哥 (中部)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
Middle East (Bahrain)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
中東 (阿拉伯聯合大公國)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
南美洲 (聖保羅)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
AWS GovCloud（美國東部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
AWS GovCloud（美國西部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

搭配 Aurora Serverless 的 Performance Insights

Aurora Serverless v2支援所有與 MySQL 相容和 PostgreSQL 相容版本的 Performance Insights。我們建議您將最小容量設定為至少 2 個 Aurora 容量單位 (ACU)。

Aurora Serverless v1 不支援績效詳情。

零 ETL 整合的支援區域和 Aurora 資料庫引擎

Amazon Aurora 零 ETL 整合是一種全受管解決方案，可讓交易資料在寫入 Aurora 叢集之後在 Amazon Redshift 或 Amazon SageMaker 中使用。如需詳細資訊，請參閱零 ETL 整合。

主題

- Aurora MySQL 零 ETL 整合
- Aurora PostgreSQL 零 ETL 整合

Aurora MySQL 零 ETL 整合

下列區域和引擎版本適用於與 Amazon Redshift 和 Amazon SageMaker 的 Aurora MySQL 零 ETL 整合。

區域	與 Amazon Redshift for Aurora MySQL 第 3 版的零 ETL 整合	與 Amazon SageMaker for Aurora MySQL 第 3 版的零 ETL 整合
美國東部 (維吉尼亞北部)	3.05.2 版及更新版本	3.05.2 版及更新版本
美國東部 (俄亥俄)	3.05.2 版及更新版本	3.05.2 版及更新版本

區域	與 Amazon Redshift for Aurora MySQL 第 3 版的零 ETL 整合	與 Amazon SageMaker for Aurora MySQL 第 3 版的零 ETL 整合
美國西部 (加利佛尼亞北部)	3.05.2 版及更新版本	不適用
美國西部 (奧勒岡)	3.05.2 版及更新版本	3.05.2 版及更新版本
非洲 (開普敦)	3.05.2 版及更新版本	不適用
亞太地區 (香港)	3.05.2 版及更新版本	3.05.2 版及更新版本
亞太地區 (海德拉巴)	3.05.2 版及更新版本	不適用
亞太地區 (雅加達)	3.05.2 版及更新版本	不適用
亞太地區 (馬來西亞)	不適用	不適用
亞太地區 (墨爾本)	3.05.2 版及更新版本	不適用
亞太地區 (孟買)	3.05.2 版及更新版本	不適用
亞太區域 (紐西蘭)	不適用	不適用
亞太地區 (大阪)	3.05.2 版及更新版本	不適用
亞太地區 (首爾)	3.05.2 版及更新版本	3.05.2 版及更新版本
亞太地區 (新加坡)	3.05.2 版及更新版本	3.05.2 版及更新版本
亞太地區 (悉尼)	3.05.2 版及更新版本	3.05.2 版及更新版本
亞太區域 (台北)	不適用	不適用
亞太區域 (泰國)	不適用	不適用
亞太地區 (東京)	3.05.2 版及更新版本	3.05.2 版及更新版本
加拿大 (中部)	3.05.2 版及更新版本	3.05.2 版及更新版本
加拿大西部 (卡加利)	3.05.2 版及更新版本	不適用

區域	與 Amazon Redshift for Aurora MySQL 第 3 版的零 ETL 整合	與 Amazon SageMaker for Aurora MySQL 第 3 版的零 ETL 整合
中國 (北京)	3.05.2 版及更新版本	不適用
中國 (寧夏)	3.05.2 版及更新版本	不適用
歐洲 (法蘭克福)	3.05.2 版及更新版本	3.05.2 版及更新版本
歐洲 (愛爾蘭)	3.05.2 版及更新版本	3.05.2 版及更新版本
歐洲 (倫敦)	3.05.2 版及更新版本	3.05.2 版及更新版本
歐洲 (米蘭)	3.05.2 版及更新版本	不適用
Europe (Paris)	3.05.2 版及更新版本	不適用
歐洲 (西班牙)	3.05.2 版及更新版本	不適用
歐洲 (斯德哥爾摩)	3.05.2 版及更新版本	3.05.2 版及更新版本
歐洲 (蘇黎世)	3.05.2 版及更新版本	不適用
以色列 (特拉維夫)	3.05.2 版及更新版本	不適用
墨西哥 (中部)	不適用	不適用
Middle East (Bahrain)	3.05.2 版及更新版本	不適用
中東 (阿拉伯聯合大公國)	3.05.2 版及更新版本	不適用
南美洲 (聖保羅)	3.05.2 版及更新版本	3.05.2 版及更新版本
AWS GovCloud （美國東部）	不適用	不適用
AWS GovCloud （美國西部）	不適用	不適用

Aurora PostgreSQL 零 ETL 整合

下列區域和引擎版本適用於與 Amazon Redshift 和 的 Aurora PostgreSQL 零 ETL 整合Amazon Sagemaker。

區域	與 Amazon Redshift for Aurora PostgreSQL 第 17 版的零 ETL 整合	與 Amazon SageMaker for Aurora PostgreSQL 第 17 版的 0-ETL 整合	與 Amazon Redshift for Aurora PostgreSQL 第 16 版的零 ETL 整合	與 Amazon SageMaker for Aurora PostgreSQL 第 16 版的 0-ETL 整合
美國東部 (俄亥俄)	17.4 版及更新版本	17.4 版及更新版本	16.4 版及更新版本	16.4 版及更新版本
美國東部 (維吉尼亞北部)	17.4 版及更新版本	17.4 版及更新版本	16.4 版及更新版本	16.4 版及更新版本
美國西部 (加利佛尼亞北部)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用
美國西部 (奧勒岡)	17.4 版及更新版本	17.4 版及更新版本	16.4 版及更新版本	16.4 版及更新版本
非洲 (開普敦)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用
亞太地區 (香港)	17.4 版及更新版本	17.4 版及更新版本	16.4 版及更新版本	16.4 版及更新版本
亞太地區 (海德拉巴)	17.4 版及更新版本	16.4 版及更新版本	不適用	不適用
亞太地區 (雅加達)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用
亞太地區 (墨爾本)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用

區域	與 Amazon Redshift for Aurora PostgreSQL 第 17 版的零 ETL 整合	與 Amazon SageMaker for Aurora PostgreSQL 第 17 版的 0-ETL 整合	與 Amazon Redshift for Aurora PostgreSQL 第 16 版的零 ETL 整合	與 Amazon SageMaker for Aurora PostgreSQL 第 16 版的 0-ETL 整合
亞太地區 (馬來西亞)	不適用	不適用		
亞太地區 (馬來西亞)	不適用	不適用		
亞太地區 (孟買)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用
亞太區域 (紐西蘭)	不適用	不適用		
亞太地區 (大阪)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用
亞太地區 (首爾)	17.4 版及更新版本	17.4 版及更新版本	16.4 版及更新版本	16.4 版及更新版本
亞太地區 (新加坡)	17.4 版及更新版本	17.4 版及更新版本	16.4 版及更新版本	16.4 版及更新版本
亞太地區 (悉尼)	17.4 版及更新版本	17.4 版及更新版本	16.4 版及更新版本	16.4 版及更新版本
亞太區域 (台北)	不適用	不適用		
亞太區域 (泰國)	不適用	不適用		
亞太區域 (台北)	不適用	不適用		
亞太區域 (泰國)	不適用	不適用		

區域	與 Amazon Redshift for Aurora PostgreSQL 第 17 版的零 ETL 整合	與 Amazon SageMaker for Aurora PostgreSQL 第 17 版的 0-ETL 整合	與 Amazon Redshift for Aurora PostgreSQL 第 16 版的零 ETL 整合	與 Amazon SageMaker for Aurora PostgreSQL 第 16 版的 0-ETL 整合
亞太地區 (東京)	17.4 版及更新版本	17.4 版及更新版本	16.4 版及更新版本	16.4 版及更新版本
加拿大 (中部)	17.4 版及更新版本	17.4 版及更新版本	16.4 版及更新版本	16.4 版及更新版本
加拿大西部 (卡加利)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用
中國 (北京)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用
中國 (寧夏)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用
歐洲 (法蘭克福)	17.4 版及更新版本	17.4 版及更新版本	16.4 版及更新版本	16.4 版及更新版本
歐洲 (愛爾蘭)	17.4 版及更新版本	17.4 版及更新版本	16.4 版及更新版本	16.4 版及更新版本
歐洲 (倫敦)	17.4 版及更新版本	17.4 版及更新版本	16.4 版及更新版本	16.4 版及更新版本
歐洲 (米蘭)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用
Europe (Paris)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用
歐洲 (西班牙)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用

區域	與 Amazon Redshift for Aurora PostgreSQL 第 17 版的零 ETL 整合	與 Amazon SageMaker for Aurora PostgreSQL 第 17 版的 0-ETL 整合	與 Amazon Redshift for Aurora PostgreSQL 第 16 版的零 ETL 整合	與 Amazon SageMaker for Aurora PostgreSQL 第 16 版的 0-ETL 整合
歐洲 (斯德哥爾摩)	17.4 版及更新版本	17.4 版及更新版本	16.4 版及更新版本	16.4 版及更新版本
歐洲 (蘇黎世)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用
以色列 (特拉維夫)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用
墨西哥 (中部)	不適用	不適用		
墨西哥 (中部)	不適用	不適用		
Middle East (Bahrain)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用
中東 (阿拉伯聯合大公國)	17.4 版及更新版本	不適用	16.4 版及更新版本	不適用
南美洲 (聖保羅)	17.4 版及更新版本	17.4 版及更新版本	16.4 版及更新版本	16.4 版及更新版本
AWS GovCloud （美國東部）	不適用	不適用	不適用	不適用
AWS GovCloud （美國西部）	不適用	不適用	不適用	不適用

Amazon RDS Proxy 的支援區域和 Aurora 資料庫引擎

Amazon RDS Proxy 是完全受管、高可用性的資料庫 Proxy，可透過共用建立的資料庫連線，讓應用程式更具可擴充性。如需 RDS Proxy 的詳細資訊，請參閱 Amazon RDS Proxy for Aurora。

主題

- 使用 Aurora MySQL 的 Amazon RDS Proxy
- 使用 Aurora PostgreSQL 的 Amazon RDS Proxy

使用 Aurora MySQL 的 Amazon RDS Proxy

下列區域和引擎版本可供搭配 Aurora MySQL 的 RDS Proxy 使用。

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
美國東部 (維吉尼亞北部)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
美國東部 (俄亥俄)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
美國西部 (加利佛尼亞北部)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
美國西部 (奧勒岡)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
非洲 (開普敦)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
亞太地區 (香港)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
亞太地區 (海德拉巴)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
亞太地區 (雅加達)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
亞太地區 (馬來西亞)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
亞太地區 (墨爾本)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
亞太地區 (孟買)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
亞太區域 (紐西蘭)	不適用	不適用
亞太地區 (大阪)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
亞太地區 (首爾)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
亞太地區 (新加坡)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
亞太地區 (悉尼)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
亞太區域 (台北)	不適用	不適用
亞太區域 (泰國)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
亞太地區 (東京)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
加拿大 (中部)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
加拿大西部 (卡加利)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
中國 (北京)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
中國 (寧夏)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
歐洲 (法蘭克福)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
歐洲 (愛爾蘭)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
歐洲 (倫敦)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
歐洲 (米蘭)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
Europe (Paris)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
歐洲 (西班牙)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
歐洲 (斯德哥爾摩)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
歐洲 (蘇黎世)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
以色列 (特拉維夫)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
墨西哥 (中部)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
Middle East (Bahrain)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
中東 (阿拉伯聯合大公國)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
南美洲 (聖保羅)	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
AWS GovCloud（美國東部）	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本
AWS GovCloud（美國西部）	3.01.0 版及更新版本	2.07 版和 2.11 版以及更新版本

使用 Aurora PostgreSQL 的 Amazon RDS Proxy

下列區域和引擎版本可供搭配 Aurora PostgreSQL 的 RDS Proxy 使用。

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
美國東部 (維吉尼亞北部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
美國東部 (俄亥俄)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
美國西部 (加利佛尼亞北部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
美國西部 (奧勒岡)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
非洲 (開普敦)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
							以及更新版本
亞太地區 (香港)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (海德拉巴)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (雅加達)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (馬來西亞)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (墨爾本)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (孟買)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太區域 (紐西蘭)	不適用	不適用	不適用	不適用	不適用	不適用	不適用

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亞太地區 (大阪)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (首爾)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (新加坡)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (悉尼)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太區域 (台北)	不適用	不適用	不適用	不適用	不適用	不適用	不適用
亞太區域 (泰國)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
亞太地區 (東京)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
加拿大 (中部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
加拿大西部 (卡加利)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
中國 (北京)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
中國 (寧夏)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
歐洲 (法蘭克福)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
歐洲 (愛爾蘭)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
歐洲 (倫敦)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
歐洲 (米蘭)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
Europe (Paris)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
歐洲 (西班牙)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
歐洲 (斯德哥爾摩)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
歐洲 (蘇黎世)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
以色列 (特拉維夫)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
墨西哥 (中部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Middle East (Bahrain)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
中東 (阿拉伯聯合大公國)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
南美洲 (聖保羅)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
AWS GovCloud（美國東部）	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本
AWS GovCloud（美國西部）	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.4 版及更新版本	12.8 版及更新版本	11.9 版和 11.13 版以及更新版本

Secrets Manager 整合的支援區域和 Aurora 資料庫引擎

使用 AWS Secrets Manager時，您可以使用 API 呼叫 Secrets Manager 以程式設計方式擷取秘密，取代程式碼中的硬式編碼登入資料，包括資料庫密碼。如需 Secrets Manager 的詳細資訊，請參閱 AWS Secrets Manager 使用者指南。

您可以指定 Amazon Aurora 在 Aurora 資料庫叢集的 Secrets Manager 中管理主要使用者密碼。Aurora 會產生密碼、將其存放在 Secrets Manager 中，並定期對其進行輪換。如需詳細資訊，請參閱使用 Amazon Aurora 和 進行密碼管理 AWS Secrets Manager。

Secrets Manager 整合適用於所有 AWS 區域。

Aurora Serverless v2 支援的區域和 Aurora 資料庫引擎

Aurora Serverless v2 是隨需、自動調整規模功能，設計成為在 Amazon Aurora 上執行間歇性或不可預測工作負載之符合成本效益的方法。它可根據應用程式的需要而自動擴展或縮減容量。比 Aurora Serverless v1 更快、更精細的擴展。利用 Aurora Serverless v2，每個叢集可包含一個寫入器資料庫執行個體和多個讀取器資料庫執行個體。您可在相同叢集中結合 Aurora Serverless v2 和傳統的佈建資料庫執行個體。如需詳細資訊，請參閱使用 Aurora Serverless v2。

主題

- Aurora Serverless v2 搭配 Aurora MySQL
- Aurora Serverless v2 搭配 Aurora PostgreSQL

Aurora Serverless v2 搭配 Aurora MySQL

下列區域和引擎版本可供搭配 Aurora MySQL 的 Aurora Serverless v2 使用。

區域	Aurora MySQL 第 3 版
美國東部 (維吉尼亞北部)	3.02.0 版及更新版本
美國東部 (俄亥俄)	3.02.0 版及更新版本
美國西部 (加利佛尼亞北部)	3.02.0 版及更新版本
美國西部 (奧勒岡)	3.02.0 版及更新版本
非洲 (開普敦)	3.02.0 版及更新版本
亞太地區 (香港)	3.02.0 版及更新版本
亞太地區 (海德拉巴)	3.02.3 版及更新版本
亞太地區 (雅加達)	3.02.0 版及更新版本
亞太地區 (馬來西亞)	3.04.3、3.05.2、3.06.1、3.07.1 版及更新版本
亞太地區 (墨爾本)	3.02.3 版及更新版本

區域	Aurora MySQL 第 3 版
亞太地區 (孟買)	3.02.0 版及更新版本
亞太區域 (紐西蘭)	3.04.3 版和更新版本、3.08.0 版和更新版本
亞太地區 (大阪)	3.02.0 版及更新版本
亞太地區 (首爾)	3.02.0 版及更新版本
亞太地區 (新加坡)	3.02.0 版及更新版本
亞太地區 (悉尼)	3.02.0 版及更新版本
亞太區域 (台北)	3.04.3 版和更新版本、3.08.1 版和更新版本
亞太區域 (泰國)	3.04.3 版和更新版本、3.08.0 版和更新版本
亞太地區 (東京)	3.02.0 版及更新版本
加拿大 (中部)	3.02.0 版及更新版本
加拿大西部 (卡加利)	3.04.0 版及更新版本
中國 (北京)	3.02.2 版及更新版本
中國 (寧夏)	3.02.2 版及更新版本
歐洲 (法蘭克福)	3.02.0 版及更新版本
歐洲 (愛爾蘭)	3.02.0 版及更新版本
歐洲 (倫敦)	3.02.0 版及更新版本
歐洲 (米蘭)	3.02.0 版及更新版本
Europe (Paris)	3.02.0 版及更新版本
歐洲 (西班牙)	3.02.3 版及更新版本
歐洲 (斯德哥爾摩)	3.02.0 版及更新版本

區域	Aurora MySQL 第 3 版
歐洲 (蘇黎世)	3.02.3 版及更新版本
以色列 (特拉維夫)	3.02.3 版及更新版本、3.03.1 版及更新版本
墨西哥 (中部)	3.04.3 版和更新版本、3.08.0 版和更新版本
Middle East (Bahrain)	3.02.0 版及更新版本
中東 (阿拉伯聯合大公國)	3.02.3 版及更新版本
南美洲 (聖保羅)	3.02.0 版及更新版本
AWS GovCloud （美國東部）	3.02.2 版及更新版本
AWS GovCloud （美國西部）	3.02.2 版及更新版本

Aurora Serverless v2 容量的 ACU 上限和下限可能會因引擎版本而有所不同。如需詳細資訊，請參閱Aurora Serverless v2 容量。

Aurora Serverless v2 搭配 Aurora PostgreSQL

下列區域和引擎版本可供搭配 Aurora PostgreSQL 的 Aurora Serverless v2 使用。

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
美國東部 (維吉尼亞北部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
美國東部 (俄亥俄)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
美國西部 (加利佛尼亞北部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
美國西部 (奧勒岡)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
非洲 (開普敦)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
亞太地區 (香港)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
亞太地區 (海德拉巴)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.6 版和更新版本	13.9 版及更新版本
亞太地區 (雅加達)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
亞太地區 (馬來西亞)	17.4 版及更新版本	16.1 版及更新版本	15.4 版及更新版本	14.6、14.9 版及更新版本	13.9、13.12 版及更新版本
亞太地區 (墨爾本)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.6 版和更新版本	13.9 版及更新版本
亞太地區 (孟買)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
亞太區域 (紐西蘭)	17.4 版及更新版本	16.8 版及更新版本	15.12 版及更新版本	14.17 版及更新版本	13.20 版及更新版本
亞太地區 (大阪)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
亞太地區 (首爾)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
亞太地區 (新加坡)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
亞太區域 (台北)	17.4 版及更新版本	16.6 版及更新版本	15.10 版及更新版本	14.15 版及更新版本	不適用
亞太地區 (悉尼)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
亞太區域 (泰國)	17.4 版及更新版本	16.4 版及更新版本	15.8 版及更新版本	14.13 版及更新版本	不適用
亞太地區 (東京)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
加拿大 (中部)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
加拿大西部 (卡加利)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.6、14.8 版及更新版本	13.9、13.11 版及更新版本
中國 (北京)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
中國 (寧夏)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
歐洲 (法蘭克福)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
歐洲 (愛爾蘭)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
歐洲 (倫敦)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
歐洲 (米蘭)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
Europe (Paris)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
歐洲 (西班牙)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.6 版和更新版本	13.9 版及更新版本
歐洲 (斯德哥爾摩)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
歐洲 (蘇黎世)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.6 版和更新版本	13.9 版及更新版本
以色列 (特拉維夫)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.6 版和更新版本	13.9 版及更新版本
墨西哥 (中部)	17.4 版及更新版本	16.4 版及更新版本	15.8 版及更新版本	14.13 版及更新版本	不適用
Middle East (Bahrain)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
中東 (阿拉伯聯合大公國)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.6 版和更新版本	13.9 版及更新版本
南美洲 (聖保羅)	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
AWS GovCloud （美國東部）	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本
AWS GovCloud （美國西部）	17.4 版及更新版本	16.1 版及更新版本	15.2 版及更新版本	14.3 版及更新版本	13.6 版及更新版本

Aurora Serverless v2 容量的 ACU 上限和下限可能會因引擎版本而有所不同。如需詳細資訊，請參閱Aurora Serverless v2 容量。

Aurora Serverless v1

> **Important**
>
> AWS 已宣布 的end-of-lifeAurora Serverless v1：2025 年 3 月 31 日。我們強烈建議您在此日期之前，將任何 Aurora Serverless v1 資料庫叢集升級至 Aurora Serverless v2。升級可能涉及資料庫引擎主要版本編號的變更。因此，請務必在終止服務日期前規劃、測試和實作此轉換。從 2025 年 1 月 8 日開始，客戶將無法再使用 或 CLI 建立新的Aurora Serverless v1叢集 AWS 管理主控台 或執行個體。如需遷移程序的詳細資訊，請參閱從 Aurora Serverless v1 叢集升級至 Aurora Serverless v2。
>
> Aurora Serverless v2 擴展速度更快，更精細。Aurora Serverless v2 也與其他 Aurora 功能 (如讀取器資料庫執行個體) 具有更高的相容性。您可以於 使用 Aurora Serverless v2 中了解 Aurora Serverless v2。

Aurora Serverless v1 是隨需、自動調整規模功能，設計成為在 Amazon Aurora 上執行間歇性或不可預測工作負載之符合成本效益的方法。其會依據您應用程式的需要，使用每個叢集中的單一資料庫執行個體自動啟動、關閉和擴展或縮減容量。如需詳細資訊，請參閱使用 Amazon Aurora Serverless v1。

主題

- Aurora Serverless v1 搭配 Aurora MySQL
- Aurora Serverless v1 搭配 Aurora PostgreSQL

Aurora Serverless v1 搭配 Aurora MySQL

下列區域和引擎版本可供搭配 Aurora MySQL 的 Aurora Serverless v1 使用。

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
美國東部 (維吉尼亞北部)	不適用	2.11.4 版
美國東部 (俄亥俄)	不適用	2.11.4 版
美國西部 (加利佛尼亞北部)	不適用	2.11.4 版

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
美國西部 (奧勒岡)	不適用	2.11.4 版
非洲 (開普敦)	不適用	不適用
亞太地區 (香港)	不適用	不適用
亞太地區 (海德拉巴)	不適用	不適用
亞太地區 (雅加達)	不適用	不適用
亞太地區 (馬來西亞)	不適用	不適用
亞太地區 (墨爾本)	不適用	不適用
亞太地區 (孟買)	不適用	2.11.4 版
亞太地區 (大阪)	不適用	不適用
亞太地區 (首爾)	不適用	2.11.4 版
亞太地區 (新加坡)	不適用	2.11.4 版
亞太地區 (雪梨)	不適用	2.11.4 版
亞太區域 (泰國)	不適用	不適用
亞太地區 (東京)	不適用	2.11.4 版
加拿大 (中部)	不適用	2.11.4 版
加拿大西部 (卡加利)	不適用	不適用
中國 (北京)	不適用	不適用
中國 (寧夏)	不適用	2.11.4 版
歐洲 (法蘭克福)	不適用	2.11.4 版
歐洲 (愛爾蘭)	不適用	2.11.4 版

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
歐洲 (倫敦)	不適用	2.11.4 版
歐洲 (米蘭)	不適用	不適用
Europe (Paris)	不適用	2.11.4 版
歐洲 (西班牙)	不適用	不適用
歐洲 (斯德哥爾摩)	不適用	不適用
歐洲 (蘇黎世)	不適用	不適用
以色列 (特拉維夫)	不適用	不適用
Middle East (Bahrain)	不適用	不適用
中東 (阿拉伯聯合大公國)	不適用	不適用
南美洲 (聖保羅)	不適用	不適用
AWS GovCloud （美國東部）	不適用	不適用
AWS GovCloud （美國西部）	不適用	不適用

Aurora Serverless v1 搭配 Aurora PostgreSQL

下列區域和引擎版本可供搭配 Aurora PostgreSQL 的 Aurora Serverless v1 使用。

區域	Aurora PostgreSQL 13
美國東部 (維吉尼亞北部)	13.12 版
美國東部 (俄亥俄)	13.12 版
美國西部 (加利佛尼亞北部)	13.12 版
美國西部 (奧勒岡)	13.12 版

區域	Aurora PostgreSQL 13
非洲 (開普敦)	不適用
亞太地區 (香港)	不適用
亞太地區 (海德拉巴)	不適用
亞太地區 (雅加達)	不適用
亞太地區 (馬來西亞)	不適用
亞太地區 (墨爾本)	不適用
亞太地區 (孟買)	13.12 版
亞太地區 (大阪)	不適用
亞太地區 (首爾)	13.12 版
亞太地區 (新加坡)	13.12 版
亞太地區 (雪梨)	13.12 版
亞太區域 (泰國)	不適用
亞太地區 (東京)	13.12 版
加拿大 (中部)	13.12 版
加拿大西部 (卡加利)	不適用
中國 (北京)	不適用
中國 (寧夏)	不適用
歐洲 (法蘭克福)	13.12 版
歐洲 (愛爾蘭)	13.12 版
歐洲 (倫敦)	13.12 版

區域	Aurora PostgreSQL 13
歐洲 (米蘭)	不適用
Europe (Paris)	13.12 版
歐洲 (西班牙)	不適用
歐洲 (斯德哥爾摩)	不適用
歐洲 (蘇黎世)	不適用
以色列 (特拉維夫)	不適用
Middle East (Bahrain)	不適用
中東 (阿拉伯聯合大公國)	不適用
南美洲 (聖保羅)	不適用
AWS GovCloud （美國東部）	不適用
AWS GovCloud （美國西部）	不適用

RDS 資料 API 的支援區域和 Aurora 資料庫引擎

RDS 資料 API (資料 API) 為 Amazon Aurora 資料庫叢集提供 Web 服務界面。您可對 HTTPS 端點執行 SQL 命令，而不是從用戶端應用程式管理資料庫連線。如需詳細資訊，請參閱使用 Amazon RDS 資料 API。

主題

- 具有 Aurora PostgreSQL Serverless 第 2 版和佈建的資料 API
- 具有 Aurora MySQL Serverless 第 2 版和佈建的資料 API
- 具有 Aurora PostgreSQL Serverless 第 1 版的資料 API
- 具有 Aurora MySQL Serverless 第 1 版的資料 API

具有 Aurora PostgreSQL Serverless 第 2 版和佈建的資料 API

下列區域和引擎版本適用於使用 Aurora PostgreSQL Serverless 第 2 版及佈建資料庫叢集的資料 API。

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
美國東部 (維吉尼亞北部)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
美國東部 (俄亥俄)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
美國西部 (加利佛尼亞北部)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
美國西部 (奧勒岡)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
非洲 (開普敦)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
亞太地區 (香港)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
亞太地區 (海德拉巴)	不適用	不適用	不適用	不適用	不適用
亞太地區 (雅加達)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
亞太地區 (馬來西亞)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
亞太地區 (墨爾本)	不適用	不適用	不適用	不適用	不適用

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
亞太地區 (孟買)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
亞太區域 (紐西蘭)	不適用	不適用	不適用	不適用	不適用
亞太地區 (大阪)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
亞太地區 (首爾)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
亞太地區 (新加坡)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
亞太地區 (悉尼)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
亞太區域 (台北)	不適用	不適用	不適用	不適用	不適用
亞太區域 (泰國)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
亞太地區 (東京)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
加拿大 (中部)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
加拿大西部 (卡加利)	不適用	不適用	不適用	不適用	不適用
中國 (北京)	不適用	不適用	不適用	不適用	不適用
中國 (寧夏)	不適用	不適用	不適用	不適用	不適用

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
歐洲 (法蘭克福)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
歐洲 (愛爾蘭)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
歐洲 (倫敦)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
歐洲 (米蘭)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
Europe (Paris)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
歐洲 (西班牙)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
歐洲 (斯德哥爾摩)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
歐洲 (蘇黎世)	不適用	不適用	不適用	不適用	不適用
以色列 (特拉維夫)	不適用	不適用	不適用	不適用	不適用
Middle East (Bahrain)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
墨西哥 (中部)	不適用	不適用	不適用	不適用	不適用
中東 (阿拉伯聯合大公國)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本
南美洲 (聖保羅)	17.4 版及更新版本	16.1 版及更新版本	15.3 版及更新版本	14.8 版及更新版本	13.11 版及更新版本

區域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
AWS GovCloud（美國東部）	不適用	不適用	不適用	不適用	不適用
AWS GovCloud（美國西部）	不適用	不適用	不適用	不適用	不適用

具有 Aurora MySQL Serverless 第 2 版和佈建的資料 API

下列區域和引擎版本適用於使用 Aurora MySQL Serverless 第 2 版及佈建資料庫叢集的資料 API。

區域	Aurora MySQL 第 3 版
美國東部 (俄亥俄)	3.07 版及更新版本
美國東部 (維吉尼亞北部)	3.07 版及更新版本
美國西部 (加利佛尼亞北部)	3.07 版及更新版本
美國西部 (奧勒岡)	3.07 版及更新版本
非洲 (開普敦)	3.07 版及更新版本
亞太地區 (香港)	3.07 版及更新版本
亞太地區 (海德拉巴)	不適用
亞太地區 (雅加達)	3.07 版及更新版本
亞太地區 (馬來西亞)	3.07 版及更新版本
亞太地區 (墨爾本)	不適用
亞太地區 (孟買)	3.07 版及更新版本

區域	Aurora MySQL 第 3 版
亞太區域 (紐西蘭)	不適用
亞太地區 (大阪)	3.07 版及更新版本
亞太地區 (首爾)	3.07 版及更新版本
亞太地區 (新加坡)	3.07 版及更新版本
亞太地區 (悉尼)	3.07 版及更新版本
亞太區域 (台北)	不適用
亞太區域 (泰國)	3.07 版及更新版本
亞太地區 (東京)	3.07 版及更新版本
加拿大 (中部)	3.07 版及更新版本
加拿大西部 (卡加利)	不適用
中國 (北京)	不適用
中國 (寧夏)	不適用
歐洲 (法蘭克福)	3.07 版及更新版本
歐洲 (愛爾蘭)	3.07 版及更新版本
歐洲 (倫敦)	3.07 版及更新版本
歐洲 (米蘭)	3.07 版及更新版本
Europe (Paris)	3.07 版及更新版本
歐洲 (西班牙)	3.07 版及更新版本
歐洲 (斯德哥爾摩)	3.07 版及更新版本
歐洲 (蘇黎世)	不適用

區域	Aurora MySQL 第 3 版
以色列 (特拉維夫)	不適用
墨西哥 (中部)	不適用
Middle East (Bahrain)	3.07 版及更新版本
中東 (阿拉伯聯合大公國)	3.07 版及更新版本
南美洲 (聖保羅)	3.07 版及更新版本
AWS GovCloud （美國東部）	不適用
AWS GovCloud （美國西部）	不適用

具有 Aurora PostgreSQL Serverless 第 1 版的資料 API

下列區域和引擎版本可供搭配 Aurora PostgreSQL Serverless 第 1 版的資料 API 使用。

區域	Aurora PostgreSQL 13	Aurora PostgreSQL 11
美國東部 (維吉尼亞北部)	13.9 版	11.18 版
美國東部 (俄亥俄)	13.9 版	11.18 版
美國西部 (加利佛尼亞北部)	13.9 版	11.18 版
美國西部 (奧勒岡)	13.9 版	11.18 版
非洲 (開普敦)	不適用	不適用
亞太地區 (香港)	不適用	不適用
亞太地區 (海德拉巴)	不適用	不適用
亞太地區 (雅加達)	不適用	不適用
亞太地區 (馬來西亞)	不適用	不適用

區域	Aurora PostgreSQL 13	Aurora PostgreSQL 11
亞太地區 (墨爾本)	不適用	不適用
亞太地區 (孟買)	13.9 版	11.18 版
亞太地區 (大阪)	不適用	不適用
亞太地區 (首爾)	13.9 版	11.18 版
亞太地區 (新加坡)	13.9 版	11.18 版
亞太地區 (悉尼)	13.9 版	11.18 版
亞太區域 (泰國)	不適用	不適用
亞太地區 (東京)	13.9 版	11.18 版
加拿大 (中部)	13.9 版	11.18 版
中國 (北京)	不適用	不適用
中國 (寧夏)	不適用	不適用
歐洲 (法蘭克福)	13.9 版	11.18 版
歐洲 (愛爾蘭)	13.9 版	11.18 版
歐洲 (倫敦)	13.9 版	11.18 版
歐洲 (米蘭)	不適用	不適用
Europe (Paris)	13.9 版	11.18 版
歐洲 (西班牙)	13.9 版	11.18 版
歐洲 (斯德哥爾摩)	不適用	不適用
歐洲 (蘇黎世)	不適用	不適用
以色列 (特拉維夫)	不適用	不適用

區域	Aurora PostgreSQL 13	Aurora PostgreSQL 11
Middle East (Bahrain)	不適用	不適用
中東 (阿拉伯聯合大公國)	不適用	不適用
南美洲 (聖保羅)	不適用	不適用
AWS GovCloud （美國東部）	不適用	不適用
AWS GovCloud （美國西部）	不適用	不適用

具有 Aurora MySQL Serverless 第 1 版的資料 API

下列區域和引擎版本可供搭配 Aurora MySQL Serverless 第 1 版的資料 API 使用。

區域	Aurora MySQL 第 2 版
美國東部 (維吉尼亞北部)	2.11.3 版
美國東部 (俄亥俄)	2.11.3 版
美國西部 (加利佛尼亞北部)	2.11.3 版
美國西部 (奧勒岡)	2.11.3 版
非洲 (開普敦)	不適用
亞太地區 (香港)	不適用
亞太地區 (海德拉巴)	不適用
亞太地區 (雅加達)	不適用
亞太地區 (馬來西亞)	不適用
亞太地區 (墨爾本)	不適用
亞太地區 (孟買)	2.11.3 版

區域	Aurora MySQL 第 2 版
亞太地區 (大阪)	不適用
亞太地區 (首爾)	2.11.3 版
亞太地區 (新加坡)	2.11.3 版
亞太地區 (悉尼)	2.11.3 版
亞太區域 (泰國)	不適用
亞太地區 (東京)	2.11.3 版
加拿大 (中部)	2.11.3 版
加拿大西部 (卡加利)	不適用
中國 (北京)	不適用
中國 (寧夏)	2.11.3 版
歐洲 (法蘭克福)	2.11.3 版
歐洲 (愛爾蘭)	2.11.3 版
歐洲 (倫敦)	2.11.3 版
歐洲 (米蘭)	不適用
Europe (Paris)	2.11.3 版
歐洲 (西班牙)	2.11.3 版
歐洲 (斯德哥爾摩)	不適用
歐洲 (蘇黎世)	不適用
以色列 (特拉維夫)	不適用
Middle East (Bahrain)	不適用

區域	Aurora MySQL 第 2 版
中東 (阿拉伯聯合大公國)	不適用
南美洲 (聖保羅)	不適用
AWS GovCloud （美國東部）	不適用
AWS GovCloud （美國西部）	不適用

支援零停機時間修補 (ZDP) 的支援區域和 Aurora 資料庫引擎

執行資料 Aurora 資料庫叢集的升級牽涉到資料庫關閉和升級時發生中斷的可能性。根據預設，如果您在資料庫忙碌時開始升級，則會遺失資料庫叢集正在處理的所有連線和交易。如果您要等到資料庫閒置才執行升級，就可能必須等待很長時間。

零停機時間修補 (ZDP) 功能以最佳作法為基礎，試圖在整個 Aurora 升級過程維持用戶端正常連線。如果 ZDP 成功完成，即可保留應用程式工作階段，資料庫引擎也會在升級期間重新啟動。資料庫引擎重新啟動可能會導致輸送量下降，持續時間為數秒到一分鐘左右。

如需在哪些條件下及哪些引擎版本中 ZDP 適用於 Aurora MySQL 升級的詳細資訊，請參閱 使用零停機時間修補。

如需在哪些條件下及哪些引擎版本中 ZDP 適用於 Aurora PostgreSQL 升級的詳細資訊，請參閱 次要版本升級和零停機時間修補。

支援的區域 Aurora PostgreSQL Limitless Database

Amazon Aurora PostgreSQL Limitless Database 提供自動化水平擴展，每秒處理數百萬筆寫入交易以及管理 PB 級資料，同時維持在單一資料庫中操作的簡單性。使用 Aurora PostgreSQL Limitless Database，您就可以專注於建置大規模應用程式，無需為跨多個資料庫執行個體擴展資料以支援工作負載，而費心建置和維護複雜的解決方案。

如需更多詳細資訊，請參閱 使用 Amazon Aurora PostgreSQL Limitless Database。

Aurora PostgreSQL Limitless Database 適用於所有 AWS 區域 (亞太區域 (台北) 除外)。

Aurora 引擎原生功能的支援區域和資料庫引擎

Aurora 資料庫引擎也支援專門針對 Aurora 的其他功能。部分引擎原生功能對於特定 Aurora 資料庫引擎、版本或區域的支援或權限可能有所限制。

主題

- Aurora MySQL 的引擎原生功能
- Aurora PostgreSQL 的引擎原生功能

Aurora MySQL 的引擎原生功能

以下是 Aurora MySQL 的引擎原生功能。

- 進階稽核
- 恢復
- 故障注入查詢
- 叢集內寫入轉送
- 平行查詢

Aurora PostgreSQL 的引擎原生功能

以下是 Aurora PostgreSQL 的引擎原生功能。

- Babelfish
- 故障注入查詢
- 查詢計劃管理

Amazon Aurora 端點連線

Amazon Aurora 一般牽涉到一個資料庫執行個體的叢集，而非單一執行個體。每個連線會由特定資料庫執行個體處理。連線到 Aurora 叢集時，您指定的主機名稱和連線埠會指向稱為端點的中繼處理常式。Aurora 會使用端點機制來抽象處理這些連線。因此，您不需將所有主要名稱硬式編碼，或編寫您自己的平衡邏輯，並在資料庫執行個體無法使用時重新路由連線。

針對某些 Aurora 任務，不同的執行個體或執行個體群組會執行不同的角色。例如，主要執行個體會處理所有資料定義語言 (DDL) 和資料操作語言 (DML) 陳述式。最多會有 15 個 Aurora 複本可處理唯讀查詢流量。

主題

- Aurora 端點的類型
- 檢視 Aurora 叢集的端點
- Aurora 端點如何與高可用性搭配使用
- Amazon Aurora 的叢集端點
- Amazon Aurora 的讀取器端點
- Amazon Aurora 的執行個體端點
- Amazon Aurora 的自訂端點

Aurora 端點的類型

使用端點，您可以將每個連線根據您的使用案例對應至適當的執行個體或執行個體群組。例如，若要執行 DDL 陳述式，您可以連線至屬於主要執行個體的執行個體。若要執行查詢，您可以連線至讀取器端點，而 Aurora 會在所有 Aurora 複本間自動執行連線平衡。針對具有不同容量或組態之資料庫執行個體的叢集，您可以連線至與不同資料庫執行個體子集相關聯的自訂端點。針對診斷或調校，您可以連線至特定執行個體端點來檢查關於特定資料庫執行個體的詳細資訊。

端點代表包含主機地址和連線埠的 Aurora 特定 URL。下列類型的端點可從 Aurora 資料庫叢集取得。

叢集端點

連線至叢集的主要執行個體以開發和測試應用程式，並執行轉換，例如 `INSERT` 陳述式和 DDL、DML 和 ETL 操作。使用 AWS 管理主控台 AWS CLI或 Amazon RDS API 尋找叢集端點位置，如中所述檢視 Aurora 叢集的端點。

如需叢集端點的詳細資訊，請參閱 Amazon Aurora 的叢集端點。

讀取器端點

執行查詢 Aurora 會自動在所有 Aurora 複本之間執行連線平衡。使用 AWS 管理主控台 AWS CLI或 Amazon RDS API 尋找讀取器端點位置，如中所述檢視 Aurora 叢集的端點。

如需讀取器端點的詳細資訊，請參閱 Amazon Aurora 的讀取器端點。

執行個體端點

檢查特定資料庫執行個體的詳細資訊，以進行診斷或調校。您可以在執行個體的執行個體詳細資訊頁面上， AWS 管理主控台 只在 中找到每個執行個體的執行個體端點位置。

如需執行個體端點的詳細資訊，請參閱 Amazon Aurora 的執行個體端點。

自訂端點

連線至資料庫叢集上的不同資料庫執行個體的子集。當您的資料庫叢集中有不同的執行個體容量和組態時，此功能非常有用。使用 AWS 管理主控台 AWS CLI或 Amazon RDS API 尋找自訂端點位置，如 中所述檢視 Aurora 叢集的端點。

如需自訂端點的詳細資訊，請參閱 Amazon Aurora 的自訂端點。

Aurora 全球資料庫寫入器端點

Aurora 全球資料庫具有特殊類型的端點，其用途與獨立 Aurora 叢集的叢集端點相同。它會同時處理寫入和讀取請求。當次要叢集因為切換或容錯移轉而成為新的主要叢集時，Aurora 會自動切換此端點以指向其他 AWS 區域中新主要叢集的叢集端點。如此一來，您就不必將 AWS 區域編碼為應用程式的連線字串，也不必在全域資料庫的配置變更時變更連線字串。當您在 AWS 管理主控台中為 Aurora 叢集選擇新增區域，Aurora 會在設定 Aurora 全球資料庫時建立此端點。

如需如何搭配 Aurora 全球資料庫使用此類型端點的詳細資訊，請參閱 連線至 Amazon Aurora 全域資料庫。

檢視 Aurora 叢集的端點

雖然您只能在 的執行個體詳細資訊頁面上找到執行個體端點位置 AWS 管理主控台，但您可以使用 主控台 AWS CLI或 Amazon RDS API 來尋找叢集、讀取器和自訂端點的位置。

Console

在 中 AWS 管理主控台，尋找叢集端點、讀取器端點，以及叢集執行個體詳細資訊頁面中的任何自訂端點。您可以在每個執行個體的詳細資訊頁面中查看執行個體端點。連線時，請將相關聯的連線埠號碼 (後面接著冒號)，附加到詳細資訊頁面上顯示的端點名稱。

AWS CLI

使用 AWS CLI，您可以在 describe-db-clusters 命令的輸出中找到寫入器、讀取器和任何自訂端點。例如，下列命令會顯示目前 AWS 區域中所有叢集的端點屬性。

```
aws rds describe-db-clusters --query '*[].
{Endpoint:Endpoint,ReaderEndpoint:ReaderEndpoint,CustomEndpoints:CustomEndpoints}'
```

Amazon RDS API

使用 Amazon RDS API，您可以呼叫 DescribeDBClusterEndpoints 操作來擷取端點。

Aurora 端點如何與高可用性搭配使用

若為高可用性很重要的叢集，請針對讀寫或一般用途連線使用寫入器端點，以及針對唯讀連線使用叢集端點。寫入者和讀取者端點比執行個體端點更能善加管理資料庫執行個體容錯移轉。與執行個體端點不同的是，如果叢集中的資料庫執行個體變得無法使用，寫入者和讀取者端點會自動變更它們要連線到哪個資料庫執行個體。如需叢集和讀取器端點的詳細資訊，請參閱 Amazon Aurora 的叢集端點 和 Amazon Aurora 的讀取器端點。

如果資料庫叢集的主要資料庫執行個體失敗，Aurora 會自動容錯移轉至新的主要資料庫執行個體。它會透過將現有的 Aurora 複本提升為新的主要資料庫執行個體，或是建立新的主要資料庫執行個體來完成。如果發生容錯移轉，您可以使用叢集端點來重新連接至新提升或建立的主要資料庫執行個體，或使用讀取器端點來重新連接至資料庫叢集的其中一個 Aurora 複本。在容錯移轉期間，在 Aurora 複本提升為新的主要資料庫執行個體之後，讀取器端點可能會短暫直接連線至資料庫叢集的新主要資料庫執行個體。

如果您要設計自己的應用程式邏輯來管理執行個體端點的連線，則可手動或以程式設計方式探索資料庫叢集中可用資料庫執行個體的結果集。使用 describe-db-clusters AWS CLI 命令或 DescribeDBClusters RDS API 操作來尋找資料庫叢集和讀取器端點、資料庫執行個體、資料庫執行個體是否為讀取器及其提升層。接著，您可以在容錯移轉後確認其執行個體類別，並連線至適當的執行個體端點。

如需容錯移轉的詳細資訊，請參閱 Aurora 資料庫叢集的容錯能力。

如需 Amazon Aurora 中高可用性的詳細資訊，請參閱 Amazon Aurora 的高可用性。

Amazon Aurora 的叢集端點

Aurora 資料庫叢集的「叢集端點」(或「寫入者端點」) 會連線至該資料庫叢集目前的主要資料庫執行個體。此端點為可執行寫入操作 (如 DDL 陳述式) 的端點。因此，當您先設定叢集時或當叢集僅包含單一資料庫執行個體時，該叢集端點為您要連線的端點。

每個 Aurora 資料庫叢集具有一個叢集端點和一個主要資料庫執行個體。

您會對資料庫叢集上的所有寫入操作，包括插入、更新、刪除和 DDL 變更使用該叢集端點。您也可以對讀取操作 (例如查詢) 使用叢集端點。

叢集端點可為資料庫叢集的讀寫連線提供容錯移轉支援。如果資料庫叢集目前的主要資料庫執行個體失敗，Aurora 會自動容錯移轉至新的主要資料庫執行個體。容錯移轉期間，資料庫叢集會繼續從新的主要資料庫執行個體對叢集端點提供連線請求，將對服務的中斷降到最低。

下列範例說明 Aurora MySQL 資料庫叢集的叢集端點。

```
mydbcluster.cluster-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

每個 Aurora 叢集有單一的內建叢集端點，其名稱和其他屬性是由 Aurora 管理。您無法建立、刪除或修改這類型的端點。

您會在管理叢集、執行「擷取、轉換、載入 (ETL)」操作，或是開發和測試應用程式時使用叢集端點。叢集端點會連線至叢集的主要執行個體。主要執行個體是您可以建立資料表和索引、執行 INSERT 陳述式，以及執行其他 DDL 和 DML 操作的唯一資料庫執行個體。

叢集端點所指向的實體 IP 地址，會在容錯移轉機制將新資料庫執行個體提升為叢集的讀寫主要執行個體時變更。如果您使用任何形式的連線集區或其他多工，請準備清空或降低任何快取的 DNS 資訊的存留時間。這麼做可確保您不會嘗試對變得無法使用或在容錯移轉之後現在為唯讀的資料庫執行個體建立讀寫連線。

Amazon Aurora 的讀取器端點

Aurora 資料庫叢集的讀取器端點可為資料庫叢集的唯讀連線提供連線平衡支援。對讀取操作 (例如查詢) 使用讀取器端點。藉由在唯讀 Aurora 複本上處理這些陳述式，此端點可降低主要執行個體上的額外負荷。它還有助於叢集擴展能力來處理同時的 SELECT 查詢，數量與叢集中的 Aurora 複本成比例。每個 Aurora 資料庫叢集有一個讀取器端點。

如果叢集包含一或多個 Aurora 複本，則讀取器端點會平衡 Aurora 複本之間的每個連線請求。在這種情況下，您在該工作階段中只能執行唯讀陳述式，例如 SELECT。如果叢集只包含主要執行個體而沒有 Aurora 複本，則讀取者端點會連線到主要執行個體。在這種情況下，您可以透過端點執行寫入操作。

下列範例說明 Aurora MySQL 資料庫叢集的讀取器端點。

```
mydbcluster.cluster-ro-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

您可以為 Aurora 叢集的唯讀連線使用讀取器端點。此端點會使用連線平衡機制來幫助您的叢集處理查詢密集的工作負載。讀取器端點是您提供給應用程式以在叢集上執行報告或其他唯讀操作的端點。

讀取器端點會對 Aurora 資料庫叢集中可用的 Aurora 複本的連線進行平衡。它不會對個別查詢進行平衡。如果您想要對每個查詢進行平衡，以分配資料庫叢集的讀取工作負載，請對每個查詢的讀取器端點開啟一個新的連線。

每個 Aurora 叢集有單一的內建讀取器端點，其名稱和其他屬性是由 Aurora 管理。您無法建立、刪除或修改這類型的端點。

如果您的叢集只包含主要目標 (執行個體或資料庫碎片群組) 而沒有 Aurora 複本，則讀取器端點會連線到主要執行個體。在這種情況下，您可以透過此端點執行寫入操作。

> **Tip**
>
> 透過 RDS Proxy，您可以為 Aurora 叢集建立其他的僅供讀取端點。這些端點執行的連線平衡類型與 Aurora 讀取器端點相同。如果讀取者執行個體變得無法使用，則應用程式可以更快速地重新連線至代理端點，而非 Aurora 讀取者端點。代理端點也可以利用其他代理功能，例如多工處理。如需更多詳細資訊，請參閱 將讀取器端點與 Aurora 叢集搭配使用。

Amazon Aurora 的執行個體端點

「執行個體端點」會連線至 Aurora 叢集內的特定資料庫執行個體。資料庫叢集中的每個資料庫執行個體都有自己唯一的執行個體端點。因此，資料庫叢集目前的主要資料庫執行個體會有一個執行個體端點，而資料庫叢集中的每個 Aurora 複本也都會有一個執行個體端點。

執行個體端點對資料庫叢集的連線提供直接控制，使用叢集端點或讀取器端點的案例可能不適用。例如，根據工作負載類型而定，您的用戶端應用程式可能要更精細的連線平衡。在此情況下，您可以設定多個用戶端來連線至資料庫叢集中的不同 Aurora 複本，以分配讀取工作負載。如需在 Aurora PostgreSQL 容錯移轉之後使用執行個體端點來改善連線速度的範例，請參閱 Amazon Aurora PostgreSQL 的快速容錯移轉。如需在 Aurora MySQL 容錯移轉之後使用執行個體端點來改善連線速度的範例，請參閱 MariaDB Connector/J 容錯移轉支援 - 案例 Amazon Aurora。

下列範例說明 Aurora MySQL 資料庫叢集中資料庫執行個體的執行個體端點。

```
mydbinstance.c7tj4example.us-east-1.rds.amazonaws.com:3306
```

Aurora 叢集中的每個資料庫執行個體會有其自己的內建執行個體端點，其名稱和其他屬性是由 Aurora 管理。您無法建立、刪除或修改這類型的端點。如果您使用 Amazon RDS，您可能很熟悉執行個體端點。不過，透過 Aurora，您通常會比執行個體端點更常使用寫入者和讀取者端點。

在日常 Aurora 操作中，您使用執行個體端點的主要方式是用來診斷會影響 Aurora 叢集中特定執行個體的容量或效能問題。連線至特定執行個體時，您可以檢查其狀態變數、指標等等。這麼做可幫助您判斷該執行個體中所發生情況，與叢集中其他執行個體所發生情況的差異。

在進階使用案例中，您可能會以不同方式設定一些資料庫執行個體。在此案例中，請使用執行個體端點來直接連線至較小型、較大型或與其他執行個體具有不同特質的執行個體。同時，設定容錯移轉優先順序，使得此特殊資料庫執行個體是要用做主要執行個體的最後選擇。在此情況下，我們建議您使用自訂端點而非執行個體端點。這麼做可以在您新增更多資料庫執行個體至您的叢集時簡化連線管理和高可用性。

Amazon Aurora 的自訂端點

Aurora 叢集的自訂端點代表您選擇的一組資料庫執行個體。連線至端點時，Aurora 會執行連線平衡，並選擇群組的其中一個執行個體來處理連線。您可以定義此端點要參考的執行個體，並且決定端點的運作目的。

Aurora 資料庫叢集沒有自訂端點，除非您加以建立。您可以為每個佈建的 Aurora 叢集或 Aurora Serverless v2 叢集建立最多五個自訂端點。您無法對 Aurora Serverless v1 叢集使用自訂端點。

自訂端點可根據資料庫執行個體的唯讀或讀寫功能以外的準則來提供平衡的資料庫連線。例如，您可以定義自訂端點來連線至使用特定 AWS 執行個體類別或特定資料庫參數群組的執行個體。之後您可以將此自訂端點告知特定使用者群組。例如，您可以將內部使用者導向低容量的執行個體，用於報告產生或臨機操作 (一次性) 查詢，並將生產流量導向高容量執行個體。

由於連線可能進入與自訂端點相關聯的任何資料庫執行個體，建議您確定該群組內的所有資料庫執行個體具有類似的特質。這麼做可確保效能、記憶體容量等等對連線至該端點的每個人來說都是一致的。

此功能預期供具有專精類型工作負載的進階使用者使用，在其工作負載中，讓叢集中的所有 Aurora 複本保持相同並不切實際。利用自訂端點，您可以預測用於每個連線的資料庫執行個體的容量。使用自訂端點時，您一般不會對該叢集使用讀取器端點。

下列範例說明 Aurora MySQL 資料庫叢集中資料庫執行個體的自訂端點。

```
myendpoint.cluster-custom-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

當您的叢集包含的資料庫執行個體具有不同容量和組態設定時，您可以使用自訂端點來簡化連線管理。

在過去，您可能是使用 CNAMES 機制從您的自己網域設定網域名稱服務 (DNS) 別名，以達成類似的結果。透過使用自訂端點，當叢集成長或縮減時，您可以避免更新 CNAME 記錄。自訂端點也表示您可以使用加密的 Transport Layer Security/Secure Sockets Layer (TLS/SSL) 連線。

不要對每個專業用途使用一個資料庫執行個體並連線至其執行個體端點，而是可以有多個專業資料庫執行個體的群組。在此情況下，每個群組會有自己的自訂端點。以此方式，Aurora 可以在專用於報告或處理生產或內部查詢之類任務的所有執行個體間執行連線平衡。自訂端點會使用 DNS 隨機傳回其中一個執行個體的 IP 位址，在執行個體之間被動分配連線。如果某個群組內的其中一個資料庫執行個體變得無法使用，Aurora 會將後續的自訂端點連線導向與同一端點相關聯的其他資料庫執行個體中的一個。

主題

- Amazon Aurora 中自訂端點的考量
- 建立自訂端點
- 檢視自訂端點
- 編輯自訂端點
- 刪除自訂端點
- AWS CLI Amazon Aurora 自訂端點的範例

Amazon Aurora 中自訂端點的考量

使用下列各節來管理、指定屬性，以及使用自訂端點的成員資格規則。

主題

- 管理自訂端點
- 指定自訂端點的屬性
- 自訂端點的成員資格規則

管理自訂端點

由於新建立的 Aurora 叢集沒有自訂端點，則您必須自行建立和管理這些物件。您可以使用 AWS 管理主控台 AWS CLI或 Amazon RDS API 來執行此操作。

> **Note**
>
> 您還必須為從快照還原的 Aurora 叢集建立和管理自訂端點。快照中不會包含自訂端點。如果還原的叢集位於與原始叢集相同的區域，您會在還原之後建立它們，並選擇端點名稱。

若要從 使用自訂端點 AWS 管理主控台，您可以導覽至 Aurora 叢集的詳細資訊頁面，並使用自訂端點區段下的控制項。

若要從 使用自訂端點 AWS CLI，您可以使用下列操作：

- create-db-cluster-endpoint
- describe-db-cluster-endpoints
- modify-db-cluster-endpoint
- delete-db-cluster-endpoint

若要透過 Amazon RDS API 使用自訂端點，您可以使用下列函數：

- CreateDBClusterEndpoint
- DescribeDBClusterEndpoints
- ModifyDBClusterEndpoint
- DeleteDBClusterEndpoint

指定自訂端點的屬性

自訂端點名稱的長度上限為 63 個字元。名稱格式如下：

```
endpoint_name.cluster-custom-customer_DNS_identifier.AWS_Region.rds.amazonaws.com
```

您不可以對同一 AWS 區域中的一個以上叢集重複使用相同的自訂端點名稱。自訂 DNS 識別符是與特定 AWS 帳戶 中的 相關聯的唯一識別符 AWS 區域。

每個自訂端點有相關聯的類型，決定哪些資料庫執行個體符合資格可與該端點產生關聯。目前，類型可以是 READER 或 ANY。自訂端點類型適用下列考量：

- 您無法在 AWS 管理主控台中選取自訂端點類型。您透過 建立的所有自訂端點 AWS 管理主控台 都有 類型ANY。

  您可以使用 AWS CLI 或 Amazon RDS API 來設定和修改自訂端點類型。
- 僅讀取器資料庫執行個體可以成為 READER 自訂端點的一部分。
- 讀取器和寫入器資料庫執行個體都可以是 ANY 自訂端點。Aurora 會以相同概率，將 ANY 類型之叢集端點的連線導向任何相關的資料庫執行個體。ANY 類型適用於使用任何複寫拓撲的叢集。

- 如果您嘗試使用未適當根據叢集的複寫組態的類型來建立自訂端點，Aurora 會傳回錯誤。

自訂端點的成員資格規則

新增資料庫執行個體至自訂端點或將它從自訂端點移除時，對該資料庫執行個體的任何現有連線會保持作用中。

您可以定義要從自訂端點包括或排除的資料庫執行個體清單。我們將這些清單分別稱為靜態和排除清單。您可以使用包含/排除機制來進一步細分資料庫執行個體的群組，以及確定自訂端點集涵蓋叢集中的所有資料庫執行個體。每個自訂端點只能包含這些清單類型的其中一個。

在 中 AWS 管理主控台：

- 此選擇會以核取方塊 Attach future instances added to this cluster (連線新增至此叢集的未來執行個體) 呈現。將該核取方塊保持未選取時，自訂端點會使用僅包含頁面上所指定資料庫執行個體的靜態清單。選取該核取方塊時，自訂端點會使用排除清單。在此情況下，自訂端點會呈現叢集中的所有資料庫執行個體 (包括您未來所新增的任何項目)，在頁面上保持未選取的那些除外。
- 主控台不允許您指定端點類型。使用主控台建立的任何自訂端點都屬於類型 ANY。

  因此，當資料庫執行個體由於容錯移轉或升級而在寫入器與讀取器之間變更角色時，Aurora 不會變更自訂端點的成員資格。

在 AWS CLI 和 Amazon RDS API 中：

- 您可以指定端點類型。因此，當端點類型設為 READER 時，端點成員資格會在容錯移轉和升級期間自動調整。

  例如，類型 READER 的自訂端點包含一個 Aurora 複本，隨後會將其提升為寫入器資料庫執行個體。新的寫入器執行個體不再是自訂端點的一部分。
- 您可以將個別成員新增至清單，並在其變更角色後，將其從清單中移除。使用 modify-db-cluster-endpoint AWS CLI 命令或 ModifyDBClusterEndpoint API 操作。

您可以將一個資料庫執行個體與多個自訂端點建立關聯。例如，假設您將新資料庫執行個體新增至叢集，或是 Aurora 透過自動擴展機制自動新增資料庫執行個體。在這些情況下，資料庫執行個體會新增至符合其資格的所有自訂端點。資料庫執行個體新增至其中的目標端點會根據 READER 或 ANY 的自訂端點類型，以及為每個端點定義的任何靜態或排除清單。例如，如果端點包含資料庫執行個體的靜態清單，新增加的 Aurora 複本不會新增至該端點。相反地，如果端點有排除清單，新增加的 Aurora 複本如果名稱未列在排除清單，且其角色符合自訂端點的類型，即會新增至端點。

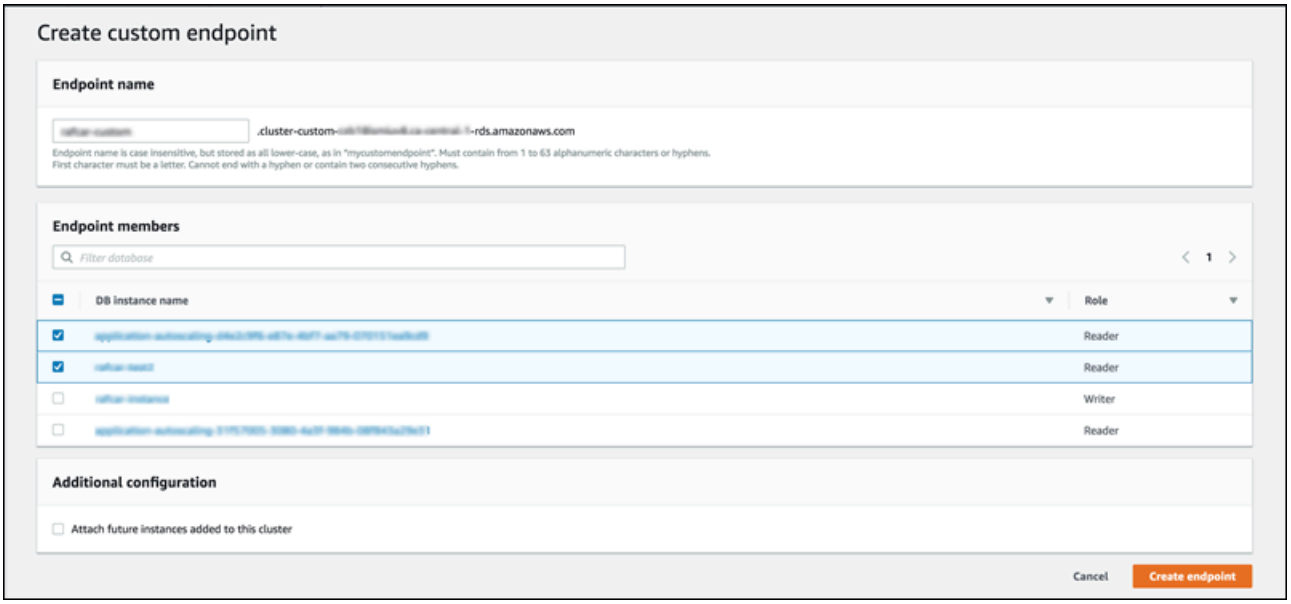
如果 Aurora 複本變得無法使用，則會保持與任何自訂端點相關聯。例如，當它狀況不良、已停止、重新開機等等，它會保留部分自訂端點。不過，除非它再次變得可供使用，否則您無法透過這些端點連線至它。

建立自訂端點

使用 AWS 管理主控台、AWS CLI 或 Amazon RDS API 建立自訂端點。

主控台

若要使用AWS 管理主控台建立自訂端點，請前往叢集詳細資訊頁面，並選擇 Endpoints (端點) 區段中的 `Create custom endpoint` 動作。選擇自訂端點的名稱，該名稱是您的使用者 ID 和區域的唯一。若要選取即使在叢集展開時仍保持相同的資料庫執行個體清單，請將核取方塊 Attach future instances added to this cluster (連線新增至此叢集的未來執行個體) 保持未選取。選擇該核取方塊時，自訂端點會在您將它們新增至叢集時動態新增任何新的執行個體。



您無法在 AWS 管理主控台中選取自訂端點類型。您透過AWS 管理主控台建立的所有自訂端點會具有 ANY 類型。

AWS CLI

若要使用 AWS CLI 建立自訂端點，請執行 create-db-cluster-endpoint 命令。

下列命令會建立連線到特定叢集的自訂端點。一開始，端點會與叢集中的所有 Aurora 複本相關聯。後續命令會將它與叢集中特定的資料庫執行個體集相關聯。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample \
  --endpoint-type reader \
  --db-cluster-identifier cluster_id

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample \
  --static-members instance_name_1 instance_name_2
```

在 Windows 中：

```
aws rds create-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample ^
  --endpoint-type reader ^
  --db-cluster-identifier cluster_id

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample ^
  --static-members instance_name_1 instance_name_2
```

RDS API

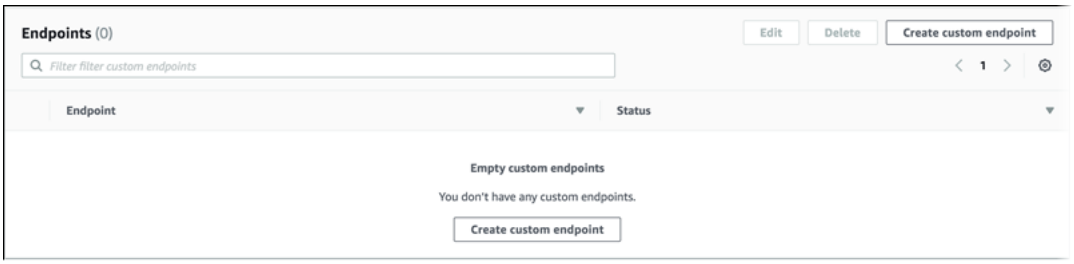
若要使用 RDS API 建立自訂端點，請執行 CreateDBClusterEndpoint 操作。

檢視自訂端點

主控台

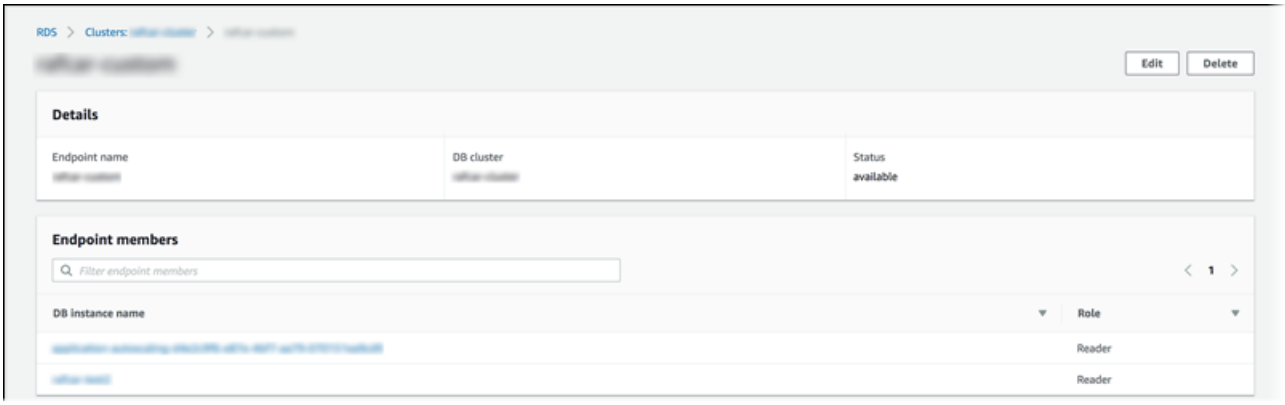
若要使用AWS 管理主控台檢視自訂端點，請前往叢集詳細資訊頁面，並查看 Endpoints (端點) 區段下方。本節僅包含有關自訂端點的資訊。內建端點的詳細資訊會列在主要的 Details (詳細資訊) 中。若要查看特定自訂端點的詳細資訊，請選取其名稱，以帶出該端點的詳細資訊頁面。

以下螢幕擷取畫面顯示最初為空白的 Aurora 叢集的自訂端點的清單。



為該叢集建立一些自訂端點之後，它們會顯示在 Endpoints (端點) 區段下。

按一下詳細資訊頁面會顯示端點目前與其關聯的資料庫執行個體。



若要查看新增至叢集的新資料庫執行個體是否也會自動新增至端點的其他詳細資訊，請開啟端點的 Edit (編輯) 頁面。

AWS CLI

若要使用 AWS CLI 檢視自訂端點，請執行 describe-db-cluster-endpoints 命令。

下列命令會顯示與特定區域中的特定叢集相關聯的自訂端點。輸出會同時包括內建端點和任何自訂端點。

對於 Linux、macOS 或 Unix：

```
aws rds describe-db-cluster-endpoints --region region_name \
  --db-cluster-identifier cluster_id
```

在 Windows 中：

```
aws rds describe-db-cluster-endpoints --region region_name ^
  --db-cluster-identifier cluster_id
```

以下顯示來自 `describe-db-cluster-endpoints` 命令的一些範例輸出。`EndpointType` 或 `WRITER` 的 `READER` 代表叢集的內建讀寫和唯讀端點。`EndpointType` 的 `CUSTOM` 代表您建立並選擇

相關聯資料庫執行個體的端點。其中一個端點有非空白的 StaticMembers 欄位，代表它與精確的一組資料庫執行個體相關聯。另一個端點有非空白的 ExcludedMembers 欄位，代表該端點與所有資料庫執行個體相關聯 (列在 ExcludedMembers 下的除外)。這種第二個類型的自訂端點會成長，以容納您新增至叢集的新執行個體。


```
{
 "DBClusterEndpoints": [
  {
   "Endpoint": "custom-endpoint-demo.cluster-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "Status": "available",
   "DBClusterIdentifier": "custom-endpoint-demo",
   "EndpointType": "WRITER"
  },
  {
   "Endpoint": "custom-endpoint-demo.cluster-ro-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "Status": "available",
   "DBClusterIdentifier": "custom-endpoint-demo",
   "EndpointType": "READER"
  },
  {
   "CustomEndpointType": "ANY",
   "DBClusterEndpointIdentifier": "powers-of-2",
   "ExcludedMembers": [],
   "DBClusterIdentifier": "custom-endpoint-demo",
   "Status": "available",
   "EndpointType": "CUSTOM",
   "Endpoint": "powers-of-2.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "StaticMembers": [
     "custom-endpoint-demo-04",
     "custom-endpoint-demo-08",
     "custom-endpoint-demo-01",
     "custom-endpoint-demo-02"
   ],
   "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
   "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:powers-of-2"
  },
  {
   "CustomEndpointType": "ANY",
```

```
    "DBClusterEndpointIdentifier": "eight-and-higher",
    "ExcludedMembers": [
      "custom-endpoint-demo-04",
      "custom-endpoint-demo-02",
      "custom-endpoint-demo-07",
      "custom-endpoint-demo-05",
      "custom-endpoint-demo-03",
      "custom-endpoint-demo-06",
      "custom-endpoint-demo-01"
    ],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "Status": "available",
    "EndpointType": "CUSTOM",
    "Endpoint": "eight-and-higher.cluster-custom-123456789012.ca-
central-1.rds.amazonaws.com",
    "StaticMembers": [],
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHYQKFU6J6NV5FHU",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:eight-and-higher"
  }
 ]
}
```

RDS API

若要使用 RDS API 檢視自訂端點，請執行 DescribeDBClusterEndpoints.html 操作。

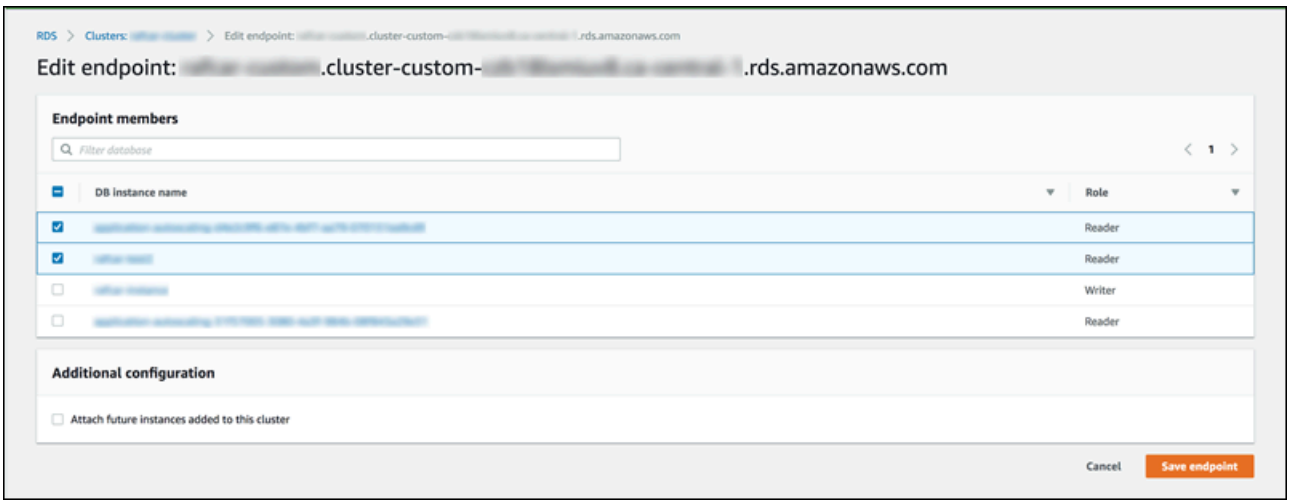
編輯自訂端點

您可以編輯自訂端點的屬性，以變更與該端點相關聯的資料庫執行個體。您也可以變更靜態清單和排除清單之間的端點。如果需要這些端點屬性的詳細資訊，請參閱 自訂端點的成員資格規則 文件。

您可以在編輯動作的變更進行中時繼續連線至或使用自訂端點。

主控台

若要使用AWS 管理主控台編輯自訂端點，您可以在叢集詳細資訊頁面上選取端點，或是帶出端點的詳細資訊頁面，並選擇 Edit (編輯) 動作。



AWS CLI

若要使用 AWS CLI 編輯自訂端點，請執行 modify-db-cluster-endpoint 命令。

下列命令會變更套用至自訂端點的資料庫執行個體集，並選擇性地在靜態或排除清單的行為之間切換。`--static-members` 和 `--excluded-members` 參數會採用以空格分隔的資料庫執行個體識別符清單。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 \
  --static-members db-instance-id-1 db-instance-id-2 db-instance-id-3 \
  --region region_name

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 \
  --excluded-members db-instance-id-4 db-instance-id-5 \
  --region region_name
```

在 Windows 中：

```
aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 ^
  --static-members db-instance-id-1 db-instance-id-2 db-instance-id-3 ^
  --region region_name

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 ^
```

```
  --excluded-members db-instance-id-4 db-instance-id-5 ^
  --region region_name
```

RDS API

若要使用 RDS API 編輯自訂端點，請執行 ModifyDBClusterEndpoint.html 操作。

刪除自訂端點

使用 AWS 管理主控台、AWS CLI 或 Amazon RDS API 刪除自訂端點。

主控台

若要使用AWS 管理主控台刪除自訂端點，請前往叢集詳細資訊頁面，選取適當的自訂端點，並選取 Delete (刪除) 動作。

AWS CLI

若要使用 AWS CLI 刪除自訂端點，請執行 delete-db-cluster-endpoint 命令。

下列命令會刪除自訂端點。您不需要指定相關聯的叢集，但必須指定區域。

對於 Linux、macOS 或 Unix：

```
aws rds delete-db-cluster-endpoint --db-cluster-endpoint-identifier custom-end-point-id
 \
  --region region_name
```

在 Windows 中：

```
aws rds delete-db-cluster-endpoint --db-cluster-endpoint-identifier custom-end-point-id
 ^
  --region region_name
```

RDS API

若要使用 RDS API 刪除自訂端點，請執行 DeleteDBClusterEndpoint 操作。

AWS CLI Amazon Aurora 自訂端點的範例

下列教學課程使用 Unix shell 語法 AWS CLI 的範例，示範如何定義具有數個「小型」資料庫執行個體和數個「大型」資料庫執行個體的叢集，並建立自訂端點以連線至每組資料庫執行個體。若要在自己的系統上執行類似的命令，您應該充分熟悉使用 Aurora 叢集和 AWS CLI 用量的基本概念，為區域、子網路群組和 VPC 安全群組等參數提供自己的值。

此範例示範初始設定步驟：建立 Aurora 叢集和新增資料庫執行個體至叢集。這是異質叢集，表示不是所有資料庫執行個體都有相同的容量。大多數執行個體使用 AWS 執行個體類別 `db.r4.4xlarge`，但最後兩個資料庫執行個體使用 `db.r4.16xlarge`。這些範例 `create-db-instance` 命令中的每個會將其輸出列印至畫面，並將 JSON 的副本儲存在檔案中，供後續檢查。

```
aws rds create-db-cluster --db-cluster-identifier custom-endpoint-demo --engine aurora-
mysql \
     --engine-version 8.0.mysql_aurora.3.04.0 --master-username $MASTER_USER --manage-
master-user-password \
     --db-subnet-group-name $SUBNET_GROUP  --vpc-security-group-ids $VPC_SECURITY_GROUP
 \
     --region $REGION

for i in 01 02 03 04 05 06 07 08
do
  aws rds create-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
     --engine aurora --db-cluster-identifier custom-endpoint-demo --db-instance-class
 db.r4.4xlarge \
     --region $REGION \
     | tee custom-endpoint-demo-${i}.json
done

for i in 09 10
do
  aws rds create-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
     --engine aurora --db-cluster-identifier custom-endpoint-demo --db-instance-class
 db.r4.16xlarge \
     --region $REGION \
     | tee custom-endpoint-demo-${i}.json
done
```

較大的執行個體保留供專業類型的報告查詢使用。為了讓它們不可能被提升為主要執行個體，下列範例會將其提升層變更至最低優先順序。此範例會指定 `--manage-master-user-password` 選項來產生主要使用者密碼，並在 Secrets Manager 中管理該密碼。如需詳細資訊，請參閱使用 Amazon Aurora

和 進行密碼管理 AWS Secrets Manager。或者，您可以使用 --master-password 選項，自行指定和管理密碼。

```
for i in 09 10
do
  aws rds modify-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
    --region $REGION --promotion-tier 15
done
```

假設您只要對最資源密集的查詢使用兩個「較大的」執行個體。若要這樣做，您可以先建立自訂唯讀端點。然後，您可以新增成員的靜態列表，以便端點僅連線到這些資料庫執行個體。這些執行個體已處於最低的提升層級，使它們兩個都不可能被提升為主要執行個體。如果已將它們其中之一提升為主要執行個體，將會變得無法透過此端點連線，因為我們指定的是 READER 類型而非 ANY 類型。

下列範例示範建立和修改端點命令，而範例 JSON 輸出顯示自訂端點的初始和修改的狀態。

```
$ aws rds create-db-cluster-endpoint --region $REGION \
    --db-cluster-identifier custom-endpoint-demo \
    --db-cluster-endpoint-identifier big-instances --endpoint-type reader
{
    "EndpointType": "CUSTOM",
    "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "DBClusterEndpointIdentifier": "big-instances",
    "DBClusterIdentifier": "custom-endpoint-demo",
    "StaticMembers": [],
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
    "ExcludedMembers": [],
    "CustomEndpointType": "READER",
    "Status": "creating",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances"
}

$ aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier big-instances \
    --static-members custom-endpoint-demo-09 custom-endpoint-demo-10 --region $REGION
{
    "EndpointType": "CUSTOM",
    "ExcludedMembers": [],
    "DBClusterEndpointIdentifier": "big-instances",
```

```
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
    "CustomEndpointType": "READER",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances",
    "StaticMembers": [
        "custom-endpoint-demo-10",
        "custom-endpoint-demo-09"
    ],
    "Status": "modifying",
    "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "DBClusterIdentifier": "custom-endpoint-demo"
}
```

叢集的預設 READER 端點可以連線至小型或大型資料庫執行個體，使得當叢集變得忙碌時，要預測查詢效能和可擴展性變得不切實際。若要在資料庫執行個體集之間清楚劃分工作負載，您可以忽略預設的 READER 端點，並建立連線至所有其他資料庫執行個體的第二個自訂端點。以下範例會透過建立自訂端點，然後新增排除清單來執行此動作。您之後新增至叢集的任何資料庫執行個體，將會自動新增至此端點。ANY 類型表示此端點與總共八個執行個體相關聯：一個主要執行個體和另外七個 Aurora 複本。如果範例使用 READER 類型，則自訂端點只會與七個 Aurora 複本相關聯。

```
$ aws rds create-db-cluster-endpoint --region $REGION --db-cluster-identifier custom-
endpoint-demo \
    --db-cluster-endpoint-identifier small-instances --endpoint-type any
{
    "Status": "creating",
    "DBClusterEndpointIdentifier": "small-instances",
    "CustomEndpointType": "ANY",
    "EndpointType": "CUSTOM",
    "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "StaticMembers": [],
    "ExcludedMembers": [],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:small-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY"
}

$ aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier small-instances \
```

```
    --excluded-members custom-endpoint-demo-09 custom-endpoint-demo-10 --region $REGION
{
    "DBClusterEndpointIdentifier": "small-instances",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:c7tj4example:cluster-
endpoint:small-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
    "CustomEndpointType": "ANY",
    "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "EndpointType": "CUSTOM",
    "ExcludedMembers": [
        "custom-endpoint-demo-09",
        "custom-endpoint-demo-10"
    ],
    "StaticMembers": [],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "Status": "modifying"
}
```

以下範例會檢查此叢集的端點狀態。叢集仍有其原始叢集端點，其 EndPointType 為 WRITER，您仍將它用於管理、ETL 和其他寫入操作。它仍會有其原始 READER 端點，但您不會使用它，因為對它的每個連線都可能會被導向至「小型」或「大型」資料庫執行個體。自訂端點讓此行為能被預測，以保證連線使用其中一個「小型」或「大型」資料庫執行個體 (根據您指定的端點)。

```
$ aws rds describe-db-cluster-endpoints --region $REGION
{
    "DBClusterEndpoints": [
        {
            "EndpointType": "WRITER",
            "Endpoint": "custom-endpoint-demo.cluster-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
        },
        {
            "EndpointType": "READER",
            "Endpoint": "custom-endpoint-demo.cluster-ro-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
        },
        {
```

```
            "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "CustomEndpointType": "ANY",
            "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:small-instances",
            "ExcludedMembers": [
                "custom-endpoint-demo-09",
                "custom-endpoint-demo-10"
            ],
            "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
            "DBClusterIdentifier": "custom-endpoint-demo",
            "StaticMembers": [],
            "EndpointType": "CUSTOM",
            "DBClusterEndpointIdentifier": "small-instances",
            "Status": "modifying"
        },
        {
            "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "CustomEndpointType": "READER",
            "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances",
            "ExcludedMembers": [],
            "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
            "DBClusterIdentifier": "custom-endpoint-demo",
            "StaticMembers": [
                "custom-endpoint-demo-10",
                "custom-endpoint-demo-09"
            ],
            "EndpointType": "CUSTOM",
            "DBClusterEndpointIdentifier": "big-instances",
            "Status": "available"
        }
    ]
}
```

最後的範例示範對自訂端點的後續資料庫連線如何連線至 Aurora 叢集中的各種資料庫執行個體。small-instances 端點一律會連線至 db.r4.4xlarge 資料庫執行個體，它是此叢集中編號較低的主機。

```
$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-02 |
+--------------------------+

$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-07 |
+--------------------------+

$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-01 |
+--------------------------+
```

big-instances 端點一律會連線至 db.r4.16xlarge 資料庫執行個體，它是此叢集中編號最高的兩個主機。

```
$ mysql -h big-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -u
 $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-10 |
+--------------------------+

$ mysql -h big-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -u
 $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
```

```
| @@aurora_server_id      |
+-------------------------+
| custom-endpoint-demo-09 |
+-------------------------+
```

Amazon Aurora 資料庫執行個體類別

資料庫執行個體類別會決定 Amazon Aurora 資料庫執行個體的運算和記憶體容量。您需要的資料庫執行個體類別取決於您的處理能力和記憶體需求。

資料庫執行個體類別由資料庫執行個體類別類型和大小組成。例如，db.r6g 是採用 AWS Graviton2 處理器的記憶體最佳化資料庫執行個體類別類型。在 db.r6g 執行個體類別類型內，db.r6g.2xlarge 是資料庫執行個體類別。此類別的大小是 2xlarge。

如需執行個體類別定價的詳細資訊，請參閱 Amazon RDS 定價。

如需資料庫執行個體類別類型、支援的資料庫引擎 AWS 區域、或資料庫執行個體類別的硬體規格的詳細資訊，請參閱下列各節。

主題

- 資料庫執行個體類別的類型
- 資料庫執行個體類別的支援資料庫引擎
- 在 中判斷資料庫執行個體類別支援 AWS 區域
- Aurora 資料庫執行個體類別的硬體規格

資料庫執行個體類別的類型

Amazon Aurora 支援下列使用案例的資料庫執行個體類別：

- Aurora Serverless v2
- 記憶體最佳化
- 爆量效能
- Optimized Reads

如需更多有關 Amazon EC2 執行個體類型的資訊，請參閱 Amazon EC2 文件中的執行個體類型。

Aurora Serverless v2 執行個體類別類型

下列 Aurora Serverless v2 類型是可用的：

- db.serverless – Aurora Serverless v2 使用的特殊資料庫執行個體類別類型。Aurora 會隨著工作負載的變化動態調整運算、記憶體與網路資源。如需用量詳細資料，請參閱 使用 Aurora Serverless v2。

記憶體優化執行個體類別類型

記憶體最佳化 X 系列支援下列執行個體類別：

- db.x2g – 針對記憶體密集型應用程式最佳化的執行個體類別，並採用 AWS Graviton2 處理器。這些執行個體類別為每 GiB 記憶體提供的成本低。

  您可以修改資料庫執行個體，以使用其中一個採用 AWS Graviton2 處理器的資料庫執行個體類別。若要這麼做，請完成與任何其他資料庫執行個體修改相同的步驟。

記憶體最佳化 R 系列支援下列執行個體類別類型：

- db.r8g：採用 AWS Graviton4 處理器的執行個體類別。這些執行個體類別非常適合用來執行記憶體密集型工作負載。這些執行個體提供的執行個體大小比第七代 AWS Graviton3-based db.r7g 執行個體多出高達 3 倍vCPUs 和記憶體。它們採用 AWS Nitro System，結合了專用硬體和輕量型 Hypervisor。
- 您可以修改資料庫執行個體，以使用其中一個採用 AWS Graviton4 處理器的資料庫執行個體類別。若要這麼做，請完成與任何其他資料庫執行個體修改相同的步驟。
- db.r7g – 採用 AWS Graviton3 處理器的執行個體類別。這些執行個體類別非常適合用來執行記憶體密集型工作負載。

  您可以修改資料庫執行個體，以使用其中一個採用 AWS Graviton3 處理器的資料庫執行個體類別。若要這麼做，請完成與任何其他資料庫執行個體修改相同的步驟。它們採用 AWS Nitro System，結合了專用硬體和輕量型 Hypervisor。
- db.r7i：採用第 4 代 Intel Xeon 可擴充處理器的執行個體類別。這些執行個體類別通過 SAP 認證，非常適合用於執行中的記憶體密集型工作負載。您可以修改資料庫執行個體，以使用其中一個採用第 4 代 Intel Xeon 可擴展處理器的資料庫執行個體類別。若要這麼做，請完成與任何其他資料庫執行個體修改相同的步驟。它們採用 AWS Nitro System，結合了專用硬體和輕量型 Hypervisor。
- db.r6g - 採用 AWS Graviton2 處理器的執行個體類別。這些執行個體類別非常適合用來它們採用 AWS Nitro System，結合了專用硬體和輕量型 Hypervisor。
- 您可以修改資料庫執行個體，以使用其中一個採用 AWS Graviton2 處理器的資料庫執行個體類別。若要這麼做，請完成與任何其他資料庫執行個體修改相同的步驟。
- db.r6i - 採用第三代 Intel Xeon 可擴充處理器的執行個體類別。這些執行個體類別通過 SAP 認證，非常適合用於像是 的記憶體密集型工作負載。

- db.r4 – 執行個體類別，已針對記憶體密集型應用程式進行最佳化處理。這些執行個體類別可提供網路和 Amazon Elastic Block Store (Amazon EBS) 效能的改善。它們採用 AWS Nitro System，結合了專用硬體和輕量型 Hypervisor。
- db.r4：Aurora MySQL 2.x 和 Aurora PostgreSQL 11 版和 12 版僅支援這些執行個體類別。針對所有使用 db.r4 資料庫執行個體類別的 Aurora 資料庫叢集，建議您儘快升級到較新一代的執行個體類別。

  db.r4 執行個體類別不適用於 Aurora I/O-Optimized 叢集儲存組態。

爆量效能執行個體類別類型

下列爆量效能資料庫執行個體類別類型是可用的：

- db.t4g – 採用 Arm 型 AWS Graviton2 處理器的一般用途執行個體類別。相較於適用於各種爆量一般用途工作負載的先前爆量效能資料庫執行個體類別，這些執行個體類別提供更好的價格/效能比。Amazon RDS db.t4g 執行個體設為無限制模式。這表示它們可以在 24 小時時段內大幅提升並超越基準，但需額外付費。

  您可以修改資料庫執行個體，以使用其中一個採用 AWS Graviton2 處理器的資料庫執行個體類別。若要這麼做，請完成與任何其他資料庫執行個體修改相同的步驟。
- db.t2 – 此執行個體類別的基準效能具有一定水準，且使用量可爆量增加，以充分利用整個 CPU。db.t3 執行個體設為無限制模式。與先前的 db.t2 執行個體類別相比，這些執行個體類別能提供更多運算容量。它們是採用 AWS Nitro 系統技術，結合了專用硬體和輕量型 Hypervisor。建議您僅將該執行個體類別用在開發、測試伺服器或其他非生產伺服器上。
- db.t2 – 此執行個體類別的基準效能具有一定水準，且使用量可爆量增加，以充分利用整個 CPU。db.t2 執行個體設為無限制模式。建議您僅將該執行個體類別用在開發、測試伺服器或其他非生產伺服器上。

  db.t2 執行個體類別不適用於 Aurora I/O-Optimized 叢集儲存組態。

> **Note**
>
> 建議您僅針對開發伺服器、測試伺服器或其他非生產伺服器，使用 T 資料庫執行個體類別。如需 T 執行個體類別的詳細建議，請參閱 使用 T 執行個體類別進行開發和測試。

如需資料庫執行個體類別硬體規格，請參閱 Aurora 資料庫執行個體類別的硬體規格。

Optimized Reads 執行個體類別類型

下列是可用的 Optimized Reads 執行個體類別類型：

- db.r8gd – 採用 Graviton4 處理器的執行個體類別。這些執行個體類別最適合執行記憶體密集型工作負載，並為需要高速、低延遲本機儲存體的應用程式提供本機 NVMe 型 SSD 區塊層級儲存體。它們提供最大 1.5 TiB 的記憶體，以及高達 11.4 TB 的直接連接 NVMe 型 SSD 儲存體。
- db.r6gd – 採用 AWS Graviton2 處理器的執行個體類別。這些執行個體類別最適合執行記憶體密集型工作負載，並為需要高速、低延遲本機儲存體的應用程式提供本機 NVMe 型 SSD 區塊層級儲存體。
- db.r6id - 採用第三代 Intel Xeon 可擴充處理器的執行個體類別。這些執行個體類別通過 SAP 認證，非常適合用於記憶體密集型工作負載。其提供最高 1TiB 記憶體，以及高達 7.6 TB 的直接連接 NVMe 型 SSD 儲存體。

資料庫執行個體類別的支援資料庫引擎

下表顯示 Amazon Aurora 資料庫引擎的支援資料庫執行個體類別。

db.serverless：具有自動容量擴展的 Aurora Serverless v2 執行個體類別

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.serverless	請參閱 Aurora Serverless v2 支援的區域和 Aurora 資料庫引擎。	請參閱 Aurora Serverless v2 支援的區域和 Aurora 資料庫引擎。

db.x2g – 採用 AWS Graviton2 處理器的記憶體最佳化執行個體類別

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.x2g.16xlarge	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.8 及更新版本、11.9、11.12 和更新版本
db.x2g.12xlarge	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版

執行個體類別	Aurora MySQL	Aurora PostgreSQL
		本、13.3 及更新版本、12.8 及更新版本、11.9、11.12 和更新版本
db.x2g.8xlarge	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.8 及更新版本、11.9、11.12 和更新版本
db.x2g.4xlarge	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.8 及更新版本、11.9、11.12 和更新版本
db.x2g.2xlarge	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.8 及更新版本、11.9、11.12 和更新版本
db.x2g.xlarge	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.8 及更新版本、11.9、11.12 和更新版本
db.x2g.large	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.8 及更新版本、11.9、11.12 和更新版本

db.r6gd – 採用 AWS Graviton2 處理器的 Optimized Reads 執行個體類別

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.r6gd.16xlarge	否	17.4 版及更新版本、16.1 版及更新版本、15.4 版及更新版本、14.9 版及更新版本

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.r6gd.12xlarge	否	17.4 版及更新版本、16.1 版及更新版本、15.4 版及更新版本、14.9 版及更新版本
db.r6gd.8xlarge	否	17.4 版及更新版本、16.1 版及更新版本、15.4 版及更新版本、14.9 版及更新版本
db.r6gd.4xlarge	否	17.4 版及更新版本、16.1 版及更新版本、15.4 版及更新版本、14.9 版及更新版本
db.r6gd.2xlarge	否	17.4 版及更新版本、16.1 版及更新版本、15.4 版及更新版本、14.9 版及更新版本
db.r6gd.xlarge	否	17.4 版及更新版本、16.1 版及更新版本、15.4 版及更新版本、14.9 版及更新版本

db.r6id – Optimized Reads 執行個體類別

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.r6id.32xlarge	否	17.4 版及更新版本、16.1 版及更新版本、15.4 版及更新版本、14.9 版及更新版本
db.r6id.24xlarge	否	17.4 版及更新版本、16.1 版及更新版本、15.4 版及更新版本、14.9 版及更新版本

db.r8g – 採用 AWS Graviton4 處理器的記憶體最佳化執行個體類別

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.r8g.48xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.r8g.24xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本
db.r8g.16xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本
db.r8g.12xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本
db.r8g.8xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本
db.r8g.4xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本
db.r8g.2xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本
db.r8g.xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本
db.r8g.large	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本

db.r8gd – 最佳化讀取採用 AWS Graviton4 處理器的執行個體類別

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.r8gd.48xlarge	否	17.4 及更高版本、16.3 及更高版本、15.7 及更高版本、14.12 及更高版本
db.r8gd.24xlarge	否	17.4 及更高版本、16.3 及更高版本、15.7 及更高版本、14.12 及更高版本
db.r8gd.16xlarge	否	17.4 及更高版本、16.3 及更高版本、15.7 及更高版本、14.12 及更高版本
db.r8gd.12xlarge	否	17.4 及更高版本、16.3 及更高版本、15.7 及更高版本、14.12 及更高版本
db.r8gd.8xlarge	否	17.4 及更高版本、16.3 及更高版本、15.7 及更高版本、14.12 及更高版本
db.r8gd.4xlarge	否	17.4 及更高版本、16.3 及更高版本、15.7 及更高版本、14.12 及更高版本
db.r8gd.2xlarge	否	17.4 及更高版本、16.3 及更高版本、15.7 及更高版本、14.12 及更高版本
db.r8g.xlarge	否	17.4 及更高版本、16.3 及更高版本、15.7 及更高版本、14.12 及更高版本

db.r7g – 採用 AWS Graviton3 處理器的記憶體最佳化執行個體類別

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.r7g.16xlarge	2.12.0 版及更新版本、3.03.1 版及更新版本	17.4 版及更新版本、16.1 版及更新版本、15.2 版及更新版本、14.7 版及更新版本、13.10 版及更新版本
db.r7g.12xlarge	2.12.0 版及更新版本、3.03.1 版及更新版本	17.4 版及更新版本、16.1 版及更新版本、15.2 版及更新版本、14.7 版及更新版本、13.10 版及更新版本
db.r7g.8xlarge	2.12.0 版及更新版本、3.03.1 版及更新版本	17.4 版及更新版本、16.1 版及更新版本、15.2 版及更新版本、14.7 版及更新版本、13.10 版及更新版本
db.r7g.4xlarge	2.12.0 版及更新版本、3.03.1 版及更新版本	17.4 版及更新版本、16.1 版及更新版本、15.2 版及更新版本、14.7 版及更新版本、13.10 版及更新版本
db.r7g.2xlarge	2.12.0 版及更新版本、3.03.1 版及更新版本	17.4 版及更新版本、16.1 版及更新版本、15.2 版及更新版本、14.7 版及更新版本、13.10 版及更新版本
db.r7g.xlarge	2.12.0 版及更新版本、3.03.1 版及更新版本	17.4 版及更新版本、16.1 版及更新版本、15.2 版及更新版本、14.7 版及更新版本、13.10 版及更新版本
db.r7g.large	2.12.0 版及更新版本、3.03.1 版及更新版本	17.4 版及更新版本、16.1 版及更新版本、15.2 版及更新版本、14.7 版及更新版本、13.10 版及更新版本

db.r7i：記憶體最佳化的執行個體類別

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.r7i.48xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本
db.r7i.24xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本
db.r7i.16xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本
db.r7i.12xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本
db.r7i.8xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本
db.r7i.4xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本
db.r7i.2xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本
db.r7i.xlarge	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本
db.r7i.large	3.08.0 及更新版本	17.4 版及更新版本、16.3 版及更新版本、15.7 版及更新版本、14.12 版及更新版本、13.15 版及更新版本

db.r6g – 採用 AWS Graviton2 處理器的記憶體最佳化執行個體類別

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.r6g.16xlarge	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.8 及更新版本、11.9、11.12 和更新版本
db.r6g.12xlarge	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.8 及更新版本、11.9、11.12 和更新版本
db.r6g.8xlarge	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.8 及更新版本、11.9、11.12 和更新版本
db.r6g.4xlarge	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.8 及更新版本、11.9、11.12 和更新版本
db.r6g.2xlarge	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.8 及更新版本、11.9、11.12 和更新版本
db.r6g.xlarge	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.8 及更新版本、11.9、11.12 和更新版本
db.r6g.large	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.8 及更新版本、11.9、11.12 和更新版本

db.r6i – 記憶體優化執行個體類別

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.r6i.32xlarge	目前可用的所有版本	17.4 及更新版本，16.1 及更新版本，15.2 及更新版本，14.3 及更新版本，13.5 及更新版本，和 12.9 及更新版本
db.r6i.24xlarge	目前可用的所有版本	17.4 及更新版本，16.1 及更新版本，15.2 及更新版本，14.3 及更新版本，13.5 及更新版本，和 12.9 及更新版本
db.r6i.16xlarge	目前可用的所有版本	17.4 及更新版本，16.1 及更新版本，15.2 及更新版本，14.3 及更新版本，13.5 及更新版本，和 12.9 及更新版本
db.r6i.12xlarge	目前可用的所有版本	17.4 及更新版本，16.1 及更新版本，15.2 及更新版本，14.3 及更新版本，13.5 及更新版本，和 12.9 及更新版本
db.r6i.8xlarge	目前可用的所有版本	17.4 及更新版本，16.1 及更新版本，15.2 及更新版本，14.3 及更新版本，13.5 及更新版本，和 12.9 及更新版本
db.r6i.4xlarge	目前可用的所有版本	17.4 及更新版本，16.1 及更新版本，15.2 及更新版本，14.3 及更新版本，13.5 及更新版本，和 12.9 及更新版本
db.r6i.2xlarge	目前可用的所有版本	17.4 及更新版本，16.1 及更新版本，15.2 及更新版本，14.3 及更新版本，13.5 及更新版本，和 12.9 及更新版本

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.r6i.xlarge	目前可用的所有版本	17.4 及更新版本，16.1 及更新版本，15.2 及更新版本，14.3 及更新版本，13.5 及更新版本，和 12.9 及更新版本
db.r6i.large	目前可用的所有版本	17.4 及更新版本，16.1 及更新版本，15.2 及更新版本，14.3 及更新版本，13.5 及更新版本，和 12.9 及更新版本

db.r5 – 記憶體優化執行個體類別

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.r5.24xlarge	目前可用的所有版本	目前可用的所有版本
db.r5.16xlarge	目前可用的所有版本	目前可用的所有版本
db.r5.12xlarge	目前可用的所有版本	目前可用的所有版本
db.r5.8xlarge	目前可用的所有版本	目前可用的所有版本
db.r5.4xlarge	目前可用的所有版本	目前可用的所有版本
db.r5.2xlarge	目前可用的所有版本	目前可用的所有版本
db.r5.xlarge	目前可用的所有版本	目前可用的所有版本
db.r5.large	目前可用的所有版本	目前可用的所有版本

db.r4 – 記憶體優化執行個體類別

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.r4.16xlarge	所有 2.x 版；3.x 版本中不支援	所有支援的 11 和 12 版

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.r4.8xlarge	所有 2.x 版；3.x 版本中不支援	所有支援的 11 和 12 版
db.r4.4xlarge	所有 2.x 版；3.x 版本中不支援	所有支援的 11 和 12 版
db.r4.2xlarge	所有 2.x 版；3.x 版本中不支援	所有支援的 11 和 12 版
db.r4.xlarge	所有 2.x 版；3.x 版本中不支援	所有支援的 11 和 12 版
db.r4.large	所有 2.x 版；3.x 版本中不支援	所有支援的 11 和 12 版

db.t4g – 採用 AWS Graviton2 處理器的爆量效能執行個體類別

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.t4g.2xlarge	否	否
db.t4g.xlarge	否	否
db.t4g.large	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.7 及更新版本、11.12 及更新版本
db.t4g.medium	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.7 及更新版本、11.12 及更新版本
db.t4g.small	否	否

db.t3 – 爆量效能執行個體類別

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.t3.2xlarge	否	否

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.t3.xlarge	否	否
db.t3.large	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.7 及更新版本、11.12 及更新版本
db.t3.medium	目前可用的所有版本	17.4 及更新版本、16.1 及更新版本、15.2 及更新版本、14.3 及更新版本、13.3 及更新版本、12.7 及更新版本、11.12 及更新版本
db.t3.small	所有 2.x 版；3.x 版本中不支援	否
db.t3.micro	否	否

db.t2 – 爆量效能執行個體類別

執行個體類別	Aurora MySQL	Aurora PostgreSQL
db.t2.medium	所有 2.x 版；3.x 版本中不支援	否
db.t2.small	所有 2.x 版；3.x 版本中不支援	否

在 中判斷資料庫執行個體類別支援 AWS 區域

若要判定特定 AWS 區域中每個資料庫引擎支援的資料庫執行個體類別，您可以採用數種方法之一。您可以使用 AWS 管理主控台、Amazon RDS 定價頁面或 describe-orderable-db-instance-options AWS CLI 命令。

> **Note**
>
> 當您使用 執行操作時 AWS 管理主控台，它會自動顯示特定資料庫引擎、資料庫引擎版本 和支援的資料庫執行個體類別 AWS 區域。您可以執行的操作範例包括建立和修改資料庫執行個體。

內容

- 使用 Amazon RDS 定價頁面來判斷 中的資料庫執行個體類別支援 AWS 區域
- 使用 AWS CLI 來判斷 中的資料庫執行個體類別支援 AWS 區域
  - 列出 中特定資料庫引擎版本支援的資料庫執行個體類別 AWS 區域
  - 列出支援 中特定資料庫執行個體類別的資料庫引擎版本 AWS 區域

使用 Amazon RDS 定價頁面來判斷 中的資料庫執行個體類別支援 AWS 區域

您可以使用 Amazon Aurora 定價頁面來確定特定 AWS 區域中每個資料庫引擎支援的資料庫執行個體類別。

使用定價頁面確定區域中每個引擎支援的資料庫執行個體類別

1. 前往 Amazon Aurora 定價。
2. 在 AWS 定價計算器區段中選擇 Amazon Aurora 引擎。
3. 在選擇區域中，選擇 AWS 區域。
4. 在叢集組態選項中，選擇組態選項。
5. 您可以使用相容執行個體區段來查看支援的資料庫執行個體類別。
6. (選擇性) 在計算器中選擇其他選項，然後選擇儲存並檢視摘要或儲存並新增服務。

使用 AWS CLI 來判斷 中的資料庫執行個體類別支援 AWS 區域

您可以使用 AWS CLI 來判斷 中特定資料庫引擎和資料庫引擎版本支援哪些資料庫執行個體類別 AWS 區域。

若要使用下列 AWS CLI 範例，請輸入資料庫引擎、資料庫引擎版本、資料庫執行個體類別和的有效值 AWS 區域。下表顯示了有效的資料庫引擎值。

引擎名稱	CLI 命令中的引擎值	如需版本的詳細資訊
MySQL 5.7 相容及 8.0 相容的 Aurora	`aurora-mysql`	Aurora MySQL 版本備註中的 Amazon Aurora MySQL 2 版的資料庫引擎更新和Amazon Aurora MySQL 3 版的資料庫引擎更新
Aurora PostgreSQL	`aurora-postgresql`	Aurora PostgreSQL 版本備註

如需 AWS 區域 名稱的資訊，請參閱 AWS 區域。

下列範例示範如何使用 describe-orderable-db-instance-options AWS CLI 命令 AWS 區域 來判斷 中的資料庫執行個體類別支援。

主題

- 列出 中特定資料庫引擎版本支援的資料庫執行個體類別 AWS 區域
- 列出支援 中特定資料庫執行個體類別的資料庫引擎版本 AWS 區域

列出 中特定資料庫引擎版本支援的資料庫執行個體類別 AWS 區域

若要列出 中特定資料庫引擎版本支援的資料庫執行個體類別 AWS 區域，請執行下列命令。

針對 Linux、macOS 或 Unix：

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version
 \
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region region
```

在 Windows 中：

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version
 ^
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region region
```

輸出也會顯示每個資料庫執行個體類別支援的引擎模式。

例如，下列命令列出美國東部 (維吉尼亞北部) 中 Aurora PostgreSQL 資料庫引擎 13.6 版支援的資料庫執行個體類別。

對於 Linux、macOS 或 Unix：

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --engine-
version 15.3 \
```

```
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region us-east-1
```

在 Windows 中：

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --engine-
version 15.3 ^
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}"  ^
    --output table ^
    --region us-east-1
```

列出支援 中特定資料庫執行個體類別的資料庫引擎版本 AWS 區域

若要列出 AWS 區域中支援特定資料庫執行個體類別的資料庫引擎版本，請執行下列命令。

對於 Linux、macOS 或 Unix：

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-
class DB_instance_class \
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region region
```

在 Windows 中：

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-
class DB_instance_class ^
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region region
```

輸出也會顯示每個資料庫引擎版本支援的引擎模式。

例如，下列命令會列出支援 US East (N. Virginia) 中 db.r5.large 資料庫執行個體類別之 Aurora PostgreSQL 資料庫引擎的資料庫引擎版本。

對於 Linux、macOS 或 Unix：

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.r7g.large \
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region us-east-1
```

在 Windows 中：

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.r7g.large ^
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region us-east-1
```

Aurora 資料庫執行個體類別的硬體規格

您可以在本區段的資料表中，找到有關 Aurora 的 Amazon RDS 資料庫執行個體類別的硬體詳細資訊。

如需各個資料庫執行個體類別的 Aurora 資料庫引擎支援的相關資訊，請參閱 資料庫執行個體類別的支援資料庫引擎。

主題

- Aurora 資料庫執行個體類別的硬體術語
- 記憶體最佳化的執行個體類別的硬體規格
- 爆量效能執行個體類別的硬體規格

Aurora 資料庫執行個體類別的硬體術語

下列術語用於敘述資料庫執行個體類別的硬體規格：

vCPU

虛擬中央處理單元 (CPU) 的數量。虛擬 CPU 即為容量單位，可用來比較資料庫執行個體類別。您不再購買或租用特定的處理器並使用數月或數年，而是以小時為單位租用容量。我們的目標是在實際基礎硬體的限制範圍內，提供一致且明確的 CPU 容量。

ECU

Amazon EC2 執行個體整數處理能力的相對測量單位。為了讓開發人員能輕鬆地比較不同執行個體類別的 CPU 容量，我們定義了 Amazon EC2 運算單位。分配給特定執行個體的 CPU 量以這些 EC2 運算單位來表示。目前，一個 ECU 所提供的 CPU 容量等同於 1.0–1.2 GHz 的 2007 Opteron 或 2007 Xeon 處理器。

記憶體 (GiB)

分配給資料庫執行個體的 RAM，以 GiB 為單位。記憶體和 vCPU 之間的比率通常是固定的。舉 db.r4 執行個體類別為例，其記憶體和 vCPU 的比例類似 db.r5 執行個體類別。然而，大多時候 db.r5 執行個體的效能比 db.r4 執行個體更好、更一致。

最大 EBS 頻寬 (Mbps)

每秒的最高 EBS 頻寬，以百萬位元為單位。若將頻寬除以 8，即可取得預期的傳輸量 (MB/秒)。

> **Note**
>
> 此圖是指資料庫執行個體內本機儲存裝置的輸入/輸出頻寬。它不適用於與 Aurora 叢集磁碟區通訊。

網路頻寬

相對於其他資料庫執行個體類別的網路速度。

如需使用 Amazon CloudWatch 指標監控 Aurora 資料庫執行個體輸送量的資訊，請參閱使用 Amazon CloudWatch 指標評估 Aurora MySQL 的資料庫執行個體用量和使用 CloudWatch 指標評估 Aurora PostgreSQL 的資料庫執行個體用量。

記憶體最佳化的執行個體類別的硬體規格

下表顯示記憶體最佳化的執行個體類別的運算、記憶體、儲存和頻寬規格。

db.x2g – 使用 AWS Graviton2 處理器的記憶體最佳化執行個體類別

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.x2g.16xlarge	64	—	1024	EBS 最佳化專用	19,000	25
db.x2g.12xlarge	48	—	768	EBS 最佳化專用	14,250	20
db.x2g.8xlarge	32	—	512	EBS 最佳化專用	9,500	12
db.x2g.4xlarge	16	—	256	EBS 最佳化專用	4,750	最高 10
db.x2g.2xlarge	8	—	128	EBS 最佳化專用	最高 4,750	最高 10
db.x2g.xlarge	4	—	64	EBS 最佳化專用	最高 4,750	最高 10
db.x2g.large	2	—	32	EBS 最佳化專用	最高 4,750	最高 10

db.r8gd – 採用 AWS Graviton4 處理器和 SSD 儲存體技術的記憶體最佳化執行個體類別

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r8gd.48xlarge	192	—	1536	6 x 1900 NVMe SSD	40,000	50
db.r8gd.24xlarge	96	—	768	3 x 1900 NVMe SSD	30,000	40
db.r8gd.16xlarge	64	—	512	2 x 1900 NVMe SSD	20,000	30

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r8gd.12xlarge	48	—	384	3 x 950 NVMe SSD	15,000	22.5
db.r8gd.8xlarge	32	—	256	1 x 1900 NVMe SSD	10,000	15
db.r8gd.4xlarge	16	—	128	1 x 950 NVMe SSD	最高 10,000	最高 15
db.r8gd.2xlarge	8	—	64	1 x 474 NVMe SSD	最高 10,000	最高 15
db.r8gd.xlarge	4	—	32	1 x 237 NVMe SSD	最高 10,000	最高 12.5
db.r8gd.large	2	—	16	1 x 118 NVMe SSD	最高 10,000	最高 12.5

db.r8g – 採用 AWS Graviton4 處理器的記憶體最佳化執行個體類別

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r8g.48xlarge	192	—	1536	EBS 最佳化專用	40,000	50
db.r8g.24xlarge	96	—	768	EBS 最佳化專用	30,000	40
db.r8g.16xlarge	64	—	512	EBS 最佳化專用	20,000	30
db.r8g.12xlarge	48	—	384	EBS 最佳化專用	15,000	22.5

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r8g.8xlarge	32	—	256	EBS 最佳化專用	10,000	15
db.r8g.4xlarge	16	—	128	EBS 最佳化專用	最高 10,000	最高 15
db.r8g.2xlarge	8	—	64	EBS 最佳化專用	最高 10,000	最高 15
db.r8g.xlarge	4	—	32	EBS 最佳化專用	最高 10,000	最高 12.5
db.r8g.large	2	—	16	EBS 最佳化專用	最高 10,000	最高 12.5

db.r7i：採用第 4 代 Intel Xeon 可擴充處理器的記憶體最佳化的執行個體類別

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r7i.48xlarge	192	—	1536	EBS 最佳化專用	40,000	50
db.r7i.24xlarge	96	—	768	EBS 最佳化專用	30,000	37.5
db.r7i.16xlarge	64	—	512	EBS 最佳化專用	20,000	25
db.r7i.12xlarge	48	—	384	EBS 最佳化專用	15,000	18.75
db.r7i.8xlarge	32	—	256	EBS 最佳化專用	10,000	12.5

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r7i.4xlarge	16	—	128	EBS 最佳化專用	最高 10,000	最高 12.5
db.r7i.2xlarge	8	—	64	EBS 最佳化專用	最高 10,000	最高 12.5
db.r7i.xlarge	4	—	32	EBS 最佳化專用	最高 10,000	最高 12.5
db.r7i.large	2	—	16	EBS 最佳化專用	最高 10,000	最高 12.5

db.r7g – 使用 AWS Graviton3 處理器的記憶體最佳化執行個體類別

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r7g.16xlarge	64	—	512	EBS 最佳化專用	20,000	30
db.r7g.12xlarge	48	—	384	EBS 最佳化專用	15,000	22.5
db.r7g.8xlarge	32	—	256	EBS 最佳化專用	10,000	15
db.r7g.4xlarge	16	—	128	EBS 最佳化專用	最高 10,000	最高 15
db.r7g.2xlarge	8	—	64	EBS 最佳化專用	最高 10,000	最高 15
db.r7g.xlarge	4	—	32	EBS 最佳化專用	最高 10,000	最高 12.5

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r7g.large	2	—	16	EBS 最佳化專用	最高 10,000	最高 12.5

db.r6id：採用第 3 代 Intel Xeon 可擴充處理器和 SSD 儲存體的記憶體最佳化的執行個體類別

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r6id.32xlarge	128	—	1,024	4x1900 NVMe SSD	40,000	50
db.r6id.24xlarge	96	—	768	4x1425 NVMe SSD	30,000	37.5

db.r6gd – 具有 AWS Graviton2 處理器和 SSD 儲存體的記憶體最佳化執行個體類別

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r6gd.16xlarge	64	—	512	2 x 1900 NVMe SSD	19,000	25
db.r6gd.12xlarge	48	—	384	2 x 1425 NVMe SSD	13,500	20
db.r6gd.8xlarge	32	—	256	1 x 1900 NVMe SSD	9,000	12
db.r6gd.4xlarge	16	—	128	1 x 950 NVMe SSD	4,750	最高 10
db.r6gd.2xlarge	8	—	64	1 x 474 NVMe SSD	最高 4,750	最高 10

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r6gd.xlarge	4	—	32	1 x 237 NVMe SSD	最高 4,750	最高 10

db.r6g – 使用 AWS Graviton2 處理器的記憶體最佳化執行個體類別

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r6g.16xlarge	64	—	512	EBS 最佳化專用	19,000	25
db.r6g.12xlarge	48	—	384	EBS 最佳化專用	13,500	20
db.r6g.8xlarge	32	—	256	EBS 最佳化專用	9,000	12
db.r6g.4xlarge	16	—	128	EBS 最佳化專用	4,750	最高 10
db.r6g.2xlarge	8	—	64	EBS 最佳化專用	最高 4,750	最高 10
db.r6g.xlarge	4	—	32	EBS 最佳化專用	最高 4,750	最高 10
db.r6g.large	2	—	16	EBS 最佳化專用	最高 4,750	最高 10

db.r6i - 採用第三代 Intel Xeon 可擴充處理器的記憶體優化執行個體類別

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r6i.32xlarge	128	—	1,024	EBS 最佳化專用	40,000	50
db.r6i.24xlarge	96	—	768	EBS 最佳化專用	30,000	37.5
db.r6i.16xlarge	64	—	512	EBS 最佳化專用	20,000	25
db.r6i.12xlarge	48	—	384	EBS 最佳化專用	15,000	18.75
db.r6i.8xlarge	32	—	256	EBS 最佳化專用	10,000	12.5
db.r6i.4xlarge	16	—	128	EBS 最佳化專用	最高 10,000	最高 12.5
db.r6i.2xlarge	8	—	64	EBS 最佳化專用	最高 10,000	最高 12.5
db.r6i.xlarge	4	—	32	EBS 最佳化專用	最高 10,000	最高 12.5
db.r6i.large	2	—	16	EBS 最佳化專用	最高 10,000	最高 12.5

db.r5 – 記憶體優化執行個體類別

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r5.24xlarge	96	347	768	EBS 最佳化專用	19,000	25

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r5.16xlarge	64	264	512	EBS 最佳化專用	13,600	20
db.r5.12xlarge	48	173	384	EBS 最佳化專用	9,500	12
db.r5.8xlarge	32	132	256	EBS 最佳化專用	6,800	10
db.r5.4xlarge	16	71	128	EBS 最佳化專用	4,750	最高 10
db.r5.2xlarge	8	38	64	EBS 最佳化專用	最高 4,750	最高 10
db.r5.xlarge	4	19	32	EBS 最佳化專用	最高 4,750	最高 10
db.r5.large	2	10	16	EBS 最佳化專用	最高 4,750	最高 10

db.r4 - 採用 Intel Xeon 可擴充處理器的記憶體優化執行個體類別

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r4.16xlarge	64	195	488	EBS 最佳化專用	14,000	25
db.r4.8xlarge	32	99	244	EBS 最佳化專用	7,000	10
db.r4.4xlarge	16	53	122	EBS 最佳化專用	3,500	最高 10

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.r4.2xlarge	8	27	61	EBS 最佳化專用	1,700	最高 10
db.r4.xlarge	4	13.5	30.5	EBS 最佳化專用	850	最高 10
db.r4.large	2	7	15.25	EBS 最佳化專用	425	最高 10

爆量效能執行個體類別的硬體規格

下表顯示爆量效能執行個體類別的運算、記憶體、儲存和頻寬規格。

db.t4g – 採用 AWS Graviton2 處理器的爆量效能執行個體類別

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.t4g.large	2	—	8	EBS 最佳化專用	最高 2,780	最高 5
db.t4g.medium	2	—	4	EBS 最佳化專用	最高 2,085	最高 5

db.t3 – 爆量效能執行個體類別

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.t3.large	2	變數	8	EBS 最佳化專用	最高 2,048	最高 5
db.t3.medium	2	變數	4	EBS 最佳化專用	最高 1,536	最高 5

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.t3.small	2	變數	2	EBS 最佳化專用	最高 1,536	最高 5

db.t2 – 爆量效能執行個體類別

執行個體類別	vCPU	ECU	記憶體 (GiB)	執行個體儲存體 (GiB)	最大 EBS 頻寬 (Mbps)	網路頻寬 (Gbps)
db.t2.medium	2	變數	4	僅 EBS	—	適中
db.t2.small	1	變數	2	僅 EBS	—	低

Amazon Aurora 儲存體

接下來，您可以了解 Aurora 儲存子系統。Aurora 使用分散式和共用儲存體架構，該架構為 Aurora 叢集的效能、可擴展性和可靠性方面的重要因素。

主題

- Amazon Aurora 儲存體的概觀
- 叢集磁碟區包含的內容
- Amazon Aurora 資料庫叢集的儲存組態
- Aurora 儲存體如何自動調整大小
- Aurora 資料儲存體的計費方式

Amazon Aurora 儲存體的概觀

Aurora 資料會存放在叢集磁碟區，該磁碟區是單一虛擬、採用固態磁碟機 (SSD) 的磁碟區。叢集磁碟區包含單一 AWS 區域中三個可用區域的資料複本。因為資料會自動跨可用性區域複寫，您的資料高度耐用且資料遺失的可能性較低。此複寫也可確保您的資料庫在容錯移轉期間更可用。這麼做是因為資料複本已存在於其他可用性區域，並繼續對您的資料庫叢集中的執行個體提供資料請求。複寫數量與您叢集中的資料庫執行個體數量無關。

Aurora 對非持久性暫存檔案會使用個別的本機儲存空間。這包括用於在查詢處理與建置索引期間排序大型資料集等用途的檔案。如需詳細資訊，請參閱Aurora MySQL 的暫存空間限制及Aurora PostgreSQL 的暫存空間限制。

叢集磁碟區包含的內容

Aurora 叢集磁碟區包含您的所有使用者資料、結構描述物件和內部中繼資料，例如系統資料表和二進位日誌。例如，Aurora 會為叢集磁碟區中的 Aurora 叢集存放所有資料表、索引、二進位大型物件 (BLOB)、預存程序等等。

Aurora 共用儲存體架構讓您的資料可獨立於叢集中的資料庫執行個體。例如，您可以快速新增資料庫執行個體，因為 Aurora 不會對資料表資料製作新的複本。相反地，資料庫執行個體會連線至已包含您的所有資料的共用儲存區。您可以從叢集移除資料庫執行個體，而不需從叢集移除任何基礎資料。只有在您刪除整個叢集時，Aurora 才會移除資料。

Amazon Aurora 資料庫叢集的儲存組態

Amazon Aurora 具有兩個資料庫叢集儲存組態：

- Aurora I/O-Optimized - 已改善 I/O 密集型應用程式的價格效能和可預測性。您只需為資料庫叢集的用量和儲存付費，而讀取和寫入 I/O 操作無須額外付費。

  當您的 I/O 支出達到 Aurora 資料庫總支出的 25% 以上時，Aurora I/O-Optimized 就是最佳選擇。

  當建立或修改其資料庫引擎版本支援 Aurora I/O-Optimized 叢集組態的資料庫叢集時，您可以選擇 Aurora I/O-Optimized。您可以隨時從 Aurora I/O-Optimized 切換到 Aurora Standard。
- Aurora Standard - 經濟實惠的定價，適合於 I/O 用量適中的許多應用程式。除了資料庫叢集的用量和儲存之外，您還會為 I/O 操作支付每 100 萬個請求的標準費率。

  當您的 I/O 支出少於 Aurora 資料庫總支出的 25% 時，Aurora Standard 就是最佳選擇。

  您可以每隔 30 天從 Aurora Standard 切換至 Aurora I/O-Optimized。當您在non-NVMe-based執行個體的 Aurora Standard 和 Aurora I/O-Optimized 儲存選項之間切換時，沒有停機時間。不過，對於 NVMe 型資料庫執行個體，在 Aurora I/O-Optimized 和 Aurora Standard 儲存選項之間切換需要重新啟動資料庫引擎，這可能會導致短暫的停機時間。

如需 AWS 區域 和版本支援的詳細資訊，請參閱 叢集儲存體組態的支援區域和 Aurora 資料庫引擎。

如需 Amazon Aurora 儲存組態的定價詳細資訊，請參閱 Amazon Aurora 定價。

如需在建立資料庫叢集時選擇儲存體組態的相關資訊，請參閱 建立資料庫叢集。如需資料庫叢集修改儲存體組態的相關資訊，請參閱 Amazon Aurora的設定。

Aurora 儲存體如何自動調整大小

Aurora 叢集磁碟區會隨著您的資料庫中的資料數量增加自動成長。如需每個引擎版本 Aurora 叢集磁碟區大小上限的資訊，請參閱 Amazon Aurora 大小限制。這種自動化儲存擴展結合了高效能和高分散式儲存子系統。當您的主要目標是可靠性和高可用性時，這使得 Aurora 對於您重要的企業資料而言是不錯的選擇。

若要顯示磁碟區狀態，請參閱顯示 Aurora MySQL 資料庫叢集的磁碟區狀態或顯示 Aurora PostgreSQL 資料庫叢集的磁碟區狀態。如需平衡儲存成本與其他優先順序的方法，儲存體擴展說明了如何監控 CloudWatch 中的 Amazon Aurora 指標 `AuroraVolumeBytesLeftTotal` 和 `VolumeBytesUsed`。

移除 Aurora 資料時，會釋放為該資料配置的空間。移除資料的範例包括捨棄或截斷資料表。這種自動減少儲存用量，可協助您將儲存費用降至最低。

> **Note**
>
> 此處討論的儲存限制和動態調整大小行為適用於儲存在叢集磁碟區中的永久性資料表和其他資料。
> 若為 Aurora PostgreSQL，暫存資料表中的資料會存放在本機資料庫執行個體中。
> 對於 Aurora MySQL 第 2 版，暫存資料表資料預設會存放在寫入器執行個體的叢集磁碟區中，以及讀取器執行個體的本機儲存體中。如需詳細資訊，請參閱磁碟上暫存資料表的儲存引擎。
> 對於 Aurora MySQL 第 3 版，暫存資料表資料會存放在本機資料庫執行個體或叢集磁碟區中。如需詳細資訊，請參閱Aurora MySQL 第 3 版的新暫時資料表行為。
> 位於本機儲存體的暫存資料表的大小上限受制於資料庫執行個體的本機儲存體大小上限。本機儲存體大小取決於您使用的執行個體類別。如需詳細資訊，請參閱Aurora MySQL 的暫存空間限制及Aurora PostgreSQL 的暫存空間限制。

某些儲存功能 (例如叢集磁碟區的大小上限，以及移除資料時自動調整大小) 取決於叢集的 Aurora 版本。如需詳細資訊，請參閱儲存體擴展。您也可以了解如何避免儲存體問題，以及如何監控叢集中配置的儲存空間和可用空間。

Aurora 資料儲存體的計費方式

即使 Aurora 叢集磁碟區對於特定引擎版本最多可增長至 256 tebibytes (TiB)，系統只會針對您在 Aurora 叢集磁碟區中使用的空間收費。在較早的 Aurora 版本中，叢集磁碟區可以重複使用移除資料時釋放的空間，但配置的儲存空間永遠不會減少。現在，當透過捨棄資料表或資料庫等來移除 Aurora 時，整體配置的空間會減少相當大的數量。因此，您可以捨棄不再需要的資料表、索引、資料庫等，以減少儲存費用。

> **Tip**
>
> 對於沒有動態調整大小功能的較早版本，重設叢集的儲存使用量，涉及執行邏輯傾印並還原至新叢集。如果資料量很大，這項操作可能需要很長的時間。如果您遇到這種情況，請考慮將叢集升級至支援動態磁碟區大小調整的版本。

如需哪些 Aurora 版本支援動態大小調整，以及如何透過監控叢集的儲存用量，將儲存費用降至最低的相關資訊，請參閱 儲存體擴展。如需 Aurora 備份儲存體計費的相關資訊，請參閱 了解 Amazon Aurora 備份儲存體用量。如需 Aurora 資料儲存體的定價資訊，請參閱 Amazon RDS for Aurora 定價。

Amazon Aurora 可靠性

Aurora 的設計目的是要既可靠、耐用且容錯。您可以架構您的 Aurora 資料庫叢集，以透過執行一些動作 (例如新增 Aurora 複本並將其放置在不同的可用區域) 來改善可用性，而 Aurora 也包括數個自動功能，因此是可靠的資料庫解決方案。

主題

- 儲存體自動修復
- 可存活的頁面快取
- 從非計劃的重新啟動中復原

儲存體自動修復

因為 Aurora 會在三個可用區域內維持您資料的多個副本，所以，因磁碟故障導致資料遺失的機率會大幅降低。Aurora 會自動偵測在組成叢集磁碟區的磁碟區中所發生的故障。磁碟區的區段失敗時，Aurora 會立即修復該區段。當 Aurora 修復磁碟區段時，它會使用組成叢集磁碟區的其他磁碟區中

資料，以確保中修復的區段的資料是最新的。因此，Aurora 可避免資料遺失並減少執行時間點還原以從磁碟失敗復原的需求。

可存活的頁面快取

在 Aurora 中，每個資料庫執行個體的頁面快取是以與資料庫不同的程序管理，這可讓頁面快取獨立於資料庫存活。(頁面快取在 Aurora MySQL 上也稱為 InnoDB 緩衝集區，在 Aurora PostgreSQL 上則稱為緩衝快取。)

萬一資料庫發生故障，頁面快取仍會保留在記憶體中，這會在資料庫重新啟動時讓目前資料頁面在頁面快取中保持為「暖」的狀態。這會避開需要初始查詢來執行讀取 I/O 操作讓頁面快取「暖機」，進而提供效能增益。

對於 Aurora MySQL，當重新啟動和容錯移轉時的頁面快取行為如下：

- 您可以重新啟動寫入器執行個體，無需重新啟動讀取器執行個體。
  - 如果讀取器執行個體在寫入器執行個體重新啟動時未重新啟動，並不會失去其頁面快取。
  - 如果讀取器執行個體在寫入器執行個體重新啟動時重新啟動，則確實會失去其頁面快取。
- 當讀取器執行個體重新啟動時，寫入器和讀取器執行個體上的頁面快取都會存在。
- 當資料庫叢集容錯移轉時，效果與寫入器執行個體重新啟動時的效果類似。在新的寫入器執行個體 (先前是讀取器執行個體) 上，頁面快取仍會存在，但在讀取器執行個體 (先前是寫入器執行個體) 上，頁面存取不會存在。

對於 Aurora PostgreSQL，您可以使用叢集快取管理來保留在容錯移轉後，成為寫入器執行個體的所指定讀取器執行個體其頁面快取。如需更多詳細資訊，請參閱 Aurora PostgreSQL 的容錯移轉後使用叢集快取管理快速復原。

從非計劃的重新啟動中復原

Aurora 旨在幾乎立即從非計劃的重新啟動中復原，並在沒有二進位日誌的情況下，繼續為應用程式資料提供服務。Aurora 會在平行執行緒上以非同步的方式復原，以便資料庫在非計劃的重新啟動之後立即開啟並可用。

如需詳細資訊，請參閱Aurora 資料庫叢集的容錯能力及最佳化以減少資料庫重新啟動時間。

下列是 Aurora MySQL 上二進位日誌和非計劃重新啟動復原的考量：

- 在 Aurora 上啟用二進位日誌會直接影響非計劃重新啟動之後的復原時間，因為它會強迫資料庫執行個體執行二進位日誌復原。

- 使用的二進位日誌類型會影響日誌的大小和效率。針對相同數量的資料庫活動，有些格式會記錄較其他二進位日誌更多的資訊。下列 `binlog_format` 參數的設定會造成不同數量的日誌資料：
  - `ROW` – 最多日誌資料
  - `STATEMENT` – 最少日誌資料
  - `MIXED` – 中等數量的日誌資料，通常可提供資料完整性和效能的最佳組合

  二進位日誌資料的數量會影響復原時間。如果二進位日誌中記錄了較多資料，資料庫執行個體必須在復原期間處理更多資料，而這會增加復原時間。
- 若要透過二進位記錄減少運算額外負荷並改善復原時間，您可以使用增強型 binlog。增強型 binlog 可改善資料庫復原時間高達 99%。如需更多詳細資訊，請參閱 設定 Aurora MySQL 的增強型 binlog。
- Aurora 不需要二進位日誌，即可在資料庫叢集內複寫資料或是執行時間點還原 (PITR)。
- 如果您不需要二進位日誌用於進行外部複寫 (或外部二進位日誌資料流)，建議您將 `binlog_format` 參數設定為 `OFF` 來停用二進位日誌。這麼做可減少復原時間。

如需 Aurora 二進位日誌和複寫的詳細資訊，請參閱 以 Amazon Aurora 進行複寫。如需不同的 MySQL 複寫類型隱含意義的詳細資訊，請參閱 MySQL 文件中的陳述式和資料列式複寫的優缺點。

Amazon Aurora 安全性

Amazon Aurora 的安全性是以三個層級來管理：

- 若要控制可在 Aurora 資料庫叢集和資料庫執行個體上執行 Amazon RDS 管理動作的人員，您可以使用 AWS Identity and Access Management (IAM)。當您使用 IAM 登入資料連線至 AWS 時，AWS 帳戶必須具備能授予所需許可的 IAM 政策，才能執行 Amazon RDS 管理操作。如需更多詳細資訊，請參閱 Amazon Aurora 的 Identity and access management。

  如果您是使用 IAM 來存取 Amazon RDS 主控台，則首先必須使用您的使用者登入資料來登入 AWS 管理主控台，然後前往 Amazon RDS 主控台：https://console.aws.amazon.com/rds。
- Aurora 資料庫叢集必須在以 Amazon VPC 服務為基礎的 Virtual Private Cloud (VPC) 中建立。若要控制哪些裝置和 Amazon EC2 執行個體可以開放對 VPC 中 Aurora 資料庫叢集的資料庫執行個體端點和連接埠的連線，您可以使用 VPC 安全群組。您可以使用 Transport Layer Security (TLS)/Secure Sockets Layer (SSL) 進行這些端點和連接埠連線。此外，貴公司的防火牆規則可控管在公司內執行的裝置是否可開啟與資料庫執行個體的連線。如需 VPC 的詳細資訊，請參閱Amazon VPC 和 Amazon Aurora。

- 若要驗證 Amazon Aurora 資料庫叢集的登入資訊與許可，您可採取下列任一方式，或搭配使用多種方法。
  - 您可採用與 MySQL 或 PostgreSQL 獨立資料庫執行個體相同的驗證方式。

    用於驗證 MySQL 或 PostgreSQL 獨立資料庫執行個體登入和許可的技術，例如使用 SQL 命令或修改資料庫結構描述資料表，也適用 Aurora。如需詳細資訊，請參閱「Amazon Aurora MySQL 的安全性」或「Amazon Aurora PostgreSQL 的安全性」。
  - 您可以使用 IAM 資料庫身分驗證。

    透過 IAM 資料庫身分驗證，您可以使用使用者或 IAM 角色以及身分驗證字符，向您的 Aurora 資料庫叢集進行身分驗證。身分驗證字符是不重複的值，由 Signature 第 4 版簽署程序所產生。使用 IAM 資料庫身分驗證，便可利用相同的登入資料控管您 AWS 資源與資料庫的存取情形。如需更多詳細資訊，請參閱 IAM 資料庫身分驗證。
  - 您可以對 Aurora PostgreSQL 和 Aurora MySQL 使用 Kerberos 身分驗證。

    在使用者連線至您的 Aurora PostgreSQL 和 Aurora MySQL 資料庫叢集時，您可以使用 Kerberos 驗證使用者身分。在此情況下，您的資料庫叢集會使用 AWS Directory Service for Microsoft Active Directory 來啟用 Kerberos 身分驗證。AWS Directory Service for Microsoft Active Directory 也稱為 AWS Managed Microsoft AD。將您的所有登入資料保留在相同目錄可以節省您的時間和精力。這樣您就有一個集中的位置來存放及管理多個資料庫叢集的登入資料。使用目錄也可以改善您的整體安全性描述檔。如需詳細資訊，請參閱搭配 Aurora PostgreSQL 使用 Kerberos 身分驗證及針對 Aurora MySQL 使用 Kerberos 身分驗證。

如需設定安全性的相關資訊，請參閱 Amazon Aurora 的安全性。

將 SSL 與 Aurora 資料庫叢集搭配使用

如果應用程式使用與 Amazon RDS 資料庫執行個體相同的程序和公有金鑰，Amazon Aurora 資料庫叢集支援來自這些應用程式的 Secure Sockets Layer (SSL) 連線。如需詳細資訊，請參閱「Amazon Aurora MySQL 的安全性」、「Amazon Aurora PostgreSQL 的安全性」或「搭配 Aurora Serverless v1 使用 TLS/SSL」。

Amazon Aurora 的高可用性

Amazon Aurora 架構涉及儲存和運算的分離。Aurora 包含部分高可用性功能，這些功能可套用至資料庫叢集中的資料。即使叢集內部分或全部資料庫執行個體變得無法使用時，資料仍然維持安全性。其他

高可用性功能套用至資料庫執行個體。這些功能可確保一或多個資料庫執行個體就緒，以協助處理您應用程式的資料庫請求。

主題

- Aurora 資料的高可用性
- Aurora 資料庫執行個體的高可用性
- 使用 Aurora 全域資料庫跨 AWS 區域提供高可用性
- Aurora 資料庫叢集的容錯能力
- 與 Amazon RDS Proxy 的高可用性

Aurora 資料的高可用性

Aurora 將資料副本存放在單一 AWS 區域跨多個可用區域的資料庫叢集中。無論資料庫叢集的執行個體是否跨多個可用區域，Aurora 都會存放這些副本。如需 Aurora 的詳細資訊，請參閱管理 Amazon Aurora 資料庫叢集。

當資料寫入主要資料庫執行個體時，Aurora 同步複寫到跨可用區域，至叢集磁碟區中相關連的六個儲存節點。如此一來可提供資料備援、排除 I/O 凍結，並降低系統備份時的延遲峰值。執行具有高可用性的資料庫執行個體，可以在規劃好的系統維護期間增強可用性，並有助於在失敗和可用區域中斷時保護資料庫。如需可用區域的詳細資訊，請參閱區域和可用區域。

Aurora 資料庫執行個體的高可用性

建立主要 (寫入器) 執行個體後，您就能建立最多 15 個唯讀 Aurora 複本。Aurora 複本也稱為讀取器執行個體。Aurora 複本使用非同步複寫來支援高可用性，而不會影響主要執行個體效能。

在日常作業期間，您可以使用讀取器執行個體來處理 SELECT 查詢，卸載讀取密集應用程式的部分工作。當問題影響主要執行個體時，其中一個讀取器執行個體會接管而成為主要執行個體。這種機制稱為容錯移轉。許多 Aurora 功能適用於容錯移轉機制。例如，Aurora 偵測資料庫問題，並在必要時自動啟用容錯移轉機制。Aurora 也具有可縮短容錯移轉完成所需時間的功能。這樣做可將資料庫在容錯移轉期間無法寫入的時間降到最低。

Aurora 旨在盡快復原，最快的復原路徑通常是重新啟動或容錯移轉至相同的資料庫執行個體。重新啟動比容錯移轉更快，且涉及的負荷更少。

若要使用即使容錯移轉提升新的主要執行個體時仍保持不變的連線字串，請連線到叢集端點。叢集端點一律代表叢集中目前的主要執行個體。如需叢集端點的詳細資訊，請參閱 Amazon Aurora 端點連線。

> **Tip**
>
> 在每個區域中 AWS 區域，可用區域 (AZs) 代表彼此不同的位置，以便在中斷時提供隔離。建議您將資料庫叢集中的主要執行個體和讀取器執行個體分配在多個可用區域中，以提升資料庫叢集的可用性。如此一來，影響整個可用區域的問題將不會導致叢集中斷。
> 您可以在建立叢集時做出簡單的選擇，以設定多個可用區資料庫叢集。您可以使用 AWS 管理主控台、 AWS CLI或 Amazon RDS API。您也可以透過新增讀取器執行個體並指定不同的可用區域，將現有 Aurora 資料庫叢集變成多可用區資料庫叢集。

使用 Aurora 全域資料庫跨 AWS 區域提供高可用性

如需跨多個資料庫的高可用性 AWS 區域，您可以設定 Aurora 全域資料庫。每個 Aurora 全域資料庫跨越多個 AWS 區域，可實現低延遲的全域讀取，並從 中斷進行災難復原 AWS 區域。Aurora 會將所有資料和更新從主要 非同步複寫 AWS 區域 到每個次要區域。如需詳細資訊，請參閱使用 Amazon Aurora 全球資料庫。

Aurora 資料庫叢集的容錯能力

Aurora 資料庫叢集特意設計為具備容錯能力。叢集磁碟區跨越單一 中的多個可用區域 (AZs) AWS 區域，而每個可用區域都包含叢集磁碟區資料的副本。此功能意味著資料庫叢集可承受可用區域故障，完全不會遺失資料，服務只會短暫中斷。

如果資料庫叢集中的主要執行個體失敗，Aurora 可透過兩種方式自動容錯移轉至新的主要執行個體：

- 將現有的 Aurora 複本提升至新的主要執行個體
- 建立新的主要執行個體

如果資料庫叢集有一或多個 Aurora 複本，則在失敗事件期間會將 Aurora 複本提升為主要執行個體。失敗事件會導致短暫中斷，在此期間，讀取和寫入操作會失敗，並引發例外狀況。不過，服務通常會在 60 秒之內還原，往往不超過 30 秒。若要提高資料庫叢集的可用性，建議在兩個以上不同的可用區域建立至少一或多個 Aurora 複本。

> **Tip**
>
> 在 Aurora MySQL 中，您可以在叢集中擁有多個讀取器資料庫執行個體，以提升容錯移轉期間的可用性。在 Aurora MySQL 中，Aurora 只會重新啟動寫入器資料庫執行個體和容錯移轉讀取

器執行個體。叢集中的其他讀取器執行個體仍可使用，容錯移轉期間可以繼續透過連線至讀取器端點來處理查詢。
您也可以在容錯移轉期間使用 RDS Proxy 搭配 Aurora 資料庫叢集，提升可用性。如需詳細資訊，請參閱與 Amazon RDS Proxy 的高可用性。

您可以指派每個 Aurora 複本的優先順序，以自訂複本在失敗之後提升為主要執行個體的順序。優先順序從 0 (代表最高優先順序) 到 15 (代表最低優先順序)。如果主要執行個體失敗，Amazon RDS 會將優先順序最高的 Aurora 複本提升為新的主要執行個體。您隨時都可修改 Aurora 複本的優先順序。修改優先順序不會觸發容錯移轉。

多個 Aurora 複本可以共用相同的優先順序，形成提升層。如果兩個以上的 Aurora 複本共用相同的優先順序，Amazon RDS 會提升最大的複本。如果兩個以上的 Aurora 複本擁有相同的優先順序和大小，Amazon RDS 會提升相同提升層中的任意複本。

> **Note**
>
> 識別容錯移轉目標時涉及幾個因素。嘗試容錯移轉失敗五次後，就不會再考慮提升層。

如果資料庫叢集未包括任何 Aurora 複本，則會在失敗事件期間於相同 AZ 中重建主要執行個體。失敗事件會導致中斷，在此期間，讀取和寫入操作會失敗，並引發例外狀況。建立新的主要執行個體後，服務就會恢復，通常不超過 10 分鐘。將 Aurora 複本提升為主要執行個體比建立新的主要執行個體快得多。

假設叢集中的主要執行個體因中斷影響整個可用區域而無法使用。在此情況下，將新的主要執行個體上線的方式，取決於您的叢集是否使用多可用區域組態：

- 若您佈建或 Aurora Serverless v2 叢集包含其他可用區域中的任何讀取器執行個體，則 Aurora 會使用容錯移轉機制，將其中一個讀取器執行個體提升為新的主要執行個體。
- 若您佈建或 Aurora Serverless v2 叢集僅包含單一資料庫執行個體，或者主要執行個體和所有讀取器執行個體位於相同的可用區域，則必須在另一個可用區域中手動建立一個或多個新的資料庫執行個體。
- 如果您的叢集使用 Aurora Serverless v1，則 Aurora 會在另一個可用區域中自動建立新的資料庫執行個體。但是，該程序涉及主機取代，因此需要比容錯移轉更長的時間。

> **Note**
>
> Amazon Aurora 也支援複寫外部 MySQL 資料庫或 RDS MySQL 資料庫執行個體。如需詳細資訊，請參閱Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。

與 Amazon RDS Proxy 的高可用性

使用 RDS Proxy，您可以建置能夠透明地容忍資料庫失敗的應用程式，而不需要撰寫複雜的失敗處理程式碼。Proxy 會自動將流量路由到新的資料庫執行個體，同時保留應用程式連線。它也會略過網域名稱系統 (DNS) 快取，將 Aurora 多可用區域資料庫的容錯移轉時間縮短高達 66%。如需詳細資訊，請參閱Amazon RDS Proxy for Aurora。

以 Amazon Aurora 進行複寫

Aurora 有多種複寫選項。每個 Aurora 資料庫叢集在同一叢集中的多個資料庫執行個體之間都有內建複寫。您也可以將 Aurora 叢集的複寫設定為來源或目標。當您將資料複寫進入或移出 Aurora 叢集時，您可以選擇內建功能，例如 Aurora 全域資料庫或 MySQL 或 PostgreSQL 資料庫引擎的傳統複寫機制。您可以根據提供高可用性、便利性和效能之正確組合的選項，選擇滿足您需求的適當選項。以下部分說明如何與何時選擇各個技巧。

主題

- Aurora 複本
- 以 Aurora MySQL 進行複寫
- 以 Aurora PostgreSQL 進行複寫

Aurora 複本

當您在 Aurora 佈建的資料庫叢集中建立第二個、第三個等資料庫執行個體時，Aurora 會自動設定從寫入器資料庫執行個體到所有其他資料庫執行個體的複寫。這些其他資料庫執行個體是唯讀的，被稱為 Aurora 複本。我們也將它們稱為讀取器執行個體，討論您可以在叢集中結合寫入器和讀取器資料庫執行個體的方式。

Aurora 複本有兩個主要目的。您可以向他們發出查詢，以縮放應用程式的讀取操作。您通常會連線到叢集的讀取者端點來執行這項操作。如此一來，Aurora 可以將唯讀連線的負載分散到叢集中的所

有 Aurora 複本。Aurora 複本也有助於提高可用性。如果叢集中的寫入器執行個體變成無法使用，則 Aurora 會自動升級其中一個讀取器執行個體，以作為新的寫入器取而代之。

Aurora 資料庫叢集最多可以包含 15 個 Aurora 複本。Aurora 複本可以分佈在 區域內資料庫叢集跨越的可用 AWS 區域。

資料庫叢集中的資料擁有自己的高可用性和可靠性功能，與叢集中的資料庫執行個體無關。如果您不熟悉 Aurora 儲存功能，請參閱Amazon Aurora 儲存體的概觀。資料庫叢集磁碟區實際由資料庫叢集的多個資料副本組成。資料庫叢集中的主要執行個體和 Aurora 複本都會將叢集磁碟區中的資料視為單一邏輯磁碟區。

因此，所有 Aurora 複本都會針對查詢結果，以最少的複本延遲傳回相同的資料複本。在主要執行個體寫入更新後，此延遲通常遠低於 100 毫秒。複本延遲會根據資料庫變更率而有所不同。亦即，在資料庫發生大量寫入操作的期間，您可能看到複本延遲增加。

> **Note**
>
> 當 Aurora PostgreSQL 在以下 Aurora PostgreSQL 版本中與寫入器資料庫執行個體失去通訊超過 60 秒時，Aurora 複本會自動重新啟動：
>
> - 14.6 和較舊版本
> - 13.9 和較舊版本
> - 12.13 和較舊版本
> - Aurora PostgreSQL 11 版本
>
> 使用讀取可用性功能時，Aurora 複本不會自動重新啟動。如需讀取可用性功能及其適用版本的詳細資訊，請參閱 改善 Aurora 複本的讀取可用性。

Aurora 複本很適用於讀取擴展，因為完全專用於叢集磁碟區上的讀取操作。寫入操作由主要執行個體管理。因為叢集磁碟區是在資料庫叢集的所有資料庫執行個體之間共用，所以只需要最少的額外工作，即可複寫每一個 Aurora 複本的資料副本。

若要增加可用性，您可以使用 Aurora 複本做為容錯移轉目標。亦即，如果主要執行個體失敗，則 Aurora 複本會提升為主要執行個體。在對主要執行個體提出的讀取和寫入請求由於例外狀況而失敗的期間，會發生短暫的中斷。

以容錯移轉提升 Aurora 複本遠比重新建立主要執行個體快得多。如果您的 Aurora 資料庫叢集不包含任何 Aurora 複本，則在資料庫執行個體從失敗事件還原期間，將無法使用該資料庫叢集。

發生容錯移轉時，某些 Aurora 複本可能會重新啟動，具體取決於資料庫引擎版本。例如，在 Aurora MySQL 中，Aurora 只會在容錯移轉期間重新啟動寫入器資料庫執行個體和容錯移轉目標。如需詳細了解不同 Aurora 資料庫引擎版本重新啟動行為，請參閱 重新啟動 Amazon Aurora 資料庫叢集或 Amazon Aurora 資料庫執行個體。如需詳細了解重新啟動或容錯移轉時，頁面快取的情況，請參閱 可存活的頁面快取。

對於高可性案例，我們建議您建立一個或多個 Aurora 複本。這些應該屬於與主要執行個體相同的資料庫執行個體類別，且位於 Aurora 資料庫叢集的不同可用區域中。如需 Aurora 複本做為容錯移轉目標的詳細資訊，請參閱 Aurora 資料庫叢集的容錯能力。

您無法為未加密的 Aurora 資料庫叢集建立加密的 Aurora 複本。您無法為加密的 Aurora 資料庫叢集建立未加密的 Aurora 複本。

> **Tip**
>
> 您可以使用 Aurora 叢集內的 Aurora 複本作為唯一的複寫形式，以保持資料的高可用性。您也可以將內建 Aurora 複寫與其他類型的複寫結合。這樣做有助於為您的資料提供額外層級的高可用性和地理分佈。

如需如何建立 Aurora 複本的詳細資訊，請參閱將 Aurora 複本新增至資料庫叢集。

以 Aurora MySQL 進行複寫

除了 Aurora 複本外，您還有下列選項可使用 Aurora MySQL 進行複寫：

- 不同 AWS 區域中的 Aurora MySQL 資料庫叢集。
  - 您可以使用 Aurora 全域資料庫，跨多個區域複寫資料。如需詳細資訊，請參閱 使用 Aurora 全域資料庫跨 AWS 區域提供高可用性。
  - 您可以使用 MySQL 二進位日誌 (binlog) 複寫，在不同 AWS 區域中建立 Aurora MySQL 資料庫叢集的 Aurora 僅供讀取複本。每個叢集能以這種方式最多建立五個僅供讀取複本，每個複本都位於不同的區域。
- 相同 區域中的兩個 Aurora MySQL 資料庫叢集，方法為使用 MySQL 二進位日誌 (binlog) 複寫。
- 透過建立 RDS for MySQL 資料庫執行個體的 Aurora 僅供讀取複本，將 RDS for MySQL 資料庫執行個體作為資料來源和 Aurora MySQL 資料庫叢集。通常，您會將此方法用於遷移至 Aurora MySQL，而非進行持續複寫。

如需以 Aurora MySQL 進行複寫的詳細資訊，請參閱以 Amazon Aurora MySQL 進行複寫。

以 Aurora PostgreSQL 進行複寫

除了 Aurora 複本外，您還有下列選項可使用 Aurora PostgreSQL 進行複寫：

- 使用 Aurora 全域資料庫，一個區域的主要資料庫叢集和不同區域中最多 10 個唯讀次要資料庫叢集。Aurora PostgreSQL 不支援跨區域 Aurora 複本。不過，您可以使用 Aurora 全域資料庫將 Aurora PostgreSQL 資料庫叢集的讀取功能擴展到多個 AWS 區域，並滿足可用性目標。如需詳細資訊，請參閱使用 Amazon Aurora 全球資料庫。
- 使用 PostgreSQL 的邏輯複寫功能，在同一區域中的兩個 Aurora PostgreSQL 資料庫叢集。
- 透過建立 RDS for PostgreSQL 資料庫執行個體的 Aurora 僅供讀取複本，將 RDS for PostgreSQL 資料庫執行個體作為資料來源和 Aurora PostgreSQL 資料庫叢集。通常，您會將此方法用於遷移至 Aurora PostgreSQL，而非進行持續複寫。

如需以 Aurora PostgreSQL 進行複寫的詳細資訊，請參閱以 Amazon Aurora PostgreSQL 進行複寫。

Aurora 資料庫執行個體計費

Amazon Aurora 叢集中的 Amazon RDS 佈建執行個體是根據下列要素計費：

- 資料庫執行個體小時數 (每小時) – 根據資料庫執行個體的資料庫執行個體類別 (例如 db.t2.small 或 db.m4.large)。定價以每小時為單位列出，但帳單已採用秒數為計算單位，並以十進位制顯示時間。RDS 用量以 1 秒遞增方式進行計費，最低計費標準為 10 分鐘。如需詳細資訊，請參閱Amazon Aurora 資料庫執行個體類別。
- 儲存 (每月每 GiB) – 您的資料庫執行個體佈建的儲存容量。如果您在月中擴展所佈建儲存容量的規模，則按比例計費。如需詳細資訊，請參閱Amazon Aurora 儲存體。
- 輸入/輸出 (I/O) 請求數 (每 100 萬個請求) – 您在計費週期中提出的儲存輸入/輸出請求總數，僅適用於 Aurora Standard 資料庫叢集組態。

  如需 Amazon Aurora I/O 計費的詳細資訊，請參閱 Amazon Aurora 資料庫叢集的儲存組態。
- 備份儲存 (每月每 GiB) – 備份儲存是指與自動資料庫備份，以及所有已建立之使用中資料庫快照相關聯的儲存。延長您的備份保留期或拍攝額外的資料庫快照，會增加資料庫所消耗的備份儲存。每秒計費不是適用於備份儲存 (以 GB 增量計算)。

  如需詳細資訊，請參閱備份與還原 Amazon Aurora 資料庫叢集。
- 資料傳輸 （每 GB) – 資料庫執行個體進出網際網路和其他 AWS 區域的資料傳輸。如需實用範例，請參閱 AWS 部落格文章探索 AWS 受管資料庫的資料傳輸成本。

Amazon RDS 提供下列購買選項，可讓您根據需求選擇最適合的成本：

- On-Demand instances (隨需執行個體) – 依您使用的資料庫執行個體小時數付費。定價以每小時為單位列出，但帳單已採用秒數為計算單位，並以十進位制顯示時間。RDS 用量現以 1 秒遞增方式進行計費，最低計費標準為 10 分鐘。
- Reserved instances (預留執行個體) – 可選擇保留資料庫執行個體一年或三年，相較於隨需執行個體定價，可獲得極高的折扣。利用預留執行個體用量，您可以在一個小時內啟動、刪除、開始及停止等多個執行個體，並且讓所有執行個體皆獲得預留執行個體的好處。
- Aurora Serverless v2 – Aurora Serverless v2 提供隨需容量，其中計費單位為 Aurora 容量單位 (ACU) 小時，而非資料庫執行個體小時。Aurora Serverless v2 容量在指定的範圍內增加或減少，取決於資料庫上的負載。您可以設定所有容量皆為 Aurora Serverless v2 的叢集。或者，您以可設定 Aurora Serverless v2 或和隨需或預留佈建執行個體的組合。如需 Aurora Serverless v2 ACU 運作方式的詳細資訊，請參閱 Aurora Serverless v2 的運作方式。
- Aurora PostgreSQL Limitless Database – Aurora PostgreSQL Limitless Database 是一種自動化的水平擴展功能，可擴展到超過單一資料庫執行個體的寫入輸送量和儲存限制。Limitless Database 會將工作負載分配到多個 Aurora 寫入器執行個體，同時維持單一資料庫的操作簡易性。Limitless Database 提供隨需容量，其中計費單位是資料庫碎片群組中的 Aurora 容量單位 (ACU) 小時。如需 Limitless Database ACU 運作方式的詳細資訊，請參閱 建立資料庫叢集，此叢集會使用 Aurora PostgreSQL Limitless Database。

如需 Aurora 定價資訊，請參閱 Aurora 定價頁面。

主題

- Aurora 的隨需資料庫執行個體
- 預留資料庫執行個體 Amazon Aurora

Aurora 的隨需資料庫執行個體

Amazon RDS 隨需資料庫執行個體依資料庫執行個體類別計費 (例如 db.t3.small 或 db.m5.large)。如需 Amazon RDS 定價資訊，請參閱 Amazon RDS 產品頁面。

資料庫執行個體的帳單週期從該資料庫執行個體可用時開始計算。定價以每小時為單位列出，但帳單已採用秒數為計算單位，並以十進位制顯示時間。Amazon RDS 用量以 1 秒增量改進方式進行計費，最低計費標準為 10 分鐘。在應計費組態變更的情況下 (像是擴展運算或儲存容量)，系統最少會向您收取 10 分鐘的費用。計費將持續到資料庫執行個體終止為止，當您刪除資料庫執行個體，或是資料庫執行個體失敗時，執行個體就會終止。

如果您不再希望支付資料庫執行個體的費用，就必須將其停止或刪除，以免產生更多應計費的執行個體小時數。如需產生計費之資料庫執行個體狀態的詳細資訊，請參閱 在 Aurora 叢集中檢視 資料庫執行個體狀態。

停止的資料庫執行個體

資料庫執行個體停止時，您需要支付佈建儲存 (包含佈建 IOPS) 的費用。您也需支付備份儲存 (含指定保留時段內的手動快照和自動備份) 的費用。您無須支付資料庫執行個體小時數的費用。

多個可用區資料庫執行個體

多可用區域設定會透過自動佈建和維護不同可用區域中的同步待命複本，來增強資料持久性和可用性。由於額外的資源和更高的可用性，多可用區域部署的價格高於單一可用區域部署，而且由於額外的待命執行個體和相關聯的資源，成本可能高達兩倍。

請考慮下列有關多可用區域定價的重要詳細資訊：

- 運算成本：針對主要和待命執行個體按資料庫執行個體小時計費。
- 儲存成本：針對主要和待命執行個體按已佈建儲存體每月 GB 收費。
- 資料傳輸成本：主要執行個體和待命執行個體之間的複寫包含在成本中，但根據您的用量，可能會收取其他資料傳輸費用。

若要根據您的特定使用案例和 AWS 區域準確估算每月成本，您可以使用 AWS 定價計算工具。此工具可讓您輸入組態詳細資訊，並提供完整的成本明細。

> **Note**
>
> 定價可能會有所變更。如需最新資訊，請參閱 Amazon RDS 定價頁面。

預留資料庫執行個體 Amazon Aurora

使用預留資料庫執行個體，即可保留資料庫執行個體一或三年。相較於隨需資料庫執行個體的定價，預留資料庫執行個體可提供您更多的折扣。預留資料庫執行個體並非實體執行個體，而是一種套用到您帳戶中特定隨需資料庫執行個體用量的計費折扣。預留資料庫執行個體的折扣依執行個體類型和 AWS 區域區域而異。

使用預留資料庫執行個體的一般流程如下：先取得關於可用的預留資料庫執行個體方案的資訊，接著購買預留資料庫執行個體方案，最後再取得關於您現有預留資料庫執行個體的資訊。

如需有關購買預留資料庫執行個體和檢視預留資料庫執行個體帳單的資訊，請參閱下列各節。

- 購買 Amazon Aurora 的預留資料庫執行個體
- 檢視 Amazon Aurora 預留資料庫執行個體的帳單

預留資料庫執行個體概觀

當您在 Amazon RDS 中購買預留資料庫執行個體時，您買到的是特定資料庫執行個體類型在預留資料庫執行個體期間內得享有折扣費率的承諾。若要使用 Amazon RDS 預留資料庫執行個體，您需要建立新的資料庫執行個體，如同建立隨需執行個體。

對於下列項目，您建立的新資料庫執行個體必須具有與預留資料庫執行個體相同的規格。

- AWS 區域
- 資料庫引擎 (資料庫引擎的版本號碼不需要相符。)
- 資料庫執行個體類型

如果新資料庫執行個體規格與您帳戶現有的預留執行個體相符，您將以預留資料庫執行個體的折扣費率計費。否則，預留資料庫執行個體將按隨需費率計費。

您可以修改做為預留資料庫執行個體的資料庫執行個體。若修改符合預留資料庫執行個體的規格，則部分或全部折扣仍適用於修改後的資料庫執行個體。若修改超出規格 (例如變更執行個體類別)，則折扣將不再適用。如需詳細資訊，請參閱彈性大小的預留資料庫執行個體。

主題

- 方案類型
- Aurora 資料庫叢集組態彈性
- 彈性大小的預留資料庫執行個體

- Aurora 預留資料庫執行個體計費範例
- 刪除預留資料庫執行個體

如需預留資料庫執行個體的詳細資訊，包括定價，請參閱 Amazon RDS 預留執行個體。

方案類型

預留資料庫執行個體分為三種—無預付、部分預付和全額預付—可讓您依據預期的使用量將 Amazon RDS 成本最佳化。

> **Note**
>
> 並非所有 RDS 執行個體類別都支援所有預留執行個體方案類型。例如，某些執行個體類別可能不會提供「不預付」選項。若要確認可用性，請檢閱 中的預留執行個體方案， AWS 管理主控台 或使用 `describe-reserved-db-instances-offerings` AWS CLI 命令。

不預付

此選項讓您不用支付預付款便能存取預留資料庫執行個體。無預付的預留資料庫執行個體在期間中以折扣後的每小時費率計費，無論是否有使用，而且不需要預付款。這個選項只適用於為期一年的預留。

部分預付

此選項需預先支付部分的預留資料庫執行個體。期間內其餘的時數會以折扣後的每小時費率計費，無論是否有使用。此選項為先前重度使用選項的替代方案。

全額預付

期間開始時便支付完整的款項，並在期間的剩餘部分不會產生其他成本或額外的每小時費用，無論使用多少小時。

如果您使用合併帳單，組織中的所有帳戶都會視為一個帳戶處理。這表示組織中的所有帳戶可以獲得其他任何帳戶購買之預留資料庫執行個體的每小時成本利益。如需更多有關合併帳單的資訊，請參閱 AWS 帳單與成本管理使用者指南中的 Amazon RDS 預留資料庫執行個體。

Aurora 資料庫叢集組態彈性

您可以使用 Aurora 預留資料庫執行個體搭配這兩種資料庫叢集組態：

- Aurora I/O-Optimized – 您只需為資料庫叢集的使用量和儲存付費，而讀取和寫入 I/O 操作無須額外付費。
- Aurora Standard – 除了資料庫叢集的使用量和儲存之外，您還會為 I/O 操作支付每 100 萬個請求的標準費率。

Aurora 會自動說明這些組態之間的價格差異。Aurora I/O-Optimized 每小時會比 Aurora Standard 多耗用 30% 標準化單位。

如需 Aurora 叢集儲存組態的詳細資訊，請參閱 Amazon Aurora 資料庫叢集的儲存組態。如需 Aurora 叢集儲存組態的定價詳細資訊，請參閱 Amazon Aurora 定價。

彈性大小的預留資料庫執行個體

當您購買預留資料庫執行個體時，您應指定的其中一項內容為執行個體類別，例如 db.r5.large。如需資料庫執行個體類別的詳細資訊，請參閱 Amazon Aurora 資料庫執行個體類別。

如果您有資料庫執行個體，而且您需要擴展為更大的容量，則您的預留資料庫執行個體將自動套用到擴展後的資料庫執行個體。也就是說，預留資料庫執行個體會自動套用到所有的資料庫執行個體類別大小。彈性大小的預留資料庫執行個體適用於具有相同 AWS 區域 和 資料庫引擎的資料庫執行個體。彈性大小的預留資料庫執行個體只能在其執行個體類別類型中擴展。例如，db.r6i.large 的預留資料庫執行個體可以套用至 db.r6i.xlarge，但不適用於 db.r6id.large 或 db.r7g.large，因為 db.r6id.large 和 db.r7g.large 是不同的執行個體類別類型。

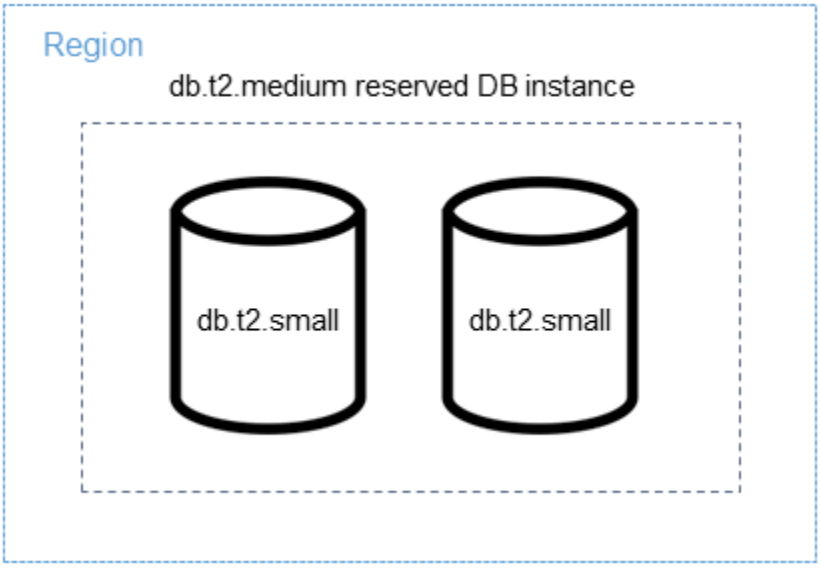
彈性大小的預留資料庫執行個體可供下列 Aurora 資料庫引擎使用：

- Aurora MySQL
- Aurora PostgreSQL

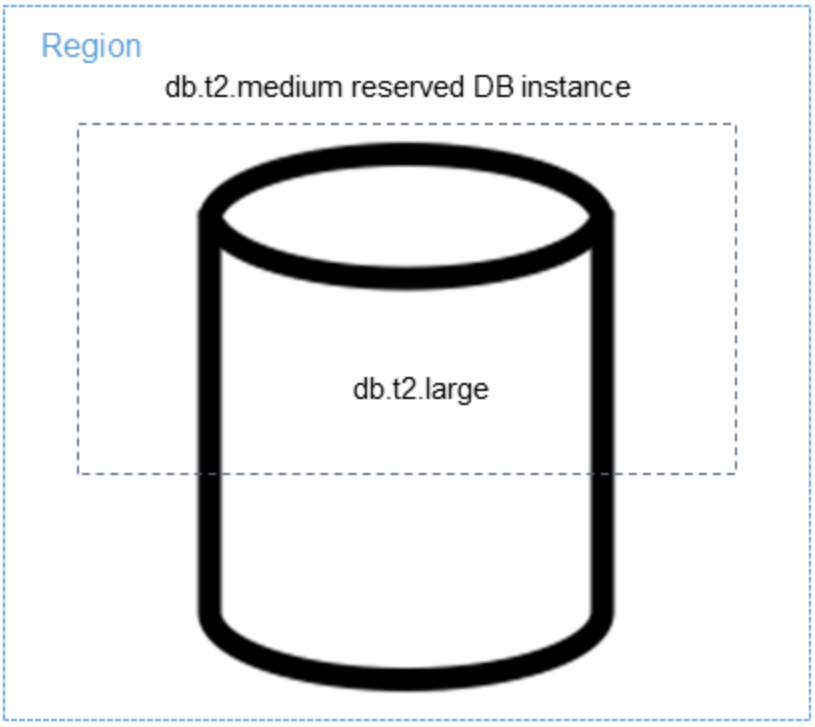
您可以使用每小時標準化單位數來比較不同預留資料庫執行個體大小的使用量。例如，兩個 db.r3.large 資料庫執行個體的一單位使用量相當於一個 db.r3.small 的每小時 8 個標準化單位使用量。下表顯示每個資料庫執行個體大小的每小時標準化單位數。

執行個體大小	一個資料庫執行個體每小時標準化單位數，Aurora Standard	一個資料庫執行個體每小時標準化單位數，Aurora I/O-Optimized	三個資料庫執行個體 (寫入器和兩個讀取器) 每小時標準化單位數，Aurora Standard	三個資料庫執行個體 (寫入器和兩個讀取器) 每小時標準化單位數，Aurora I/O-Optimized
小型	1	1.3	3	3.9
中型	2	2.6	6	7.8
大型	4	5.2	12	15.6
xlarge	8	10.4	24	31.2
2xlarge	16	20.8	48	62.4
4xlarge	32	41.6	96	124.8
8xlarge	64	83.2	192	249.6
12xlarge	96	124.8	288	374.4
16xlarge	128	166.4	384	499.2
24xlarge	192	249.6	576	748.8
32xlarge	256	332.8	768	998.4

例如，假設您購買 `db.t2.medium` 預留資料庫執行個體，且在相同 AWS 區域中您帳戶內有兩個執行中的 `db.t2.small` 資料庫執行個體。在此情況下，計費利益便會立即完整套用到兩個執行個體。



或者，如果您的帳戶中有一個db.t2.large執行個體在相同 中執行 AWS 區域，則帳單利益會套用至資料庫執行個體 50% 的使用量。



> **Note**
>
> 建議您在開發、測試伺服器或其他非生產伺服器時，僅使用 T 資料庫執行個體類別。如需詳細了解 T 執行個體類別，請參閱 資料庫執行個體類別的類型。

Aurora 預留資料庫執行個體計費範例

下列範例針對同時使用 Aurora Standard 和 Aurora I/O-Optimized 資料庫叢集組態的 Aurora 資料庫叢集說明預留資料庫執行個體的定價。

使用 Aurora Standard 的範例

預留資料庫執行個體的價格不會為與儲存、備份和 I/O 相關聯的成本提供折扣。以下範例說明預留資料庫執行個體的每月成本總計：

- 對於美國東部 (維吉尼亞北部) 的 Aurora MySQL 預留單一可用區域 db.r5.large 資料庫執行個體類別，費用為每小時 $0.19，或每月 $138.70
- 對於 Aurora 儲存，費用為每 GiB 每月 $0.10 (此範例的費用為每月 $45.60)
- 對於 Aurora 輸入/輸出，費用為每 1 百萬次請求 $0.20 (此範例的費用為每月 $20)
- 對於 Aurora 備份儲存，費用為每 GiB 每月 $0.021 (此範例的費用為每月 $30)

對於預留資料庫執行個體新增所有這些選項 ($138.70 + $45.60 + $20 + $30)，每月總成本為 $234.30。

如果您選擇使用隨需資料庫執行個體，而非預留資料庫執行個體，對於美國東部 (維吉尼亞北部) 的 Aurora MySQL 預留單一可用區域 db.r5.large 資料庫執行個體類別，費用為每小時 $0.29，或每月 $217.50。因此，對於隨需資料庫執行個體，新增所有這些選項 ($217.50 + $45.60 + $20 + $30)，每月總成本為 $313.10。您可以使用預留資料庫執行個體，每月節省將近 79 美元。

使用 Aurora Standard 資料庫叢集搭配兩個讀取器執行個體的範例

若要將預留執行個體用於 Aurora 資料庫叢集，只需針對叢集中的每個資料庫執行個體購買一個預留執行個體。

延伸第一個範例，您具有一個 Aurora MySQL 資料庫叢集，其中包含一個寫入器資料庫執行個體和兩個 Aurora 複本，因此在叢集中總共有三個資料庫執行個體。兩個 Aurora 複本不會產生額外的儲存或備份費用。如果您購買三個 db.r5.large Aurora MySQL 預留資料庫執行個體，您的成本將為 $234.30 (適用於寫入器資料庫執行個體) + 2 * (每個 Aurora 複本 $138.70 + $20 I/O)，每月總計為 $551.70。

對於具有一個寫入器資料庫執行個體和兩個 Aurora 複本的 Aurora MySQL 資料庫叢集，相應的隨需費用為 $313.10 + 2 * (每個執行個體 $217.50 + $20 I/O)，每月總計為 $788.10。您可以使用預留資料庫執行個體，每月節省 $236.40。

使用 Aurora I/O-Optimized 的範例

您可以搭配 Aurora I/O-Optimized 重複使用現有的 Aurora Standard 預留資料庫執行個體。若要搭配 Aurora I/O-Optimized 充分利用預留執行個體折扣的優勢，您可以購買與目前預留執行個體類似的 30% 額外預留執行個體。

下表顯示如何在使用 Aurora I/O-Optimized 時預估額外預留執行個體的範例。如果所需的預留執行個體只是一小部分，您可以利用預留執行個體提供的大小彈性來獲得整數。在這些範例中，「目前」是指您現在擁有的 Aurora Standard 預留執行個體。額外的預留執行個體是您必須購買的 Aurora Standard 預留執行個體數量，才能在使用 Aurora I/O-Optimized 時維持目前的預留執行個體折扣。

DB instance class (資料庫執行個體類別)	目前 Aurora Standard 預留執行個體	Aurora I/O-Optimized 所需的預留執行個體	需要額外的預留執行個體	需要額外的預留執行個體，使用大小彈性
db.r6g.large	10	10 * 1.3 = 13	3 * db.r6g.large	3 * db.r6g.large
db.r6g.4xlarge	20	20 * 1.3 = 26	6 * db.r6g.4xlarge	6 * db.r6g.4xlarge
db.r6g.12xlarge	5	5 * 1.3 = 6.5	1.5 * db.r6g.12xlarge	db.r6g.12xlarge、r6g.4xlarge 和 r6g.2xlarge 每個一個 (0.5 * db.r6g.12xlarge = 1 * db.r6g.4xlarge + 1 * db.r6g.2xlarge)
db.r6i.24xlarge	15	15 * 1.3 = 19.5	4.5 * db.r6i.24xlarge	4 * db.r6i.24xlarge + 1 * db.r6i.12xlarge

DB instance class (資料庫執行個體類別)	目前 Aurora Standard 預留執行個體	Aurora I/O-Optimized 所需的預留執行個體	需要額外的預留執行個體	需要額外的預留執行個體，使用大小彈性
				(0.5 * db.r6i.24xlarge = 1 * db.r6i.12xlarge)

使用 Aurora I/O-Optimized 資料庫叢集搭配兩個讀取器執行個體的範例

您具有一個 Aurora MySQL 資料庫叢集，其中包含一個寫入器資料庫執行個體和兩個 Aurora 複本，因此在叢集中總共有三個資料庫執行個體。它們使用 Aurora I/O-Optimized 資料庫叢集組態。若要針對此叢集使用預留資料庫執行個體，您需要購買相同資料庫執行個體類別的四個預留資料庫執行個體。相較於三個使用 Aurora Standard 的資料庫執行個體每小時 3 個標準化單位，三個使用 Aurora I/O-Optimized 的資料庫執行個體每小時則耗用 3.9 個標準化單位。不過，您可以節省每個資料庫執行個體的每月 I/O 成本。

> **Note**
>
> 這些範例中的價格是範例價格，可能與實際價格不符。如需 Aurora 定價資訊，請參閱 Amazon Aurora 定價。

刪除預留資料庫執行個體

預留資料庫執行個體的期間一般為一年或三年承諾。您無法取消預留資料庫執行個體。但是，您可以刪除預留資料庫執行個體折扣所涵蓋的資料庫執行個體。刪除預留資料庫執行個體折扣所涵蓋之資料庫執行個體的流程，與其他任何資料庫執行個體相同。

無論您是否使用這些資源，都會向您收取預付費用。

如果刪除預留資料庫執行個體折扣所涵蓋的資料庫執行個體，您仍可以啟動其他規格相容的資料庫執行個體。在此情況下，您仍可以在保留時間 (一或三年) 內繼續享有折扣費率。

購買 Amazon Aurora 的預留資料庫執行個體

您可以使用 AWS 管理主控台 AWS CLI、 和 RDS API 來使用預留資料庫執行個體。

主控台

您可以使用 AWS 管理主控台 來處理預留資料庫執行個體，如下列程序所示。

取得可用的預留資料庫執行個體方案的定價與資訊

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Reserved instances (預留執行個體)。
3. 選擇 Purchase Reserved DB Instance (購買預留資料庫執行個體)。
4. 在 Product description (產品說明) 中，選擇資料庫引擎和授權類型。
5. 在 DB instance class (資料庫執行個體類別) 中，選擇資料庫執行個體類別。
6. 針對部署選項，選擇您要單一可用區還是多可用區資料庫執行個體部署。

   > **Note**
   >
   > 預留 Amazon Aurora 執行個體一律會將部署選項設為單一可用區資料庫執行個體。不過，當您建立 Aurora 資料庫叢集時，預設部署選項為在不同的可用區中建立 Aurora 複本或讀取器 (多可用區)。
   > 您必須為計劃使用的每個執行個體購買預留資料庫執行個體，包括 Aurora 複本。因此，對於 Aurora 上的多可用區域部署，您必須購買額外的預留資料庫執行個體。

7. 在期限中，選擇您想要預留資料庫執行個體的時間長度。
8. 在 Offering type (方案類型) 中，選擇方案類型。

   選擇方案類型後，便會顯示定價資訊。

   > **Important**
   >
   > 選擇 Cancel (取消)，將不會購買預留資料庫執行個體，也不會產生任何費用。

取得可用的預留資料庫執行個體方案資訊後，您便能利用這些資訊來購買方案，如以下程序所示。

購買預留資料庫執行個體

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。

2. 在導覽窗格中，選擇 Reserved instances (預留執行個體)。
3.

> **Important**
>
> 繼續操作前，請先確認您位於正確的 AWS 區域中。預留資料庫執行個體具有區域性，且無法在區域之間轉移。請查看主控台右上角的區域選擇器，以確保您是在預期的區域中購買預留執行個體。

4. 選擇 Purchase reserved DB instance (購買預留資料庫執行個體)。
5. 在 Product description (產品說明) 中，選擇資料庫引擎和授權類型。
6. 在 DB instance class (資料庫執行個體類別) 中，選擇資料庫執行個體類別。
7. 在多可用區部署中，選擇您要單一可用區還是多可用區資料庫執行個體部署。

> **Note**
>
> 預留 Amazon Aurora 執行個體一律會將部署選項設為單一可用區資料庫執行個體。如果您從預留資料庫執行個體建立 Amazon Aurora 資料庫叢集，資料庫叢集將自動建立為多可用區域。請務必為計劃使用的每個資料庫執行個體購買預留資料庫執行個體，包括 Aurora 複本。

8. 在 Term (期限) 中，選擇您想要預留資料庫執行個體的時間長度。
9. 在 Offering type (方案類型) 中，選擇方案類型。

   選擇方案類型後，便會顯示定價資訊。
   購買預留資料庫執行個體主控台
10. (選用) 您可將自己的識別碼指派至所購買的預留執行個體，以協助您追蹤這些執行個體。針對 Reserved Id (預留 ID) 中，輸入預留資料庫執行個體的識別碼。
11. 選擇提交。

    您的預留資料庫執行個體已購買，然後顯示在 Reserved instances (預留執行個體) 清單中。

購買預留資料庫執行個體後，您將取得預留資料庫執行個體的資訊，如以下程序所示。

取得 AWS 您帳戶預留資料庫執行個體的相關資訊

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。

2. 在 Navigation (導覽) 窗格中，選擇 Reserved instances (預留執行個體)。

   將顯示您帳戶的預留資料庫執行個體。若要查看特定預留資料庫執行個體的詳細資訊，請選擇清單中的該執行個體。在主控台底端的詳細資訊窗格中，您即可看到該執行個體的詳細資訊。

AWS CLI

您可以使用 AWS CLI 來使用預留資料庫執行個體，如下列範例所示。

Example取得可用的預留資料庫執行個體方案

若要取得可用預留資料庫執行個體產品的相關資訊，請呼叫 AWS CLI 命令 describe-reserved-db-instances-offerings。

```
aws rds describe-reserved-db-instances-offerings
```

此呼叫會傳回類似以下的輸出：

```
OFFERING  OfferingId                            Class         Multi-AZ  Duration  Fixed
 Price  Usage Price  Description  Offering Type
OFFERING  438012d3-4052-4cc7-b2e3-8d3372e0e706  db.r3.large   y         1y
 1820.00 USD  0.368 USD    mysql        Partial  Upfront
OFFERING  649fd0c8-cf6d-47a0-bfa6-060f8e75e95f  db.r3.small   n         1y
 227.50 USD  0.046 USD    mysql        Partial  Upfront
OFFERING  123456cd-ab1c-47a0-bfa6-12345667232f  db.r3.small   n         1y
 162.00 USD   0.00 USD    mysql        All      Upfront
    Recurring Charges:   Amount  Currency  Frequency
    Recurring Charges:   0.123   USD       Hourly
OFFERING  123456cd-ab1c-37a0-bfa6-12345667232d  db.r3.large   y         1y
 700.00 USD   0.00 USD    mysql        All      Upfront
    Recurring Charges:   Amount  Currency  Frequency
    Recurring Charges:   1.25    USD       Hourly
OFFERING  123456cd-ab1c-17d0-bfa6-12345667234e  db.r3.xlarge  n         1y
 4242.00 USD   2.42 USD    mysql        No       Upfront
```

取得可用的預留資料庫執行個體方案資訊後，您便能利用這些資訊來購買方案。

若要購買預留資料庫執行個體，請使用 AWS CLI 命令purchase-reserved-db-instances-offering搭配下列參數：

- `--reserved-db-instances-offering-id` – 您想要購買之方案的 ID。請參閱上述範例，取得方案的 ID。
- `--reserved-db-instance-id` – 您可將自己的識別符指派至所購買的預留資料庫執行個體，以幫助追蹤這些執行個體。

Example購買預留資料庫執行個體

下列範例將購買 ID *649fd0c8-cf6d-47a0-bfa6-060f8e75e95f* 的預留資料庫執行個體方案，並指派識別碼 *MyReservation*。

對於 Linux、macOS 或 Unix：

```
aws rds purchase-reserved-db-instances-offering \
    --reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f \
    --reserved-db-instance-id MyReservation
```

在 Windows 中：

```
aws rds purchase-reserved-db-instances-offering ^
    --reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f ^
    --reserved-db-instance-id MyReservation
```

此命令會傳回類似以下的輸出：

```
RESERVATION  ReservationId      Class        Multi-AZ  Start Time
 Duration  Fixed Price  Usage Price  Count  State           Description  Offering Type
RESERVATION  MyReservation      db.r3.small  y         2011-12-19T00:30:23.247Z  1y
    455.00 USD   0.092 USD    1      payment-pending  mysql        Partial  Upfront
```

購買預留資料庫執行個體後，您將取得預留資料庫執行個體的資訊。

若要取得 AWS 您帳戶預留資料庫執行個體的相關資訊，請呼叫 AWS CLI 命令 [describe-reserved-db-instances](#)，如下列範例所示。

Example取得您的預留資料庫執行個體

```
aws rds describe-reserved-db-instances
```

此命令會傳回類似以下的輸出：

```
RESERVATION  ReservationId     Class        Multi-AZ  Start Time
 Duration  Fixed Price  Usage Price  Count  State    Description  Offering Type
RESERVATION  MyReservation     db.r3.small  y         2011-12-09T23:37:44.720Z  1y
   455.00 USD   0.092 USD    1      retired  mysql        Partial  Upfront
```

RDS API

您可以使用 RDS API 來處理預留資料庫執行個體：

- 若要取得可用的預留資料庫執行個體方案資訊，請呼叫 Amazon RDS API 操作 DescribeReservedDBInstancesOfferings。
- 取得可用的預留資料庫執行個體方案資訊後，您便能利用這些資訊來購買方案。請使用下列參數呼叫 PurchaseReservedDBInstancesOffering RDS API 操作：
  - `--reserved-db-instances-offering-id` – 您想要購買之方案的 ID。
  - `--reserved-db-instance-id` – 您可將自己的識別符指派至所購買的預留資料庫執行個體，以幫助追蹤這些執行個體。
- 購買預留資料庫執行個體後，您將取得預留資料庫執行個體的資訊。請呼叫 DescribeReservedDBInstances RDS API 操作。

檢視 Amazon Aurora 預留資料庫執行個體的帳單

您可以在 AWS 管理主控台中的 Billing Dashboard (帳單儀表板) 中檢視預留資料庫執行個體的帳單。

檢視預留資料庫執行個體帳單

1. 登入 AWS 管理主控台。
2. 從右上角的 account menu (帳戶選單) 中，選擇 Billing Dashboard (帳單儀表板)。
3. 選擇儀表板右上角的 Bill Details (帳單詳細資訊)。
4. 在 AWS Service Charges (AWS 服務費用) 中，展開 Relational Database Service (關聯式資料庫服務)。
5. 展開預留資料庫執行個體所在的 AWS 區域，例如 US West (Oregon) (美國西部 (奧勒岡))。

   當月的預留資料庫執行個體及其每小時費用會顯示在適用於#####預留執行個體的 Amazon Relational Database Service。

此範例中的預留資料庫執行個體是全部預付購買的，因此不會有每小時的收費。

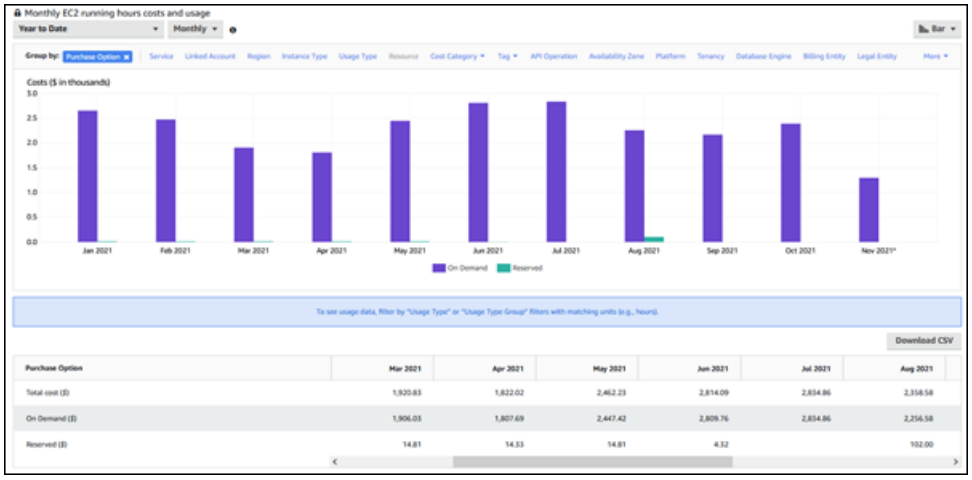
6. 選擇 Reserved Instances (預留執行個體) 標題旁邊的 Cost Explorer (長條圖) 圖示。

   Cost Explorer 會顯示 Monthly EC2 running hours costs and usage (每月 EC2 執行時數成本與用量) 圖形。

7. 清除圖表右側的 Usage Type Group (用量類型群組) 篩選條件。

8. 選擇要檢查用量成本的時間期間和時間單位。

   下列範例顯示以月份為單位的今年迄今隨需和預留資料庫執行個體的用量成本。



   2021 年 1 月到 6 月的預留資料庫執行個體成本是部分預付執行個體的每月費用，而 2021 年 8 月的成本則是全部預付執行個體的一次性費用。

   部分預付執行個體的預留執行個體折扣在 2021 年 6 月到期，但是資料庫執行個體不會刪除。到期日之後，只會按隨需費率收費。

AWS 免費方案上的 Amazon Aurora

Aurora PostgreSQL 也可以透過 AWS 免費方案取得使用快速組態建立。除了快速組態限制之外，將 Aurora PostgreSQL 與 AWS 免費方案搭配使用時，適用下列限制：

- 僅支援使用快速組態建立的叢集 （完全組態不可用）
- 每個叢集最多 4 個 Aurora 容量單位 (ACUs)
- 每個叢集最多 1 GB 的儲存空間
- 每個帳戶最多 2 個叢集和 2 個執行個體

- 不支援 Aurora 全域資料庫和零 ETL 整合

設定您的 Amazon Aurora 環境

首次使用 Amazon Aurora 之前，請先完成下列作業：

主題

- 註冊 AWS 帳戶
- 建立具有管理存取權的使用者
- 授與程式設計存取權
- 判定需求
- 建立安全群組以存取 VPC 中的資料庫叢集

如果您已有 AWS 帳戶、了解您的 Aurora 需求，並偏好使用 IAM 和 VPC 安全群組的預設值，請跳到 Amazon Aurora 入門。

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

   當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 https://aws.amazon.com/ 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊 後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center和建立管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者AWS 管理主控台身分登入 。在下一頁中，輸入您的密碼。

   如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的以根使用者身分登入。

2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

   如需說明，請參閱《IAM 使用者指南》中的為您的 AWS 帳戶 根使用者 （主控台） 啟用虛擬 MFA 裝置。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的啟用 AWS IAM Identity Center。

2. 在 IAM Identity Center 中，將管理存取權授予使用者。

   如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱AWS IAM Identity Center 《 使用者指南》中的使用預設值設定使用者存取 IAM Identity Center 目錄。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

  如需使用 IAM Identity Center 使用者登入的說明，請參閱AWS 登入 《 使用者指南》中的登入 AWS 存取入口網站。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的建立許可集。

2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的新增群組。

授與程式設計存取權

如果使用者想要與 AWS 外部互動，則需要程式設計存取 AWS 管理主控台。授予程式設計存取權的方式取決於正在存取的使用者類型 AWS。

若要授予使用者程式設計存取權，請選擇下列其中一個選項。

哪個使用者需要程式設計存取權？	到	根據
IAM	（建議）使用主控台登入資料做為臨時登入資料，以簽署對 AWS CLI、 AWS SDKs程式設計請求。 AWS APIs	請依照您要使用的介面所提供的指示操作。 • 如需 AWS CLI，請參閱AWS Command Line Interface《使用者指南》中的登入以進行 AWS 本機開發。 • AWS SDKs請參閱 AWS SDKs和工具參考指南中的登入以進行 AWS 本機開發。
人力資源身分 (IAM Identity Center 中管理的使用者)	使用暫時登入資料簽署對 AWS CLI、 AWS SDKs程式設計請求。 AWS APIs	請依照您要使用的介面所提供的指示操作。 • 如需 AWS CLI，請參閱AWS Command Line Interface《使用者指南》中的設定 AWS CLI 要使用 AWS IAM Identity Center的 。 • AWS SDKs、工具和 AWS APIs，請參閱 AWS SDK 和工具參考指南中的 SDKsIAM Identity Center 身分驗證。
IAM	使用暫時登入資料簽署對 AWS CLI、 AWS SDKs程式設計請求。 AWS APIs	遵循《IAM 使用者指南》中將臨時登入資料與 AWS 資源搭配使用的指示。

哪個使用者需要程式設計存取權？	到	根據
IAM	(不建議使用) 使用長期登入資料來簽署對 AWS CLI、 AWS SDKs程式設計請求。 AWS APIs	請依照您要使用的介面所提供的指示操作。 • 如需 AWS CLI，請參閱AWS Command Line Interface 《 使用者指南》中的使用 IAM 使用者憑證進行身分驗證。 • AWS SDKs和工具，請參閱 AWS SDKs和工具參考指南中的使用長期憑證進行身分驗證。 • 對於 AWS APIs，請參閱《 IAM 使用者指南》中的管理 IAM 使用者的存取金鑰。

判定需求

Aurora 的基本建置區塊為資料庫叢集。資料庫叢集可以有一或多個資料庫執行個體。資料庫叢集提供一個網路地址，稱為叢集端點。每當您的應用程式需要存取資料庫叢集中建立的資料庫時，就會連線到該資料庫叢集公開的叢集端點。您建立資料庫叢集時所指定的資訊控制了組態元素，像是記憶體、資料庫引擎和版本、網路組態、安全性、維護期間。

在建立資料庫叢集和安全群組之前，您必須知道您的資料庫叢集和網路需求。這裡是一些要考慮的注意事項：

- 資源需求 – 您的應用程式或服務的記憶體和處理器需求為何？ 當您建立資料庫叢集時，將使用這些設定來判定要使用的資料庫執行個體類別。如需關於資料庫執行個體類別的規格，請參閱 Amazon Aurora 資料庫執行個體類別。
- VPC、子網路和安全群組 – 您的資料庫叢集將位於 Virtual Private Cloud (VPC) 中。您必須設定安全群組規則，才能連線至資料庫叢集。下列清單說明每個 VPC 選項的規則：
  - 預設 VPC — 如果 AWS 您的帳戶在 AWS 區域中具有預設 VPC，則該 VPC 會設定為支援資料庫叢集。如果您在建立資料庫叢集時指定預設 VPC：

    - 確認建立 VPC 安全群組，授權從應用程式或服務到 Aurora 資料庫叢集的連線。使用 VPC 主控台上的安全群組選項或 AWS CLI 來建立 VPC 安全群組。如需相關資訊，請參閱步驟 3：建立 VPC 安全群組。
    - 您必須指定預設的資料庫子網路群組。如果這是您在 AWS 區域中建立的第一個資料庫叢集，Amazon RDS 會在建立資料庫叢集時建立預設資料庫子網路群組。
  - 使用者定義的 VPC— 如果您要在建立資料庫叢集時指定使用者定義的 VPC：
    - 確認建立 VPC 安全群組，授權從應用程式或服務到 Aurora 資料庫叢集的連線。使用 VPC 主控台上的安全群組選項或 AWS CLI 來建立 VPC 安全群組。如需相關資訊，請參閱步驟 3：建立 VPC 安全群組。
    - VPC 必須符合某些需求才能主控資料庫叢集，例如具有至少兩個子網路，各位在不同的可用區域中。如需相關資訊，請參閱「Amazon VPC 和 Amazon Aurora」。
    - 您必須指定資料庫子網路群組，該群組定義在該 VPC 中可由資料庫叢集使用的子網路。如需詳細資訊，請參閱在 VPC 中使用資料庫叢集中的資料庫子網路群組小節。
- 高可用性：您需要容錯移轉支援嗎？ 在 Aurora 上，異地同步備份部署會建立主要執行個體和 Aurora 複本。您可以將主要執行個體和 Aurora 複本設定為位於不同的可用區域，以便支援容錯移轉。建議對生產工作負載使用異地同步備份部署以保有高可用性。若為了開發和測試目的，您可以使用非異地同步備份的部署。如需詳細資訊，請參閱Amazon Aurora 的高可用性。
- IAM 政策： AWS 您的帳戶是否有授予執行 Amazon RDS 操作所需許可的政策？ 如果您 AWS 使用 IAM 登入資料連線至 ，您的 IAM 帳戶必須具有授予執行 Amazon RDS 操作所需許可的 IAM 政策。如需詳細資訊，請參閱Amazon Aurora 的 Identity and access management。
- 開放連接埠：您的資料庫會在哪個 TCP/IP 上偵聽？ 某些公司的防火牆可能會封鎖與您的資料庫引擎預設連接埠的連線。如果您的公司防火牆會封鎖預設連接埠，請為新的資料庫叢集選擇另一個連接埠。請注意，建立在指定連接埠上接聽的資料庫叢集後，可以透過修改資料庫叢集來變更連接埠。
- AWS 區域：您希望資料庫位於哪個 AWS 區域？ 將資料庫放在鄰近應用程式或 Web 服務的位置可以減少網路延遲。如需更多詳細資訊，請參閱 區域和可用區域。

備妥建立安全群組和資料庫叢集所需的資訊後，請繼續進行下一個步驟。

建立安全群組以存取 VPC 中的資料庫叢集

您的資料庫叢集群將會建立在 VPC 中。安全群組提供 VPC 中資料庫叢集的存取權。其作用就像相關聯資料庫叢集的防火牆，從叢集層級控制傳入和傳出流量。資料庫叢集建立時預設提供防火牆以及可避免資料庫叢集遭到存取的預設安全群組。因此您必須將規則新增至安全群組，讓您能連線到資料庫叢集。使用您在先前步驟中判斷的網路和組態資訊，來建立允許存取資料庫叢集的規則。

例如，假設您有一個應用程式會存取 VPC 中資料庫叢集上的資料庫，則您必須新增自訂 TCP 規則，指定應用程式用來存取該資料庫的連接埠範圍和 IP 地址。如果您的應用程式在 Amazon EC2 執行個體上，則可以使用您為 Amazon EC2 執行個體設定的 VPC 安全群組。

您可以在建立資料庫叢集時，將與 Amazon EC2 執行個體之間的連線設定為資料庫叢集。如需詳細資訊，請參閱設定與 EC2 執行個體的自動網路連線。

> **Tip**
>
> 您可以在建立資料庫叢集時自動設定 Amazon EC2 執行個體和資料庫叢集之間的網路連線。如需詳細資訊，請參閱設定與 EC2 執行個體的自動網路連線。

如需有關如何將 中的資源Amazon Lightsail連接到資料庫叢集的資訊，請參閱AWS 服務 使用 VPC 對等互連將Lightsail資源連接到 。

如需建立 VPC 來搭配 Aurora 使用的更多資訊，請參閱教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4)。如需存取資料庫執行個體常見案例的相關資訊，請參閱在 VPC 中存取資料庫叢集的案例。

建立 VPC 安全群組

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/vpc：// 開啟 Amazon VPC 主控台。

   > **Note**
   >
   > 請確認您位於 VPC 主控台中，而不是 RDS 主控台。

2. 在 的右上角 AWS 管理主控台，選擇您要建立 VPC 安全群組和資料庫叢集 AWS 的區域。在 AWS 區域的 Amazon VPC 資源清單中，您應該會看到至少一個 VPC 和數個子網路。如果沒有，表示在該 AWS 區域中沒有預設 VPC。
3. 在導覽窗格中，選擇 Security Groups (安全群組)。
4. 選擇 Create Security Group (建立安全群組)。

   隨即會顯示 Create security group (建立安全群組) 頁面。
5. 在 Basic details (基本詳細資訊) 中，分別在 Security group name (安全群組名稱) 和 Description (描述) 中輸入相應內容。對於 VPC，選擇要建立資料庫叢集所在的 VPC。
6. 在 Inbound rules (入站規則) 中，選擇 Add rule (新增規則)。

a. 針對 Type (類型)，請選擇 Custom TCP (自訂 TCP)。
b. 在 Port range (連接埠範圍) 中，輸入要用於資料庫叢集的連接埠值。
c. 針對 Source (來源)，選擇要從中存取資料庫叢集的安全群組名稱或輸入 IP 地址範圍 (CIDR 值)。如果選擇 My IP (我的 IP)，此舉允許透過您的瀏覽器中偵測到的 IP 地址存取資料庫叢集。

7. 如果需要新增更多 IP 地址或不同的連接埠範圍，請選擇 Add rule (新增規則) 並輸入規則的資訊。
8. (選用) 在 Outbound Rules (輸出規則) 中，新增輸出流量的規則。預設會允許所有傳出流量。
9. 選擇建立安全群組。

建立資料庫叢集時，您可以使用剛才建立的 VPC 安全群組作為資料庫叢集的安全群組。

> **Note**
>
> 如果您使用預設 VPC，系統會為您建立橫跨所有 VPC 子網路的預設子網路群組。建立資料庫叢集時，您可以選取預設的 VPC，並使用 DB Subnet Group (資料庫子網路群組) 的 default (預設)。

完成設定需求後，您可以依照 建立 Amazon Aurora 資料庫叢集 中的指示，使用您的需求和安全群組建立資料庫叢集。如需建立使用特定資料庫引擎之資料庫叢集的相關資訊，請參閱Amazon Aurora 入門。

Amazon Aurora 入門

您可在本節中找出如何使用 Amazon RDS 來建立及連線至 Aurora 資料庫叢集。

下列程序為示範 Aurora 入門基礎知識的教學課程。稍後各節介紹更進階的 Aurora 概念和程序，例如不同類型的端點以及如何向上和向下擴展 Aurora 叢集。

> **Important**
>
> 您必須完成 設定您的 Amazon Aurora 環境 中的任務，才能建立或連線至資料庫叢集。

主題

- 建立 Aurora MySQL 資料庫叢集並與之連線
- 建立 Aurora PostgreSQL 資料庫叢集並與之連線
- 教學：建立 Web 伺服器和 Amazon Aurora 資料庫叢集

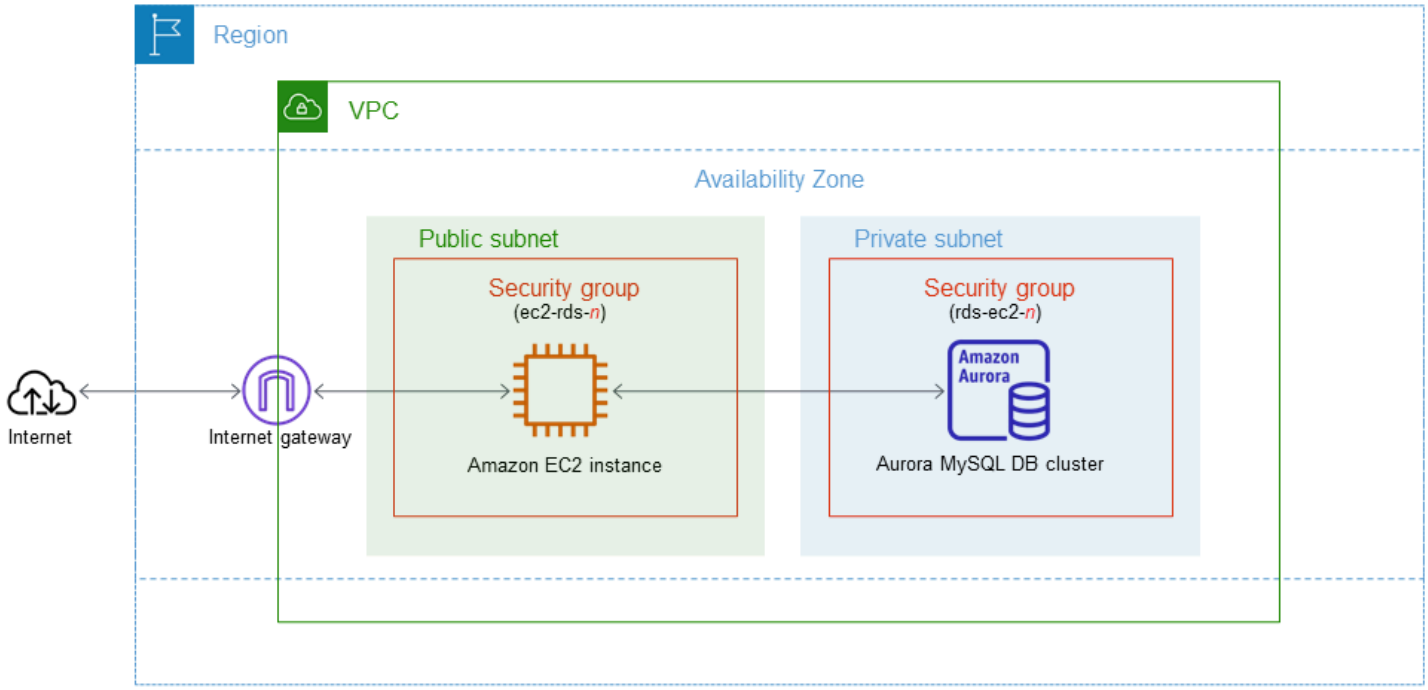
建立 Aurora MySQL 資料庫叢集並與之連線

本教學課程會建立 EC2 執行個體和 Aurora MySQL 資料庫叢集。本教學課程說明如何使用標準 MySQL 用戶端，從 EC2 執行個體存取資料庫叢集。本教學課程為最佳實務，會在虛擬私有雲端 (VPC) 中建立私有資料庫叢集。在多數情況下，相同 VPC 中的其他資源 (例如 EC2 執行個體) 可以存取該資料庫叢集，但 VPC 以外的資源便無法存取該執行個體。

完成教學課程後，在您的 VPC 中，每個可用區域都有一個公有子網路和私有子網路。在可用區域中，EC2 執行個體位於公有子網路中，而資料庫執行個體則可於私有子網路中。

> **Important**
>
> 建立 AWS 帳戶無需付費。不過，透過完成本教學課程，您可能會因為使用 AWS 的資源而產生費用。如果不再需要這些資源，您可以在完成教學課程後刪除這些資源。

下圖顯示此教學課程完成時的組態。



本教學課程可讓您使用下列其中一種方法來建立資源：

1. 使用 AWS 管理主控台 - 步驟 1：建立 EC2 執行個體和 步驟 2：建立 Aurora MySQL 資料庫叢集
2. 使用 CloudFormation 建立資料庫執行個體和 EC2 執行個體 - （選用）使用 建立 VPC、EC2 執行個體和 Aurora MySQL 叢集 CloudFormation

第一個方法使用輕鬆建立，來建立具有 AWS 管理主控台的私有 Aurora MySQL 資料庫叢集。在這裡，您僅能指定資料庫引擎類型、資料庫執行個體大小，與資料庫叢集識別符。輕鬆建立會使用其他組態選項的預設設定。

改用標準建立時，您可以在建立資料庫叢集時指定更多組態選項。這些選項包括可用性、安全性、備份和維護的設定。若要建立公有資料庫叢集，您必須使用標準建立。如需相關資訊，請參閱the section called “建立資料庫叢集”。

主題

- 先決條件
- 步驟 1：建立 EC2 執行個體
- 步驟 2：建立 Aurora MySQL 資料庫叢集
- （選用）使用 建立 VPC、EC2 執行個體和 Aurora MySQL 叢集 CloudFormation
- 步驟 3：連線至 Aurora MySQL 資料庫叢集

- 步驟 4：刪除 EC2 執行個體和資料庫叢集
- (選用) 刪除使用 CloudFormation 建立的 EC2 執行個體和資料庫叢集
- (選用) 將資料庫叢集連線至 Lambda 函數

先決條件

在開始之前，請先完成下節所含步驟：

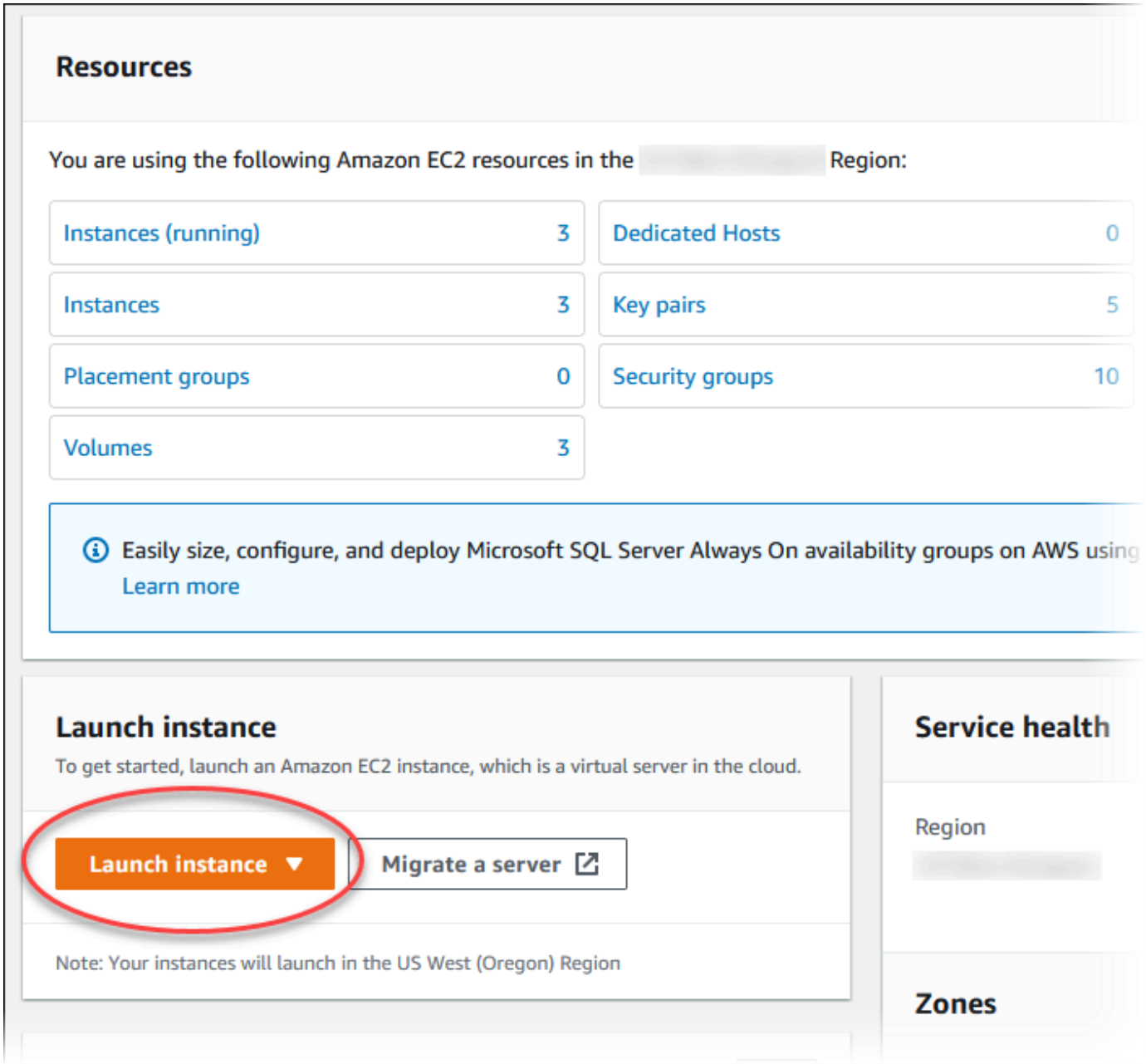
- 註冊 AWS 帳戶
- 建立具有管理存取權的使用者

步驟 1：建立 EC2 執行個體

建立您會用來連線至資料庫的 Amazon EC2 執行個體。

建立 EC2 執行個體

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/ec2/：// 開啟 Amazon EC2 主控台。
2. 在 的右上角 AWS 管理主控台，選擇您要在 AWS 區域 其中建立 EC2 執行個體的 。
3. 選擇 EC2 儀表板，然後選擇啟動執行個體，如下圖所示。

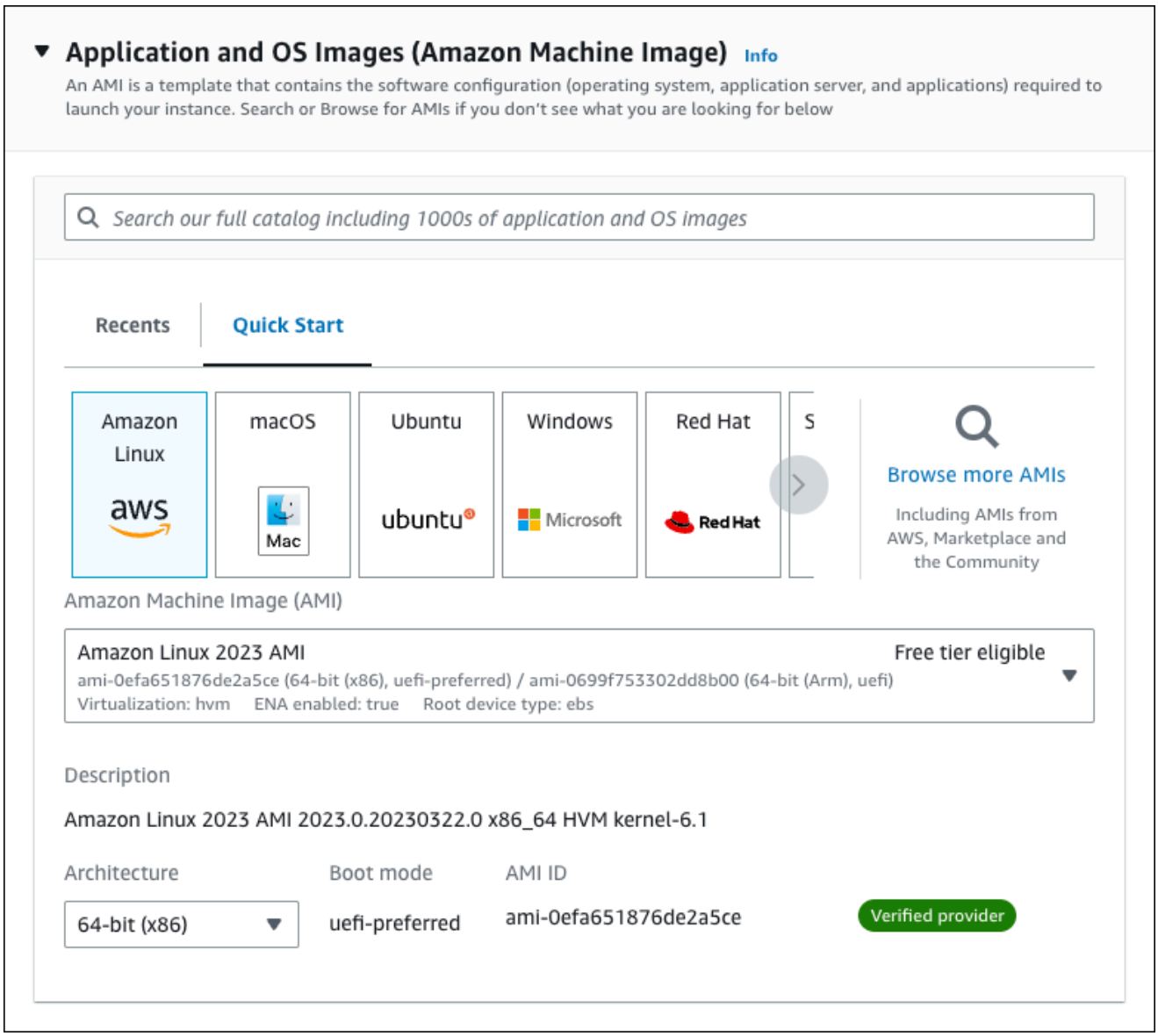


啟動執行個體頁面即開啟。

4. 在啟動執行個體頁面中選擇下列設定。

   a. 在 Name and tags (名稱與標籤) 下，對於 Name (名稱)，輸入 **`ec2-database-connect`**。

   b. 在應用程式和作業系統映像 (Amazon Machine Image) 中，選擇 Amazon Linux，然後選擇 Amazon Linux 2023 AMI。保留其他選項的預設選擇。



c. 在Instance type (執行個體類型) 下，選擇 t2.micro。

d. 在 Key pair (login) (金鑰對 (登入)) 下，選擇 Key pair name (金鑰對名稱)，以使用現有金鑰對。若要為 Amazon EC2 執行個體建立新的金鑰對，請選擇 Create new key pair (建立新的金鑰對)，然後使用 Create key pair (建立金鑰對) 視窗來建立金鑰對。

   如需有關建立新的金鑰對的詳細資訊，請參閱《Amazon EC2 使用者指南》中的建立金鑰對。

e. 對於網路設定中的允許 SSH 流量，選擇 EC2 執行個體的 SSH 連線來源。

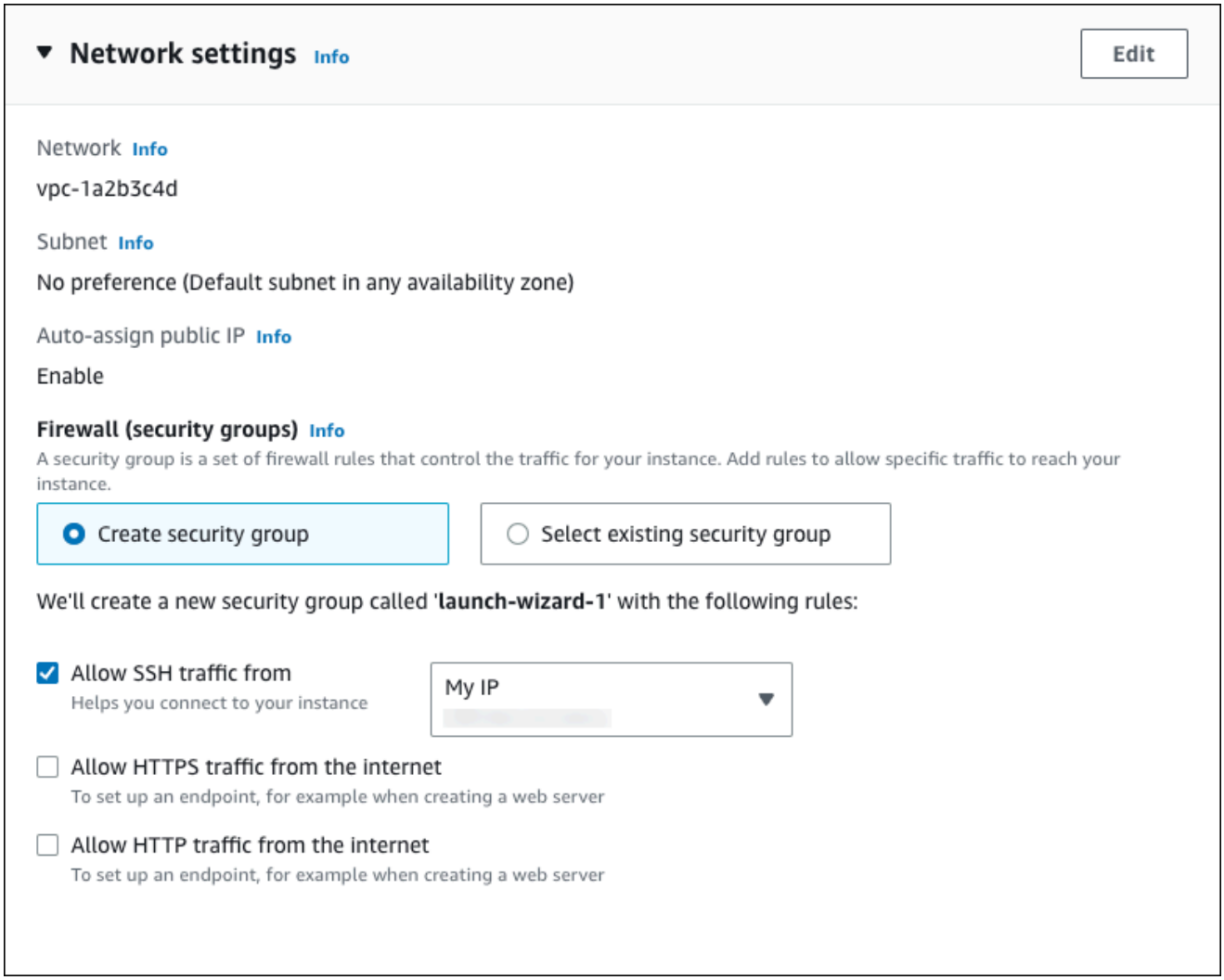
   如果顯示的 IP 地址對 SSH 連線而言是正確的，您可以選擇 My IP (我的 IP)。否則，您可以決定用於使用 Secure Shell (SSH) 連線至 VPC 中 EC2 執行個體的 IP 地址。若要判斷公有

IP 地址，您可以在不同的瀏覽器視窗或索引標籤中使用 https://checkip.amazonaws.com 中的服務。IP 地址的範例為 192.0.2.1/32。

在許多情況下，您可能透過網際網路服務供應商 (ISP) 或是從沒有靜態 IP 地址的防火牆進行連線。若是如此，請務必確定用戶端電腦所使用的 IP 地址範圍。

> **Warning**
>
> 如果您使用 `0.0.0.0/0` 進行 SSH 存取，則可讓所有 IP 地址使用 SSH 存取您的公有 EC2 執行個體。通常在測試環境中短暫使用此方法是沒有問題的，但在生產環境則不安全。在生產環境中，您應只授權特定 IP 地址或特定範圍的地址可使用 SSH 存取您的 EC2 執行個體。

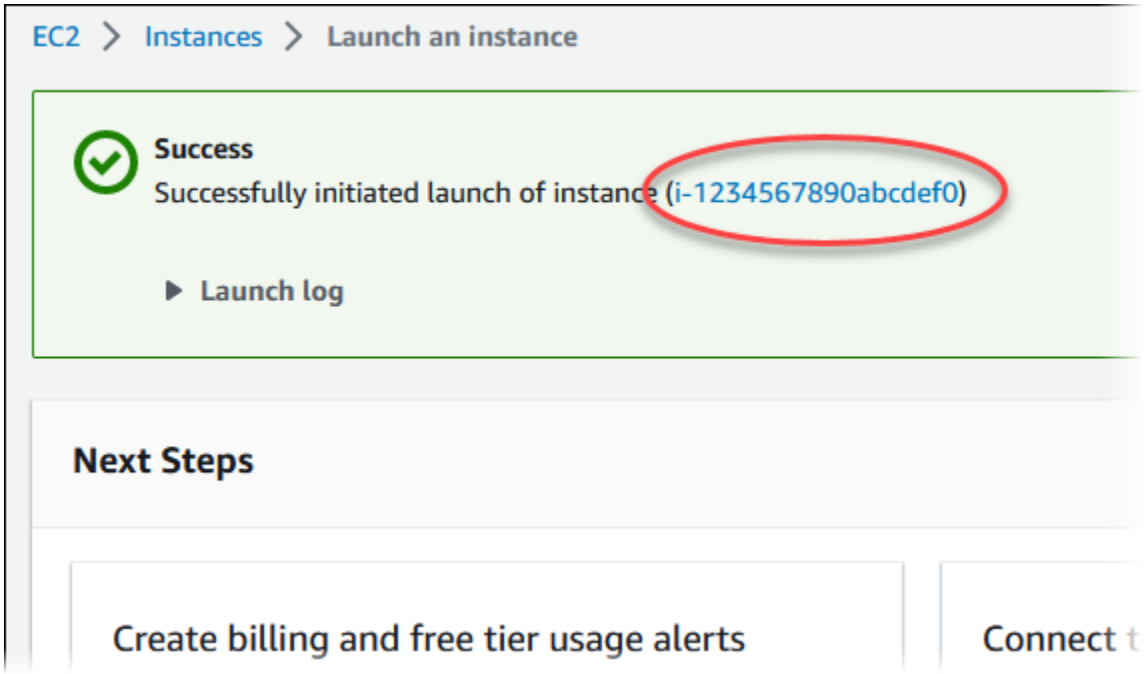
下圖顯示網路設定區段的範例。



f. 讓剩餘區段保留預設值。

g. 檢閱摘要面板中 EC2 執行個體組態的摘要，並在準備就緒時選擇啟動執行個體。

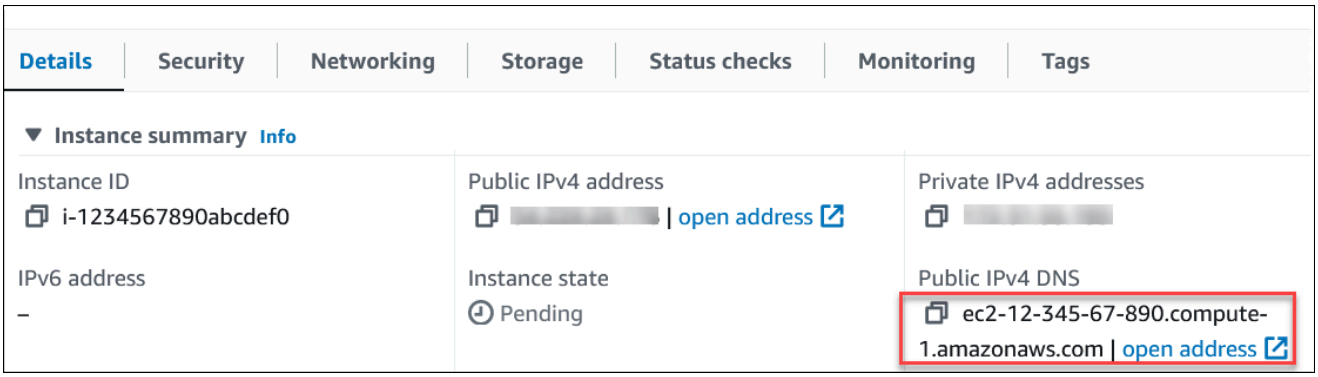
5. 在啟動狀態頁面上，記下新的 EC2 執行個體的識別碼，例如：`i-1234567890abcdef0`。



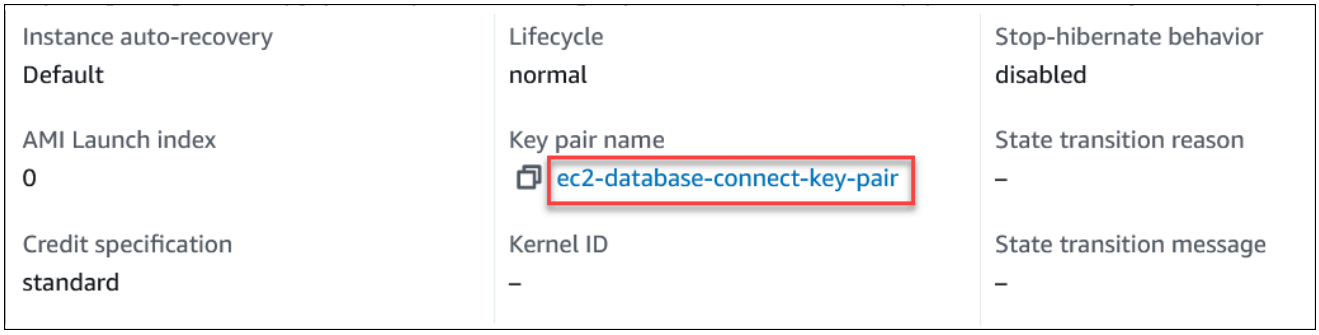
6. 選擇 EC2 執行個體識別符，以開啟 EC2 執行個體清單，然後選取您的 EC2 執行個體。

7. 在詳細資訊索引標籤中，請記下以下值，當您使用 SSH 進行連線時需要這些值：

a. 在執行個體摘要中，記下公用 IPv4 DNS 的值。



b. 在執行個體詳細資訊中，記下金鑰對名稱的值。



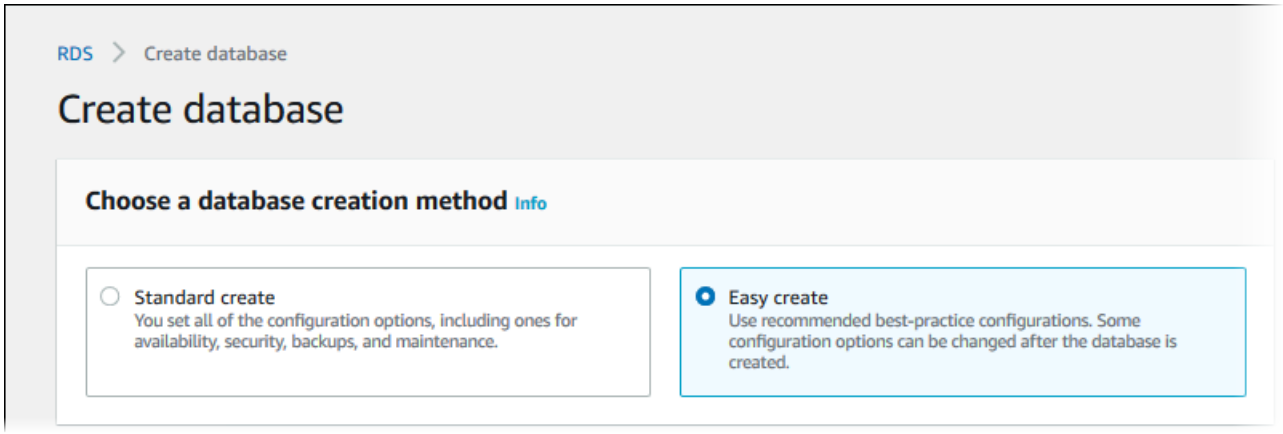
8. 請等待 EC2 執行個體的執行個體狀態為執行中，然後再繼續動作。

步驟 2：建立 Aurora MySQL 資料庫叢集

在此範例中，您使用輕鬆建立來建立 db.r6g.large 資料庫執行個體類別的 Aurora MySQL 資料庫叢集。

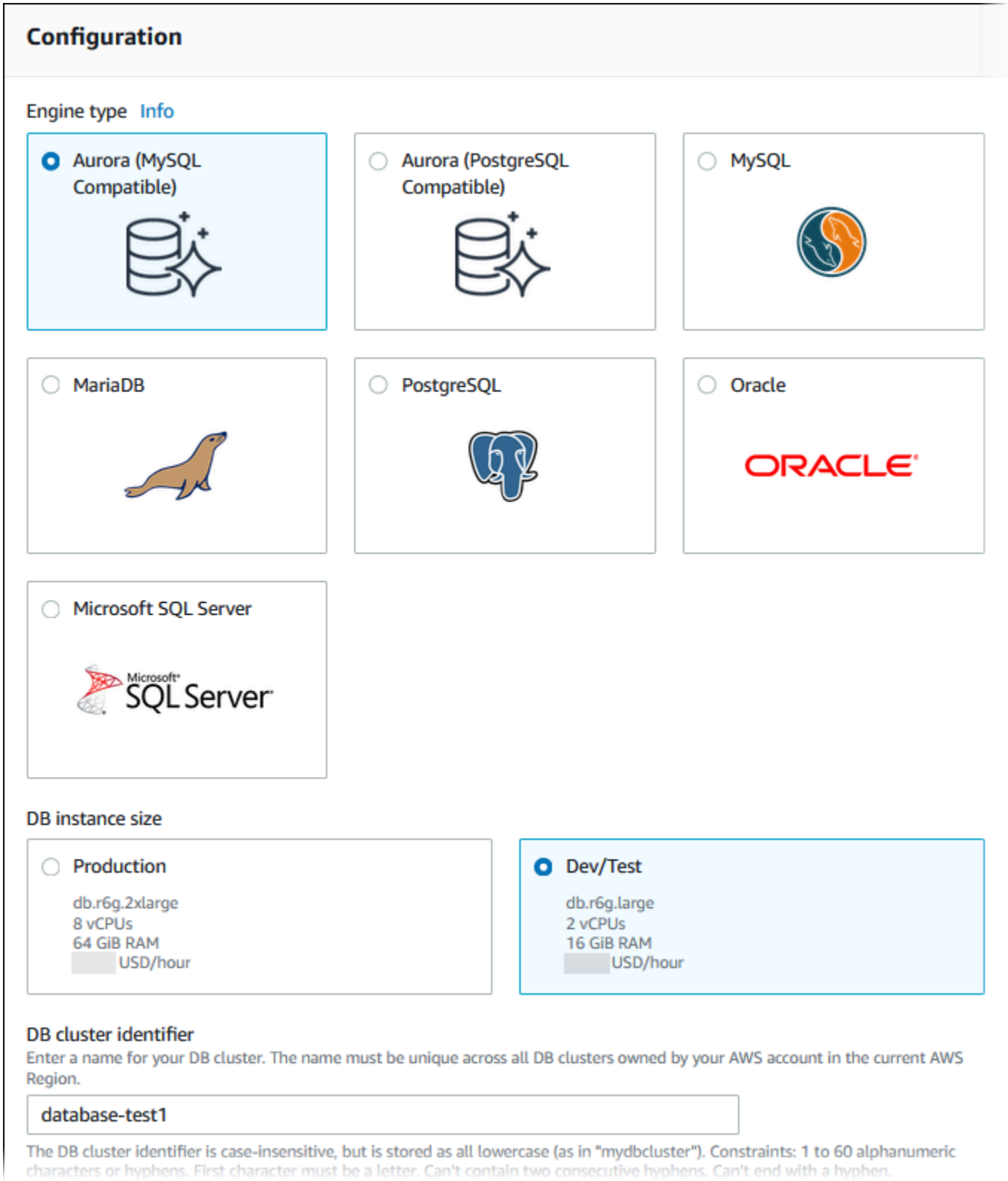
以輕鬆建立來建立 Aurora MySQL 資料庫叢集

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在 Amazon RDS 主控台的右上角，選擇您要在 AWS 區域 其中建立資料庫叢集的 。
3. 在導覽窗格中，選擇 Databases (資料庫)。
4. 選擇 Create database (建立資料庫)，並確定選擇 Easy Create (輕鬆建立)。



5. 在組態中，針對引擎類型選擇 Aurora (MySQL 相容)。
6. 在 DB instance size (資料庫執行個體大小) 中，選擇 Dev/Test (開發/測試)。
7. 針對資料庫叢集識別符輸入 **`database-test1`**。

   Create database (建立資料庫) 頁面看起來應該會如下圖所示。

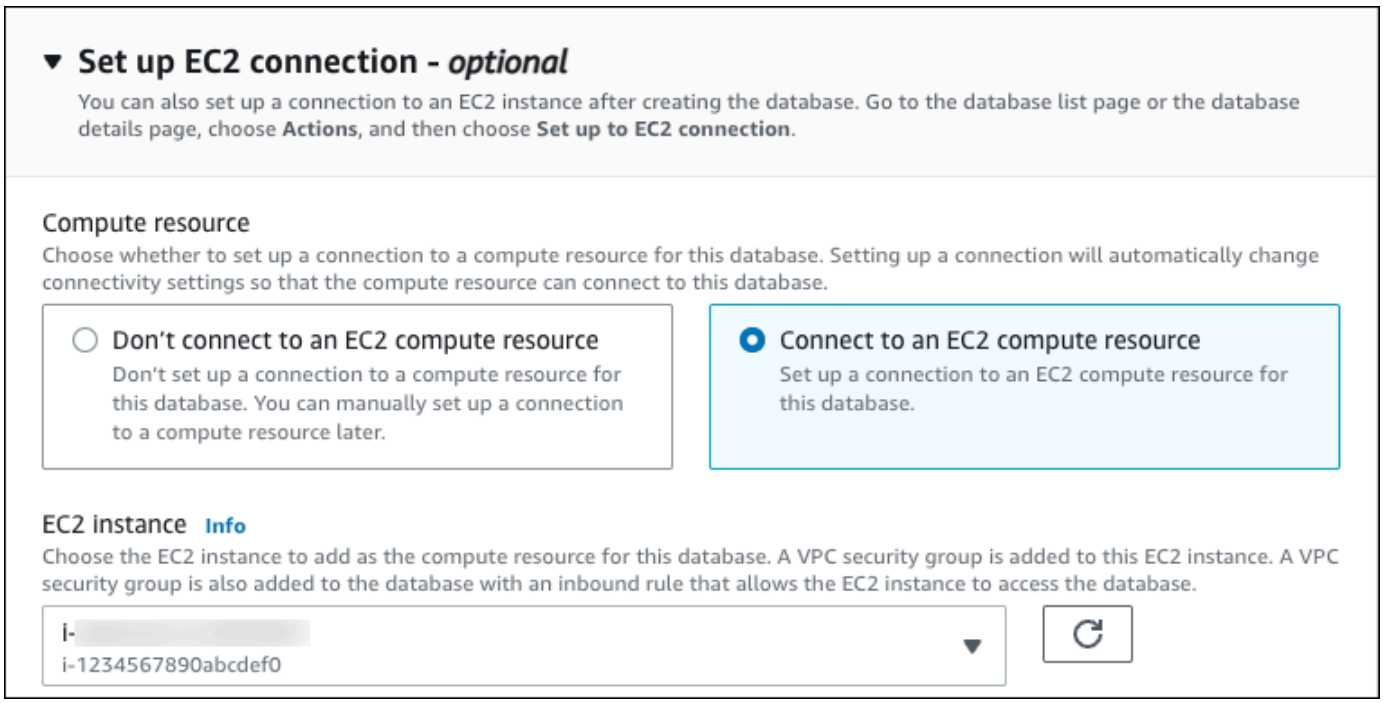


8. 在主要使用者名稱，輸入主要使用者的名稱，或保留預設名稱。
9. 如要將自動產生的主要密碼用在資料庫叢集，請選取自動產生密碼。

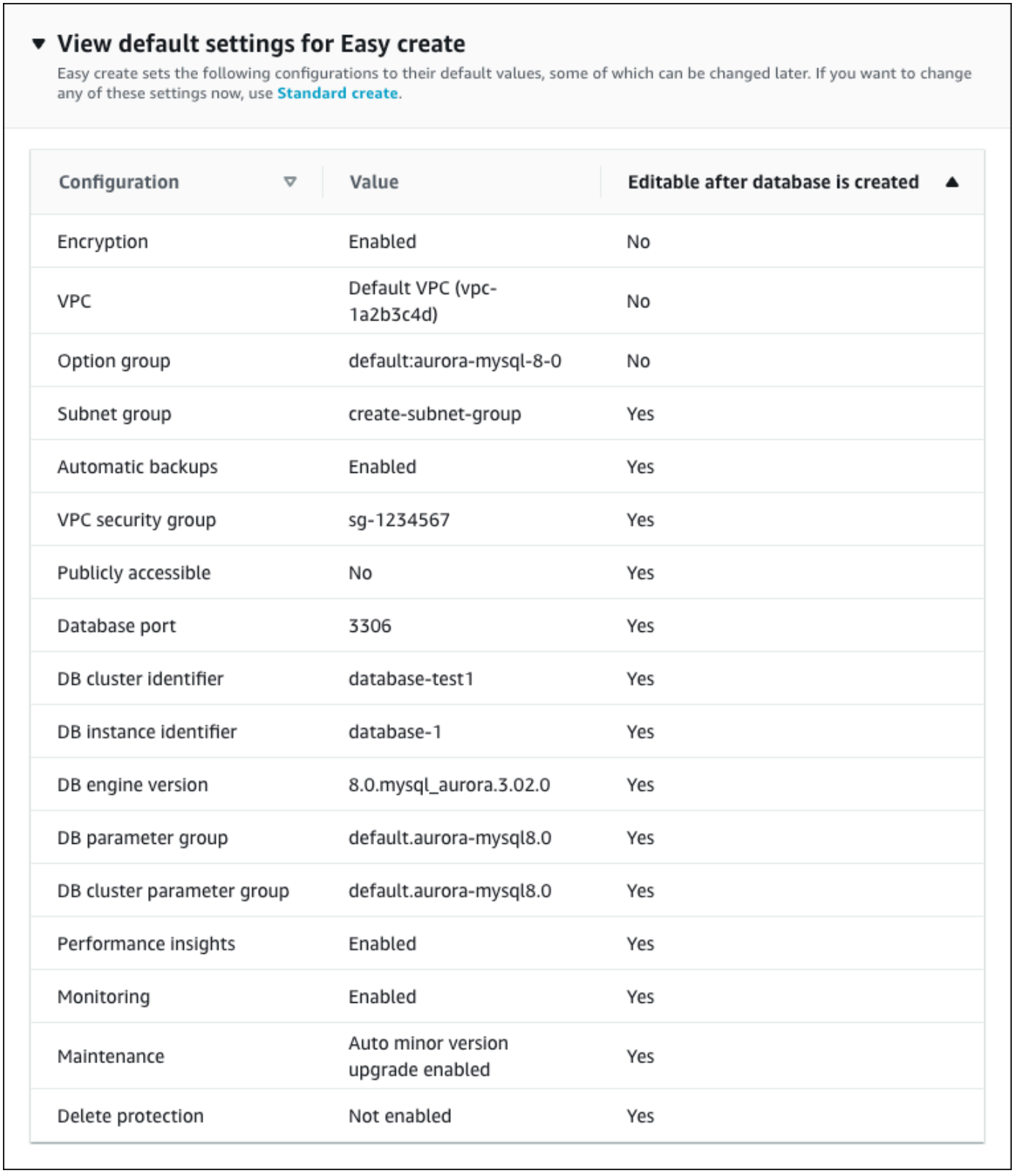
如要輸入您的主要密碼，請確認清除自動產生密碼方塊，然後在主要密碼和確認密碼中輸入相同的密碼。

10. 若要設定與先前建立之 EC2 執行個體的連線，請開啟設定 EC2 連線 – 選用。

    選取連線至 EC2 運算資源。選擇先前建立的 EC2 執行個體。



11. 開啟檢視輕鬆建立的預設設定。



您可以檢查與 Easy Create (輕鬆建立) 一起使用的預設設定。資料庫建立後可編輯欄顯示您可以在資料庫建立後變更的選項。

- 若該欄的設定為否，而您想要其他設定，可以使用標準建立來建立資料庫叢集。
- 若該欄的設定為是，而您想要其他設定，可以使用標準建立來建立資料庫叢集，或在建立後修改該資料庫叢集的設定。

12. 選擇建立資料庫。

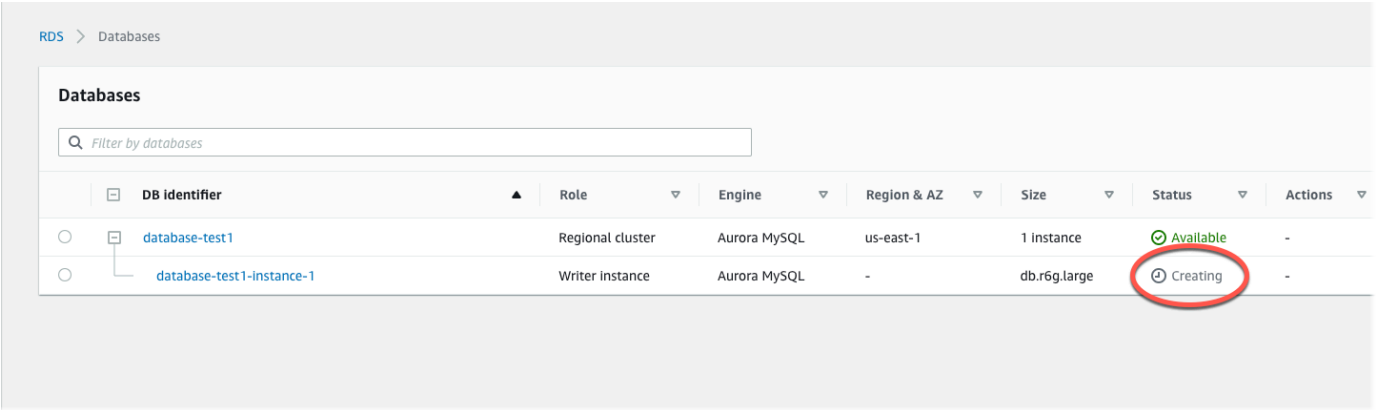
    若要檢視資料庫叢集的主要使用者名稱和密碼，請選擇檢視登入資料詳細資訊。

    您可以使用出現的使用者名稱和密碼，來以主要使用者的身分連線至資料庫叢集。

    > **Important**
    >
    > 您無法再次檢視主要使用者密碼。如果您沒有記錄下來，您可能需要進行變更。
    > 若您需要在資料庫叢集可供使用後變更主要使用者密碼，您可以將資料庫叢集修改為這麼做。如需修改資料庫叢集的詳細資訊，請參閱修改 Amazon Aurora 資料庫叢集。

13. 資料庫清單中，選擇新 Aurora MySQL 資料庫叢集的名稱，以顯示其詳細資訊。

    在資料庫叢集可供使用之前，寫入器執行個體會處於建立中狀態。



    寫入器執行個體狀態變更為可用時，您便能連線至資料庫叢集。視資料庫執行個體類別和儲存體數量而定，可能需要最多 20 分鐘的時間，新的資料庫叢集才會可用。

（選用）使用 建立 VPC、EC2 執行個體和 Aurora MySQL 叢集 CloudFormation

您可以使用 CloudFormation 將基礎設施視為程式碼來佈建 AWS 資源，而不是使用 主控台來建立 VPC、EC2 執行個體和 Aurora MySQL 資料庫叢集。為了協助您將 AWS 資源組織成較小且更易於管

理的單位，您可以使用 CloudFormation 巢狀堆疊功能。如需詳細資訊，請參閱在 CloudFormation 主控台上建立堆疊和使用巢狀堆疊。

> **Important**
>
> CloudFormation 是免費的，但 CloudFormation 建立的資源是即時的。您需要支付這些資源的標準使用費，直到將其終止為止。如需詳細資訊，請參閱 Amazon Aurora 定價。

若要使用 CloudFormation 主控台建立資源，請完成下列步驟：

- 步驟 1：下載 CloudFormation 範本
- 步驟 2：使用 CloudFormation 設定資源

下載 CloudFormation 範本

CloudFormation 範本是 JSON 或 YAML 文字檔案，其中包含您要在堆疊中建立之資源的相關組態資訊。此範本也會為您以及 Aurora 叢集建立 VPC 和堡壘主機。

若要下載範本檔案，請開啟以下連結 Aurora MySQL CloudFormation 範本。

在 Github 頁面中，按一下下載原始檔案按鈕，即可儲存範本 YAML 檔案。

使用 CloudFormation 設定資源

> **Note**
>
> 開始此程序之前，請確定 AWS 帳戶中有 EC2 執行個體的金鑰對。如需詳細資訊，請參閱 Amazon EC2 金鑰對與 Linux 執行個體。

使用 CloudFormation 範本時，您必須選取正確的參數，以確保您的資源已正確建立。請遵循下列步驟：

1. 登入 AWS 管理主控台 並在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。
2. 選擇 Create Stack (建立堆疊)。
3. 在 [指定範本] 區段中，選取從電腦上傳範本檔案，然後選擇下一步。
4. 在指定堆疊詳細資訊頁面，設定下列參數：

a. 將堆疊名稱設為 AurMySQLTestStack。

b. 在參數下，選取兩個可用區域來設定可用區域。

c. 在 Linux 堡壘主機組態下，針對金鑰名稱，選取金鑰對以登入 EC2 執行個體。

d. 在 Linux 堡壘主機組態設定中，將允許的 IP 範圍設定為 IP 位址。若要使用 Secure Shell (SSH) 連線到 VPC 中的 EC2 執行個體，請使用位於 https://checkip.amazonaws.com 的服務來判斷公有 IP 位址。IP 地址的範例為 192.0.2.1/32。

> **Warning**
>
> 如果您使用 `0.0.0.0/0` 進行 SSH 存取，則可讓所有 IP 地址使用 SSH 存取您的公有 EC2 執行個體。通常在測試環境中短暫使用此方法是沒有問題的，但在生產環境則不安全。在生產環境中，您應只授權特定 IP 地址或特定範圍的地址可使用 SSH 存取您的 EC2 執行個體。

e. 在資料庫一般組態下，將資料庫執行個體類別設為 db.r6g.large。

f. 將資料庫名稱設為 **`database-test1`**。

g. 對於資料庫主要使用者名稱，輸入主要使用者的名稱。

h. 在本教學課程中，將使用 Secrets Manager 管理資料庫主要使用者密碼設為 `false`。

i. 針對資料庫密碼，設定您選擇的密碼。請記住此密碼，以供教學課程中的後續步驟使用。

j. 將多可用區域部署設為 `false`。

k. 保留所有其他設定作為預設值。按一下下一步以繼續。

5. 在設定堆疊選項頁面中，保留所有預設選項。按一下下一步以繼續。

6. 在檢閱堆疊頁面中，在檢查資料庫和 Linux 堡壘主機選項後選取提交。

堆疊建立程序完成後，請檢視名稱為 BastionStack 和 AMSNS 的堆疊，以記下連線至資料庫所需的資訊。如需詳細資訊，請參閱在 上檢視 CloudFormation 堆疊資料和資源 AWS 管理主控台。

步驟 3：連線至 Aurora MySQL 資料庫叢集

您可以使用任何標準 SQL 用戶端應用程式來連線至資料庫叢集。在此範例中，您會使用 mysql 命令列用戶端來連線至 Aurora MySQL 資料庫叢集。

連線至 Aurora MySQL 資料庫叢集

1. 為資料庫叢集尋找寫入器執行個體的端點 (DNS 名稱) 和連線埠號碼。

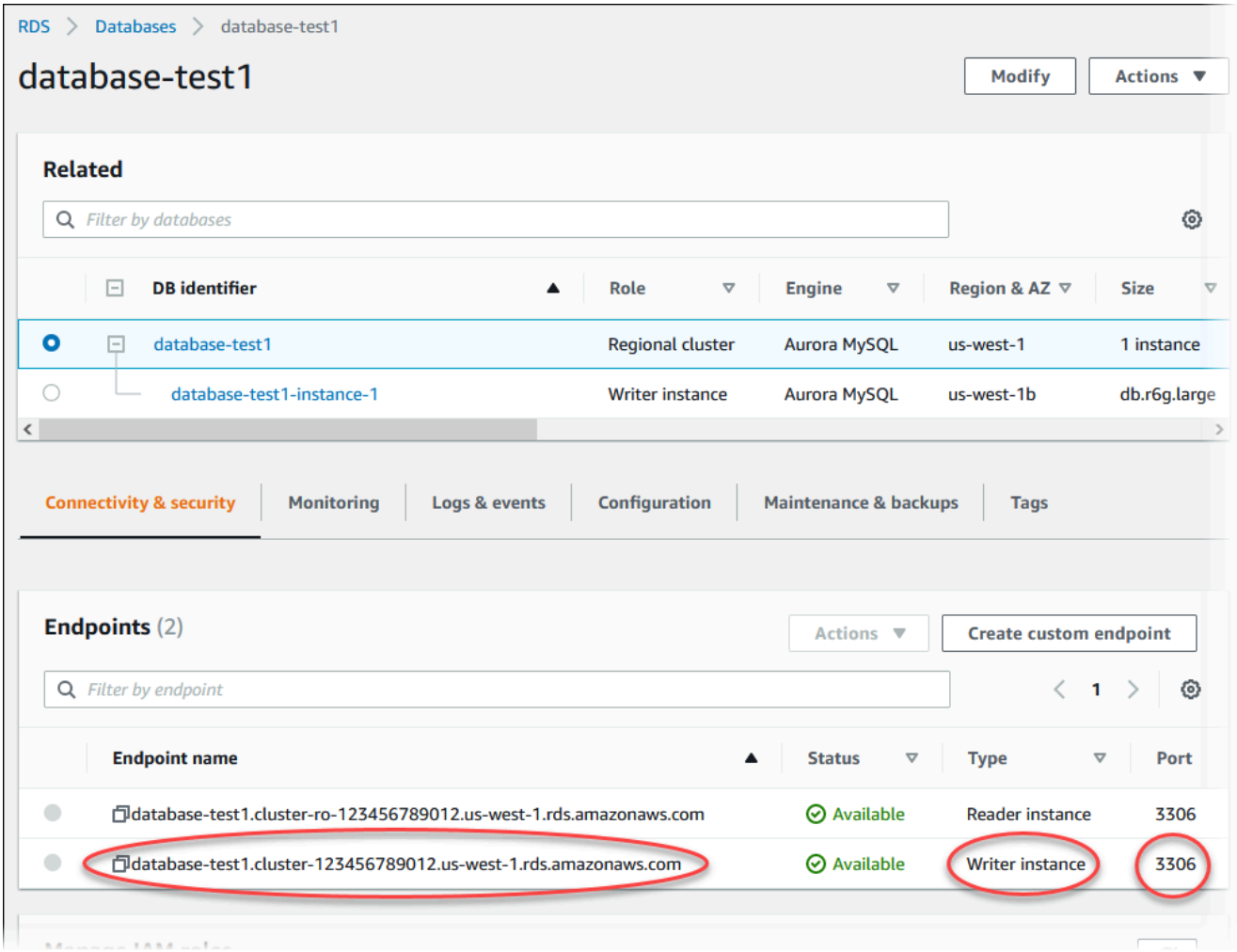
a. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。

b. 在 Amazon RDS 主控台的右上角，選擇資料庫叢集的 AWS 區域 。

c. 在導覽窗格中，選擇 Databases (資料庫)。

d. 選擇一個 Aurora MySQL 資料庫叢集名稱以顯示其詳細資訊。

e. 在連線能力和安全性索引標籤上，複製寫入器執行個體的端點。另外，請記下連線埠號碼。您需要同時有端點和連線埠號碼，才能連線至資料庫叢集。



2. 請遵循《Amazon EC2 使用者指南》中的連線至 Linux 執行個體步驟，以連線您先前所建的 EC2 執行個體。

   建議您使用 SSH 連線至 EC2 執行個體。如果 SSH 用戶端公用程式已安裝在 Windows、Linux 或 Mac 上，您可以使用下列命令格式連線至執行個體：

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

例如，假設 ec2-database-connect-key-pair.pem 存放在 Linux 上 的 /dir1 中，而 EC2 執行個體的公用 IPv4 DNS 為 ec2-12-345-678-90.compute-1.amazonaws.com。然後，您的 SSH 命令看起來如下所示：

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

3. 更新 EC2 執行個體上的軟體，以取得最新的錯誤修正和安全性更新。為此，請使用下列命令。

> **Note**
>
> -y 選項不要求確認就會安裝更新。若要先檢查更新再安裝，請省略此選項。

```
sudo dnf update -y
```

4. 若要在 Amazon Linux 2023 上安裝 MariaDB 的 mysql 命令列用戶端，請執行下列命令：

```
sudo dnf install mariadb105
```

5. 連線至 Aurora MySQL 資料庫叢集。例如，輸入下列命令。此動作可讓您使用 MySQL 用戶端，連線至 Aurora MySQL 資料庫叢集。

   取代 *endpoint* 的寫入器執行個體端點，以及取代您用於 *admin* 的主要使用者名稱。提示您輸入密碼時，請提供您使用的主要密碼。

```
mysql -h endpoint -P 3306 -u admin -p
```

   輸入使用者的密碼之後，您應該會看到類似如下的輸出。

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 217
Server version: 8.0.23 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
MySQL [(none)]>
```

如需連線至 Aurora MySQL 資料庫叢集的詳細資訊，請參閱 連接至 Amazon Aurora MySQL 資料庫叢集。若您無法連線至資料庫叢集，請參閱 無法連線至 Amazon RDS 資料庫執行個體。

基於安全考量，最佳做法是使用加密連線。僅當用戶端和伺服器位於同一 VPC 且網路受信任時，才使用未加密的 MySQL 連線。如需使用加密連線的詳細資訊，請參閱 使用 SSL 連線至 Aurora MySQL。

6. 執行 SQL 命令。

   例如，下列 SQL 命令會顯示目前的日期和時間：

```
SELECT CURRENT_TIMESTAMP;
```

步驟 4：刪除 EC2 執行個體和資料庫叢集

在連線至您已建立的範例 EC2 執行個體與資料庫叢集，並探索這些執行個體之後，請將其刪除，才不會再對您收費。

如果您使用 CloudFormation 建立資源，請略過此步驟並前往下一個步驟。

刪除 EC2 執行個體

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/ec2/：// 開啟 Amazon EC2 主控台。
2. 在導覽窗格中，選擇執行個體。
3. 選取 EC2 執行個體，並選取執行個體狀態、終止執行個體。
4. 出現確認提示時，請選擇 Terminate (終止)。

如需刪除 EC2 執行個體的詳細資訊，請參閱《Amazon EC2 使用者指南》中的終止執行個體。

刪除資料庫叢集

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 選擇 Databases (資料庫)，然後選擇與該資料庫叢集相關聯的資料庫執行個體。

3. 對於 Actions (動作)，請選擇 Delete (刪除)。
4. 清除建立最終快照？。
5. 完成確認，然後選擇刪除。

在刪除與資料庫叢集相關聯的所有資料庫執行個體後，就會自動刪除該資料庫叢集。

(選用) 刪除使用 CloudFormation 建立的 EC2 執行個體和資料庫叢集

如果您使用 CloudFormation 來建立資源，請在連線至 後刪除 CloudFormation 堆疊，並探索範例 EC2 執行個體和資料庫叢集，因此您不再需要支付這些堆疊的費用。

刪除 CloudFormation 資源

1. 開啟 CloudFormation 主控台。
2. 在 CloudFormation 主控台的堆疊頁面，選取根堆疊 (不含名稱 VPCStack、BastionStack 或 AMSNS 的堆疊)。
3. 選擇 刪除。
4. 出現確認提示時，請選取刪除堆疊。

如需在 CloudFormation 中刪除堆疊的詳細資訊，請參閱AWS CloudFormation 《 使用者指南》中的刪除 CloudFormation 主控台上的堆疊。

(選用) 將資料庫叢集連線至 Lambda 函數

您也可以將 Aurora MySQL 資料庫叢集連線至 Lambda 無伺服器運算資源。Lambda 函數允許您在沒有佈建或管理基礎設施的情況下執行程式碼。Lambda 函數還允許您自動回應任何規模的程式碼執行請求，從每天十幾個事件到每秒數百個事件。如需更多詳細資訊，請參閱 自動連線 Lambda 函數和 Aurora 資料庫叢集。

建立 Aurora PostgreSQL 資料庫叢集並與之連線

有兩種方式可以建立 Aurora PostgreSQL 叢集：完整組態和快速組態。請參閱以下各節，進一步了解這 2 種類型的 建立。

主題

- 使用完整組態建立
- 使用快速組態建立

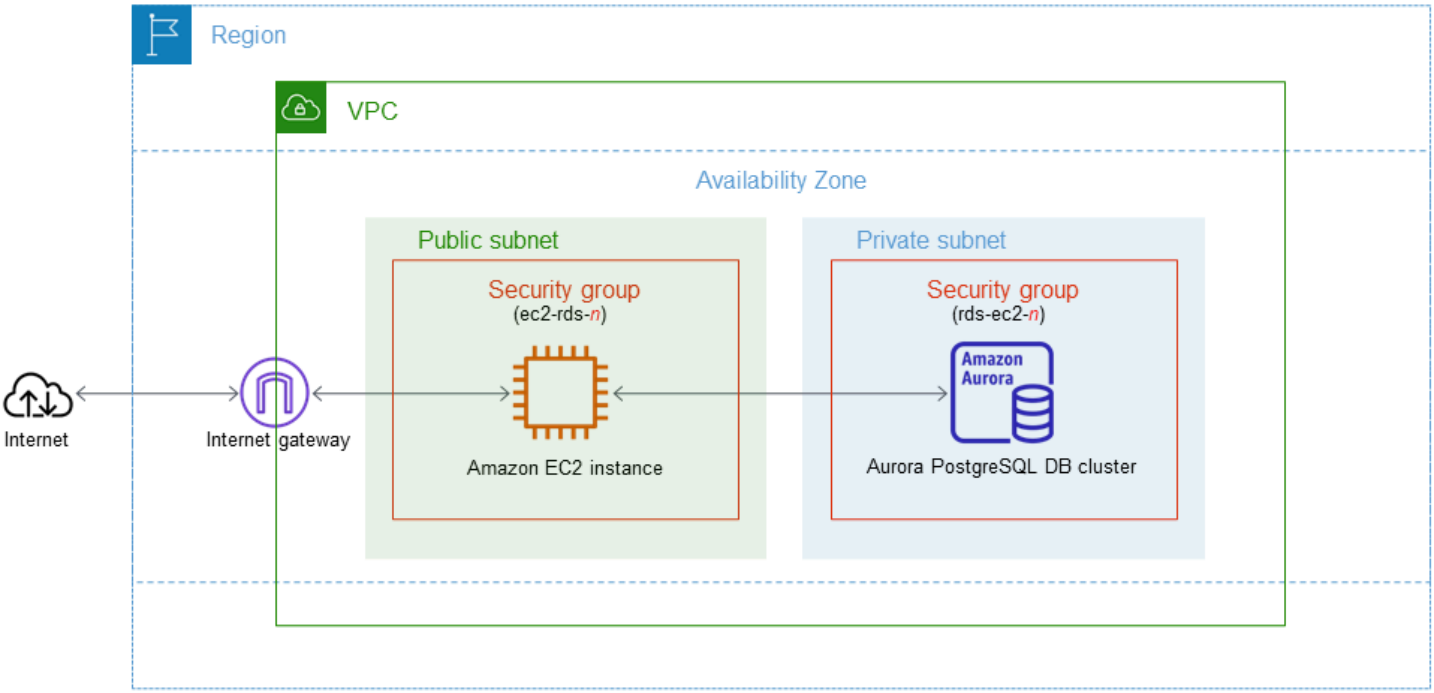
使用完整組態建立

本教學課程會建立 EC2 執行個體和 Aurora PostgreSQL 資料庫叢集。本教學課程說明如何使用標準 PostgreSQL 用戶端，從 EC2 執行個體存取資料庫叢集。本教學課程為最佳實務，會在虛擬私有雲端 (VPC) 中建立私有資料庫叢集。在多數情況下，相同 VPC 中的其他資源 (例如 EC2 執行個體) 可以存取該資料庫叢集，但 VPC 以外的資源便無法存取該執行個體。

完成教學課程後，在您的 VPC 中，每個可用區域都有一個公有子網路和私有子網路。在可用區域中，EC2 執行個體位於公有子網路中，而資料庫執行個體則可於私有子網路中。

> **Important**
>
> 建立 AWS 帳戶無需付費。不過，透過完成本教學課程，您可能會為您使用 AWS 的資源產生費用。如果不再需要這些資源，您可以在完成教學課程後刪除這些資源。

下圖顯示此教學課程完成時的組態。



本教學課程可讓您使用下列其中一種方法來建立資源：

1. 使用 AWS 管理主控台 - 步驟 1：建立 EC2 執行個體和 步驟 2：建立 Aurora PostgreSQL 資料庫叢集

2. 使用 CloudFormation 建立資料庫執行個體和 EC2 執行個體 - （選用） 使用 建立 VPC、EC2 執行個體和 Aurora PostgreSQL 叢集 CloudFormation

第一個方法使用輕鬆建立，來建立具有 AWS 管理主控台的私有 Aurora PostgreSQL 資料庫叢集。在這裡，您僅能指定資料庫引擎類型、資料庫執行個體大小，與資料庫叢集識別符。輕鬆建立會使用其他組態選項的預設設定。

改用標準建立時，您可以在建立資料庫叢集時指定更多組態選項。這些選項包括可用性、安全性、備份和維護的設定。若要建立公有資料庫叢集，您必須使用標準建立。如需相關資訊，請參閱the section called “建立資料庫叢集”。

主題

- 先決條件
- 步驟 1：建立 EC2 執行個體
- 步驟 2：建立 Aurora PostgreSQL 資料庫叢集
- （選用） 使用 建立 VPC、EC2 執行個體和 Aurora PostgreSQL 叢集 CloudFormation
- 步驟 3：連線至 Aurora PostgreSQL 資料庫叢集
- 步驟 4：刪除 EC2 執行個體和資料庫叢集
- (選用) 刪除使用 CloudFormation 建立的 EC2 執行個體和資料庫叢集
- (選用) 將資料庫叢集連線至 Lambda 函數

先決條件

在開始之前，請先完成下節所含步驟：

- 註冊 AWS 帳戶
- 建立具有管理存取權的使用者

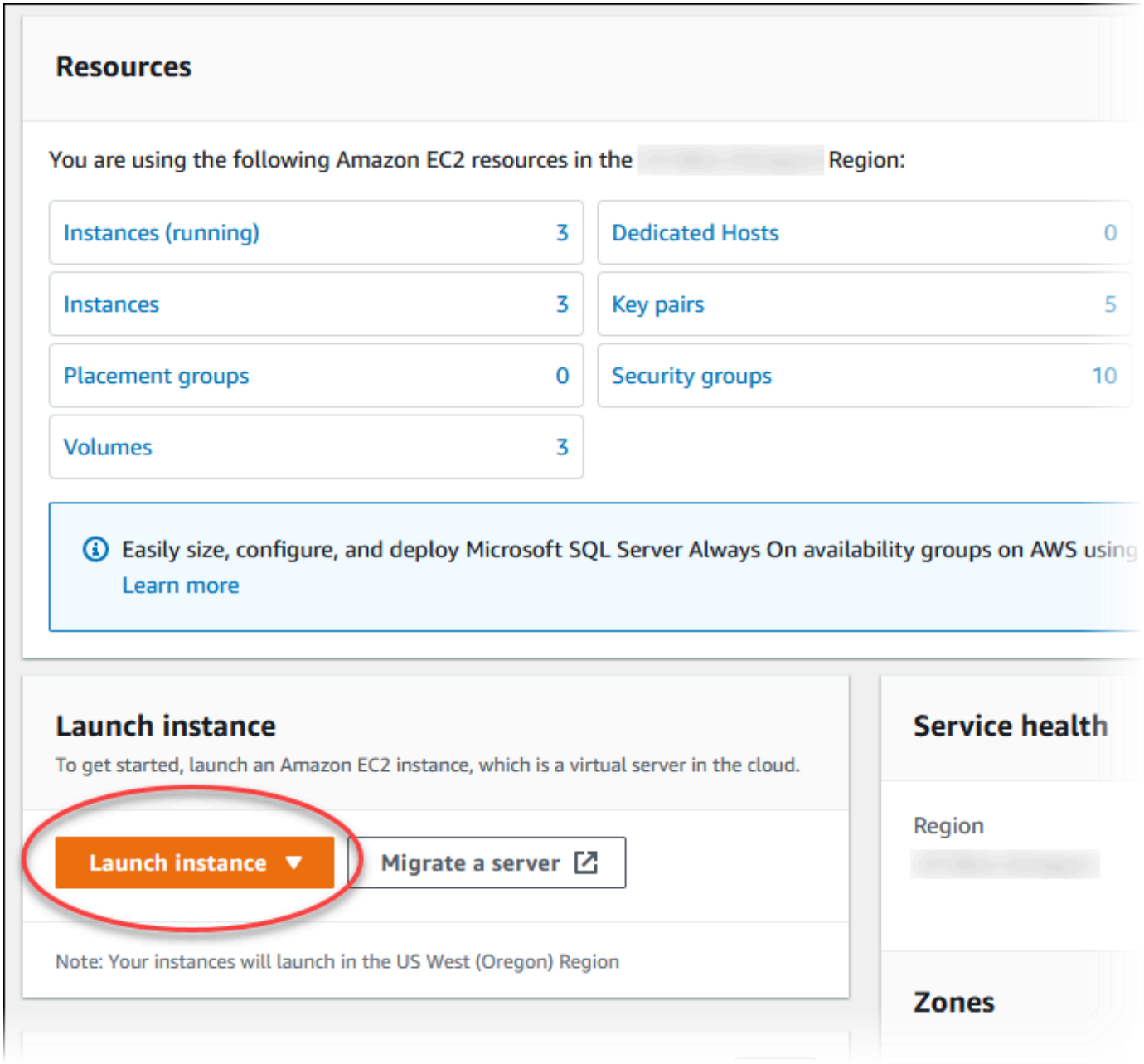
步驟 1：建立 EC2 執行個體

建立您會用來連線至資料庫的 Amazon EC2 執行個體。

建立 EC2 執行個體

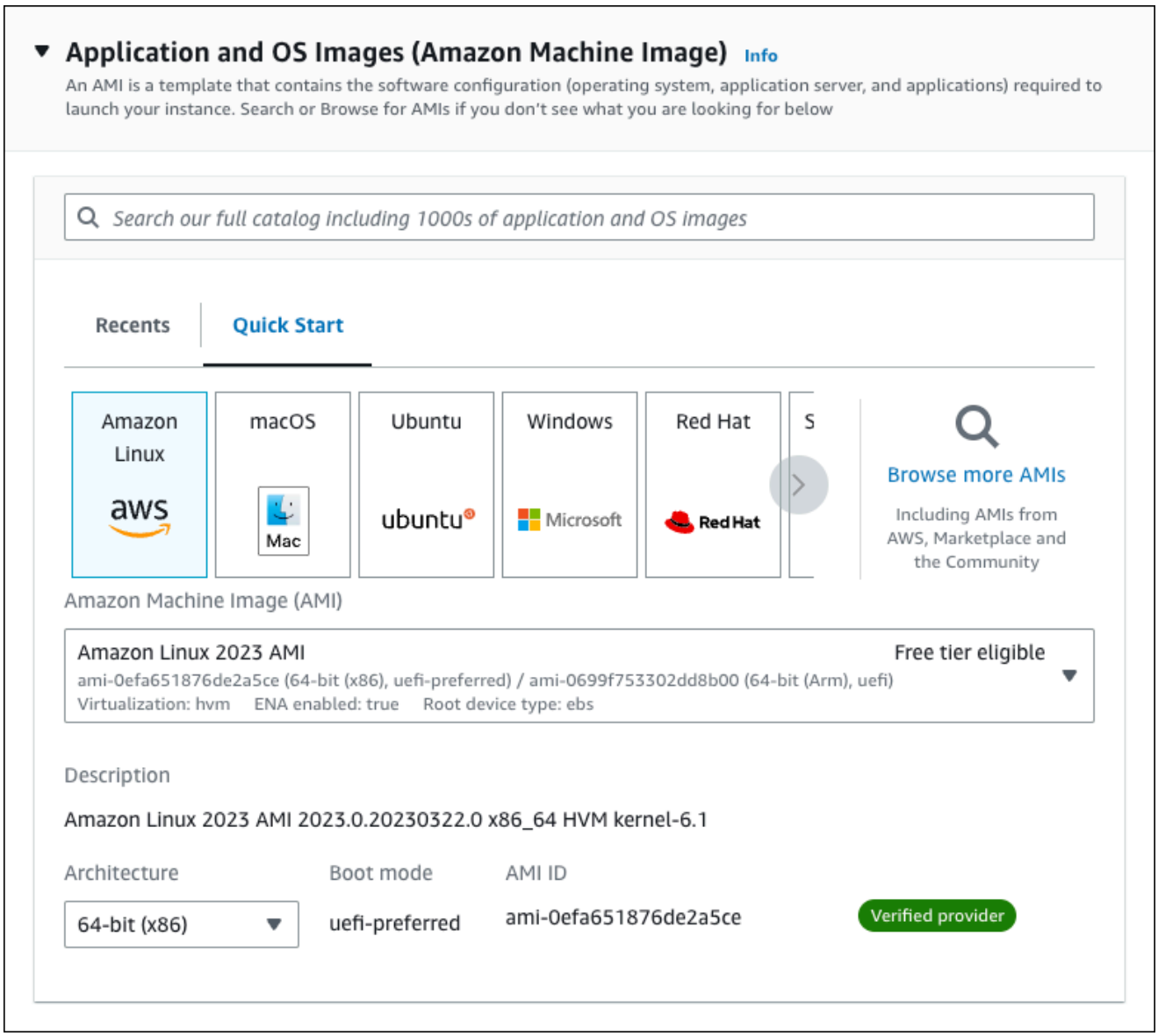
1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/ec2/：// 開啟 Amazon EC2 主控台。

2. 在 的右上角 AWS 管理主控台，選擇您要在 AWS 區域 其中建立 EC2 執行個體的 。
3. 選擇 EC2 儀表板，然後選擇啟動執行個體，如下圖所示。



啟動執行個體頁面即開啟。

4. 在啟動執行個體頁面中選擇下列設定。
   a. 在 Name and tags (名稱與標籤) 下，對於 Name (名稱)，輸入 **`ec2-database-connect`**。
   b. 在應用程式和作業系統映像 (Amazon Machine Image) 中，選擇 Amazon Linux，然後選擇 Amazon Linux 2023 AMI。保留其他選項的預設選擇。



c. 在Instance type (執行個體類型) 下，選擇 t2.micro。

d. 在 Key pair (login) (金鑰對 (登入)) 下，選擇 Key pair name (金鑰對名稱)，以使用現有金鑰對。若要為 Amazon EC2 執行個體建立新的金鑰對，請選擇 Create new key pair (建立新的金鑰對)，然後使用 Create key pair (建立金鑰對) 視窗來建立金鑰對。

   如需有關建立新的金鑰對的詳細資訊，請參閱《Amazon EC2 使用者指南》中的建立金鑰對。

e. 對於網路設定中的允許 SSH 流量，選擇 EC2 執行個體的 SSH 連線來源。

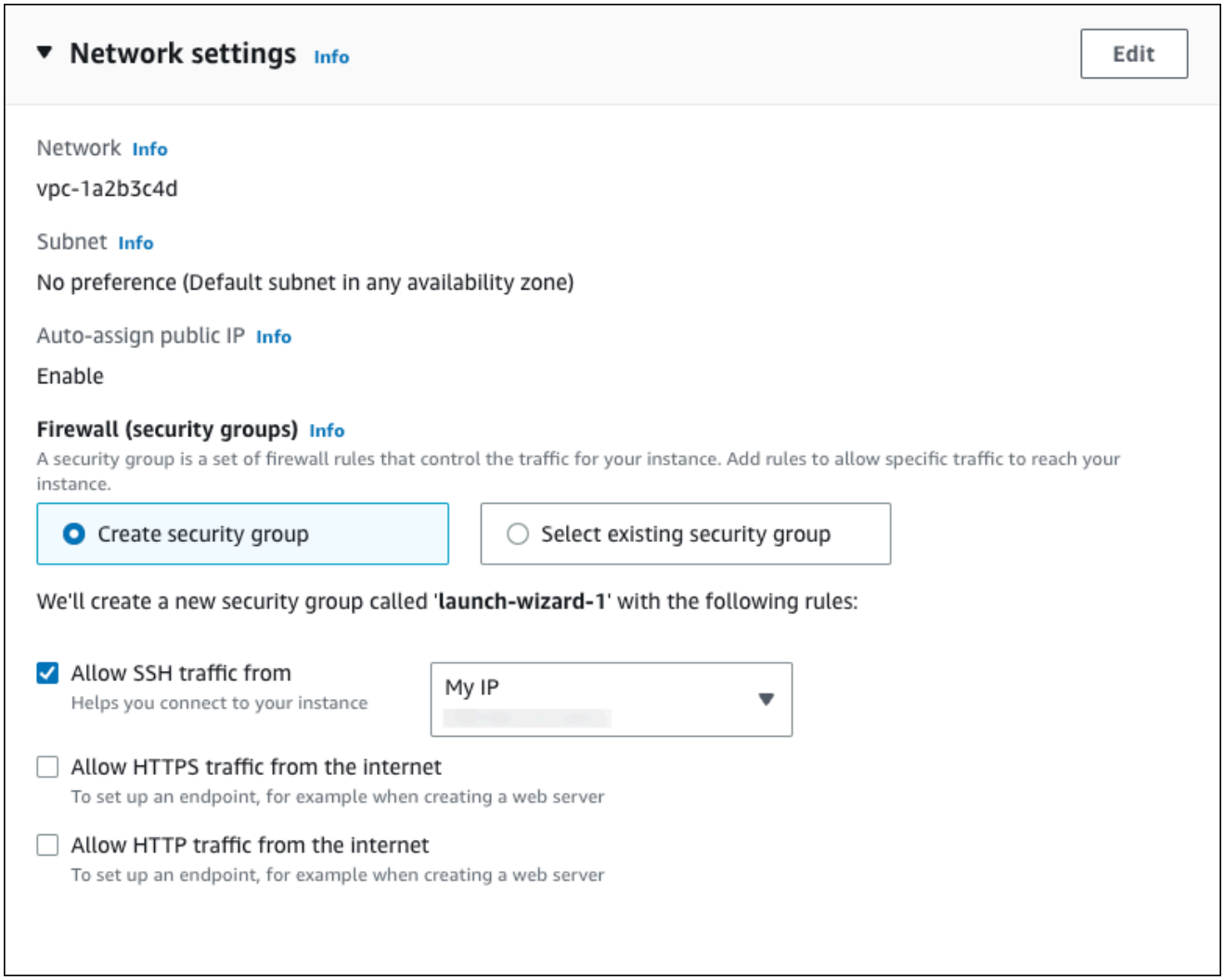
   如果顯示的 IP 地址對 SSH 連線而言是正確的，您可以選擇 My IP (我的 IP)。否則，您可以決定用於使用 Secure Shell (SSH) 連線至 VPC 中 EC2 執行個體的 IP 地址。若要判斷公有

IP 地址，您可以在不同的瀏覽器視窗或索引標籤中使用 https://checkip.amazonaws.com 中的服務。IP 地址的範例為 192.0.2.1/32。

在許多情況下，您可能透過網際網路服務供應商 (ISP) 或是從沒有靜態 IP 地址的防火牆進行連線。若是如此，請務必確定用戶端電腦所使用的 IP 地址範圍。

> **Warning**
>
> 如果您使用 `0.0.0.0/0` 進行 SSH 存取，則可讓所有 IP 地址使用 SSH 存取您的公有 EC2 執行個體。通常在測試環境中短暫使用此方法是沒有問題的，但在生產環境則不安全。在生產環境中，您應只授權特定 IP 地址或特定範圍的地址可使用 SSH 存取您的 EC2 執行個體。

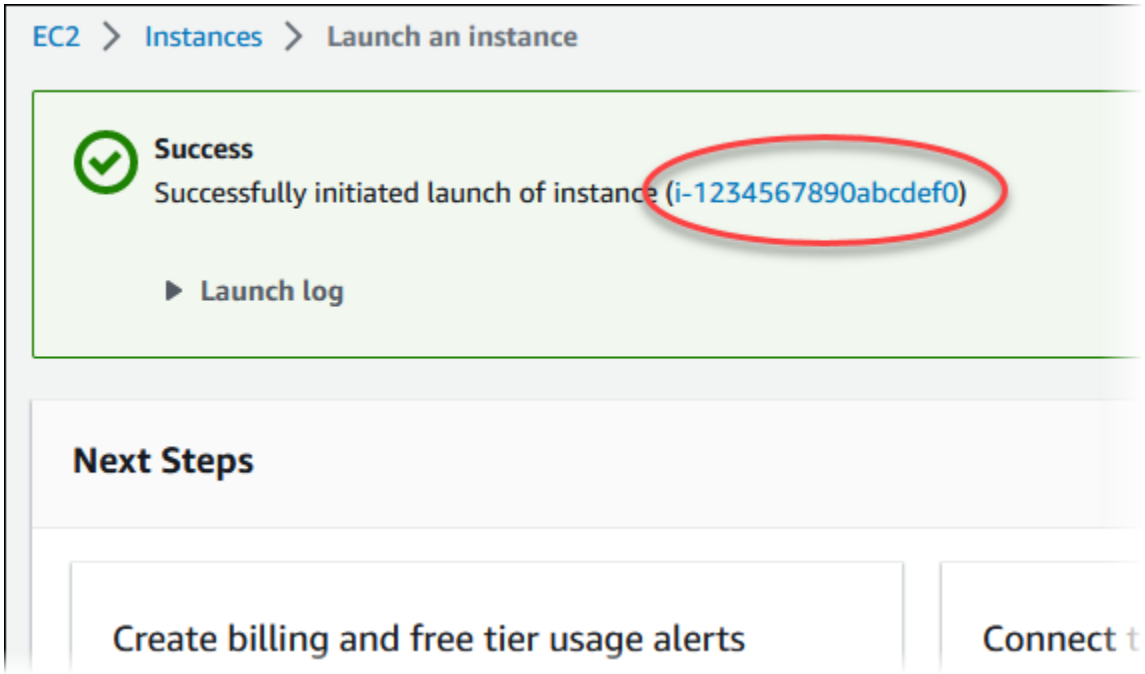
下圖顯示網路設定區段的範例。



f. 讓剩餘區段保留預設值。

g. 檢閱摘要面板中 EC2 執行個體組態的摘要，並在準備就緒時選擇啟動執行個體。

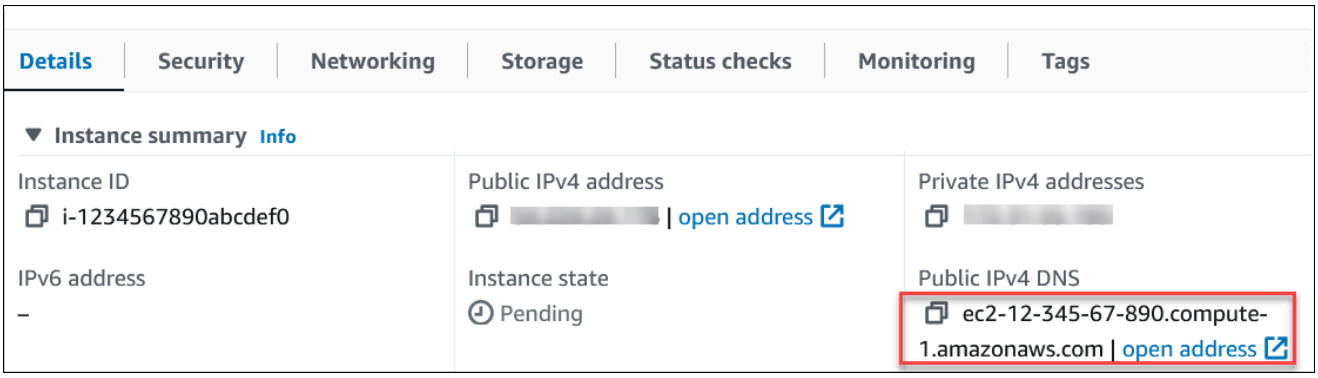
5. 在啟動狀態頁面上，記下新的 EC2 執行個體的識別碼，例如：`i-1234567890abcdef0`。



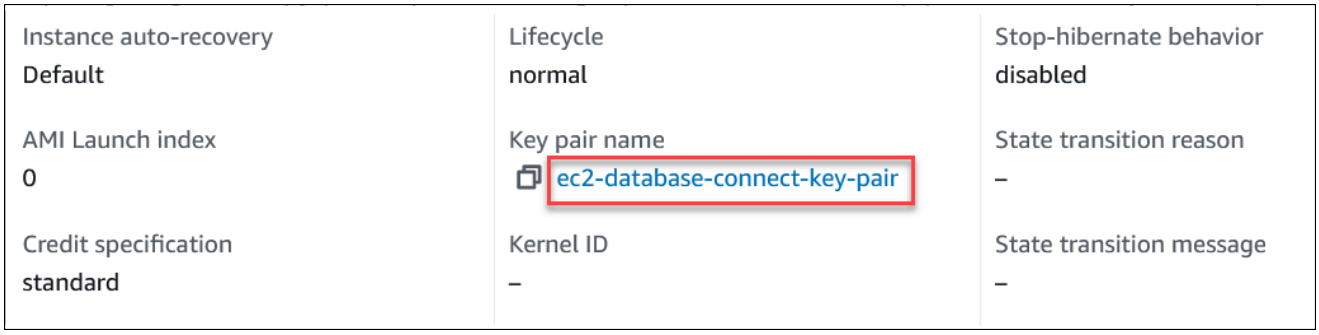
6. 選擇 EC2 執行個體識別符，以開啟 EC2 執行個體清單，然後選取您的 EC2 執行個體。

7. 在詳細資訊索引標籤中，請記下以下值，當您使用 SSH 進行連線時需要這些值：

a. 在執行個體摘要中，記下公用 IPv4 DNS 的值。



b. 在執行個體詳細資訊中，記下金鑰對名稱的值。



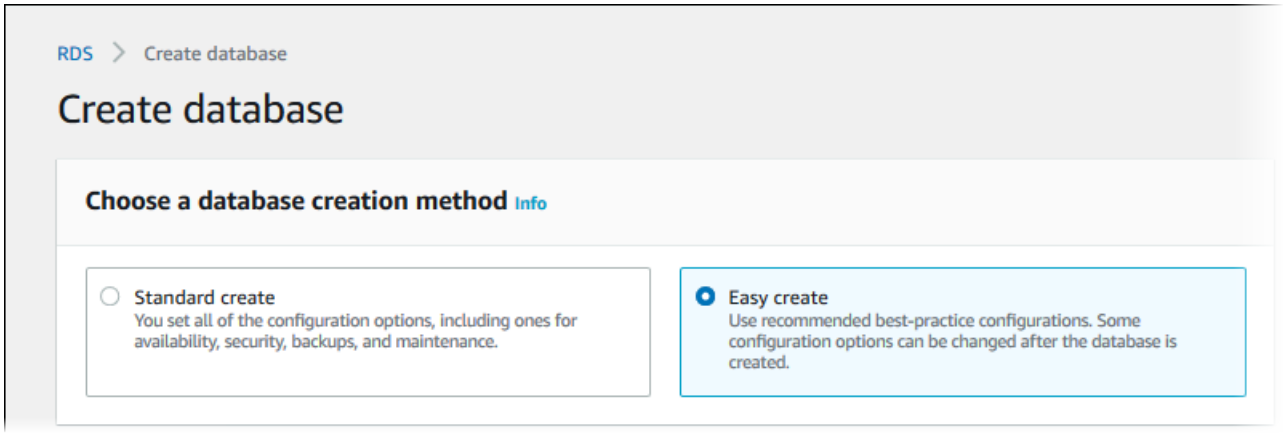
8. 請等待 EC2 執行個體的執行個體狀態為執行中，然後再繼續動作。

步驟 2：建立 Aurora PostgreSQL 資料庫叢集

在此範例中，您使用輕鬆建立來建立 db.t4g.large 資料庫執行個體類別的 Aurora PostgreSQL 資料庫叢集。

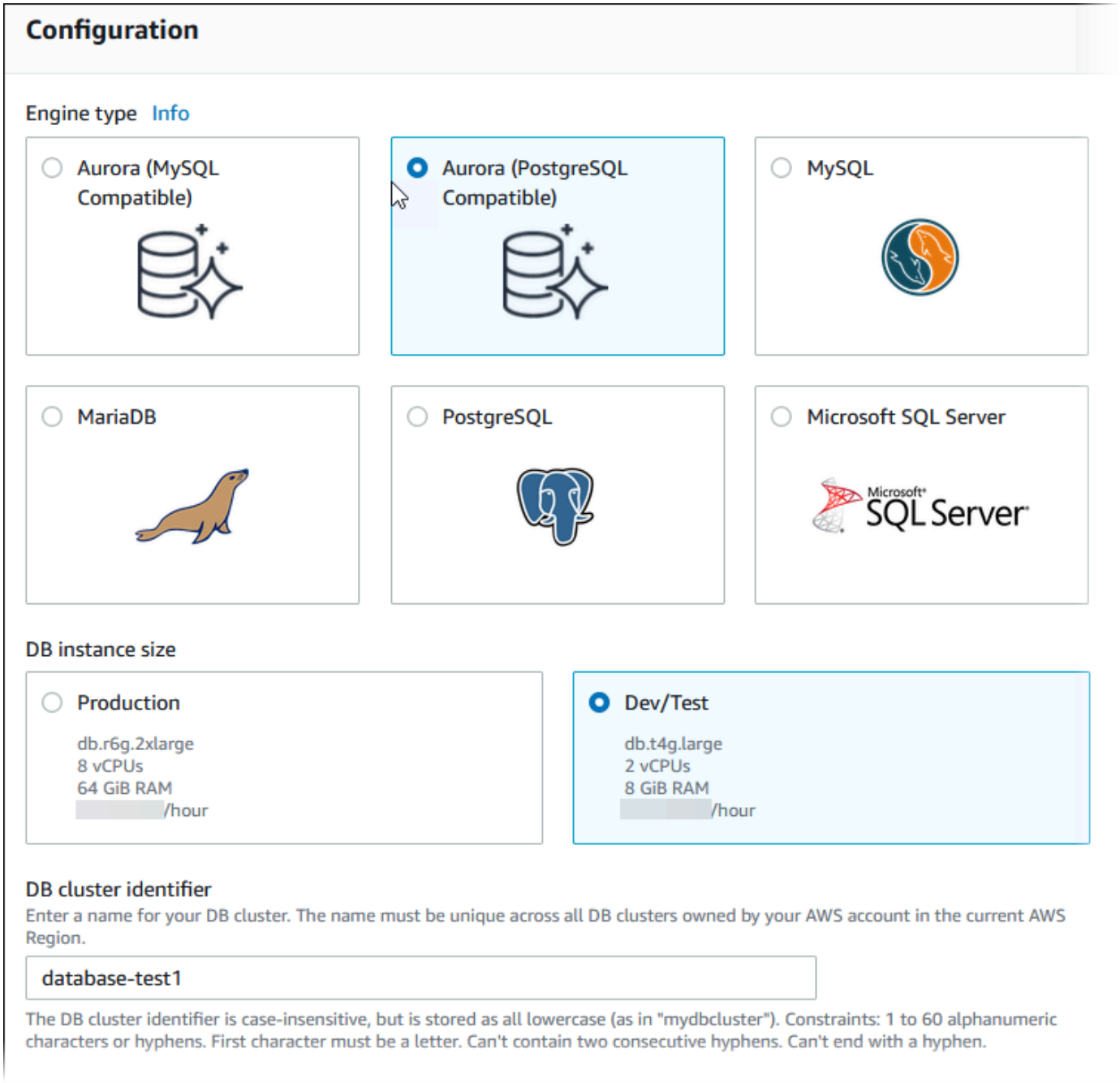
以輕鬆建立來建立 Aurora PostgreSQL 資料庫叢集

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在 Amazon RDS 主控台的右上角，選擇要建立資料庫叢集的 AWS 區域 。
3. 在導覽窗格中，選擇 Databases (資料庫)。
4. 選擇建立資料庫，並確定選擇輕鬆建立。



5. 在組態中，針對引擎類型選擇 Aurora (PostgreSQL 相容)。
6. 在 DB instance size (資料庫執行個體大小) 中，選擇 Dev/Test (開發/測試)。
7. 針對資料庫叢集識別符輸入 **`database-test1`**。

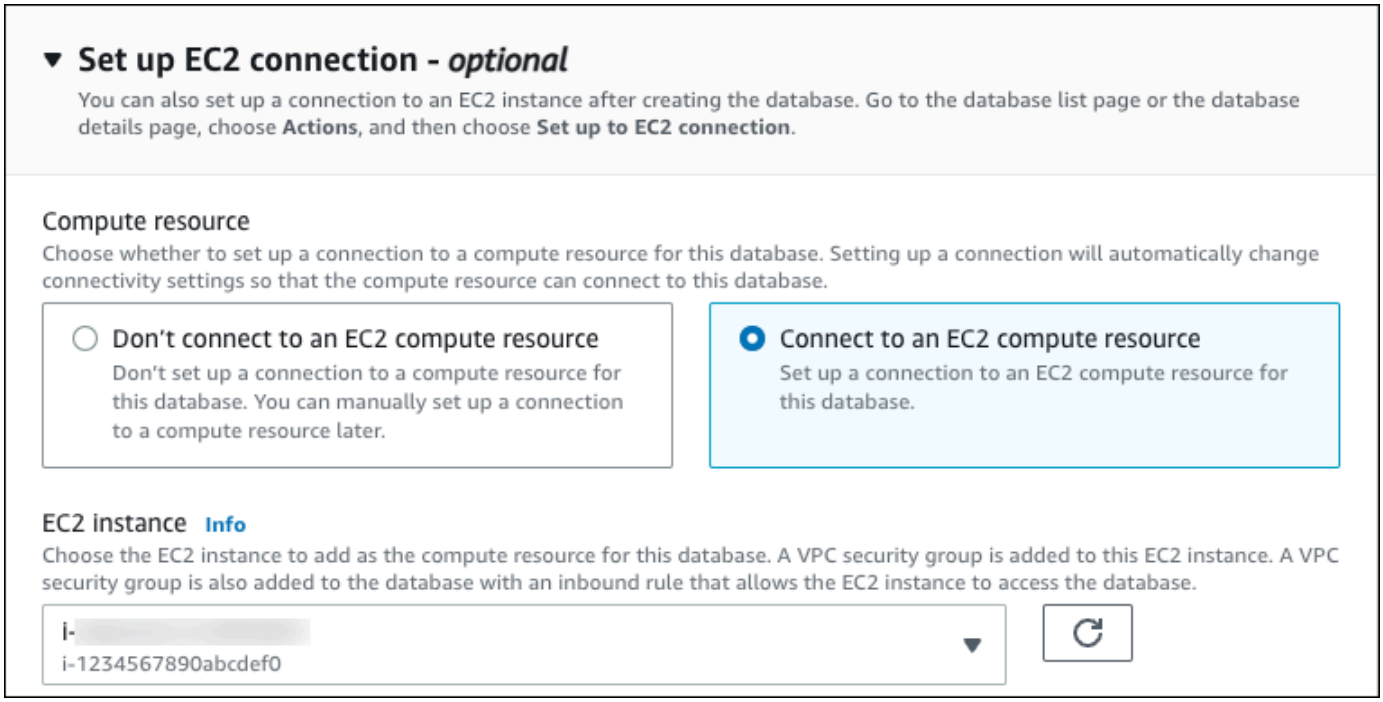
   Create database (建立資料庫) 頁面看起來應該會如下圖所示。



8. 針對主要使用者名稱，輸入使用者的名稱，或保留預設名稱 (**postgres**)。
9. 如要將自動產生的主要密碼用在資料庫叢集，請選取自動產生密碼。

   如要輸入您的主要密碼，請確認清除自動產生密碼方塊，然後在主要密碼和確認密碼中輸入相同的密碼。
10. 若要設定與先前建立之 EC2 執行個體的連線，請開啟設定 EC2 連線 - 選用。

    選取連線至 EC2 運算資源。選擇先前建立的 EC2 執行個體。



11. 開啟檢視輕鬆建立的預設設定。

View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:aurora-postgresql-13	No
Subnet group	create-subnet-group	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-1234567	Yes
Publicly accessible	No	Yes
Database port	5432	Yes
DB cluster identifier	database-test1	Yes
DB instance identifier	database-1	Yes
DB engine version	13.6	Yes
DB parameter group	default.aurora-postgresql13	Yes
DB cluster parameter group	default.aurora-postgresql13	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

您可以檢查與 Easy Create (輕鬆建立) 一起使用的預設設定。資料庫建立後可編輯欄顯示您可以在資料庫建立後變更的選項。

- 若該欄的設定為否，而您想要其他設定，可以使用標準建立來建立資料庫叢集。
- 若該欄的設定為是，而您想要其他設定，可以使用標準建立來建立資料庫叢集，或在建立後修改該資料庫叢集的設定。

12. 選擇建立資料庫。

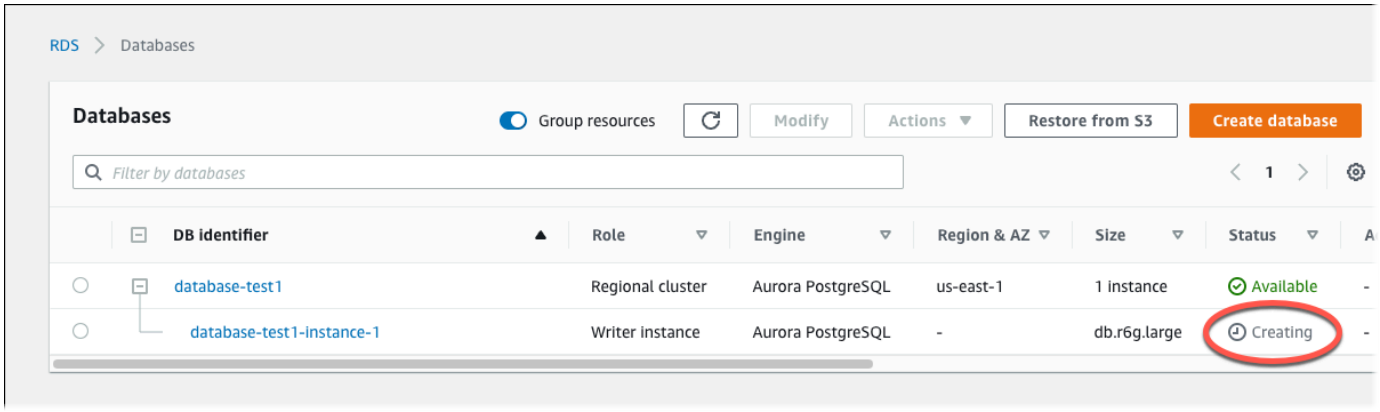
    若要檢視資料庫叢集的主要使用者名稱和密碼，請選擇檢視登入資料詳細資訊。

    您可以使用出現的使用者名稱和密碼，來以主要使用者的身分連線至資料庫叢集。

    > **Important**
    >
    > 您無法再次檢視主要使用者密碼。如果您沒有記錄下來，您可能需要進行變更。
    > 若您需要在資料庫叢集可供使用後變更主要使用者密碼，您可以將資料庫叢集修改為這麼做。如需修改資料庫叢集的詳細資訊，請參閱修改 Amazon Aurora 資料庫叢集。

13. 資料庫清單中，選擇新 Aurora PostgreSQL 資料庫叢集的名稱，以顯示其詳細資訊。

    在資料庫叢集可供使用之前，寫入器執行個體會處於建立中狀態。



寫入器執行個體狀態變更為可用時，您便能連線至資料庫叢集。視資料庫執行個體類別和儲存體數量而定，可能需要最多 20 分鐘的時間，新的資料庫叢集才會可用。

（選用）使用 建立 VPC、EC2 執行個體和 Aurora PostgreSQL 叢集 CloudFormation

您可以使用 CloudFormation 將基礎設施視為程式碼來佈建 AWS 資源，而不是使用 主控台來建立 VPC、EC2 執行個體和 Aurora PostgreSQL 資料庫叢集。為了協助您將 AWS 資源組織成更小且更易於管理的單位，您可以使用 CloudFormation 巢狀堆疊功能。如需詳細資訊，請參閱在 CloudFormation 主控台上建立堆疊和使用巢狀堆疊。

> **Important**
>
> CloudFormation 是免費的，但 CloudFormation 建立的資源是即時的。您需要支付這些資源的標準使用費，直到將其終止為止。如需詳細資訊，請參閱 Amazon Aurora 定價。

若要使用 CloudFormation 主控台建立資源，請完成下列步驟：

- 步驟 1：下載 CloudFormation 範本
- 步驟 2：使用 CloudFormation 設定資源

下載 CloudFormation 範本

CloudFormation 範本是 JSON 或 YAML 文字檔案，其中包含您要在堆疊中建立之資源的相關組態資訊。此範本也會為您以及 Aurora 叢集建立 VPC 和堡壘主機。

若要下載範本檔案，請開啟以下連結 Aurora PostgreSQL CloudFormation 範本。

在 Github 頁面中，按一下下載原始檔案按鈕，即可儲存範本 YAML 檔案。

使用 CloudFormation 設定資源

> **Note**
>
> 開始此程序之前，請確定 AWS 帳戶中有 EC2 執行個體的金鑰對。如需詳細資訊，請參閱 Amazon EC2 金鑰對與 Linux 執行個體。

使用 CloudFormation 範本時，您必須選取正確的參數，以確保您的資源已正確建立。請遵循下列步驟：

1. 登入 AWS 管理主控台 並在 https：//https://console.aws.amazon.com/cloudformation 開啟 CloudFormation 主控台。

2. 選擇 Create Stack (建立堆疊)。
3. 在 [指定範本] 區段中，選取從電腦上傳範本檔案，然後選擇下一步。
4. 在指定堆疊詳細資訊頁面，設定下列參數：
   a. 將堆疊名稱設為 AurPostgreSQLTestStack。
   b. 在參數下，選取兩個可用區域來設定可用區域。
   c. 在 Linux 堡壘主機組態下，針對金鑰名稱，選取金鑰對以登入 EC2 執行個體。
   d. 在 Linux 堡壘主機組態設定中，將允許的 IP 範圍設定為 IP 位址。若要使用 Secure Shell (SSH) 連線到 VPC 中的 EC2 執行個體，請使用位於 https://checkip.amazonaws.com 的服務來判斷公有 IP 位址。IP 地址的範例為 192.0.2.1/32。

      > **Warning**
      >
      > 如果您使用 `0.0.0.0/0` 進行 SSH 存取，則可讓所有 IP 地址使用 SSH 存取您的公有 EC2 執行個體。通常在測試環境中短暫使用此方法是沒有問題的，但在生產環境則不安全。在生產環境中，您應只授權特定 IP 地址或特定範圍的地址可使用 SSH 存取您的 EC2 執行個體。

   e. 在資料庫一般組態下，將資料庫執行個體類別設為 db.t4g.large。
   f. 將資料庫名稱設為 **`database-test1`**。
   g. 對於資料庫主要使用者名稱，輸入主要使用者的名稱。
   h. 在本教學課程中，將使用 Secrets Manager 管理資料庫主要使用者密碼設為 `false`。
   i. 針對資料庫密碼，設定您選擇的密碼。請記住此密碼，以供教學課程中的後續步驟使用。
   j. 將多可用區域部署設為 `false`。
   k. 保留所有其他設定作為預設值。按一下下一步以繼續。
5. 在設定堆疊選項頁面中，保留所有預設選項。按一下下一步以繼續。
6. 在檢閱堆疊頁面中，在檢查資料庫和 Linux 堡壘主機選項後選取提交。

堆疊建立程序完成後，請檢視名稱為 BastionStack 和 APGNS 的堆疊，以記下連線至資料庫所需的資訊。如需詳細資訊，請參閱在 上檢視 CloudFormation 堆疊資料和資源 AWS 管理主控台。

步驟 3：連線至 Aurora PostgreSQL 資料庫叢集

您可以使用任何標準 PostgreSQL 用戶端應用程式來連線至資料庫叢集。在此範例中，您會使用 psql 命令列用戶端來連線至 Aurora PostgreSQL 資料庫叢集。

連線至 Aurora PostgreSQL 資料庫叢集

1. 為資料庫叢集尋找寫入器執行個體的端點 (DNS 名稱) 和連線埠號碼。

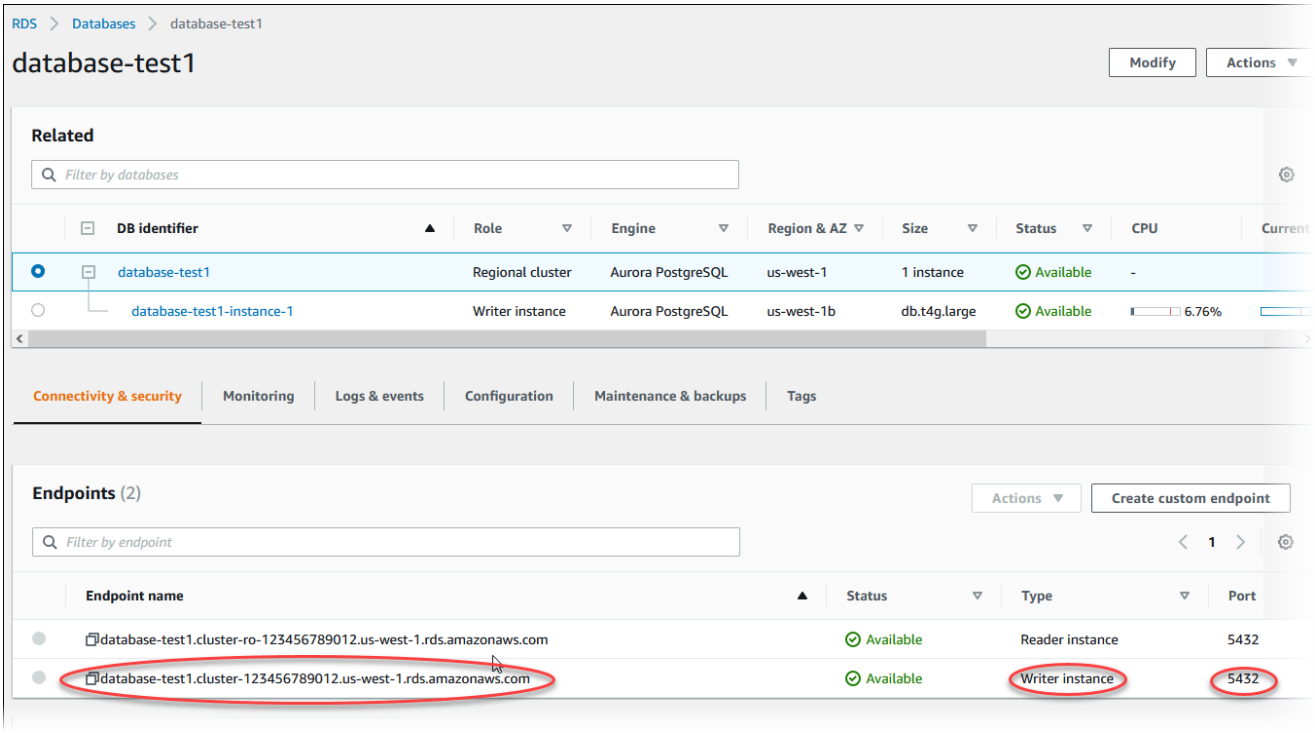
   a. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。

   b. 在 Amazon RDS 主控台的右上角， AWS 區域 選擇資料庫叢集的 。

   c. 在導覽窗格中，選擇 Databases (資料庫)。

   d. 選擇 Aurora PostgreSQL 資料庫叢集名稱以顯示其詳細資訊。

   e. 在連線能力和安全性索引標籤上，複製寫入器執行個體的端點。另外，請記下連線埠號碼。您需要同時有端點和連線埠號碼，才能連線至資料庫叢集。



2. 請遵循《Amazon EC2 使用者指南》中的連線至 Linux 執行個體步驟，以連線您先前所建的 EC2 執行個體。

   建議您使用 SSH 連線至 EC2 執行個體。如果 SSH 用戶端公用程式已安裝在 Windows、Linux 或 Mac 上，您可以使用下列命令格式連線至執行個體：

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

例如，假設 ec2-database-connect-key-pair.pem 存放在 Linux 上 的 /dir1 中，而 EC2 執行個體的公用 IPv4 DNS 為 ec2-12-345-678-90.compute-1.amazonaws.com。您的 SSH 命令看起來如下所示：

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

3. 更新 EC2 執行個體上的軟體，以取得最新的錯誤修正和安全性更新。為此，請使用下列命令。

> **Note**
>
> -y 選項不要求確認就會安裝更新。若要先檢查更新再安裝，請省略此選項。

```
sudo dnf update -y
```

4. 請執行下列命令，在 Amazon Linux 2023 上安裝 PostgreSQL 的 psql 命令列用戶端：

```
sudo dnf install postgresql15
```

5. 連線至 Aurora PostgreSQL 資料庫叢集。例如，輸入下列命令。此動作可讓您使用 psql 用戶端，連線至 Aurora PostgreSQL 資料庫叢集。

   取代 *endpoint* 的寫入器執行個體端點，取代 *postgres* 要連線的資料庫名稱 --dbname，以及取代您用於 *postgres* 的主要使用者名稱。提示您輸入密碼時，請提供您使用的主要密碼。

```
psql --host=endpoint --port=5432 --dbname=postgres --username=postgres
```

   輸入使用者的密碼之後，您應該會看到類似如下的輸出。

```
psql (14.3, server 14.6)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
 compression: off)
Type "help" for help.

postgres=>
```

如需連線至 Aurora PostgreSQL 資料庫叢集的詳細資訊，請參閱 連接至 Amazon Aurora PostgreSQL 資料庫叢集。若您無法連線至資料庫叢集，請參閱 無法連線至 Amazon RDS 資料庫執行個體。

基於安全考量，最佳做法是使用加密連線。僅當用戶端和伺服器位於同一 VPC 且網路受信任時，才使用未加密的 PostgreSQL 連線。如需使用加密連線的詳細資訊，請參閱 使用 SSL/TLS 保護 Aurora PostgreSQL 資料的安全。

6. 執行 SQL 命令。

   例如，下列 SQL 命令會顯示目前的日期和時間：

```
SELECT CURRENT_TIMESTAMP;
```

步驟 4：刪除 EC2 執行個體和資料庫叢集

在連線至您已建立的範例 EC2 執行個體與資料庫叢集，並探索這些執行個體之後，請將其刪除，才不會再對您收費。

如果您使用 CloudFormation 建立資源，請略過此步驟並前往下一個步驟。

刪除 EC2 執行個體

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/ec2/：// 開啟 Amazon EC2 主控台。
2. 在導覽窗格中，選擇執行個體。
3. 選取 EC2 執行個體，並選取執行個體狀態、終止執行個體。
4. 出現確認提示時，請選擇 Terminate (終止)。

如需刪除 EC2 執行個體的詳細資訊，請參閱《Amazon EC2 使用者指南》中的終止執行個體。

刪除資料庫叢集

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 選擇 Databases (資料庫)，然後選擇與該資料庫叢集相關聯的資料庫執行個體。
3. 對於 Actions (動作)，請選擇 Delete (刪除)。

4. 選擇 Delete (刪除)。

   在刪除與資料庫叢集相關聯的所有資料庫執行個體後，就會自動刪除該資料庫叢集。

(選用) 刪除使用 CloudFormation 建立的 EC2 執行個體和資料庫叢集

如果您使用 CloudFormation 來建立資源，請在連線至 後刪除 CloudFormation 堆疊，並探索範例 EC2 執行個體和資料庫叢集，因此您不再需要支付這些堆疊的費用。

刪除 CloudFormation 資源

1. 開啟 CloudFormation 主控台。
2. 在 CloudFormation 主控台的堆疊頁面，選取根堆疊 (不含名稱 VPCStack、BastionStack 或 APGNS 的堆疊)。
3. 選擇 刪除。
4. 出現確認提示時，請選取刪除堆疊。

如需在 CloudFormation 中刪除堆疊的詳細資訊，請參閱AWS CloudFormation 《 使用者指南》中的刪除 CloudFormation 主控台上的堆疊。

(選用) 將資料庫叢集連線至 Lambda 函數

您也可以將 Aurora PostgreSQL 資料庫叢集連線至 Lambda 無伺服器運算資源。Lambda 函數允許您在沒有佈建或管理基礎設施的情況下執行程式碼。Lambda 函數還允許您自動回應任何規模的程式碼執行請求，從每天十幾個事件到每秒數百個事件。如需更多詳細資訊，請參閱 自動連線 Lambda 函數和 Aurora 資料庫叢集。

使用快速組態建立

您可以使用快速組態在幾秒鐘內建立並連線至 Aurora PostgreSQL 無伺服器資料庫，透過預先設定的預設值提供簡化的資料庫建立體驗，協助您輕鬆快速地開始使用。

只要按兩下，您就可以讓 Aurora 叢集與無伺服器執行個體準備好在幾秒鐘內使用。您也可以使用 AWS Command Line Interface (AWS CLI) 或 AWS SDKs搭配 參數`--express-configuration`，透過單一 API 呼叫在叢集內建立叢集和執行個體，以便在幾秒鐘內準備好執行查詢。您可以在建立資料庫之後，彈性地修改大部分的設定，以及大部分的其他設定。如需這些設定的詳細資訊，請參閱 快速組態設定。建立之後，您就可以存取 Aurora 的大部分功能和優點，例如新增讀取器以獲得高可用性和可擴展性的能力，以及自動容錯移轉功能。只有在使用 the section called “使用完整組態建立”選

項時，才能使用某些選項，例如指定客戶受管加密金鑰、選取特定引擎版本，或將叢集與 Amazon Virtual Private Cloud (VPC) 建立關聯。此 Aurora PostgreSQL 的建立體驗也適用於 AWS 免費方案。

主題

- 先決條件
- 支援的區域
- 網際網路存取閘道
- 建立具有快速組態的資料庫叢集
- 快速組態設定
- 連線至具有快速組態的叢集
- 還原透過快速組態建立的叢集
- 限制

先決條件

您必須先擁有下列許可，才能建立具有快速組態的 Aurora PostgreSQL 資料庫叢集：

- `ec2:DescribeAvailabilityZones`
- `iam:CreateServiceLinkedRole`
- `rds:CreateDBCluster`
- `rds:CreateDBInstance`
- `rds:EnableInternetAccessGateway`

支援的區域

您可以在除 AWS 中國、 AWS GovCloud (US)、中東 (UAE) 和中東 （ 巴林 ） 區域 AWS 區域 以外所有 中建立具有快速組態的 Aurora PostgreSQL 無伺服器叢集。

網際網路存取閘道

Aurora 也提供預設啟用的網際網路存取閘道，以安全地連線至透過快速組態建立的叢集。目前僅快速組態叢集支援此功能。它支援 PostgreSQL 線路通訊協定，可讓您從可能在您 AWS 基礎設施外部執行的各種工具和開發平台，透過網際網路進行連線 - 不需要 VPN 或 AWS Direct Connect。閘道分散在多個可用區域，提供與 Aurora 叢集相同的高可用性層級。這是 Aurora 的受管元件，因此您不需要套用任何軟體更新或修補程式。

網際網路存取閘道與 AWS Identity and Access Management (IAM) 整合，可讓您強制執行身分型許可，以授權存取並使用暫時性字符型存取來保護您的資料。使用閘道時，會自動為您管理資料庫管理員使用者的必要設定，包括授予 rds_iam 角色。

建立具有快速組態的資料庫叢集

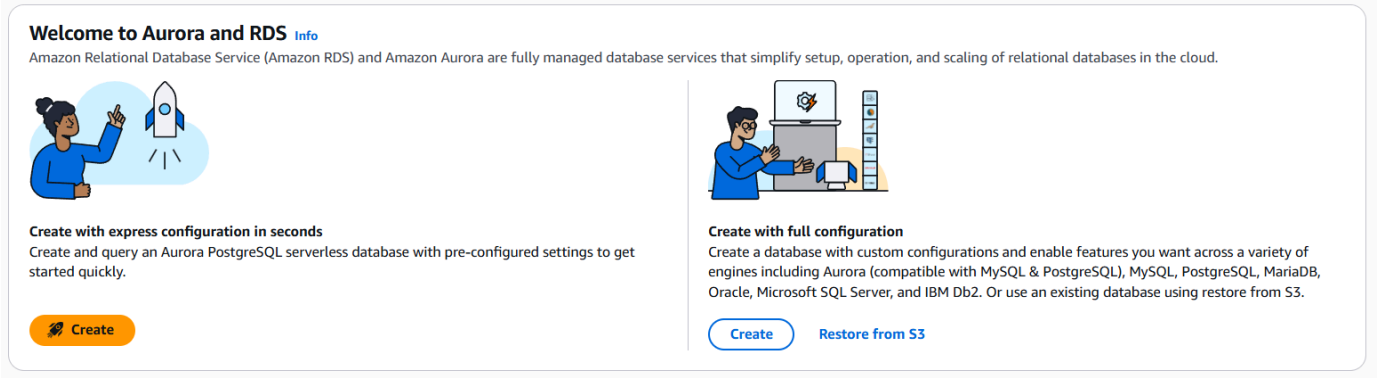
您可以使用 AWS 管理主控台 AWS CLI、 或 RDS API 建立具有快速組態的 Aurora PostgreSQL 資料庫叢集。

主控台

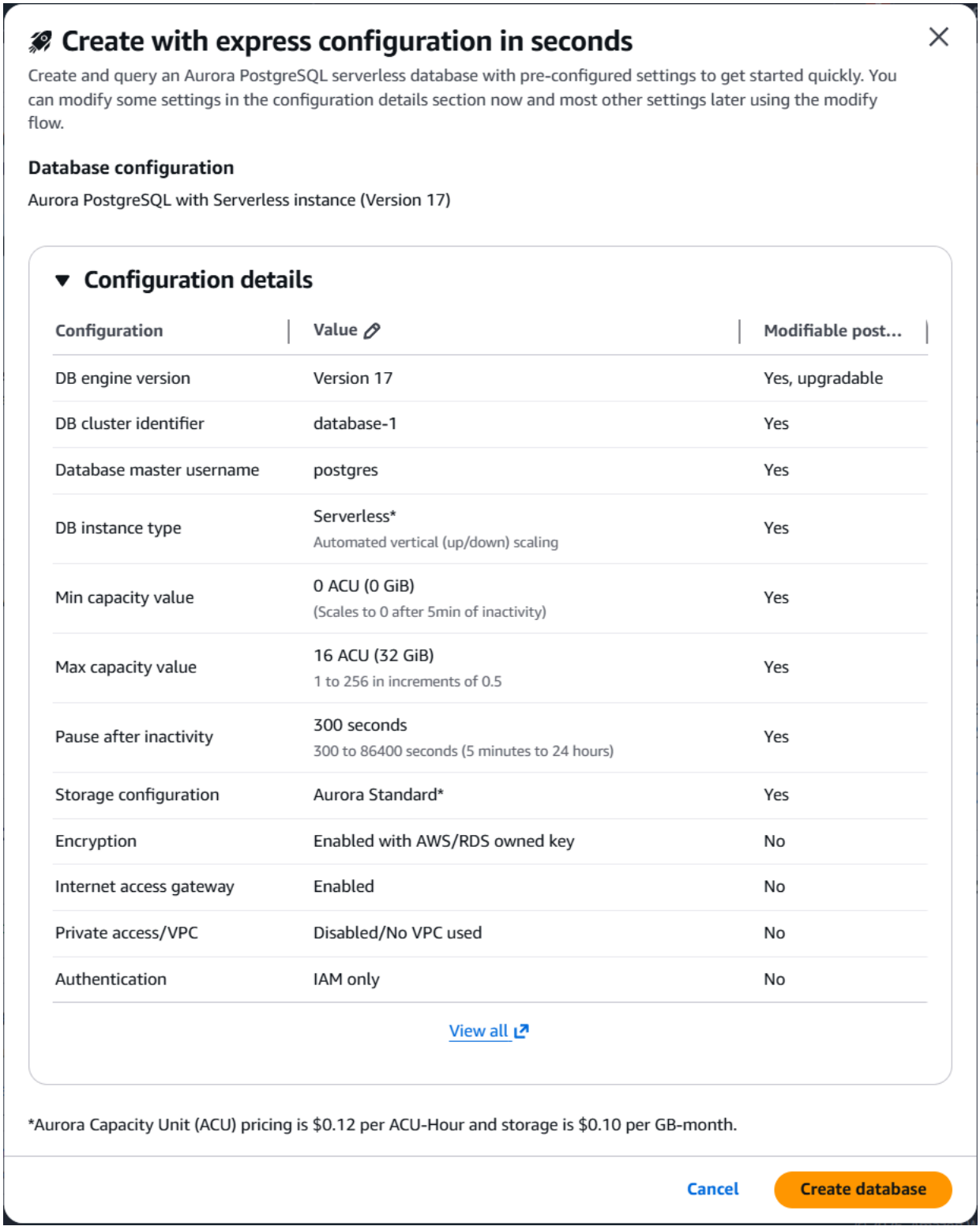
若要使用主控台建立具有快速組態的 ：

使用主控台以快速組態建立

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在 的右上角 AWS 管理主控台，選擇您要在其中建立資料庫叢集 AWS 的區域。
3. 在導覽窗格中，選擇 Databases (資料庫)。
4. 在歡迎使用 Aurora 和 RDS 頁面上，找到左側以秒為單位快速設定建立區段，然後選擇建立。

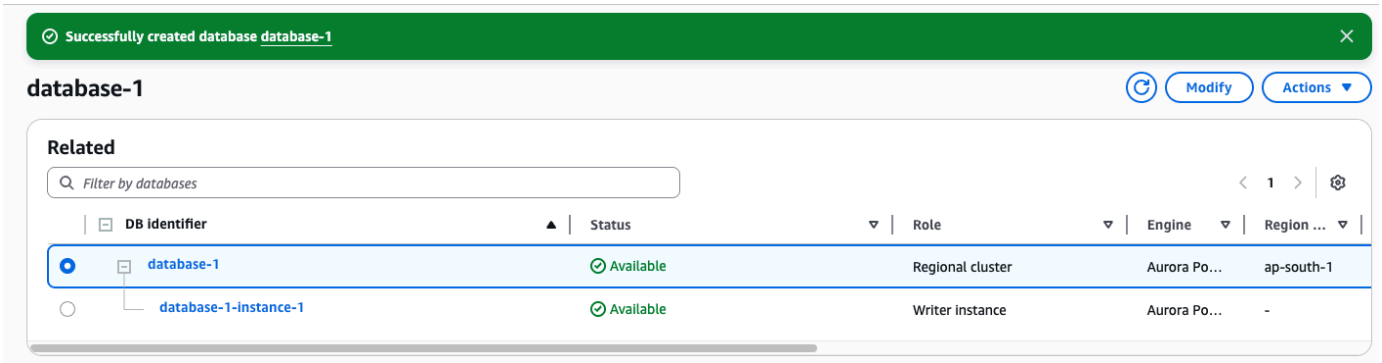


5. 在建立快速組態對話方塊中檢閱預先設定的設定。



6. （選用）根據您的需求修改資料庫叢集識別符或容量範圍。
7. 選擇建立資料庫。

8. 您的 Aurora PostgreSQL Serverless 資料庫應該會在幾秒鐘內準備就緒。成功橫幅會確認建立，且資料庫狀態會變更為「可用」。



CLI

您可以使用 AWS CLI 來建立具有快速組態的 Aurora PostgreSQL 叢集。

針對 Linux 或 MacOS。

```
aws rds create-db-cluster --db-cluster-identifier sample-express-cluster \
    --engine aurora-postgresql \
    --with-express-configuration
```

針對 Windows：

```
aws rds create-db-cluster --db-cluster-identifier sample-express-cluster ^
    --engine aurora-postgresql ^
    --with-express-configuration
```

快速組態參數負責建立資料庫叢集、Aurora 無伺服器執行個體、設定網際網路存取閘道，以及管理員資料庫使用者的 IAM 身分驗證 （ 預設值 = postgres)。

API

您可以使用相關的 AWS SDK 並呼叫 CreateDBCluster 操作，以使用快速組態建立資料庫叢集和資料庫執行個體。

快速組態設定

下表說明快速組態的設定，並指出它們是否可以在建立後修改。

設定	設定說明	預設設定和限制
Auto minor version upgrade (自動次要版本升級)	此設定可讓您的 Aurora 資料庫叢集在資料庫引擎可用時自動接收偏好的次要版本升級。如需 Aurora PostgreSQL 引擎更新的詳細資訊，請參閱 Amazon Aurora PostgreSQL 的資料庫引擎更新。	預設啟用。可以在建立操作完成後變更。
AWS KMS 金鑰	具有快速組態的叢集會使用 AWS/RDS Service 擁有的金鑰加密。	使用 AWS/RDS 服務擁有的金鑰啟用。無法修改。
憑證認證機構	使用使用 AWS 根憑證的網際網路存取閘道啟用具有快速組態的叢集。如需詳細資訊，請參閱 ACM 根 CAs	不適用於透過網際網路存取閘道連線
叢集儲存組態	資料庫叢集的儲存類型：Aurora Standard 或 Aurora I/O-Optimized。如需詳細資訊，請參閱 Amazon Aurora 資料庫叢集的儲存組態。	根據預設，Aurora 標準。可以在建立操作完成後變更。
Copy tags to snapshots (將標籤複製到快照)	選擇此選項，可在您建立資料庫快照時將任何資料庫執行個體標籤複製到此快照。如需詳細資訊，請參閱標記 Amazon Aurora 和 Amazon RDS 資源。	預設為停用。可以在建立期間或建立之後變更。
資料庫身分驗證	具有快速組態和網際網路存取閘道的叢集僅支援 IAM 身分驗證。如需詳細資訊，請參閱 IAM 資料庫身分驗證。	預設使用 IAM 身分驗證啟用。無法修改。
Database port (資料庫連線埠)	具有快速組態和網際網路存取閘道的叢集僅支援預設 PostgreSQL 連接埠	預設值 = 5432。無法修改。
DB cluster identifier (資料庫叢集識別符)	輸入資料庫叢集的名稱，該名稱對您選擇的 AWS 區域中的帳戶是唯一的。此識別符用於資料庫叢集中的叢集端點位址。如需叢集端點的資訊，	必要參數。可以在建立期間或建立之後變更。

設定	設定說明	預設設定和限制
	請參閱 Amazon Aurora 端點連線。資料庫叢集識別符有下列限制：必須包含 1 到 63 個英數字元或連字號。第一個字元必須是字母。不能以一個連字號結尾或是連續包含兩個連字號。每個區域每個 AWS 帳戶的所有資料庫叢集都必須是唯一的 AWS 。	
DB cluster parameter group (資料庫叢集參數群組)	具有快速組態的叢集使用 Aurora 預設資料庫叢集參數群組。	預設參數群組。可以在建立操作完成後變更。
DB instance class (資料庫執行個體類別)	具有快速組態的叢集從 Aurora Serverless 執行個體開始。	無伺服器 v2 執行個體。可以在建立操作完成後變更。
DB subnet group (資料庫子網路群組)	您的 VPC 內不會建立具有快速組態的叢集。透過 Aurora 網際網路存取閘道存取	沒有相關聯的 VPC。無法設定。
啟用刪除保護	選擇 Enable deletion protection (啟用刪除保護) 以避免您的資料庫叢集遭意外刪除。如果您使用主控台建立生產資料庫叢集，預設會啟用刪除保護功能。	預設為停用。可以在建立完成期間或之後變更。
Enable encryption (啟用加密)	具有快速組態的叢集會使用 AWS/RDS Service 擁有的金鑰加密。	使用 AWS/RDS 服務擁有的金鑰啟用。無法修改。
Enable Enhanced Monitoring (啟用增強型監控)	選擇 Enable enhanced monitoring (啟用增強型監控)，以針對資料庫叢集執行所在的作業系統即時收集指標。如需詳細資訊，請參閱使用增強型監控來監控 OS 指標。	預設為停用。可以在建立操作完成後變更。

設定	設定說明	預設設定和限制
啟用 RDS 資料 API	選擇啟用 RDS Data API 以啟用 RDS Data API (資料 API)。資料 API 提供安全的 HTTP 端點，無需管理連線即可執行 SQL 陳述式。如需詳細資訊，請參閱使用 Amazon RDS Data API。	預設為停用。可以在建立操作完成後變更。
引擎類型	只有 Aurora PostgreSQL 支援具有快速組態叢集的叢集	必要的參數。支援的值：「aurora-postgresql」。無法修改。
引擎版本	具有快速組態的叢集會使用預設的主要和次要版本。如需預設版本的詳細資訊，請參閱 Aurora 版本。	預設主要和次要版本。引擎版本稍後可以升級。
容錯移轉優先順序	選擇執行個體的容錯移轉優先順序。如果您未選擇值，則預設值為 tier-1 (第一層)。此優先順序決定從主要執行個體失敗中復原時提升 Aurora 複本的順序。如需詳細資訊，請參閱 Aurora 資料庫叢集的容錯能力。	預設值 = 1。可以在建立操作完成後變更。
初始資料庫名稱	預設資料庫的名稱。如果您未提供 Aurora PostgreSQL 資料庫叢集的名稱，Amazon RDS 會建立名為 postgres 的資料庫。對於 Aurora PostgreSQL，預設資料庫名稱具有這些限制：它必須包含 1-63 個英數字元。必須以字母開頭。後續字元可以是字母、底線或數字 (0–9)。不能是資料庫引擎的保留字。若要建立其他資料庫，請連接至資料庫叢集，並使用 SQL 命令 CREATE DATABASE。如需連線至資料庫叢集的詳細資訊，請參閱使用網際網路存取閘道連線至 Amazon Aurora 資料庫叢集。	預設值 = postgres。可以在建立完成期間或之後變更。

設定	設定說明	預設設定和限制
Log exports (日誌匯出)	在 Log exports (日誌匯出) 區段中，選擇您要開始發佈至 Amazon CloudWatch Logs 的日誌。如需將 Aurora MySQL 日誌發佈至 CloudWatch Logs 的詳細資訊，請參閱將 Amazon Aurora MySQL 日誌發佈至 Amazon CloudWatch Logs。如需將 Aurora PostgreSQL 日誌發佈至 CloudWatch Logs 的詳細資訊，請參閱將 Aurora PostgreSQL 日誌發佈至 Amazon CloudWatch Logs。	預設為停用。可以在建立期間或之後變更。
Maintenance window (維護時段)	選擇 Select window (選取時段)，並指定可能發生系統維護的每週時間範圍。或者，選擇 No preference (無偏好設定)，讓 Amazon RDS 隨機指派期間。	預設維護時段因區域而異。可以在建立期間或之後變更。
在 AWS Secrets Manager 中管理主登入資料	具有快速組態和網際網路存取閘道的叢集僅支援 IAM 身分驗證。此設定不適用。	N/A
Master password (主要密碼)	具有快速組態和網際網路存取閘道的叢集僅支援 IAM 身分驗證。此設定不適用。	N/A
主要使用者身分驗證類型	具有快速組態的叢集會自動將身分驗證類型設定為 IAM 身分驗證。	支援的值 = iam-db-auth。無法修改。
主要使用者名稱	輸入要用來登入資料庫叢集的主要使用者名稱的名稱。針對 Aurora PostgreSQL，其中必須包含 1–63 個英數字元。第一個字元必須是字母。名稱不能是資料庫引擎的保留字。您無法在建立資料庫叢集之後變更主要使用者名稱。	預設值 = postgres。可以在建立期間或之後變更。

設定	設定說明	預設設定和限制
多可用區域部署	具有快速組態的叢集不會與 VPC 相關聯，且會自動為您選擇可用區域。您可以在建立資料庫之後新增僅供讀取複本。	系統會自動選擇可用區域。寫入器和讀取器執行個體會放置在不同的可用區域。
Network type (網路類型)	具有快速組態的叢集不會與 VPC 相關聯。	不適用
公用存取	使用網際網路存取閘道建立具有快速組態的叢集，該閘道透過與 AWS Identity and Access Management 和 Shield AWS 的內建整合提供安全的網際網路存取。	不適用
RDS 延長支援	選取啟用 RDS 延長支援，以允許支援的主要引擎版本在超過 Aurora 標準支援結束日期後繼續執行。當您建立資料庫叢集時，Amazon Aurora 預設為 RDS 延長支援。若要防止在 Aurora 標準支援日期結束後建立新的資料庫叢集，並避免 RDS 延長支援的費用，請停用此設定。在 RDS 延長支援定價開始日期之前，您現有的資料庫叢集不會產生費用。如需詳細資訊，請參閱 Amazon RDS Extended Support with Amazon Aurora。	不支援
RDS Proxy	與 VPC 無關的叢集不支援 RDS Proxy。如需詳細資訊，請參閱 Amazon RDS Proxy for Aurora。	不支援
保留期間	選擇從 1 到 35 天的時間長度，Aurora 會在此時間內保留資料庫備份副本。備份副本可用來進行資料庫的 point-in-time 還原 (PITR) 作業，單位可計算至秒。	預設值 = 1。可以在建立操作完成後變更。

設定	設定說明	預設設定和限制
備份時段	從 8 小時的時段中隨機選取的 30 分鐘時段，以自動備份資料庫。	使用預設參數群組。稍後可以使用參數群組進行變更。
Turn on DevOps Guru (開啟 DevOps Guru)	DevOps Guru for RDS 提供效能異常的詳細分析，必須開啟績效詳情。如需詳細資訊，請參閱設定 DevOps Guru for RDS。	預設為停用。可以在建立操作完成後變更。
開啟績效詳情	Amazon RDS Performance Insights 為您的資料庫提供監控資訊。如需詳細資訊，請參閱在 Amazon Aurora 上使用 Performance Insights 監控資料庫負載。	預設為停用。可以在建立操作完成後變更。
Virtual Private Cloud (VPC)	具有快速組態的叢集不會與 VPC 相關聯	N/A
VPC security group (firewall) (VPC 安全群組 (防火牆))	具有快速組態的叢集不會與 VPC 相關聯	N/A
寫入轉送	可讓您將讀取器端點上傳送的寫入轉送至寫入器節點進行處理。實現read-after-write一致性。如需詳細資訊，請參閱 Aurora PostgreSQL 中的本機寫入轉送。	預設為停用。可以在建立操作完成後變更。

連線至具有快速組態的叢集

使用快速組態建立的叢集會自動設定網際網路存取閘道，透過 IAM 身分驗證提供資料庫的公開存取。請確定您用來連線至資料庫的 IAM 身分具有 rds-db：connect 的許可，以產生存取資料庫所需的身分驗證字符。若要進一步了解，請參閱 IAM 資料庫身分驗證。

下列指南示範如何使用 中「連線與安全」索引標籤中的各種選項 AWS 管理主控台 來連線至資料庫。

RDS 主控台會顯示相關資訊，例如程式碼片段、端點詳細資訊和其他連線詳細資訊，以協助您連線至資料庫。它也提供公用程式來產生字符，並透過 AWS CloudShell 直接存取。會使用您在建立資料庫時設定的主要使用者名稱來產生字符公用程式和 CloudShell 連線。

使用連線與安全索引標籤

建立資料庫後，導覽至連線與安全索引標籤以存取連線選項。連線至資料庫區段提供三種連線方法：

程式碼片段

透過 SDK、APIs 或第三方工具連線時使用 。主控台為您選擇的程式設計語言提供ready-to-use程式碼片段。這些程式碼片段也會動態反映資料庫的身分驗證組態。

下列步驟顯示使用 psql 命令列公用程式程式碼片段連線至資料庫叢集。

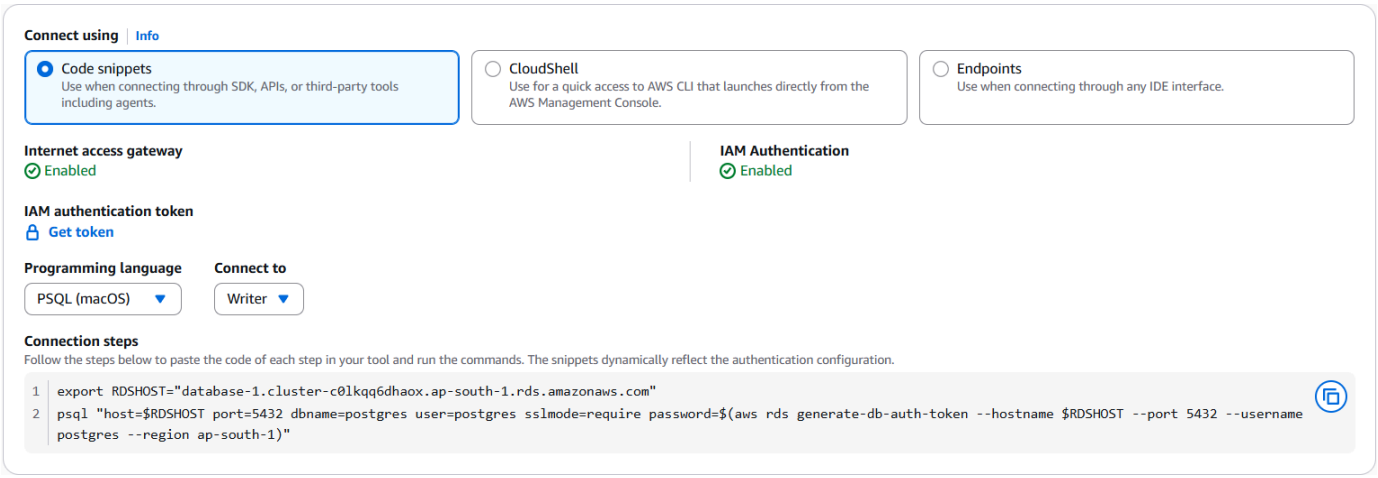
先決條件

- 您可以使用 PostgreSQL 互動式終端機 psql 等工具來連線至 Aurora PostgreSQL 資料庫叢集。若要安裝 psql 並進一步了解如何使用 PostgreSQL 互動式終端機，請參閱 PostgreSQL 文件中的 psql。
- 您必須 AWS CLI 安裝最新版本的 。若要安裝或更新 AWS CLI，請參閱安裝或更新至最新版本的 AWS CLI。

使用 psql 連線

與 psql 連線

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 從資料庫清單索引標籤導覽至您的資料庫叢集。
3. 在連線與安全索引標籤下，選取個別作業系統 (macOS、linux 或 Windows) 的 PSQL 程式碼片段。
4. 複製連線步驟模態中顯示的程式碼。



5. 在本機開發環境/機器上開啟終端機。
6. （選用）如果您已經設定 AWS CLI 和 登入資料，您可以略過此步驟。否則，請使用下列命令進行暫時登入工作階段。

```
aws login
```

7. 貼上您在步驟 4 中複製的命令。您應該會看到 psql 登入和「postgres =>」提示，讓您執行 SQL 命令。

CloudShell

主控台提供使用 AWS CloudShell 連線至叢集的自動化方式。下列指南顯示使用 psql 和 CloudShell 連線的步驟。

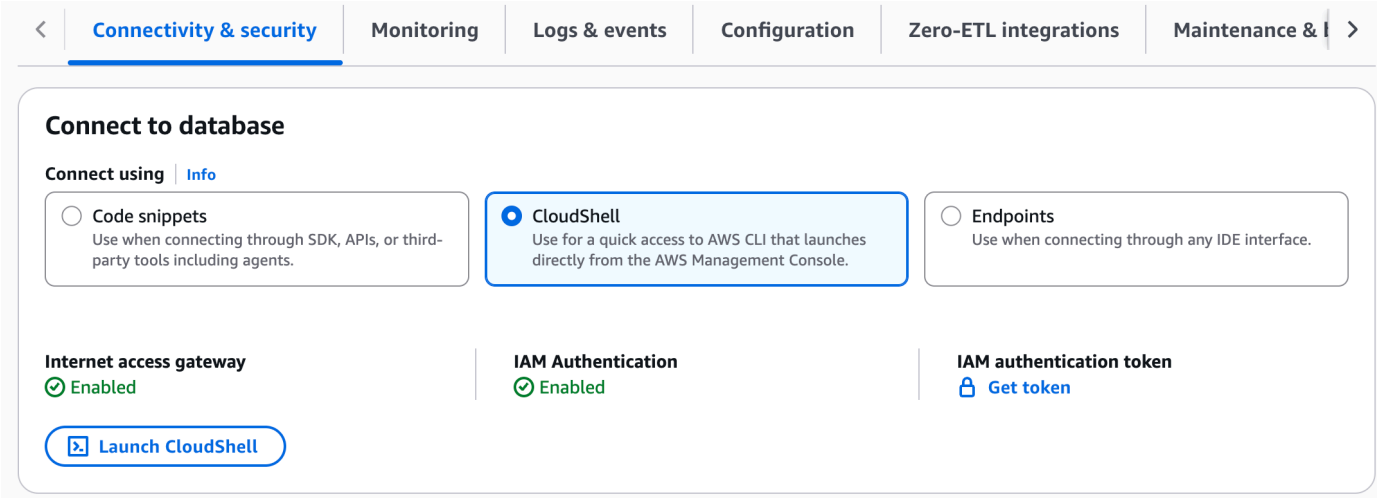
先決條件

無

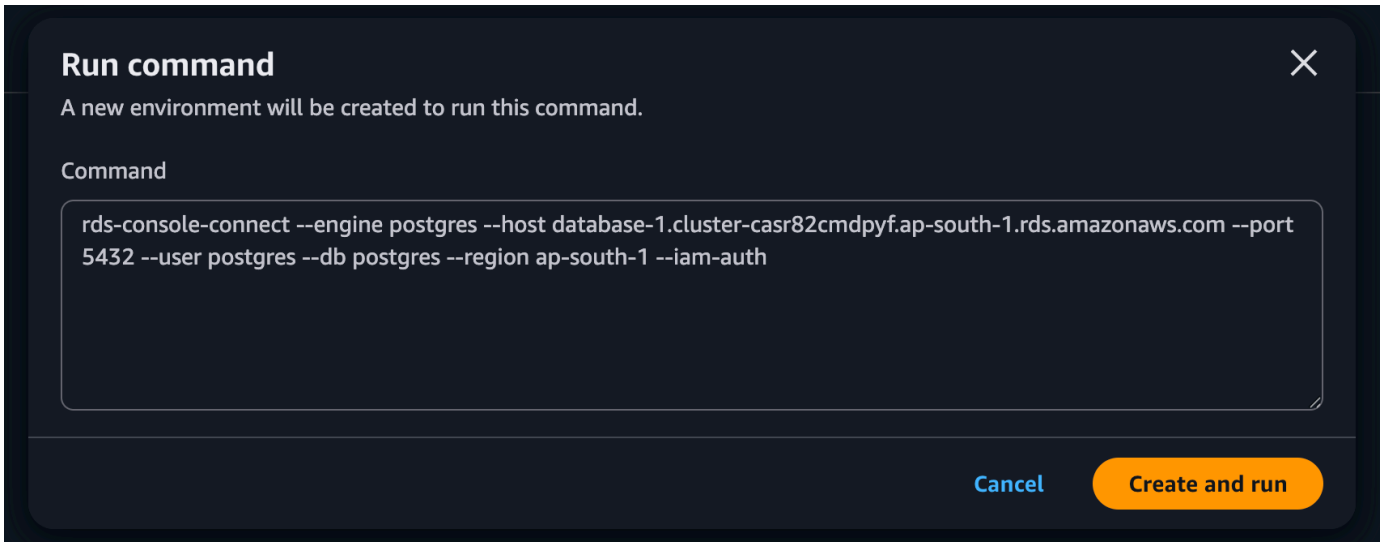
使用 CloudShell 和 PSQL

使用 CloudShell 和 PSQL 進行連線

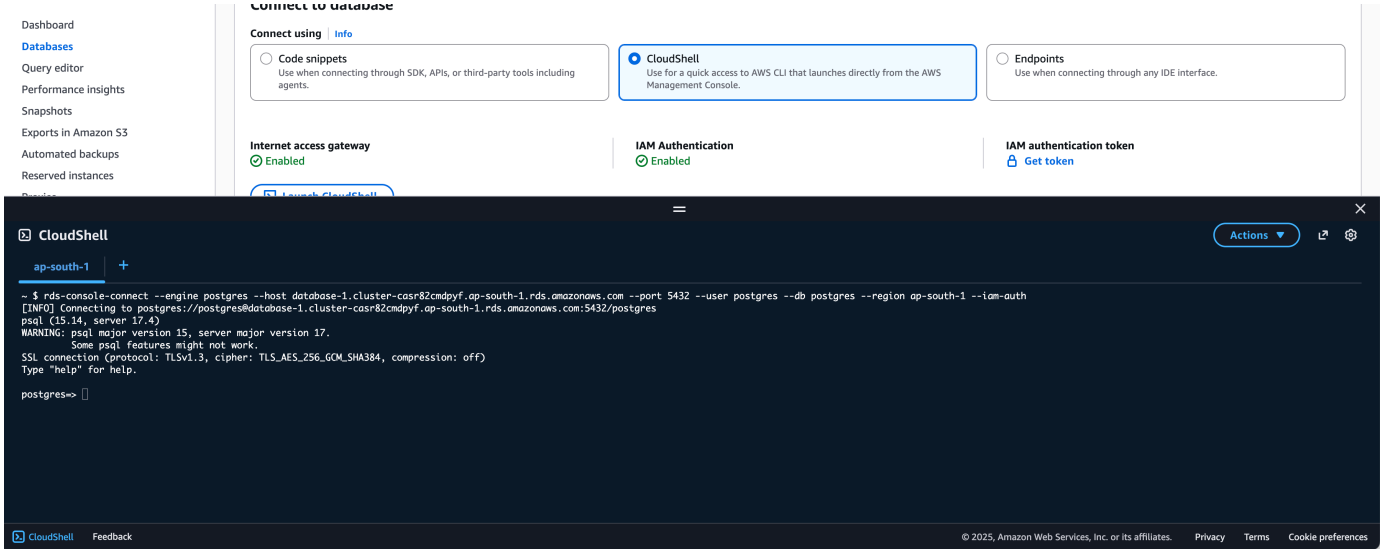
1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 從資料庫清單索引標籤導覽至您的資料庫叢集。
3. 在連線與安全索引標籤下，選取 CloudShell。
4. 按一下啟動 Cloudshell



5. 按一下在提示視窗上執行。命令會預先填入相關資訊，以連線至您的特定叢集。



6. 您應該會看到 psql 登入和「postgres =>」提示，讓您執行 SQL 命令。



使用 以快速存取直接從 AWS CLI 啟動的 AWS 管理主控台。CloudShell 提供以瀏覽器為基礎的 shell 環境，具有預先驗證的 AWS CLI 存取。

具有 Get Token 公用程式的端點

如果您使用的是僅支援使用使用者名稱和密碼登入資料的工具，您可以使用 Get Token 公用程式進行連線。在此情況下，您可以使用 公用程式在密碼欄位中產生的字符。系統會針對您在建立資料庫時設定的主要使用者名稱產生字符。字符一次有效時間為 15 分鐘。如果您使用的工具終止連線，您將需要再次產生字符。

下列步驟顯示如何使用 Get Token 公用程式，使用 pgAdmin 連線至資料庫叢集。

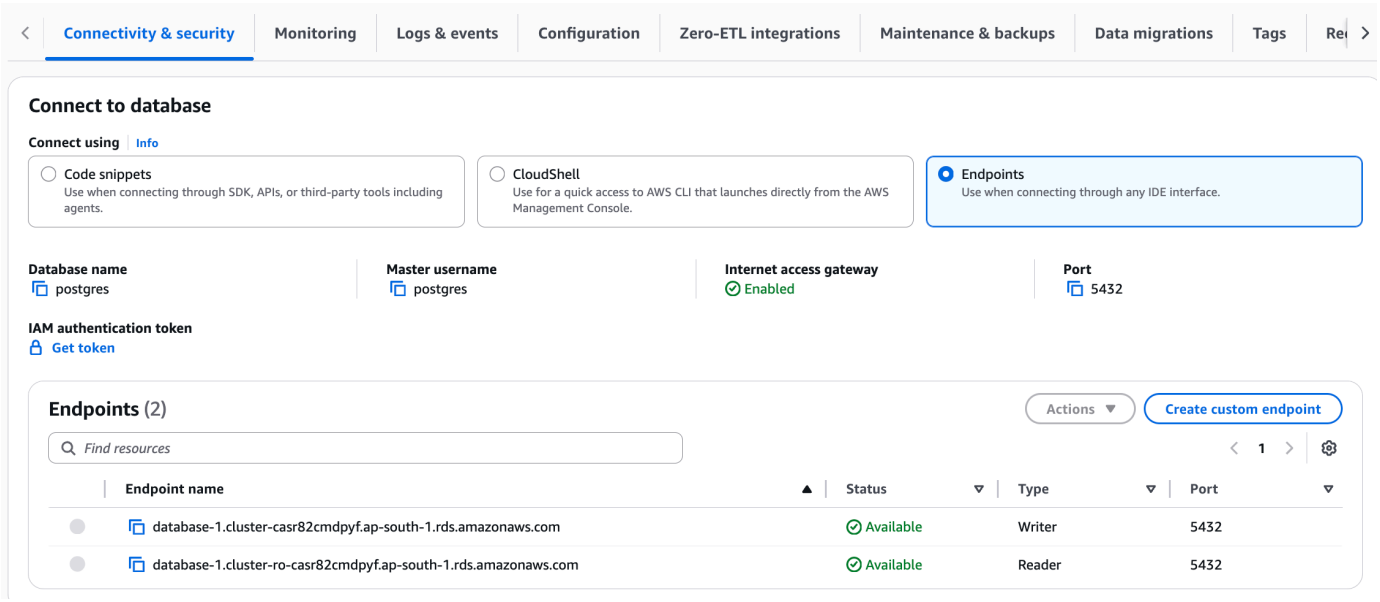
先決條件

- 您可以使用 pgAdmin 公用程式，透過 UI 界面連線至 Aurora PostgreSQL 資料庫叢集。如需詳細資訊，請參閱 pgAdmin 網站提供的下載頁面。

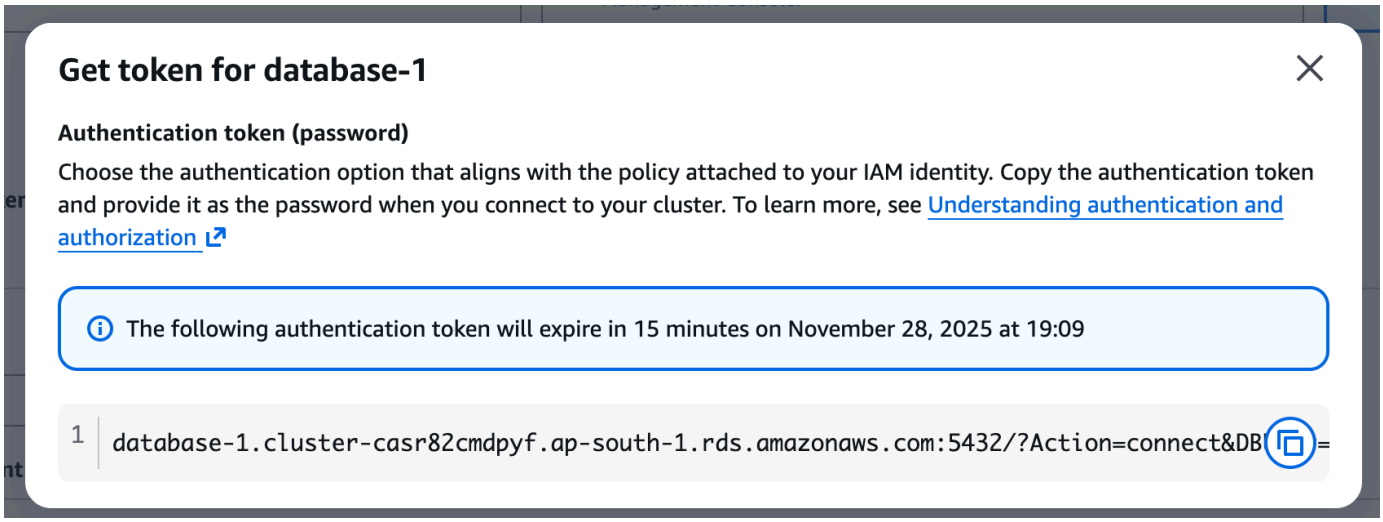
連線至 pgAdmin

使用 pgAdmin 連線

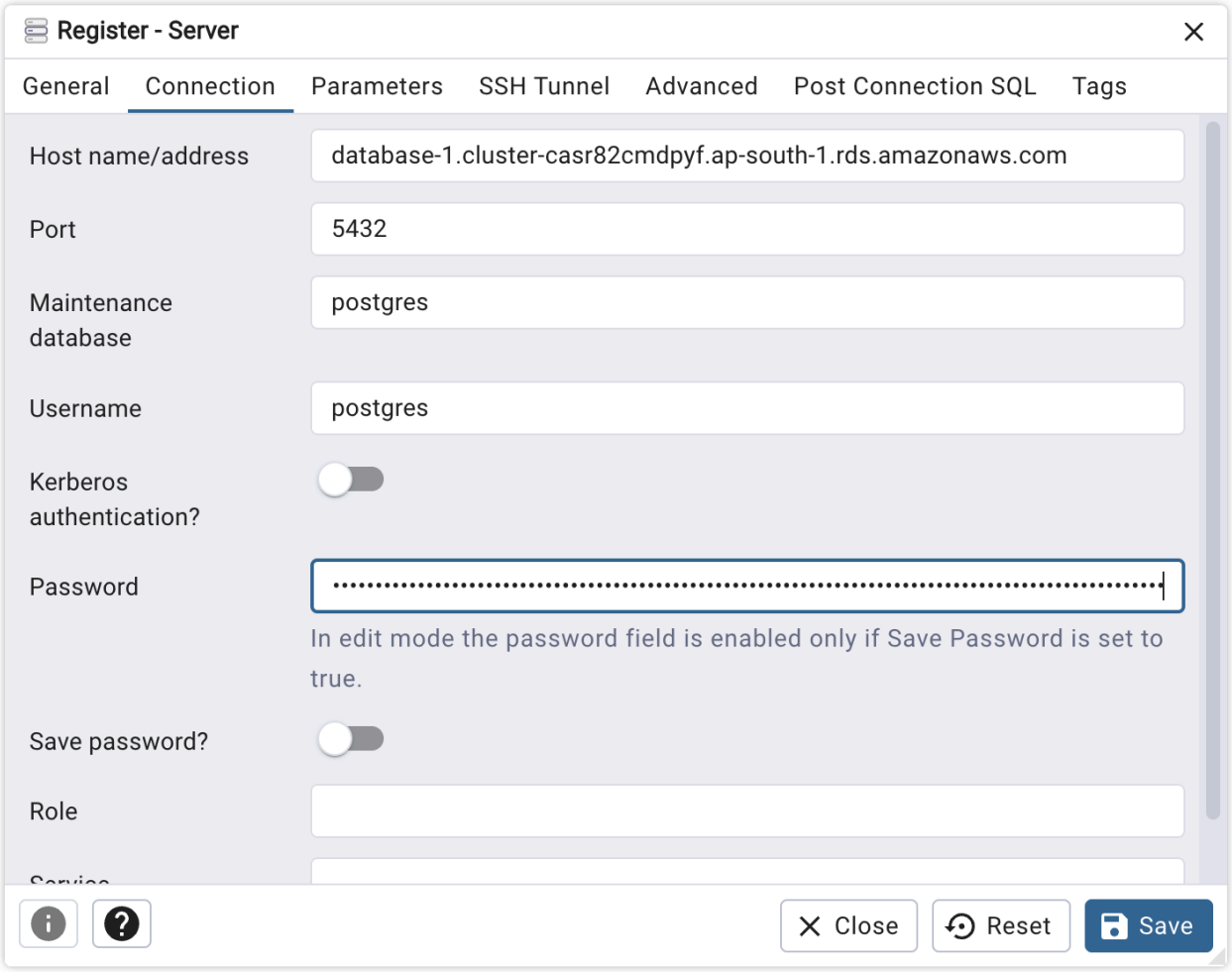
1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 從資料庫清單索引標籤導覽至您的資料庫叢集。
3. 在連線與安全索引標籤下，選取「端點」索引標籤。



4. 複製相關資訊，包括寫入器端點、連接埠、主要使用者名稱和資料庫名稱。
5. 從取得字符公用程式產生並複製新的字符（短期有效密碼）。



6. 在機器上開啟 pgAdmin，以及「新增伺服器」。在連線索引標籤中輸入詳細資訊。在密碼欄位中使用字符。



連線面板會顯示：網際網路存取閘道狀態（已啟用）、IAM 身分驗證狀態（已啟用）和 IAM 身分驗證字符選項。從下拉式清單中選取您的程式設計語言，然後依照連線步驟貼上程式碼並執行命令。

還原透過快速組態建立的叢集

快速組態叢集的還原程序遵循與標準叢集相同的工作流程。您可以使用 restore-db-cluster-from-snapshot 操作從手動快照還原，或使用 restore-db-cluster-to-point-in-time 操作還原至特定時間點。您可以將具有快速組態的叢集還原至具有完整組態的叢集，或具有快速組態的叢集。如果您想要還原至具有快速組態的叢集，這兩種還原操作都需要兩個額外的參數：VPCNetworkingEnabled 必須設定為 false，而 InternetAccessGatewayEnabled 必須設定為 true。如果您要將使用主要使用者名稱和密碼身分驗證的一般叢集還原至快速組態叢集，則需要修改還原的資料庫叢集，將 MasterUserAuthenticationType 更新為 iam-db-auth。

限制

下列限制適用於使用快速組態建立的叢集，以及預設啟用的網際網路存取閘道。

- 您只能搭配 Aurora PostgreSQL 使用快速組態。
- 使用快速組態建立的叢集會使用 AWS/RDS 服務受管金鑰加密。您無法使用自訂 AWS KMS 金鑰來加密資料庫。
- 您無法將快速叢集與 Amazon Virtual Private Cloud (VPC) 建立關聯。從 VPC 內部的機器連線至已啟用網際網路存取閘道的快速叢集資料庫時，請確定機器允許來自網際網路的傳入和傳出流量。
- 您無法停用使用快速組態建立之叢集的網際網路存取閘道。
- 您只能搭配網際網路存取閘道使用 RDS IAM 身分驗證。如需限制和建議的清單，請參閱 IAM 資料庫身分驗證。與其他身分驗證形式相關的功能，例如 AWS Secrets Manager 中的受管憑證，以及 Kerberos 身分驗證無法與已啟用網際網路存取閘道的叢集搭配使用。您也無法停用 IAM 身分驗證。

  > **Warning**
  >
  > 如果您停用主使用者名稱`rds_iam`的角色，您將無法存取資料庫的新連線。您可以透過修改叢集並將 設定為 `--master-user-authentication-type` 來還原存取權`iam-db-auth`。請參閱 modify-db-cluster。

- 您無法選取具有快速組態的特定引擎版本。引擎版本可以升級，但無法降級。
- 具有快速組態的叢集僅支援建立 Aurora Serverless v2 執行個體。您可以在建立資料庫之後修改執行個體類型或新增執行個體。
- 具有快速組態的叢集僅支援預設參數群組。您可以在建立資料庫後更新參數群組。

- 使用快速組態建立的叢集不支援下列功能，因為它們未與 VPC 相關聯。
  - Aurora 無限資料庫
  - Aurora 全球資料庫
  - RDS Proxy
  - Aurora 零 ETL 整合
  - RDS 查詢編輯器
  - 藍/綠部署
  - 資料庫活動串流
  - 零停機時間修補
  - Babelfish
- 使用 ModifyDBCluster 操作建立後，可為具有快速組態的叢集啟用資料 API。不過，它不支援使用主要使用者名稱/密碼進行身分驗證。您必須建立新的使用者登入資料才能存取資料 API。
- 使用 ModifyDBCluster 操作建立後，可以在具有快速組態的叢集上啟用 Database Insights 進階模式。
- 具有快速組態的叢集僅支援 IPv4 地址。不支援 IPv6。
- 具有快速組態的叢集不支援在建立操作時變更所有設定。例如，您只能在使用快速組態建立 時選擇 Aurora 標準儲存。您可以在建立叢集之後變更儲存類型。請參閱快速組態設定，以取得所有其他設定清單，以及適用於快速組態的預設值。
- 透過啟用網際網路存取閘道的快速組態建立的叢集不支援可設定的加密套件。雖然 Amazon RDS for PostgreSQL 可讓您設定 ssl_ciphers 參數來限制允許的 SSL 密碼套件，但在啟用網際網路存取閘道的叢集上會忽略此參數，這會改用預設密碼套件組態。

教學：建立 Web 伺服器和 Amazon Aurora 資料庫叢集

本教學課程說明如何安裝支援 PHP 的 Apache Web 伺服器，並建立 MariaDB、MySQL 或 PostgreSQL 資料庫。Web 伺服器會在使用 Amazon Linux 2023 的 Amazon EC2 執行個體上執行，而且您可以選擇 Aurora MySQL 或 Aurora PostgreSQL 資料庫叢集。Amazon EC2 執行個體和 資料庫叢集都會在以 Amazon VPC 服務為基礎的 Virtual Private Cloud (VPC) 中執行。

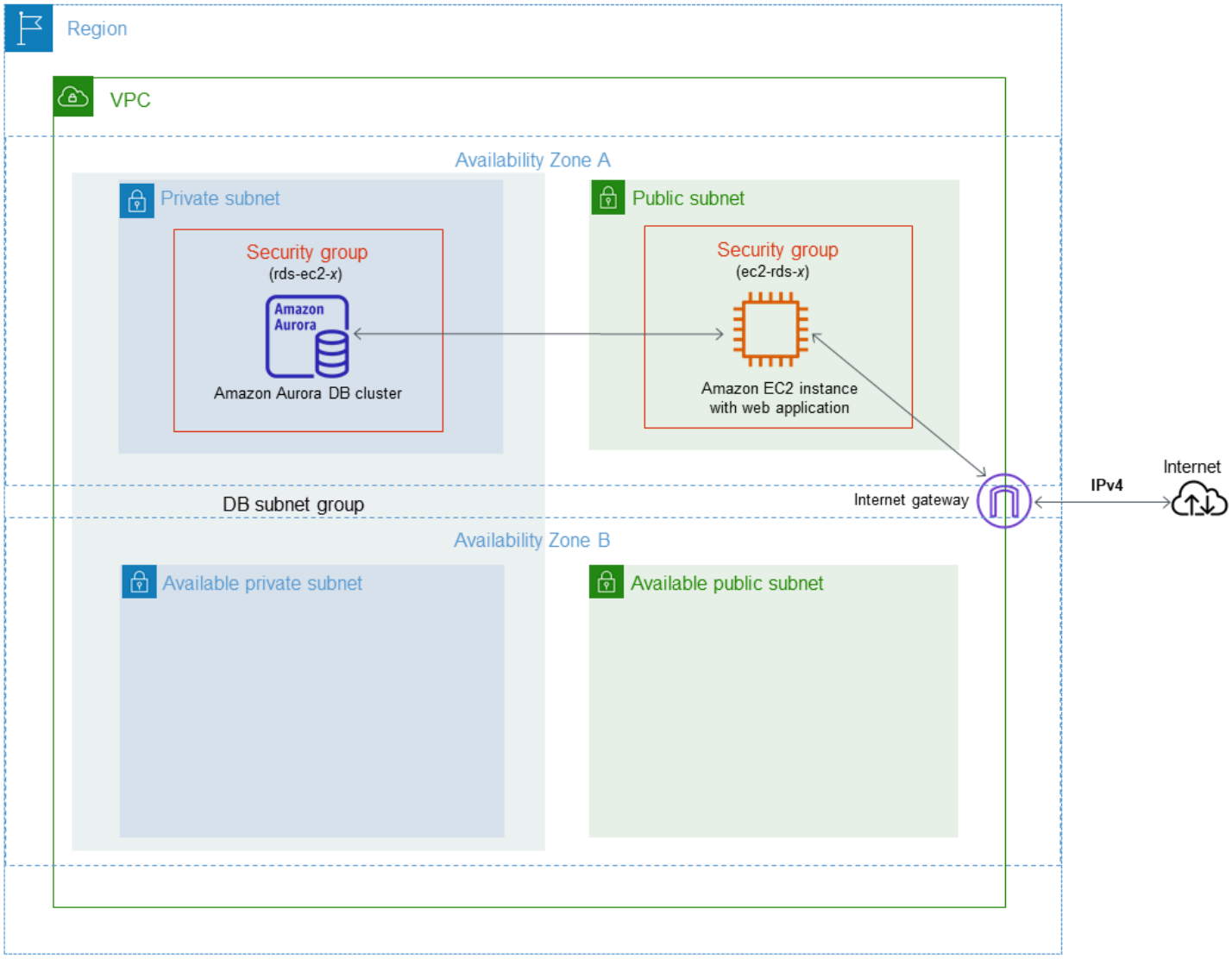
> **Important**
>
> 建立 AWS 帳戶無需付費。不過，透過完成本教學課程，您可能會因為使用 AWS 的資源而產生費用。如果不再需要這些資源，您可以在完成教學課程後刪除這些資源。

> **Note**
>
> 本教學課程使用 Amazon Linux 2023，可能不適用於其他 Linux 版本。

在後續的教學課程中，您會為 AWS 帳戶建立使用預設 VPC、子網路和安全群組的 EC2 執行個體。本教學課程向您介紹如何建立資料庫叢集，並自動設定與您建立的 EC2 執行個體的連線。然後，本教學課程將向您展示如何在 EC2 執行個體上安裝 Web 伺服器。您可以使用資料庫叢集寫入器端點，將 Web 伺服器連線到 VPC 中的資料庫叢集。

1. 啟動 EC2 執行個體以與資料庫叢集連線
2. 建立 Amazon Aurora 資料庫叢集
3. 在您的 EC2 執行個體上安裝 Web 伺服器

下圖顯示此教學課程完成時的組態。



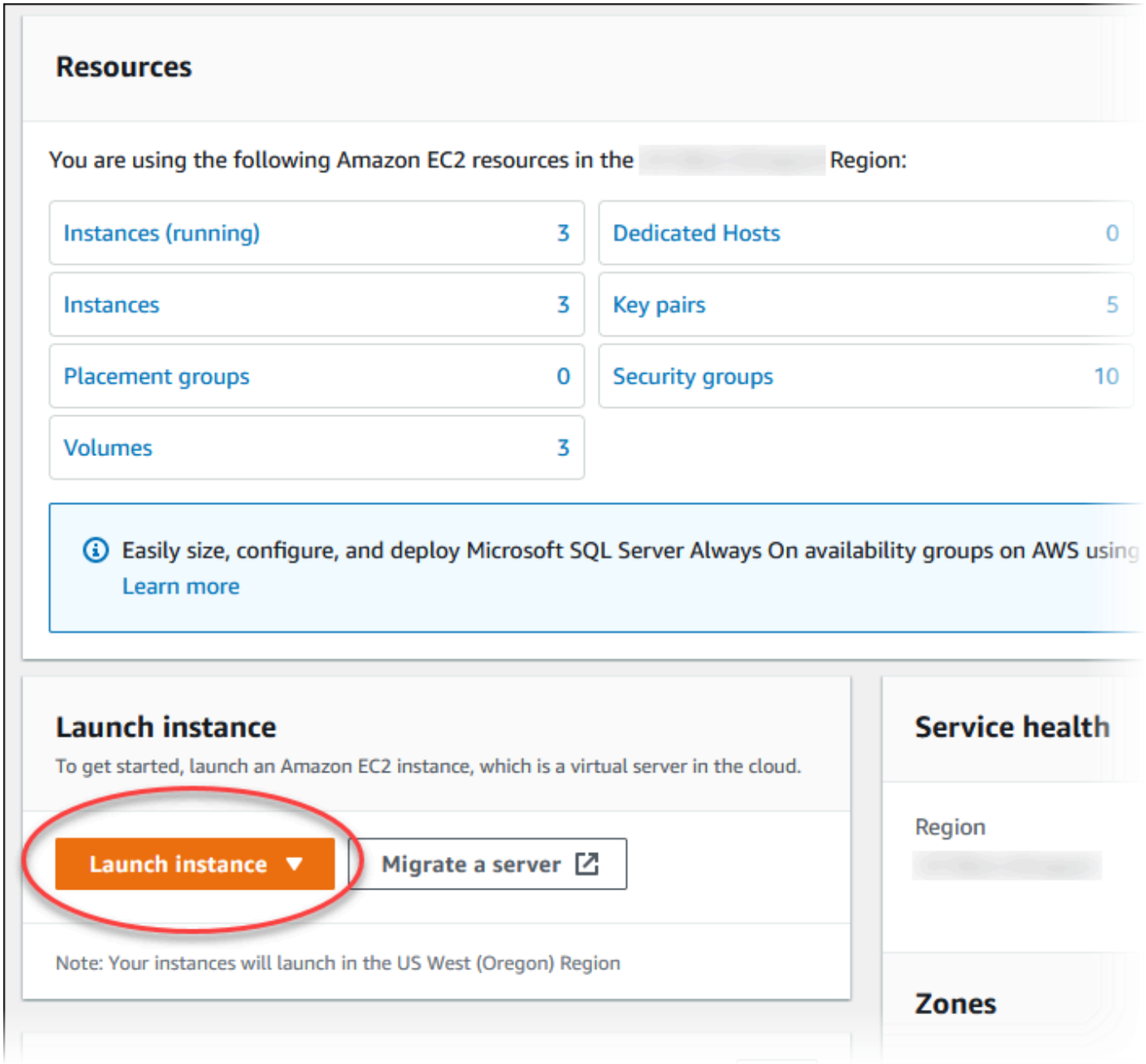
> **Note**
>
> 完成教學課程後，在您的 VPC 中，每個可用區域都有一個公有子網路和私有子網路。本教學課程使用 的預設 VPC， AWS 帳戶 並自動設定 EC2 執行個體與資料庫叢集之間的連線。如果您想改為針對此案例設定新的 VPC，請完成 教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4) 中的任務。

啟動 EC2 執行個體以與資料庫叢集連線

在 VPC 的公有子網路中建立 Amazon EC2 執行個體。

啟動 EC2 執行個體

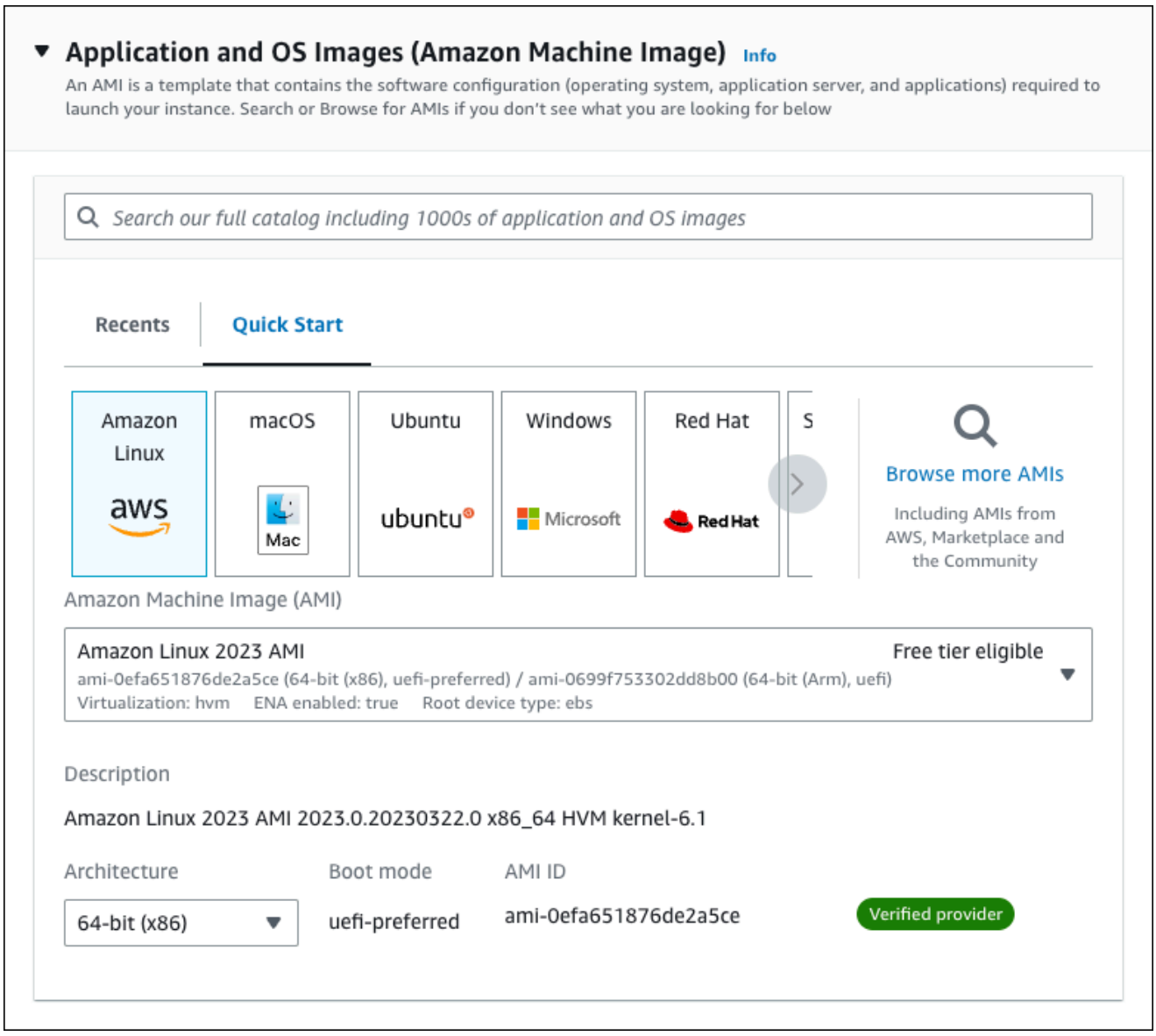
1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/ec2/ ：// 開啟 Amazon EC2 主控台。
2. 在 的右上角 AWS 管理主控台，選擇您要建立 EC2 執行個體的 AWS 區域 。
3. 選擇 EC2 Dashboard (EC2 儀表板)，然後選擇 Launch Instance (啟動執行個體)，如下所示。



4. 在啟動執行個體頁面中選擇下列設定。

a. 在 Name and tags (名稱與標籤) 下，對於 Name (名稱)，輸入 **`tutorial-ec2-instance-web-server`**。

b. 在應用程式和作業系統映像 (Amazon Machine Image) 中，選擇 Amazon Linux，然後選擇 Amazon Linux 2023 AMI。保留其他選項的預設值。



c. 在Instance type (執行個體類型) 下，選擇 t2.micro。

d. 在 Key pair (login) (金鑰對 (登入)) 下，選擇 Key pair name (金鑰對名稱)，以使用現有金鑰對。若要為 Amazon EC2 執行個體建立新的金鑰對，請選擇 Create new key pair (建立新的金鑰對)，然後使用 Create key pair (建立金鑰對) 視窗來建立金鑰對。

   如需有關建立新的金鑰對的詳細資訊，請參閱《Amazon EC2 使用者指南》中的建立金鑰對。

e. 在 Network settings (網路設定) 下設定這些值，其他值則維持預設值：

- 對於 Allow SSH traffic from (允許 SSH 流量來自)，選擇 EC2 執行個體做為 SSH 的連線來源。

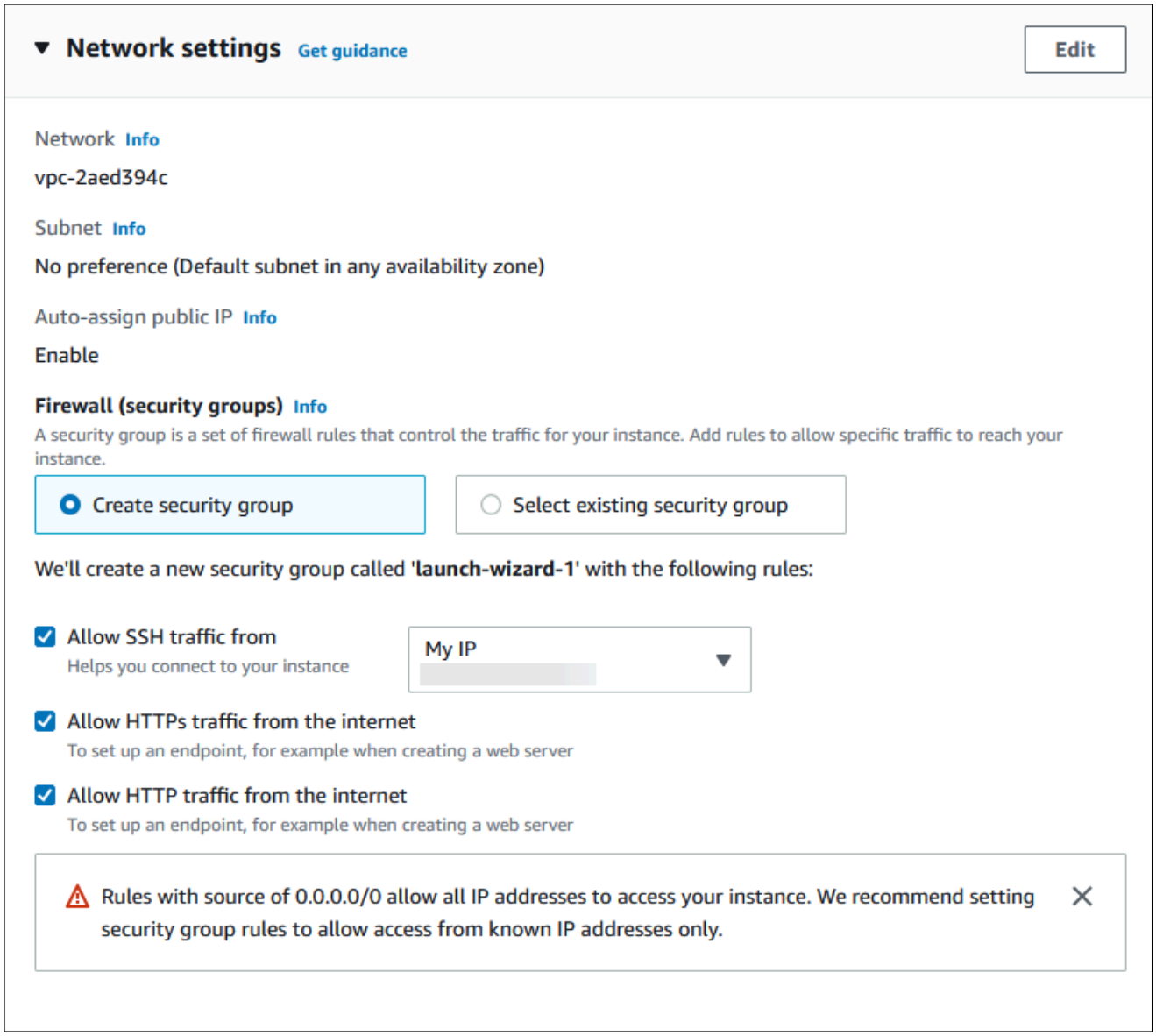
  如果顯示的 IP 地址對 SSH 連線而言是正確的，您可以選擇 My IP (我的 IP)。

  否則，您可以決定用於使用 Secure Shell (SSH) 連線至 VPC 中 EC2 執行個體的 IP 地址。若要判斷公有 IP 地址，您可以在不同的瀏覽器視窗或索引標籤中使用 https://checkip.amazonaws.com 中的服務。IP 地址的範例為 `203.0.113.25/32`。

  在許多情況下，您可能透過網際網路服務供應商 (ISP) 或是從沒有靜態 IP 地址的防火牆進行連線。若是如此，請務必確定用戶端電腦所使用的 IP 地址範圍。

  > ⚠ Warning
  >
  > 如果您使用 `0.0.0.0/0` 進行 SSH 存取，則可讓所有 IP 地址使用 SSH 存取您的公有執行個體。通常在測試環境中短暫使用此方法是沒有問題的，但在生產環境則不安全。在生產環境中，您應只授權特定 IP 地址或特定範圍的地址可使用 SSH 存取您的執行個體。

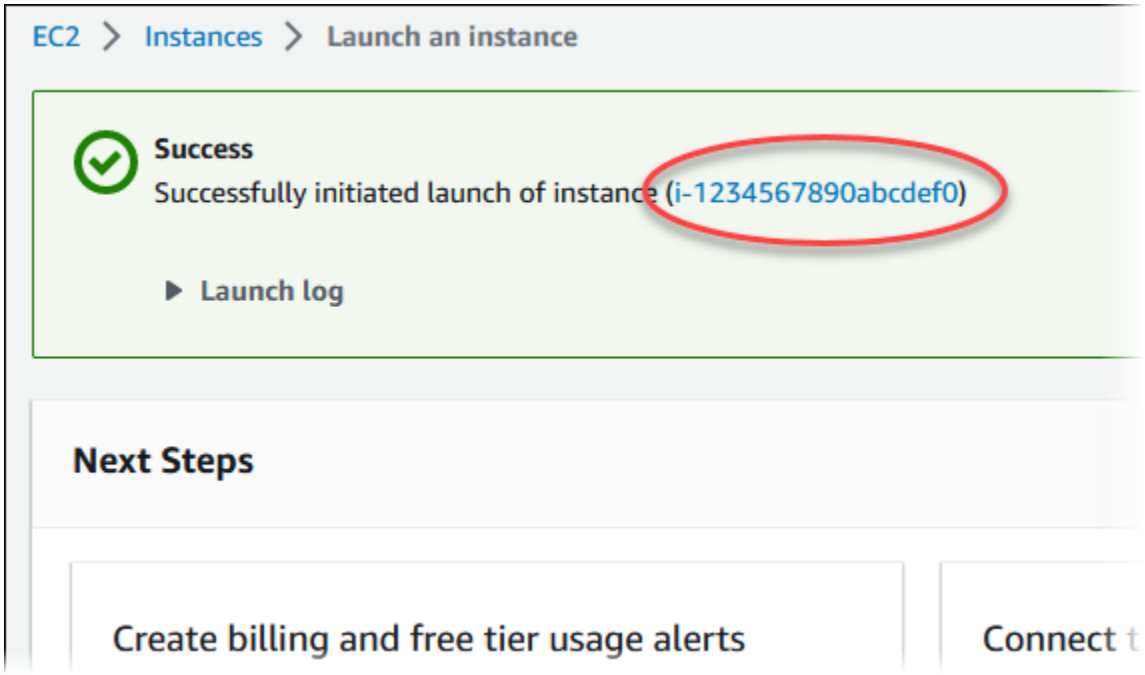
- 開啟 Allow HTTPs traffic from the internet (允許來自網際網路的 HTTPS 流量)。
- 開啟 Allow HTTP traffic from the internet (允許來自網際網路的 HTTP 流量)。



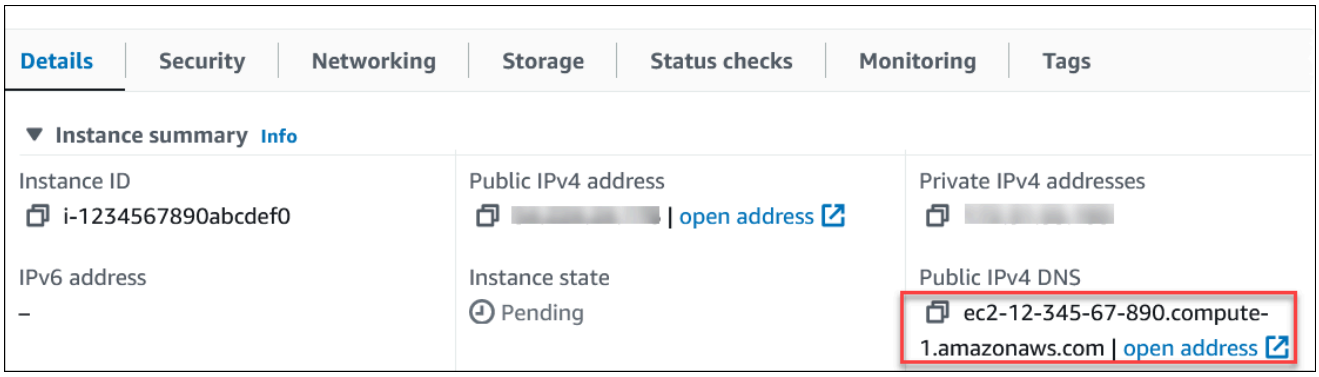
f. 讓剩餘區段保留預設值。

g. 檢閱 Summary (摘要) 面板中執行個體組態的摘要，並在準備就緒時選擇 Launch instance (啟動執行個體)。

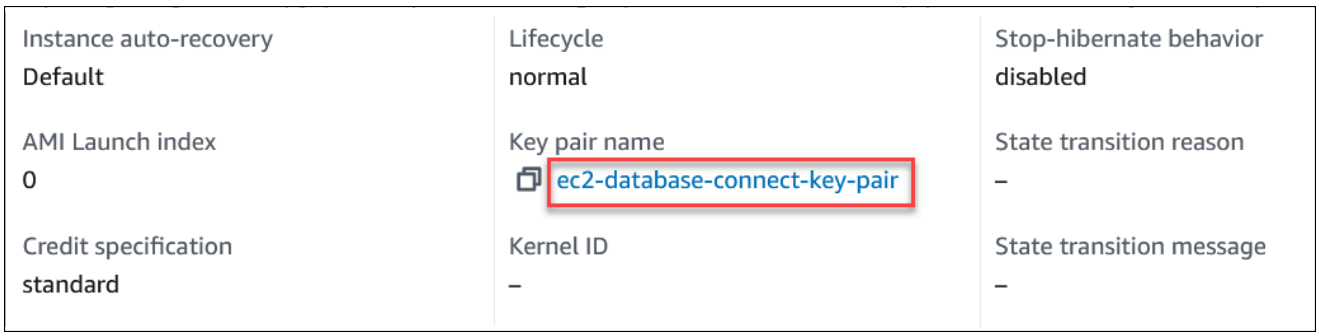
5. 在啟動狀態頁面上，記下新的 EC2 執行個體的識別碼，例如：`i-1234567890abcdef0`。



6. 選擇 EC2 執行個體識別符，以開啟 EC2 執行個體清單，然後選取您的 EC2 執行個體。
7. 在詳細資訊索引標籤中，請記下以下值，當您使用 SSH 進行連線時需要這些值：
   a. 在執行個體摘要中，記下公用 IPv4 DNS 的值。



   b. 在執行個體詳細資訊中，記下金鑰對名稱的值。



8. 繼續之前，等待執行個體的 Instance Status (執行個體狀態) 顯示為 Running (執行中)。

9. 完成 建立 Amazon Aurora 資料庫叢集。

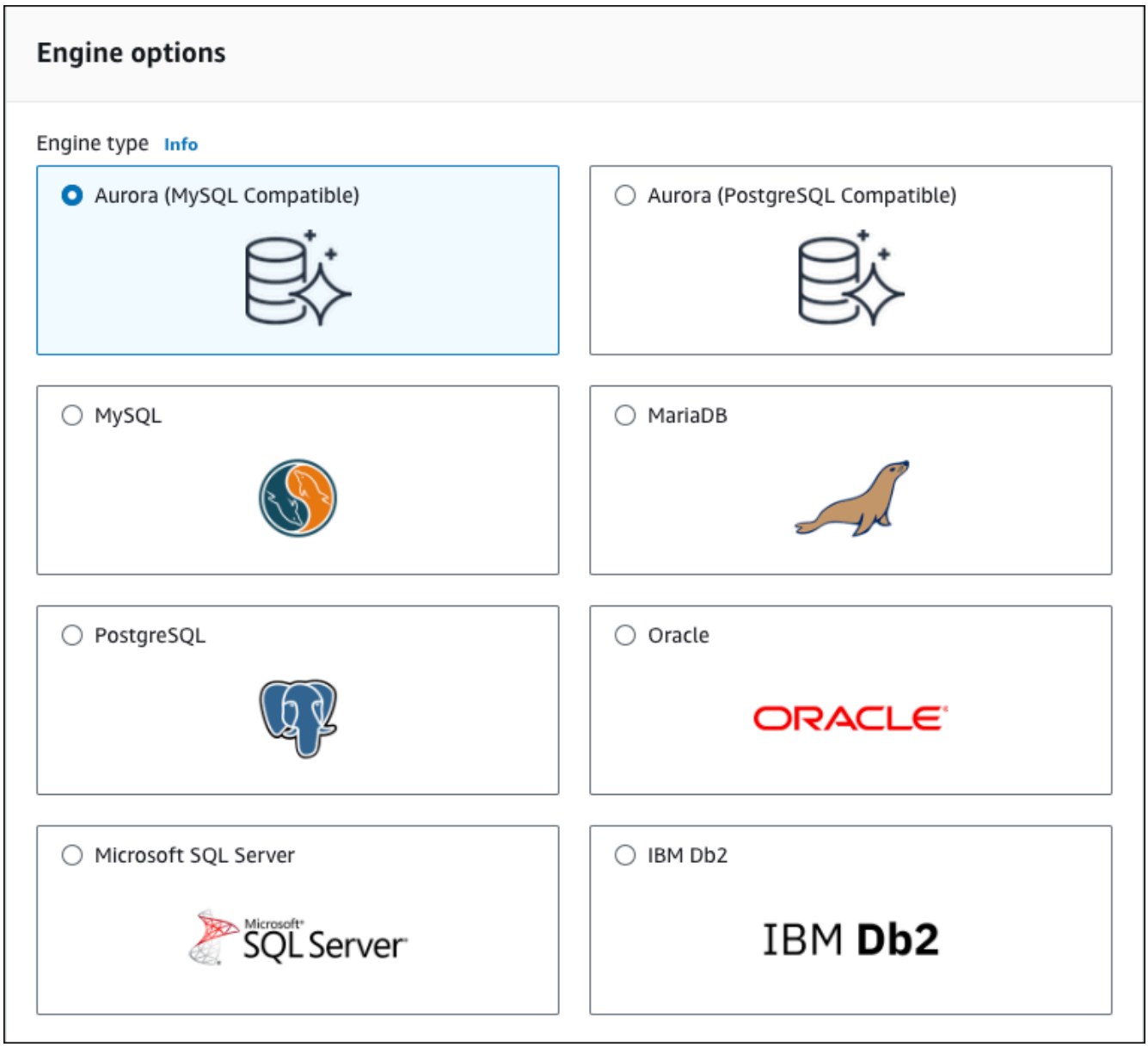
建立 Amazon Aurora 資料庫叢集

建立 Amazon Aurora MySQL 或 Aurora PostgreSQL 資料庫叢集，以維護 Web 應用程式所使用的資料。

Aurora MySQL

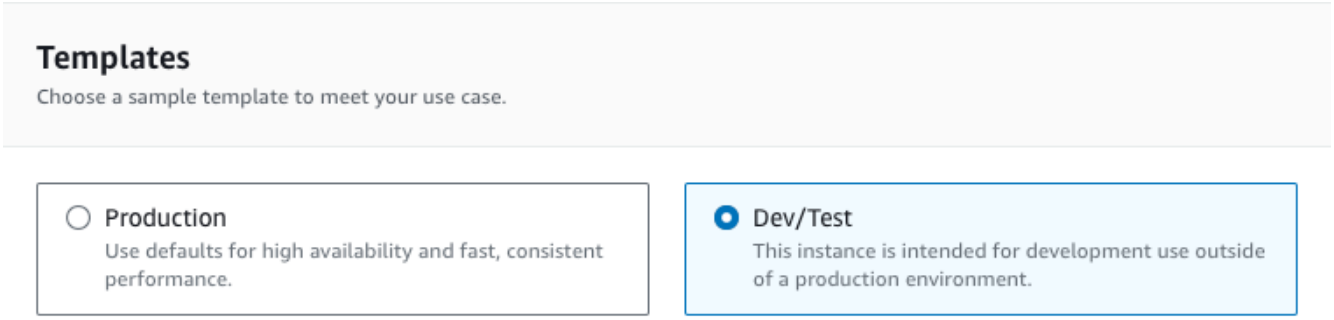
要建立 Aurora MySQL 資料庫叢集

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在 AWS 管理主控台 的右上角，請確定 AWS 區域 與您建立 EC2 執行個體時的區域相同。
3. 在導覽窗格中，選擇 Databases (資料庫)。
4. 選擇建立資料庫。
5. 在建立資料庫頁面上，選擇標準建立。
6. 針對引擎選項，請選擇 Aurora (MySQL 相容)。



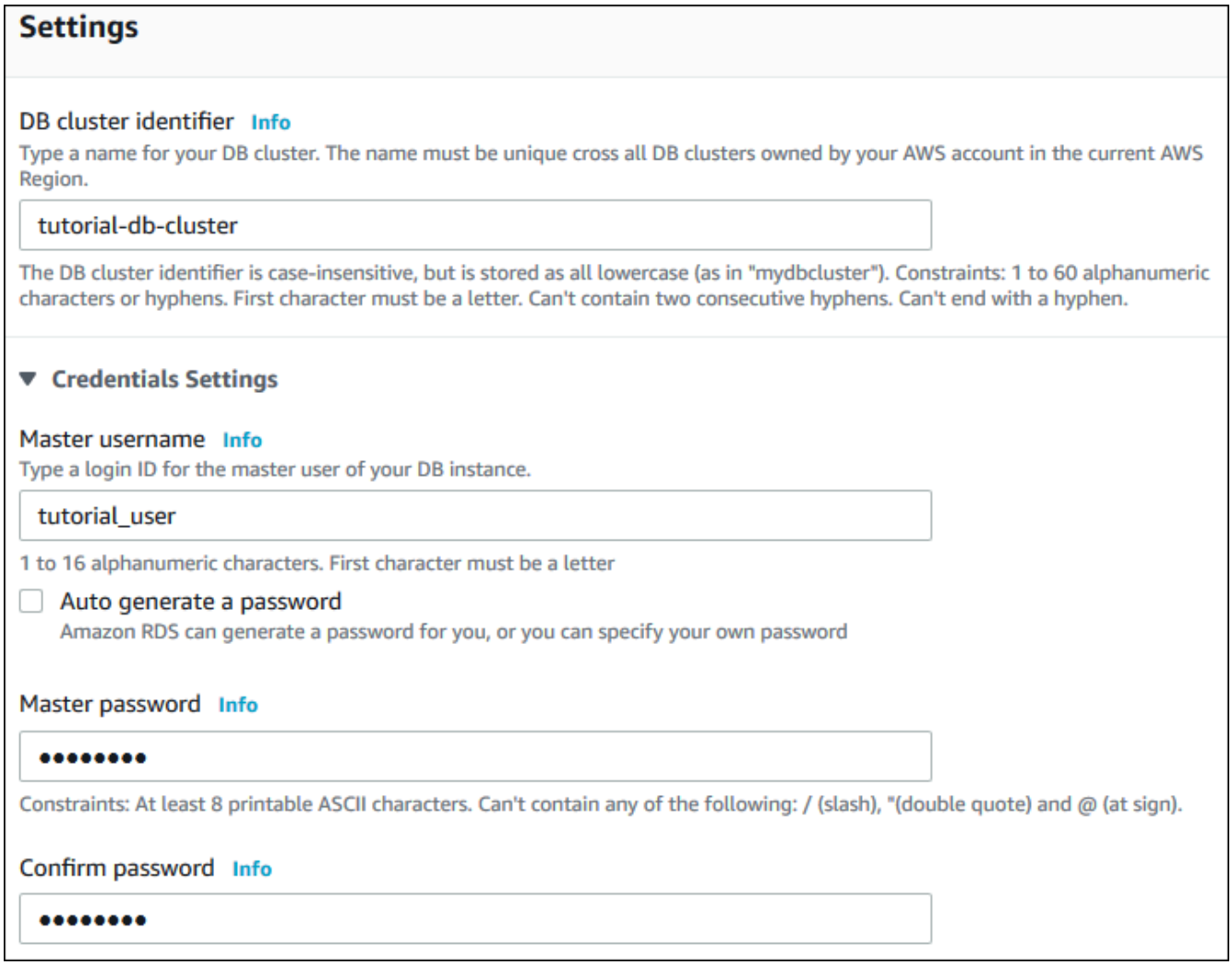
保留 Version (版本) 和其他引擎選項的預設值。

7. 在 Templates (範本) 區段中，選擇 Dev/Test (Dev/Test)。



8. 在 Settings (設定) 區段中，設定這些值：

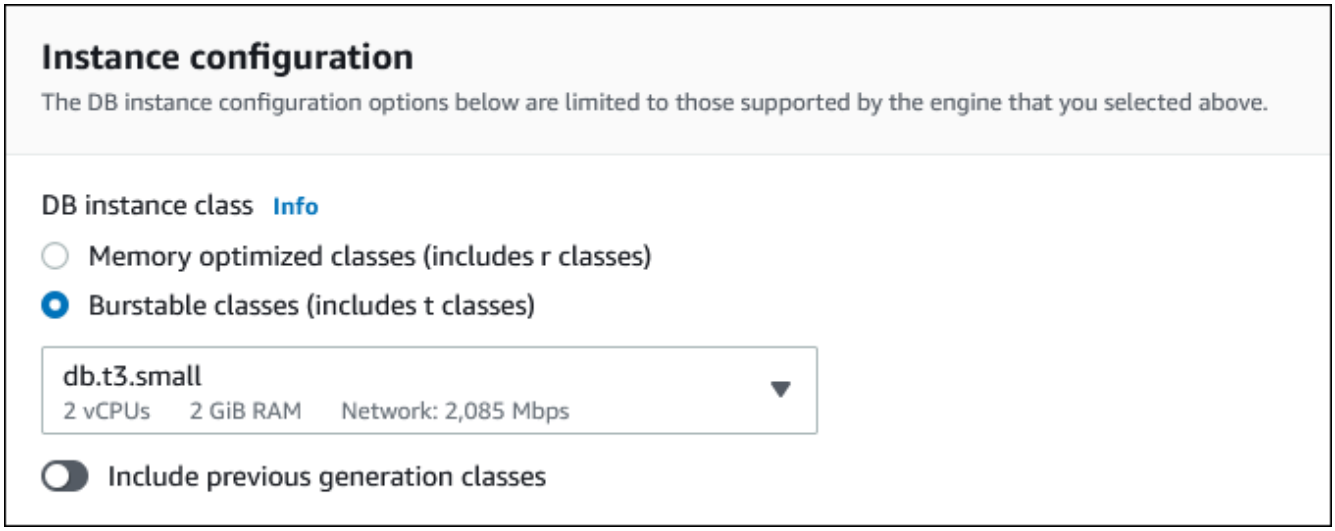
   - DB cluster identifier (資料庫叢集識別符) – 輸入 **`tutorial-db-cluster`**。
   - Master username (主要使用者名稱) – 輸入 **`tutorial_user`**。
   - Auto generate a password (自動產生密碼) – 保持選項關閉。
   - Master password (主要密碼) – 輸入密碼。
   - Confirm password (確認密碼) – 重新輸入密碼。



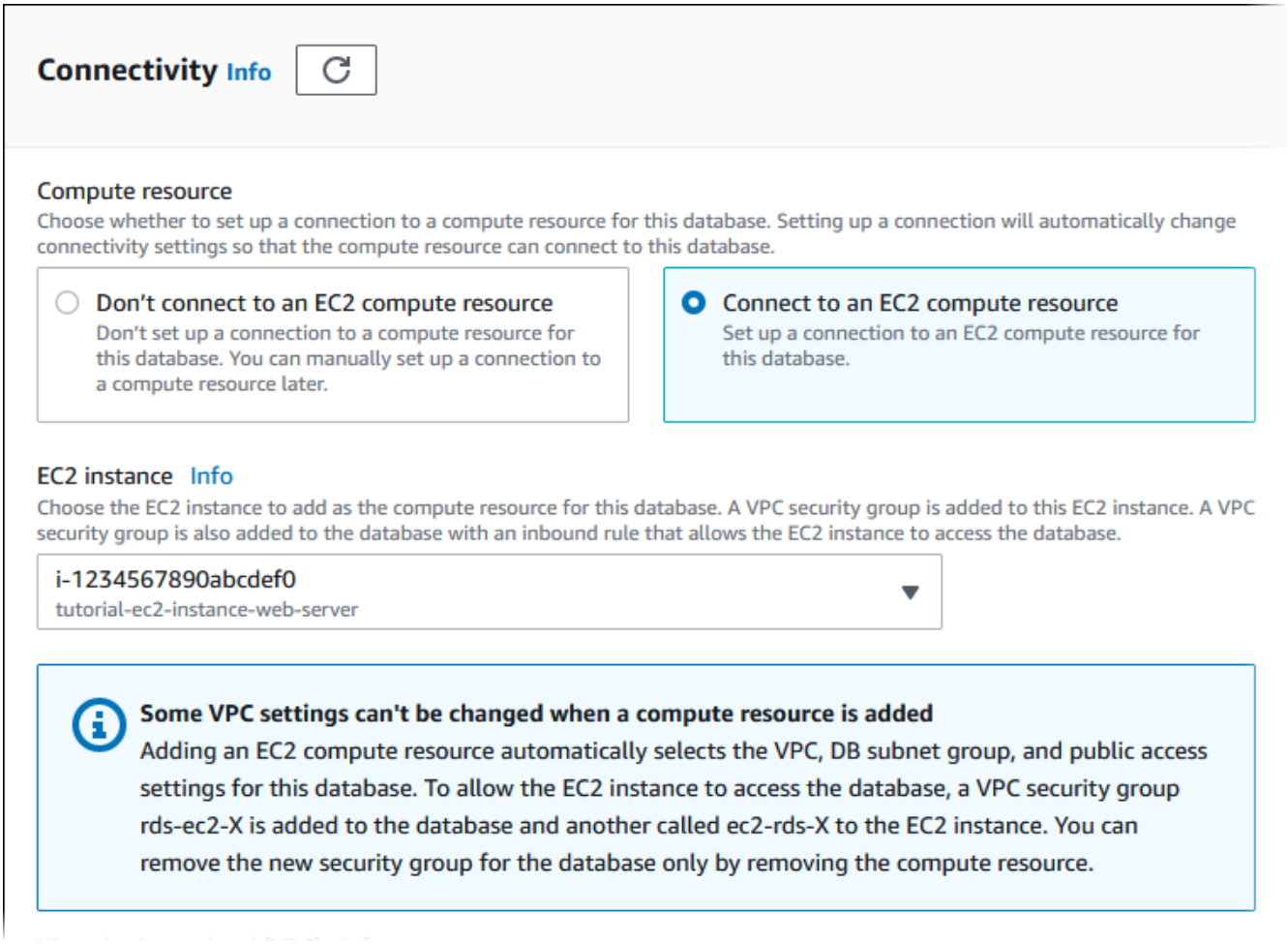
9. 在 Instance configuration (執行個體組態) 區段中，設定這些值：

   - Burstable classes (includes t classes) (高載類別 (包括 t 類別))
   - db.t3.small 或 db.t3.medium

> **Note**
>
> 建議您在開發、測試伺服器或其他非生產伺服器時，僅使用 T 資料庫執行個體類別。如需詳細了解 T 執行個體類別，請參閱 資料庫執行個體類別的類型。

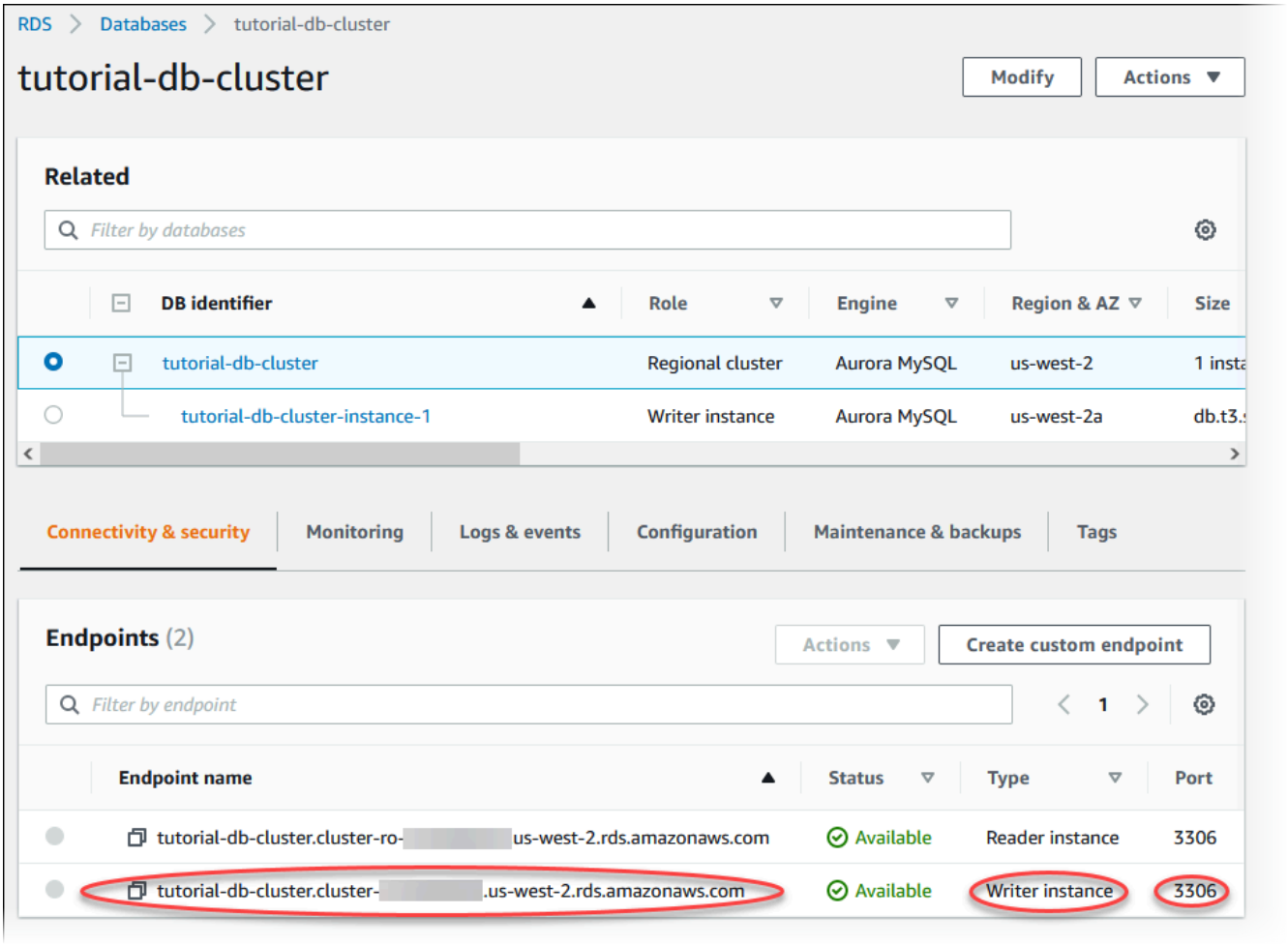


10. 在 Availability and durability (可用性與耐久性) 區段中，使用預設值。
11. 在 Connectivity (連線能力) 區段中，設定這些值，而其他值都維持預設值：
    - 對於 Compute resource (運算資源)，選擇 Connect to an EC2 compute resource (連線至 EC2 運算資源)。
    - 對於 EC2 instance (EC2 執行個體)，選擇之前建立的 EC2 執行個體，例如 tutorial-ec2-instance-web-server。



12. 在 Additional configuration (其他設定) 區段中，輸入 **`sample`** 做為 Initial database name (初始資料庫名稱)。其他選項都保留預設設定。
13. 若要建立 Aurora MySQL 資料庫叢集，請選擇 Create database (建立資料庫)。

    您的新資料庫叢集會出現在 Databases (資料庫) 清單中，狀態為 Creating (建立中)。
14. 等待新資料庫叢集的 Status (狀態) 顯示為 Available (可用)。接著，選擇資料庫叢集的名稱，以顯示其詳細資訊。
15. 在 Connectivity & security (連線能力與安全性) 區段中，檢視寫入器資料庫執行個體的 Endpoint (端點) 和 Port (連接埠)。



記下寫入器資料庫執行個體的端點和連接埠。您會使用此資訊，將 Web 伺服器連接至資料庫叢集。

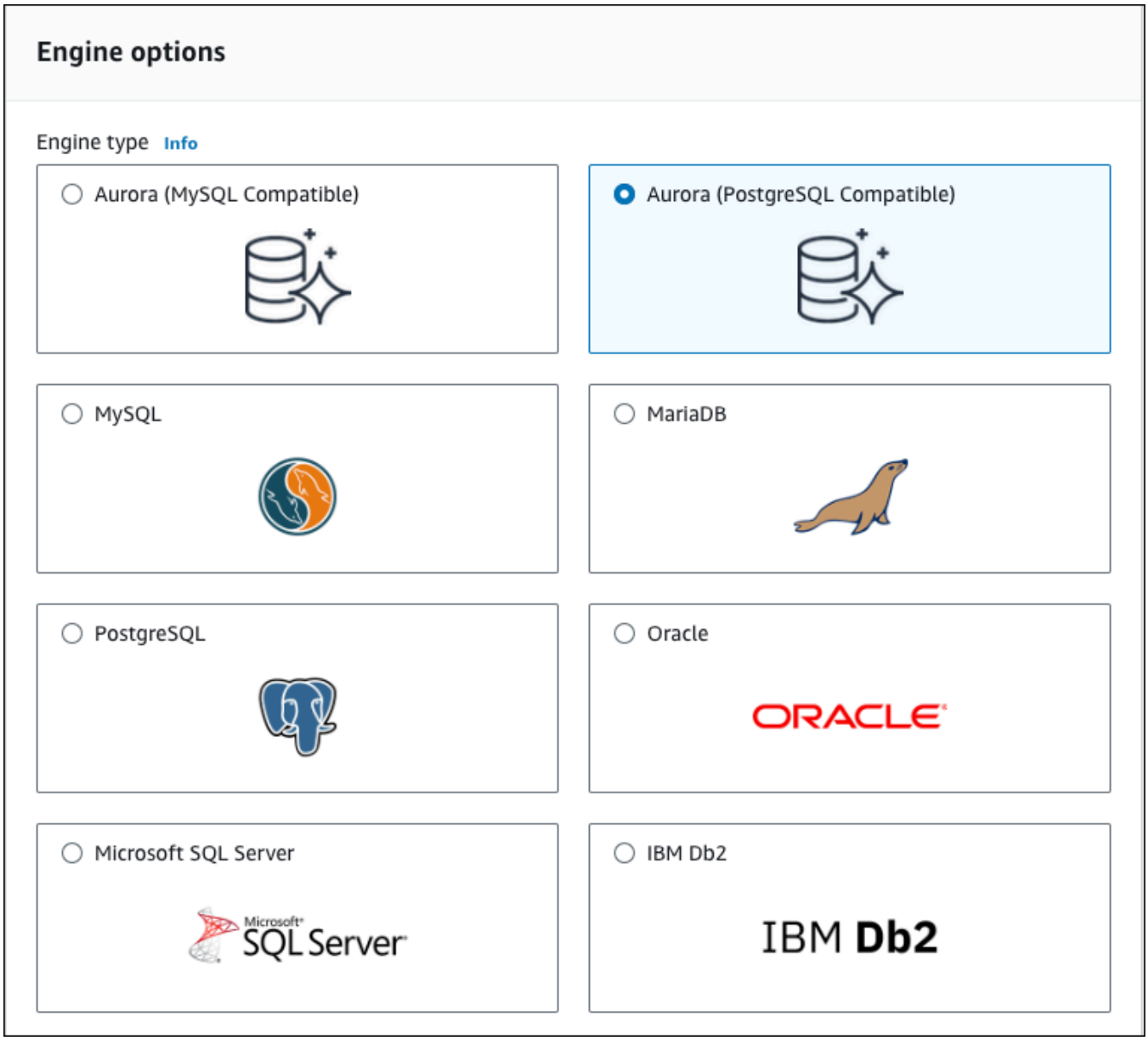
16. 完成 在您的 EC2 執行個體上安裝 Web 伺服器。

Aurora PostgreSQL

若要建立 Aurora PostgreSQL 資料庫叢集

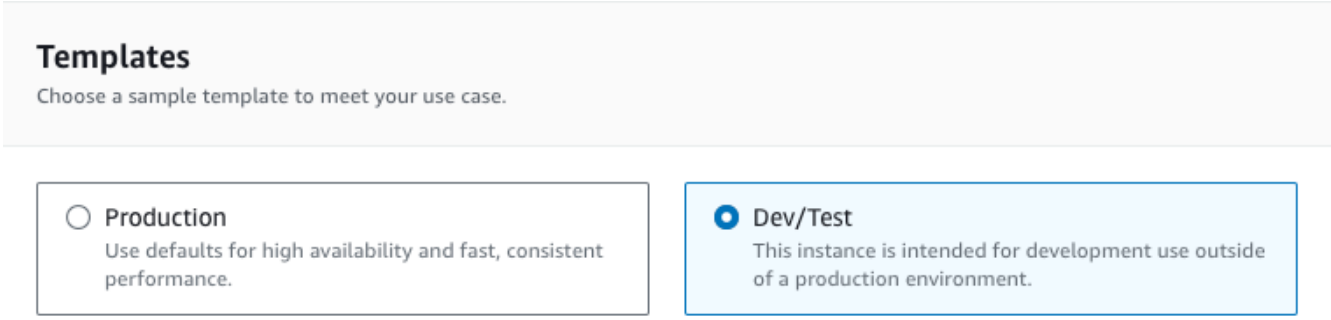
1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在 AWS 管理主控台 的右上角，請確定 AWS 區域 與您建立 EC2 執行個體時的區域相同。
3. 在導覽窗格中，選擇 Databases (資料庫)。
4. 選擇建立資料庫。
5. 在建立資料庫頁面上，選擇標準建立。

6. 針對引擎選項，選擇 Aurora (PostgreSQL 相容)。



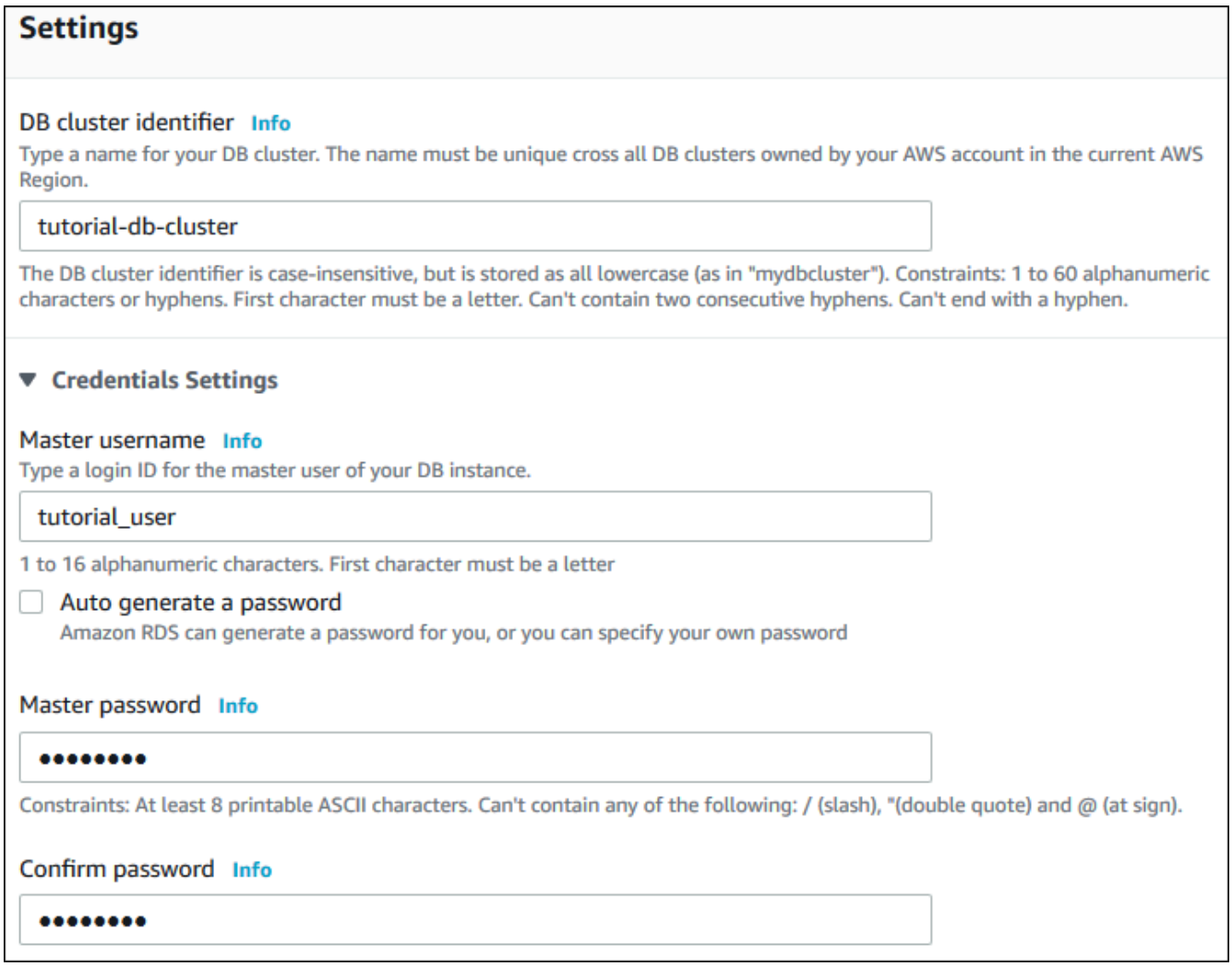
保留 Version (版本) 和其他引擎選項的預設值。

7. 在 Templates (範本) 區段中，選擇 Dev/Test (Dev/Test)。



8. 在 Settings (設定) 區段中，設定這些值：

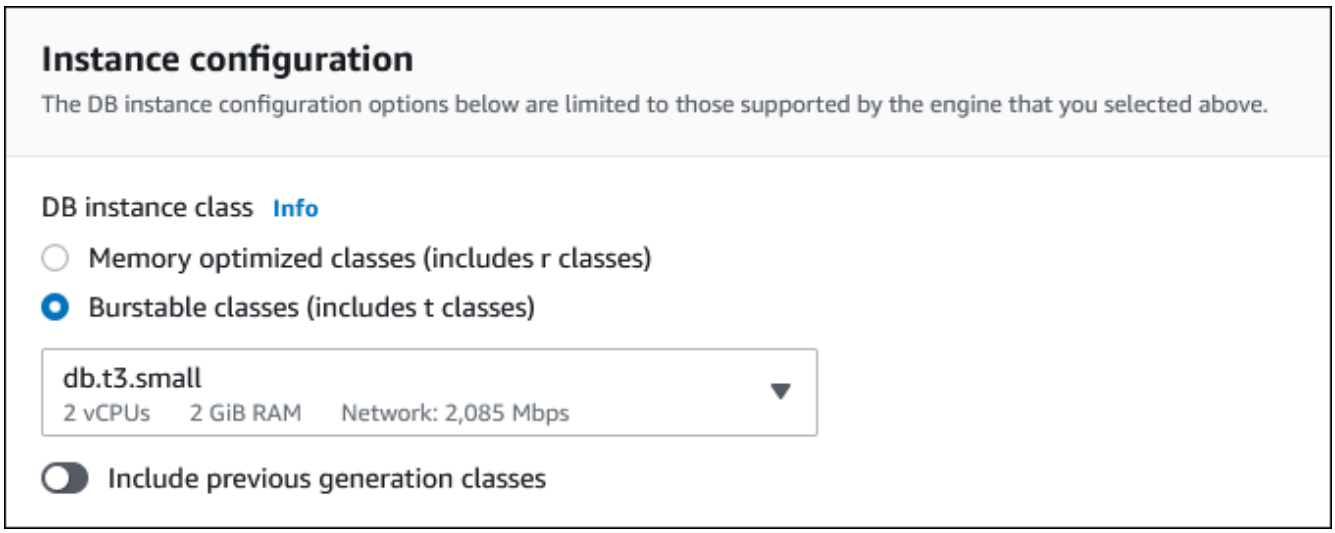
   - DB cluster identifier (資料庫叢集識別符) – 輸入 **`tutorial-db-cluster`**。
   - Master username (主要使用者名稱) – 輸入 **`tutorial_user`**。
   - Auto generate a password (自動產生密碼) – 保持選項關閉。
   - Master password (主要密碼) – 輸入密碼。
   - Confirm password (確認密碼) – 重新輸入密碼。



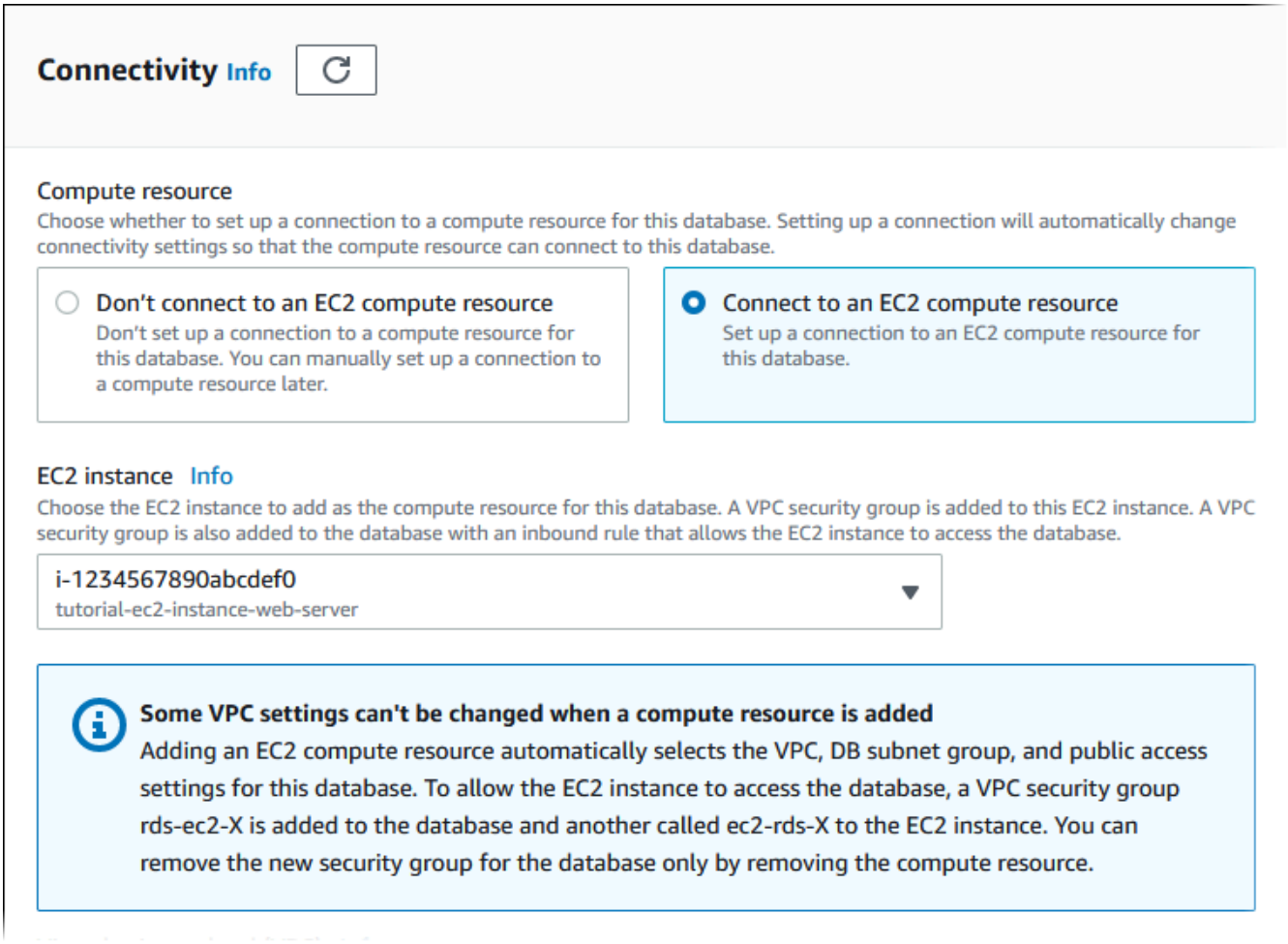
9. 在 Instance configuration (執行個體組態) 區段中，設定這些值：

   - Burstable classes (includes t classes) (高載類別 (包括 t 類別))
   - db.t3.small 或 db.t3.medium

   > **Note**
   >
   > 建議您在開發、測試伺服器或其他非生產伺服器時，僅使用 T 資料庫執行個體類別。如需詳細了解 T 執行個體類別，請參閱 資料庫執行個體類別的類型。

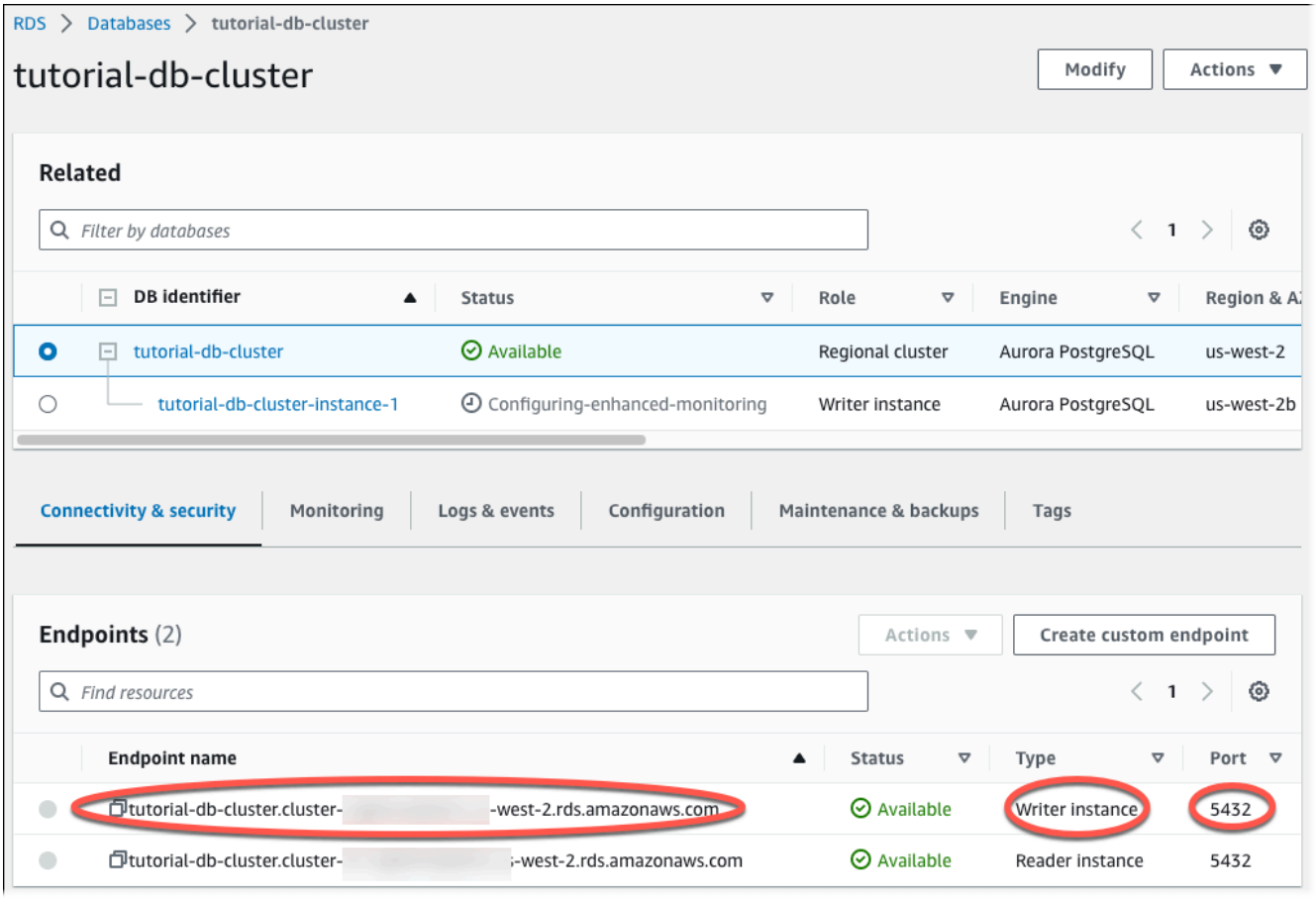


10. 在 Availability and durability (可用性與耐久性) 區段中，使用預設值。
11. 在 Connectivity (連線能力) 區段中，設定這些值，而其他值都維持預設值：
    - 對於 Compute resource (運算資源)，選擇 Connect to an EC2 compute resource (連線至 EC2 運算資源)。
    - 對於 EC2 instance (EC2 執行個體)，選擇之前建立的 EC2 執行個體，例如 tutorial-ec2-instance-web-server。



12. 在 Additional configuration (其他設定) 區段中，輸入 **`sample`** 做為 Initial database name (初始資料庫名稱)。其他選項都保留預設設定。
13. 若要建立 Aurora PostgreSQL 資料庫叢集，請選擇建立資料庫。

    您的新資料庫叢集會出現在 Databases (資料庫) 清單中，狀態為 Creating (建立中)。
14. 等待新資料庫叢集的 Status (狀態) 顯示為 Available (可用)。接著，選擇資料庫叢集的名稱，以顯示其詳細資訊。
15. 在 Connectivity & security (連線能力與安全性) 區段中，檢視寫入器資料庫執行個體的 Endpoint (端點) 和 Port (連接埠)。



記下寫入器資料庫執行個體的端點和連接埠。您會使用此資訊，將 Web 伺服器連接至資料庫叢集。

16. 完成 在您的 EC2 執行個體上安裝 Web 伺服器。

在您的 EC2 執行個體上安裝 Web 伺服器

您在 EC2; 中的 Linux 執行個體上安裝 Web 伺服器啟動 EC2 執行個體以與資料庫叢集連線。Web 伺服器將連接至在中建立的 Amazon Aurora 資料庫叢集建立 Amazon Aurora 資料庫叢集。

使用 PHP 和 MariaDB 安裝 Apache Web 伺服器

連線至 EC2 執行個體並安裝 Web 伺服器。

連接至 EC2 執行個體並安裝支援 PHP 的 Apache Web 伺服器

1. 請遵循《Amazon EC2 使用者指南》中的連線至 Linux 執行個體步驟，以連線您先前所建的 EC2 執行個體。

建議您使用 SSH 連線至 EC2 執行個體。如果 SSH 用戶端公用程式已安裝在 Windows、Linux 或 Mac 上，您可以使用下列命令格式連線至執行個體：

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

例如，假設 ec2-database-connect-key-pair.pem 存放在 Linux 上 的 /dir1 中，而 EC2 執行個體的公用 IPv4 DNS 為 ec2-12-345-678-90.compute-1.amazonaws.com。您的 SSH 命令看起來如下所示：

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

2. 更新 EC2 執行個體上的軟體，以取得最新的錯誤修正和安全性更新。若要執行此操作，請使用以下命令。

> **Note**
>
> -y 選項不要求確認就會安裝更新。若要先檢查更新再安裝，請省略此選項。

```
sudo dnf update -y
```

3. 更新完成後，使用下列命令安裝 Apache Web 伺服器、PHP、MariaDB 和 PostgreSQL 軟體。此命令會同時安裝多個軟體套件和相關的依存項目。

   MariaDB & MySQL

   ```
   sudo dnf install -y httpd php php-mysqli mariadb105
   ```

   PostgreSQL

   ```
   sudo dnf install -y httpd php php-pgsql postgresql15
   ```

   如果收到錯誤，您的執行個體可能不是以 Amazon Linux 2023 AMI 啟動。您可能正在改用 Amazon Linux 2 AMI。您可以使用下列命令來檢視您的 Amazon Linux 版本。

```
cat /etc/system-release
```

如需詳細資訊，請參閱更新執行個體軟體。

4. 使用如下所示的命令來啟動 Web 伺服器。

```
sudo systemctl start httpd
```

您可以測試 Web 伺服器是否已正確安裝並啟動。若要執行此動作，請在 Web 瀏覽器的網址列中輸入 EC2 執行個體的公有網域名稱系統 (DNS) 名稱，例如：`http://ec2-42-8-168-21.us-west-1.compute.amazonaws.com`。如果 Web 伺服器正在執行，您會看到 Apache 測試頁面。

如果您沒有看到 Apache 測試頁面，請檢查您在 教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4) 中建立的 VPC 安全性群組的輸入規則。請確定輸入規則包含允許 HTTP (連接埠 80) 存取連線至 Web 伺服器的 IP 地址的規則。

> **Note**
>
> 只有當文件根目錄 `/var/www/html` 中沒有內容時，Apache 測試頁面才會出現。當您將內容新增至文件根目錄之後，您的內容會出現在 EC2 執行個體的公有 DNS 地址。在此之前，它會出現在 Apache 測試頁面上。

5. 使用 `systemctl` 命令，設定 Web 伺服器在每次系統開機時啟動。

```
sudo systemctl enable httpd
```

若要允許 `ec2-user` 管理 Apache Web 伺服器之預設根目錄中的檔案，請修改 `/var/www` 目錄的所有權和許可。有多種方法可以達成這件任務。在本教學中，您會將 `ec2-user` 新增至 `apache` 群組，以向 `apache` 群組授予 `/var/www` 目錄的所有權，並指派寫入許可。

設定 Apache Web 伺服器的檔案許可

1. 將 `ec2-user` 使用者新增至 `apache` 群組。

```
sudo usermod -a -G apache ec2-user
```

2. 登出以重新整理許可並包含新的 `apache` 群組。

```
exit
```

3. 重新登入並使用 apache 命令來確認 groups 群組存在。

```
groups
```

您的輸出結果類似如下：

```
ec2-user adm wheel apache systemd-journal
```

4. 將 /var/www 目錄及其內容的群組所有權變更至 apache 群組。

```
sudo chown -R ec2-user:apache /var/www
```

5. 變更 /var/www 及其子目錄的目錄許可，以新增群組寫入許可，並設定未來建立之子目錄的群組 ID。

```
sudo chmod 2775 /var/www
find /var/www -type d -exec sudo chmod 2775 {} \;
```

6. 對 /var/www 目錄及其子目錄中的檔案，遞迴地變更許可，以新增群組寫入許可。

```
find /var/www -type f -exec sudo chmod 0664 {} \;
```

現在，ec2-user (以及 apache 群組的任何未來成員) 可以新增、刪除和編輯 Apache 文件根目錄中的檔案。這可讓您新增內容，例如靜態網站或 PHP 應用程式。

> **Note**
>
> 執行 HTTP 通訊協定的 Web 伺服器不會為其傳送或接收的資料提供傳輸安全性。當您使用 Web 瀏覽器連線到 HTTP 伺服器時，網路路徑上任何一處的竊聽者都可以看到許多資訊。此資訊包括您造訪的 URL、您收到的網頁內容，以及任何 HTML 表單內容 (包括密碼)。
> 保護您的 Web 伺服器的最佳實務是安裝對 HTTPS (HTTP Secure) 的支援。此通訊協定會使用 SSL/TLS 加密保護您的資料。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的教學課程：使用 Amazon Linux AMI 設定 SSL/TLS。

將 Apache Web 伺服器連接至資料庫個體

接下來，您需要將內容新增至 Apache Web 伺服器 (連接到 Amazon Aurora 資料庫叢集)。

將內容新增至連接到資料庫叢集的 Apache Web 伺服器

1. 在仍然連接至 EC2 執行個體的情況下，切換至 `/var/www` 目錄，建立名為 `inc` 的新子目錄。

```
cd /var/www
mkdir inc
cd inc
```

2. 在 `inc` 目錄中建立名為 `dbinfo.inc` 的新檔案，然後呼叫 nano (或您選擇的編輯器) 來編輯此檔案。

```
>dbinfo.inc
nano dbinfo.inc
```

3. 將下列內容新增至 `dbinfo.inc` 檔案：在此處，*`db_instance_endpoint`* 是資料庫叢集寫入器端點，沒有連接埠，適用於您的資料庫叢集。

> **Note**
>
> 建議您將使用者名稱和密碼資訊放在不屬於 Web 伺服器文件根目錄的資料夾中。這樣做會降低您暴露安全性資訊的可能性。
>
> 確定在您的應用程式中將 `master password` 變更為合適的密碼。

```
<?php

define('DB_SERVER', 'db_cluster_writer_endpoint');
define('DB_USERNAME', 'tutorial_user');
define('DB_PASSWORD', 'master password');
define('DB_DATABASE', 'sample');
?>
```

4. 儲存並關閉 `dbinfo.inc` 檔案。若您使用 nano，請以 Ctrl+S 加 Ctrl+X，儲存並關閉檔案。
5. 將目錄切換至 `/var/www/html`。

```
cd /var/www/html
```

6. 在 html 目錄中建立名為 SamplePage.php 的新檔案，然後呼叫 nano (或您選擇的編輯器) 來編輯此檔案。

```
>SamplePage.php
nano SamplePage.php
```

7. 將下列內容新增至 SamplePage.php 檔案：

MariaDB & MySQL

```
<?php include "../inc/dbinfo.inc"; ?>
<html>
<body>
<h1>Sample page</h1>
<?php

  /* Connect to MySQL and select the database. */
  $connection = mysqli_connect(DB_SERVER, DB_USERNAME, DB_PASSWORD);

  if (mysqli_connect_errno()) echo "Failed to connect to MySQL: " .
 mysqli_connect_error();

  $database = mysqli_select_db($connection, DB_DATABASE);

  /* Ensure that the EMPLOYEES table exists. */
  VerifyEmployeesTable($connection, DB_DATABASE);

  /* If input fields are populated, add a row to the EMPLOYEES table. */
  $employee_name = htmlentities($_POST['NAME']);
  $employee_address = htmlentities($_POST['ADDRESS']);

  if (strlen($employee_name) || strlen($employee_address)) {
    AddEmployee($connection, $employee_name, $employee_address);
  }
?>

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
```

```
    <tr>
      <td>
        <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
        <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
        <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
</form>

<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = mysqli_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = mysqli_fetch_row($result)) {
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>

</table>

<!-- Clean up. -->
<?php

  mysqli_free_result($result);
  mysqli_close($connection);

?>
```

```
</body>
</html>


<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = mysqli_real_escape_string($connection, $name);
   $a = mysqli_real_escape_string($connection, $address);

   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

   if(!mysqli_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
}

/* Check whether the table exists and, if not, create it. */
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID int(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";

     if(!mysqli_query($connection, $query)) echo("<p>Error creating table.</
p>");
  }
}

/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = mysqli_real_escape_string($connection, $tableName);
  $d = mysqli_real_escape_string($connection, $dbName);

  $checktable = mysqli_query($connection,
      "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME = '$t'
 AND TABLE_SCHEMA = '$d'");

  if(mysqli_num_rows($checktable) > 0) return true;
```

```
  return false;
}
?>
```

PostgreSQL

```
<?php include "../inc/dbinfo.inc"; ?>

<html>
<body>
<h1>Sample page</h1>
<?php

/* Connect to PostgreSQL and select the database. */
$constring = "host=" . DB_SERVER . " dbname=" . DB_DATABASE . " user=" .
 DB_USERNAME . " password=" . DB_PASSWORD ;
$connection = pg_connect($constring);

if (!$connection){
 echo "Failed to connect to PostgreSQL";
 exit;
}

/* Ensure that the EMPLOYEES table exists. */
VerifyEmployeesTable($connection, DB_DATABASE);

/* If input fields are populated, add a row to the EMPLOYEES table. */
$employee_name = htmlentities($_POST['NAME']);
$employee_address = htmlentities($_POST['ADDRESS']);

if (strlen($employee_name) || strlen($employee_address)) {
  AddEmployee($connection, $employee_name, $employee_address);
}

?>

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
```

```
    </tr>
    <tr>
      <td>
    <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
    <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
    <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
</form>
<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = pg_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = pg_fetch_row($result)) {
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>
</table>

<!-- Clean up. -->
<?php

  pg_free_result($result);
  pg_close($connection);
?>
</body>
</html>
```

```
<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = pg_escape_string($name);
   $a = pg_escape_string($address);
   echo "Forming Query";
   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

   if(!pg_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
}

/* Check whether the table exists and, if not, create it. */
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID serial PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";

     if(!pg_query($connection, $query)) echo("<p>Error creating table.</p>");
  }
}
/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = strtolower(pg_escape_string($tableName)); //table name is case sensitive
  $d = pg_escape_string($dbName); //schema is 'public' instead of 'sample' db
 name so not using that

  $query = "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME =
 '$t';";
  $checktable = pg_query($connection, $query);

  if (pg_num_rows($checktable) >0) return true;
  return false;

}
?>
```

8. 儲存並關閉 SamplePage.php 檔案。
9. 開啟 Web 瀏覽器並瀏覽至 http://*EC2 instance endpoint*/SamplePage.php，以驗證 Web 伺服器是否成功連接至叢集，例如：http://ec2-12-345-67-890.us-west-2.compute.amazonaws.com/SamplePage.php。

您可以使用 SamplePage.php 將資料新增至資料庫叢集。您新增的資料將會顯示在頁面上。若要驗證資料是否已插入資料表中，請在 Amazon EC2 執行個體上安裝 MySQL 用戶端。然後連線至資料庫叢集，並查詢資料表。

如需連接至資料庫叢集的相關資訊，請參閱連接至 Amazon Aurora 資料庫叢集。

為確保資料庫叢集盡可能安全，請確認 VPC 以外的來源無法連線到資料庫叢集。

測試完 Web 伺服器和資料庫之後，您應該刪除資料庫叢集和 Amazon EC2 執行個體。

- 若要刪除資料庫叢集，請遵循 刪除 Aurora 資料庫叢集和資料庫執行個體 中的指示。您不需要建立最終快照。
- 若要終止 Amazon EC2 執行個體。請按照《Amazon EC2 使用指南》中終止執行個體的指示進行操作。

Amazon Aurora 教學課程和範本程式碼

AWS 文件包含數個教學課程，引導您完成常見的 Amazon Aurora 使用案例。其中許多教學課程會示範如何將 Amazon Aurora 與其他 AWS 服務搭配使用。此外，您可以在中存取範本程式碼。

> **Note**
>
> 您可以在 AWS 資料庫部落格找到更多教學課程。如需培訓的詳細資訊，請參閱 AWS 培訓與認證。

主題

- 本指南中的教學課程
- 其他 AWS 指南中的教學課程
- GitHub 中的教學課程和範本程式碼
- AWS 資料庫技術指南
- AWS Amazon Aurora PostgreSQL 的研討會和實驗室內容入口網站
- AWS Amazon Aurora MySQL 的研討會和實驗室內容入口網站
- 搭配 AWS SDK 使用此服務

本指南中的教學課程

本指南中的以下教學課程會向您展示如何使用 Amazon Aurora 執行常見的任務。

- 教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4)

  了解如何在以 Amazon VPC 服務為基礎的虛擬私有雲端 (VPC) 中包括資料庫叢集。在此情況下，VPC 會與在相同 VPC 的 Amazon EC2 執行個體上執行的 Web 伺服器共用資料。

- 教學課程：建立要與資料庫叢集搭配使用的 (VPC)(雙堆疊模式)

  了解如何在以 Amazon VPC 服務為基礎的虛擬私有雲端 (VPC) 中包括資料庫叢集。在此情況下，VPC 會與相同 VPC 中的 Amazon EC2 執行個體共用資料。於本教學課程中，您將為此案例建立與以雙堆疊模式執行之資料庫搭配使用的 VPC。

- 教學：建立 Web 伺服器和 Amazon Aurora 資料庫叢集

了解如何安裝支援 PHP 的 Apache Web 伺服器，並建立 MySQL 資料庫。Web 伺服器在使用 Amazon Linux 的 Amazon EC2 執行個體上執行，而 MySQL 資料庫是 ，Aurora MySQL 資料庫叢集。Amazon EC2 執行個體和資料庫叢集皆在 Amazon VPC 中執行。

- 教學：從資料庫叢集快照還原 Amazon Aurora 資料庫叢集

  了解如何使用從資料庫叢集快照還原資料庫叢集。

- 教學：使用標籤指定要停止哪些 Aurora 資料庫叢集

  了解如何使用標籤指定要停止哪些 Aurora 資料庫叢集。

- 教學課程：使用 Amazon EventBridge 來記錄資料庫執行個體狀態變更

  了解如何使用 Amazon EventBridge 和 記錄資料庫執行個體狀態變更 AWS Lambda。

其他 AWS 指南中的教學課程

其他 AWS 指南中的下列教學課程說明如何使用 Amazon Aurora 執行常見任務：

> **Note**
>
> 一些教學課程使用了 Amazon RDS 資料庫執行個體，但它們可以適應使用 Aurora 資料庫叢集。

- AWS AppSync 開發人員指南中的教學課程：Aurora Serverless

  了解如何使用 AWS AppSync 為已啟用資料 API Aurora Serverless 的資料庫叢集提供執行 SQL 命令的資料來源。您可以使用 AWS AppSync 解析程式，使用 GraphQL 查詢、變動和訂閱針對 Data API 執行 SQL 陳述式。

- AWS Secrets Manager 《使用者指南》中的教學課程：輪換 AWS 資料庫的秘密

  了解如何建立 AWS 資料庫的秘密，並設定秘密以排程輪換。手動觸發一個輪換，然後確認秘密新版本持續提供存取。

- AWS Elastic Beanstalk 開發人員指南中的教學課程和範例

  了解如何部署使用 Amazon RDS 資料庫的應用程式 AWS Elastic Beanstalk。

- 在 Amazon Machine Learning Developer Guide 中使用 Amazon RDS 資料庫中的資料建立 Amazon ML 資料來源

了解如何從存放在 MySQL 資料庫執行個體的資料中建立 Amazon Machine Learning (Amazon ML) 資料來源物件。

- 《Amazon Quick 使用者指南》中的在 VPC 中手動啟用對 Amazon RDS 執行個體的存取

  了解如何在 VPC 中啟用 Amazon RDS 資料庫執行個體的快速存取。

GitHub 中的教學課程和範本程式碼

GitHub 中的以下教學課程和範本程式碼會向您展示如何使用 Amazon Aurora 執行常見的任務：

- 建立 Aurora Serverless v2 出借圖書館

  了解如何建立出借圖書館應用程式，讓顧客可以借書與還書。此範例使用 Aurora Serverless v2和 適用於 Python (Boto3) 的 AWS SDK。

- 使用查詢的 Spring REST API 建立 Amazon Aurora 項目追蹤應用程式，以使用 SDK for Java 2.x 查詢 Aurora Serverless v2 的資料

  了解如何建立一個查詢 Aurora Serverless v2 資料的 Spring REST API。它是供使用 SDK for Java 2.x 的 React 應用程式使用。

- 建立使用 查詢Aurora Serverless v2資料的 Amazon Aurora 項目追蹤器應用程式 適用於 PHP 的 AWS SDK

  了解如何建立一個使用資料 API 的 `RdsDataClient` 和 Aurora Serverless v2 追蹤和報告工作項目的應用程式。此範例使用 適用於 PHP 的 AWS SDK。

- 建立 Amazon Aurora 項目追蹤器應用程式來查詢使用 適用於 Python (Boto3) 的 AWS SDK查詢 Aurora Serverless v2資料

  了解如何建立一個使用資料 API 的 `RdsDataClient` 和 Aurora Serverless v2 追蹤和報告工作項目的應用程式。此範例使用 適用於 Python (Boto3) 的 AWS SDK。

AWS 資料庫技術指南

AWS 資料庫技術指南是全方位的資料庫指南，說明如何建置、部署和管理高效能、符合成本效益的資料庫解決方案 AWS。逐步教學課程會引導您建立生產就緒的應用程式，以及使用 CloudFormation 範本部署應用程式。您將在建置基礎設施、實作聯網、開發無伺服器架構、管理資料庫和整合生成式 AI 時學習基本 AWS 服務。了解 AWS 最佳實務，協助您建立安全、可擴展的解決方案，同時最佳化成

本。無論您是初次接觸 AWS 還是經驗豐富的專業人員， AWS 資料庫技術指南都可協助您開發技能，以解決常見的資料庫挑戰並實作企業就緒的解決方案。技術指南包含下列章節：

- AWS 開始使用適用於 資料庫應用程式的 - 了解如何設定您的帳戶和 Jupyter 筆記本環境等 AWS 基本概念。
- 資料庫基礎知識 – 探索基本資料庫概念並比較 AWS 資料庫服務，為您的工作負載選擇正確的解決方案。
- 使用 Amazon Aurora 的無伺服器 Web 應用程式：使用 Amazon Aurora PostgreSQL 建置端對端零售應用程式，以處理庫存、訂單和客戶資料。
- 監控和可觀測性：設定效能追蹤和設定警示，以在潛在資料庫問題影響應用程式之前加以識別。
- 使用 Amazon Aurora 進行擴展：了解如何使用 Aurora DSQL 建置彈性多區域部署，以及如何擴展資料庫以增加處理能力，或跨多個執行個體擴展以增加容量。
- 最佳化效能和成本：使用經驗證的調校策略最佳化資料庫效能與降低成本。
- 前往 AWS 專用資料庫的旅程 – 建立安全可靠的基礎設施，將生成式 AI 解決方案和資料驅動型應用程式從原型擴展到企業部署。
- 使用 RAG 的 GenAI 應用程式：為保險和醫療保健文件建置智慧搜尋系統，此系統會使用檢索增強生成 (RAG) 來提供準確、內容感知的結果。

AWS Amazon Aurora PostgreSQL 的研討會和實驗室內容入口網站

以下精選的研討會和其他實作內容可協助您了解 Amazon Aurora PostgreSQL 的特性和功能：

- 手動建立新的 Aurora 叢集

  了解如何手動建立 Amazon Aurora PostgreSQL 叢集。
- 設定 Cloud9 和初始化資料庫

  了解如何設定 Cloud9 和初始化 PostgreSQL 資料庫。
- 快速複製

  了解如何建立 Aurora 快速複製。
- 查詢計畫管理

  了解如何使用查詢計畫管理來控制一組陳述式的執行計畫。
- 叢集快取管理

了解 Aurora PostgreSQL 中的叢集快取功能。

- 資料庫活動串流

  了解如何使用此功能監控與稽核資料庫活動。

- 使用績效詳情

  了解如何使用績效詳情來監控和調整資料庫執行個體。

- 使用 RDS 工具進行效能監控

  了解如何使用 AWS 和 Postgres 工具 (Cloudwatch、增強型監控、慢查詢日誌、績效詳情、PostgreSQL 目錄檢視） 來了解效能問題，並找出改善資料庫效能的方法。

- 自動擴展僅供讀取複本

  了解 Aurora 僅供讀取複本自動擴展在使用負載產生器指令碼時實際上如何運作。

- 測試容錯

  了解資料庫叢集如何容忍失敗。

- Aurora 全球資料庫

  了解 Aurora 全球資料庫。

- 使用機器學習

  了解 Aurora Machine Learning。

- Aurora Serverless v2

  了解 Aurora Serverless v2。

- 適用於 Aurora PostgreSQL 的受信任語言延伸模組

  了解如何建置可在 Aurora PostgreSQL 上安全執行的高效能延伸模組。

AWS Amazon Aurora MySQL 的研討會和實驗室內容入口網站

以下精選的研討會和其他實作內容可協助您了解 Amazon Aurora MySQL 的特性和功能：

- 建立 Aurora 叢集

  了解如何手動建立 Amazon Aurora MySQL 叢集。

- 建立 Cloud9 雲端型 IDE 環境以連線到您的資料庫

  了解如何設定 Cloud9 和初始化 MySQL 資料庫。
- 快速複製

  了解如何建立 Aurora 快速複製。
- 恢復叢集

  了解如何恢復資料庫叢集。
- 使用績效詳情

  了解如何使用績效詳情來監控和調整資料庫執行個體。
- 使用 RDS 工具進行效能監控

  了解如何使用 AWS 和 SQL 工具來了解效能問題，並找出改善資料庫效能的方法。
- 分析查詢效能

  了解如何使用不同的工具，針對與 SQL 效能相關的問題進行疑難排解。
- 自動擴展僅供讀取複本

  了解自動擴展僅供讀取複本的運作方式。
- 測試容錯

  了解 Aurora MySQL 中的高可用性和容錯功能。
- Aurora 全球資料庫

  了解 Aurora 全球資料庫。
- Aurora Serverless v2

  了解 Aurora Serverless v2。
- 使用機器學習

  了解 Aurora Machine Learning。

搭配 AWS SDK 使用此服務

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

SDK 文件	代碼範例
適用於 C++ 的 AWS SDK	適用於 C++ 的 AWS SDK 程式碼範例
AWS CLI	AWS CLI 程式碼範例
適用於 Go 的 AWS SDK	適用於 Go 的 AWS SDK 程式碼範例
適用於 Java 的 AWS SDK	適用於 Java 的 AWS SDK 程式碼範例
適用於 JavaScript 的 AWS SDK	適用於 JavaScript 的 AWS SDK 程式碼範例
適用於 Kotlin 的 AWS SDK	適用於 Kotlin 的 AWS SDK 程式碼範例
適用於 .NET 的 AWS SDK	適用於 .NET 的 AWS SDK 程式碼範例
適用於 PHP 的 AWS SDK	適用於 PHP 的 AWS SDK 程式碼範例
AWS Tools for PowerShell	AWS Tools for PowerShell 程式碼範例
適用於 Python (Boto3) 的 AWS SDK	適用於 Python (Boto3) 的 AWS SDK 程式碼範例
適用於 Ruby 的 AWS SDK	適用於 Ruby 的 AWS SDK 程式碼範例
適用於 Rust 的 AWS SDK	適用於 Rust 的 AWS SDK 程式碼範例
適用於 SAP ABAP 的 AWS SDK	適用於 SAP ABAP 的 AWS SDK 程式碼範例
適用於 Swift 的 AWS SDK	適用於 Swift 的 AWS SDK 程式碼範例

如需此服務的特定範例，請參閱 Aurora AWS SDKs的程式碼範例。

可用性範例

找不到所需的內容嗎？ 請使用本頁面底部的提供意見回饋連結申請程式碼範例。

以程式設計方式存取 Amazon Aurora

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

AWS Command Line Interface (AWS CLI)

您可以在命令列 shell 中使用 AWS CLI 來建立和管理 Amazon Aurora 資源。AWS CLI 可讓您針對 AWS 服務直接存取 API，例如 Amazon RDS。如需 Amazon RDS 命令的語法和範例，請參閱 AWS CLI 命令參考中的 rds。

AWS CloudFormation

使用此 AWS 基礎結構即程式碼 (IaC) 工具，您可以建立範本來描述您想要的所有 Amazon Aurora 資源，且 AWS CloudFormation 會為您佈建和設定那些資源。如需更多詳細資訊，請參閱 the section called “使用 建立資源 AWS CloudFormation”。

AWS 軟體開發套件 (SDK)

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

Amazon RDS API

此 API 是 Amazon RDS 的通訊協定層級介面。使用此 API 時，您必須正確格式化每個 HTTPS 請求，並為每個請求新增有效的數位簽章。如需更多詳細資訊，請參閱 Amazon RDS API 參考。

Console-to-Code

使用此工具，您可以為在 Amazon RDS 主控台中執行的動作產生程式碼，並在其他工具中使用該程式碼，例如 AWS CloudFormation。如需更多詳細資訊，請參閱 the section called “Console-to-Code”。

RDS 資料 API

RDS 資料 API (資料 API) 可讓您以程式設計方式存取 Amazon Aurora Serverless 資料庫中的資料。使用資料 API，您可以使用安全 HTTP 端點來執行 SQL 陳述式，而無需管理連線。如需更多詳細資訊，請參閱 使用 Amazon RDS 資料 API。

使用 Console-to-Code 為您的 Amazon Aurora 主控台動作產生程式碼

該主控台提供建立資源和測試原型的指導路徑。如果您想要大規模建立相同的資源，則需要自動化程式碼。Console-to-Code 是 Amazon Q Developer 的一項功能，可協助您開始使用自動化程式碼。Console-to-Code 會記錄您的主控台動作，包括預設值和您提供的參數值。然後，它會使用生成式 AI，針對您選擇的動作，以您偏好的語言和格式建議程式碼。由於主控台工作流程會確保您指定的參數值同時有效，因此您使用 Console-to-Code 產生的程式碼具有相容的參數值。您可以使用該程式碼作為起點，然後對其進行自訂以使其針對特定使用案例準備好投入生產。

例如，使用 Console-to-Code，您可以記錄建立 Aurora 資料庫叢集，並選擇以 AWS CloudFormation JSON 格式產生程式碼。然後，您可以複製該程式碼並進行自訂，以便在 AWS CloudFormation 範本中使用。

Console-to-Code 目前可以使用下列語言和格式產生基礎設施即程式碼 (IaC)：

- CDK Java
- CDK Python
- CDK TypeScript
- CloudFormation JSON
- CloudFormation YAML

如需了解如何使用 Console-to-Code 的詳細資訊和指示，請參閱《Amazon Q Developer 使用者指南》中的搭配 Amazon Q Developer Console-to-Code 將 AWS 服務自動化。

設定 Amazon Aurora 資料庫叢集

本節說明如何設定 Aurora 資料庫叢集。在建立 Aurora 資料庫叢集之前，決定將執行資料庫叢集的資料庫執行個體。此外，選擇 AWS 區域來決定資料庫叢集的執行位置。接下來，建立資料庫叢集。如果您具有 Aurora 以外的資料，則可將該資料遷移至 Aurora 資料庫叢集。

主題

- 建立 Amazon Aurora 資料庫叢集
- 使用 建立 Amazon Aurora 資源 AWS CloudFormation
- 連接至 Amazon Aurora 資料庫叢集
- Amazon Aurora 的參數群組
- 將資料遷移至 Amazon Aurora 資料庫叢集
- 使用 Aurora 資料庫叢集設定來建立 Amazon ElastiCache 快取
- 使用 自動將 EC2 資料庫遷移至 Amazon Aurora AWS Database Migration Service
- 教學課程：使用自訂參數群組建立 MySQL 資料庫叢集

建立 Amazon Aurora 資料庫叢集

Amazon Aurora 資料庫叢集包括可與 MySQL 或 PostgreSQL 相容的資料庫執行個體，以及放置資料庫叢集之資料的叢集磁碟區，而此叢集磁碟區是跨三個可用區域當做單一虛擬磁碟區所複製的。依預設，Aurora 資料庫叢集包含執行讀取和寫入的主要資料庫執行個體，而且最多可選擇性地包含 15 個 Aurora 複本 (讀取器資料庫執行個體)。如需 Aurora 資料庫叢集的詳細資訊，請參閱 Amazon Aurora 資料庫叢集。

Aurora 有兩種主要類型的資料庫叢集：

- 已佈建的 Aurora – 您可以根據預期的工作負載選擇寫入器和讀取器執行個體的資料庫執行個體類別。如需詳細資訊，請參閱Amazon Aurora 資料庫執行個體類別。已佈建的 Aurora 有數個選項，包括 Aurora 全域資料庫。如需詳細資訊，請參閱使用 Amazon Aurora 全球資料庫。
- Aurora Serverless – Aurora Serverless v2 是 Aurora 的隨需自動擴展組態。容量會根據應用程式需求自動調整。您只需支付資料庫叢集取用的資源費用。此自動化對於具有高度變化和不可預測工作負載的環境特別有用。如需詳細資訊，請參閱使用 Aurora Serverless v2。

接下來，您可以了解如何建立 Aurora 資料庫叢集。若要開始使用，請先參閱 資料庫叢集先決條件。

如需連線至 Aurora 資料庫叢集的指示，請參閱 連接至 Amazon Aurora 資料庫叢集。

內容

- 資料庫叢集先決條件
  - 設定資料庫叢集的網路
    - 設定與 EC2 執行個體的自動網路連線
    - 手動設定網路
  - 其他先決條件
- 建立資料庫叢集
  - 建立主要 (寫入器) 資料庫執行個體
- Aurora 資料庫叢集的設定
- Amazon Aurora 資料庫叢集不適用的設定
- Amazon Aurora 資料庫執行個體不適用的設定

資料庫叢集先決條件

> **Important**
>
> 您必須先完成 設定您的 Amazon Aurora 環境 中的任務，然後才能建立 Aurora 資料庫叢集。

以下是建立資料庫叢集前必須完成的先決條件。

主題

- 設定資料庫叢集的網路
- 其他先決條件

設定資料庫叢集的網路

您只能在具有至少兩個可用區域的 中，根據 Amazon VPC 服務在虛擬私有雲端 (VPC) 中建立 AWS 區域 Amazon Aurora 資料庫叢集。您為資料庫叢集選擇的資料庫子網路群組必須至少包含兩個可用區域。此組態可確保您的資料庫叢集一律至少有一個資料庫執行個體，以便萬一可用區域失敗時用於容錯移轉。

如果您計劃在相同 VPC 中設定新資料庫叢集與 EC2 執行個體之間的連線，則可在建立資料庫叢集期間這麼做。如果您計劃從相同 VPC 中 EC2 執行個體以外的資源連接到資料庫叢集，則可手動設定網路連線。

主題

- 設定與 EC2 執行個體的自動網路連線
- 手動設定網路

設定與 EC2 執行個體的自動網路連線

當您建立 Aurora 資料庫叢集時，您可以使用 AWS 管理主控台 來設定 Amazon EC2 執行個體與新資料庫叢集之間的連線。當您這樣做時，RDS 會自動設定您的 VPC 和網路設定。資料庫叢集會在與 EC2 執行個體相同的 VPC 中建立，讓 EC2 執行個體可以存取資料庫叢集。

以下是將 EC2 執行個體與資料庫叢集連接的要求：

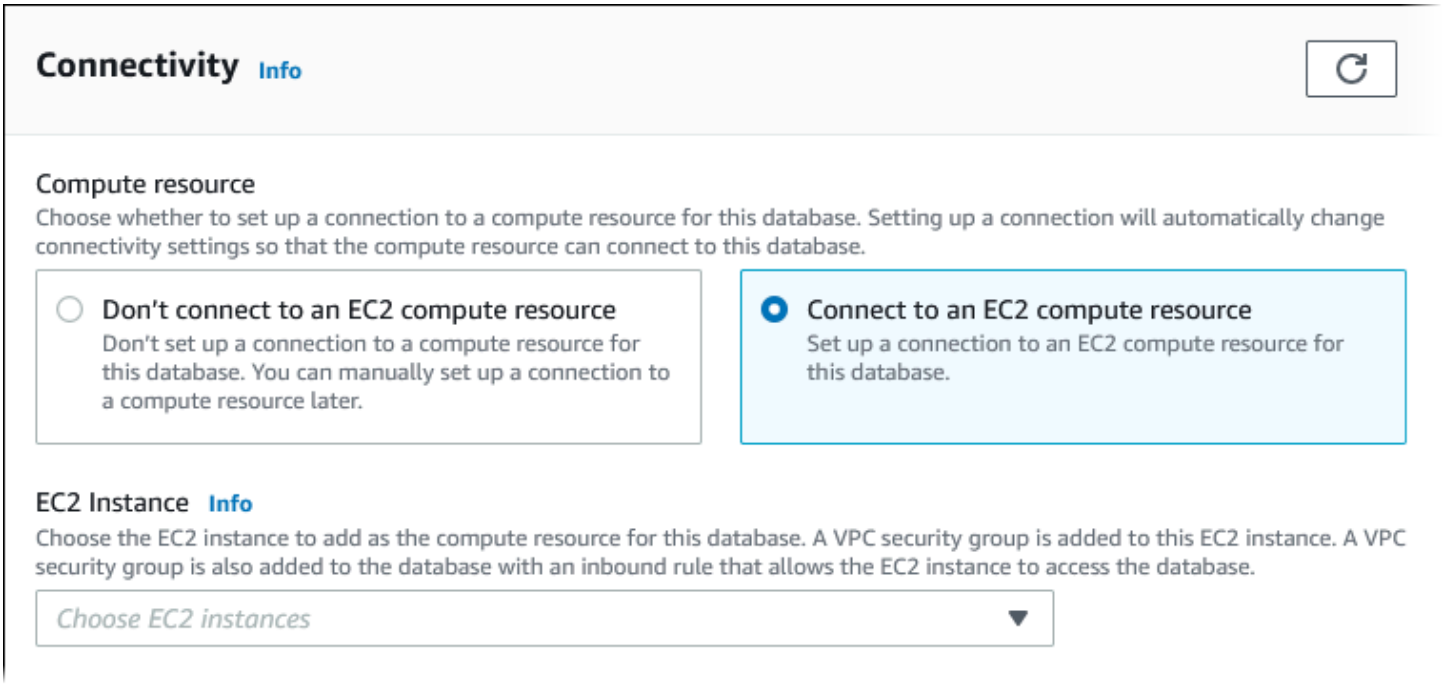
- 建立資料庫叢集 AWS 區域 之前，EC2 執行個體必須存在於 中。

如果 中不存在 EC2 執行個體 AWS 區域，主控台會提供建立 EC2 執行個體的連結。

- 目前，資料庫叢集不能是 Aurora Serverless 資料庫叢集或 Aurora 全球資料庫的一部分。
- 建立資料庫執行個體的使用者必須擁有執行下列操作的許可：
  - `ec2:AssociateRouteTable`
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateRouteTable`
  - `ec2:CreateSubnet`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeRouteTables`
  - `ec2:DescribeSecurityGroups`
  - `ec2:DescribeSubnets`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

使用此選項建立私有資料庫叢集。資料庫叢集使用僅包含私有子網路的資料庫子網路群組，以限制對 VPC 內資源的存取。

若要將 EC2 執行個體連接至資料庫叢集，請在 Create database (建立資料庫) 頁面上的 Connectivity (連線) 區段，選擇 Connect to an EC2 compute resource (連線至 EC2 運算資源)。



當您選擇 Connect to an EC2 compute resource (連線至 EC2 運算資源) 時，RDS 會自動設定下列選項。除非您透過選擇 Don't connect to an EC2 compute resource (不要連線至 EC2 運算資源)，選擇不設定與 EC2 執行個體的連線，否則無法變更設定。

主控台選項	自動設定
Network type (網路類型)	RDS 將網路類型設定為 IPv4。目前，當您設定 EC2 執行個體和資料庫叢集之間的連線時，不支援雙堆疊模式。
Virtual Private Cloud (VPC)	RDS 會將 VPC 設定為與 EC2 執行個體關聯的 VPC。
DB subnet group (資料庫子網路群組)	RDS 需要在相同可用區域中具有私有子網路的資料庫子網路群組作為 EC2 執行個體。如果存在符合此要求的資料庫子網路群組，則 RDS 會使用現有的資料庫子網路群組。依預設，此選項會設為 Automatic setup (自動設定)。 當您選擇 Automatic setup (自動設定)，且沒有符合此需求的資料庫子網路群組時，則會發生下列動作。RDS 在三個可用區域中使用三個可用的私有子網路，其中一個可用區域與 EC2 執行個體相同。如果可用區域中無法使用私有子網路，RDS 會在可用區域中建立私有子網路。RDS 接著建立資料庫子網路群組。

主控台選項	自動設定
	當私有子網路可用時，RDS 會使用與該子網路相關聯的路由表，並將其建立的任何子網路新增至此路由表。當沒有可用的私有子網路時，RDS 會建立沒有網際網路閘道存取權的路由表，並將其建立的子網路新增至路由表。 RDS 也可讓您使用現有的資料庫子網路群組。如果想要使用您選擇的現有資料庫子網路群組，請選取 Choose existing (選擇現有的)。
公用存取	RDS 選擇 No (否)，以便無法公開存取資料庫叢集。 為了安全起見，最佳實務是保持資料庫為私有，並確保無法從網際網路存取該資料庫。
VPC security group (firewall) (VPC 安全群組 (防火牆))	RDS 會建立與資料庫叢集關聯的新安全群組。安全群組已命名為 `rds-ec2-`*n*，其中 *n* 是數字。此安全群組包含以 EC2 VPC 安全群組 (防火牆) 做為來源的傳入規則。與資料庫叢集關聯的此一安全群組可讓 EC2 執行個體存取資料庫叢集。 RDS 也會建立與 EC2 執行個體關聯的新安全群組。安全群組命名為 `ec2-rds-`*n*，其中 *n* 是數字。此安全群組包含傳出規則，並將資料庫叢集的 VPC 安全群組做為來源。此安全群組允許 EC2 執行個體將流量傳送到資料庫叢集。 您可以選擇 Create New (建立新的)，並輸入新安全群組的名稱，以新增其他的新安全群組。 您可以選擇 Choose existing (選擇現有)，然後選取要新增的安全群組，以新增現有的安全群組。

主控台選項	自動設定
可用區域	當您在建立資料庫叢集 (單一可用區域部署) 期間，沒有在 Availability & durability (可用性和耐用性) 中建立 Aurora 複本時，RDS 會選擇 EC2 執行個體的可用區域。 當您在建立資料庫叢集 (多可用區部署) 期間建立 Aurora 複本時，RDS 會為資料庫叢集中的一個資料庫執行個體選擇 EC2 執行個體的可用區域。RDS 會隨機地為資料庫叢集中的其他資料庫執行個體選擇不同的可用區域。主要資料庫執行個體或 Aurora 複本是在與 EC2 執行個體相同的可用區域中建立的。如果主要資料庫執行個體和 EC2 執行個體位於不同的可用區域，可能會產生跨可用區域成本。

如需這些設定的詳細資訊，請參閱 Aurora 資料庫叢集的設定。

如果您在建立資料庫叢集之後對這些設定進行任何變更，這些變更可能會影響 EC2 執行個體和資料庫叢集之間的連線。

手動設定網路

如果您計劃從相同 VPC 中 EC2 執行個體以外的資源連接到資料庫叢集，則可手動設定網路連線。如果您使用 AWS 管理主控台 來建立資料庫叢集，則可讓 Amazon RDS 自動為您建立 VPC。或者，您可以使用現有的 VPC 或為您的 Aurora 資料庫叢集建立新的 VPC。無論您使用哪個方式，在至少兩個可用區域的每一個中，您的 VPC 必須有至少一個子網路，您才能與 Amazon Aurora 資料庫叢集搭配使用。

Amazon RDS 預設會自動為您建立主要資料庫執行個體，並在可用區域中建立 Aurora 複本。若要選擇特定的可用區域，您需要將 Availability & durability (可用性與持久性) 多可用區部署設定更改為 Don't create an Aurora Replica (不建立 Aurora 複本)。這樣做會暴露一個可讓您從 VPC 的可用區域中選擇的 Availability Zone (可用區域) 設定。然而，我們強烈建議您保留預設設定，讓 Amazon RDS 代您建立多可用區部署並選擇可用區域。這樣一來，您的 Aurora 資料庫叢集會在建立時具有快速容錯移轉和高可用性功能，這些功能是 Aurora 的兩個主要優勢。

如果您沒有預設 VPC 或尚未建立 VPC，可以在使用主控台建立資料庫叢集時讓 Amazon RDS 自動為您建立 VPC。否則您必須執行以下操作：

- 在 AWS 區域 您要部署資料庫叢集的 中，在至少兩個可用區域中各建立一個子網路的 VPC。如需詳細資訊，請參閱在 VPC 中使用資料庫叢集及教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4)。
- 指定可授權連線到資料庫叢集的 VPC 安全群組。如需詳細資訊，請參閱建立安全群組以存取 VPC 中的資料庫叢集及使用安全群組控制存取。
- 指定 RDS DB 子網路群組，該子網路群組會在可由 資料庫叢集使用的 VPC 中，至少定義兩個子網路。如需詳細資訊，請參閱使用資料庫子網路群組。

如需 VPC 的相關資訊，請參閱Amazon VPC 和 Amazon Aurora。如需為私有資料庫叢集設定網路的教學課程，請參閱 教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4)。

如果您想要連線至與 Aurora 資料庫叢集不在相同 VPC 中的資源，請參閱 在 VPC 中存取資料庫叢集的案例 中的適當案例。

其他先決條件

建立資料庫叢集之前，請考慮下列額外的先決條件：

- 如果您要 AWS 使用 AWS Identity and Access Management (IAM) 登入資料連線至 ， AWS 您的帳戶必須具有授予執行 Amazon RDS 操作所需許可的 IAM 政策。如需詳細資訊，請參閱Amazon Aurora 的 Identity and access management。

  如果您使用 IAM 存取 Amazon RDS 主控台，您必須先 AWS 管理主控台 使用使用者憑證登入 。接著前往 Amazon RDS 主控台：https://console.aws.amazon.com/rds/。
- 如果您想要為資料庫叢集量身打造組態參數，則必須使用所需的參數設定來指定資料庫叢集參數群組及資料庫參數群組。如需建立或修改資料庫叢集參數群組或資料庫參數群組的相關資訊，請參閱Amazon Aurora 的參數群組。
- 決定要為資料庫叢集指定的 TCP/IP 連接埠號碼。某些公司的防火牆會封鎖與預設 Aurora 連接埠 (若為 MySQL，則為 3306；若為 PostgreSQL，則為 5432) 的連線。如果您的公司防火牆會封鎖預設連接埠，請為您的資料庫叢集選擇另一個連接埠。資料庫叢集中的所有執行個體都使用相同的連接埠。
- 如果資料庫的主要引擎版本已達到 RDS 標準支援結束日期，您必須使用延長支援 CLI 選項或 RDS API 參數。如需詳細資訊，請參閱「Aurora 資料庫叢集的設定 中的 RDS 延長支援」。

建立資料庫叢集

您可以使用 AWS 管理主控台、 AWS CLI或 RDS API 建立 Aurora 資料庫叢集。

主控台

您可以使用已啟用或不已啟用輕鬆建立 AWS 管理主控台 的 來建立資料庫叢集。 在 Easy create (輕鬆建立) 啟用的情形下，您僅指定資料庫引擎類型、資料庫執行個體大小與資料庫執行個體識別符。Easy create (輕鬆建立) 會將預設設定用於其他設定選項。在 Easy create (輕鬆建立) 未啟用的情形下，您會在建立資料庫時指定更多設定選項，包含可用性、安全性、備份和維護的選項。

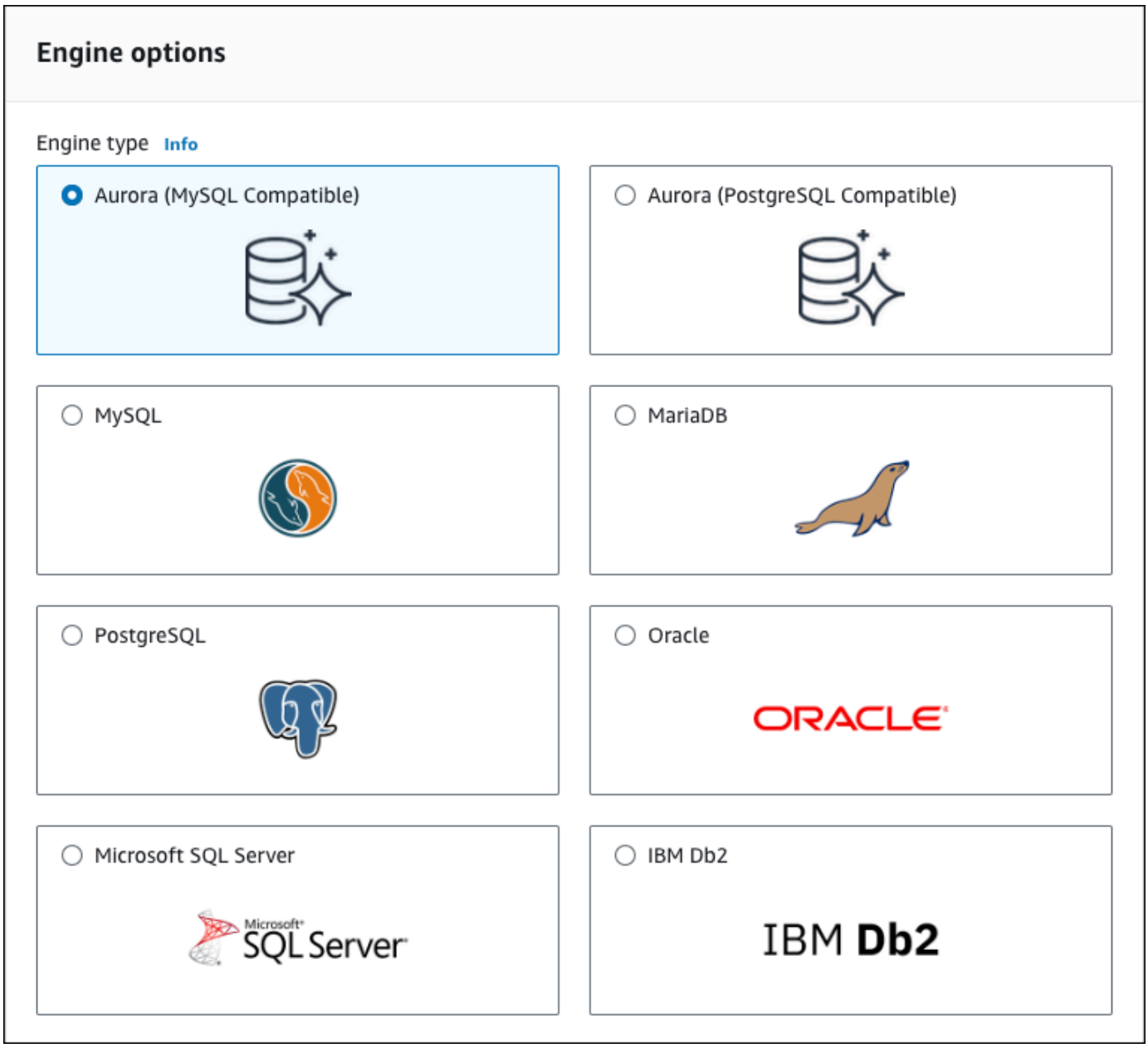
> **Note**
>
> 若為此範例，Standard Create (標準建立) 已遭啟用，Easy Create (輕鬆建立) 尚未啟用。如需有關建立已啟用輕鬆建立的資料庫叢集的資訊，請參閱 Amazon Aurora 入門。

使用主控台建立 Aurora 資料庫叢集

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在 的右上角 AWS 管理主控台，選擇您要在其中建立資料庫叢集 AWS 的區域。

   Aurora 並非所有 AWS 區域都提供。如需可使用 Aurora AWS 的區域清單，請參閱 區域可用性。
3. 在導覽窗格中，選擇 Databases (資料庫)。
4. 選擇 Create database (建立資料庫)。
5. 針對選擇資料庫建立方法，請選擇標準建立。
6. 針對引擎類型，選擇以下其中一項：

   - Aurora (相容於 MySQL)
   - Aurora (相容於 PostgreSQL)



7. 選擇引擎版本。

   如需詳細資訊，請參閱Amazon Aurora 版本。您可以使用篩選條件來選擇與您想要的功能相容的版本，例如 Aurora Serverless v2。如需詳細資訊，請參閱使用 Aurora Serverless v2。

8. 在 Templates (範本) 中，選擇符合您使用案例的範本。

9. 若要輸入您的主要密碼，請執行以下動作：

   a. 在設定區段，展開憑證設定。

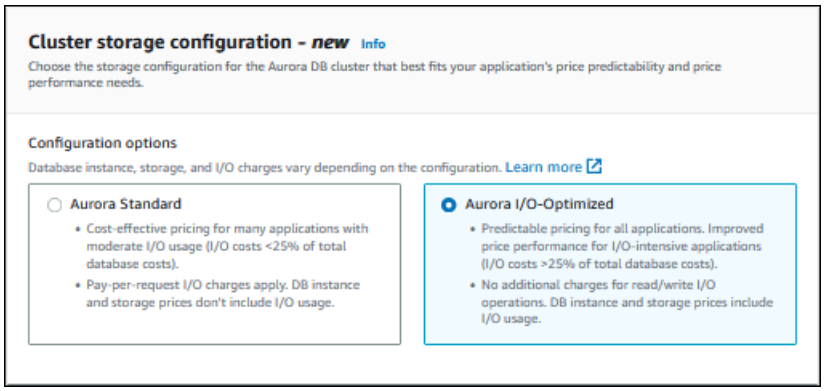
b. 清除 Auto generate a password (自動產生密碼) 核取方塊。

c. (選用) 變更 Master username (主要使用者名稱) 值並在 Master password (主要密碼) 和 Confirm password (確認密碼) 中輸入相同的密碼。

依預設，新資料庫執行個體會將自動產生的密碼用在主要使用者。

10. 在 VPC 安全群組 (防火牆) 下的連線能力區段中，如果您選取建立新的，則系統會建立 VPC 安全群組並提供傳入規則，允許本機電腦的 IP 地址存取資料庫。

11. 對於叢集儲存組態，選擇 Aurora I/O-Optimized 或 Aurora Standard。如需詳細資訊，請參閱Amazon Aurora 資料庫叢集的儲存組態。



12. (選用) 為此資料庫叢集設定與運算資源的連線。

您可以在建立資料庫叢集期間設定 Amazon EC2 執行個體和新資料庫叢集之間的連線。如需詳細資訊，請參閱設定與 EC2 執行個體的自動網路連線。

13. 在其餘區段，指定資料庫叢集設定。如需每項設定的相關資訊，請參閱 Aurora 資料庫叢集的設定。

14. 選擇 Create database (建立資料庫)。

如果您選擇使用自動產生的密碼，View credential details (檢視登入資料詳細資訊) 按鈕會出現在 Databases (資料庫) 頁面。

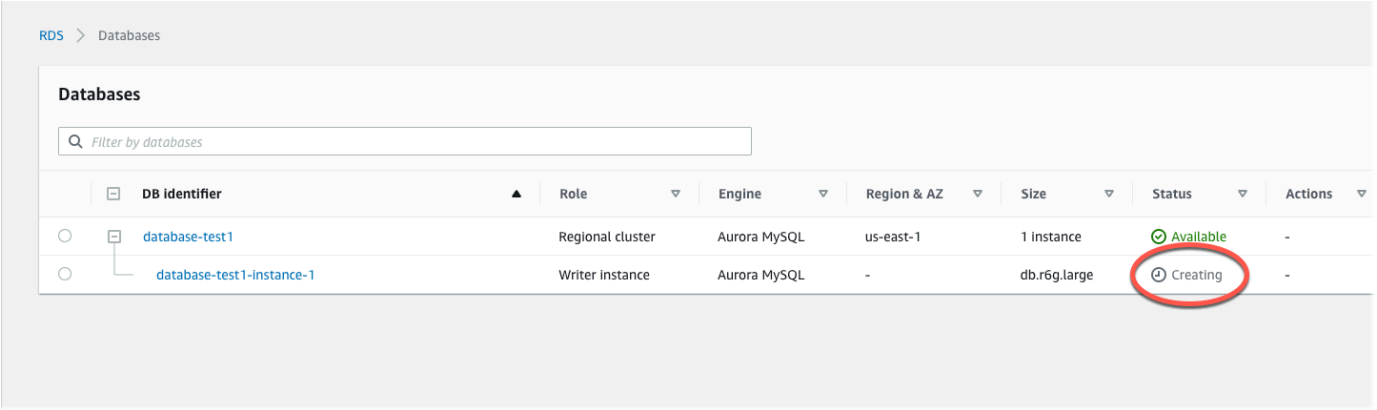
若要檢視資料庫叢集的主要使用者名稱和密碼，請選擇 View credential details (檢視登入資料詳細資訊)。

若要以主要使用者的身分連接至資料庫執行個體，請使用出現的使用者名稱和密碼。

> **Important**
>
> 您無法再次檢視主要使用者密碼。如果您沒有記錄下來，您可能需要進行變更。如果您需要在資料庫執行個體可供使用後變更主要使用者密碼，您可以將資料庫執行個體修改為這麼做。如需修改 資料庫執行個體的詳細資訊，請參閱修改 Amazon Aurora 資料庫叢集。

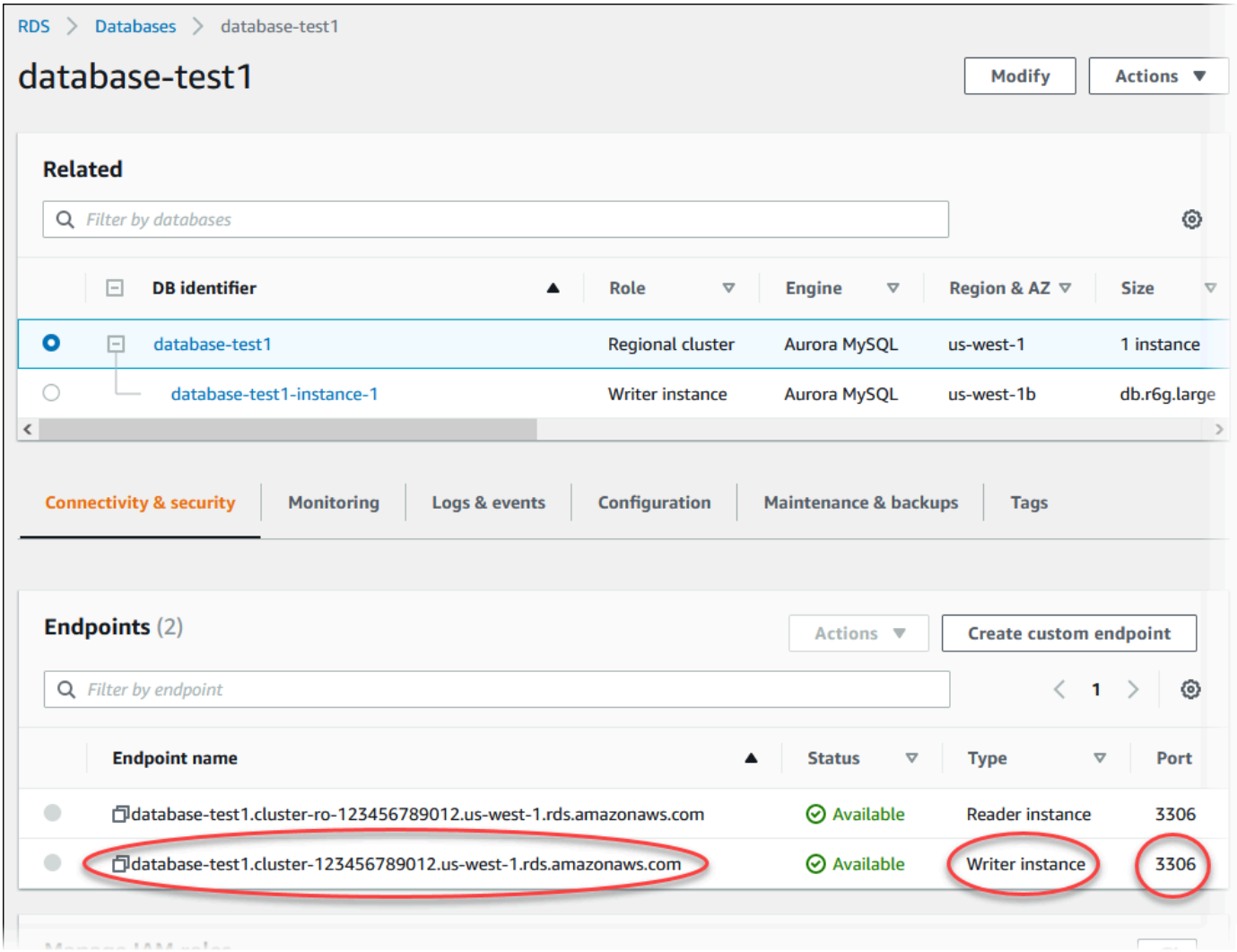
15. 在 Databases (資料庫) 中，選擇新 Aurora 資料庫叢集的名稱。

    在 RDS 主控台上，新資料庫叢集的詳細資訊即會出現。在資料庫叢集可供使用之前，資料庫叢集和其資料庫執行個體會處於 Creating (建立中) 的狀態。



    兩者狀態都變更為 Available (可用) 時，您便能連線至資料庫叢集。視資料庫執行個體類別和儲存體數量而定，可能需要最多 20 分鐘的時間，新的資料庫叢集才會可用。

    若要檢視剛建立的叢集，請在 Amazon RDS 主控台中的導覽窗格選擇 Databases (資料庫)。然後，選擇資料庫叢集來顯示資料庫叢集詳細資訊。如需更多詳細資訊，請參閱 檢視 Amazon Aurora 資料庫叢集。



在 Connectivity & security (連線能力和安全性) 索引標籤上，記下寫入器資料庫執行個體的連接埠和端點。對於任何執行寫入或讀取操作的應用程式，在您的 JDBC 和 ODBC 連線字串中使用叢集的端點和連接埠。

AWS CLI

> **Note**
>
> 您必須先 AWS CLI滿足必要的先決條件，例如建立 VPC 和 RDS 資料庫子網路群組，才能使用 建立 Aurora 資料庫叢集。如需詳細資訊，請參閱資料庫叢集先決條件。

您可以使用 AWS CLI 來建立 Aurora MySQL 資料庫叢集或 Aurora PostgreSQL 資料庫叢集。

使用 建立 Aurora MySQL 資料庫叢集 AWS CLI

當您建立 Aurora MySQL 8.0 相容或 MySQL 5.7 相容的資料庫叢集或資料庫執行個體時，請將 `--engine` 選項指定為 `aurora-mysql`。

完成下列步驟：

1. 識別新資料庫叢集的資料庫子網路群組和 VPC 安全群組 ID，然後呼叫 create-db-cluster AWS CLI 命令，來建立 Aurora MySQL 資料庫叢集。

   例如，下列命令建立新的 MySQL 8.0 相容資料庫叢集，名為 `sample-cluster`。該叢集會使用預設引擎版本和 Aurora I/O-Optimized 儲存類型。

   對於 Linux、macOS 或 Unix：

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster \
       --engine aurora-mysql --engine-version 8.0 \
       --storage-type aurora-iopt1 \
       --master-username user-name --manage-master-user-password \
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   在 Windows 中：

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
       --engine aurora-mysql --engine-version 8.0 ^
       --storage-type aurora-iopt1 ^
       --master-username user-name --manage-master-user-password ^
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   下列命令建立新的 MySQL 5.7 相容資料庫叢集，名為 `sample-cluster`。該叢集會使用預設引擎版本和 Aurora Standard 儲存類型。

   對於 Linux、macOS 或 Unix：

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster  \
       --engine aurora-mysql --engine-version 5.7 \
       --storage-type aurora \
       --master-username user-name --manage-master-user-password \
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   在 Windows 中：

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster sample-cluster  ^
    --engine aurora-mysql --engine-version 5.7 ^
    --storage-type aurora ^
    --master-username user-name --manage-master-user-password ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

2. 如果您使用主控台來建立資料庫叢集，則 Amazon RDS 會自動建立資料庫叢集的主要執行個體 (寫入器)。如果您使用 AWS CLI 來建立資料庫叢集，則必須明確建立資料庫叢集的主要執行個體。主要執行個體是資料庫叢集內第一個建立的執行個體。直到您建立主要資料庫執行個體為止，資料庫叢集端點都會保持在 `Creating` 狀態。

   呼叫 create-db-instance AWS CLI 命令，為您的資料庫叢集建立主要執行個體。包含資料庫叢集的名稱做為 `--db-cluster-identifier` 選項值。

   > **Note**
   >
   > 您無法針對資料庫執行個體設定 `--storage-type` 選項。您只能針對資料庫叢集設定它。

   例如，下列命令建立新的 MySQL 5.7 相容或 MySQL 8.0 相容資料庫執行個體，名為 `sample-instance`。

   對於 Linux、macOS 或 Unix：

   ```
   aws rds create-db-instance --db-instance-identifier sample-instance \
        --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-
   class db.r5.large
   ```

   在 Windows 中：

   ```
   aws rds create-db-instance --db-instance-identifier sample-instance ^
        --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-
   class db.r5.large
   ```

使用 建立 Aurora PostgreSQL 資料庫叢集 AWS CLI

1. 識別新資料庫叢集的資料庫子網路群組和 VPC 安全群組 ID，然後呼叫 create-db-cluster AWS CLI 命令，來建立 Aurora PostgreSQL 資料庫叢集。

   例如，下列命令會建立新的資料庫叢集，名為 `sample-cluster`。該叢集會使用預設引擎版本和 Aurora I/O-Optimized 儲存類型。

   對於 Linux、macOS 或 Unix：

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster \
       --engine aurora-postgresql \
       --storage-type aurora-iopt1 \
       --master-username user-name --manage-master-user-password \
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   在 Windows 中：

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
       --engine aurora-postgresql ^
       --storage-type aurora-iopt1 ^
       --master-username user-name --manage-master-user-password ^
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

2. 如果您使用主控台來建立資料庫叢集，則 Amazon RDS 會自動建立資料庫叢集的主要執行個體 (寫入器)。如果您使用 AWS CLI 來建立資料庫叢集，則必須明確建立資料庫叢集的主要執行個體。主要執行個體是資料庫叢集內第一個建立的執行個體。直到您建立主要資料庫執行個體為止，資料庫叢集端點都會保持在 `Creating` 狀態。

   呼叫 create-db-instance AWS CLI 命令，為您的資料庫叢集建立主要執行個體。包含資料庫叢集的名稱做為 `--db-cluster-identifier` 選項值。

   對於 Linux、macOS 或 Unix：

   ```
   aws rds create-db-instance --db-instance-identifier sample-instance \
        --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-
   instance-class db.r5.large
   ```

   在 Windows 中：

```
aws rds create-db-instance --db-instance-identifier sample-instance ^
     --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-
instance-class db.r5.large
```

這些範例會指定 `--manage-master-user-password` 選項來產生主要使用者密碼，並在 Secrets Manager 中管理該密碼。如需詳細資訊，請參閱使用 Amazon Aurora 和 進行密碼管理 AWS Secrets Manager。或者，您可以使用 `--master-password` 選項，自行指定和管理密碼。

RDS API

> **Note**
>
> 您必須先 AWS CLI滿足必要的先決條件，例如建立 VPC 和 RDS 資料庫子網路群組，才能使用 建立 Aurora 資料庫叢集。如需詳細資訊，請參閱資料庫叢集先決條件。

識別新資料庫叢集的資料庫子網路群組和 VPC 安全群組 ID，然後呼叫 CreateDBInstance 作業來建立資料庫叢集。

當您建立 Aurora MySQL 第 2 版或第 3 版資料庫叢集或資料庫執行個體時，請為 `Engine` 參數指定 `aurora-mysql`。

當您建立 Aurora PostgreSQL 資料庫叢集或資料庫執行個體時，請為 `aurora-postgresql` 參數指定 `Engine`。

如果您使用主控台來建立資料庫叢集，則 Amazon RDS 會自動建立資料庫叢集的主要執行個體 (寫入器)。如果您使用 RDS API 來建立資料庫叢集，則必須使用 CreateDBInstance 明確地建立資料庫叢集的主要執行個體。主要執行個體是資料庫叢集內第一個建立的執行個體。直到您建立主要資料庫執行個體為止，資料庫叢集端點都會保持在 `Creating` 狀態。

建立主要 (寫入器) 資料庫執行個體

如果您使用 AWS 管理主控台 建立資料庫叢集，Amazon RDS 會自動為您的資料庫叢集建立主要執行個體 （寫入器）。如果您使用 AWS CLI 或 RDS API 來建立資料庫叢集，則必須明確建立資料庫叢集的主要執行個體。主要執行個體是資料庫叢集內第一個建立的執行個體。直到您建立主要資料庫執行個體為止，資料庫叢集端點都會保持在 `Creating` 狀態。

如需詳細資訊，請參閱建立資料庫叢集。

> **Note**
>
> 如果您有不含寫入器資料庫執行個體的資料庫叢集，也稱為無周邊叢集，則無法使用主控台建立寫入器執行個體。您必須使用 AWS CLI 或 RDS API。

下列範例使用 create-db-instance AWS CLI 命令，為名為 的 Aurora PostgreSQL 資料庫叢集建立寫入器執行個體`headless-test`。

```
aws rds create-db-instance \
    --db-instance-identifier no-longer-headless \
    --db-cluster-identifier headless-test \
    --engine aurora-postgresql \
    --db-instance-class db.t4g.medium
```

Aurora 資料庫叢集的設定

下表包含您在建立 Aurora 資料庫叢集時選擇之設定的詳細資訊。

主控台設定	設定說明	CLI 選項和 RDS API 參數
Auto minor version upgrade (自動次要版本升級)	如果您想要讓 Aurora 資料庫叢集可以自動接收資料庫引擎偏好的次要版本升級，請選擇 Enable auto minor version upgrade (啟用自動次要版本升級)。 Auto minor version upgrade (自動次要版本升級) 設定同時適用於 Aurora PostgreSQL 和 Aurora MySQL 資料庫叢集。 如需 Aurora PostgreSQL 引擎更新的詳細資訊，請參閱 Amazon Aurora PostgreSQL 的資料庫引擎更新。	請對 Aurora 叢集中的每個資料庫執行個體設定此值。如果叢集中的任何資料庫執行個體已關閉此設定，則叢集不會自動升級。 使用 AWS CLI執行`create-db-instance` 並設定 `--auto-minor-version-upgrade|--no-auto-minor-version-upgrade` 選項。 使用 RDS API，呼叫 `CreateDBInstance` ，並設定 `AutoMinorVersionUpgrade` 參數。

主控台設定	設定說明	CLI 選項和 RDS API 參數
	如需 Aurora MySQL 引擎更新的詳細資訊，請參閱 Amazon Aurora MySQL 的資料庫引擎更新。	
AWS KMS key	只有在 Encryption (加密) 設為 Enable encryption (啟用加密) 時才能使用。選擇要用於加密此資料庫叢集的 AWS KMS key 。如需詳細資訊，請參閱加密 Amazon Aurora 資源。	使用 AWS CLI執行create-db-cluster並設定 --kms-key-id選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 KmsKeyId 參數。
恢復	僅適用於 Aurora MySQL。選擇 Enable Backtrack (啟用恢復) 以啟用恢復，或選取 Disable Backtrack (停用恢復) 以停用恢復。恢復功能可讓您將資料庫叢集恢復到特定時間，而不需建立新的資料庫叢集。預設為停用。如果要啟用恢復功能，還需指定您希望能夠恢復資料庫叢集的時間長度 (目標恢復時段)。如需詳細資訊，請參閱恢復 Aurora 資料庫叢集。	使用 AWS CLI執行create-db-cluster並設定 --backtrack-window 選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 BacktrackWindow 參數。
憑證授權單位	資料庫叢集中資料庫執行個體所使用之伺服器憑證的憑證授權單位 (CA)。 如需詳細資訊，請參閱使用 SSL/TLS 加密與資料庫叢集的連線。	使用 AWS CLI執行create-db-instance 並設定 --ca-certificate-identifier 選項。 使用 RDS API，呼叫 CreateDBInstance ，並設定 CACertificateIdentifier 參數。

主控台設定	設定說明	CLI 選項和 RDS API 參數
叢集儲存組態	資料庫叢集的儲存類型：Aurora I/O-Optimized 或 Aurora Standard。 如需詳細資訊，請參閱Amazon Aurora 資料庫叢集的儲存組態。	使用 AWS CLI執行create-db-cluster並設定 --storage-type選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 StorageType 參數。
Copy tags to snapshots (將標籤複製到快照)	選擇此選項，可在您建立資料庫快照時將任何資料庫執行個體標籤複製到此快照。 如需詳細資訊，請參閱標記 Amazon Aurora 和 Amazon RDS 資源。	使用 AWS CLI執行create-db-cluster並設定 --copy-tags-to-snapshot | --no-copy-tags-to-snapshot 選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 CopyTagsToSnapshot 參數。

主控台設定	設定說明	CLI 選項和 RDS API 參數
Database authentication (資料庫身分驗證	您想要使用的資料庫身分驗證。 適用於 MySQL： • 選擇 Password authentication (密碼身分驗證)，僅使用資料庫密碼驗證資料庫使用者。 • 選擇 Password and IAM database authentication (密碼和 IAM 資料庫身分驗證)，透過 IAM 使用者和角色搭配資料庫密碼和使用者登入資料對資料庫使用者進行身分驗證。如需更多詳細資訊，請參閱 IAM 資料庫身分驗證。 適用於 PostgreSQL： • 選擇 IAM database authentication (IAM 資料庫身分驗證)，透過使用者和角色搭配資料庫密碼和使用者登入資料對資料庫使用者進行身分驗證。如需詳細資訊，請參閱 IAM 資料庫身分驗證。 • 選擇 Kerberos authentication (身分驗證)，以使用 Kerberos 身分驗證對資料庫密碼和使用者登入資料進行身分驗證。如需詳細資訊，請參閱搭配 Aurora PostgreSQL 使用 Kerberos 身分驗證。	若要搭配 使用 IAM 資料庫身分驗證 AWS CLI，請執行 create-db-cluster 並設定 --enable-iam-database-authentication | --no-enable-iam-database-authentication 選項。 如要將 IAM 資料庫身分驗證與 RDS API 搭配使用，請呼叫 CreateDBCluster，然後設定 EnableIAMDatabaseAuthentication 參數。 若要搭配 使用 Kerberos 身分驗證 AWS CLI，請執行create-db-cluster並設定 --domain 和 --domain-iam-role-name 選項。 如要將 Kerberos 身分驗證與 RDS API 搭配使用，請呼叫 CreateDBCluster，然後設定 Domain 和 DomainIAMRoleName 參數。

主控台設定	設定說明	CLI 選項和 RDS API 參數
Database port (資料庫連線埠)	指定應用程式和公用程式將用於存取資料庫的連接埠。Aurora MySQL 資料庫叢集預設為預設的 MySQL 連接埠 3306，而 Aurora PostgreSQL 資料庫叢集則預設為預設的 PostgreSQL 連接埠 5432。某些公司的防火牆會封鎖與這些預設連接埠的連線。如果您的公司防火牆會封鎖預設連接埠，請為新的資料庫叢集選擇另一個連接埠。	使用 AWS CLI執行`create-db-cluster`並設定 `--port`選項。 使用 RDS API，呼叫 `CreateDBCluster`，並設定 `Port` 參數。
DB cluster identifier (資料庫叢集識別符)	輸入資料庫叢集的名稱，該名稱對您選擇的 AWS 區域中的帳戶是唯一的。此識別符用於資料庫叢集中的叢集端點位址。如需叢集端點的詳細資訊，請參閱Amazon Aurora 端點連線。 資料庫叢集識別符有下列限制： • 必須包含 1 到 63 個英數字元或連字號。 • 第一個字元必須是字母。 • 不能以一個連字號結尾或是連續包含兩個連字號。 • 它對於每個 AWS 區域每個帳戶的所有資料庫叢集必須是唯一的 AWS 。	使用 AWS CLI執行`create-db-cluster`並設定 `--db-cluster-identifier` 選項。 使用 RDS API，呼叫 `CreateDBCluster`，並設定 `DBClusterIdentifier` 參數。

主控台設定	設定說明	CLI 選項和 RDS API 參數
DB cluster parameter group (資料庫叢集參數群組)	選擇資料庫叢集參數群組 Aurora 有一個預設資料庫叢集參數群組供您使用，您也可以建立自己的資料庫叢集參數群組。如需資料庫叢集參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。	使用 AWS CLI執行create-db-cluster 並設定 --db-cluster-parameter-group-name選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 DBClusterParameterGroupName 參數。
DB instance class (資料庫執行個體類別)	僅適用於已佈建的容量類型。選擇資料庫執行個體類別，為資料庫叢集中的每個執行個體定義處理和記憶體要求。如需資料庫執行個體類別的詳細資訊，請參閱Amazon Aurora 資料庫執行個體類別。	請對 Aurora 叢集中的每個資料庫執行個體設定此值。 使用 AWS CLI執行create-db-instance 並設定 --db-instance-class 選項。 使用 RDS API，呼叫 CreateDBInstance ，並設定 DBInstanceClass 參數。
DB parameter group (資料庫參數群組)	選擇一個 parameter groups (參數群組)。Aurora 有一個預設參數群組供您使用，您也可以建立自己的參數群組。如需參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。	請對 Aurora 叢集中的每個資料庫執行個體設定此值。 使用 AWS CLI執行create-db-instance 並設定 --db-parameter-group-name 選項。 使用 RDS API，呼叫 CreateDBInstance ，並設定 DBParameterGroupName 參數。

主控台設定	設定說明	CLI 選項和 RDS API 參數
DB subnet group (資料庫子網路群組)	您要用於資料庫叢集的資料庫子網路群組。 選取 Choose existing (選擇現有的) 來使用現有的資料庫子網路群組。然後，從 Existing DB subnet groups (現有資料庫子網路群組) 下拉式清單中選擇所需的子網路群組。 選擇 Automatic setup (自動設定) 以讓 RDS 選取相容的資料庫子網路群組。如果不存在，RDS 會為您的叢集建立新的子網路群組。 如需詳細資訊，請參閱資料庫叢集先決條件。	使用 AWS CLI執行create-db-cluster並設定 `--db-subnet-group-name` 選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 `DBSubnetGroupName` 參數。
啟用刪除保護	選擇 Enable deletion protection (啟用刪除保護) 以避免您的資料庫叢集遭意外刪除。如果您使用主控台建立生產資料庫叢集，預設會啟用刪除保護功能。	使用 AWS CLI執行create-db-cluster並設定 `--deletion-protection | --no-deletion-protection` 選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 `DeletionProtection` 參數。
Enable encryption (啟用加密)	選擇 `Enable encryption` ，啟用此資料庫叢集的靜態加密。如需詳細資訊，請參閱加密 Amazon Aurora 資源。	使用 AWS CLI執行create-db-cluster 並設定 `--storage-encrypted | --no-storage-encrypted` 選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 `StorageEncrypted` 參數。

主控台設定	設定說明	CLI 選項和 RDS API 參數
Enable Enhanced Monitoring (啟用增強型監控)	選擇 Enable enhanced monitoring (啟用增強型監控)，以針對資料庫叢集執行所在的作業系統即時收集指標。如需詳細資訊，請參閱使用增強型監控來監控作業系統指標。	請對 Aurora 叢集中的每個資料庫執行個體設定這些值。 使用 AWS CLI，執行create-db-instance 並設定 --monitoring-interval 和 --monitoring-role-arn 選項。 使用 RDS API，呼叫 CreateDBInstance ，並設定 MonitoringInterval 和 MonitoringRoleArn 參數。
啟用 RDS 資料 API	選擇啟用 RDS Data API 以啟用 RDS Data API (資料 API)。資料 API 提供安全的 HTTP 端點，無需管理連線即可執行 SQL 陳述式。如需詳細資訊，請參閱使用 Amazon RDS 資料 API。	使用 AWS CLI執行create-db-cluster 並設定 --enable-http-endpoint | --no-enable-http-endpoint 選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 EnableHttpEndpoint 參數。
Engine type (引擎類型)	選擇要用於此資料庫執行個體叢集的資料庫引擎。	使用 AWS CLI執行create-db-cluster並設定 --engine選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 Engine 參數。
引擎版本	僅適用於已佈建的容量類型。選擇資料庫引擎版本編號。	使用 AWS CLI執行create-db-cluster 並設定 --engine-version 選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 EngineVersion 參數。

主控台設定	設定說明	CLI 選項和 RDS API 參數
容錯移轉優先順序	選擇執行個體的容錯移轉優先順序。如果您未選擇值，則預設值為 tier-1 (第一層)。此優先順序決定從主要執行個體失敗中復原時提升 Aurora 複本的順序。如需詳細資訊，請參閱Aurora 資料庫叢集的容錯能力。	請對 Aurora 叢集中的每個資料庫執行個體設定此值。 使用 AWS CLI執行`create-db-instance` 並設定 `--promotion-tier` 選項。 使用 RDS API，呼叫 `CreateDBInstance` ，並設定 `PromotionTier` 參數。

主控台設定	設定說明	CLI 選項和 RDS API 參數
初始資料庫名稱	輸入您預設資料庫的名稱。如果您未提供 Aurora MySQL 資料庫叢集名稱，Amazon RDS 則不會在您所建立的資料庫叢集上建立資料庫。如果您沒有為 Aurora PostgreSQL 資料庫叢集提供名稱，則 Amazon RDS 會建立名為 `postgres` 的資料庫。 針對 Aurora MySQL，預設資料庫名稱具有下列條件限制： • 必須包含 1–64 個英數字元。 • 不能是資料庫引擎的保留字。 針對 Aurora PostgreSQL，預設資料庫名稱具有下列條件限制： • 名稱必須包含 1–63 個英數字元。 • 必須以字母開頭。後續字元可以是字母、底線或數字 (0–9)。 • 不能是資料庫引擎的保留字。 若要建立其他資料庫，請連接至資料庫叢集，並使用 SQL 命令 CREATE DATABASE。如需連接至資料庫叢集的詳細資訊，請參閱連接至 Amazon Aurora 資料庫叢集。	使用 AWS CLI執行`create-db-cluster`並設定 `--database-name`選項。 使用 RDS API，呼叫 `CreateDBCluster` ，並設定 `DatabaseName` 參數。

主控台設定	設定說明	CLI 選項和 RDS API 參數
Log exports (日誌匯出)	在 Log exports (日誌匯出) 區段中，選擇您要開始發佈至 Amazon CloudWatch Logs 的日誌。如需將 Aurora MySQL 日誌發佈至 CloudWatch Logs 的詳細資訊，請參閱 將 Amazon Aurora MySQL 日誌發佈至 Amazon CloudWatch Logs。如需將 Aurora PostgreSQL 日誌發佈至 CloudWatch Logs 的詳細資訊，請參閱 將 Aurora PostgreSQL 日誌發佈至 Amazon CloudWatch Logs。	使用 AWS CLI執行create-db-cluster 並設定 --enable-cloudwatch-logs-exports 選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 EnableCloudwatchLogsExports 參數。
Maintenance window (維護時段)	選擇 Select window (選取時段)，並指定可能發生系統維護的每週時間範圍。或者，選擇 No preference (無偏好設定)，讓 Amazon RDS 隨機指派期間。	使用 AWS CLI執行create-db-cluster並設定 --preferred-maintenance-window 選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 PreferredMaintenanceWindow 參數。
在 中管理主登入資料 AWS Secrets Manager	選取 Manage master credentials in AWS Secrets Manager (管理 AWS Secrets Manager中的主要憑證) 以秘密管理 Secrets Manager 中的主要使用者密碼。 選擇性地選擇要用來保護機密的 KMS 金鑰。從您帳戶中的 KMS 金鑰進行選擇，或輸入來自不同帳戶的金鑰。 如需詳細資訊，請參閱使用 Amazon Aurora 和 進行密碼管理 AWS Secrets Manager。	使用 AWS CLI執行create-db-cluster 並設定 --manage-master-user-password | --no-manage-master-user-password 和 --master-user-secret-kms-key-id選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 ManageMasterUserPassword 和 MasterUserSecretKmsKeyId 參數。

主控台設定	設定說明	CLI 選項和 RDS API 參數
Master password (主要密碼)	輸入密碼以登入您的資料庫叢集： • 針對 Aurora MySQL，密碼必須包含 8–41 個可印刷的 ASCII 字元。 • 針對 Aurora PostgreSQL，密碼必須包含 8 – 99 個可印刷的 ASCII 字元。 • 其中不能包含 /、"、@ 或空格。	使用 AWS CLI執行create-db-cluster 並設定 --master-user-password 選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 MasterUserPassword 參數。
主要使用者名稱	輸入名稱以做為主要使用者名稱使用，來登入您的資料庫叢集： • 針對 Aurora MySQL，名稱必須包含 1–16 個英數字元。 • 針對 Aurora PostgreSQL，其中必須包含 1–63 個英數字元。 • 第一個字元必須是字母。 • 名稱不能是資料庫引擎的保留字。 建立資料庫叢集後，無法變更主要使用者名稱。	使用 AWS CLI執行create-db-cluster 並設定 --master-username 選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 MasterUsername 參數。

主控台設定	設定說明	CLI 選項和 RDS API 參數
Multi-AZ deployment (異地同步備份部署)	僅適用於已佈建的容量類型。判斷您是否想要在其他可用區域中建立 Aurora 複本，以取得容錯移轉支援。如果您選擇 Create Replica in Different Zone (在不同區域中建立複本)，則 Amazon RDS 會在您的資料庫叢集為您建立 Aurora 複本，而此資料庫叢集所在的可用區域與資料庫叢集之主要執行個體所在的可用區域不同。如需多個可用區域的詳細資訊，請參閱區域和可用區域。	使用 AWS CLI執行create-db-cluster 並設定 --availability-zones 選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 AvailabilityZones 參數。
Network type (網路類型)	資料庫叢集支援的 IP 定址通訊協定。 IPv4 指定資源只能透過網際網路通訊協定第 4 版 (IPv4) 定址通訊協定與資料庫叢集通訊。 Dual-stack mode (雙堆疊模式) 指定資源可透過 IPv4、IPv6 或兩者與資料庫叢集通訊。如果您有任何資源必須透過 IPv6 定址通訊協定與您的資料庫叢集通訊，請使用雙堆疊模式。若要使用雙堆疊模式，請確定至少有兩個跨越兩個可用區域的子網路，同時支援 IPv4 和 IPv6 網路通訊協定。此外，請確保將 IPv6 CIDR 區塊與您指定的資料庫子網路群組中的所有子網路相關聯。 如需詳細資訊，請參閱Amazon Aurora IP 定址。	使用 AWS CLI執行create-db-cluster 並設定 -network-type 選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 NetworkType 參數。

主控台設定	設定說明	CLI 選項和 RDS API 參數
公用存取	選擇 Publicly accessible (可公開存取)，為資料庫叢集提供公有 IP 地址，或選擇 Not publicly accessible (不可公開存取)。資料庫叢集內的執行個體可以混合公有和私有資料庫執行個體。如需隱藏執行個體而不提供公開存取的詳細資訊，請參閱在 VPC 中的網際網路中隱藏資料庫叢集。 若要從其 Amazon VPC 之外連接至資料庫執行個體，資料庫執行個體必須可公開存取、必須使用資料庫執行個體之安全群組的傳入規則授予存取權，且必須符合其他要求。如需詳細資訊，請參閱無法連線至 Amazon RDS 資料庫執行個體。 如果您的資料庫執行個體無法公開存取，您也可以使用 AWS Site-to-Site連線或 Direct Connect 連線從私有網路存取。如需詳細資訊，請參閱網際網路流量隱私權。	請對 Aurora 叢集中的每個資料庫執行個體設定此值。 使用 AWS CLI執行`create-db-instance` 並設定 `--publicly-accessible | --no-publicly-accessible` 選項。 使用 RDS API，呼叫 `CreateDBInstance`，並設定 `PubliclyAccessible` 參數。

主控台設定	設定說明	CLI 選項和 RDS API 參數
RDS 延長支援	選取啟用 RDS 延長支援，以允許支援的主要引擎版本在超過 Aurora 標準支援結束日期後繼續執行。 當您建立資料庫叢集時，Amazon Aurora 預設為 RDS 延長支援。若要防止在 Aurora 標準支援日期結束後建立新的資料庫叢集，並避免 RDS 延長支援的費用，請停用此設定。在 RDS 延長支援定價開始日期之前，您現有的資料庫叢集不會產生費用。 如需詳細資訊，請參閱使用 Amazon Aurora 的 Amazon RDS 延伸支援。	使用 AWS CLI執行create-db-cluster 並設定 --engine-lifecycle-support 選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 EngineLifecycleSupport 參數。
RDS Proxy	選擇 Create an RDS Proxy (建立 RDS 代理)，為您的資料庫叢集建立一個代理。Amazon RDS 會自動為代理建立 IAM 角色和 Secrets Manager 機密。 如需詳細資訊，請參閱Amazon RDS Proxy for Aurora。	無法在建立資料庫叢集時使用。
保留期間	選擇從 1 到 35 天的時間長度，Aurora 會在此時間內保留資料庫備份副本。備份副本可用來進行資料庫的 point-in-time 還原 (PITR) 作業，單位可計算至秒。	使用 AWS CLI執行create-db-cluster 並設定 --backup-retention-period 選項。 使用 RDS API，呼叫 CreateDBCluster ，並設定 BackupRetentionPeriod 參數。

主控台設定	設定說明	CLI 選項和 RDS API 參數
Turn on DevOps Guru (開啟 DevOps Guru)	選擇 Turn on DevOps Guru (開啟 DevOps Guru) 以開啟 Aurora 資料庫的 Amazon DevOps Guru。若要讓 DevOps Guru for RDS 提供對效能異常的詳細分析，必須開啟 Performance Insights。如需詳細資訊，請參閱設定 DevOps Guru for RDS。	您可以從 RDS 主控台開啟 DevOps Guru for RDS，但無法使用 RDS API 或 CLI。如需開啟 DevOps Guru 的相關詳細資訊，請參閱《Amazon DevOps Guru 使用者指南》。
開啟績效詳情	選擇 Turn on Performance Insights (開啟績效詳情)，以開啟 Amazon RDS 績效詳情。如需詳細資訊，請參閱在 Amazon Aurora 上使用績效詳情監控資料庫負載。	請對 Aurora 叢集中的每個資料庫執行個體設定這些值。 使用 AWS CLI來執行 `create-db-instance` 並設定 `--enable-performance-insights | --no-enable-performance-insights` 、`--performance-insights-kms-key-id` 和 `--performance-insights-retention-period` 選項。 使用 RDS API，呼叫 `CreateDBInstance`，並設定 `EnablePerformanceInsights` 、`PerformanceInsightsKMSKeyId` 和 `PerformanceInsightsRetentionPeriod` 參數。
Virtual Private Cloud (VPC)	選擇託管資料庫叢集的 VPC。選擇 Create a New VPC (建立新的 VPC)，由 Amazon RDS 為您建立 VPC。如需詳細資訊，請參閱資料庫叢集先決條件。	對於 AWS CLI 和 API，您可以指定 VPC 安全群組 IDs。

主控台設定	設定說明	CLI 選項和 RDS API 參數
VPC security group (firewall) (VPC 安全群組 (防火牆))	選擇 Create new (建立新項目)，讓 Amazon RDS 為您建立 VPC 安全群組。或者，選擇 Choose existing (選擇現有項目)，並指定一個或多個 VPC 安全群組，來保護資料庫叢集的網路存取。 在 RDS 主控台中選擇 Create new (建立新項目) 時，即會建立新的安全群組，其中具有的傳入規則會允許透過您的瀏覽器中偵測到的 IP 地址存取資料庫執行個體。 如需詳細資訊，請參閱資料庫叢集先決條件。	使用 AWS CLI執行`create-db-cluster` 並設定 `--vpc-security-group-ids` 選項。 使用 RDS API，呼叫 `CreateDBCluster` ，並設定 `VpcSecurityGroupIds` 參數。

Amazon Aurora 資料庫叢集不適用的設定

AWS CLI 命令`create-db-cluster`和 RDS API 操作中的下列設定`CreateDBCluster`不適用於 Amazon Aurora 資料庫叢集。

> **Note**
>
> AWS 管理主控台 不會顯示 Aurora 資料庫叢集的這些設定。

AWS CLI 設定	RDS API 設定
`--allocated-storage`	`AllocatedStorage`
`--auto-minor-version-upgrade | --no-auto-minor-version-upgrade`	`AutoMinorVersionUpgrade`
`--db-cluster-instance-class`	`DBClusterInstanceClass`

AWS CLI 設定	RDS API 設定
`--enable-performance-insights | --no-enable-performance-insights`	`EnablePerformanceInsights`
`--iops`	`Iops`
`--monitoring-interval`	`MonitoringInterval`
`--monitoring-role-arn`	`MonitoringRoleArn`
`--option-group-name`	`OptionGroupName`
`--performance-insights-kms-key-id`	`PerformanceInsightsKMSKeyId`
`--performance-insights-retention-period`	`PerformanceInsightsRetentionPeriod`
`--publicly-accessible | --no-publicly-accessible`	`PubliclyAccessible`

Amazon Aurora 資料庫執行個體不適用的設定

AWS CLI 命令`create-db-instance`和 RDS API 操作中的下列設定`CreateDBInstance`不適用於資料庫執行個體 Amazon Aurora 資料庫叢集。

> **Note**
>
> AWS 管理主控台 不會顯示 Aurora 資料庫執行個體的這些設定。

AWS CLI 設定	RDS API 設定
`--allocated-storage`	`AllocatedStorage`
`--availability-zone`	`AvailabilityZone`

AWS CLI 設定	RDS API 設定
--backup-retention-period	BackupRetentionPeriod
--backup-target	BackupTarget
--character-set-name	CharacterSetName
--character-set-name	CharacterSetName
--custom-iam-instance-profile	CustomIamInstanceProfile
--db-security-groups	DBSecurityGroups
--deletion-protection | --no-deletion-protection	DeletionProtection
--domain	Domain
--domain-iam-role-name	DomainIAMRoleName
--enable-cloudwatch-logs-exports	EnableCloudwatchLogsExports
--enable-customer-owned-ip | --no-enable-customer-owned-ip	EnableCustomerOwnedIp
--enable-iam-database-authentication | --no-enable-iam-database-authentication	EnableIAMDatabaseAuthentication
--engine-version	EngineVersion
--iops	Iops
--kms-key-id	KmsKeyId
--master-username	MasterUsername
--master-user-password	MasterUserPassword
--max-allocated-storage	MaxAllocatedStorage

AWS CLI 設定	RDS API 設定
--multi-az | --no-multi-az	MultiAZ
--nchar-character-set-name	NcharCharacterSetName
--network-type	NetworkType
--option-group-name	OptionGroupName
--preferred-backup-window	PreferredBackupWindow
--processor-features	ProcessorFeatures
--storage-encrypted | --no-storage-encrypted	StorageEncrypted
--storage-type	StorageType
--tde-credential-arn	TdeCredentialArn
--tde-credential-password	TdeCredentialPassword
--timezone	Timezone
--vpc-security-group-ids	VpcSecurityGroupIds

使用 建立 Amazon Aurora 資源 AWS CloudFormation

Amazon Aurora 已與 整合 AWS CloudFormation，這項服務可協助您模型化和設定 AWS 資源，以減少建立和管理資源和基礎設施的時間。您可以建立範本，描述您想要的所有 AWS 資源 、 （例如資料庫叢集和資料庫叢集參數群組），以及為您 CloudFormation 佈建和設定這些資源。

使用 時 CloudFormation，您可以重複使用範本來一致且重複地設定 Aurora資源。描述您的資源一次，然後在多個 AWS 帳戶和區域中逐一佈建相同的資源。

Aurora和 CloudFormation 範本

CloudFormation 範本是 JSON 或 YAML 格式的文字檔案。這些範本說明您要在 CloudFormation 堆疊中佈建的資源。如果您不熟悉 JSON 或 YAML，您可以使用 CloudFormation 設計工具來協助您開始使用 CloudFormation 範本。如需詳細資訊，請參閱《AWS CloudFormation 使用者指南》 中的什麼是 CloudFormation 設計器？。

Aurora支援在 中建立資源 CloudFormation。如需更多詳細資訊 (包括這些資源的 JSON 和 YAML 範本範例)，請參閱 AWS CloudFormation 使用者指南中的 RDS 資源類型參考。

進一步了解 CloudFormation

若要進一步了解 CloudFormation，請參閱下列資源：

- AWS CloudFormation
- AWS CloudFormation 使用者指南
- CloudFormation API 參考
- AWS CloudFormation 命令列界面使用者指南

連接至 Amazon Aurora 資料庫叢集

您可以使用與用來連接至 MySQL 或 PostgreSQL 資料庫的同一工具，來連接至 Aurora 資料庫叢集。您使用連接至 MySQL 或 PostgreSQL 資料庫執行個體的任何指令碼、公用程式或應用程式，來指定連接字串。您會使用相同的公有金鑰進行 Secure Sockets Layer (SSL) 連線。

在連接字串中，您通常會透過與資料庫叢集關聯的特殊端點來使用主機和連接埠資料。不論叢集中有多少資料庫執行個體，您都可以透過這些端點來使用相同的連接參數。對於疑難排解之類的專業任務，您也可以透過 Aurora 資料庫叢集中的特定資料庫執行個體來使用主機和連接埠資訊。

> **Note**
>
> 對於 Aurora Serverless 資料庫叢集，您可以連線到資料庫端點，而非資料庫執行個體。您可以在 Aurora Serverless 的 Connectivity & security (連線與安全) 索引標籤上找到 AWS 管理主控台資料庫叢集的資料庫端點。如需詳細資訊，請參閱使用 Amazon Aurora Serverless v1。

無論您用於與資料庫叢集或執行個體搭配使用的 Aurora 資料庫引擎和特定工具為何，端點都必須可存取。Aurora 資料庫叢集只能在以 Amazon VPC 服務為基礎的 Virtual Private Cloud (VPC) 中建立。這表示您可以使用下列其中一種方法，從 VPC 內部或 VPC 外部存取端點。

- 存取 VPC 內的 Aurora 資料庫叢集 – 啟用透過 VPC 存取 Aurora 資料庫叢集。您可以在 VPC 的安全群組上編輯 Inbound (入站) 規則，以允許存取特定 Aurora 資料庫叢集。如需進一步了解，包括如何針對不同的 Aurora 資料庫叢集案例設定 VPC，請參閱 Amazon Virtual Private Cloud (VPC) 和 Amazon Aurora。
- 存取 VPC 外部的 Aurora 資料庫叢集 – 若要從 VPC 外部存取 Aurora 資料庫叢集，請使用資料庫叢集的公有端點地址。

如需詳細資訊，請參閱對 Aurora 連線失敗進行故障診斷。

內容

- 使用 AWS 驅動程式連線至 Aurora 資料庫叢集
- 連接至 Amazon Aurora MySQL 資料庫叢集
  - Aurora MySQL 的連線公用程式
  - 使用 MySQL 公用程式連線至 Aurora MySQL
  - 使用 Amazon Web Services (AWS) JDBC 驅動程式連線至 Aurora MySQL

- 使用 Amazon Web Services (AWS) Python 驅動程式連線至 Aurora MySQL
- 使用 Amazon Web Services (AWS) 適用於 MySQL 的 ODBC 驅動程式連線至 Aurora MySQL
- 使用 Amazon Web Services (AWS) 進階 NodeJS 包裝函式連線至 Aurora MySQL
- 使用 SSL 連線至 Aurora MySQL
- 連接至 Amazon Aurora PostgreSQL 資料庫叢集
  - Aurora PostgreSQL 的連線公用程式
  - 使用 Amazon Web Services (AWS) JDBC 驅動程式連線至 Aurora PostgreSQL
  - 使用 Amazon Web Services (AWS) Python 驅動程式連線至 Aurora PostgreSQL
  - 使用 Amazon Web Services (AWS) 進階 NodeJS 包裝函式連線至 Aurora PostgreSQL
- 對 Aurora 連線失敗進行故障診斷

使用 AWS 驅動程式連線至 Aurora 資料庫叢集

驅動程式 AWS 套件旨在支援更快的切換和容錯移轉時間，以及使用 AWS Secrets Manager AWS Identity and Access Management (IAM) 和聯合身分進行身分驗證。 AWS 驅動程式依賴監控資料庫叢集狀態，並注意叢集拓撲以判斷新的寫入器。相較於開放原始碼驅動程式的數十秒，此方法可將切換和容錯移轉時間縮短為短短幾秒。

下表列出每個驅動程式支援的功能。隨著推出新的服務功能，驅動程式 AWS 套件的目標是為這些服務功能提供內建支援。

功能	AWS JDBC 驅動程式	AWS Python 驅動程式	AWS 適用於 MySQL 的 ODBC 驅動程式	AWS 進階 NodeJS 包裝函式
容錯移轉支援	是	是	是	是
增強型容錯移轉監控	是	是	是	是
讀取/寫入分割	是	是	否	是
Aurora 連線追蹤器	是	是	否	是

功能	AWS JDBC 驅動程式	AWS Python 驅動程式	AWS 適用於 MySQL 的 ODBC 驅動程式	AWS 進階 NodeJS 包裝函式
驅動程式中繼資料連線	是	N/A	N/A	N/A
遙測	是	是	否	是
Secrets Manager	是	是	是	是
IAM 身分驗證	是	是	是	是
聯合身分 (ADFS)	是	是	否	是
聯合身分 (Okta)	是	是	是	是
Aurora PostgreSQL Limitless Database	是 (僅限 Aurora PostgreSQL)	否	否	是 (僅限 Aurora PostgreSQL)

如需 AWS 驅動程式的詳細資訊，請參閱 Aurora MySQL 或 Aurora PostgreSQL 資料庫叢集的對應語言驅動程式。

連接至 Amazon Aurora MySQL 資料庫叢集

若要對您的 Aurora MySQL 資料庫叢集進行身分驗證，您可以使用 MySQL 使用者名稱和密碼身分驗證或 AWS Identity and Access Management (IAM) 資料庫身分驗證。如需使用 MySQL 使用者名稱和密碼身分驗證的詳細資訊，請參閱 MySQL 文件中的存取控制和帳戶管理。如需使用 IAM 資料庫身分驗證的詳細資訊，請參閱 IAM 資料庫身分驗證。

當您連線至與 MySQL 8.0 相容的 Amazon Aurora 資料庫叢集時，您可以執行任何與 MySQL 8.0 版相容的 SQL 命令。最低相容版本是 MySQL 8.0.23。如需 MySQL 8.0 SQL 語法的詳細資訊，請參閱 MySQL 8.0 參考手冊。如需有關 Aurora MySQL 第 3 版的限制資訊，請參閱 比較 Aurora MySQL 第 3 版與 MySQL 8.0 社群版。

當您連線至與 MySQL 5.7 相容的 Amazon Aurora 資料庫叢集時，您可以執行任何與 MySQL 5.7 版相容的 SQL 命令。如需 MySQL 5.7 SQL 語法的詳細資訊，請參閱 MySQL 5.7 參考手冊。如需有關 Aurora MySQL 5.7 的限制資訊，請參閱 Aurora MySQL 第 2 版與 MySQL 5.7 相容。

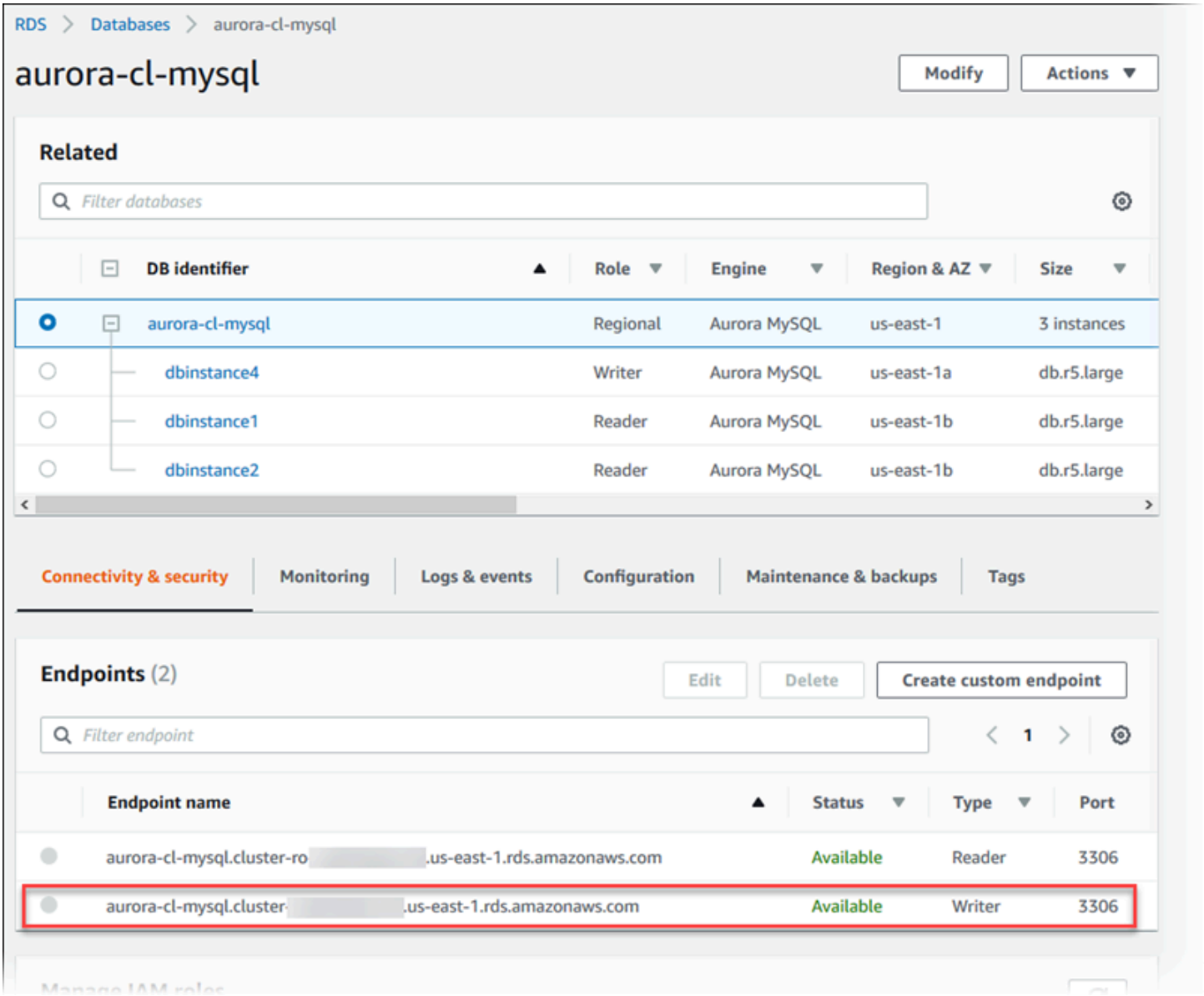
> **Note**
>
> 如需實用且詳細的指南，以了解如何連線至 Amazon Aurora MySQL 資料庫叢集，請參閱 Aurora 連線管理手冊。

在資料庫叢集的詳細資訊檢視中，您可以尋找能用於 MySQL 連線字串的叢集端點。端點是由資料庫叢集的網域名稱和連接埠組成。例如，如果端點值為 `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com:3306`，則您可以在 MySQL 連線字串中指定下列值：

- 對於主機或主機名稱，指定 `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`
- 對於連接埠，指定 `3306` 或您在建立資料庫叢集時所使用的連接埠值

叢集端點會將您連接至資料庫叢集的主要執行個體。您可以使用叢集端點來同時執行讀取和寫入操作。您的資料庫叢集也可以具有最多 15 個 Aurora 複本，這些複本支援資料庫叢集中資料的唯讀存取。主要執行個體和每一個 Aurora 複本都有一個獨立於叢集端點的唯一端點，並可讓您直接連接至叢集中的特定資料庫執行個體。叢集端點一律指向主要執行個體。如果主要執行個體失敗且被取代，則叢集端點會指向新的主要執行個體。

若要檢視叢集端點 (寫入者端點)，請在 Amazon RDS 主控台上選擇 Databases (資料庫)，並選擇資料庫叢集的名稱以顯示資料庫叢集詳細資訊。



主題

- Aurora MySQL 的連線公用程式
- 使用 MySQL 公用程式連線至 Aurora MySQL
- 使用 Amazon Web Services (AWS) JDBC 驅動程式連線至 Aurora MySQL
- 使用 Amazon Web Services (AWS) Python 驅動程式連線至 Aurora MySQL
- 使用 Amazon Web Services (AWS) 適用於 MySQL 的 ODBC 驅動程式連線至 Aurora MySQL
- 使用 Amazon Web Services (AWS) 進階 NodeJS 包裝函式連線至 Aurora MySQL
- 使用 SSL 連線至 Aurora MySQL

Aurora MySQL 的連線公用程式

一些您可以使用的連線公用程式如下：

- 命令列 – 您可以使用 MySQL 命令列公用程式之類的工具，來連線至 Amazon Aurora 資料庫叢集。如需使用 MySQL 公用程式的詳細資訊，請參閱 MySQL 文件中的 mysql - MySQL 命令列用戶端。
- GUI – 您可以使用 MySQL Workbench 公用程式，以使用 UI 介面進行連線。如需詳細資訊，請參閱下載 MySQL Workbench 頁面。
- AWS 驅動程式：
  - 使用 Amazon Web Services (AWS) JDBC 驅動程式連線至 Aurora MySQL
  - 使用 Amazon Web Services (AWS) Python 驅動程式連線至 Aurora MySQL
  - 使用 Amazon Web Services (AWS) 適用於 MySQL 的 ODBC 驅動程式連線至 Aurora MySQL
  - 使用 Amazon Web Services (AWS) 進階 NodeJS 包裝函式連線至 Aurora MySQL

使用 MySQL 公用程式連線至 Aurora MySQL

使用下列程序。它假設您在 VPC 的私有子網路中設定資料庫叢集。您可以根據教學：建立 Web 伺服器和 Amazon Aurora 資料庫叢集中的教學課程，使用 Amazon EC2 執行個體進行連線。

> **Note**
>
> 此程序不需要安裝教學課程中的 Web 伺服器，但確實需要安裝 MariaDB 10.5。

使用 MySQL 公用程式連接至資料庫叢集

1. 登入至您用於連線至資料庫叢集的 EC2 執行個體。

   您應該會看到類似下列的輸出。

```
Last login: Thu Jun 23 13:32:52 2022 from xxx.xxx.xxx.xxx

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-10-0-xxx.xxx ~]$
```

2. 在命令提示字元中輸入下列命令，以連線至資料庫叢集的主要資料庫執行個體。

   若為 -h 參數，請將端點 DNS 名稱替換成您的主要執行個體。若為-u 參數，請替換資料庫使用者帳戶的使用者 ID。

   ```
   mysql -h primary-instance-endpoint.AWS_account.AWS_Region.rds.amazonaws.com -P 3306
    -u database_user -p
   ```

   例如：

   ```
   mysql -h my-aurora-cluster-instance.c1xy5example.123456789012.eu-
   central-1.rds.amazonaws.com -P 3306 -u admin -p
   ```

3. 輸入資料庫使用者的密碼。

   您應該會看到類似下列的輸出。

   ```
   Welcome to the MariaDB monitor.  Commands end with ; or \g.
   Your MySQL connection id is 1770
   Server version: 8.0.23 Source distribution

   Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

   Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

   MySQL [(none)]>
   ```

4. 輸入 SQL 命令。

使用 Amazon Web Services (AWS) JDBC 驅動程式連線至 Aurora MySQL

Amazon Web Services (AWS) JDBC 驅動程式設計為進階 JDBC 包裝函式。此包裝函式與現有的 JDBC 驅動程式互補並擴展其功能，以協助應用程式利用叢集資料庫的功能，例如 Aurora MySQL。驅動程式與社群 MySQL 連接器/J 驅動程式和社群 MariaDB 連接器/J 驅動程式相容。

若要安裝 AWS JDBC 驅動程式，請附加 AWS JDBC 驅動程式 .jar 檔案（位於應用程式 CLASSPATH)，並保留對個別社群驅動程式的參考。更新個別的連線 URL 字首，如下所示：

- `jdbc:mysql://` 至 `jdbc:aws-wrapper:mysql://`
- `jdbc:mariadb://` 至 `jdbc:aws-wrapper:mariadb://`

如需 AWS JDBC 驅動程式的詳細資訊和完整的使用說明，請參閱 Amazon Web Services (AWS) JDBC 驅動程式 GitHub 儲存庫。

> **Note**
>
> MariaDB Connector/J 公用程式的 3.0.3 版捨棄對 Aurora 資料庫叢集的支援，因此我們強烈建議移至 AWS JDBC 驅動程式。

使用 Amazon Web Services (AWS) Python 驅動程式連線至 Aurora MySQL

Amazon Web Services (AWS) Python 驅動程式設計為進階 Python 包裝函式。此包裝函式是開放原始碼 Psycopg 驅動程式的補充和延伸功能。 AWS Python 驅動程式支援 Python 3.8 版和更新版本。您可以使用 `pip` 命令以及`psycopg`開放原始碼`aws-advanced-python-wrapper`套件來安裝套件。

如需 AWS Python 驅動程式的詳細資訊以及完整的使用說明，請參閱 Amazon Web Services (AWS) Python Driver GitHub 儲存庫。

使用 Amazon Web Services (AWS) 適用於 MySQL 的 ODBC 驅動程式連線至 Aurora MySQL

AWS ODBC Driver for MySQL 是用戶端驅動程式，專為 Aurora MySQL 的高可用性而設計。驅動程式可以與 MySQL Connector/ODBC 驅動程式一起存在，並與相同的工作流程相容。

如需有關 AWS ODBC Driver for MySQL 的詳細資訊，以及安裝和使用它的完整說明，請參閱 Amazon Web Services (AWS) ODBC Driver for MySQL GitHub 儲存庫。

使用 Amazon Web Services (AWS) 進階 NodeJS 包裝函式連線至 Aurora MySQL

AWS 進階 NodeJS 包裝函式是現有 NodeJS 驅動程式的補充和延伸功能。它有助於應用程式利用叢集資料庫的功能，例如 Aurora MySQL。

如需 AWS 進階 NodeJS 包裝函式的詳細資訊以及完整的使用說明，請參閱 Amazon Web Services (AWS) 進階 NodeJS 包裝函式 GitHub 儲存庫。

使用 SSL 連線至 Aurora MySQL

您可以在 Aurora MySQL 資料庫執行個體的連線上使用 SSL 加密。如需相關資訊，請參閱Aurora MySQL 資料庫叢集的 TLS 連線。

若要使用 SSL 進行連線，請依下列程序所述使用 MySQL 公用程式。如果您是使用 IAM 資料庫身分驗證，則必須使用 SSL 連線。如需相關資訊，請參閱「IAM 資料庫身分驗證」。

> **Note**
>
> 要使用 SSL 連接至叢集端點，您的用戶端連線公用程式必須支援主體別名 (SAN)。如果您的用戶端連線公用程式不支援 SAN，則您可以直接連接至 Aurora 資料庫叢集中的執行個體。如需 Aurora 端點的詳細資訊，請參閱 Amazon Aurora 端點連線。

使用 MySQL 公用程式搭配 SSL 連接至資料庫叢集

1. 下載公開金鑰以供 Amazon RDS 簽署憑證。

   如需有關下載憑證的詳細資訊，請參閱使用 SSL/TLS 加密與資料庫叢集的連線。

2. 在命令提示字元中輸入下列命令，以使用 MySQL 公用程式搭配 SSL 來連接至資料庫叢集的主要執行個體。若為 -h 參數，請將端點 DNS 名稱替換成您的主要執行個體。若為-u 參數，請替換資料庫使用者帳戶的使用者 ID。在 --ssl-ca 參數中，換成適當的 SSL 憑證檔名稱。出現提示時，請輸入主要使用者密碼。

   ```
   mysql -h mycluster-primary.123456789012.us-east-1.rds.amazonaws.com -u admin_user -p --ssl-ca=[full path]global-bundle.pem --ssl-verify-server-cert
   ```

您應該會看到類似下列的輸出。

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 350
Server version: 8.0.26-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

如需有關建構 RDS for MySQL 連線字串以及尋找 SSL 連線公有金鑰的一般說明，請參閱連線至執行 MySQL 資料庫引擎的資料庫執行個體。

連接至 Amazon Aurora PostgreSQL 資料庫叢集

您可以使用與用來連接至 PostgreSQL 資料庫的同一工具，來連接至 Amazon Aurora PostgreSQL 資料庫叢集中的資料庫執行個體。在此過程中，您會使用相同的公有金鑰進行 Secure Sockets Layer (SSL) 連線。在連線至 PostgreSQL 資料庫執行個體之任何指令碼、公用程式或應用程式的連線字串中，您可以使用來自 Aurora PostgreSQL 資料庫叢集中主要執行個體或 Aurora 複本的端點和連接埠資訊。在連線字串中，指定來自主要執行個體或 Aurora 複本端點的 DNS 位址做為主機參數。指定來自端點的連接埠號碼做為連接埠參數。

當您連接至 Amazon Aurora PostgreSQL 資料庫叢集中的資料庫執行個體時，您可以執行任何與 PostgreSQL 相容的 SQL 命令。

在 Aurora PostgreSQL 資料庫叢集的詳細資訊檢視中，您可以找到叢集端點名稱、狀態、類型及連接埠號碼。您在 PostgreSQL 連線字串中使用此端點和連接埠號碼。例如，如果端點值為 `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`，則您可以在 PostgreSQL 連線字串中指定下列值：

- 對於主機或主機名稱，指定 `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`
- 對於連接埠，指定 `5432` 或您在建立資料庫叢集時所使用的連接埠值

叢集端點會將您連接至資料庫叢集的主要執行個體。您可以使用叢集端點來同時執行讀取和寫入操作。您的資料庫叢集也可以具有最多 15 個 Aurora 複本，這些複本支援資料庫叢集中資料的唯讀存取。Aurora 叢集中的每個資料庫執行個體 (即主要執行個體和每一個 Aurora 複本) 都有一個獨立於叢集端點的唯一端點。此唯一端點可讓您直接連接至叢集中的特定資料庫執行個體。叢集端點一律指向主要執行個體。如果主要執行個體失敗且遭取代，則叢集端點會指向新的主要執行個體。

若要檢視叢集端點 (寫入者端點)，請在 Amazon RDS 主控台上選擇 Databases (資料庫)，並選擇資料庫叢集的名稱以顯示資料庫叢集詳細資訊。
Aurora PostgreSQL 寫入器端點。

Aurora PostgreSQL 的連線公用程式

一些您可以使用的連線公用程式如下：

- 命令列 – 您可以使用 psql (PostgreSQL 互動式終端機) 之類的工具，來連線至 Aurora PostgreSQL 資料庫叢集。如需使用 PostgreSQL 互動式終端機的詳細資訊，請參閱 PostgreSQL 文件中的 psql。

- GUI – 您可以使用 pgAdmin 公用程式，以使用 UI 界面連線至 Aurora PostgreSQL 資料庫叢集。如需詳細資訊，請參閱 pgAdmin 網站提供的下載頁面。
- AWS 驅動程式：
  - 使用 Amazon Web Services (AWS) JDBC 驅動程式連線至 Aurora PostgreSQL
  - 使用 Amazon Web Services (AWS) Python 驅動程式連線至 Aurora PostgreSQL
  - 使用 Amazon Web Services (AWS) 進階 NodeJS 包裝函式連線至 Aurora PostgreSQL

使用 Amazon Web Services (AWS) JDBC 驅動程式連線至 Aurora PostgreSQL

Amazon Web Services (AWS) JDBC 驅動程式設計為進階 JDBC 包裝函式。此包裝函式與現有的 JDBC 驅動程式互補並擴展其功能，以協助應用程式利用叢集資料庫的功能，例如 Aurora PostgreSQL。驅動程式與社群 pgJDBC 驅動程式相容。

若要安裝 AWS JDBC 驅動程式，請附加 AWS JDBC 驅動程式 .jar 檔案（位於應用程式 CLASSPATH)，並保留對 pgJDBC 社群驅動程式的參考。將連線 URL 字首從 `jdbc:postgresql://` 更新為 `jdbc:aws-wrapper:postgresql://`。

如需 AWS JDBC 驅動程式的詳細資訊和完整的使用說明，請參閱 Amazon Web Services (AWS) JDBC 驅動程式 GitHub 儲存庫。

使用 Amazon Web Services (AWS) Python 驅動程式連線至 Aurora PostgreSQL

Amazon Web Services (AWS) Python 驅動程式設計為進階 Python 包裝函式。此包裝函式是開放原始碼 Psycopg 驅動程式的補充和延伸功能。 AWS Python 驅動程式支援 Python 3.8 版和更新版本。您可以使用 `pip` 命令以及`psycopg`開放原始碼`aws-advanced-python-wrapper`套件來安裝套件。

如需 AWS Python 驅動程式的詳細資訊和完整的使用說明，請參閱 Amazon Web Services (AWS) Python Driver GitHub 儲存庫。

使用 Amazon Web Services (AWS) 進階 NodeJS 包裝函式連線至 Aurora PostgreSQL

AWS 進階 NodeJS 包裝函式是現有 NodeJS 驅動程式的補充和延伸功能。它可協助應用程式利用叢集資料庫的功能，例如 Aurora PostgreSQL。

如需 AWS 進階 NodeJS 包裝函式的詳細資訊以及完整的使用說明，請參閱 Amazon Web Services (AWS) 進階 NodeJS 包裝函式 GitHub 儲存庫。

對 Aurora 連線失敗進行故障診斷

無法連接至新 Aurora 資料庫叢集的常見原因如下：

- VPC 中的安全群組不允許存取 – 您的 VPC 需要透過 VPC 中安全群組的正確設定來允許您的裝置或 Amazon EC2 執行個體的連線。若要解決問題，請修改 VPC 的安全群組 Inbound (入站) 規則以允許連線。如需範例，請參閱 教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4)。
- 防火牆規則封鎖的連接埠 – 檢查為您的 Aurora 資料庫叢集設定的連接埠值。如果防火牆規則封鎖該連接埠，您可以使用不同的連接埠重新建立執行個體。
- IAM 組態不完整或不正確 – 如果您建立 Aurora 資料庫執行個體以使用 IAM – 型身分驗證，請確定已正確設定。如需更多詳細資訊，請參閱 IAM 資料庫身分驗證。

如需 Aurora 資料庫連線疑難排解問題的詳細資訊，請參閱無法連線至 Amazon RDS 資料庫執行個體。

Amazon Aurora 的參數群組

資料庫參數指定資料庫的配置方式。例如，資料庫參數可以指定要配置給資料庫的資源量 (例如記憶體)。

您可利用參數群組建立資料庫執行個體和 Aurora 資料庫叢集的關聯性，以此管理資料庫組態。Aurora 使用預設設定來定義參數群組。您也可以使用自訂設定，定義自己的參數群組。

主題

- 參數群組概觀
- Amazon Aurora 資料庫叢集的資料庫叢集參數群組
- Amazon Aurora 資料庫執行個體的資料庫參數群組
- 比較資料庫參數群組
- 指定資料庫參數

參數群組概觀

資料庫叢集參數群組扮演引擎組態值的容器，而這些值套用到資料庫叢集的每個 Aurora 資料庫執行個體。例如，Aurora 共用儲存模型要求所有 Aurora 叢集中的資料庫執行個體使用相同參數設定，如 `innodb_file_per_table`。因此，影響實體儲存配置的參數是叢集參數群組的一部份。資料庫叢集參數群組也包含所有執行個體層級參數的預設值。

資料庫參數群組扮演引擎組態值的容器，以套用至一或多個資料庫執行個體。資料庫參數群組可應用於 Amazon RDS 和 Aurora 的資料庫執行個體。這些群態設定適用於因 Aurora 叢集內資料庫執行個體而有所不同的屬性，如記憶體緩衝的尺寸。

主題

- 預設和自訂參數群組
- 靜態和動態資料庫叢集參數
- 靜態和動態資料庫執行個體參數
- 字元集參數
- 支援的參數和參數值

預設和自訂參數群組

如果建立的資料庫執行個體未指定資料庫參數群組，則會使用預設的資料庫參數群組。同樣地，如果您建立 Aurora 資料庫叢集，而未指定資料庫叢集參數群組，資料庫叢集將使用預設的資料庫叢集參數群組。每個預設的參數群組將依引擎、運算等級和執行個體分配儲存包含資料庫引擎預設值和 Amazon RDS 系統預設值。

您無法修改預設參數群組的參數設定。相反地，您可以執行下列作業：

1. 建立新的參數群組。
2. 變更所需參數的設定。並非參數群組中的所有資料庫引擎參數都有資格進行修改。
3. 然後修改資料庫執行個體或資料庫叢集，以與新的參數群組建立關聯。

   如需修改資料庫叢集或資料庫執行個體的相關資訊，請參閱 修改 Amazon Aurora 資料庫叢集。

> **Note**
>
> 如果您已將資料庫執行個體修改為使用自訂參數群組，並啟動資料庫執行個體，RDS 會在啟動過程中自動重新啟動資料庫執行個體。對於已啟用 AlwaysOn 或鏡像選項的 RDS for SQL Server 多可用區執行個體，在啟動程序後將執行個體重新開機時，預期會發生容錯移轉。

只有在將資料庫執行個體重新開機之後，RDS 才會在新關聯的參數群組中套用已修改的靜態參數和動態參數。不過，如果您在將資料庫參數群組與資料庫執行個體建立關聯之後修改該群組中的動態參數，則會立即套用這些變更，而不需重新開機。如需變更資料庫叢集參數群組的詳細資訊，請參閱 修改 Amazon Aurora 資料庫叢集。

如果您在資料庫參數群組內更新參數，變更會應用到所有和參數群組建立連結的資料庫執行個體。同樣地，如果您在 Aurora 資料庫叢集參數群組內更新參數，變更會套用到所有與資料庫叢集參數群組相關聯的 Aurora 資料庫叢集。

如果您不想從頭開始建立參數群組，您可以使用 AWS CLI copy-db-parameter-group 命令或 copy-db-cluster-parameter-group 命令複製現有的參數群組。在某些情況下，您可能會發現複製參數群組很有用。例如，您可能想要將其大部份現有參數群組的自訂參數及值併入新的參數群組。

靜態和動態資料庫叢集參數

資料庫叢集參數可為靜態或動態。它們在下列方面有所不同：

- 當您變更靜態參數並儲存資料庫叢集參數群組時，參數變更會在您手動重新啟動每個相關資料庫叢集中的資料庫執行個體後生效。當您使用 AWS 管理主控台 變更靜態資料庫叢集參數值時，一律會pending-reboot針對 使用 ApplyMethod。
- 變更動態參數時，參數變更預設會立即生效，無需重新啟動。當您使用主控台時，其一律使用 immediate 作為 ApplyMethod。若要將參數變更延遲到重新啟動相關聯資料庫叢集中的資料庫執行個體之後，請使用 AWS CLI 或 RDS API。將 ApplyMethod 設定為 pending-reboot 以進行參數變更。

如需使用 AWS CLI 變更參數值的詳細資訊，請參閱 modify-db-cluster-parameter-group。如需使用 RDS API 來變更參數值的更多資訊，請參閱 ModifyDBClusterParameterGroup。

如果變更與資料庫叢集相關聯的資料庫叢集參數群組，請重新啟動資料庫叢集中的資料庫執行個體。重新啟動會將變更套用至資料庫叢集中的所有資料庫執行個體。如要判斷資料庫叢集的資料庫執行個體是否必須重新啟動才能套用變更，請執行下列 AWS CLI 命令。

```
aws rds describe-db-clusters --db-cluster-identifier db_cluster_identifier
```

檢查輸出中主要資料庫執行個體的 DBClusterParameterGroupStatus 值。若值為 pending-reboot，則重新啟動資料庫叢集的資料庫執行個體。

靜態和動態資料庫執行個體參數

資料庫執行個體參數可為靜態或動態。它們不同之處如下：

- 當您變更靜態參數並儲存資料庫參數群組時，參數變更會在您手動重新啟動相關聯的資料庫執行個體後生效。若為靜態參數，主控台一律將 pending-reboot 用於 ApplyMethod。
- 變更動態參數時，參數變更預設會立即生效，無需重新啟動。當您使用 AWS 管理主控台 變更資料庫執行個體參數值時，一律會將 immediate用於ApplyMethod動態參數。若要將參數變更延遲到重新啟動相關聯的資料庫執行個體之後，請使用 AWS CLI 或 RDS API。將 ApplyMethod 設定為 pending-reboot 以進行參數變更。

如需使用 AWS CLI 變更參數值的詳細資訊，請參閱 modify-db-parameter-group。如需使用 RDS API 來變更參數值的更多資訊，請參閱 ModifyDBParameterGroup。

如果資料庫執行個體未在其相關聯的資料庫參數群組中使用最新的變更，主控台會將資料庫參數群組的狀態顯示為 pending-reboot。此狀態不會在下一個維護時段期間造成自動重新啟動。如欲對該資料庫執行個體套用最新參數變更，請手動重新啟動資料庫執行個體。

字元集參數

在建立資料庫叢集之前，先在參數群組中設定與字元集或資料庫定序相關的任何參數。也會在其中建立資料庫之前這樣做。透過此方式，您可以確保預設資料庫和新資料庫皆使用您指定的字元集和定序值。如果您變更字元集和定序參數，參數變更將不會套用到現有資料庫。

針對部分資料庫引擎，您可以使用 ALTER DATABASE 命令變更現有資料庫的字元集或定序值，例如：

```
ALTER DATABASE database_name CHARACTER SET character_set_name COLLATE collation;
```

如需變更資料庫的字元集或定序值的詳細資訊，請參閱適用於您資料庫引擎的文件。

支援的參數和參數值

若要判斷資料庫引擎的支援參數，您可以檢視資料庫執行個體或資料庫叢集所使用的資料庫參數群組和資料庫叢集參數群組中的參數。如需詳細資訊，請參閱在 Amazon Aurora 中檢視資料庫參數群組的參數值及在 Amazon Aurora 中檢視資料庫叢集參數群組的參數值。

在許多情況下，您可使用運算式、公式和函數來指定整數及布林值參數。函數可包含數學的對數表達式。但是，並非所有參數都支援參數值的表達式、公式和函數。如需詳細資訊，請參閱指定資料庫參數。

如果是 Aurora 全球資料庫，您可以針對個別的 Aurora 叢集來指定不同的組態設定。請確保這些設定有足夠的相似度，而足以在您將次要叢集提升為主要叢集時產生一致的行為。例如，在 Aurora 全球資料庫的所有叢集上，對時區和字元集使用相同的設定。

未正確設定參數群組中的參數，可能產生各種意外影響，包括效能降低和系統不穩定。修改資料庫參數時請務必謹慎，在修改參數群組之前，請備份資料。在將這些參數群組變更套用到生產資料庫執行個體或資料庫叢集之前，請在測試資料庫執行個體或資料庫叢集上嘗試參數群組設定變更。

Amazon Aurora 資料庫叢集的資料庫叢集參數群組

Amazon Aurora 資料庫叢集使用資料庫叢集參數群組。以下各節介紹資料庫叢集參數群組的設定和管理。

主題

- Amazon Aurora 資料庫叢集和資料庫執行個體參數
- 在 Amazon Aurora 中建立資料庫叢集參數群組
- 將資料庫叢集參數群組與 Amazon Aurora 中的資料庫叢集建立關聯
- 在 Amazon Aurora 中修改資料庫叢集參數群組中的參數

- 在 Amazon Aurora 中重設資料庫叢集參數群組中的參數
- 在 Amazon Aurora 中複製資料庫叢集參數群組
- 在 Amazon Aurora 中列出資料庫叢集參數群組
- 在 Amazon Aurora 中檢視資料庫叢集參數群組的參數值
- 在 Amazon Aurora 中刪除資料庫叢集參數群組

Amazon Aurora 資料庫叢集和資料庫執行個體參數

Aurora 使用組態設定的兩層系統：

- 資料庫叢集參數群組中的參數將套用到資料庫叢集中的每個資料庫執行個體。您的資料儲存在 Aurora 共享儲存子系統中。因此，所有和資料表資料的實體配置相關的參數都必須和 Aurora 叢集中的所有資料庫執行個體相同。同樣地，因為 Aurora 資料庫執行個體透過複寫連線，所有複寫設定的參數都必須和整個 Aurora 叢集相同。
- 資料庫參數群組中的參數將套用到 Aurora 資料庫叢集中的單一資料庫執行個體。這些參數和某些面向相關，如相同的 Aurora 叢集中依資料庫執行個體而不同的記憶體用量。例如，叢集常包括不同 AWS 執行個體類別的資料庫執行個體。

每個 Aurora 叢集都會和一個資料庫叢集參數群組建立關聯。此參數群組為相應資料庫引擎的每個組態值分配預設值。資料庫叢集參數群組包含叢集層級參數和執行個體層級參數兩者的預設值。佈建或 Aurora Serverless v2 叢集中的每個資料庫執行個體會繼承來自該資料庫叢集參數群組的設定。

每個資料庫執行個體也會與資料庫參數群組相關聯。資料庫參數群組中的值可覆寫來自叢集參數群組的預設值。例如，若叢集中的一個執行個體遇到問題，您可將自訂資料庫參數群組指派給該執行個體。自訂參數群組可能具有與偵錯或效能調校相關參數的特定設定。

當您根據指定的資料庫引擎和版本，建立叢集或新資料庫執行個體時，Aurora 會指派預設參數群組。您可改為指定自訂參數群組。您可自行建立參數群組，並可編輯參數值。您可在建立時指定這些自訂參數群組。您還可於稍後修改資料庫叢集或執行個體以使用自訂參數群組。

若為已佈建和 Aurora Serverless v2 執行個體，您在資料庫叢集參數群組中修改的任何組態值，都會覆寫資料庫參數群組中的預設值。若您在資料庫參數群組中編輯對應的值，則那些值會覆寫來自資料庫叢集參數群組的設定。

即使您將組態參數改回預設值，您修改的資料庫參數設定之優先順序高於資料庫叢集參數群組值。您可以使用 describe-db-parameters AWS CLI 命令或 DescribeDBParameters RDS API 操作來查看要覆寫哪些參數。如果您修改該參數，則 `Source` 欄位包括值 `user`。若要重設一或多個參數，讓資料庫叢

集參數群組的值優先，請使用 reset-db-parameter-group AWS CLI 命令或 ResetDBParameterGroup RDS API 操作。

Aurora 中可供您使用的資料庫叢集和資料庫執行個體參數，根據資料庫引擎相容性而有所不同。

資料庫引擎	參數
Aurora MySQL	請參閱 Aurora MySQL 組態參數。 若為 Aurora Serverless 叢集，請參閱 使用 Aurora Serverless v2 的參數群組 和 Aurora Serverless v1 的參數群組 中的其他詳細資料。
Aurora PostgreSQL	請參閱 Amazon Aurora PostgreSQL 參數。 若為 Aurora Serverless 叢集，請參閱 使用 Aurora Serverless v2 的參數群組 和 Aurora Serverless v1 的參數群組 中的其他詳細資料。

> **Note**
>
> Aurora Serverless v1 叢集僅具有資料庫叢集參數群組，而非資料庫參數群組。若為 Aurora Serverless v2 叢集，您可在資料庫叢集參數群組中對自訂參數進行所有變更。
> Aurora Serverless v2 使用資料庫叢集參數群組和資料庫參數群組兩者。利用 Aurora Serverless v2，您可修改幾乎所有的組態參數。Aurora Serverless v2 會覆寫某些與容量相關組態參數的設定，則當 Aurora Serverless v2 執行個體縮減規模時，不會中斷您的工作負載。如需進一步了解 Aurora Serverless 組態設定和您可修改的設定，請參閱 使用 Aurora Serverless v2 的參數群組 和 Aurora Serverless v1 的參數群組。

在 Amazon Aurora 中建立資料庫叢集參數群組

您可以使用 AWS CLI、 或 RDS API AWS 管理主控台建立新的資料庫叢集參數群組。

建立資料庫叢集參數群組後，請等待至少 5 分鐘，然後再建立使用該資料庫叢集參數群組的資料庫叢集。執行此動作允許 Amazon RDS 完整建立參數群組，再將其供新資料庫叢集使用。您可以使用 Amazon RDS 主控台的 Parameter groups (參數群組) 頁面或 describe-db-cluster-parameters 命令確認是否已建立資料庫叢集參數群組。

下列限制適用於資料庫叢集參數群組名稱：

- 名稱必須為 1 到 255 個字母、數字或連字號。

  預設參數群組名稱可以包括句點，例如 `default.aurora-mysql5.7`。不過，自訂參數群組名稱不能包括句點。
- 第一個字元必須是字母。
- 名稱不能以連字號結尾或包含兩個連續連字號。

主控台

建立資料庫叢集參數群組

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。
3. 選擇 Create parameter group (建立參數群組)。
4. 對於參數群組名稱，輸入新資料庫叢集參數群組的名稱。
5. 對於描述，輸入新資料庫叢集參數群組的描述。
6. 對於引擎類型，選擇您的資料庫引擎。
7. 對於參數群組系列，選擇資料庫參數群組系列。
8. 選擇建立。

AWS CLI

若要建立資料庫叢集參數群組，請使用 AWS CLI `create-db-cluster-parameter-group` 命令。

以下範例將為 Aurora MySQL 5.7 版建立名為 mydbclusterparametergroup，描述為 My new cluster parameter group (我的新叢集參數群組) 的資料庫叢集參數群組。

包含下列必要參數：

- `--db-cluster-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

若要列出所有可用的參數群組系列，請使用下列命令：

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
>
> 輸出包含重覆項目。

Example

對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --db-parameter-group-family aurora-mysql5.7 \
    --description "My new cluster parameter group"
```

在 Windows 中：

```
aws rds create-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --db-parameter-group-family aurora-mysql5.7 ^
    --description "My new cluster parameter group"
```

此命令會產生類似下列的輸出：

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "mydbclusterparametergroup",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "My new cluster parameter group",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
    }
}
```

RDS API

若要建立資料庫叢集參數群組，請使用 RDS API CreateDBClusterParameterGroup 動作。

包含下列必要參數：

- `DBClusterParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

將資料庫叢集參數群組與 Amazon Aurora 中的資料庫叢集建立關聯

您可以使用自訂設定，建立自己的資料庫叢集參數群組。您可以使用 AWS 管理主控台、AWS CLI 或 RDS API，將資料庫叢集參數群組與資料庫叢集建立關聯。您可以在建立或修改資料庫叢集時執行此動作。

如需建立資料庫叢集參數群組的詳細資訊，請參閱在 Amazon Aurora 中建立資料庫叢集參數群組。如需建立資料庫叢集的詳細資訊，請參閱建立 Amazon Aurora 資料庫叢集。如需修改資料庫叢集的詳細資訊，請參閱修改 Amazon Aurora 資料庫叢集。

> **Note**
>
> 對於 Aurora PostgreSQL 15.2、14.7、13.10、12.14 和所有的第 11 版，當您變更與資料庫叢集相關聯的資料庫叢集參數群組後，請將每個複本執行個體重新開機，以套用變更。
> 若要判斷資料庫叢集的主要資料庫執行個體是否必須重新啟動才能套用變更，請執行下列 AWS CLI 命令：
> `aws rds describe-db-clusters --db-cluster-identifier` *`db_cluster_identifier`*
> 檢查輸出中主要資料庫執行個體的 `DBClusterParameterGroupStatus` 值。如果值為 `pending-reboot`，則重新啟動資料庫叢集的主要資料庫執行個體。

主控台

將資料庫叢集參數群組與資料庫叢集建立關聯

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中選擇 Databases (資料庫)，然後選取您要修改的資料庫叢集。
3. 選擇 Modify (修改)。Modify DB cluster (修改資料庫叢集) 頁面隨即出現。
4. 變更 DB cluster parameter group (資料庫叢集參數群組) 設定。

5. 選擇 Continue (繼續)，並檢查修改的摘要。

   無論 Scheduling of modifications (修改排程) 設定為何，都會立即套用變更。

6. 在確認頁面上，檢閱您的變更。如果都正確，請選擇 Modify cluster (修改叢集) 以儲存您的變更。

   或者，選擇 Back (上一步) 以編輯變更，或是選擇 Cancel (取消) 以取消變更。

AWS CLI

若要將資料庫叢集參數群組與資料庫叢集建立關聯，請使用 AWS CLI `modify-db-cluster` 命令搭配下列選項：

- `--db-cluster-name`
- `--db-cluster-parameter-group-name`

下面的範例將 `mydbclpg` 資料庫參數群組與 `mydbcluster` 資料庫叢集建立關聯。

Example

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --db-cluster-parameter-group-name mydbclpg
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --db-cluster-parameter-group-name mydbclpg
```

RDS API

若要將資料庫叢集參數群組與資料庫叢集建立關聯，請使用 RDS API `ModifyDBCluster` 操作搭配下列參數：

- `DBClusterIdentifier`
- `DBClusterParameterGroupName`

在 Amazon Aurora 中修改資料庫叢集參數群組中的參數

您可以修改客戶建立的資料庫叢集參數群組中參數值。您無法變更預設資料庫叢集參數群組中的參數值。客戶建立的資料庫叢集參數群組中的參數變更會套用到與該資料庫叢集參數群組關聯的所有資料庫叢集。

主控台

修改資料庫叢集參數群組

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。
3. 在清單中，選擇您要修改的參數群組。
4. 針對 Parameter group actions (參數群組動作)，選擇 Edit (編輯)。
5. 變更您想要修改的參數值。您可用對話方塊右上角的箭頭鍵來捲動參數。

   您無法變更預設參數群組中的值。
6. 選擇儲存變更。
7. 將叢集中的主要 (寫入器) 資料庫執行個體重新開機，以套用變更。
8. 然後，將讀取器資料庫執行個體重新開機，將套用其變更。

   若未將資料庫執行個體重新開機，容錯移轉操作可能會需要比平常更長的時間。

AWS CLI

若要修改資料庫叢集參數群組，請使用 AWS CLI `modify-db-cluster-parameter-group` 命令搭配下列必要參數：

- `--db-cluster-parameter-group-name`
- `--parameters`

以下範例將修改名稱為 mydbclusterparametergroup 的資料庫叢集參數群組的 `server_audit_logging` 和 `server_audit_logs_upload` 值。

Example

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" \

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

在 Windows 中：

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

該命令會產生類似以下的輸出：

```
DBCLUSTERPARAMETERGROUP  mydbclusterparametergroup
```

RDS API

若要修改資料庫叢集參數群組，請使用 RDS API ModifyDBClusterParameterGroup 命令並搭配下列必要參數：

- `DBClusterParameterGroupName`
- `Parameters`

在 Amazon Aurora 中重設資料庫叢集參數群組中的參數

您可以將參數重設為客戶建立的資料庫叢集參數群組中的預設值。客戶建立的資料庫叢集參數群組中的參數變更會套用到與該資料庫叢集參數群組關聯的所有資料庫叢集。

> **Note**
>
> 在預設的資料庫叢集參數群組中，參數永遠設定為其預設值。

主控台

將資料庫叢集參數群組中的參數重設為其預設值

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。
3. 在清單中，選擇參數群組。
4. 針對 Parameter group actions (參數群組動作)，選擇 Edit (編輯)。
5. 選擇您要重設為預設值的參數。您可用對話方塊右上角的箭頭鍵來捲動參數。

   您無法重設預設參數群組中的值。
6. 選擇「Reset (重設) 」，然後選擇「Reset parameters (重設參數) 」來確認。
7. 重新啟動資料庫叢集中的主要資料庫執行個體，以將變更套用至資料庫叢集中的所有資料庫執行個體。

AWS CLI

若要將資料庫叢集參數群組中的參數重設為其預設值，請使用 AWS CLI `reset-db-cluster-parameter-group`命令搭配下列必要選項：`--db-cluster-parameter-group-name`。

若要重設資料庫叢集參數群組中的所有參數，請指定 `--reset-all-parameters` 選項。若要重設特定參數，請指定 `--parameters` 選項。

下列範例會將資料庫參數群組中所有名為 mydbparametergroup 的參數重設為其預設值。

Example

對於 Linux、macOS 或 Unix：

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

在 Windows 中：

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbparametergroup ^
    --reset-all-parameters
```

以下範例會將名為 mydbclusterparametergroup 的資料庫叢集參數群組中的 `server_audit_logging` 和 `server_audit_logs_upload` 重設為其預設值。

Example

對於 Linux、macOS 或 Unix：

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --parameters "ParameterName=server_audit_logging,ApplyMethod=immediate" \
                 "ParameterName=server_audit_logs_upload,ApplyMethod=immediate"
```

在 Windows 中：

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

該命令會產生類似以下的輸出：

```
DBClusterParameterGroupName  mydbclusterparametergroup
```

RDS API

若要將資料庫叢集參數群組中的參數重設為其預設值，請使用 RDS API [ResetDBClusterParameterGroup](#) 命令與下列必要參數：`DBClusterParameterGroupName`。

若要重設資料庫叢集參數群組中的所有參數，請將 `ResetAllParameters` 參數設定為 `true`。若要重設特定參數，請指定 `Parameters` 參數。

在 Amazon Aurora 中複製資料庫叢集參數群組

您可複製所建立的自訂資料庫叢集參數群組。如果您已建立資料庫叢集參數群組，並且希望在新資料庫叢集參數群組中包含該組中的大多數自訂參數和值，則複製參數群組是一種方便的解決方案。您可以使用 AWS CLI [copy-db-cluster-parameter-group](#) 命令或 RDS API [CopyDBClusterParameterGroup](#) 操作來複製資料庫叢集參數群組。

複製資料庫叢集參數群組後，請等待至少 5 分鐘，然後再建立使用該資料庫叢集參數群組的資料庫叢集。執行此動作允許 Amazon RDS 完整複製參數群組，再將其供新資料庫叢集使用。您可以使用

Amazon RDS 主控台的 Parameter groups (參數群組) 頁面或 describe-db-cluster-parameters 命令確認是否已建立資料庫叢集參數群組。

> **Note**
>
> 您無法複製預設參數群組。但您可以依照預設參數群組建立新的參數群組。
> 您無法將資料庫叢集參數群組複製到不同的 AWS 帳戶 或 AWS 區域。

主控台

複製資料庫叢集參數群組

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。
3. 在清單中，選擇您要複製的自訂參數群組。
4. 針對 Parameter group actions (參數群組動作)，選擇 Copy (複製)。
5. 在 New DB parameter group identifier (新資料庫參數群組識別符) 中，輸入新參數群組的名稱。
6. 在 Description (描述) 中，輸入新參數群組的描述。
7. 請選擇 Copy (複製)。

AWS CLI

若要複製資料庫叢集參數群組，請使用 AWS CLI `copy-db-cluster-parameter-group` 命令搭配下列必要參數：

- `--source-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-description`

以下範例會建立名為 `mygroup2` 的新資料庫叢集參數群組，該群組為資料庫叢集參數群組 `mygroup1` 的複本。

Example

對於 Linux、macOS 或 Unix：

```
aws rds copy-db-cluster-parameter-group \
    --source-db-cluster-parameter-group-identifier mygroup1 \
    --target-db-cluster-parameter-group-identifier mygroup2 \
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

在 Windows 中：

```
aws rds copy-db-cluster-parameter-group ^
    --source-db-cluster-parameter-group-identifier mygroup1 ^
    --target-db-cluster-parameter-group-identifier mygroup2 ^
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

RDS API

若要複製資料庫叢集參數群組，請使用 RDS API CopyDBClusterParameterGroup 操作並搭配下列必要參數：

- SourceDBClusterParameterGroupIdentifier
- TargetDBClusterParameterGroupIdentifier
- TargetDBClusterParameterGroupDescription

在 Amazon Aurora 中列出資料庫叢集參數群組

您可以列出您為 AWS 帳戶建立的資料庫叢集參數群組。

> **Note**
>
> 當您為特定資料庫引擎和版本建立資料庫叢集時，會從預設的參數範本自動建立預設的參數群組。這些預設參數群組包含偏好的參數設定，無法修改。當您建立自訂參數群組時，您可以修改參數設定。

主控台

列出 AWS 帳戶的所有資料庫叢集參數群組

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。

2. 在導覽窗格中，選擇 Parameter groups (參數群組)。

   資料庫叢集參數群組會以 DB cluster parameter group (資料庫叢集參數群組) 出現在 Type (類型) 清單中。

AWS CLI

若要列出 AWS 帳戶的所有資料庫叢集參數群組，請使用 AWS CLI [describe-db-cluster-parameter-groups](#) 命令。

Example

以下範例列出 AWS 帳戶的所有可用資料庫叢集參數群組。

```
aws rds describe-db-cluster-parameter-groups
```

以下範例說明 mydbclusterparametergroup 參數群組。

對於 Linux、macOS 或 Unix：

```
aws rds describe-db-cluster-parameter-groups \
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

在 Windows 中：

```
aws rds describe-db-cluster-parameter-groups ^
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

此命令會傳回類似以下的回應：

```
{
    "DBClusterParameterGroups": [
        {
            "DBClusterParameterGroupName": "mydbclusterparametergroup",
            "DBParameterGroupFamily": "aurora-mysql5.7",
            "Description": "My new cluster parameter group",
            "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
        }
    ]
```

```
}
```

RDS API

若要列出 AWS 帳戶的所有資料庫叢集參數群組，請使用 RDS API [DescribeDBClusterParameterGroups](#)動作。

在 Amazon Aurora 中檢視資料庫叢集參數群組的參數值

您可以從資料庫叢集參數群組取得所有參數與其值的清單。

主控台

檢視資料庫叢集參數群組的參數值

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。

   資料庫叢集參數群組會以 DB cluster parameter group (資料庫叢集參數群組) 出現在 Type (類型) 清單中。
3. 選擇資料庫叢集參數群組的名稱，以查看參數清單。

AWS CLI

若要檢視資料庫叢集參數群組的參數值，請使用 AWS CLI [describe-db-cluster-parameters](#) 命令搭配下列必要參數。

- `--db-cluster-parameter-group-name`

Example

以下範例列出資料庫叢集參數群組 mydbclusterparametergroup 的參數和參數值，為 JSON 格式。

此命令會傳回類似以下的回應：

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-
name mydbclusterparametergroup
```

```
{
```

```
    "Parameters": [
        {
            "ParameterName": "allow-suspicious-udfs",
            "Description": "Controls whether user-defined functions that have only an
 xxx symbol for the main function can be loaded",
            "Source": "engine-default",
            "ApplyType": "static",
            "DataType": "boolean",
            "AllowedValues": "0,1",
            "IsModifiable": false,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },
        {
            "ParameterName": "aurora_binlog_read_buffer_size",
            "ParameterValue": "5242880",
            "Description": "Read buffer size used by master dump thread when the switch
 aurora_binlog_use_large_read_buffer is ON.",
            "Source": "engine-default",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "8192-536870912",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },

...
```

RDS API

若要檢視資料庫叢集參數群組的參數值，請搭配下列必要參數使用 RDS API [DescribeDBClusterParameters](#) 命令。

- `DBClusterParameterGroupName`

在某些情況下，不會顯示允許的參數值。這些一律為參數，其中來源是資料庫引擎預設值。

若要檢視這些參數的值，您可以執行下列 SQL 陳述式：

- MySQL：

```
-- Show the value of a particular parameter
mysql$ SHOW VARIABLES LIKE '%parameter_name%';

-- Show the values of all parameters
mysql$ SHOW VARIABLES;
```

- PostgreSQL：

```
-- Show the value of a particular parameter
postgresql=> SHOW parameter_name;

-- Show the values of all parameters
postgresql=> SHOW ALL;
```

在 Amazon Aurora 中刪除資料庫叢集參數群組

您可以使用 AWS 管理主控台 AWS CLI或 RDS API 刪除資料庫叢集參數群組。資料庫叢集參數群組參數群組只有在未與資料庫叢集相關聯時，才符合刪除條件。

主控台

刪除參數群組

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。

   參數群組隨即出現在清單中。
3. 選擇要刪除的資料庫叢集參數群組的名稱。
4. 選擇動作，然後選擇刪除。
5. 檢閱參數群組名稱，然後選擇刪除。

AWS CLI

若要刪除資料庫叢集參數群組，請使用 AWS CLI `delete-db-cluster-parameter-group` 命令搭配下列必要參數。

- `--db-parameter-group-name`

Example

下列範例會刪除名為 mydbparametergroup 的資料庫叢集參數群組。

```
aws rds delete-db-cluster-parameter-group --db-parameter-group-name mydbparametergroup
```

RDS API

若要刪除資料庫叢集參數群組，請使用 RDS API `DeleteDBClusterParameterGroup` 命令搭配下列必要參數。

- `DBParameterGroupName`

Amazon Aurora 資料庫執行個體的資料庫參數群組

資料庫執行個體會使用資料庫參數群組。以下各節介紹資料庫執行個體參數群組的設定和管理。

主題

- 在 Amazon Aurora 中建立資料庫參數群組
- 將資料庫參數群組與 Amazon Aurora 中的資料庫執行個體建立關聯
- 修改 Amazon Aurora 中的資料庫參數群組中的參數
- 在 Amazon Aurora 中將資料庫參數群組中的參數重設為預設值
- 在 Amazon Aurora 中複製資料庫參數群組
- 列出 Amazon Aurora 中的資料庫參數群組
- 在 Amazon Aurora 中檢視資料庫參數群組的參數值
- 在 Amazon Aurora 中刪除資料庫參數群組

在 Amazon Aurora 中建立資料庫參數群組

您可以使用 AWS 管理主控台、 AWS CLI或 RDS API 建立新的資料庫參數群組。

下列限制適用於資料庫參數群組名稱：

- 名稱必須為 1 到 255 個字母、數字或連字號。

預設參數群組名稱可以包括句點，例如 `default.mysql8.0`。不過，自訂參數群組名稱不能包括句點。

- 第一個字元必須是字母。
- 名稱不能以連字號結尾或包含兩個連續連字號。

主控台

建立資料庫參數群組

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。
3. 選擇 Create parameter group (建立參數群組)。
4. 對於參數群組名稱，輸入新資料庫參數群組的名稱。
5. 對於描述，輸入新資料庫參數群組的描述。
6. 針對引擎類型，選擇您的資料庫引擎。
7. 對於參數群組系列，選擇資料庫參數群組系列。
8. 對於類型 (如果適用)，選擇資料庫參數群組。
9. 選擇建立。

AWS CLI

若要建立資料庫參數群組，請使用 AWS CLI `create-db-parameter-group` 命令。下列範例將為 MySQL 8.0 版建立一個名為 mydbparametergroup 的資料庫參數群組，其說明為「My new parameter group (我的新參數群組)」。

包含下列必要參數：

- `--db-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

若要列出所有可用的參數群組系列，請使用下列命令：

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
>
> 輸出包含重覆項目。

Example

針對 Linux、macOS 或 Unix：

```
aws rds create-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --db-parameter-group-family aurora-mysql5.7 \
    --description "My new parameter group"
```

在 Windows 中：

```
aws rds create-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --db-parameter-group-family aurora-mysql5.7 ^
    --description "My new parameter group"
```

此命令會產生類似下列的輸出：

```
DBPARAMETERGROUP  mydbparametergroup  aurora-mysql5.7  My new parameter group
```

RDS API

若要建立資料庫參數群組，請使用 RDS API CreateDBParameterGroup 操作。

包含下列必要參數：

- DBParameterGroupName
- DBParameterGroupFamily
- Description

將資料庫參數群組與 Amazon Aurora 中的資料庫執行個體建立關聯

您可以使用自訂設定，建立自己的資料庫參數群組。您可以使用 AWS 管理主控台 AWS CLI、 或 RDS API，將資料庫參數群組與資料庫執行個體建立關聯。您可以在建立或修改資料庫執行個體時執行此動作。

如需建立資料庫參數群組的資訊，請參閱在 Amazon Aurora 中建立資料庫參數群組。如需修改資料庫執行個體的相關資訊，請參閱 修改資料庫叢集中的資料庫執行個體。

> **Note**
>
> 當您建立新資料庫參數群組與資料庫執行個體的關聯時，只有在資料庫執行個體重新開機之後，才會套用修改過的靜態參數和動態參數。不過，如果您在將資料庫參數群組與資料庫執行個體建立關聯之後修改該群組中的動態參數，則會立即套用這些變更，而不需重新開機。

主控台

將資料庫參數群組與資料庫執行個體建立關聯

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中選擇 Databases (資料庫)，然後選擇您要修改的資料庫執行個體。
3. 選擇 Modify (修改)。Modify DB instance (修改資料庫執行個體) 頁面隨即出現。
4. 變更 DB parameter group (資料庫參數群組) 設定。
5. 選擇 Continue (繼續)，並檢查修改的摘要。
6. (選用) 選擇 Apply immediately (立即套用) 以立即套用變更。在某些情況下，選擇此選項會導致停機。
7. 在確認頁面上，檢閱您的變更。如果都正確，請選擇 Modify DB instance (修改資料庫執行個體) 以儲存您的變更。

   或者，選擇 Back (上一步) 以編輯變更，或是選擇 Cancel (取消) 以取消變更。

AWS CLI

若要將資料庫參數群組與資料庫執行個體建立關聯，請使用 AWS CLI `modify-db-instance`命令搭配下列選項：

- `--db-instance-identifier`
- `--db-parameter-group-name`

下面的範例將 mydbpg 資料庫參數群組與 database-1 資料庫執行個體建立關聯。透過 `--apply-immediately`，即可立即套用變更。使用 `--no-apply-immediately` 在下次維護時段套用變更。

Example

在 Linux、macOS 或 Unix 中：

```
aws rds modify-db-instance \
    --db-instance-identifier database-1 \
    --db-parameter-group-name mydbpg \
    --apply-immediately
```

在 Windows 中：

```
aws rds modify-db-instance ^
    --db-instance-identifier database-1 ^
    --db-parameter-group-name mydbpg ^
    --apply-immediately
```

RDS API

若要將資料庫參數群組與資料庫執行個體建立關聯，請使用 RDS API `ModifyDBInstance` 操作搭配下列參數：

- `DBInstanceName`
- `DBParameterGroupName`

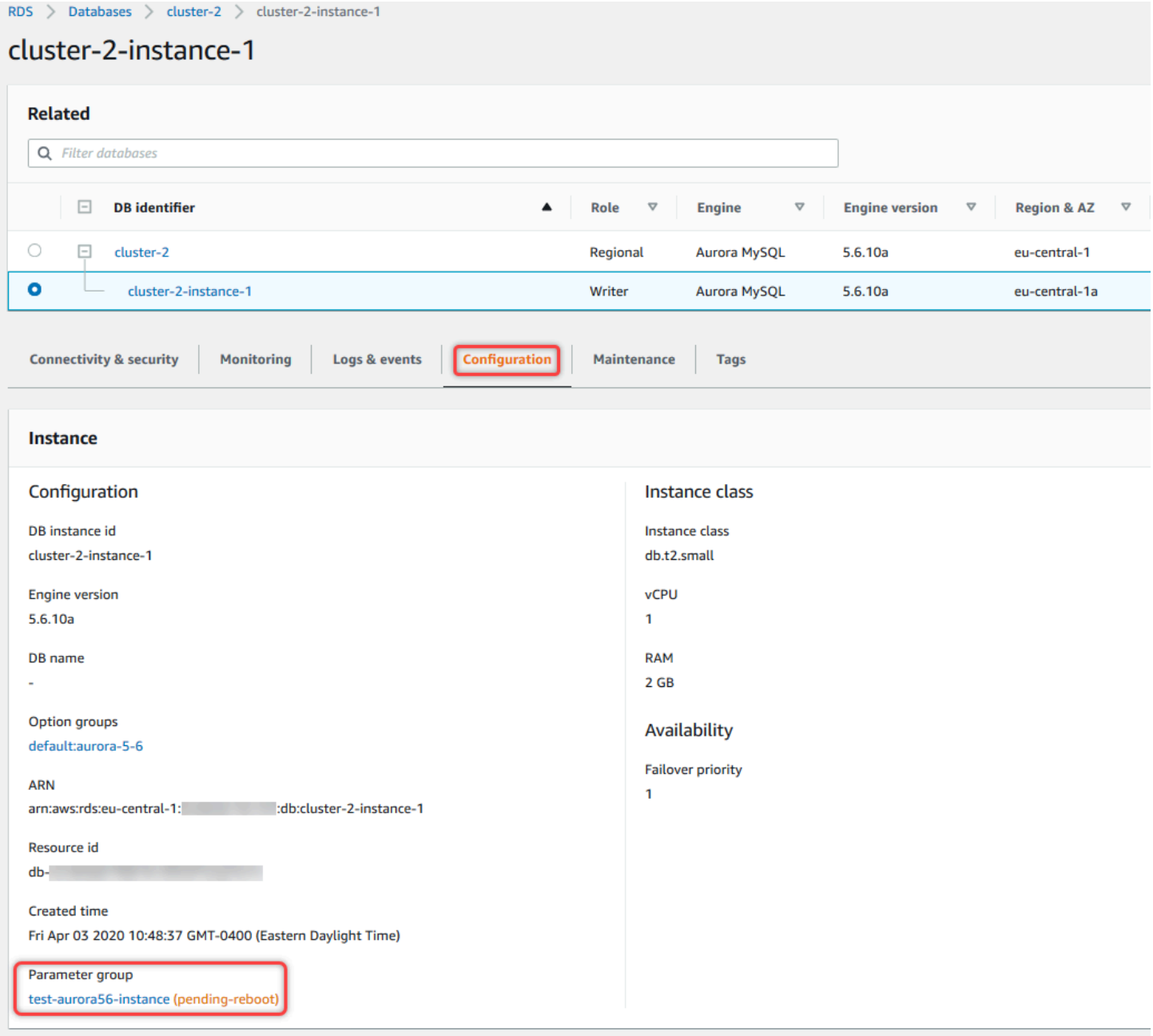
修改 Amazon Aurora 中的資料庫參數群組中的參數

您可修改客戶建立的資料庫參數群組中的參數值；但無法變更預設資料庫參數群組中的參數值。客戶建立的資料庫參數群組中的參數變更會套用到與該資料庫參數群組關聯的所有資料庫執行個體。

有兩種類型的參數：動態參數和靜態參數。動態參數的變更會立即套用至資料庫執行個體，而不會重新啟動。靜態參數的變更只會在資料庫執行個體重新啟動後套用。

RDS 主控台會在 Configuration (組態) 標籤上，顯示與資料庫執行個體相關聯的資料庫參數群組的狀態。例如，假如資料庫執行個體未在其關聯的資料庫參數群組中使用最新的變更，RDS 主控台將顯示

資料庫參數群組的狀態為 pending-reboot (等待重新啟動)。如欲對該資料庫執行個體套用最新參數變更，請手動重新啟動資料庫執行個體。



主控台

修改資料庫參數群組中的參數

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。

3. 在清單中，選擇您要修改的參數群組的名稱。
4. 針對 Parameter group actions (參數群組動作)，選擇 Edit (編輯)。
5. 變更您要修改的參數值。您可用對話方塊右上角的箭頭鍵來捲動參數。

   您無法變更預設參數群組中的值。
6. 選擇儲存變更。

AWS CLI

若要修改資料庫參數群組，請使用 AWS CLI `modify-db-parameter-group` 命令搭配下列必要選項：

- `--db-parameter-group-name`
- `--parameters`

以下範例將修改名稱為 mydbparametergroup 的資料庫參數群組的 `max_connections` 和 `max_allowed_packet` 值。

Example

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" \

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

在 Windows 中：

```
aws rds modify-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" ^

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

該命令會產生類似以下的輸出：

```
DBPARAMETERGROUP  mydbparametergroup
```

RDS API

若要修改資料庫參數群組，請使用 RDS API `ModifyDBParameterGroup` 操作，並搭配下列必要參數：

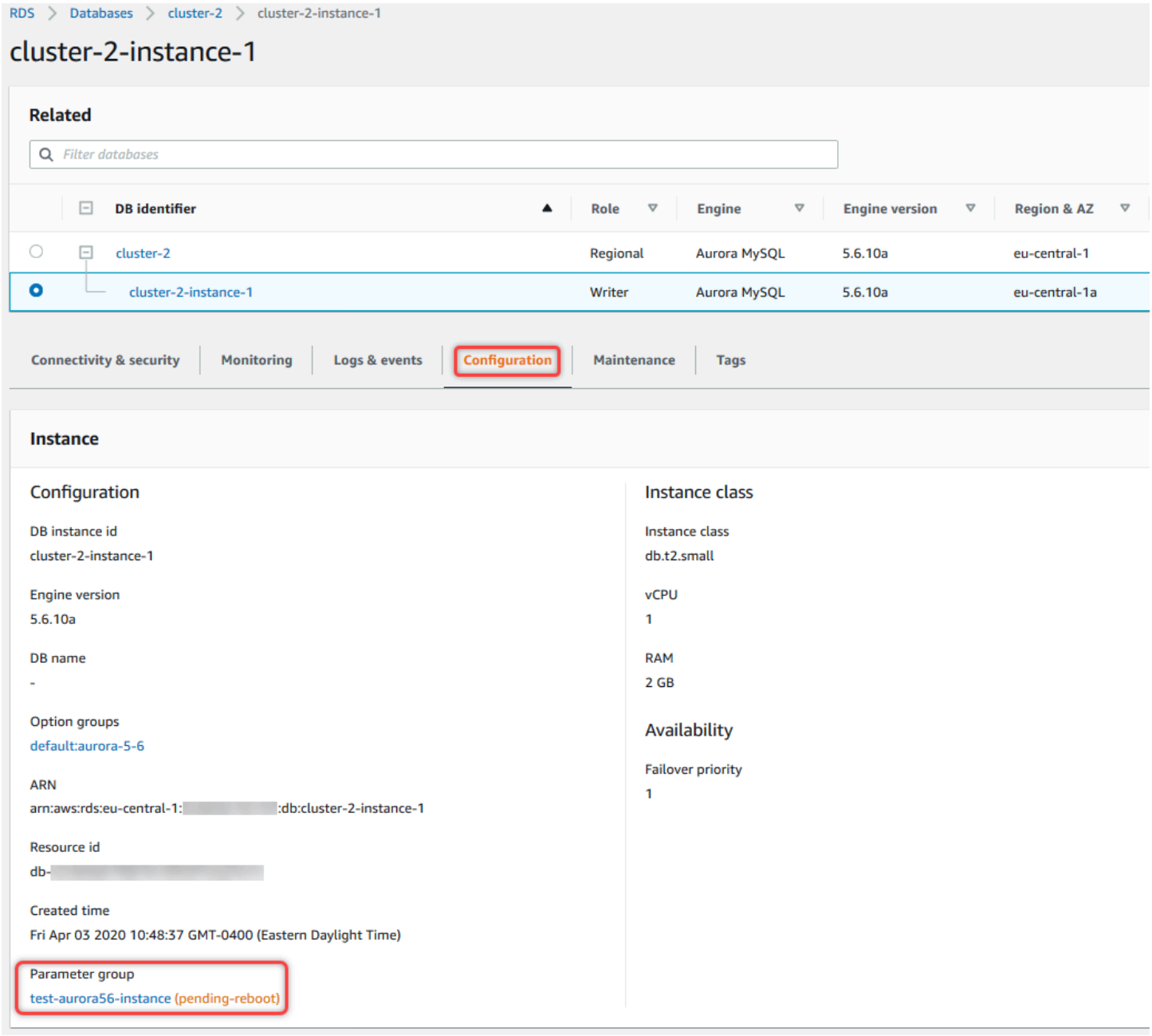
- `DBParameterGroupName`
- `Parameters`

在 Amazon Aurora 中將資料庫參數群組中的參數重設為預設值

您可以將客戶建立的資料庫參數群組中的參數值重設為其預設值。客戶建立的資料庫參數群組中的參數變更會套用到與該資料庫參數群組關聯的所有資料庫執行個體。

使用主控台時，您可以將特定參數重設為其預設值。不過，您無法輕易地一次重設資料庫參數群組中的所有參數。當您使用 AWS CLI 或 RDS API 時，您可以將特定參數重設為其預設值。您也可以一次重設資料庫參數群組中的所有參數。

某些參數的變更會立即套用到資料庫執行個體，而不需要重新啟動。其他參數的變更只有在資料庫執行個體重新啟動之後才會套用。RDS 主控台會在 Configuration (組態) 標籤上，顯示與資料庫執行個體相關聯的資料庫參數群組的狀態。例如，假設資料庫執行個體未使用其相關聯之資料庫參數群組的最新變更。若是如此，RDS 主控台會顯示狀態為 pending-reboot 的資料庫參數群組。如欲對該資料庫執行個體套用最新參數變更，請手動重新啟動資料庫執行個體。



> **Note**
>
> 在預設資料庫參數群組中，參數始終設定為其預設值。

主控台

將資料庫參數群組中的參數重設為其預設值

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。
3. 在清單中，選擇參數群組。
4. 針對 Parameter group actions (參數群組動作)，選擇 Edit (編輯)。
5. 選擇您要重設為預設值的參數。您可用對話方塊右上角的箭頭鍵來捲動參數。

   您無法重設預設參數群組中的值。
6. 選擇「Reset (重設) 」，然後選擇「Reset parameters (重設參數) 」來確認。

AWS CLI

若要重設資料庫參數群組中的部分或全部參數，請使用 AWS CLI `reset-db-parameter-group`命令搭配下列必要選項：`--db-parameter-group-name`。

若要重設資料庫參數群組中的所有參數，請指定 `--reset-all-parameters` 選項。若要重設特定參數，請指定 `--parameters` 選項。

下列範例會將資料庫參數群組中所有名為 mydbparametergroup 的參數重設為其預設值。

Example

對於 Linux、macOS 或 Unix：

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

在 Windows 中：

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --reset-all-parameters
```

下列範例會在名為 mydbparametergroup 的資料庫參數群組中，將 max_connections 與 max_allowed_packet 選項重設為其預設值。

Example

對於 Linux、macOS 或 Unix：

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" \
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

在 Windows 中：

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" ^
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

該命令會產生類似以下的輸出：

```
DBParameterGroupName  mydbparametergroup
```

RDS API

若要將資料庫參數群組中的參數重設為其預設值，請使用 RDS API ResetDBParameterGroup 命令與下列必要參數：DBParameterGroupName。

若要重設資料庫參數群組中的所有參數，請將 ResetAllParameters 參數設定為 true。若要重設特定參數，請指定 Parameters 參數。

在 Amazon Aurora 中複製資料庫參數群組

您可複製所建立的自訂資料庫參數群組。複製參數群組可以是便利的解決方案。範例是在您已建立資料庫參數群組，並希望將其大部份的自訂參數及值併入新的資料庫參數群組時。您可以使用 複製資料庫參數群組 AWS 管理主控台。您也可以使用 AWS CLI copy-db-parameter-group 命令或 RDS API CopyDBParameterGroup 操作。

複製資料庫參數群組後，請等待至少 5 分鐘，然後再建立第一個使用該資料庫參數群組做為預設參數群組的資料庫執行個體。執行此動作允許 Amazon RDS 在使用參數群組前，完整完成複製動作。這對

建立資料庫執行個體預設資料庫時的關鍵參數尤其重要。例如 `character_set_database` 參數定義的預設資料庫字元集。請使用 Amazon RDS 主控台的 Parameter Groups (參數群組) 選項或 describe-db-parameters 命令確認是否已建立資料庫參數群組。

> **Note**
>
> 您無法複製預設參數群組。但您可以依照預設參數群組建立新的參數群組。
> 您無法將資料庫參數群組複製到不同的 AWS 帳戶 或 AWS 區域。

主控台

複製資料庫參數群組

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。
3. 在清單中，選擇您要複製的自訂參數群組。
4. 針對 Parameter group actions (參數群組動作)，選擇 Copy (複製)。
5. 在 New DB parameter group identifier (新資料庫參數群組識別符) 中，輸入新參數群組的名稱。
6. 在 Description (描述) 中，輸入新參數群組的描述。
7. 請選擇 Copy (複製)。

AWS CLI

若要複製資料庫參數群組，請使用 AWS CLI `copy-db-parameter-group` 命令搭配下列必要選項：

- `--source-db-parameter-group-identifier`
- `--target-db-parameter-group-identifier`
- `--target-db-parameter-group-description`

以下範例會建立名為 `mygroup2` 的新資料庫參數群組，該群組為資料庫參數群組 `mygroup1` 的複本。

Example

針對 Linux、macOS 或 Unix：

```
aws rds copy-db-parameter-group \
    --source-db-parameter-group-identifier mygroup1 \
    --target-db-parameter-group-identifier mygroup2 \
    --target-db-parameter-group-description "DB parameter group 2"
```

在 Windows 中：

```
aws rds copy-db-parameter-group ^
    --source-db-parameter-group-identifier mygroup1 ^
    --target-db-parameter-group-identifier mygroup2 ^
    --target-db-parameter-group-description "DB parameter group 2"
```

RDS API

若要複製資料庫參數群組，請使用 RDS API CopyDBParameterGroup 參數並搭配下列必要參數：

- SourceDBParameterGroupIdentifier
- TargetDBParameterGroupIdentifier
- TargetDBParameterGroupDescription

列出 Amazon Aurora 中的資料庫參數群組

您可以列出您為 AWS 帳戶建立的資料庫參數群組。

> **Note**
>
> 當您為特定資料庫引擎和版本建立資料庫執行個體時，會從預設的參數範本自動建立預設的參數群組。這些預設參數群組包含偏好的參數設定，無法修改。當您建立自訂參數群組時，您可以修改參數設定。

主控台

列出 AWS 帳戶的所有資料庫參數群組

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。

資料庫參數群組隨即會出現在清單中。

AWS CLI

若要列出 AWS 帳戶的所有資料庫參數群組，請使用 AWS CLI `describe-db-parameter-groups` 命令。

Example

以下範例列出 AWS 帳戶的所有可用資料庫參數群組。

```
aws rds describe-db-parameter-groups
```

此命令會傳回類似以下的回應：

```
DBPARAMETERGROUP  default.mysql8.0     mysql8.0  Default parameter group for MySQL8.0
DBPARAMETERGROUP  mydbparametergroup   mysql8.0  My new parameter group
```

以下範例說明 mydbparamgroup1 參數群組。

針對 Linux、macOS 或 Unix：

```
aws rds describe-db-parameter-groups \
    --db-parameter-group-name mydbparamgroup1
```

在 Windows 中：

```
aws rds describe-db-parameter-groups ^
    --db-parameter-group-name mydbparamgroup1
```

此命令會傳回類似以下的回應：

```
DBPARAMETERGROUP  mydbparametergroup1  mysql8.0  My new parameter group
```

RDS API

若要列出 AWS 帳戶的所有資料庫參數群組，請使用 RDS API `DescribeDBParameterGroups`操作。

在 Amazon Aurora 中檢視資料庫參數群組的參數值

您可以從資料庫參數群組取得所有參數與其值的清單。

主控台

檢視資料庫參數群組的參數值

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。

   資料庫參數群組隨即會出現在清單中。
3. 選擇參數群組的名稱，以查看其參數清單。

AWS CLI

若要檢視資料庫參數群組的參數值，請使用 AWS CLI `describe-db-parameters` 命令搭配下列必要參數。

- `--db-parameter-group-name`

Example

以下範例列出資料庫參數群組 mydbparametergroup 的參數和參數值。

```
aws rds describe-db-parameters --db-parameter-group-name mydbparametergroup
```

此命令會傳回類似以下的回應：

```
DBPARAMETER  Parameter Name           Parameter Value  Source           Data Type
 Apply Type  Is Modifiable
DBPARAMETER  allow-suspicious-udfs                    engine-default   boolean
 static      false
DBPARAMETER  auto_increment_increment                 engine-default   integer
 dynamic     true
DBPARAMETER  auto_increment_offset                    engine-default   integer
 dynamic     true
DBPARAMETER  binlog_cache_size        32768           system           integer
 dynamic     true
```

```
DBPARAMETER  socket                    /tmp/mysql.sock  system          string
 static      false
```

RDS API

若要檢視資料庫參數群組的參數值，請搭配下列必要參數使用 RDS API `DescribeDBParameters` 命令。

- `DBParameterGroupName`

在 Amazon Aurora 中刪除資料庫參數群組

您可以使用 AWS 管理主控台 AWS CLI或 RDS API 刪除資料庫參數群組。參數群組只有在未與資料庫執行個體相關聯時，才符合刪除條件。

主控台

刪除資料庫參數群組

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。

   資料庫參數群組隨即會出現在清單中。
3. 選擇要刪除的參數群組的名稱。
4. 選擇動作，然後選擇刪除。
5. 檢閱參數群組名稱，然後選擇刪除。

AWS CLI

若要刪除資料庫參數群組，請使用 AWS CLI `delete-db-parameter-group` 命令搭配下列必要參數。

- `--db-parameter-group-name`

Example

下列範例會刪除名為 mydbparametergroup 的資料庫參數群組。

```
aws rds delete-db-parameter-group --db-parameter-group-name mydbparametergroup
```

RDS API

若要刪除資料庫參數群組，請使用 RDS API DeleteDBParameterGroup 命令搭配下列必要參數。

- DBParameterGroupName

比較資料庫參數群組

您可以使用 AWS 管理主控台 檢視兩個資料庫參數群組之間的差異。

指定的參數群組必須都是資料庫參數群組，或兩者都必須是資料庫叢集參數群組。當資料庫引擎和版本相同，也是如此。例如，您無法比較 aurora-mysql8.0 (Aurora MySQL 第 3 版) 資料庫參數群組與 aurora-mysql8.0 資料庫叢集參數群組。

您可以比較 Aurora MySQL 與 RDS for MySQL 資料庫參數群組，即使是不同的版本也可以，但無法比較 Aurora PostgreSQL 與 RDS for PostgreSQL 資料庫參數群組。

比較兩個資料庫參數群組

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。
3. 在清單中，選擇您要比較的兩個參數群組。

   > **Note**
   >
   > 若要比較預設參數群組與自訂參數群組，請先在預設索引標籤上選擇預設參數群組，然後在自訂索引標籤上選擇自訂參數群組。

4. 在動作中，選擇比較。

指定資料庫參數

資料庫參數類型包含下列各項：

- Integer

- 布林值
- 字串
- Long
- Double
- 時間戳記
- 其他定義資料類型的物件
- integer、Boolean、string、long、double、timestamp 或 object 參數類型的值陣列

您還可以使用運算式、公式和函數來指定整數及布林值參數。

內容

- 資料庫參數公式
  - 資料庫參數公式變數
  - 資料庫參數公式運算子
- 資料庫參數函數
- 資料庫參數對數運算式
- 資料庫參數值範例

資料庫參數公式

資料庫參數公式是可以解析成整數值或布林值的運算式。以括號 {} 括住運算式。您可以針對資料庫參數值使用公式，或將公式指定為資料庫參數函數的引數。

語法

```
{FormulaVariable}
{FormulaVariable*Integer}
{FormulaVariable*Integer/Integer}
{FormulaVariable/Integer}
```

資料庫參數公式變數

每個公式變數都會傳回整數或布林值。變數的名稱區分大小寫。

AllocatedStorage

返回表示資料磁碟區大小的整數 (字元組)。

DBInstanceClassMemory

傳回一個整數，表示資料庫程序可用的記憶體位元組數。此數字是在內部從資料庫執行個體類別的總記憶體量開始計算得出的。其中，計算會減去為作業系統和管理執行個體的 RDS 程序保留的記憶體。因此，此數字永遠會低於 Amazon Aurora 資料庫執行個體類別 上執行個體類別表中顯示的記憶體容量。確切值取決於幾個綜合因素。其中包括執行個體類別、資料庫引擎，以及要套用至 RDS 執行個體還是屬於 Aurora 叢集的執行個體。

DBInstanceVCPU

傳回表示 Amazon RDS 管理執行個體所用虛擬中央處理單元 (vCPU) 數目的整數。

EndPointPort

返回表示連線至資料庫執行個體時所用連接埠的整數。

TrueIfReplica

如果資料庫執行個體為僅供讀取複本，則傳回 `1`，若不是，則傳回 `0`。這是 Aurora MySQL 中 `read_only` 參數的預設值。

資料庫參數公式運算子

資料庫參數公式支援兩種運算子：除法和乘法。

除法運算子：/

用除數除以被除數，並傳回整數之商。商中的小數不進位，直接截斷。

語法

```
dividend / divisor
```

被除數和除數引數必須為整數運算式。

乘法運算子：*

乘以運算式，並傳回運算式的乘積。運算式中的小數不進位，直接截斷。

語法

```
expression * expression
```

兩個運算式都必須為整數。

資料庫參數函數

您可以將資料庫參數函數的引數指定為整數或公式。每個函數至少必須有一個引數。以逗號分隔的清單指定多個引數。清單不能有任何空成員，例如 argument1、argument3。函數名稱區分大小寫。

IF

傳回引數。

語法

```
IF(argument1, argument2, argument3)
```

若第一個引數計算結果為 true，傳回第二個引數。否則，傳回第三個引數。

GREATEST

從整數或參數公式清單中傳回最大值。

語法

```
GREATEST(argument1, argument2,...argumentn)
```

傳回整數。

LEAST

從整數或參數公式清單中傳回最小值。

語法

```
LEAST(argument1, argument2,...argumentn)
```

傳回整數。

SUM

新增指定整數或參數公式的值。

語法

```
SUM(argument1, argument2,...argumentn)
```

傳回整數。

資料庫參數對數運算式

您可以將整數資料庫參數值設定為對數運算式。以括號 {} 括住運算式。例如：

```
{log(DBInstanceClassMemory/8187281418)*1000}
```

log 函數表示對數底數 2。此範例還會使用 DBInstanceClassMemory 公式變數。請參閱資料庫參數公式變數。

資料庫參數值範例

這些範例顯示使用資料庫參數值的公式、函數和運算式。

> **Warning**
>
> 未正確設定資料庫參數群組中的參數，可能產生各種意外影響。這些影響可能包括降低效能和系統不穩定。修改資料庫參數時請謹慎，在修改資料庫參數群組之前，請備份您的資料。請在測試資料庫執行個體上嘗試進行參數群組變更，再將參數群組變更套用至生產資料庫執行個體。

Example使用資料庫參數函數 LEAST

您可以在 Aurora MySQL LEAST 參數值中指定 table_definition_cache 函數。使用該函數將可存放在定義快取中的資料表定義數量設為 DBInstanceClassMemory/393040 或 20,000 (取較小者)。

```
LEAST({DBInstanceClassMemory/393040}, 20000)
```

將資料遷移至 Amazon Aurora 資料庫叢集

若要從現有的資料庫將資料遷移至 Amazon Aurora 資料庫叢集，視資料庫引擎相容性而定，您有幾個選項。遷移選項還取決於要遷移的資料庫以及要遷移的資料大小。

將資料遷移至 Amazon Aurora MySQL 資料庫叢集

您可以從下列其中一個來源，將資料遷移至 Amazon Aurora MySQL 資料庫叢集。

- RDS for MySQL 資料庫執行個體
- Amazon RDS 外部的 MySQL 資料庫
- 與 MySQL 不相容的資料庫

如需更多詳細資訊，請參閱 將資料遷移至 Amazon Aurora MySQL 資料庫叢集。

將資料遷移至 Amazon Aurora PostgreSQL 資料庫叢集

您可以從下列其中一個來源，將資料遷移至 Amazon Aurora PostgreSQL 資料庫叢集。

- Amazon RDS PostgreSQL 資料庫執行個體
- 與 PostgreSQL 不相容的資料庫

如需更多詳細資訊，請參閱 將資料遷移至與 PostgreSQL 相容的 Amazon Aurora。

使用 Aurora 資料庫叢集設定來建立 Amazon ElastiCache 快取

ElastiCache 是全受管記憶體內快取服務，其會提供可支援彈性即時使用案例的微秒讀取和寫入延遲。ElastiCache 可協助您加速提高應用程式和資料庫效能。您可以將 ElastiCache 用作不需要資料持久性之使用案例的主要資料存放區，例如遊戲排行榜、串流和資料分析。ElastiCache 有助於消除與部署和管理分散式快取環境相關聯的複雜性。如需詳細資訊，請參閱常見的 ElastiCache 使用案例及 ElastiCache 如何提供協助 (若為 Memcached) 和常見的 ElastiCache 使用案例及 ElastiCache 如何提供協助 (若為 Redis OSS)。您可以使用 Amazon RDS 主控台建立 ElastiCache 快取。

您可以操作兩種格式的 Amazon ElastiCache。您可以開始使用無伺服器快取，或選擇自行設計快取叢集。如果您選擇設計自己的快取叢集，ElastiCache 可同時與 Redis OSS 和 Memcached 引擎搭配使用。如果您不確定要使用哪個引擎，請參閱比較 Memcached 和 Redis OSS。如需 Amazon ElastiCache 的詳細資訊，請參閱 Amazon ElastiCache 使用者指南。

主題

- 使用 Aurora 資料庫叢集設定建立 ElastiCache 快取的概觀
- 使用來自 Aurora 資料庫叢集的設定建立 ElastiCache 快取

使用 Aurora 資料庫叢集設定建立 ElastiCache 快取的概觀

您可以使用與新建立的或現有的 Aurora 資料庫叢集相同的組態設定，從 Amazon RDS 建立 ElastiCache 快取。

將 ElastiCache 快取與資料庫叢集建立關聯的若干使用案例：

- 您可以將 ElastiCache 與 RDS 搭配使用 (而不是單獨在 RDS 上執行) 以節省成本並改善效能。
- 您可以使用 ElastiCache 快取作為不需要資料持久性之應用程式的主要資料存放區。使用 Redis OSS 或 Memcached 的應用程式可以使用 ElastiCache，幾乎不需要任何修改。

當您從 RDS 建立 ElastiCache 快取時，ElastiCache 快取會從相關聯的 Aurora 資料庫叢集繼承下列設定：

- ElastiCache 連線設定
- ElastiCache 安全設定

您也可以根據需求設定快取組態設定。

在您的應用程式中設定 ElastiCache

應用程式必須設定為使用 ElastiCache 快取。您也可以最佳化和改善快取效能，方法為將您的應用程式設定為使用快取策略，取決於您的需求。

- 若要存取 ElastiCache 快取並開始使用，請參閱開始使用 ElastiCache (Redis OSS) 和開始使用 ElastiCache (Memcached)。
- 如需快取策略的詳細資訊，請參閱快取策略與最佳實務 (若為 Memcached) 和快取策略與最佳實務 (若為 Redis OSS)。
- 如需 ElastiCache (Redis OSS) 叢集中高可用性的詳細資訊，請參閱使用複寫群組的高可用性。
- 您可能會產生與備份儲存、區域內或跨區域資料傳輸或使用 相關的成本 AWS Outposts。如需定價詳細資訊，請參閱 Amazon ElastiCache 定價。

使用來自 Aurora 資料庫叢集的設定建立 ElastiCache 快取

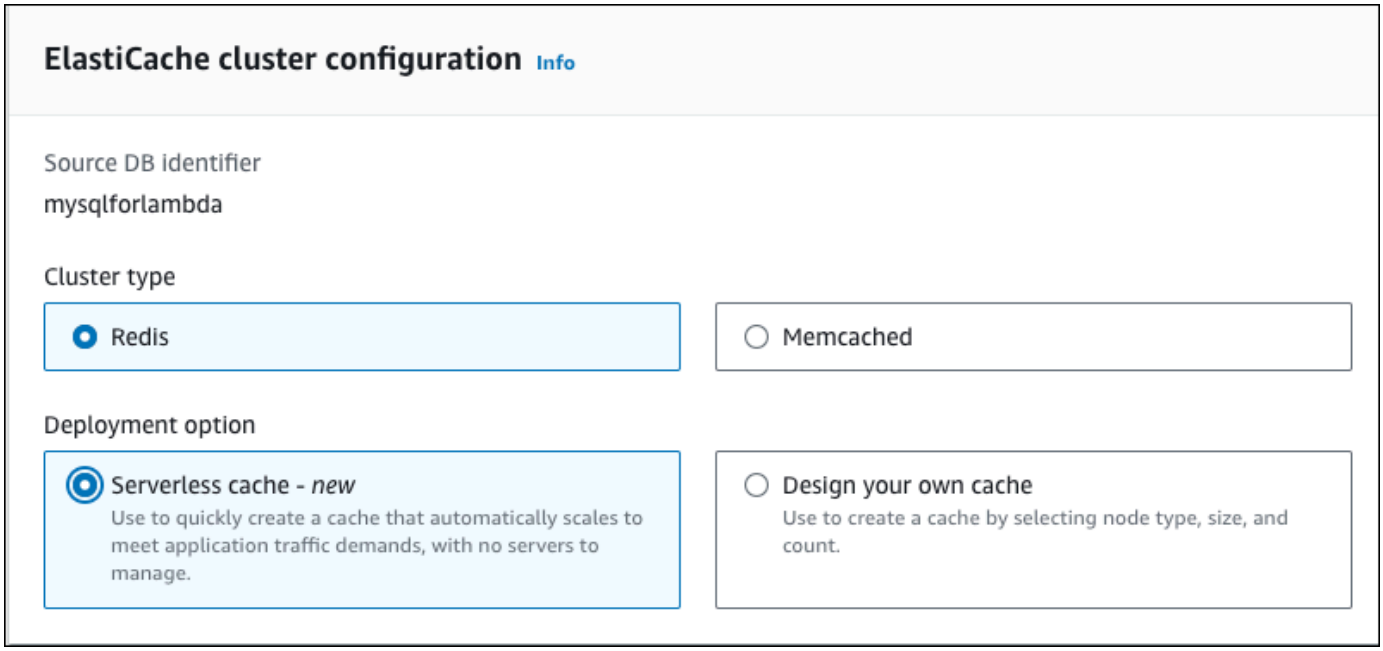
您可以使用繼承自資料庫叢集的設定，為 Aurora 資料庫叢集建立 ElastiCache 快取。

使用來自資料庫叢集的設定建立 ElastiCache 快取

1. 若要建立資料庫叢集，請遵循 建立 Amazon Aurora 資料庫叢集 中的指示。
2. 在建立 Aurora 資料庫叢集之後，主控台會顯示建議的附加元件視窗。選取使用您的資料庫設定從 RDS 建立 ElastiCache 叢集。

   針對現有資料庫，在資料庫頁面中，選取所需的資料庫叢集。在動作下拉式功能表中，選擇建立 ElastiCache 叢集，以在 RDS 中建立與現有的 Aurora 資料庫叢集具有相同設定的 ElastiCache 快取。

   在 ElastiCache 組態區段中，來源資料庫識別符會顯示 ElastiCache 快取從哪個資料庫叢集繼承設定。
3. 選擇要建立 Redis OSS 還是 Memcached 叢集。如需詳細資訊，請參閱比較 Memcached 與 Redis OSS。



4. 之後，選擇您要建立無伺服器快取還是設計您自己的快取。如需詳細資訊，請參閱選擇部署選項。

   如果您選擇無伺服器快取：

   a. 在快取設定中，輸入名稱和描述的值。

   b. 在檢視預設設定下，保留預設設定，以在快取和資料庫叢集之間建立連線。

   c. 您也可以選擇自訂預設設定來編輯預設設定。選取 ElastiCache 連線設定、ElastiCache 安全設定和用量上限。

5. 如果您選擇設計您自己的快取：

   a. 如果您選擇 Redis OSS 叢集，請選擇要將叢集模式保留為已啟用還是已停用。如需詳細資訊，請參閱複寫：Redis OSS (叢集模式已停用) 與 Redis OSS (叢集模式已啟用)。

   b. 輸入名稱、描述和引擎版本的值。

      對於引擎版本，建議的預設值是最新的引擎版本。您也可以為 ElastiCache 快取選擇最符合您需求的引擎版本。

   c. 在節點類型選項中選擇節點類型。如需詳細資訊，請參閱管理節點。

      如果您選擇在叢集模式設定為已啟用的情況下建立 Redis OSS 叢集，請在碎片數目選項中輸入碎片數目 (分割區/節點群組)。

      在複本數目中輸入每個碎片的複本數目。

> **Note**
>
> 選取的節點類型、碎片數目，以及複本數目都會影響您的快取效能和資源成本。確定這些設定符合您的資料庫需求。如需定價資訊，請參閱 Amazon ElastiCache 定價。

d. 選取 ElastiCache 連線設定和 ElastiCache 安全設定。您可以保留預設設定，或根據需求自訂這些設定。

6. 驗證 ElastiCache 快取的預設和繼承設定。部分設定在建立後無法變更。

> **Note**
>
> RDS 可能會調整 ElastiCache 快取的備份時段，以符合 60 分鐘的最低時段需求。來源資料庫的備份視窗會保持不變。

7. 當您準備好時，請選擇建立 ElastiCache 快取。

主控台會顯示用於建立 ElastiCache 快取的確認橫幅。遵循橫幅中的連結前往 ElastiCache 主控台，以檢視快取詳細資訊。ElastiCache 主控台會顯示新建立的 ElastiCache 快取。

使用 自動將 EC2 資料庫遷移至 Amazon Aurora AWS Database Migration Service

您可以使用 Aurora 主控台將 EC2 資料庫遷移至 Aurora。Aurora RDS使用 AWS Database Migration Service (AWS DMS) 遷移來源 EC2 資料庫。 AWS DMS 可讓您將關聯式資料庫遷移至 AWS 雲端。如需詳細資訊 AWS Database Migration Service，請參閱AWS Database Migration Service 《 使用者指南》中的什麼是 AWS Database Migration Service？。

若要開始遷移，您必須建立同等的 Aurora 資料庫叢集 ，以將資料遷移至其中。建立目標資料庫之後，您可以將 EC2 資料庫匯入其中。對於小於 1TiB 的來源資料庫，此遷移動作可減少將資料遷移至 Aurora 所需的時間和資源。

概觀

Aurora 主控台可讓您將 EC2 資料庫遷移至同等的 Aurora 資料庫。您必須建立 Aurora 資料庫，才能從主控台啟用遷移。

您可以遷移下列資料庫引擎的 EC2 資料庫：

- MySQL
- PostgreSQL

此遷移程序涉及下列步驟：

- 在 Aurora 中建立同等資料庫。若要讓資料庫成為相等，它們必須具有相同的資料庫引擎和相容的引擎版本。它們也必須位於相同的 VPC 中。如需建立資料庫的指示，請參閱 建立 Amazon Aurora 資料庫叢集 。
- 選擇資料庫的複寫類型：
  - 完整載入遷移 – Aurora 會將完整的來源資料庫複製到目標資料庫，必要時在目標中建立新的資料表。

    > **Note**
    >
    > 此選項會導致 Aurora 資料庫中斷。

  - 完整載入和變更資料擷取 (CDC) 遷移 – 類似於完整載入遷移，使用此選項，Aurora 會將完整來源資料庫複製到目標資料庫。不過，在完整載入遷移之後，Aurora 會將來源中擷取的任何變更套用至目標資料庫。變更資料擷取會使用資料庫引擎的原生 API 收集資料庫日誌的變更。

> **Note**
>
> 此選項會導致 Aurora 資料庫中斷。

- 變更資料擷取 (CDC) – 使用此選項，透過遷移保持您的目標資料庫可用。Aurora 會將來源資料庫中的持續變更遷移至目標資料庫。

- Aurora 會建立必要的聯網資源，以促進遷移。Aurora 建立所需的資源後，它會通知您已建立的資源，並允許您啟動資料傳輸。

  完成遷移所需的時間取決於複寫類型和來源資料庫的大小。

先決條件

MySQL

開始使用 MySQL 資料庫做為來源資料庫之前，請務必做好以下事前準備。這些先決條件適用於 AWS 受管來源。

您必須擁有具有複寫管理員角色 AWS DMS 的 帳戶。此角色需要以下權限：

- REPLICATION CLIENT：僅 CDC 任務需要此權限。換言之，完全載入任務不需要此權限。
- REPLICATION SLAVE：僅 CDC 任務需要此權限。換言之，完全載入任務不需要此權限。

AWS DMS 使用者也必須具有指定用於複寫之來源資料表的 SELECT 權限。

如果您使用 MySQL 特定的預遷移評估，請授予下列權限。

```
grant select on mysql.user to <dms_user>;
grant select on mysql.db to <dms_user>;
grant select on mysql.tables_priv to <dms_user>;
grant select on mysql.role_edges to <dms_user>  #only for MySQL version 8.0.11 and
 higher
```

PostgreSQL

從 AWS受管 PostgreSQL 來源資料庫遷移資料之前，請執行下列動作：

- 我們建議您使用具有 PostgreSQL 資料庫執行個體最低必要許可的 AWS 使用者帳戶，做為 PostgreSQL 來源端點的使用者帳戶 AWS DMS。不建議使用主帳戶。此帳戶必須具有 `rds_superuser` 角色和 `rds_replication` 角色。授權來管理邏輯槽和利用邏輯槽來串流資料的 `rds_replication` 角色

> **Note**
>
> 有些 AWS DMS 交易會在 DMS 引擎再次使用它們之前閒置一段時間。透過使用 PostgreSQL 9.6 版和更高版本的參數 `idle_in_transaction_session_timeout`，可讓您將閒置交易變成逾時而失敗。

限制

下列限制適用於自動遷移程序：

- 您的目標資料庫狀態必須為可用，才能開始來源資料庫遷移。
- 從 MySQL 來源資料庫遷移時，您的 Aurora 帳戶必須具有複寫管理員角色。您還必須針對該角色套用適當的權限。
- 您的 EC2 執行個體和目標資料庫必須位於相同的 VPC 中。
- 使用從 EC2 資料庫遷移資料動作時，您無法將 EC2 資料庫遷移至下列目標資料庫：
  - Aurora global database
  - Aurora Limitless database
  - Aurora Serverless v1
  - MySQL 版本低於 5.7 的資料庫
  - PostgreSQL 版本低於 10.4 的資料庫

為同質遷移建立 IAM 資源

Aurora RDS使用 AWS DMS遷移您的資料。若要存取資料庫和遷移資料， 會為同質資料遷移AWS DMS建立無伺服器環境。在此環境中， AWS DMS需要存取 VPC 對等互連、路由表、安全群組和其他AWS資源。此外， 會在 Amazon CloudWatch 中AWS DMS儲存每個資料遷移的日誌、指標和進度。若要建立資料遷移專案， AWS DMS需要存取這些服務。

此外， AWS DMS需要存取代表一組使用者登入資料的秘密，以驗證來源和目標連線的資料庫連線。

> **Note**
>
> 透過使用從 EC2 執行個體遷移資料動作，您可以使用 Aurora 主控台來產生這些 IAM 資源。如果您使用主控台產生的 IAM 資源，請略過此步驟。

對於此程序，您需要下列 IAM 資源：

主題

- 建立同質資料遷移的 IAM 政策
- 為同質資料遷移建立 IAM 角色
- 建立秘密存取政策和角色
- 建立 AWS DMS 的 IAM 角色以管理 Amazon VPC

建立同質資料遷移的 IAM 政策

在此步驟中，您會建立 IAM 政策，AWS DMS提供 Amazon EC2 和 CloudWatch 資源的存取權。接下來，請建立 IAM 角色並連接政策。

若要為資料遷移建立 IAM 政策

1. 登入 AWS 管理主控台並開啟位於 https：//https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在導覽窗格中，選擇政策。
3. 選擇 Create policy (建立政策)。
4. 在建立政策頁面中，選擇 JSON 索引標籤。
5. 將以下 JSON 貼至編輯器中。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeRouteTables",
```

```
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeVpcPeeringConnections",
                "ec2:DescribeVpcs",
                "ec2:DescribePrefixLists",
                "logs:DescribeLogGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "servicequotas:GetServiceQuota"
            ],
            "Resource": "arn:aws:servicequotas:*:*:vpc/L-0EA8095F"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:dms-data-migration-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:dms-data-migration-*:log-
stream:dms-data-migration-*"
        },
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateRoute",
                "ec2:DeleteRoute"
            ],
            "Resource": "arn:aws:ec2:*:*:route-table/*"
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:security-group-rule/*",
                "arn:aws:ec2:*:*:route-table/*",
                "arn:aws:ec2:*:*:vpc-peering-connection/*",
                "arn:aws:ec2:*:*:vpc/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress"
            ],
            "Resource": "arn:aws:ec2:*:*:security-group-rule/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:RevokeSecurityGroupIngress"
            ],
            "Resource": "arn:aws:ec2:*:*:security-group/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AcceptVpcPeeringConnection",
                "ec2:ModifyVpcPeeringConnectionOptions"
            ],
            "Resource": "arn:aws:ec2:*:*:vpc-peering-connection/*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:AcceptVpcPeeringConnection",
            "Resource": "arn:aws:ec2:*:*:vpc/*"
```

```
        }
    ]
}
```

6. 選擇 Next: Tags (下一步：標籤) 和 Next: Review (下一步：檢閱)。
7. 在名稱*中輸入 **HomogeneousDataMigrationsPolicy**，然後選擇建立政策。

為同質資料遷移建立 IAM 角色

在此步驟中，您會建立提供存取 AWS Secrets Manager、Amazon EC2 和 CloudWatch 的 IAM 角色。

為資料遷移建立 IAM 角色

1. 登入 AWS 管理主控台並開啟位於 https：//https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在導覽窗格中，選擇 Roles (角色)。
3. 選擇 Create Role (建立角色)。
4. 在選取受信任實體頁面上，針對信任的實體類型選擇 AWS 服務。在其他 AWS 服務的使用案例中，選擇 DMS
5. 選取 DMS 核取方塊，然後選擇下一步。
6. 在新增許可頁面上，選擇您之前建立的 HomogeneousDataMigrationsPolicy 。選擇下一步。
7. 在命名、檢閱和建立頁面上的角色名稱輸入 **HomogeneousDataMigrationsRole**，然後選擇建立角色。
8. 在角色頁面，針對角色名稱輸入 **HomogeneousDataMigrationsRole**。選擇 HomogeneousDataMigrationsRole。
9. 在 HomogeneousDataMigrationsRole 頁面上，選擇信任關係標籤。選擇編輯信任政策。
10. 在編輯信任原則頁面上，將下列 JSON 貼到編輯器中以取代現有的文字。

    JSON

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

```
                "Service": [
                    "dms-data-migrations.amazonaws.com",
                    "dms.your_region.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

在上述範例中，*# your_region* 取代為 的名稱AWS 區域。

上述以資源為基礎的政策為AWS DMS服務主體提供根據客戶管理的 HomogeneousDataMigrationsPolicy 政策執行任務的許可。

11. 選擇更新政策。

建立秘密存取政策和角色

請依照下列程序建立秘密存取政策和角色，允許 DMS 存取來源和目標資料庫的使用者憑證。

建立秘密存取政策和角色，以允許 Amazon RDS 存取 AWS Secrets Manager 您的適當秘密

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/iam/ 的 AWS Identity and Access Management (IAM) 主控台。
2. 選擇政策，然後選擇建立政策。
3. 選擇 JSON 並輸入以下政策以啟用機密的存取和解密。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:SecretName-ABCDEF"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ]
}
```

在這裡， *secret_arn* 是您秘密的 ARN，您可以SecretsManagerSecretId視需要從 取得，而 *kms_key_arn*是您用來加密秘密之 AWS KMS 金鑰的 ARN，如下列範例所示。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:us-
east-2:123456789012:secret:MySQLTestSecret-qeHamH"
        },
        {
             "Effect": "Allow",
             "Action": [
                        "kms:Decrypt",
                        "kms:DescribeKey"
                      ],
             "Resource": "arn:aws:kms:us-
east-2:123456789012:key/761138dc-0542-4e58-947f-4a3a8458d0fd"
        }
     ]
}
```

> **Note**
>
> 如果您使用 建立的預設加密金鑰 AWS Secrets Manager，則不需要指定 的 AWS KMS 許可*kms_key_arn*。
> 如果您希望政策提供對這兩個機密的存取權，只需為另一個 *secret_arn* 指定額外的 JSON 資源對象即可。

4. 檢閱並使用易記名稱建立政策，並視需要提供說明。
5. 選擇角色，然後選擇建立角色。
6. 選擇 AWS 服務作為信任的實體類型。
7. 從服務清單中選擇 DMS 作為信任的服務，然後選擇下一步：許可。
8. 查詢並附加您在步驟 4 中建立的政策，然後繼續新增任何標籤並檢閱您的角色。此時，請編輯角色的信任關係，以使用您的 Amazon RDS 區域服務主體作為信任的實體。此主體的格式如下。

```
dms.region-name.amazonaws.com
```

其中，*region-name* 是區域的名稱，例如 `us-east-1`。因此，此區域的 Amazon RDS 區域服務主體如下。

```
dms.us-east-1.amazonaws.com
dms-data-migrations.amazonaws.com
```

建立 AWS DMS 的 IAM 角色以管理 Amazon VPC

您必須建立 AWS DMS 的 IAM 角色，才能管理資源的 VPC 設定。此角色必須可用於成功遷移。

建立資料庫遷移的 **dms-vpc-role**

<result>

這會建立 DMS 的角色，以管理遷移的 VPC 設定。
</result>

1. 簽署 AWS 管理主控台，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在主控台的導覽窗格中，選擇 Roles (角色)，然後選擇 Create role (建立角色)。
3. 選擇選取信任的實體選項的 AWS 服務 選項。

對於使用案例，選取 DMS。

4. 對於新增許可步驟，選取 AmazonDMSVPCManagementRole，然後選擇下一步。
5. 在命名、檢閱和建立頁面中，將角色名稱設定為 dms-vpc-role，然後選擇建立角色。

設定 EC2 資料庫的資料遷移

若要開始從 EC2 來源資料庫遷移資料，您必須建立同等的 Aurora 資料庫。如需建立資料庫的指示，請參閱 建立 Amazon Aurora 資料庫叢集 。

建立目標資料庫之後，請使用下列步驟來設定資料遷移：

設定資料遷移專案

1. 在 RDS 主控台的資料庫頁面上選取目標資料庫。
2. 選擇動作下拉式清單，然後選取從 EC2 資料庫遷移資料選項。若要查看支援的目標資料庫，請參閱 限制。
3. 在選取來源 EC2 資料庫區段下：

    1. 檢查引擎類型，並確認其與您的來源資料庫相同。

        此外，請檢查引擎版本是否相容。
    2. 對於 EC2 執行個體，選擇來源資料庫所在的 EC2 執行個體。
    3. 對於連接埠，輸入來源資料庫允許流量的連接埠。
    4. 對於秘密，如果您沒有現有的秘密，請選擇建立並使用新的秘密。輸入來源資料庫的使用者名稱和密碼。同時請選擇要用來加密秘密的 KMS 金鑰。

        如果您有現有的秘密，請選取使用現有的秘密，然後從下拉式清單中選擇秘密。
    5. 對於秘密的 IAM 角色，如果您有現有的 IAM 角色，請選取使用現有的 IAM 角色，然後從下拉式清單中選擇可存取上一個步驟中秘密 ID 的 IAM 角色。

        如果您沒有現有的 IAM 角色，請選擇 建立並使用新的 IAM 角色。在 IAM 角色名稱中輸入角色的新名稱。您可以在以下連結中查看與此角色相關聯的許可。
4. 在檢視目標 RDS 資料庫區段下：

    1. 在區段頂端確認目標資料庫的設定。
    2. 對於秘密，如果您沒有保存目標資料庫憑證的現有秘密，請選擇建立並使用新的秘密。

如果您有現有的秘密，請從下拉式清單中選取秘密。

3. 對於秘密的 IAM 角色，選取可從上一個步驟存取秘密的 IAM 角色。如果您沒有現有的 IAM 角色，也可以建立新的 IAM 角色。

   如果下拉式清單未填入 IAM 角色，請以格式 `arn:aws:iam:account_id:role/roleName` 指定 IAM 角色 ARN。

5. 在設定資料遷移區段下：

   1. 選取完整載入、完整載入和變更資料擷取 (CDC) 或變更資料擷取 (CDC)，以選取資料遷移的類型。如需關於這些選項的詳細資訊，請參閱 概觀。

      您無法在遷移開始後修改遷移類型。

   2. 對於資料遷移的 IAM 角色，如果您有現有的 IAM 角色，請選取使用現有的 IAM 角色，然後從下拉式清單中選擇 IAM 角色，授予 DMS 建立遷移所需資源的許可。如果您沒有現有的 IAM 角色，請選擇 建立並使用新的 IAM 角色。

6. 確認檢視遷移設定索引標籤顯示成功設定資料遷移所需的設定。
7. 選取遷移以完成遷移設定。

完成這些步驟後，您可以在主控台的進度橫幅中選擇檢視詳細資訊，以查看為資料遷移設定的資源。設定必要的資源後，遷移會自動開始。如果您建立

若要將多個資料庫遷移至目標資料庫，請使用新 EC2 資料庫的詳細資訊再次啟動此程序。

管理資料遷移

從 RDS 主控台使用從 EC2 資料庫遷移資料動作之後，Aurora 會自動開始遷移。

如果您使用 AWS DMS 主控台建立遷移資源，您可以開始遷移程序。

開始資料遷移

請依照下列步驟開始資料遷移：

開始資料遷移

1. 在 RDS 主控台的資料庫頁面上選擇目標資料庫。
2. 在資料庫詳細資訊頁面上，選擇資料遷移索引標籤。
3. 在資料遷移索引標籤下，相關聯資料遷移會列出可用的資料遷移。

設定所需的資源後，使用 Aurora 主控台設定的遷移會自動開始。

使用 DMS 主控台設定的遷移會設定為就緒。

若要開始這些遷移，請選取動作下拉式清單，然後選取開始。

4. 這會開始 EC2 資料庫的資料遷移。

停止資料遷移

對於複寫類型為完整載入的資料遷移，停止遷移會導致程序停止且無法繼續。停止後，您必須重新啟動遷移。

對於將複寫類型設定為變更資料擷取 (CDC) 或完整載入和 CDC 的遷移，您可以停止連續複寫程序，並在稍後繼續程序。

停止資料遷移

1. 在 RDS 主控台的資料庫頁面上選擇目標資料庫。
2. 在資料庫詳細資訊頁面上，選擇資料遷移索引標籤。
3. 在資料遷移索引標籤下，相關聯資料遷移會列出進行中的資料遷移。

   若要停止遷移，請選取資料遷移，然後在動作下拉式清單中選取停止。
4. 這會停止 EC2 資料庫的資料遷移。

繼續資料遷移

對於複寫類型為完整載入和變更資料擷取 (CDC) 或變更資料擷取 (CDC) 遷移的資料遷移，您可以從最後一個停止點繼續 CDC 程序。

繼續資料遷移

1. 在 RDS 主控台的資料庫頁面上選擇目標資料庫。
2. 在資料庫詳細資訊頁面上，選擇資料遷移索引標籤。
3. 在資料遷移索引標籤下，相關聯資料遷移會列出已停止的資料遷移。

   若要繼續遷移，請選取資料遷移，然後在動作下拉式清單中選取繼續處理。
4. 這會繼續 EC2 資料庫的資料遷移。

刪除資料遷移

若要刪除相關聯的資料遷移，請使用下列指示

刪除資料遷移

1. 在 RDS 主控台的資料庫頁面上選擇目標資料庫。
2. 在資料庫詳細資訊頁面上，選擇資料遷移索引標籤。
3. 若要刪除遷移，請選取資料遷移，然後在動作下拉式清單中選取刪除。
4. 這會刪除資料遷移。

刪除進行中的資料遷移不會影響任何已載入目標資料庫的資料。

重新啟動資料遷移

若要從 CDC 起點重新啟動相關聯的資料遷移，請使用下列指示

重新啟動資料遷移

1. 在 RDS 主控台的資料庫頁面上選擇目標資料庫。
2. 在資料庫詳細資訊頁面上，選擇資料遷移索引標籤。
3. 若要重新啟動遷移，請選取資料遷移，然後在動作下拉式清單中選取重新啟動。
4. 這會從 CDC 起點重新啟動資料遷移。

重新啟動進行中的資料遷移不會影響任何已載入目標資料庫的資料。

監控資料遷移

資料遷移開始之後，您可以監控其狀態和進度。大型資料集的資料遷移需要數小時才能完成。為保持資料遷移的可靠性、可用性與高效能，請定期監控資料遷移的進度。

檢查資料遷移的狀態和進度

1. 在 RDS 主控台的資料庫頁面上選擇目標資料庫。
2. 在資料庫詳細資訊頁面上，選擇資料遷移索引標籤。
3. 相關聯資料遷移區段會列出您的資料遷移。檢查狀態欄。
4. 對於執行中的資料遷移，遷移進度欄會顯示遷移資料的百分比。

5. 若要監控 CloudWatch 中的程序，請使用 CloudWatch 欄中的連結。

遷移狀態

對於您執行的每個資料遷移，Aurora 主控台會顯示狀態。下列清單包含狀態：

- `Ready`：資料遷移已準備好開始。
- `Starting`：Aurora 正在為您的資料遷移建立無伺服器環境。
- `Load running`：Aurora 正在執行完整載入遷移。
- `Load complete, replication ongoing`：Aurora 已完成完整載入，現在會複寫進行中的變更。此狀態僅適用於完整載入和 CDC 類型遷移。
- `Replication ongoing`：Aurora 正在複寫進行中的變更。此狀態僅適用於 CDC 類型遷移。
- `Stopping`：Aurora 正在停止資料遷移。當您從動作功能表選擇停止資料遷移時，就會套用此狀態。
- `Stopped`：Aurora 已停止資料遷移。
- `Failed`：資料遷移失敗。請參閱日誌檔案以取得詳細資訊。
- `Restarting`：資料遷移已從 CDC 起點重新啟動進行中的資料複寫。

教學課程：使用自訂參數群組建立 MySQL 資料庫叢集

在本教學課程中，您會使用自訂參數群組建立 MySQL 資料庫叢集。如需參數群組的詳細資訊，請參閱 Amazon Aurora 資料庫叢集的資料庫叢集參數群組。

> **Important**
>
> 建立 AWS 帳戶無需付費。不過，透過完成本教學課程，您可能會因為使用 AWS 的資源而產生費用。如果不再需要這些資源，您可以在完成教學課程後刪除這些資源。

若要建立具有自訂組態和設定的資料庫叢集，您可以使用自訂參數群組。如果您使用多個資料庫，並想要統一地為它們進行設定，自訂參數群組特別有用。

完成這些步驟的過程中，您將了解：

- 如何透過自訂參數群組，使用 Amazon Aurora 建立 MySQL 資料庫叢集。
- 如何針對 MySQL 資料庫叢集使用特定參數。

請執行下列任務以完成本教學：

1. 使用 MySQL 參數 `default_password_lifetime` 建立資料庫叢集參數群組。
2. 使用您建立的自訂資料庫叢集參數群組，建立 MySQL 資料庫叢集。

主題

- 先決條件
- 建立 Amazon Aurora 資料庫叢集參數群組
- 修改自訂參數群組中的參數值
- 使用資料庫叢集參數群組建立 MySQL 資料庫叢集

先決條件

本教學課程需要您擁有具有管理存取權的 AWS 帳戶 和 使用者。如果您尚未完成設定，請完成下列區段中的步驟：

- 註冊 AWS 帳戶

- 建立具有管理存取權的使用者

建立 Amazon Aurora 資料庫叢集參數群組

在本教學課程中，您會了解如何在主控台中為 MySQL 資料庫叢集建立具有 default_password_lifetime 的自訂參數群組。default_password_lifetime 參數控制用戶端密碼自動過期之前的天數。如需 MySQL 資料庫叢集可用之其他參數的詳細資訊，請參閱 Aurora MySQL 組態參數。

建立參數群組

1. 開啟 Amazon RDS 主控台，然後選擇參數群組。
2. 對於自訂參數群組，選擇建立參數群組。
3. 設定參數群組詳細資訊。
    1. 輸入參數群組的名稱。
    2. 輸入參數群組的描述。
    3. 對於引擎類型，選擇 。Aurora MySQL
    4. 對於參數群組系列，選擇 aurora-mysql8.0。
    5. 對於類型，選擇資料庫叢集參數群組。
4. 選擇建立。

新的資料庫叢集參數群組會出現在 Amazon RDS 主控台的參數群組頁面上。下列步驟說明如何修改參數值以自訂參數群組。

修改自訂參數群組中的參數值

使用下列步驟來修改您在 建立 Amazon Aurora 資料庫叢集參數群組 中建立之參數群組中的參數值。

修改參數群組中的參數值

1. 開啟 Amazon RDS 主控台，然後選擇參數群組。
2. 對於自訂參數群組，選擇您建立的資料庫叢集參數群組名稱。
3. 選擇編輯。
4. 在篩選參數搜尋方塊中，搜尋自訂參數 default_password_lifetime。
5. 選取參數旁的核取方塊，然後輸入要為此密碼生命週期參數設定的天數值。
6. 選取 Save Changes (儲存變更)。

自訂參數群組現在可以與 Amazon Aurora for MySQL 8.0 資料庫叢集建立關聯。

使用資料庫叢集參數群組建立 MySQL 資料庫叢集

最後，使用您在先前步驟中建立的自訂參數群組，建立 MySQL 資料庫叢集。下列步驟說明如何使用自訂參數群組建立 MySQL 資料庫叢集。

使用自訂參數和新選項群組建立資料庫叢集

1. 開啟 Amazon RDS 主控台並選擇資料庫。
2. 選擇建立資料庫。
3. 對於選擇資料庫建立方法，選擇標準建立。
4. 對於引擎選項，選擇 Aurora (MySQL 相容)。
5. 選取其他組態。
   - 對於初始資料庫名稱，選擇資料庫叢集的名稱。
   - 在資料庫叢集參數群組下拉式清單底下，選取您先前建立之資料庫叢集參數群組的名稱。
6. 對於本教學課程，您可以保留任何其他資料庫設定的預設設定，或視需要進行修改。
7. 選擇建立資料庫。

RDS 會使用自訂參數群組群組建立新的 MySQL 資料庫叢集。若要查看此資料庫的詳細資訊，請參閱 Amazon RDS 主控台的資料庫頁面。

在本教學課程中，您使用自訂參數群組，以量身打造的設定來設定 MySQL 資料庫叢集。此新建立的 MySQL 資料庫叢集會使用參數 `default_password_lifetime` 來管理使用者密碼生命週期。若要最佳化資料庫，您可以在自訂參數群組中套用其他設定，並新增選項。

完成建立自訂資料庫叢集之後，您應該刪除資源，以避免產生不必要的成本。若要刪除資料庫叢集，請遵循 刪除 Aurora 資料庫叢集和資料庫執行個體 中的指示。

管理 Amazon Aurora 資料庫叢集

本節說明如何管理和維護 Aurora 資料庫叢集。Aurora 涉及在複寫拓撲中連線之資料庫伺服器的叢集。因此，管理 Aurora 通常需要對多個伺服器部署變更，並確保所有 Aurora 複本與來源伺服器同步。Aurora 隨著您的資料增加而透明地擴展基礎儲存空間，因此 Aurora 在磁碟儲存空間上需要的管理相對較少。同樣地，Aurora 會自動執行連續備份，因此 Aurora 叢集不需要大規模規劃或停機時間即可執行備份。

主題

- 停用和啟動 Amazon Aurora 資料庫叢集
- 自動連線 EC2 執行個體和 Aurora 資料庫叢集
- 自動連線 Lambda 函數和 Aurora 資料庫叢集
- 修改 Amazon Aurora 資料庫叢集
- 將 Aurora 複本新增至資料庫叢集
- 管理 Aurora 資料庫叢集的效能和擴展
- 複製 Amazon Aurora 資料庫叢集的一個磁碟區
- 將 Aurora 與其他 AWS 服務整合
- 維持 為 Amazon Aurora 資料庫叢集
- 重新啟動 Amazon Aurora 資料庫叢集或 Amazon Aurora 資料庫執行個體
- 容錯移轉 Amazon Aurora 資料庫叢集
- 刪除 Aurora 資料庫叢集和資料庫執行個體
- 標記 Amazon Aurora 和 Amazon RDS 資源
- Amazon RDS 中的 Amazon Resource Name (ARN)
- Amazon Aurora 更新

停用和啟動 Amazon Aurora 資料庫叢集

停用和啟動 Aurora 叢集可協助您管理開發和測試環境的成本。您可以暫時停用叢集中的所有資料庫執行個體，而非每次使用叢集時，設定和卸除所有資料庫執行個體。

主題

- 停用和啟動 Aurora 資料庫叢集的概觀
- 停用和啟動 Aurora 資料庫叢集的限制
- 停用 Aurora 資料庫叢集
- 停用 Aurora 資料庫叢集時的可能操作
- 啟動 Aurora 資料庫叢集

停用和啟動 Aurora 資料庫叢集的概觀

在不需要 Aurora 資料庫叢集的期間，您可以一次停用該叢集中的所有執行個體。一旦您需要叢集，即可隨時重新啟動它。啟動和停用可簡化用於下列操作之叢集的設定和卸除程序：開發、測試或不需要連續可用性的類似活動。無論叢集中有多少執行個體，您都只能執行與單一動作相關的所有 AWS 管理主控台 程序。

資料庫叢集停用時，只需支付您指定的保留時段內叢集儲存、手動快照和自動備份儲存的費用。您無須支付任何資料庫執行個體小時數的費用。

> **Important**
>
> 您可以停用資料庫叢集長達七天。如果七天後您沒有手動啟動資料庫叢集，您的資料庫叢集會自動啟動，如此一來便不會落後於任何必要的維護更新。

若要將輕度載入之 Aurora 叢集的費用降至最低，您可以停用該叢集，而非刪除其所有 Aurora 複本。對於具有多個或兩個執行個體的叢集，經常刪除和重新建立資料庫執行個體只會使用 AWS CLI 或 Amazon RDS API。這樣一系列的操作也可能難以按正確的順序執行，例如，先刪除所有 Aurora 複本，然後再刪除主要執行個體，以避免啟動容錯移轉機制。

如果您需要保持資料庫叢集執行中，但它具有的容量超過所需，請勿使用啟動和停用。如果您的叢集太昂貴或不是非常忙碌，請刪除一個或多個資料庫執行個體，或將您的所有資料庫執行個體變更為小型執行個體類別。您無法停用個別 Aurora 資料庫執行個體。

停止資料庫叢集的時間會根據資料庫執行個體類別、網路狀態、資料庫引擎類型和資料庫狀態等因素而有所不同。該程序需要幾分鐘的時間來完成。Amazon RDS 服務會執行下列動作：

- 關閉資料庫引擎程序。
- 關閉 RDS 平台程序。
- 終止基礎 Amazon EC2 執行個體。

重新啟動資料庫叢集的時間取決於資料庫大小、資料庫執行個體類別、網路狀態、資料庫引擎類型，以及叢集關閉時的資料庫狀態等因素。啟動程序可能需要幾分鐘到幾小時，但通常需要幾分鐘的時間。我們建議您在建立可用性計劃時，考慮啟動時間的變動性。

為了啟動資料庫叢集，服務會執行下列動作：

- 佈建基礎 Amazon EC2 執行個體。
- 啟動 RDS 平台程序。
- 啟動資料庫引擎程序。
- 復原資料庫執行個體 (即使在正常關閉後也會復原)。

停用和啟動 Aurora 資料庫叢集的限制

部分 Aurora 叢集無法停用和啟動，如下方所述：

- 只有在叢集是全域資料庫中的唯一叢集時，您才能停止和啟動屬於 Aurora 全球資料庫的叢集。
- 您無法停止和啟動具跨區域僅供讀取複本的叢集。
- 您無法停止和啟動屬於藍/綠部署一部分的叢集。
- 您無法停止並啟動 Aurora Serverless v1 叢集。利用 Aurora Serverless v2，您無法停止和啟動叢集。

停用 Aurora 資料庫叢集

若要使用 Aurora 資料庫叢集或執行管理，一律以執行中的 Aurora 資料庫叢集開始，接著停用叢集，然後重新啟動叢集。您的叢集停用時，需支付您指定的保留時段內叢集儲存、手動快照和自動備份儲存的費用，但無須支付資料庫執行個體小時數。

停用操作首先會停用 Aurora 複本執行個體，然後停用主要執行個體，以避免啟動容錯移轉機制。

主控台

停用 Aurora 叢集

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)，然後選擇一個叢集。您可以從這個頁面執行停用操作，或導覽至欲停用資料庫叢集的詳細資訊頁面。
3. 針對 Actions (動作)，選擇 Stop temporarily (暫時停止)。
4. 在 Stop DB cluster temporarily (暫時停止資料庫叢集) 視窗中，選取資料庫叢集將在 7 天後自動重新啟動的確認。
5. 選擇 Stop temporarily (暫時停止) 來停止資料庫叢集，或選擇 Cancel (取消) 來取消操作。

AWS CLI

若要使用 停止資料庫執行個體 AWS CLI，請使用下列參數呼叫 stop-db-cluster 命令：

- `--db-cluster-identifier` – Aurora 叢集的名稱。

Example

```
aws rds stop-db-cluster --db-cluster-identifier mydbcluster
```

RDS API

若要使用 Amazon RDS API 停止資料庫執行個體，請搭配下列參數呼叫 StopDBCluster 操作：

- `DBClusterIdentifier` – Aurora 叢集的名稱。

停用 Aurora 資料庫叢集時的可能操作

Aurora 叢集停用時，您可以執行時間點還原，以還原到您所指定自動備份保留時段中的任何時間點。如需執行時間點還原的詳細資訊，請參閱還原資料。

Aurora 資料庫叢集停用時，您無法修改此叢集的組態或其任何資料庫執行個體。您也無法新增或移除叢集中的資料庫執行個體，或如果叢集仍有任何相關聯的資料庫執行個體，則無法刪除此叢集。您必須先啟動叢集，然後才能執行任何這類管理動作。

停用資料庫叢集會移除擱置中動作，但資料庫叢集參數群組或資料庫叢集執行個體的資料庫參數群組除外。

在您已停用的叢集重新啟動之後，Aurora 會將任何排程的維護套用至這個叢集。請記住，Aurora 會在七天後自動啟動任何已停用的叢集，以便它們不會落後其維護狀態太遠。

Aurora 也不會執行任何自動備份，因為當叢集停用時，基礎資料無法變更。Aurora 不會在叢集停止時延長備份保留期間。

啟動 Aurora 資料庫叢集

您一律會啟動 Aurora 資料庫叢集，從已處於停用狀態的 Aurora 叢集開始。當您啟動叢集時，所有其資料庫執行個體會再次變為可用。叢集會保留其組態設定，例如端點、參數群組及 VPC 安全群組。

啟動您的資料庫叢集通常需要幾分鐘的時間。

主控台

啟動 Aurora 叢集

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)，然後選擇一個叢集。您可以從這個頁面執行啟動操作，或導覽至欲啟動資料庫叢集的詳細資訊頁面。
3. 針對 Actions (動作)，選擇 Start (啟動)。

AWS CLI

若要使用 啟動資料庫叢集 AWS CLI，請使用下列參數呼叫 start-db-cluster 命令：

- `--db-cluster-identifier` – Aurora 叢集的名稱。此名稱不是您在建立叢集時選擇的特定叢集識別碼，就是您選擇並有 `-cluster` 附加至其結尾的資料庫執行個體識別碼。

Example

```
aws rds start-db-cluster --db-cluster-identifier mydbcluster
```

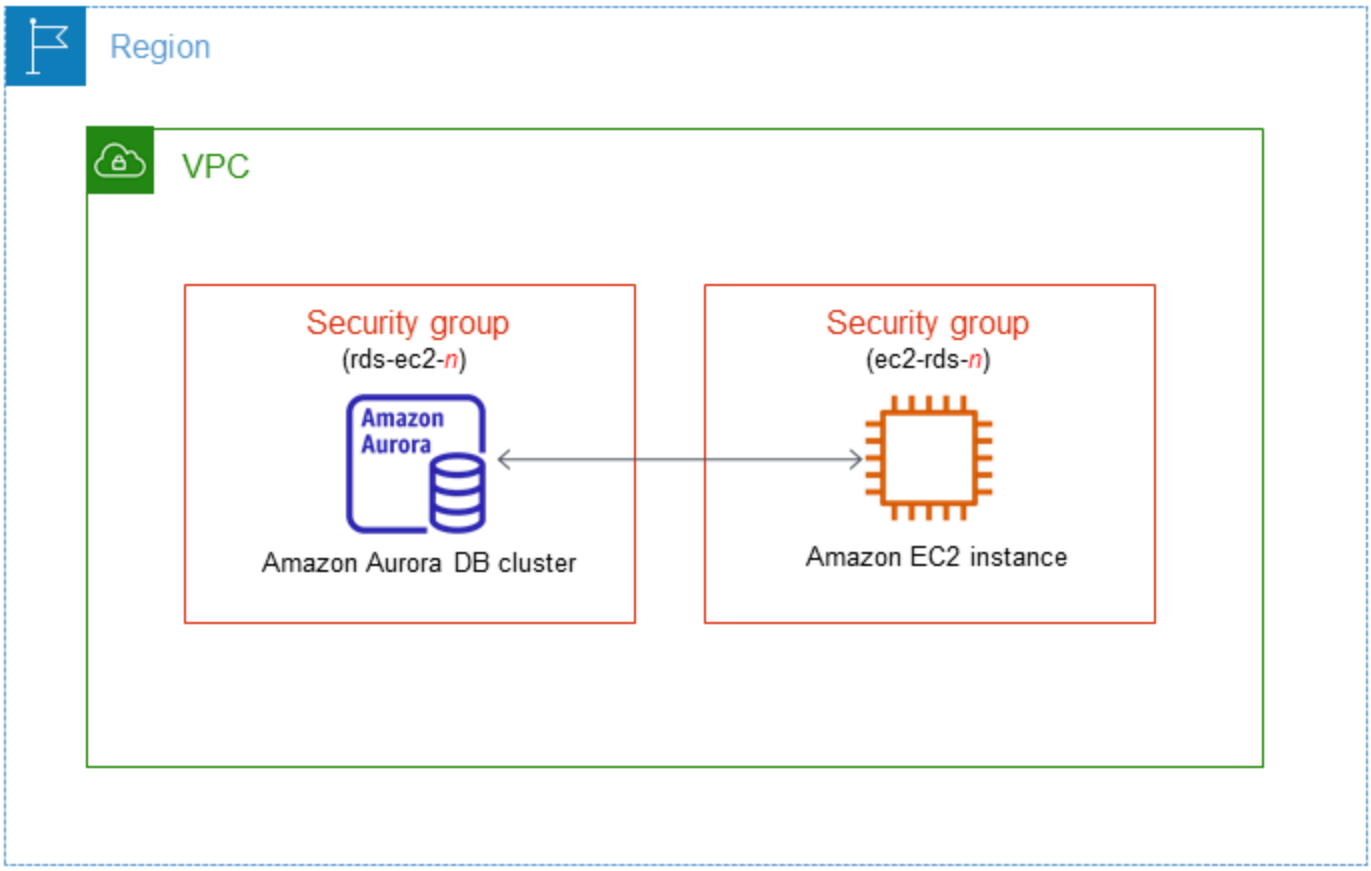
RDS API

若要使用 Amazon RDS API 啟動 Aurora 資料庫叢集，請搭配下列參數呼叫 StartDBCluster 操作：

- DBCluster – Aurora 叢集的名稱。此名稱不是您在建立叢集時選擇的特定叢集識別碼，就是您選擇並有 -cluster 附加至其結尾的資料庫執行個體識別碼。

自動連線 EC2 執行個體和 Aurora 資料庫叢集

您可以使用 Amazon RDS 主控台，簡化 Amazon Elastic Compute Cloud (Amazon EC2) 執行個體與 Aurora 資料庫叢集之間的連線設定。通常，您的資料庫叢集位於私有子網路中，而 EC2 執行個體位於 VPC 內的公有子網路中。您可以在 EC2 執行個體上使用 SQL 用戶端，連線到資料庫叢集。EC2 執行個體也可以執行存取私有資料庫叢集的 Web 伺服器或應用程式。



如果您想要連線至與 Aurora 資料庫叢集不在相同 VPC 中的 EC2 執行個體，請參閱 在 VPC 中存取資料庫叢集的案例 中的案例。

主題

- 與 EC2 執行個體自動連線的概要
- 自動連線 EC2 執行個體和 Aurora 資料庫叢集
- 檢視已連線的運算資源
- 連線至執行特定資料庫引擎的資料庫執行個體

與 EC2 執行個體自動連線的概要

當您在 EC2 執行個體與 Aurora 資料庫叢集之間自動設定連線時，Amazon RDS 會為您的 EC2 執行個體以及 資料庫叢集設定 VPC 安全群組。

以下是連線 EC2 執行個體與 Aurora 資料庫叢集的要求：

- EC2 執行個體必須與 資料庫叢集存在於相同的 VPC 中。

  如果沒有 EC2 執行個體存在於相同的 VPC 中，則主控台會提供一個連結來建立該執行個體。
- 目前，資料庫叢集不能是 Aurora Serverless 資料庫叢集或 Aurora 全球資料庫的一部分。
- 設定連線的使用者必須擁有執行下列 Amazon EC2 操作的許可：
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeSecurityGroups`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

如果資料庫執行個體和 EC2 執行個體位於不同的可用區域，您的帳戶可能會產生跨可用區域成本。

當您設定 EC2 執行個體的連線時，Amazon RDS 會根據與 資料庫叢集和 EC2 執行個體相關聯之安全群組的目前組態執行動作，如下表所述。

目前 RDS 安全群組組態	目前 EC2 安全群組組態	RDS 動作
有一或多個與 資料庫叢集相關聯的安全群組，其名稱符合模式`rds-ec2-`*n* (其中 *n* 是數字)。符合模式的安全群組尚未修改。此安全群組只包含一個傳入規則，其具有 EC2 執行個體的 VPC 安全群組做為來源。	有一或多個與 EC2 執行個體相關聯的安全群組，其名稱符合模式 `ec2-rds-`*n* (其中 *n* 是數字)。符合模式的安全群組尚未修改。此安全群組只包含一個傳出規則，其具有 資料庫叢集的 VPC 安全群組做為來源。	RDS 不會採取任何動作。 已在 EC2 執行個體與 資料庫叢集之間自動設定連線。由於 EC2 執行個體與 RDS 資料庫之間已存在連線，因此不會修改安全群組。

目前 RDS 安全群組組態	目前 EC2 安全群組組態	RDS 動作
以下任一種條件均適用： • 沒有與 資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-ec2-`*n*。 • 有一或多個與 資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-ec2-`*n*。不過，Amazon RDS 無法使用這些安全群組中的任一個與 EC2 執行個體連線。Amazon RDS 無法使用沒有傳入規則的安全群組，並將具有 EC2 執行個體的 VPC 安全群組做為來源。Amazon RDS 也無法使用經過修改的安全群組。修改範例包括新增規則或變更現有規則的連線埠。	以下任一種條件均適用： • 沒有與 EC2 執行個體相關聯的安全群組，其名稱符合模式 `ec2-rds-`*n*。 • 有一或多個與 EC2 執行個體相關聯的安全群組，其名稱符合模式 `ec2-rds-`*n*。不過，Amazon RDS 無法使用這些安全群組中的任一個與資料庫叢集連線。Amazon RDS 無法使用沒有傳出規則的安全群組，並將具有 資料庫叢集的 VPC 安全群組做為來源。Amazon RDS 也無法使用經過修改的安全群組。	RDS action: create new security groups
有一或多個與 資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-ec2-`*n*。符合模式的安全群組尚未修改。此安全群組只包含一個傳入規則，其具有 EC2 執行個體的 VPC 安全群組做為來源。	有一或多個與 EC2 執行個體相關聯的安全群組，其名稱符合模式 `ec2-rds-`*n*。不過，Amazon RDS 無法使用這些安全群組中的任一個與 資料庫叢集連線。Amazon RDS 無法使用沒有傳出規則的安全群組，並將具有 資料庫叢集的 VPC 安全群組做為來源。Amazon RDS 也無法使用經過修改的安全群組。	RDS action: create new security groups

目前 RDS 安全群組組態	目前 EC2 安全群組組態	RDS 動作
有一或多個與 資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-ec2-`*n*。符合模式的安全群組尚未修改。此安全群組只包含一個傳入規則，其具有 EC2 執行個體的 VPC 安全群組做為來源。	存在用於連線的有效 EC2 安全群組，但與 EC2 執行個體沒有相關聯。此安全群組具有符合模式 `ec2-rds-`*n* 的名稱。尚未將其修改。其只包含一個傳出規則，具有 資料庫叢集的 VPC 安全群組做為來源。	RDS action: associate EC2 security group
以下任一種條件均適用： • 沒有與 資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-ec2-`*n*。 • 有一或多個與 資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-ec2-`*n*。不過，Amazon RDS 無法使用這些安全群組中的任一個與 EC2 執行個體連線。Amazon RDS 無法使用沒有傳入規則的安全群組，並將具有 EC2 執行個體的 VPC 安全群組做為來源。Amazon RDS 也無法使用經過修改的安全群組。	有一或多個與 EC2 執行個體相關聯的安全群組，其名稱符合模式 `ec2-rds-`*n*。符合模式的安全群組尚未修改。此安全群組只包含一個傳出規則，其具有 資料庫叢集的 VPC 安全群組做為來源。	RDS action: create new security groups

RDS 動作：建立新的安全群組

Amazon RDS 會採取下列動作：

- 建立符合模式 `rds-ec2-`*n* 的新安全群組。此安全群組具有一個傳入規則，其具有 EC2 執行個體的 VPC 安全群組做為來源。此安全群組與 資料庫叢集相關聯，並可讓 EC2 執行個體存取 資料庫叢集。

- 建立符合模式 `ec2-rds-n` 的新安全群組。此安全群組具有一個傳出規則，其具有 資料庫叢集的 VPC 安全群組作為目標。此安全群組與 EC2 執行個體相關聯，並可讓 EC2 執行個體將流量傳送到 資料庫叢集。

RDS 動作：關聯 EC2 安全群組

Amazon RDS 會將有效的現有 EC2 安全群組與 EC2 執行個體建立關聯。此安全群組允許 EC2 執行個體將流量傳送到 資料庫叢集。

自動連線 EC2 執行個體和 Aurora 資料庫叢集

在設定 EC2 執行個體與 Aurora 資料庫叢集之間的連線之前，請確定您符合 與 EC2 執行個體自動連線的概要 中所述的要求。

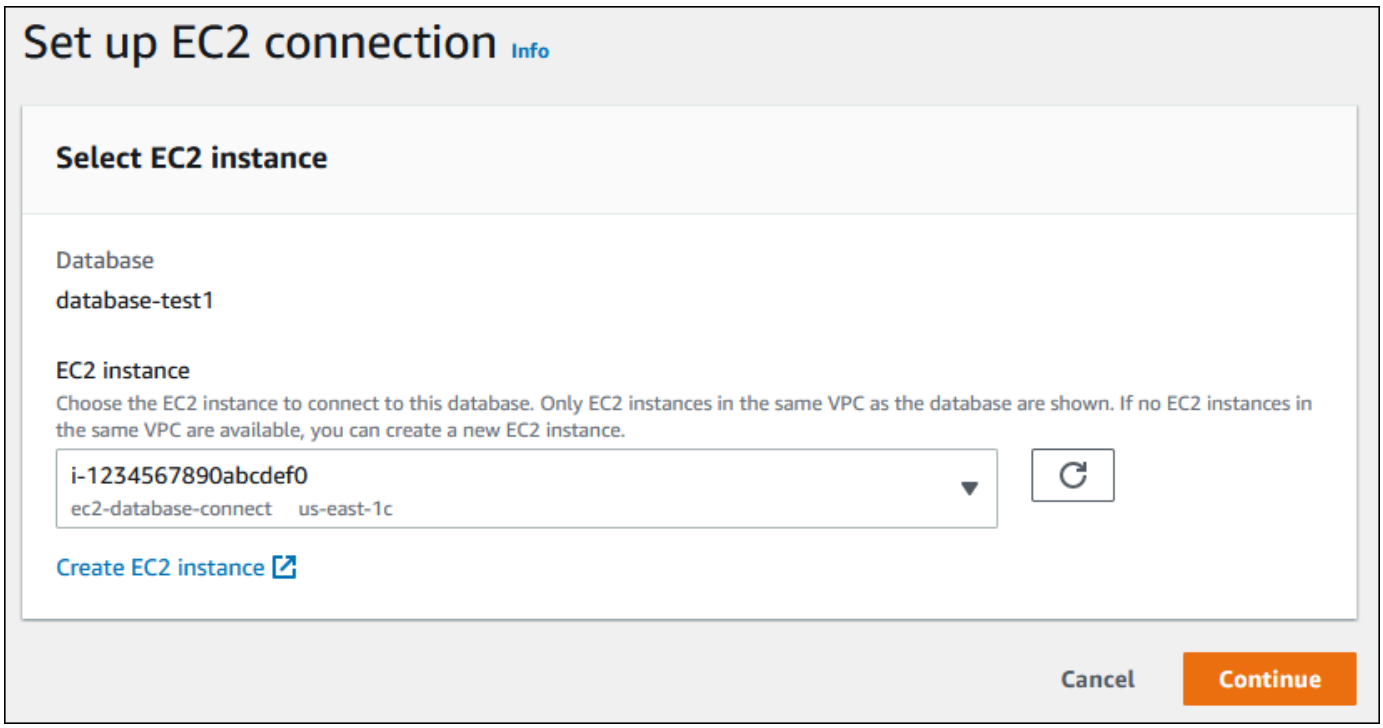
如果您在設定連線之後變更這些安全群組，這些變更可能會影響 EC2 執行個體與 Aurora 資料庫叢集之間的連線。

> **Note**
>
> 您只能使用 AWS 管理主控台，在 EC2 執行個體與 Aurora 資料庫叢集之間自動設定連線。您無法自動設定與 AWS CLI 或 RDS API 的連線。

自動連線 EC2 執行個體與 Aurora 資料庫叢集

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ :// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)，然後選擇 資料庫叢集。
3. 從動作，選擇設定 EC2 連線。

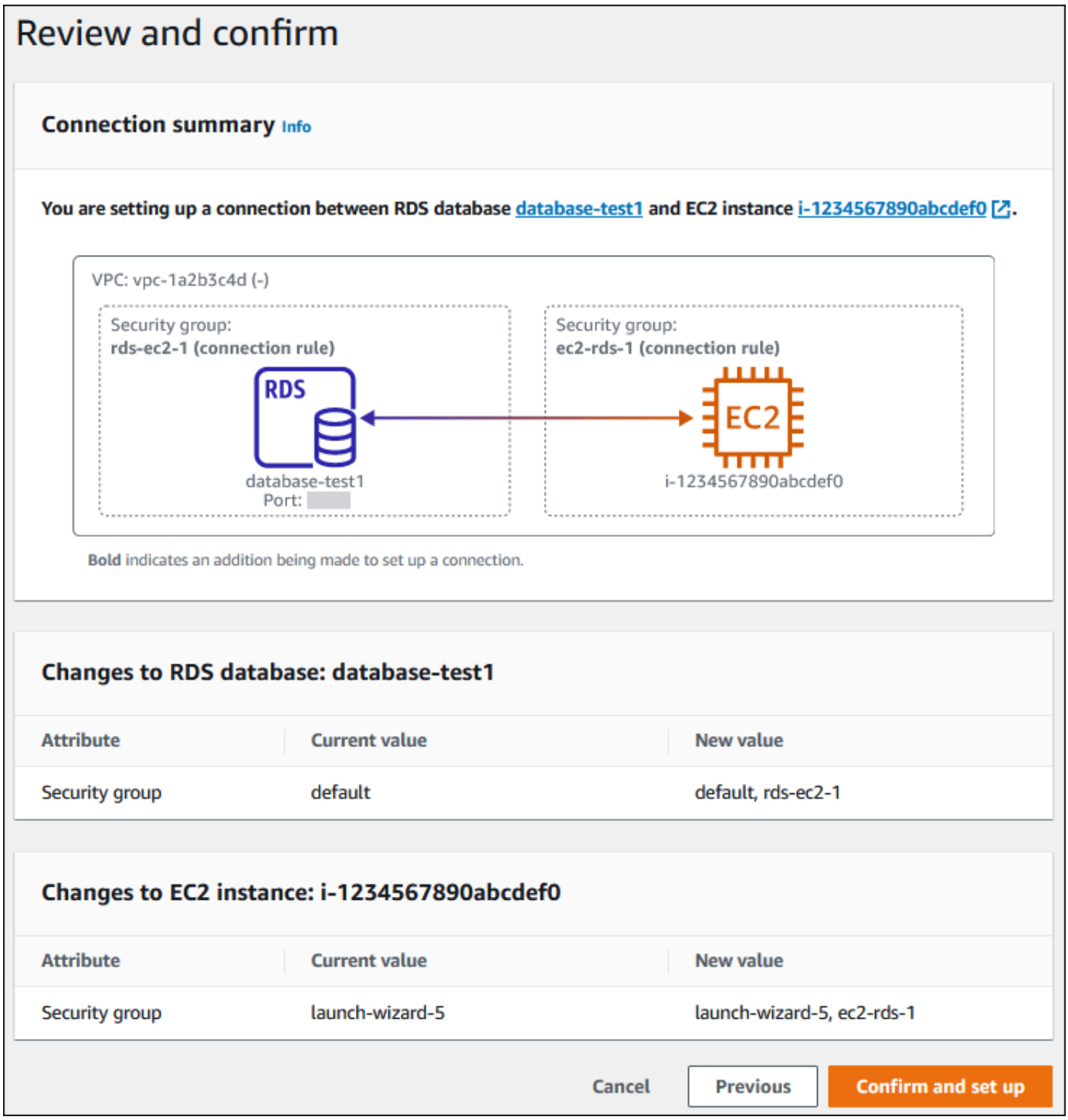
   Set up EC2 connection (設定 EC2 連線) 頁面即會出現。
4. 在 Set up EC2 connection (設定 EC2 連線) 頁面上，選擇 EC2 執行個體。



如果相同 VPC 中沒有 EC2 執行個體，請選擇 Create EC2 instance (建立 EC2 執行個體) 來建立執行個體。在此情況下，請確定新 EC2 執行個體與 資料庫叢集位於相同的 VPC 中。

5. 選擇繼續。

   Review and confirm (檢閱並確認) 頁面即會出現。



6. 在 Review and confirm (檢閱並確認) 頁面上，檢閱 RDS 將做出以設定與 EC2 執行個體連線的變更。

   如果變更正確，請選擇確認並設定。

   如果變更不正確，請選擇 Previous (上一步) 或 Cancel (取消)。

檢視已連線的運算資源

您可以使用 AWS 管理主控台 檢視連線至 Aurora 資料庫叢集的運算資源。顯示的資源包括已自動設定的運算資源連線。您可以使用下列方式自動設定與運算資源的連線：

- 您可以在建立資料庫時選取運算資源。

  如需詳細資訊，請參閱建立 Amazon Aurora 資料庫叢集。
- 您可以設定現有資料庫與運算資源之間的連線。

  如需詳細資訊，請參閱自動連線 EC2 執行個體和 Aurora 資料庫叢集。

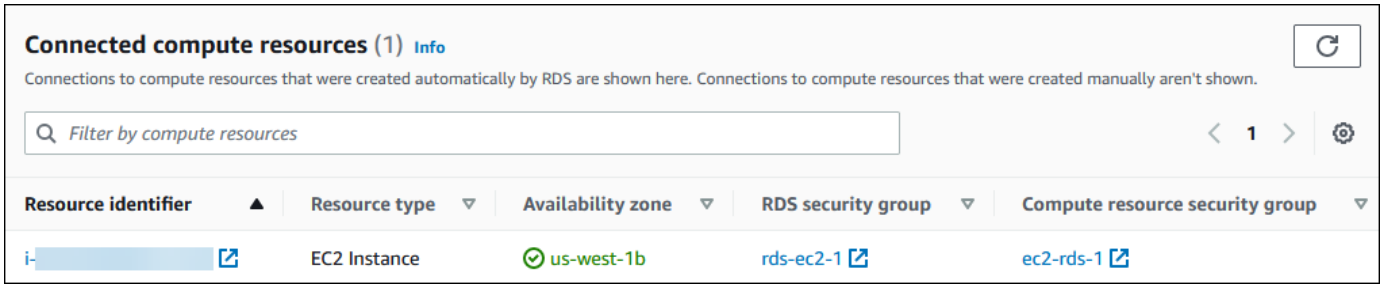
列出的運算資源不包括已手動連線至資料庫的運算資源。例如，您可以透過將規則新增至與資料庫關聯的 VPC 安全群組，來允許運算資源手動存取資料庫。

針對要列出的運算資源，必須滿足下列條件：

- 與運算資源相關聯的安全群組，其名稱符合模式 `ec2-rds-n` (其中 *n* 是數字)。
- 與運算資源相關聯的安全群組具有傳出規則，其連接埠範圍設為 資料庫叢集所使用的連接埠。
- 與運算資源相關聯的安全群組具有傳出規則，其來源設為與 資料庫叢集相關聯的安全群組。
- 與 資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-ec2-n` (其中 *n* 是數字)。
- 與 資料庫叢集相關聯的安全群組具有傳入規則，其連接埠範圍設為 資料庫叢集所使用的連接埠。
- 與 資料庫叢集相關聯的安全群組具有傳入規則，其來源設為與運算資源相關聯的安全群組。

若要檢視連線至 Aurora 資料庫叢集的運算資源。

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)，然後選擇 資料庫叢集的名稱。
3. 在 Connectivity & security (連線和安全) 索引標籤上，檢視 Connected compute resources (已連線的運算資源)。



連線至執行特定資料庫引擎的資料庫執行個體

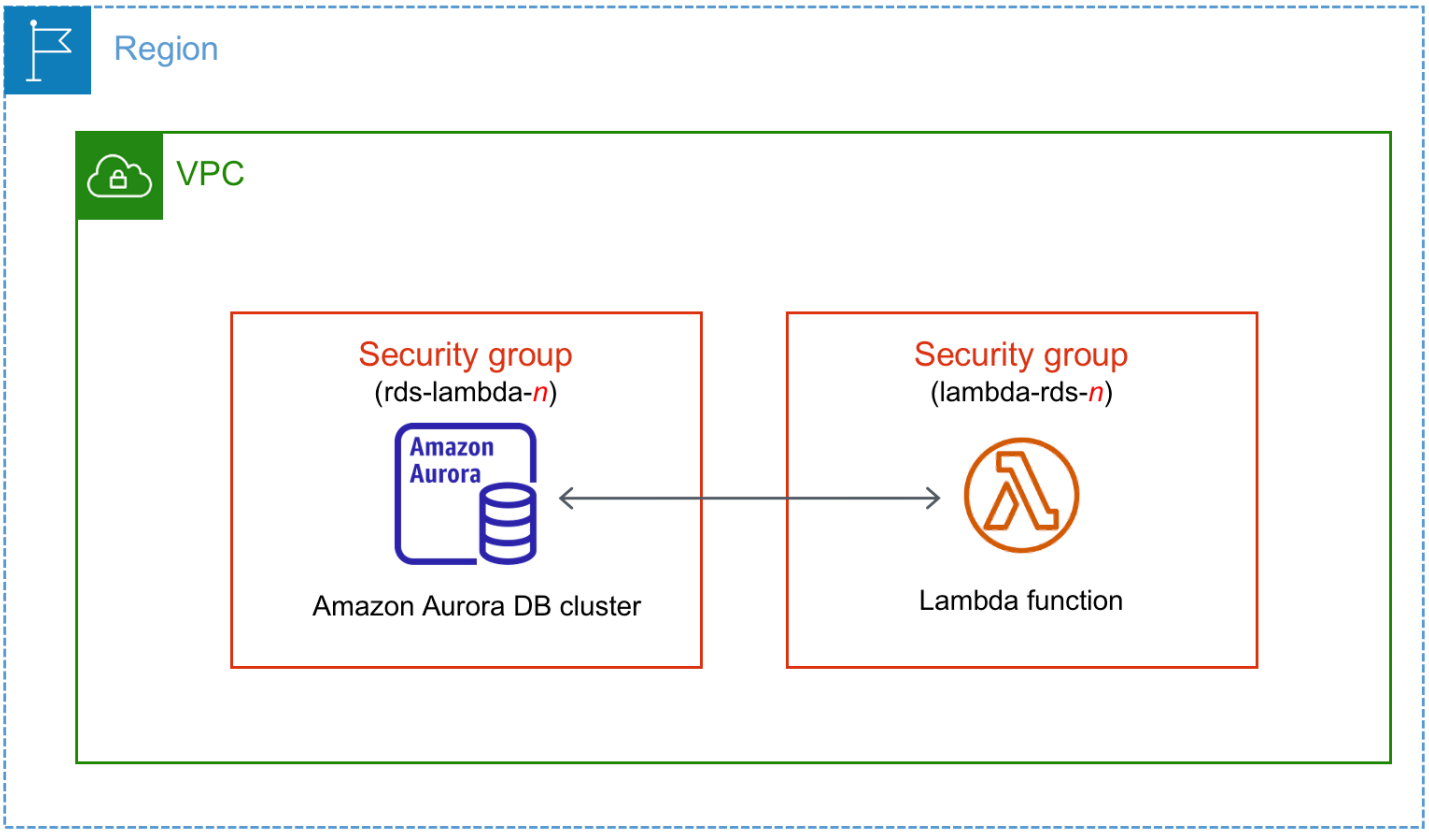
如需連線至執行特定資料庫引擎之資料庫執行個體的相關資訊，請遵循資料庫引擎的指示：

- 連接至 Amazon Aurora MySQL 資料庫叢集
- 連接至 Amazon Aurora PostgreSQL 資料庫叢集

自動連線 Lambda 函數和 Aurora 資料庫叢集

您可以使用 Amazon RDS 主控台，簡化 Lambda 函數與 Aurora 資料庫叢集之間的連線設定。通常，您的資料庫叢集位於 VPC 內的私有子網路中。Lambda 函數可供應用程式用來存取您的私有資料庫叢集。

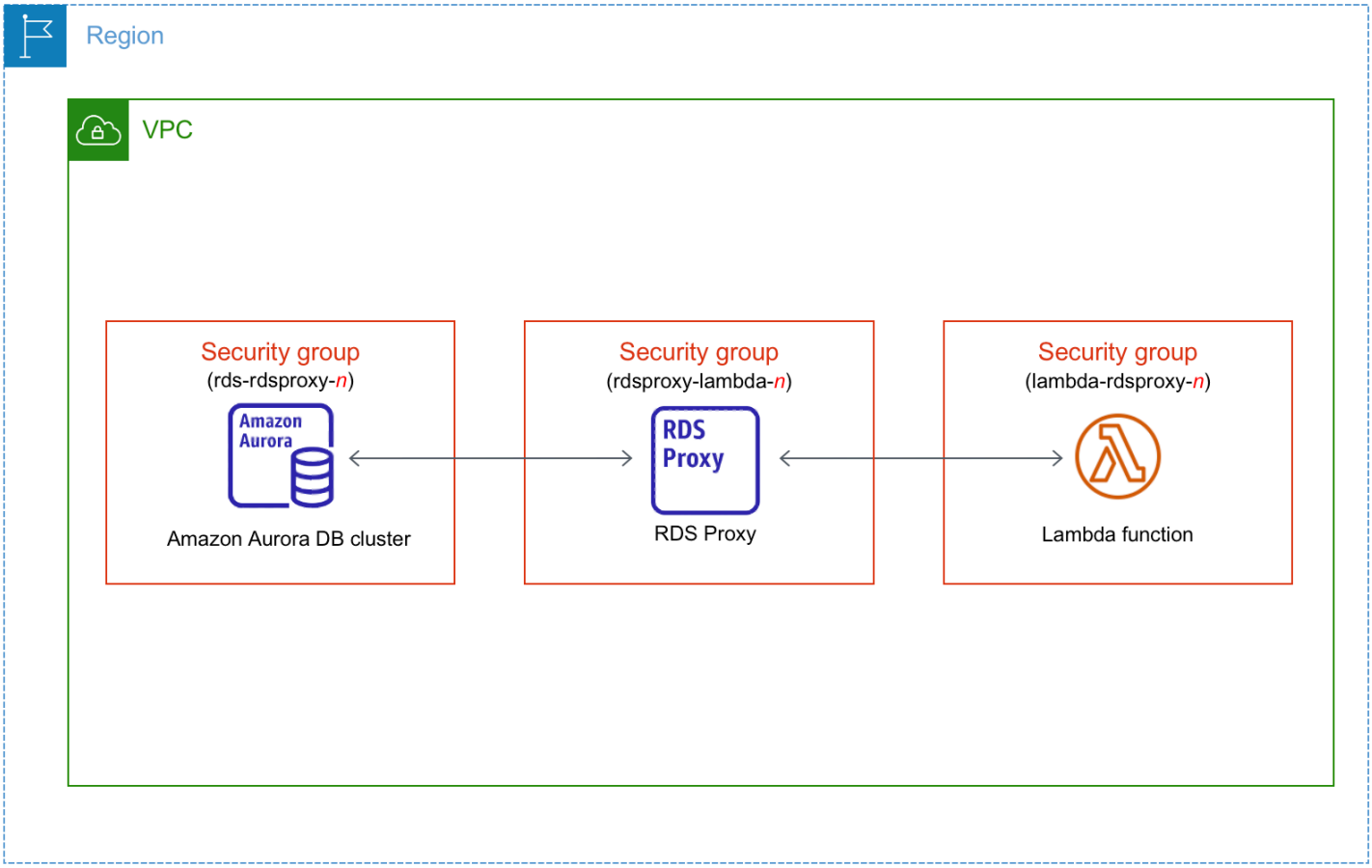
下圖顯示您的資料庫叢集和 Lambda 函數之間的直接連線。



您可以透過 RDS Proxy 設定 Lambda 函數和資料庫叢集來改善資料庫效能和復原能力。通常，Lambda 函數經常進行短暫的資料庫連線，這些連線將受益於 RDS Proxy 提供的連線集區。您可以利用 Lambda 函數已擁有的任何 AWS Identity and Access Management (IAM) 身分驗證，而不是在 Lambda 應用程式程式碼中管理資料庫登入資料。如需詳細資訊，請參閱Amazon RDS Proxy for Aurora。

當您使用主控台與現有代理連線時，Amazon RDS 會更新代理安全群組，以允許來自資料庫叢集和 Lambda 函數的連線。

您也可以從相同的主控台頁面建立新的代理。在主控台中建立代理時，若要存取資料庫叢集，您必須輸入資料庫憑證或選取 AWS Secrets Manager 秘密。



Tip

若要快速將 Lambda 函數連線至Aurora 資料庫叢集，您也可以使用主控台內引導精靈。若要開啟精靈，請執行下列操作：

1. 開啟 Lambda 主控台中的函數頁面。
2. 選取您要連接資料庫的函數。
3. 在組態索引標籤，選取 RDS 資料庫。
4. 選擇連接至 RDS 資料庫。

將函數連接至資料庫之後，您可以選擇新增代理來建立代理。

主題

- 使用 Lambda 函數自動連線概觀
- 自動連線 Lambda 函數和 Aurora 資料庫叢集

- 檢視已連線的運算資源

使用 Lambda 函數自動連線概觀

以下是連線 Lambda 函數與 Aurora 資料庫叢集的要求：

- Lambda 函數必須存在於與資料庫叢集相同的 VPC 內。
- 目前，資料庫叢集不能是 Aurora Serverless 資料庫叢集或 Aurora 全球資料庫的一部分。
- 設定連線的使用者必須擁有執行下列 Amazon RDS、Amazon EC2、Lambda、Secrets Manager 和 IAM 操作的許可：
  - Amazon RDS
    - `rds:CreateDBProxies`
    - `rds:DescribeDBClusters`
    - `rds:DescribeDBProxies`
    - `rds:ModifyDBCluster`
    - `rds:ModifyDBProxy`
    - `rds:RegisterProxyTargets`
  - Amazon EC2
    - `ec2:AuthorizeSecurityGroupEgress`
    - `ec2:AuthorizeSecurityGroupIngress`
    - `ec2:CreateSecurityGroup`
    - `ec2:DeleteSecurityGroup`
    - `ec2:DescribeSecurityGroups`
    - `ec2:RevokeSecurityGroupEgress`
    - `ec2:RevokeSecurityGroupIngress`
  - Lambda
    - `lambda:CreateFunctions`
    - `lambda:ListFunctions`
    - `lambda:UpdateFunctionConfiguration`
  - Secrets Manager
    - `secretsmanager:CreateSecret`
    - `secretsmanager:DescribeSecret`

- IAM
  - `iam:AttachPolicy`
  - `iam:CreateRole`
  - `iam:CreatePolicy`
- AWS KMS
  - `kms:describeKey`

> **Note**
>
> 如果資料庫叢集和 Lambda 函數位於不同的可用區域，您的帳戶可能會產生跨可用區域成本。

當您在 Lambda 函數與 Aurora 資料庫叢集之間自動設定連線時，Amazon RDS 會為您的函數以及資料庫叢集設定 VPC 安全群組。如果您使用 RDS Proxy，則 Amazon RDS 也會為代理設定 VPC 安全群組。Amazon RDS 會根據與資料庫叢集、Lambda 函數和代理相關聯之安全群組的目前組態執行動作，如下表所述。

目前 RDS 安全群組組態	目前 Lambda 安全群組組態	目前代理安全群組組態	RDS 動作
有一或多個與資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-lambda-` *n*，或者如果代理已連線至您的資料庫叢集，RDS 將檢查相關聯代理的 `TargetHealth` 是否為 `AVAILABLE`。 符合模式的安全群組尚未修改。此安全群組只有一個傳入規則，其具有 Lambda 函	有一或多個與 Lambda 函數相關聯的安全群組，其名稱符合模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n* (其中 *n* 是數字)。 符合模式的安全群組尚未修改。此安全群組只有一個傳出規則，其使用資料庫叢集的 VPC 安全群組或代理做為目的地。	有一或多個與代理相關聯的安全群組，其名稱符合模式 `rdsproxy-lambda-` *n* (其中 *n* 是數字)。 符合模式的安全群組尚未修改。此安全群組具有傳入和傳出規則，其中包含 Lambda 函數和資料庫叢集的 VPC 安全群組。	Amazon RDS 不會採取任何動作。 Lambda 函數、代理 (選用) 和資料庫叢集之間已自動設定連線。由於函數、代理與資料庫之間已存在連線，因此不會修改安全群組。

目前 RDS 安全群組組態	目前 Lambda 安全群組組態	目前代理安全群組組態	RDS 動作
數或代理的 VPC 安全群組做為來源。			

目前 RDS 安全群組組態	目前 Lambda 安全群組組態	目前代理安全群組組態	RDS 動作
以下任一種條件均適用： • 沒有與資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-lambda-` *n*，或者如果相關聯代理的 `TargetHealth` 是 `AVAILABLE` 。 • 有一或多個與資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-lambda-` *n*，或者如果相關聯代理的 `TargetHealth` 是 `AVAILABLE` 。不過，這些安全群組都不能用於與 Lambda 函數的連線。 Amazon RDS 無法使用沒有一個傳入規則 (使用 Lambda 函數的 VPC 安全群組或代理做為來源) 的安全群組。Amazon RDS 也無法使用經過修改的安全群組。修改範例	以下任一種條件均適用： • 沒有與 Lambda 函數相關聯的安全群組，其名稱符合模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n*。 • 有一或多個與 Lambda 函數相關聯的安全群組，其名稱符合模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n*。不過，Amazon RDS 無法使用這些安全群組中的任一個與資料庫叢集連線。 Amazon RDS 無法使用沒有傳出規則的安全群組，並將具有資料庫叢集或代理的 VPC 安全群組做為目的地。Amazon RDS 也無法使用經過修改的安全群組。	以下任一種條件均適用： • 沒有與代理相關聯的安全群組，其名稱符合模式 `rdsproxy-lambda-` *n*。 • 有一或多個與代理相關聯的安全群組，其名稱符合 `rdsproxy-lambda-` *n*。不過，Amazon RDS 無法使用這些安全群組中的任一個與資料庫叢集或 Lambda 函數連線。 Amazon RDS 無法使用沒有傳入和傳出規則 (包含資料庫叢集和 Lambda 函數的 VPC 安全群組) 的安全群組。Amazon RDS 也無法使用經過修改的安全群組。	RDS action: create new security groups

目前 RDS 安全群組組態	目前 Lambda 安全群組組態	目前代理安全群組組態	RDS 動作
包括新增規則或變更現有規則的連線埠。			
有一或多個與資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-lambda-` *n*，或者如果相關聯代理的 `TargetHealth` 是 `AVAILABLE` 。 符合模式的安全群組尚未修改。此安全群組只有一個傳入規則，其具有 Lambda 函數或代理的 VPC 安全群組做為來源。	有一或多個與 Lambda 函數相關聯的安全群組，其名稱符合模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n*。 不過，Amazon RDS 無法使用這些安全群組中的任一個與資料庫叢集連線。Amazon RDS 無法使用沒有傳出規則的安全群組，並將具有資料庫叢集或代理的 VPC 安全群組做為目的地。Amazon RDS 也無法使用經過修改的安全群組。	有一或多個與代理相關聯的安全群組，其名稱符合模式 `rdsproxy-lambda-` *n*。 不過，Amazon RDS 無法使用這些安全群組中的任一個與資料庫叢集或 Lambda 函數連線。Amazon RDS 無法使用沒有傳入和傳出規則 (包含資料庫叢集和 Lambda 函數的 VPC 安全群組) 的安全群組。Amazon RDS 也無法使用經過修改的安全群組。	RDS action: create new security groups

目前 RDS 安全群組組態	目前 Lambda 安全群組組態	目前代理安全群組組態	RDS 動作
有一或多個與資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-lambda-` *n*，或者如果相關聯代理的 `TargetHealth` 是 `AVAILABLE` 。 符合模式的安全群組尚未修改。此安全群組只有一個傳入規則，其具有 Lambda 函數或代理的 VPC 安全群組做為來源。	存在用於連線的有效 Lambda 安全群組，但與 Lambda 函數沒有相關聯。此安全群組具有符合模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n* 的名稱。尚未將其修改。只有一個輸出規則，其中的 VPC 安全群組資料庫叢集或代理做為目的地。	存在用於連線的有效代理安全群組，但與代理沒有相關聯。此安全群組具有符合模式 `rdsproxy-lambda-` *n* 的名稱。尚未將其修改。具有包含資料庫叢集的 VPC 安全群組的傳入和傳出規則，和 Lambda 函數。	RDS action: associate Lambda security group

目前 RDS 安全群組組態	目前 Lambda 安全群組組態	目前代理安全群組組態	RDS 動作
以下任一種條件均適用： • 沒有與資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-lambda-` *n*，或者如果相關聯代理的 `TargetHealth` 是 `AVAILABLE` 。 • 有一或多個與資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-lambda-` *n*，或者如果相關聯代理的 `TargetHealth` 是 `AVAILABLE` 。不過，Amazon RDS 無法使用這些安全群組中的任一個與 Lambda 函數或代理連線。 Amazon RDS 無法使用沒有一個傳入規則 (使用 Lambda 函數的 VPC 安全群組或代理做為來源) 的安全群組。Amazon RDS 也無法使用經過修改的安全群組。	有一或多個與 Lambda 函數相關聯的安全群組，其名稱符合模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n*。 符合模式的安全群組尚未修改。此安全群組只有一個輸出規則，其中的 VPC 安全群組資料庫執行個體或代理做為目的地。	有一或多個與代理相關聯的安全群組，其名稱符合模式 `rdsproxy-lambda-` *n*。 符合模式的安全群組尚未修改。此安全群組具有傳入和傳出規則，其中包含資料庫叢集和 Lambda 函數的 VPC 安全群組。	RDS action: create new security groups

目前 RDS 安全群組組態	目前 Lambda 安全群組組態	目前代理安全群組組態	RDS 動作
以下任一種條件均適用： • 沒有與資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-lambda-` *n*，或者如果相關聯代理的 `TargetHealth` 是 `AVAILABLE` 。 • 有一或多個與資料庫叢集相關聯的安全群組，其名稱符合模式 `rds-lambda-` *n*，或者如果相關聯代理的 `TargetHealth` 是 `AVAILABLE` 。不過，Amazon RDS 無法使用這些安全群組中的任一個與 Lambda 函數或代理連線。 Amazon RDS 無法使用沒有一個傳入規則 (使用 Lambda 函數的 VPC 安全群組或代理做為來源) 的安全群組。Amazon RDS 也無法使用經過修改的安全群組。	以下任一種條件均適用： • 沒有與 Lambda 函數相關聯的安全群組，其名稱符合模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n*。 • 有一或多個與 Lambda 函數相關聯的安全群組，其名稱符合模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n*。不過，Amazon RDS 無法使用這些安全群組中的任一個與資料庫資料庫叢集連線。 Amazon RDS 無法使用沒有傳出規則的安全群組，並將具有資料庫叢集或代理的 VPC 安全群組做為來源。Amazon RDS 也無法使用經過修改的安全群組。	以下任一種條件均適用： • 沒有與代理相關聯的安全群組，其名稱符合模式 `rdsproxy-lambda-` *n*。 • 有一或多個與代理相關聯的安全群組，其名稱符合 `rdsproxy-lambda-` *n*。不過，Amazon RDS 無法使用這些安全群組中的任一個與資料庫叢集或 Lambda 函數連線。 Amazon RDS 無法使用沒有傳入和傳出規則 (包含資料庫叢集和 Lambda 函數的 VPC 安全群組) 的安全群組。Amazon RDS 也無法使用經過修改的安全群組。	RDS action: create new security groups

RDS 動作：建立新的安全群組

Amazon RDS 會採取下列動作：

- 建立符合模式的新安全群組 `rds-lambda-`*n* 或者 `rds-rdsproxy-`*n* (如果您選擇使用 RDS Proxy)。此安全群組具有一個傳入規則，其具有 Lambda 函數或代理的 VPC 安全群組做為來源。與資料庫叢集關聯的此一安全群組可讓函數或代理存取資料庫叢集。
- 建立符合模式 `lambda-rds-`*n* 或 `lambda-rdsproxy-`*n* 的新安全群組。此安全群組有一個輸出規則，其中的 VPC 安全群組資料庫叢集或代理做為目的地。此安全群組與 Lambda 函數相關聯，並允許函數將流量傳送至資料庫叢集或通過代理傳送流量。
- 建立符合模式 `rdsproxy-lambda-`*n* 的新安全群組。此安全群組具有傳入和傳出規則，其中包含資料庫叢集和 Lambda 函數的 VPC 安全群組的傳入和傳出規則。

RDS 動作：與 Lambda 相關聯的安全群組

Amazon RDS 將有效、現有的 Lambda 安全群組與 Lambda 函數建立關聯。此安全群組允許函數將流量傳送至資料庫叢集或通過代理傳送流量。

自動連線 Lambda 函數和 Aurora 資料庫叢集

您可以使用 Amazon RDS 主控台將 Lambda 函數自動連線到您的資料庫叢集。簡化資源之間建立連線的過程。

您也可以使用 RDS Proxy 將代理包含在連線中。Lambda 函數經常進行短暫的資料庫連線，這些連線將受益於 RDS Proxy 提供的連線集區。您還可以使用 Lambda 函數中已有的任何 IAM 身分驗證，而不是在 Lambda 應用程式的程式碼中管理資料庫憑證。

您可以使用設定 Lambda 連線頁面，將現有的資料庫叢集連線至新的和現有的 Lambda 函數。設定程序會自動為您設定所需的安全群組。

在設定 Lambda 函數和資料庫叢集之間的連線之前，請確定：

- 你的 Lambda 函數和資料庫叢集在相同 VPC 中。
- 您擁有正確的使用者帳戶權限。有關需求的詳細資訊，請參閱 使用 Lambda 函數自動連線概觀。

如果您在設定連線之後變更安全群組，這些變更可能會影響 Lambda 函數與資料庫叢集之間的連線。

> **Note**
>
> 您只能在 中自動設定資料庫叢集與 Lambda 函數之間的連線 AWS 管理主控台。若要連線一個 Lambda 函數，資料庫叢集中的所有執行個體必須處於可用狀態。

若要自動連線 Lambda 函數和資料庫叢集

<result>

在您確認設定之後，Amazon RDS 會開始連線 Lambda 函數、RDS Proxy (如果您使用代理) 程序，以及資料庫叢集。主控台會顯示連線詳細資訊對話方塊，此對話方塊會列出允許資源之間連線的安全群組變更。
</result>

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇資料庫，然後選擇您要將其連線至 Lambda 函數的資料庫叢集。
3. 針對動作，選擇設定 Lambda 連線。
4. 在設定 Lambda 連線頁面中的選取 Lambda 函數，執行下列任一操作：
   - 如果您在相同的 VPC 中擁有現有 Lambda 函數資料庫叢集，選擇選擇現有函數，然後選擇函數。
   - 如果您在同一個 VPC 中沒有 Lambda 函數，請選擇建立新函數，然後輸入函數名稱。預設執行期會設定為 Nodejs.18。完成連線設定後，您可以在 Lambda 主控台中修改新 Lambda 函數的設定。
5. (選用) 在 RDS Proxy 中，選取使用 RDS Proxy 連線，然後執行下列任一項操作：
   - 若您要使用現有的代理，請選擇選擇現有的代理，然後選擇代理。
   - 如果您沒有代理，並且您希望 Amazon RDS 自動為您建立代理，請選擇建立新的代理。然後，為資料庫憑證，執行下列任一操作：
     a. 選擇資料庫使用者名稱和密碼，然後為您的資料庫叢集輸入使用者名稱和密碼。
     b. 選擇 Secrets Manager 密碼。然後，針對選取秘密，選擇 AWS Secrets Manager 秘密。如果您沒有 Secrets Manager 密碼，請選擇建立新的 Secrets Manager 密碼，以建立新的密碼。建立密碼之後，針對選取密碼，選擇新的密碼。

建立新的代理之後，請選擇選擇現有的代理，然後選擇代理。請注意，您的代理可能需要一些時間才能用於連線。

6. (選用) 展開連線摘要並為您的資源驗證反白顯示的更新。
7. 選擇 Set up (設定)。

檢視已連線的運算資源

您可以使用 AWS 管理主控台 檢視連線至資料庫叢集的 Lambda 函數。顯示的資源包括 Amazon RDS 自動設定的運算資源連線。

列出的運算資源不含手動連線至資料庫叢集。例如，您可以透過將規則新增至與資料庫關聯的 VPC 安全群組，來允許運算資源手動存取您的資料庫叢集。

若要讓主控台列出 Lambda 函數，必須符合下列條件：

- 與運算資源相關聯的安全群組，其名稱符合模式 `lambda-rds-`*n* 或 `lambda-rdsproxy-`*n* (其中 *n* 是數字)。
- 與運算資源相關聯的安全群組具有傳出規則，其連線埠範圍設為資料庫叢集或關聯代理所使用的連線埠。傳出規則的目的地必須設定為與資料庫叢集或關聯代理相關聯的安全群組。
- 如果組態包含代理，則附加到資料庫相關聯代理的安全群組，名稱符合模式 `rdsproxy-lambda-`*n* (其中 *n* 是數字)。
- 與函數相關聯的安全群組具有傳出規則，其連線埠設定為資料庫叢集或相關的代理所使用的連線埠。目的地必須設定為與資料庫叢集或關聯代理相關聯的安全群組。

若要檢視自動連線至資料庫叢集的運算資源

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇資料庫，然後選擇資料庫叢集。
3. 在連線與安全性索引標籤中，檢視已連線的運算資源中的運算資源。

修改 Amazon Aurora 資料庫叢集

您可以變更資料庫叢集的設定，以達成各種任務，像是變更其備份保留期或資料庫連線埠。您也可以修改資料庫叢集中的資料庫執行個體，以達成各種任務，像是變更其資料庫執行個體類別或啟用 Performance Insights (效能洞見)。本主題將引導您修改 Aurora 資料庫叢集和其資料庫執行個體，並說明兩者的設定。

建議您先在測試資料庫叢集或資料庫執行個體上測試任何變更，然後再修改生產資料庫叢集或資料庫執行個體，以便充分了解每一項變更的影響。這在升級資料庫版本時特別重要。

主題

- 使用主控台、CLI 和 API 修改資料庫叢集
- 修改資料庫叢集中的資料庫執行個體
- 變更資料庫主要使用者的密碼
- Amazon Aurora的設定
- Amazon Aurora 資料庫叢集不適用的設定
- Amazon Aurora 資料庫執行個體不適用的設定

使用主控台、CLI 和 API 修改資料庫叢集

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 修改資料庫叢集。

> **Note**
>
> 大多數修改都可以立即套用，或下次維護時段時套用。部分修改 (例如開啟刪除保護) 會立即套用，無論您何時選擇套用這些修改。
> 變更 AWS 管理主控台 中的主要密碼一律會立即套用。
> 如果您使用 SSL 端點並變更資料庫叢集識別符，請停止並重新啟動資料庫叢集，以更新 SSL 端點。如需更多詳細資訊，請參閱 停用和啟動 Amazon Aurora 資料庫叢集。

主控台

修改資料庫叢集

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。

2. 在導覽窗格中選擇 Databases (資料庫)，然後選取您要修改的資料庫叢集。
3. 選擇 Modify (修改)。Modify DB cluster (修改資料庫叢集) 頁面隨即出現。
4. 變更您要的任何設定。如需每項設定的相關資訊，請參閱 Amazon Aurora的設定。

> **Note**
>
> 在 AWS 管理主控台 中，某些執行個體層級變更僅會套用到目前的資料庫執行個體，而其他的變更則會套用到整個資料庫叢集。至於設定會套用至資料庫執行個體或資料庫叢集，詳細資訊請參閱Amazon Aurora的設定 中的設定範圍。若要變更在 AWS 管理主控台中執行個體層級修改整個資料庫叢集的任一設定，請遵循修改資料庫叢集中的資料庫執行個體中的說明。

5. 當所有變更都如您所願時，請選擇 Continue (繼續) 並查看修改摘要。
6. 若要立即套用變更，請選取 Apply immediately (立即套用)。
7. 在確認頁面上，檢閱您的變更。如果都正確，請選擇 Modify cluster (修改叢集) 以儲存您的變更。

   或者，選擇 Back (上一步) 以編輯變更，或是選擇 Cancel (取消) 以取消變更。

AWS CLI

若要使用 AWS CLI 修改資料庫叢集，呼叫 modify-db-cluster 命令。指定資料庫叢集識別符，以及您要修改設定的值。如需每項設定的相關資訊，請參閱 Amazon Aurora的設定。

> **Note**
>
> 部分設定僅會套用至資料庫執行個體。若要變更這些設定，請遵循修改資料庫叢集中的資料庫執行個體中的說明。

Example

以下命令會修改 `mydbcluster`，將備份保留期設為 1 週 (7 天)。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
```

```
    --backup-retention-period 7
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --backup-retention-period 7
```

RDS API

若要使用 Amazon RDS API 修改資料庫叢集，呼叫 ModifyDBCluster 操作。指定資料庫叢集識別符，以及您要修改設定的值。如需每項參數的相關資訊，請參閱 Amazon Aurora的設定。

> **Note**
>
> 部分設定僅會套用至資料庫執行個體。若要變更這些設定，請遵循修改資料庫叢集中的資料庫執行個體中的說明。

修改資料庫叢集中的資料庫執行個體

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 修改資料庫叢集中的資料庫執行個體。

修改資料庫執行個體時，您可以立即套用變更。若要立即套用變更，請在 AWS 管理主控台 中選取 Apply Immediately (立即套用) 選項，呼叫 AWS CLI 時使用 `--apply-immediately` 參數，或是在使用 Amazon RDS API 時將 `ApplyImmediately` 參數設為 `true`。

如果您不選擇立即套用變更，變更會延遲到下一個維護時段為止。在下一個維護時段期間，會套用任何這些延遲變更。如果您選擇立即套用變更，則會套用您的新變更以及先前延遲的變更。

若要查看下一個維護時段待定的修改，請使用 describe-db-clusters AWS CLI 命令並檢查 `PendingModifiedValues` 欄位。

> **Important**
>
> 如果有任何延遲的修改需要停機才能執行，則Apply immediately (立即套用) 選項可能會導致資料庫執行個體未預期的停機。資料庫叢集中的其他資料庫執行個體沒有任何停機時間。`describe-pending-maintenance-actions` CLI 命令的輸出中不會列出您延遲的修改。維護動作只包括您為下一個維護時段排程的系統升級。

主控台

修改資料庫叢集中的資料庫執行個體

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中選擇 Databases (資料庫)，然後選取您要修改的資料庫執行個體。
3. 在 Actions (動作) 中，選擇 Modify (修改)。Modify DB instance (修改資料庫執行個體) 頁面隨即出現。
4. 變更您要的任何設定。如需每項設定的相關資訊，請參閱 Amazon Aurora的設定。

   > **Note**
   >
   > 有些設定會套用到整個資料庫叢集，且必須在叢集層級進行變更。若要變更這些設定，請遵循使用主控台、CLI 和 API 修改資料庫叢集中的說明。
   > 在 AWS 管理主控台 中，某些執行個體層級變更僅會套用到目前的資料庫執行個體，而其他的變更則會套用到整個資料庫叢集。至於設定會套用至資料庫執行個體或資料庫叢集，詳細資訊請參閱Amazon Aurora的設定 中的設定範圍。

5. 當所有變更都如您所願時，請選擇 Continue (繼續) 並查看修改摘要。
6. 若要立即套用變更，請選取 Apply immediately (立即套用)。
7. 在確認頁面上，檢閱您的變更。如果都正確，請選擇 Modify DB instance (修改資料庫執行個體) 以儲存您的變更。

   或者，選擇 Back (上一步) 以編輯變更，或是選擇 Cancel (取消) 以取消變更。

AWS CLI

若要使用 AWS CLI 修改資料庫叢集中的資料庫執行個體，請呼叫 modify-db-instance 命令。指定資料庫執行個體識別符，以及您要修改設定的值。如需每項參數的相關資訊，請參閱 Amazon Aurora的設定。

> **Note**
>
> 部分設定會套用至整個資料庫叢集。若要變更這些設定，請遵循使用主控台、CLI 和 API 修改資料庫叢集中的說明。

Example

以下程式碼會將資料庫執行個體類別設定為 mydbinstance，藉此修改 db.r4.xlarge。使用 --no-apply-immediately，會在下一次維護時段期間由系統套用變更。使用 --apply-immediately 可立即套用變更。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --db-instance-class db.r4.xlarge \
    --no-apply-immediately
```

在 Windows 中：

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --db-instance-class db.r4.xlarge ^
    --no-apply-immediately
```

RDS API

若要使用 Amazon RDS API 來修改資料庫執行個體，請呼叫 ModifyDBInstance 操作。指定資料庫執行個體識別符，以及您要修改設定的值。如需每項參數的相關資訊，請參閱 Amazon Aurora的設定。

> **Note**
>
> 部分設定會套用至整個資料庫叢集。若要變更這些設定，請遵循使用主控台、CLI 和 API 修改資料庫叢集中的說明。

變更資料庫主要使用者的密碼

您可使用 AWS 管理主控台 或 AWS CLI 來變更主要使用者的密碼。

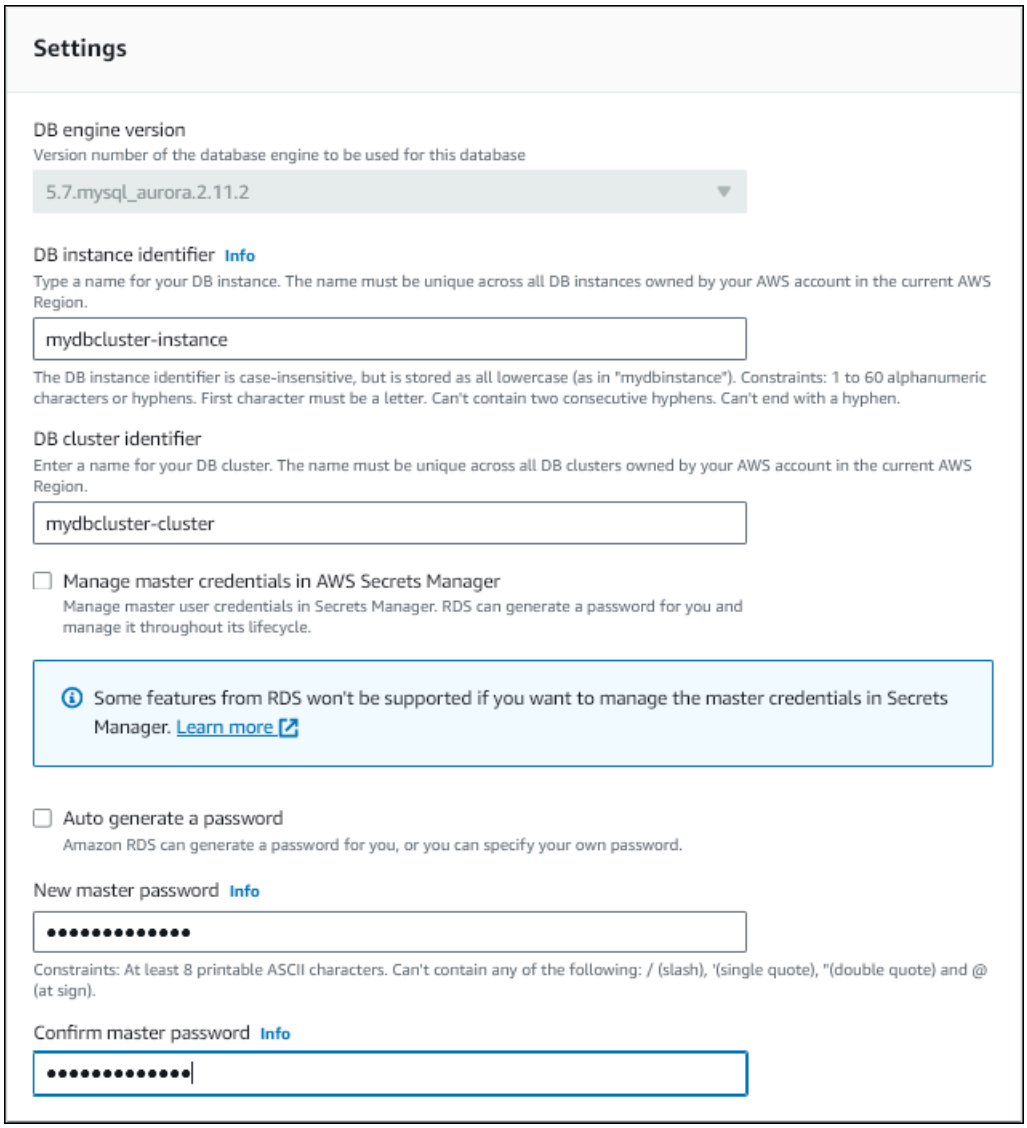
主控台

您可以使用 AWS 管理主控台 修改寫入器資料庫執行個體，以變更主要使用者密碼。

若要變更主要使用者的密碼

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中選擇 Databases (資料庫)，然後選取您要修改的資料庫執行個體。
3. 在 Actions (動作) 中，選擇 Modify (修改)。

   Modify DB instance (修改資料庫執行個體) 頁面隨即出現。
4. 輸入新的主要密碼。
5. 針對確認主要密碼，輸入相同的新密碼。



6. 選擇 Continue (繼續)，並檢查修改的摘要。

> **Note**
>
> 密碼變更一律會立即套用。

7. 在確認頁面上，選擇 Modify DB instance (修改資料庫執行個體)。

CLI

您可以使用 AWS CLI 呼叫 modify-db-cluster 命令來變更主要使用者密碼。指定資料庫叢集識別符和新密碼，如下列範例所示。

您不需要指定 `--apply-immediately|--no-apply-immediately`，因為密碼變更一律會立即套用。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --master-user-password mynewpassword
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --master-user-password mynewpassword
```

Amazon Aurora的設定

下表所述的詳細資訊說明您可以修改的設定、修改設定的方法、設定的範圍。範圍決定了設定是否會套用至整個資料庫叢集，或是僅為特定資料庫執行個體而設定。

> **Note**
>
> 若您正在修改 Aurora Serverless v1 或 Aurora Serverless v2 資料庫叢集，則可使用其他設定。如需這些設定的詳細資訊，請參閱 修改 Aurora Serverless v1 資料庫叢集 和 管理 Aurora Serverless v2 資料庫叢集。
>
> 由於 Aurora Serverless v1 與 Aurora Serverless v2 限制之故，無法使用某些設定。如需詳細資訊，請參閱Aurora Serverless v1 的限制及Aurora Serverless v2 的要求與限制。

設定和說明	方法	範圍	停機時間備註
Auto minor version upgrade (自動次要版本升級) 是否要讓資料庫執行個體自動接收可用的建議次要引擎版本升級。只會在排定的維護時段安裝升級。 如需引擎更新的詳細資訊，請參閱 Amazon Aurora PostgreSQL 的資料庫引擎更新和 Amazon Aurora MySQL 的資料庫引擎更新。如需有關 Aurora MySQL 的 Auto minor version upgrade (自動次要版本升級) 設定的詳細資訊，請參閱 啟用次要 Aurora MySQL 版本之間的自動升級。	**Note** 此設定預設為啟用。針對每個新叢集，請根據此設定的重要性、預期壽命上限，以及每次升級之後所執行的驗證測試量，為其選擇適當的值。 變更此設定時，請對 Aurora 叢集中的每個資料庫執行個體執行此修改。如果叢集中的任何資料庫執行個體已關閉此設定，則叢集不會自動升級。 使用 AWS 管理主控台：修改資料庫叢集中的資料庫執行個體。 使用 AWS CLI，執行 modify-db-instance，然後設定 `--auto-minor-version-upgrade|--no-auto-`	整個資料庫叢集	進行此變更時，不會發生停機。當 Aurora 套用自動升級時，將會在未來的維護期間發生中斷。

設定和說明	方法	範圍	停機時間備註
	`minor-version-upgrade` 選項。 使用 RDS API，呼叫 ModifyDBInstance，並設定 `AutoMinorVersionUpgrade` 參數。		
Backup retention period (備份保留期間) 自動備份保留的天數。最小值為 1。 如需更多詳細資訊，請參閱 備份。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 modify-db-cluster 並設定 `--backup-retention-period` 選項。 使用 RDS API，呼叫 ModifyDBCluster，並設定 `BackupRetentionPeriod` 參數。	整個資料庫叢集	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
備份時段 (開始時間) 進行自動資料庫備份的時間範圍。備份時段是國際標準時間 (UTC) 格式的開始時間和以小時為單位的持續時間。 Aurora 備份是持續且增量的，但是備份時段是用來建立在備份保留期間內保留的每日系統備份。您可以複製該檔案，在超出保留期間後仍然保留。 資料庫叢集的維護時段和備份時段不能重疊。 如需更多詳細資訊，請參閱 備份時段。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 modify-db-cluster 並設定 `--preferred-backup-window` 選項。 使用 RDS API，呼叫 ModifyDBCluster，並設定 `PreferredBackupWindow` 參數。	整個資料庫叢集。	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
容量設定 Aurora Serverless v1 資料庫叢集的擴展屬性。您只能在 `serverless` 資料庫引擎模式下修改資料庫叢集的擴展屬性。 如需 Aurora Serverless v1 的相關資訊，請參閱 使用 Amazon Aurora Serverless v1。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 modify-db-cluster 並設定 `--scaling-configuration` 選項。 使用 RDS API，呼叫 ModifyDBCluster，並設定 `ScalingConfiguration` 參數。	整個資料庫叢集	進行此變更時，不會發生停機。 系統會立即進行變更。此設定會忽略立即套用設定。
憑證授權單位 資料庫執行個體所使用之伺服器憑證的憑證授權單位 (CA)。	使用 AWS 管理主控台：修改資料庫叢集中的資料庫執行個體。 使用 AWS CLI，執行 modify-db-instance，然後設定 `--ca-certificate-identifier` 選項。 使用 RDS API，呼叫 ModifyDBInstance，並設定 `CACertificateIdentifier` 參數。	僅限指定的資料庫執行個體	只有在資料庫引擎不支援無需重新啟動即可輪換時，才會發生中斷。您可以使用 describe-db-engine-versions AWS CLI 命令，來判斷資料庫引擎是否支援無需重新啟動即可輪換。

設定和說明	方法	範圍	停機時間備註
叢集儲存組態 資料庫叢集的儲存類型：Aurora I/O-Optimized 或 Aurora Standard。 如需更多詳細資訊，請參閱 Amazon Aurora 資料庫叢集的儲存組態。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 modify-db-cluster 並設定 `--storage-type` 選項。 使用 RDS API，呼叫 ModifyDBCluster，並設定 `StorageType` 參數。	整個資料庫叢集	使用 Optimized Reads 執行個體類別變更 Aurora PostgreSQL 資料庫叢集的儲存類型會導致中斷。變更具有其他執行個體類別類型的叢集儲存類型時，不會發生這種情況。如需資料庫執行個體類別類型的詳細資訊，請參閱 資料庫執行個體類別的類型。

設定和說明	方法	範圍	停機時間備註
Copy tags to snapshots (將標籤複製到快照) 選取以指定讓針對此資料庫叢集定義的標籤，複製到從這個資料庫叢集建立的資料庫快照中。如需更多詳細資訊，請參閱 標記 Amazon Aurora 和 Amazon RDS 資源。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-cluster.html，並設定 `--copy-tags-to-snapshot` 或 `--no-copy-tags-to-snapshot` 選項。 使用 RDS API，呼叫 https://docs.aws.amazon.com/AmazonRDS/latest/APIReference/API_ModifyDBCluster.html，並設定 `CopyTagsToSnapshot` 參數。	整個資料庫叢集	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
Data API (資料 API) 您可以使用以 Web 服務為基礎的應用程式 (包括 AWS Lambda 和 AWS AppSync) 來存取 Aurora Serverless v1。 此設定僅適用於 Aurora Serverless v1 資料庫叢集。 如需更多詳細資訊，請參閱 使用 Amazon RDS 資料 API。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 `modify-db-cluster`，並設定 `--enable-http-endpoint` 選項。 使用 RDS API，呼叫 `ModifyDBCluster`，並設定 `EnableHttpEndpoint` 參數。	整個資料庫叢集	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
Database authentication (資料庫身分驗證 您想要使用的資料庫身分驗證。 適用於 MySQL： • 選擇 Password authentication (密碼身分驗證)，僅使用資料庫密碼驗證資料庫使用者。 • 選擇 Password and IAM database authentication (密碼和 IAM 資料庫身分驗證)，透過 IAM 使用者和角色搭配資料庫密碼和使用者登入資料對資料庫使用者進行身分驗證。如需更多詳細資訊，請參閱 IAM 資料庫身分驗證。 適用於 PostgreSQL： • 選擇 IAM database authentication (IAM 資料庫身分驗證)，透過使用者和角色搭配資料庫密碼和使用者登入資料對資料庫使用者進行	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 modify-db-cluster 並設定下列選項： • 對於 IAM 身分驗證，請設定 `--enable-iam-database-authentication|--no-enable-iam-database-authentication` 選項。 • 對於 Kerberos 身分驗證，請設定 `--domain` 和 `--domain-iam-role-name` 選項。 使用 RDS API，呼叫 ModifyDBCluster 並設定下列參數： • 對於 IAM 身分驗證，請設定 `EnableIAMDatabaseA`	整個資料庫叢集	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
身分驗證。如需更多詳細資訊，請參閱 IAM 資料庫身分驗證。 • 選擇 Kerberos authentication (身分驗證)，以使用 Kerberos 身分驗證對資料庫密碼和使用者登入資料進行身分驗證。如需更多詳細資訊，請參閱 搭配 Aurora PostgreSQL 使用 Kerberos 身分驗證。	uthentica tion 參數。 • 對於 Kerberos 身分驗證，請設定 Domain 和 DomainIAM RoleName 參數。		
Database port (資料庫連線埠) 您要用來存取資料庫叢集的連線埠。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 modify-db-cluster，並設定 --port 選項。 使用 RDS API，呼叫 ModifyDBCluster，並設定 Port 參數。	整個資料庫叢集	進行此變更時，會發生停機。資料庫叢集中的所有資料庫執行個體皆會立即重新開機。

設定和說明	方法	範圍	停機時間備註
DB cluster identifier (資料庫叢集識別符) 資料庫叢集識別符 此值會以小寫字母字串的形式儲存。 當您變更資料庫叢集識別符時，資料庫叢集端點會變更。資料庫叢集中資料庫執行個體的端點不會變更。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 `modify-db-cluster`，並設定 `--new-db-cluster-identifier` 選項。 使用 RDS API，呼叫 `ModifyDBCluster`，並設定 `NewDBClusterIdentifier` 參數。	整個資料庫叢集	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
DB cluster parameter group (資料庫叢集參數群組) 要與資料庫叢集建立關聯的資料庫叢集參數群組。 如需更多詳細資訊，請參閱 Amazon Aurora 的參數群組。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 `modify-db-cluster`，並設定 `--db-cluster-parameter-group-name` 選項。 使用 RDS API，呼叫 `ModifyDBCluster`，並設定 `DBClusterParameterGroupName` 參數。	整個資料庫叢集	進行此變更時，不會發生停機。當您變更參數群組時，某些參數的變更會立即套用到資料庫叢集的資料庫執行個體，而不需要重新啟動。其他參數的變更只有在資料庫叢集的資料庫執行個體重新啟動之後才會套用。
DB instance class (資料庫執行個體類別) 您要使用的資料庫執行個體類別。 如需更多詳細資訊，請參閱 Amazon Aurora 資料庫執行個體類別。	使用 AWS 管理主控台：修改資料庫叢集中的資料庫執行個體。 使用 AWS CLI，執行 `modify-db-instance`，並設定 `--db-instance-class` 選項。 使用 RDS API，呼叫 `ModifyDBInstance`，並設定 `DBInstanceClass` 參數。	僅限指定的資料庫執行個體	進行此變更時，會發生停機。

設定和說明	方法	範圍	停機時間備註
DB instance identifier (資料庫執行個體識別符)： 資料庫執行個體識別符。此值會以小寫字母字串的形式儲存。	使用 AWS 管理主控台：修改資料庫叢集中的資料庫執行個體。 使用 AWS CLI，執行 `modify-db-instance`，並設定 `--new-db-instance-identifier` 選項。 使用 RDS API，呼叫 `ModifyDBInstance`，並設定 `NewDBInstanceIdentifier` 參數。	僅限指定的資料庫執行個體	進行此變更時，會發生停機。 RDS 會重新啟動資料庫執行個體以更新下列項目： • Aurora MySQL – `information_schema.replica_host_status` 資料表中的資料 `SERVER_ID` 欄 • Aurora PostgreSQL – `aurora_replica_status()` 函數中的資料 `server_id` 欄

設定和說明	方法	範圍	停機時間備註
DB parameter group (資料庫參數群組) 要與資料庫執行個體建立關聯的資料庫參數群組。 如需更多詳細資訊，請參閱 Amazon Aurora 的參數群組。	使用 AWS 管理主控台：修改資料庫叢集中的資料庫執行個體。 使用 AWS CLI，執行 `modify-db-instance` ，並設定 `--db-parameter-group-name` 選項。 使用 RDS API，呼叫 `ModifyDBInstance` ，並設定 `DBParameterGroupName` 參數。	僅限指定的資料庫執行個體	進行此變更時，不會發生停機。 當您建立新資料庫參數群組與資料庫執行個體的關聯時，只有在資料庫執行個體重新開機之後，才會套用修改過的靜態參數和動態參數。不過，如果您在將資料庫參數群組與資料庫執行個體建立關聯之後修改該群組中的動態參數，則會立即套用這些變更，而不需重新開機。 如需詳細資訊，請參閱Amazon Aurora 的參數群組及重新啟動 Amazon Aurora 資料庫叢集或 Amazon Aurora 資料庫執行個體。

設定和說明	方法	範圍	停機時間備註
刪除保護 啟用刪除保護可避免您的資料庫叢集遭意外刪除。如需更多詳細資訊，請參閱 Aurora 叢集的刪除保護。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 modify-db-cluster，並設定 --deletion-protection|--no-deletion-protection 選項。 使用 RDS API，呼叫 ModifyDBCluster，並設定 DeletionProtection 參數。	整個資料庫叢集	進行此變更時，不會發生停機。
引擎版本 您要使用的資料庫引擎的版本。升級生產資料庫叢集之前，建議您先在測試資料庫叢集上測試升級程序，以確認其持續時間及驗證您的應用程式。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 modify-db-cluster，並設定 --engine-version 選項。 使用 RDS API，呼叫 ModifyDBCluster，並設定 EngineVersion 參數。	整個資料庫叢集	進行此變更時，會發生停機。

設定和說明	方法	範圍	停機時間備註
Enhanced monitoring (增強型監控) Enable enhanced monitoring (啟用增強型監控)，以針對資料庫執行個體執行所在的作業系統即時收集指標。 如需更多詳細資訊，請參閱 使用增強型監控來監控作業系統指標。	使用 AWS 管理主控台：修改資料庫叢集中的資料庫執行個體。 使用 AWS CLI，執行 `modify-db-instance`，並設定 `--monitoring-role-arn` 和 `--monitoring-interval` 選項。 使用 RDS API，呼叫 `ModifyDBInstance`，並設定 `MonitoringRoleArn` 和 `MonitoringInterval` 參數。	僅限指定的資料庫執行個體	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
Log exports (日誌匯出) 選取要發佈到 Amazon CloudWatch Logs 的日誌類型。 如需更多詳細資訊，請參閱 Aurora MySQL 資料庫日誌檔案。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 `modify-db-cluster`，並設定 `--cloudwatch-logs-export-configuration` 選項。 使用 RDS API，呼叫 `ModifyDBCluster`，並設定 `CloudwatchLogsExportConfiguration` 參數。	整個資料庫叢集	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
Maintenance window (維護時段) 進行系統維護的時間範圍。系統維護包括升級 (如適用)。維護時段是國際標準時間 (UTC) 格式的開始時間和以小時為單位的持續時間。 如果您將時段設定為目前時間，則目前時間與結束時段之間必須至少間隔 30 分鐘，以確保能套用任何擱置的變更。 您可以為資料庫叢集和資料庫叢集中的每個資料庫執行個體分開設定維護時段。當修改的範圍是整個資料庫叢集時，修改會在資料庫叢集的維護時段期間進行。當修改的範圍是資料庫執行個體時，修改會在該資料庫執行個體的維護時段期間進行。 資料庫叢集的維護時段和備份時段不能重疊。	使用 AWS 管理主控台 變更資料庫叢集的維護時段：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS 管理主控台變更資料庫執行個體的維護時段：修改資料庫叢集中的資料庫執行個體。 若要使用 AWS CLI 變更資料庫叢集的維護時段，請執行 `modify-db-cluster`，並設定 `--preferred-maintenance-window` 選項。 若要使用 AWS CLI 變更資料庫執行個體的維護時段，請執行 `modify-db-instance`，並設定 `--preferred-maintenance-window` 選項。 若要使用 RDS API 變更資料庫叢集的維護時段，請呼叫 `ModifyDBCluster`，並設定 `Preferred`	整個資料庫叢集或單一資料庫執行個體	如果您有一個或多個會導致停機的擱置動作，並將維護時段變更為涵蓋目前時間，則系統會立即套用這些擱置動作，最終導致停機。

設定和說明	方法	範圍	停機時間備註
如需更多詳細資訊，請參閱 Amazon RDS 維護時段。	PreferredMaintenanceWindow 參數。 若要使用 RDS API 變更資料庫執行個體的維護時段，請呼叫 ModifyDBInstance，並設定 PreferredMaintenanceWindow 參數。		

設定和說明	方法	範圍	停機時間備註
管理 AWS Secrets Manager 中的主要憑證 選取 Manage master credentials in AWS Secrets Manager (管理 AWS Secrets Manager 中的主要憑證) 以秘密管理 Secrets Manager 中的主要使用者密碼。 選擇性地選擇要用來保護機密的 KMS 金鑰。從您帳戶中的 KMS 金鑰進行選擇，或輸入來自不同帳戶的金鑰。 如需更多詳細資訊，請參閱 使用 Amazon Aurora 和 進行密碼管理 AWS Secrets Manager。 如果 Aurora 已在管理資料庫叢集的主要使用者密碼，您可以選擇 Rotate secret immediately (立即輪換秘密) 來輪換主要使用者密碼。 如需更多詳細資訊，請參閱 使用 Amazon	使用 AWS 管理主控台：修改資料庫叢集中的資料庫執行個體。 使用 AWS CLI，執行 `modify-db-cluster`，並設定 `--manage-master-user-password | --no-manage-master-user-password` 和 `--master-user-secret-kms-key-id` 選項。若要立即輪換主要使用者密碼，請設定 `--rotate-master-user-password` 選項。 使用 RDS API，呼叫 `ModifyDBCluster`，並設定 `ManageMasterUserPassword` 和 `MasterUserSecretKmsKeyId` 參數。若要立即輪換主要使用者密碼，請將 `RotateMasterUserPa`	整個資料庫叢集	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
Aurora 和 進行密碼管理 AWS Secrets Manager。	ssword 參數設為 true。		

設定和說明	方法	範圍	停機時間備註
Network type (網路類型) 資料庫叢集支援的 IP 定址通訊協定。 IPv4 指定資源只能透過網際網路通訊協定第 4 版 (IPv4) 定址通訊協定與資料庫叢集通訊。 Dual-stack mode (雙堆疊模式) 指定資源可透過 IPv4、IPv6 或兩者與資料庫叢集通訊。如果您有任何資源必須透過 IPv6 定址通訊協定與您的資料庫叢集通訊，請使用雙堆疊模式。若要使用雙堆疊模式，請確定至少有兩個跨越兩個可用區域的子網路，同時支援 IPv4 和 IPv6 網路通訊協定。此外，請確保將 IPv6 CIDR 區塊與您指定的資料庫子網路群組中的所有子網路相關聯。 如需更多詳細資訊，請參閱 Amazon Aurora IP 定址。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 `modify-db-cluster`，並設定 `--network-type` 選項。 使用 RDS API，呼叫 `ModifyDBCluster`，並設定 `NetworkType` 參數。	整個資料庫叢集	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
新主要使用者密碼 主要使用者的密碼。 • 針對 Aurora MySQL，密碼必須包含 8–41 個可印刷的 ASCII 字元。 • 針對 Aurora PostgreSQL，密碼必須包含 8 – 99 個可印刷的 ASCII 字元。 • 其中不能包含 /、"、@ 或空格。	使用 AWS 管理主控台：修改資料庫叢集中的資料庫執行個體。 使用 AWS CLI，執行 `modify-db-cluster`，並設定 `--master-user-password` 選項。 使用 RDS API，呼叫 `ModifyDBCluster`，並設定 `MasterUserPassword` 參數。	整個資料庫叢集	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
Performance Insights (績效詳情) 是否要啟用 Performance Insights，這個工具會監控您的資料庫執行個體負載，讓您可分析資料庫效能並對其進行故障排除。 如需更多詳細資訊，請參閱 在 Amazon Aurora 上使用績效詳情監控資料庫負載。	使用 AWS 管理主控台：修改資料庫叢集中的資料庫執行個體。 使用 AWS CLI，執行 `modify-db-instance` ，並設定 `--enable-performance-insights|--no-enable-performance-insights` 選項。 使用 RDS API，呼叫 `ModifyDBInstance` ，並設定 `EnablePerformanceInsights` 參數。	僅限指定的資料庫執行個體	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
績效詳情 AWS KMS key 用於績效詳情資料加密的 AWS KMS key 識別碼。KMS 金鑰識別碼是 KMS 金鑰的 Amazon 資源名稱 (ARN)、金鑰識別碼或金鑰別名。 如需更多詳細資訊，請參閱 開啟和關閉 Aurora 的 Performance Insights。	使用 AWS 管理主控台：修改資料庫叢集中的資料庫執行個體。 使用 AWS CLI，執行 `modify-db-instance`，並設定 `--performance-insights-kms-key-id` 選項。 使用 RDS API，呼叫 `ModifyDBInstance`，並設定 `PerformanceInsightsKMSKeyId` 參數。	僅限指定的資料庫執行個體	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
Performance Insights 保留期間 Performance Insights 資料的保留時間，單位為天。保留設定為預設值 (7 天)。若要更長時間保留績效資料，請指定 1 - 24 個月。如需保留期間的詳細資訊，請參閱 Performance Insights 的定價和資料保留。 如需更多詳細資訊，請參閱 開啟和關閉 Aurora 的 Performance Insights。	使用 AWS 管理主控台：修改資料庫叢集中的資料庫執行個體。 使用 AWS CLI，執行 `modify-db-instance` ，並設定 `--performance-insights-retention-period` 選項。 使用 RDS API，呼叫 `ModifyDBInstance` ，並設定 `PerformanceInsightsRetentionPeriod` 參數。	僅限指定的資料庫執行個體	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
提升層 此值指定在現有主要執行個體失敗後，Aurora 複本提升為資料庫叢集主要執行個體的順序。 如需更多詳細資訊，請參閱 Aurora 資料庫叢集的容錯能力。	使用 AWS 管理主控台：修改資料庫叢集中的資料庫執行個體。 使用 AWS CLI，執行 `modify-db-instance`，並設定 `--promotion-tier` 選項。 使用 RDS API，呼叫 `ModifyDBInstance`，並設定 `PromotionTier` 參數。	僅限指定的資料庫執行個體	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
公用存取 Publicly accessible (可公開存取) 用來給予資料庫執行個體一個公有 IP 地址，這表示可在 VPC 外加以存取。資料庫執行個體也必須位於 VPC 的公有子網路中，才能公開存取。 Not publicly accessible (不可公開存取) 將使得資料庫執行個體只能從 VPC 內部存取。 如需詳細資訊，請參閱 在 VPC 中的網際網路中隱藏資料庫叢集。 若要從其 Amazon VPC 之外連線至資料庫執行個體，資料庫執行個體必須可公開存取、必須使用資料庫執行個體之安全群組的傳入規則授予存取權，且必須符合其他要求。如需詳細資訊，請參閱 無法連線至 Amazon RDS 資料庫執行個體。	使用 AWS 管理主控台：修改資料庫叢集中的資料庫執行個體。 使用 AWS CLI，執行 `modify-db-instance` ，並設定 `--publicly-accessible|--no-publicly-accessible` 選項。 使用 RDS API，呼叫 `ModifyDBInstance` ，並設定 `PubliclyAccessible` 參數。	僅限指定的資料庫執行個體	進行此變更時，不會發生停機。

設定和說明	方法	範圍	停機時間備註
如果您的資料庫執行個體無法公開存取，您也可以使用 AWS Site-to-Site VPN 連線或 Direct Connect 連線從私有網路存取。如需更多詳細資訊，請參閱 網際網路流量隱私權。			
Serverless v2 容量設定 Aurora Serverless v2 資料庫叢集的資料庫容量，以 Aurora 容量單位 (ACU) 為單位進行測量。 如需更多詳細資訊，請參閱 設定叢集的 Aurora Serverless v2 容量範圍。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 `modify-db-cluster`，並設定 `--serverless-v2-scaling-configuration` 選項。 使用 RDS API，呼叫 `ModifyDBCluster`，並設定 `ServerlessV2ScalingConfiguration` 參數。	整個資料庫叢集	進行此變更時，不會發生停機。 系統會立即進行變更。此設定會忽略立即套用設定。

設定和說明	方法	範圍	停機時間備註
安全群組 要與資料庫叢集建立關聯的安全群組。 如需更多詳細資訊，請參閱 使用安全群組控制存取。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 modify-db-cluster，並設定 --vpc-security-group-ids 選項。 使用 RDS API，呼叫 ModifyDBCluster，並設定 VpcSecurityGroupIds 參數。	整個資料庫叢集	進行此變更時，不會發生停機。
目標恢復時段 您希望資料庫叢集能夠恢復的時間量，單位為秒。此設定僅適用於 Aurora MySQL，且僅能用於恢復功能啟用時所建立的資料庫叢集。	使用 AWS 管理主控台：使用主控台、CLI 和 API 修改資料庫叢集。 使用 AWS CLI，執行 modify-db-cluster，並設定 --backtrack-window 選項。 使用 RDS API，呼叫 ModifyDBCluster，並設定 BacktrackWindow 參數。	整個資料庫叢集	進行此變更時，不會發生停機。

Amazon Aurora 資料庫叢集不適用的設定

AWS CLI 命令、modify-db-cluster 和 RDS API 操作 ModifyDBCluster 的下列設定不適用於 Amazon Aurora 資料庫叢集。

> **Note**
>
> 您無法使用 AWS 管理主控台 來修改 Aurora 資料庫叢集的這些設定。

AWS CLI 設定	RDS API 設定
`--allocated-storage`	`AllocatedStorage`
`--auto-minor-version-upgrade | --no-auto-minor-version-upgrade`	`AutoMinorVersionUpgrade`
`--db-cluster-instance-class`	`DBClusterInstanceClass`
`--enable-performance-insights | --no-enable-performance-insights`	`EnablePerformanceInsights`
`--iops`	`Iops`
`--monitoring-interval`	`MonitoringInterval`
`--monitoring-role-arn`	`MonitoringRoleArn`
`--option-group-name`	`OptionGroupName`
`--performance-insights-kms-key-id`	`PerformanceInsightsKMSKeyId`
`--performance-insights-retention-period`	`PerformanceInsightsRetentionPeriod`

Amazon Aurora 資料庫執行個體不適用的設定

AWS CLI 命令、`modify-db-instance` 和 RDS API 操作 `ModifyDBInstance` 的下列設定不適用於 Amazon Aurora 資料庫執行個體。

> **Note**
>
> 您無法使用 AWS 管理主控台 來修改 Aurora 資料庫執行個體的這些設定。

AWS CLI 設定	RDS API 設定
`--allocated-storage`	`AllocatedStorage`
`--allow-major-version-upgrade|--no-allow-major-version-upgrade`	`AllowMajorVersionUpgrade`
`--copy-tags-to-snapshot|--no-copy-tags-to-snapshot`	`CopyTagsToSnapshot`
`--domain`	`Domain`
`--db-security-groups`	`DBSecurityGroups`
`--db-subnet-group-name`	`DBSubnetGroupName`
`--domain-iam-role-name`	`DomainIAMRoleName`
`--multi-az|--no-multi-az`	`MultiAZ`
`--iops`	`Iops`
`--license-model`	`LicenseModel`
`--network-type`	`NetworkType`
`--option-group-name`	`OptionGroupName`
`--processor-features`	`ProcessorFeatures`

AWS CLI 設定	RDS API 設定
--storage-type	StorageType
--tde-credential-arn	TdeCredentialArn
--tde-credential-password	TdeCredentialPassword
--use-default-processor-features|--no-use-default-processor-features	UseDefaultProcessorFeatures

將 Aurora 複本新增至資料庫叢集

在含複寫的 Aurora 資料庫叢集中，有一個主要資料庫執行個體，以及最多 15 個 Aurora 複本。主要資料庫執行個體支援讀寫操作，並對叢集磁碟區執行所有資料修改。Aurora 複本連接到與主要資料庫執行個體相同的儲存磁碟區，僅支援讀取操作。您可以使用 Aurora，從主要資料庫執行個體中卸載讀取工作負載。如需詳細資訊，請參閱Aurora 複本。

Amazon Aurora 複本具有下列限制：

- 您無法為 Aurora Serverless v1 資料庫叢集建立 Aurora 複本。Aurora Serverless v1 具有單一資料庫執行個體，可自動擴展和縮減，以支援所有資料庫讀取和寫入操作。

  不過，您可將讀取器執行個體新增至 Aurora Serverless v2 資料庫叢集。如需詳細資訊，請參閱新增 Aurora Serverless v2 讀取器。

建議您將 Aurora 資料庫叢集的主要執行個體和 Aurora 複本分配在數個可用區域上，以改善資料庫叢集的可用性。如需更多詳細資訊，請參閱 區域可用性。

若要從 Aurora 資料庫叢集移除 Aurora 複本，請遵循從 Aurora 個體資料庫叢集刪除資料庫執行個體中的說明刪除 Aurora。

> **Note**
>
> Amazon Aurora 也支援以外部資料庫 (例如 RDS 資料庫執行個體) 進行複寫。RDS 資料庫執行個體必須與 Amazon Aurora 位於相同的AWS區域。如需詳細資訊，請參閱以 Amazon Aurora 進行複寫。

您可以使用 AWS 管理主控台AWS CLI、 或 RDS API，將 Aurora 複本新增至資料庫叢集。

主控台

將 Aurora 複本新增至資料庫叢集

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)，然後選取您要新增新資料庫執行個體的資料庫叢集。
3. 確定叢集和主要執行個體都處於 Available (可用) 狀態。如果資料庫叢集或主要執行個體處於轉換狀態 (例如 Creating (建立中))，則您無法新增複本。

如果叢集沒有主要執行個體，請使用 create-db-instanceAWS CLI 命令建立一個。如果您已使用 CLI 來還原資料庫叢集快照集，然後檢視 AWS 管理主控台 中的叢集，則會引發此情況。

4. 針對 Actions (動作)，選擇 Add reader (新增讀取器)。

   Add reader (新增讀取器) 頁面隨即出現。

5. 在 Add reader (新增讀取器) 頁面上，指定您 Aurora 複本的選項。下表顯示 Aurora 複本的設定。

若為此選項	執行此作業
Availability zone (可用區域)	決定您是否要指定特定的可用區域。清單只包含對應至您在建立資料庫叢集時所選資料庫子網路群組的可用區域。如需可用區域的詳細資訊，請參閱區域和可用區域。
可公開存取	選取 Yes 以給予 Aurora 複本一個公有 IP 地址；否則，選取 No。如需隱藏 Aurora 複本免於公開存取的詳細資訊，請參閱在 VPC 中的網際網路中隱藏資料庫叢集。
加密	選取 Enable encryption ，以啟用此 Aurora 複本的靜態加密。如需更多詳細資訊，請參閱 加密 Amazon Aurora 資源。
DB instance class (資料庫執行個體類別)	選取資料庫執行個體類別，定義 Aurora 複本的處理和記憶體要求。如需資料庫執行個體類別選項的詳細資訊，請參閱Amazon Aurora 資料庫執行個體類別。
Aurora replica source (&AUR; 複本來源)	選取要為其建立 Aurora 複本之主要執行個體的識別符。
DB instance identifier (資料庫執行個體識別符)：	為您選擇的AWS區域中的帳戶輸入唯一的執行個體名稱。您可以選擇將一些智慧新增至名稱，例如包含您選取的AWS區域和資料庫引擎，例如 **aurora-read-instance1** 。
優先順序	選擇執行個體的容錯移轉優先順序。如果您未選取值，則預設值為 tier-1 (第一層)。此優先順序決定從主要執行

若為此選項	執行此作業
	個體失敗中復原時提升 Aurora 複本的順序。如需更多詳細資訊，請參閱 Aurora 資料庫叢集的容錯能力。
Database port (資料庫連接埠)	Aurora 複本的連接埠與資料庫叢集的連接埠相同。
DB parameter group (資料庫參數群組)	選取參數群組。Aurora 有一個預設參數群組供您使用，您也可以建立自己的參數群組。如需參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。
Performance Insights (績效詳情)	Turn on Performance Insights (開啟績效詳情) 核取方塊預設為已選取。該值不會從寫入器執行個體繼承。如需詳細資訊，請參閱在 Amazon Aurora 上使用績效詳情監控資料庫負載。
Enhanced monitoring (增強型監控)	選擇 Enable enhanced monitoring (啟用增強型監控)，以針對資料庫叢集執行所在的作業系統即時收集指標。如需更多詳細資訊，請參閱 使用增強型監控來監控作業系統指標。
監控角色	只有在 Enhanced Monitoring (增強型監控) 設為 Enable enhanced monitoring (啟用增強型監控) 時才能使用。選擇您所建立的 IAM 角色，以允許 Amazon RDS 為您與 Amazon CloudWatch Logs 進行通訊，或選擇 Default (預設值)，讓 RDS 為您建立名為 `rds-monitoring-role` 的角色。如需更多詳細資訊，請參閱 使用增強型監控來監控作業系統指標。
精細程度	只有在 Enhanced Monitoring (增強型監控) 設為 Enable enhanced monitoring (啟用增強型監控) 時才能使用。針對資料庫叢集，設定收集指標之間的間隔 (以秒為單位)。

若為此選項	執行此作業
Auto minor version upgrade (自動次要版本升級)	如果您想要讓 Aurora 資料庫叢集可以自動接收可用的次要資料庫引擎版本升級，請選取 Enable auto minor version upgrade (啟用自動次要版本升級)。 Auto minor version upgrade (自動次要版本升級) 設定同時適用於 Aurora PostgreSQL 和 Aurora MySQL 資料庫叢集。對於 Aurora MySQL 2.x 叢集，此設定會將叢集升級至 2.07.2 的最高版本。 如需 Aurora PostgreSQL 引擎更新的詳細資訊，請參閱 Amazon Aurora PostgreSQL 的資料庫引擎更新。 如需 Aurora MySQL 引擎更新的詳細資訊，請參閱 Amazon Aurora MySQL 的資料庫引擎更新。

6. 選擇 Add reader (新增讀取器) 來建立 Aurora 複本。

AWS CLI

若要在資料庫叢集中建立 Aurora 複本，請執行 create-db-instanceAWS CLI 命令。包含資料庫叢集的名稱做為 `--db-cluster-identifier` 選項。您可以選擇性地使用 `--availability-zone` 參數，指定 Aurora 複本的可用區域，如下列範例所示。

例如，下列命令會建立新的 MySQL 5.7 –相容 Aurora 複本，名為 `sample-instance-us-west-2a`。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large \
    --availability-zone us-west-2a
```

在 Windows 中：

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large ^
```

```
    --availability-zone us-west-2a
```

下列命令會建立新的 MySQL 5.7 –相容 Aurora 複本，名為 `sample-instance-us-west-2a`。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large \
    --availability-zone us-west-2a
```

在 Windows 中：

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora --db-instance-class
 db.r5.large ^
    --availability-zone us-west-2a
```

下列命令會建立新的 PostgreSQL 相容 Aurora 複本，名為 `sample-instance-us-west-2a`。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-
class db.r5.large \
    --availability-zone us-west-2a
```

在 Windows 中：

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-
class db.r5.large ^
    --availability-zone us-west-2a
```

RDS API

若要在資料庫叢集中建立 Aurora 複本，請呼叫 CreateDBInstance 操作。包含資料庫叢集的名稱做為 `DBClusterIdentifier` 參數。您可以選擇使用 `AvailabilityZone` 參數，來指定 Aurora 複本的可用區域。

如需 Amazon Aurora 自動擴展搭配 Aurora 複本的相關資訊，請參閱下列各節。

主題

- 使用 Amazon Aurora 自動擴展搭配 Aurora 複本
- 將自動擴展政策新增至 Amazon Aurora 資料庫叢集
- 編輯 Amazon Aurora 資料庫叢集的自動擴展政策
- 從 Amazon Aurora 資料庫叢集刪除自動擴展政策

使用 Amazon Aurora 自動擴展搭配 Aurora 複本

為了滿足您的連線和工作負載要求，Aurora Auto Scaling 會動態調整為針對 Aurora 資料庫叢集佈建的 Aurora 複本 (讀取器資料庫執行個體) 數量。Aurora Auto Scaling 可同時適用於 Aurora MySQL 和 Aurora PostgreSQL。Aurora Auto Scaling 可讓您的 Aurora 資料庫叢集處理突然增加的連線或工作負載。當連線或工作負載減少時，Aurora Auto Scaling 會移除不必要的 Aurora 複本，讓您不需為了用不到的已佈建資料庫執行個體支付費用。

您定義並套用 Aurora 資料庫叢集的規模調整政策。擴展政策定義了 Aurora Auto Scaling 能管理的 Aurora 複本數量下限和上限。Aurora Auto Scaling 會根據政策調整 Aurora 複本的數量，使用 Amazon CloudWatch 指標和目標值來判斷配合實際的工作負載要增加或減少複本數量。

> **Note**
>
> Aurora 自動擴展不適用於寫入器資料庫執行個體上的工作負載。Aurora 自動擴展僅在讀取器執行個體上協助處理工作負載。

您可以使用 根據預先定義的指標AWS 管理主控台來套用擴展政策。或者，您可以使用 AWS CLI或 Aurora Auto Scaling API，根據預先定義的或自訂指標來套用擴展政策。

主題

- 開始之前
- Aurora Auto Scaling 政策
- 資料庫執行個體 ID 和標記
- Aurora Auto Scaling 和績效詳情

開始之前

在使用 Aurora Auto Scaling 擴展 Aurora 資料庫叢集前，您必須先建立一個具有主要 (寫入器) 資料庫執行個體的 Aurora 資料庫叢集。如需建立 Aurora 資料庫叢集的詳細資訊，請參閱建立 Amazon Aurora 資料庫叢集。

只有當資料庫叢集處於可使用狀態，Aurora Auto Scaling 才會擴展資料庫叢集。

當 Aurora Auto Scaling 新增 Aurora 複本時，新 Aurora 複本的資料庫執行個體會和主要執行個體所使用的相同。如需資料庫執行個體類別的詳細資訊，請參閱Amazon Aurora 資料庫執行個體類別。同時，新 Aurora 複本的提升層優先等級將設為最後 (即預設的 15)。這表示在容錯移轉期間，優先等級較

高的複本 (例如手動建立的複本) 將會優先提升。如需更多詳細資訊，請參閱 Aurora 資料庫叢集的容錯能力。

此外，Aurora Auto Scaling 只會移除它建立的 Aurora 複本。

若要得益於 Aurora Auto Scaling，您的應用程式必須支援連線到新的 Aurora 複本。若要這樣做，建議使用 Aurora 讀取器端點。您可以使用 AWSJDBC 驅動程式等驅動程式。如需詳細資訊，請參閱連接至 Amazon Aurora 資料庫叢集。

> **Note**
>
> Aurora 全域資料庫目前不支援次要資料庫叢集的 Aurora Auto Scaling。

Aurora Auto Scaling 政策

Aurora Auto Scaling 使用規模調整政策來調整 Aurora 資料庫叢集中 Aurora 複本的數量。Aurora Auto Scaling 有下列元件：

- 服務連結角色
- 目標指標
- 容量下限和上限
- 冷卻時間

主題

- 服務連結角色
- 目標指標
- 容量下限和上限
- 冷卻時間
- 啟用或停用規模縮減活動
- 新增、編輯或刪除自動擴展政策

服務連結角色

Aurora Auto Scaling 使用 `AWSServiceRoleForApplicationAutoScaling_RDSCluster` 服務連結角色。如需詳細資訊，請參閱《Application Auto Scaling 使用者指南》中的適用於 Application Auto Scaling 的服務連結角色。

目標指標

在這種類型的政策中，會在目標追蹤規模調整政策的設定中指定預先定義的或自訂的指標以及指標的目標值。Aurora Auto Scaling 會建立和管理可觸發規模調整政策的 CloudWatch 警示，並根據指標和目標值來計算規模調整。規模調整政策會視需要新增或移除 Aurora 複本，以讓指標保持在等於或接近指定目標值。除了讓指標保持在接近目標值之外，目標追蹤規模調整政策也會配合因為變更工作負載所造成的指標波動而進行調整。這樣的政策也能將資料庫叢集可用 Aurora 複本數量的快速波動減到最低。

例如，規模調整政策使用預先定義的平均 CPU 使用率指標。這個政策可以讓 CPU 使用率維持在 (或接近) 指定的使用率百分比，像是百分之 40。

> **Note**
>
> 在每個 Aurora 資料庫叢集上，您只能為每一個目標指標建立一個 Auto Scaling 政策。

當您設定 Aurora 自動擴展時，目標指標值會計算為叢集中所有讀取器執行個體的平均值。執行此計算的方式如下：

- 包括 Aurora 叢集中的所有讀取器執行個體，無論它們是由自動擴展管理或手動新增。
- 包括與自訂端點相關聯的執行個體。自訂端點不會影響目標指標的計算。
- 不包含叢集的寫入器執行個體。

指標衍生自使用下列維度的 CloudWatch：

- `DBClusterIdentifier`
- `Role=READER`

例如，請考慮具有下列設定的 Aurora MySQL 叢集：

- 手動執行個體 (非由自動擴展控制)：
  - 具有 50% CPU 使用率的寫入器
  - 具有 90% CPU 使用率的讀取器 1 (自訂端點：`custom-reader-1`)
  - 具有 90% CPU 使用率的讀取器 2 (自訂端點：`custom-reader-2`)
- 自動擴展執行個體：
  - 具有 10% CPU 使用率的讀取器 3 (使用自動擴展新增)

在此案例中，自動擴展政策的目標指標計算方式如下：

```
Target metric = (CPU utilization of reader 1 + reader 2 + reader 3) / total number of
 readers

Target metric = (90 + 90 + 10) / 3 = 63.33%
```

自動擴展政策會使用此值來評估是否要根據定義的閾值縮減或橫向擴充。

考慮下列各項：

- 雖然自訂端點會判斷流量如何路由至特定讀取器，但不會將讀取器排除在指標計算之外。
- 手動執行個體一律包含在目標指標計算中。
- 為了避免意外擴展行為，請確定自動擴展組態會考慮叢集中的所有讀取器執行個體。
- 如果叢集沒有讀取器，則不會計算指標，且自動擴展政策警示會保持非作用中狀態。若要讓自動擴展政策有效運作，至少必須隨時有一個讀取器。

容量下限和上限

您可以指定將由 Application Auto Scaling 管理的 Aurora 複本數量上限。此值必須介於 0 – 15，而且必須大於或等於 Aurora 複本數量下限的指定值。

您也可以指定將由 Application Auto Scaling 管理的 Aurora 複本數量下限。此值必須介於 0 – 15，而且必須小於或等於 Aurora 複本數量上限的指定值。

至少必須有一個讀取器資料庫執行個體，Aurora 自動擴展才能運作。如果資料庫叢集沒有讀取器執行個體，且您將最小容量設定為 0，則 Aurora 自動擴展將無法運作。

> **Note**
>
> Aurora 資料庫叢集需設定容量下限和上限。指定的值會套用至與該 Aurora 資料庫叢集相關聯的所有政策。

冷卻時間

藉由新增冷卻時間 (會影響 Aurora 資料庫叢集的擴展和縮減)，您可以調整目標追蹤規模調整政策的靈活性。冷卻時間會封鎖後續的擴展或縮減請求，直到冷卻時間到期。這些封鎖會拖慢規模縮減請求刪除 Aurora 資料庫叢集中 Aurora 複本的動作，以及拖慢規模擴展請求建立 Aurora 複本的動作。

您可以指定下列其中一種冷卻時間：

- 縮減動作會減少 Aurora 資料庫叢集中 Aurora 複本的數量。規模縮減冷卻時間會指定在規模縮減動作完成之後，另一個規模縮減動作可以再開始執行之前的等待時間長度 (秒)。
- 擴增動作會增加 Aurora 資料庫叢集中 Aurora 複本的數量。規模擴展冷卻時間會指定在規模擴展動作完成之後，可以再開始執行另一個規模擴展動作之前的等待時間長度 (秒)。

> **Note**
>
> 如果後續橫向擴展請求的 Aurora 複本數量大於第一個請求，則會忽略橫向擴展冷卻時間。

如果您未設定縮減或橫向擴展的冷卻時間，則每個的預設值為 300 秒。

啟用或停用規模縮減活動

您可以啟用或停用政策的規模縮減動作。啟用規模縮減動作可讓規模調整政策刪除 Aurora 複本。規模縮減動作啟用時，規模調整政策中的規模縮減冷卻時間會套用至規模縮減動作。停用規模縮減動作可防止規模調整政策刪除 Aurora 複本。

> **Note**
>
> 規模擴展動作會一律啟用，如此規模調整政策即可根據需要來建立 Aurora 複本。

新增、編輯或刪除自動擴展政策

您可以使用 或 Application Auto Scaling API 新增AWS 管理主控台AWS CLI、編輯或刪除自動擴展政策。如需新增、編輯或刪除自動擴展政策的詳細資訊，請參閱下列各節。

- 將自動擴展政策新增至 Amazon Aurora 資料庫叢集
- 編輯 Amazon Aurora 資料庫叢集的自動擴展政策
- 從 Amazon Aurora 資料庫叢集刪除自動擴展政策

資料庫執行個體 ID 和標記

當 Aurora Auto Scaling 新增複本時，其資料庫執行個體 ID 會以 `application-autoscaling-` 為前綴，例如 `application-autoscaling-61aabbcc-4e2f-4c65-b620-ab7421abc123`。

下列標籤會自動新增至資料庫執行個體。您可以在資料庫執行個體詳細資訊頁面的 Tags (標籤) 頁籤上檢視。

標籤	Value
application-autoscaling:resourceId	cluster:mynewcluster-cluster

如需 Amazon RDS 資源標籤的詳細資訊，請參閱標記 Amazon Aurora 和 Amazon RDS 資源。

Aurora Auto Scaling 和績效詳情

和任何 Aurora 讀取器資料庫執行個體一樣，您可以使用績效詳情來監控 Aurora Auto Scaling 新增的複本。

如需了解如何使用績效詳情來監控 Aurora 資料庫叢集，請參閱 在 Amazon Aurora 上使用績效詳情監控資料庫負載。

將自動擴展政策新增至 Amazon Aurora 資料庫叢集

您可以使用 AWS 管理主控台、 AWS CLI或 Application Auto Scaling API 新增擴展政策。

> **Note**
>
> 如需使用 新增擴展政策的範例 CloudFormation，請參閱《 使用者指南》中的宣告 Aurora 資料庫叢集的擴展政策。 AWS CloudFormation

主控台

您可以使用 將擴展政策新增至 Aurora 資料庫叢集 AWS 管理主控台。

將自動規模調整政策新增至 Aurora 資料庫叢集

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您要新增政策的 Aurora 資料庫叢集。

4. 選擇 Logs & events (日誌與事件) 標籤。
5. 在 Auto scaling policies (自動調整規模政策) 區段中，選擇 Add (新增)。

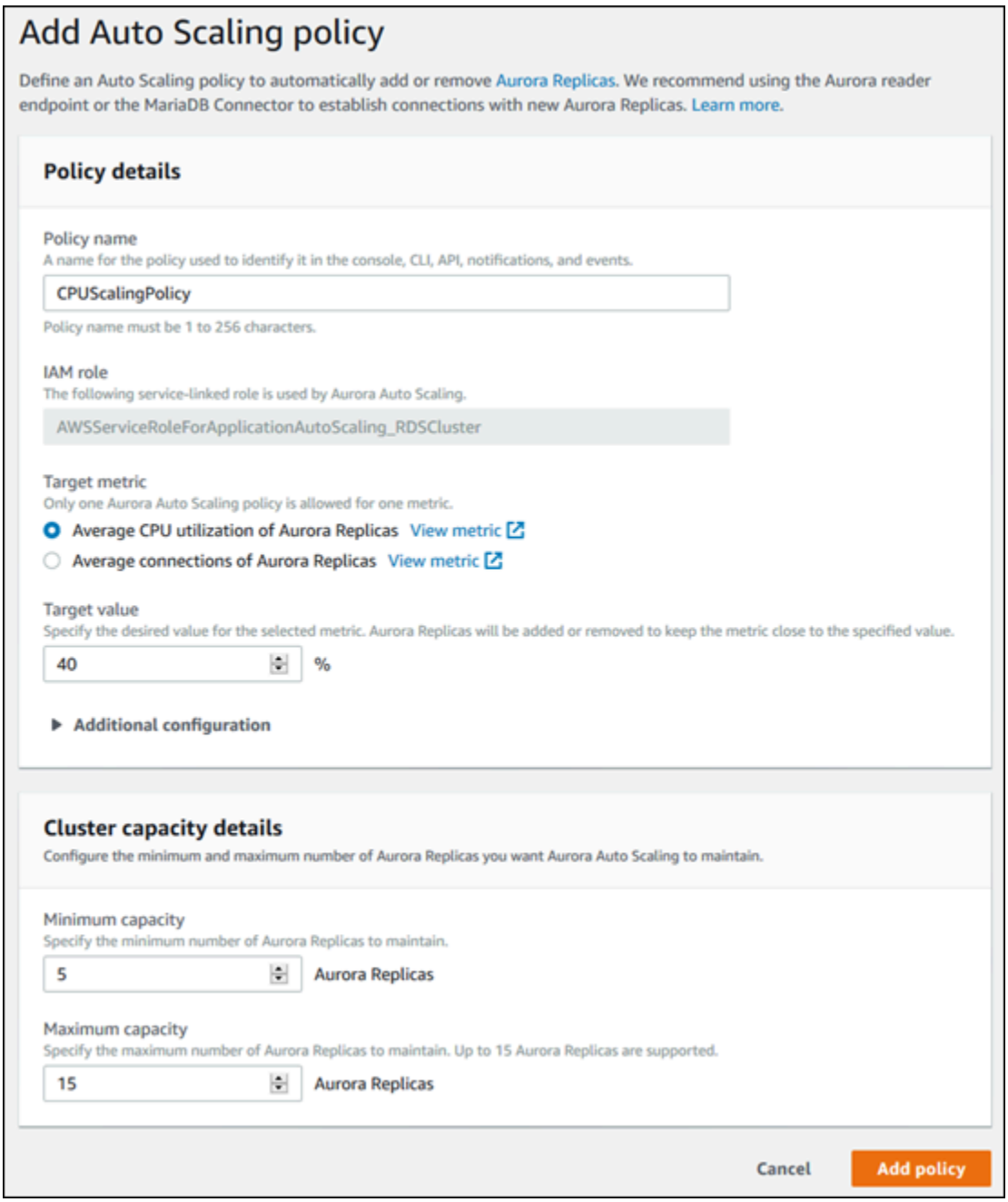
   Add Auto Scaling policy (新增自動調整規模政策) 對話方塊隨即顯示。
6. 針對 Policy Name (政策名稱)，請輸入政策名稱。
7. 目標指標請選擇下列其中一個：

   - Average CPU utilization of Aurora Replicas (&AUR; 複本的平均 CPU 使用率)，可根據平均 CPU 使用率建立政策。
   - Average connections of Aurora Replicas (&AUR; 複本的平均連線數)，可根據連線至 Aurora 複本的平均連線數目建立政策。
8. 目標值請輸入下列其中一個：

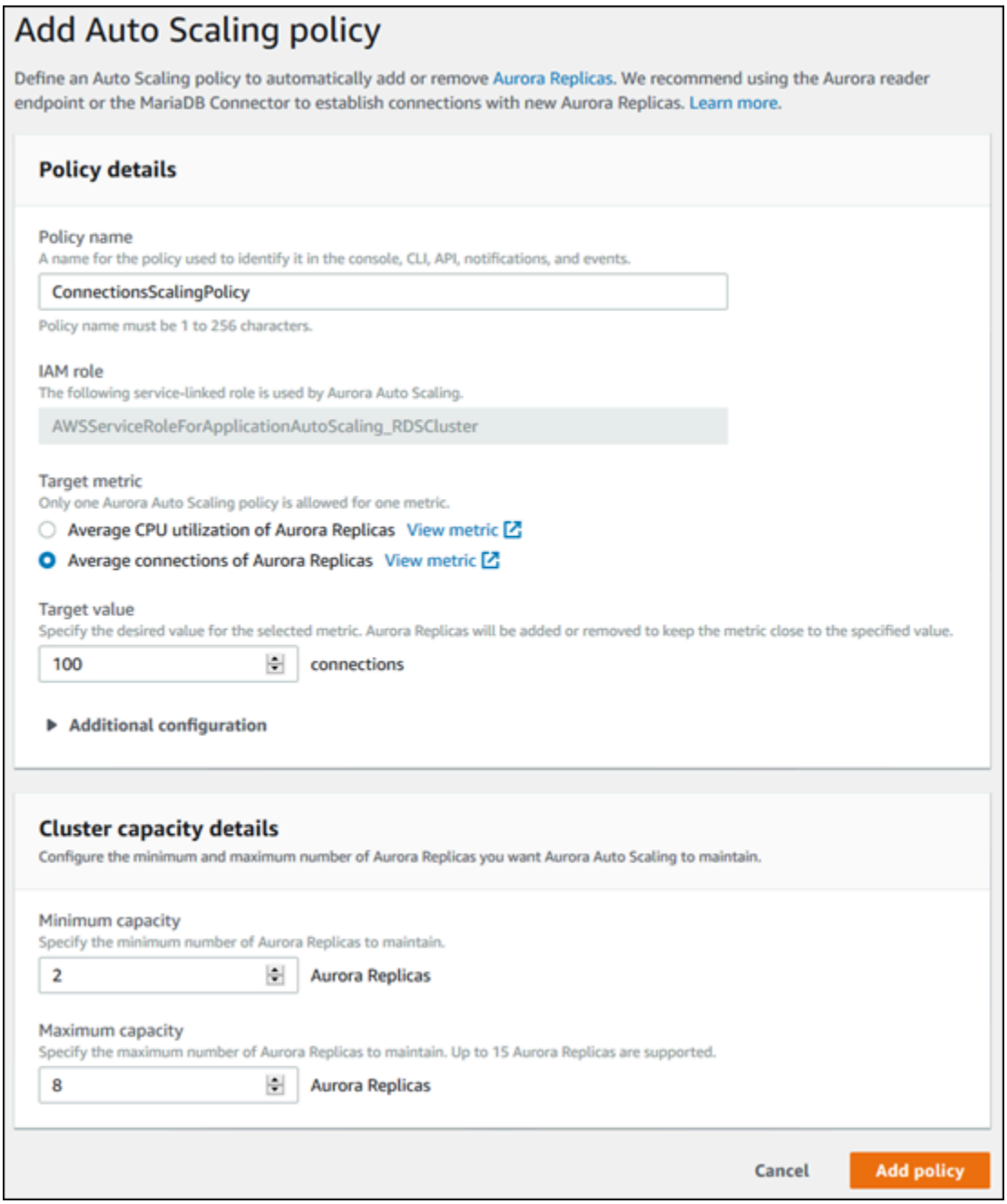
   - 如果在前一步驟選擇 Average CPU utilization of Aurora Replicas (&AUR; 複本的平均 CPU 使用率)，輸入您想在 Aurora 複本上維持的 CPU 使用率百分比。
   - 如果在前一步驟選擇 Average connections of Aurora Replicas (&AUR; 複本的平均連線數)，輸入您想維持的連線數目。

   系統會新增或移除 Aurora 複本，讓指標接近指定的值。
9. (選用) 展開 Additional Configuration (其他組態) 以建立縮減或橫向擴展冷卻時間。
10. 在 Minimum capacity (容量下限) 輸入 Aurora Auto Scaling 政策需要維持的 Aurora 複本數量下限。
11. 在 Maximum capacity (容量上限) 輸入 Aurora Auto Scaling 政策需要維持的 Aurora 複本數量上限。
12. 選擇 Add Policy (新增政策)。

以下對話方塊會根據平均 CPU 使用率百分之 40 來建立 Auto Scaling 政策。此政策會指定最少 5 個 Aurora 複本以及最多 15 個 Aurora 複本。



以下對話方塊會根據平均連線數 100 使用率來建立自動規模調整政策。此政策會指定最少 2 個 Aurora 複本及最多 8 個 Aurora 複本。



AWS CLI 或 Application Auto Scaling API

您可以根據預先定義的指標或自訂指標，套用規模調整政策。若要這樣做，您可以使用 AWS CLI 或 Application Auto Scaling API。第一步是將您的 Aurora 資料庫叢集註冊到 Application Auto Scaling。

註冊 Aurora 資料庫叢集

在使用 Aurora Auto Scaling 擴展 Aurora 資料庫叢集前，您應先將您的 Aurora 資料庫叢集註冊到 Application Auto Scaling。這是為了定義要套用到叢集的擴展維度和限制。Application Auto Scaling 會

隨 rds:cluster:ReadReplicaCount 代表 Aurora 複本數目的可擴展維度，動態擴展 Aurora 資料庫叢集。

若要註冊 Aurora 資料庫叢集，您可以使用 AWS CLI 或 Application Auto Scaling API。

AWS CLI

若要註冊您的 Aurora 資料庫叢集，請使用 register-scalable-target AWS CLI 命令搭配下列參數：

- --service-namespace – 將此值設定為 rds。
- --resource-id– Aurora 資料庫叢集的資源識別符。這項參數的資源類型為 cluster，且唯一識別符是 Aurora 資料庫叢集的名稱，例如 cluster:myscalablecluster。
- --scalable-dimension – 將此值設定為 rds:cluster:ReadReplicaCount。
- --min-capacity – Application Auto Scaling 要管理的讀取者資料庫執行個體數量下限。如需了解有關 --min-capacity、--max-capacity 以及叢集中資料庫執行個體數目之間的關係資訊，請參閱 容量下限和上限。
- --max-capacity – Application Auto Scaling 要管理的讀取者資料庫執行個體數量上限。如需了解有關 --min-capacity、--max-capacity 以及叢集中資料庫執行個體數目之間的關係資訊，請參閱 容量下限和上限。

Example

在下列範例中，您會註冊名為 myscalablecluster 的 Aurora 資料庫叢集。註冊中表明應該動態縮放資料庫叢集，使其具有 1 到 8 個 Aurora 複本。

對於 Linux、macOS 或 Unix：

```
aws application-autoscaling register-scalable-target \
    --service-namespace rds \
    --resource-id cluster:myscalablecluster \
    --scalable-dimension rds:cluster:ReadReplicaCount \
    --min-capacity 1 \
    --max-capacity 8 \
```

在 Windows 中：

```
aws application-autoscaling register-scalable-target ^
```

```
    --service-namespace rds ^
    --resource-id cluster:myscalablecluster ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
    --min-capacity 1 ^
    --max-capacity 8 ^
```

Application Auto Scaling API

若要向 Application Auto Scaling 註冊您的 Aurora 資料庫叢集，請使用 RegisterScalableTarget Application Auto Scaling API 操作搭配下列參數：

- `ServiceNamespace` – 將此值設定為 `rds`。
- `ResourceID`– Aurora 資料庫叢集的資源識別符。這項參數的資源類型為 `cluster`，且唯一識別符是 Aurora 資料庫叢集的名稱，例如 `cluster:myscalablecluster`。
- `ScalableDimension` – 將此值設定為 `rds:cluster:ReadReplicaCount`。
- `MinCapacity` – Application Auto Scaling 要管理的讀取者資料庫執行個體數量下限。如需了解有關 `MinCapacity`、`MaxCapacity` 以及叢集中資料庫執行個體數目之間的關係資訊，請參閱 容量下限和上限。
- `MaxCapacity` – Application Auto Scaling 要管理的讀取者資料庫執行個體數量上限。如需了解有關 `MinCapacity`、`MaxCapacity` 以及叢集中資料庫執行個體數目之間的關係資訊，請參閱 容量下限和上限。

Example

在下列範例中，您會使用 Application Auto Scaling API 註冊名為 `myscalablecluster` 的 Aurora 資料庫叢集。此註冊中表明應該動態縮放資料庫叢集，使其具有 1 到 8 個 Aurora 複本。

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.RegisterScalableTarget
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "ServiceNamespace": "rds",
```

```
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "MinCapacity": 1,
    "MaxCapacity": 8
}
```

定義 Aurora 資料庫叢集的擴展原則

目標追踪規模調整政策組態由 JSON 區塊表示，其中定義了指標和目標值。您可以將規模調整政策的組態設定，儲存為文字檔案中的 JSON 區塊。叫用 AWS CLI 或 Application Auto Scaling API 時，您可以使用該文字檔案。如需政策組態語法的詳細資訊，請參閱 Application Auto Scaling API 參考中的 `TargetTrackingScalingPolicyConfiguration`。

您可使用下列的選項，來定義目標追蹤規模調整政策的組態設定。

主題

- 使用預先定義的指標
- 使用自訂的指標
- 使用冷卻時間
- 停用規模縮減活動

使用預先定義的指標

使用預先定義的指標，您可以為搭配 Aurora Auto Scaling 中的目標追踪和動態擴展都運作良好的 Aurora 資料庫叢集，快速定義目標追踪規模調整政策。

目前 Aurora 在 Aurora Auto Scaling 中支援下列預先定義的指標：

- RDSReaderAverageCPUUtilization – 在 CloudWatch 中，Aurora 資料庫叢集中所有 Aurora 複本的 `CPUUtilization` 指標平均值。
- RDSReaderAverageDatabaseConnections – 在 CloudWatch 中，Aurora 資料庫叢集中所有 Aurora 複本的 `DatabaseConnections` 指標平均值。

如需 `CPUUtilization` 和 `DatabaseConnections` 指標的詳細資訊，請參閱 Amazon Aurora 的 Amazon CloudWatch 指標。

若要在您的規模調整政策中使用預先定義的指標，請為規模調整政策建立目標追蹤組態設定。此組態設定必須加入用於預先定義指標的 `PredefinedMetricSpecification`，以及用於該指標目標值的 `TargetValue`。

Example

下列的範例描述 Aurora 資料庫叢集目標追蹤規模調整的典型政策組態設定。在此組態中，會使用 RDSReaderAverageCPUUtilization 這個預先定義的指標，根據所有 Aurora 複本平均 CPU 使用率百分之 40，來調整 Aurora 資料庫叢集。

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    }
}
```

使用自訂的指標

使用自訂的指標，您可以定義目標追蹤規模調整政策來滿足您的自訂需求。您可以根據與規模調整成比例變動的任何 Aurora 指標，來定義自訂的指標。

並非所有的 Aurora 指標都適用於目標追蹤。指標必須是有效的使用率指標，而且能夠表示執行個體的忙碌程度。指標的值必須與 Aurora 資料庫叢集中 Aurora 複本的數量，按比例增加或減少。對於使用指標資料按比例規模擴展或縮減 Aurora 複本數量來說，這種按比例增加或減少是必要的。

Example

下列的範例描述規模調整政策的目標追蹤組態設定。在此組態中，自訂指標會根據 Aurora 資料庫叢集 my-db-cluster 中所有 Aurora 複本平均 CPU 使用率百分之 50，來調整 Aurora 資料庫叢集。

```
{
    "TargetValue": 50,
    "CustomizedMetricSpecification":
    {
        "MetricName": "CPUUtilization",
        "Namespace": "AWS/RDS",
        "Dimensions": [
            {"Name": "DBClusterIdentifier","Value": "my-db-cluster"},
            {"Name": "Role","Value": "READER"}
        ],
        "Statistic": "Average",
        "Unit": "Percent"
    }
}
```

使用冷卻時間

您可以指定一個值 (單位為秒)，讓 ScaleOutCooldown 為您的 Aurora 資料庫叢集規模擴展新增冷卻時間。同樣的，您可以指定一個值 (單位為秒)，讓 ScaleInCooldown 為您的 Aurora 資料庫叢集規模縮減新增冷卻時間。如需 ScaleInCooldown 和 ScaleOutCooldown 的詳細資訊，請參閱《Application Auto Scaling API 參考》中的 TargetTrackingScalingPolicyConfiguration。

Example

下列的範例描述規模調整政策的目標追蹤組態設定。在此組態中，會使用 RDSReaderAverageCPUUtilization 這個預先定義的指標，根據 Aurora 資料庫叢集中所有 Aurora 複本平均 CPU 使用率百分之 40，來調整 Aurora 資料庫叢集。這個組態設定分別提供了 10 分鐘的規模縮減冷卻時間，和 5 分鐘的規模擴展冷卻時間。

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    },
    "ScaleInCooldown": 600,
    "ScaleOutCooldown": 300
}
```

停用規模縮減活動

藉由停用規模縮減動作，您可以防止目標追蹤規模調整政策的組態設定，對您的 Aurora 資料庫叢集進行規模調整。停用規模縮減的動作，可防止規模調整政策刪除 Aurora 複本，同時讓規模調整政策仍然能夠視需要來建立 &AUR; 複本。

您可以為 DisableScaleIn 指定布林值，來啟用或停用 Aurora 資料庫叢集的規模縮減活動。如需 DisableScaleIn 的詳細資訊，請參閱《Application Auto Scaling API 參考》中的 TargetTrackingScalingPolicyConfiguration。

Example

下列的範例描述規模調整政策的目標追蹤組態設定。在此組態中，RDSReaderAverageCPUUtilization 這個預先定義的指標會根據 Aurora 資料庫叢集中所有 Aurora 複本平均 CPU 使用率百分之 40，來調整 Aurora 資料庫叢集。此組態設定停用了規模調整政策的規模縮減動作。

```
{
```

```
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    },
    "DisableScaleIn": true
}
```

將擴展原則套用至 Aurora 資料庫叢集

向 Application Auto Scaling 註冊您的 Aurora 資料庫叢集，並定義擴展政策之後，請將擴展政策套用到已註冊的 Aurora 資料庫叢集。若要將擴展政策套用至 Aurora 資料庫叢集，您可以使用 AWS CLI 或 Application Auto Scaling API。

AWS CLI

若要將擴展政策套用至 Aurora 資料庫叢集，請使用 `put-scaling-policy` AWS CLI 命令搭配下列參數：

- `--policy-name` – 擴展政策的名稱。
- `--policy-type` – 將此值設定為 `TargetTrackingScaling`。
- `--resource-id`– Aurora 資料庫叢集的資源識別符。這項參數的資源類型為 `cluster`，且唯一識別符是 Aurora 資料庫叢集的名稱，例如 `cluster:myscalablecluster`。
- `--service-namespace` – 將此值設定為 `rds`。
- `--scalable-dimension` – 將此值設定為 `rds:cluster:ReadReplicaCount`。
- `--target-tracking-scaling-policy-configuration` – 要用於 Aurora 資料庫叢集的目標追蹤擴展政策組態。

Example

在下列範例中，您會使用 Application Auto Scaling 將名為 `myscalablepolicy` 的目標追蹤擴展政策，套用到名為 `myscalablecluster` 的 Aurora 資料庫叢集。做法是使用儲存於 `config.json` 檔案中的政策組態設定。

對於 Linux、macOS 或 Unix：

```
aws application-autoscaling put-scaling-policy \
    --policy-name myscalablepolicy \
```

```
    --policy-type TargetTrackingScaling \
    --resource-id cluster:myscalablecluster \
    --service-namespace rds \
    --scalable-dimension rds:cluster:ReadReplicaCount \
    --target-tracking-scaling-policy-configuration file://config.json
```

在 Windows 中：

```
aws application-autoscaling put-scaling-policy ^
    --policy-name myscalablepolicy ^
    --policy-type TargetTrackingScaling ^
    --resource-id cluster:myscalablecluster ^
    --service-namespace rds ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
    --target-tracking-scaling-policy-configuration file://config.json
```

Application Auto Scaling API

若要使用 Application Auto Scaling API 將擴展政策套用到您的 Aurora 資料庫叢集，請使用 PutScalingPolicy Application Auto Scaling API 操作搭配下列參數：

- PolicyName – 擴展政策的名稱。
- ServiceNamespace – 將此值設定為 rds。
- ResourceID– Aurora 資料庫叢集的資源識別符。這項參數的資源類型為 cluster，且唯一識別符是 Aurora 資料庫叢集的名稱，例如 cluster:myscalablecluster。
- ScalableDimension – 將此值設定為 rds:cluster:ReadReplicaCount。
- PolicyType – 將此值設定為 TargetTrackingScaling。
- TargetTrackingScalingPolicyConfiguration – 要用於 Aurora 資料庫叢集的目標追蹤擴展政策組態。

Example

在下列範例中，您會使用 Application Auto Scaling 將名為 myscalablepolicy 的目標追蹤擴展政策，套用到名為 myscalablecluster 的 Aurora 資料庫叢集。您使用的政策組態設定，是以 RDSReaderAverageCPUUtilization 這個預先定義的指標為根據。

```
POST / HTTP/1.1
```

```
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.PutScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "PolicyType": "TargetTrackingScaling",
    "TargetTrackingScalingPolicyConfiguration": {
        "TargetValue": 40.0,
        "PredefinedMetricSpecification":
        {
            "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
        }
    }
}
```

編輯 Amazon Aurora 資料庫叢集的自動擴展政策

您可以使用 AWS 管理主控台、AWS CLI 或 Application Auto Scaling API 來編輯擴展政策。

主控台

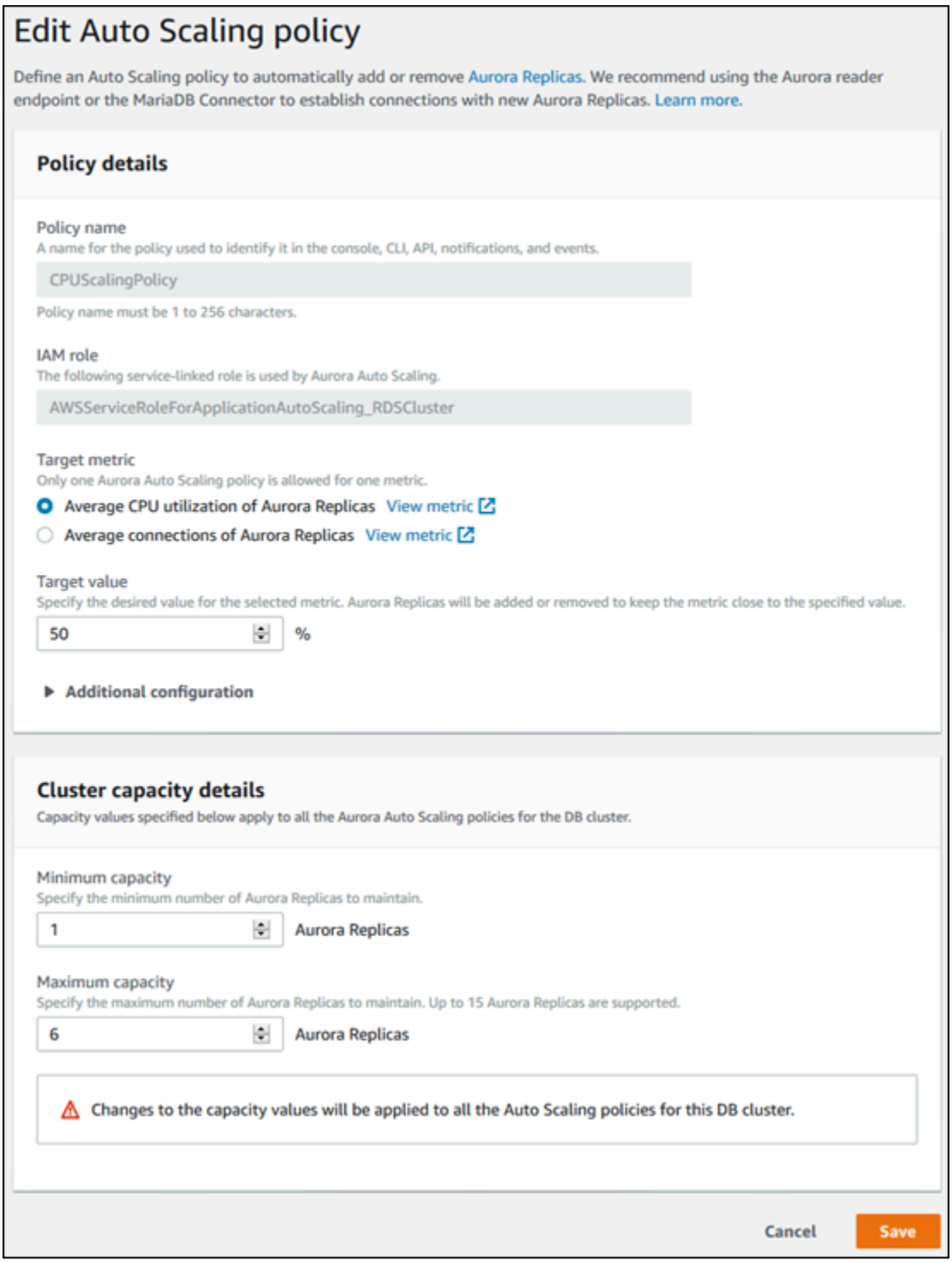
您可以使用 AWS 管理主控台 編輯規模調整政策。

編輯 Aurora 資料庫叢集的自動規模調整政策

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您要編輯其自動規模調整政策的 Aurora 資料庫叢集。
4. 選擇 Logs & events (日誌與事件) 標籤。
5. 在 Auto Scaling Policies (自動規模調整政策) 區段中，選擇一個自動規模調整政策，然後選擇 Edit (編輯)。

6. 對政策進行變更。
7. 選擇 Save (儲存)。

以下是 Edit Auto Scaling policy (編輯 Auto Scaling 政策) 對話方塊的範例。



AWS CLI 或 Application Auto Scaling API

您可以使用 AWS CLI 或 Application Auto Scaling API，以套用擴展政策時所採用的相同的方式，來編輯擴展政策：

- 使用 AWS CLI 時，請在 `--policy-name` 參數中指定您所要編輯之政策的名稱。針對您想要變更的參數指定新的參數值。
- 使用 Application Auto Scaling API 時，請在 `PolicyName` 參數中指定您所要編輯之政策的名稱。針對您想要變更的參數指定新的參數值。

如需更多詳細資訊，請參閱 將擴展原則套用至 Aurora 資料庫叢集。

從 Amazon Aurora 資料庫叢集刪除自動擴展政策

您可以使用 AWS 管理主控台、AWS CLI 或 Application Auto Scaling API，來刪除擴展政策。

主控台

您可以使用 AWS 管理主控台 刪除規模調整政策。

刪除 Aurora 資料庫叢集的自動規模調整政策

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您要刪除其自動規模調整政策的 Aurora 資料庫叢集。
4. 選擇 Logs & events (日誌與事件) 標籤。
5. 在 Auto Scaling Policies (自動規模調整政策) 區段中，選擇 Auto Scaling 政策，然後選擇 Delete (刪除)。

AWS CLI

若要從 Aurora 資料庫叢集刪除擴展政策，請使用 `delete-scaling-policy` AWS CLI 命令搭配下列參數：

- `--policy-name` – 擴展政策的名稱。
- `--resource-id`– Aurora 資料庫叢集的資源識別符。這項參數的資源類型為 `cluster`，且唯一識別符是 Aurora 資料庫叢集的名稱，例如 `cluster:myscalablecluster`。

- `--service-namespace` – 將此值設定為 `rds`。
- `--scalable-dimension` – 將此值設定為 `rds:cluster:ReadReplicaCount`。

Example

在下列範例中，您會將名為 `myscalablepolicy` 的目標追蹤擴展政策，從名為 `myscalablecluster` 的 Aurora 資料庫叢集中刪除。

對於 Linux、macOS 或 Unix：

```
aws application-autoscaling delete-scaling-policy \
    --policy-name myscalablepolicy \
    --resource-id cluster:myscalablecluster \
    --service-namespace rds \
    --scalable-dimension rds:cluster:ReadReplicaCount \
```

在 Windows 中：

```
aws application-autoscaling delete-scaling-policy ^
    --policy-name myscalablepolicy ^
    --resource-id cluster:myscalablecluster ^
    --service-namespace rds ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
```

Application Auto Scaling API

若要從 Aurora 資料庫叢集刪除擴展政策，請使用 DeleteScalingPolicy Application Auto Scaling API 操作搭配下列參數：

- `PolicyName` – 擴展政策的名稱。
- `ServiceNamespace` – 將此值設定為 `rds`。
- `ResourceID`– Aurora 資料庫叢集的資源識別符。這項參數的資源類型為 `cluster`，且唯一識別符是 Aurora 資料庫叢集的名稱，例如 `cluster:myscalablecluster`。
- `ScalableDimension` – 將此值設定為 `rds:cluster:ReadReplicaCount`。

Example

在下列範例中，您會使用 Application Auto Scaling API 將名為 `myscalablepolicy` 的目標追蹤擴展政策，從名為 `myscalablecluster` 的 Aurora 資料庫叢集中刪除。

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.DeleteScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount"
}
```

管理 Aurora 資料庫叢集的效能和擴展

您可以使用下列選項來管理 Aurora 資料庫叢集和資料庫執行個體的效能與擴展：

主題

- 儲存體擴展
- 執行個體擴展
- 讀取擴展
- 管理連線
- 管理查詢執行計劃

儲存體擴展

Aurora 儲存體會隨著叢集磁碟區中的資料而自動擴展。隨著資料成長，叢集磁碟區儲存體會根據資料庫引擎版本擴展。如需每個引擎版本 Aurora 叢集磁碟區大小上限的資訊，請參閱 Amazon Aurora 大小限制。若要了解叢集磁碟區中包含哪些類型的資料，請參閱 Amazon Aurora 儲存體。如需特定版本大小上限的詳細資訊，請參閱 Amazon Aurora 大小限制。

每小時會檢查一次叢集磁碟區的大小，以決定您的儲存成本。如需定價資訊，請參閱 Aurora 定價頁面。

即使 Aurora 叢集磁碟區的大小可以擴展至數個 TiB，您只需支付磁碟區中所使用空間的費用。判斷計費儲存空間的機制取決於 Aurora 叢集的版本。

- 從叢集磁碟區移除 Aurora 資料時，整體計費空間將會大幅減少。當捨棄或重整基礎資料表空間以使用較少的空間時，就會發生此動態調整大小的行為。因此，您可以捨棄不再需要的資料表和資料庫，以減少儲存費用。動態調整大小適用於特定 Aurora 版本。下列是當您移除資料時，叢集磁碟區動態調整大小的 Aurora 版本：

資料庫引擎	具有動態調整大小的版本
Aurora MySQL	• 第 3 版 (與 MySQL 8.0 相容)：所有支援的版本 • 第 2 版 (與 MySQL 5.7 相容)：2.11 和更新版本

資料庫引擎	具有動態調整大小的版本
Aurora PostgreSQL	所有支援的版本
Aurora Serverless v2	所有支援的版本
Aurora Serverless v1	不適用

- 在低於上述清單中的 Aurora 版本中，叢集磁碟區可以重複使用移除資料時釋放的空間，但磁碟區本身的大小永遠不會減少。

動態調整大小適用於實際移除叢集磁碟區內資料表空間或調整大小的操作。因此，它適用於 SQL 陳述式，例如 `DROP TABLE`、`DROP DATABASE`、`TRUNCATE TABLE` 和 `ALTER TABLE ... DROP PARTITION`。它不適用於使用 `DELETE` 陳述式刪除資料列。如果您從資料表中刪除大量的資料列，之後您可以執行 Aurora MySQL `OPTIMIZE TABLE` 陳述式或使用 Aurora PostgreSQL `pg_repack` 擴充功能來重組資料表，並動態調整叢集磁碟區的大小。

對於 Aurora MySQL，適用下列考量：

- 將資料庫叢集升級至支援動態調整大小的資料庫引擎版本後，當該功能在該特定中啟用時 AWS 區域，可`DROP TABLE`回收稍後由特定 SQL 陳述式釋放的任何空間，例如 。

  如果功能在特定 中明確停用 AWS 區域，則即使支援動態調整大小的版本，空間可能只能重複使用，也不能回收。

  此功能已在 2020 年 11 月至 2022 年 3 月期間針對特定資料庫引擎版本 (1.23.0–1.23.4、2.09.0–2.09.3 和 2.10.0) 啟用，並在任何後續版本上預設啟用。
- 資料表會在內部存放在大小不同的一或多個連續片段中。執行 `TRUNCATE TABLE` 操作時，對應到第一個片段的空間是可重複使用且不可回收的。其他片段可回收。在 `DROP TABLE` 操作期間，對應到整個資料表空間的空間是可回收的。
- `innodb_file_per_table` 參數會影響資料表儲存的組織方式。當資料表屬於系統資料表空間時，刪除資料表並不會減少系統資料表空間的大小。因此，請務必將 Aurora MySQL 資料庫叢集的 `innodb_file_per_table` 設為 1，以充分利用動態調整大小。
- 在 2.11 版和更新版本中，在重新啟動時會捨棄並重新建立 InnoDB 暫存資料表空間。這會將暫存資料表空間所佔用的空間釋放給系統，叢集磁碟區隨後也會調整大小。若要充分利用動態調整大小功能，建議將資料庫叢集升級至 2.11 版或更新版本。

> **Note**
>
> 動態調整大小功能不會在資料表空間中的資料表捨棄時立即回收空間，而是以每天大約 10 TB 的速率逐漸回收空間。系統資料表空間中的空間不會回收，因為永遠不會移除系統資料表空間。當操作需要資料表空間中的空間時，就會重複使用該資料表空間中未回收的可用空間。只有當叢集處於可用狀態時，動態調整大小功能才能回收儲存空間。

您可以透過監控 CloudWatch 中的 `VolumeBytesUsed` 指標來檢查叢集使用了多少儲存空間。如需儲存體帳單的詳細資訊，請參閱 Aurora 資料儲存體的計費方式。

- 在 中 AWS 管理主控台，您可以檢視叢集詳細資訊頁面上的`Monitoring`標籤，在圖表中看到此圖。
- 使用 AWS CLI，您可以執行類似下列 Linux 範例的命令。將開始和結束時間以及叢集的名稱替換為您自己的值。

```
aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '6 hours ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" \
  --statistics Average Maximum Minimum \
  --dimensions Name=DBClusterIdentifier,Value=my_cluster_identifier
```

此命令會產生類似下列的輸出。

```
{
    "Label": "VolumeBytesUsed",
    "Datapoints": [
        {
            "Timestamp": "2020-08-04T21:25:00+00:00",
            "Average": 182871982080.0,
            "Minimum": 182871982080.0,
            "Maximum": 182871982080.0,
            "Unit": "Bytes"
        }
    ]
}
```

下列範例示範如何在 Linux 系統上使用 AWS CLI 命令，追蹤 Aurora 叢集一段時間內的儲存用量。`--start-time` 和 `--end-time` 參數將整體時間間隔定義為一天。`--period` 參數會要求每隔一小時

測量一次。選擇較小的 --period 值是沒有意義的，因為指標是以間隔時間收集，而非持續收集。此外，Aurora 儲存操作有時會在相關的 SQL 陳述式完成後，繼續在後台運作一段時間。

第一個範例以預設 JSON 格式傳回輸出。資料點以任意順序傳回，而非按時間戳記排序。您可以將此 JSON 資料匯入圖表工具，以進行排序和視覺化。

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_cluster_id
{
    "Label": "VolumeBytesUsed",
    "Datapoints": [
        {
            "Timestamp": "2020-08-04T19:40:00+00:00",
            "Maximum": 182872522752.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-05T00:40:00+00:00",
            "Maximum": 198573719552.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-05T05:40:00+00:00",
            "Maximum": 206827454464.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-04T17:40:00+00:00",
            "Maximum": 182872522752.0,
            "Unit": "Bytes"
        },
... output omitted ...
```

此範例會傳回與前一個相同的資料。--output 參數表示精簡純文字格式的資料。aws cloudwatch 命令會將其輸出傳送至 sort 命令。-k 命令的 sort 參數會依第三個欄位排序輸出，也就是國際標準時間 (UTC) 格式的時間戳記。

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600 \
```

```
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_cluster_id \
  --output text | sort -k 3
VolumeBytesUsed
DATAPOINTS  182872522752.0  2020-08-04T17:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T18:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T19:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T20:41:00+00:00 Bytes
DATAPOINTS  187667791872.0  2020-08-04T21:41:00+00:00 Bytes
DATAPOINTS  190981029888.0  2020-08-04T22:41:00+00:00 Bytes
DATAPOINTS  195587244032.0  2020-08-04T23:41:00+00:00 Bytes
DATAPOINTS  201048915968.0  2020-08-05T00:41:00+00:00 Bytes
DATAPOINTS  205368492032.0  2020-08-05T01:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T02:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T03:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T04:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T05:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T06:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T07:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T08:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T09:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T10:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T11:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T12:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T13:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T14:41:00+00:00 Bytes
DATAPOINTS  206833664000.0  2020-08-05T15:41:00+00:00 Bytes
DATAPOINTS  206833664000.0  2020-08-05T16:41:00+00:00 Bytes
```

排序的輸出會顯示監控期間開始和結束時使用了多少儲存空間。當 Aurora 為叢集配置更多儲存體時，您也可以在該期間找到這些點。下列範例使用 Linux 命令，將開始和結束的 VolumeBytesUsed 值重新格式化為 Gigabyte (GB) 和 Gibibyte (GiB)。GB 代表以 10 的乘冪計算單位，通常用於討論旋轉式硬碟的儲存裝置。GiB 代表以 2 的乘冪計算的單位。Aurora 儲存量測和限制通常以 2 的乘冪為表示單位，例如 GiB 和 TiB。

```
$ GiB=$((1024*1024*1024))
$ GB=$((1000*1000*1000))
$ echo "Start: $((182872522752/$GiB)) GiB, End: $((206833664000/$GiB)) GiB"
Start: 170 GiB, End: 192 GiB
$ echo "Start: $((182872522752/$GB)) GB, End: $((206833664000/$GB)) GB"
Start: 182 GB, End: 206 GB
```

VolumeBytesUsed 指標會告訴您叢集中有多少儲存體會產生費用。因此，最好在實用的情況下盡量降低此數字。不過，此指標不包含 Aurora 在叢集內部使用且不收費的部分儲存體。如果您的叢集已接近儲存限制且空間不足，則監控 AuroraVolumeBytesLeftTotal 指標並嘗試將該數字最大化會更有幫助。下列範例會執行與前一個類似的計算，但它是針對 AuroraVolumeBytesLeftTotal 而非 VolumeBytesUsed。

```
$ aws cloudwatch get-metric-statistics --metric-name "AuroraVolumeBytesLeftTotal" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_old_cluster_id \
  --output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS      140530528288768.0       2023-02-23T19:25:00+00:00       Count
$ TiB=$((1024*1024*1024*1024))
$ TB=$((1000*1000*1000*1000))
$ echo "$((69797067915264 / $TB)) TB remaining for this cluster"
69 TB remaining for this cluster
$ echo "$((69797067915264 / $TiB)) TiB remaining for this cluster"
63 TiB remaining for this cluster
```

對於執行 Aurora MySQL 2.09 版或更高版本，或 Aurora PostgreSQL 的叢集，新增資料時，VolumeBytesUsed 報告的可用大小會增加，移除資料時則會減少。下列範例會顯示作法。此報告顯示每隔 15 分鐘叢集的最大和最小儲存體大小，因為會建立和刪除含有暫存資料的表格。此報告會在最小值之前列出最大值。因此，若要了解儲存體使用量在 15 分鐘間隔內的變更，請從右至左解讀數字。

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '4 hours ago')" --end-time "$(date -d 'now')" --period 1800 \
  --namespace "AWS/RDS" --statistics Maximum Minimum --dimensions
 Name=DBClusterIdentifier,Value=my_new_cluster_id
  --output text | sort -k 4
VolumeBytesUsed
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T20:49:00+00:00 Bytes
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T21:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 14545305600.0 2020-08-05T21:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 15614263296.0 2020-08-05T23:19:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-05T23:49:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-06T00:19:00+00:00 Bytes
```

下列範例針對執行 Aurora MySQL 或 Aurora PostgreSQL 相容版本的叢集，顯示 `AuroraVolumeBytesLeftTotal` 報告的可用大小如何反映 256 TiB 大小限制。如需關於相容版本的詳細資訊，請參閱 Amazon Aurora 大小限制。

```
$ aws cloudwatch get-metric-statistics --region us-east-1 --metric-name
 "AuroraVolumeBytesLeftTotal" \
  --start-time "$(date -d '4 hours ago')" --end-time "$(date -d 'now')" --period 1800 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBClusterIdentifier,Value=pq-57 \
  --output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS 140515818864640.0 2020-08-05T20:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:26:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:56:00+00:00 Count
DATAPOINTS 140514866757632.0 2020-08-05T22:26:00+00:00 Count
DATAPOINTS 140511020580864.0 2020-08-05T22:56:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:26:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-06T00:26:00+00:00 Count
$ TiB=$((1024*1024*1024*1024))
$ TB=$((1000*1000*1000*1000))
$ echo "$((140515818864640 / $TB)) TB remaining for this cluster"
140 TB remaining for this cluster
$ echo "$((140515818864640 / $TiB)) TiB remaining for this cluster"
256 TiB remaining for this cluster
```

執行個體擴展

您可以修改資料庫叢集中每個資料庫執行個體的資料庫執行個體類別，視需要擴展 Aurora 資料庫叢集。視資料庫引擎相容性而定，Aurora 支援數個針對 Aurora 最佳化的資料庫執行個體類別。

資料庫引擎	執行個體擴展
Amazon Aurora MySQL	請參閱 擴展 Aurora MySQL 資料庫執行個體
Amazon Aurora PostgreSQL	請參閱 擴展 Aurora PostgreSQL 資料庫執行個體

讀取擴展

您可以在資料庫叢集中建立最多 15 個 Aurora 複本，以達成 Aurora 資料庫叢集的讀取擴展。在主要執行個體寫入更新之後，每個 Aurora 複本會以最短的複本延遲從叢集磁碟區傳回相同的資料，通常遠低於 100 毫秒。當讀取流量增加時，您可以建立更多 Aurora 複本，並直接連線這些複本，以分散資料庫叢集的讀取負載。Aurora 複本的資料庫執行個體類別不必與主要執行個體相同。

如需新增 Aurora 複本到資料庫叢集的詳細資訊，請參閱將 Aurora 複本新增至資料庫叢集。

管理連線

允許對 Aurora 資料庫執行個體建立的連線數上限，由資料庫執行個體的執行個體層級參數群組中的 `max_connections` 參數決定。根據用於資料庫執行個體的資料庫執行個體類別及資料庫引擎相容性，此參數的預設值有所不同。

資料庫引擎	max_connections 預設值
Amazon Aurora MySQL	請參閱 對 Aurora MySQL 資料庫執行個體的連線數上限
Amazon Aurora PostgreSQL	請參閱 對 Aurora PostgreSQL 資料庫執行個體的連線數上限

> **Tip**
>
> 如果您的應用程式經常開啟和關閉連線，或者保持大量長期連線開啟，我們建議您使用 Amazon RDS Proxy。RDS 代理是個完全受管、高可用性的資料庫代理，其使用連線集區，安全且高效地共用資料庫連線。如要進一步了解 RDS 代理，請參閱 Amazon RDS Proxy for Aurora。

管理查詢執行計劃

如果您使用的是 Aurora PostgreSQL 的查詢計劃管理，您能夠控制最佳化工具將執行哪些計劃。如需更多詳細資訊，請參閱 管理 Aurora PostgreSQL 的查詢執行計劃。

複製 Amazon Aurora 資料庫叢集的一個磁碟區

透過使用 Aurora 複製，您可以建立最初與原始叢集共用相同資料頁面，但為個別且獨立磁碟區的新叢集。該程序旨在快速且具有成本效益。與其相關聯的資料磁碟區新叢集稱為複製。與使用不同的技術 (如還原快照) 以物理方式複製資料相比，建立複製更快也更節省空間。

主題

- Aurora 複製的概觀
- Aurora 複製的限制
- Aurora 複製的運作方法
- 建立 Amazon Aurora 複製
- 使用 Amazon Aurora 進行跨 VPC 複製
- 使用 AWS RAM 和 Amazon Aurora 進行跨帳戶複製

Aurora 複製的概觀

Aurora 使用寫入時複製通訊協定來建立複製。此機制會使用最少的其他空間來建立初始複製。首次建立複製時，Aurora 會保留來源 Aurora 資料庫叢集和新 (複製) Aurora 資料庫叢集所使用的資料單一副本。只有在來源 Aurora 資料庫叢集或 Aurora 資料庫叢集複製 (在 Aurora 儲存磁碟區上) 變更資料時，才會配置其他的儲存體。若要進一步了解寫入時複製通訊協定，請參閱 Aurora 複製的運作方法。

Aurora 複製對於使用您的生產資料快速設定測試環境特別有用，且不會造成資料損毀風險。您可以為許多類型的應用程式使用複製，例如以下類型：

- 潛在變更實驗 (例如結構描述變更和參數群組變更)，以評估所有影響。
- 執行工作負載密集的操作，例如在複製上匯出資料或執行分析查詢。
- 為了開發、測試或其他用途建立生產資料庫叢集的副本。

您可以從同一個 Aurora 資料庫叢集建立多個複製。您也可以從另一個複製建立多個複製。

在建立 Aurora 複製之後，您可以將 Aurora 資料庫執行個體設定為與來源 Aurora 資料庫叢集不同。例如，您可能不需要出於開發目的的複製，來滿足與來源生產 Aurora資料庫叢集相同的高可用性需求。在此情況下，您可以使用單一 Aurora 資料庫執行個體來設定複製，而不是利用 Aurora 資料庫叢集所用的多個資料庫執行個體來設定。

在您使用與來源不同的方部署組態建立複製時，會使用來源之 Aurora DB 引擎的最新次要版本建立複製。

在您從 Aurora 資料庫叢集建立複製時，該複製會建立在您的 AWS 帳戶 (即與擁有來源 Aurora 資料庫叢集相同的帳戶) 中。然而，您也可以與其他 AWS 帳戶共用 Aurora Serverless v2 和已佈建的 Aurora 資料庫叢集和複製。如需更多詳細資訊，請參閱 使用 AWS RAM 和 Amazon Aurora 進行跨帳戶複製。

在您完成使用複製來進行測試、開發或其他用途時，便可將其刪除。

Aurora 複製的限制

Aurora 複製限制如下：

- 您可以根據需要建立數量不拘的複製項，最多可達到 AWS 區域 中允許的資料庫叢集數量上限。
- 您可以使用寫入時複製通訊協定建立最多 15 個複製。在具有 15 個複製項之後，下一個您建立的複製項為完整複本。完整複製通訊協定的作用類似於時間點復原。
- 您無法在與來源 Aurora 資料庫叢集不同的 AWS 區域中建立複製項。
- 您無法從沒有平行查詢功能的 Aurora 資料庫叢集建立一個複製到使用平行查詢的叢集中。若要將資料引入使用平行查詢的叢集，請建立原始叢集的快照，然後將其還原到使用平行查詢功能的叢集。
- 您無法從沒有資料庫執行個體的 Aurora 資料庫叢集中建立複製。您僅能複製至少具有一個資料庫執行個體的 Aurora 資料庫叢集。
- 您可以在與 Aurora 資料庫叢集不同的 Virtual Private Cloud (VPC) 中建立複製。不過，VPC 中的子網路必須映射至相同的可用區域。
- 您可以從已佈建的 Aurora 資料庫叢集中建立 Aurora 已佈建複製。
- 具有 Aurora Serverless v2 執行個體的叢集遵循與佈建叢集相同的規則。
- 在 Aurora Serverless v1 中：
  - 您可以從 Aurora Serverless v1 資料庫叢集建立已佈建的複製。
  - 您可以從 Aurora Serverless v1 或已佈建的資料庫叢集建立 Aurora Serverless v1 複製。
  - 您無法從未加密的已佈建 Aurora 資料庫叢集中建立 Aurora Serverless v1 複製。
  - 跨帳戶複製目前佈支援複製 Aurora Serverless v1 資料庫叢集。如需更多詳細資訊，請參閱 跨帳戶複製的限制。
  - 複製的 Aurora Serverless v1 資料庫叢集具有與任何 Aurora Serverless v1 資料庫叢集相同的行為和限制。如需更多詳細資訊，請參閱 使用 Amazon Aurora Serverless v1。
  - Aurora Serverless v1 資料庫叢集一律會加密。在您複製 Aurora Serverless v1 資料庫叢集至已佈建的 Aurora 資料庫叢集中時，已佈建的 Aurora 資料庫叢集會加密。您可以選擇加密金鑰，但無

法停用加密。若要從已佈建的 Aurora 資料庫叢集複製至 Aurora Serverless v1，您必須先從已佈建已加密且佈建的 Aurora 資料庫叢集開始。

Aurora 複製的運作方法

Aurora 複製會在 Aurora 資料庫叢集的儲存層運作。其會使用寫入時複製通訊協定，在支援 Aurora 儲存磁碟區的基礎耐用媒體方面，既快速又節省空間。您可以在 Amazon Aurora 儲存體的概觀 中進一步了解 Aurora 叢集磁碟區。

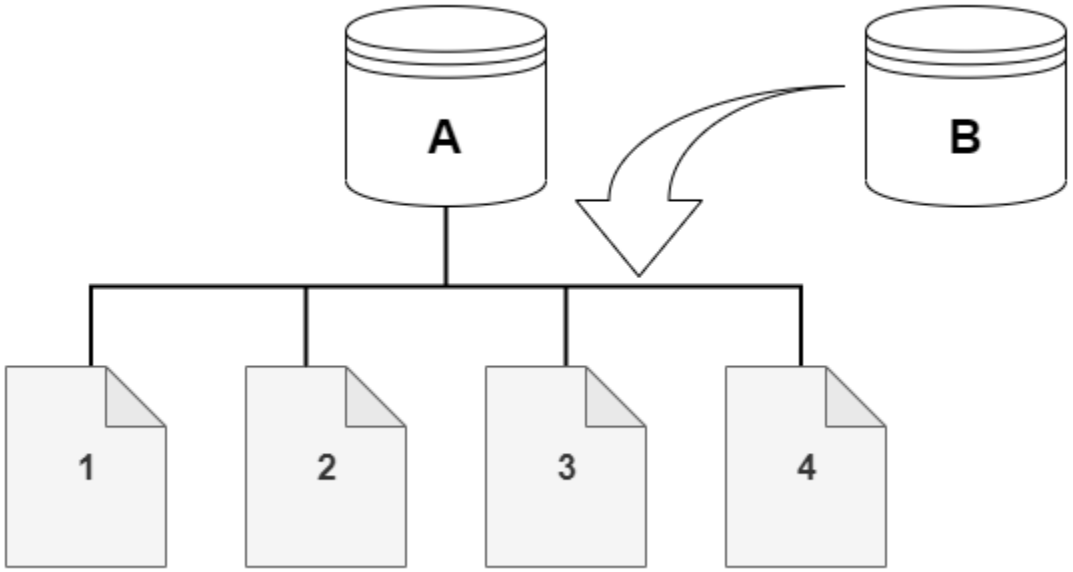
主題

- 了解寫入時複製通訊協定
- 刪除來源叢集磁碟區

了解寫入時複製通訊協定

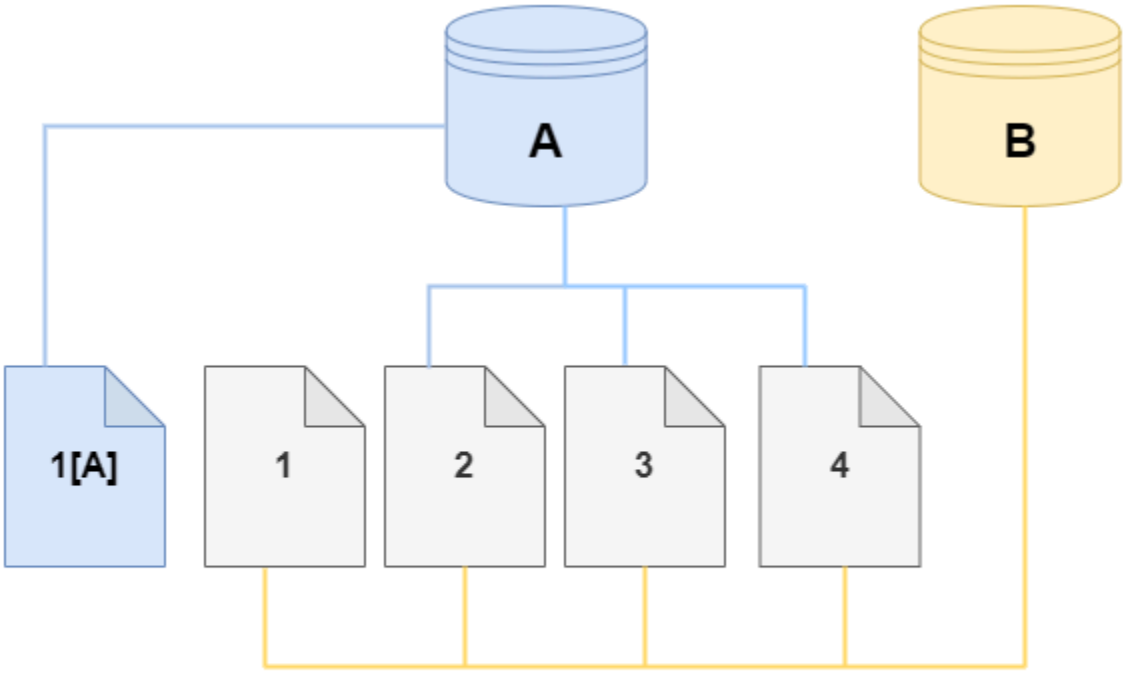
Aurora 資料庫叢集會將資料儲存在基礎 Aurora 儲存磁碟區的頁面中。

例如，在以下圖表中，您可以找到具有 4 個資料頁面 (1、2、3 和 4) 的 Aurora資料庫叢集 (A)。想像一下，複製 B 是從 Aurora資料庫叢集建立。建立複製時，不會複製任何資料。相反地，複製會指向與來源 Aurora 資料庫叢集相同的一組頁面。

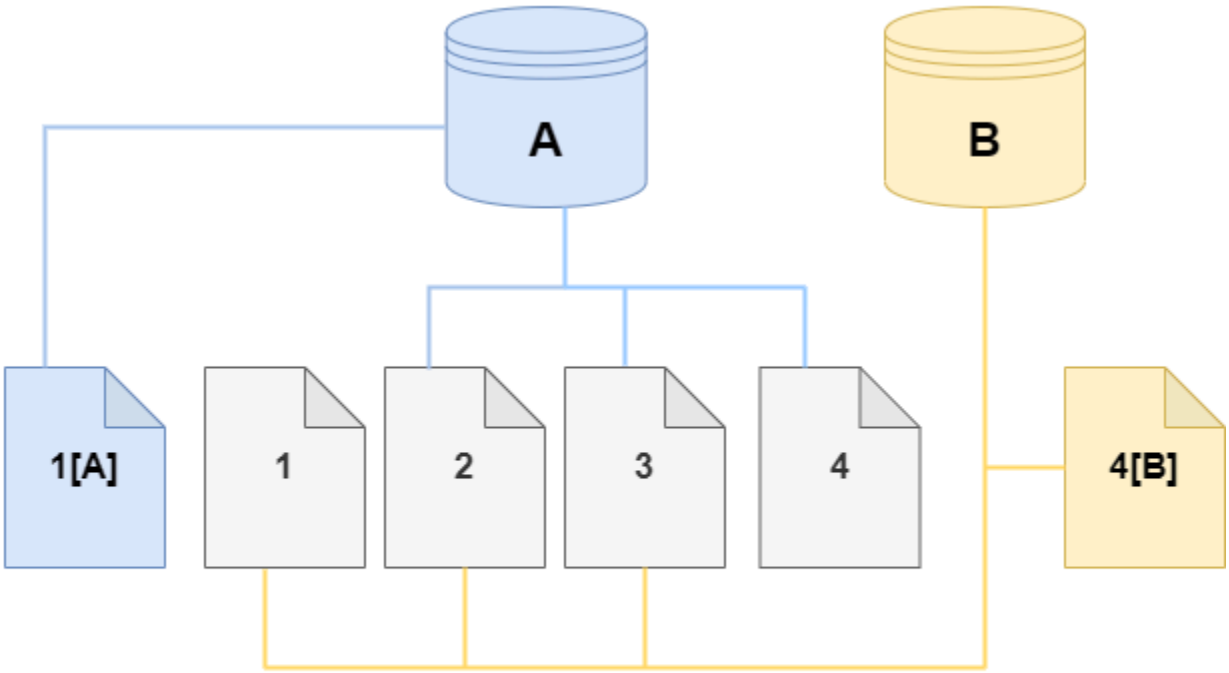


在建立複製時，通常不需要其他的儲存空間。寫入時複製通訊協定會使用與實體儲存媒體上相同的區段，作為來源區段。只有在來源區段的容量不足以容納整個複製區段時，才需要其他的儲存空間。如果是這種情況，來源區段會複製到另一個實體裝置上。

在下圖中，您可以找到使用相同叢集 A 及其複製 B 之動作的寫入時複製通訊協定範例，如上所示。假設您變更 Aurora 資料庫叢集 (A)，導致第 1 頁保留的資料變更。Aurora 不會寫入至原始第 1 頁，而是建立新的第 1 頁 [A]。叢集 (A) 的 Aurora 資料庫叢集磁碟區現在會指向第 1 頁 [A]、2、3 和 4 頁，而複製 (B) 仍會參考原始頁面。



至於複製上，已對儲存磁碟區上的第 4 頁進行變更。Aurora 不會寫入至原始第 4 頁，而是建立新的第 4 頁 [B]。複製現在會指向第 1、2、3 和 4 頁 [B]，而叢集 (A) 會繼續指向 1 [A]、2、3 和 4。



隨著來源 Aurora 資料庫叢集磁碟區和複製在一段時間後發生更多變更，您需要不斷增加更多的儲存空間來擷取並存放變更。

刪除來源叢集磁碟區

最初，複製磁碟區與建立複製的原始磁碟區共用相同的資料頁面。只要原始磁碟區存在，複製磁碟區只會被視為建立或修改複製之頁面的擁有者。因此，複製磁碟區的 `VolumeBytesUsed` 指標會從小處開始，並且只會隨著原始叢集與複製之間的資料差異而成長。對於來源磁碟區和複製之間的頁面，儲存費用僅適用於原始叢集。如需關於 `VolumeBytesUsed` 指標的詳細資訊，請參閱Amazon Aurora 的叢集層級指標。

在您刪除具有一個或多個與其相關聯複製的來源叢集磁碟區時，複製的叢集磁碟區中的資料不會變更。Aurora 會保留先前由來源叢集磁碟區擁有的頁面。Aurora 會重新分配已刪除叢集所擁有之頁面的儲存體計費。例如，假設原始叢集有兩個複製，然後刪除原始叢集。原始叢集擁有的一半資料頁面現在將由一個複製擁有。另一半的頁面將由另一個複製擁有。

如果您刪除原始叢集，則在建立或刪除更多複製時，Aurora 會繼續在共用相同頁面的所有複製之間重新分配資料頁面的擁有權。因此，您可能會發現複製叢集磁碟區的 `VolumeBytesUsed` 指標值變更。

隨著建立更多複製，且頁面擁有權分散到更多叢集，指標值可能會減少。刪除複製並將頁面擁有權指派給較少數量的叢集時，指標值也會增加。如需寫入操作如何影響複製磁碟區上資料頁面的資訊，請參閱了解寫入時複製通訊協定。

當原始叢集和複製由相同AWS帳戶擁有時，這些叢集的所有儲存費用都會套用至該相同 AWS 帳戶。如果某些叢集是跨帳戶複製，刪除原始叢集可能會對擁有跨帳戶複製的 AWS 帳戶產生額外的儲存費用。

例如，假設叢集磁碟區在您建立任何複製之前有 1000 個已使用的資料頁面。當您複製該叢集時，複製磁碟區一開始使用的頁面為零。如果複製對 100 個資料頁面進行修改，則只會將這 100 個頁面存放在複製磁碟區並標記為已使用。兩個叢集會共用父磁碟區中的其他 900 個未變更頁面。在此情況下，父叢集會有 1000 頁的儲存費用，以及 100 頁的複製磁碟區費用。

如果您刪除來源磁碟區，複製的儲存費用會包含其變更的 100 頁，以及原始磁碟區的 900 個共用頁面，總共 1000 頁。

建立 Amazon Aurora 複製

您可以在相同的 AWS 帳戶中建立複製，作為來源 Aurora 資料庫叢集。若要執行此操作，您可以使用 AWS 管理主控台 或 AWS CLI，並依下列程序進行。

若要允許另一個 AWS 帳戶來建立複製，或與其他 AWS 帳戶共享，請使用 使用 AWS RAM 和 Amazon Aurora 進行跨帳戶複製 中的程序。

主控台

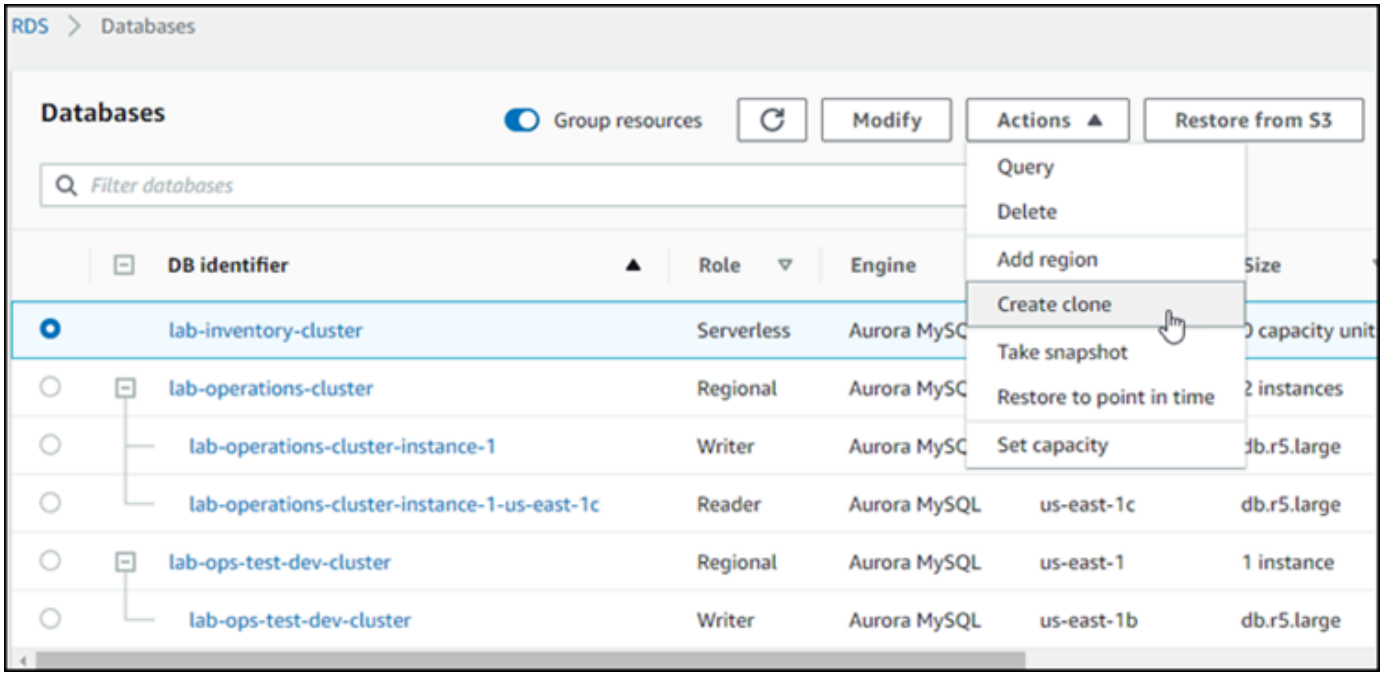
以下程序說明如何使用 複製 AWS 管理主控台 來複製 Aurora 資料庫叢集。

在 Aurora 資料庫叢集中搭配使用 AWS 管理主控台 結果與一個 Aurora 資料庫執行個體，以建立複製。

這些說明適用於與建立複製 AWS 帳戶相同帳戶所擁有的資料庫叢集。若資料庫叢集是由不同 AWS 帳戶擁有，請改為參閱使用 AWS RAM 和 Amazon Aurora 進行跨帳戶複製。

使用 AWS 建立由您 AWS 管理主控台 帳戶所擁有資料庫叢集的複製

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 從清單中選擇您的 Aurora 資料庫叢集，並為 Actions (動作)，選擇 Create clone (建立複製)。



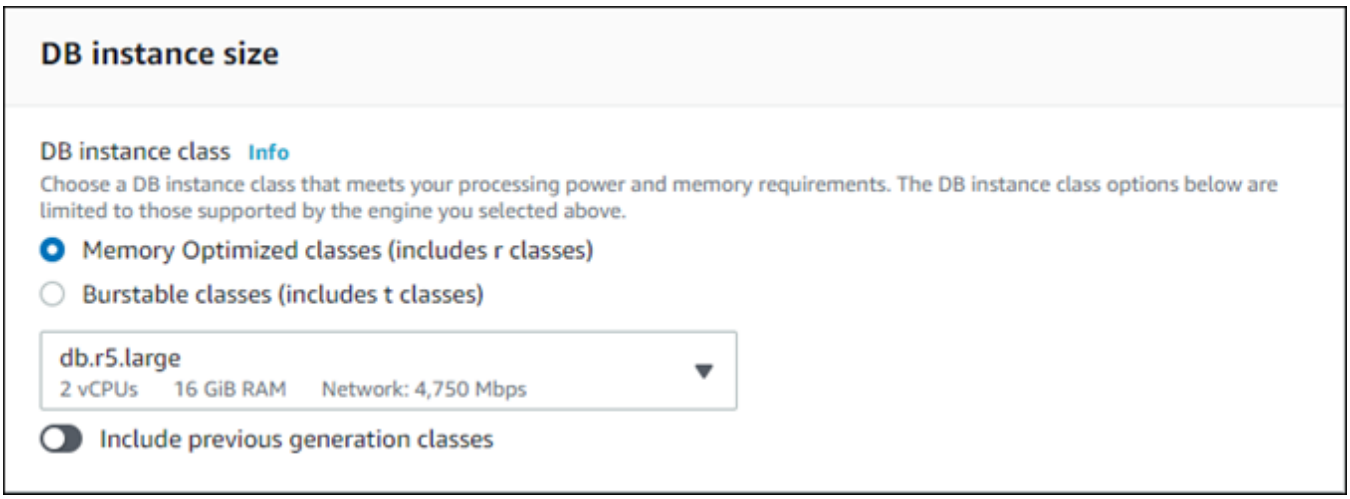
「建立複製」頁面即會開啟，您可以在其中設定 Aurora 資料庫叢集複製的設定、連線能力和其他選項。

4. 對於資料庫執行個體識別符，請輸入您要給予所複製 Aurora 資料庫叢集的名稱。
5. 對於 Aurora Serverless v1 資料庫叢集，針對容量類型選擇已佈建或無伺服器。

   僅在來源 Aurora 資料庫叢集為 Aurora Serverless v1 資料庫叢集，或為已加密之已佈建 Aurora 資料庫叢集時，才可選擇 Serverless (無伺服器)。
6. 對於 Aurora Serverless v2 或已佈建的資料庫叢集，針對叢集儲存組態選擇 Aurora I/O-Optimized 或 Aurora Standard。

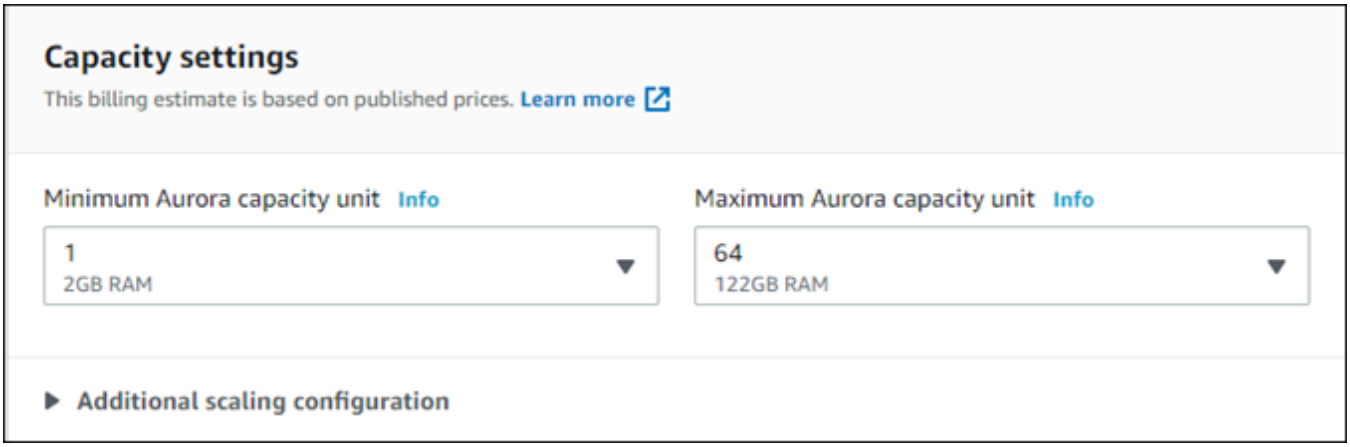
   如需更多詳細資訊，請參閱 Amazon Aurora 資料庫叢集的儲存組態。
7. 選擇資料庫執行個體大小或資料庫叢集容量：

   - 對於已佈建的複製，選擇資料庫執行個體類別。



您可以接受提供的設定，也可以為您的複製使用不同的資料庫執行個體類別。

- 針對 Aurora Serverless v1 或 Aurora Serverless v2 複製，選擇容量設定。



您可以接受提供的設定，也可以為您的複製變更設定。

8. 視需要為您的複製選擇其他設定。若要進一步了解 Aurora 資料庫叢集和執行個體設定，請參閱 建立 Amazon Aurora 資料庫叢集。
9. 選擇建立複製。

在建立複製時，其會與您其他的 Aurora 資料庫叢集一起列於主控台的 Databases (資料庫) 區段中，並顯示其目前狀態。在您複製的狀態為 Available (可用) 時，表示已可使用。

AWS CLI

使用 AWS CLI 複製您的 Aurora 資料庫叢集涉及建立複製叢集，並將一或多個資料庫執行個體新增至其中的幾個步驟。

您使用的 `restore-db-cluster-to-point-in-time` AWS CLI 命令會產生 Aurora 資料庫叢集，其儲存資料與原始叢集相同，但沒有 Aurora 資料庫執行個體。複製可用後，您可以分別建立資料庫執行個體。您可以選擇資料庫執行個體數量及其執行個體類別，讓複製的運算容量比原始叢集更多或更少。程序的步驟如下：

1. 透過使用 restore-db-cluster-to-point-in-time CLI 命令來建立複製。
2. 使用 create-db-instance CLI 命令建立複製的寫入器資料庫執行個體。
3. (選用) 執行其他 create-db-instance CLI 命令，將一或多個讀取器執行個體新增至複製叢集。使用讀取器執行個體有助於改善複製的高可用性和讀取可擴展性方面。如果您只打算使用複製進行開發和測試，可以略過此步驟。

主題

- 建立複製
- 檢查狀態並取得複製詳細資訊
- 建立您複製的 Aurora 資料庫執行個體
- 用於複製的參數

建立複製

使用 `restore-db-cluster-to-point-in-time` CLI 命令來建立初始複製叢集。

若要從來源 Aurora 資料庫叢集建立複製

- 使用 `restore-db-cluster-to-point-in-time` CLI 命令。指定下列參數的值。在此典型情況下，複製會使用與原始叢集相同的引擎模式，無論是已佈建或 Aurora Serverless v1。
  - `--db-cluster-identifier` – 為您的複製選擇一個有意義的名稱。在您使用 restore-db-cluster-to-point-in-time CLI 命令時，為複製命名。然後，您會在 create-db-instance CLI 命令中傳遞複製的名稱。
  - `--restore-type` – 使用 `copy-on-write` 來建立來源資料庫叢集的複製。如果沒有這個參數，`restore-db-cluster-to-point-in-time` 會還原 Aurora 資料庫叢集，而非建立複製。
  - `--source-db-cluster-identifier` – 使用您要複製之來源 Aurora 資料庫叢集的名稱。
  - `--use-latest-restorable-time` – 此值會指向來源資料庫叢集的最新可還原磁碟區資料。使用它來建立複製。

下列範例會從名為 my-source-cluster 的叢集建立名為 my-clone 的複製。

對於 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

在 Windows 中：

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

該命令會返回包含複製詳細資訊的 JSON 對象。在嘗試為複製建立資料庫執行個體之前，請檢查以確定您複製的資料庫叢集是否可用。如需更多詳細資訊，請參閱 檢查狀態並取得複製詳細資訊。

例如，假設您有一個想要複製的叢集名為 tpch100g。下列 Linux 範例會為新叢集建立名為 tpch100g-clone 的複製叢集、名為 tpch100g-clone-instance 的 Aurora Serverless v2 寫入器執行個體，以及名為 tpch100g-clone-instance-2 的已佈建讀取器執行個體。

您不需要提供部分參數，例如 --master-username 和 --master-user-password。Aurora 會自動從原始叢集中判斷這些參數。您確實需要指定要使用的資料庫引擎。因此，此範例會測試新叢集，以判斷 --engine 參數所要使用的正確值。

此範例也包含建立複製叢集時的 --serverless-v2-scaling-configuration 選項。如此一來，即使原始叢集未使用 Aurora Serverless v2，您也可以將 Aurora Serverless v2 執行個體新增至複製。

```
$ aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier tpch100g \
  --db-cluster-identifier tpch100g-clone \
  --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16 \
  --restore-type copy-on-write \
  --use-latest-restorable-time

$ aws rds describe-db-clusters \
  --db-cluster-identifier tpch100g-clone \
    --query '*[].[Engine]' \
```

```
    --output text
aurora-mysql

$ aws rds create-db-instance \
  --db-instance-identifier tpch100g-clone-instance \
  --db-cluster-identifier tpch100g-clone \
  --db-instance-class db.serverless \
  --engine aurora-mysql

$ aws rds create-db-instance \
  --db-instance-identifier tpch100g-clone-instance-2 \
  --db-cluster-identifier tpch100g-clone \
  --db-instance-class db.r6g.2xlarge \
  --engine aurora-mysql
```

建立與來源 Aurora 資料庫叢集不同的引擎類型複製。

- 此程序僅適用於支援 Aurora Serverless v1 的較舊引擎版本。假設您擁有 Aurora Serverless v1 叢集，而且想要建立屬於已佈建叢集的複製。在這種情況下，請使用 `restore-db-cluster-to-point-in-time` CLI 命令，並指定與先前範例中類似參數值的值，加上下列其他參數：
  - `--engine-mode` – 只能使用此參數來建立與來源 Aurora 資料庫叢集不同引擎模式的複製。此參數僅適用於支援 Aurora Serverless v1 的較舊引擎版本。選擇要與以下 `--engine-mode` 一起傳遞的值：
    - 使用 `--engine-mode provisioned` 從 Aurora Serverless 資料庫叢集建立已佈建的 Aurora 資料庫叢集複製。

      > **Note**
      >
      > 如果您想要搭配從 Aurora Serverless v1 複製的叢集使用 Aurora Serverless v2，您仍然會將複製的引擎模式指定為 `provisioned`。接著您會執行額外的升級和遷移步驟。

    - 使用 `--engine-mode serverless` 從已佈建的 Aurora 資料庫叢集建立 Aurora Serverless v1 複製。在您指定 `serverless` 引擎模式時，也可以選擇 `--scaling-configuration`。
  - `--scaling-configuration` – (選用) 與 `--engine-mode serverless` 搭配使用，以設定 Aurora Serverless v1 複製的容量下限和上限。如果您不使用此參數，Aurora 會使用資料庫引擎的預設 Aurora Serverless v1 容量值來建立 Aurora Serverless v1 複製。

以下範例會從名為 `my-source-cluster` 的 Aurora Serverless v1 資料庫叢集來建立名為 `my-clone` 的已佈建複製。

對於 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --engine-mode provisioned \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

在 Windows 中：

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --engine-mode provisioned ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

這些命令會返回 JSON 對象，其會包含您需要用來建立資料庫執行個體的複製。在複製的狀態 (空白 Aurora 資料庫叢集) 的狀態為 可用 (Available) 之前，您無法執行該操作。

> **Note**
>
> restore-db-cluster-to-point-in-time AWS CLI 命令只會還原資料庫叢集，而不會還原該資料庫叢集的資料庫執行個體。若要為已還原的資料庫叢集建立資料庫執行個體，請執行 create-db-instance 命令。使用該命令，您可以將還原的資料庫叢集識別符指定為 `--db-cluster-identifier` 參數。只在 `restore-db-cluster-to-point-in-time` 命令完成且資料庫叢集可用時，您才能建立資料庫執行個體。
>
> 假設您從 Aurora Serverless v1 叢集開始，並打算將其遷移至 Aurora Serverless v2 叢集。您可以建立Aurora Serverless v1 叢集的已佈建複製，做為遷移中的初始步驟。如需完整程序，包括任何必要的版本升級，請參閱 從 Aurora Serverless v1 叢集升級至 Aurora Serverless v2。

檢查狀態並取得複製詳細資訊

您可以使用下列命令，來檢查您新建立之複製叢集的狀態。

```
$ aws rds describe-db-clusters --db-cluster-identifier my-clone --query '*[].[Status]'
 --output text
```

或者，您可以使用以下 AWS CLI 查詢，來取得您建立複製的資料庫執行個體時所需的狀態和其他值。

對於 Linux、macOS 或 Unix：

```
aws rds describe-db-clusters --db-cluster-identifier my-clone \
  --query '*[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}'
```

在 Windows 中：

```
aws rds describe-db-clusters --db-cluster-identifier my-clone ^
  --query "*[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}"
```

此查詢會返回類似以下的內容：

```
[
  {
        "Status": "available",
        "Engine": "aurora-mysql",
        "EngineVersion": "8.0.mysql_aurora.3.04.1",
        "EngineMode": "provisioned"
    }
]
```

建立您複製的 Aurora 資料庫執行個體

使用 create-db-instance CLI 命令來建立您的 Aurora Serverless v2 或已佈建複製的資料庫執行個體。您不會為 Aurora Serverless v1 複製建立資料庫執行個體。

資料庫執行個體會從來源資料庫叢集繼承 `--master-username` 和 `--master-user-password` 屬性。

下列範例會建立已佈建複製的資料庫執行個體。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
    --db-instance-identifier my-new-db \
```

```
    --db-cluster-identifier my-clone \
    --db-instance-class db.r6g.2xlarge \
    --engine aurora-mysql
```

在 Windows 中：

```
aws rds create-db-instance ^
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class db.r6g.2xlarge ^
    --engine aurora-mysql
```

下列範例會為使用支援 Aurora Serverless v2 之引擎版本的複製建立 Aurora Serverless v2 資料庫執行個體。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
    --db-instance-class db.serverless \
  --engine aurora-postgresql
```

在 Windows 中：

```
aws rds create-db-instance ^
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class db.serverless ^
    --engine aurora-mysql
```

用於複製的參數

下表摘要說明與 `restore-db-cluster-to-point-in-time` 搭配使用以複製 Aurora 資料庫叢集的各種參數。

參數	描述
`--source-db-cluster-identifier`	使用您要複製之來源 Aurora 資料庫叢集的名稱。

參數	描述
`--db-cluster-identifier`	當您使用 `restore-db-cluster-to-point-in-time` 命令建立複製時，為您的複製選擇有意義的名稱。然後，您將此名稱傳遞給 `create-db-instance` 命令。
`--restore-type`	指定 `copy-on-write` 為 `--restore-type` ，以建立來源資料庫叢集的複製，而非還原來源 Aurora 資料庫叢集。
`--use-latest-restorable-time`	此值會指向來源資料庫叢集的最新可還原磁碟區資料。使用它來建立複製。
`--serverless-v2-scaling-configuration`	(支援 Aurora Serverless v2 的較新版本) 使用此參數來設定 Aurora Serverless v2 複製的最小和最大容量。如果您未指定此參數，則在修改叢集以新增此屬性之前，您無法在複製叢集中建立任何 Aurora Serverless v2 執行個體。
`--engine-mode`	(僅支援 Aurora Serverless v1 的舊版本) 使用此參數建立與來源 Aurora 資料庫叢集不同類型的複製，並具有下列其中一個值： • 使用 `provisioned` 從 Aurora Serverless v1 資料庫叢集建立已佈建的複製。 • 使用 `serverless` 從已佈建或 Aurora Serverless v2 資料庫叢集建立 Aurora Serverless v1 複製。 在您指定 `serverless` 引擎模式時，也可以選擇 `--scaling-configuration` 。
`--scaling-configuration`	(僅支援 Aurora Serverless v1 的舊版本) 使用此參數來設定 Aurora Serverless v1 複製的最小和最大容量。如果您不指定此參數，Aurora 會使用資料庫引擎的預設容量值來建立複製。

如需跨 VPC 和跨帳戶複製的資訊，請參閱下列各節。

主題

- 使用 Amazon Aurora 進行跨 VPC 複製
- 使用 AWS RAM 和 Amazon Aurora 進行跨帳戶複製

使用 Amazon Aurora 進行跨 VPC 複製

假設您想要對原始叢集和複製施加不同的網路存取控制。例如，您可以使用複製來建立不同 VPC 中的生產 Aurora 叢集的複本，以進行開發和測試。或者，您可以在從公有子網路遷移到私有子網路的過程中建立複製，以增強資料庫安全性。

下列各節示範如何設定複製的網路組態，以便原始叢集和複製都可以存取相同的 Aurora 儲存節點，即使是來自不同的子網路或不同的 VPC。事先確認網路資源可以避免複製期間可能難以診斷的錯誤。

如果您不熟悉 Aurora 如何與 VPC、子網路和資料庫子網路群組互動，請先參閱 Amazon VPC 和 Amazon Aurora。您可以完成該部分中的教學課程，在 AWS 主控台中建立這些類型的資源，並了解它們如何結合在一起。

由於這些步驟涉及在 RDS 和 EC2 服務之間切換，因此範例使用 AWS CLI 命令來協助您了解如何自動化此類操作並儲存輸出。

主題

- 開始之前
- 收集網路環境的相關資訊
- 建立複製的網路資源
- 使用新的網路設定建立 Aurora 複製
- 將叢集從公有子網路移至私有子網路
- 建立跨 VPC 複製的端對端範例

開始之前

開始設定跨 VPC 複製之前，請務必擁有下列資源：

- 用作複製來源的 Aurora 資料庫叢集。如果這是您第一次建立 Aurora 資料庫叢集，請參閱 Amazon Aurora 入門 中的教學課程，以使用 MySQL 或 PostgreSQL 資料庫引擎設定叢集。
- 如果您想要建立跨 VPC 複製，則為第二個 VPC。如果您沒有用於複製的 VPC，請參閱 教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4) 或 教學課程：建立要與資料庫叢集搭配使用的 (VPC)(雙堆疊模式)。

收集網路環境的相關資訊

使用跨 VPC 複製時，網路環境在原始叢集及其複製之間可能大不相同。建立複製之前，請收集並記錄原始叢集中使用的 VPC、子網路、資料庫子網路群組和 AZ 的相關資訊。如此一來，您就可以將發生問題的機率降至最低。如果發生網路問題，您不必中斷任何故障診斷活動，即可搜尋診斷資訊。下列各節顯示收集這些類型資訊的 CLI 範例。您可以在建立複製和進行任何故障診斷時，以方便查閱的任何格式儲存詳細資訊。

- 步驟 1：檢查原始叢集的可用區域
- 步驟 2：檢查原始叢集的資料庫子網路群組
- 步驟 3：檢查原始叢集的子網路
- 步驟 4：檢查原始叢集中資料庫執行個體的可用區域
- 步驟 5：檢查您可以用於複製的 VPC

步驟 1：檢查原始叢集的可用區域

建立複製之前，確認原始叢集用於其儲存的 AZ。如 Amazon Aurora 儲存體 中所述，每個 Aurora 叢集的儲存體僅與三個 AZ 相關聯。由於 Amazon Aurora 資料庫叢集 會利用運算和儲存體的分離，因此無論叢集中有多少執行個體，此規則都是正確的。

例如，執行如下所示的 CLI 命令，以您自己的叢集名稱取代 *my_cluster*。下列範例會依 AZ 名稱的字母順序產生清單。

```
aws rds describe-db-clusters \
  --db-cluster-identifier my_cluster \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' \
  --output text
```

以下範例示範上述 `describe-db-clusters` 命令的範例輸出。它示範 Aurora 叢集的儲存體一律使用三個 AZ。

```
us-east-1c
us-east-1d
us-east-1e
```

若要在網路環境中建立複製，而該網路環境沒有所有連線到這些 AZ 的資源，您必須建立至少兩個與這些 AZ 相關聯的子網路，然後建立包含這兩個或三個子網路的資料庫子網路群組。下列範例示範了做法。

步驟 2：檢查原始叢集的資料庫子網路群組

如果您想要使用與原始叢集中相同的複製子網路數目，您可以從原始叢集的資料庫子網路群組取得子網路數目。Aurora 資料庫子網路群組至少包含兩個子網路，每個子網路都與不同的可用區域相關聯。請記下子網路相關聯的 AZ。

下列範例顯示如何尋找原始叢集的資料庫子網路群組，然後反過來處理至對應的可用 AZ。在第一個命令中取代 *my_cluster* 的 叢集名稱。在第二個命令中取代 *my_subnet* 的資料庫子網路群組名稱。

```
aws rds describe-db-clusters --db-cluster-identifier my_cluster \
  --query '*[].DBSubnetGroup' --output text

aws rds describe-db-subnet-groups --db-subnet-group-name my_subnet_group \
  --query '*[].Subnets[].[SubnetAvailabilityZone.Name]' --output text
```

對於資料庫子網路群組包含兩個子網路的叢集，範例輸出看起來可能會類似以下內容。在此情況下，two-subnets 是在建立資料庫子網路群組時指定的名稱。

```
two-subnets

us-east-1d
us-east-1c
```

對於資料庫子網路群組包含三個子網路的叢集，輸出看起來可能會類似以下內容。

```
three-subnets

us-east-1f
us-east-1d
us-east-1c
```

步驟 3：檢查原始叢集的子網路

如果您需要有關原始叢集中子網路的詳細資訊，請執行類似如下的 AWS CLI 命令。您可以檢查子網路屬性，例如 IP 位址範圍、擁有者等。如此一來，您就可以判斷是否在相同的 VPC 中使用不同的子網路，或是在不同的 VPC 中建立具有類似特性的子網路。

尋找 VPC 中所有可用子網路的子網路 ID。

```
aws ec2 describe-subnets --filters Name=vpc-id,Values=my_vpc \
```

```
  --query '*[].[SubnetId]' --output text
```

尋找資料庫子網路群組中使用的確切子網路。

```
aws rds describe-db-subnet-groups --db-subnet-group-name my_subnet_group \
  --query '*[].Subnets[].[SubnetIdentifier]' --output text
```

然後在清單中指定您要調查的子網路，如下列命令所示。將子網路的名稱取代為 *my_subnet_1* 等。

```
aws ec2 describe-subnets \
  --subnet-ids '["my_subnet_1","my_subnet2","my_subnet3"]'
```

下列範例顯示來自此類 describe-subnets 命令的部分輸出。輸出會顯示您可以看到對於每個子網路的一些重要屬性，例如其相關聯的 AZ 及其所屬的 VPC。

```
{
    'Subnets': [
        {
            'AvailabilityZone': 'us-east-1d',
            'AvailableIpAddressCount': 54,
            'CidrBlock': '10.0.0.64/26',
            'State': 'available',
            'SubnetId': 'subnet-000a0bca00e0b0000',
            'VpcId': 'vpc-3f3c3fc3333b3ffb3',
            ...
        },
        {
            'AvailabilityZone': 'us-east-1c',
            'AvailableIpAddressCount': 55,
            'CidrBlock': '10.0.0.0/26',
            'State': 'available',
            'SubnetId': 'subnet-4b4dbfe4d4a4fd4c4',
            'VpcId': 'vpc-3f3c3fc3333b3ffb3',
            ...
```

步驟 4：檢查原始叢集中資料庫執行個體的可用區域

您可以使用此程序來了解原始叢集中資料庫執行個體所使用的 AZ。如此一來，您可以在複製中為資料庫執行個體設定完全相同的 AZ。您也可以在複製中使用更多或更少的資料庫執行個體，取決於複製是否用於生產、開發和測試等。

對於原始叢集中的每個執行個體，執行如下所示的命令。請確定執行個體已完成建立，且先處於 Available 狀態。取代 *my_instance* 的執行個體識別符。

```
aws rds describe-db-instances --db-instance-identifier my_instance \
  --query '*[].AvailabilityZone' --output text
```

以下範例顯示執行上述 describe-db-instances 命令的輸出：Aurora 叢集有四個資料庫執行個體。因此，我們會執行命令四次，每次都會取代不同的資料庫執行個體識別符。輸出顯示這些資料庫執行個體如何分散到最多三個 AZ。

```
us-east-1a
us-east-1c
us-east-1d
us-east-1a
```

建立複製並新增資料庫執行個體之後，您可以在 create-db-instance 命令中指定這些相同的 AZ 名稱。您可以在為與原始叢集完全相同 AZ 設定的新叢集中設定資料庫執行個體。

步驟 5：檢查您可以用於複製的 VPC

如果您想要在與原始版本不同的 VPC 中建立複製，則可以取得您帳戶可用的 VPC ID 清單。如果您需要在與原始叢集相同的 VPC 中建立任何其他子網路，也可以執行此步驟。當您執行命令來建立子網路時，您可以將 VPC ID 指定為參數。

若要列出所有帳戶中的 VPC，請執行下列 CLI 命令:

```
aws ec2 describe-vpcs --query '*[].[VpcId]' --output text
```

以下範例示範上述 describe-vpcs 命令的範例輸出。輸出顯示目前 AWS 帳戶中有四個 VPC，可用作跨 VPC 複製的來源或目的地。

```
vpc-fd111111
vpc-2222e2cd2a222f22e
vpc-33333333a33333d33
vpc-4ae4d4de4a4444dad
```

您可以使用與複製目的地相同的 VPC，或不同的 VPC。如果原始叢集和複製位於相同的 VPC 中，您可以重複使用相同的資料庫子網路群組進行複製。您也可以建立不同的資料庫子網路群組。例如，新的資料庫子網路群組可能會使用私有子網路，而原始叢集的資料庫子網路群組可能會使用公有子網路。如

果您在不同的 VPC 中建立複製，請確定新的 VPC 中有足夠的子網路，而且子網路與原始叢集中的正確 AZ 相關聯。

建立複製的網路資源

如果在收集網路資訊時發現複製需要額外的網路資源，您可以在嘗試設定複製之前建立這些資源。例如，您可能需要建立更多子網路、與特定 AZ 相關聯的子網路，或新的資料庫子網路群組。

- 步驟 1：建立複製的子網路
- 步驟 2：建立複製的資料庫子網路群組

步驟 1：建立複製的子網路

如果您需要為複製建立新的子網路，請執行類似以下的命令。在不同的 VPC 中建立複製時，或進行一些其他網路變更時，例如使用私有子網路而非公有子網路時，您可能需要執行此操作。

AWS 會自動產生子網路的 ID。將複製的 VPC 名稱取代為 *my_vpc*。選擇 `--cidr-block` 選項的位址範圍，以允許該範圍內至少 16 個 IP 位址。您可以包含要指定的任何其他屬性。執行 `aws ec2 create-subnet help` 命令以查看所有選項。

```
aws ec2 create-subnet --vpc-id my_vpc \
  --availability-zone AZ_name --cidr-block IP_range
```

下列範例顯示新建立子網路的一些重要屬性。

```
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1b',
        'AvailableIpAddressCount': 59,
        'CidrBlock': '10.0.0.64/26',
        'State': 'available',
        'SubnetId': 'subnet-44b4a44f4e44db444',
        'VpcId': 'vpc-555fc5df555e555dc',
        ...
        }
}
```

步驟 2：建立複製的資料庫子網路群組

如果您要在不同的 VPC 或相同 VPC 內的不同子網路集中建立複製，則需建立新的資料庫子網路群組，並在建立複製時指定它。

請確定您知道下列所有詳細資訊。您可以從上述範例的輸出中找到所有這些項目。

1. 原始叢集的 VPC。如需說明，請參閱步驟 3：檢查原始叢集的子網路。
2. 您可能在不同的 VPC 中建立複製的 VPC。如需說明，請參閱步驟 5：檢查您可以用於複製的 VPC。
3. 與原始叢集的 Aurora 儲存體相關聯的三個 AZ。如需說明，請參閱步驟 1：檢查原始叢集的可用區域。
4. 與原始叢集的資料庫子網路群組相關聯的兩個或三個 AZ。如需說明，請參閱步驟 2：檢查原始叢集的資料庫子網路群組。
5. 您要用於複製之 VPC 中所有子網路的子網路 ID 和相關聯的 AZ。使用與 步驟 3：檢查原始叢集的子網路 相同的 `describe-subnets` 命令，取代目的地 VPC 的 VPC ID。

檢查有多少 AZ 與原始叢集的儲存體相關聯，以及與目的地 VPC 中的子網路相關聯。若要成功建立複製，必須有兩個或三個共同的 AZ。如果您的常見可用 AZ 少於兩個，請返回 步驟 1：建立複製的子網路。建立一個、兩個或三個與原始叢集儲存體相關聯之 AZ 相關聯的新子網路。

選擇目的地 VPC 中與原始叢集中 Aurora 儲存體相同 AZ 相關聯的子網路。理想情況下，請選擇三個 AZ。這樣做可讓您靈活地將複製的資料庫執行個體分散到多個 AZ，以獲得高可用性的運算資源。

執行類似下列的命令來建立新的資料庫子網路群組。取代清單中子網路的 ID。如果您使用環境變數指定子網路 ID，請小心以保留 ID 周圍雙引號的方式來引用 `--subnet-ids` 參數清單。

```
aws rds create-db-subnet-group --db-subnet-group-name my_subnet_group \
  --subnet-ids '["my_subnet_1","my_subnet_2","my_subnet3"]' \
  --db-subnet-group-description 'DB subnet group with 3 subnets for clone'
```

以下範例顯示 `create-db-subnet-group` 命令的部分輸出：

```
{
    'DBSubnetGroup': {
        'DBSubnetGroupName': 'my_subnet_group',
        'DBSubnetGroupDescription': 'DB subnet group with 3 subnets for clone',
        'VpcId': 'vpc-555fc5df555e555dc',
        'SubnetGroupStatus': 'Complete',
        'Subnets': [
            {
                'SubnetIdentifier': 'my_subnet_1',
                'SubnetAvailabilityZone': {
                    'Name': 'us-east-1c'
                },
```

```
                'SubnetStatus': 'Active'
            },
            {
                'SubnetIdentifier': 'my_subnet_2',
                'SubnetAvailabilityZone': {
                    'Name': 'us-east-1d'
                },
                'SubnetStatus': 'Active'
            }
            ...
        ],
        'SupportedNetworkTypes': [
            'IPV4'
        ]
    }
}
```

此時，您尚未實際建立複製。您已建立所有相關 VPC 和子網路資源，因此您可以在建立複製時為 `restore-db-cluster-to-point-in-time` 和 `create-db-instance`命令指定適當的參數。

使用新的網路設定建立 Aurora 複製

一旦確定 VPC、子網路、AZ 和子網路群組的正確組態已就緒可供新叢集使用，您就可以執行實際的複製操作。下列 CLI 範例反白顯示您在命令上指定的選項，例如 `--db-subnet-group-name`、`--availability-zone` 和 `--vpc-security-group-ids`，以設定複製及其資料庫執行個體。

- 步驟 1：指定複製的資料庫子網路群組
- 步驟 2：指定複製中執行個體的網路設定
- 步驟 3：建立從用戶端系統到複製的連線

步驟 1：指定複製的資料庫子網路群組

建立複製時，您可以透過指定資料庫子網路群組來設定所有正確的 VPC、子網路和 AZ 設定。使用上述範例中的命令來確認進入資料庫子網路群組的所有關係和映射。

例如，下列命令示範將原始叢集複製到複製。在第一個範例中，來源叢集與兩個子網路相關聯，而複製與三個子網路相關聯。第二個範例顯示相反的情況，從具有三個子網路的叢集複製到具有兩個子網路的叢集。

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier cluster-with-3-subnets \
```

```
  --db-cluster-identifier cluster-cloned-to-2-subnets \
  --restore-type copy-on-write --use-latest-restorable-time \
  --db-subnet-group-name two-subnets
```

如果您想要在複製中使用 Aurora Serverless v2 執行個體，請在建立複製時包含 `--serverless-v2-scaling-configuration` 選項，如下所示。這樣做可讓您在複製中建立資料庫執行個體時使用 `db.serverless` 類別。您也可以稍後修改複製，以新增此擴展組態屬性。此範例中的容量號碼允許叢集中的每個 Serverless v2 執行個體在 2 到 32 個 Aurora 容量單位 (ACU) 之間擴展。如需 Aurora Serverless v2 功能以及如何選擇容量範圍的資訊，請參閱 使用 Aurora Serverless v2。

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier cluster-with-2-subnets \
  --db-cluster-identifier cluster-cloned-to-3-subnets \
  --restore-type copy-on-write --use-latest-restorable-time \
  --db-subnet-group-name three-subnets \
  --serverless-v2-scaling-configuration 'MinCapacity=2,MaxCapacity=32'
```

無論資料庫執行個體使用的子網路數目為何，來源叢集和複製的 Aurora 儲存體都會與三個 AZ 相關聯。下列範例針對上述範例中的兩個 `restore-db-cluster-to-point-in-time` 命令，列出與原始叢集和複製相關聯的 AZ。

```
aws rds describe-db-clusters --db-cluster-identifier cluster-with-3-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f

aws rds describe-db-clusters --db-cluster-identifier cluster-cloned-to-2-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f

aws rds describe-db-clusters --db-cluster-identifier cluster-with-2-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1a
us-east-1c
us-east-1d
```

```
aws rds describe-db-clusters --db-cluster-identifier cluster-cloned-to-3-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1a
us-east-1c
us-east-1d
```

由於每對原始和複製叢集之間至少有兩個 AZ 重疊，因此兩個叢集都可以存取相同的基礎 Aurora 儲存體。

步驟 2：指定複製中執行個體的網路設定

當您在複製中建立資料庫執行個體時，預設會從叢集本身繼承資料庫子網路群組。如此一來，Aurora 會自動將每個執行個體指派給特定子網路，並在與子網路相關聯的 AZ 中建立執行個體。這個選擇很方便，特別是在開發和測試系統中，因為在將新執行個體新增至複製時，您不需要追蹤子網路 ID 或 AZ。

或者，您可以在為複製建立 Aurora 資料庫執行個體時指定 AZ。您指定的 AZ 必須是來自與複製相關聯的一組 AZ。如果您用於複製的資料庫子網路群組只包含兩個子網路，則您只能從與這兩個子網路相關聯的 AZ 中進行挑選。此選項可為高可用性系統提供靈活性和彈性，因為您可以確保寫入器執行個體和待命讀取器執行個體位於不同的 AZ。或者，如果您將其他讀取器新增至叢集，則可以確保它們分散在三個 AZ。如此一來，即使在極少數的 AZ 失敗情況下，您在其他兩個 AZ 中仍有一個寫入器執行個體和另一個讀取器執行個體。

下列範例會將佈建的資料庫執行個體新增至使用自訂資料庫子網路群組的複製 Aurora PostgreSQL 叢集。

```
aws rds create-db-instance --db-cluster-identifier my_aurora_postgresql_clone \
  --db-instance-identifier my_postgres_instance \
  --db-subnet-group-name my_new_subnet \
  --engine aurora-postgresql \
  --db-instance-class db.t4g.medium
```

下列範例顯示來自此類命令的部分輸出。

```
{
  'DBInstanceIdentifier': 'my_postgres_instance',
  'DBClusterIdentifier': 'my_aurora_postgresql_clone',
  'DBInstanceClass': 'db.t4g.medium',
  'DBInstanceStatus': 'creating'
```

```
  ...
}
```

下列範例會將 Aurora Serverless v2 資料庫執行個體新增至使用自訂資料庫子網路群組的 Aurora MySQL 複製。若要能夠使用 Serverless v2 執行個體，請務必指定 `restore-db-cluster-to-point-in-time` 命令的 `--serverless-v2-scaling-configuration` 選項，如上述範例所示。

```
aws rds create-db-instance --db-cluster-identifier my_aurora_mysql_clone \
  --db-instance-identifier my_mysql_instance \
  --db-subnet-group-name my_other_new_subnet \
  --engine aurora-mysql \
  --db-instance-class db.serverless
```

下列範例顯示來自此類命令的部分輸出。

```
{
  'DBInstanceIdentifier': 'my_mysql_instance',
  'DBClusterIdentifier': 'my_aurora_mysql_clone',
  'DBInstanceClass': 'db.serverless',
  'DBInstanceStatus': 'creating'
  ...
}
```

步驟 3：建立從用戶端系統到複製的連線

如果您已從用戶端系統連線至 Aurora 叢集，建議您允許與新複製相同的連線類型。例如，您可以從 Amazon Cloud9 執行個體或 EC2 執行個體連線到原始叢集。若要允許來自相同用戶端系統的連線，或您在目的地 VPC 中建立的新連線，請設定與 VPC 中同等的資料庫子網路群組和 VPC 安全群組。然後在建立複製時指定子網路群組和安全群組。

下列範例會設定 Aurora Serverless v2 複製。該組態是根據建立資料庫叢集時 `--engine-mode provisioned` 和 `--serverless-v2-scaling-configuration` 的組合，然後是在叢集中建立每個資料庫執行個體時的 `--db-instance-class db.serverless`。`provisioned` 引擎模式是預設值，因此您可以視需要省略該選項。

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier serverless-sql-postgres\
  --db-cluster-identifier serverless-sql-postgres-clone \
  --db-subnet-group-name 'default-vpc-1234' \
  --vpc-security-group-ids 'sg-4567' \
```

```
  --serverless-v2-scaling-configuration 'MinCapacity=0.5,MaxCapacity=16' \
  --restore-type copy-on-write \
  --use-latest-restorable-time
```

然後，在複製中建立資料庫執行個體時，請指定相同的 `--db-subnet-group-name` 選項。或者，您可以包含 `--availability-zone` 選項，並指定與該子網路群組中子網路相關聯的其中一個 AZ。該 AZ 也必須是與原始叢集相關聯的其中一個 AZ。

```
aws rds create-db-instance \
  --db-cluster-identifier serverless-sql-postgres-clone \
  --db-instance-identifier serverless-sql-postgres-clone-instance \
  --db-instance-class db.serverless \
  --db-subnet-group-name 'default-vpc-987zyx654' \
  --availability-zone 'us-east-1c' \
  --engine aurora-postgresql
```

將叢集從公有子網路移至私有子網路

您可以使用複製在公有和私有子網路之間遷移叢集。在將應用程式部署到生產環境之前，您可能會在為應用程式新增額外的安全層時執行此操作。在此範例中，您應該已在使用 Aurora 開始複製程序之前設定私有子網路和 NAT 閘道。

對於涉及 Aurora 的步驟，您可以遵循與上述 收集網路環境的相關資訊 和 使用新的網路設定建立 Aurora 複製 範例相同的一般步驟。主要差別在於，即使您有從原始叢集映射到所有 AZ 公有子網路，現在您必須確認有足夠的私有子網路用於 Aurora 叢集，而且這些子網路與原始叢集中用於 Aurora 儲存體的所有相同 AZ 相關聯。與其他複製使用案例類似，您可以使用與所需 AZ 相關聯的三個或兩個私有子網路來建立複製的資料庫子網路群組。不過，如果您在資料庫子網路群組中使用兩個私有子網路，則必須有一個與原始叢集中用於 Aurora 儲存體之與地三個 AZ 相關聯的第三個私有子網路。

您可以參考此檢查清單，確認所有執行此類型複製操作的需求都已就緒。

- 記錄與原始叢集相關聯的三個 AZ。如需說明，請參閱步驟 1：檢查原始叢集的可用區域。
- 在原始叢集的資料庫子網路群組中，記錄與公有子網路相關聯的三個或兩個 AZ。如需說明，請參閱步驟 3：檢查原始叢集的子網路。
- 建立私有子網路，映射至與原始叢集相關聯的所有三個 AZ。此外，請執行任何其他網路設定，例如建立 NAT 閘道，以便能夠與私有子網路通訊。如需相關指示，請參閱《Amazon Virtual Private Cloud 使用者指南》中的刪除子網路。
- 建立新的資料庫子網路群組，其中包含從一開始與 AZ 相關聯的三個或兩個私有子網路。如需說明，請參閱步驟 2：建立複製的資料庫子網路群組。

當所有先決條件都就緒時，您可以在建立複製時暫停原始叢集上的資料庫活動，並切換應用程式以使用它。建立複製並確認您可以連線到複製、執行應用程式碼，以此類推，之後您可以停止使用原始叢集。

建立跨 VPC 複製的端對端範例

在與原始 VPC 不同的 VPC 中建立複製會使用與上述範例中相同的一般步驟。由於 VPC ID 是子網路的屬性，因此您在執行任何 RDS CLI 命令時，不會實際指定 VPC ID 作為參數。主要差別在於，您更可能需要建立新的子網路、映射至特定 AZ 的新子網路、VPC 安全群組和新的資料庫子網路群組。如果這是您在該 VPC 中建立的第一個 Aurora 叢集，則尤其如此。

您可以參考此檢查清單，確認所有執行此類型複製操作的需求都已就緒。

- 記錄與原始叢集相關聯的三個 AZ。如需說明，請參閱步驟 1：檢查原始叢集的可用區域。
- 記錄與原始叢集資料庫子網路群組中子網路相關聯的三個或兩個 AZ。如需說明，請參閱步驟 2：檢查原始叢集的資料庫子網路群組。
- 建立子網路，映射至與原始叢集相關聯的所有三個 AZ。如需說明，請參閱步驟 1：建立複製的子網路。
- 為用戶端系統、應用程式伺服器等進行任何其他網路設定，例如設定 VPC 安全群組，以便能夠與複製中的資料庫執行個體通訊。如需說明，請參閱使用安全群組控制存取。
- 建立新的資料庫子網路群組，其中包含從一開始與 AZ 相關聯的三個或兩個子網路。如需說明，請參閱步驟 2：建立複製的資料庫子網路群組。

當所有先決條件都就緒時，您可以在建立複製時暫停原始叢集上的資料庫活動，並切換應用程式以使用它。建立複製並驗證您可以連線到複製、執行應用程式碼等之後，您可以考慮是否要同時讓原始和複製保持執行，或停止使用原始叢集。

下列 Linux 範例顯示將 Aurora 資料庫叢集從一個 VPC 複製到另一個 VPC 的 AWS CLI 操作順序。有些與範例無關的欄位不會顯示在命令輸出中。

首先，我們會檢查來源和目的地 VPC 的 ID。您在建立 VPC 時指派給 VPC 的描述性名稱，會在 VPC 中繼資料中以標籤表示。

```
$ aws ec2 describe-vpcs --query '*[].[VpcId,Tags]'
[
    [
        'vpc-0f0c0fc0000b0ffb0',
        [
            {
                'Key': 'Name',
                'Value': 'clone-vpc-source'
```

```
            }
        ]
    ],
    [
        'vpc-9e99d9f99a999bd99',
        [
            {
                'Key': 'Name',
                'Value': 'clone-vpc-dest'
            }
        ]
    ]
]
```

原始叢集已存在於來源 VPC 中。若要使用 Aurora 儲存體的相同一組 AZ 來設定複製，我們會檢查原始叢集使用的 AZ。

```
$ aws rds describe-db-clusters --db-cluster-identifier original-cluster \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f
```

我們確保有與原始叢集使用的 AZ 對應的子網路：us-east-1d、us-east-1c 和 us-east-1f。

```
$ aws ec2 create-subnet --vpc-id vpc-9e99d9f99a999bd99 \
  --availability-zone us-east-1c --cidr-block 10.0.0.128/28
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1c',
        'SubnetId': 'subnet-3333a33be3ef3e333',
        'VpcId': 'vpc-9e99d9f99a999bd99',
    }
}

$ aws ec2 create-subnet --vpc-id vpc-9e99d9f99a999bd99 \
--availability-zone us-east-1d --cidr-block 10.0.0.160/28
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1d',
        'SubnetId': 'subnet-4eeb444cd44b4d444',
        'VpcId': 'vpc-9e99d9f99a999bd99',
```

```
    }
}

$ aws ec2 create-subnet --vpc-id vpc-9e99d9f99a999bd99 \
--availability-zone us-east-1f --cidr-block 10.0.0.224/28
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1f',
        'SubnetId': 'subnet-66eea6666fb66d66c',
        'VpcId': 'vpc-9e99d9f99a999bd99',
    }
}
```

此範例會確認有子網路映射到目的地 VPC 中的必要 AZ。

```
aws ec2 describe-subnets --query 'sort_by(*[] | [?VpcId == `vpc-9e99d9f99a999bd99`] |
[].{SubnetId:SubnetId,VpcId:VpcId,AvailabilityZone:AvailabilityZone},
 &AvailabilityZone)' --output table

-------------------------------------------------------------------------
|                            DescribeSubnets                            |
+------------------+---------------------------+-------------------------+
| AvailabilityZone |         SubnetId          |          VpcId          |
+------------------+---------------------------+-------------------------+
|  us-east-1a      |  subnet-000ff0e00000c0aea  |  vpc-9e99d9f99a999bd99  |
|  us-east-1b      |  subnet-1111d111111ca11b1  |  vpc-9e99d9f99a999bd99  |
|  us-east-1c      | subnet-3333a33be3ef3e333   |  vpc-9e99d9f99a999bd99  |
|  us-east-1d      | subnet-4eeb444cd44b4d444   |  vpc-9e99d9f99a999bd99  |
|  us-east-1f      | subnet-66eea6666fb66d66c   |  vpc-9e99d9f99a999bd99  |
+------------------+---------------------------+-------------------------+
```

在 VPC 中建立 Aurora 資料庫叢集之前，您必須有一個資料庫子網路群組，其子網路會映射至用於 Aurora 儲存體的 AZ。建立一般叢集時，您可以使用任何一組三個 AZ。當您複製現有的叢集時，子網路群組必須至少符合其用於 Aurora 儲存體的三個 AZ中的兩個。

```
$ aws rds create-db-subnet-group \
  --db-subnet-group-name subnet-group-in-other-vpc \
  --subnet-ids
 '["subnet-3333a33be3ef3e333","subnet-4eeb444cd44b4d444","subnet-66eea6666fb66d66c"]' \
  --db-subnet-group-description 'DB subnet group with 3 subnets:
 subnet-3333a33be3ef3e333,subnet-4eeb444cd44b4d444,subnet-66eea6666fb66d66c'

{
```

```
    'DBSubnetGroup': {
        'DBSubnetGroupName': 'subnet-group-in-other-vpc',
        'DBSubnetGroupDescription': 'DB subnet group with 3 subnets:
 subnet-3333a33be3ef3e333,subnet-4eeb444cd44b4d444,subnet-66eea6666fb66d66c',
        'VpcId': 'vpc-9e99d9f99a999bd99',
        'SubnetGroupStatus': 'Complete',
        'Subnets': [
            {
                'SubnetIdentifier': 'subnet-4eeb444cd44b4d444',
                'SubnetAvailabilityZone': { 'Name': 'us-east-1d' }
            },
            {
                'SubnetIdentifier': 'subnet-3333a33be3ef3e333',
                'SubnetAvailabilityZone': { 'Name': 'us-east-1c' }
            },
            {
                'SubnetIdentifier': 'subnet-66eea6666fb66d66c',
                'SubnetAvailabilityZone': { 'Name': 'us-east-1f' }
            }
        ]
    }
}
```

現在子網路和資料庫子網路群組已就緒。下列範例顯示複製叢集的 `restore-db-cluster-to-point-in-time`。`--db-subnet-group-name` 選項會將複製與一組正確的子網路建立關聯，這些子網路映射至原始叢集中一組正確的 AZ。

```
$ aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier original-cluster \
  --db-cluster-identifier clone-in-other-vpc \
  --restore-type copy-on-write --use-latest-restorable-time \
  --db-subnet-group-name subnet-group-in-other-vpc

{
  'DBClusterIdentifier': 'clone-in-other-vpc',
  'DBSubnetGroup': 'subnet-group-in-other-vpc',
  'Engine': 'aurora-postgresql',
  'EngineVersion': '15.4',
  'Status': 'creating',
  'Endpoint': 'clone-in-other-vpc.cluster-c0abcdef.us-east-1.rds.amazonaws.com'
}
```

下列範例確認複製中的 Aurora 儲存體使用與原始叢集相同的一組 AZ。

```
$ aws rds describe-db-clusters --db-cluster-identifier clone-in-other-vpc \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f
```

此時，您可以為複製建立資料庫執行個體。請確定與每個執行個體相關聯的 VPC 安全群組允許來自您用於 EC2 執行個體、應用程式伺服器等目的地 VPC 中 IP 位址範圍的連線。

使用 AWS RAM 和 Amazon Aurora 進行跨帳戶複製

透過將 AWS Resource Access Manager (AWS RAM) 與 Amazon Aurora 搭配使用，您可以與其他 AWS 帳戶或組織共用屬於您 AWS 帳戶的 Aurora 資料庫叢集和複製。例如跨帳戶複製會比建立並還原資料庫快照更快。您可以建立其中一個 Aurora 資料庫叢集的複製，並共享該複製。或者，您可以與其他 AWS 帳戶共用 Aurora 資料庫叢集，並讓帳戶持有人建立複製。您選擇的方式會依使用案例而定。

例如，您可能需要定期與組織內部稽核團隊共享財務資料庫的複製。在此情況下，您的稽核團隊會有其使用之應用程式的自有 AWS 帳戶。您可以授予稽核團隊的 AWS 帳戶存取 Aurora 資料庫叢集的許可，並視需要複製它。

另一方面，如果外部供應商稽核您的財務資料，您可能會偏好由您自行建立複製。然後，您僅需授予外部供應商複製的存取權。

您也可以使用跨帳戶複製來支援許多相同的使用案例，以便在同一個 AWS 帳戶中複製，例如開發和測試。例如，您的組織可能會使用不同的 AWS 帳戶進行生產、開發、測試等。如需詳細資訊，請參閱Aurora 複製的概觀。

因此，您可能想要與另一個 AWS 帳戶共用複製，或允許另一個 AWS 帳戶建立 Aurora 資料庫叢集的複製。在任一情況下，請先使用 來 AWS RAM 建立共用物件。如需在帳戶之間 AWS 共用 AWS 資源的完整資訊，請參閱AWS RAM 《 使用者指南》。

建立跨帳戶複製需要擁有原始叢集 AWS 的帳戶以及建立複製 AWS 的帳戶執行動作。首先，原始叢集擁有者需要修改叢集，允許一或多個其他帳戶複製該叢集。如果有任何帳戶位於不同的 AWS 組織中，AWS 會產生共用邀請。另一個帳戶必須接受邀請才能繼續進行。然後，每個獲得授權的帳戶便可以複製叢集。在整個過程中，可使用叢集的唯一 Amazon Resource Name (ARN) 來識別叢集。

如同在相同 AWS 帳戶中複製一樣，只有在來源或複製對資料進行變更時，才會使用額外的儲存空間。然後，此時會收取儲存裝置費用。若來源叢集遭到刪除，儲存成本便會平均分配到剩餘的複製叢集。

主題

- 跨帳戶複製的限制
- 允許其他 AWS 帳戶複製您的叢集
- 複製另一個 AWS 帳戶擁有的叢集

跨帳戶複製的限制

Aurora 跨帳戶複製的限制如下：

- 您無法跨 AWS 帳戶複製Aurora Serverless v1叢集。
- 您無法檢視或接受與 共用資源的邀請 AWS 管理主控台。使用 AWS CLI、Amazon RDS API 或 AWS RAM 主控台來檢視和接受共用資源的邀請。
- 您只能從與 共用的資源建立新的複製 AWS 帳戶。這適用於共用資源是原始 Aurora 資料庫叢集或先前建立的複製。
- 您只能從與您 AWS 帳戶共用的複製中建立新的複製。
- 您無法共用已與 AWS 您的帳戶共用的資源 （區塊或 Aurora 資料庫叢集）。
- 您可從任何單一 Aurora 資料庫叢集建立最多 15 個跨帳戶複製。
- 15 個跨帳戶複製的每個都必須由不同的 AWS 帳戶擁有。也就是說，您只能在任何帳戶中建立一個叢集的跨帳戶複製 AWS 。
- 複製叢集後，為了對跨帳戶複製強制實施限制，原始叢集及其複製將視為相同。您無法在同一帳戶中同時建立原始叢集和複製叢集的跨 AWS 帳戶複製。原始叢集及其任何複製的跨帳戶複製總數不得超過 15 個。
- 除非叢集處於 `ACTIVE` 狀態，否則您無法與其他 AWS 帳戶共用 Aurora 資料庫叢集。
- 您無法重新命名已與其他 AWS 帳戶共用的 Aurora 資料庫叢集。
- 您無法針對使用預設 RDS 金鑰加密的叢集建立跨帳戶複製。
- 您無法從另一個 AWS 帳戶共用的加密 Aurora 資料庫叢集，在一個 AWS 帳戶中建立未加密的複製。叢集擁有者必須授予許可，才能存取來源叢集的 AWS KMS key。不過，您可以在建立複製時使用不同金鑰。

允許其他 AWS 帳戶複製您的叢集

若要允許其他 AWS 帳戶複製您擁有的叢集，請使用 AWS RAM 設定共用許可。這樣做也會向不同 AWS 組織中的每個其他帳戶傳送邀請。

如需在 AWS RAM 主控台中共享您擁有的資源的程序，請參閱AWS RAM 《 使用者指南》中的共享您擁有的資源。

主題

- 將複製叢集的許可授予其他 AWS 帳戶
- 檢查您擁有的叢集是否與其他 AWS 帳戶共用

將複製叢集的許可授予其他 AWS 帳戶

若您要共用的叢集經過加密，您也要共用叢集的 AWS KMS key 。您可以允許一個 AWS 帳戶中的 AWS Identity and Access Management (IAM) 使用者或角色在不同帳戶中使用 KMS 金鑰。

若要這樣做，請先將外部帳戶 （根使用者） 新增至 KMS 金鑰的金鑰政策 AWS KMS。您不需要將個別使用者或角色新增到金鑰政策，只需新增擁有這些使用者或角色的外部帳戶。您只能共用您建立的 KMS 金鑰，而非預設的 RDS 服務金鑰。如需 KMS 金鑰存取控制的相關資訊，請參閱AWS KMS的身分驗證及存取控制。

主控台

授予複製您叢集的許可

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您希望共享的資料庫叢集，來查看其 Details (詳細資訊) 頁面，然後選擇 Connectivity & security (連線能力與安全) 標籤。
4. 在與其他 AWS 帳戶共用資料庫叢集區段中，輸入您要允許 複製此叢集之 AWS 帳戶的數值帳戶 ID。針對相同組織中的帳戶 ID，您可以在方塊中開始輸入，然後從選單內選擇。

   > **Important**
   >
   > 在某些情況下，您可能會希望讓並非位於與您帳戶相同 AWS 組織中的帳戶複製叢集。在這些情況下，基於安全理由，主控台 不會報告擁有該帳戶 ID 的人員，或是該帳戶是否存在。
   >
   > 請小心輸入與您帳戶不在相同 AWS 組織中 AWS 的帳戶號碼。請立即驗證您已和您計劃的目標帳戶進行共享。

5. 在確認頁面上，驗證您指定的帳戶 ID 是否正確。在確認方塊中輸入 `share` 以進行確認。

在詳細資訊頁面上，會出現一個項目，顯示與此資料庫叢集共用之 AWS 帳戶底下指定的帳戶 ID。 Status (狀態) 欄一開始會顯示 Pending (待定) 狀態。

6. 請聯絡其他 AWS 帳戶的擁有者，或如果您同時擁有該帳戶，則登入該帳戶。指示另一個帳戶的擁有者接受共享邀請並複製資料庫叢集，如以下說明所示。

AWS CLI

授予複製您叢集的許可

1. 收集必要參數的資訊。您需要叢集的 ARN 和其他 AWS 帳戶的數值 ID。
2. 執行 AWS RAM CLI 命令 `create-resource-share`。

   針對 Linux、macOS 或 Unix：

   ```
   aws ram create-resource-share --name descriptive_name \
     --region region \
     --resource-arns cluster_arn \
     --principals other_account_ids
   ```

   在 Windows 中：

   ```
   aws ram create-resource-share --name descriptive_name ^
     --region region ^
     --resource-arns cluster_arn ^
     --principals other_account_ids
   ```

   若要針對 `--principals` 包含多個帳戶 ID，請使用空格分隔不同的 ID。若要指定獲得允許的帳戶 ID 是否可在您 AWS 組織的外部，請針對 `--allow-external-principals` 包含 `--no-allow-external-principals` 或 `create-resource-share` 參數。

AWS RAM API

授予複製您叢集的許可

1. 收集必要參數的資訊。您需要叢集的 ARN 和其他 AWS 帳戶的數值 ID。
2. 呼叫 AWS RAM API 操作 CreateResourceShare，並指定下列值：

   - 指定一或多個帳戶的帳戶 ID AWS 做為 `principals` 參數。

- 將一或多個 Aurora 資料庫叢集的 ARN 指定為 `resourceArns` 參數。
- 針對 `allowExternalPrincipals` 參數包含布林值，指定獲得允許的帳戶 ID 是否可位於您 AWS 組織的外部。

重新建立使用預設 RDS 金鑰的叢集

若您預計使用預設 RDS 金鑰來共享的加密叢集，請務必重新建立叢集。若要執行此作業，請建立資料庫叢集的手動快照、使用 AWS KMS key，然後將叢集還原至新的叢集。然後共用新叢集。若要執行此程序，請進行下列步驟。

重新建立使用預設 RDS 金鑰的加密叢集

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 從導覽窗格選擇 Snapshots (快照)。
3. 選擇您的快照。
4. 針對 Actions (動作)，選擇 Copy Snapshot (複製快照)，然後選擇 Enable encryption (啟用加密)。
5. 針對 AWS KMS key，選擇您希望使用的新加密金鑰。
6. 還原複製的快照。若要執行此作業，請依照 從資料庫叢集快照還原 中的程序進行。新的資料庫執行個體會使用您的新加密金鑰。
7. (選用) 若您不再需要舊的資料庫叢集，您可以刪除它。若要執行此作業，請依照 刪除資料庫叢集快照 中的程序進行。在您執行該作業前，請確認新的叢集已具備所有必要的資料，並且您的應用程式可以成功存取新的叢集。

檢查您擁有的叢集是否與其他 AWS 帳戶共用

您可以檢查其他使用者是否具備共享叢集的許可。執行此作業有助您了解叢集是否已接近跨帳戶複製的數量上限。

如需使用 AWS RAM 主控台共用資源的程序，請參閱AWS RAM 《 使用者指南》中的共用您擁有的資源。

AWS CLI

了解您擁有的叢集是否與其他 AWS 帳戶共用

- 呼叫 AWS RAM CLI 命令 `list-principals`，使用您的帳戶 ID 做為資源擁有者，並使用叢集的 ARN 做為資源 ARN。您可以使用以下命令查看所有共享。結果指出允許哪些 AWS 帳戶複製叢集。

```
aws ram list-principals \
    --resource-arns your_cluster_arn \
    --principals your_aws_id
```

AWS RAM API

了解您擁有的叢集是否與其他 AWS 帳戶共用

- 呼叫 AWS RAM API 操作 ListPrincipals。使用您的帳戶 ID 做為資源擁有者，以及您叢集的 ARN 做為資源 ARN。

複製另一個 AWS 帳戶擁有的叢集

若要複製另一個 AWS 帳戶擁有的叢集，請使用 AWS RAM 取得進行複製的許可。在您擁有必要的許可後，便可以使用複製 Aurora 叢集的標準程序。

您也可以檢查您擁有的叢集是否為不同 AWS 帳戶所擁有叢集的複製。

如需在 AWS RAM 主控台中使用其他人擁有之資源的程序，請參閱《 使用者指南》中的存取與您共用的資源。 AWS RAM

主題

- 檢視複製其他 AWS 帳戶所擁有叢集的邀請
- 接受共用其他 AWS 帳戶所擁有叢集的邀請
- 複製另一個 AWS 帳戶擁有的 Aurora 叢集
- 檢查資料庫叢集是否為跨帳戶複製

檢視複製其他 AWS 帳戶所擁有叢集的邀請

若要使用邀請來複製其他 AWS 組織中 AWS 帳戶擁有的叢集，請使用 AWS CLI、 AWS RAM 主控台或 AWS RAM API。目前，您無法使用 Amazon RDS 主控台執行這項程序。

如需在 AWS RAM 主控台中使用邀請的程序，請參閱AWS RAM 《 使用者指南》中的存取與您共用的資源。

AWS CLI

查看複製其他 AWS 帳戶所擁有叢集的邀請

1. 執行 AWS RAM CLI 命令 get-resource-share-invitations。

```
aws ram get-resource-share-invitations --region region_name
```

   上述命令的結果會顯示所有複製叢集的邀請，包括任何您已接受和拒絕的邀請。

2. (選用) 篩選清單，讓您可以只查看需要您採取動作的邀請。若要執行此作業，請新增 `--query 'resourceShareInvitations[?status==`PENDING`]'` 參數。

AWS RAM API

查看複製其他 AWS 帳戶所擁有叢集的邀請

1. 呼叫 AWS RAM API 操作 GetResourceShareInvitations。此操作會傳回所有這類邀請，包括任何您已接受或拒絕的邀請。
2. (選用) 透過針對 `resourceShareAssociations` 的 `status` 值選取 `PENDING` 傳回欄位，來只尋找需要您採取動作的邀請。

接受共用其他 AWS 帳戶所擁有叢集的邀請

您可以接受邀請，以共用不同 AWS 組織中其他 AWS 帳戶擁有的叢集。若要使用這些邀請，請使用 AWS CLI、 AWS RAM 和 RDS APIs 或 AWS RAM 主控台。目前，您無法使用 RDS 主控台執行這項程序。

如需在 AWS RAM 主控台中使用邀請的程序，請參閱AWS RAM 《 使用者指南》中的存取與您共用的資源。

AWS CLI

接受從另一個 AWS 帳戶共用叢集的邀請

1. 執行 AWS RAM CLI 命令 尋找邀請 ARNget-resource-share-invitations，如下所示。
2. 呼叫 AWS RAM CLI 命令 接受邀請accept-resource-share-invitation，如下所示。

   針對 Linux、macOS 或 Unix：

```
aws ram accept-resource-share-invitation \
  --resource-share-invitation-arn invitation_arn \
  --region region
```

   在 Windows 中：

```
aws ram accept-resource-share-invitation ^
  --resource-share-invitation-arn invitation_arn ^
  --region region
```

AWS RAM 和 RDS API

接受共享其他人叢集的邀請

1. 呼叫 AWS RAM API 操作 尋找邀請 ARNGetResourceShareInvitations，如前所示。
2. 將該 ARN 做為 `resourceShareInvitationArn` 參數傳遞至 RDS API 操作 AcceptResourceShareInvitation。

複製另一個 AWS 帳戶擁有的 Aurora 叢集

接受來自擁有資料庫叢集 AWS 之帳戶的邀請後，您可以複製叢集，如下所示。

主控台

複製另一個 AWS 帳戶擁有的 Aurora 叢集

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。

在資料庫清單頂端，您應該會看到一或多個 Role (角色) 值為 `Shared from account #account_id` 的項目。基於安全理由，您只能看到關於原始叢集的有限資訊。您可以看到的屬性為資料庫引擎和版本等在您複製叢集中也必須相同的屬性。

3. 請選擇您想要複製的叢集。
4. 針對 Actions (動作)，選擇 Create clone (建立複製)。
5. 遵循 主控台 中的程序來完成設定複製叢集。
6. 若需要的話，請啟用複製叢集的加密。若您要複製的叢集經過加密，您必須為複製叢集啟用加密。與您共用叢集的 AWS 帳戶也必須共用用來加密叢集的 KMS 金鑰。您可以使用相同的 KMS 金鑰或是您自己的 KMS 金鑰來加密複製。您無法為使用預設 KMSS 金鑰加密的叢集建立跨帳戶複製。

   擁有加密金鑰的帳戶必須使用金鑰政策，將使用金鑰的許可授予目標帳戶。這項程序與共享加密快照的程序相似，因為他們都是使用將利用金鑰的許可授予目標帳戶的金鑰政策。

AWS CLI

複製另一個 AWS 帳戶擁有的 Aurora 叢集

1. 接受來自擁有資料庫叢集 AWS 之帳戶的邀請，如下所示。
2. 在 RDS CLI 命令 `source-db-cluster-identifier` 的 `restore-db-cluster-to-point-in-time` 參數中指定來源叢集的完整 ARN，如以下所示。

   若做為 `source-db-cluster-identifier` 傳遞的 ARN 尚未共享，則會傳回相同的錯誤，就好像指定的叢集不存在般。

   對於 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier=arn:aws:rds:arn_details \
  --db-cluster-identifier=new_cluster_id \
  --restore-type=copy-on-write \
  --use-latest-restorable-time
```

   在 Windows 中：

```
aws rds restore-db-cluster-to-point-in-time ^
  --source-db-cluster-identifier=arn:aws:rds:arn_details ^
```

```
  --db-cluster-identifier=new_cluster_id ^
  --restore-type=copy-on-write ^
  --use-latest-restorable-time
```

3. 若您要複製的叢集經過加密，請透過在其中包含 kms-key-id 參數來加密您的複製叢集。這個 kms-key-id 值可以和用來加密原始資料庫叢集的值相同，或是使用您自己的 KMS 金鑰。您的帳戶也必須具備使用該加密金鑰的許可。

   對於 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier=arn:aws:rds:arn_details \
  --db-cluster-identifier=new_cluster_id \
  --restore-type=copy-on-write \
  --use-latest-restorable-time \
  --kms-key-id=arn:aws:kms:arn_details
```

   在 Windows 中：

```
aws rds restore-db-cluster-to-point-in-time ^
  --source-db-cluster-identifier=arn:aws:rds:arn_details ^
  --db-cluster-identifier=new_cluster_id ^
  --restore-type=copy-on-write ^
  --use-latest-restorable-time ^
  --kms-key-id=arn:aws:kms:arn_details
```

   擁有加密金鑰的帳戶必須使用金鑰政策，將使用金鑰的許可授予目標帳戶。這項程序與共享加密快照的程序相似，因為他們都是使用將利用金鑰的許可授予目標帳戶的金鑰政策。以下是金鑰政策的範例。

   JSON

```
{
    "Id": "key-policy-1",
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
```

```
                    "arn:aws:iam::111122223333:user/KeyUser",
                    "arn:aws:iam::111122223333:root"
                ]
            },
            "Action": [
                "kms:CreateGrant",
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Allow attachment of persistent resources",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:user/KeyUser",
                    "arn:aws:iam::111122223333:root"
                ]
            },
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "kms:GrantIsForAWSResource": true
                }
            }
        }
    ]
}
```

> **Note**
>
> restore-db-cluster-to-point-in-time AWS CLI 命令只會還原資料庫叢集，而不會還原該資料庫叢集的資料庫執行個體。若要為已還原的資料庫叢集建立資料庫執行個體，請呼叫 create-db-instance 命令。以 `--db-cluster-identifier` 指定已還原資料庫叢集的識別碼。
> 只在 `restore-db-cluster-to-point-in-time` 命令完成且資料庫叢集可用時，您才能建立資料庫執行個體。

RDS API

複製另一個 AWS 帳戶擁有的 Aurora 叢集

1. 接受來自擁有資料庫叢集 AWS 之帳戶的邀請，如下所示。
2. 在 RDS API 操作 `SourceDBClusterIdentifier` 的 `RestoreDBClusterToPointInTime` 參數中指定來源叢集的完整 ARN 來複製叢集。

   若做為 `SourceDBClusterIdentifier` 傳遞的 ARN 尚未共享，則會傳回相同的錯誤，就好像指定的叢集不存在般。
3. 若您要複製的叢集經過加密，請在其中包含 `KmsKeyId` 參數來加密您的複製叢集。這個 `kms-key-id` 值可以和用來加密原始資料庫叢集的值相同，或是使用您自己的 KMS 金鑰。您的帳戶也必須具備使用該加密金鑰的許可。

   在您複製磁碟區時，目標帳戶必須具備用來加密來源叢集的加密金鑰使用許可。Aurora 會使用在 `KmsKeyId` 中指定的加密金鑰來加密新的複製叢集。

   擁有加密金鑰的帳戶必須使用金鑰政策，將使用金鑰的許可授予目標帳戶。這項程序與共享加密快照的程序相似，因為他們都是使用將利用金鑰的許可授予目標帳戶的金鑰政策。以下是金鑰政策的範例。

   JSON

```
{
    "Id": "key-policy-1",
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
```

```
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:user/KeyUser",
                    "arn:aws:iam::111122223333:root"
                ]
            },
            "Action": [
                "kms:CreateGrant",
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Allow attachment of persistent resources",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:user/KeyUser",
                    "arn:aws:iam::111122223333:root"
                ]
            },
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "kms:GrantIsForAWSResource": true
                }
            }
        }
    ]
}
```

> **Note**
>
> RestoreDBClusterToPointInTime RDS API 操作只會還原資料庫叢集，而不會還原該資料庫叢集的資料庫執行個體。若要為已還原的資料庫叢集建立資料庫執行個體，請呼叫 CreateDBInstance RDS API 操作。以 DBClusterIdentifier 指定已還原資料庫叢集的識別碼。只有在 RestoreDBClusterToPointInTime 操作完成後，且資料庫叢集為可用時，您才能建立資料庫執行個體。

檢查資料庫叢集是否為跨帳戶複製

DBClusters 物件可識別每個叢集是否是跨帳戶複製。您可以在執行 RDS CLI 命令 include-shared 時，使用 describe-db-clusters 選項來查看您具備複製許可的叢集。但是，您無法看到這類叢集的大部分組態詳細資訊。

AWS CLI

檢查資料庫叢集是否為跨帳戶複製

- 請呼叫 RDS CLI 命令 describe-db-clusters。

  以下範例會示範實際或潛在跨帳戶複製資料庫叢集在 describe-db-clusters 輸出中的顯示方式。對於您 AWS 帳戶擁有的現有叢集， CrossAccountClone 欄位會指出叢集是否為另一個 AWS 帳戶擁有之資料庫叢集的複製。

  在某些情況下，項目的 AWS 帳號可能與 DBClusterArn 欄位中的帳號不同。在此情況下，該項目代表由不同 AWS 帳戶擁有且您可以複製的叢集。這類項目除了 DBClusterArn 之外還有一些欄位。在建立複製叢集時，請指定與原始叢集相同的 StorageEncrypted、Engine 和 EngineVersion 值。

```
$aws rds describe-db-clusters --include-shared --region us-east-1
{
  "DBClusters": [
      {
          "EarliestRestorableTime": "2023-02-01T21:17:54.106Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": false,
...
      },
      {
```

```
            "EarliestRestorableTime": "2023-02-09T16:01:07.398Z",
            "Engine": "aurora-mysql",
            "EngineVersion": "8.0.mysql_aurora.3.02.0",
            "CrossAccountClone": true,
...
        },
        {
            "StorageEncrypted": false,
            "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-
abcdefgh",
            "Engine": "aurora-mysql",
            "EngineVersion": "8.0.mysql_aurora.3.02.0
    ]
}
```

RDS API

檢查資料庫叢集是否為跨帳戶複製

- 請呼叫 RDS API 操作 DescribeDBClusters。

  對於您 AWS 帳戶擁有的現有叢集，`CrossAccountClone` 欄位會指出叢集是否為另一個 AWS 帳戶擁有之資料庫叢集的複製。`DBClusterArn` 欄位中具有不同 AWS 帳號的項目代表您可以複製且由其他 AWS 帳戶擁有的叢集。這些項目除了 `DBClusterArn` 之外還有一些欄位。在建立複製叢集時，請指定與原始叢集相同的 `StorageEncrypted`、`Engine` 和 `EngineVersion` 值。

  以下範例會顯示傳回值，示範實際和潛在的複製叢集。

```
{
  "DBClusters": [
      {
          "EarliestRestorableTime": "2023-02-01T21:17:54.106Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": false,
...
      },
      {
          "EarliestRestorableTime": "2023-02-09T16:01:07.398Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": true,
```

```
...
      },
      {
          "StorageEncrypted": false,
          "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-
abcdefgh",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0"
      }
  ]
}
```

將 Aurora 與其他 AWS 服務整合

將 Amazon Aurora 與其他 AWS 服務整合，讓您可以擴展 Aurora 資料庫叢集，以使用 AWS 雲端中的其他功能。

主題

- 整合 AWS 服務與 Amazon Aurora MySQL
- 將 AWS 服務與 Amazon Aurora PostgreSQL 整合

整合 AWS 服務與 Amazon Aurora MySQL

Amazon Aurora MySQL 與其他 AWS 服務整合，因此您可以擴展 Aurora MySQL 資料庫叢集，以在 AWS 雲端中使用其他功能。您的 Aurora MySQL 資料庫叢集可以使用 AWS 服務來執行下列動作：

- 使用原生 AWS Lambda 函數 或 同步或非同步叫用 `lambda_sync` 函數`lambda_async`。或是，使用 AWS Lambda 程序非同步叫用 `mysql.lambda_async` 函式。
- 使用 `LOAD DATA FROM S3` 或 `LOAD XML FROM S3` 命令，從存放在 Amazon S3 儲存貯體中的文字檔或 XML 檔案，將資料載入資料庫叢集。
- 使用 `SELECT INTO OUTFILE S3` 命令，將資料從資料庫叢集儲存至在 Amazon S3 儲存貯體中存放的文字檔案。
- 使用 Application Auto Scaling 自動新增或移除 Aurora 複本。如需詳細資訊，請參閱使用 Amazon Aurora 自動擴展搭配 Aurora 複本。

如需整合 Aurora MySQL 與其他 AWS 服務的詳細資訊，請參閱 將 Amazon Aurora MySQL 與其他 AWS 服務整合。

將 AWS 服務與 Amazon Aurora PostgreSQL 整合

Amazon Aurora PostgreSQL 與其他 AWS 服務整合，因此您可以擴展 Aurora PostgreSQL 資料庫叢集，以在 AWS 雲端中使用其他功能。您的 Aurora PostgreSQL 資料庫叢集可以使用 AWS 服務來執行下列動作：

- 使用 Performance Insights 來快速收集、檢視和評估關聯式資料庫工作負載上的效能。
- 使用 Aurora Auto Scaling 自動新增或移除 Aurora 複本。如需詳細資訊，請參閱使用 Amazon Aurora 自動擴展搭配 Aurora 複本。

如需整合 Aurora PostgreSQL 與其他 AWS 服務的詳細資訊，請參閱 將 Amazon Aurora PostgreSQL 與其他 AWS 服務整合。

維持 為 Amazon Aurora 資料庫叢集

Amazon RDS 會定期在 Amazon RDS 資源上執行維護。下列主題說明這些維護動作，以及如何套用這些動作。

資料庫叢集維護更新概觀

維護通常涉及更新資料庫叢集中的以下資源：

- 基礎硬體
- 基礎作業系統 (OS)
- 資料庫引擎版本

作業系統更新大多是因為安全性問題。建議您盡快執行這些動作。如需作業系統更新的詳細資訊，請參閱 the section called “作業系統更新”。

主題

- 維護更新期間的離線資源
- 延遲資料庫執行個體和資料庫叢集修改
- DescribePendingMaintenanceActions API 的最終一致性

維護更新期間的離線資源

進行某些維護項目時，Amazon RDS 需要將您的資料庫叢集短暫離線。需要資源離線的維護項目包括必要的作業系統或資料庫修補。所需的修補程式僅會針對與安全性和執行個體可靠性相關的修補程式自動安排。這類修補不常發生，通常每隔幾個月進行一次。維護僅需片刻的時間即可完成。

延遲資料庫執行個體和資料庫叢集修改

您已選擇不立即套用的延遲資料庫叢集與執行個體修改，將會在下一次維護時段套用。例如，您可以選擇在維護時段變更資料庫執行個體類別或叢集或資料庫參數群組。您使用待定重新開機設定所指定的這類修改不會顯示在待定維護清單中。如需修改資料庫叢集的詳細資訊，請參閱修改 Amazon Aurora 資料庫叢集。

若要查看下一個維護時段待定的修改，請使用 describe-db-clusters AWS CLI 命令並檢查 `PendingModifiedValues` 欄位。

DescribePendingMaintenanceActions API 的最終一致性

Amazon RDS DescribePendingMaintenanceActions API 遵循最終一致性模式。這表示 DescribePendingMaintenanceActions 命令的結果可能不會立即顯示在所有後續 RDS 命令中。當您在使用先前的 API 命令後立即使用 DescribePendingMaintenanceActions 時，請記住這一點。

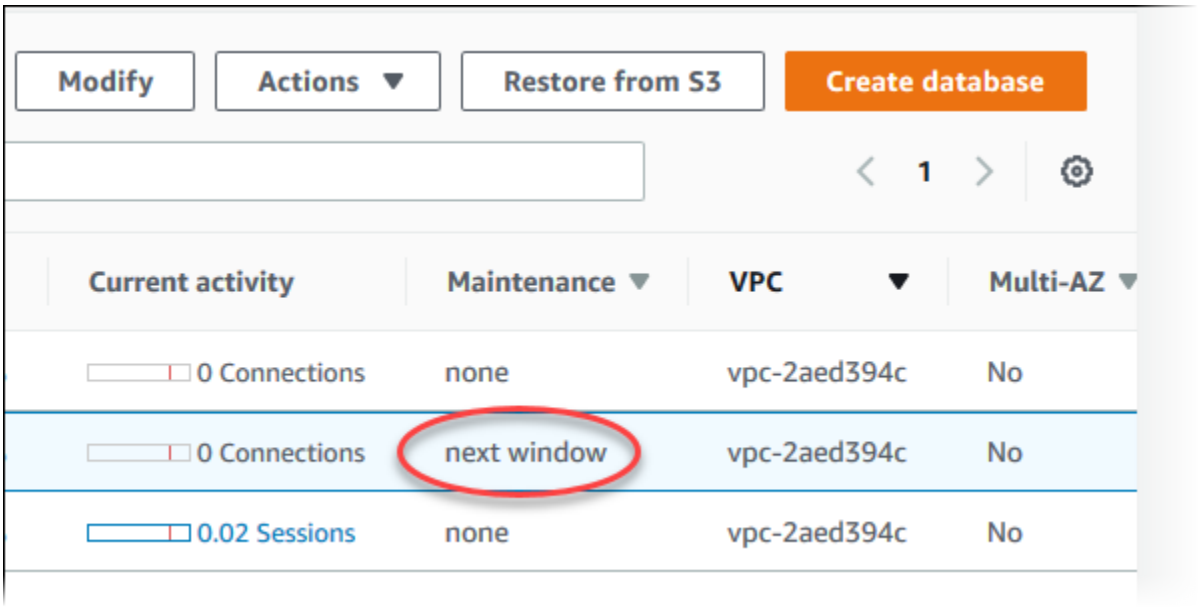
最終一致性可能會影響您管理維護更新的方式。例如，如果您執行 ApplyPendingMaintenanceActions 命令來更新資料庫叢集的資料庫引擎版本，它最終會顯示在 DescribePendingMaintenanceActions 中。在此案例中，DescribePendingMaintenanceActions 可能會顯示未套用維護動作，即使實際上已套用。

若要管理最終一致性，您可以執行下列操作：

- 在您執行命令來修改資料庫叢集之前，請先確認其狀態。使用指數退避演算法執行適當的 DescribePendingMaintenanceActions 命令，以確保您有足夠的時間讓上一個命令透過系統傳播。若要這樣做，請重複執行 DescribePendingMaintenanceActions 命令，從幾秒鐘的等待時間開始，然後逐漸增加到五分鐘的等待時間。
- 在後續命令之間增加等待時間，即使 DescribePendingMaintenanceActions 命令傳回準確的回應。從幾秒鐘的等待時間開始套用指數退避演算法，並逐漸增加到大約五分鐘的等待時間。

檢視待處理維護更新

使用 RDS 主控台 AWS CLI、 或 RDS API，檢視您的資料庫叢集是否可使用維護更新。若有可用的更新，Amazon RDS 主控台就會在資料庫叢集的維護欄中表示，如下圖所示。



如果資料庫叢集沒有可用的維護更新，欄的值就會是 none (無)。

如果資料庫叢集有可用的維護更新，就可能是以下的欄值：

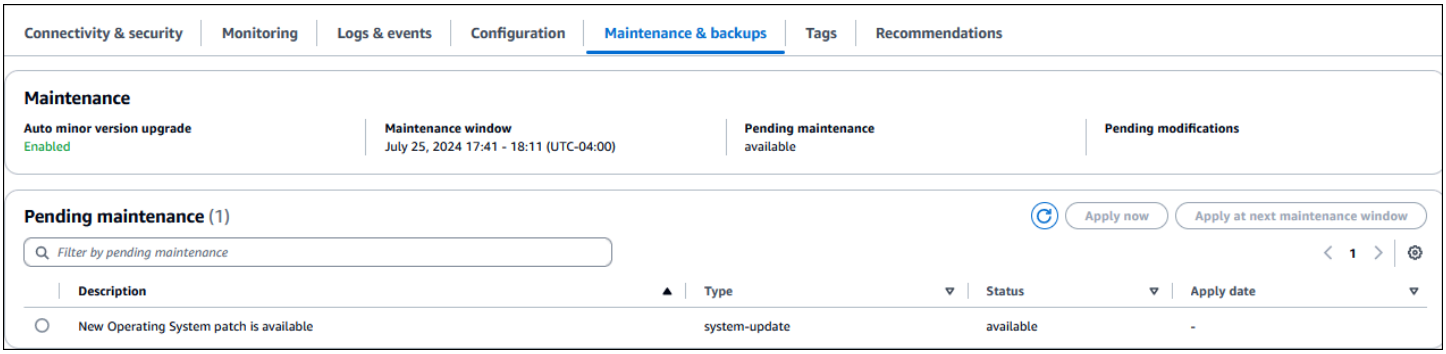
- 必要 – 維護動作將套用至資源，無法無限期延遲。
- available (可用) – 維護動作可用，但不會自動套用至資源。您可手動套用。
- next window (下個時段) – 維護動作會在下個維護時段中套用到資源。
- 進行中 – 維護動作正在套用至資源。

如有更新可用，您可執行下列操作之一：

- 如果維護值為下個時段，請由動作選擇延遲升級以延遲維護動作。如果維護動作已經啟動，則無法延遲維護動作。
- 立即套用維護動作。
- 在下一個維護時段期間套用維護動作。
- 不採取動作。

使用 採取動作 AWS 管理主控台

1. 選擇資料庫執行個體或叢集，以顯示其詳細資訊。
2. 選擇 Maintenance & backups (維護和備份)。待處理維護動作隨即顯示。
3. 選擇要採取的動作，然後選擇何時套用。



維護時段決定等待中的操作何時開始，但不限制這些操作的總執行時間。維護操作不保證在維護時段結束之前完成，可能持續到超過指定的結束時間。如需詳細資訊，請參閱Amazon RDS 維護時段。

您也可以執行 describe-pending-maintenance-actions AWS CLI 命令，檢視資料庫叢集是否有可用的維護更新。

如需套用維護更新的詳細資訊，請參閱 將更新套用至資料庫叢集。

Amazon Aurora 的維護動作

下列維護動作適用於 Aurora 資料庫叢集：

- `os-upgrade` – 使用滾動升級更新資料庫叢集中所有資料庫執行個體的作業系統。如需詳細資訊，請參閱the section called “作業系統更新”。
- `system-update` – 修補 Aurora PostgreSQL 的資料庫引擎。

下列維護動作適用於 Aurora 資料庫執行個體：

- `ca-certificate-rotation` – 更新資料庫執行個體的 Amazon RDS 憑證授權機構憑證。
- `hardware-maintenance` – 對資料庫執行個體的基礎硬體執行維護。
- `system-update` – 更新資料庫執行個體的作業系統。

選擇 Aurora MySQL 維護更新的頻率

您可以控制各個資料庫叢集經常或很少進行 Aurora MySQL 升級。最佳選擇依據您的 Aurora MySQL 使用情形以及在 Aurora 上執行的應用程式優先順序而定。如需不需經常更新的 Aurora MySQL 長期穩定性 (LTS) 版本的詳細資訊，請參閱 Aurora MySQL 長期支援 (LTS) 版本。

如果符合以下部分或所有條件，您或許可以選擇很少升級 Aurora MySQL 叢集：

- 每次 Aurora MySQL 資料庫引擎更新時，您的應用程式測試週期都需要較長的時間。
- 您有許多資料庫叢集或許多應用程式皆執行於相同的 Aurora MySQL 版本。您偏好同時升級所有資料庫叢集和關聯的應用程式。
- 您可以同時使用 Aurora MySQL 和 RDS for MySQL。您偏好維持 Aurora MySQL 叢集和 RDS for MySQL 資料庫執行個體與相同層級的 MySQL 相容。
- 您的 Aurora MySQL 應用程式已在生產環境或者是商業關鍵的。除了極少發生的關鍵修補程式之外，您無法承擔為了升級而停機所帶來的後果。
- 您的 Aurora MySQL 應用程式並未受到後續 Aurora MySQL 版本所解決的效能問題或功能差距的限制。

如果前述因素適用於您的情況，您可以限制 Aurora MySQL 資料庫叢集的強制升級次數。若要這麼做，請在建立或升級資料庫叢集時，選擇所謂「長期支援」(LTS) 的特定 Aurora MySQL 版本。這麼做可大幅減少資料庫叢集的升級週期、測試週期，以及升級相關停機的次數。

如果符合以下部分或所有條件，您或許可以選擇經常升級 Aurora MySQL 叢集：

- 您應用程式的測試週期很直接且簡短。
- 您的應用程式尚處於開發階段。
- 您的資料庫環境使用各種 Aurora MySQL 版本，或 Aurora MySQL 和 RDS for MySQL 版本。每個 Aurora MySQL 叢集皆各有其升級週期。
- 您正在等待特定效能或功能提升，然後才要增加 Aurora MySQL 的使用量。

如果上述因素適用於您的情況，您可讓 Aurora 更頻繁地套用重要的升級。若要這樣做，請將 Aurora MySQL 資料庫叢集升級至比 LTS 版本更新的 Aurora MySQL 版本。這麼做可讓您更快地取得最新的效能提升、錯誤修正和功能。

Amazon RDS 維護時段

維護時段是每週套用任何系統變更的時間間隔。每個資料庫叢集都有每週維護時段。維護時段是控制何時進行修改及軟體修補的時機。如需調整維護時段的詳細資訊，請參閱 調整偏好的資料庫叢集維護時段。

套用維護作業的期間，RDS 會使用資料庫叢集上的部分資源。您可能會發現，該操作對效能會造成些許影響。在極少數情況下，資料庫執行個體需要執行異地同步備份容錯移轉，才能完成維護更新作業。

若在特定某週排定維護事件，系統將在指定的 30 分鐘維護時段內啟動該事件。此外，多數維護事件也能在 30 分鐘的維護時段內完成，但較大型的維護事件可能需要 30 分鐘以上才能完成。資料庫叢集停止時，維護時段會暫停。

30 分鐘的維護時段是從每個區域之 8 小時時段內隨機選取的。若您在資料庫叢集建立期間，沒有指定維護時段，則 RDS 會在一週內隨機選取一天，並指派 30 分鐘的維護時段。

下表顯示指派 AWS 區域 預設維護時段的每個時段的時段。

區域名稱	區域	時間區塊
美國東部 (維吉尼亞北部)	us-east-1	03:00–11:00 UTC

區域名稱	區域	時間區塊
美國東部 (俄亥俄)	us-east-2	上午 3 時至 11 時 (UTC)
美國西部 (加利佛尼亞北部)	us-west-1	上午 6 時至下午 2 時 (UTC)
美國西部 (奧勒岡)	us-west-2	上午 6 時至下午 2 時 (UTC)
非洲 (開普敦)	af-south-1	上午 3 時至 11 時 (UTC)
亞太區域 (香港)	ap-east-1	上午 6 時至下午 2 時 (UTC)
亞太地區 (海德拉巴)	ap-south-2	06:30–14:30 UTC
亞太地區 (雅加達)	ap-southeast-3	08:00–16:00 UTC
亞太地區 (馬來西亞)	ap-southeast-5	09:00–17:00 UTC
亞太地區 (墨爾本)	ap-southeast-4	上午 11 時至下午 7 時 (UTC)
亞太區域 (孟買)	ap-south-1	上午 6 時至下午 2 時 (UTC)
亞太區域 (紐西蘭)	ap-southeast-6	下午 1 時至下午 9 時 (UTC)
亞太地區 (大阪)	ap-northeast-3	下午 10 時至 11 時 59 分 (UTC)
亞太區域 (首爾)	ap-northeast-2	下午 1 時至 9 時 (UTC)
亞太區域 (新加坡)	ap-southeast-1	下午 2 時至 10 時 (UTC)
亞太區域 (雪梨)	ap-southeast-2	中午 12 時至下午 8 時 (UTC)
亞太區域 (台北)	ap-east-2	9:00–17:00 UTC
亞太區域 (泰國)	ap-southeast-7	8:00–16:00 UTC
亞太區域 (東京)	ap-northeast-1	下午 1 時至 9 時 (UTC)
加拿大 (中部)	ca-central-1	03:00–11:00 UTC
加拿大西部 (卡加利)	ca-west-1	18:00–02:00 UTC

區域名稱	區域	時間區塊
中國 (北京)	cn-north-1	上午 6 時至下午 2 時 (UTC)
中國 (寧夏)	cn-northwest-1	上午 6 時至下午 2 時 (UTC)
歐洲 (法蘭克福)	eu-central-1	下午 9 時至上午 5 時 (UTC)
歐洲 (愛爾蘭)	eu-west-1	下午 10 時至上午 6 時 (UTC)
歐洲 (倫敦)	eu-west-2	22:00–06:00 UTC
歐洲 (米蘭)	eu-south-1	上午 2 時至 10 時 (UTC)
Europe (Paris)	eu-west-3	下午 11 時 59 分至上午 7 時 29 分 (UTC)
歐洲 (西班牙)	eu-south-2	上午 2 時至 10 時 (UTC)
歐洲 (斯德哥爾摩)	eu-north-1	下午 11 時至上午 7 時 (UTC)
歐洲 (蘇黎世)	eu-central-2	上午 2 時至 10 時 (UTC)
以色列 (特拉維夫)	il-central-1	03:00–11:00 UTC
墨西哥 (中部)	mx-central-1	19:00–3:00 UTC
Middle East (Bahrain)	me-south-1	上午 6 時至下午 2 時 (UTC)
中東 (阿拉伯聯合大公國)	me-central-1	上午 5 時至下午 1 時 (UTC)
南美洲 (聖保羅)	sa-east-1	上午 12 時至 8 時 (UTC)
AWS GovCloud （美國東部）	us-gov-east-1	下午 5 時至上午 1 時 (UTC)
AWS GovCloud （美國西部）	us-gov-west-1	上午 6 時至下午 2 時 (UTC)

主題

- 調整偏好的資料庫叢集維護時段

調整偏好的資料庫叢集維護時段

Aurora 資料庫叢集的維護時段應落在使用量最低的時間，因此可能需要不時進行調整。這段時間內，只有套用的更新需要停機時，資料庫叢集才會無法使用。停機是完成必要更新所需的最短時間。

> **Note**
>
> 執行資料庫引擎升級作業時，Amazon Aurora 會管理資料庫叢集的偏好維護時段，而非個別執行個體的維護時段。

主控台

調整偏好的資料庫叢集維護時段

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇資料庫叢集，以變更其維護時段。
4. 選擇 Modify (修改)。
5. 在 Maintenance (維護) 區段中，更新維護時段。
6. 選擇 Continue (繼續)。

   在確認頁面上，檢閱您的變更。
7. 若要立即將變更套用至維護時段，請在 Schedule of modifications (修改排程) 區段中選擇 Immediately (立即)。
8. 選擇 Modify cluster (修改叢集)，以儲存變更。

   或者，選擇 Back (上一步) 以編輯變更，或是選擇 Cancel (取消) 以取消變更。

AWS CLI

若要調整偏好的資料庫叢集維護時段，請使用 AWS CLI `modify-db-cluster`命令搭配下列參數：

- `--db-cluster-identifier`

- `--preferred-maintenance-window`

Example

以下程式碼範例將維護時段設為星期二早上 4:00–4:30 UTC。

針對 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
--db-cluster-identifier my-cluster \
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

在 Windows 中：

```
aws rds modify-db-cluster ^
--db-cluster-identifier my-cluster ^
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

RDS API

若要調整偏好的資料庫叢集維護時段，請使用 Amazon RDS `ModifyDBCluster` API 操作，並搭配下列參數：

- `DBClusterIdentifier`
- `PreferredMaintenanceWindow`

將更新套用至資料庫叢集

透過 Amazon RDS，即可自行選擇套用維護操作的時機。您可以使用 AWS 管理主控台 AWS CLI或 RDS API 來決定 Amazon RDS 何時套用更新。

主控台

管理資料庫叢集的更新內容

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇具有必要更新的資料庫叢集。
4. 針對 Actions (動作)，選擇下列其中一項：

   - 立即修補
   - 在下一個時段修補

   > **Note**
   >
   > 如果您選擇在下一個時段修補，但稍後想要延遲更新，則可以選擇延遲升級。如果維護動作已經啟動，則無法延遲維護動作。
   > 若要取消維護動作，請修改資料庫執行個體並停用 Auto minor version upgrade (自動次要版本升級)。

AWS CLI

若要將待定更新套用至資料庫叢集，請使用 apply-pending-maintenance-action AWS CLI 命令。

Example

針對 Linux、macOS 或 Unix：

```
aws rds apply-pending-maintenance-action \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db \
    --apply-action system-update \
    --opt-in-type immediate
```

在 Windows 中：

```
aws rds apply-pending-maintenance-action ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db ^
    --apply-action system-update ^
    --opt-in-type immediate
```

> **Note**
>
> 若要延期維護動作，請指定 undo-opt-in 的 --opt-in-type。如果維護動作已經啟動，則無法指定 undo-opt-in 的 --opt-in-type。
> 若要取消維護動作，請執行 modify-db-instance AWS CLI 命令並指定 --no-auto-minor-version-upgrade。

若要傳回至少有一個待處理更新的資源清單，請使用 describe-pending-maintenance-actions AWS CLI 命令。

Example

針對 Linux、macOS 或 Unix：

```
aws rds describe-pending-maintenance-actions \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

在 Windows 中：

```
aws rds describe-pending-maintenance-actions ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

您也可以指定 describe-pending-maintenance-actions AWS CLI 命令的 --filters 參數，以傳回資料庫叢集的資源清單。--filters 命令的格式為：Name=*filter-name*,Value=*resource-id*,...。

以下為篩選條件中，Name 參數可接受的值：

- db-instance-id – 可接受資料庫執行個體識別符或 Amazon Resource Name (ARN) 的清單。在系統所傳回的資料庫執行個體待處理維護動作清單中，只會包含以這些識別符或 ARN 識別的項目。
- db-cluster-id – 可接受資料庫叢集識別符或 Amazon Aurora ARN 的清單。在系統所傳回的資料庫叢集待處理維護動作清單中，只會包含以這些識別符或 ARN 識別的項目。

例如，下方範例將傳回 `sample-cluster1` 與 `sample-cluster2` 資料庫叢集的待處理維護動作。

Example

針對 Linux、macOS 或 Unix：

```
aws rds describe-pending-maintenance-actions \
 --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

在 Windows 中：

```
aws rds describe-pending-maintenance-actions ^
 --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

RDS API

若要將更新內容套用至資料庫叢集，請呼叫 Amazon RDS API `ApplyPendingMaintenanceAction` 操作。

若要傳回至少具有一項待處理更新的資源清單，請呼叫 Amazon RDS API `DescribePendingMaintenanceActions` 操作。

Aurora 資料庫叢集的自動次要版本升級

自動次要版本升級設定會指定 Aurora 是否自動將升級套用至資料庫叢集。這些升級包括新次要版本，其中包含其他功能，以及含有錯誤修正的修補程式。

自動次要版本升級會定期將資料庫更新為最新的資料庫引擎版本。不過，升級不一定包含最新的資料庫引擎版本。如果您需要在特定時間將資料庫維持在特定版本上，建議您根據所需的排程手動升級至所需的資料庫版本。如果發生重大安全問題或版本達到終止支援的日期，Amazon Aurora 可能會套用次要版本升級，即使您尚未啟用自動次要版本升級選項，也是如此。如需詳細資訊，請參閱適用於您特定資料庫引擎的升級文件。

請參閱 升級 Aurora MySQL 資料庫叢集的次要版本或修補程式層級 和 執行次要版本升級。

> **Note**
>
> Aurora 全球資料庫不支援自動次要版本升級。

此設定預設為開啟。針對每個新資料庫叢集，為此設定選擇適當的值。此值基於其重要性、預期的生命週期，以及每次升級之後所執行的驗證測試量。

如需了解如何關開啟或關閉自動次要版本升級設定，請參閱下列內容：

- 啟用 Aurora 資料庫叢集的自動次要版本升級
- 啟用 Aurora 資料庫叢集中別資料庫執行個體的自動次要版本升級

> **Important**
>
> 對於新的和現有的資料庫叢集，強烈建議將此設定套用至資料庫叢集，而不是個別套用至叢集中的資料庫執行個體。如果叢集中的任何資料庫執行個體已關閉此設定，則資料庫叢集不會自動升級。

下表顯示在叢集和執行個體層級套用自動次要版本升級設定時的運作方式。

Action	叢集設定	執行個體設定	叢集是否自動升級？
您在資料庫叢集上將其設定為 True。	True	對於所有新的和現有的執行個體為 False	是
您在資料庫叢集上將其設定為 False。	False	對於所有新的和現有的執行個體為 False	否
其先前在資料庫叢集上設定為 True。 您至少在一個資料庫執行個體上將其設定為 False。	變更為 False	對於一或多個執行個體為 False	否
其先前在資料庫叢集上設定為 False。 您至少在一個資料庫執行個體 (但並非所有	False	對於一或多個執行個體 (但並非所有執行個體) 為 True	否

Action	叢集設定	執行個體設定	叢集是否自動升級？
執行個體) 上將其設定為 False。			
其先前在資料庫叢集上設定為 False。 您在所有資料庫執行個體上將其設定為 True。	變更為 True	對於所有執行個體為 True	是

自動次要版本升級會透過 Amazon RDS 資料庫叢集事件預先傳達，該事件的類別為 `maintenance`，而其 ID 為 `RDS-EVENT-0156`。如需詳細資訊，請參閱Aurora 的 Amazon RDS 事件類別和事件訊息。

自動升級會在維護時段期間進行。如果資料庫叢集中的個別資料庫執行個體維護時段不同於叢集維護時段，則會優先考慮叢集維護時段。

如需 Aurora PostgreSQL 引擎更新的詳細資訊，請參閱 Amazon Aurora PostgreSQL 的資料庫引擎更新。

如需有關 Aurora MySQL 的 Auto minor version upgrade (自動次要版本升級) 設定的詳細資訊，請參閱 啟用次要 Aurora MySQL 版本之間的自動升級。如需 Aurora MySQL 引擎更新的一般資訊，請參閱 Amazon Aurora MySQL 的資料庫引擎更新。

啟用 Aurora 資料庫叢集的自動次要版本升級

依循 使用主控台、CLI 和 API 修改資料庫叢集 中的一般程序。

主控台

在修改資料庫叢集頁面的維護區段，選擇啟用自動次要版本升級的核取方塊。

AWS CLI

呼叫 modify-db-cluster AWS CLI 命令。為 `--db-cluster-identifier` 選項指定資料庫叢集的名稱，為 `true` 選項選擇 `--auto-minor-version-upgrade`。或者，指定 `--apply-immediately` 選項以立即為資料庫叢集啟用此設定。

RDS API

呼叫 ModifyDBCluster API 操作，並為 DBClusterIdentifier 參數指定您資料庫叢集的名稱，並為 AutoMinorVersionUpgrade 參數指定 true。或者，將 ApplyImmediately 參數設定為 true 以立即為資料庫叢集啟用此設定。

啟用 Aurora 資料庫叢集中別資料庫執行個體的自動次要版本升級

依循 修改資料庫叢集中的資料庫執行個體 中的一般程序。

主控台

在修改資料庫執行個體頁面的維護區段，選擇啟用自動次要版本升級的核取方塊。

AWS CLI

呼叫 modify-db-instance AWS CLI 命令。為 --db-instance-identifier 選項指定資料庫執行個體的名稱，為 true 選項選擇 --auto-minor-version-upgrade。或者，指定 --apply-immediately 選項以立即為資料庫執行個體啟用此設定。為叢集中的每個資料庫執行個體各自執行 modify-db-instance 命令。

RDS API

呼叫 ModifyDBInstance API 操作，並為 DBInstanceIdentifier 參數指定您資料庫叢集的名稱，並為 AutoMinorVersionUpgrade 參數指定 true。或者，將 ApplyImmediately 參數設定為 true 以立即為資料庫執行個體啟用此設定。為叢集中的每個資料庫執行個體各別呼叫 ModifyDBInstance 操作。

您可以使用下列 CLI 命令來檢查 Aurora MySQL 叢集中所有資料庫執行個體的AutoMinorVersionUpgrade升級設定狀態。

```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

此命令會產生類似下列的輸出：

```
[
  {
      "DBInstanceIdentifier": "db-writer-instance",
```

```
        "DBClusterIdentifier": "my-db-cluster-57",
        "AutoMinorVersionUpgrade": true
    },
    {
        "DBInstanceIdentifier": "db-reader-instance1",
        "DBClusterIdentifier": "my-db-cluster-57",
        "AutoMinorVersionUpgrade": false
    },
    {
        "DBInstanceIdentifier": "db-writer-instance2",
        "DBClusterIdentifier": "my-db-cluster-80",
        "AutoMinorVersionUpgrade": true
    },
... output omitted ...
```

在此範例中，資料庫叢集 `my-db-cluster-57` 的啟用自動次要版本升級已關閉，因為叢集的其中一個資料庫執行個體已關閉此功能。

Aurora 資料庫叢集的作業系統更新

Aurora MySQL 和 Aurora PostgreSQL 資料庫叢集中的資料庫執行個體偶爾需要作業系統更新。Amazon RDS 將作業系統升級至較新版本，以改善資料庫效能和客戶的整體安全狀態。通常，更新大約需要 10 分鐘。作業系統更新不會變更資料庫執行個體的資料庫引擎版本或資料庫執行個體類別。

作業系統更新有兩種類型，其區別在於待定維護動作的描述：

- 作業系統發佈升級 - 用來遷移至最新受支援的 Amazon Linux 主要版本。其描述為 `New Operating System upgrade is available`。
- 作業系統修補程式 - 用來套用各種安全性修正程式，有時可改善資料庫效能。其描述為 `New Operating System patch is available`。

作業系統更新可以是選用的，也可以是強制的。

- 可以隨時套用選用更新。雖然這些更新是選用的，但建議您定期套用更新，讓 RDS 機群保持最新狀態。RDS 不會自動套用這些更新。

  若要在新的選用作業系統修補程式可用時收到通知，您可以訂閱安全修補事件類別中的 RDS-EVENT-0230。如需訂閱 RDS 事件的相關資訊，請參閱 訂閱 Amazon RDS 事件通知。

> **Note**
>
> RDS-EVENT-0230 不適用於作業系統發行版升級。

- 需要強制更新，而且我們會在強制更新之前傳送通知。通知可能包含到期日。請規劃將更新排程在此到期日之前。在指定的到期日之後，Amazon RDS 會在您指派的其中一個維護時段期間自動將資料庫執行個體的作業系統升級至最新版本。

  作業系統發行版升級是必要的。

> **Note**
>
> 為了履行各種合規義務，可能需要將所有選用與強制更新保持為最新的狀態。建議您在維護時段期間定期套用 RDS 提供的所有更新。

對於 Aurora 資料庫叢集，您可以使用叢集層級維護選項來執行作業系統 (OS) 更新。在主控台中選取資料庫叢集的名稱時，在維護與備份索引標籤中尋找執行叢集層級更新的選項，或在 AWS CLI中使用 `os-upgrade` 命令。此方法透過滾動升級來保持讀取可用性，一次自動將更新套用至一些讀取器資料庫執行個體。為了防止多個容錯移轉並減少不必要的停機時間，Aurora 最後會升級寫入器資料庫執行個體。

叢集層級作業系統更新會在您為叢集指定的維護時段期間進行。這可確保整個叢集的協調更新。

為了回溯相容性，Aurora 也會維護執行個體層級維護選項。不過，我們建議您改用叢集層級更新。如果您必須使用執行個體層級更新，請先更新資料庫叢集中的讀取器資料庫執行個體，然後更新寫入器資料庫執行個體。如果您同時更新讀取器和寫入器執行個體，會增加容錯移轉相關停機時間的機會。在主控台中選取資料庫執行個體的名稱時，在維護與備份索引標籤中尋找執行執行個體層級更新的選項，或在 AWS CLI中使用 `system-update` 命令。

執行個體層級作業系統更新會在您為每個個別執行個體指定的維護時段期間進行。例如，如果叢集和兩個讀取器執行個體的維護時段不同，則叢集層級的作業系統更新會與叢集維護時段一致。

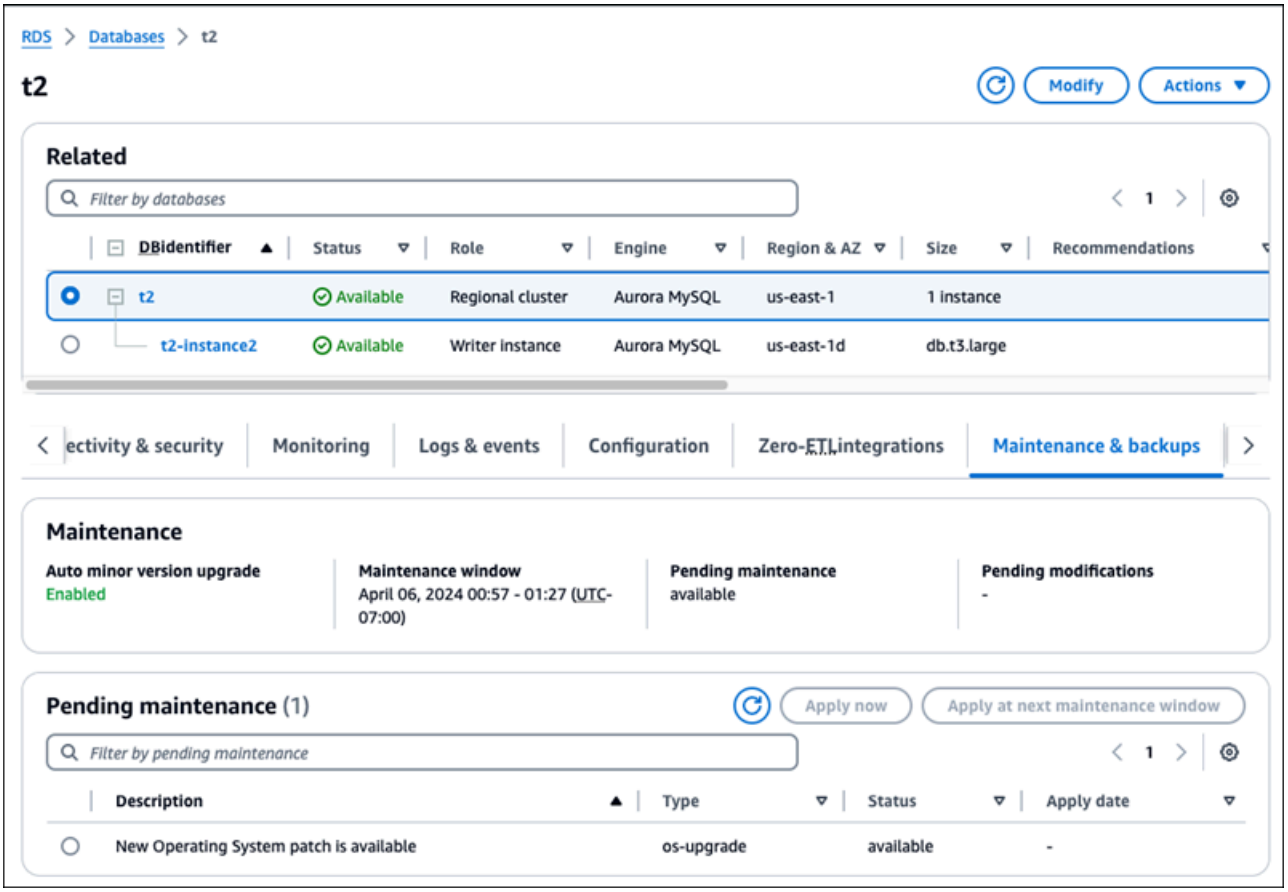
您可以使用 AWS 管理主控台 或 AWS CLI 來取得作業系統升級類型的相關資訊。

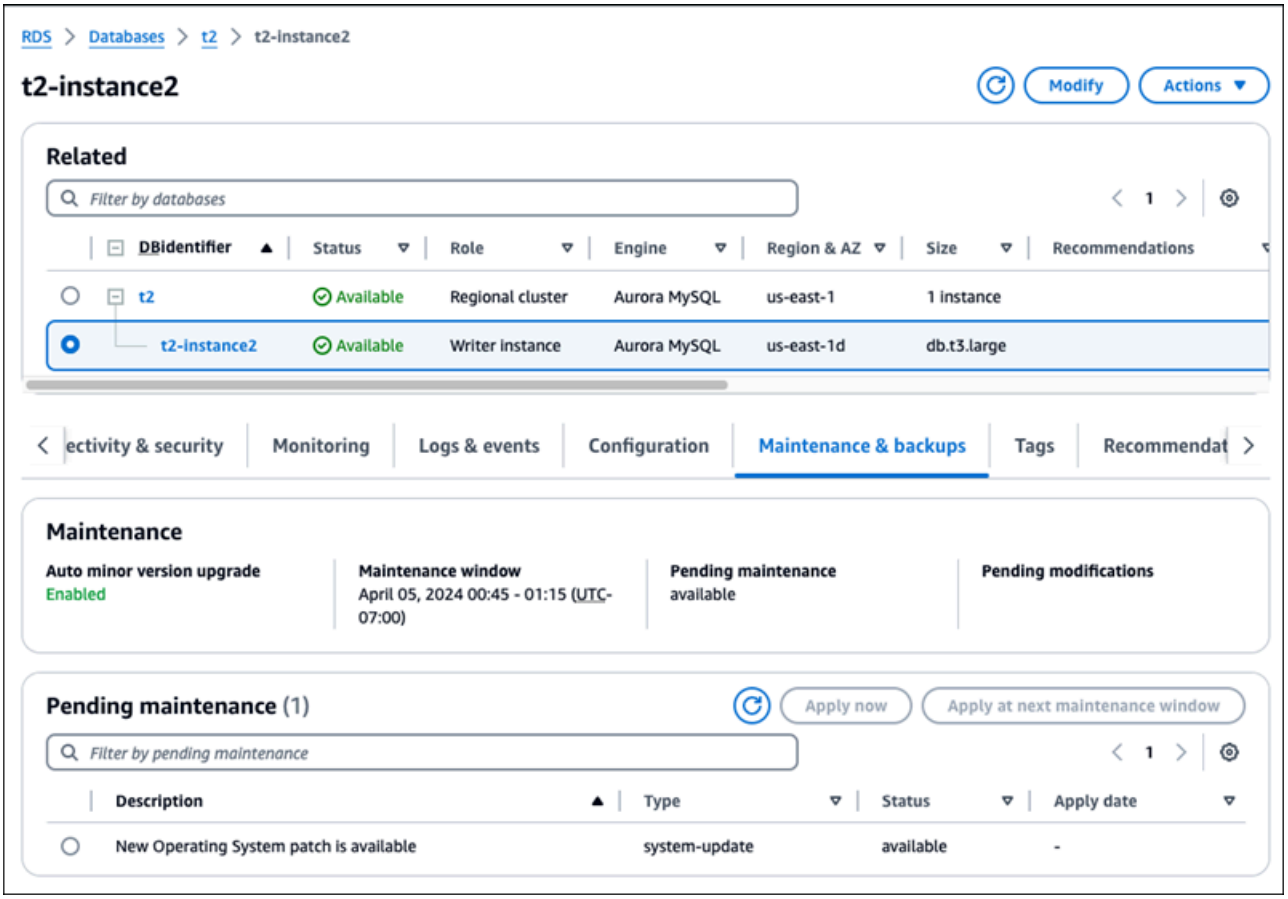
主控台

使用 取得更新資訊 AWS 管理主控台

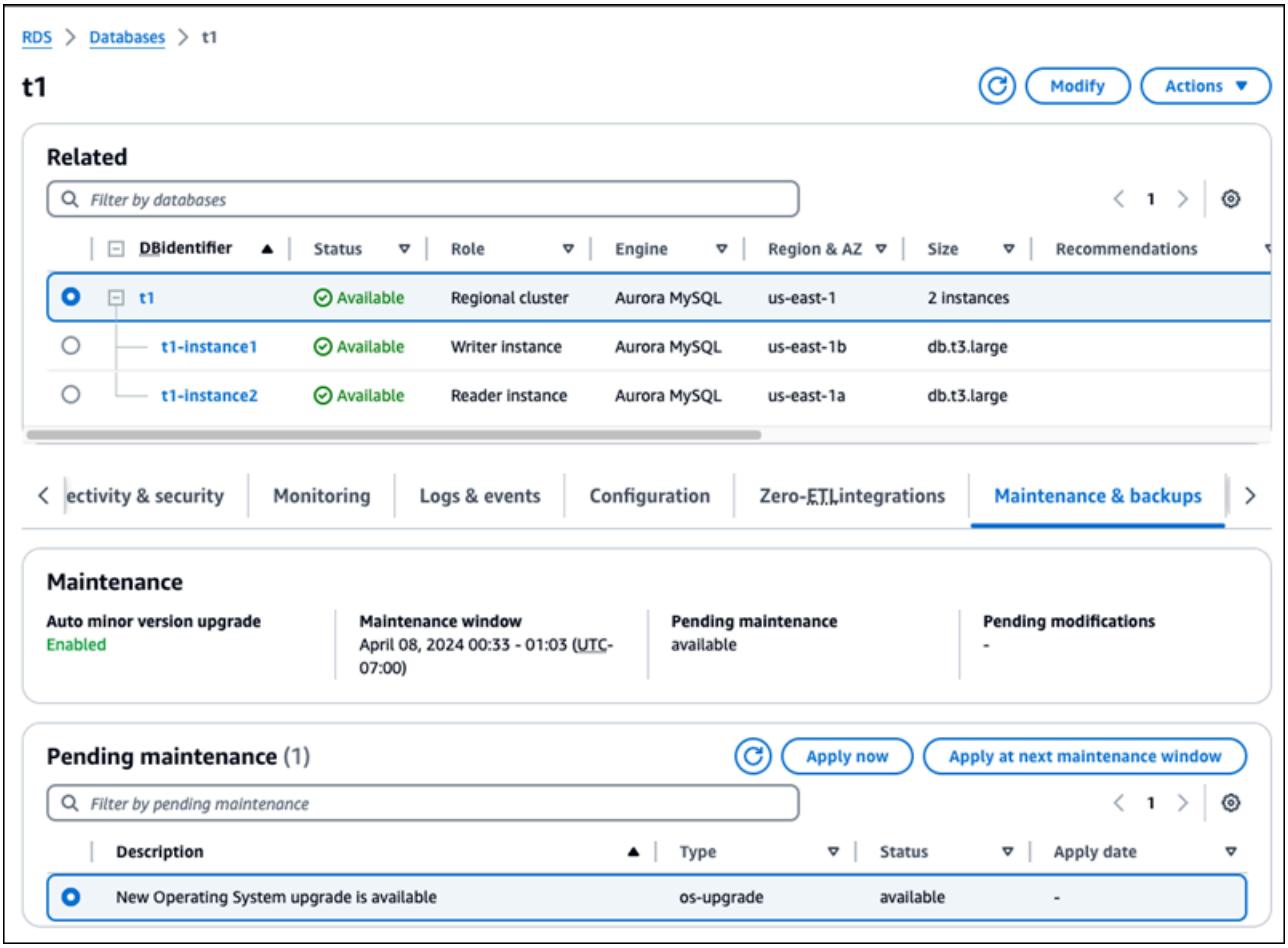
1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/rds/ 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)，然後選取資料庫執行個體。
3. 選擇 Maintenance & backups (維護和備份)。
4. 在待定維護區段中，尋找作業系統更新，然後檢查描述值。

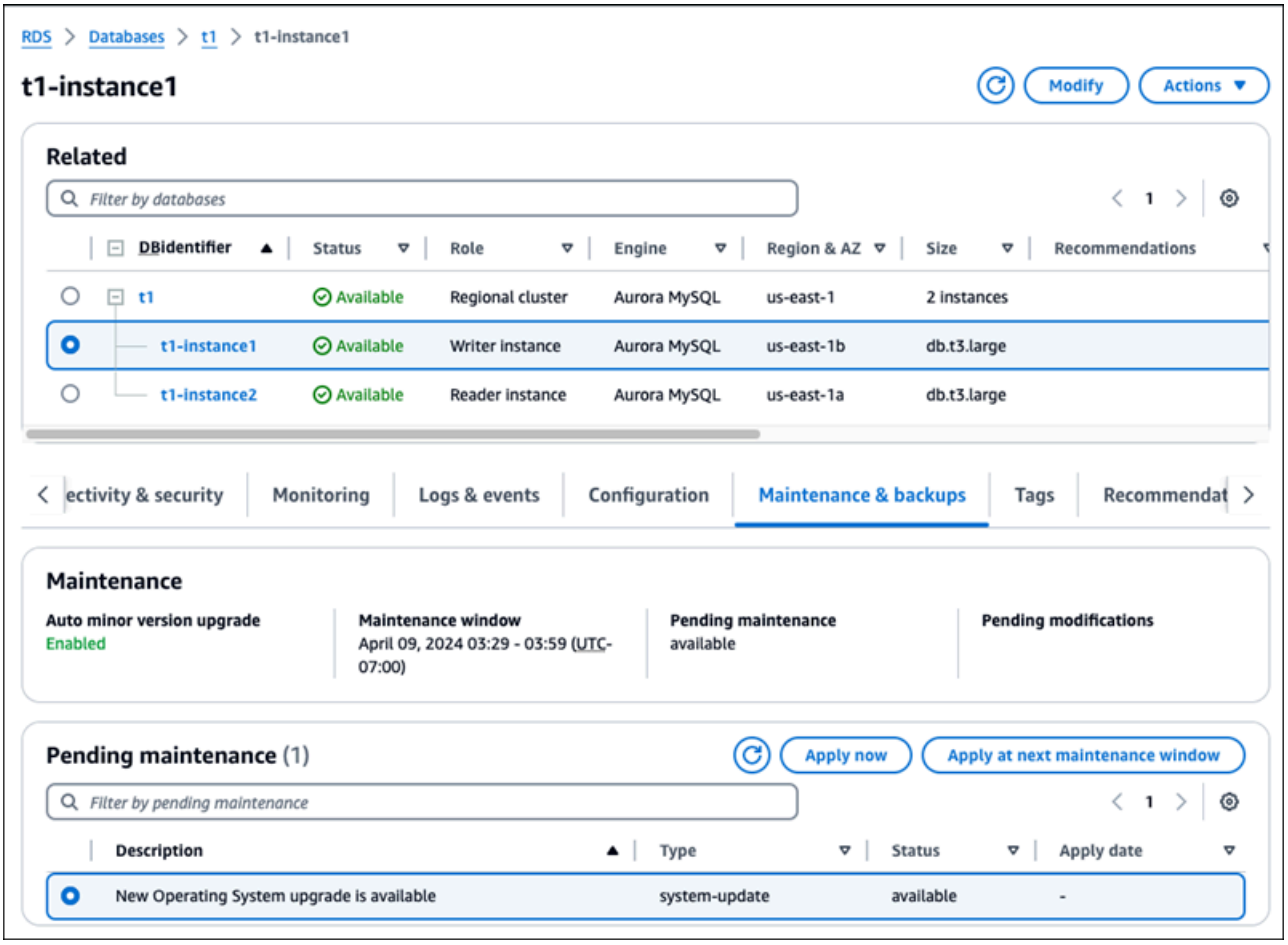
下圖顯示具有可用作業系統修補程式之寫入器資料庫執行個體的資料庫叢集。

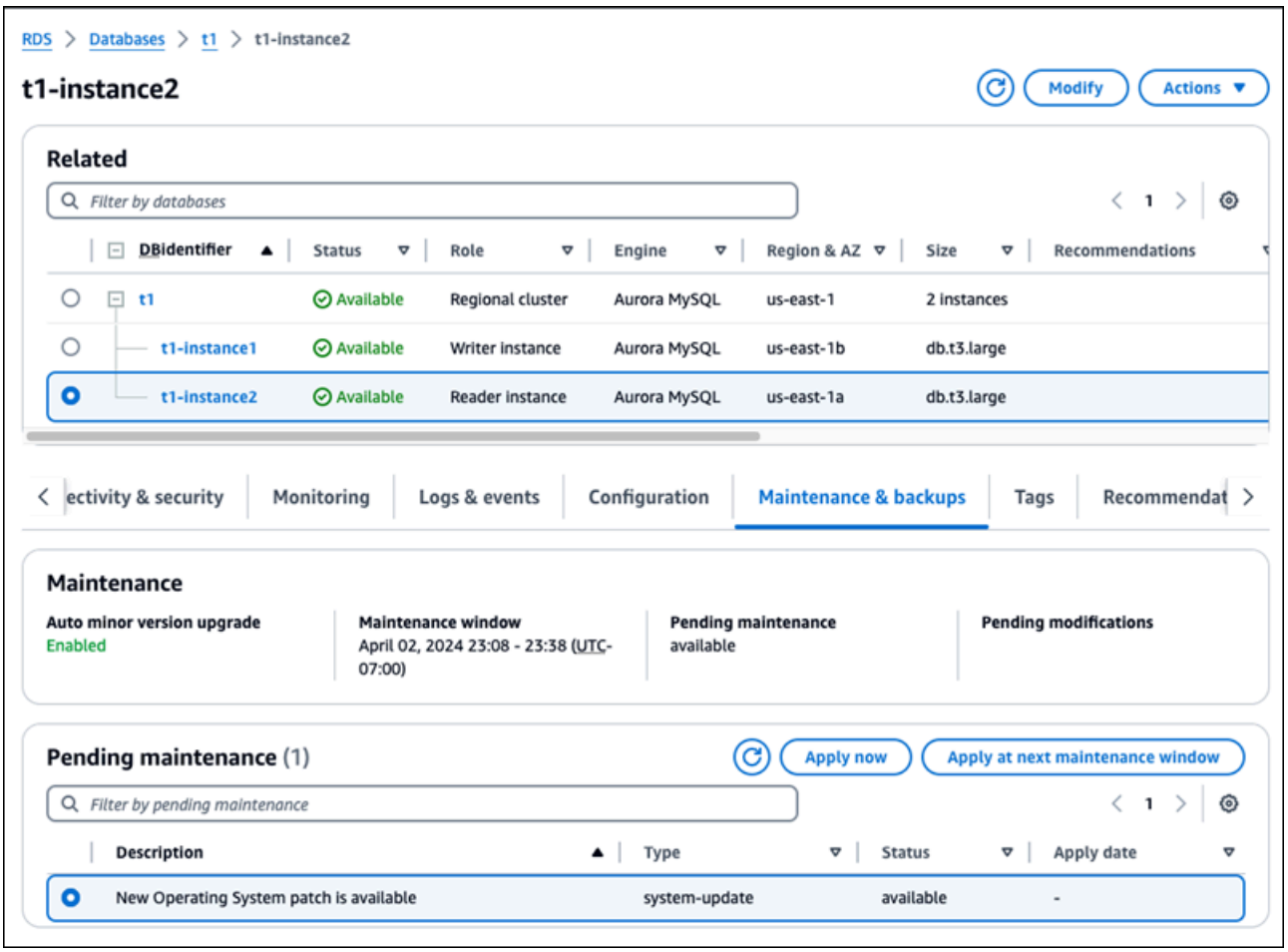




下圖顯示具有寫入器資料庫執行個體和讀取器資料庫執行個體的資料庫叢集。寫入器執行個體有可用的強制性作業系統升級。讀取器執行個體有可用的作業系統修補程式。


RDS > Databases > t1
t1
Modify
Actions
Related
Filter by databases
1
DBidentifier
Status
Role
Engine
Region & AZ
Size
Recommendations
t1
Available
Regional cluster
Aurora MySQL
us-east-1
2 instances
t1-instance1
Available
Writer instance
Aurora MySQL
us-east-1b
db.t3.large
t1-instance2
Available
Reader instance
Aurora MySQL
us-east-1a
db.t3.large
ectivity & security
Monitoring
Logs & events
Configuration
Zero-ETL integrations
Maintenance & backups
Maintenance
Auto minor version upgrade
Enabled
Maintenance window
April 08, 2024 00:33 - 01:03 (UTC-07:00)
Pending maintenance
available
Pending modifications
-
Pending maintenance (1)
Apply now
Apply at next maintenance window
Filter by pending maintenance
1
Description
Type
Status
Apply date
New Operating System upgrade is available
os-upgrade
available
-


RDS > Databases > t1 > t1-instance1
t1-instance1
Modify
Actions
Related
Filter by databases
1
DBidentifier
Status
Role
Engine
Region & AZ
Size
Recommendations
t1
Available
Regional cluster
Aurora MySQL
us-east-1
2 instances
t1-instance1
Available
Writer instance
Aurora MySQL
us-east-1b
db.t3.large
t1-instance2
Available
Reader instance
Aurora MySQL
us-east-1a
db.t3.large
ectivity & security
Monitoring
Logs & events
Configuration
Maintenance & backups
Tags
Recommendat
Maintenance
Auto minor version upgrade
Enabled
Maintenance window
April 09, 2024 03:29 - 03:59 (UTC-07:00)
Pending maintenance
available
Pending modifications
Pending maintenance (1)
Apply now
Apply at next maintenance window
Filter by pending maintenance
1
Description
Type
Status
Apply date
New Operating System upgrade is available
system-update
available
-



AWS CLI

若要從 取得更新資訊 AWS CLI，請使用 describe-pending-maintenance-actions 命令。

```
aws rds describe-pending-maintenance-actions
```

以下輸出顯示資料庫叢集和資料庫執行個體的作業系統發行版升級。

```
{
  "PendingMaintenanceActions": [
    {
      "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:cluster:t3",
      "PendingMaintenanceActionDetails": [
        {
          "Action": "os-upgrade",
          "Description": "New Operating System upgrade is available"
        }
      ]
```

```
    },
    {
      "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:db:t3-instance1",
      "PendingMaintenanceActionDetails": [
        {
          "Action": "system-update",
          "Description": "New Operating System upgrade is available"
        }
      ]
    },
  ]
}
```

以下輸出顯示資料庫執行個體的作業系統修補程式。

```
{
  "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:db:mydb2",
  "PendingMaintenanceActionDetails": [
    {
      "Action": "system-update",
      "Description": "New Operating System patch is available"
    }
  ]
}
```

作業系統更新的可用性

作業系統更新為資料庫引擎版本與資料庫執行個體類別所特有。因此，資料庫執行個體會在不同的時間接收或需要更新。當根據引擎版本和執行個體類別，有作業系統更新可供您的資料庫執行個體使用時，該更新會出現在主控台中。它也可以透過執行 describe-pending-maintenance-actions AWS CLI 命令或呼叫 DescribePendingMaintenanceActions RDS API 操作來檢視。如果執行個體有可用的更新，您可以依照將更新套用至資料庫叢集中的指示更新作業系統。

使用 AWS Organizations 升級推展政策進行自動次要版本升級

Aurora 支援 AWS Organizations 升級推展政策，以管理跨多個資料庫資源和 的自動次要版本升級 AWS 帳戶。此政策可協助您透過下列方式為叢集實作受控升級策略：

升級推展政策的運作方式

當新的次要引擎版本符合自動升級的資格時，政策會根據定義的順序控制升級順序：

- 標示為 【first】 （通常是開發環境） 的資源在其維護時段內符合升級資格。
- 在指定的製作時間之後，標記為 【秒】 的資源即符合資格。
- 在另一個指定的製作時間之後，標記為 【最後】 （通常為生產環境） 的資源即符合資格。
- 透過 AWS 運作狀態通知監控升級進度。

您可以透過以下方式定義升級訂單：

- 帳戶層級政策 – 套用至指定帳戶中的所有合格資源。
- 資源標籤 – 根據標籤套用至特定資源。

> **Note**
>
> 未設定升級政策或從政策中排除的資源會自動收到 【秒】 的升級順序。

先決條件

- 您的 AWS 帳戶 必須屬於 Organizations 中已啟用升級推展政策的組織
- 為您的叢集啟用自動次要版本升級
- 升級推展政策不需要標籤。如果您想要定義不同環境的特定升級訂單 （例如開發、測試、QA、生產），您可以使用標籤。如果您未在政策中包含標籤設定，則該政策下的所有資源都會遵循預設升級順序。對於 Aurora 資源，只有叢集層級標籤會用於升級推展政策，即使您的標籤是在執行個體層級定義。

標記您的 資源

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您要標記的叢集。
4. 選擇動作，然後選擇管理標籤。
5. 選擇 Add tag (新增標籤)。
6. 輸入您的標籤金鑰 （例如 'Environment') 和值 （例如 'Development')
7. 選擇新增標籤，然後選擇儲存。

您也可以使用 新增標籤 AWS CLI：

```
aws rds add-tags-to-resource \
    --resource-name arn:aws:rds:region:account-number:cluster:cluster-name \
    --tags Key=Environment,Value=Development
```

升級順序和階段

升級推展政策支援三個升級訂單：

- 【first】 - 通常用於開發或測試環境
- 【秒】 - 通常用於 QA 環境。如果未特別設定政策，資源的預設順序
- 【最後】 - 通常保留給生產環境

當新的次要引擎版本符合自動升級的資格時：

- 具有升級順序 【第一個】 的資源在其設定的維護時段內符合升級資格。
- 在指定的製作時間之後，具有升級順序 【秒】 的資源在其維護時段內符合升級資格。
- 在另一個指定的製作時間之後，具有升級順序 【最後】 的資源在其維護時段內符合升級資格。
- 自動次要版本升級行銷活動會在升級訂單 【first】、【second】 和 【last】 的所有合格資源升級後，或行銷活動達到排定的結束日期後關閉，以先到者為準。

> **Note**
>
> 所有自動次要版本升級都會在每個叢集設定的維護時段期間執行，以將對應用程式的潛在影響降至最低。

可觀測性

AWS 運作狀態和監控

您會收到 AWS 運作狀態通知：

- 開始自動次要版本升級行銷活動之前
- 在每個階段轉換之間協助追蹤和監控升級進度
- 進度更新顯示 AWS 運作狀態主控台中機群升級的資源數量

Amazon RDS 事件通知：

- 已啟用自動次要版本升級之資源的通知，包括：
  - 當您的資源根據升級順序 (【first】、【second】 或 【last】) 符合升級資格時
  - 維護時段期間的排程升級時間表
  - 個別資料庫升級開始和完成狀態
- 透過 Amazon EventBridge0 訂閱這些事件以進行自動化監控

考量事項

要記住的一些考量事項：

- 此政策適用於所有未來的自動次要版本升級行銷活動，包括在作用中行銷活動期間所做的政策變更。
- 如果您加入進行中的升級行銷活動，您的資源會遵循目前執行中的升級順序，而且不會等待設定的政策。
- 未設定升級政策或從政策中排除的資源會自動收到 【秒】 的升級順序。
- 此政策會在升級階段之間提供驗證期間，再繼續進行下一個階段。
- 政策或資源標籤的變更需要時間傳播，才能套用新的升級順序。
- 此政策僅適用於已啟用自動次要版本升級的 Aurora 資源。
- 如果您在 環境中偵測到問題，您可以關閉後續環境的自動次要版本升級，或使用驗證期間來解決問題，然後再繼續進行下一個升級訂單。

如需標記 RDS 資源的詳細資訊，請參閱 標記 Amazon Aurora 和 Amazon RDS 資源。如需設定和使用升級推展政策的詳細說明，請參閱AWS Organizations 《 使用者指南》中的 入門 AWS Organizations。

重新啟動 Amazon Aurora 資料庫叢集或 Amazon Aurora 資料庫執行個體

您可能需要重新啟動資料庫叢集或叢集中的某些執行個體，這通常是基於維護原因。例如，假設您修改參數群組內的參數，或將不同的參數群組與您的叢集建立關聯。在這些情況下，您必須重新啟動叢集，才能讓變更生效。同樣地，您可以在叢集中重新啟動一或多個讀取器資料庫執行個體。您可以安排個別執行個體的重新啟動操作，以將整個叢集的停機時間降到最低。

重新啟動叢集中每個資料庫執行個體所需的時間，取決於重新啟動時的資料庫活動。這也取決於您的特定資料庫引擎的復原流程。如果可行，請在開始重新啟動流程之前減少該特定執行個體上的資料庫活動。這樣做可以減少重新啟動資料庫所需的時間。

只有在叢集中的每個資料庫執行個體處於可用狀態時，您才能重新啟動每個資料庫執行個體。資料庫執行個體可能會因多種原因無法使用。其中包括叢集處於停止狀態、有修改套用至執行個體，以及存在維護時段動作，例如版本升級。

重新啟動資料庫執行個體，將重新啟動資料庫引擎流程。重新啟動資料庫執行個體會暫時中斷，在此期間，資料庫執行個體狀態設定為 rebooting (重新啟動中)。

> **Note**
>
> 例如，假如資料庫執行個體未在其關聯的資料庫參數群組中使用最新的變更，AWS 管理主控台 將顯示資料庫參數群組的狀態為 pending-reboot (等待重新啟動)。pending-reboot (等待重新啟動) 參數群組狀態不會在下一個維護時段期間造成自動重新啟動。如欲對該資料庫執行個體套用最新參數變更，請手動重新啟動資料庫執行個體。如需參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。

主題

- 在 Aurora 叢集中重新啟動資料庫執行個體
- 使用讀取可用性功能重新啟動 Aurora 叢集
- 在無讀取可用性的情況下重新啟動 Aurora 叢集
- 檢查 Aurora 叢集和執行個體的執行時間
- Aurora 重新啟動操作的範例

在 Aurora 叢集中重新啟動資料庫執行個體

此程序是您在使用 Aurora 執行重新啟動時的最重要操作。許多維護程序涉及以特定順序重新啟動一或多個 Aurora 資料庫執行個體。

主控台

重新啟動資料庫執行個體

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中選擇 Databases (資料庫)，然後選擇您要重新啟動的資料庫執行個體。
3. 針對 Actions (動作)，選擇 Reboot (重新啟動)。

   Reboot DB Instance (重新啟動資料庫執行個體) 頁面隨即出現。
4. 選擇 Reboot (重新啟動)，以重新啟動您的資料庫執行個體。

   或者選擇 Cancel (取消)。

AWS CLI

若要使用 AWS CLI 重新啟動資料庫執行個體，請呼叫 `reboot-db-instance` 命令。

Example

對於 Linux、macOS 或 Unix：

```
aws rds reboot-db-instance \
    --db-instance-identifier mydbinstance
```

在 Windows 中：

```
aws rds reboot-db-instance ^
    --db-instance-identifier mydbinstance
```

RDS API

若要使用 Amazon RDS API 重新啟動資料庫執行個體，請呼叫 `RebootDBInstance` 操作。

使用讀取可用性功能重新啟動 Aurora 叢集

憑藉讀取可用性功能，您可以重新啟動 Aurora 叢集的寫入器執行個體，而無需重新啟動主要或次要資料庫叢集中的讀取器執行個體。這樣做有助於維護叢集的高可用性，以進行讀取操作，同時重新啟動寫入器執行個體。您可以稍後依照您方便的排程重新啟動讀取器執行個體。例如，在生產叢集中，您可以一次重新啟動一個讀取器執行個體，只在主要執行個體重新啟動完成之後才開始。對於要重新啟動的每個資料庫執行個體，請遵循 在 Aurora 叢集中重新啟動資料庫執行個體 中的程序。

主要資料庫叢集的讀取可用性功能在 Aurora MySQL 2.10 版及更高版本中均有提供。Aurora MySQL 3.06 版及更新版本提供次要資料庫叢集的讀取可用性。

對於 Aurora PostgreSQL，此功能適用於下列版本：

- 15.2 版和更新的 15 版本
- 14.7 版和更新的 14 版本
- 13.10 版和更新的 13 版本
- 12.14 版和更新的 12 版本

如需 Aurora PostgreSQL 讀取可用性功能的詳細資訊，請參閱 改善 Aurora 複本的讀取可用性。

在此功能之前，重新啟動主要執行個體會導致所有讀取器執行個體同時重新啟動。如果您的 Aurora 叢集正在執行較舊版本，請改用 在無讀取可用性的情況下重新啟動 Aurora 叢集 中的重新啟動程序。

> **Note**
>
> 在具有讀取可用性的 Aurora 資料庫叢集中，對重新啟動行為所做的變更，與在低於 3.06 的 Aurora MySQL 版本中的 Aurora 全球資料庫的情況有所不同。如果重新啟動 Aurora 全域資料庫中主要叢集的寫入器執行個體，主要叢集中的讀取器執行個體仍然可用。不過，任何次要叢集中的資料庫執行個體會同時重新啟動。
>
> Aurora PostgreSQL 版本 12.16、13.12、14.9、15.4 及更高版本的 Aurora 全球資料庫支援有限版本的改善讀取可用性功能。

對叢集參數群組進行變更後，您經常重新啟動叢集。您可以依照 Amazon Aurora 的參數群組 中的程序進行參數變更。假設您重新啟動 Aurora 叢集中的寫入器資料庫執行個體，以將變更套用至叢集參數。部分或所有讀取器資料庫執行個體可能會繼續使用舊的參數設定。不過，不同的參數設定不會影響叢集的資料完整性。任何影響資料檔案組織的叢集參數，僅由寫入器資料庫執行個體使用。

例如在 Aurora MySQL 叢集中，您可以在讀取器執行個體之前更新叢集參數，例如寫入器執行個體上的 `binlog_format` 和 `innodb_purge_threads`。只有寫入器執行個體會寫入二進位日誌並清除復原紀錄。對於變更查詢解譯 SQL 陳述式或查詢輸出方式的參數，您可能需要謹慎地立即重新啟動讀取器執行個體。如果要在查詢期間避免未預期的應用程式行為，請您執行這項操作。例如，假設您變更 `lower_case_table_names` 參數並重新啟動寫入器執行個體。在這種情況下，在重新啟動讀取器執行個體之前，讀取器執行個體可能無法存取新建立的資料表。

如需所有 Aurora MySQL 叢集參數的清單，請參閱叢集層級參數。

如需所有 Aurora PostgreSQL 叢集參數的清單，請參閱 Aurora PostgreSQL 叢集層級參數。

> **Tip**
>
> 如果您的叢集正在處理具有高輸送量的工作負載，則 Aurora MySQL 可能仍會重新啟動部分讀取器執行個體以及寫入器執行個體。
> 在容錯移轉操作期間，重新啟動次數也會減少。Aurora MySQL 只會在容錯移轉期間重新啟動寫入器資料庫執行個體和容錯移轉目標。叢集中的其他讀取器資料庫執行個體仍可使用，以繼續透過連線至讀取器端點來處理查詢。因此，您可以在叢集中擁有多個讀取器資料庫執行個體，以提高容錯移轉期間的可用性。

在無讀取可用性的情況下重新啟動 Aurora 叢集

若無讀取可用性功能，要重新啟動整個 Aurora 資料庫叢集，您需要重新啟動該叢集的寫入器資料庫執行個體。若要執行此作業，請依照 中的程序進行在 Aurora 叢集中重新啟動資料庫執行個體

重新啟動寫入器資料庫執行個體，也會為叢集中的每個讀取器資料庫執行個體啟動重新啟動。如此一來，任何叢集範圍的參數變更都會同時套用至所有資料庫執行個體。不過，重新啟動所有資料庫執行個體會導致叢集的短暫中斷。讀取器資料庫執行個體會保持為無法使用，直到寫入器資料庫執行個體完成重新啟動並變得可已使用。

此重新開機行為適用於 Aurora MySQL 2.09 及較低版本中建立的所有資料庫叢集。

對於 Aurora PostgreSQL，此行為適用於下列版本：

- 14.6 和較低的 14 版本
- 13.9 和較低的 13 版本
- 12.13 和較低的 12 版本
- 所有 PostgreSQL 11 版本

在 RDS 主控台中，寫入器資料庫執行個體在 Databases (資料庫) 頁面的 Role (角色) 資料欄下具有的值為 Writer (寫入器)。在 RDS CLI 中，`describe-db-clusters` 命令的輸出包括 `DBClusterMembers` 部分。代表寫入器資料庫執行個體的 `DBClusterMembers` 元素，其 `true` 欄位的值為 `IsClusterWriter`。

> **Important**
>
> 使用讀取可用性功能時，在 Aurora MySQL 與 Aurora PostgreSQL 中的重新啟動行為有所不同：讀取器資料庫執行個體通常會在您重新啟動寫入器執行個體時保持為可用。然後，您可以在方便的時間重新啟動讀取器執行個體。如果您希望某些讀取器執行個體始終可用，您可以在交錯排程中重新啟動讀取器執行個體。如需更多詳細資訊，請參閱 使用讀取可用性功能重新啟動 Aurora 叢集。

檢查 Aurora 叢集和執行個體的執行時間

您可以檢查並監控 Aurora 叢集中自每個資料庫執行個體自上次重新啟動以來的時間長度。Amazon CloudWatch 指標 `EngineUptime` 報告自上次啟動資料庫執行個體以來的秒數。您可以在某個時間點檢查此指標，以找出資料庫執行個體的執行時間。您也可以隨時間監控此指標，以偵測何時重新啟動執行個體。

您也可以在叢集層級檢查 `EngineUptime` 指標。`Minimum` 和 `Maximum` 維度會報告叢集中所有資料庫執行個體的最小和最大執行時間值。若要檢查叢集中任何讀取器執行個體重新啟動或因其他原因而重新啟動的最近時間，請使用 `Minimum` 維度監控叢集層級指標。若要檢查叢集中的哪個執行個體在不重新啟動的情況下工作了最長時間，請使用 `Maximum` 維度監控叢集層級指標。例如，您可能想要確認叢集中的所有資料庫執行個體在組態變更後重新啟動。

> **Tip**
>
> 針對長期監控，我們建議您監控個別執行個體的 `EngineUptime` 指標，而不是在叢集層級監控。將新的資料庫執行個體新增至叢集時，叢集層級 `EngineUptime` 指標會設為零。這類叢集變更可能會作為維護和擴展操作的一部分發生，例如由 Auto Scaling 執行的操作。

下列 CLI 範例顯示如何檢查叢集中寫入器和讀取器執行個體的 `EngineUptime` 指標。這些範例使用名為 `tpch100g` 的叢集。此叢集具有寫入器資料庫執行個體 `instance-1234`。它還具有兩個讀取器資料庫執行個體 (`instance-7448` 和 `instance-6305`)。

首先，reboot-db-instance 命令會重新啟動其中一個讀取器執行個體。wait 命令會一直等待執行個體完成重新啟動。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceStatus": "rebooting",
...
$ aws rds wait db-instance-available --db-instance-id instance-6305
```

CloudWatch get-metric-statistics 命令會以一分鐘的間隔檢查過去五分鐘的 EngineUptime 指標。instance-6305 執行個體的執行時間會重設為零，並再次開始向上計數。此 Linux AWS CLI 範例使用$()變數替換，將適當的時間戳記插入 CLI 命令。它也會使用 Linux sort 命令，依據收集指標的時間排序輸出。該時間戳記值是輸出的每一行中的第三個欄位。

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 231.0 2021-03-16T18:19:00+00:00 Seconds
DATAPOINTS 291.0 2021-03-16T18:20:00+00:00 Seconds
DATAPOINTS 351.0 2021-03-16T18:21:00+00:00 Seconds
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
```

叢集的最短執行時間會重設為零，因為叢集中的一個執行個體已重新啟動。叢集的執行時間上限不會重設，因為叢集中至少有一個資料庫執行個體仍然可用。

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Minimum \
  --dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63099.0 2021-03-16T18:12:00+00:00 Seconds
DATAPOINTS 63159.0 2021-03-16T18:13:00+00:00 Seconds
DATAPOINTS 63219.0 2021-03-16T18:14:00+00:00 Seconds
DATAPOINTS 63279.0 2021-03-16T18:15:00+00:00 Seconds
DATAPOINTS 51.0 2021-03-16T18:16:00+00:00 Seconds
```

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63389.0 2021-03-16T18:16:00+00:00 Seconds
DATAPOINTS 63449.0 2021-03-16T18:17:00+00:00 Seconds
DATAPOINTS 63509.0 2021-03-16T18:18:00+00:00 Seconds
DATAPOINTS 63569.0 2021-03-16T18:19:00+00:00 Seconds
DATAPOINTS 63629.0 2021-03-16T18:20:00+00:00 Seconds
```

然後另一個 reboot-db-instance 命令會重新啟動叢集的寫入器執行個體。另一個 wait 命令會暫停，直到寫入器執行個體完成重新啟動為止。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
  "DBInstanceIdentifier": "instance-1234",
  "DBInstanceStatus": "rebooting",
...
$ aws rds wait db-instance-available --db-instance-id instance-1234
```

現在，寫入器執行個體的 EngineUptime 指標顯示最近已重新啟動執行個體 instance-1234。讀取器執行個體 instance-6305 也隨著寫入器執行個體自動重新啟動。這個叢集正在執行 Aurora MySQL 2.09，這不會讓讀取器執行個體在寫入器執行個體重新開機時執行。

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-1234 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63749.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 63809.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 63869.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 41.0 2021-03-16T18:25:00+00:00 Seconds
DATAPOINTS 101.0 2021-03-16T18:26:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
```

```
  --dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 531.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 49.0 2021-03-16T18:26:00+00:00 Seconds
```

Aurora 重新啟動操作的範例

下列 Aurora MySQL 範例顯示 Aurora 資料庫叢集中讀取器和寫入器資料庫執行個體的重新啟動操作的不同組合。每次重新啟動之後，SQL 查詢會展示叢集中執行個體的執行時間。

主題

- 尋找 Aurora 叢集的寫入器和讀取器執行個體
- 重新啟動單一讀取器執行個體
- 重新啟動寫入器執行個體
- 單獨重新啟動寫入器和讀取器
- 將叢集參數變更套用至 Aurora MySQL 版本 2.10 叢集

尋找 Aurora 叢集的寫入器和讀取器執行個體

在具有多個資料庫執行個體的 Aurora MySQL 叢集中，需要知道哪個是寫入器、哪些是讀取器。寫入器和讀取器執行個體也可以在發生容錯移轉操作時切換角色。因此，在執行任何需要寫入器或讀取器執行個體的操作之前，最好執行如下所示的檢查。在這種情況下，`False` 的 `IsClusterWriter` 值識別讀取器執行個體 (`instance-6305` 和 `instance-7448`)。`True` 值識別寫入器執行個體 `instance-1234`。

```
$ aws rds describe-db-clusters --db-cluster-id tpch100g \
  --query "*[].['Cluster:',DBClusterIdentifier,DBClusterMembers[*].
['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
  --output text
Cluster:     tpch100g
Instance:    instance-6305    False
Instance:    instance-7448    False
Instance:    instance-1234    True
```

在開始重新啟動的範例之前，寫入器執行個體擁有大約一週的執行時間。這個範例中的 SQL 查詢顯示了檢查執行時間的 MySQL 特定方式。您可以在資料庫應用程式中使用這種技術。如需使用 AWS CLI 和適用於兩個 Aurora 引擎的其他技術，請參閱 檢查 Aurora 叢集和執行個體的執行時間。

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status
    -> where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-08 17:49:06.000000 | 174h 42m|
+----------------------------+---------+
```

重新啟動單一讀取器執行個體

此範例會重新啟動一個讀取器資料庫執行個體。這個執行個體或許因一個巨大的查詢或許多並行連線而過載。其也可能因為網絡問題而落後於寫入器執行個體。開始重新啟動操作之後，範例會使用 `wait` 命令暫停，直到執行個體變成可用為止。此時，執行個體的執行時間為幾分鐘。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceStatus": "rebooting",
...
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-6305
$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status
    -> where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:35:02.000000 | 00h 03m |
+----------------------------+---------+
```

重新啟動讀取器執行個體並不會影響寫入器執行個體的執行時間。它仍然有大約一個星期的執行時間。

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+----------+
| Last Startup               | Uptime   |
+----------------------------+----------+
| 2021-03-08 17:49:06.000000 | 174h 49m |
+----------------------------+----------+
```

重新啟動寫入器執行個體

此範例會重新啟動寫入器執行個體。此叢集正在執行 Aurora MySQL 版本 2.09。因為 Aurora MySQL 版本低於 2.10，重新啟動寫入器執行個體也會重新啟動叢集中的任何讀取器執行個體。

`wait` 命令會暫停，直到重新啟動完成為止。現在該執行個體的執行時間會重設為零。對於寫入器和讀取器資料庫執行個體，重新啟動操作所用的時間可能大不相同。寫入器和讀取器資料庫執行個體會根據其角色執行不同種類的清理操作。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-1234",
        "DBInstanceStatus": "rebooting",
...
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-1234
$ mysql -h instance-1234.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:27.000000 | 00h 00m |
+----------------------------+---------+
```

在寫入器資料庫執行個體重新啟動之後，兩個讀取器資料庫執行個體也會重設其執行時間。重新啟動寫入器執行個體也會導致讀取器執行個體重新啟動。這種行為適用於 Aurora PostgreSQL 叢集和 2.10 版本之前的 Aurora MySQL 叢集。

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:35.000000 | 00h 00m |
+----------------------------+---------+

$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:33.000000 | 00h 01m |
+----------------------------+---------+
```

單獨重新啟動寫入器和讀取器

下列範例顯示執行 2.10 Aurora MySQL 版的叢集。在此 Aurora MySQL 版本及更高版本中，您可以重新啟動寫入器執行個體，而不會造成所有讀取器執行個體的重新啟動。如此一來，當您重新啟動寫入器執行個體時，查詢密集型應用程式就不會遇到任何中斷。您可以稍後重新啟動讀取器執行個體。您可以在查詢流量較低時執行這些重新啟動。您也可以一次重新啟動一個讀取器執行個體。如此，至少一個讀取器執行個體永遠可用於您的應用程式的查詢流量。

下列範例使用名為 `cluster-2393` 且執行 Aurora MySQL 版本 `5.7.mysql_aurora.2.10.0` 的叢集。這個叢集有一個名為 `instance-9404` 的寫入器執行個體，以及三個名為 `instance-6772`、`instance-2470` 和 `instance-5138` 的讀取器執行個體。

```
$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
  --query "*[].['Cluster:',DBClusterIdentifier,DBClusterMembers[*].
['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
  --output text
```

```
Cluster:        cluster-2393
Instance:       instance-5138        False
Instance:       instance-2470        False
Instance:       instance-6772        False
Instance:       instance-9404        True
```

使用 uptime 命令檢查每個資料庫執行個體的 mysql 值，會發現每個資料庫執行個體的執行時間大致相同。例如，這裡是 instance-5138 的執行時間。

```
mysql> SHOW GLOBAL STATUS LIKE 'uptime';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Uptime        | 3866  |
+---------------+-------+
```

透過使用 CloudWatch，我們無需實際登入到執行個體，就可以取得相應的執行時間資訊。如此一來，管理員就可以監控資料庫，但無法檢視或變更任何資料表資料。在這種情況下，我們指定跨越五分鐘的時間間隔，並每分鐘檢查執行時間值。不斷增加的執行時間值表明，該期間內未重新啟動執行個體。

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4648.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4708.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4768.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4828.0 2021-03-17T23:45:00+00:00 Seconds
DATAPOINTS 4888.0 2021-03-17T23:46:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-6772 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4315.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4375.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4435.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4495.0 2021-03-17T23:45:00+00:00 Seconds
```

```
DATAPOINTS 4555.0 2021-03-17T23:46:00+00:00 Seconds
```

現在，我們重新啟動其中一個讀取器執行個體 instance-5138。我們會等待執行個體在重新啟動後再次變成可用。現在，監控五分鐘間隔的執行時間顯示，執行時間在這段時間內重設為零。最近的執行時間值是在重新啟動完成後的五秒鐘測量。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-5138

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-5138 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4500.0 2021-03-17T23:46:00+00:00 Seconds
DATAPOINTS 4560.0 2021-03-17T23:47:00+00:00 Seconds
DATAPOINTS 4620.0 2021-03-17T23:48:00+00:00 Seconds
DATAPOINTS 4680.0 2021-03-17T23:49:00+00:00 Seconds
DATAPOINTS  5.0 2021-03-17T23:50:00+00:00 Seconds
```

接下來，我們為寫入器執行個體 instance-9404 執行重新啟動。我們會比較寫入器執行個體和其中一個讀取器執行個體的執行時間值。如此，我們可以看到重新啟寫入器並未導致讀取器重新啟動。在 Aurora MySQL 2.10 之前的版本中，所有讀取器的執行時間值將和寫入器在同一時間重設。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-9404
{
  "DBInstanceIdentifier": "instance-9404",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-9404

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
```

```
DATAPOINTS 371.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 431.0 2021-03-17T23:58:00+00:00 Seconds
DATAPOINTS 491.0 2021-03-17T23:59:00+00:00 Seconds
DATAPOINTS 551.0 2021-03-18T00:00:00+00:00 Seconds
DATAPOINTS  37.0  2021-03-18T00:01:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-6772 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 5215.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 5275.0 2021-03-17T23:58:00+00:00 Seconds
DATAPOINTS 5335.0 2021-03-17T23:59:00+00:00 Seconds
DATAPOINTS 5395.0 2021-03-18T00:00:00+00:00 Seconds
DATAPOINTS 5455.0 2021-03-18T00:01:00+00:00 Seconds
```

若要確定所有的讀取器執行個體具有與寫入器執行個體相同的組態參數變更，請在寫入器之後重新啟動所有讀取器執行個體。這個範例會重新啟動所有讀取器，然後等到所有讀取器都可以使用後再繼續。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6772
{
  "DBInstanceIdentifier": "instance-6772",
  "DBInstanceStatus": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
  "DBInstanceIdentifier": "instance-2470",
  "DBInstanceStatus": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}

$ aws rds wait db-instance-available --db-instance-id instance-6772
$ aws rds wait db-instance-available --db-instance-id instance-2470
$ aws rds wait db-instance-available --db-instance-id instance-5138
```

現在我們可以看到，寫入器資料庫執行個體具有最高的執行時間。此執行個體的執行時間值在整個監控期間穩定增加。讀取器資料庫執行個體都在讀取器之後重新啟動。我們可以看到監控期間內重新啟動每個讀取器以及將其執行時間重設為零的時間點。

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 457.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS 517.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 577.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 637.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS  697.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-2470 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 5819.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS  35.0  2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 95.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 155.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 215.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-5138 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 1085.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS 1145.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 1205.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS  49.0  2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 109.0 2021-03-18T00:12:00+00:00 Seconds
```

將叢集參數變更套用至 Aurora MySQL 版本 2.10 叢集

下列範例會展示如何將參數變更套用至 Aurora MySQL 2.10 叢集中的所有資料庫執行個體。使用此 Aurora MySQL 版本，您可以單獨地重新啟動寫入器執行個體和所有讀取器執行個體。

此範例會使用 MySQL 組態參數 lower_case_table_names 進行說明。當寫入器和讀取器資料庫執行個體之間的這個參數設定不一致時，查詢可能無法存取以大寫或大小寫混合名稱宣告的資料表。或者，如果兩個資料表名稱僅在大寫和小寫字母方面不同，則查詢可能會存取錯誤的資料表。

此範例會展示如何透過檢查每個執行個體的 IsClusterWriter 屬性來判定叢集中的寫入器和讀取器執行個體。叢集已命名為 cluster-2393。叢集具有名為 instance-9404 的寫入器執行個體。叢集中的讀取器執行個體名為 instance-5138 和 instance-2470。

```
$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' \
  --output text
cluster-2393
instance-5138        False
instance-2470        False
instance-9404        True
```

為了展示改變 lower_case_table_names 參數的影響，我們設定了兩個資料庫叢集參數群組。lower-case-table-names-0 參數群組將此參數設定為 0。lower-case-table-names-1 參數群組將此參數群組設定為 1。

```
$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-0' \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-cluster-parameter-group-name lower-case-table-names-0
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-0",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "lower-case-table-names-0"
    }
}

$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-1' \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-cluster-parameter-group-name lower-case-table-names-1
{
```

```
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-1",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "lower-case-table-names-1"
    }
}

$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lower-case-table-names-0 \
  --parameters
 ParameterName=lower_case_table_names,ParameterValue=0,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-0"
}

$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lower-case-table-names-1 \
    --parameters
 ParameterName=lower_case_table_names,ParameterValue=1,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-1"
}
```

lower_case_table_names 的預設值為 0。使用此參數設定時，資料表 foo 與表格 FOO 不同。此範例會驗證參數是否仍處於其預設設定。然後，此範例會建立三個資料表，它們的名稱僅存在大小寫差異。

```
mysql> create database lctn;
Query OK, 1 row affected (0.07 sec)

mysql> use lctn;
Database changed
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        0 |
+--------------------------+

mysql> create table foo (s varchar(128));
mysql> insert into foo values ('Lowercase table name foo');

mysql> create table Foo (s varchar(128));
```

```
mysql> insert into Foo values ('Mixed-case table name Foo');

mysql> create table FOO (s varchar(128));
mysql> insert into FOO values ('Uppercase table name FOO');

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+---------------------------+
| s                          |
+---------------------------+
| Mixed-case table name Foo |
+---------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Uppercase table name FOO |
+--------------------------+
```

接下來，我們將資料庫參數群組與叢集建立關聯，以將 lower_case_table_names 參數設定為 1。此變更只會在重新啟動每個資料庫執行個體後生效。

```
$ aws rds modify-db-cluster --db-cluster-identifier cluster-2393 \
  --db-cluster-parameter-group-name lower-case-table-names-1
{
  "DBClusterIdentifier": "cluster-2393",
  "DBClusterParameterGroup": "lower-case-table-names-1",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.0"
}
```

我們首先為寫入器資料庫執行個體執行重新啟動。然後，我們等待執行個體再次變成可用。此時，我們會連線到寫入器端點，並確認寫入器執行個體具有變更的參數值。SHOW TABLES 命令確認資料庫包含三個不同的資料表。不過，參考名為 foo Foo、或 FOO 資料表的查詢皆存取名稱為全小寫的資料表 foo。

```
# Rebooting the writer instance
$ aws rds reboot-db-instance --db-instance-identifier instance-9404
$ aws rds wait db-instance-available --db-instance-id instance-9404
```

現在，使用叢集端點的查詢會顯示參數變更的影響。無論查詢中的資料表名稱是大寫、小寫或大小寫混合，SQL 陳述式能存取名稱為全小寫的資料表。

```
mysql> select @@lower_case_table_names;
+---------------------------+
| @@lower_case_table_names |
+---------------------------+
|                         1 |
+---------------------------+

mysql> use lctn;
mysql> show tables;
+----------------+
| Tables_in_lctn |
+----------------+
| FOO            |
| Foo            |
| foo            |
+----------------+

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
```

```
+--------------------------+
```

下一個範例顯示與前一個查詢相同的查詢。在此情況下，查詢會使用讀取器端點，並在其中一個讀取器資料庫執行個體上執行。這些執行個體尚未重新啟動。因此，它們仍然具有 lower_case_table_names 參數的原始設定。這表示查詢可以存取每個 foo、Foo 和 FOO 資料表。

```
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        0 |
+--------------------------+

mysql> use lctn;

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+---------------------------+
| s                         |
+---------------------------+
| Mixed-case table name Foo |
+---------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Uppercase table name FOO |
+--------------------------+
```

接下來，我們重新啟動其中一個讀取器執行個體，並等待它再次變成可用。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
  "DBInstanceIdentifier": "instance-2470",
  "DBInstanceStatus": "rebooting"
```

```
}
$ aws rds wait db-instance-available --db-instance-id instance-2470
```

連線到 instance-2470 的執行個體端點時，查詢會顯示新參數有效。

```
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        1 |
+--------------------------+
```

此時，叢集中的兩個讀取器執行個體會以不同的 lower_case_table_names 設定執行。因此，叢集的讀取器端點的任何連線都會使用這個無法預期的設定值。請務必立即重新啟動其他讀取器執行個體，以便兩者都具有一致的設定。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-5138
```

下列範例會確認所有讀取器執行個體具有相同的 lower_case_table_names 參數設定。這些命令會檢查每個讀取器執行個體上的 lower_case_table_names 設定值。然後，使用讀取器端點的相同命令會展示每個與讀取器端點的連線都會使用其中一個讀取器執行個體，但哪一個是無法預測的。

```
# Check lower_case_table_names setting on each reader instance.

$ mysql -h instance-5138.a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-5138            |                        1 |
+--------------------------+--------------------------+

$ mysql -h instance-2470.a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
```

```
+--------------------------+--------------------------+
| instance-2470            |                        1 |
+--------------------------+--------------------------+

# Check lower_case_table_names setting on the reader endpoint of the cluster.

$ mysql -h cluster-2393.cluster-ro-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-5138            |                        1 |
+--------------------------+--------------------------+

# Run query on writer instance

$ mysql -h cluster-2393.cluster-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-9404            |                        1 |
+--------------------------+--------------------------+
```

隨著參數變更套用至所有位置，我們可以看到設定 lower_case_table_names=1 的效果。無論該資料表被稱為 foo、Foo 或 FOO，查詢都會將名稱轉換為 foo，並在每種情況下存取相同的資料表。

```
mysql> use lctn;

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+
```

```
mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+
```

容錯移轉 Amazon Aurora 資料庫叢集

您可以執行 Aurora 資料庫叢集的手動容錯移轉，例如，當您想要將佈建寫入器資料庫執行個體取代為 Aurora Serverless v2 寫入器執行個體時。

Aurora 會以下列兩種方式之一容錯移轉至新的主要資料庫執行個體：

- 將現有的讀取器資料庫執行個體提升至新的主要執行個體
- 建立新的主要執行個體

如果資料庫叢集有一或多個讀取器執行個體，則在失敗事件期間會將讀取器提升為主要執行個體。若要提高資料庫叢集的可用性，建議在兩個以上不同的可用區域建立至少一或多個讀取器執行個體。如需有關容錯移轉機制的詳細資訊，請參閱 Aurora 資料庫叢集的容錯能力。

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 來執行手動容錯移轉。

主控台

若要容錯移轉資料庫叢集

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中選擇資料庫，然後在您要容錯移轉的資料庫叢集中選取資料庫執行個體。
3. 針對 Actions (動作)，選擇 Failover (容錯移轉)。

   確認頁面隨即出現。
4. 選擇 Failover (容錯移轉)。

   資料庫頁面顯示資料庫叢集狀態為 Failing-over。狀態會在容錯移轉完成時傳回可用，並顯示新和舊主要資料庫執行個體的角色。

AWS CLI

若要使用 AWS CLI 容錯移轉資料庫叢集，請呼叫 failover-db-cluster 命令。指定下列參數：

- `--db-cluster-identifier` – 您要容錯移轉的資料庫叢集。
- `--target-db-instance-identifier` – 要提升為主要資料庫執行個體的資料庫執行個體名稱。

Example

對於 Linux、macOS 或 Unix：

```
aws rds failover-db-cluster \
    --db-cluster-identifier mydbcluster \
    --target-db-instance-identifier mydbcluster-instance-2
```

在 Windows 中：

```
aws rds failover-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --target-db-instance-identifier mydbcluster-instance-2
```

RDS API

若要使用 Amazon RDS API 修改資料庫叢集，請呼叫 FailoverDBCluster 操作。指定下列參數：

- DBClusterIdentifier
- TargetDBInstanceIdentifier

刪除 Aurora 資料庫叢集和資料庫執行個體

當您不再需要 Aurora 資料庫叢集時，您可以刪除它。刪除叢集會移除包含所有資料的叢集磁碟區。刪除叢集之前，您可以儲存資料的快照。您可以在稍後還原快照，以建立包含相同資料的新叢集。

您也可以從叢集刪除資料庫執行個體，同時保留叢集本身及其包含的資料。如果叢集不忙碌，或您不需要多個資料庫執行個體的計算容量，刪除資料庫執行個體可協助您減少費用。

主題

- 刪除 Aurora 資料庫叢集
- Aurora 叢集的刪除保護
- 刪除已停止的 Aurora 叢集
- 刪除僅供讀取複本的 Aurora MySQL 叢集
- 刪除叢集時的最終快照
- 從 Aurora 個體資料庫叢集刪除資料庫執行個體

刪除 Aurora 資料庫叢集

Aurora 不提供刪除資料庫叢集的單一步驟方法。此設計選項旨在防止您意外遺失資料或讓應用程式離線。Aurora 應用程式通常是關鍵任務，而且需要高可用性。因此，Aurora 透過新增和移除資料庫執行個體，可以輕鬆擴充和縮減叢集的容量。移除叢集本身需要您另行刪除。

使用下列一般程序，從叢集中移除所有資料庫執行個體，然後刪除叢集本身。

1. 刪除叢集中任一個讀取器執行個體。使用 從 Aurora 個體資料庫叢集刪除資料庫執行個體 中的程序。

   如果叢集有任一個讀取器執行個體，則刪除其中一個執行個體只會減少叢集的運算容量。首先刪除讀取器執行個體可確保叢集能夠在整個程序中保持可用，而且不會執行不必要的容錯移轉作業。
2. 從叢集中刪除寫入器執行個體。同樣地，請使用 從 Aurora 個體資料庫叢集刪除資料庫執行個體 中的程序。

   當您刪除資料庫執行個體時，即使刪除所有資料庫執行個體，叢集及其關聯的叢集磁碟區仍會保留。
3. 刪除資料庫叢集。
   - AWS 管理主控台 - 選擇叢集，然後從動作選單選擇刪除。您可以選擇下列選項來保留叢集中的資料，以備日後需要：

- 建立叢集磁碟區的最終快照。預設設定是建立最終快照。
- 保留自動備份。預設設定不保留自動備份。

> **Note**
>
> Aurora Serverless v1 資料庫叢集的自動備份不會保留。

Aurora 也會要求您確認是否要刪除叢集。

- CLI 和 API - 呼叫 `delete-db-cluster` CLI 命令或 `DeleteDBCluster` API 操作。您可以選擇下列選項來保留叢集中的資料，以備日後需要：
  - 建立叢集磁碟區的最終快照。
  - 保留自動備份。

> **Note**
>
> Aurora Serverless v1 資料庫叢集的自動備份不會保留。

主題

- 建立空 Aurora 叢集
- 使用單一資料庫執行個體刪除 Aurora 叢集
- 刪除具有多個資料庫執行個體的 Aurora 叢集

建立空 Aurora 叢集

您可以使用 AWS 管理主控台 AWS CLI或 Amazon RDS API 刪除空的資料庫叢集。

> **Tip**
>
> 您可以保留沒有資料庫執行個體的叢集，以保留資料，而不會對叢集產生 CPU 費用。您可以為叢集建立一個或多個新資料庫執行個體，快速開始再次使用叢集。不過，您只能使用 AWS CLI 或 RDS API 新增資料庫執行個體。您無法使用 Amazon RDS 主控台新增資料庫執行個體。當叢集沒有任何關聯的資料庫執行個體時，您可以在叢集上執行 Aurora 特定的管理作業。您只是無法存取資料或執行任何需要連線到資料庫執行個體的操作。

主控台

刪除資料庫叢集

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中選擇資料庫，然後選擇您要刪除的資料庫叢集。
3. 對於 Actions (動作)，請選擇 Delete (刪除)。
4. 若要建立資料庫叢集的最終資料庫快照，請選擇是否建立最終快照？。這是預設設定。
5. 如果您選擇建立最終快照，請輸入最終快照名稱。
6. 若要保留自動備份，請選擇 Retain automated backups (保留自動備份)。這不是預設設定。
7. 在方塊中輸入 **`delete me`**。
8. 選擇 刪除。

CLI

若要使用 刪除空的 Aurora 資料庫叢集 AWS CLI，請呼叫 delete-db-cluster 命令。

假設空叢集 `deleteme-zero-instances` 僅用於開發和測試，並且不包含任何重要資料。在這種情況下，刪除叢集時，不需要保留叢集磁碟區的快照。下列範例會示範叢集不包含任何資料庫執行個體，然後刪除空叢集，而不需建立最終快照或保留自動備份。

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-zero-instances --output
 text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-zero-instances

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-zero-instances \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-zero-instances",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

RDS API

若要使用 Amazon RDS API 刪除空的 Aurora 資料庫叢集，請呼叫 DeleteDBCluster 操作。

使用單一資料庫執行個體刪除 Aurora 叢集

即使資料庫叢集已啟用刪除保護功能，您仍然可以刪除最後一個資料庫執行個體。在這種情況下，資料庫叢集本身仍會存在，資料將獲得保留。您可將新的資料庫執行個體連接至叢集，即可再次存取資料。

下列範例顯示叢集仍有相關聯的資料庫執行個體時，delete-db-cluster 命令無法運作。此叢集具有單一寫入器資料庫執行個體。當我們檢查叢集中的資料庫執行個體時，我們會檢查每個執行個體的 IsClusterWriter 屬性。叢集可以有零個或一個寫入器資料庫執行個體。true 的值表示寫入器資料庫執行個體。false 的值表示讀取器資料庫執行個體。叢集可以有零個、一個或多個讀取器資料庫執行個體。在這種情況下，我們使用 delete-db-instance 命令刪除寫入器資料庫執行個體。一旦資料庫執行個體進入 deleting 狀態，我們也可以刪除叢集。在此範例中，假設叢集不包含任何值得保留的資料。因此，我們不會建立叢集磁碟區的快照或保留自動備份。

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only --skip-final-
snapshot
An error occurred (InvalidDBClusterStateFault) when calling the DeleteDBCluster
 operation:
  Cluster cannot be deleted, it still contains DB instances in non-deleting state.

$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-only \
  --query '*[].[DBClusterIdentifier,Status,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]'
[
    [
        "deleteme-writer-only",
        "available",
        [
            [
                "instance-2130",
                true
            ]
        ]
    ]
]

$ aws rds delete-db-instance --db-instance-identifier instance-2130
{
  "DBInstanceIdentifier": "instance-2130",
```

```
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-writer-only",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

刪除具有多個資料庫執行個體的 Aurora 叢集

如果您的叢集包含多個資料庫執行個體，通常會有單一寫入器執行個體和一個或多個讀取器執行個體。讀取器執行個體提供具備高可用性的協助，藉由處於待命狀態，在寫入器執行個體遇到問題來進行接管。您也可以使用讀取器執行個體來調整叢集以處理大量讀取的工作負載，而不會為寫入器執行個體增加額外負荷。

若要刪除具有多個讀取器資料庫執行個體的叢集，請先刪除讀取器執行個體，然後刪除寫入器執行個體。刪除寫入器執行個體會保留叢集及其資料。您可透過個別動作刪除叢集。

- 如需刪除 Aurora 資料庫執行個體的程序，請參閱 從 Aurora 個體資料庫叢集刪除資料庫執行個體。
- 如需刪除 Aurora 叢集中寫入器資料庫執行個體的程序，請參閱 使用單一資料庫執行個體刪除 Aurora 叢集。
- 如需刪除空 Aurora 叢集的程序，請參閱 建立空 Aurora 叢集。

此 CLI 範例顯示如何刪除包含寫入器資料庫執行個體和單一讀取器資料庫執行個體的叢集。`describe-db-clusters` 輸出顯示 `instance-7384` 是寫入器執行個體，`instance-1039` 是讀取器執行個體。此範例會先刪除讀取器執行個體，因為在讀取器執行個體仍然存在時刪除寫入器執行個體會導致容錯移轉作業。如果您打算也刪除該執行個體，將讀取器執行個體提升為寫入器是沒有意義的。同樣地，假設這些 `db.t2.small` 執行個體僅用於開發和測試，刪除作業也會略過最後的快照也不會保留自動備份。

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader --skip-
final-snapshot

An error occurred (InvalidDBClusterStateFault) when calling the DeleteDBCluster
 operation:
```

```
  Cluster cannot be deleted, it still contains DB instances in non-deleting state.

$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-and-reader --
output text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-writer-and-reader
Instance:       instance-1039  False
Instance:       instance-7384   True

$ aws rds delete-db-instance --db-instance-identifier instance-1039
{
  "DBInstanceIdentifier": "instance-1039",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-7384
{
  "DBInstanceIdentifier": "instance-7384",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-writer-and-reader",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

下列範例顯示如何刪除包含寫入器資料庫執行個體和多個讀取器資料庫執行個體的資料庫叢集。它使用來自 `describe-db-clusters` 命令的簡潔輸出來獲取寫入器和讀取器執行個體的報告。同樣地，在刪除寫入器資料庫執行個體之前，我們會先刪除所有的讀取器執行個體。我們刪除讀取器資料庫執行個體的順序無關緊要。

假設這個具有多個資料庫執行個體的叢集確實包含值得保留的資料。因此，此範例中的 `delete-db-cluster` 命令包含 `--no-skip-final-snapshot` 和 `--final-db-snapshot-identifier` 參數，以指定要建立的快照詳細資訊。它還包括 `--no-delete-automated-backups` 保留自動備份的參數。

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-multiple-readers --
output text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-multiple-readers
Instance:       instance-1010   False
Instance:       instance-5410   False
Instance:       instance-9948   False
Instance:       instance-8451   True

$ aws rds delete-db-instance --db-instance-identifier instance-1010
{
  "DBInstanceIdentifier": "instance-1010",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-5410
{
  "DBInstanceIdentifier": "instance-5410",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-9948
{
  "DBInstanceIdentifier": "instance-9948",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-8451
{
  "DBInstanceIdentifier": "instance-8451",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-multiple-readers \
  --no-delete-automated-backups \
  --no-skip-final-snapshot \
  --final-db-snapshot-identifier deleteme-multiple-readers-final-snapshot
{
```

```
  "DBClusterIdentifier": "deleteme-multiple-readers",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

下列範例顯示如何確認 Aurora 建立要求的快照。您可以指定特定快照的識別碼 deleteme-multiple-readers-final-snapshot 來要求詳細資訊。您也可以透過指定叢集識別碼 deleteme-multiple-readers，取得已刪除叢集的所有快照報告。這兩個命令都會傳回相同快照的相關資訊。

```
$ aws rds describe-db-cluster-snapshots \
  --db-cluster-snapshot-identifier deleteme-multiple-readers-final-snapshot
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-final-snapshot",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
...

$ aws rds describe-db-cluster-snapshots --db-cluster-identifier deleteme-multiple-
readers
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-final-snapshot",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
...
```

Aurora 叢集的刪除保護

您無法刪除已啟用刪除保護的叢集。您可以刪除叢集中的資料庫執行個體，但不能刪除叢集本身。如此一來，您便可避免意外刪除包含所有資料的叢集磁碟區。無論您嘗試使用主控台 AWS CLI、 或 RDS API 刪除叢集，Aurora 都會對資料庫叢集強制執行刪除保護。

當您使用 AWS 管理主控台建立生產資料庫叢集時，預設會啟用刪除保護。不過，如果您使用 AWS CLI 或 API 建立叢集，則預設會停用刪除保護。啟用或停用刪除保護不會導致停機。若要能夠刪除叢

集，請修改叢集並停用刪除保護。如需更多詳細資訊，瞭解如何開啟及關閉刪除保護功能，請參閱使用主控台、CLI 和 API 修改資料庫叢集。

> **Tip**
>
> 即使刪除所有資料庫執行個體，您也可以在叢集中建立新的資料庫執行個體來存取資料。

刪除已停止的 Aurora 叢集

如果叢集處於 `stopped` 狀態，則無法刪除叢集。在此情況下，請先啟動叢集，再刪除叢集。如需更多詳細資訊，請參閱 啟動 Aurora 資料庫叢集。

刪除僅供讀取複本的 Aurora MySQL 叢集

針對 Aurora MySQL，如果以下條件成立，則無法刪除資料庫叢集中的資料庫執行個體：

- 此資料庫叢集為另一個 Aurora 資料庫叢集的僅供讀取複本。
- 此資料庫執行個體是資料庫叢集中唯一的執行個體。

若要在此情況下刪除資料庫執行個體，請先提升資料庫叢集，使其不再是僅供讀取複本。升級完畢後，您可以刪除資料庫叢集中的最後一個資料庫執行個體。如需更多詳細資訊，請參閱 跨 AWS 區域 複寫 Amazon Aurora MySQL 資料庫叢集。

刪除叢集時的最終快照

在本節中，範例會顯示如何在刪除 Aurora 叢集時選擇是否要拍攝最終快照。如果您選擇拍攝最終快照，但您指定的名稱與現有快照相符，則操作會停止並顯示錯誤。在此情況下，請檢查快照詳細資訊，以確認它是否代表您目前的詳細資料，或是否為較舊的快照。如果現有的快照集沒有您想要保留的最新資料，請重新命名快照集後再試一次，或為最終快照參數指定不同的名稱。

從 Aurora 個體資料庫叢集刪除資料庫執行個體

您可以從 Aurora 資料庫叢集刪除資料庫執行個體，作為刪除整個叢集程序的一部分。如果您的叢集包含一定數量的資料庫執行個體，則刪除叢集需要刪除每個資料庫執行個體。您也可以在叢集保持執行狀態時刪除叢集中的一個或多個讀取器執行個體。如果叢集不忙碌，您可以這麼做來減少運算容量和相關費用。

若要刪除資料庫執行個體，您必須指定執行個體的名稱。

您可以使用 AWS CLI、 或 RDS API AWS 管理主控台刪除資料庫執行個體。

> **Note**
>
> 在刪除 Aurora 複本時，隨即會移除其執行個體端點，並且會從讀取器端點移除 Aurora 複本。若在要刪除的 Aurora 複本上有陳述式正在執行，則會有三分鐘的寬限期。現有陳述式在寬限期期間內可以完成。在寬限期結束時，Aurora 複本會關閉並刪除。

對於 Aurora 資料庫叢集，刪除資料庫執行個體不一定會刪除整個叢集。您可以刪除 Aurora 叢集中的資料庫執行個體，以便在叢集不忙碌時減少運算容量和相關費用。如需了解具有一個資料庫執行個體或零個資料庫執行個體之 Aurora 叢集的特殊情況，請參閱 使用單一資料庫執行個體刪除 Aurora 叢集 和 建立空 Aurora 叢集。

> **Note**
>
> 啟用資料庫叢集的刪除保護時，無法刪除該資料庫執行個體。如需更多詳細資訊，請參閱 Aurora 叢集的刪除保護。
> 您可以修改資料庫叢集來停用刪除保護。如需詳細資訊，請參閱修改 Amazon Aurora 資料庫叢集。

主控台

刪除資料庫叢集中的資料庫執行個體

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中選擇 Databases (資料庫)，然後選擇您要刪除的資料庫執行個體。
3. 對於 Actions (動作)，請選擇 Delete (刪除)。
4. 在方塊中輸入 **`delete me`**。
5. 選擇 刪除。

AWS CLI

若要使用 刪除資料庫執行個體 AWS CLI，請呼叫 delete-db-instance 命令並指定 `--db-instance-identifier`值。

Example

對於 Linux、macOS 或 Unix：

```
aws rds delete-db-instance \
    --db-instance-identifier mydbinstance
```

在 Windows 中：

```
aws rds delete-db-instance ^
    --db-instance-identifier mydbinstance
```

RDS API

若要使用 Amazon RDS API 刪除資料庫執行個體，請呼叫 DeleteDBInstance 操作並指定 `DBInstanceIdentifier` 參數。

> **Note**
>
> 當資料庫執行個體的狀態為 時`deleting`，其 CA 憑證值不會出現在 RDS 主控台或 AWS CLI 命令或 RDS API 操作的輸出中。如需憑證授權機構憑證的詳細資訊，請參閱使用 SSL/TLS 加密與資料庫叢集的連線。

標記 Amazon Aurora 和 Amazon RDS 資源

Amazon RDS 標籤是由您定義的名稱/值組，且其與 Amazon RDS 資源 (例如資料庫執行個體或資料庫快照) 相關聯。此名稱叫做金鑰。您可以選擇性地為索引鍵提供值。

您可以使用 AWS 管理主控台、 AWS CLI或 Amazon RDS API 來新增、列出和刪除 Amazon RDS 資源上的標籤。使用 CLI 或 API 時，請務必提供 RDS 資源的 Amazon Resource Name (ARN) 來使用。如需建構 ARN 的詳細資訊，請參閱 建構 Amazon RDS 的 ARN。

您可以使用標籤將中繼資料新增至 Aurora 和 Amazon RDS 資源。您可以使用標籤來新增關於資料庫執行個體、快照、Aurora 叢集等的標記法。這樣做可以幫助您記錄您的 Aurora 和 Amazon RDS 資源。您也可以將標籤與自動化維護程序一起使用。

尤其，您可以搭配 IAM 政策使用這些標籤。您可以使用它們來管理 Aurora 和 Amazon RDS 資源的存取，以及控制可將哪些動作套用到那些資源。透過將類似標記資源的費用進行分組，您也可以使用這些標籤來追蹤成本。

您可以標記以下 Aurora 和 Amazon RDS 資源：

- 資料庫執行個體
- 資料庫叢集
- Aurora 全域叢集
- 資料庫叢集端點
- 僅供讀取複本
- 資料庫快照
- 資料庫叢集快照
- 預留資料庫執行個體
- 事件訂閱
- 資料庫選項群組
- 資料庫參數群組
- 資料庫叢集參數群組
- 資料庫子網路群組
- RDS Proxy
- RDS Proxy 端點

- 藍/綠部署
- 零 ETL 整合
- 自動備份
- 叢集自動備份

> **Note**
>
> 當您標記資料庫執行個體時， Aurora會自動將這些標籤套用至相關聯的 Performance Insights 資源。目前，您無法使用 標記 RDS Proxies 和 RDS Proxy 端點 AWS 管理主控台。

主題

- 為什麼要使用 Amazon RDS 資源標籤？
- Amazon RDS 資源標籤的運作方式
- 標記 Amazon RDS 資源的最佳實務
- 複製標籤到資料庫叢集快照
- 標記自動備份資源
- 在 Amazon RDS 中新增和刪除標籤
- 教學：使用標籤指定要停止哪些 Aurora 資料庫叢集

為什麼要使用 Amazon RDS 資源標籤？

您可以使用標籤來執行以下作業：

- 依應用程式、專案、部門、環境等項目將 RDS 資源分類。例如，您可以使用標籤索引鍵來定義類別，此標籤值可以是該類別中的項目。您可以建立標籤 `environment=prod`。或者，您可以定義 `project` 的標籤索引鍵及 `Salix` 的標籤值，表示指派給 Salix 專案的 Amazon RDS 資源。
- 自動化資源管理任務。例如，您可以為標記為 `environment=prod` 的執行個體建立維護時段，該時段與標記為 `environment=test` 的執行個體的時段不同。您也可以為標記為 `environment=prod` 的執行個體設定自動資料庫快照。
- 在 IAM 政策中控制對 RDS 資源的存取。您可以使用全域 `aws:ResourceTag/`*`tag-key`* 條件索引鍵來執行此操作。例如，政策可能僅允許 `DBAdmin` 群組中的使用者修改標記為 `environment=prod` 的資料庫執行個體。如需有關使用 IAM 政策管理已標記資源存取的資訊，請

參閱《 AWS Identity and Access Management 使用者指南》中的 Amazon Aurora 的 Identity and access management 和控制對 AWS 資源的存取。

- 根據標籤監控資源。例如，您可以對標記為 `environment=prod` 的資料庫執行個體建立 Amazon CloudWatch 儀表板。
- 將類似的標記資源分類為群組以追蹤成本。例如，如果您使用 `project=Salix` 標記與 Salix 專案相關聯的 RDS 資源，您可以產生成本報告，並將費用配置給此專案。如需詳細資訊，請參閱AWS 帳單如何在 Amazon RDS 中使用標籤。

Amazon RDS 資源標籤的運作方式

AWS 不會將任何語意意義套用至您的標籤。標籤會嚴格解譯為字元字串。

主題

- Amazon RDS 中的標籤集
- Amazon RDS 中的標籤結構
- 符合標記資格的 Amazon RDS 資源
- AWS 帳單如何在 Amazon RDS 中使用標籤

Amazon RDS 中的標籤集

每個 Amazon RDS 資源都有一個稱為標籤集的容器。容器包含指派給資源的所有標籤。資源只有一個標籤集。

標籤集包含 0-50 個標籤。如果您將標籤新增至 RDS 資源，而其擁有與現有標籤相同的金鑰，則新值會覆寫舊值。

Amazon RDS 中的標籤結構

RDS 標籤的結構如下：

標籤鍵

標籤索引鍵是標籤必要的名稱。字串值長度可以是 1 到 128 個 Unicode 字元，不可在前面加上 `aws:` 或 `rds:`。此字串只能包含一組 Unicode 字母、數字、空格、_、.、:、/、=、+、- 和 @。Java regex 為 `"^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)$"`。標籤鍵區分大小寫。因此，索引鍵 `project` 和 `Project` 是不同的。

索引鍵對標籤集是唯一的。例如，標籤集內的鍵組不能有相同的鍵但使用不同的值，例如 `project=Trinity` 和 `project=Xanadu`。

標籤值

標籤值即為選用的標籤字串值。字串值長度可以是 1 到 256 個 Unicode 字元。此字串只能包含一組 Unicode 字母、數字、空格、_、.、:、/、=、+、- 和 @。Java regex 為 "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)$"。標籤值區分大小寫。因此，值 `prod` 和 `Prod` 是不同的。

標籤集內的值不必是唯一的，並且可以是 null。例如，您可以在 `project=Trinity` 和 `cost-center=Trinity` 標籤集中有一個鍵值對。

符合標記資格的 Amazon RDS 資源

您可以標記以下 Amazon RDS 資源：

- 資料庫執行個體
- 資料庫叢集
- 資料庫叢集端點
- 僅供讀取複本
- 資料庫快照
- 資料庫叢集快照
- 預留資料庫執行個體
- 事件訂閱
- 資料庫選項群組
- 資料庫參數群組
- 資料庫叢集參數群組
- 資料庫子網路群組
- RDS Proxy
- RDS Proxy 端點

> **Note**
>
> 目前無法使用 AWS 管理主控台標記 RDS Proxy 和 RDS Proxy 端點。

- 藍/綠部署
- 零 ETL 整合 (預覽)
- 自動備份
- 叢集自動備份

AWS 帳單如何在 Amazon RDS 中使用標籤

使用標籤來整理 AWS 帳單，以反映您自己的成本結構。若要這樣做，請註冊 以取得包含標籤索引鍵值的 AWS 帳戶 帳單。接著，若要查看合併資源的成本，請根據具有相同標籤鍵值的資源來整理您的帳單資訊。例如，您可以使用特定應用程式名稱來標記數個資源，然後整理帳單資訊以查看該應用程式跨數項服務的總成本。如需詳細資訊，請參閱《AWS Billing 》使用者指南中的使用成本分配標籤。

成本分配標籤如何與資料庫叢集快照搭配使用

您可以將標籤新增到資料庫叢集快照。不過，您的帳單不會反映這個分組。若要讓成本分配標籤套用到資料庫叢集快照，必須符合下列條件：

- 標籤必須連接至父系資料庫執行個體。
- 父資料庫執行個體必須存在於 AWS 帳戶 與資料庫叢集快照相同的 中。
- 父資料庫執行個體必須存在於 AWS 區域 與資料庫叢集快照相同的 中。

如果資料庫叢集快照不存在於與父系資料庫執行個體相同的區域中，或父系資料庫執行個體已刪除，則視為孤立。孤立資料庫快照不支援成本分配標籤。孤立快照的成本會彙總在單一未標記的明細項目中。當符合下列條件時，跨帳戶資料庫叢集快照不會被視為孤立：

- 它們與父系資料庫執行個體位於相同的區域。
- 父系資料庫執行個體由來源帳戶擁有。

> **Note**
>
> 如果父系資料庫執行個體由不同帳戶擁有，則成本分配標籤不適用於目的地帳戶中的跨帳戶快照。

標記 Amazon RDS 資源的最佳實務

當您使用標籤時，建議您遵循下列最佳實務：

- 記錄您組織中所有團隊遵循的標籤使用慣例。特別是，確保名稱具有描述性和一致性。例如，將格式 `environment:prod` 標準化，而不是使用 `env:production` 標記一些資源。

  > **Important**
  >
  > 請勿將個人識別資訊 (PII) 或其他機密或敏感資訊儲存在標籤中。

- 自動化標記以確保一致性。例如，您可以使用下列技術：
  - 在 CloudFormation 範本中包含標籤。當您使用範本建立資源時，會自動標記資源。
  - 使用 AWS Lambda 函數定義和套用標籤。
  - 建立 SSM 文件，其中包含將標籤新增至 RDS 資源的步驟。
- 僅在必要時使用標籤。您可以為單一 RDS 資源新增最多 50 個標籤，但最佳實務是避免不必要的標籤擴散和複雜性。
- 定期檢閱標籤的相關性和正確性。視需要移除或修改過時的標籤。
- 請考慮使用 AWS 中的標籤編輯器建立標籤 AWS 管理主控台。您可以使用標籤編輯器，同時將標籤新增至多個支援 AWS 的資源，包括 RDS 資源。如需詳細資訊，請參閱 AWS Resource Groups 使用者指南中的標籤編輯器。

複製標籤到資料庫叢集快照

當您建立或還原資料庫叢集時，您可以指定將叢集的標籤複製到資料庫叢集的快照。複製標籤可確保資料庫快照的中繼資料符合來源資料庫叢集的中繼資料。它也確保資料庫快照的任何存取政策也符合來源資料庫叢集的存取政策。根據預設，不會複製標籤。

您可以指定標籤複製到資料庫快照以用於下列動作：

- 建立資料庫叢集
- 還原資料庫叢集
- 建立僅供讀取複本
- 複製資料庫叢集快照

> **Note**
>
> 在某些情況下，您可能會包含 create-db-snapshot AWS CLI 命令的 `--tags` 參數值。或者，您可以至少提供一個標籤給 CreateDBSnapshot API 作業。在這些情況下，RDS 不會將標籤從

來源資料庫執行個體複製到新的資料庫快照。即使來源資料庫執行個體開啟了 `--copy-tags-to-snapshot` (CopyTagsToSnapshot) 選項，此功能仍適用。
如果採取此方法，您可以從資料庫快照建立資料庫執行個體的複本。這種方法可避免新增不適用於新資料庫執行個體的標籤。您可以使用 命令 （ 或 CreateDBSnapshot RDS API 操作 ） 建立資料庫快照 AWS CLI `create-db-snapshot`。在建立資料庫快照之後，您可以依照本主題稍後所述來新增標籤。

標記自動備份資源

當您將備份保留期值從 0 設定為大於 0 時，就會建立自動備份資源。您可以使用 `--tag-specifications` 參數在建立期間標記執行個體或叢集自動備份資源。

標籤規格參數

支援`--tag-specifications`請求參數APIs （ 例如 create-db-instance、restore-db-instance-from-db-snapshot、create-db-cluster 等 ） 可以在建立期間標記自動備份 （ 資源類型： `auto-backup`或 `cluster-auto-backup`)。

標記叢集自動備份

建立已啟用自動備份的資料庫叢集`ResourceType=cluster-auto-backup`時，請使用 `--tag-specifications`搭配 。

> **Note**
>
> - 自動化備份標籤與來源資料庫執行個體標籤、資料庫叢集標籤或資料庫快照標籤無關。

在 Amazon RDS 中新增和刪除標籤

您可以執行下列動作：

- 建立資源時建立標籤，例如當您執行 AWS CLI 命令 時`create-db-instance`。
- 使用 `add-tags-to-resource` 命令將標籤新增至現有資源。
- 使用命令 `list-tags-for-resource` 列出與特定資源相關聯的標籤。
- 使用命令 `add-tags-to-resource` 更新標籤。

- 使用命令 `remove-tags-from-resource` 從資源移除標籤。

下列程序顯示如何對與資料庫執行個體和 Aurora 資料庫叢集相關的資源，執行典型標記操作。請注意，標籤會進行快取，以供授權使用。因此，當您在 Amazon RDS 資源上新增或更新標籤時，可能需要數分鐘的時間才能使用修改功能。

主控台

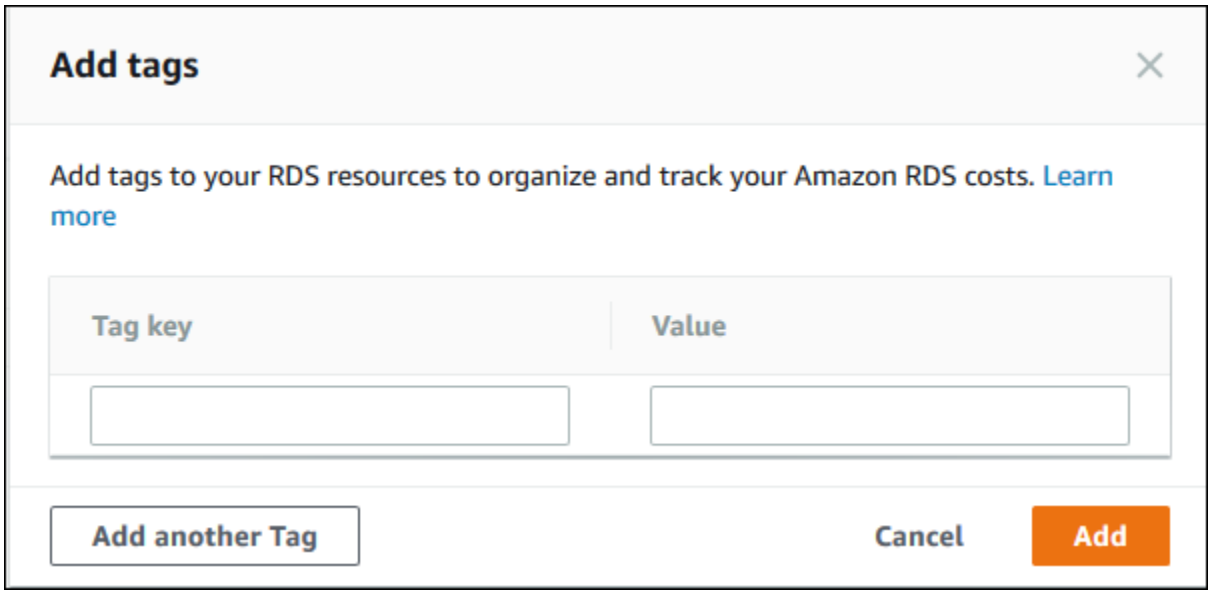
設定 Amazon RDS 資源標籤的程序類似為所有資源設定標籤的程序。以下程序會說明如何為 Amazon RDS 資料庫執行個體新增標籤。

新增標籤至資料庫資訊個體

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。

> **Note**
>
> 若要篩選 Databases (資料庫) 窗格中的資料庫執行個體清單，請在 Filter databases (篩選資料庫) 中輸入文字字串。只有包含此字串的資料庫執行個體會顯示出來。

3. 選擇您要標記的資料庫執行個體名稱，以顯示其詳細資訊。
4. 在詳細內容區段，向下捲動至 Tags (標籤) 區段。
5. 選擇 Add (新增)。此時將會顯示 Add tags (新增標籤) 視窗。



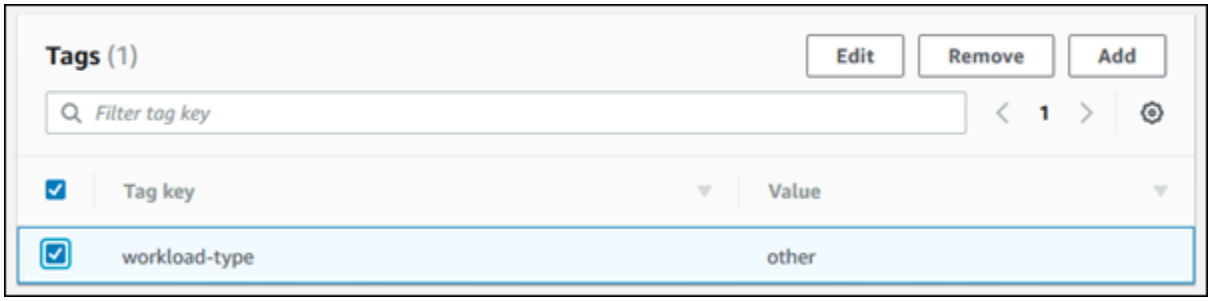
6. 輸入 Tag key (標籤索引鍵) 和 Value (值) 的值。
7. 若要新增另一個標籤，您可以選擇 Add another Tag (新增另一個標籤)，然後輸入其 Tag key (標籤索引鍵) 和 Value (值) 的值。

   視需要重複此步驟。
8. 選擇 Add (新增)。

從資料庫資訊個體刪除標籤

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。

   > **Note**
   >
   > 若要篩選 Databases (資料庫) 窗格中的資料庫執行個體清單，請在 Filter databases (篩選資料庫) 方塊中輸入文字字串。只有包含此字串的資料庫執行個體會顯示出來。
3. 選擇資料庫執行個體的名稱，以顯示其詳細資訊。
4. 在詳細內容區段，向下捲動至 Tags (標籤) 區段。
5. 選擇您要刪除的標籤。



6. 選擇 Delete (刪除)，接著在 Delete tags (刪除標籤) 視窗中選擇 Delete (刪除)。

AWS CLI

您可以使用 AWS CLI新增、列出或移除資料庫執行個體的標籤。

- 若要將一或多個標籤新增至 Amazon RDS 資源，請使用 AWS CLI 命令 `add-tags-to-resource`。
- 若要列出 Amazon RDS 資源上的標籤，請使用 AWS CLI 命令 `list-tags-for-resource`。

- 若要從 Amazon RDS 資源移除一或多個標籤，請使用 AWS CLI 命令 `remove-tags-from-resource`。

若要進一步了解如何建構必要的 ARN，請參閱建構 Amazon RDS 的 ARN。

RDS API

您可以使用 Amazon RDS API 新增、列出或移除資料庫執行個體的標籤。

- 若要為 Amazon RDS 資源新增標籤，請使用 `AddTagsToResource` 操作。
- 若要列出指派給 Amazon RDS 資源的標籤，請使用 `ListTagsForResource`。
- 若要移除 Amazon RDS 資源的標籤，請使用 `RemoveTagsFromResource` 操作。

若要進一步了解如何建構必要的 ARN，請參閱建構 Amazon RDS 的 ARN。

搭配 XML 使用 Amazon RDS API 時，標籤使用以下結構描述：

```
<Tagging>
      <TagSet>
          <Tag>
              <Key>Project</Key>
              <Value>Trinity</Value>
          </Tag>
          <Tag>
              <Key>User</Key>
              <Value>Jones</Value>
          </Tag>
      </TagSet>
  </Tagging>
```

下表列出允許的 XML 標籤及其特性。`Key` 和 `Value` 的值區分大小寫。例如，`project=Trinity` 和 `PROJECT=Trinity` 是兩個不同的標籤。

標記元素	描述
TagSet	標籤集是一個容器，其中存放指派給 Amazon RDS 資源的所有標籤。每個資源只能有一個標籤集。此外，您僅能透過 Amazon RDS API 來使用 TagSet。
標籤	標籤是使用者定義的鍵值組。標籤集內可以有 1 到 50 個標籤。

標記元素	描述
Key	鍵是標籤的必要名稱。對於限制，請參閱 Amazon RDS 中的標籤結構。 字串值長度可以是 1 到 128 個 Unicode 字元，不可在前面加上 aws: 或 rds:。該字串僅能包含一組 Unicode 字母、數字、空格、「_」、「.」、「/」、「=」、「+」、「-」(Java regex："^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$")。 鍵在標籤集內必須是唯一的。舉例來說，標籤集不能擁有索引鍵相同，但值不同的索引鍵組，例如：project/Trinity 和 project/Xanadu。
Value	值是標籤的選用值。對於限制，請參閱 Amazon RDS 中的標籤結構。 字串值長度可以是 1 到 256 個 Unicode 字元，不可在前面加上 aws: 或 rds:。該字串僅能包含一組 Unicode 字母、數字、空格、「_」、「.」、「/」、「=」、「+」、「-」(Java regex："^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$")。 標籤組中的值不必是唯一的，並且可以是 null。例如，在 project/Trinity 及 cost-center/Trinity 標籤集中，均能擁有一個索引鍵/值組。

教學：使用標籤指定要停止哪些 Aurora 資料庫叢集

假設您正在開發或測試環境中建立大量 Aurora 資料庫叢集。您必須保留所有這些叢集數天。有些叢集會在夜間執行測試。其他叢集可以在夜間停止，並在隔天再次啟動。下列範例顯示如何將標籤指派給適合在夜間停止的叢集。然後，該範例會顯示腳本如何檢測哪些叢集具有該標籤，然後停止這些叢集。在這個範例中，鍵值對的值部分並不重要。stoppable 標籤的存在表示叢集具有此使用者定義的屬性。

若要指定要停止的 Aurora 資料庫叢集

1. 確定您要指定為可停止的叢集 ARN。

   用於標記的命令和 API 可以與 ARN 一起使用。如此一來，他們就可以在 AWS 區域、AWS 帳戶以及可能具有相同短名稱的不同類型資源之間順暢運作。您可以在於叢集上操作的 CLI 命令中指定 ARN 而不是叢集 ID。將您自己的叢集名稱替換為 *dev-test-cluster*。在使用 ARN 參數的後續命令中，替換您自己的叢集 ARN。ARN 包含您自己的 AWS 帳戶 ID 和叢集所在的 AWS 區域名稱。

   ```
   $ aws rds describe-db-clusters --db-cluster-identifier dev-test-cluster \
     --query "*[].{DBClusterArn:DBClusterArn}" --output text
   ```

```
arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
```

2. 將標籤 stoppable 新增至此叢集。

   您會選擇此標籤的名稱。這種方法表示您可以避免設計命名慣例，對名稱中的所有相關資訊進行編碼。在這類慣例中，您可能會對資料庫執行個體名稱或其他資源名稱中的資訊進行編碼。由於此範例會將標籤視為存在或不存在的屬性，因此會省略 Value= 參數的 --tags 部分。

```
$ aws rds add-tags-to-resource \
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster \
  --tags Key=stoppable
```

3. 確認標籤存在於叢集中。

   這些命令以 JSON 格式和純定位鍵分隔文字擷取叢集的標籤資訊。

```
$ aws rds list-tags-for-resource \
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
{
    "TagList": [
        {
            "Key": "stoppable",
            "Value": ""

        }
    ]
}
$ aws rds list-tags-for-resource \
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster --output
 text
TAGLIST stoppable
```

4. 若要停止指定為 stoppable 的所有叢集，請準備所有叢集的清單。對清單執行迴圈並檢查每個叢集是否標記了相關屬性。

   此 Linux 範例使用 shell 指令碼將叢集 ARN 清單儲存到暫存檔案，然後為每個叢集執行 CLI 命令。

```
$ aws rds describe-db-clusters --query "*[].[DBClusterArn]" --output text >/tmp/
cluster_arns.lst
$ for arn in $(cat /tmp/cluster_arns.lst)
do
```

```
  match="$(aws rds list-tags-for-resource --resource-name $arn --output text | grep
 'TAGLIST\tstoppable')"
  if [[ ! -z "$match" ]]
  then
      echo "Cluster $arn is tagged as stoppable. Stopping it now."
# Note that you can specify the full ARN value as the parameter instead of the
 short ID 'dev-test-cluster'.
      aws rds stop-db-cluster --db-cluster-identifier $arn
  fi
done

Cluster arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster is tagged as
 stoppable. Stopping it now.
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "us-east-1e",
            "us-east-1c",
            "us-east-1d"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "dev-test-cluster",
        ...
```

您可以在每天結束時執行這樣的指令碼，以確保非必要的叢集已停止。您也可以使用 cron 之類的公用程式來排程要執行的任務，例如每天晚上檢查。例如，如果某些叢集錯誤地保持執行中狀態，您可能會執行此操作。在這裡，您可以微調準備要檢查的叢集清單的命令。

下列命令會產生叢集的清單，但只會產生 available 狀態的叢集清單。指令碼可以忽略已停止的叢集，因為它們會有不同的狀態值，例如 stopped 或 stopping。

```
$ aws rds describe-db-clusters \
  --query '*[].{DBClusterArn:DBClusterArn,Status:Status}|[?Status == `available`]|[].
{DBClusterArn:DBClusterArn}' \
  --output text
arn:aws:rds:us-east-1:123456789:cluster:cluster-2447
arn:aws:rds:us-east-1:123456789:cluster:cluster-3395
arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
arn:aws:rds:us-east-1:123456789:cluster:pg2-cluster
```

> **Tip**
>
> 您可以使用指派標籤，並尋找具有這些標籤的叢集，以其他方式降低成本。例如，採取此案例搭配用於開發和測試的 Aurora 資料庫叢集。在這裡，您可以指定一些要在每天結束時刪除的叢集，或者只刪除讀取器資料庫執行個體。或者，您可以指定一些，在預期的低使用量期間將其資料庫執行個體變更為小型資料庫執行個體類別。

Amazon RDS 中的 Amazon Resource Name (ARN)

在 Amazon Web Services 中建立的資源，都是用 Amazon 資源名稱 (ARN) 做為唯一識別符。當您在執行特定的 Amazon RDS 操作時，必須要指定可識別 Amazon RDS 資源的唯一 ARN。舉例來說，建立 RDS 資料庫執行個體僅供讀取複本時，請務必提供來源資料庫執行個體的 ARN。

如需有關建構 ARN 和取得現有 ARN 的詳細資訊，請參閱下列主題。

主題

- 建構 Amazon RDS 的 ARN
- 取得 Amazon RDS 的現有 ARN

建構 Amazon RDS 的 ARN

在 Amazon Web Services 中建立的資源，都是用 Amazon 資源名稱 (ARN) 做為唯一識別碼。若要建構 Amazon RDS 資源的 ARN，則可使用下列語法。

`arn:aws:rds:<region>:<account number>:<resourcetype>:<name>`

對於全域叢集資源，ARN 不包含 AWS 區域：`arn:aws:rds::<account number>:global-cluster:<name>`。全域叢集ARNs 不會顯示在 中 AWS 管理主控台。

區域名稱	區域	端點	通訊協定	
美國東部 (俄亥俄)	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
		rds-fips.us-east-2.api.aws	HTTPS	
		rds.us-east-2.api.aws	HTTPS	
		rds-fips.us-east-2.amazonaws.com	HTTPS	
美國東部 (維吉尼亞北部)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	
		rds-fips.us-east-1.api.aws	HTTPS	
		rds-fips.us-east-1.amazonaws.com	HTTPS	
		rds.us-east-1.api.aws	HTTPS	

區域名稱	區域	端點	通訊協定
美國西部 (加利佛尼亞北部)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS
		rds.us-west-1.api.aws	HTTPS
		rds-fips.us-west-1.amazonaws.com	HTTPS
		rds-fips.us-west-1.api.aws	HTTPS
美國西部 (奧勒岡)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS
		rds-fips.us-west-2.amazonaws.com	HTTPS
		rds.us-west-2.api.aws	HTTPS
		rds-fips.us-west-2.api.aws	HTTPS
Africa (Cape Town)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS
		rds.af-south-1.api.aws	HTTPS
亞太區域 (香港)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS
		rds.ap-east-1.api.aws	HTTPS
亞太區域 (海德拉巴)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS
		rds.ap-south-2.api.aws	HTTPS
亞太區域 (雅加達)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS
		rds.ap-southeast-3.api.aws	HTTPS
亞太地區 (馬來西亞)	ap-southeast-5	rds.ap-southeast-5.amazonaws.com	HTTPS
亞太區域 (墨爾本)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com	HTTPS
		rds.ap-southeast-4.api.aws	HTTPS

區域名稱	區域	端點	通訊協定
亞太區域 (孟買)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS
		rds.ap-south-1.api.aws	HTTPS
亞太區域 (紐西蘭)	ap-southeast-6	rds.ap-southeast-6.amazonaws.com	HTTPS
亞太區域 (大阪)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS
		rds.ap-northeast-3.api.aws	HTTPS
亞太區域 (首爾)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS
		rds.ap-northeast-2.api.aws	HTTPS
亞太區域 (新加坡)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS
		rds.ap-southeast-1.api.aws	HTTPS
亞太區域 (雪梨)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS
		rds.ap-southeast-2.api.aws	HTTPS
亞太區域 (台北)	ap-east-2	rds.ap-east-2.amazonaws.com	HTTPS
亞太區域 (泰國)	ap-southeast-7	rds.ap-southeast-7.amazonaws.com	HTTPS
亞太區域 (東京)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS
		rds.ap-northeast-1.api.aws	HTTPS

區域名稱	區域	端點	通訊協定
加拿大 (中部)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS
		rds.ca-central-1.api.aws	HTTPS
		rds-fips.ca-central-1.api.aws	HTTPS
		rds-fips.ca-central-1.amazonaws.com	HTTPS
加拿大西部 (卡加利)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS
		rds-fips.ca-west-1.amazonaws.com	HTTPS
歐洲 (法蘭克福)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS
		rds.eu-central-1.api.aws	HTTPS
歐洲 (愛爾蘭)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS
		rds.eu-west-1.api.aws	HTTPS
歐洲 (倫敦)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS
		rds.eu-west-2.api.aws	HTTPS
歐洲 (米蘭)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS
		rds.eu-south-1.api.aws	HTTPS
歐洲 (巴黎)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS
		rds.eu-west-3.api.aws	HTTPS
歐洲 (西班牙)	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS
		rds.eu-south-2.api.aws	HTTPS
歐洲 (斯德哥爾摩)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS
		rds.eu-north-1.api.aws	HTTPS

區域名稱	區域	端點	通訊協定
歐洲 (蘇黎世)	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS
		rds.eu-central-2.api.aws	HTTPS
以色列 (特拉維夫)	il-central-1	rds.il-central-1.amazonaws.com	HTTPS
		rds.il-central-1.api.aws	HTTPS
墨西哥 (中部)	mx-central-1	rds.mx-central-1.amazonaws.com	HTTPS
中東 (巴林)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS
		rds.me-south-1.api.aws	HTTPS
中東 (阿拉伯聯合大公國)	me-central-1	rds.me-central-1.amazonaws.com	HTTPS
		rds.me-central-1.api.aws	HTTPS
南美洲 (聖保羅)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS
		rds.sa-east-1.api.aws	HTTPS
AWS GovCloud （美國東部）	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS
		rds.us-gov-east-1.api.aws	HTTPS
AWS GovCloud （美國西部）	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS
		rds.us-gov-west-1.api.aws	HTTPS

下表顯示您在為特定 Amazon RDS 資源類型建構 ARN 時應使用的格式。

資源類型	ARN 格式
資料庫執行個體	arn:aws:rds:*<region>*:*<account>* :db:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :db:my-mysql-instance-1`
資料庫叢集	arn:aws:rds:*<region>*:*<account>* :cluster:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :cluster: my-aurora-cluster-1`
事件訂閱	arn:aws:rds:*<region>*:*<account>* :es:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :es:my-subscription`
DB parameter group (資料庫參數群組)	arn:aws:rds:*<region>*:*<account>* :pg:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :pg:my-param-enable-logs`
DB cluster parameter group (資料庫叢集參數群組)	arn:aws:rds:*<region>*:*<account>* :cluster-pg:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :cluster-pg: my-cluster-param-timezone`
預留資料庫執行個體	arn:aws:rds:*<region>*:*<account>* :ri:*<name>*

資源類型	ARN 格式
	例如： `arn:aws:rds: us-east-2 :123456789012 :ri:my-reserved-postgresql`
資料庫安全群組	arn:aws:rds:*<region>*:*<account>* :secgrp:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :secgrp:my-public`
自動化的資料庫快照	arn:aws:rds:*<##>* : *<##>* : snapshot:rds : *<##>* 例如： `arn:aws:rds: us-east-2 :123456789012 :snapshot:rds: my-mysql-db-2019-07-22-07-23`
自動化的資料庫叢集快照	arn:aws:rds:*<##>* : *<##>* : cluster-snapshot:rds : *<##>* 例如： `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot:rds:  my-aurora-cluster-2019-07-22-16-16`
手動資料庫快照	arn:aws:rds:*<region>*:*<account>* :snapshot:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :snapshot: my-mysql-db-snap`

資源類型	ARN 格式
手動資料庫叢集快照	arn:aws:rds:*<region>*:*<account>* :cluster-snapshot:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot: my-aurora-cluster-snap`
資料庫子網路群組	arn:aws:rds:*<region>*:*<account>* :subgrp:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :subgrp:my-subnet-10`
全域叢集	arn：aws：rds：：*<account>* ：global-cluster：*<name>* 例如： `arn:aws:rds:: 123456789012 :global-cluster: my-aurora-global-cluster-1`

取得 Amazon RDS 的現有 ARN

您可以使用 AWS 管理主控台、AWS Command Line Interface(AWS CLI) 或 RDS API 來取得 RDS 資源的 ARN。

主控台

若要從 取得 ARNAWS 管理主控台，請導覽至您想要 ARN 的資源，並檢視該資源的詳細資訊。

例如，您可以從資料庫叢集詳細資訊的組態標籤中取得資料庫叢集的 ARN。
資料庫叢集 ARN。

AWS CLI

若要從特定 RDS 資源AWS CLI的 取得 ARN，您可以使用該資源的 `describe`命令。下表顯示每個 AWS CLI命令，以及與命令搭配使用以取得 ARN 的 ARN 屬性。

AWS CLI命令	ARN 屬性
describe-event-subscriptions	EventSubscriptionArn
describe-certificates	CertificateArn
describe-db-parameter-groups	DBParameterGroupArn
describe-db-cluster-parameter-groups	DBClusterParameterGroupArn
describe-db-instances	DBInstanceArn
describe-db-security-groups	DBSecurityGroupArn
describe-db-snapshots	DBSnapshotArn
describe-events	SourceArn
describe-reserved-db-instances	ReservedDBInstanceArn
describe-db-subnet-groups	DBSubnetGroupArn
describe-db-clusters	DBClusterArn
describe-db-cluster-snapshots	DBClusterSnapshotArn

例如，下列AWS CLI命令會取得資料庫執行個體的 ARN。

Example

針對 Linux、macOS 或 Unix：

```
aws rds describe-db-instances \
--db-instance-identifier DBInstanceIdentifier \
--region us-west-2 \
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

在 Windows 中：

```
aws rds describe-db-instances ^
```

```
--db-instance-identifier DBInstanceIdentifier ^
--region us-west-2 ^
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

該命令的輸出與以下內容相似：

```
[
    {
        "DBInstanceArn": "arn:aws:rds:us-west-2:account_id:db:instance_id",
        "DBInstanceIdentifier": "instance_id"
    }
]
```

RDS API

若要取得特定 RDS 資源的 ARN，則可呼叫下列 RDS API 操作，並搭配使用下方所示的 ARN 屬性。

RDS API 操作	ARN 屬性
DescribeEventSubscriptions	EventSubscriptionArn
DescribeCertificates	CertificateArn
DescribeDBParameterGroups	DBParameterGroupArn
DescribeDBClusterParameterGroups	DBClusterParameterGroupArn
DescribeDBInstances	DBInstanceArn
DescribeDBSecurityGroups	DBSecurityGroupArn
DescribeDBSnapshots	DBSnapshotArn
DescribeEvents	SourceArn
DescribeReservedDBInstances	ReservedDBInstanceArn
DescribeDBSubnetGroups	DBSubnetGroupArn
DescribeDBClusters	DBClusterArn

RDS API 操作	ARN 屬性
DescribeDBClusterSnapshots	DBClusterSnapshotArn

Amazon Aurora 更新

Amazon Aurora 版本會定期更新。更新會在系統維護時段套用到 Amazon Aurora 資料庫叢集。套用更新的時間取決於資料庫叢集的區域和維護時段設定，以及更新類型。更新作業需要資料庫重新啟動，因此您通常會經歷 20–30 秒的停機時間。在此停機時間過後，您就可以繼續使用資料庫叢集。您可以從 AWS 管理主控台檢視或變更您的維護時段設定。

> **Note**
>
> 重新啟動資料庫執行個體所需的時間取決於損毀復原程序、重新啟動當時的資料庫活動，以及特定資料庫引擎的行為。若要改善重新啟動時間，建議在重新啟動程序期間盡可能減少資料庫活動。減少資料庫活動，可降低傳輸中交易的轉返活動。

如需 Amazon Aurora 作業系統更新的相關資訊，請參閱 Aurora 資料庫叢集的作業系統更新。

某些更新是專屬於 Aurora 支援的資料庫引擎。如需資料庫引擎更新的詳細資訊，請參閱下表。

資料庫引擎	更新
Amazon Aurora MySQL	請參閱 Amazon Aurora MySQL 的資料庫引擎更新
Amazon Aurora PostgreSQL	請參閱 Amazon Aurora PostgreSQL 的資料庫引擎更新

識別您的 Amazon Aurora 版本

Amazon Aurora 中包含某些對 Aurora 通用的功能，且適用於所有 Aurora 資料庫叢集。Aurora 中亦包含其他專屬於 Aurora 所支援特定資料庫引擎的功能。這些功能僅適用於使用該資料庫引擎的 Aurora 資料庫叢集，例如 Aurora PostgreSQL。

Aurora 資料庫執行個體提供兩個版本號碼：Aurora 版本號碼和 Aurora 資料庫引擎版本號碼。Aurora 版本號碼使用以下格式。

```
<major version>.<minor version>.<patch version>
```

若要從使用特定資料庫引擎的 Aurora 資料庫執行個體取得 Aurora 版本編號，請使用下列其中一個查詢。

資料庫引擎	查詢
Amazon Aurora MySQL	`SELECT AURORA_VERSION();` `SHOW @@aurora_version;`
Amazon Aurora PostgreSQL	`SELECT AURORA_VERSION();`

使用 Amazon Aurora 的 Amazon RDS 延伸支援

RDS 延長支援可讓您在 Aurora 標準支援結束日期過後，繼續在主要引擎版本上執行資料庫，但須另付費用。

只有在第一次建立或還原資料庫執行個體時啟用 RDS 延伸支援，才能在 RDS 延伸支援中註冊資料庫。除非您要還原在現有資料庫執行個體的 RDS 延長支援註冊狀態，否則您無法加以更新。

如果您在建立或還原資料庫執行個體期間啟用 RDS 延伸支援，則在 Aurora 標準支援結束日期之後，Amazon Aurora 會自動在 RDS 延伸支援中註冊資料庫執行個體。自動註冊 RDS 延長支援不會變更資料庫引擎，也不會影響資料庫執行個體的運行時間或效能。

RDS 延伸支援提供下列更新和技術支援：

- 資料庫執行個體或資料庫叢集的關鍵和高 CVE 安全更新，包括資料庫引擎
- 關鍵問題的錯誤修正和修補程式
- 在標準 Amazon RDS 服務水準協議中開立支援案例並接收疑難排解協助的能力

此付費服務可讓您有更多時間升級至支援的主要引擎版本。例如，Aurora MySQL 第 2 版的 Aurora 標準支援結束日期為 2024 年 10 月 31 日。不過，您尚未準備好在該日期之前手動升級至 Aurora MySQL 第 3 版。在此情況下，Amazon Aurora 會在 2024 年 10 月 31 日在 RDS 延伸支援中自動註冊叢集，而且您可以繼續執行 Aurora MySQL 第 2 版。從 2024 年 12 月 1 日開始，Amazon Aurora 會自動向您收取 RDS 延長支援的費用。

可在主要引擎版本的社群生命週期結束日期後提供最多 3 年的 RDS 延長支援 (Aurora MySQL 第 2 版為 3 年 4 個月)。此時間後，如果您尚未將主要引擎版本升級至支援的版本，則 Amazon Aurora 將自動升級主要引擎版本。我們建議您盡快升級至支援的主要引擎版本。

如需 Aurora 標準支援結束日期和 RDS 延長支援結束日期的詳細資訊，請參閱 Aurora MySQL 主要版本的發佈行事曆和 Aurora PostgreSQL 主要版本的發佈行事曆。

主題

- Amazon RDS 延長支援概觀
- Amazon RDS 延長支援費用
- 具有 Amazon RDS 延伸支援的版本
- 使用 Amazon RDS 延伸支援時 Amazon Aurora 和客戶責任

- 使用 Amazon RDS 延伸支援，建立Aurora 資料庫叢集或全域叢集
- 檢視在 Amazon RDS 延伸支援中Aurora 資料庫叢集或全域叢集的註冊
- 在 Amazon RDS 延伸支援中檢視引擎版本的支援日期
- 使用 Amazon RDS 延伸支援還原Aurora 資料庫叢集或全域叢集

Amazon RDS 延長支援概觀

在 Aurora 標準支援結束日期之後，如果您未在建立或還原資料庫執行個體期間停用 RDS 延伸支援，則 Amazon Aurora 會自動在 RDS 延伸支援中進行註冊。如果您尚未執行該版本，Aurora 會自動將資料庫執行個體升級至 Aurora 標準支援結束日期前發佈的最後一個次要版本。Amazon Aurora 不會在主要引擎版本的 Aurora 標準支援結束日期之前升級次要版本。

您可以使用已達到 Aurora 標準支援結束日期的主要引擎版本建立新資料庫。Aurora 會自動在 RDS 延伸支援中註冊這些新資料庫，並向您收取此方案的費用。

如果您在 Aurora 標準支援結束日期之前升級到仍在 Aurora 標準支援的引擎，Amazon Aurora 不會在 RDS 延伸支援中註冊引擎。

如果您嘗試還原與超過 Aurora 標準支援結束日期，但未註冊 RDS 延伸支援之引擎相容的資料庫快照，則 Amazon Aurora 會嘗試升級快照，使其與仍在 Aurora 標準支援下的最新引擎版本相容。如果還原失敗，Amazon Aurora 會使用與快照相容的版本，自動在 RDS 延伸支援中註冊引擎。

您可以隨時在 RDS 延長支援中結束註冊。若要結束註冊，請將每個註冊的引擎升級至仍在 Aurora 標準支援的較新引擎版本。RDS 延長支援註冊的結束將在您完成升級至較新引擎版本的那一天生效，該版本仍在 Aurora 標準支援下。

如需 Aurora 標準支援結束日期和 RDS 延長支援結束日期的詳細資訊，請參閱 Aurora MySQL 主要版本的發佈行事曆和 Aurora PostgreSQL 主要版本的發佈行事曆。

Amazon RDS 延長支援費用

從 Aurora 標準支援結束日期的隔天開始，註冊 RDS 延長支援的所有引擎都會產生費用。如需了解 Aurora 標準支援結束日期，請參閱 Amazon Aurora 主要版本。

當您採取下列其中一個動作時，RDS 延伸支援的額外收費會自動停止：

- 升級至標準支援涵蓋的引擎版本。
- 刪除在 Aurora 標準支援日期結束後執行主要版本的資料庫。

如果目標引擎版本在未來進入 RDS 延長支援，則會重新開始收費。

例如，Aurora PostgreSQL 11 在 2024 年 3 月 1 日進入延伸支援，但在 2024 年 4 月 1 日之前都不會產生費用。您可以在 2024 年 4 月 30 日將 Aurora PostgreSQL 11 資料庫升級至 Aurora PostgreSQL 12。您只需支付 Aurora PostgreSQL 11 的 30 天延長支援費用。在 RDS 標準支援結束日期 2025 年 2 月 28 日過後，您仍繼續在此資料庫執行個體執行 Aurora PostgreSQL 12。自 2025 年 3 月 1 日起，資料庫將再次產生 RDS 延長支援費用。

如需詳細資訊，請參閱 Amazon Aurora 定價。

避免 Amazon RDS 延長支援的費用

您可以透過防止 Aurora 在 Aurora 標準支援結束日期之後建立或還原Aurora 資料庫叢集或全域叢集，來避免 RDS 延伸支援的費用。若要這樣做，請使用 AWS CLI 或 RDS API。

在 中 AWS CLI，`open-source-rds-extended-support-disabled`為 `--engine-lifecycle-support`選項指定 。在 RDS API 中，為 `LifeCycleSupport` 參數指定 `open-source-rds-extended-support-disabled`。如需詳細資訊，請參閱 建立Aurora 資料庫叢集或全域叢集 或 還原Aurora 資料庫叢集或全域叢集 。

具有 Amazon RDS 延伸支援的版本

RDS 延長支援適用於 Aurora MySQL 以及 Aurora PostgreSQL。如需更多詳細資訊，請參閱 Amazon Aurora 主要版本。

RDS 延長支援僅在特定次要版本上提供。僅在主要版本達到社群生命週期結束日期後，才會將次要版本標示為支援 RDS 延伸支援。如需詳細資訊，請參閱《Aurora MySQL 版本備註》中的 Aurora MySQL 發佈行事曆和《Aurora PostgreSQL 版本備註》中的 Aurora PostgreSQL 發佈行事曆。

僅在 Aurora Serverless v2 上提供 RDS 延長支援。其不適用於 Aurora Serverless v1。

您也可以使用 AWS CLI 或 RDS API 來檢視引擎版本支援日期的相關資訊。如需更多詳細資訊，請參閱 在 Amazon RDS 延伸支援中檢視引擎版本的支援日期。

使用 Amazon RDS 延伸支援時 Amazon Aurora 和客戶責任

下列內容說明在使用 RDS 延伸支援時，Amazon Aurora 的責任以及您的責任。

主題

- Amazon Aurora 責任
- 您的責任

Amazon Aurora 責任

在 Aurora 標準支援結束日期之後，Amazon Aurora 將為在 RDS 延伸支援中註冊的引擎提供修補程式、錯誤修正和升級。此支援的提供時間長達 3 年，或直到您停止使用引擎，以先發生者為準。

將根據國家漏洞資料庫 (NVD) CVSS 嚴重性評級所定義的嚴重和高 CVE 提供修補程式。如需詳細資訊，請參閱漏洞指標。

您的責任

您有責任套用針對 RDS 延伸支援中註冊的Aurora 資料庫叢集或全域叢集套用修補程式、錯誤修正和升級。Amazon Aurora 保留隨時變更、取代或撤銷此類修補程式、錯誤修正和升級的權利。如果需要修補程式來解決安全或關鍵穩定性問題，Amazon Aurora 保留使用修補程式更新Aurora 資料庫叢集或全域叢集，或要求您安裝修補程式的權利。

您也需負責在 RDS 延長支援結束日期之前，將引擎升級至較新的引擎版本。RDS 延長支援結束日期通常是 社群生命週期結束後 3 年。。如需資料庫主要引擎版本的 RDS 延長支援結束日期，請參閱 Amazon Aurora 主要版本。

如果您未升級引擎，則在 RDS 延長支援結束日期之後，Amazon Aurora 會嘗試將引擎升級至 Aurora 標準支援下支援的較新引擎版本。如果升級失敗，Amazon Aurora 保留刪除在 Aurora 標準支援日期後執行引擎的 Aurora 資料庫叢集或全域叢集的權利。不過，在這樣做之前，Amazon Aurora 會保留該引擎中的資料。

使用 Amazon RDS 延伸支援，建立Aurora 資料庫叢集或全域叢集

當您Aurora 資料庫叢集或全域叢集時，請在 主控台中選取啟用 RDS 延伸支援，或使用 中的延伸支援選項 AWS CLI 或 RDS API 中的 參數。當您在 Amazon RDS 延伸支援中註冊Aurora 資料庫叢集或全域叢集時，其會在Aurora 資料庫叢集或全域叢集的生命週期內永久註冊 RDS 延伸支援。

如果您使用主控台，則必須選取啟用 RDS 延伸支援。預設不會選取此設定。

如果您使用 AWS CLI 或 RDS API，但未指定 RDS 延伸支援設定，Amazon RDS 會預設為啟用 RDS 延伸支援。當您使用 CloudFormation 或其他服務進行自動化時，此預設行為會在 Aurora 標準支援結束日期後維持資料庫的可用性。

您可以使用 AWS CLI 或 RDS API 來建立Aurora 資料庫叢集或全域叢集，以防止註冊 RDS 延伸支援。

主題

- RDS 延伸支援行為
- RDS 延伸支援的考量事項
- 使用 RDS 延伸支援建立Aurora 資料庫叢集或全域叢集

RDS 延伸支援行為

下表摘要說明當主要引擎版本達到 Aurora 標準支援結束時會發生的情況。

RDS 延長支援狀態*	Behavior (行為)
已啟用	Amazon RDS 會向您收取 RDS 延伸支援的費用。
Disabled	Amazon RDS 會將 Aurora 資料庫叢集或全域叢集升級至支援的引擎版本。此升級會在 Aurora 標準支援結束日期當天或稍後進行。

* 在 RDS 主控台中，RDS 延長支援狀態顯示為是或否。在 AWS CLI 或 RDS API 中，RDS 延長支援狀態顯示為 `open-source-rds-extended-support`或 `open-source-rds-extended-support-disabled`。

RDS 延伸支援的考量事項

在建立Aurora 資料庫叢集或全域叢集前，請考慮下列項目：

- 在 Aurora 標準支援結束日期過後，您可以防止建立新的 Aurora 資料庫叢集或新的全域叢集，並避免 RDS 延長支援費用。若要這樣做，請使用 AWS CLI 或 RDS API。在 中 AWS CLI，`open-source-rds-extended-support-disabled`為 `--engine-lifecycle-support`選項指定 。在 RDS API 中，為 `LifeCycleSupport` 參數指定 `open-source-rds-extended-support-disabled`。如果您指定 `open-source-rds-extended-support-disabled` 且 Aurora 標準支援結束日期已過，則建立 Aurora 資料庫叢集或全域叢集一律會失敗。
- RDS 延伸支援是在叢集層級設定。在 RDS 主控台、 `--engine-lifecycle-support` 和 `EngineLifecycleSupport` RDS API 中 AWS CLI，叢集的成員一律會有相同的 RDS 延伸支援設定。

如需詳細資訊，請參閱Amazon Aurora 版本。

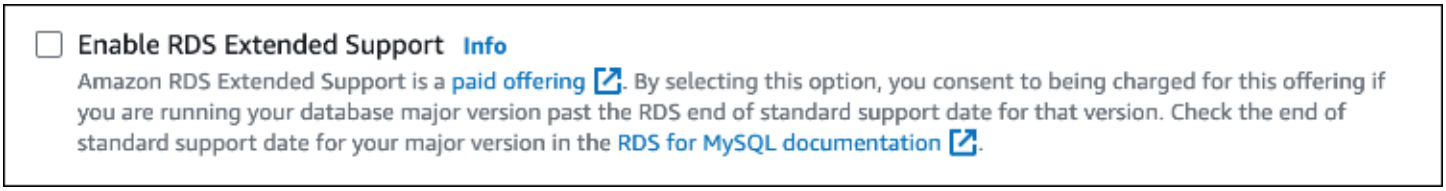
使用 RDS 延伸支援建立Aurora 資料庫叢集或全域叢集

您可以使用 Aurora 資料庫叢集或全域叢集。 AWS 管理主控台 AWS CLI

主控台

當您建立 Aurora 資料庫叢集或全域叢集，請在引擎選項區段中，選取啟用 RDS 延伸支援。預設不會選取此設定。

下圖顯示啟用 RDS 延伸支援設定：



AWS CLI

當您執行 create-db-cluster 或 create-global-cluster AWS CLI 命令時，`open-source-rds-extended-support`請為 `--engine-lifecycle-support`選項指定 ，以選取 RDS 延伸支援。根據預設，此選項會設為 `open-source-rds-extended-support`。

若要防止在 Aurora 標準支援日期結束後建立新的新的 Aurora 資料庫叢集或全域叢集，請為 `--engine-lifecycle-support` 選項指定 `open-source-rds-extended-support-disabled`。這樣做可以避免任何相關聯的 RDS 延長支援費用。

RDS API

當您使用 CreateDBCluster 或 CreateGlobalCluster Amazon RDS API 操作時，請將 `EngineLifecycleSupport` 參數設定為 `open-source-rds-extended-support`，即可選取 RDS 延伸支援。根據預設，此參數會設定為 `open-source-rds-extended-support`。

若要防止在 Aurora 標準支援日期結束後建立新的 Aurora 資料庫叢集或全域叢集，請為 `EngineLifecycleSupport` 參數指定 `open-source-rds-extended-support-disabled`。這樣做可以避免任何相關聯的 RDS 延長支援費用。

如需詳細資訊，請參閱下列主題：

- 若要建立 Aurora 資料庫叢集，請遵循建立 Amazon Aurora 資料庫叢集中的資料庫引擎指示。

- 若要建立全域叢集，請遵循建立 Amazon Aurora 全域資料庫中的資料庫引擎指示。

檢視在 Amazon RDS 延伸支援中Aurora 資料庫叢集或全域叢集的註冊

您可以使用 AWS 管理主控台、 Aurora 資料庫叢集或全域叢集的註冊。 AWS CLI

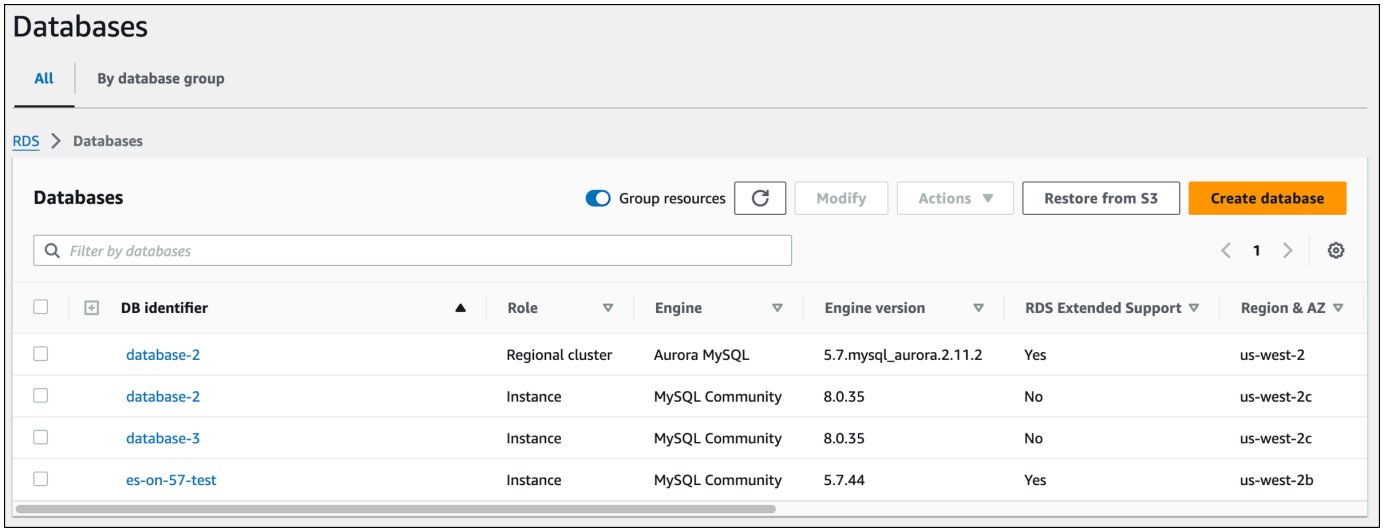
> **Note**
>
> 主控台中的 RDS 延伸支援欄、 中的 `--engine-lifecycle-support` 選項 AWS CLI，以及 RDS API 中的 `EngineLifecycleSupport` 參數，只會指出 RDS 延伸支援中的註冊。只有在資料庫引擎版本達到 Aurora 標準支援結束時，才會開始收取 RDS 延伸支援的費用。如需詳細資訊，請參閱Amazon Aurora 主要版本。
> 例如，Aurora PostgreSQL 11 資料庫已註冊 RDS 延伸支援。在 2024 年 4 月 1 日，Amazon RDS 開始向您收取此資料庫的 RDS 延伸支援費用。在 2024 年 7 月 31 日，您已將此資料庫升級至 Aurora PostgreSQL 12。此資料庫的 RDS 延伸支援狀態會保持啟用狀態。不過，此資料庫的 RDS 延長支援費用已停止，因為 Aurora PostgreSQL 12 尚未達到 Aurora 結束標準支援。在 2025 年 3 月 1 日之前 (也就是 Aurora PostgreSQL 12 的 Aurora 標準支援結束時間)，Amazon RDS 不會向您收取此資料庫的 RDS 延長支援費用。

主控台

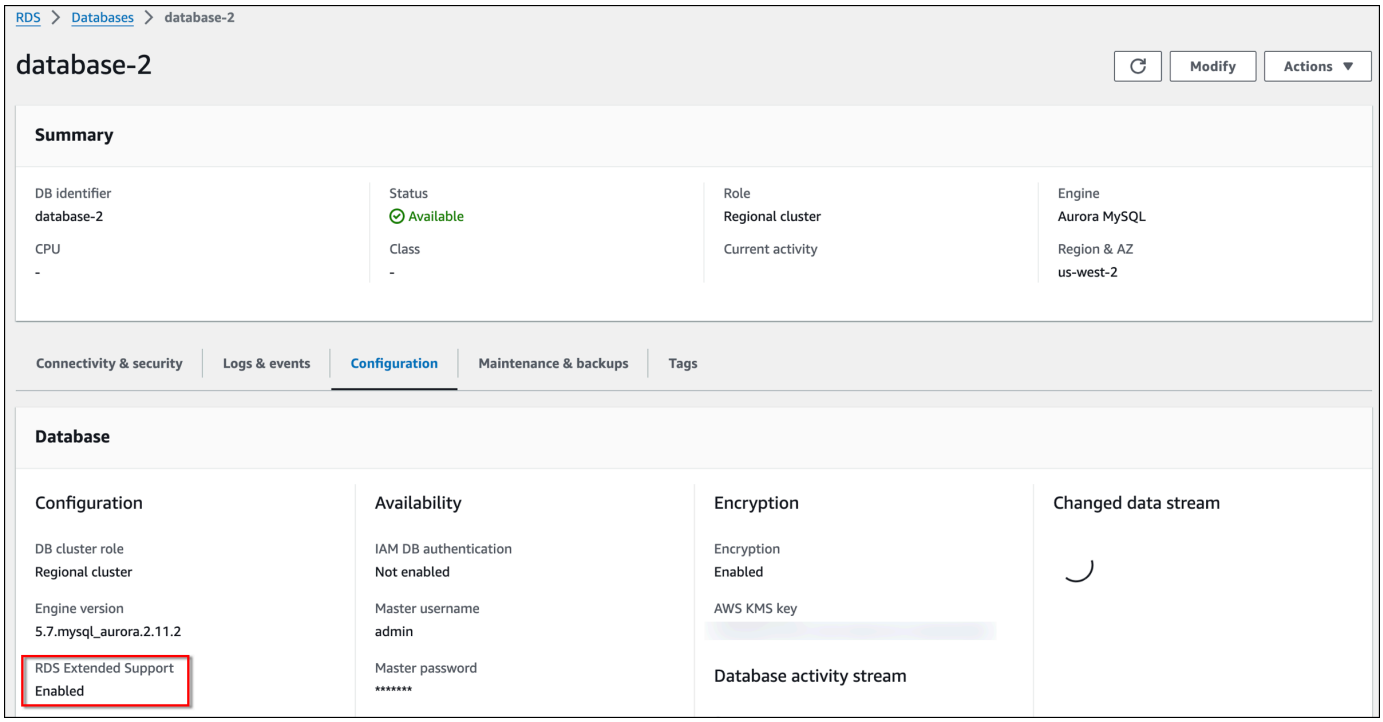
檢視在 RDS 延伸支援中Aurora 資料庫叢集或全域叢集的註冊

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。RDS 延伸支援下的值指出Aurora 資料庫叢集或全域叢集是否已註冊 RDS 延伸支援。如果沒有顯示任何值，則資料庫無法使用 RDS 延伸支援。

   > **Tip**
   >
   > 如果 RDS 延長支援欄未顯示，請選擇偏好設定圖示，然後開啟 RDS 延長支援。



3. 您也可以在每個資料庫的組態索引標籤上檢視註冊。在資料庫識別符下選擇資料庫。在組態索引標籤查看延伸支援下，確認資料庫是否已註冊。



AWS CLI

若要使用 在 RDS 延伸支援中檢視資料庫的註冊 AWS CLI，請執行 describe-db-clusters 或 describe-global-clusters命令。

如果 RDS 延伸支援可供資料庫使用，則回應會包含參數 `EngineLifecycleSupport`。值 `open-source-rds-extended-support` 表示Aurora 資料庫叢集或全域叢集已註冊 RDS 延伸支援。值 `open-source-rds-extended-support-disabled` 表示在 RDS 延伸支援中的Aurora 資料庫叢集或全域叢集註冊已停用。

範例

下列命令會傳回所有 Aurora 資料庫叢集的資訊：

```
aws rds describe-db-clusters
```

下列回應顯示在 Aurora 資料庫叢集 `database-1` 執行的 Aurora PostgreSQL 引擎已註冊 RDS 延伸支援：

```
{
    "DBClusterIdentifier": "database-1",
    ...
    "Engine": "aurora-postgresql",
    ...
    "EngineLifecycleSupport": "open-source-rds-extended-support"
}
```

RDS API

若要使用 Amazon RDS API 在 RDS 延伸支援中檢視資料庫的註冊，請使用 DescribeDBClusters 或 DescribeGlobalClusters 操作。

如果 RDS 延伸支援可供資料庫使用，則回應會包含參數 `EngineLifecycleSupport`。值 `open-source-rds-extended-support` 表示Aurora 資料庫叢集或全域叢集已註冊 RDS 延伸支援。值 `open-source-rds-extended-support-disabled` 表示在 RDS 延伸支援中的Aurora 資料庫叢集或全域叢集註冊已停用。

在 Amazon RDS 延伸支援中檢視引擎版本的支援日期

您可以使用 的引擎版本支援日期的資訊Aurora 資料庫叢集或全域叢集。 AWS CLI 此資訊可協助您規劃升級。

AWS CLI 命令和 RDS API 操作會傳回 Aurora標準支援和 RDS 延伸支援的開始和結束日期。您也可以在主要引擎版本資料表中找到這些日期。如需詳細資訊，請參閱 Amazon Aurora 主要版本。

AWS CLI

若要使用 檢視 Aurora標準支援的開始和結束日期，以及主要引擎版本的 RDS 延伸支援 AWS CLI，請執行 describe-db-major-engine-versions 命令。

此命令會傳回下列相關參數：

- `SupportedEngineLifecycles`：此參數是包含 `LifecycleSupportName`、`LifecycleSupportStartDate` 和 `LifecycleSupportEndDate` 的物件陣列。
- `LifecycleSupportName`：此參數表示引擎版本所在的支援類型：Aurora 標準支援 (`open-source-rds-standard-support`) 或 RDS 延伸支援 (`open-source-rds-extended-support`)。
- `LifecycleSupportStartDate`：根據 `LifecycleSupportName` 的值，此參數會列出主要引擎版本的 Aurora 標準支援或 RDS 延伸支援的開始日期。
- `LifecycleSupportEndDate`：根據 `LifecycleSupportName` 的值，此參數會列出主要引擎版本的 Aurora 標準支援或 RDS 延伸支援的結束日期。

範例

回應範例顯示 Aurora MySQL 第 2 版 (MySQL 5.7) 支援的引擎生命週期 `open-source-rds-standard-support` 以及 `open-source-rds-extended-support` 的開始和結束日期。RDS 延伸支援適用於 Aurora MySQL 第 2 版 (MySQL 5.7)。

```
{
    "DBMajorEngineVersions": [
        {
            "Engine": "aurora-mysql",
            "MajorEngineVersion": "5.7",
            "SupportedEngineLifecycles": [
                {
                    "LifecycleSupportName": "open-source-rds-standard-support",
                    "LifecycleSupportStartDate": "2018-02-06T00:00:00+00:00",
                    "LifecycleSupportEndDate": "2024-10-31T23:59:59.999000+00:00"
                },
                {
                    "LifecycleSupportName": "open-source-rds-extended-support",
                    "LifecycleSupportStartDate": "2024-11-01T00:00:00+00:00",
                    "LifecycleSupportEndDate": "2027-02-28T23:59:59.999000+00:00"
```

```
                }
            ]
        },
        ...
    ]
}
```

RDS API

若要使用 RDS API 檢視主要引擎版本的 Aurora 標準支援以及 RDS 延伸支援的開始和結束日期，請使用 DescribeDBMajorEngineVersions 操作。

如果 RDS 延伸支援適用於引擎版本，則回應會包含參數 `SupportedEngineLifeCycles` 作為具有兩個物件的陣列。一個物件包含 Aurora 標準支援的開始和結束日期。第二個物件包含 RDS 延伸支援的開始和結束日期。

如果 RDS 延伸支援不適用於引擎版本，則回應只會包含參數 `SupportedEngineLifeCycles` 作為具有單一物件的陣列。此物件包含 Aurora 標準支援的開始和結束日期。

使用 Amazon RDS 延伸支援還原Aurora 資料庫叢集或全域叢集

當您還原Aurora 資料庫叢集或全域叢集時，請選取主控台中的啟用 RDS 延伸支援，或使用 中的延伸支援選項 AWS CLI 或 RDS API 中的 參數。當您在 RDS 延伸支援中註冊Aurora 資料庫叢集或全域叢集時，其會在Aurora 資料庫叢集或全域叢集的生命週期內永久註冊 RDS 延伸支援。

RDS 延伸支援設定的預設值取決於您使用主控台 AWS CLI、 或 RDS API 還原資料庫。如果您使用主控台，則不會選取啟用 RDS 延伸支援，而且您要還原的主要引擎版本已超過 Aurora 標準支援結束時間，則 Amazon Aurora 會自動將資料庫執行個體升級至較新的引擎版本。如果您使用 AWS CLI 或 RDS API，但未指定 RDS 延伸支援設定，則 Amazon RDS 預設為啟用 RDS 延伸支援。當您使用 CloudFormation 或其他服務進行自動化時，此預設行為會在 Aurora 標準支援結束日期後維持資料庫的可用性。您可以使用 AWS CLI 或 RDS API 來停用 RDS 延伸支援。

主題

- RDS 延伸支援行為
- RDS 延伸支援的考量事項
- 使用 RDS 延伸支援還原Aurora 資料庫叢集資料庫叢集或全域叢集

RDS 延伸支援行為

下表摘要說明當Aurora 資料庫叢集或您要還原的全域叢集的主要引擎版本已達到 Aurora 標準支援結束時會發生的情況。

RDS 延長支援狀態*	Behavior (行為)
已啟用	Amazon RDS 會向您收取 RDS 延伸支援的費用。
已停用	還原完成後，Amazon RDS 會自動將Aurora 資料庫叢集或全域叢集升級至較新的引擎版本 (在未來的維護時段)。

* 在 RDS 主控台中，RDS 延長支援狀態顯示為是或否。在 AWS CLI 或 RDS API 中，RDS 延長支援狀態顯示為 `open-source-rds-extended-support`或 `open-source-rds-extended-support-disabled`。

RDS 延伸支援的考量事項

在還原Aurora 資料庫叢集或全域叢集前，請考慮下列項目：

- 在 Aurora標準支援結束日期之後，如果您想要從 Amazon S3 Aurora 資料庫叢集或全域叢集，您只能使用 AWS CLI 或 RDS API 來執行此操作。使用 restore-db-cluster-from-s3 AWS CLI command 中的 `--engine-lifecycle-support`選項，或 RestoreDBClusterFromS3 RDS API 操作中的 `EngineLifecycleSupport` 參數。
- 如果您想要防止 Aurora將資料庫還原至 RDS 延伸支援版本，請在 AWS CLI 或 RDS API `open-source-rds-extended-support-disabled`中指定 。這樣做可以避免任何相關聯的 RDS 延長支援費用。

  如果您指定此設定，Amazon Aurora 會自動將還原的資料庫升級至較新的支援主要版本。如果升級無法預先升級檢查，Amazon Aurora 將安全地復原至 RDS 延長支援引擎版本。此資料庫將保持 RDS 延長支援模式，Amazon Aurora 會向您收取 RDS 延長支援的費用，直到您手動升級資料庫為止。
- RDS 延伸支援是在叢集層級設定。在 RDS 主控台、 `--engine-lifecycle-support` 和 `EngineLifecycleSupport` RDS API 中 AWS CLI，叢集的成員一律會有相同的 RDS 延伸支援設定。

如需詳細資訊，請參閱Amazon Aurora 版本。

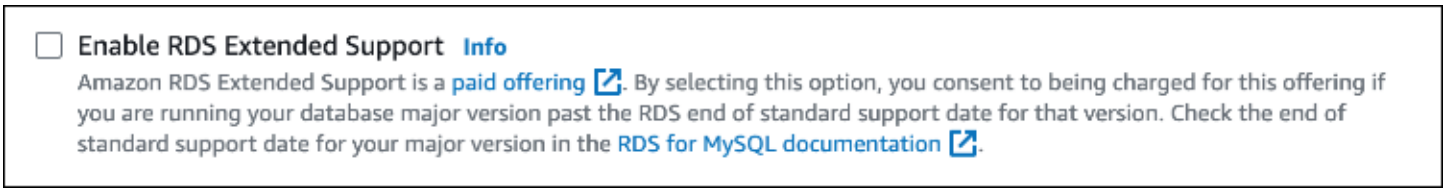
使用 RDS 延伸支援還原Aurora 資料庫叢集資料庫叢集或全域叢集

您可以使用 Aurora 資料庫叢集或具有 RDS 延伸支援版本的全域叢集。 AWS 管理主控台 AWS CLI

主控台

當您還原 Aurora 資料庫叢集或全域叢集，請在引擎選項區段中選取啟用 RDS 延伸支援。如果您未選取此設定，且您要還原的主要引擎版本已超過 Aurora 標準支援結束時間，則 Amazon Aurora 會自動將 Aurora 資料庫叢集或全域叢集升級至 Aurora 標準支援下的版本。

下圖顯示啟用 RDS 延伸支援設定：



AWS CLI

當您執行 restore-db-cluster-from-snapshot AWS CLI 命令時，請`open-source-rds-extended-support`針對 `--engine-lifecycle-support`選項指定 RDS 延伸支援。

如果您想要避免與 RDS 延長支援相關的費用，請將 `--engine-lifecycle-support` 選項設為 `open-source-rds-extended-support-disabled`。根據預設，此選項會設為 `open-source-rds-extended-support`。

您也可以使用下列 AWS CLI 命令指定此值：

- restore-db-cluster-from-s3
- restore-db-cluster-to-point-in-time

RDS API

當您使用 RestoreDBClusterFromSnapshot Amazon RDS API 操作時，請將 `EngineLifecycleSupport` 參數設定為 `open-source-rds-extended-support`，來選取 RDS 延伸支援。

如果您想要避免與 RDS 延長支援相關的費用，請將 `EngineLifecycleSupport` 參數設為 `open-source-rds-extended-support-disabled`。根據預設，此參數會設定為 `open-source-rds-extended-support`。

您也可以使用下列 RDS API 操作來指定此值：

- RestoreDBClusterFromS3
- RestoreDBClusterToPointInTime

如需還原 Aurora 資料庫叢集的詳細資訊，請遵循備份與還原 Amazon Aurora 資料庫叢集中資料庫引擎的指示。

使用 Amazon Aurora 藍/綠部署進行資料庫更新

藍/綠部署會將生產資料庫環境複製到個別已同步的預備環境。透過使用 Amazon Aurora 藍/綠部署，您可以在預備環境中對資料庫進行變更，而不會影響生產環境。例如，您可以升級主要或次要資料庫引擎版本，或在預備環境中變更資料庫參數。備妥後，您可以將預備環境提升為新的生產資料庫環境，停機時間通常不到一分鐘。

Amazon Aurora 在生產環境中複製基礎 Aurora 儲存磁碟區，進而建立預備環境。預備環境中的叢集磁碟區只會存放對該環境所做的增量變更。

> **Note**
>
> 目前，Aurora MySQL、Aurora PostgreSQL 和 Aurora Global Database。如需了解 Amazon RDS 引擎可用性，請參閱《Amazon RDS 使用者指南》中的使用 Amazon RDS 藍/綠部署進行資料庫更新。

主題

- Amazon Aurora 藍/綠部署概觀
- 在 Amazon Aurora 中建立藍/綠部署
- 在 Amazon Aurora 中檢視藍/綠部署
- 在 Amazon Aurora 中切換藍/綠部署
- 刪除 Amazon Aurora 中的藍/綠部署

Amazon Aurora 藍/綠部署概觀

透過使用 Amazon Aurora 藍/綠部署，您可以進行並測試資料庫變更，然後在生產環境中實作這些變更。藍/綠部署會建立一個複製生產環境的預備環境。在藍/綠部署中，藍色環境是目前的生產環境。綠色環境是預備環境，並與目前生產環境保持同步。

您可以對綠色環境中的 Aurora 資料庫叢集進行變更，而不會影響生產工作負載。例如，您可以升級主要或次要資料庫引擎版本或在預備環境中變更資料庫參數。您可以在綠色環境中徹底測試變更。備妥後，您可以切換環境，將綠色環境轉換為新的生產環境。切換通常只需不到一分鐘的時間，不會遺失資料，也不需要變更應用程式。

因為綠色環境是生產環境拓撲的副本，所以資料庫叢集及其所有資料庫執行個體都會在部署中複製。綠色環境也包含資料庫叢集所使用的功能，例如資料庫叢集快照、Performance Insights、增強型監控和 Aurora Serverless v2。

> **Note**
>
> Aurora MySQL、Aurora PostgreSQL 和 Aurora Global Database 支援藍/綠部署。如需 Amazon RDS 可用性，請參閱《Amazon RDS 使用者指南》中的 Amazon RDS 藍/綠部署概觀。

主題

- 區域和版本可用性
- 使用 Amazon RDS 藍/綠部署的優點
- 藍/綠部署的工作流程
- 授予 Amazon Aurora 藍/綠部署操作的存取權
- Amazon Aurora 藍/綠部署的限制和考量事項
- Amazon Aurora 藍/綠部署的最佳實務

區域和版本可用性

功能可用性和支援會因每個資料庫引擎的特定版本以及 AWS 區域而有所不同。如需詳細資訊，請參閱the section called “藍/綠部署”。

使用 Amazon RDS 藍/綠部署的優點

透過使用 Amazon RDS 藍/綠部署，您可以隨時掌握最新的安全修補程式、改善資料庫效能，以及採用較新的資料庫功能，其停機時間短暫且可預測。藍/綠部署可減少資料庫更新 (例如主要或次要引擎版本升級) 的風險和停機時間。

藍/綠部署提供下列優點：

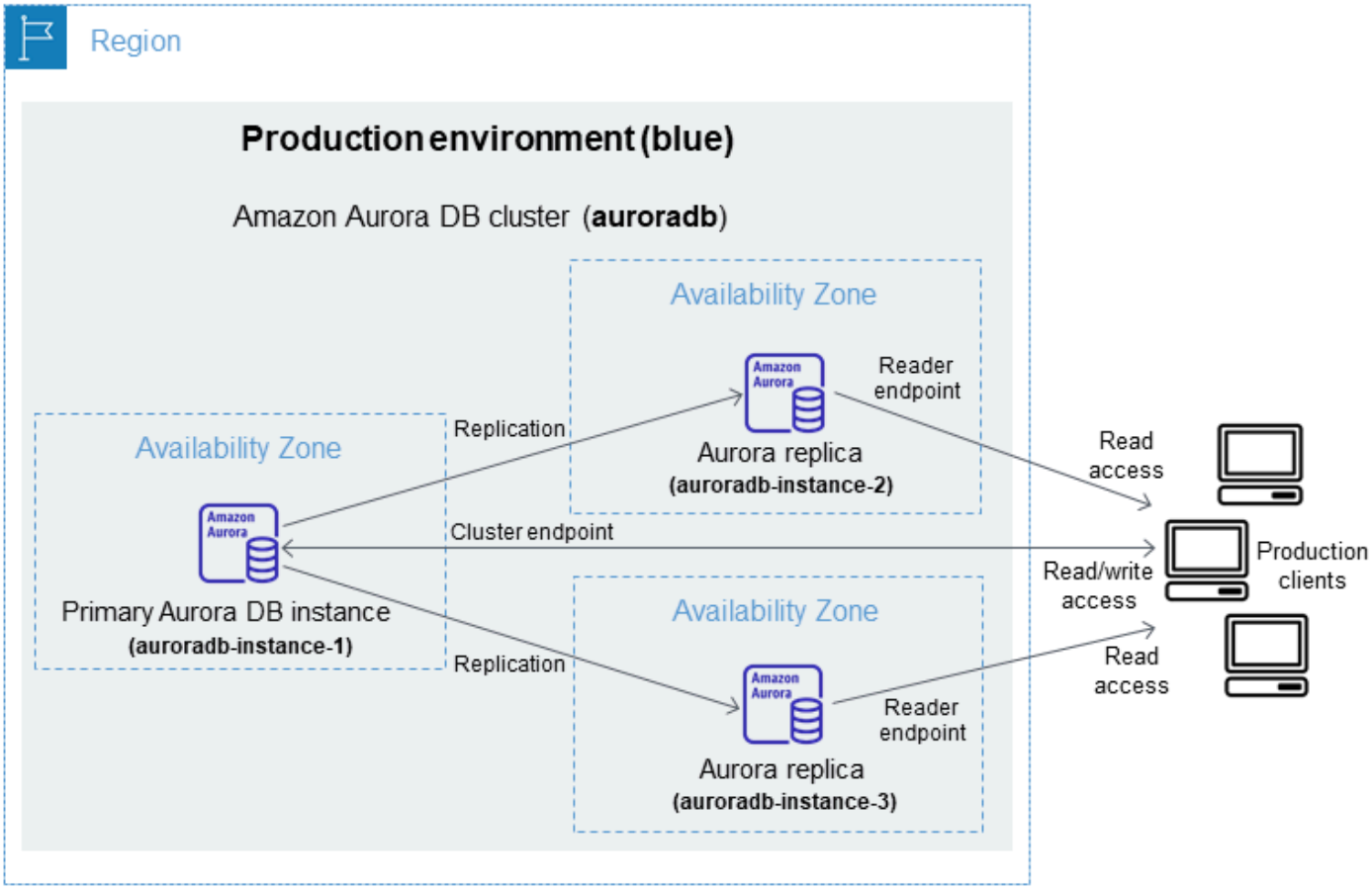
- 輕鬆建立生產就緒的預備環境。
- 自動將資料庫變更從生產環境複寫到預備環境。
- 在安全預備環境中測試資料庫變更，而不會影響生產環境。
- 隨時掌握最新的資料庫修補程式和系統更新。
- 實作和測試較新的資料庫功能。
- 切換預備環境以成為新的生產環境，而無需對應用程式進行變更。
- 透過使用內建的防護機制安全切換。
- 消除切換期間的資料遺失情況。
- 通常在一分鐘內快速切換，取決於您的工作負載。

藍/綠部署的工作流程

當您使用藍/綠部署進行 Aurora 資料庫叢集更新時，請完成下列主要步驟。

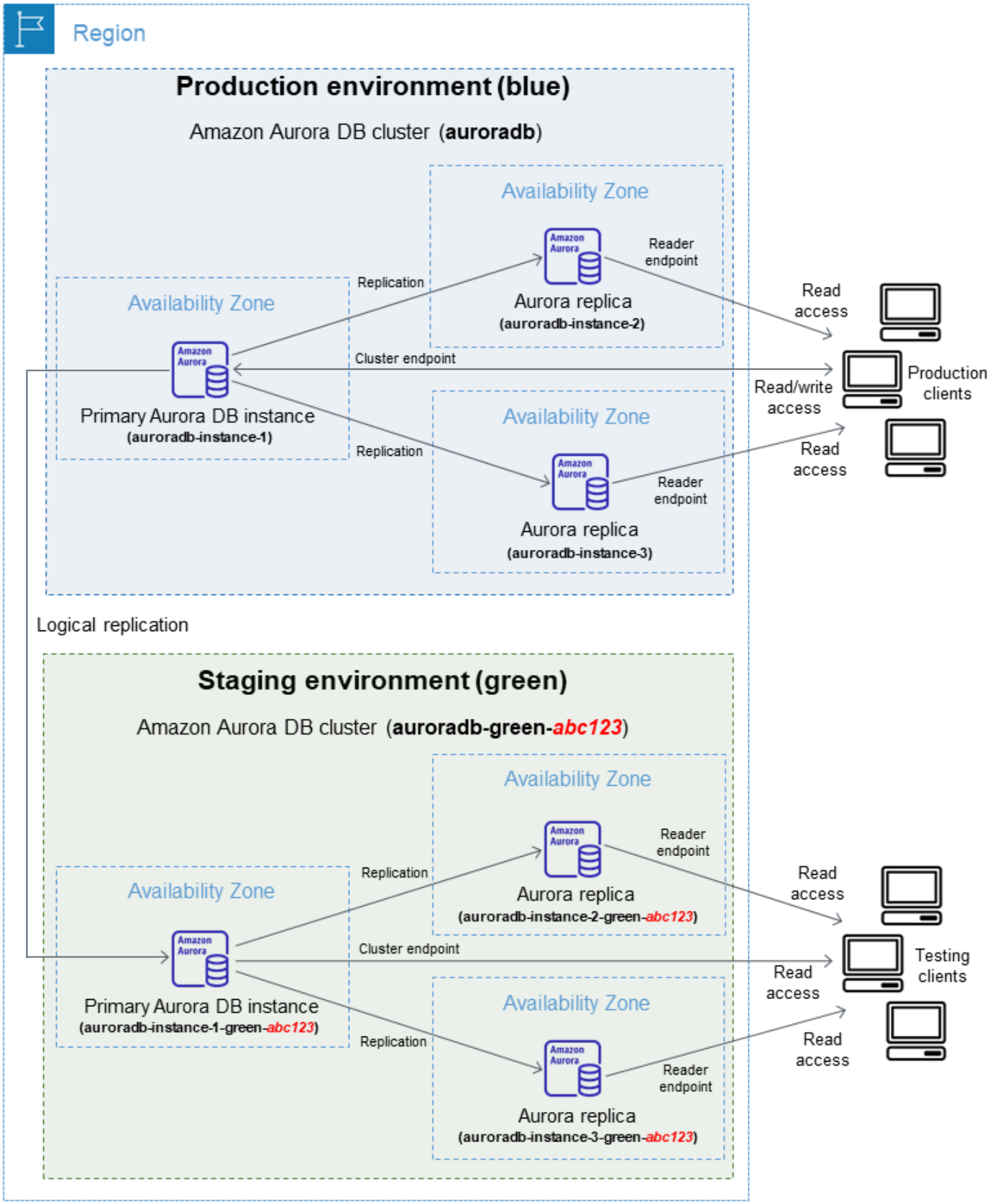
1. 識別需要更新的生產資料庫叢集。

   下圖顯示生產資料庫叢集的範例。



2. 建立藍/綠部署。如需說明，請參閱在 Amazon Aurora 中建立藍/綠部署。

   下圖顯示一個範例，說明如何從步驟 1 開始生產環境的藍/綠部署範例。在建立藍/綠部署時，RDS 會複製 Aurora 資料庫叢集的完整拓撲和組態，以建立綠色環境。複製的資料庫叢集和資料庫執行個體名稱都會附加 `-green-random-characters`。影像中的預備環境包含資料庫叢集 (auroradb-green-*abc123*)。它還包含資料庫叢集中的三個資料庫執行個體 (auroradb-instance1-green-*abc123*、auroradb-instance2-green-*abc123* 和 auroradb-instance3-green-*abc123*)。


Region
Production environment (blue)
Amazon Aurora DB cluster (auroradb)
Availability Zone
Aurora replica
(auroradb-instance-2)
Reader endpoint
Replication
Availability Zone
Primary Aurora DB instance
(auroradb-instance-1)
Cluster endpoint
Read access
Read/write access
Production clients
Availability Zone
Aurora replica
(auroradb-instance-3)
Replication
Reader endpoint
Read access
Logical replication
Staging environment (green)
Amazon Aurora DB cluster (auroradb-green-abc123)
Availability Zone
Aurora replica
(auroradb-instance-2-green-abc123)
Reader endpoint
Replication
Availability Zone
Primary Aurora DB instance
(auroradb-instance-1-green-abc123)
Cluster endpoint
Read access
Testing clients
Availability Zone
Aurora replica
(auroradb-instance-3-green-abc123)
Replication
Reader endpoint
Read access

建立藍/綠部署時，您可以指定較高的資料庫引擎版本，並針對綠色環境中的資料庫叢集指定不同的資料庫叢集參數群組。您也可以針對資料庫叢集中的資料庫執行個體指定不同的資料庫參數群組。

RDS 也會設定從藍色環境中主要資料庫執行個體到綠色環境中主要資料庫執行個體的複寫。

> **Important**
>
> 針對 Aurora MySQL 第 3 版，在您建立藍/綠部署之後，綠色環境中的資料庫叢集預設不允許寫入操作。然而，此設定不適用於具有 `CONNECTION_ADMIN` 權限的使用者 (包含 Aurora 主要使用者)。具有此權限的使用者可以覆寫 `read_only` 行為。如需詳細資訊，請參閱角色型權限模型。

3. 對預備環境進行變更。

   例如，您可能會變更綠色環境中一或多個資料庫執行個體所使用的資料庫執行個體類別。

   如需修改資料庫叢集的詳細資訊，請參閱 修改 Amazon Aurora 資料庫叢集。

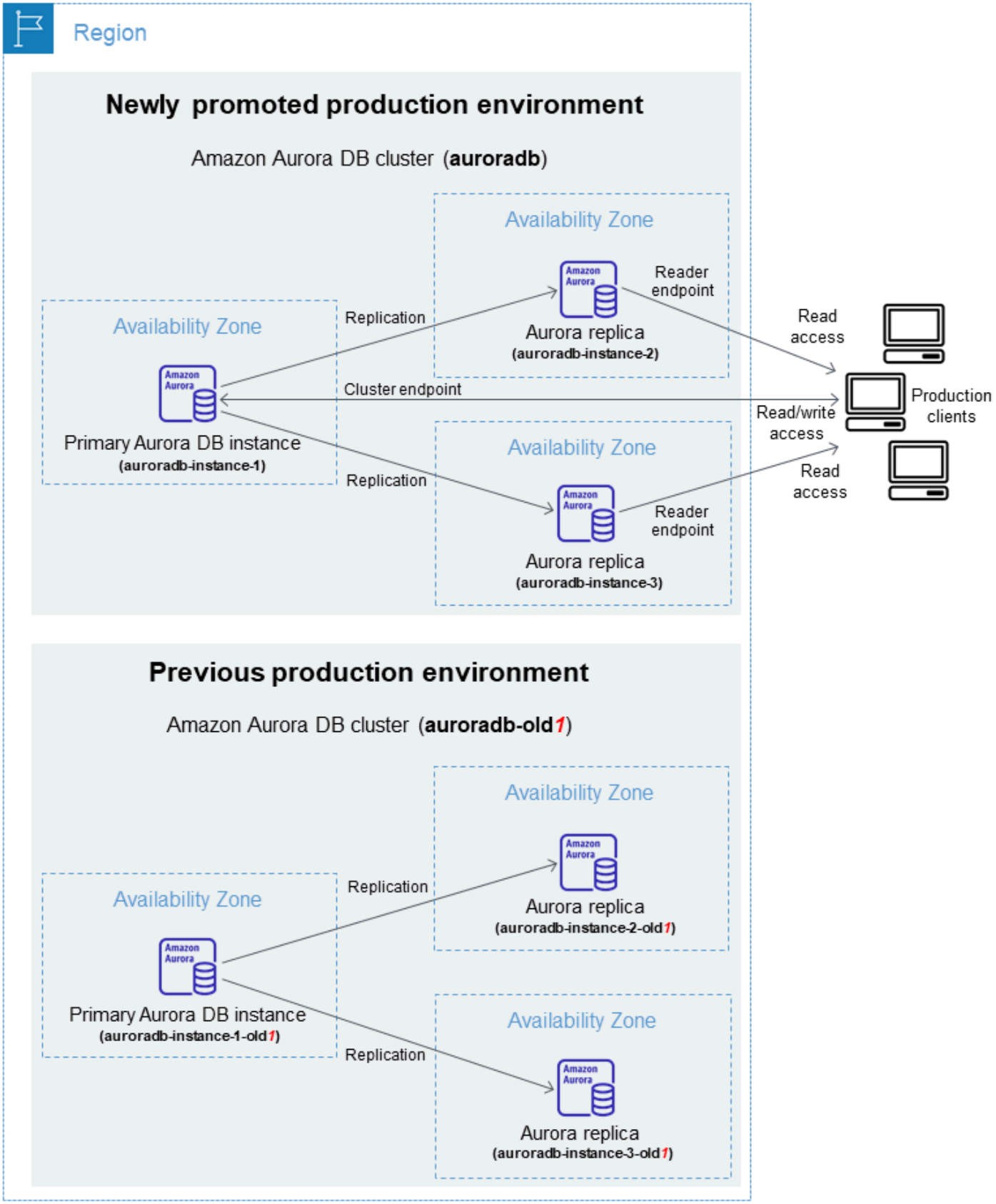
4. 測試您的預備環境。

   在測試期間，建議您將綠色環境中的資料庫保持唯讀狀態。在綠色環境上小心啟用寫入操作，因為它們可能會在綠色環境中造成複寫衝突。它們也可能會在切換後於生產資料庫中產生非預期的資料。若要啟用 Aurora MySQL 的寫入操作，請將 `read_only` 參數設為 `0`，然後重新啟動資料庫執行個體。對於 Aurora PostgreSQL，在工作階段層級將 `default_transaction_read_only` 參數設為 `off`。

5. 備妥後，請切換以將預備環境轉換為新的生產環境。如需說明，請參閱在 Amazon Aurora 中切換藍/綠部署。

   切換會產生停機時間。停機時間通常不到一分鐘，但可能更長，取決於您的工作負載。

   下圖顯示切換後的資料庫叢集。


Region
Newly promoted production environment
Amazon Aurora DB cluster (auroradb)
Availability Zone
Availability Zone
Amazon Aurora
Reader endpoint
Replication
Aurora replica
(auroradb-instance-2)
Read access
Cluster endpoint
Production clients
Primary Aurora DB instance
(auroradb-instance-1)
Read/write access
Availability Zone
Replication
Read access
Reader endpoint
Aurora replica
(auroradb-instance-3)
Previous production environment
Amazon Aurora DB cluster (auroradb-old1)
Availability Zone
Availability Zone
Replication
Aurora replica
(auroradb-instance-2-old1)
Primary Aurora DB instance
(auroradb-instance-1-old1)
Availability Zone
Replication
Aurora replica
(auroradb-instance-3-old1)

切換後，綠色環境中的 Aurora 資料庫叢集會成為新的生產資料庫叢集。目前生產環境中的名稱和端點會指派給新切換的生產環境，不需要對您的應用程式進行任何變更。因此，您的生產流量現在會流向新的生產環境。藍色環境中的資料庫叢集和資料庫執行個體會重新命名，方法是將 `-old`*n* 附加至目前名稱 (其中 *n* 是數字)。例如，假設藍色環境中資料庫執行個體的名稱為 `auroradb-instance-1`。切換後，資料庫執行個體名稱可能是 `auroradb-instance-1-old1`。

在影像的範例中，切換期間會發生下列變更：

- 綠色環境資料庫叢集 `auroradb-green-abc123` 會成為名為 `auroradb` 的生產資料庫叢集。
- 名為 `auroradb-instance1-green-abc123` 的綠色環境資料庫執行個體會成為生產資料庫執行個體 `auroradb-instance1`。
- 名為 `auroradb-instance2-green-abc123` 的綠色環境資料庫執行個體會成為生產資料庫執行個體 `auroradb-instance2`。
- 名為 `auroradb-instance3-green-abc123` 的綠色環境資料庫執行個體會成為生產資料庫執行個體 `auroradb-instance3`。
- 名為 `auroradb` 的藍色環境資料庫叢集會成為 `auroradb-old1`。
- 名為 `auroradb-instance1` 的藍色環境資料庫執行個體會成為 `auroradb-instance1-old1`。
- 名為 `auroradb-instance2` 的藍色環境資料庫執行個體會成為 `auroradb-instance2-old1`。
- 名為 `auroradb-instance3` 的藍色環境資料庫執行個體會成為 `auroradb-instance3-old1`。

6. 如果不再需要藍/綠部署，您可以將其刪除。如需說明，請參閱[刪除 Amazon Aurora 中的藍/綠部署](#)。

   切換後，不會刪除先前的生產環境，以便您可以在必要時使用它進行迴歸測試。

授予 Amazon Aurora 藍/綠部署操作的存取權

使用者必須具有必要的許可，才能執行與藍/綠部署相關的操作。您可以建立 IAM 政策，許可使用者和角色對其需要的指定資源執行特定 API 操作。然後，您可以將這些政策連線至需要這些許可的 IAM 許可集或角色。如需詳細資訊，請參閱[Amazon Aurora 的 Identity and access management](#)。

建立藍/綠部署的使用者必須具有執行下列 RDS 操作的許可：

- `rds:CreateBlueGreenDeployment`

- rds:AddTagsToResource
- rds:CreateDBCluster
- rds:CreateDBInstance
- rds:CreateDBClusterEndpoint

切換藍/綠部署的使用者必須具有執行下列 RDS 操作的許可：

- rds:SwitchoverBlueGreenDeployment
- rds:ModifyDBCluster
- rds:PromoteReadReplicaDBCluster

刪除藍/綠部署的使用者必須具有執行下列 RDS 操作的許可：

- rds:DeleteBlueGreenDeployment
- rds:DeleteDBCluster
- rds:DeleteDBInstance
- rds:DeleteDBClusterEndpoint

Aurora 會代表您佈建和修改預備環境中的資源。這些資源包含使用內部定義命名慣例的資料庫執行個體。因此，連接的 IAM 政策不能包含部分資源名稱模式，例如 my-db-prefix-*。僅支援萬用字元 (*)。一般而言，建議使用資源標籤和其他支援的屬性 (而不是萬用字元)，來控制對這些資源的存取權。如需詳細資訊，請參閱適用於 Amazon RDS 的動作、資源和條件索引鍵。

Aurora Global Database 藍/綠部署的其他許可

為 Aurora Global Database 叢集建立藍/綠部署時，除了上述許可之外，使用者還需要下列許可才能執行 操作來管理全域叢集拓撲。

建立藍/綠部署的使用者必須具有執行下列 RDS 操作的許可：

- rds:CreateGlobalCluster

切換藍/綠部署的使用者必須具有執行下列 RDS 操作的許可：

- rds:ModifyGlobalCluster
- rds:PromoteReadReplicaDBCluster

刪除藍/綠部署的使用者必須具有執行下列 RDS 操作的許可：

- `rds:DeleteGlobalCluster`

Amazon Aurora 藍/綠部署的限制和考量事項

若要在 Amazon RDS 中進行藍/綠部署，需要仔細考量複寫槽、資源管理、執行個體大小，以及對資料庫效能的潛在影響等因素。以下各節提供指引，協助您最佳化部署策略，以確保資料庫環境的最短停機時間、無縫轉換和有效管理。

主題

- 藍/綠部署的限制
- 藍/綠部署的 Aurora Global Database 限制
- 藍/綠部署考量

藍/綠部署的限制

下列限制適用於藍/綠部署。

主題

- 藍/綠部署的一般限制
- 藍/綠部署的 Aurora MySQL 限制
- Aurora PostgreSQL 藍/綠部署限制

藍/綠部署的一般限制

下列一般限制適用於藍/綠部署：

- 您無法停止和啟動屬於藍/綠部署一部分的叢集。
- 藍/綠部署不支援使用 AWS Secrets Manager管理主要使用者密碼。
- 如果您嘗試在藍色資料庫叢集強制執行恢復，則藍/綠部署會中斷，而轉換會遭到封鎖。
- 在切換期間，藍色和綠色環境無法與 Amazon Redshift 進行零 ETL 整合。您必須先刪除整合再進行切換，然後重新建立整合。
- 建立藍/綠部署時，必須在綠色環境上停用事件排程器 (`event_scheduler` 參數)。這樣可以防止在綠色環境中產生事件並導致不一致。

- 在藍色資料庫叢集設定的自動擴展政策不會複製到綠色環境。您必須在轉換後進行重新設定，無論這些政策最初是在藍色或綠色環境中設定。
- 您無法將未加密的資料庫叢集變更為加密的資料庫叢集。此外，您無法將加密的資料庫叢集變更為未加密的資料庫叢集。
- 您無法將藍色資料庫叢集變更為高於其對應綠色資料庫叢集的引擎版本。
- 藍色環境和綠色環境中的資源必須在同一 AWS 帳戶中。
- 下列功能不支援藍/綠部署：
  - Amazon RDS Proxy
  - 跨區域僅供讀取複本
  - Aurora Serverless v1 資料庫叢集
  - CloudFormation

藍/綠部署的 Aurora MySQL 限制

下列限制適用於 Aurora MySQL 藍/綠部署：

- 來源資料庫叢集不能包含任何名為 `tmp` 的資料庫。具有此名稱的資料庫不會複製到綠色環境。
- 藍色資料庫叢集不能是外部 binlog 複本。
- 如果是已啟用恢復的來源資料庫叢集，則會在沒有恢復支援的情況下建立綠色資料庫叢集。這是因為恢復不適用於二進位日誌 (binlog) 複寫，但藍/綠部署需要此功能。如需詳細資訊，請參閱the section called “恢復資料庫叢集”。
- 藍/綠部署不支援 AWS JDBC Driver for MySQL。如需詳細資訊，請參閱在 GitHub 的已知限制。

Aurora PostgreSQL 藍/綠部署限制

下列限制適用於 Aurora PostgreSQL 藍/綠部署 ()。。

- 除非在藍色資料庫叢集上將 `rds.logically_replicate_unlogged_tables` 參數設定為 `1`，否則不會將未記錄的資料表複寫到綠色環境。在建立藍/綠部署之後不要修改此參數值，以避免可能在未記錄的資料表發生複寫錯誤。
- 藍色資料庫叢集不能是邏輯來源 (發布者) 或複本 (訂閱用戶)。
- 如果藍色資料庫叢集設定為外部資料包裝函式 (FDW) 延伸模組的外部伺服器，您必須使用叢集端點名稱，而非 IP 位址。這可讓組態在轉換之後保持運作狀態。

- 在藍/綠部署中，每個資料庫都需要邏輯複寫槽。隨著資料庫數量的增加，資源負荷會增加，並可能導致複寫延遲，特別是當資料庫叢集未充分擴展時。此影響取決於資料庫工作負載和連線數量等因素。若要緩解此問題，請考慮擴展資料庫執行個體類別，或減少來源叢集的資料庫數目。
- 針對 15.7 版和更新的 15 版，以及 16.3 版和更新的 16 版，僅支援Babelfish for Aurora PostgreSQL 的藍/綠部署。
- 如果想要在 Aurora 複本中擷取執行計劃，您必須在呼叫 `apg_plan_mgmt.create_replica_plan_capture` 函數時提供藍色資料庫叢集端點。這可確保計劃擷取在轉換之後繼續運作。如需詳細資訊，請參閱the section called “擷取複本中的 Aurora PostgreSQL 執行計畫”。
- 在綠色環境中邏輯複寫套用程序為單執行緒。如果藍色環境產生大量寫入流量，則綠色環境可能無法跟上進度。這可能會導致複寫延遲或失敗，對於產生持續高寫入輸送量的工作負載尤其如此。請務必徹底測試工作負載。對於需要主要版本升級和處理大量寫入工作負載的案例，請考慮使用 AWS Database Migration Service (AWS DMS) 或自我管理邏輯複寫等替代方法。
- 在 Aurora 的藍/綠部署期間，不支援在分割資料表建立新的分割區。建立新的分割區涉及資料定義語言 (DDL) 操作 (例如 `CREATE TABLE`)，系統不會將這些操作從藍色環境複寫到綠色環境。不過，系統會將現有的分割資料表及其資料複寫至綠色環境。
- 下列限制適用於 PostgreSQL 延伸模組：
  - 建立藍/綠部署時，必須在藍色環境中停用 `pg_partman` 延伸模組。延伸模組會執行 DDL 作業 (例如 `CREATE TABLE`)，其會中斷從藍色環境到綠色環境的邏輯複寫。
  - 建立藍/綠部署之後，所有綠色資料庫上的 `pg_cron` 延伸模組必須保持停用狀態。延伸模組具有以超級使用者身分執行的背景工作者，且會略過綠色環境的唯讀設定，這可能會造成複寫衝突。
  - `apg_plan_mgmt` 延伸模組必須在所有綠色資料庫上將 `apg_plan_mgmt.capture_plan_baselines` 參數設定為 `off`，以避免在藍色環境中擷取相同的計劃時，發生主索引鍵衝突。如需詳細資訊，請參閱the section called “Aurora PostgreSQL 查詢計劃管理的概觀”。
  - 建立藍/綠部署時，必須在藍色環境上停用 `pglogical` 和 `pgactive` 延伸模組。將綠色環境切換為新的生產環境之後，您可以再次啟用延伸模組。此外，藍色資料庫不能是外部執行個體的邏輯訂閱者。
  - 如果您使用的是 `pgAudit` 延伸模組，其必須保留在藍色和綠色資料庫執行個體的自訂資料庫參數群組上的共用程式庫 (`shared_preload_libraries`) 中。如需詳細資訊，請參閱the section called “設定 pgAudit 擴充功能”。

藍/綠部署的邏輯複寫特定限制

PostgreSQL 具有某些與邏輯複寫相關的限制，其會在針對 Aurora PostgreSQL 資料庫叢集建立藍/綠部署時轉換為限制。

下表描述適用於 Aurora PostgreSQL 藍/綠部署的邏輯複寫限制。如需詳細資訊，請參閱 PostgreSQL 邏輯複寫文件中的限制。

限制	說明
資料定義語言 (DDL) 陳述式 (例如 `CREATE TABLE` 和 `CREATE SCHEMA`) 不會從藍色環境複寫到綠色環境。	如果 Aurora 在藍色環境中偵測到 DDL 變更，則您的綠色資料庫會進入複寫降級狀態。您必須刪除藍/綠部署和所有綠色資料庫，然後重新建立該部署。
系統不會將資料控制語言 (DCL) 陳述式 (例如 `GRANT` 和 `REVOKE`) 從藍色環境複寫到綠色環境。	如果 Aurora 偵測到有人嘗試在藍色環境中執行 DCL 陳述式，您將看到警告訊息。沒有可供變更此行為的組態或 API，因為其是藍/綠部署程序的限制。
序列物件上的 `NEXTVAL` 作業不會在藍色環境與綠色環境之間同步。	在轉換期間，Aurora 會在綠色環境中遞增序列值，以符合藍色環境中的序列值。如果您有數千個序列，這可能會延遲轉換。
藍色環境中的大型物件不會複寫到綠色環境。這包括現有的大型物件，以及藍/綠部署程序期間任何新建或修改的大型物件。	如果 Aurora 在藍色環境中偵測到 `pg_largeobject` 系統資料表中儲存的大型物件建立或修改，則您的綠色資料庫將進入複寫降級狀態。您必須刪除藍/綠部署和所有綠色資料庫，然後重新建立該部署。

限制	說明
重新整理具體化視觀表會中斷複寫。	在藍色環境中重新整理具體化視觀表會中斷在綠色環境中的複寫。避免在藍色環境中重新整理具體化視觀表。轉換後，您可以使用 REFRESH MATERIALIZED VIEW 命令手動重新整理，或安排重新整理。
不允許在沒有主索引鍵的資料表上執行 UPDATE 和 DELETE 作業。	在建立藍/綠部署之前，請確定所有資料表都有主索引鍵或使用 `REPLICA IDENTITY FULL`。不過，只有在不存在主索引鍵或唯一索引鍵時才能使用 `REPLICA IDENTITY FULL`，因為其會影響複寫效能。如需詳細資訊，請參閱 PostgreSQL 文件。

藍/綠部署的 Aurora Global Database 限制

除了上述一般和引擎特定限制之外，下列限制適用於 Aurora Global Database 的藍/綠部署：

- 所有操作都必須從與全球資料庫寫入器叢集相同的區域啟動。
- 執行全域切換或全域容錯移轉會導致作用中的藍/綠部署失效。藍綠部署需要從新的主要區域刪除並重新建立。
- 對於 Aurora PostgreSQL，如果您已在生產環境中啟用全域寫入轉送並建立藍/綠部署，則會在綠色叢集上停用寫入轉送。只有在綠色環境成為新的生產環境時，才會在藍/綠切換之後，才會在綠色環境中啟用。切換後，會在`-old1`叢集上停用寫入轉送。
- 在建立藍/綠部署之後修改全域資料庫的拓撲，會導致作用中的藍/綠部署失效。藍綠部署必須從新的主要區域刪除並重新建立。
- 自動化快照會根據最初在舊藍色環境中設定的備份保留天數進行保留。舊藍叢集的自動快照不會複製到綠色。
- 在藍/綠轉換期間支援全域容錯移轉，但在藍/綠轉換期間不支援全域轉換。
- 確保綠色環境的資料庫叢集和資料庫參數群組存在於名稱相同的所有次要區域中。如果任何區域中的參數群組無法使用，則會使用區域中的預設參數群組。
- 在藍/綠部署切換期間，避免在任何全域資料庫成員上使用 RDS Proxy。

藍/綠部署考量

Amazon RDS 會使用每個資源的 `DbiResourceId` 和 `DbClusterResourceId` 追蹤藍/綠部署中的資源。此資源 ID 是 資源 AWS 區域的唯一不可變識別符。

資源 ID 與資料庫叢集 ID 不同。在 RDS 主控台的資料庫組態中會列出每一項。

當您切換藍/綠部署時，資源的名稱 (叢集 ID) 會變更，但每個資源都保留相同的資源 ID。例如，資料庫叢集識別符可能已是藍色環境中的 `mycluster`。切換後，相同的資料庫叢集可能重新命名為 `mycluster-old1`。不過，資料庫叢集的資源 ID 在切換期間不會變更。因此，將綠色資源轉換為新的生產資源時，其資源 ID 與先前位於生產環境中的藍色資源 ID 不符。

在切換藍/綠部署之後，請考慮將資源 ID 更新為新轉換之生產資源的 ID，以取得與生產資源搭配使用的整合功能和服務。具體來說，請考慮下列更新：

- 如果您使用 RDS API 和資源 ID 執行篩選，請在切換後調整用於篩選的資源 ID。
- 如果您使用 CloudTrail 來稽核資源，請調整 CloudTrail 的取用者，以在切換後追蹤新的資源 ID。如需詳細資訊，請參閱在 AWS CloudTrail 中監控 Amazon Aurora API 呼叫。
- 如果您針對藍色環境中的資源使用資料庫活動串流，請調整您的應用程式，以在切換後監控新串流的資料庫事件。如需詳細資訊，請參閱資料庫活動串流的支援區域和 Aurora 資料庫引擎。
- 如果您使用 Performance Insights API，請在切換後調整 API 呼叫中的資源 ID。如需詳細資訊，請參閱在 Amazon Aurora 上使用績效詳情監控資料庫負載。

  您可以在切換後監控具有相同名稱的資料庫，但其不包含切換前的資料。
- 如果您在 IAM 政策中使用資源 ID，請務必在必要時新增新轉換資源的資源 ID。如需詳細資訊，請參閱Amazon Aurora 的 Identity and access management。
- 如果您有與資料庫叢集相關聯的 IAM 角色，請務必在轉換後重新建立關聯。連接的角色不會自動複製到綠色環境。
- 如果您使用 IAM 資料庫身份驗證對資料庫叢集進行身分驗證，請確定用於資料庫存取的 IAM 政策同時具有政策 `Resource` 元素下方列出的藍色和綠色資料庫。這是必要的，以便在切換後連線到綠色資料庫。如需詳細資訊，請參閱the section called “建立並使用 IAM 政策進行 IAM 資料庫存取”。
- 如果您想要還原屬於藍/綠部署之資料庫叢集的手動資料庫叢集快照，請確定透過檢查取得快照的時間來還原正確的資料庫叢集快照。如需詳細資訊，請參閱從資料庫叢集快照還原。
- 切換後，由於藍色環境的檢查點在綠色環境中無效，因此 AWS Database Migration Service (AWS DMS) 複寫任務無法繼續。您必須使用新檢查點重新建立 DMS 任務，才能繼續複寫。
- Amazon Aurora 在藍色環境中複製基礎 Aurora 儲存磁碟區，進而建立綠色環境。綠色叢集磁碟區只會儲存對綠色環境所做的增量變更。若您刪除藍色環境中的資料庫叢集，綠色環境中的基礎 Aurora 儲存磁碟區大小會增加到完整大小。如需詳細資訊，請參閱the section called “複製 Aurora 資料庫叢集的一個磁碟區”。
- 若您在藍/綠部署的綠色環境中將資料庫執行個體新增至資料庫叢集，則在切換時，新的資料庫執行個體不會取代藍色環境中的資料庫執行個體。不過，新的資料庫執行個體會保留在資料庫叢集中，並在新的生產環境中成為資料庫執行個體。

- 在藍/綠部署的綠色環境中刪除資料庫叢集中的資料庫執行個體時，您無法建立新的資料庫執行個體，以在藍/綠部署中取代該執行個體。

  如果您使用與所刪除資料庫執行個體相同的名稱和 ARN 建立的新資料庫執行個體，則它具有不同的 `DbiResourceId`，因此不屬於綠色環境。

  如果您在綠色環境中刪除資料庫叢集中的資料庫執行個體，則會產生下列行為：

  - 如果藍色環境中存在名稱相同的資料庫執行個體，則它不會切換至綠色環境中的資料庫執行個體。此資料庫執行個體不會透過將 `-old`*n* 附加至資料庫執行個體名稱來重新命名。
  - 指向藍色環境中資料庫執行個體的任何應用程式都會在切換後繼續使用相同的資料庫執行個體。
- 如果您使用資源標籤進行存取控制或操作管理，則需要了解在切換之前，藍色和綠色環境之間不會同步標籤變更。當您建立藍/綠部署時，藍環境的標籤會複製到綠色環境。建立之後，您對任一環境所做的任何標籤修改都不會自動同步。在切換期間，藍色環境標籤會取代綠色環境中的所有標籤。在建立藍/綠部署之前，請將所有必要的標籤套用至藍環境，或在切換後將必要的標籤重新套用至新的生產環境。如需標籤的詳細資訊，請參閱標記 Amazon Aurora 和 Amazon RDS 資源。

Amazon Aurora 藍/綠部署的最佳實務

下列是藍/綠部署的最佳實務。

主題

- 藍/綠部署的一般最佳實務
- Aurora MySQL 藍/綠部署最佳實務
- Aurora PostgreSQL 藍/綠部署的最佳實務
- Aurora 全域資料庫藍/綠部署最佳實務

藍/綠部署的一般最佳實務

建立藍/綠部署時，請考慮下列一般最佳實務。

- 切換前，請在綠色環境中徹底測試 Aurora 資料庫叢集。
- 在綠色環境中將您的資料庫保留唯讀狀態。建議您在綠色環境上小心啟用寫入操作，因為它們可能會在綠色環境中造成複寫衝突。它們也可能會在切換後於生產資料庫中產生非預期的資料。
- 使用藍/綠部署實作結構描述變更時，請僅進行複寫相容的變更。

例如，您可以在資料表尾端新增欄，而不中斷從藍色部署到綠色部署的複寫。不過，結構描述變更 (例如重新命名資料欄或重新命名資料表) 會中斷綠色部署的複寫。

如需有關複寫相容變更的詳細資訊，請參閱 MySQL 文件中的在來源和複本上使用不同的資料表定義進行複寫，以及 PostgreSQL 邏輯複寫文件中的限制。

- 針對兩個環境中的所有連線，使用叢集端點、讀取器端點或自訂端點。請不要搭配靜態或排除清單使用執行個體端點或自訂端點。
- 當您切換藍/綠部署時，請遵循切換最佳實務。如需詳細資訊，請參閱the section called “切換最佳實務”。

Aurora MySQL 藍/綠部署最佳實務

當您從 Aurora MySQL 資料庫叢集建立藍/綠部署時，請考慮下列最佳實務。

- 如果綠色環境遇到複本延遲，請考慮下列事項：
  - 如果不需要，請停用二進位日誌保留，或暫時停用，直到複寫趕上進度為止。若要這樣做，請將 `binlog_format` 資料庫叢集參數設回 `0`，然後重新啟動綠色寫入器資料庫執行個體。
  - 暫時在綠色資料庫參數群組中將 `innodb_flush_log_at_trx_commit` 參數設為 `0`。在複寫趕上進度後，在轉換之前還原為預設值 `1`。如果使用暫時參數值時發生非預期的關機或當機，請重建綠色環境，以避免未偵測到的資料損毀。如需詳細資訊，請參閱 the section called “設定日誌緩衝區的排清頻率”。

Aurora PostgreSQL 藍/綠部署的最佳實務

當您從 Aurora PostgreSQL 資料庫叢集建立藍/綠部署時，請考慮下列最佳實務。

- 監控 Aurora PostgreSQL 邏輯複寫直接寫入式快取，如有必要，請對快取緩衝區進行調整。如需詳細資訊，請參閱the section called “監控邏輯複寫直接寫入式快取”。
- 增加藍色環境中 `logical_decoding_work_mem` 資料庫參數的值。這樣做允許減少磁碟上的解碼，並改用記憶體。如需詳細資訊，請參閱the section called “調整邏輯解碼的工作記憶體”。
  - 您可以使用 `ReplicationSlotDiskUsage` CloudWatch 指標監控正在寫入至磁碟的交易溢出。此指標可讓您深入了解複寫槽的磁碟使用情況，協助識別交易資料何時超過記憶體容量與何時存放在磁碟。您可以使用 `FreeableMemory` CloudWatch 指標來監控可用記憶體。如需詳細資訊，請參閱the section called “Amazon Aurora 的執行個體層級指標”。

  - 在 Aurora PostgreSQL 第 14 版及更新版本中，您可以使用 pg_stat_replication_slots 系統檢視來監控邏輯溢出檔案的大小。
- 建立藍/綠部署之前，請先將所有 PostgreSQL 延伸模組更新至最新版本。如需詳細資訊，請參閱the section called “升級 PostgreSQL 擴充功能”。
- 如果您使用 aws_s3 延伸模組，請務必在建立綠色環境之後透過 IAM 角色，將綠色資料庫叢集存取權授與 Amazon S3。這允許匯入和匯出命令在轉換之後繼續運作。如需說明，請參閱the section called “設定對 Amazon S3 儲存貯體的存取權”。
- 如果您為綠色環境指定較高的引擎版本，請在所有資料庫執行 ANALYZE 操作以重新整理 pg_statistic 資料表。主要版本升級期間不會傳輸最佳化工具統計資料，因此您必須重新產生所有統計資料以避免效能問題。如需主要版本升級期間的其他最佳實務，請參閱the section called “執行主要版本升級”。
- 如果在來源使用觸發條件來操作資料，請避免將觸發條件設為 ENABLE REPLICA 或 ENABLE ALWAYS。否則，複寫系統會傳播變更與執行觸發條件，而會導致複寫。
- 長時間執行的交易可能會導致嚴重的複本延遲。若要縮短複本延遲，請考慮執行下列動作：
  - 減少長時間執行的交易和子交易，這些交易可能會延遲，直到綠色環境趕上藍色環境為止。
  - 減少藍色環境的大量操作，直到綠色環境趕上藍色環境為止。
  - 在建立藍/綠部署之前，對忙碌資料表啟動手動清空凍結操作。如需說明，請參閱執行手動清空凍結。
  - 在 PostgreSQL 第 12 版及更新版本中，停用大型或忙碌資料表的 index_cleanup 參數，以提高藍色資料庫定期維護的效率。如需詳細資訊，請參閱盡快清空資料表。

    > **Note**
    >
    > 在清空期間定期略過索引清除可能會導致索引膨脹，而可能會掃描效能下降。根據最佳實務，只有在使用藍/綠部署時，才使用此方法。部署完成後，建議您繼續定期維護和清除索引。

  - 如果工作負載的藍色和綠色資料庫執行個體大小過小，則可能會發生複本延遲。確保資料庫執行個體未達到執行個體類型的資源限制。如需詳細資訊，請參閱the section called “使用 CloudWatch 指標分析資源用量”。
- 慢速複寫可能會導致寄件者和接收者經常重新啟動，進而延遲同步。若要確保其保持作用中狀態，請在藍色環境中將 wal_sender_timeout 參數設為 0，並在綠色環境中將 wal_receiver_timeout 參數設為 0 以停用逾時。

- 檢閱 UPDATE 和 DELETE 陳述式的效能，以及評估在 WHERE 子句中使用的欄建立索引是否可以最佳化這些查詢。這可以在綠色環境中重播操作時增強效能。如需詳細資訊，請參閱 the section called “對產生等待的查詢檢查述詞篩選條件”。
- 如果您使用的是觸發條件，請確定其不會干擾建立、更新和捨棄名稱開頭為「rds」的 `pg_catalog.pg_publication`、`pg_catalog.pg_subscription` 和 `pg_catalog.pg_replication_slots` 物件。
- 如果在來源資料庫叢集啟用 Babelfish，則下列參數在綠色環境的目標資料庫叢集參數群組中，必須具有與來源資料庫叢集參數群組中相同的設定：
  - rds.babelfish_status
  - babelfishpg_tds.tds_default_numeric_precision
  - babelfishpg_tds.tds_default_numeric_scale
  - babelfishpg_tsql.default_locale
  - babelfishpg_tsql.migration_mode
  - babelfishpg_tsql.server_collation_name

  如需這些參數的相關資訊，請參閱 the section called “Babelfish 的資料庫叢集參數群組設定 ”。

Aurora 全域資料庫藍/綠部署最佳實務

除了上述一般和引擎特定最佳實務之外，請考慮 Aurora Global Database 的最佳實務。

- 監控下列 CloudWatch 指標，以識別生產環境中活動量低的期間：
  - `DatabaseConnections`
  - `ActiveTransactions`

  在計劃的維護時段或活動量低的期間，安排藍/綠轉換。
- 藍/綠轉換持續時間取決於您的工作負載和次要區域的數量。當您啟動藍/綠切換時，服務會等待複本延遲達到零，然後再繼續。我們建議您在開始切換之前檢查複本延遲。
- 如果您想要使用綠色環境預設參數以外的資料庫參數或資料庫叢集參數群組，請先在所有次要區域中建立名稱相同的所需參數群組，再啟動藍/綠部署。

在 Amazon Aurora 中建立藍/綠部署

RDS 將藍色環境的拓撲和功能複製到預備區域。當藍色資料庫執行個體具有僅供讀取複本時，系統會將僅供讀取複本複製為綠色執行個體的複本。所有綠色複本的分配儲存符合綠色主要執行個體，而其他儲存參數則繼承自藍色複本。

建立藍/綠部署時，您可以指定要在部署中複製的資料庫叢集。您選擇的資料庫叢集是生產資料庫叢集，而且其會成為藍色環境中的資料庫叢集。RDS 將藍色環境的拓撲複製到預備區域，以及複製其設定的功能。資料庫叢集會複製到綠色環境，而且 RDS 會設定從藍色環境中的資料庫叢集複寫到綠色環境中的資料庫叢集。RDS 也會複製資料庫叢集中的所有資料庫執行個體。

主題

- 準備進行藍/綠部署
- 在建立藍/綠部署時指定變更
- 建立藍/綠部署
- 建立藍/綠部署的設定

準備進行藍/綠部署

根據 Aurora 資料庫叢集所執行的引擎而定，建立藍/綠部署之前，您必須執行某些步驟。

主題

- 準備 Aurora MySQL 資料庫叢集以進行藍/綠部署
- 準備 Aurora PostgreSQL 資料庫叢集以進行藍/綠部署
- 準備 Aurora 全域資料庫資料庫叢集以進行藍/綠部署

準備 Aurora MySQL 資料庫叢集以進行藍/綠部署

在針對 Aurora MySQL 資料庫叢集建立藍/綠部署之前，叢集必須與已開啟二進位記錄 (`binlog_format`) 的自訂資料庫叢集參數群組相關聯。從藍色環境複寫到綠色環境時，需要二進位記錄。雖然任何 binlog 格式都可以運作，但我們建議 `ROW`，以降低複寫不一致的風險。如需建立自訂資料庫叢集參數群組和設定參數的相關資訊，請參閱 the section called “資料庫叢集參數群組”。

> **Note**
>
> 啟用二進位日誌記錄會增加資料庫叢集的寫入磁碟 I/O 操作次數。您可以使用 `VolumeWriteIOPs` CloudWatch 指標來監控 IOPS 使用情況。

啟用二進位記錄之後，請務必重新啟動資料庫叢集，以讓變更生效。藍/綠部署要求寫入器執行個體與資料庫叢集參數群組同步，否則建立作業將會失敗。如需詳細資訊，請參閱在 Aurora 叢集中重新啟動資料庫執行個體。

此外，我們建議將二進位日誌保留期變更為 NULL 以外的值，以防止二進位日誌檔案遭到清除。如需詳細資訊，請參閱the section called “設定和顯示二進位日誌組態”。

準備 Aurora PostgreSQL 資料庫叢集以進行藍/綠部署

在針對 Aurora PostgreSQL 資料庫叢集建立藍/綠部署之前，請務必執行下列動作。

- 將叢集與已啟用邏輯複寫 (`rds.logical_replication`) 的自訂資料庫叢集參數群組建立關聯。從藍色環境複寫到綠色環境時，需要邏輯複寫。

  啟用邏輯複寫時，您也需要調校特定叢集參數，例如 `max_replication_slots`、`max_logical_replication_workers` 和 `max_worker_processes`。如需啟用邏輯複寫和調校這些參數的指示，請參閱the section called “設定邏輯複寫”。

  此外，請確保將 `synchronous_commit` 參數設為 `on`。

  設定必要的參數之後，請重新啟動資料庫叢集，讓變更生效。藍/綠部署要求寫入器執行個體與資料庫叢集參數群組同步，否則建立作業將會失敗。如需詳細資訊，請參閱在 Aurora 叢集中重新啟動資料庫執行個體。
- 確定資料庫叢集執行的是與藍/綠部署相容的 Aurora PostgreSQL 版本。如需相容版本的清單，請參閱 the section called “透過 Aurora PostgreSQL 進行藍/綠部署”。
- 請確定資料庫叢集中的所有資料表都有主索引鍵。PostgreSQL 邏輯複寫不允許對沒有主索引鍵的資料表進行 UPDATE 或 DELETE 操作。

準備 Aurora 全域資料庫資料庫叢集以進行藍/綠部署

為您的 Aurora Global Database 資料庫叢集建立藍/綠部署之前，請注意下列幾點：

- 所有操作都必須從與全球資料庫寫入器叢集相同的區域啟動。
- 參數群組組態：
  - 綠色環境會使用您指定的新參數群組，或與藍色叢集相同的參數群組 （預設）。
  - 自訂參數群組會複製到綠色環境。
  - 如果指定的參數群組不存在於次要區域中，則次要區域中的預設參數群組會用於綠色環境。

在建立藍/綠部署時指定變更

建立藍/綠部署時，您可以在綠色環境中對資料庫叢集進行下列變更。

您可以在部署之後，於綠色環境中對資料庫叢集及其資料庫執行個體進行其他修改。例如，您可以指定較高的引擎版本或不同的參數群組。

如需修改資料庫叢集的詳細資訊，請參閱 修改 Amazon Aurora 資料庫叢集。

主題

- 指定較高的引擎版本
- 指定不同的資料庫參數群組

指定較高的引擎版本

如果想要測試資料庫引擎升級，您可以指定更高的引擎版本。轉換時，資料庫會升級至您指定的主要或次要資料庫引擎版本。

指定不同的資料庫參數群組

指定資料庫叢集參數群組，需與資料庫叢集所使用的資料庫叢集參數群組不同。您可以測試參數變更如何影響綠色環境中的資料庫叢集，或在升級時針對新的主要資料庫引擎版本指定參數群組。

如果您指定不同的資料庫叢集參數群組，則指定的參數群組會與綠色環境中的資料庫叢集相關聯。如果您未指定不同的資料庫叢集參數群組，則綠色環境中的資料庫叢集會與藍色資料庫叢集相同的參數群組相關聯。

建立藍/綠部署

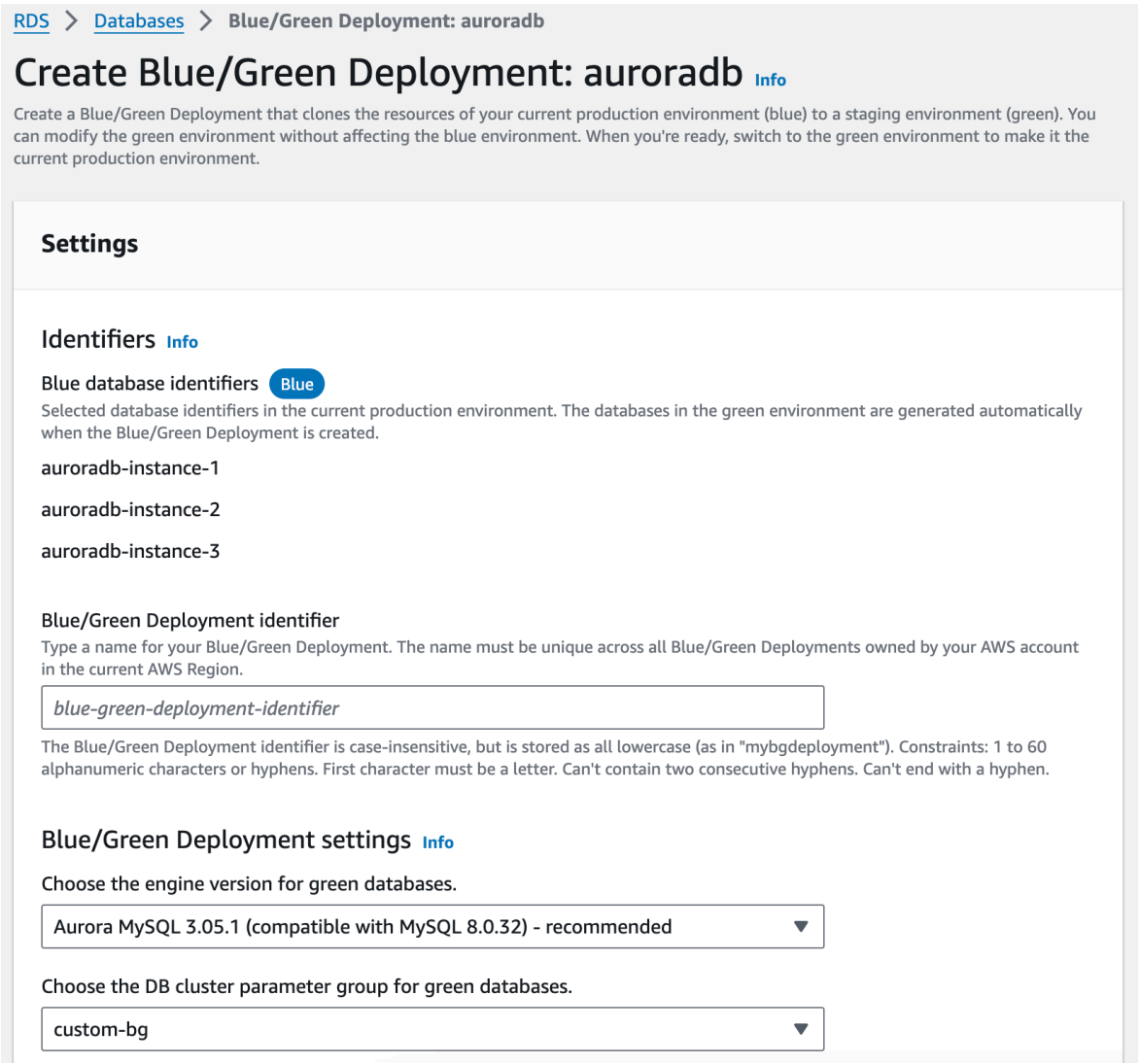
您可以使用 AWS 管理主控台、 AWS CLI或 RDS API 建立藍/綠部署。

主控台

建立藍/綠部署

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)，然後選擇您要將其複製到綠色環境的資料庫叢集。
3. 選擇動作、建立藍/綠部署。

   建立藍/綠部署頁面即會出現。



4. 檢閱藍色資料庫識別符。確定其符合您在藍色環境中預期的資料庫執行個體。如果不符，請選擇 Cancel (取消)。
5. 針對藍/綠部署名稱，輸入藍/綠部署的名稱。

6. 在其餘區段中，指定綠色環境的設定。如需每項設定的相關資訊，請參閱 the section called “可用設定”。

   您可以在部署之後，於綠色環境中對資料庫進行其他修改。

7. 選擇建立。

AWS CLI

若要使用 建立藍/綠部署 AWS CLI，請使用 create-blue-green-deployment 命令。如需所有可用選項的資訊，請參閱 the section called “可用設定”。

Example

針對 Linux、macOS 或 Unix：

```
aws rds create-blue-green-deployment \
    --blue-green-deployment-name aurora-blue-green-deployment \
    --source arn:aws:rds:us-east-2:123456789012:cluster:auroradb \
    --target-engine-version 8.0 \
    --target-db-cluster-parameter-group-name mydbclusterparametergroup
```

在 Windows 中：

```
aws rds create-blue-green-deployment ^
    --blue-green-deployment-name aurora-blue-green-deployment ^
    --source arn:aws:rds:us-east-2:123456789012:cluster:auroradb ^
    --target-engine-version 8.0 ^
    --target-db-cluster-parameter-group-name mydbclusterparametergroup
```

RDS API

若要使用 Amazon RDS API 建立藍/綠部署，請使用 `CreateBlueGreenDeployment` 操作。如需每個選項的詳細資訊，請參閱the section called “可用設定”。

建立藍/綠部署的設定

下表說明您在建立藍/綠部署時可以選擇的設定。如需 AWS CLI 選項的詳細資訊，請參閱 create-blue-green-deployment。如需 RDS API 參數的詳細資訊，請參閱 CreateBlueGreenDeployment。

主控台設定	設定說明	CLI 選項和 RDS API 參數

藍/綠部署識別符	藍/綠部署的名稱。	CLI 選項：
		`--blue-green-deployment-name`
		API 參數：
		`BlueGreenDeploymentName`

藍色資料庫識別符	您想要複製到綠色環境之叢集的識別符。使用 CLI 或 API 時，請指定叢集 Amazon Resource Name (ARN)。	CLI 選項：
		`--source`
		API 參數：
		`Source`

綠色資料庫的資料庫叢集參數群組	在綠色環境中要與資料庫相關聯的參數群組。	CLI 選項：
		`--target-db-cluster-parameter-group-name`
		API 參數：
		`TargetDBClusterParameterGroupName`

綠色資料庫的引擎版本	將綠色環境中的資料庫叢集升級至指定的資料庫引擎版本。
	如果您選擇 Aurora PostgreSQL 資料庫叢集，請檢閱並確認邏輯複寫限制。如需詳細資訊，請參閱the section called “藍/綠部署的邏輯複寫特定限制”。
		CLI 選項：
		`--target-engine-version`
		RDS API 參數：
		`TargetEngineVersion`

在 Amazon Aurora 中檢視藍/綠部署

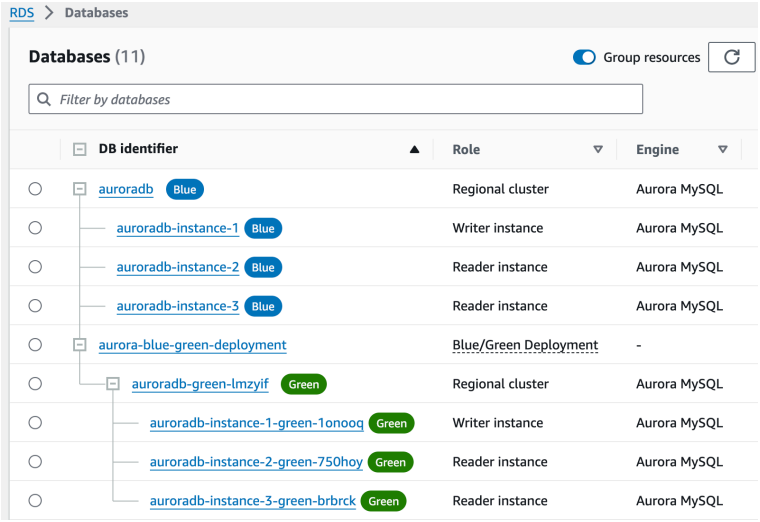
您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 檢視藍/綠部署的詳細資訊。

您也可以檢視和訂閱事件，以取得藍/綠部署的相關資訊。如需更多詳細資訊，請參閱 藍/綠部署事件。

主控台

檢視藍/綠部署的詳細資訊

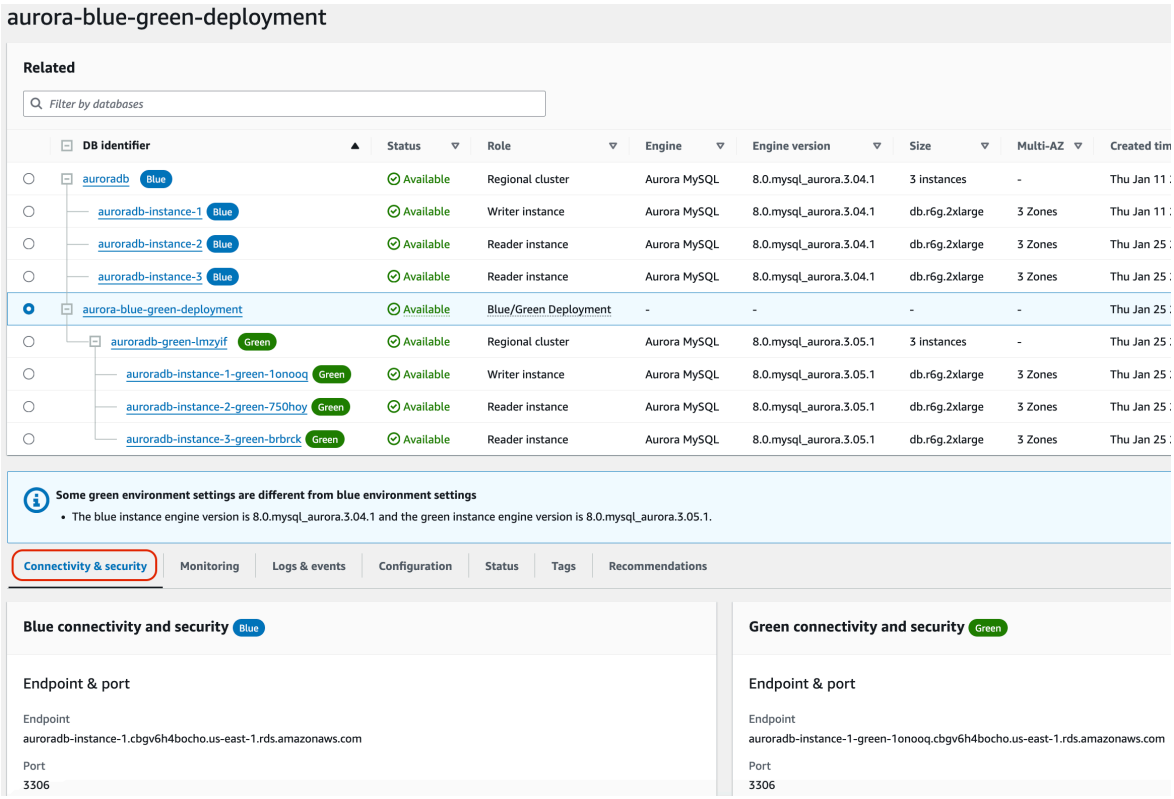
1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)，然後在清單中尋找藍/綠部署。



   藍/綠部署的 Role (角色) 值是 Blue/Green Deployment (藍/綠部署)。
3. 選擇您要檢視以顯示其詳細資訊的藍/綠部署名稱。

   每個索引標籤都有藍色部署的區段，以及綠色部署的區段。例如，在組態索引標籤上，如果您要升級綠色環境中的資料庫引擎版本，則資料庫引擎版本在藍色環境和綠色環境中可能會有所不同。

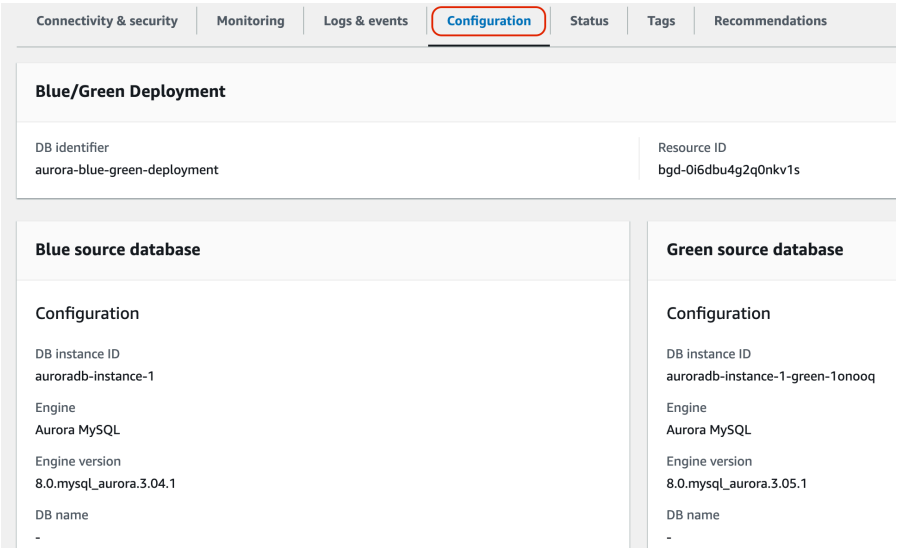
   下圖顯示連線和安全索引標籤的範例：



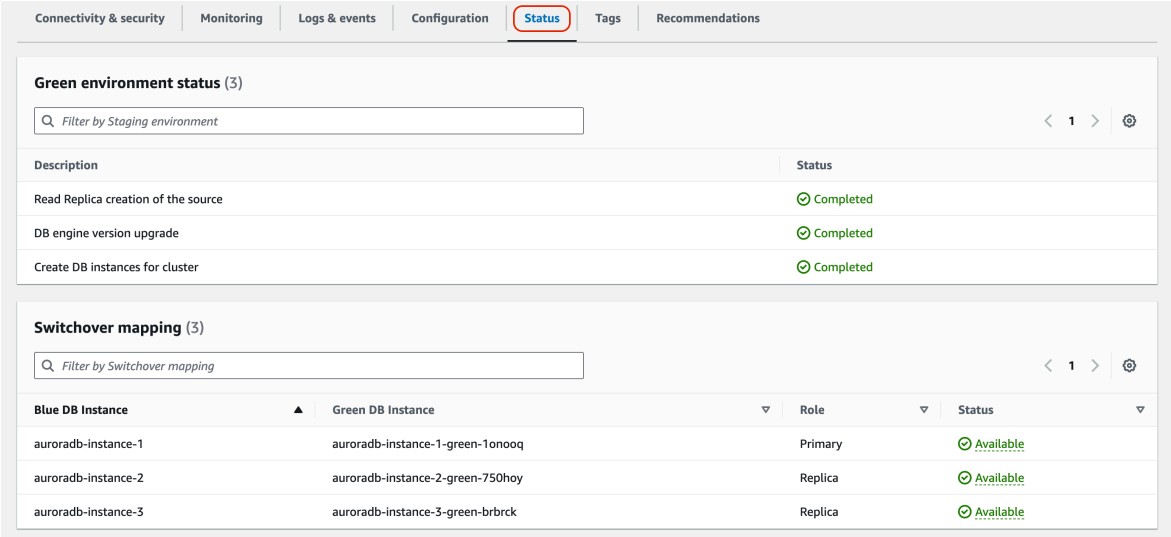
連線與安全性索引標籤也包含名為複寫的區段，其中顯示目前的複寫狀態，以及藍色環境與綠色環境之間的複本延遲。如果複寫狀態為 `Replicating`，則表示藍/綠部署成功複寫。

對於 Aurora PostgreSQL 藍/綠部署 ()，如果您在藍色環境中進行不支援的 DDL 或大型物件變更，則複寫狀態可能會變更為 `Replication degraded`。如需更多詳細資訊，請參閱 the section called “藍/綠部署的邏輯複寫特定限制”。

下圖顯示組態索引標籤的範例：



下圖顯示狀態索引標籤的範例：



AWS CLI

若要使用 AWS CLI 檢視藍/綠部署的詳細資訊，請使用 describe-blue-green-deployments 命令。

Example透過篩選藍/綠部署的名稱來檢視其詳細資訊

使用 describe-blue-green-deployments 命令時，您可以依 `--blue-green-deployment-name` 篩選。

下列範例顯示藍/綠部署 (名為 *my-blue-green-deployment*) 的詳細資訊。

```
aws rds describe-blue-green-deployments \
  --filters Name=blue-green-deployment-name,Values=my-blue-green-deployment
```

Example透過指定藍/綠部署的識別符來檢視其詳細資訊

使用 describe-blue-green-deployments 命令時，您可以指定 `--blue-green-deployment-identifier` 選項。

下列範例顯示識別符為 *bgd-1234567890abcdef* 的藍/綠部署的詳細資訊。

```
aws rds describe-blue-green-deployments \
  --blue-green-deployment-identifier bgd-1234567890abcdef
```

RDS API

若要使用 Amazon RDS API 檢視藍/綠部署的詳細資訊，請使用 `DescribeBlueGreenDeployments` 操作並指定 `BlueGreenDeploymentIdentifier`。

在 Amazon Aurora 中切換藍/綠部署

轉換會在綠色環境中將資料庫叢集 (包括其資料庫執行個體) 轉換為生產資料庫叢集。切換前，生產流量會路由到藍色環境中的叢集。切換後，生產流量會路由到綠色環境中的資料庫叢集。

轉換藍/綠部署與在藍/綠部署中提升綠色資料庫叢集不同。如果您透過從動作功能表中選擇提升，手動提升綠色資料庫叢集，則藍色和綠色環境之間的複寫會中斷，且藍/綠部署會進入無效組態的狀態。

主題

- 切換逾時
- 切換防護機制
- 切換動作
- 切換最佳實務
- 在切換前驗證 CloudWatch 指標
- 轉換前監控複本延遲
- 切換藍/綠部署
- 切換後

切換逾時

您可以指定介於 30 秒與 3,600 秒(一小時) 之間的切換逾時期間。如果切換所花費的時間超過指定的持續時間，則會復原任何變更，且不會對任一環境進行任何變更。預設逾時期間為 300 秒 (五分鐘)。

切換防護機制

當您開始切換時，Amazon RDS 會執行一些基本檢查，以測試藍色和綠色環境是否準備好進行切換。這些檢查稱為切換防護機制。如果環境還沒有做好準備，這些切換防護機制可防止切換。因此，它們可避免超過預期的停機時間，並防止若切換開始，可能導致藍色和綠色環境之間遺失資料。

Amazon RDS 會在綠色環境上執行下列防護機制檢查：

- 複寫運作狀態 – 檢查綠色資料庫叢集複寫狀態是否良好。綠色資料庫叢集是藍色資料庫叢集的複本。
- 複寫延遲 – 檢查綠色資料庫叢集的複本延遲是否在轉換的允許限制內。允許的限制是以指定的逾時期間為基礎。複本延遲指出綠色資料庫叢集落後於其藍色資料庫叢集多遠。如需詳細資訊，請參閱the section called “轉換前監控複本延遲”。
- 作用中寫入 – 確定綠色資料庫叢集上沒有作用中寫入。

Amazon RDS 會在藍色環境上執行下列防護機制檢查：

- 外部複寫：對於 Aurora PostgreSQL，確保藍色環境不是自我管理的邏輯來源 (發布者) 或複本 (訂閱用戶)。若是如此，我們建議您捨棄藍色環境中所有資料庫的自我管理複寫槽和訂閱，繼續進行轉換，然後將其重新建立以繼續複寫。對於 Aurora MySQL，檢查藍色資料庫是否不是外部 binlog 複本。如果是，請確定其不是主動複寫。
- 長時間執行的作用中寫入 – 確定藍色資料庫叢集上沒有長時間執行的作用中寫入，因為這些寫入可能會增加複本延遲。
- 長時間執行的 DDL 陳述式 – 確定藍色資料庫叢集上沒有長時間執行的 DDL 陳述式，因為這些陳述式可能會增加複本延遲。
- 不支援的 PostgreSQL 變更：對於 Aurora PostgreSQL，請確定沒有任何 DDL 變更，也未在藍色環境執行大型物件的新增或修改。如需詳細資訊，請參閱the section called “藍/綠部署的邏輯複寫特定限制”。

  如果 Amazon RDS 偵測到不支援的 PostgreSQL 變更，其會將複寫狀態變更為 `Replication degraded`，並通知您無法對藍/綠部署進行轉換。若要繼續進行轉換，建議您刪除並重新建立藍/綠部署和所有綠色資料庫。若要這樣做，請選擇動作、刪除綠色資料庫。

切換動作

當您切換藍/綠部署時，RDS 會執行下列動作：

1. 執行防護機制檢查，以驗證藍色和綠色環境是否已準備好進行切換。
2. 在這兩個環境中停止資料庫叢集上的新寫入操作。
3. 捨棄與這兩個環境中資料庫執行個體的連線，而且不允許新的連線。
4. 等待複寫在綠色環境中趕上進度，以便綠色環境與藍色環境同步。
5. 在這兩個環境中重新命名資料庫叢集和資料庫執行個體。

RDS 會重新命名綠色環境中的資料庫叢集和資料庫執行個體，以符合藍色環境中對應的資料庫叢集和資料庫執行個體。例如，假設藍色環境中資料庫執行個體的名稱為 mydb。也會假設綠色環境中對應資料庫執行個體的名稱為 mydb-green-abc123。在切換期間，綠色環境中資料庫執行個體的名稱會變更為 mydb。

RDS 會重新命名藍色環境中的資料庫叢集和資料庫執行個體，方法是將 -old*n* 附加至目前名稱 (其中 *n* 是數字)。例如，假設藍色環境中資料庫執行個體的名稱為 mydb。切換後，資料庫執行個體名稱可能是 mydb-old1。

RDS 也會重新命名綠色環境中的端點，以符合藍色環境中的對應端點，以便不需要應用程式變更。

6. 允許連線至這兩個環境中的資料庫。
7. 在新的生產環境中允許資料庫叢集上的寫入操作。

   轉換後，之前的生產資料庫叢集只允許讀取操作。即使您在資料庫叢集上啟用寫入，其仍會保持唯讀狀態，直到您刪除藍/綠部署為止。

您可以使用 Amazon EventBridge 監控切換狀態。如需詳細資訊，請參閱the section called “藍/綠部署事件”。

在切換期間，藍色環境中的標籤會取代綠色環境中資源上的所有標籤。在此過程中，您直接新增至綠色環境資源的任何標籤都會遭到覆寫。如需標籤的詳細資訊，請參閱標記 Amazon Aurora 和 Amazon RDS 資源。

如果切換開始，然後在完成前由於任何原因而停止，則會復原任何變更，且不會對任一環境進行任何變更。

切換最佳實務

在您轉換之前，我們強烈建議您完成下列任務，以遵守最佳實務：

- 徹底測試綠色環境中的資源。確保它們正常有效地執行。
- 監控相關 Amazon CloudWatch 指標。如需詳細資訊，請參閱the section called “在切換前驗證 CloudWatch 指標”。
- 識別切換的最佳時間。

  切換期間，會中斷這兩種環境中資料庫的寫入。識別生產環境上流量最低的時間。長時間執行的交易 (例如作用中 DDL) 可能會增加您的切換時間，因而延長生產工作負載的停機時間。

如果您的資料庫叢集和資料庫執行個體上有大量連線，請考慮在切換藍/綠部署之前，手動將其減少到應用程式所需的最低數量。達成此目標的方法之一是建立一項指令碼，用以監控藍/綠部署狀態，並在偵測到狀態變更至 SWITCHOVER_IN_PROGRESS 時開始清除連線。

- 確定這兩個環境中的資料庫叢集和資料庫執行個體處於 Available 狀態。
- 確定綠色環境中的資料庫叢集運作良好且複寫中。
- 請確定您的網路和用戶端組態不會將 DNS 快取存留時間 (TTL) 增加到五秒以上，這是 Aurora DNS 區域的預設值。否則，應用程式會在切換後繼續將寫入流量傳送至藍色環境。
- 對於 Aurora PostgreSQL 藍/綠部署，請執行下列動作：
  - 檢閱邏輯複寫限制，並在轉換之前採取任何必要的動作。如需詳細資訊，請參閱the section called “藍/綠部署的邏輯複寫特定限制”。
  - 執行 ANALYZE 操作以重新整理 pg_statistics 資料表。這可降低轉換後發生效能問題的風險。
  - 啟動藍/綠部署切換之前，請確認您的應用程式不會覆寫工作階段層級的 default_transaction_read_only 參數。在切換期間，此參數會在綠色環境寫入器on上設定為 ，以防止寫入，直到提升完成為止。如果您的應用程式或交易將此組態覆寫為 off，您的應用程式可能會在切換過程中將資料寫入綠色環境。如果轉換必須轉返，藍色環境中無法使用這些寫入，要求您手動解決資料不一致的問題。我們強烈建議稽核您的應用程式查詢，以確保它們遵守default_transaction_read_only設定，然後再繼續切換。

> **Note**
>
> 切換期間，您無法修改切換中包含的任何資料庫叢集。

在切換前驗證 CloudWatch 指標

轉換藍/綠部署之前，建議您檢查 Amazon CloudWatch 中下列指標的值。

- DatabaseConnections – 使用此指標估計藍/綠部署的活動層級，並在切換前確認該值處於部署的可接受層級。如果績效詳情已開啟，則 DBLoad 為更準確的指標。
- ActiveTransactions – 若在任何資料庫執行個體中，資料庫參數群組中的 innodb_monitor_enable 設為 all，請使用此指標來查看是否有大量作用中交易可能封鎖切換。

如需詳細資訊，請參閱the section called “Aurora 的 CloudWatch 指標”。

轉換前監控複本延遲

在轉換藍/綠部署之前，請確定複本延遲接近零，以縮短停機時間。

Aurora MySQL

對於 MySQL 藍/綠部署，請檢查綠色環境中的 AuroraBinlogReplicaLag CloudWatch 指標，以識別目前的複本延遲。如需詳細資訊，請參閱the section called “診斷和解決僅供讀取複本之間的延遲”。

Aurora PostgreSQL

對於的 PostgreSQL 藍/綠部署，請檢查藍色環境中的 OldestReplicationSlotLag CloudWatch 指標，以識別目前的複本延遲。如需詳細資訊，請參閱the section called “Amazon Aurora 的執行個體層級指標”。

此外，您可以在藍色環境中執行下列 SQL 查詢：

```
SELECT slot_name,
       confirmed_flush_lsn as flushed,
       pg_current_wal_lsn(),
       (pg_current_wal_lsn() - confirmed_flush_lsn) AS lsn_distance
FROM pg_catalog.pg_replication_slots
WHERE slot_type = 'logical';

slot_name        |    flushed    | pg_current_wal_lsn | lsn_distance
-----------------+---------------+--------------------+------------
logical_replica1 | 47D97/CF32980 | 47D97/CF3BAC8      | 37192
```

confirmed_flush_lsn 代表傳送至複本的最後一個日誌序號 (LSN)。pg_current_wal_lsn 代表資料庫現在所在的位置。lsn_distance 為 0 表示複本已跟上進度。

切換藍/綠部署

您可以使用 AWS CLI、 或 RDS API AWS 管理主控台切換藍/綠部署。

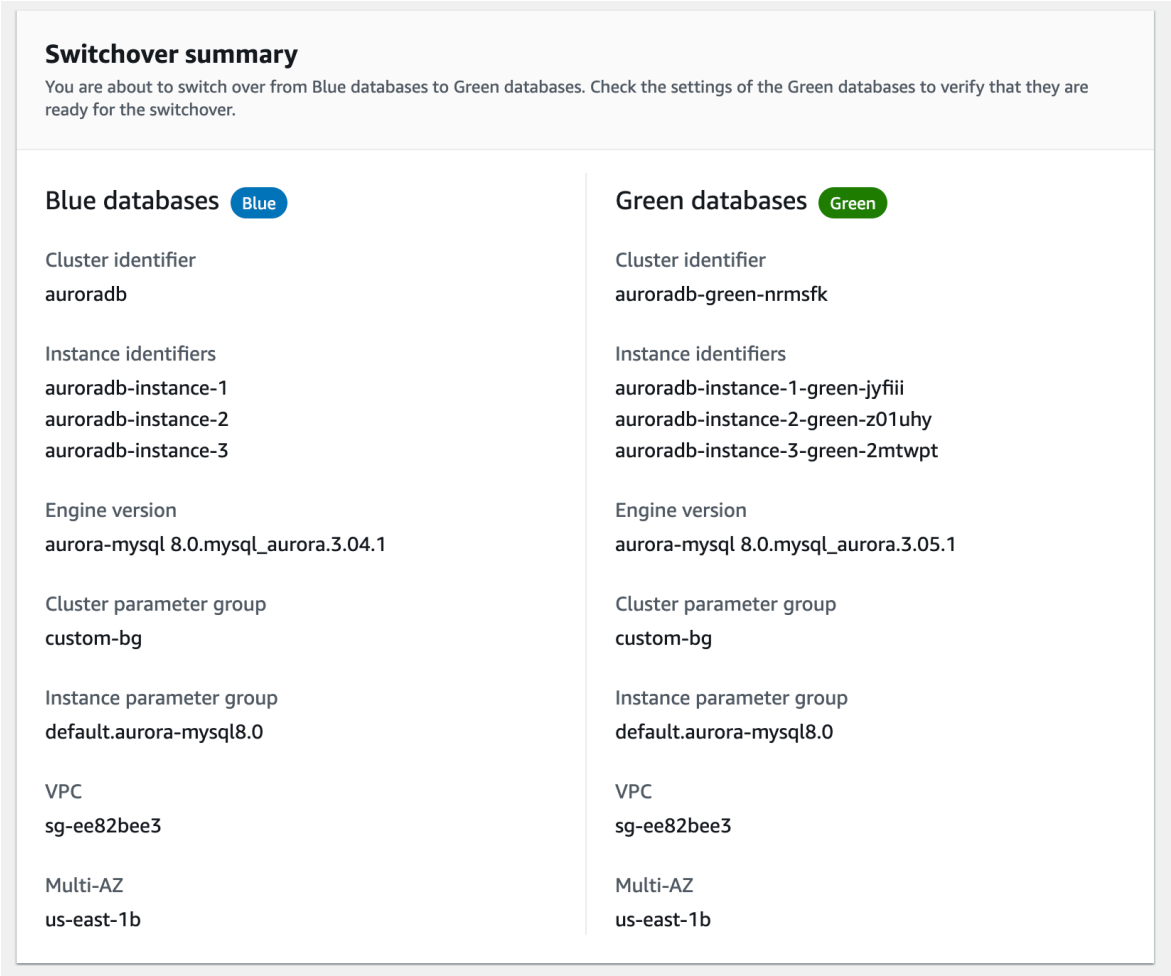
主控台

切換藍/綠部署

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中選擇 Databases (資料庫)，然後選擇您要切換的藍/綠部署。

3. 針對 Actions (動作)，選擇 Switch over (切換)。

   Switch over (切換) 頁面即會出現。



4. 在 Switch over (切換) 頁面上，檢閱切換摘要。請確定這兩個環境中的資源符合您預期的資源。如果不符，請選擇 Cancel (取消)。
5. 針對逾時設定，輸入轉換的時間限制。
6. 如果您的叢集正在執行 Aurora PostgreSQL，請檢閱並確認轉換前建議。如需詳細資訊，請參閱the section called “藍/綠部署的邏輯複寫特定限制”。
7. 選擇 Switch over (切換)。

AWS CLI

若要使用 切換藍/綠部署 AWS CLI，請使用 switchover-blue-green-deployment 命令搭配下列選項：

- `--blue-green-deployment-identifier`：指定藍/綠部署的資源 ID。

- `--switchover-timeout` – 指定切換的時間限制，以秒為單位。預設值為 300。

Example切換藍/綠部署

對於 Linux、macOS 或 Unix：

```
aws rds switchover-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --switchover-timeout 600
```

在 Windows 中：

```
aws rds switchover-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --switchover-timeout 600
```

RDS API

若要使用 Amazon RDS API 切換藍/綠部署，請搭配下列參數使用 SwitchoverBlueGreenDeployment 操作：

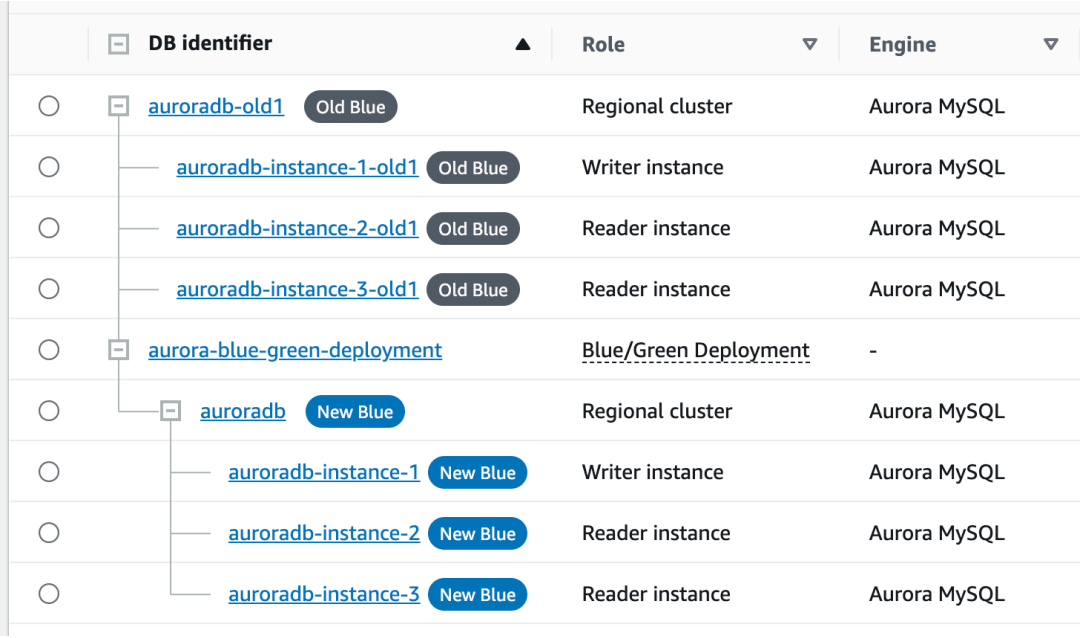
- `BlueGreenDeploymentIdentifier`：指定藍/綠部署的資源 ID。
- `SwitchoverTimeout` – 指定切換的時間限制，以秒為單位。預設值為 300。

切換後

切換後，先前藍色環境中的資料庫叢集和資料庫執行個體會保留下來。標準成本適用於這些資源。藍色和綠色環境之間的複寫和二進位記錄會停止。

RDS 會重新命名藍色環境中的資料庫叢集和資料庫執行個體，方法是將 `-old`*n* 附加至目前資源名稱 (其中 *n* 是數字)。藍色資料庫叢集會強制進入唯讀狀態。即使您啟用寫入操作，其仍會保持唯讀狀態，直到您刪除藍/綠部署為止。RDS 會在綠色環境 `-new`*n* 中為資料庫叢集和資料庫執行個體命名。

如果您刪除藍/綠部署資源，RDS 會保留 `-old`*n* 和 `-new`*n* 資源。



DB identifier	Role	Engine
auroradb-old1 Old Blue	Regional cluster	Aurora MySQL
auroradb-instance-1-old1 Old Blue	Writer instance	Aurora MySQL
auroradb-instance-2-old1 Old Blue	Reader instance	Aurora MySQL
auroradb-instance-3-old1 Old Blue	Reader instance	Aurora MySQL
aurora-blue-green-deployment	Blue/Green Deployment	-
auroradb New Blue	Regional cluster	Aurora MySQL
auroradb-instance-1 New Blue	Writer instance	Aurora MySQL
auroradb-instance-2 New Blue	Reader instance	Aurora MySQL
auroradb-instance-3 New Blue	Reader instance	Aurora MySQL

更新消費者的父節點

RDS 提供全受管僅供讀取複本。不過，其也提供設定自我管理複本 (也稱為外部複本) 的選項。外部複本可讓您使用第三方資源作為複寫目標。

在您轉換 Aurora MySQL 藍/綠部署之後，如果藍色資料庫叢集在轉換之前有任何外部複本或二進位日誌消費者，您必須在轉換後更新其父節點，以維持複寫持續性。

更新父節點

1. 轉換後，先前在綠色環境中的寫入器資料庫執行個體會發出事件，其中包含主日誌檔案名稱和主日誌位置。若要尋找事件，請瀏覽至 RDS 主控台，然後從左側導覽窗格中選擇事件。
2. 在轉換之前，依事件進行篩選，在這些事件中來源為舊的綠色寫入器資料庫執行個體的名稱。
3. 尋找包含二進位日誌座標的事件。事件訊息類似於：`Binary log coordinates in green environment after switchover: file mysql-bin-changelog.000003 and position 40134574`。
4. 確保消費者或複本已套用舊的藍色環境中的所有二進位日誌。然後，使用提供的二進位日誌座標，在消費者端繼續複寫。例如，如果在 EC2 執行 MySQL 複本，則可以使用下列命令：

   MySQL 8.0.22 和更低的主要和次要版本

   ```
   CHANGE MASTER TO MASTER_HOST='{new-writer-endpoint}', MASTER_LOG_FILE='mysql-bin-changelog.000003', MASTER_LOG_POS=40134574;
   ```

MySQL 8.0.23 和更新的主要和次要版本

```
CHANGE REPLICATION SOURCE TO SOURCE_HOST='{new-writer-endpoint}',
 SOURCE_LOG_FILE='mysql-bin-changelog.000003', SOURCE_LOG_POS=40134574;
```

刪除 Amazon Aurora 中的藍/綠部署

您可以在切換藍/綠部署之前或之後將其刪除。

若您在切換藍/綠部署之前將其刪除，Amazon RDS 會選擇性地刪除綠色環境中的資料庫叢集：

- 如果您選擇刪除綠色環境 (`--delete-target`) 中的資料庫叢集，其必須關閉刪除保護。
- 如果未刪除綠色環境 (`--no-delete-target`) 中的資料庫叢集，則會保留該叢集，但們不再是藍/綠部署的一部分。對於 Aurora MySQL，複寫會在環境之間繼續。對於 Aurora PostgreSQL，綠色環境會提升為獨立環境，因此複寫會停止。

切換後，刪除綠色資料庫的選項無法在主控台中使用。當您使用 刪除藍/綠部署時 AWS CLI，如果部署狀態為 ，則無法指定 `--delete-target`選項`SWITCHOVER_COMPLETED`。

> **Important**
>
> 刪除藍/綠部署之後，RDS 會從先前的生產資料庫叢集中移除唯讀保護。如果資料庫叢集的 `read_only` 參數已停用，則會開始再次允許寫入操作。

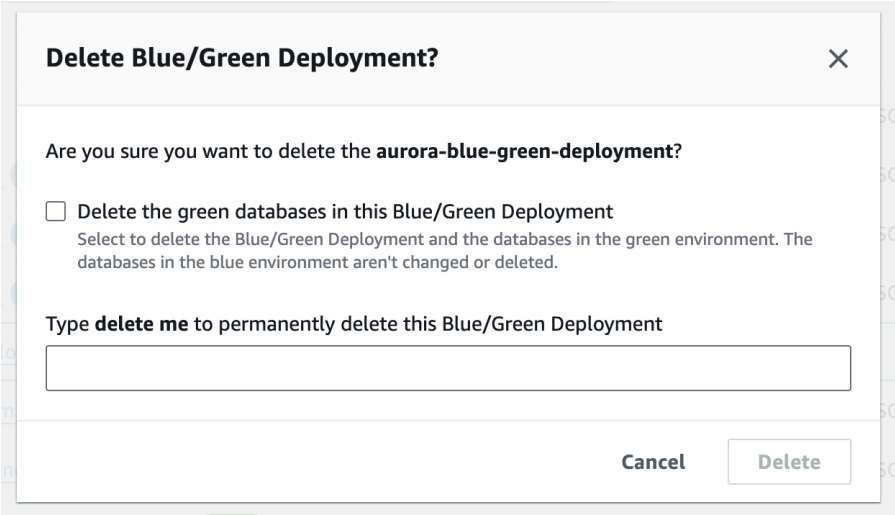
您可以使用 AWS 管理主控台 AWS CLI、 或 RDS API 刪除藍/綠部署。

主控台

刪除藍/綠部署

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中選擇 Databases (資料庫)，然後選擇您要刪除的藍/綠部署。
3. 對於 Actions (動作)，請選擇 Delete (刪除)。

   刪除藍/綠部署？視窗即會出現。



若要刪除綠色資料庫，請選取刪除此藍/綠部署中的綠色資料庫。

4. 在方塊中輸入 **`delete me`**。
5. 選擇 刪除。

AWS CLI

若要使用 刪除藍/綠部署 AWS CLI，請使用 delete-blue-green-deployment 命令搭配下列選項：

- `--blue-green-deployment-identifier`：要刪除的藍/綠部署資源 ID。
- `--delete-target` – 指定刪除綠色環境中的資料庫叢集。如果藍/綠部署的狀態為 `SWITCHOVER_COMPLETED`，則您無法指定此選項。
- `--no-delete-target` – 指定保留綠色環境中的資料庫叢集。

Example在綠色環境中刪除藍/綠部署和資料庫叢集

針對 Linux、macOS 或 Unix：

```
aws rds delete-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --delete-target
```

在 Windows 中：

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
```

```
    --delete-target
```

Example刪除藍/綠部署，但將資料庫叢集保留在綠色環境中

針對 Linux、macOS 或 Unix：

```
aws rds delete-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --no-delete-target
```

在 Windows 中：

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --no-delete-target
```

RDS API

若要使用 Amazon RDS API 刪除藍/綠部署，請搭配下列參數使用 DeleteBlueGreenDeployment 操作：

- BlueGreenDeploymentIdentifier：要刪除的藍/綠部署資源 ID。
- DeleteTarget – 指定 TRUE 以刪除綠色環境中的資料庫叢集，或指定 FALSE 以保留它。如果藍/綠部署的狀態為 SWITCHOVER_COMPLETED，則不能為 TRUE。

備份與還原 Amazon Aurora 資料庫叢集

這些主題提供備份並還原 Amazon Aurora 資料庫叢集的詳細資訊。

> **Tip**
>
> Aurora 高可用性功能和自動備份功能有助於保護您的資料安全，而不需要進行大量設定。在執行備份策略之前，請先了解 Aurora 如何維護多個資料複本，並協助您跨多個資料庫執行個體和 AWS 區域存取這些複本。如需詳細資訊，請參閱 Amazon Aurora 的高可用性。

主題

- 備份與還原 Aurora 資料庫叢集的概觀
- 保留自動備份
- 了解 Amazon Aurora 備份儲存體用量
- 建立資料庫叢集快照
- 從資料庫叢集快照還原
- 資料庫叢集快照複製
- 共享資料庫叢集快照
- 將資料庫叢集資料匯出至 Amazon S3
- 將資料庫叢集快照資料匯出至 Amazon S3
- 將資料庫叢集還原至指定時間
- 刪除資料庫叢集快照
- 教學：從資料庫叢集快照還原 Amazon Aurora 資料庫叢集

備份與還原 Aurora 資料庫叢集的概觀

以下主題說明 Aurora 備份和如何還原 Aurora 資料庫叢集。

內容

- 備份
  - 使用 AWS Backup
- 備份時段
- 還原資料
- Aurora 的資料庫複製
- 恢復

備份

Aurora 會自動備份您的叢集磁碟區，並在備份保留期間保留還原資料。Aurora 自動備份具有連續性和增量性，因此，您可以快速還原到備份保留期內的任何時間點。寫入備份資料時不會影響資料庫服務的效能或中斷服務。您可以在建立或修改或還原資料庫叢集時，指定 1-35 天的備份保留期。Aurora 自動備份存放在 Amazon S3 中。如需保留自動備份的詳細資訊，請參閱 保留自動備份。

如果想要保留資料超過備份保留期，您可以在叢集磁碟區中建立資料快照。Aurora 資料庫叢集快照不會過期。您可以從快照建立新的資料庫叢集。如需詳細資訊，請參閱建立資料庫叢集快照。

在還原操作期間，您可以選擇為 Amazon Aurora 資料庫叢集指定備份保留期間。當您未明確設定此值時，還原的叢集會從來源快照或叢集繼承備份保留期間。請注意，此繼承行為對於還原操作而言是唯一的 - 建立新叢集時，系統會改為套用預設保留期間。

> **Note**
>
> - 無論資料庫叢集的建立方式為何，Amazon Aurora 資料庫叢集的預設備份保留期為一天。
> - 您無法停用 Aurora 上的自動備份。Aurora 的備份保留期是由資料庫叢集管理。

備份儲存體成本取決於您保留的 Aurora 備份和快照資料量，以及要保留的時間長度。如需與 Aurora 備份和快照相關聯的儲存體的資訊，請參閱 了解 Amazon Aurora 備份儲存體用量。如需 Aurora 備份儲存體定價的詳細資訊，請參閱 Amazon RDS for Aurora 定價。刪除與快照相關聯的 Aurora 叢集之後，存放該快照會衍生 Aurora 的標準備份儲存體費用。

使用 AWS Backup

您可以使用 AWS Backup 來管理 Amazon Aurora 資料庫叢集的備份。

由 管理的快照 AWS Backup 會被視為手動資料庫叢集快照，但不計入 Aurora 的資料庫叢集快照配額。使用 建立的快照 AWS Backup 具有 的名稱`awsbackup:job-`*`AWS-Backup-job-number`*。如需的詳細資訊 AWS Backup，請參閱 AWS Backup 開發人員指南。

您也可以使用 AWS Backup 來管理 Amazon Aurora 資料庫叢集的自動備份。如果您的資料庫叢集與中的備份計劃相關聯 AWS Backup，您可以使用該備份計劃進行point-in-time復原。由 AWS Backup 管理的自動 (連續) 備份，其名稱會包含 `continuous:cluster-`*`AWS-Backup-job-number`*。如需詳細資訊，請參閱使用 AWS Backup 將資料庫叢集還原至指定的時間。

備份時段

自動備份會每天在偏好的備份時段內執行。如果備份需要的時間超過所分配的備份時段，備份會在時段結束後繼續執行直到完成。備份時段不可與每週資料庫叢集維護時段重疊。

Aurora 自動備份是持續且增量的，但是備份時段是用來建立在備份保留期間內保留的每日系統備份。您可以複製備份檔案，在超出保留期間後仍然保留。

> **Note**
>
> 當您使用 建立資料庫叢集時 AWS 管理主控台，您無法指定備份時段。不過，您可以在使用 AWS CLI 或 RDS API 建立資料庫叢集時指定備份時段。

當您建立資料庫叢集時，若未指定偏好的備份時段，Aurora 將會指派預設的 30 分鐘備份時段。此時段是從每個時段的 8 小時時段中隨機選取 AWS 區域。下表列出指派 AWS 區域 預設備份時段的每個時間範圍。

區域名稱	區域	時間區塊
美國東部 (維吉尼亞北部)	us-east-1	03:00–11:00 UTC
美國東部 (俄亥俄)	us-east-2	上午 3 時至 11 時 (UTC)
美國西部 (加利佛尼亞北部)	us-west-1	上午 6 時至下午 2 時 (UTC)

區域名稱	區域	時間區塊
美國西部 (奧勒岡)	us-west-2	上午 6 時至下午 2 時 (UTC)
非洲 (開普敦)	af-south-1	上午 3 時至 11 時 (UTC)
亞太區域 (香港)	ap-east-1	上午 6 時至下午 2 時 (UTC)
亞太地區 (海德拉巴)	ap-south-2	06:30–14:30 UTC
亞太地區 (雅加達)	ap-southeast-3	08:00–16:00 UTC
亞太地區 (馬來西亞)	ap-southeast-5	09:00–17:00 UTC
亞太地區 (墨爾本)	ap-southeast-4	上午 11 時至下午 7 時 (UTC)
亞太區域 (孟買)	ap-south-1	下午 4 時 30 分至上午 12 時 30 分 (UTC)
亞太區域 (紐西蘭)	ap-southeast-6	下午 1 時至下午 9 時 (UTC)
亞太地區 (大阪)	ap-northeast-3	上午 12 時至上午 8 時 (UTC)
亞太區域 (首爾)	ap-northeast-2	下午 1 時至 9 時 (UTC)
亞太區域 (新加坡)	ap-southeast-1	下午 2 時至 10 時 (UTC)
亞太區域 (雪梨)	ap-southeast-2	中午 12 時至下午 8 時 (UTC)
亞太區域 (台北)	ap-east-2	9:00–17:00 UTC
亞太區域 (泰國)	ap-southeast-7	8:00–16:00 UTC
亞太區域 (東京)	ap-northeast-1	下午 1 時至 9 時 (UTC)
加拿大 (中部)	ca-central-1	03:00–11:00 UTC
加拿大西部 (卡加利)	ca-west-1	18:00–02:00 UTC
中國 (北京)	cn-north-1	上午 6 時至下午 2 時 (UTC)
中國 (寧夏)	cn-northwest-1	上午 6 時至下午 2 時 (UTC)

區域名稱	區域	時間區塊
歐洲 (法蘭克福)	eu-central-1	下午 8 時至上午 4 時 (UTC)
歐洲 (愛爾蘭)	eu-west-1	下午 10 時至上午 6 時 (UTC)
歐洲 (倫敦)	eu-west-2	22:00–06:00 UTC
歐洲 (米蘭)	eu-south-1	上午 2 時至 10 時 (UTC)
Europe (Paris)	eu-west-3	上午 7 時 29 分至下午 2 時 29 分 (UTC)
歐洲 (西班牙)	eu-south-2	上午 2 時至 10 時 (UTC)
歐洲 (斯德哥爾摩)	eu-north-1	下午 11 時至上午 7 時 (UTC)
歐洲 (蘇黎世)	eu-central-2	上午 2 時至 10 時 (UTC)
以色列 (特拉維夫)	il-central-1	03:00–11:00 UTC
墨西哥 (中部)	mx-central-1	19:00–03:00 UTC
Middle East (Bahrain)	me-south-1	上午 6 時至下午 2 時 (UTC)
中東 (阿拉伯聯合大公國)	me-central-1	上午 5 時至下午 1 時 (UTC)
南美洲 (聖保羅)	sa-east-1	下午 11 時至次日上午 7 時 (UTC)
AWS GovCloud （美國東部）	us-gov-east-1	下午 5 時至上午 1 時 (UTC)
AWS GovCloud （美國西部）	us-gov-west-1	上午 6 時至下午 2 時 (UTC)

還原資料

您可以從 Aurora 保留的備份資料或您已儲存的資料庫叢集快照來建立新的 Aurora 資料庫叢集，或是從保留的自動備份以復原資料。您可以利用從備份資料所建立之資料庫叢集的新副本，快速還原至備份

保留期之內的任何時間點。由於 Aurora 在備份保留期間內會自動連續以增量方式進行備份，意即您不需要經常建立資料的快照來改善還原時間。

資料庫叢集的最新的可還原時間是指可讓您還原資料庫叢集的最近時間點。對於使用中的資料庫叢集，通常是目前時間的 5 分鐘之內，對於保留的自動備份，通常是叢集刪除時間的 5 分鐘。

最早可還原時間指定您在備份保留期間之內可將叢集磁碟區還原到多久以前。

若要判斷資料庫叢集的最晚或最早可還原時間，請在 RDS 主控台尋找 `Latest restorable time` 或 `Earliest restorable time` 值。如需有關檢視這些值的詳細資訊，請參閱檢視 Amazon Aurora 的保留自動備份。

您可以檢查 `Latest restorable time` 和 `Earliest restorable time` 值，以判斷資料庫叢集還原何時完成。還原操作完成之前，這些值會傳回 NULL。如果 `Latest restorable time` 或 `Earliest restorable time` 傳回 NULL，則您無法要求備份或還原操作。

如需將資料庫叢集還原至指定時間的詳細資訊，請參閱 將資料庫叢集還原至指定時間。

Aurora 的資料庫複製

您也可以使用資料庫複製，將 Aurora 資料庫叢集的資料庫複製到新的資料庫叢集，而不是還原資料庫叢集快照。複製資料庫第一次建立時只使用最少的額外空間。無論在來源資料庫或複製資料庫上，只有在資料變更時才會複製資料。您可以從相同的資料庫叢集建立多個複製品，甚至從其他複製品建立更多複製品。如需詳細資訊，請參閱複製 Amazon Aurora 資料庫叢集的一個磁碟區。

恢復

Aurora MySQL 現在支援將資料庫叢集「倒轉」至特定時間，而不需從備份還原資料。如需更多詳細資訊，請參閱 恢復 Aurora 資料庫叢集。

保留自動備份

當您刪除已佈建或 Aurora Serverless v2 資料庫叢集時，您可以保留自動備份。這可讓您將資料庫叢集還原至備份保留期間內的特定時間點，即便叢集已遭到刪除。

保留的自動備份包含來自資料庫叢集的系統快照和交易日誌。它們還包括資料庫叢集屬性，例如：資料庫執行個體類別，這是將其還原到作用中叢集所需的屬性。

您可以使用 AWS 管理主控台、RDS API 和 AWS CLI 來還原或移除保留的自動備份。

> **Note**
>
> 您無法保留 Aurora Serverless v1 資料庫叢集的自動備份。

主題

- 保留期間
- 保留成本
- 防止自動刪除備份
- 限制
- 檢視 Amazon Aurora 的保留自動備份
- 刪除為 Amazon Aurora 保留的自動備份

保留期間

保留的自動備份中的系統快照和交易日誌會像在來源資料庫叢集中一樣過期。來源叢集保留期間的設定也會套用至自動備份。因為沒有為此叢集建立新的快照或日誌，保留的自動備份最後會完全過期。保留期間結束後，您可以繼續保留手動資料庫叢集快照，但所有自動備份都會過期。

您可以使用主控台、AWS CLI 或 RDS API，以移除保留的自動備份。如需更多詳細資訊，請參閱 刪除為 Amazon Aurora 保留的自動備份。

與保留的自動備份不同，最終快照不會過期。即使您保留自動備份，仍強烈建議您建立最終快照，因為保留的自動備份最後會過期。

保留成本

對於每個 Aurora 資料庫叢集，備份儲存體最高可達 Aurora 資料庫總儲存體的 100%，無需額外付費。如果您在刪除資料庫叢集之後保留自動備份，則最多一天內也不會收取額外費用。保留超過一天的備份需支付費用。

交易日誌或執行個體中繼資料不會產生額外費用。備份的所有其他定價規則適用於可還原的叢集。如需詳細資訊，請參閱 Amazon Aurora 定價頁面。

防止自動刪除備份

Amazon RDS 會在多種情況下刪除自動備份：

- 在其保留期結束時。
- 當您刪除資料庫叢集時。

如果想要讓自動備份保留期間更長，請複製自動備份來建立手動快照，即可保留到您刪除為止。如果手動快照超出預設儲存空間，則其可能產生 Amazon RDS 儲存成本。

如需複製資料庫叢集快照的詳細資訊，請參閱 資料庫叢集快照複製。

如需備份儲存成本的詳細資訊，請參閱 Amazon RDS 定價。

限制

保留的自動備份有下列限制：

- 在一個 AWS 區域 中，保留的自動備份最多 40 個。未包含在資料庫叢集的配額中。您可以同時為資料庫叢集建立最多達 40 個執行中的資料庫叢集、40 個執行中的資料庫執行個體，以及 40 個保留的自動備份。

  如需更多詳細資訊，請參閱 Amazon Aurora 中的配額。
- 保留的自動備份不含參數或選項群組的相關資訊。
- 您可以將已刪除的叢集還原到刪除當時的保留期間內的某個時間點。
- 您無法修改保留的自動備份，因為它包含您刪除來源叢集時存在的系統快照、交易日誌和資料庫叢集屬性。
- Aurora 資料庫叢集不支援跨區域自動備份複寫。Aurora 不支援自動將快照和交易日誌複寫到另一個 AWS 區域。若要跨區域進行災難復原，您必須手動將 Aurora 快照複製到所需的目的地區域。

檢視 Amazon Aurora 的保留自動備份

若要在 RDS 主控台中檢視保留的自動備份，請在導覽窗格中選擇自動備份，然後選擇保留。若要檢視與保留的自動備份相關聯的個別快照，請在導覽窗格中選擇 Snapshots (快照)。或者，您可以描述與保留的自動備份相關聯的個別快照。您可以從那裡的其中一個快照直接還原資料庫執行個體。

若要使用 AWS CLI 來描述保留的自動備份，請使用下列命令：

```
aws rds describe-db-cluster-automated-backups --db-cluster-resource-
id DB_cluster_resource_ID
```

若要使用 RDS API 來描述保留的自動備份，請呼叫 `DescribeDBClusterAutomatedBackups` 動作並搭配 `DbClusterResourceId` 參數。

刪除為 Amazon Aurora 保留的自動備份

不再需要保留的自動備份時，刪除即可。若要使用 AWS 管理主控台、AWS CLI 或 Amazon RDS API 刪除保留的自動備份，請使用下列程序。

主控台

刪除保留的自動備份

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Automated backups (自動備份)。
3. 選擇保留標籤。

4. 選擇您要刪除的已保留自動備份。
5. 對於 Actions (動作)，請選擇 Delete (刪除)。
6. 在確認頁面上，輸入 **`delete me`**，然後選擇 Delete (刪除)。

AWS CLI

您可以使用 AWS CLI 命令 delete-db-cluster-automated-backup，透過下列選項刪除保留的自動備份：

- `--db-cluster-resource-id` – 來源資料庫叢集的資源識別符。

  您可以執行 AWS CLI 命令 describe-db-cluster-automated-backups，刪除已保留自動備份的來源資料庫叢集的資源識別符。

Example

此範例會刪除具有資源 IDcluster-123ABCEXAMPLE 來源資料庫叢集保留的自動備份。

對於 Linux、macOS 或 Unix：

```
aws rds delete-db-cluster-automated-backup \
    --db-cluster-resource-id cluster-123ABCEXAMPLE
```

在 Windows 中：

```
aws rds delete-db-cluster-automated-backup ^
    --db-cluster-resource-id cluster-123ABCEXAMPLE
```

RDS API

您可以使用 Amazon RDS API 操作 DeleteDBClusterAutomatedBackup，透過下列參數刪除保留的自動備份：

- `DbClusterResourceId` – 來源資料庫叢集的資源識別符。

  您可以使用 Amazon RDS API 操作 DescribeDBClusterAutomatedBackups，尋找已保留自動備份的來源資料庫執行個體的資源識別符。

了解 Amazon Aurora 備份儲存體用量

Amazon Aurora 會維護兩種類型的備份：自動 (連續) 備份和快照。

自動備份儲存

叢集的自動 (連續) 備份會以增量方式儲存所指定保留期間內的所有資料庫變更，以能夠還原到該保留期間內的任何時間點。保留期間的範圍可為 1-35 天。自動備份是增量的，並根據還原到保留期間內任何時間所需的儲存量收費。

Aurora 也提供免費的備份使用量。這個免費的使用量等於最新的叢集磁碟區大小 (如 `VolumeBytesUsed` Amazon CloudWatch 指標所代表)。此數量會從計算出的自動備份使用量中扣除。對於保留期間僅為 1 天的自動備份，也不會對其收取任何費用。

例如，您的自動備份具有 7 天的保留期間，而您想要將叢集還原為 4 天前的狀態。Aurora 會使用存放在自動備份中的增量資料，重新建立叢集在四天前該確切時間的狀態。

自動備份會存放所有必要資訊，以能夠還原保留期間中任何時間點的叢集。這表示它會存放保留時段的所有變更，包括寫入新資訊或刪除現有資訊。對於發生許多變更的資料庫，自動備份的大小會隨著時間而增加。在資料庫停止發生變更之後，您可以預期自動備份的大小減少，因為先前存放的變更會結束保留時段。

自動備份的總計費用量永遠不會超過保留期間內累計的叢集磁碟區大小。例如，如果您的保留期間為 7 天，而您的叢集磁碟區每天為 100 GB，則計費的自動備份用量永遠不會超過 700 GB (100 GB * 7)。

快照儲存

資料庫叢集快照一律是完整備份，可在建立快照時擷取叢集磁碟區的大小。無論您是手動或透過 AWS Backups 計劃建立快照，Aurora 都會將其視為手動快照。Aurora 為自動備份保留期間內的快照提供無限制的免費儲存空間。手動快照超過此期間後，將會以每月每 GB 計費。除非您複製自動化系統快照，否則這些快照仍然免費。由於自動備份不涵蓋快照副本， AWS 因此 一律會為其計費。

如需 Aurora 備份的一般資訊，請參閱 備份。如需 Aurora 備份儲存體定價的相關資訊，請參閱 Amazon Aurora 定價網頁。

Aurora 備份儲存體的 Amazon CloudWatch 指標

您可以透過 CloudWatch 主控台使用 Amazon CloudWatch 指標來監控 Aurora 叢集和建立報告。您可以使用 CloudWatch 指標來檢閱和監控 Aurora 備份使用的儲存體用量，如下所示：系統會針對每個 Aurora 資料庫叢集獨立計算這些指標。

- `BackupRetentionPeriodStorageUsed` - 表示目前用於儲存自動備份的備份儲存量 (以位元組為單位)。
  - 此值取決於叢集磁碟區的大小，以及在保留期間對資料庫叢集所做的變更 (寫入和更新) 次數。這是因為自動備份必須存放對叢集所做的所有增量變更，才能還原到任何時間點。
  - 此指標不會減去 Aurora 提供的免費備份用量方案。
  - 此指標會針對當天記錄的自動備份用量發出單一每日資料點。
- `SnapshotStorageUsed` – 表示備份儲存量 (以位元組為單位)，用於存放在自動備份保留期間外的手動快照。
  - 此值取決於您在自動備份保留期間外保留的快照數量，以及每個快照的大小。
  - 每個快照的大小是您取得快照時的叢集磁碟區大小。
  - 快照是完整備份，而不是增量備份。
  - 此指標會針對每個收費的快照發出一個每日資料點。若要擷取每日總快照用量，請取得此指標 1 天內的總和。
- `TotalBackupStorageBilled` – 代表指定叢集之所有計費備份用量的指標 (以位元組為單位)：

  ```
  BackupRetentionPeriodStorageUsed + SnapshotStorageUsed - free tier
  ```

  - 此指標會針對 `BackupRetentionPeriodStorageUsed` 值發出每日資料點，而該值會「減去」Aurora 提供的免費備份用量方案。此免費方案等於資料庫叢集磁碟區的最新記錄大小。此資料點代表自動備份的實際計費用量。
  - 此指標會針對所有 `SnapshotStorageUsed` 值發出個別的每日資料點。
  - 若要擷取每日計費備份用量總計，請取得此指標 1 天內的總和。這會將所有計費的快照用量與計費的自動備份用量相加，以提供計費備份用量總計。

如需如何使用 CloudWatch 指標的詳細資訊，請參閱Amazon RDS 主控台中 Aurora 指標的可用性。

計算備份儲存用量

自動備份的用量是透過查看必須存放的所有增量記錄來計算，以能夠還原到備份保留期間內的任何時間點。這些變更不僅包括寫入操作的數量，還包括資料修改的大小和範圍。每種操作類型 (INSERT、UPDATE、DELETE) 都會建立變更記錄，這些記錄必須保留才能進行時間點復原。因此，雖然兩個資料庫可能會有相同數量的寫入操作 (IOPS)，但其備份儲存需求可能會因為每次交易中修改的資料量而有很大的差異。

例如，您具有保留期間為 7 天的自動備份。您在保留期間之前的叢集磁碟區大小為 100 GB，因此這是 Aurora 需要存放的最低數量。然後，您會在接下來的 7 天內進行下列活動，其中增量記錄大小是存放變更記錄所需的儲存空間量，而這些記錄來自資料庫寫入和更新。

天	增量記錄大小 (GB)
1	10
2	15
3	25
4	20
5	10
6	25
7	30
總計	135

此資料表示備份的計算自動備份用量如下：

```
100 GB (volume size before retention period) + 135 GB (size of incremental records) =
 235 GB total backup usage
```

然後，計費用量會減去免費方案的用量。假設您磁碟區的最新大小為 200 GB：

```
235 GB total backup usage - 200 GB (latest volume size) = 35 GB billed backup usage
```

FAQs

何時向我收取快照費用？

若手動快照超出 (早於) 自動備份的保留期間，則會向您收費。

什麼是手動快照？

手動快照是適用下列其中一個條件的快照：

- 由您手動請求
- 由自動化備份服務所採用，例如 AWS Backup
- 從自動化系統快照複製，以在保留期間外將其保留

如果我刪除資料庫叢集，我的手動快照會發生什麼情況？

手動快照不會過期，直到您將其刪除為止。

刪除資料庫叢集時，先前建立的手動快照會繼續存在。如果這些快照先前未因其在自動備份保留期間內計費，則現在不再涵蓋這些快照，而且全部開始按其用量的完整大小計費。

如何降低備份儲存成本？

有幾種方法可以減少與備份用量相關的成本：

- 刪除自動備份保留期間外的手動快照。這包括您拍攝的快照，以及 AWS 備份計畫可能已拍攝的快照。請務必檢查您的 AWS Backup 計劃，以確保它不會將快照保留在您未預期的保留期間之外。
- 評估您對資料庫的寫入和更新，以查看是否可以減少所做的變更次數。由於我們的自動備份會存放保留期間內的所有增量變更，因此減少您進行的更新次數也會降低自動備份費用。
- 評估縮短自動備份的保留期間是否合理。縮短保留期間代表著備份會存放更少天的增量資料，這可以降低整體備份成本。不過，縮短此保留期間也可能導致某些快照開始計費，因為其現在超出保留期間。在決定是否為正確的做法之前，請務必檢查您可能產生的所有額外快照成本。

備份儲存如何計費？

備份儲存體按 GB 月計費。

這表示備份儲存體使用量是以指定月份使用量的加權平均計費。以下是一個月 30 天的幾個範例：

- 當月所有 30 天的計費備份用量為 100 GB。您的費用如下：

```
(100 GB * 30) / 30 = 100 GB-month
```

- 當月前 15 天的計費備份用量為 100 GB，後 15 天為 0 GB。您的費用如下：

```
(100 GB * 15 + 0 GB * 15) / 30 = 50 GB-month
```

- 當月前 10 天的計費備份用量為 50 GB，接下來 10 天為 100 GB，最後 10 天則為 150 GB。您的費用如下：

```
(50 GB * 10 + 100 GB * 10 + 150 GB * 10) / 30 = 100 GB-month
```

資料庫叢集的恢復設定如何影響備份儲存用量？

Aurora 資料庫叢集的恢復設定不會影響該叢集的備份資料量。Amazon 會就用於恢復資料的儲存體另行收費。如需 Aurora 恢復操作的定價資訊，請參閱 Amazon Aurora 定價頁面。

儲存成本如何套用至共用快照？

若您與另一位使用者共享快照，您仍是該快照的擁有者，快照擁有者須支付該儲存費用。若您將擁有的共享快照刪除，那就無人能存取該快照。

如要針對其他人擁有的共享快照保留其存取權，您可以複製該快照，如此一來，您就是新快照的擁有者。對於所複製快照的儲存費用，系統將向您的帳戶收取。

如需有關共用快照的詳細資訊，請參閱 共享資料庫叢集快照。如需有關複製快照的詳細資訊，請參閱資料庫叢集快照複製。

建立資料庫叢集快照

Amazon RDS 會建立資料庫叢集的儲存體磁碟區快照，因此會備份整個資料庫叢集，而不只是個別的資料庫。建立資料庫叢集快照時，您必須找出要進行備份的資料庫叢集，並為該資料庫叢集快照命名，以便您稍後可透過它進行還原。建立資料庫叢集快照所需的時間長短隨資料庫的大小而異。由於快照包括整個儲存體磁碟區，檔案大小，例如暫存檔案，也會影響建立快照所需的時間量。

> **Note**
>
> 您的資料庫叢集必須處於 `available` 狀態，才能取得資料庫叢集快照。

與自動備份不同，手動快照不受備份保留期限的限制。快照不會過期。

針對非常長期的備份，建議您將快照資料匯出至 Amazon S3。如果資料庫引擎的主要版本不再受到支援，您則無法從快照還原至該版本。如需詳細資訊，請參閱將資料庫叢集快照資料匯出至 Amazon S3。

您可以使用 AWS CLI、 或 RDS API AWS 管理主控台建立資料庫叢集快照。

主控台

建立資料庫叢集快照

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Snapshots (快照)。

   手動快照清單即會出現。
3. 選擇 Take Snapshot (擷取快照)。

   Take DB Snapshot (建立資料庫快照) 視窗隨即顯示。
4. 對於快照類型，選取資料庫叢集。
   擷取資料庫快照。
5. 選擇您要擷取快照的資料庫叢集。
6. 輸入快照名稱。
7. 選擇 Take Snapshot (擷取快照)。

手動快照清單隨即顯示，且新的資料庫叢集快照的狀態顯示為 `Creating`。之後，其狀態為 `Available`，你可以看到其建立時間。

AWS CLI

當您使用 建立資料庫叢集快照時AWS CLI，您需要識別您要備份的資料庫叢集，然後為資料庫叢集快照命名，以便稍後可以從中還原。您可以使用 AWS CLI`create-db-cluster-snapshot`命令搭配下列參數來執行此操作：

- `--db-cluster-identifier`
- `--db-cluster-snapshot-identifier`

在此範例中，您會為 *mydbsnapshot* 資料庫叢集建立名為 *mydbsnapshot* 的資料庫叢集快照。

Example

對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster-snapshot \
    --db-cluster-identifier mydbcluster \
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

在 Windows 中：

```
aws rds create-db-cluster-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

RDS API

使用 Amazon RDS API 建立資料庫叢集快照時，您必須找出要進行備份的資料庫叢集，並為該資料庫叢集快照命名，以便您稍後可透過它進行還原。您可以使用 Amazon RDS API `CreateDBClusterSnapshot` 命令，並搭配下列參數來執行此動作：

- DBClusterIdentifier
- DBClusterSnapshotIdentifier

判斷資料庫叢集快照是否可用

您可以查看 中叢集詳細資訊頁面上維護與備份索引標籤的快照、AWS 管理主控台使用 CLI `describe-db-cluster-snapshots` 命令或使用 `DescribeDBClusterSnapshots` API 動作，來檢查資料庫叢集快照是否可用。

您也可以使用 `wait db-cluster-snapshot-available` CLI 命令每隔 30 秒輪詢 API，直到快照可用為止。

從資料庫叢集快照還原

Amazon RDS 會建立資料庫叢集的儲存體磁碟區快照，因此會備份整個資料庫叢集，而不只是個別的資料庫。您可從資料庫快照還原來建立新的資料庫叢集。您提供要從中還原之資料庫叢集快照的名稱，然後提供一個從還原建立之新資料庫叢集的名稱。您無法從資料庫叢集快照還原為現有的資料庫叢集；還原時會建立新的資料庫叢集。

> **Important**
>
> 您無法將快照還原至已超過其 Aurora 標準支援結束日期的資料庫引擎版本。只有在資料庫成功升級至支援的版本之後，您才能存取該資料庫。如需支援的 Aurora 資料庫引擎版本的詳細資訊，請參閱 使用 Amazon Aurora 的 Amazon RDS 延伸支援。
> 如果升級到叢集支援的版本失敗，叢集狀態會變更為 `upgrade_failed`，Aurora 會建立字首為 的最終快照`rds-final`。如需在升級失敗後存取已取代版本的還原資料庫，請聯絡 AWS Support。

若還原的資料庫叢集狀態為 `available`，您便可使用該叢集。

您可以使用 從資料庫叢集快照 CloudFormation 還原資料庫叢集。如需詳細資訊，請參閱《AWS CloudFormation 使用者指南》中的 AWS::RDS::DBCluster。

> **Note**
>
> 共用手動資料庫叢集快照，無論是加密或未加密，都可讓授權 AWS 帳戶直接從快照還原資料庫叢集，而不是複製資料庫叢集並從中還原。如需詳細資訊，請參閱共享資料庫叢集快照。

如需使用 RDS 延長支援版本還原 Aurora 資料庫叢集或全域叢集的相關資訊，請參閱 使用 Amazon RDS 延伸支援還原Aurora 資料庫叢集或全域叢集。

參數群組考量

建議您針對您建立的任何資料庫叢集快照，保留資料庫參數群組和資料庫叢集參數群組，這樣才能將還原的資料庫叢集與正確的參數群組建立關聯。

預設的資料庫參數群組和資料庫叢集參數群組會與還原的叢集建立關聯，除非您選擇不同叢集。預設參數群組中沒有可用的自訂參數設定。

還原資料庫叢集時，可以指定參數群組。

如需資料庫參數群組和資料庫叢集參數群組的詳細資訊，請參閱 Amazon Aurora 的參數群組。

安全群組考量

還原資料庫叢集時，預設的虛擬私有雲端 (VPC)、資料庫子網路群組和 VPC 安全群組會與還原的執行個體建立關聯，除非您選擇不同的執行個體。

- 若您使用 Amazon RDS 主控台，可指定要與叢集建立關聯的自訂 VPC 安全群組，或建立新的 VPC 安全群組。
- 如果您使用的是 AWS CLI，您可以在 `restore-db-cluster-from-snapshot`命令中包含 `--vpc-security-group-ids`選項，以指定要與叢集建立關聯的自訂 VPC 安全群組。
- 如果您是使用 Amazon RDS API，則可以在 `VpcSecurityGroupIds.VpcSecurityGroupId.N` 動作中包括 `RestoreDBClusterFromSnapshot` 參數。

一旦還原完成且新的資料庫叢集可用，您還可修改資料庫叢集來變更 VPC 設定。如需詳細資訊，請參閱修改 Amazon Aurora 資料庫叢集。

Amazon Aurora 考量

使用 Aurora 來還原資料庫叢集快照至資料庫叢集。

使用 Aurora MySQL 和 Aurora PostgreSQL，您也能將資料庫叢集快照還原為 Aurora Serverless 資料庫叢集。如需詳細資訊，請參閱還原 Aurora Serverless v1 資料庫叢集。

使用 Aurora MySQL，您可從不含平行查詢的叢集將資料庫叢集快照還原至含平行查詢的叢集。平行查詢通常用於大型資料表，因此快照機制會是能最快將大量資料擷取到 Aurora MySQL 平行查詢啟用叢集的方式。如需詳細資訊，請參閱Amazon Aurora MySQL 的平行查詢。

從快照還原

您可使用 AWS 管理主控台、 AWS CLI或 RDS API 從資料庫叢集快照還原資料庫叢集。

主控台

從資料庫叢集快照還原資料庫叢集

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。

2. 在導覽窗格中，選擇 Snapshots (快照)。
3. 選擇想要從中還原的資料庫叢集快照。
4. 針對 Actions (動作)，選擇 Restore snapshot (還原快照)。

   還原快照頁面隨即顯示。
5. 選擇您要將資料庫叢集還原至其中的資料庫引擎版本。

   根據預設，快照會還原至與來源資料庫叢集相同的資料庫引擎版本 (如果該版本可用)。
6. 對於 DB instance identifier (資料庫執行個體識別符)，輸入所還原資料庫執行個體的名稱。請注意，Amazon RDS 會從您指定的資料庫執行個體識別符衍生資料庫叢集識別符。
7. 指定其他設定，例如資料庫叢集儲存組態。

   如需每項設定的相關資訊，請參閱 Aurora 資料庫叢集的設定。
8. 選擇 Restore DB Cluster (還原資料庫叢集)。

AWS CLI

若要從資料庫叢集快照還原資料庫叢集，請使用 AWS CLI 命令 restore-db-cluster-from-snapshot。

在此範例中，您會從之前建立、名稱為 `mydbclustersnapshot` 的資料庫快照還原。您會還原至名稱為 `mynewdbcluster` 的新資料庫叢集。

您可以指定其他設定，例如，資料庫引擎版本。如果您未指定引擎版本，則資料庫叢集會還原為預設引擎版本。

如需每項設定的相關資訊，請參閱 Aurora 資料庫叢集的設定。

Example

在 Linux、macOS 或 Unix 中：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine aurora-mysql|aurora-postgresql
```

在 Windows 中：

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mynewdbcluster ^
    --snapshot-identifier mydbclustersnapshot ^
    --engine aurora-mysql|aurora-postgresql
```

資料庫叢集還原之後，如果您要擁有與先前的資料庫叢集相同的功能，必須將資料庫叢集新增至資料庫叢集用來建立資料庫叢集快照的安全群組。

> **Important**
>
> 如果您使用主控台來還原資料庫叢集，則 Amazon RDS 會自動建立資料庫叢集的主要資料庫執行個體 (寫入器)。如果您使用 AWS CLI 來還原資料庫叢集，則必須明確地建立資料庫叢集的主要執行個體。主要執行個體是資料庫叢集內第一個建立的執行個體。如果您未建立主要資料庫執行個體，資料庫叢集端點會保持在 `creating` 狀態。
>
> 呼叫 create-db-instance AWS CLI 命令，為您的資料庫叢集建立主要執行個體。包含資料庫叢集的名稱做為 `--db-cluster-identifier` 選項值。

RDS API

若要從資料庫叢集快照還原資料庫叢集，請使用下列參數來呼叫 RDS API 操作 RestoreDBClusterFromSnapshot：

- `DBClusterIdentifier`
- `SnapshotIdentifier`

> **Important**
>
> 如果您使用主控台來還原資料庫叢集，則 Amazon RDS 會自動建立資料庫叢集的主要資料庫執行個體 (寫入器)。如果您使用 RDS API 來還原資料庫叢集，則必須明確地建立資料庫叢集的主要執行個體。主要執行個體是資料庫叢集內第一個建立的執行個體。如果您未建立主要資料庫執行個體，資料庫叢集端點會保持在 `creating` 狀態。
>
> 呼叫 RDS API 操作 CreateDBInstance 命令，來建立資料庫叢集的主要執行個體。包含資料庫叢集的名稱做為 `DBClusterIdentifier` 參數值。

資料庫叢集快照複製

透過 Amazon Aurora，即可複製自動備份或手動資料庫叢集快照。複製快照之後，副本即為手動快照。您可以製作自動備份或手動快照的多個複本，但每個複本都必須具有唯一的識別碼。

您可以在相同 中複製快照 AWS 區域、跨 複製快照 AWS 區域，以及複製共用快照。您可以在單一步驟中將快照複製到另一個 AWS 區域 或 帳戶。

> **Note**
>
> Amazon 會根據您保留的 Amazon Aurora 備份和快照資料量及保留的期間長短來向您收費。如需與 Aurora 備份和快照相關聯的儲存體的資訊，請參閱 了解 Amazon Aurora 備份儲存體用量。如需 Aurora 儲存的定價資訊，請參閱 Amazon RDS for Aurora 定價。

檢閱資料庫叢集快照複製的限制和考量事項。若要複製資料庫叢集快照，請參閱下列其中一個主題。

- 使用 AWS 管理主控台 複製資料庫叢集快照
- 使用 AWS CLI 或 Amazon RDS API 複製未加密的資料庫叢集快照
- 使用 AWS CLI 或 Amazon RDS API 複製已加密的資料庫叢集快照
- 跨帳戶複製資料庫叢集快照

使用 AWS 管理主控台 複製資料庫叢集快照

使用本主題中的程序來複製資料庫叢集快照。如果來源資料庫引擎是 Aurora，則快照即為資料庫叢集快照。

對於每個 AWS 帳戶，您一次最多可以將五個資料庫叢集快照從一個 AWS 區域 複製到另一個。支援複製加密和未加密的資料庫叢集快照。如果您將資料庫叢集快照複製到另一個 AWS 區域，您會建立一個保留在該 AWS 區域 中的手動資料庫叢集快照。從來源 AWS 區域 中複製叢集資料庫快照會產生 Amazon RDS 資料傳輸費。

如需資料傳輸定價的詳細資訊，請參閱 Amazon RDS 定價。

在新的 AWS 區域 中建立資料庫叢集快照副本之後，資料庫叢集快照副本的行為就如同該 AWS 區域的其他所有資料庫叢集快照一樣。

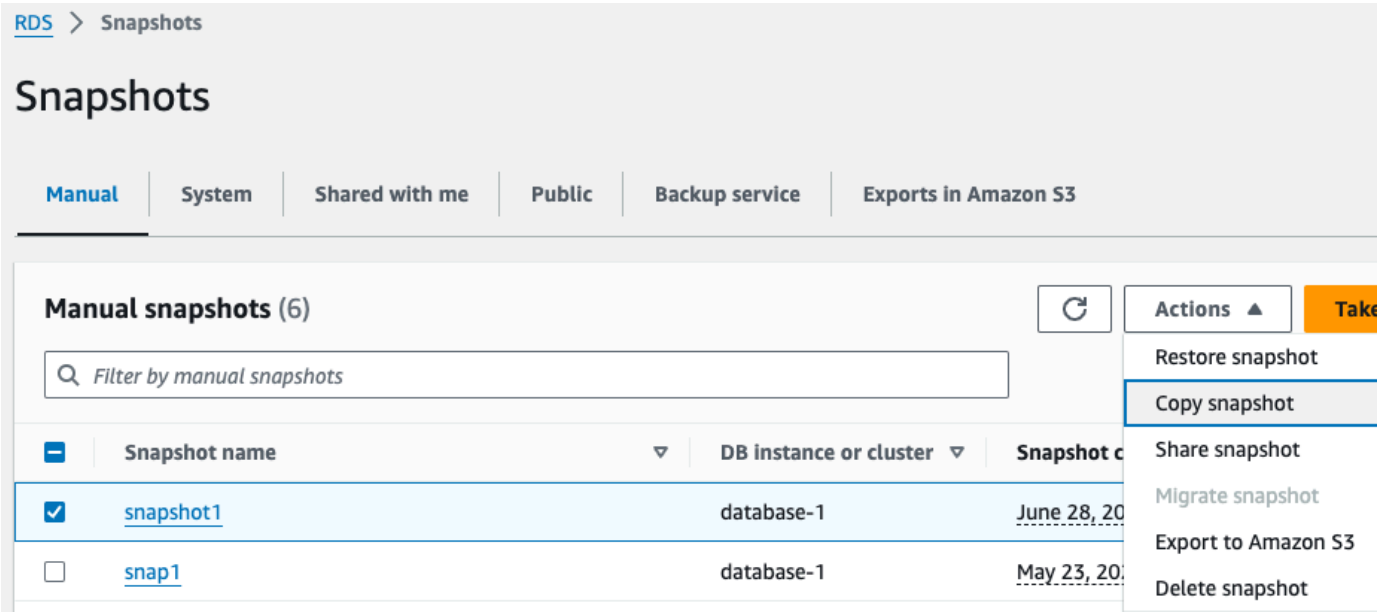
此程序適用於在相同 AWS 區域 或跨區域複製已加密或未加密的資料庫叢集快照。

若要取消已開始進行的複製操作，當目標資料庫叢集快照處於 copying (複製中) 狀態時，請刪除該資料庫叢集快照。

複製資料庫叢集快照之前，請檢閱 限制 和 快照複製的考量事項。

複製資料庫叢集快照

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Snapshots (快照)。
3. 選取您要複製的資料庫叢集快照。
4. 選擇動作，然後選擇複製快照。



5. (選用) 若要將資料庫叢集快照複製到不同的 AWS 區域，請在 Destination Region (目的地區域) 中選擇該 AWS 區域。
6. 在 New DB Snapshot Identifier (新的資料庫快照識別符) 中輸入資料庫叢集快照副本的名稱。
7. 若要將快照中的標籤和值複製到快照的副本，請選擇 Copy Tags (複製標籤)。
8. 選擇 Copy Snapshot (複製快照)。

限制

以下是複製快照時的一些限制：

- 您無法將快照複製到下列項目或從中複製快照 AWS 區域：

  - 中國 (北京)
  - 中國 (寧夏)
- 您可以在 AWS GovCloud （美國東部） 和 AWS GovCloud （美國西部） 之間複製快照。不過，您無法在這些 AWS GovCloud (US) 區域與商業之間複製快照 AWS 區域。
- 如果您在目標快照變成可用之前刪除來源快照，則快照副本可能會失敗。刪除來源快照之前，請確認目標快照的狀態為 AVAILABLE。
- 對於單一目的地區域，每一帳戶最多可有 5 個快照複製請求在進行中。
- 當您為相同來源資料庫執行個體請求多個快照副本時，其將會在內部排隊。稍後請求的副本在先前的快照副本完成前將不會啟動。如需詳細資訊，請參閱 AWS 知識中心的為什麼我的 EC2 AMI 或 EBS 快照建立速度緩慢？。
- 根據 AWS 區域 涉及的 和要複製的資料量，跨區域快照複本可能需要數小時才能完成。有時某個來源區域會出現大量跨區域快照複製請求。此時，在一些進行中的複製完成之前，Amazon RDS 可能會將該來源區域新提出的跨區域複製請求排入佇列。位於佇列中的複製請求不會顯示進度資訊。複製開始時才會顯示進度資訊。
- Aurora 不支援增量式快照。Aurora 資料庫叢集快照副本一律會儲存為完整副本。完整的快照複本內含所有還原資料庫叢集所需的資料及中繼資料。

快照複製的考量事項

以下是複製快照時的考量事項。

主題

- 共用快照複製的考量事項
- 加密資料庫叢集快照複製的考量事項
- 跨區域快照複製的考量事項
- 參數群組的考量事項

共用快照複製的考量事項

您可以複製其他 AWS 帳戶共用給您的快照。在某些情況下，您可以複製已從另一個 AWS 帳戶共用的加密快照。在這些情況下，您必須能夠存取 AWS KMS key 用來加密快照的 。如需詳細資訊，請參閱共用加密快照。

單一步驟中的跨區域和跨帳戶複製

若要在單一動作中複製快照跨區域和跨帳戶，您必須先與目標 AWS 帳戶共用快照。如果快照已加密，您還必須與目標 AWS 帳戶共用 AWS KMS 金鑰。如果快照使用預設 AWS KMS 金鑰加密，您必須先複製快照，以使用客戶受管金鑰重新加密快照，然後再與目標帳戶共用快照。共用後，您可以從目標帳戶啟動該帳戶（區域內或跨區域）的副本。

加密資料庫叢集快照複製的考量事項

您可以複製使用 KMS 金鑰所加密的快照。如果您複製加密快照，則快照的副本也必須加密。如果您在相同的 中複製加密快照 AWS 區域，您可以使用與原始快照相同的 KMS 金鑰來加密複本。或者，您可以指定不同的 KMS 金鑰。

若要跨區域複製加密的快照，就必須指定在目的地 AWS 區域中有效的 KMS 金鑰。可以是特定區域專用的 KMS 金鑰，也可以是多區域金鑰。如需多區域 KMS 金鑰的詳細資訊，請參閱使用 AWS KMS中的多區域金鑰。

如需 Amazon RDS AWS KMS 金鑰管理的詳細資訊，請參閱 AWS KMS key 管理。

在整個複製過程中來源快照仍會保持加密狀態。如需詳細資訊，請參閱Amazon Aurora 加密資料庫叢集的限制。

> **Note**
>
> 對於 Amazon Aurora 資料庫快照，當您複製快照時，無法將未加密的資料庫叢集快照加密。

若要複製加密的資料庫叢集快照，請參閱下列主題。

- 使用 AWS CLI 或 Amazon RDS API 複製已加密的資料庫叢集快照
- 跨帳戶複製資料庫叢集快照

跨區域快照複製的考量事項

您可以跨 AWS 區域複製資料庫叢集快照。但是，跨區域快照複製具有特定的限制和考量。

根據 AWS 區域 涉及的 和要複製的資料量，跨區域快照複本可能需要數小時才能完成。

有時某個來源 AWS 區域會出現大量跨區域快照複製請求。在這種情況下，Amazon RDS 可能會將來自該來源的新跨區域複製請求 AWS 區域 放入佇列，直到某些進行中的複製完成為止。位於佇列中的複製請求不會顯示進度資訊。複製開始時才會顯示進度資訊。

跨區域快照複製需支付資料傳輸費用。跨區域快照複製會在目標資料中建立完整複本，但資料傳輸費用會遞增。增量資料包括自上次複製後新增至客戶資料庫的新資料，以及對現有資料所做的任何變更。如需詳細資訊，請參閱《AWS Backup 開發人員指南》中的跨 AWS 區域建立備份複本。

> **Note**
>
> Aurora 會複製在目的地區域中建立快照完整複本所需的最低資料量。在區域之間複製快照時，需支付資料傳輸費用。

參數群組的考量事項

當您跨區域複製快照時，副本不包含原始資料庫叢集所使用的參數群組。當您還原快照以建立新的資料庫叢集時，該資料庫叢集會取得 AWS 區域 其建立所在 的預設參數群組。若要提供原始的相同參數給新的資料庫叢集，需要執行下列動作：

1. 在目的地中 AWS 區域，建立與原始資料庫叢集具有相同設定的資料庫叢集參數群組。如果新的 中已存在， AWS 區域您可以使用該項目。
2. 在目的地還原快照後 AWS 區域，請修改新的資料庫叢集，並從上一個步驟新增新的或現有的參數群組。

使用 AWS CLI 或 Amazon RDS API 複製未加密的資料庫叢集快照

請根據以下幾節中的程序，使用 AWS 管理主控台、AWS CLI 或 Amazon RDS API 來複製未加密的資料庫叢集快照。

若要取消已開始進行的複製操作，請在資料庫叢集快照處於 copying (複製中) 狀態時，刪除 `--target-db-cluster-snapshot-identifier` 或 `TargetDBClusterSnapshotIdentifier` 所識別的目標資料庫叢集快照。

主控台

若要使用 AWS 管理主控台 複製資料庫叢集快照，請參閱 使用 AWS 管理主控台 複製資料庫叢集快照。

AWS CLI

如要複製資料庫叢集快照，請使用 AWS CLI copy-db-cluster-snapshot 命令。若您要將快照複製至另一個 AWS 區域，請在快照將複製到的 AWS 區域 中執行此命令。

下列選項用來複製未加密的資料庫叢集快照：

- `--source-db-cluster-snapshot-identifier` – 要複製之資料庫叢集快照的識別符。若您要將快照複製到另一個 AWS 區域，此識別符必須是來源 AWS 區域 的 ARN 格式。
- `--target-db-cluster-snapshot-identifier` – 資料庫叢集快照之新副本的識別符。

下列程式碼會在執行命令的 AWS 區域 中，建立資料庫叢集快照 `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805` 的複本，名稱為 `myclustersnapshotcopy`。建立副本時，原始快照的所有標籤會複製到快照副本。

Example

對於 Linux、macOS 或 Unix：

```
aws rds copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 \
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy \
  --copy-tags
```

在 Windows 中：

```
aws rds copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 ^
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
  --copy-tags
```

RDS API

如要複製資料庫叢集快照，請使用 Amazon RDS API CopyDBClusterSnapshot 作業。若要將快照複製至另一個 AWS 區域，請在快照將複製到的 AWS 區域 中執行此動作。

下列參數用來複製未加密的資料庫叢集快照：

- `SourceDBClusterSnapshotIdentifier` – 要複製之資料庫叢集快照的識別符。若您要將快照複製到另一個 AWS 區域，此識別符必須是來源 AWS 區域 的 ARN 格式。
- `TargetDBClusterSnapshotIdentifier` – 資料庫叢集快照之新副本的識別符。

下列程式碼會在 美國西部 (加州北部) 區域中建立快照 arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 的複本，名稱為 myclustersnapshotcopy。建立副本時，原始快照的所有標籤會複製到快照副本。

Example

```
https://rds.us-west-1.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-east-1%3A123456789012%3Acluster-snapshot%3Aaurora-cluster1-snapshot-20130805
   &TargetDBSnapshotIdentifier=myclustersnapshotcopy
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20140429/us-west-1/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
   &X-Amz-Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

使用 AWS CLI 或 Amazon RDS API 複製已加密的資料庫叢集快照

請根據以下幾節中的程序，使用 AWS 管理主控台、AWS CLI 或 Amazon RDS API 來複製已加密的資料庫叢集快照。

若要取消已開始進行的複製操作，請在資料庫叢集快照處於 copying (複製中) 狀態時，刪除 --target-db-cluster-snapshot-identifier 或 TargetDBClusterSnapshotIdentifier 所識別的目標資料庫叢集快照。

主控台

若要使用 AWS 管理主控台 複製資料庫叢集快照，請參閱 使用 AWS 管理主控台 複製資料庫叢集快照。

AWS CLI

如要複製資料庫叢集快照，請使用 AWS CLI copy-db-cluster-snapshot 命令。若您要將快照複製至另一個 AWS 區域，請在快照將複製到的 AWS 區域 中執行此命令。

下列選項用來複製已加密的資料庫叢集快照：

- `--source-db-cluster-snapshot-identifier` – 要複製之加密資料庫叢集快照的識別符。若您要將快照複製到另一個 AWS 區域，此識別符必須是來源 AWS 區域 的 ARN 格式。
- `--target-db-cluster-snapshot-identifier` – 加密資料庫叢集快照之新副本的識別符。
- `--kms-key-id` – 金鑰的 KMS 金鑰識別碼，用來加密資料庫叢集快照的複本。

  如果資料庫叢集快照已加密、您在相同的 AWS 區域 中複製快照，且您想要指定新的 KMS 金鑰來加密複本，則可以選擇性使用此選項。否則會使用與來源資料庫叢集快照相同的 KMS 金鑰，以加密資料庫叢集快照的副本。

  如果資料庫叢集快照已加密，且您要將快照複製到另一個 AWS 區域，則必須使用此選項。在該情況下，您必須為目的地 AWS 區域 指定 KMS 金鑰。

下列程式碼範例從 美國西部 (奧勒岡) 區域將已加密的資料庫叢集快照複製到 US East (N. Virginia) 區域。該命令是在 US East (N. Virginia) 區域中呼叫。

Example

對於 Linux、macOS 或 Unix：

```
aws rds copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20161115 \
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy \
  --kms-key-id my-us-east-1-key
```

在 Windows 中：

```
aws rds copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20161115 ^
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
  --kms-key-id my-us-east-1-key
```

當您在 AWS GovCloud (美國東部) 和 AWS GovCloud (美國西部) 區域之間複製加密的資料庫叢集快照時，需要 `--source-region` 參數。對於 `--source-region`，請指定來源資料庫執行個體的 AWS 區域。指定於 `source-db-cluster-snapshot-identifier` 中的 AWS 區域 必須與為 `--source-region` 指定的 AWS 區域 相符。

若未指定 `--source-region`，請指定 `--pre-signed-url` 值。presigned URL (預先簽章的 URL) 為包含對來源 AWS 區域 中呼叫 `copy-db-cluster-snapshot` 命令之 Signature 第 4 版簽章請

求的 URL。如要進一步了解 `pre-signed-url` 選項，請參閱《AWS CLI 命令參考》中的 copy-db-cluster-snapshot。

RDS API

如要複製資料庫叢集快照，請使用 Amazon RDS API CopyDBClusterSnapshot 作業。若要將快照複製至另一個 AWS 區域，請在快照將複製到的 AWS 區域 中執行此動作。

下列參數用來複製已加密的資料庫叢集快照：

- `SourceDBClusterSnapshotIdentifier` – 要複製之加密資料庫叢集快照的識別符。若您要將快照複製到另一個 AWS 區域，此識別符必須是來源 AWS 區域 的 ARN 格式。
- `TargetDBClusterSnapshotIdentifier` – 加密資料庫叢集快照之新副本的識別符。
- `KmsKeyId` – 金鑰的 KMS 金鑰識別碼，用來加密資料庫叢集快照的複本。

  如果資料庫叢集快照已加密、您在相同的 AWS 區域 中複製快照，且您指定新的 KMS 金鑰來加密複本，則可選擇使用此參數。否則會使用與來源資料庫叢集快照相同的 KMS 金鑰，以加密資料庫叢集快照的副本。

  如果資料庫叢集快照已加密，且您要將快照複製至另一個 AWS 區域，則必須使用此參數。在該情況下，您必須為目的地 AWS 區域 指定 KMS 金鑰。
- `PreSignedUrl` – 若您要將快照複製至另一個 AWS 區域，則必須指定 `PreSignedUrl` 參數。`PreSignedUrl` 值必須是包含 Signature 第 4 版簽署請求的 URL，而該請求是以來源 AWS 區域 (從中複製資料庫叢集快照) 中要呼叫的 `CopyDBClusterSnapshot` 動作做為對象。若要進一步了解使用預先簽章的 URL，請參閱 CopyDBClusterSnapshot。

下列程式碼範例從 美國西部 (奧勒岡) 區域將已加密的資料庫叢集快照複製到 US East (N. Virginia) 區域。該動作是在 US East (N. Virginia) 區域中呼叫。

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CopyDBClusterSnapshot
    &KmsKeyId=my-us-east-1-key
    &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCopyDBClusterSnapshot
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
```

```
        %2526SourceDBClusterSnapshotIdentifier%253Darn%25253Aaws%25253Ards
%25253Aus-west-2%25253A123456789012%25253Acluster-snapshot%25253Aaurora-cluster1-
snapshot-20161115
        %2526SignatureMethod%253DHmacSHA256
        %2526SignatureVersion%253D4
        %2526Version%253D2014-10-31
        %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
        %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-west-2%252Frds
%252Faws4_request
        %2526X-Amz-Date%253D20161117T215409Z
        %2526X-Amz-Expires%253D3600
        %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-amz-
content-sha256%253Bx-amz-date
        %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &SourceDBClusterSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-
west-2%3A123456789012%3Acluster-snapshot%3Aaurora-cluster1-snapshot-20161115
    &TargetDBClusterSnapshotIdentifier=myclustersnapshotcopy
    &Version=2014-10-31
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
    &X-Amz-Date=20161117T221704Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=da4f2da66739d2e722c85fcfd225dc27bba7e2b8dbea8d8612434378e52adccf
```

當您在 AWS GovCloud (美國東部) 和 AWS GovCloud (美國西部) 區域之間複製加密的資料庫叢集快照時，需要 `PreSignedUrl` 參數。`PreSignedUrl` 值必須是包含 Signature 第 4 版簽署請求的 URL，而該請求是以來源 AWS 區域 (從中複製資料庫叢集快照) 中要呼叫的 `CopyDBClusterSnapshot` 作業做為對象。若要進一步了解使用預先簽章的 URL，請參閱《Amazon RDS API 參考》中的 CopyDBClusterSnapshot。

如要自動而非手動產生預先簽章的 URL，請改為使用 AWS CLI copy-db-cluster-snapshot 命令搭配 `--source-region` 選項。

跨帳戶複製資料庫叢集快照

您可以使用 Amazon RDS API `ModifyDBClusterSnapshotAttribute` 和 `CopyDBClusterSnapshot` 動作，讓其他 AWS 帳戶可以複製您指定的資料庫叢集快照。您只能在相

同 AWS 區域 中跨帳戶複製資料庫叢集快照。跨帳戶複製程序如下所示，其中帳戶 A 提供可複製的快照，而帳戶 B 會複製快照。

1. 使用帳戶 A，呼叫 ModifyDBClusterSnapshotAttribute，並在 **restore** 參數中指定 AttributeName，在 ValuesToAdd 參數中指定帳戶 B 的 ID。
2. (如果快照已加密) 使用帳戶 A，更新 KMS 金鑰的金鑰政策，先將帳戶 B 的 ARN 新增為 Principal，然後允許 kms:CreateGrant 動作。
3. (如果快照已加密) 使用帳戶 B，選擇或建立使用者，並將 IAM 政策連接到該使用者，以允許使用者以您的 KMS 金鑰來複製已加密的資料庫叢集快照。
4. 使用帳戶 B，呼叫 CopyDBClusterSnapshot，並使用 SourceDBClusterSnapshotIdentifier 參數來指定要複製之資料庫叢集快照的 ARN，其中必須包含帳戶 A 的 ID。

若要列出所有允許還原資料庫叢集快照的 AWS 帳戶，請使用 DescribeDBSnapshotAttributes 或 DescribeDBClusterSnapshotAttributes API 操作。

若要移除 AWS 帳戶的共用許可，請使用 ModifyDBSnapshotAttribute 或 ModifyDBClusterSnapshotAttribute 動作，並將 AttributeName 設為 restore，在 ValuesToRemove 參數中指定要移除之帳戶的 ID。

將未加密的資料庫叢集快照複製到另一個帳戶

使用下列程序，將未加密的資料庫叢集快照複製至相同 AWS 區域 中的另一個帳戶。

1. 在資料庫叢集快照的來源帳戶中，呼叫 ModifyDBClusterSnapshotAttribute，並在 **restore** 參數中指定 AttributeName，在 ValuesToAdd 參數中指定目標帳戶的 ID。

   使用帳戶 987654321 執行下列範例，以允許兩個 AWS 帳戶識別符 (123451234512 和 123456789012) 還原名為 manual-snapshot1 的資料庫叢集快照。

```
https://rds.us-west-2.amazonaws.com/
 ?Action=ModifyDBClusterSnapshotAttribute
 &AttributeName=restore
 &DBClusterSnapshotIdentifier=manual-snapshot1
 &SignatureMethod=HmacSHA256&SignatureVersion=4
 &ValuesToAdd.member.1=123451234512
 &ValuesToAdd.member.2=123456789012
 &Version=2014-10-31
 &X-Amz-Algorithm=AWS4-HMAC-SHA256
```

```
 &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
 &X-Amz-Date=20150922T220515Z
 &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
 &X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36dddbb3
```

2. 在目標帳戶中，呼叫 CopyDBClusterSnapshot，並使用 SourceDBClusterSnapshotIdentifier 參數來指定要複製之資料庫叢集快照的 ARN，其中必須包含來源帳戶的 ID。

   使用帳戶 123451234512 執行下列範例，以便從帳戶 aurora-cluster1-snapshot-20130805 複製資料庫叢集快照 987654321，並建立名為 dbclustersnapshot1 的資料庫叢集快照。

```
https://rds.us-west-2.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
   &TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-
date
   &X-Amz-
Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

將已加密的資料庫叢集快照複製到另一個帳戶

使用下列程序，將已加密的資料庫叢集快照複製至相同 AWS 區域 中的另一個帳戶。

1. 在資料庫叢集快照的來源帳戶中，呼叫 ModifyDBClusterSnapshotAttribute，並在 **restore** 參數中指定 AttributeName，在 ValuesToAdd 參數中指定目標帳戶的 ID。

   使用帳戶 987654321 執行下列範例，以允許兩個 AWS 帳戶識別符 (123451234512 和 123456789012) 還原名為 manual-snapshot1 的資料庫叢集快照。

```
https://rds.us-west-2.amazonaws.com/
```

```
 ?Action=ModifyDBClusterSnapshotAttribute
 &AttributeName=restore
 &DBClusterSnapshotIdentifier=manual-snapshot1
 &SignatureMethod=HmacSHA256&SignatureVersion=4
 &ValuesToAdd.member.1=123451234512
 &ValuesToAdd.member.2=123456789012
 &Version=2014-10-31
 &X-Amz-Algorithm=AWS4-HMAC-SHA256
 &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
 &X-Amz-Date=20150922T220515Z
 &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
 &X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36dddbb3
```

2. 在資料庫叢集快照的來源帳戶中，在與加密資料庫叢集快照相同的 AWS 區域 中建立自訂 KMS 金鑰。建立客戶自管金鑰時，您可以提供目標 AWS 帳戶 的存取權。如需更多詳細資訊，請參閱 建立客戶自管金鑰並授予存取權。
3. 將快照複製到目標 AWS 帳戶 並共用。如需更多詳細資訊，請參閱 從來源帳戶複製和共用快照。
4. 在目標帳戶中，呼叫 CopyDBClusterSnapshot，並使用 SourceDBClusterSnapshotIdentifier 參數來指定要複製之資料庫叢集快照的 ARN，其中必須包含來源帳戶的 ID。

   使用帳戶 123451234512 執行下列範例，以便從帳戶 aurora-cluster1-snapshot-20130805 複製資料庫叢集快照 987654321，並建立名為 dbclustersnapshot1 的資料庫叢集快照。

```
https://rds.us-west-2.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
   &TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-
date
   &X-Amz-
Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

共享資料庫叢集快照

使用 Amazon RDS 時，您可以透過下列方式共用手動資料庫叢集快照：

- 共用手動資料庫叢集快照 (無論加密或未加密) 可讓授權的 AWS 帳戶複製快照。
- 分享手動資料庫叢集快照 (無論加密或未加密) 可讓授權的 AWS 帳戶直接從快照還原資料庫叢集，而不需要先複製再還原。

> **Note**
>
> 若要共用自動資料庫叢集快照，請複製該自動快照來建立手動資料庫叢集快照，然後共用該複本。此程序也適用 AWS Backup 產生的資源。

如需有關複製快照的詳細資訊，請參閱資料庫叢集快照複製。如需有關從資料庫叢集快照還原資料庫執行個體的詳細資訊，請參閱 從資料庫叢集快照還原。

如需有關從資料庫叢集快照還原資料庫叢集的詳細資訊，請參閱備份與還原 Aurora 資料庫叢集的概觀。

您可以與其他最多 20 個 共用手動快照 AWS 帳戶。

與其他 共用手動快照時，適用下列限制 AWS 帳戶：

- 當您使用 AWS Command Line Interface (AWS CLI) 或 Amazon RDS API 從共用快照還原資料庫叢集時，您必須指定共用快照的 Amazon Resource Name (ARN) 做為快照識別符。

在下列各節中了解如何共用快照、公有快照和加密快照。您也可以了解如何停止共用快照。

主題

- 共用快照
- 共用公有快照
- 共用加密快照
- 停止快照共用

共用快照

您可以使用 AWS CLI、 或 RDS API AWS 管理主控台共用資料庫叢集快照。

主控台

使用 Amazon RDS 主控台，您可以與最多 20 個 共用手動資料庫叢集快照 AWS 帳戶。您也可以使用主控台來停止將手動快照共用給一或多個帳戶。

使用 Amazon RDS 主控台來分享手動資料庫叢集快照

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Snapshots (快照)。
3. 選取您要共享的手動快照。
4. 針對 Actions (動作)，選擇 Share snapshot (共用快照)。
5. 在 DB snapshot visibility (資料庫快照可見度) 中，選擇下列其中一個選項。
   - 如果來源未加密，請選擇公有 AWS 帳戶 以允許所有 從您的手動資料庫叢集快照還原資料庫叢集，或選擇私有以僅允許 AWS 帳戶 您指定 從手動資料庫叢集快照還原資料庫叢集。

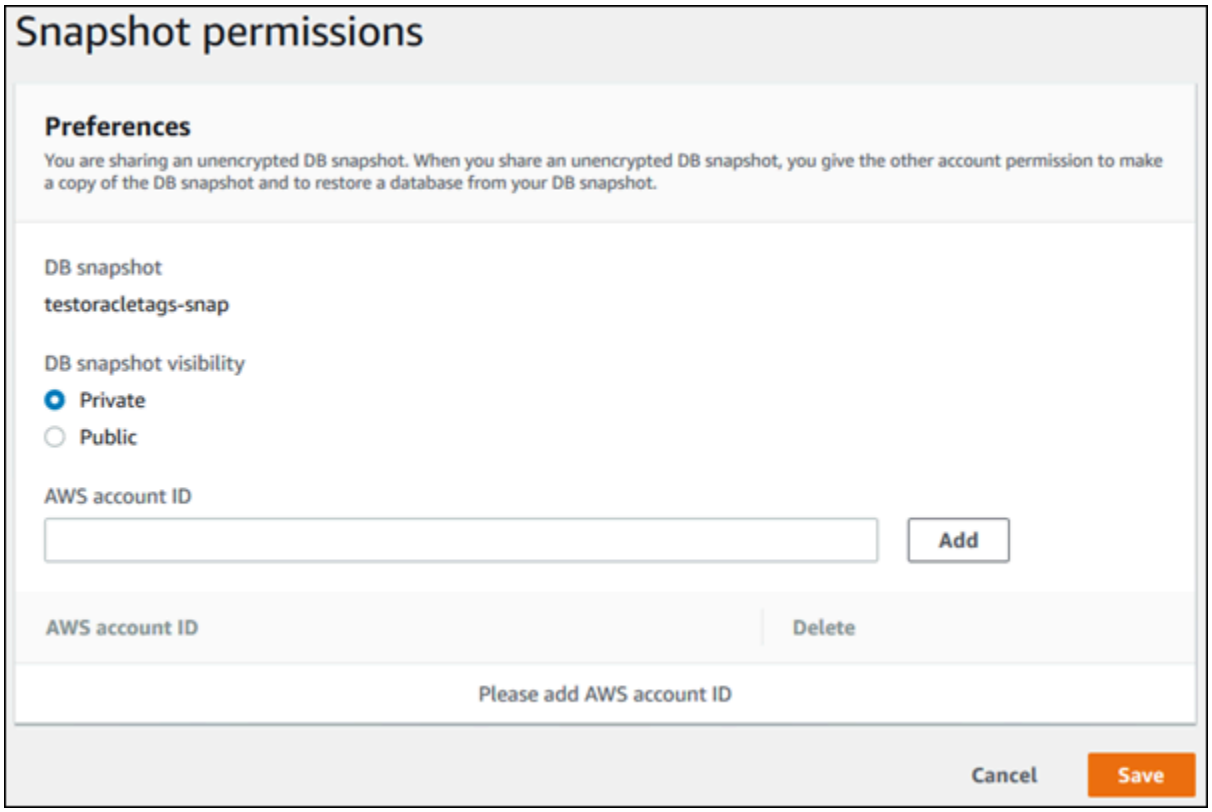
     > **Warning**
     >
     > 如果您將資料庫快照可見性設定為公有，則所有 AWS 帳戶 都可以從手動資料庫叢集快照還原資料庫叢集，並存取您的資料。請勿將任何包含私人資訊的手動資料庫叢集快照以 Public (公有) 形式共用。
     > 如需詳細資訊，請參閱共用公有快照。

   - 如果來源資料庫叢集已加密，DB snapshot visibility (資料庫快照可見度) 會設為 Private (私有)，因為加密快照無法以公有形式共用。

     > **Note**
     >
     > 使用預設加密的快照 AWS KMS key 無法共用。如需如何解決此問題的資訊，請參閱 共用加密快照。

6. 針對AWS 帳戶 ID，輸入您要允許 從手動快照還原資料庫叢集之帳戶的 AWS 帳戶 識別符，然後選擇新增。重複 以包含其他識別 AWS 帳戶 符，最多 20 個 AWS 帳戶。

如果您在將 AWS 帳戶 識別符新增至允許的帳戶清單時發生錯誤，您可以選擇不正確 AWS 帳戶 識別符右側的刪除，將其從清單中刪除。



7. 為 AWS 帳戶 要允許還原手動快照的所有 新增識別符後，請選擇儲存以儲存變更。

AWS CLI

如要共享資料庫叢集快照，請使用 `aws rds modify-db-cluster-snapshot-attribute` 命令。使用 `--values-to-add` 參數來新增 AWS 帳戶 的 ID 清單，以授權這些帳戶還原手動快照。

Example與單一帳戶共用快照

下列範例可讓 AWS 帳戶 識別符`123456789012`還原名為 的資料庫叢集快照`cluster-3-snapshot`。

針對 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier cluster-3-snapshot \
--attribute-name restore \
--values-to-add 123456789012
```

在 Windows 中：

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier cluster-3-snapshot ^
--attribute-name restore ^
--values-to-add 123456789012
```

Example與多個帳戶共用快照

下列範例會啟用兩個 AWS 帳戶 識別符 111122223333和 444455556666，以還原名為 的資料庫叢集快照manual-cluster-snapshot1。

針對 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-add {"111122223333","444455556666"}
```

在 Windows 中：

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
--attribute-name restore ^
--values-to-add "[\"111122223333\",\"444455556666\"]"
```

> **Note**
>
> 使用 Windows 命令提示字元時，您必須在 JSON 程式碼中的雙引號 (") 開頭加上反斜線 (\)，以逸出雙引號。

若要列出 AWS 帳戶 已啟用以還原快照的 ，請使用 describe-db-cluster-snapshot-attributes AWS CLI 命令。

RDS API

您也可以 AWS 帳戶 使用 Amazon RDS API 與其他 共用手動資料庫叢集快照。若要這樣做，請呼叫 ModifyDBClusterSnapshotAttribute 作業。將 `restore` 指定為 `AttributeName`，並使用 `ValuesToAdd` 參數來新增 AWS 帳戶 的 ID 清單，以授權這些帳戶還原手動快照。

若要公開手動快照並供所有人還原 AWS 帳戶，請使用 值 `all`。不過，對於任何手動快照，如果含有您不想讓所有 `all` 取得的私人資訊，請小心不要新增 AWS 帳戶值。另外，也不要對加密快照指定 `all`，因為不支援將這種快照變成公有。

若要列出所有 AWS 帳戶 允許還原快照的 ，請使用 `DescribeDBClusterSnapshotAttributes` API 操作。

共用公有快照

您可以將未加密的手動快照以公有形式共用，讓所有 AWS 帳戶 都可使用此快照。確保在公開共用快照時，公用快照中不包含您的任何私人資訊。

公開共用快照時，其會授予所有 AWS 帳戶 許可權限複製快照並從中建立資料庫叢集。

您無需對其他帳戶所擁有的公有快照備份儲存支付費用。您只需為您擁有的快照付費。

若您複製公有快照，則您擁有該副本。您需要為快照副本的備份儲存支付費用。若你從公用快照建立一個資料庫叢集，則需支付該資料庫叢集的費用。如需 Amazon Aurora 定價資訊，請參閱 Aurora 定價頁面。

您僅能刪除您擁有的公用快照。如要刪除共用或公用的快照，則必須登入擁有該快照的 AWS 帳戶。

檢視其他 AWS 帳戶 所擁有的公有快照

您可在 Amazon RDS 主控台中快照頁面的公用索引標籤上檢視特定 AWS 區域 中其他賬戶擁有的公用快照。您的快照 (您帳戶擁有的快照) 不會顯示在此索引標籤上。

如要檢視公有快照

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Snapshots (快照)。
3. 選擇 Public (公有) 索引標籤。

   公有快照隨即顯示。您可在 Owner (擁有者) 資料欄中查看哪個帳戶擁有公有快照。

> **Note**
>
> 您可能需要選取 Public snapshots (公用快照) 清單右上角的齒輪圖示來修改頁面偏好設定，以查看此資料欄。

檢視您自己的公有快照

您可以使用下列 AWS CLI 命令 (僅限 Unix)，在特定 AWS 區域 中檢視 AWS 帳戶 擁有的公有快照。

```
aws rds describe-db-cluster-snapshots --snapshot-type public --include-public |
 grep account_number
```

若您有公有快照，則傳回的輸出類似下列範例。

```
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:myclustersnapshot1",
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:myclustersnapshot2",
```

從已棄用的資料庫引擎版本共用公有快照

不支援從已棄用的資料庫引擎版本還原或複製公有快照。若要讓現有不支援的公有快照可供還原或複製，請執行下列步驟：

1. 將快照標記為私有。
2. 還原快照。
3. 將還原的資料庫叢集升級至支援的引擎版本。
4. 建立新快照。
5. 公開重新共用快照。

共用加密快照

您可以共用已使用 AES-256 加密演算法來「靜態」加密的資料庫叢集快照，如 加密 Amazon Aurora 資源 所述。

共用加密快照有下列限制：

- 您無法將加密快照以公有形式共用。
- 如果快照的加密方式是使用共用快照的 AWS 帳戶 之預設 KMS 金鑰，則您無法共用此快照。

  如需有關 Amazon RDS AWS KMS 金鑰管理的詳細資訊，請參閱 AWS KMS key 管理。

若要解決預設 KMS 金鑰問題，請執行下列任務：

1. 建立客戶自管金鑰並授予存取權.
2. 從來源帳戶複製和共用快照.
3. 複製目標帳戶中的共用快照.

建立客戶自管金鑰並授予存取權

首先，在與加密資料庫叢集快照相同的 AWS 區域 中建立自訂 KMS 金鑰。建立客戶自管金鑰時，您可以為另一個 AWS 帳戶 提供存取權。

建立客戶自管金鑰並授予存取權

1. 從來源 AWS 帳戶 登入 AWS 管理主控台。
2. 開啟位於 https://console.aws.amazon.com/kms 的 AWS KMS 主控台。
3. 若要變更 AWS 區域，請使用頁面右上角的區域選取器。
4. 在導覽窗格中，選擇 Customer managed keys (客戶受管金鑰)。
5. 選擇建立金鑰。
6. 在設定金鑰頁面上：
    a. 對於金鑰類型，選取對稱。
    b. 對於金鑰用途，選取加密和解密。
    c. 展開 Advanced options (進階選項)。
    d. 對於金鑰材料來源，選取 KMS。
    e. 對於區域性，選取單一區域金鑰。
    f. 選擇下一步。
7. 在新增標籤頁面上：
    a. 對於別名，輸入 KMS 金鑰的顯示名稱，例如 **`share-snapshot`**。
    b. (選用) 輸入 KMS 金鑰的描述。
    c. (選用) 將標籤新增至您的 KMS 金鑰。
    d. 選擇下一步。
8. 在定義金鑰管理許可頁面上，選擇下一步。
9. 在定義金鑰使用許可頁面上：
    a. 對於其他 AWS 帳戶，選擇新增另一個 AWS 帳戶。

b. 輸入您要授予存取權之 AWS 帳戶 的 ID。

您可以對多個 AWS 帳戶 授予存取權。

c. 選擇下一步。

10. 檢閱您的 KMS 金鑰，然後選擇完成。

從來源帳戶複製和共用快照

接下來，您可以使用客戶自管金鑰將來源資料庫叢集快照複製到新的快照。然後與目標 AWS 帳戶 共用。

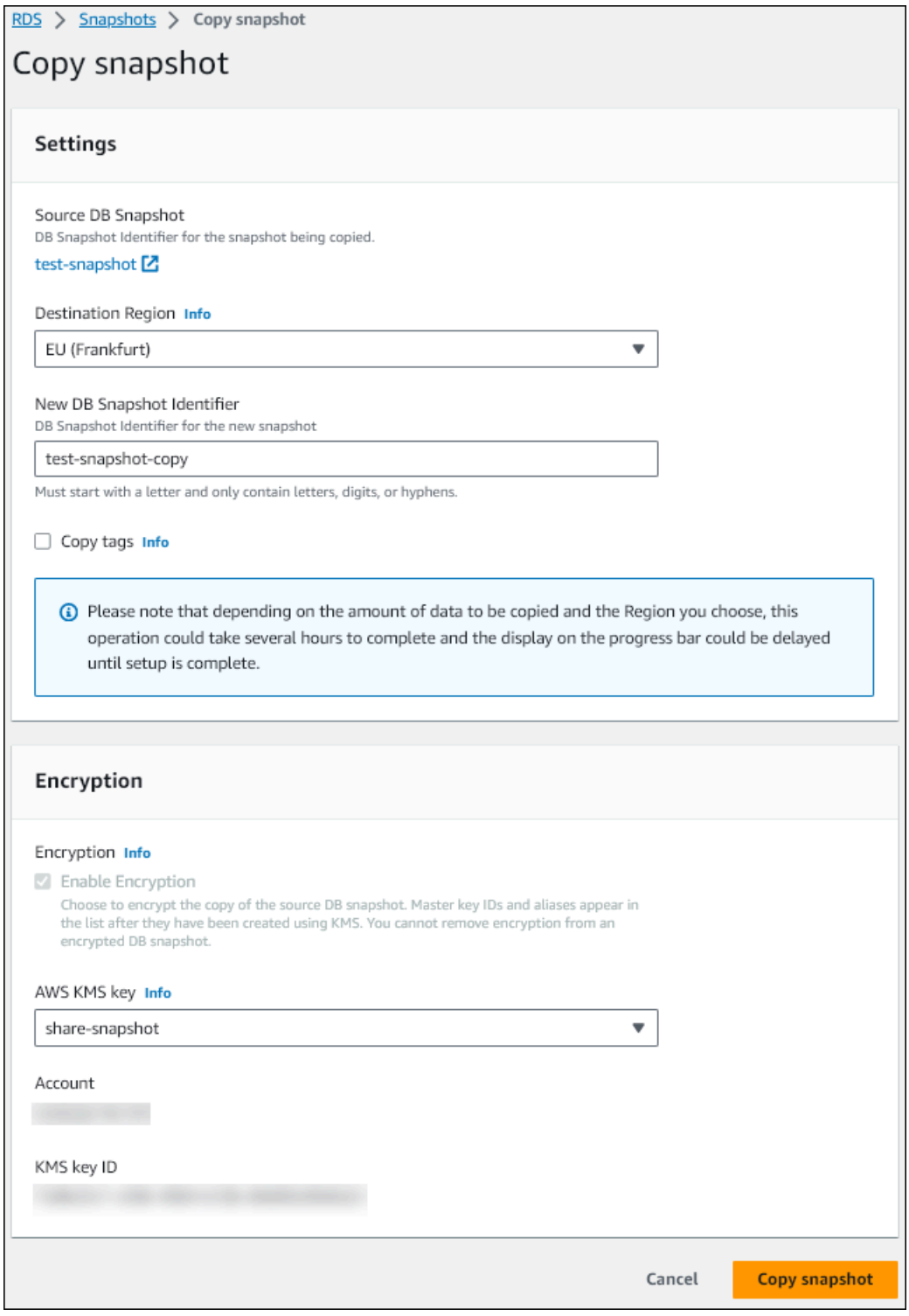
複製和共用快照

1. 從來源 AWS 帳戶 登入 AWS 管理主控台。
2. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
3. 在導覽窗格中，選擇 Snapshots (快照)。
4. 選取您要複製的資料庫叢集快照。
5. 針對 Actions (動作) 選擇 Copy snapshot (複製快照)。
6. 在複製快照頁面上：

a. 對於目的地區域，選擇您在先前程序中建立客戶自管金鑰所在的 AWS 區域。

b. 在 New DB Snapshot Identifier (新的資料庫快照識別符) 中輸入資料庫叢集快照副本的名稱。

c. 對於 AWS KMS key，選擇您建立的客戶自管金鑰。



d. 選擇 Copy Snapshot (複製快照)。

7. 當快照複本可用時，請選取它。
8. 針對 Actions (動作)，選擇 Share snapshot (共用快照)。
9. 在快照許可頁面上：

a. 輸入您要共用快照複本的 AWS 帳戶 ID，然後選擇新增。
b. 選擇儲存。

快照即會共用。

複製目標帳戶中的共用快照

現在，您可以在目標 AWS 帳戶 中複製共用快照。

複製共用快照

1. 從目標 AWS 帳戶 登入 AWS 管理主控台。
2. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
3. 在導覽窗格中，選擇 Snapshots (快照)。
4. 選擇與我共用索引標籤。
5. 選取共用快照。
6. 針對 Actions (動作) 選擇 Copy snapshot (複製快照)。
7. 選擇您複製快照的設定，如先前程序所示，但使用屬於目標帳戶的 AWS KMS key。

   選擇 Copy Snapshot (複製快照)。

停止快照共用

若要停止共用資料庫叢集快照，請從目標移除許可 AWS 帳戶。

主控台

停止與 共用手動資料庫叢集快照 AWS 帳戶

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Snapshots (快照)。
3. 選取您要停止共享的手動快照。
4. 選擇 Actions (動作)，然後選擇 Share snapshot (共用快照)。
5. 若要移除 的許可 AWS 帳戶，請從授權 AWS 帳戶清單中選擇刪除該帳戶的帳戶識別符。

6. 選擇儲存，以儲存變更。

CLI

若要從清單中移除 AWS 帳戶 識別符，請使用 `--values-to-remove` 參數。

Example停止快照共用

下列範例可防止 AWS 帳戶 ID 444455556666 還原快照。

針對 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-remove 444455556666
```

在 Windows 中：

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
--attribute-name restore ^
--values-to-remove 444455556666
```

RDS API

若要移除 的共用許可 AWS 帳戶，請使用 ModifyDBClusterSnapshotAttribute操作，並將 `AttributeName` 設定為 `restore`和 `ValuesToRemove` 參數。若要將手動快照標示為私有，請從 `all` 屬性的值清單移除 `restore` 值。

將資料庫叢集資料匯出至 Amazon S3

您可以將資料從即時 Amazon Aurora 資料庫叢集匯出至 Amazon S3 儲存貯體。匯出程序會在背景中執行，不會影響您作用中資料庫叢集的效能。

根據預設，會匯出資料庫叢集中的所有資料。但是，您可以選擇匯出特定資料庫、結構描述或資料表集。

Amazon Aurora 會複製資料庫叢集、從複製品中擷取資料，並將資料存放在 Amazon S3 儲存貯體中。資料會以壓縮且一致的 Apache Parquet 格式存放。個別 Parquet 檔案的大小通常約為 1–10 MB。

您可以透過匯出 Aurora MySQL 第 2 版和第 3 版的快照資料而取得的更快效能，不適用於匯出資料庫叢集資料。如需更多詳細資訊，請參閱 將資料庫叢集快照資料匯出至 Amazon S3。

無論您匯出全部或部分資料，都需支付匯出整個資料庫叢集的費用。如需詳細資訊，請參閱 Amazon Aurora 定價頁面。

匯出資料後，您可以直接透過 Amazon Athena 或 Amazon Redshift Spectrum 等工具分析匯出後的資料。有關使用 Athena 來讀取 Parquet 資料的詳細資訊，請參閱Amazon Athena 使用者指南中的 Parquet SerDe。如需有關使用 Redshift Spectrum 來讀取 Parquet 資料的詳細資訊，請參閱《Amazon Redshift 資料庫開發人員指南》中的從單欄式資料格式的 COPY。

功能可用性和支援會因每個資料庫引擎的特定版本以及 AWS 區域 而有所不同。如需將資料庫叢集資料匯出至 S3 功能之版本和區域可用性的詳細資訊，請參閱 將叢集資料匯出至 Amazon S3 的支援區域和 Aurora 資料庫引擎。

您可以使用下列程序，將資料庫叢集資料匯出至 Amazon S3 儲存貯體。如需詳細資訊，請參閱下列各節。

匯出資料庫叢集資料的概觀

1. 識別您要匯出其資料的資料庫叢集。
2. 設定對 Amazon S3 儲存貯體的存取。

   「儲存貯體」是 Amazon S3 物件或檔案的容器。如要提供存取儲存貯體的資訊，請採取下列步驟：

   a. 識別要匯出資料庫叢集資料的目標 S3 儲存貯體。S3 儲存貯體必須與資料庫叢集位於相同的 AWS 區域。如需更多詳細資訊，請參閱 識別要匯出的 Amazon S3 儲存貯體。

b. 建立 AWS Identity and Access Management (IAM) 角色，授予資料庫叢集匯出任務對 S3 儲存貯體的存取權。如需更多詳細資訊，請參閱 使用 IAM 角色提供對 Amazon S3 儲存貯體的存取權。

3. 建立用於伺服器端加密的對稱式加密 AWS KMS key。叢集匯出任務會在將匯出資料寫入至 S3 時，使用 KMS 金鑰來設定 AWS KMS 伺服器端加密。

   KMS 金鑰政策必須同時包含 `kms:CreateGrant` 和 `kms:DescribeKey` 許可。如需在 Amazon Aurora 中使用 KMS 金鑰的詳細資訊，請參閱 AWS KMS key 管理。

   如果您的 KMS 金鑰政策中有拒絕陳述式，請務必明確排除 AWS 服務主體 `export.rds.amazonaws.com`。

   您可以使用 AWS 帳戶內的 KMS 金鑰，或使用跨帳戶 KMS 金鑰。如需更多詳細資訊，請參閱 使用跨帳戶 AWS KMS key。

4. 使用主控台或 `start-export-task` CLI 命令，將資料庫叢集匯出至 Amazon S3。如需更多詳細資訊，請參閱 建立資料庫叢集匯出任務。

5. 若要存取 Amazon S3 儲存貯體中您匯出的資料，請參閱《Amazon Simple Storage Service 使用者指南》中的上傳、下載及管理物件。

在以下各節中了解如何設定、匯出、監控、取消資料庫叢集匯出任務，以及進行疑難排解。

主題

- 資料庫叢集匯出的考量事項
- 設定對 Amazon S3 儲存貯體的存取權
- 建立資料庫叢集匯出任務
- 監控資料庫叢集匯出任務
- 取消資料庫叢集匯出任務
- 對資料庫叢集匯出內容進行疑難排解

資料庫叢集匯出的考量事項

使用下列各節，來了解將資料庫叢集資料匯出至 Amazon S3 時的限制、檔案命名慣例，以及資料轉換和儲存。

主題

- 限制
- 檔案命名慣例
- 資料轉換和存放格式

限制

將資料庫叢集資料匯出至 Amazon S3 時有下列限制：

- 您無法同時針對相同的資料庫叢集執行多個匯出任務。這同時適用於完整和部分匯出。
- 您每個 AWS 帳戶最多可以有五個正在同時進行的資料庫快照匯出任務。
- Aurora Serverless v1 資料庫叢集不支援匯出至 S3。
- 對於已佈建引擎模式，Aurora MySQL 和 Aurora PostgreSQL 僅支援匯出至 S3。
- 匯出至 S3 不支援內含冒號 (:) 的 S3 字首。
- 在匯出過程中，S3 檔案路徑中的以下字元將轉換為底線 (_)：

```
\ ` " (space)
```

- 如果資料庫、結構描述或資料表的名稱中包含下列字元以外的字元，則不支援部分匯出。不過，您可以匯出整個資料庫叢集。
  - 拉丁字母 (A–Z)
  - 數字 (0–9)
  - 美元符號 ($)
  - 底線 (_)
- 資料庫資料表資料欄名稱不支援空格 ( ) 和某些字元。資料行名稱中具備下列字元的資料表會在匯出時跳過：

```
, ; { } ( ) \n \t = (space)
```

- 匯出時會略過名稱中具備斜線 (/) 的表格。
- 匯出期間，系統會略過 Aurora PostgreSQL 的暫存和未記錄資料表。
- 若資料包含接近或超過 500 MB 的大型物件 (例如 BLOB 或 CLOB)，則匯出會失敗。
- 如果資料表包含接近或大於 2 GB 的大型資料列，則在匯出期間會略過該資料表。
- 對於部分匯出，`ExportOnly` 清單的大小上限為 200 KB。

- 強烈建議您對每個匯出任務使用唯一的名稱。如果不使用唯一的任務名稱，可能會收到下列錯誤訊息：

  ExportTaskAlreadyExistsFault：呼叫 StartExportTask 操作時出現錯誤 (ExportTaskAlreadyExists)：ID *xxxxx* 的匯出任務已存在。
- 由於某些資料表可能會略過，因此建議您在匯出之後驗證資料中的資料列和資料表計數。

檔案命名慣例

特定表格的匯出資料會以 *base_prefix*/*files* 的格式儲存，其中基本前綴如下：

```
export_identifier/database_name/schema_name.table_name/
```

例如：

```
export-1234567890123-459/rdststcluster/mycluster.DataInsert_7ADB5D19965123A2/
```

輸出檔案使用下列命名慣例，其中 *partition_index* 是英數字元：

```
partition_index/part-00000-random_uuid.format-based_extension
```

例如：

```
1/part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet
    a/part-00000-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet
```

檔案命名慣例可能會有所變更。因此，在讀取目標資料表時，建議您讀取資料表基本字首內的所有內容。

資料轉換和存放格式

當您將資料庫叢集匯出至 Amazon S3 儲存貯體時，Amazon Aurora 會以 Parquet 格式轉換、匯出以及存放資料。如需更多詳細資訊，請參閱 匯出至 Amazon S3 儲存貯體時的資料轉換。

設定對 Amazon S3 儲存貯體的存取權

您識別 Amazon S3 儲存貯體，然後許可資料庫叢集匯出任務存取該儲存貯體。

主題

- 識別要匯出的 Amazon S3 儲存貯體
- 使用 IAM 角色提供對 Amazon S3 儲存貯體的存取權
- 使用跨帳戶 Amazon S3 儲存貯體

識別要匯出的 Amazon S3 儲存貯體

識別要將資料庫叢集資料匯出至其中的目標 Amazon S3 儲存貯體。使用現有的 S3 儲存貯體或建立新的 S3 儲存貯體。

> **Note**
>
> S3 儲存貯體必須與資料庫叢集位於相同的 AWS 區域。

如需使用 Amazon S3 儲存貯體的詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的下列內容：

- 如何檢視 S3 儲存貯體的屬性？
- 如何啟用 Amazon S3 儲存貯體的預設加密？
- 如何建立 S3 儲存貯體？

使用 IAM 角色提供對 Amazon S3 儲存貯體的存取權

在您將資料庫叢集資料匯出至 Amazon S3 前，請給與匯出任務對 Amazon S3 儲存貯體的寫入存取許可。

若要授予此許可，請建立 IAM 政策，提供儲存貯體的存取權，然後建立 IAM 角色並將該政策附加至其中。稍後，您可以將 IAM 角色指派給資料庫叢集匯出任務。

> **Important**
>
> 如果您打算使用 AWS 管理主控台 匯出資料庫叢集，您可以選擇在匯出資料庫叢集時自動建立 IAM 政策和角色。如需說明，請參閱建立資料庫叢集匯出任務。

給與任務 Amazon S3 的存取權

1. 建立 IAM 政策。此政策會提供儲存貯體和物件許可，允許您的資料庫叢集匯出任務存取 Amazon S3。

   在政策中，包含下列必要動作，以允許將檔案從 Amazon Aurora 傳輸至 S3 儲存貯體：

   - `s3:PutObject*`
   - `s3:GetObject*`
   - `s3:ListBucket`
   - `s3:DeleteObject*`
   - `s3:GetBucketLocation`

   在政策中，包含下列資源，以識別 S3 儲存貯體和該儲存貯體中的物件。以下資源清單會顯示用於存取 Amazon S3 的 Amazon Resource Name (ARN) 格式。

   - `arn:aws:s3:::amzn-s3-demo-bucket`
   - `arn:aws:s3:::amzn-s3-demo-bucket/*`

   如需為 Amazon Aurora 建立 IAM 政策的詳細資訊，請參閱 建立並使用 IAM 政策進行 IAM 資料庫存取。另請參閱《IAM 使用者指南》中的教學：建立和連接您的第一個客戶受管原則。

   下列 AWS CLI 命令會使用這些選項建立名為 `ExportPolicy`的 IAM 政策。其將存取權授予名為 *amzn-s3-demo-bucket* 的儲存貯體。

   > **Note**
   >
   > 在您建立政策後，請記下政策的 ARN。在後續步驟中將政策附加至 IAM 角色時，您會需要此 ARN。

```
aws iam create-policy  --policy-name ExportPolicy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExportPolicy",
            "Effect": "Allow",
```

```
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}'
```

2. 建立 IAM 角色，讓 Aurora 可以代表您擔任此 IAM 角色，以存取您的 Amazon S3 儲存貯體。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以將許可委派給 IAM 使用者。

   下列範例顯示使用 AWS CLI 命令來建立名為 的角色rds-s3-export-role。

```
aws iam create-role  --role-name rds-s3-export-role  --assume-role-policy-document
 '{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
           "Service": "export.rds.amazonaws.com"
         },
        "Action": "sts:AssumeRole"
      }
    ]
  }'
```

3. 將您建立的 IAM 政策附加至您建立的 IAM 角色。

   下列 AWS CLI 命令會將先前建立的政策連接至名為 的角色rds-s3-export-role。將 *your-policy-arn* 取代成您在稍早步驟中記下的政策 ARN。

```
aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
export-role
```

使用跨帳戶 Amazon S3 儲存貯體

您可以跨 AWS 帳戶使用 S3 儲存貯體。如需詳細資訊，請參閱使用跨帳戶 Amazon S3 儲存貯體。

建立資料庫叢集匯出任務

建立匯出任務，將資料從 Aurora 資料庫叢集匯出至 Amazon S3 儲存貯體。您每個 AWS 帳戶最多可以有五個正在同時進行的資料庫叢集匯出任務。

> **Note**
>
> 匯出資料庫叢集資料可能需要一段時間，取決於您的資料庫類型和大小。匯出任務會先複製和擴展整個資料庫，然後再將資料擷取到 Amazon S3。此階段期間的工作進度會顯示為STARTING (開始)。當任務切換到將資料匯出到 S3 時，進度會顯示為 In progress (進行中)。
>
> 匯出完成所需的時間取決於儲存在資料庫中的資料。例如，如果資料表有分散均勻的數值主索引鍵或索引資料欄，則匯出速度最快。未包含適用於資料分割之資料欄的資料表，以及在字串型資料欄上只有一個索引的資料表，因為匯出使用較慢的單一執行緒程序，所以將會需要更長的時間。

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API，將資料庫叢集資料匯出至 Amazon S3。

如果您使用 Lambda 函數匯出資料庫叢集資料，請將 `kms:DescribeKey` 動作新增至 Lambda 函數政策。如需詳細資訊，請參閱 AWS Lambda 許可。

主控台

Export to Amazon S3 (匯出至 Amazon S3) 主控台選項僅會針對可匯出至 Amazon S3 的資料庫叢集顯示。由於下列原因，資料庫叢集可能無法匯出：

- S3 匯出不支援此資料庫引擎。
- S3 匯出不支援資料庫叢集版本。
- 已建立資料庫叢集的 AWS 區域不支援 S3 匯出。

匯出資料庫叢集資料

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。

2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您要匯出其資料的資料庫叢集。
4. 針對 Actions (動作)，選擇 Export to Amazon S3 (匯出至 Amazon S3)。

   隨即出現 Export to Amazon S3 (匯出至 Amazon S3) 視窗。
5. 針對 Export identifier (匯出識別符)，輸入名稱以識別匯出任務。這個值也會用來做為在 S3 儲存貯體中建立的檔案名稱。
6. 選擇匯出的資料：

   - 選擇 All (全部) 來匯出資料庫叢集中的所有資料。
   - 選擇 Partial (部分) 來匯出資料庫叢集的特定部分。若要識別要匯出的叢集部分，請針對 Identifier (識別符) 輸入一或多個資料庫、結構描述或表格，並以空格分隔。

     使用下列格式：

     ```
     database[.schema][.table] database2[.schema2][.table2] ... databasen[.schemaN]
     [.tablen]
     ```

     例如：

     ```
     mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1
      mydatabase2.myschema2.mytable2
     ```
7. 針對 S3 bucket (S3 儲存貯體)，選擇要匯出的儲存貯體。

   如要將匯出資料指派給 S3 儲存貯體中的資料夾路徑，請針對 S3 prefix (S3 字首) 輸入選用的路徑。
8. 針對 IAM role (IAM 角色)，您可選擇授予對您所選擇 S3 儲存貯體寫入存取權限的角色，或建立新角色。

   - 如果您透過遵循 使用 IAM 角色提供對 Amazon S3 儲存貯體的存取權 中的步驟建立了角色，請選擇該角色。
   - 如果您沒有建立授予您所選擇 S3 儲存貯體寫入存取權限的角色，請選擇 Create a new role (建立新角色) 以自動建立角色。接下來，為 IAM 角色名稱 (IAM role name) 中的角色輸入名稱。
9. 針對 KMS key (KMS 金鑰)，輸入金鑰的 ARN 以加密匯出的資料。
10. 選擇 Export to Amazon S3 (匯出至 Amazon S3)。

AWS CLI

若要使用 AWS CLI 將資料庫叢集資料匯出至 Amazon S3，請搭配下列必要選項使用 start-export-task 命令：

- `--export-task-identifier`
- `--source-arn` – 資料庫叢集的 Amazon Resource Name (ARN)。
- `--s3-bucket-name`
- `--iam-role-arn`
- `--kms-key-id`

在下列範例中，匯出任務名為 *my-cluster-export*，其會將資料匯出至名為 *amzn-s3-demo-destination-bucket* 的 S3 儲存貯體。

Example

對於 Linux、macOS 或 Unix：

```
aws rds start-export-task \
    --export-task-identifier my-cluster-export \
    --source-arn arn:aws:rds:us-west-2:123456789012:cluster:my-cluster \
    --s3-bucket-name amzn-s3-demo-destination-bucket \
    --iam-role-arn iam-role \
    --kms-key-id my-key
```

在 Windows 中：

```
aws rds start-export-task ^
    --export-task-identifier my-DB-cluster-export ^
    --source-arn arn:aws:rds:us-west-2:123456789012:cluster:my-cluster ^
    --s3-bucket-name amzn-s3-demo-destination-bucket ^
    --iam-role-arn iam-role ^
    --kms-key-id my-key
```

範例輸出如下。

```
{
    "ExportTaskIdentifier": "my-cluster-export",
    "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:my-cluster",
    "S3Bucket": "amzn-s3-demo-destination-bucket",
```

```
    "IamRoleArn": "arn:aws:iam:123456789012:role/ExportTest",
    "KmsKeyId": "my-key",
    "Status": "STARTING",
    "PercentProgress": 0,
    "TotalExtractedDataInGB": 0,
}
```

若要為資料庫叢集匯出提供 S3 儲存貯體中的資料夾路徑，請在 start-export-task 命令中包含 `--s3-prefix` 選項。

RDS API

若要使用 Amazon RDS API 將資料庫叢集資料匯出至 Amazon S3，請搭配下列必要參數使用 StartExportTask 操作：

- `ExportTaskIdentifier`
- `SourceArn` – 資料庫叢集的 ARN。
- `S3BucketName`
- `IamRoleArn`
- `KmsKeyId`

監控資料庫叢集匯出任務

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 監控資料庫叢集匯出。

主控台

監控資料庫叢集匯出

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Exports in Amazon S3 (在 Amazon S3 中匯出)。

   資料庫叢集匯出會在 Source type (來源類型) 欄中指出。匯出狀態會顯示在 Status (狀態) 欄中。
3. 若要檢視特定資料庫叢集匯出的詳細資訊，請選擇匯出任務。

AWS CLI

若要使用 AWS CLI 監控資料庫叢集匯出任務，請使用 describe-export-tasks 命令。

以下範例示範如何顯示您所有資料庫叢集匯出的目前資訊。

Example

```
aws rds describe-export-tasks

{
    "ExportTasks": [
        {
            "Status": "CANCELED",
            "TaskEndTime": "2022-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "S3Bucket": "amzn-s3-demo-bucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/K7MDENG/
bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:parameter-groups-
test"
        },
{
            "Status": "COMPLETE",
            "TaskStartTime": "2022-10-31T20:58:06.998Z",
            "TaskEndTime": "2022-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general
\":[{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
            "S3Bucket": "amzn-s3-demo-bucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "SourceArn": "arn:aws:rds:us-
west-2:123456789012:cluster:example-1-2019-10-31-06-44"
        },
        {
            "Status": "FAILED",
            "TaskEndTime": "2022-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket amzn-s3-demo-bucket2 isn't located in the
 current AWS Region. Please, review your S3 bucket name and retry the export.",
```

```
            "S3Prefix": "",
            "S3Bucket": "amzn-s3-demo-bucket2",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "SourceArn": "arn:aws:rds:us-
west-2:123456789012:cluster:example-1-2019-10-30-06-45"
        }
    ]
}
```

若要顯示特定匯出任務的相關資訊，請在 `--export-task-identifier` 命令中包含 `describe-export-tasks` 選項。如要篩選輸出，請包含 `--Filters` 選項。如需更多選項，請參閱 describe-export-tasks 命令。

RDS API

若要使用 Amazon RDS API 顯示資料庫叢集匯出的相關資訊，請使用 DescribeExportTasks 操作。

若要追蹤匯出工作流程的完成或起始其他工作流程，您可以訂閱 Amazon Simple Notification Service 主題。如需 Amazon SNS 的詳細資訊，請參閱使用 Amazon RDS 事件通知。

取消資料庫叢集匯出任務

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 取消資料庫叢集匯出任務。

> **Note**
>
> 取消匯出匯出任務不會移除任何已匯出到 Amazon S3 的資料。如需如何使用主控台刪除資料的資訊，請參閱如何從 S3 儲存貯體刪除物件？ 如要使用 CLI 刪除資料，請使用 delete-object 命令。

主控台

取消資料庫叢集匯出任務

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。

2. 在導覽窗格中，選擇 Exports in Amazon S3 (在 Amazon S3 中匯出)。

   資料庫叢集匯出會在 Source type (來源類型) 欄中指出。匯出狀態會顯示在 Status (狀態) 欄中。
3. 選擇您想要取消的匯出任務。
4. 選擇 Cancel (取消)。
5. 選擇確認頁面上的 Cancel export task (取消匯出任務)。

AWS CLI

若要使用 AWS CLI 取消匯出任務，請使用 cancel-export-task 命令。命令需要 `--export-task-identifier` 選項。

Example

```
aws rds cancel-export-task --export-task-identifier my-export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "S3Bucket": "amzn-s3-demo-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my-export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:export-example-1"
}
```

RDS API

若要使用 Amazon RDS API 取消匯出任務，請搭配 `ExportTaskIdentifier` 參數使用 CancelExportTask 操作。

對資料庫叢集匯出內容進行疑難排解

使用下列各節，來協助針對資料庫叢集匯出任務到 Amazon S3 的失敗訊息和 PostgreSQL 許可錯誤進行疑難排解。

Amazon S3 匯出任務的失敗訊息

下表說明 Amazon S3 匯出任務失敗時傳回的訊息。

失敗訊息	描述
無法找到或存取來源資料庫叢集:[叢集名稱]	無法複製來源資料庫叢集。
發生未知的內部錯誤。	因為不明的錯誤、例外或故障，所以任務失敗。
將匯出任務的中繼資料寫入 S3 儲存貯體 [儲存貯體名稱] 時發生未知的內部錯誤。	因為不明的錯誤、例外或故障，所以任務失敗。
RDS 匯出無法寫入匯出任務的中繼資料，因為無法擔任 IAM 角色 [角色 ARN]。	匯出任務會擔任您的 IAM 角色，以驗證是否允許將中繼資料寫入 S3 儲存貯體。如果任務無法擔任您的 IAM 角色，則會失敗。
RDS 匯出無法使用具有 KMS 金鑰 [金鑰 ID] 的 IAM 角色 [角色 ARN]，將匯出任務的中繼資料寫入 S3 儲存貯體 [儲存貯體名稱]。錯誤代碼：[錯誤代碼]	缺少一或多個許可，因此匯出任務無法存取 S3 儲存貯體。收到下列其中一個錯誤碼時會引發此失敗訊息： • AWSSecurityTokenServiceException 和錯誤代碼 AccessDenied • AmazonS3Exception 和錯誤代碼 NoSuchBucket 、AccessDenied 、KMS.KMSInvalidStateException 、403 Forbidden 或 KMS.DisabledException 這些錯誤碼表示 IAM 角色、S3 儲存貯體或 KMS 金鑰的設定錯誤。
IAM 角色 [角色 ARN] 未獲得授權在 S3 儲存貯體 [儲存貯體名稱] 上呼叫 [S3 動作]。檢閱您的許可，然後重試匯出。	IAM 政策設定錯誤。缺少 S3 儲存貯體上特定 S3 動作的許可，這會導致匯出任務失敗。
KMS 金鑰檢查失敗。請檢查 KMS 金鑰上的憑證，然後再試一次。	KMS 金鑰憑證檢查失敗。
S3 憑證檢查失敗。檢查您的 S3 儲存貯體與 IAM 政策的許可。	S3 憑證檢查失敗。

失敗訊息	描述
S3 儲存貯體 [儲存貯體名稱] 無效。可能不是位於目前的 AWS 區域或不存在。檢閱您的 S3 儲存貯體名稱並重試匯出。	S3 儲存貯體無效。
S3 儲存貯體 [儲存貯體名稱] 並非位於目前的 AWS 區域。檢閱您的 S3 儲存貯體名稱並重試匯出。	S3 儲存貯體位於錯誤的 AWS 區域。

對 PostgreSQL 許可錯誤進行故障診斷

將 PostgreSQL 資料庫匯出至 Amazon S3 時，您可能會看到 PERMISSIONS_DO_NOT_EXIST 錯誤，指出某些資料表已略過。您在建立資料庫叢集時指定的超級使用者，若沒有存取這些資料表的許可，通常就會發生此錯誤。

若要修正此錯誤，請執行下列命令：

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

如需超級使用者權限的詳細資訊，請參閱主要使用者帳戶權限。

將資料庫叢集快照資料匯出至 Amazon S3

您可將資料庫叢集快照資料匯出至 Amazon S3 儲存貯體。匯出程序會在背景中執行，不會影響您作用中資料庫叢集的效能。

當您匯出資料庫叢集快照時，Amazon Aurora 會從快照擷取資料，並將其存放於 Amazon S3 儲存貯體中。您可以匯出手動快照和自動化系統快照。根據預設會匯出快照中的所有資料。但是，您可以選擇匯出特定資料庫、結構描述或資料表集。

> **Note**
>
> 從資料庫叢集快照匯出資料需要還原快照。還原時間會受到各種因素的影響，例如 AWS 區域接收相對於其可用頻寬的網路流量。當流量突然增加時，您可能會遇到比預期更長的完成時間。
>
> 降低 Aurora 資料庫 S3 匯出時間的替代方案是將實際資料庫叢集匯出至 S3。資料庫叢集匯出的開始時間比資料庫快照匯出更短，因為不需要還原快照。如需更多詳細資訊，請參閱 將資料庫叢集資料匯出至 Amazon S3。

資料會以壓縮且一致的 Apache Parquet 格式存放。個別 Parquet 檔案的大小通常約為 1–10 MB。

匯出資料後，您可以直接透過 Amazon Athena 或 Amazon Redshift Spectrum 等工具分析匯出後的資料。有關使用 Athena 來讀取 Parquet 資料的詳細資訊，請參閱Amazon Athena 使用者指南中的 Parquet SerDe。如需有關使用 Redshift Spectrum 來讀取 Parquet 資料的詳細資訊，請參閱《Amazon Redshift 資料庫開發人員指南》中的從單欄式資料格式的 COPY。

功能可用性和支援會因每個資料庫引擎的特定版本以及 AWS 區域 而有所不同。如需將資料庫叢集快照資料匯出至 S3 功能之版本和區域可用性的詳細資訊，請參閱 將快照資料匯出至 Amazon S3 的支援區域和 Aurora 資料庫引擎。

您可以使用下列程序，將資料庫快照資料匯出至 Amazon S3 儲存貯體。如需詳細資訊，請參閱下列各節。

匯出快照資料概觀

1. 識別要匯出的快照

   使用現有的自動化或手動快照，或是建立資料庫執行個體的手動快照。
2. 設定對 Amazon S3 儲存貯體的存取。

「儲存貯體」是 Amazon S3 物件或檔案的容器。如要提供存取儲存貯體的資訊，請採取下列步驟：

a. 識別要匯出快照的目標 S3 儲存貯體。S3 儲存貯體必須位於和快照相同的 AWS 區域。如需更多詳細資訊，請參閱 識別要匯出的 Amazon S3 儲存貯體。

b. 建立 AWS Identity and Access Management (IAM) 角色，授予快照會出任務對 S3 儲存貯體的存取權限。如需更多詳細資訊，請參閱 使用 IAM 角色提供對 Amazon S3 儲存貯體的存取權。

3. 建立用於伺服器端加密的對稱式加密 AWS KMS key。快照匯出任務會在將匯出資料寫入 S3 時，使用 KMS 金鑰來設定 AWS KMS 伺服器端加密。

   KMS 金鑰政策必須同時包含 `kms:CreateGrant` 和 `kms:DescribeKey` 許可。如需在 Amazon Aurora 中使用 KMS 金鑰的詳細資訊，請參閱 AWS KMS key 管理。

   如果您的 KMS 金鑰政策中有拒絕陳述式，請務必明確排除 AWS 服務主體 `export.rds.amazonaws.com`。

   您可以使用 AWS 帳戶內的 KMS 金鑰，或使用跨帳戶 KMS 金鑰。如需更多詳細資訊，請參閱 使用跨帳戶 AWS KMS key。

4. 使用主控台或 `start-export-task` CLI 命令將快照匯出至 Amazon S3。如需更多詳細資訊，請參閱 建立快照匯出任務。

5. 若要存取 Amazon S3 儲存貯體中您匯出的資料，請參閱《Amazon Simple Storage Service 使用者指南》中的上傳、下載及管理物件。

在以下各節中了解如何設定、匯出、監控、取消和故障診斷資料庫叢集快照匯出任務。

主題

- 資料庫叢集快照匯出的考量
- 設定對 Amazon S3 儲存貯體的存取權
- 建立快照匯出任務
- 監控快照匯出
- 取消快照匯出任務
- Aurora MySQL 中的匯出效能
- 故障診斷快照匯出

資料庫叢集快照匯出的考量

限制

將資料庫快照資料匯出至 Amazon S3 時有下列限制：

- 您無法同時針對相同的資料庫叢集快照執行多個匯出任務。這同時適用於完整和部分匯出。
- 每個 最多可以有五個同時進行的資料庫快照匯出任務 AWS 帳戶。
- 您無法從 Aurora Serverless v1 資料庫叢集將快照資料匯出至 S3。
- 匯出至 S3 不支援包含冒號 (:) 的 S3 字首。
- 在匯出過程中，S3 檔案路徑中的以下字元將轉換為底線 (_)：

```
\ ` " (space)
```

- 如果資料庫、結構描述或資料表的名稱中包含下列字元以外的字元，則不支援部分匯出。但是，您可以匯出整個資料庫快照。
  - 拉丁字母 (A–Z)
  - 數字 (0–9)
  - 美元符號 ($)
  - 底線 (_)
- 資料庫資料表資料欄名稱不支援空格 ( ) 和某些字元。資料行名稱中具備下列字元的資料表會在匯出時跳過：

```
, ; { } ( ) \n \t = (space)
```

- 匯出時會略過名稱中具備斜線 (/) 的表格。
- 匯出期間，系統會略過 Aurora PostgreSQL 的暫存和未記錄資料表。
- 若資料包含接近或超過 500 MB 的大型物件 (例如 BLOB 或 CLOB)，則匯出會失敗。
- 如果資料表包含接近或大於 2 GB 的大型資料列，則在匯出期間會略過該資料表。
- 對於部分匯出，`ExportOnly` 清單的大小上限為 200 KB。
- 強烈建議您對每個匯出任務使用唯一的名稱。如果不使用唯一的任務名稱，可能會收到下列錯誤訊息：

  ExportTaskAlreadyExistsFault：呼叫 StartExportTask 操作時出現錯誤 (ExportTaskAlreadyExists)：ID *xxxxx* 的匯出任務已存在。

- 您在將快照資料匯出至 S3 時可以刪除該快照，但在匯出任務完成之前，仍會向您收取該快照的儲存費用。
- 您無法將匯出的快照資料從 S3 還原到新的資料庫叢集。

檔案命名慣例

特定表格的匯出資料會以 *base_prefix*/*files* 的格式儲存，其中基本前綴如下：

```
export_identifier/database_name/schema_name.table_name/
```

例如：

```
export-1234567890123-459/rdststdb/rdststdb.DataInsert_7ADB5D19965123A2/
```

檔案命名方式有兩種慣例：

- 目前慣例：

  ```
  batch_index/part-partition_index-random_uuid.format-based_extension
  ```

  批次索引是序號，代表從資料表讀取的資料批次。如果我們無法將您的資料表分割成要平行匯出的小型區塊，將會有多個批次索引。如果您的資料表分割為多個資料表，則會發生相同的情況。此時會有多個批次索引，一個用於主資料表的每個資料表分割區。

  如果我們可以將您的資料表分割成要平行讀取的小區塊，則只會有批次索引 1 資料夾。

  在批次索引資料夾中，有一或多個 Parquet 檔案包含資料表的資料。Parquet 檔案名稱的字首為 `part-`*partition_index*。如果您的資料表已分割，則會有多個以分割區索引 `00000` 開頭的檔案。

  分割區索引序列中可能會有間隙。這是因為每個分割區是從資料表中的範圍查詢取得。如果該分割區的範圍內沒有資料，則會略過該序號。

  例如，假設 `id` 欄是資料表的主索引鍵，且其最小值和最大值為 `100` 和 `1000`。當我們嘗試使用九個分割區匯出此資料表時，我們會使用平行查詢來讀取它，如下所示：

  ```
  SELECT * FROM table WHERE id <= 100 AND id < 200
   SELECT * FROM table WHERE id <= 200 AND id < 300
  ```

這應該會產生從 part-00000-*random_uuid*.gz.parquet 到 part-00008-*random_uuid*.gz.parquet 的九個檔案。不過，如果沒有 ID 介於 200 和 350 之間的資料列，則其中一個已完成的分割區是空的，而且不會為其建立檔案。在上一個範例中，part-00001-*random_uuid*.gz.parquet 不會建立。

- 較舊的慣例：

```
part-partition_index-random_uuid.format-based_extension
```

這與目前的慣例相同，但沒有 *batch_index* 字首，例如：

```
part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet
 part-00001-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet
 part-00002-f5a991ab-59aa-7fa6-3333-d41eccd340a7-c000.gz.parquet
```

檔案命名慣例可能會有所變更。因此，在讀取目標資料表時，建議您讀取資料表基本字首內的所有內容。

匯出至 Amazon S3 儲存貯體時的資料轉換

當您將資料庫快照匯出至 Amazon S3 儲存貯體時，Amazon Aurora 會以 Parquet 格式轉換資料、匯出資料，以及存放資料。如需 Parquet 的詳細資訊，請參閱 Apache Parquet 網站。

Parquet 會將所有資料以下列其中一種基本類型存放：

- BOOLEAN
- INT32
- INT64
- INT96
- FLOAT
- DOUBLE
- BYTE_ARRAY – 長度可變的位元組陣列，也稱為二進位
- FIXED_LEN_BYTE_ARRAY – 長度固定的位元組陣列，用於值具備固定大小時

Parquet 的資料類型相當少，可減少讀取和寫入格式的複雜性。Parquet 提供擴充基本類型的邏輯類型。「邏輯類型」會實作為標註，並將資料存放在 `LogicalType` 中繼資料欄位中。邏輯類型標註會說明如何解譯基本類型。

當 `STRING` 邏輯類型標註 `BYTE_ARRAY` 類型時，會指出位元組陣列應解譯為 UTF-8 編碼的字元字串。匯出任務完成後，Amazon Aurora 會通知您是否發生任何字串轉換。匯出的基礎資料一律與來源資料相同。但是，由於 UTF-8 中的編碼存在差異，在 Athena 等工具中讀取時，有些字元的顯示方式可能會和來源中的顯示方式不同。

如需詳細資訊，請參閱 Parquet 文件中的 Parquet Logical Type Definitions。

主題

- MySQL 資料類型映射至 Parquet
- PostgreSQL 資料類型對 Parquet 的映射

MySQL 資料類型映射至 Parquet

下表顯示轉換資料並匯出至 Amazon S3 時，從 MySQL 資料類型到 Parquet 資料類型的映射。

來源資料類型	Parquet 基本類型	邏輯類型標註	轉換備註
數值資料類型			
BIGINT	INT64		
BIGINT UNSIGNED	FIXED_LEN _BYTE_ARRAY(9)	DECIMAL(20,0)	Parquet 只支援帶正負號的類型，因此映射需要額外的位元組 (8 加 1) 來存放 BIGINT_UNSIGNED 類型。
BIT	BYTE_ARRAY		
DECIMAL	INT32	DECIMAL(p,s)	如果來源值小於 2^{31}，則會以 INT32 存放。

來源資料類型	Parquet 基本類型	邏輯類型標註	轉換備註
	INT64	DECIMAL(p,s)	如果來源值等於或大於 2^{31}，但小於 2^{63}，則會以 INT64 存放。
	FIXED_LEN _BYTE_ARRAY(N)	DECIMAL(p,s)	如果來源值等於或大於 2^{63}，則會以 FIXED_LEN _BYTE_ARRAY(N) 存放。
	BYTE_ARRAY	STRING	Parquet 不支援大於 38 的 Decimal 精確度。Decimal 值會轉換成 BYTE_ARRAY 類型中的字串，並以 UTF8 編碼。
DOUBLE	DOUBLE		
FLOAT	DOUBLE		
INT	INT32		
INT UNSIGNED	INT64		
MEDIUMINT	INT32		
MEDIUMINT UNSIGNED	INT64		
NUMERIC	INT32	DECIMAL(p,s)	如果來源值小於 2^{31}，則會以 INT32 存放。
	INT64	DECIMAL(p,s)	如果來源值等於或大於 2^{31}，但小於 2^{63}，則會以 INT64 存放。

來源資料類型	Parquet 基本類型	邏輯類型標註	轉換備註
	FIXED_LEN _ARRAY(N)	DECIMAL(p,s)	如果來源值等於或大於 2^{63}，則會以 FIXED_LEN _BYTE_ARRAY(N) 存放。
	BYTE_ARRAY	STRING	Parquet 不支援大於 38 的 Numeric 精確度。這個 Numeric 值會轉換成 BYTE_ARRAY 類型中的字串，並以 UTF8 編碼。
SMALLINT	INT32		
SMALLINT UNSIGNED	INT32		
TINYINT	INT32		
TINYINT UNSIGNED	INT32	INT(16, true)	
字串資料類型			
BINARY	BYTE_ARRAY		
BLOB	BYTE_ARRAY		
CHAR	BYTE_ARRAY		
ENUM	BYTE_ARRAY	STRING	
LINESTRING	BYTE_ARRAY		
LONGBLOB	BYTE_ARRAY		
LONGTEXT	BYTE_ARRAY	STRING	

來源資料類型	Parquet 基本類型	邏輯類型標註	轉換備註
MEDIUMBLOB	BYTE_ARRAY		
MEDIUMTEXT	BYTE_ARRAY	STRING	
MULTILINESTRING	BYTE_ARRAY		
SET	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	
TINYBLOB	BYTE_ARRAY		
TINYTEXT	BYTE_ARRAY	STRING	
VARBINARY	BYTE_ARRAY		
VARCHAR	BYTE_ARRAY	STRING	
日期和時間資料類型			
DATE	BYTE_ARRAY	STRING	日期會轉換成 BYTE_ARRAY 類型中的字串，並以 UTF8 編碼。
DATETIME	INT64	TIMESTAMP_MICROS	
TIME	BYTE_ARRAY	STRING	TIME 類型會轉換成 BYTE_ARRAY 類型中的字串，並以 UTF8 編碼。
TIMESTAMP	INT64	TIMESTAMP_MICROS	
YEAR	INT32		
幾何資料類型			

來源資料類型	Parquet 基本類型	邏輯類型標註	轉換備註
GEOMETRY	BYTE_ARRAY		
GEOMETRYCOLLECTION	BYTE_ARRAY		
MULTIPOINT	BYTE_ARRAY		
MULTIPOLYGON	BYTE_ARRAY		
POINT	BYTE_ARRAY		
POLYGON	BYTE_ARRAY		
JSON 資料類型			
JSON	BYTE_ARRAY	STRING	

PostgreSQL 資料類型對 Parquet 的映射

下表顯示轉換資料並匯出至 Amazon S3 時，PostgreSQL 資料類型到 Parquet 資料類型的映射。

PostgreSQL 資料類型	Parquet 基本類型	邏輯類型標註	映射備註
數值資料類型			
BIGINT	INT64		
BIGSERIAL	INT64		
DECIMAL	BYTE_ARRAY	STRING	DECIMAL 類型會轉換成 BYTE_ARRAY 類型中的字串，並以 UTF8 編碼。 此轉換是為了避免因非數字 (NaN) 而導致

PostgreSQL 資料類型	Parquet 基本類型	邏輯類型標註	映射備註
			的資料精確度和資料值複雜性。
DOUBLE PRECISION	DOUBLE		
INTEGER	INT32		
MONEY	BYTE_ARRAY	STRING	
REAL	FLOAT		
SERIAL	INT32		
SMALLINT	INT32	INT(16, true)	
SMALLSERIAL	INT32	INT(16, true)	
字串和相關資料類型			
ARRAY	BYTE_ARRAY	STRING	陣列會轉換成字串，並以 BINARY (UTF8) 編碼。 此轉換是為了避免因非數字 (NaN) 和時間資料值而導致的資料精確度複雜性。
BIT	BYTE_ARRAY	STRING	
BIT VARYING	BYTE_ARRAY	STRING	
BYTEA	BINARY		
CHAR	BYTE_ARRAY	STRING	
CHAR(N)	BYTE_ARRAY	STRING	
ENUM	BYTE_ARRAY	STRING	

PostgreSQL 資料類型	Parquet 基本類型	邏輯類型標註	映射備註
NAME	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	
TEXT SEARCH	BYTE_ARRAY	STRING	
VARCHAR(N)	BYTE_ARRAY	STRING	
XML	BYTE_ARRAY	STRING	
日期和時間資料類型			
DATE	BYTE_ARRAY	STRING	
INTERVAL	BYTE_ARRAY	STRING	
TIME	BYTE_ARRAY	STRING	
TIME WITH TIME ZONE	BYTE_ARRAY	STRING	
TIMESTAMP	BYTE_ARRAY	STRING	
TIMESTAMP WITH TIME ZONE	BYTE_ARRAY	STRING	
幾何資料類型			
BOX	BYTE_ARRAY	STRING	
CIRCLE	BYTE_ARRAY	STRING	
LINE	BYTE_ARRAY	STRING	
LINESEGMENT	BYTE_ARRAY	STRING	
PATH	BYTE_ARRAY	STRING	
POINT	BYTE_ARRAY	STRING	

PostgreSQL 資料類型	Parquet 基本類型	邏輯類型標註	映射備註
POLYGON	BYTE_ARRAY	STRING	
JSON 資料類型			
JSON	BYTE_ARRAY	STRING	
JSONB	BYTE_ARRAY	STRING	
其他資料類型			
BOOLEAN	BOOLEAN		
CIDR	BYTE_ARRAY	STRING	網路資料類型
COMPOSITE	BYTE_ARRAY	STRING	
DOMAIN	BYTE_ARRAY	STRING	
INET	BYTE_ARRAY	STRING	網路資料類型
MACADDR	BYTE_ARRAY	STRING	
OBJECT IDENTIFIER	N/A		
PG_LSN	BYTE_ARRAY	STRING	
RANGE	BYTE_ARRAY	STRING	
UUID	BYTE_ARRAY	STRING	

設定對 Amazon S3 儲存貯體的存取權

識別出 Amazon S3 儲存貯體後，接著授予快照許可以存取該儲存貯體。

主題

- 識別要匯出的 Amazon S3 儲存貯體
- 使用 IAM 角色提供對 Amazon S3 儲存貯體的存取權
- 使用跨帳戶 Amazon S3 儲存貯體

- 使用跨帳戶 AWS KMS key

識別要匯出的 Amazon S3 儲存貯體

識別要匯出資料庫快照的目標 Amazon S3 儲存貯體。使用現有的 S3 儲存貯體或建立新的 S3 儲存貯體。

> **Note**
>
> 要匯出至 的 S3 儲存貯體必須與快照位於相同的 AWS 區域。

如需使用 Amazon S3 儲存貯體的詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的下列內容：

- 如何檢視 S3 儲存貯體的屬性？
- 如何啟用 Amazon S3 儲存貯體的預設加密？
- 如何建立 S3 儲存貯體？

使用 IAM 角色提供對 Amazon S3 儲存貯體的存取權

在您將資料庫快照資料匯出至 Amazon S3 前，請給予快照匯出任務對 Amazon S3 儲存貯體的存取許可。

若要授予此許可，請建立提供儲存貯體之存取權的 IAM 政策，然後建立 IAM 角色並將該政策附加至其中。稍後，您可以將 IAM 角色指派給快照匯出任務。

> **Important**
>
> 如果您打算使用 AWS 管理主控台 匯出快照，您可以選擇在匯出快照時自動建立 IAM 政策和角色。如需說明，請參閱建立快照匯出任務。

給予資料庫快照任務對 Amazon S3 的存取權限

1. 建立 IAM 政策。此政策會提供儲存貯體和物件許可，允許您的快照匯出任務存取 Amazon S3。

   在政策中，包含下列必要動作，以允許將檔案從 Amazon Aurora 傳輸至 S3 儲存貯體：

- s3:PutObject*
- s3:GetObject*
- s3:ListBucket
- s3:DeleteObject*
- s3:GetBucketLocation

在政策中，包含下列資源，以識別 S3 儲存貯體和該儲存貯體中的物件。以下資源清單會顯示用於存取 Amazon S3 的 Amazon Resource Name (ARN) 格式。

- arn:aws:s3:::*amzn-s3-demo-bucket*
- arn:aws:s3:::*amzn-s3-demo-bucket*/*

如需為 Amazon Aurora 建立 IAM 政策的詳細資訊，請參閱 建立並使用 IAM 政策進行 IAM 資料庫存取。另請參閱《IAM 使用者指南》中的教學：建立和連接您的第一個客戶受管原則。

下列 AWS CLI 命令會使用這些選項建立名為 ExportPolicy的 IAM 政策。其將存取權授予名為 *amzn-s3-demo-bucket* 的儲存貯體。

> **Note**
>
> 在您建立政策後，請記下政策的 ARN。在後續步驟中將政策附加至 IAM 角色時，您會需要此 ARN。

```
aws iam create-policy  --policy-name ExportPolicy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExportPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
```

```
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}'
```

2. 建立 IAM 角色，讓 Aurora 可以代表您擔任此 IAM 角色，以存取您的 Amazon S3 儲存貯體。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以將許可委派給 IAM 使用者。

   下列範例顯示使用 AWS CLI 命令來建立名為 的角色rds-s3-export-role。

```
aws iam create-role  --role-name rds-s3-export-role  --assume-role-policy-document
 '{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
           "Service": "export.rds.amazonaws.com"
         },
        "Action": "sts:AssumeRole"
      }
    ]
  }'
```

3. 將您建立的 IAM 政策附加至您建立的 IAM 角色。

   下列 AWS CLI 命令會將先前建立的政策連接至名為 的角色rds-s3-export-role。將 *your-policy-arn* 取代成您在稍早步驟中記下的政策 ARN。

```
aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
export-role
```

使用跨帳戶 Amazon S3 儲存貯體

您可以跨 AWS 帳戶使用 Amazon S3 儲存貯體。若要使用跨帳戶儲存貯體，請新增儲存貯體政策，以將存取權授予您用於 S3 匯出的 IAM 角色。如需詳細資訊，請參閱範例 2：儲存貯體擁有者授予跨帳戶儲存貯體許可。

- 將儲存貯體政策附加至您的儲存貯體，如下列範例所示。

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/Admin"
            },
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-destination-bucket",
                "arn:aws:s3:::amzn-s3-demo-destination-bucket/*"
            ]
        }
    ]
}
```

使用跨帳戶 AWS KMS key

您可以使用跨帳戶 AWS KMS key 來加密 Amazon S3 匯出。首先，將金鑰政策新增至本機帳戶，然後在外部帳戶中新增 IAM 政策。如需詳細資訊，請參閱允許其他帳戶中的使用者使用 KMS 金鑰。

使用跨帳戶 KMS 金鑰

1. 將金鑰政策新增至本機帳戶。

   下列範例將本機帳戶 123456789012 中的許可給予外部帳戶 444455556666 中的 `ExampleRole` 和 `ExampleUser`。

```
{
```

```
    "Sid": "Allow an external account to use this KMS key",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::444455556666:role/ExampleRole",
            "arn:aws:iam::444455556666:user/ExampleUser"
        ]
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
        "kms:RetireGrant"
    ],
    "Resource": "*"
}
```

2. 將 IAM 政策新增至外部帳戶。

以下 IAM 政策範例允許委託人使用帳戶 123456789012 中的 KMS 金鑰來進行密碼編譯操作。若要提供此許可給帳戶 444455556666 中的 ExampleRole 和 ExampleUser，請在該帳戶中附加政策給他們。

```
{
    "Sid": "Allow use of KMS key in account 123456789012",
    "Effect": "Allow",
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
        "kms:RetireGrant"
    ],
    "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
}
```

建立快照匯出任務

建立快照匯出任務，將資料從快照匯出至 Amazon S3 儲存貯體。您每個 AWS 帳戶最多可以有五個正在同時進行的資料庫快照匯出任務。

> **Note**
>
> 匯出 RDS 快照可能需要一段時間，依您的資料庫類型和大小而定。匯出任務會先還原和擴展整個資料庫，然後再將資料擷取到 Amazon S3。此階段期間的工作進度會顯示為STARTING (開始)。當任務切換到將資料匯出到 S3 時，進度會顯示為 In progress (進行中)。
> 匯出完成所需的時間取決於儲存在資料庫中的資料。例如，如果資料表有分散均勻的數值主索引鍵或索引資料欄，則匯出速度最快。如果資料表不含適合分割的資料欄，或是資料表只在字串型資料欄上有一個索引，則會花更多時間。因為匯出使用較慢的單一執行緒處理序，所以將會耗費更長的時間。

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 將資料庫快照匯出至 Amazon S3。

如果您使用 Lambda 函式匯出快照集，請將 `kms:DescribeKey` 動作新增至 Lambda 函式政策。如需詳細資訊，請參閱 AWS Lambda 許可。

主控台

Export to Amazon S3 (匯出至 Amazon S3) 主控台選項僅會針對可匯出至 Amazon S3 的快照顯示。由於下列原因，快照可能無法匯出：

- S3 匯出不支援此資料庫引擎。
- S3 匯出不支援此資料庫執行個體版本。
- 已建立快照的 AWS 區域不支援 S3 匯出。

匯出資料庫快照

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Snapshots (快照)。
3. 從標籤中，選擇您希望匯出的快照類型。
4. 在快照清單中，選擇您希望匯出的快照。
5. 針對 Actions (動作)，選擇 Export to Amazon S3 (匯出至 Amazon S3)。

隨即出現 Export to Amazon S3 (匯出至 Amazon S3) 視窗。

6. 針對 Export identifier (匯出識別符)，輸入名稱以識別匯出任務。這個值也會用來做為在 S3 儲存貯體中建立的檔案名稱。
7. 選擇匯出的資料：

   - 選擇 All (全部) 來匯出快照中的所有資料。
   - 選擇 Partial (部分) 來匯出快照的特定部分。若要識別要匯出的快照部分，請針對 Identifier (識別符) 輸入一或多個資料庫、結構描述或表格，並以空格分隔。

     使用下列格式：

     ```
     database[.schema][.table] database2[.schema2][.table2] ... databasen[.schemaN]
     [.tablen]
     ```

     例如：

     ```
     mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1
      mydatabase2.myschema2.mytable2
     ```
8. 針對 S3 bucket (S3 儲存貯體)，選擇要匯出的儲存貯體。

   如要將匯出資料指派給 S3 儲存貯體中的資料夾路徑，請針對 S3 prefix (S3 字首) 輸入選用的路徑。
9. 針對 IAM role (IAM 角色)，您可選擇授予對您所選擇 S3 儲存貯體寫入存取權限的角色，或建立新角色。

   - 如果您透過遵循 使用 IAM 角色提供對 Amazon S3 儲存貯體的存取權 中的步驟建立了角色，請選擇該角色。
   - 如果您沒有建立授予您所選擇 S3 儲存貯體寫入存取權限的角色，請選擇 Create a new role (建立新角色) 以自動建立角色。接下來，為 IAM 角色名稱 (IAM role name) 中的角色輸入名稱。
10. 針對 AWS KMS key，輸入金鑰的 ARN 以加密匯出的資料。
11. 選擇 Export to Amazon S3 (匯出至 Amazon S3)。

AWS CLI

若要使用 AWS CLI 將資料庫快照匯出至 Amazon S3，請搭配下列必要選項使用 start-export-task 命令：

- `--export-task-identifier`
- `--source-arn`
- `--s3-bucket-name`
- `--iam-role-arn`
- `--kms-key-id`

在下列範例中，快照匯出任務名為 *my-snapshot-export*，並且會將快照匯出至名為 *amzn-s3-demo-destination-bucket* 的 S3 儲存貯體。

Example

對於 Linux、macOS 或 Unix：

```
aws rds start-export-task \
    --export-task-identifier my-snapshot-export \
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name \
    --s3-bucket-name amzn-s3-demo-destination-bucket \
    --iam-role-arn iam-role \
    --kms-key-id my-key
```

在 Windows 中：

```
aws rds start-export-task ^
    --export-task-identifier my-snapshot-export ^
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name ^
    --s3-bucket-name amzn-s3-demo-destination-bucket ^
    --iam-role-arn iam-role ^
    --kms-key-id my-key
```

範例輸出如下。

```
{
    "Status": "STARTING",
    "IamRoleArn": "iam-role",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "amzn-s3-demo-destination-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "my-key",
    "ExportTaskIdentifier": "my-snapshot-export",
```

```
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name"
}
```

如要為快照匯出提供 S3 儲存貯體中的資料夾路徑，請在 start-export-task 命令中包含 `--s3-prefix` 選項。

RDS API

如要使用 Amazon RDS API 將資料庫快照匯出至 Amazon S3，請搭配下列必要參數使用 StartExportTask 操作：

- `ExportTaskIdentifier`
- `SourceArn`
- `S3BucketName`
- `IamRoleArn`
- `KmsKeyId`

監控快照匯出

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 監控資料庫快照匯出。

主控台

監控資料庫快照匯出

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Exports in Amazon S3 (在 Amazon S3 中匯出)。

   資料庫快照匯出會在 Source type (來源類型) 資料欄中指出。匯出狀態會顯示在 Status (狀態) 資料欄中。
3. 若要檢視特定快照匯出的詳細資訊，請選擇匯出任務。

AWS CLI

如要使用 AWS CLI 監控資料庫快照匯出，請使用 describe-export-tasks 命令。

以下範例示範如何顯示您所有快照匯出的目前資訊。

Example

```
aws rds describe-export-tasks

{
    "ExportTasks": [
        {
            "Status": "CANCELED",
            "TaskEndTime": "2019-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "ExportTime": "2019-10-24T20:23:48.364Z",
            "S3Bucket": "amzn-s3-demo-bucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/
bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-25T19:10:58.885Z",
            "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:parameter-
groups-test"
        },
{
            "Status": "COMPLETE",
            "TaskEndTime": "2019-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general
\":[{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
            "ExportTime": "2019-10-31T06:44:53.452Z",
            "S3Bucket": "amzn-s3-demo-bucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "TaskStartTime": "2019-10-31T20:58:06.998Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-31-06-44"
        },
        {
            "Status": "FAILED",
```

```
            "TaskEndTime": "2019-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket my-exports isn't located in the current AWS
 Region. Please, review your S3 bucket name and retry the export.",
            "S3Prefix": "",
            "ExportTime": "2019-10-30T06:45:04.526Z",
            "S3Bucket": "amzn-s3-demo-bucket2",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-30T22:43:40.034Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-30-06-45"
        }
    ]
}
```

如要顯示特定快照匯出的相關資訊，請在 `--export-task-identifier` 命令中包含 `describe-export-tasks` 選項。如要篩選輸出，請包含 `--Filters` 選項。如需更多選項，請參閱 describe-export-tasks 命令。

RDS API

如要使用 Amazon RDS API 顯示資料庫快照匯出的相關資訊，請使用 DescribeExportTasks 操作。

若要追蹤匯出工作流程的完成或起始其他工作流程，您可以訂閱 Amazon Simple Notification Service 主題。如需 Amazon SNS 的詳細資訊，請參閱使用 Amazon RDS 事件通知。

取消快照匯出任務

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 取消資料庫快照匯出任務。

> **Note**
>
> 取消快照匯出任務不會移除任何已匯出到 Amazon S3 的資料。如需如何使用主控台刪除資料的資訊，請參閱如何從 S3 儲存貯體刪除物件？ 如要使用 CLI 刪除資料，請使用 delete-object 命令。

主控台

取消快照匯出任務

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Exports in Amazon S3 (在 Amazon S3 中匯出)。

   資料庫快照匯出會在 Source type (來源類型) 資料欄中指出。匯出狀態會顯示在 Status (狀態) 資料欄中。
3. 選擇您希望取消的快照匯出任務。
4. 選擇 Cancel (取消)。
5. 選擇確認頁面上的 Cancel export task (取消匯出任務)。

AWS CLI

如要使用 AWS CLI 取消快照匯出任務，請使用 cancel-export-task 命令。命令需要 `--export-task-identifier` 選項。

Example

```
aws rds cancel-export-task --export-task-identifier my_export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "amzn-s3-demo-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my_export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:export-example-1"
}
```

RDS API

如要使用 Amazon RDS API 取消快照匯出任務，請搭配 `ExportTaskIdentifier` 參數使用 CancelExportTask 操作。

Aurora MySQL 中的匯出效能

Aurora MySQL 第 2 版和第 3 版資料庫叢集快照會使用進階匯出機制，來改善效能並縮短匯出時間。此機制包含多個匯出執行緒和 Aurora MySQL 平行查詢等最佳化功能，以利用 Aurora 共用儲存體架構。最佳化會以自適應方式套用，取決於資料集大小和結構。

您不需要開啟不行查詢，即可使用更快的匯出程序，但此程序確實具有與平行查詢相同的限制。此外，不支援某些資料值，例如幾月幾日為 `0` 或年份為 `0000` 的日期。如需更多詳細資訊，請參閱 Amazon Aurora MySQL 的平行查詢。

套用效能最佳化時，您也可能看到大很多 (約 200 GB) 的 Parquet 檔案，進行 Aurora MySQL 第 2 版和第 3 版匯出。

如果無法使用更快的匯出程序 (例如因為資料類型或值不相容)，Aurora 會自動切換至單一執行緒匯出模式，而沒有平行查詢。根據使用的程序以及要匯出的資料量，匯出效能可能會有所不同。

故障診斷快照匯出

使用下列各節來協助針對資料庫叢集匯出任務到 Amazon S3 的失敗訊息和 PostgreSQL 許可錯誤進行故障診斷。

Amazon S3 匯出任務的失敗訊息

下表說明 Amazon S3 匯出任務失敗時傳回的訊息。

失敗訊息	Description
發生未知的內部錯誤。	因為不明的錯誤、例外或故障，所以任務失敗。
將匯出任務的中繼資料寫入 S3 儲存貯體 [儲存貯體名稱] 時發生未知的內部錯誤。	因為不明的錯誤、例外或故障，所以任務失敗。
RDS 匯出無法寫入匯出任務的中繼資料，因為無法擔任 IAM 角色 [角色 ARN]。	匯出任務會擔任您的 IAM 角色，以驗證是否允許將中繼資料寫入 S3 儲存貯體。如果任務無法擔任您的 IAM 角色，則會失敗。
RDS 匯出無法使用具有 KMS 金鑰 [金鑰 ID] 的 IAM 角色 [角色 ARN]，將匯出任務的中繼資料寫入 S3 儲存貯體 [儲存貯體名稱]。錯誤代碼：[錯誤代碼]	缺少一或多個許可，因此匯出任務無法存取 S3 儲存貯體。收到下列其中一個錯誤碼時會引發此失敗訊息： • `AWSSecurityTokenServiceException` 和錯誤代碼 `AccessDenied`

失敗訊息	Description
	• AmazonS3Exception 和錯誤代碼 NoSuchBucket 、AccessDenied 、KMS.KMSInvalidStateException 、403 Forbidden 或 KMS.DisabledException 這些錯誤碼表示 IAM 角色、S3 儲存貯體或 KMS 金鑰的設定錯誤。
IAM 角色 [角色 ARN] 未獲得授權在 S3 儲存貯體 [儲存貯體名稱] 上呼叫 [S3 動作]。檢閱您的許可，然後重試匯出。	IAM 政策設定錯誤。缺少 S3 儲存貯體上特定 S3 動作的許可，這會導致匯出任務失敗。
KMS 金鑰檢查失敗。請檢查 KMS 金鑰上的憑證，然後再試一次。	KMS 金鑰憑證檢查失敗。
S3 憑證檢查失敗。檢查您的 S3 儲存貯體與 IAM 政策的許可。	S3 憑證檢查失敗。
S3 儲存貯體 [儲存貯體名稱] 無效。可能不是位於目前的 AWS 區域 或不存在。檢閱您的 S3 儲存貯體名稱並重試匯出。	S3 儲存貯體無效。
S3 儲存貯體 [儲存貯體名稱] 並非位於目前的 AWS 區域。檢閱您的 S3 儲存貯體名稱並重試匯出。	S3 儲存貯體錯誤 AWS 區域。

對 PostgreSQL 許可錯誤進行故障診斷

將 PostgreSQL 資料庫匯出至 Amazon S3 時，您可能會看到 PERMISSIONS_DO_NOT_EXIST 錯誤，指出某些資料表已略過。您在建立資料庫執行個體時指定的超級使用者，若沒有存取這些資料表的許可，通常就會發生此錯誤。

若要修正此錯誤，請執行下列命令：

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

如需超級使用者權限的詳細資訊，請參閱主要使用者帳戶權限。

將資料庫叢集還原至指定時間

您可以將資料庫叢集還原到特定的時間點，建立新的資料庫叢集。

將資料庫叢集還原至某個時間點時，您可以選擇預設的 Virtual Private Cloud (VPC) 安全群組。或者，可以將自訂 VPC 安全群組套用至您的資料庫叢集。

還原的資料庫叢集會自動與預設資料庫叢集和資料庫參數群組產生關聯。不過，您可以在還原期間指定要套用的自訂參數群組。

Amazon Aurora 會持續將資料庫叢集的日誌記錄上傳至 Amazon S3。若要查看資料庫叢集的最新可還原時間，請使用 AWS CLI describe-db-clusters 命令，並查看資料庫叢集`LatestRestorableTime`欄位中傳回的值。

您可以還原至備份保留期間內的任何時間點。若要查看資料庫叢集最早的可還原時間，請使用 AWS CLI describe-db-clusters 命令，並查看資料庫叢集`EarliestRestorableTime`欄位中傳回的值。

還原的資料庫叢集的備份保留期間與來源資料庫叢集相同。

> **Note**
>
> 本主題中的資訊適用於 Amazon Aurora。如需還原 Amazon RDS 資料庫執行個體的相關資訊，請參閱將資料庫執行個體還原至指定的時間。
> 如需備份並還原 Aurora 資料庫叢集的詳細資訊，請參閱 備份與還原 Aurora 資料庫叢集的概觀。
> 以 Aurora MySQL 而言，您可將佈建的資料庫叢集還原為 Aurora Serverless 資料庫叢集。如需詳細資訊，請參閱還原 Aurora Serverless v1 資料庫叢集。
> 您也可以使用 AWS Backup 來管理 Amazon Aurora 資料庫叢集的備份。如果您的資料庫叢集與 AWS Backup中的備份計劃相關聯，該備份計劃會用於時間點復原。如需相關資訊，請參閱使用 AWS Backup 將資料庫叢集還原至指定的時間。

如需使用 RDS 延長支援版本還原 Aurora 資料庫叢集或全域叢集的相關資訊，請參閱 使用 Amazon RDS 延伸支援還原Aurora 資料庫叢集或全域叢集。

從自動備份、保留的自動備份或使用 將資料庫叢集還原至指定的時間 AWS Backup。

主題

- 將資料庫叢集還原至時間點

- 從保留的自動備份將資料庫叢集還原到指定的時間
- 使用 AWS Backup 將資料庫叢集還原至指定的時間

將資料庫叢集還原至時間點

您可以使用 AWS CLI、 或 RDS API AWS 管理主控台將資料庫叢集還原至某個時間點。

主控台

將資料庫叢集還原至指定時間

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Automated backups (自動備份)。

   自動備份顯示在 Current Region (目前區域) 索引標籤上。

3. 選擇您要還原的 資料庫叢集。
4. 針對 Actions (動作)，選擇 Restore to point in time (還原至時間點)。

   Restore to point in time (還原到時間點) 視窗隨即出現。
5. 選擇 Latest restorable time (最近的可還原時間) 以還原最近一次的可能時間，或選擇 Custom (自訂) 以選擇一個時間。

   如果您選擇 Custom (自訂)，請輸入您希望資料庫叢集還原到什麼日期和時間。

   > **Note**
   >
   > 會以您的當地時區顯示時間，根據國際標準時間 (UTC) 的時差來表示。例如，UTC-5 是東部標準時間/中部日光節約時間。
6. 對於資料庫叢集識別符，輸入目標還原資料庫叢集的名稱。名稱必須是唯一的。
7. 視需要選擇其他選項，例如資料庫執行個體類別和資料庫叢集儲存組態。

如需每項設定的相關資訊，請參閱 Aurora 資料庫叢集的設定。

8. 選擇 Restore to point in time (還原至時間點)。

AWS CLI

若要將資料庫叢集還原至指定的時間，請使用 restore-db-cluster-to-point-in-time AWS CLI 命令來建立新的資料庫叢集。

您可以指定其他設定。如需每項設定的相關資訊，請參閱 Aurora 資料庫叢集的設定。

如果在請求中提供標籤，則提供的標籤會套用至還原的資料庫叢集。如果請求中未提供標籤，且來源資料庫叢集處於區域作用中且有標籤，Aurora 會將來源資料庫叢集的最新標籤新增至還原的資料庫叢集。

Example

針對 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-to-point-in-time \
       --source-db-cluster-identifier mysourcedbcluster \
       --db-cluster-identifier mytargetdbcluster \
       --restore-to-time 2017-10-14T23:45:00.000Z
```

在 Windows 中：

```
aws rds restore-db-cluster-to-point-in-time ^
       --source-db-cluster-identifier mysourcedbcluster ^
       --db-cluster-identifier mytargetdbcluster ^
       --restore-to-time 2017-10-14T23:45:00.000Z
```

Important

如果您使用主控台來將資料庫叢集還原至指定時間，則 Amazon RDS 會自動建立資料庫叢集的主要執行個體 (寫入器)。如果您使用 AWS CLI 將資料庫叢集還原至指定的時間，則必須明確建立資料庫叢集的主要執行個體。主要執行個體是資料庫叢集內第一個建立的執行個體。呼叫 create-db-instance AWS CLI 命令，為資料庫叢集建立主要執行個體。包含資料庫叢集的名稱做為 `--db-cluster-identifier` 選項值。

RDS API

若要將資料庫叢集還原至指定的時間，請搭配下列參數呼叫 Amazon RDS API [RestoreDBClusterToPointInTime](#) 操作：

- `SourceDBClusterIdentifier`
- `DBClusterIdentifier`
- `RestoreToTime`

> **Important**
>
> 如果您使用主控台來將資料庫叢集還原至指定時間，則 Amazon RDS 會自動建立資料庫叢集的主要執行個體 (寫入器)。如果您使用 RDS API 來將資料庫叢集還原至指定時間，請務必明確建立資料庫叢集的主要執行個體。主要執行個體是資料庫叢集內第一個建立的執行個體。若要建立資料庫叢集的主要執行個體，請呼叫 RDS API 操作 [CreateDBInstance](#)。包含資料庫叢集的名稱做為 `DBClusterIdentifier` 參數值。

從保留的自動備份將資料庫叢集還原到指定的時間

如果備份在來源叢集的保留期間內，您可以在刪除來源資料庫叢集之後，從保留的自動備份還原資料庫叢集。此程序類似於從自動備份還原資料庫叢集。

> **Note**
>
> 您無法使用此程序還原 Aurora Serverless v1 資料庫叢集，因為 Aurora Serverless v1 叢集的自動備份不會保留。

主控台

將資料庫叢集還原至指定時間

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Automated backups (自動備份)。
3. 選擇保留標籤。

4. 選擇您要還原的 資料庫叢集。
5. 針對 Actions (動作)，選擇 Restore to point in time (還原至時間點)。

   Restore to point in time (還原到時間點) 視窗隨即出現。
6. 選擇 Latest restorable time (最近的可還原時間) 以還原最近一次的可能時間，或選擇 Custom (自訂) 以選擇一個時間。

   如果您選擇 Custom (自訂)，請輸入您希望資料庫叢集還原到什麼日期和時間。

   > **Note**
   >
   > 會以您的當地時區顯示時間，根據國際標準時間 (UTC) 的時差來表示。例如，UTC-5 是東部標準時間/中部日光節約時間。
7. 對於資料庫叢集識別符，輸入目標還原資料庫叢集的名稱。名稱必須是唯一的。
8. 視需要選擇其他選項，例如資料庫執行個體類別。

   如需每項設定的相關資訊，請參閱 Aurora 資料庫叢集的設定。
9. 選擇 Restore to point in time (還原至時間點)。

AWS CLI

若要將資料庫叢集還原至指定時間，請使用 AWS CLI 命令 restore-db-cluster-to-point-in-time 來建立新的資料庫叢集。

您可以指定其他設定。如需每項設定的相關資訊，請參閱 Aurora 資料庫叢集的設定。

此操作支援資源標記。使用 `--tags` 選項時，會忽略來源資料庫叢集標籤，並使用提供的標籤。否則，會使用來源叢集中的最新標籤。

Example

對於 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-resource-id cluster-123ABCEXAMPLE \
    --db-cluster-identifier mytargetdbcluster \
    --restore-to-time 2017-10-14T23:45:00.000Z
```

在 Windows 中：

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-resource-id cluster-123ABCEXAMPLE ^
    --db-cluster-identifier mytargetdbcluster ^
    --restore-to-time 2017-10-14T23:45:00.000Z
```

Important

如果您使用主控台來將資料庫叢集還原至指定時間，則 Amazon RDS 會自動建立資料庫叢集的主要執行個體 (寫入器)。如果您使用 AWS CLI 來將資料庫叢集還原至指定時間，則必須明確地建立資料庫叢集的主要執行個體。主要執行個體是資料庫叢集內第一個建立的執行個體。呼叫 create-db-instance AWS CLI 命令，為資料庫叢集建立主要執行個體。包含資料庫叢集的名稱做為 `--db-cluster-identifier` 選項值。

RDS API

若要將資料庫叢集還原至指定的時間，請搭配下列參數呼叫 Amazon RDS API `RestoreDBClusterToPointInTime` 操作：

- `SourceDbClusterResourceId`
- `DBClusterIdentifier`
- `RestoreToTime`

Important

如果您使用主控台來將資料庫叢集還原至指定時間，則 Amazon RDS 會自動建立資料庫叢集的主要執行個體 (寫入器)。如果您使用 RDS API 來將資料庫叢集還原至指定時間，請務必明確建立資料庫叢集的主要執行個體。主要執行個體是資料庫叢集內第一個建立的執行個體。若要建立資料庫叢集的主要執行個體，請呼叫 RDS API 操作 CreateDBInstance。包含資料庫叢集的名稱做為 `DBClusterIdentifier` 參數值。

使用 AWS Backup 將資料庫叢集還原至指定的時間

您可以使用 AWS Backup 管理您的自動備份，然後將其還原到指定的時間。若要這樣做，請在 AWS Backup 中建立備份計劃，並將您的資料庫叢集指派為資源。然後在備份規則中為 PITR 啟用連續備份。如需備份計劃和備份規則的詳細資訊，請參閱《AWS Backup 開發人員指南》。

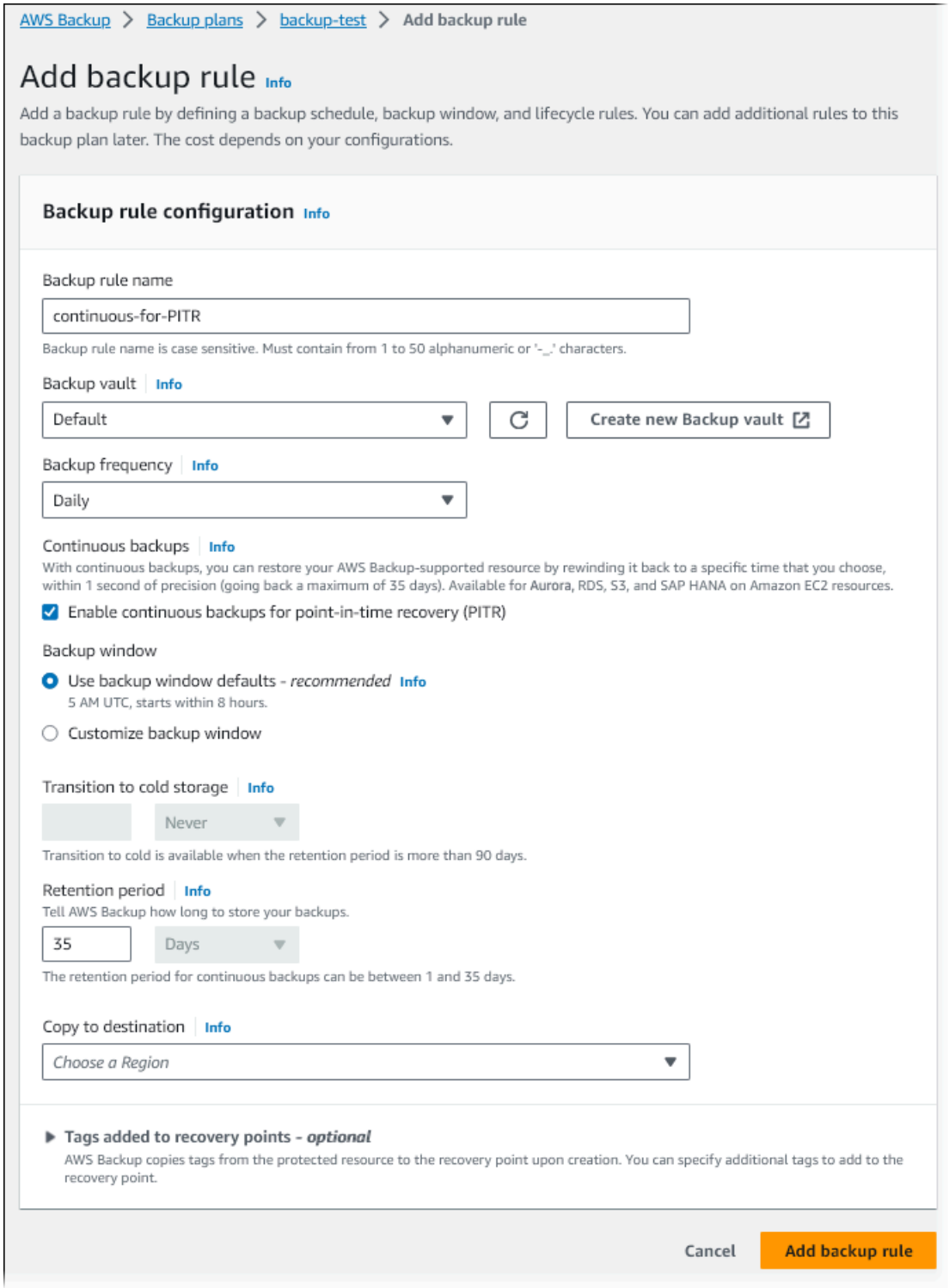
在 AWS Backup 中啟用連續備份

您會在備份規則中啟用連續備份。

若要為 PITR 啟用連續備份

1. 請登入 AWS 管理主控台，並開啟位於 https://console.aws.amazon.com/backup 的 AWS Backup 主控台。
2. 在導覽窗格中，選擇 Backup plans (備份計劃)。
3. 在備份計劃名稱下，選取用來備份資料庫叢集的備份計劃。
4. 在備份規則區段下，選擇新增備份規則。

   新增備份規則頁面便會顯示。
5. 勾選為時間點復原 (PITR) 啟用連續備份核取方塊。



6. 視需要選擇其他設定，然後選擇新增備份規則。

在 AWS Backup 中從連續備份還原

您會從備份文件庫還原至指定的時間。

主控台

您可以使用 AWS 管理主控台 將資料庫叢集還原至指定的時間。

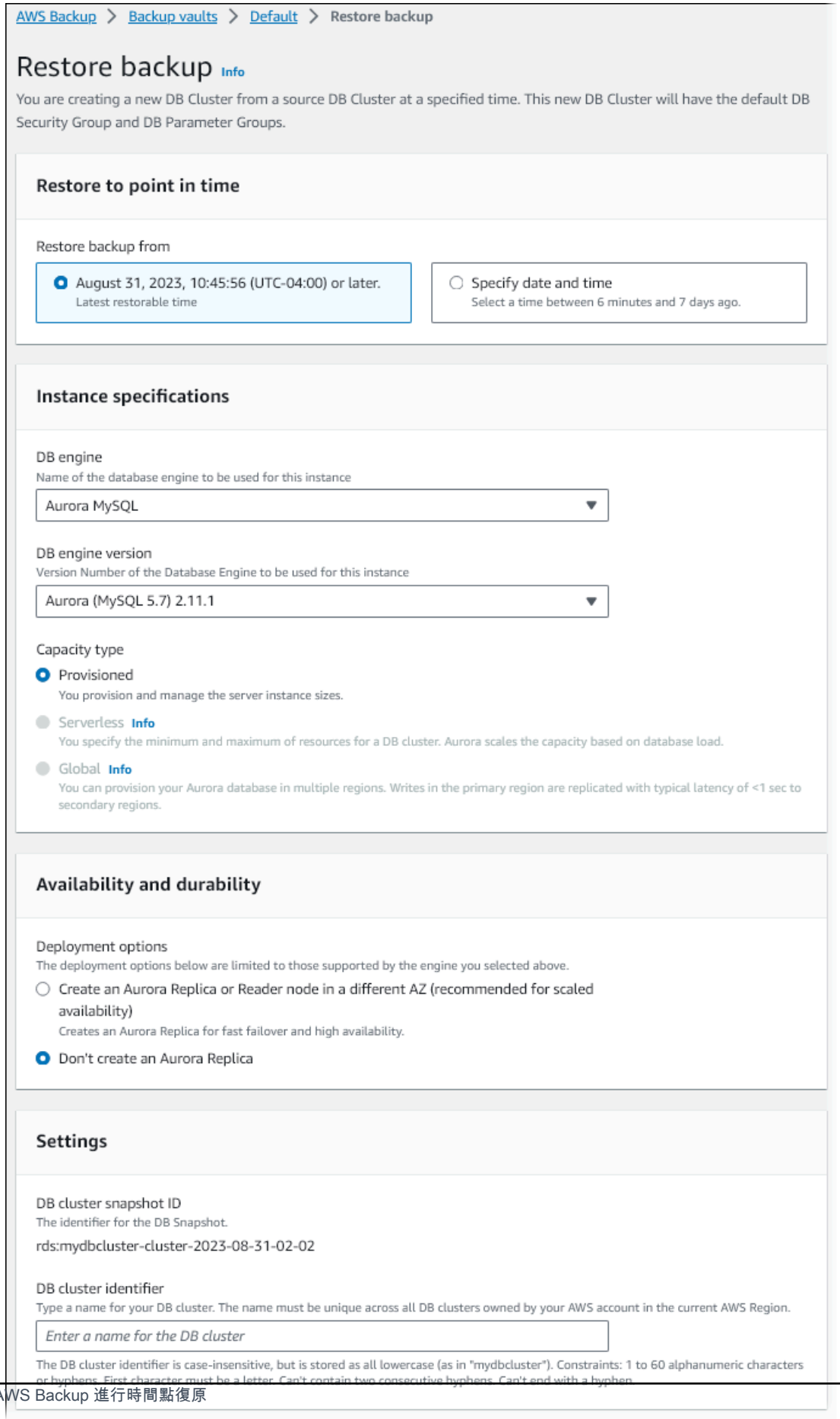
若要在 AWS Backup 中從連續備份還原

1. 請登入 AWS 管理主控台，並開啟位於 https://console.aws.amazon.com/backup 的 AWS Backup 主控台。
2. 在導覽窗格中，選擇 Backup vaults (備份文件庫)。
3. 選擇包含連續備份的備份文件庫，例如預設。

   備份文件庫詳細資訊頁面隨即顯示。
4. 在復原點下，選取自動備份的復原點。

   它的備份類型為連續，且名字包含 `continuous:cluster-`*`AWS-Backup-job-number`*。
5. 針對動作，選擇還原。

   還原備份頁面隨即顯示。


AWS Backup > Backup vaults > Default > Restore backup
Restore backup Info
You are creating a new DB Cluster from a source DB Cluster at a specified time. This new DB Cluster will have the default DB Security Group and DB Parameter Groups.
Restore to point in time
Restore backup from
August 31, 2023, 10:45:56 (UTC-04:00) or later.
Latest restorable time
Specify date and time
Select a time between 6 minutes and 7 days ago.
Instance specifications
DB engine
Name of the database engine to be used for this instance
Aurora MySQL
DB engine version
Version Number of the Database Engine to be used for this instance
Aurora (MySQL 5.7) 2.11.1
Capacity type
Provisioned
You provision and manage the server instance sizes.
Serverless Info
You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load.
Global Info
You can provision your Aurora database in multiple regions. Writes in the primary region are replicated with typical latency of <1 sec to secondary regions.
Availability and durability
Deployment options
The deployment options below are limited to those supported by the engine you selected above.
Create an Aurora Replica or Reader node in a different AZ (recommended for scaled availability)
Creates an Aurora Replica for fast failover and high availability.
Don't create an Aurora Replica
Settings
DB cluster snapshot ID
The identifier for the DB Snapshot.
rds:mydbcluster-cluster-2023-08-31-02-02
DB cluster identifier
Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.
Enter a name for the DB cluster
The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

6. 對於還原至時間點，選取指定日期和時間以還原至特定時間點。
7. 視需要選擇用於還原資料庫叢集的其他設定，然後選擇還原備份。

   任務頁面隨即出現，並顯示還原任務窗格。頁面頂端的訊息提供還原任務的相關資訊。

資料庫叢集還原後，您必須將主要 (寫入器) 資料庫執行個體新增至其中。呼叫 create-db-instance AWS CLI 命令，為資料庫叢集建立主要執行個體。包含資料庫叢集的名稱做為 `--db-cluster-identifier` 參數值。

CLI

請使用 start-restore-job AWS CLI 命令將資料庫叢集還原至指定的時間。下列是必要參數：

- `--recovery-point-arn` - 要從該處還原之復原點的 Amazon Resource Name (ARN)。
- `--resource-type` - 使用 Aurora。
- `--iam-role-arn` - 用於 AWS Backup 操作之 IAM 角色的 ARN。
- `--metadata` - 用來還原資料庫叢集的中繼資料。下列是必要參數：
  - `DBClusterIdentifier`
  - `Engine`
  - `RestoreToTime` 或 `UseLatestRestorableTime`

以下範例說明如何將資料庫叢集還原至指定的時間。

```
aws backup start-restore-job \
--recovery-point-arn arn:aws:backup:eu-central-1:123456789012:recovery-
point:continuous:cluster-itsreallyjustanexample1234567890-487278c2 \
--resource-type Aurora \
--iam-role-arn arn:aws:iam::123456789012:role/service-role/AWSBackupDefaultServiceRole
 \
--metadata '{"DBClusterIdentifier":"backup-pitr-test","Engine":"aurora-
mysql","RestoreToTime":"2023-09-01T17:00:00.000Z"}'
```

以下範例說明如何將資料庫叢集還原至最新的可還原時間。

```
aws backup start-restore-job \
--recovery-point-arn arn:aws:backup:eu-central-1:123456789012:recovery-
point:continuous:cluster-itsreallyjustanexample1234567890-487278c2 \
--resource-type Aurora \
```

```
--iam-role-arn arn:aws:iam::123456789012:role/service-role/AWSBackupDefaultServiceRole
 \
--metadata '{"DBClusterIdentifier":"backup-pitr-latest","Engine":"aurora-
mysql","UseLatestRestorableTime":"true"}'
```

資料庫叢集還原後，您必須將主要 (寫入器) 資料庫執行個體新增至其中。呼叫 create-db-instance AWS CLI 命令，為資料庫叢集建立主要執行個體。包含資料庫叢集的名稱做為 `--db-cluster-identifier` 參數值。

刪除資料庫叢集快照

當您不再需要時，可以透過 Amazon RDS 刪除資料庫 叢集 快照。

> **Note**
>
> 使用 AWS Backup 主控台，刪除由 AWS Backup 管理的備份。如需 AWS Backup 的詳細資訊，請參閱《AWS Backup 開發人員指南》。

刪除資料庫叢集快照

您可以使用主控台、AWS CLI 或 RDS API 刪除資料庫叢集快照。

要刪除共用或公開的快照，您必須登入擁有該快照的 AWS 帳戶。

主控台

刪除資料庫叢集快照

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Snapshots (快照)。
3. 選擇想要刪除的資料庫叢集快照。
4. 針對 Actions (動作)，選擇 Delete Snapshot (刪除快照)。
5. 在確認頁面上，選擇 Delete (刪除)。

AWS CLI

您可以使用 AWS CLI 指令 刪除資料庫叢集快照 來刪除資料庫叢集快照。

下列選項可用來刪除資料庫叢集快照。

- `--db-cluster-snapshot-identifier` – 資料庫叢集快照的識別符。

Example

以下程式碼會刪除 `mydbclustersnapshot` 資料庫叢集快照。

對於 Linux、macOS 或 Unix：

```
aws rds delete-db-cluster-snapshot \
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

在 Windows 中：

```
aws rds delete-db-cluster-snapshot ^
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

RDS API

您可以使用 Amazon RDS API 操作 DeleteDBClusterSnapshot (刪除資料庫叢集快照) 來刪除資料庫叢集快照。

下列參數用來刪除資料庫叢集快照。

- `DBClusterSnapshotIdentifier` – 資料庫叢集快照的識別符。

教學：從資料庫叢集快照還原 Amazon Aurora 資料庫叢集

使用 Amazon Aurora 的一個常見情況是，您有一個偶爾使用的資料庫執行個體，但該執行個體不需全時段可用。例如，您可能使用資料庫叢集來保留每個季度執行的報表資料。在此情況下，節省支出的方式就是在報告完成後建立資料庫叢集的資料庫叢集快照。然後，您便可以刪除資料庫叢集，並在下個季度需要上傳新資料並執行報表時將其還原。

還原資料庫叢集時，您需要提供要還原之資料庫叢集快照的名稱。然後，您需提供從還原作業中建立的新資料庫叢集名稱。如需有關從快照還原資料庫叢集的詳細資訊，請參閱 從資料庫叢集快照還原。

在本教學課程中，我們也將 Aurora MySQL 版本 2 (與 MySQL 5.7 相容) 資料庫叢集升級還原至 Aurora MySQL 版本 3 (與 MySQL 8.0 相容) 的資料庫叢集。

使用 Amazon RDS 主控台或 AWS CLI，從資料庫叢集快照將資料庫叢集還原至指定的時間。

如需有關 Amazon RDS AWS KMS 金鑰管理的詳細資訊，請參閱 AWS KMS key 管理。

主題

- 教學課程：使用 Amazon RDS 主控台，從資料庫叢集快照還原資料庫叢集
- 教學課程：使用 AWS CLI 從資料庫叢集快照還原資料庫叢集

教學課程：使用 Amazon RDS 主控台，從資料庫叢集快照還原資料庫叢集

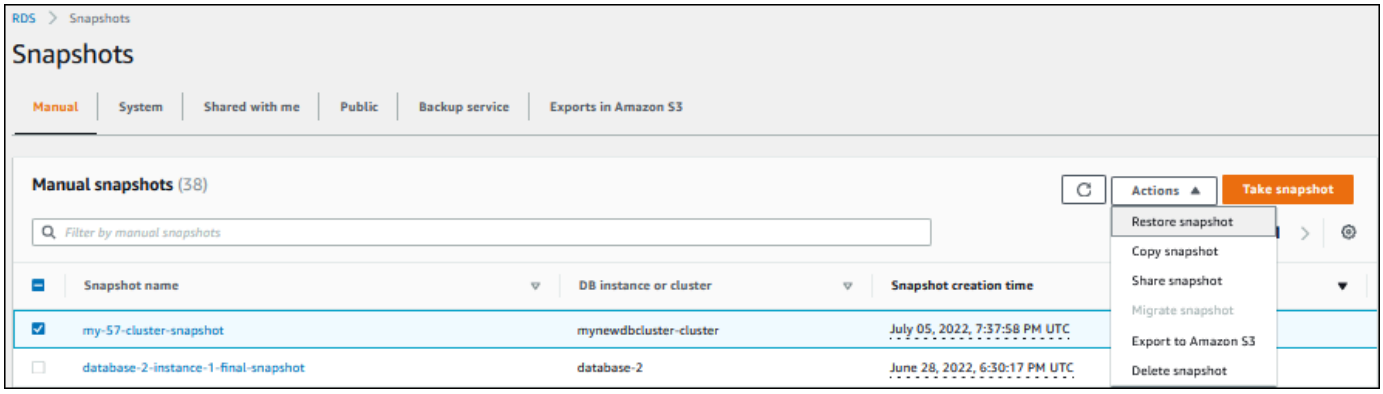
在本教學課程中，您會使用 Amazon RDS 主控台，從資料庫叢集快照還原資料庫叢集。使用 AWS 管理主控台 從快照還原資料庫叢集時，也會建立主要 (寫入器) 資料庫執行個體。

> **Note**
>
> 建立主要資料庫執行個體時，會顯示為讀取器執行個體，但是，完成建立之後便是寫入器執行個體。

從資料庫叢集快照還原資料庫叢集

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Snapshots (快照)。

3. 選擇想要從中還原的資料庫叢集快照。
4. 針對 Actions (動作)，選擇 Restore snapshot (還原快照)。

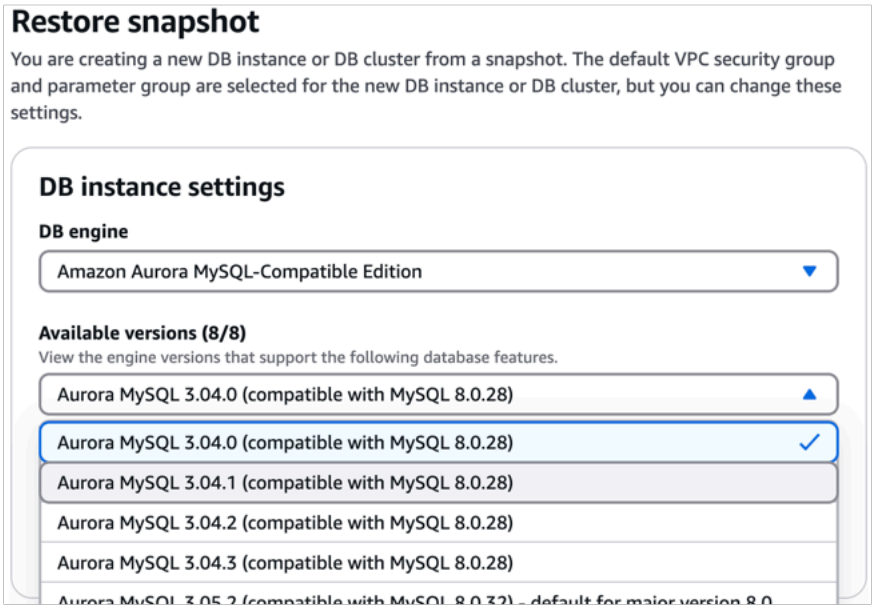


Restore snapshot (還原快照) 頁面隨即出現。

5. 在 DB instance Settings (資料庫執行個體設定) 中執行下列動作：

   a. 對於 DB engine (資料庫引擎)，請使用預設設定。

   b. 對於可用版本，選擇與 MySQL 8.0 相容的版本，例如 Aurora MySQL 3.04.0 (與 MySQL 8.0.28 相容)。



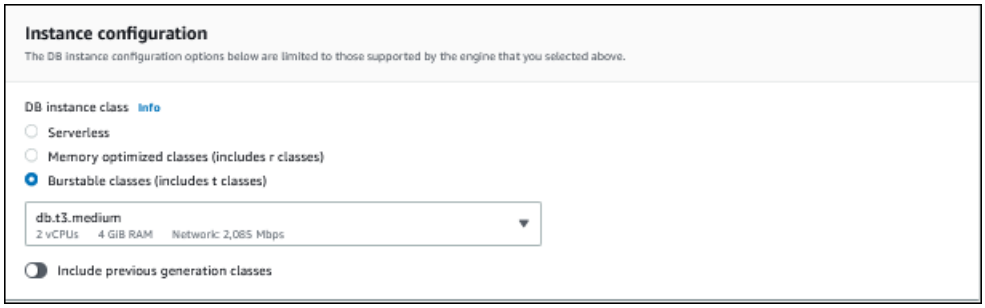
6. 在 Settings (設定) 下的 DB instance identifier (資料庫執行個體識別符) 中，輸入還原的資料庫執行個體要使用的唯一名稱，如 **`my-80`**。

> **Note**
>
> 若要建立資料庫叢集識別符，Amazon RDS 會將 `-cluster` 附加至您指定的資料庫執行個體識別符。

7. 在 Connectivity (連線) 底下，使用以下項目的預設設定：
   - Virtual Private Cloud (VPC) (虛擬私有雲端 (VPC))
   - DB subnet group (資料庫子網路群組)
   - 公用存取
   - VPC security group (firewall) (VPC 安全群組 (防火牆))
8. 選擇 DB instance class (資料庫執行個體類別)。

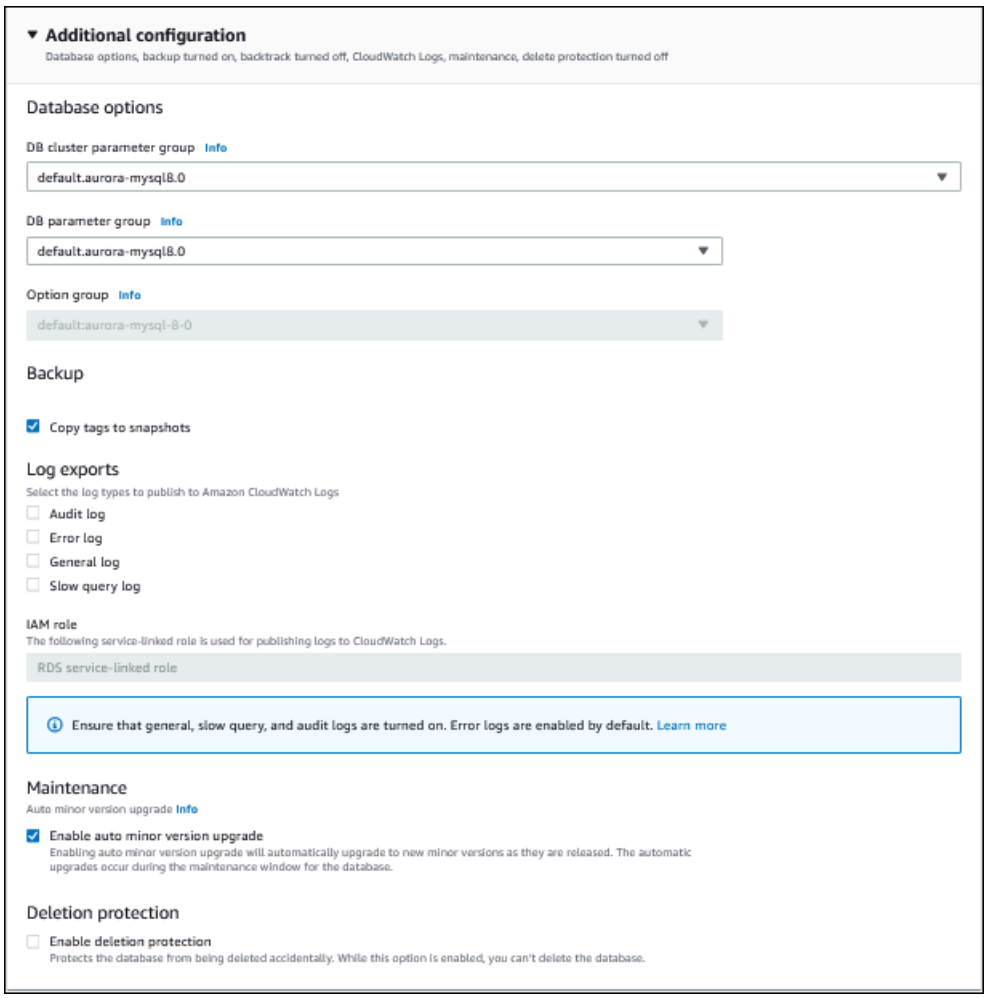
   在本教學課程中，請選擇 Burstable classes (includes t classes) (高載類別 (包括 t 類別))，然後選擇 db.t3.medium。

> **Note**
>
> 建議您在開發、測試伺服器或其他非生產伺服器時，僅使用 T 資料庫執行個體類別。如需詳細了解 T 執行個體類別，請參閱 資料庫執行個體類別的類型。



9. 對於 Database authentication (資料庫身分驗證)，請使用預設設定。
10. Encryption (加密) 請使用預設設定。

    如果快照的來源資料庫叢集已加密，還原的資料庫叢集也會加密。無法使還原項目處於未加密狀態。
11. 展開頁面底部的 Additional configuration (其他組態)。



12. 進行下列選擇：

    a. 在本教學課程中，對於 DB cluster parameter group (資料庫叢集參數群組)，請使用預設值。

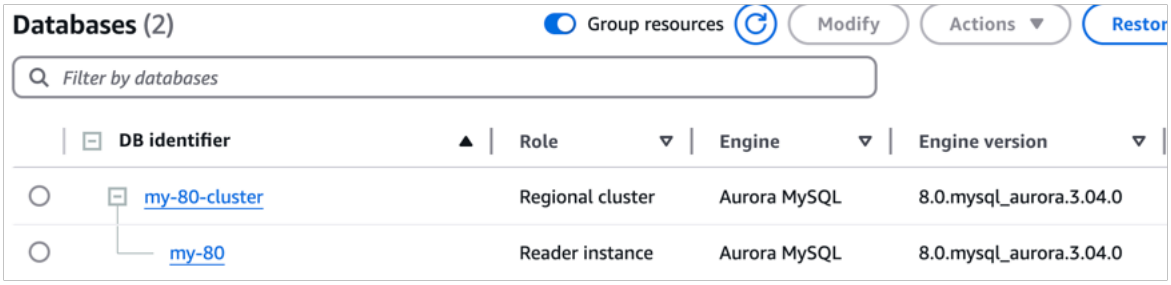
    b. 在本教學課程中，對於 DB parameter group (資料庫參數群組)，請使用預設值。

    c. 對於 Log exports (日誌匯出)，請選取所有核取方塊。

    d. 在 Deletion protection (刪除保護) 選取 Enable deletion protection (啟用刪除保護) 核取方塊。

13. 選擇 Restore DB instance (還原資料庫執行個體)。

Databases (資料庫) 頁面會顯示還原的資料庫叢集，其狀態為 `Creating`。



建立主要資料庫執行個體時，會顯示為讀取器執行個體，但是，完成建立之後便是寫入器執行個體。

教學課程：使用 AWS CLI 從資料庫叢集快照還原資料庫叢集

在本教學課程中，您會使用 AWS CLI 從資料庫叢集快照還原資料庫叢集。使用 AWS CLI 從快照還原資料庫叢集有兩個步驟：

1. 使用 restore-db-cluster-from-snapshot 命令的 還原資料庫叢集
2. 使用 create-db-instance 命令的 建立主要 (寫入器) 資料庫執行個體

還原資料庫叢集

您可以使用 `restore-db-cluster-from-snapshot` 命令。下列選項是必要的：

- `--db-cluster-identifier` – 要還原的資料庫叢集名稱。
- `--snapshot-identifier` – 用來還原的資料庫快照名稱。
- `--engine` – 還原資料庫叢集的資料庫引擎。必須與來源資料庫叢集的資料庫叢集的資料庫引擎相容。

  下列選項是選用的：

  - `aurora-mysql` – 與 Aurora MySQL 5.7 及 8.0 相容。
  - `aurora-postgresql` – 與 Aurora PostgreSQL 相容。

  在此範例中，我們使用 `aurora-mysql`.
- `--engine-version` – 還原資料庫叢集的版本。在此範例中，我們使用與 MySQL 8.0 相容的版本。

以下範例會從名為 `my-57-cluster-snapshot` 的資料庫叢集快照，還原名為 `my-new-80-cluster` 與 Aurora MySQL 8.0 相容的資料庫叢集。

還原資料庫叢集

- 請使用下列其中一個命令。

  對於 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier my-new-80-cluster \
    --snapshot-identifier my-57-cluster-snapshot \
    --engine aurora-mysql \
    --engine-version 8.0.mysql_aurora.3.02.0
```

  在 Windows 中：

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier my-new-80-cluster ^
    --snapshot-identifier my-57-cluster-snapshot ^
    --engine aurora-mysql ^
    --engine-version 8.0.mysql_aurora.3.02.0
```

輸出結果與以下內容相似。

```
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "eu-central-1b",
            "eu-central-1c",
            "eu-central-1a"
        ],
        "BackupRetentionPeriod": 14,
        "DatabaseName": "",
        "DBClusterIdentifier": "my-new-80-cluster",
        "DBClusterParameterGroup": "default.aurora-mysql8.0",
        "DBSubnetGroup": "default",
        "Status": "creating",
        "Endpoint": "my-new-80-cluster.cluster-############.eu-
central-1.rds.amazonaws.com",
        "ReaderEndpoint": "my-new-80-cluster.cluster-ro-############.eu-
central-1.rds.amazonaws.com",
        "MultiAZ": false,
```

```
        "Engine": "aurora-mysql",
        "EngineVersion": "8.0.mysql_aurora.3.02.0",
        "Port": 3306,
        "MasterUsername": "admin",
        "PreferredBackupWindow": "01:55-02:25",
        "PreferredMaintenanceWindow": "thu:21:14-thu:21:44",
        "ReadReplicaIdentifiers": [],
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-########",
                "Status": "active"
            }
        ],
        "HostedZoneId": "Z1RLNU0EXAMPLE",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:eu-central-1:123456789012:key/#######-5ccc-49cc-8aaa-
############",
        "DbClusterResourceId": "cluster-ZZ12345678ITSJUSTANEXAMPLE",
        "DBClusterArn": "arn:aws:rds:eu-central-1:123456789012:cluster:my-new-80-
cluster",
        "AssociatedRoles": [],
        "IAMDatabaseAuthenticationEnabled": false,
        "ClusterCreateTime": "2022-07-05T20:45:42.171000+00:00",
        "EngineMode": "provisioned",
        "DeletionProtection": false,
        "HttpEndpointEnabled": false,
        "CopyTagsToSnapshot": false,
        "CrossAccountClone": false,
        "DomainMemberships": [],
        "TagList": []
    }
}
```

建立主要 (寫入器) 資料庫執行個體

若要建立主要 (寫入器) 資料庫執行個體，請使用 `create-db-instance` 命令。下列選項是必要的：

- `--db-cluster-identifier` – 要還原的資料庫叢集名稱。
- `--db-instance-identifier` – 主要資料庫執行個體的名稱。
- `--db-instance-class` – 要用於主要資料庫執行個體的執行個體類別。在此範例中，我們使用 `db.t3.medium`.

> **Note**
>
> 建議您在開發、測試伺服器或其他非生產伺服器時，僅使用 T 資料庫執行個體類別。如需詳細了解 T 執行個體類別，請參閱 資料庫執行個體類別的類型。

- `--engine` – 主要資料庫執行個體的資料庫引擎。必須與還原的資料庫叢集使用相同的資料庫引擎。

  下列選項是選用的：

  - `aurora-mysql` – 與 Aurora MySQL 5.7 及 8.0 相容。
  - `aurora-postgresql` – 與 Aurora PostgreSQL 相容。

  在此範例中，我們使用 `aurora-mysql`.

以下範例會在名為 `my-new-80-cluster` 的已還原的 Aurora MySQL 8.0 相容資料庫叢集中，建立名為 `my-new-80-cluster-instance` 的主要 (寫入器) 資料庫執行個體。

建立主要資料庫執行個體

- 請使用下列其中一個命令。

  對於 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
    --db-cluster-identifier my-new-80-cluster \
    --db-instance-identifier my-new-80-cluster-instance \
    --db-instance-class db.t3.medium \
    --engine aurora-mysql
```

  在 Windows 中：

```
aws rds create-db-instance ^
    --db-cluster-identifier my-new-80-cluster ^
    --db-instance-identifier my-new-80-cluster-instance ^
    --db-instance-class db.t3.medium ^
    --engine aurora-mysql
```

輸出結果與以下內容相似。

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "my-new-80-cluster-instance",
        "DBInstanceClass": "db.t3.medium",
        "Engine": "aurora-mysql",
        "DBInstanceStatus": "creating",
        "MasterUsername": "admin",
        "AllocatedStorage": 1,
        "PreferredBackupWindow": "01:55-02:25",
        "BackupRetentionPeriod": 14,
        "DBSecurityGroups": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-########",
                "Status": "active"
            }
        ],
        "DBParameterGroups": [
            {
                "DBParameterGroupName": "default.aurora-mysql8.0",
                "ParameterApplyStatus": "in-sync"
            }
        ],
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-2305ca49",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1a"
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1b"
                    },
                    "SubnetOutpost": {},
```

```
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1c"
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
                }
            ]
        },
        "PreferredMaintenanceWindow": "sat:02:41-sat:03:11",
        "PendingModifiedValues": {},
        "MultiAZ": false,
        "EngineVersion": "8.0.mysql_aurora.3.02.0",
        "AutoMinorVersionUpgrade": true,
        "ReadReplicaDBInstanceIdentifiers": [],
        "LicenseModel": "general-public-license",
        "OptionGroupMemberships": [
            {
                "OptionGroupName": "default:aurora-mysql-8-0",
                "Status": "in-sync"
            }
        ],
        "PubliclyAccessible": false,
        "StorageType": "aurora",
        "DbInstancePort": 0,
        "DBClusterIdentifier": "my-new-80-cluster",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:eu-central-1:534026745191:key/#######-5ccc-49cc-8aaa-
############",
        "DbiResourceId": "db-5C6UT5PU0YETANOTHEREXAMPLE",
        "CACertificateIdentifier": "rds-ca-2019",
        "DomainMemberships": [],
        "CopyTagsToSnapshot": false,
        "MonitoringInterval": 0,
        "PromotionTier": 1,
        "DBInstanceArn": "arn:aws:rds:eu-central-1:123456789012:db:my-new-80-cluster-
instance",
        "IAMDatabaseAuthenticationEnabled": false,
        "PerformanceInsightsEnabled": false,
        "DeletionProtection": false,
        "AssociatedRoles": [],
```

```
        "TagList": []
    }
}
```

在 Amazon Aurora 叢集中監控指標

Amazon Aurora 會使用複寫的資料庫伺服器叢集。通常，監控 Aurora 叢集需要檢查多個資料庫執行個體的運作狀態。這些執行個體可能具有專業的角色，這些角色通常會處理寫入操作、唯讀操作或兩者的組合。您也可以透過測量複寫延遲，來監控叢集的整體運作狀態。這是一個資料庫執行個體所做變更可供其他執行個體使用的持續時間。

主題

- 監控計畫
- 效能基準
- 效能指導方針
- Amazon Aurora 的監控工具
- 檢視叢集狀態
- Amazon Aurora 的建議
- 在 Amazon RDS 主控台中檢視指標
- 使用 Performance Insights 儀表板檢視組合指標
- 使用 Amazon CloudWatch 監控 Amazon Aurora 指標
- 使用 CloudWatch Database Insights 監控 Amazon Aurora 資料庫
- 在 Amazon Aurora 上使用績效詳情監控資料庫負載
- 使用 Amazon DevOps Guru for Amazon RDS 分析 Aurora 效能異常
- 使用增強型監控來監控作業系統指標
- Amazon Aurora 的指標參考

監控計畫

開始監控 Amazon Aurora，請先建立監控計劃。此計畫應該回答下列問題：

- 監控目標是什麼？
- 監控哪些資源？
- 監控這些資源的頻率為何？
- 將使用哪些監控工具？

- 誰將執行監控任務？
- 發生問題時應該通知誰？

效能基準

若要達到監控目標，您需要建立一個基準。若要這麼做，請在 Amazon Aurora 環境中以不同的時間、不同的負載條件下測量效能。您可以監控如下所示的指標：

- 網路輸送量
- 用戶端連線
- 用於讀取、寫入或中繼資料操作的 I/O
- 資料庫執行個體的爆量點數餘額

建議您儲存 Amazon Aurora 的歷史效能資料。使用儲存的資料，您可以比較當前的效能與過去的趨勢。您也可以區分正常與異常的效能模式，並設計技術來解決問題。

效能指導方針

一般來說，效能指標的可接受值依據您應用程式相對於基準所執行的內容而定。調查距離基準的一致或趨勢變異。下列指標通常是效能問題的來源：

- 高 CPU 或 RAM 耗用量 – CPU 或 RAM 耗用量若符合應用程式的目標 (如輸送量或並行) 且預期的值較高時，這樣的消耗量值就可能是合理的。
- 磁碟空間消耗量 – 如果使用的空間持續保持在等於或高於總磁碟空間的 85%，請調查磁碟空間消耗量。看看從執行個體刪除資料或將資料封存至不同的系統來釋出空間是否可行。
- 網路流量 – 對於網路流量，請洽系統管理員，以了解您的網域網路和網際網路連線預期的輸送量。調查網路流量的傳輸量是否如預期一致地降低。
- 資料庫連線 – 如果您看到大量使用者連線，同時執行個體效能下降且回應時間延長，請考慮限制資料庫連線。資料庫執行個體使用者連接的最佳數量，將因執行個體類別和要執行的操作複雜性而不同。若要判定資料庫連線的數目，方法是將資料庫執行個體與 `User Connections` 參數設為 0 (無限制) 以外之值的參數群組建立關聯。您可以使用現有的參數群組或建立新的參數群組。如需更多詳細資訊，請參閱 Amazon Aurora 的參數群組。
- IOPS 指標 – IOPS 指標的預期值視磁碟規格和伺服器組態而定，因此請使用您的基準來了解何謂典型。調查值是否與您的基準一致地不同。為獲得最佳 IOPS 效能，請確定您的一般工作集將放入記憶體中，以將讀取和寫入操作降到最低。

當效能落在您建立的基準之外時，您可能需要進行變更，以最佳化工作負載的資料庫可用性。例如，您可能需要變更資料庫執行個體的執行個體類別。或者，您可能需要變更用戶端可用的資料庫執行個體和讀取複本的數目。

Amazon Aurora 的監控工具

監控是維護 Amazon Aurora 及其他 AWS 解決方案可靠性、可用性和效能的重要部分。 AWS 提供各種監控工具，可讓您監看 Amazon Aurora、在發生錯誤時回報，以及適時採取自動動作。

主題

- 自動化監控工具
- 手動監控工具

自動化監控工具

建議您盡可能自動化監控任務。

主題

- Amazon Aurora 叢集狀態和建議
- Amazon Aurora 的 Amazon CloudWatch 指標
- Amazon RDS Performance Insights 和作業系統監控
- 整合服務

Amazon Aurora 叢集狀態和建議

您可以使用下列自動化工具來監看 Amazon Aurora，並在發生錯誤時回報：

- Amazon Aurora 叢集狀態 — 使用 Amazon RDS 主控台 AWS CLI、 或 RDS API 檢視叢集目前狀態的詳細資訊。
- Amazon Aurora建議 — 回應針對資料庫資源提供的自動化建議，例如資料庫執行個體、資料庫叢集、以及資料庫叢集參數群組。如需詳細資訊，請參閱Amazon Aurora 的建議。

Amazon Aurora 的 Amazon CloudWatch 指標

Amazon Aurora 會與 Amazon CloudWatch 整合，以取得其他監控功能。

- Amazon CloudWatch – 此服務 AWS 會即時監控您的 AWS 資源和您在 上執行的應用程式。您可以搭配 Amazon Aurora 使用下列 Amazon CloudWatch 功能：

- Amazon CloudWatch 指標 – Amazon Aurora 會每分鐘針對每個作用中資料庫傳送指標至 CloudWatch。CloudWatch 中的 Amazon RDS 指標不會收取額外費用。如需詳細資訊，請參閱 Amazon Aurora 的 Amazon CloudWatch 指標。
- Amazon CloudWatch 警示 – 您可以在特定期間內觀看單一 Amazon Aurora 指標。然後，您可以根據相對於您所設定臨界值的指標值執行一或多個動作。

Amazon RDS Performance Insights 和作業系統監控

您能夠使用下列自動化工具來監控 Amazon Aurora 效能：

- Amazon RDS Performance Insights：評估資料庫的負載，以及判斷在何時何處採取動作。如需詳細資訊，請參閱在 Amazon Aurora 上使用績效詳情監控資料庫負載。
- Amazon RDS Enhanced Monitoring (Amazon RDS 增強型監控)：即時查看作業系統的指標。如需詳細資訊，請參閱使用增強型監控來監控作業系統指標。

整合服務

下列 AWS 服務已與 Amazon Aurora 整合：

- Amazon EventBridge 為無伺服器事件匯流排服務，可讓您輕鬆將應用程式與來自各種來源的資料互相連線。如需詳細資訊，請參閱監控 Amazon Aurora 事件。
- Amazon CloudWatch Logs 可讓您監控、存放及存取來自 Amazon Aurora 執行個體、CloudTrail 或其他來源的日誌檔案。如需詳細資訊，請參閱監控 Amazon Aurora 日誌檔案。
- AWS CloudTrail 擷取您 AWS 帳戶 發出或代表發出的 API 呼叫和相關事件，並傳送日誌檔案至您指定的 Amazon S3 儲存貯體。如需詳細資訊，請參閱在 AWS CloudTrail 中監控 Amazon Aurora API 呼叫。
- 資料庫活動串流是一項 Amazon Aurora 功能，提供有關您 資料庫叢集的近乎即時活動串流。如需詳細資訊，請參閱使用資料庫活動串流來監控 Amazon Aurora。
- DevOps Guru for RDS 是 Amazon DevOps Guru 的一項功能，可將機器學習套用至 Amazon Aurora 資料庫的 Performance Insights 指標。如需詳細資訊，請參閱使用 Amazon DevOps Guru for Amazon RDS 分析 Aurora 效能異常。

手動監控工具

您必須手動監控 CloudWatch 警示未涵蓋的項目。Amazon RDS、 AWS Trusted Advisor CloudWatch 和其他 AWS 主控台儀表板可讓您at-a-glance檢視 AWS 環境的狀態。建議您也檢查資料庫執行個體上的日誌檔。

- 從 Amazon RDS 主控台中，您可以監控資源的下列項目：
  - 資料庫執行個體的連線數目
  - 資料庫執行個體的讀取和寫入操作數量
  - 資料庫執行個體目前正在使用的儲存體數量
  - 正在針對資料庫執行個體使用的記憶體和 CPU 數量
  - 進出資料庫執行個體的網路流量
- 從 Trusted Advisor 儀表板，您可以檢閱下列成本最佳化、安全性、容錯能力和效能改善檢查：
  - Amazon RDS 閒置資料庫執行個體
  - Amazon RDS 安全群組存取風險
  - Amazon RDS 備份
  - Amazon RDS 異地同步備份
  - Aurora 資料庫執行個體存取能力

  如需這些檢查的詳細資訊，請參閱 Trusted Advisor 最佳實務 (檢查)。
- CloudWatch 首頁顯示：
  - 目前警示與狀態
  - 警示與資源的圖表
  - 服務運作狀態

  此外，您可以使用 CloudWatch 執行下列動作：
  - 建立自訂儀表板以監控您注重的服務。
  - 用於疑難排解問題以及探索驅勢的圖形指標資料。
  - 搜尋和瀏覽您的所有 AWS 資源指標。
  - 建立與編輯要通知發生問題的警示。

檢視叢集狀態

您可以使用 Amazon RDS 主控台快速存取資料庫叢集的狀態。

主題

- 檢視 Amazon Aurora 資料庫叢集
- 檢視資料庫叢集狀態
- 在 Aurora 叢集中檢視 資料庫執行個體狀態

檢視 Amazon Aurora 資料庫叢集

您有數個選項，可用於檢視 Amazon Aurora 資料庫叢集以及資料庫叢集中資料庫執行個體的相關資訊。

- 您可由導覽窗格選擇 Databases (資料庫)，以便於 Amazon RDS 主控台檢視資料庫叢集和資料庫執行個體。
- 您可以使用 AWS Command Line Interface () 取得資料庫叢集和資料庫執行個體資訊AWS CLI。
- 您可以使用 Amazon RDS API，來取得資料庫叢集和資料庫執行個體資訊。

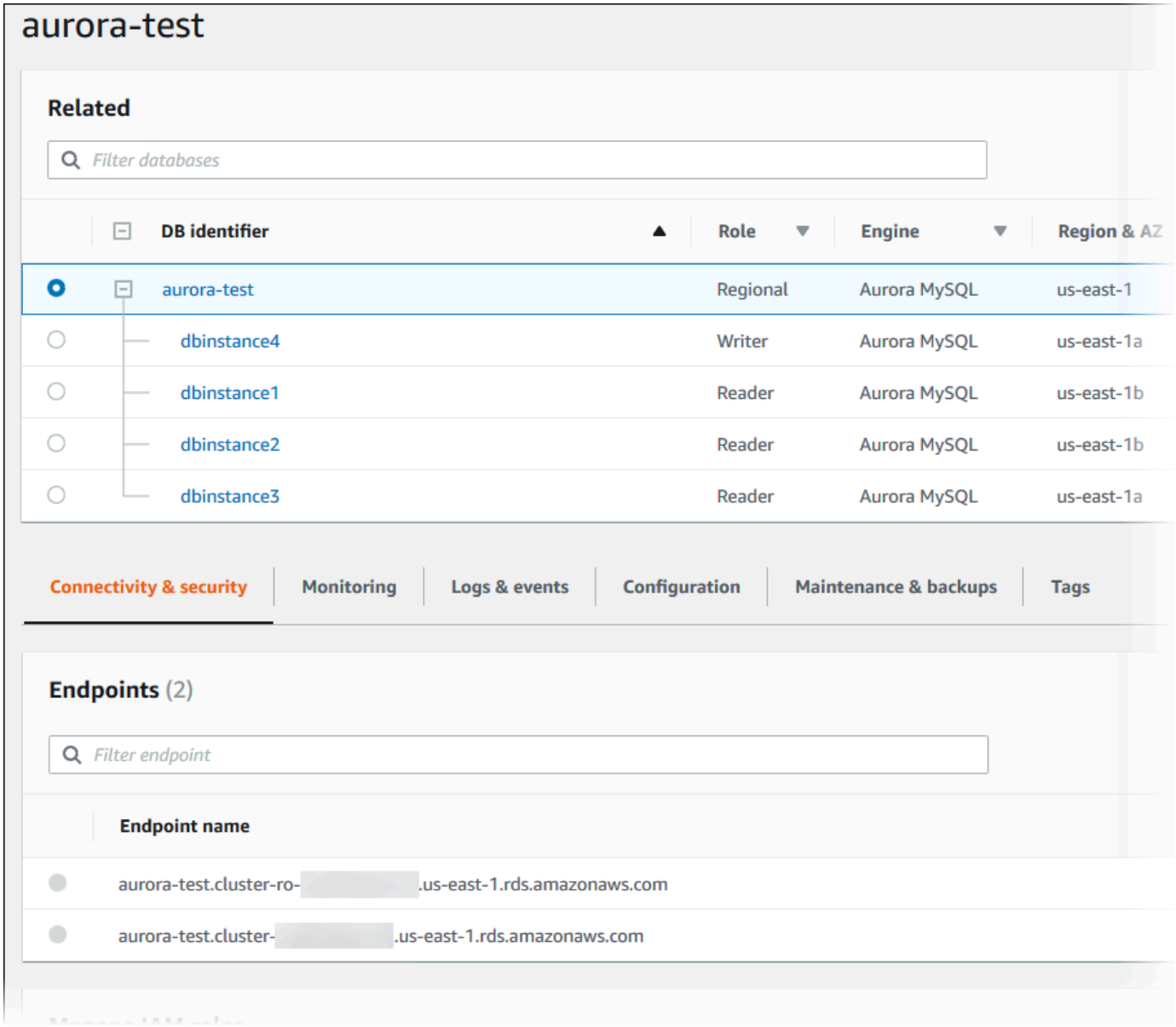
主控台

在 Amazon RDS 主控台中，透過從主控台的導覽窗格中選擇 Databases (資料庫)，可檢視資料庫叢集詳細資訊。您也可以檢視屬於 Amazon Aurora 資料庫叢集成員的資料庫執行個體詳細資訊。

在 Amazon RDS 主控台中檢視或修改資料庫叢集

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 從清單中選擇您要檢視的 Aurora 資料庫叢集名稱。

   例如，下圖顯示名為 `aurora-test` 之資料庫叢集的詳細資訊頁面。資料庫叢集有四個資料庫執行個體出現在 DB identifier (資料庫識別符) 清單中。寫入器資料庫執行個體 `dbinstance4` 是資料庫叢集的主要資料庫執行個體。

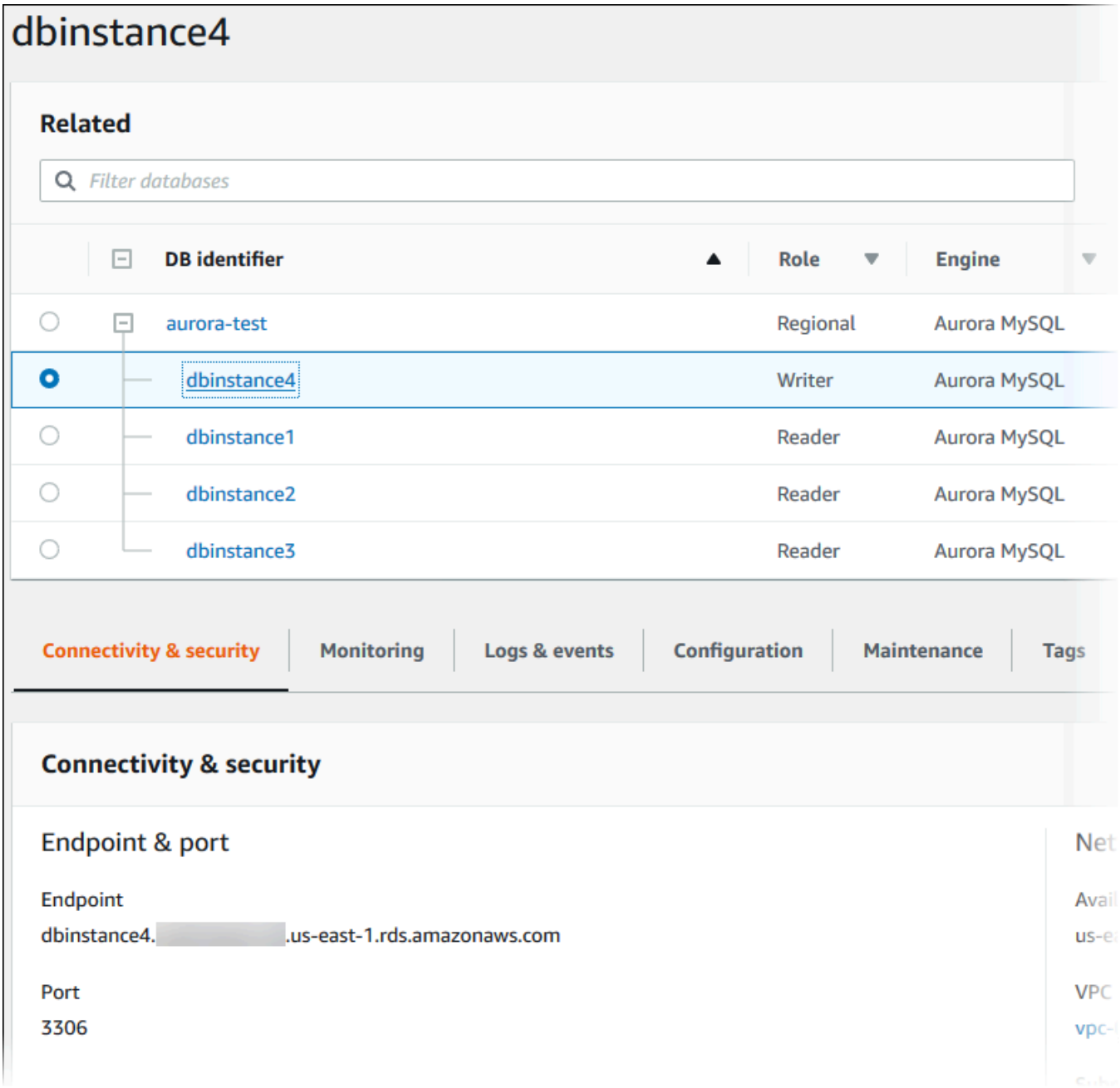


4. 若要修改資料庫叢集，請從清單選取資料庫叢集，然後選擇 Modify (修改)。

在 Amazon RDS 主控台中檢視或修改資料庫叢集的資料庫執行個體

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/rds/ 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 執行以下任意一項：
   - 若要檢視資料庫執行個體，請從清單中選擇一個身為 Aurora 資料庫叢集成員的資料庫執行個體。

例如，如果您選擇 dbinstance4 資料庫執行個體識別符，主控台會顯示 dbinstance4 資料庫執行個體的詳細資訊頁面，如下圖所示。



- 若要修改資料庫執行個體，請從清單中選擇資料庫執行個體，然後選擇 Modify (修改)。如需修改資料庫叢集的詳細資訊，請參閱修改 Amazon Aurora 資料庫叢集。

AWS CLI

若要使用 檢視資料庫叢集資訊 AWS CLI，請使用 describe-db-clusters 命令。例如，下列 AWS CLI 命令會列出所設定 AWS 帳戶修改區域中所有資料庫叢集us-east-1的資料庫叢集資訊。

```
aws rds describe-db-clusters --region us-east-1
```

如果您的 AWS CLI 設定為 JSON 輸出，命令會傳回下列輸出。

```
{
    "DBClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "Endpoint": "sample-cluster1.cluster-123456789012.us-
east-1.rds.amazonaws.com"
            "AllocatedStorage": 1,
            "DBClusterIdentifier": "sample-cluster1",
            "MasterUsername": "mymasteruser",
            "EarliestRestorableTime": "2023-03-30T03:35:42.563Z",
            "DBClusterMembers": [
                {
                    "IsClusterWriter": false,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "sample-replica"
                },
                {
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "sample-primary"
                }
            ],
            "Port": 3306,
            "PreferredBackupWindow": "03:34-04:04",
            "VpcSecurityGroups": [
                {
                    "Status": "active",
                    "VpcSecurityGroupId": "sg-ddb65fec"
                }
            ],
            "DBSubnetGroup": "default",
            "StorageEncrypted": false,
            "DatabaseName": "sample",
```

```
            "EngineVersion": "5.7.mysql_aurora.2.11.0",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "BackupRetentionPeriod": 1,
            "AvailabilityZones": [
                "us-east-1b",
                "us-east-1c",
                "us-east-1d"
            ],
            "LatestRestorableTime": "2023-03-31T20:06:08.903Z",
            "PreferredMaintenanceWindow": "wed:08:15-wed:08:45"
        },
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "Endpoint": "aurora-sample.cluster-123456789012.us-
east-1.rds.amazonaws.com",
            "AllocatedStorage": 1,
            "DBClusterIdentifier": "aurora-sample-cluster",
            "MasterUsername": "mymasteruser",
            "EarliestRestorableTime": "2023-03-30T10:21:34.826Z",
            "DBClusterMembers": [
                {
                    "IsClusterWriter": false,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "aurora-replica-sample"
                },
                {
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "aurora-sample"
                }
            ],
            "Port": 3306,
            "PreferredBackupWindow": "10:20-10:50",
            "VpcSecurityGroups": [
                {
                    "Status": "active",
                    "VpcSecurityGroupId": "sg-55da224b"
                }
            ],
            "DBSubnetGroup": "default",
            "StorageEncrypted": false,
            "DatabaseName": "sample",
            "EngineVersion": "5.7.mysql_aurora.2.11.0",
```

```
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "BackupRetentionPeriod": 1,
            "AvailabilityZones": [
                "us-east-1b",
                "us-east-1c",
                "us-east-1d"
            ],
            "LatestRestorableTime": "2023-03-31T20:00:11.491Z",
            "PreferredMaintenanceWindow": "sun:03:53-sun:04:23"
        }
    ]
}
```

RDS API

若要使用 Amazon RDS API 檢視資料庫叢集資訊，請使用 DescribeDBClusters 操作。

檢視資料庫叢集狀態

資料庫叢集的狀態會指出其目前的操作狀態。您可以使用 Amazon RDS 主控台 AWS CLI、 或 API 來檢視資料庫叢集和叢集執行個體的狀態。

> **Note**
>
> Aurora 也使用另一個狀態，即顯示在 Amazon RDS 主控台的 Maintenance (維護) 欄中的 maintenance status (維護狀態)。此值會指出任何維護修補程式的狀態，這些維護修補程式皆需套用至資料庫叢集。維護狀態與資料庫叢集的狀態互不相關。如需維護狀態的詳細資訊，請參閱將更新套用至資料庫叢集。

您可以在下表中找到資料庫叢集的可能狀態值。

資料庫叢集狀態	計費	描述
可用性	計費	資料庫叢集可供修改。當 Aurora 無伺服器叢集可用且已暫停時，系統只會向您收取儲存體費用。
Backing-up (備份)	計費	目前正在備份資料庫叢集。
Backtracking (恢復)	計費	目前正在恢復資料庫叢集。此狀態僅適用於 Aurora MySQL。
Cloning-failed	不計費	複製資料庫叢集失敗。
正在建立	不計費	正在建立資料庫叢集。無法存取正在建立的資料庫叢集。
正在刪除	不計費	正在刪除資料庫叢集。
Failing-over	計費	正在從主要執行個體容錯移轉至 Aurora 複本。
Inaccessible-encryption-credentials	不計費	AWS KMS key 用於加密或解密資料庫叢集的 無法存取或復原。

資料庫叢集狀態	計費	描述
Inaccessible-encryption-credentials-recoverable	針對儲存計費	無法存取用來加密或解密資料庫叢集的 KMS 金鑰。但是，如果 KMS 金鑰處於作用中狀態，則重新啟動資料庫叢集可以將其復原。 如需詳細資訊，請參閱加密 Amazon Aurora 資料庫叢集。
Maintenance (維護)	計費	Amazon RDS 正在將維護更新套用到資料庫叢集。此狀態適用於 RDS 事先排程的資料庫叢集層級維護。
Migrating	計費	正在將資料庫叢集快照還原至資料庫叢集。
Migration-failed	不計費	無法進行遷移。
Modifying (正在修改)	計費	客戶請求修改資料庫叢集，因此正在進行修改。
Promoting	計費	正在將僅供讀取複本提升為獨立的資料庫叢集。
Preparing-data-migration	計費	Amazon RDS 正在準備將資料遷移至 Aurora。
重新命名	計費	客戶請求重新命名資料庫叢集，因此正在重新命名。
Resetting-master-credentials (重新設定主要登入資料)	計費	客戶請求重新設定資料庫叢集的主要登入資料，因此正在重新設定。
啟動	針對儲存計費	已啟動資料庫叢集。
已停止	針對儲存計費	已停止資料庫叢集。
Stopping (正在停止)	針對儲存計費	正在停止資料庫叢集。

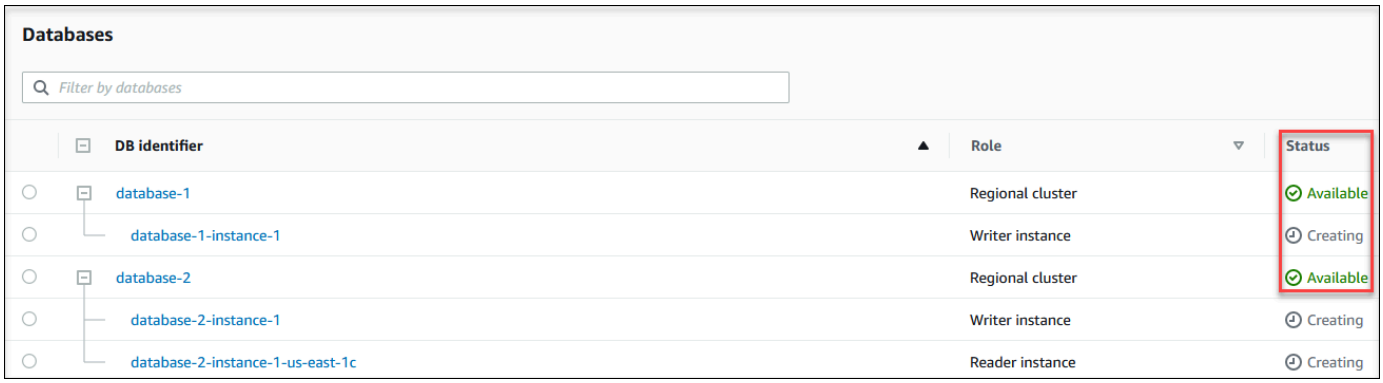
資料庫叢集狀態	計費	描述
Storage-optimization (儲存最佳化)	計費	正在修改您的資料庫執行個體以變更儲存大小或類型。資料庫執行個體完全可以正常運作。不過，雖然資料庫執行個體的狀態是 storage-optimization (儲存最佳化)，但您無法請求對資料庫執行個體的儲存做任何變更。儲存最佳化程序通常很短，但有時甚至可能會超過 24 小時。
Update-iam-db-auth	計費	正在更新資料庫叢集的 IAM 授權。
Upgrading (正在升級)	計費	正在升級資料庫叢集引擎或作業系統版本。
upgrade_failed	不計費	資料庫叢集無法升級至支援的版本。Aurora 會建立字首為 的最終快照`rds-final` 。

主控台

檢視資料庫叢集的狀態

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。

   Databases page (資料庫頁面) 會與資料庫叢集的清單一起顯示。針對每個資料庫叢集，會顯示狀態值。



CLI

若要僅檢視資料庫叢集的狀態，請在 中使用下列查詢 AWS CLI。

```
aws rds describe-db-clusters --query 'DBClusters[*].[DBClusterIdentifier,Status]' --
output table
```

在 Aurora 叢集中檢視 資料庫執行個體狀態

Aurora 叢集中資料庫執行個體的狀態會指出其目前的操作狀態。您可以使用下列程序，在 Amazon RDS 主控台、 AWS CLI 命令或 API 操作中檢視叢集的資料庫執行個體狀態。

> **Note**
>
> Amazon RDS 也使用另一個狀態，即顯示在 Amazon RDS 主控台的 Maintenance (維護) 欄中的 maintenance status (維護狀態)。此值指出需要套用至資料庫執行個體的任何維護修補程式的狀態。維護狀態與資料庫執行個體的狀態互不相關。如需維護狀態的詳細資訊，請參閱將更新套用至資料庫叢集。

您可以在下表中找到資料庫執行個體的可能狀態值。此表還列出計費方式為依資料庫執行個體和儲存計費、只針對儲存計費、或不計費。在全是資料庫執行個體狀態的情況下，一律要為備份用途付費。

資料庫執行個體狀態	計費	描述
有用	計費	資料庫執行個體可供修改。
backing-up (備份)	計費	目前正在備份資料庫執行個體。
backtracking (恢復)	計費	目前正在恢復資料庫執行個體。此狀態僅適用於 Aurora MySQL。
configuring-enhanced-monitoring (設定增強型監控)	計費	正在啟用或停用此資料庫執行個體的增強型監控。
configuring-iam-database-auth	計費	AWS Identity and Access Management 正在啟用或停用此資料庫執行個體的 (IAM) 資料庫身分驗證。
configuring-log-exports	計費	針對此資料庫執行個體，正在啟用或停用將日誌檔案發佈至 Amazon CloudWatch Logs。
converting-to-vpc	計費	資料庫執行個體正在從不是在 Amazon Virtual Private Cloud (Amazon VPC) 中的資料庫執行個體，轉換成位於 Amazon VPC 中的資料庫執行個體。

資料庫執行個體狀態	計費	描述
建立	不計費 (非 PITR) 計費 (僅限 PITR)	正在建立資料庫執行個體。資料庫執行個體正在建立時無法存取。 如果您在時間點復原 (PITR) 期間還原資料庫，則會在資料庫處於建立狀態時向您收取費用。這是建立狀態產生費用的唯一案例。
delete-precheck	不計費	Amazon RDS 正在驗證僅供讀取複本是否可安全刪除。
正在刪除	不計費	正在刪除資料庫執行個體。
失敗	不計費	資料庫執行個體失效，Amazon RDS 無法復原它。請執行時間點還原，還原到資料庫執行個體的最後可還原時間，以復原資料。
inaccessible-encryption-credentials	不計費	AWS KMS key 用於加密或解密資料庫執行個體的 無法存取或復原。
inaccessible-encryption-credentials-recoverable	針對儲存計費	無法存取用來加密或解密資料庫執行個體的 KMS 金鑰。但是，如果 KMS 金鑰處於作用中狀態，則重新啟動資料庫執行個體可以將其復原。 如需詳細資訊，請參閱加密 Amazon Aurora 資料庫叢集。
incompatible-create	不計費	Amazon RDS 正在嘗試建立資料庫執行個體，但無法這麼做，因為資源與您的資料庫執行個體不相容。例如，如果資料庫執行個體的執行個體設定檔沒有正確的許可，則可能會發生此狀態。
不相容網路	不計費	Amazon RDS 正在嘗試對資料庫執行個體執行復原動作，但失敗，因為 VPC 所處的狀態無法完成此動作。例如，如果子網路中的所有可用 IP 地址都在使用中，Amazon RDS 無法取得資料庫執行個體的 IP 地址，就會出現此狀態。

資料庫執行個體狀態	計費	描述
incompatible-option-group (不相容選項群組)	計費	Amazon RDS 嘗試套用選項群組變更但不成功，且 Amazon RDS 無法轉返到上一個選項群組狀態。如需詳細資訊，請查看資料庫執行個體的 Recent Events (最近事件) 清單。例如，如果選項組包含 TDE 等選項，而資料庫執行個體沒有加密資訊，就會出現此狀態。
incompatible-parameters (不相容參數)	計費	因為資料庫執行個體的資料庫參數群組中指定的參數與資料庫執行個體不相容，Amazon RDS 無法啟動資料庫執行個體。請回復參數變更，或使之與資料庫執行個體相容，以恢復存取您的資料庫執行個體。如需有關不相容參數的詳細資訊，請查看資料庫執行個體的 Recent Events (最近事件) 清單。
incompatible-restore (不相容還原)	不計費	Amazon RDS 無法執行時間點還原。此狀態的常見原因包括使用臨時資料表或將 MyISAM 資料表與 MySQL。
容量不足	不計費	Amazon RDS 無法建立您的執行個體，因為目前無法使用足夠的容量。若要在具有相同執行個體類型的相同可用區域中建立資料庫執行個體，請刪除您的資料庫執行個體，等候幾個小時，然後嘗試再次建立。或者，使用不同的執行個體類別或可用區域建立新的執行個體。
維護	計費	Amazon RDS 正在將維護更新套用到資料庫執行個體。此狀態用於 RDS 事先排程的執行個體層級維護。
修改	計費	由於客戶請求修改資料庫執行個體，正在修改資料庫執行個體。
moving-to-vpc	計費	正在將資料庫執行個體移到新的 Amazon Virtual Private Cloud (Amazon VPC)。

資料庫執行個體狀態	計費	描述
重新開機中 (Rebooting)	計費	由於客戶請求或 Amazon RDS 程序需要重新啟動資料庫執行個體，正在重新啟動資料庫執行個體。
resetting-master-credentials (重新設定主要登入資料)	計費	由於客戶請求重設資料庫執行個體的主要登入資料，正在重設主要登入資料。
重新命名	計費	由於客戶請求重新命名資料庫執行個體，正在重新命名資料庫執行個體。
restore-error (還原錯誤)	計費	資料庫執行個體在嘗試還原到某個時間點或從快照還原時發生錯誤。
開始	針對儲存計費	正在啟動資料庫執行個體。
已停止 (Stopped)	針對儲存計費	已停止資料庫執行個體。
停止中	針對儲存計費	正在停止資料庫執行個體。
storage-config-upgrade	計費	資料庫執行個體的儲存檔案系統組態正在升級中。此狀態僅適用於藍/綠部署內的綠色資料庫，或資料庫執行個體僅供讀取複本。
storage-full (儲存已滿)	計費	資料庫執行個體已達到其儲存容量配置。這是一個緊急狀態，我們建議您立即解決此問題。做法是修改資料庫執行個體以擴展儲存空間。若要避免此情況，可設定 Amazon CloudWatch 警示在儲存空間變少時警告您。
storage-initialization	計費	資料庫執行個體正在從 Amazon S3 載入資料區塊，以便在從快照還原之後最佳化磁碟區效能。它仍然可供操作使用，但在初始化完成之前，效能可能不會處於最高狀態。

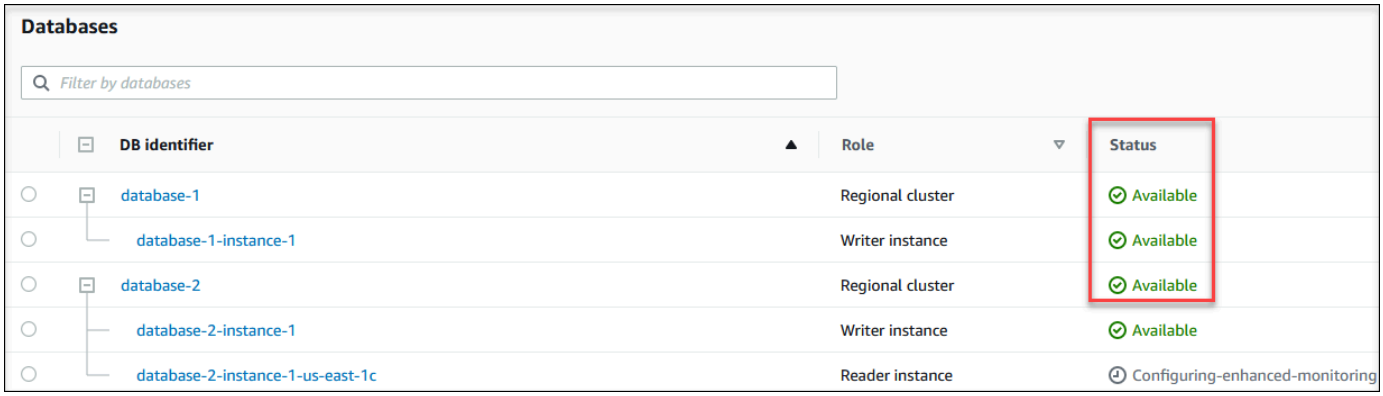
資料庫執行個體狀態	計費	描述
storage-optimization (儲存最佳化)	計費	Amazon RDS 正在最佳化您資料庫執行個體的儲存空間。儲存最佳化程序通常很短，但有時甚至可能會超過 24 小時。 在儲存最佳化期間，資料庫執行個體仍然可用。儲存最佳化是不會影響執行個體可用性的背景程序。
正在升級	計費	正在升級資料庫引擎或作業系統版本。
upgrade_failed	不計費	資料庫執行個體無法升級至支援的版本。Aurora 會建立字首為 的最終快照`rds-final` 。

主控台

檢視資料庫執行個體的狀態

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。

   Databases page (資料庫頁面) 會與資料庫執行個體清單一起顯示。針對叢集中的每個資料庫執行個體，會顯示狀態值。



CLI

若要使用 檢視資料庫執行個體及其狀態資訊 AWS CLI，請使用 describe-db-instances 命令。例如，下列 AWS CLI 命令會列出所有資料庫執行個體資訊 。

```
aws rds describe-db-instances
```

若要檢視特定資料庫執行個體及其狀態，請搭配下列選項呼叫 describe-db-instances 命令：

- `DBInstanceIdentifier` – 資料庫執行個體名稱。

```
aws rds describe-db-instances --db-instance-identifier mydbinstance
```

若要僅檢視所有資料庫執行個體的狀態，請在 中使用下列查詢 AWS CLI。

```
aws rds describe-db-instances --query 'DBInstances[*].
[DBInstanceIdentifier,DBInstanceStatus]' --output table
```

API

若要使用 Amazon RDS API 檢視資料庫執行個體的狀態，請呼叫 DescribeDBInstances 操作。

Amazon Aurora 的建議

Amazon Aurora 針對資料庫資源提供自動化建議，例如資料庫執行個體、資料庫叢集、以及資料庫叢集參數群組。這些建議會分析資料庫叢集組態、資料庫執行個體組態、使用情形及效能資料，以提供最佳實務指南。

Amazon RDS Performance Insights 會監控特定指標，以及透過分析哪些層級可能對特定資源帶來問題，來自動建立閾值。當新的指標值在指定期間內超過預先定義的閾值時，Performance Insights 會產生主動建議。此建議有助於防止未來的資料庫效能影響。例如，當連線至資料庫的工作階段未執行作用中工作，但可以封鎖資料庫資源時，Aurora PostgreSQL 執行個體會產生「交易中閒置」建議。若要接收主動建議，您必須開啟具有付費方案保留期的 Performance Insights。如需開啟 Performance Insights 的詳細資訊，請參閱開啟和關閉 Aurora 的 Performance Insights。如需 Performance Insights 的定價和資料保留詳細資訊，請參閱 Performance Insights 的定價和資料保留。

DevOps Guru for RDS 會監控特定指標，以偵測指標的行為何時變得高度不尋常或異常。系統會將這些異常報告為具有建議的被動深入分析。例如，DevOps Guru for RDS 可能建議您考慮增加 CPU 容量，或調查導致資料庫負載的等待事件。DevOps Guru for RDS 也提供閾值型主動建議。對於這些建議，您必須開啟 DevOps Guru for RDS。如需開啟 DevOps Guru for RDS 的詳細資訊，請參閱開啟 DevOps Guru 並指定資源涵蓋範圍。

建議將處於下列任何狀態：作用中、已關閉、待定或已解決。已解決的建議有效期為 365 天。

您可以檢視或關閉建議。您可以立即套用以組態為基礎的作用中建議、將其排定在下一個維護時段，或將其關閉。對於閾值型主動和機器學習型被動建議，您需要檢閱問題的建議原因，然後執行建議的動作來修正問題。

下列 AWS 區域 中支援建議：

- 美國東部 (俄亥俄)
- 美國東部 (維吉尼亞北部)
- 美國西部 (加利佛尼亞北部)
- 美國西部 (奧勒岡)
- 亞太區域 (孟買)
- 亞太區域 (首爾)
- 亞太區域 (新加坡)
- 亞太區域 (雪梨)
- 亞太區域 (東京)

- 加拿大 (中部)
- 歐洲 (法蘭克福)
- 歐洲 (愛爾蘭)
- 歐洲 (倫敦)
- 歐洲 (巴黎)
- 歐洲 (斯德哥爾摩)
- 南美洲 (聖保羅)

在下列各節中，了解如何檢視、套用、關閉和修改 Amazon Aurora 中的建議。

主題

- 檢視 Amazon Aurora 建議
- 套用 Amazon Aurora 建議
- 關閉 Amazon Aurora 建議
- 將已關閉的 Amazon Aurora 建議修改為作用中建議
- Amazon Aurora 建議參考

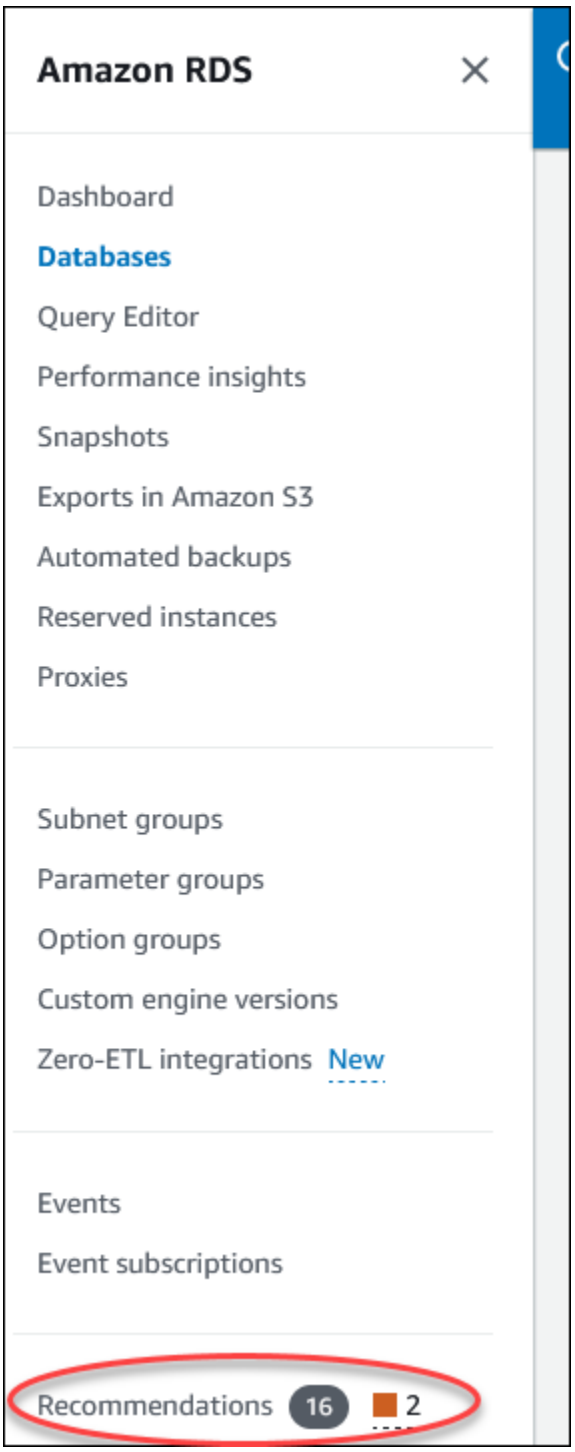
檢視 Amazon Aurora 建議

使用 Amazon RDS 主控台，您可以檢視資料庫資源的 Amazon Aurora 建議。對於資料庫叢集，會顯示資料庫叢集及其執行個體的建議。

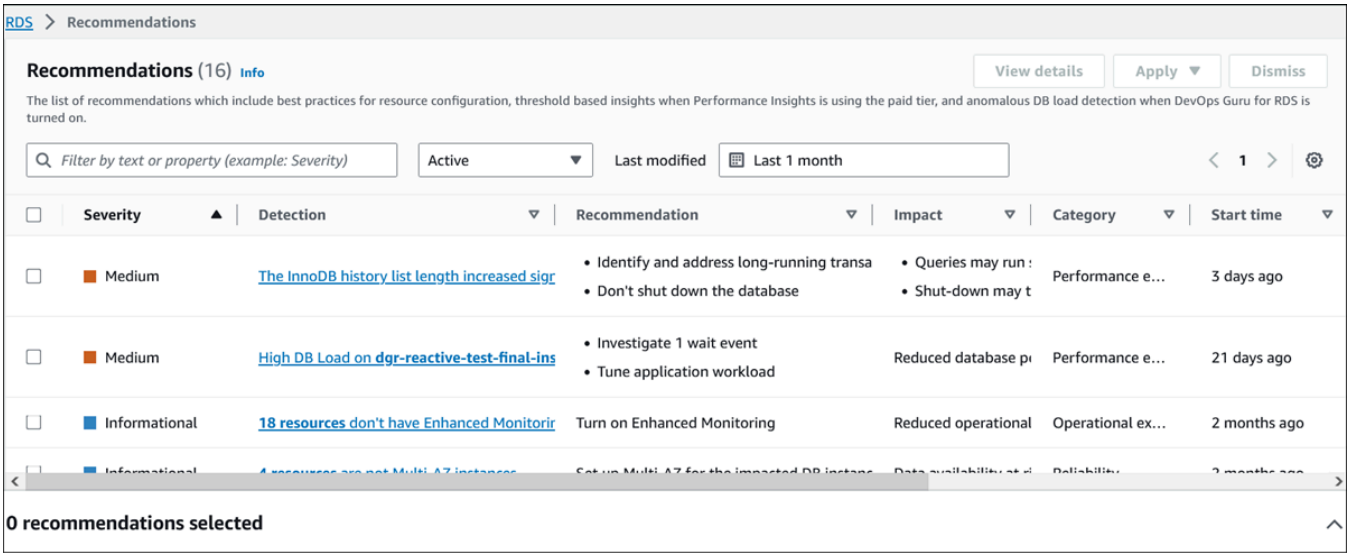
主控台

檢視 Amazon Aurora 建議

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，執行下列任一操作：
   - 選擇建議。您可以在建議旁取得資源的作用中建議數量，以及上個月產生之最高嚴重性的建議數量。若要尋找每個嚴重性的作用中建議數目，請選擇顯示最高嚴重性的數目。

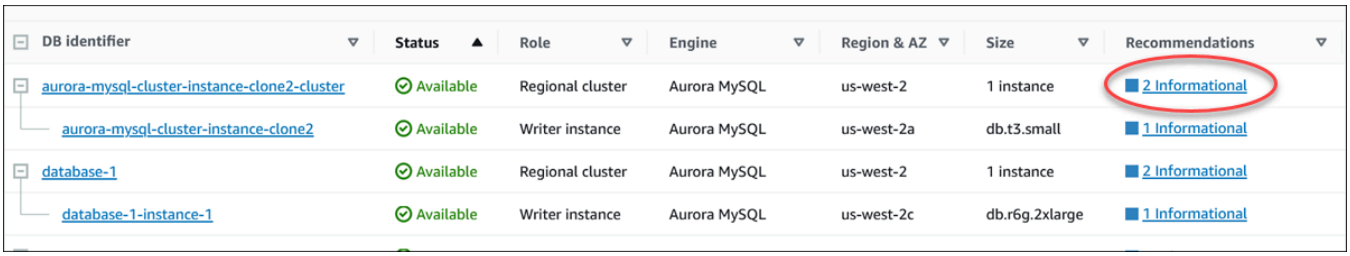


根據預設，建議頁面會顯示上個月的新建議清單。Amazon Aurora 會為您帳戶中的所有資源提供建議，並依其嚴重性排序建議。

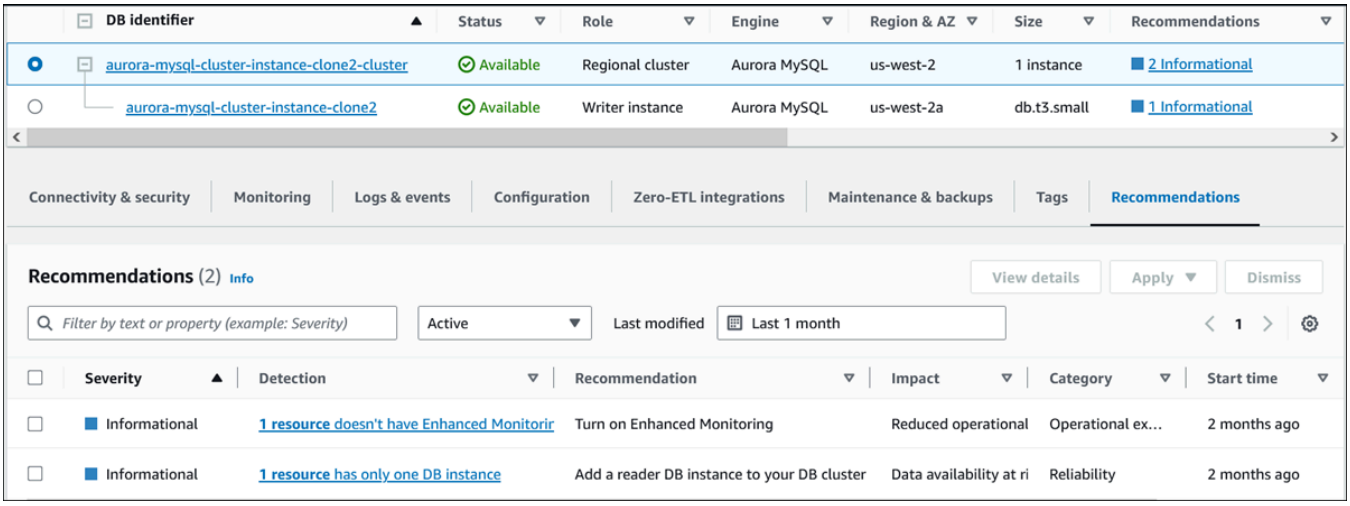


您可以選擇建議來檢視頁面底部的區段，其中包含受影響的資源，以及如何套用建議的詳細資訊。

- 在資料庫頁面中，選擇資源的建議。



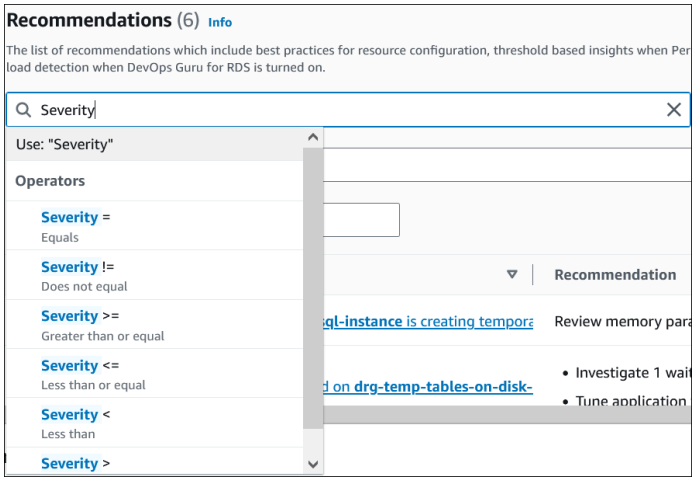
建議索引標籤會顯示所選取資源的建議及其詳細資訊。



建議有下列詳細資訊：

- 嚴重性 – 問題的隱含層級。嚴重性等級為高、中、低和資訊。
- 偵測 – 受影響資源的數量和問題的簡短描述。選擇此連結以檢視建議和分析詳細資訊。
- 建議 – 要套用之建議動作的簡短描述。
- 影響 – 不套用建議時可能影響的簡短描述。
- 類別 – 建議的類型。這些類別包括效能效率、安全性、可靠性、成本最佳化、卓越營運和永續性。
- 狀態 – 建議的目前狀態。可能的狀態為全部、作用中、已關閉、已解決和待定。
- 開始時間 – 問題開始的時間。例如，18 小時前。
- 上次修改 – 系統因為嚴重性變更的上次更新建議的時間，或您回應建議的時間。例如，10 小時前。
- 結束時間 – 問題結束的時間。不會顯示任何持續問題的時間。
- 資源識別符 – 一或多個資源的名稱。

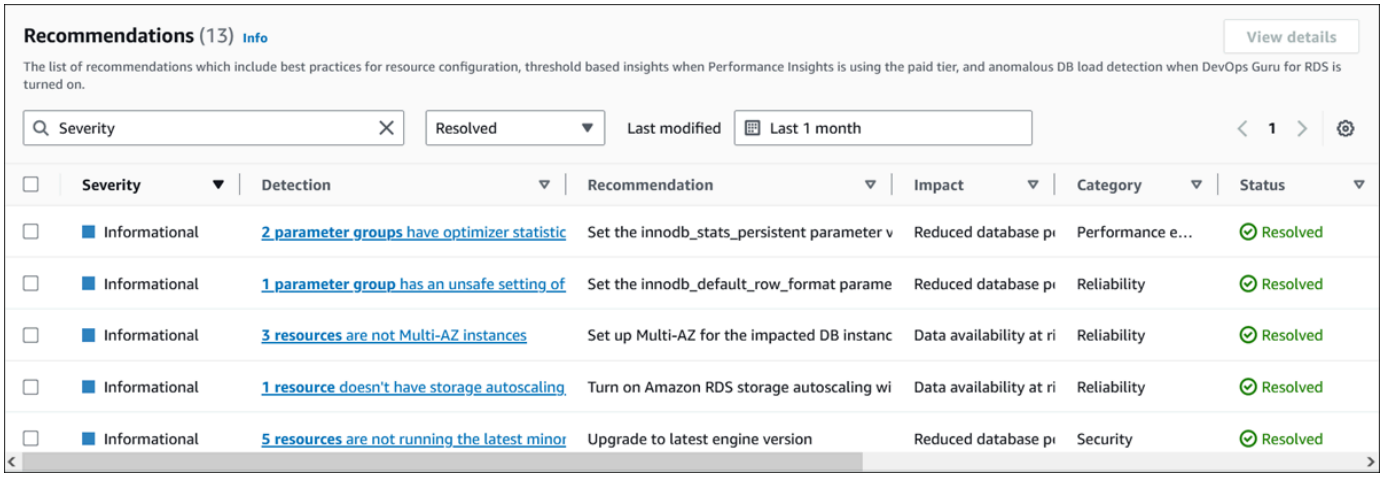
3. (選用) 在欄位中選擇嚴重性或類別運算子，以篩選建議清單。



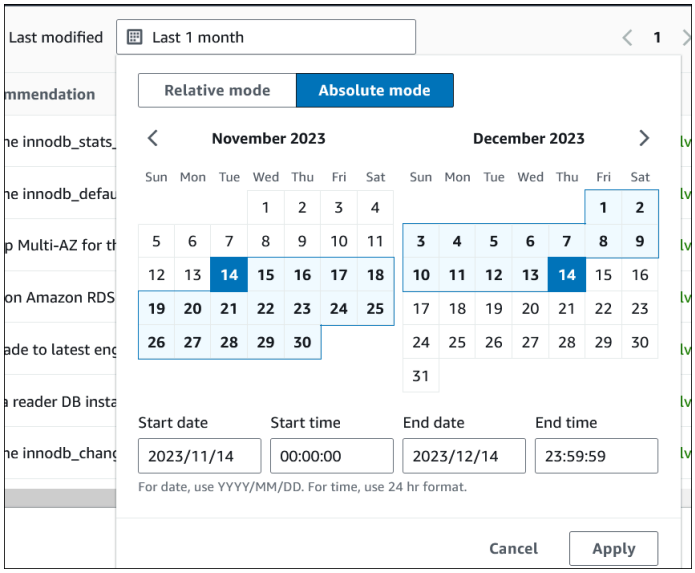
所選取操作的建議隨即顯示。

4. (選用) 選擇下列任一建議狀態：

- 作用中 (預設值) – 顯示您可以套用、排程到下一個維護時段或關閉的目前建議。
- 全部 – 顯示具有目前狀態的所有建議。
- 已關閉 – 顯示已關閉的建議。
- 已解決 – 顯示已解決的建議。
- 待定 – 顯示建議動作正在進行中或排程到下一個維護時段的建議。



5. (選用) 在上次修改中選擇相對模式或絕對模式，以修改時段。建議頁面會顯示時間期間內產生的建議。預設時間期間是上個月。在絕對模式中，您可以選擇時間期間，或在開始日期和結束日期欄位中輸入時間。

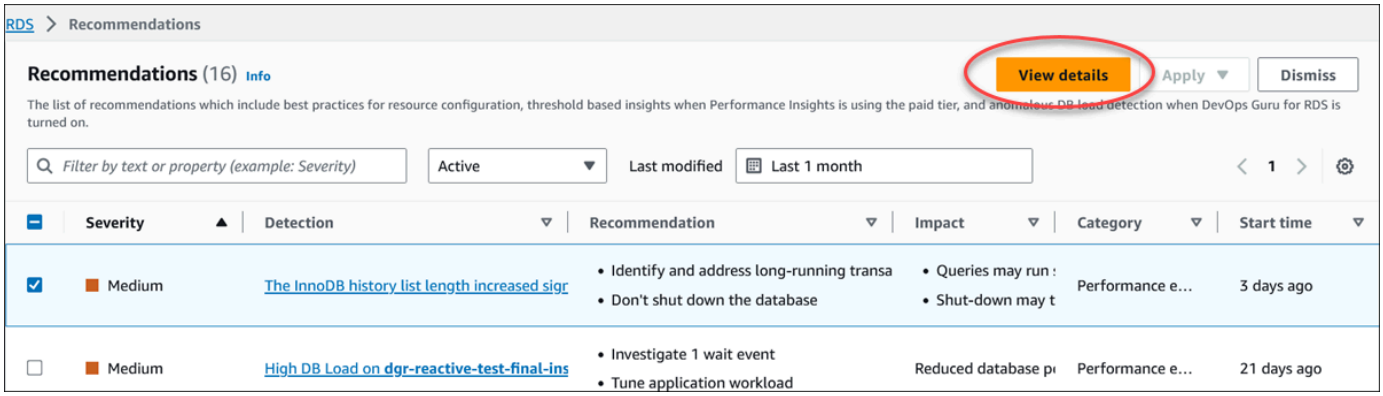


已設定時間期間的建議隨即顯示。

請注意，您可以將範圍設定為全部，以查看帳戶中資源的所有建議。

6. (選用) 選擇右側的偏好設定，以自訂要顯示的詳細資訊。您可以選擇頁面大小、換行文字行，以及允許或隱藏資料欄。

7. (選用) 選擇建議，然後選擇檢視詳細資訊。



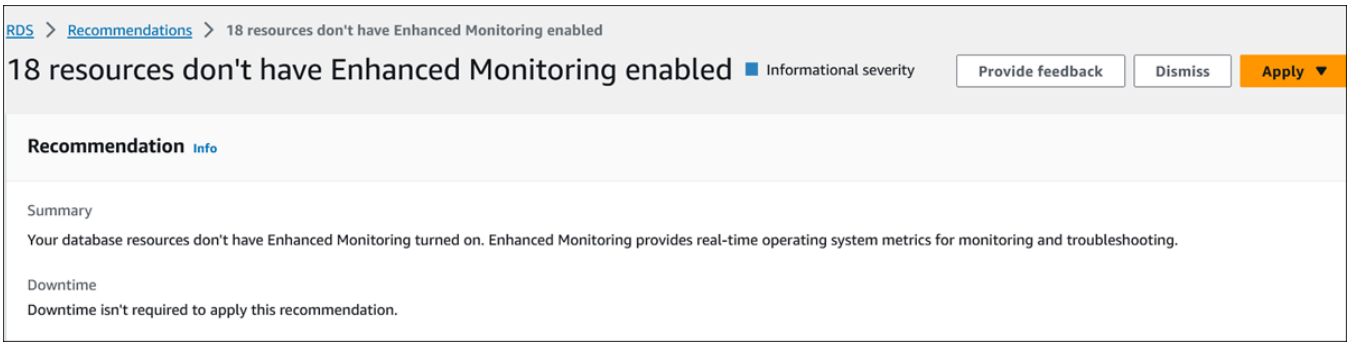
建議詳細資訊頁面隨即出現。標題提供偵測到問題的資源總數和嚴重性。

如需異常型被動建議詳細資訊頁面上元件的詳細資訊，請參閱《Amazon DevOps Guru 使用者指南》中的檢視被動異常。

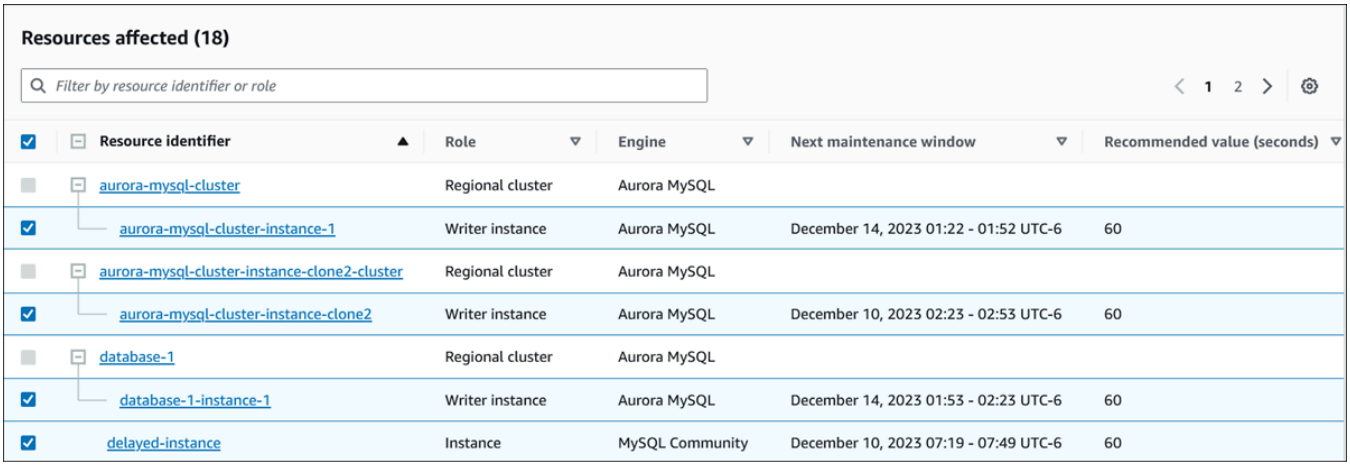
如需閾值型主動建議詳細資訊頁面上元件的詳細資訊，請參閱 檢視 Performance Insights 主動建議。

其他自動化建議會在建議詳細資訊頁面上顯示下列元件：

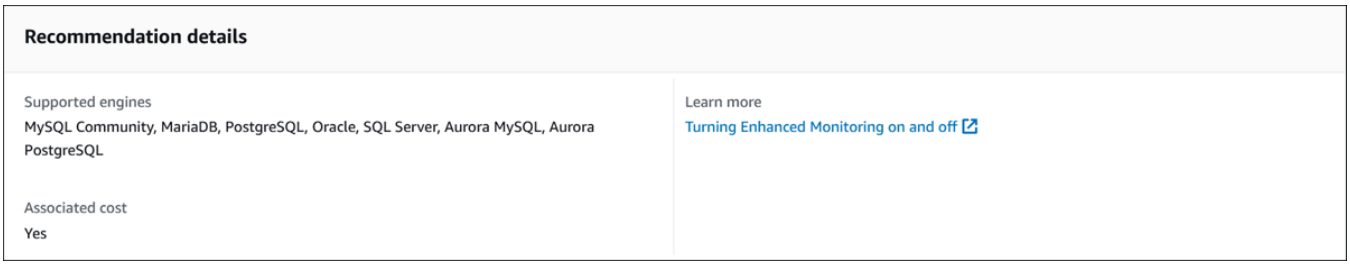
- 建議 – 建議的摘要，以及是否需要停機時間才能套用建議。



- 受影響的資源 – 受影響資源的詳細資訊。



- 建議詳細資訊 – 支援的引擎資訊、套用建議所需的任何相關聯成本，以及可進一步了解的文件連結。



CLI

若要檢視資料庫執行個體或資料庫叢集的 Amazon RDS 建議，請在 AWS CLI中使用下列命令。

```
aws rds describe-db-recommendations
```

RDS API

若要使用 Amazon RDS API 檢視 Amazon RDS 建議，請使用 DescribeDBRecommendations 操作。

套用 Amazon Aurora 建議

若要使用 Amazon RDS 主控台套用 Amazon Aurora 建議，請選取詳細資訊頁面中的組態型建議或受影響的資源。然後，選擇立即套用建議，或排程在下一個維護時段套用。資源可能需要重新啟動，變更才會生效。對於一些資料庫參數群組建議，您可能需要重新啟動資源。

閾值型主動或異常型被動建議不會有套用選項，可能需要額外的檢閱。

主控台

套用組態型建議

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，執行下列任何一項：

   - 選擇建議。

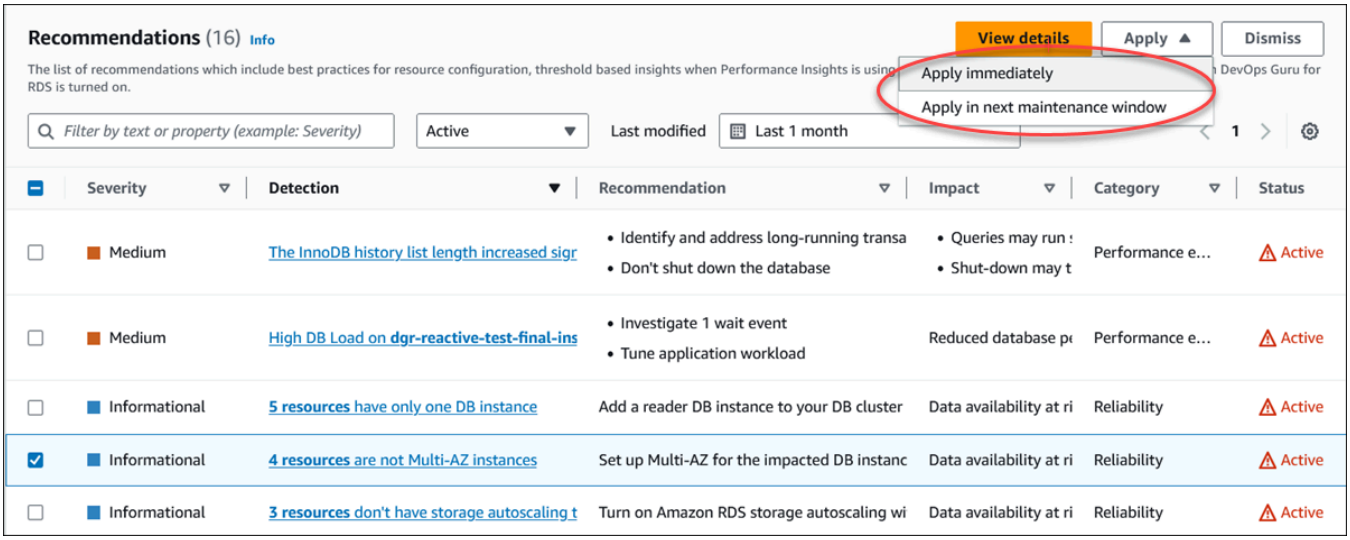
     建議頁面會顯示所有建議的清單。
   - 選擇資料庫，然後在資料庫頁面中選擇資源的建議。

     詳細資訊會顯示在所選取建議的建議索引標籤中。
   - 在建議頁面中針對作用中建議選擇偵測，或在資料庫頁面的建議索引標籤中選擇。

     建議詳細資訊頁面隨即出現。
3. 在建議詳細資訊頁面中選擇建議或一或多個受影響的資源，並執行下列任一操作：

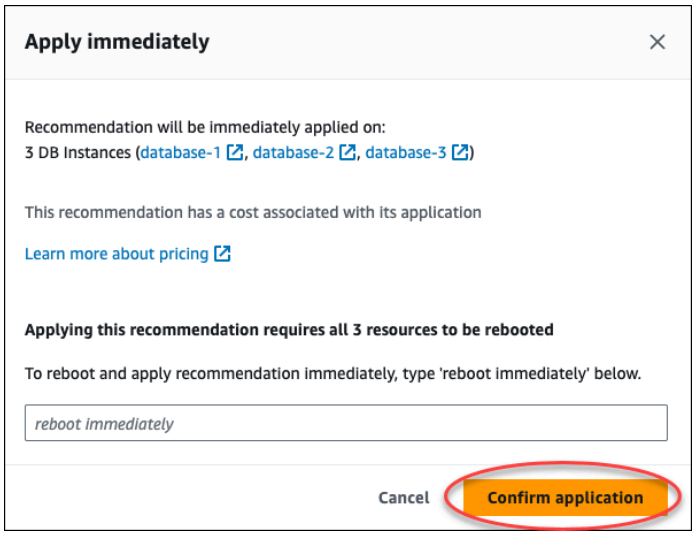
   - 選擇套用，然後選擇立即套用以立即套用建議。
   - 選擇套用，然後選擇在下一個維護時段套用以在下一個維護時段排程。

     選取的建議狀態會更新為待定，直到下一個維護時段為止。

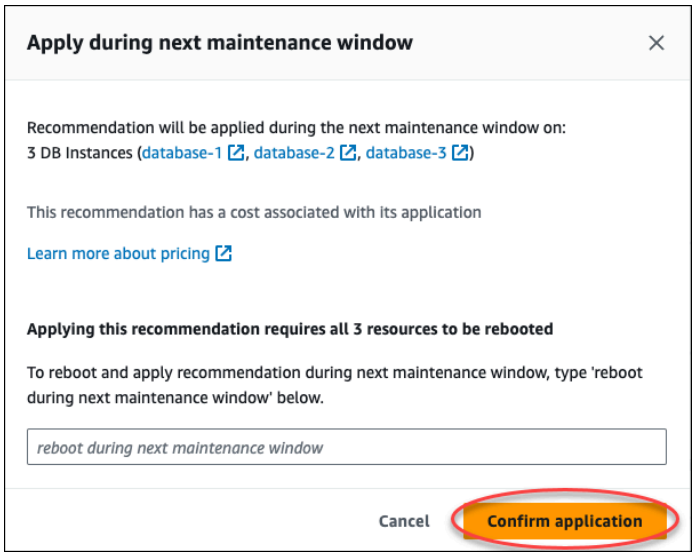


     確認視窗隨即顯示。
4. 選擇確認套用以套用建議。此視窗會確認資源是否需要自動或手動重新啟動，變更才會生效。

下列範例顯示立即套用建議的確認視窗。



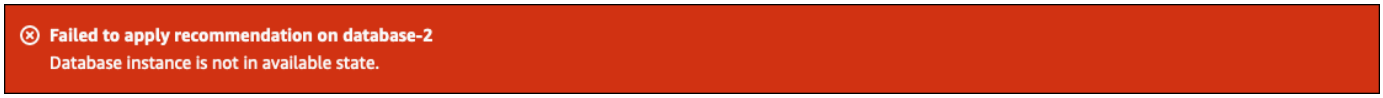
下列範例顯示要排程在下一個維護時段套用建議的確認視窗。



當套用的建議成功或失敗時，橫幅會顯示訊息。

下列範例顯示成功的訊息橫幅。

下列範例顯示失敗的訊息橫幅。



RDS API

使用 Amazon RDS API 套用組態型 Aurora 建議

1. 使用 DescribeDBRecommendations 操作。輸出中的 RecommendedActions 可以有一或多個建議的動作。
2. 針對步驟 1 的每個建議動作使用 RecommendedAction 物件。輸出包含 Operation 和 Parameters。

   下列範例顯示具有一個建議動作的輸出。

```
   "RecommendedActions": [
       {
           "ActionId": "0b19ed15-840f-463c-a200-b10af1b552e3",
           "Title": "Turn on auto backup", // localized
           "Description": "Turn on auto backup for my-mysql-instance-1", //
localized
           "Operation": "ModifyDbInstance",
           "Parameters": [
               {
                   "Key": "DbInstanceIdentifier",
                   "Value": "my-mysql-instance-1"
               },
               {
                   "Key": "BackupRetentionPeriod",
                   "Value": "7"
               }
           ],
           "ApplyModes": ["immediately", "next-maintenance-window"],
           "Status": "applied"
       },
       ... // several others
   ],
```

3. 針對步驟 2 中輸出的每個建議動作使用 operation，並且輸入 Parameters 值。
4. 步驟 2 中的操作成功後，請使用 ModifyDBRecommendation 操作來修改建議狀態。

關閉 Amazon Aurora 建議

您可以使用 主控台或 Amazon RDS API 關閉一或多個 AWS CLI Amazon RDSAmazon Aurora 建議。

主控台

關閉一或多個建議

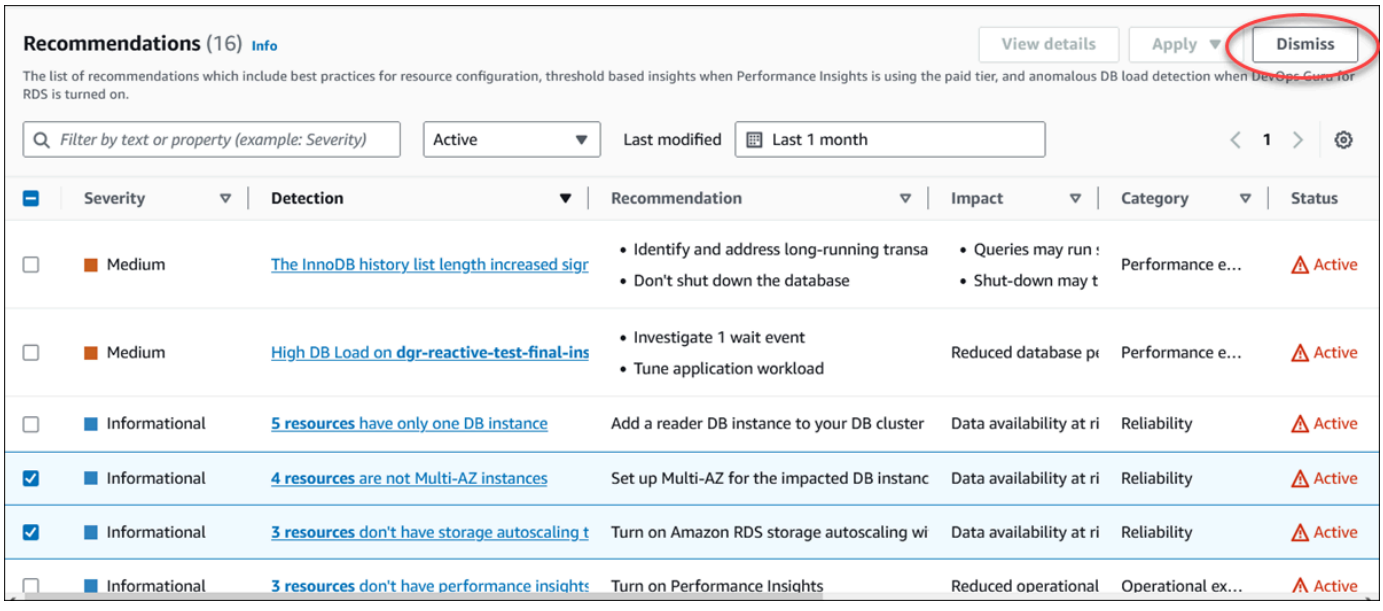
1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，執行下列任何一項：
   - 選擇建議。

     建議頁面會顯示所有建議的清單。
   - 選擇資料庫，然後在資料庫頁面中選擇資源的建議。

     詳細資訊會顯示在所選取建議的建議索引標籤中。
   - 在建議頁面中針對作用中建議選擇偵測，或在資料庫頁面的建議索引標籤中選擇。

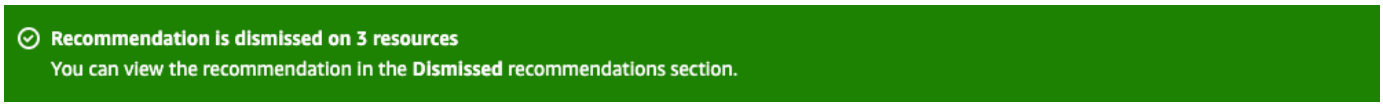
     建議詳細資訊頁面會顯示受影響的資源清單。
3. 在建議詳細資訊頁面中選擇一或多個建議，或選擇一或多個受影響的資源，然後選擇關閉。

   下列範例顯示建議頁面，其中已選取要關閉的多個作用中建議。



   當選取的一或多個建議關閉時，橫幅會顯示訊息。

下列範例顯示成功的訊息橫幅。



下列範例顯示失敗的訊息橫幅。

CLI

使用 關閉 Aurora 建議 AWS CLI

1. 執行 `aws rds describe-db-recommendations --filters "Name=status,Values=active"` 命令。

   輸出提供處於 `active` 狀態的建議清單。
2. 尋找您要從步驟 1 關閉之建議的 `recommendationId`。
3. 執行命令 `>aws rds modify-db-recommendation --status dismissed --recommendationId <ID>` 與步驟 2 的 `recommendationId` 以關閉建議。

RDS API

若要使用 Amazon RDS API 關閉 Aurora 建議，請使用 ModifyDBRecommendation 操作。

將已關閉的 Amazon Aurora 建議修改為作用中建議

您可以使用 主控台或 Amazon AWS CLI RDS API，將一或多個已關閉的 Amazon RDSAmazon Aurora 建議移至作用中建議。

主控台

將一或多個已關閉的建議移至作用中建議

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，執行下列任何一項：

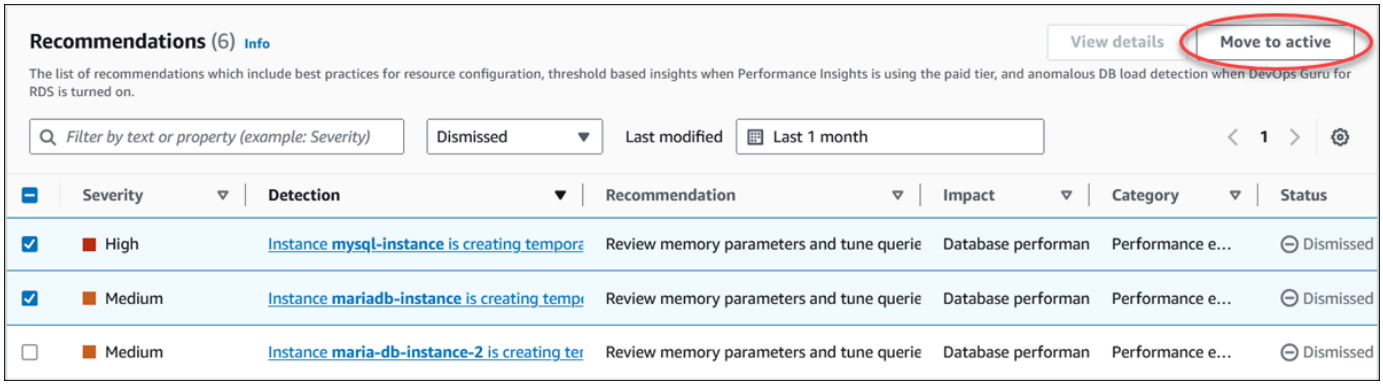
- 選擇建議。

  建議頁面會顯示依您帳戶中所有資源的嚴重性排序的建議清單。

- 選擇資料庫，然後在資料庫頁面中選擇資源的建議。

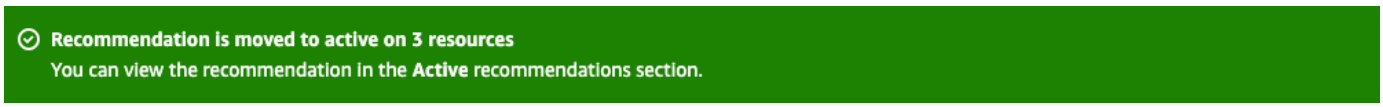
  建議索引標籤會顯示所選取資源的建議及其詳細資訊。

3. 從清單中選擇一或多個已關閉的建議，然後選擇移至作用中。



將所選取建議從已關閉移至作用中狀態時，橫幅會顯示成功或失敗訊息。

下列範例顯示成功的訊息橫幅。



下列範例顯示失敗的訊息橫幅。

CLI

使用 將已關閉的 Aurora建議變更為作用中建議 AWS CLI

1. 執行 `aws rds describe-db-recommendations --filters "Name=status,Values=dismissed"` 命令。

   輸出提供處於 `dismissed` 狀態的建議清單。

2. 尋找您要從步驟 1 變更狀態之建議的 `recommendationId`。

3. 執行命令 `>aws rds modify-db-recommendation --status active --recommendationId <ID>` 與步驟 2 的 `recommendationId` 以變更為作用中建議。

RDS API

若要使用 Amazon RDS API 將已關閉的 Aurora 建議變更為作用中建議，請使用 ModifyDBRecommendation 操作。

Amazon Aurora 建議參考

Amazon Aurora 會在資源建立或修改時，就資源產生建議。您可以在下表中找到 Amazon Aurora 建議的範例。

Type	描述	建議	需要停機時間	其他資訊
資源自動備份已關閉	您的資料庫執行個體不會開啟自動備份。建議自動備份，因為可啟用資料庫執行個體的時間點復原。	開啟保留期間最長為 14 天的自動備份。	是	備份與還原 Aurora 資料庫叢集的概觀 AWS 資料庫部落格上的揭密 Amazon RDS 備份儲存成本
需要引擎次要版本升級	您的資料庫資源未執行最新的次要資料庫引擎版本。最新的次要版本包含最新的安全性修正和其他改進。	升級至最新的引擎版本。	是	維持 為 Amazon Aurora 資料庫叢集
增強型監控已關閉	您的資料庫資源未開啟增強型監控。增強型監控針對監控及疑難排解，提供即時的作業系統指標。	開啟增強型監控。	否	使用增強型監控來監控作業系統指標
儲存加密已關閉	Amazon RDS 使用您在 AWS Key	為您的資料庫叢集開啟靜態資料加密。	是	Amazon Aurora 的安全性

Type	描述	建議	需要停機時間	其他資訊
	Management Service (AWS KMS) 中管理的金鑰，支援所有資料庫引擎的靜態加密。在具有 Amazon RDS 加密的作用中資料庫執行個體上，儲存體中存放的靜態資料會加密，類似於自動備份、僅供讀取複本和快照。 如果在建立 Aurora 資料庫叢集時未開啟加密，您必須將解密快照還原至加密的資料庫叢集。			
具有所有執行個體的資料庫叢集都位於相同的可用區域。	資料庫叢集目前位於單一可用區域。使用多個可用區域來改善可用性。	將資料庫執行個體新增至資料庫叢集中的多個可用區域。	否	Amazon Aurora 的高可用性
叢集中具有異質執行個體大小的資料庫執行個體	建議您在您的資料庫叢集中，對所有資料庫執行個體皆使用相同的資料庫執行個體類別和大小。	在您的資料庫叢集中，對所有資料庫執行個體皆使用相同的資料庫執行個體類別和大小。	是	以 Amazon Aurora 進行複寫

Type	描述	建議	需要停機時間	其他資訊
叢集中具有異質執行個體類別的資料庫執行個體	建議您在您的資料庫叢集中，對所有資料庫執行個體皆使用相同的資料庫執行個體類別和大小。	在您的資料庫叢集中，對所有資料庫執行個體皆使用相同的資料庫執行個體類別和大小。	是	以 Amazon Aurora 進行複寫
叢集中具有異質參數群組的資料庫執行個體	建議資料庫叢集中的所有資料庫執行個體都使用相同的資料庫參數群組。	將資料庫執行個體與資料庫叢集中寫入器執行個體相關聯的資料庫參數群組建立關聯。	否	Amazon Aurora 的參數群組
Amazon RDS 資料庫叢集有一個資料庫執行個體	再新增至少一個資料庫執行個體到您的資料庫叢集，以改善可用性和效能。	新增讀取器資料庫執行個體到您的資料庫叢集。	否	Amazon Aurora 的高可用性
Performance Insights 已關閉	Performance Insights 會監控您的資料庫執行個體負載，協助您分析並解決資料庫效能問題。建議您開啟 Performance Insights。	開啟績效詳情。	否	在 Amazon Aurora 上使用績效詳情監控資料庫負載
需要 RDS 資源主要版本更新	不支援具有資料庫引擎目前主要版本的資料庫。建議您升級至包含新功能和增強功能的最新主要版本。	升級至資料庫引擎的最新主要版本。	是	Amazon Aurora 更新 在 Amazon Aurora 中建立藍/綠部署

Type	描述	建議	需要停機時間	其他資訊
資料庫叢集磁碟區大小上限	較新的引擎版本可為您的資料庫叢集支援較大的儲存磁碟區。	建議您將資料庫叢集的引擎版本升級至最新版本，以受益於增加的儲存容量。	是	Amazon Aurora 大小限制
具有所有讀取器執行個體的資料庫叢集都位於相同的可用區域	在每個 AWS 區域內，可用區域 (AZ) 是彼此不同的位置，以便在中斷時提供隔離。建議您將資料庫叢集中的主要執行個體和讀取器執行個體分配到多個可用區域中，以提升資料庫叢集的可用性。您可以在建立叢集時使用 AWS 管理主控台、 AWS CLI 或 Amazon RDS API 來建立多可用區域叢集。您可以透過新增讀取器執行個體並指定不同的可用區域，將現有 Aurora 叢集修改為多可用區叢集。	在您的資料庫叢集之中，所有讀取執行個體都位於相同的可用區域。建議您將讀取器執行個體分散至多個可用區域。如此能增加可用性，並減少用戶端和資料庫之間的網路延遲，進而縮短回應時間。	否	Amazon Aurora 的高可用性

Type	描述	建議	需要停機時間	其他資訊
資料庫記憶體參數與預設值不同	資料庫執行個體的記憶體參數與預設值明顯不同。這些設定可能會影響效能並導致錯誤。 建議您在資料庫參數群組中，將資料庫執行個體的自訂記憶體參數重設為其預設值。	將記憶體參數重設為其預設值。	否	Amazon Aurora 的參數群組
查詢快取參數已開啟	當變更需要清除查詢快取時，您的資料庫執行個體會顯示為停滯。大部分工作負載並不會受益於查詢快取。MySQL 8.0 版和更新版本已移除查詢快取。建議您將 query_cache_type 參數設定為 0。	將您資料庫參數群組中的 query_cache_type 參數值設定為 0。	是	Amazon Aurora 的參數群組
log_output 參數設定為資料表	當 log_output 設定為 TABLE 時，使用的儲存體會比 log_output 設定為 FILE 時多。建議您將參數設定為 FILE，以避免達到儲存體大小限制。在 MySQL 8.4 和更新版本中，預設設定為 FILE。	將您資料庫參數群組中的 log_output 參數值設定為 FILE。	否	Aurora MySQL資料庫日誌檔案

Type	描述	建議	需要停機時間	其他資訊
autovacuum 參數已關閉	資料庫叢集的自動清空參數已關閉。關閉自動清空功能會增加資料表和索引膨脹並影響效能。 建議您在資料庫參數群組中開啟自動清空。	在您的資料庫叢集參數群組中開啟自動清空參數。	否	在 AWS 資料庫部落格上了解 Amazon RDS for PostgreSQL 環境中的自動清空
synchronous_commit 參數已關閉	當 synchronous_commit 參數關閉時，資料可能會在資料庫當機時遺失。資料庫的耐久性存在風險。 建議您開啟 synchronous_commit 參數。	在您的資料庫參數群組中開啟 synchronous_commit 參數。	是	Amazon Aurora PostgreSQL 參數：資料庫部落格上的複寫、安全性和記錄 AWS
track_counts 參數已關閉	當 track_counts 參數關閉時，資料庫就不會收集資料庫活動統計資料。自動清空功能需要這些統計資料才能正常運作。 建議您將 track_counts 參數設定為 1。	將 track_counts 參數設定為 1。	否	PostgreSQL 的執行階段統計資料

Type	描述	建議	需要停機時間	其他資訊
enable_indexonlyscan 參數已關閉	僅索引掃描計畫類型關閉時，查詢規劃器或最佳化工具皆無法使用該功能。 建議您將 enable_indexonlyscan 參數值設定為 1。	將 enable_indexonlyscan 參數值設定為 1。	否	PostgreSQL 的規劃器方法組態
enable_indexscan 參數已關閉	索引掃描計畫類型關閉時，查詢規劃器或最佳化工具皆無法使用該功能。 建議您將 enable_indexscan 值設定為 1。	將 enable_indexscan 參數值設定為 1。	否	PostgreSQL 的規劃器方法組態
innodb_flush_log_at_trx 參數已關閉	資料庫執行個體的 innodb_flush_log_at_trx 參數值不是安全的值。此參數控制提交操作至磁碟的持續性。 建議您將 innodb_flush_log_at_trx 參數設定為 1。	將 innodb_flush_log_at_trx 參數值設定為 1。	否	設定日誌緩衝區的排清頻率

Type	描述	建議	需要停機時間	其他資訊
innodb_stats_persistent 參數已關閉	您的資料庫執行個體未設定將 InnoDB 統計資料保留於磁碟。未儲存統計資料時，會在每次執行個體重新啟動和存取資料表時重新計算統計資料。這會導致查詢執行計畫的變化。您可以在資料表層級修改此全域參數的值。 建議您將 innodb_stats_persistent 參數值設定為 ON。	將 innodb_stats_persistent 參數值設定為 ON。	否	Amazon Aurora 的參數群組
innodb_open_files 參數過低	innodb_open_files 參數控制 InnoDB 一次可以開啟的檔案數量。InnoDB 會在 mysqld 執行時，開啟所有日誌和系統資料表空間檔案。 針對 InnoDB 一次能開啟的最大檔案數量，您的資料庫執行個體設定值很低。建議您將 innodb_open_files 參數設定為 65 的下限。	將 innodb_open_files 參數設定為最小值 65。	是	InnoDB 開啟 MySQL 的檔案

Type	描述	建議	需要停機時間	其他資訊
max_user_connections 參數過低	針對每個資料庫帳戶能同時連線的數量上限，您的資料庫執行個體設定值很低。 建議您將 max_user_connections 參數設定為大於 5 的數字。	將 max_user_connections 參數的值增加到大於 5 的數字。	是	設定 MySQL 的帳戶資源限制
僅供讀取複本以可寫入模式開啟	您的資料庫執行個體具有可寫入模式的僅供讀取複本，允許從用戶端進行更新。 建議您將 read_only 參數設定為 TrueIfReplica ，讓僅供讀取複本不處於可寫入模式。	將 read_only 參數值設定為 TrueIfReplica 。	否	Amazon Aurora 的參數群組

Type	描述	建議	需要停機時間	其他資訊
innodb_default_row_format 參數設定不安全	您的資料庫執行個體遇到已知問題：在低於 8.0.26 的 MySQL 版本中建立的資料表，且當索引超過 767 個位元組時，將 row_format 設定為 COMPACT 或 REDUNDANT 將無法存取且無法復原。 建議您將 innodb_default_row_format 參數值設定為 DYNAMIC。	將 innodb_default_row_format 參數值設定為 DYNAMIC。	否	MySQL 8.0.26 的變更
general_logging 參數已開啟	您的資料庫執行個體已開啟一般記錄功能。此設定在針對資料庫問題進行疑難排解時非常有用。不過，開啟一般記錄功能會增加 I/O 操作和配置儲存空間量，進而導致爭用和效能降低。 檢查您的要求是否有一般記錄用量。建議您將 general_logging 參數值設定為 0。	檢查您的要求是否有一般記錄用量。如果不是強制性的，建議您將 general_logging 參數值設定為 0。	否	Aurora MySQL 資料庫日誌概觀

Type	描述	建議	需要停機時間	其他資訊
資料庫叢集的讀取工作負載佈建不足	建議您將讀取器資料庫執行個體新增至資料庫叢集，其執行個體類別和大小與叢集中的寫入器資料庫執行個體相同。目前的組態有一個資料庫執行個體，資料庫負載持續偏高，主要是由讀取操作造成。將另一個資料庫執行個體新增至叢集，並將讀取工作負載導向資料庫叢集唯讀端點，以分配這些操作。	新增讀取器資料庫執行個體到叢集中。	否	將 Aurora 複本新增至資料庫叢集 管理 Aurora 資料庫叢集的效能和擴展 Amazon RDS 定價
RDS 執行個體的系統記憶體容量佈建不足	建議您調整查詢以使用較少的記憶體，或使用配置較高記憶體的資料庫執行個體類型。當執行個體記憶體不足時，資料庫效能會受到影響。	使用具有較高記憶體容量的資料庫執行個體	是	資料庫 AWS 部落格上的垂直和水平擴展 Amazon RDS 執行個體 Amazon RDS 執行個體類型 Amazon RDS 定價

Type	描述	建議	需要停機時間	其他資訊
RDS 執行個體的系統 CPU 容量佈建不足	建議您調整查詢以使用較少的 CPU，或修改資料庫執行個體以使用配置較高 vCPU 的資料庫執行個體類別。當資料庫執行個體上執行的 CPU 偏低時，資料庫效能可能會降低。	使用具有較高 CPU 容量的資料庫執行個體	是	資料庫 AWS 部落格上的垂直和水平擴展 Amazon RDS 執行個體 Amazon RDS 執行個體類型 Amazon RDS 定價
RDS 資源未正確使用連線集區	建議您啟用 Amazon RDS Proxy，以有效率地將現有資料庫連線集合起來並共用。如果您已經為資料庫使用代理，請正確設定它，以改善跨多個資料庫執行個體的連線集區和負載平衡。RDS Proxy 可協助降低連線耗盡和停機的風險，同時改善可用性和可擴展性。	啟用 RDS Proxy 或修改現有的 Proxy 組態	否	資料庫 AWS 部落格上的垂直和水平擴展 Amazon RDS 執行個體 Amazon RDS Proxy for Aurora Amazon RDS Proxy 定價

在 Amazon RDS 主控台中檢視指標

Amazon RDS 會與 Amazon CloudWatch 整合，以在 RDS 主控台中顯示各種 Aurora 資料庫叢集指標。部分指標適用於叢集層級，而其他指標則適用於執行個體層級。如需有關執行個體層級和叢集層級指標的說明，請參閱 Amazon Aurora 的指標參考。

對於 Aurora 資料庫叢集，會監控以下類別的指標：

- CloudWatch - 顯示您可以於 RDS 主控台中存取的 Aurora 的 Amazon CloudWatch 指標。您也可以在 CloudWatch 主控台中存取這些指標。每一個指標都包括一個圖形，其中顯示在特定時間範圍內所監控的指標。如需 CloudWatch 指標的清單，請參閱 Amazon Aurora 的 Amazon CloudWatch 指標。
- Enhanced monitoring (增強型監控)：在 Aurora 資料庫叢集開啟增強型監控後，顯示作業系統指標的摘要。RDS 可將增強型監控的指標傳送至您的 Amazon CloudWatch Logs 帳戶中。每一個作業系統指標都包括一個圖形，其中顯示在特定時間範圍內所監控的指標。如需概觀，請參閱 使用增強型監控來監控作業系統指標。如需增強型監控指標的清單，請參閱 增強型監控中的作業系統指標。
- OS Process list (作業系統程序清單)：顯示您資料庫叢集中執行的每個程序的詳細資訊。
- Performance Insights：開啟您 Aurora 資料庫叢集中資料庫執行個體的 Amazon RDS Performance Insights 儀表板。叢集層級不支援 Performance Insights。如需 Performance Insights 概觀，請參閱 在 Amazon Aurora 上使用績效詳情監控資料庫負載。如需 Performance Insights 指標的清單，請參閱 Amazon RDS Performance Insights 的 Amazon CloudWatch 指標。

Amazon RDS 現在會在績效詳情儀表板中提供績效詳情和 CloudWatch 指標的合併檢視。必須為您的資料庫叢集開啟績效詳情，才能使用此檢視。您可以在監控索引標籤或在導覽窗格的績效詳情中選擇新的監控檢視。若要檢視選擇此檢視的指示，請參閱 使用 Performance Insights 儀表板檢視組合指標。

使用 Performance Insights 儀表板檢視組合指標

> **Important**
>
> AWS 已宣布績效詳情的end-of-life：2026 年 6 月 30 日。在此日期之後，Amazon RDS 將不再支援 Performance Insights 主控台體驗、彈性保留期間 (1-24 個月) 及其相關定價。Performance Insights API 將繼續存在，不會有任何定價變更。Performance Insights API 的成本會顯示在您的 AWS 帳單中，其中包含 CloudWatch Database Insights 的成本。
> 我們建議您在 2026 年 6 月 30 日之前，使用已付費的績效詳情方案，將任何資料庫叢集升級至資料庫詳情的進階模式。如需有關升級至 Database Insights 進階模式的資訊，請參閱 開啟適用於 Amazon Aurora 的 Database Insights 的進階模式。
> 如果您不採取任何動作，則使用 Performance Insights 的資料庫叢集依預設會使用 Database Insights 的標準模式。使用 Database Insights 的標準模式時，您可能會失去超過 7 天的效能資料歷史記錄存取權，而且可能無法在 Amazon RDS 主控台中使用執行計畫和隨需分析功能。2026 年 6 月 30 日之後，只有進階模式的 Database Insights 將支援執行計畫和隨需分析。
> 透過 CloudWatch Database Insights，您可以監控資料庫機群的資料庫負載，以及大規模分析效能和進行疑難排解。如需 Database Insights 的詳細資訊，請參閱 使用 CloudWatch Database Insights 監控 Amazon Aurora 資料庫。如需定價資訊，請參閱 Amazon CloudWatch 定價。

Amazon RDS 會在 Performance Insights 儀表板中為您的資料庫執行個體提供 Performance Insights 和 CloudWatch 指標的合併檢視。您可以使用預先設定的儀表板或建立自訂儀表板。預先設定的儀表板提供最常用的指標，協助診斷資料庫引擎的效能問題。或者，您可以為資料庫引擎建立自訂儀表板，其中包含符合您分析需求的指標。然後，針對您 AWS 帳戶中該資料庫引擎類型的所有資料庫執行個體使用此儀表板。

您可以在監控索引標籤或在導覽窗格的 Performance Insights 中選擇監控檢視。

您必須為資料庫叢集開啟績效詳情，才能在「績效詳情」儀表板中檢視組合指標。如需開啟績效詳情的詳細資訊，請參閱 開啟和關閉 Aurora 的 Performance Insights。

在下列各節中，您可以了解顯示 Performance Insights 和 CloudWatch 指標。

主題

- 在監控索引標籤中選擇新的監控檢視
- 選擇具有 Performance Insights 頁面的新監控檢視

- 建立具有 Performance Insights 的自訂儀表板
- 選擇具有 Performance Insights 的預先設定儀表板

在監控索引標籤中選擇新的監控檢視

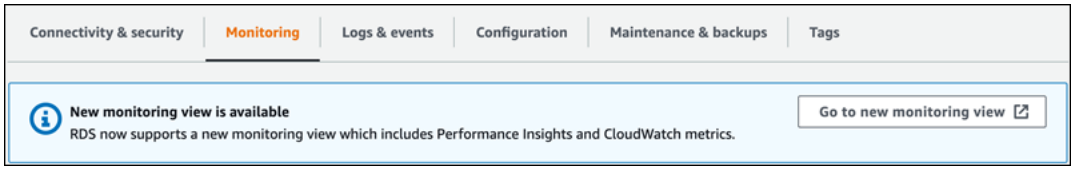
> **Important**
>
> AWS 已宣布績效詳情的end-of-life：2026 年 6 月 30 日。在此日期之後，Amazon RDS 將不再支援 Performance Insights 主控台體驗、彈性保留期間 (1-24 個月) 及其相關定價。Performance Insights API 將繼續存在，不會有任何定價變更。Performance Insights API 的成本會顯示在您的 AWS 帳單中，其中包含 CloudWatch Database Insights 的成本。
> 我們建議您在 2026 年 6 月 30 日之前，使用已支付的效能詳情方案，將任何資料庫叢集升級至資料庫詳情的進階模式。如需有關升級至 Database Insights 進階模式的資訊，請參閱 開啟適用於 Amazon Aurora 的 Database Insights 的進階模式。
> 如果您不採取任何動作，則使用 Performance Insights 的資料庫叢集依預設會使用 Database Insights 的標準模式。使用 Database Insights 的標準模式時，您可能會失去超過 7 天的效能資料歷史記錄存取權，而且可能無法在 Amazon RDS 主控台中使用執行計畫和隨需分析功能。2026 年 6 月 30 日之後，只有進階模式的 Database Insights 將支援執行計畫和隨需分析。
> 透過 CloudWatch Database Insights，您可以監控資料庫機群的資料庫負載，以及大規模分析效能和進行疑難排解。如需 Database Insights 的詳細資訊，請參閱 使用 CloudWatch Database Insights 監控 Amazon Aurora 資料庫。如需定價資訊，請參閱 Amazon CloudWatch 定價。

從 Amazon RDS 主控台，您可以選擇新的監控檢視，以檢視資料庫執行個體的 Performance Insights 和 CloudWatch 指標。

若要在監控索引標籤中選擇新的監控檢視

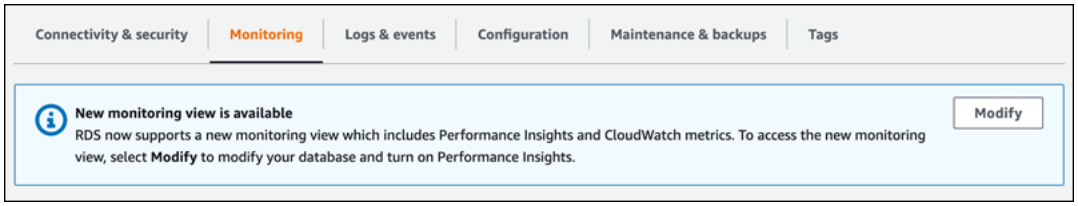
1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在左側的導覽窗格中，選擇資料庫。
3. 選擇您想要監控的 Aurora 資料庫叢集。
4. 向下捲動並選擇監控。

   橫幅隨即出現，其中包含了選擇新監控檢視的選項。以下範例顯示了選擇新監控檢視的橫幅。



5. 選擇移至新的監控檢視以開啟績效詳情儀表板，其中包含資料庫叢集的績效詳情和 CloudWatch 指標。
6. (選用) 如果資料庫執行個體的績效詳情已關閉，則會出現橫幅，其中包含可修改資料庫執行個體並開啟績效詳情的選項。

   下列範例顯示橫幅，以在監控索引標籤中修改資料庫執行個體。



   選擇修改以修改您的資料庫執行個體，並開啟績效詳情。如需開啟績效詳情的詳細資訊，請參閱 開啟和關閉 Aurora 的 Performance Insights

選擇具有 Performance Insights 頁面的新監控檢視

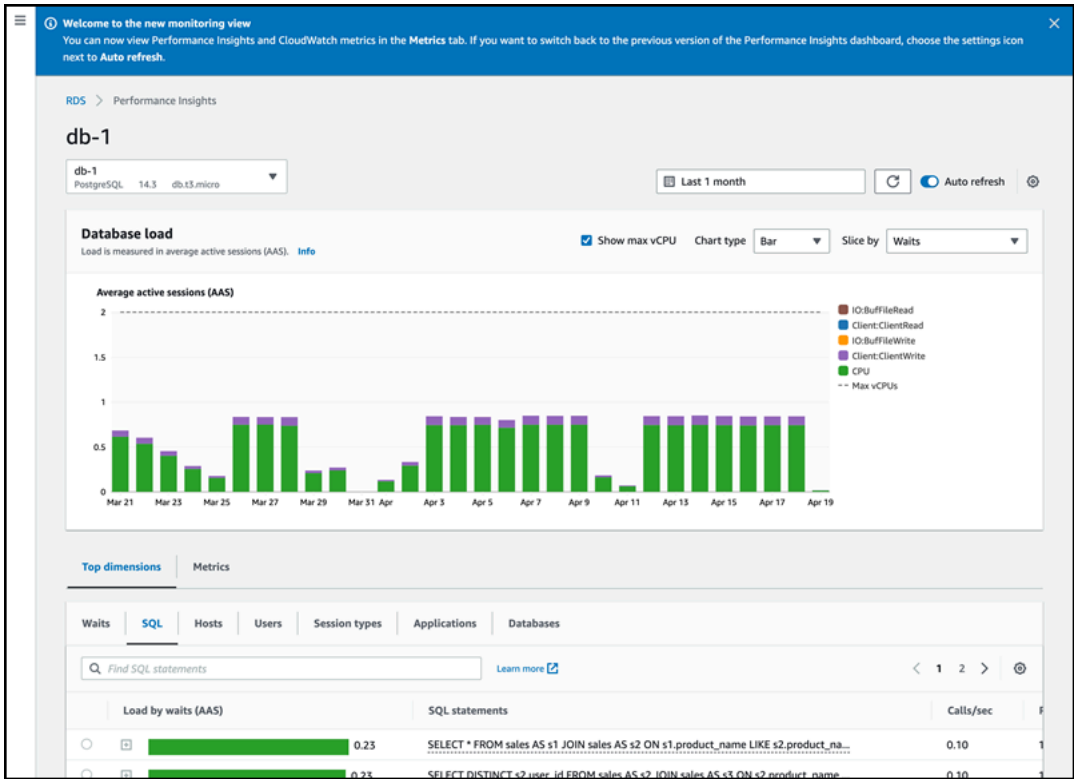
> **Important**
>
> AWS 已宣布績效詳情的end-of-life：2026 年 6 月 30 日。在此日期之後，Amazon RDS 將不再支援 Performance Insights 主控台體驗、彈性保留期間 (1-24 個月) 及其相關定價。Performance Insights API 將繼續存在，不會有任何定價變更。Performance Insights API 的成本會顯示在您的 AWS 帳單中，其中包含 CloudWatch Database Insights 的成本。
> 我們建議您在 2026 年 6 月 30 日之前，使用已付費的績效詳情方案，將任何資料庫叢集升級至資料庫詳情的進階模式。如需有關升級至 Database Insights 進階模式的資訊，請參閱 開啟適用於 Amazon Aurora 的 Database Insights 的進階模式。
> 如果您不採取任何動作，則使用 Performance Insights 的資料庫叢集依預設會使用 Database Insights 的標準模式。使用 Database Insights 的標準模式時，您可能會失去超過 7 天的效能資料歷史記錄存取權，而且可能無法在 Amazon RDS 主控台中使用執行計劃和隨需分析功能。2026 年 6 月 30 日之後，只有進階模式的 Database Insights 將支援執行計畫和隨需分析。
> 透過 CloudWatch Database Insights，您可以監控資料庫機群的資料庫負載，以及大規模分析效能和進行疑難排解。如需 Database Insights 的詳細資訊，請參閱 使用 CloudWatch

Database Insights 監控 Amazon Aurora 資料庫。如需定價資訊，請參閱 Amazon CloudWatch 定價。

從 Amazon RDS 主控台，您可以選擇新的監控檢視，以檢視資料庫執行個體的 Performance Insights 和 CloudWatch 指標。

若要在導覽窗格中選擇具有 Performance Insights 的新監控檢視

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在左側導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體以檢視 Performance Insights 儀表板，其中同時顯示資料庫執行個體的 Performance Insights 和 CloudWatch 指標。



建立具有 Performance Insights 的自訂儀表板

Important

AWS 已宣布績效詳情的end-of-life：2026 年 6 月 30 日。在此日期之後，Amazon RDS 將不再支援 Performance Insights 主控台體驗、彈性保留期間 (1-24 個月) 及其相關定價。Performance Insights API 將繼續存在，不會有任何定價變更。Performance Insights API 的成本會顯示在您的 AWS 帳單中，其中包含 CloudWatch Database Insights 的成本。
我們建議您在 2026 年 6 月 30 日之前，使用已付費的績效詳情方案，將任何資料庫叢集升級至資料庫詳情的進階模式。如需有關升級至 Database Insights 進階模式的資訊，請參閱 開啟適用於 Amazon Aurora 的 Database Insights 的進階模式。
如果您不採取任何動作，則使用 Performance Insights 的資料庫叢集依預設會使用 Database Insights 的標準模式。使用 Database Insights 的標準模式時，您可能會失去超過 7 天的效能資料歷史記錄存取權，而且可能無法在 Amazon RDS 主控台中使用執行計劃和隨需分析功能。2026 年 6 月 30 日之後，只有進階模式的 Database Insights 將支援執行計畫和隨需分析。
透過 CloudWatch Database Insights，您可以監控資料庫機群的資料庫負載，以及大規模分析效能和進行疑難排解。如需 Database Insights 的詳細資訊，請參閱 使用 CloudWatch Database Insights 監控 Amazon Aurora 資料庫。如需定價資訊，請參閱 Amazon CloudWatch 定價。

在新的監控檢視中，您可以建立自訂儀表板，其中具有您符合分析需求所需的指標。

您可以選取資料庫執行個體的績效詳情和 CloudWatch 指標，以建立自訂儀表板。您可以將此自訂儀表板用於 AWS 帳戶中相同資料庫引擎類型的其他資料庫執行個體。

Note

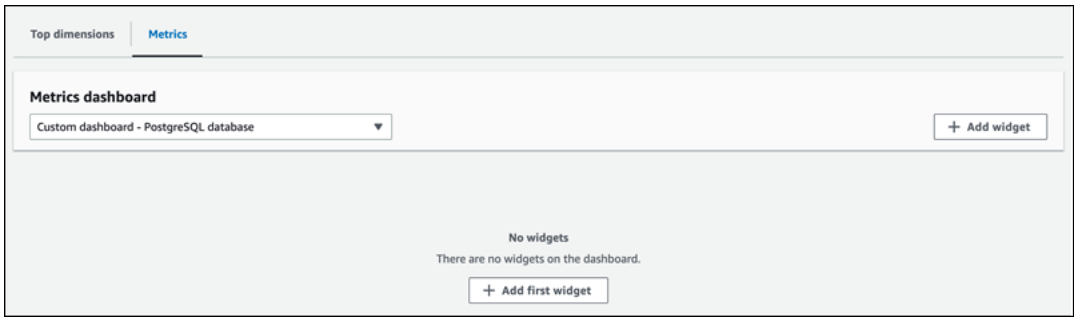
自訂的儀表板最多支援 50 個指標。

使用 Widget 設定功能表來編輯或刪除儀表板，以及移動 Widget 視窗或調整其大小。

若要在導覽窗格中建立具有 Performance Insights 的自訂儀表板

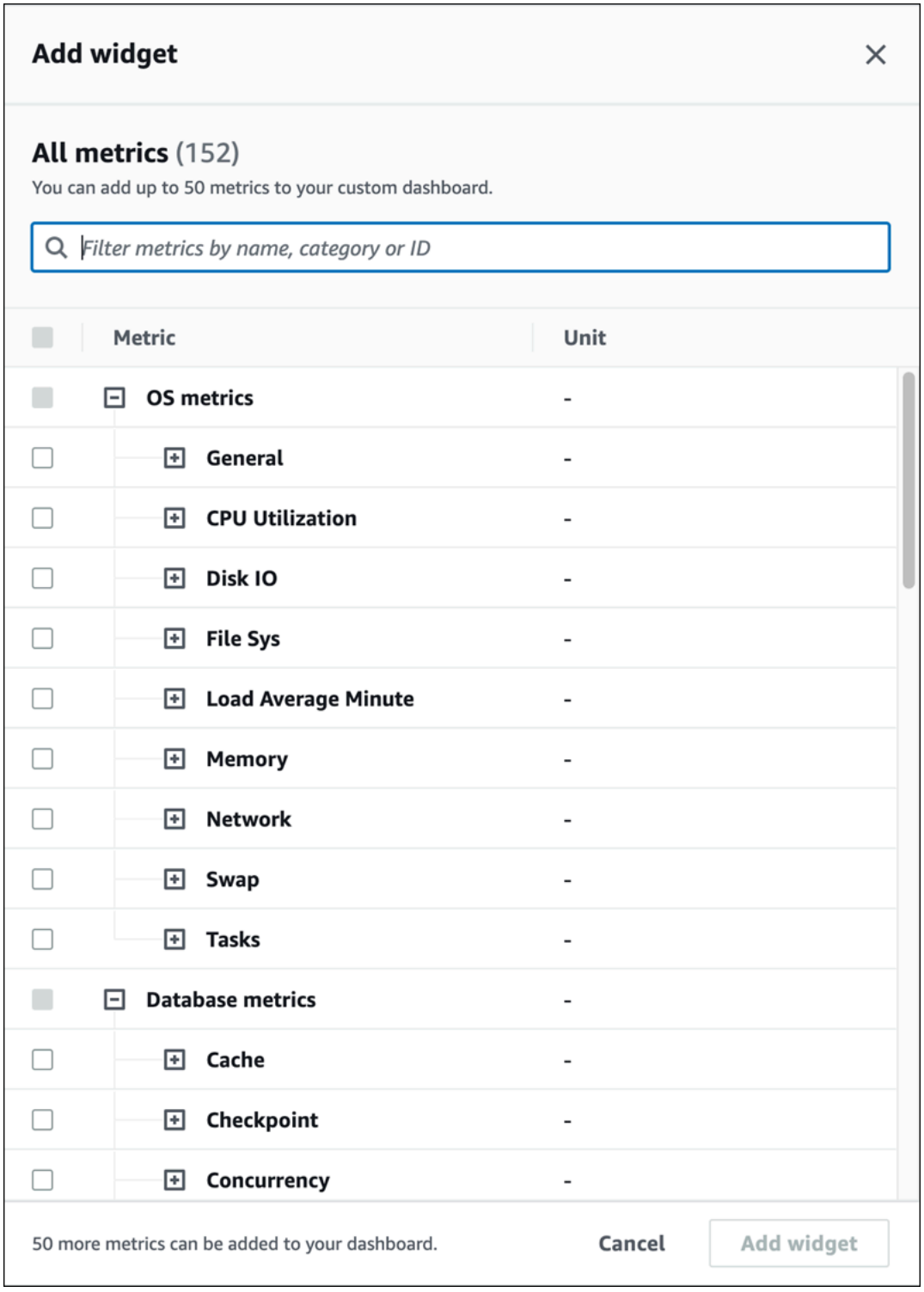
1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。

2. 在左側導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。
4. 向下捲動至視窗中的指標索引標籤。
5. 從下拉式清單中選取自訂儀表板。下列範例說明如何建立自訂儀表板。



6. 選擇新增 Widget 以開啟新增 Widget 視窗。您可以在視窗中開啟並檢視可用的作業系統 (OS) 指標、資料庫指標和 CloudWatch 指標。

   下列範例顯示了具有指標的新增 Widget 視窗。



7. 選取要在儀表板中檢視的指標，然後選擇新增 Widget。您可以使用搜尋欄位來尋找特定指標。

   選取的指標會出現在您的儀表板上。

8. (選用) 如果您想要修改或刪除儀表板，請選擇 Widget 右上角的設定圖示，然後在功能表中選取下列其中一個動作。

- 編輯 - 修改視窗中的指標清單。在您選取儀表板的指標之後，選擇更新 Widget。
- 刪除 - 刪除 Widget。在確認視窗中選擇刪除。

選擇具有 Performance Insights 的預先設定儀表板

Important

AWS 已宣布績效詳情的end-of-life：2026 年 6 月 30 日。在此日期之後，Amazon RDS 將不再支援 Performance Insights 主控台體驗、彈性保留期間 (1-24 個月) 及其相關定價。Performance Insights API 將繼續存在，不會有任何定價變更。Performance Insights API 的成本會顯示在您的 AWS 帳單中，其中包含 CloudWatch Database Insights 的成本。
我們建議您在 2026 年 6 月 30 日之前，使用已付費的績效詳情方案，將任何資料庫叢集升級至資料庫詳情的進階模式。如需有關升級至 Database Insights 進階模式的資訊，請參閱 開啟適用於 Amazon Aurora 的 Database Insights 的進階模式。
如果您不採取任何動作，則使用 Performance Insights 的資料庫叢集依預設會使用 Database Insights 的標準模式。使用 Database Insights 的標準模式時，您可能會失去超過 7 天的效能資料歷史記錄存取權，而且可能無法在 Amazon RDS 主控台中使用執行計劃和隨需分析功能。2026 年 6 月 30 日之後，只有進階模式的 Database Insights 將支援執行計畫和隨需分析。
透過 CloudWatch Database Insights，您可以監控資料庫機群的資料庫負載，以及大規模分析效能和進行疑難排解。如需 Database Insights 的詳細資訊，請參閱 使用 CloudWatch Database Insights 監控 Amazon Aurora 資料庫。如需定價資訊，請參閱 Amazon CloudWatch 定價。

您可以使用預先設定的儀表板檢視最常用的指標。此儀表板可協助診斷資料庫引擎的效能問題，並將平均復原時間從數小時縮短為數分鐘。

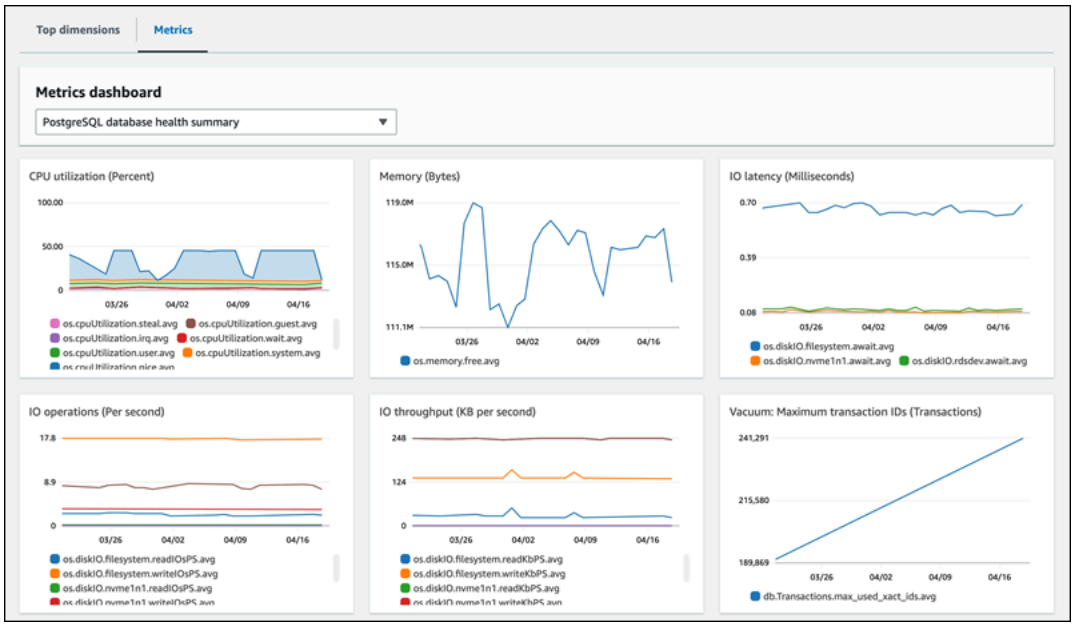
Note

無法編輯此儀表板。

若要在導覽窗格中選擇具有 Performance Insights 的預先設定儀表板

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在左側導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。
4. 向下捲動至視窗中的指標索引標籤
5. 從下拉式清單中選取預先設定的儀表板。

   您可以在儀表板中檢視資料庫執行個體的指標。下列範例顯示預先設定的指標儀表板。



使用 Amazon CloudWatch 監控 Amazon Aurora 指標

Amazon CloudWatch 是指標儲存體。儲存體會收集並處理來自 Amazon Aurora 的原始資料，進而將這些資料轉換為便於讀取且幾近即時的指標。如需傳送至 CloudWatch 的 Amazon Aurora 指標完整清單，請參閱 Amazon Aurora 的指標參考。

若要大規模分析資料庫的效能以及進行疑難排解，請使用 CloudWatch Database Insights。

主題

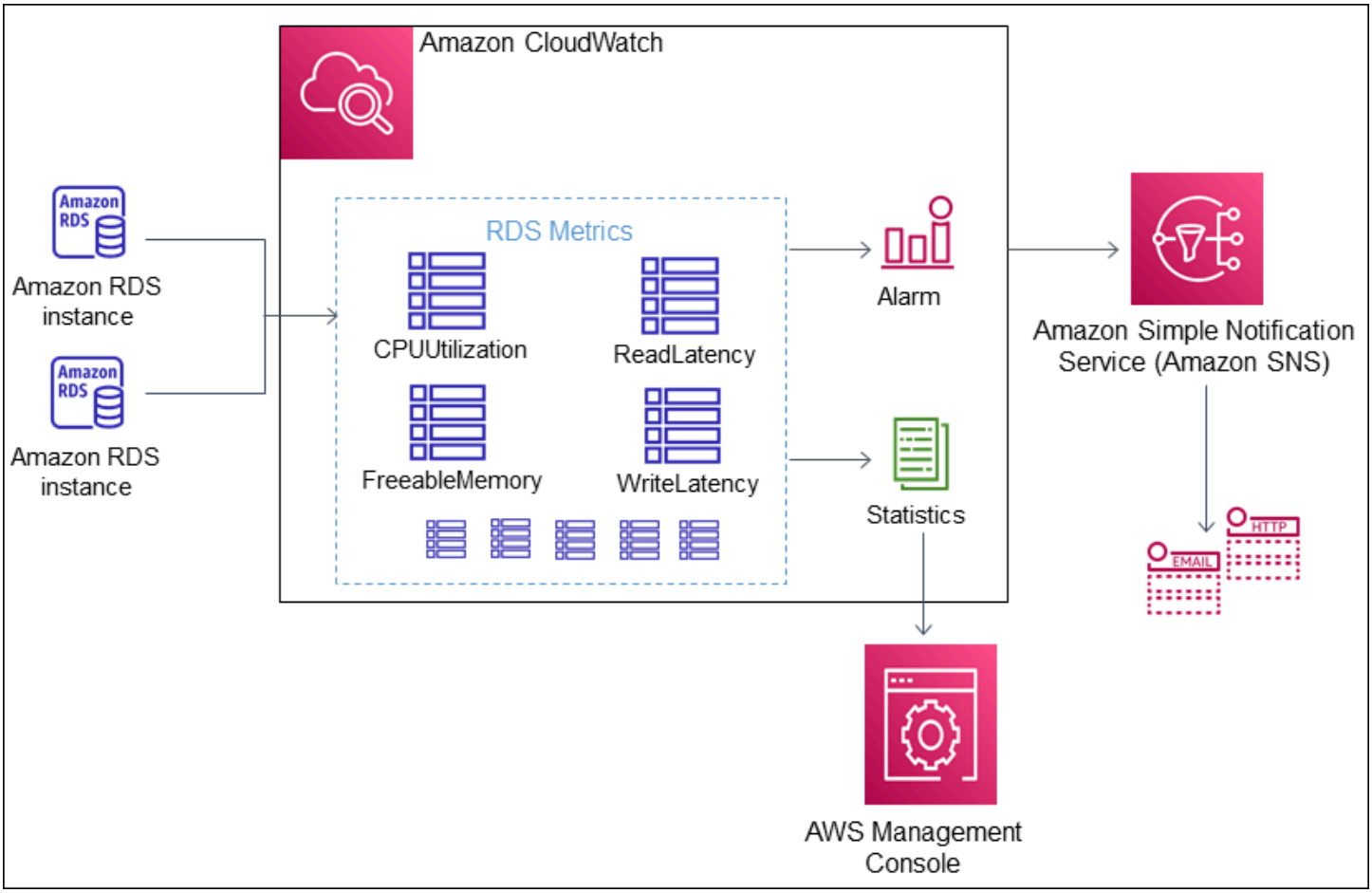
- Amazon Aurora 和 Amazon CloudWatch 的概觀
- 在 CloudWatch 主控台和 中檢視資料庫叢集指標 AWS CLI
- 將 Performance Insights 指標匯出至 CloudWatch
- 建立 CloudWatch 警示來監控 Amazon Aurora

Amazon Aurora 和 Amazon CloudWatch 的概觀

根據預設，Amazon Aurora 每隔 1 分鐘會自動將指標資料傳送至 CloudWatch。例如，`CPUUtilization` 指標記錄一段時間內資料庫執行個體的 CPU 利用率百分比。含少於 60 秒期間 (1 分鐘) 的資料點可供使用 15 天。這表示您可以存取歷史資訊，並查看 Web 應用程式或服務的執行效能。

您現在可以將 Performance Insights 指標儀表板從 Amazon RDS 匯出到 Amazon CloudWatch。您可以將預先設定或自訂的指標儀表板匯出為新的儀表板，或將其新增至現有的 CloudWatch 儀表板。匯出的儀表板可在 CloudWatch 主控台中檢視。如需如何將 Performance Insights 指標儀表板匯出至 CloudWatch 的詳細資訊，請參閱將 Performance Insights 指標匯出至 CloudWatch。

如下圖所示，您可為 CloudWatch 指標設定警示。例如，您可能會建立一個警示，當執行個體的 CPU 利用率超過 70% 時發出訊號。您可以設定 Amazon Simple Notification Service 在超過閾值時傳送電子郵件給您。



Amazon RDS 會向 Amazon CloudWatch 發佈下列類型的指標：

- 在叢集和執行個體兩個層級的 Aurora 指標

  如需關於這些指標的表格，請參閱 Amazon Aurora 的 Amazon CloudWatch 指標。
- 績效詳情指標

  如需關於這些指標的表格，請參閱 Amazon RDS Performance Insights 的 Amazon CloudWatch 指標 和 Performance Insights 計數器指標。
- 增強型監控指標 (發佈至 Amazon CloudWatch Logs)

  如需關於這些指標的表格，請參閱 增強型監控中的作業系統指標。
- 您 中 Amazon RDS 服務配額的使用指標 AWS 帳戶

  如需關於這些指標的表格，請參閱 Amazon Aurora 的 Amazon CloudWatch 用量指標。如需 Amazon RDS 配額的詳細資訊，請參閱 Amazon Aurora 的配額和條件限制。

如需 CloudWatch 的詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的什麼是 Amazon CloudWatch？。如需有關 CloudWatch 指標保留的詳細資訊，請參閱指標保留。

在 CloudWatch 主控台和 中檢視資料庫叢集指標 AWS CLI

接下來，您可以找到有關如何使用 CloudWatch 檢視資料庫執行個體指標的詳細資料。如需使用 CloudWatch Logs 即時監控資料庫執行個體之作業系統指標的相關資訊，請參閱使用增強型監控來監控作業系統指標。

當您使用 Amazon Aurora 資源時，Amazon Aurora 每分鐘都會將指標和維度傳送至 Amazon CloudWatch。

您現在可以將 Performance Insights 指標儀表板從 Amazon RDS 匯出到 Amazon CloudWatch，並在 CloudWatch 主控台中檢視這些指標。如需如何將 Performance Insights 指標儀表板匯出至 CloudWatch 的詳細資訊，請參閱 將 Performance Insights 指標匯出至 CloudWatch。

請使用下列程序，在 CloudWatch 主控台和 CLI 中檢視 Amazon Aurora 的指標。

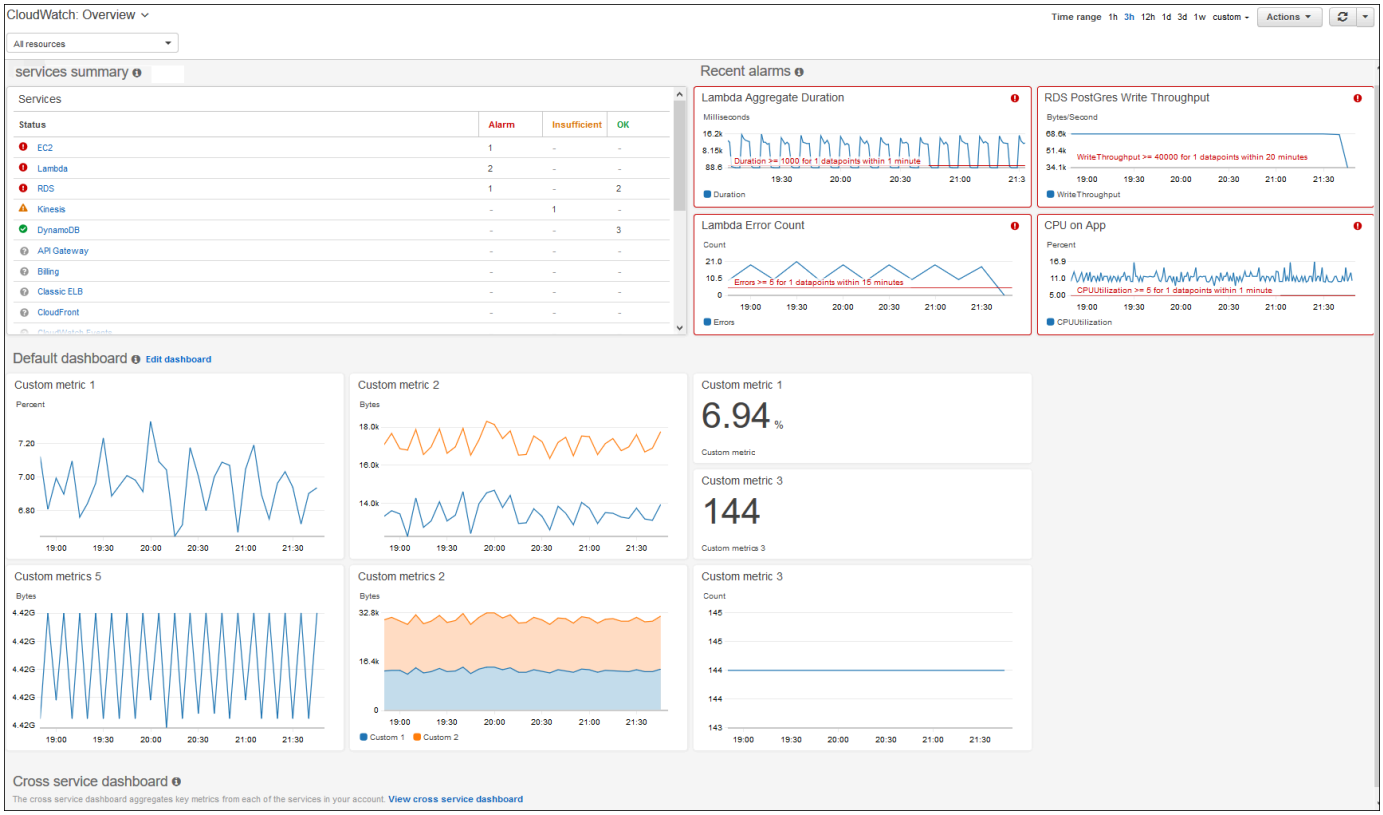
主控台

使用 Amazon CloudWatch 主控台檢視指標

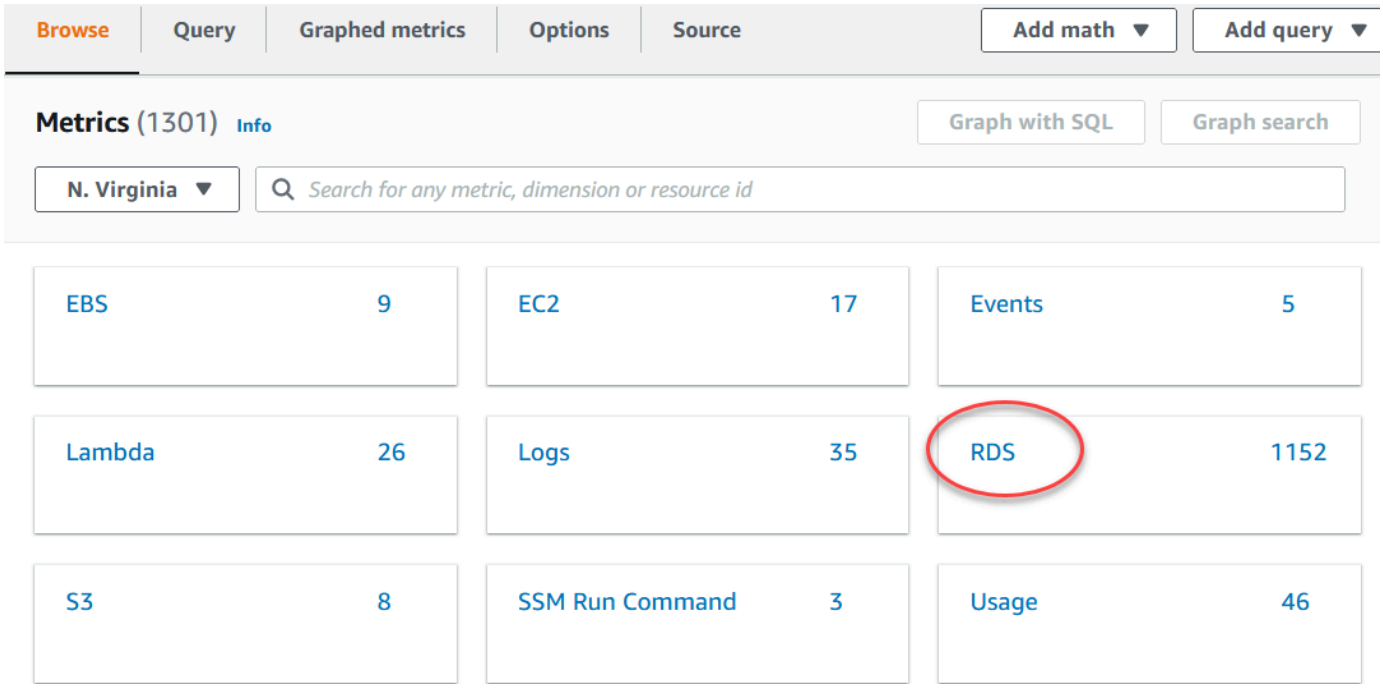
指標會先依服務命名空間分組，再依各命名空間內不同的維度組合分類。

1. 在 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。

CloudWatch 概觀首頁隨即顯示。

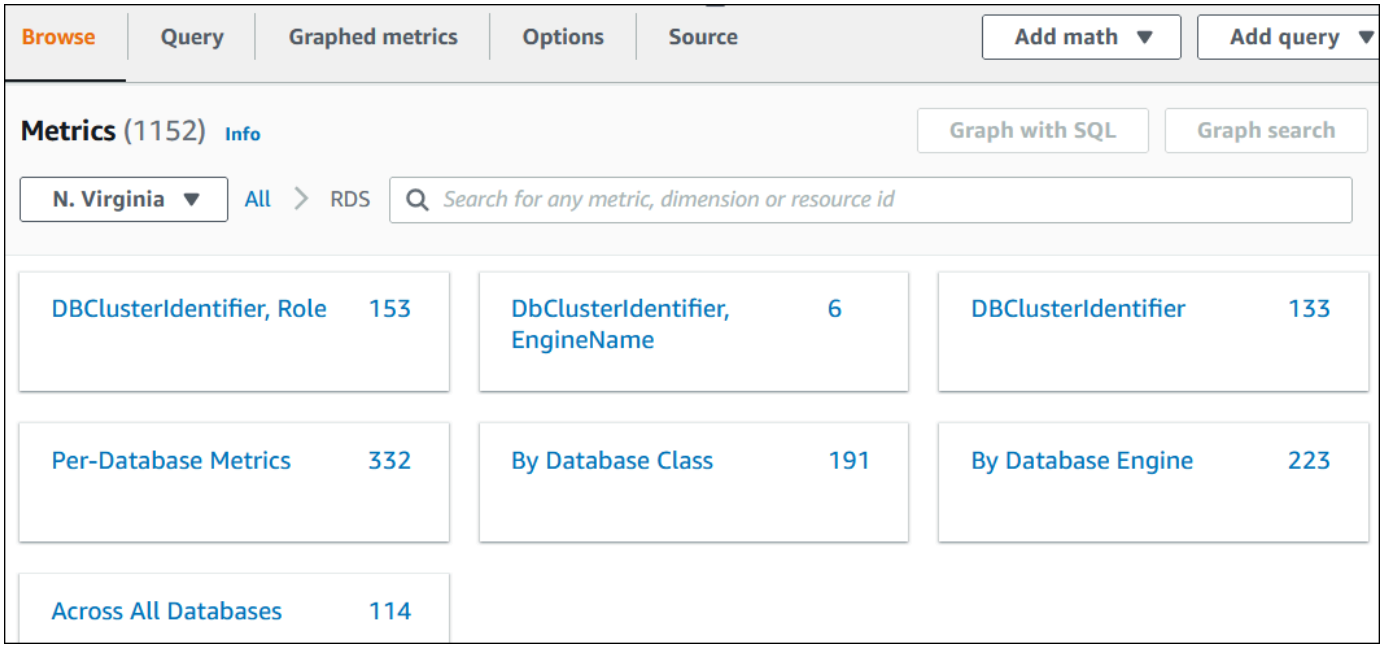


2. 如有需要，請變更 AWS 區域。從導覽列中，選擇 AWS 資源 AWS 區域 所在的 。如需詳細資訊，請參閱 Regions and endpoints。
3. 在導覽窗格中，選擇 Metrics (指標)，然後選擇 All metrics (所有指標)。

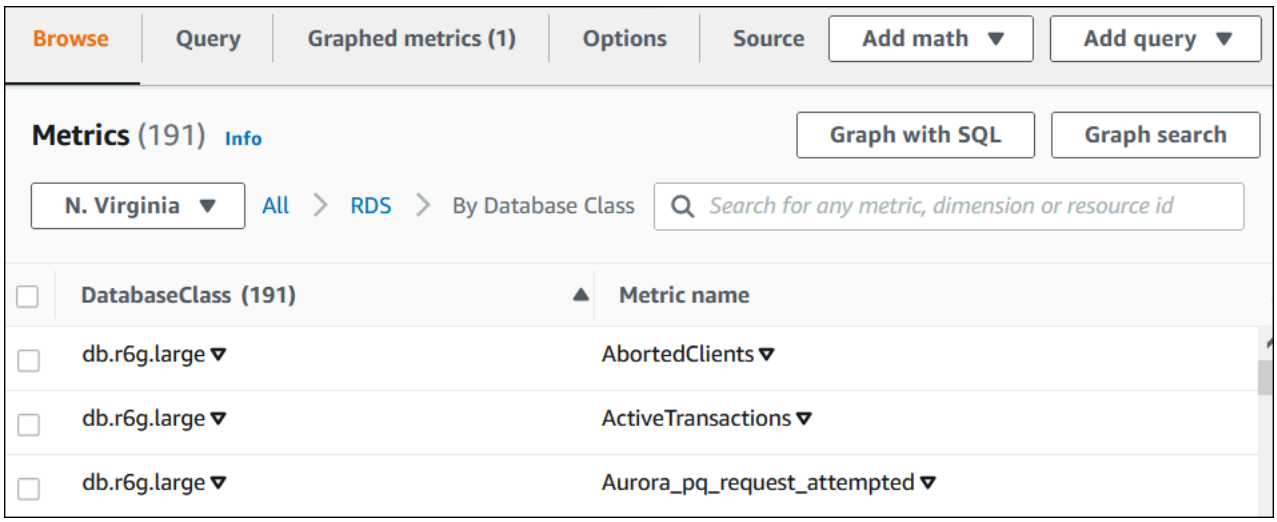


4. 向下捲動並選擇 RDS 指標命名空間。

   此頁面顯示 Amazon Aurora 維度。如需這些維度的說明，請參閱 Aurora 的 Amazon CloudWatch 維度。



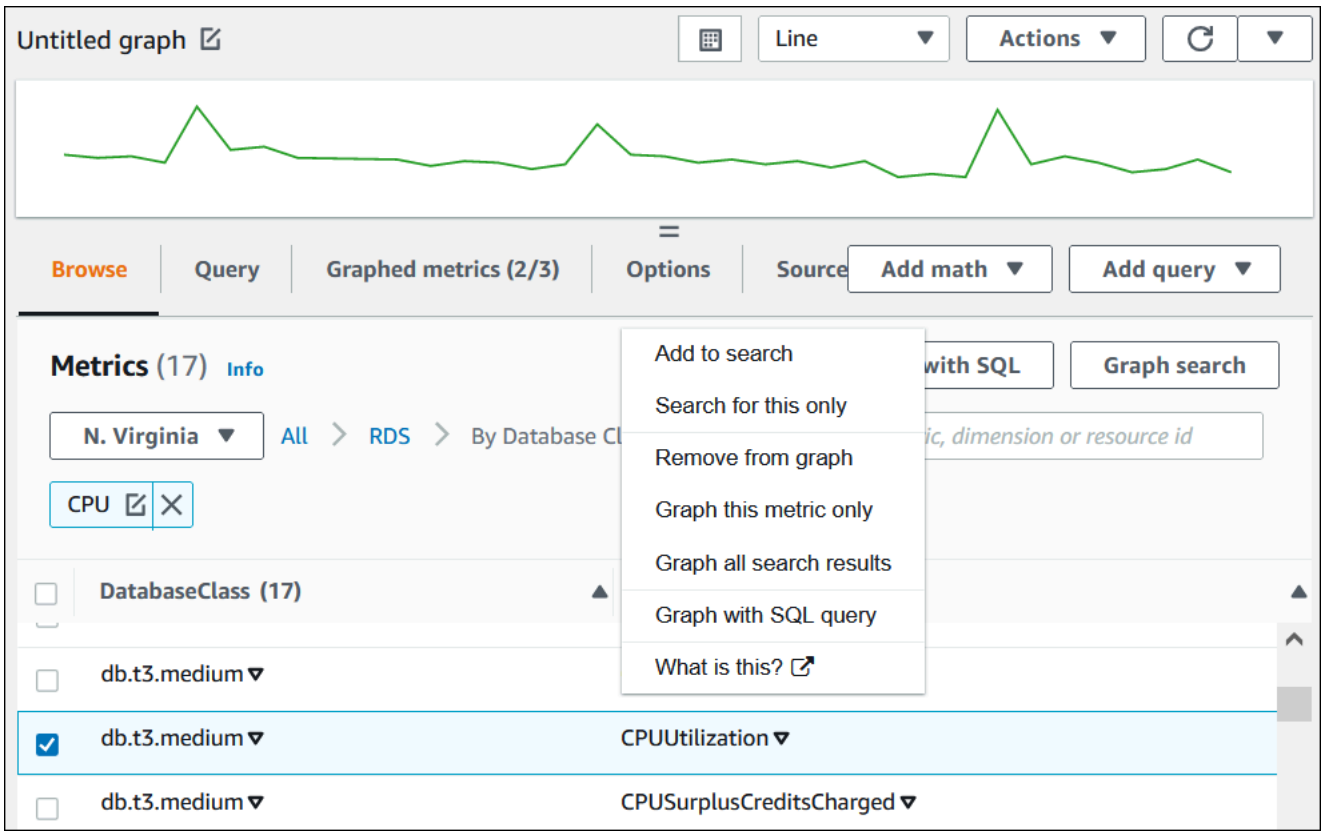
5. 選擇指標維度，例如 By Database Class (依照資料庫類別)。



6. 執行下列其中一個動作：

   - 若要排序指標，請使用直欄標題。
   - 若要將指標圖形化，請選取指標旁的核取方塊。
   - 若要依資源篩選，請選擇資源 ID，然後選擇 Add to search (新增至搜尋)。
   - 若要依指標篩選，請選擇指標名稱，然後選擇 Add to search (新增至搜尋)。

   下面的範例會篩選 db.t3.medium 類別並繪製 CPUUtilization 指標。



您可以有關如何使用 CloudWatch 指標來分析 Aurora PostgreSQL 資源用量的詳細資訊。如需詳細資訊，請參閱使用 Amazon CloudWatch 指標來分析 Aurora PostgreSQL 的資源用量

AWS CLI

若要使用 取得指標資訊 AWS CLI，請使用 CloudWatch 命令 `list-metrics`。以下範例會列出 AWS/RDS 命名空間中的所有指標。

```
aws cloudwatch list-metrics --namespace AWS/RDS
```

若要取得指標資料，請使用 `get-metric-data` 命令。

以下範例會取得執行個體 `my-instance` 指定 24 小時期間的 `CPUUtilization` 統計數字，精細程度為 5 分鐘。

使用下列內容建立 JSON 檔案 `CPU_metric.json`。

```
{
   "StartTime" : "2023-12-25T00:00:00Z",
   "EndTime" : "2023-12-26T00:00:00Z",
```

```
   "MetricDataQueries" : [{
     "Id" : "cpu",
     "MetricStat" : {
    "Metric" : {
        "Namespace" : "AWS/RDS",
        "MetricName" : "CPUUtilization",
        "Dimensions" : [{ "Name" : "DBInstanceIdentifier" , "Value" : my-instance}]
    },
       "Period" : 360,
       "Stat" : "Minimum"
     }
   }]
}
```

Example

對於 Linux、macOS 或 Unix：

```
aws cloudwatch get-metric-data \
    --cli-input-json file://CPU_metric.json
```

在 Windows 中：

```
aws cloudwatch get-metric-data ^
     --cli-input-json file://CPU_metric.json
```

範例輸出如下所示：

```
{
    "MetricDataResults": [
        {
            "Id": "cpu",
            "Label": "CPUUtilization",
            "Timestamps": [
                "2023-12-15T23:48:00+00:00",
                "2023-12-15T23:42:00+00:00",
                "2023-12-15T23:30:00+00:00",
                "2023-12-15T23:24:00+00:00",
                ...
            ],
            "Values": [
                13.299778337027714,
                13.677507543049558,
```

```
                14.24976250395827,
                13.02521708695145,
                ...
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的取得指標的統計數字。

將 Performance Insights 指標匯出至 CloudWatch

Performance Insights 可讓您將資料庫執行個體的預先設定或自訂指標儀表板匯出至 Amazon CloudWatch。您可以將此指標儀表板匯出為新的儀表板，或將其新增至現有的 CloudWatch 儀表板。當選擇要將儀表板新增至現有的 CloudWatch 儀表板時，您可以建立標頭標籤，以便指標出現在 CloudWatch 儀表板的個別區段中。

您可以在 CloudWatch 主控台中檢視匯出的指標儀表板。如果您在匯出 Performance Insights 指標儀表板之後，將新指標新增至其中，則必須再次匯出此儀表板，才能在 CloudWatch 主控台中檢視新指標。

您也可以在 Performance Insights 儀表板中選取指標小工具，並在 CloudWatch 主控台中檢視指標資料。

如需有關在 CloudWatch 主控台中檢視指標的詳細資訊，請參閱 在 CloudWatch 主控台和 中檢視資料庫叢集指標 AWS CLI。

在下列各節中，將 Performance Insights 指標匯出至 CloudWatch 作為新的或現有的儀表板，以及在 CloudWatch 中檢視 Performance Insights 指標。

主題

- 將 Performance Insights 指標當作新儀表板匯出至 CloudWatch
- 將 Performance Insights 指標新增至現有的 CloudWatch 儀表板
- 在 CloudWatch 中檢視 Performance Insights 指標小工具

將 Performance Insights 指標當作新儀表板匯出至 CloudWatch

從 Performance Insights 儀表板中選擇預先設定或自訂指標儀表板，並將其當作新儀表板匯出至 CloudWatch。您可以在 CloudWatch 主控台中檢視匯出的儀表板。

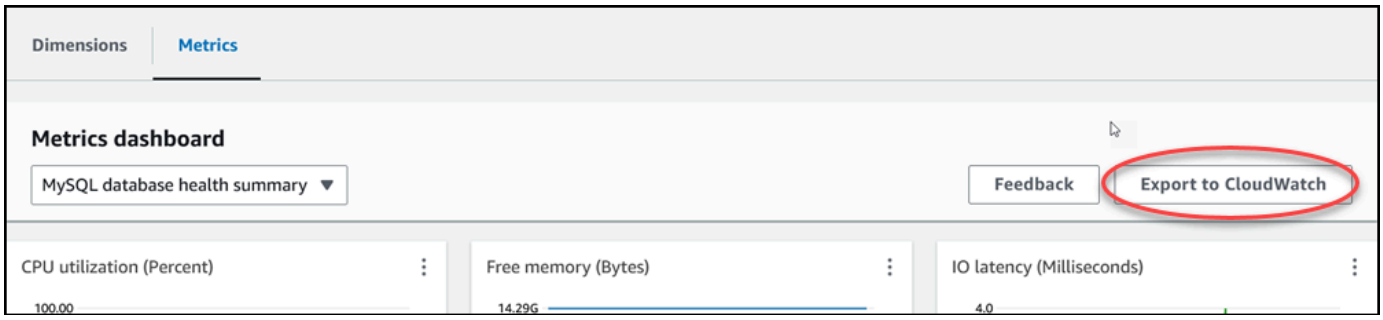
將 Performance Insights 指標儀表板當作新儀表板匯出至 CloudWatch

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在左側導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。

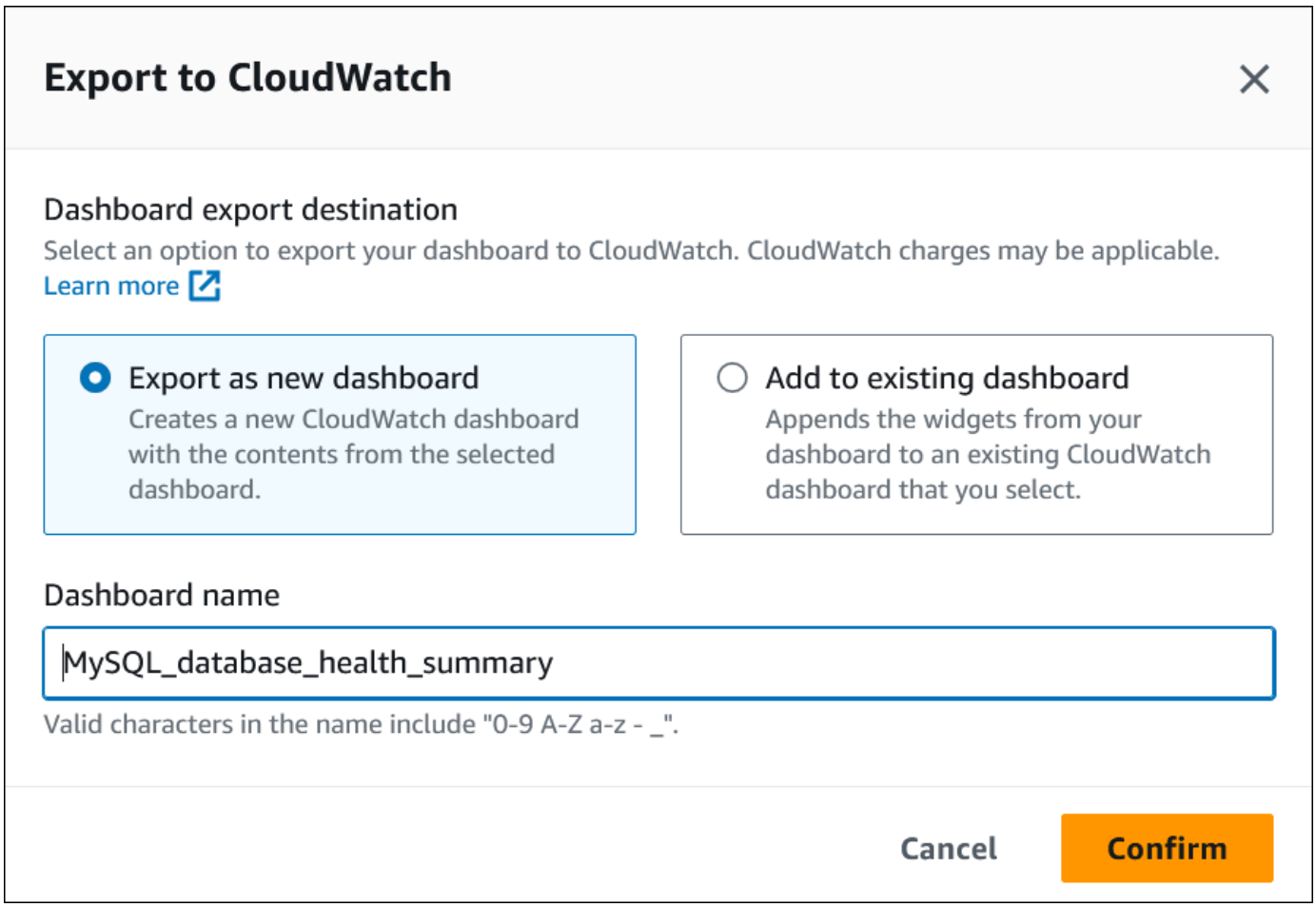
   資料庫執行個體的績效詳情儀表板即會出現。
4. 向下捲動並選擇指標。

   根據預設，具有 Performance Insights 指標的預先設定儀表板即會出現。
5. 選擇預先設定或自訂儀表板，然後選擇匯出至 CloudWatch。

   匯出至 CloudWatch 視窗隨即出現。

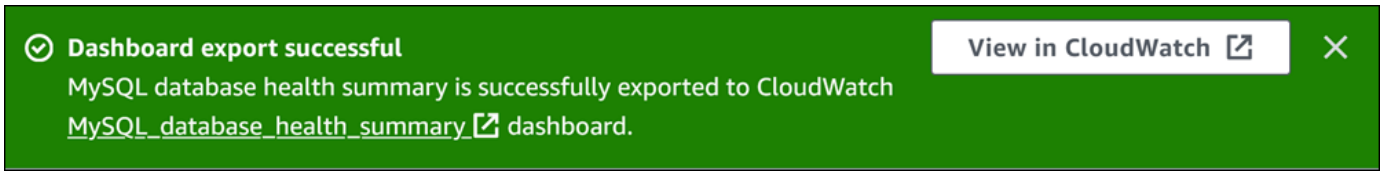


6. 選擇匯出為新儀表板。



7. 在儀表板名稱欄位中輸入新儀表板的名稱，然後選擇確認。

   在成功匯出儀表板之後，橫幅會顯示一則訊息。



8. 在橫幅中選擇連結或在 CloudWatch 中檢視，以在 CloudWatch 主控台中檢視指標儀表板。

將 Performance Insights 指標新增至現有的 CloudWatch 儀表板

將預先設定或自訂指標儀表板新增至現有的 CloudWatch 儀表板。您可以將標籤新增至指標儀表板，以出現在 CloudWatch 儀表板的個別區段中。

將指標匯出至現有的 CloudWatch 儀表板

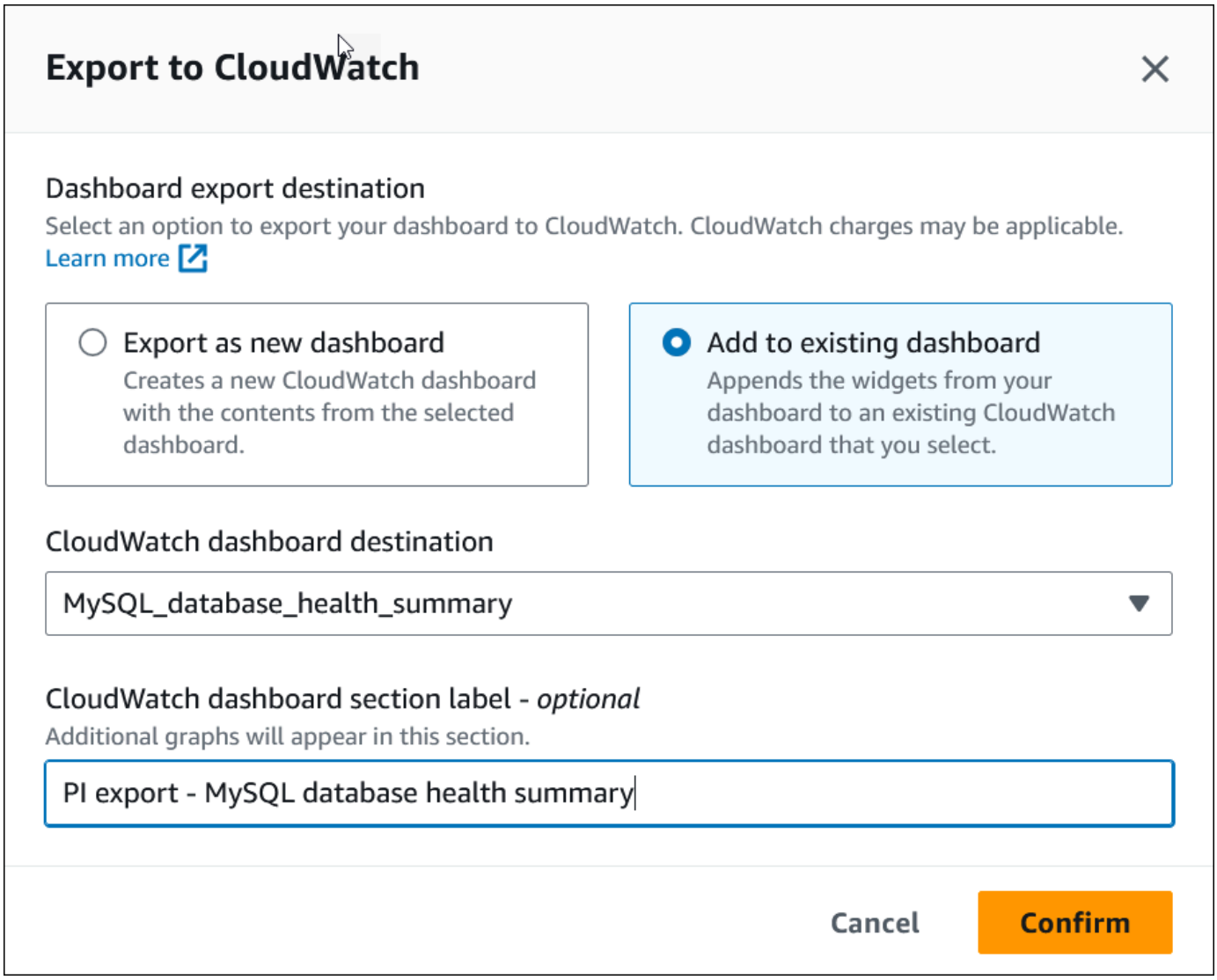
1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。

2. 在左側導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。

   資料庫執行個體的績效詳情儀表板即會出現。
4. 向下捲動並選擇指標。

   依預設，具有 Performance Insights 指標的預先設定儀表板即會出現。
5. 選擇預先設定或自訂儀表板，然後選擇匯出至 CloudWatch。

   匯出至 CloudWatch 視窗隨即出現。
6. 選擇新增至現有儀表板。



7. 指定儀表板目的地和標籤，然後選擇確認。

- CloudWatch 儀表板目的地 – 選擇現有的 CloudWatch 儀表板。
- CloudWatch 儀表板區段標籤 – 選用 – 輸入 Performance Insights 指標的名稱，以出現在 CloudWatch 儀表板的這個區段中。

在成功匯出儀表板之後，橫幅會顯示一則訊息。

8. 在橫幅中選擇連結或在 CloudWatch 中檢視，以在 CloudWatch 主控台中檢視指標儀表板。

在 CloudWatch 中檢視 Performance Insights 指標小工具

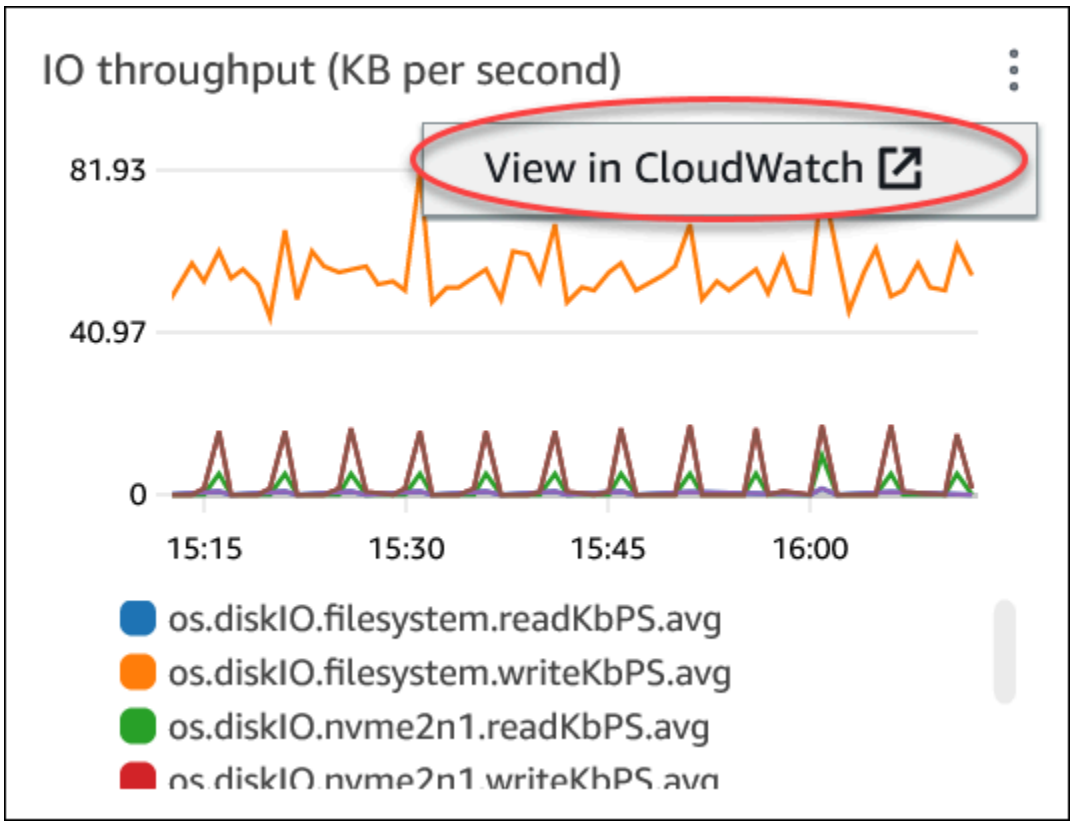
在 Amazon RDS Performance Insights 儀表板中選取 Performance Insights 指標小工具，並在 CloudWatch 主控台中檢視指標資料。

在 CloudWatch 主控台中匯出指標小工具和檢視指標資料

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在左側導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。

   資料庫執行個體的績效詳情儀表板即會出現。
4. 向下捲動至指標。

   依預設，具有 Performance Insights 指標的預先設定儀表板即會出現。
5. 選擇指標小工具，然後在功能表中選擇在 CloudWatch 中檢視。



指標資料會出現在 CloudWatch 主控台中。

建立 CloudWatch 警示來監控 Amazon Aurora

您可以建立 CloudWatch 警報，在警示變更狀態時傳送 Amazon SNS 訊息。警示會在您指定的期間監看單一指標。警示會根據在數段期間內與指定閾值相關的指標值，來執行一個或多個動作。此動作是傳送到 Amazon SNS 主題或 Amazon EC2 Auto Scaling 政策的通知。

警示僅會針對持續狀態變更呼叫動作。CloudWatch 警示不會僅因為它們處於特定狀態而叫用動作。狀態必須發生變更並維持一段指定的時間。

> **Note**
>
> 針對 Aurora，使用 `WRITER` 或 `READER` 角色指標來設定警示，而不是依賴特定資料庫執行個體的指標。Aurora 資料庫執行個體角色可以隨時間變更角色。您可以在 CloudWatch 主控台中找到這些以角色為基礎的指標。
>
> Aurora Auto Scaling 會根據 `READER` 角色指標自動設定警示。如需 Aurora Auto Scaling 的詳細資訊，請參閱 使用 Amazon Aurora 自動擴展搭配 Aurora 複本。

您可以使用 CloudWatch 主控台中的 DB_PERF_INSIGHTS 指標數學函數查詢 Amazon RDS for Performance Insights 計數器指標。DB_PERF_INSIGHTS 函數也包含次分鐘間隔的 DBLoad 指標。您可以根據這些指標設定 CloudWatch 警示。

如需如何建立警示的詳細資訊，請參閱從 AWS 資料庫建立 Performance Insights 計數器指標的警示。

使用 AWS CLI 設定警示

- 呼叫 `put-metric-alarm`。如需更多詳細資訊，請參閱 AWS CLI 命令參考。

使用 CloudWatch API 設定警示

- 呼叫 `PutMetricAlarm`。如需更多詳細資訊，請參閱 Amazon CloudWatch API 參考。

如需有關設定 Amazon SNS 主題和建立警示的詳細資訊，請參閱使用 Amazon CloudWatch 警示。

使用 CloudWatch Database Insights 監控 Amazon Aurora 資料庫

使用 Database Insights 監控 Amazon Aurora 資料庫執行個體機群的資料庫負載 (資料庫負載)。資料庫負載會測量資料庫中工作階段活動的層級。您可以使用 Database Insights 大規模分析 Amazon Aurora 資料庫的效能，並進行疑難排解。

利用 Database Insights，您可以將資料庫機群的資料庫負載視覺化，並依等候、SQL 陳述式、主機或使用者篩選負載。

根據預設，RDS 會為您的 Amazon Aurora 資料庫啟用 Database Insights 的標準模式。當您為資料庫叢集開啟 Database Insights 的進階模式時，RDS 會為叢集中的每個資料庫執行個體啟用 Database Insights。

如需在 Amazon CloudWatch 主控台中使用 Database Insights 的詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的 CloudWatch Database Insights。

定價

如需定價的詳細資訊，請參閱 Amazon CloudWatch 定價。

主題

- Database Insights 的 Amazon Aurora 資料庫引擎、區域和執行個體類別支援
- 開啟適用於 Amazon Aurora 的 Database Insights 的進階模式
- 開啟 Amazon Aurora 的 Database Insights 的標準模式
- 使用 Amazon Aurora 的 Database Insights 設定資料庫以監控慢速 SQL 查詢
- Amazon Aurora 資料庫洞見的考量事項

Database Insights 的 Amazon Aurora 資料庫引擎、區域和執行個體類別支援

下表提供可支援 Database Insights 的 Amazon Aurora 資料庫引擎。

Amazon Aurora 資料庫引擎	支援的引擎版本和區域	執行個體類別限制
Amazon Aurora MySQL-Compatible Edition	如需 Database Insights 搭配 Aurora MySQL 的版本和區域可用性的詳細資訊，請參閱 搭配 Aurora MySQL 的 Performance Insights。	Database Insights 具有以下引擎類別限制： • db.t2 – 不支援 • db.t3 – 不支援 • db.t4g.micro 和 db.t4g.small – 不支援
Amazon Aurora PostgreSQL-Compatible Edition	如需 Database Insights 搭配 Aurora PostgreSQL 的版本和區域可用性的詳細資訊，請參閱 搭配 Aurora PostgreSQL 的 Performance Insights。	不適用
Aurora PostgreSQL Limitless Database	如需搭配 Aurora PostgreSQL Limitless Database 使用 Database Insights 的詳細資訊，請參閱 使用 CloudWatch Database Insights 監控 Aurora PostgreSQL Limitless Database。	不適用

Database Insights 支援 Amazon Aurora Serverless v2。

Amazon Aurora 資料庫引擎、區域和執行個體類別支援 Database Insights 功能

下表提供可支援 Database Insights 功能的 Amazon Aurora 資料庫引擎。

功能	定價方案	支援的區域	支援的資料庫引擎	支援的執行個體類別
績效詳情的 SQL 統計數字	全部	全部	全部	全部
分析一段時間區間的資料庫效能	僅限付費方案	全部	全部	全部，但 db.serverless (Aurora Serverless v2) 除外
檢視 Performance Insights 主動建議	僅限付費方案	• 美國東部 (俄亥俄) • 美國東部 (維吉尼亞北部) • 美國西部 (加利佛尼亞北部) • 美國西部 (奧勒岡) • 亞太區域 (孟買) • 亞太區域 (首爾) • 亞太區域 (新加坡) • 亞太地區 (雪梨) • 亞太區域 (東京) • 加拿大 (中部) • 歐洲 (法蘭克福)	全部	全部，但 db.serverless (Aurora Serverless v2) 除外

功能	定價方案	支援的區域	支援的資料庫引擎	支援的執行個體類別
		• 歐洲 (愛爾蘭) • 歐洲 (倫敦) • 歐洲 (巴黎) • 歐洲 (斯德哥爾摩) • 南美洲 (聖保羅)		

Amazon Aurora 區域支援 Database Insights

Aurora 在以下支援 Database Insights AWS 區域。

- 美國東部 (維吉尼亞北部)
- 美國東部 (俄亥俄)
- 美國西部 (加利佛尼亞北部)
- 美國西部 (奧勒岡)
- 非洲 (開普敦)
- 亞太地區 (香港)
- 亞太區域 (海德拉巴)
- 亞太地區 (雅加達)
- 亞太地區 (馬來西亞)
- 亞太地區 (墨爾本)
- 亞太地區 (孟買)
- 亞太區域 (大阪)
- 亞太區域 (首爾)
- 亞太區域 (新加坡)
- 亞太地區 (雪梨)
- 亞太區域 (東京)
- 加拿大 (中部)

- 加拿大西部 (卡加利)
- 歐洲 (法蘭克福)
- 歐洲 (愛爾蘭)
- 歐洲 (倫敦)
- 歐洲 (米蘭)
- Europe (Paris)
- 歐洲 (西班牙)
- 歐洲 (斯德哥爾摩)
- 歐洲 (蘇黎世)
- 以色列 (特拉維夫)
- Middle East (Bahrain)
- 中東 (阿拉伯聯合大公國)
- 南美洲 (聖保羅)
- AWS GovCloud （美國東部）
- AWS GovCloud （美國西部）

開啟適用於 Amazon Aurora 的 Database Insights 的進階模式

若要開啟 Amazon Aurora 的 Database Insights 的進階模式，請使用下列程序。

在建立資料庫叢集時，開啟 Database Insights 的進階模式

建立 Amazon Aurora 的資料庫時，開啟 Database Insights 的進階模式。

Console

在此主控台中，您可以在建立資料庫叢集時開啟 Database Insights 的進階模式。Database Insights 的設定適用於資料庫叢集中的所有資料庫執行個體。

使用主控台建立資料庫叢集時，開啟 Database Insights 的進階模式

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 選擇 Databases (資料庫)。
3. 選擇建立資料庫。

4. 在 Database Insights 區段中，選取進階模式。然後選擇下列選項：
   - Retention (保留) – 保留 Performance Insights 資料的時間。Database Insights 進階模式的保留期間必須為 15 個月。
   - AWS KMS key – 指定您的 KMS 金鑰。績效詳情使用您的 KMS 金鑰來加密所有可能的敏感資料。將會對傳輸中與靜態資料進行加密。如需詳細資訊，請參閱加密 Amazon Aurora 資源。
5. 選擇建立資料庫。

AWS CLI

若要在建立資料庫叢集時開啟 Database Insights 的進階模式，請呼叫 create-db-cluster AWS CLI 命令並提供下列值：

- `--database-insights-mode advanced` 以開啟 Database Insights 的進階模式。
- `--engine` – 資料庫叢集的資料庫引擎。
- `--db-cluster-identifier` – 資料庫叢集的識別碼。
- `--enable-performance-insights` 以開啟適用於 Database Insights 的 Performance Insights。
- `--performance-insights-retention-period` – 資料庫叢集的資料保留期間。若要開啟 Database Insights，保留期間必須至少為 465 天。

下列範例會在建立資料庫叢集時啟用 Database Insights 的進階模式。

針對 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
    --database-insights-mode advanced \
    --engine aurora-postgresql \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 465
```

針對 Windows：

```
aws rds create-db-cluster ^
    --database-insights-mode advanced ^
    --engine aurora-postgresql ^
```

```
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

若要在建立資料庫叢集時開啟 Database Insights 的進階模式，請為 CreateDBCluster Amazon RDS API 操作指定下列參數。

- `DatabaseInsightsMode` 至 `advanced`
- `EnablePerformanceInsights` 至 `True`
- `PerformanceInsightsRetentionPeriod` 到至少 465 天

在修改資料庫叢集時，開啟 Database Insights 的進階模式

修改 Amazon Aurora 的資料庫時，開啟 Database Insights。修改資料庫叢集以啟用 Database Insights 的進階模式不會導致停機時間。

> **Note**
>
> 若要啟用 Database Insights，資料庫叢集中的每個資料庫執行個體都必須具有相同的 Performance Insights 和增強型監控設定。

Console

在主控台中，您可以在修改資料庫叢集時開啟 Database Insights 的進階模式。Database Insights 的設定適用於資料庫叢集中的所有資料庫執行個體。

使用主控台修改資料庫叢集時，開啟 Database Insights 的進階模式

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 選擇 Databases (資料庫)。
3. 選擇一個資料庫執行個體，然後選擇 Modify (修改)。
4. 在 Database Insights 區段中，選取進階模式。然後選擇下列選項：
    - Retention (保留) – 保留 Performance Insights 資料的時間。Database Insights 進階模式的保留期間必須為 15 個月。

- AWS KMS key – 指定您的 KMS 金鑰。績效詳情使用您的 KMS 金鑰來加密所有可能的敏感資料。將會對傳輸中與靜態資料進行加密。如需更多詳細資訊，請參閱 加密 Amazon Aurora 資源。

5. 選擇 Continue (繼續)。
6. 在 Scheduling of Modifications (修改排程) 中，選擇 Apply immediately (立即套用)。如果您選擇在下一個排定的維護時段套用，則資料庫會忽略此設定，並立即開啟 Database Insights 的進階模式。
7. 選擇修改叢集。

AWS CLI

若要在修改資料庫叢集時開啟 Database Insights 的進階模式，請呼叫 modify-db-cluster AWS CLI 命令並提供下列值：

- `--database-insights-mode advanced` 以開啟 Database Insights 的進階模式。
- `--db-cluster-identifier` – 資料庫叢集的識別碼。
- `--enable-performance-insights` 以開啟適用於 Database Insights 的 Performance Insights。
- `--performance-insights-retention-period`：資料庫叢集的資料保留期間。若要開啟 Database Insights 的進階模式，保留期必須至少為 465 天。

下列範例會在修改資料庫叢集時啟用 Database Insights 的進階模式。

針對 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --database-insights-mode advanced \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 465
```

針對 Windows：

```
aws rds modify-db-cluster ^
    --database-insights-mode advanced ^
    --db-cluster-identifier sample-db-identifier ^
```

```
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

若要在修改資料庫叢集時開啟 Database Insights 的進階模式，請為 ModifyDBCluster Amazon RDS API 操作指定下列參數。

- `DatabaseInsightsMode` 至 `advanced`
- `EnablePerformanceInsights` 至 `True`
- `PerformanceInsightsRetentionPeriod` 到至少 465 天

開啟 Amazon Aurora 的 Database Insights 的標準模式

若要開啟 Amazon Aurora 的 Database Insights 的標準模式，請使用下列程序。

在建立資料庫叢集時，開啟 Database Insights 的標準模式

建立 Amazon Aurora 的資料庫時，開啟 Database Insights 的標準模式。

Console

在此主控台中，您可以在建立資料庫叢集時開啟 Database Insights 的標準模式。Database Insights 的設定適用於資料庫叢集中的所有資料庫執行個體。

使用主控台建立資料庫叢集時，開啟 Database Insights 的標準模式

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 選擇 Databases (資料庫)。
3. 選擇建立資料庫。
4. 在 Database Insights 區段中，選取標準模式。然後，從下列選項中選擇以開啟或關閉 Performance Insights：
   - 若要關閉 Performance Insights，請取消選取啟用 Performance Insights。
   - 若要開啟 Performance Insights，請選取啟用 Performance Insights。若要設定 Performance Insights，請指定下列選項：
     - Retention (保留) – 保留 Performance Insights 資料的時間。保留期間必須至少為 7 天。

- AWS KMS key – 指定您的 KMS 金鑰。績效詳情使用您的 KMS 金鑰來加密所有可能的敏感資料。將會對傳輸中與靜態資料進行加密。如需詳細資訊，請參閱加密 Amazon Aurora 資源。

5. 選擇建立資料庫。

AWS CLI

若要在建立資料庫叢集時開啟 Database Insights 的標準模式，請呼叫 create-db-cluster AWS CLI 命令並提供下列值：

- `--database-insights-mode standard` 以開啟 Database Insights 的標準模式。
- `--engine` – 資料庫叢集的資料庫引擎。
- `--db-cluster-identifier` – 資料庫叢集的識別碼。
- `--enable-performance-insights` 或 `--no-enable-performance-insights` 以開啟或關閉 Performance Insights。如果您指定 `--enable-performance-insights`，還必須指定 `--performance-insights-retention-period` – 資料庫叢集資料的保留期間。保留期間必須至少為 7 天。

下列範例會在建立資料庫叢集時啟用 Database Insights 和 Performance Insights 的標準模式。

針對 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
    --database-insights-mode standard \
    --engine aurora-postgresql \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 7
```

針對 Windows：

```
aws rds create-db-cluster ^
    --database-insights-mode standard ^
    --engine aurora-postgresql ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 7
```

下列範例會在建立資料庫叢集時，啟用 Database Insights 的標準模式及停用 Performance Insights。

針對 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
    --database-insights-mode standard \
    --engine aurora-postgresql \
    --db-cluster-identifier sample-db-identifier \
    --no-enable-performance-insights
```

針對 Windows：

```
aws rds create-db-cluster ^
    --database-insights-mode standard ^
    --engine aurora-postgresql ^
    --db-cluster-identifier sample-db-identifier ^
    --no-enable-performance-insights
```

RDS API

若要在建立資料庫叢集時開啟 Database Insights 的標準模式，請為您的 CreateDBCluster Amazon RDS API 操作指定下列參數。

- `DatabaseInsightsMode` 至 `standard`
- `EnablePerformanceInsights` 設為 `True` 或 `False`。如果您將 `EnablePerformanceInsights` 設定為 `True`，您必須將 `PerformanceInsightsRetentionPeriod` 設定為至少 7 天。

在修改資料庫叢集時，開啟 Database Insights 的標準模式

修改 Amazon Aurora 的資料庫時，開啟 Database Insights 的標準模式。修改資料庫叢集以啟用 Database Insights 的標準模式不會導致停機時間。

> **Note**
>
> 若要啟用 Database Insights，資料庫叢集中的每個資料庫執行個體都必須具有相同的 Performance Insights 和增強型監控設定。

Console

在此主控台中，您可以在修改資料庫叢集時開啟 Database Insights 的標準模式。Database Insights 的設定適用於資料庫叢集中的所有資料庫執行個體。

使用主控台修改資料庫叢集時，開啟 Database Insights 的標準模式

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 選擇 Databases (資料庫)。
3. 選擇一個資料庫執行個體，然後選擇 Modify (修改)。
4. 在 Database Insights 區段中，選取標準模式。然後，從以下選項中選擇：
   - 若要關閉 Performance Insights，請取消選取啟用 Performance Insights。
   - 若要開啟 Performance Insights，請選取啟用 Performance Insights。若要設定 Performance Insights，請指定下列選項：
     - Retention (保留) – 保留 Performance Insights 資料的時間。保留期間必須至少為 7 天。
     - AWS KMS key – 指定您的 KMS 金鑰。績效詳情使用您的 KMS 金鑰來加密所有可能的敏感資料。將會對傳輸中與靜態資料進行加密。如需更多詳細資訊，請參閱 加密 Amazon Aurora 資源。
5. 選擇 Continue (繼續)。
6. 在 Scheduling of Modifications (修改排程) 中，選擇 Apply immediately (立即套用)。如果您選擇在下一個排定的維護時段套用，則資料庫會忽略此設定，並立即開啟 Database Insights 的標準模式。
7. 選擇修改叢集。

AWS CLI

若要在修改資料庫叢集時開啟 Database Insights 的標準模式，請呼叫 modify-db-cluster AWS CLI 命令並提供下列值：

- `--database-insights-mode standard` 以開啟 Database Insights 的標準模式。
- `--db-cluster-identifier` – 資料庫叢集的識別碼。
- `--enable-performance-insights` 或 `--no-enable-performance-insights` 以開啟或關閉 Performance Insights。如果您指定 `--enable-performance-insights`，還必須指定

`--performance-insights-retention-period` – 資料庫叢集的資料保留期間。保留期間必須至少為 7 天。

下列範例會在修改資料庫叢集時啟用 Database Insights 的標準模式及啟用 Performance Insights。

針對 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --database-insights-mode standard \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 7
```

針對 Windows：

```
aws rds modify-db-cluster ^
    --database-insights-mode standard ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 7
```

下列範例會在修改資料庫叢集時啟用 Database Insights 的標準模式及停用 Performance Insights。

針對 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --database-insights-mode standard \
    --db-cluster-identifier sample-db-identifier \
    --no-enable-performance-insights
```

針對 Windows：

```
aws rds modify-db-cluster ^
    --database-insights-mode standard ^
    --db-cluster-identifier sample-db-identifier ^
    --no-enable-performance-insights
```

RDS API

若要在修改資料庫叢集時開啟 Database Insights 的標準模式，請為 ModifyDBCluster Amazon RDS API 操作指定下列參數。

- `DatabaseInsightsMode` 至 `standard`
- `EnablePerformanceInsights` 設為 `True` 或 `False`。如果您將 `EnablePerformanceInsights` 設定為 `True`，您必須將 `PerformanceInsightsRetentionPeriod` 設定為至少 7 天。

使用 Amazon Aurora 的 Database Insights 設定資料庫以監控慢速 SQL 查詢

若要監控資料庫的慢速 SQL 查詢，您可以使用 Database Insights 儀表板中的慢速 SQL 查詢區段。在設定資料庫以監控慢速 SQL 查詢之前，慢速 SQL 查詢區段為空白。

如需在 Database Insights 儀表板中監控慢速 SQL 查詢的詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的檢視 CloudWatch Database Insights 的資料庫執行個體儀表板。

若要設定資料庫以使用 Database Insights 監控慢速 SQL 查詢，請完成下列步驟：

1. 啟用日誌匯出至 CloudWatch Logs。
2. 建立或修改資料庫叢集的資料庫叢集參數群組。

如需設定日誌匯出的詳細資訊，請參閱《Amazon Aurora 使用者指南》中的將資料庫日誌發佈至 Amazon CloudWatch Logs。

若要建立或修改資料庫叢集參數群組，請參閱下列主題。

- 在 Amazon Aurora 中建立資料庫叢集參數群組
- 在 Amazon Aurora 中修改資料庫叢集參數群組中的參數

Amazon Aurora MySQL

若要設定 Amazon Aurora MySQL 資料庫叢集來監控慢速 SQL 查詢，您可以使用下列參數組合做為範例：

- `slow_query_log`：設定為 `1`
- `long_query_time`：設定為 `1.0`
- `log_output`：設定為 `FILE`

這是一個可能的組態。如需 MySQL 慢速查詢日誌參數和其他組態選項的完整指南，請參閱慢速查詢日誌的 MySQL 文件。

Amazon Aurora PostgreSQL

若要設定 Amazon Aurora PostgreSQL 資料庫叢集來監控慢速 SQL 查詢，您可以使用下列參數組合做為範例。請注意，設定這些參數可能會降低資料庫叢集的效能。

- `log_min_duration_statement`：設定為 `1000`
- `log_statement`：設定為 `none`
- `log_destination`：設定為 `stderr`

這是一個可能的組態。如需 PostgreSQL 記錄參數和其他組態選項的完整指南，請參閱記錄組態的 PostgreSQL 文件。

> **Note**
>
> 對於 Aurora MySQL，您可以使用 1-毫秒精細程度設定參數 `long_query_time`。例如，您可以將此參數設為 `0.000001`。根據資料庫執行個體上的查詢數量，參數 `long_query_time` 的值會降低效能。從值 `1.0` 開始，然後根據您的工作負載進行調整。當您將此參數設定為 `0` 時，Database Insights 會記錄所有查詢。

如需 Aurora MySQL 和 Aurora PostgreSQL 日誌的詳細資訊，請參閱以下內容。

- Aurora MySQL資料庫日誌檔案
- Aurora PostgreSQL 資料庫日誌檔案

Amazon Aurora 資料庫洞見的考量事項

以下是 Amazon Aurora Database Insights 的考量事項。

- 您無法管理資料庫叢集中資料庫執行個體的 Database Insights。
- 若要啟用 Database Insights 的進階模式，您必須啟用 Performance Insights，並將 Performance Insights 保留期間設定為至少 465 天 (15 個月)。除了 Database Insights 的成本之外，將 Performance Insights 保留期間設定為 15 個月無需額外費用。如需 Database Insights 定價的資訊，請參閱 Amazon CloudWatch 定價。
- 若要啟用 Database Insights，資料庫叢集中的每個資料庫執行個體都必須具有相同的 Performance Insights 和增強型監控設定。

- 修改資料庫叢集以啟用 Database Insights 的任一模式不會導致停機時間。

在 Amazon Aurora 上使用績效詳情監控資料庫負載

Important

AWS 已宣布績效詳情的end-of-life：2026 年 6 月 30 日。在此日期之後，Amazon RDS 將不再支援 Performance Insights 主控台體驗、彈性保留期間 (1-24 個月) 及其相關定價。Performance Insights API 將繼續存在，不會有任何定價變更。Performance Insights API 的成本會顯示在您的 AWS 帳單中，其中包含 CloudWatch Database Insights 的成本。
我們建議您在 2026 年 6 月 30 日之前，使用已付費的績效詳情方案，將任何資料庫叢集升級至資料庫詳情的進階模式。如需有關升級至 Database Insights 進階模式的資訊，請參閱 開啟適用於 Amazon Aurora 的 Database Insights 的進階模式。
如果您不採取任何動作，則使用 Performance Insights 的資料庫叢集依預設會使用 Database Insights 的標準模式。使用 Database Insights 的標準模式時，您可能會失去超過 7 天的效能資料歷史記錄存取權，而且可能無法在 Amazon RDS 主控台中使用執行計畫和隨需分析功能。2026 年 6 月 30 日之後，只有進階模式的 Database Insights 將支援執行計畫和隨需分析。
透過 CloudWatch Database Insights，您可以監控資料庫機群的資料庫負載，以及大規模分析效能和進行疑難排解。如需 Database Insights 的詳細資訊，請參閱 使用 CloudWatch Database Insights 監控 Amazon Aurora 資料庫。如需定價資訊，請參閱 Amazon CloudWatch 定價。

績效詳情會擴展現有 Amazon Aurora 監控功能，以說明並協助您分析叢集效能。利用績效詳情儀表板，您可將 Amazon Aurora 叢集負載 上的資料庫負載視覺化，並依等候、SQL 陳述式、主機或使用者篩選負載。如需有關搭配 Amazon DocumentDB 使用績效詳情的資訊，請參閱《Amazon DocumentDB 開發人員指南》。

主題

- Amazon Aurora 上的績效詳情概觀
- 開啟和關閉 Aurora 的 Performance Insights
- Aurora MySQL 上 Performance Insights 的效能結構描述概觀
- 設定績效詳情的存取政策
- 使用績效詳情儀表板來分析指標
- 檢視 Performance Insights 主動建議
- 使用適用於 Aurora 的 Performance Insights API 來擷取指標

- 使用 AWS CloudTrail 記錄績效詳情呼叫
- Performance Insights API 和界面 VPC 端點 (AWS PrivateLink)

Amazon Aurora 上的績效詳情概觀

> **Important**
>
> AWS 已宣布績效詳情的end-of-life：2026 年 6 月 30 日。在此日期之後，Amazon RDS 將不再支援 Performance Insights 主控台體驗、彈性保留期間 (1-24 個月) 及其相關定價。Performance Insights API 將繼續存在，不會有任何定價變更。Performance Insights API 的成本會顯示在您的 AWS 帳單中，其中包含 CloudWatch Database Insights 的成本。
> 我們建議您在 2026 年 6 月 30 日之前，使用已付費的績效詳情方案，將任何資料庫叢集升級至資料庫詳情的進階模式。如需有關升級至 Database Insights 進階模式的資訊，請參閱 開啟適用於 Amazon Aurora 的 Database Insights 的進階模式。
> 如果您不採取任何動作，則使用 Performance Insights 的資料庫叢集依預設會使用 Database Insights 的標準模式。使用 Database Insights 的標準模式時，您可能會失去超過 7 天的效能資料歷史記錄存取權，而且可能無法在 Amazon RDS 主控台中使用執行計畫和隨需分析功能。2026 年 6 月 30 日之後，只有進階模式的 Database Insights 將支援執行計畫和隨需分析。
> 透過 CloudWatch Database Insights，您可以監控資料庫機群的資料庫負載，以及大規模分析效能和進行疑難排解。如需 Database Insights 的詳細資訊，請參閱 使用 CloudWatch Database Insights 監控 Amazon Aurora 資料庫。如需定價資訊，請參閱 Amazon CloudWatch 定價。

根據預設，RDS 會在所有 Amazon RDS 引擎的主控台建立精靈中啟用 Performance Insights。若您在資料庫叢集層級開啟 Performance Insights，則 RDS 會為叢集中的每個資料庫執行個體啟用 Performance Insights。如果您在一個資料庫執行個體上有多個資料庫，績效詳情會彙整效能資料。

您可於下方影片中找到 Amazon Aurora 的績效詳情概觀。

使用績效詳情來分析 Amazon Aurora PostgreSQL 效能

主題

- 資料庫負載
- 最大 CPU
- Performance Insights 的 Amazon Aurora 資料庫引擎、區域和執行個體類別支援

- Performance Insights 的定價和資料保留

資料庫負載

資料庫負載 (資料庫負載) 會測量資料庫中工作階段活動的層級。DBLoad 是 Performance Insights 中的關鍵指標，而 Performance Insights 會收集每秒資料庫負載。

主題

- 作用中的工作階段
- 平均作用中工作階段
- 平均作用中執行數
- 維度

作用中的工作階段

資料庫工作階段代表應用程式與關聯式資料庫的對話。作用中工作階段是已提交工作給資料庫引擎且正在等待回應的連線。

工作階段處於作用中是指工作階段正在 CPU 上執行，或等待資源變成可用以繼續執行。例如，作用中工作階段可能等待分頁 (或區塊) 讀入記憶體中，然後從分頁讀取資料時耗用 CPU。

平均作用中工作階段

平均作用中工作階段 (AAS) 是 DBLoad 績效詳情的單位。它會測量資料庫上同時處於作用中狀態的工作階段數目。

每一秒，績效詳情都會取樣同時執行查詢的工作階段數目。針對每個作用中的工作階段，績效詳情會收集下列資料：

- SQL 陳述式
- 工作階段狀態 (正在 CPU 上執行或等待中)
- 主機
- 執行 SQL 的使用者

績效詳情會計算 AAS，方法是將工作階段總數除以特定時段的樣本數。例如，下表顯示執行查詢的 5 個連續範例，間隔至少為 1 秒。

樣本	執行查詢的工作階段數目	AAS	算式
1	2	2	工作階段總數 2 / 1 個樣本
2	0	1	工作階段總數 2 / 2 個樣本
3	4	2	工作階段總數 6 / 3 個樣本
4	0	1.5	工作階段總數 6 / 4 個樣本
5	4	2	工作階段總數 10 / 5 個樣本

在上述範例中，時間間隔的資料庫負載為 2 AAS。此測量表示在採集 5 個樣本的間隔期間內，於任何特定時間平均有 2 個工作階段處於作用中。

平均作用中執行數

每秒平均作用中執行數 (AAE) 與 AAS 相關。若要計算 AAE，績效詳情會將查詢的總執行時間除以時間間隔。下表顯示上表中同一個查詢的 AAE 計算。

經過時間 (秒)	總執行時間 (秒)	AAE	算式
60	120	2	120 執行秒 / 60 秒經過
120	120	1	120 執行秒 / 120 秒經過
180	380	2.11	380 執行秒/180 秒經過
240	380	1.58	380 執行秒/240 秒經過
300	600	2	600 執行秒 / 300 秒經過

在大多數情況下，查詢的 AAS 和 AAE 大致相同。不過，由於計算的輸入是不同的資料來源，所以計算通常會略有不同。

維度

db.load 指標與其他時間序列指標不同，因為您可以將它分為名為維度的子元件。您可以將維度視為 DBLoad 指標不同特性的「配量依據」類別。

診斷效能問題時，下列維度通常最實用：

主題

- 等待事件
- 最高 SQL

如需 Aurora 引擎的維度完整清單，請參閱 資料庫負載依維度配量。

等待事件

等待事件會導致 SQL 陳述式等待特定事件發生後，才能繼續執行。等待事件指出工作在何處受阻，是資料庫負載的重要維度或類別。

每個使用中的工作階段都在 CPU 上執行或等待中。例如，工作階段在記憶體中搜尋緩衝區、執行計算或執行程序程式碼時會耗用 CPU。當工作階段不耗用 CPU 時，可能是在等待記憶體緩衝區變成可用、要讀取的資料檔或要寫入的記錄檔。工作階段等待資源越久，在 CPU 上執行的時間就越短。

調校資料庫時，您通常會嘗試查明工作階段正在等待的資源。例如，兩個或三個等待事件可能佔資料庫負載的 90%。此量值表示作用中工作階段平均花最多時間等待少量資源。如果您可以找出這些等待的原因，就可以嘗試解決方案。

等待事件依據資料庫引擎而有所差異：

- 如需 Aurora MySQL 的常用等待事件清單，請參閱 Aurora MySQL 等待事件。若要了解如何使用這些等待事件來調校，請參閱調校 Aurora MySQL。
- 如需關於所有 MySQL 等待事件的資訊，請參閱 MySQL 文件中的等待事件摘要表格。
- 如需 Aurora PostgreSQL 的常用等待事件清單，請參閱 Amazon Aurora PostgreSQL 等待事件。若要了解如何使用這些等待事件來調校，請參閱調校 Aurora PostgreSQL 的等待事件。
- 如需關於所有 PostgreSQL 等待事件的資訊，請參閱 PostgreSQL 文件中的統計數字收集器 > 等待事件表。

最高 SQL

等待事件會顯示瓶頸，而最高 SQL 則顯示哪些查詢對資料庫負載造成最大影響。例如，許多查詢目前可能正在資料庫上執行，但單一查詢可能會耗用 99% 的資料庫負載。在此情況下，高負載可能表示查詢發生問題。

根據預設，績效詳情主控台會顯示造成資料庫負載的常用 SQL 查詢。主控台也會顯示每個陳述式的相關統計資料。若要診斷特定陳述式的效能問題，您可以檢查其執行計劃。

最大 CPU

在儀表板中，資料庫負載圖表會收集、彙整並顯示工作階段資訊。若要查看作用中的工作階段是否超過最大 CPU，請查看它與最大 vCPU 數線的關係。Performance Insights 會依據資料庫執行個體的 vCPU (虛擬 CPU) 核心數目決定最大 vCPU 值。針對 Aurora Serverless v2，Max vCPU (vCPU 上限) 表示 vCPU 的估計數量。

一次只能在 vCPU 上執行一個程序。如果程序數目超過 vCPU 數目，則程序會開始排入佇列。佇列增加時，資料庫效能會降低。若資料庫負載通常高於最大 vCPU 數線，而主要等待狀態為 CPU，則 CPU 會超過負載。在此狀況下，您可能會想要節制與執行個體間的連線、以高 CPU 負載來微調任何 SQL 查詢、或者考慮使用較大的執行個體類別。處於任何等待狀態的密集且穩定的執行個體表示可能有您應解決的瓶頸或資源爭用問題。即使資料庫負載未超過最大 vCPU 數線，仍可能會有上述的問題。

Performance Insights 的 Amazon Aurora 資料庫引擎、區域和執行個體類別支援

Important

AWS 已宣布績效詳情的end-of-life：2026 年 6 月 30 日。在此日期之後，Amazon RDS 將不再支援 Performance Insights 主控台體驗、彈性保留期間 (1-24 個月) 及其相關定價。Performance Insights API 將繼續存在，不會有任何定價變更。Performance Insights API 的成本會顯示在您的 AWS 帳單中，其中包含 CloudWatch Database Insights 的成本。
我們建議您在 2026 年 6 月 30 日之前，使用已付費的績效詳情方案，將任何資料庫叢集升級至資料庫詳情的進階模式。如需有關升級至 Database Insights 進階模式的資訊，請參閱 開啟適用於 Amazon Aurora 的 Database Insights 的進階模式。
如果您不採取任何動作，則使用 Performance Insights 的資料庫叢集依預設會使用 Database Insights 的標準模式。使用 Database Insights 的標準模式時，您可能會失去超過 7 天的效能資料歷史記錄存取權，而且可能無法在 Amazon RDS 主控台中使用執行計劃和隨需分析功能。2026 年 6 月 30 日之後，只有進階模式的 Database Insights 將支援執行計畫和隨需分析。
透過 CloudWatch Database Insights，您可以監控資料庫機群的資料庫負載，以及大規模分析效能和進行疑難排解。如需 Database Insights 的詳細資訊，請參閱 使用 CloudWatch Database Insights 監控 Amazon Aurora 資料庫。如需定價資訊，請參閱 Amazon CloudWatch 定價。

下列資料表提供您可找到支援 Performance Insights 的 Amazon Aurora 資料庫引擎。

Amazon Aurora 資料庫引擎	支援的引擎版本和區域	執行個體類別限制
Amazon Aurora MySQL-Compatible Edition	如需 Performance Insights 搭配 Aurora MySQL 的版本和區域可用性的詳細資訊，請參閱 搭配 Aurora MySQL 的 Performance Insights。	績效詳情具有以下引擎類限制： • db.t2 – 不支援 • db.t3 – 不支援 • db.t4g.micro 和 db.t4g.small – 不支援
Amazon Aurora PostgreSQL-Compatible Edition	如需 Performance Insights 搭配 Aurora PostgreSQL 的版本和區域可用性的詳細資訊，請參閱 搭配 Aurora PostgreSQL 的 Performance Insights。	N/A

Performance Insights 功能的 Amazon Aurora 資料庫引擎、區域和執行個體類別支援

下列資料表提供支援 Performance Insights 功能的 Amazon Aurora 資料庫引擎。

功能	定價方案	支援的區域	支援的資料庫引擎	支援的執行個體類別
績效詳情的 SQL 統計數字	全部	全部	全部	全部
分析一段時間區間的資料庫效能	僅限付費方案	全部	全部	全部，但 db.serverless (Aurora Serverless v2) 除外

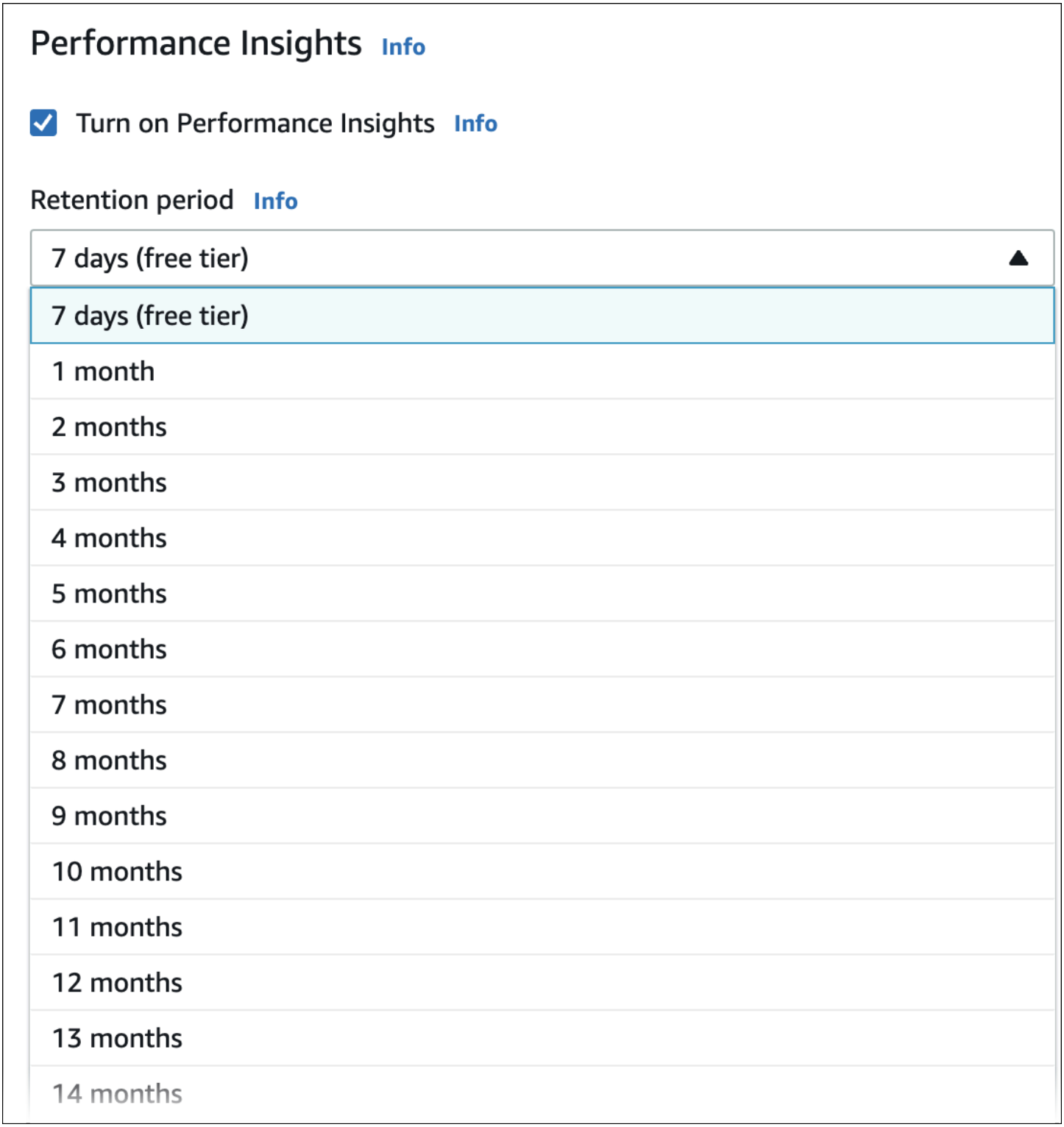
功能	定價方案	支援的區域	支援的資料庫引擎	支援的執行個體類別
檢視 Performance Insights 主動建議	僅限付費方案	• 美國東部 (俄亥俄) • 美國東部 (維吉尼亞北部) • 美國西部 (加利佛尼亞北部) • 美國西部 (奧勒岡) • 亞太區域 (孟買) • 亞太區域 (首爾) • 亞太區域 (新加坡) • 亞太地區 (雪梨) • 亞太區域 (東京) • 加拿大 (中部) • 歐洲 (法蘭克福) • 歐洲 (愛爾蘭) • 歐洲 (倫敦) • 歐洲 (巴黎) • 歐洲 (斯德哥爾摩) • 南美洲 (聖保羅)	全部	全部，但 db.serverless (Aurora Serverless v2) 除外

Performance Insights 的定價和資料保留

根據預設，Performance Insights 包括每月 7 天的效能資料歷程記錄和 1 百萬次 API 請求。您也可以購買較長的保留期間。如需有關費用的資訊，請參閱 Performance Insights 定價。

在 RDS 主控台中，您可以為 Performance Insights 資料選擇下列任一保留期間：

- 預設值 (7 天)
- ***n*** 月，其中 ***n*** 是介於 1 - 24 之間的數字



若要了解如何使用 AWS CLI 設定保留期間，請參閱 開啟和關閉 Aurora 的 Performance Insights。

> **Note**
>
> 停止已啟用 Performance Insights 的資料庫叢集不會影響資料保留。當資料庫叢集停止時，Performance Insights 不會收集任何資料。

開啟和關閉 Aurora 的 Performance Insights

> **Important**
>
> AWS 已宣布績效詳情的end-of-life：2026 年 6 月 30 日。在此日期之後，Amazon RDS 將不再支援 Performance Insights 主控台體驗、彈性保留期間 (1-24 個月) 及其相關定價。Performance Insights API 將繼續存在，不會有任何定價變更。Performance Insights API 的成本會顯示在您的 AWS 帳單中，其中包含 CloudWatch Database Insights 的成本。
> 建議您在 2026 年 6 月 30 日之前，使用已付費的績效詳情方案，將任何資料庫叢集升級至資料庫詳情的進階模式。如需有關升級至 Database Insights 進階模式的資訊，請參閱 開啟適用於 Amazon Aurora 的 Database Insights 的進階模式。
> 如果您不採取任何動作，則使用 Performance Insights 的資料庫叢集依預設會使用 Database Insights 的標準模式。使用 Database Insights 的標準模式時，您可能會失去超過 7 天的效能資料歷史記錄存取權，而且可能無法在 Amazon RDS 主控台中使用執行計畫和隨需分析功能。2026 年 6 月 30 日之後，只有進階模式的 Database Insights 將支援執行計畫和隨需分析。
> 透過 CloudWatch Database Insights，您可以監控資料庫機群的資料庫負載，以及大規模分析效能和進行疑難排解。如需 Database Insights 的詳細資訊，請參閱 使用 CloudWatch Database Insights 監控 Amazon Aurora 資料庫。如需定價資訊，請參閱 Amazon CloudWatch 定價。

您可在建立資料庫叢集時為其開啟 Performance Insights。如有需要，您可以稍後從主控台修改資料庫叢集來將其關閉。開啟和關閉績效詳情不會造成停機、重新開機或容錯移轉。

> **Note**
>
> 效能結構描述為 Aurora MySQL 使用的選用效能工具。若您開啟或關閉「效能架構」，則需要重新開機。然而，如果您開啟或關閉績效詳情，則無需重新開機。如需詳細資訊，請參閱Aurora MySQL 上 Performance Insights 的效能結構描述概觀。

如果您搭配使用 Performance Insights 和 Aurora 全域資料庫，請在每個 AWS 區域中分別為資料庫開啟 Performance Insights。如需詳細資訊，請參閱利用 Amazon RDS 績效詳情來監控 Amazon Aurora 全域資料庫。

績效詳情代理程式會耗用資料庫主機上有限的 CPU 和記憶體。當資料庫負載過高時，代理程式會減少收集資料的頻率來限制效能影響。

Console

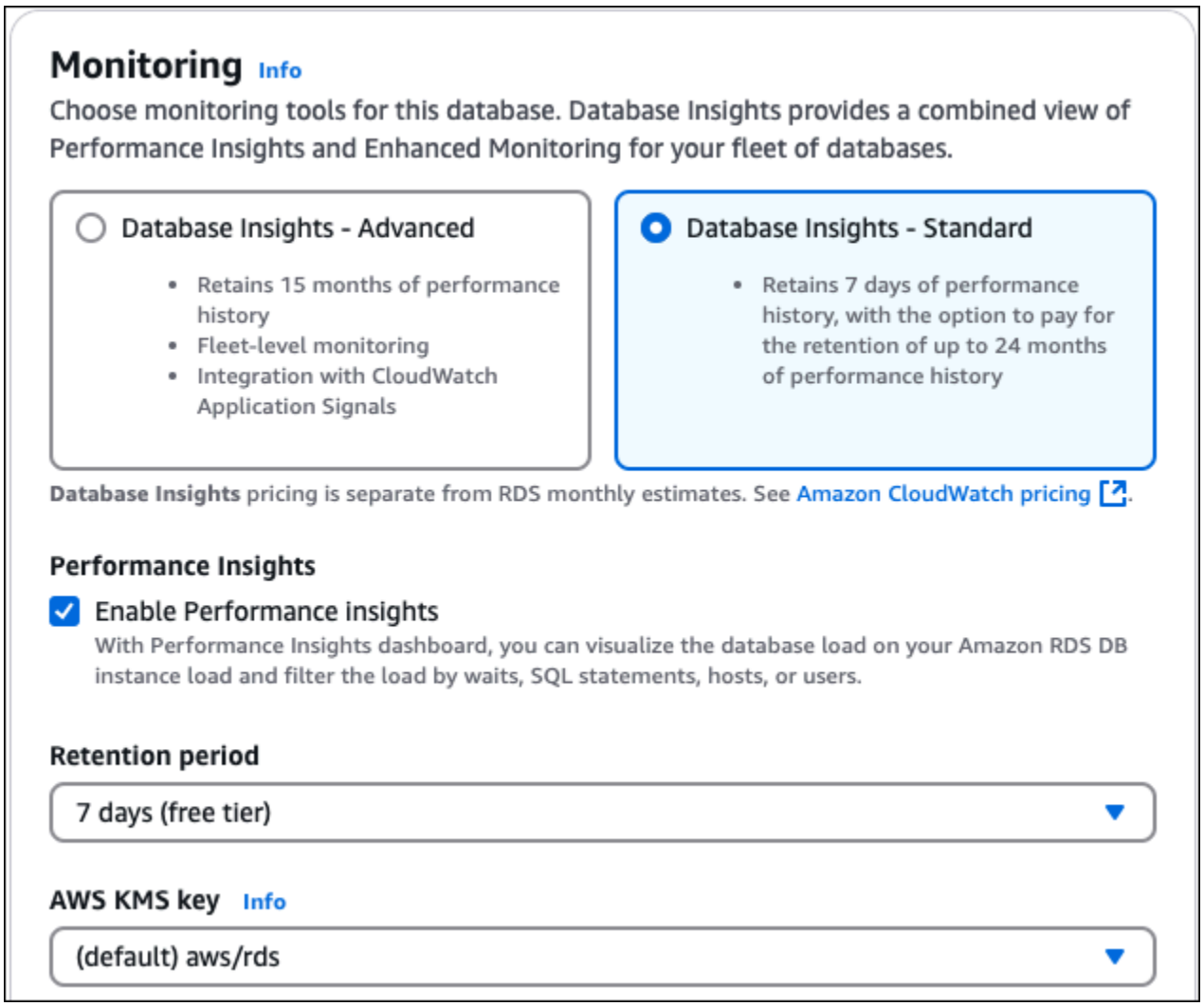
在主控台中，您可以在建立資料庫叢集時啟用或關閉 Performance Insights。啟用 Performance Insights 可讓您管理資料庫叢集的 Performance Insights 設定和選項。叢集層級設定會套用至叢集中的所有資料庫執行個體。

在建立資料庫叢集時，開啟或關閉 Performance Insights

建立新的資料庫叢集之後，Amazon RDS 預設會啟用 Performance Insights。若要關閉資料庫叢集的 Performance Insights，請選擇Database Insights – 標準選項，然後取消選取啟用 Performance Insights 選項。

若要建立資料庫叢集，請遵循 建立 Amazon Aurora 資料庫叢集 中的資料庫引擎指示。

下列螢幕擷取畫面顯示 Performance Insights (績效詳情) 區段。



若您選擇 Enable Performance Insights (啟用績效詳情)，會有下列選項：

- 保留 (僅適用於 Database Insights 的標準模式) – 保留 Performance Insights 資料的時間長度。保留設定為預設值 (7 天)。若要更長時間保留績效資料，請指定 1 - 24 個月。如需保留期間的詳細資訊，請參閱 Performance Insights 的定價和資料保留。
- AWS KMS key – 指定您的 AWS KMS key。績效詳情使用您的 KMS 金鑰來加密所有可能的敏感資料。將會對傳輸中與靜態資料進行加密。如需詳細資訊，請參閱變更績效詳情 AWS KMS 的政策。

在修改資料庫叢集時，開啟或關閉 Performance Insights

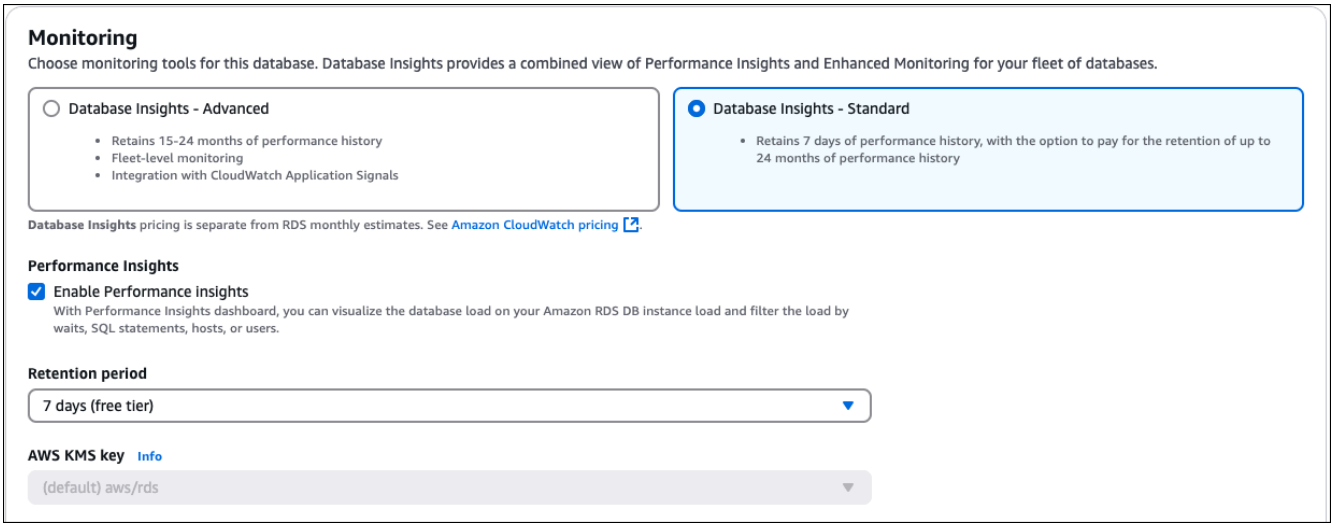
在主控台中，您可以修改資料庫叢集，以管理 Performance Insights。

使用主控台為資料庫叢集管理 Performance Insights

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 選擇 Databases (資料庫)。
3. 選擇資料庫叢集，然後選擇修改。
4. 若要開啟 Performance Insights，請選取啟用 Performance Insights。若要關閉資料庫叢集的 Performance Insights，請選擇Database Insights – 標準選項，然後取消選取啟用 Performance Insights 選項。

   若您選擇 Enable Performance Insights (啟用績效詳情)，會有下列選項：

   - 保留 (僅適用於 Database Insights 的標準模式) – 保留 Performance Insights 資料的時間長度。保留設定為預設值 (7 天)。若要更長時間保留績效資料，請指定 1 - 24 個月。如需保留期間的詳細資訊，請參閱 Performance Insights 的定價和資料保留。
   - AWS KMS key – 指定您的 KMS 金鑰。績效詳情使用您的 KMS 金鑰來加密所有可能的敏感資料。將會對傳輸中與靜態資料進行加密。如需詳細資訊，請參閱加密 Amazon Aurora 資源。



5. 選擇 Continue (繼續)。
6. 在 Scheduling of Modifications (修改排程) 中，選擇 Apply immediately (立即套用)。如果您選擇 Apply during the next scheduled maintenance window (在下一個排定的維護時段套用)，則執行個體會忽略此設定，並立即開啟 Performance Insights。
7. 選擇 Modify instance (修改執行個體)。

AWS CLI

當您使用 create-db-instance AWS CLI 命令時，請指定 `--enable-performance-insights`並`--database-insights-mode`設定為 `advanced`或 以開啟績效詳情`standard`。若要關閉 Performance Insights，請指定 `--no-enable-performance-insights` 並將 `database-insights-mode` 設定為 `standard`。

您也可以使用下列 AWS CLI 命令來指定這些值：

- create-db-cluster
- modify-db-cluster
- create-db-instance-read-replica
- modify-db-instance
- restore-db-instance-from-s3

使用 管理資料庫叢集的績效詳情 AWS CLI

- 呼叫 modify-db-cluster AWS CLI 命令並提供下列值：
  - `--db-cluster-identifier` – 資料庫叢集中的資料庫執行個體名稱。
  - `--enable-performance-insights` 用來開啟，`--no-enable-performance-insights` 用來關閉
  - `database-insights-mode` – 資料庫叢集的 Database Insights 模式。若要關閉 Performance Insights，請將此值設定為 `standard`。

  以下範例會開啟 `sample-db-cluster` 的 Performance Insights。

  針對 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
 --database-insights-mode standard \
    --db-cluster-identifier sample-db-instance \
    --enable-performance-insights
```

  在 Windows 中：

```
aws rds modify-db-cluster ^
 --database-insights-mode standard ^
```

```
    --db-cluster-identifier sample-db-instance ^
    --enable-performance-insights
```

當您開啟績效詳情時，可選擇使用 `--performance-insights-retention-period` 選項指定保留績效詳情資料的天數。您可指定為 7、*#* * 31 (其中*#*是指 1 - 23) 的數字，或 731。例如，如果您想要保留效能資料 3 個月，請指定 93，也就是 3 個 * 31。預設值是 7 天。如需保留期間的詳細資訊，請參閱 Performance Insights 的定價和資料保留。

下列範例會開啟 `sample-db-cluster` 的 Performance Insights，並指定 Performance Insights 資料保留 93 天 (3 個月)。

針對 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
 --database-insights-mode standard \
    --db-cluster-identifier sample-db-instance \
    --enable-performance-insights \
    --performance-insights-retention-period 93
```

在 Windows 中：

```
aws rds modify-db-cluster ^
 --database-insights-mode standard ^
    --db-cluster-identifier sample-db-instance ^
    --enable-performance-insights ^
    --performance-insights-retention-period 93
```

如果您指定保留期間 (例如 94 天)，這不是有效值，RDS 就會發出錯誤。

```
An error occurred (InvalidParameterValue) when calling the CreateDBInstance
 operation:
Invalid Performance Insights retention period. Valid values are: [7, 31, 62, 93,
 124, 155, 186, 217,
248, 279, 310, 341, 372, 403, 434, 465, 496, 527, 558, 589, 620, 651, 682, 713, 731]
```

> **Note**
>
> 您只能在 Performance Insights 不在叢集層級進行管理的資料庫叢集中，切換執行個體的 Performance Insights。

RDS API

當您使用 CreateDBInstance 作業 Amazon RDS API 作業來建立資料庫叢集中新的資料庫執行個體時，請將 `EnablePerformanceInsights` 設為 `True` 以開啟績效詳情。若要關閉 Performance Insights，請將 `EnablePerformanceInsights` 設定為 `False`，以及將 `DatabaseInsightsMode` 設定為 `standard`。

您也可以使用下列 API 操作來指定 `EnablePerformanceInsights` 值：

- CreateDBCluster
- ModifyDBCluster
- ModifyDBInstance
- CreateDBInstanceReadReplica
- RestoreDBInstanceFromS3

當您開啟 Performance Insights 時，可以使用 `PerformanceInsightsRetentionPeriod` 參數來選擇保留 Performance Insights 資料的時間 (以天數為單位)。您可指定為 7、# * 31 (其中#是指 1 - 23) 的數字，或 731。例如，如果您想要保留效能資料 3 個月，請指定 93，也就是 3 個 * 31。預設值是 7 天。如需保留期間的詳細資訊，請參閱 Performance Insights 的定價和資料保留。

Aurora MySQL 上 Performance Insights 的效能結構描述概觀

效能結構描述為選用功能，用來在詳細資料低層級監控 Aurora MySQL 執行時間效能。效能結構描述專為盡量降低對資料庫效能的影響所設計。您在有或無效能結構描述的狀況下皆可使用績效詳情。

主題

- 效能結構描述概觀
- Performance Insights 和 Performance Schema
- 透過績效詳情自動管理效能結構描述
- 重新開機對效能結構描述的影響
- 決定績效詳情是否管理效能結構描述
- 在 Aurora MySQL 上開啟效能結構描述

效能結構描述概觀

效能結構描述會監控 Aurora MySQL資料庫 中的事件。事件是個佔用時間的資料庫伺服器動作，並經過分析，可收集計時資訊。事件的範例如下：

- 函數呼叫
- 等待作業系統
- SQL 執行的階段
- SQL 陳述式的群組

PERFORMANCE_SCHEMA 儲存引擎是一種用於實作效能結構描述功能的機制。此引擎會使用資料庫來源碼中的檢測來收集事件資料。引擎會將事件儲存於 performance_schema 資料庫的僅限記憶體表格中。您可查詢 performance_schema，就像您可以查詢任何其他表格一樣。如需 MySQL 效能結構描述的詳細資訊，請參閱 MySQL 參考手冊中的 MySQL 效能結構描述。

Performance Insights 和 Performance Schema

績效詳情和效能結構描述是獨立的功能，但彼此互相連接。Performance Insights 對於 Aurora MySQL 的行為取決於 Performance Schema 是否開啟，如果是的話，Performance Insights 是否自動管理 Performance Schema。下表說明行為。

Performance Schema 已開啟	Performance Insights 管理模式	Performance Insights 行為
是	自動	• 收集詳細的低階監控資訊 • 收集每秒作用中工作階段指標 • 顯示依詳細等待事件分類的資料庫負載，您可以用來識別瓶頸
是	手動	• 收集等待事件和每個 SQL 指標 • 收集每秒作用中工作階段指標 •

Performance Schema 已開啟	Performance Insights 管理模式	Performance Insights 行為
		報告使用者狀態，例如插入和傳送，無法協助您識別瓶頸
否	N/A	• 不收集等待事件、每個 SQL 指標或其他詳細的低階監控資訊 • 收集每五秒 (而不是每秒) 作用中工作階段指標 • 報告使用者狀態，例如插入和傳送，無法協助您識別瓶頸

透過績效詳情自動管理效能結構描述

當您在開啟績效詳情的狀況下建立 Aurora MySQL 資料庫執行個體時，也會開啟效能結構描述。在此狀況下，績效詳情會自動管理您的效能結構描述參數。這是建議的組態。

當 Performance Insights 自動管理效能結構描述時，`performance_schema` 的來源為 `System default`。

> **Note**
>
> t4g.medium 執行個體類別不支援效能結構描述的自動管理。

您也可以手動管理效能結構描述。如果選擇此選項，請根據下表中的值設定參數。

參數名稱	參數值
`performance_schema`	1 (Source (來源) 欄具有值 `Modified`)
`performance-schema-consumer-events-waits-current`	`ON`
`performance-schema-instrument`	`wait/%=ON`

參數名稱	參數值
performance_schema_consumer_global_instrumentation	1
performance_schema_consumer_thread_instrumentation	1

若您手動變更 performance_schema 參數，但稍後想變更為自動管理，請參閱在 Aurora MySQL 上開啟效能結構描述。

> **Important**
>
> 當績效詳情開啟效能結構描述時，其不會變更參數群組值。不過，這些值會在執行的資料庫執行個體上進行變更。查看變更值的唯一方法是執行 SHOW GLOBAL VARIABLES 命令。

重新開機對效能結構描述的影響

績效詳情和效能結構描述在資料庫執行個體重新開機的要求方面有所不同：

效能結構描述

若要開啟或關閉此功能，您必須重新開機資料庫執行個體。

Performance Insights

若要開啟或關閉此功能，不需要重新開機資料庫執行個體。

如果效能結構描述目前未開啟，並且您在未重新開機資料庫執行個體的情況下開啟績效詳情，則不會開啟效能結構描述。

決定績效詳情是否管理效能結構描述

若要了解 Performance Insights 目前是否正在管理所有受支援主要引擎版本的效能結構描述，請查看下表。

performance_schema 參數的設定	來源欄的設定	績效詳情管理效能結構描述
0	System default	是
0 或 1	Modified	否

在下列程序中，您會判斷 Performance Insights 是否自動管理效能結構描述。

若要判斷績效詳情是否自動管理效能結構描述

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 選擇 Parameter groups (參數群組)。
3. 選擇資料庫執行個體的參數群組名稱。
4. 在搜尋列中，輸入 **performance_schema**。
5. 檢查來源是否為系統預設值，而值是否為 0。若是如此，績效詳情正在自動管理效能結構描述。

   在這裡顯示的範例中，Performance Insights 並未自動管理效能結構描述。

在 Aurora MySQL 上開啟效能結構描述

假設您的資料庫執行個體的績效詳情已啟用，但目前並未管理效能結構描述。如果要允許績效詳情自動管理效能結構描述，請完成以下步驟。

設定效能結構描述為自動管理

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 選擇 Parameter groups (參數群組)。
3. 選擇資料庫執行個體的參數群組名稱。
4. 選擇編輯。

5. 在搜尋列中，輸入 **performance_schema**。
6. 選取 performance_schema 參數。
7. 選擇設定為預設值。
8. 選擇將值設定為預設值來確認。
9. 選擇 Save Changes (儲存變更)。
10. 重新啟動資料庫執行個體。

> **Important**
>
> 每當您開啟或關閉效能結構描述時，請務必重新開機資料庫執行個體。

如需修改執行個體參數的相關資訊，請參閱 修改 Amazon Aurora 中的資料庫參數群組中的參數。如需儀表板中的詳細資訊，請參閱使用績效詳情儀表板來分析指標。如需 MySQL 效能結構描述的詳細資訊，請參閱 MySQL 文件中的 MySQL 效能結構描述 (適用於 8.0) 和 MySQL 效能結構描述 (適用於 8.4)。

設定績效詳情的存取政策

若要存取績效詳情，委託人必須具有來自 AWS Identity and Access Management (IAM) 的適當許可。

> **Note**
>
> 若要搭配客戶受管金鑰使用績效詳情，請授予使用者 AWS AWS KMS 金鑰的 kms:Decrypt和 kms:GenerateDataKey許可。

使用以下方法存取績效詳情：

- 連接 AmazonRDSPerformanceInsightsReadOnly 受管政策以進行唯讀存取
- 連接AmazonRDSPerformanceInsightsFullAccess受管政策以存取 Performance Insights API 的所有操作
- 建立具有特定許可的自訂 IAM 政策
- 設定加密績效詳情資料的 AWS KMS 許可
- 使用資源層級許可設定精細存取
- 使用標籤型存取控制，透過資源標籤管理許可

將 AmazonRDSPerformanceInsightsReadOnly 政策連接至 IAM 主體

AmazonRDSPerformanceInsightsReadOnly 是一種 AWS 受管政策，可授予 Amazon RDS Performance Insights API 所有唯讀操作的存取權。

如果您連接到AmazonRDSPerformanceInsightsReadOnly許可集或角色，您還必須連接下列 CloudWatch 許可：

- GetMetricStatistics
- ListMetrics
- GetMetricData

透過這些許可，收件人可以使用績效詳情搭配其他主控台功能。

如需 CloudWatch 許可的詳細資訊，請參閱 Amazon CloudWatch 許可參考。

如需有關 AmazonRDSPerformanceInsightsReadOnly 的詳細資訊，請參閱 AWS 受管政策：AmazonRDSPerformanceInsightsReadOnly。

將 AmazonRDSPerformanceInsightsFullAccess 政策連接至 IAM 主體

AmazonRDSPerformanceInsightsFullAccess 是一種 AWS 受管政策，可授予 Amazon RDS Performance Insights API 所有操作的存取權。

如果您連接到AmazonRDSPerformanceInsightsFullAccess許可集或角色，您還必須連接下列 CloudWatch 許可：

- GetMetricStatistics
- ListMetrics
- GetMetricData

透過這些許可，收件人可以使用績效詳情搭配其他主控台功能。

如需 CloudWatch 許可的詳細資訊，請參閱 Amazon CloudWatch 許可參考。

如需有關 AmazonRDSPerformanceInsightsFullAccess 的詳細資訊，請參閱 AWS 受管政策：AmazonRDSPerformanceInsightsFullAccess。

建立績效詳情的自訂 IAM 政策

對於沒有 `AmazonRDSPerformanceInsightsReadOnly` 或 `AmazonRDSPerformanceInsightsFullAccess` 政策提供的使用者，您可以建立或修改由使用者管理的 IAM 政策來授予 Performance Insights 的存取權。在將該政策連接至許可集或角色時，收件人可以使用績效詳情。

建立自訂政策

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在導覽窗格中，選擇政策。
3. 選擇 Create policy (建立政策)。
4. 在建立政策頁面上，選擇 JSON 選項。
5. 針對 AmazonRDSPerformanceInsightsReadOnly 或 AmazonRDSPerformanceInsightsFullAccess 政策，複製並貼上在《AWS 受管政策參考指南》 中的 JSON 政策文件區段中提供的文字。
6. 選擇 Review policy (檢閱政策)。
7. 為政策提供名稱並選擇性輸入描述，然後選擇 Create policy (建立政策)。

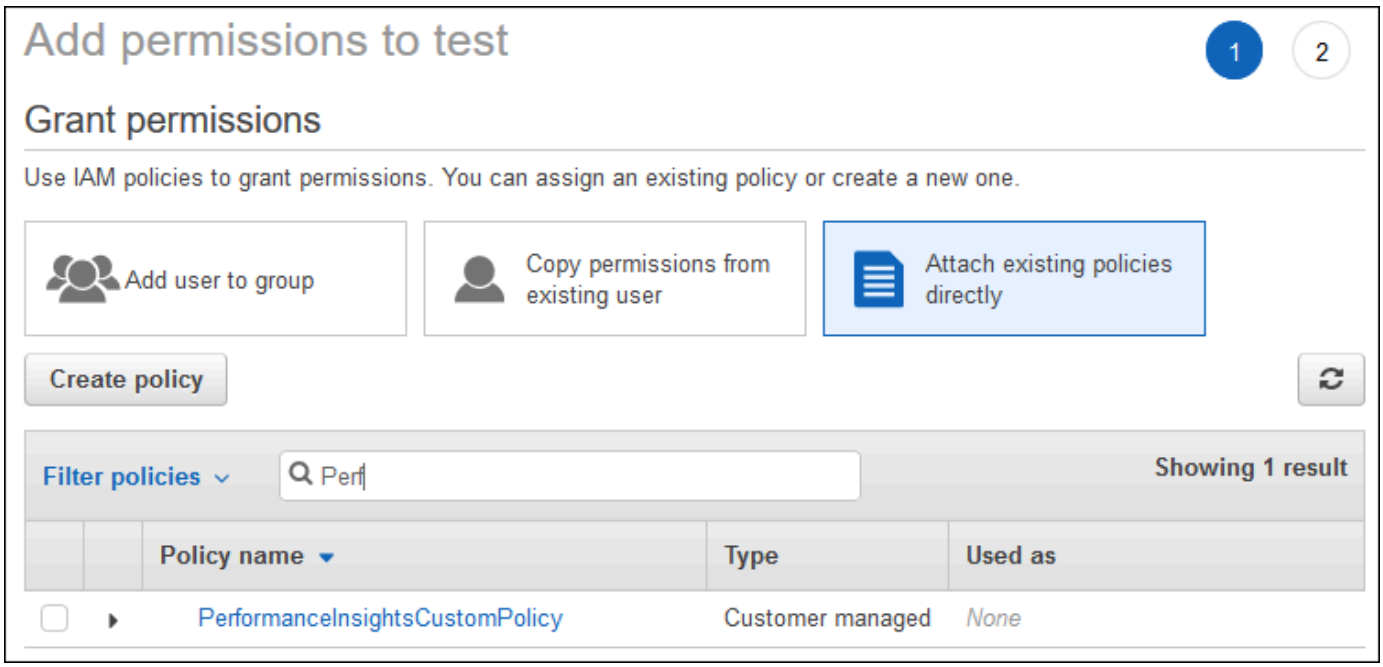
現在您可以將政策連接到許可集或角色。以下程序假設您已有基於此用途使用的使用者。

將政策連接至使用者

1. 在以下網址開啟 IAM 主控台：https://console.aws.amazon.com/iam/。
2. 在導覽窗格中，選擇 Users (使用者)。
3. 從清單中選擇現有的使用者。

> **Important**
>
> 若要使用績效詳情，除了自訂政策以外，請務必確認您還擁有存取 Amazon RDS 的權限。例如，`AmazonRDSPerformanceInsightsReadOnly` 預先定義政策提供對 Amazon RDS 的唯讀存取。如需更多詳細資訊，請參閱 使用政策管理存取權。

4. 在 Summary (摘要) 頁面上，選擇 Add permissions (新增許可)。
5. 選擇 Attach existing policies directly (直接連接現有政策)。對於搜尋，請輸入政策名稱的前幾個字母，如下圖所示。



6. 選擇您的政策，然後選擇 Next: Review (下一步：檢閱)。
7. 選擇 Add permissions (新增許可)。

變更績效詳情 AWS KMS 的政策

Performance Insights 使用 AWS KMS key 來加密敏感資料。當您透過 API 或主控台啟用績效詳情時，可以執行下列任一操作：

- 選擇預設值 AWS 受管金鑰。

  Amazon RDS 會將 AWS 受管金鑰 用於您的新資料庫執行個體。Amazon RDS 會為您的 AWS 帳戶建立 AWS 受管金鑰 。每個 AWS 受管金鑰 的 Amazon RDS AWS 帳戶 都有不同的 AWS 區域。
- 選擇客戶受管金鑰。

  如果您指定客戶受管金鑰，則您帳戶中呼叫績效詳情 API 的使用者需要 KMS 金鑰的 `kms:Decrypt` 和 `kms:GenerateDataKey` 許可。您可以透過 IAM 政策設定這些許可。不過，我們建議您透過 KMS 金鑰政策來管理這些許可。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的在 AWS KMS中使用金鑰政策。

Example

下列範例說明如何將陳述式新增至 KMS 金鑰政策。這些陳述式允許存取績效詳情。視您使用 KMS 金鑰的方式而定，您可能想要變更某些限制。在將陳述式新增至您的政策之前，請先移除所有註解。

JSON

```
{
    "Version":"2012-10-17",
    "Id" : "your-policy",
    "Statement" : [
        {
            "Sid" : "AllowViewingRDSPerformanceInsights",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::444455556666:role/Role1"
                ]
                },
             "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
                ],
            "Resource": "*",
            "Condition" : {
            "StringEquals" : {
                "kms:ViaService" : "rds.us-east-1.amazonaws.com"
                },
            "ForAnyValue:StringEquals": {
                "kms:EncryptionContext:aws:pi:service": "rds",
                "kms:EncryptionContext:service": "pi",
                "kms:EncryptionContext:aws:rds:db-id": "db-
AAAAABBBBBCCCCDDDDDEEEEE"
                }
            }
        }
    ]
}
```

Performance Insights 如何使用 AWS KMS 客戶受管金鑰

Performance Insights 使用客戶壽館金鑰來加密敏感資料。當您開啟 Performance Insights 時，可以透過 API 提供 AWS KMS 金鑰。Performance Insights 會建立此金鑰的 AWS KMS 許可。它會使用金鑰並執行必要的操作來處理敏感資料。敏感資料包括使用者、資料庫、應用程式及 SQL 查詢文字等欄位。Performance Insights 可確保靜態資料和傳輸中資料保持加密狀態。

Performance Insights IAM 如何使用 AWS KMS

IAM 會提供許可給特定 API。Performance Insights 具有下列公有 API，您可以使用 IAM 政策來限制這些 API：

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetadata`
- `GetResourceMetrics`
- `ListAvailableResourceDimensions`
- `ListAvailableResourceMetrics`

您可以使用下列 API 請求來取得敏感資料。

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetrics`

當您使用 API 取得敏感資料時，Performance Insights 會利用呼叫者的憑證。此檢查可確保敏感資料的存取權限於可存取 KMS 金鑰的人。

呼叫這些 APIs時，您需要透過 IAM 政策呼叫 API 的許可，以及透過 AWS KMS 金鑰政策叫用 `kms:decrypt`動作的許可。

`GetResourceMetrics` API 可能會同時傳回敏感和非敏感資料。請求參數會決定回應是否應包含敏感資料。當請求的 filter 或 group-by 參數中包含敏感維度時，API 會傳回敏感資料。

如需有關可搭配 `GetResourceMetrics` API 使用之維度的詳細資訊，請參閱 DimensionGroup。

Example範例

以下範例會請求 `db.user` 群組的敏感資料：

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
Content-Type: application/x-amz-json-1.1
```

```
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
  "ServiceType": "RDS",
  "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
  "MetricQueries": [
    {
      "Metric": "db.load.avg",
      "GroupBy": {
        "Group": "db.user",
        "Limit": 2
      }
    }
  ],
  "StartTime": 1693872000,
  "EndTime": 1694044800,
  "PeriodInSeconds": 86400
}
```

Example

以下範例會請求 db.load.avg 指標的非敏感資料：

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
Content-Type: application/x-amz-json-1.1
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "MetricQueries": [
        {
            "Metric": "db.load.avg"
```

```
        }
    ],
    "StartTime": 1693872000,
    "EndTime": 1694044800,
    "PeriodInSeconds": 86400
}
```

授予 Performance Insights 的精細存取

精細存取控制可提供額外的方式，讓您控制對您 Performance Insights 的存取。此存取控制可以允許或拒絕存取 GetResourceMetrics、DescribeDimensionKeys 和 GetDimensionKeyDetails Performance Insights 動作的個別維度。若要使用精細存取，請使用條件索引鍵在 IAM 政策中指定維度。存取的評估遵循 IAM 政策評估邏輯。如需詳細資訊，請參閱 IAM User Guide 中的 Policy evaluation logic。如果 IAM 政策陳述式未指定任何維度，則陳述式會控制對指定動作之所有維度的存取。如需可用維度的清單，請參閱 DimensionGroup。

若要了解您的憑證獲授權可存取的維度，請使用 ListAvailableResourceDimensions 中的 AuthorizedActions 參數並指定動作。AuthorizedActions 的允許值如下所示：

- GetResourceMetrics
- DescribeDimensionKeys
- GetDimensionKeyDetails

例如，如果您將 GetResourceMetrics 指定為 AuthorizedActions 參數，則 ListAvailableResourceDimensions 會傳回 GetResourceMetrics 動作獲授權可存取的維度清單。如果您在 AuthorizedActions 參數中指定多個動作，則 ListAvailableResourceDimensions 會傳回這些動作獲授權可存取的維度交集。

Example

下列範例提供 GetResourceMetrics 和 DescribeDimensionKeys 動作指定維度的存取權。

JSON

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

```
            "Sid": "AllowToDiscoverDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:ListAvailableResourceDimensions"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
        },
        {
            "Sid": "SingleAllow",
            "Effect": "Allow",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "pi:Dimensions": [
                        "db.sql_tokenized.id",
                        "db.sql_tokenized.statement"
                    ]
                }
            }
        }


    ]
}
```

以下是所請求維度的回應：

```
 // ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
```

```
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["DescribeDimensionKeys"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                  //  { "Identifier": "db.sql_tokenized.db_id" }, // not included
 because not allows in the IAM Policy
                    { "Identifier": "db.sql_tokenized.statement" }
                ]
            }

        ] }
    ]
}
```

下列範例會指定維度的一個允許和兩個拒絕存取。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
          {
            "Sid": "AllowToDiscoverDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:ListAvailableResourceDimensions"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
          },
```

```
          {
            "Sid": "O01AllowAllWithoutSpecifyingDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
        },

        {
            "Sid": "O01DenyAppDimensionForAll",
            "Effect": "Deny",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "pi:Dimensions": [
                        "db.application.name"
                    ]
                }
            }
        },

        {
            "Sid": "O01DenySQLForGetResourceMetrics",
            "Effect": "Deny",
            "Action": [
                "pi:GetResourceMetrics"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
```

```
                "ForAnyValue:StringEquals": {
                    "pi:Dimensions": [
                        "db.sql_tokenized.statement"
                    ]
                }
            }
        }
    ]
}
```

以下是所請求維度的回應：

```
   // ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["GetResourceMetrics"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.application",
                "Dimensions": [

                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.application.name" }
                ]
            },
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                    { "Identifier": "db.sql_tokenized.db_id" },
```

```
                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.sql_tokenized.statement" }
                ]
            },
            ...
        ] }
    ]
}
```

```
// ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["DescribeDimensionKeys"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.application",
                "Dimensions": [
                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.application.name" }
                ]
            },
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                    { "Identifier": "db.sql_tokenized.db_id" },

                  // allowed for DescribeDimensionKeys because our IAM Policy
                  // denies it only for GetResourceMetrics
                    { "Identifier": "db.sql_tokenized.statement" }
                ]
            },
            ...
```

```
        ] }
    ]
}
```

使用績效詳情的標籤型存取控制

您可以使用從父資料庫執行個體繼承的標籤來控制對績效詳情指標的存取。若要控制對績效詳情操作的存取，請使用 IAM 政策。這些政策可以檢查資料庫執行個體上的標籤，以判斷許可。

標籤如何與績效詳情搭配使用

Performance Insights 會自動套用資料庫執行個體標籤，以授權 Performance Insights 指標。當您將標籤新增至資料庫執行個體時，您可以立即使用這些標籤來控制對 Performance Insights 資料的存取。

- 若要新增或更新 Performance Insights 指標的標籤，請修改資料庫執行個體上的標籤。
- 若要檢視績效詳情指標的標籤，請在績效詳情指標資源`ListTagsForResource`上呼叫 。它會從與指標相關聯的資料庫執行個體傳回標籤。

> **Note**
>
> 如果您嘗試直接在績效詳情指標上使用它們， `TagResource`和 `UntagResource`操作會傳回錯誤。

建立標籤型 IAM 政策

若要控制對 Performance Insights 操作的存取，請在 IAM 政策中使用 `aws:ResourceTag`條件索引鍵。這些政策會檢查yourDB執行個體上的標籤。

Example

此政策可防止存取生產資料庫的績效詳情指標。此政策會針對任何以 標記的資料庫資源，拒絕 Performance Insights 中的 `pi:GetResourceMetrics`操作`env:prod`。

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

```
            "Action": "pi:GetResourceMetrics",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/env": "prod"
                }
            }
        }
    ]
}
```

使用績效詳情儀表板來分析指標

Important

AWS 已宣布績效詳情的end-of-life：2026 年 6 月 30 日。在此日期之後，Amazon RDS 將不再支援 Performance Insights 主控台體驗、彈性保留期間 (1-24 個月) 及其相關定價。Performance Insights API 將繼續存在，不會有任何定價變更。Performance Insights API 的成本會顯示在您的 AWS 帳單中，其中包含 CloudWatch Database Insights 的成本。

建議您在 2026 年 6 月 30 日之前，使用已付費的績效詳情方案，將任何資料庫叢集升級至資料庫詳情的進階模式。如需有關升級至 Database Insights 進階模式的資訊，請參閱 開啟適用於 Amazon Aurora 的 Database Insights 的進階模式。

如果您不採取任何動作，則使用 Performance Insights 的資料庫叢集依預設會使用 Database Insights 的標準模式。使用 Database Insights 的標準模式時，您可能會失去超過 7 天的效能資料歷史記錄存取權，而且可能無法在 Amazon RDS 主控台中使用執行計畫和隨需分析功能。2026 年 6 月 30 日之後，只有進階模式的 Database Insights 將支援執行計畫和隨需分析。

透過 CloudWatch Database Insights，您可以監控資料庫機群的資料庫負載，以及大規模分析效能和進行疑難排解。如需 Database Insights 的詳細資訊，請參閱 使用 CloudWatch Database Insights 監控 Amazon Aurora 資料庫。如需定價資訊，請參閱 Amazon CloudWatch 定價。

績效詳情儀表板包含資料庫效能資訊，可協助您分析效能問題並對其進行故障排除。在主要儀表板頁面上，您可以檢視關於資料庫負載的資訊。您可以依據等待事件或 SQL 等維度「配量」資料庫負載。

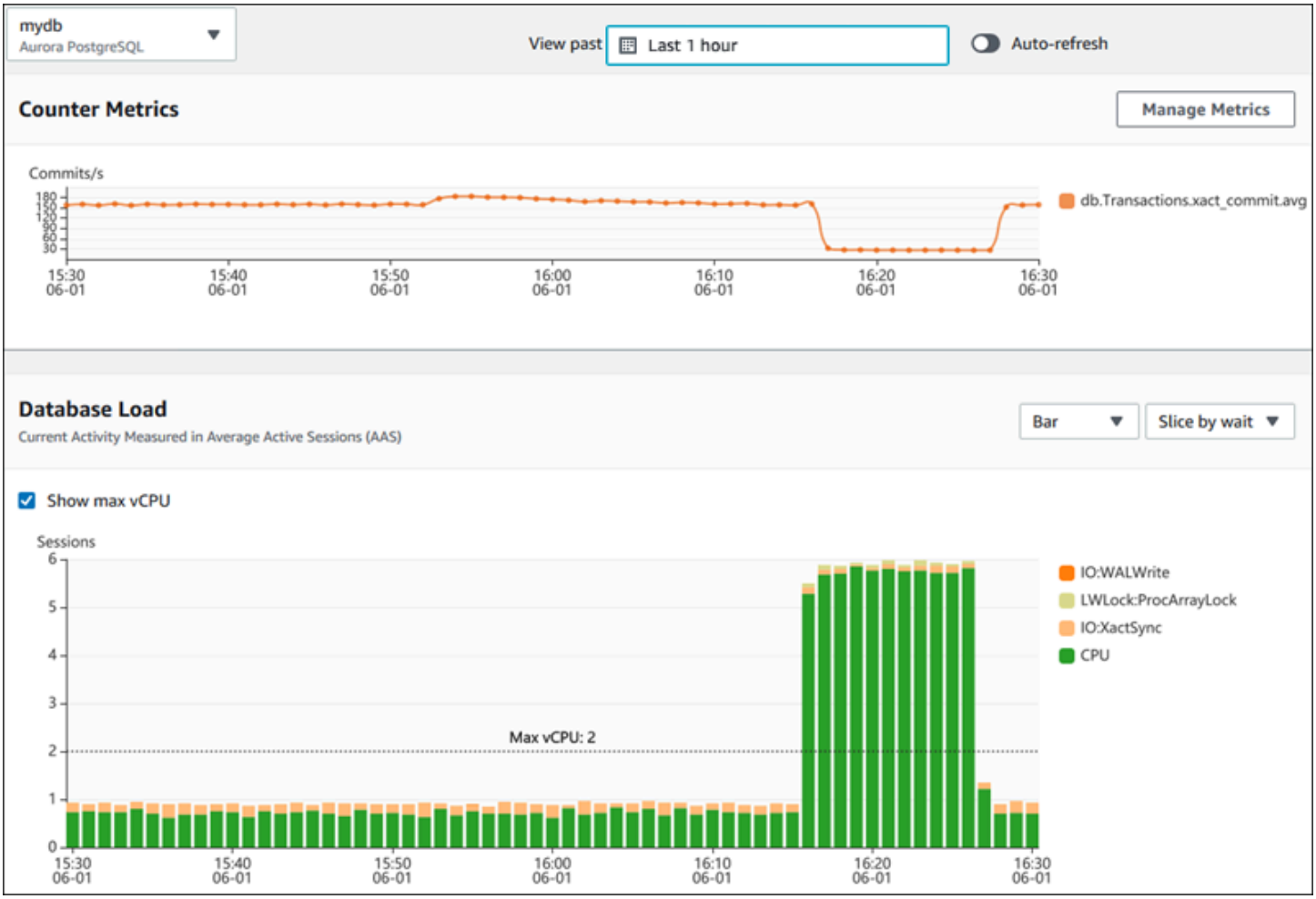
績效詳情儀表板

- 績效詳情儀表板概觀

- 存取績效詳情儀表板
- 按等待事件分析資料庫負載
- 分析一段時間區間的資料庫效能
- 使用 Performance Insights 中的最高 SQL 索引標籤分析查詢

績效詳情儀表板概觀

儀表板是與績效詳情進行互動的最簡單方式。下列範例顯示 PostgreSQL 資料庫執行個體的儀表板。



主題

- 時間範圍篩選
- 計數器指標圖表
- 資料庫負載圖表
- 最高維度表格

時間範圍篩選

依預設，績效詳情儀表板會顯示過去一小時的資料庫負載。您可以將此範圍調整為短至 5 分鐘或長達 2 年。您也可以選取自訂相對範圍。

您可以選擇具有開始和結束日期時間的絕對範圍。以下範例顯示從 24/9/25 午夜開始到 24/9/28 晚上 11:59 結束的時間範圍。

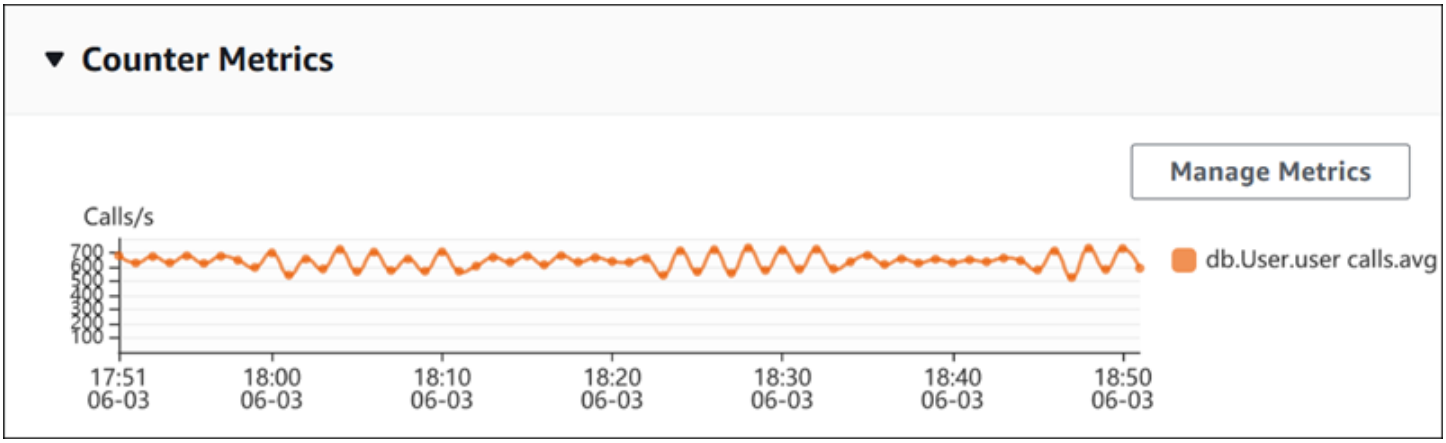
根據預設，Performance Insights 儀表板的時區是國際標準時間 (UTC)。您也可以選擇當地時區。

計數器指標圖表

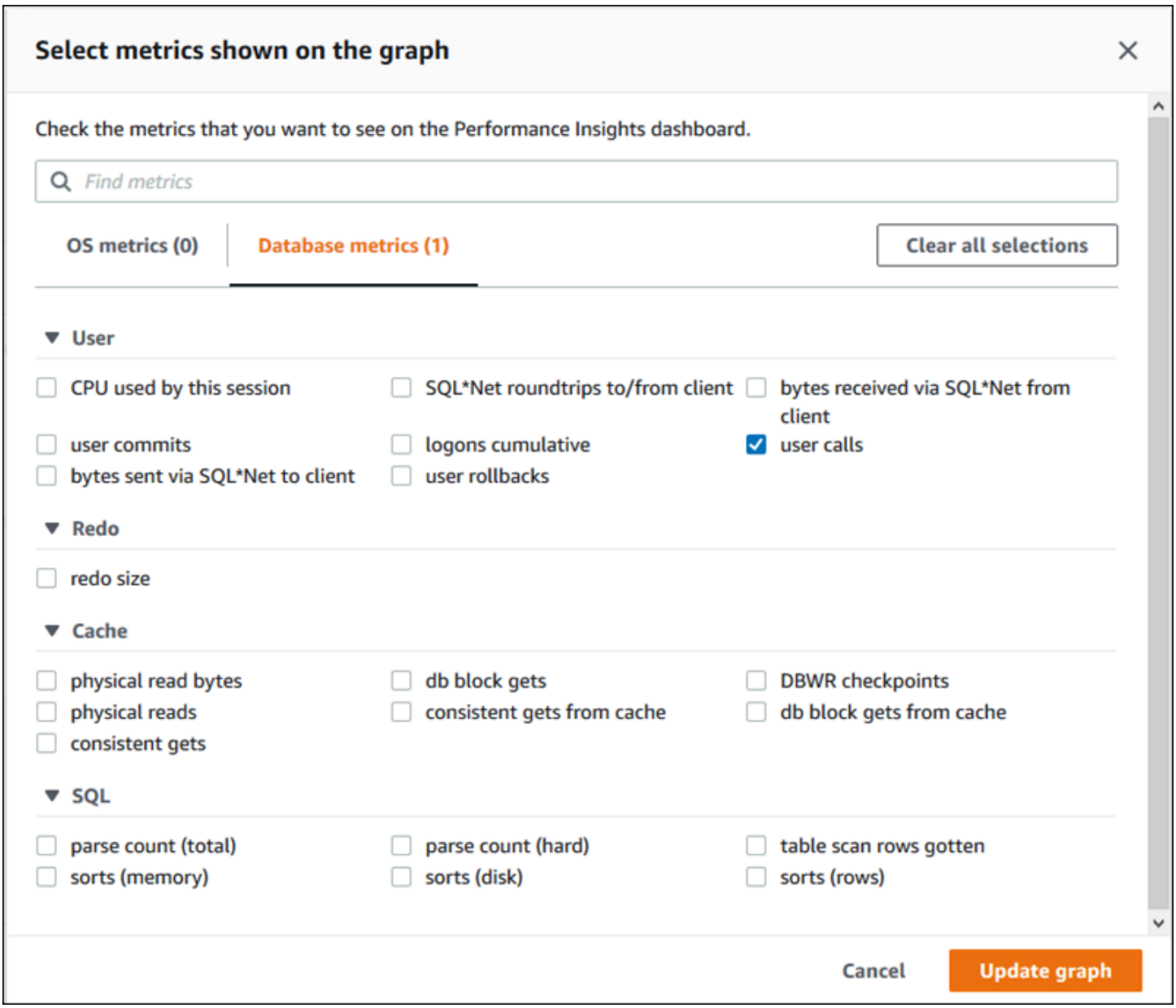
您可以使用計數器指標來自訂績效詳情儀表板，以包含高達 10 個其它圖表。這些圖表顯示很多作業系統和資料庫效能指標。您可以將此資訊與資料庫負載相互關聯，以協助識別和分析效能問題。

Counter Metrics (計數器指標) 圖表顯示效能計數器的資料。預設指標取決於資料庫引擎：

- Aurora MySQL – `db.SQL.Innodb_rows_read.avg`
- Aurora PostgreSQL：`db.Transactions.xact_commit.avg`



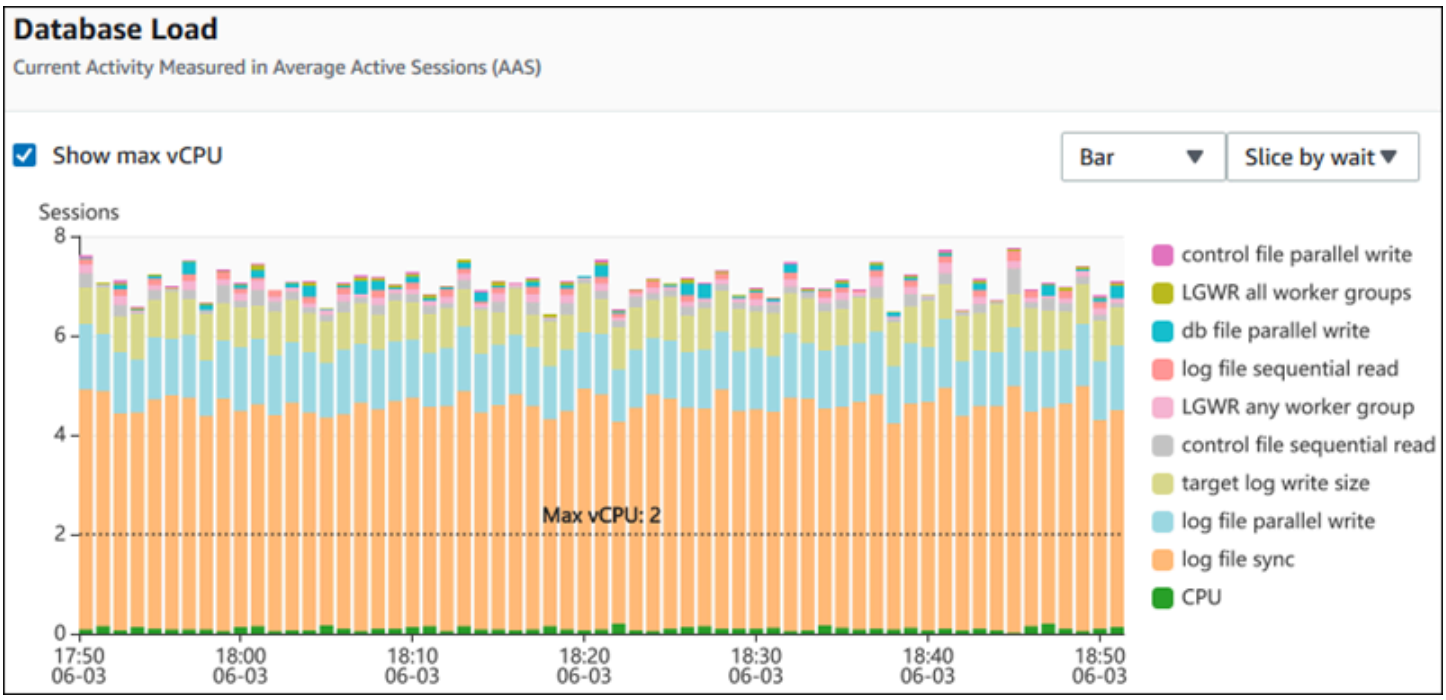
若要變更效能計數器，請選擇 Manage Metrics (管理指標)。您可以選取多個作業系統指標或資料庫指標，如下列螢幕擷取畫面所示。若要查看任何指標的詳細資訊，請將游標移到指標名稱上。



如需有關可為每個資料庫引擎新增之計數器指標的說明，請參閱 Performance Insights 計數器指標。

資料庫負載圖表

資料庫負載圖表顯示資料庫活動與資料庫執行個體容量間的比較值，以最大 vCPU 數線表示。依預設，堆疊折線圖以每個時間單位的平均作用中工作階段數來表示資料庫負載。資料庫負載依等待狀態切割 (分組)。

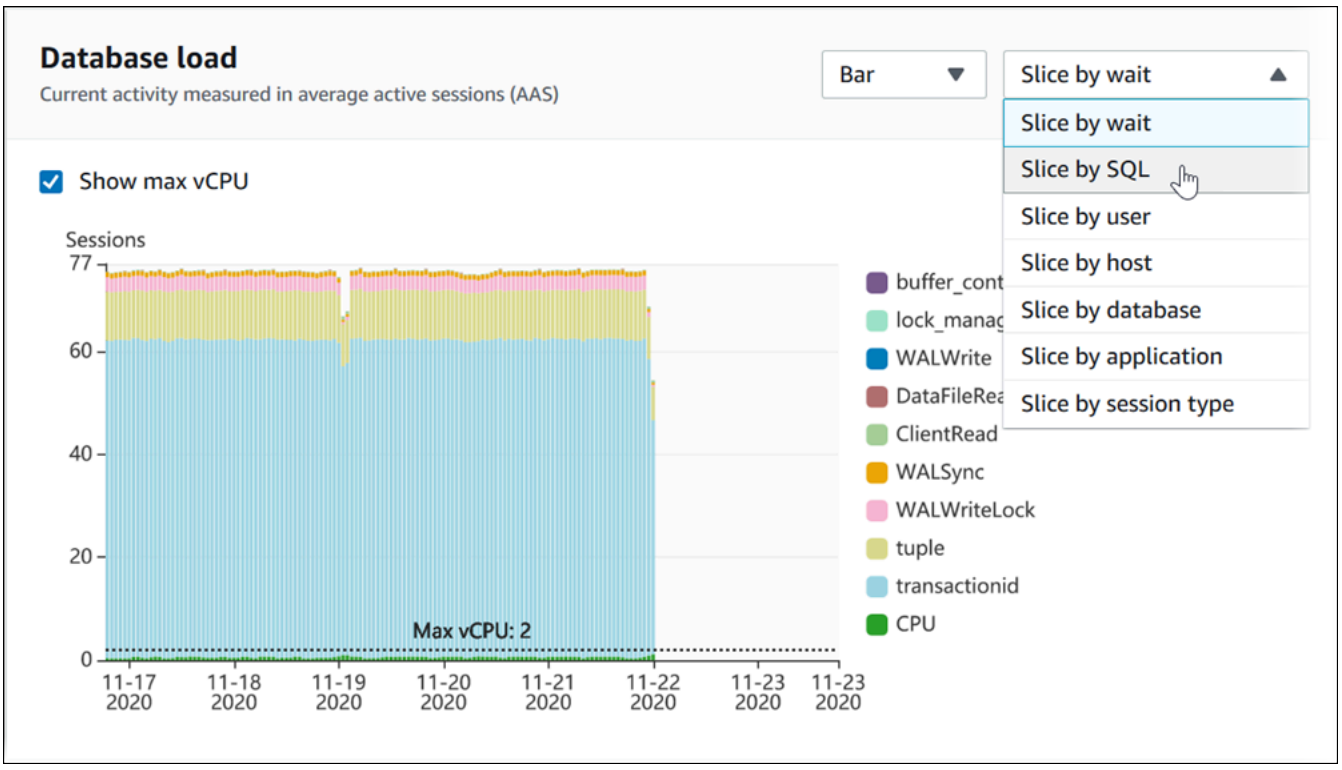


資料庫負載依維度配量

您可以選擇將負載顯示為作用中工作階段 (依任何支援維度分組)。下表顯示不同引擎各自支援的維度。

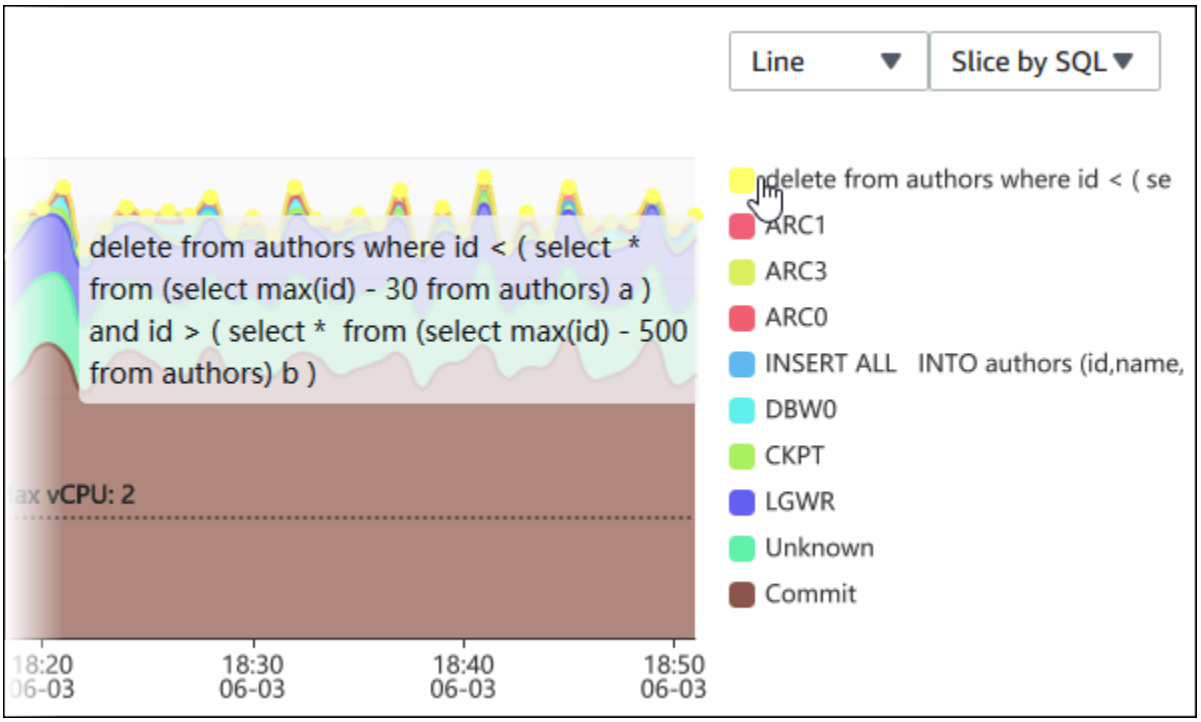
維度	Aurora PostgreSQL	Aurora MySQL
主機	是	是
SQL	是	是
使用者	是	是
等待	是	是
應用程式	是	否
資料庫	是	是
工作階段類型	是	否

下圖顯示 PostgreSQL 資料庫執行個體的維度。

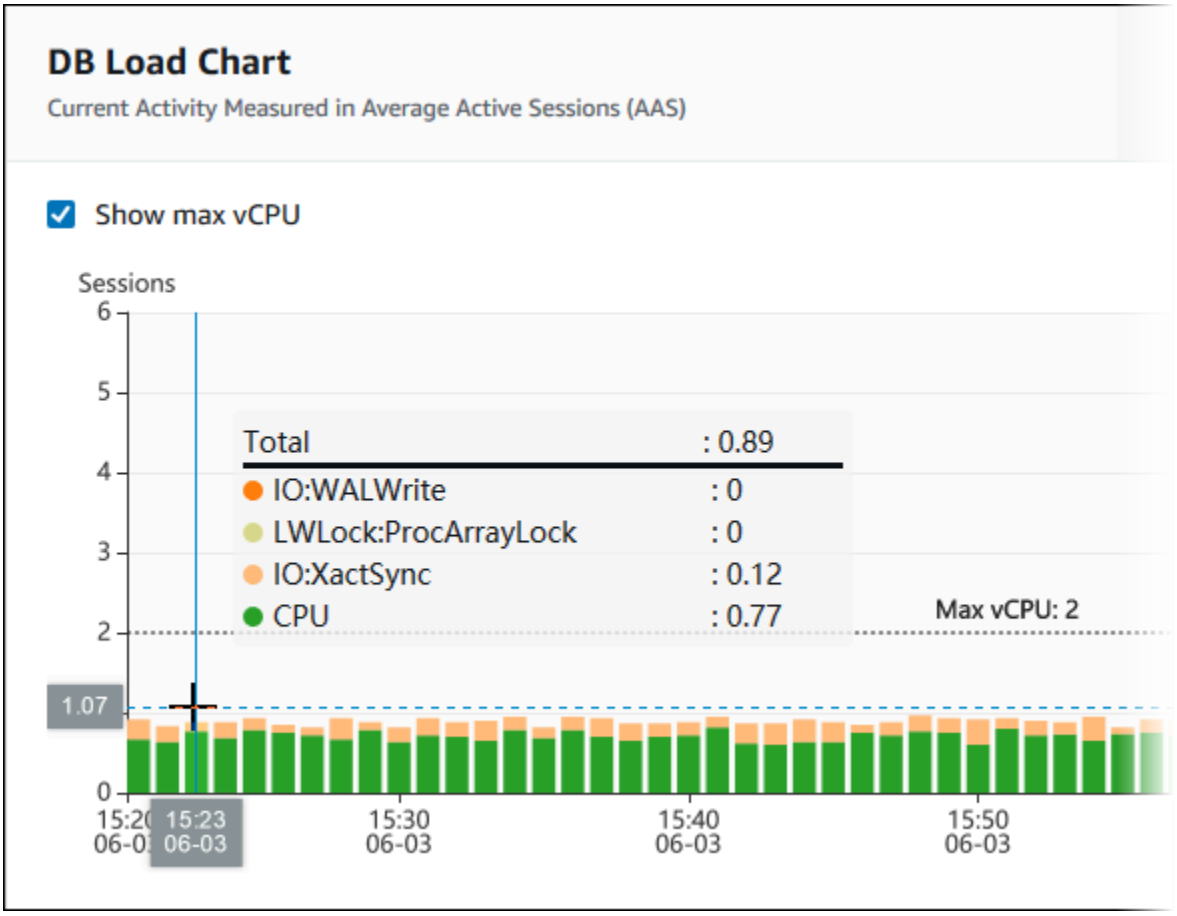


維度項目的資料庫負載詳細資訊

若要查看有關維度內資料庫負載項目的詳細資訊，請將游標移到項目名稱上。下圖顯示 SQL 陳述式的詳細資訊。

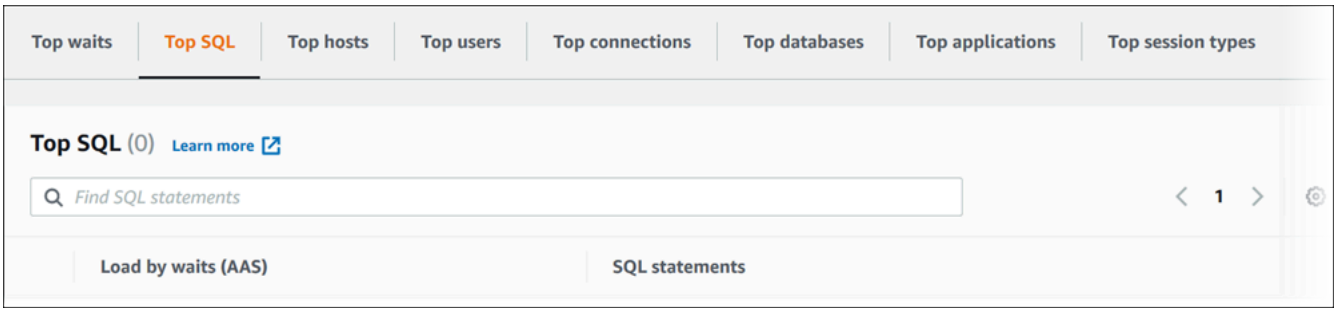


若要以圖例查看在所選時段內任何項目的詳細資訊，請將游標移到該項目上。



最高維度表格

最高維度表格依不同維度切割資料庫負載。維度是資料庫負載各種特性的類別或「配量依據」。如果維度是 SQL，Top SQL (最高 SQL) 會顯示在資料庫負載中佔最大比例的 SQL 陳述式。



選擇下列任一維度索引標籤：

Tab	Description	支援的引擎
最高 SQL	目前正在執行的 SQL 陳述式	全部
最高等待	資料庫後端正在等待的事件	全部
最高主機	連線用戶端的主機名稱	全部
最高使用者	登入資料庫的使用者	全部
最高應用程式	連線至資料庫的應用程式名稱	僅限 Aurora PostgreSQL
最高工作階段類型	目前工作階段的類型	僅限 Aurora PostgreSQL

若要了解如何使用 Top SQL (最高 SQL) 索引標籤來分析查詢，請參閱 最高 SQL 索引標籤概觀。

存取績效詳情儀表板

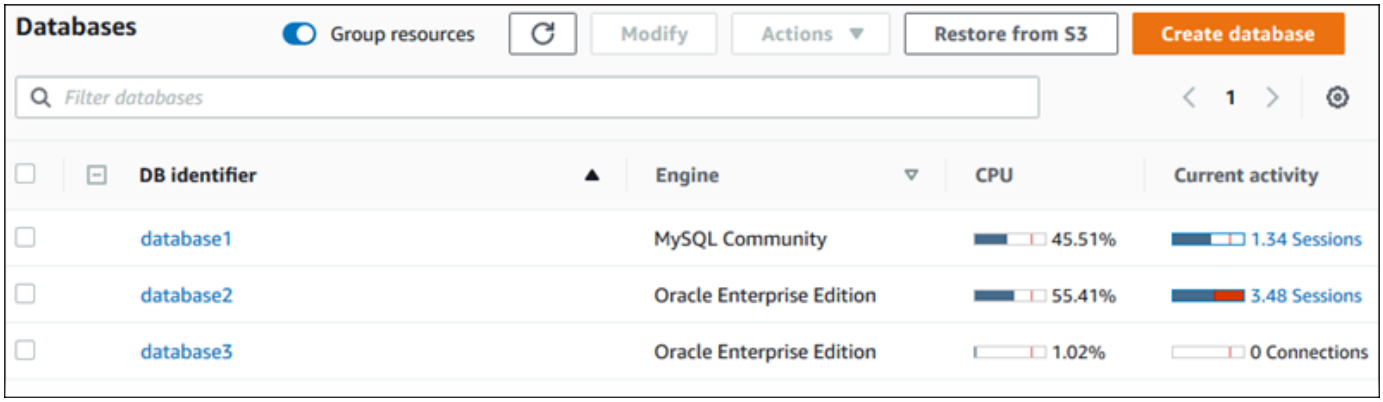
Amazon RDS 會在績效詳情儀表板中提供績效詳情和 CloudWatch 指標的合併檢視。

若要存取績效詳情儀表板，請使用下列程序。

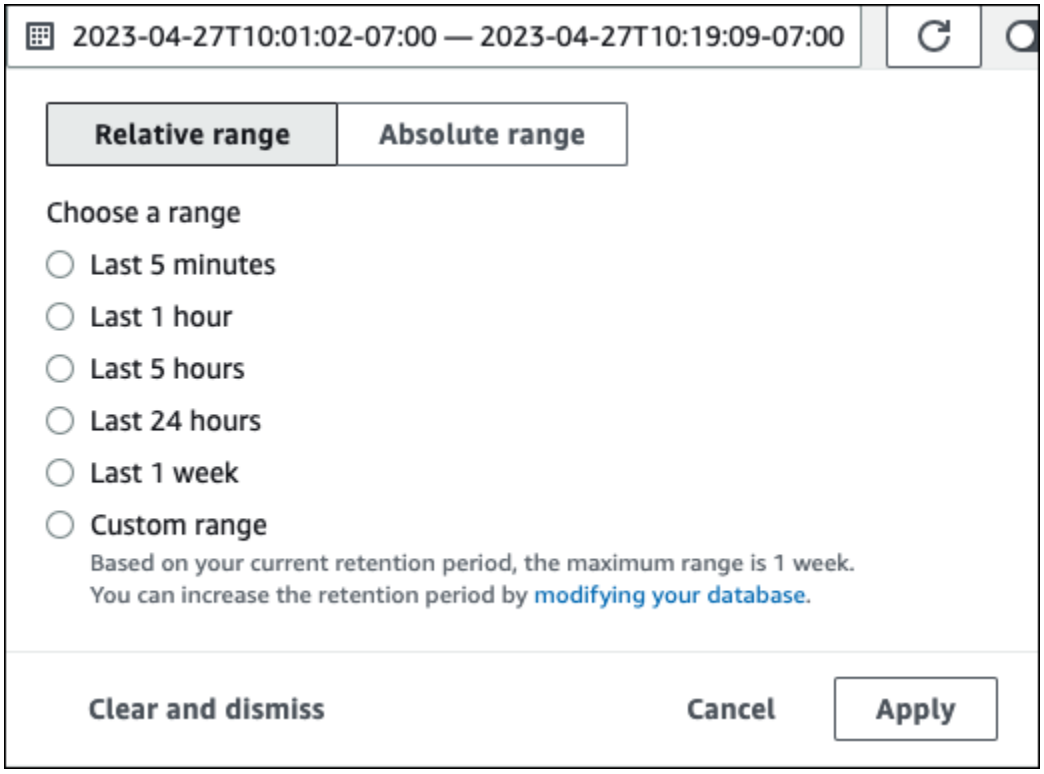
檢閱 AWS 管理主控台中的績效詳情儀表板

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在左側導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。

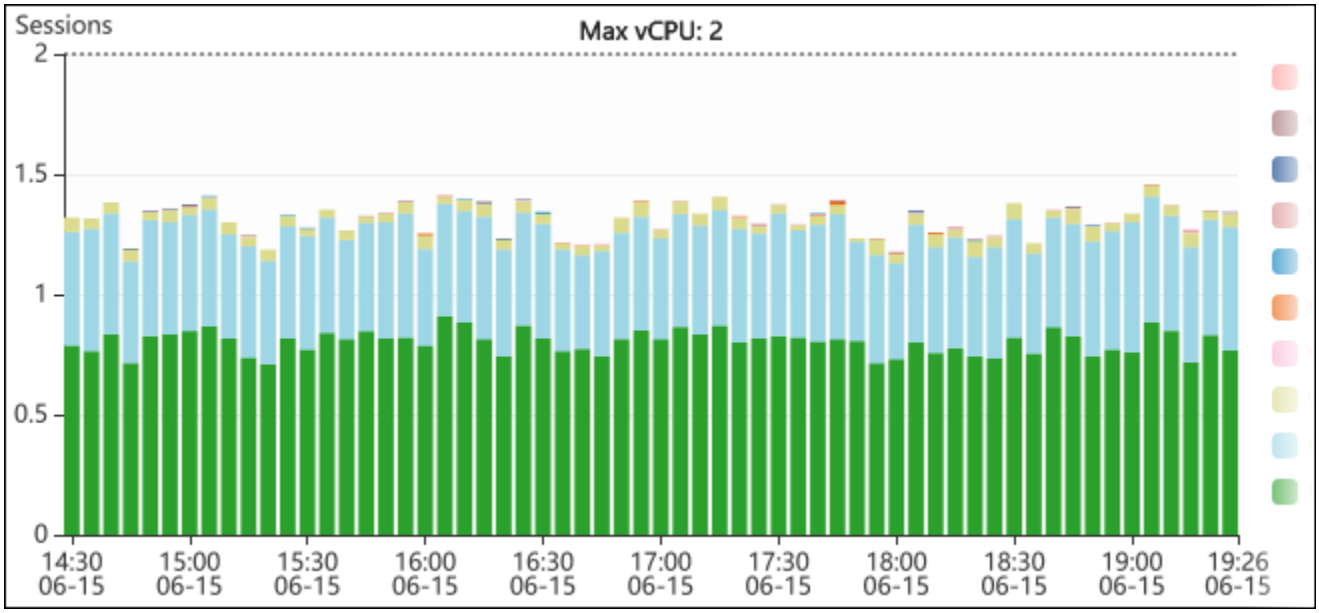
   對於已開啟 Performance Insights 的資料庫執行個體，您也可以在資料庫執行個體清單中選擇工作階段項目，來存取 Performance Insights 儀表板。在 Current activity (目前活動) 中，Sessions (工作階段) 項目顯示了過去五分鐘內平均作用中工作階段的資料庫負載。負載以進度條圖形的方式顯示。當進度條為空時，代表資料庫執行個體閒置中。隨著負載增加，進度條會填入藍色。當負載超過資料庫執行個體類別上的虛擬 CPU (vCPU) 數量時，進度條會轉紅，指出可能遇到瓶頸。



4. (選用) 選擇右上角的日期和時間範圍，並指定不同的相對或絕對時間間隔。您現在可以指定時間區間，並產生資料庫績效分析報告。該報告提供識別出的洞見和建議。如需更多詳細資訊，請參閱[在 Performance Insights 中建立效能分析報告](#)。

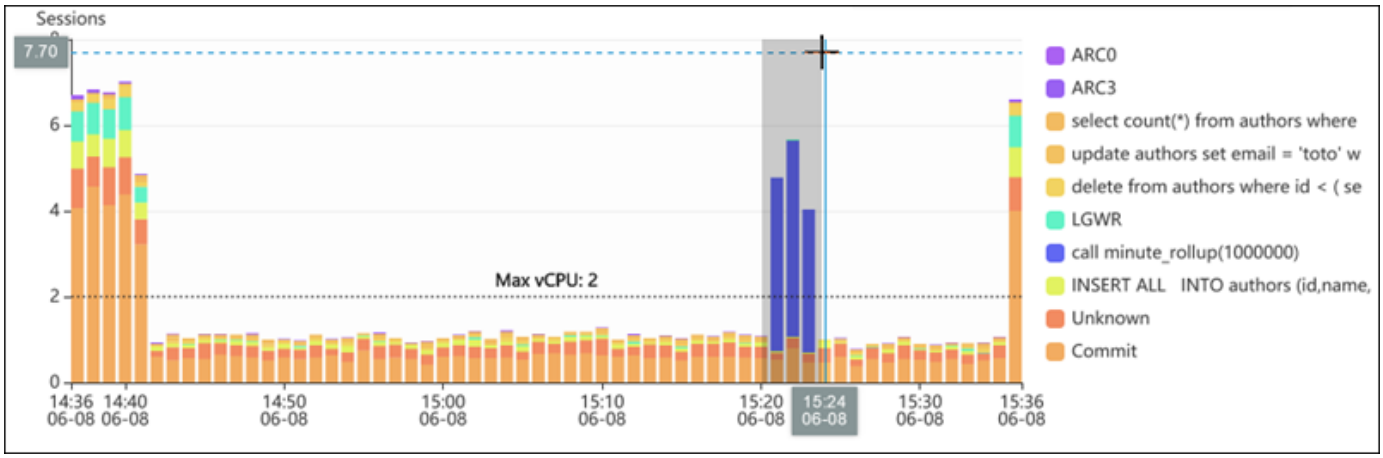


在下列螢幕擷取畫面中，資料庫負載為間隔為 5 小時。

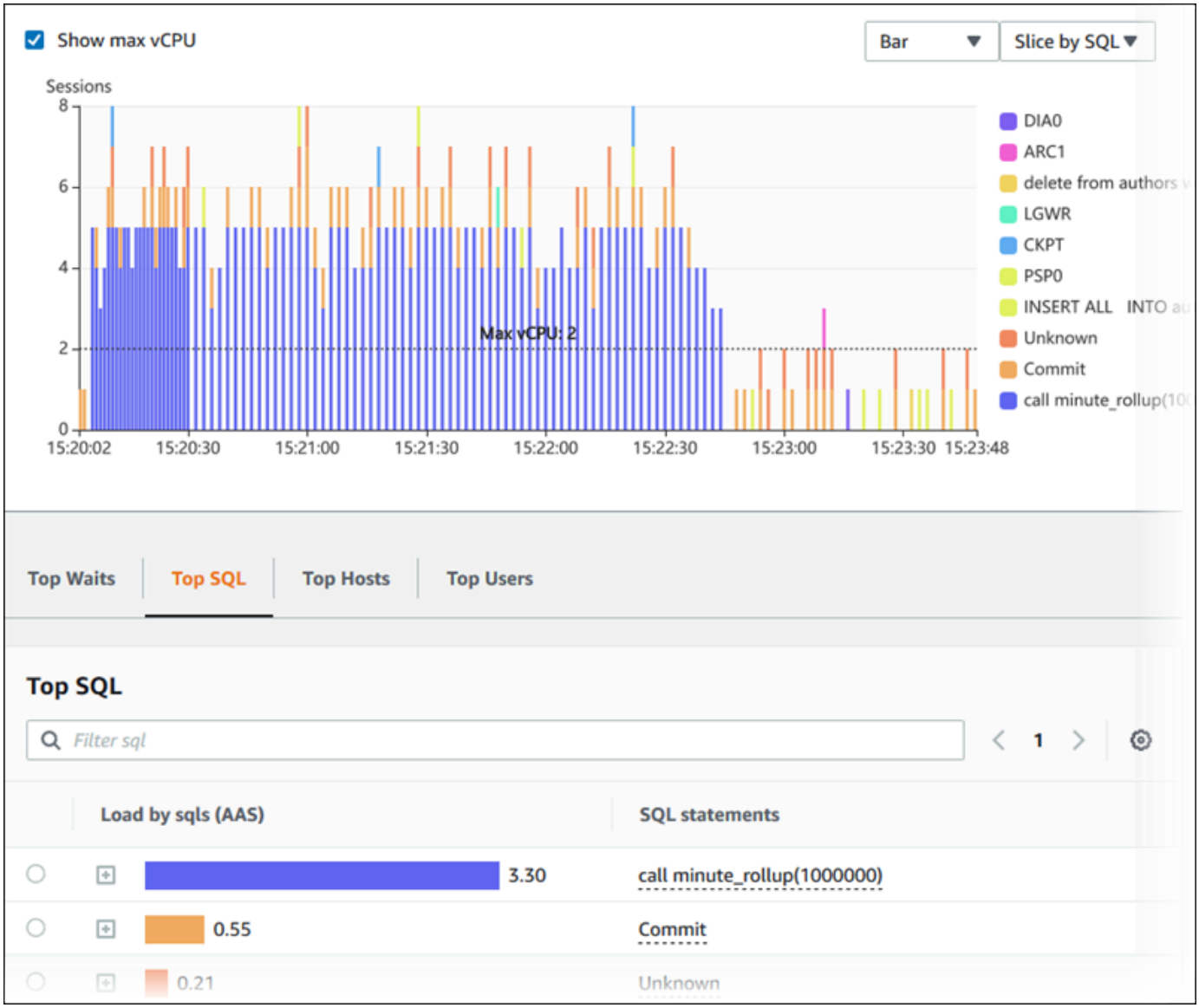


5. (選用) 若要放大一部分的資料庫負載圖表，請選擇開始時間並拖曳到您想要的時間範圍結束時間。

   所選區域將在資料庫負載圖表中突出顯示。



   放開滑鼠時，資料庫負載圖表會在所選 AWS 區域上放大，而 Top dimensions (最高維度) 資料表則會重新計算。



6. (選用) 若要自動重新整理資料，請選取自動重新整理。

績效詳情儀表板會自動以新資料進行重新整理。重新整理速度取決於顯示的資料量：

- 5 分鐘的資料每 10 秒重新整理一次。
- 1 小時的資料每 5 分鐘重新整理一次。
- 5 小時的資料每 5 分鐘重新整理一次。
- 24 小時的資料每 30 分鐘重新整理一次。
- 1 週的資料每天重新整理一次。

- 1 個月的資料每天重新整理一次。

按等待事件分析資料庫負載

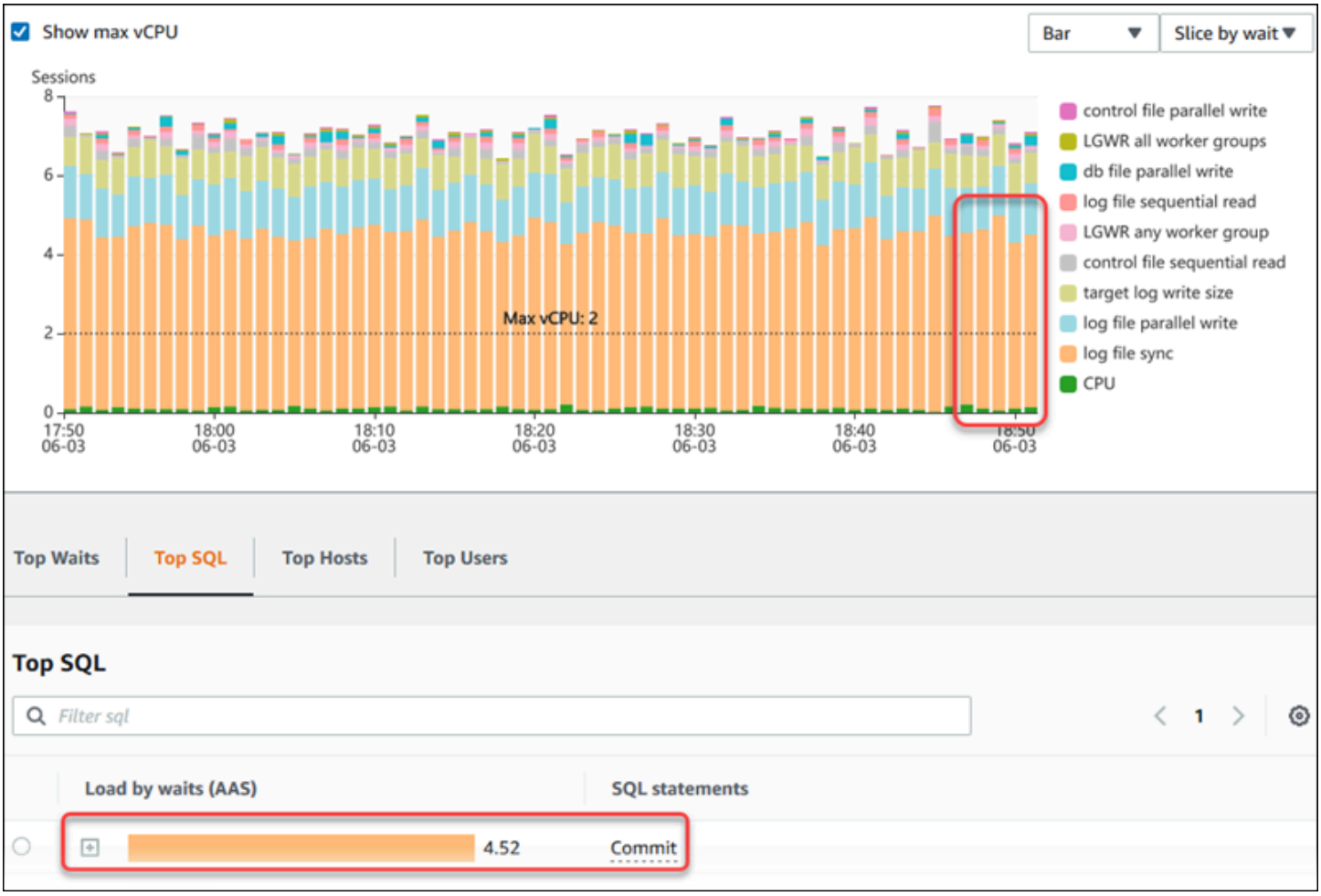
如果 Database load (資料庫負載) 圖表指出有瓶頸，您可以查明負載的來源。若要這麼做，請查看資料庫負載圖表下的最高負載項目表格。選擇特定項目，例如 SQL 查詢或使用者，來深入探討該項目並查看關於該項目的詳細資訊。

依等待和最高 SQL 查詢分組的資料庫負載是預設的績效詳情儀表板檢視。此組合通常可提供效能問題的最多見解。根據等待分組的資料庫負載顯示該資料庫中是否有任何資源或正在發生的瓶頸。在此情況下，最高負載項目表格的 SQL 標籤會顯示哪些查詢帶來這些負載量。

診斷效能問題的典型工作流程如下：

1. 檢閱資料庫負載圖表並查看是否有任何資料庫負載超越最高 CPU 線的情況。
2. 若有，請查看資料庫負載圖表，並找出哪一個或那幾個等待狀態是主因。
3. 利用檢視最高負載項目資料表上的 SQL 標籤之查詢對於那些等待狀態影響較大，藉此找出造成負載的摘要查詢。您可以根據等待列出資料庫負載欄來找出這些。
4. 選擇 SQL 標籤中的其中一個摘要查詢，展開並查看其中組成的子查詢。

例如，在下列儀表板中，日誌檔案同步等待佔了大部分的資料庫負載。LGWR 所有工作者群組等待也很高。最高 SQL 圖表顯示導致日誌檔案同步等待的原因：頻繁的 `COMMIT` 陳述式。在這種情況下，減少遞交頻率將能降低資料庫負載。



分析一段時間區間的資料庫效能

透過建立時間期間的效能分析報告，使用隨需分析來分析資料庫效能。檢視效能分析報告以尋找效能問題，例如資料庫執行個體中查詢的資源瓶頸或變更。Performance Insights 儀表板可讓您選取時間段並建立績效分析報告。您也可以將一或多個標籤新增到報告。

若要使用此功能，您必須使用付費方案的保留期。如需詳細資訊，請參閱Performance Insights 的定價和資料保留

該報告可在績效分析報告 - 新標籤中選取和檢視。該報告包含洞見、相關指標和解決效能問題的建議。您可以在 Performance Insights 保留期間內檢視報告。

如果報告分析期間的開始時間超過保留期，則會刪除報告。您也可以在保留期結束之前刪除報告。

若要偵測效能問題並產生資料庫執行個體的分析報告，您必須開啟 Performance Insights。如需開啟績效詳情的詳細資訊，請參閱 開啟和關閉 Aurora 的 Performance Insights。

如需此功能的區域、資料庫引擎和執行個體類別支援的資訊，請參閱 Performance Insights 功能的 Amazon Aurora 資料庫引擎、區域和執行個體類別支援

在下列各節中，您可以建立、檢視、新增標籤和刪除效能分析報告。

主題

- 在 Performance Insights 中建立效能分析報告
- 在 Performance Insights 中檢視效能分析報告
- 在 Performance Insights 中將標籤新增至效能分析報告
- 在 Performance Insights 中刪除效能分析報告

在 Performance Insights 中建立效能分析報告

您可以在 Performance Insights 儀表板中針對特定區間建立績效分析報告。您可以選取時間區間，並將一或多個標籤新增至分析報告。

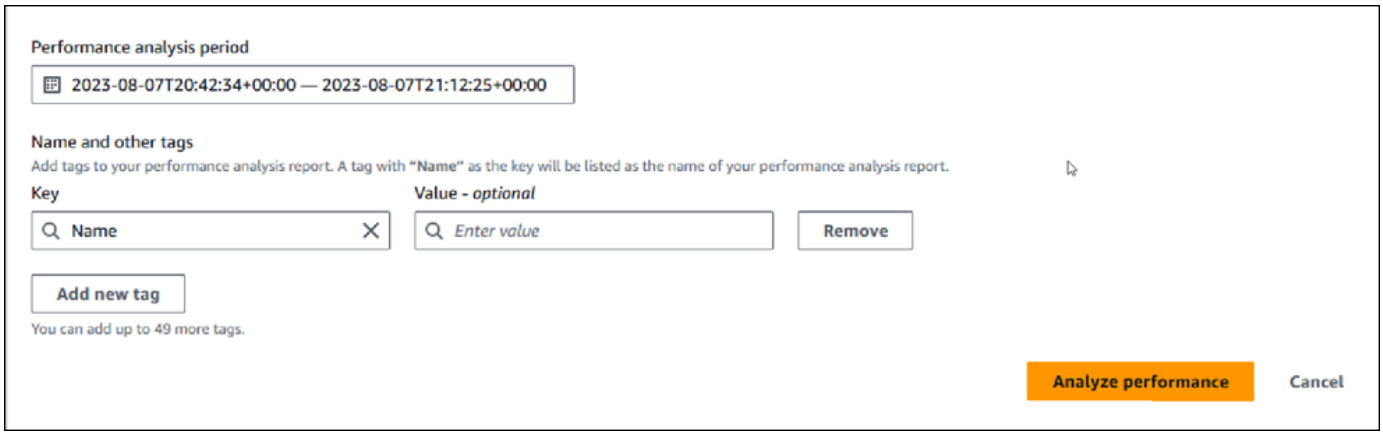
分析期間的範圍可以從 5 分鐘到 6 天不等。在分析開始時間之前，必須至少有 24 小時的效能資料。

如需此功能的區域、資料庫引擎和執行個體類別支援的資訊，請參閱 Performance Insights 功能的 Amazon Aurora 資料庫引擎、區域和執行個體類別支援

若要建立一段時間區間的績效分析報告

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在左側導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。
4. 在 Performance Insights 儀表板上的資料庫負載區段中選擇分析效能。

   設定要顯示的時間區間並新增一或多個標籤至績效分析報告的欄位。

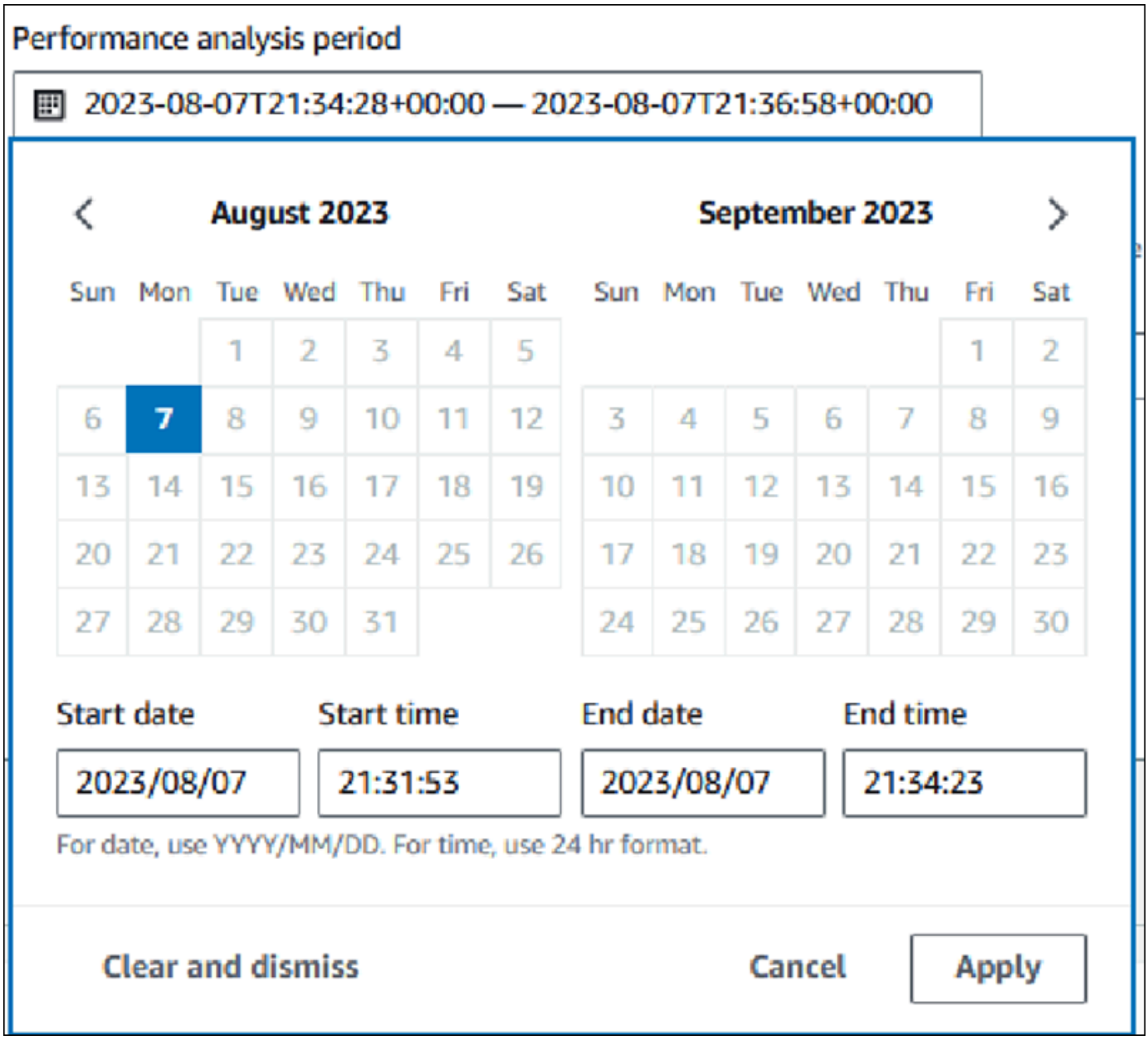


5. 選擇時間區間。如果您在右上角的相對範圍或者絕對範圍內設定時間區間，您只能輸入或選取此時間區間內的報告日期和時間。如果您選取此時間區間以外的分析區間，則會顯示錯誤訊息。

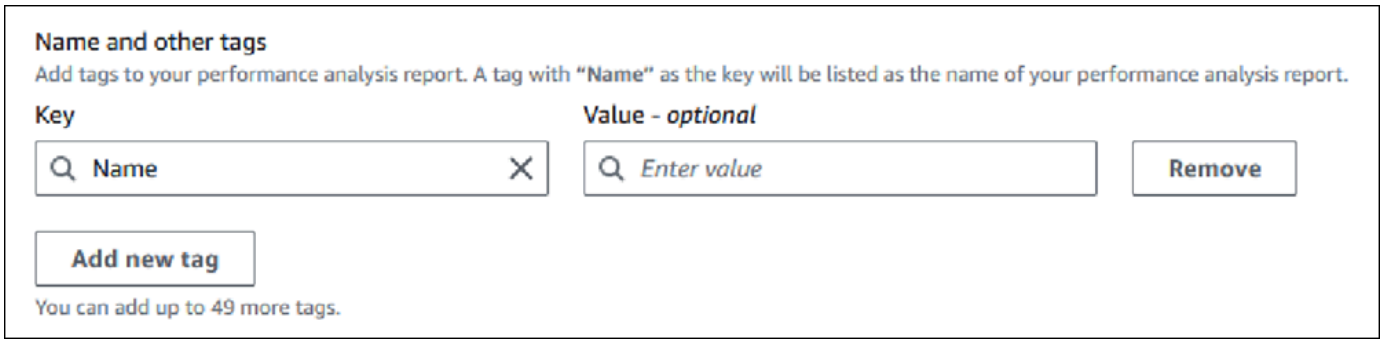
   若要設定時間區間，您可以執行下列任一項操作：

   - 按下並拖曳資料庫負載圖表上的任何滑桿。

     該績效分析區間方塊會顯示所選的時段，而資料庫負載圖表會反白顯示所選取的時間區間。
   - 在績效分析區間方塊中選擇開始日期、開始時間、結束日期和結束時間。



6. (選用) 輸入索引鍵和值 - 選用，以新增報告的標籤。



7. 選擇分析效能。

   橫幅會顯示報告產生成功或失敗的訊息。該訊息還會提供檢視報告的連結。

   下列範例顯示成功建立報告的訊息橫幅。

   該報告可以於績效分析報告 - 新標籤中檢視。

您可以使用 AWS CLI建立效能分析報告。如需如何使用 建立報告的範例 AWS CLI，請參閱 建立一段時間區間的績效分析報告。

在 Performance Insights 中檢視效能分析報告

該績效分析報告 - 新標籤會列出為資料庫執行個體建立的所有報告。為每個報告顯示以下內容：

- ID：報告的唯一識別碼。
- 名稱：新增至報告的標籤索引鍵。
- 報告建立時間：您建立報告的時間。
- 分析開始時間：報告中分析的開始時間。
- 分析結束時間：報告中分析的結束時間。

檢視績效分析報告

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在左側導覽窗格中，選擇 Performance Insights (績效詳情)。

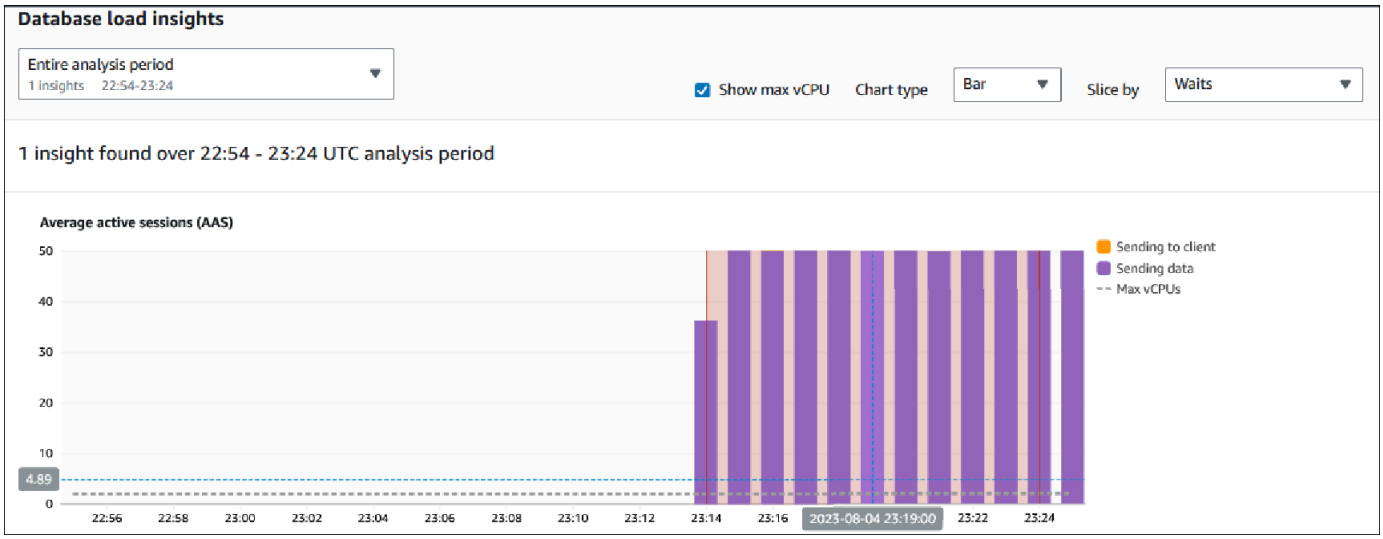
3. 選擇您要檢視其分析報告的資料庫執行個體。
4. 在 Performance Insights 儀表板中向下捲動，並選擇效能分析報告 - 新索引標籤。

   會顯示不同時間區間的所有分析報告。
5. 選擇您要檢視報告的 ID。

   如果識別出多個洞見，資料庫負載圖表預設會顯示整個分析區間。如果報告識別出一個洞見，則資料庫負載圖表會依預設顯示洞見。

   儀表板也會列出報告標籤區段的標籤。

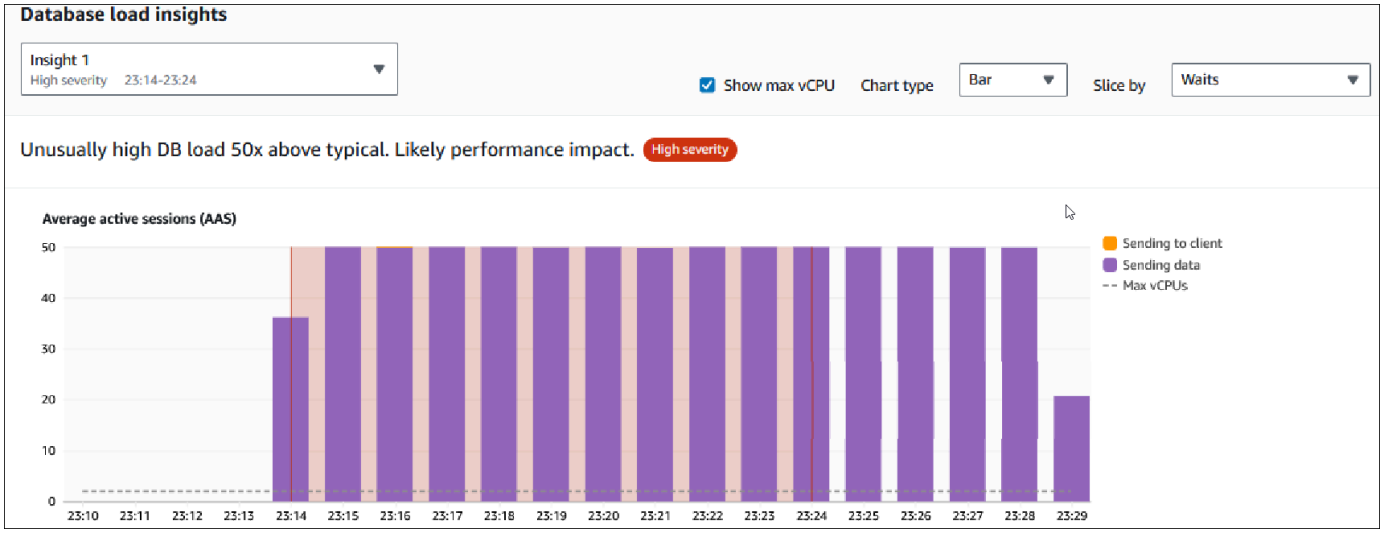
   下列範例顯示報告的整個分析區間。

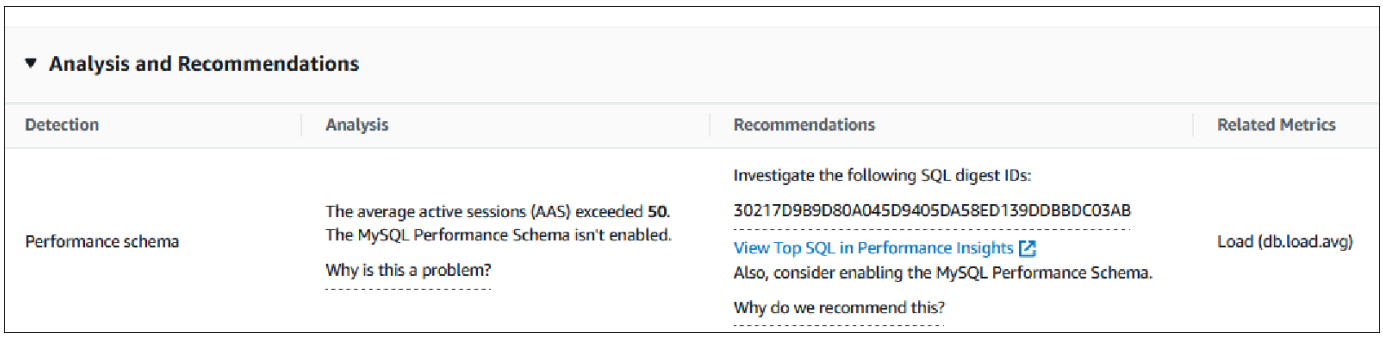


6. 如果報告中識別出一個以上的洞見，請選擇資料庫負載洞見清單中您要檢視的洞見。

   儀表板會顯示洞見訊息、資料庫負載圖表會反白顯示洞見的時間區間、分析和建議，以及報告標籤的清單。

   下列範例會顯示報告中的資料庫負載洞見。





在 Performance Insights 中將標籤新增至效能分析報告

您可以在建立或檢視報告時新增標籤。您最多可以為報告新增 50 個標籤。

您需要許可以新增標籤。如需 Performance Insights 存取政策的詳細資訊，請參閱 設定績效詳情的存取政策。

若要在建立報告時新增一個或多個標籤，請參閱程序 在 Performance Insights 中建立效能分析報告 中的步驟 6。

在檢視報告時新增一個或多個標籤

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在左側導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。

   資料庫執行個體的績效詳情儀表板即會出現。
4. 向下捲動並選擇績效分析報告 - 新標籤。

5. 選擇您要新增標籤的報告。

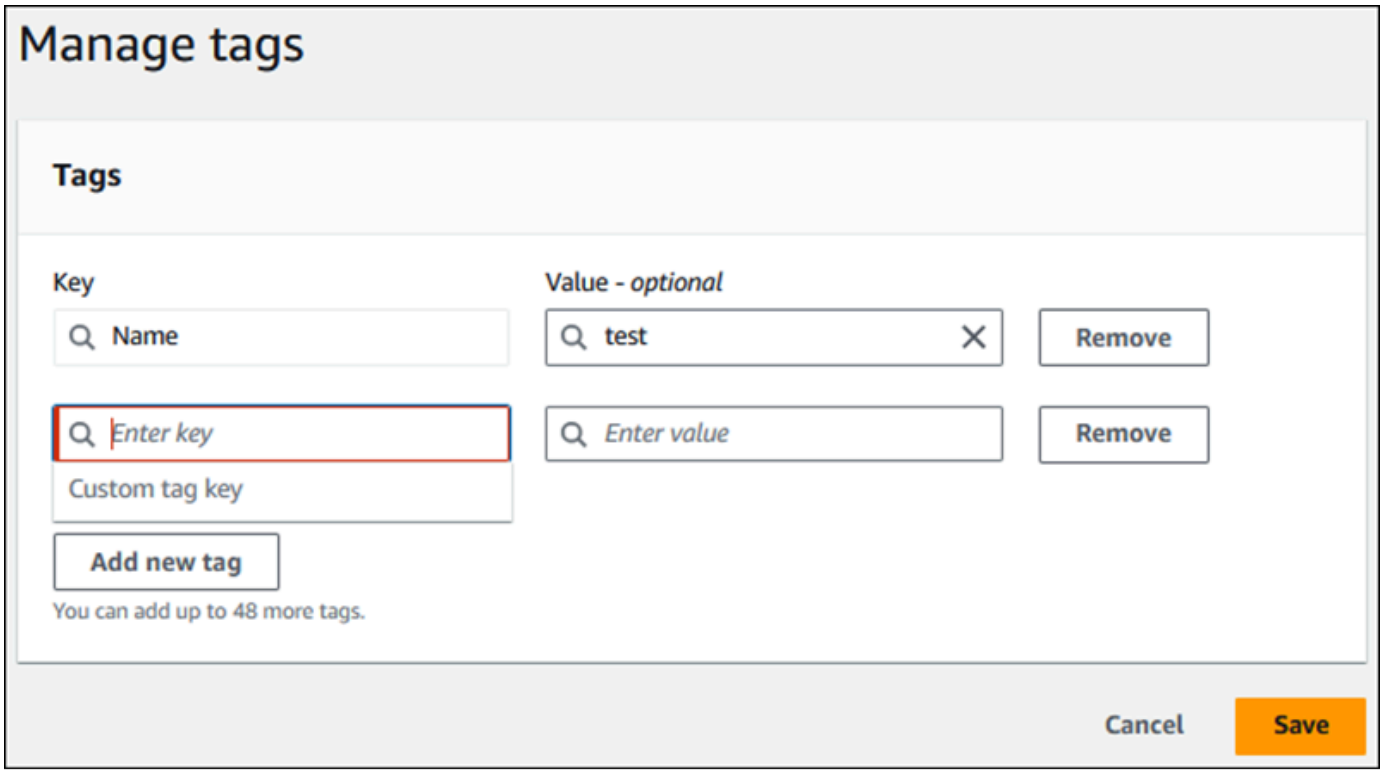
   儀表板會顯示報告。

6. 向下捲動至標籤並選擇管理標籤。

7. 選擇 Add new tag (新增標籤)。

8. 輸入索引鍵和值 - 選用，然後選擇新增標籤。

   下列範例提供為所選報告新增標籤的選項。



   將為報告建立一個新標籤。

   報告的標籤清單會顯示在儀表板上的標籤區段。如果您要從報告中移除標籤，請選擇標籤旁邊的移除。

在 Performance Insights 中刪除效能分析報告

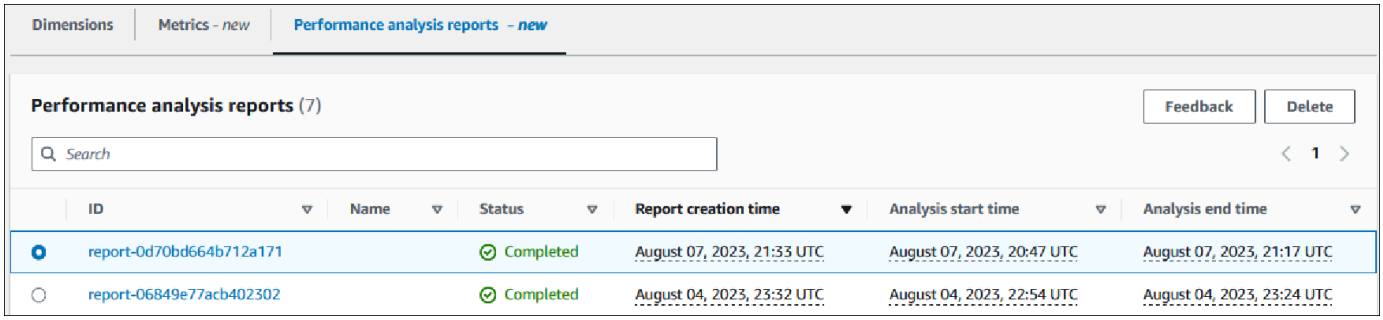
您可以從績效分析報告標籤所顯示的報告清單中刪除報告，或是檢視報告時刪除報告。

刪除報告

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。

2. 在左側導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。

   資料庫執行個體的績效詳情儀表板即會出現。
4. 向下捲動並選擇績效分析報告 - 新標籤。
5. 選取您要刪除的報告，然後選擇右上角的刪除。



   此時會顯示確認視窗。選擇確認後，即會刪除報告。
6. (選用) 選擇您要刪除報告的 ID。

   在報告頁面，選擇右上角的刪除。

   此時會顯示確認視窗。選擇確認後，即會刪除報告。

使用 Performance Insights 中的最高 SQL 索引標籤分析查詢

在 Amazon RDS 績效詳情儀表板中，您可以在 Top dimensions (最高維度) 表格的 Top SQL (最高 SQL) 索引標籤中找到執行中和近期查詢的相關資訊。您可以使用此資訊來調校查詢。

主題

- 最高 SQL 索引標籤概觀
- 在績效詳情儀表板中存取更多 SQL 文字
- 在績效詳情儀表板中檢視 SQL 統計數字

最高 SQL 索引標籤概觀

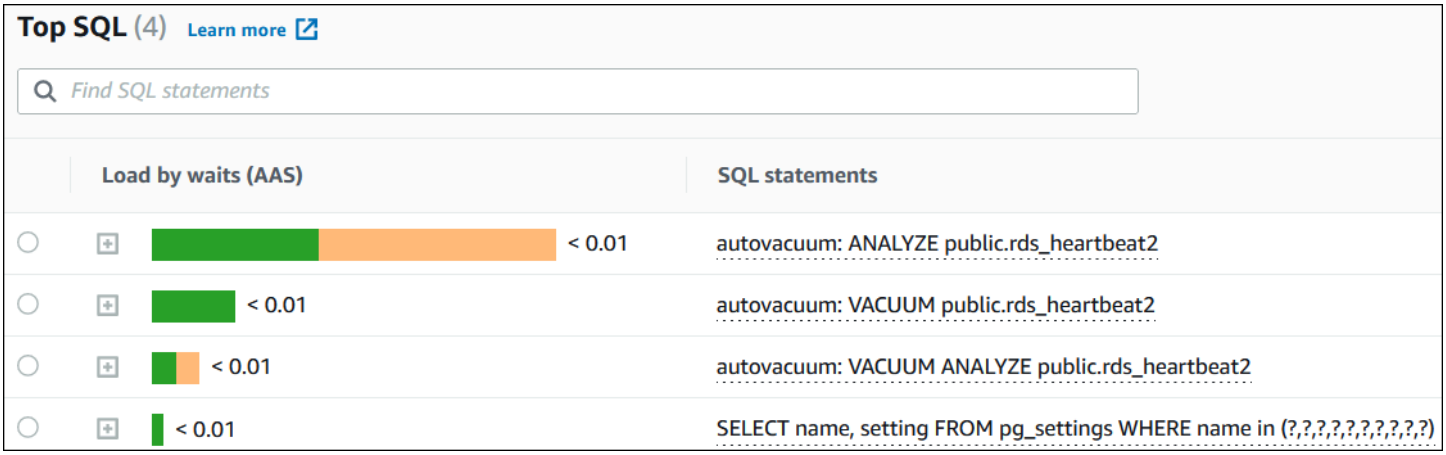
依預設，Top SQL (最高 SQL) 索引標籤會顯示在資料庫負載中佔最大比例的 25 個查詢。為了協助調校查詢，您可以分析查詢文字和 SQL 統計資料等資訊。您還可以選擇要出現在 Top SQL (最高 SQL) 索引標籤中的統計數字。

主題

- SQL 文字
- SQL Statistics
- 依等待分組的負載 (AAS)
- 檢視 SQL 資訊
- 選擇統計資料偏好設定

SQL 文字

根據預設，Top SQL (最高 SQL) 資料表中的每個資料列都會顯示每個 SQL 陳述式的 500 位元組文字。

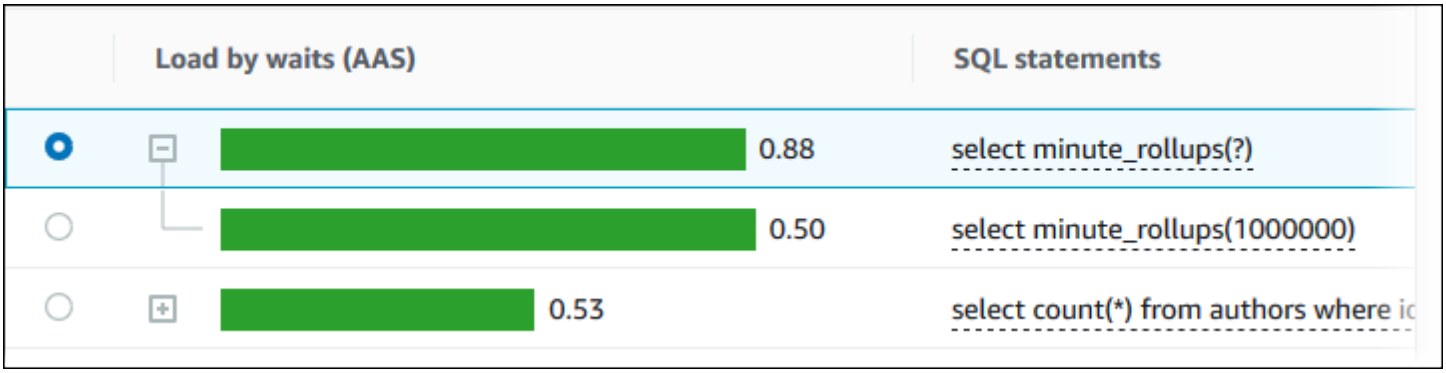


若要了解如何查看超過預設 500 位元組的 SQL 文字，請參閱 在績效詳情儀表板中存取更多 SQL 文字。

SQL 摘要綜合顯示結構相似但可能有不同常值的多個實際查詢。摘要中，問號會取代硬式編碼值。例如，摘要可能是 `SELECT * FROM emp WHERE lname= ?`。此摘要可能包含下列子查詢：

```
SELECT * FROM emp WHERE lname = 'Sanchez'
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

若要逐句查看摘要中的 SQL 陳述式，請選取查詢，然後選擇加號 (+)。在下列範例中，選取的查詢是摘要。

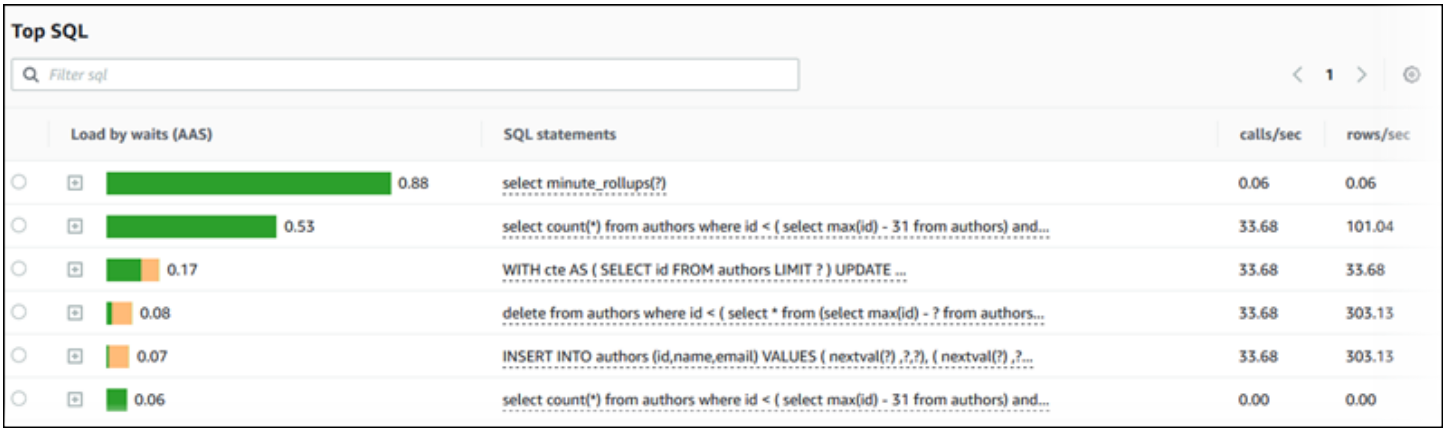


> **Note**
>
> SQL 摘要會將類似的 SQL 陳述式分組，但不會修訂敏感資訊。

SQL Statistics

SQL 統計數字是 SQL 查詢的效能相關指標。例如，績效詳情可能顯示每秒的執行次數或每秒處理的資料列數。績效詳情只收集最常用查詢的統計數字。這些通常會符合績效詳情儀表板中依負載顯示的熱門查詢。

Top SQL (最高 SQL) 表格的每一行顯示 SQL 陳述式或摘要的相關統計數字，如下列範例所示。



績效詳情可將 SQL 統計數字報告為 `0.00` 和 `-` (不明)。在下列情況下會發生此情況：

- 只有一個樣本存在。例如，績效詳情會根據 `pg_stat_statements` 檢視中的多個樣本來計算 Aurora PostgreSQL 查詢。當工作負載執行時間很短時，績效詳情可能只收集到一個樣本，這代表它無法計算變化速率。不明值用破折號 (-) 表示。
- 兩個樣本具有相同的值。績效詳情無法計算變化速率，因為沒有發生變化，因此它將速率報告為 `0.00`。

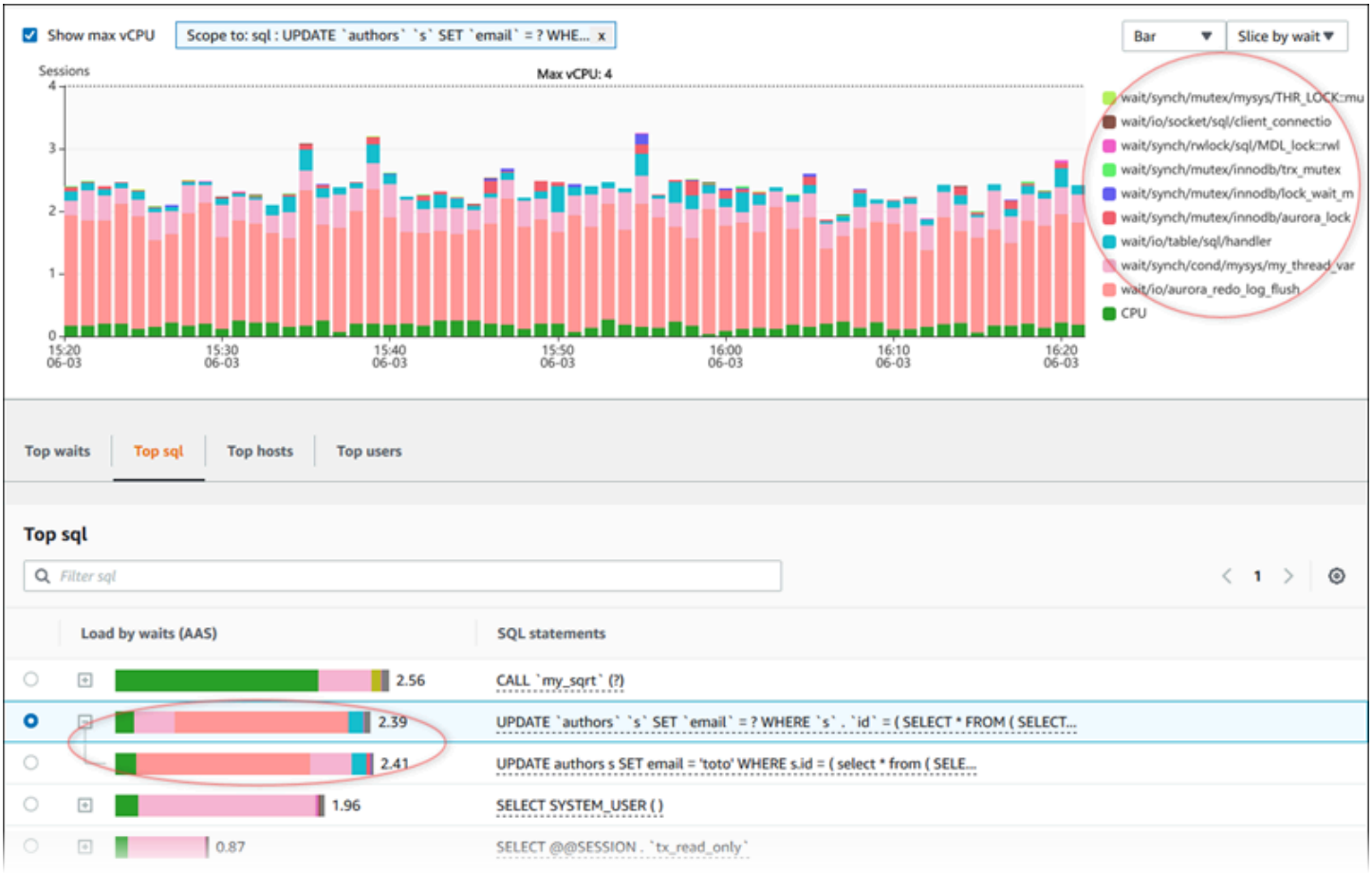
- Aurora PostgreSQL 陳述式缺少有效識別符。只有在解析和分析之後，PostgreSQL 才會為陳述式建立識別符。因此，陳述式可以存在於 PostgreSQL 內部記憶體中結構，而沒有識別符。由於績效詳情每秒對內部記憶體中結構採樣一次，所以低延遲查詢可能只顯示一個樣本。如果查詢識別符無法用於此樣本，則績效詳情無法將此陳述式與其統計數字產生關聯。不明值用破折號 (-) 表示。

如需 Aurora 引擎之 SQL 統計數字的說明，請參閱績效詳情的 SQL 統計數字。

依等待分組的負載 (AAS)

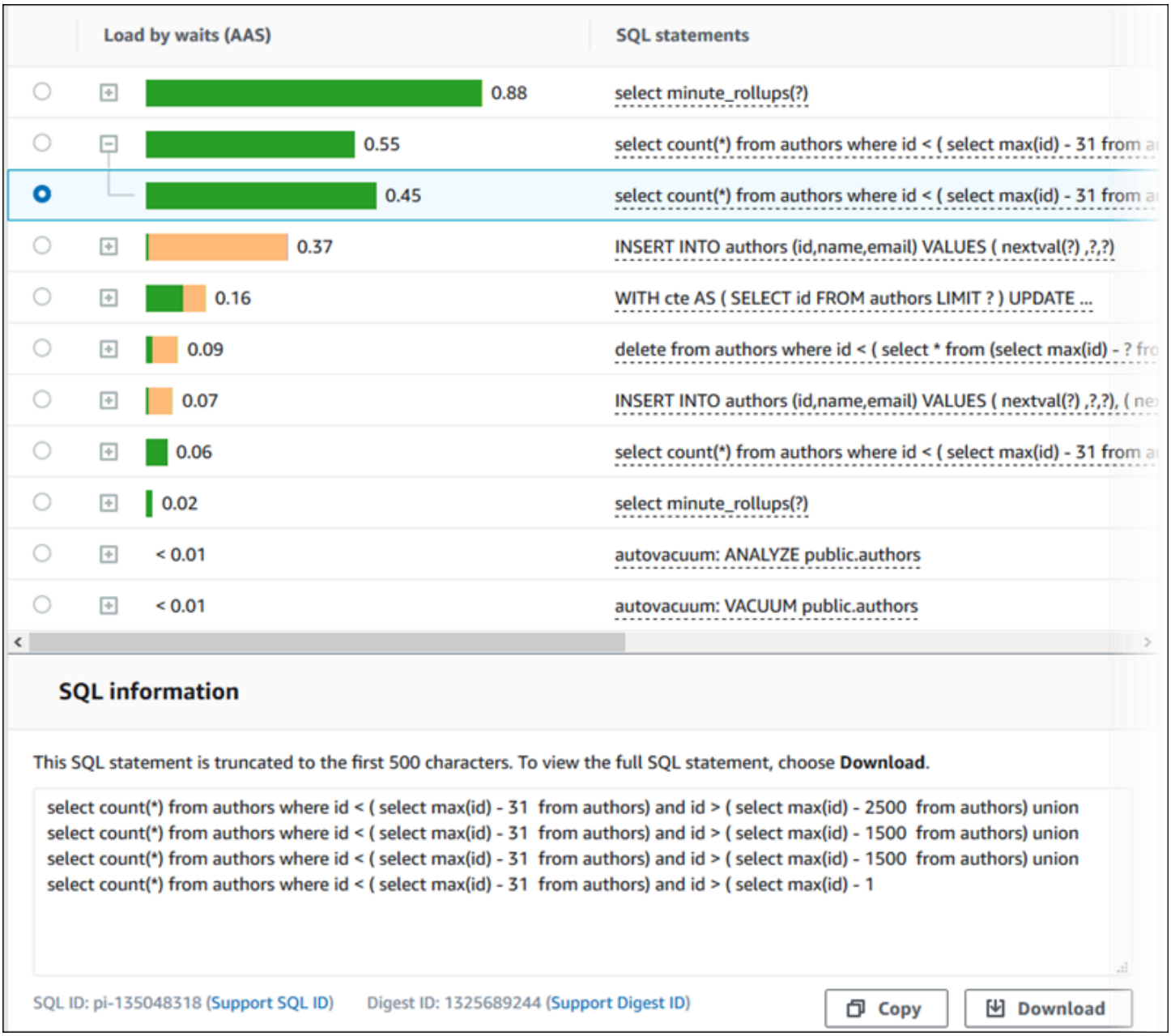
在最高 SQL 中，根據等待的負載 (AAS) 欄說明了與每個最高負載項目相關聯的資料庫負載百分比。此欄根據目前在資料庫負載圖表中所選的群組依據來反映出該項目的負載。如需平均作用中工作階段 (AAS) 的詳細資訊，請參閱 平均作用中工作階段。

例如，您可以依等待狀態將 DB load (資料庫負載) 圖表分組。您可檢查最高負載項目資料表中的 SQL 查詢。在此情況下，根據等待列出資料庫負載列較大且分段，並以顏色為代碼來顯示該查詢所帶來的指定等待狀態程度。這也會顯示哪些等待狀態會影響選取的查詢。



檢視 SQL 資訊

在最高 SQL 資料表中，您可以開啟陳述式以檢視其資訊。資訊會顯示在底部窗格中。

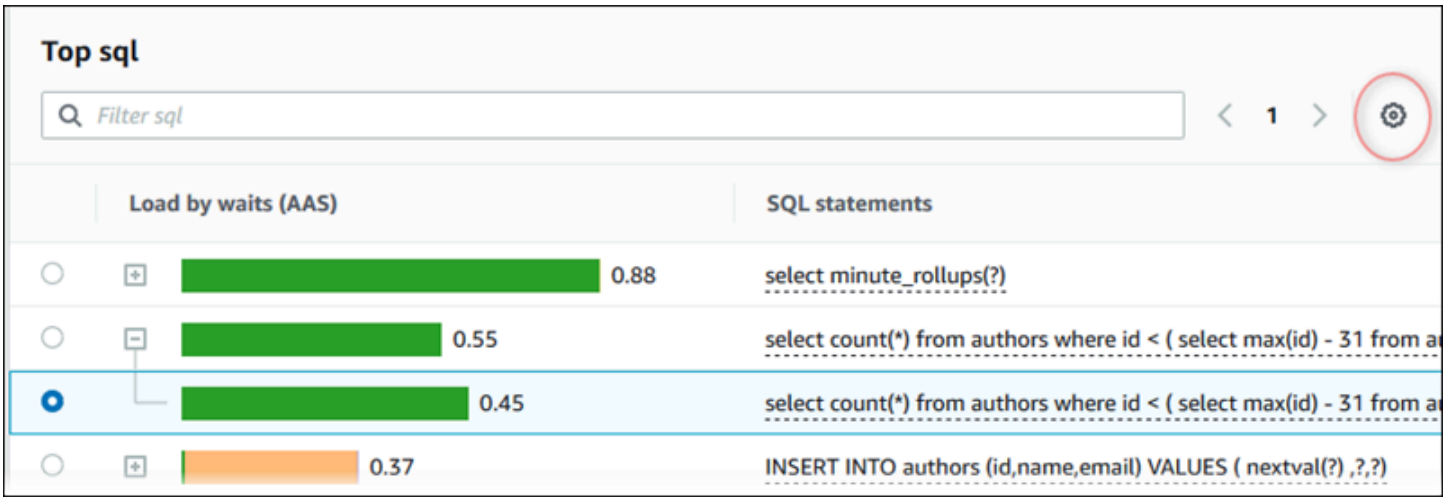


以下幾種與 SQL 陳述式相關聯的識別符 (ID)：

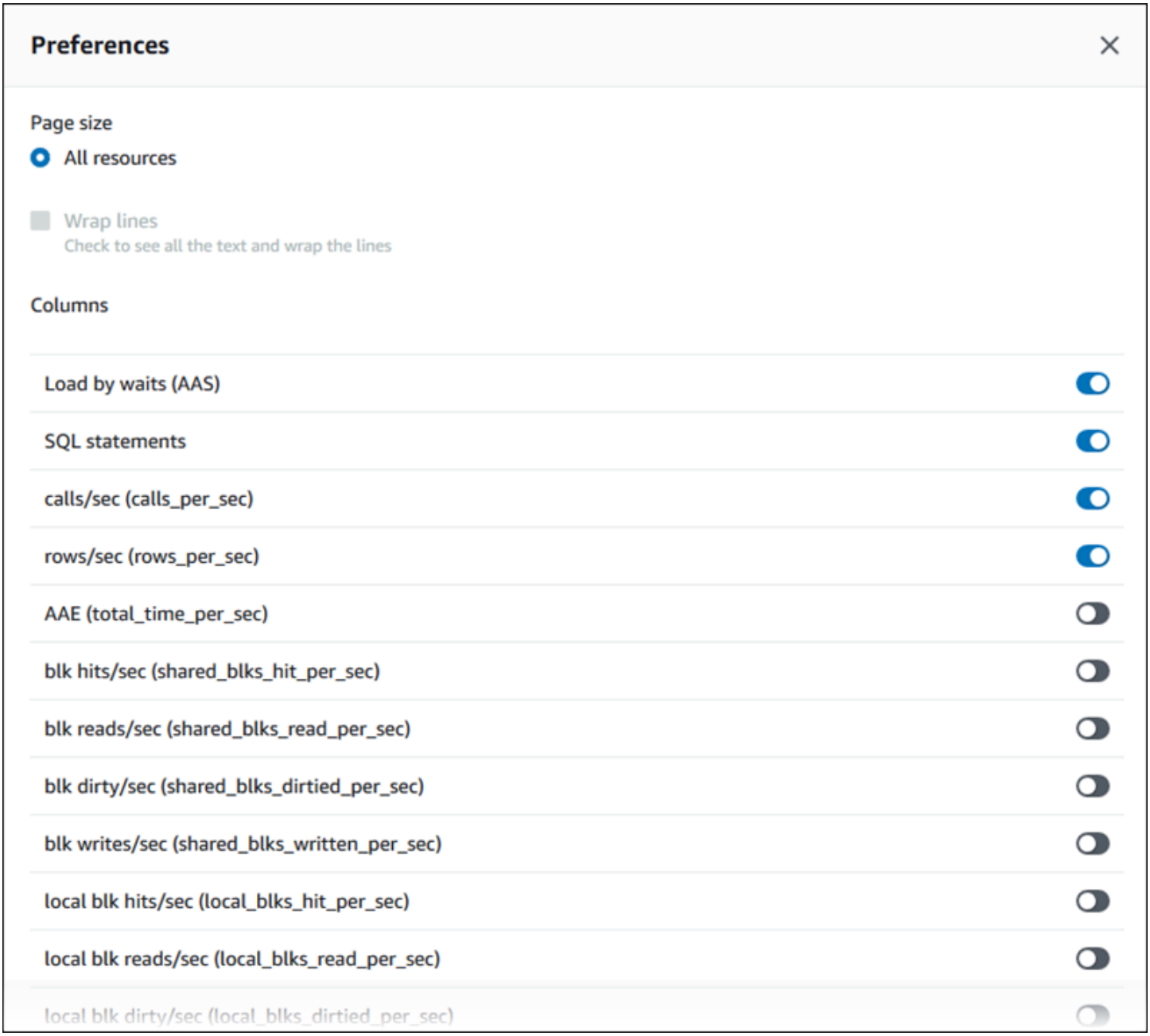
- 支援 SQL ID – SQL ID 的雜湊值。此值只是在您與 AWS Support 互動時用來參考 SQL ID。AWSSupport 無法存取您的實際 SQL ID 和 SQL 文字。
- 支援摘要 ID – 摘要 ID 的雜湊值。此值只是在您與 AWS Support 互動時用來參考摘要 ID。AWSSupport 無法存取您的實際摘要 ID 和 SQL 文字。

選擇統計資料偏好設定

您可以選擇 Preferences (偏好設定) 圖示，以控制 Top SQL (最高 SQL) 索引標籤中顯示的統計資料。



當您選擇 Preferences (偏好設定) 圖示時，Preferences (偏好設定) 視窗會開啟。下列螢幕擷取畫面是 Preferences (偏好設定) 視窗的範例。



若要讓您要的統計數字出現在 Top SQL (最高 SQL) 索引標籤中，請使用滑鼠捲動到視窗底部，然後選擇 Continue (繼續)。

如需 Aurora 引擎每秒或每次呼叫統計資料的詳細資訊，請參閱 [績效詳情的 SQL 統計數字] 中的引擎特定 SQL 統計資料一節

在績效詳情儀表板中存取更多 SQL 文字

根據預設，最高 SQL 資料表中的每個列會顯示每個 SQL 陳述式的 500 位元組的 SQL 文字。

SQL 陳述式超過 500 位元組時，您可以在 Top SQL 表格下方的 SQL text (SQL 文字) 區段檢視更多文字。在此情況下，SQL text (SQL 文字) 所顯示查詢的最大長度為 4 KB。此限制是由主控台引入，並受限於資料庫引擎設定的限制。若要儲存 SQL text (SQL 文字) 中顯示的文字，請選擇 Download (下載)。

主題

- Aurora MySQL 的文字大小限制
- 為 Aurora PostgreSQL 資料庫執行個體設定 SQL 文字限制
- 在 Performance Insights 儀表板中檢視和下載更多 SQL 文字

Aurora MySQL 的文字大小限制

當您下載 SQL 陳述式時，資料庫引擎會決定文字的最大長度。您可以下載的文字最大限制為以下每個引擎的限制。

資料庫引擎	下載文字長度上限
Aurora MySQL	長度固定為 4,096 個位元組。

Performance Insights 主控台的 SQL text (SQL 文字) 區段顯示上限為引擎傳回的最大值。例如，如果 Aurora MySQL 最多傳回 1 KB 內容到 Performance Insights，則其僅能收集並顯示 1 KB 內容，即使原始查詢較大也如此。因此，當您查看 SQL text (SQL 文字) 或下載查詢時，Performance Insights 會傳回相同的位元組數量。

如果您使用 AWS CLI 或 API，Performance Insights 便不會有主控台強制施加的 4 KB 限制。`DescribeDimensionKeys` 和 `GetResourceMetrics` 會傳回最多 500 個位元組。

> **Note**
>
> `GetDimensionKeyDetails` 會傳回完整查詢，但大小會受限於引擎限制。

為 Aurora PostgreSQL 資料庫執行個體設定 SQL 文字限制

Aurora PostgreSQL 處理文字的方式不同。您可以使用資料庫執行個體參數 `track_activity_query_size` 設定文字大小限制。此參數具有下列特性：

預設文字大小

在 Aurora PostgreSQL 9.6 版中，`track_activity_query_size` 參數的預設設定為 1,024 位元組。在 Aurora PostgreSQL 10 版中，其預設設定為 4,096 位元組。

文字大小上限

Aurora PostgreSQL 第 12 版及較低版本中，`track_activity_query_size` 的限制為 102,400 位元組。第 13 版及更高版本適用的最大值為 1 MB。

如果引擎傳回 1 MB 內容到 Performance Insights，則主控台只會顯示前 4 KB 的內容。如果您下載查詢，則會取得完整的 1 MB 內容。在此情況下，檢視和下載會傳回不同的位元組數。如需 `track_activity_query_size` 資料庫執行個體參數的更多相關資訊，請參閱 PostgreSQL 文件中的執行時間統計資料。

若要增加 SQL 文字大小，請提高 `track_activity_query_size` 限制。若要修改參數，請在與 Aurora PostgreSQL 資料庫執行個體關聯的參數群組中變更參數設定。

在執行個體使用預設參數群組時變更設定

1. 為適當的資料庫引擎和資料庫引擎版本建立新的資料庫執行個體參數群組。
2. 在新的數群組中設定參數。
3. 將新的參數群組與資料庫執行個體建立關聯。

如需設定資料庫執行個體參數的相關資訊，請參閱 修改 Amazon Aurora 中的資料庫參數群組中的參數。

在 Performance Insights 儀表板中檢視和下載更多 SQL 文字

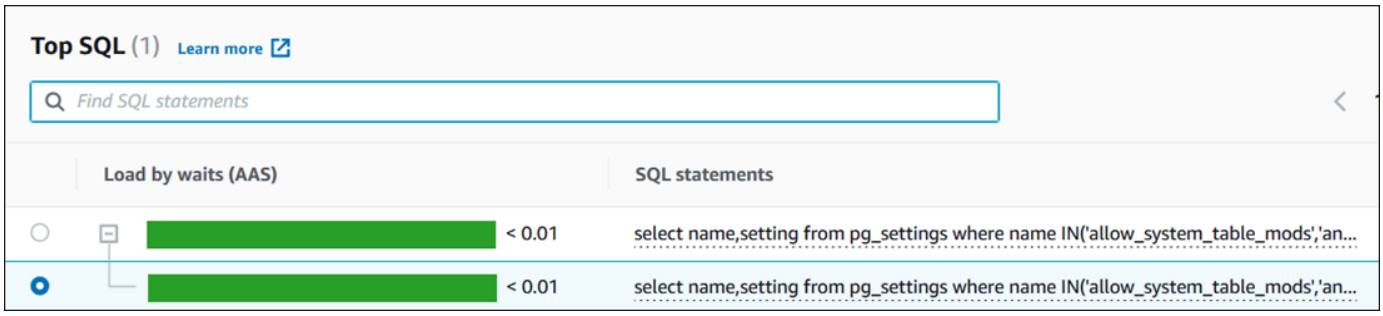
在 Performance Insights 儀表板中，您可以檢視和下載更多 SQL 文字。

在績效詳情儀表板內檢視更多 SQL 文字

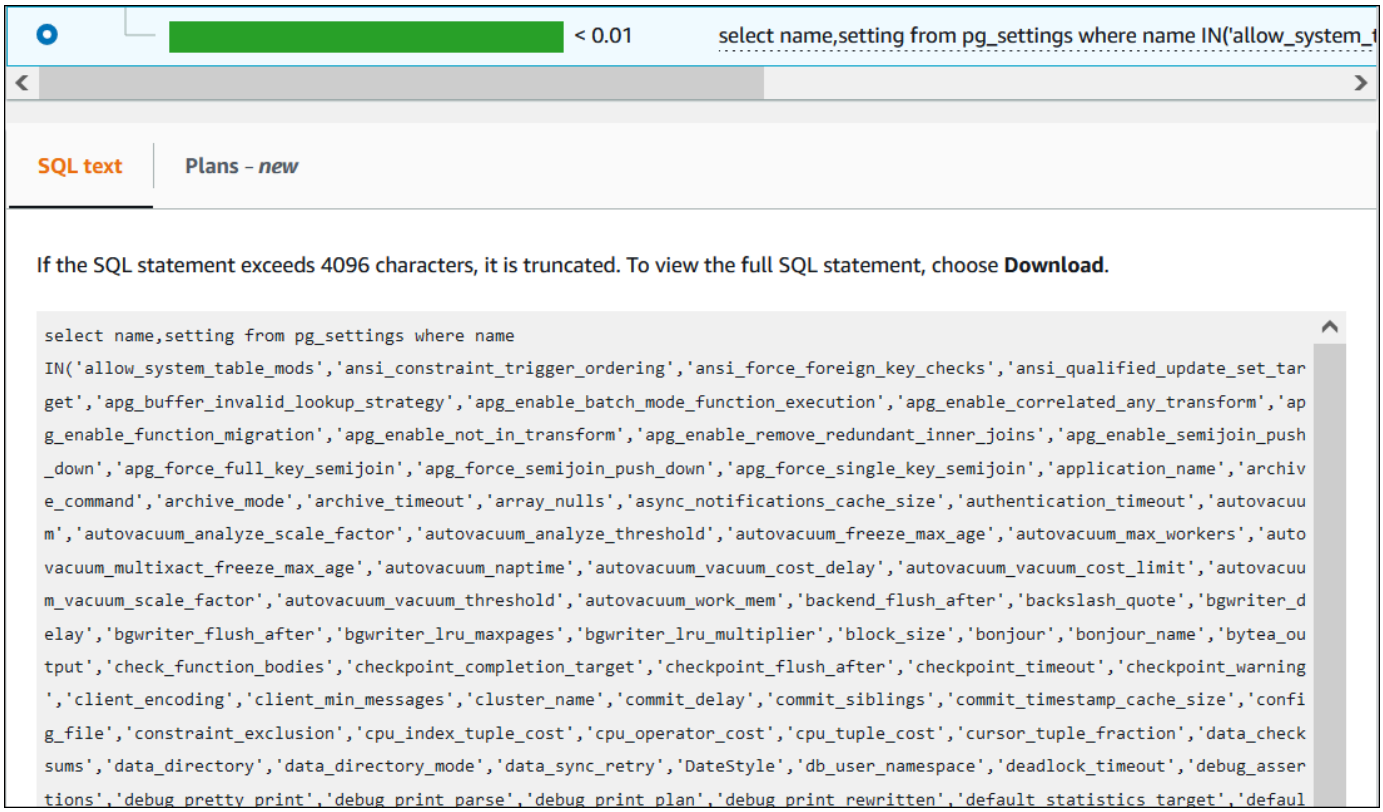
1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Performance Insights (績效詳情)。

3. 選擇資料庫執行個體。
4. 向下捲動至 Performance Insights 儀表板中的最高 SQL 索引標籤。
5. 選擇加號以展開 SQL 摘要，然後選擇其中一個摘要的子查詢。

   內含大於 500 位元組文字的 SQL 陳述式看起來與以下影像類似。



6. 向下捲動至 SQL text (SQL 文字) 索引標籤。



   績效詳情儀表板可以為每個 SQL 陳述式顯示高達 4,096 位元組。

7. (選用) 選擇複製來複製顯示的 SQL 陳述式，或選擇下載來下載 SQL 陳述式，檢視達到資料庫引擎限制的 SQL 文字。

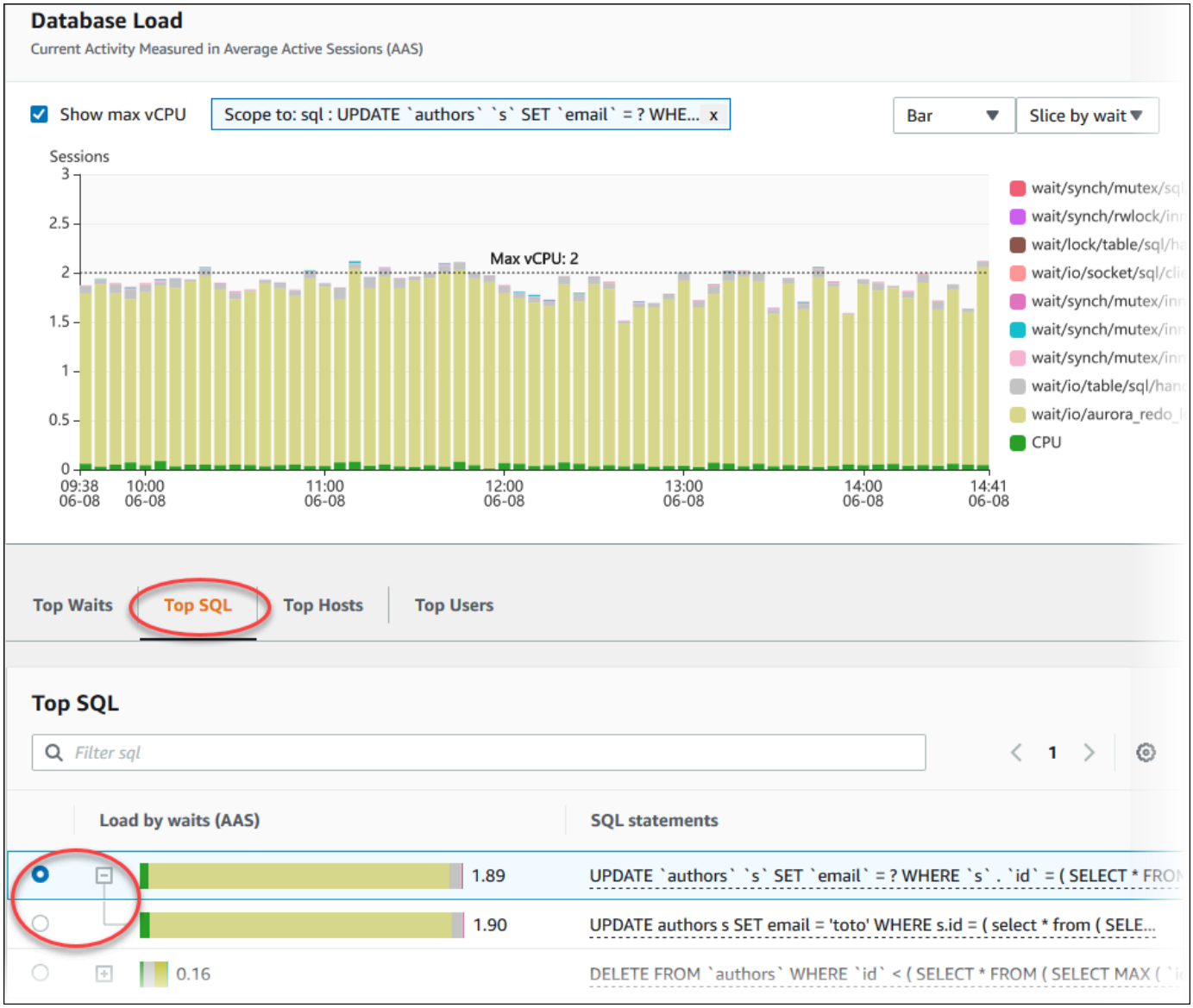
> **Note**
>
> 若要複製或下載 SQL 陳述式，停用彈出式封鎖程式。

在績效詳情儀表板中檢視 SQL 統計數字

在績效詳情儀表板中，Database load (資料庫負載) 圖表的 Top SQL (最高 SQL) 索引標籤會提供 SQL 統計數字。

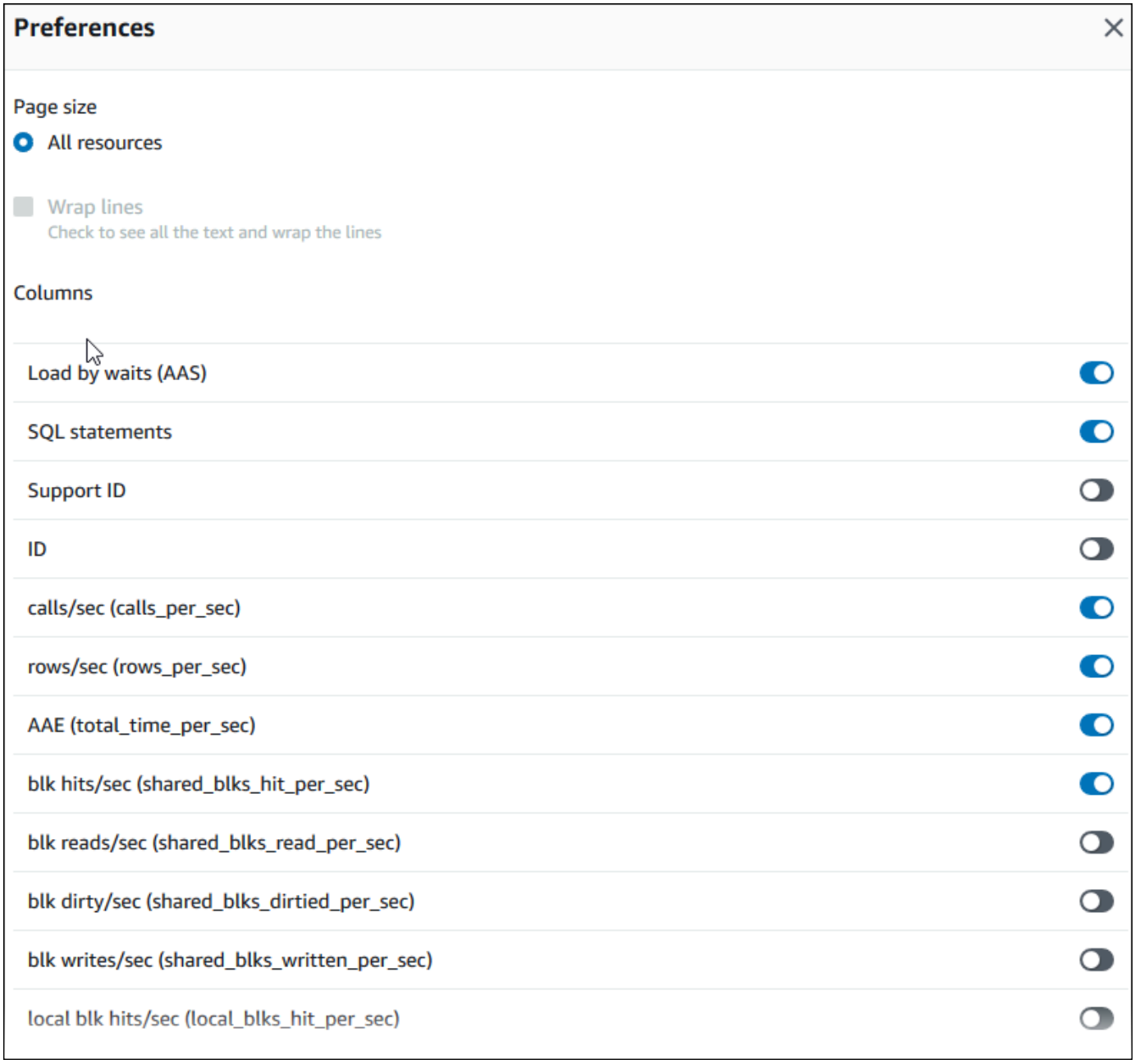
檢視 SQL 統計數字

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在左側導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 在頁面頂端，選擇您想要查看其 SQL 統計數字的資料庫。
4. 捲動到頁面底部並選擇 Top SQL (最高 SQL)。
5. 選擇個別陳述 (僅限 Aurora MySQL) 或摘要查詢。

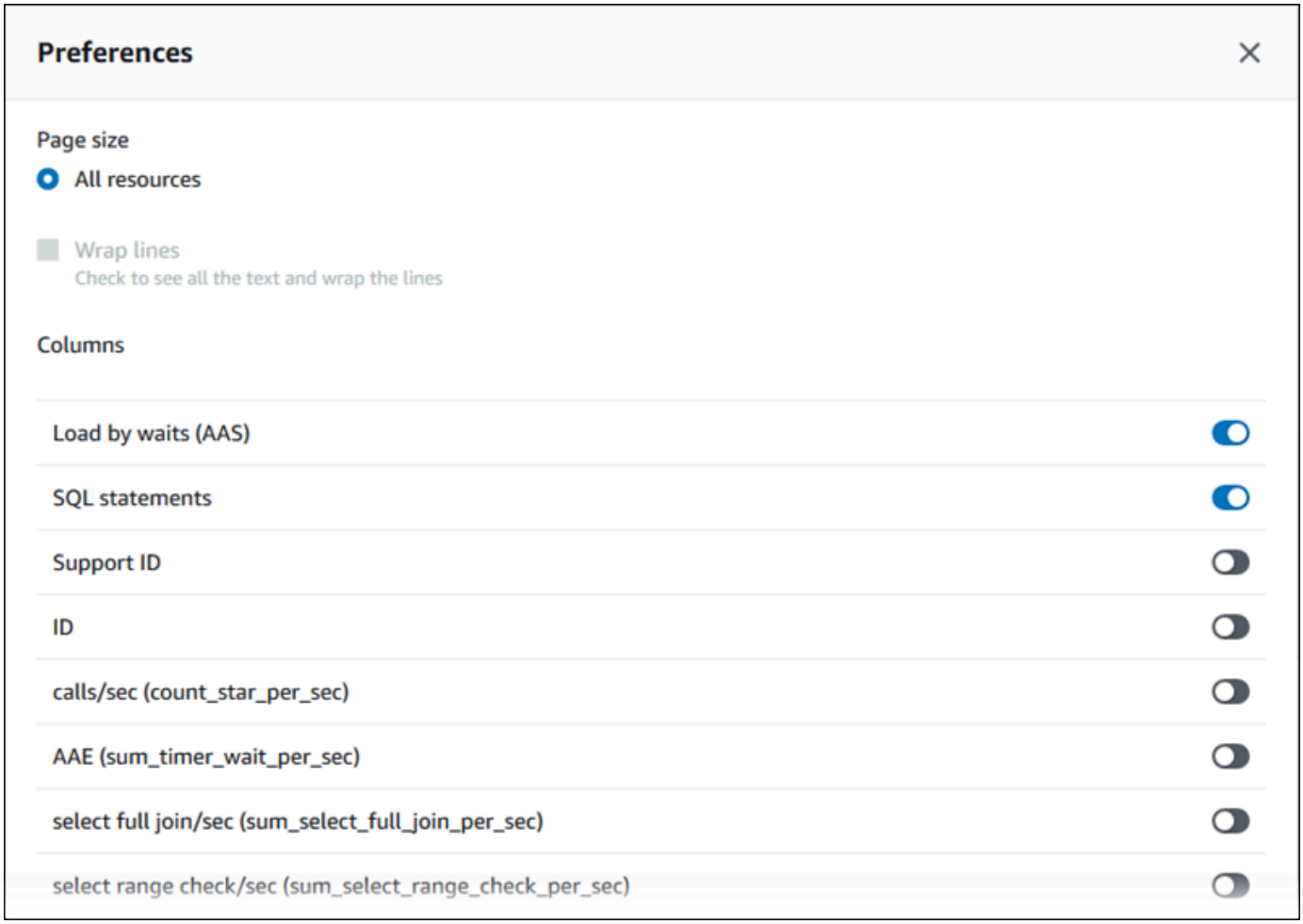


6. 選擇圖表右上角的齒輪圖示，以選擇要顯示的統計資料。如需 Amazon RDSAurora 引擎之 SQL 統計數字的說明，請參閱 [績效詳情的 SQL 統計數字]。

   下列範例顯示 Aurora PostgreSQL 的偏好設定。



下列範例顯示 Aurora MySQL 資料庫執行個體的偏好設定。



7. 選擇 Save (儲存) 儲存偏好設定。

   Top SQL (最高 SQL) 表格會重新整理。

檢視 Performance Insights 主動建議

Amazon RDS Performance Insights 會監控特定指標，以及透過分析特定資源可能有問題的層級，來自動建立閾值。當新的指標值在指定期間內超過預先定義的閾值時，Performance Insights 會產生主動建議。此建議有助於防止未來的資料庫效能影響。若要接收這些主動建議，您必須開啟具有付費方案保留期的 Performance Insights。

如需開啟績效詳情的詳細資訊，請參閱 開啟和關閉 Aurora 的 Performance Insights。如需 Performance Insights 的定價和資料保留詳細資訊，請參閱 Performance Insights 的定價和資料保留。

若要了解主動建議支援的區域、資料庫引擎和執行個體類別，請參閱 Performance Insights 功能的 Amazon Aurora 資料庫引擎、區域和執行個體類別支援。

您可以在建議詳細資訊頁面中檢視詳細分析和主動建議的建議調查。

如需建議的詳細資訊，請參閱 Amazon Aurora 的建議。

檢視主動建議的詳細分析

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，執行下列任一操作：
   - 選擇建議。

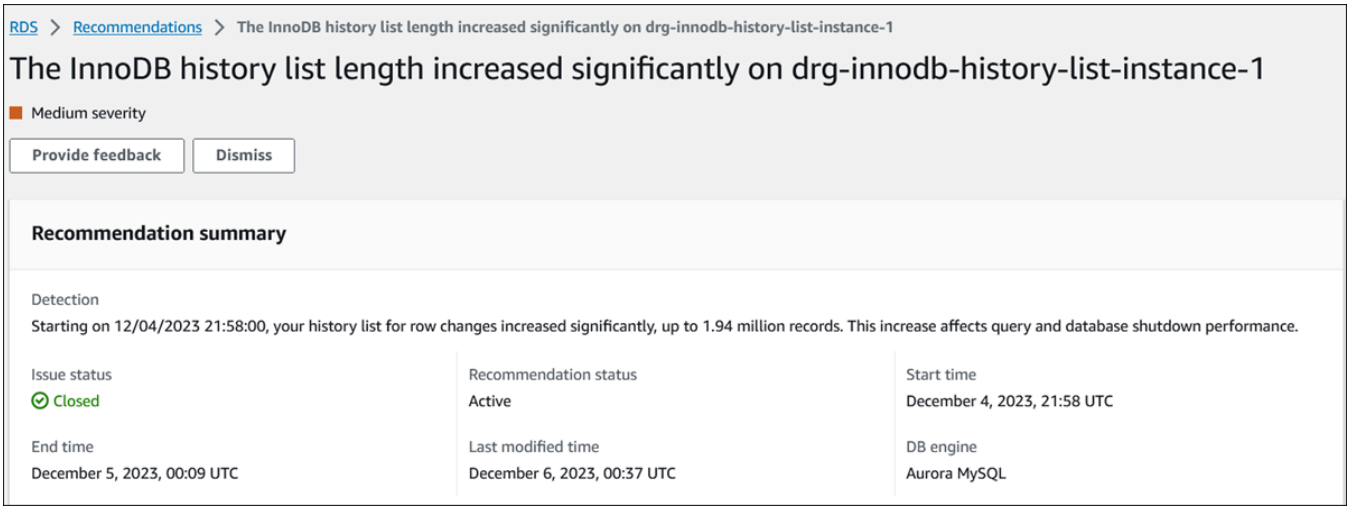
     建議頁面會顯示依您帳戶中所有資源的嚴重性排序的建議清單。
   - 選擇資料庫，然後在資料庫頁面中選擇資源的建議。

     建議索引標籤會顯示所選取資源的建議及其詳細資訊。
3. 尋找主動建議，然後選擇檢視詳細資訊。

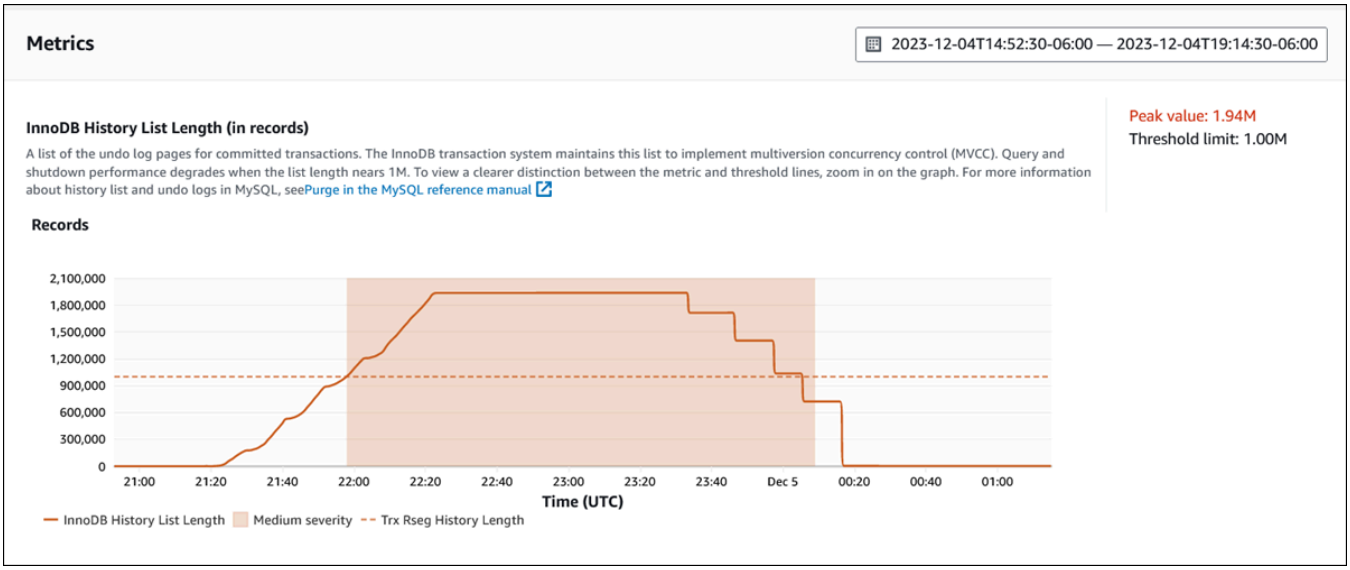
   建議詳細資訊頁面隨即出現。標題提供受影響資源的名稱，其中包含偵測到的問題和嚴重性。

   以下是建議詳細資訊頁面上的元件：

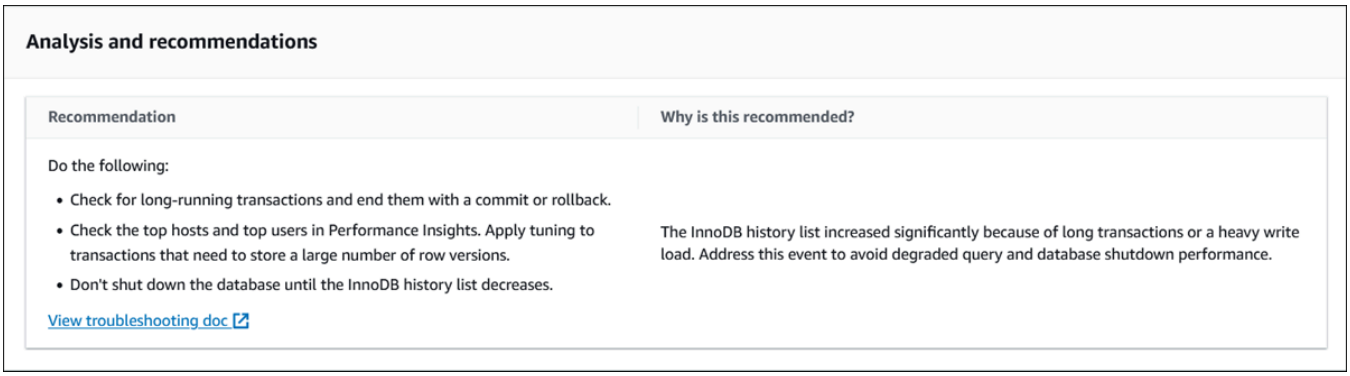
   - 建議摘要 – 偵測到的問題、建議和問題狀態、問題開始和結束時間、建議修改時間和引擎類型。



   - 指標 – 偵測到的問題的圖表。每個圖表會顯示由資源的基準行為和從問題開始時間回報的指標資料所決定的閾值。



- 分析和建議 – 建議和建議的原因。



您可以檢閱問題的原因，然後執行建議的動作來修正問題，或選擇右上角的關閉以關閉建議。

使用適用於 Aurora 的 Performance Insights API 來擷取指標

在啟用 Performance Insights 時，API 會提供對執行個體效能的可見性。Amazon CloudWatch Logs 提供用於 AWS 服務之販售監控指標的授權來源。

績效詳情提供以平均作用中工作階段 (AAS) 評估的資料庫負載特定網域檢視。此指標在 API 消費者看來是二維時間序列資料集。資料的時間維度提供查詢的時間範圍內各時間點的資料庫負載資料。每個時間點會根據請求的維度來分解整體負載，例如 `SQL`、`Wait-event`、`User`、或者 `Host`，在該時間點所測得。

Amazon RDS 績效詳情會監控您的 Amazon Aurora 叢集，讓您可分析資料庫效能並對其進行故障診斷。檢視績效詳情資料的一個方法就是使用 AWS 管理主控台。績效詳情也提供公有 API，讓您可以查詢自己的資料。您可以使用 API 執行下列動作：

- 將資料卸載至資料庫
- 將績效詳情資料新增至現有監控儀表板
- 建置監控工具

若要使用績效詳情 API，請在其中一個 Amazon RDS 資料庫執行個體上啟用績效詳情。如需啟用績效詳情的相關資訊，請參閱 開啟和關閉 Aurora 的 Performance Insights。如需績效詳情 API 的相關詳細資訊，請參閱 Amazon RDS 績效詳情 API 參考。

績效詳情 API 提供下列操作。

績效詳情動作	AWS CLI 命令	描述
CreatePerformanceAnalysisReport	aws pi create-performance-analysis-report	針對資料庫執行個體的特定時間區間建立績效分析報告。結果是 AnalysisReportId ，也是報告的唯一識別符。
DeletePerformanceAnalysisReport	aws pi delete-performance-analysis-report	刪除績效分析報告。
DescribeDimensionKeys	aws pi describe-dimension-keys	針對特定時段，擷取其指標的前 N 個維度金鑰。
GetDimensionKeyDetails	aws pi get-dimension-key-details	擷取資料庫執行個體或資料來源之指定維度群組的屬性。比方說，如果指定 SQL ID，且有維度詳細資訊可用，則 GetDimensionKeyDetails 會擷取與此 ID db.sql.statement 相關聯之維度的完整文字。這項操作很有用，因為 GetResourceMetrics

績效詳情動作	AWS CLI 命令	描述
		和 DescribeDimensionKeys 不支援擷取大量的 SQL 陳述式文字。
GetPerformanceAnalysisReport	aws pi get-performance-analysis-report	擷取報告，包括報告洞見。結果包括報告狀態、報告 ID、報告時間詳細資訊、洞見和建議。
GetResourceMetadata	aws pi get-resource-metadata	檢索不同功能的中繼資料。例如，中繼資料可能指出特定資料庫執行個體上某項功能已開啟或關閉。
GetResourceMetrics	aws pi get-resource-metrics	擷取一組資料來源某個時段的績效詳情指標。您可以提供特定維度群組和維度，以及為每個群組提供彙總和篩選條件。
ListAvailableResourceDimensions	aws pi list-available-resource-dimensions	檢索指定執行個體上每個指定指標類型可查詢的維度。
ListAvailableResourceMetrics	aws pi list-available-resource-metrics	檢索可為指定資料庫執行個體查詢的指定指標類型中所有可用的指標。
ListPerformanceAnalysisReports	aws pi list-performance-analysis-reports	擷取資料庫執行個體可用的所有分析報告。報告會根據每個報告的開始時間列出。
ListTagsForResource	aws pi list-tags-for-resource	列出所有新增至資源的中繼資料標籤。清單包括標籤的名稱和值。

績效詳情動作	AWS CLI 命令	描述
TagResource	aws pi tag-resource	將中繼資料標籤新增到 Amazon RDS 資源。標籤包括一個名稱和一個值。
UntagResource	aws pi untag-resource	從資源移除中繼資料標籤。

如需擷取 Performance Insights 時間序列指標和 AWS CLI 範例的詳細資訊，請參閱下列主題。

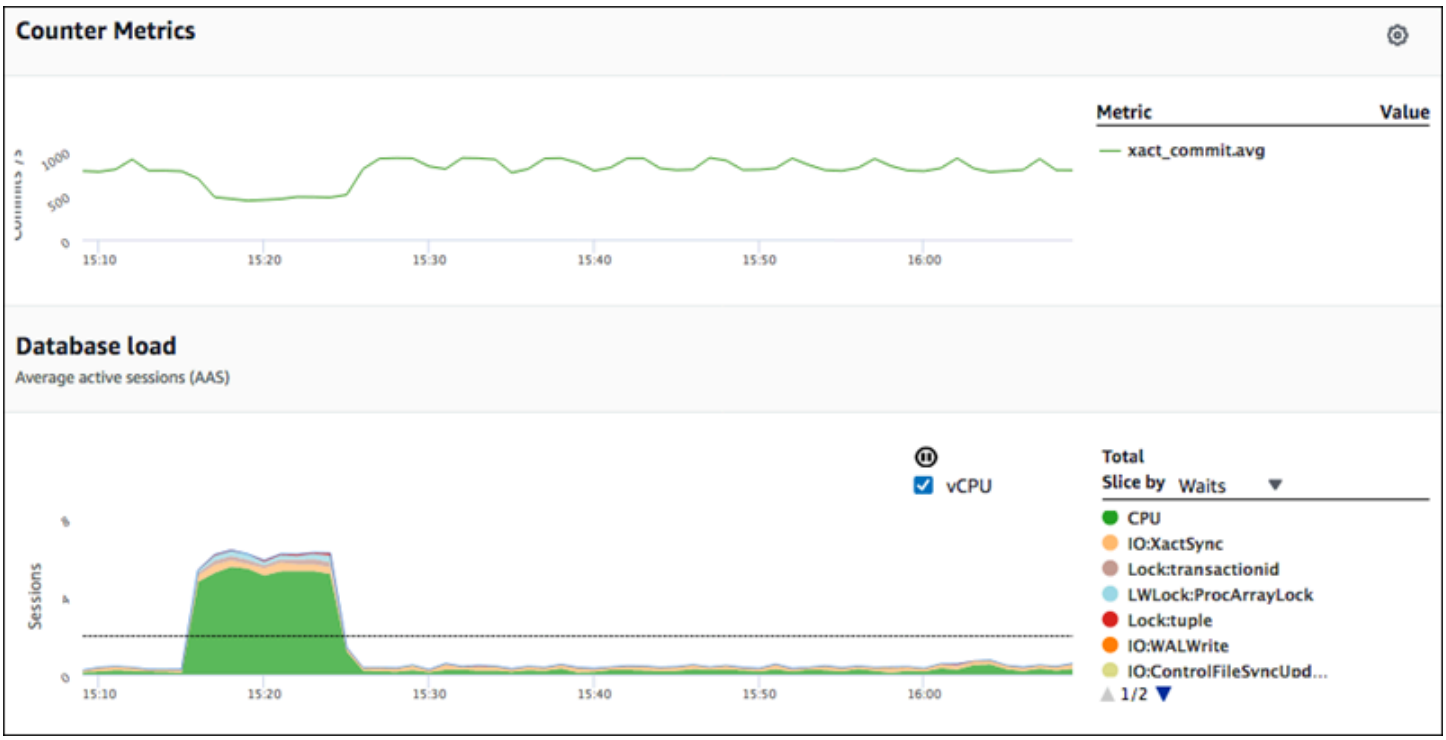
主題

- 擷取 Performance Insights 的時間序列指標
- AWS CLI 績效詳情的範例

擷取 Performance Insights 的時間序列指標

`GetResourceMetrics` 操作會從績效詳情資料中擷取一或多個時間時間序列指標。`GetResourceMetrics` 需要指標和時間間隔，並傳回含資料點清單的回應。

例如，AWS 管理主控台 會將 `GetResourceMetrics` 用於填入 Counter Metrics (計數器指標) 圖表和 Database Load (資料庫負載) 圖表，如下圖所示。



GetResourceMetrics 傳回的所有指標，除 db.load 之外，皆為標準的時間序列指標。此指標會顯示在 Database Load (資料庫負載) 圖表中。db.load 指標與其他時間序列指標不同，因為您可以將它分為名為維度的子元件。在先前的影像中，db.load 已被細分，分組依據為組成 db.load 的等待狀態。

> **Note**
>
> GetResourceMetrics 也可以傳回 db.sampleload 指標，但 db.load 指標適用於大部分情況。

如需 GetResourceMetrics 所傳回指標的相關資訊，請參閱Performance Insights 計數器指標。

這些指標支援下列計算：

- 平均值 – 指標在一段時間內的平均值。將 .avg 附加至指標名稱。
- 最小值 – 指標在一段時間內的最小值。將 .min 附加至指標名稱。
- 最大值 – 指標在一段時間內的最大值。將 .max 附加至指標名稱。
- 總和 – 指標值在一段時間內的總和。將 .sum 附加至指標名稱。
- 取樣計數 – 在一段時間內收集指標的次數。將 .sample_count 附加至指標名稱。

例如，假設收集指標的時間為 300 秒 (5 分鐘)，且每分鐘收集一次指標。每分鐘的值為 1、2、3、4 和 5。在此情況下，會傳回下列計算：

- 平均值 – 3
- 最小值 – 1
- 最大值 – 5
- 總和 – 15
- 取樣計數 – 5

如需使用 `get-resource-metrics` AWS CLI 命令的相關資訊，請參閱 `get-resource-metrics`。

對於 `--metric-queries` 選項，請指定您要取得結果的一或多個查詢。每個查詢的組成為必要的 `Metric` 和選用的 `GroupBy` 及 `Filter` 參數。以下是 `--metric-queries` 選項規格的範例。

```
{
   "Metric": "string",
   "GroupBy": {
     "Group": "string",
     "Dimensions": ["string", ...],
     "Limit": integer
   },
   "Filter": {"string": "string"
     ...}
```

AWS CLI 績效詳情的範例

在下列各節中，進一步了解績效詳情的 AWS Command Line Interface (AWS CLI)，並使用 AWS CLI 範例。

主題

- for Performance Insights AWS CLI 的內建說明
- 擷取計數器指標
- 擷取最久等待事件的資料庫負載平均值
- 擷取最高 SQL 的資料庫負載平均值
- 擷取依據 SQL 篩選的資料庫負載平均值
- 擷取 SQL 陳述式的完整文字

- 建立一段時間區間的績效分析報告
- 擷取績效分析報告
- 列出資料庫執行個體的所有績效分析報告
- 刪除績效分析報告
- 將標籤新增至績效分析報告
- 列出績效分析報告的所有標籤
- 從績效分析報告中刪除標籤

for Performance Insights AWS CLI 的內建說明

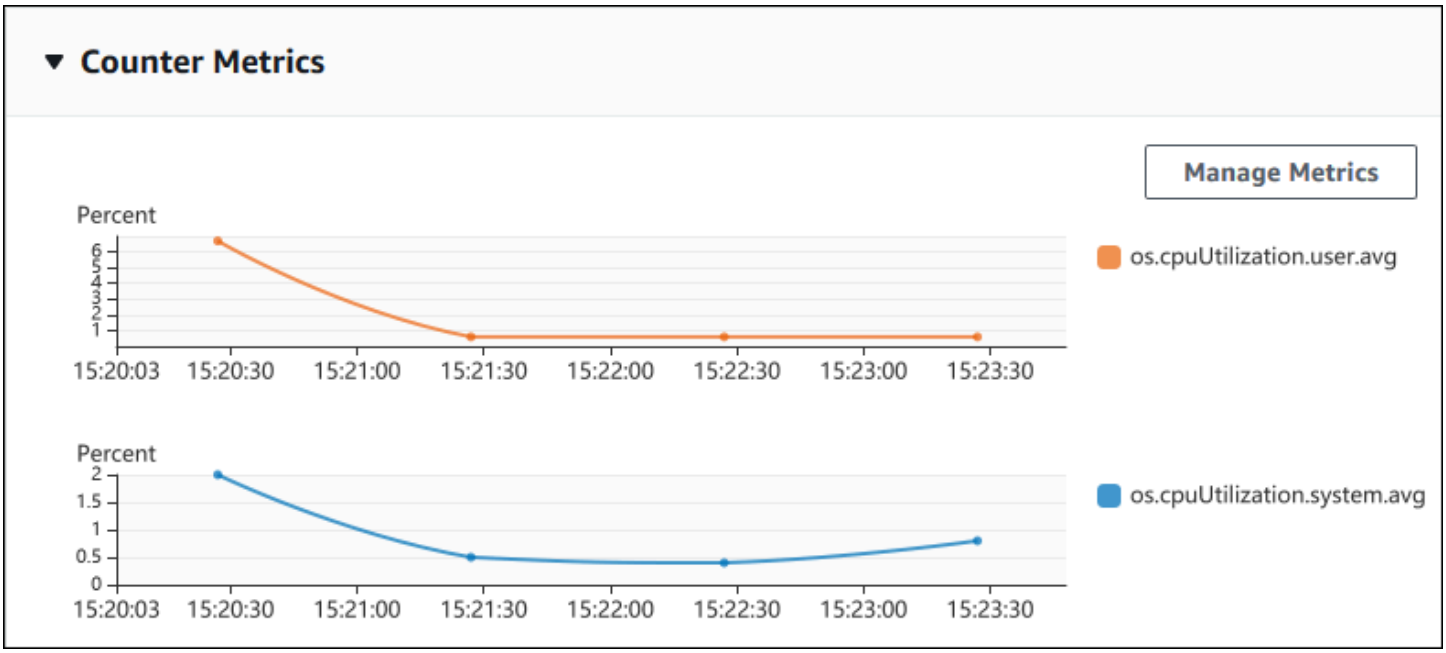
您可以使用 AWS CLI檢視績效詳情資料。您可以在 AWS CLI 命令列中輸入以下內容，以檢視 Performance Insights 命令的說明。

```
aws pi help
```

如果您尚未 AWS CLI 安裝 ，請參閱AWS CLI 《 使用者指南》中的安裝 AWS CLI ，以取得安裝相關資訊。

擷取計數器指標

下列螢幕擷取畫面顯示 AWS 管理主控台中的兩個計數器指標圖表。



下列範例示範如何收集 AWS 管理主控台 用來產生兩個計數器指標圖表的相同資料。

針對 Linux、macOS 或 Unix：

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

在 Windows 中：

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

您也可以透過指定 `--metrics-query` 選項的檔案來提高命令的可讀性。以下範例會將名為 query.json 的檔案用於此選項。此檔案的內容如下。

```
[
    {
        "Metric": "os.cpuUtilization.user.avg"
    },
    {
        "Metric": "os.cpuUtilization.idle.avg"
    }
]
```

執行下列命令來使用檔案。

對於 Linux、macOS 或 Unix：

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
```

```
    --start-time 2018-10-30T00:00:00Z \
    --end-time   2018-10-30T01:00:00Z \
    --period-in-seconds 60 \
    --metric-queries file://query.json
```

在 Windows 中：

```
aws pi get-resource-metrics ^
    --service-type RDS ^
    --identifier db-ID ^
    --start-time 2018-10-30T00:00:00Z ^
    --end-time   2018-10-30T01:00:00Z ^
    --period-in-seconds 60 ^
    --metric-queries file://query.json
```

先前的範例會為選項指定下列值：

- `--service-type` – RDS for Amazon RDS
- `--identifier`– 資料執行個體的資源 ID
- `--start-time` 和 `--end-time` – 要查詢期間的 ISO 8601 `DateTime` 值，支援多種格式

它會查詢一小時的時間範圍：

- `--period-in-seconds`–`60` 適用於每分鐘的查詢
- `--metric-queries`– 兩個查詢的陣列，一個指標剛好一個查詢。

  此指標名稱會使用點將指標分類在實用的類別，其中最後一個元素則做為函數。在此範例中，此函數是每個查詢的 `avg`。如同 Amazon CloudWatch，支援的函數是 `min`、`max`、`total` 和 `avg`。

回應看起來類似以下的內容。

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                "Metric": "os.cpuUtilization.user.avg" //Metric1
            },
```

```
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 10.0
                }
                //... 60 datapoints for the os.cpuUtilization.user.avg metric
            ]
        },
        {
            "Key": {
                "Metric": "os.cpuUtilization.idle.avg" //Metric2
            },
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 12.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 13.5
                },
                //... 60 datapoints for the os.cpuUtilization.idle.avg metric
            ]
        }
    ] //end of MetricList
} //end of response
```

回應具有 Identifier、AlignedStartTime 和 AlignedEndTime。--period-in-seconds 值為 60，開始和結束時間皆一致使用分鐘。如果 --period-in-seconds 是 3600，開始和結束時間則會一致使用小時。

回應中的 MetricList 擁有許多項目，每個都包含 Key 和 DataPoints 項目。每個 DataPoint 都有 Timestamp 和 Value。每個 Datapoints 清單有 60 個資料點，因為查詢是適用於一小時中的每

分鐘資料，內含 Timestamp1/Minute1、Timestamp2/Minute2 等，最多可達 Timestamp60/Minute60。

因為此查詢是適用於兩個不同的計數器指標，回應 MetricList 中會有兩個元素。

擷取最久等待事件的資料庫負載平均值

下列範例與 AWS 管理主控台 用來產生堆疊區域折線圖的查詢相同。此範例會使用根據前七個最久的等待事件而區分的負載來擷取前一小時的 db.load.avg。此命令與 擷取計數器指標 中的命令相同。然而，查詢 query.json 檔案有以下內容。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 7 }
    }
]
```

執行下列命令。

對於 Linux、macOS 或 Unix：

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

在 Windows 中：

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

此範例會指定 db.load.avg 指標與前七個最久等待事件的 GroupBy。如需此範例有效值的詳細資訊，請參閱績效詳情 API 參考中的DimensionGroup。

回應看起來類似以下的內容。

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                //A Metric with no dimensions. This is the total db.load.avg
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 0.5166666666666667
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.38333333333333336
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 0.26666666666666666
                }
                //... 60 datapoints for the total db.load.avg key
            ]
        },
        {
            "Key": {
                //Another key. This is db.load.avg broken down by CPU
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.name": "CPU",
                    "db.wait_event.type": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 0.35
                },
```

```
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.15
                },
                //... 60 datapoints for the CPU key
            ]
        },
        //... In total we have 8 key/datapoints entries, 1) total, 2-8) Top Wait Events
    ] //end of MetricList
} //end of response
```

在此回應中，MetricList 中有八個項目。有一個項目適用於總計 db.load.avg，有七個項目，分別適用於根據前七個最久等待事件區份的 db.load.avg。與第一個範例不同，因為其中有分組維度，每個指標分組都必須有一個索引鍵。每個指標不能只有一個索引鍵，如同基本計數器指標使用案例。

擷取最高 SQL 的資料庫負載平均值

以下範例會根據前 10 個 SQL 陳述式來分組 db.wait_events。SQL 陳述式有兩個不同的分組：

- db.sql– 完整的 SQL 陳述式，例如 `select * from customers where customer_id = 123`
- db.sql_tokenized– 字符化的 SQL 陳述式，例如 `select * from customers where customer_id = ?`

分析資料庫效能時，將僅參數不同的 SQL 陳述式視為單一邏輯項目可能會很有幫助。因此，您可以在查詢時使用 db.sql_tokenized。然而，特別是在您對說明計畫感興趣時，使用參數來檢查完整 SQL 陳述式並依據 db.sql 來查詢分組有時候會更有幫助。這是字符化與完整 SQL 之間的父子關係，內含使用相同字符化 SQL (父項) 分組的多個完整 SQL (子項)。

此範例中的命令與 擷取最久等待事件的資料庫負載平均值 中的命令類似。然而，查詢 query.json 檔案有以下內容。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.sql_tokenized", "Limit": 10 }
    }
]
```

以下範例使用 db.sql_tokenized。

對於 Linux、macOS 或 Unix：

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-29T00:00:00Z \
   --end-time   2018-10-30T00:00:00Z \
   --period-in-seconds 3600 \
   --metric-queries file://query.json
```

在 Windows 中：

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-29T00:00:00Z ^
   --end-time   2018-10-30T00:00:00Z  ^
   --period-in-seconds 3600 ^
   --metric-queries file://query.json
```

此範例的查詢會持續 24 小時，其中 period-in-seconds 為一小時。

此範例會指定 db.load.avg 指標與前七個最久等待事件的 GroupBy。如需此範例有效值的詳細資訊，請參閱績效詳情 API 參考中的DimensionGroup。

回應看起來類似以下的內容。

```
{
    "AlignedStartTime": 1540771200.0,
    "AlignedEndTime": 1540857600.0,
    "Identifier": "db-XXX",

    "MetricList": [ //11 entries in the MetricList
        {
            "Key": { //First key is total
                "Metric": "db.load.avg"
            }
            "DataPoints": [ //Each DataPoints list has 24 per-hour Timestamps and a
 value
                {
                    "Value": 1.6964980544747081,
                    "Timestamp": 1540774800.0
```

```
                },
                //... 24 datapoints
            ]
        },
        {
            "Key": { //Next key is the top tokenized SQL
                "Dimensions": {
                    "db.sql_tokenized.statement": "INSERT INTO authors (id,name,email)
 VALUES\n( nextval(?)  ,?,?)",
                    "db.sql_tokenized.db_id": "pi-2372568224",
                    "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE"
                },
                "Metric": "db.load.avg"
            },
            "DataPoints": [ //... 24 datapoints
            ]
        },
        // In total 11 entries, 10 Keys of top tokenized SQL, 1 total key
    ] //End of MetricList
} //End of response
```

此回應在 MetricList 中有 11 個項目 (1 個總計，前 10 個字符化的 SQL)，每個項目擁有 24 個每小時 DataPoints。

對於字符化的 SQL，每個維度清單中有三個項目：

- db.sql_tokenized.statement– 字符化的 SQL 陳述式。
- db.sql_tokenized.db_id – 參考 SQL 所用的原生資料庫 ID，或是無法使用原生資料庫 ID 時，績效詳情為您產生的合成 ID。此範例會傳回 pi-2372568224 合成 ID。
- db.sql_tokenized.id– 績效詳情中查詢的 ID。

  在 中 AWS 管理主控台，此 ID 稱為支援 ID。其名稱如下，因為 ID 是 AWS Support 可以檢查的資料，以協助您對資料庫的問題進行故障診斷。 非常重視資料 AWS 的安全性和隱私權，而且幾乎所有資料都會以您的 AWS KMS 金鑰加密存放。因此，內部沒有人 AWS 可以查看此資料。在先前的範例中，tokenized.statement 和 tokenized.db_id 都同時會以加密的形式存放。如果您的資料庫發生問題， AWS Support 可以透過參考支援 ID 來協助您。

進行查詢時，在 Group 中指定 GroupBy 可能會讓您省下不少心力。然而，如需對已傳回的資料進行更精細的控制，請指定維度的清單。例如，如果所需的是 db.sql_tokenized.statement，則可將 Dimensions 屬性新增至 query.json 檔案。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": {
            "Group": "db.sql_tokenized",
            "Dimensions":["db.sql_tokenized.statement"],
            "Limit": 10
        }
    }
]
```

擷取依據 SQL 篩選的資料庫負載平均值

先前的影像顯示已選取特定的查詢，最高平均作用中工作階段堆疊區域折線圖的範圍仍涵蓋至該查詢。雖然此查詢仍適用於前七個整體等待事件，系統仍會將回應值篩選出來。此篩選條件會在工作階段符合特定篩選條件時，才進行篩選。

此範例中的對應 API 查詢與 擷取最高 SQL 的資料庫負載平均值 中的命令類似。然而，查詢 query.json 檔案有以下內容。

```
[
 {
```

```
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 5  },
        "Filter": { "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE" }
    }
]
```

對於 Linux、macOS 或 Unix：

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

在 Windows 中：

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

回應看起來類似以下的內容。

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1556215200.0,
    "MetricList": [
        {
            "Key": {
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.4878117913832196
                },
                {
```

```
                    "Timestamp": 1556222400.0,
                    "Value": 1.192823803967328
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
                    "db.wait_event.name": "wait/io/aurora_redo_log_flush"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.1360544217687074
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 1.058051341890315
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
                    "db.wait_event.name": "wait/io/table/sql/handler"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.16241496598639457
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05163360560093349
                }
            ]
        },
        {
```

```
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/
aurora_lock_thread_slot_futex"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.11479591836734694
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.013127187864644107
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "CPU",
                    "db.wait_event.name": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.05215419501133787
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05805134189031505
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/lock_wait_mutex"
```

```
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.017573696145124718
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.002333722287047841
                }
            ]
        }
    ],
    "AlignedEndTime": 1556222400.0
} //end of response
```

在此回應中，系統會根據 query.json file.檔案中指定的字符化 SQL AKIAIOSFODNN7EXAMPLE 的影響程度來篩選所有值。此索引鍵遵循的順序可能會與不含篩選條件的查詢不同，因為這是影響篩選 SQL 的前五個等待事件。

擷取 SQL 陳述式的完整文字

下列範例會擷取資料庫執行個體 db-10BCD2EFGHIJ3KL4M5NO6PQRS5 之 SQL 陳述式的完整文字。`--group` 即為 `db.sql`，而 `--group-identifier` 即為 `db.sql.id`。在此範例中，*my-sql-id* 代表藉由呼叫 `pi get-resource-metrics` 或 `pi describe-dimension-keys` 擷取的 SQL ID。

執行下列命令。

對於 Linux、macOS 或 Unix：

```
aws pi get-dimension-key-details \
   --service-type RDS \
   --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 \
   --group db.sql \
   --group-identifier my-sql-id \
   --requested-dimensions statement
```

在 Windows 中：

```
aws pi get-dimension-key-details ^
```

```
    --service-type RDS ^
    --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 ^
    --group db.sql ^
    --group-identifier my-sql-id ^
    --requested-dimensions statement
```

在此範例中，維度詳細資訊可供使用。因此，績效詳情會擷取 SQL 陳述式的完整文字，而不會將其截斷。

```
{
    "Dimensions":[
    {
        "Value": "SELECT e.last_name, d.department_name FROM employees e, departments d
 WHERE e.department_id=d.department_id",
        "Dimension": "db.sql.statement",
        "Status": "AVAILABLE"
    },
    ...
    ]
}
```

建立一段時間區間的績效分析報告

下列範例會以 db-loadtest-0 資料庫的 1682969503 開始時間和 1682979503 結束時間建立績效分析報告。

```
aws pi create-performance-analysis-report \
        --service-type RDS \
        --identifier db-loadtest-0 \
        --start-time 1682969503 \
        --end-time 1682979503 \
        --region us-west-2
```

回應為 report-0234d3ed98e28fb17，是報告的唯一識別碼。

```
{
   "AnalysisReportId": "report-0234d3ed98e28fb17"
}
```

擷取績效分析報告

下列範例擷取 report-0d99cc91c4422ee61 報告的分析報告詳細資訊。

```
aws pi get-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
--region us-west-2
```

回應提供報告狀態、ID、時間詳細資訊和洞見。

```
        {
    "AnalysisReport": {
        "Status": "Succeeded",
        "ServiceType": "RDS",
        "Identifier": "db-loadtest-0",
        "StartTime": 1680583486.584,
        "AnalysisReportId": "report-0d99cc91c4422ee61",
        "EndTime": 1680587086.584,
        "CreateTime": 1680587087.139,
        "Insights": [
           ... (Condensed for space)
        ]
    }
}
```

列出資料庫執行個體的所有績效分析報告

下列範例會列出 db-loadtest-0 資料庫所有可用的績效分析報告。

```
aws pi list-performance-analysis-reports \
--service-type RDS \
--identifier db-loadtest-0 \
--region us-west-2
```

回應將列出所有報告，其中包含報告 ID，狀態和時間區間的詳細資訊。

```
{
    "AnalysisReports": [
        {
            "Status": "Succeeded",
            "EndTime": 1680587086.584,
            "CreationTime": 1680587087.139,
```

```
            "StartTime": 1680583486.584,
            "AnalysisReportId": "report-0d99cc91c4422ee61"
        },
        {
            "Status": "Succeeded",
            "EndTime": 1681491137.914,
            "CreationTime": 1681491145.973,
            "StartTime": 1681487537.914,
            "AnalysisReportId": "report-002633115cc002233"
        },
        {
            "Status": "Succeeded",
            "EndTime": 1681493499.849,
            "CreationTime": 1681493507.762,
            "StartTime": 1681489899.849,
            "AnalysisReportId": "report-043b1e006b47246f9"
        },
        {
            "Status": "InProgress",
            "EndTime": 1682979503.0,
            "CreationTime": 1682979618.994,
            "StartTime": 1682969503.0,
            "AnalysisReportId": "report-01ad15f9b88bcbd56"
        }
    ]
}
```

刪除績效分析報告

下列範例會刪除 db-loadtest-0 資料庫的績效分析報告。

```
aws pi delete-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
--region us-west-2
```

將標籤新增至績效分析報告

下列範例會將帶有金鑰 name 和值 test-tag 的標籤新增至 report-01ad15f9b88bcbd56 報告。

```
aws pi tag-resource \
```

```
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tags Key=name,Value=test-tag \
--region us-west-2
```

列出績效分析報告的所有標籤

下列範例會列出 report-01ad15f9b88bcbd56 報告的所有標籤。

```
aws pi list-tags-for-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--region us-west-2
```

回應會列出新增至報告的所有標籤值和金鑰：

```
{
    "Tags": [
        {
            "Value": "test-tag",
            "Key": "name"
        }
    ]
}
```

從績效分析報告中刪除標籤

下列範例會從 report-01ad15f9b88bcbd56 報告刪除 name 標籤。

```
aws pi untag-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tag-keys name \
--region us-west-2
```

刪除標籤之後，呼叫 list-tags-for-resource API 不會列出此標籤。

使用 AWS CloudTrail 記錄績效詳情呼叫

績效詳情與 AWS CloudTrail 服務搭配運作，此服務會記錄使用者、角色或績效詳情中 AWS 服務所採取的動作。CloudTrail 會將績效詳情的所有 API 呼叫擷取為事件。此擷取包含來自 Amazon RDS 主控台的呼叫，以及從程式碼呼叫到績效詳情 API 操作的呼叫。

如果您建立線索，就可以讓 CloudTrail 事件持續交付至 Amazon S3 儲存貯體，包括績效詳情的事件。即使您未設定權仗，依然可以透過 CloudTrail 主控台中的 Event history (事件歷史記錄) 檢視最新事件。使用由 CloudTrail 收集的資訊，您就可以判斷特定詳細資訊。此資訊包括對績效詳情提出的請求、提出請求的 IP 地址、何人提出請求，以及提出請求的時間。此外也包括其他詳細資訊。

若要進一步了解 CloudTrail，請參閱「AWS CloudTrail 使用者指南」。

在 CloudTrail 中使用績效詳情資訊

當您建立帳戶時，系統即會在 AWS 帳戶中啟用 CloudTrail。當績效詳情中發生活動時，該活動會記錄在 CloudTrail 事件中，其他 AWS 服務事件則記錄於 CloudTrail 主控台的事件歷程記錄中。您可以檢視、搜尋和下載 AWS 帳戶的最新事件。如需更多詳細資訊，請參閱 AWS CloudTrail 使用者指南中的使用 CloudTrail 事件歷史記錄檢視事件。

若要持續記錄 AWS 帳戶的事件 (包括績效詳情事件)，請建立線索。線索能讓 CloudTrail 將日誌檔案交付至 Amazon S3 儲存貯體。根據預設，當您在主控台建立線索時，線索會套用到所有 AWS 區域。權杖會記錄來自 AWS 分割區中所有 AWS 區域的事件，然後將記錄檔案交付到您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中所收集的事件資料。如需更多詳細資訊，請參閱 AWS CloudTrail 使用者指南中的以下主題：

- 建立追蹤的概觀
- CloudTrail 支援的服務和整合
- 設定 CloudTrail 的 Amazon SNS 通知
- 從多個區域接收 CloudTrail 日誌檔案，以及從多個帳戶接收 CloudTrail 日誌檔案

所有績效詳情操作都是由 CloudTrail 所記錄並記載在績效詳情 API 參考中。例如，對 `DescribeDimensionKeys` 和 `GetResourceMetrics` 操作的呼叫都會在 CloudTrail 日誌檔案中產生項目。

每一筆事件或日誌項目都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 該請求是否使用根或 IAM 使用者憑證提出。

- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 該請求是否由另一項 AWS 服務提出。

如需詳細資訊，請參閱 CloudTrail userIdentity Element。

績效詳情日誌檔案項目

權杖是一種組態，能讓事件以記錄檔案的形式交付到您指定的 Amazon S3 儲存貯體。CloudTrail 日誌檔案包含一個或多個日誌項目。「事件」代表來自任何來源的單一請求。每個事件皆包含請求操作、操作日期和時間、請求參數等相關資訊。CloudTrail 日誌檔案並非依公有 API 呼叫的堆疊追蹤排序，因此不會以任何特定順序出現。

以下範例顯示的 CloudTrail 日誌項目會示範 `GetResourceMetrics` 操作：

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
         "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T19:28:46Z",
    "eventSource": "pi.amazonaws.com",
    "eventName": "GetResourceMetrics",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "72.21.198.67",
    "userAgent": "aws-cli/1.16.240 Python/3.7.4 Darwin/18.7.0 botocore/1.12.230",
    "requestParameters": {
        "identifier": "db-YTDU5J5V66X7CXSCVDFD2V3SZM",
        "metricQueries": [
            {
                "metric": "os.cpuUtilization.user.avg"
            },
            {
                "metric": "os.cpuUtilization.idle.avg"
            }
        ],
        "startTime": "Dec 18, 2019 5:28:46 PM",
        "periodInSeconds": 60,
```

```
        "endTime": "Dec 18, 2019 7:28:46 PM",
        "serviceType": "RDS"
    },
    "responseElements": null,
    "requestID": "9ffbe15c-96b5-4fe6-bed9-9fccff1a0525",
    "eventID": "08908de0-2431-4e2e-ba7b-f5424f908433",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

Performance Insights API 和界面 VPC 端點 (AWS PrivateLink)

您可以使用 在 VPC 和 Amazon RDS Performance Insights 之間 AWS PrivateLink 建立私有連線。您可以像在 VPC 中一樣存取績效詳情，無需使用網際網路閘道、NAT 裝置、VPN 連接或 Direct Connect 連線。VPC 中的執行個體無需公有 IP 位址，即可存取 Performance Insights。

您可以建立由 AWS PrivateLink提供支援的介面端點來建立此私有連線。我們會在您為介面端點啟用的每個子網中建立端點網路介面。這些是請求者管理的網路界面，可作為目的地為 Performance Insights 之流量的進入點。

如需詳細資訊，請參閱《 AWS PrivateLink 指南》中的AWS 服務 透過 存取 AWS PrivateLink 。

Performance Insights的考量事項

在您為 Performance Insights 設定界面端點之前，請檢閱《AWS PrivateLink 指南》中的考量事項。

Performance Insights支援透過界面端點呼叫其所有 API 動作。

根據預設，允許透過界面端點完整存取 Performance Insights。若要控制透過界面端點傳輸至 Performance Insights 的流量，請將安全群組與端點網路界面建立關聯。

可用性

Performance Insights API 目前支援支援 Performance Insights AWS 區域 的 中的 VPC 端點。如需 Performance Insights 可用性的詳細資訊，請參閱 Performance Insights 的支援區域和 Aurora 資料庫引擎。

建立 Performance Insights 的界面端點

您可以使用 Amazon VPC 主控台或 AWS Command Line Interface () 為績效詳情建立介面端點AWS CLI。如需詳細資訊，請參閱《AWS PrivateLink 指南》中的「建立介面端點」。

使用下列服務名稱為 Performance Insights 建立界面端點：

如果您為該介面端點啟用私有 DNS，您可以使用其區域的預設 DNS 名稱向 Performance Insights 發出 API 要求。例如 `pi.us-east-1.amazonaws.com`。

為 Performance Insights API 建立 VPC 端點政策

端點政策為 IAM 資源，您可將其連接至介面端點。預設端點政策可允許透過界面端點完整存取 Performance Insights。若要控制 VPC 對 Performance Insights 的存取權限，請將自訂端點政策連接至界面端點。

端點政策會指定以下資訊：

- 可執行動作 (AWS 帳戶、IAM 使用者和 IAM 角色) 的主體。
- 可執行的動作。
- 可供執行動作的資源。

如需詳細資訊，請參閱《AWS PrivateLink 指南》中的「使用端點政策控制對服務的存取」。

範例：Performance Insights 動作的 VPC 端點政策

以下是自訂端點政策的範例。將此政策附加至界面端點後，此政策會針對所有資源上的所有主體，授予列出的 Performance Insights 動作的存取權限。

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "rds:CreatePerformanceAnalysisReport",
            "rds:DeletePerformanceAnalysisReport",
            "rds:GetPerformanceAnalysisReport"
         ],
         "Resource":"*"
      }
   ]
}
```

範例：拒絕來自指定 AWS 帳戶的所有存取的 VPC 端點政策

下列 VPC 端點政策拒絕 AWS 使用端點帳戶對資源123456789012的所有存取。此政策允許來自其他帳戶的所有動作。

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
   ]
}
```

Performance Insights 的 IP 定址

IP 地址可讓您 VPC 中的資源彼此互相通訊，也能和網際網路上的資源通訊。Performance Insights 同時支援 IPv4 和 IPv6 定址通訊協定。預設情況下，Performance Insights 和 Amazon VPC 都使用 IPv4 定址協議。您無法關閉此行為。當您建立 VPC 時，請務必指定 IPv4 CIDR 區塊 (私有 IPv4 地址的範圍)。

您可以選擇將 IPv6 CIDR 區塊指派給 VPC 和子網路，並將 IPv6 位址從該區塊指派給子網路中的 RDS 資源。對 IPv6 通訊協定的支援擴展受支援的 IP 地址的數量。使用 IPv6 通訊協定，您可以確保擁有足夠的可用地址，可應對網際網路的未來發展。新的和現有的 RDS 資源可於您的 VPC 中使用 IPv4 和 IPv6 地址。在應用程式的不同部分中使用的兩個通訊協定之間進行設定、保護和轉譯網路流量可能會造成操作額外負荷。您可在 IPv6 通訊協定上對 Amazon RDS 資源進行標準化，以簡化您的網路組態。如需服務端點和配額的詳細資訊，請參閱 Amazon Relational Database Service 端點和配額。

如需 AuroraIP 定址的詳細資訊，請參閱 Aurora IP 定址。

使用 Amazon DevOps Guru for Amazon RDS 分析 Aurora 效能異常

Amazon DevOps Guru 是全受管操作服務，可協助開發人員和操作員改善其應用程式的效能和可用性。DevOps Guru 分擔任務找出操作問題，讓您快速落實建議來改善應用程式。如需詳細資訊，請參閱《Amazon DevOps Guru 使用者指南》中的什麼是 Amazon DevOps Guru？。

DevOps Guru 偵測、分析所有 Amazon RDS 資料庫引擎的現有操作問題，並提出建議。DevOps Guru for RDS 擴大這種能力，將機器學習套用至 Amazon Aurora 資料庫的績效詳情指標。這些監控功能可讓 DevOps Guru for RDS 偵測和診斷效能瓶頸，並建議具體的修正動作。DevOps Guru for RDS 也可以在 Aurora 資料庫 中，於問題發生前偵測到異常情況。

您現在可以在 RDS 主控台中檢視這些建議。如需詳細資訊，請參閱Amazon Aurora 的建議。

> **Important**
>
> Amazon DevOps Guru 不適用於 Aurora Serverless 資料庫叢集。

下列影片為 DevOps Guru for RDS.的概觀。

若要深入了解此主題，請參閱 Amazon DevOps Guru for RDS under the hood (Amazon DevOps Guru for RDS 幕後)。

主題

- DevOps Guru for RDS 的好處
- DevOps Guru for RDS 的運作方式
- 設定 DevOps Guru for RDS

DevOps Guru for RDS 的好處

如果您負責 Amazon Aurora 資料庫，您可能不知道已發生事件或退化而正在影響該資料庫。得知問題時，您可能不知道為何發生或如何處理。您不需要轉向資料庫管理員 (DBA) 求助，或依賴第三方工具，您可以遵循 DevOps Guru for RDS 的建議。

您可以從 DevOps Guru for RDS 的詳細分析中獲得下列好處：

快速診斷

DevOps Guru for RDS 持續監控並分析資料庫遙測。績效詳情、增強型監控和 Amazon CloudWatch 為您的資料庫叢集收集遙測資料。DevOps Guru for RDS 使用統計和機器學習技術來挖掘此資料並偵測異常。若要進一步了解遙測資料，請參閱《Amazon Aurora 使用者指南》中的在 Amazon Aurora 以績效詳情監控資料庫負載和以增強型監控來監控 OS 指標，以及。

快速解決

每個異常都指出效能問題，並建議調查途徑或更正行動。例如，DevOps Guru for RDS 可能建議您調查特定的等待事件。或者，可能建議您調整應用程式集區設定，以限制資料庫連線的數目。採用這些建議，解決效能問題會比手動疑難排解更快。

主動式洞察

DevOps Guru for RDS 會使用資源中的指標，在潛在問題行為變得更嚴重之前進行偵測。例如，它可以偵測資料庫何時使用越來越多磁碟上暫存資料表，因為這可能會影響效能。DevOps Guru 會接著提供建議，協助您在問題變得更嚴重之前加以解決。

Amazon 工程師和機器學習的深厚知識

為了偵測效能問題並協助您解決瓶頸，DevOps Guru for RDS 仰賴機器學習 (ML) 和進階數學公式。Amazon 資料庫工程師以多年來管理數十萬個資料庫的底蘊，努力發展 DevOps Guru for RDS 的研究結果。憑藉此共同知識，DevOps Guru for RDS 可以教導您最佳實務。

DevOps Guru for RDS 的運作方式

DevOps Guru for RDS 從 Amazon RDS 績效詳情收集 Aurora 資料庫的相關資料。最重要的指標是 `DBLoad`。DevOps Guru for RDS 取用績效詳情指標、使用機器學習來分析指標，然後將洞察發佈至儀表板。

「洞察」是指 DevOps Guru 偵測到一堆相關異常。

在 DevOps Guru for RDS 中，異常是偏離所謂 Amazon Aurora 資料庫正常效能的一種模式。

主動式洞察

主動洞察可讓您在異常行為發生前了解該行為。它包含具有建議和相關指標的異常情況，可幫助您在問題擴大之前解決 Amazon Aurora 資料庫的問題。這些洞察會發佈在 DevOps Guru 儀表板。

例如，DevOps Guru 可能會偵測到您的 Aurora PostgreSQL 資料庫正在建立許多磁碟上暫存資料表。若未解決問題，之後可能導致效能問題。每個主動洞察都包含修正行為的建議，以及 使用 Amazon

DevOps Guru 主動洞察，調校 Aurora MySQL 或 使用 Amazon DevOps Guru 主動洞察，調校 Aurora PostgreSQL 中相關主題的連結。如需詳細資訊，請參閱《Amazon DevOps Guru 使用者指南》中的在 DevOps Guru 中使用洞察。

反應式洞察

反應式洞察會在發生異常行為時有效識別。如果 DevOps Guru for RDS 在您的 Amazon Aurora 資料庫執行個體中發現效能問題，則會在 DevOps Guru 儀表板中發佈反應式洞察。如需詳細資訊，請參閱《Amazon DevOps Guru 使用者指南》中的在 DevOps Guru 中使用洞察。

因果異常

因果異常是反應式洞察中最高等級的異常。資料庫負載 (DB 負載) 是 DevOps Guru for RDS 所謂的因果異常。

異常指派高、中、低的嚴重性層級來測量效能影響。如需進一步了解，請參閱《Amazon DevOps Guru 使用者指南》中的 DevOps Guru for RDS 的重要概念。

如果 DevOps Guru 在您的資料庫執行個體上偵測到目前異常，則會在 RDS 主控台的 Databases (資料庫) 頁面中提醒您。主控台還會提醒您過去 24 小時內發生的異常。若要從 RDS 主控台移至異常頁面，請選擇提醒訊息中的連結。RDS 主控台也會在頁面中提醒您注意 Amazon Aurora 資料庫叢集 。

情境異常

情境異常是資料庫負載內的研究結果，與反應式洞察相關。每個情境異常描述一個需要調查的特定 Amazon Aurora 效能問題。例如，DevOps Guru for RDS 可能建議您考慮增加 CPU 容量，或調查導致資料庫負載的等待事件。

> **Important**
>
> 建議您先在測試執行個體上測試任何變更，然後再修改生產執行個體。如此就可以了解變更的影響。

如需進一步了解，請參閱《Amazon DevOps Guru 使用者指南》中的分析 Amazon RDS 中的異常。

設定 DevOps Guru for RDS

若要讓 DevOps Guru for Amazon RDS 發佈有關 Amazon Aurora 資料庫的洞察，請完成下列任務。

主題

- 設定 DevOps Guru for RDS 的 IAM 存取政策
- 對 Aurora 資料庫執行個體開啟績效詳情
- 開啟 DevOps Guru 並指定資源涵蓋範圍

設定 DevOps Guru for RDS 的 IAM 存取政策

若要在 RDS 主控台中檢視 DevOps Guru 的提醒，您的 AWS Identity and Access Management (IAM) 使用者或角色必須具有下列任一政策：

- AWS 受管政策 `AmazonDevOpsGuruConsoleFullAccess`
- AWS 受管政策和下列`AmazonDevOpsGuruConsoleReadOnlyAccess`任一政策：
  - AWS 受管政策 `AmazonRDSFullAccess`
  - 包含 `pi:GetResourceMetrics` 和 `pi:DescribeDimensionKeys` 的客戶受管政策

如需詳細資訊，請參閱設定績效詳情的存取政策。

對 Aurora 資料庫執行個體開啟績效詳情

DevOps Guru for RDS 依賴績效詳情取得資料。如果沒有績效詳情，DevOps Guru 只發佈異常，但不包括詳細的分析和建議。

在建立 Aurora 資料庫叢集或修改資料庫執行個體時，您可以啟用績效詳情。如需詳細資訊，請參閱開啟和關閉 Aurora 的 Performance Insights。

開啟 DevOps Guru 並指定資源涵蓋範圍

您可以開啟 DevOps Guru，使其以下列其中一個方式監控 Amazon Aurora 資料庫。

主題

- 在 RDS 主控台開啟 DevOps Guru
- 在 DevOps Guru 主控台中新增 Aurora 資源
- 使用 新增 Aurora 資源 CloudFormation

在 RDS 主控台開啟 DevOps Guru

Amazon RDS 主控台中有多種途徑可供您開啟 DevOps Guru。

主題

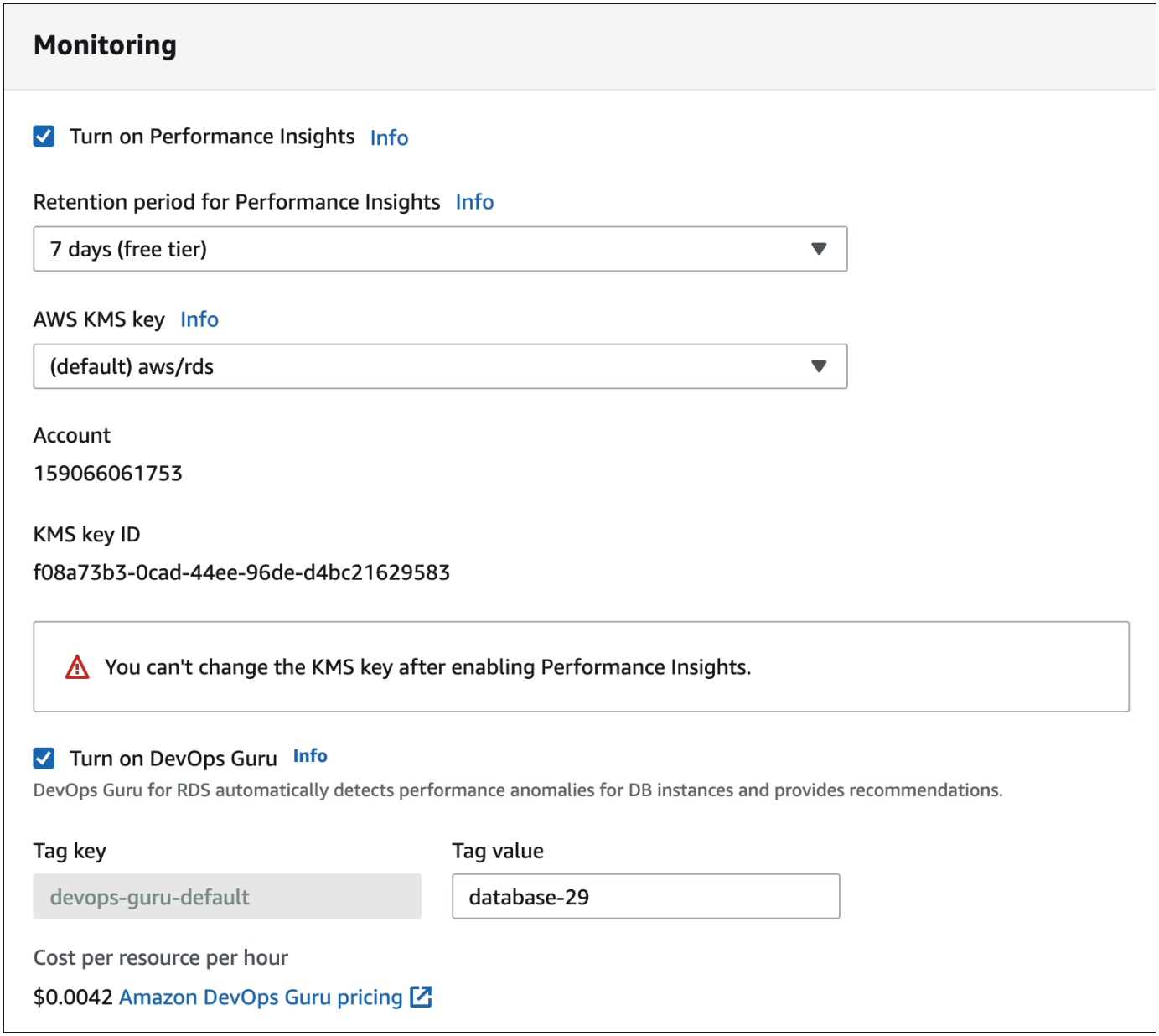
- 建立 Aurora 資料庫時開啟 DevOps Guru
- 從通知橫幅開啟 DevOps Guru
- 開啟 DevOps Guru 時回應許可錯誤

建立 Aurora 資料庫時開啟 DevOps Guru

此建立工作流程中有一個設定可為資料庫開啟 DevOps Guru 涵蓋範圍。當您選擇 Production (生產) 範本時，此設定預設為開啟。

建立 Aurora 資料庫時開啟 DevOps Guru

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 請按照 建立資料庫叢集 中的步驟進行，一直執行到 (但不包括) 您選擇監控設定的步驟。
3. 在 Monitoring (監控) 中，選擇 Turn on Performance Insights (開啟績效詳情)。若要讓 DevOps Guru for RDS 提供對效能異常的詳細分析，必須開啟 Performance Insights。
4. 選擇 Turn on DevOps Guru (開啟 DevOps Guru)。



5. 為資料庫建立一個標籤讓 DevOps Guru 可以監控資料庫。請執行下列操作：

   - 在 Tag key (標籤鍵) 的文字欄位中，輸入開頭為 **`Devops-Guru-`** 的名稱。
   - 在 Tag value (標籤值) 的文字欄位中，輸入任何一個值。例如，如果您輸入 **`rds-database-1`** 作為 Aurora 資料庫的名稱，您也可以輸入 **`rds-database-1`** 作為標籤值。

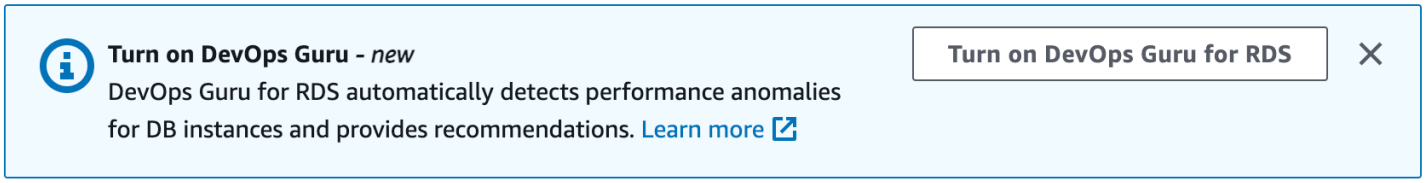
   如需標籤的詳細資訊，請參閱《Amazon DevOps Guru 使用者指南》中的使用標籤識別 DevOps Guru 應用程式中的資源。

6. 完成建立資料庫叢集 中剩餘的步驟。

從通知橫幅開啟 DevOps Guru

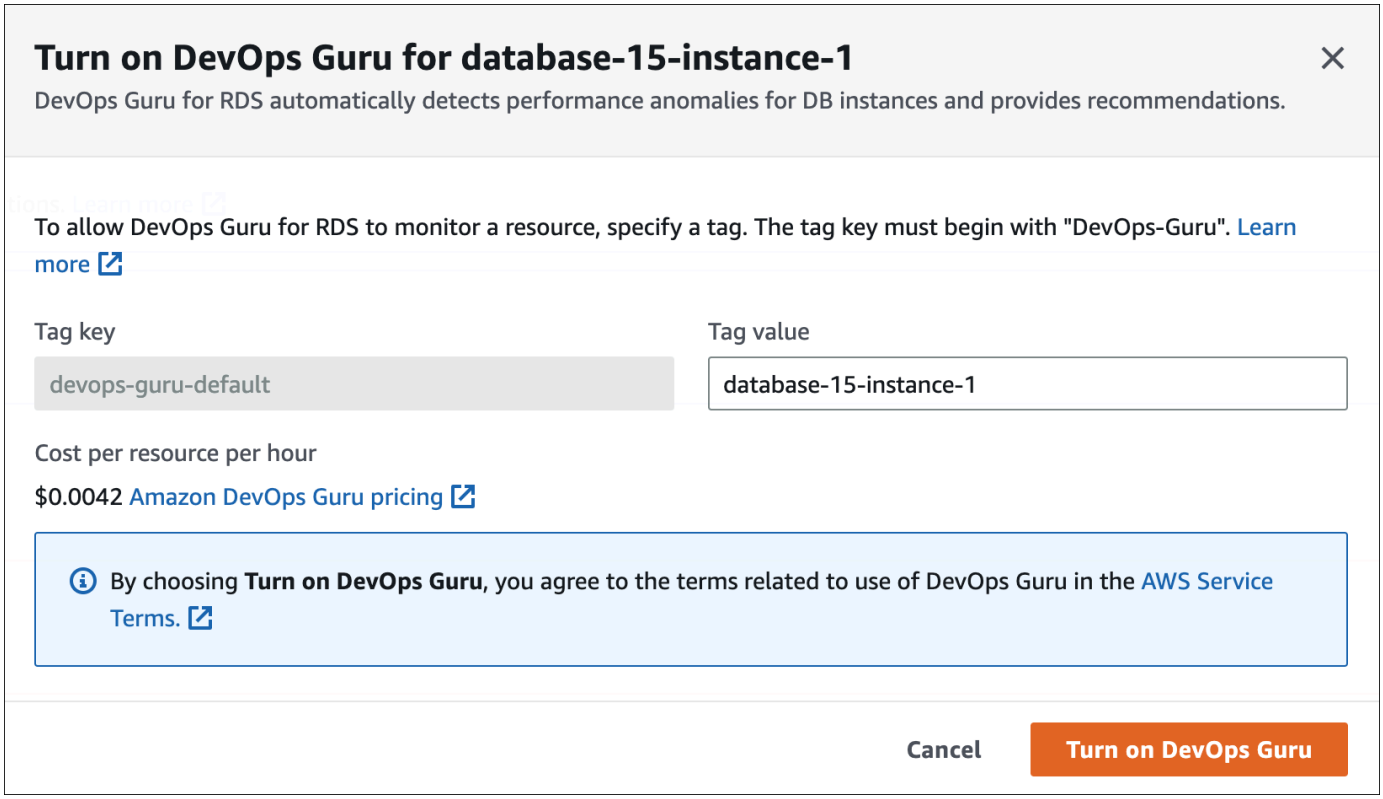
如果 DevOps Guru 未涵蓋您的資源，則 Amazon RDS 會在以下位置透過橫幅通知您：

- 資料庫叢集執行個體的 Monitoring (監控) 索引標籤
- 績效詳情儀表板



為 Aurora 資料庫開啟 DevOps Guru

1. 在橫幅中，選擇 Turn on DevOps Guru for RDS (開啟 DevOps Guru for RDS)。
2. 輸入標籤金鑰名稱與值。如需標籤的詳細資訊，請參閱《Amazon DevOps Guru 使用者指南》中的使用標籤識別 DevOps Guru 應用程式中的資源。



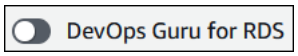
3. 選擇 Turn on DevOps Guru (開啟 DevOps Guru)。

開啟 DevOps Guru 時回應許可錯誤

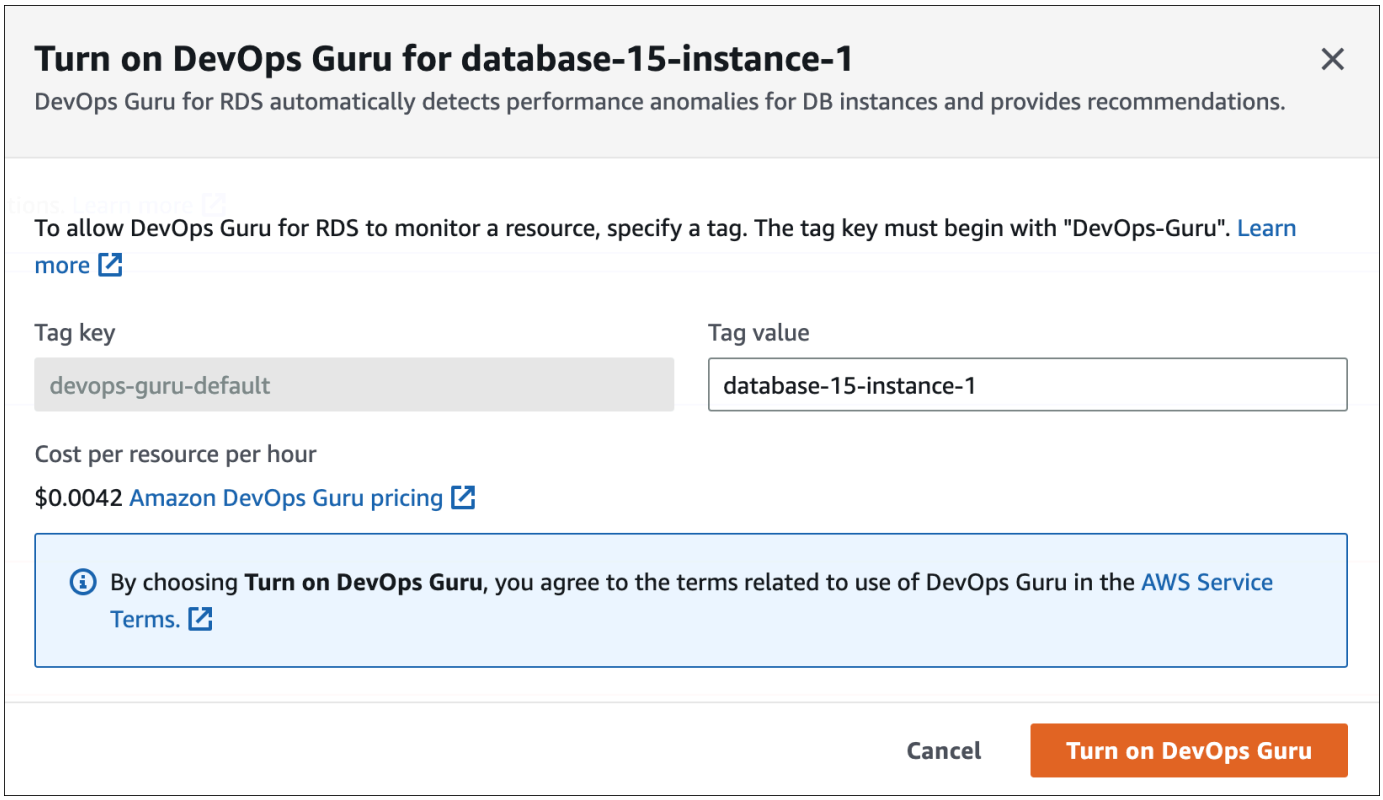
如果您在建立資料庫時從 RDS 主控台開啟 DevOps Guru，RDS 可能會顯示下列與缺少許可相關的橫幅。

回應許可錯誤

1. 將使用者管理的角色 `AmazonDevOpsGuruConsoleFullAccess` 授予您的 IAM 使用者或角色。如需詳細資訊，請參閱設定 DevOps Guru for RDS 的 IAM 存取政策。
2. 開啟 RDS 主控台。
3. 在導覽窗格中，選擇 Performance Insights (績效詳情)。
4. 在叢集中選擇您剛剛建立的資料庫執行個體。
5. 選擇切換以開啟DevOps Guru for RDS。

6. 選擇一個標籤值。如需詳細資訊，請參閱《Amazon DevOps Guru 使用者指南》中的使用標籤識別 DevOps Guru 應用程式中的資源。

7. 選擇 Turn on DevOps Guru (開啟 DevOps Guru)。

在 DevOps Guru 主控台中新增 Aurora 資源

您可以在 DevOps Guru 主控台上指定您的 DevOps Guru 資源涵蓋範圍。依照《Amazon DevOps Guru 使用者指南》中指定 DevOps Guru 資源涵蓋範圍中所述的步驟進行。在編輯分析的資源時，請選擇下列其中一個選項：

- 選擇所有帳戶資源，以分析 AWS 帳戶 和 區域中所有支援的資源，包括 Aurora 資料庫。
- 選取 CloudFormation 堆疊以分析您所選堆疊中的 Aurora 資料庫。如需詳細資訊，請參閱《Amazon DevOps Guru 使用者指南》中的使用 AWS CloudFormation 堆疊來識別 DevOps Guru 應用程式中的資源。 DevOps
- 選擇標籤分析已標記的 Aurora 資料庫。如需詳細資訊，請參閱《Amazon DevOps Guru 使用者指南》中的使用標籤識別 DevOps Guru 應用程式中的資源。

如需詳細資訊，請參閱《Amazon DevOps Guru 使用者指南》中的啟用 DevOps Guru。

使用 新增 Aurora 資源 CloudFormation

您可以使用標籤，將 Aurora 資源的涵蓋圍範新增至 CloudFormation 範本。下列程序假設您具有同時適用於 Aurora 資料庫執行個體和 DevOps Guru 堆疊的 CloudFormation 範本。

使用 CloudFormation 標籤指定 Aurora 資料庫執行個體

1. 在適用於資料庫執行個體的 CloudFormation 範本中，使用金鑰/值對定義標籤。

   下列範例會將值 `my-aurora-db-instance1` 指派給 Aurora 資料庫執行個體的 `Devops-guru-cfn-default`。

```
MyAuroraDBInstance1:
  Type: "AWS::RDS::DBInstance"
  Properties:
    DBClusterIdentifier: my-aurora-db-cluster
    DBInstanceIdentifier: my-aurora-db-instance1
    Tags:
      - Key: Devops-guru-cfn-default
        Value: devopsguru-my-aurora-db-instance1
```

2. 在 DevOps Guru 堆疊的 CloudFormation 範本中，於資源集合篩選條件中指定相同的標籤。

下列範例會設定 DevOps Guru，針對標籤值為 `my-aurora-db-instance1` 的資源提供涵蓋範圍。

```
DevOpsGuruResourceCollection:
  Type: AWS::DevOpsGuru::ResourceCollection
  Properties:
    ResourceCollectionFilter:
      Tags:
        - AppBoundaryKey: "Devops-guru-cfn-default"
          TagValues:
          - "devopsguru-my-aurora-db-instance1"
```

下列範例提供應用程式邊界 `Devops-guru-cfn-default` 內所有資源的涵蓋範圍。

```
DevOpsGuruResourceCollection:
  Type: AWS::DevOpsGuru::ResourceCollection
  Properties:
    ResourceCollectionFilter:
      Tags:
        - AppBoundaryKey: "Devops-guru-cfn-default"
          TagValues:
          - "*"
```

如需詳細資訊，請參閱《CloudFormation 使用者指南》中的 AWS::DevOpsGuru::ResourceCollection 和 AWS::RDS::DBInstance。

使用增強型監控來監控作業系統指標

使用增強型監控可即時監控資料庫執行個體的作業系統。當您想查看不同的程序或執行緒如何使用 CPU 時，增強型監控指標很有用。

主題

- 增強型監視概觀
- 設定並啟用增強型監控
- 在 RDS 主控台中檢視作業系統指標
- 使用 CloudWatch Logs 檢視作業系統指標

增強型監視概觀

Amazon RDS 針對資料庫執行個體執行所在的作業系統 (OS) 即時提供指標。您可以在主控台上檢視 RDS 資料庫執行個體的所有系統指標和處理程序資訊。您可以管理要為每個執行個體監控的指標，並根據需求自訂儀表板。如需增強型監控指標的說明，請參閱 增強型監控中的作業系統指標。

RDS 可將增強型監控的指標提供到您的 Amazon CloudWatch Logs 帳戶中。您可以從 CloudWatch Logs，在 CloudWatch 中建立指標篩選條件，並在 CloudWatch Logs 儀表板上顯示圖表。您可以在您所選的監控系統中，使用來自 CloudWatch Logs 的增強型監控 JSON 輸出。如需詳細資訊，請參閱「Amazon RDS 常見問答集」中的增強型監控。

主題

- CloudWatch 與增強型監控指標之間的差異
- 保留增強型監控指標
- 增強型監控的成本

CloudWatch 與增強型監控指標之間的差異

hypervisor 會建立並執行虛擬機器 (VM)。執行個體可使用 hypervisor，透過虛擬化共用記憶體和 CPU 來支援多個訪客虛擬機器。CloudWatch 會從資料庫執行個體的 hypervisor 收集有關 CPU 使用率指標。相反地，增強型監控會從資料庫執行個體上的代理程式中收集其指標。

您可能會發現 CloudWatch 與增強型監控測量之間的差異，因為 Hypervisor 層會執行少量工作。如果您的資料庫執行個體使用較小的執行個體類別，差異可能會更大。在此案例中，單一實體執行個體上的 Hypervisor 層可能會管理更多虛擬機器 (VM)。

如需增強型監控指標的說明，請參閱 增強型監控中的作業系統指標。如需有關 Amazon CloudWatch 的詳細資訊，請參閱 Amazon CloudWatch 使用者指南。

保留增強型監控指標

在預設情況下，增強型監控指標會在 CloudWatch Logs 存放 30 天。此保留期與一般的 CloudWatch 指標不同。

若要修改指標在 CloudWatch Logs 中存放的時間長短，請在 CloudWatch 主控台變更 `RDSOSMetrics` 日誌群組的保留期間。如需詳細資訊，請參閱《Amazon CloudWatch Logs User Guide》中的變更 CloudWatch 日誌中的日誌資料保留期間。

增強型監控的成本

增強型監控指標存放在 CloudWatch Logs 中，而不是在 CloudWatch 指標中。增強型監控的成本取決於下列因素：

- 只有在「增強型監控」用量超過 Amazon CloudWatch Logs 提供的資料傳輸和儲存量時，我們才會向您收取費用。費用是根據 CloudWatch Logs 資料傳輸和儲存費率計算。
- 針對 RDS 執行個體傳輸的資訊量與針對「增強型監控」功能定義的精密度成正比。較短的監控時間間隔會導致較頻繁的作業系統指標報告，並增加您的監控成本。若要管理成本，請針對帳戶中的不同執行個體設定不同的精密度。
- 增強型監控的使用成本將套用至已啟用增強型監控的每個資料庫執行個體。監控大量資料庫執行個體的成本比監控少量資料庫執行個體昂貴得多。
- 支援運算更為密集之工作負載的資料庫執行個體，有更多的作業系統處理程序活動可以報告，其增強型監控的成本也比較高。

如需定價的詳細資訊，請參閱 Amazon CloudWatch 定價。

設定並啟用增強型監控

若要使用增強型監控，您必須建立 IAM 角色，然後啟用增強型監控。

主題

- 為增強型監控建立 IAM 角色
- 開啟和關閉增強型監控
- 防範混淆代理人問題

為增強型監控建立 IAM 角色

增強型監控需要許可才能代表您將作業系統指標資訊傳送至 CloudWatch Logs。您可以使用 AWS Identity and Access Management (IAM) 角色授予增強型監控許可。您可以在啟用增強型監控時建立此角色，也可以事先建立。

主題

- 啟用增強型監控時建立 IAM 角色
- 在啟用增強型監控之前建立 IAM 角色

啟用增強型監控時建立 IAM 角色

當您在 RDS 主控台中啟用增強監控時，Amazon RDS 可以為您建立所需的 IAM 角色。角色已命名 `rds-monitoring-role`。RDS 會將此角色用於指定的資料庫執行個體、僅供讀取複本或多可用區域資料庫叢集。

啟用增強型監控時建立 IAM 角色

1. 請遵循 開啟和關閉增強型監控 中的步驟。
2. 在您選擇角色的步驟中，將 「Monitoring Role (監控角色)」設定為「Default (預設) 」。

在啟用增強型監控之前建立 IAM 角色

您可以在啟用增強型監控之前建立必要的角色。啟用增強型監控時，請指定新角色的名稱。如果您使用 AWS CLI 或 RDS API 啟用增強型監控，您將必須建立此必要的角色。

必須將 `PassRole` 許可授予會啟用增強型監控的使用者。如需詳細資訊，請參閱《IAM 使用者指南》中的授予使用者將角色傳遞至 AWS 服務之許可的範例 2。

為 Amazon RDS 增強型監控建立 IAM 角色

1. 前往 https://console.aws.amazon.com 開啟 IAM 主控台
2. 在導覽窗格中，選擇 Roles (角色)。
3. 選擇 Create Role (建立角色)。
4. 選擇 AWS service (AWS 服務) 索引標籤，然後從服務清單中選擇 RDS。
5. 選擇 RDS - Enhanced Monitoring (RDS - 增強型監控)，然後選擇 Next (下一步)。
6. 請確定 Permissions policies (許可政策) 顯示 AmazonRDSEnhancedMonitoringRole，然後選擇 Next (下一步)。

7. 針對 Role name (角色名稱)，輸入您的角色名稱。例如，輸入 **emaccess**。

   您角色的信任實體是 monitoring.rds.amazonaws.com AWS 服務。

8. 選擇建立角色。

開啟和關閉增強型監控

您可以使用 AWS 管理主控台 AWS CLI或 RDS API 來管理增強型監控。您可以針對每個資料庫叢集上的指標收集設定不同的精密度。您也可以從主控台切換現有資料庫叢集的增強型監控。

主控台

您可以在建立資料庫叢集或僅供讀取複本時，或在您修改資料庫叢集時，開啟增強型監控。如果您修改資料庫執行個體或叢集以開啟增強型監控，您的資料庫執行個體無需重新啟動，改變即可生效。

您在 Databases (資料庫) 頁面上執行以下任一動作時，可在 RDS 主控台中開啟增強型監控：

- 建立資料庫叢集：選擇 Create database (建立資料庫)。
- 建立僅供讀取複本 – 選擇 Actions (動作)，然後選 Create read replica (建立僅供讀取複本)。
- 修改資料庫執行個體或資料庫叢集 – 選擇修改。

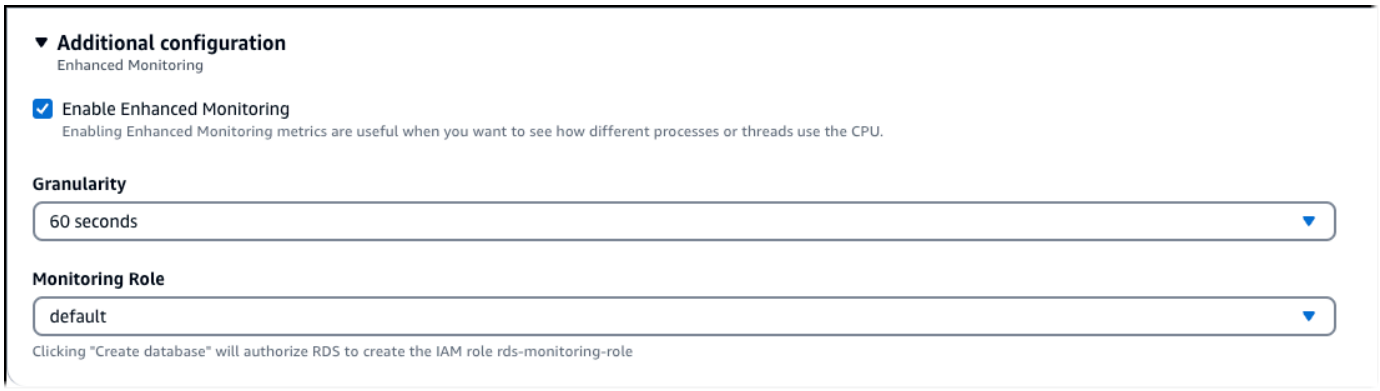
> **Note**
>
> 當您為資料庫叢集啟用增強型監控時，您無法管理叢集內個別資料庫執行個體的增強型監控。

如果您正在為資料庫叢集中的個別資料庫執行個體管理增強型監控，且精細程度針對不同執行個體設定為不同值，則資料庫叢集在增強型監控方面具有異質性。在這種情況下，您無法修改資料庫叢集以在叢集層級管理增強型監控。

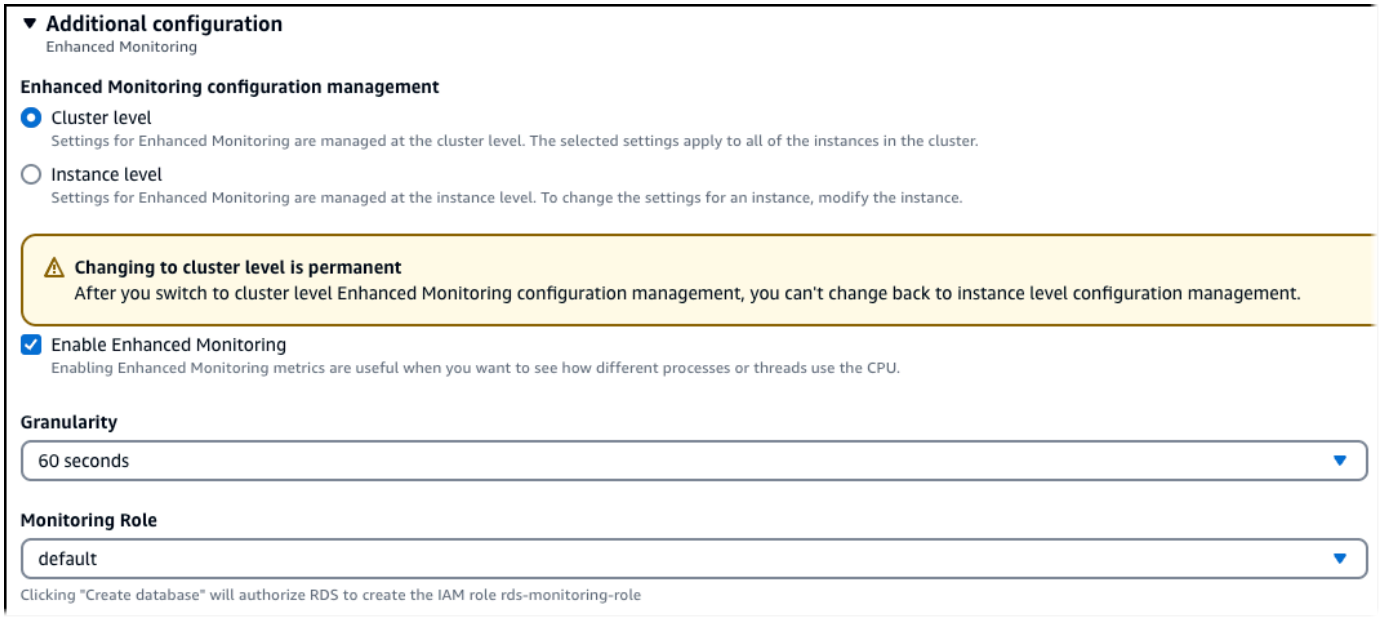
在 RDS 主控台中開啟或關閉增強型監控

1. 捲動至 Additional configuration (其他組態)。
2. 在監控中，為資料庫叢集或僅供讀取複本選擇啟用增強型監控。在叢集層級啟用增強型監控可讓您在叢集層級管理增強型監控設定和選項。叢集層級設定會套用至叢集的所有資料庫執行個體。取消選取選項以在叢集層級停用增強型監控。您稍後可以修改叢集中個別資料庫執行個體的增強型監控設定。

在建立資料庫頁面中，您可以選取在叢集層級開啟增強型監控。



如果您在建立叢集時未啟用增強型監控，您可以在修改資料庫叢集頁面中修改叢集。



> **Note**
>
> 您無法為已在叢集層級管理增強型監控的資料庫叢集中的個別資料庫執行個體管理增強型監控。

3. 如果叢集在增強型監控方面具有異質性，則您無法選擇在叢集層級管理增強型監控。若要在叢集層級管理增強型監控，請變更每個執行個體的增強型監控設定，使其相符。您現在可以在修改叢集時，選擇在叢集層級管理增強型監控。
4. 將 Monitoring Role (監控角色) 屬性設定為您所建立的 IAM 角色，以允許 Amazon RDS 代表您與 Amazon CloudWatch Logs 進行通訊。或者，選擇 Default (預設)，RDS 即會自動建立名為 `rds-monitoring-role` 的角色。

5. 系統會收集資料庫執行個體、資料庫叢集或僅供讀取複本的指標，請將精細程度屬性設定為該資料點的時間間隔 (以秒為單位)。Granularity (精細程度) 屬性可設定為以下其中一個值：1、5、10、15、30 或 60。

   RDS 主控台重新整理的最快時間為每 5 秒一次。如果您在 RDS 主控台中，將精細程度設定為 1 秒，您還是只能每 5 秒鐘才能看到更新後的指標。您可以使用 CloudWatch Logs 來擷取 1 秒指標更新。

AWS CLI

若要使用 開啟增強型監控 AWS CLI，請在下列命令中，將 `--monitoring-interval`選項設定為 以外的值，`0`並將 `--monitoring-role-arn`選項設定為您在 中建立的角色為增強型監控建立 IAM 角色。

- create-db-cluster
- modify-db-cluster
- create-db-instance
- create-db-instance-read-replica
- modify-db-instance

`--monitoring-interval` 為資料庫執行個體收集增強型監控指標點之間的間隔 (秒)。選項的有效值為 `0`、`1`、`5`、`10`、`15`、`30` 和 `60`。

若要使用 關閉增強型監控 AWS CLI，請在這些命令`0`中將 `--monitoring-interval`選項設定為 。

Example

下方範例會為資料庫執行個體開啟增強型監控：

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

在 Windows 中：

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Example

下方範例會為資料庫叢集開啟增強型監控：

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbinstance \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbinstance ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Example

下方範例會為多可用區域資料庫叢集開啟增強型監控：

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

RDS API

若要使用 RDS API 開啟增強型監控，請將 `MonitoringInterval` 參數設定為 `0` 以外的值，然後將 `MonitoringRoleArn` 參數設定為您在 為增強型監控建立 IAM 角色 中建立的角色。在下列動作中設定這些參數：

- CreateDBCluster
- ModifyDBCluster
- CreateDBInstance
- CreateDBInstanceReadReplica
- ModifyDBInstance

`MonitoringInterval` 參數會指定資料庫執行個體收集增強型監控指標點之間的間隔 (秒)。有效值為：`0`、`1`、`5`、`10`、`15`、`30` 和 `60`。

若要使用 RDS API 關閉增強型監控，請將 `MonitoringInterval` 設定為 `0`。

防範混淆代理人問題

混淆代理人問題屬於安全性議題，其中沒有執行動作許可的實體可以強制具有更多許可的實體執行該動作。在 中 AWS，跨服務模擬可能會導致混淆代理人問題。在某個服務 (呼叫服務) 呼叫另一個服務 (被呼叫服務) 時，可能會發生跨服務模擬。可以操縱呼叫服務來使用其許可，以其不應有存取許可的方式對其他客戶的資源採取動作。為了預防這種情況， AWS 提供的工具可協助您保護所有服務的資料，而這些服務主體已獲得您帳戶中資源的存取權。如需詳細資訊，請參閱混淆代理人問題。

若要限制 Amazon RDS 可將資源提供給另一項服務的許可，建議針對增強型監控角色，在信任政策中使用 `aws:SourceArn` 和 `aws:SourceAccount` 全域條件內容索引鍵。如果您同時使用兩個全域條件內容索引鍵，兩者必須使用相同的帳戶 ID。

防範混淆代理人問題的最有效方法是使用 `aws:SourceArn` 全域條件內容索引鍵，以及資源的完整 ARN。為 Amazon RDS 將 `aws:SourceArn` 設為 `arn:aws:rds:`*`Region`*`:`*`my-account-id`*`:db:`*`dbname`*。

下列範例展示如何在信任政策中使用 `aws:SourceArn` 和 `aws:SourceAccount` 全域條件內容索引鍵，來預防混淆代理人問題。

JSON

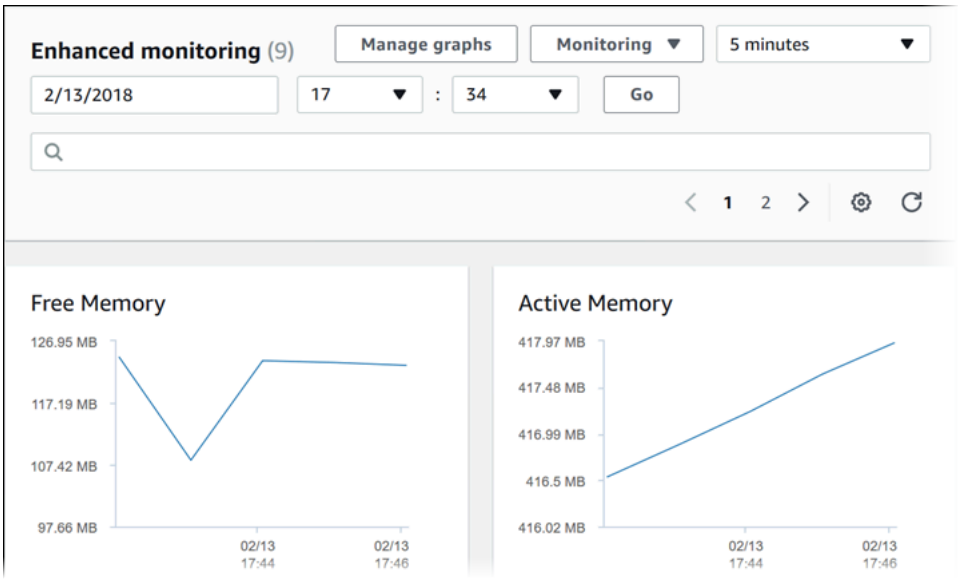
```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "monitoring.rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringLike": {
          "aws:SourceArn": "arn:aws:rds:Region:my-123456789012:db:dbname"
        },
        "StringEquals": {
          "aws:SourceAccount": "my-123456789012"
        }
      }
    }
  ]
}
```

在 RDS 主控台中檢視作業系統指標

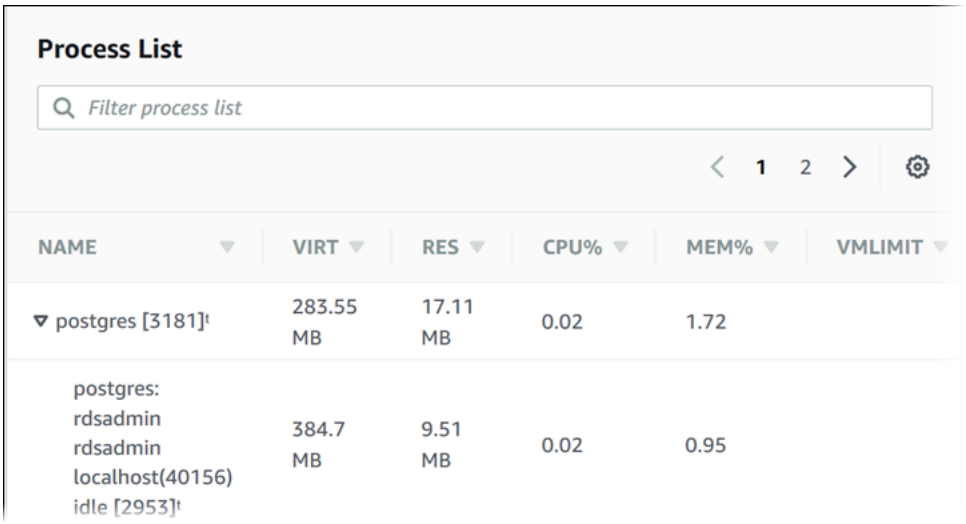
您可在 RDS 主控台的 Monitoring (監控) 選擇 Enhanced monitoring (增強型監控)，以檢視增強型監控回報的作業系統指標。

下列範例會顯示「增強型監控」頁面。如需增強型監控指標的說明，請參閱 [增強型監控中的作業系統指標](#)。



如果您要檢視資料庫執行個體上執行的處理程序，則請在 Monitoring (監控) 中選擇 OS process list (作業系統程序清單)，即可查看詳細資訊。

Process List (程序清單) 檢視畫面顯示如下。



顯示於 Process list (程序清單) 檢視畫面中的增強型監控指標整理如下：

- RDS child processes (RDS 子程序) – 顯示支援資料庫執行個體的 RDS 程序摘要，例如 Amazon Aurora 資料庫叢集的 `aurora`，以及 。程序執行緒以巢狀顯示在父程序之下。程序執行緒僅顯示 CPU 使用率，因為其他指標與該程序的所有執行緒相同。主控台最多顯示 100 個程序與執行緒。其結果為消耗最多 CPU 與記憶體的程序與執行緒的組合。如果有 50 個以上的程序及 50 個以上的執行緒，主控台將顯示各類別中消耗量最高的 50 個程序與執行緒。此顯示有助於您識別哪些程序對於效能的影響最大。
- RDS 程序 – 顯示 RDS 管理代理程式所使用的資源摘要、診斷監控程序，以及支援 RDS 資料庫執行個體所需的其他 AWS 程序。
- OS processes (作業系統程序) – 顯示核心與系統程序的摘要，這些程序對效能的影響通常很小。

各程序所列出的項目為：

- VIRT (VIRT) – 顯示程序的虛擬記憶體大小。
- RES (RES) – 顯示程序實際使用的實體記憶體。
- CPU% – 會顯示程序正在使用的 CPU 頻寬總量百分比。
- MEM% – 會顯示程序正在使用的記憶體總量百分比。

RDS 主控台顯示的監控資料擷取自 Amazon CloudWatch Logs。您也可以從 CloudWatch Logs 擷取資料庫執行個體的指標做為日誌串流。如需更多詳細資訊，請參閱 使用 CloudWatch Logs 檢視作業系統指標。

在以下期間不會傳回增強型監控指標：

- 資料庫執行個體的容錯移轉。
- 變更資料庫執行個體的執行個體類別 (擴充運算)。

在資料庫執行個體重新開機過程中會傳回增強型監控指標，因為只有資料庫引擎會重新開機。作業系統指標仍會繼續回報。

使用 CloudWatch Logs 檢視作業系統指標

在您為資料庫叢集啟用增強型監控之後，即可使用 CloudWatch Logs 與代表受監控之單一資料庫執行個體或資料庫叢集的日誌串流來檢視資料庫執行個體。日誌串流識別符為該資料庫執行個體或資料庫叢集的資源識別符 (`DbiResourceId`)。

檢視增強型監控日誌資料

1. 透過 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 如有必要，請選擇您資料庫叢集所在的 AWS 區域。如需詳細資訊，請參閱《Amazon Web Services 一般參考》中的區域和端點。
3. 在導覽窗格中，選擇 Logs (日誌)。
4. 在日誌群組清單中，選擇 RDSOSMetrics。
5. 在日誌串流清單中，選擇您要檢視的日誌串流。

Amazon Aurora 的指標參考

在此參考中，您可以找到 Amazon CloudWatch、Performance Insights 和增強型監控的 Amazon Aurora 指標描述。

主題

- Amazon Aurora 的 Amazon CloudWatch 指標
- Aurora 的 Amazon CloudWatch 維度
- Amazon RDS 主控台中 Aurora 指標的可用性
- Amazon RDS Performance Insights 的 Amazon CloudWatch 指標
- Performance Insights 計數器指標
- 績效詳情的 SQL 統計數字
- 增強型監控中的作業系統指標

Amazon Aurora 的 Amazon CloudWatch 指標

AWS/RDS 命名空間有以下指標適用於 Amazon Aurora 上執行的資料庫實體。某些指標適用於 Aurora MySQL、Aurora PostgreSQL，或兩者。此外，某些指標專用於資料庫叢集、主要資料庫執行個體、複本資料庫執行個體或所有資料庫執行個體。

如需 Aurora 全球資料庫指標，請參閱 Aurora MySQL 中寫入轉送的 Amazon CloudWatch 指標 和 Aurora PostgreSQL 中寫入轉送的 Amazon CloudWatch 指標。如需 Aurora 平行查詢指標，請參閱監控 Aurora MySQL 的平行查詢。

主題

- Amazon Aurora 的叢集層級指標
- Amazon Aurora 的執行個體層級指標
- Amazon Aurora 的 Amazon CloudWatch 用量指標

Amazon Aurora 的叢集層級指標

下表說明 Aurora 叢集專用的指標。

指標	描述	適用對象	單位
AuroraGlobalDBDataTransferBytes	在 Aurora 全域資料庫中，從來源 AWS 區域傳輸到次要 AWS 區域的重做日誌資料量。 Note 此指標僅適用於次要 AWS 區域。	Aurora MySQL 和 Aurora PostgreSQL	位元組
AuroraGlobalDBProgressLag	在 Aurora 全球資料庫中，他測量次要叢集的儲存磁碟區在使用者交易和系統交易中落後主要叢集的儲存磁碟區的程度。 Note 此指標僅適用於次要 AWS 區域。	Aurora MySQL 和 Aurora PostgreSQL	毫秒
AuroraGlobalDBReplicatedWriteIO	在 Aurora 全域資料庫中，從主要 AWS 區域複寫到次要區域中叢集磁碟區的寫入 I/O 操作數目 AWS 。全域資料庫中次要 AWS 區域的帳單計算VolumeWriteIOPs ，用於考慮叢集內執行的寫入。全球資料庫中主要 AWS 區域的帳單計算VolumeWriteIOPs ，用於考慮該叢集內的寫入活動，以及AuroraGlobalDBReplicatedWri	Aurora MySQL 和 Aurora PostgreSQL	計數

指標	描述	適用對象	單位
	teIO 考慮全球資料庫中的跨區域複寫。 **Note** 此指標僅適用於次要 AWS 區域。		
AuroraGlobalDBReplicationLag	對於 Aurora 全域資料庫，主要叢集複寫伺服器與次要叢集複寫伺服器之間經過的平均複寫更新時間。 **Note** 此指標僅適用於次要 AWS 區域。	Aurora MySQL 和 Aurora PostgreSQL	毫秒
AuroraGlobalDBRPOLag	在 Aurora 全球資料庫中，復原點目標 (RPO) 延遲時間。此指標會測量次要叢集在使用者交易的主要叢集之後的距離。 **Note** 此指標僅適用於次要 AWS 區域。	Aurora MySQL 和 Aurora PostgreSQL	毫秒

指標	描述	適用對象	單位
AuroraVolumeBytesLeftTotal	叢集磁碟區的剩餘可用空間。當叢集磁碟區增大時，此值會減小。若達到零，叢集會報告空間不足錯誤。 若您想要偵測 Aurora MySQL 叢集是否接近 128 tebibytes (TiB) 大小限制，此值比監控 VolumeBytesUsed 更為簡單且可靠。AuroraVolumeBytesLeftTotal 會考量用於內務管理的帳戶儲存空間，並且不會影響您儲存空間計費的其他配置。	Aurora MySQL	位元組
BacktrackChangeRecordsCreationRate	在 5 分鐘內為您的資料庫叢集建立的恢復變更記錄數量。	Aurora MySQL	每 5 分鐘計數
BacktrackChangeRecordsStored	您的資料庫叢集使用的恢復變更記錄數量。	Aurora MySQL	計數
BackupRetentionPeriodStorageUsed	在 Aurora 資料庫叢集的備份保留時段內，用來支援時間點還原功能的備份儲存總量。此數量包含在 TotalBackupStorageBilled 指標報告的總計中。該值會針對每個 Aurora 叢集單獨計算。如需說明，請參閱「了解 Amazon Aurora 備份儲存體用量」。	Aurora MySQL 和 Aurora PostgreSQL	位元組

指標	描述	適用對象	單位
ServerlessDatabaseCapacity	Aurora Serverless 資料庫叢集的當前容量。	Aurora MySQL 和 Aurora PostgreSQL	計數
SnapshotStorageUsed	Aurora 資料庫叢集在其備份保留時段外，由所有 Aurora 快照取用的備份儲存總量。此數量包含在 TotalBackupStorageBilled 指標報告的總計中。該值會針對每個 Aurora 叢集單獨計算。如需說明，請參閱「了解 Amazon Aurora 備份儲存體用量」。	Aurora MySQL 和 Aurora PostgreSQL	位元組
TotalBackupStorageBilled	您針對某個 Aurora 資料庫叢集而付費的備份儲存總量 (位元組)。該指標包含以 BackupRetentionPeriodStorageUsed 和 SnapshotStorageUsed 指標計量的備份儲存。此指標是針對每個 Aurora 叢集單獨計算。如需說明，請參閱「了解 Amazon Aurora 備份儲存體用量」。	Aurora MySQL 和 Aurora PostgreSQL	位元組

指標	描述	適用對象	單位
VolumeBytesUsed	您的 Aurora 資料庫叢集使用的儲存量。 此值會影響 Aurora 資料庫叢集的成本 (如需定價資訊，請參閱 Amazon RDS 定價頁面)。 此值不會反映一些不影響儲存空間計費的內部儲存空間配置。對於 Aurora MySQL，您可以測試 AuroraVolumeBytesLeftTotal 是否正在接近零，而不是對照 128 TiB 儲存空間限制來比較 VolumeBytesUsed ，藉以更精確地採取行動來防止空間不足問題。 對於複製的叢集，此指標的值取決於複製上新增或變更的資料量。刪除原始叢集時，或新增或刪除新複製時，指標也可能會增加或減少。如需詳細資訊，請參閱刪除來源叢集磁碟區 請注意，選擇較小的 --period 值是沒有意義的，因為 Amazon RDS 是以間隔時間收集此指標，而非持續收集。	Aurora MySQL 和 Aurora PostgreSQL	位元組

指標	描述	適用對象	單位
VolumeReadIOPs	叢集磁碟區每隔 5 分鐘的計費讀取輸入/輸出操作次數。 計費的讀取操作以叢集磁碟區層級計算，彙整來自 Aurora 資料庫叢集中的所有執行個體，每隔 5 分鐘回報一次。此值以每隔 5 分鐘取得的讀取操作指標值進行計算。您可以藉由取得計費讀取操作指標值並除以 300 秒，計算出每秒的計費讀取操作量。例如，如果計費讀取操作傳回 13,686，則每秒的計費讀取操作為 45 (13,686 / 300 = 45.62)。 您的查詢提出資料庫頁面請求，該頁面不存在於緩衝區快取，而且必須從儲存空間載入，因而產生計費讀取操作。您可能會在計費讀取操作中發現峰值，那是因為從儲存空間讀取查詢結果，然後載入至緩衝區快取。 請注意，選擇較小的 `--period` 值是沒有意義的，因為 Amazon RDS 是以間隔時間收集此指標，而非持續收集。 **Tip** 如果您的 Aurora MySQL 叢集使用平	Aurora MySQL 和 Aurora PostgreSQL	每 5 分鐘計數

指標	描述	適用對象	單位
	行查詢，您可能會看到 VolumeReadIOPS 值增加。平行查詢不會使用緩衝集區。因此，雖然查詢速度很快，但這種最佳化處理可能會導致讀取操作和相關費用增加。		
VolumeWriteIOPs	叢集磁碟區的寫入磁碟輸入/輸出操作次數，每隔 5 分鐘回報一次。如需計費寫入操作計算方式的詳細說明，請參閱 VolumeReadIOPs 。 請注意，選擇較小的 --period 值是沒有意義的，因為 Amazon RDS 是以間隔時間收集此指標，而非持續收集。	Aurora MySQL 和 Aurora PostgreSQL	每 5 分鐘計數

Amazon Aurora 的執行個體層級指標

除非另有說明，否則下列執行個體特定的 Amazon CloudWatch 指標適用於所有 Aurora MySQL 和 Aurora PostgreSQL 執行個體。

指標	描述	適用對象	個單位
AbortedClients	尚未正確關閉的用戶端連線數目。	Aurora MySQL	計數
ActiveTransactions	目前在 Aurora 資料庫執行個體上執行的平均每秒交易數量。	Aurora MySQL	每秒計數

指標	描述	適用對象	個單位
	根據預設，Aurora 不會啟用此指標。如要開始測量此值，請針對特定資料庫執行個體設定資料庫參數群組的 `innodb_monitor_enable='all'` 。		
`ACUUtilization`	`ServerlessDatabaseCapacity` 指標的值除以資料庫叢集的最大 ACU 值。 此指標僅適用於 Aurora Serverless v2。	Aurora MySQL 和 Aurora PostgreSQL	百分比

指標	描述	適用對象	個單位
AuroraBinlogReplicaLag	執行於 Aurora MySQL 的二進位日誌複本資料庫叢集落後於二進位日誌複寫來源的時間量。延遲意味著來源生成記錄的速度比複本能夠套用記錄的速度快。 此指標根據引擎版本報告不同的值： Aurora MySQL 第 2 版 MySQL SHOW SLAVE STATUS 的 Seconds_Behind_Master 欄位 Aurora MySQL 第 3 版 SHOW REPLICA STATUS 您可以使用此指標來監控作為二進制日誌複本的叢集錯誤和複本延遲。指標值表示以下內容： 較高值 複本落後於複寫來源。 **0** 或接近 **0** 的值 複本程序處於作用中且最新。 **-1** Aurora 無法判斷延遲，可能會在複本設定期間或複本處於錯誤狀態時發生。	主要適用於 Aurora MySQL	秒鐘

指標	描述	適用對象	個單位
	因為二進位日誌複寫僅在叢集的寫入器執行個體上發生，建議您使用與 WRITER 角色相關聯的此指標的版本。 如需管理複寫的詳細資訊，請參閱 跨 AWS 區域 複寫 Amazon Aurora MySQL 資料庫叢集。如需疑難排解的詳細資訊，請參閱 Amazon Aurora MySQL 複寫問題。		
AuroraDMLRejectedMasterFull	由於寫入器資料庫執行個體上的工作階段已滿，因此拒絕的轉送查詢數量。	主要適用於 Aurora MySQL 第 2 版。	計數
AuroraDMLRejectedWriterFull	由於寫入器資料庫執行個體上的工作階段已滿，因此拒絕的轉送查詢數量。	主要適用於 Aurora MySQL 第 3 版。	計數
AuroraEstimatedSharedMemoryBytes	上次設定的輪詢間隔期間，主動使用的共用緩衝區或緩衝集區記憶體的預估數量。	Aurora PostgreSQL	位元組
AuroraMemoryHealthState	指出記憶體運作狀態。0 的值等於 NORMAL。10 的值等於 RESERVED，這表示伺服器即將接近記憶體用量的臨界層級。 如需詳細資訊，請參閱故障診斷 Aurora MySQL 資料庫記憶體不足的問題。	Aurora MySQL 3.06.1 版及更新版本	量測計

指標	描述	適用對象	個單位
AuroraMemoryNumDeclinedSqlTotal	避免記憶體不足 (OOM) 期間，查詢的增量數量遭拒。 如需詳細資訊，請參閱故障診斷 Aurora MySQL 資料庫記憶體不足的問題。	Aurora MySQL 3.06.1 版及更新版本	計數
AuroraMemoryNumKillConnTotal	避免 OOM 期間關閉的連線增量數量。 如需詳細資訊，請參閱故障診斷 Aurora MySQL 資料庫記憶體不足的問題。	Aurora MySQL 3.06.1 版及更新版本	計數
AuroraMemoryNumKillQueryTotal	避免 OOM 期間而結束的查詢增量數量。 如需詳細資訊，請參閱故障診斷 Aurora MySQL 資料庫記憶體不足的問題。	Aurora MySQL 3.06.1 版及更新版本	計數
AuroraMillisecondsSpentInOomRecovery	從記憶體運作狀態下降到低於正常狀態起的時間量。 如需詳細資訊，請參閱故障診斷 Aurora MySQL 資料庫記憶體不足的問題。	Aurora MySQL 版本 3.08.0 及更高版本	毫秒
AuroraNumOomRecoverySuccessful	記憶體運作狀態還原至正常狀態的次數。 如需詳細資訊，請參閱故障診斷 Aurora MySQL 資料庫記憶體不足的問題。	Aurora MySQL 版本 3.08.0 及更高版本	計數

指標	描述	適用對象	個單位
AuroraNumOomRecoveryTriggered	記憶體運作狀態低於正常狀態的次數。 如需詳細資訊，請參閱故障診斷 Aurora MySQL 資料庫記憶體不足的問題。	Aurora MySQL 版本 3.08.0 及更高版本	計數
AuroraOptimizedReadsCacheHitRatio	由最佳化讀取快取提供服務的要求的百分比。 值會使用下列公式來計算： orcache_blks_hit/(orcache_blks_hit + storage_blks_read) 當 AuroraOptimizedReadsCacheHitRatio 為 100% 時，表示已從最佳化讀取快取讀取所有頁面。如果 AuroraOptimizedReadsCacheHitRatio 是 0，表示未從最佳化讀取快取讀取任何頁面。	主要適用於 Aurora PostgreSQL	百分比
AuroraReplicaLag	在單一區域 Aurora 叢集或全域資料庫主要叢集中，從相同區域中的寫入器執行個體複寫更新至複本執行個體所經過的時間。在全球資料庫次要叢集中，相同區域中複本執行個體和次要叢集複寫伺服器的複寫更新經過的時間量。	適用於 Aurora MySQL 和 Aurora PostgreSQL 的複本	毫秒

指標	描述	適用對象	個單位
AuroraReplicaLagMaximum	主要執行個體與資料庫叢集中任何 Aurora 資料庫執行個體之間的最大延遲量。 刪除或重新命名僅供讀取複本時，當舊資源經過回收程序時，複寫延遲可能會暫時遽增。若要在該期間內準確呈現複寫延遲，建議您監控每個僅供讀取複本執行個體上的 AuroraReplicaLag 指標。	主要適用於 Aurora MySQL 和 Aurora PostgreSQL	毫秒
AuroraReplicaLagMinimum	主要執行個體與資料庫叢集中任何 Aurora 資料庫執行個體之間的最小延遲量。	主要適用於 Aurora MySQL 和 Aurora PostgreSQL	毫秒
AuroraSlowConnectionHandleCount	等待開始交握的時間超過兩秒的連線數目。 此指標僅適用於 Aurora MySQL 第 3 版。	Aurora MySQL	計數
AuroraSlowHandshakeCount	等待完成交握的時間超過 50 毫秒的連線數目。 此指標僅適用於 Aurora MySQL 第 3 版。	Aurora MySQL	計數
BacktrackWindowActual	目標 Backtrack window 與實際 Backtrack window 之間的差異。	主要適用於 Aurora MySQL	分鐘

指標	描述	適用對象	個單位
BacktrackWindowAlert	在指定時間內，實際 Backtrack window 小於目標 Backtrack window 的次數。	主要適用於 Aurora MySQL	計數
BlockedTransactions	被封鎖的資料庫中的平均每秒交易數量。	Aurora MySQL	每秒計數
BufferCacheHitRatio	由緩衝區快取提供服務的請求的百分比。	Aurora MySQL 和 Aurora PostgreSQL	百分比
CommitLatency	引擎和儲存體完成遞交操作所花的平均時間長度。	Aurora MySQL 和 Aurora PostgreSQL	毫秒
CommitThroughput	遞交操作的每秒平均數量。	Aurora MySQL 和 Aurora PostgreSQL	每秒計數
ConnectionAttempts	嘗試連線至執行個體的次數 (無論是否成功)。	Aurora MySQL	計數

指標	描述	適用對象	個單位
CPUCreditBalance	執行個體累積的 CPU 額度，每隔 5 分鐘報告一次。您可以使用此指標來判斷資料庫執行個體能以給定速率大幅提升其基準效能水準達到多長的時間。 此指標僅適用於下列執行個體類別： • Aurora MySQL：db.t2.small 、db.t2.medium 、db.t3 和 db.t4g • Aurora PostgreSQL：db.t3 和 db.t4g Note 建議您在開發、測試伺服器或其他非生產伺服器時，僅使用 T 資料庫執行個體類別。如需詳細了解 T 執行個體類別，請參閱資料庫執行個體類別的類型。 啟動積分在 Amazon RDS 中的運作方式與 Amazon EC2 相同。如需詳細資訊，請參閱《Amazon Elastic Compute	Aurora MySQL 和 Aurora PostgreSQL	計數

指標	描述	適用對象	個單位
	Cloud Linux 執行個體使用者指南》中的啟動積分。		
CPUCreditUsage	指定期間內耗用的 CPU 額度，每隔 5 分鐘報告一次。此指標會計量實體 CPU 配置到資料庫執行個體之虛擬 CPU 用於處理指令的時間量。 此指標僅適用於下列執行個體類別： • Aurora MySQL：db.t2.small 、db.t2.medium 、db.t3 和 db.t4g • Aurora PostgreSQL：db.t3 和 db.t4g **Note** 建議您在開發、測試伺服器或其他非生產伺服器時，僅使用 T 資料庫執行個體類別。如需詳細了解 T 執行個體類別，請參閱資料庫執行個體類別的類型。	Aurora MySQL 和 Aurora PostgreSQL	計數

指標	描述	適用對象	個單位
CPUSurplusCreditBalance	無限制執行個體已在其 CPUCreditBalance 值為 0 時支出的剩餘額度數量。 CPUSurplusCreditBalance 值由獲得的 CPU 額度支付。如果剩餘額度超過執行個體在 24 小時期間可獲得的最大額度數量，超過最大值的支出剩餘額度將必須負擔額外的費用。 CPU 額度指標僅提供 5 分鐘頻率。	Aurora MySQL 和 Aurora PostgreSQL	額度 (vCPU-分鐘)
CPUSurplusCreditsCharged	若支出剩餘額度數量未由獲得的 CPU 額度付清，會產生額外的費用。 發生以下任何情況時，將收取支出剩餘額度的費用。 • 支出剩餘額度超過執行個體在 24 小時期間可獲得的最大額度數量。在小時結束時，將收取超過最大值的支出剩餘額度的費用。 • 執行個體已停止或終止。 • 執行個體從 unlimited 切換至 standard。 CPU 額度指標僅提供 5 分鐘頻率。	Aurora MySQL 和 Aurora PostgreSQL	額度 (vCPU-分鐘)

指標	描述	適用對象	個單位
CPUUtilization	由 Aurora 資料庫執行個體使用的 CPU 的百分比。	Aurora MySQL 和 Aurora PostgreSQL	百分比
DatabaseConnections	連線至資料庫執行個體的用戶端網路連線數。 資料庫工作階段數可能高於指標值，因為指標值不包括以下項目： • 不再有網路連線但尚未清理資料庫的工作階段 • 資料庫引擎為供自己使用建立的工作階段 • 資料庫引擎的平行執行功能建立的工作階段 • 資料庫引擎任務排程器建立的工作階段 • Amazon Aurora 連線	Aurora MySQL 和 Aurora PostgreSQL	計數
DDLLatency	請求的平均持續時間，例如，建立、更改和丟棄請求。	Aurora MySQL	毫秒
DDLThroughput	平均每秒的 DDL 請求數量。	Aurora MySQL	每秒計數
Deadlocks	資料庫中平均每秒的死鎖數量。	Aurora MySQL 和 Aurora PostgreSQL	每秒計數
DeleteLatency	刪除操作的平均持續時間。	Aurora MySQL	毫秒

指標	描述	適用對象	個單位
DeleteThroughput	平均每秒刪除查詢的數量。	Aurora MySQL	每秒計數
DiskQueueDepth	等待存取磁碟的未完成讀/寫請求的數量。	Aurora MySQL 和 Aurora PostgreSQL	計數
DMLLatency	插入、更新和刪除的平均持續時間。	Aurora MySQL	毫秒
DMLThroughput	平均每秒插入、更新及刪除的數量。	Aurora MySQL	每秒計數
EngineUptime	執行個體已執行的時間量。	Aurora MySQL 和 Aurora PostgreSQL	秒鐘
FreeableMemory	可用的隨機存取記憶體的數量。對於 Aurora MySQL 和 Aurora PostgreSQL 資料庫，此指標會報告 MemAvailable 欄位的值 /proc/meminfo 。	Aurora MySQL 和 Aurora PostgreSQL	位元組
FreeEphemeralStorage	可用的暫時性 NVMe 儲存量。	Aurora PostgreSQL	位元組

指標	描述	適用對象	個單位
FreeLocalStorage	本機可用儲存空間的數量。 不同於其他資料庫引擎，針對 Aurora 資料庫執行個體，此指標回報可供各資料庫執行個體使用的儲存空間。此值取決於資料庫執行個體類別 (如需定價資訊，請參閱 Amazon RDS 定價頁面)。您可以為執行個體選擇較大的資料庫執行個體，以增加執行個體的可用儲存空間量。 (這不適用於 Aurora Serverless v2。)	Aurora MySQL 和 Aurora PostgreSQL	位元組
InsertLatency	插入操作的平均持續時間。	Aurora MySQL	毫秒
InsertThroughput	插入操作的每秒平均數量。	Aurora MySQL	每秒計數
LoginFailures	平均每秒失敗的登入嘗試的數量。	Aurora MySQL	每秒計數
MaximumUsedTransactionIDs	最舊的未清理交易 ID 的存留期，以交易為單位。如果此值達到 2,146,483,648 (2^31 - 1,000,000)，資料庫將強制進入唯讀模式，以避免交易 ID 包圍。如需詳細資訊，請參閱 PostgreSQL 文件中的避免交易 ID 包圍失敗。	Aurora PostgreSQL	計數

指標	描述	適用對象	個單位
NetworkReceiveThroughput	Aurora 資料庫叢集中的各個執行個體從用戶端接收到的網路輸送量。此傳輸量不包含 Aurora 資料庫叢集中的執行個體與叢集磁碟區之間的網路流量。	Aurora MySQL 和 Aurora PostgreSQL	每秒位元組 (主控台每秒顯示 MB)
NetworkThroughput	Aurora 資料庫叢集中的各個執行個體從用戶端接收及傳送至用戶端的網路輸送量。此傳輸量不包含 Aurora 資料庫叢集中的執行個體與叢集磁碟區之間的網路流量。	Aurora MySQL 和 Aurora PostgreSQL	每秒位元組數
NetworkTransmitThroughput	Aurora 資料庫叢集中的各個執行個體傳送至用戶端的網路傳輸量。此輸送量不包含資料庫叢集中的執行個體與叢集磁碟區之間的網路流量。	Aurora MySQL 和 Aurora PostgreSQL	每秒位元組 (主控台每秒顯示 MB)
NumBinaryLogFiles	產生的 Binlog 檔案數目。	Aurora MySQL	計數
OldestReplicationSlotLag	以接收到 WAL 資料而言，複本遲延程度最大的遲延大小。	Aurora PostgreSQL	位元組
PurgeBoundary	允許 InnoDB 清除的交易編號上限。如果此指標長時間沒有推進，則表示 InnoDB 清除被長時間執行的交易封鎖。若要調查，請檢查 Aurora MySQL 資料庫叢集上的作用中交易。	Aurora MySQL 第 2 版、2.11 版及更新版本 Aurora MySQL 第 3 版、3.08 版及更新版本	計數

指標	描述	適用對象	個單位
PurgeFinishedPoint	執行 InnoDB 清除的交易編號上限。此指標可協助您檢查進行 InnoDB 清除的速度。	Aurora MySQL 第 2 版、2.11 版及更新版本 Aurora MySQL 第 3 版、3.08 版及更新版本	計數
Queries	平均每秒執行的查詢數量。	Aurora MySQL	每秒計數
RDSToAuroraPostgreSQLReplicaLag	將來自主要 RDS PostgreSQL 執行個體之更新複寫至叢集中其他節點時的延遲量。	Aurora PostgreSQL 的複本	秒鐘
ReadIOPS	每秒磁碟 I/O 操作的平均數量。但每隔一分鐘分別回報讀取與寫入。	Aurora MySQL 和 Aurora PostgreSQL	每秒計數
ReadIOPSEphemeralStorage	暫時性 NVMe 儲存的磁碟讀取 I/O 操作的平均數量。	Aurora PostgreSQL	每秒計數
ReadLatency	平均每次磁碟輸入/輸出操作耗用的時間量。	Aurora MySQL 和 Aurora PostgreSQL	秒鐘
ReadLatencyEphemeralStorage	暫時性 NVMe 儲存每次磁碟讀取 I/O 操作所花費的平均時間。	Aurora PostgreSQL	秒鐘

指標	描述	適用對象	個單位
ReadThroughput	平均每秒從磁碟讀取的位元組數目。	Aurora MySQL 和 Aurora PostgreSQL	每秒位元組數
ReadThroughputEphemeralStorage	暫時性 NVMe 儲存每秒從磁碟讀取的平均位元組數。	Aurora PostgreSQL	每秒位元組數
ReplicationSlotDiskUsage	複寫插槽檔案佔用的磁碟空間量。	Aurora PostgreSQL	位元組
ResultSetCacheHitRatio	由 Resultset 快取提供服務的請求的百分比。	Aurora MySQL 第 2 版	百分比
RollbackSegmentHistoryListLength	記錄具有刪除標記記錄之已遞交交易的復原日誌。這些記錄已排程由 InnoDB 清除操作處理。	Aurora MySQL	計數
RowLockTime	針對 InnoDB 資料表取得資料列鎖定所花費的時間總計。	Aurora MySQL	毫秒
SelectLatency	選取操作的平均時間量。	Aurora MySQL	毫秒
SelectThroughput	平均每秒選取查詢的數量。	Aurora MySQL	每秒計數
ServerlessDatabaseCapacity	Aurora Serverless 資料庫叢集的當前容量。	Aurora MySQL 和 Aurora PostgreSQL	計數

指標	描述	適用對象	個單位
StorageNetworkReceiveThroughput	資料庫叢集中的各個執行個體從 Aurora 儲存子系統接收到的網路輸送量。	Aurora MySQL 和 Aurora PostgreSQL	每秒位元組數
StorageNetworkThroughput	Aurora 資料庫叢集中的各個執行個體從 Aurora 儲存子系統接收和向其輸入的網路輸送量。	Aurora MySQL 和 Aurora PostgreSQL	每秒位元組數
StorageNetworkTransmitThroughput	Aurora 資料庫叢集中的各個執行個體傳送至 Aurora 儲存子系統的網路輸送量。	Aurora MySQL 和 Aurora PostgreSQL	每秒位元組數
SumBinaryLogSize	Binlog 檔案的大小總計。	Aurora MySQL	位元組
SwapUsage	交換空間的量。此指標不適用於下列資料庫執行個體類別： • db.r3.*、db.r4.* 和 db.r7g.* (Aurora MySQL) • db.r7g.* (Aurora PostgreSQL)	Aurora MySQL 和 Aurora PostgreSQL	位元組
TempStorageIOPS	在附加至資料庫執行個體的本機儲存體上讀取和寫入兩者的 IOPS 數目。此指標表示計數，且每秒測量一次。 此指標僅適用於 Aurora Serverless v2。	Aurora MySQL 和 Aurora PostgreSQL	每秒計數

指標	描述	適用對象	個單位
TempStorageThroughput	傳入和傳出與資料庫執行個體相關聯之本機儲存體的資料量。此指標表示位元組，且每秒測量一次。 此指標僅適用於 Aurora Serverless v2。	Aurora MySQL 和 Aurora PostgreSQL	每秒位元組數
TransactionAgeMaximum	最舊作用中執行交易的存留期。	Aurora MySQL 第 3 版、3.08 版及更新版本	秒鐘
TransactionLogsDiskUsage	交易日誌在 Aurora PostgreSQL 資料庫執行個體上佔用的磁碟空間量。 只有在 Aurora PostgreSQL 使用邏輯複寫或 時，才會產生此指標 AWS Database Migration Service。根據預設，Aurora PostgreSQL 會使用日誌記錄，而不是交易日誌。交易日誌不在使用中時，此指標的數值為 -1。	主要適用於 Aurora PostgreSQL	位元組
TruncateFinishedPoint	執行復原截斷的交易識別符上限。	Aurora MySQL 第 2 版、2.11 版及更新版本 Aurora MySQL 第 3 版、3.08 版及更新版本	計數

指標	描述	適用對象	個單位
UpdateLatency	更新操作所需要的平均時間。	Aurora MySQL	毫秒
UpdateThroughput	平均每秒更新的數量。	Aurora MySQL	每秒計數
WriteIOPS	每秒產生的 Aurora 儲存寫入記錄數目。這或多或少是資料庫所產生的日誌記錄數目。這些不會對應至 8K 分頁寫入，且不會對應至傳送的網路封包。	Aurora MySQL 和 Aurora PostgreSQL	每秒計數
WriteIOPSEphemeralStorage	暫時性 NVMe 儲存的磁碟寫入 I/O 操作的平均數量。	Aurora PostgreSQL	每秒計數
WriteLatency	平均每次磁碟輸入/輸出操作耗用的時間量。	Aurora MySQL 和 Aurora PostgreSQL	秒鐘
WriteLatencyEphemeralStorage	暫時性 NVMe 儲存每次磁碟寫入 I/O 操作所花費的平均時間。	Aurora PostgreSQL	秒鐘
WriteThroughput	平均每秒寫入永久儲存體的位元組數。	Aurora MySQL 和 Aurora PostgreSQL	每秒位元組數
WriteThroughputEphemeralStorage	暫時性 NVMe 儲存每秒寫入磁碟的平均位元組數。	Aurora PostgreSQL	每秒位元組數

Amazon Aurora 的 Amazon CloudWatch 用量指標

Amazon CloudWatch 中的 AWS/Usage 命名空間包括 Amazon RDS service quotas 的帳戶層級用量指標。CloudWatch 會自動收集所有 的使用指標 AWS 區域。

如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的 CloudWatch 用量指標。如需詳細資訊，請參閱《Service Quotas 使用者指南》中的請求提高配額。

指標	Description	單位*
AuthorizationsPerDBSecurityGroup	您 AWS 帳戶中每個資料庫安全群組的輸入規則數目。使用的值是帳戶中資料庫安全群組中輸入規則的數量上限。帳戶中的其他資料庫安全群組可能會有較少的輸入規則。	計數
CustomEndpointsPerDBCluster	您 AWS 帳戶中每個資料庫叢集的自訂端點數量。使用的值是帳戶中資料庫叢集中自訂端點的數量上限。帳戶中的其他資料庫叢集可能會有較少數量的自訂端點。	計數
DBClusterParameterGroups	AWS 帳戶中資料庫叢集參數群組的數量上限。計數不包括預設參數群組。	計數
DBClusterRoles	您 中每個資料庫叢集的 associated AWS Identity and Access Management (IAM) 角色數量 AWS 帳戶。使用的值是帳戶中資料庫叢集的關聯 IAM 角色數量上限。帳戶中的其他資料庫叢集可能有較少的相關聯 IAM 角色。	計數
DBClusters	AWS 帳戶中 Amazon Aurora 資料庫叢集的數量。	計數
DBInstanceRoles	您 中每個資料庫執行個體的 associated AWS Identity and Access Management (IAM) 角色數量 AWS 帳戶。使用的值是帳戶中資料庫執行個體的關聯 IAM 角色數量上限。帳戶中的其他資料庫執行個體可能有較少的關聯 IAM 角色。	計數
DBInstances	AWS 帳戶中的資料庫執行個體數量。	計數
DBParameterGroups	AWS 帳戶中資料庫參數群組的數量。計數不包括預設的資料庫參數群組。	計數

指標	Description	單位*
DBSubnetGroups	AWS 帳戶中的資料庫子網路群組數量。計數不包括預設子網路組。	計數
EventSubscriptions	AWS 帳戶中事件通知訂閱的數量。	計數
Integrations	您 AWS 帳戶中與 Amazon Redshift 的零 ETL 整合次數。	計數
ManualClusterSnapshots	AWS 帳戶中手動建立的資料庫叢集快照數量。計數不包括無效的快照。	計數
OptionGroups	AWS 帳戶中的選項群組數量。計數不包括預設選項群組。	計數
Proxies	您 AWS 帳戶中 RDS 代理的數目。	計數
ReadReplicasPerMaster	您帳戶中每個資料庫執行個體的僅供讀取複本數量。使用的值是帳戶中資料庫執行個體的僅供讀取複本數量上限。帳戶中的其他資料庫執行個體可能會有較少的僅供讀取複本數量。	計數
ReservedDBInstances	AWS 帳戶中的保留資料庫執行個體數量。計數不包括淘汰或拒絕的執行個體。	計數
SubnetsPerDBSubnetGroup	您 AWS 帳戶中每個資料庫子網路群組的子網路數量。帳戶中資料庫子網路群組的子網路數量上限。帳戶中的其他資料庫子網路群組可能有較少的子網路數量。	計數

> **Note**
>
> * Amazon RDS 不會將使用量指標的單位發佈到 CloudWatch。這些單位僅顯示於文件中。

Aurora 的 Amazon CloudWatch 維度

您可以使用下表中的任何維度來篩選 Aurora 指標資料。

維度	篩選請求的資料 . . .
DBInstanceIdentifier	特定的資料庫執行個體。
DBClusterIdentifier	特定的 Aurora 資料庫叢集。
DBClusterIdentifier, Role	特定的 Aurora 資料庫叢集，依執行個體角色 (WRITER/READER) 彙總指標。例如，您可以為屬於某個叢集的所有讀取者執行個體彙總指標。
DbClusterIdentifier, EngineName	特定的 Aurora 資料庫叢集和引擎名稱組合。例如，您可檢視叢集 ams1 和引擎 aurora 的 VolumeReadIOPs 指標。
DatabaseClass	資料庫類別中的所有執行個體。例如，您可以為屬於資料庫類別 db.r5.large 的所有執行個體彙總指標。
EngineName	僅已識別的引擎名稱。例如，您可以為具有引擎名稱 aurora-postgresql 的所有執行個體彙總指標。
SourceRegion	僅指定的區域。例如，您可以針對 us-east-1 區域中的所有資料庫執行個體來彙總指標。

Amazon RDS 主控台中 Aurora 指標的可用性

在 Amazon RDS 主控台中，並非所有由 Amazon Aurora 提供的指標都可以使用。您可以使用 CLI AWS 和 CloudWatch API 等工具來檢視這些指標。此外，在 Amazon RDS 主控台中的某些指標只針對特定執行個體類別顯示，或以不同的名稱和測量單位顯示。

主題

- 「過去一小時」檢視中可用的 Aurora 指標
- 在特定情況下可用的 Aurora 指標
- 在主控台中無法使用的 Aurora 指標

「過去一小時」檢視中可用的 Aurora 指標

您可以在 Amazon RDS 主控台的預設 Last Hour (過去一小時) 檢視中檢視已分類之 Aurora 指標的子集。下表列出針對 Aurora 執行個體顯示在 Amazon RDS 主控台中的類別和關聯指標。

類別	指標
SQL	ActiveTransactions BlockedTransactions BufferCacheHitRatio CommitLatency CommitThroughput DatabaseConnections DDLLatency DDLThroughput Deadlocks DMLLatency DMLThroughput LoginFailures ResultSetCacheHitRatio SelectLatency SelectThroughput
系統	AuroraReplicaLag AuroraReplicaLagMaximum AuroraReplicaLagMinimum CPUCreditBalance CPUCreditUsage CPUUtilization

類別	指標
	FreeableMemory FreeLocalStorage (這不適用於 Aurora Serverless v2。) NetworkReceiveThroughput
部署	AuroraReplicaLag BufferCacheHitRatio ResultSetCacheHitRatio SelectThroughput

在特定情況下可用的 Aurora 指標

此外，可在 Aurora 主控台中使用的部分指標只針對特定執行個體類別或只針對資料庫執行個體顯示，或以不同的名稱和不同的測量單位顯示：

- 僅對 Aurora MySQL CPUCreditBalance 執行個體類別和 Aurora PostgreSQL CPUCreditUsage 執行個體類別顯示 db.t2 和 db.t3 指標。
- 以下列出以不同名稱顯示的指標：

指標	顯示名稱
AuroraReplicaLagMaximum	最大複本延遲
AuroraReplicaLagMinimum	最小複本延遲
DDLThroughput	DDL
NetworkReceiveThroughput	網路輸送量
VolumeBytesUsed	[已計費] 已使用的磁碟區位元組
VolumeReadIOPs	[已計費] 磁碟區讀取 IOPS
VolumeWriteIOPs	[已計費] 磁碟區寫入 IOPS

- 下列指標會套用至整個 Aurora 資料庫叢集，但僅當在 Amazon RDS 主控台中檢視 Aurora 資料庫叢集的資料庫執行個體時才會顯示：
  - VolumeBytesUsed
  - VolumeReadIOPs
  - VolumeWriteIOPs
- 下列指標在 Amazon RDS 主控台中是以 MB 而非位元組為單位顯示：
  - FreeableMemory
  - FreeLocalStorage
  - NetworkReceiveThroughput
  - NetworkTransmitThroughput
- 下列指標適用於具有 Aurora Optimized Reads 的 Aurora PostgreSQL 資料庫叢集：
  - AuroraOptimizedReadsCacheHitRatio
  - FreeEphemeralStorage
  - ReadIOPSEphemeralStorage
  - ReadLatencyEphemeralStorage
  - ReadThroughputEphemeralStorage
  - WriteIOPSEphemeralStorage
  - WriteLatencyEphemeralStorage
  - WriteThroughputEphemeralStorage

在主控台中無法使用的 Aurora 指標

Amazon RDS 主控台中未提供下列 Aurora 指標：

- AuroraBinlogReplicaLag
- DeleteLatency
- DeleteThroughput
- EngineUptime
- InsertLatency
- InsertThroughput
- NetworkThroughput
- Queries

- UpdateLatency
- UpdateThroughput

Amazon RDS Performance Insights 的 Amazon CloudWatch 指標

Performance Insights 會自動將某些指標發佈至 Amazon CloudWatch。您可以透過績效詳情來查詢相同的資料，但在 CloudWatch 中具備這些指標可讓您輕鬆新增 CloudWatch 警示。您也可以輕鬆將指標新增至現有的 CloudWatch 儀表板。

指標	描述
DBLoad	資料庫的作用中工作階段數量。您通常會需要平均作用中工作階段數量的資料。在績效詳情中，系統會以 db.load.avg 的形式來查詢此資料。
DBLoadCPU	當等待事件類型為 CPU 時，作用中工作階段的數量。在績效詳情中，系統會以 db.load.avg 的形式來查詢此資料，篩選依據為等待事件類型 CPU。
DBLoadNonCPU	當等待事件類型不是 CPU 時，作用中工作階段的數量。
DBLoadRelativeToNumVCPUs	資料庫負載與資料庫虛擬 CPU 數量的比率。

> **Note**
>
> 只有當資料庫執行個體上有負載時，這些指標才會發佈至 CloudWatch。

您可以使用 CloudWatch 主控台、AWS CLI 或 CloudWatch API 來檢視這些指標。您也可以使用特殊指標數學函數來檢查其他 Performance Insights 計數器指標。如需更多詳細資訊，請參閱 在 CloudWatch 中查詢其他 Performance Insights 計數器指標。

例如，您可以執行 get-metric-statistics 命令來取得 DBLoad 指標的統計。

```
aws cloudwatch get-metric-statistics \
    --region us-west-2 \
    --namespace AWS/RDS \
    --metric-name DBLoad  \
    --period 60 \
    --statistics Average \
    --start-time 1532035185 \
    --end-time 1532036185 \
    --dimensions Name=DBInstanceIdentifier,Value=db-loadtest-0
```

此範例會產生類似下列範例的輸出結果。

```
{
  "Datapoints": [
  {
  "Timestamp": "2021-07-19T21:30:00Z",
  "Unit": "None",
  "Average": 2.1
  },
  {
  "Timestamp": "2021-07-19T21:34:00Z",
  "Unit": "None",
  "Average": 1.7
  },
  {
  "Timestamp": "2021-07-19T21:35:00Z",
  "Unit": "None",
  "Average": 2.8
  },
  {
  "Timestamp": "2021-07-19T21:31:00Z",
  "Unit": "None",
  "Average": 1.5
  },
  {
  "Timestamp": "2021-07-19T21:32:00Z",
  "Unit": "None",
  "Average": 1.8
  },
  {
  "Timestamp": "2021-07-19T21:29:00Z",
  "Unit": "None",
  "Average": 3.0
```

```
},
{
"Timestamp": "2021-07-19T21:33:00Z",
"Unit": "None",
"Average": 2.4
}
],
"Label": "DBLoad"
}
```

如需 CloudWatch 的詳細資訊，請參閱 Amazon CloudWatch 使用者指南 中的什麼是 Amazon CloudWatch？。

在 CloudWatch 中查詢其他 Performance Insights 計數器指標

> **Note**
>
> 如果您啟用 Database Insights 的進階模式，Amazon RDS 會將 Performance Insights 計數器指標發佈至 Amazon CloudWatch。透過 Database Insights，您不需要使用 `DB_PERF_INSIGHTS` 指標數學函數。您可以使用 CloudWatch Database Insights 儀表板來搜尋、查詢和設定 Performance Insights 計數器指標的警示。

您可以從 CloudWatch 對 RDS Performance Insights 指標進行查詢、發出警示及繪製圖表。您可以使用 CloudWatch 的 `DB_PERF_INSIGHTS` 指標數學函數來存取資料庫叢集的相關資訊。此函數可讓您使用未直接報告給 CloudWatch 的 Performance Insights 指標來建立新的時間序列。

您可以在 CloudWatch 主控台的選取指標畫面中按一下新增數學下拉式功能表，以使用新的指標數學函數。您可以使用它在 Performance Insights 指標或 CloudWatch 和 Performance Insights 指標的組合上建立警示和圖表，包括次分鐘指標的高解析度警示。您也可以在 `get-metric-data` 請求中包含指標數學運算式，以程式設計方式使用函數。如需詳細資訊，請參閱指標數學語法和函數，以及從 AWS 資料庫在 Performance Insights 計數器指標上建立警示。

Performance Insights 計數器指標

計數器指標是 Performance Insights 儀表板中的作業系統和資料庫效能指標。若要協助識別並分析效能問題，您可以將計數器指標與資料庫負載相互關聯。您必須將統計函數附加至指標，以取得指標值。例如，`os.memory.active` 指標支援的函數為 `.avg`、`.min`、`.max`、`.sum` 和 `.sample_count`。

每分鐘收集一次計數器指標。作業系統指標收集方式取決於增強型監控是開啟或關閉。如果關閉增強型監控，系統每分鐘收集一次作業系統指標。如果開啟增強型監控，系統會在選定期間內收集作業系統指標。如需增強型監控的詳細資訊，請參閱 開啟和關閉增強型監控。

主題

- 績效詳情作業系統計數器
- Aurora MySQL 的績效詳情計數器
- Aurora PostgreSQL 的績效詳情計數器

績效詳情作業系統計數器

對於 Aurora PostgreSQL 和 Aurora MySQL，下列作業系統計數器 (其字首為 os) 可與 Performance Insights 搭配使用。

您可以將 `ListAvailableResourceMetrics` API 用於資料庫執行個體的可用計數器指標清單。如需詳細資訊，請參閱《Amazon RDS Performance Insights API 參考指南》中的 ListAvailableResourceMetrics。

計數器	類型	單位	指標	Description
作用中	記憶體	KB	os.memory.active	已指派的記憶體數量，以 KB 為單位。
緩衝區域	記憶體	KB	os.memory.buffers	在寫入至儲存裝置之前，用於緩衝 I/O 請求的記憶體數量，以 KB 為單位。
已快取	記憶體	KB	os.memory.cached	用於快取檔案系統型 I/O 的記憶體數量，以 KB 為單位。
資料庫快取	記憶體	位元組	os.memory.db.cache	資料庫程序 (包括 tmpfs (shmem)) 用於頁面快取的

計數器	類型	單位	指標	Description
				記憶體數量，以位元組為單位。
資料庫常駐集大小	記憶體	位元組	os.memory.db.residentSetSize	資料庫程序 (不包括 tmpfs (shmem)) 用於匿名和交換快取的記憶體數量，以位元組為單位。
資料庫交換	記憶體	位元組	os.memory.db.swap	資料庫程序用於交換的記憶體數量，以位元組為單位。
髒	記憶體	KB	os.memory.dirty	RAM 之中已修改但尚未寫入至儲存裝置中相關資料區塊的記憶體分頁數量，以 KB 為單位。
免費	記憶體	KB	os.memory.free	未指派的記憶體數量，以 KB 為單位。
釋出的大內存頁	記憶體	頁面	os.memory.hugePagesFree	自由巨型分頁的數量。巨型分頁為 Linux 核心的功能。
保留的大內存頁	記憶體	頁面	os.memory.hugePagesRsvd	已遞交的巨型分頁的數量。
大內存頁尺寸	記憶體	KB	os.memory.hugePagesSize	每個巨型分頁的大小，以 KB 為單位。

計數器	類型	單位	指標	Description
抑制的大內存頁	記憶體	頁面	os.memory .hugePagesSurp	超過總數的可用剩餘巨型分頁的數量。
大內存頁總數	記憶體	頁面	os.memory .hugePagesTotal	大內存頁總數。
非作用中	記憶體	KB	os.memory .inactive	使用頻率最低的記憶體分頁數量，以 KB 為單位。
已對應	記憶體	KB	os.memory .mapped	在程序地址空間內映射的檔案系統內容的總量，以 KB 為單位。
記憶體不足終止計數	記憶體	終止	os.memory .outOfMem oryKillCount	上次收集間隔內發生的 OOM 終止數目。
內存頁資料表	記憶體	KB	os.memory .pageTables	分頁表使用的記憶體數量，以 KB 為單位。
板	記憶體	KB	os.memory.slab	個重複使用的核心資料結構數量，以 KB 為單位。
總計	記憶體	KB	os.memory.total	記憶體總量，以 KB 為單位。

計數器	類型	單位	指標	Description
回寫	記憶體	KB	os.memory.writeback	RAM 之中仍被寫入至支援儲存裝置的中途分頁數量，以 KB 為單位。
訪客	CPU 使用率	百分比	os.cpuUtilization.guest	客體程式使用中的 CPU 百分比。
閒置	CPU 使用率	百分比	os.cpuUtilization.idle	CPU 閒置的百分比。
Irq	CPU 使用率	百分比	os.cpuUtilization.irq	軟體中斷使用中的 CPU 百分比。
良好	CPU 使用率	百分比	os.cpuUtilization.nice	以最低優先順序執行之程式使用中的 CPU 百分比。
挪用	CPU 使用率	百分比	os.cpuUtilization.steal	其他虛擬機器使用中的 CPU 百分比。
系統	CPU 使用率	百分比	os.cpuUtilization.system	核心使用中的 CPU 百分比。
總計	CPU 使用率	百分比	os.cpuUtilization.total	使用中的 CPU 總百分比。此值包含良好值。
使用者	CPU 使用率	百分比	os.cpuUtilization.user	使用者程式使用中的 CPU 百分比。

計數器	類型	單位	指標	Description
等候	CPU 使用率	百分比	os.cpuUtilization.wait	等待 I/O 存取時未使用的 CPU 百分比。
Aurora 儲存 Aurora 儲存接收的位元組數目	磁碟 IO	每秒位元組數	os.diskIO.auroraStorage.auroraStorageBytesRx	Aurora 儲存體每秒接收的位元組數目。
Aurora 儲存 Aurora 儲存傳輸的位元組數目	磁碟 IO	每秒位元組數	os.diskIO.auroraStorage.auroraStorageBytesTx	每秒上傳至 Aurora 儲存體的位元組數目。
Aurora 儲存磁碟佇列深度	磁碟 IO	要求	os.diskIO.auroraStorage.diskQueueDepth	Aurora 儲存體磁碟佇列的長度。
Aurora 儲存讀取 IO PS	磁碟 IO	每秒請求數	os.diskIO.auroraStorage.readIOsPS	每秒讀取操作的次數。
Aurora Storage 讀取延遲	磁碟 IO	毫秒	os.diskIO.auroraStorage.readLatency	Aurora 儲存體的讀取 I/O 要求平均延遲，以毫秒為單位。
Aurora 儲存體讀取輸送量	磁碟 IO	每秒位元組數	os.diskIO.auroraStorage.readThroughput	要求資料庫叢集所使用的網路輸送量，以每秒位元組為單位。

計數器	類型	單位	指標	Description
Aurora 儲存寫入 IO PS	磁碟 IO	每秒請求數	os.diskIO .auroraSt orage.wri teIOsPS	每秒寫入操作的次數。
Aurora 儲存寫入延遲	磁碟 IO	毫秒	os.diskIO .auroraSt orage.wri teLatency	Aurora 儲存體的寫入 I/O 要求平均延遲，以毫秒為單位。
Aurora Storage 寫入輸送量	磁碟 IO	每秒位元組數	os.diskIO .auroraSt orage.wri teThroughput	資料庫叢集回應所使用的網路輸送量，以每秒位元組為單位。
Rdstemp 平均佇列長度	磁碟 IO	要求	os.diskIO .rdstemp. avgQueueLen	在 I/O 裝置的佇列中等待的要求數量。
Rdstemp 平均請求大小	磁碟 IO	要求	os.diskIO .rdstemp. avgReqSz	在 I/O 裝置的佇列中等待的要求數量。
Rdstemp 等候	磁碟 IO	毫秒	os.diskIO .rdstemp.await	回應請求時所需的毫秒數，包括佇列時間與服務時間。
Rdstemp 讀取 IO PS	磁碟 IO	要求	os.diskIO .rdstemp. readIOsPS	每秒讀取操作的次數。
Rdstemp 讀取 KB	磁碟 IO	KB	os.diskIO .rdstemp.readKb	讀取的 KB 總數。

計數器	類型	單位	指標	Description
Rdstemp 讀取 KB PS	磁碟 IO	每秒 KB	os.diskIO .rdstemp. readKbPS	每秒讀取的 KB 總數。
Rdstemp Rrqm PS	磁碟 IO	每秒請求數	os.diskIO .rdstemp.rrqmPS	每秒佇列的合併讀取請求數量。
Rdstemp TPS	磁碟 IO	每秒交易數	os.diskIO .rdstemp.tps	每秒的 I/O 交易數量。
Rdstemp 使用率	磁碟 IO	百分比	os.diskIO .rdstemp.util	發出請求的 CPU 時間百分比。
Rdstemp 寫入 IO PS	磁碟 IO	每秒請求數	os.diskIO .rdstemp. writeIOsPS	每秒寫入操作的次數。
Rdstemp 寫入 KB	磁碟 IO	KB	os.diskIO .rdstemp.writeKb	寫入的 KB 總數。
Rdstemp 寫入 KB PS	磁碟 IO	每秒 KB	os.diskIO .rdstemp. writeKbPS	每秒寫入的 KB 總數。
Rdstemp Wrqm PS	磁碟 IO	每秒請求數	os.diskIO .rdstemp. wrqmPS	每秒佇列的合併寫入請求數量。
封鎖	任務	任務	os.tasks.blocked	封鎖的任務數量。
執行中	任務	任務	os.tasks.running	執行中的任務數量。
休眠中	任務	任務	os.tasks.sleeping	睡眠中的任務數量。

計數器	類型	單位	指標	Description
已停止	任務	任務	os.tasks.stopped	已停止的任務數量。
總計	任務	任務	os.tasks.total	任務的總數。
廢止中	任務	任務	os.tasks.zombie	與作用中父任務進行互動的子任務數量。
一	負載平均分鐘	Processes	os.loadAverageMinute.one	過去 1 分鐘內請求 CPU 時間的程序數量。
十五	負載平均分鐘	Processes	os.loadAverageMinute.fifteen	過去 15 分鐘內請求 CPU 時間的程序數量。
五	負載平均分鐘	Processes	os.loadAverageMinute.five	過去 5 分鐘內請求 CPU 時間的程序數量。
已快取	交換	KB	os.swap.cached	做為快取記憶體使用的 swap 記憶體數量，以 KB 為單位。
免費	交換	KB	os.swap.free	可用的交換記憶體數量，以 KB 為單位。
In (入)	交換	KB	os.swap.in	從磁碟交換輸入的記憶體數量，以 KB 為單位。
Out (出)	交換	KB	os.swap.out	交換輸出到磁碟的記憶體數量，以 KB 為單位。

計數器	類型	單位	指標	Description
總計	交換	KB	os.swap.total	可用的交換記憶體總量，以 KB 為單位。
檔案數上限	檔案系統	檔案	os.fileSys.maxFiles	可在所有儲存磁碟區中為檔案系統建立的檔案數量上限。
已使用的檔案	檔案系統	檔案	os.fileSys.usedFiles	檔案系統中所有儲存磁碟區的檔案數量。
已使用的檔案百分比	檔案系統	檔案	os.fileSys.usedFilePercent	所有儲存磁碟區使用中的可用檔案百分比。
已使用百分比	檔案系統	百分比	os.fileSys.usedPercent	所有儲存磁碟區使用中的檔案系統磁碟空間百分比。
已使用	檔案系統	KB	os.fileSys.used	檔案系統中所有儲存磁碟區的檔案所使用的磁碟空間量，以 KB 為單位。
總計	檔案系統	KB	os.fileSys.total	所有儲存磁碟區中檔案系統可用的總磁碟空間，以 KB 為單位。
檔案數上限	檔案系統	檔案	os.fileSys.<volumeName>.maxFiles	可為儲存磁碟區建立的檔案數目上限。

計數器	類型	單位	指標	Description
已使用的檔案	檔案系統	檔案	os.fileSys.<volumeName>.usedFiles	儲存磁碟區中的檔案數量。
已使用的檔案百分比	檔案系統	檔案	os.fileSys.<volumeName>.usedFilePercent	儲存磁碟區中使用的可用檔案百分比。
已使用百分比	檔案系統	百分比	os.fileSys.<volumeName>.usedPercent	使用中的儲存磁碟區磁碟空間百分比。
已使用	檔案系統	KB	os.fileSys.<volumeName>.used	儲存磁碟區中檔案使用的磁碟空間量，以 KB 為單位。
總計	檔案系統	KB	os.fileSys.<volumeName>.total	儲存磁碟區中可用的總磁碟空間，以 KB 為單位。
Rx	網路	每秒位元組數	os.network.rx	每秒接收的位元組數量。
Tx	網路	每秒位元組數	os.network.tx	每秒上傳的位元組數量。
ACU 使用率	一般	百分比	os.general.acuUtilization	目前容量佔最大設定容量的百分比。

計數器	類型	單位	指標	Description
最大設定的 ACU	一般	ACU	os.general.maxConfiguredAcu	使用者設定的最大容量，以 Aurora Capacity Units (ACU) 為單位。
最小設定的 ACU	一般	ACU	os.general.minConfiguredAcu	使用者設定的最小容量，以 ACU 為單位。
VCPU 數目	一般	vCPUs	os.general.numVCPUs	資料庫執行個體的虛擬 CPU (vCPU) 數量。
無伺服器資料庫容量	一般	ACU	os.general.serverlessDatabaseCapacity	執行個體的目前容量，以 ACU 為單位。

Aurora MySQL 的績效詳情計數器

以下資料庫計數器適用於 Aurora MySQL 的績效詳情。

主題

- Aurora MySQL 的原生計數器
- Aurora MySQL 的非原生計數器

Aurora MySQL 的原生計數器

原生指標由資料庫引擎定義，而不是由 Amazon RDS。您可以在 MySQL 文件的伺服器狀態變數中找到這些原生指標的定義。

計數器	類型	單位	指標
Com_analyze	SQL	每秒查詢數	db.SQL.Com_analyze
Com_optimize	SQL	每秒查詢數	db.SQL.Com_optimize
Com_select	SQL	每秒查詢數	db.SQL.Com_select
Innodb_rows_deleted	SQL	每秒列數	db.SQL.Innodb_rows_deleted
Innodb_rows_inserted	SQL	每秒列數	db.SQL.Innodb_rows_inserted
Innodb_rows_read	SQL	每秒列數	db.SQL.Innodb_rows_read
Innodb_rows_updated	SQL	每秒列數	db.SQL.Innodb_rows_updated
查詢	SQL	每秒查詢數	db.SQL.Queries
問題	SQL	每秒查詢數	db.SQL.Questions
Select_full_join	SQL	每秒查詢數	db.SQL.Select_full_join
Select_full_range_join	SQL	每秒查詢數	db.SQL.Select_full_range_join
Select_range	SQL	每秒查詢數	db.SQL.Select_range
Select_range_check	SQL	每秒查詢數	db.SQL.Select_range_check
Select_scan	SQL	每秒查詢數	db.SQL.Select_scan

計數器	類型	單位	指標
Slow_queries	SQL	每秒查詢數	db.SQL.Slow_queries
Sort_merge_passes	SQL	每秒查詢數	db.SQL.Sort_merge_passes
Sort_range	SQL	每秒查詢數	db.SQL.Sort_range
Sort_rows	SQL	每秒查詢數	db.SQL.Sort_rows
Sort_scan	SQL	每秒查詢數	db.SQL.Sort_scan
Total_query_time	SQL	毫秒	db.SQL.Total_query_time
Table_locks_immediate	鎖定	每秒請求數	db.Locks.Table_locks_immediate
Table_locks_waited	鎖定	每秒請求數	db.Locks.Table_locks_waited
Innodb_row_lock_time	鎖定	毫秒 (平均)	db.Locks.Innodb_row_lock_time
Aborted_clients	使用者	連線	db.Users.Aborted_clients
Aborted_connects	使用者	連線	db.Users.Aborted_connects
連線	使用者	連線	db.Users.Connections
External_threads_connected	使用者	連線	db.Users.External_threads_connected
max_connections	使用者	連線	db.Users.max_connections
Threads_connected	使用者	連線	db.Users.Threads_connected
Threads_created	使用者	連線	db.Users.Threads_created

計數器	類型	單位	指標
Threads_running	使用者	連線	db.Users.Threads_running
Created_tmp_disk_tables	暫存	每秒資料表數	db.Temp.Created_tmp_disk_tables
Created_tmp_tables	暫存	每秒資料表數	db.Temp.Created_tmp_tables
Innodb_buffer_pool_pages_data	快取	頁面	db.Cache.Innodb_buffer_pool_pages_data
Innodb_buffer_pool_pages_total	快取	頁面	db.Cache.Innodb_buffer_pool_pages_total
Innodb_buffer_pool_read_requests	快取	每秒頁面數	db.Cache.Innodb_buffer_pool_read_requests
Innodb_buffer_pool_reads	快取	每秒頁面數	db.Cache.Innodb_buffer_pool_reads
Opened_tables	快取	資料表	db.Cache.Opened_tables
Opened_table_definitions	快取	資料表	db.Cache.Opened_table_definitions
Qcache_hits	快取	查詢	db.Cache.Qcache_hits

Aurora MySQL 的非原生計數器

非原生計數器指標是 Amazon RDS 定義的計數器。非原生指標可以是您使用特定查詢取得的指標。非原生指標也可以是衍生的指標，其中會將兩個以上的原生計數器用在計算中的比率、命中率或延遲。

計數器	類型	單位	指標	描述	定義
active_transactions	交易	db.Transactions.active_transaction	作用中交易總計。	`SELECT COUNT(1) AS active_transactions FROM  INFOR`	

計數器	類型	單位	指標	描述	定義
				MATION_SCHEMA.INNODB_TRX	
innodb_buffer_pool_hit_rate	快取	db.Cache.innoDB.buffer_pool_hit_rat	InnoDB 可從緩衝集區獲得的讀取百分比。	100 * innodb_buffer_pool_read_requests / (innodb_buffer_pool_read_requests + innodb_buffer_pool_reads)	
innodb_buffer_pool_hits	快取	每秒頁面數	db.Cache.innoDB_buffer_pool_hits	InnoDB 可從緩衝集區獲得的讀取數。	innodb_buffer_pool_read_requests - innodb_buffer_pool_reads

計數器	類型	單位	指標	描述	定義
innodb_buffer_pool_usage	快取	百分比	db.Cache.innoDB_buffer_pool_usage	包含資料 (頁面) 的 InnoDB 緩衝集區百分比。 **Note** 使用壓縮的資料表時，此值可能會有所不同。如需詳細資訊，請參閱 MySQL 文件伺服器狀態變數中與 `Innodb_buffer_pool_pages_data` 和 `Innodb_buffer_pool_pages_total` 相關的資訊。	`Innodb_buffer_pool_pages_data / Innodb_buffer_pool_pages_total * 100.0`
innodb_deadlocks	鎖定	db.Lock.innodb_deadlocks	死鎖總數。	`SELECT COUNT AS innodb_deadlocks FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_deadlocks'`	

計數器	類型	單位	指標	描述	定義
innodb_lock_timeouts	鎖定	db.Lock.innodb_lock_timeouts	逾時的死鎖總數。	SELECT COUNT AS innodb_lock_timeouts FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_timeouts'	
innodb_row_lock_waits	鎖定	db.Lock.innodb_row_lock_waits	造成等待的列鎖定總數。	SELECT COUNT AS innodb_row_lock_waits FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_row_lock_waits'	
innodb_rows_changed	SQL	db.SQL.innodb_rows_changed	InnoDB 列操作總計。	db.SQL.Innodb_rows_inserted + db.SQL.Innodb_rows_deleted + db.SQL.Innodb_rows_updated	

計數器	類型	單位	指標	描述	定義
query_cache_hit_rate	快取	百分比	db.Cache.query_cache_hit_rate	MySQL 結果集快取 (查詢快取) 命中率。	`Qcache_hits / (QCache_hits + Com_select) * 100`
temp_disk_tables_percent	暫存	db.Temp.tmp_disk_tables_percent	執行陳述式時，伺服器在磁碟上建立的暫時資料表百分比。	`(db.Temp.Created_tmp_disk_tables / db.Temp.Created_tmp_tables) * 100`	
trx_rseg_history_len	交易	無	db.Transactions.trx_rseg_history_len	InnoDB 交易系統所維護之已確認交易的還原日誌頁面清單，用於實作多版本並行控制。如需還原日誌詳情的詳細資訊，請參閱 MySQL 文件中的 https://dev.mysql.com/doc/refman/8.0/en/innodb-multi-versioning.html。	`SELECT COUNT AS trx_rseg_history_len FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='trx_rseg_history_len'`

Aurora PostgreSQL 的績效詳情計數器

以下資料庫計數器適用於 Aurora PostgreSQL 的績效詳情。

主題

- Aurora PostgreSQL 的原生計數器
- Aurora PostgreSQL 的非原生計數器

Aurora PostgreSQL 的原生計數器

原生指標由資料庫引擎定義，而不是由 Amazon RDS。您可以在 PostgreSQL 文件的檢視統計資料中找到這些原生指標的定義。

計數器	類型	單位	指標
tup_deleted	SQL	每秒元組數	db.SQL.tup_deleted
tup_fetched	SQL	每秒元組數	db.SQL.tup_fetched
tup_inserted	SQL	每秒元組數	db.SQL.tup_inserted
tup_returned	SQL	每秒元組數	db.SQL.tup_returned
tup_updated	SQL	每秒元組數	db.SQL.tup_updated
blks_hit	快取	每秒區塊數	db.Cache.blks_hit
buffers_alloc	快取	每秒區塊數	db.Cache.buffers_alloc
buffers_checkpoint	檢查點	每秒區塊數	db.Checkpoint.buffers_checkpoint
checkpoints_req	檢查點	每分鐘檢查點	db.Checkpoint.checkpoints_req
checkpoint_sync_time	檢查點	每個檢查點的毫秒數	db.Checkpoint.checkpoint_sync_time
checkpoints_timed	檢查點	每分鐘檢查點	db.Checkpoint.checkpoints_timed
checkpoint_write_time	檢查點	每個檢查點的毫秒數	db.Checkpoint.checkpoint_write_time
maxwritten_clean	檢查點	每分鐘的 Bgwriter 清除停止數	db.Checkpoint.maxwritten_clean

計數器	類型	單位	指標
deadlocks	並行數量	每分鐘的鎖死數	db.Concurrency.deadlocks
blk_read_time	輸入/輸出	毫秒	db.IO.blk_read_time
blks_read	輸入/輸出	每秒區塊數	db.IO.blks_read
buffers_backend	輸入/輸出	每秒區塊數	db.IO.buffers_backend
buffers_backend_fsync	輸入/輸出	每秒區塊數	db.IO.buffers_backend_fsync
buffers_clean	輸入/輸出	每秒區塊數	db.IO.buffers_clean
temp_bytes	暫存	每秒位元組數	db.Temp.temp_bytes
temp_files	暫存	每分鐘的檔案數	db.Temp.temp_files
xact_commit	交易	每秒遞交數	db.Transactions.xact_commit
xact_rollback	交易	每秒轉返數	db.Transactions.xact_rollback
numbackends	使用者	連線	db.User.numbackends
archived_count	WAL	每分鐘的檔案數	db.WAL.archived_count

Aurora PostgreSQL 的非原生計數器

非原生計數器指標是 Amazon Aurora 定義的計數器。非原生指標可以是您使用特定查詢取得的指標。非原生指標也可以是衍生的指標，其中會將兩個以上的原生計數器用在計算中的比率、命中率或延遲。

計數器	類型	單位	指標	描述	定義
checkpoint_sync_latency	Checkpoint	毫秒	db.Checkpoint.checkpoint_sync_latency	在檔案同步至磁碟的檢查點處理過程中所用的時間總計。	`checkpoint_sync_time / (checkpoints_timed + checkpoints_req)`
checkpoint_write_latency	Checkpoint	毫秒	db.Checkpoint.checkpoint_write_latency	在檔案寫入磁碟的檢查點處理過程中所用的時間總計。	`checkpoint_write_time / (checkpoints_timed + checkpoints_req)`
local_blks_read	輸入/輸出	區塊	db.IO.local_blks_read	讀取的本機區塊總數。	不適用
local_blk_read_time	輸入/輸出	毫秒	db.IO.local_blk_read_time	如果啟用 `track_io_timing` ，它會追蹤讀取本機資料檔案區塊所花費的總時間 (以毫秒為單位)，否則值為零。如需詳細資訊，請參閱 track_io_timing。	不適用
num_blocked_sessions	鎖定	db.Locks.num_blocked_sessions	封鎖的工作階段數目。	–	
orcache_blks_hit	輸入/輸出	查詢	db.IO.orcache_blks_hit	來自最佳化讀取快取的共用區塊命中總數。	不適用

計數器	類型	單位	指標	描述	定義
orcache_blk_read_time	輸入/輸出	毫秒	db.IO.orcache_blk_read_time	如果啟用 `track_io_timing` ，它會追蹤從 Optimized Reads 快取讀取資料檔案區塊所花費的總時間 (以毫秒為單位)，否則值為零。如需詳細資訊，請參閱 track_io_timing。	不適用
read_latency	輸入/輸出	毫秒	db.IO.read_latency	在此執行個體中後端讀取資料檔案區塊所用的時間。	`blk_read_time / blks_read`
storage_blks_read	輸入/輸出	區塊	db.IO.storage_blks_read	從 Aurora 儲存體讀取的共用區塊總數。	不適用
storage_blk_read_time	輸入/輸出	毫秒	db.IO.storage_blk_read_time	如果啟用 `track_io_timing` ，它會追蹤從 Aurora 儲存體讀取資料檔案區塊所花費的總時間 (以毫秒為單位)，否則值為零。如需詳細資訊，請參閱 track_io_timing。	不適用
num_blocked_sessions	鎖定	db.Locks.num_blocked_sessions	封鎖的工作階段數目。	–	
active_count	State	工作階段	db.state.active_count	處於 `active` 狀態的工作階段數目。	不適用

計數器	類型	單位	指標	描述	定義
idle_count	State	工作階段	db.state.idle_count	處於 idle 狀態的工作階段數目。	不適用
idle_in_transaction_aborted_count	State	工作階段	db.state.idle_in_transaction_aborted_count	處於 idle in transaction (aborted) 狀態的工作階段數目。	不適用
idle_in_transaction_count	State	工作階段	db.state.idle_in_transaction_count	處於 idle in transaction 狀態的工作階段數目。	不適用
idle_in_transaction_max_time	State	秒鐘	db.state.idle_in_transaction_max_time	處於 idle in transaction 狀態的最長執行時間交易的持續時間，以秒為單位。	不適用
logical_reads	SQL	區塊	db.SQL.logical_reads	命中和讀取的區塊總數。	blks_hit + blks_read
queries_started	SQL	查詢	db.SQL.queries	已啟動的查詢數目。	不適用
queries_finished	SQL	查詢	db.SQL.queries	已完成的查詢數目。	不適用
total_query_time	SQL	毫秒	db.SQL.total_query_time	執行陳述式所花費的時間總計，以毫秒為單位。	不適用

計數器	類型	單位	指標	描述	定義
active_transactions	交易	交易	db.Transactions.active_transactions	作用中交易的數目。	不適用
blocked_transactions	交易	交易	db.Transactions.blocked_transactions	封鎖的交易數目。	不適用
commit_latency	交易	微秒	db.Transactions.commit_latency	遞交操作的平均持續時間。	`db.Transactions.duration_commits / db.Transactions.xact_commit`
duration_commits	交易	毫秒	db.Transactions.duration_commits	最後一分鐘花費的交易時間總計，以毫秒為單位。	不適用
max_used_xact_ids	交易	交易	db.Transactions.max_used_xact_ids	尚未清空的交易數目。	不適用
oldest_inactive_logical_replication_slot_xid_age	交易	Length	db.Transactions.oldest_inactive_logical_replication_slot_xid_age	非作用中邏輯複寫插槽中最舊交易的存留期。	不適用

計數器	類型	單位	指標	描述	定義
oldest_active_logical_replication_slot_xid_age	交易	Length	db.Transactions.oldest_active_logical_replication_slot_xid_age	作用中邏輯複寫插槽中最舊交易的存留期。	不適用
oldest_reader_feedback_xid_age	交易	Length	db.Transactions.oldest_reader_feedback_xid_age	Aurora 讀取器執行個體或 Aurora 全域資料庫讀取器執行個體上長時間執行交易的最舊交易的存留期。	不適用
oldest_prepared_transaction_xid_age	交易	Length	db.Transactions.oldest_prepared_transaction_xid_age	最舊備妥交易的存留期。	不適用
oldest_running_transaction_xid_age	交易	Length	db.Transactions.oldest_running_transaction_xid_age	最舊執行交易的存留期。	不適用
max_connections	使用者	使用者	db.User.max_connections	在 `max_connections` 參數中設定的資料庫允許的最大連線數。	不適用

計數器	類型	單位	指標	描述	定義
total_auth_attempts	使用者	使用者	db.User.total_auth_attempts	對此執行個體的連線嘗試次數。	不適用
archive_failed_count	WAL	每分鐘的檔案數	db.WAL.archive_failed_count	封存 WAL 檔案的失敗嘗試次數，以每分鐘檔案數為單位。	不適用

績效詳情的 SQL 統計數字

SQL 統計數字是績效詳情所收集有關 SQL 查詢的效能相關指標。績效詳情會收集查詢執行的每一秒和每個 SQL 呼叫的統計資料。SQL 統計資料是所選時間範圍的平均值。

SQL 摘要是具有給定模式但不一定具有相同字面值的所有查詢的複合。摘要會以問號來取代字面值。例如 `SELECT * FROM emp WHERE lname= ?`。此摘要可能包含下列子查詢：

```
SELECT * FROM emp WHERE lname = 'Sanchez'
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

所有引擎都支援摘要查詢的 SQL 統計資料。

如需此功能的區域、資料庫引擎和執行個體類別支援的資訊，請參閱 Performance Insights 功能的 Amazon Aurora 資料庫引擎、區域和執行個體類別支援

主題

- Aurora MySQL 的 SQL 統計資料
- Aurora PostgreSQL 的 SQL 統計資料

Aurora MySQL 的 SQL 統計資料

Aurora MySQL 只收集摘要層級的 SQL 統計數字。不會顯示陳述式層級的統計數字。

主題

- Aurora MySQL 的摘要統計資料

- Aurora MySQL 的每秒統計數字
- Aurora MySQL 的每次呼叫統計數字
- Aurora MySQL 的主要統計資料

Aurora MySQL 的摘要統計資料

績效詳情會從 `events_statements_summary_by_digest` 資料表收集 SQL 摘要統計數字。`events_statements_summary_by_digest` 資料表由資料庫管理。

摘要資料表沒有移出政策。當資料表已滿時， AWS 管理主控台 會顯示下列訊息：

```
Performance Insights is unable to collect SQL Digest statistics on new queries because
 the table events_statements_summary_by_digest is full.
Please truncate events_statements_summary_by_digest table to clear the issue. Check the
 User Guide for more details.
```

在這種情況下，Aurora MySQL 不會追蹤 SQL 查詢。如要解決此問題，績效詳情會在滿足下列兩個條件時自動截斷摘要資料表：

- 表格已滿。
- 績效詳情會自動管理效能結構描述。

  若要進行自動管理，`performance_schema` 參數必須設定為 `0`，而 Source (來源) 不可設定為 `user`。如果績效詳情未自動管理效能結構描述，請參閱 Aurora MySQL 上 Performance Insights 的效能結構描述概觀。

在 中 AWS CLI，執行 describe-db-parameters 命令來檢查參數值的來源。

Aurora MySQL 的每秒統計數字

以下 SQL 統計資料適用於 Aurora MySQL 資料庫 叢集。

指標	單位
db.sql_tokenized.stats.count_star_per_sec	每秒呼叫數
db.sql_tokenized.stats.sum_timer_wait_per_sec	每秒平均延遲 (毫秒)

指標	單位
db.sql_tokenized.stats.sum_select_full_join_per_sec	選取每秒完整聯結
db.sql_tokenized.stats.sum_select_range_check_per_sec	選取每秒範圍檢查
db.sql_tokenized.stats.sum_select_scan_per_sec	選取每秒掃瞄
db.sql_tokenized.stats.sum_sort_merge_passes_per_sec	排序每秒合併路徑
db.sql_tokenized.stats.sum_sort_scan_per_sec	排序每秒掃描
db.sql_tokenized.stats.sum_sort_range_per_sec	排序每秒範圍
db.sql_tokenized.stats.sum_sort_rows_per_sec	排序每秒列數
db.sql_tokenized.stats.sum_rows_affected_per_sec	每秒影響的列數
db.sql_tokenized.stats.sum_rows_examined_per_sec	每秒檢查的列數
db.sql_tokenized.stats.sum_rows_sent_per_sec	每秒傳送的列數
db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_sec	每秒建立的暫存磁碟資料表
db.sql_tokenized.stats.sum_created_tmp_tables_per_sec	每秒建立的暫存資料表
db.sql_tokenized.stats.sum_lock_time_per_sec	每秒鎖定時間 (毫秒)

Aurora MySQL 的每次呼叫統計數字

下列指標提供 SQL 陳述式的每次呼叫統計數字。

指標	單位
db.sql_tokenized.stats.sum_timer_wait_per_call	每次呼叫平均延遲 (毫秒)
db.sql_tokenized.stats.sum_select_full_join_per_call	選取每個呼叫的完整聯結
db.sql_tokenized.stats.sum_select_range_check_per_call	選取每個呼叫的範圍檢查
db.sql_tokenized.stats.sum_select_scan_per_call	選取每個呼叫的掃描
db.sql_tokenized.stats.sum_sort_merge_passes_per_call	排序每個呼叫的合併路徑
db.sql_tokenized.stats.sum_sort_scan_per_call	排序每個呼叫的掃描
db.sql_tokenized.stats.sum_sort_range_per_call	排序每個呼叫的範圍
db.sql_tokenized.stats.sum_sort_rows_per_call	排序每個呼叫的列數
db.sql_tokenized.stats.sum_rows_affected_per_call	每個呼叫受影響的列數
db.sql_tokenized.stats.sum_rows_examined_per_call	每個呼叫所檢查的列數
db.sql_tokenized.stats.sum_rows_sent_per_call	每個呼叫傳送的列數
db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_call	每個呼叫建立的暫存磁碟資料表
db.sql_tokenized.stats.sum_created_tmp_tables_per_call	每個呼叫建立的暫存資料表
db.sql_tokenized.stats.sum_lock_time_per_call	每個呼叫鎖定時間 (毫秒)

Aurora MySQL 的主要統計資料

以下 SQL 統計資料適用於 Aurora MySQL 資料庫 叢集。

指標	單位
db.sql_tokenized.stats.count_star	呼叫
db.sql_tokenized.stats.sum_timer_wait	等待時間 (毫秒)
db.sql_tokenized.stats.sum_select_full_join	選取完整聯結
db.sql_tokenized.stats.sum_select_range_check	選取範圍檢查
db.sql_tokenized.stats.sum_select_scan	選取掃描
db.sql_tokenized.stats.sum_sort_merge_passes	排序合併傳遞
db.sql_tokenized.stats.sum_sort_scan	排序掃描
db.sql_tokenized.stats.sum_sort_range	排序範圍
db.sql_tokenized.stats.sum_sort_rows	排序資料列
db.sql_tokenized.stats.sum_rows_affected	受影響的資料列
db.sql_tokenized.stats.sum_rows_examined	已檢查的資料列
db.sql_tokenized.stats.sum_rows_sent	已傳送的資料列
db.sql_tokenized.stats.sum_created_tmp_disk_tables	已建立暫存磁碟資料表
db.sql_tokenized.stats.sum_created_tmp_tables	已建立暫存資料表。
db.sql_tokenized.stats.sum_lock_time	鎖定時間 (毫秒)

Aurora PostgreSQL 的 SQL 統計資料

Performance Insights 會針對每一次 SQL 呼叫和執行查詢的每一秒收集 SQL 統計資料。所有 Aurora 引擎只收集摘要層級的統計資料。

您可以在下文中了解 Aurora PostgreSQL、 摘要層級統計資料的相關資訊。

主題

- Aurora PostgreSQL 的摘要統計數字
- Aurora PostgreSQL 的每秒摘要統計資料
- Aurora PostgreSQL 的每次呼叫摘要統計資料
- Aurora PostgreSQL 的主要統計資料

Aurora PostgreSQL 的摘要統計數字

若要檢視 SQL 摘要統計資料，必須載入 `pg_stat_statements` 程式庫。針對與 PostgreSQL 10 相容的 Aurora PostgreSQL 資料庫叢集，依預設會載入此程式庫。針對與 PostgreSQL 9.6 相容的 Aurora PostgreSQL 資料庫叢集，您可以手動啟用此程式庫。若要手動啟用，請在與資料庫執行個體相關聯的資料庫參數群組中，將 `pg_stat_statements` 新增至 `shared_preload_libraries`。然後，重新啟動您的資料庫執行個體。如需更多詳細資訊，請參閱 Amazon Aurora 的參數群組。

> **Note**
>
> 績效詳情只能在 `pg_stat_activity` 中收集未截斷的查詢的統計資料。根據預設，PostgreSQL 資料庫會截斷超過 1,024 位元組的查詢。若要增加查詢大小，請變更與資料庫執行個體相關聯的資料庫參數群組中的 `track_activity_query_size` 參數。當您變更此參數時，需要重新啟動資料庫執行個體。

Aurora PostgreSQL 的每秒摘要統計資料

Aurora PostgreSQL 資料庫執行個體有下列 SQL 摘要統計資料。

指標	單位
db.sql_tokenized.stats.calls_per_sec	每秒呼叫數

指標	單位
db.sql_tokenized.stats.rows_per_sec	每秒列數
db.sql_tokenized.stats.total_time_per_sec	每秒平均作用中執行數 (AAE)
db.sql_tokenized.stats.shared_blks_hit_per_sec	每秒區塊命中數
db.sql_tokenized.stats.shared_blks_read_per_sec	每秒區塊讀取數
db.sql_tokenized.stats.shared_blks_dirtied_per_sec	每秒區塊變動數
db.sql_tokenized.stats.shared_blks_written_per_sec	每秒區塊寫入數
db.sql_tokenized.stats.local_blks_hit_per_sec	每秒本機區塊命中數
db.sql_tokenized.stats.local_blks_read_per_sec	每秒本機區塊讀取數
db.sql_tokenized.stats.local_blks_dirtied_per_sec	每秒本機區塊變動數
db.sql_tokenized.stats.local_blks_written_per_sec	每秒本機區塊寫入數
db.sql_tokenized.stats.temp_blks_written_per_sec	每秒暫時寫入數
db.sql_tokenized.stats.temp_blks_read_per_sec	每秒暫時讀取數
db.sql_tokenized.stats.blk_read_time_per_sec	每秒平均並行讀取數
db.sql_tokenized.stats.blk_write_time_per_sec	每秒平均並行寫入數

Aurora PostgreSQL 的每次呼叫摘要統計資料

下列指標提供 SQL 陳述式的每次呼叫統計數字。

指標	單位
db.sql_tokenized.stats.rows_per_call	每次呼叫列數
db.sql_tokenized.stats.avg_latency_per_call	每次呼叫平均延遲 (毫秒)
db.sql_tokenized.stats.shared_blks_hit_per_call	每次呼叫區塊命中數
db.sql_tokenized.stats.shared_blks_read_per_call	每次呼叫區塊讀取數
db.sql_tokenized.stats.shared_blks_written_per_call	每次呼叫區塊寫入數
db.sql_tokenized.stats.shared_blks_dirtied_per_call	每次呼叫區塊變動數
db.sql_tokenized.stats.local_blks_hit_per_call	每次呼叫本機區塊命中數
db.sql_tokenized.stats.local_blks_read_per_call	每次呼叫本機區塊讀取數
db.sql_tokenized.stats.local_blks_dirtied_per_call	每次呼叫本機區塊變動數
db.sql_tokenized.stats.local_blks_written_per_call	每次呼叫本機區塊寫入數
db.sql_tokenized.stats.temp_blks_written_per_call	每次呼叫暫時區塊寫入數
db.sql_tokenized.stats.temp_blks_read_per_call	每次呼叫暫時區塊讀取數
db.sql_tokenized.stats.blk_read_time_per_call	每次呼叫讀取時間 (毫秒)
db.sql_tokenized.stats.blk_write_time_per_call	每次呼叫寫入時間 (毫秒)

Aurora PostgreSQL 的主要統計資料

下列 SQL 統計資料適用於 Aurora PostgreSQL 資料庫執行個體。

指標	單位
db.sql_tokenized.stats.calls	呼叫
db.sql_tokenized.stats.rows	資料列
db.sql_tokenized.stats.total_time	時間總計 (毫秒)
db.sql_tokenized.stats.shared_blks_hit	區塊命中
db.sql_tokenized.stats.shared_blks_read	區塊讀取
db.sql_tokenized.stats.shared_blks_dirtied	區塊變動數
db.sql_tokenized.stats.shared_blks_written	區塊寫入
db.sql_tokenized.stats.local_blks_hit	本機區塊命中
db.sql_tokenized.stats.local_blks_read	本機區塊讀取
db.sql_tokenized.stats.local_blks_dirtied	本機區塊變動數
db.sql_tokenized.stats.local_blks_written	本機區塊寫入
db.sql_tokenized.stats.temp_blks_written	暫時寫入
db.sql_tokenized.stats.temp_blks_read	暫時讀取
db.sql_tokenized.stats.blk_read_time	平均並行讀取 (毫秒)
db.sql_tokenized.stats.blk_write_time	平均並行寫入 (毫秒)

如需這些指標的詳細資訊，請參閱 PostgreSQL 文件中的 pg_stat_statements。

增強型監控中的作業系統指標

Amazon Aurora 可針對資料庫叢集執行所在的作業系統 (OS) 即時提供指標。Aurora 可將增強型監控的指標提供到您的 Amazon CloudWatch Logs 帳戶中。下表列出使用 Amazon CloudWatch Logs 時可用的作業系統指標。

主題

- Aurora 的作業系統指標

Aurora 的作業系統指標

Group	指標	主控台名稱	描述
General	engine	不適用	資料庫執行個體的資料庫引擎。
	instanceID	不適用	資料庫執行個體識別符。
	instanceResourceID	不適用	AWS 區域唯一資料庫執行個體的不可變識別符，也用作日誌串流識別符。
	numVCPUs	不適用	資料庫執行個體的虛擬 CPU 數量。
	timestamp	不適用	擷取指標的時間。
	uptime	不適用	資料庫執行個體作用中的時間長度。
	version	不適用	作業系統指標的串流 JSON 格式的版本。
cpuUtilization	guest	CPU 訪客	客體程式使用中的 CPU 百分比。
	idle	CPU 閒置	CPU 閒置的百分比。
	irq	CPU IRQ	軟體中斷使用中的 CPU 百分比。
	nice	CPU 不錯	以最低優先順序執行之程式使用中的 CPU 百分比。
	steal	CPU 竊取	其他虛擬機器使用中的 CPU 百分比。
	system	CPU 系統	核心使用中的 CPU 百分比。
	total	CPU 總計	使用中的 CPU 總百分比。此數值包含 nice 值。

Group	指標	主控台名稱	描述
	user	CPU 使用者	使用者程式使用中的 CPU 百分比。
	wait	CPU 等待	等待 I/O 存取時未使用的 CPU 百分比。
diskIO	avgQueueLen	平均佇列大小	在 I/O 裝置的佇列中等待的請求數量。
	avgReqSz	平均請求大小	平均的請求大小，以 KB 為單位。
	await	磁碟 I/O 等待	回應請求時所需的毫秒數，包括佇列時間與服務時間。
	device	不適用	使用中磁碟裝置的識別符。
	readIOsPS	讀取 IO/秒	每秒讀取操作的次數。
	readKb	讀取總數	讀取的 KB 總數。
	readKbPS	讀取 KB/秒	每秒讀取的 KB 總數。
	readLatency	讀取延遲	提交讀取 I/O 要求與完成之間的經過時間，以毫秒為單位。 此指標僅適用於 Amazon Aurora。
	readThroughput	讀取輸送量	要求資料庫叢集所使用的網路輸送量，以每秒位元組為單位。 此指標僅適用於 Amazon Aurora。
	rrqmPS	Rrqms	每秒佇列的合併讀取請求數量。
	tps	TPS	每秒的 I/O 交易數量。
	util	磁碟 I/O 公用程式	發出請求的 CPU 時間百分比。

Group	指標	主控台名稱	描述
	writeIOsPS	寫入 IO/秒	每秒寫入操作的次數。
	writeKb	寫入總計	寫入的 KB 總數。
	writeKbPS	寫入 KB/秒	每秒寫入的 KB 總數。
	writeLatency	寫入延遲	送出寫入 I/O 要求與完成之間的平均經過時間 (毫秒)。 此指標僅適用於 Amazon Aurora。
	writeThroughput	寫入輸送量	資料庫叢集回應所使用的網路輸送量，以每秒位元組為單位。 此指標僅適用於 Amazon Aurora。
	wrqmPS	Wrqs	每秒佇列的合併寫入請求數量。
mountPoint	不適用	指向檔案系統的路徑。	
fileSys	maxFiles	最大節點	檔案相同可建立的最大檔案數量。
	mountPoint	不適用	指向檔案系統的路徑。當這是 時/rdsdbdata* ，代表所有儲存磁碟區的彙總。
	name	不適用	檔案系統的名稱。
	total	檔案系統總數	檔案系統可用的磁碟空間總容量，以 KB 為單位。
	used	已使用的檔案系統	檔案系統中的檔案已使用的磁碟空間容量，以 KB 為單位。
	usedFilePercent	已使用的節點	使用中的可用檔案百分比。

Group	指標	主控台名稱	描述
	usedFiles	已使用比率	檔案系統中的檔案數量。
	usedPercent	已使用的檔案系統	使用中檔案系統磁碟空間的百分比。
loadAverageMinute	fifteen	平均載入時間 15 分鐘	過去 15 分鐘內請求 CPU 時間的程序數量。
	five	平均載入時間 5 分鐘	過去 5 分鐘內請求 CPU 時間的程序數量。
	one	平均載入 1 分鐘	過去 1 分鐘內請求 CPU 時間的程序數量。
memory	active	作用中記憶體	已指派的記憶體數量，以 KB 為單位。
	buffers	緩衝記憶體	在寫入至儲存裝置之前，用於緩衝 I/O 請求的記憶體數量，以 KB 為單位。
	cached	快取記憶體	用來快取檔案系統輸入/輸出的記憶體數量。
	dirty	未清理的記憶體	RAM 之中已修改但尚未寫入至儲存裝置中相關資料區塊的記憶體分頁數量，以 KB 為單位。
	free	可用記憶體	未指派的記憶體數量，以 KB 為單位。
	hugePagesFree	釋出的大內存頁	自由巨型分頁的數量。巨型分頁為 Linux 核心的功能。
	hugePagesRsvd	保留的大內存頁	已遞交的巨型分頁的數量。
	hugePagesSize	大內存頁尺寸	每個巨型分頁的大小，以 KB 為單位。

Group	指標	主控台名稱	描述
	hugePagesSurp	抑制的大內存頁	超過總數的可用剩餘巨型分頁的數量。
	hugePagesTotal	大內存頁總數	大內存頁總數。
	inactive	停用的記憶體	使用頻率最低的記憶體分頁數量，以 KB 為單位。
	mapped	對應記憶體	在程序地址空間內映射的檔案系統內容的總量，以 KB 為單位。
	pageTables	內存頁資料表	分頁表使用的記憶體數量，以 KB 為單位。
	slab	Slab 記憶體	個重複使用的核心資料結構數量，以 KB 為單位。
	total	記憶體總計	記憶體總量，以 KB 為單位。
	writeback	回寫記憶體	RAM 之中仍被寫入至支援儲存裝置的中途分頁數量，以 KB 為單位。
network	interface	不適用	用於資料庫執行個體之網路介面的識別符。
	rx	RX	每秒接收的位元組數量。
	tx	TX	每秒上傳的位元組數量。
processList	cpuUsedPc	CPU %	程序所使用的 CPU 百分比。
	id	不適用	程序的識別符。
	memoryUsedPc	MEM%	程序使用的記憶體百分比。

Group	指標	主控台名稱	描述
	name	不適用	程序的名稱。
	parentID	不適用	程序之父程序的程序識別符。
	rss	RES	配置於程序的 RAM 數量，以 KB 為單位。
	tgid	不適用	執行緒群組識別符，它代表執行緒所屬之程序 ID 號碼。此識別符用於將來自相同程序的執行緒進行分組。
	vss	VIRT	配置於程序的虛擬記憶體數量，以 KB 為單位。
swap	total	交換	可用的交換記憶體數量，以 KB 為單位。
	in	換入	從磁碟交換輸入的記憶體數量，以 KB 為單位。
	out	換出	交換輸出到磁碟的記憶體數量，以 KB 為單位。
	free	自由交換	可用的交換記憶體數量，以 KB 為單位。
	cached	認可的交換	做為快取記憶體使用的 swap 記憶體數量，以 KB 為單位。
tasks	blocked	封鎖的任務	封鎖的任務數量。
	running	執行中的任務	執行中的任務數量。
	sleeping	睡眠的任務	睡眠中的任務數量。
	stopped	任務已停止	已停止的任務數量。
	total	任務總計	任務的總數。
	zombie	任務殭屍	與作用中父任務進行互動的子任務數量。

監控 Amazon Aurora 資料庫叢集中的事件、日誌和串流

當您監控 Amazon Aurora 資料庫和其他 AWS 解決方案時，您的目標是維護下列項目：

- 可靠性
- 可用性
- 效能
- 安全

在 Amazon Aurora 叢集中監控指標 解釋如何使用指標監控叢集。完整的解決方案也必須監控資料庫事件、日誌檔案和活動串流。 AWS 為您提供下列監控工具：

- Amazon EventBridge 為無伺服器事件匯流排服務，可讓您輕鬆將應用程式與來自各種來源的資料互相連線。EventBridge 可從您自己的應用程式、Software-as-a-Service(SaaS) 應用程式 AWS 和服務提供即時資料串流。EventBridge 會將該資料路由傳送至目標，例如 AWS Lambda。如此一來，您可以監控在服務中發生的事件，並建置事件導向的架構。如需詳細資訊，請參閱「Amazon EventBridge 使用者指南」。
- Amazon CloudWatch Logs 提供一種方法來監控、存放和存取來自 Amazon Aurora 執行個體 AWS CloudTrail和其他來源的日誌檔案。Amazon CloudWatch Logs 可監控日誌檔案中的資訊，並在達到特定閾值時通知您。您也可以將日誌資料存檔在高耐用性的儲存空間。如需詳細資訊，請參閱 Amazon CloudWatch Logs 使用者指南。
- AWS CloudTrail 會擷取來自或代表 AWS 帳戶發出的 API 呼叫和相關事件。CloudTrail 會將日誌檔案交付到您指定的 Amazon S3 儲存貯體。您可以識別呼叫的使用者和帳戶 AWS、進行呼叫的來源 IP 地址，以及呼叫的時間。如需詳細資訊，請參閱「AWS CloudTrail 使用者指南」。
- 資料庫活動串流是一項 Amazon Aurora 功能，提供有關您資料庫叢集的近乎即時活動串流。Amazon Aurora 會將活動推送至 Amazon Kinesis 資料串流。系統會自動建立 Kinesis 串流。從 Kinesis，您可以設定 Amazon Data Firehose 和 等 AWS 服務 AWS Lambda ，以取用串流並存放資料。

主題

- 在 Amazon RDS 主控台中檢視日誌、事件和串流
- 監控 Amazon Aurora 事件
- 監控 Amazon Aurora 日誌檔案
- 在 AWS CloudTrail 中監控 Amazon Aurora API 呼叫

- 使用資料庫活動串流來監控 Amazon Aurora
- 使用 Amazon GuardDuty RDS Protection for Amazon Aurora監控威脅

在 Amazon RDS 主控台中檢視日誌、事件和串流

Amazon RDS 與 整合 AWS 服務 ，以在 RDS 主控台中顯示日誌、事件和資料庫活動串流的相關資訊。

Aurora 資料庫叢集的日誌和事件索引標籤會顯示以下資訊：

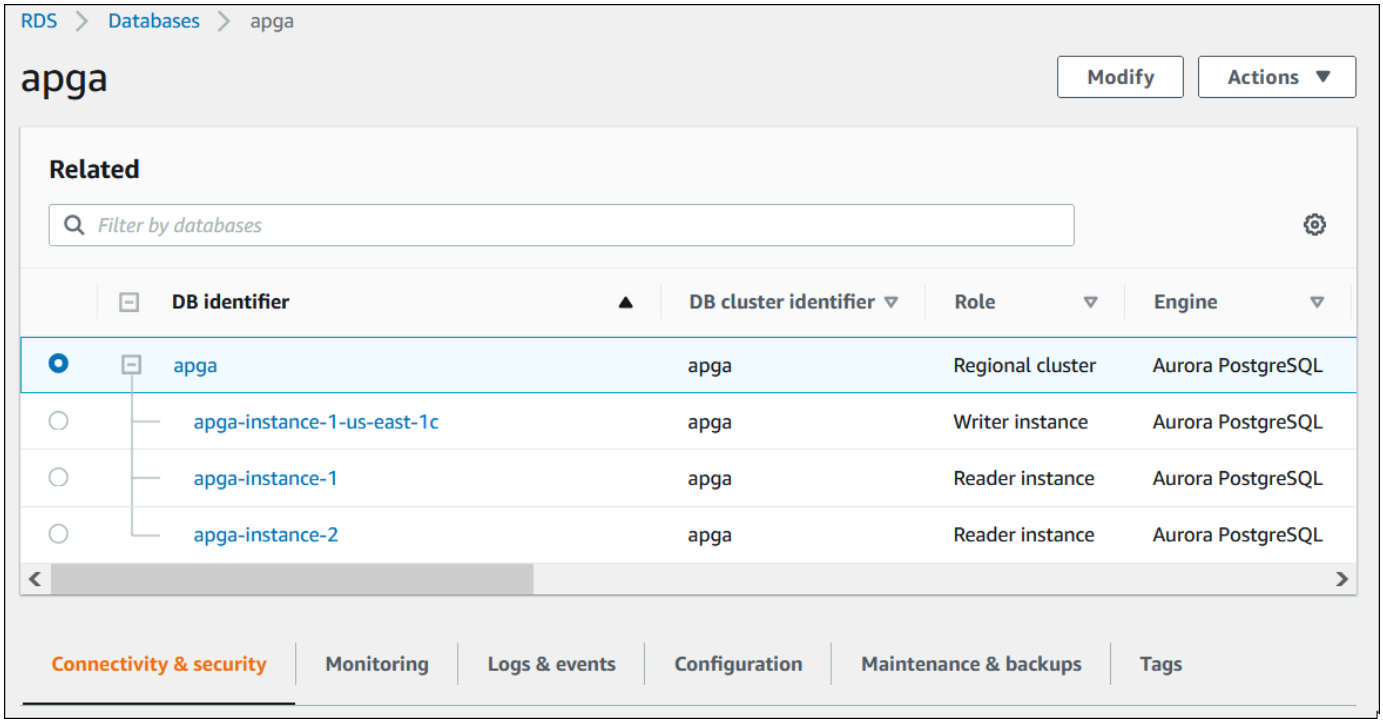
- 自動擴展政策和活動：顯示與 Aurora 自動擴展功能相關的政策和活動。此資訊僅顯示在叢集層級的 Logs & events (日誌和事件) 索引標籤中。
- Amazon CloudWatch alarms (Amazon CloudWatch 警示)：顯示您為 Aurora 叢集中的資料庫執行個體設定的任何指標警示。如果您尚未設定警示，則可以在 RDS 主控台中加以建立。
- Recent events (最近事件)：顯示 Aurora 資料庫執行個體或叢集的事件摘要 (環境變更)。如需詳細資訊，請參閱檢視 Amazon RDS 事件。
- Logs (日誌)：顯示 Aurora 叢集中的資料庫執行個體產生的資料庫日誌檔案。如需詳細資訊，請參閱監控 Amazon Aurora 日誌檔案。

Configuration (組態) 索引標籤會顯示資料庫活動串流的資訊。

在 RDS 主控台中檢視 Aurora 資料庫叢集的日誌、事件和串流

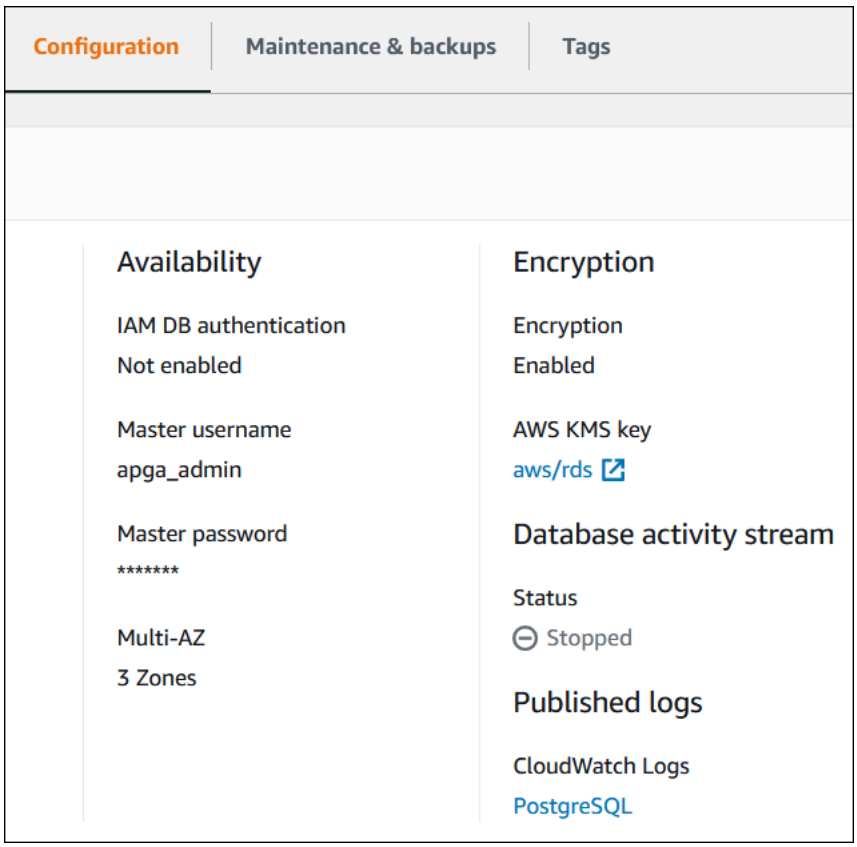
1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您想要監控的 Aurora 資料庫叢集的名稱。

   資料庫頁面隨即出現。下列範例顯示名為 apga 的 Amazon Aurora PostgreSQL 資料庫叢集。



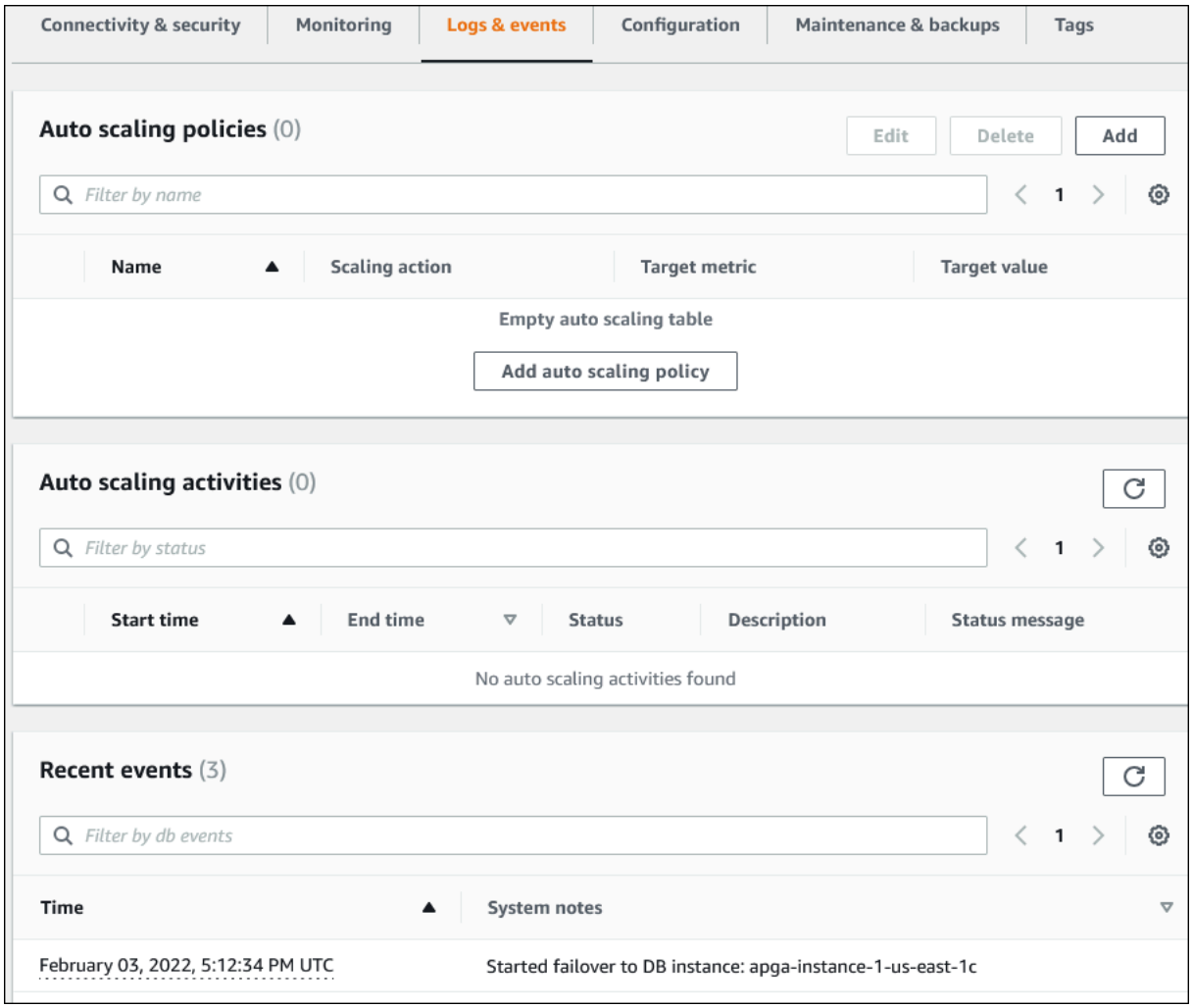
4. 向下捲動並選擇 Configuration (組態)。

下列範例顯示叢集的資料庫活動串流的狀態。



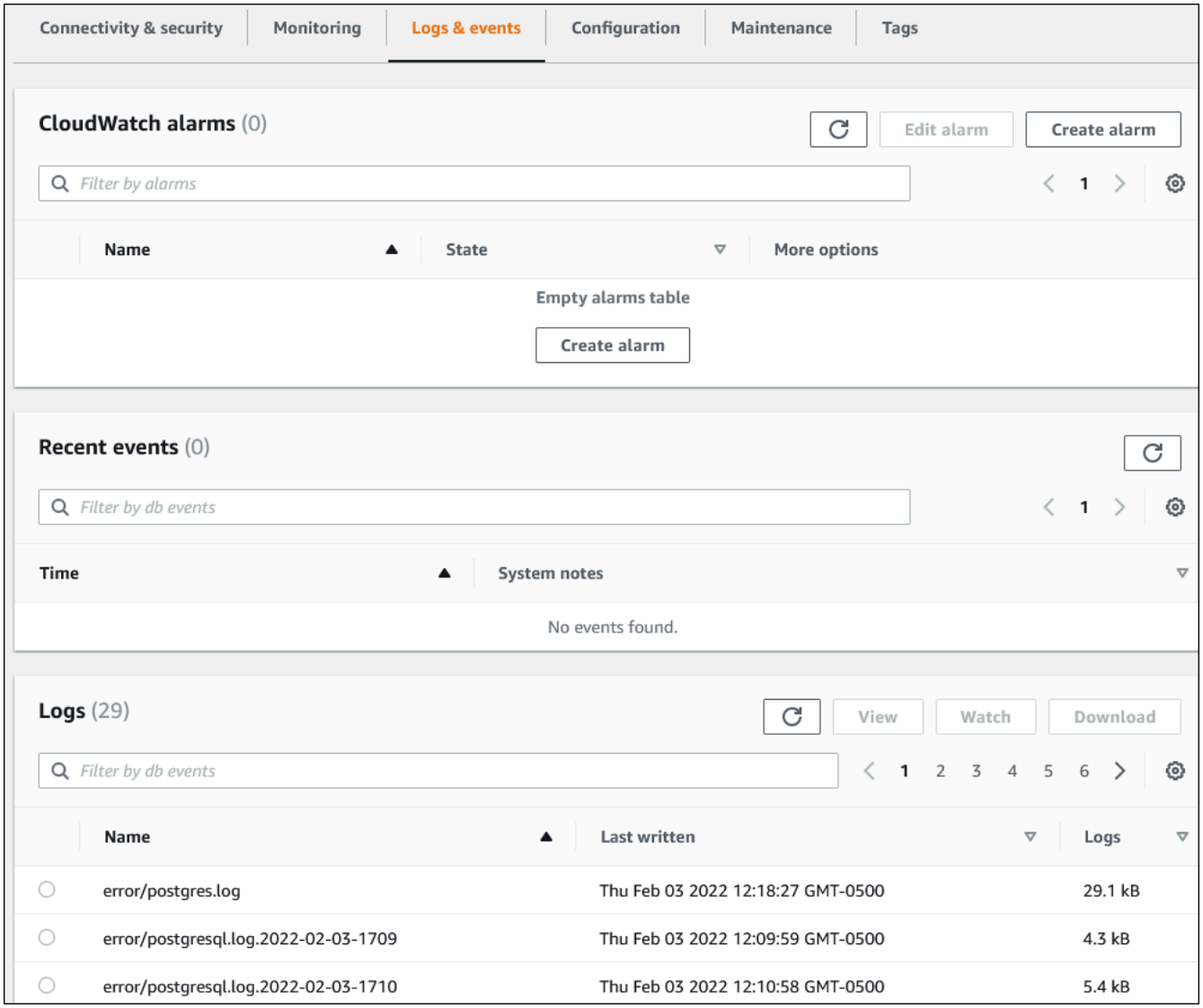
5. 選擇 Logs & events (日誌和事件)。

   此時將顯示「日誌和事件」區段。



6. 在 Aurora 叢集中選擇資料庫執行個體，然後選擇執行個體的 Logs & events (日誌和事件)。

   以下範例顯示資料庫執行個體頁面和資料庫叢集頁面之間的內容有所不同。資料庫執行個體頁面顯示日誌和警示。


Connectivity & security
Monitoring
Logs & events
Configuration
Maintenance
Tags
CloudWatch alarms (0)
Edit alarm
Create alarm
Filter by alarms
1
Name
State
More options
Empty alarms table
Create alarm
Recent events (0)
Filter by db events
1
Time
System notes
No events found.
Logs (29)
View
Watch
Download
Filter by db events
1 2 3 4 5 6
Name
Last written
Logs
error/postgres.log
Thu Feb 03 2022 12:18:27 GMT-0500
29.1 kB
error/postgresql.log.2022-02-03-1709
Thu Feb 03 2022 12:09:59 GMT-0500
4.3 kB
error/postgresql.log.2022-02-03-1710
Thu Feb 03 2022 12:10:58 GMT-0500
5.4 kB

監控 Amazon Aurora 事件

事件表示環境中有變更的事件。這可以是 AWS 環境、SaaS 合作夥伴服務或應用程式，或是自訂應用程式或服務。有關 Aurora 事件的說明，請參閱 Aurora 的 Amazon RDS 事件類別和事件訊息。

主題

- Aurora 的事件概觀
- 檢視 Amazon RDS 事件
- 使用 Amazon RDS 事件通知
- 建立由 Amazon Aurora 事件觸發的規則
- Aurora 的 Amazon RDS 事件類別和事件訊息

Aurora 的事件概觀

RDS 事件表示 Aurora 環境中的變更。例如，當修補資料庫叢集時，Amazon Aurora 會產生一個事件。Amazon Aurora 會近乎即時地將事件傳遞至 EventBridge。

> **Note**
>
> Amazon RDS 會全力發出事件。我們建議您避免編寫會根據通知事件的順序或存在的程式，因為這些事件可能會被移出序列或遺漏。

Amazon RDS 記錄與下列資源相關的事件：

- 資料庫叢集

  如需叢集事件的清單，請參閱 資料庫叢集事件。
- 資料庫執行個體

  如需資料庫執行個體事件清單，請參閱 資料庫執行個體事件。
- 資料庫參數群組

  如需有關資料庫參數群組事件的列表，請參閱 資料庫參數群組事件。
- 資料庫安全群組

  如需有關資料庫安全群組事件的清單，請參閱 資料庫安全群組事件。

- 資料庫叢集快照

  如需資料庫叢集快照事件的清單，請參閱 資料庫叢集快照事件。
- RDS Proxy 事件

  如需 RDS Proxy 事件的清單，請參閱 RDS Proxy 事件。
- 藍/綠部署事件

  如需藍/綠部署事件的清單，請參閱 藍/綠部署事件。

此資訊包含下列項目：

- 事件的日期和時間
- 事件的來源名稱和來源類型
- 與事件相關聯的訊息
- 事件通知包括訊息傳送時的標籤，可能不會反映事件發生時的標籤

檢視 Amazon RDS 事件

您可以擷取資料庫 (主機台會顯示資訊)Amazon Aurora 資源以下的版本資訊：

- 資源名稱
- 資源類型
- 事件的時間
- 活動的訊息摘要

您可以在 AWS 管理主控台 的下列部分存取事件：

- 事件索引標籤，顯示過去 24 小時內發生的事件。
- 資料庫索引標籤中日誌與事件區段中的最近事件資料表，可顯示過去 2 週內發生的事件。

您還可使用 describe-events AWS CLI 命令或 DescribeEvents RDS API 作業來擷取事件。若您是使用 AWS CLI 或 RDS API 檢視事件，則最多可擷取過去 14 天內的事件。

> **Note**
>
> 如果您需要長時間儲存事件，則可將 Amazon RDS 事件傳送至 EventBridge。如需詳細資訊，請參閱建立由 Amazon Aurora 事件觸發的規則

如需有關 Amazon Aurora 事件的說明，請參閱 Aurora 的 Amazon RDS 事件類別和事件訊息。

若要使用 AWS CloudTrail 存取事件的詳細資訊，包括請求參數，請參閱 CloudTrail 事件。

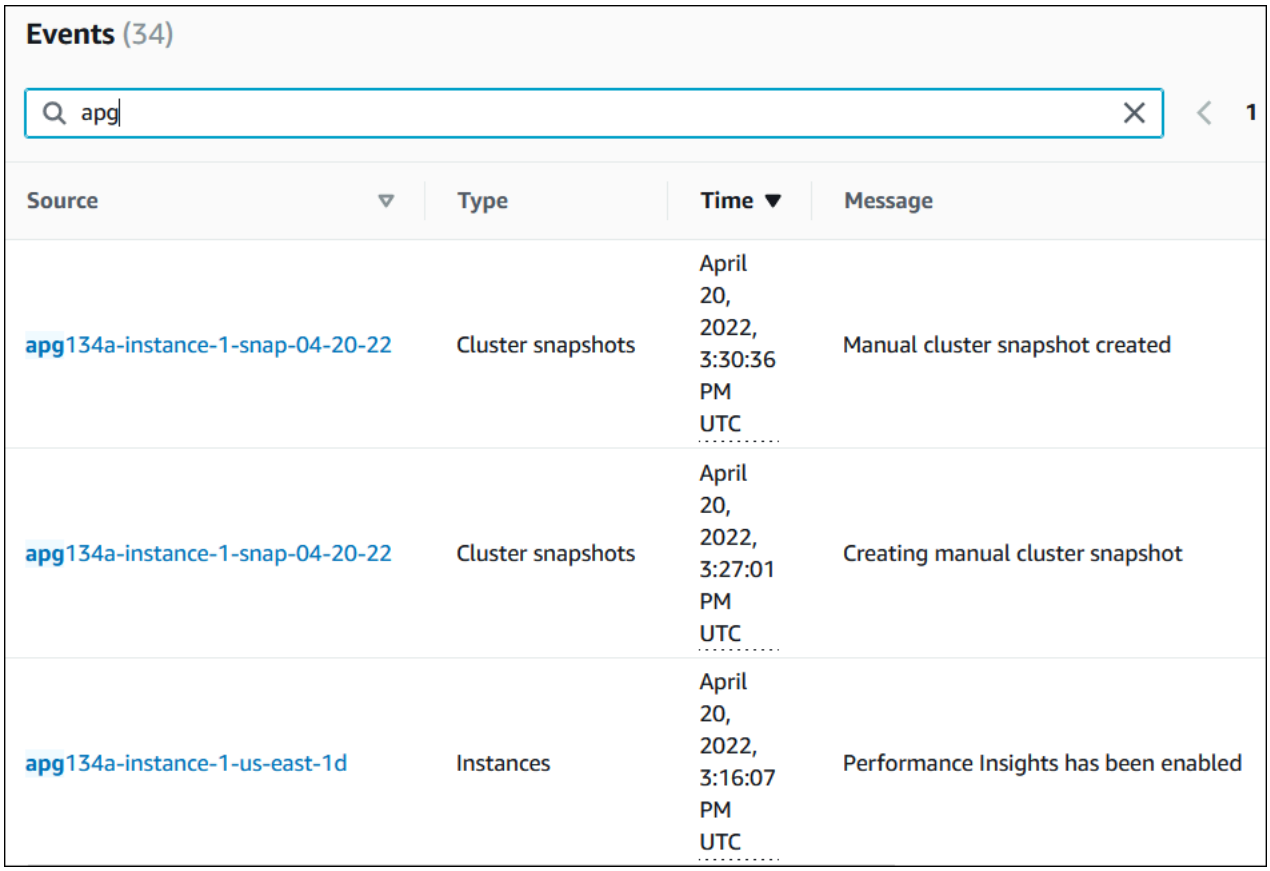
主控台

如要檢視過去 24 小時內的所有 Amazon RDS 事件

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Events (事件)。

   可用的事件隨即會出現在清單中。
3. (選用) 輸入搜尋詞來篩選結果。

下列範例所顯示的事件清單，是依字元 **apg** 篩選的結果。



AWS CLI

如要檢視過去一小時內產生的所有事件，請呼叫不含參數的 describe-events。

```
aws rds describe-events
```

下列範例輸出顯示資料庫叢集執行個體已開始復原。

```
{
    "Events": [
        {
            "EventCategories": [
                "recovery"
            ],
            "SourceType": "db-instance",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mycluster-instance-1",
            "Date": "2022-04-20T15:02:38.416Z",
```

```
            "Message": "Recovery of the DB instance has started. Recovery time will
 vary with the amount of data to be recovered.",
            "SourceIdentifier": "mycluster-instance-1"
        }, ...
```

如要檢視過去 10080 分鐘 (7 天) 內的所有 Amazon RDS 事件，請呼叫 describe-events AWS CLI 命令，並將 `--duration` 參數設定為 `10080`。

```
aws rds describe-events --duration 10080
```

下列範例顯示資料庫執行個體 *test-instance* 在指定時間範圍內的事件。

```
aws rds describe-events \
    --source-identifier test-instance \
    --source-type db-instance \
    --start-time 2022-03-13T22:00Z \
    --end-time 2022-03-13T23:59Z
```

下列範例輸出顯示備份的狀態。

```
{
    "Events": [
        {
            "SourceType": "db-instance",
            "SourceIdentifier": "test-instance",
            "EventCategories": [
                "backup"
            ],
            "Message": "Backing up DB instance",
            "Date": "2022-03-13T23:09:23.983Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
        },
        {
            "SourceType": "db-instance",
            "SourceIdentifier": "test-instance",
            "EventCategories": [
                "backup"
            ],
            "Message": "Finished DB Instance backup",
            "Date": "2022-03-13T23:15:13.049Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
        }
```

```
    ]
}
```

API

如果要檢視過去 14 天內的所有 Amazon RDS 執行個體事件，請呼叫 DescribeEvents RDS API 操作，並將 `Duration` 參數設定為 `20160`。

使用 Amazon RDS 事件通知

Amazon RDS 使用 Amazon Simple Notification Service (Amazon SNS) 在 Amazon RDS 事件發生時提供通知。這些通知可以採用任何 Amazon SNS 在 AWS 區域中支援的通知形式，例如電子郵件、文字訊息或呼叫 HTTP 端點。

主題

- Amazon RDS 事件通知概觀
- 授予將通知發佈至 Amazon SNS 主題的許可
- 訂閱 Amazon RDS 事件通知
- Amazon RDS 事件通知標籤與屬性
- 列出 Amazon RDS 事件通知訂閱
- 修改 Amazon RDS 事件通知訂閱
- 將來源識別碼新增至 Amazon RDS 事件通知訂閱
- 將來源識別碼從 Amazon RDS 事件通知訂閱中移除
- 列出 Amazon RDS 事件通知類別
- 刪除 Amazon RDS 事件通知訂閱

Amazon RDS 事件通知概觀

Amazon RDS 將事件分成幾個類別供您訂閱，讓您在該類別的事件發生時收到通知。

主題

- 符合事件訂閱資格的 RDS 資源
- 訂閱 Amazon RDS 事件通知的基本程序
- RDS 事件通知的傳遞
- Amazon RDS 事件通知的帳單
- 使用 Amazon EventBridge 的 Aurora 事件範例

符合事件訂閱資格的 RDS 資源

對 Amazon Aurora 來說，事件發生於資料庫叢集和資料庫執行個體層級。您可以針對下列資源訂閱事件類別：

- 資料庫執行個體
- 資料庫叢集
- 資料庫叢集快照
- DB parameter group (資料庫參數群組)
- 資料庫安全群組
- RDS Proxy
- 自訂引擎版本

例如，如果您訂閱指定資料庫執行個體的備份類別，當發生會影響資料庫執行個體的備份相關事件時，您將會收到通知。如果您訂閱資料庫執行個體的組態變更類別，當資料庫執行個體變更時，您將會收到通知。當事件通知訂閱變更時，您也會收到通知。

您可能會想要建立數個不同的訂閱。例如，您可能會建立一個訂閱以接收所有資料庫執行個體的所有事件通知，並建立另一個訂閱以僅包含資料庫執行個體子集的重要事件。對於第二個訂閱，請在篩選條件中指定一或多個資料庫執行個體。

訂閱 Amazon RDS 事件通知的基本程序

訂閱 Amazon RDS 事件通知的程序如下：

1. 您可以使用 Amazon RDS 主控台 AWS CLI或 API 建立 Amazon RDS 事件通知訂閱。

   Amazon RDS 使用 Amazon SNS 主題的 ARN 來識別每個訂閱。Amazon RDS 主控台會在您建立訂閱時為您建立 ARN。使用 Amazon SNS 主控台 AWS CLI、 或 Amazon SNS API 建立 ARN。
2. Amazon RDS 會將核准電子郵件或 SMS 訊息傳送至您在訂閱時提交的地址。
3. 您可以選擇收到的通知中的連結，以確認訂閱。
4. Amazon RDS 主控台會以您的訂閱狀態更新 My Event Subscriptions (我的事件訂閱) 區段。
5. Amazon RDS 會將通知傳送到您在建立訂閱時提供的地址。

若要了解使用 Amazon SNS 時的 Identity and Access Management，請參閱《Amazon Simple Notification Service 開發人員指南》中的 Amazon SNS 中的 Identity and Access Management。

您可以使用 AWS Lambda 來處理資料庫執行個體的事件通知。如需詳細資訊，請參閱《 AWS Lambda 開發人員指南》中的搭配使用 AWS Lambda 與 Amazon RDS。

RDS 事件通知的傳遞

Amazon RDS 會將通知傳送到您在建立訂閱時提供的地址。通知可以包含訊息屬性，其會提供有關訊息的結構化中繼資料。如需訊息屬性的詳細資訊，請參閱 Aurora 的 Amazon RDS 事件類別和事件訊息

事件通知的傳送可能需要 5 分鐘。

> **Important**
>
> Amazon RDS 不保證事件串流中傳送事件的順序。事件順序可能會改變。

當 Amazon SNS 傳送通知至已訂閱之 HTTP 或 HTTPS 端點時，已傳送至端點的 POST 訊息具有包含 JSON 文件的訊息內文。如需詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的 Amazon SNS 訊息與 JSON 格式。

您可以將 SNS 設定為以簡訊通知您。如需詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的手機簡訊 (SMS)。

若要關閉通知而不刪除訂閱項目，請在 Amazon RDS 主控台中，為 Enabled (啟用) 選擇 No (否)。或者，您可以使用 AWS CLI 或 Amazon RDS API `false` 將 `Enabled` 參數設定為 。

Amazon RDS 事件通知的帳單

Amazon RDS 事件通知的帳單是透過 Amazon SNS 傳送。使用事件通知需要支付 Amazon SNS 費用。如需 Amazon SNS 帳單的詳細資訊，請參閱 Amazon Simple Notification Service 定價。

使用 Amazon EventBridge 的 Aurora 事件範例

下列範例以 JSON 格式說明不同類型的 Aurora 事件。如需演示如何擷取和檢視 JSON 格式事件的教學課程，請參閱 教學課程：使用 Amazon EventBridge 來記錄資料庫執行個體狀態變更。

主題

- 資料庫叢集事件範例
- 資料庫參數群組事件的範例
- 資料庫叢集快照事件範例

資料庫叢集事件範例

以下是 JSON 格式的資料庫叢集事件範例。此事件顯示名為 my-db-cluster 的叢集已修補。事件 ID 是 RDS-EVENT-0173。

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Cluster Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster"
  ],
  "detail": {
    "EventCategories": [
      "notification"
    ],
    "SourceType": "CLUSTER",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Database cluster has been patched",
    "SourceIdentifier": "my-db-cluster",
    "EventID": "RDS-EVENT-0173"
  }
}
```

資料庫參數群組事件的範例

以下是 JSON 格式的資料庫參數群組事件範例。事件顯示參數群組 time_zone 的參數 my-db-param-group 已更新。事件 ID 是 RDS-EVENT-0037。

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Parameter Group Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
```

```
    "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group"
  ],
  "detail": {
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PARAM",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Updated parameter time_zone to UTC with apply method immediate",
    "SourceIdentifier": "my-db-param-group",
    "EventID": "RDS-EVENT-0037"
  }
}
```

資料庫叢集快照事件範例

以下是 JSON 格式的資料庫叢集快照事件範例。此事件會顯示建立名為 my-db-cluster-snapshot 的快照。事件 ID 是 RDS-EVENT-0074。

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Cluster Snapshot Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-cluster-snapshot"
  ],
  "detail": {
    "EventCategories": [
      "backup"
    ],
    "SourceType": "CLUSTER_SNAPSHOT",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-
cluster-snapshot",
    "Date": "2018-10-06T12:26:13.882Z",
    "SourceIdentifier": "my-db-cluster-snapshot",
    "Message": "Creating manual cluster snapshot",
    "EventID": "RDS-EVENT-0074"
  }
```

```
}
```

授予將通知發佈至 Amazon SNS 主題的許可

若要授予 Amazon RDS 將通知發佈至 Amazon Simple Notification Service (Amazon SNS) 主題的許可，請將 AWS Identity and Access Management(IAM) 政策附加至目的地主題。有關許可的詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的 Amazon Simple Notification Service 存取控制的範例案例。

依預設，Amazon SNS 主題具有一項政策，可允許同一帳戶內的所有 Amazon RDS 資源向其發佈通知。您可以連接自訂政策以允許跨帳戶通知，或限制對特定資源的存取權。

以下是您連接至目的地 Amazon SNS 主題之 IAM 政策的範例。它將主題限制為名稱符合指定字首的資料庫執行個體。若要使用此政策，請指定下列值：

- `Resource` – Amazon SNS 主題的 Amazon Resource Name (ARN)
- `SourceARN` – 您的 RDS 資源 ARN
- `SourceAccount` – 您的 AWS 帳戶ID

若要查看資源類型清單及其 ARN，請參閱服務授權參考中的 Amazon RDS 定義的資源。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "events.rds.amazonaws.com"
      },
      "Action": [
        "sns:Publish"
      ],
      "Resource": "arn:aws:sns:us-east-1:123456789012:topic_name",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:prefix-*"
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
```

```
      }
    }
  ]
}
```

訂閱 Amazon RDS 事件通知

建立訂閱最簡單的方式是使用 RDS 主控台。如果您選擇使用 CLI 或 API 建立事件通知訂閱，您必須建立 Amazon Simple Notification Service 主題，並使用 Amazon SNS 主控台或 Amazon SNS API 訂閱該主題。您也將必須保留主題的 Amazon Resource Name (ARN)，因為在提交 CLI 命令或 API 操作時會用到它。如需建立和訂閱 SNS 主題的詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的 Amazon SNS 入門。

您可以指定想要收到通知的來源類型，以及觸發事件的 Amazon RDS 來源。

Source type (來源類型)

來源類型。例如：Source type (來源類型) 可能是 Instances (執行個體)。您必須選擇來源類型。

要包含的**##**

正在產生事件的 Amazon RDS 資源。例如，您可以選擇 Select specific instances (選取特定執行個體) 然後選擇 myDBInstane1。

下表說明了指定或不指定要包含的**##**時的結果。

要包含的資源	描述	範例
指定	RDS 只會通知您指定資源的所有資料庫執行個體事件。	如果您的 Source type (來源類型) 是 Instances (執行個體)，您的資源是 myDBInstance1，RDS 只會通知您有關 myDBInstance1。
未指定	您會收到所有 Amazon RDS 資源中指定來源類型事件的通知。	如果您的 Source type (來源類型) 是 Instances (執行個體)，RDS 會通知您帳戶中所有與執行個體相關的事件。

根據預設，Amazon SNS 主題訂閱者會接收發佈到主題的每個訊息。若要僅接收一部分的訊息，訂閱者必須將篩選政策指派給主題訂閱。如需有關 SNS 訊息篩選的詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的 Amazon SNS 訊息篩選

主控台

訂閱 RDS 事件通知

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Event subscriptions (事件訂閱)。
3. 在 Event subscriptions (事件訂閱) 窗格中，選擇 Create event subscription (建立事件訂閱)。
4. 輸入您的訂閱詳細資訊，如下所示：

   a. 在 Name (名稱) 中，輸入事件通知訂閱的名稱。

   b. 對於 Send notification to: (傳送通知給：)，執行以下其中一項：

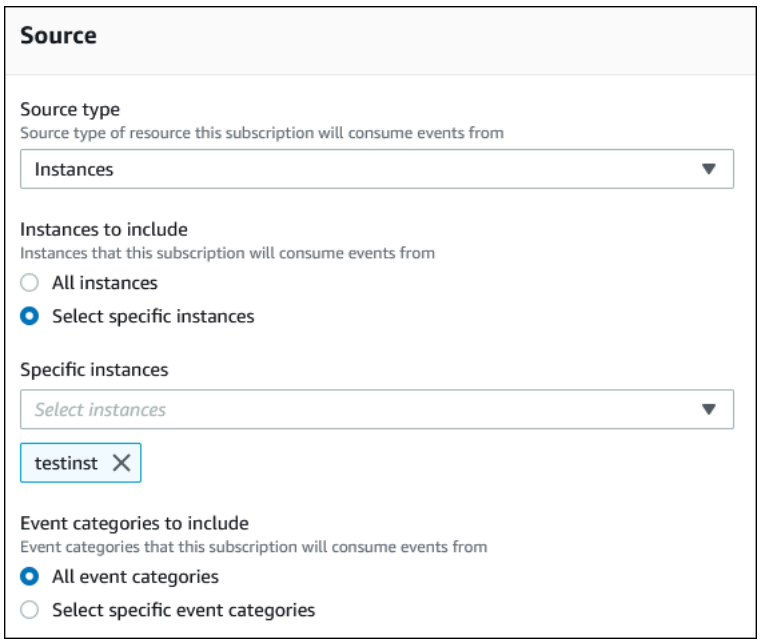
      - 選擇 New email topic (新的電子郵件主題)。輸入電子郵件主題的名稱和收件者清單。建議您將事件訂閱設定為與主要帳戶聯絡人相同的電子郵件地址。建議、服務事件和個人健康訊息會使用不同的通道傳送。訂閱相同電子郵件地址可確保所有郵件都合併在一個位置。
      - 選擇 Amazon Resource Name (ARN)。然後在 Amazon SNS 主題中選擇現有 Amazon SNS ARN。

        如果您想要使用已為伺服器端加密 (SSE) 啟用的主題，請授予 Amazon RDS 存取 AWS KMS key 所需的許可。如需詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的啟用 AWS 服務的事件來源與加密主題之間的相容性。

   c. 在 Source type (來源類型) 中選擇來源類型。例如，選擇 Cluster (叢集) 或 Cluster snapshots (叢集快照)。

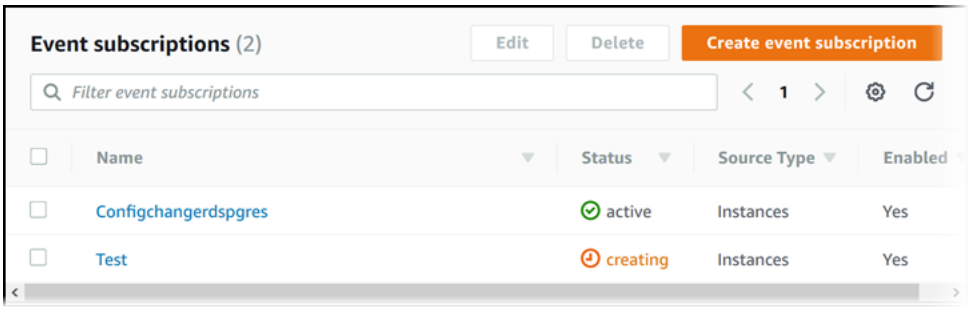
   d. 選擇您要接收通知的事件類型和資源。

      下列範例設定名為 `testinst` 之資料庫執行個體的事件通知。



e. 選擇 Create (建立)。

Amazon RDS 主控台顯示正在建立訂閱。



AWS CLI

使用 AWS CLI `create-event-subscription` 命令來訂閱 RDS 事件通知。包含下列必要參數：

- `--subscription-name`
- `--sns-topic-arn`

Example

對於 Linux、macOS 或 Unix：

```
aws rds create-event-subscription \
    --subscription-name myeventsubscription \
```

```
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS \
    --enabled
```

在 Windows 中：

```
aws rds create-event-subscription ^
    --subscription-name myeventsubscription ^
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS ^
    --enabled
```

API

如要訂閱 Amazon RDS 事件通知，請呼叫 Amazon RDS API 函數 CreateEventSubscription。包含下列必要參數：

- SubscriptionName
- SnsTopicArn

Amazon RDS 事件通知標籤與屬性

當 Amazon RDS 將事件通知傳送至 Amazon Simple Notification Service (SNS) 或 Amazon EventBridge 時，通知會包含訊息屬性和事件標籤。RDS 會隨訊息單獨傳送訊息屬性，而事件標籤位於訊息內文中。使用訊息屬性和 Amazon RDS 標籤，將中繼資料新增至資源。您可以使用關於資料庫執行個體、Aurora 叢集等的表示法。如需標記 Amazon RDSAurora 資源的詳細資訊，請參閱 標記 Amazon Aurora 和 Amazon RDS 資源。

根據預設，Amazon SNS 和 Amazon EventBridge 會接收傳送給它們的每則訊息。SNS 和 EventBridge 可以篩選訊息並將通知傳送到偏好的通訊模式，例如電子郵件、簡訊或呼叫 HTTP 端點。

> **Note**
>
> 透過電子郵件或簡訊傳送的通知不會有事件標籤。

下表顯示傳送到主題訂閱者的 RDS 事件的訊息屬性。

Amazon RDS 事件屬性	Description
EventID	RDS 事件訊息的識別碼，例如 RDS-EVENT-0006。
資源	發出事件之資源的 ARN 識別符，例如 `arn:aws:rds:ap-southeast-2:123456789012:db:database-1` 。

RDS 標籤提供受服務事件影響之資源的相關資料。當通知傳送至 SNS 或 EventBridge 時，RDS 會在訊息內文中新增標籤的目前狀態。

如需有關 SNS 訊息篩選屬性的詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的 Amazon SNS 訊息篩選

如需為 EventBridge 篩選事件標籤的詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的在 Amazon EventBridge 中用於事件模式的比較運算子。

如需篩選 SNS 承載型標籤的詳細資訊，請參閱 Amazon SNS 承載型訊息篩選簡介

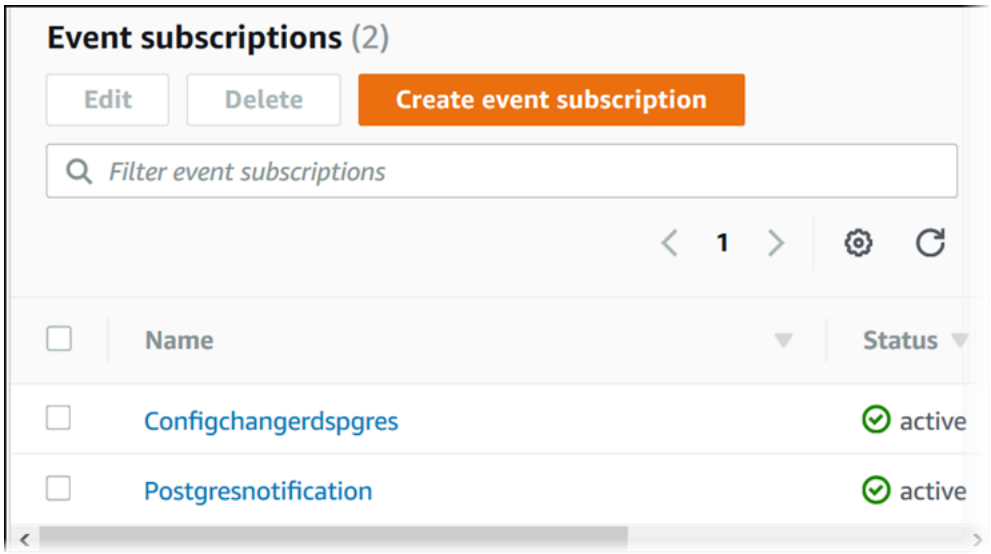
列出 Amazon RDS 事件通知訂閱

您可以列出您目前的 Amazon RDS 事件通知訂閱。

主控台

列出您目前的 Amazon RDS 事件通知訂閱

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Event subscriptions (事件訂閱)。Event subscriptions (事件訂閱) 窗格顯示所有事件通知訂閱。



AWS CLI

使用 AWS CLI `describe-event-subscriptions` 命令來列出您目前的 Amazon RDS 事件通知訂閱。

Example

以下範例說明所有事件訂閱。

```
aws rds describe-event-subscriptions
```

以下範例說明 `myfirsteventsubscription`。

```
aws rds describe-event-subscriptions --subscription-name myfirsteventsubscription
```

API

呼叫 Amazon RDS API DescribeEventSubscriptions 動作來列出您目前的 Amazon RDS 事件通知訂閱。

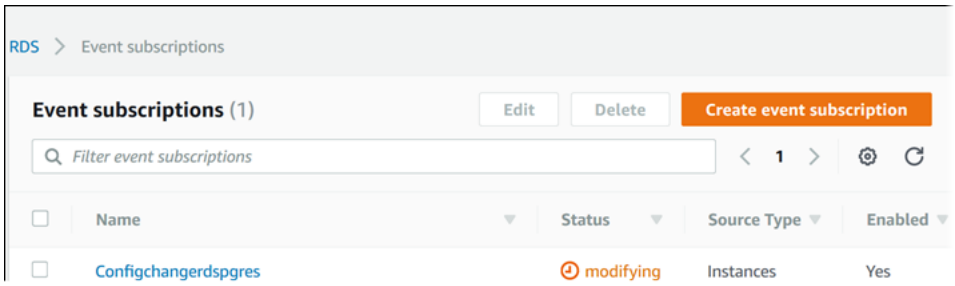
修改 Amazon RDS 事件通知訂閱

在您建立訂閱之後，您可以變更訂閱名稱、來源識別碼、類別或主題 ARN。

主控台

修改 Amazon RDS 事件通知訂閱

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Event subscriptions (事件訂閱)。
3. 在 Event subscriptions (事件訂閱) 窗格中，選擇您要修改的訂閱，然後選擇 Edit (編輯)。
4. 在 Target (目標) 或 Source (來源) 區段中，對訂閱進行變更。
5. 選擇編輯。Amazon RDS 主控台顯示正在修改訂閱。



AWS CLI

使用 AWS CLI `modify-event-subscription` 命令來修改 Amazon RDS 事件通知訂閱。包含下列必要參數：

- `--subscription-name`

Example

下列範例程式碼啟動 `myeventsubscription`。

對於 Linux、macOS 或 Unix：

```
aws rds modify-event-subscription \
    --subscription-name myeventsubscription \
    --enabled
```

在 Windows 中：

```
aws rds modify-event-subscription ^
    --subscription-name myeventsubscription ^
    --enabled
```

API

若要修改 Amazon RDS 事件，請呼叫 Amazon RDS API 操作 ModifyEventSubscription。包含下列必要參數：

- SubscriptionName

將來源識別碼新增至 Amazon RDS 事件通知訂閱

您可以將來源識別碼 (產生事件的 Amazon RDS 來源) 新增至現有的訂閱。

主控台

您可以在修改訂閱時，使用 Amazon RDS 主控台選取或取消選取來源識別碼，以輕鬆地新增或移除來源識別碼。如需更多詳細資訊，請參閱 修改 Amazon RDS 事件通知訂閱。

AWS CLI

若要將來源識別符新增至 Amazon RDS 事件通知訂閱，請使用 AWS CLI `add-source-identifier-to-subscription`命令。包含下列必要參數：

- `--subscription-name`
- `--source-identifier`

Example

下列範例將來源識別碼 `mysqldb` 新增至 `myrdseventsubscription` 訂閱。

對於 Linux、macOS 或 Unix：

```
aws rds add-source-identifier-to-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

在 Windows 中：

```
aws rds add-source-identifier-to-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

呼叫 Amazon RDS API `AddSourceIdentifierToSubscription` 將來源識別碼新增至 Amazon RDS 事件通知訂閱。包含下列必要參數：

- `SubscriptionName`
- `SourceIdentifier`

將來源識別碼從 Amazon RDS 事件通知訂閱中移除

如果您不再希望收到某個來源的事件通知，您可以從訂閱中移除該來源識別碼 (產生事件的 Amazon RDS 來源)。

主控台

您可以在修改訂閱時，使用 Amazon RDS 主控台選取或取消選取來源識別碼，以輕鬆地新增或移除來源識別碼。如需更多詳細資訊，請參閱 修改 Amazon RDS 事件通知訂閱。

AWS CLI

若要從 Amazon RDS 事件通知訂閱中移除來源識別符，請使用 AWS CLI `remove-source-identifier-from-subscription`命令。包含下列必要參數：

- `--subscription-name`
- `--source-identifier`

Example

下列範例將來源識別碼 `mysqldb` 從 `myrdseventsubscription` 訂閱中移除。

對於 Linux、macOS 或 Unix：

```
aws rds remove-source-identifier-from-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

在 Windows 中：

```
aws rds remove-source-identifier-from-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

使用 Amazon RDS API `RemoveSourceIdentifierFromSubscription` 命令將來源識別碼從 Amazon RDS 事件通知訂閱中移除。包含下列必要參數：

- `SubscriptionName`

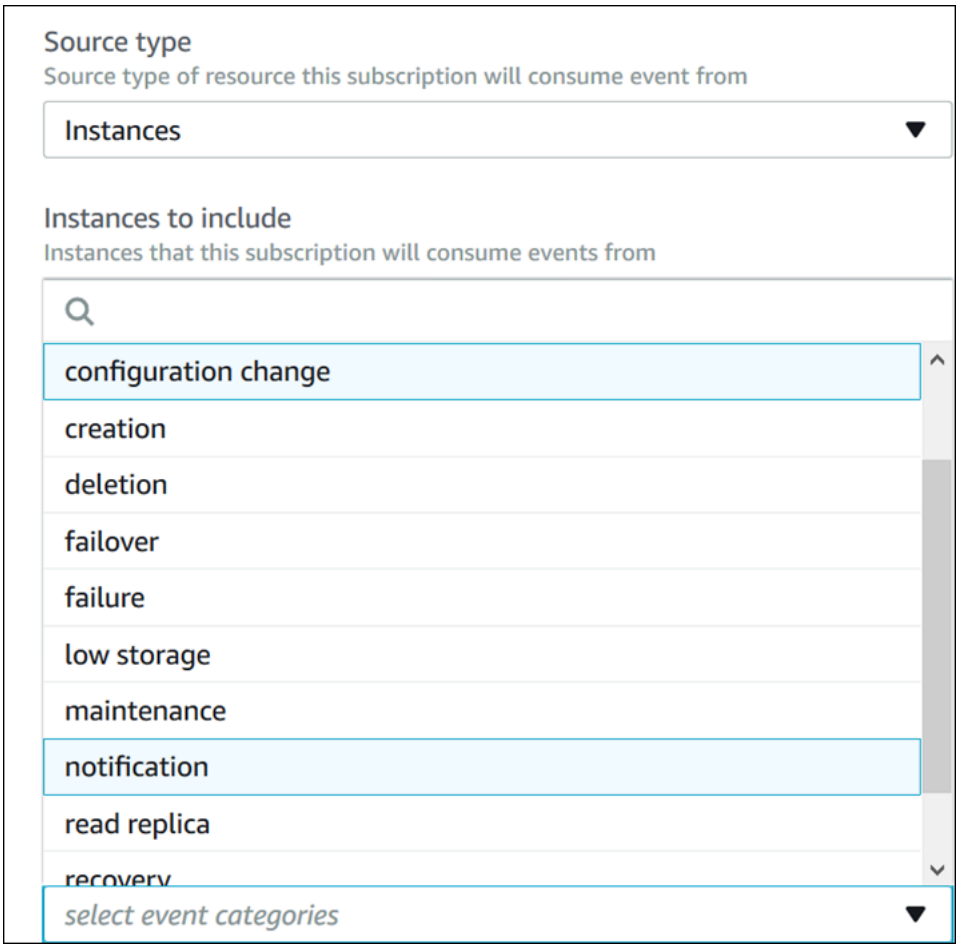
- SourceIdentifier

列出 Amazon RDS 事件通知類別

一種資源類型的所有事件會群組為各種類別。若要查看可用的類別清單，請使用下列程序。

主控台

當您建立或修改事件通知訂閱時，事件類別會顯示在 Amazon RDS 主控台。如需更多詳細資訊，請參閱 修改 Amazon RDS 事件通知訂閱。



AWS CLI

使用 AWS CLI `describe-event-categories` 命令來列出 Amazon RDS 事件通知訂閱類別。此命令沒有要求的參數。

Example

```
aws rds describe-event-categories
```

API

使用 Amazon RDS API `DescribeEventCategories` 命令來列出 Amazon RDS 事件通知訂閱類別。此命令沒有要求的參數。

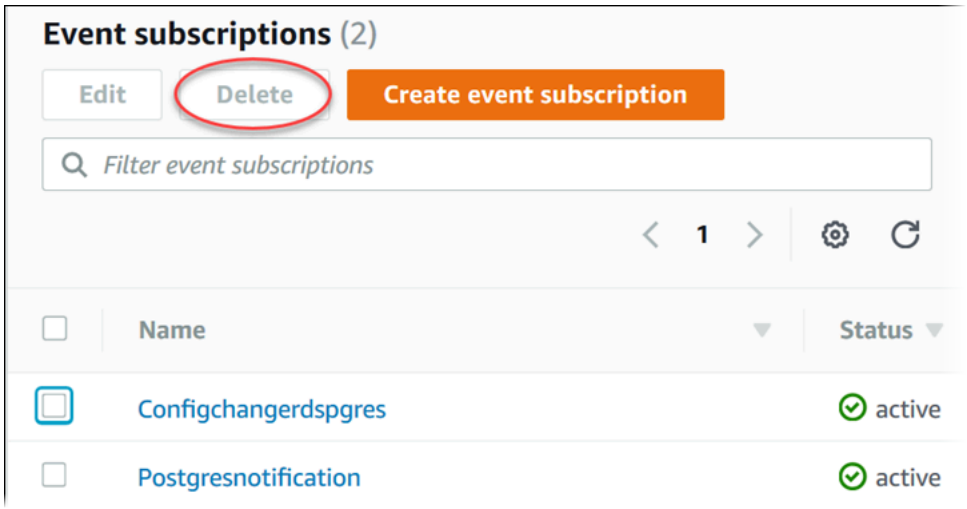
刪除 Amazon RDS 事件通知訂閱

您可以刪除不再需要的訂閱。該主題的所有訂閱者將不會再收到該訂閱指定的事件通知。

主控台

刪除 Amazon RDS 事件通知訂閱

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 DB Event Subscriptions (資料庫事件訂閱)。
3. 在 My DB Event Subscriptions (我的資料庫事件訂閱) 窗格中，選擇您要刪除的訂閱。
4. 選擇 刪除。
5. Amazon RDS 主控台顯示正在刪除訂閱。



AWS CLI

使用 AWS CLI `delete-event-subscription` 命令來刪除 Amazon RDS 事件通知訂閱。包含下列必要參數：

- `--subscription-name`

Example

以下範例刪除訂閱 `myrdssubscription`。

```
aws rds delete-event-subscription --subscription-name myrdssubscription
```

API

使用 RDS API DeleteEventSubscription 命令來刪除 Amazon RDS 事件通知訂閱。包含下列必要參數：

- SubscriptionName

建立由 Amazon Aurora 事件觸發的規則

使用 Amazon EventBridge，您可以自動化 AWS 服務並回應系統事件，例如應用程式可用性問題或資源變更。

主題

- 教學課程：使用 Amazon EventBridge 來記錄資料庫執行個體狀態變更

教學課程：使用 Amazon EventBridge 來記錄資料庫執行個體狀態變更

在本教學課程中，您會建立 AWS Lambda 函數，記錄 執行個體的狀態變更。然後，您建立一個規則，在現有 RDS 資料庫執行個體的狀態變更時執行該函數。本教學假設您擁有可以暫時關閉的小型執行中測試執行個體。

> **Important**
>
> 請勿在執行生產資料庫執行個體上執行本教學課程。

主題

- 步驟 1：建立 AWS Lambda 函數
- 步驟 2：建立規則
- 步驟 3：測試規則

步驟 1：建立 AWS Lambda 函數

建立 Lambda 函數以記錄狀態變更事件。當您在建立規則時指定此函數。

建立 Lambda 函數

1. 在 https：//https://console.aws.amazon.com/lambda/ 開啟 AWS Lambda 主控台。
2. 如果您是第一次使用 Lambda，將會看到歡迎頁面。選擇 Get Started Now (立即開始)。否則，請選擇 Create function (建立函數)。
3. 選擇 Author from scratch (從頭開始撰寫)。
4. 在 Create function (建立函數) 頁面上，執行下列動作：

    a. 輸入 Lambda 函數的名稱和描述。例如，將函數命名為 **`RDSInstanceStateChange`**。

b. 在 Runtime (執行時間) 中，選取 Node.js 16x。

c. 對於 Architecture (架構)，選擇 x86_64。

d. 對於 Execution role (執行角色)，執行下列任何一項：

- 選擇 Create a new role with basic Lambda permissions (建立具備基本 Lambda 許可的新角色)。
- 針對 Existing role (現有角色)，選擇 Use an existing role (使用現有的角色)。選擇您想使用的角色。

e. 選擇 Create function (建立函式)。

5. 在 RDSInstanceStateChange 頁面，執行下列動作：

a. 在 Code source (程式碼來源) 中，選取 index.js。

b. 在 index.js 窗格中，刪除現有的程式碼。

c. 輸入下列程式碼：

```
console.log('Loading function');

exports.handler = async (event, context) => {
    console.log('Received event:', JSON.stringify(event));
};
```

d. 選擇 Deploy (部署)。

步驟 2：建立規則

建立規則，在您啟動 Amazon RDS 執行個體時，執行您的 Lambda 函數。

若要建立 EventBridge 規則

1. 在 https://console.aws.amazon.com/events/ 開啟 Amazon EventBridge 主控台。
2. 在導覽窗格中，選擇規則。
3. 選擇建立規則。
4. 輸入規則的名稱和描述。例如，輸入 **`RDSInstanceStateChangeRule`**。
5. 選擇 Rule with an event pattern (具有事件模式的規則)，然後選擇 Next (下一步)。
6. 在事件來源欄位中，選擇 AWS 事件或 EventBridge 合作夥伴事件。
7. 向下捲動到 Event pattern (事件模式) 區段中。

8. 在 Event source (事件來源)，選擇 AWS 服務。
9. 對於 AWS Service (服務)，選擇 Relational Database Service (RDS) (關聯式資料庫服務)。
10. 對於 Event type (事件類型)，選擇 RDS DB Instance Event (RDS 資料庫執行個體事件)。
11. 保留預設事件模式。然後選擇下一步。
12. 在目標類型欄位中，選擇 AWS 服務。
13. 對於 Select a target (選取目標)，選擇 Lambda function (Lambda 函數)。
14. 針對 Function (函數)，選擇您建立的 Lambda 函數。然後選擇下一步。
15. 在 Configure tags (設定標籤) 中，選擇 Next (下一步)。
16. 檢閱規則中的步驟。然後，選擇 Create role (建立角色)。

步驟 3：測試規則

若要測試您的規則，請關閉 RDS 資料庫執行個體。等待幾分鐘讓執行個體關閉，驗證您的 Lambda 函數是否被叫用。

停用資料庫執行個體以測試您的規則

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 停止 RDS 資料庫執行個體。
3. 在 https://console.aws.amazon.com/events/ 開啟 Amazon EventBridge 主控台。
4. 在導覽窗格中，選擇 Rules (規則)，然後選擇您建立的規則名稱。
5. 在規則詳細資料中，選擇監控。

   您會被重新導向至 Amazon CloudWatch 主控台。如果您未重新導向，請按一下在 CloudWatch 中檢視指標。
6. 在 All metrics (所有指標) 中，選擇您建立的規則名稱。

   該圖表應指示已叫用的規則。
7. 在導覽窗格中，選擇 Log groups (日誌群組)。
8. 選擇您的 Lambda 函數的日誌群組名稱 (/aws/lambda/*function-name*)。
9. 選擇日誌串流名稱以檢視函數為您啟動的執行個體所提供的資料。您應該會看到類似以下接收的事件：

```
{
    "version": "0",
```

```
    "id": "12a345b6-78c9-01d2-34e5-123f4ghi5j6k",
    "detail-type": "RDS DB Instance Event",
    "source": "aws.rds",
    "account": "111111111111",
    "time": "2021-03-19T19:34:09Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:rds:us-east-1:111111111111:db:testdb"
    ],
    "detail": {
        "EventCategories": [
            "notification"
        ],
        "SourceType": "DB_INSTANCE",
        "SourceArn": "arn:aws:rds:us-east-1:111111111111:db:testdb",
        "Date": "2021-03-19T19:34:09.293Z",
        "Message": "DB instance stopped",
        "SourceIdentifier": "testdb",
        "EventID": "RDS-EVENT-0087"
    }
}
```


如需更多 JSON 格式的 RDS 事件範例，請參閱 Aurora 的事件概觀。

10. (選用) 完成後，您可以開啟 Amazon RDS 主控台並啟用您停止的執行個體。

Aurora 的 Amazon RDS 事件類別和事件訊息

Amazon RDS 在您可以使用 Amazon RDS 主控台 AWS CLI或 API 訂閱的類別中產生大量事件。

主題

- 資料庫叢集事件
- 資料庫叢集快照事件
- 資料庫執行個體事件
- 資料庫參數群組事件
- 資料庫安全群組事件
- 資料庫碎片群組事件
- RDS Proxy 事件
- 藍/綠部署事件

資料庫叢集事件

下表顯示當資料庫叢集為來源類型時的事件類別和事件清單。

> **Note**
>
> 資料庫叢集事件類型中不存在 Aurora Serverless 的事件類別。Aurora 無伺服器事件範圍從 RDS-EVENT-0141 到 RDS-EVENT-0149。

類別	RDS 事件 ID	訊息	備註
組態變更	RDS-EVENT-0016	重設主要憑證。	無
組態變更	RDS-EVENT-0179	資料庫叢集上的資料庫活動串流已啟動。	如需更多詳細資訊，請參閱使用資料庫活動串流來監控 Amazon Aurora。
設定更改	RDS-EVENT-0180	資料庫叢集上的資料庫活動串流已停止。	如需更多詳細資訊，請參閱使用資料庫活動串流來監控 Amazon Aurora。

類別	RDS 事件 ID	訊息	備註
建立	RDS-EVENT-0170	已建立資料庫叢集。	無
刪除	RDS-EVENT-0171	資料庫叢集已刪除。	無
容錯移轉	RDS-EVENT-0069	叢集容錯移轉失敗，請檢查叢集執行個體的運作狀態，然後再試一次。	無
容錯移轉	RDS-EVENT-0070	再次提升上一個主要項目：*name*。	無
容錯移轉	RDS-EVENT-0071	已完成容錯移轉至資料庫執行個體：*name*。	無
容錯移轉	RDS-EVENT-0072	已啟動相同的 AZ 容錯移轉至資料庫執行個體：*name*。	無
容錯移轉	RDS-EVENT-0073	已啟動跨 AZ 容錯移轉至資料庫執行個體：*name*。	無
失敗	RDS-EVENT-0083	Amazon RDS 無法建立憑證，為您的資料庫叢集 *name* 存取您的 Amazon S3 儲存貯體。這是因為您的帳戶中未正確設定 S3 快照擷取 IAM 角色，或找不到指定的 Amazon S3 儲存貯體。如需進一步的詳細資訊，請參閱 Amazon RDS 文件中的疑難排解一節。	如需詳細資訊，請參閱使用 Percona XtraBackup 和 Amazon S3 從 MySQL 進行實體遷移 。
失敗	RDS-EVENT-0143	由於以下原因，資料庫叢集無法從 *units* 擴展至 *units*：*reason*。	Aurora Serverless 資料庫叢集擴展失敗。

類別	RDS 事件 ID	訊息	備註
失敗	RDS-EVENT-0354	由於資源不相容，您無法建立資料庫叢集。*##*。	該*##*包含有關故障的詳細資料。
失敗	RDS-EVENT-0355	由於資源限制不足，無法建立資料庫叢集。*##*。	該*##*包含有關故障的詳細資料。
失敗	RDS-EVENT-0387	無法在資料庫叢集*##*中分割資料庫執行個體以進行修補。	資料庫叢集中資料庫執行個體的作業系統升級失敗。
全域容錯移轉	RDS-EVENT-0181	已啟動區域 *name* 中的全球轉換至資料庫叢集 *name*。	此為轉換操作的事件 (舊稱為「受管計畫容錯移轉」)。 因為其他作業正在資料庫叢集上執行，所以此程序可能延遲。
全域容錯移轉	RDS-EVENT-0182	區域 *name* 中的舊主要資料庫叢集 *name* 成功關閉。	此為轉換操作的事件 (舊稱為「受管計畫容錯移轉」)。 全域資料庫中的舊主要執行個體不接受寫入。所有磁碟區都會同步化。
全域容錯移轉	RDS-EVENT-0183	等待跨全域叢集成員的資料同步。目前落後於主要資料庫叢集：*reason*。	此為轉換操作的事件 (舊稱為「受管計畫容錯移轉」)。 複寫延遲發生在全域資料庫容錯移轉的同步階段期間。
全域容錯移轉	RDS-EVENT-0184	區域 *name* 中的新主要資料庫叢集 *name* 成功提升。	此為轉換操作的事件 (舊稱為「受管計畫容錯移轉」)。 全域資料庫的磁碟區拓撲會以新的主要磁碟區重新建立。

類別	RDS 事件 ID	訊息	備註
全域容錯移轉	RDS-EVENT-0185	已完成區域 *name* 中的全球轉換至資料庫叢集 *name*。	此為轉換操作的事件 (舊稱為「受管計畫容錯移轉」)。 已在主要資料庫叢集上完成全球資料庫轉換。容錯移轉完成後，複本可能需要很長時間才能上線。
全域容錯移轉	RDS-EVENT-0186	已取消區域 *name* 中的全球轉換至資料庫叢集 *name*。	此為轉換操作的事件 (舊稱為「受管計畫容錯移轉」)。
全域容錯移轉	RDS-EVENT-0187	區域 *name* 中的全球轉換至資料庫叢集 *name* 失敗。	此為轉換操作的事件 (舊稱為「受管計畫容錯移轉」)。
全域容錯移轉	RDS-EVENT-0238	已完成區域 *name* 中的全球容錯移轉至資料庫叢集 *name*。	無
全域容錯移轉	RDS-EVENT-0239	區域 *name* 中的全域容錯移轉至資料庫叢集 *name* 失敗。	無
全域容錯移轉	RDS-EVENT-0240	全球容錯移轉後，已開始重新同步處理區域 *name* 中資料庫叢集 *name* 的成員。	無
全域容錯移轉	RDS-EVENT-0241	全球容錯移轉後，已完成重新同步處理區域 *name* 中資料庫叢集 *name* 的成員。	無
全域容錯移轉	RDS-EVENT-0397	Aurora 已完成變更全域寫入器端點解析的 DNS 名稱。	無

類別	RDS 事件 ID	訊息	備註
全域容錯移轉	RDS-EVENT-0423	等待與目標資料庫叢集的資料同步。目前目標資料庫叢集落後於主要資料庫叢集：*%s*。	此為轉換操作的事件 (舊稱為「受管計畫容錯移轉」)。 複寫延遲發生在全域資料庫容錯移轉的同步階段期間。
維護	RDS-EVENT-0156	資料庫叢集有可用的資料庫引擎次要版本升級。	無
維護	RDS-EVENT-0173	資料庫叢集引擎版本已升級。	已完成資料庫叢集的修補。
維護	RDS-EVENT-0174	資料庫叢集處於無法升級的狀態。	無
維護	RDS-EVENT-0176	資料庫叢集引擎主要版本已升級。	無
維護	RDS-EVENT-0177	資料庫叢集升級正在進行中。	無
維護	RDS-EVENT-0286	資料庫叢集引擎 *version_number* 版本升級已開始。叢集保持線上狀態。	無
維護	RDS-EVENT-0287	偵測到作業系統升級需求。	無
維護	RDS-EVENT-0288	叢集作業系統升級開始中。	無
維護	RDS-EVENT-0289	叢集作業系統升級已完成。	無
維護	RDS-EVENT-0290	已修補資料庫叢集：來源版本 *version_number* => *new_version_number* 。	無

類別	RDS 事件 ID	訊息	備註
維護	RDS-EVENT-0363	升級準備進行中：*cluster_name*	資料庫叢集的升級前檢查已開始。
維護	RDS-EVENT-0388	在資料庫叢集##的分割區##/##中啟動資料庫執行個體的離線修補程式：*partition_n* 。	啟動資料庫叢集中資料庫執行個體的作業系統升級。
維護	RDS-EVENT-0389	我們無法升級您的資料庫叢集作業系統。您可以等待下一個維護時段，也可以手動升級資料庫叢集作業系統。	無
維護	RDS-EVENT-0424	資料庫叢集##正在執行版本##，此版本高於全域叢集的目標升級版本##。我們不建議在高於全域叢集的版本上使用次要叢集，因為它可能會在容錯移轉或轉換期間造成問題。請考慮升級您的全域叢集以符合條件。	無
notification	RDS-EVENT-0076	無法從 *name* 遷移至 *name*。原因：*reason*。	遷移至 Aurora 資料庫叢集失敗。
則通知	RDS-EVENT-0077	無法將 *name.name* 轉換為 InnoDB。原因：*reason*。	在遷移至 Aurora 資料庫叢集時，嘗試從來源資料庫將資料表轉換為 InnoDB 失敗。

類別	RDS 事件 ID	訊息	備註
則通知	RDS-EVENT-0085	無法升級資料庫叢集 *name*，因為執行個體 *name* 的狀態為 *name*。解決問題或刪除執行個體，然後再試一次。	嘗試修補 Aurora 資料庫叢集時發生錯誤。檢查您的執行個體狀態，解決問題，然後再試一次。如需更多詳細資訊，請參閱 維持 為 Amazon Aurora 資料庫叢集。
則通知	RDS-EVENT-0141	由於以下原因，正在將資料庫叢集從 *units* 擴展至 *units*：*reason*。	Aurora Serverless 資料庫叢集已起始擴展。
則通知	RDS-EVENT-0142	資料庫叢集已從 *units* 擴展至 *units*。	Aurora Serverless 資料庫叢集已完成擴展。
則通知	RDS-EVENT-0144	資料庫叢集正在暫停。	已針對 Aurora Serverless 資料庫叢集啟動自動暫停。
notification	RDS-EVENT-0145	資料庫叢集已暫停。	Aurora Serverless 資料庫叢集已暫停。
notification	RDS-EVENT-0146	已取消暫停資料庫叢集。	已取消暫停 Aurora Serverless 資料庫叢集。
notification	RDS-EVENT-0147	正在恢復資料庫叢集。	已啟動 Aurora Serverless 資料庫叢集的恢復操作。
notification	RDS-EVENT-0148	已恢復資料庫叢集。	已完成 Aurora Serverless 資料庫叢集的恢復操作。
notification	RDS-EVENT-0149	資料庫叢集已從 *units* 擴展至 *units*，但由於以下原因，擴展並不順暢：*reason*。	使用 Aurora Serverless 資料庫叢集強制選項完成無縫擴展。可能已依要求中斷連線。
則通知	RDS-EVENT-0150	資料庫叢集已停止。	無

類別	RDS 事件 ID	訊息	備註
notification	RDS-EVENT-0151	資料庫叢集已啟動。	無
notification	RDS-EVENT-0152	資料庫叢集停止失敗。	無
notification	RDS-EVENT-0153	資料庫叢集正在啟動，因為它超過允許停止的時間上限。	無
notification	RDS-EVENT-0172	叢集已從 *name* 重新命名為 *name*。	無
notification	RDS-EVENT-0234	匯出任務失敗。	資料庫叢集匯出任務失敗。
notification	RDS-EVENT-0235	已取消匯出任務。	已取消資料庫叢集匯出任務。
notification	RDS-EVENT-0236	已完成匯出任務。	已完成資料庫叢集匯出任務。
notification	RDS-EVENT-0386	資料庫叢集##中的資料庫執行個體已分割：*list_of_partitions* 。資料庫叢集已上線。	資料庫叢集中資料庫執行個體的作業系統升級成功。
notification	RDS-EVENT-0426	由於交易閒置或長時間執行，RDS 無法將資料庫叢集##設定為複寫來源。等待交易完成或取消它們，然後再試一次。	無

資料庫叢集快照事件

下表顯示當資料庫叢集快照為來源類型時的事件類別和事件清單。

Category	RDS 事件 ID	訊息	備註
備份	RDS-EVENT-0074	正在建立手動叢集快照。	無
備份	RDS-EVENT-0075	已建立手動叢集快照。	無
notification	RDS-EVENT-0162	叢集快照匯出任務失敗。	無
notification	RDS-EVENT-0163	已取消叢集快照匯出任務。	無
notification	RDS-EVENT-0164	已完成叢集快照匯出任務。	無
備份	RDS-EVENT-0168	建立自動叢集快照。	無
備份	RDS-EVENT-0169	已建立自動叢集快照。	無
失敗	RDS-EVENT-0489	匯出任務##：由於##，資料表##的資料表擷取失敗。	更快速的匯出不支援此事件。
notification	RDS-EVENT-0491	匯出任務##：資料表##將以較慢的單執行緒模式匯出。請預期匯出完成會延遲。	更快速的匯出不支援此事件。
notification	RDS-EVENT-0492	匯出任務##：資料表##正在進行中，匯出的大小為##GB。	此事件只會針對需要超過 1 小時才能完成的資料表傳送。 更快速的匯出不支援此事件。
notification	RDS-EVENT-0493	匯出任務##處於##階段。	無
		匯出任務##：匯出進度為*number*%，###資料表（進行中）和##資料表（已完成）。已完成資料表的總大小為## GB。	更快速的匯出不支援此事件。

資料庫執行個體事件

下表顯示當資料庫執行個體為來源類型時的事件類別和事件清單。

Category	RDS 事件 ID	訊息	備註
可用性	RDS-EVENT-0004	資料庫執行個體關機。	無
可用性	RDS-EVENT-0006	資料庫執行個體已重新啟動。	無
可用性	RDS-EVENT-0022	重新啟動 mysql 時發生錯誤：*message*。	重新啟動 Aurora MySQL 或 RDS for MariaDB 時發生錯誤。
可用性	RDS-EVENT-0419	Amazon RDS 無法存取資料庫執行個體##的 KMS 加密金鑰。此資料庫將進入無法存取的狀態。如需進一步的詳細資訊，請參閱 Amazon RDS 文件中的疑難排解一節。	無
恢復	RDS-EVENT-0131	實際恢復時段比您指定的目標恢復時段還小。請考量減少目標恢復時段中的時數。	如需更多有關恢復的資訊，請參閱 恢復 Aurora 資料庫叢集。
恢復	RDS-EVENT-0132	實際恢復時段與目標恢復時段相同。	無
設定更改	RDS-EVENT-0011	已更新為使用 DBParameterGroup ##。	無
組態變更	RDS-EVENT-0012	套用修改至資料庫執行個體類別。	無
組態變更	RDS-EVENT-0014	將修改套用至資料庫執行個體類別已完成。	無

Category	RDS 事件 ID	訊息	備註
組態變更	RDS-EVENT-0017	將修改套用至配置的儲存已完成。	無
組態變更	RDS-EVENT-0025	套用修改以轉換為多可用區域資料庫執行個體已完成。	無
組態變更	RDS-EVENT-0029	套用修改以轉換為標準 (單一可用區域) 資料庫執行個體已完成。	無
組態變更	RDS-EVENT-0033	有 *number* 個使用者符合主要使用者名稱；僅重設未繫結至特定主機的使用者。	無
組態變更	RDS-EVENT-0067	無法重設您的密碼。錯誤資訊：*message*。	無
組態變更	RDS-EVENT-0078	監控間隔已變更為 *number*。	增強型監控組態已變更。
組態變更	RDS-EVENT-0092	已完成資料庫參數群組的更新。	無
建立	RDS-EVENT-0005	已建立資料庫執行個體。	無
刪除	RDS-EVENT-0003	已刪除資料庫執行個體。	無
失敗	RDS-EVENT-0035	資料庫執行個體進入 *state*。*message*。	資料庫執行個體有無效的參數。例如，如果此執行個體類別與記憶體相關的參數設定過高造成資料庫叢集無法啟動，則您的動作將為修改記憶體參數並重新啟動資料庫執行個體。

Category	RDS 事件 ID	訊息	備註
失敗	RDS-EVENT-0036	資料庫執行個體處於 *state*。*message*。	資料庫執行個體位於不相容的網路中。部分指定的子網路 ID 無效或不存在。
失敗	RDS-EVENT-0079	Amazon RDS 無法建立用於增強型監控的憑證，且此功能已停用。這可能是因為 rds-monitoring-role 不存在，且未在您的帳戶中正確設定所致。如需進一步的詳細資訊，請參閱 Amazon RDS 文件中的疑難排解一節。	若沒有增強型監控 IAM 角色，無法啟用增強型監控。如需建立 IAM 角色的相關資訊，請參閱 為 Amazon RDS 增強型監控建立 IAM 角色。
失敗	RDS-EVENT-0080	Amazon RDS 無法在您的執行個體上設定增強型監控：*name*，且此功能已停用。這可能是因為 rds-monit oring-role 不存在，且未在您的帳戶中正確設定所致。如需進一步的詳細資訊，請參閱 Amazon RDS 文件中的疑難排解一節。	因為組態變更期間發生錯誤，所以增強型監控已停用。可能是未正確設定增強型監控 IAM 角色。如需建立增強型監控 IAM 角色的相關資訊，請參閱 為 Amazon RDS 增強型監控建立 IAM 角色。
失敗	RDS-EVENT-0082	Amazon RDS 無法建立憑證，為您的資料庫執行個體 *name* 存取您的 Amazon S3 儲存貯體。這是因為您的帳戶中未正確設定 S3 快照擷取 IAM 角色，或找不到指定的 Amazon S3 儲存貯體。如需進一步的詳細資訊，請參閱 Amazon RDS 文件中的疑難排解一節。	Aurora 無法從 Amazon S3 儲存貯體複製備份資料。可能是 Aurora 存取 Amazon S3 儲存貯體的權限設定不正確。如需更多詳細資訊，請參閱使用 Percona XtraBackup 和 Amazon S3 從 MySQL 進行實體遷移 。

Category	RDS 事件 ID	訊息	備註
失敗	RDS-EVENT-0353	由於資源限制不足，無法建立資料庫執行個體。*##*。	該*##*包含有關故障的詳細資料。
失敗	RDS-EVENT-0418	Amazon RDS 無法存取資料庫執行個體*##*的 KMS 加密金鑰。這可能是由於金鑰遭停用或 Amazon RDS 無法存取該金鑰。如果繼續執行，資料庫將進入無法存取的狀態。如需進一步的詳細資訊，請參閱 Amazon RDS 文件中的疑難排解一節。	無
失敗	RDS-EVENT-0420	Amazon RDS 現在可成功存取資料庫執行個體*##*的 KMS 加密金鑰。	無
低儲存	RDS-EVENT-0007	配置的儲存已用盡。請配置額外的儲存來解決此問題。	已經使用為資料庫執行個體分配的儲存。若要解決此問題，為此資料庫執行個體分配額外儲存空間。如需詳細資訊，請參閱 RDS 常見問答集。您可以使用 Free Storage Space (可用儲存空間) 指標來監控資料庫執行個體的儲存空間。
低儲存	RDS-EVENT-0089	資料庫執行個體 *name* 的可用儲存容量低至所佈建儲存體的 *percentage* [佈建的儲存體：*size*，可用的儲存體：*size*]。您可能想要增加已佈建的儲存體來解決此問題。	資料庫執行個體已消耗超過其分配儲存容量的 90%。您可以使用 Free Storage Space (可用儲存空間) 指標來監控資料庫執行個體的儲存空間。

Category	RDS 事件 ID	訊息	備註
低儲存	RDS-EVENT-0227	Aurora 叢集的儲存體極低，只剩下 *amount* TB。請採取措施來減少叢集上的儲存負載。	Aurora 儲存子系統的空間不足。
維護	RDS-EVENT-0026	正在將離線修補程式套用至資料庫執行個體。	資料庫執行個體的離線維護正在進行。資料庫執行個體目前無法使用。
維護	RDS-EVENT-0027	已完成將離線修補程式套用至資料庫執行個體。	資料庫執行個體的離線維護已完成。資料庫執行個體現在可用。
維護	RDS-EVENT-0047	已修補資料庫執行個體。	無
維護	RDS-EVENT-0155	資料庫執行個體有可用的資料庫引擎次要版本升級。	無
維護	RDS-EVENT-0178	資料庫執行個體升級正在進行中。	無
維護	RDS-EVENT-0422	由於待定維護動作，RDS 將取代資料庫執行個體##的主機。	無
notification	RDS-EVENT-0044	*message*	這是操作員發出的通知。如需詳細資訊，請參閱事件訊息。
則通知	RDS-EVENT-0048	延遲資料庫引擎升級，因為此執行個體具有需要先升級的僅供讀取複本。	資料庫執行個體的修補作業已延遲。
則通知	RDS-EVENT-0087	資料庫執行個體已停止。	無
notification	RDS-EVENT-0088	資料庫執行個體已啟動。	無

Category	RDS 事件 ID	訊息	備註
notification	RDS-EVENT-0365	時區檔案已更新。重新啟動 RDS 執行個體，讓變更生效。	無
通知、無伺服器	RDS-EVENT-0370	已起始資料庫執行個體的暫停。	已啟動暫停閒置 Aurora Serverless v2 資料庫執行個體的新嘗試。
通知、無伺服器	RDS-EVENT-0371	已取消暫停資料庫執行個體。	由於工作負載，嘗試暫停閒置 Aurora Serverless v2 資料庫執行個體失敗。
通知、無伺服器	RDS-EVENT-0372	成功暫停資料庫執行個體。	Aurora Serverless v2 資料庫執行個體已暫停。
通知、無伺服器	RDS-EVENT-0373	已起始資料庫執行個體的繼續。	由於新的工作負載或管理或維護活動，Aurora Serverless v2 資料庫執行個體開始繼續。
通知、無伺服器	RDS-EVENT-0374	已成功繼續資料庫執行個體。	Aurora Serverless v2 資料庫執行個體已繼續。
notification	RDS-EVENT-0385	叢集拓撲已更新。	資料庫執行個體的資料庫叢集有 DNS 變更。這包括新增或刪除新的資料庫執行個體，或發生容錯移轉時。
通知、全域資料庫	RDS-EVENT-0390	嘗試封鎖區域 *region_id* 中資料庫叢集 *cluster_id* 的寫入成功。	Aurora 開始封鎖儲存體層級的寫入，以準備 Aurora 全域資料庫的轉換或容錯移轉。

Category	RDS 事件 ID	訊息	備註
通知、全域資料庫	RDS-EVENT-0391	嘗試封鎖區域 *region_id* 中資料庫叢集 *cluster_id* 的寫入逾時。	Aurora 無法封鎖儲存體層級的寫入，以準備 Aurora 全域資料庫的轉換或容錯移轉。轉換或容錯移轉會繼續進行，但您可能需要從原始主要叢集的快照復原最近寫入的資料。
僅供讀取複本	RDS-EVENT-0045	複寫已停止。	當複寫期間發生錯誤時，會顯示此訊息。若要判斷錯誤類型，請參閱針對 MySQL 僅供讀取複本問題進行疑難排解。
僅供讀取複本	RDS-EVENT-0046	僅供讀取複本已恢復複寫。	此訊息會在您首次建立僅供讀取複本時出現，或做為確認複寫正常運作的監控訊息顯示。如果此訊息出現在 RDS-EVENT-0045 通知之後，則在錯誤或複寫停止之後已恢復複寫。
僅供讀取複本	RDS-EVENT-0057	已終止複寫串流。	無
復原	RDS-EVENT-0020	資料庫執行個體的復原已啟動。復原時間依據恢復的資料量而有不同。	無
復原	RDS-EVENT-0021	資料庫執行個體的復原已完成。	無
復原	RDS-EVENT-0023	緊急快照請求：*message*。	已請求手動備份，但 Amazon RDS 目前正在建立資料庫快照。請在 Amazon RDS 完成資料庫快照後，再次提交請求。

Category	RDS 事件 ID	訊息	備註
復原	RDS-EVENT-0052	多可用區域執行個體復原已啟動。	復原時間依據恢復的資料量而有不同。
復原	RDS-EVENT-0053	多可用區域執行個體復原已完成。擱置容錯移轉或啟動。	此訊息表示 Amazon RDS 已備妥您的資料庫執行個體，以在必要時啟動次要執行個體的容錯移轉。
復原	RDS-EVENT-0361	待命資料庫執行個體的復原已啟動。	待命資料庫執行個體會在復原程序期間重建。資料庫效能會在復原程序期間受到影響。
復原	RDS-EVENT-0362	待命資料庫執行個體的復原已完成。	待命資料庫執行個體會在復原程序期間重建。資料庫效能會在復原程序期間受到影響。
還原	RDS-EVENT-0019	已從資料庫執行個體 *name* 還原至 *name*。	資料庫執行個體已從時間點備份還原。
安全性修補程式	RDS-EVENT-0230	系統更新適用於您的資料庫執行個體。如需套用更新的相關資訊，請參閱《RDS 使用者指南》中的「維護資料庫執行個體」。	新的次要版本作業系統修補程式適用於您的資料庫執行個體。如需套用更新的相關資訊，請參閱 Aurora 資料庫叢集的作業系統更新。
維護	RDS-EVENT-0425	Amazon RDS 無法執行作業系統升級，因為指定的子網路中沒有可用的 IP 地址。選擇具有可用 IP 地址的子網路，然後再試一次。	無

Category	RDS 事件 ID	訊息	備註
維護	RDS-EVENT-0429	由於##可用區域中##執行個體類型的可用容量不足，Amazon RDS 無法執行作業系統升級	無
維護	RDS-EVENT-0501	Amazon RDS 資料庫執行個體的伺服器憑證需要輪換待定的維護動作。	資料庫執行個體的伺服器憑證需要輪換待定的維護動作。Amazon RDS 會在此維護期間重新啟動資料庫，以完成憑證輪換。若要排程此維護，請前往維護與備份索引標籤，然後選擇立即套用或排程下一個維護時段。如果未排定變更，Amazon RDS 會在您的維護動作中顯示的自動套用日期，自動將其套用至您的維護時段。
維護	RDS-EVENT-0502	Amazon RDS 已在下一個維護時段排定資料庫執行個體的伺服器憑證輪換。此維護需要重新啟動資料庫。	無

資料庫參數群組事件

下表顯示當資料庫參數群組為來源類型時的事件類別和事件清單。

類別	RDS 事件 ID	訊息	備註
組態變更	RDS-EVENT-0037	已將參數 *name* 更新為 *value*，套用方法為 *method*。	無

資料庫安全群組事件

下表顯示當資料庫安全群組為來源類型時的事件類別和事件清單。

> **Note**
>
> 資料庫安全群組是 EC2-Classic 的資源。EC2-Classic 在 2022 年 8 月 15 日淘汰。如果您還沒有從 EC2-Classic 遷移至 VPC，建議您盡快這麼做。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的從 EC2-Classic 遷移至 VPC，以及部落格文章 EC2-Classic Networking is Retiring – Here's How to Prepare 。

Category	RDS 事件 ID	訊息	備註
組態變更	RDS-EVENT-0038	已將變更套用至安全群組。	無
失敗	RDS-EVENT-0039	正以 *user* 身分撤銷授權。	由 *user* 擁有的安全群組不存在。已撤銷安全群組的授權，因為它無效。

資料庫碎片群組事件

下表顯示當資料庫碎片群組為來源類型時的事件類別和事件清單。

Category	RDS 事件 ID	訊息	備註
失敗	RDS-EVENT-0324	資料載入任務失敗。	由於錯誤訊息中出現的原因，資料載入任務失敗。
維護	RDS-EVENT-0302	需要您的動作才能完成資料庫碎片群組*##*中子叢集 ID *##*的待定分割碎片任務 *job_ID*。 若要完成操作調用 SQL： `SELECT rds_auror a.limitless_finali`	無

Category	RDS 事件 ID	訊息	備註
		`ze_split_ shard( job_ID);`	
維護	RDS-EVENT-0303	針對資料庫碎片群組##中的子叢集 ID ##，已開始完成分割碎片任務 *job_ID*。	無
維護	RDS-EVENT-0304	資料庫碎片群組##中子叢集 ID ##的分割碎片任務 *job_ID* 已成功完成。具有子叢集 ID ##的新碎片已新增至資料庫碎片群組##。	無
維護	RDS-EVENT-0305	資料庫碎片群組##中子叢集 ID ##的分割碎片任務 *job_ID* 已失敗。	無
維護	RDS-EVENT-0366	資料庫碎片群組##中子叢集 ID ##的分割碎片任務 *job_ID* 已開始。	無
維護	RDS-EVENT-0367	在資料庫碎片群組##中新增路由器任務 *job_ID* 已啟動。	無
維護	RDS-EVENT-0368	新增路由器任務 *job_ID* 已成功完成。具有子叢集 ID ##的新路由器已新增至資料庫碎片群組##。	無
維護	RDS-EVENT-0369	在資料庫碎片群組##中新增路由器任務 *job_ID* 已失敗。	無
維護	RDS-EVENT-0394	在資料庫碎片群組##中新增路由器任務 *job_ID* 已取消。	無

Category	RDS 事件 ID	訊息	備註
維護	RDS-EVENT-0395	在資料庫碎片群組*##*中分割碎片任務 *job_ID* 已取消。	無
notification	RDS-EVENT-0321	初始化資料載入任務的基礎設施。	無
notification	RDS-EVENT-0322	資料載入任務正在進行中。	無
notification	RDS-EVENT-0323	資料載入任務已成功完成。	無
notification	RDS-EVENT-0325	根據客戶的請求取消資料載入任務。	無
notification	RDS-EVENT-0326	資料載入任務已取消。	無

RDS Proxy 事件

下表顯示 RDS Proxy 為來源類型時的事件類別和事件清單。

Category	RDS 事件 ID	訊息	備註
組態變更	RDS-EVENT-0204	RDS 已修改資料庫代理 *name*。	無
組態變更	RDS-EVENT-0207	RDS 已修改資料庫代理 *name* 的端點。	無
組態變更	RDS-EVENT-0213	RDS 偵測到新增了資料庫執行個體，並自動將其新增到資料庫代理 *name* 的目標群組。	無
組態變更	RDS-EVENT-0214	RDS 偵測到刪除了資料庫執行個體 *name*，並自動將其從資料庫代理 *name* 的目標群組 *name* 中移除。	無

Category	RDS 事件 ID	訊息	備註
組態變更	RDS-EVENT-0215	RDS 偵測到刪除了資料庫叢集 *name*，並自動將其從資料庫代理 *name* 的目標群組 *name* 中移除。	無
建立	RDS-EVENT-0203	RDS 已建立資料庫代理 *name*。	無
建立	RDS-EVENT-0206	RDS 已建立資料庫代理 *name* 的端點 *name*。	無
刪除	RDS-EVENT-0205	RDS 已刪除資料庫代理 *name*。	無
刪除	RDS-EVENT-0208	RDS 已刪除資料庫代理 *name* 的端點 *name*。	無
失敗	RDS-EVENT-0243	RDS 無法佈建代理 *name* 的容量，因為您的子網路中沒有足夠的可用 IP 地址：*name*。若要修正此問題，請確定您的子網路具有最低數量的未使用 IP 地址，如 RDS Proxy 文件所建議。	若要確定執行個體類別的建議數量，請參閱 規劃 IP 地址容量。
失敗	RDS-EVENT-0275	RDS 對部分連往資料庫代理*##*的連線進行限流。從用戶端到代理的同時連線要求數目已超過限制。	無

藍/綠部署事件

下表顯示當藍/綠部署為來源類型時的事件類別和事件清單。

如需藍/綠部署的詳細資訊，請參閱 使用 Amazon Aurora 藍/綠部署進行資料庫更新。

Category	Amazon RDS 事件 ID	訊息	備註
建立	RDS-EVENT-0244	藍/綠部署任務已完成。您可以在部署之後，於綠色環境中對資料庫進行其他修改。	無
失敗	RDS-EVENT-0245	因為*##*，建立藍/綠部署失敗。	無
刪除	RDS-EVENT-0246	已刪除藍/綠部署。	無
notification	RDS-EVENT-0247	從 *blue* 至 *green* 的轉換已開始。	無
notification	RDS-EVENT-0248	已完成藍/綠部署上的切換。	無
失敗	RDS-EVENT-0249	已取消藍/綠部署上的切換。	無
notification	RDS-EVENT-0259	從資料庫叢集 *blue* 至 *green* 的轉換已開始。	無
notification	RDS-EVENT-0260	從資料庫叢集 *blue* 至 *green* 的轉換已完成。已將 *blue* 重新命名為 *blue-old*，並將 *green* 重新命名為 *blue*。	無
失敗	RDS-EVENT-0261	由於 *reason*，已取消從資料庫叢集 *blue* 至 *green* 的轉換。	無
notification	RDS-EVENT-0311	將 資料庫叢集 *blue* 轉換至 *green* 的序列同步已啟動。使用序列時轉換可能會導致停機時間延長。	無
notification	RDS-EVENT-0312	將 資料庫叢集 *blue* 轉換至 *green* 的序列同步已完成。	無

Category	Amazon RDS 事件 ID	訊息	備註
失敗	RDS-EVENT-0314	由於序列同步失敗，已取消將資料庫叢集 *blue* 轉換至 *green* 的序列同步。	無
notification	RDS-EVENT-0409	*message*	無

監控 Amazon Aurora 日誌檔案

每個 RDS 資料庫引擎都會產生日誌，供您存取以進行稽核和疑難排解。日誌檔類型視您的資料庫引擎而定。

您可以使用 AWS 管理主控台、AWS Command Line Interface (AWS CLI) 或 Amazon RDS API 來存取資料庫執行個體的資料庫日誌。您無法檢視、觀看或下載交易日誌。

> **Note**
>
> 在部分案例中，日誌包含隱藏的資料。因此，AWS 管理主控台 可能顯示出日誌檔案中的內容，但您下載日誌檔案時可能是空的。

主題

- 檢視並列出資料庫日誌檔案
- 下載資料庫日誌檔案
- 查看資料庫日誌檔案
- 將資料庫日誌發佈至 Amazon CloudWatch Logs
- 使用 REST 讀取日誌檔案內容
- Aurora MySQL資料庫日誌檔案
- Aurora PostgreSQL 資料庫日誌檔案

檢視並列出資料庫日誌檔案

您可使用 AWS 管理主控台 來檢視 Amazon Aurora DB 引擎的資料庫日誌檔案。您可以使用 AWS CLI 或 Amazon RDS API 來列出可用於下載或監控的記錄檔案。

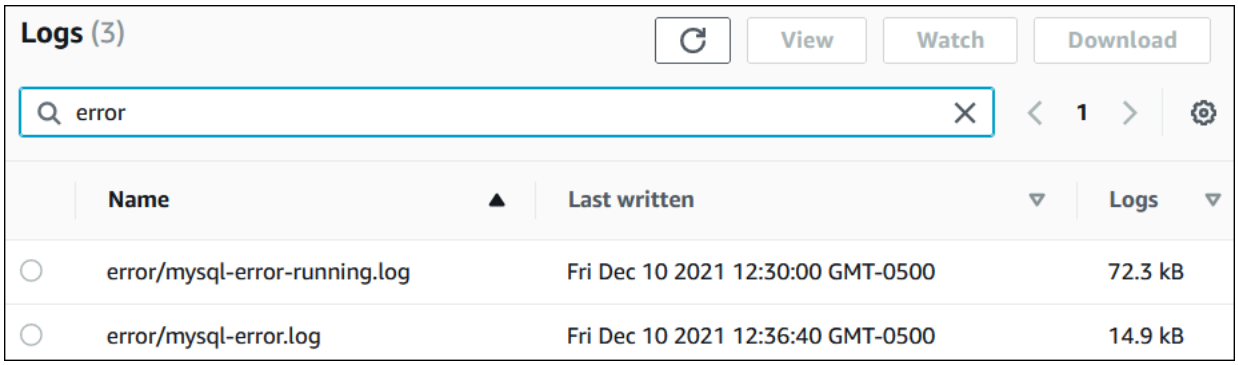
> **Note**
>
> 您無法在 RDS 主控台中檢視 Aurora Serverless v1 資料庫叢集的日誌檔。然而，您可以在 https://console.aws.amazon.com/cloudwatch/ 的 Amazon CloudWatch 主控台進行檢視。

主控台

檢視資料庫日誌檔案

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您想檢視的日誌檔案所在的資料庫執行個體的名稱。
4. 選擇 Logs & events (日誌與事件) 標籤。
5. 向下捲動至 Logs (日誌) 區段。
6. (選用) 輸入搜尋詞來篩選結果。

   下列範例列出由文字 **`error`** 篩選的日誌。



7. 選擇您想要檢視的日誌，然後選擇 View (檢視)。

AWS CLI

若要檢視資料庫執行個體的可用執行個體日誌檔案，請使用 AWS CLI `describe-db-log-files` 命令。

下列範例會回傳名為 `my-db-instance` 的資料庫執行個體之日誌檔案列表。

Example

```
aws rds describe-db-log-files --db-instance-identifier my-db-instance
```

RDS API

若要列出資料庫執行個體的可用資料庫日誌檔案，請使用 Amazon RDS API `DescribeDBLogFiles` 動作。

下載資料庫日誌檔案

您可以使用 AWS 管理主控台、AWS CLI 或 API 來下載資料庫日誌檔案。

主控台

下載資料庫日誌檔案

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您想檢視的日誌檔案所在的資料庫執行個體的名稱。
4. 選擇 Logs & events (日誌與事件) 標籤。
5. 向下捲動至 Logs (日誌) 區段。
6. 在 Logs (日誌) 區段中，選擇您想要下載的日誌旁的按鈕，然後選擇 Download (下載)。
7. 開啟提供之連結的內容 (右鍵) 功能表，然後選擇 Save Link As (另存連結為)。輸入您要儲存日誌檔案的位置，然後選擇 Save (儲存)。

AWS CLI

若要下載資料庫日誌檔案，請使用 AWS CLI 命令 `download-db-log-file-portion`。在預設情況下，這個命令只會下載日誌檔案的最新部分。不過，您可以指定參數 `--starting-token 0` 來下載整個檔案。

下列範例顯示下載名為 log/ERROR.4 之日誌檔案的完整內容方法，並將該檔案儲存於名為 errorlog.txt 的本機檔案中。

Example

對於 Linux、macOS 或 Unix：

```
aws rds download-db-log-file-portion \
    --db-instance-identifier myexampledb \
    --starting-token 0 --output text \
    --log-file-name log/ERROR.4 > errorlog.txt
```

在 Windows 中：

```
aws rds download-db-log-file-portion ^
    --db-instance-identifier myexampledb ^
    --starting-token 0 --output text ^
    --log-file-name log/ERROR.4 > errorlog.txt
```

RDS API

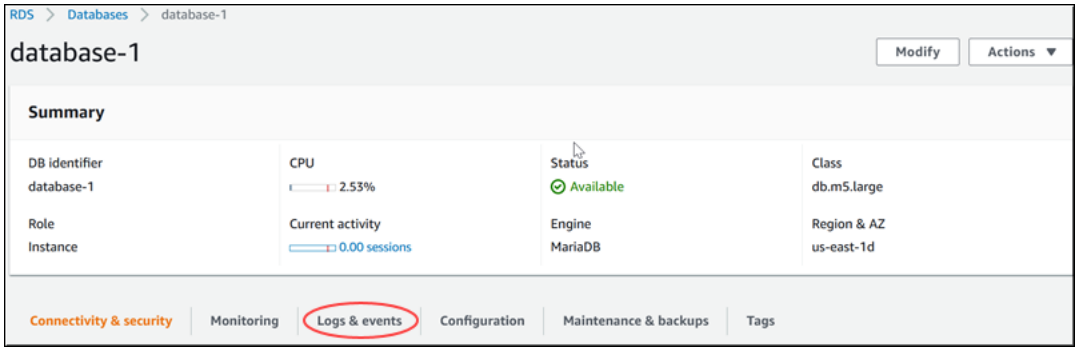
若要下載資料庫日誌檔案，請使用 Amazon RDS API DownloadDBLogFilePortion 動作。

查看資料庫日誌檔案

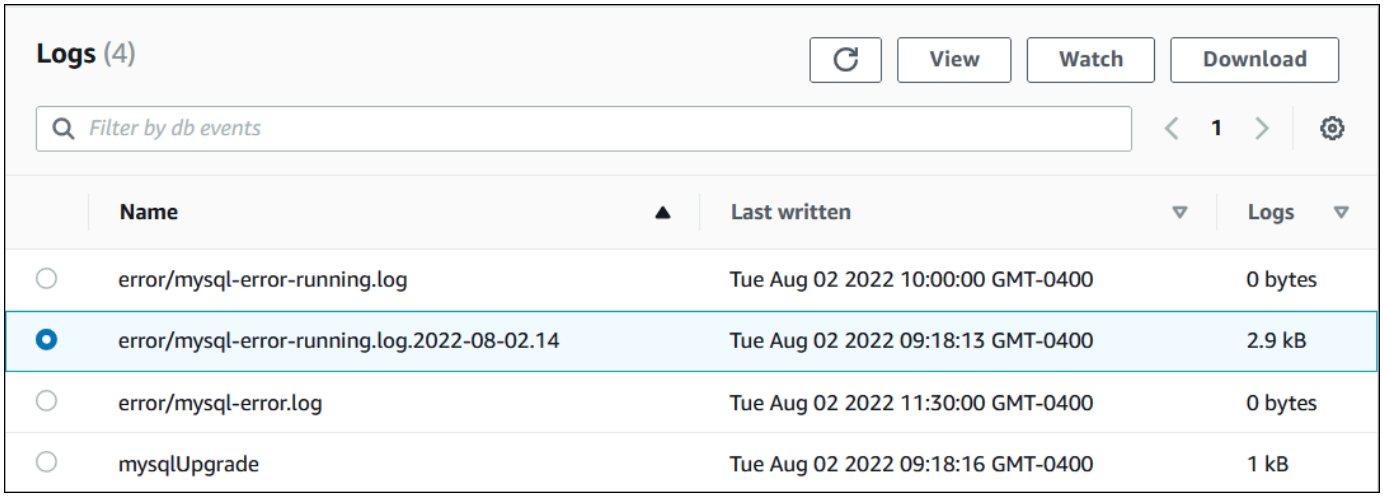
監視資料庫日誌檔等同於在 UNIX 或 Linux 系統上追蹤檔案。您可以使用 AWS 管理主控台 來觀看日誌檔。RDS 每 5 秒會重新整理日誌的結尾。

查看資料庫日誌檔案

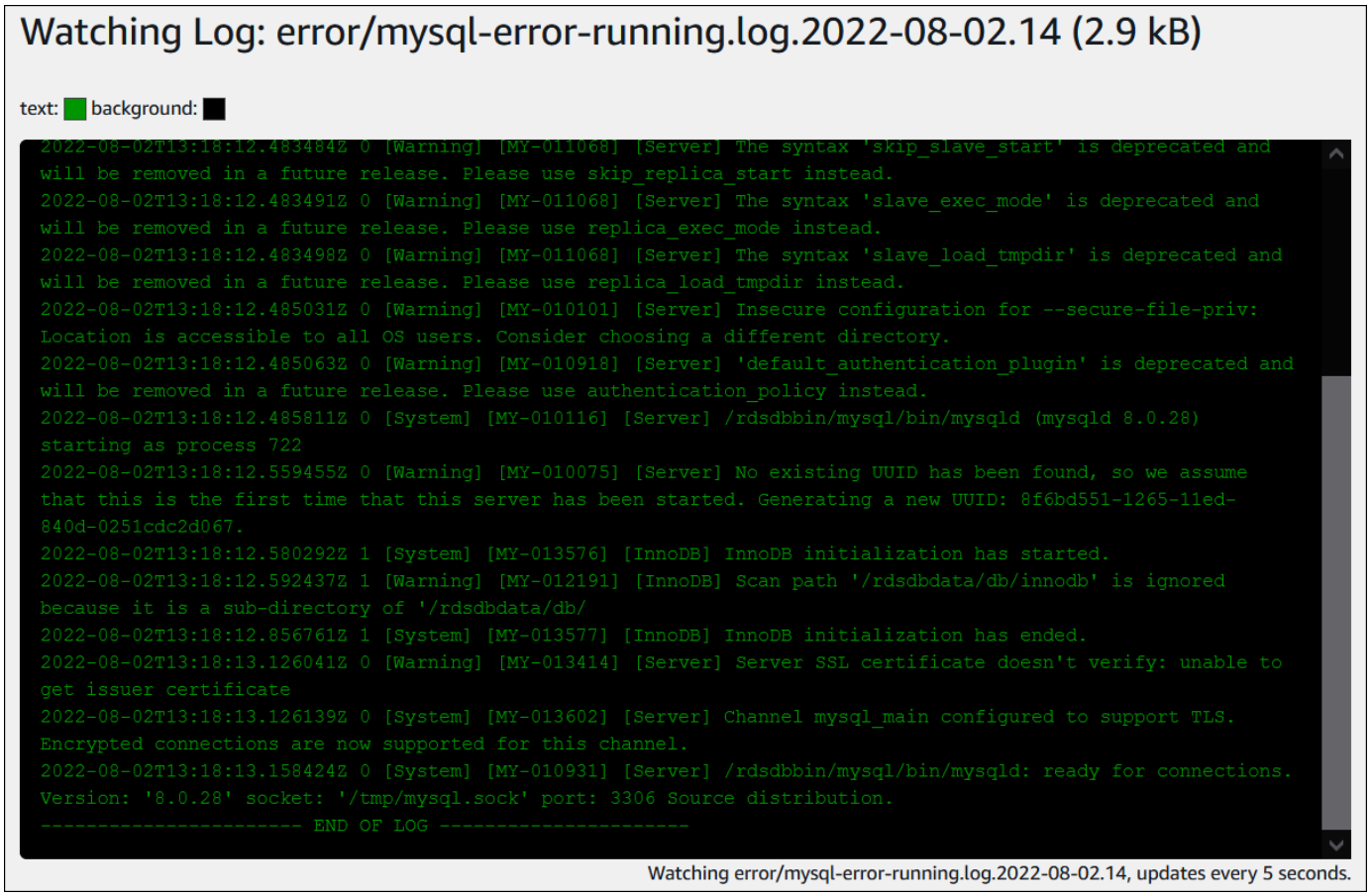
1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您想檢視的日誌檔案所在的資料庫執行個體的名稱。
4. 選擇 Logs & events (日誌與事件) 標籤。



5. 在 Logs (日誌) 區段中，選擇日誌檔案，然後選擇 Watch (監看)。



RDS 會顯示日誌檔的尾端，如下列 MySQL 範例所示。



將資料庫日誌發佈至 Amazon CloudWatch Logs

在內部部署資料庫中，資料庫日誌位於檔案系統上。Amazon RDS 不提供對資料庫叢集的檔案系統上資料庫日誌的主機存取。因此，Amazon RDS 可讓您將資料庫日誌匯出至 Amazon CloudWatch

Logs。您可以使用 CloudWatch Logs，執行日誌資料的即時分析。您也可以將資料存放在高耐久的儲存體中，並使用 CloudWatch Logs 代理程式來管理資料。

主題

- RDS 與 CloudWatch Logs 整合概觀
- 決定要發佈到 CloudWatch Logs 的日誌
- 指定要發佈到 CloudWatch Logs 的日誌
- 在 CloudWatch Logs 中搜尋和篩選您的日誌

RDS 與 CloudWatch Logs 整合概觀

在 CloudWatch Logs 日誌串流是一系列共用相同來源的日誌事件。CloudWatch Logs 中的每個單獨日誌來源組成單獨的日誌串流。日誌群組是共用相同保留、監控和存取控制設定的日誌串流群組。

Amazon Aurora 持續將您的資料庫叢集日誌串流至日誌群組。例如，您所發佈每種類型的日誌都有一個日誌群組 `/aws/rds/cluster/`*`cluster_name`*`/`*`log_type`*。此日誌群組與產生日誌的資料庫執行個體位於相同的 AWS 區域中。

AWS 會長期保留發佈至 CloudWatch Logs 的日誌資料，除非您指定保留期間。如需詳細資訊，請參閱更改在 CloudWatch Logs 中的日誌資料保留期。

決定要發佈到 CloudWatch Logs 的日誌

每個 RDS 資料庫引擎都支援自己的日誌集。若要瞭解資料庫引擎適用的選項，請檢閱下列主題：

- the section called “將 Aurora MySQL 日誌發佈至 CloudWatch Logs”
- the section called “將 Aurora PostgreSQL 日誌發佈至 CloudWatch Logs”

指定要發佈到 CloudWatch Logs 的日誌

您可以指定要在主控台中發佈的日誌。確定您在 AWS Identity and Access Management (IAM) 中有服務連結角色。如需服務連結角色的詳細資訊，請參閱使用 Amazon Aurora 的服務連結角色。

若要指定要發佈的日誌

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。

3. 執行下列任何一項：

   - 選擇建立資料庫。
   - 從清單中，選擇您的資料庫，然後選擇 Modify (修改)。

4. 在 Logs exports (日誌匯出) 中，選擇要發佈的日誌。

   下列範例指定稽核日誌、錯誤日誌、一般日誌、執行個體日誌、IAM 資料庫身分驗證錯誤日誌，以及 Aurora MySQL 資料庫叢集的慢速查詢日誌。

在 CloudWatch Logs 中搜尋和篩選您的日誌

您可以使用 CloudWatch Logs 主控台搜尋與指定條件相符的日誌項目。您可以透過導向 CloudWatch Logs 主控台的 RDS 主控台存取日誌，或從 CloudWatch Logs 主控台直接存取。

若要使用 RDS 主控台搜尋 RDS 日誌

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇一個資料庫叢集或資料庫執行個體。
4. 選擇 Configuration (組態)。
5. 在 Published logs (發佈日誌) 下方，選擇您要檢視的資料庫日誌。

使用 CloudWatch Logs 主控台搜尋 RDS 日誌

1. 透過 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在導覽窗格中，選擇 Log groups (日誌群組)。
3. 在篩選方塊中，輸入 **`/aws/rds`**。
4. 針對 Log Groups (日誌群組)，輸入包含要搜尋之日誌串流的日誌群組名稱。
5. 對於 Log Streams (日誌串流)，選擇要搜尋的日誌串流名稱。
6. 在 Log events (日誌事件) 下方，輸入要使用的篩選條件語法。

如需詳細資訊，請參閱《Amazon CloudWatch Logs 使用者指南》中的搜尋和篩選日誌資料。如需說明如何監控 RDS 日誌的教學課程，請參閱使用 Amazon CloudWatch Logs、AWS Lambda 和 Amazon SNS 為 Amazon RDS 建置主動資料庫監控。

使用 REST 讀取日誌檔案內容

Amazon RDS 提供允許存取資料庫執行個體日誌檔案的 REST 端點。若您需要編寫應用程式來串流 Amazon RDS 日誌檔案內容，此操作非常有幫助。

語法是：

```
GET /v13/downloadCompleteLogFile/DBInstanceIdentifier/LogFileName HTTP/1.1
Content-type: application/json
host: rds.region.amazonaws.com
```

下列是必要參數：

- *DBInstanceIdentifier* — 含有您想要下載之日誌檔案的資料庫執行個體名稱。
- *LogFileName* — 要下載的日誌檔案名稱。

回應包括要求之日誌檔案的內容，做為串流形式。

下列範例下載名為 log/ERROR.6 的日誌檔案供名為 sample-sql 的資料庫執行個體使用，位於 us-west-2 區域中。

```
GET /v13/downloadCompleteLogFile/sample-sql/log/ERROR.6 HTTP/1.1
host: rds.us-west-2.amazonaws.com
X-Amz-Security-Token: AQoDYXdzEIH//////////
wEa0AIXLhngC5zp9CyB1R6abwKrXHVR5efnAVN3XvR7IwqKYalFSn6UyJuEFTft9nObglx4QJ+GXV9cpACkETq=
X-Amz-Date: 20140903T233749Z
X-Amz-Algorithm: AWS4-HMAC-SHA256
X-Amz-Credential: AKIADQKE4SARGYLE/20140903/us-west-2/rds/aws4_request
X-Amz-SignedHeaders: host
X-Amz-Content-SHA256: e3b0c44298fc1c229afbf4c8996fb92427ae41e4649b934de495991b7852b855
X-Amz-Expires: 86400
X-Amz-Signature: 353a4f14b3f250142d9afc34f9f9948154d46ce7d4ec091d0cdabbcf8b40c558
```

若您指定不存在的資料庫執行個體，回應會包含下列錯誤：

- DBInstanceNotFound —*DBInstanceIdentifier* 不代表現有的資料庫執行個體。(HTTP 狀態碼：404)

Aurora MySQL資料庫日誌檔案

您可以透過 Amazon RDS 主控台、Amazon RDS API AWS CLI或 AWS SDKs 直接監控 Aurora MySQL 日誌。您可以將日誌指向主要資料庫中的資料庫表並查詢該表格，藉此存取 MySQL 日誌。您可以使用 mysqlbinlog 公用程式來下載二進位日誌。

如需關於檢視、下載與查看資料庫日誌檔案的資訊，請參閱監控 Amazon Aurora 日誌檔案。

主題

- Aurora MySQL 資料庫日誌概觀
- 將 Aurora MySQL日誌輸出傳送至資料表
- 為單一可用區資料庫設定 Aurora MySQL 二進位記錄
- 存取 MySQL 二進位日誌

Aurora MySQL 資料庫日誌概觀

您可以監控下列類型的 Aurora MySQL 日誌檔案：

- 錯誤日誌
- 慢查詢日誌
- 一般日誌
- 稽核日誌
- 執行個體日誌
- IAM 資料庫身分驗證錯誤日誌

預設情況下會產生 Aurora MySQL 錯誤日誌。透過在資料庫參數群組中設定參數，產生慢查詢日誌和一般日誌。

主題

- Aurora MySQL 錯誤日誌
- Aurora MySQL 慢查詢與一般查詢
- Aurora MySQL 稽核日誌
- Aurora MySQL 執行個體日誌
- Aurora MySQL 的日誌輪換與保留

- 將 Aurora MySQL 日誌發佈至 Amazon CloudWatch Logs

Aurora MySQL 錯誤日誌

Aurora MySQL 會將錯誤寫入 `mysql-error.log` 檔案中。每個日誌檔案的產生時間 (UTC 時區) 皆會附加於檔案名稱中。日誌檔案也有時間戳記，可協助您判定日誌項目寫入的時間。

只有在開機、當機以及發生錯誤時，Aurora MySQL 才會寫入錯誤日誌。資料庫執行個體可在未寫入新項目到錯誤日誌的情況下持續執行數小時或數日。若您沒有看到最近的項目，這是因為伺服器未遇到需寫入日誌項目的錯誤。

根據設計，系統會篩選錯誤日誌，以僅顯示未預期的事件，例如錯誤。不過，錯誤日誌還包含一些其他未顯示的資料庫資訊，例如查詢進度。因此即使沒有任何實際錯誤，但錯誤日誌的大小可能會因為在進行的資料庫活動而增加。雖然您可能會在 中看到錯誤日誌的特定位元組或 KB 大小 AWS 管理主控台，但當您下載它們時，它們可能會有 0 個位元組。

Aurora MySQL 每 5 分鐘將 `mysql-error.log` 寫入磁碟一次。這會將日誌的內容追加到 `mysql-error-running.log`。

Aurora MySQL 每小時會輪換 `mysql-error-running.log` 檔案一次。

> **Note**
>
> Amazon RDS 和 Aurora 的日誌保留期間不同。

Aurora MySQL 慢查詢與一般查詢

您可以將 Aurora MySQL 慢查詢日誌與一般日誌寫入至檔案或資料庫表格。若要這樣做，請在您的資料庫參數群組中設定參數。如需建立和修改資料庫參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。您必須先設定這些參數，才能在 Amazon RDS 主控台或使用 Amazon RDS API、Amazon RDS CLI 或 AWS SDKs來檢視慢查詢日誌或一般日誌。

您可以使用清單中的參數來控制 Aurora MySQL 日誌記錄：

- `slow_query_log`：若要建立慢查詢，請設為 1。預設為 0。
- `general_log`：若要建立一般日誌，請設為 1。預設值為 0。
- `long_query_time`：若要避免快速執行查詢記錄於慢查詢日誌中，請為需記錄之最短查詢執行時間指定一個值，以秒為單位。預設為 10 秒，最短時間為 0。若 log_output = FILE，您可以指定以毫秒

解析度為單位的浮點值。若 log_output = TABLE，您必須指定以秒為單位的整數值。只會記錄執行時間超過 long_query_time 值的查詢。例如，將 long_query_time 設為 0.1 可避免記錄任何在 100 毫秒內執行之查詢。

- log_queries_not_using_indexes：若要將所有不使用索引的查詢記錄於慢查詢日誌中，請設為 1。系統會記錄不使用索引的查詢，即使其執行時間低於 long_query_time 參數的值。預設值為 0。
- log_output *option*：您可為 log_output 參數指定下列其中一個選項。
  - TABLE – 將一般查詢寫入 mysql.general_log 表格，而慢查詢則寫入 mysql.slow_log 表格。
  - FILE – 同時將一般與慢查詢日誌寫入檔案系統中。
  - NONE – 停用日誌記錄。

  對於 Aurora MySQL 第 2 版和第 3 版，log_output 的預設值為 FILE。

若要讓慢速查詢資料出現在 Amazon CloudWatch Logs 中，必須符合下列條件：

- CloudWatch Logs 必須設定為包含慢速查詢日誌。
- 必須啟用 slow_query_log。
- log_output 必須設定為 FILE。
- 查詢必須花費超過為 long_query_time 設定的時間。

如需慢查詢與一般日誌的詳細資訊，請參閱 MySQL 文件中的下列主題：

- 慢查詢日誌
- 一般查詢日誌

Aurora MySQL 稽核日誌

Aurora MySQL 的稽核日誌稱為進階稽核。若要開啟進階稽核，請設定某些資料庫叢集參數。如需詳細資訊，請參閱使用進階稽核與 Amazon Aurora MySQL 資料庫叢集搭配。

Aurora MySQL 執行個體日誌

Aurora 會為已啟用自動暫停的資料庫執行個體建立單獨的日誌檔案。此 instance.log 檔案會記錄這些資料庫執行個體無法如預期暫停的任何原因。如需執行個體日誌檔案行為和 Aurora 自動暫停功能的詳細資訊，請參閱監控 Aurora Serverless v2 暫停和繼續活動。

Aurora MySQL 的日誌輪換與保留

日誌記錄功能啟用時，Amazon Aurora 會定期輪換或刪除日誌檔案。此方法為預防措施，可降低大型日誌檔封鎖資料庫使用或影響效能的可能性。Aurora MySQL 會處理輪換與刪除，如下所示：

- Aurora MySQL 錯誤日誌檔案的大小限制為不能超過資料庫執行個體本機儲存空間的 15%。為保持此閾值，日誌會每小時自動輪換。Aurora MySQL 會在 30 天後或達到磁碟空間的 15% 時移除日誌。若在移除舊日誌檔案後總日誌檔案大小仍超過閾值，將會從最舊的日誌檔案開始刪除，直到日誌檔案大小不再超過閾值為止。
- Aurora MySQL 會在 24 小時後或在儲存空間使用量達 15% 時，移除稽核、一般和慢查詢日誌。
- 當 `FILE` 日誌記錄啟用時，每個小時將檢視一般日誌與慢查詢日誌檔一次，而超過 24 小時的日誌檔將會刪除。在部分情況下，刪除後剩餘的總日誌檔案大小可能超過資料庫執行個體本機空間的 15% 閾值。於這些狀況中，最舊的日誌檔將遭刪除，直到日誌檔大小不再超過閾值。
- 當 `TABLE` 日誌已啟用時，不會輪換或刪除日誌資料表。當所有日誌合併後的大小過大時，日誌資料表將被截斷。您可訂閱 `low storage` 事件類別，當應手動輪換或刪除日誌資料表以釋放空間時就會通知您。如需詳細資訊，請參閱使用 Amazon RDS 事件通知。

  您可以呼叫 `mysql.rds_rotate_general_log` 程序來手動輪換 `mysql.general_log` 表格。您可以呼叫 `mysql.slow_log` 程序來輪換 `mysql.rds_rotate_slow_log` 表格。

  手動輪換日誌資料表時，目前日誌資料表會複製到備份日誌資料表，並移除目前日誌資料表中的項目。如果備份日誌資料表已存在，則其會在目前日誌資料表複製到備份之前刪除。如有需要，您可以查詢備份日誌資料表。`mysql.general_log` 資料表的備份日誌資料表名為 `mysql.general_log_backup`。`mysql.slow_log` 資料表的備份日誌資料表名為 `mysql.slow_log_backup`。
- 當檔案大小達到 100 MB 時，會輪換 Aurora MySQL 稽核日誌，並在 24 小時後刪除。
- Amazon RDS 會輪換大於 10 MB 的 IAM 資料庫身分驗證錯誤日誌檔案。Amazon RDS 會移除超過五天或大於 100 MB 的 IAM 資料庫身分驗證錯誤日誌檔案。

若要使用來自 Amazon RDS 主控台、Amazon RDS API、Amazon RDS CLI 或 AWS SDKs日誌，請將 `log_output` 參數設定為 FILE。如同 Aurora MySQL 錯誤日誌，這些日誌檔案也會每小時輪換。前 24 小時之間產生的日誌檔案將會保留。請注意，Amazon RDS 和 Aurora 的保留期間不同。

將 Aurora MySQL 日誌發佈至 Amazon CloudWatch Logs

您可以設定 Aurora MySQL 資料庫叢集，以將日誌資料發佈至 Amazon CloudWatch Logs 中的日誌群組。使用 CloudWatch Logs，您可以執行日誌資料的即時分析，並使用 CloudWatch 來建立警示和檢

視指標。您可以使用 CloudWatch Logs 將日誌記錄存放在高耐用性的儲存裝置中。如需詳細資訊，請參閱將 Amazon Aurora MySQL 日誌發佈至 Amazon CloudWatch Logs。

將 Aurora MySQL日誌輸出傳送至資料表

您可以建立資料庫參數群組，並將 `log_output` 伺服器參數設為 `TABLE`，將一般與慢查詢日誌導向資料庫執行個體上的表格。一般查詢會記錄於 `mysql.general_log` 表格，而慢查詢會記錄至 `mysql.slow_log` 表格。您可以查詢表格來存取日誌資訊。啟用此日誌記錄會增加寫入至資料庫的資料總數量，可能造成效能降低。

一般日誌和慢查詢日誌根據預設將會停用。為啟用日誌記錄至表格，您也必須將 `general_log` 與 `slow_query_log` 伺服器參數設為 `1`。

日誌表格將持續擴增，直到重設適用參數為 `0` 時，各日誌記錄活動才會關閉。大量資料通常會隨時間累積，可能會佔用配得之儲存空間的大量比例。Amazon Aurora Amazon RDS 不允許截斷記錄資料表，但是您可以移動其中的內容。輪換表格可將其內容儲存至備份表格，然後建立新的空白日誌表格。您可以透過以下命令列程序來手動輪換日誌表格，命令提示由 PROMPT> 表示：

```
PROMPT> CALL mysql.rds_rotate_slow_log;
PROMPT> CALL mysql.rds_rotate_general_log;
```

為完全移除舊資料並重新取得磁碟空間，請連續呼叫兩次適用的程序。

為單一可用區資料庫設定 Aurora MySQL 二進位記錄

二進位日誌是一組日誌檔案，其中包含對 AuroraMySQL 伺服器執行個體所做資料修正的相關資訊。二進位日誌包含的資訊如下：

- 描述資料庫變更的事件 (如建立資料表或修改資料列)
- 有關更新了資料的每個陳述式持續時間的資訊
- 本應更新資料但未更新的陳述式事件

二進位日誌會記錄複寫過程中傳送的陳述式。某些復原操作也需要這些日誌。如需詳細資訊，請參閱 MySQL 文件中的二進位日誌。

二進位日誌只能從主要資料庫執行個體存取，而不能從複本存取。

Amazon Aurora 上的 MySQL 支援 row-based (列型)、statement-based (陳述式型)，和 mixed (混和式) 二進位記錄格式。除非需要特定的 binlog 格式，否則我們建議混合使用。如需各種 Aurora MySQL 二進位日誌格式的詳細資訊，請參閱 MySQL 說明文件中的二進位記錄格式。

如果您打算使用複寫，二進位日誌記錄格式很重要，因為這決定資料變更的記錄，而此記錄會記錄在來源中並傳送到複寫目標。如需有關複寫時各種二進位日誌記錄格式的優缺點的資訊，請參閱 MySQL 文件中的基於陳述式和基於列的複寫的優缺點。

> **Important**
>
> 使用 MySQL 8.0.34 時，MySQL 已棄用 `binlog_format` 參數。在較新的 MySQL 版本中，MySQL 計劃移除參數，並僅支援資料列型複寫。因此，我們建議將資料列型記錄用於新的 MySQL 複寫設定。如需詳細資訊，請參閱 MySQL 文件中的 binlog_format。
> MySQL 8.0 版和 8.4 版接受參數 `binlog_format`。使用此參數時，MySQL 會發出棄用警告。在未來的主要版本中，MySQL 會移除參數 `binlog_format`。
> 基於陳述式的複寫可能會造成來源資料庫叢集與僅供讀取複本不一致。如需詳細資訊，請參閱 MySQL 文件中的二進位日誌記錄中安全和不安全陳述式的判定。
> 啟用二進位記錄會增加資料庫叢集的寫入磁碟 I/O 操作次數。您可以使用 `VolumeWriteIOPs` CloudWatch 指標來監控 IOPS 使用情況。

設定 MySQL 二進位記錄格式

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。
3. 選擇您想要修改的與資料庫叢集相關聯的資料庫叢集參數群組。

   您無法修改預設參數群組。如果資料庫叢集使用預設參數群組，請建立新的參數群組，並將它與資料庫叢集建立關聯。

   如需參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。
4. 在動作中選擇編輯。
5. 將 `binlog_format` 參數設為您選擇的二進位日誌記錄格式 (`ROW`、`STATEMENT` 或 `MIXED`)。您還可使用值 `OFF`，來關閉二進位記錄。

   > **Note**
   >
   > 在資料庫叢集參數群組中將 `binlog_format` 設定為 `OFF` 會停用 `log_bin` 工作階段變數。這會停用 Aurora MySQL 資料庫叢集上的二進位記錄，進而在資料庫中將 `binlog_format` 工作階段變數重設為 `ROW` 的預設值。
6. 選擇 Save changes (儲存變更) 來儲存對資料庫叢集參數群組的更新。

執行這些步驟後，您必須重新啟動資料庫叢集中的寫入器執行個體，才能套用變更。在 Aurora MySQL 2.09 版和更低版本中，當您將寫入器執行個體重新開機時，資料庫叢集中的所有讀取器執行個體也會重新開機。在 Aurora MySQL 2.10 版及更新版本中，您必須手動將所有讀取器執行個體重新開機。如需詳細資訊，請參閱重新啟動 Amazon Aurora 資料庫叢集或 Amazon Aurora 資料庫執行個體。

> **Important**
>
> 變更資料庫叢集參數群組會影響使用該參數群組的所有資料庫叢集。如果您想要為 AWS 區域中的不同 Aurora MySQL 資料庫叢集指定不同的二進位記錄格式，資料庫叢集必須使用不同的資料庫叢集參數群組。這些參數群組會識別不同的記錄格式。將適當的資料庫叢集參數群組指派給每個資料庫叢集。如需 Aurora MySQL 參數的詳細資訊，請參閱 Aurora MySQL 組態參數。

存取 MySQL 二進位日誌

您可使用 mysqlbinlog 公用程式，從 RDS for MySQL 資料庫執行個體下載或串流二進位日誌。二進位日誌會下載至您的本機電腦，您可於此執行動作，例如使用 mysql 公用程式來重新執行日誌。如需使用 mysqlbinlog 公用程式的詳細資訊，請參閱 MySQL 文件中的使用 mysqlbinlog 備份二進位日誌檔案。

若要在 Amazon RDS 執行個體上執行 mysqlbinlog 公用程式，請使用下列選項：

- `--read-from-remote-server` - 必要。
- `--host` – 來自執行個體端點的 DNS 名稱。
- `--port` – 執行個體使用的連接埠。
- `--user` – 已授予 `REPLICATION SLAVE` 許可的 MySQL 使用者。
- `--password` – MySQL 使用者的密碼，或者省略密碼值，讓公用程式提示您密碼。
- `--raw` – 以二進位格式下載檔案。
- `--result-file` – 接收列輸出的本機檔案。
- `--stop-never` – 串流二進位日誌檔。
- `--verbose` – 當您使用 `ROW` binlog 格式時，請加入此選項，將資料列事件視為虛擬 SQL 陳述式。如需 `--verbose` 選項的詳細資訊，請參閱 MySQL 文件中的 mysqlbinlog row event display。
- 指定一個或一個以上的二進位日誌檔案名稱。若要取得可用日誌清單，請使用 SQL 命令 `SHOW BINARY LOGS`。

如需 mysqlbinlog 選項的詳細資訊，請參閱 MySQL 文件中的 mysqlbinlog — 處理二進位日誌檔案的公用程式。

下列範例顯示如何使用 mysqlbinlog 公用程式。

對於 Linux、macOS 或 Unix：

```
mysqlbinlog \
    --read-from-remote-server \
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com \
    --port=3306  \
    --user ReplUser \
    --password \
    --raw \
    --verbose \
    --result-file=/tmp/ \
    binlog.00098
```

在 Windows 中：

```
mysqlbinlog ^
    --read-from-remote-server ^
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com ^
    --port=3306  ^
    --user ReplUser ^
    --password ^
    --raw ^
    --verbose ^
    --result-file=/tmp/ ^
    binlog.00098
```

二進位日誌必須在資料庫執行個體上保持可用，mysqlbinlog 公用程式才能存取它們。若要確保其可用性，請使用 mysql.rds_set_configuration 預存程序，並指定有足夠時間供您下載日誌的期間。如果未設定此組態，Amazon RDS 會盡快清除二進位日誌，導致 mysqlbinlog 公用程式擷取的二進位日誌出現差距。

下列範例將保留期間設定為 1 天。

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

若要顯示目前設定，請使用 mysql.rds_show_configuration 預存程序。

```
call mysql.rds_show_configuration;
```

Aurora PostgreSQL 資料庫日誌檔案

您可以監控以下 Aurora PostgreSQL 日誌檔案類型：

- PostgreSQL 日誌
- 執行個體日誌
- IAM 資料庫身分驗證錯誤日誌

> **Note**
>
> 若要啟用 IAM 資料庫身分驗證錯誤日誌，您必須先為 Aurora PostgreSQL 資料庫叢集啟用 IAM 資料庫身分驗證。如需啟用 IAM 資料庫身分驗證的詳細資訊，請參閱 啟用和停用 IAM 資料庫身分驗證。

Aurora PostgreSQL 會將資料庫活動記錄到預設的 PostgreSQL 日誌檔。對於內部部署 PostgreSQL 資料庫執行個體，這些訊息會在本機存放於 `log/postgresql.log` 中。對於 Aurora PostgreSQL 資料庫叢集，日誌檔可在 Aurora 叢集上取得。這些日誌也可以透過 存取 AWS 管理主控台，您可以在其中檢視或下載。預設的記錄層級會擷取登入失敗、嚴重的伺服器錯誤、死鎖和查詢失敗。

如需有關如何檢視、下載和監看檔案型資料庫日誌的詳細資訊，請參閱 監控 Amazon Aurora 日誌檔案。若要進一步了解 PostgreSQL 日誌，請參閱 Working with Amazon RDS and Aurora PostgreSQL logs: Part 1 (使用 Amazon RDS 和 Aurora PostgreSQL 日誌：第 1 部分) 以及 Working with Amazon RDS and Aurora PostgreSQL logs: Part 2 (使用 Amazon RDS 和 Aurora PostgreSQL 日誌：第 2 部分)。

除了本主題中討論的標準 PostgreSQL 日誌之外，Aurora PostgreSQL 也支援 PostgreSQL 稽核擴充功能 (`pgAudit`)。大多數受管制的產業和政府機構都需要維護對資料所做變更的稽核日誌或稽核線索，以符合法律要求。如需安裝與使用 pgAudit 的資訊，請參閱 使用 PgAudit 記錄資料庫活動。

Aurora 會為已啟用自動暫停的資料庫執行個體建立單獨的日誌檔案。此 instance.log 檔案會記錄這些資料庫執行個體無法如預期暫停的任何原因。如需執行個體日誌檔案行為和 Aurora 自動暫停功能的詳細資訊，請參閱監控 Aurora Serverless v2 暫停和繼續活動。

主題

- 用於在 Aurora PostgreSQL 中進行記錄的參數
- 開啟 Aurora PostgreSQL 資料庫叢集 的查詢記錄

用於在 Aurora PostgreSQL 中進行記錄的參數

您可以修改各種參數，為 Aurora PostgreSQL 資料庫叢集自訂記錄行為。在下表中，您可以找到影響日誌檔存放時間、何時輪換日誌，以及是否以 CSV (逗號分隔值) 格式輸出日誌。您也可以找到已傳送至 STDERR 的文字輸出，以及其他設定。若要變更可修改之參數的設定，請將自訂資料庫叢集參數群組用於 Aurora PostgreSQL 資料庫叢集。如需詳細資訊，請參閱 Amazon Aurora 的參數群組。

參數	預設	Description
log_destination	stderr	設定日誌的輸出格式。預設值是 `stderr`，但您也可以將 `csvlog` 新增至設定來指定逗號分隔值 (CSV)。如需詳細資訊，請參閱設定日誌目標 (stderr、csvlog)。
log_filename	postgresql.log.%Y-%m-%d-%H%M	指定日誌檔名稱的模式。除了預設值之外，此參數還支援檔案名稱模式的 `postgresql.log.%Y-%m-%d` 和 `postgresql.log.%Y-%m-%d-%H`。對於 Aurora PostgreSQL 17.4 版和更新版本，您無法修改此參數。
log_line_prefix	%t:%r:%u@%d:[%p]:	定義寫入至 `stderr` 的每個日誌行的字首，以記錄時間 (%t)、遠端主機 (%r)、使用者 (%u)、資料庫 (%d) 和程序 ID (%p)。
log_rotation_age	60	日誌檔會多少分鐘後自動轉換。您可以在 1 到 1440 分鐘的範圍內變更此值。如需詳細資訊，請參閱設定日誌檔案輪換。
log_rotation_size	–	日誌檔自動轉換的大小 (kB)。您可以在 50,000 到 1,000,000 KB 的範圍內變更此值。如需詳細資訊，請參閱 設定日誌檔案輪換。
rds.log_retention_period	4320	早於指定分鐘數的 PostgreSQL 日誌將遭到刪除。預設值 4320 分鐘將在 3 天後刪除日誌檔案。如需詳細資訊，請參閱設定日誌保留期間。

如要識別應用程式問題，您可在日誌中尋找查詢失敗、登入失敗、鎖死和致命的伺服器錯誤。例如，假設您已將舊版應用程式從 Oracle 轉換為 Aurora PostgreSQL，但並非所有查詢都已正確轉換。這些格

式不正確的查詢會產生您可在日誌中尋找的錯誤訊息，以協助識別問題。如需記錄查詢的詳細資訊，請參閱 開啟 Aurora PostgreSQL 資料庫叢集 的查詢記錄。

在下列主題中，您可以找到如何設定各種參數的相關資訊，這些參數控制 PostgreSQL 日誌的基本詳細資訊。

主題

- 設定日誌保留期間
- 設定日誌檔案輪換
- 設定日誌目標 (stderr、csvlog)
- 了解 log_line_prefix 參數

設定日誌保留期間

`rds.log_retention_period` 參數指定 Aurora PostgreSQL 資料庫叢集保留其日誌檔的時間長度。預設設定為 3 天 (4,320 分鐘)，但您可以將此值設為 1 天 (1,440 分鐘) 至 7 天 (10,080 分鐘)。請確定您的 Aurora PostgreSQL 資料庫叢集具有足夠的儲存空間來保留日誌檔一段時間。

我們建議您定期將日誌發佈至 Amazon CloudWatch Logs 中，如此，您便可在日誌從 Aurora PostgreSQL 資料庫叢集中移除後的很長時間內檢視並分析系統資料。如需詳細資訊，請參閱 將 Aurora PostgreSQL 日誌發佈至 Amazon CloudWatch Logs。設定 CloudWatch 發佈之後，Aurora 在日誌發佈至 CloudWatch Logs 之前不會將其刪除。

在資料庫執行個體的儲存體達到臨界值時，Amazon Aurora 會壓縮較舊的 PostgreSQL 記錄。Aurora 會使用 gzip 壓縮公用程式來壓縮檔案。如需詳細資訊，請參閱 gzip 網站。

當資料庫執行個體的儲存體不足且所有可用的日誌皆已壓縮時，您會收到類似如下的警告：

```
Warning: local storage for PostgreSQL log files is critically low for
this Aurora PostgreSQL instance, and could lead to a database outage.
```

若儲存體不足，Aurora 可能會在指定的保留期間結束之前刪除壓縮的 PostgreSQL 日誌。若發生這種狀況，您會看到類似下列內容的訊息：

```
The oldest PostgreSQL log files were deleted due to local storage constraints.
```

設定日誌檔案輪換

依預設，Aurora 每小時都會建立新的日誌檔。時間由 log_rotation_age 參數控制。此參數的預設值為 60 (分鐘)，但您可以將其設為從 1 分鐘至 24 小時 (1,440 分鐘) 的任何時間。在輪換時，會建立一個新的不同日誌檔案。該檔案的命名是依據 log_filename 參數所指定的模式。

日誌檔案也可依其大小進行旋轉，如 log_rotation_size 參數中所指定。此參數指定當日誌達到指定大小 (以 KB 為單位) 時應輪換日誌。Aurora PostgreSQL 資料庫叢集的預設 log_rotation_size 為 100000 KB，但您可將此值設為 50,000 到 1,000,000 KB 間的任意值。

日誌檔案名稱會以 log_filename 參數中指定的檔案名稱模式為基礎。此參數的可用設定如下所示：

- postgresql.log.%Y-%m-%d – 日誌檔名稱的預設格式。在日誌檔的名稱中包含年、月和日期。
- postgresql.log.%Y-%m-%d-%H – 在日誌檔名稱格式中包括小時。
- postgresql.log.%Y-%m-%d-%H%M – 在日誌檔名稱格式中包括小時:分鐘。

如果您將 log_rotation_age 參數設為少於 60 分鐘，請將 log_filename 參數設為分鐘格式：

如需詳細資訊，請參閱 PostgreSQL 文件中的 log_rotation_age 和 log_rotation_size。

設定日誌目標 (**stderr**、**csvlog**)

預設情況下，Aurora PostgreSQL 生成標準錯誤 (stderr) 格式的日誌。此格式為 log_destination 參數的預設設定。每則訊息都會使用 log_line_prefix 參數中指定的模式作為字首。如需詳細資訊，請參閱了解 log_line_prefix 參數。

Aurora PostgreSQL 也會以 csvlog 格式產生日誌檔。將日誌資料當作逗號分隔值 (CSV) 進行分析時，csvlog 很有用。例如，假設您使用 log_fdw 延伸模組，將日誌作為外部資料表處理。在 stderr 日誌檔案上建立的外部資料表包含一個具日誌事件資料的單一欄。透過將 csvlog 新增至 log_destination 參數，您可以取得 CSV 格式的日誌檔，其中包含外部資料表的多個資料欄的分界。您現在可以更輕鬆地排序和分析日誌。

如果您為此參數指定 csvlog，請注意會同時產生 stderr 和 csvlog 檔案。請務必監控日誌所使用的儲存體，同時考慮 rds.log_retention_period 及影響日誌儲存體和更換的其他設定。使用 stderr 和 csvlog 會使日誌所使用的儲存體空間增加一倍以上。

如果您將 csvlog 新增至 log_destination，並且想要單獨還原為 stderr，則需要重設參數。若要這麼做，請開啟 Amazon RDS 主控台，然後您的執行個體開啟自訂資料庫叢集參數群組。選擇 log_destination 參數、選擇 Edit parameter (編輯參數]，然後選擇 Reset (重設)。

如需有關設定日誌記錄的詳細資訊，請參閱使用 Amazon RDS 和 Aurora PostgreSQL 日誌：第 1 部分。

了解 log_line_prefix 參數

stderr 日誌格式會將 log_line_prefix 參數指定的詳細資訊作為每個日誌訊息的字首。預設值為：

```
%t:%r:%u@%d:[%p]:t
```

從 Aurora PostgreSQL 第 16 版開始，您也可以選擇：

```
%m:%r:%u@%d:[%p]:%l:%e:%s:%v:%x:%c:%q%a
```

每個傳送至 stderr 的日誌項目都包含根據所選取值的下列資訊：

- %t – 不到毫秒的日誌項目時間
- %m – 毫秒的日誌項目時間
- %r – 遠端主機地址
- %u@%d – 使用者名稱 @ 資料庫名稱
- [%p] – 程序 ID (若可用)
- %l – 每個工作階段的日誌行號
- %e – SQL 錯誤代碼
- %s – 程序開始時間戳記
- %v – 虛擬交易 ID
- %x – 交易 ID
- %c – 工作階段 ID
- %q – 非工作階段結束字元
- %a – 應用程式名稱

開啟 Aurora PostgreSQL 資料庫叢集 的查詢記錄

您可以設定下表中列出的一些參數，來收集有關資料庫活動的詳細資訊，包括查詢、等待鎖定的查詢、檢查點，以及許多其他詳細資訊。本主題著重於記錄查詢。

參數	預設	Description
log_connections	–	記錄每個成功連線。若要了解如何使用此參數搭配 log_disconnections 來偵測連線流失，請參閱 使用集區管理 Aurora PostgreSQL 連線流失。
log_disconnections	–	記錄每個工作階段的結束及其持續時間。若要了解如何使用此參數搭配 log_connections 來偵測連線流失，請參閱 使用集區管理 Aurora PostgreSQL 連線流失。
log_checkpoints	–	不適用於 Aurora PostgreSQL
log_lock_waits	–	記錄長鎖定等待。根據預設，不會設定此參數。
log_min_duration_sample	–	(毫秒) 設定執行時間下限，超出此時間就會記錄陳述式樣本。使用 log_statement_sample_rate 參數設定範例大小。
log_min_duration_statement	–	至少執行指定時間或更長時間的任何 SQL 陳述式都會被記錄下來。根據預設，不會設定此參數。開啟此參數可以協助您尋找未最佳化的查詢。
log_statement	–	設定已記錄的陳述式類型。依預設，不會設定此參數，但您可以將其變更為 all、ddl 或 mod，以指定您要記錄的 SQL 陳述式類型。如果您針對這個參數指定了 none 以外的任何值，您也應該採取額外的步驟，以防止在日誌檔中暴露密碼。如需詳細資訊，請參閱降低使用查詢記錄時密碼暴露的風險。
log_statement_sample_rate	–	超過要記錄之 log_min_duration_sample 中所指定時間的陳述式百分比，以介於 0.0 與 1.0 之間的浮點值表示。
log_statement_stats	–	將累積效能統計資訊寫入至伺服器日誌。

使用記錄來尋找執行緩慢的查詢

您可以記錄 SQL 陳述式和查詢，以協助尋找執行緩慢的查詢。您可以依照本節所述修改 `log_statement` 和 `log_min_duration` 參數中的設定來開啟此功能。在針對您的 Aurora PostgreSQL 資料庫叢集 開啟查詢記錄之前，您應該注意到日誌檔中可能暴露密碼，以及如何降低風險。如需詳細資訊，請參閱降低使用查詢記錄時密碼暴露的風險。

接下來，您可以尋找有關 `log_statement` 和 `log_min_duration` 參數的參考資訊。

log_statement

此參數指定應該傳送至日誌的 SQL 陳述式類型。預設值為 none。如果您將此參數變更為 `all`、`ddl` 或 `mod`，請務必套用建議的動作，以降低在日誌檔中暴露密碼的風險。如需詳細資訊，請參閱降低使用查詢記錄時密碼暴露的風險。

全部

記錄所有陳述式。此設定是基於偵錯用途而建議的。

DDL

記錄所有資料定義語言 (DDL) 陳述式，例如 CREATE、ALTER、DROP 等。

MOD

記錄所有 DDL 陳述式和資料操作語言 (DML) 陳述式，例如 INSERT、UPDATE 和 DELETE)，這些陳述式會修改資料。

無

不會記錄任何 SQL 陳述式。建議您使用此設定，以避免在日誌中暴露密碼的風險。

log_min_duration_statement

至少執行指定時間或更長時間的任何 SQL 陳述式都會被記錄下來。根據預設，不會設定此參數。開啟此參數可以協助您尋找未最佳化的查詢。

–1–2147483647

記錄陳述式之執行時間的毫秒數。

設定查詢記錄

這些步驟假設您的 Aurora PostgreSQL 資料庫叢集使用自訂資料庫叢集參數群組。

1. 將 log_statement 參數設為 all。下列範例顯示使用此參數設定寫入至 postgresql.log 檔案的資訊。

```
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: statement:
 SELECT feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: QUERY
 STATISTICS
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:DETAIL: ! system
 usage stats:
! 0.017355 s user, 0.000000 s system, 0.168593 s elapsed
! [0.025146 s user, 0.000000 s system total]
! 36644 kB max resident size
! 0/8 [0/8] filesystem blocks in/out
! 0/733 [0/1364] page faults/reclaims, 0 [0] swaps
! 0 [0] signals rcvd, 0/0 [0/0] messages rcvd/sent
! 19/0 [27/0] voluntary/involuntary context switches
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: SELECT
 feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:ERROR: syntax error
 at or near "ORDER" at character 1
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: ORDER BY
 s.confidence DESC;
---------------------- END OF LOG ----------------------
```

2. 設定 log_min_duration_statement 參數。下列範例說明此參數設定為 postgresql.log 時寫入至 1 檔案的資訊：

   系統會記錄超過 log_min_duration_statement 參數中所指定持續時間的查詢。下列顯示一個範例。您可以在 Amazon RDS 主控台中檢視 Aurora PostgreSQL 資料庫叢集的日誌檔。

```
2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: statement: DROP
 table comments;
2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: duration:
 167.754 ms
2022-10-05 19:08:07 UTC::@:[355]:LOG: checkpoint starting: time
2022-10-05 19:08:08 UTC::@:[355]:LOG: checkpoint complete: wrote 11 buffers
 (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write=1.013 s, sync=0.006 s,
 total=1.033 s; sync files=8, longest=0.004 s, average=0.001 s; distance=131028 kB,
 estimate=131028 kB
```

```
---------------------- END OF LOG ----------------------
```

降低使用查詢記錄時密碼暴露的風險

建議您保持 `log_statement` 設定為 `none` 以避免密碼暴露。如果您將 `log_statement` 設定為 `all`、`ddl` 或 `mod`，建議您採取下列一或多個步驟。

- 對於用戶端，請加密敏感資訊。如需的詳細資訊，請參閱 PostgreSQL 文件中的加密選項。使用 `CREATE` 和 `ALTER` 陳述式的 `ENCRYPTED` (和 `UNENCRYPTED`) 選項。如需詳細資訊，請參閱 PostgreSQL 文件中的 CREATE USER。
- 對於您的 Aurora 資料庫叢集，請設定並使用 PostgreSQL 稽核 (pgAudit) 擴充功能。此擴充功能會刪減傳送至日誌的 CREATE 和 ALTER 陳述式中的敏感資訊。如需詳細資訊，請參閱使用 PgAudit 記錄資料庫活動。
- 限制對 CloudWatch 日誌的存取。
- 使用更強大的身分驗證機制，例如 IAM。

在 AWS CloudTrail 中監控 Amazon Aurora API 呼叫

AWS CloudTrail 是一項可協助您稽核 AWS 帳戶的 AWS 服務。AWS CloudTrail 會在您建立 AWS 帳戶時開啟。如需有關 CloudTrail 的相關資訊，請參閱 AWS CloudTrail 使用者指南。

主題

- CloudTrail 與 Amazon Aurora 整合
- Amazon Aurora 日誌檔案項目

CloudTrail 與 Amazon Aurora 整合

CloudTrail 會記錄所有 Amazon Aurora 動作。對於使用者、角色或 AWS 服務在 Amazon Aurora 中採取的動作，CloudTrail 會提供記錄。

CloudTrail 事件

CloudTrail 會擷取 Amazon Aurora 的 API 呼叫當作事件。一個事件為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、請求參數等資訊。事件包括從 Amazon RDS 主控台的呼叫，以及對 Amazon RDS API 操作的程式碼呼叫。

Amazon Aurora 活動會記錄在 Event history (事件歷史記錄) 的 CloudTrail 事件中。您可以使用 CloudTrail 主控台檢視 AWS 區域中過去 90 天所記錄的 API 活動和事件。如需詳細資訊，請參閱使用 CloudTrail 事件歷程記錄檢視事件。

CloudTrail 線索

如需 AWS 帳戶中正在記錄事件 (包括 Amazon Aurora 的事件)，請建立 trail (追蹤)。權杖是一種組態，能讓事件交付到指定的 Amazon S3 儲存貯體。CloudTrail 通常會在帳戶活動的 15 分鐘內交付日誌檔案。

> **Note**
>
> 即使您未設定權仗，依然可以透過 CloudTrail 主控台中的 Event history (事件歷史記錄) 檢視最新事件。

您可以為 AWS 帳戶建立兩種類型的追蹤：套用至所有區域的追蹤，或套用至一個區域的追蹤。根據預設，當您在主控台建立線索時，線索會套用到所有 區域。

此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中所收集的事件資料。如需詳細資訊，請參閱：

- 建立追蹤的概觀
- CloudTrail 支援的服務和整合
- 設定 CloudTrail 的 Amazon SNS 通知
- 接收多個區域的 CloudTrail 日誌檔案及接收多個帳戶的 CloudTrail 日誌檔案

Amazon Aurora 日誌檔案項目

CloudTrail 日誌檔案包含一或多個日誌專案。CloudTrail 日誌檔案並非依公有 API 呼叫追蹤記錄的堆疊排序，因此不會以任何特定順序出現。

以下範例顯示的是展示 `CreateDBInstance` 動作的 CloudTrail 日誌項目。

```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2018-07-30T22:14:06Z",
    "eventSource": "rds.amazonaws.com",
    "eventName": "CreateDBInstance",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.15.42 Python/3.6.1 Darwin/17.7.0 botocore/1.10.42",
    "requestParameters": {
        "enableCloudwatchLogsExports": [
            "audit",
            "error",
            "general",
            "slowquery"
        ],
        "dBInstanceIdentifier": "test-instance",
        "engine": "mysql",
```

```
        "masterUsername": "myawsuser",
        "allocatedStorage": 20,
        "dBInstanceClass": "db.m1.small",
        "masterUserPassword": "****"
    },
    "responseElements": {
        "dBInstanceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance",
        "storageEncrypted": false,
        "preferredBackupWindow": "10:27-10:57",
        "preferredMaintenanceWindow": "sat:05:47-sat:06:17",
        "backupRetentionPeriod": 1,
        "allocatedStorage": 20,
        "storageType": "standard",
        "engineVersion": "8.0.28",
        "dbInstancePort": 0,
        "optionGroupMemberships": [
            {
                "status": "in-sync",
                "optionGroupName": "default:mysql-8-0"
            }
        ],
        "dBParameterGroups": [
            {
                "dBParameterGroupName": "default.mysql8.0",
                "parameterApplyStatus": "in-sync"
            }
        ],
        "monitoringInterval": 0,
        "dBInstanceClass": "db.m1.small",
        "readReplicaDBInstanceIdentifiers": [],
        "dBSubnetGroup": {
            "dBSubnetGroupName": "default",
            "dBSubnetGroupDescription": "default",
            "subnets": [
                {
                    "subnetAvailabilityZone": {"name": "us-east-1b"},
                    "subnetIdentifier": "subnet-cbfff283",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1e"},
                    "subnetIdentifier": "subnet-d7c825e8",
                    "subnetStatus": "Active"
                },
```

```
                {
                    "subnetAvailabilityZone": {"name": "us-east-1f"},
                    "subnetIdentifier": "subnet-6746046b",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1c"},
                    "subnetIdentifier": "subnet-bac383e0",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1d"},
                    "subnetIdentifier": "subnet-42599426",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1a"},
                    "subnetIdentifier": "subnet-da327bf6",
                    "subnetStatus": "Active"
                }
            ],
            "vpcId": "vpc-136a4c6a",
            "subnetGroupStatus": "Complete"
        },
        "masterUsername": "myawsuser",
        "multiAZ": false,
        "autoMinorVersionUpgrade": true,
        "engine": "mysql",
        "cACertificateIdentifier": "rds-ca-2015",
        "dbiResourceId": "db-ETDZIIXHEWY5N7GXVC4SH7H5IA",
        "dBSecurityGroups": [],
        "pendingModifiedValues": {
            "masterUserPassword": "****",
            "pendingCloudwatchLogsExports": {
                "logTypesToEnable": [
                    "audit",
                    "error",
                    "general",
                    "slowquery"
                ]
            }
        },
        "dBInstanceStatus": "creating",
        "publiclyAccessible": true,
```

```
        "domainMemberships": [],
        "copyTagsToSnapshot": false,
        "dBInstanceIdentifier": "test-instance",
        "licenseModel": "general-public-license",
        "iAMDatabaseAuthenticationEnabled": false,
        "performanceInsightsEnabled": false,
        "vpcSecurityGroups": [
            {
                "status": "active",
                "vpcSecurityGroupId": "sg-f839b688"
            }
        ]
    },
    "requestID": "daf2e3f5-96a3-4df7-a026-863f96db793e",
    "eventID": "797163d3-5726-441d-80a7-6eeb7464acd4",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

如上述範例中的 `userIdentity` 元素所示，每個事件或記錄項目都包含產生要求者的相關資訊。身分資訊可協助您判斷下列事項：

- 該請求是否使用根或 IAM 使用者憑證提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 該請求是否由另一項 AWS 服務提出。

如需 `userIdentity` 的詳細資訊，請參閱 CloudTrail userIdentity 元素。如需有關 `CreateDBInstance` 和其他 Amazon Aurora 動作的詳細資訊，請參閱《Amazon RDS API 參考》。

使用資料庫活動串流來監控 Amazon Aurora

透過使用資料庫活動串流，您就可以監控資料庫活動的近乎即時的串流。

主題

- 資料庫活動串流概觀
- Aurora MySQL 資料庫活動串流的聯網先決條件
- 開始資料庫活動串流
- 取得資料庫活動串流的狀態
- 停用資料庫活動串流
- 監控資料庫活動串流
- 資料庫活動串流的 IAM 政策範例

資料庫活動串流概觀

作為 Amazon Aurora 資料庫管理員，您需要保護資料庫並遵守合規與法規要求。其中一項策略是整合資料庫活動串流與監控工具。透過此方式，您可以監控 Amazon Aurora 叢集中的稽核活動，並設定警示。

外部和內部都有安全威脅。若要防範內部威脅，您可以設定資料庫活動串流功能來控制管理員對資料串流的存取。 資料庫管理員沒有存取收集、傳輸、儲存和處理串流的權限。

內容

- 資料庫活動串流運作方式
- 資料庫活動串流的非同步和同步模式
- 資料庫活動串流的要求與限制
- 區域和版本可用性
- 支援資料庫活動串流的資料庫執行個體類別

資料庫活動串流運作方式

於 Amazon Aurora 中，您可在叢集層級啟動資料庫活動串流。叢集中的所有資料庫執行個體皆啟用了資料庫活動串流。

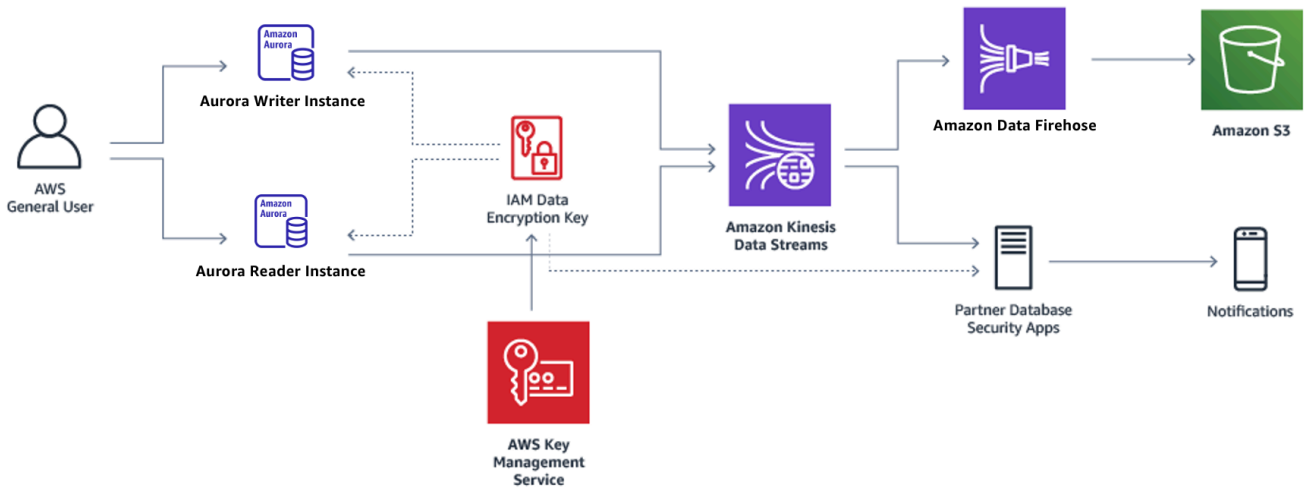
您的 Aurora 資料庫叢集 會近乎即時地將活動推送至 Amazon Kinesis 資料串流。系統會自動建立 Kinesis 串流。從 Kinesis，您可以設定 Amazon Data Firehose 和 等 AWS 服務 AWS Lambda ，以取用串流並存放資料。

> **Important**
>
> 在 Amazon Aurora 中使用資料庫活動串流功能是一項免費功能，但 Amazon Kinesis 會收取資料串流費用。如需詳細資訊，請參閱 Amazon Kinesis Data Streams 定價。

若您使用 Aurora 全域資料庫，請在每個資料庫叢集上分別啟動資料庫活動串流。每個叢集在其自己的 AWS 區域中將稽核資料傳送至其自己的 Kinesis 串流。在容錯移轉期間，活動串流的作業方式不會有所不同。其繼續像往常一樣稽核您的全域資料庫。

您可為合規管理設定應用程式以使用資料庫活動串流。若為 Aurora PostgreSQL，合規應用程式包括 IBM 的 Security Guardium 和 Imperva 的 SecureSphere Database Audit and Protection。這些應用程式可以使用串流來產生警示，並稽核您的 Aurora 資料庫叢集。

下圖展示使用 Amazon Data Firehose 設定的 Aurora 資料庫叢集。



資料庫活動串流的非同步和同步模式

您可以選擇讓資料庫工作階段使用以下任一模式來處理資料庫活動事件：

- 非同步模式 – 在資料庫工作階段產生活動串流事件時，工作階段會立即傳回正常的活動。系統會在背景中將活動串流事件變成耐久的記錄。如果背景任務中發生錯誤，就會傳送 RDS 事件。此事件會指出活動串流事件記錄可能遺失的任何時段的開頭和結尾。

  非同步模式對資料庫效能的幫助較大，對活動串流精準度的幫助較小。

  > **Note**
  >
  > 非同步模式適用於 Aurora PostgreSQL 和 Aurora MySQL。

- 同步模式 – 在資料庫工作階段產生活動串流事件時，在該事件變得耐久前工作階段都會封鎖。如果事件因故而無法變得耐久，資料庫工作階段會傳回正常的活動。然而，系統會傳送 RDS 事件，指出活動串流記錄可能已遺失一段時間。在系統回到運作良好的狀態後，就會傳送第二個 RDS 事件。

  同步模式對活動串流精準度的幫助較大，對資料庫效能的幫助較小。

  > **Note**
  >
  > 同步模式可用於 Aurora PostgreSQL。您不能搭配 Aurora MySQL 使用同步模式。

資料庫活動串流的要求與限制

在 Aurora 中，資料庫活動串流具有以下要求和限制：

- 資料庫活動串流需要使用 Amazon Kinesis。
- AWS Key Management Service 資料庫活動串流需要 (AWS KMS)，因為它們一律會加密。
- 將其他加密套用至 Amazon Kinesis 資料串流與已使用 AWS KMS 金鑰加密的資料庫活動串流不相容。
- 在資料庫叢集層級啟動資料庫活動串流。若您將資料庫執行個體新增至叢集，則無需在執行個體上啟動活動串流：其會自動進行稽核。
- 在 Aurora 全域資料庫中，請確保個別在每個資料庫叢集上啟動活動串流。每個叢集在其自己的 AWS 區域中將稽核資料傳送至其自己的 Kinesis 串流。
- 在 Aurora PostgreSQL 中，請務必在主要版本升級之前停止資料庫活動串流。升級完成後，您可以啟動資料庫活動串流。

區域和版本可用性

功能可用性和支援會因每個 Aurora 資料庫引擎的特定版本以及 AWS 區域而有所不同。如需有關 Aurora 和資料庫活動串流版本和區域可用性的詳細資訊，請參閱 資料庫活動串流的支援區域和 Aurora 資料庫引擎。

支援資料庫活動串流的資料庫執行個體類別

針對 Aurora MySQL,您可以搭配下列資料庫執行個體類別來使用資料庫活動串流：

- db.r8g.*large
- db.r7g.*large
- db.r7i.*large
- db.r6g.*large
- db.r6i.*large
- db.r5.*large
- db.x2g.*

針對 Aurora PostgreSQL,您可以搭配下列資料庫執行個體類別來使用資料庫活動串流：

- db.r8g.*large
- db.r7i.*large
- db.r7g.*large
- db.r6g.*large
- db.r6i.*large
- db.r6id.*large
- db.r5.*large
- db.r4.*large
- db.x2g.*

Aurora MySQL 資料庫活動串流的聯網先決條件

在下文中，您可以了解如何設定 Virtual Private Cloud (VPC) 以與資料庫活動串流搭配使用。

> **Note**
>
> Aurora MySQL 網路先決條件適用於下列引擎版本：
>
> - Aurora MySQL 第 2 版，最高至 2.11.3
> - Aurora MySQL 第 2.12.0 版
> - Aurora MySQL 第 3 版，最高至 3.04.2

主題

- AWS KMS 端點的先決條件
- 公開可用性的先決條件
- 私有可用性的先決條件

AWS KMS 端點的先決條件

Aurora MySQL 叢集中使用活動串流的執行個體必須能夠存取 AWS KMS 端點。在啟用 Aurora MySQL 叢集的資料庫活動串流之前，請確定已滿足此需求。如果 Aurora 叢集可公開使用，則會自動滿足此要求。

> **Important**
>
> 如果 Aurora MySQL 資料庫叢集無法存取 AWS KMS 端點，活動串流便會停止。在這種情況下，Aurora 會使用 RDS 事件通知您有關此問題。

公開可用性的先決條件

Aurora 資料庫叢集必須符合下列條件才能公開：

- AWS 管理主控台叢集詳細資訊頁面中的 Publicly Accessible (可公開存取) 為 Yes (是)。
- 資料庫叢集位於 Amazon VPC 公有子網路中。如需可公開存取之資料庫執行個體的詳細資訊，請參閱在 VPC 中使用資料庫叢集。如需有關公有 Amazon VPC 子網路的更多資訊，請參閱您的 VPC 與子網路。

私有可用性的先決條件

如果您的 Aurora 資料庫叢集位於 VPC 公有子網路中且不可公開存取，則為私有。若要將您的叢集保持為私有狀態，並將其與資料庫活動串流一起使用，您有下列選項：

- 在 VPC 中設定網路位址轉譯 (NAT)。如需更多詳細資訊，請參閱 NAT 閘道。
- 在 VPC 中建立 AWS KMS 端點。建議使用此選項，因為設定更容易。

在 VPC 中建立 AWS KMS 端點

1. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
2. 在導覽窗格中選擇 Endpoints (端點)。
3. 選擇建立端點。

   Create Endpoint (建立端點) 頁面隨即出現。
4. 請執行下列操作：
   - 在 Service category (服務類別) 中，選擇 AWS services (服務)。
   - 在 Service Name (服務名稱) 中，選擇 com.amazonaws.***region***.kms，其中 *region* 是叢集所在的 AWS 區域。
   - 在 VPC 中，選擇叢集所在的 VPC。
5. 選擇建立端點。

如需設定 VPC 端點的詳細資訊，請參閱 VPC 端點。

開始資料庫活動串流

如要監控您 Aurora 資料庫叢集中所有執行個體的資料庫活動，請在叢集層級開始活動串流。您新增至該叢集的任何資料庫執行個體也會自動受到監控。若您使用 Aurora 全域資料庫，請在每個資料庫叢集上分別啟動資料庫活動串流。每個叢集在其自己的 AWS 區域 中將稽核資料傳送至其自己的 Kinesis 串流。

在開啟的活動串流時，您在稽核政策中設定的每個資料庫活動事件都會產生活動串流事件。`CONNECT` 和 `SELECT` 之類的 SQL 命令會產生存取事件。`CREATE` 和 `INSERT` 之類的 SQL 命令會產生變更事件。

Console

若要開始資料庫活動串流

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您要在其上啟動活動串流的資料庫叢集 。
4. 針對 Actions (動作)，選擇 Start activity stream (啟動活動串流)。

   Start database activity stream: *name* (開始資料庫活動串流：name) 視窗隨即出現，其中 *name* 是您的 資料庫叢集。
5. 輸入以下設定：

   - 對於 AWS KMS key，請從 AWS KMS keys清單中選擇一個金鑰。

     > **Note**
     >
     > 如果您的 Aurora MySQL 叢集無法存取 KMS 金鑰，請依照 Aurora MySQL 資料庫活動串流的聯網先決條件 中的指示，先啟用此類存取。

     Aurora 會使用 KMS 金鑰來加密金鑰，此金鑰會依序加密資料庫活動。請選擇預設金鑰以外的 KMS 金鑰。如需更多有關加密金鑰和 AWS KMS 的資訊，請參閱 AWS Key Management Service 開發人員指南中的什麼是 AWS Key Management Service ?。
   - 對於 Database activity stream mode (資料庫活動串流模式)，選擇 Asynchronous (非同步) 或 Synchronous (同步)。

     > **Note**
     >
     > 此選項僅適用於 Aurora PostgreSQL。針對 Aurora MySQL，您只能使用非同步模式。

   - 選擇 Immediately (立即)。

     在您選擇 Immediately (立即) 時，資料庫叢集會立即重新啟動。如果您選擇 During the next maintenance window (下個維護時段期間)，則資料庫叢集不會立即重新啟動。在這種情況下直到下一個維護時段前，資料庫活動串流都不會啟動。
6. 選擇 Start database activity stream (啟動資料庫活動串流)。

資料庫叢集的狀態會顯示活動串流正在開始。

> **Note**
>
> 如果收到錯誤訊息 `You can't start a database activity stream in this configuration`，請檢查 支援資料庫活動串流的資料庫執行個體類別 以查看資料庫叢集是否使用支援的執行個體類別。

AWS CLI

若要開始資料庫叢集的資料庫活動串流，請使用 start-activity-stream AWS CLI 命令來設定資料庫叢集。

- `--resource-arn` *arn*– 指定資料庫的 Amazon 資源名稱 (ARN) 叢集。
- `--mode` *sync-or-async* – 指定同步 (sync) 或非同步 (async) 模式。對於 Aurora PostgreSQL，您可以選擇其中一個值。針對 Aurora MySQL，請指定 `async`。
- `--kms-key-id` *key* – 指定用於加密資料庫活動串流中訊息的 KMS 金鑰識別碼。AWS KMS 金鑰識別碼是 AWS KMS key的金鑰 ARN、金鑰 ID、別名 ARN 或別名。

下列範例會在非同步模式下開始資料庫叢集的活動串流。

對於 Linux、macOS 或 Unix：

```
aws rds start-activity-stream \
    --mode async \
    --kms-key-id my-kms-key-arn \
    --resource-arn my-cluster-arn \
    --apply-immediately
```

在 Windows 中：

```
aws rds start-activity-stream ^
    --mode async ^
    --kms-key-id my-kms-key-arn ^
    --resource-arn my-cluster-arn ^
    --apply-immediately
```

Amazon RDS API

若要開始資料庫叢集的資料庫活動串流，請使用 StartActivityStream 操作來設定叢集。

使用以下參數呼叫動作：

- `Region`
- `KmsKeyId`
- `ResourceArn`
- `Mode`

> **Note**
>
> 如果您收到錯誤，指出無法使用目前版本的 Aurora PostgreSQL 資料庫啟動資料庫活動串流，請在啟動資料庫活動串流之前套用 Aurora PostgreSQL 的最新修補程式。如需升級 Aurora PostgreSQL 資料庫的詳細資訊，請參閱升級 Amazon Aurora 資料庫叢集。
> 以下是使用 Aurora PostgreSQL 啟動資料庫活動串流的最低修補程式版本。
>
> - 3.4.15 (11.9.15)、11.21.10
> - 12.9.15、12.15.9、12.16.10、12.17.7、12.18.5、12.19.4、12.20.3、12.22.3
> - 13.9.12、13.11.9、13.12.10、13.13.7、13.14.5、13.15.4、13.16.3、13.18.3
> - 14.6.12、14.8.9、14.9.10、14.10.7、14.11.5、、14.12.4、14.13.3、14.15.3
> - 15.3.9、15.4.10、15.5.7、15.6.5、、15.7.4、15.8.3、、15.10.3
> - 16.1.7、16.2.5、16.3.4、16.4.3、、16.6.3

取得資料庫活動串流的狀態

您可以使用主控台或 AWS CLI 來取得 的活動串流狀態。

主控台

取得資料庫活動串流的狀態

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)，然後選擇 DB cluster (資料庫叢集) (資料庫執行個體) 連結。

3. 選擇 Configuration (組態) 標籤，然後確認 Database activity stream (資料庫活動串流) 的狀態。

AWS CLI

您可以取得資料庫叢集的活動串流組態，作為對 describe-db-clusters CLI 請求的回應。

以下範例描述 *my-cluster*。

```
aws rds --region my-region describe-db-clusters --db-cluster-identifier my-cluster
```

JSON 回應如以下範例所示。下列欄位會顯示：

- `ActivityStreamKinesisStreamName`
- `ActivityStreamKmsKeyId`
- `ActivityStreamStatus`
- `ActivityStreamMode`
-

這些欄位對於 Aurora PostgreSQL 和 Aurora MySQL 是相同的，除非 `ActivityStreamMode` 對於 Aurora MySQL 一律是 `async`，而對於 Aurora PostgreSQL，它可能是 `sync` 或 `async`。

```
{
    "DBClusters": [
        {
      "DBClusterIdentifier": "my-cluster",
            ...
            "ActivityStreamKinesisStreamName": "aws-rds-das-cluster-
A6TSYXITZCZXJHIRVFUBZ5LTWY",
            "ActivityStreamStatus": "starting",
            "ActivityStreamKmsKeyId": "12345678-abcd-efgh-ijkl-bd041f170262",
            "ActivityStreamMode": "async",
            "DbClusterResourceId": "cluster-ABCD123456"
            ...
        }
    ]
}
```

RDS API

您可以取得資料庫叢集的活動串流組態，作為對 DescribeDBClusters 操作的回應。

停用資料庫活動串流

您可以使用主控台或 AWS CLI 來停止活動串流。

若您刪除資料庫叢集 ，則會停止活動串流，並自動刪除底層的 Amazon Kinesis 串流。

主控台

關閉活動串流

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您要停止資料庫活動串流的資料庫叢集。
4. 針對 Actions (動作)，選擇 Stop activity stream (停止活動串流)。Database Activity Stream (資料庫活動串流) 即會顯示。

   a. 選擇 Immediately (立即)。

      在您選擇 Immediately (立即) 時，資料庫叢集會立即重新啟動。如果您選擇 During the next maintenance window (下個維護時段期間)，則資料庫叢集 不會立即重新啟動。在此情況下，直到下一個維護時段前，資料庫活動串流都不會被停用。

   b. 選擇 Continue (繼續)。

AWS CLI

若要停止您資料庫叢集的資料庫活動串流，請使用 AWS CLI 命令 stop-activity-stream 來設定資料庫叢集。使用 `--region` 參數來識別資料庫叢集的 AWS 區域。`--apply-immediately` 為選用參數。

對於 Linux、macOS 或 Unix：

```
aws rds --region MY_REGION \
    stop-activity-stream \
    --resource-arn MY_CLUSTER_ARN \
    --apply-immediately
```

在 Windows 中：

```
aws rds --region MY_REGION ^
    stop-activity-stream ^
```

```
    --resource-arn MY_CLUSTER_ARN ^
    --apply-immediately
```

RDS API

若要停止您的資料庫叢集的資料庫活動串流，請使用 StopActivityStream 操作來設定叢集。使用 Region 參數來識別資料庫叢集的 AWS 區域。ApplyImmediately 為選用參數。

監控資料庫活動串流

資料庫活動串流會監控和報告活動。活動資料串流會收集並傳送至 Amazon Kinesis。從 Kinesis 中，您可以監視活動串流，或者其他服務和應用程式可以使用活動串流以供進一步分析。您可以使用 AWS CLI 命令 describe-db-clusters 或 RDS API DescribeDBClusters 操作來尋找基礎 Kinesis 串流名稱。

Aurora 會為您管理 Kinesis 串流，如下所示：

- Aurora 會自動建立保留時間為 24 小時的 Kinesis 串流。
- 如有必要，Aurora 可擴展 Kinesis 串流。
- 如果您停止資料庫活動串流或刪除資料庫叢集，則 Aurora 會刪除 Kinesis 串流。

系統會監控以下類別的活動並將其放在活動串流稽核日誌中：

- SQL 命令 – 所有 SQL 命令都經稽核，也是預備陳述式、內建函數和 PL/SQL 函數。對預存程序的呼叫會進行稽核。在儲存的程序或函數中發出的任何 SQL 語句也被稽核。
- 其他資料庫資訊 – 監控的活動包含完整的 SQL 陳述式、因 DML 命令而受影響的資料列數、經存取的物件和唯一的資料庫名稱。若為 Aurora PostgreSQL，資料庫活動串流還會監控連結變數和已儲存的程序參數。

> **Important**
>
> 活動資料串流稽核記錄中會顯示每個陳述式的完整 SQL 文字，包括任何敏感資料。但是，如果 Aurora 可以從內容 (例如下列 SQL 陳述式) 判斷資料庫使用者密碼，則該密碼將會被修訂。
>
> ```
> ALTER ROLE role-name WITH password
> ```

- 連線資訊 – 監控的活動包含工作階段和網路資訊、伺服器程序 ID 和結束代碼。

如果活動串流在監控資料庫執行個體時失敗，系統會透過 RDS 事件來通知您。

在下列各節中，您可以存取、稽核和處理資料庫活動串流。

主題

- 透過 Amazon Kinesis 存取活動串流
- 資料庫活動串流的稽核日誌內容和範例
- 適用於資料庫活動串流的 databaseActivityEventList JSON 陣列
- 使用 AWS SDK 來處理資料庫活動串流

透過 Amazon Kinesis 存取活動串流

在您啟用資料庫叢集的活動串流時，系統就會為您建立 Kinesis 串流。透過 Kinesis，您就可以即時監控資料庫活動。若要進一步分析資料庫活動，您可以將 Kinesis 串流連線至消費者應用程式。您也可以將串流連線至合規性管理應用程式，例如 IBM 的 Security Guardium 或 Imperva 的 SecureSphere Database Audit and Protection。

您可以從 RDS 主控台或 Kinesis 主控台存取 Kinesis 串流。

使用 RDS 主控台從 Kinesis 存取活動串流

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇已在其上啟動活動串流的資料庫叢集。
4. 選擇 Configuration (組態)。
5. 在 Database activity stream (資料庫活動串流) 下，選擇 Kinesis stream (Kinesis 串流) 下的連結。
6. 在 Kinesis 主控台中，選擇 Monitoring (監控) 來開始觀察資料庫活動。

使用 Kinesis 主控台從 Kinesis 存取活動串流

1. 在 https://console.aws.amazon.com/kinesis 上開啟 Kinesis 主控台。
2. 從 Kinesis 串流清單中選擇活動串流。

   活動串流的名稱包含字首 `aws-rds-das-cluster-`，後接資料庫叢集的資源 ID。以下是範例。

   ```
   aws-rds-das-cluster-NHVOV4PCLWHGF52NP
   ```

若要使用 Amazon RDS 主控台來尋找資料庫叢集的資源 ID，請從資料庫清單中選擇您的資料庫叢集，然後選擇Configuration (組態) 索引標籤。

若要使用 AWS CLI 來尋找活動串流的完整 Kinesis 串流名稱，請使用 describe-db-clusters CLI 請求，並記下回應中的 `ActivityStreamKinesisStreamName` 值。

3. 選擇 Monitoring (監控) 來開始觀察資料庫活動。

如需使用 Amazon Kinesis 的詳細資訊，請參閱什麼是 Amazon Kinesis Data Streams？。

資料庫活動串流的稽核日誌內容和範例

受監控的事件會以 JSON 字串的形式在資料庫活動串流中顯示。此結構包含 JSON 物件，內含的 `DatabaseActivityMonitoringRecord` 會依序包含活動事件的 `databaseActivityEventList` 陣列。

> **Note**
>
> 對於資料庫活動串流，`paramList` JSON 陣列不包含休眠應用程式的 null 值。

主題

- 活動串流的稽核記錄範例
- DatabaseActivityMonitoringRecords JSON 物件
- databaseActivityEvents JSON 物件

活動串流的稽核記錄範例

以下是活動事件記錄的範例解密 JSON 稽核日誌。

Example Aurora PostgreSQL CONNECT SQL 陳述式的活動事件記錄

以下活動事件記錄顯示 psql 用戶端 (`clientApplication`) 使用 `CONNECT` SQL 陳述式 (`command`) 登入。

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
```

```
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-10-30 00:39:49.940668+00",
          "logTime": "2019-10-30 00:39:49.990579+00",
          "statementId": 1,
          "substatementId": 1,
          "objectType": null,
          "command": "CONNECT",
          "objectName": null,
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "49804",
          "sessionId": "5ce5f7f0.474b",
          "rowCount": null,
          "commandText": null,
          "paramList": [],
          "pid": 18251,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "MISC",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```

Example Aurora MySQL CONNECT SQL 陳述式的活動事件記錄

以下是 mysql 用戶端 (`clientApplication`) 使用 `CONNECT` SQL 陳述式 (`command`) 登入的活動事件記錄。

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:13.267214+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"rdsadmin",
      "databaseName":"",
      "remoteHost":"localhost",
      "remotePort":"11053",
      "command":"CONNECT",
      "commandText":"",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"",
      "statementId":0,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"725121",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:13.267207+00",
      "endTime":"2020-05-22 18:07:13.267213+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

Example Aurora PostgreSQL CREATE TABLE 陳述式的活動事件記錄

以下是 Aurora PostgreSQL 的 `CREATE TABLE` 事件範例。

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-05-24 00:36:54.403455+00",
          "logTime": "2019-05-24 00:36:54.494235+00",
          "statementId": 2,
          "substatementId": 1,
          "objectType": null,
          "command": "CREATE TABLE",
          "objectName": null,
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "34534",
          "sessionId": "5ce73c6f.7e64",
          "rowCount": null,
          "commandText": "create table my_table (id serial primary key, name
 varchar(32));",
          "paramList": [],
          "pid": 32356,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "DDL",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```

Example Aurora MySQL CREATE TABLE 陳述式的活動事件記錄

以下範例顯示 Aurora MySQL 的 CREATE TABLE 陳述式。該操作會以兩個不同的事件記錄表示。一個活動具有 "class":"MAIN"。另一個活動具有 "class":"AUX"。這些訊息可能以任何順序到達。logTime 事件的 MAIN 欄位永遠早於任何對應 logTime 事件的 AUX 欄位。

下列範例會顯示 class 值為 MAIN 的事件。


```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:12.250221+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"QUERY",
      "commandText":"CREATE TABLE test1 (id INT)",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65459278,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"725118",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:12.226384+00",
      "endTime":"2020-05-22 18:07:12.250222+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
```

```
    }
  ]
}
```

下列範例會顯示 `class` 值為 AUX 的對應事件。

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:12.247182+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"CREATE",
      "commandText":"test1",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65459278,
      "substatementId":2,
      "exitCode":"",
      "sessionId":"725118",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:12.226384+00",
      "endTime":"2020-05-22 18:07:12.247182+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"AUX"
    }
  ]
```

```
}
```

Example Aurora PostgreSQLSELECT 陳述式的活動事件記錄

下列範例顯示 的 SELECT 事件。

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-05-24 00:39:49.920564+00",
          "logTime": "2019-05-24 00:39:49.940668+00",
          "statementId": 6,
          "substatementId": 1,
          "objectType": "TABLE",
          "command": "SELECT",
          "objectName": "public.my_table",
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "34534",
          "sessionId": "5ce73c6f.7e64",
          "rowCount": 10,
          "commandText": "select * from my_table;",
          "paramList": [],
          "pid": 32356,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "READ",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
```

```
      ]
    },
   "key":"decryption-key"
}
```


```
{
    "type": "DatabaseActivityMonitoringRecord",
    "clusterId": "",
    "instanceId": "db-4JCWQLUZVFYP7DIWP6JVQ77O3Q",
    "databaseActivityEventList": [
        {
            "class": "TABLE",
            "clientApplication": "Microsoft SQL Server Management Studio - Query",
            "command": "SELECT",
            "commandText": "select * from [testDB].[dbo].[TestTable]",
            "databaseName": "testDB",
            "dbProtocol": "SQLSERVER",
            "dbUserName": "test",
            "endTime": null,
            "errorMessage": null,
            "exitCode": 1,
            "logTime": "2022-10-06 21:24:59.9422268+00",
            "netProtocol": null,
            "objectName": "TestTable",
            "objectType": "TABLE",
            "paramList": null,
            "pid": null,
            "remoteHost": "local machine",
            "remotePort": null,
            "rowCount": 0,
            "serverHost": "172.31.30.159",
            "serverType": "SQLSERVER",
            "serverVersion": "15.00.4073.23.v1.R1",
            "serviceName": "sqlserver-ee",
            "sessionId": 62,
            "startTime": null,
            "statementId": "0x03baed90412f564fad640ebe51f89b99",
            "substatementId": 1,
            "transactionId": "4532935",
            "type": "record",
            "engineNativeAuditFields": {
                "target_database_principal_id": 0,
                "target_server_principal_id": 0,
```

```
                "target_database_principal_name": "",
                "server_principal_id": 2,
                "user_defined_information": "",
                "response_rows": 0,
                "database_principal_name": "dbo",
                "target_server_principal_name": "",
                "schema_name": "dbo",
                "is_column_permission": true,
                "object_id": 581577110,
                "server_instance_name": "EC2AMAZ-NFUJJNO",
                "target_server_principal_sid": null,
                "additional_information": "",
                "duration_milliseconds": 0,
                "permission_bitmask": "0x00000000000000000000000000000001",
                "data_sensitivity_information": "",
                "session_server_principal_name": "test",
                "connection_id": "AD3A5084-FB83-45C1-8334-E923459A8109",
                "audit_schema_version": 1,
                "database_principal_id": 1,
                "server_principal_sid":
 "0x010500000000000515000000bdc2795e2d0717901ba6998cf4010000",
                "user_defined_event_id": 0,
                "host_name": "EC2AMAZ-NFUJJNO"
            }
        }
    ]
}
```

Example Aurora MySQL SELECT 陳述式的活動事件記錄

下列範例顯示 SELECT 事件。

下列範例會顯示 `class` 值為 `MAIN` 的事件。

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:29:57.986467+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
```

```
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"QUERY",
      "commandText":"SELECT * FROM test1 WHERE id < 28",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65469218,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"726571",
      "rowCount":2,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:29:57.986364+00",
      "endTime":"2020-05-22 18:29:57.986467+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

下列範例會顯示 class 值為 AUX 的對應事件。

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:29:57.986399+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
```

```
      "remotePort":"11054",
      "command":"READ",
      "commandText":"test1",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65469218,
      "substatementId":2,
      "exitCode":"",
      "sessionId":"726571",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:29:57.986364+00",
      "endTime":"2020-05-22 18:29:57.986399+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"AUX"
    }
  ]
}
```

DatabaseActivityMonitoringRecords JSON 物件

資料庫活動事件記錄位於 JSON 物件中，其中包含下列資訊。

JSON 欄位	資料類型	描述
`type`	string	JSON 記錄類型。值為 `DatabaseActivityMonitoringRecords` 。
`version`	string	資料庫活動監控記錄的版本。 產生的資料庫活動記錄版本取決於資料庫叢集的引擎版本：

JSON 欄位	資料類型	描述
		• 針對執行引擎 10.10 版及更新版本的次要版本和引擎 11.5 版及更新版本的 Aurora PostgreSQL 資料庫叢集，會產生 1.1 版資料庫活動記錄。 • 針對執行引擎 10.7 和 11.4 版的 Aurora PostgreSQL 資料庫叢集，會產生 1.0 版資料庫活動記錄。 除非特別註明，否則下列所有欄位都在 1.0 版和 1.1 版中。
databaseActivityEvents	string	包含活動事件的 JSON 物件。
金鑰	string	您用來解密 databaseActivityEventList JSON 陣列 的加密金鑰

databaseActivityEvents JSON 物件

`databaseActivityEvents` JSON 物件包含以下資訊。

JSON 記錄中的最上層欄位

稽核記錄檔中的每個事件都會包裝在 JSON 格式的記錄中。此記錄包含下列欄位。

type

此欄位永遠具有值 `DatabaseActivityMonitoringRecords`。

version

此欄位代表資料庫活動串流資料通訊協定或合約的版本。其會定義哪些欄位可用。

1.0 版代表 Aurora PostgreSQL 10.7 和 11.4 版的原始資料活動串流支援。1.1 版代表 Aurora PostgreSQL 10.10 及更高版本和 Aurora PostgreSQL 11.5 及更高版本的資料活動串流支援。1.1 版包含其他欄位 `errorMessage` 和 `startTime`。1.2 版代表 Aurora MySQL 2.08 及更高版本的資料活動串流支援。1.2 版包含其他欄位 `endTime` 和 `transactionId`。

databaseActivityEvents

代表一或多個活動事件的加密字串。它被表示為一個 base64 位元組陣列。在您解密字串時，結果會是 JSON 格式的記錄，其中包含欄位，如本節範例所示。

金鑰

用來加密 databaseActivityEvents 字串的加密資料金鑰。這與您啟動資料庫活動串流時 AWS KMS key 提供的相同。

下列範例顯示此記錄的格式。

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":"encrypted audit records",
  "key":"encrypted key"
}
```

採取下列步驟來解密 databaseActivityEvents 欄位的內容：

1. 使用您在啟動資料庫活動串流時提供的 KMS 金鑰，解密 key JSON 欄位中的值。這麼做會以純文字傳回資料加密金鑰。
2. Base64 解碼 databaseActivityEvents JSON 欄位中的值，以取得稽核承載的二進位格式的加密文字。
3. 使用您在第一個步驟中解碼的資料加密金鑰來解密二進位密文。
4. 解壓縮已解密的承載。
   - 加密的承載在 databaseActivityEvents 欄位。
   - 該 databaseActivityEventList 欄位包含稽核記錄的陣列。陣列中的 type 欄位可以是 record 或 heartbeat。

稽核日誌活動事件記錄是包含以下資訊的 JSON 物件。

JSON 欄位	資料類型	描述
type	string	JSON 記錄類型。值為 DatabaseActivityMonitoringRecord 。

JSON 欄位	資料類型	描述
clusterId	string	資料庫叢集資源識別符。其對應於資料庫叢集屬性 DbClusterResourceId 。
instanceId	string	資料庫執行個體資源識別符。它對應於資料庫執行個體屬性 DbiResourceId 。
databaseActivityEventList JSON 陣列	string	活動稽核記錄或活動訊號訊息的陣列。

適用於資料庫活動串流的 databaseActivityEventList JSON 陣列

稽核日誌承載是加密的 databaseActivityEventList JSON 陣列。以下資料表列出稽核記錄中已解密 DatabaseActivityEventList 陣列中，每個活動事件的欄位 (按英文字母順序列出)。這些欄位會根據您使用 Aurora PostgreSQL 或 Aurora MySQL 而有所不同。請參閱適用於資料庫引擎的表格。

> **Important**
>
> 事件結構可能會改變。Aurora 可能會在未來將新的欄位新增至活動事件。在剖析 JSON 資料的應用程式中，請確定程式碼可以忽略，或針對未知欄位名稱採取適當的動作。

Aurora PostgreSQL 的 databaseActivityEventList 欄位

以下是 Aurora PostgreSQL 的 databaseActivityEventList 欄位。

欄位	資料類型	描述
class	string	活動事件的類別。Aurora PostgreSQL 的有效值如下： • ALL • CONNECT – 連線或中斷連線的事件。 • DDL – DDL 陳述式，不包含在 ROLE 類別的陳述式清單中。 • FUNCTION– 函數呼叫或 DO 區塊。

欄位	資料類型	描述
		• MISC – DISCARD、FETCH、CHECKPOINT 或 VACUUM 之類的其他命令。 • NONE • READ – 來源為關聯或查詢時 SELECT 或 COPY 陳述式。 • ROLE – 與包含 GRANT、REVOKE 和 CREATE/ALTER/DROP ROLE 之角色和權限相關的陳述式。 • WRITE – 目的地為關聯時，INSERT、UPDATE、DELETE、TRUNCATE 或 COPY 陳述式。
clientApplication	string	用戶端報告用來連接的應用程式。用戶端不需提供此資訊，因此此值可以是 Null。
command	string	SQL 命令名稱，其中不含任何命令詳細資訊。
commandText	string	使用者傳遞的實際 SQL 陳述式。針對 Aurora PostgreSQL，該值與原始 SQL 陳述式相同。此欄位可用於連接或中斷連接記錄以外的所有記錄類型，在前述兩種例外類型中值為 Null。 **Important** 活動資料串流稽核記錄中會顯示每個陳述式的完整 SQL 文字，包括任何敏感資料。但是，如果 Aurora 可以從內容 (例如下列 SQL 陳述式) 判斷資料庫使用者密碼，則會編寫密文。 `ALTER ROLE role-name WITH password`
databaseName	string	使用者連接的資料庫。
dbProtocol	string	資料庫通訊協定，例如 Postgres 3.0。
dbUserName	string	用戶端驗證所用的資料庫使用者。

欄位	資料類型	描述
errorMessage (僅限 1.1 版資料庫活動記錄)	string	如果有任何錯誤，該欄位會填入由資料庫伺服器產生的錯誤訊息。對於未導致錯誤的正常陳述式，此 errorMessage 值為 null。 錯誤是定義為任何會產生用戶端可見 PostgreSQL 錯誤日誌事件的活動，其嚴重性層級為 ERROR 或更高。如需詳細資訊，請參閱 PostgreSQL 訊息嚴重性等級。例如，語法錯誤和查詢取消會產生錯誤訊息。 內部 PostgreSQL 伺服器錯誤，例如背景檢查點指標處理程序錯誤，並不會產生錯誤訊息。不過，無論日誌嚴重性層級的設定為何，仍會發出這類事件的記錄。這可以防止攻擊者關閉記錄日誌以嘗試避免偵測。 另請參閱 exitCode 欄位。
exitCode	int	工作階段結束記錄所用的值。清除結束時，此會包含結束代碼。在某些失敗狀況下可能無法隨時取得結束代碼。範例為 PostgreSQL 執行 exit() 或運算子執行 kill -9 之類的命令。 如果發生任何錯誤，此 exitCode 欄位會顯示 SQL 錯誤代碼 SQLSTATE，如 PostgreSQL 錯誤代碼中所列。 另請參閱 errorMessage 欄位。
logTime	string	在稽核程式碼路徑中記錄的時間戳記。這代表 SQL 陳述式執行結束時間。另請參閱 startTime 欄位。
netProtocol	string	網路通訊協定。
objectName	string	SQL 陳述式在其中操作的資料庫物件名稱。僅在 SQL 陳述式在資料庫物件上操作時才會使用此欄位。如果 SQL 陳述式沒有在物件上操作，則此值為 Null。

欄位	資料類型	描述
objectType	string	資料表、索引、檢視等之類的資料庫物件類型。僅在 SQL 陳述式在資料庫物件上操作時才會使用此欄位。如果 SQL 陳述式沒有在物件上操作，則此值為 Null。有效值包括以下項目： • COMPOSITE TYPE • FOREIGN TABLE • FUNCTION • INDEX • MATERIALIZED VIEW • SEQUENCE • TABLE • TOAST TABLE • VIEW • UNKNOWN
paramList	string	傳遞至 SQL 陳述式的參數陣列 (以逗號分隔)。如果 SQL 陳述式沒有任何參數，此值會是空的陣列。
pid	int	後端程序的程序 ID，此程序的配置是用來提供用戶端連線。
remoteHost	string	用戶端 IP 地址或主機名稱。針對 Aurora PostgreSQL，使用哪一個取決於資料庫的 log_hostname 參數設定。此 remoteHost 值也包含 [local]，且 localhost 表示來自 rdsadmin 使用者的活動。
remotePort	string	用戶端連接埠號碼。
rowCount	int	SQL 陳述式影響或擷取的資料列數。僅會對資料處理語言 (DML) 陳述式的 SQL 陳述式使用此欄位。如果 SQL 陳述式不是 DML 陳述式，則此值為 Null。
serverHost	string	資料庫伺服器主機 IP 地址。此 serverHost 值也包含 [local]，且 localhost 表示來自 rdsadmin 使用者的活動。

欄位	資料類型	描述
serverType	string	資料庫伺服器類型，例如 PostgreSQL 。
serverVersion	string	資料庫伺服器版本，例如 Aurora PostgreSQL 的 2.3.1。
serviceName	string	服務名稱，例如 Amazon Aurora PostgreSQL-Compatible edition 。
sessionId	int	虛擬唯一的工作階段識別符。
sessionId	int	虛擬唯一的工作階段識別符。
startTime (僅限 1.1 版資料庫活動記錄)	string	SQL 陳述式開始執行的時間。 若要計算 SQL 陳述式大約的執行時間，請使用 logTime - startTime 。另請參閱 logTime 欄位。
statementId	int	用戶端 SQL 陳述式的識別符。記數器是使用工作階段層級，且會隨著用戶端輸入的每個 SQL 陳述式遞增。
substatementId	int	SQL 子陳述式的識別符。此值會計算由 statementId 欄位識別的每個 SQL 陳述式包含的子陳述式。
type	string	事件類型。有效值為 record 或 heartbeat 。

Aurora MySQL 的 databaseActivityEventList 欄位

以下是 Aurora MySQL 的 databaseActivityEventList 欄位。

欄位	資料類型	描述
class	string	活動事件的類別。 Aurora MySQL 的有效值如下： • MAIN – 代表 SQL 陳述式的主要事件。

欄位	資料類型	描述
		• AUX – 包含其他詳細資訊的補充事件。例如，重新命名物件的陳述式可能具有反映新名稱的類別 AUX 的事件。 若要尋找 MAIN 與 AUX 相同陳述式相對應的事件，請檢查 pid 欄位和 statementId 欄位值相同的不同事件。
clientApplication	string	用戶端報告用來連接的應用程式。用戶端不需提供此資訊，因此此值可以是 Null。
command	string	SQL 陳述式的一般類別。此欄位的值取決於 class 的值。 當 class 為 MAIN 時的值包括以下內容： • CONNECT – 用戶端工作階段已連線時。 • QUERY – SQL 陳述式。伴隨著一個或多個 class 的值為 AUX 的事件。 • DISCONNECT – 用戶端工作階段中斷連線時。 • FAILED_CONNECT – 當用戶端嘗試連線但無法連線時。 • CHANGEUSER – 屬於 MySQL 網路通訊協定的一部分狀態變更，而不是來自您發出的陳述式。 當 class 為 AUX 時的值包括以下內容： • READ – 來源為關聯或查詢時 SELECT 或 COPY 陳述式。 • WRITE – 目的地為關聯時，INSERT、UPDATE、DELETE、TRUNCATE 或 COPY 陳述式。 • DROP – 刪除物件。 • CREATE – 建立物件。 • RENAME – 重新命名物件。 • ALTER – 檢視物件的屬性。

欄位	資料類型	描述
commandText	string	對於 class 值為 MAIN 的事件，此欄位代表使用者傳入的實際 SQL 陳述式。此欄位可用於連接或中斷連接記錄以外的所有記錄類型，在前述兩種例外類型中值為 Null。 對於 class 值為 AUX 的事件，此欄位包含有關事件涉及之物件的補充資訊。 針對 Aurora MySQL，引號之類的字元前面加上反斜線，代表逸出字元。 **Important** 稽核記錄中會顯示每個陳述式的完整 SQL 文字，包括任何敏感資料。但是，如果 Aurora 可以從內容 (例如下列 SQL 陳述式) 判斷資料庫使用者密碼，則會編寫密文。 `mysql> SET PASSWORD = 'my-password ';` **Note** 指定此處所顯示提示以外的密碼，作為安全最佳實務。
databaseName	string	使用者連接的資料庫。
dbProtocol	string	資料庫通訊協定。目前，Aurora MySQL 的這個值永遠是 MySQL。
dbUserName	string	用戶端驗證所用的資料庫使用者。

欄位	資料類型	描述
endTime (僅限 1.2 版資料庫活動記錄)	string	SQL 陳述式結束執行的時間。以國際標準時間 (UTC) 格式表示。 若要計算 SQL 陳述式的執行時間，請使用 endTime - startTime 。另請參閱 startTime 欄位。
errorMessage (僅限 1.1 版資料庫活動記錄)	string	如果有任何錯誤，該欄位會填入由資料庫伺服器產生的錯誤訊息。對於未導致錯誤的正常陳述式，此 errorMessage 值為 null。 錯誤是定義為任何會產生用戶端可見 MySQL 錯誤日誌事件的活動，其嚴重性層級為 ERROR 或更高。如需更多資訊，請參閱 MySQL 參考手冊中的錯誤記錄。例如，語法錯誤和查詢取消會產生錯誤訊息。 內部 MySQL 伺服器錯誤，例如背景檢查點指標處理程序錯誤，並不會產生錯誤訊息。不過，無論日誌嚴重性層級的設定為何，仍會發出這類事件的記錄。這可以防止攻擊者關閉記錄日誌以嘗試避免偵測。 另請參閱 exitCode 欄位。
exitCode	int	工作階段結束記錄所用的值。清除結束時，此會包含結束代碼。在某些失敗狀況下可能無法隨時取得結束代碼。在這種情況下，此值可能是零，也可能是空白。
logTime	string	在稽核程式碼路徑中記錄的時間戳記。以國際標準時間 (UTC) 格式表示。如需計算陳述式持續時間的最準確方式，請參閱 startTime 和 endTime 欄位。
netProtocol	string	網路通訊協定。目前，Aurora MySQL 的這個值永遠是 TCP。
objectName	string	SQL 陳述式在其中操作的資料庫物件名稱。僅在 SQL 陳述式在資料庫物件上操作時才會使用此欄位。如果 SQL 陳述式沒有在物件上操作，則此值為空白。若要建構物件的完整名稱，請結合 databaseName 和 objectName 。如果查詢涉及多個物件，則此欄位可以是以逗號分隔的名稱清單。

欄位	資料類型	描述
objectType	string	資料表、索引、檢視等之類的資料庫物件類型。僅在 SQL 陳述式在資料庫物件上操作時才會使用此欄位。如果 SQL 陳述式沒有在物件上操作，則此值為 Null。 Aurora MySQL 的有效值包括以下項目： • INDEX • TABLE • UNKNOWN
paramList	string	此欄位不會用於 Aurora MySQL 且一律為空。
pid	int	後端程序的程序 ID，此程序的配置是用來提供用戶端連線。重新啟動資料庫伺服器時，pid 變更和 statementId 欄位的計數器會重新啟動。
remoteHost	string	發出 SQL 陳述式之用戶端的 IP 地址或主機名稱。針對 Aurora MySQL，使用哪一個取決於資料庫的 skip_name_resolve 參數設定。localhost 的值表示來自 rdsadmin 特殊使用者的活動。
remotePort	string	用戶端連接埠號碼。
rowCount	int	SQL 陳述式傳回的資料列數目。例如，如果一個 SELECT 陳述式傳回 10 個資料列，則 rowCount 為 10。對於 INSERT 或 UPDATE 陳述式，rowCount 為 0。
serverHost	string	資料庫伺服器執行個體識別符。
serverType	string	資料庫伺服器類型，例如 MySQL。
serverVersion	string	資料庫伺服器版本。目前，Aurora MySQL 的這個值永遠是 MySQL 5.7.12。
serviceName	string	服務的名稱。目前，Aurora MySQL 的這個值永遠是 Amazon Aurora MySQL。

欄位	資料類型	描述
sessionId	int	虛擬唯一的工作階段識別符。
startTime (僅限 1.1 版資料庫活動記錄)	string	SQL 陳述式開始執行的時間。以國際標準時間 (UTC) 格式表示。 若要計算 SQL 陳述式的執行時間，請使用 endTime - startTime 。另請參閱 endTime 欄位。
statementId	int	用戶端 SQL 陳述式的識別符。計數器會隨著用戶端輸入的每個 SQL 陳述式而增加。當資料庫執行個體重新啟動時，計數器會重設。
substatementId	int	SQL 子陳述式的識別符。對於具有類別 MAIN 的事件此值是 1，對於具有類別 AUX 的事件則為 2。使用此 statementId 欄位可識別由相同陳述式產生的所有事件。
transactionId (僅限 1.2 版資料庫活動記錄)	int	交易的識別符。
type	string	事件類型。有效值為 record 或 heartbeat 。

使用 AWS SDK 來處理資料庫活動串流

您可以透過使用 AWS 開發套件來以程式設計的方式處理活動串流。以下是功能完整的 Java 和 Python 範例，其示範您可以如何處理 Kinesis 資料串流。

Java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.NoSuchAlgorithmException;
```

```
import java.security.NoSuchProviderException;
import java.security.Security;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.zip.GZIPInputStream;

import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CryptoInputStream;
import com.amazonaws.encryptionsdk.jce.JceMasterKey;
import
 com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ThrottlingException;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorFactory;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.InitialPositionInStream;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.KinesisClientLibConfiguration;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker.Builder;
import com.amazonaws.services.kinesis.model.Record;
import com.amazonaws.services.kms.AWSKMS;
import com.amazonaws.services.kms.AWSKMSClientBuilder;
import com.amazonaws.services.kms.model.DecryptRequest;
import com.amazonaws.services.kms.model.DecryptResult;
import com.amazonaws.util.Base64;
import com.amazonaws.util.IOUtils;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.annotations.SerializedName;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
```

```
public class DemoConsumer {

    private static final String STREAM_NAME = "aws-rds-das-[cluster-external-
resource-id]";
    private static final String APPLICATION_NAME = "AnyApplication"; //unique
 application name for dynamo table generation that holds kinesis shard tracking
    private static final String AWS_ACCESS_KEY =
 "[AWS_ACCESS_KEY_TO_ACCESS_KINESIS]";
    private static final String AWS_SECRET_KEY =
 "[AWS_SECRET_KEY_TO_ACCESS_KINESIS]";
    private static final String DBC_RESOURCE_ID = "[cluster-external-resource-id]";
    private static final String REGION_NAME = "[region-name]"; //us-east-1, us-
east-2...
    private static final BasicAWSCredentials CREDENTIALS = new
 BasicAWSCredentials(AWS_ACCESS_KEY, AWS_SECRET_KEY);
    private static final AWSStaticCredentialsProvider CREDENTIALS_PROVIDER = new
 AWSStaticCredentialsProvider(CREDENTIALS);

    private static final AwsCrypto CRYPTO = new AwsCrypto();
    private static final AWSKMS KMS = AWSKMSClientBuilder.standard()
            .withRegion(REGION_NAME)
            .withCredentials(CREDENTIALS_PROVIDER).build();

    class Activity {
        String type;
        String version;
        String databaseActivityEvents;
        String key;
    }

    class ActivityEvent {
        @SerializedName("class") String _class;
        String clientApplication;
        String command;
        String commandText;
        String databaseName;
        String dbProtocol;
        String dbUserName;
        String endTime;
        String errorMessage;
        String exitCode;
        String logTime;
        String netProtocol;
```

```
        String objectName;
        String objectType;
        List<String> paramList;
        String pid;
        String remoteHost;
        String remotePort;
        String rowCount;
        String serverHost;
        String serverType;
        String serverVersion;
        String serviceName;
        String sessionId;
        String startTime;
        String statementId;
        String substatementId;
        String transactionId;
        String type;
    }

    class ActivityRecords {
        String type;
        String clusterId;
        String instanceId;
        List<ActivityEvent> databaseActivityEventList;
    }

    static class RecordProcessorFactory implements IRecordProcessorFactory {
        @Override
        public IRecordProcessor createProcessor() {
            return new RecordProcessor();
        }
    }

    static class RecordProcessor implements IRecordProcessor {

        private static final long BACKOFF_TIME_IN_MILLIS = 3000L;
        private static final int PROCESSING_RETRIES_MAX = 10;
        private static final long CHECKPOINT_INTERVAL_MILLIS = 60000L;
        private static final Gson GSON = new
GsonBuilder().serializeNulls().create();

        private static final Cipher CIPHER;
        static {
            Security.insertProviderAt(new BouncyCastleProvider(), 1);
```

```
            try {
                CIPHER = Cipher.getInstance("AES/GCM/NoPadding", "BC");
            } catch (NoSuchAlgorithmException | NoSuchPaddingException |
NoSuchProviderException e) {
                throw new ExceptionInInitializerError(e);
            }
        }

        private long nextCheckpointTimeInMillis;

        @Override
        public void initialize(String shardId) {
        }

        @Override
        public void processRecords(final List<Record> records, final
IRecordProcessorCheckpointer checkpointer) {
            for (final Record record : records) {
                processSingleBlob(record.getData());
            }

            if (System.currentTimeMillis() > nextCheckpointTimeInMillis) {
                checkpoint(checkpointer);
                nextCheckpointTimeInMillis = System.currentTimeMillis() +
CHECKPOINT_INTERVAL_MILLIS;
            }
        }

        @Override
        public void shutdown(IRecordProcessorCheckpointer checkpointer,
ShutdownReason reason) {
            if (reason == ShutdownReason.TERMINATE) {
                checkpoint(checkpointer);
            }
        }

        private void processSingleBlob(final ByteBuffer bytes) {
            try {
                // JSON $Activity
                final Activity activity = GSON.fromJson(new String(bytes.array(),
StandardCharsets.UTF_8), Activity.class);

                // Base64.Decode
```

```
                final byte[] decoded =
Base64.decode(activity.databaseActivityEvents);
                final byte[] decodedDataKey = Base64.decode(activity.key);

                Map<String, String> context = new HashMap<>();
                context.put("aws:rds:dbc-id", DBC_RESOURCE_ID);

                // Decrypt
                final DecryptRequest decryptRequest = new DecryptRequest()

.withCiphertextBlob(ByteBuffer.wrap(decodedDataKey)).withEncryptionContext(context);
                final DecryptResult decryptResult = KMS.decrypt(decryptRequest);
                final byte[] decrypted = decrypt(decoded,
getByteArray(decryptResult.getPlaintext()));

                // GZip Decompress
                final byte[] decompressed = decompress(decrypted);
                // JSON $ActivityRecords
                final ActivityRecords activityRecords = GSON.fromJson(new
String(decompressed, StandardCharsets.UTF_8), ActivityRecords.class);

                // Iterate throught $ActivityEvents
                for (final ActivityEvent event :
activityRecords.databaseActivityEventList) {
                    System.out.println(GSON.toJson(event));
                }
            } catch (Exception e) {
                // Handle error.
                e.printStackTrace();
            }
        }

        private static byte[] decompress(final byte[] src) throws IOException {
            ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(src);
            GZIPInputStream gzipInputStream = new
GZIPInputStream(byteArrayInputStream);
            return IOUtils.toByteArray(gzipInputStream);
        }

        private void checkpoint(IRecordProcessorCheckpointer checkpointer) {
            for (int i = 0; i < PROCESSING_RETRIES_MAX; i++) {
                try {
                    checkpointer.checkpoint();
```

```
                    break;
                } catch (ShutdownException se) {
                    // Ignore checkpoint if the processor instance has been shutdown
(fail over).
                    System.out.println("Caught shutdown exception, skipping
checkpoint." + se);
                    break;
                } catch (ThrottlingException e) {
                    // Backoff and re-attempt checkpoint upon transient failures
                    if (i >= (PROCESSING_RETRIES_MAX - 1)) {
                        System.out.println("Checkpoint failed after " + (i + 1) +
"attempts." + e);
                        break;
                    } else {
                        System.out.println("Transient issue when checkpointing -
attempt " + (i + 1) + " of " + PROCESSING_RETRIES_MAX + e);
                    }
                } catch (InvalidStateException e) {
                    // This indicates an issue with the DynamoDB table (check for
table, provisioned IOPS).
                    System.out.println("Cannot save checkpoint to the DynamoDB table
used by the Amazon Kinesis Client Library." + e);
                    break;
                }
                try {
                    Thread.sleep(BACKOFF_TIME_IN_MILLIS);
                } catch (InterruptedException e) {
                    System.out.println("Interrupted sleep" + e);
                }
            }
        }
    }

    private static byte[] decrypt(final byte[] decoded, final byte[] decodedDataKey)
throws IOException {
        // Create a JCE master key provider using the random key and an AES-GCM
encryption algorithm
        final JceMasterKey masterKey = JceMasterKey.getInstance(new
SecretKeySpec(decodedDataKey, "AES"),
                "BC", "DataKey", "AES/GCM/NoPadding");
        try (final CryptoInputStream<JceMasterKey> decryptingStream =
CRYPTO.createDecryptingStream(masterKey, new ByteArrayInputStream(decoded));
             final ByteArrayOutputStream out = new ByteArrayOutputStream()) {
            IOUtils.copy(decryptingStream, out);
```

```
            return out.toByteArray();
        }
    }

    public static void main(String[] args) throws Exception {
        final String workerId = InetAddress.getLocalHost().getCanonicalHostName() +
 ":" + UUID.randomUUID();
        final KinesisClientLibConfiguration kinesisClientLibConfiguration =
                new KinesisClientLibConfiguration(APPLICATION_NAME, STREAM_NAME,
 CREDENTIALS_PROVIDER, workerId);

 kinesisClientLibConfiguration.withInitialPositionInStream(InitialPositionInStream.LATEST);
        kinesisClientLibConfiguration.withRegionName(REGION_NAME);
        final Worker worker = new Builder()
                .recordProcessorFactory(new RecordProcessorFactory())
                .config(kinesisClientLibConfiguration)
                .build();

        System.out.printf("Running %s to process stream %s as worker %s...\n",
 APPLICATION_NAME, STREAM_NAME, workerId);

        try {
            worker.run();
        } catch (Throwable t) {
            System.err.println("Caught throwable while processing data.");
            t.printStackTrace();
            System.exit(1);
        }
        System.exit(0);
    }

    private static byte[] getByteArray(final ByteBuffer b) {
        byte[] byteArray = new byte[b.remaining()];
        b.get(byteArray);
        return byteArray;
    }
}
```

Python

```
import base64
import json
import zlib
```

```
import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy
from aws_encryption_sdk.internal.crypto import WrappingKey
from aws_encryption_sdk.key_providers.raw import RawMasterKeyProvider
from aws_encryption_sdk.identifiers import WrappingAlgorithm, EncryptionKeyType
import boto3

REGION_NAME = '<region>'                    # us-east-1
RESOURCE_ID = '<external-resource-id>'      # cluster-ABCD123456
STREAM_NAME = 'aws-rds-das-' + RESOURCE_ID  # aws-rds-das-cluster-ABCD123456

enc_client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.FORBID_ENCRYPT_AL

class MyRawMasterKeyProvider(RawMasterKeyProvider):
    provider_id = "BC"

    def __new__(cls, *args, **kwargs):
        obj = super(RawMasterKeyProvider, cls).__new__(cls)
        return obj

    def __init__(self, plain_key):
        RawMasterKeyProvider.__init__(self)
        self.wrapping_key =
 WrappingKey(wrapping_algorithm=WrappingAlgorithm.AES_256_GCM_IV12_TAG16_NO_PADDING,
                                        wrapping_key=plain_key,
 wrapping_key_type=EncryptionKeyType.SYMMETRIC)

    def _get_raw_key(self, key_id):
        return self.wrapping_key


def decrypt_payload(payload, data_key):
    my_key_provider = MyRawMasterKeyProvider(data_key)
    my_key_provider.add_master_key("DataKey")
    decrypted_plaintext, header = enc_client.decrypt(
        source=payload,

 materials_manager=aws_encryption_sdk.materials_managers.default.DefaultCryptoMaterialsManag
    return decrypted_plaintext


def decrypt_decompress(payload, key):
    decrypted = decrypt_payload(payload, key)
```

```
    return zlib.decompress(decrypted, zlib.MAX_WBITS + 16)


def main():
    session = boto3.session.Session()
    kms = session.client('kms', region_name=REGION_NAME)
    kinesis = session.client('kinesis', region_name=REGION_NAME)

    response = kinesis.describe_stream(StreamName=STREAM_NAME)
    shard_iters = []
    for shard in response['StreamDescription']['Shards']:
        shard_iter_response = kinesis.get_shard_iterator(StreamName=STREAM_NAME,
 ShardId=shard['ShardId'],

 ShardIteratorType='LATEST')
        shard_iters.append(shard_iter_response['ShardIterator'])

    while len(shard_iters) > 0:
        next_shard_iters = []
        for shard_iter in shard_iters:
            response = kinesis.get_records(ShardIterator=shard_iter, Limit=10000)
            for record in response['Records']:
                record_data = record['Data']
                record_data = json.loads(record_data)
                payload_decoded =
 base64.b64decode(record_data['databaseActivityEvents'])
                data_key_decoded = base64.b64decode(record_data['key'])
                data_key_decrypt_result =
 kms.decrypt(CiphertextBlob=data_key_decoded,

 EncryptionContext={'aws:rds:dbc-id': RESOURCE_ID})
                print (decrypt_decompress(payload_decoded,
 data_key_decrypt_result['Plaintext']))
            if 'NextShardIterator' in response:
                next_shard_iters.append(response['NextShardIterator'])
        shard_iters = next_shard_iters


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

資料庫活動串流的 IAM 政策範例

具有資料庫活動串流適當 AWS Identity and Access Management(IAM) 角色權限的任何使用者可以建立、啟動、停止和修改資料庫叢集的活動串流設定。這些動作會包含在串流的稽核日誌中。為了達到最佳合規實務，我們建議您不要將這些權限提供給 DBA。

您可以使用 IAM 政策來設定資料庫活動串流的存取。如需更多有關 Aurora 身分驗證的資訊，請參閱 Amazon Aurora 的 Identity and access management。如需建立 IAM 政策的詳細資訊，請參閱 建立並使用 IAM 政策進行 IAM 資料庫存取。

Example允許資料庫活動串流設定的原則

若要提供使用者更精細的存取權來修改活動串流，請在 IAM 政策中使用服務特定操作內容金鑰 `rds:StartActivityStream` 與 `rds:StopActivityStream` 。以下 IAM 政策範例會允許使用者或角色設定活動串流。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ConfigureActivityStreams",
            "Effect": "Allow",
            "Action": [
                "rds:StartActivityStream",
                "rds:StopActivityStream"
            ],
            "Resource": "*"
        }
    ]
}
```

Example允許資料庫活動串流開始的原則

以下 IAM 政策範例會允許使用者或角色開始活動串流。

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowStartActivityStreams",
            "Effect":"Allow",
            "Action":"rds:StartActivityStream",
            "Resource":"*"
        }
    ]
}
```

Example允許資料庫活動串流停用的原則

以下 IAM 政策範例會允許使用者或角色停止活動串流。

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowStopActivityStreams",
            "Effect":"Allow",
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
     ]
}
```

Example拒絕資料庫活動串流開始的原則

以下 IAM 政策範例會防止使用者或角色開始活動串流。

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStartActivityStreams",
```

```
            "Effect":"Deny",
            "Action":"rds:StartActivityStream",
            "Resource":"*"
        }
     ]
}
```

Example拒絕資料庫活動串流停用的原則

以下 IAM 政策範例會防止使用者或角色停止活動串流。

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStopActivityStreams",
            "Effect":"Deny",
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
    ]
}
```

使用 Amazon GuardDuty RDS Protection for Amazon Aurora監控威脅

Amazon GuardDuty 是一種威脅偵測服務，可協助保護您的帳戶、容器、工作負載和 AWS 環境中的資料。GuardDuty 使用機器學習 (ML) 模型，以及異常和威脅偵測功能，持續監控不同的日誌來源和執行時期活動，以識別環境中的潛在安全風險和惡意活動和排定其優先順序。

GuardDuty RDS 保護會分析和描述登入事件，以找出 Amazon Aurora 資料庫的潛在存取威脅。當您開啟 RDS 保護時，GuardDuty 會從 Aurora 資料庫取用 RDS 登入事件。RDS Protection 會監控這些事件並描述它們，以找出潛在的內部威脅或外部演員。

如需了解如何啟用 GuardDuty RDS Protection，請參閱《Amazon GuardDuty 使用者指南》中的 GuardDuty RDS Protection。

當 RDS Protection 偵測到潛在威脅 (例如成功或失敗登入嘗試中的異常模式) 時，GuardDuty 會產生新的問題清單，其中包含可能洩露之資料庫的詳細資訊。您可以在 Amazon GuardDuty 主控台的問題清單摘要區段中檢視問題清單的詳細資料。問題清單的詳細資料會根據問題清單類型而有所不同。主要詳細資訊、資源類型和資源角色會決定所有問題清單可用的資訊種類。如需有關問題清單和問題清單類型的常用詳細資訊，請分別參閱《Amazon GuardDuty 使用者指南》中的問題清單詳細資訊和 GuardDuty RDS Protection 問題清單類型。

您可以在可使用 AWS 區域 此功能的任何 AWS 帳戶 中，開啟或關閉任何 的 RDS 保護功能。RDS 保護未啟用時，GuardDuty 不會偵測可能洩露的 Aurora 資料庫，也不會提供洩露的詳細資訊。

現有的 GuardDuty 帳戶可以在 30 天試用期內啟用 RDS Protection。新 GuardDuty 帳戶的 RDS Protection 已啟用，且包含在 30 天免費試用期內。如需詳細資訊，請參閱《Amazon GuardDuty 使用者指南》中的估算 GuardDuty 成本。

如需有關 GuardDuty 尚未支援 RDS 保護 AWS 區域 的 的資訊，請參閱《Amazon GuardDuty 使用者指南》中的區域特定功能可用性。

如需 GuardDuty RDS 保護支援的 Aurora 資料庫版本的資訊，請參閱《Amazon GuardDuty 使用者指南》中的支援的 Amazon Aurora、Amazon RDS 和 Aurora Limitless 資料庫。

使用 Amazon Aurora MySQL

Amazon Aurora 是與全受管 MySQL 相容的關聯式資料庫引擎，結合了高端商用資料庫的速度和可靠性，以及開放原始碼資料庫的簡單與經濟實惠。Aurora MySQL 是 MySQL 的便利替代方案，以簡單且經濟實惠的方式設定、操作及擴展新的及現有的 MySQL 部署，讓您得以心無旁，專注於業務和應用程式。Amazon RDS 會處理例行資料庫任務 (例如佈建、修補、備份、復原、故障偵測和修復)，以提供 Aurora 的管理。Amazon RDS 會提供按鈕遷移工具，以將現有的 Amazon RDS for MySQL 應用程式轉換為 Aurora MySQL。

主題

- Amazon Aurora MySQL 概觀
- Amazon Aurora MySQL 的安全性
- 將應用程式更新為使用新的 TLS 憑證來連線至 Aurora MySQL 資料庫叢集
- 針對 Aurora MySQL 使用 Kerberos 身分驗證
- 將資料遷移至 Amazon Aurora MySQL 資料庫叢集
- 管理 Amazon Aurora MySQL
- 調校 Aurora MySQL
- Amazon Aurora MySQL 的平行查詢
- 使用進階稽核與 Amazon Aurora MySQL 資料庫叢集搭配
- 以 Amazon Aurora MySQL 進行複寫
- 在 Amazon Aurora MySQL 資料庫叢集中使用本機寫入轉送
- 將 Amazon Aurora MySQL 與其他 AWS 服務整合
- Amazon Aurora MySQL 實驗室模式
- Amazon Aurora MySQL 的最佳實務
- 對 Amazon Aurora MySQL 資料庫效能進行故障診斷
- Amazon Aurora MySQL 參考
- Amazon Aurora MySQL 的資料庫引擎更新

Amazon Aurora MySQL 概觀

下列各節提供 Amazon Aurora MySQL 的概觀。

主題

- Amazon Aurora MySQL 效能增強功能
- Amazon Aurora MySQL 和空間資料
- Aurora MySQL 第 3 版與 MySQL 8.0 相容
- Aurora MySQL 第 2 版與 MySQL 5.7 相容

Amazon Aurora MySQL 效能增強功能

Amazon Aurora 包括效能增強功能以支援高階商用資料庫多樣化的需求。

快速插入

快速插入可加速依主要索引鍵排序的平行插入，並專門套用至 `LOAD DATA` 和 `INSERT INTO ... SELECT ...` 陳述式。在執行陳述式時，快速插入會快取索引周遊中游標的位置。這可避免不必要地重新周遊索引。

快速插入功能僅適用於 Aurora MySQL 3.03.2 版和更新版本中的一般 InnoDB 資料表。此最佳化不適用於 InnoDB 暫存資料表。它在所有 2.11 和 2.12 版本的 Aurora MySQL 第 2 版中已停用。快速插入最佳化只有在停用自適應雜湊索引最佳化時才有效。

您可以監控下列指標，以判斷資料庫叢集的快速插入效果：

- `aurora_fast_insert_cache_hits`：成功擷取並驗證快取游標時會遞增的計數器。
- `aurora_fast_insert_cache_misses`：快取游標不再有效，且 Aurora 執行正常索引周遊時會遞增的計數器。

您可以使用下列命令，擷取快速插入指標的目前值：

```
mysql> show global status like 'Aurora_fast_insert%';
```

您將會得到類似下列的輸出：

```
+----------------------------------+-----------+
| Variable_name                    | Value     |
+----------------------------------+-----------+
```

```
| Aurora_fast_insert_cache_hits   | 3598300   |
| Aurora_fast_insert_cache_misses | 436401336 |
+---------------------------------+-----------+
```

Amazon Aurora MySQL 和空間資料

下列清單彙總主要 Aurora MySQL 空間特性，以及說明它們如何對應至 MySQL 中的空間特性。

- Aurora MySQL 第 2 版支援與 MySQL 5.7 相同的空間資料類型和空間關聯式函式。如需這些資料類型和函式的詳細資訊，請參閱 MySQL 5.7 文件中的空間資料類型和空間關聯式函式。
- Aurora MySQL 第 3 版支援與 MySQL 8.0 相同的空間資料類型和空間關聯式函式。如需這些資料類型和函式的詳細資訊，請參閱 MySQL 8.0 文件中的空間資料類型和空間關聯式函式。
- Aurora MySQL 支援 InnoDB 資料表上的空間檢索。空間檢索可改善大型資料集上對空間資料進行查詢的查詢效能。在 MySQL 中，InnoDB 資料表的空間檢索可用於 MySQL 5.7 和 8.0。

  Aurora MySQL 會使用來自 MySQL 的不同空間檢索策略，以便可以高效能進行空間查詢。Aurora 空間索引實作會在 B 樹狀結構上使用空間填滿曲線，其目的旨在為空間範圍掃描提供比 R 樹狀結構更高的效能。

  > **Note**
  >
  > 在 Aurora MySQL 中，資料表上的交易若有空間索引定義在具有空間參考識別符 (SRID) 的資料欄上，則無法插入至另一個交易為了更新而選取的區域。

支援下列資料定義語言 (DDL) 陳述式，可在使用空間資料類型的資料欄上建立索引。

CREATE TABLE

您可以在 `SPATIAL INDEX` 陳述式中使用 `CREATE TABLE` 關鍵字，將空間索引新增至新的資料表。以下是範例。

```
CREATE TABLE test (shape POLYGON NOT NULL, SPATIAL INDEX(shape));
```

ALTER TABLE

您可以在 `SPATIAL INDEX` 陳述式中使用 `ALTER TABLE` 關鍵字，將空間索引新增至現有資料表中的資料欄。以下是範例。

```
ALTER TABLE test ADD SPATIAL INDEX(shape);
```

CREATE INDEX

您可以在 SPATIAL 陳述式中使用 CREATE INDEX 關鍵字，將空間索引新增至現有資料表中的資料欄。以下是範例。

```
CREATE SPATIAL INDEX shape_index ON test (shape);
```

Aurora MySQL 第 3 版與 MySQL 8.0 相容

您可以使用 Aurora MySQL 第 3 版，取得與 MySQL 相容的最新功能、效能增強功能和錯誤修正。你可以在下文了解 Aurora MySQL 第 3 版，與 MySQL 8.0 的相容性。您可以了解如何將叢集和應用程式升級至 Aurora MySQL 第 3 版。

一些 Aurora 功能，例如 Aurora Serverless v2, 需要 Aurora MySQL 第 3 版。

主題

- MySQL 8.0 社群版的功能
- Aurora MySQL Serverless v2 的 Aurora MySQL 第 3 版先決條件
- Aurora MySQL 第 3 版的版本備註
- 新的平行查詢最佳化
- 最佳化以減少資料庫重新啟動時間
- Aurora MySQL 第 3 版的新暫時資料表行為
- 比較 Aurora MySQL 第 2 版和 Aurora MySQL 第 3 版
- 比較 Aurora MySQL 第 3 版與 MySQL 8.0 社群版
- 升級至 Aurora MySQL 第 3 版

MySQL 8.0 社群版的功能

最初發行的 Aurora MySQL 第 3 版與 MySQL 8.0.23 社群版相容。MySQL 8.0 引進了幾個新功能，包括以下功能：

- 單元資料定義語言 (DDL) 支援。如需更多詳細資訊，請參閱 單元資料定義語言 (DDL) 支援。。
- JSON 函數 如需用法資訊，請參閱《MySQL 參考手冊》中的 JSON 函數。
- 視窗函數。如需用法資訊，請參閱《MySQL 參考手冊》中的視窗函數。

- 一般資料表表達式 (CTE)，使用 WITH 子句。如需用法資訊，請參閱《MySQL 參考手冊》中的 WITH (一般資料表表達式)。
- 已針對 ALTER TABLE 陳述式最佳化 ADD COLUMN 和 RENAME COLUMN 子句。這些最佳化稱為「即時 DDL」。Aurora MySQL 第 3 版與社群 MySQL 即時 DLL 功能相容。未使用早期的 Aurora 快速 DLL 功能。如需即時 DDL 的用法資訊，請參閱 即時 DDL (Aurora MySQL 第 3 版)。
- 遞減索引、功能索引和隱藏索引。如需用法資訊，請參閱《MySQL 參考手冊》中的隱藏索引、遞減索引，以及CREATE INDEX 陳述式。
- 透過 SQL 陳述式控制的角色型權限。如需權限模型變更的詳細資訊，請參閱角色型權限模型。
- 搭配 SELECT ... FOR SHARE 陳述式的 NOWAIT 和 SKIP LOCKED 子句。這些子句避免等待其他交易釋放資料列鎖定。如需用法資訊，請參閱《MySQL 參考手冊》中的鎖定讀取。
- 二進位日誌 (binlog) 複寫的改進。如需 Aurora MySQL 詳細資訊，請參閱二進位日誌複寫。尤其，您可以執行篩選的複寫。如需篩選複寫的用法資訊，請參閱《MySQL 參考手冊》中的伺服器如何評估複寫篩選規則。
- 提示。一些 MySQL 8.0 相容的提示已向後移植到 Aurora MySQL 第 2 版。如需搭配使用提示與 Aurora MySQL 的詳細資訊，請參閱 Aurora MySQL 提示。如需社群 MySQL 8.0 中的完整提示清單，請參閱《MySQL 參考手冊》中的最佳化工具提示。

如需新增到 MySQL 8.0 社群的完整功能清單，請參閱部落格文章 The complete list of new features in MySQL 8.0。

Aurora MySQL 第 3 版還包括對包容性語言之關鍵字的變更，從社群 MySQL 8.0.26 向後移植。如需這些變更的詳細資訊，請參閱Aurora MySQL 第 3 版的包容性語言變更。

Aurora MySQL Serverless v2 的 Aurora MySQL 第 3 版先決條件

Aurora MySQL 第 3 版是 Aurora MySQL Serverless v2 叢集中所有資料庫執行個體的先決條件。Aurora MySQL Serverless v2 包括對資料庫叢集中的讀取器執行個體的支援，及其他 Aurora 功能，這些功能不適用於 Aurora MySQL Serverless v1。其還具有比 Aurora MySQL Serverless v1 更快、更精細的擴展。

Aurora MySQL 第 3 版的版本備註

如需所有 Aurora MySQL 第 3 版的版本備註，請參閱 Aurora MySQL 版本備註中的 Amazon Aurora MySQL 第 3 版的資料庫引擎更新。

新的平行查詢最佳化

Aurora 平行查詢最佳化現在適用於更多的 SQL 作業：

- 平行查詢現在適用於包含資料類型 TEXT、BLOB、JSON、GEOMETRY 和 VARCHAR，以及 CHAR (長度超過 768 個位元組) 的資料表。
- 平行查詢可以最佳化涉及分割資料表的查詢。
- 平行查詢可以最佳化查詢，其中涉及選取清單中的彙總函數呼叫，以及 HAVING 子句。

如需這些增強功能的詳細資訊，請參閱將平行查詢叢集升級至 Aurora MySQL 第 3 版。如需 Aurora 平行查詢的一般資訊，請參閱Amazon Aurora MySQL 的平行查詢。

最佳化以減少資料庫重新啟動時間

您的 Aurora MySQL 資料庫叢集在計畫中斷和意外中斷期間都必須具有高可用性。

資料庫管理員需要偶爾執行資料庫維護。此維護包括資料庫修補、升級、需要手動重新開機的資料庫參數修改、執行容錯移轉以減少執行個體類別變更所需的時間等等。這些計劃的動作需要停機。

不過，停機時間也可能是因為意外的動作造成，例如因為基礎硬體故障或資料庫資源限流而導致意外的容錯移轉。所有這些計劃和未計劃的動作都會導致資料庫重新啟動。

在 Aurora MySQL 3.05 及更高版本中，我們引入了可縮短資料庫重新啟動時間的最佳化功能。與沒有最佳化相比，這些最佳化可減少多達 65% 的停機時間，並且在重新啟動後減少資料庫工作負載的中斷情況。

在資料庫啟動期間，會初始化許多內部記憶體元件。其中最大的是 InnoDB 緩衝區集區，在 Aurora MySQL 中預設為執行個體記憶體大小的 75%。我們的測試發現初始化時間與 InnoDB 緩衝池的大小成正比，因此可以隨著資料庫執行個體類別大小進行擴展。在此初始化階段，資料庫無法接受連線，這會在重新啟動期間造成較長的停機時間。Aurora MySQL 快速重新啟動的第一階段會最佳化緩衝區集區初始化，以減少資料庫初始化的時間，進而減少整體重新啟動時間。

如需詳細資訊，請參閱部落格：利用 Amazon Aurora MySQL 資料庫重新啟動時間最佳化減少停機時間。

Aurora MySQL 第 3 版的新暫時資料表行為

Aurora MySQL 第 3 版會以與舊版 Aurora MySQL 不同的方式處理暫時資料表。這種新行為是從 MySQL 8.0 社群版繼承的。有兩種類型的暫時資料表可以使用 Aurora MySQL 第 3 版建立：

- 內部 (或隱含) 暫時資料表 – 由 Aurora MySQL 引擎建立，可處理如排序彙總、衍生資料表，或通用資料表表達式 (CTE) 等操作。
- 使用者建立 (或明確) 的暫時資料表 –當您使用 `CREATE TEMPORARY TABLE` 陳述式時由 Aurora MySQL 引擎建立。

Aurora 讀取器資料庫執行個體上的內部暫時資料表和使用者建立的暫時資料表皆有其他注意事項。我們會在下列各節討論這些變更。

主題

- 內部 (隱含) 暫時資料表的儲存引擎
- 限制記憶體內部暫存資料表的大小
- 緩解 Aurora 複本上內部暫時資料表的完整性問題
- 在 Aurora MySQL 資料庫執行個體上最佳化 temptable_max_mmap 參數
- 讀取器資料庫執行個體上的使用者建立 (明確) 的暫時資料表
- 暫時資料表建立時發生的錯誤和緩解措施

內部 (隱含) 暫時資料表的儲存引擎

產生中繼結果集時，Aurora MySQL 一開始會嘗試寫入記憶體內暫時資料表。這可能不成功，因為資料類型不相容或設定了限制。若是如此，則暫存資料表會轉換為磁碟上暫存資料表，而不是保留在記憶體中。如需詳細資訊，請參閱 MySQL 文件中的 MySQL 中內部暫時資料表的使用。

在 Aurora MySQL 第 3 版中，內部暫時資料表的運作方式與早期的 Aurora MySQL 版本不同。現在您可以在 與 InnoDB 儲存引擎之間進行選擇，而不是為這類暫時資料表，在 `TempTable` 和 `MEMORY` 儲存引擎之間進行選擇。

使用 `TempTable` 儲存引擎，您可以另外選擇如何處理特定資料。受影響的資料溢出記憶體集區，此記憶體集區保留資料庫執行個體的所有內部暫時資料表。

這些選擇可能會影響產生大量暫存資料之查詢的效能，例如在大型資料表上執行彙總 (例如 `GROUP BY`) 時。

> **Tip**
>
> 如果您的工作負載包含產生內部暫時資料表的查詢，請執行基準化分析並監控效能相關指標，以確認您的應用程式如何搭配此變更執行。
>
> 在某些情況下，暫存資料量容納於 `TempTable` 記憶體集區內，或僅少量溢出記憶體集區。在這些情況下，我們建議將 `TempTable` 設定用於內部暫時資料表，並使用記憶體映射檔案來保留任何溢位資料。此設定是預設值。

`TempTable` 儲存引擎是預設值。`TempTable` 會針對使用此引擎的所有暫時資料表使用一般記憶體集區，而不是每個資料表的最大記憶體限制。此記憶體集區的大小是由 temptable_max_ram 參數指定。

其在具有 16 GiB 或更多記憶體的資料庫執行個體上預設為 1 GiB，而在記憶體少於 16 GiB 的資料庫執行個體上則預設為 16 MB。記憶體集區的大小會影響工作階段層級的記憶體耗用量。

在某些情況下，當您使用 TempTable 儲存引擎時，暫存資料可能會超過記憶體集區的大小。若是如此，Aurora MySQL 會使用次要機制來存放溢位資料。

您可以設定 temptable_max_mmap 參數來指定資料是否溢出到記憶體對應的暫存檔案，還是溢出到磁碟上的 InnoDB 內部暫時資料表。這些溢位機制的不同資料格式和溢位準則可能會影響查詢效能。它們會藉由影響寫入至磁碟的資料量以及對磁碟儲存輸送量的需求來達到此目的。

Aurora MySQL 第 3 版會以下列方式存放溢出資料：

- 在寫入器資料庫執行個體上，溢出至 InnoDB 內部暫存資料表或記憶體映射暫存檔案的資料會存放在執行個體上的本機儲存體中。
- 在讀取器執行個體上，溢出資料始終位於本機儲存上的記憶體映射暫存檔中。

  唯讀執行個體無法在 Aurora 叢集磁碟區上存放任何資料。

與內部暫時資料表相關的組態參數會以不同方式套用至叢集中的寫入器和讀取器執行個體：

- 在讀取器執行個體上，MySQL Aurora 始終使用 TempTable 儲存引擎。
- 對於寫入器和讀取器執行個體，不論資料庫執行個體的記憶體大小為何，temptable_max_mmap 的大小都會預設為 1 GiB。您可以在寫入器和讀取器執行個體上調整此值。
- 將 temptable_max_mmap 設為 0 會關閉在寫入器執行個體上使用記憶體映射暫存檔案的功能。
- 您無法在讀取器執行個體上將 temptable_max_mmap 設為 0。

> **Note**
>
> 我們不建議使用 temptable_use_mmap 參數。該參數已被棄用，且預計未來的 MySQL 版本不會再支援該參數。

限制記憶體內部暫存資料表的大小

正如 內部 (隱含) 暫時資料表的儲存引擎 中所討論，您可以使用 temptable_max_ram 和 temptable_max_mmap 設定控制全域資源暫存資料表。

您也可以使用 tmp_table_size 資料庫參數限制個別內部、記憶體內暫存資料表的大小。此限制是為了防止個別查詢耗用大量的全域資源暫存資料表資源，這可能會影響同時需要這些資源的查詢效能。

tmp_table_size 參數定義 MEMORY 儲存體引擎在 Aurora MySQL 3 版中建立暫存資料表的大小上限。

Aurora MySQL 3.04 版及更新版本中，tmp_table_size 還定義 aurora_tmptable_enable_per_table_limit 資料庫參數設定為 ON 時，TempTable 儲存體引擎建立臨時資料表的大小上限。此行為預設為停用 (OFF)，這與 Aurora MySQL 3.03 版及更低版本中的行為相同。

- 當 aurora_tmptable_enable_per_table_limit 為 OFF 時，透過 TempTable 儲存體引擎建立記憶體內部暫存資料表不用考慮 tmp_table_size。

  但是，全域 TempTable 資源限制仍然適用。當全域 TempTable 資源達到限制時，Aurora MySQL 具有下列行為：

  - 寫入器資料庫執行個體 – Aurora MySQL 會自動將記憶體內暫存資料表轉換為 InnoDB 磁碟上的暫存資料表。
  - 讀取器資料庫執行個體 – 查詢因錯誤而結束。

    ```
    ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlxx_xxx' is full
    ```

- 當 aurora_tmptable_enable_per_table_limit 為 ON 時，全域 tmp_table_size 資源達到限制，Aurora MySQL 具有下列行為：

  - 寫入器資料庫執行個體 – Aurora MySQL 會自動將記憶體內暫存資料表轉換為 InnoDB 磁碟上的暫存資料表。
  - 讀取器資料庫執行個體 – 查詢因錯誤而結束。

    ```
    ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlxx_xxx' is full
    ```

  在這種情況下，全域 TempTable 資源限制和每個資料表限制都適用。

> **Note**
>
> 當 internal_tmp_mem_storage_engine 設定為 MEMORY 時，aurora_tmptable_enable_per_table_limit 參數沒有作用。在這種情況下，記憶

體內暫存資料表的大小上限由 tmp_table_size 或者 max_heap_table_size 值定義，以較小者為準。

下列範例顯示 aurora_tmptable_enable_per_table_limit 參數對寫入器和讀取器資料庫執行個體的行為。

Example或者將寫入器資料庫執行個體 aurora_tmptable_enable_per_table_limit 設定為 OFF

記憶體內的暫存資料表不會轉換為 InnoDB 磁碟上的暫存資料表。

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+-----------------+-----------------------------------------
+----------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+-----------------+-----------------------------------------
+----------------------+----------------------+
|                  0 | 3.04.0          |                                        0 |
       1073741824 |          1073741824 |
+--------------------+-----------------+-----------------------------------------
+----------------------+----------------------+
1 row in set (0.00 sec)

mysql> show status like '%created_tmp_disk%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| Created_tmp_disk_tables | 0     |
+--------------------------+-------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 60000000) SELECT max(n) FROM cte;
+----------+
| max(n)   |
```

```
+----------+
| 60000000 |
+----------+
1 row in set (13.99 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)
```

Example或者將寫入器資料庫執行個體 aurora_tmptable_enable_per_table_limit 設定為 ON

記憶體內的暫存資料表會轉換為 InnoDB 磁碟上的暫存資料表。

```
mysql> set aurora_tmptable_enable_per_table_limit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@tmp_table_size;
+--------------------+------------------+-----------------------------------------
+------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@tmp_table_size |
+--------------------+------------------+-----------------------------------------
+------------------+
|                  0 | 3.04.0           |                                        1 |
      16777216 |
+--------------------+------------------+-----------------------------------------
+------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)
```

```
mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 6000000) SELECT max(n) FROM cte;
+---------+
| max(n)  |
+---------+
| 6000000 |
+---------+
1 row in set (4.10 sec)

mysql> show status like '%created_tmp_disk%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| Created_tmp_disk_tables | 1     |
+--------------------------+-------+
1 row in set (0.00 sec)
```

Example將讀取器資料庫執行個體 aurora_tmptable_enable_per_table_limit 設定為 OFF

查詢結束時沒有錯誤，因為 tmp_table_size 不適用，並且全域 TempTable 資源尚未達到限制。

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+-----------------+-----------------------------------------
+---------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+-----------------+-----------------------------------------
+---------------------+----------------------+
|                  1 | 3.04.0           |                                        0 |
       1073741824 |           1073741824 |
+--------------------+-----------------+-----------------------------------------
+---------------------+----------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 60000000) SELECT max(n) FROM cte;
```

```
+----------+
| max(n)   |
+----------+
| 60000000 |
+----------+
1 row in set (14.05 sec)
```

Example將讀取器資料庫執行個體 aurora_tmptable_enable_per_table_limit 設定為 OFF

此查詢已達到全域 TempTable 資源限制，將 `aurora_tmptable_enable_per_table_limit` 設定為關閉。查詢因讀取器執行個體上發生錯誤而結束。

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+------------------+-----------------------------------------
+----------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+------------------+-----------------------------------------
+----------------------+----------------------+
|                  1 | 3.04.0           |                                        0 |
       1073741824 |           1073741824 |
+--------------------+------------------+-----------------------------------------
+----------------------+----------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.01 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 120000000) SELECT max(n) FROM cte;
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlfd_1586_2' is full
```

Example將讀取器資料庫執行個體 aurora_tmptable_enable_per_table_limit 設定為 ON

已達到 `tmp_table_size` 限制，查詢因發生錯誤而結束。

```
mysql> set aurora_tmptable_enable_per_table_limit=1;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@tmp_table_size;
+--------------------+------------------+----------------------------------------
+------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@tmp_table_size |
+--------------------+------------------+----------------------------------------
+------------------+
|                  1 | 3.04.0           |                                        1 |
      16777216 |
+--------------------+------------------+----------------------------------------
+------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 6000000) SELECT max(n) FROM cte;
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlfd_8_2' is full
```

緩解 Aurora 複本上內部暫時資料表的完整性問題

若要避免發生暫時資料表的大小限制問題，請將 `temptable_max_ram` 和 `temptable_max_mmap` 參數設定為符合工作負載需求的合併值。

設定 `temptable_max_ram` 參數的值時請小心。將此值設定得過高，會減少資料庫執行個體上的可用記憶體，這可能會導致發生記憶體不足的狀況。監控資料庫執行個體上的平均可用記憶體。然後為 `temptable_max_ram` 決定適合的值，以便執行個體上仍能保留合理的可用記憶體量。如需詳細資訊，請參閱Amazon Aurora 中的可用記憶體問題。

監控本機儲存空間的大小和暫時資料表空間的耗用狀況，也很重要。您可以使用 `FreeLocalStorage` Amazon CloudWatch 指標來監控特定資料庫執行個體可用的暫存空間，如 Amazon Aurora 的 Amazon CloudWatch 指標 中所述。

> **Note**
>
> 當 `aurora_tmptable_enable_per_table_limit` 參數設定為 `ON` 時，此程序不適用。如需詳細資訊，請參閱限制記憶體內部暫存資料表的大小。

Example 1

您知識暫時資料表的大小會累積成長至 20 GiB。您希望將記憶體內暫時資料表設定為 2 GiB，並在磁碟上成長到上限 20 GiB。

將 `temptable_max_ram` 設定為 **`2,147,483,648`**，將 `temptable_max_mmap` 設定為 **`21,474,836,480`**。這些值以位元組為單位。

這些參數設定確保暫時資料表可累積成長到總共 22 GiB。

Example 2

您目前的執行個體大小為 16xlarge 或以上。您不知道您可能需要的暫時資料表總大小。您希望能夠在記憶體中使用多達 4 GiB，最多達磁碟上的可用儲存空間大小上限。

將 `temptable_max_ram` 設定為 **`4,294,967,296`**，將 `temptable_max_mmap` 設定為 **`1,099,511,627,776`**。這些值以位元組為單位。

在此您將 `temptable_max_mmap` 設定為 1 TiB，這小於 16xlarge Aurora 資料庫執行個體上 1.2 TiB 的本機儲存空間上限。

在大小較小的執行個體上，調整 `temptable_max_mmap` 的值，使其不會填滿可用的本機儲存空間。例如，一個 2xlarge 執行個體只有 160 GiB 的可用本機儲存空間。因此，建議將該值設定為小於 160 GiB。如需資料庫執行個體大小的可用本機儲存空間詳細資訊，請參閱 Aurora MySQL 的暫存空間限制。

在 Aurora MySQL 資料庫執行個體上最佳化 temptable_max_mmap 參數

Aurora MySQL 中的 `temptable_max_mmap` 參數控制記憶體映射檔案在溢出至磁碟上 InnoDB 暫存資料表 (寫入器資料庫執行個體上) 或導致錯誤 (讀取器資料庫執行個體上) 之前可使用的本機磁碟空間上限。正確設定此資料庫執行個體參數有助於最佳化資料庫執行個體的效能。

先決條件

1. 確定已啟用效能結構描述。您可以透過執行以下 SQL 命令來進行確認。

   ```
   SELECT @@performance_schema;
   ```

   1 的輸出值表示已啟用。
2. 確認已啟用暫存資料表記憶體檢測。您可以透過執行以下 SQL 命令來進行確認。

```
SELECT name, enabled FROM performance_schema.setup_instruments WHERE name LIKE
 '%memory%temptable%';
```

enabled 欄會針對相關的暫存資料表記憶體檢測項目顯示 YES。

監控暫存資料表用量

設定 temptable_max_mmap 的初始值時，建議您從使用的資料庫執行個體類別的本機儲存體大小的 80% 開始。這可確保暫存資料表有足夠的磁碟空間可有效運作，同時為執行個體上的其他磁碟使用保留空間。

若要尋找資料庫執行個體類別的本機儲存體大小，請參閱 Aurora MySQL 的暫存空間限制。

例如，如果您使用的是 db.r5.large 資料庫執行個體類別，則本機儲存體大小為 32 GiB。在這種情況下，您一開始會將 temptable_max_mmap 參數設定為 32 GiB 的 80%，也就是 25.6 GiB。

設定初始 temptable_max_mmap 值後，在 Aurora MySQL 執行個體上執行尖峰工作負載。使用下列 SQL 查詢監控目前和高暫存資料表磁碟用量：

```
SELECT event_name, current_count, current_alloc, current_avg_alloc, high_count,
 high_alloc, high_avg_alloc
FROM sys.memory_global_by_current_bytes WHERE event_name LIKE 'memory/temptable/%';
```

此查詢會擷取下列資訊：

- event_name – 暫存資料表記憶體或磁碟用量事件的名稱。
- current_count – 目前配置的暫存資料表記憶體或磁碟區塊數量。
- current_alloc – 針對暫存資料表配置的目前記憶體或磁碟數量。
- current_avg_alloc – 暫存資料表記憶體或磁碟區塊的目前平均大小。
- high_count – 配置的暫存資料表記憶體或磁碟區塊數量上限。
- high_alloc – 針對暫存資料表配置的最高記憶體或磁碟數量。
- high_avg_alloc – 暫存資料表記憶體或磁碟區塊的平均大小上限。

如果您的查詢失敗，且使用此設定時發生資料表已滿錯誤，則表示您的工作負載需要更多磁碟空間來執行暫存資料表操作。在此情況下，請考慮將資料庫執行個體大小增加為具有更多本機儲存空間的執行個體大小。

設定最佳 **temptable_max_mmap** 值

使用下列程序來監控和設定 temptable_max_mmap 參數的正確大小。

1. 檢閱上一個查詢的輸出，並識別尖峰暫存資料表磁碟用量，如 high_alloc 欄所示。
2. 根據尖峰暫存資料表磁碟用量，調整 Aurora MySQL 資料庫執行個體資料庫參數群組中的 temptable_max_mmap 參數。

   將值設為稍微高於尖峰暫存資料表磁碟用量，以適應未來的成長。
3. 將參數群組變更套用至資料庫執行個體。
4. 在尖峰工作負載期間再次監控暫存資料表磁碟用量，以確保新 temptable_max_mmap 值是適當的。
5. 視需要重複上述步驟，以微調 temptable_max_mmap 參數。

讀取器資料庫執行個體上的使用者建立 (明確) 的暫時資料表

您可以在 CREATE TABLE 陳述式中使用 TEMPORARY 的關鍵字建立明確的暫時資料表。Aurora 資料庫叢集中寫入器資料庫執行個體支援明確暫時資料表。您也可以在讀取器資料庫執行個體上使用明確暫時資料表，但該資料表無法強制使用 InnoDB 儲存引擎。

若要避免在 Aurora MySQL 讀取器資料庫執行個體上建立明確暫時資料表時發生錯誤，請確定您以下列其中一種或兩種方式執行所有 CREATE TEMPORARY TABLE 陳述句：

- 不指定 ENGINE=InnoDB 子句。
- 不將 SQL 模式設為 NO_ENGINE_SUBSTITUTION。

暫時資料表建立時發生的錯誤和緩解措施

您收到的錯誤會有所不同，取決於您使用純 CREATE TEMPORARY TABLE 陳述式，還是變異 CREATE TEMPORARY TABLE AS SELECT。下列範例顯示不同類型的錯誤。

此暫時資料表行為僅適用於唯讀執行個體。這個第一個範例確認其是工作階段連線到的執行個體類型。

```
mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
```

對於純 CREATE TEMPORARY TABLE 陳述式，陳述式會在 NO_ENGINE_SUBSTITUTION SQL 模式開啟時失敗。當 NO_ENGINE_SUBSTITUTION 關閉時 (預設值),會進行適當的引擎替換，並會成功建立暫時資料表。

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql>  CREATE TEMPORARY TABLE tt2 (id int) ENGINE=InnoDB;
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> CREATE TEMPORARY TABLE tt4 (id int) ENGINE=InnoDB;

mysql> SHOW CREATE TABLE tt4\G
*************************** 1. row ***************************
       Table: tt4
Create Table: CREATE TEMPORARY TABLE `tt4` (
  `id` int DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

對於 CREATE TEMPORARY TABLE AS SELECT 陳述式，陳述式會在 NO_ENGINE_SUBSTITUTION SQL 模式開啟時失敗。當 NO_ENGINE_SUBSTITUTION 關閉時 (預設值),會進行適當的引擎替換，並會成功建立暫時資料表。

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql> CREATE TEMPORARY TABLE tt1 ENGINE=InnoDB AS SELECT * FROM t1;
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> show create table tt3;
+-------+-----------------------------------------------------------+
| Table | Create Table                                              |
+-------+-----------------------------------------------------------+
| tt3   | CREATE TEMPORARY TABLE `tt3` (
  `id` int DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+-------+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

如需有關 Aurora MySQL 第 3 版中臨時資料表儲存方面和效能影響的詳細資訊，請參閱部落格文章 Use the TempTable storage engine on Amazon RDS for MySQL and Amazon Aurora MySQL (在 Amazon RDS for MySQL 和 Amazon Aurora MySQL 上使用 TempTable 儲存引擎)。

比較 Aurora MySQL 第 2 版和 Aurora MySQL 第 3 版

使用以下內容來了解當您將 Aurora MySQL 第 2 版叢集升級至第 3 版時，應注意的變更。

主題

- 單元資料定義語言 (DDL) 支援。
- Aurora MySQL 第 2 版與第 3 版之間的功能差異
- 執行個體類別支援
- Aurora MySQL 第 3 版的參數變更
- 狀態變數
- Aurora MySQL 第 3 版的包容性語言變更
- AUTO_INCREMENT 值
- 二進位日誌複寫

單元資料定義語言 (DDL) 支援。

從 MySQL 5.7 到 8.0 的最大變更之一是引入單元資料字典。在 MySQL 8.0 之前，MySQL 資料字典使用以檔案為基礎的方法來存放中繼資料，例如資料表定義 (.frm)、觸發 (.trg)，以及與儲存引擎中繼資料 (例如 InnoDB 的) 分開的函數。這有一些問題，包括如果在 DDL 操作期間發生意外，導致檔案型和儲存引擎中繼資料不同步，資料表會變成「孤立」的風險。

為了修正此問題，MySQL 8.0 推出了單元資料字典，該字典會將所有中繼資料存放在 `mysql` 結構描述中的一組內部 InnoDB 資料表中。這個新架構提供交易、ACID 合規的方式來管理資料庫中繼資料，從舊的檔案型方法解決「單元 DDL」問題。如需單元資料字典的詳細資訊，請參閱 MySQL 參考手冊中的移除檔案型中繼資料儲存和單元資料定義陳述式支援。

由於此架構變更，從 Aurora MySQL 第 2 版升級至第 3 版時，您必須考量下列事項：

- 第 2 版的檔案型中繼資料必須在升級到第 3 版的過程中遷移到新的資料字典資料表。視遷移多少資料庫物件而定，這可能需要一些時間。
- 這些變更也引入了一些新的不相容，可能需要先解決，才能從 MySQL 5.7 升級到 8.0。例如，8.0 有一些新的預留關鍵字，可能與現有的資料庫物件名稱衝突。

為了協助您在升級引擎之前識別這些不相容，Aurora MySQL 會執行一系列的升級相容性檢查 (預先檢查)，以確定資料庫字典中是否有任何不相容的物件，然後再執行資料字典升級。如需關於預先檢查的詳細資訊，請參閱 Aurora MySQL 的主要版本升級預先檢查。

Aurora MySQL 第 2 版與第 3 版之間的功能差異

Aurora MySQL for MySQL 5.7 中支援下列 Amazon Aurora MySQL 功能，但 Aurora MySQL for MySQL 8.0 中不支援這些功能。

- 您無法將 Aurora MySQL 第 3 版用於 Aurora Serverless v1 叢集。Aurora MySQL 第 3 版可與 Aurora Serverless v2 搭配使用。
- 實驗室模式不適用於 Aurora MySQL 第 3 版。Aurora MySQL 第 3 版中沒有任何實驗室模式功能。即時 DDL 會取代先前可在實驗室模式中使用的快速線上 DDL 功能。如需範例，請參閱「即時 DDL (Aurora MySQL 第 3 版)」。
- 查詢快取會從社群 MySQL 8.0 移除，也會從 Aurora MySQL 第 3 版移除。
- Aurora MySQL 第 3 版與社群 MySQL 雜湊聯結功能相容。未使用 Aurora MySQL 第 2 版中雜湊聯結的 Aurora 特定實作。如需搭配使用雜湊聯結與 Aurora 平行查詢的相關資訊，請參閱開啟平行查詢叢集的雜湊聯結和 Aurora MySQL 提示。如需雜湊聯結的一般用法資訊，請參閱《MySQL 參考手冊》中的雜湊聯結最佳化。
- 已在 Aurora MySQL 第 2 版中取代的 `mysql.lambda_async` 預存程序會在第 3 版中移除。對於第 3 版，請改用非同步函數 `lambda_async`。
- Aurora MySQL 第 3 版中的預設字元集是 `utf8mb4`。在 Aurora MySQL 第 2 版中，預設字元集是 `latin1`。如需此字元集的相關資訊，請參閱《MySQL 參考手冊》中的 utf8mb4 字元集 (4 位元組 UTF-8 Unicode 編碼)。

某些 Aurora MySQL 功能可用於 AWS 區域和資料庫引擎版本的特定組合。如需詳細資訊，請參閱 AWS 區域 和 Aurora 資料庫引擎在 Amazon Aurora 中支援的功能。

執行個體類別支援

Aurora MySQL 第 3 版支援的執行個體類別集與 Aurora MySQL 第 2 版支援的不同：

- 對於較大的執行個體，您可以使用新式執行個體類別，例如 `db.r5`、`db.r6g` 和 `db.x2g`。
- 對於較小的執行個體，您可以使用新式執行個體類別，例如 `db.t3` 和 `db.t4g`。

> **Note**
>
> 建議您在開發、測試伺服器或其他非生產伺服器時，僅使用 T 資料庫執行個體類別。如需詳細了解 T 執行個體類別，請參閱 使用 T 執行個體類別進行開發和測試。

來自 Aurora MySQL 第 2 版的以下執行個體類別不適用於 Aurora MySQL 第 3 版：

- `db.r4`
- `db.r3`
- `db.t3.small`
- `db.t2`

檢查您的管理指令碼是否有任何建立 Aurora MySQL 資料庫執行個體的 CLI 陳述式。無法用於 Aurora MySQL 第 3 版的硬式編碼執行個體類別名稱。如有必要，請將執行個體類別名稱修改為 Aurora MySQL 第 3 版支援的名稱。

> **Tip**
>
> 若要檢查您可以用於 Aurora MySQL 版本和 AWS 區域之特定組合的執行個體類別，請使用 `describe-orderable-db-instance-options` AWS CLI 命令。

如需 Aurora 執行個體類別的完整詳細資訊，請參閱 Amazon Aurora 資料庫執行個體類別。

Aurora MySQL 第 3 版的參數變更

Aurora MySQL 第 3 版包含新的叢集層級和執行個體層級組態參數。Aurora MySQL 第 3 版也移除一些存在於 Aurora MySQL 第 2 版中的參數。有些參數名稱會由於包容性語言的倡議而變更。為了回溯相容性，您仍然可以使用舊名稱或新名稱來擷取參數值。不過，您必須使用新名稱來指定自訂參數群組中的參數值。

在 Aurora MySQL 第 3 版中，`lower_case_table_names` 參數的值會在建立叢集時永久設定。如果您對此選項使用非預設值，請在升級之前設定您的 Aurora MySQL 第 3 版自訂參數群組。然後，在建立叢集或快照還原作業期間指定參數群組。

> **Note**
>
> 使用以 Aurora MySQL 為基礎的 Aurora 全域資料庫時，若啟用 `lower_case_table_names` 參數，即無法從 Aurora MySQL 第 2 版就地升級至第 3 版。請改用快照還原技術。

在 Aurora MySQL 第 3 版中，`init_connect` 和 `read_only` 參數不適用於具有 `CONNECTION_ADMIN` 權限的使用者。這包括 Aurora 主要使用者。如需更多詳細資訊，請參閱 角色型權限模型。

如需 Aurora MySQL 叢集參數的完整清單，請參閱叢集層級參數。資料表涵蓋了來自 Aurora MySQL 第 2 和 3 版的所有參數。資料表包含附註，其中顯示哪些參數在 Aurora MySQL 第 3 版中是新的，或是哪些參數已從 Aurora MySQL 第 3 版中移除。

如需 Aurora MySQL 執行個體參數的完整清單，請參閱執行個體層級參數。資料表涵蓋了來自 Aurora MySQL 第 2 和 3 版的所有參數。資料表包含附註，其中顯示哪些參數在 Aurora MySQL 第 3 版中是新的，以及哪些參數已從 Aurora MySQL 第 3 版中移除。它還包括其他附註，其中顯示哪些參數在早期版本中是可修改的，但不是 Aurora MySQL 第 3 版。

如需已變更之參數名稱的相關資訊，請參閱 Aurora MySQL 第 3 版的包容性語言變更。

狀態變數

如需不適用於 Aurora MySQL 之狀態變數的相關資訊，請參閱不適用於 Aurora MySQL 的 MySQL 狀態變數。

Aurora MySQL 第 3 版的包容性語言變更

Aurora MySQL 第 3 版與來自 MySQL 社群版的 8.0.23 版相容。Aurora MySQL 第 3 版還包括來自 MySQL 8.0.26 的變更，這些變更與包容性語言之關鍵字和系統結構描述相關。例如，`SHOW REPLICA STATUS` 命令現在是偏好的命令，而不是 `SHOW SLAVE STATUS`。

下列 Amazon CloudWatch 指標在 Aurora MySQL 第 3 版中具有新名稱。

在 Aurora MySQL 第 3 版中，只有新的指標名稱可用。升級至 Aurora MySQL 第 3 版時，請務必更新任何警示或其他依賴指標名稱的自動化。

舊名稱	新名稱	
`ForwardingMasterDMLLatency`	`ForwardingWriterDMLLatency`	

舊名稱	新名稱	
ForwardingMasterOpenSessions	ForwardingWriterOpenSessions	
AuroraDMLRejectedMasterFull	AuroraDMLRejectedWriterFull	
ForwardingMasterDMLThroughput	ForwardingWriterDMLThroughput	

下列狀態變數在 Aurora MySQL 第 3 版中具有新名稱。

為了相容性，您可以在初始的 Aurora MySQL 第 3 版中使用任一個名稱。舊的狀態變數名稱將在未來版本中移除。

要移除的名稱	新名稱或偏好名稱	
Aurora_fwd_master_dml_stmt_duration	Aurora_fwd_writer_dml_stmt_duration	
Aurora_fwd_master_dml_stmt_count	Aurora_fwd_writer_dml_stmt_count	
Aurora_fwd_master_select_stmt_duration	Aurora_fwd_writer_select_stmt_duration	
Aurora_fwd_master_select_stmt_count	Aurora_fwd_writer_select_stmt_count	
Aurora_fwd_master_errors_session_timeout	Aurora_fwd_writer_errors_session_timeout	
Aurora_fwd_master_open_sessions	Aurora_fwd_writer_open_sessions	

要移除的名稱	新名稱或偏好名稱	
Aurora_fwd_master_errors_session_limit	Aurora_fwd_writer_errors_session_limit	
Aurora_fwd_master_errors_rpc_timeout	Aurora_fwd_writer_errors_rpc_timeout	

下列組態參數在 Aurora MySQL 第 3 版中具有新名稱。

為了相容性，您可以檢查 mysql 用戶端中的參數名稱，方法是在初始 Aurora MySQL 第 3 版中使用任一名稱。修改自訂參數群組中的值時，您只能使用新名稱。舊的參數名稱將在未來版本中移除。

要移除的名稱	新名稱或偏好名稱	
aurora_fwd_master_idle_timeout	aurora_fwd_writer_idle_timeout	
aurora_fwd_master_max_connections_pct	aurora_fwd_writer_max_connections_pct	
master_verify_checksum	source_verify_checksum	
sync_master_info	sync_source_info	
init_slave	init_replica	
rpl_stop_slave_timeout	rpl_stop_replica_timeout	
log_slow_slave_statements	log_slow_replica_statements	
slave_max_allowed_packet	replica_max_allowed_packet	

要移除的名稱	新名稱或偏好名稱	
slave_compressed_protocol	replica_compressed_protocol	
slave_exec_mode	replica_exec_mode	
slave_type_conversions	replica_type_conversions	
slave_sql_verify_checksum	replica_sql_verify_checksum	
slave_parallel_type	replica_parallel_type	
slave_preserve_commit_order	replica_preserve_commit_order	
log_slave_updates	log_replica_updates	
slave_allow_batching	replica_allow_batching	
slave_load_tmpdir	replica_load_tmpdir	
slave_net_timeout	replica_net_timeout	
sql_slave_skip_counter	sql_replica_skip_counter	
slave_skip_errors	replica_skip_errors	
slave_checkpoint_period	replica_checkpoint_period	
slave_checkpoint_group	replica_checkpoint_group	

要移除的名稱	新名稱或偏好名稱	
slave_transaction_retries	replica_transaction_retries	
slave_parallel_workers	replica_parallel_workers	
slave_pending_jobs_size_max	replica_pending_jobs_size_max	
pseudo_slave_mode	pseudo_replica_mode	

下列預存程序在 Aurora MySQL 第 3 版中具有新名稱。

為了相容性，您可以在初始的 Aurora MySQL 第 3 版中使用任一個名稱。未來版本將移除舊的程序名稱。

要移除的名稱	新名稱或偏好名稱	
mysql.rds_set_master_auto_position	mysql.rds_set_source_auto_position	
mysql.rds_set_external_master	mysql.rds_set_external_source	
mysql.rds_set_external_master_with_auto_position	mysql.rds_set_external_source_with_auto_position	
mysql.rds_reset_external_master	mysql.rds_reset_external_source	
mysql.rds_next_master_log	mysql.rds_next_source_log	

AUTO_INCREMENT 值

在 Aurora MySQL 第 3 版中，Aurora 會在其重新啟動每個資料庫執行個體時保留每個資料表的 AUTO_INCREMENT 值。在 Aurora MySQL 第 2 版中，重新啟動後未保留 AUTO_INCREMENT 值。

從快照還原、執行時間點復原，以及複製叢集來設定新叢集時，不會保留 AUTO_INCREMENT 值。在這些情況下，AUTO_INCREMENT 值會在建立快照時初始化為基於資料表中最大資料欄值的值。這種行為與 RDS for MySQL 8.0 中不同，其中 AUTO_INCREMENT 值會在這些作業期間保留。

二進位日誌複寫

在 MySQL 8.0 社群版中，預設為開啟二進位日誌複寫。在 Aurora MySQL 第 3 版中，預設為關閉二進位日誌複寫。

> **Tip**
>
> 如果 Aurora 內建複寫功能滿足您的高可用性需求，您可以將二進位日誌複寫保留關閉狀態。如此一來，就可以避免二進位日誌複寫的效能負荷。也可以避免管理二進位日誌複寫所需的相關聯監控和疑難排解。

Aurora 支援二進位日誌從 MySQL 5.7 相容來源複寫到 Aurora MySQL 第 3 版。來源系統可以是 Aurora MySQL 資料庫叢集、RDS for MySQL 資料庫執行個體或內部部署 MySQL 執行個體。

與社群 MySQL 一樣，Aurora MySQL 支援從執行特定版本的來源複寫到執行相同主要版本或更高主要版本的目標。例如，不支援從 MySQL 5.6 相容系統複寫到 Aurora MySQL 第 3 版。不支援從 Aurora MySQL 第 3 版複寫到 MySQL 5.7 相容系統或 MySQL 5.6 相容系統。如需使用二進位日誌複寫的詳細資訊，請參閱 Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。

Aurora MySQL 第 3 版包含對社群 MySQL 8.0 中二進位日誌複寫的改進，例如篩選的複寫。如需有關社群 MySQL 8.0 改進的詳細資訊，請參閱《MySQL 參考手冊》中的伺服器如何評估複寫篩選規則。

二進位日誌複寫的交易壓縮

如需二進位日誌壓縮的用法資訊，請參閱《MySQL 參考手冊》中的二進位日誌交易壓縮。

下列限制適用於 Aurora MySQL 第 3 版中的二進位日誌壓縮：

- 其二進位日誌資料大於最大允許封包大小的交易不會進行壓縮。無論 Aurora MySQL 二進位日誌壓縮設定是否開啟，都會發生此情況。這類交易會在不壓縮的情況下進行複寫。

- 如果您對尚未支援 MySQL 8.0 的變更資料擷取 (CDC) 使用連接器，則無法使用此功能。建議您使用二進位日誌壓縮，完整測試任何第三方連接器。此外，建議您先這樣做，然後在針對 CDC 使用 binlog 複寫的系統上開啟 binlog 壓縮。

比較 Aurora MySQL 第 3 版與 MySQL 8.0 社群版

您可以使用下列資訊，來了解當您從不同的 MySQL 8.0 相容系統轉換為 Aurora MySQL 第 3 版時，應注意的變更。

通常，Aurora MySQL 第 3 版支援社群 MySQL 8.0.23 的功能集。來自 MySQL 8.0 社群版的一些新功能不適用於 Aurora MySQL。其中有些功能與 Aurora 的某些方面不相容，例如 Aurora 儲存架構。不需要其他功能，因為 Amazon RDS 管理服務會提供同等功能。社群 MySQL 8.0 中的下列功能不受支援，或在 Aurora MySQL 第 3 版中以不同方式運作。

如需所有 Aurora MySQL 第 3 版的版本備註，請參閱 Aurora MySQL 版本備註中的 Amazon Aurora MySQL 第 3 版的資料庫引擎更新。

主題

- MySQL 8.0 功能不適用於 Aurora MySQL 第 3 版
- 角色型權限模型
- 尋找資料庫伺服器 ID
- 身分驗證

MySQL 8.0 功能不適用於 Aurora MySQL 第 3 版

來自社群 MySQL 8.0 中的下列功能無法使用，或在 Aurora MySQL 第 3 版中以不同方式運作。

- Aurora MySQL 中不支援資源群組和相關聯的 SQL 陳述式。
- Aurora MySQL 不支援使用者定義的復原資料表空間和相關聯的 SQL 陳述式，例如 `CREATE UNDO TABLESPACE`、`ALTER UNDO TABLESPACE ... SET INACTIVE` 和 `DROP UNDO TABLESPACE`。
- 對於低於 3.06 的 Aurora MySQL 版本，Aurora MySQL 不支援復原資料表空間截斷。在 Aurora MySQL 3.06 版和更新版本中，支援自動復原資料表空間截斷。
- 支援密碼驗證外掛程式。
- 您無法修改任何 MySQL 外掛程式的設定，包括密碼驗證外掛程式。
- 不支援 X 外掛程式。

- 不支援多來源複寫。

角色型權限模型

使用 Aurora MySQL 第 3 版，您無法直接修改 mysql 資料庫中的資料表。尤其，您無法藉由插入至 mysql.user 資料表來設定使用者。相反地，您可以使用 SQL 陳述式來授與角色型權限。您也無法建立其他類型的物件，例如 mysql 資料庫中已存放的程序。您仍然可以查詢 mysql 資料表。如果您使用二進位日誌複寫，則直接對來源叢集上 mysql 資料表所做的變更不會複寫至目標叢集。

在某些情況下，您的應用程式可能會使用捷徑來建立使用者或其他物件，方法是插入至 mysql 資料表。若是這樣，請變更您應用程式的程式碼來使用對應的陳述式，例如 CREATE USER。如果您的應用程式在 mysql 資料庫中建立已存放的程序或其他物件，請改為使用不同的資料庫。

若要在遷移期間從外部 MySQL 資料庫匯出資料庫使用者的中繼資料，您可以使用 MySQL Shell 命令，而不是 mysqldump。如需詳細資訊，請參閱執行個體傾印公用程式、結構描述傾印公用程式和資料表傾印公用程式。

若要簡化許多使用者或應用程式的權限管理，您可以使用 CREATE ROLE 陳述式來建立具有一組權限的角色。然後，您可以使用 GRANT 和 SET ROLE 陳述式，以及 current_role 函數，將角色指派給使用者或應用程式、切換目前角色，並檢查哪些角色有效。如需 MySQL 8.0 中角色型權限系統的詳細資訊，請參閱《MySQL 參考手冊》中的使用角色。

> **Important**
>
> 我們強烈建議您不要直接在您的應用程式中使用主要使用者。而是遵循最佳實務，使用以應用程式所需的最低權限建立的資料庫使用者。

主題

- rds_superuser_role
- 權限檢查使用者是否有二進位日誌複寫
- 存取其他 AWS 服務的角色

rds_superuser_role

Aurora MySQL 第 3 版包含具有下列所有權限的特殊角色。此角色已命名為 rds_superuser_role。每個叢集的主要管理使用者已授與此角色。rds_superuser_role 角色包含所有資料庫物件的下列權限：

- ALTER
- APPLICATION_PASSWORD_ADMIN
- ALTER ROUTINE
- CONNECTION_ADMIN
- CREATE
- CREATE ROLE
- CREATE ROUTINE
- CREATE TEMPORARY TABLES
- CREATE USER
- CREATE VIEW
- DELETE
- DROP
- DROP ROLE
- EVENT
- EXECUTE
- FLUSH_OPTIMIZER_COSTS (Aurora MySQL 3.09 版及更新版本)
- FLUSH_STATUS (Aurora MySQL 3.09 版及更新版本)
- FLUSH_TABLES (Aurora MySQL 3.09 版及更新版本)
- FLUSH_USER_RESOURCES (Aurora MySQL 3.09 版及更新版本)
- INDEX
- INSERT
- LOCK TABLES
- PROCESS
- REFERENCES
- RELOAD
- REPLICATION CLIENT
- REPLICATION SLAVE
- ROLE_ADMIN
- SET_USER_ID
- SELECT

- SHOW DATABASES
- SHOW_ROUTINE (Aurora MySQL 3.04 版及更新版本)
- SHOW VIEW
- TRIGGER
- UPDATE
- XA_RECOVER_ADMIN

角色定義還包括 WITH GRANT OPTION，以便管理使用者可以將該角色授與其他使用者。尤其，管理員必須授與執行二進位日誌複寫所需的任何權限，以 Aurora MySQL 叢集做為目標。

> **Tip**
>
> 若要查看權限的完整詳細資訊，請輸入下列陳述式。
>
> ```
> SHOW GRANTS FOR rds_superuser_role@'%';
> SHOW GRANTS FOR name_of_administrative_user_for_your_cluster@'%';
> ```

權限檢查使用者是否有二進位日誌複寫

Aurora MySQL 第 3 版包含權限檢查使用者是否有二進位日誌 (binlog) 複寫，rdsrepladmin_priv_checks_user。除了 rds_superuser_role 的權限之外，此使用者還有 replication_applier 權限。

當您呼叫 mysql.rds_start_replication 預存程序來開啟 binlog 複寫時，rdsrepladmin_priv_checks_user 會建立。

rdsrepladmin_priv_checks_user@localhost 使用者是預留使用者。請勿修改它。

存取其他 AWS 服務的角色

Aurora MySQL 第 3 版包含您可用來存取其他 AWS 服務的角色。您可以設定許多這些角色作為授予權限的替代方案。例如，您可以指定 GRANT AWS_LAMBDA_ACCESS TO *user*，而不是 GRANT INVOKE LAMBDA ON *.* TO *user*。如需存取其他 AWS 服務的程序，請參閱 將 Amazon Aurora MySQL 與其他 AWS 服務整合。Aurora MySQL 第 3 版包含下列與存取其他服務相關的角色 AWS ：

- AWS_LAMBDA_ACCESS – 作為 INVOKE LAMBDA 權限的替代項目。如需使用方式的資訊，請參閱 從 Amazon Aurora MySQL 資料庫叢集叫用 Lambda 函式。

- AWS_LOAD_S3_ACCESS – 作為 LOAD FROM S3 權限的替代項目。如需使用方式的資訊，請參閱 從 Amazon S3 儲存貯體中的文字檔案將資料載入 Amazon Aurora MySQL 資料庫叢集。
- AWS_SELECT_S3_ACCESS – 作為 SELECT INTO S3 權限的替代項目。如需使用方式的資訊，請參閱 將來自 Amazon Aurora MySQL 資料庫叢集的資料儲存至 Amazon S3 儲存貯體中的文字檔案。
- AWS_COMPREHEND_ACCESS – 作為 INVOKE COMPREHEND 權限的替代項目。如需使用方式的資訊，請參閱 授與資料庫使用者存取 Aurora Machine Learning 的權限。
- AWS_SAGEMAKER_ACCESS – 作為 INVOKE SAGEMAKER 權限的替代項目。如需使用方式的資訊，請參閱 授與資料庫使用者存取 Aurora Machine Learning 的權限。
- AWS_BEDROCK_ACCESS – Amazon Bedrock 沒有類似 INVOKE 權限。如需使用方式的資訊，請參閱 授與資料庫使用者存取 Aurora Machine Learning 的權限。

當使用 Aurora MySQL 第 3 版中的角色授與存取權時，您也可以使用 SET ROLE *role_name* 或 SET ROLE ALL 陳述式啟用角色。下列範例會顯示作法。以適當的角色名稱替代 AWS_SELECT_S3_ACCESS。

```
# Grant role to user.

mysql> GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'

# Check the current roles for your user. In this case, the AWS_SELECT_S3_ACCESS role
 has not been activated.
# Only the rds_superuser_role is currently in effect.
mysql> SELECT CURRENT_ROLE();
+--------------------------+
| CURRENT_ROLE()           |
+--------------------------+
| `rds_superuser_role`@`%` |
+--------------------------+
1 row in set (0.00 sec)

# Activate all roles associated with this user using SET ROLE.
# You can activate specific roles or all roles.
# In this case, the user only has 2 roles, so we specify ALL.
mysql> SET ROLE ALL;
Query OK, 0 rows affected (0.00 sec)

# Verify role is now active
mysql> SELECT CURRENT_ROLE();
```

```
+-----------------------------------------------------+
| CURRENT_ROLE()                                      |
+-----------------------------------------------------+
| `AWS_SELECT_S3_ACCESS`@`%`,`rds_superuser_role`@`%` |
+-----------------------------------------------------+
```

尋找資料庫伺服器 ID

二進位記錄 (binlog) 複寫需要資料庫伺服器 ID (`server_id`)。在 Aurora MySQL 和 Community MySQL 中尋找伺服器 ID MySQL 的方式不同。

在 Community MySQL 中，伺服器 ID 是您在登入伺服器時使用下列語法取得的數字：

```
mysql> select @@server_id;

+-------------+
| @@server_id |
+-------------+
| 2           |
+-------------+
1 row in set (0.00 sec)
```

在 Aurora MySQL 中，伺服器 ID 是您在登入資料庫執行個體時使用下列語法取得的資料庫執行個體 ID：

```
mysql> select @@aurora_server_id;

+-------------------------+
| @@aurora_server_id      |
+-------------------------+
| mydbcluster-instance-2 |
+-------------------------+
1 row in set (0.00 sec)
```

如需 binlog 複寫的詳細資訊，請參閱 Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。

身分驗證

在社群 MySQL 8.0 中，預設身分驗證外掛程式為 `caching_sha2_password`。Aurora MySQL 第 3 版仍會使用 `mysql_native_password` 外掛程式。您無法變更

default_authentication_plugin 設定。不過，您可以建立新的使用者並變更目前的使用者，而且其個別密碼會使用新的身分驗證外掛程式。以下是範例。

```
mysql> CREATE USER 'testnewsha'@'%' IDENTIFIED WITH caching_sha2_password BY
 'aNewShaPassword';
Query OK, 0 rows affected (0.74 sec)
```

升級至 Aurora MySQL 第 3 版

如需將資料庫從 Aurora MySQL 第 2 版升級到第 3 版的相關資訊，請參閱 升級 Amazon Aurora MySQL 資料庫叢集的主要版本。

Aurora MySQL 第 2 版與 MySQL 5.7 相容

本主題描述 Aurora MySQL 第 2 版與 MySQL 5.7 Community Edition。

> **Important**
>
> Aurora MySQL 第 2 版已在 2024 年 10 月 31 日達到標準支援結束。如需更多詳細資訊，請參閱 為 Amazon Aurora MySQL 相容版本第 2 版標準支援結束做好準備。

Aurora MySQL 第 2 版中不支援的功能

在 MySQL 5.7 中支援下列功能，但在 Aurora MySQL 2 版中目前不支援：

- CREATE TABLESPACE SQL 陳述式
- 群組複寫外掛程式
- 已增加的頁面大小
- 啟動時載入 InnoDB 緩衝集區
- InnoDB 全文剖析器外掛程式
- 多來源複寫
- 線上緩衝集區大小調整
- 密碼驗證外掛程式 – 您可安裝外掛程式，但其不受支援。您無法自訂外掛程式。
- 查詢重寫外掛程式
- 複寫篩選
- X 通訊協定

如需這些功能的詳細資訊，請參閱 MySQL 5.7 文件。

Aurora MySQL 第 2 版的暫存資料表行為

在 MySQL 5.7 中，暫存資料表空間會自動擴充，並視需要增加大小以容納磁碟上的暫存資料表。捨棄暫存資料表時，釋放的空間可以重複用於新的暫存資料表，但暫存資料表空間仍會維持延伸大小，而且不會縮小。重新啟動引擎時，會捨棄並重新建立暫存資料表空間。

在 Aurora MySQL 第 2 版中，下列行為適用：

- 對於使用 2.10 版和更新版本建立的新 Aurora MySQL 資料庫叢集，會在您重新啟動資料庫時移除並重新建立暫存資料表空間。這允許動態調整大小功能回收儲存空間。
- 對於升級至下列版本的現有 Aurora MySQL 資料庫叢集：
  - 2.10 版或更新版本 - 在您重新啟動資料庫時，系統會移除並重新建立暫存資料表空間。這允許動態調整大小功能回收儲存空間。
  - 2.09 版 - 重新啟動資料庫時，不會移除暫存資料表空間。

您可以使用下列查詢，檢查 Aurora MySQL 第 2 版資料庫叢集上暫存資料表空間的大小：

```
SELECT
    FILE_NAME,
    TABLESPACE_NAME,
    ROUND((TOTAL_EXTENTS * EXTENT_SIZE) / 1024 / 1024 / 1024, 4) AS SIZE
FROM
    INFORMATION_SCHEMA.FILES
WHERE
    TABLESPACE_NAME = 'innodb_temporary';
```

如需詳細資訊，請參閱 MySQL 文件中的暫存資料表空間。

磁碟上暫存資料表的儲存引擎

Aurora MySQL 第 2 版會針對磁碟上內部暫存資料表使用不同的儲存引擎，取決於執行個體的角色。

- 在寫入器執行個體上，磁碟上暫存資料表預設會使用 InnoDB 儲存引擎。它們會存放在 Aurora 叢集磁碟區的暫存資料表空間中。

  您可以修改資料庫參數 `internal_tmp_disk_storage_engine` 的值，以在寫入器執行個體上變更此行為。如需更多詳細資訊，請參閱 執行個體層級參數。

- 在讀取器執行個體上，磁碟上暫存資料表會使用 MyISAM 儲存引擎，此儲存引擎會使用本機儲存體。這是因為唯讀執行個體無法在 Aurora 叢集磁碟區上存放任何資料。

Amazon Aurora MySQL 的安全性

Amazon Aurora MySQL 的安全性是以三個層級來管理：

- 若要控制誰可以在 Aurora MySQL 資料庫叢集和資料庫執行個體上執行 Amazon RDS 管理動作，您可以使用 AWS Identity and Access Management (IAM)。當您 AWS 使用 IAM 登入資料連線至 時，AWS 您的帳戶必須具有授予執行 Amazon RDS 管理操作所需許可的 IAM 政策。如需詳細資訊，請參閱Amazon Aurora 的 Identity and access management

  如果您使用 IAM 存取 Amazon RDS 主控台，請務必先 AWS 管理主控台 使用 IAM 使用者憑證登入 。接著前往 Amazon RDS 主控台：https://console.aws.amazon.com/rds/。
- Aurora MySQL 資料庫叢集必須在以 Amazon VPC 服務為基礎的虛擬公有雲端 (VPC) 中建立。如要控制哪些裝置和 Amazon EC2 執行個體可以開放對 VPC 中 Aurora MySQL 資料庫叢集的資料庫執行個體端點和連線埠的連線，可使用 VPC 安全群組。您可以使用 Transport Layer Security (TLS) 進行這些端點和連線埠連線。此外，貴公司的防火牆規則可控管在公司內執行的裝置是否可開啟與資料庫執行個體的連線。如需 VPC 的詳細資訊，請參閱Amazon VPC 和 Amazon Aurora。

  支援的 VPC 租用取決於您的 Aurora MySQL 資料庫叢集所使用的資料庫執行個體類別。使用 `default` VPC 租用時，VPC 在共用硬體上執行。使用 `dedicated` VPC 租用時，VPC 會在專用硬體執行個體上執行。爆量效能資料庫執行個體類別僅支援預設的 VPC 租用。爆量效能資料庫執行個體類別包括 db.t2、db.t3 和 db.t4g 資料庫執行個體類別。其他所有 Aurora MySQL 資料庫執行個體類別都支援預設和專用的 VPC 租用。

  > **Note**
  >
  > 建議您在開發、測試伺服器或其他非生產伺服器時，僅使用 T 資料庫執行個體類別。如需詳細了解 T 執行個體類別，請參閱 使用 T 執行個體類別進行開發和測試。

  如需執行個體類別的詳細資訊，請參閱Amazon Aurora 資料庫執行個體類別。如需 `default` 和 `dedicated` VPC 租用的詳細資訊，請參閱《Amazon Elastic Compute Cloud 使用者指南》中的專用執行個體。
- 若要驗證 Amazon Aurora MySQL 資料庫叢集的登入資訊與許可，您可採取下列任一方式，或搭配使用多種方法：
  - 您可採用與 MySQL 獨立執行個體相同的驗證方式。

CREATE USER、RENAME USER、GRANT、REVOKE 及 SET PASSWORD 等命令的運作方式與現場部署資料庫所使用的命令相同，會直接修改資料庫結構描述資料表。如需詳細資訊，請參閱 MySQL 文件中的存取控制和帳戶管理。

- 您也可以使用 IAM 資料庫身分驗證。

  透過 IAM 資料庫身分驗證，您可以使用 IAM 使用者或 IAM 角色以及身分驗證字符，驗證您的資料庫叢集。身分驗證字符是不重複的值，由 Signature 第 4 版簽署程序所產生。透過使用 IAM 資料庫身分驗證，您可以使用相同的登入資料來控制對 AWS 資源和資料庫的存取。如需詳細資訊，請參閱 IAM 資料庫身分驗證。

> **Note**
>
> 如需詳細資訊，請參閱 Amazon Aurora 的安全性。

在下列各節中，請參閱 Aurora MySQL 的使用者許可以及使用 Aurora MySQL 資料庫叢集進行 TLS 連線的相關資訊。

主題

- Amazon Aurora MySQL 的主要使用者權限
- Aurora MySQL 資料庫叢集的 TLS 連線

Amazon Aurora MySQL 的主要使用者權限

在您建立 Amazon Aurora MySQL 資料庫執行個體後，主要使用者具有 主要使用者帳戶權限 中列出的預設權限：

若要為每個資料庫叢集提供管理服務，建立資料庫執行個體時，系統會一併建立 admin 和 rdsadmin 使用者。若企圖移除、重新命名 rdsadmin 帳戶或變更其密碼或權限，皆會導致發生錯誤。

在 Aurora MySQL 第 2 版資料庫叢集中，admin 和 rdsadmin 使用者會在建立資料庫叢集時一併建立。在 Aurora MySQL 第 3 版資料庫叢集中，會建立 admin、rdsadmin 和 rds_superuser_role 使用者。

> **Important**
>
> 我們強烈建議您不要直接在您的應用程式中使用主要使用者。而是遵循最佳實務，使用以應用程式所需的最低權限建立的資料庫使用者。

在管理 Aurora MySQL 資料庫叢集時，標準 `kill` 與 `kill_query` 命令均限制使用。請改用 Amazon RDS 命令 `rds_kill` 與 `rds_kill_query`，以終止 Aurora MySQL 資料庫執行個體上的使用者工作階段或查詢。

> **Note**
>
> 中國 (寧夏) 區域不支援資料庫執行個體和快照的加密。

Aurora MySQL 資料庫叢集的 TLS 連線

如果應用程式使用與 RDS for MySQL 資料庫執行個體相同的程序和公有金鑰，Amazon Aurora MySQL 資料庫叢集支援來自這些應用程式的 Transport Layer Security (TLS) 連線。

在 Amazon RDS 佈建執行個體時，Amazon RDS 會建立 TLS 憑證，並將該憑證安裝在資料庫執行個體上。憑證由憑證授權機構簽署。TLS 憑證會以一般名稱 (CN) 納入資料庫執行個體端點，讓 TLS 憑證免於詐騙攻擊。因此，只有當您的用戶端支援主體別名 (SAN) 時，您才能透過 TLS 使用資料庫叢集端點以連線至資料庫叢集。否則，您必須使用寫入器執行個體的執行個體端點。

如需有關下載憑證的詳細資訊，請參閱使用 SSL/TLS 加密與資料庫叢集的連線。

我們建議 AWS JDBC 驅動程式做為支援 SAN 搭配 TLS 的用戶端。如需 AWS JDBC 驅動程式的詳細資訊和完整的使用說明，請參閱 Amazon Web Services (AWS) JDBC 驅動程式 GitHub 儲存庫。

主題

- 需要以 TLS 連線至 Aurora MySQL 資料庫叢集
- 適用於 Aurora MySQL 的 TLS 版本
- 為 Aurora MySQL 資料庫叢集的連線設定密碼套件
- 加密 Aurora MySQL 資料庫叢集的連線

需要以 TLS 連線至 Aurora MySQL 資料庫叢集

您可以要求連線至 Aurora MySQL 資料庫叢集的所有使用者，藉由使用 `require_secure_transport` 資料庫叢集參數，來使用 TLS。依預設，`require_secure_transport` 參數設為 `OFF`。您可以將 `require_secure_transport` 參數設為 `ON`，以要求對資料庫叢集的連線使用 TLS。

您可以更新資料庫叢集的資料庫叢集參數群組，以設定 `require_secure_transport` 參數值。您不需要重新啟動資料庫叢集，變更即可生效。如需參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。

> **Note**
>
> `require_secure_transport` 參數適用於 Aurora MySQL 第 2 版和第 3 版。您可以在自訂資料庫叢集參數群組中設定此參數。該參數不適用於資料庫執行個體參數群組。

當資料庫叢集的 `require_secure_transport` 參數設為 `ON` 時，如果可以建立加密的連線，則資料庫用戶端即可連線到該資料庫叢集。否則，類似下列內容的錯誤訊息會傳回至用戶端：

```
MySQL Error 3159 (HY000): Connections using insecure transport are prohibited while --
require_secure_transport=ON.
```

適用於 Aurora MySQL 的 TLS 版本

Aurora MySQL 支援 Transport Layer Security (TLS) 1.0、1.1、1.2 和 1.3 版。從 Aurora MySQL 3.04.0 及更新版本開始，您可以使用 TLS 1.3 通訊協定來保護您的連線。下表顯示 Aurora MySQL 版本的 TLS 支援。

Aurora MySQL version	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	預設
Aurora MySQL 第 2 版	已棄用	已棄用	支援	不支援	所有支援的 TLS 版本
Aurora MySQL 第	已棄用	已棄用	支援	不支援	所有支援的 TLS 版本

Aurora MySQL version	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	預設
3 版 (早於 3.04.0 的版本)					
Aurora MySQL 第 3 版 (3.04.0 和更新版本)	不支援	不支援	支援	支援	所有支援的 TLS 版本

Important

如果您在版本 2 和版本低於 3.04.0 的 Aurora MySQL 叢集上使用自訂參數群組，我們建議您使用 TLS 1.2，因為 TLS 1.0 和 1.1 的安全性較低。MySQL 8.0.26 和 Aurora MySQL 3.03 的社群版本及其次要版本已淘汰對 TLS 版本 1.1 和 1.0 的支援。

MySQL 8.0.28 和相容的 Aurora MySQL 3.04.0 及更新版本的社群版本不支援 TLS 1.1 和 TLS 1.0。如果您使用的是極 Aurora MySQL 3.04 或及更新版本，請勿在自訂參數群組中將 TLS 通訊協定設定為 1.0 和 1.1。

對於 Aurora MySQL 3.04.0 及更新版本，預設設定為 TLS 1.3 和 TLS 1.2。

您可以使用 `tls_version` 資料庫叢集參數來指示允許的通訊協定版本。大多數用戶端工具或資料庫驅動程式都有類似的用戶端參數 某些較舊的用戶端可能不支援較新的 TLS 版本。預設情況下，資料庫叢集會嘗試使用伺服器和用戶端配置允許的最高 TLS 通訊協定版本。

將 `tls_version` 資料庫叢集參數設定為下值之一：

- `TLSv1.3`
- `TLSv1.2`
- `TLSv1.1`
- `TLSv1`

您也可以設定 tls_version 參數作為逗號分隔的字符串列表。如果您想要同時使用 TLS 1.2 和 TLS 1.3 通訊協定，tls_version 參數必須包括從最低到最高協議的所有協議。在這種情況下，tls_version 設定為：

```
tls_version=TLSv1.2,TLSv1.3
```

如需修改資料庫叢集參數群組中的參數之相關資訊，請參閱 在 Amazon Aurora 中修改資料庫叢集參數群組中的參數。如果您使用 tls_version AWS CLI 修改資料庫叢集參數，ApplyMethod則必須將設定為 pending-reboot。當套用方法是 pending-reboot，在停止並重新啟動與參數群組相關聯的資料庫叢集之後，都會套用至參數變更。

為 Aurora MySQL 資料庫叢集的連線設定密碼套件

透過使用可設定的密碼套件，您可以更進一步控制資料庫連線的安全性。您可以指定要允許的密碼套件清單，以保護您資料庫的用戶端 TLS 連線安全。您可以使用可設定的密碼套件來控制資料庫伺服器接受的連線加密。這樣做可避免使用不安全或已作廢的密碼。

Aurora MySQL 3 版和 Aurora MySQL 2 版支援可設定的密碼套件。要指定用於加密連線的許可 TLS 1.2、TLS 1.1、TLS 1.0 密碼列表，請修改 ssl_cipher 叢集參數。在使用 AWS 管理主控台、 AWS CLI或 RDS API 的叢集參數群組中設定 ssl_cipher 參數。

將 ssl_cipher 參數設定為 TLS 版本以逗號分隔的密碼值字串。對於用戶端應用程式，您可在連線至資料庫時使用 --ssl-cipher 選項指定用於加密連線的密碼套件。如需有關連線至資料庫的詳細資訊，請參閱 連接至 Amazon Aurora MySQL 資料庫叢集。

從 Aurora MySQL 3.04.0 及更新版本開始，您可以指定 TLS 1.3 加密套件。若要指定許可的 TLS 1.3 加密套件，請修改 tls_ciphersuites 參數群組中的參數。TLS 1.3 已減少可用加密套件的數量，因為命名慣例中的變更會移除金鑰交換機制和使用的憑證。將 tls_ciphersuites 設定為以逗號分隔的 TLS 1.3 密碼值字串。

下表顯示支援的密碼及 TLS 加密通訊協定，及每個密碼的有效 Aurora MySQL 引擎版本。

加密	加密通訊協定	支援的 Aurora MySQL 版本
ECDHE-RSA-AES128-SHA	TLS 1.0	3.04.0 版和更新版本、2.11.0 版和更新版本
ECDHE-RSA-AES128-SHA256	TLS 1.2	3.04.0 版和更新版本、2.11.0 版和更新版本

加密	加密通訊協定	支援的 Aurora MySQL 版本
ECDHE-RSA-AES128-GCM-SHA256	TLS 1.2	3.04.0 版和更新版本、2.11.0 版和更新版本
ECDHE-RSA-AES256-SHA	TLS 1.0	3.04.0 版和更新版本、2.11.0 版和更新版本
ECDHE-RSA-AES256-GCM-SHA384	TLS 1.2	3.04.0 版和更新版本、2.11.0 版和更新版本
ECDHE-RSA-CHACHA20-POLY1305	TLS 1.2	3.04.0 版和更新版本、2.11.0 版和更新版本
ECDHE-ECDSA-AES128-SHA	TLS 1.0	3.04.0 版和更新版本、2.11.0 版和更新版本
ECDHE-ECDSA-AES256-SHA	TLS 1.0	3.04.0 版和更新版本、2.11.0 版和更新版本
ECDHE-ECDSA-AES128-GCM-SHA256	TLS 1.2	3.04.0 版和更新版本、2.11.0 版和更新版本
ECDHE-ECDSA-AES256-GCM-SHA384	TLS 1.2	3.04.0 版和更新版本、2.11.0 版和更新版本
ECDHE-ECDSA-CHACHA20-POLY1305	TLS 1.2	3.04.0 版和更新版本、2.11.0 版和更新版本
TLS_AES_128_GCM_SHA256	TLS 1.3	3.04.0 及更新版本
TLS_AES_256_GCM_SHA384	TLS 1.3	3.04.0 及更新版本
TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	3.04.0 及更新版本

如需修改資料庫叢集參數群組中的參數之相關資訊，請參閱 在 Amazon Aurora 中修改資料庫叢集參數群組中的參數。若您使用 CLI 修改 ssl_cipher 資料庫叢集參數，請務必將 ApplyMethod 設定為 pending-reboot。當套用方法是 pending-reboot，在停止並重新啟動與參數群組相關聯的資料庫叢集之後，都會套用至參數變更。

您也可以使用 describe-engine-default-cluster-parameters CLI 命令決定特定參數群組系列目前支援哪些密碼套件。下列範例顯示如何取得適用於 Aurora MySQL 第 2 版的 ssl_cipher 叢集參數允許值。

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-
mysql5.7

        ...some output truncated...
 {
        "ParameterName": "ssl_cipher",
        "ParameterValue": "ECDHE-RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-
AES128-GCM-SHA256,ECDHE-RSA-AES256-SHA,ECDHE-RSA-AES256-GCM-SHA384,ECDHE-RSA-CHACHA20-
POLY1305,ECDHE-ECDSA-AES256-SHA,ECDHE-ECDSA-CHACHA20-POLY1305,ECDHE-ECDSA-AES256-GCM-
SHA384,ECDHE-ECDSA-AES128-GCM-SHA256,ECDHE-ECDSA-AES128-SHA",
        "Description": "The list of permissible ciphers for connection encryption.",
        "Source": "system",
        "ApplyType": "static",
        "DataType": "list",
        "AllowedValues": "ECDHE-RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-
AES128-GCM-SHA256,ECDHE-RSA-AES256-SHA,ECDHE-RSA-AES256-GCM-SHA384,ECDHE-RSA-CHACHA20-
POLY1305,ECDHE-ECDSA-AES256-SHA,ECDHE-ECDSA-CHACHA20-POLY1305,ECDHE-ECDSA-AES256-GCM-
SHA384,ECDHE-ECDSA-AES128-GCM-SHA256,ECDHE-ECDSA-AES128-SHA",
        "IsModifiable": true,
        "SupportedEngineModes": [
            "provisioned"
        ]
    },
       ...some output truncated...
```

如需有關密碼的詳細資訊，請參閱 MySQL 文件中的 ssl_cipher 變數。如需有關密碼套件格式的詳細資訊，請在 OpenSSL 網站上參閱 openssl-ciphers 列表格式和 openssl-ciphers 字串格式文件。

加密 Aurora MySQL 資料庫叢集的連線

如要使用預設的 mysql 用戶端來加密連線，請使用 --ssl-ca 參數公有金鑰，以啟動 mysql 用戶端，例如：

MySQL 5.7 和 8.0 適用：

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com
--ssl-ca=full_path_to_CA_certificate --ssl-mode=VERIFY_IDENTITY
```

MySQL 5.6 適用：

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com
--ssl-ca=full_path_to_CA_certificate --ssl-verify-server-cert
```

用憑證授權單位 (CA) 憑證的完整路徑取代 *full_path_to_CA_certificate*。如需有關下載憑證的資訊，請參閱 使用 SSL/TLS 加密與資料庫叢集的連線。

您可以要求特定使用者帳戶使用 TLS 連線。例如，您可以根據 MySQL 版本，使用下列任一陳述式，要求使用者帳戶 `encrypted_user` 使用 TLS 連線。

MySQL 5.7 和 8.0 適用：

```
ALTER USER 'encrypted_user'@'%' REQUIRE SSL;
```

MySQL 5.6 適用：

```
GRANT USAGE ON *.* TO 'encrypted_user'@'%' REQUIRE SSL;
```

使用 RDS Proxy 時，您會連線到代理端點，而不是一般叢集端點。您可以使用直接連線到 Aurora 資料庫叢集一樣的方式，將 SSL/TLS 做為連線至代理的必要或選用條件。如需有關使用 RDS Proxy 的資訊，請參閱 Amazon RDS Proxy for Aurora。

> **Note**
>
> 如需有關對 MySQL 使用 TLS 連線的詳細資訊，請參閱 MySQL 說明文件。

將應用程式更新為使用新的 TLS 憑證來連線至 Aurora MySQL 資料庫叢集

自 2023 年 1 月 13 日起，Amazon RDS 已發佈新的憑證認證機構 (CA) 憑證，使用 Transport Layer Security (TLS) 來連線至 Aurora 資料庫叢集。接下來，您可以找到更新應用程式使用新憑證的相關資訊。

本主題可協助您判斷任何用戶端應用程式是否使用 TLS 連線至您的資料庫叢集。若是如此，您可以進一步檢查那些應用程式是否需要驗證憑證才能連線。

> **Note**
>
> 有些應用程式設定為只有在成功驗證伺服器上的憑證時，才能連線至 Aurora MySQL 資料庫叢集。
> 對於這些應用程式，您必須更新用戶端應用程式信任存放區來包含新的 CA 憑證。

更新用戶端應用程式信任存放區中的 CA 憑證之後，您就可以在資料庫叢集輪換憑證。強烈建議先在開發或預備環境中測試這些步驟，再於生產環境中實作。

如需憑證輪換的詳細資訊，請參閱輪換您的 SSL/TLS 憑證。如需下載憑證的詳細資訊，請參閱使用 SSL/TLS 加密與資料庫叢集的連線。如需搭配 Aurora MySQL 資料庫叢集使用 /TLS 的相關資訊，請參閱 Aurora MySQL 資料庫叢集的 TLS 連線。

主題

- 判斷任何應用程式是否使用 TLS 連線至 Aurora MySQL 資料庫叢集
- 判斷用戶端是否需要驗證憑證才能連線
- 更新應用程式信任存放區
- 建立 TLS 連線的 Java 程式碼範例

判斷任何應用程式是否使用 TLS 連線至 Aurora MySQL 資料庫叢集

如果您使用 Aurora MySQL 第 2 版 (與 MySQL 5.7 相容)，且效能結構描述已啟用，請執行下列查詢，以檢查連線是否使用 TLS。如需啟用效能結構描述的資訊，請參閱 MySQL 文件中的效能結構描述快速入門。

```
mysql> SELECT id, user, host, connection_type
```

```
        FROM performance_schema.threads pst
        INNER JOIN information_schema.processlist isp
        ON pst.processlist_id = isp.id;
```

在此輸出範例中，可看到您自己的工作階段 (admin) 和以 webapp1 登入的應用程式都使用 TLS。

```
+----+-----------------+------------------+-----------------+
| id | user            | host             | connection_type |
+----+-----------------+------------------+-----------------+
|  8 | admin           | 10.0.4.249:42590 | SSL/TLS         |
|  4 | event_scheduler | localhost        | NULL            |
| 10 | webapp1         | 159.28.1.1:42189 | SSL/TLS         |
+----+-----------------+------------------+-----------------+
3 rows in set (0.00 sec)
```

判斷用戶端是否需要驗證憑證才能連線

您可以檢查 JDBC 用戶端和 MySQL 用戶端是否需要驗證憑證才能連線。

JDBC

以下 MySQL Connector/J 8.0 範例指出一種方式來檢查應用程式的 JDBC 連線屬性，以判斷是否需要有效憑證才能成功連線。如需 MySQL 的所有 JDBC 連線選項的詳細資訊，請參閱 MySQL 文件中的組態屬性。

使用 MySQL Connector/J 8.0 時，如果連線屬性的 sslMode 設為 VERIFY_CA 或 VERIFY_IDENTITY，則需要以伺服器 CA 憑證來驗證 TLS 連線，如下列範例所示。

```
Properties properties = new Properties();
properties.setProperty("sslMode", "VERIFY_IDENTITY");
properties.put("user", DB_USER);
properties.put("password", DB_PASSWORD);
```

> **Note**
>
> 如果您使用 MySQL Java Connector v5.1.38 或更高版本，或 MySQL Java Connector v8.0.9 或更高版本來連線至您的資料庫，即使您尚未明確設定應用程式在連線到資料庫時使用 TLS，這些用戶端驅動程式會預設為使用 TLS。此外，使用 TLS 時，它們會執行部分憑證驗證，如果資料庫伺服器憑證已過期，則無法連線。

MySQL

以下 MySQL 用戶端範例指出兩種方式來檢查指令碼的 MySQL 連線，以判斷是否需要有效憑證才能成功連線。如需 MySQL 用戶端所有連線選項的詳細資訊，請參閱 MySQL 文件中的加密連線的用戶端組態。

使用 MySQL 5.7 或 MySQL 8.0 用戶端時，如果您將 `--ssl-mode` 選項指定為 `VERIFY_CA` 或 `VERIFY_IDENTITY`，則需要以伺服器 CA 憑證來驗證 TLS 連線，如下列範例所示。

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-mode=VERIFY_CA
```

使用 MySQL 5.6 用戶端時，如果您指定 `--ssl-verify-server-cert` 選項，則需要以伺服器 CA 憑證來驗證 SSL 連線，如下列範例所示。

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-verify-server-cert
```

更新應用程式信任存放區

如需為 MySQL 應用程式更新信任存放區的資訊，請參閱 MySQL 文件中的安裝 SSL 憑證。

> **Note**
>
> 更新信任存放區時，除了新增憑證，您還可以保留舊憑證。

為 JDBC 更新應用程式信任存放區

您可以為使用 JDBC 建立 TLS 連線的應用程式更新信任存放區。

如需下載根憑證的資訊，請參閱 使用 SSL/TLS 加密與資料庫叢集的連線。

如需匯入憑證的範例指令碼，請參閱 將憑證匯入信任存放區的範例指令碼。

如果您在應用程式中使用 mysql JDBC 驅動程式，請在應用程式中設定下列屬性。

```
System.setProperty("javax.net.ssl.trustStore", certs);
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

> **Note**
>
> 指定此處所顯示提示以外的密碼，作為安全最佳實務。

啟動應用程式時，設定下列屬性。

```
java -Djavax.net.ssl.trustStore=/path_to_truststore/MyTruststore.jks -
Djavax.net.ssl.trustStorePassword=my_truststore_password com.companyName.MyApplication
```

建立 TLS 連線的 Java 程式碼範例

下列程式碼範例示範如何使用 JDBC 設定 SSL 連線，以驗證伺服器憑證。

```
public class MySQLSSLTest {

        private static final String DB_USER = "user name";
        private static final String DB_PASSWORD = "password";
        // This key store has only the prod root ca.
        private static final String KEY_STORE_FILE_PATH = "file-path-to-keystore";
        private static final String KEY_STORE_PASS = "keystore-password";

    public static void test(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver");


        System.setProperty("javax.net.ssl.trustStore", KEY_STORE_FILE_PATH);
        System.setProperty("javax.net.ssl.trustStorePassword", KEY_STORE_PASS);

        Properties properties = new Properties();
        properties.setProperty("sslMode", "VERIFY_IDENTITY");
        properties.put("user", DB_USER);
        properties.put("password", DB_PASSWORD);

        Connection connection = DriverManager.getConnection("jdbc:mysql://jagdeeps-ssl-
test.cni62e2e7kwh.us-east-1.rds.amazonaws.com:3306",properties);
        Statement stmt=connection.createStatement();

        ResultSet rs=stmt.executeQuery("SELECT 1 from dual");

        return;
    }
```

```
}
```

> **Important**
>
> 在確定了資料庫連線使用 TLS 並已更新應用程式信任存放區之後，您可以將資料庫更新為使用 rds-ca-rsa2048-g1 憑證。如需說明，請參閱透過修改資料庫執行個體來更新憑證授權機構憑證中的步驟 3。

針對 Aurora MySQL 使用 Kerberos 身分驗證

您可以使用 Kerberos 身分驗證來在使用者連線到您的 Aurora MySQL 資料庫叢集時對其進行身分驗證。若要這麼做，請將資料庫叢集設定為使用 AWS Directory Service for Microsoft Active Directory 進行 Kerberos 身分驗證。AWS Directory Service for Microsoft Active Directory 也被稱為 AWS Managed Microsoft AD。其是一個可與 Directory Service 搭配使用的功能。如需進一步了解，請參閱《AWS Directory Service 管理指南》中的什麼是 Directory Service？。

請先建立 AWS Managed Microsoft AD 目錄來存放使用者登入資料。然後，將 Active Directory 網域和其他資訊提供給 Aurora MySQL 資料庫叢集。當使用者向 Aurora MySQL 資料庫叢集進行驗證時，身分驗證請求會轉送到 AWS Managed Microsoft AD 目錄。

將您的所有登入資料保留在相同目錄可以節省您的時間和精力。透過這種方式，這樣您就有一個集中的位置來存放及管理多個資料庫叢集的登入資料。使用目錄也可以改善您的整體安全性描述檔。

除此之外，您也可以從自己的內部部署 Microsoft Active Directory 存取登入資料。若要執行這項操作，請建立信任網域關聯，讓 AWS Managed Microsoft AD 目錄信任您的內部部署 Microsoft Active Directory。如此一來 ，使用者就可以透過在存取您內部部署網路的工作負載時的相同 Windows 單一登入 (SSO) 體驗，來存取 Aurora MySQL 資料庫叢集。

資料庫可以使用 Kerberos、AWS Identity and Access Management (IAM)，或同時使用 Kerberos 和 IAM 身分驗證。不過，因為 Kerberos 和 IAM 身分驗證提供了不同的身分驗證方法，所以特定使用者只能使用一種或其他身分驗證方法登入資料庫，但不能同時使用兩者。如需 IAM 身分驗證的詳細資訊，請參閱 IAM 資料庫身分驗證。

內容

- Aurora MySQL 資料庫叢集的 Kerberos 身分驗證概觀
- Aurora MySQL 的 Kerberos 身分驗證限制
- 為 Aurora MySQL 資料庫叢集設定 Kerberos 身分驗證
  - 步驟 1：使用 建立目錄 AWS Managed Microsoft AD
  - 步驟 2：(選擇性) 為內部部署 Active Directory 建立信任
  - 步驟 3：建立供 Amazon Aurora 使用的 IAM 角色
  - 步驟 4：建立和設定使用者
  - 步驟 5：建立或修改 Aurora MySQL 資料庫叢集
  - 步驟 6：建立使用 Kerberos 身分驗證的 Aurora MySQL 使用者
    - 修改現有的 Aurora MySQL 登入

- 第 7 步：設定 MySQL 用戶端
- 步驟 8：(選用) 設定不區分大小寫的使用者名稱比較
- 使用 Kerberos 身分驗證連線至 Aurora MySQL
  - 使用 Aurora MySQL Kerberos 登入，以連線至資料庫叢集
  - 使用 Aurora 全域資料庫進行 Kerberos 身分驗證
  - 從 RDS for MySQL 遷移至 Aurora MySQL
  - 防止票證快取
  - Kerberos 身分驗證的記錄
- 管理網域中的資料庫叢集
  - 了解網域成員資格

Aurora MySQL 資料庫叢集的 Kerberos 身分驗證概觀

若要設定 Aurora MySQL 資料庫叢集的 Kerberos 身分驗證，請完成下列一般步驟。稍後會提供這些步驟的詳細說明。

1. 使用 AWS Managed Microsoft AD 來建立 AWS Managed Microsoft AD 目錄。您可以使用 AWS 管理主控台、AWS CLI 或 Directory Service 來建立目錄。如需更多說明，請參閱《AWS Directory Service 管理指南》中的建立您的 AWS Managed Microsoft AD 目錄。
2. 建立使用受管 IAM 政策 `AmazonRDSDirectoryServiceAccess` 的 AWS Identity and Access Management (IAM) 角色。該角色允許 Amazon Aurora 對您的目錄進行呼叫。

   針對可允許存取權的角色，必須在 AWS 區域 中啟用您 AWS 帳戶的 AWS Security Token Service (AWS STS) 端點。AWS STS 端點在所有 AWS 區域 中預設為啟用，您無須採取任何進一步的動作就可以使用這些端點。如需詳細資訊，請參閱《IAM 使用者指南》中的在 AWS 區域 中啟用與停用 AWS STS。
3. 使用 Microsoft Active Directory 工具，在 AWS Managed Microsoft AD 目錄中建立和設定使用者。如需在 Active Directory 建立使用者的詳細資訊，請參閱《AWS 管理指南》中的管理 AWS Directory Service 受管 Microsoft AD 中的使用者和群組。
4. 建立或修改 Aurora MySQL 資料庫叢集。如果您在建立請求中使用 CLI 或 RDS API，請使用 `Domain` 參數指定網域識別符。使用您在建立目錄時產生的 `d-*` 識別符，以及您建立的 IAM 角色名稱。

   如果您修改了現有的 Aurora MySQL 資料庫叢集以使用 Kerberos 身分驗證，將設定資料庫叢集的網域和 IAM 角色參數。在與網域目錄相同的 VPC 中尋找資料庫叢集。

5. 使用 Amazon RDS 主要使用者登入資料來連線到 Aurora MySQL 資料庫叢集。使用 步驟 6：建立使用 Kerberos 身分驗證的 Aurora MySQL 使用者 中的指示，在 Aurora MySQL 中建立資料庫使用者。

   您透過這種方式建立的使用者可以使用 Kerberos 身分驗證登入 Aurora MySQL 資料庫叢集。如需更多詳細資訊，請參閱 使用 Kerberos 身分驗證連線至 Aurora MySQL。

若要使用內部部署或自行託管的 Microsoft Active Directory 來使用 Kerberos 身分驗證，請建立樹系信任。樹系信任是兩個網域群組之間的信任關係。信任可以是單向或雙向。如需使用 Directory Service 設定樹系信任的詳細資訊，請參閱《AWS Directory Service 管理指南》中的何時建立信任關係。

Aurora MySQL 的 Kerberos 身分驗證限制

下列限制適用於 Aurora MySQL 的 Kerberos 身分驗證：

- Aurora MySQL 的 3.03 及更高版本支援 Kerberos 身分驗證。

  如需 AWS 區域 支援的相關資訊，請參閱 使用 Aurora MySQL 進行 Kerberos 身分驗證。
- 若要搭配使用 Kerberos 身分驗證與 Aurora MySQL，您的 MySQL 用戶端或連接器必須在 Unix 平台上使用 8.0.26 或更高版本，在 Windows 上使用 8.0.27 或更高版本。否則，用戶端 `authentication_kerberos_client` 外掛程式將不可用，您便無法進行身分驗證。
- Aurora MySQL 僅支援 AWS Managed Microsoft AD。不過，您可以將 Aurora MySQL 資料庫叢集加入至相同 AWS 區域 中，不同帳戶所擁有的共用受管 Microsoft AD 網域。

  您也可以使用您自己的內部部署 Active Directory。如需更多詳細資訊，請參閱 步驟 2：(選擇性) 為內部部署 Active Directory 建立信任。
- 使用 Kerberos 驗證從 MySQL 用戶端或 Windows 作業系統上驅動程式連線至 Aurora MySQL 叢集的使用者時，資料庫使用者名稱的字元大小寫預設必須與 Active Directory 中的使用者大小寫相符。例如，若 Active Directory 中的使用者顯示為 `Admin`，則資料庫使用者名稱必須為 `Admin`。

  不過，您現在可以搭配 `authentication_kerberos` 外掛程式使用不區分大小寫的使用者名稱比較。如需更多詳細資訊，請參閱 步驟 8：(選用) 設定不區分大小寫的使用者名稱比較。
- 開啟此功能後，您必須重新啟動讀取器資料庫執行個體才能安裝 `authentication_kerberos` 外掛程式。
- 不支援 `authentication_kerberos` 外掛程式的資料庫執行個體複寫可能會導致複寫失敗。
- 若要讓 Aurora 全域資料庫使用 Kerberos 身分驗證，您必須為全域資料庫中的每個資料庫叢集進行設定。

- 網域名稱必須少於 62 個字元。
- 開啟 Kerberos 身分驗證後，請勿修改資料庫叢集連接埠。若您修改連接埠，Kerberos 身分驗證會無法運作。

為 Aurora MySQL 資料庫叢集設定 Kerberos 身分驗證

使用 AWS Managed Microsoft AD 設定 Aurora MySQL 資料庫叢集的 Kerberos 身分驗證。若要設定 Kerberos 身分驗證，請遵循下列步驟。

主題

- 步驟 1：使用 建立目錄 AWS Managed Microsoft AD
- 步驟 2：(選擇性) 為內部部署 Active Directory 建立信任
- 步驟 3：建立供 Amazon Aurora 使用的 IAM 角色
- 步驟 4：建立和設定使用者
- 步驟 5：建立或修改 Aurora MySQL 資料庫叢集
- 步驟 6：建立使用 Kerberos 身分驗證的 Aurora MySQL 使用者
- 第 7 步：設定 MySQL 用戶端
- 步驟 8：(選用) 設定不區分大小寫的使用者名稱比較

步驟 1：使用 建立目錄 AWS Managed Microsoft AD

Directory Service 在 AWS 雲端中建立全受管 Active Directory。當您建立 AWS Managed Microsoft AD 目錄時， 會代表您 Directory Service 建立兩個網域控制站和網域名稱系統 (DNS) 伺服器。目錄伺服器是在 VPC 的不同子網路中建立。此備援有助於確保即使發生故障，您仍然可以存取目錄。

當您建立 AWS Managed Microsoft AD 目錄時， 會代表您 Directory Service 執行下列任務：

- 在 VPC 內設定 Active Directory。
- 建立含有使用者名稱 `Admin` 與指定密碼的目錄管理員帳戶。您可以使用此帳戶來管理目錄。

> **Note**
>
> 請務必儲存此密碼。 Directory Service 不會儲存它。您可以重設此密碼，但是無法擷取此密碼。

- 建立目錄控制器的安全群組。

當您啟動 時 AWS Managed Microsoft AD， 會 AWS 建立組織單位 (OU)，其中包含目錄的所有物件。此 OU 有您在建立目錄時所輸入的 NetBIOS 名稱。它位於網域根目錄中，由 擁有和管理 AWS。

使用 AWS Managed Microsoft AD 目錄建立Admin的帳戶具有 OU 最常見管理活動的許可，包括：

- 建立、更新或刪除使用者
- 將資源 (例如檔案或列印伺服器) 新增至您的網域，然後對您 OU 中的使用者指派這些資源的許可
- 建立額外的 OU 和容器
- 委派授權
- 從 Active Directory 資源回收筒還原已刪除的物件
- 在 Active Directory Web 服務上執行 AD 和 DNS Windows PowerShell 模組

Admin 帳戶也有權執行下列全網域活動：

- 管理 DNS 組態 (新增、移除或更新記錄、區域和轉寄站)
- 檢視 DNS 事件日誌
- 檢視安全事件日誌

使用 建立目錄 AWS Managed Microsoft AD

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/directoryservicev2/ 開啟 Directory Service 主控台。
2. 在導覽窗格中，選擇 Directories (目錄)，然後選擇 Set up directory (設定目錄)。
3. 選擇 AWS Managed Microsoft AD. AWS Managed Microsoft AD 是您目前可與 Amazon RDS 搭配使用的唯一選項。
4. 輸入下列資訊：

   目錄 DNS 名稱

   目錄的完全合格名稱，例如 **corp.example.com**。

   目錄 NetBIOS 名稱

   目錄的簡短名稱，例如：**CORP**。

   目錄描述

   (選用) 目錄的描述。

管理員密碼

目錄管理員的密碼。目錄建立程序會建立含有使用者名稱 Admin 與這組密碼的管理員帳戶。

目錄管理員密碼不得包含 "admin" 字組。密碼區分大小寫，長度須為 8 至 64 個字元。至少須有一位字元屬於以下四種類型中的三類：

- 小寫字母 (a–z)
- 大寫字母 (A–Z)
- 數字 (0–9)
- 非英數字元 (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

Confirm password (確認密碼)

重新輸入的管理員密碼。

5. 選擇下一步。
6. 在 Networking (聯網) 區段輸入以下資訊，然後選擇 Next (下一步)。

VPC

目錄的 VPC。在相同的 VPC 中建立 Aurora MySQL 資料庫叢集。

子網路

目錄伺服器的子網路。這兩個子網路必須位於不同的可用區域。

7. 檢閱目錄資訊，並進行必要的變更。若資訊無誤，請選擇 Create directory (建立目錄)。
建立期間的目錄詳細資訊頁面

建立目錄需要幾分鐘的時間。成功建立時，Status (狀態) 值會變更為 Active (作用中)。

如要查看您目錄的資訊，請在目錄清單中選擇目錄名稱。請記下目錄 ID，因為您在建立或修改 Aurora MySQL 資料庫叢集時將需要這個值。
在目錄詳細資訊頁面上的目錄 ID

步驟 2：(選擇性) 為內部部署 Active Directory 建立信任

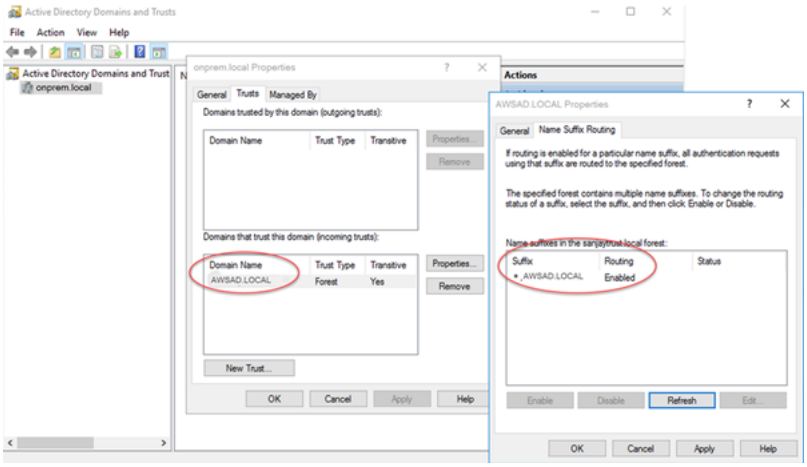
如果您不打算使用自己的內部部署 Microsoft Active Directory，請跳至步驟 3：建立供 Amazon Aurora 使用的 IAM 角色.

若要搭配現場部署 Active Directory 使用 Kerberos 身分驗證，您需要使用現場部署 Microsoft Active Directory 與 AWS Managed Microsoft AD 目錄 （ 在 中建立 ） 之間的樹系信任來建立信任網域關

係步驟 1：使用 建立目錄 AWS Managed Microsoft AD。信任可以是單向的，其中 AWS Managed Microsoft AD 目錄信任內部部署 Microsoft Active Directory。信任也可以是雙向，其中兩個 Active Directory 互相信任。如需使用 設定信任的詳細資訊 Directory Service，請參閱《 AWS Directory Service 管理指南》中的何時建立信任關係。

> **Note**
>
> 如果您使用內部部署 Microsoft Active Directory：
>
> - Windows 用戶端無法使用 Aurora 自訂端點進行連線。如需詳細資訊，請參閱 Amazon Aurora 端點連線。
> - 針對全球資料庫：
>   - Windows 用戶端只能使用全域資料庫主要 AWS 區域 中的執行個體端點或叢集端點進行連線。
>   - Windows 用戶端無法使用次要叢集端點進行連線 AWS 區域。

請確定您的內部部署 Microsoft Active Directory 網域名稱包含對應至新建立之信任關係的 DNS 尾碼路由。以下螢幕擷取畫面顯示了一個範例。



步驟 3：建立供 Amazon Aurora 使用的 IAM 角色

若要讓 Amazon Aurora Directory Service 為您呼叫 ，您需要使用 受管 IAM 政策 的 AWS Identity and Access Management (IAM) 角色`AmazonRDSDirectoryServiceAccess`。此角色允許 Aurora 呼叫 Directory Service。

當您使用 建立資料庫叢集 AWS 管理主控台並擁有 `iam:CreateRole`許可時，主控台會自動建立此角色。在此情況下，角色名稱為 `rds-directoryservice-kerberos-access-role`。否則，您必須

手動建立 IAM 角色。當您建立此 IAM 角色時，請選擇 `Directory Service`，然後將 AWS 受管政策連接到`AmazonRDSDirectoryServiceAccess`該角色。

如需為服務建立 IAM 角色的詳細資訊，請參閱《IAM 使用者指南》中的建立角色以將許可委派給 AWS 服務。

您可以選擇性地建立具有必要許可的政策，而不是使用受管 IAM 政策 `AmazonRDSDirectoryServiceAccess`。在此情況下，IAM 角色必須有以下 IAM 信任政策。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

此角色也須具有下列 IAM 角色政策：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
```

```
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

步驟 4：建立和設定使用者

您可以使用 Active Directory 使用者和運算集區來建立使用者。這個工具是 Active Directory Domain Services 和 Active Directory 輕量型目錄服務工具的一部分。使用者代表具有目錄存取權的個人或實體。

若要在 Directory Service 目錄中建立使用者，您可以使用已加入 Directory Service 目錄的 Microsoft Windows 為基礎的內部部署或 Amazon EC2 執行個體。您必須以具有建立使用者之許可的使用者身分來登入執行個體。如需詳細資訊，請參閱《 AWS Managed Microsoft AD Directory Service 管理指南》中的管理AWS 中的使用者和群組。

步驟 5：建立或修改 Aurora MySQL 資料庫叢集

建立或修改 Aurora MySQL 資料庫叢集以搭配您的目錄使用。您可以使用 主控台 AWS CLI或 RDS API，將資料庫叢集與目錄建立關聯。您可採用下列其中一種方式來執行此動作：

- 使用主控台、 create-db-cluster CLI 命令或 CreateDBCluster RDS API 操作，建立新的 Aurora MySQL 資料庫叢集。

  如需說明，請參閱建立 Amazon Aurora 資料庫叢集。
- 使用主控台、modify-db-cluster CLI 命令或 ModifyDBCluster RDS API 操作，修改現有的 Aurora MySQL 資料庫叢集。

  如需說明，請參閱修改 Amazon Aurora 資料庫叢集。
- 使用主控台、restore-db-cluster-from-snapshot CLI 命令或 RestoreDBClusterFromSnapshot RDS API 操作，從資料庫快照還原 Aurora MySQL 資料庫叢集。

  如需說明，請參閱從資料庫叢集快照還原。
- 使用主控台、 restore-db-cluster-to-point-in-time CLI 命令或 RestoreDBClusterToPointInTime RDS API 操作，將 Aurora MySQL 資料庫叢集還原至某個時間點。

如需說明，請參閱將資料庫叢集還原至指定時間。

僅有 VPC 中的 Aurora MySQL 資料庫叢集支援 Kerberos 身分驗證。資料庫叢集可在與目錄相同的 VPC 中，或在不同 VPC 中。資料庫叢集的 VPC 必須具備能允許對目錄進行傳出通訊的 VPC 安全群組。

主控台

使用主控台建立、修改或還原資料庫叢集時，請在 Database authentication (資料庫身分驗證) 區段中選擇 Kerberos authentication (Kerberos 身分驗證)。選擇 Browse Directory (瀏覽目錄) 並選取目錄，或是選擇 Create a new directory (建立新目錄)。
建立資料庫叢集時的 Kerberos 身分驗證設定

AWS CLI

當您使用 AWS CLI 或 RDS API 時，請將資料庫叢集與目錄建立關聯。資料庫叢集需要下列參數，才能使用您建立的網域目錄：

- 針對 `--domain` 參數，使用您建立目錄時產生的網域識別碼 ("d-*" 識別碼)。
- 針對 `--domain-iam-role-name` 參數，使用您建立的規則，其會使用受管 IAM 政策 `AmazonRDSDirectoryServiceAccess`。

例如，下列 CLI 命令會修改資料庫叢集來使用目錄。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --domain d-ID \
    --domain-iam-role-name role-name
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --domain d-ID ^
    --domain-iam-role-name role-name
```

> **Important**
>
> 如果您修改資料庫叢集，以啟用 Kerberos 身分驗證，請在進行變更後重新啟動讀取器資料庫執行個體。

步驟 6：建立使用 Kerberos 身分驗證的 Aurora MySQL 使用者

資料庫叢集會加入 AWS Managed Microsoft AD 網域。因此，您可以從您網域中的 Active Directory 使用者中，建立 Aurora MySQL 使用者。資料庫許可會透過標準 Aurora MySQL 許可管理，並從這些使用者授予及撤銷。

您可以允許 Active Directory 使用者與 Aurora MySQL 進行身分驗證。若要執行此作業，請先使用 Amazon RDS 主要使用者登入資料連線到 Aurora MySQL 資料庫叢集，方法與其他任何的資料庫叢集相同。登入之後，請在 Aurora MySQL 中建立具有 Kerberos 身分驗證的外部身分驗證使用者，如下所示：

```
CREATE USER user_name@'host_name' IDENTIFIED WITH 'authentication_kerberos' BY
 'realm_name';
```

- 請將 *user_name* 換成使用者名稱。來自您網域的使用者 (人員和應用程式兩者) 現在可以使用 Kerberos 身分驗證從加入網域的用戶端機器連線至資料庫叢集。
- 以主機名稱取代 *host_name*。您可以使用 % 做為萬用字元。您也可以使用特定的 IP 地址做為主機名稱。
- 以網域的目錄領域名稱取代 *realm_name*。領域名稱通常是大寫的 DNS 網域名稱，例如 `CORP.EXAMPLE.COM`。領域是使用相同 Kerberos 金鑰發佈中心的系統群組。

下列範例會建立名為 `Admin` 的資料庫使用者，該名稱會以領域名稱 `MYSQL.LOCAL` 對 Active Directory 進行驗證。

```
CREATE USER Admin@'%' IDENTIFIED WITH 'authentication_kerberos' BY 'MYSQL.LOCAL';
```

修改現有的 Aurora MySQL 登入

您也可以使用下列語法，修改現有的 Aurora MySQL 登入，以使用 Kerberos 身分驗證：

```
ALTER USER user_name IDENTIFIED WITH 'authentication_kerberos' BY 'realm_name';
```

第 7 步：設定 MySQL 用戶端

若要設定 MySQL 用戶端，請執行下列步驟：

1. 建立可指向網域的 `krb5.conf` 檔案 (或同等檔案)。
2. 確認流量可以在用戶端主機與 之間流動 Directory Service。請使用 Netcat 等網路公用程式檢查以下各項：
   - 確認透過 DNS 傳送至連接埠 53 的流量。
   - 確認透過 TCP/UDP 傳送至連接埠 53 和 Kerberos 的流量，其中包括用於 Directory Service的連接埠 88 和 464。
3. 確定流量可透過資料庫連接埠在用戶端主機和資料庫執行個體之間往來。例如，您可以使用 `mysql` 來連接和存取資料庫。

以下是 的範例`krb5.conf`內容 AWS Managed Microsoft AD。

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

下列是內部部署 Microsoft Active Directory 的範例 `krb5.conf` 內容。

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
 ONPREM.COM = {
  kdc = onprem.com
  admin_server = onprem.com
 }
[domain_realm]
```

```
.example.com = EXAMPLE.COM
example.com = EXAMPLE.COM
.onprem.com = ONPREM.COM
onprem.com = ONPREM.COM
.rds.amazonaws.com = EXAMPLE.COM
.amazonaws.com.cn = EXAMPLE.COM
.amazon.com = EXAMPLE.COM
```

步驟 8：(選用) 設定不區分大小寫的使用者名稱比較

根據預設，MySQL 資料庫使用者名稱的字元大小寫必須符合 Active Directory 登入的字元大小寫。不過，您現在可以搭配 `authentication_kerberos` 外掛程式使用不區分大小寫的使用者名稱比較。若要這麼做，請將 `authentication_kerberos_caseins_cmp` 資料庫叢集參數設定為 `true`。

使用不區分大小寫的使用者名稱比較

1. 建立自訂資料庫叢集參數群組。請遵循在 Amazon Aurora 中建立資料庫叢集參數群組中的程序。
2. 編輯新的參數群組，將 `authentication_kerberos_caseins_cmp` 的值設定為 `true`。請遵循在 Amazon Aurora 中修改資料庫叢集參數群組中的參數中的程序。
3. 將資料庫叢集參數群組與您的 Aurora MySQL 資料庫叢集建立關聯。請遵循將資料庫叢集參數群組與 Amazon Aurora 中的資料庫叢集建立關聯中的程序。
4. 重新啟動資料庫叢集。

使用 Kerberos 身分驗證連線至 Aurora MySQL

為避免發生錯誤，請在 Unix 平台上使用 8.0.26 或更高版本的 MySQL 用戶端，在 Windows 上則使用 8.0.27 或更高版本。

使用 Aurora MySQL Kerberos 登入，以連線至資料庫叢集

若要使用 Kerberos 身分驗證連線至 Aurora MySQL，您可以使用 步驟 6：建立使用 Kerberos 身分驗證的 Aurora MySQL 使用者 的指示所建立的資料庫使用者身分登入。

在命令提示字元中，連線至與 Aurora MySQL 資料庫叢集相關聯的其中一個端點。當系統提示您輸入密碼時，請輸入與該使用者名稱相關聯的 Kerberos 密碼。

當您透過 Kerberos 進行身分驗證時，如果票證授予票證 (TGT) 尚未存在，系統會產生一份 TGT。`authentication_kerberos` 外掛程式會使用 TGT 來取得服務票證，然後將其顯示給 Aurora MySQL 資料庫伺服器。

您可以在 Windows 或 Unix 上，透過 Kerberos 身分驗證，以 MySQL 用戶端連線至 Aurora MySQL。

Unix

您可以使用下列其中一種方法來連線：

- 手動取得 TGT。在此情況下，您不需要向 MySQL 用戶端提供密碼。
- 直接向 MySQL 用戶端提供 Active Directory 登入密碼。

在 Unix 平台上，MySQL 用戶端 8.0.26 和更新版本可支援用戶端外掛程式。

手動取得 TGT 以進行連線

1. 在命令列界面中，請使用以下命令取得 TGT。

```
kinit user_name
```

2. 使用下列 `mysql` 命令登入資料庫叢集的資料庫執行個體端點。

```
mysql -h DB_instance_endpoint -P 3306 -u user_name -p
```

> **Note**
>
> 如果在資料庫執行個體上輪換 keytab，身分驗證便可能失敗。在此情況下，請重新執行 `kinit` 以取得新的 TGT。

直接連線

1. 在命令列界面中，使用下列 `mysql` 命令登入資料庫叢集的資料庫執行個體端點。

```
mysql -h DB_instance_endpoint -P 3306 -u user_name -p
```

2. 輸入 Active Directory 使用者的密碼。

Windows

在 Windows 上，身分驗證通常會在登入時完成，因此您不需要手動取得 TGT 即可連線至 Aurora MySQL 資料庫叢集。資料庫使用者名稱的大小寫必須符合 Active Directory 中使用者的字元大小寫。例如，若 Active Directory 中的使用者顯示為 `Admin`，則資料庫使用者名稱必須為 `Admin`。

在 Windows 上，MySQL 用戶端 8.0.27 和更新版本可支援用戶端外掛程式。

直接連線

- 在命令列界面中，使用下列 `mysql` 命令登入資料庫叢集的資料庫執行個體端點。

```
mysql -h DB_instance_endpoint -P 3306 -u user_name
```

使用 Aurora 全域資料庫進行 Kerberos 身分驗證

Aurora 全域資料庫可支援 Aurora MySQL 的 Kerberos 身分驗證。若要使用主要資料庫叢集的 Active Directory，來驗證次要資料庫叢集上的使用者，請將 Active Directory 複寫到次要 AWS 區域。您可以使用與主要叢集相同的網域 ID，在次要叢集上開啟 Kerberos 身分驗證。只有企業版的 Active Directory 支援 AWS Managed Microsoft AD 複寫。如需詳細資訊，請參閱《AWS Directory Service 管理指南》中的多區域複寫。

從 RDS for MySQL 遷移至 Aurora MySQL

從啟用 Kerberos 身分驗證的 RDS for MySQL 遷移到 Aurora MySQL 之後，請修改使用 `auth_pam` 外掛程式建立的使用者，以使用 `authentication_kerberos` 外掛程式。例如：

```
ALTER USER user_name IDENTIFIED WITH 'authentication_kerberos' BY 'realm_name';
```

防止票證快取

如果 MySQL 用戶端應用程式啟動時不存在有效 TGT，則應用程式可以取得且快取 TGT。若要防止快取 TGT，請在 `/etc/krb5.conf` 檔案中設定組態參數。

> **Note**
>
> 此組態僅適用於執行 Unix 的用戶端主機，而非執行 Windows 的用戶端主機。

為了防止 TGT 快取

- 請新增 [appdefaults] 區段至 /etc/krb5.conf，如下所示：

```
[appdefaults]
  mysql = {
    destroy_tickets = true
  }
```

Kerberos 身分驗證的記錄

AUTHENTICATION_KERBEROS_CLIENT_LOG 環境變數會設定 Kerberos 身分驗證的記錄層級。您可以使用日誌進行用戶端偵錯。

允許值為 1 至 5。日誌訊息會寫入標準錯誤輸出。下表描述各個記錄層級。

Logging level (記錄層級)	描述
1 或未設定	沒有記錄
2	錯誤訊息
3	錯誤和警告訊息
4	錯誤、警告和資訊訊息
5	錯誤、警告、資訊和偵錯訊息

管理網域中的資料庫叢集

您可以使用 AWS CLI 或 RDS API 來管理資料庫叢集，以及其與受管 Active Directory 的關係。例如，您可以與適用於 Kerberos 身分驗證的 Active Directory 建立關聯，也可以解除關聯 Active Directory 來停用 Kerberos 身分驗證。您也可以將要由某個 Active Directory 於外部進行身分識別的資料庫叢集移至另一個 Active Directory。

例如，使用 Amazon RDS API，您可以執行下列動作：

- 如要重新嘗試為失敗的成員資格啟用 Kerberos 身分驗證，請使用 ModifyDBInstance API 操作並指定目前成員資格的目錄 ID。

- 如要更新成員資格的 IAM 角色名稱，請使用 `ModifyDBInstance` API 操作並指定目前成員資格的目錄 ID，以及新的 IAM 角色。
- 停用資料庫叢集上的 Kerberos 身分驗證，使用 `ModifyDBInstance` API 操作並指定 `none` 做為網域參數。
- 如要在網域之間移動資料庫叢集，請使用 `ModifyDBInstance` API 操作，並指定新網域的網域識別符做為網域參數。
- 如要列出每個資料庫叢集的成員資格，請使用 `DescribeDBInstances` API 操作。

了解網域成員資格

在您建立或修改資料庫叢集之後，該執行個體會成為網域的成員。您可以執行 describe-db-clusters CLI 命令，來檢視資料庫叢集的網域成員資格狀態。資料庫叢集的狀態可以是下列其中一個：

- `kerberos-enabled` – 資料庫叢集已開啟 Kerberos 身分驗證。
- `enabling-kerberos` – AWS 正在此資料庫叢集上啟用 Kerberos 身分驗證。
- `pending-enable-kerberos` – 此資料庫叢集上的 Kerberos 身分驗證啟用處於待處理狀態。
- `pending-maintenance-enable-kerberos` – AWS 將在下次排定的維護時段嘗試在資料庫叢集上啟用 Kerberos 身分驗證。
- `pending-disable-kerberos` – 此資料庫叢集上的 Kerberos 身分驗證停用處於待處理狀態。
- `pending-maintenance-disable-kerberos` – AWS 將在下次排定的維護時段嘗試在資料庫叢集上停用 Kerberos 身分驗證。
- `enable-kerberos-failed` – 有一個組態問題已禁止 AWS 在資料庫叢集上啟用 Kerberos 身分驗證。請在重新發出資料庫叢集修改命令之前檢查並修正您的組態。
- `disabling-kerberos` – AWS 正在此資料庫叢集上停用 Kerberos 身分驗證。

由於網路連線問題或 IAM 角色不正確，請求啟用 Kerberos 身分驗證可能失敗。例如，假設您建立了資料庫叢集或修改了現有的資料庫叢集，並且嘗試啟用 Kerberos 身分驗證失敗。在此情況下，請重新發出修改命令，或修改新建立的資料庫叢集以加入網域。

將資料遷移至 Amazon Aurora MySQL 資料庫叢集

您有幾個選項可從現有的資料庫將資料遷移至 Amazon Aurora MySQL 資料庫叢集。遷移選項還取決於要遷移的資料庫以及要遷移的資料大小。

有兩種不同類型的遷移：實體和邏輯。實體遷移是指使用資料庫檔案的實體副本來遷移資料庫。邏輯遷移是指套用邏輯資料庫變更來完成遷移，例如插入、更新和刪除。

實體遷移有下列優點：

- 實體遷移比邏輯遷移更快，尤其對於大型資料庫。
- 為實體遷移而建立備份時，資料庫效能不受影響。
- 實體遷移可以遷移來源資料庫的所有內容，包括複雜的資料庫元件。

實體遷移有下列限制：

- `innodb_page_size` 參數必須設為預設值 (16KB)。
- `innodb_data_file_path` 參數只能使用一個資料檔案 (預設資料檔案名稱 `"ibdata1:12M:autoextend"`) 來設定。具有兩個資料檔或具有不同名稱之資料檔的資料庫無法使用此方法移轉。

  以下是不允許的檔案名稱範例：`"innodb_data_file_path=ibdata1:50M; ibdata2:50M:autoextend"` 和 `"innodb_data_file_path=ibdata01:50M:autoextend"`。
- `innodb_log_files_in_group` 參數必須設為預設值 (2)。

邏輯遷移有下列優點：

- 您可以遷移資料庫的子集，例如特定的資料表，或資料表的某些部分。
- 無論實體儲存結構如何，資料皆可遷移。

邏輯遷移有下列限制：

- 邏輯遷移通常比實體遷移更慢。
- 複雜的資料庫元件可能使邏輯遷移程序變慢。在某些情況下，複雜的資料庫元件甚至會阻擋邏輯遷移。

下表說明您的選項及各選項的遷移類型。

遷移來源	Migration type (遷移類型)	解決方案
RDS for MySQL 資料庫執行個體	實體	若要從 RDS for MySQL 資料庫執行個體遷移，您可以先建立 MySQL 資料庫執行個體的 Aurora MySQL 僅供讀取複本。當 MySQL 資料庫執行個體和 Aurora MySQL 僅供讀取複本之間的複本延遲為 0 時，您可以指示用戶端應用程式讀取 Aurora 僅供讀取複本，然後停止複寫，使 Aurora MySQL 僅供讀取複本變成用於讀取和寫入的獨立 Aurora MySQL 資料庫叢集。如需詳細資訊，請參閱使用 Aurora 讀取複本，從 RDS for MySQL 資料庫執行個體將資料遷移至 Amazon Aurora MySQL 資料庫叢集。
RDS for MySQL 資料庫快照	實體	您可以直接從 RDS for MySQL 資料庫快照將資料遷移至 Amazon Aurora MySQL 資料庫叢集。如需詳細資訊，請參閱 將 RDS for MySQL 快照遷移至 Aurora。
Amazon RDS 外部的 MySQL 資料庫	邏輯	您可以使用 `mysqldump` 公用程式來建立資料的傾出，然後將該資料匯入現有的 Amazon Aurora MySQL 資料庫叢集。如需詳細資訊，請參閱 使用 mysqldump 從 MySQL 到 Amazon Aurora 的邏輯遷移。 若要在遷移期間從外部 MySQL 資料庫匯出資料庫使用者的中繼資料，您可以使用 MySQL Shell 命令，而不是 `mysqldump` 。如需詳細資訊，請參閱執行個體傾印公用程式、結構描述傾印公用程式和資料表傾印公用程式。

遷移來源	Migration type (遷移類型)	解決方案
		Note 自 MySQL 8.0.34 起，mysqlpump 公用程式已棄用。
Amazon RDS 外部的 MySQL 資料庫	實體	您可以從資料庫將備份檔案複製到 Amazon Simple Storage Service (Amazon S3) 儲存貯體，然後從這些檔案還原 Amazon Aurora MySQL 資料庫叢集。比起使用 `mysqldump` ，此選項遷移資料的速度更快。如需詳細資訊，請參閱使用 Percona XtraBackup 和 Amazon S3 從 MySQL 進行實體遷移。
Amazon RDS 外部的 MySQL 資料庫	邏輯	您可以將資料庫中的資料儲存為文字檔案，再將這些檔案複製到 Amazon S3 儲存貯體。然後，您可以使用 `LOAD DATA FROM S3` MySQL 命令，將該資料載入現有的 Aurora MySQL 資料庫叢集。如需更多詳細資訊，請參閱 從 Amazon S3 儲存貯體中的文字檔案將資料載入 Amazon Aurora MySQL 資料庫叢集。
與 MySQL 不相容的資料庫	Logical (邏輯)	您可以使用 AWS Database Migration Service (AWS DMS)，從與 MySQL 不相容的資料庫遷移資料。如需 AWS DMS 的詳細資訊，請參閱什麼是 AWS Database Migration Service？

Note

如果您要遷移 Amazon RDS 外部的 MySQL 資料庫，則僅在您的資料庫支援 InnoDB 或 MyISAM 資料表空間時，才支援資料表中所述的遷移選項。

如果您要遷移至 Aurora MySQL 的 MySQL 資料庫使用 memcached，請先移除 memcached 再遷移。

您無法從某些舊版 MySQL 8.0 版本 (包括 8.0.11、8.0.13 和 8.0.15) 遷移到 Aurora MySQL 版本 3.05 及更高版本。建議您在遷移之前，先升級至 MySQL 8.0.28 版。

從外部 MySQL 資料庫將資料遷移至 Amazon Aurora MySQL 資料庫叢集

如果您的資料庫支援 InnoDB 或 MyISAM 資料表空間，在將資料遷移至 Amazon Aurora MySQL 資料庫叢集時，您有下列選項：

- 您可以使用 mysqldump 公用程式來建立資料的傾出，然後將該資料匯入現有的 Amazon Aurora MySQL 資料庫叢集。如需更多詳細資訊，請參閱 使用 mysqldump 從 MySQL 到 Amazon Aurora 的邏輯遷移。
- 您可以從資料庫將完整和增量備份檔案複製到 Amazon S3 儲存貯體，然後從這些檔案還原至 Amazon Aurora MySQL 資料庫叢集。比起使用 mysqldump，此選項遷移資料的速度更快。如需更多詳細資訊，請參閱 使用 Percona XtraBackup 和 Amazon S3 從 MySQL 進行實體遷移。

主題

- 使用 Percona XtraBackup 和 Amazon S3 從 MySQL 進行實體遷移
- 使用 mysqldump 從 MySQL 到 Amazon Aurora 的邏輯遷移

使用 Percona XtraBackup 和 Amazon S3 從 MySQL 進行實體遷移

您可以從來源 MySQL 5.7 或 8.0 版資料庫將完整和增量備份檔案複製到 Amazon S3 儲存貯體。然後，您可以從這些檔案還原到具有相同主要資料庫引擎版本的 Amazon Aurora MySQL 資料庫叢集。

此選項比使用 mysqldump 來遷移資料快得多，因為使用 mysqldump 會重新執行所有命令，在新的 Aurora MySQL 資料庫叢集中重新建立來源資料庫的結構描述和資料。經由複製來源 MySQL 資料檔案，Aurora MySQL 可以立即使用這些檔案做為 Aurora MySQL 資料庫叢集的資料。

此外，您還可以在遷移過程中使用二進位日誌複寫來盡可能縮短停機時間。如果您使用二進位日誌複寫，則當資料正在遷移至 Aurora MySQL 資料庫叢集時，外部 MySQL 資料庫仍然可能發生交易。建立 Aurora MySQL 資料庫叢集之後，您可以使用二進位日誌複寫，以同步 Aurora MySQL 資料庫叢集與備份之後發生的交易。當 Aurora MySQL 資料庫叢集與 MySQL 資料庫達到同步時，請完全切換至 Aurora MySQL 資料庫叢集來處理新交易，以便完成遷移。如需詳細資訊，請參閱使用複寫來同步 Amazon Aurora MySQL 資料庫叢集與 MySQL 資料庫。

內容

- 限制及考量
- 開始之前
  - 安裝 Percona XtraBackup

  - 所需的許可
  - 建立 IAM 服務角色
- 備份要還原為 Amazon Aurora MySQL 資料庫叢集的檔案
  - 使用 Percona XtraBackup 來建立完整備份
  - 透過 Percona XtraBackup 來使用增量備份
  - 備份考量
- 從 Amazon S3 儲存貯體還原 Amazon Aurora MySQL 資料庫叢集
- 使用複寫來同步 Amazon Aurora MySQL 資料庫叢集與 MySQL 資料庫
  - 針對加密複寫來設定外部 MySQL 資料庫和 Aurora MySQL 資料庫叢集
  - 同步 Amazon Aurora MySQL 資料庫叢集與外部 MySQL 資料庫
- 縮短實體遷移到 Amazon Aurora MySQL 的時間
  - 不支援的資料表類型
  - 具有不支援權限的使用者帳戶
  - Aurora MySQL 第 3 版中的動態權限
  - 使用「rdsadmin'@'localhost」作為 DEFINER 的預存物件

限制及考量

從 Amazon S3 儲存貯體還原至 Amazon Aurora MySQL 資料庫叢集時，須注意下列限制和考量：

- 您只能將資料遷移至新的資料庫叢集，而非現有資料庫叢集。
- 您必須使用 Percona XtraBackup 將資料備份到 S3。如需詳細資訊，請參閱安裝 Percona XtraBackup。
- Amazon S3 儲存貯體和 Aurora MySQL 資料庫叢集必須位於相同的 AWS 區域。
- 您無法從以下項目還原：
  - 從資料庫叢集快照匯出還原至 Amazon S3。您無法將資料從資料庫叢集快照匯出遷移到 S3 儲存貯體。
  - 加密的來源資料庫，但可以加密要遷移的資料。您也可以在遷移過程中維持不加密資料。
  - MySQL 5.5 或 5.6 資料庫
- Percona Server for MySQL 不支援作為來源資料庫，因為它可以包含 `mysql` 結構描述中的 `compression_dictionary*` 資料表。
- 您無法還原至 Aurora Serverless 資料庫叢集。

- 主要版本或次要版本都不支援回溯遷移。例如，您無法從 MySQL 8.0 版遷移至 Aurora MySQL 第 2 版 (與 MySQL 5.7 相容)，也無法從 MySQL 8.0.32 版遷移至 Aurora MySQL 3.03 版 (與 MySQL Community 8.0.26 版相容)。
- 您無法從某些舊版 MySQL 8.0 版本 (包括 8.0.11、8.0.13 和 8.0.15) 遷移到 Aurora MySQL 版本 3.05 及更高版本。建議您在遷移之前，先升級至 MySQL 8.0.28 版。
- db.t2.micro 資料庫執行個體類別上不支援從 Amazon S3 匯入。不過，您可以先還原至不同的資料庫執行個體類別，稍後再變更資料庫執行個體類別。如需資料庫執行個體類別的詳細資訊，請參閱 Amazon Aurora 資料庫執行個體類別。
- Amazon S3 對上傳至 S3 儲存貯體的檔案大小限制為 5 TB。如果備份檔案超過 5 TB，您必須將備份檔案分割為較小的檔案。
- Amazon RDS 隊上傳至 S3 儲存貯體的檔案數量限制為 1 百萬個。如果資料庫的備份資料 (包括所有完整和增量備份) 超過 1 百萬個檔案，請使用 Gzip (.gz)、tar (.tar.gz) 或 Percona xbstream (.xbstream) 檔案將完整和增量備份檔案儲存在 S3 儲存貯體中。Percona XtraBackup 8.0 只支援壓縮時的 Percona xbstream。
- 若要為每個資料庫叢集提供管理服務，建立資料庫執行個體時，系統會一併建立 `rdsadmin` 使用者。由於這是 RDS 中預留的使用者，因此適用以下限制：
  - 不會匯入使用 `'rdsadmin'@'localhost'` DEFINER 的函數、程序、檢視、事件和觸發。如需詳細資訊，請參閱使用「rdsadmin'@'localhost」作為 DEFINER 的預存物件及Amazon Aurora MySQL 的主要使用者權限。
  - 建立 Aurora MySQL 資料庫叢集時，會一併建立具有所支援最大權限的主要使用者。從備份還原時，若指派任何不支援的權限給要匯入的使用者，這些權限都會在匯入過程中自動移除。

    若要找出可能受此影響的使用者，請參閱 具有不支援權限的使用者帳戶。如需有關 Aurora MySQL 中所支援權限的詳細資訊，請參閱 角色型權限模型。
- 對於 Aurora MySQL 第 3 版，不會匯入動態權限。Aurora 支援的動態權限可在遷移後匯入。如需詳細資訊，請參閱Aurora MySQL 第 3 版中的動態權限。
- 不會遷移 `mysql` 結構描述中使用者建立的資料表。
- `innodb_data_file_path` 參數只能使用一個資料檔案 (預設資料檔案名稱 `ibdata1:12M:autoextend`) 來設定。具有兩個資料檔或具有不同名稱之資料檔的資料庫無法使用此方法移轉。

  以下是不允許的檔案名稱範
  例：`innodb_data_file_path=ibdata1:50M、ibdata2:50M:autoextend` 和 `innodb_data_file_path=ibdata01:50M:autoextend`。
- 如果來源資料庫有資料表是在預設 MySQL 資料目錄外定義，您無法從這個來源資料庫遷移。

- 使用此方法可支援的最大未壓縮備份大小目前限制為 64 TiB。對於壓縮備份，此限制會降低以考量未壓縮空間需求。在這種情況下，支援的最大備份大小為 (`64 TiB – compressed backup size`)。
- Aurora MySQL 不支援匯入 MySQL 及其他外部元件和外掛程式。
- Aurora MySQL 不會從資料庫還原任何內容。建議您從來源 MySQL 資料庫儲存資料庫結構描述和下列項目的值，然後在還原的 Aurora MySQL 資料庫叢集建立之後，將這些項目新增至其中：
  - 使用者帳戶
  - 函數
  - 預存程序
  - 時區資訊。時區資訊是從 Aurora MySQL 資料庫叢集的本機作業系統載入。如需詳細資訊，請參閱Amazon Aurora 資料庫叢集的本機時區。

開始之前

您必須執行下列動作，才能將資料複製到 Amazon S3 儲存貯體，並從這些檔案還原至資料庫叢集：

- 將 Percona XtraBackup 安裝在本機伺服器。
- 允許 Aurora MySQL 代替您存取 Amazon S3 儲存貯體。

安裝 Percona XtraBackup

Amazon Aurora 可以從使用 Percona XtraBackup 所建立的檔案來還原資料庫叢集。您可以從 Software Downloads - Percona (軟體下載 - Percona) 安裝 Percona XtraBackup。

若要遷移 MySQL 5.7，請使用 Percona XtraBackup 2.4。

若要遷移 MySQL 8.0，請使用 Percona XtraBackup 8.0。請確定 Percona XtraBackup 版本與來源資料庫的引擎版本相容。

所需的許可

若要將 MySQL 資料遷移至 Amazon Aurora MySQL 資料庫叢集，需要具備幾種許可：

- 請求 Aurora 從 Amazon S3 儲存貯體建立新叢集的使用者必須具有列出您 AWS 帳戶儲存貯體的許可。您可以使用 AWS Identity and Access Management (IAM) 政策授予使用者此許可。
- Aurora 需要許可來代表您存取 Amazon S3 儲存貯體，其中存放用來建立 Amazon Aurora MySQL 資料庫叢集的檔案。您需要使用 IAM 服務角色將必要許可授予 Aurora。

- 提出請求的使用者也必須具備許可才能列出 AWS 帳戶的 IAM 角色。
- 如果提出請求的使用者想要建立 IAM 服務角色，或要求 Aurora 建立 IAM 服務角色 (使用主控台)，則該使用者必須具備許可才能為您的 AWS 帳戶建立 IAM 角色。
- 如果您計劃在遷移過程中加密資料，請更新將執行遷移之使用者的 IAM 政策，以授予 RDS 對 AWS KMS keys 用於加密備份的 的存取權。如需說明，請參閱建立 IAM 政策以存取 AWS KMS資源。

例如，下列 IAM 政策將最低必要許可授予使用者，讓使用者可利用主控台來列出 IAM 角色、建立 IAM 角色、列出您帳戶的 Amazon S3 儲存貯體，以及列出 KMS 金鑰。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "s3:ListBucket",
                "kms:ListKeys"
            ],
            "Resource": "*"
        }
    ]
}
```

此外，若要讓使用者將 IAM 角色與 Amazon S3 儲存貯體相關聯，IAM 使用者必須具備該 IAM 角色的 `iam:PassRole` 許可。此許可允許管理員限制使用者可將哪些 IAM 角色與 Amazon S3 儲存貯體建立關聯。

例如，下列 IAM 政策可讓使用者將名為 `S3Access` 的角色與 Amazon S3 儲存貯體建立關聯。

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowS3AccessRole",
            "Effect":"Allow",
            "Action":"iam:PassRole",
            "Resource":"arn:aws:iam::123456789012:role/S3Access"
        }
    ]
}
```

如需 IAM 使用者許可的詳細資訊，請參閱 使用政策管理存取權。

建立 IAM 服務角色

您可以選擇建立新角色選項，讓 AWS 管理主控台 為您建立角色（本主題稍後顯示）。如果您選取此選項並指定新角色的名稱，Aurora 會建立必要的 IAM 服務角色，讓 Aurora 能夠存取您提供其名稱的 Amazon S3 儲存貯體。

或者，您也可以使用下列程序來手動建立角色。

為 Aurora 建立 IAM 角色以存取 Amazon S3

1. 完成「建立 IAM 政策來存取 Amazon S3 資源」中的步驟。
2. 完成「建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務」中的步驟。
3. 完成「將 IAM 角色與 Amazon Aurora MySQL 資料庫叢集建立關聯」中的步驟。

備份要還原為 Amazon Aurora MySQL 資料庫叢集的檔案

您可以使用 Percona XtraBackup 來建立 MySQL 資料庫檔案的完整備份，然後將備份檔案上傳至 Amazon S3 儲存貯體。或者，如果您已使用 Percona XtraBackup 來備份 MySQL 資料庫檔案，您可以將現有的完整和增量備份目錄及檔案上傳至 Amazon S3 儲存貯體。

主題

- 使用 Percona XtraBackup 來建立完整備份
- 透過 Percona XtraBackup 來使用增量備份
- 備份考量

使用 Percona XtraBackup 來建立完整備份

若要建立 MySQL 資料庫檔案的完整備份，以便能夠從 Amazon S3 還原來建立 Aurora MySQL 資料庫叢集，請使用 Percona XtraBackup 公用程式 (xtrabackup) 來備份資料庫。

例如，下列命令會建立 MySQL 資料庫的備份，並將檔案存放在 /on-premises/s3-restore/backup 資料夾中。

```
xtrabackup --backup --user=<myuser> --password=<password> --target-dir=</on-premises/s3-restore/backup>
```

如果要將備份壓縮成單一檔案 (需要時可以分割)，您可以使用 --stream 選項將備份儲存為下列其中一種格式：

- Gzip (.gz)
- tar (.tar)
- Percona xbstream (.xbstream)

下列命令建立 MySQL 資料庫的備份，並分割成多個 Gzip 檔案。

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \
   --target-dir=</on-premises/s3-restore/backup> | gzip - | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.tar.gz
```

下列命令建立 MySQL 資料庫的備份，並分割成多個 tar 檔案。

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \
   --target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.tar
```

下列命令建立 MySQL 資料庫的備份，並分割成多個 xbstream 檔案。

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=xbstream \
   --target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.xbstream
```

> **Note**
>
> 如果您看到下列錯誤，可能是因為在命令中混合檔案格式所致：

```
ERROR:/bin/tar: This does not look like a tar archive
```

使用 Percona XtraBackup 公用程式來備份 MySQL 資料庫之後，您可以將備份目錄和檔案複製到 Amazon S3 儲存貯體。

如需有關建立檔案並上傳至 Amazon S3 儲存貯體的資訊，請參閱 Amazon S3 入門指南中的 Amazon Simple Storage Service 入門。

透過 Percona XtraBackup 來使用增量備份

Amazon Aurora MySQL 支援使用 Percona XtraBackup 所建立的完整和增量備份。如果您已使用 Percona XtraBackup 來執行 MySQL 資料庫檔案的完整和增量備份，則不需要建立完整備份並將備份檔案上傳至 Amazon S3。反之，您可以將完整和增量備份的現有備份目錄及檔案複製到 Amazon S3 儲存貯體，以節省大量時間。如需詳細資訊，請參閱 Percona 網站上的 Create an incremental backup (建立增量備份)。

將現有的完整和增量備份檔案複製到 Amazon S3 儲存貯體時，您必須遞迴複製基本目錄的內容。這些內容包括完整備份，以及所有增量備份目錄和檔案。此副本必須保留 Amazon S3 儲存貯體中的目錄結構。Aurora 會逐一查看所有檔案和目錄。Aurora 使用包含在每個增量備份中的 `xtrabackup-checkpoints` 檔案，以識別基本目錄，以及依記錄序號 (LSN) 範圍來排序增量備份。

如需有關建立檔案並上傳至 Amazon S3 儲存貯體的資訊，請參閱 Amazon S3 入門指南中的 Amazon Simple Storage Service 入門。

備份考量

Aurora 不支援使用 Percona XtraBackup 所建立的局部備份。當您備份資料庫的來源檔案時，您無法使用 `--tables`、`--tables-exclude`、`--tables-file`、`--databases`、`--databases-exclude` 或 `--databases-file` 選項來建立局部備份。

如需使用 Percona XtraBackup 備份資料庫的詳細資訊，請參閱 Percona 網站上的 Percona XtraBackup - Documentation (Percona XtraBackup - 文件) 和 Work with binary logs (使用二進位日誌)。

Aurora 支援使用 Percona XtraBackup 所建立的增量備份。如需詳細資訊，請參閱 Percona 網站上的 Create an incremental backup (建立增量備份)。

Aurora 根據檔案名稱來取用備份檔案。務必根據檔案格式，以適當副檔案名稱來命名備份檔案 — 例如，使用 Percona xbstream 格式儲存的檔案應該採用 `.xbstream` 副檔案名稱。

Aurora 依字母順序和自然數順序來取用備份檔案。發出 `split` 命令時，一律使用 `xtrabackup` 選項，以確保依適當順序寫入和命名備份檔案。

Amazon S3 將上傳至 Amazon S3 儲存貯體的檔案大小限制為 5 TB。如果資料庫的備份資料超過 5 TB，請使用 `split` 命令將備份檔案分割成多個小於 5 TB 的檔案。

Aurora 將上傳至 Amazon S3 儲存貯體的來源檔案數量限制為 1 百萬個檔案。在某些情況下，資料庫的備份資料 (包括所有完整和增量備份) 可能包含大量檔案。在這些情況下，請使用 tarball (.tar.gz) 檔案將完整和增量備份檔案存放在 Amazon S3 儲存貯體中。

當您將檔案上傳至 Amazon S3 儲存貯體時，您可以使用伺服器端加密將資料加密。您之後就可以從這些加密的檔案還原 Amazon Aurora MySQL 資料庫叢集。Amazon Aurora MySQL 可以使用經過以下幾種伺服器端加密來還原有加密檔案的資料庫叢集：

- 伺服器端加密搭配 Amazon S3 管理的金鑰 (SSE-S3) – 以採用強式多重因素加密的唯一金鑰來加密每個物件。
- 伺服器端加密搭配 AWS KMS受管金鑰 (SSE-KMS) – 類似於 SSE-S3，但您可以選擇自行建立和管理加密金鑰，以及其他差異。

如需將檔案上傳至 Amazon S3 儲存貯體時使用伺服器端加密的相關資訊，請參閱《Amazon S3 開發人員指南》中的使用伺服器端加密保護資料。

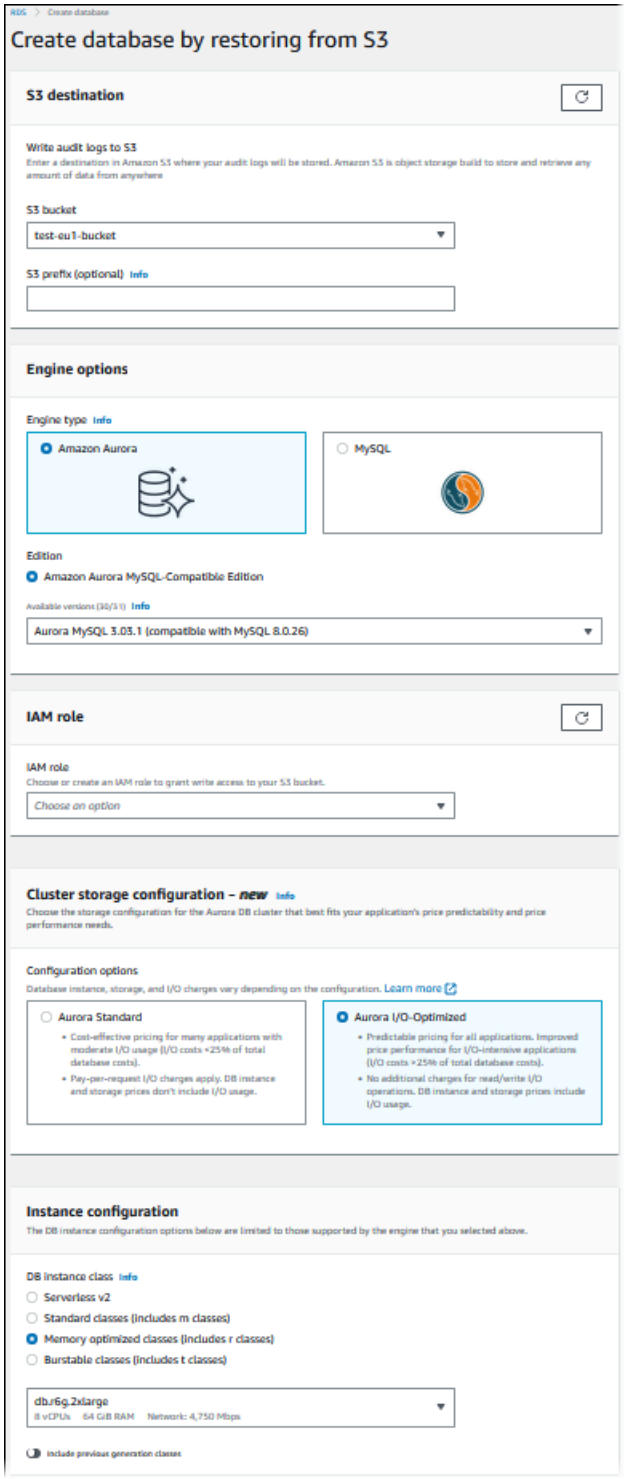
從 Amazon S3 儲存貯體還原 Amazon Aurora MySQL 資料庫叢集

您可以使用 Amazon RDS 主控台，從 Amazon S3 儲存貯體還原備份檔案，以建立新的 Amazon Aurora MySQL 資料庫叢集。

從 Amazon S3 儲存貯體上的檔案還原 Amazon Aurora MySQL 資料庫叢集

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在 Amazon RDS 主控台的右上角，選擇要建立資料庫叢集 AWS 的區域。選擇與包含資料庫備份的 Amazon S3 儲存貯體相同的 AWS 區域。
3. 在導覽窗格中，選擇 Databases (資料庫)，然後選擇 Restore from S3 (從 S3 還原)。
4. 選擇 Restore From S3 (從 S3 還原)。

   系統會顯示 Create database by restoring from S3 (從 S3 還原以建立資料庫) 頁面。



5. 在 S3 目的地 下：

   a. 選擇包含備份檔案的 S3 儲存貯體。

   b. (選用) 在 S3 folder path prefix (S3 資料夾路徑字首) 中，針對存放在 Amazon S3 儲存貯體中的檔案，輸入檔案路徑字首。

如果不指定字首，RDS 會使用 S3 儲存貯體之根資料夾中的所有檔案和資料夾來建立資料庫執行個體。如果指定字首，RDS 會使用 S3 儲存貯體中的檔案和資料夾來建立資料庫執行個體，而且檔案的路徑以指定的字首開頭。

例如，假設您將備份檔案儲存在 S3 中的一個名為 backups 的子資料夾，而且有多組備份檔案，各存放於自己的目錄中 (gzip_backup1、gzip_backup2 等等)。在此例子中，指定字首 backups/gzip_backup1，即可從 gzip_backup1 資料夾中的檔案還原。

6. 在 Engine options (引擎選項)：

   a. 針對 Engine type (引擎類型)，請選擇 Amazon Aurora。

   b. 針對 Version (版本)，請為已還原的資料庫執行個體選擇 Aurora MySQL 引擎版本。

7. 針對 IAM Role (IAM 角色)，您可以選擇現有的 IAM 角色。

8. (選用) 您也可以選擇 Create a new role (建立新角色)，讓系統為您建立新的 IAM 角色。若是如此：

   a. 請輸入 IAM role name (IAM 角色名稱)。

   b. 請選擇是否 Allow access to KMS key (允許存取 KMS 金鑰)：

      - 如果您未加密備份檔案，請選擇 No (否)。
      - 如果您將備份檔案上傳送至 Amazon S3 時以 AES-256 (SSE-S3) 加密，請選擇 No (否)。在此情況下，資料會自動解密。
      - 如果您在將備份檔案上傳至 Amazon S3 時，使用 AWS KMS (SSE-KMS) 伺服器端加密來加密備份檔案，請選擇是。接下來，為 AWS KMS key選擇正確的 KMS 金鑰。

        AWS 管理主控台 會建立 IAM 政策，讓 Aurora 能夠解密資料。

   如需詳細資訊，請參閱《Amazon S3 開發人員指南》中的使用伺服器端加密保護資料。

9. 選擇資料庫叢集的設定，例如資料庫叢集儲存組態、資料庫執行個體類別、資料庫叢集識別符，以及登入憑證。如需每項設定的相關資訊，請參閱 Aurora 資料庫叢集的設定。

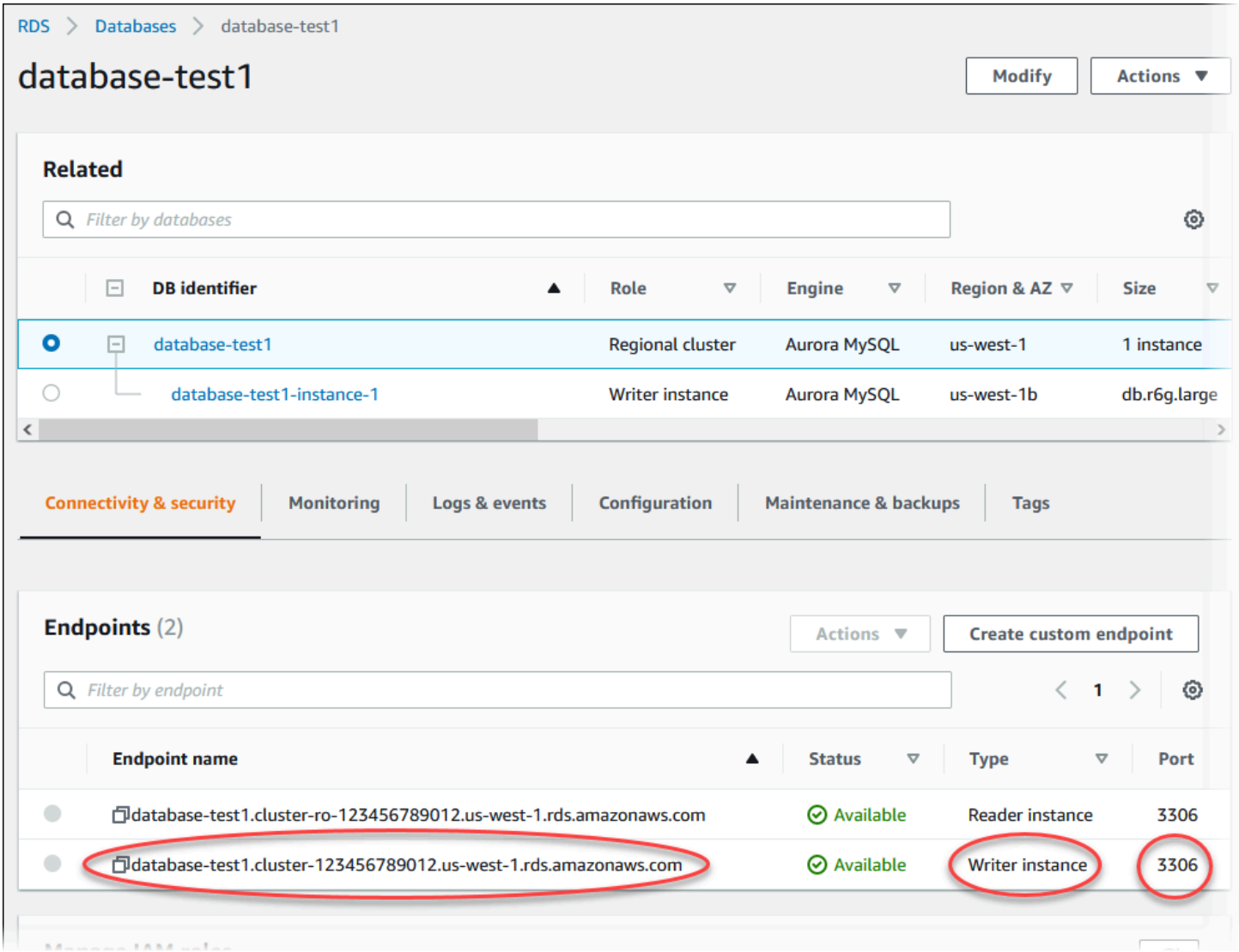
10. 視需為您的 Aurora MySQL 資料庫叢集自訂其他設定。

11. 選擇 Create database (建立資料庫)，以啟動 Aurora 資料庫執行個體。

在 Amazon RDS 主控台上，新的資料庫執行個體會顯示在資料庫執行個體清單中。在資料庫執行個體建立完成且可供使用之前，資料庫執行個體會處於 Creating (建立中) 狀態。狀態變更為 Available (可

用) 時，您便能連接到資料庫叢集的主執行個體。視資料庫執行個體類別和分配的存放區而定，新的執行個體可能需要幾分鐘才能使用。

若要檢視新建立的叢集，請在 Amazon RDS 主控台中選擇 Databases (資料庫) 檢視，然後選擇資料庫叢集。如需更多詳細資訊，請參閱 檢視 Amazon Aurora 資料庫叢集。



請記下資料庫叢集的連接埠和寫入器端點。您可以針對任何會執行寫入或讀取操作的應用程式，在 JDBC 和 ODBC 連線字串中使用資料庫叢集的寫入器端點和連接埠。

使用複寫來同步 Amazon Aurora MySQL 資料庫叢集與 MySQL 資料庫

若要在遷移過程中將停機情況降到最低或完全不停機，您可以將 MySQL 資料庫上已認可的交易複寫至 Aurora MySQL 資料庫叢集。複寫可讓資料庫叢集與遷移期間在 MySQL 資料庫上發生的交易達到同步。當資料庫叢集完全同步時，您就可以停止複寫，完成遷移至 Aurora MySQL。

主題

- 針對加密複寫來設定外部 MySQL 資料庫和 Aurora MySQL 資料庫叢集
- 同步 Amazon Aurora MySQL 資料庫叢集與外部 MySQL 資料庫

針對加密複寫來設定外部 MySQL 資料庫和 Aurora MySQL 資料庫叢集

若要安全地複寫資料，您可以使用加密複寫。

> **Note**
>
> 如果不需要使用加密複寫，您可以略過這些步驟，並前往同步 Amazon Aurora MySQL 資料庫叢集與外部 MySQL 資料庫中的指示。

下列是使用加密複寫的先決條件：

- 必須在外部 MySQL 主要資料庫上啟用 Secure Sockets Layer (SSL)。
- 必須為 Aurora MySQL 資料庫叢集準備用戶端金鑰和用戶端憑證。

在加密複寫期間，Aurora MySQL 資料庫叢集充當 MySQL 資料庫伺服器的用戶端。Aurora MySQL 用戶端的憑證和金鑰位於 .pem 格式的檔案中。

針對加密複寫來設定外部 MySQL 資料庫和 Aurora MySQL 資料庫叢集

1. 確保您已準備好進行加密複寫：

   - 如果您在外部 MySQL 主要資料庫上未啟用 SSL，也沒有準備用戶端金鑰和用戶端憑證，請在 MySQL 資料庫伺服器上啟用 SSL，並產生所需的用戶端金鑰和用戶端憑證。
   - 如果外部主要資料庫上已啟用 SSL，請為 Aurora MySQL 資料庫叢集提供用戶端金鑰和憑證。如果您沒有這些資料，請為 Aurora MySQL 資料庫叢集產生新的金鑰和憑證。若要簽署用戶端憑證，您必須有用於外部 MySQL 主要資料庫上設定 SSL 的憑證授權單位金鑰。

   如需詳細資訊，請參閱 MySQL 文件中的使用 openssl 建立 SSL 憑證和金鑰。

   您需要憑證授權單位憑證、用戶端金鑰和用戶端憑證。
2. 使用 SSL 以主要使用者的身分連接至 Aurora MySQL 資料庫叢集。

如需以 SSL 連接至 Aurora MySQL 資料庫叢集的相關資訊，請參閱Aurora MySQL 資料庫叢集的 TLS 連線。

3. 執行 mysql.rds_import_binlog_ssl_material 預存程序將 SSL 資訊匯入 Aurora MySQL 資料庫叢集。

   對於 `ssl_material_value` 參數，將 Aurora MySQL 資料庫叢集之 .pem 格式檔案中的資訊，插入正確的 JSON 承載中。

   下列範例將 SSL 資訊匯入 Aurora MySQL 資料庫叢集。在 .pem 格式檔案中，內文程式碼通常比範例所示的內文程式碼更長。

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

   如需詳細資訊，請參閱mysql.rds_import_binlog_ssl_material及Aurora MySQL 資料庫叢集的 TLS 連線。

   > **Note**
   >
   > 執行程序之後，密碼會儲存在檔案中。若稍後要清除這些檔案，您可以執行 mysql.rds_remove_binlog_ssl_material 預存程序。

同步 Amazon Aurora MySQL 資料庫叢集與外部 MySQL 資料庫

您可以使用複寫來同步 Amazon Aurora MySQL 資料庫叢集與 MySQL 資料庫。

使用複寫來同步 Aurora MySQL 資料庫叢集與 MySQL 資料庫

1. 請確定外部 MySQL 資料庫的 /etc/my.cnf 檔案有相關的項目。

   如果不需要加密複寫，則啟動外部 MySQL 資料庫時，務必啟用二進位日誌 (binlog) 並停用 SSL。下列是未加密資料 /etc/my.cnf 檔案中的相關項目。

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1
```

   如果需要加密複寫，則啟動外部 MySQL 資料庫時，務必啟用 SSL 和 binlog。/etc/my.cnf 檔案中的項目包括 MySQL 資料庫伺服器的 .pem 檔案位置。

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1

# Setup SSL.
ssl-ca=/home/sslcerts/ca.pem
ssl-cert=/home/sslcerts/server-cert.pem
ssl-key=/home/sslcerts/server-key.pem
```

   您可以使用下列命令確認 SSL 已啟用。

```
mysql> show variables like 'have_ssl';
```

   輸出類似如下。

```
+~-~-~-
```

```
+~-~-~-
```

使用者需要 REPLICATION CLIENT 和 REPLICATION SLAVE 權限。授予這些權限給該使用者。

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO
 '<user_name>'@'<domain_name>';
```

如果需要使用加密複寫，請對複寫使用者要求 SSL 連接。例如，您可以使用以下陳述式要求使用者帳戶 *<user_name>*.

```
GRANT USAGE ON *.* TO '<user_name>'@'<domain_name>' REQUIRE SSL;
```

> **Note**
>
> 如果未包含 REQUIRE SSL，則複寫連線可能會以無訊息方式回復為未加密的連線。

4. 在 Amazon RDS 主控台中，將託管外部 MySQL 資料庫之伺服器的 IP 地址，新增至 Aurora MySQL 資料庫叢集的 VPC 安全群組。如需有關修改 VPC 安全群組的詳細資訊，請參閱《Amazon Virtual Private Cloud 使用者指南》中的 VPC 安全群組。

   您可能還需要設定本機網路，以允許來自 Aurora MySQL 資料庫叢集之 IP 地址的連線，使其能與外部 MySQL 資料庫通訊。若要找出 Aurora MySQL 資料庫叢集的 IP 地址，請使用 host 命令。

   ```
   host <db_cluster_endpoint>
   ```

   主機名稱是來自 Aurora MySQL 資料庫叢集端點的 DNS 名稱。

5. 執行 mysql.rds_reset_external_master (Aurora MySQL 第 2 版) 或 mysql.rds_reset_external_source (Aurora MySQL 第 3 版) 預存程序以啟用二進位日誌複寫。此儲存的程序採用下列語法。

   ```
   CALL mysql.rds_set_external_master (
     host_name
     , host_port
     , replication_user_name
     , replication_user_password
     , mysql_binary_log_file_name
   ```

```
  , mysql_binary_log_file_location
  , ssl_encryption
);

CALL mysql.rds_set_external_source (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

如需有關參數的詳細資訊，請參閱 mysql.rds_reset_external_master (Aurora MySQL 第 2 版) 和 mysql.rds_reset_external_source (Aurora MySQL 第 3 版)。

在 `mysql_binary_log_file_name` 和 `mysql_binary_log_file_location` 中，使用您稍早在 Recovered from Binary log filename (已從二進位日誌檔案名稱還原) 事件中記下的位置。

如果 Aurora MySQL 資料庫叢集的資料未加密，則 `ssl_encryption` 參數必須設為 `0`。如果資料已加密，則 `ssl_encryption` 參數必須設為 `1`。

下列範例對具有加密資料的 Aurora MySQL 資料庫叢集執行此程序。

```
CALL mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'password',
  'mysql-bin.000010',
  120,
  1);

CALL mysql.rds_set_external_source(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'password',
  'mysql-bin.000010',
  120,
```

```
  1);
```

此儲存的程序會設定參數，供 Aurora MySQL 資料庫叢集用來連接至外部 MySQL 資料庫和讀取其二進位日誌。如果資料已加密，則它也會將 SSL 憑證授權單位憑證、用戶端憑證和用戶端金鑰下載至本機磁碟。

6. 執行 mysql.rds_start_replication 儲存的程序以啟動二進位日誌複寫。

```
CALL mysql.rds_start_replication;
```

7. 監控 Aurora MySQL 資料庫叢集落後於 MySQL 複寫主要資料庫的程度。若要這麼做，請連接至 Aurora MySQL 資料庫叢集並執行下列命令。

```
Aurora MySQL version 2:
SHOW SLAVE STATUS;

Aurora MySQL version 3:
SHOW REPLICA STATUS;
```

在命令輸出中，`Seconds Behind Master` 欄位顯示 Aurora MySQL 資料庫叢集落後於 MySQL 主要資料庫的程度。當此值為 `0` (零) 時，表示 Aurora MySQL 資料庫叢集與主要資料庫已同步，而您可以繼續下一步來停止複寫。

8. 連接至 MySQL 複寫主要資料庫並停止複寫。若要這樣做，請執行 mysql.rds_stop_replication 預存程序。

```
CALL mysql.rds_stop_replication;
```

縮短實體遷移到 Amazon Aurora MySQL 的時間

您可以進行以下資料庫修改來加快將資料庫遷移至 Amazon Aurora MySQL 的程序。

> **Important**
>
> 請務必對生產資料庫的副本進行這些更新，而不是直接更新生產資料庫。接著您可以備份副本並將它還原至 Aurora MySQL 資料庫叢集，以避免生產資料庫發生任何服務中斷的情況。

不支援的資料表類型

Aurora MySQL 針對資料庫資料表僅支援 InnoDB 引擎。如果您的資料庫中有 MyISAM 資料表，這些資料表必須先轉換，再遷移至 Aurora MySQL。在遷移過程中，轉換程序需要額外的空間將 MyISAM 轉換為 InnoDB。

若要盡可能避免空間不足，或想要加速遷移程序，請先將所有 MyISAM 資料表轉換成 InnoDB 資料表再遷移。產生的 InnoDB 資料表大小相當於 Aurora MySQL 針對該資料表所需的大小。若要將 MyISAM 資料表轉換成 InnoDB，請執行下列命令：

```
ALTER TABLE schema.table_name engine=innodb, algorithm=copy;
```

Aurora MySQL 不支援壓縮的資料表或頁面 (也就是以 ROW_FORMAT=COMPRESSED 或 COMPRESSION = {"zlib"|"lz4"} 建立的資料表)。

若要盡可能避免空間不足，或想要加速遷移程序，請將 ROW_FORMAT 設為 DEFAULT、COMPACT、DYNAMIC 或 REDUNDANT，以展開壓縮資料表。對於壓縮的頁面，請設定 COMPRESSION="none"。

如需詳細資訊，請參閱 MySQL 文件中的 InnoDB 資料列格式和 InnoDB 資料表與頁面壓縮。

您可以在現有的 MySQL 資料庫執行個體上使用下列 SQL 指令碼，以列出資料庫中的 MyISAM 資料表或壓縮資料表。

```
-- This script examines a MySQL database for conditions that block
-- migrating the database into Aurora MySQL.
-- It must be run from an account that has read permission for the
-- INFORMATION_SCHEMA database.

-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
  (
  select
    'This script should be run on MySQL version 5.6 or higher. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
      cast(substring_index(substring_index(version(), '.', 2), '.', -1)
      as unsigned)
    as major_minor
  ) as T
```

```
where major_minor <> 506;


-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
  ENGINE <> 'InnoDB'
  and
  (
    -- User tables
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                         'information_schema')
    or
    -- Non-standard system tables
    (
      TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
        (
          'columns_priv', 'db', 'event', 'func', 'general_log',
          'help_category', 'help_keyword', 'help_relation',
          'help_topic', 'host', 'ndb_binlog_index', 'plugin',
          'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
          'tables_priv', 'time_zone', 'time_zone_leap_second',
          'time_zone_name', 'time_zone_transition',
          'time_zone_transition_type', 'user'
        )
    )
  )
  or
  (
    -- Compressed tables
       ROW_FORMAT = 'Compressed'
  );
```

具有不支援權限的使用者帳戶

具有 Aurora MySQL 不支援之權限的使用者帳戶在匯出時，不支援的權限不會一起匯出。如需支援的權限清單，請參閱 角色型權限模型。

您可以在來源資料庫上執行下列 SQL 查詢，以列出具有不支援權限的使用者帳戶。

```
SELECT
```

```
    user,
    host
FROM
    mysql.user
WHERE
    Shutdown_priv = 'y'
    OR File_priv = 'y'
    OR Super_priv = 'y'
    OR Create_tablespace_priv = 'y';
```

Aurora MySQL 第 3 版中的動態權限

不會匯入動態權限。Aurora MySQL 第 3 版支援以下動態權限。

```
'APPLICATION_PASSWORD_ADMIN',
'CONNECTION_ADMIN',
'REPLICATION_APPLIER',
'ROLE_ADMIN',
'SESSION_VARIABLES_ADMIN',
'SET_USER_ID',
'XA_RECOVER_ADMIN'
```

以下範例指令碼會將支援的動態權限授予 Aurora MySQL 資料庫叢集中的使用者帳戶。

```
-- This script finds the user accounts that have Aurora MySQL supported dynamic
 privileges
-- and grants them to corresponding user accounts in the Aurora MySQL DB cluster.

/home/ec2-user/opt/mysql/8.0.26/bin/mysql -uusername -pxxxxx -P8026 -h127.0.0.1 -BNe
 "SELECT
  CONCAT('GRANT ', GRANTS, ' ON *.* TO ', GRANTEE ,';') AS grant_statement
  FROM (select GRANTEE, group_concat(privilege_type) AS GRANTS FROM
 information_schema.user_privileges
      WHERE privilege_type IN (
        'APPLICATION_PASSWORD_ADMIN',
        'CONNECTION_ADMIN',
        'REPLICATION_APPLIER',
        'ROLE_ADMIN',
        'SESSION_VARIABLES_ADMIN',
        'SET_USER_ID',
        'XA_RECOVER_ADMIN')
      AND GRANTEE NOT IN (\"'mysql.session'@'localhost'\",
\"'mysql.infoschema'@'localhost'\",\"'mysql.sys'@'localhost'\") GROUP BY GRANTEE)
```

```
      AS PRIVGRANTS; " | /home/ec2-user/opt/mysql/8.0.26/bin/mysql -u master_username -
p master_password -h DB_cluster_endpoint
```

使用「rdsadmin'@'localhost」作為 DEFINER 的預存物件

不會匯入使用 'rdsadmin'@'localhost' 作為 DEFINER 的函數、程序、檢視、事件和觸發。

您可以在來源 MySQL 資料庫上使用以下 SQL 指令碼列出具有不支援 DEFINER 的預存物件。

```
-- This SQL query lists routines with `rdsadmin`@`localhost` as the definer.

SELECT
    ROUTINE_SCHEMA,
    ROUTINE_NAME
FROM
    information_schema.routines
WHERE
    definer = 'rdsadmin@localhost';

-- This SQL query lists triggers with `rdsadmin`@`localhost` as the definer.

SELECT
    TRIGGER_SCHEMA,
    TRIGGER_NAME,
    DEFINER
FROM
    information_schema.triggers
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists events with `rdsadmin`@`localhost` as the definer.

SELECT
    EVENT_SCHEMA,
    EVENT_NAME
FROM
    information_schema.events
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists views with `rdsadmin`@`localhost` as the definer.
SELECT
    TABLE_SCHEMA,
    TABLE_NAME
```

```
FROM
    information_schema.views
WHERE
    DEFINER = 'rdsadmin@localhost';
```

使用 mysqldump 從 MySQL 到 Amazon Aurora 的邏輯遷移

因為 Amazon Aurora MySQL 是 MySQL 相容資料庫，您可以使用 `mysqldump` 公用程式，從 MySQL 資料庫或 `mariadb-dump` 公用程式複製資料，將 MariaDB 資料庫的資料複製到現有的 Aurora MySQL 資料庫叢集。

如需如何使用非常大的 MySQL 或 MariaDB 資料庫進行討論，請參閱《Amazon Relational Database Service 使用者指南》中的下列主題：

- MySQL – 將資料匯入至 Amazon RDS for MySQL 資料庫，並減少停機時間
- MariaDB – 將資料匯入至 Amazon RDS for MariaDB 資料庫，並減少停機時間

對於資料量較少的 MySQL 或 MariaDB 資料庫，請參閱《Amazon Relational Database Service 使用者指南》中的下列主題：

- MySQL – 從外部 MySQL 資料庫將資料匯入 Amazon RDS for MySQL 資料庫執行個體
- MariaDB – 從外部 MariaDB 資料庫將資料匯入 Amazon RDS for MariaDB 資料庫執行個體

將資料從 RDS for MySQL 資料庫執行個體遷移到 Amazon Aurora MySQL 資料庫叢集

您可以將資料從 RDS for MySQL 資料庫執行個體遷移 (複製) 至 Amazon Aurora MySQL 資料庫叢集。

主題

- 將 RDS for MySQL 快照遷移至 Aurora
- 使用 Aurora 讀取複本，從 RDS for MySQL 資料庫執行個體將資料遷移至 Amazon Aurora MySQL 資料庫叢集

> **Note**
>
> 因為 Amazon Aurora MySQL 與 MySQL 相容，您可以在 MySQL 資料庫和 Amazon Aurora MySQL 資料庫叢集之間設定複寫，以便從 MySQL 資料庫遷移資料。如需詳細資訊，請參閱以 Amazon Aurora 進行複寫。

將 RDS for MySQL 快照遷移至 Aurora

您可以遷移 RDS for MySQL 資料庫執行個體的資料庫快照，以建立 Aurora MySQL 資料庫叢集。新的 Aurora MySQL 資料庫叢集會使用來自原始 RDS for MySQL 資料庫執行個體的資料填入。資料庫快照必須是從執行 MySQL 版本 (與 Aurora MySQL 相容) 的 Amazon RDS 資料庫執行個體建立。

您可以遷移手動或自動資料庫快照。建立資料庫叢集之後，您就可以建立選用的 Aurora 複本。

> **Note**
>
> 您也可以建立來源 RDS for MySQL 資料庫執行個體的 Aurora 僅供讀取複本，以便將 RDS for MySQL 資料庫執行個體遷移至 Aurora MySQL 資料庫叢集。如需詳細資訊，請參閱使用 Aurora 讀取複本，從 RDS for MySQL 資料庫執行個體將資料遷移至 Amazon Aurora MySQL 資料庫叢集。
> 您無法從某些舊版 MySQL 8.0 版本 (包括 8.0.11、8.0.13 和 8.0.15) 遷移到 Aurora MySQL 版本 3.05 及更高版本。建議您在遷移之前，先升級至 MySQL 8.0.28 版。

您必須採取的一般步驟如下：

1. 決定要佈建給 Aurora MySQL 資料庫叢集的空間數量。如需詳細資訊，請參閱我需要多少空間？
2. 使用 主控台在 Amazon RDS MySQL 執行個體所在的 AWS 區域中建立快照。如需建立資料庫快照的相關資訊，請參閱建立資料庫快照。
3. 如果資料庫快照與您的資料庫叢集不在同一個 AWS 區域中，請使用 Amazon RDS 主控台將資料庫快照複製到該 AWS 區域。如需複製資料庫快照的相關資訊，請參閱複製資料庫快照。
4. 使用主控台來遷移資料庫快照，並使用與原始 MySQL 資料庫執行個體相同的資料庫建立 Aurora MySQL 資料庫叢集。

> **Warning**
>
> Amazon RDS 會限制每個 AWS 帳戶一次一個快照複製到每個 AWS 區域。

我需要多少空間？

當您將 MySQL 資料庫執行個體的快照遷移至 Aurora MySQL 資料庫叢集時，Aurora 在遷移快照之前，將會先使用 Amazon Elastic Block Store (Amazon EBS) 磁碟區來格式化快照中的資料。在某些情況下，需要額外的空間以格式化準備遷移的資料。

如果資料表不是 MyISAM 資料表且未壓縮，則最大為 16 TB。如果您有 MyISAM 資料表，則 Aurora 必須使用磁碟區中額外的空間，將資料表轉換成與 Aurora MySQL 相容。如果您有壓縮的資料表，則 Aurora 必須使用磁碟區中額外的空間將這些資料表解壓縮，再存放於 Aurora 叢集磁碟區。由於需要此額外空間，請確定要從 MySQL 資料庫執行個體遷移的 MyISAM 和壓縮資料表的大小皆未超過 8 TB。

減少將資料遷移至 Amazon Aurora MySQL 所需的空間量

您可能想要先修改資料庫結構描述再遷移至 Amazon Aurora。在下列情況中，如此修改可能相當實用：

- 您想要加快遷移程序的速度。
- 您不確定必須佈建多少空間。
- 您已嘗試遷移資料，但因為佈建的空間不足導致遷移失敗。

您可以進行下列變更，以改善將資料庫移遷移至 Amazon Aurora 的程序。

> **Important**
>
> 務必在從生產資料庫的快照所還原的新資料庫執行個體上執行這些更新，而非在生產執行個體上。然後，您可以從新資料庫執行個體的快照將資料遷移至 Aurora 資料庫叢集，以避免生產資料庫上的任何服務中斷。

資料表類型	限制或指導方針
MyISAM 資料表	Aurora MySQL 僅支援 InnoDB 資料表。如果您的資料庫中有 MyISAM 資料表，這些資料表必須先轉換再遷移至 Aurora MySQL。在遷移過程中，轉換程序需要額外的空間將 MyISAM 轉換為 InnoDB。 若要盡可能避免空間不足，或想要加速遷移程序，請先將所有 MyISAM 資料表轉換成 InnoDB 資料表再遷移。產生的 InnoDB 資料表大小相當於 Aurora MySQL 針對該資料表所需的大小。若要將 MyISAM 資料表轉換成 InnoDB，請執行下列命令： `alter table <schema>.<table_name> engine=innodb, algorithm=copy;`
壓縮資料表	Aurora MySQL 不支援壓縮資料表 (即以 `ROW_FORMAT=COMPRESSED` 建立的資料表)。 若要盡可能避免空間不足，或想要加速遷移程序，請將 `ROW_FORMAT` 設為 `DEFAULT`、`COMPACT`、`DYNAMIC` 或 `REDUNDANT`，以展開壓縮資料表。如需詳細資訊，請參閱 MySQL 文件中的 InnoDB 資料列格式。

您可以在現有的 MySQL 資料庫執行個體上使用下列 SQL 指令碼，以列出資料庫中的 MyISAM 資料表或壓縮資料表。

```
-- This script examines a MySQL database for conditions that block
-- migrating the database into Amazon Aurora.
-- It needs to be run from an account that has read permission for the
-- INFORMATION_SCHEMA database.
```

```
-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
  (
  select
    'This script should be run on MySQL version 5.6 or higher. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
      cast(substring_index(substring_index(version(), '.', 2), '.', -1)
      as unsigned)
    as major_minor
  ) as T
where major_minor <> 506;


-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
  ENGINE <> 'InnoDB'
  and
  (
    -- User tables
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                         'information_schema')
    or
    -- Non-standard system tables
    (
      TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
        (
          'columns_priv', 'db', 'event', 'func', 'general_log',
          'help_category', 'help_keyword', 'help_relation',
          'help_topic', 'host', 'ndb_binlog_index', 'plugin',
          'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
          'tables_priv', 'time_zone', 'time_zone_leap_second',
          'time_zone_name', 'time_zone_transition',
          'time_zone_transition_type', 'user'
        )
    )
  )
  or
```

```
  (
    -- Compressed tables
       ROW_FORMAT = 'Compressed'
  );
```

指令碼產生的輸出類似於下列範例中的輸出。範例顯示兩個必須從 MyISAM 轉換成 InnoDB 的資料表。輸出也包含每個資料表的大約大小 (以 MB 為單位)。

```
+---------------------------------+------------------+
| ==> MyISAM or Compressed Tables | Approx size (MB) |
+---------------------------------+------------------+
| test.name_table                 |          2102.25 |
| test.my_table                   |            65.25 |
+---------------------------------+------------------+
2 rows in set (0.01 sec)
```

將 RDS for MySQL 資料庫快照遷移至 Aurora MySQL 資料庫叢集

您可以使用 AWS 管理主控台 或 遷移 RDS for MySQL 資料庫執行個體的資料庫快照，以建立 Aurora MySQL 資料庫叢集 AWS CLI。新的 Aurora MySQL 資料庫叢集會使用來自原始 RDS for MySQL 資料庫執行個體的資料填入。如需建立資料庫快照的相關資訊，請參閱建立資料庫快照。

如果資料庫快照不在您要尋找資料的 AWS 區域中，請將資料庫快照複製到該 AWS 區域。如需複製資料庫快照的相關資訊，請參閱複製資料庫快照。

主控台

當您使用 遷移資料庫快照時 AWS 管理主控台，主控台會採取僅建立資料庫叢集所需的動作。

您也可以選擇使用 AWS KMS key，為新的 Aurora MySQL 資料庫叢集進行靜態加密。

使用 遷移 MySQL 資料庫快照 AWS 管理主控台

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 從 MySQL 資料庫執行個體或從快照來開始遷移：

   從資料庫執行個體開始遷移：

   1. 在導覽窗格中，選擇 Databases (資料庫)，然後選取 MySQL 資料庫執行個體。
   2. 在 Actions (動作) 中，選擇 Migrate latest snapshot (遷移最新的快照)。

若要從快照開始遷移，請執行以下操作：

1. 選擇 Snapshots (快照)。
2. 在 Snapshots (快照) 頁面上，選擇您要遷移至 Aurora MySQL 資料庫叢集的快照。
3. 選擇 Snapshot Actions (快照動作)，然後選擇 Migrate Snapshot (遷移快照)。

Migrate Database (遷移資料庫) 頁面隨即出現。

3. 在 Migrate Database (遷移資料庫) 頁面上設定下列值：

- Migrate to DB Engine (遷移至資料庫引擎)：選取 `aurora`。
- DB Engine Version (資料庫引擎版本)：選取 Aurora MySQL 資料庫叢集的資料庫引擎版本。
- DB Instance Class (資料庫執行個體類別)：為您的資料庫選取具有所需儲存體和容量的資料庫執行個體類別，例如 `db.r3.large`。Aurora 叢集磁碟區會隨著您的資料庫中的資料數量增加自動成長。Aurora 叢集磁碟區的大小最多可增長至 128 tebibytes (TiB)。因此，您只需選擇滿足目前儲存體需求的資料庫執行個體。如需詳細資訊，請參閱Amazon Aurora 儲存體的概觀。
- 資料庫執行個體識別符：輸入您所選 AWS 區域中帳戶唯一的資料庫叢集名稱。此識別符用於資料庫叢集內執行個體的端點位址。您可以選擇將一些智慧新增至名稱，例如包含您選取的 AWS 區域和資料庫引擎，例如 **`aurora-cluster1`**。

  該資料庫執行個體識別符有下列限制：

  - 必須包含 1 到 63 個英數字元或連字號。
  - 第一個字元必須是字母。
  - 不能以一個連字號結尾或是連續包含兩個連字號。
  - 每個 AWS 區域每個 AWS 帳戶的所有資料庫執行個體都必須是唯一的。
- Virtual Private Cloud (VPC)：如果具有現有的 VPC，您可以選取 VPC 識別符，例如 `vpc-a464d1c1`，將該 VPC 用於 Aurora MySQL 資料庫叢集。如需建立 VPC 的詳細資訊，請參閱教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4)。

  否則，您可以選擇由 Aurora 為您建立 VPC，方法為選取 Create a new VPC (建立新的 VPC)。
- DB Subnet group (資料庫子網路群組)：如果具有現有的子網路群組，您可以選取子網路群組識別符，例如 `gs-subnet-group1`，將該子網路群組用於 Aurora MySQL 資料庫叢集。

  否則，您可以選擇由 Aurora 為您建立子網路群組，方法為選取 Create a new subnet group (建立新的子網路群組)。

- Public a ccessibility (公開存取性)：選擇 No (否)，以指定資料庫叢集中的執行個體只能由 VPC 內的資源存取。選取 Yes (是)，以指定公有網路上的資源可以存取資料庫叢集內的執行個體。預設值為 Yes (是)。

  > **Note**
  >
  > 生產資料庫叢集可能不必位於公有子網路中，因為只有應用程式伺服器才需要存取資料庫叢集。如果您的資料庫叢集不必位於公有子網路中，將 Publicly Accessible (公開存取性) 設為 No (否)。

- Availability zone (可用區域)：選取可用區域以託管 Aurora MySQL 資料庫叢集的主要執行個體。若要讓 Aurora 為您選擇可用區域，請選取 No Preference (無偏好設定)。
- Database Port (資料庫連接埠)：輸入要在連接至 Aurora MySQL 資料庫叢集中的執行個體時使用的預設連接埠。預設值為 3306。

  > **Note**
  >
  > 您可能在公司防火牆的後方，而此防火牆不允許存取預設連接埠，例如 MySQL 預設連接埠 3306。在此情況下，提供公司防火牆允許的連接埠值。稍後連線至 Aurora MySQL 資料庫叢集時，請記住該連接埠值。

- Encryption (加密)：選擇 Enable Encryption (啟用加密)，對新的 Aurora MySQL 資料庫叢集進行靜態加密。如果您選擇 Enable Encryption (啟用加密)，則必須選擇 KMS 金鑰作為 AWS KMS key值。

  如果資料庫快照未加密，請指定加密金鑰，以便對資料庫叢集進行靜態加密。

  如果資料庫快照已加密，請指定加密金鑰，以使用指定的加密金鑰對資料庫叢集進行靜態加密。您可以指定資料庫快照所使用的加密金鑰，或不同的金鑰。您無法從已加密的資料庫快照來建立未加密的資料庫叢集。

- Auto Minor Version Upgrade (自動次要版本升級)：此設定不適用於 Aurora MySQL 資料庫叢集。

  如需 Aurora MySQL 引擎更新的詳細資訊，請參閱 Amazon Aurora MySQL 的資料庫引擎更新。

4. 選擇 Migrate (遷移) 以遷移您的資料庫快照。

5. 選擇 Instances (執行個體)，然後選擇箭頭圖示以顯示資料庫叢集詳細資訊並監控遷移進度。在詳細資訊頁面上，您可以找到用於連接至資料庫叢集之主要執行個體的叢集端點。如需連接至 Aurora MySQL 資料庫叢集的詳細資訊，請參閱連接至 Amazon Aurora 資料庫叢集。

AWS CLI

您可以使用 `restore-db-cluster-from-snapshot` 命令和下列參數，從 RDS for MySQL 資料庫執行個體的資料庫快照來建立 Aurora 資料庫叢集：

- `--db-cluster-identifier` – 要建立的資料庫叢集名稱。
- `--engine aurora-mysql` – 適用於 MySQL 5.7 相容或 8.0 相容的資料庫叢集。
- `--kms-key-id` – AWS KMS key 選擇性地使用 加密資料庫叢集，取決於您的資料庫快照是否已加密。
  - 如果資料庫快照未加密，請指定加密金鑰，以便對資料庫叢集進行靜態加密。否則，資料庫叢集不會加密。
  - 如果資料庫快照已加密，請指定加密金鑰，以使用指定的加密金鑰對資料庫叢集進行靜態加密。否則會使用資料庫快照的加密金鑰，對資料庫叢集進行靜態加密。

  > **Note**
  >
  > 您無法從已加密的資料庫快照來建立未加密的資料庫叢集。

- `--snapshot-identifier` – 要遷移之資料庫快照的 Amazon 資源名稱 (ARN)。如需 Amazon RDS ARN 的詳細資訊，請參閱 Amazon Relational Database Service (Amazon RDS)。

當您使用 `RestoreDBClusterFromSnapshot` 命令來遷移資料庫快照時，此命令會建立資料庫叢集和主要執行個體。

在此範例中，您從 ARN 設為 *`mydbsnapshotARN`* 的資料庫快照，建立 MySQL 5.7 相容的資料庫叢集，名為 *`mydbcluster`*。

對於 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mydbcluster \
    --snapshot-identifier mydbsnapshotARN \
    --engine aurora-mysql
```

在 Windows 中：

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --snapshot-identifier mydbsnapshotARN ^
    --engine aurora-mysql
```

在此範例中，您從 ARN 設為 *mydbsnapshotARN* 的資料庫快照，建立 MySQL 5.7 相容的資料庫叢集，名為 *mydbcluster*。

對於 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mydbcluster \
    --snapshot-identifier mydbsnapshotARN \
    --engine aurora-mysql
```

在 Windows 中：

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --snapshot-identifier mydbsnapshotARN ^
    --engine aurora-mysql
```

使用 Aurora 讀取複本，從 RDS for MySQL 資料庫執行個體將資料遷移至 Amazon Aurora MySQL 資料庫叢集

Aurora 使用 MySQL 資料庫引擎的二進位日誌複寫功能，為來源 RDS for MySQL 資料庫執行個體建立一種特殊類型的資料庫叢集，稱為 Aurora 讀取複本。對來源 RDS for MySQL 資料庫執行個體進行的更新會以非同步方式複寫至 Aurora 讀取複本。

我們建議使用此功能建立來源 RDS for MySQL 資料庫執行個體的 Aurora 讀取複本，以便從 RDS for MySQL 資料庫執行個體遷移至 Aurora MySQL 資料庫叢集。當 RDS for MySQL 資料庫執行個體和 Aurora 讀取複本之間的複本延遲為 0 時，您可以將用戶端應用程式引導至 Aurora 讀取複本，然後停止複寫，使 Aurora 讀取複本變成獨立的 Aurora MySQL 資料庫叢集。遷移需要一段時間，每個一兆位元組 (TiB) 資料大約需要幾個小時，請做好準備。

如需可使用 Aurora 的區域清單，請參閱 AWS 一般參考 中的 Amazon Aurora。

當您建立 RDS for MySQL 資料庫執行個體的 Aurora 讀取複本時，Amazon RDS 會建立來源 RDS for MySQL 資料庫執行個體的資料庫快照 (為 Amazon RDS 私有，不會產生費用)。然後 Amazon RDS 會將資料從資料庫快照遷移至 Aurora 僅供讀取複本。當資料庫快照的資料遷移至新的 Aurora MySQL 資料庫叢集之後，Amazon RDS 會開始在 RDS for MySQL 資料庫執行個體和 Aurora MySQL 資料庫叢集之間複寫。如果 RDS for MySQL 資料庫執行個體中有資料表使用 InnoDB 以外的儲存引擎，或使用壓縮資料列格式，則在建立 Aurora 讀取複本之前，您可以更改這些資料表來使用 InnoDB 儲存引擎和動態資料列格式，使建立 Aurora 讀取複本的程序加快。如需將 MySQL 資料庫快照複製到 Aurora MySQL 資料庫叢集之程序的詳細資訊，請參閱將資料從 RDS for MySQL 資料庫執行個體遷移到 Amazon Aurora MySQL 資料庫叢集。

一個 RDS for MySQL 資料庫執行個體只能有一個 Aurora 讀取複本。

> **Note**
>
> 由於 Aurora MySQL 與 RDS for MySQL 資料庫執行個體 (複寫主節點) 的 MySQL 資料庫引擎版本之間的功能差異，複寫可能會出現問題。如果您遇到錯誤，您可以在 Amazon RDS 社群論壇或聯絡 找到說明 AWS 支援。
>
> 如果 RDS for MySQL 資料庫執行個體已是跨區域讀取複本的來源，您就無法建立 Aurora 讀取複本。
>
> 您無法從某些較舊的 RDS for MySQL 8.0 版本 (包括 8.0.11、8.0.13 和 8.0.15) 遷移到 Aurora MySQL 版本 3.05 及更高版本。建議您在遷移之前，先升級至 RDS for MySQL 8.0.28 版。

如需 MySQL 僅供讀取複本的詳細資訊，請參閱使用 MariaDB、MySQL 及 PostgreSQL DB 執行個體的僅供讀取複本。

建立 Aurora 僅供讀取複本

您可以使用 主控台 AWS CLI、 或 RDS API，為 RDS for MySQL 資料庫執行個體建立 Aurora 僅供讀取複本。

主控台

從來源 RDS for MySQL 資料庫執行個體建立 Aurora 讀取複本

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇要作為 Aurora 僅供讀取複本來源的 MySQL 資料庫執行個體。
4. 在 Actions (動作) 中選擇 Create Aurora read replica (建立僅供讀取複本)。
5. 選擇要用於 Aurora 僅供讀取複本的資料庫叢集規格，如下表所述。

選項	描述
DB instance class (資料庫執行個體類別)	選擇資料庫執行個體類別，為資料庫叢集內的主要執行個體定義處理和記憶體要求。如需資料庫執行個體類別選項的詳細資訊，請參閱Amazon Aurora 資料庫執行個體類別。
Multi-AZ deployment (異地同步備份部署)	選擇 Create Replica in Different Zone (在不同區域建立複本)，在目標 AWS 區域的另一個可用區域中建立新資料庫叢集的待命複本，以獲得容錯移轉支援。如需多個可用區域的詳細資訊，請參閱區域和可用區域。
DB instance identifier (資料庫執行個體識別符)：	鍵入 Aurora 僅供讀取複本資料庫叢集內主要執行個體的名稱。此識別符用於新資料庫叢集內主要執行個體的端點位址。 該資料庫執行個體識別符有下列限制： • 必須包含 1 到 63 個英數字元或連字號。 • 第一個字元必須是字母。

選項	描述
	• 不能以一個連字號結尾或是連續包含兩個連字號。 • 對於每個 AWS 區域，每個帳戶的所有資料庫執行個體都必須是唯一的 AWS 。 因為 Aurora 僅供讀取複本資料庫叢集是以來源資料庫執行個體的快照所建立，Aurora 僅供讀取複本的主要使用者名稱與主要密碼會和來源資料庫執行個體的主要使用者名稱與主要密碼相同。
Virtual Private Cloud (VPC)	選取 VPC 來託管資料庫叢集。選取 Create a new VPC (建立新的 VPC)，讓 Aurora 為您建立 VPC。如需詳細資訊，請參閱資料庫叢集先決條件。
DB subnet group (資料庫子網路群組)	選取要用於資料庫叢集的資料庫子網路群組。選取 Create a new DB subnet group (建立新的資料庫子網路群組)，讓 Aurora 為您建立資料庫子網路群組。如需更多詳細資訊，請參閱 資料庫叢集先決條件。
Public accessibility (公開存取性)	選取 Yes 以給與資料庫叢集一個公有 IP 地址；否則，請選取 No。資料庫叢集內的執行個體可以混合公有和私有資料庫執行個體。如需隱藏執行個體而不提供公開存取的詳細資訊，請參閱在 VPC 中的網際網路中隱藏資料庫叢集。
Availability zone (可用區域)	決定您是否要指定特定的可用區域。如需可用區域的詳細資訊，請參閱 區域和可用區域。
VPC security group (firewall) (VPC 安全群組 (防火牆))	選取 Create a new VPC security group (建立新的 VPC 安全群組)，讓 Aurora 為您建立 VPC 安全群組。選取 Select existing VPC security groups (選取現有的 VPC 安全群組)，以指定一個或多個 VPC 安全群組來保護資料庫叢集的網路存取。如需更多詳細資訊，請參閱 資料庫叢集先決條件。

選項	描述
Database port (資料庫連接埠)	指定應用程式和公用程式將用於存取資料庫的連接埠。Aurora MySQL 資料庫叢集預設為 MySQL 預設連接埠 3306。某些公司的防火牆會封鎖與此預設連接埠的連線。如果您的公司防火牆會封鎖預設連接埠，請為新的資料庫叢集選擇另一個連接埠。
DB parameter group (資料庫參數群組)	選取 Aurora MySQL 資料庫叢集的資料庫參數群組。Aurora 有一個預設資料庫參數群組供您使用，您也可以建立自己的資料庫參數群組。如需資料庫參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。
DB cluster parameter group (資料庫叢集參數群組)	選取 Aurora MySQL 資料庫叢集的資料庫叢集參數群組。Aurora 有一個預設資料庫叢集參數群組供您使用，您也可以建立自己的資料庫叢集參數群組。如需資料庫叢集參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。
加密	如果您不要將新的 Aurora 資料庫叢集加密，請選擇 Disable encryption (停用加密)。選擇 Enable encryption (啟用加密)，以便對新的 Aurora 資料庫叢集進行靜態加密。如果您選擇 Enable Encryption (啟用加密)，則必須選擇 KMS 金鑰作為 AWS KMS key值。 如果 MySQL 資料庫執行個體未加密，請指定加密金鑰，以便對資料庫叢集進行靜態加密。 如果 MySQL 資料庫執行個體已加密，請指定加密金鑰，以使用指定的加密金鑰對資料庫叢集進行靜態加密。您可以指定 MySQL 資料庫執行個體所使用的加密金鑰，或不同的金鑰。您無法從已加密的 MySQL 資料庫執行個體來建立未加密的資料庫叢集。

選項	描述
優先順序	選擇資料庫執行個體的容錯移轉優先順序。如果您未選取值，則預設值為 tier-1 (第一層)。此優先順序決定從主要執行個體失敗中復原時提升 Aurora 複本的順序。如需更多詳細資訊，請參閱 Aurora 資料庫叢集的容錯能力。
Backup retention period (備份保留期間)	選取從 1 到 35 天的時間長度，Aurora 會在此時間內保留資料庫備份副本。備份副本可用來進行資料庫的 point-in-time 還原 (PITR) 作業，單位可計算至秒。
Enhanced Monitoring (增強型監控)	選擇 Enable enhanced monitoring (啟用增強型監控)，以針對資料庫叢集執行所在的作業系統即時收集指標。如需更多詳細資訊，請參閱 使用增強型監控來監控作業系統指標。
監控角色	只有在 Enhanced Monitoring (增強型監控) 設為 Enable enhanced monitoring (啟用增強型監控) 時才能使用。選擇您所建立的 IAM 角色，以允許 Aurora 為您與 Amazon CloudWatch Logs 進行通訊，或選擇 Default (預設值)，讓 Aurora 為您建立名為 `rds-monitoring-role` 的角色。如需更多詳細資訊，請參閱 使用增強型監控來監控作業系統指標。
精細程度	只有在 Enhanced Monitoring (增強型監控) 設為 Enable enhanced monitoring (啟用增強型監控) 時才能使用。針對資料庫叢集，設定收集指標之間的間隔 (以秒為單位)。
Auto minor version upgrade (自動次要版本升級)	此設定不適用於 Aurora MySQL 資料庫叢集。 如需 Aurora MySQL 引擎更新的詳細資訊，請參閱 Amazon Aurora MySQL 的資料庫引擎更新。
Maintenance window (維護時段)	選取 Select window (選取時段)，並指定可能發生系統維護的每週時間範圍。或者，選取 No preference (無偏好設定)，讓 Aurora 隨機指派期間。

6. 選擇 Create read replica (建立僅供讀取複本)。

AWS CLI

若要從來源 RDS for MySQL 資料庫執行個體建立 Aurora 僅供讀取複本，請使用 `create-db-cluster`和 `create-db-instance` AWS CLI 命令建立新的 Aurora MySQL 資料庫叢集。當您呼叫 `create-db-cluster` 命令時，請加上 `--replication-source-identifier` 參數，以識別來源 MySQL 資料庫執行個體的 Amazon Resource Name (ARN)。如需 Amazon RDS ARN 的詳細資訊，請參閱 Amazon Relational Database Service (Amazon RDS)。

請勿指定主要使用者名稱、主要密碼或資料庫名稱，因為 Aurora 僅供讀取複本會使用與來源 MySQL 資料庫執行個體相同的主要使用者名稱、主要密碼、資料庫名稱。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine
 aurora \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 \
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-
mysql-instance
```

在 Windows 中：

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine
 aurora ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 ^
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-
mysql-instance
```

如果您使用主控台建立 Aurora 僅供讀取複本，則 Aurora 會為您的資料庫叢集 Aurora 僅供讀取複本自動建立主要執行個體。如果您使用 AWS CLI 建立 Aurora 僅供讀取複本，則必須明確建立資料庫叢集的主要執行個體。主要執行個體是資料庫叢集內第一個建立的執行個體。

您可以使用 `create-db-instance` AWS CLI 命令搭配下列參數，為您的資料庫叢集建立主要執行個體。

- `--db-cluster-identifier`

  資料庫叢集的名稱。
- `--db-instance-class`

要用於主要執行個體的執行個體類別名稱。

- `--db-instance-identifier`

  主要執行個體的名稱。

- `--engine aurora`

在此範例中，您會使用在 *myinstanceclass* 中指定的資料庫執行個體類別，為 *myreadreplicacluster* 資料庫叢集建立主要執行個體 *myreadreplicainstance*。

Example

對於 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
    --db-cluster-identifier myreadreplicacluster \
    --db-instance-class myinstanceclass \
    --db-instance-identifier myreadreplicainstance \
    --engine aurora
```

在 Windows 中：

```
aws rds create-db-instance ^
    --db-cluster-identifier myreadreplicacluster ^
    --db-instance-class myinstanceclass ^
    --db-instance-identifier myreadreplicainstance ^
    --engine aurora
```

RDS API

若要從來源 RDS for MySQL 資料庫執行個體建立 Aurora 讀取複本，請使用 `CreateDBCluster` 和 `CreateDBInstance` Amazon RDS API 命令建立新的 Aurora 資料庫叢集和主要執行個體。請勿指定主要使用者名稱、主要密碼或資料庫名稱，因為 Aurora 讀取複本會使用與來源 RDS for MySQL 資料庫執行個體相同的主要使用者名稱、主要密碼、資料庫名稱。

您可以使用 `CreateDBCluster` Amazon RDS API 命令搭配以下參數，從來源 RDS for MySQL 資料庫執行個體為 Aurora 讀取複本建立新的 Aurora 資料庫叢集：

- `DBClusterIdentifier`

要建立的資料庫叢集名稱。

- `DBSubnetGroupName`

  要與此資料庫叢集關聯之資料庫子網路群組的名稱。

- `Engine=aurora`
- `KmsKeyId`

  AWS KMS key 選擇性地使用 加密資料庫叢集，取決於您的 MySQL 資料庫執行個體是否已加密。

  - 如果 MySQL 資料庫執行個體未加密，請指定加密金鑰，以便對資料庫叢集進行靜態加密。否則會使用您的帳戶的預設加密金鑰，對資料庫叢集進行靜態加密。
  - 如果 MySQL 資料庫執行個體已加密，請指定加密金鑰，以使用指定的加密金鑰對資料庫叢集進行靜態加密。否則會使用 MySQL 資料庫執行個體的加密金鑰，對資料庫叢集進行靜態加密。

  > **Note**
  >
  > 您無法從已加密的 MySQL 資料庫執行個體來建立未加密的資料庫叢集。

- `ReplicationSourceIdentifier`

  來源 MySQL 資料庫執行個體的 Amazon Resource Name (ARN)。如需 Amazon RDS ARN 的詳細資訊，請參閱 Amazon Relational Database Service (Amazon RDS)。

- `VpcSecurityGroupIds`

  要與此資料庫叢集關聯的 EC2 VPC 安全群組的清單。

在此範例中，您會從來源 MySQL 資料庫執行個體建立資料庫叢集 *myreadreplicacluster*，此來源的 ARN 設為 *mysqlprimaryARN*，並與 *mysubnetgroup* 資料庫子網路群組及 *mysecuritygroup* VPC 安全群組相關聯。

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CreateDBCluster
    &DBClusterIdentifier=myreadreplicacluster
    &DBSubnetGroupName=mysubnetgroup
    &Engine=aurora
    &ReplicationSourceIdentifier=mysqlprimaryARN
```

```
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-10-31
    &VpcSecurityGroupIds=mysecuritygroup
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20150927/us-east-1/rds/aws4_request
    &X-Amz-Date=20150927T164851Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=6a8f4bd6a98f649c75ea04a6b3929ecc75ac09739588391cd7250f5280e716db
```

如果您使用主控台建立 Aurora 僅供讀取複本，則 Aurora 會為您的資料庫叢集 Aurora 僅供讀取複本自動建立主要執行個體。如果您使用 AWS CLI 建立 Aurora 僅供讀取複本，則必須明確建立資料庫叢集的主要執行個體。主要執行個體是資料庫叢集內第一個建立的執行個體。

您可以使用 CreateDBInstance Amazon RDS API 命令搭配以下參數來建立資料庫叢集的主要執行個體：

- `DBClusterIdentifier`

  資料庫叢集的名稱。

- `DBInstanceClass`

  要用於主要執行個體的執行個體類別名稱。

- `DBInstanceIdentifier`

  主要執行個體的名稱。

- `Engine=aurora`

在此範例中，您會使用在 *myinstanceclass* 中指定的資料庫執行個體類別，為 *myreadreplicacluster* 資料庫叢集建立主要執行個體 *myreadreplicainstance*。

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CreateDBInstance
    &DBClusterIdentifier=myreadreplicacluster
    &DBInstanceClass=myinstanceclass
    &DBInstanceIdentifier=myreadreplicainstance
    &Engine=aurora
    &SignatureMethod=HmacSHA256
```

```
    &SignatureVersion=4
    &Version=2014-09-01
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20140424/us-east-1/rds/aws4_request
    &X-Amz-Date=20140424T194844Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=bee4aabc750bf7dad0cd9e22b952bd6089d91e2a16592c2293e532eeaab8bc77
```

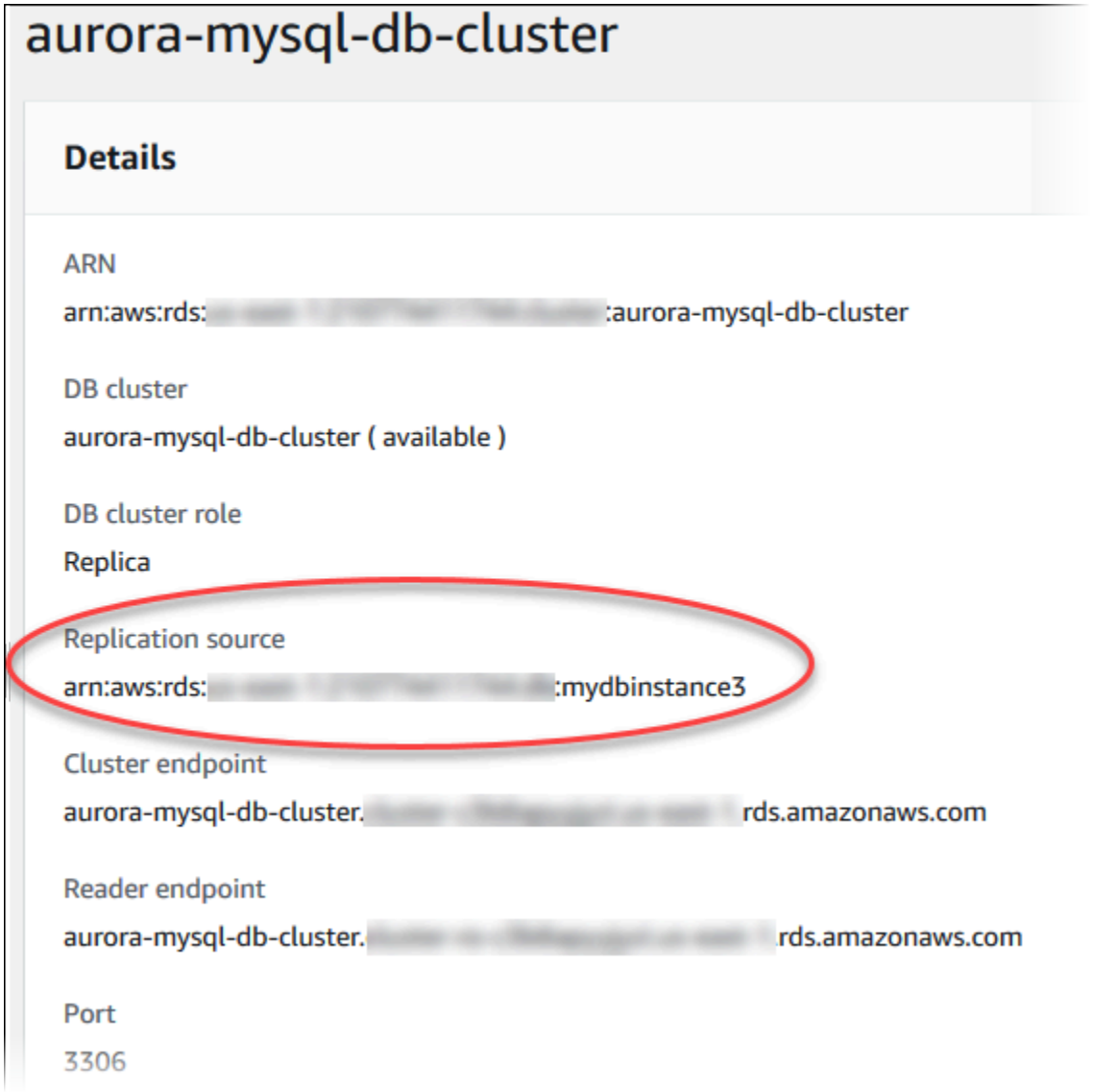
檢視 Aurora 僅供讀取複本

您可以使用 AWS 管理主控台 或 AWS CLI，以檢視 Aurora MySQL 資料庫叢集的 MySQL 至 Aurora MySQL 複寫關係。

主控台

檢視 Aurora 僅供讀取複本的主要 MySQL 資料庫執行個體

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇 Aurora 僅供讀取複本的資料庫叢集，以顯示其詳細資訊。主要 MySQL 資料庫執行個體資訊在 Replication source (複寫來源) 欄位中。



AWS CLI

若要使用 檢視 Aurora MySQL 資料庫叢集的 MySQL 對 Aurora MySQL 複寫關係 AWS CLI，請使用 `describe-db-clusters`和 `describe-db-instances`命令。

若要判斷哪個 MySQL 資料庫執行個體是主節點，請使用 `describe-db-clusters`，並在 `--db-cluster-identifier` 選項中指定 Aurora 僅供讀取複本的叢集識別符。請查閱輸出中的 `ReplicationSourceIdentifier` 元素，以取得做為複寫主節點之資料庫執行個體的 ARN。

若要判斷哪個資料庫叢集是 Aurora 僅供讀取複本，請使用 `describe-db-instances`，並在 `--db-instance-identifier` 選項中指定 MySQL 資料庫執行個體的執行個體識別符。請參閱輸出中的 `ReadReplicaDBClusterIdentifiers` 元素，以取得 Aurora 僅供讀取複本的資料庫叢集識別符。

Example

對於 Linux、macOS 或 Unix：

```
aws rds describe-db-clusters \
    --db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances \
    --db-instance-identifier mysqlprimary
```

在 Windows 中：

```
aws rds describe-db-clusters ^
    --db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances ^
    --db-instance-identifier mysqlprimary
```

提升 Aurora 僅供讀取複本

遷移完成後，您可以使用 AWS 管理主控台 或 將 Aurora 僅供讀取複本提升為獨立的資料庫叢集 AWS CLI。

接著您可以將用戶端應用程式導向 Aurora 僅供讀取複本的端點。如需 Aurora 端點的詳細資訊，請參閱 Amazon Aurora 端點連線。提升應該會很快完成，而且您在提升期間可以讀取和寫入 Aurora 僅供讀取複本。但在此期間，您無法刪除主要 MySQL 資料庫執行個體，或解除資料庫執行個體與 Aurora 僅供讀取複本之間的連結。

在提升 Aurora 僅供讀取複本之前，請先停止任何交易寫入來源 MySQL 資料庫執行個體，並等待 Aurora 僅供讀取複本的複本延遲達到 0。您可以檢視 Aurora 僅供讀取複本的複本延遲，方法是在您的 Aurora 僅供讀取複本上呼叫 `SHOW SLAVE STATUS` (Aurora MySQL 第 2 版) 或 `SHOW REPLICA STATUS` (Aurora MySQL 第 3 版) 命令。檢查 Seconds behind master 值。

當交易停止寫入主節點且複本延遲為 0 之後，您就可以開始寫入 Aurora 僅供讀取複本。如果您在此之前寫入 Aurora 僅供讀取複本，且修改 MySQL 主節點上也正在修改的資料表，則可能會中斷複寫至 Aurora 的作業。如果發生這種情況，您必須刪除並重新建立 Aurora 僅供讀取複本。

主控台

將 Aurora 僅供讀取複本提升為 Aurora 資料庫叢集

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇 Aurora 僅供讀取複本的資料庫叢集。
4. 針對 Actions (動作)，選擇 Promote (提升)。
5. 選擇 Promote Read Replica (提升僅供讀取複本)。

升級後，請使用下列程序確認提升已完成。

確認已提升 Aurora 僅供讀取複本

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Events (事件)。
3. 在 Events (事件) 頁面上，確認您提升的叢集有 `Promoted Read Replica cluster to a stand-alone database cluster` 事件。

提升完成後，主要 MySQL 資料庫執行個體與 Aurora 僅供讀取複本會解除連結，您可以安心地刪除資料庫執行個體 (如果想要這麼做)。

AWS CLI

若要將 Aurora 僅供讀取複本提升為獨立資料庫叢集，請使用 `promote-read-replica-db-cluster` AWS CLI 命令。

Example

對於 Linux、macOS 或 Unix：

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier myreadreplicacluster
```

在 Windows 中：

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier myreadreplicacluster
```

管理 Amazon Aurora MySQL

下列小節討論如何管理 Amazon Aurora MySQL 資料庫叢集。

主題

- 管理 Amazon Aurora MySQL 的效能和擴展
- 恢復 Aurora 資料庫叢集
- 使用錯誤注入查詢測試 Amazon Aurora MySQL
- 使用快速 DDL 更改 Amazon Aurora 中的資料表
- 顯示 Aurora MySQL 資料庫叢集的磁碟區狀態

管理 Amazon Aurora MySQL 的效能和擴展

擴展 Aurora MySQL 資料庫執行個體

有兩種方式可以擴展 Aurora MySQL 資料庫執行個體：執行個體擴展和讀取擴展。如需讀取擴展的詳細資訊，請參閱讀取擴展。

您可以修改資料庫叢集中每個資料庫執行個體的資料庫執行個體類別，來擴展 Aurora MySQL 資料庫叢集。Aurora MySQL 支援數個針對 Aurora 進行最佳化的資料庫執行個體類別。請勿針對大小超過 40 TB 的較大 Aurora 叢集，請使用 db.t2 或 db.t3 執行個體類別。如需 Aurora MySQL 支援資料庫執行個體類別的規格，請參閱Amazon Aurora 資料庫執行個體類別。

> **Note**
>
> 建議您在開發、測試伺服器或其他非生產伺服器時，僅使用 T 資料庫執行個體類別。如需詳細了解 T 執行個體類別，請參閱 使用 T 執行個體類別進行開發和測試。

對 Aurora MySQL 資料庫執行個體的連線數上限

允許對 Aurora MySQL 資料庫執行個體建立的連線數上限，由資料庫執行個體的執行個體層級參數群組中的 `max_connections` 參數決定。

下表針對 Aurora MySQL 可用的每個資料庫執行個體類別，列出產生的 `max_connections` 預設值。您可以透過將執行個體向上擴展至具有更多記憶體的資料庫執行個體類別，或是為資料庫參數群組中 `max_connections` 參數設定較大的值 (最高 16,000)，藉此為您的 Aurora MySQL 資料庫執行個體增加連線數上限。

> **Tip**
>
> 如果您的應用程式經常開啟和關閉連線，或者保持大量長期連線開啟，我們建議您使用 Amazon RDS Proxy。RDS 代理是個完全受管、高可用性的資料庫代理，其使用連線集區，安全且高效地共用資料庫連線。如要進一步了解 RDS 代理，請參閱 Amazon RDS Proxy for Aurora。

如需 Aurora Serverless v2 執行個體處理此參數的詳細資訊，請參閱 Aurora Serverless v2 的連線數上限。

執行個體類別	max_connections 預設值		
db.t2.small	45		
db.t2.medium	90		
db.t3.small	45		
db.t3.medium	90		
db.t3.large	135		
db.t4g.medium	90		
db.t4g.large	135		
db.r3.large	1000		
db.r3.xlarge	2000		
db.r3.2xlarge	3000		
db.r3.4xlarge	4000		
db.r3.8xlarge	5000		
db.r4.large	1000		

執行個體類別	max_connections 預設值		
db.r4.xlarge	2000		
db.r4.2xlarge	3000		
db.r4.4xlarge	4000		
db.r4.8xlarge	5000		
db.r4.16xlarge	6000		
db.r5.large	1000		
db.r5.xlarge	2000		
db.r5.2xlarge	3000		
db.r5.4xlarge	4000		
db.r5.8xlarge	5000		
db.r5.12xlarge	6000		
db.r5.16xlarge	6000		
db.r5.24xlarge	7000		
db.r6g.large	1000		
db.r6g.xlarge	2000		
db.r6g.2xlarge	3000		
db.r6g.4xlarge	4000		
db.r6g.8xlarge	5000		
db.r6g.12xlarge	6000		

執行個體類別	max_connections 預設值		
db.r6g.16xlarge	6000		
db.r6i.large	1000		
db.r6i.xlarge	2000		
db.r6i.2xlarge	3000		
db.r6i.4xlarge	4000		
db.r6i.8xlarge	5000		
db.r6i.12xlarge	6000		
db.r6i.16xlarge	6000		
db.r6i.24xlarge	7000		
db.r6i.32xlarge	7000		
db.r7g.large	1000		
db.r7g.xlarge	2000		
db.r7g.2xlarge	3000		
db.r7g.4xlarge	4000		
db.r7g.8xlarge	5000		
db.r7g.12xlarge	6000		
db.r7g.16xlarge	6000		
db.r7i.large	1000		
db.r7i.xlarge	2000		

執行個體類別	max_connections 預設值		
db.r7i.2xlarge	3000		
db.r7i.4xlarge	4000		
db.r7i.8xlarge	5000		
db.r7i.12xlarge	6000		
db.r7i.16xlarge	6000		
db.r7i.24xlarge	7000		
db.r7i.48xlarge	8000		
db.r8g.large	1000		
db.r8g.xlarge	2000		
db.r8g.2xlarge	3000		
db.r8g.4xlarge	4000		
db.r8g.8xlarge	5000		
db.r8g.12xlarge	6000		
db.r8g.16xlarge	6000		
db.r8g.24xlarge	7000		
db.r8g.48xlarge	8000		
db.x2g.large	2000		
db.x2g.xlarge	3000		
db.x2g.2xlarge	4000		

執行個體類別	max_connections 預設值		
db.x2g.4xlarge	5000		
db.x2g.8xlarge	6000		
db.x2g.12xlarge	7000		
db.x2g.16xlarge	7000		

> **Tip**
>
> `max_connections` 參數計算使用對數底數 2 (與自然對數不同)，以及所選 Aurora MySQL 執行個體類別的位元組 `DBInstanceClassMemory` 值。參數僅接受整數值，小數部分會從計算中截斷。公式會實作連線限制，如下所示：
>
> - 大型 R3, R4 和 R5 執行個體的連線遞增為 1000
> - T2 和 T3 執行個體記憶體變體的連線遞增為 45
>
> 範例：對於 db.r6g.large，當公式計算 1069.2 時，系統會實作 1000 以維持一致的遞增模式。

如果您建立新的參數群組以針對連線限制自訂您自己的預設值，則會看到系統根據 `DBInstanceClassMemory` 值使用公式來衍生預設連線限制。如上表所示，公式產生的連線限制會增加 1000，因為記憶體會在逐漸增大的 R3、R4 與 R5 執行個體之間加倍，而對於 T2 及 T3 執行個體的不同記憶體大小則增加 45。

請參閱 指定資料庫參數 以取得更多 `DBInstanceClassMemory` 計算方式的詳細資訊。

Aurora MySQL 和 RDS for MySQL 資料庫執行個體具有不同的記憶體額外負荷量。因此，使用相同執行個體類別的 Aurora MySQL 和 RDS for MySQL 資料庫執行個體的 `max_connections` 值可能會有所不同。資料表中的數值僅適用於 Aurora MySQL 資料庫執行個體。

> **Note**
>
> T2 及 T3 執行個體的連線限制相對較低，原因是 Aurora 執行個體類別僅供開發和測試案例使用，而不用於生產工作負載。

預設連線限制是針對使用其他主要記憶體耗用者 (例如緩衝集區和查詢快取) 之預設值的系統而調整的。如果您針對叢集變更那些其他設定，請根據資料庫執行個體上可用記憶體的增加或減少比例來考慮調整連線限制。

Aurora MySQL 的暫存空間限制

Aurora MySQL 會將資料表和索引存放在 Aurora 儲存子系統。Aurora MySQL 對非持久性暫存檔案和非 InnoDB 暫存資料表會使用個別的暫存或本機儲存體。本機儲存體也包括用於查詢處理期間排序大型資料集或用於索引建置作業等目的的檔案。它不包括 InnoDB 暫存資料表。

如需 Aurora MySQL 第 3 版中暫存資料表的詳細資訊，請參閱 Aurora MySQL 第 3 版的新暫時資料表行為。如需第 2 版中暫存資料表的詳細資訊，請參閱 Aurora MySQL 第 2 版的暫存資料表行為。

這些磁碟區上的資料和暫存檔案會在啟動和停止資料庫執行個體，以及主機取代期間遺失。

這些本機儲存磁碟區由 Amazon Elastic Block Store (EBS) 支援，且可透過使用更大的資料庫執行個體類別來擴充。如需儲存體的詳細資訊，請參閱Amazon Aurora 儲存體。

本機儲存體也用於使用 `LOAD DATA FROM S3` 或 `LOAD XML FROM S3` 從 Amazon S3 匯入資料，以及使用 SELECT INTO OUTFILE S3 將資料匯出至 S3。如需從 S3 匯入和匯出的詳細資訊，請參閱下列內容：

- 從 Amazon S3 儲存貯體中的文字檔案將資料載入 Amazon Aurora MySQL 資料庫叢集
- 將來自 Amazon Aurora MySQL 資料庫叢集的資料儲存至 Amazon S3 儲存貯體中的文字檔案

Aurora MySQL 對大多數 Aurora MySQL 資料庫執行個體類別 (不包括爆量效能執行個體類別類型，例如 db.t2、db.t3 和 db.t4g) 的錯誤日誌、一般日誌、緩慢查詢日誌和稽核日誌使用單獨的永久儲存體。啟動和停止資料庫執行個體時，以及在主機取代期間，系統會保留此磁碟區上的資料。

此永久儲存磁碟區也由 Amazon EBS 提供支援，並根據資料庫執行個體類別具有固定大小。它無法使用較大的資料庫執行個體類別進行擴展。

下表顯示每個 Aurora MySQL 資料庫執行個體類別可用的暫存和永久空間容量上限。如需 Aurora 之資料庫執行個體類別支援的詳細資訊，請參閱 Amazon Aurora 資料庫執行個體類別。

DB instance class (資料庫執行個體類別)	可用的暫存/本機空間上限 (GiB)	日誌檔案可用的額外最大儲存空間 (GiB)
db.x2g.16xlarge	1280	500
db.x2g.12xlarge	960	500
db.x2g.8xlarge	640	500
db.x2g.4xlarge	320	500
db.x2g.2xlarge	160	60
db.x2g.xlarge	80	60
db.x2g.large	40	60
db.r8g.48xlarge	3840	500
db.r8g.24xlarge	1920	500
db.r8g.16xlarge	1280	500
db.r8g.12xlarge	960	500
db.r8g.8xlarge	640	500
db.r8g.4xlarge	320	500
db.r8g.2xlarge	160	60
db.r8g.xlarge	80	60
db.r8g.large	32	60
db.r7i.48xlarge	3840	500
db.r7i.24xlarge	1920	500
db.r7i.16xlarge	1280	500
db.r7i.12xlarge	960	500

DB instance class (資料庫執行個體類別)	可用的暫存/本機空間上限 (GiB)	日誌檔案可用的額外最大儲存空間 (GiB)
db.r7i.8xlarge	640	500
db.r7i.4xlarge	320	500
db.r7i.2xlarge	160	60
db.r7i.xlarge	80	60
db.r7i.large	32	60
db.r7g.16xlarge	1280	500
db.r7g.12xlarge	960	500
db.r7g.8xlarge	640	500
db.r7g.4xlarge	320	500
db.r7g.2xlarge	160	60
db.r7g.xlarge	80	60
db.r7g.large	32	60
db.r6i.32xlarge	2560	500
db.r6i.24xlarge	1920	500
db.r6i.16xlarge	1280	500
db.r6i.12xlarge	960	500
db.r6i.8xlarge	640	500
db.r6i.4xlarge	320	500
db.r6i.2xlarge	160	60
db.r6i.xlarge	80	60

DB instance class (資料庫執行個體類別)	可用的暫存/本機空間上限 (GiB)	日誌檔案可用的額外最大儲存空間 (GiB)
db.r6i.large	32	60
db.r6g.16xlarge	1280	500
db.r6g.12xlarge	960	500
db.r6g.8xlarge	640	500
db.r6g.4xlarge	320	500
db.r6g.2xlarge	160	60
db.r6g.xlarge	80	60
db.r6g.large	32	60
db.r5.24xlarge	1920	500
db.r5.16xlarge	1280	500
db.r5.12xlarge	960	500
db.r5.8xlarge	640	500
db.r5.4xlarge	320	500
db.r5.2xlarge	160	60
db.r5.xlarge	80	60
db.r5.large	32	60
db.r4.16xlarge	1280	500
db.r4.8xlarge	640	500
db.r4.4xlarge	320	500
db.r4.2xlarge	160	60

DB instance class (資料庫執行個體類別)	可用的暫存/本機空間上限 (GiB)	日誌檔案可用的額外最大儲存空間 (GiB)
db.r4.xlarge	80	60
db.r4.large	32	60
db.t4g.large	32	–
db.t4g.medium	32	–
db.t3.large	32	–
db.t3.medium	32	–
db.t3.small	32	–
db.t2.medium	32	–
db.t2.small	32	–

Important

這些值代表每個資料庫執行個體中理論上可用的儲存容量上限。您可用的實際本機儲存空間可能較低。Aurora 會將一些本機儲存空間用於其管理過程，甚至在您載入任何資料之前，資料庫執行個體都會使用一些本機儲存空間。您可以使用 `FreeLocalStorage` CloudWatch 指標來監控特定資料庫執行個體可用的暫存空間，如 Amazon Aurora 的 Amazon CloudWatch 指標中所述。您可以檢查目前可用的免費儲存空間量。您也可以將一段時間內的免費儲存空間量繪製成圖表。監控一段時間內的免費儲存空間量，可協助您判斷值是增加還是減少，或可找出最小值、最大值或平均值。
(這不適用於 Aurora Serverless v2。)

恢復 Aurora 資料庫叢集

利用 Amazon Aurora MySQL 相容版本，您可以將資料庫叢集恢復到特定時間，而不需從備份還原資料。

內容

- 恢復概觀
  - 恢復時段
  - 恢復時間
  - 恢復限制
- 區域和版本可用性
- 啟用恢復之叢集的升級考量
- 設定恢復 Aurora MySQL 資料庫叢集
- 執行 Aurora MySQL 資料庫叢集的恢復
- 監控 Aurora MySQL 資料庫叢集的恢復
- 使用主控台訂閱恢復事件
- 擷取現有恢復
- 停用 Aurora MySQL 資料庫叢集的恢復

恢復概觀

恢復會將資料庫叢集「倒轉」至您指定的時間。恢復不是備份資料庫叢集的替代方式，因此您可以將它還原至某個時間點。不過，相較於傳統的備份和還原，恢復提供下列優點：

- 您可以輕鬆復原錯誤。如果您不慎地執行破壞性的動作 (例如不帶 WHERE 子句的 DELETE)，您可以將資料庫叢集恢復到破壞性動作之前的某個時間點，而使得對服務造成的中斷降到最低。
- 您可以快速恢復資料庫叢集。將資料庫叢集還原到某個時間點會啟動新的資料庫叢集，並將它從備份資料或資料庫叢集快照還原，這可能需要數小時的時間。恢復資料庫叢集不需要新的資料庫叢集，並且倒轉資料庫叢集只需要幾分鐘。
- 您可以瀏覽稍早的資料變更。您可以重複地往返恢復時間點中的資料庫叢集，以幫助判定何時發生了資料變更。例如，您可以將資料庫叢集往前恢復三小時，然後再恢復至該時間點的一小時後。在此情況下，恢復時間為原始時間前的兩小時。

> **Note**
>
> 如需將資料庫叢集還原至某個時間點的詳細資訊，請參閱 備份與還原 Aurora 資料庫叢集的概觀。

恢復時段

使用恢復時，會有目標恢復時段和實際恢復時段：

- 目標恢復時段為您希望資料庫叢集能夠恢復的時間量。啟用恢復時，您會指定目標恢復時段。例如，如果您希望能夠恢復一天的資料庫叢集，可以指定 24 小時的目標恢復時段。
- 實際恢復時段為您的資料庫叢集可以恢復的實際時間量，其可能小於目標恢復時段。實際恢復時段會基於您的工作負載和可用於存放資料庫變更相關資訊的儲存體 (稱為變更記錄)。

當您在啟用恢復功能的情況下更新 Aurora 資料庫叢集時，即會產生變更記錄。Aurora 會保留目標恢復視窗的變更記錄，而且您需要按小時付費才能儲存它們。您的資料庫叢集上的目標恢復時段和工作負載都會決定您存放的變更記錄數量。工作負載為在指定時間中您對資料庫叢集的變更量。如果工作負載繁重，即您在恢復時段時存放的變更記錄量會比工作負載較輕時所存放的變更記錄量還多。

您可以將目標恢復時段視為您希望資料庫叢集能夠恢復的時間上限目標。在多數情況中，您可以恢復您指定的時間上限。不過，在部分情況中，資料庫叢集無法儲存足夠變更記錄來恢復時間上限，因此您的實際恢復時段會小於您的目標。一般來說，當資料庫叢集上的工作負載極為繁重時，實際恢復時段會小於目標。實際恢復時段小於目標時，我們會傳送通知給您。

對資料庫叢集啟用恢復，而您刪除儲存在資料庫叢集中的資料表時，Aurora 會將該資料表保留在恢復變更記錄中。它會這麼做，使得您可以還原到刪除資料表之前的時間。如果在您的恢復時段中沒有足夠空間可存放該資料表，則最終可能會從恢復變更記錄中移除該資料表。

恢復時間

Aurora 一律會恢復到對資料庫叢集一致的時間。這麼做可將恢復完成時交易未遞交的可能性降到最低。指定恢復的時間時，Aurora 會自動選擇最接近的可能一致時間。此方法表示已完成的恢復可能不完全符合您指定的時間，但您可以使用 describe-db-cluster-backtracks AWS CLI 命令來判斷恢復的確切時間。如需詳細資訊，請參閱擷取現有恢復。

恢復限制

下列限制適用恢復：

- 您僅在透過將恢復功能啟用時所建立的資料庫叢集上才可使用恢復。您無法修改資料庫叢集來啟用恢復功能。您可以在建立新資料庫叢集或還原資料庫叢集的快照時啟用恢復功能。
- 恢復時段的限制為 72 小時。
- 恢復會影響整個資料庫叢集。例如，您不可以選擇性地恢復單一資料表或單一資料更新。

- 您無法從啟用恢復功能的叢集建立跨區域僅供讀取複本，但您仍然可以在叢集上啟用二進位日誌 (binlog) 複寫。，如果您嘗試恢復已啟用二進位日誌的資料庫叢集，除非您選擇強制恢復，否則會發生一般錯誤。任何強制恢復的嘗試都會使下游僅供讀取複本停頓，並干擾其他操作，例如藍/綠部署。
- 您不可以將資料庫複製恢復到建立資料庫複製之前的時間。不過，您可以使用原始資料庫來恢復到建立複製之前的時間。如需資料庫複製的詳細資訊，請參閱 複製 Amazon Aurora 資料庫叢集的一個磁碟區。
- 恢復會造成短暫的資料庫執行個體干擾。您必須在開始恢復操作之前停止或暫停應用程式，以確保沒有新的讀取或寫入請求。在恢復操作期間，Aurora 會暫停資料庫、關閉任何開啟中連線，並捨棄任何未遞交的讀取和寫入。然後會等候恢復操作完成。
- 您無法在不支援恢復的 AWS 區域中還原已啟用恢復功能的叢集的跨區域快照。
- 如果您對啟用恢復的叢集從 Aurora MySQL 版本 2 就地升級至版本 3，無法恢復到升級發生之前的時間點。

區域和版本可用性

回溯不適用於 Aurora PostgreSQL。

下列是 Aurora MySQL 進行恢復時支援的引擎和區域可用性。

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
美國東部 (維吉尼亞北部)	所有版本	所有版本
美國東部 (俄亥俄)	所有版本	所有版本
美國西部 (加利佛尼亞北部)	所有版本	所有版本
美國西部 (奧勒岡)	所有版本	所有版本
非洲 (開普敦)	–	–

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
Asia Pacific (Hong Kong)	–	–
亞太地區 (雅加達)	–	–
亞太地區 (馬來西亞)	–	–
亞太地區 (墨爾本)	–	–
亞太地區 (孟買)	所有版本	所有版本
亞太區域 (紐西蘭)	–	–
亞太地區 (大阪)	所有版本	2.07.3 版及更新版本
亞太地區 (首爾)	所有版本	所有版本
亞太地區 (新加坡)	所有版本	所有版本
亞太地區 (雪梨)	所有版本	所有版本
亞太區域 (台北)	–	–
亞太區域 (泰國)	–	–
亞太地區 (東京)	所有版本	所有版本
加拿大 (中部)	所有版本	所有版本
加拿大西部 (卡加利)	–	–
中國 (北京)	–	–

區域	Aurora MySQL 第 3 版	Aurora MySQL 第 2 版
中國 (寧夏)	–	–
歐洲 (法蘭克福)	所有版本	所有版本
歐洲 (愛爾蘭)	所有版本	所有版本
歐洲 (倫敦)	所有版本	所有版本
Europe (Milan)	–	–
Europe (Paris)	所有版本	所有版本
歐洲 (西班牙)	–	–
歐洲 (斯德哥爾摩)	–	–
歐洲 (蘇黎世)	–	–
以色列 (特拉維夫)	–	–
墨西哥 (中部)	–	–
Middle East (Bahrain)	–	–
中東 (阿拉伯聯合大公國)	–	–
南美洲 (聖保羅)	–	–
AWS GovCloud（美國東部）	–	–
AWS GovCloud（美國西部）	–	–

啟用恢復之叢集的升級考量

您可以從 Aurora MySQL 第 2 版和第 3 版升級啟用恢復的資料庫叢集，因為 Aurora MySQL 第 3 版的所有次要版本都支援恢復。

設定恢復 Aurora MySQL 資料庫叢集

若要使用恢復功能, 您必須啟用恢復並指定目標恢復時段。否則會停用恢復。

針對目標恢復時段，指定您希望能夠使用恢復倒轉資料庫的時間。Aurora 會嘗試保留足夠的變更記錄來支援該時間時段。

主控台

您可以在建立新資料庫叢集時使用主控台來設定恢復。您還可以修改資料庫叢集，以變更啟用恢復之叢集的恢復視窗。如果您透過將恢復視窗設定為 0 來完全關閉叢集的恢復功能，則無法再次為該叢集啟用恢復。

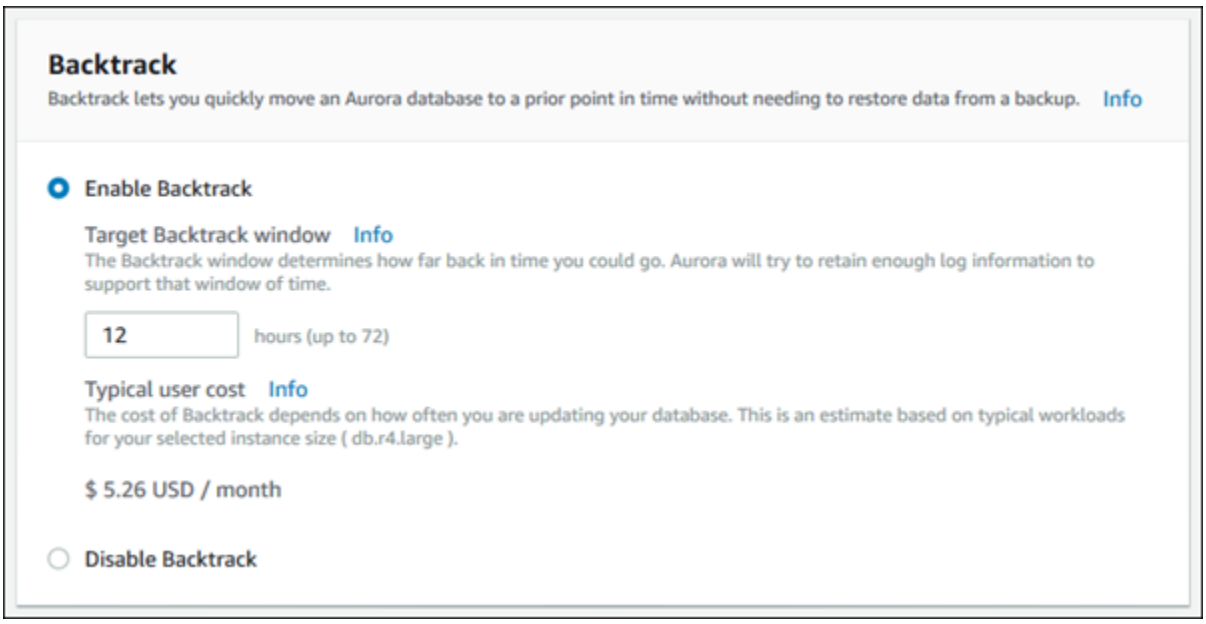
主題

- 建立資料庫叢集時使用主控台來設定恢復
- 修改資料庫叢集時使用主控台設定恢復

建立資料庫叢集時使用主控台來設定恢復

建立新 Aurora MySQL 資料庫叢集時，您會在 Backtrack (恢復) 區段中選擇 Enable Backtrack (啟用恢復)，並指定大於零的 Target Backtrack window (目標恢復時段) 值時設定恢復。

若要建立資料庫叢集，請遵循 建立 Amazon Aurora 資料庫叢集 中的指示。下圖顯示 Backtrack (恢復) 區段。



建立新資料庫叢集時，Aurora 沒有資料庫叢集工作負載的資料。因此它不能專為新資料庫叢集估計成本。相反地，主控台會根據一般工作負載為指定的目標恢復時段呈現一般使用者成本。一般成本的目的是為恢復功能的成本提供一般參考。

> **Important**
>
> 您的實際成本可能會符合一般成本，因為您的實際成本是根據資料庫叢集的工作負載。

修改資料庫叢集時使用主控台設定恢復

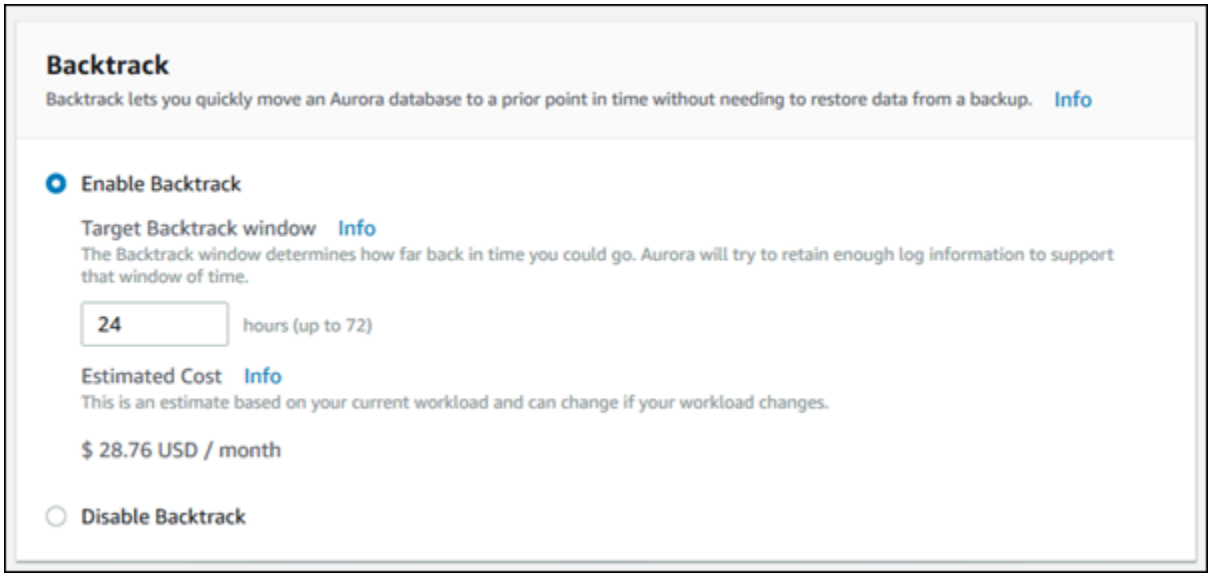
您可以使用主控台修改資料庫叢集的恢復。

> **Note**
>
> 目前，您只能為已啟用恢復功能的資料庫叢集修改恢復。對於在已停用恢復功能的情況下而建立的資料庫叢集，或如果資料庫叢集已停用「恢復」功能，則不會顯示 Backtrack (恢復) 區段。

使用主控台修改資料庫叢集的恢復

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。

2. 選擇 Databases (資料庫)。
3. 選擇您要修改的叢集，並選擇 Modify (修改)。
4. 針對 Target Backtrack window (目標恢復時段)，修改您希望能夠恢復的時間。限制為 72 小時。



主控台會根據資料庫叢集的過去工作負載顯示指定時間的估計成本：

- 如果已在資料庫叢集上停用恢復，成本的估計會根據 Amazon CloudWatch 中資料庫叢集的 `VolumeWriteIOPS` 指標。
- 如果先前已在資料庫叢集上啟用恢復，成本的估計會根據 Amazon CloudWatch 中資料庫叢集的 `BacktrackChangeRecordsCreationRate` 指標。

5. 選擇 Continue (繼續)。
6. 對於 Scheduling of Modifications (修改排程)，選擇以下其中一項：
   - Apply during the next scheduled maintenance window (在下一個排定的維護時段套用) – 在下一個維護時段前，等候套用 Target Backtrack window (目標恢復時段) 修改。
   - Apply immediately (立即套用) – 盡快套用 Target Backtrack window (目標恢復時段) 修改。
7. 選擇 Modify cluster (修改叢集)。

AWS CLI

當您使用 create-db-cluster AWS CLI 命令建立新的 Aurora MySQL 資料庫叢集時，當您指定大於零`--backtrack-window`的值時，會設定恢復。`--backtrack-window` 值會指定目標恢復時段。如需詳細資訊，請參閱建立 Amazon Aurora 資料庫叢集。

您也可以使用下列 CLI AWS 命令指定 `--backtrack-window`值：

- modify-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

下列程序說明如何使用 AWS CLI來修改資料庫叢集的目標恢復時段。

使用 修改資料庫叢集的目標恢復時段 AWS CLI

- 呼叫 modify-db-cluster AWS CLI 命令並提供下列值：
  - `--db-cluster-identifier` – 資料庫叢集的名稱。
  - `--backtrack-window` – 您希望資料庫叢集能夠恢復的秒數上限。

  下列範例會將 `sample-cluster` 的目標恢復時段設定為一天 (86,400 秒)。

  對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-window 86400
```

  在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-window 86400
```

> **Note**
>
> 目前您可以只對透過恢復功能啟用時所建立的資料庫叢集啟用恢復。

RDS API

使用 CreateDBCluster Amazon RDS API 操作建立新的 Aurora MySQL 資料庫叢集時，當您指定大於零的 `BacktrackWindow` 值時，即會設定恢復。`BacktrackWindow` 值可指定 `DBClusterIdentifier` 值中指定之資料庫叢集的目標恢復時段。如需更多詳細資訊，請參閱 建立 Amazon Aurora 資料庫叢集。

您也可以使用下列 API 操作來指定 `BacktrackWindow` 值：

- ModifyDBCluster
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime

> **Note**
>
> 目前您可以只對透過恢復功能啟用時所建立的資料庫叢集啟用恢復。

執行 Aurora MySQL 資料庫叢集的恢復

您可以將資料庫叢集恢復至指定的恢復時間戳記。如果恢復時間戳記不早於最早可能的恢復時間，並且不是在未來，資料庫叢集即會恢復至該時間戳記。

否則會發生一般錯誤。同時，如果您嘗試恢復已啟用二進位日誌的資料庫叢集，除非您選擇強制發生恢復，否則會發生一般錯誤。強制發生恢復可能會干擾使用二進位日誌的其他操作。

> **Important**
>
> 恢復不會為其進行的變更產生 binlog 項目。如果您已為該資料庫叢集啟用二進位日誌，恢復可能不會與您的 binlog 實作相容。

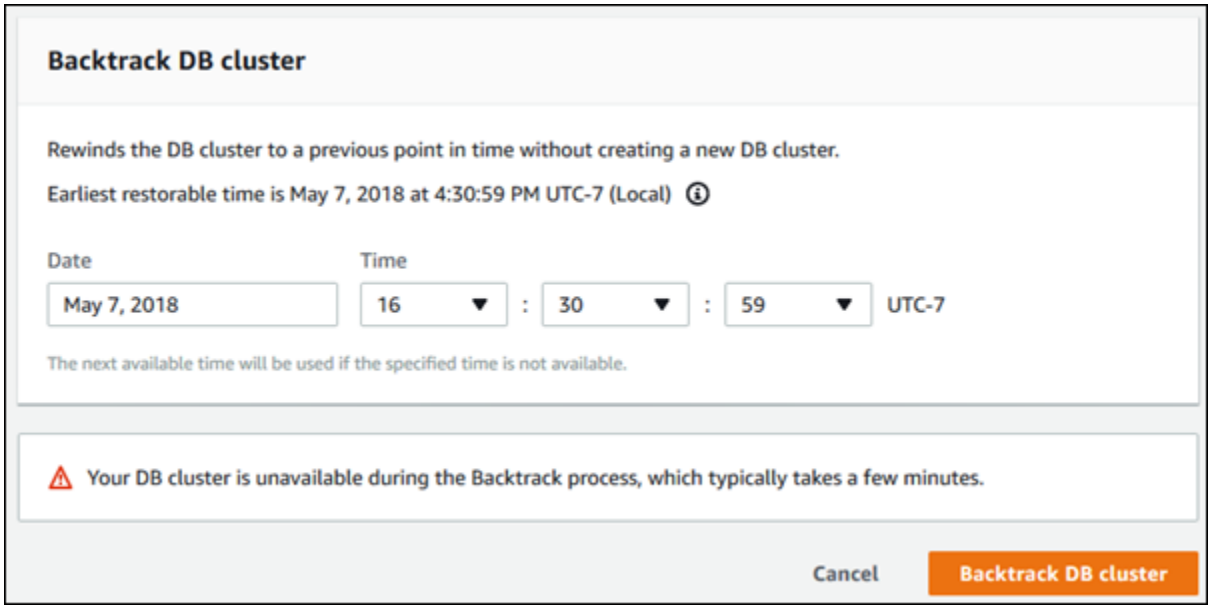
> **Note**
>
> 針對資料庫複製，您不可以恢復早於建立該複製的日期和時間的資料庫叢集。如需資料庫複製的詳細資訊，請參閱 複製 Amazon Aurora 資料庫叢集的一個磁碟區。

主控台

下列程序說明如何使用主控台來執行資料庫叢集的恢復操作。

使用主控台來執行恢復操作

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Instances (執行個體)。
3. 選擇您要恢復之資料庫叢集的主要執行個體。
4. 在 Actions (動作) 中，選擇 Backtrack DB cluster (恢復資料庫叢集)。
5. 在 Backtrack DB cluster (恢復資料庫叢集) 頁面上，輸入恢復資料庫叢集的目標恢復時間戳記。



6. 選擇 Backtrack DB cluster (恢復資料庫叢集)。

AWS CLI

下列程序說明如何使用 AWS CLI 恢復資料庫叢集。

使用 AWS CLI 恢復資料庫叢集

- 呼叫 backtrack-db-cluster AWS CLI 命令並提供下列值：
  - `--db-cluster-identifier` – 資料庫叢集的名稱。
  - `--backtrack-to` – 要恢復資料庫叢集的目標恢復時間戳記，指定格式為 ISO 8601。

下列範例會將資料庫叢集 `sample-cluster` 恢復至 2018 年 3 月 19 日上午 10:00。

對於 Linux、macOS 或 Unix：

```
aws rds backtrack-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-to 2018-03-19T10:00:00+00:00
```

在 Windows 中：

```
aws rds backtrack-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-to 2018-03-19T10:00:00+00:00
```

RDS API

若要使用 Amazon RDS API 恢復資料庫叢集，請使用 BacktrackDBCluster 操作。此操作會將 `DBClusterIdentifier` 值中指定的資料庫叢集恢復到指定的時間。

監控 Aurora MySQL 資料庫叢集的恢復

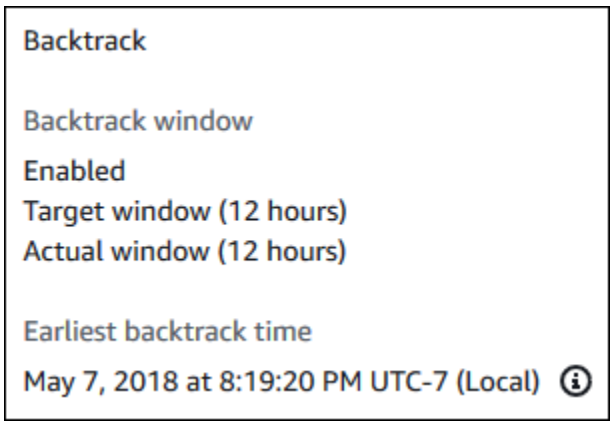
您可以檢視恢復資訊並監控資料庫叢集的恢復指標。

主控台

若要使用主控台檢視恢復資訊並監控恢復指標

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 選擇 Databases (資料庫)。
3. 選擇要開啟相關資訊之資料庫叢集的名稱。

   恢復資訊位在 Backtrack (恢復) 區段。



啟用恢復時，下列資訊可供使用：

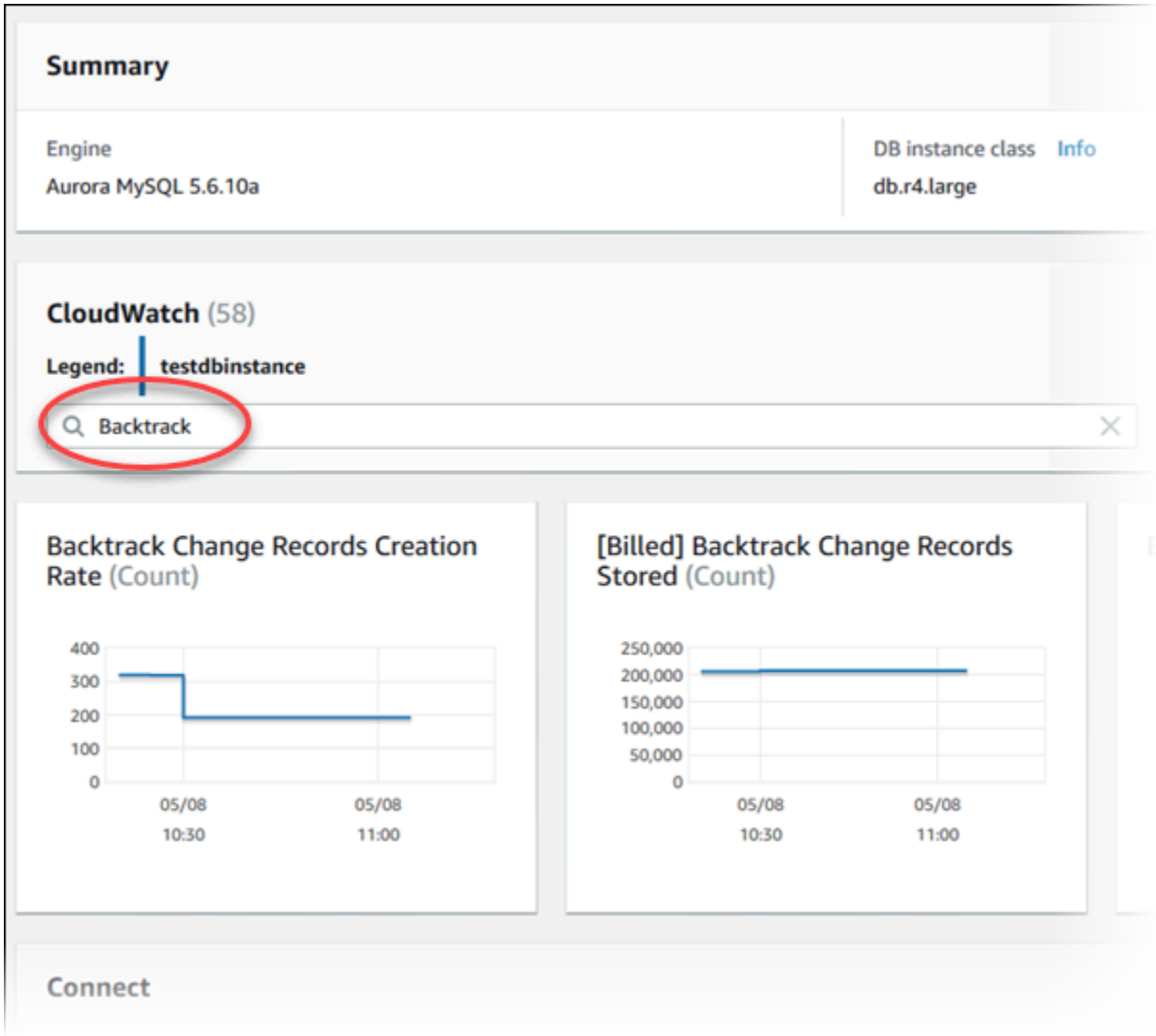
- Target window (目標時段) – 針對目標恢復時段指定的目前時間。目標為有足夠儲存體時您可以恢復的時間上限。
- Actual window (實際時段) – 您可以恢復的實際時間量，其可以小於目標恢復時段。實際恢復時段是根據您的工作負載和可用於保留恢復變更記錄的儲存體。
- Earliest backtrack time (最早恢復時間) – 資料庫叢集的最早可能恢復時間。您不可以將資料庫叢集恢復至較所顯示時間更早的時間。

4. 執行下列動作來檢視資料庫叢集的恢復指標：

   a. 在導覽窗格中，選擇 Instances (執行個體)。

   b. 選擇要顯示其資料庫叢集詳細資訊之主要執行個體的名稱。

   c. 在 CloudWatch 區段中，於 CloudWatch 方塊中輸入 **`Backtrack`** 來僅顯示恢復指標。



下列指標會顯示：

- Backtrack Change Records Creation Rate (Count) (恢復變更記錄建立速率 (計數)) – 此指標會顯示在五分鐘當中為您的資料庫叢集建立的恢復變更記錄的數量。您可以使用這個指標來估計目標恢復時段的恢復成本。
- [Billed] Backtrack Change Records Stored (Count) ([已計費] 恢復存放的變更記錄 (計數)) – 這個指標顯示您的資料庫叢集使用的恢復變更記錄的實際數量。
- Backtrack Window Actual (Minutes) (恢復時段實際 (分鐘)) – 這個指標顯示目標恢復時段與實際恢復時段之間的差異。例如，如果您的目標恢復時段為 2 小時 (120 分鐘)，並且這個指標顯示實際恢復時段為 100 分鐘，那麼實際恢復時段會小於目標。
- Backtrack Window Alert (Count) (恢復時段警示 (計數)) – 這個指標顯示實際恢復時段小於指定期間的目標恢復時段。

> **Note**
>
> 下列指標可能落後於目前的時間：
>
> - Backtrack Change Records Creation Rate (Count) (恢復變更記錄建立速率 (計數))
> - [Billed] Backtrack Change Records Stored (Count) ([已計費] 已存放的恢復變更記錄 (計數))

AWS CLI

下列程序說明如何使用 AWS CLI檢視資料庫叢集的恢復資訊。

使用 檢視資料庫叢集的恢復資訊 AWS CLI

- 呼叫 describe-db-clusters AWS CLI 命令並提供下列值：
  - `--db-cluster-identifier` – 資料庫叢集的名稱。

  下列範例列出 `sample-cluster` 的恢復資訊。

  對於 Linux、macOS 或 Unix：

```
aws rds describe-db-clusters \
    --db-cluster-identifier sample-cluster
```

  在 Windows 中：

```
aws rds describe-db-clusters ^
    --db-cluster-identifier sample-cluster
```

RDS API

若要使用 Amazon RDS API 檢視資料庫叢集的恢復資訊，請使用 DescribeDBClusters 操作。此操作會傳回 `DBClusterIdentifier` 值中指定之資料庫叢集的恢復資訊。

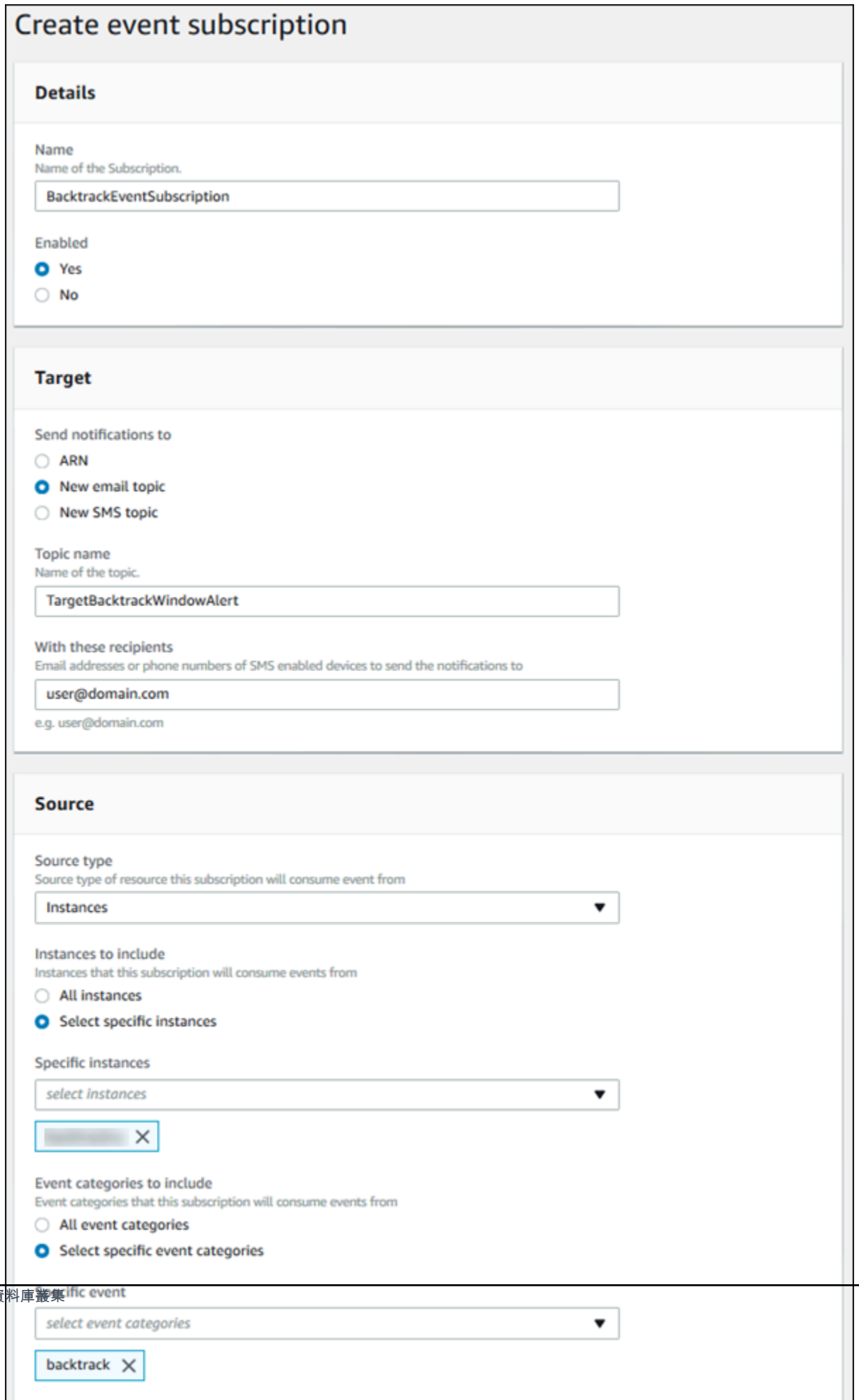
使用主控台訂閱恢復事件

下列程序說明如何使用主控台訂閱恢復事件。在您的實際恢復時段小於您的目標恢復時段時，該事件會傳送電子郵件或文字通知給您。

使用主控台檢視恢復資訊

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 選擇 Event subscriptions (事件訂閱)。
3. 選擇 Create event subscription (建立事件訂閱)。
4. 在 Name (名稱) 方塊中，輸入事件訂閱的名稱，並確保對 Enabled (已啟用) 選取 Yes (是)。
5. 在 Target (目標) 區段中，選擇 New email topic (新電子郵件主題)。
6. 針對 Topic name (主題名稱)，輸入主題的名稱，並針對 With these recipients (含有以下收件人)，輸入用來接收通知的電子郵件地址或電話號碼。
7. 在 Source (來源) 區段中，針對 Source type (來源類型) 選擇 Instances (執行個體)。
8. 針對 Instances to include (要併入的執行個體)，選擇 Select specific instances (選取特定執行個體)，並選擇您的資料庫執行個體。
9. 針對 Event categories to include (要併入的事件類別)，選擇 Select specific event categories (選取特定事件類別)，並選擇 backtrack (恢復)。

   您的頁面應該看起來類似下列頁面。


Create event subscription
Details
Name
Name of the Subscription.
BacktrackEventSubscription
Enabled
Yes
No
Target
Send notifications to
ARN
New email topic
New SMS topic
Topic name
Name of the topic.
TargetBacktrackWindowAlert
With these recipients
Email addresses or phone numbers of SMS enabled devices to send the notifications to
user@domain.com
e.g. user@domain.com
Source
Source type
Source type of resource this subscription will consume event from
Instances
Instances to include
Instances that this subscription will consume events from
All instances
Select specific instances
Specific instances
select instances
Event categories to include
Event categories that this subscription will consume events from
All event categories
Select specific event categories
select event categories
backtrack

10. 選擇建立。

擷取現有恢復

您可以擷取關於資料庫叢集現有恢復的資訊。此資訊包括恢復的唯一識別符、恢復的開始和結束日期和時間、要求恢復的日期和時間，以及恢復的目前狀態。

> **Note**
>
> 目前，您無法使用主控台擷取現有恢復。

AWS CLI

下列程序說明如何使用 AWS CLI擷取資料庫叢集的現有恢復。

使用 擷取現有的恢復 AWS CLI

- 呼叫 describe-db-cluster-backtracks AWS CLI 命令並提供下列值：
  - `--db-cluster-identifier` – 資料庫叢集的名稱。

  下列範例會擷取 `sample-cluster` 的現有恢復。

  對於 Linux、macOS 或 Unix：

```
aws rds describe-db-cluster-backtracks \
    --db-cluster-identifier sample-cluster
```

  在 Windows 中：

```
aws rds describe-db-cluster-backtracks ^
    --db-cluster-identifier sample-cluster
```

RDS API

若要使用 Amazon RDS API 擷取資料庫叢集恢復的相關資訊，請使用 DescribeDBClusterBacktracks 操作。此操作會傳回 `DBClusterIdentifier` 值中指定的資料庫叢集恢復的相關資訊。

停用 Aurora MySQL 資料庫叢集的恢復

您可以停用資料庫叢集的恢復功能。

主控台

您可以使用主控台停用資料庫叢集的恢復。完全關閉叢集的恢復功能後，無法再次為該叢集啟用恢復。

使用主控台停用資料庫叢集的恢復功能

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 選擇 Databases (資料庫)。
3. 選擇您要修改的叢集，並選擇 Modify (修改)。
4. 在 Backtrack (恢復) 區段中，選擇 Disable Backtrack (停用恢復)。
5. 選擇 Continue (繼續)。
6. 對於 Scheduling of Modifications (修改排程)，選擇以下其中一項：
   - Apply during the next scheduled maintenance window (在下一個排定的維護時段套用) – 等候在下一個維護時段時套用修改。
   - Apply immediately (立即套用) – 盡快套用修改。
7. 選擇 Modify Cluster (修改叢集)。

AWS CLI

您可以使用 AWS CLI 停用資料庫叢集的恢復功能，方法是將目標恢復時段設定為 `0` (零)。完全關閉叢集的恢復功能後，無法再次為該叢集啟用恢復。

使用 AWS CLI 來修改資料庫叢集的目標恢復時段

- 呼叫 modify-db-cluster AWS CLI 命令並提供下列值：
  - `--db-cluster-identifier` – 資料庫叢集的名稱。

- `--backtrack-window` – 指定 `0` 以關閉恢復。

下列範例會停用 `sample-cluster` 的恢復功能，方法是將 `--backtrack-window` 設定為 `0`。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-window 0
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-window 0
```

RDS API

若要使用 Amazon RDS API 停用資料庫叢集的恢復功能，請使用 ModifyDBCluster 操作。將 `BacktrackWindow` 值設為 `0` (零)，並在 `DBClusterIdentifier` 值中指定資料庫叢集。完全關閉叢集的恢復功能後，無法再次為該叢集啟用恢復。

使用錯誤注入查詢測試 Amazon Aurora MySQL

您可以使用錯誤注入查詢來測試 Aurora MySQL 資料庫叢集的容錯能力。錯誤注入查詢作為 SQL 命令發出到 Amazon Aurora 執行個體。它們可讓您排程下列其中一個事件的模擬出現：

- 寫入器或讀取器資料庫執行個體當機
- Aurora 複本失敗
- 磁碟失敗
- 磁碟擁塞

當錯誤注入查詢指定當機時，它會強制 Aurora MySQL 資料庫執行個體發生當機。其他錯誤注入查詢會造成失敗事件的模擬，但不會造成事件發生。提交錯誤注入查詢時，您也可以指定讓失敗事件模擬發生的時間。

透過連線至 Aurora 複本的端點，您可以將錯誤注入查詢提交給您的其中一個 Aurora 複本執行個體。如需詳細資訊，請參閱Amazon Aurora 端點連線。

執行錯誤注入查詢需要所有主要使用者權限。如需詳細資訊，請參閱主要使用者帳戶權限。

測試執行個體當機

您可以使用 `ALTER SYSTEM CRASH` 錯誤注入查詢來強制讓 Amazon Aurora 執行個體當機。

針對此錯誤注入查詢，將不會發生容錯移轉。如果您要測試容錯移轉，則可以在 RDS 主控台為您的資料庫叢集選擇 Failover (容錯移轉) 執行個體動作，或是使用 failover-db-cluster AWS CLI 命令或 FailoverDBCluster RDS API 操作。

語法

```
ALTER SYSTEM CRASH [ INSTANCE | DISPATCHER | NODE ];
```

選項

此錯誤注入查詢需要下列其中一個當機類型：

- **INSTANCE**—模擬 Amazon Aurora 執行個體之 MySQL 相容資料庫的當機。
- **DISPATCHER**—模擬 Aurora 資料庫叢集寫入器執行個體上發送器的當機。發送器會將更新寫入 Amazon Aurora 資料庫叢集的叢集磁碟區。
- **NODE**—模擬 MySQL 相容資料庫和 Amazon Aurora 執行個體發送器的當機。針對此錯誤注入模擬，也會刪除快取。

預設的當機類型為 `INSTANCE`。

測試 Aurora 複本失敗

您可以使用 `ALTER SYSTEM SIMULATE READ REPLICA FAILURE` 錯誤注入查詢來模擬 Aurora 複本的失敗。

Aurora 複本失敗將會對指定的時間間隔，封鎖對 Aurora 複本或資料庫叢集中所有 Aurora 複本來自寫入器執行個體的所有請求。當時間間隔完成，受影響的 Aurora 複本將與寫入器執行個體自動同步。

語法

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT READ REPLICA FAILURE
```

```
    [ TO ALL | TO "replica name" ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
SECOND };
```

選項

此錯誤注入查詢會使用下列參數：

- **percentage_of_failure**—在失敗事件期間請求封鎖的百分比。此值可以是介於 0 與 100 之間的雙位數數字。如果指定 0，則不會封鎖任何請求。如果指定 100，則會封鎖所有請求。
- 失敗類型 — 要模擬的失敗類型。指定 TO ALL 以模擬資料庫叢集中所有 Aurora 複本的失敗。指定 TO 和 Aurora 複本的名稱以模擬單一 Aurora 複本的失敗。預設的失敗類型為 TO ALL。
- **quantity** — 模擬 Aurora 複本失敗的時間量。間隔為數量接著時間單位。模擬將發生該指定單位的時間。例如，20 MINUTE 將造成模擬執行 20 分鐘。

> **Note**
>
> 指定 Aurora 複本失敗事件的時間間隔時請注意。如果指定的時間間隔太長，而您的寫入器執行個體在失敗事件期間寫入大量資料，則 Aurora 資料庫叢集可能會假設 Aurora 複本已當機並加以取代。

測試磁碟失敗

您可以使用 ALTER SYSTEM SIMULATE DISK FAILURE 錯誤注入查詢來模擬 Aurora 資料庫叢集的磁碟失敗。

在磁碟失敗模擬期間，Aurora 資料庫叢集會隨機將磁碟區段標示為發生錯誤。在模擬期間將封鎖對那些區段的請求。

語法

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK FAILURE
    [ IN DISK index | NODE index ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
SECOND };
```

選項

此錯誤注入查詢會使用下列參數：

- **percentage_of_failure**—在失敗事件期間標示為發生錯誤的磁碟百分比。此值可以是介於 0 與 100 之間的雙位數數字。如果指定 0，則不會將任何磁碟標示為發生錯誤。如果指定 100，則會將整個磁碟標示為發生錯誤。
- **DISK index**—模擬失敗事件的特定資料邏輯區塊。如果超出可用的資料邏輯區塊的範圍，您會收到錯誤，告訴您可以指定的最大索引值。如需更多詳細資訊，請參閱 顯示 Aurora MySQL 資料庫叢集的磁碟區狀態。
- **NODE index**—模擬失敗事件的特定儲存節點。如果超出儲存節點的可用範圍，您會收到錯誤，告訴您可以指定的最大索引值。如需更多詳細資訊，請參閱 顯示 Aurora MySQL 資料庫叢集的磁碟區狀態。
- **quantity**—模擬磁碟失敗的時間。間隔為數量接著時間單位。模擬將發生該指定單位的時間。例如，`20 MINUTE` 將造成模擬執行 20 分鐘。

測試磁碟壅塞

您可以使用 `ALTER SYSTEM SIMULATE DISK CONGESTION` 錯誤注入查詢來模擬 Aurora 資料庫叢集的磁碟失敗。

在磁碟擁塞模擬期間，Aurora 資料庫叢集會隨機將磁碟區段標示為擁塞。對那些區段的請求延遲將介於模擬期間的指定最小和最大延遲時間之間。

語法

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK CONGESTION
    BETWEEN minimum AND maximum MILLISECONDS
    [ IN DISK index | NODE index ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

選項

此錯誤注入查詢會使用下列參數：

- **percentage_of_failure**—在失敗事件期間標示為擁塞的磁碟百分比。此值可以是介於 0 與 100 之間的雙位數數字。如果指定 0，則不會將任何磁碟標示為擁塞。如果指定 100，則會將整個磁碟標示為擁塞。
- **DISK index** 或 **NODE index** — 對其模擬失敗事件的特定磁碟或節點。如果超出磁碟或節點索引的範圍，您會收到錯誤，告訴您可以指定的最大索引值。

- **minimum** 和 **maximum** — 最小和最大數量壅塞延遲 (以毫秒為單位)。標示為擁塞的磁碟區段將會於模擬期間在最小和最大數量的範圍內延遲隨機的時間 (毫秒)。
- **quantity**—模擬磁碟擁塞的時間。間隔為數量接著時間單位。模擬將發生達該指定的時間單位數量。例如，`20 MINUTE` 將造成模擬執行 20 分鐘。

使用快速 DDL 更改 Amazon Aurora 中的資料表

Amazon Aurora 包括就地近乎執行 `ALTER TABLE` 作業的最佳化。此操作不需要複製資料表即可完成，且對其他 DML 陳述式沒有實質影響。由於此操作不會耗用複製資料表所需的暫存儲存體，即使是小型執行個體類別上的大型資料表，DDL 陳述式依然很實用。

Aurora MySQL 第 3 版與稱為即時 DLL 的社群 MySQL 8.0 功能相容。Aurora MySQL 第 2 版使用稱為快速 DDL 的不同實作。

主題

- 即時 DDL (Aurora MySQL 第 3 版)
- 快速 DDL (Aurora MySQL 第 2 版)

即時 DDL (Aurora MySQL 第 3 版)

由 Aurora MySQL 第 3 版執行以提高一些 DDL 作業效率的最佳化稱為即時 DDL。

Aurora MySQL 第 3 版與來自社群 MySQL 8.0 的即時 DLL 相容。您可以使用子句 `ALGORITHM=INSTANT` 搭配 `ALTER TABLE` 陳述式，執行即時 DDL 作業。如需有關即時 DDL 的語法和用法詳細資料，請參閱 MySQL 說明文件中的 ALTER TABLE 和線上 DDL 操作。

下列範例會示範即時 DDL 功能。`ALTER TABLE` 陳述式會新增資料欄，並變更預設資料欄值。這些範例同時包括一般和虛擬資料欄，以及一般和分割資料表。在每個步驟中，您可以發出 `SHOW CREATE TABLE` 和 `DESCRIBE` 陳述式來查看結果。

```
mysql> CREATE TABLE t1 (a INT, b INT, KEY(b)) PARTITION BY KEY(b) PARTITIONS 6;
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE t1 RENAME TO t2, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ALTER COLUMN b SET DEFAULT 100, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> ALTER TABLE t2 ALTER COLUMN b DROP DEFAULT, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ADD COLUMN c ENUM('a', 'b', 'c'), ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 MODIFY COLUMN c ENUM('a', 'b', 'c', 'd', 'e'), ALGORITHM =
 INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ADD COLUMN (d INT GENERATED ALWAYS AS (a + 1) VIRTUAL), ALGORITHM
 = INSTANT;
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE t2 ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t3 (a INT, b INT) PARTITION BY LIST(a)(
    ->   PARTITION mypart1 VALUES IN (1,3,5),
    ->   PARTITION MyPart2 VALUES IN (2,4,6)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE t3 ALTER COLUMN a SET DEFAULT 20, ALTER COLUMN b SET DEFAULT 200,
 ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t4 (a INT, b INT) PARTITION BY RANGE(a)
    ->   (PARTITION p0 VALUES LESS THAN(100), PARTITION p1 VALUES LESS THAN(1000),
    ->   PARTITION p2 VALUES LESS THAN MAXVALUE);
Query OK, 0 rows affected (0.05 sec)

mysql> ALTER TABLE t4 ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

/* Sub-partitioning example */
mysql> CREATE TABLE ts (id INT, purchased DATE, a INT, b INT)
    ->   PARTITION BY RANGE( YEAR(purchased) )
    ->     SUBPARTITION BY HASH( TO_DAYS(purchased) )
    ->     SUBPARTITIONS 2 (
    ->       PARTITION p0 VALUES LESS THAN (1990),
    ->       PARTITION p1 VALUES LESS THAN (2000),
```

```
    ->       PARTITION p2 VALUES LESS THAN MAXVALUE
    ->    );
Query OK, 0 rows affected (0.10 sec)

mysql> ALTER TABLE ts ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)
```

快速 DDL (Aurora MySQL 第 2 版)

Aurora MySQL 中的快速 DDL 是一種最佳化，旨在透過減少停機時間和資源用量來改善特定結構描述變更的效能，例如新增或捨棄資料欄。相較於傳統 DDL 方法，它可讓這些操作更有效率地完成。

> **Important**
>
> 目前，您必須啟用 Aurora 實驗室模式，才能使用快速 DDL。如需啟用實驗室模式的詳細資訊，請參閱 Amazon Aurora MySQL 實驗室模式。
> 快速 DDL 最佳化最初是在 Aurora MySQL 第 2 版的實驗室模式中推出，以增強特定 DDL 操作的效率。在 Aurora MySQL 第 3 版中，實驗室模式已停止使用，而 Fast DDL 已由 MySQL 8.0 即時 DDL 功能取代。

在 MySQL 中，許多資料定義語言 (DDL) 操作有明顯的效能影響。

例如，假設您使用 `ALTER TABLE` 操作來將資料欄新增至資料表。根據為操作指定的演算法，此操作可能牽涉下列：

- 建立資料表的完整複本
- 建立暫存資料表來處理並行的資料處理語言 (DML) 操作
- 重新建置資料表的所有索引
- 套用並行 DML 變更時套用資料表鎖定
- 顯示並行的 DML 傳輸量

在具有大型資料表或高交易量的環境中，這種效能影響可能特別具有挑戰性。快速 DDL 可透過最佳化結構描述變更來協助緩解這些挑戰，從而實現更快速且較不耗資源的操作。

快速 DDL 限制

快速 DDL 目前具有下列限制：

- 快速 DDL 僅支援將不帶預設值、可為 Null 的資料欄新增至現有資料表的結尾。
- 快速 DDL 不適用於分割資料表。
- 快速 DDL 不適用使用 REDUNDANT 原始格式的 InnoDB 資料表。
- 快速 DDL 不適用於具有全文搜尋索引的資料表。
- 如果 DDL 操作最大可能的記錄大小太大，即不會使用快速 DDL。如果記錄大小大於頁面大小的一半便太大。記錄的最大大小是透過增加所有資料欄的最大大小來計算。根據 InnoDB 標準，針對變動大小的資料欄，extern 位元組不會併入在計算中。

快速 DDL 語法

```
ALTER TABLE tbl_name ADD COLUMN col_name column_definition
```

此陳述式使用下列選項：

- **tbl_name**—要修改之資料表的名稱。
- **col_name**—要新增之資料欄的名稱。
- **col_definition** — 要新增之資料欄的定義。

> **Note**
>
> 您必須指定不帶預設值、可為 Null 的資料欄定義。否則，將不會使用快速 DDL。

快速 DDL 範例

下面範例示範了透過快速 DDL 操作的加速。第一個 SQL 範例會在大型資料表上執行 ALTER TABLE 陳述式，而不使用快速 DDL。此操作需要大量的時間。CLI 範例顯示如何為叢集啟用快速 DDL。然後另一個 SQL 範例在相同的資料表上執行相同的 ALTER TABLE 陳述式。啟用快速 DDL 時，操作速度非常快。

此範例使用 TPC-H 基準測試中的 ORDERS 資料表，其中包含 1.5 億個資料列。此叢集有意使用相對較小型的執行個體類別，以示範當您無法使用快速 DDL 時，ALTER TABLE 陳述式可能需要多長時間。此範例會建立包含相同資料的原始資料表複製。因為未啟用實驗室模式，檢查 aurora_lab_mode 設定會確認叢集無法使用快速 DDL。然後 ALTER TABLE ADD COLUMN 陳述式需要大量時間在資料表結尾新增資料行。

```
mysql> create table orders_regular_ddl like orders;
```

```
Query OK, 0 rows affected (0.06 sec)

mysql> insert into orders_regular_ddl select * from orders;
Query OK, 150000000 rows affected (1 hour 1 min 25.46 sec)

mysql> select @@aurora_lab_mode;
+------------------+
| @@aurora_lab_mode |
+------------------+
|                0 |
+------------------+

mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (40 min 31.41 sec)

mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_coverletter varchar(512);
Query OK, 0 rows affected (40 min 44.45 sec)
```

此範例會對大型資料表執行與前一個範例相同的操作。然而，您不能只是啟用互動式 SQL 工作階段內的實驗室模式。必須在自訂參數群組中啟用該設定。這樣做需要切換離開 mysql 工作階段，執行一些 AWS CLI 命令，或者使用 AWS 管理主控台。

```
mysql> create table orders_fast_ddl like orders;
Query OK, 0 rows affected (0.02 sec)

mysql> insert into orders_fast_ddl select * from orders;
Query OK, 150000000 rows affected (58 min 3.25 sec)

mysql> set aurora_lab_mode=1;
ERROR 1238 (HY000): Variable 'aurora_lab_mode' is a read only variable
```

若要為叢集啟用實驗室模式，需要使用參數群組執行一些操作。此 AWS CLI 範例使用叢集參數群組，以確保叢集中的所有資料庫執行個體使用相同的實驗室模式設定值。

```
$ aws rds create-db-cluster-parameter-group \
  --db-parameter-group-family aurora5.7 \
    --db-cluster-parameter-group-name lab-mode-enabled-57 --description 'TBD'
$ aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --query '*[*].[ParameterName,ParameterValue]' \
      --output text | grep aurora_lab_mode
aurora_lab_mode 0
```

```
$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --parameters ParameterName=aurora_lab_mode,ParameterValue=1,ApplyMethod=pending-
reboot
{
    "DBClusterParameterGroupName": "lab-mode-enabled-57"
}

# Assign the custom parameter group to the cluster that's going to use Fast DDL.
$ aws rds modify-db-cluster --db-cluster-identifier tpch100g \
  --db-cluster-parameter-group-name lab-mode-enabled-57
{
  "DBClusterIdentifier": "tpch100g",
  "DBClusterParameterGroup": "lab-mode-enabled-57",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.2",
  "Status": "available"
}

# Reboot the primary instance for the cluster tpch100g:
$ aws rds reboot-db-instance --db-instance-identifier instance-2020-12-22-5208
{
  "DBInstanceIdentifier": "instance-2020-12-22-5208",
  "DBInstanceStatus": "rebooting"
}

$ aws rds describe-db-clusters --db-cluster-identifier tpch100g \
  --query '*[].[DBClusterParameterGroup]' --output text
lab-mode-enabled-57

$ aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
      --output text | grep aurora_lab_mode
aurora_lab_mode 1
```

下列範例顯示參數群組變更生效後的其餘步驟。它會測試 aurora_lab_mode 設定，以確定叢集可以使用快速 DDL。然後會執行 ALTER TABLE 陳述式，將資料欄新增至另一個大型資料表的末尾。此時，陳述式會很快完成。

```
mysql> select @@aurora_lab_mode;
+------------------+
| @@aurora_lab_mode |
```

```
+-------------------+
|                 1 |
+-------------------+

mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (1.51 sec)

mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_coverletter varchar(512);
Query OK, 0 rows affected (0.40 sec)
```

顯示 Aurora MySQL 資料庫叢集的磁碟區狀態

在 Amazon Aurora 中，資料庫叢集磁碟區包含邏輯區塊的集合。每個項目都代表 10 GB 的配置儲存體。這些區塊稱為保護群組。

每個保護群組中的資料會在六個實體儲存體裝置 (稱為儲存節點) 間複寫。這些儲存節點會在資料庫叢集所在 AWS 區域中的三個可用區域 (AZ) 中配置。每個儲存節點依次包含資料庫叢集磁碟區的一或多個資料邏輯區塊。如需保護群組和儲存節點的詳細資訊，請參閱 AWS 資料庫部落格上的介紹 Aurora Storage Engine。

您可以模擬整個儲存節點或儲存節點內單一資料邏輯區塊的失敗。若要這樣做，您可以使用 ALTER SYSTEM SIMULATE DISK FAILURE 錯誤注入陳述式。針對陳述式，您可以指定特定資料邏輯區塊或儲存節點的索引值。不過，如果您指定的索引值大於資料邏輯區塊或資料庫叢集磁碟區使用的儲存節點的數量，陳述式會傳回錯誤。如需錯誤注入查詢的詳細資訊，請參閱 使用錯誤注入查詢測試 Amazon Aurora MySQL。

您可以使用 SHOW VOLUME STATUS 陳述式來避免該錯誤。陳述式會傳回兩個伺服器狀態變數，Disks 和 Nodes。這些變數分別代表資料庫叢集磁碟區的資料邏輯區塊和儲存節點的數量。

語法

```
SHOW VOLUME STATUS
```

範例

下列範例說明一般的 SHOW VOLUME STATUS 結果。

```
mysql> SHOW VOLUME STATUS;
+---------------+-------+
```

```
| Variable_name | Value |
+---------------+-------+
| Disks         | 96    |
| Nodes         | 74    |
+---------------+-------+
```

調校 Aurora MySQL

等待事件和執行緒狀態是 Aurora MySQL 的重要調校工具。如果您可以了解工作階段為何等待資源及其運作情形，就更能夠減少瓶頸。您可以使用本節的資訊來尋找可能的原因和更正動作。

Amazon DevOps Guru for RDS 可以主動判斷您的 Aurora MySQL 資料庫是否出現可能導致嚴重後果的問題。Amazon DevOps Guru for RDS 會在主動洞察中發佈修正動作的說明和建議。本區段包含常見問題的洞察。

> **Important**
>
> 本節中的等待事件和執行緒狀態是 Aurora MySQL 特有的。本節資訊僅適用於調校 Amazon Aurora，而不適用於 Amazon RDS for MySQL。
> 本節部分等待事件在這些資料庫引擎的開放原始碼版本中沒有對等的事件。其他等待事件與開放原始碼引擎中的事件同名，但行為不同。例如，Amazon Aurora 儲存與開放原始碼儲存的運作方式不同，因此與儲存相關的等待事件表示不同的資源條件。

主題

- Aurora MySQL 調校的基本概念
- 使用等待事件調校 Aurora MySQL
- 使用執行緒狀態調校 Aurora MySQL
- 使用 Amazon DevOps Guru 主動洞察，調校 Aurora MySQL

Aurora MySQL 調校的基本概念

調校 Aurora MySQL 資料庫之前，請務必先了解什麼是等待事件和執行緒狀態及其發生原因。使用 InnoDB 儲存引擎時，亦請檢閱 Aurora MySQL 的基本記憶體和磁碟架構。如需實用的架構圖，請參閱 MySQL 參考手冊。

主題

- Aurora MySQL 等待事件
- Aurora MySQL 執行緒狀態
- Aurora MySQL 記憶體
- Aurora MySQL 處理

Aurora MySQL 等待事件

「等待事件」表示工作階段正在等待的資源。例如，等待事件 `io/socket/sql/client_connection` 表示執行緒正在處理新連線。工作階段等待的典型資源包括：

- 透過單一執行緒存取緩衝區，例如，當工作階段嘗試修改緩衝區時
- 另一個工作階段目前鎖定的資料列
- 資料檔讀取
- 日誌檔寫入

例如，為了滿足查詢，工作階段可能執行完整的資料表掃描。如果資料不在記憶體中，工作階段會等待磁碟輸入/輸出完成。將緩衝區讀入記憶體後，工作階段可能需要等待，因為其他工作階段正在存取這些緩衝區。資料庫使用預先定義的等待事件來記錄等待。這些事件分組為多個類別。

等待事件本身不表示有效能問題。例如，如果請求的資料不在記憶體中，則需要從磁碟讀取資料。如果一個工作階段鎖定資料列來更新，則另一個工作階段要等待此資料列解除鎖定才能更新。遞交需要等待寫入日誌檔完成。等待是資料庫正常運作所不可或缺。

大量等待事件通常表示有效能問題。在這種情況下，您可以使用等待事件資料來判斷工作階段將時間花在何處。例如，如果報告通常執行幾分鐘，但現在執行數小時，您可以識別佔總等待時間最多的等待事件。如果您可以查出最常等待事件的原因，通常就能做些改變來改善效能。例如，如果工作階段等待的資料列被另一個工作階段鎖定，您可以結束該鎖定工作階段。

Aurora MySQL 執行緒狀態

「一般執行緒狀態」是與一般查詢處理相關聯的 `State` 值。例如，執行緒狀態 `sending data` 表示執行緒正在讀取和篩選查詢的資料列，以決定正確的結果集。

您可以使用執行緒狀態，以類似於使用等待事件的方式調校 Aurora MySQL。例如，頻繁發生 `sending data` 通常表示查詢未使用索引。如需執行緒狀態的詳細資訊，請參閱《MySQL 參考手冊》中的一般執行緒狀態。

使用績效詳情時，下列其中一個條件為 true：

- 效能結構描述已開啟 – Aurora MySQL 顯示等待事件，而不是執行緒狀態。
- 效能結構描述未開啟 – Aurora MySQL 顯示執行緒狀態。

建議您設定效能結構描述進行自動管理。效能結構描述提供其他洞察和更好的工具，以調查潛在的效能問題。如需更多詳細資訊，請參閱 Aurora MySQL 上 Performance Insights 的效能結構描述概觀。

Aurora MySQL 記憶體

在 Aurora MySQL 中，最重要的記憶體區域是緩衝集區和日誌緩衝區。

主題

- 緩衝集區

緩衝集區

「緩衝集區」是 Aurora MySQL 快取資料表和檢索資料的共用記憶體區域。查詢可以直接從記憶體存取經常使用的資料，而無需從磁碟讀取。

緩衝集區會建構為頁面的鏈結清單。一個「頁面」可以保留多個資料列。Aurora MySQL 會使用最近最少使用 (LRU) 演算法，使頁面過時而移出集區。

如需詳細資訊，請參閱《MySQL 參考手冊》中的緩衝集區。

Aurora MySQL 處理

Aurora MySQL 使用的程序模型與 Aurora PostgreSQL 非常不同。

主題

- MySQL 伺服器 (mysqld)
- 執行緒
- 執行緒集區

MySQL 伺服器 (mysqld)

MySQL 伺服器是名為 mysqld 的單一作業系統程序。MySQL 伺服器不會產生額外的程序。因此，Aurora MySQL 資料庫會使用 mysqld 來執行其大部分工作。

當 MySQL 伺服器啟動時，它會接聽來自 MySQL 用戶端的網路連線。當用戶端連接到資料庫時，mysqld 會開啟一個執行緒。

執行緒

連線管理員執行緒會將每個用戶端連線與專用執行緒建立關聯。此執行緒會管理身分驗證、執行陳述式，並將結果傳回至用戶端。連線管理員會在必要時建立新的執行緒。

「執行緒快取」是一組可用的執行緒。當連線結束時，如果快取未滿，MySQL 會將執行緒傳回至執行緒快取。`thread_cache_size` 系統變數會決定執行緒快取大小。

執行緒集區

「執行緒集區」包含許多執行緒群組。每個群組都會管理一組用戶端連線。當用戶端連線到資料庫時，執行緒集區會以循環方式將連線指派給執行緒群組。執行緒集區會將連線和執行緒隔開。連線與執行從這些連線收到之陳述式的執行緒之間沒有固定關係。

使用等待事件調校 Aurora MySQL

下表彙總了最常表示效能問題的 Aurora MySQL 等待事件。下列等待事件是 Aurora MySQL 等待事件中的清單子集。

等待事件	說明
cpu	此事件表示執行緒活躍於 CPU 中或正在等待 CPU。
io/aurora_redo_log_flush	當工作階段將持久性資料寫入至 Aurora 儲存體時，此事件便會發生。
io/aurora_respond_to_client	當執行緒正在等待將結果集傳回至用戶端時，此事件便會發生。
io/redo_log_flush	當工作階段將持久性資料寫入至 Aurora 儲存體時，此事件便會發生。
io/socket/sql/client_connection	當執行緒正在處理新連線時，此事件便會發生。
io/table/sql/handler	當工作已委派給儲存引擎時，此事件便會發生。
synch/cond/innodb/row_lock_wait	當一個工作階段已鎖定資料列進行更新，而另一個工作階段嘗試更新同一資料列時，此事件便會發生。

等待事件	說明
synch/cond/innodb/row_lock_wait_cond	當一個工作階段已鎖定資料列進行更新，而另一個工作階段嘗試更新同一資料列時，此事件便會發生。
synch/cond/sql/MDL_context::COND_wait_status	有執行緒正在等待資料表中繼資料鎖定時，此事件便會發生。
synch/mutex/innodb/aurora_lock_thread_slot_futex	當一個工作階段已鎖定資料列進行更新，而另一個工作階段嘗試更新同一資料列時，此事件便會發生。
synch/mutex/innodb/buf_pool_mutex	當執行緒已對 InnoDB 緩衝集區取得鎖定來存取記憶體中的頁面時，此事件便會發生。
synch/mutex/innodb/fil_mutex	當工作階段正在等待存取資料表空間記憶體快取時，此事件便會發生。
synch/mutex/innodb/fil_mutex	由於大量交易而有高資料庫活動時，此事件便會發生。
synch/sxlock/innodb/hash_table_locks	當緩衝集區中找不到的頁面必須從檔案讀取時，此事件便會發生。
synch/mutex/innodb/temp_pool_manager_mutex	當工作階段正在等待取得互斥以管理工作階段暫存資料表空間集區時，就會發生此事件。

cpu

cpu 等待事件表示執行緒活躍於 CPU 中或正在等待 CPU。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因
- 動作

支援的引擎版本

下列引擎版本支援這個等待事件資訊：

- Aurora MySQL 2 版和 3 版

Context

對於每個 vCPU，連線都可以在此 CPU 上執行工作。在某些情況下，準備好執行的作用中連線數量高於 vCPU 的數量。這種不平衡會導致連線等待 CPU 資源。如果作用中連線的數量持續地高於 vCPU 的數量，則您的執行個體會遭遇 CPU 爭用。爭用會導致 cpu 等待事件發生。

> **Note**
>
> CPU 的績效詳情指標為 DBLoadCPU。DBLoadCPU 的值與 CloudWatch 指標 CPUUtilization 的值可能不同。後者是從 HyperVisor 為資料庫執行個體收集的指標。

績效詳情作業系統指標提供 CPU 使用率的詳細資訊。例如，您可以顯示下列指標：

- os.cpuUtilization.nice.avg
- os.cpuUtilization.total.avg
- os.cpuUtilization.wait.avg
- os.cpuUtilization.idle.avg

績效詳情以 os.cpuUtilization.nice.avg 報告資料庫引擎的 CPU 使用率。

等待變多的可能原因

此事件比平時更常發生時，可能表示有效能問題，典型原因包括：

- 分析查詢
- 高度並行交易
- 長時間執行的交易
- 連線數量突然增加，稱為「登入風暴」
- 內容切換變多

動作

如果 cpu 等待事件在資料庫活動中佔多數，則不見得表示有效能問題。效能降低時才需要回應此事件。

根據 CPU 使用率增加的原因，請考慮下列策略：

- 增加主機的 CPU 容量。這種方法通常只是應急而已。
- 識別潛在最佳化的熱門查詢。
- 如適用，請將一些唯讀工作負載重新導向至讀取器節點

主題

- 識別造成問題的工作階段或查詢
- 分析並最佳化高 CPU 工作負載

識別造成問題的工作階段或查詢

若要尋找工作階段和查詢，請查看績效詳情中的 Top SQL (最高 SQL) 資料表，以取得 CPU 負載最高的 SQL 陳述句。如需詳細資訊，請參閱使用績效詳情儀表板來分析指標。

通常，一或兩個 SQL 陳述式會耗用大部分的 CPU 週期。將精力集中在這些陳述式上。假設您的資料庫執行個體有 2 個 vCPU，資料庫負載為 3.1 平均作用中工作階段 (AAS)，全部都處於 CPU 狀態。在此情況下，您的執行個體受到 CPU 限制。請考慮下列策略：

- 升級至 vCPU 更多的大型執行個體類別。
- 調校您的查詢以降低 CPU 負載。

在此範例中，最高 SQL 查詢具有 1.5 AAS 的資料庫負載，全部都處於 CPU 狀態。另一個 SQL 陳述式在 CPU 狀態下有 0.1 的負載。在此範例中，如果您已停止負載最低的 SQL 陳述式，則不會大幅減少資料庫負載。不過，如果您將兩個高負載查詢最佳化為效率的兩倍，則可以消除 CPU 瓶頸。如果您將 1.5 AAS 的 CPU 負載減少 50%，則每個陳述式的 AAS 會降低至 0.75。CPU 上花費的總資料庫負載現在為 1.6 AAS。此值低於最大 vCPU 數線，即 2.0。

如需使用績效詳情進行疑難排解的實用概觀，請參閱部落格文章利用績效詳情分析 Amazon Aurora MySQL 工作負載。另請參閱 AWS 支援文章 如何疑難排解和解決 Amazon RDS for MySQL 執行個體上的高 CPU 使用率？。

分析並最佳化高 CPU 工作負載

在找出增加 CPU 使用率的查詢之後，您可以將其最佳化或結束連線。下列範例示範如何結束連線。

```
CALL mysql.rds_kill(processID);
```

如需詳細資訊，請參閱mysql.rds_kill。

如果您結束工作階段，動作可能會觸發長時間回復。

遵循最佳化查詢的指導方針

若要最佳化查詢，請考慮以下指導方針：

- 執行 EXPLAIN 陳述式。

  這個命令會顯示執行查詢所涉及的個別步驟。如需詳細資訊，請參閱 MySQL 文件中的利用 EXPLAIN 最佳化查詢。
- 執行 SHOW PROFILE 陳述式。

  使用此陳述式可檢閱設定檔詳細資訊，此詳細資訊可以指出在目前工作階段期間執行之陳述式的資源使用情形。如需詳細資訊，請參閱 MySQL 文件中的 SHOW PROFILE 陳述式。
- 執行 ANALYZE TABLE 陳述式。

  使用此陳述式，可重新整理高 CPU 耗用查詢所存取之資料表的索引統計資料。藉由分析敘述句，您可以協助最佳化工具選擇適當的執行計劃。如需詳細資訊，請參閱 MySQL 文件中的 ANALYZE TABLE 陳述式。

遵循改善 CPU 使用率的指導方針

若要改善資料庫執行個體中的 CPU 使用率，請遵循下列指導方針：

- 確保所有查詢都是使用適當的索引。
- 了解您是否可以使用 Aurora 平行查詢。您可以使用此技術，減少前端節點上的 CPU 使用量，方法是下推 WHERE 子句的函數處理、資料列篩選和資料欄投影。
- 了解每秒 SQL 執行次數是否符合預期的閾值。
- 了解索引維護或建立新索引是否佔用生產工作負載所需的 CPU 週期。將維護活動排定在尖峰活動時間之外。

- 了解您是否可以使用分割來協助減少查詢資料集。如需詳細資訊，請參閱部落格文章如何為合併的工作負載規劃並最佳化 Amazon Aurora 與 MySQL 相容性。

檢查連線風暴

如果 DBLoadCPU 指標不是非常高，但 CPUUtilization 指標很高，CPU 使用率高的原因是在資料庫引擎之外造成的。典型範例是連線風暴。

檢查下列條件是否為 true：

- 績效詳情 CPUUtilization 指標和 Amazon CloudWatch DatabaseConnections 指標都有所增加。
- CPU 中的執行緒數量大於 vCPU 的數量。

如果上述條件為 true，請考慮減少資料庫連線數量。例如，您可以使用連線集區，如 RDS Proxy。若要了解有效連線管理和擴展的最佳實務，請參閱白皮書用於連線管理的 Amazon Aurora MySQL DBA 手冊。

io/aurora_redo_log_flush

當工作階段將持久性資料寫入至 Amazon Aurora 儲存體時，io/aurora_redo_log_flush 事件便會發生。

主題

- 支援的引擎版本
- Context
- 等待時間增加的可能原因
- 動作

支援的引擎版本

下列引擎版本支援這個等待事件資訊：

- Aurora MySQL 第 2 版

Context

io/aurora_redo_log_flush 事件適用於 Aurora MySQL 中的寫入輸入/輸出 (輸入/輸出) 作業。

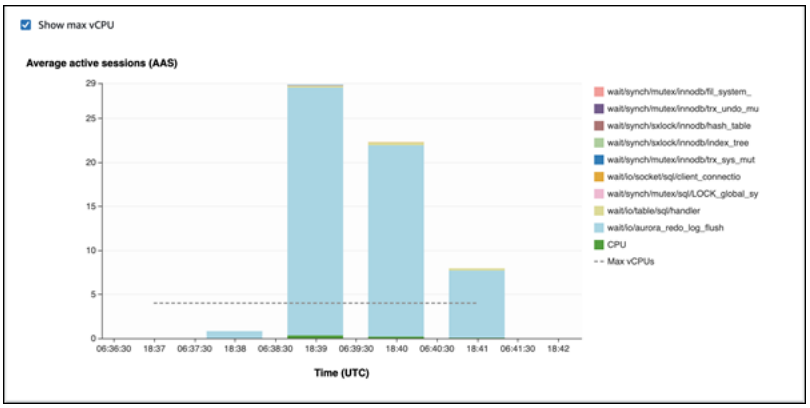
> **Note**
>
> 在 Aurora MySQL 第 3 版中，此等待事件名為 io/redo_log_flush。

等待時間增加的可能原因

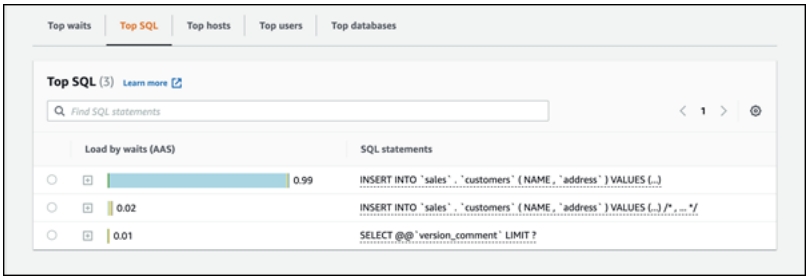
對於資料持久性，遞交需要持久寫入至穩定的儲存。如果資料庫執行太多的遞交，寫入輸入/輸出作業上會有等待事件，即 `io/aurora_redo_log_flush` 等待事件。

在下列範例中，會使用 db.r5.xlarge 資料庫執行個體類別，將 50,000 筆記錄插入至 Aurora MySQL 資料庫叢集：

- 在第一個範例中，每個工作階段逐列插入 10,000 筆記錄。根據預設，如果資料處理語言 (DML) 命令不在交易內，Aurora MySQL 會使用隱含遞交。系統會開啟自動遞交。這表示對於每個資料列插入都有一個遞交。績效詳情顯示，連線花費其大部分時間等待 `io/aurora_redo_log_flush` 等待事件。

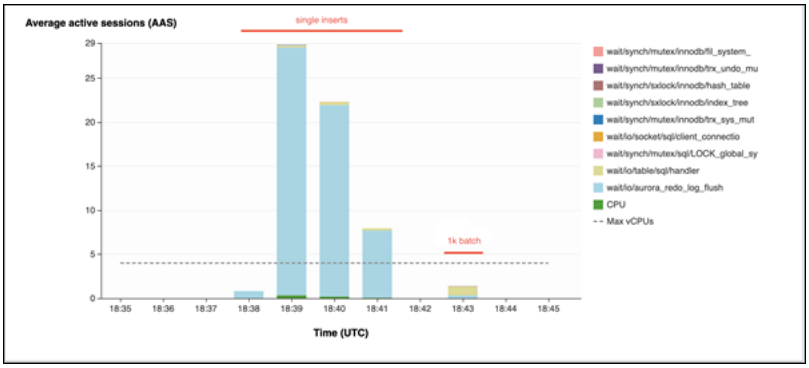


這是由使用的簡單插入陳述式引起的。



插入 50,000 筆記錄需要 3.5 分鐘。

- 在第二個範例中，插入是以 1,000 個批次進行，亦即每個連線執行 10 個遞交，而不是 10,000 個。績效詳情顯示，連線不會將其大部分時間花費在 `io/aurora_redo_log_flush` 等待事件上。



插入 50,000 筆記錄需要 4 秒。

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 識別有問題的工作階段和查詢
- 將您的寫入作業分組
- 關閉自動遞交
- 使用交易
- 使用批次

識別有問題的工作階段和查詢

如果您的資料庫執行個體遭遇瓶頸，您的第一項任務是尋找造成瓶頸的工作階段和查詢。如需實用的 AWS 資料庫部落格文章，請參閱使用績效詳情分析 Amazon Aurora MySQL 工作負載。

識別造成瓶頸的工作階段和查詢

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。
4. 在 Database load (資料庫負載) 中，選擇 Slice by wait (依等待建立配量)。
5. 在頁面底端，選擇 Top SQL (最高 SQL)。

   清單頂端的查詢對資料庫造成最高負載。

將您的寫入作業分組

下列範例會觸發 io/aurora_redo_log_flush 等待事件。(系統會開啟自動遞交)。

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

若要減少等待 io/aurora_redo_log_flush 等待事件所花費的時間，將您的寫入作業邏輯分組為單一遞交，以減少持續呼叫儲存。

關閉自動遞交

在進行不在交易內的大規模變更之前，請先關閉自動遞交，如下列範例所示。

```
SET SESSION AUTOCOMMIT=OFF;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
-- Other DML statements here
COMMIT;

SET SESSION AUTOCOMMIT=ON;
```

使用交易

您可以使用交易，如下列範例所示。

```
BEGIN
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;

-- Other DML statements here
END
```

使用批次

您也可以批次進行變更，如下列範例所示。不過，使用過大的批次可能會導致效能問題，尤其是在僅供讀取複本中或在執行時間點復原 (PITR) 時。

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES
('xxxx','xxxxx'),('xxxx','xxxxx'),...,('xxxx','xxxxx'),('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1 BETWEEN xx AND
 xxx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1<xx;
```

io/aurora_respond_to_client

當執行緒正在等待將結果集傳回至用戶端時，`io/aurora_respond_to_client` 事件便會發生。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因
- 動作

支援的引擎版本

下列引擎版本支援這個等待事件資訊：

- Aurora MySQL 第 2 版

Context

事件 `io/aurora_respond_to_client` 指出執行緒正在等待將結果集傳回至用戶端時。

查詢處理完成，且結果會傳回至應用程式用戶端。不過，由於資料庫叢集上沒有足夠的網路頻寬，因此執行緒正在等待傳回結果集。

等待變多的可能原因

`io/aurora_respond_to_client` 事件比平時更常出現時，可能表示有效能問題，典型原因包括：

資料庫執行個體類別不足以處理工作負載

資料庫叢集所使用的資料庫執行個體類別沒有必要的網路頻寬來有效處理工作負載。

大型結果集

傳回的結果集大小增加，因為查詢傳回大量的資料列。結果集越大，耗用的網路頻寬就越多。

用戶端的負載增加

用戶端可能面臨 CPU 壓力、記憶體壓力或網路飽和。用戶端的負載增加會延遲從 Aurora MySQL 資料庫叢集接收資料。

網路延遲較久

Aurora MySQL 資料庫叢集與用戶端之間的網路延遲可能變長。網路延遲越久會導致用戶端需要更多時間來接收資料。

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 識別造成事件的工作階段和查詢

- 擴展資料庫執行個體類別
- 檢查工作負載是否有非預期結果
- 使用讀取器執行個體分配工作負載
- 使用 SQL_BUFFER_RESULT 修飾詞

識別造成事件的工作階段和查詢

您可以使用績效詳情來顯示 `io/aurora_respond_to_client` 等待事件所封鎖的查詢。通常，具有中等至重大負載的資料庫會有等待事件。如果效能是最佳的，則等待事件可能是可以接受的。如果效能不是最佳的，則檢查資料庫在何處花費最多時間。查看造成最高負載的等待事件，並了解您是否可以最佳化資料庫和應用程式，以減少這些事件。

尋找負責高負載的 SQL 查詢

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。即會顯示該資料庫執行個體的績效詳情儀表板。
4. 在 Database load (資料庫負載) 圖表中，選擇 Slice by wait (依等待建立配量)。
5. 在頁面底端，選擇 Top SQL (最高 SQL)。

   此圖表會列出負責負載的 SQL 查詢。位於清單頂端者負最大責任。若要解決瓶頸，請專注於這些陳述式。

如需使用績效詳情進行故障診斷的實用概觀，請參閱 AWS 資料庫部落格文章使用績效詳情分析 Amazon Aurora MySQL 工作負載。

擴展資料庫執行個體類別

檢查與網路輸送量 (例如 `NetworkReceiveThroughput` 和 `NetworkTransmitThroughput`) 相關的 Amazon CloudWatch 指標值是否增加。如果達到資料庫執行個體類別的網路頻寬，您可以修改資料庫叢集來擴展資料庫叢集所使用的資料庫執行個體。具有較大網路頻寬的資料庫執行個體類別可更有效率地將資料傳回至用戶端

如需監控 Amazon CloudWatch 指標的相關資訊，請參閱在 Amazon RDS 主控台中檢視指標。如需資料庫執行個體類別的相關資訊，請參閱 Amazon Aurora 資料庫執行個體類別。如需修改資料庫叢集的詳細資訊，請參閱 修改 Amazon Aurora 資料庫叢集。

檢查工作負載是否有非預期結果

檢查資料庫叢集上的工作負載，並確定它不會產生非預期的結果。例如，可能有查詢傳回比預期還要多的資料列。在此情況下，您可以使用績效詳情計數器指標，例如 Innodb_rows_read。如需詳細資訊，請參閱Performance Insights 計數器指標。

使用讀取器執行個體分配工作負載

您可以使用 Aurora 複本分配唯讀工作負載。您可以新增更多的 Aurora 複本來水平擴展。這樣做可能會提高網路頻寬的調節限制。如需詳細資訊，請參閱Amazon Aurora 資料庫叢集。

使用 SQL_BUFFER_RESULT 修飾詞

您可以將 SQL_BUFFER_RESULT 修飾詞新增至 SELECT 陳述式，在將結果傳回至用戶端之前，強制將這些結果放入暫時資料表中。當 InnoDB 鎖定由於查詢處於 io/aurora_respond_to_client 等待狀態而未釋放時，此修飾詞可以協助解決效能問題。如需詳細資訊，請參閱 MySQL 文件中的 SELECT PROFILE 陳述式。

io/redo_log_flush

當工作階段將持久性資料寫入至 Amazon Aurora 儲存體時，io/redo_log_flush 事件便會發生。

主題

- 支援的引擎版本
- Context
- 等待時間增加的可能原因
- 動作

支援的引擎版本

下列引擎版本支援這個等待事件資訊：

- Aurora MySQL 第 3 版

Context

io/redo_log_flush 事件適用於 Aurora MySQL 中的寫入輸入/輸出 (輸入/輸出) 作業。

> **Note**
>
> 在 Aurora MySQL 第 2 版中，此等待事件名為 io/aurora_redo_log_flush。

等待時間增加的可能原因

對於資料持久性，遞交需要持久寫入至穩定的儲存。如果資料庫執行太多的遞交，寫入輸入/輸出作業上會有等待事件，即 `io/redo_log_flush` 等待事件。

如需此等待事件行為的範例，請參閱 io/aurora_redo_log_flush。

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 識別有問題的工作階段和查詢
- 將您的寫入作業分組
- 關閉自動遞交
- 使用交易
- 使用批次

識別有問題的工作階段和查詢

如果您的資料庫執行個體遭遇瓶頸，您的第一項任務是尋找造成瓶頸的工作階段和查詢。如需實用的 AWS 資料庫部落格文章，請參閱使用績效詳情分析 Amazon Aurora MySQL 工作負載。

識別造成瓶頸的工作階段和查詢

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。
4. 在 Database load (資料庫負載) 中，選擇 Slice by wait (依等待建立配量)。
5. 在頁面底端，選擇 Top SQL (最高 SQL)。

清單頂端的查詢對資料庫造成最高負載。

將您的寫入作業分組

下列範例會觸發 io/redo_log_flush 等待事件。(系統會開啟自動遞交)。

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

若要減少等待 io/redo_log_flush 等待事件所花費的時間，將您的寫入作業邏輯分組為單一遞交，以減少持續呼叫儲存。

關閉自動遞交

在進行不在交易內的大規模變更之前，請先關閉自動遞交，如下列範例所示。

```
SET SESSION AUTOCOMMIT=OFF;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
-- Other DML statements here
COMMIT;

SET SESSION AUTOCOMMIT=ON;
```

使用交易

您可以使用交易，如下列範例所示。

```
BEGIN
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;

-- Other DML statements here
END
```

使用批次

您也可以批次進行變更，如下列範例所示。不過，使用過大的批次可能會導致效能問題，尤其是在僅供讀取複本中或在執行時間點復原 (PITR) 時。

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES
('xxxx','xxxxx'),('xxxx','xxxxx'),...,('xxxx','xxxxx'),('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1 BETWEEN xx AND
 xxx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1<xx;
```

io/socket/sql/client_connection

當執行緒正在處理新連線時，`io/socket/sql/client_connection` 事件便會發生。

主題

- 支援的引擎版本
- Context

- 等待變多的可能原因
- 動作

支援的引擎版本

下列引擎版本支援這個等待事件資訊：

- Aurora MySQL 2 版和 3 版

Context

事件 `io/socket/sql/client_connection` 指出 mysqld 忙於建立執行緒來處理傳入的新用戶端連線。在此情況中，當連線等待要指派的執行緒時，服務新的用戶端連線請求的處理速度會減慢。如需詳細資訊，請參閱MySQL 伺服器 (mysqld)。

等待變多的可能原因

此事件比平時更常出現時，可能表示有效能問題，典型原因包括：

- 從應用程式到 Amazon RDS 執行個體的新使用者連線突然增加。
- 您的資料庫執行個體無法處理新連線，因為網路、CPU 或記憶體正在調節。

動作

如果 `io/socket/sql/client_connection` 主導資料庫活動，則不見得表示有效能問題。在不是閒置的資料庫中，等待事件始終位於頂端。只有在效能降低時才採取行動。根據等待事件的原因，我們會建議不同的動作。

主題

- 識別有問題的工作階段和查詢
- 遵循連線管理的最佳實務
- 如果資源正在調節，請擴展執行個體
- 檢查最高主機和最高使用者
- 查詢 performance_schema 資料表
- 檢查查詢的執行緒狀態
- 稽核請求和查詢

- 集中資料庫連線

識別有問題的工作階段和查詢

如果您的資料庫執行個體遭遇瓶頸，您的第一項任務是尋找造成瓶頸的工作階段和查詢。如需實用的部落格文章，請參閱利用績效詳情分析 Amazon Aurora MySQL 工作負載。

識別造成瓶頸的工作階段和查詢

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。
4. 在 Database load (資料庫負載) 中，選擇 Slice by wait (依等待建立配量)。
5. 在頁面底端，選擇 Top SQL (最高 SQL)。

   清單頂端的查詢對資料庫造成最高負載。

遵循連線管理的最佳實務

若要管理連線，請考慮下列策略：

- 使用連線集區

  您可以視需要逐漸增加連線數量。如需詳細資訊，請參閱白皮書 Amazon Aurora MySQL 資料庫管理員手冊。
- 使用讀取器節點重新分配唯讀流量。

  如需更多詳細資訊，請參閱 Aurora 複本及 Amazon Aurora 端點連線。

如果資源正在調節，請擴展執行個體

在下列資源中尋找調節的範例：

- CPU

  檢查 Amazon CloudWatch 指標是否有高 CPU 使用率。
- 網路

檢查 CloudWatch 指標 `network receive throughput` 和 `network transmit throughput` 的值是否增加。如果您的執行個體已達到執行個體類別的網路頻寬限制，請考慮將 RDS 執行個體擴展到更高的執行個體類別類型。如需詳細資訊，請參閱Amazon Aurora 資料庫執行個體類別。

- 可用記憶體

  檢查 CloudWatch 指標 `FreeableMemory` 是否下降。此外，請考慮開啟增強型監控。如需詳細資訊，請參閱使用增強型監控來監控作業系統指標。

檢查最高主機和最高使用者

使用績效詳情來檢查最高主機和最高使用者。如需詳細資訊，請參閱使用績效詳情儀表板來分析指標。

查詢 performance_schema 資料表

若要取得目前和總連線的準確計數，請查詢 `performance_schema` 資料表。使用此技術，您可以識別負責建立大量連線的來源使用者或主機。例如，查詢 `performance_schema` 資料表，如下所示。

```
SELECT * FROM performance_schema.accounts;
SELECT * FROM performance_schema.users;
SELECT * FROM performance_schema.hosts;
```

檢查查詢的執行緒狀態

如果效能問題持續發生，請檢查查詢的執行緒狀態。在 `mysql` 用戶端中，發出下列命令。

```
show processlist;
```

稽核請求和查詢

若要從使用者帳戶檢查請求和查詢的性質，請使用 Aurora MySQL 進階稽核。若要了解如何開啟稽核，請參閱使用進階稽核與 Amazon Aurora MySQL 資料庫叢集搭配。

集中資料庫連線

考慮使用 Amazon RDS Proxy 進行連線管理。透過使用 RDS Proxy，您可以允許應用程式集中和共用資料庫連線，以改善其擴展能力。RDS Proxy 會自動連線至待命資料庫執行個體，同時保留應用程式連線，使應用程式更具有資料庫故障彈性。如需詳細資訊，請參閱Amazon RDS Proxy for Aurora。

io/table/sql/handler

當工作已委派給儲存引擎時，io/table/sql/handler 事件便會發生。

主題

- 支援的引擎版本
- Context
- 等待時間增加的可能原因
- 動作

支援的引擎版本

下列引擎版本支援這個等待事件資訊：

- Aurora MySQL 2 版和 3 版

Context

事件 io/table 指出等待存取資料表。無論是在緩衝集區中快取資料，還是在磁碟上存取資料，都會發生此事件。io/table/sql/handler 事件表示工作負載活動增加。

「處理常式」是專門處理特定類型的資料或專注於某些特殊任務的常式。例如，事件處理常式會接收和摘錄來自作業系統或使用者界面的事件和訊號。記憶體處理常式會執行與記憶體相關的任務。文件輸入處理常式是接收檔案輸入，並根據內容對資料執行特殊任務的函數。

當實際等待是巢狀等待事件 (例如鎖定) 時，performance_schema.events_waits_current 這類的檢視經常顯示 io/table/sql/handler。當實際的等待不是 io/table/sql/handler 時，績效詳情會報告巢狀等待事件。當 Performance Insights 報告 io/table/sql/handler 時，它代表 I/O 請求的 InnoDB 處理，而不是隱藏的巢狀等待事件。如需詳細資訊，請參閱 MySQL 參考手冊中的效能結構描述原子和分子事件。

io/table/sql/handler 事件通常會與 I/O 等待 (例如 io/aurora_redo_log_flush) 一起出現在最久等待事件中。

等待時間增加的可能原因

在績效詳情中，io/table/sql/handler 事件中的突然峰值表示工作負載活動增加。活動增加意味著輸入/輸出增加。

績效詳情會篩選巢狀事件 ID，而且在基礎巢狀事件是鎖等待時不會報告 io/table/sql/handler 等待。例如，如果根本原因事件是 synch/mutex/innodb/aurora_lock_thread_slot_futex，績效詳情會在最高等待事件中顯示此等待，而不是 io/table/sql/handler。

在 performance_schema.events_waits_current 這類的檢視中，當實際等待是巢狀等待事件 (例如鎖定) 時，io/table/sql/handler 的等待通常就會出現。當實際等待與 io/table/sql/handler 不同時，績效詳情會查閱巢狀等待，並報告實際等待，而不是 io/table/sql/handler。當績效詳情報告 io/table/sql/handler 時，真正的等待是 io/table/sql/handler，而不是隱藏的巢狀等待事件。如需詳細資訊，請參閱 MySQL 5.7 參考手冊中的效能結構描述原子和分子事件。

動作

如果此等待事件主導資料庫活動，則不見得表示有效能問題。當資料庫作用中時，等待事件一律在頂端。只有在效能降低時才需要採取行動。

我們會建議不同的動作，取決於您看到的其他等待事件。

主題

- 識別造成事件的工作階段和查詢
- 檢查是否與績效詳情計數器指標關聯
- 檢查是否有其他關聯的等待事件

識別造成事件的工作階段和查詢

通常，具有中等至重大負載的資料庫會有等待事件。如果效能是最佳的，則等待事件可能是可以接受的。如果效能不是最佳的，則檢查資料庫在何處花費最多時間。查看造成最高負載的等待事件，並了解您是否可以最佳化資料庫和應用程式，以減少這些事件。

尋找負責高負載的 SQL 查詢

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。即會顯示該資料庫執行個體的績效詳情儀表板。
4. 在 Database load (資料庫負載) 圖表中，選擇 Slice by wait (依等待建立配量)。
5. 在頁面底端，選擇 Top SQL (最高 SQL)。

此圖表會列出負責負載的 SQL 查詢。位於清單頂端者負最大責任。若要解決瓶頸，請專注於這些陳述式。

如需使用績效詳情進行疑難排解的實用概觀，請參閱部落格文章利用績效詳情分析 Amazon Aurora MySQL 工作負載。

檢查是否與績效詳情計數器指標關聯

檢查是否有績效詳情指標，例如 `Innodb_rows_changed`。如果計數器指標與 `io/table/sql/handler` 關聯，請遵循下列步驟：

1. 在績效詳情中，尋找說明 `io/table/sql/handler` 最高等待事件的 SQL 陳述式。如果可能，請最佳化此陳述式，以便其傳回較少的資料列。
2. 從 `schema_table_statistics` 和 `x$schema_table_statistics` 檢視中擷取最高資料表。這些檢視會顯示每個資料表所花費的時間量。如需詳細資訊，請參閱《MySQL 參考手冊》中的 schema_table_statistics 和 x$schema_table_statistics 檢視。

   根據預設，資料列會依總等待時間遞減排序。具有最多爭用的資料表會最先出現。輸出指示時間是花費在讀取、寫入、提取、插入、更新，還是刪除。

```
mysql> select * from sys.schema_table_statistics limit 1\G

*************************** 1. row ***************************
     table_schema: read_only_db
       table_name: sbtest41
    total_latency: 54.11 m
     rows_fetched: 6001557
    fetch_latency: 39.14 m
    rows_inserted: 14833
   insert_latency: 5.78 m
     rows_updated: 30470
   update_latency: 5.39 m
     rows_deleted: 14833
   delete_latency: 3.81 m
 io_read_requests: NULL
          io_read: NULL
  io_read_latency: NULL
io_write_requests: NULL
         io_write: NULL
 io_write_latency: NULL
```

```
 io_misc_requests: NULL
  io_misc_latency: NULL
1 row in set (0.11 sec)
```

檢查是否有其他關聯的等待事件

如果 synch/sxlock/innodb/btr_search_latch 和 io/table/sql/handler 是造成資料庫負載異常的主因，請檢查 innodb_adaptive_hash_index 變數是否已開啟。若是，請考慮增加 innodb_adaptive_hash_index_parts 參數值。

如果自適應雜湊索引已關閉，請考慮將其開啟。若要進一步了解 MySQL 自適應雜湊索引，請參閱下列資源：

- Percona 網站上的文章 InnoDB 中的自適應雜湊索引是否適合我的工作負載？
- 《MySQL 參考手冊》中的自適應雜湊索引
- Percona 網站上的文章 MySQL InnoDB 中的爭用：來自旗號區段的有用資訊

> **Note**
>
> Aurora 讀取器資料庫執行個體不支援自適應雜湊索引。
> 在某些情況下，當 synch/sxlock/innodb/btr_search_latch 和 io/table/sql/handler 主導時，讀取器執行個體的效能可能不佳。若是這樣，請考慮將工作負載暫時重新導向至寫入器節點，並開啟自適應雜湊索引。

synch/cond/innodb/row_lock_wait

當一個工作階段已鎖定資料列進行更新，而另一個工作階段嘗試更新同一資料列時，synch/cond/innodb/row_lock_wait 事件便會發生。如需詳細資訊，請參閱 MySQL 文件中的 InnoDB 鎖定。

支援的引擎版本

下列引擎版本支援這個等待事件資訊：

- Aurora MySQL 第 3 版

等待時間增加的可能原因

多個資料處理語言 (DML) 陳述式正在同時存取相同的資料列。

動作

我們會建議不同的動作，取決於您看到的其他等待事件。

主題

- 尋找並回應負責此等待事件的 SQL 陳述式
- 尋找並回應封鎖工作階段

尋找並回應負責此等待事件的 SQL 陳述式

使用績效詳情來識別負責此等待事件的 SQL 陳述式。請考慮下列策略：

- 如果資料列鎖定是持續發生的問題，請考慮重寫應用程式以使用樂觀鎖定。
- 使用多列陳述式。
- 將工作負載分散到不同的資料庫物件上。您可以透過分割來執行此動作。
- 檢查 innodb_lock_wait_timeout 參數的值。它控制交易在產生逾時錯誤之前等待多長時間。

如需使用績效詳情進行疑難排解的實用概觀，請參閱部落格文章利用績效詳情分析 Amazon Aurora MySQL 工作負載。

尋找並回應封鎖工作階段

判斷封鎖工作階段是閒置還是作用中。此外，了解工作階段來自應用程式還是作用中的使用者。

若要識別保留鎖定的工作階段，您可以執行 SHOW ENGINE INNODB STATUS。下列範例顯示範例輸出。

```
mysql> SHOW ENGINE INNODB STATUS;

---TRANSACTION 1688153, ACTIVE 82 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 2 row lock(s)
MySQL thread id 4244, OS thread handle 70369524330224, query id 4020834 172.31.14.179
 reinvent executing
select id1 from test.t1 where id1=1 for update
------- TRX HAS BEEN WAITING 24 SEC FOR THIS LOCK TO BE GRANTED:
```

```
RECORD LOCKS space id 11 page no 4 n bits 72 index GEN_CLUST_INDEX of table test.t1 trx
 id 1688153 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
```

或者您可以使用下列查詢來擷取目前鎖定的詳細資訊。

```
mysql> SELECT p1.id waiting_thread,
    p1.user waiting_user,
    p1.host waiting_host,
    it1.trx_query waiting_query,
    ilw.requesting_engine_transaction_id waiting_transaction,
    ilw.blocking_engine_lock_id blocking_lock,
    il.lock_mode blocking_mode,
    il.lock_type blocking_type,
    ilw.blocking_engine_transaction_id blocking_transaction,
    CASE it.trx_state
        WHEN 'LOCK WAIT'
        THEN it.trx_state
        ELSE p.state end blocker_state,
    concat(il.object_schema,'.', il.object_name) as locked_table,
    it.trx_mysql_thread_id blocker_thread,
    p.user blocker_user,
    p.host blocker_host
FROM performance_schema.data_lock_waits ilw
JOIN performance_schema.data_locks il
ON ilw.blocking_engine_lock_id = il.engine_lock_id
AND ilw.blocking_engine_transaction_id = il.engine_transaction_id
JOIN information_schema.innodb_trx it
ON ilw.blocking_engine_transaction_id = it.trx_id join information_schema.processlist p
ON it.trx_mysql_thread_id = p.id join information_schema.innodb_trx it1
ON ilw.requesting_engine_transaction_id = it1.trx_id join
 information_schema.processlist p1
ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
waiting_thread: 4244
waiting_user: reinvent
waiting_host: 123.456.789.012:18158
waiting_query: select id1 from test.t1 where id1=1 for update
waiting_transaction: 1688153
blocking_lock: 70369562074216:11:4:2:70369549808672
blocking_mode: X
blocking_type: RECORD
```

```
blocking_transaction: 1688142
blocker_state: User sleep
locked_table: test.t1
blocker_thread: 4243
blocker_user: reinvent
blocker_host: 123.456.789.012:18156
1 row in set (0.00 sec)
```

當您識別工作階段時，您的選項包括下列項目：

- 聯絡應用程式擁有者或使用者。
- 如果封鎖工作階段閒置，請考慮結束封鎖工作階段。此動作可能會觸發長時間回復。若要了解如何結束工作階段，請參閱 結束工作階段或查詢。

如需識別封鎖交易的詳細資訊，請參閱 MySQL 文件中的使用 InnoDB 交易與鎖定資訊。

synch/cond/innodb/row_lock_wait_cond

當一個工作階段已鎖定資料列進行更新，而另一個工作階段嘗試更新同一資料列時，synch/cond/innodb/row_lock_wait_cond 事件便會發生。如需詳細資訊，請參閱 MySQL 文件中的 InnoDB 鎖定。

支援的引擎版本

下列引擎版本支援這個等待事件資訊：

- Aurora MySQL 第 2 版

等待時間增加的可能原因

多個資料處理語言 (DML) 陳述式正在同時存取相同的資料列。

動作

我們會建議不同的動作，取決於您看到的其他等待事件。

主題

- 尋找並回應負責此等待事件的 SQL 陳述式
- 尋找並回應封鎖工作階段

尋找並回應負責此等待事件的 SQL 陳述式

使用績效詳情來識別負責此等待事件的 SQL 陳述式。請考慮下列策略：

- 如果資料列鎖定是持續發生的問題，請考慮重寫應用程式以使用樂觀鎖定。
- 使用多列陳述式。
- 將工作負載分散到不同的資料庫物件上。您可以透過分割來執行此動作。
- 檢查 innodb_lock_wait_timeout 參數的值。它控制交易在產生逾時錯誤之前等待多長時間。

如需使用績效詳情進行疑難排解的實用概觀，請參閱部落格文章利用績效詳情分析 Amazon Aurora MySQL 工作負載。

尋找並回應封鎖工作階段

判斷封鎖工作階段是閒置還是作用中。此外，了解工作階段來自應用程式還是作用中的使用者。

若要識別保留鎖定的工作階段，您可以執行 SHOW ENGINE INNODB STATUS。下列範例顯示範例輸出。

```
mysql> SHOW ENGINE INNODB STATUS;

---TRANSACTION 2771110, ACTIVE 112 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 24, OS thread handle 70369573642160, query id 13271336 172.31.14.179
 reinvent Sending data
select id1 from test.t1 where id1=1 for update
------- TRX HAS BEEN WAITING 43 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 11 page no 3 n bits 0 index GEN_CLUST_INDEX of table test.t1 trx
 id 2771110 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
```

或者您可以使用下列查詢來擷取目前鎖定的詳細資訊。

```
mysql> SELECT p1.id waiting_thread,
              p1.user waiting_user,
              p1.host waiting_host,
              it1.trx_query waiting_query,
              ilw.requesting_trx_id waiting_transaction,
              ilw.blocking_lock_id blocking_lock,
```

```
              il.lock_mode blocking_mode,
              il.lock_type blocking_type,
              ilw.blocking_trx_id blocking_transaction,
              CASE it.trx_state
                WHEN 'LOCK WAIT'
                THEN it.trx_state
                ELSE p.state
              END blocker_state,
              il.lock_table locked_table,
              it.trx_mysql_thread_id blocker_thread,
              p.user blocker_user,
              p.host blocker_host
       FROM information_schema.innodb_lock_waits ilw
       JOIN information_schema.innodb_locks il
         ON ilw.blocking_lock_id = il.lock_id
        AND ilw.blocking_trx_id = il.lock_trx_id
       JOIN information_schema.innodb_trx it
         ON ilw.blocking_trx_id = it.trx_id
       JOIN information_schema.processlist p
         ON it.trx_mysql_thread_id = p.id
       JOIN information_schema.innodb_trx it1
         ON ilw.requesting_trx_id = it1.trx_id
       JOIN information_schema.processlist p1
         ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
      waiting_thread: 3561959471
        waiting_user: reinvent
        waiting_host: 123.456.789.012:20485
       waiting_query: select id1 from test.t1 where id1=1 for update
 waiting_transaction: 312337314
       blocking_lock: 312337287:261:3:2
       blocking_mode: X
       blocking_type: RECORD
blocking_transaction: 312337287
       blocker_state: User sleep
        locked_table: `test`.`t1`
      blocker_thread: 3561223876
        blocker_user: reinvent
        blocker_host: 123.456.789.012:17746
1 row in set (0.04 sec)
```

當您識別工作階段時，您的選項包括下列項目：

- 聯絡應用程式擁有者或使用者。
- 如果封鎖工作階段閒置，請考慮結束封鎖工作階段。此動作可能會觸發長時間回復。若要了解如何結束工作階段，請參閱 結束工作階段或查詢。

如需識別封鎖交易的詳細資訊，請參閱 MySQL 文件中的使用 InnoDB 交易與鎖定資訊。

synch/cond/sql/MDL_context::COND_wait_status

有執行緒正在等待資料表中繼資料鎖定時，synch/cond/sql/MDL_context::COND_wait_status 事件便會發生。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因
- 動作

支援的引擎版本

下列引擎版本支援這個等待事件資訊：

- Aurora MySQL 2 版和 3 版

Context

事件 synch/cond/sql/MDL_context::COND_wait_status 指出有執行緒正在等待資料表中繼資料鎖定。在某些情況下，一個工作階段對資料表保有中繼資料鎖定，而另一個工作階段嘗試對同一資料表取得相同的鎖定。在這類情況下，第二個工作階段會等待 synch/cond/sql/MDL_context::COND_wait_status 等待事件。

MySQL 使用中繼資料鎖定來管理資料庫物件的並行存取，並確保資料一致性。中繼資料鎖定適用於資料表、結構描述、排程事件、資料表空間，以及使用 get_lock 函數和預存程式取得的使用者鎖定。預存程式包含程序、函數和觸發程序。如需詳細資訊，請參閱 MySQL 文件中的中繼資料鎖定。

MySQL 程序清單顯示這個工作階段處於狀態 waiting for metadata lock 中。在績效詳情中，如果 Performance_schema 已開啟，事件 synch/cond/sql/MDL_context::COND_wait_status 即會出現。

等待中繼資料鎖定之查詢的預設逾時是基於 lock_wait_timeout 參數的值，預設值為 31,536,000 秒 (365 天)。

如需有關不同 InnoDB 鎖定和可能導致衝突之鎖定類型的詳細資訊，請參閱 MySQL 文件中的 InnoDB 鎖定。

等待變多的可能原因

synch/cond/sql/MDL_context::COND_wait_status 事件比平時更常出現時，可能表示有效能問題，典型原因包括：

長時間執行的交易

一個或多個交易正在修改大量的資料，並對資料表保留鎖定很長一段時間。

交易閒置

一個或多個交易保持開啟狀態很長一段時間，沒有進行遞交或復原。

大型表格上的 DDL 陳述式

一或多個資料定義語言 (DDL) 陳述式，例如 ALTER TABLE 命令，已在非常大的資料表上執行。

明確資料表鎖定

未及時釋放的資料表上有明確的鎖定。例如，應用程式可能會不當地執行 LOCK TABLE 陳述式。

動作

我們會建議不同的動作，取決於等待事件的原因，以及 Aurora MySQL 資料庫叢集的版本。

主題

- 識別造成事件的工作階段和查詢
- 檢查是否有過去的活動
- 在 Aurora MySQL 第 2 版上執行查詢
- 回應封鎖工作階段

識別造成事件的工作階段和查詢

您可以使用績效詳情來顯示 synch/cond/sql/MDL_context::COND_wait_status 等待事件所封鎖的查詢。不過，若要識別封鎖工作階段，請從資料庫叢集上的 performance_schema 和 information_schema 查詢中繼資料表。

通常，具有中等至重大負載的資料庫會有等待事件。如果效能是最佳的，則等待事件可能是可以接受的。如果效能不是最佳的，則檢查資料庫在何處花費最多時間。查看造成最高負載的等待事件，並了解您是否可以最佳化資料庫和應用程式，以減少這些事件。

尋找負責高負載的 SQL 查詢

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。該資料庫執行個體的績效詳情儀表板即會出現。
4. 在 Database load (資料庫負載) 圖表中，選擇 Slice by wait (依等待建立配量)。
5. 在頁面底端，選擇 Top SQL (最高 SQL)。

   此圖表會列出負責負載的 SQL 查詢。位於清單頂端者負最大責任。若要解決瓶頸，請專注於這些陳述式。

如需使用績效詳情進行故障診斷的實用概觀，請參閱 AWS 資料庫部落格文章使用績效詳情分析 Amazon Aurora MySQL 工作負載。

檢查是否有過去的活動

您可以洞察這個等待事件，以檢查其是否發生過。若要這樣做，請完成下列動作。

- 檢查資料處理語言 (DML) 和 DDL 輸送量和延遲，以查看工作負載是否有任何變更。

  您可以使用績效詳情來尋找問題發生時等待此事件的查詢。此外，您也可以檢視接近問題發生時的查詢執行摘要。
- 如果開啟資料庫叢集的稽核日誌或一般日誌，您可以檢查所有查詢是否在等待交易中涉及的物件 (schema.table) 上執行。您也可以檢查是否有查詢在交易之前已完成執行。

疑難排解過去事件的可用資訊有限。執行這些檢查不會顯示哪個物件正在等待資訊。不過，您可以識別事件發生時負載繁重的資料表，以及發生問題時導致衝突的常操作資料列集。然後，您可以使用此資訊，在測試環境中重現問題，並提供有關其原因的洞察。

在 Aurora MySQL 第 2 版上執行查詢

在 Aurora MySQL 第 2 版中，您可以查詢 `performance_schema` 資料表或 `sys` 結構描述檢視，直接識別已封鎖的工作階段。範例可說明如何查詢資料表，以識別封鎖查詢和工作階段。

在下列程序清單輸出中，連線ID 89 正在等待中繼資料鎖定，並且正在執行 TRUNCATE TABLE 命令。在查詢中，於 performance_schema 資料表或 sys 結構描述檢視上，輸出會顯示封鎖工作階段是 76。

```
MySQL [(none)]> select @@version, @@aurora_version;
+-----------+------------------+
| @@version | @@aurora_version |
+-----------+------------------+
| 5.7.12    | 2.11.5           |
+-----------+------------------+
1 row in set (0.01 sec)

MySQL [(none)]> show processlist;
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+-------------------------------+
| Id | User            | Host               | db        | Command | Time | State
                    | Info                          |
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+-------------------------------+
|  2 | rdsadmin        | localhost          | NULL      | Sleep   |    0 | NULL
                    | NULL                          |
|  4 | rdsadmin        | localhost          | NULL      | Sleep   |    2 | NULL
                    | NULL                          |
|  5 | rdsadmin        | localhost          | NULL      | Sleep   |    1 | NULL
                    | NULL                          |
| 20 | rdsadmin        | localhost          | NULL      | Sleep   |    0 | NULL
                    | NULL                          |
| 21 | rdsadmin        | localhost          | NULL      | Sleep   |  261 | NULL
                    | NULL                          |
| 66 | auroramysql5712 | 172.31.21.51:52154 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 67 | auroramysql5712 | 172.31.21.51:52158 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 68 | auroramysql5712 | 172.31.21.51:52150 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 69 | auroramysql5712 | 172.31.21.51:52162 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 70 | auroramysql5712 | 172.31.21.51:52160 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 71 | auroramysql5712 | 172.31.21.51:52152 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 72 | auroramysql5712 | 172.31.21.51:52156 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
```

```
| 73 | auroramysql5712 | 172.31.21.51:52164 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 74 | auroramysql5712 | 172.31.21.51:52166 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 75 | auroramysql5712 | 172.31.21.51:52168 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 76 | auroramysql5712 | 172.31.21.51:52170 | NULL      | Query   |    0 | starting
                    | show processlist              |
| 88 | auroramysql5712 | 172.31.21.51:52194 | NULL      | Query   |   22 | User sleep
                    | select sleep(10000)           |
| 89 | auroramysql5712 | 172.31.21.51:52196 | NULL      | Query   |    5 | Waiting for
 table metadata lock | truncate table sbtest.sbtest1 |
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+------------------------------+
18 rows in set (0.00 sec)
```

接下來，performance_schema 資料表或 sys 結構描述檢視上的查詢會顯示封鎖工作階段是 76。

```
MySQL [(none)]> select * from sys.schema_table_lock_waits;

+---------------+-------------+-------------------+-------------
+-------------------------------+-------------------+-----------------------
+-------------------------------+--------------------+-----------------------------
+-------------------------------+--------------------+--------------
+-------------------------------+--------------------+-----------------------
+--------------------------+------------------------------+
| object_schema | object_name | waiting_thread_id | waiting_pid | waiting_account
           | waiting_lock_type | waiting_lock_duration | waiting_query
   | waiting_query_secs | waiting_query_rows_affected | waiting_query_rows_examined |
 blocking_thread_id | blocking_pid | blocking_account             | blocking_lock_type
 | blocking_lock_duration | sql_kill_blocking_query | sql_kill_blocking_connection |
+---------------+-------------+-------------------+-------------
+-------------------------------+-------------------+-----------------------
+-------------------------------+--------------------+-----------------------------
+-------------------------------+--------------------+--------------
+-------------------------------+--------------------+-----------------------
+--------------------------+------------------------------+
| sbtest        | sbtest1     |               121 |          89 |
 auroramysql5712@192.0.2.0    | EXCLUSIVE         | TRANSACTION           | truncate
 table sbtest.sbtest1 |                 10 |                           0 |
                0 |                108 |           76 | auroramysql5712@192.0.2.0    |
 SHARED_READ        | TRANSACTION            | KILL QUERY 76           | KILL 76
               |
```

```
+---------------+-------------+-------------------+-------------
+------------------------------+--------------------+----------------------
+------------------------------+--------------------+---------------------------
+------------------------------+--------------------+--------------
+------------------------------+--------------------+-----------------------
+--------------------------+-------------------------------+
1 row in set (0.00 sec)
```

回應封鎖工作階段

當您識別工作階段時，您的選項包括下列項目：

- 聯絡應用程式擁有者或使用者。
- 如果封鎖工作階段閒置，請考慮結束封鎖工作階段。此動作可能會觸發長時間回復。若要了解如何結束工作階段，請參閱 結束工作階段或查詢。

如需識別封鎖交易的詳細資訊，請參閱 MySQL 文件中的使用 InnoDB 交易與鎖定資訊。

synch/mutex/innodb/aurora_lock_thread_slot_futex

當一個工作階段已鎖定資料列進行更新，而另一個工作階段嘗試更新同一資料列時，`synch/mutex/innodb/aurora_lock_thread_slot_futex` 事件便會發生。如需詳細資訊，請參閱《MySQL 參考》中的 InnoDB 鎖定。

支援的引擎版本

下列引擎版本支援這個等待事件資訊：

- Aurora MySQL 第 2 版

等待時間增加的可能原因

多個資料處理語言 (DML) 陳述式正在同時存取相同的資料列。

動作

我們會建議不同的動作，取決於您看到的其他等待事件。

主題

- 尋找並回應負責此等待事件的 SQL 陳述式

- 尋找並回應封鎖工作階段

尋找並回應負責此等待事件的 SQL 陳述式

使用績效詳情來識別負責此等待事件的 SQL 陳述式。請考慮下列策略：

- 如果資料列鎖定是持續發生的問題，請考慮重寫應用程式以使用樂觀鎖定。
- 使用多列陳述式。
- 將工作負載分散到不同的資料庫物件上。您可以透過分割來執行此動作。
- 檢查 innodb_lock_wait_timeout 參數的值。它控制交易在產生逾時錯誤之前等待多長時間。

如需使用績效詳情進行疑難排解的實用概觀，請參閱部落格文章利用績效詳情分析 Amazon Aurora MySQL 工作負載。

尋找並回應封鎖工作階段

判斷封鎖工作階段是閒置還是作用中。此外，了解工作階段來自應用程式還是作用中的使用者。

若要識別保留鎖定的工作階段，您可以執行 SHOW ENGINE INNODB STATUS。下列範例顯示範例輸出。

```
mysql> SHOW ENGINE INNODB STATUS;

----------------------TRANSACTION 302631452, ACTIVE 2 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 376, 1 row lock(s)
MySQL thread id 80109, OS thread handle 0x2ae915060700, query id 938819 10.0.4.12
 reinvent updating
UPDATE sbtest1 SET k=k+1 WHERE id=503
------- TRX HAS BEEN WAITING 2 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 148 page no 11 n bits 30 index `PRIMARY` of table
 `sysbench2`.`sbtest1` trx id 302631452 lock_mode X locks rec but not gap waiting
Record lock, heap no 30 PHYSICAL RECORD: n_fields 6; compact format; info bits 0
```

或者您可以使用下列查詢來擷取目前鎖定的詳細資訊。

```
mysql> SELECT p1.id waiting_thread,
              p1.user waiting_user,
              p1.host waiting_host,
              it1.trx_query waiting_query,
```

```
              ilw.requesting_trx_id waiting_transaction,
              ilw.blocking_lock_id blocking_lock,
              il.lock_mode blocking_mode,
              il.lock_type blocking_type,
              ilw.blocking_trx_id blocking_transaction,
              CASE it.trx_state
                WHEN 'LOCK WAIT'
                THEN it.trx_state
                ELSE p.state
              END blocker_state,
              il.lock_table locked_table,
              it.trx_mysql_thread_id blocker_thread,
              p.user blocker_user,
              p.host blocker_host
       FROM information_schema.innodb_lock_waits ilw
       JOIN information_schema.innodb_locks il
         ON ilw.blocking_lock_id = il.lock_id
        AND ilw.blocking_trx_id = il.lock_trx_id
       JOIN information_schema.innodb_trx it
         ON ilw.blocking_trx_id = it.trx_id
       JOIN information_schema.processlist p
         ON it.trx_mysql_thread_id = p.id
       JOIN information_schema.innodb_trx it1
         ON ilw.requesting_trx_id = it1.trx_id
       JOIN information_schema.processlist p1
         ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
      waiting_thread: 3561959471
        waiting_user: reinvent
        waiting_host: 123.456.789.012:20485
       waiting_query: select id1 from test.t1 where id1=1 for update
 waiting_transaction: 312337314
       blocking_lock: 312337287:261:3:2
       blocking_mode: X
       blocking_type: RECORD
blocking_transaction: 312337287
       blocker_state: User sleep
        locked_table: `test`.`t1`
      blocker_thread: 3561223876
        blocker_user: reinvent
        blocker_host: 123.456.789.012:17746
1 row in set (0.04 sec)
```

當您識別工作階段時，您的選項包括下列項目：

- 聯絡應用程式擁有者或使用者。
- 如果封鎖工作階段閒置，請考慮結束封鎖工作階段。此動作可能會觸發長時間回復。若要了解如何結束工作階段，請參閱 結束工作階段或查詢。

如需識別封鎖交易的詳細資訊，請參閱《MySQL 參考手冊》中的使用 InnoDB 交易與鎖定資訊。

synch/mutex/innodb/buf_pool_mutex

當執行緒已對 InnoDB 緩衝集區取得鎖定來存取記憶體中的頁面時，`synch/mutex/innodb/buf_pool_mutex` 事件便會發生。

主題

- 相關的引擎版本
- Context
- 等待變多的可能原因
- 動作

相關的引擎版本

下列引擎版本支援這個等待事件資訊：

- Aurora MySQL 第 2 版

Context

`buf_pool` 互斥是單一互斥，其會保護緩衝集區的控制資料結構。

如需詳細資訊，請參閱 MySQL 文件中的使用效能結構描述監控 InnoDB 互斥等待。

等待變多的可能原因

這是工作負載特定的等待事件。`synch/mutex/innodb/buf_pool_mutex` 出現在最高等待事件之間的常見原因包括：

- 緩衝集區不夠大，無法容納工作資料集。
- 工作負載更特定於來自資料庫中特定資料表的特定頁面，從而導致緩衝集區中的爭用。

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 識別造成事件的工作階段和查詢
- 使用績效詳情
- 建立 Aurora 複本
- 檢查緩衝集區大小
- 監控全域狀態歷史記錄

識別造成事件的工作階段和查詢

通常，具有中等至重大負載的資料庫會有等待事件。如果效能是最佳的，則等待事件可能是可以接受的。如果效能不是最佳的，則檢查資料庫在何處花費最多時間。查看造成最高負載的等待事件，並了解您是否可以最佳化資料庫和應用程式，以減少這些事件。

在 AWS 管理主控台中檢視最高 SQL 圖表

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。即會顯示該資料庫執行個體的績效詳情儀表板。
4. 在 Database load (資料庫負載) 圖表中，選擇 Slice by wait (依等待建立配量)。
5. 在 Database load (資料庫負載) 圖表下方，選擇 Top SQL (最高 SQL)。

   此圖表會列出負責負載的 SQL 查詢。位於清單頂端者負最大責任。若要解決瓶頸，請專注於這些陳述式。

如需使用績效詳情進行疑難排解的實用概觀，請參閱部落格文章利用績效詳情分析 Amazon Aurora MySQL 工作負載。

使用績效詳情

此事件與工作負載相關。您可以使用績效詳情執行下列動作：

- 從應用程式日誌或相關來源識別等待事件開始的時間，以及工作負載在該段時間是否有任何變更。

- 識別負責此等待事件的 SQL 陳述式。檢查查詢的執行計劃，以確定這些查詢已最佳化，並使用適當的索引。

  如果負責等待事件的最高查詢與相同的資料庫物件或資料表相關，請考慮分割該物件或資料表。

建立 Aurora 複本

您可以建立 Aurora 複本來提供唯讀流量。您也可以使用 Aurora Auto Scaling 來處理讀取流量的突增。務必在 Aurora 複本上執行排定的唯讀任務和邏輯備份。

如需更多詳細資訊，請參閱 使用 Amazon Aurora 自動擴展搭配 Aurora 複本。

檢查緩衝集區大小

查看指標 `innodb_buffer_pool_wait_free`，檢查緩衝集區大小是否足以處理工作負載。如果此指標的值很高且持續增加，這表示緩衝集區的大小不足以處理工作負載。如果 `innodb_buffer_pool_size` 已正確設定，則 `innodb_buffer_pool_wait_free` 的值應該很小。如需詳細資訊，請參閱 MySQL 文件中的 Innodb_buffer_pool_wait_free。

如果資料庫執行個體具有足夠的記憶體，可供工作階段緩衝區和作業系統任務使用，請增加緩衝區集區大小。如果沒有，請將資料庫執行個體變更為較大的資料庫執行個體類別，以取得可配置給緩衝集區的額外記憶體。

> **Note**
>
> Aurora MySQL 會根據設定的 `innodb_buffer_pool_size` 自動調整 `innodb_buffer_pool_instances` 的值。

監控全域狀態歷史記錄

透過監控狀態變數的變更率，您可以在資料庫執行個體上偵測鎖定或記憶體問題。開啟全域狀態歷史記錄 (GoSH)，如果尚未開啟的話。如需 GoSh 的詳細資訊，請參閱管理全域狀態歷史記錄。

您也可以建立自訂 Amazon CloudWatch 指標來監控狀態變數。如需詳細資訊，請參閱發佈自訂指標。

synch/mutex/innodb/fil_mutex

當工作階段正在等待存取資料表空間記憶體快取時，`synch/mutex/innodb/fil_system_mutex` 事件便會發生。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因
- 動作

支援的引擎版本

下列引擎版本支援這個等待事件資訊：

- Aurora MySQL 2 版和 3 版

Context

InnoDB 會使用資料表空間來管理資料表和日誌檔案的儲存區域。「資料表空間記憶體快取」是全域記憶體結構，用於維護資料表空間的相關資訊。MySQL 會使用 `synch/mutex/innodb/fil_system_mutex` 等待來控制資料表空間記憶體快取的並行存取。

事件 `synch/mutex/innodb/fil_system_mutex` 指出目前有一個以上的作業需要擷取和操控相同資料表空間之資料表空間記憶體快取中的資訊。

等待變多的可能原因

`synch/mutex/innodb/fil_system_mutex` 事件比平時更常出現時，可能表示有效能問題，這通常發生在下列所有情況都存在時：

- 更新或刪除同一資料表中資料的並行資料處理語言 (DML) 作業增加。
- 此資料表的資料表空間非常大，並有許多資料頁面。
- 這些資料頁面的填滿係數很低。

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 識別造成事件的工作階段和查詢
- 在離峰時間重組大型資料表

識別造成事件的工作階段和查詢

通常，具有中等至重大負載的資料庫會有等待事件。如果效能是最佳的，則等待事件可能是可以接受的。如果效能不是最佳的，請檢查資料庫在何處花費最多時間。查看造成最高負載的等待事件，並了解您是否可以最佳化資料庫和應用程式，以減少這些事件。

尋找負責高負載的 SQL 查詢

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。該資料庫執行個體的績效詳情儀表板即會出現。
4. 在 Database load (資料庫負載) 圖表中，選擇 Slice by wait (依等待建立配量)。
5. 在頁面底端，選擇 Top SQL (最高 SQL)。

   此圖表會列出負責負載的 SQL 查詢。位於清單頂端者負最大責任。若要解決瓶頸，請專注於這些陳述式。

如需使用績效詳情進行疑難排解的實用概觀，請參閱部落格文章利用績效詳情分析 Amazon Aurora MySQL 工作負載。

了解哪些查詢導致大量 `synch/mutex/innodb/fil_system_mutex` 等待的另一種方法是檢查 `performance_schema`，如下列範例所示。

```
mysql> select * from performance_schema.events_waits_current where EVENT_NAME='wait/
synch/mutex/innodb/fil_system_mutex'\G
*************************** 1. row ***************************
            THREAD_ID: 19
             EVENT_ID: 195057
         END_EVENT_ID: 195057
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:6700
          TIMER_START: 1010146190118400
            TIMER_END: 1010146196524000
           TIMER_WAIT: 6405600
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
```

```
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
*************************** 2. row ***************************
            THREAD_ID: 23
             EVENT_ID: 5480
         END_EVENT_ID: 5480
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:5906
          TIMER_START: 995269979908800
            TIMER_END: 995269980159200
           TIMER_WAIT: 250400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
*************************** 3. row ***************************
            THREAD_ID: 55
             EVENT_ID: 23233794
         END_EVENT_ID: NULL
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:449
          TIMER_START: 1010492125341600
            TIMER_END: 1010494304900000
           TIMER_WAIT: 2179558400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: 23233786
   NESTING_EVENT_TYPE: WAIT
            OPERATION: lock
```

```
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
```

在離峰時間重組大型資料表

在生產時間以外的維護時段期間，重組您識別為大量 synch/mutex/innodb/fil_system_mutex 等待事件之來源的大型資料表。這樣做可確保內部資料表空間映射清除不會在快速存取資料表至關重要時發生。如需重組資料表的相關資訊，請參閱《MySQL 參考》中的 OPTIMIZE TABLE 陳述式。

synch/mutex/innodb/fil_mutex

由於大量交易而有高資料庫活動時，synch/mutex/innodb/trx_sys_mutex 事件便會發生。

主題

- 相關的引擎版本
- Context
- 等待變多的可能原因
- 動作

相關的引擎版本

下列引擎版本支援這個等待事件資訊：

- Aurora MySQL 2 版和 3 版

Context

在內部，InnoDB 資料庫引擎會使用可重複讀取隔離層級搭配快照，來提供讀取一致性。這可為您提供在建立快照時資料庫的時間點檢視。

在 InnoDB 中，所有變更一到達就會套用到資料庫，無論是否已遞交它們。這種方法表示，若沒有多版本並行控制 (MVCC)，連線到資料庫的所有使用者都會看到所有的變更和最新的資料列。因此，InnoDB 需要一種方法來追蹤變更，以了解在必要時要復原什麼。

若要這樣做，InnoDB 會使用交易系統 (trx_sys) 來追蹤快照。交易系統會執行下列動作：

- 追蹤復原日誌中每個資料列的交易 ID。
- 使用名為 ReadView 的內部 InnoDB 結構，其有助於識別快照可以看到哪些交易 ID。

等待變多的可能原因

以一致且受控方式處理 (建立、讀取、更新和刪除) 交易 ID 的任何資料庫作業都會從 `trx_sys` 產生對互斥的呼叫。

這些呼叫發生在三個函數內：

- `trx_sys_mutex_enter` – 建立互斥。
- `trx_sys_mutex_exit` – 釋放互斥。
- `trx_sys_mutex_own` – 測試是否擁有互斥。

InnoDB 效能結構描述檢測會追蹤所有 `trx_sys` 互斥呼叫。追蹤包括但不限於在資料庫啟動或關閉時管理 `trx_sys`、復原作業、復原清除、資料列讀取存取，以及緩衝集區載入。具有大量交易的高資料庫活動會導致 `synch/mutex/innodb/trx_sys_mutex` 出現在最高等待事件之間。

如需詳細資訊，請參閱 MySQL 文件中的使用效能結構描述監控 InnoDB 互斥等待。

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 識別造成事件的工作階段和查詢
- 檢查其他等待事件

識別造成事件的工作階段和查詢

通常，具有中等至重大負載的資料庫會有等待事件。如果效能是最佳的，則等待事件可能是可以接受的。如果效能不是最佳的，則檢查資料庫在何處花費最多時間。查看導致最高負載的等待事件。了解您是否可以最佳化資料庫和應用程式，以減少這些事件。

檢視 AWS 管理主控台 中的最高 SQL 圖表

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。即會顯示該資料庫執行個體的績效詳情儀表板。
4. 在 Database load (資料庫負載) 圖表中，選擇 Slice by wait (依等待建立配量)。
5. 在 Database load (資料庫負載) 圖表下，選擇 Top SQL (最高 SQL)。

此圖表會列出負責負載的 SQL 查詢。位於清單頂端者負最大責任。若要解決瓶頸，請專注於這些陳述式。

如需使用績效詳情進行疑難排解的實用概觀，請參閱部落格文章利用績效詳情分析 Amazon Aurora MySQL 工作負載。

檢查其他等待事件

檢查與 synch/mutex/innodb/trx_sys_mutex 等待事件相關聯的其他等待事件。執行此動作可提供工作負載本質的詳細資訊。大量的交易可能會減少輸送量，但工作負載也可能使此變得必要。

如需如何最佳化交易的詳細資訊，請參閱 MySQL 文件中的最佳化 InnoDB 交易管理。

synch/sxlock/innodb/hash_table_locks

當緩衝集區中找不到的頁面必須從儲存體讀取時，synch/sxlock/innodb/hash_table_locks 事件便會發生。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因
- 動作

支援的引擎版本

下列版本支援這個等待事件資訊：

- Aurora MySQL 2 版和 3 版

Context

事件 synch/sxlock/innodb/hash_table_locks 指出工作負載經常存取未存放在緩衝集區中的資料。此等待事件與新的頁面新增相關，而且舊資料會從緩衝集區移出。存放在緩衝集區的資料過時，而且新資料必須進行快取，因此會將過時頁面移出，以允許快取新頁面。MySQL 會使用最近最少使用 (LRU) 演算法，從緩衝集區移出頁面。工作負載嘗試存取尚未載入至緩衝集區的資料或已從緩衝集區移出的資料。

當工作負載必須存取磁碟上檔案中的資料，或從緩衝集區的 LRU 清單釋放區塊，或將區塊新增至其中時，此等待事件便會發生。這些作業等待取得共用的排除鎖定 (SX-lock)。此 SX-lock 用於透過「雜湊表」進行同步，此雜湊表是記憶體中的資料表，旨在改善緩衝集區存取效能。

如需詳細資訊，請參閱 MySQL 文件中的緩衝集區。

等待變多的可能原因

synch/sxlock/innodb/hash_table_locks 等待事件比平時更常出現時，可能表示有效能問題，典型原因包括：

緩衝集區大小過小

緩衝集區的大小太小，無法將所有經常存取的頁面保留在記憶體中。

工作負載繁重

工作負載導致頻繁的移出，並在緩衝區快取中重新載入資料頁面。

讀取頁面時發生錯誤

讀取緩衝集區中的頁面時發生錯誤，這可能表示資料損毀。

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 增加緩衝集區的大小
- 改善資料存取模式
- 減少或避免完整資料表掃描
- 檢查錯誤日誌是否有頁面毀損

增加緩衝集區的大小

確定緩衝集區已適當地調整為適合工作負載的大小。若要這樣做，您可以檢查緩衝區快取命中率。通常，如果值低於 95%，請考慮增加緩衝集區大小。越大的緩衝集區可以將經常存取的頁面保留在記憶體中的時間越長 若要增加緩衝集區的大小，請修改 innodb_buffer_pool_size 參數的值。此參數的預設值是基於資料庫執行個體類別大小。如需詳細資訊，請參閱 Amazon Aurora MySQL 資料庫組態的最佳實務。

改善資料存取模式

檢查受此等待及其執行計劃影響的查詢。考慮改善資料存取模式。例如，如果您是使用 mysqli_result::fetch_array，則可以嘗試增加陣列擷取大小。

您可以使用績效詳情來顯示可能造成 `synch/sxlock/innodb/hash_table_locks` 等待事件的查詢和工作階段。

尋找負責高負載的 SQL 查詢

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。即會顯示該資料庫執行個體的績效詳情儀表板。
4. 在 Database load (資料庫負載) 圖表中，選擇 Slice by wait (依等待建立配量)。
5. 在頁面底端，選擇 Top SQL (最高 SQL)。

   此圖表會列出負責負載的 SQL 查詢。位於清單頂端者負最大責任。若要解決瓶頸，請專注於這些陳述式。

如需使用績效詳情進行疑難排解的實用概觀，請參閱 AWS 資料庫部落格文章利用績效詳情分析 Amazon Aurora MySQL 工作負載。

減少或避免完整資料表掃描

監控工作負載，以查看它是否正在執行完整資料表掃描，若是，請減少或避免它們。例如，您可以監控狀態變數，例如 `Handler_read_rnd_next`。如需詳細資訊，請參閱 MySQL 文件中的伺服器狀態變數。

檢查錯誤日誌是否有頁面毀損

你可以檢查 mysql-error.log，是否在接近問題發生時偵測到毀損相關訊息。您可以用來解決問題的訊息位於錯誤日誌中。您可能需要重新建立已報告為毀損的物件。

synch/mutex/innodb/temp_pool_manager_mutex

當工作階段正在等待取得互斥以管理工作階段暫存資料表空間集區時，會發生`synch/mutex/innodb/temp_pool_manager_mutex`等待事件。

主題

- 支援的引擎版本
- Context
- 等待時間增加的可能原因
- 動作

支援的引擎版本

下列引擎版本支援這個等待事件資訊：

- Aurora MySQL 第 3 版

Context

Aurora MySQL 3.x 版和更新版本使用 `temp_pool_manager_mutex` 來控制同時存取暫存資料表空間集區的多個工作階段。Aurora MySQL 透過 Aurora 叢集磁碟區管理持久性資料儲存，以及管理暫存檔案的本機儲存。當工作階段在 Aurora 叢集磁碟區上建立暫存資料表時，需要暫存資料表空間。

當工作階段第一次請求暫存資料表空間時，MySQL 會從共用集區配置工作階段暫存資料表空間。對於下列資料表類型，工作階段一次最多可保留 2 個暫存資料表空間：

- 使用者建立的暫存資料表
- Optimizer 產生的內部暫時資料表

預設`TempTable`引擎使用下列溢位機制來處理暫存資料表：

- 將資料表存放在 RAM 中，最多可達 `temptable_max_ram`限制。
- 當 RAM 已滿時，移至本機儲存體上的記憶體映射檔案。
- 當記憶體映射檔案達到其`temptable_max_mmap`限制時，使用共用叢集磁碟區。

暫存資料表超過 RAM 和本機儲存限制後，MySQL 會使用磁碟上資料表空間來管理它們。

當工作階段需要磁碟上暫存資料表時，MySQL：

- 尋找集區中要重複使用的可用`INACTIVE`資料表空間。
- 如果沒有`INACTIVE`空格，則建立新的 10 個資料表空間。

當工作階段中斷連線時，MySQL：

- 截斷工作階段的暫存資料表空間。
- 在集區中將其標記為非作用中以重複使用。
- 維護目前的集區大小，直到伺服器重新啟動為止。
- 重新啟動後，返回預設集區大小 (10 個資料表空間）。

等待時間增加的可能原因

導致此等待事件的常見情況：

- 在叢集磁碟區上建立內部暫存資料表的並行工作階段。
- 在叢集磁碟區上建立使用者暫存資料表的並行工作階段。
- 使用作用中資料表空間突然終止工作階段。
- 大量寫入工作負載期間的資料表空間集區擴展。
- 並行查詢存取 `INFORMATION_SCHEMA`.

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 監控和最佳化暫存資料表用量
- 使用 INFORMATION_SCHEMA 檢閱查詢
- 增加 innodb_sync_array_size 參數
- 實作連線集區

監控和最佳化暫存資料表用量

若要監控和最佳化暫存資料表用量，請使用下列其中一種方法：

- 檢查績效詳情中的`Created_tmp_disk_tables`計數器，以追蹤整個 Aurora 叢集的磁碟上暫存資料表建立。
- 在資料庫中執行此命令，以直接監控暫存資料表的建立：`mysql> show status like '%created_tmp_disk%'`。

> **Note**
>
> Aurora MySQL 讀取器節點和寫入器節點之間的暫時資料表行為不同。如需詳細資訊，請參閱Aurora MySQL 第 3 版的新暫時資料表行為。

識別建立暫存資料表的查詢之後，請採取下列最佳化步驟：

- 使用 EXPLAIN 檢查查詢執行計劃，並識別建立暫存資料表的位置和原因。
- 盡可能修改查詢以減少暫時資料表用量。

如果僅查詢最佳化無法解決效能問題，請考慮調整這些組態參數：

- temptable_max_ram- 控制暫存資料表的 RAM 用量上限。
- temptable_max_mmap - 設定記憶體映射檔案儲存的限制。
- tmp_table_size- 啟用 aurora_tmptable_enable_per_table_limit 時套用（預設為停用）。

> **Important**
>
> 請注意，無論組態設定為何，某些查詢條件一律需要磁碟上暫存資料表。如需詳細資訊TempTable，請參閱在 Amazon RDS for MySQL 和 Amazon Aurora MySQL 上使用 TempTable 儲存引擎。

使用 INFORMATION_SCHEMA 檢閱查詢

當您查詢INFORMATION_SCHEMA資料表時，MySQL 會在叢集磁碟區上建立 InnoDB 暫存資料表。每個工作階段都需要這些資料表的暫時資料表空間，這可能會在高並行存取期間導致效能問題。

若要改善效能：

- INFORMATION_SCHEMA 盡可能使用 PERFORMANCE_SCHEMA而非 。
- 如果您必須使用 INFORMATION_SCHEMA，請減少執行這些查詢的頻率。

增加 innodb_sync_array_size 參數

`innodb_sync_array_size` 參數控制 MySQL 中互斥/鎖定等待陣列的大小。的預設值1適用於一般工作負載，但增加預設值可以減少高並行期間的執行緒爭用。

當您的工作負載顯示等待執行緒數量增加時：

- 監控工作負載中等待執行緒的數量。
- 將 設定為`innodb_sync_array_size`等於或高於執行個體的 vCPU 計數，以分割執行緒協調結構並減少爭用。

> **Note**
>
> 若要判斷 RDS 執行個體上可用的 vCPUs 數量，請參閱 Amazon RDS 執行個體類型 中的 vCPU 規格。

實作連線集區

MySQL 會將專用資料表空間指派給建立暫存資料表的每個工作階段。此資料表空間會保持作用中狀態，直到資料庫連線結束為止。若要更有效率地管理您的 資源：

- 實作連線集區以限制作用中暫存資料表空間的數量。
- 重複使用現有的連線，而不是為每個操作建立新的連線。

使用執行緒狀態調校 Aurora MySQL

下表彙總了 Aurora MySQL 最常見的一般執行緒狀態。

一般執行緒狀態	描述
???	此執行緒狀態指出執行緒正在處理 SELECT 陳述式，而此陳述式需要使用內部暫時資料表來排序資料。
???	此執行緒狀態指出執行緒正在讀取和篩選查詢的資料列，以決定正確的結果集。

正在建立排序索引

`creating sort index` 執行緒狀態指出執行緒正在處理 SELECT 陳述式，而此陳述式需要使用內部暫時資料表來排序資料。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因
- 動作

支援的引擎版本

下列版本支援這個執行緒狀態資訊：

- Aurora MySQL 第 2 版，最高至 2.09.2

Context

當具有 `ORDER BY` 或 `GROUP BY` 子句的查詢無法使用現有的索引來執行作業時，`creating sort index` 狀態便會出現。在此情況下，MySQL 需要執行更昂貴的 `filesort` 作業。如果結果集不太大，此作業通常會在記憶體中執行。否則，它涉及在磁碟上建立檔案。

等待變多的可能原因

出現 `creating sort index` 本身並不表示有問題。如果效能不佳，並且您經常看到 `creating sort index` 的執行個體，最可能的原因是搭配 `ORDER BY` 或 `GROUP BY` 運算子的慢速查詢。

動作

一般指導方針是尋找搭配 `ORDER BY` 或 `GROUP BY` 子句的查詢，這些子句與 `creating sort index` 狀態中的增加相關聯。然後查看新增索引或增加排序緩衝區大小是否可以解決問題。

主題

- 如果未開啟效能結構描述，請將其開啟
- 識別問題查詢
- 檢查說明計劃以取得檔案排序使用情形
- 增加排序緩衝區大小

如果未開啟效能結構描述，請將其開啟

只在效能結構描述檢測未開啟時，績效詳情才會報告執行緒狀態。當效能結構描述檢測開啟時，績效詳情會改為報告等待事件。效能結構描述檢測會在您調查潛在的效能問題時提供其他洞察和更好的工具。因此，建議您開啟效能結構描述。如需詳細資訊，請參閱Aurora MySQL 上 Performance Insights 的效能結構描述概觀。

識別問題查詢

若要識別目前哪些查詢正在造成 `creating sort index` 狀態中的增加，請執行 `show processlist` 並查看是否有任何查詢具有 `ORDER BY` 或 `GROUP BY`。選擇性地執行 `explain for connection N`，其中 N 是搭配 `filesort` 之查詢的程序清單 ID。

若要識別過去哪些查詢造成這些增加，請開啟慢速查詢日誌，並尋找搭配 `ORDER BY` 的查詢。對慢速查詢執行 `EXPLAIN`，並尋找「使用檔案排序」。如需詳細資訊，請參閱檢查說明計劃以取得檔案排序使用情形。

檢查說明計劃以取得檔案排序使用情形

識別哪些陳述式搭配導致 `creating sort index` 狀態的 `ORDER BY` 或 `GROUP BY` 子句。

下列範例顯示如何對查詢執行 `explain`：`Extra` 資料欄顯示此查詢使用 `filesort`。

```
mysql> explain select * from mytable order by c1 limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: mytable
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2064548
     filtered: 100.00
        Extra: Using filesort
1 row in set, 1 warning (0.01 sec)
```

下列範例顯示在資料欄 c1 上建立索引之後對同一查詢執行 EXPLAIN 的結果。

```
mysql> alter table mytable add index (c1);
```

```
mysql> explain select * from mytable order by c1 limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: mytable
   partitions: NULL
         type: index
possible_keys: NULL
          key: c1
      key_len: 1023
          ref: NULL
         rows: 10
     filtered: 100.00
        Extra: Using index
1 row in set, 1 warning (0.01 sec)
```

如需使用索引進行排序最佳化的相關資訊，請參閱 MySQL 文件中的 Orice By 最佳化。

增加排序緩衝區大小

若要查看特定查詢是否需要已在磁碟上建立檔案的 filesort 程序，請在執行查詢之後檢查 sort_merge_passes 變數值。下列顯示一個範例。

```
mysql> show session status like 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.01 sec)

--- run query
mysql> select * from mytable order by u limit 10;
--- run status again:

mysql> show session status like 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.01 sec)
```

如果 sort_merge_passes 的值很高，請考慮增加排序緩衝區大小。在工作階段層級套用增加，因為全域增加它可以大幅地增加 RAM MySQL 使用數量。下列範例顯示如何在執行查詢之前變更排序緩衝區大小。

```
mysql> set session sort_buffer_size=10*1024*1024;
Query OK, 0 rows affected (0.00 sec)
-- run query
```

正在傳送資料

sending data 執行緒狀態表示執行緒正在讀取和篩選查詢的資料列，以決定正確的結果集。名稱產生誤導，因為它暗示狀態正在傳輸資料，而不是收集和準備稍後傳送的資料。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因
- 動作

支援的引擎版本

下列版本支援這個執行緒狀態資訊：

- Aurora MySQL 第 2 版，最高至 2.09.2

Context

許多執行緒狀態是短暫的。`sending data` 期間發生的作業傾向於執行大量磁碟或快取讀取。因此，`sending data` 通常是執行時間最長的狀態，涵蓋給定查詢的生命週期。當 Aurora MySQL 執行下列動作時，此狀態便會出現：

- 讀取和處理 SELECT 陳述式的資料列
- 從磁碟或記憶體執行大量讀取
- 從特定查詢中完成所有資料的完整讀取
- 從資料表、索引或預存程序的工作中讀取資料
- 排序、分組或排序資料

在 `sending data` 狀態完成準備資料之後，執行緒狀態 `writing to net` 指出資料傳回至用戶端。通常，只在結果集非常大或嚴重的網路延遲正在減慢傳輸速度時，才會擷取 `writing to net`。

等待變多的可能原因

出現 `sending data` 本身並不表示有問題。如果效能不佳，並且您經常看到 `sending data` 的執行個體，最可能的原因如下。

主題

- 無效率查詢
- 次佳伺服器組態

無效率查詢

在大多數情況下，應對此狀態負責的是未使用適當的索引來尋找特定查詢之結果集的查詢。例如，考慮一個查詢，其為加州所有訂單讀取 1,000 萬筆記錄資料表，其中狀態資料行未建立索引或未好好建立索引。在後者情況下，索引可能存在，但最佳化工具由於低基數而忽略它。

次佳伺服器組態

如果數個查詢出現在 `sending data` 狀態中，則可能未好好設定資料庫伺服器。尤其，伺服器可能有下列問題：

- 資料庫伺服器沒有足夠的運算容量：磁碟輸入/輸出、磁碟類型和速度、CPU 或 CPU 數量。
- 伺服器缺乏配置的資源，例如 InnoDB 資料表的 InnoDB 緩衝集區或 MyISAM 資料表的索引鍵緩衝區。
- 每個執行緒記憶體設定 (例如 `sort_buffer`、`read_buffer` 和 `join_buffer`) 都會耗用比所需還要多的 RAM，因而使實體伺服器缺乏記憶體資源。

動作

一般指導方針是檢查效能結構描述來尋找傳回大量資料列的查詢。如果不使用索引的記錄查詢已開啟，您也可以檢查來自慢速日誌的結果。

主題

- 如果未開啟效能結構描述，請將其開啟
- 檢查記憶體設定
- 檢查說明計劃以取得索引使用情形
- 檢查傳回的資料量
- 檢查是否有並行問題
- 檢查請求的結構

如果未開啟效能結構描述，請將其開啟

只在效能結構描述檢測未開啟時，績效詳情才會報告執行緒狀態。當效能結構描述檢測開啟時，績效詳情會改為報告等待事件。效能結構描述檢測會在您調查潛在的效能問題時提供其他洞察和更好的工具。因此，建議您開啟效能結構描述。如需詳細資訊，請參閱Aurora MySQL 上 Performance Insights 的效能結構描述概觀。

檢查記憶體設定

檢查主要緩衝集區的記憶體設定。確定這些集區已適當地調整為適合工作負載的大小。如果您的資料庫使用多個緩衝集區執行個體，請確定它們不會分成許多小型緩衝集區。資料表一次只能使用一個緩衝集區。

請確定下列用於每個執行緒的記憶體設定，其大小適當：

- read_buffer
- read_rnd_buffer
- sort_buffer
- join_buffer
- binlog_cache

除非您有修改設定的特定原因，否則請使用預設值。

檢查說明計劃以取得索引使用情形

對於處於 sending data 執行緒狀態的查詢，請檢查計劃，以判斷是否使用適當的索引。如果查詢未使用有用的索引，請考慮新增 USE INDEX 或 FORCE INDEX 之類的提示。提示可大幅增加或減少執行查詢所需的時間，因此在新增它們之前請小心。

檢查傳回的資料量

檢查正在查詢的資料表，以及它們包含的資料量。可以封存此資料的任何一個嗎？ 在許多情況下，查詢執行時間不佳的原因不是查詢計劃的結果，而是要處理的資料量。許多開發人員在將資料新增到資料庫時非常有效率，但在設計和開發階段很少考慮資料集生命週期。

尋找在低容量資料庫中表現良好，但在目前系統中表現不佳的查詢。有時設計特定查詢的開發人員可能無法意識到這些查詢正在傳回 350,000 個資料列。開發人員可能已在較低容量環境 (其具有的資料集比生產環境具有的還要小) 開發了查詢。

檢查是否有並行問題

檢查相同類型的多個查詢是否同時執行。某些形式的查詢在單獨執行時可以有效地執行。不過，如果類似形式的查詢一起執行或大量執行，它們可能會導致並發問題。通常，當資料庫使用暫時資料表來呈現結果時，就會造成這些問題。限制性交易隔離層級也可能會導致並行問題。

如果同時讀取和寫入至資料表，則資料庫可能正在使用鎖定。若要協助識別效能不佳的期間，請透過大規模的批次程序檢查資料庫的使用情形。若要查看最近的鎖定和回復，請檢查 SHOW ENGINE INNODB STATUS 命令的輸出。

檢查請求的結構

檢查從這些狀態擷取的查詢是否使用子查詢。這種類型的查詢通常會導致效能不佳，因為資料庫會在內部編譯結果，然後將它們代入查詢以呈現資料。此程序是資料庫的額外步驟。在許多情況下，這個步驟可能會在高度並行載入條件中造成不佳的效能。

亦請檢查您的查詢是否使用大量的 ORDER BY 和 GROUP BY 子句。在這類操作中，通常資料庫必須首先在記憶體中形成整個資料集。然後，必須在將其傳送至用戶端之前以特定方式將其排序或分組。

使用 Amazon DevOps Guru 主動洞察，調校 Aurora MySQL

DevOps Guru 主動洞察會在 Aurora MySQL 資料庫叢集上的異常狀況發生之前，即偵測到這些問題。DevOps Guru 可以執行下列動作：

- 透過交叉檢查一般建議設定與您的資料庫設定，避免許多常見的資料庫問題。
- 警告您機群內的重大問題，若未勾選，可能導致更嚴重的問題。
- 提醒您新發現的問題。

每個主動洞察都包含問題原因分析和修正動作建議。

主題

- InnoDB 歷史記錄清單長度顯著增加
- 資料庫正在磁碟上建立暫存資料表

InnoDB 歷史記錄清單長度顯著增加

從##到 *db-instance* 的##，針對資料列變更的歷史記錄清單長度皆顯著增加。這會影響查詢和資料庫關閉效能。

主題

- 支援的引擎版本
- Context
- 造成此問題的可能原因
- 動作
- 相關指標

支援的引擎版本

所有版本的 Aurora MySQL 都支援此洞察資訊。

Context

InnoDB 交易系統會維護多版本並行控制 (MVCC)。修改資料列時，修改前的版本會在還原日誌中儲存為還原記錄。每個還原記錄都有對先前重做記錄的參照，進而形成一份連結清單。

InnoDB 歷史記錄清單是已提交交易的還原記錄全域清單。交易不再需要歷史記錄時，MySQL 會用歷史記錄清單來清除記錄和日誌頁面。歷史記錄清單長度是清單中所有修改項目的還原記錄總數。每個日誌都包含一個或多個修改項目。若 InnoDB 歷史記錄清單長度太長，表示該清單具有過多舊資料列版本，導致查詢和資料庫關閉變慢。

造成此問題的可能原因

歷史記錄清單過長的典型原因包括：

- 長時間執行的交易 (讀取或寫入)
- 繁重的寫入負載

動作

根據洞察的原因，我們會建議不同的動作。

主題

- 在 InnoDB 歷史記錄清單減少之前，請不要開始任何涉及資料庫關閉的操作
- 找出並關閉長時間執行的交易
- 使用績效詳情，找出最高主機和最高使用者。

在 InnoDB 歷史記錄清單減少之前，請不要開始任何涉及資料庫關閉的操作

InnoDB 歷史記錄清單過長會降低資料庫關閉速度，因此請在啟動涉及資料庫關閉的操作之前減少清單大小。這些操作包括主要版本資料庫升級。

找出並關閉長時間執行的交易

您可以透過查詢 `information_schema.innodb_trx`，找出長時間執行的交易。

> **Note**
>
> 也請務必在僅供讀取複本上尋找長時間執行的交易。

若要找出並關閉長時間執行的交易

1. 在您的 SQL 用戶端執行下列查詢：

```
SELECT a.trx_id,
      a.trx_state,
      a.trx_started,
      TIMESTAMPDIFF(SECOND,a.trx_started, now()) as "Seconds Transaction Has Been
 Open",
      a.trx_rows_modified,
      b.USER,
      b.host,
      b.db,
      b.command,
      b.time,
      b.state
FROM  information_schema.innodb_trx a,
      information_schema.processlist b
WHERE a.trx_mysql_thread_id=b.id
  AND TIMESTAMPDIFF(SECOND,a.trx_started, now()) > 10
ORDER BY trx_started
```

2. 使用預存程序 mysql.rds_kill 結束每個長時間執行的交易。

使用績效詳情，找出最高主機和最高使用者。

最佳化交易，以立即遞交大量已修改的資料列。

相關指標

下列指標與此洞察相關：

- `trx_rseg_history_len` – 您可以在績效詳情和 `INFORMATION_SCHEMA.INNODB_METRICS` 資料表中檢視此計數器指標。如需詳細資訊，請參閱 MySQL 文件中的 InnoDB INFORMATION_SCHEMA 指標資料表。
- `RollbackSegmentHistoryListLength` – 此 Amazon CloudWatch 指標會衡量記錄具有刪除標記記錄之已遞交交易的復原日誌。這些記錄已排程由 InnoDB 清除操作處理。指標 `trx_rseg_history_len` 具有與 `RollbackSegmentHistoryListLength` 相同的值。
- `PurgeBoundary` – 允許 InnoDB 清除的交易編號上限。如果此 CloudWatch 指標長時間沒有推進，則表示 InnoDB 清除被長時間執行的交易封鎖。若要調查，請檢查 Aurora MySQL 資料庫叢集上的作用中交易。此指標僅適用於 Aurora MySQL 2.11 版及更高版本，以及 3.08 版及更高版本。

- `PurgeFinishedPoint` – 執行 InnoDB 清除的交易編號上限。此 CloudWatch 指標可協助您檢查進行 InnoDB 清除的速度。此指標僅適用於 Aurora MySQL 2.11 版及更高版本，以及 3.08 版及更高版本。
- `TransactionAgeMaximum` – 最舊作用中執行交易的存留期。此 CloudWatch 指標僅適用於 Aurora MySQL 3.08 版及更高版本。
- `TruncateFinishedPoint` – 執行復原截斷的交易編號上限。此 CloudWatch 指標僅適用於 Aurora MySQL 2.11 版及更高版本，以及 3.08 版及更高版本。

如需 CloudWatch 指標的相關資訊，請參閱 Amazon Aurora 的執行個體層級指標。

資料庫正在磁碟上建立暫存資料表

您最近的磁碟上暫存資料表用量大幅增加，已達###。資料庫現在每秒建立約##份暫存資料表。這可能會影響 *db-instance* 的效能並增加磁碟操作。

主題

- 支援的引擎版本
- Context
- 造成此問題的可能原因
- 動作
- 相關指標

支援的引擎版本

所有版本的 Aurora MySQL 都支援此洞察資訊。

Context

有時候，MySQL 伺服器需要在處理查詢時建立內部暫存資料表。Aurora MySQL 可以在記憶體中保存內部暫存資料表。可以由 Temptable 或 MEMORY 儲存引擎處理，或由 InnoDB 儲存在磁碟上。如需詳細資訊，請參閱《MySQL 參考手冊》中的 MySQL 中的內部暫存資料表使用。

造成此問題的可能原因

磁碟上暫存資料表數量增加，表示有使用複雜的查詢。若所設記憶體不足以將暫存資料表儲存在記憶體， Aurora MySQL 會在磁碟上建立資料表。這可能會影響效能並增加磁碟操作。

動作

根據洞察的原因，我們會建議不同的動作。

- 針對 Aurora MySQL 第 3 版，建議您使用 TempTable 儲存引擎。
- 只選取必要欄，以最佳化查詢，傳回較少的資料。

  若您在啟用並計時所有 `statement` 工具的情況下開啟效能結構描述，您可以查詢 `SYS.statements_with_temp_tables`，以擷取使用暫存資料表的查詢清單。如需詳細資訊，請參閱 MySQL 文件中的使用 sys 結構描述的先決條件。
- 考慮涉及排序和分組操作的索引欄。
- 重寫查詢以避免 `BLOB` 和 `TEXT` 欄。這些欄始終使用磁碟。
- 調校下列資料庫參數：`tmp_table_size` 和 `max_heap_table_size`。

  這些參數的預設值為 16 MiB。針對記憶體內暫存資料表使用 MEMORY 儲存引擎時，大小上限由 `tmp_table_size` 或 `max_heap_table_size` 值定義，以較小值為準。當達到此大小上限時，MySQL 會自動將記憶體內的內部暫存資料表轉換為 InnoDB 磁碟上內部暫存資料表。如需詳細資訊，請參閱在 Amazon RDS for MySQL 和 Amazon Aurora MySQL 上使用 TempTable 儲存引擎。

  > **Note**
  >
  > 使用 CREATE TABLE 明確建立 MEMORY 資料表時，只有 `max_heap_table_size` 變數會決定資料表的大小。也沒有轉換為磁碟上的格式。

相關指標

下列績效詳情指標與此洞察相關：

- Created_tmp_disk_tables
- Created_tmp_tables

如需詳細資訊，請參閱 MySQL 文件中的 RCreated_tmp_disk_tables。

Amazon Aurora MySQL 的平行查詢

此主題說明 Amazon Aurora MySQL 相容版本的平行查詢效能最佳化。此功能使用特定資料密集查詢的特殊處理路徑，同時善用 Aurora 共用儲存架構。平行查詢最適用於其資料表具有數百萬資料列的 Aurora MySQL 資料庫叢集，以及需要數分鐘或數小時來完成的分析查詢。

主題

- Aurora MySQL 平行查詢的概觀
- 在 Aurora MySQL 中建立平行查詢資料庫叢集
- 在 Aurora MySQL 中開啟和關閉平行查詢。
- 在 Aurora MySQL 中最佳化平行查詢
- 確認哪些陳述式使用平行查詢
- 監控 Aurora MySQL 的平行查詢
- Aurora MySQL 中平行查詢的 SQL 建構

Aurora MySQL 平行查詢的概觀

Aurora MySQL 平行查詢是一種最佳化操作，它可以將處理資料密集查詢時牽涉到的一些輸入/輸出和運算平行化。平行化的工作包括從儲存體擷取資料列、擷取資料行值，以及判斷哪些資料列符合 `WHERE` 子句和 join 子句中的條件。這類資料密集的工作會被委派 (資料庫最佳化術語叫做下推) 給 Aurora 分散式儲存層中的多個節點。少了平行查詢，每個查詢會將所有掃描到的資料帶到 Aurora MySQL 叢集 (前端節點) 中的單一節點，然後在那裡執行所有的查詢處理。

> **Tip**
>
> PostgreSQL 資料庫引擎也有一個功能稱為「平行查詢」。該功能與 Aurora 平行查詢無關。

開啟平行查詢功能時，Aurora MySQL 引擎會自動判斷查詢何時可以受益，而不需要如提示或表格屬性之類的 SQL 變更。下列各節說明平行查詢何時適用於查詢。您也可以了解如何確保將平行查詢應用至效益最高之處。

> **Note**
>
> 平行查詢最佳化可為需要幾分鐘或幾小時才能完成的長時間執行的查詢提供最大效益。Aurora MySQL 通常不會對廉價查詢最佳化平行查詢。如果另一種最佳化技術更有意義，例如查詢快

取、緩衝集區快取或索引查詢，則通常不會執行平行查詢最佳化。如果發現系統未如您預期般使用平行查詢，請參閱 確認哪些陳述式使用平行查詢。

主題

- 優勢
- 架構
- 先決條件
- 限制
- 平行查詢的 I/O 成本

優勢

使用平行查詢，您可以在 Aurora MySQL 資料表上執行資料密集的分析查詢。在很多情況下，與傳統的查詢處理分工相比，可以獲得數量級的效能提升。

平行查詢的優點如下：

- 提升輸入/輸出效能，原因是可跨多個儲存節點將實體讀取請求平行化。
- 減少網路流量。Aurora 不會將整個資料頁面從儲存節點傳輸至前端節點，然後篩選掉不需要的資料列和資料欄。反之，Aurora 會傳輸精簡的 Tuple，其中僅包含結果集所需的資料欄值。
- 減少前端節點上的 CPU 使用量，原因是下推 `WHERE` 子句的函數處理，資料列篩選和資料欄投影。
- 減少緩衝集區的記憶體壓力。平行查詢處理的網頁不會新增至緩衝集區。此方法可減少資料密集型掃描從緩衝集區中收回常用資料的機會。
- 潛在減少擷取轉換負載 (ETL) 管線中的資料複製，方法為使得在現有資料上執行長時間執行的分析查詢變成可行。

架構

平行查詢功能使用 Aurora MySQL 的主要架構原理：將資料庫引擎與儲存子系統分離，以及簡化通訊協定來減少網路流量。Aurora MySQL 使用這些技術來加速寫入密集操作，例如重做日誌處理。平行查詢可將相同原理套用至讀取操作。

> **Note**
>
> Aurora MySQL 平行查詢的架構不同於其他資料庫系統中名稱相似功能的架構。Aurora MySQL 平行查詢不涉及對稱式多工處理 (SMP)，因此不依賴資料庫伺服器的 CPU 容量。平行處理發生於儲存層，與充當查詢協調器的 Aurora MySQL 伺服器無關。

依預設，若沒有平行查詢，Aurora 查詢的處理會涉及將原始資料傳輸至 Aurora 叢集內的單一節點 (前端節點)。接著 Aurora 會在該單一節點上的單一執行緒中，針對該查詢執行所有進一步處理。使用平行查詢時，這類輸入/輸出密集的工作大部分會被委派給儲存層中的節點。只有結果集的精簡資料列會傳回到前端節點，而已篩選的資料列，以及已擷取和轉換的資料欄值也會一起傳回。效能優點來自網路流量減少、前端節點上 CPU 使用量減少，以及跨儲存節點將輸入/輸出平行化。平行輸入/輸出、篩選和投影的數量與執行查詢之 Aurora 叢集中的資料庫執行個體數量無關。

先決條件

使用並行查詢的所有功能需要執行 2.09 或更新版本的 Aurora MySQL 資料庫叢集。如果您已經有要與平行查詢搭配使用的叢集，您可以將其升級為相容版本，並在之後開啟平行查詢。在這種情況下，請務必遵循 平行查詢的升級考量 中的升級程序，因為這些較新版本中的組態設定名稱和預設值不同。

叢集中的資料庫執行個體必須使用 `db.r*` 執行個體類別。

請確定您的叢集已開啟雜湊聯結最佳化。若要瞭解如何操作，請參閱開啟平行查詢叢集的雜湊聯結。

若要自訂參數 (例如 `aurora_parallel_query` 和) `aurora_disable_hash_join`，您必須擁有與叢集搭配使用的自訂參數群組。您可以使用資料庫參數群組，為每個資料庫執行個體個別指定這些參數。不過，我們建議您在資料庫叢集參數群組中指定它們。如此一來，叢集中的所有資料庫執行個體都會繼承這些參數的相同設定。

限制

下列限制適用於平行查詢功能：

- Aurora I/O-Optimized 資料庫叢集儲存組態不支援平行查詢。
- 您不能將平行查詢與 db.t2 或 db.t3 執行個體類別一起使用。即使您使用 `aurora_pq_force` 工作階段變數要求平行查詢，此限制也適用。
- 平行查詢不適用於使用 `COMPRESSED` 或 `REDUNDANT` 資料列格式的資料表。針對您計劃搭配平行查詢使用的資料表，請使用 `COMPACT` 或 `DYNAMIC` 資料列格式。

- Aurora 會使用成本型演算法來判斷是否要針對每個 SQL 陳述式使用平行查詢機制。在陳述式中使用某些 SQL 建構可以防止平行查詢，或使該陳述式不太可能進行平行查詢。如需 SQL 建構與平行查詢相容性的相關資訊，請參閱 Aurora MySQL 中平行查詢的 SQL 建構。
- 每個 Aurora 資料庫執行個體一次只能執行特定數目的平行查詢工作階段。如果一個查詢具有多個使用平行查詢的部分，例如子查詢、聯結或 UNION 運算子，則這些階段會依序執行。任何時候陳述式只會視為單一平行查詢工作階段。您可以使用平行查詢狀態變數，監控作用中工作階段的數目。您可以查詢狀態變數 `Aurora_pq_max_concurrent_requests`，檢查指定之資料庫執行個體的並行工作階段限制。
- 平行查詢可用於 Aurora 支援的所有 AWS 區域。對於大多數 AWS 區域，使用平行查詢的 Aurora MySQL 最低要求版本為 2.09。
- 平行查詢是為了改善資料密集型查詢的效能而設計。並非為輕量級查詢而設計。
- 建議您針對 SELECT 陳述式使用讀取器節點，尤其是資料密集型陳述式。

平行查詢的 I/O 成本

如果您的 Aurora MySQL 叢集使用平行查詢，您可能會看到 `VolumeReadIOPS` 值增加。平行查詢不會使用緩衝集區。因此，雖然查詢速度很快，但這種最佳化處理可能會導致讀取操作和相關費用增加。

您查詢的平行查詢 I/O 成本是在儲存層進行計量，並且在平行查詢開啟的情況下相同或更大。您得到的好處是查詢效能的改善。使用平行查詢可能會導致 I/O 成本較高的原因有兩個：

- 即使資料表中的某些資料位於緩衝集區中，平行查詢仍需要在儲存層掃描所有資料，因而產生 I/O 成本。
- 執行平行查詢不會對緩衝集區進行暖機。因此，連續執行相同的平行查詢會產生完整的 I/O 成本。

在 Aurora MySQL 中建立平行查詢資料庫叢集

若要建立一個具有平行查詢的 Aurora MySQL 叢集，請在其中加入新增的執行個體，或執行其他管理操作。您可以使用您搭配其他 Aurora MySQL 叢集執行的相同 AWS 管理主控台 和 AWS CLI 技術。您可以建立新叢集來使用平行查詢。您也可以建立資料庫叢集來使用平行查詢，方法為從 MySQL 相容 Aurora 資料庫叢集的快照還原。如果不熟悉建立新 Aurora MySQL 叢集的程序，您可以在建立 Amazon Aurora 資料庫叢集尋找背景資訊和必要條件。

當您選擇 Aurora MySQL 引擎版本時，建議選擇最新可用版本。目前，所有可用的 Aurora MySQL 版本皆支援平行查詢。如果您使用最新版本，則可以更靈活地打開和關閉平行查詢，或者對現有叢集使用平行查詢。

無論您建立新叢集，還是從快照還原，都會使用您搭配其他 Aurora MySQL 叢集執行的相同技術來新增資料庫執行個體。

您可以使用 Amazon RDS 主控台或 AWS CLI 建立平行查詢叢集。

內容

- 使用主控台建立平行查詢叢集
- 使用 CLI 建立平行查詢叢集

使用主控台建立平行查詢叢集

您可以使用主控台來建立新的平行查詢叢集，如下所述。

使用 AWS 管理主控台建立平行查詢叢集

1. 遵循AWS 管理主控台中的一般 建立 Amazon Aurora 資料庫叢集程序。
2. 對於引擎類型，選擇 [Aurora MySQL]。
3. 對於其他組態，請選擇您為資料庫叢集參數群組建立的參數群組。對於 Aurora MySQL 2.09 或更新版本，需要使用此類自訂參數群組。在資料庫叢集參數群組中，指定參數設定 `aurora_parallel_query=ON` 和 `aurora_disable_hash_join=OFF`。這樣做會開啟叢集的平行查詢，並開啟與平行查詢結合運作的雜湊聯結最佳化。

驗證新叢集是否可以使用平行查詢

1. 使用上述技術建立叢集。
2. (若為 Aurora MySQL 第 2 版或第 3 版) 檢查 `aurora_parallel_query` 組態設定為 True。

```
mysql> select @@aurora_parallel_query;
+-------------------------+
| @@aurora_parallel_query |
+-------------------------+
|                       1 |
+-------------------------+
```

3. (若為 Aurora MySQL 第 2 版) 檢查 `aurora_disable_hash_join` 設定是否為 false。

```
mysql> select @@aurora_disable_hash_join;
+---------------------------+
```

```
| @@aurora_disable_hash_join |
+----------------------------+
|                          0 |
+----------------------------+
```

4. 使用某些大型資料表和資料密集型查詢時，請檢查查詢計畫，以確認某些查詢正在使用平行查詢最佳化。若要執行此作業，請依照 確認哪些陳述式使用平行查詢 中的程序進行。

使用 CLI 建立平行查詢叢集

您可以使用 CLI 來建立新的平行查詢叢集，如下所述。

使用 AWS CLI建立平行查詢叢集

1. (選用) 檢查哪些 Aurora MySQL 版本與平行查詢叢集相容。若要執行這項操作，請使用 `describe-db-engine-versions` 指令並檢查 `SupportsParallelQuery` 欄位的值。如需範例，請參閱 檢查平行查詢的 Aurora MySQL 版本相容性。
2. (選用) 建立具有設定 `aurora_parallel_query=ON` 和 `aurora_disable_hash_join=OFF` 的自訂資料庫叢集參數群組。使用如下命令。

```
aws rds create-db-cluster-parameter-group --db-parameter-group-family aurora-
mysql8.0 --db-cluster-parameter-group-name pq-enabled-80-compatible
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-
enabled-80-compatible \
  --parameters
 ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-
enabled-80-compatible \
  --parameters
 ParameterName=aurora_disable_hash_join,ParameterValue=OFF,ApplyMethod=pending-
reboot
```

   如果您執行此步驟，請在後續 `--db-cluster-parameter-group-name` *`my_cluster_parameter_group`* 陳述式中指定選項 `create-db-cluster`。替代您自己的參數群組名稱。如果省略此步驟，請建立參數群組，並在稍後將其與叢集產生關聯，如 在 Aurora MySQL 中開啟和關閉平行查詢。 中所述。
3. 遵循AWS CLI中的一般 建立 Amazon Aurora 資料庫叢集程序。
4. 指定以下一組選項：

- 對於 --engine選項，使用 aurora-mysql。這些值會產生與 MySQL 5.7 或 8.0 分別相容的平行查詢叢集。
- 針對 --db-cluster-parameter-group-name 此選項，指定您建立的 DB 叢集參數群組的名稱，並指定參數值 aurora_parallel_query=ON。如果省略此選項，您可以使用預設參數群組建立叢集，並在稍後加以修改來使用此類自訂參數群組。
- 對於 --engine-version 選項，請使用與平行查詢相容的 Aurora MySQL 版本。如有必要，請使用 在 Aurora MySQL 中最佳化平行查詢 中的程序來取得版本清單。

  以下程式碼顯示做法。將您自己的值替換為每個環境變數，例如 *$ Cluster_ID*。此範例也會指定 --manage-master-user-password 選項來產生主要使用者密碼，並在 Secrets Manager 中管理該密碼。如需更多詳細資訊，請參閱 使用 Amazon Aurora 和 進行密碼管理 AWS Secrets Manager。或者，您可以使用 --master-password 選項，自行指定和管理密碼。

```
aws rds create-db-cluster --db-cluster-identifier $CLUSTER_ID \
  --engine aurora-mysql --engine-version 8.0.mysql_aurora.3.04.1 \
  --master-username $MASTER_USER_ID --manage-master-user-password \
  --db-cluster-parameter-group-name $CUSTOM_CLUSTER_PARAM_GROUP

aws rds create-db-instance --db-instance-identifier ${INSTANCE_ID}-1 \
  --engine same_value_as_in_create_cluster_command \
  --db-cluster-identifier $CLUSTER_ID --db-instance-class $INSTANCE_CLASS
```

5. 驗證您已建立或還原的叢集是否具有可用的平行查詢功能。

   檢查 aurora_parallel_query 組態設定是否存在。如果此設定值為 1，則平行查詢已準備好供您使用。如果此設定值為 0，請先將其設定為 1，才能使用平行查詢。無論哪種方式，叢集都能夠執行平行查詢。

```
mysql> select @@aurora_parallel_query;
+------------------------+
| @@aurora_parallel_query|
+------------------------+
|                      1 |
+------------------------+
```

使用 AWS CLI將快照還原至平行查詢叢集

1. 檢查哪些 Aurora MySQL 版本與平行查詢叢集相容。若要執行這項操作，請使用 `describe-db-engine-versions` 指令並檢查 `SupportsParallelQuery` 欄位的值。如需範例，請參閱 檢查平行查詢的 Aurora MySQL 版本相容性。決定要用於還原叢集的版本。
2. 找出 Aurora MySQL 相容的叢集快照。
3. 遵循AWS CLI中的一般 從資料庫叢集快照還原程序。

```
aws rds restore-db-cluster-from-snapshot \
  --db-cluster-identifier mynewdbcluster \
  --snapshot-identifier mydbclustersnapshot \
  --engine aurora-mysql
```

4. 驗證您已建立或還原的叢集是否具有可用的平行查詢功能。請使用與 使用 CLI 建立平行查詢叢集 中相同的驗證程序。

在 Aurora MySQL 中開啟和關閉平行查詢。

開啟平行查詢時，Aurora MySQL 會判定是否要在執行時針對每個查詢使用它。在聯結、聯集、子查詢等等的情況下，Aurora MySQL 會判定是否要在執行時針對每個查詢區塊使用平行查詢。如需詳細資訊，請參閱 確認哪些陳述式使用平行查詢 和 Aurora MySQL 中平行查詢的 SQL 建構。

您可以使用 aurora_parallel_query 選項，同時在資料庫執行個體的全域和工作階段層級動態開啟和關閉平行查詢。您可以變更資料庫叢集群組中的 `aurora_parallel_query` 設定，依預設開啟或關閉平行查詢。

```
mysql> select @@aurora_parallel_query;
+------------------------+
| @@aurora_parallel_query|
+------------------------+
|                      1 |
+------------------------+
```

若要在工作階段層級切換 `aurora_parallel_query` 參數，請使用標準方法來變更用戶端組態設定。例如，您可以透過 `mysql` 命令列或 JDBC 或 ODBC 應用程式中執行此操作。例如，標準 MySQL 用戶端上的命令為 `set session aurora_parallel_query = {'ON'/'OFF'}`。您也可以將工作階段層級參數新增至 JDBC 組態，或在您的應用程式碼內新增此參數，來動態開啟或關閉平行查詢。

您可以針對特定資料庫執行個體或整個叢集，永久變更 aurora_parallel_query 參數的設定。如果您在資料庫參數群組中指定參數值，則該值僅適用於叢集中特定資料庫執行個體。如果您在資料庫叢集參數群組中指定參數值，則叢集中所有資料庫執行個體都會繼承相同的設定。若要在叢集層級切換 aurora_parallel_query 參數，請使用適合使用參數群組的技術，如Amazon Aurora 的參數群組中所述。請遵循下列步驟：

1. 建立自訂叢集參數群組 (建議使用) 或自訂資料庫參數群組。
2. 在這個參數群組中，將 parallel_query 更新為您想要的值。
3. 視您建立的是資料庫叢集參數群組或資料庫參數群組而定，將參數群組附加到 Aurora 叢集或您計劃使用平行查詢功能的特定資料庫執行個體。

> **Tip**
>
> 由於 aurora_parallel_query 是動態參數，因此您毋需在變更此設定後重新啟動叢集。但是，在切換選項之前使用平行查詢的所有連線皆將繼續執行此作業，直至連線關閉或執行個體重新啟動為止。

您可以使用 ModifyDBClusterParameterGroup 或 ModifyDBParameterGroup API 操作或 AWS 管理主控台修改平行查詢參數。

您可以為平行查詢叢集開啟雜湊聯結、使用 Amazon RDS 主控台或 開啟和關閉平行查詢 AWS CLI，以及覆寫平行查詢最佳化工具。

內容

- 開啟平行查詢叢集的雜湊聯結
- 使用主控台開啟和關閉平行查詢
- 使用 CLI 開啟和關閉平行查詢
- 覆寫平行查詢最佳化工具

開啟平行查詢叢集的雜湊聯結

受益於雜湊聯結最佳化的資源密集型查詢，通常使用平行查詢。因此，確保您計劃使用平行查詢的叢集開啟雜湊聯結能提供幫助。如需如何有效使用雜湊聯結的相關資訊，請參閱使用雜湊聯結，將大型 Aurora MySQL 聯結查詢最佳化。

使用主控台開啟和關閉平行查詢

您可以在資料庫執行個體層級使用參數群組，來開啟或關閉平行查詢。

使用 開啟或關閉資料庫叢集的平行查詢 AWS 管理主控台

1. 建立自訂參數群組，如Amazon Aurora 的參數群組中所述。
2. 將 aurora_parallel_query 更新為 1 (開啟) 或 0 (關閉)。對於可以使用平行查詢功能的叢集，依預設會關閉 aurora_parallel_query。
3. 如果您使用自訂叢集參數群組，請將它附加到您計劃使用平行查詢功能的 Aurora 資料庫叢集。如果您使用自訂資料庫參數群組，請將其連線到叢集中的一或多個資料庫執行個體。建議您使用叢集參數群組。這樣做可確保叢集中的所有資料庫執行個體都具有相同的平行查詢設定，以及相關聯的功能 (例如雜湊聯結)。

使用 CLI 開啟和關閉平行查詢

您可以使用 `modify-db-cluster-parameter-group` 或 `modify-db-parameter-group` 命令，來修改平行查詢參數。根據您是透過資料庫叢集參數群組或資料庫參數群組指定 `aurora_parallel_query` 的值，選擇適當的命令。

使用 CLI 開啟或關閉資料庫叢集的平行查詢

- 使用 `modify-db-cluster-parameter-group` 命令，來修改平行查詢參數。使用如下命令。替代您自己的自訂參數群組的適當名稱。將 `ON` 選項的 `OFF` 部分替代為 `ParameterValue` 或 `--parameters`。

```
$ aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-
name cluster_param_group_name \
  --parameters
 ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "cluster_param_group_name"
}

aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-
name cluster_param_group_name \
  --parameters ParameterName=aurora_pq,ParameterValue=ON,ApplyMethod=pending-reboot
```

您也可以在工作階段層級，例如透過 mysql 命令列或在 JDBC 或 ODBC 應用程式內開啟或關閉平行查詢。若要這樣做，請使用標準方法來變更用戶端組態設定。例如，對於 Aurora MySQL 標準 MySQL 用戶端上的命令為 `set session aurora_parallel_query = {'ON'/'OFF'}`。

您也可以將工作階段層級參數新增至 JDBC 組態，或在您的應用程式碼內新增此參數，來動態開啟或關閉平行查詢。

覆寫平行查詢最佳化工具

您可以使用 aurora_pq_force 工作階段變數來覆寫平行查詢最佳化工具，並針對每個查詢要求平行查詢。我們建議您僅針對測試目的這樣做。下列範例顯示如何在工作階段中使用 aurora_pq_force。

```
set SESSION aurora_parallel_query = ON;
set SESSION aurora_pq_force = ON;
```

若要關閉覆寫，請執行下列命令：

```
set SESSION aurora_pq_force = OFF;
```

在 Aurora MySQL 中最佳化平行查詢

若要最佳化資料庫叢集以進行平行查詢，請考慮哪些資料庫叢集將受益於平行查詢，以及是否要針對平行查詢進行升級。然後，調校工作負載並建立平行查詢的結構描述物件。

內容

- 規劃平行查詢叢集
  - 檢查平行查詢的 Aurora MySQL 版本相容性
- 平行查詢的升級考量
  - 將平行查詢叢集升級至 Aurora MySQL 第 3 版
  - 升級至 Aurora MySQL 2.09 及更新版本
- 平行查詢的效能調校
- 建立結構描述物件以充分利用平行查詢

規劃平行查詢叢集

規劃已開啟平行查詢的資料庫叢集需要做出一些選擇。其中包括執行安裝步驟 (建立或還原完整 Aurora MySQL 叢集)，以及決定在資料庫叢集間開啟平行查詢的廣泛程度。

請在規劃中考慮下列項目：

- 如果您使用與 MySQL 5.7 相容的 Aurora MySQL，則必須建立佈建叢集。然後，您可以使用 `aurora_parallel_query` 參數開啟平行查詢。

  如果您有現有 Aurora MySQL 叢集，則不需要建立新叢集即可使用平行查詢。您可以將叢集或叢集中的特定資料庫執行個體，與已開啟 `aurora_parallel_query` 參數的參數群組建立關聯。如此一來，您可以減少設定相關資料來搭配平行查詢使用的時間和精力。
- 規劃您需要重新組織的任何大型資料表，以便存取它們時可以使用平行查詢。您可能需要建立一些能發揮平行查詢效用的新版大型資料表。例如，您可能需要移除全文搜尋索引。如需詳細資訊，請參閱 建立結構描述物件以充分利用平行查詢。

檢查平行查詢的 Aurora MySQL 版本相容性

若要檢查哪些 Aurora MySQL 版本與平行查詢叢集相容，請使用 `describe-db-engine-versions` AWS CLI 命令並檢查 `SupportsParallelQuery` 欄位的值。下列程式碼範例顯示如何查看指定 AWS 區域中平行查詢叢集可用的組合。請務必在單行上指定完整的 `--query` 參數字串。

```
aws rds describe-db-engine-versions --region us-east-1 --engine aurora-mysql \
--query '*[]|[?SupportsParallelQuery == `true`].[EngineVersion]' --output text
```

上述命令會產生類似下列的輸出：輸出可能會因指定 AWS 區域中可用的 Aurora MySQL 版本而有所不同。

```
5.7.mysql_aurora.2.11.1
5.7.mysql_aurora.2.11.2
5.7.mysql_aurora.2.11.3
5.7.mysql_aurora.2.11.4
5.7.mysql_aurora.2.11.5
5.7.mysql_aurora.2.11.6
5.7.mysql_aurora.2.12.0
5.7.mysql_aurora.2.12.1
5.7.mysql_aurora.2.12.2
5.7.mysql_aurora.2.12.3
5.7.mysql_aurora.2.12.4
```

```
8.0.mysql_aurora.3.04.0
8.0.mysql_aurora.3.04.1
8.0.mysql_aurora.3.04.2
8.0.mysql_aurora.3.04.3
8.0.mysql_aurora.3.05.2
8.0.mysql_aurora.3.06.0
8.0.mysql_aurora.3.06.1
8.0.mysql_aurora.3.07.0
8.0.mysql_aurora.3.07.1
```

開始使用平行查詢與叢集之後，您可以監視效能，並移除平行查詢使用的障礙。如需相關說明，請參閱平行查詢的效能調校。

平行查詢的升級考量

視您升級平行查詢叢集時的原始版本和目的地版本而定，您可能會在平行查詢可以最佳化的查詢類型中找到增強功能。您可能也會發現不需要為平行查詢指定特殊的引擎模式參數。下列各節說明當您升級已開啟平行查詢的叢集時的考量。

將平行查詢叢集升級至 Aurora MySQL 第 3 版

從 Aurora MySQL 第 3 版開始，數個 SQL 陳述式、子句和資料類型都有新的或改善的平行查詢支援。從第 3 版之前的版本升級時，請檢查其他查詢是否可以受益於平行查詢最佳化。如需這些平行查詢增強功能的相關資訊，請參閱資料欄資料類型、分割的資料表和彙整函數、GROUP BY 子句和 HAVING 子句。

如果您是從 Aurora MySQL 2.08 或更舊版本升級平行查詢叢集，也應了解如何開啟平行查詢中的變更。若要這樣做，請閱讀升級至 Aurora MySQL 2.09 及更新版本。

在 Aurora MySQL 第 3 版中，預設為開啟雜湊聯結最佳化。不會使用來自舊版的 `aurora_disable_hash_join` 組態選項。

升級至 Aurora MySQL 2.09 及更新版本

在 Aurora MySQL 2.09 及更新版本中，平行查詢適用於佈建的叢集，並且不需要 `parallelquery` 引擎模式參數。因此，您不需要建立新的叢集或從現有的快照還原，即可使用這些版本的平行查詢。您可以使用 升級 Aurora MySQL 資料庫叢集的次要版本或修補程式層級 中所述的升級程序，將叢集升級為此類版本。您可以升級較舊的叢集，無論它是平行查詢叢集還是佈建的叢集。若要減少 Engine version (引擎版本) 選單中的選項數目，您可以選擇 Show versions that support the parallel query feature (顯示支援平行查詢功能)的版本來篩選該選單中的項目。然後選擇 Aurora MySQL 2.09 或更新版本。

將較早的平行查詢叢集升級為 Aurora MySQL 2.09 或更新版本之後，您可以在升級的叢集中開啟平行查詢。在這些版本中，平行查詢預設為關閉，它的啟用程序也不同。雜湊聯結最佳化預設也會關閉，而且必須個別開啟。因此，請確定您在升級之後再次開啟這些設定。如需執行這項操作的指示說明，請參閱 在 Aurora MySQL 中開啟和關閉平行查詢。 和 開啟平行查詢叢集的雜湊聯結。

特別的是，您可以使用組態參數，而不是 aurora_parallel_query=ON 和 aurora_disable_hash_join=OFF 的 aurora_pq_supported 和 aurora_pq 來開啟平行查詢。在較新的 Aurora MySQL 版本中已取代 aurora_pq_supported 和 aurora_pq 參數。

在升級的叢集中，EngineMode 屬性具有 provisioned 值，而不是 parallelquery。若要檢查平行查詢是否適用於指定的引擎版本，現在您檢查 SupportsParallelQuery describe-db-engine-versions 命令輸出中的 AWS CLI 欄位值。在較早的 Aurora MySQL 版本中，您已檢查 parallelquery 清單中是否存在 SupportedEngineModes。

升級至 Aurora MySQL 2.09 或更新版本之後，您可以利用下列功能。這些功能不適用於執行 Aurora MySQL 舊版的平行查詢叢集。

- 績效詳情。如需更多詳細資訊，請參閱 在 Amazon Aurora 上使用績效詳情監控資料庫負載。
- 恢復功能 如需更多詳細資訊，請參閱 恢復 Aurora 資料庫叢集。
- 停止和啟動叢集。如需更多詳細資訊，請參閱 停用和啟動 Amazon Aurora 資料庫叢集。

平行查詢的效能調校

若要使用平行查詢管理工作負載的效能，請確定平行查詢用於此最佳化最有助益的查詢。

若要這樣做，您可以執行下列操作：

- 確保您的最大資料表與平行查詢相容。您可能會變更資料表屬性或重新建立某些資料表，讓這些資料表的查詢可以利用平行查詢最佳化。若要瞭解如何操作，請參閱建立結構描述物件以充分利用平行查詢。
- 監控哪些查詢使用平行查詢。若要瞭解如何操作，請參閱監控 Aurora MySQL 的平行查詢。
- 確認平行查詢是用於資料密集型和長時間執行的查詢，以及適合您工作負載的並行層級。若要瞭解如何操作，請參閱確認哪些陳述式使用平行查詢。
- 微調您的 SQL 程式碼以開啟平行查詢，並套用至您預期的查詢。若要瞭解如何操作，請參閱Aurora MySQL 中平行查詢的 SQL 建構。

建立結構描述物件以充分利用平行查詢

建立或修改計劃用於平行查詢的表格之前，請務必熟悉 先決條件 和 限制 中所述的需求。

因為平行查詢需要資料表才能使用 ROW_FORMAT=Compact 或 ROW_FORMAT=Dynamic 設定，所以請檢查您的 Aurora 組態設定，以找出 INNODB_FILE_FORMAT 組態選項的任何變更。發出 SHOW TABLE STATUS 陳述式，以確認資料庫中所有資料表的資料列格式。

在變更結構描述以開啟平行查詢以使用更多資料表之前，請務必進行測試。您的測試應該確認平行查詢是否會導致這些資料表的效能出現淨增加。此外，請確定平行查詢的結構描述需求符合您的目標。

例如，從 ROW_FORMAT=Compressed 切換至 ROW_FORMAT=Compact 或 ROW_FORMAT=Dynamic 之前，針對原始和新的資料表測試工作負載的效能。此外，考量其他潛在效果，例如資料量增加。

確認哪些陳述式使用平行查詢

在一般操作中，您不需要執行任何特殊動作即可善用平行查詢。在查詢符合平行查詢的必要需求之後，查詢最佳化器會自動決定是否要針對每個特定查詢使用平行查詢。

如果您在開發或測試環境中執行實驗，則可能發現並未使用平行查詢，因為資料表的資料列數目太小或整個資料量太少。資料表的資料也可能完整在緩衝集區中，尤其是您最近建立來執行實驗的資料表。

監控或調整叢集效能時，您需要決定是否要在適當內容中使用平行查詢。您可能需要調整資料庫結構描述、設定、SQL 查詢，甚至是叢集拓撲和應用程式連線設定，才能完善利用此功能。

若要檢查查詢是否使用平行查詢，請執行 EXPLAIN 陳述式，以檢查查詢執行計畫 (又稱為「explain plan (解釋計畫)」)。如需 SQL 陳述式、子句和運算式如何影響平行查詢 EXPLAIN 輸出的範例，請參閱Aurora MySQL 中平行查詢的 SQL 建構。

以下範例示範傳統查詢計畫與平行查詢計劃之間的差異。這個解釋計畫來自 TPC-H 基準測試的查詢 3。本節中的許多查詢範例都是使用來自 TPC-H 資料集的資料表。您可以從 TPC-H 網站取得資料表定義、查詢和產生範例資料的 dbgen 程式。

```
EXPLAIN SELECT l_orderkey,
  sum(l_extendedprice * (1 - l_discount)) AS revenue,
  o_orderdate,
  o_shippriority
FROM customer,
  orders,
  lineitem
WHERE c_mktsegment = 'AUTOMOBILE'
```

```
AND c_custkey = o_custkey
AND l_orderkey = o_orderkey
AND o_orderdate < date '1995-03-13'
AND l_shipdate > date '1995-03-13'
GROUP BY l_orderkey,
  o_orderdate,
  o_shippriority
ORDER BY revenue DESC,
  o_orderdate LIMIT 10;
```

根據預設，查詢可能具有如下所示的計畫。如果查詢計劃中沒有看到雜湊聯結使用，請確定先開啟最佳化。

```
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len |
 ref  | rows     | filtered | Extra                                              |
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
|  1 | SIMPLE      | customer | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL |  1480234 |    10.00 | Using where; Using temporary; Using filesort       |
|  1 | SIMPLE      | orders   | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL | 14875240 |     3.33 | Using where; Using join buffer (Block Nested Loop) |
|  1 | SIMPLE      | lineitem | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL | 59270573 |     3.33 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
```

針對 Aurora MySQL 第 3 版，您可以透過發出以下陳述式在工作階段層級開啟雜湊聯結。

```
SET optimizer_switch='block_nested_loop=on';
```

針對 Aurora MySQL 2.09 及更高版本，您可以將 aurora_disable_hash_join 資料庫參數或資料庫叢集參數設定為 0 (關閉)。若關閉 aurora_disable_hash_join，optimizer_switch 的值將為 hash_join=on。

開啟雜湊聯結之後，請嘗試再次執行 EXPLAIN 陳述式。如需如何有效使用雜湊聯結的相關資訊，請參閱使用雜湊聯結，將大型 Aurora MySQL 聯結查詢最佳化。

開啟雜湊聯結但關閉平行查詢時，查詢具備的計劃可能如下所示；該計劃會使用雜湊聯結，但不會使用平行查詢。

```
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
| id | select_type | table    |...| rows      | Extra
                          |
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
|  1 | SIMPLE      | customer |...|   5798330 | Using where; Using index; Using
 temporary; Using filesort       |
|  1 | SIMPLE      | orders   |...| 154545408 | Using where; Using join buffer (Hash
 Join Outer table orders)   |
|  1 | SIMPLE      | lineitem |...| 606119300 | Using where; Using join buffer (Hash
 Join Outer table lineitem) |
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
```

在開啟平行查詢之後，此解釋計劃中的兩個步驟可以使用平行查詢最佳化，如 Extra 輸出中的 EXPLAIN 資料欄下所示。系統會將這些步驟的輸入/輸出密集和 CPU 密集處理程序下推至儲存層。

```
+----+...
+-----------------------------------------------------------------------------------------------------------
+
| id |...| Extra
                                                    |
+----+...
+-----------------------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using index; Using temporary; Using filesort
                                                    |
|  1 |...| Using where; Using join buffer (Hash Join Outer table orders); Using
 parallel query (4 columns, 1 filters, 1 exprs; 0 extra)   |
|  1 |...| Using where; Using join buffer (Hash Join Outer table lineitem); Using
 parallel query (4 columns, 1 filters, 1 exprs; 0 extra) |
+----+...
+-----------------------------------------------------------------------------------------------------------
+
```

如需如何解譯平行查詢之 EXPLAIN 輸出的相關資訊，以及平行查詢可以套用至 SQL 陳述的哪些部分，請參閱Aurora MySQL 中平行查詢的 SQL 建構。

以下範例輸出顯示在具有冷緩衝集區的 db.r4.2xlarge 執行個體上執行平行查詢的結果。使用平行查詢時，查詢的執行速度會變得相當快。

> **Note**
>
> 因為時機取決於許多環境因素，所以您的結果可能有所不同。一律進行您自己的效能測試，以利用自己的環境、工作負載等等來確認結果。

```
-- Without parallel query
+------------+-------------+-------------+----------------+
| l_orderkey | revenue     | o_orderdate | o_shippriority |
+------------+-------------+-------------+----------------+
|   92511430 | 514726.4896 | 1995-03-06  |              0 |
.
.
|   28840519 | 454748.2485 | 1995-03-08  |              0 |
+------------+-------------+-------------+----------------+
10 rows in set (24 min 49.99 sec)
```

```
-- With parallel query
+------------+-------------+-------------+----------------+
| l_orderkey | revenue     | o_orderdate | o_shippriority |
+------------+-------------+-------------+----------------+
|   92511430 | 514726.4896 | 1995-03-06  |              0 |
.
.
|   28840519 | 454748.2485 | 1995-03-08  |              0 |
+------------+-------------+-------------+----------------+
10 rows in set (1 min 49.91 sec)
```

本節中的許多範例查詢都會使用來自 TPC-H 資料集的資料表，尤其是 PART 資料表，其具有 2 千萬個資料列和下列定義。

```
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| p_partkey     | int(11)       | NO   | PRI | NULL    |       |
| p_name        | varchar(55)   | NO   |     | NULL    |       |
| p_mfgr        | char(25)      | NO   |     | NULL    |       |
| p_brand       | char(10)      | NO   |     | NULL    |       |
| p_type        | varchar(25)   | NO   |     | NULL    |       |
| p_size        | int(11)       | NO   |     | NULL    |       |
| p_container   | char(10)      | NO   |     | NULL    |       |
```

```
| p_retailprice | decimal(15,2) | NO   |     | NULL    |       |
| p_comment     | varchar(23)   | NO   |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
```

試驗您的工作負載，以了解個別 SQL 陳述式是否可以善用平行查詢。然後，使用下列監控技術來協助驗證平行查詢在一段時間內用於真正工作負載的頻率。對於真正的工作負載，會套用並行限制之類的額外因素。

監控 Aurora MySQL 的平行查詢

如果您的 Aurora MySQL 叢集使用平行查詢，您可能會看到 `VolumeReadIOPS` 值增加。平行查詢不會使用緩衝集區。因此，雖然查詢速度很快，但這種最佳化處理可能會導致讀取操作和相關費用增加。

除了在 Amazon RDS 主控台中檢視指標中描述的 Amazon CloudWatch 指標之外，Aurora 還會提供其他全域狀態變數。您可以使用這些全域狀態變數來協助監視平行查詢執行。它們可以讓您深入了解為什麼最佳化程式可能在指定的情況下使用或不使用平行查詢。若要存取這些變數，您可以使用 `SHOW GLOBAL STATUS` 命令。您也可以尋找這些列示在下面的變數。

平行查詢工作階段與資料庫所執行的查詢不一定是一對一映射。例如，假設您的執行計劃中有兩個步驟會用到平行查詢。在此情況下，查詢涉及兩個平行工作階段，而且嘗試請求和成功請求的計數器都會增加 2。

當您發出 `EXPLAIN`陳述式試驟平行查詢時，預期看到指定為「未選擇」的計數器增加，即使查詢實際上並未執行也一樣。在生產中使用平行查詢時，您可以檢查「未選擇」計數器的增加速度是否超過您預期的速度。在這點上，您可以調整，讓平行查詢執行您預期的查詢。若要執行這項操作，您可以變更叢集設定、查詢混合、開啟平行查詢的資料庫執行個體等。

這些計數器是在資料庫執行個體層級進行追蹤。當您連線至不同端點時，可能看到不同指標，因為每個資料庫執行個體都執行自己的一組平行查詢。當讀取器端點連線至每個工作階段的不同資料庫執行個體時，您也可能看到不同指標。

名稱	描述
`Aurora_pq_bytes_returned`	在平行查詢期間已傳輸至前端節點之 Tuple 資料結構的位元組數目。除以 16,384 以針對 `Aurora_pq_pages_pushed_down` 進行比較。

名稱	描述
Aurora_pq_max_concurrent_requests	可以同時在此 Aurora 資料庫執行個體上執行之平行查詢工作階段的數目上限。這是取決於 AWS 資料庫執行個體類別的固定數目。
Aurora_pq_pages_pushed_down	資料頁面的數目 (每個頁面的固定大小為 16 KiB)，在這些資料頁面中平行查詢已避免透過網路將資料傳輸至前端節點。
Aurora_pq_request_attempted	已請求的平行查詢工作階段數目。此值可能代表每個查詢多個工作階段，取決於 SQL 建構，例如子查詢和聯結。
Aurora_pq_request_executed	已成功執行的平行查詢工作階段數目。
Aurora_pq_request_failed	已傳回錯誤至用戶端的平行查詢工作階段數目。在某些情況下，平行查詢的請求可能失敗，例如，因為儲存層中發生問題。在這些情況下，會使用非平行查詢機制來重試失敗的查詢部分。如果重試的查詢也失敗，則錯誤會傳回至用戶端，而且此計數器會遞增。
Aurora_pq_request_in_progress	目前進行中的平行查詢工作階段數目。此數目適用於您已連線的特定 Aurora 資料庫執行個體，但不適用於整個 Aurora 資料庫叢集。若要查看資料庫執行個體是否接近並行限制，請將此值與 Aurora_pq_max_concurrent_requests 比較。
Aurora_pq_request_not_chosen	未選擇平行查詢以滿足查詢的次數。此值是數個其他更精細計數器的總和。EXPLAIN 陳述式可以增加此計數器，即使查詢實際上並未執行。
Aurora_pq_request_not_chosen_below_min_rows	由於資料表中的資料列數而未選擇平行查詢的次數。EXPLAIN 陳述式可以增加此計數器，即使查詢實際上並未執行。

名稱	描述
Aurora_pq_request_not_chosen_column_bit	使用非平行查詢處理路徑，因為投影的資料欄清單中不支援的資料類型的平行查詢要求數目。
Aurora_pq_request_not_chosen_column_geometry	因為 GEOMETRY 資料表具有資料類型的資料行，所以使用非平行查詢處理路徑的平行查詢要求數目。如需移除此限制之 Aurora MySQL 版本的相關資訊，請參閱將平行查詢叢集升級至 Aurora MySQL 第 3 版。
Aurora_pq_request_not_chosen_column_lob	使用非平行查詢處理路徑的平行查詢要求數目，因為資料表具有 LOB 資料類型的 VARCHAR 資料欄，或因宣告長度而儲存在外部的資料欄。如需移除此限制之 Aurora MySQL 版本的相關資訊，請參閱將平行查詢叢集升級至 Aurora MySQL 第 3 版。
Aurora_pq_request_not_chosen_column_virtual	因為資料表包含虛擬資料欄，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_custom_charset	因為資料表具有自訂字元集的資料欄，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_fast_ddl	使用非平行查詢處理路徑的平行查詢要求數目，因為資料表目前正在變更快速的 DDL ALTER 陳述式。
Aurora_pq_request_not_chosen_ few_pages_outside_buffer_pool	即使小於 95% 的資料表資料在緩衝集區中，也未選擇平行查詢的次數，因為沒有足夠的未置於緩衝的資料表資料，讓平行查詢值得執行。
Aurora_pq_request_not_chosen_full_text_index	因為資料表具有全文檢索索引，所以會使用非平行查詢處理路徑的平行查詢要求數目。

名稱	描述
Aurora_pq_request_not_chosen_high_buffer_pool_pct	因為高百分比的資料表資料 (目前，大於 95%) 已在緩衝集區中，所以未選擇平行查詢的次數。在這些情況下，最佳化器判定從緩衝集區讀取資料最有效率。EXPLAIN 陳述式可以增加此計數器，即使查詢實際上並未執行。
Aurora_pq_request_not_chosen_index_hint	因為查詢包含索引提示，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_innodb_table_format	因為資料表使用不支援的 InnoDB 資料列格式，所以會使用非平行查詢處理路徑的平行查詢要求數目。Aurora 平行查詢只適用於 COMPACT、REDUNDANT 和 DYNAMIC 資料列格式。
Aurora_pq_request_not_chosen_long_trx	由於在長時間執行的交易內啟動查詢，而使用非平行查詢處理路徑的平行查詢請求數目。EXPLAIN 陳述式可以增加此計數器，即使查詢實際上並未執行。
Aurora_pq_request_not_chosen_no_where_clause	因為查詢不包含任何 WHERE 子句，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_range_scan	因為查詢在索引上使用範圍掃描，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_row_length_too_long	因為所有資料欄的總合長度太長，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_small_table	由於資料表中的整體大小 (由資料列數和平均資料列長度決定) 而未選擇平行查詢的次數。EXPLAIN 陳述式可以增加此計數器，即使查詢實際上並未執行。

名稱	描述
Aurora_pq_request_not_chosen_temporary_table	因為查詢參考使用不支援 MyISAM 或 memory 資料表類型的暫存資料表，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_tx_isolation	因為查詢使用不支援的交易隔離層級，所以會使用非平行查詢處理路徑的平行查詢要求數目。在讀取器資料庫執行個體上，平行查詢僅適用於 REPEATABLE READ 和 READ COMMITTED 隔離層級。
Aurora_pq_request_not_chosen_update_delete_stmts	因為查詢是 UPDATE 或 DELETE 陳述式的一部分，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_unsupported_access	因為 WHERE 子句不符合平行查詢的條件，所以使用非平行查詢處理路徑的平行查詢請求數目。如果查詢不需要資料密集掃描，或如果查詢是 DELETE 或 UPDATE 陳述式，則會發生此結果。
Aurora_pq_request_not_chosen_unsupported_storage_type	由於 Aurora MySQL 資料庫叢集未使用支援的 Aurora 叢集儲存組態，因此平行查詢數目請求是使用非平行查詢處理路徑。此參數適用於 Aurora MySQL 3.04 版及更新版本。如需更多詳細資訊，請參閱 限制。
Aurora_pq_request_throttled	由於已在特定 Aurora 資料庫執行個體上執行的並行平行查詢數目已達到上限，而未選擇平行查詢的次數。

Aurora MySQL 中平行查詢的 SQL 建構

在下節中，您可以找到更多有關特定 SQL 陳述式為何使用或不使用平行查詢的詳細資訊。本節還會詳細說明 Aurora MySQL 功能與平行查詢互動的方式。這些詳細資訊可協助您診斷使用平行查詢之叢集的效能問題，或了解平行查詢如何適用於您的特定工作負載。

使用平行查詢的決策依賴許多發生在陳述式執行時的因素。因此，平行查詢可能一律、從未或只在特定條件下用於特定查詢。

> **Tip**
>
> 當您以 HTML 檢視這些範例時，您可以使用每個程式碼清單右上角的 Copy (複製) 小工具，複製 SQL 程式碼來自行嘗試。使用 Copy (複製) 小工具可避免複製 `mysql>` 提示和 `->` 接續資料行周圍的額外字元。

主題

- EXPLAIN 陳述式
- WHERE 子句
- 資料定義語言 (DDL)
- 資料欄資料類型
- 分割的資料表
- 彙整函數、GROUP BY 子句和 HAVING 子句
- WHERE 子句中的函數呼叫
- LIMIT 子句
- 比較運算子
- 聯結
- 子查詢
- UNION
- 檢視
- 資料處理語言 (DML) 陳述式
- 交易和鎖定
- B 型樹狀結構索引
- 全文搜尋 (FTS) 索引
- 虛擬資料欄
- 內建快取機制
- 最佳化工具提示
- MyISAM 暫存資料表

EXPLAIN 陳述式

如本節中的範例所示，EXPLAIN 陳述式指出查詢的每個階段目前是否符合平行查詢的資格。該陳述式也會指出查詢的哪些層面可以下推至儲存層。解釋計劃中的最重要項目如下：

- 在 NULL 資料欄中，key 以外的值表示使用索引查詢才能有效地執行查詢，而不是平行查詢。
- rows 資料欄若值不大 (亦即，不是數百萬的值) 代表查詢未存取足夠資料，平行查詢不值得執行，因此不可能進行平行查詢。這代表不太可能進行平行查詢。
- Extra 資料欄顯示是否預期要使用平行查詢。此輸出看起來如以下範例所示。

```
Using parallel query (A columns, B filters, C exprs; D extra)
```

  columns 數字代表查詢區塊中參照多少資料欄。

  filters 數字代表 WHERE 述詞的數目，而這些述詞代表資料欄值與常數的簡單比較。比較可用於等式、不等式或範圍。Aurora 可以最有效地平行化這些類型的述詞。

  exprs 數字代表表達式 (例如函數呼叫、運算子或其他也可以平行化的表達式) 的數目，但是不如篩選條件一樣有效。

  extra 數字代表不能下推且由前端節點執行的表達式數目。

例如，考量以下 EXPLAIN 輸出。

```
mysql> explain select p_name, p_mfgr from part
    -> where p_brand is not null
    -> and upper(p_type) is not null
    -> and round(p_retailprice) is not null;
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type | table |...| rows     | Extra
                                |
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
|  1 | SIMPLE      | part  |...| 20427936 | Using where; Using parallel query (5
 columns, 1 filters, 2 exprs; 0 extra) |
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
```

來自 Extra 資料欄的資訊顯示，從每一個資料列擷取五個資料欄，以評估查詢條件並建構結果集。一個 WHERE 述詞涉及一個篩選條件，亦即，一個直接在 WHERE 子句中測試的資料欄。在此涉及函數呼叫的情況下，兩個 WHERE 子句需要評估更複雜的表達式。0 extra 欄位確認 WHERE 子句中的所有操作都下推至儲存層，做為平行查詢處理的部分。

在未選擇平行查詢的情況下，您通常可以從 EXPLAIN 輸出的其他資料欄推斷原因。例如，rows 值可能太小，或 possible_keys 資料欄可能指出查詢可以使用索引查詢，而不是資料密集掃描。下例示範的查詢中，最佳化程式可以估計查詢將只掃描少量的資料行。它會根據主要金鑰的特性來執行此操作。在此情況下，不需要平行查詢。

```
mysql> explain select count(*) from part where p_partkey between 1 and 100;
+----+-------------+-------+-------+---------------+---------+---------+------+------
+--------------------------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref  | rows |
 Extra                    |
+----+-------------+-------+-------+---------------+---------+---------+------+------
+--------------------------+
|  1 | SIMPLE      | part  | range | PRIMARY       | PRIMARY | 4       | NULL |   99 |
 Using where; Using index |
+----+-------------+-------+-------+---------------+---------+---------+------+------
+--------------------------+
```

顯示是否將使用平行查詢的輸出會將 EXPLAIN 陳述式執行時所有可用的因素納入考量。當查詢實際執行時，如果情況同時變更，則最佳化器可能做出不同的選擇。例如，EXPLAIN 可能報告陳述式將使用平行查詢。但是，當查詢稍後實際執行時，根據那時的條件可能不使用平行查詢。這類條件可以包括同時執行的數個其他平行查詢。它們也可以包括從資料表中刪除的資料列、正在建立的新索引、在開啟的交易中傳遞太多的時間，以此類推。

WHERE 子句

對於使用平行查詢最佳化的查詢，它必須包括一個 WHERE 子句。

平行查詢最佳化可加速多種在 WHERE 子句中使用的表達式：

- 資料欄值與常數的簡單比較，稱為篩選條件。這些比較從下推至儲存層中受益最多。EXPLAIN 輸出中會報告查詢中篩選條件表達式的數目。
- WHERE 子句中其他類型的表達式也會儘可能下推至儲存層。EXPLAIN 輸出中會報告查詢中這類表達式的數目。這些表達式可以是函數呼叫、LIKE 運算子、CASE 表達式等等。
- 平行查詢目前不會下推特定函數和運算子。查詢中這類表達式的數目會報告為 extra 輸出中的 EXPLAIN 計數器。查詢的其餘部分仍可以使用平行查詢。

- 未下推選取清單中的表達式時，包含這類函數的查詢仍可從平行查詢的中繼結果減少網路流量中受益。例如，呼叫選取清單中彙總函數的查詢可從平行查詢中受益，即使未下推彙總函數也一樣。

例如，以下查詢執行完整資料表掃描，並處理 P_BRAND 資料欄的所有值。不過，它不會使用平行查詢，因為查詢未包括任何 WHERE 子句。

```
mysql> explain select count(*), p_brand from part group by p_brand;
+----+-------------+-------+------+---------------+------+---------+------+----------
+---------------------------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows     |
 Extra                           |
+----+-------------+-------+------+---------------+------+---------+------+----------
+---------------------------------+
|  1 | SIMPLE      | part  | ALL  | NULL          | NULL | NULL    | NULL | 20427936 |
 Using temporary; Using filesort |
+----+-------------+-------+------+---------------+------+---------+------+----------
+---------------------------------+
```

反之，以下查詢包括篩選結果的 WHERE 述詞，因此可以套用平行查詢：

```
mysql> explain select count(*), p_brand from part where p_name is not null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
    -> group by p_brand;
+----+...+----------
+-------------------------------------------------------------------------------------------------
+
| id |...| rows     | Extra
                                          |
+----+...+----------
+-------------------------------------------------------------------------------------------------
+
|  1 |...| 20427936 | Using where; Using temporary; Using filesort; Using parallel
 query (5 columns, 1 filters, 2 exprs; 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------------------------
+
```

如果最佳化器預估針對查詢區塊傳回的資料列數目很小，則平行查詢不會用於該查詢區塊。以下範例顯示一種案例，其中主要索引鍵資料欄上的大於運算子套用至數百萬個資料列，因而導致使用平行查詢。預估反向小於測試僅套用至少數資料列，因此不會使用平行查詢。

```
mysql> explain select count(*) from part where p_partkey > 10;
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+

mysql> explain select count(*) from part where p_partkey < 10;
+----+...+------+--------------------------+
| id |...| rows | Extra                    |
+----+...+------+--------------------------+
|  1 |...|    9 | Using where; Using index |
+----+...+------+--------------------------+
```

資料定義語言 (DDL)

在 Aurora MySQL 第 2 版，平行查詢只適用於未擱置任何快速資料定義語言 (DDL) 作業的資料表。在 Aurora MySQL 第 3 版中，您可以同時在資料表上使用並行查詢，做為即時 DDL 作業。

Aurora MySQL 第 3 版中的即時 DDL 取代了 Aurora MySQL 第 2 版中的快速 DDL 功能。如需即時 DDL 的相關資訊，請參閱即時 DDL (Aurora MySQL 第 3 版)。

資料欄資料類型

在 Aurora MySQL 第 3 版中，平行查詢可以使用包含資料欄的資料表，資料類型為 TEXT、BLOB、JSON 和 GEOMETRY。它也可以使用最大宣告長度超過 768 個位元組的 VARCHAR 和 CHAR 資料欄。如果您的查詢參照任何包含此類大型物件類型的資料欄，則擷取它們的額外工作確實會為查詢處理增加一些負荷。在此情況下，檢查查詢是否可以省略對這些資料欄的參照。如果不可以，請執行基準化分析，以確認在平行查詢開啟或關閉的情況下，這類查詢是否更快。

在 Aurora MySQL 第 2 版，平行查詢對於大型物件類型具有以下限制：

- 不支援 TEXT、BLOB、JSON 和 GEOMETRY 資料類型搭配平行查詢使用。查詢若參照這些類型的任何資料欄，則無法使用平行查詢。
- 可變長度資料欄 (VARCHAR 及 CHAR 資料類型) 與平行查詢相容，而宣告的長度最多可為 768 個位元組。查詢若參照以更長長度上限宣告之類型的任何資料欄，則無法使用平行查詢。對於使用多位元組

字元集的資料欄，位元組限制會將字元集的位元組數目上限納入考量。例如，對於字元集 utf8mb4 (字元長度上限為 4 個位元組)，VARCHAR(192) 資料欄與平行查詢相容，但 VARCHAR(193) 資料欄與其不相容。

分割的資料表

在 Aurora MySQL 第 3 版中，您可以使用分割的資料表搭配平行查詢。因為分割的資料表在內部表示為多個較小的資料表，所以在非分割的資料表上使用平行查詢的查詢可能不會在相同的分割資料表上使用平行查詢。Aurora MySQL 會考慮每個分割區是否足夠大到有資格進行平行查詢最佳化，而不是評估整個資料表的大小。當您預期分割資料表上的查詢會使用平行查詢時，若未使用平行查詢，請檢查 Aurora_pq_request_not_chosen_small_table 狀態變數是否增加。

例如，考慮一個使用 PARTITION BY HASH (*column*) PARTITIONS 2 分割的資料表，以及另一個使用 PARTITION BY HASH (*column*) PARTITIONS 10 分割的資料表。在具有兩個分割區的資料表中，這些分割區是具有十個分割區之資料表的五倍大。因此，平行查詢更有可能用於針對分割區較少之資料表的查詢。在下列範例中，資料表 PART_BIG_PARTITIONS 有兩個分割區，而 PART_SMALL_PARTITIONS 有十個分割區。使用相同的資料，平行查詢更有可能用於大型分割區較少的資料表。

```
mysql> explain select count(*), p_brand from part_big_partitions where p_name is not
 null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
 group by p_brand;
+----+-------------+---------------------+------------
+-------------------------------------------------------------------------------------------
+
| id | select_type | table               | partitions | Extra
                                                                                          |
+----+-------------+---------------------+------------
+-------------------------------------------------------------------------------------------
+
|  1 | SIMPLE      | part_big_partitions | p0,p1      | Using where; Using temporary;
 Using parallel query (4 columns, 1 filters, 1 exprs; 0 extra; 1 group-bys, 1 aggrs) |
+----+-------------+---------------------+------------
+-------------------------------------------------------------------------------------------
+

mysql> explain select count(*), p_brand from part_small_partitions where p_name is not
 null
```

```
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
 group by p_brand;
+----+-------------+----------------------+------------------------------
+------------------------------+
| id | select_type | table                | partitions                   | Extra
                   |
+----+-------------+----------------------+------------------------------
+------------------------------+
|  1 | SIMPLE      | part_small_partitions | p0,p1,p2,p3,p4,p5,p6,p7,p8,p9 | Using
 where; Using temporary |
+----+-------------+----------------------+------------------------------
+------------------------------+
```

彙整函數、GROUP BY 子句和 HAVING 子句

涉及彙線函數的查詢通常是平行查詢的理想候選者，因為它們涉及掃描大型資料表內的大量資料列。

在 Aurora MySQL 3 中，平行查詢可以在選取清單和 HAVING 子句中最佳化彙總函數呼叫。

在 Aurora MySQL 3 之前，選取清單或 HAVING 子句中的彙總函數呼叫不會下推至儲存層。不過，平行查詢仍可以利用彙總函數改善這類查詢的效能。其做法是首先從儲存層的原始資料頁面中平行擷取資料欄值。然後，它會以壓縮的 Tuple 格式而不是整個資料頁面，將那些值傳回至前端節點。一如往常，查詢需要至少一個 WHERE 述詞，才能啟動平行查詢。

以下簡單範例說明哪些類型的彙總查詢可從平行查詢受益。其做法是以壓縮格式將中繼結果傳回至前端節點、篩選中繼結果中不相符的資料列，或兩者。

```
mysql> explain select sql_no_cache count(distinct p_brand) from part where p_mfgr =
 'Manufacturer#5';
+----+...+-------------------------------------------------------------------------+
| id |...| Extra                                                                   |
+----+...+-------------------------------------------------------------------------+
|  1 |...| Using where; Using parallel query (2 columns, 1 filters, 0 exprs; 0 extra) |
+----+...+-------------------------------------------------------------------------+

mysql> explain select sql_no_cache p_mfgr from part where p_retailprice > 1000 group by
 p_mfgr having count(*) > 100;
+----+...
+-------------------------------------------------------------------------------------------------------------------------------
+
| id |...| Extra
                                  |
```

```
+----+...
+-------------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using filesort; Using parallel query (3
 columns, 0 filters, 1 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------------
+
```

WHERE 子句中的函數呼叫

Aurora 可將平行查詢最佳化套用至對 WHERE 子句中大部分內建函數的呼叫。平行化這些函數呼叫可從前端節點中卸載一些 CPU 工作。在最早查詢階段期間平行評估述詞函數，可協助 Aurora 將後續階段期間傳輸和處理的資料量降至最低。

目前，平行化不適用於選取清單中的函數呼叫。那些函數是由前端節點評估，即使相同的函數呼叫出現在 WHERE 子句中也一樣。來自相關資料欄的原始值會併入從儲存節點傳回至前端節點的 Tuple 中。前端節點會執行任何轉換 (例如 UPPER、CONCATENATE 等等)，以產生結果集的最終值。

在以下範例中，平行查詢會平行化對 LOWER 的呼叫，因為它出現在 WHERE 子句中。平行查詢不會影響對 SUBSTR 和 UPPER 的呼叫，因為它們會顯示在選取清單中。

```
mysql> explain select sql_no_cache distinct substr(upper(p_name),1,5) from part
    -> where lower(p_name) like '%cornflower%' or lower(p_name) like '%goldenrod%';
+----+...
+-------------------------------------------------------------------------------------------------
+
| id |...| Extra
                |
+----+...
+-------------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using parallel query (2 columns, 0 filters, 1
 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------------
+
```

相同考量適用於其他表達式，例如 CASE 表達式或 LIKE 運算子。例如，以下範例顯示平行查詢評估 CASE 子句中的 LIKE 表達式和 WHERE 運算子。

```
mysql> explain select p_mfgr, p_retailprice from part
```

```
    -> where p_retailprice > case p_mfgr
    ->   when 'Manufacturer#1' then 1000
    ->   when 'Manufacturer#2' then 1200
    ->   else 950
    -> end
    -> and p_name like '%vanilla%'
    -> group by p_retailprice;
+----+...
+-------------------------------------------------------------------------------------------------
+
| id |...| Extra
                                  |
+----+...
+-------------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using filesort; Using parallel query (4
 columns, 0 filters, 2 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------------
+
```

LIMIT 子句

目前，平行查詢不會用於任何包含 LIMIT 子句的查詢區塊。平行查詢仍然可以透過 GROUP 或聯結搭配 ORDER BY 使用早期的查詢階段。

比較運算子

最佳化器會預估要掃描多少資料列來評估比較運算子，並根據該預估來判定是否要使用平行查詢。

以下第一個範例顯示在沒有平行查詢的情況下，可以有效地執行針對主要索引鍵資料欄的等式比較。以下第二個範例顯示針對未檢索資料欄的類似比較需要掃描數百萬個資料列，因此可從平行查詢中受益。

```
mysql> explain select * from part where p_partkey = 10;
+----+...+------+-------+
| id |...| rows | Extra |
+----+...+------+-------+
|  1 |...|    1 | NULL  |
+----+...+------+-------+

mysql> explain select * from part where p_type = 'LARGE BRUSHED BRASS';
+----+...+----------
+-------------------------------------------------------------------------+
```

```
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

相同考量適用於不等於測試，也適用於範圍比較，例如小於、大於或等於，或 BETWEEN。最佳化器會預估要掃描的資料列數目，並根據輸入/輸出的整體數量來判定平行查詢是否值得。

聯結

具有大型資料表的聯結查詢通常涉及可從平行查詢最佳化中受益的資料密集操作。目前不會平行化多個資料表之間的資料欄值比較 (亦即，聯結述詞本身)。不過，平行查詢可以下推其他聯結階段的某些內部處理，例如在雜湊聯結期間建構 Bloom 篩選條件。即使沒有 WHERE 子句，平行查詢也可以套用至聯結查詢。因此，聯結查詢是 WHERE 子句需要使用平行查詢之規則的例外。

聯結處理的每個階段都會進行評估，以檢查它是否符合平行查詢的資格。如果多個階段可以使用平行查詢，則會依序執行這些階段。因此，根據並行限制，每個聯結查詢都會視為單一平行查詢工作階段。

例如，當聯結查詢包括 WHERE 述詞，來篩選其中一個聯結資料表中的資料列時，該篩選選項可以使用平行查詢。另一個範例是假設聯結查詢使用雜湊聯結機制，例如來聯結大型資料表與小型資料表。在此情況下，產生 Bloom 篩選條件資料結構的資料表掃描或許能夠使用平行查詢。

> **Note**
>
> 受益於雜湊聯結最佳化的資源密集型查詢，通常使用平行查詢。開啟雜湊聯結最佳化的方法取決於 Aurora MySQL 版本。如需每個版本的詳細資訊，請參閱開啟平行查詢叢集的雜湊聯結。如需如何有效使用雜湊聯結的相關資訊，請參閱使用雜湊聯結，將大型 Aurora MySQL 聯結查詢最佳化。

```
mysql> explain select count(*) from orders join customer where o_custkey = c_custkey;
+----+...+----------+-------+---------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------------------
+
| id |...| table    | type  | possible_keys | key         |...| rows      | Extra

             |
```

```
+----+...+----------+-------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------------
+
|  1 |...| customer | index | PRIMARY       | c_nationkey |...|  15051972 | Using index

               |
|  1 |...| orders   | ALL   | o_custkey     | NULL        |...| 154545408 | Using join
 buffer (Hash Join Outer table orders); Using parallel query (1 columns, 0 filters, 1
 exprs; 0 extra) |
+----+...+----------+-------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------------
+
```

對於使用巢狀迴圈機制的聯結查詢，最外層的巢狀迴圈區塊可能使用平行查詢。是否使用平行查詢取決於與往常相同的因素，例如 WHERE 子句中是否存在額外的篩選條件。

```
mysql> -- Nested loop join with extra filter conditions can use parallel query.
mysql> explain select count(*) from part, partsupp where p_partkey != ps_partkey and
 p_name is not null and ps_availqty > 0;
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type | table    |...| rows     | Extra
                                  |
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
|  1 | SIMPLE      | part     |...| 20427936 | Using where; Using parallel query (2
 columns, 1 filters, 0 exprs; 0 extra) |
|  1 | SIMPLE      | partsupp |...| 78164450 | Using where; Using join buffer (Block
 Nested Loop)                          |
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
```

子查詢

外部查詢區塊和內部子查詢區塊可能每個使用平行查詢，或者不使用。他們是否會這樣做，需視資料表、WHERE 子句等對每個區塊來說的通常特徵而定。例如，以下查詢會針對子查詢區塊，但不會針對外部區塊使用平行查詢。

```
mysql> explain select count(*) from part where
   --> p_partkey < (select max(p_partkey) from part where p_name like '%vanilla%');
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
```

```
| id | select_type |...| rows     | Extra
                        |
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
|  1 | PRIMARY     |...|     NULL | Impossible WHERE noticed after reading const tables
                        |
|  2 | SUBQUERY    |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
```

目前，相關的子查詢無法使用平行查詢最佳化。

UNION

根據 UNION 每個部分之資料表、WHERE 子句等等的一般特性，UNION 查詢中的每個查詢區塊可以或不可以使用平行查詢。

```
mysql> explain select p_partkey from part where p_name like '%choco_ate%'
    -> union select p_partkey from part where p_name like '%vanil_a%';
+----+----------------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type    |...| rows     | Extra
                           |
+----+----------------+...+----------
+-------------------------------------------------------------------------------+
|  1 | PRIMARY        |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
|  2 | UNION          |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
| NULL | UNION RESULT | <union1,2> |...|     NULL | Using temporary
                         |
+----+--------------+...+----------
+-------------------------------------------------------------------------------+
```

> **Note**
>
> 查詢內的每個 UNION 子句都會循序執行。即使查詢包括多個全都使用平行查詢的階段，它在任何情況下都只會執行單一平行查詢。因此，即使複雜的多階段查詢也會當做 1 計入並行平行查詢的限制之中。

檢視

最佳化器會使用檢視做為使用基礎資料表的更長查詢，來重新撰寫任何查詢。因此，無論資料表參照是檢視還是真正資料表，平行查詢的運作方式都相同。關於是否要在查詢中使用平行查詢，以及下推哪些部分的所有相同考量，都適用於最終重新撰寫的查詢。

例如，以下解釋計劃顯示通常不會使用平行查詢的檢視定義。利用額外的 WHERE 子句來查詢該檢視時，Aurora MySQL 會使用平行查詢。

```
mysql> create view part_view as select * from part;
mysql> explain select count(*) from part_view where p_partkey is not null;
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (1 columns, 0 filters, 0 exprs;
 1 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

資料處理語言 (DML) 陳述式

如果 INSERT 部分符合平行查詢的其他條件，則 SELECT 陳述式可對處理的 SELECT 階段使用平行查詢。

```
mysql> create table part_subset like part;
mysql> explain insert into part_subset select * from part where p_mfgr =
 'Manufacturer#1';
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

> **Note**
>
> 通常，在 INSERT 陳述式之後，新插入之資料列的資料位於緩衝集區中。因此，在插入大量資料列之後，資料表可能無法立即符合平行查詢的資格。稍後，在正常操作期間從緩衝集區移出資料之後，針對資料表的查詢可能開始再次使用平行查詢。

即使 CREATE TABLE AS SELECT 陳述式的 SELECT 部分將符合平行查詢的資格，此陳述式也不會使用平行查詢。此陳述式的 DDL 層面使得它與平行查詢處理不相容。反之，在 INSERT ... SELECT 陳述式中，SELECT 部分可以使用平行查詢。

無論 DELETE子句中資料表和述詞的大小為何，平行查詢決不用於 UPDATE 或 WHERE 陳述式。

```
mysql> explain delete from part where p_name is not null;
+----+-------------+...+----------+-------------+
| id | select_type |...| rows     | Extra       |
+----+-------------+...+----------+-------------+
|  1 | SIMPLE      |...| 20427936 | Using where |
+----+-------------+...+----------+-------------+
```

交易和鎖定

您可以在 Aurora 主要執行個體上使用所有隔離層級。

在 Aurora 讀取器資料庫執行個體上，平行查詢會套用至 REPEATABLE READ 隔離層級下執行的陳述式。Aurora MySQL 版本 2.09 或更新版本也可以在讀取器資料庫執行個體上使用 READ COMMITTED 隔離級別。REPEATABLE READ 是 Aurora 讀取器資料庫執行個體的預設隔離等級。若要在讀取器資料庫執行個體上使用 READ COMMITTED 隔離等級，您需要在工作階段層級設定 aurora_read_replica_read_committed 組態選項。讀取器執行個體的 READ COMMITTED 隔離層級符合 SQL 標準行為。不過，讀取器執行個體上的隔離不太嚴格，但當查詢在寫入器執行個體上使用 READ COMMITTED 隔離層級時更為嚴格。

如需 Aurora 隔離層級的詳細資訊，尤其是 READ COMMITTED 之間的差異，請參閱 Aurora MySQL 隔離層級。

在完成大型交易之後，資料表統計資料可能過時。這類過時的統計資料可能需要 ANALYZE TABLE 陳述式，然後 Aurora 才能精確地預估資料列數目。大規模的 DML 陳述式也可能將資料表資料的很大一部分帶入緩衝集區中。在緩衝集區中具有此資料，可能導致不常針對該資料表選擇平行查詢，直到從集區移出資料。

當您的工作階段是在長時間執行的交易 (預設為 10 分鐘) 內時，該工作階段內的進一步查詢不會使用平行查詢。在單一長時間執行的查詢期間也可能發生逾時。在平行查詢處理開始之前，如果查詢的執行時間超過間隔上限 (目前為 10 分鐘)，則此類型的逾時可能會發生。

您可以在執行臨時 (一次性) 查詢的 autocommit=1 工作階段中設定 mysql，來減少意外啟動長時間執行之交易的機會。針對資料表的 SELECT 陳述式甚至會建立讀取檢視來開始交易。讀取檢視是適用於後續查詢的一致資料集，一直留到確定交易為止。使用 JDBC 或 ODBC 應用程式與 Aurora 搭配時，請注意此限制，因為應用程式可能在 autocommit 設定關閉的情況下執行。

以下範例顯示在 autocommit 設定關閉的情況下，針對資料表執行查詢時，如何建立隱含地開始交易的讀取檢視。之後短暫執行的查詢仍可以使用平行查詢。不過，在暫停數分鐘之後，查詢不再符合平行查詢的資格。利用 COMMIT 或 ROLLBACK 結束交易，可還原平行查詢資格。

```
mysql> set autocommit=0;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+----+...+---------
+-------------------------------------------------------------------------------+
| id |...| rows    | Extra
         |
+----+...+---------
+-------------------------------------------------------------------------------+
|  1 |...| 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+---------
+-------------------------------------------------------------------------------+

mysql> select sleep(720); explain select sql_no_cache count(*) from part where
 p_retailprice > 10.0;
+------------+
| sleep(720) |
+------------+
|          0 |
+------------+
1 row in set (12 min 0.00 sec)

+----+...+---------+-------------+
| id |...| rows    | Extra       |
+----+...+---------+-------------+
|  1 |...| 2976129 | Using where |
+----+...+---------+-------------+
```

```
mysql> commit;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+----+...+---------
+-------------------------------------------------------------------------------+
| id |...| rows    | Extra
         |
+----+...+---------
+-------------------------------------------------------------------------------+
|  1 |...| 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+---------
+-------------------------------------------------------------------------------+
```

若要查看查詢有多少次未符合平行查詢的資格，因為它們是在長時間執行的交易內，請檢查狀態變數 Aurora_pq_request_not_chosen_long_trx。

```
mysql> show global status like '%pq%trx%';
+--------------------------------------+-------+
| Variable_name                        | Value |
+--------------------------------------+-------+
| Aurora_pq_request_not_chosen_long_trx | 4     |
+---------------------------------+-------+
```

任何獲得鎖定的 SELECT 陳述式 (例如 SELECT FOR UPDATE 或 SELECT LOCK IN SHARE MODE 語法) 都無法使用平行查詢。

平行查詢可以用於 LOCK TABLES 陳述式鎖定的資料表。

```
mysql> explain select o_orderpriority, o_shippriority from orders where o_clerk =
 'Clerk#000095055';
+----+...+-----------
+-------------------------------------------------------------------------------+
| id |...| rows      | Extra
           |
+----+...+-----------
+-------------------------------------------------------------------------------+
|  1 |...| 154545408 | Using where; Using parallel query (3 columns, 1 filters, 0
 exprs; 0 extra) |
+----+...+-----------
+-------------------------------------------------------------------------------+
```

```
mysql> explain select o_orderpriority, o_shippriority from orders where o_clerk =
 'Clerk#000095055' for update;
+----+...+-----------+-------------+
| id |...| rows      | Extra       |
+----+...+-----------+-------------+
|  1 |...| 154545408 | Using where |
+----+...+-----------+-------------+
```

B 型樹狀結構索引

ANALYZE TABLE 陳述式收集的統計資料可協助最佳化器根據每個資料欄的資料特性，決定何時使用平行查詢或索引查閱。在對資料表內的資料進行大幅變更的 DML 操作之後，請執行 ANALYZE TABLE 來保持最新的統計資料。

如果索引檢閱可以有效執行查詢，而沒有資料密集掃描，則 Aurora 可能使用索引檢閱。這樣做可以避免平行查詢處理的額外負荷。對於可在任何 Aurora 資料庫叢集上同時執行的平行查詢數目也有並行限制。確定使用最佳實務來製作資料表的索引，讓您最常使用和最高度並行的查詢可以使用索引檢閱。

全文搜尋 (FTS) 索引

目前，平行查詢不會用於包含全文搜尋索引的資料表，無論查詢是參照這類索引資料欄，還是使用 MATCH 運算子。

虛擬資料欄

目前，平行查詢不會用於包含虛擬資料欄的資料表，無論查詢是否參考任何虛擬資料欄。

內建快取機制

Aurora 包括內建快取機制，即緩衝集區和查詢快取。Aurora 最佳化器會在這些快取機制進行選擇，並根據哪一個機制對特定查詢最有效來選擇平行查詢。

當平行查詢篩選資料列，以及轉換並擷取資料欄值時，資料會傳回至前端節點，做為 Tuple 而非做為資料頁面。因此，執行平行查詢不會新增任何頁面至緩衝集區，或移出已在緩衝集區的頁面。

Aurora 會檢查緩衝集區中呈現的資料表頁面數目，以及該數目代表多少比例的資料表資料。Aurora 會使用該資訊來決定，使用平行查詢 (以及略過緩衝集區中的資料) 是否更有效。或者，Aurora 可能使用非平行查詢處理路徑，這會使用緩衝集區中快取的資料。快取哪些頁面和資料密集查詢如何影響快取及移出，取決於與緩衝集區相關的組態設定。因此，難以預測任何查詢是否使用平行查詢，因為選擇取決於緩衝集區內不斷變更的資料。

此外，Aurora 也會對平行查詢強加並行限制。因為不是每個查詢都會使用平行查詢，所以多個查詢同時存取的資料表通常會具有緩衝集區中資料的很大一部分。因此，Aurora 通常不會選擇這些資料表進行平行查詢。

當您在相同的資料表上執行一連串的非平行查詢時，由於資料未在緩衝集區中，第一個查詢可能很慢。然後，第二個與後續查詢會快得多，因為緩衝集區現在完成了「暖機」。從最初針對資料表的第一個查詢開始，平行查詢通常會顯示一致的效能。當進行效能測試時，會同時使用冷和暖緩衝集區，對非平行查詢進行基準測試。在某些情況下，具有暖緩衝集區的結果可以充分地與平行查詢時間進行比較。在這些情況下請考慮因素，例如針對該資料表的查詢頻率。也請考慮是否值得將該資料表的資料保留在緩衝集區中。

當提交相同的查詢，以及基礎資料表的資料未變更時，查詢快取可避免重新執行查詢。由平行查詢功能最佳化的查詢可以移至查詢快取，如此可在重新執行時有效地讓它們成為即時查詢。

> **Note**
>
> 進行效能比較時，查詢快取可以產生人為的低時序數。因此，在類似基準測試的情況中，您可以使用 `sql_no_cache` 提示。此提示可防止從查詢快取提供結果，即使先前已執行相同的查詢也一樣。提示緊跟在查詢中的 `SELECT` 陳述式後面。本主題中的許多平行查詢範例都包括此提示，以使得可在平行查詢開啟和關閉的查詢版本之間進行查詢時間比較。
> 當您移至生產用途的平行查詢時，確定從您的來源中移除此提示。

最佳化工具提示

控制最佳化工具的另一種方式是使用最佳化工具提示，您可以在個別陳述式中指定這些提示。例如，您可以在一個陳述式中為一份資料表開啟最佳化，為另一份資料表關閉最佳化。如需這些提示的詳細資訊，請參閱《MySQL 參考手冊》中的最佳化工具提示。

您可以將 SQL 提示與 Aurora MySQL 查詢搭配使用，以微調效能。您也可以使用提示來防止重要查詢的執行計劃因不可預期的情況而變更。

我們已擴充 SQL 提示功能，協助您控制查詢計畫的最佳化工具選擇。這些提示適用於使用平行查詢最佳化的查詢。如需更多詳細資訊，請參閱 Aurora MySQL 提示。

MyISAM 暫存資料表

平行查詢最佳化僅適用於 InnoDB 資料表。因為 Aurora MySQL 會在幕後針對暫時資料表使用 MyISAM，所以涉及暫時資料表的內部查詢階段永遠不會使用平行查詢。這些查詢階段是由 `Using temporary` 輸出中的 `EXPLAIN` 指出。

使用進階稽核與 Amazon Aurora MySQL 資料庫叢集搭配

您可以使用 Amazon Aurora MySQL 的高效能「進階稽核」功能來稽核資料庫活動。若要這麼做，您需要設定幾個資料庫叢集參數以便收集稽核日誌。進階稽核啟用時，您可將它用來記錄任意組合的支援事件。

您可以檢視或下載稽核日誌，一次檢閱一個資料庫執行個體的稽核資訊。若要執行此操作，您可以使用監控 Amazon Aurora 日誌檔案中說明的程序。

> **Tip**
>
> 如果 Aurora 資料庫叢集包含多個資料庫執行個體，您可能會發現檢查叢集中所有執行個體的稽核日誌更為方便。若要執行此操作，可以使用 CloudWatch Logs。您可以在叢集層級開啟設定，將 Aurora MySQL 稽核日誌資料發佈至 CloudWatch 中的日誌群組。然後您可以透過 CloudWatch 界面檢視、篩選和搜尋稽核日誌。如需更多詳細資訊，請參閱 將 Amazon Aurora MySQL 日誌發佈至 Amazon CloudWatch Logs。

啟用進階稽核

使用本節所述的參數來啟用和設定資料庫叢集的進階稽核。

使用 `server_audit_logging` 參數來啟用或停用進階日誌。

使用 `server_audit_events` 參數來指定要記錄的事件。

使用 `server_audit_incl_users` 和 `server_audit_excl_users` 參數來指定要受到稽核的使用者。預設情況下，所有使用者會受到稽核。如需當一或兩個參數保留空白，或在兩者中指定了相同的使用者名稱時，這些參數如何運作的詳細資訊，請參閱 server_audit_incl_users和 server_audit_excl_users。

在資料庫叢集使用的參數群組中設定這些參數，以設定進階稽核。您可以依照修改 Amazon Aurora 中的資料庫參數群組中的參數所示的程序，使用 AWS 管理主控台修改資料庫叢集參數。您可以使用 modify-db-cluster-parameter-group AWS CLI 命令或 ModifyDBClusterParameterGroup Amazon RDS API 操作，以程式設計方式來修改資料庫叢集參數。

如果參數群組已與您的叢集建立關聯，不需要重新啟動資料庫叢集即可修改這些參數。首次為參數群組與叢集建立關聯時，需要重新啟動叢集。

主題

- server_audit_logging
- server_audit_events
- server_audit_incl_users
- server_audit_excl_users

server_audit_logging

啟用或停用進階稽核。此參數預設為 OFF；設為 ON 即可啟用進階稽核。

日誌中不會顯示任何稽核資料，除非您使用 `server_audit_events` 參數定義了一或多個要稽核的事件類型。

若要確認已為資料庫執行個體記錄稽核資料，請檢查該執行個體的部分日誌檔案的名稱格式是否為 `audit/audit.log.`*`other_identifying_information`*。若要查看日誌檔案的名稱，請依照檢視並列出資料庫日誌檔案中的程序操作。

server_audit_events

包含要記錄之事件的逗號分隔清單。事件必須以全部大寫指定，且清單元素之間不能有空格，例如：`CONNECT,QUERY_DDL`。此參數預設為空字串。

您可以記錄下列事件的任意組合：

- CONNECT – 記錄成功與失敗連線，還有中斷連線。此事件包含使用者資訊。
- QUERY – 以純文字記錄所有查詢，包括因為語法或許可錯誤而失敗的查詢。

> **Tip**
>
> 開啟此事件類型後，稽核資料中會包括 Aurora 自動執行的持續監控的相關資訊和運作狀態檢查資訊。如果您只對特定類型的操作感興趣，可以使用更具體的事件類型。您也可以使用 CloudWatch 界面，在日誌中搜尋與特定資料庫、資料表或使用者相關的事件。

- QUERY_DCL – 類似於 QUERY 事件，但只傳回資料控制語言 (DCL) 查詢 (GRANT、REVOKE 等)。
- QUERY_DDL – 類似於 QUERY 事件，但只傳回資料定義語言 (DDL) 查詢 (CREATE、ALTER 等)。
- QUERY_DML – 類似於 QUERY 事件，但只傳回資料操作語言 (DML) 查詢 (INSERT、UPDATE 和 SELECT 等)。
- TABLE – 記錄因為執行查詢而受影響的資料表。

> **Note**
>
> Aurora 中沒有排除稽核日誌中特定查詢的篩選條件。若要排除 SELECT 查詢，您必須排除所有 DML 陳述式。
> 如果特定使用者在稽核日誌中報告這些內部 SELECT 查詢，則您可以透過設定 server_audit_excl_users 資料庫叢集參數來排除該使用者。不過，如果該使用者也用於其他活動且無法省略，則沒有排除 SELECT 查詢的其他選項。

server_audit_incl_users

包含其活動受到記錄之使用者的使用者名稱之逗號分隔清單。清單元素之間不能有空格，例如：user_3,user_4。此參數預設為空字串。長度上限為 1024 個字元。指定的使用者名稱必須符合 User 資料表的 mysql.user 欄的對應值。如需使用者名稱的詳細資訊，請參閱 MySQL 文件中的帳戶使用者名稱和密碼。

如果 server_audit_incl_users 和 server_audit_excl_users 都空白 (預設值)，則所有使用者都會受到稽核。

如果您將使用者新增至 server_audit_incl_users，並將 server_audit_excl_users 保留空白，則只有那些使用者會受到稽核。

如果您使用者新增至 server_audit_excl_users，並將 server_audit_incl_users 保留空白，則除了 server_audit_excl_users 中列出的使用者以外，其餘所有使用者都會受到稽核。

如果將相同的使用者同時新增至 server_audit_excl_users 和 server_audit_incl_users，則這些使用者會受到稽核。當同一個使用者在兩個設定中都有列出時，server_audit_incl_users 的優先順序較高。

連接和中斷連接事件不受此變數所影響；只要指定就一定會加以記錄。即使 server_audit_excl_users 參數中有指定某個使用者，還是會記錄該使用者，因為 server_audit_incl_users 的優先順序較高。

server_audit_excl_users

包含其活動不受記錄之使用者的使用者名稱之逗號分隔清單。清單元素之間不能有空格，例如：rdsadmin,user_1,user_2。此參數預設為空字串。長度上限為 1024 個字元。指定的使用者名稱必須符合 User 資料表的 mysql.user 欄的對應值。如需使用者名稱的詳細資訊，請參閱 MySQL 文件中的帳戶使用者名稱和密碼。

如果 server_audit_incl_users 和 server_audit_excl_users 都空白 (預設值)，則所有使用者都會受到稽核。

如果您將使用者新增至 server_audit_excl_users，並將 server_audit_incl_users 保留空白，則只有 server_audit_excl_users 中列出的使用者不會受到稽核，其他所有使用者都會受到稽核。

如果將相同的使用者同時新增至 server_audit_excl_users 和 server_audit_incl_users，則這些使用者會受到稽核。當同一個使用者在兩個設定中都有列出時，server_audit_incl_users 的優先順序較高。

連接和中斷連接事件不受此變數所影響；只要指定就一定會加以記錄。如果 server_audit_incl_users 參數中有指定某個使用者，則也會記錄該使用者，因為此設定的優先順序高於 server_audit_excl_users。

檢視稽核日誌

您可以使用主控台來檢視和下載稽核日誌。在 Database (資料庫) 頁面上，選擇資料庫執行個體以顯示其詳細資訊，然後捲動至 Logs (日誌) 區段。進階稽核功能產生的稽核日誌名稱格式為 audit/audit.log.*other_identifying_information*。

若要下載日誌檔案，請在 Logs (日誌) 區段中選取該檔案，然後選擇 Download (下載)。

您也可以使用 describe-db-log-files AWS CLI 命令來取得日誌檔案清單。您可以使用 download-db-log-file-portion AWS CLI 命令來下載日誌檔案的內容。如需更多詳細資訊，請參閱 檢視並列出資料庫日誌檔案 及 下載資料庫日誌檔案。

稽核日誌詳細資訊

日誌檔案以 UTF-8 格式的逗號分隔變數 (CSV) 檔案表示。查詢也會包在單引號 (') 中。

稽核記錄分別存放在每個 Aurora MySQL 資料庫執行個體的本機儲存區中。每個執行個體一次將寫入分發到四個日誌檔案。日誌檔案大小上限為 100 MB。達到此無法設定的限制時，Aurora 會旋轉檔案並產生一個新檔案。

> **Tip**
>
> 日誌檔案項目不會循序排列。若要排序項目，請使用時間戳記值。若要查看最新事件，您可能必須檢閱所有日誌檔案。為了更靈活排序和搜尋日誌資料，請開啟設定，將稽核日誌上傳到 CloudWatch，然後使用 CloudWatch 界面檢視這些日誌。

若要查看包含更多欄位類型和以 JSON 格式輸出的稽核資料，您也可以使用資料庫活動串流功能。如需更多詳細資訊，請參閱 使用資料庫活動串流來監控 Amazon Aurora。

稽核日誌檔案的每一列依指定順序包含以下逗號分隔的資訊：

欄位	描述
timestamp	所記錄事件的 Unix 時間戳記 (精確度達到微秒)。
serverhost	為其記錄事件之執行個體的名稱。
username	使用者的連線使用者名稱。
host	使用者連線來源的主機。
connectionid	所記錄操作的連線 ID 號碼。
queryid	查詢 ID 號碼，可用來尋找關聯式資料表事件和相關的查詢。若為 TABLE 事件，則會新增多行。
operation	記錄的動作類型。可能值為：CONNECT、QUERY、READ、WRITE、CREATE、ALTER、RENAME 及 DROP。
database	由 USE 命令設定的作用中資料庫。
物件	若為 QUERY 事件，此值指出資料庫所執行的查詢。若為 TABLE 事件，則指出資料表名稱。
retcode	所記錄操作的傳回碼。

以 Amazon Aurora MySQL 進行複寫

Aurora MySQL 複寫功能是叢集高可用性和效能的關鍵。Aurora 可讓您輕鬆建立最多可有 15 Aurora 個複本的叢集，或調整其大小。

所有複本都從相同的基礎資料運作。如果部分資料庫執行個體離線，則其他資料庫執行個體仍可繼續用來處理查詢，或在需要時以寫入器的方式接管。Aurora 會自動將您的唯讀連線分散至多個資料庫執行個體，協助 Aurora 叢集以支援查詢密集的工作負載。

在以下主題中，您可以找到 Aurora MySQL 複寫如何運作，以及如何微調複寫設定以取得最佳可用性和效能的相關資訊。

主題

- 使用 Aurora 複本
- Amazon Aurora MySQL 的複寫選項
- Amazon Aurora MySQL 複寫的效能考量
- 使用 Aurora MySQL 設定複寫篩選條件
- 監控 Amazon Aurora MySQL 複寫
- 跨 AWS 區域 複寫 Amazon Aurora MySQL 資料庫叢集
- Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)
- 使用 GTID 式複寫

使用 Aurora 複本

Aurora 複本是 Aurora 資料庫叢集中的獨立端點，最適合用於擴展讀取操作和提高可用性。最多 15 個 Aurora 複本可以分佈在 AWS 區域 內資料庫叢集跨越的可用區域。雖然資料庫叢集磁碟區由資料庫叢集的多個資料副本組成，但叢集磁碟區中的資料是以單一邏輯磁碟區的形式呈現給主要執行個體，以及資料庫叢集內的 Aurora 複本。如需 Aurora 複本的詳細資訊，請參閱 Aurora 複本。

Aurora 複本很適用於讀取擴展，因為完全專用於叢集磁碟區上的讀取操作。寫入操作由主要執行個體管理。因為叢集磁碟區是在 Aurora MySQL 資料庫叢集的所有資料庫執行個體之間共用，所以不需要額外工作，即可複寫每一個 Aurora 複本的資料副本。相反地，MySQL 僅供讀取複本必須在單一執行緒上，從來源資料庫執行個體將所有寫入操作重播至其本機資料存放區。此限制可能導致 MySQL 僅供讀取複本難以支援大量讀取流量。

有了 Aurora MySQL，在刪除 Aurora 複本時，隨即會移除其執行個體端點，並且會從讀取器端點移除 Aurora 複本。如果在要刪除的 Aurora 複本上有陳述式正在執行，則會有三分鐘的寬限期。現有陳述式在寬限期期間可以從容地完成。在寬限期結束時，Aurora 複本會關閉並刪除。

> **Important**
>
> Aurora MySQL 的 Aurora 複本一律會對 InnoDB 資料表上的操作使用 `REPEATABLE READ` 預設交易隔離層級。您僅可以對 Aurora MySQL 資料庫叢集的主要執行個體使用 `SET TRANSACTION ISOLATION LEVEL` 命令來變更交易層級。此限制可避免 Aurora 複本上的使用者層級鎖定，並允許 Aurora 複本擴展來支援上千個作用中使用者的連線，同時將複本延遲保持在最小值。

> **Note**
>
> 主要執行個體上執行的 DDL 陳述式可能會中斷相關聯 Aurora 複本上的資料庫連線。如果 Aurora 複本連線正在主動使用資料庫物件，例如資料表，並且在主要執行個體上使用 DDL 陳述式來修改該物件，則會中斷 Aurora 複本連線。

> **Note**
>
> 中國 (寧夏) 區域不支援跨區域的僅供讀取複本。

Amazon Aurora MySQL 的複寫選項

您可以設定下列任何選項之間的複寫：

- 不同 AWS 區域 中的兩個 Aurora MySQL 資料庫叢集，方法為建立 Aurora MySQL 資料庫叢集的跨區域僅供讀取複本。

  如需更多詳細資訊，請參閱 跨 AWS 區域 複寫 Amazon Aurora MySQL 資料庫叢集。
- 相同 AWS 區域 中的兩個 Aurora MySQL 資料庫叢集，方法為使用 MySQL 二進位日誌 (binlog) 複寫。

  如需更多詳細資訊，請參閱 Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。

- 一個 RDS for MySQL 資料庫執行個體作為來源及一個 Aurora MySQL 資料庫叢集，方法為建立 RDS for MySQL 資料庫執行個體的 Aurora 僅供讀取複本。

  您可以使用此方法，在遷移至 Aurora 期間，將現有和持續的資料變更帶到 Aurora MySQL。如需更多詳細資訊，請參閱 使用 Aurora 讀取複本，從 RDS for MySQL 資料庫執行個體將資料遷移至 Amazon Aurora MySQL 資料庫叢集。

  您還可以使用此方法，來增加資料讀取查詢的可擴展性。您可以使用唯讀 Aurora MySQL 叢集中的一個或多個資料庫執行個體，來查詢資料。如需更多詳細資訊，請參閱 使用 Amazon Aurora 為 MySQL 資料庫擴展讀取。
- 在一個 AWS 區域 中具有 Aurora MySQL 資料庫叢集，以及在不同區域中具有最多五個 Aurora 唯讀 Aurora MySQL 資料庫叢集，方法為建立 Aurora 全域資料庫。

  您可以使用 Aurora 全域資料庫，來支援具有全球覆蓋的應用程式。主要 Aurora MySQL 資料庫叢集，具有寫入器執行個體以及最多 15 個 Aurora 複本。唯讀次要 Aurora MySQL 資料庫叢集，每個叢集最多可包含 16 個 Aurora 複本。如需更多詳細資訊，請參閱 使用 Amazon Aurora 全球資料庫。

> **Note**
>
> 重新啟動 Amazon Aurora 資料庫叢集的主要執行個體時，也會自動重新啟動該資料庫叢集的 Aurora 複本，以重新建立進入點來保證整個資料庫叢集的讀取/寫入一致性。

Amazon Aurora MySQL 複寫的效能考量

下列功能可協助您微調 Aurora MySQL 複寫的效能。

複本日誌壓縮功能會自動減少複寫訊息的網路頻寬。因為每則訊息都會傳輸至所有 Aurora 複本，所以叢集越大，得到的好處就越大。此功能涉及寫入器節點上執行壓縮的一些 CPU 額外負荷。它在 Aurora MySQL 第 2 版和第 3 版中一律為啟用狀態。

binlog 篩選功能會自動減少複寫訊息的網路頻寬。因為 Aurora 複本不會使用複寫訊息中包含的 binlog 資訊，所以傳送至那些節點的訊息中會省略該資料。

在 Aurora MySQL 第 2 版中，您可以變更 `aurora_enable_repl_bin_log_filtering` 參數以控制此功能。此參數預設為開啟。因為此最佳化旨在透明化，所以您可能只在對複寫相關問題進行診斷或故障診斷期間才會關閉此設定。例如，比對無法使用此功能之舊版 Aurora MySQL 叢集的行為。

Binlog 篩選功能在 Aurora MySQL 第 3 版中一律為啟用狀態。

使用 Aurora MySQL 設定複寫篩選條件

您可以使用複寫篩選條件來指定要與僅供讀取複本一起複寫的資料庫和資料表。複寫篩選條件可以包含複寫中的資料庫和資料表，或將其排除在複寫之外。

下列是複寫篩選條件的一些應用案例：

- 要縮小僅供讀取複本的大小。使用複寫篩選，您可以排除僅供讀取複本不需要的資料庫和資料表。
- 基於安全考量，要將資料庫和資料表從僅供讀取複本中排除。
- 為不同僅供讀取複本的特定應用案例複寫不同的資料庫和資料表。例如，您可以使用特定僅供讀取複本進行分析或分區。
- 對於在不同 AWS 區域 中具有僅供讀取複本的資料庫叢集，可在不同 AWS 區域 中複寫不同的資料庫或資料表。
- 指定要與 Aurora MySQL 資料庫叢集 (設定為輸入複寫拓撲中的複寫) 一起複寫的資料庫和資料表。如需此組態的詳細資訊，請參閱「Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)」。

主題

- 設定適用於 Aurora MySQL 的複寫篩選參數
- Aurora MySQL 的複寫篩選限制
- Aurora MySQL 的複寫篩選範例
- 檢視僅供讀取複本的複寫篩選條件

設定適用於 Aurora MySQL 的複寫篩選參數

若要設定複寫篩選條件，請設定下列參數：

- `binlog-do-db` - 將變更複寫到指定的二進位日誌。當您為 binlog 來源叢集設定此參數時，只會複寫參數中指定的二進位日誌。
- `binlog-ignore-db` - 請不要將變更複寫到指定的二進位日誌。為 binlog 來源叢集設定 `binlog-do-db` 參數時，不會評估此參數。
- `replicate-do-db` – 將變更複寫至指定的資料庫。當您為 binlog 複本叢集設定此參數時，只會複寫參數中指定的資料庫。
- `replicate-ignore-db` – 請勿將變更複寫至指定的資料庫。為 binlog 複本叢集設定 `replicate-do-db` 參數時，不會評估此參數。

- `replicate-do-table` – 將變更複製到指定的資料表。當您為僅供讀取複本設定此參數時，只會複寫參數中指定的資料表。此外，設定 `replicate-do-db` 或 `replicate-ignore-db` 參數時，請務必使用 binlog 複本叢集將包含指定資料表的資料庫納入複寫中。
- `replicate-ignore-table` – 請勿將變更複寫至指定的資料表。為 binlog 複本叢集設定 `replicate-do-table` 參數時，不會評估此參數。
- `replicate-wild-do-table` – 根據指定的資料庫和資料表名稱模式複寫資料表。支援 `%` 和 `_` 萬用字元。設定 `replicate-do-db` 或 `replicate-ignore-db` 參數時，請務必使用 binlog 複本叢集將包含指定資料表的資料庫納入複寫中。
- `replicate-wild-ignore-table` – 請勿根據指定的資料庫和資料表名稱模式複寫資料表。支援 `%` 和 `_` 萬用字元。為 binlog 複本叢集設定 `replicate-do-table` 或 `replicate-wild-do-table` 參數時，不會評估此參數。

系統會按照列出的順序對參數進行評估。如需有關這些參數如何運作的詳細資訊，請參閱 MySQL 文件：

- 如需一般資訊，請參閱複本伺服器選項和變數。
- 如需有關如何評估資料庫複寫篩選參數的資訊，請參閱評估資料庫層級複寫和二進位日誌記錄選項。
- 如需如何評估資料表複寫篩選參數的詳細資訊，請參閱評估資料表層級複寫選項。

根據預設，這些參數中的每個參數都有一個空值。在每個 binlog 叢集上，您可以使用這些參數來設定、變更和刪除複寫篩選條件。當您設定其中一個參數時，請使用逗號將每個篩選條件與其他篩選條件分隔。

您可以在 `%` 和 `_` 參數中使用 `replicate-wild-do-table` 和 `replicate-wild-ignore-table` 萬用字元。`%` 萬用字元等同於任意數目的字元，而 `_` 萬用字元只會等同於一個字元。

來源資料庫執行個體的二進位記錄格式對複寫非常重要，因為它會決定資料變更的記錄。`binlog_format` 參數的設定會決定複寫是以資料列為基礎還是以陳述式為基礎。如需詳細資訊，請參閱 為單一可用區資料庫設定 Aurora MySQL 二進位記錄。

> **Note**
>
> 無論來源資料庫執行個體上的 `binlog_format` 設定為何，所有資料定義語言 (DDL) 陳述式都會複寫為陳述式。

Aurora MySQL 的複寫篩選限制

下列限制適用於 Aurora MySQL 的複寫篩選：

- 僅 Aurora MySQL 第 3 版支援複寫篩選。
- 每個複寫篩選參數都有 2,000 個字元的限制。
- 複寫篩選條件不支援逗號。
- 複寫篩選不支援 XA 交易。

  如需詳細資訊，請參閱 MySQL 文件中的 XA 交易的限制。

Aurora MySQL 的複寫篩選範例

若要設定僅供讀取複本的複寫篩選，請修改與僅供讀取複本關聯之資料庫叢集參數群組中的複寫篩選參數。

> **Note**
>
> 您無法修改預設資料庫叢集參數群組。如果僅供讀取複本使用預設參數群組，請建立新的參數群組，並將它與僅供讀取複本建立關聯。如需資料庫叢集參數群組的詳細資訊，請參閱 Amazon Aurora 的參數群組。

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 在資料庫叢集參數群組中設定參數。如需有關設定參數的詳細資訊，請參閱 修改 Amazon Aurora 中的資料庫參數群組中的參數。當您在資料庫叢集參數群組中設定參數時，與參數群組關聯的所有資料庫叢集都會使用參數設定。如果您在資料庫叢集參數群組中設定複寫篩選參數，請確定參數群組僅與僅供讀取複本叢集相關聯。將來源資料庫執行個體的複寫篩選參數保留空白。

下列範例會使用 AWS CLI 設定參數。這些範例將 `ApplyMethod` 設定為 `immediate`，以便在 CLI 命令完成後立即發生參數變更。如果您想要在僅供讀取複本重新啟動後套用擱置變更，請將設定 `ApplyMethod` 為 `pending-reboot`。

下列範例會設定複寫篩選條件：

- Including databases in replication
- Including tables in replication
- Including tables in replication with wildcard characters

- Excluding databases from replication
- Excluding tables from replication
- Excluding tables from replication using wildcard characters

Example在複寫中包含資料庫

下列範例包含複寫中的 mydb1 和 mydb2 資料庫。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

在 Windows 中：

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

Example在複寫中包含資料表

下列範例包含複寫資料庫 table1 中的 table2 和 mydb1 資料表。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

在 Windows 中：

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
```

```
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

Example使用萬用字元在複寫中包含資料表

下列範例包含複寫時在資料庫 order 中名稱開頭為 return 和 mydb 的資料表。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

在 Windows 中：

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

Example從複寫中排除資料庫

下列範例會從複寫中排除 mydb5 和 mydb6 資料庫。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6',ApplyMethod=immediate"
```

在 Windows 中：

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6,ApplyMethod=immediate"
```

Example從複寫中排除資料表

下列範例會從複寫中排除資料庫 mydb5 中的資料表 table1 和資料庫 mydb6 中的 table2。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

在 Windows 中：

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

Example使用萬用字元從複寫中排除資料表

下列範例會從複寫中排除資料庫 order 中名稱開頭為 return 和 mydb7 的資料表。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

在 Windows 中：

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

檢視僅供讀取複本的複寫篩選條件

您可以使用下列方式檢視僅供讀取複本的複寫篩選條件：

- 檢查與僅供讀取複本關聯之參數群組中複寫篩選參數的設定。

  如需說明，請參閱「在 Amazon Aurora 中檢視資料庫參數群組的參數值」。
- 在 MySQL 用戶端中，連線至僅供讀取複本並執行 `SHOW REPLICA STATUS` 陳述式。

  在輸出中，下列欄位會顯示僅供讀取複本的複寫篩選條件：
  - `Binlog_Do_DB`
  - `Binlog_Ignore_DB`
  - `Replicate_Do_DB`
  - `Replicate_Ignore_DB`
  - `Replicate_Do_Table`
  - `Replicate_Ignore_Table`
  - `Replicate_Wild_Do_Table`
  - `Replicate_Wild_Ignore_Table`

  如需有關這些欄位的詳細資訊，請參閱 MySQL 文件中的檢查複寫狀態。

監控 Amazon Aurora MySQL 複寫

讀取擴展和高可用性取決於最短延遲時間。您可以監控 Amazon CloudWatch `AuroraReplicaLag` 指標，以監控 Aurora 複本落後於 Aurora MySQL 資料庫叢集之主要執行個體的程度。`AuroraReplicaLag` 指標會記錄在每個 Aurora 複本中。

主要資料庫執行個體也會記錄 `AuroraReplicaLagMaximum` 和 `AuroraReplicaLagMinimum` Amazon CloudWatch 指標。`AuroraReplicaLagMaximum` 指標會記錄主要執行個體與資料庫叢集中各個 Aurora 資料庫執行個體之間的最大延遲量。`AuroraReplicaLagMinimum` 指標會記錄主要執行個體與資料庫叢集中各個 Aurora 資料庫執行個體之間的最小延遲量。

如果需要 Aurora 複本延遲的最新值，您可以在 Amazon CloudWatch 中檢查 `AuroraReplicaLag` 值。Aurora 複本延遲也會記錄在 `information_schema.replica_host_status` 資料表中 Aurora MySQL 資料庫叢集中的每個 Aurora 複本上。如需此資料表的詳細資訊，請參閱 information_schema.replica_host_status。

如需監控 RDS 執行個體和 CloudWatch 指標的詳細資訊，請參閱 在 Amazon Aurora 叢集中監控指標。

跨 AWS 區域 複寫 Amazon Aurora MySQL 資料庫叢集

您可以建立 Amazon Aurora MySQL 資料庫叢集，作為與來源資料庫叢集不同 AWS 區域 中的僅供讀取複本。採取此方法可以改善您的災難復原功能，讓您將讀取操作擴展至靠近您使用者的 AWS 區域，並讓您更輕鬆在 AWS 區域 之間進行遷移。

您可以建立加密和未加密的資料庫叢集的僅供讀取複本。如果來源資料庫叢集已加密，則必須加密僅供讀取複本。

對於每個來源資料庫叢集，您最多可有五個屬於僅供讀取複本的跨區域資料庫叢集。

> **Note**
>
> 除了跨區域僅供讀取複本之外，您可以使用 Aurora 全域資料庫，以最小的延遲時間調整讀取操作。Aurora 全域資料庫在一個 AWS 區域 中具有主要 Aurora 資料庫叢集，以及在不同區域中具有最多 10 個次要唯讀資料庫叢集。每個次要資料庫叢集最多可包含 16 個 (而非 15) Aurora 複本。從主要資料庫叢集複寫到所有次要資料庫叢集，由 Aurora 儲存層而非資料庫引擎處理，因此複寫變更的延遲時間 (通常小於 1 秒)。將資料庫引擎保留在複寫程序之外，意味著資料庫引擎專用於處理工作負載。這也意味著您不需要設定或管理 Aurora MySQL 的 binlog (二進位日誌記錄) 複寫。如需詳細資訊，請參閱 使用 Amazon Aurora 全球資料庫。

在另一個 AWS 區域 中建立 Aurora MySQL 資料庫叢集僅供讀取複本時，您應該注意下列各項：

- 來源資料庫叢集和跨區域僅供讀取複本資料庫叢集，連同資料庫叢集的主要執行個體，最多可有 15 個 Aurora 複本。您可以使用此功能，同時針對來源 AWS 區域 和複寫目標 AWS 區域 擴展讀取作業。
- 在跨區域案例中，由於 AWS 區域 之間較長的網路通道，使得來源資料庫叢集和僅供讀取複本之間有較多的延遲時間。
- 跨區域複寫傳輸的資料會衍生 Amazon RDS 數據傳輸費。對於傳出來源 AWS 區域 的資料，下列跨區域複寫動作會產生費用：
  - 當您建立僅供讀取複本時，Amazon RDS 會取得來源叢集的快照，並將快照傳輸至僅供讀取複本的 AWS 區域。
  - 在來源資料庫中每次修改資料時，Amazon RDS 會從來源區域將資料傳輸至僅供讀取複本 AWS 區域。

  如需 Amazon RDS 資料傳輸定價的詳細資訊，請參閱 Amazon Aurora 定價。

- 您可以針對參考相同來源資料庫叢集的僅供讀取複本執行多個並行建立或刪除動作。不過，您必須維持每個來源資料庫叢集五個以內僅供讀取複本的限制。
- 若希望複寫作業順利運作，每個僅供讀取複本具備的運算和儲存資源數量應與來源資料庫叢集相同。若您擴展來源資料庫叢集，您也應該擴展僅供讀取複本。

主題

- 開始之前
- 建立 Aurora MySQL 的跨區域僅供讀取資料庫叢集。
- 將僅供讀取複本提升為 Aurora MySQL 的資料庫叢集
- 對 Amazon Aurora MySQL 跨區域複本進行故障診斷

開始之前

在可以建立屬於跨區域僅供讀取複本的 Aurora MySQL 資料庫叢集之前，必須在來源 Aurora MySQL 資料庫叢集上開啟二進位日誌。Aurora MySQL 的跨區域複寫使用 MySQL 二進位複寫在跨區域僅供讀取複本資料庫叢集上重播變更。

若要在 Aurora MySQL 資料庫叢集上開啟二進位日誌，請更新來源資料庫叢集的 `binlog_format` 參數。`binlog_format` 參數為叢集層級參數，位於預設的叢集參數群組中。如果資料庫叢集使用預設的資料庫叢集參數群組，請建立新資料庫叢集參數群組來修改 `binlog_format` 設定。建議您將 `binlog_format` 設定為 `MIXED`。不過，如果需要特定的 binlog 格式，也可以將 `binlog_format` 設定為 `ROW` 或 `STATEMENT`。將 Aurora 資料庫叢集重新開機，讓變更生效。

如需搭配 Aurora MySQL 使用二進位記錄的詳細資訊，請參閱 Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。如需修改 Aurora MySQL 組態參數的詳細資訊，請參閱 Amazon Aurora 資料庫叢集和資料庫執行個體參數 和 Amazon Aurora 的參數群組。

建立 Aurora MySQL 的跨區域僅供讀取資料庫叢集。

您可以使用 AWS 管理主控台、AWS Command Line Interface (AWS CLI) 或 Amazon RDS API 來建立屬於跨區域僅供讀取複本的 Aurora 資料庫叢集。您可以透過加密和未加密的資料庫叢集建立跨區域僅供讀取複本。

使用 AWS 管理主控台 為 Aurora MySQL 建立跨區域僅供讀取複本時，Amazon RDS 會在目標 AWS 區域 中建立資料庫叢集，然後自動建立資料庫執行個體，它是該資料庫叢集的主要執行個體。

使用 AWS CLI 或 RDS API 建立跨區域僅供讀取複本時，您會先在目標 AWS 區域 中建立資料庫叢集，並等候它成為作用中。一旦它處於作用中狀態，您便可以建立資料庫執行個體，即該資料庫叢集的主要執行個體。

當僅供讀取複本資料庫叢集的主要執行個體變得可用時，複寫便會開始。

使用下列程序透過 Aurora MySQL 資料庫叢集建立跨區域僅供讀取複本。這些程序適用透過加密或未加密的資料庫叢集建立僅供讀取複本。

主控台

使用 AWS 管理主控台 建立屬於跨區域僅供讀取複本的 Aurora MySQL 資料庫叢集

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在 AWS 管理主控台 的右上角，選取裝載來源資料庫叢集的 AWS 區域。
3. 在導覽窗格中，選擇 Databases (資料庫)。
4. 選擇要建立跨區域僅供讀取複本的資料庫叢集。
5. 若為 Actions (動作)，請選擇 Creat cross region replica (建立跨區域僅供讀取複本)。
6. 在 Create cross region read replica (建立跨區域僅供讀取複本) 頁面上，選擇跨區域僅供讀取複本資料庫叢集的選項設定，如下表所述。

選項	描述
目的地區域	選擇要託管新跨區域僅供讀取複本資料庫叢集的 AWS 區域。
Destination DB subnet group (目的地資料庫子網路群組)	選擇要用於跨區域僅供讀取複本資料庫叢集的資料庫子網路群組。
可公開存取	選擇 Yes (是) 以提供跨區域僅供讀取複本資料庫叢集一個公有 IP 地址；否則，選取 No (否)。
加密	選取 Enable Encryption (啟用加密) 以開啟此資料庫叢集的靜態加密。如需更多詳細資訊，請參閱 加密 Amazon Aurora 資源。
AWS KMS key	只有在 Encryption (加密) 設為 Enable Encryption (啟用加密) 時才能使用。選取要用於加密此資料庫叢集

選項	描述
	的 AWS KMS key。如需更多詳細資訊，請參閱 加密 Amazon Aurora 資源。
DB instance class (資料庫執行個體類別)	選擇資料庫執行個體類別，為資料庫叢集內的主要執行個體定義處理和記憶體要求。如需資料庫執行個體類別選項的詳細資訊，請參閱Amazon Aurora 資料庫執行個體類別。
Multi-AZ deployment (異地同步備份部署)	選擇 Yes (是)，在目標 AWS 區域 的另一個可用區域中建立新資料庫叢集的僅供讀取複本，以獲得容錯移轉支援。如需多個可用區域的詳細資訊，請參閱區域和可用區域。
Read replica source (僅供讀取複本來源)	選擇要為其建立跨區域僅供讀取複本的來源資料庫叢集。
DB instance identifier (資料庫執行個體識別符)：	輸入跨區域僅供讀取複本資料庫叢集中主要執行個體的名稱。此識別符用於新資料庫叢集內主要執行個體的端點位址。 該資料庫執行個體識別符有下列限制： • 必須包含 1 到 63 個英數字元或連字號。 • 第一個字元必須是字母。 • 不能以一個連字號結尾或是連續包含兩個連字號。 • 其對每個 AWS 區域、每個 AWS 帳戶 的所有資料庫執行個體都必須是唯一的。 因為跨區域僅供讀取複本資料庫叢集是從來源資料庫叢集的快照建立而來，僅供讀取複本的主要使用者名稱和主要密碼會與來源資料庫叢集的主要使用者名稱與主要密碼相同。

選項	描述
DB cluster identifier (資料庫叢集識別符)	鍵入跨區域僅供讀取複本資料庫叢集的名稱，其對複本的目標 AWS 區域 中的帳戶是唯一的。此識別符用於資料庫叢集中的叢集端點位址。如需叢集端點的詳細資訊，請參閱Amazon Aurora 端點連線。 資料庫叢集識別符有下列限制： • 必須包含 1 到 63 個英數字元或連字號。 • 第一個字元必須是字母。 • 不能以一個連字號結尾或是連續包含兩個連字號。 • 其對每個 AWS 區域 內每個 AWS 帳戶 的所有資料庫叢集都必須是唯一的。
優先順序	選擇新資料庫叢集主要執行個體的容錯移轉優先順序。此優先順序決定從主要執行個體失敗中復原時提升 Aurora 複本的順序。如果您未選取值，則預設值為 tier-1 (第一層)。如需更多詳細資訊，請參閱 Aurora 資料庫叢集的容錯能力。
Database port (資料庫連接埠)	指定應用程式和公用程式將用於存取資料庫的連接埠。Aurora 資料庫叢集預設為 MySQL 預設連接埠 3306。某些公司的防火牆會封鎖與此預設連接埠的連線。如果您的公司防火牆會封鎖預設連接埠，請為新的資料庫叢集選擇另一個連接埠。
Enhanced monitoring (增強型監控)	選擇 Enable enhanced monitoring (啟用增強型監控)，以針對資料庫叢集執行所在的作業系統即時開啟收集指標。如需更多詳細資訊，請參閱 使用增強型監控來監控作業系統指標。

選項	描述
監控角色	只有在 Enhanced Monitoring (增強型監控) 設為 Enable enhanced monitoring (啟用增強型監控) 時才能使用。選擇您所建立的 IAM 角色，以允許 Amazon RDS 為您與 Amazon CloudWatch Logs 進行通訊，或選擇 Default (預設值)，讓 RDS 為您建立名為 `rds-monitoring-role` 的角色。如需更多詳細資訊，請參閱使用增強型監控來監控作業系統指標。
精細程度	只有在 Enhanced Monitoring (增強型監控) 設為 Enable enhanced monitoring (啟用增強型監控) 時才能使用。針對資料庫叢集，設定收集指標之間的間隔 (以秒為單位)。
Auto minor version upgrade (自動次要版本升級)	此設定不適用於 Aurora MySQL 資料庫叢集。 如需 Aurora MySQL 引擎更新的詳細資訊，請參閱 Amazon Aurora MySQL 的資料庫引擎更新。

7. 選擇 Create (建立) 來建立您的 Aurora 跨區域僅供讀取複本。

AWS CLI

使用 CLI 建立屬於跨區域僅供讀取複本的 Aurora MySQL 資料庫叢集

1. 在您想要建立僅供讀取複本資料庫叢集的 AWS 區域 中呼叫 AWS CLI create-db-cluster 命令。包含 `--replication-source-identifier` 選項，並指定要為其建立僅供讀取複本的來源資料庫叢集的 Amazon Resource Name (ARN)。

   對於由 `--replication-source-identifier` 所識別資料庫叢集已加密的跨區域複寫，必須指定 `--kms-key-id` 選項和 `--storage-encrypted` 選項。

   > **Note**
   >
   > 您可以透過指定 `--storage-encrypted` 並提供 `--kms-key-id` 的值，設定從未加密的資料庫叢集對加密的僅供讀取複本的跨區域複寫。

您無法指定 `--master-username` 和 `--master-user-password` 參數。那些值是從來源資料庫叢集中取得。

下列程式碼範例會透過 us-west-2 區域中未加密的資料庫叢集快照，在 us-east-1 區域中建立僅供讀取複本。此命令是在 us-east-1 區域中呼叫。此範例會指定 `--manage-master-user-password` 選項來產生主要使用者密碼，並在 Secrets Manager 中管理該密碼。如需更多詳細資訊，請參閱 使用 Amazon Aurora 和 進行密碼管理 AWS Secrets Manager。或者，您可以使用 `--master-password` 選項，自行指定和管理密碼。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
  --db-cluster-identifier sample-replica-cluster \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.08.0 \
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster
```

在 Windows 中：

```
aws rds create-db-cluster ^
  --db-cluster-identifier sample-replica-cluster ^
  --engine aurora-mysql ^
  --engine-version 8.0.mysql_aurora.3.08.0 ^
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster
```

下列程式碼範例會透過 us-west-2 區域中加密的資料庫叢集快照，在 us-east-1 區域中建立僅供讀取複本。此命令是在 us-east-1 區域中呼叫。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
  --db-cluster-identifier sample-replica-cluster \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.08.0 \
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster \
  --kms-key-id my-us-east-1-key \
```

```
  --storage-encrypted
```

在 Windows 中：

```
aws rds create-db-cluster ^
  --db-cluster-identifier sample-replica-cluster ^
  --engine aurora-mysql ^
  --engine-version 8.0.mysql_aurora.3.08.0 ^
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster ^
  --kms-key-id my-us-east-1-key ^
  --storage-encrypted
```

AWS GovCloud (美國東部) 和 AWS GovCloud (美國西部) 區域之間的跨區域複寫需要 `--source-region` 選項，其中 `--replication-source-identifier` 識別的資料庫叢集已加密。對於 `--source-region`，請指定來源資料庫叢集的 AWS 區域。

若未指定 `--source-region`，請指定 `--pre-signed-url` 值。presigned URL (預先簽章的 URL) 為包含對來源 AWS 區域 中呼叫 `create-db-cluster` 命令之 Signature 第 4 版簽章請求的 URL。如要進一步了解 `pre-signed-url` 選項，請參閱《AWS CLI 命令參考》中的 create-db-cluster。

2. 使用 AWS CLI describe-db-clusters 命令來檢查資料庫叢集是否已可供使用，如下列範例所示。

```
aws rds describe-db-clusters --db-cluster-identifier sample-replica-cluster
```

當 **`describe-db-clusters`** 結果顯示 `available` 的狀態時，請建立資料庫叢集的主要執行個體以便複寫可以開始。若要這麼做，請使用 AWS CLI create-db-instance 命令，如下列範例所示。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
  --db-cluster-identifier sample-replica-cluster \
  --db-instance-class db.r5.large \
  --db-instance-identifier sample-replica-instance \
  --engine aurora-mysql
```

在 Windows 中：

```
aws rds create-db-instance ^
  --db-cluster-identifier sample-replica-cluster ^
  --db-instance-class db.r5.large ^
  --db-instance-identifier sample-replica-instance ^
  --engine aurora-mysql
```

當資料庫執行個體建立並可用時，複寫即會開始。您可呼叫 AWS CLI describe-db-instances 命令來判斷資料庫執行個體是否可用。

RDS API

使用 API 建立屬於跨區域僅供讀取複本的 Aurora MySQL 資料庫叢集

1. 在您想要建立僅供讀取複本資料庫叢集的 AWS 區域 中呼叫 RDS API CreateDBCluster 作業。包含 `ReplicationSourceIdentifier` 參數，並指定要為其建立僅供讀取複本之來源資料庫叢集的 Amazon Resource Name (ARN)。

   對於由 `ReplicationSourceIdentifier` 所識別資料庫叢集已加密的跨區域複寫，須指定 `KmsKeyId` 參數，並將 `StorageEncrypted` 參數設定為 `true`。

   > **Note**
   >
   > 您可以透過將 `StorageEncrypted` 指定為 **`true`** 並提供 `KmsKeyId` 的值，設定從未加密的資料庫叢集對加密的僅供讀取複本的跨區域複寫。在此情況下，您不需要指定 `PreSignedUrl`。

   您不需要包含 `MasterUsername` 和 `MasterUserPassword` 參數，因為這些值取自來源資料庫叢集。

   下列程式碼範例會透過 us-west-2 區域中未加密的資料庫叢集快照，在 us-east-1 區域中建立僅供讀取複本。此動作是在 us-east-1 區域中呼叫。

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBCluster
  &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
  &DBClusterIdentifier=sample-replica-cluster
```

```
  &Engine=aurora-mysql
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T001547Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=a04c831a0b54b5e4cd236a90dcb9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

下列程式碼範例會透過 us-west-2 區域中加密的資料庫叢集快照，在 us-east-1 區域中建立僅供讀取複本。此動作是在 us-east-1 區域中呼叫。

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBCluster
  &KmsKeyId=my-us-east-1-key
  &StorageEncrypted=true
  &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCreateDBCluster
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
         %2526ReplicationSourceIdentifier%253Darn%25253Aaws%25253Ards%25253Aus-
west-2%25253A123456789012%25253Acluster%25253Asample-master-cluster
         %2526SignatureMethod%253DHmacSHA256
         %2526SignatureVersion%253D4
         %2526Version%253D2014-10-31
         %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
         %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-
west-2%252Frds%252Faws4_request
         %2526X-Amz-Date%253D20161117T215409Z
         %2526X-Amz-Expires%253D3600
         %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-
amz-content-sha256%253Bx-amz-date
         %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
  &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
  &DBClusterIdentifier=sample-replica-cluster
  &Engine=aurora-mysql
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
```

```
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T001547Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=a04c831a0b54b5e4cd236a90dcb9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

對於 AWS GovCloud (美國東部) 和 AWS GovCloud (美國西部) 區域之間的跨區域複寫，其中 `ReplicationSourceIdentifier` 識別的資料庫叢集已加密，亦請指定 `PreSignedUrl` 參數。預先簽章的 URL 必須是對 `CreateDBCluster` API 作業的有效請求，而此動作可執行於包含要複寫之加密資料庫叢集的來源 AWS 區域 中。KMS 金鑰識別碼用來加密僅供讀取複本，而且必須是對目的地 AWS 區域 有效的 KMS 金鑰。如要自動而非手動產生預先簽章的 URL，請改為使用 AWS CLI create-db-cluster 命令搭配 `--source-region` 選項。

2. 使用 RDS API DescribeDBClusters 作業來檢查資料庫叢集是否已可供使用，如下列範例所示。

```
https://rds.us-east-1.amazonaws.com/
  ?Action=DescribeDBClusters
  &DBClusterIdentifier=sample-replica-cluster
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T002223Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=84c2e4f8fba7c577ac5d820711e34c6e45ffcd35be8a6b7c50f329a74f35f426
```

當 `DescribeDBClusters` 結果顯示 `available` 的狀態時，請建立資料庫叢集的主要執行個體，則複寫可隨即開始。若要這麼做，請使用 RDS API CreateDBInstance 動作，如下列範例所示。

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBInstance
  &DBClusterIdentifier=sample-replica-cluster
  &DBInstanceClass=db.r5.large
  &DBInstanceIdentifier=sample-replica-instance
  &Engine=aurora-mysql
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
```

```
&X-Amz-Date=20160201T003808Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=125fe575959f5bbcebd53f2365f907179757a08b5d7a16a378dfa59387f58cdb
```

當資料庫執行個體建立並可用時，複寫即會開始。您可呼叫 AWS CLI DescribeDBInstances 命令來判斷資料庫執行個體是否可用。

檢視 Amazon Aurora MySQL 跨區域複本

您可呼叫 describe-db-clusters AWS CLI 命令或 DescribeDBClusters RDS API 作業來檢視您 Amazon Aurora MySQL 資料庫叢集的跨區域複寫關係。於回應中，請參閱 `ReadReplicaIdentifiers` 欄位，以取得任何跨區域讀取複本資料庫叢集的資料庫叢集識別符。有關作為複寫來源之來源資料庫叢集的 ARN，請參閱 `ReplicationSourceIdentifier` 元素。

將僅供讀取複本提升為 Aurora MySQL 的資料庫叢集

您可以提升 Aurora MySQL 僅供讀取複本為獨立的資料庫叢集。當您提升 Aurora MySQL 僅供讀取複本時，其資料庫執行個體將在可使用前重新啟動。

一般而言，您會在來源資料庫叢集失敗時，才提升 Aurora MySQL 僅供讀取複本為獨立的資料庫叢集，以做為資料復原結構描述。

要進行此操作，請先建立僅供讀取複本，然後監控來源資料庫叢集的故障。若發生故障，請執行下列程序：

1. 提升僅供讀取複本。
2. 將資料庫流量引導至提升的資料庫叢集。
3. 以提升的資料庫叢集做為來源，建立替換的僅供讀取複本。

當您提升僅供讀取複本時，該僅供讀取複本就成為獨立的 Aurora 資料庫叢集。該提升程序可耗費數分鐘或更長的時間來完成，視僅供讀取複本大小而定。在您提升僅供讀取複本為新的資料庫叢集後，它與其他資料庫叢集無異。例如，您可以從其中建立僅供讀取複本，並執行時間點還原操作。您也可以建立資料庫叢集的 Aurora 複本。

因為提升的資料庫叢集將不再是僅供讀取複本，因此您不得將其用做複寫目標。

以下步驟顯示了提升僅供讀取複本至資料庫叢集的一般流程：

1. 停止至僅供讀取複本來源資料庫叢集的任何寫入交易，然後等待針對僅供讀取複本的所有更新。
   僅供讀取複本在來源資料庫叢集上發生資料庫更新後，此複寫延遲可能會有很大差異。使用

`ReplicaLag` 指標以確定針對僅供讀取複本進行的所有更新時間。`ReplicaLag` 指標會記錄讀取複本資料庫執行個體落後於來源資料庫執行個體的時間量。當 `ReplicaLag` 指標到達 `0`，讀取複本即已跟上來源資料庫執行個體。

2. 使用 Amazon RDS 主控台上的 Promote (提升) 選項、AWS CLI 命令 promote-read-replica-db-cluster 或 PromoteReadReplicaDBCluster Amazon RDS API 作業，提升僅供讀取複本。

   您可選擇一個 Aurora MySQL 資料庫執行個體以提升僅供讀取複本。在僅供讀取複本提升後，Aurora MySQL 資料庫叢集即已提升為獨立資料庫叢集。具有最高容錯移轉優先順序的資料庫執行個體，已提升為資料庫叢集的資料庫執行個體。其他成為 Aurora 複本的資料庫執行個體。

   > **Note**
   >
   > 提升程序可能需要幾分鐘來完成。當您提升僅供讀取複本時，複寫便停止了，且資料庫執行個體將重新啟動。當重新啟動完成，該僅供讀取複本便可用做新的資料庫叢集。

主控台

將 Aurora MySQL 僅供讀取複本提升為資料庫叢集

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在主控台中，選擇 Instances (執行個體)。

   Instance (執行個體) 窗格隨即出現。
3. 在 Instances (執行個體) 窗格中，選擇您想提升的僅供讀取複本。

   該僅供讀取複本將顯示為 Aurora MySQL 資料庫執行個體。
4. 在 Actions (動作) 中選擇 Promote read replica (提升僅供讀取複本)。
5. 在確認頁面上，選擇 Promote read replica (提升僅供讀取複本)。

AWS CLI

如要將僅供讀取複本提升為資料庫叢集，請使用 AWS CLI promote-read-replica-db-cluster 命令。

Example

對於 Linux、macOS 或 Unix：

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier mydbcluster
```

在 Windows 中：

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier mydbcluster
```

RDS API

如要將讀取複本提升為資料庫叢集，請呼叫 PromoteReadReplicaDBCluster。

對 Amazon Aurora MySQL 跨區域複本進行故障診斷

在以下您可以找到建立 Amazon Aurora 跨區域僅供讀取複本時可能遇到的常見錯誤訊息清單，以及如何解決指定的錯誤。

來源叢集 [資料庫叢集 ARN] 未啟用 binlog

若要解決此問題，請在來源資料庫叢集上開啟二進位日誌。如需更多詳細資訊，請參閱 開始之前。

來源叢集 [資料庫叢集 ARN] 沒有與寫入器同步的叢集參數群組

如果已更新 `binlog_format` 資料庫叢集參數，但尚未將資料庫叢集的主要執行個體重新開機，則會收到此錯誤。請將資料庫叢集的主要執行個體 (即寫入器) 重新開機並重試。

來源叢集 [資料庫叢集 ARN] 在此區域中已具有僅供讀取複本

對於任何 AWS 區域 中每個來源資料庫叢集，您最多可有五個屬於僅供讀取複本的跨區域資料庫叢集。如果特定 AWS 區域 中對某個資料庫叢集的僅供讀取複本數目已達到上限，則必須先刪除現有的僅供讀取複本，才能在該區域中建立新的跨區域資料庫叢集。

資料庫叢集 [資料庫叢集 ARN] 需要資料庫引擎升級，才能支援跨區域複寫

若要解決此問題，請將來源資料庫叢集中所有執行個體的資料庫引擎版本升級至最新的資料庫引擎版本，然後嘗試再次建立跨區域僅供讀取複本資料庫。

Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)

因為 Amazon Aurora MySQL 與 MySQL 相容，您可以設定 MySQL 資料庫與 Amazon Aurora MySQL 資料庫叢集之間的複寫。這種類型的複寫使用 MySQL 二進位記錄複寫，也稱為 binlog 複寫。如果

您使用二進位複寫搭配 Aurora，我們建議您的 MySQL 資料庫執行 5.5 版或更新版本。您可以設定複寫，其中您的 Aurora MySQL 資料庫叢集是複寫來源或複本。您可以使用 Amazon RDS MySQL 資料庫執行個體、Amazon RDS 外部 MySQL 資料庫或其他 Aurora MySQL 資料庫叢集進行複寫。

> **Note**
>
> 您無法對特定類型的 Aurora 資料庫叢集使用 Binlog 複寫。特別是 Binlog 複寫不適用於 Aurora Serverless v1 叢集。如果 `SHOW MASTER STATUS` 和 `SHOW SLAVE STATUS` (Aurora MySQL 第 2 版) 或 `SHOW REPLICA STATUS` (Aurora MySQL 第 3 版) 陳述式未傳回任何輸出，請檢查您使用的叢集是否支援 binlog 複寫。

您也可以在另一個 AWS 區域中使用 RDS for MySQL 資料庫執行個體或 Aurora MySQL 資料庫叢集來進行複寫。當您跨 執行複寫時 AWS 區域，請確定您的資料庫叢集和資料庫執行個體可公開存取。如果 Aurora MySQL 資料庫叢集位於 VPC 的私有子網路中，請在 AWS 區域之間使用 VPC 對等互連。如需詳細資訊，請參閱由不同 VPC 中的 EC2 叢集存取 VPC 中的資料庫執行個體。

如果您想要在 Aurora MySQL 資料庫叢集和另一個 Aurora MySQL 資料庫叢集之間設定複寫 AWS 區域，您可以在 AWS 區域 與來源資料庫叢集不同的 中建立 Aurora MySQL 資料庫叢集做為僅供讀取複本。如需詳細資訊，請參閱跨 AWS 區域 複寫 Amazon Aurora MySQL 資料庫叢集。

在 Aurora MySQL 第 2 版本和第 3 版，您可以在 Aurora MySQL 與外部資源之間進行複寫，或是使用全域交易識別碼 (GTIDs) 的目標進行複寫。確保資料庫叢集 Aurora MySQL 中GTID 相關的參數，設定為相容外部資料庫的狀態。若要了解如何操作，請參閱 使用 GTID 式複寫。在 Aurora MySQL 3.01 版及更新版本中，您可以選擇如何將 GTID 指派給從不使用 GTID 的來源複寫的交易。如需控制該設定之預存程序的相關資訊，請參閱 mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL 第 3 版)。

> **Warning**
>
> 在 Aurora MySQL 與 MySQL 之間進行複寫時，請確定您僅使用 InnoDB 資料表。如果有您想要複寫的 MyISAM 資料表，您可以在使用下列命令設定複寫之前，將它們轉換為 InnoDB。
>
> ```
> alter table <schema>.<table_name> engine=innodb, algorithm=copy;
> ```

在下列各節中，設定複寫、停止複寫、擴展資料庫的讀取、最佳化 binlog 複寫，以及設定增強型 binlog。

主題

- 設定 Aurora MySQL 的二進位日誌複寫
- 停止 Aurora MySQL 的二進位日誌複寫
- 使用 Amazon Aurora 為 MySQL 資料庫擴展讀取
- 最佳化 Aurora MySQL 的二進位日誌複寫
- 設定 Aurora MySQL 的增強型 binlog

設定 Aurora MySQL 的二進位日誌複寫

使用 Aurora MySQL 設定 MySQL 複寫涉及下列詳細討論的步驟：

內容

- 1. 在複寫來源上開啟二進位日誌
- 2. 在複寫來源上保留二進位日誌，直到不再需要為止
- 3. 建立複寫來源的複製或傾印
- 4. 將傾印載入複本目標 (如有需要)
- 5. 在複製來源上建立複寫使用者
- 6. 在複本目標上開啟複寫
  - 設定位置以停止僅供讀取複本的複寫作業
- 7. 監控複本
- 同步複寫來源和目標之間的密碼

1. 在複寫來源上開啟二進位日誌

在以下尋找如何在資料庫引擎的複寫來源上開啟二進位日誌的相關指示。

資料庫引擎	指示
Aurora MySQL	在 Aurora MySQL 資料庫叢集上開啟二進位日誌 建議將 `binlog_format` 資料庫叢集參數設為 `ROW`、`STATEMENT` 或 `MIXED`. `MIXED`，除非您需要特定的 binlog 格式。(預設值為 `OFF`。)

資料庫引擎	指示
	若要變更 binlog_format 參數，請建立自訂資料庫叢集參數群組，並將該自訂參數群組與您的資料庫叢集建立關聯。您無法變更預設資料庫叢集參數群組中的參數。 如果您要將 binlog_format 參數從 OFF 變更為另一個值，請將 Aurora 資料庫叢集重新開機，變更才能生效。 如需詳細資訊，請參閱Amazon Aurora 資料庫叢集和資料庫執行個體參數及Amazon Aurora 的參數群組。
RDS for MySQL	在 Amazon RDS 資料庫執行個體上開啟二進位日誌 您不可以直接為 Amazon RDS 資料庫執行個體開啟二進位日誌，但您可以透過以下其中一個方式開啟它： • 為資料庫執行個體開啟自動備份。您可以在建立資料庫執行個體時開啟自動備份，或可以透過修改現有資料庫執行個體來開啟備份。如需詳細資訊，請參閱《Amazon RDS 使用者指南》中的建立資料庫執行個體。 • 建立資料庫執行個體的僅供讀取複本。如需詳細資訊，請參閱《Amazon RDS 使用者指南》中的使用僅供讀取複本。

資料庫引擎	指示
MySQL (外部)	設定加密複寫 若要使用 Aurora MySQL 第 2 版安全地複寫資料，您可以使用加密複寫。 **Note** 如果您不需要使用加密複寫，可以略過這些步驟。 下列是使用加密複寫的先決條件： • 必須在外部 MySQL 來源資料庫上啟用 Secure Sockets Layer (SSL)。 • 必須為 Aurora MySQL 資料庫叢集準備用戶端金鑰和用戶端憑證。 在加密複寫期間，Aurora MySQL 資料庫叢集充當 MySQL 資料庫伺服器的用戶端。Aurora MySQL 用戶端的憑證和金鑰位於 .pem 格式的檔案中。 1. 確保您已準備好進行加密複寫： • 如果您在外部 MySQL 來源資料庫上未啟用 SSL，也沒有準備用戶端金鑰和用戶端憑證，請在 MySQL 資料庫伺服器上開啟 SSL，並產生所需的用戶端金鑰和用戶端憑證。 • 如果外部來源上已啟用 SSL，請為 Aurora MySQL 資料庫叢集提供用戶端金鑰和憑證。如果您沒有這些資料，請為 Aurora MySQL 資料庫叢集產生新的金鑰和憑證。若要簽署用戶端憑證，您必須有用於外部 MySQL 來源資料庫上設定 SSL 的憑證授權單位金鑰。 如需詳細資訊，請參閱 MySQL 文件中的使用 openssl 建立 SSL 憑證和金鑰。 您需要憑證授權單位憑證、用戶端金鑰和用戶端憑證。 2. 使用 SSL 以主要使用者的身分連接至 Aurora MySQL 資料庫叢集。 如需以 SSL 連接至 Aurora MySQL 資料庫叢集的相關資訊，請參閱Aurora MySQL 資料庫叢集的 TLS 連線。

資料庫引擎	指示

	3. 執行 mysql.rds_import_binlog_ssl_material 預存程序將 SSL 資訊匯入 Aurora MySQL 資料庫叢集。
	
	對於 ssl_material_value 參數，將 Aurora MySQL 資料庫叢集之 .pem 格式檔案中的資訊，插入正確的 JSON 承載中。
	
	下列範例將 SSL 資訊匯入 Aurora MySQL 資料庫叢集。在 .pem 格式檔案中，內文程式碼通常比範例所示的內文程式碼更長。
	
	`call mysql.rds_import_binlog_ssl_material(`
	`'{"ssl_ca":"-----BEGIN CERTIFICATE-----`
	`AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcj`
	`qP3maAhDFcvBS7O6V`
	`hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96`
	`xbiFveSFJuOp/d6RJhJOI0iBXr`
	`lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/`
	`i8SeJtjnV3iAoG/cQk+0FzZ`
	`qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz22`
	`1CBt5IMucxXPkX4rWi+z7wB3Rb`
	`BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE`
	`-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICA`
	`TE-----`
	`AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcj`
	`qP3maAhDFcvBS7O6V`
	`hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96`
	`xbiFveSFJuOp/d6RJhJOI0iBXr`
	`lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/`
	`i8SeJtjnV3iAoG/cQk+0FzZ`
	`qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz22`
	`1CBt5IMucxXPkX4rWi+z7wB3Rb`
	`BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE`
	`-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE`
	` KEY-----`
	`AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pc`
	`jqP3maAhDFcvBS7O6V`
	`hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSF`
	`JuOp/d6RJhJOI0iBXr`
	`lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJ`
	`tjnV3iAoG/cQk+0FzZ`

資料庫引擎	指示

```
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMu
cxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

如需詳細資訊，請參閱mysql.rds_import_binlog_ssl_material及Aurora MySQL 資料庫叢集的 TLS 連線。

> **Note**
>
> 執行程序之後，密碼會儲存在檔案中。若稍後要清除這些檔案，您可以執行 mysql.rds_remove_binlog_ssl_material 預存程序。

在外部 MySQL 資料庫上開啟二進位日誌

1. 從命令 shell 停用 mysql 服務。

   ```
   sudo service mysqld stop
   ```

2. 編輯 my.cnf 檔案 (此檔案通常位於 /etc 之下)。

   ```
   sudo vi /etc/my.cnf
   ```

   將 log_bin 和 server_id 選項新增至 [mysqld] 部分。log_bin 選項會提供二進位記錄檔的檔案名稱識別符。server_id 選項會為來源與複本關係提供伺服器唯一識別碼。

   如果不需要加密複寫，則啟動外部 MySQL 資料庫時，請確保啟用 binlog 並關閉 SSL。

   下列是未加密資料 /etc/my.cnf 檔案中的相關項目。

   ```
   log-bin=mysql-bin
   server-id=2133421
   innodb_flush_log_at_trx_commit=1
   ```

資料庫引擎	指示

```
sync_binlog=1
```

如果需要加密複寫，則啟動外部 MySQL 資料庫時，務必啟用 SSL 和 binlog。

`/etc/my.cnf` 檔案中的項目包括 MySQL 資料庫伺服器的 .pem 檔案位置。

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1

# Setup SSL.
ssl-ca=/home/sslcerts/ca.pem
ssl-cert=/home/sslcerts/server-cert.pem
ssl-key=/home/sslcerts/server-key.pem
```

此外，您 MySQL 資料庫執行個體的 `sql_mode` 選項必須設為 0，或必須包含在您的 My.cnf 檔案中。

在已連接外部 MySQL 資料庫的情況下，記錄外部 MySQL 資料庫的二進位日誌位置。

```
mysql> SHOW MASTER STATUS;
```

您的輸出應類似以下內容：

```
+------------------+----------+--------------+------------------
+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
 Executed_Gtid_Set |
+------------------+----------+--------------+------------------
+-------------------+
| mysql-bin.000031 |      107 |              |                  |
              |
+------------------+----------+--------------+------------------
+-------------------+
1 row in set (0.00 sec)
```

資料庫引擎	指示
	如需更多詳細資訊，請參閱 MySQL 文件中的Setting the replication source configuration。 3. 啟動 mysql 服務。 `sudo service mysqld start`

2. 在複寫來源上保留二進位日誌，直到不再需要為止

使用 MySQL 二進位日誌複寫時，Amazon RDS 不會管理複寫程序。因此，您需要確保會保留複寫來源上的 binlog 檔案，直到變更已套用至複本後為止。此維護可協助您在發生故障時將您的來源資料庫還原。

使用以下保留資料庫引擎二進位日誌的相關指示。

資料庫引擎	指示
Aurora MySQL	在 Aurora MySQL 資料庫叢集上保留二進位日誌 您無法存取 Aurora MySQL 資料庫叢集的 binlog 檔案。因此，您必須選擇要在複寫來源上保留 binlog 檔案的足夠時間範圍，以確保在 Amazon RDS 刪除 binlog 檔案之前，變更已套用至複本。您可以在 Aurora MySQL 資料庫叢集上保留 binlog 檔案最多 90 天。 如果您要將使用 MySQL 資料庫或 RDS for MySQL 資料庫執行個體的複寫設定為複本，並且要為其建立複本的資料庫非常大，請選擇較大的時間範圍來保留二進位日誌檔案，直到要複寫資料庫的初始複製完成，且複本延遲已到達 0 為止。 若要設定二進位日誌保留時間範圍，請使用 mysql.rds_set_configuration 程序，並指定 `'binlog retention hours'`組態參數加上資料庫叢集上保留 binlog 檔案的時數。Aurora MySQL 2.11.0 版和更新版本以及第 3 版的最大值是 2160 (90 天)。 下列範例會將 binlog 檔案的保留期間設定為 6 天：

資料庫引擎	指示
	`CALL mysql.rds_set_configuration('binlog retention hours', 144);` 複寫開始之後，您可以在複本上執行 `SHOW SLAVE STATUS` (Aurora MySQL 第 2 版) 或 `SHOW REPLICA STATUS` (Aurora MySQ 第 3 版) 命令並檢查 `Seconds behind master` 欄位，來確認變更已套用至複本。如果 `Seconds behind master` 欄位為 0，則沒有複本延遲。沒有複本延遲時，請減少保留 binlog 檔案的時間長度，方法是將 `binlog retention hours` 組態參數設定為較小的時間範圍。 如果未指定此設定，Aurora MySQL 的預設值為 24 (1 天)。 如果您指定 `'binlog retention hours'` 的值大於最大值，則 Aurora MySQL 會使用最大值。
RDS for MySQL	在 Amazon RDS 資料庫執行個體上保留二進位日誌 您可以在 Amazon RDS 資料庫執行個體上保留二進位日誌檔案，方法是如同 Aurora MySQL 資料庫叢集一般設定 binlog 保留時數，如前一列中所述。 您也可以透過為資料庫執行個體建立僅供讀取複本，在 Amazon RDS 資料庫執行個體上保留 binlog 檔案。這個僅供讀取複本是暫時性的，並專為保留 binlog 檔案目的使用。建立了僅供讀取複本之後，請在僅供讀取複本上呼叫 mysql.rds_stop_replication 預存程序。複寫停用時，Amazon RDS 不會刪除複寫來源上的任何 binlog 檔案。設定使用永久複本的複寫之後，當複寫來源與永久複本之間的複本延遲 (`Seconds behind master` 欄位) 到達 0 時，您可以刪除僅供讀取複本。
MySQL (外部)	在外部 MySQL 資料庫上保留二進位日誌 因為外部 MySQL 資料庫上的 binlog 檔案不是由 Amazon RDS 管理，系統會保留這些檔案直到您刪除它們為止。 複寫開始之後，您可以在複本上執行 `SHOW SLAVE STATUS` (Aurora MySQL 第 2 版) 或 `SHOW REPLICA STATUS` (Aurora MySQ 第 3 版) 命令並檢查 `Seconds behind master` 欄位，來確認變更已套用至複本。如果 `Seconds behind master` 欄位為 0，則沒有複本延遲。沒有複本延遲時，您可以刪除舊的 binlog 檔案。

3. 建立複寫來源的複製或傾印

您可以使用複寫來源的快照、複製或傾印，將資料的基準複製載入到複本上。然後，您會從該點開始複寫。

使用以下為資料庫引擎建立複寫來源複製或傾印的相關指示。

資料庫引擎	指示
Aurora MySQL	建立 Aurora MySQL 資料庫叢集的複製 使用下列其中一種方法： • 從資料庫叢集快照還原。 1. 建立您 Amazon Aurora 資料庫叢集的資料庫叢集快照。如需更多詳細資訊，請參閱 建立資料庫叢集快照。 2. 從您剛剛建立的資料庫叢集快照還原來建立新的 Aurora 資料庫叢集。 請確保為您所還原資料庫叢集保留與原始資料庫叢集相同的資料庫參數群組。這麼做可確保資料庫叢集的複本已啟用二進位日誌。如需詳細資訊，請參閱從資料庫叢集快照還原。 • 複製資料庫叢集。如需詳細資訊，請參閱複製 Amazon Aurora 資料庫叢集的一個磁碟區。 判斷 binlog 檔案名稱和位置 使用下列其中一種方法： • 在 中 AWS 管理主控台： 1. 在主控台中，選擇資料庫，然後選擇您的新 Aurora 資料庫叢集的主要執行個體 (寫入器) 來顯示其詳細資訊。 2. 捲動至 Recent Events (最近的事件)。包含 binlog 檔案名稱和位置的事件訊息隨即顯示。事件訊息採用下列格式。 `Binlog position from crash recovery is binlog-file-name  binlog-position` 3. 開始複寫時儲存 binlog 檔案名稱和位置值。

資料庫引擎	指示

- 呼叫 describe-events AWS CLI 命令，如下列範例所示。

```
aws rds describe-events

{
    "Events": [
        {
            "EventCategories": [],
            "SourceType": "db-instance",
            "SourceArn": "arn:aws:rds:us-west-2:123456789012:
db:sample-restored-instance",
            "Date": "2016-10-28T19:43:46.862Z",
            "Message": "Binlog position from crash recovery is mysql-
bin-changelog.000003 4278",
            "SourceIdentifier": "sample-restored-instance"
        }
    ]
}
```

- 檢查 MySQL 錯誤日誌中的最後一個 MySQL binlog 檔案位置。

建立 Aurora MySQL 資料庫叢集的傾印

如果您的複本目標是外部 MySQL 資料庫或 RDS for MySQL 資料庫執行個體，則必須從 Aurora 資料庫叢集建立傾印檔案。

請確保對您所建立來源資料庫叢集的複本執行 `mysqldump` 命令。這是為了避免在建立傾印時的鎖定考量。如果直接在來源資料庫叢集上取得傾印，則需要鎖定來源資料表，以防止傾印進行時同時寫入它們。

1. 使用 MySQL 用戶端來連線至資料庫叢集
2. 發出 `mysqldump` 命令。例如：

```
PROMPT> mysqldump --databases database_name --single-transaction
--order-by-primary -r backup.sql -u local_user s -p
```

3. 建立傾印檔案之後，您可以刪除資料庫叢集複製。

資料庫引擎	指示
RDS for MySQL	建立 Amazon RDS 資料庫執行個體的快照 建立 Amazon RDS 資料庫執行個體的僅供讀取複本。如需詳細資訊，請參閱《Amazon Relational Database Service 使用者指南》中的建立僅供讀取複本。 1. 連接至僅供讀取複本並執行 mysql.rds_stop_replication 程序來停止複寫。 2. 當僅供讀取複本已停止時，請連接至僅供讀取複本並執行 SHOW SLAVE STATUS (Aurora MySQL 第 2 版) 或 SHOW REPLICA STATUS (Aurora MySQL第 3 版) 命令。從 Relay_Master_Log_File 欄位擷取目前的二進位日誌檔案名稱，以及從 Exec_Master_Log_Pos 欄位擷取日誌檔案位置。開始複寫時儲存這些值。 3. 在僅供讀取複本保持為 Stopped (已停止) 時，建立僅供讀取複本的資料庫快照。如需詳細資訊，請參閱《Amazon Relational Database Service 使用者指南》中的建立資料庫快照。 4. 刪除僅供讀取複本。

MySQL (外部)	建立外部 MySQL 資料庫的快照或傾印
	
	1. 建立傾印之前，您需要確保傾印的 binlog 位置對來源執行個體中的資料而言是最新的。若要這麼做，您必須先使用下列命令來停止對執行個體的任何寫入操作：
	
	`mysql> FLUSH TABLES WITH READ LOCK;`
	
	2. 使用 mysqldump 命令來建立您的 MySQL 資料庫的傾印，如下所示：
	
	`PROMPT> sudo mysqldump --databases database_name --master-data=2 --single-transaction \`
	`--order-by-primary -r backup.sql -u local_user -p`
	
	3. 建立傾印之後，使用下列命令解除鎖定您 MySQL 資料庫中的資料表：
	
	`mysql> UNLOCK TABLES;`

4. 將傾印載入複本目標 (如有需要)

如果您計劃從位於 Amazon RDS 外部的 MySQL 資料庫傾印載入資料，那麼，您可能需要建立供複製傾印檔案的 EC2 執行個體。然後，您可以從該 EC2 執行個體將資料載入資料庫叢集或資料庫執行個體。使用此方法，您可以在將傾印檔案複製到 EC2 執行個體之前加以壓縮，以便減少與複製資料至 Amazon RDS 相關聯的網路成本。您也可以加密一或多個傾印檔案，以於網路間傳輸資料時加以保護。

> **Note**
>
> 如果您建立新的 Aurora MySQL 資料庫叢集作為複本目標，則不需要載入傾印檔案：
>
> - 您可以從資料庫叢集快照進行還原來建立新的資料庫叢集。如需詳細資訊，請參閱從資料庫叢集快照還原。
> - 您可以複製來源資料庫叢集來建立新的資料庫叢集。如需詳細資訊，請參閱複製 Amazon Aurora 資料庫叢集的一個磁碟區。
> - 您可以將資料從資料庫執行個體快照遷移到新的資料庫叢集。如需詳細資訊，請參閱將資料遷移至 Amazon Aurora MySQL 資料庫叢集。

使用以下為資料庫引擎將複寫來源的傾印載入複寫目標的相關指示。

資料庫引擎	指示

Aurora MySQL	將傾印載入至 Aurora MySQL 資料庫叢集
	1. 將 `mysqldump` 命令的輸出從複寫來源複製到也可以連接至您的 Aurora MySQL 資料庫叢集的位置。
	2. 使用 `mysql` 命令連接至您的 Aurora MySQL 資料庫叢集。以下是範例。
	`PROMPT> mysql -h host_name -port=3306 -u db_master_user -p`
	3. 出現 `mysql` 提示時，請執行 `source` 命令並傳入資料庫傾印檔案名稱，以將資料載入 Aurora MySQL 資料庫叢集，例如：
	`mysql> source backup.sql;`

資料庫引擎	指示

RDS for MySQL	將傾印載入至 Amazon RDS 資料庫執行個體
	1. 將 mysqldump 命令的輸出從複寫來源複製到也可以連接至 MySQL 資料庫執行個體的位置。
	2. 使用 mysql 命令連接至 MySQL 資料庫執行個體。以下是範例。
	`PROMPT> mysql -h host_name -port=3306 -u db_master_user -p`
	3. 出現 mysql 提示時，請執行 source 命令並傳入資料庫傾印檔案名稱，以將資料載入 MySQL 資料庫執行個體，例如：
	`mysql> source backup.sql;`

MySQL (外部)	將傾印載入外部 MySQL 資料庫
	您無法將資料庫快照或資料庫叢集快照載入至外部 MySQL 資料庫。而是必須使用來自 mysqldump 命令的輸出。
	1. 將 mysqldump 命令的輸出從複寫來源複製到也可以連接至 MySQL 資料庫的位置。
	2. 使用 mysql 命令連接至 MySQL 資料庫。以下是範例。
	`PROMPT> mysql -h host_name -port=3306 -u db_master_user -p`
	3. 出現 mysql 提示時，請執行 source 命令並傳入資料庫傾印檔案名稱，以將資料載入 MySQL 資料庫。以下是範例。
	`mysql> source backup.sql;`

5. 在複製來源上建立複寫使用者

在僅供複寫使用的資源上建立使用者 ID。下列範例適用於 RDS for MySQL 或外部 MySQL 來源資料庫。

```
mysql> CREATE USER 'repl_user'@'domain_name' IDENTIFIED BY 'password';
```

對於 Aurora MySQL 來源資料庫，skip_name_resolve 資料庫叢集參數設定為 1 (ON) 且無法修改，因此您必須使用主機的 IP 位址，而非網域名稱。如需詳細資訊，請參閱 MySQL 文件中的 skip_name_resolve。

```
mysql> CREATE USER 'repl_user'@'IP_address' IDENTIFIED BY 'password';
```

使用者需要 REPLICATION CLIENT 和 REPLICATION SLAVE 權限。授予這些權限給該使用者。

如果需要使用加密複寫，請對複寫使用者要求 SSL 連接。例如，您可以使用以下其中一個陳述式來對使用者帳戶 repl_user 要求 SSL 連線。

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'IP_address';
```

```
GRANT USAGE ON *.* TO 'repl_user'@'IP_address' REQUIRE SSL;
```

> **Note**
>
> 如果未包含 REQUIRE SSL，則複寫連線可能會以無訊息方式回復為未加密的連線。

6. 在複本目標上開啟複寫

開啟複寫之前，建議您手動取得 Aurora MySQL 資料庫叢集的快照或 RDS for MySQL 資料庫執行個體複本目標。如果發生問題，而您需要使用資料庫叢集或資料庫執行個體複本目標重新建立複寫，則可以從此快照還原資料庫叢集或資料庫執行個體，而不需再次將資料匯入至複本目標。

使用以下開啟資料庫引擎複寫功能的相關指示。

資料庫引擎	指示
Aurora MySQL	從 Aurora MySQL 資料庫叢集開啟複寫 1. 尋找複寫的起始位置。您需要 binlog 檔案名稱和 binlog 位置。 若您的資料庫叢集複本目標是從下列項目建立：

資料庫引擎	指示

- 資料庫叢集快照或複製 — 從新資料庫叢集的最近事件中，擷取 binlog 檔案名稱和位置，如 3. 建立複寫來源的複製或傾印 中所示。
- 資料庫快照 – 建立複寫來源的快照時，您已從 `SHOW SLAVE STATUS` (Aurora MySQL 第 2 版) 或 `SHOW REPLICA STATUS` (Aurora MySQL 第 3 版) 命令擷取 binlog 檔案名稱和位置。

2. 連線至資料庫叢集，並呼叫下列程序，以使用來自上一個步驟的二進位日誌檔名稱和位置，搭配您的複寫來源開始複寫：
   - mysql.rds_set_external_source (Aurora MySQL 第 3 版)
   - mysql.rds_set_external_master (Aurora MySQL 第 2 版)
   - mysql.rds_start_replication (所有版本)

   下列範例適用於 Aurora MySQL 第 3 版。

```
CALL mysql.rds_set_external_source ('mydbinstance.123456789012
.us-east-1.rds.amazonaws.com', 3306,
    'repl_user', 'password', 'mysql-bin-changelog.000031', 107,
 0);
CALL mysql.rds_start_replication;
```

若要使用 SSL 加密，請將最終值設定為 `1`，而不是 `0`。

資料庫引擎	指示
RDS for MySQL	從 Amazon RDS 資料庫執行個體開啟複寫 1. 如果資料庫執行個體複本目標是從資料庫快照建立，那麼您需要用於複寫開始位置的 binlog 檔案和 binlog 位置。建立複寫來源的快照時，您已從 `SHOW SLAVE STATUS` (Aurora MySQL 第 2 版) 或 `SHOW REPLICA STATUS` (Aurora MySQL 第 3 版) 命令擷取這些值。 2. 連接至資料庫執行個體，並呼叫 mysql.rds_set_external_master (Aurora MySQL 第 2 版) 或 mysql.rds_set_external_source (Aurora MySQL 第 3 版) 和 mysql.rds_start_replication 程序，開始以您的複寫來源進行複寫。使用上一個步驟的二進位日誌檔案名稱和位置。下列是 範例。 `CALL mysql.rds_set_external_master ('mydbcluster.cluster-12345 6789012.us-east-1.rds.amazonaws.com', 3306, 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0); CALL mysql.rds_start_replication;` 若要使用 SSL 加密，請將最終值設定為 `1`，而不是 `0`。

資料庫引擎	指示

MySQL (外部)	從外部 MySQL 資料庫開啟複寫
	1. 擷取屬於複寫開始位置的 binlog 檔案和 binlog 位置。建立複寫來源的快照時，您已從 SHOW SLAVE STATUS (Aurora MySQL 第 2 版) 或 SHOW REPLICA STATUS (Aurora MySQL 第 3 版) 命令擷取這些值。如果您的外部 MySQL 複本目標是從 mysqldump 命令搭配 --master-data=2 選項的輸出填入，則會在輸出中包含 binlog 檔案和 binlog 位置。下列是 範例。
	`-- -- Position to start replication or point-in-time recovery from -- -- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin-changelog.000031', MASTER_LOG_POS=107;`
	2. 連接至外部 MySQL 複本目標，並發出 CHANGE MASTER TO 和 START SLAVE (Aurora MySQL 第 2 版) 或 START REPLICA (Aurora MySQL 第 3 版)，以使用來自上一個步驟的二進位日誌檔案名稱和位置對複寫來源開始複寫，例如：
	`CHANGE MASTER TO MASTER_HOST = 'mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com', MASTER_PORT = 3306, MASTER_USER = 'repl_user', MASTER_PASSWORD = 'password', MASTER_LOG_FILE = 'mysql-bin-changelog.000031', MASTER_LOG_POS = 107; -- And one of these statements depending on your engine version: START SLAVE; -- Aurora MySQL version 2 START REPLICA; -- Aurora MySQL version 3`

如果複寫失敗，可能會導致複本上的非預期輸入/輸出大幅增加，進而降低效能。如果複寫失敗或不再需要複寫，可以執行 mysql.rds_reset_external_master (Aurora MySQL 第 2 版) 或 mysql.rds_reset_external_source (Aurora MySQL 第 3 版) 預存程序來移除複寫組態。

設定位置以停止僅供讀取複本的複寫作業

在 Aurora MySQL 3.04 版及更新版本中，您可以使用 mysql.rds_start_replication_until(Aurora MySQL 第 3 版） 預存程序開始複寫，然後在指定的二進位日誌檔案位置讓它停止。

啟動僅供讀取複本的複寫作業，並在特定位置停止複寫

1. 透過 MySQL 用戶端，以主要使用者的身分連接至複本 Aurora MySQL 資料庫叢集。
2. 執行 mysql.rds_start_replication_until(Aurora MySQL 第 3 版） 預存程序。

   以下範例會啟動複寫並複寫變更，直到達到 120 二進位日誌檔案中的位置 mysql-bin-changelog.000777 為止。若要使用災難復原功能，請在發生損毀前將位置預設為 120。

```
call mysql.rds_start_replication_until(
  'mysql-bin-changelog.000777',
  120);
```

達到停止點時，複寫作業即會自動停止。而且，系統還會產生以下 RDS 事件：Replication has been stopped since the replica reached the stop point specified by the rds_start_replication_until stored procedure。

如果您使用 GTID 式複寫，請使用 mysql.rds_start_replication_until_gtid(Aurora MySQL 第 3 版） 預存程序，而非 mysql.rds_start_replication_until(Aurora MySQL 第 3 版） 預存程序。如需 GTID 式複寫的詳細資訊，請參閱使用 GTID 式複寫。

7. 監控複本

設定使用 Aurora MySQL 資料庫叢集的 MySQL 複寫時，當 Aurora MySQL 資料庫叢集為複本目標時，您必須監控其容錯移轉事件。若發生容錯移轉，則屬於複本目標的資料庫叢集可能會以不同的網路地址，在新主機上重新建立。如需如何監控容錯移轉事件的資訊，請參閱使用 Amazon RDS 事件通知。

您也可以監控複本目標落後複寫來源的程度，方法是連接至複本目標並執行 SHOW SLAVE STATUS (Aurora MySQL 第 2 版) 或 SHOW REPLICA STATUS (Aurora MySQL 第 3 版) 命令。在命令輸出中，Seconds Behind Master 欄位會告知您複本目標落後於來源的程度。

> **Important**
>
> 若您升級資料庫叢集並指定自訂參數群組，請務必在升級完成後手動重新啟動叢集。這樣做可讓叢集使用您的新自訂參數設定，並重新開始 binlog 複寫。

同步複寫來源和目標之間的密碼

當您使用 SQL 陳述式變更複寫來源上的使用者帳戶和密碼時，這些變更會自動複寫到複寫目標。

如果您使用 AWS 管理主控台 AWS CLI、 或 RDS API 變更複寫來源上的主密碼，這些變更不會自動複寫到複寫目標。如果您要同步來源系統和目標系統之間的主要使用者和主要密碼，則必須自行對複寫目標進行相同的變更。

停止 Aurora MySQL 的二進位日誌複寫

若要停止 MySQL 資料庫執行個體、外部 MySQL 資料庫，或另一個 Aurora 資料庫叢集的二進位日誌複寫，請遵循這些步驟，本主題後面將詳細討論此部分。

1. 在複本目標上停止二進位日誌複寫

2. 在複寫來源上關閉二進位日誌

1. 在複本目標上停止二進位日誌複寫

請遵循下列指示，為資料庫引擎停止二進位日誌複寫。

資料庫引擎	指示
Aurora MySQL	在 Aurora MySQL 資料庫叢集複本目標上停止二進位日誌複寫 連接至屬於複本目標的 Aurora 資料庫叢集，並呼叫 mysql.rds_stop_replication 程序。
RDS for MySQL	在 Amazon RDS 資料庫執行個體上停止二進位日誌複寫 連接至屬於複本目標的 RDS 資料庫執行個體，並呼叫 mysql.rds_stop_replication 程序。
MySQL (外部)	在外部 MySQL 資料庫上停止二進位日誌複寫

資料庫引擎	指示
	連線到 MySQL 資料庫並執行 `STOP SLAVE` (第 5.7 版) 或 `STOP REPLICA` (第 8.0 版) 命令。

2. 在複寫來源上關閉二進位日誌

請遵循下表指示，為資料庫引擎關閉複寫來源上的二進位日誌。

資料庫引擎	指示
Aurora MySQL	在 Amazon Aurora 資料庫叢集上關閉二進位日誌 1. 連線至做為複寫來源的 Aurora 資料庫叢集。 2. 使用 mysql.rds_set_configuration 程序並以值 `NULL` 指定組態參數 `binlog retention hours`，如下範例所示。 `CALL mysql.rds_set_configuration('binlog retention hours', NULL);` Note 不可針對 `binlog retention hours` 使用值 `0`。 3. 在複寫來源上將 `binlog_format` 參數設定為 `OFF`。`binlog_format` 參數群組位於自訂資料庫叢集參數群組中，而該群組與您的資料庫叢集相關聯。 變更 `binlog_format` 參數值之後，將資料庫叢集重新開機，以讓變更生效。 如需詳細資訊，請參閱Amazon Aurora 資料庫叢集和資料庫執行個體參數及修改 Amazon Aurora 中的資料庫參數群組中的參數。
RDS for MySQL	在 Amazon RDS 資料庫執行個體上關閉二進位日誌 您不可以直接為 Amazon RDS 資料庫執行個體關閉二進位日誌，但您可以透過以下方式關閉它：

資料庫引擎	指示
	1. 為資料庫執行個體關閉自動備份。您可以關閉自動備份，方法是修改現有的資料庫執行個體，並將 Backup Retention Period (備份保留期間) 設定為 0。如需詳細資訊，請參閱《Amazon Relational Database Service 使用者指南》中的修改執行 Amazon RDS 資料庫執行個體和使用備份。 2. 刪除資料庫執行個體的所有僅供讀取複本。如需詳細資訊，請參閱《Amazon Relational Database Service 使用者指南》中的使用 MariaDB、MySQL 及 PostgreSQL 資料庫執行個體的僅供讀取複本。

MySQL (外部)	在外部 MySQL 資料庫上關閉二進位日誌
	連接至 MySQL 資料庫並呼叫 `STOP REPLICATION` 命令。
	1. 從命令 shell 停止 `mysqld` 服務，
	`sudo service mysqld stop`
	2. 編輯 `my.cnf` 檔案 (此檔案通常位於 `/etc` 之下)。
	`sudo vi /etc/my.cnf`
	從 `log_bin` 區段刪除 `server_id` 和 `[mysqld]` 選項。
	如需更多詳細資訊，請參閱 MySQL 文件中的Setting the replication source configuration。
	3. 啟動 mysql 服務。
	`sudo service mysqld start`

使用 Amazon Aurora 為 MySQL 資料庫擴展讀取

您可以使用 Amazon Aurora 搭配 MySQL 資料庫執行個體來利用 Amazon Aurora 的讀取擴展功能，並為 MySQL 資料庫執行個體擴展讀取工作負載。若要使用 Aurora 來讀取擴展 MySQL 資料庫執行個體，請建立 Amazon Aurora MySQL 資料庫叢集，並讓它成為您 MySQL DB 執行個體的讀取複本。這可套用至 RDS for MySQL 資料庫執行個體，或在 Amazon RDS 外部執行的 MySQL 資料庫。

如需建立 Amazon Aurora 資料庫叢集的詳細資訊，請參閱建立 Amazon Aurora 資料庫叢集。

設定 MySQL 資料庫執行個體與 Amazon Aurora 資料庫叢集之間的複寫時，務必遵循這些準則：

- 參考 Amazon Aurora MySQL 資料庫叢集時，請使用 Amazon Aurora 資料庫叢集端點地址。如果發生容錯移轉，則提升至 Aurora MySQL 資料庫叢集主要執行個體的 Aurora 複本，將繼續使用資料庫叢集端點地址。
- 直到您已驗證二進位日誌已套用至 Aurora 複本前，都要將二進位日誌保存在寫入器執行個體上。如此一來，發生故障時，您就可以還原寫入器執行個體。

> **Important**
>
> 使用自我管理的複寫時，您需負責監控和解決可能發生的任何複寫問題。如需詳細資訊，請參閱診斷和解決僅供讀取複本之間的延遲。

> **Note**
>
> 在 Aurora MySQL 資料庫叢集上啟動複寫功能所需的許可受到限制，並無法供 Amazon RDS 主要使用者使用。因此，您必須使用 mysql.rds_set_external_master (Aurora MySQL 第 2 版) 或 mysql.rds_set_external_source (Aurora MySQL 第 3 版) 和 mysql.rds_start_replication 程序，設定 Aurora MySQL 資料庫叢集與 MySQL 資料庫執行個體之間的複寫。

啟動外部來源執行個體與 Aurora MySQL 資料庫叢集之間的複寫

1. 將來源 MySQL 資料庫執行個體設成唯讀：

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

2. 在來源 MySQL 資料庫執行個體上執行 SHOW MASTER STATUS 命令，以確定二進位記錄檔的位置。您會獲得類似下列範例的輸出結果：

```
File                        Position
------------------------------------
 mysql-bin-changelog.000031      107
------------------------------------
```

3. 使用 mysqldump，從外部 MySQL 資料庫執行個體將資料庫複製到 Amazon Aurora MySQL 資料庫叢集。若為非常大型的資料庫，您可能想要使用《Amazon Relational Database Service 使用者指南》中將資料匯入至 Amazon RDS for MySQL 資料庫，並減少停機時間中的程序。

   對於 Linux、macOS 或 Unix：

```
mysqldump \
    --databases <database_name> \
    --single-transaction \
    --compress \
    --order-by-primary \
    -u local_user \
    -p local_password | mysql \
        --host aurora_cluster_endpoint_address \
        --port 3306 \
        -u RDS_user_name \
        -p RDS_password
```

   在 Windows 中：

```
mysqldump ^
    --databases <database_name> ^
    --single-transaction ^
    --compress ^
    --order-by-primary ^
    -u local_user ^
    -p local_password | mysql ^
        --host aurora_cluster_endpoint_address ^
        --port 3306 ^
        -u RDS_user_name ^
        -p RDS_password
```

   > **Note**
   >
   > 請注意 -p 選項與輸入的密碼之間不能有空格。

   在 --host 命令中，使用 --user (-u)、--port、-p 和 mysql 選項來指定主機名稱、使用者名稱、連接埠和密碼，以連接至 Aurora 資料庫叢集。主機名稱是來自 Amazon Aurora 資料庫叢集端點的 DNS 名稱，例如 mydbcluster.cluster-123456789012.us-

east-1.rds.amazonaws.com。您可在 Amazon RDS 管理主控台中的叢集詳細資訊中，找到端點值。

4. 將來源 MySQL 資料庫執行個體重新設為可寫入：

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

如需如何製作備份以搭配複寫作業使用的詳細資訊，請參閱 MySQL 文件中的 Backing up a source or replica by making it read only。

5. 在 Amazon RDS 管理主控台，將託管來源 MySQL 資料庫之伺服器的 IP 地址，新增至 Amazon Aurora 資料庫叢集的 VPC 安全群組。如需有關修改 VPC 安全群組的詳細資訊，請參閱《Amazon Virtual Private Cloud 使用者指南》中的 VPC 安全群組。

您可能還需要設定本機網路，以允許來自 Amazon Aurora 資料庫叢集之 IP 地址的連線，這樣就能與來源 MySQL 執行個體通訊。若要找出 Amazon Aurora 資料庫叢集的 IP 地址，請使用 host 命令。

```
host aurora_endpoint_address
```

主機名稱是來自 Amazon Aurora 資料庫叢集端點的 DNS 名稱。

6. 使用您選擇的用戶端，連接至外部 MySQL 執行個體，並建立用於複寫的 MySQL 使用者。此帳戶只供複寫作業使用，務必限制其存取您的網域，以提升安全性。以下是範例。

```
CREATE USER 'repl_user'@'example.com' IDENTIFIED BY 'password';
```

7. 若為外部 MySQL 執行個體，請將 REPLICATION CLIENT 和 REPLICATION SLAVE 權限授予複寫使用者。舉例來說，若要將所有資料庫的 REPLICATION CLIENT 和 REPLICATION SLAVE 權限授予您網域中的「repl_user」使用者，請發出下列命令。

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'example.com'
    IDENTIFIED BY 'password';
```

8. 設定複寫之前，手動取得要成為讀取複本之 Aurora MySQL 資料庫叢集的快照。如果您需要對資料庫叢集重新建立複寫做為讀取複本，您可以從這個快照還原 Aurora MySQL 資料庫叢集，而不需從 MySQL 資料庫執行個體匯入資料至新的 Aurora MySQL 資料庫叢集。

9. 將 Amazon Aurora 資料庫叢集變成複本。以主要使用者身分連接至 Amazon Aurora 資料庫叢集，然後使用 mysql.rds_set_external_master (Aurora MySQL 第 2 版) 或

mysql.rds_set_external_source (Aurora MySQL 第 3 版) 和 mysql.rds_start_replication 程序，將來源 MySQL 資料庫辨識為複寫來源。

使用您在步驟 2 中所確定的 binlog 檔案名稱與位置。下列是 範例。

```
For Aurora MySQL version 2:
CALL mysql.rds_set_external_master ('mymasterserver.example.com', 3306,
    'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);

For Aurora MySQL version 3:
CALL mysql.rds_set_external_source ('mymasterserver.example.com', 3306,
    'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);
```

10. 在 Amazon Aurora 資料庫叢集上，呼叫 mysql.rds_start_replication 程序以開始複寫。

```
CALL mysql.rds_start_replication;
```

建立來源 MySQL 資料庫執行個體與 Amazon Aurora 資料庫叢集之間的複寫之後，您可以將 Aurora 複本新增至 Amazon Aurora 資料庫叢集。您接著可以連接至 Aurora 複本來讀取擴展您的資料。如需建立 Aurora 複本的詳細資訊，請參閱將 Aurora 複本新增至資料庫叢集。

最佳化 Aurora MySQL 的二進位日誌複寫

接下來，您可以學習如何最佳化二進位日誌複寫效能，並對 Aurora MySQL 中的相關問題進行疑難排解。

> **Tip**
>
> 本討論假設您熟悉 MySQL 二進位日誌複寫機制及其運作方式。如需背景資訊，請參閱 MySQL 文件中的複寫實作。

多執行緒二進位日誌複寫

使用多執行緒二進位日誌複寫，SQL 執行緒會從轉送日誌讀取事件，並將它們排入佇列，以供 SQL 工作者執行緒套用。SQL 工作者執行緒是由協調器執行緒管理。可能的話，會平行套用二進位日誌事件。平行處理的程度取決於版本、參數、結構描述設計和工作負載特性等因素。

Aurora MySQL 第 3 版和 Aurora MySQL 第 2.12.1 版及更新版本支援多執行緒二進位日誌複寫。若要讓多執行緒複本有效平行處理 binlog 事件，您必須設定多執行緒二進位日誌複寫的來源，且來源必須使用內含其二進位日誌檔案上平行處理資訊的版本。

將 Aurora MySQL 資料庫執行個體設定為使用二進位日誌複寫時，複本執行個體預設會使用低於 3.04 版 Aurora MySQL 的單一執行緒複寫。若要啟用多執行緒複寫，您可以在自訂參數群組中將 `replica_parallel_workers` 參數更新為大於 1 的值。

對於 Aurora MySQL 3.04 版及更新版本，複寫預設為多執行緒，且 `replica_parallel_workers` 會設為 4。您可以在自訂參數群組中修改此參數。

若要提高資料庫的彈性，以防止意外停止，建議您在來源啟用 GTID 複寫，然後允許在複本的 GTID。若要允許 GTID 複寫，請在來源和複本將 `gtid_mode` 設為 `ON_PERMISSIVE`。如需 GTID 式複寫的詳細資訊，請參閱使用 GTID 式複寫。

下列組態選項可協助您微調多執行緒複寫。如需用法資訊，請參閱《MySQL 參考手冊》中的複寫和二進位記錄選項和變數。如需多執行緒複寫的詳細資訊，請參閱 MySQL 部落格使用寫入集型相依性追蹤改善平行套用者。

最佳參數值取決於幾個因素。例如，二進位日誌複寫的效能會受到資料庫工作負載特性和複本執行所在的資料庫執行個體類別影響。因此，建議您在將新參數設定套用至生產執行個體之前，先徹底測試這些組態參數的所有變更：

- `binlog_format recommended value`：設定為 `ROW`
- `binlog_group_commit_sync_delay`
- `binlog_group_commit_sync_no_delay_count`
- `binlog_transaction_dependency_history_size`
- `binlog_transaction_dependency_tracking`：建議值為 `WRITESET`
- `replica_preserve_commit_order`
- `replica_parallel_type`：建議值為 `LOGICAL_CLOCK`
- `replica_parallel_workers`
- `replica_pending_jobs_size_max`
- `transaction_write_set_extraction`：建議值為 `XXHASH64`

結構描述和工作負載特性是影響平行複寫的因素。最常見的因素如下。

- 沒有主索引鍵：RDS 無法為沒有主索引鍵的資料表建立寫入集相依性。使用 ROW 格式時，單一多列陳述式可以透過來源的單一完整資料表掃描來完成，但在複本會導致每修改一列就執行一次完整資料表掃描。缺少主索引鍵會大幅減少複寫輸送量。
- 存在外部索引鍵：如果存在外部索引鍵，Amazon RDS 就無法使用寫入集相依性，來平行處理具有 FK 關係的資料表。
- 交易大小：如果單一交易跨越數十或數百 MB 或 GB，協調器執行緒和其中一個工作者執行緒可能會需要很長的時間，才能處理該交易。在此期間，所有其他工作者執行緒可能會在完成其先前交易的處理後保持閒置狀態。

在 Aurora MySQL 第 3.06 版及更新版本中，當複寫具有多個次要索引的大型資料表的交易時，您就可以改善二進位日誌複本的效能。此功能會導入執行緒集區，以在 binlog 複本平行套用次要索引變更。此功能由 aurora_binlog_replication_sec_index_parallel_workers 資料庫叢集參數所控制，該參數會控制可用於套用次要索引變更的平行執行緒總數。依預設，參數會設為 0 (停用)。啟用此功能不需要重新啟動執行個體。若要啟用此功能，請停止持續複寫、設定所需的平行工作者執行緒數量，然後再次開始複寫。

最佳化 binlog 複寫

在 Aurora MySQL 2.10 及更高版本中，Aurora 會自動將名為 binlog 輸入/輸出快取的最佳化套用至二進位日誌複寫。藉由快取最近遞交的 binlog 事件，此最佳化旨在改善 binlog 傾印執行緒效能，同時限制 binlog 來源執行個體上對前景交易的影響。

> **Note**
>
> 此功能所使用的記憶體與 MySQL binlog_cache 設定無關。
> 此功能不適用於使用 db.t2 和 db.t3 執行個體類別的 Aurora 資料庫執行個體。

您不需要調整任何組態參數，即可開啟此最佳化。特別是如果您在 Aurora MySQL 的較舊版本中，將組態參數 aurora_binlog_replication_max_yield_seconds 調整為非零值，請將其設定回零以取得目前可用的版本。

這些狀態變數 aurora_binlog_io_cache_reads 和 aurora_binlog_io_cache_read_requests 可協助您監控從 binlog 輸入/輸出快取讀取資料的頻率。

- aurora_binlog_io_cache_read_requests 顯示對快取發起的 binlog 輸入/輸出讀取要求的次數。

- aurora_binlog_io_cache_reads 顯示從快取擷取資訊的 binlog 輸入/輸出讀取次數。

下列 SQL 查詢會計算利用快取資訊的 binlog 讀取要求百分比。在這種情況下，比例越接近 100 越好。

```
mysql> SELECT
  (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME='aurora_binlog_io_cache_reads')
  / (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME='aurora_binlog_io_cache_read_requests')
  * 100
  as binlog_io_cache_hit_ratio;
+---------------------------+
| binlog_io_cache_hit_ratio |
+---------------------------+
|         99.99847949080622 |
+---------------------------+
```

binlog 輸入/輸出快取功能也包含與 binlog 傾印執行緒相關的新指標。「傾印執行緒」是新的 binlog 複本連線到 binlog 來源執行個體時所建立的執行緒。

傾印執行緒指標會每隔 60 秒列印至資料庫日誌，字首為 [Dump thread metrics]。此指標包括每個 binlog 複本的資訊，例如 Secondary_id、Secondary_uuid、binlog 檔案名稱，以及每個複本正在讀取的位置。此指標也包括 Bytes_behind_primary，表示複寫來源與複本之間的距離 (以位元組為單位)。此指標會測量複本輸入/輸出執行緒的延遲。該圖與複本 SQL 套用者執行緒的延遲不同，此執行緒由 binlog 複本上的 seconds_behind_master 指標表示。您可以透過檢查距離是否減少或增加，判斷 binlog 複本是趕上還是落後於來源。

記憶體中轉送日誌

在 Aurora MySQL 第 3.10 版及更新版本中，Aurora 推出名為記憶體中轉送日誌的最佳化，以改善複寫輸送量。此最佳化透過快取記憶體中所有中繼轉送日誌內容，來增強轉送日誌 I/O 效能。因此，其透過盡可能減少儲存體 I/O 操作來減少遞交延遲，因為轉送日誌內容仍可在記憶體中輕鬆存取。

根據預設，當複本符合下列任何組態時，系統會自動為 Aurora 受管複寫案例 (包括藍/綠部署、Aurora-Aurora 複寫和跨區域複本) 啟用記憶體中轉送日誌功能：

- 單執行緒複寫模式 (replica_parallel_workers = 0)
- 啟用 GTID 模式的多執行緒複寫：
  - 自動定位已啟用

  - 複本的 GTID 模式設為 ON
- 以檔案為基礎的複寫搭配 replica_preserve_commit_order = ON

大於 t3.large 的執行個體類別支援記憶體中轉送日誌功能，但無法在 Aurora Serverless 執行個體上使用。轉送日誌循環緩衝區的固定大小為 128 MB。若要監控此功能的記憶體使用量，您可以執行下列查詢：

```
SELECT event_name, current_alloc FROM sys.memory_global_by_current_bytes WHERE
 event_name = 'memory/sql/relaylog_io_cache';
```

記憶體中轉送日誌功能是由 aurora_in_memory_relaylog 參數控制，您可在資料庫叢集或執行個體層級設定該參數。您可以動態啟用或停用此功能，而無需重新啟動執行個體：

1. 停止持續複寫
2. 在參數群組中將 aurora_in_memory_relaylog 設定為 ON (啟用) 或 OFF (停用)
3. 重新啟動複寫

範例：

```
CALL mysql.rds_stop_replication;
set aurora_in_memory_relaylog to ON to enable or OFF to disable in cluster parameter
 group
CALL mysql.rds_start_replication;
```

即使將 aurora_in_memory_relaylog 設定為 ON，在特定情況下，記憶體中轉送日誌功能仍可能遭停用。若要驗證功能的目前狀態，您可以使用下列命令：

```
SHOW GLOBAL STATUS LIKE 'Aurora_in_memory_relaylog_status';
```

如果此功能意外停用，您可以透過執行下列動作來識別原因：

```
SHOW GLOBAL STATUS LIKE 'Aurora_in_memory_relaylog_disabled_reason';
```

此命令會傳回訊息，說明此功能目前停用的原因。

設定 Aurora MySQL 的增強型 binlog

增強型 Binlog 可減少開啟 Binlog 所造成的運算效能額外負荷，在某些情況下最高可達 50%。使用增強型 Binlog，此額外負荷可以減少至大約 13%。為了減少額外負荷，增強型 Binlog 會將二進位和交易日誌平行寫入至儲存體，這會將交易認可時寫入的資料降到最低。

相較於社群 MySQL Binlog，使用增強型 Binlog 還可以改善重新啟動和容錯移轉後的資料庫復原時間，最高可達 99%。增強型 Binlog 與現有的 Binlog 型工作負載相容，而且您與其互動的方式同於與社群 MySQL Binlog 互動的方式。

增強型 binlog 可在 Aurora MySQL 3.03.1 版及更新版本上使用。

主題

- 設定增強型 Binlog 參數
- 其他相關參數
- 增強型 Binlog 與社群 MySQL Binlog 之間的差異
- 增強型 Binlog 的 Amazon CloudWatch 指標
- 增強型 Binlog 限制

設定增強型 Binlog 參數

您可以透過開啟/關閉增強型 Binlog 參數，在社群 MySQL Binlog 與增強型 Binlog 之間切換。現有的 Binlog 取用者可以繼續讀取和取用 Binlog 檔案，而不會在 Binlog 檔案序列中有任何間隙。

若要開啟增強型 binlog，請設定下列參數：

參數	預設	描述
`binlog_format`	–	將 `binlog_format` 參數設定為您選擇的二進位記錄格式，以開啟增強型 Binlog。確定 `binlog_format parameter` 未設定為 OFF。如需詳細資訊，請參閱設定 Aurora MySQL 二進位記錄。
`aurora_enhanced_binlog`	`0`	將此參數的值設定為與 Aurora MySQL 叢集相關聯之資料

參數	預設	描述
		庫叢集參數群組中的 1。當您變更此參數的值時，必須在 DBClusterParameterGroupStatus 值顯示為 pending-reboot 時重新啟動寫入器執行個體。
binlog_backup	1	關閉此參數以開啟增強型 Binlog。若要這樣做，請將此參數的值設定為 0。
binlog_replication_globaldb	1	關閉此參數以開啟增強型 Binlog。若要這樣做，請將此參數的值設定為 0。

Important

僅在使用增強型 Binlog 時，您才能關閉 binlog_backup 和 binlog_replication_globaldb 參數。

若要關閉增強型 binlog，請設定下列參數：

參數	描述
aurora_enhanced_binlog	將此參數的值設定為與 Aurora MySQL 叢集相關聯之資料庫叢集參數群組中的 0。每當您變更此參數的值時，就必須在 DBClusterParameterGroupStatus 值顯示為 pending-reboot 時重新啟動寫入器執行個體。
binlog_backup	在您關閉增強型 Binlog 時開啟此參數。若要這樣做，請將此參數的值設定為 1。

參數	描述
binlog_replication_globaldb	在您關閉增強型 Binlog 時開啟此參數。若要這樣做，請將此參數的值設定為 1。

若要檢查增強型 Binlog 是否已開啟，請在 MySQL 用戶端中使用下列命令：

```
mysql>show status like 'aurora_enhanced_binlog';

+------------------------+--------+
| Variable_name          | Value  |
+------------------------+--------+
| aurora_enhanced_binlog | ACTIVE |
+------------------------+--------+
1 row in set (0.00 sec)
```

當增強型 Binlog 開啟時，輸出會針對 aurora_enhanced_binlog 顯示 ACTIVE

其他相關參數

當您開啟增強型 Binlog 時，下列參數會受到影響：

- max_binlog_size 參數可見，但無法修改。當增強型 Binlog 開啟時，它的預設值 134217728 會自動調整為 268435456。
- 與社群 MySQL Binlog 不同，當增強型 Binlog 開啟時，binlog_checksum 不會充當動態參數。若要使對此參數所做的變更生效，您必須手動重新啟動資料庫叢集，即使 ApplyMethod 為 immediate 也是如此。
- 增強型 Binlog 開啟時，您在 binlog_order_commits 參數上設定的值不會影響認可的順序。認可總是排序，沒有任何進一步的效能隱憂。

增強型 Binlog 與社群 MySQL Binlog 之間的差異

相較於社群 MySQL binlog，增強型 binlog 會以不同方式與複本，備份和 Aurora 全球資料庫互動。建議您在使用增強型 Binlog 之前先了解下列差異。

- 來源資料庫叢集中的增強型 binlog 檔案無法在複製的資料庫叢集上使用。

- Aurora 備份中不包含增強型 binlog 檔案。因此，在還原資料庫叢集之後，儘管在其上設定了任何保留期間，仍無法使用來源資料庫叢集中的增強型 Binlog 檔案。
- 與 Aurora 全球資料庫搭配使用時，主要資料庫叢集的增強型 Binlog 檔案不會複寫至次要區域中的資料庫叢集。

範例

下列範例說明增強型 Binlog 與社群 MySQL Binlog 之間的差異。

在還原或複製的資料庫叢集上

當增強型 Binlog 開啟時，在已還原或複製的資料庫叢集中無法使用歷史 Binlog 檔案。在還原或複製操作之後，如果開啟了 Binlog，新的資料庫叢集會開始寫入自己的 Binlog 檔案序列，從 1 (mysql-bin-changelog.000001) 開始。

若要在還原或複製操作之後開啟增強型 Binlog，請在還原或複製的資料庫叢集上設定所需的資料庫叢集參數。如需更多詳細資訊，請參閱 設定增強型 Binlog 參數。

Example範例：增強型 binlog 開啟時所執行的複製或還原操作

來源資料庫叢集：

```
mysql> show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        |
| mysql-bin-changelog.000003 |       156 | No        |
| mysql-bin-changelog.000004 |       156 | No        | --> Enhanced Binlog turned on
| mysql-bin-changelog.000005 |       156 | No        | --> Enhanced Binlog turned on
| mysql-bin-changelog.000006 |       156 | No        | --> Enhanced Binlog turned on
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

在還原或複製的資料庫叢集上，增強型 Binlog 開啟時，不會備份 Binlog 檔案。為了避免 Binlog 資料中的不連續性，在開啟增強型 Binlog 之前寫入的 Binlog 檔案也無法使用。

```
mysql>show binary logs;
```

```
+---------------------------+-----------+-----------+
| Log_name                  | File_size | Encrypted |
+---------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        | --> New sequence of Binlog files
+---------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Example範例：增強型 binlog 關閉時所執行的複製或還原操作

來源資料庫叢集：

```
mysql>show binary logs;

+---------------------------+-----------+-----------+
| Log_name                  | File_size | Encrypted |
+---------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000003 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+---------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

增強型 binlog 會在 `mysql-bin-changelog.000003` 之後停用。在還原或複製的資料庫叢集上，可以使用在關閉增強型 Binlog 之後寫入的 Binlog 檔案。

```
mysql>show binary logs;

+---------------------------+-----------+-----------+
| Log_name                  | File_size | Encrypted |
+---------------------------+-----------+-----------+
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+---------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

在 Amazon Aurora 全球資料庫上

在 Amazon Aurora 全球資料庫上，主要資料庫叢集的 Binlog 資料不會複寫到次要資料庫叢集。在跨區域容錯移轉程序之後，Binlog 資料無法在新提升的主要資料庫叢集中使用。如果開啟了 Binlog，新提升的資料庫叢集會啟動自己的 Binlog 檔案序列，從 1 (mysql-bin-changelog.000001) 開始。

若要在容錯移轉之後開啟增強型 Binlog，您必須在次要資料庫叢集上設定所需的資料庫叢集參數。如需更多詳細資訊，請參閱 設定增強型 Binlog 參數。

Example範例：開啟增強型 binlog 時，便會執行全球資料庫容錯移轉操作

舊的主要資料庫叢集 (容錯移轉前)：

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        |
| mysql-bin-changelog.000003 |       156 | No        |
| mysql-bin-changelog.000004 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000005 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000006 |       156 | No        | --> Enhanced Binlog enabled
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

新的主要資料庫叢集 (容錯移轉後)：

開啟增強型 Binlog 時，Binlog 檔案不會複寫到次要區域。為了避免 Binlog 資料中的不連續性，無法使用在開啟增強型 Binlog 之前寫入的 Binlog 檔案。

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        | --> Fresh sequence of Binlog
 files
```

```
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Example範例：關閉增強型 binlog 時，便會執行全球資料庫容錯移轉操作

來源資料庫叢集：

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000003 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

還原或複製的資料庫叢集：

增強型 binlog 會在 `mysql-bin-changelog.000003` 之後停用。在關閉增強型 Binlog 之後寫入的 Binlog 檔案會被複寫，並可在新提升的資料庫叢集中使用。

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
3 rows in set (0.00 sec)
```

增強型 Binlog 的 Amazon CloudWatch 指標

只在開啟增強型 Binlog 時，才會發佈下列 Amazon CloudWatch 指標。

CloudWatch 指標	描述	個單位
ChangeLogBytesUsed	增強型 Binlog 使用的儲存體數量。	位元組
ChangeLogReadIOPs	增強型 Binlog 中 5 分鐘間隔內執行的讀取 I/O 操作次數。	每 5 分鐘計數
ChangeLogWriteIOPs	增強型 Binlog 中 5 分鐘間隔內執行的寫入磁碟 I/O 操作次數。	每 5 分鐘計數

增強型 Binlog 限制

當增強型 Binlog 開啟時，下列限制適用於 Amazon Aurora 資料庫叢集。

- 僅在 Aurora MySQL 3.03.1 版及更新版本上，才支援增強型 binlog。
- 在主要資料庫叢集上寫入的增強型 Binlog 檔案不會複製到複製或還原的資料庫叢集。
- 與 Amazon Aurora 全球資料庫搭配使用時，主要資料庫叢集的增強型 Binlog 檔案不會複寫至次要資料庫叢集。因此，在容錯移轉程序之後，無法在新的主要資料庫叢集中使用歷史 Binlog 資料。
- 系統會忽略下列 Binlog 組態參數：
  - `binlog_group_commit_sync_delay`
  - `binlog_group_commit_sync_no_delay_count`
  - `binlog_max_flush_queue_time`
- 您無法捨棄或重新命名資料庫中損毀的資料表。若要捨棄這些資料表，您可以聯絡 支援。
- 開啟增強型 Binlog 時，會停用 Binlog I/O 快取。如需更多詳細資訊，請參閱 最佳化 Aurora MySQL 的二進位日誌複寫。

> **Note**
>
> 增強型 Binlog 提供與 Binlog I/O 快取類似的讀取效能改進，以及更好的寫入效能改進。

- 不支援恢復功能。無法在下列情況下的資料庫叢集中開啟增強型 Binlog：
  - 目前已啟用恢復功能的資料庫叢集。
  - 先前已啟用但現在已停用恢復功能的資料庫叢集。
  - 從來源資料庫叢集或已啟用恢復功能的快照中還原的資料庫叢集。

使用 GTID 式複寫

接下來，您可以了解如何在 Aurora MySQL 叢集和外部來源間搭配使用全域交易識別符 (GTID) 與二進位日誌 (binlog) 複寫。

> **Note**
>
> 若為 Aurora，您僅可對會在外部 MySQL 資料庫間進行 binlog 複寫的 Aurora MySQL 叢集使用這項功能。其他資料庫可能是 Amazon RDS MySQL 執行個體、內部部署的 MySQL 資料庫，或是位於不同 AWS 區域 中的 Aurora 資料庫叢集。若要了解如何設定該類型的複寫作業，請參閱Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。

如果您是使用 binlog 複寫，且不熟悉如何在 MySQL 使用 GTID 式複寫，則可參閱 MySQL 文件中的使用全域交易識別符的複寫。

Aurora MySQL 第 2 版和第 3 版不支援 GTID 型複寫。

主題

- 全域交易識別符 (GTID) 的概觀
- GTID 式複寫的參數
- 為 Aurora MySQL 叢集啟用 GTID 式複寫
- 為Aurora MySQL 資料庫叢集停用 GTID 式複寫

全域交易識別符 (GTID) 的概觀

「全域交易識別符 (GTID)」是系統為遞交的 MySQL 交易所產生的唯一識別符。GTID 能讓 binlog 複寫的操作更簡單，也更容易進行故障診斷。

> **Note**
>
> 當 Aurora 在叢集中的資料庫執行個體間同步資料時，該複寫機制並不會用到二進位日誌 (binlog)。在 Aurora MySQL 中，唯有您同時使用 binlog 複寫功能來複寫資料至外部 MySQL 相容資料庫的 Aurora MySQL 資料庫叢集，或從中複寫資料的情況下，才能套用 GTID 式複寫。

進行 binlog 複寫作業時，MySQL 會使用兩種不同類型的交易：

- GTID 交易 – 透過 GTID 識別的交易。
- 匿名交易 – 未指派 GTID 的交易。

在複寫組態中，全部的資料庫執行個體都有各自不同的 GTID。GTID 可簡化複寫組態，因為使用時不需要參照日誌檔案位置。GTID 也使得追蹤複寫的交易更容易，而且可決定來源執行個體和複本是否一致。

一般而言，當您從外部 MySQL 相容資料庫複寫至 Aurora 叢集時，即可對 Aurora 叢集使用 GTID 式複寫。您可以設定這個複寫組態，將其做為從現場部署資料庫或 Amazon RDS 資料庫到 Aurora MySQL 遷移流程的一部分。如果外部資料庫已使用 GTID，則可為 Aurora 叢集啟用 GTID 式複寫功能，藉此簡化複寫程序。

若要為 Aurora MySQL 叢集設定 GTID 式複寫，您首先要在資料庫叢集參數群組中設定相關的組態參數。接著，請建立該參數群組與叢集間的關聯。

GTID 式複寫的參數

使用以下參數來設定 GTID 式複寫。

參數	有效值	描述
`gtid_mode`	`OFF`, `OFF_PERMISSIVE` ，`ON_PERMISSIVE` ，`ON`	`OFF` 指定新交易是匿名交易 (也就是沒有 GTID)，而且交易必須是匿名交易才能複寫。 `OFF_PERMISSIVE` 指定新交易是匿名交易，但全部交易都可以複寫。 `ON_PERMISSIVE` 指定新交易是 GTID 交易，而且全部交易都可以複寫。

參數	有效值	描述
		ON 指定新交易是 GTID 交易，而且交易必須是 GTID 交易才能複寫。
enforce_gtid_consistency	OFF, ON, WARN	OFF 允許交易違反 GTID 一致性。 ON 可避免交易違反 GTID 一致性。 WARN 允許交易違反 GTID 一致性，但會在出現違反行為時產生警告。

> **Note**
>
> 在 AWS 管理主控台 中，gtid_mode 參數會顯示為 gtid-mode。

進行 GTID 式複寫時，您可以使用下列設定來配置 Aurora MySQL 資料庫叢集的資料庫叢集參數群組：

- ON 和 ON_PERMISSIVE 僅適用於從 Aurora MySQL 叢集進行的傳出複寫作業。這兩個值都會導致 Aurora 資料庫叢集使用 GTID 來執行複寫至外部資料庫的交易。如果設定 ON，外部資料庫需同時使用 GTID 式複寫。如果設定 ON_PERMISSIVE，則外部資料庫不一定要使用 GTID 式複寫。
- 若設定 OFF_PERMISSIVE，表示 Aurora 資料庫叢集可接受來自外部資料庫的傳入複寫作業。無論外部資料庫是否使用 GTID 式複寫，此值都能實現這項功能。
- 若設定 OFF，表示唯有在外部資料庫未使用 GTID 式複寫的情況下，Aurora 資料庫叢集才能接受來自該資料庫的傳入複寫作業。

> **Tip**
>
> 傳入複寫是 Aurora MySQL 叢集最常見的 binlog 複寫案例。如果要進行傳入複寫，建議您將 GTID 模式設定為 OFF_PERMISSIVE。該設定可允許來自外部資料庫的傳入複寫作業，不管複寫來源的 GTID 設定為何都是如此。

如需參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。

為 Aurora MySQL 叢集啟用 GTID 式複寫

當 Aurora MySQL 資料庫叢集的 GTID 式複寫處於啟用狀態時，即可將 GTID 設定套用至傳入和傳出 binlog 複寫作業。

為 Aurora MySQL 叢集啟用 GTID 式複寫

1. 使用以下參數設定來建立或編輯資料庫叢集參數群組：
   - `gtid_mode` – `ON` 或 `ON_PERMISSIVE`
   - `enforce_gtid_consistency` – `ON`
2. 建立該資料庫叢集參數群組與 Aurora MySQL 叢集間的關聯。若要執行此作業，請遵循Amazon Aurora 的參數群組中的程序。
3. (選用) 指定如何將 GTID 指派給不包含 GTID 的交易。若要這樣做，請呼叫 mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL 第 3 版) 中的預存程序。

為Aurora MySQL 資料庫叢集停用 GTID 式複寫

您可以為 Aurora MySQL 資料庫叢集停用 GTID 式複寫。如此一來，Aurora 叢集就無法透過採用 GTID 式複寫的外部資料庫執行傳入或傳出 binlog 複寫作業。

> **Note**
>
> 在下列程序中，「僅供讀取複本」指的是 Aurora 組態中的複寫目標，其會在外部資料庫間進行 binlog 複寫。僅供讀取複本並不是指唯讀 Aurora 複本資料庫執行個體。舉例來說，Aurora 叢集接受來自外部來源的傳入複寫作業時，Aurora 主要執行個體會以僅供讀取複本運作，供 binlog 複寫。

如需本節中提及的預存程序詳細資訊，請參閱 Aurora MySQL 預存程序參考。

為 Aurora MySQL 資料庫叢集 停用 GTID 式複寫

1. 在 Aurora 複本，執行下列程序：

   對於第 3 版

   ```
   CALL mysql.rds_set_source_auto_position(0);
   ```

對於第 2 版

```
CALL mysql.rds_set_master_auto_position(0);
```

2. 將 gtid_mode 重設為 ON_PERMISSIVE。

   a. 確定與 Aurora MySQL 叢集相關聯的資料庫叢集參數群組有設定為 gtid_mode 的 ON_PERMISSIVE。

      如需使用參數群組設定組態參數的詳細資訊，請參閱Amazon Aurora 的參數群組。

   b. 重新啟動 Aurora MySQL 資料庫叢集。

3. 將 gtid_mode 重設為 OFF_PERMISSIVE。

   a. 確定與 Aurora MySQL 叢集相關聯的資料庫叢集參數群組有設定為 gtid_mode 的 OFF_PERMISSIVE。

   b. 重新啟動 Aurora MySQL 資料庫叢集。

4. 等待系統將全部的 GTID 交易套用至 Aurora 主要執行個體。若要檢查這些設定是否已套用，請執行下列步驟：

   a. 在 Aurora 主要執行個體上，執行 SHOW MASTER STATUS 命令。

      輸出內容應會與下列輸入類似。

```
File                        Position
------------------------------------
mysql-bin-changelog.000031      107
------------------------------------
```

      請注意輸出中的檔案和位置。

   b. 對於每個僅供讀取複本，在上一步中使用來源執行個體的檔案和位置資訊來執行下列查詢：

      對於第 3 版

```
SELECT SOURCE_POS_WAIT('file', position);
```

      對於第 2 版

```
SELECT MASTER_POS_WAIT('file', position);
```

例如，如果檔案名稱是 mysql-bin-changelog.000031，而且位置是 107，則執行下列陳述式：

對於第 3 版

```
SELECT SOURCE_POS_WAIT('mysql-bin-changelog.000031', 107);
```

對於第 2 版

```
SELECT MASTER_POS_WAIT('mysql-bin-changelog.000031', 107);
```

5. 重設 GTID 參數，以停用 GTID 式複寫。

   a. 確定與 Aurora MySQL 叢集相關聯的資料庫叢集參數群組有下列參數設定：

      - gtid_mode – OFF
      - enforce_gtid_consistency – OFF

   b. 重新啟動 Aurora MySQL 資料庫叢集。

在 Amazon Aurora MySQL 資料庫叢集中使用本機寫入轉送

本機 (叢集內) 寫入轉送允許您的應用程式直接在 Aurora 複本上發出讀取/寫入交易。然後，這些交易會轉送至要提交的寫入器資料庫執行個體。您可以當應用程式要求先寫後讀一致性時使用本機寫入轉送，能夠讀取交易中最新寫入的能力。

僅供讀取複本會以非同步方式從寫入器接收更新。如果沒有寫入轉送，您必須在寫入器資料庫執行個體上處理任何需要先寫後讀一致性的讀取。否則，您必須開發複雜的自訂應用程式邏輯，才能利用多個僅供讀取複本來實現可擴展性。您的應用程式必須完全拆分所有讀取和寫入流量，維護兩組資料庫連線，才能將流量傳送到正確的端點。當查詢是應用程式內單一邏輯工作階段的一部分，或是交易時，這類部署會讓應用程式設計過於複雜。此外，由於僅供讀取複本的複寫延遲可能不同，因此很難在資料庫中的所有執行個體達到全域讀取一致性。

寫入轉送可以避免拆分交易或以專門的方式傳送至寫入器，進而簡化應用程式開發作業。借助這項新功能，可讓您輕鬆現需要讀取交易最新寫入並且工作負載對寫入延遲不敏感的讀取擴展。

本機寫入轉送與全域寫入轉送不同，全域寫入轉送會將寫入從次要資料庫叢集轉送至 Aurora 全球資料庫中的主要資料庫叢集。您可以在屬於 Aurora 全球資料庫一部分的資料庫叢集中使用本機寫入轉送。如需更多詳細資訊，請參閱 在 Amazon Aurora 全域資料庫中使用寫入轉送。

本機寫入轉送需要 Aurora MySQL 3.04 版或更新版本。

主題

- 啟用本機寫入轉送
- 檢查資料庫叢集是否已啟用寫入轉送
- 應用程式和 SQL 與寫入轉送的相容性
- 寫入轉送的隔離層級
- 寫入轉送的讀取一致性
- 使用寫入轉送執行多部分陳述式
- 具有寫入轉送的交易
- 寫入轉送的組態參數
- 用於寫入轉送的 Amazon CloudWatch 指標和 Aurora MySQL 狀態變數
- 識別轉送的交易和查詢

啟用本機寫入轉送

根據預設，Aurora MySQL 資料庫叢集不會啟用本機寫入轉送功能。您能夠在叢集層級啟用本機寫入轉送，而非執行個體層級。

> **Important**
>
> 您也可以針對使用二進位記錄的跨區域僅供讀取複本啟用本機寫入轉送，但寫入操作不會轉送至來源 AWS 區域。寫入操作會轉送至 binlog 僅供讀取複本叢集的寫入器資料庫執行個體。只有當您擁有寫入次要 binlog 僅供讀取複本的使用案例時，才使用此方法 AWS 區域。否則，您最終可能會面臨「大腦分裂」的情況，其複寫的資料集彼此不一致。
>
> 除非絕對必要，否則我們建議您在全球資料庫上使用全域寫入轉送，而不要在跨區域僅供讀取複本上使用本機寫入轉送。如需更多詳細資訊，請參閱 在 Amazon Aurora 全域資料庫中使用寫入轉送。

主控台

使用 AWS 管理主控台，當您建立或修改資料庫叢集時，選取僅供讀取複本寫入轉送中的開啟本機寫入轉送核取方塊。

AWS CLI

若要使用 AWS CLI 啟用寫入轉送，請使用 `--enable-local-write-forwarding` 選項。當您使用 `create-db-cluster` 命令建立新的資料庫叢集時，此選項會起作用。當您使用 `modify-db-cluster` 命令修改現有的資料庫叢集時，此選項也會起作用。您可以使用 `--no-enable-local-write-forwarding` 選項搭配這些相同的 CLI 命令來停用寫入轉送。

下列範例會建立啟用寫入轉送的 Aurora MySQL 資料庫叢集。

```
aws rds create-db-cluster \
  --db-cluster-identifier write-forwarding-test-cluster \
  --enable-local-write-forwarding \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.04.0 \
  --master-username myuser \
  --master-user-password mypassword \
  --backup-retention 1
```

然後，您可以建立寫入器和讀取器資料庫執行個體，以便您可以使用寫入轉送。如需更多詳細資訊，請參閱 建立 Amazon Aurora 資料庫叢集。

RDS API

若要使用 Amazon RDS API 啟用寫入轉送，請將 EnableLocalWriteForwarding 參數設定為 true。當您使用 CreateDBCluster 操作建立新的資料庫叢集時，此參數會起作用。當您使用 ModifyDBCluster 操作修改現有的資料庫叢集時，此選項也會起作用。您可以將 EnableLocalWriteForwarding 參數設定為 false 來停用寫入轉送。

啟用資料庫工作階段的寫入轉送

aurora_replica_read_consistency 參數是啟用寫入轉送的資料庫參數和資料庫叢集參數。您可以指定 EVENTUAL SESSION 或 GLOBAL 以實現讀取一致性層級。若要進一步了解一致性層級，請參閱寫入轉送的讀取一致性。

下列規則適用於此參數：

- 預設值為 '' (null)。
- 只有將 aurora_replica_read_consistency 設定為 EVENTUAL、SESSION 或 GLOBAL 時，才能使用寫入轉送。此參數僅在啟用寫入轉送之資料庫叢集的讀取器執行個體中啟用。
- 您無法在多陳述式交易中設定此參數 (當為空時) 或取消其設定 (當已經設定時)。這類交易期間，您可以將其從一個有效值變更為另一個有效值，但我們不建議您執行此動作。

檢查資料庫叢集是否已啟用寫入轉送

若要確定您是否可以在資料庫叢集中使用寫入轉送，請確認該叢集的屬性 LocalWriteForwardingStatus 設定為 enabled。

在 AWS 管理主控台 中，叢集的詳細資訊頁面上的組態標籤，您可以看到本機僅供讀取複本寫入轉送的狀態為啟用。

若要查看所有叢集的寫入轉送設定狀態，請執行下列 AWS CLI 命令。

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,LocalWriteForwardingStatus:LocalWriteForwardingStatus}

[
```

```
    {
        "LocalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "write-forwarding-test-cluster-1"
    },
    {
        "LocalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "write-forwarding-test-cluster-2"
    },
    {
        "LocalWriteForwardingStatus": "requested",
        "DBClusterIdentifier": "test-global-cluster-2"
    },
    {
        "LocalWriteForwardingStatus": "null",
        "DBClusterIdentifier": "aurora-mysql-v2-cluster"
    }
]
```

資料庫叢集可以具有以下值 `LocalWriteForwardingStatus`：

- `disabled` – 寫入轉送已停用。
- `disabling` – 寫入轉送正處於停用的過程。
- `enabled` – 寫入轉送已啟用。
- `enabling` – 寫入轉送正處於啟用的過程。
- `null` – 寫入轉送不適用於此資料庫叢集。
- `requested` – 已請求寫入轉送，但尚未作用中。

應用程式和 SQL 與寫入轉送的相容性

您可以使用以下類型的 SQL 陳述式搭配寫入轉送：

- 資料操作語言 (DML) 陳述式，例如 `INSERT`、`DELETE` 和 `UPDATE`。這些陳述式的屬性有一些限制，您可以將這些屬性與寫入轉送搭配使用，如下所述。
- `SELECT ... LOCK IN SHARE MODE` 和 `SELECT FOR UPDATE` 陳述式。
- `PREPARE` 和 `EXECUTE` 陳述式。

當您在具有寫入轉送的資料庫叢集中使用某些陳述式時，系統不允許使用這些陳述式或這些陳述式可能會產生過時的結果。此外，不支援使用者定義的函數和使用者定義的程序。因此，資料庫叢集將

EnableLocalWriteForwarding 設定預設為停用。在啟用此功能之前，請檢查以確定您的應用程式的程式碼不受上述任何限制的影響。

下列限制適用於您與寫入轉送搭配使用的 SQL 陳述式。在某些情況下，您可以在啟用寫入轉送的資料庫叢集上使用陳述式。如果在工作階段中的寫入轉送啟用方式不是透過 aurora_replica_read_consistency 組態參數，則此方法有效。在不允許此方法時嘗試使用陳述式，因為寫入轉送導致，您將看到類似於下列的錯誤訊息：

```
ERROR 1235 (42000): This version of MySQL doesn't yet support 'operation with write
 forwarding'.
```

資料定義語言 (DDL)

連線到寫入器資料庫執行個體以執行 DDL 陳述式。您無法從讀取器資料庫執行個體執行。

使用臨時資料表中的資料更新永久資料表

您可以在啟用寫入轉送的資料庫叢集上使用臨時資料表。但是，如果陳述式參照臨時資料表，則無法使用 DML 陳述式來修改永久資料表。例如，您不能使用從臨時資料表取得資料的 INSERT ... SELECT 陳述式。

XA 交易

在工作階段中啟用寫入轉送時，您無法在資料庫叢集上使用下列陳述式。您可以在未啟用寫入轉送的資料庫叢集上，或在 aurora_replica_read_consistency 設定為空的工作階段中使用這些陳述式。在工作階段中啟用寫入轉送之前，請檢查您的程式碼是否使用這些陳述式。

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER [CONVERT XID]
```

永久資料表的 LOAD 陳述式

您無法在啟用寫入轉送的資料庫叢集上使用下列陳述式。

```
LOAD DATA INFILE 'data.txt' INTO TABLE t1;
LOAD XML LOCAL INFILE 'test.xml' INTO TABLE t1;
```

外掛程式陳述式

您無法在啟用寫入轉送的資料庫叢集上使用下列陳述式。

```
INSTALL PLUGIN example SONAME 'ha_example.so';
UNINSTALL PLUGIN example;
```

儲存點陳述式

在工作階段中啟用寫入轉送時，您無法在資料庫叢集上使用下列陳述式。您可以在未啟用寫入轉送的資料庫叢集上，或在 aurora_replica_read_consistency 設定為空的工作階段中使用這些陳述式。在工作階段中啟用寫入轉送之前，請檢查您的程式碼是否使用這些陳述式。

```
SAVEPOINT t1_save;
ROLLBACK TO SAVEPOINT t1_save;
RELEASE SAVEPOINT t1_save;
```

寫入轉送的隔離層級

在使用寫入轉送的工作階段中，您只能使用 REPEATABLE READ 隔離層級。雖然您也可以使用 Aurora 複本的 READ COMMITTED 隔離層級，但該隔離層級不適用於寫入轉送。如需 REPEATABLE READ 和 READ COMMITTED 隔離層級的相關資訊，請參閱 Aurora MySQL 隔離層級。

寫入轉送的讀取一致性

您可以控制資料庫叢集上的讀取一致性程度。讀取一致性層級會決定資料庫叢集在每次讀取操作之前的等待時間，以確保從寫入器複寫部分或全部變更。您可以調整讀取一致性層級，以確保在任何後續查詢之前，您都可以在資料庫叢集中看見工作階段中的所有轉送寫入操作。您也可以使用此設定，確保資料庫叢集上的查詢永遠會看到寫入器的最新更新。此設定也適用於由其他工作階段或其他叢集提交的查詢。若要為應用程式指定這種行為類型，請選擇 aurora_replica_read_consistency 資料庫叢集參數的值或資料庫叢及參數。

> **Important**
>
> 當您要轉送寫入時，永遠設定 aurora_replica_read_consistency 資料庫參數或資料庫叢集參數。如果您不這樣做，則 Aurora 不會轉送寫入。此參數預設為空值，因此當您使用此參數時，請選擇特定值。aurora_replica_read_consistency 參數只會影響資料庫叢集或已啟用寫入轉送的執行個體。

當您提高一致性層級時，您的應用程式會花費更多時間，等待在資料庫執行個體之間傳播變更。您可以選擇在快速回應時間之間的平衡，並確保在查詢執行之前，在其他資料庫執行個體所做的變更完全可用。

您可以為 aurora_replica_read_consistency 參數指定下列參數值：

- EVENTUAL – 在寫入器資料庫執行個體上執行寫入作業之前，不會顯示相同工作階段中的寫入作業結果。查詢不會等待更新的結果變成可用。因此，它可能會擷取較舊的資料或更新的資料，視陳述式的時間和複寫延遲量而定。這與沒有使用寫入轉送的 Aurora MySQL 資料庫叢集的一致性相同。
- SESSION – 使用寫入轉送的所有查詢都會查看在該工作階段中所做的所有變更。無論交易是否已遞交，這些變更都是可見的。如有必要，查詢會等待轉送寫入操作複寫的結果。
- GLOBAL – 工作階段會查看資料庫叢集中所有工作階段和執行個體的所有已提交變更。每個查詢可能會等待一段時間，長短取決於工作階段的延遲量。自查詢開始的時間起，當資料庫叢集與寫入器中的所有遞交資料都是最新時，查詢就會繼續。

如需寫入轉送中所含組態參數的詳細資訊，請參閱 寫入轉送的組態參數。

> **Note**
>
> 您也可以使用 aurora_replica_read_consistency 作為工作階段變數，例如：
>
> ```
> mysql> set aurora_replica_read_consistency = 'session';
> ```

使用寫入轉送的範例

下列範例顯示在執行 INSERT 陳述式後接著執行 SELECT 陳述式時 aurora_replica_read_consistency 參數的效果。視 aurora_replica_read_consistency 值和陳述式時間而定，結果可能會有所不同。

為了實現更高的一致性，在發出 SELECT 陳述式之前，您可能需要稍作等待。或者，Aurora 可以自動等待結果完成複寫，然後再繼續進行 SELECT。

如需設定資料庫參數的資訊，請參閱Amazon Aurora 的參數群組。

Example並將 aurora_replica_read_consistency 設為 EVENTUAL

執行 INSERT 陳述式，緊接著 SELECT 陳述式，傳回 COUNT(*) 的值與插入新一列之前的資料列數。稍後再次執行 SELECT 會傳回更新的資料列計數。這些 SELECT 陳述式不會等待。

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

mysql> insert into t1 values (6); select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)
```

Example並將 aurora_replica_read_consistency 設為 SESSION

INSERT 之後的 SELECT 陳述式會立即等待，直到看見 INSERT 陳述式的變更。後續的 SELECT 陳述式不會等待。

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.01 sec)

mysql> insert into t1 values (6); select count(*) from t1; select count(*) from t1;
Query OK, 1 row affected (0.08 sec)
+----------+
| count(*) |
+----------+
|        7 |
```

```
+----------+
1 row in set (0.37 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.00 sec)
```

將讀取一致性設定仍設為 SESSION 後，執行 INSERT 陳述式後稍等一下，讓更新的資料列計數可在下一個 SELECT 陳述式執行時使用。

```
mysql> insert into t1 values (6); select sleep(2); select count(*) from t1;
Query OK, 1 row affected (0.07 sec)
+----------+
| sleep(2) |
+----------+
|        0 |
+----------+
1 row in set (2.01 sec)
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

Example並將 aurora_replica_read_consistency 設為 GLOBAL

每個 SELECT 陳述式都會等待，以確保自陳述式開始時間起的所有資料變更都可見，然後再執行查詢。每個 SELECT 陳述式等待的時間會有所不同，根據複寫延遲的數量而定。

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.75 sec)

mysql> select count(*) from t1;
+----------+
```

```
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.37 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.66 sec)
```

使用寫入轉送執行多部分陳述式

DML 陳述式可能包含多個部分，例如 INSERT ... SELECT 陳述式或 DELETE ... WHERE 陳述式。在這種情況下，系統會將整個陳述式轉送到寫入器資料庫執行個體並在該處執行陳述式。

具有寫入轉送的交易

如果交易存取模式設定為唯讀，則不會使用寫入轉送。您可以使用 SET TRANSACTION 陳述式或 START TRANSACTION 陳述式，來指定交易的存取模式。您也可以透過變更 transaction_read_only 工作階段變數的值，來指定交易存取模式。您只能在連線至已啟用寫入轉送的資料庫叢集時，變更此工作階段值。

如果長時間執行的交易經過很長一段時間內都未發出任何陳述式，則可能會超過閒置逾時期間。此期間的預設值為一分鐘。您可以將 aurora_fwd_writer_idle_timeout 參數設定為最多增加一天。超過閒置逾時的交易會被寫入器執行個體取消。您提交的下一個後續陳述式會收到逾時錯誤。然後 Aurora 會復原交易。

當寫入轉送變成無法使用時，可能會發生這種類型的錯誤。例如，如果您重新啟動資料庫叢集或停用寫入轉送，Aurora 就會取消任何使用寫入轉送的交易。

當叢集中使用本機寫入轉送的寫入器執行個體重新啟動時，使用本機寫入轉送的讀取器執行個體上任何作用中的轉送交易和查詢都會自動關閉。當寫入器執行個體再次可用之後，您可以重試這些交易。

寫入轉送的組態參數

Aurora 資料庫參數群組包含寫入轉送功能的設定。下表列出了有關這些參數的詳細資訊，並在表格後面附有使用注意事項。

參數	範圍	類型	預設值	有效值
aurora_fwd_writer_idle_timeout	叢集	不帶正負號整數	60	1–86,400
aurora_fwd_writer_max_connections_pct	叢集	不帶正負號長整數	10	0–90
aurora_replica_read_consistency	叢集或執行個體	列舉	" (null)	EVENTUAL, SESSION, GLOBAL

若要控制傳入的寫入請求，請使用下列設定：

- aurora_fwd_writer_idle_timeout – 寫入器資料庫執行個體在關閉讀取器執行個體之前，等待從讀取器執行個體轉送的連線上活動的秒數。如果工作階段在此期間之後仍處於閒置狀態，則 Aurora 會取消工作階段。
- aurora_fwd_writer_max_connections_pct – 可在寫入器資料庫執行個體上，用來處理從讀取器執行個體轉送之查詢的資料庫連線上限。此上限的表示方式是寫入器的 max_connections 設定百分比。例如，如果 max_connections 是 800，且 aurora_fwd_master_max_connections_pct 或 aurora_fwd_writer_max_connections_pct 是 10，則寫入器允許最多 80 個同時轉送的工作階段。這些連線來自 max_connections 設定所管理的相同連線集區。

  此設定僅適用於已啟用寫入轉送功能的寫入器。如果您減少此值，現有的連線不會受到影響。Aurora 在嘗試從資料庫叢集建立新連線時，會考慮設定的新值。預設值為 10，代表該 max_connections 值的 10%。

> **Note**
>
> 因為 aurora_fwd_writer_idle_timeout 和 aurora_fwd_writer_max_connections_pct是資料庫叢集參數，所以每個叢集中的所有資料庫執行個體都有這些參數的相同值。

如需 aurora_replica_read_consistency 的相關資訊，請參閱 寫入轉送的讀取一致性。

如需資料庫參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。

用於寫入轉送的 Amazon CloudWatch 指標和 Aurora MySQL 狀態變數

當您使用 Aurora MySQL 的寫入轉送時，下列 Amazon CloudWatch 指標和 Aurora MySQL 狀態變數適用於資料庫叢集。如需 Aurora MySQL 寫入器和讀取器資料庫執行個體指標的詳細資訊，請參閱下列主題。

主題

- Aurora MySQL 寫入器資料庫執行個體的寫入轉送指標
- Aurora MySQL 讀取器資料庫執行個體的寫入轉送指標

Aurora MySQL 寫入器資料庫執行個體的寫入轉送指標

當您在一或多個資料庫叢集上使用寫入轉送時，下列 Amazon CloudWatch 指標適用於資料庫叢集。這些指標都是在寫入器資料庫執行個體上測量的。

CloudWatch 指標	單位	Description
`ForwardingWriterDMLLatency`	毫秒	處理寫入器資料庫執行個體上每個轉送 DML 陳述式的平均時間。 它不包括資料庫叢集轉送寫入請求的時間，或將變更複寫至寫入器的時間。
`ForwardingWriterDMLThroughput`	每秒計數	此寫入器資料庫執行個體每秒處理的轉送 DML 陳述式數目。
`ForwardingWriterOpenSessions`	計數	寫入器資料庫執行個體上轉送的工作階段數目。

當您在一或多個資料庫叢集上使用寫入轉送時，下列 Aurora MySQL 狀態變數適用於資料庫叢集。這些狀態變數都是在寫入器資料庫執行個體上測量的。

Aurora MySQL 狀態變數	單位	Description
Aurora_fwd_writer_dml_stmt_count	計數	轉送至此寫入器資料庫執行個體的 DML 陳述式總數。
Aurora_fwd_writer_dml_stmt_duration	微秒	轉送至此寫入器資料庫執行個體的 DML 陳述式總持續時間。
Aurora_fwd_writer_open_sessions	計數	寫入器資料庫執行個體上轉送的工作階段數目。
Aurora_fwd_writer_select_stmt_count	計數	轉送至此寫入器資料庫執行個體的 SELECT 陳述式總數。
Aurora_fwd_writer_select_stmt_duration	微秒	轉送至此寫入器資料庫執行個體的 SELECT 陳述式總持續時間。

Aurora MySQL 讀取器資料庫執行個體的寫入轉送指標

下列 CloudWatch 指標是在啟用寫入轉送的資料庫叢集中的每個讀取器資料庫執行個體上進行測量。

CloudWatch 指標	單位	Description
ForwardingReplicaDMLLatency	毫秒	複本上轉送 DML 的平均回應時間。
ForwardingReplicaDMLThroughput	每秒計數	每秒處理的轉送 DML 陳述式數目。
ForwardingReplicaOpenSessions	計數	在讀取器資料庫執行個體上使用寫入轉送的工作階段數目。
ForwardingReplicaReadWaitLatency	毫秒	讀取器資料庫執行個體上的 SELECT 陳述式等待以追上寫入器的平均等待時間。

CloudWatch 指標	單位	Description
		讀取器資料庫執行個體在處理查詢之前等待的程度取決於 aurora_replica_read_consistency 設定。
ForwardingReplicaReadWaitThroughput	每秒計數	轉寄寫入的所有工作階段中每秒處理的 SELECT 陳述式總數。
ForwardingReplicaSelectLatency	毫秒	轉送的 SELECT 延遲，平均計算監控期間內所有轉送的 SELECT 陳述式。
ForwardingReplicaSelectThroughput	每秒計數	在監控期間內平均每秒轉送的 SELECT 輸送量。

下列 Aurora MySQL 狀態變數是在啟用寫入轉送的資料庫叢集中的每個讀取器資料庫執行個體上進行測量。

Aurora MySQL 狀態變數	單位	Description
Aurora_fwd_replica_dml_stmt_count	計數	從此讀取器資料庫執行個體轉送的 DML 陳述式總數。
Aurora_fwd_replica_dml_stmt_duration	微秒	從此讀取器資料庫執行個體轉送的所有 DML 陳述式總持續時間。
Aurora_fwd_replica_errors_session_limit	計數	主要叢集因下列其中一個錯誤狀況拒絕的工作階段數目： • 寫入器已滿 • 處理過多的轉送陳述式。
Aurora_fwd_replica_open_sessions	計數	在讀取器資料庫執行個體上使用寫入轉送的工作階段數目。

Aurora MySQL 狀態變數	單位	Description
Aurora_fwd_replica_read_wait_count	計數	此讀取器資料庫執行個體上先寫後讀的等待總數。
Aurora_fwd_replica_read_wait_duration	微秒	由於此讀取器資料庫執行個體的讀取一致性設定，而造成的等待總持續時間。
Aurora_fwd_replica_select_stmt_count	計數	從此讀取器資料庫執行個體轉送的 SELECT 陳述式總數。
Aurora_fwd_replica_select_stmt_duration	微秒	從此讀取器資料庫執行個體轉送的 SELECT 陳述式總持續時間。

識別轉送的交易和查詢

您可以使用 information_schema.aurora_forwarding_processlist 資料表來識別轉送的交易和查詢。如需此資料表的詳細資訊，請參閱 information_schema.aurora_forwarding_processlist。

下列範例顯示寫入器資料庫執行個體上所有轉送的連線。

```
mysql> select * from information_schema.AURORA_FORWARDING_PROCESSLIST where
 IS_FORWARDED=1 order by REPLICA_SESSION_ID;

+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+--------------+--------------------
+---------------------------------+----------------------+----------------+
| ID  | USER     | HOST               | DB       | COMMAND | TIME | STATE        |
 INFO                                      | IS_FORWARDED | REPLICA_SESSION_ID |
 REPLICA_INSTANCE_IDENTIFIER     | REPLICA_CLUSTER_NAME | REPLICA_REGION |
+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+--------------+--------------------
+---------------------------------+----------------------+----------------+
| 648 | myuser   | IP_address:port1   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest58 SET k=k+1 WHERE id=4802579 |            1 |                637 | my-
db-cluster-instance-2        | my-db-cluster        | us-west-2      |
```

```
| 650 | myuser   | IP_address:port2   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest54 SET k=k+1 WHERE id=2503953 |            1 |               639 | my-
db-cluster-instance-2        | my-db-cluster        | us-west-2      |
+-----+----------+--------------------+----------+---------+------+--------------
+---------------------------------------------+--------------+--------------------
+----------------------------------+---------------------+----------------+
```

在轉送讀取器資料庫執行個體上，您可以透過執行 SHOW PROCESSLIST 以查看與這些寫入器資料庫連線相關聯的執行緒。寫入器上 REPLICA_SESSION_ID 的值 637 和 639，與讀取器上的 Id 值相同。

```
mysql> select @@aurora_server_id;

+----------------------------------+
| @@aurora_server_id               |
+----------------------------------+
| my-db-cluster-instance-2         |
+----------------------------------+
1 row in set (0.00 sec)

mysql> show processlist;

+-----+----------+--------------------+----------+---------+------+--------------
+---------------------------------------------+
| Id  | User     | Host               | db       | Command | Time | State        | Info
                                        |
+-----+----------+--------------------+----------+---------+------+--------------
+---------------------------------------------+
| 637 | myuser   | IP_address:port1   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest12 SET k=k+1 WHERE id=4802579  |
| 639 | myuser   | IP_address:port2   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest61 SET k=k+1 WHERE id=2503953  |
+-----+----------+--------------------+----------+---------+------+--------------
+---------------------------------------------+
12 rows in set (0.00 sec)
```

將 Amazon Aurora MySQL 與其他 AWS 服務整合

Amazon Aurora MySQL 與其他 AWS 服務整合，因此您可以擴展 Aurora MySQL 資料庫叢集，以在 AWS 雲端中使用其他功能。您的 Aurora MySQL 資料庫叢集可以使用 AWS 服務來執行下列動作：

- 使用原生 AWS Lambda 函數 或 同步或非同步叫用 `lambda_sync` 函數`lambda_async`。如需詳細資訊，請參閱從 Amazon Aurora MySQL 資料庫叢集叫用 Lambda 函式。
- 使用 `LOAD DATA FROM S3` 或 `LOAD XML FROM S3` 命令，從存放在 Amazon Simple Storage Service (Amazon S3) 儲存貯體中的文字或 XML 檔案將資料載入資料庫叢集。如需更多詳細資訊，請參閱 從 Amazon S3 儲存貯體中的文字檔案將資料載入 Amazon Aurora MySQL 資料庫叢集。
- 使用 `SELECT INTO OUTFILE S3` 命令，將資料從資料庫叢集儲存至在 Amazon S3 儲存貯體中存放的文字檔案。如需更多詳細資訊，請參閱 將來自 Amazon Aurora MySQL 資料庫叢集的資料儲存至 Amazon S3 儲存貯體中的文字檔案。
- 使用 Application Auto Scaling 自動新增或移除 Aurora 複本。如需詳細資訊，請參閱使用 Amazon Aurora 自動擴展搭配 Aurora 複本。
- 使用 Amazon Comprehend 執行情緒分析，或使用 SageMaker AI 執行多種機器學習演算法。如需詳細資訊，請參閱使用 Amazon Aurora 機器學習。

Aurora 使用 AWS Identity and Access Management (IAM) 保護存取其他 AWS 服務的能力。您可以透過建立具有必要許可的 IAM 角色，然後將角色與資料庫叢集建立關聯，授予存取其他 AWS 服務的許可。如需如何允許 Aurora MySQL 資料庫叢集代表您存取其他 AWS 服務的詳細資訊和指示，請參閱 授權 Amazon Aurora MySQL 代表您存取其他 AWS 服務。

授權 Amazon Aurora MySQL 代表您存取其他 AWS 服務

若要讓 Aurora MySQL 資料庫叢集代表您存取其他 服務，請建立並設定 AWS Identity and Access Management (IAM) 角色。此角色授權資料庫叢集中的資料庫使用者存取其他 AWS 服務。如需詳細資訊，請參閱設定 IAM 角色以存取 AWS 服務。

您還必須設定 Aurora 資料庫叢集，以允許對目標 AWS 服務的傳出連線。如需詳細資訊，請參閱啟用從 Amazon Aurora 到其他服務的網路通訊 AWS。

如果這樣做，則您的資料庫使用者可以利用其他 AWS 服務來執行下列動作：

- 使用原生 AWS Lambda 函數 或 同步或非同步叫用 `lambda_sync` 函數`lambda_async`。或是，使用 AWS Lambda 程序非同步叫用 `mysql.lambda_async` 函式。如需更多詳細資訊，請參閱 使用 Aurora MySQL 原生函式叫用 Lambda 函式。

- 使用 LOAD DATA FROM S3 或 LOAD XML FROM S3 陳述式，從存放在 Amazon S3 儲存貯體的文字檔案或 XML 檔案中，將資料載入資料庫叢集。如需更多詳細資訊，請參閱 從 Amazon S3 儲存貯體中的文字檔案將資料載入 Amazon Aurora MySQL 資料庫叢集。
- 使用 SELECT INTO OUTFILE S3 陳述式，將資料從資料庫叢集儲存至 Amazon S3 儲存貯體中存放的文字檔案。如需更多詳細資訊，請參閱 將來自 Amazon Aurora MySQL 資料庫叢集的資料儲存至 Amazon S3 儲存貯體中的文字檔案。
- 將日誌資料匯出至 Amazon CloudWatch Logs MySQL。如需更多詳細資訊，請參閱 將 Amazon Aurora MySQL 日誌發佈至 Amazon CloudWatch Logs。
- 使用 Application Auto Scaling 自動新增或移除 Aurora 複本。如需詳細資訊，請參閱使用 Amazon Aurora 自動擴展搭配 Aurora 複本。

設定 IAM 角色以存取 AWS 服務

若要允許 Aurora 資料庫叢集存取其他服務 AWS ，請執行下列動作：

1. 建立授予 AWS 服務許可的 IAM 政策。如需詳細資訊，請參閱下列主題。
   - 建立 IAM 政策來存取 Amazon S3 資源
   - 建立 IAM 政策以存取 AWS Lambda 資源
   - 建立 IAM 政策來存取 CloudWatch Logs 資源
   - 建立 IAM 政策以存取 AWS KMS資源
2. 建立 IAM 角色並附加您建立的政策。如需更多詳細資訊，請參閱 建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務。
3. 將此 IAM 角色與 Aurora 資料庫叢集產生關聯。如需詳細資訊，請參閱將 IAM 角色與 Amazon Aurora MySQL 資料庫叢集建立關聯。

建立 IAM 政策來存取 Amazon S3 資源

Aurora 可以存取 Amazon S3 資源，以便從 Aurora 資料庫叢集載入資料或儲存資料。不過，您必須先建立 IAM 政策來提供儲存貯體和物件許可，以允許 Aurora 存取 Amazon S3。

下表列出可代表您存取 Amazon S3 儲存貯體的 Aurora 功能，以及每一項功能所需的最低儲存貯體和物件許可。

功能	儲存貯體許可	物件許可
LOAD DATA FROM S3	ListBucket	GetObject

功能	儲存貯體許可	物件許可
		GetObjectVersion
LOAD XML FROM S3	ListBucket	GetObject GetObjectVersion
SELECT INTO OUTFILE S3	ListBucket	AbortMultipartUpload DeleteObject GetObject ListMultipartUploadParts PutObject

下列政策會新增 Aurora 代您存取 Amazon S3 儲存貯體所需的許可。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAuroraToExampleBucket",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:AbortMultipartUpload",
                "s3:ListBucket",
                "s3:DeleteObject",
                "s3:GetObjectVersion",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/*",
                "arn:aws:s3:::amzn-s3-demo-bucket"
```

```
            ]
        }
    ]
}
```

> **Note**
>
> `Resource` 值務必包含這兩個項目。Aurora 需要儲存貯體本身和儲存貯體內所有對象的許可權限。
> 根據您的使用案例，可能不需要在範例政策中新增所有許可。此外，還可能需要其他許可。例如，若您的 Amazon S3 儲存貯體經過加密，您便需要新增 `kms:Decrypt` 許可。

您可以使用下列步驟來建立 IAM 政策，以提供讓 Aurora 代您存取 Amazon S3 儲存貯體所需的最低許可。若要允許 Aurora 存取您的所有 Amazon S3 儲存貯體，您可以略過這些步驟，並使用 `AmazonS3ReadOnlyAccess` 或 `AmazonS3FullAccess` 預先定義的 IAM 政策，而不需自行建立。

建立 IAM 政策以授權存取 Amazon S3 資源

1. 開啟 IAM 管理主控台。
2. 在導覽窗格中，選擇政策。
3. 選擇 Create policy (建立政策)。
4. 在 Visual editor (視覺化編輯器) 標籤中，選擇 Choose a service (選擇服務)，然後選擇 S3。
5. 請在 Actions (動作) 中選擇 Expand all (全部展開)，然後選擇 IAM 政策所需的儲存貯體許可和物件許可。

   物件許可是在 Amazon S3 中執行物件操作的許可，必須授予儲存貯體中的物件，而非儲存貯體本身。如需 Amazon S3 中執行物件操作所需許可的詳細資訊，請參閱物件操作的許可。
6. 選擇 Resources (資源)，然後對 bucket (儲存貯體) 選擇 Add ARN (新增 ARN)。
7. 在 Add ARN(s) (新增 ARN) 對話方塊中，提供資源的詳細資訊，然後選擇 Add (新增)。

   指定允許存取的 Amazon S3 儲存貯體。例如，若要允許 Aurora 存取名為 *`amzn-s3-demo-bucket`* 的 Amazon S3 儲存貯體，請將 Amazon Resource Name (ARN) 值設為 `arn:aws:s3:::`*`amzn-s3-demo-bucket`*。
8. 如果 object (物件) 資源列出，請對 object (物件) 選擇 Add ARN (新增 ARN)。
9. 在 Add ARN(s) (新增 ARN) 對話方塊中，提供資源的詳細資訊。

對於 Amazon S3 儲存貯體，請指定允許存取的 Amazon S3 儲存貯體。對於物件，您可以選擇 Any (任何)，以授權存取儲存貯體中的任何物件。

> **Note**
>
> 您可以將 Amazon Resource Name (ARN) 設為更具體的 ARN 值，以允許 Aurora 只能存取 Amazon S3 儲存貯體中的特定檔案或資料夾。如需如何為 Amazon S3 定義存取原則的詳細資訊，請參閱管理 Amazon S3 資源的存取許可。

10. (選用) 為 bucket (儲存貯體) 選擇 Add ARN (新增 ARN)，以將另一個 Amazon S3 儲存貯體新增至政策，並對此儲存貯體重複先前的步驟。

> **Note**
>
> 對於您想讓 Aurora 存取的每個 Amazon S3 儲存貯體，您可以重複此程序，在政策中新增對應的儲存貯體許可陳述式。(選擇性) 您也可以授權存取 Amazon S3 中的所有儲存貯體和物件。

11. 選擇 Review policy (檢閱政策)。
12. 在 Name (名稱) 輸入您的 IAM 政策名稱，例如 `AllowAuroraToExampleBucket`。當您建立要與 Aurora 資料庫叢集相關聯的 IAM 角色時，您可以使用此名稱。您也可以新增選用的 Description (描述) 值。
13. 選擇 Create policy (建立政策)。
14. 完成「建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務」中的步驟。

建立 IAM 政策以存取 AWS Lambda 資源

您可以建立 IAM 政策，提供 Aurora 代表您叫用 AWS Lambda 函數所需的最低許可。

下列政策新增讓 Aurora 代表您叫用 AWS Lambda 函式所需的許可。

JSON

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

```
    {
      "Sid": "AllowAuroraToExampleFunction",
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-
east-1:123456789012:function:example_function"
    }
  ]
}
```

您可以使用下列步驟建立 IAM 政策，提供 Aurora 代表您叫用 AWS Lambda 函數所需的最低許可。若要允許 Aurora 叫用所有 AWS Lambda 函數，您可以略過這些步驟，並使用預先定義的`AWSLambdaRole`政策，而不是建立自己的政策。

建立 IAM 政策以將調用授予 AWS Lambda 函數

1. 開啟 IAM 主控台。
2. 在導覽窗格中，選擇政策。
3. 選擇 Create policy (建立政策)。
4. 在 Visual editor (視覺化編輯器) 標籤中，選擇 Choose a service (選擇服務)，然後選擇 Lambda。
5. 選擇 Actions (動作) 下的 Expand all (全部展開)，然後選擇 IAM 政策所需的 AWS Lambda 許可。

   確認已選取 `InvokeFunction`。這是讓 Amazon Aurora 叫用 AWS Lambda 函數所需的最低許可。
6. 選擇 Resources (資源)，然後對 function (函數) 選擇 Add ARN (新增 ARN)。
7. 在 Add ARN(s) (新增 ARN) 對話方塊中，提供資源的詳細資訊。

   指定允許存取的 Lambda 函數。例如，若要允許 Aurora 存取名為 `example_function` 的 Lambda 函式，請將 ARN 值設為 `arn:aws:lambda:::function:example_function`。

   如需如何定義 存取政策的詳細資訊 AWS Lambda，請參閱 的身分驗證和存取控制 AWS Lambda。
8. 或者，選擇新增其他許可以將另一個 AWS Lambda 函數新增至政策，然後重複該函數的先前步驟。

> **Note**
>
> 您可以重複此操作，將對應的函數許可陳述式新增至您希望 Aurora 存取的每個 AWS Lambda 函數的政策。

9. 選擇 Review policy (檢閱政策)。
10. 將 Name (名稱) 設為您的 IAM 政策名稱，例如 `AllowAuroraToExampleFunction`。當您建立要與 Aurora 資料庫叢集相關聯的 IAM 角色時，您可以使用此名稱。您也可以新增選用的 Description (描述) 值。
11. 選擇 Create policy (建立政策)。
12. 完成「建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務」中的步驟。

建立 IAM 政策來存取 CloudWatch Logs 資源

Aurora 可以存取 CloudWatch Logs，以便從 Aurora 資料庫叢集匯出稽核日誌資料。不過，您必須先建立 IAM 政策來提供日誌群組和日誌串流許可，以允許 Aurora 存取 CloudWatch Logs。

下列政策新增讓 Aurora 代表您存取 Amazon CloudWatch Logs 所需的許可，以及建立日誌群組和匯出資料所需的最低許可。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:GetLogEvents",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/rds/*:log-stream:*"
        },
        {
            "Sid":
 "EnableCreationAndManagementOfRDSCloudwatchLogGroupsAndStreams",
            "Effect": "Allow",
            "Action": [
```

```
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutRetentionPolicy",
                "logs:CreateLogGroup"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/rds/*"
        }
    ]
}
```

您可以修改政策中的 ARNs，以限制對特定AWS區域和帳戶的存取。

您可以使用下列步驟來建立 IAM 政策，以提供讓 Aurora 代表您存取 CloudWatch Logs 所需的最低許可。若要允許 Aurora 完整存取 CloudWatch Logs，您可以略過這些步驟，並使用 `CloudWatchLogsFullAccess` 預先定義的 IAM 政策，而不需自行建立。如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的針對 CloudWatch Logs 使用以身分為基礎的政策 (IAM 政策)。

建立 IAM 政策以授權存取 CloudWatch Logs 資源

1. 開啟 IAM 主控台。
2. 在導覽窗格中，選擇政策。
3. 選擇 Create policy (建立政策)。
4. 在 Visual editor (視覺化編輯器) 標籤中，選擇 Choose a service (選擇服務)，然後選擇 CloudWatch Logs。
5. 對於 Actions (動作)，選擇 Expand all (全部展開) (右側)，然後選擇 IAM 政策所需的 Amazon CloudWatch Logs 許可。

   確保已選取下列許可：

   - `CreateLogGroup`
   - `CreateLogStream`
   - `DescribeLogStreams`
   - `GetLogEvents`
   - `PutLogEvents`
   - `PutRetentionPolicy`
6. 選擇 Resources (資源)，然後對 log-group 選擇 Add ARN (新增 ARN)。

7. 在 Add ARN(s) (新增 ARN) 對話方塊中，輸入下列值：
   - 區域 – AWS區域或 *
   - Account (帳戶) – 帳號或 *
   - Log Group Name (日誌群組名稱) – `/aws/rds/*`
8. 在 Add ARN(s) (新增 ARN) 對話方塊中，選擇 Add (新增)。
9. 對於 log-stream，選擇 Add ARN (新增 ARN)。
10. 在 Add ARN(s) (新增 ARN) 對話方塊中，輸入下列值：
    - 區域 – AWS區域或 *
    - Account (帳戶) – 帳號或 *
    - Log Group Name (日誌群組名稱) – `/aws/rds/*`
    - Log Stream Name (日誌串流名稱)* –
11. 在 Add ARN(s) (新增 ARN) 對話方塊中，選擇 Add (新增)。
12. 選擇 Review policy (檢閱政策)。
13. 將 Name (名稱) 設為您的 IAM 政策名稱，例如 `AmazonRDSCloudWatchLogs`。當您建立要與 Aurora 資料庫叢集相關聯的 IAM 角色時，您可以使用此名稱。您也可以新增選用的 Description (描述) 值。
14. 選擇 Create policy (建立政策)。
15. 完成「建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務」中的步驟。

建立 IAM 政策以存取 AWS KMS資源

Aurora 可以存取用於加密其資料庫備份的 AWS KMS keys。不過，您必須先建立可提供許可的 IAM 政策，以允許 Aurora 存取 KMS 金鑰。

下列政策新增讓 Aurora 代表您存取 KMS 金鑰所需的許可。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAuroraToAccessKey",
```

```
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-east-1:123456789012:key/key-ID"
    }
  ]
}
```

您可以使用下列步驟來建立 IAM 政策，以提供讓 Aurora 代表您存取 KMS 金鑰所需的最低許可。

建立 IAM 政策以授權存取 KMS 金鑰

1. 開啟 IAM 主控台。
2. 在導覽窗格中，選擇政策。
3. 選擇 Create policy (建立政策)。
4. 在 Visual editor (視覺化編輯器) 標籤中，選擇 Choose a service (選擇服務)，然後選擇 KMS。
5. 在 Actions (動作) 下選擇 Write (寫入)，然後選擇 Decrypt (解密)。
6. 選擇 Resources (資源)，然後選擇 Add ARN (新增 ARN)。
7. 在 Add ARN(s) (新增 ARN) 對話方塊中，輸入下列值：
   - 區域 – 輸入AWS區域，例如 `us-west-2`。
   - Account (帳戶) – 輸入使用者帳戶號碼。
   - Log Stream Name (日誌串流名稱) – 輸入 KMS 金鑰識別碼。
8. 在 Add ARN(s) (新增 ARN) 對話方塊中，選擇 Add (新增)。
9. 選擇 Review policy (檢閱政策)。
10. 將 Name (名稱) 設為您的 IAM 政策名稱，例如 `AmazonRDSKMSKey`。當您建立要與 Aurora 資料庫叢集相關聯的 IAM 角色時，您可以使用此名稱。您也可以新增選用的 Description (描述) 值。
11. 選擇 Create policy (建立政策)。
12. 完成「建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務」中的步驟。

建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務

建立 IAM 政策以允許 Aurora 存取 AWS 資源後，您必須建立 IAM 角色，並將 IAM 政策連接至新的 IAM 角色。

若要建立 IAM 角色以允許 Amazon RDS 叢集代表您與其他 AWS 服務進行通訊，請採取下列步驟。

建立 IAM 角色以允許 Amazon RDS 存取 AWS 服務

1. 開啟 IAM 主控台。
2. 在導覽窗格中，選擇 Roles (角色)。
3. 選擇 Create Role (建立角色)。
4. 在AWS service (AWS 服務) 下，選擇 RDS。
5. 在 Select your use case (選擇使用案例) 下，選擇 RDS – Add Role to Database (RDS - 新增角色至資料庫)。
6. 選擇下一步。
7. 在 Permissions policies (許可政策) 頁面的 Search (搜尋) 欄位中，輸入您的政策名稱。
8. 出現在清單中時，選取您稍早使用下列其中一節的指示所定義的政策：
   - 建立 IAM 政策來存取 Amazon S3 資源
   - 建立 IAM 政策以存取 AWS Lambda 資源
   - 建立 IAM 政策來存取 CloudWatch Logs 資源
   - 建立 IAM 政策以存取 AWS KMS資源
9. 選擇下一步。
10. 在 Role name (角色名稱) 中輸入 IAM 角色名稱，例如 `RDSLoadFromS3`。您也可以新增選用的 Description (描述) 值。
11. 選擇 Create Role (建立角色)。
12. 完成「將 IAM 角色與 Amazon Aurora MySQL 資料庫叢集建立關聯」中的步驟。

將 IAM 角色與 Amazon Aurora MySQL 資料庫叢集建立關聯

若要允許 Amazon Aurora 資料庫叢集中的資料庫使用者存取其他服務 AWS ，請將您在 中建立的 IAM 角色建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務與該資料庫叢集建立關聯。您也可以直接關聯服務，使 AWS 建立新的 IAM 角色。

> **Note**
>
> 您無法建立 IAM 角色與 Aurora Serverless v1 資料庫叢集的關聯。如需詳細資訊，請參閱使用 Amazon Aurora Serverless v1。
> 您可以建立 IAM 角色與 Aurora Serverless v2 資料庫叢集的關聯。

若要將 IAM 角色與資料庫叢集建立關聯，請執行兩個動作：

1. 使用 RDS 主控台、 add-role-to-db-cluster AWS CLI 命令或 AddRoleToDBCluster RDS API 操作，將角色新增至資料庫叢集的關聯角色清單。

   您可以為每個 Aurora 資料庫叢集新增最多 5 個 IAM 角色。

2. 將相關 AWS 服務的叢集層級參數設定為相關 IAM 角色的 ARN。

   下表針對用於存取其他 AWS 服務的 IAM 角色，描述叢集層級參數名稱。

叢集層級參數	描述
`aws_default_lambda_role`	從資料庫叢集呼叫 Lambda 函數時使用。
`aws_default_logs_role`	從資料庫叢集將日誌資料匯出至 Amazon CloudWatch Logs 時，不再需要此參數。Aurora MySQL 現在會使用服務連結角色來提供必要許可。如需服務連結角色的詳細資訊，請參閱使用 Amazon Aurora 的服務連結角色。
`aws_default_s3_role`	從資料庫叢集呼叫 `LOAD DATA FROM S3`、`LOAD XML FROM S3` 或 `SELECT INTO OUTFILE S3` 陳述式時使用。 在 Aurora MySQL 第 2 版中，如果未針對適當陳述式的 `aurora_load_from_s3_role` 或 `aurora_select_into_s3_role` 指定 IAM 角色，則會使用此參數中指定的 IAM 角色。 在 Aurora MySQL 第 3 版中，一律使用針對此參數指定的 IAM 角色。
`aurora_load_from_s3_role`	從資料庫叢集叫用 `LOAD DATA FROM S3` 或 `LOAD XML FROM S3` 陳述式時使用。如果此參數未指定 IAM 角色，則會使用 `aws_default_s3_role` 中指定的 IAM 角色。

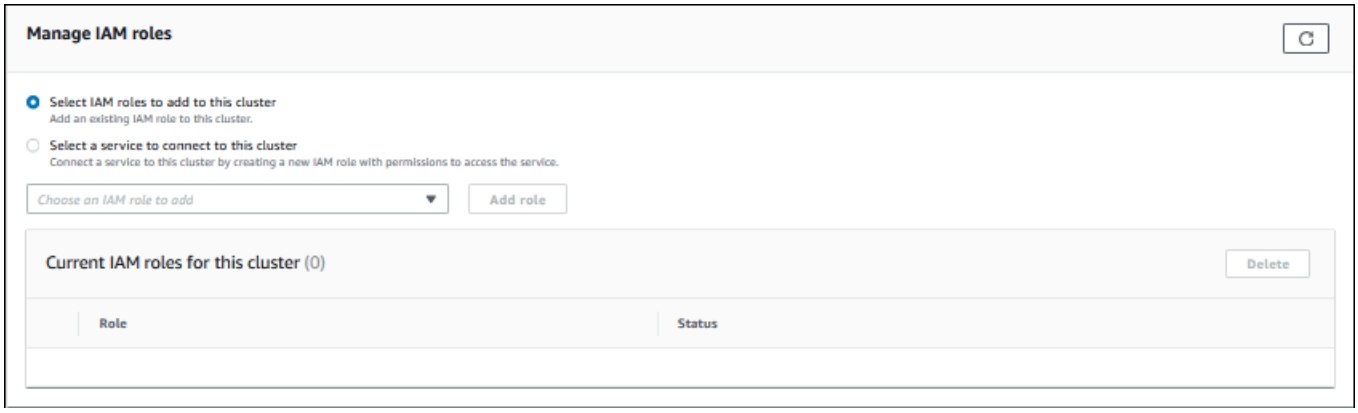
叢集層級參數	描述	
	在 Aurora MySQL 第 3 版中，無法使用此參數。	
aurora_select_into_s3_role	從資料庫叢集叫用 SELECT INTO OUTFILE S3 陳述式時使用。如果此參數未指定 IAM 角色，則會使用 aws_default_s3_role 中指定的 IAM 角色。 在 Aurora MySQL 第 3 版中，無法使用此參數。	

若要建立 IAM 角色的關聯，以允許 Amazon RDS 叢集代表您與其他 AWS 服務通訊，請執行下列步驟。

主控台

使用主控台將 IAM 角色與 Aurora 資料庫叢集建立關聯

1. 請在 https://console.aws.amazon.com/rds/ 開啟 RDS 主控台。
2. 選擇 Databases (資料庫)。
3. 選擇您要關聯 IAM 角色的 Aurora 資料庫叢集名稱，以顯示其詳細資訊。
4. 在 Connectivity & security (連線和安全) 索引標籤的 Manage IAM roles (管理 IAM 角色) 區段中，執行下列其中一項：
   - Select IAM roles to add to this cluster (選取要新增至此叢集的 IAM 角色) (預設值)
   - Select a service to connect to this cluster (選取服務以連線至這個叢集)



5. 若要使用現有的 IAM 角色，請從選單中選擇該角色，然後選擇 Add role (新增角色)。

   如果新增角色成功，則其狀態會顯示為 `Pending`，然後顯示為 `Available`。

6. 若要直接連線服務：

   a. 選擇 Select a service to connect to this cluster (選取服務以連線至這個叢集)。

   b. 從功能表中選擇服務，然後選擇 Connect service (連線服務)。

   c. 對於 Connect cluster to ***`Service Name`*** (將叢集連線至 服務名稱)，請輸入要用來連線至服務的 Amazon Resource Name (ARN)，然後選擇 Connect service (連線服務)。

   AWS 會建立新的 IAM 角色以連線至 服務。其狀態會顯示為 `Pending`，然後顯示為 `Available`。

7. (選用) 若要停止 IAM 角色與資料庫叢集的關聯並移除相關的許可，請選擇角色，然後選擇 Delete (刪除)。

設定相關聯 IAM 角色的叢集層級參數

1. 在 RDS 主控台的導覽窗格中，選擇 Parameter Groups (參數群組)。
2. 如果您已使用自訂的資料庫參數群組，則可以選取該群組來使用，而不需要建立新的資料庫叢集參數群組。如果您使用預設的資料庫叢集參數群組，請建立新的資料庫叢集參數群組，如下列步驟所述：

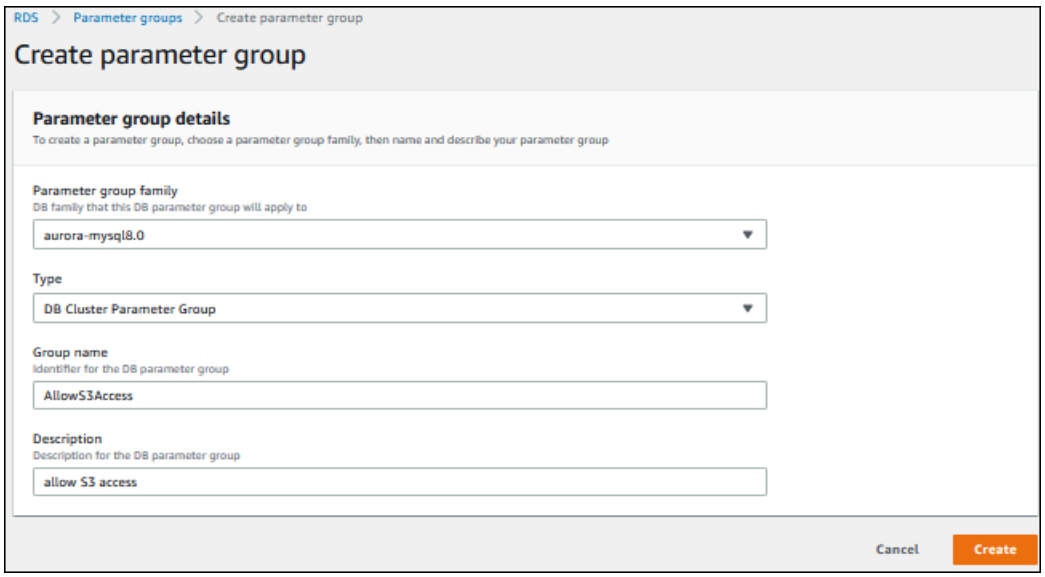
   a. 選擇 Create parameter group (建立參數群組)。

   b. 針對 參數群組系列，若為 Aurora MySQL 8.0 相容的資料庫叢集，請選擇 `aurora-mysql8.0`，若為 Aurora MySQL 5.7 相容的資料庫叢集，請選擇 `aurora-mysql5.7`。

c. 在 Type (類型) 中，選擇 DB Cluster Parameter Group (資料庫叢集參數群組)。

d. 在 Group name (群組名稱) 中，輸入新資料庫叢集參數群組的名稱。

e. 在 Description (描述) 中，輸入新資料庫叢集參數群組的描述。



f. 選擇建立。

3. 在 Parameter groups (參數群組) 頁面上，選取您的資料庫叢集參數群組，然後針對 Parameter group actions (參數群組動作) 選擇 Edit (編輯)。

4. 將適當的叢集層級參數設為相關的 IAM 角色 ARN 值。

   例如，您可以只將 `aws_default_s3_role` 參數設為 `arn:aws:iam::123456789012:role/AllowS3Access`。

5. 選擇儲存變更。

6. 若要變更您資料庫叢集的資料庫叢集參數群組，請完成下列步驟：

   a. 選擇 Databases (資料庫)，然後選擇您的 Aurora 資料庫叢集。

   b. 選擇 Modify (修改)。

   c. 捲動至 Database options (資料庫選項)，然後將 DB cluster parameter group (資料庫叢集參數群組) 設為資料庫叢集參數群組。

   d. 選擇 Continue (繼續)。

   e. 驗證您所做的變更，然後選擇 Apply immediately (立即套用)。

   f. 選擇 Modify cluster (修改叢集)。

   g. 選擇 Databases (資料庫)，然後選擇您資料庫叢集的主要執行個體。

   h. 針對 Actions (動作)，選擇 Reboot (重新啟動)。

當執行個體重新啟動後，IAM 角色即與資料庫叢集產生關聯。

如需叢集參數群組的詳細資訊，請參閱 Aurora MySQL 組態參數。

CLI

使用 將 IAM 角色與資料庫叢集建立關聯 AWS CLI

1. 從 呼叫 `add-role-to-db-cluster`命令，將 IAM 角色ARNs AWS CLI 新增至資料庫叢集，如下所示。

```
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-
arn arn:aws:iam::123456789012:role/AllowAuroraS3Role
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-
arn arn:aws:iam::123456789012:role/AllowAuroraLambdaRole
```

2. 如果您使用預設的資料庫叢集參數群組，請建立新的資料庫叢集參數群組。如果您已使用自訂的資料庫參數群組，則可以使用該群組，而不需要建立新的資料庫叢集參數群組。

   若要建立新的資料庫叢集參數群組，請從 呼叫 `create-db-cluster-parameter-group`命令 AWS CLI，如下所示。

```
PROMPT> aws rds create-db-cluster-parameter-group  --db-cluster-parameter-group-
name AllowAWSAccess \
     --db-parameter-group-family aurora5.7 --description "Allow access to Amazon S3
 and AWS Lambda"
```

   若為 Aurora MySQL 5.7 相容的資料庫叢集，請對 `aurora-mysql5.7` 指定 `--db-parameter-group-family`。若為 Aurora MySQL 8.0 相容的資料庫叢集，請對 `aurora-mysql8.0` 指定 `--db-parameter-group-family`。

3. 在資料庫叢集參數群組中，設定一或多個適當的叢集層級參數，以及相關的 IAM 角色 ARN 值，如下所示。

```
PROMPT> aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name
 AllowAWSAccess \
    --parameters
 "ParameterName=aws_default_s3_role,ParameterValue=arn:aws:iam::123456789012:role/
AllowAuroraS3Role,method=pending-reboot" \
```

```
    --parameters
 "ParameterName=aws_default_lambda_role,ParameterValue=arn:aws:iam::123456789012:role/
AllowAuroraLambdaRole,method=pending-reboot"
```

4. 將資料庫叢集修改為使用新的資料庫叢集參數群組，然後重新啟動叢集，如下所示。

```
PROMPT> aws rds modify-db-cluster --db-cluster-identifier my-cluster --db-cluster-
parameter-group-name AllowAWSAccess
PROMPT> aws rds reboot-db-instance --db-instance-identifier my-cluster-primary
```

當執行個體重新啟動後，IAM 角色與資料庫叢集就產生關聯。

如需叢集參數群組的詳細資訊，請參閱 Aurora MySQL 組態參數。

啟用從 Amazon Aurora 到其他服務的網路通訊 AWS

若要 AWS 搭配 Amazon Aurora 使用某些其他服務，Aurora 資料庫叢集的網路組態必須允許對這些服務的端點進行傳出連線。下列操作需要此網路組態。

- 叫用 AWS Lambda 函數。若要了解此功能，請參閱 使用 Aurora MySQL 原生函式叫用 Lambda 函式。
- 從 Amazon S3 存取檔案。若要了解此功能，請參閱 從 Amazon S3 儲存貯體中的文字檔案將資料載入 Amazon Aurora MySQL 資料庫叢集 和 將來自 Amazon Aurora MySQL 資料庫叢集的資料儲存至 Amazon S3 儲存貯體中的文字檔案。
- 需要存取 AWS KMS endpoint. AWS KMS access 才能搭配 Aurora MySQL 使用資料庫活動串流。若要了解此功能，請參閱 使用資料庫活動串流來監控 Amazon Aurora。
- 存取 SageMaker AI 端點。SageMaker AI 存取需要使用 SageMaker AI 機器學習搭配 Aurora MySQL。若要了解此功能，請參閱 將 Amazon Aurora Machine Learning 與 Aurora MySQL 搭配使用。

如果無法連接至服務端點，Aurora 會傳回下列錯誤訊息。

```
ERROR 1871 (HY000): S3 API returned error: Network Connection
```

```
ERROR 1873 (HY000): Lambda API returned error: Network Connection. Unable to connect to
 endpoint
```

```
ERROR 1815 (HY000): Internal error: Unable to initialize S3Stream
```

對於使用 Aurora MySQL 的資料庫活動串流，如果資料庫叢集無法存取 AWS KMS 端點，則活動串流會停止運作。Aurora 會使用 RDS 事件通知您有關此問題。

如果您在使用對應的 AWS 服務時遇到這些訊息，請檢查您的 Aurora 資料庫叢集是公有還是私有。如果 Aurora 資料庫叢集是私有，則必須設定來啟用連線。

Aurora 資料庫叢集必須標記為可公開存取，才是公有。如果您查看 中資料庫叢集的詳細資訊 AWS 管理主控台，如果是這種情況，可公開存取為是。資料庫叢集也必須在 Amazon VPC 公有子網路中。如需可公開存取之資料庫執行個體的詳細資訊，請參閱在 VPC 中使用資料庫叢集。如需公有 Amazon VPC 子網路的詳細資訊，請參閱您的 VPC 與子網路。

如果 Aurora 資料庫叢集不可公開存取且位於 VPC 公有子網路中，則為私有。您可能擁有私有的資料庫叢集，而且想要使用需要此網路組態的其中一項功能。若是如此，請將叢集設定為可透過網路位址轉譯 (NAT) 來連接至網際網路位址。作為 Amazon S3、Amazon SageMaker AI 和 的替代方案 AWS Lambda，您可以改為將 VPC 設定為具有與資料庫叢集路由表相關聯之其他服務的 VPC 端點，請參閱 在 VPC 中使用資料庫叢集。如需在 VPC 中設定 NAT 的詳細資訊，請參閱 NAT 閘道。如需設定 VPC 端點的詳細資訊，請參閱 VPC 端點。您也可以建立 S3 閘道端點來存取 S3 儲存貯體。如需詳細資訊，請參閱 Amazon S3 的閘道端點。

您可能還必須在 VPC 安全群組的傳出規則中，為網路存取控制清單 (ACL) 開啟暫時性連接埠。如需網路 ACL 的暫時性連接埠詳細資訊，請參閱《Amazon Virtual Private Cloud 使用者指南》中的暫時性連接埠。

相關主題

- 將 Aurora 與其他 AWS 服務整合
- 管理 Amazon Aurora 資料庫叢集

從 Amazon S3 儲存貯體中的文字檔案將資料載入 Amazon Aurora MySQL 資料庫叢集

您可以使用 `LOAD DATA FROM S3` 或 `LOAD XML FROM S3` 陳述式，從存放在 Amazon S3 儲存貯體的檔案載入資料。在 Aurora MySQL 中，檔案會先儲存在本機磁碟上，然後匯入資料庫。匯入資料庫完成後，系統會刪除本機檔案。

> **Note**
>
> 不支援 Aurora Serverless v1 從文字檔案將資料載入資料表。Aurora Serverless v2 則支援。

內容

- 授權 Aurora 存取 Amazon S3
- 授權在 Amazon Aurora MySQL 中載入資料
- 指定 Amazon S3 儲存貯體的路徑 (URI)
- LOAD DATA FROM S3
  - 語法
  - Parameters
  - 使用資訊清單指定要載入的資料檔案
    - 使用 aurora_s3_load_history 資料表來驗證載入的檔案
  - 範例
- LOAD XML FROM S3
  - 語法
  - Parameters

授權 Aurora 存取 Amazon S3

您必須先授權 Aurora MySQL 資料庫叢集存取 Amazon S3，才能從 Amazon S3 儲存貯體載入資料。

授權 Aurora MySQL 存取 Amazon S3

1. 建立 AWS Identity and Access Management (IAM) 政策，提供允許 Aurora MySQL 資料庫叢集存取 Amazon S3 的儲存貯體和物件許可。如需說明，請參閱建立 IAM 政策來存取 Amazon S3 資源。

   > **Note**
   >
   > 在 Aurora MySQL 3.05 及更高版本中，您可以載入使用客戶受管 AWS KMS keys加密的物件。若要這麼做，請在 IAM 政策中包含 `kms:Decrypt` 許可。如需詳細資訊，請參閱建立 IAM 政策以存取 AWS KMS資源。

您不需要此許可，即可載入使用 AWS 受管金鑰 或 Amazon S3 受管金鑰 (SSE-S3) 加密的物件。

2. 建立 IAM 角色，並將您於建立 IAM 政策來存取 Amazon S3 資源中建立的 IAM 政策連接至新的 IAM 角色。如需說明，請參閱「建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務」。
3. 確保資料庫叢集使用自訂資料庫叢集參數群組。

   如需建立自訂資料庫叢集參數群組的詳細資訊，請參閱在 Amazon Aurora 中建立資料庫叢集參數群組。
4. 對於 Aurora MySQL 第 2 版，將 `aurora_load_from_s3_role` 或 `aws_default_s3_role` 資料庫叢集參數設定為新 IAM 角色的 Amazon Resource Name (ARN)。如果 `aurora_load_from_s3_role` 中未指定 IAM 角色，Aurora 會使用 `aws_default_s3_role` 中指定的 IAM 角色。

   對於 Aurora MySQL 第 3 版，請使用 `aws_default_s3_role`。

   如果叢集屬於 Aurora 全球資料庫，請為全球資料庫中的每個 Aurora 叢集設定此參數。雖然 Aurora 全球資料庫中只有主要叢集可以載入資料，但其他叢集可能經由容錯移轉機制提升，而成為主要叢集。

   如需資料庫叢集參數的詳細資訊，請參閱 Amazon Aurora 資料庫叢集和資料庫執行個體參數。
5. 若要允許 Aurora MySQL 資料庫叢集的資料庫使用者存取 Amazon S3，請將您在建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務中建立的角色與資料庫叢集建立關聯。對於 Aurora 全球資料庫，請將此角色與全球資料庫中的每個 Aurora 叢集建立關聯。如需將 IAM 角色與資料庫叢集建立關聯的相關資訊，請參閱將 IAM 角色與 Amazon Aurora MySQL 資料庫叢集建立關聯。
6. 設定 Aurora MySQL 資料庫叢集來允許對外連接至 Amazon S3。如需說明，請參閱「啟用從 Amazon Aurora 到其他服務的網路通訊 AWS」。

   如果 資料庫叢集不可公開存取且位於 VPC 公有子網路中，則為私有。您可以建立 S3 閘道端點來存取 S3 儲存貯體。如需詳細資訊，請參閱 Amazon S3 的閘道端點。

   對於 Aurora 全球資料庫，請對全球資料庫中的每個 Aurora 叢集啟用傳出連線。

授權在 Amazon Aurora MySQL 中載入資料

發出 `LOAD DATA FROM S3` 或 `LOAD XML FROM S3` 陳述式的資料庫使用者必須具備特定的角色或權限，才能發出任一陳述式。在 Aurora MySQL 第 3 版中，您授予 `AWS_LOAD_S3_ACCESS` 角色。在

Aurora MySQL 第 2 版中，您授予 LOAD FROM S3 權限。根據預設，會將適當的角色或權限授予資料庫叢集的管理使用者。您可以使用下列其中一個陳述式，將此權限授予另一個使用者。

請針對 Aurora MySQL 第 3 版使用下列陳述式：

```
GRANT AWS_LOAD_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

> **Tip**
>
> 在 Aurora MySQL 第 3 版中使用角色技術時，您也可以使用 SET ROLE *role_name* 或 SET ROLE ALL 陳述式啟用角色。如果您不熟悉 MySQL 8.0 角色系統，您可以在角色型權限模型進一步了解。如需詳細資訊，請參閱《MySQL 參考手冊》中的使用角色。
> 此僅適用於目前的作用中工作階段。重新連接時，您必須再次執行 SET ROLE 陳述式以授予權限。如需詳細資訊，請參閱 MySQL Reference Manual (MySQL 參考手冊) 中的 SET ROLE 陳述式。
> 您可以使用 activate_all_roles_on_login 資料庫叢集參數，在使用者連線至資料庫執行個體時自動啟動所有角色。設定此參數後，您通常不需要明確呼叫 SET ROLE 陳述式，即可啟用角色。如需詳細資訊，請參閱 MySQL Reference Manual (MySQL 參考手冊) 中的 activate_all_roles_on_login。
> 不過，當不同使用者呼叫預存程序時，您必須在預存程序開始時明確呼叫 SET ROLE ALL 以啟用角色。

請針對 Aurora MySQL 第 2 版使用下列陳述式：

```
GRANT LOAD FROM S3 ON *.* TO 'user'@'domain-or-ip-address'
```

AWS_LOAD_S3_ACCESS 角色和 LOAD FROM S3 權限是 Amazon Aurora MySQL 特定的，而且不適用於外部 MySQL 資料庫或 RDS for MySQL 資料庫執行個體。如果您在 Aurora 資料庫叢集 (複寫來源) 和 MySQL 資料庫 (複寫用戶端) 之間設定複寫，則角色或權限的 GRANT 陳述式會導致複寫停止並產生錯誤。您可以放心略過此錯誤並繼續複寫。若要在 RDS for MySQL 執行個體上略過此錯誤，請使用 mysql_rds_skip_repl_error 程序。若要略過外部 MySQL 資料庫上的錯誤，請使用 slave_skip_errors 系統變數 (Aurora MySQL 第 2 版) 或 replica_skip_errors 系統變數 (Aurora MySQL 第 3 版)。

> **Note**
>
> 資料庫使用者必須具有其所載入資料之資料庫的 INSERT 權限。

指定 Amazon S3 儲存貯體的路徑 (URI)

對存放於 Amazon S3 儲存貯體中的檔案，指定路徑 (URI) 的語法如下。

```
s3-region://amzn-s3-demo-bucket/file-name-or-prefix
```

路徑包含以下值：

- `region`（選用）– 包含要載入之 Amazon S3 儲存貯 AWS 體的區域。此值是選用的。如果未指定 `region` 值，Aurora 會從資料庫叢集所在同一個區域中的 Amazon S3 載入檔案。
- `bucket-name` – Amazon S3 儲存貯體的名稱，其中包含要載入的資料。支援表示虛擬資料夾路徑的物件字首。
- `file-name-or-prefix` – Amazon S3 文字檔案或 XML 檔案的名稱，或表示一或多個要載入之文字檔案或 XML 檔案的字首。您也可以指定資訊清單檔案，以指出一或多個要載入的文字檔案。如需使用資訊清單檔案從 Amazon S3 載入文字檔案的詳細資訊，請參閱使用資訊清單指定要載入的資料檔案。

複製 S3 儲存貯體中檔案的 URI

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/s3/ ：// 開啟 Amazon S3 主控台。
2. 在導覽窗格中，選擇儲存貯體，然後選擇您要複製其 URI 的儲存貯體。
3. 選取您要從 S3 載入的字首或檔案。
4. 選擇複製 S3 URI。

LOAD DATA FROM S3

您可以使用 `LOAD DATA FROM S3` 陳述式從 MySQL LOAD DATA INFILE 陳述式支援的任何文字檔案格式中載入資料，例如以逗號分隔的文字資料。不支援壓縮檔案。

> **Note**
>
> 確定您的 Aurora MySQL 資料庫叢集允許 S3 的傳出連線。如需詳細資訊，請參閱啟用從 Amazon Aurora 到其他服務的網路通訊 AWS。

語法

```
LOAD DATA [FROM] S3 [FILE | PREFIX | MANIFEST] 'S3-URI'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
    [IGNORE number {LINES | ROWS}]
    [(col_name_or_user_var,...)]
    [SET col_name = expr,...]
```

> **Note**
>
> 在 Aurora MySQL 版本 3.05 及更高版本中，關鍵字 FROM 是選用的。

Parameters

LOAD DATA FROM S3 陳述式會使用下列必要和選用參數。您可以在 MySQL 文件的 LOAD DATA 陳述式中找到某些參數的詳細資訊。

FILE | PREFIX | MANIFEST

識別要從單一檔案、符合指定字首的所有檔案，還是指定資訊清單中的所有檔案來載入資料。FILE 是預設值。

S3-URI

指定要載入之文字檔案或資訊清單檔案的 URI，或要使用的 Amazon S3 字首。請使用指定 Amazon S3 儲存貯體的路徑 (URI)所述的語法來指定 URI。

REPLACE | IGNORE

當輸入資料列與資料庫資料表中現有的資料列具有相同的唯一索引鍵值時，決定要採取什麼動作。

- 如果要讓輸入資料列取代資料表中現有的資料列，請指定 REPLACE。
- 如果要捨棄輸入資料列，請指定 IGNORE。

INTO TABLE

識別要將輸入資料列載入其中的資料庫資料表名稱。

PARTITION

要求在指定之逗號分隔分割區名稱清單所指出的分割區，插入所有輸入資料列。如果有輸入資料列無法插入其中一個指定的分割區，則陳述式會失敗並傳回錯誤。

CHARACTER SET

識別輸入檔案中資料的字元集。

FIELDS | COLUMNS

識別輸入檔案中的欄位或資料欄如何分隔。依預設以 Tab 鍵分隔欄位。

LINES

識別輸入檔案中的行如何分隔。依預設，行會以換行字元 ('\n') 分隔。

IGNORE ***number*** LINES | ROWS

指定忽略輸入檔案開頭一定數量的行或資料列。例如，您可以使用 IGNORE 1 LINES 來跳過含有欄名稱的起始標頭行，或使用 IGNORE 2 ROWS 來跳過輸入檔案中的前兩列資料。如果您也使用 PREFIX，IGNORE 會略過第一個輸入檔案開頭的特定行數或列數。

col_name_or_user_var, ...

指定逗號分隔清單，列出一或多欄名稱，或識別要依名稱載入哪些資料欄的使用者變數。做為此用途的使用者變數名稱必須符合文字檔中的元素名稱，字首為 @。您可以利用使用者變數來存放對應的欄位值，供以後重複使用。

例如，下載陳述式將輸入檔案的第一欄載入 table1 的第一欄，並將 table_column2 的 table1 欄的值設為第二欄的輸入值除以 100。

```
LOAD DATA FROM S3 's3://amzn-s3-demo-bucket/data.txt'
    INTO TABLE table1
    (column1, @var1)
    SET table_column2 = @var1/100;
```

SET

指定逗號分隔清單，列出指派操作，其會將資料表中資料欄的值設為不包括在輸入檔案中的值。

例如，下載陳述式將 table1 的前兩欄設為輸入檔案中前兩欄的值，然後將 column3 中的 table1 的值設為目前的時間戳記。

```
LOAD DATA FROM S3  's3://amzn-s3-demo-bucket/data.txt'
    INTO TABLE table1
    (column1, column2)
    SET column3 = CURRENT_TIMESTAMP;
```

您可以在 SET 指派的右邊使用子查詢。如果子查詢會傳回值來指派給一個欄，則您只能使用純量子查詢。您也不能使用子查詢來從載入的資料表中選取。

如果您從 Amazon S3 儲存貯體載入資料，則無法使用 LOAD DATA FROM S3 陳述式的 LOCAL 關鍵字。

使用資訊清單指定要載入的資料檔案

您可以使用 LOAD DATA FROM S3 陳述式搭配 MANIFEST 關鍵字，以指定 JSON 格式的資訊清單檔案，其中列出要在資料庫叢集的資料表中載入的文字檔案。

下列 JSON 結構描述說明資訊清單檔案的格式和內容。

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "additionalProperties": false,
    "definitions": {},
    "id": "Aurora_LoadFromS3_Manifest",
    "properties": {
        "entries": {
            "additionalItems": false,
            "id": "/properties/entries",
            "items": {
                "additionalProperties": false,
                "id": "/properties/entries/items",
                "properties": {
                    "mandatory": {
                        "default": "false",
                        "id": "/properties/entries/items/properties/mandatory",
                        "type": "boolean"
                    },
                    "url": {
                        "id": "/properties/entries/items/properties/url",
```

```
                        "maxLength": 1024,
                        "minLength": 1,
                        "type": "string"
                    }
                },
                "required": [
                    "url"
                ],
                "type": "object"
            },
            "type": "array",
            "uniqueItems": true
        }
    },
    "required": [
        "entries"
    ],
    "type": "object"
}
```

資訊清單檔案中的每個 `url` 必須指定 URL 和儲存貯體名稱，以及檔案的完整物件路徑，而不只是字首。您可以使用資訊清單從不同儲存貯體、不同區域載入檔案，或載入不共用相同字首的檔案。如果 URL 中未指定某個區域，則會使用目標 Aurora 資料庫叢集的區域。下列範例顯示的資訊清單檔案會從不同儲存貯體載入四個檔案。

```
{
  "entries": [
    {
      "url":"s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory":true
    },
    {
      "url":"s3-us-west-2://aurora-bucket-usw2/2013-10-05-customerdata",
      "mandatory":true
    },
    {
      "url":"s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory":false
    },
    {
      "url":"s3://aurora-bucket/2013-10-05-customerdata"
    }
  ]
```

```
}
```

選用的 mandatory 旗標指定如果找不到檔案時，LOAD DATA FROM S3 是否傳回錯誤。mandatory 旗標預設為 false。無論 mandatory 如何設定，如果找不到檔案，LOAD DATA FROM S3 就會終止。

資訊清單檔案可以有任何副檔名。下列範例會搭配上一個範例中的資訊清單 (名為 LOAD DATA FROM S3) 執行 **customer.manifest** 陳述式。

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://aurora-bucket/customer.manifest'
    INTO TABLE CUSTOMER
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, EMAIL);
```

陳述式完成之後，每個成功載入的檔案都有一個項目寫入 aurora_s3_load_history 資料表中。

使用 aurora_s3_load_history 資料表來驗證載入的檔案

每個成功的 LOAD DATA FROM S3 陳述式會針對每個已載入的檔案，以一個項目更新 aurora_s3_load_history 結構描述中的 mysql 資料表。

執行 LOAD DATA FROM S3 陳述式之後，您可以查詢 aurora_s3_load_history 資料表來驗證已載入哪些檔案。若要查看反覆運算之陳述式所載入的檔案，請依陳述式中使用的資訊清單檔案，使用 WHERE 子句來篩選 Amazon S3 URI 上的記錄。如果您之前已用過相同的資訊清單檔案，請使用 timestamp 欄位來篩選結果。

```
select * from mysql.aurora_s3_load_history where load_prefix = 'S3_URI';
```

下表描述 aurora_s3_load_history 資料表中的欄位。

欄位	描述
load_prefix	Load 陳述式中指定的 URI。此 URI 可以映射至下列任何一項： • LOAD DATA FROM S3 FILE 陳述式使用的單一資料檔案 • LOAD DATA FROM S3 PREFIX 陳述式中映射至多個資料檔案的 Amazon S3 字首

欄位	描述
	• LOAD DATA FROM S3 MANIFEST 陳述式使用的單一資訊清單檔案，其中包含要載入的檔案名稱
file_name	使用 load_prefix 欄位所指定的 URI，從 Amazon S3 載入到 Aurora 的檔案名稱。
version_number	由 file_name 欄位所識別之已載入檔案的版本編號 (如果 Amazon S3 儲存貯體有版本編號)。
bytes_loaded	載入的檔案大小 (以位元組為單位)。
load_timestamp	LOAD DATA FROM S3 陳述式完成時的時間戳記。

範例

下列陳述式從 Aurora 資料庫叢集所在同一個區域中的 Amazon S3 儲存貯體載入資料。此陳述式從 *amzn-s3-demo-bucket* Amazon S3 儲存貯體中的檔案 customerdata.txt 讀取逗號分隔資料，然後將資料載入資料表 store-schema.customer-table。

```
LOAD DATA FROM S3 's3://amzn-s3-demo-bucket/customerdata.csv'
    INTO TABLE store-schema.customer-table
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, ADDRESS, EMAIL, PHONE);
```

下列陳述式從不是位於 Aurora 資料庫叢集所在區域中的 Amazon S3 儲存貯體載入資料。此陳述式在 us-west-2 區域的 *amzn-s3-demo-bucket* Amazon S3 儲存貯體中，從符合 employee-data 物件字首的所有檔案中讀取逗號分隔資料，然後將資料載入 employees 資料表。

```
LOAD DATA FROM S3 PREFIX 's3-us-west-2://amzn-s3-demo-bucket/employee_data'
    INTO TABLE employees
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, EMAIL, SALARY);
```

下列陳述式從名為 q1_sales.json 的 JSON 資訊清單檔案所指定的檔案中，將資料載入 sales 資料表。

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://amzn-s3-demo-bucket1/q1_sales.json'
    INTO TABLE sales
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (MONTH, STORE, GROSS, NET);
```

LOAD XML FROM S3

您可以使用 LOAD XML FROM S3 陳述式，從存放在 Amazon S3 儲存貯體的 XML 檔案 (有三種不同的 XML 格式) 載入資料：

- 欄名稱做為 <row> 元素的屬性。屬性值指出資料表欄位的內容。

  ```
  <row column1="value1" column2="value2" .../>
  ```

- 欄名稱做為 <row> 元素的子元素。子元素的值指出資料表欄位的內容。

  ```
  <row>
    <column1>value1</column1>
    <column2>value2</column2>
  </row>
  ```

- 在 name 元素的 <field> 元素中，<row> 屬性中的欄名稱。<field> 元素的值指出資料表欄位的內容。

  ```
  <row>
    <field name='column1'>value1</field>
    <field name='column2'>value2</field>
  </row>
  ```

語法

```
LOAD XML FROM S3 'S3-URI'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [ROWS IDENTIFIED BY '<element-name>']
    [IGNORE number {LINES | ROWS}]
    [(field_name_or_user_var,...)]
```

```
    [SET col_name = expr,...]
```

Parameters

LOAD XML FROM S3 陳述式會使用下列必要和選用參數。您可以在 MySQL 文件的 LOAD XML 陳述式中找到某些參數的詳細資訊。

FILE | PREFIX

識別要從單一檔案還是從符合指定字首的所有檔案中載入資料。FILE 是預設值。

REPLACE | IGNORE

當輸入資料列與資料庫資料表中現有的資料列具有相同的唯一索引鍵值時，決定要採取什麼動作。

- 如果要讓輸入資料列取代資料表中現有的資料列，請指定 REPLACE。
- 如果要捨棄輸入資料列，請指定 IGNORE。IGNORE 是預設值。

INTO TABLE

識別要將輸入資料列載入其中的資料庫資料表名稱。

PARTITION

要求在指定之逗號分隔分割區名稱清單所指出的分割區，插入所有輸入資料列。如果有輸入資料列無法插入其中一個指定的分割區，則陳述式會失敗並傳回錯誤。

CHARACTER SET

識別輸入檔案中資料的字元集。

ROWS IDENTIFIED BY

識別元素名稱，其會識別輸入檔案中的一列。預設值為 <row>。

IGNORE ***number*** LINES | ROWS

指定忽略輸入檔案開頭一定數量的行或資料列。例如，您可以使用 IGNORE 1 LINES 來跳過文字檔案的第一行，或使用 IGNORE 2 ROWS 來跳過輸入 XML 中的前兩列資料。

field_name_or_user_var, ...

指定逗號分隔清單，列出一或多個 XML 元素名稱，或識別要依名稱載入哪些元素的使用者變數。做為此用途的使用者變數名稱必須符合 XML 檔案中的元素名稱，字首為 @。您可以利用使用者變數來存放對應的欄位值，供以後重複使用。

例如，下載陳述式將輸入檔案的第一欄載入 table1 的第一欄，並將 table_column2 的 table1 欄的值設為第二欄的輸入值除以 100。

```
LOAD XML FROM S3 's3://amzn-s3-demo-bucket/data.xml'
   INTO TABLE table1
   (column1, @var1)
   SET table_column2 = @var1/100;
```

SET

指定逗號分隔清單，列出指派操作，其會將資料表中資料欄的值設為不包括在輸入檔案中的值。

例如，下載陳述式將 table1 的前兩欄設為輸入檔案中前兩欄的值，然後將 column3 中的 table1 的值設為目前的時間戳記。

```
LOAD XML FROM S3 's3://amzn-s3-demo-bucket/data.xml'
   INTO TABLE table1
   (column1, column2)
   SET column3 = CURRENT_TIMESTAMP;
```

您可以在 SET 指派的右邊使用子查詢。如果子查詢會傳回值來指派給一個欄，則您只能使用純量子查詢。您也不能使用子查詢，從載入的資料表中選取。

將來自 Amazon Aurora MySQL 資料庫叢集的資料儲存至 Amazon S3 儲存貯體中的文字檔案

您可以使用 SELECT INTO OUTFILE S3 陳述式從 Amazon Aurora MySQL 資料庫叢集查詢資料，然後將資料儲存至 Amazon S3 儲存貯體中存放的文字檔案。在 Aurora MySQL 中，檔案會先儲存在本機磁碟上，然後匯出至 S3。匯出完成後，系統會刪除本機檔案。

您可以使用 Amazon S3 受管金鑰 (SSE-S3) 或 AWS KMS key (SSE-KMS： AWS 受管金鑰 或客戶受管金鑰) 來加密 Amazon S3 儲存貯體。

LOAD DATA FROM S3 陳述式可以使用 SELECT INTO OUTFILE S3 陳述式所建立的檔案，以將資料載入 Aurora 資料庫叢集。如需詳細資訊，請參閱從 Amazon S3 儲存貯體中的文字檔案將資料載入 Amazon Aurora MySQL 資料庫叢集。

> **Note**
>
> Aurora Serverless v1 資料庫叢集不支援此功能。Aurora Serverless v2 資料庫叢集支援它。

您也可以使用 AWS 管理主控台 AWS CLI、 或 Amazon RDS API，將資料庫叢集資料和資料庫叢集快照資料儲存至 Amazon S3。如需詳細資訊，請參閱將資料庫叢集資料匯出至 Amazon S3及將資料庫叢集快照資料匯出至 Amazon S3。

內容

- 授權 Aurora MySQL 存取 Amazon S3
- 授權在 Aurora MySQL 中儲存資料
- 指定 Amazon S3 儲存貯體的路徑
- 建立資訊清單以列出資料檔案
- SELECT INTO OUTFILE S3
  - 語法
  - Parameters
  - 考量事項
  - 範例

授權 Aurora MySQL 存取 Amazon S3

您必須先授權 Aurora MySQL 資料庫叢集存取 Amazon S3，才能將資料儲存至 Amazon S3 儲存貯體。

授權 Aurora MySQL 存取 Amazon S3

1. 建立 AWS Identity and Access Management (IAM) 政策，提供允許 Aurora MySQL 資料庫叢集存取 Amazon S3 的儲存貯體和物件許可。如需說明，請參閱建立 IAM 政策來存取 Amazon S3 資源。

   > **Note**
   >
   > 在 Aurora MySQL 3.05 版及更高版本中，您可以使用 AWS KMS 客戶受管金鑰來加密物件。若要這麼做，請在 IAM 政策中包含 `kms:GenerateDataKey` 許可。如需詳細資訊，請參閱建立 IAM 政策以存取 AWS KMS資源。
   > 您不需要此許可，即可使用 AWS 受管金鑰 或 Amazon S3 受管金鑰 (SSE-S3) 加密物件。

2. 建立 IAM 角色，並將您於建立 IAM 政策來存取 Amazon S3 資源中建立的 IAM 政策連接至新的 IAM 角色。如需說明，請參閱「建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務」。
3. 對於 Aurora MySQL 第 2 版，將 `aurora_select_into_s3_role` 或 `aws_default_s3_role` 資料庫叢集參數設定為新 IAM 角色的 Amazon Resource Name (ARN)。如果 `aurora_select_into_s3_role` 中未指定 IAM 角色，Aurora 會使用 `aws_default_s3_role` 中指定的 IAM 角色。

   對於 Aurora MySQL 第 3 版，請使用 `aws_default_s3_role`。

   如果叢集屬於 Aurora 全球資料庫，請為全球資料庫中的每個 Aurora 叢集設定此參數。

   如需資料庫叢集參數的詳細資訊，請參閱 Amazon Aurora 資料庫叢集和資料庫執行個體參數。
4. 若要允許 Aurora MySQL 資料庫叢集的資料庫使用者存取 Amazon S3，請將您在建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務中建立的角色與資料庫叢集建立關聯。

   對於 Aurora 全球資料庫，請將此角色與全球資料庫中的每個 Aurora 叢集建立關聯。

   如需將 IAM 角色與資料庫叢集建立關聯的相關資訊，請參閱將 IAM 角色與 Amazon Aurora MySQL 資料庫叢集建立關聯。
5. 設定 Aurora MySQL 資料庫叢集來允許對外連接至 Amazon S3。如需說明，請參閱「啟用從 Amazon Aurora 到其他服務的網路通訊 AWS」。

   對於 Aurora 全球資料庫，請對全球資料庫中的每個 Aurora 叢集啟用傳出連線。

授權在 Aurora MySQL 中儲存資料

發出 `SELECT INTO OUTFILE S3` 陳述式的資料庫使用者必須具備特定的角色或權限。在 Aurora MySQL 第 3 版中，您授予 `AWS_SELECT_S3_ACCESS` 角色。在 Aurora MySQL 第 2 版中，您授予 `SELECT INTO S3` 權限。根據預設，會將適當的角色或權限授予資料庫叢集的管理使用者。您可以使用下列其中一個陳述式，將此權限授予另一個使用者。

請針對 Aurora MySQL 第 3 版使用下列陳述式：

```
GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

> **Tip**
>
> 在 Aurora MySQL 第 3 版中使用角色技術時，您也可以使用 `SET ROLE role_name` 或 `SET ROLE ALL` 陳述式啟用角色。如果您不熟悉 MySQL 8.0 角色系統，您可以在角色型權限模型進一步了解。如需詳細資訊，請參閱《MySQL 參考手冊》中的使用角色。
> 此僅適用於目前的作用中工作階段。重新連接時，您必須再次執行 `SET ROLE` 陳述式以授予權限。如需詳細資訊，請參閱 MySQL Reference Manual (MySQL 參考手冊) 中的 SET ROLE 陳述式。
> 您可以使用 `activate_all_roles_on_login` 資料庫叢集參數，在使用者連線至資料庫執行個體時自動啟動所有角色。設定此參數後，您通常不需要明確呼叫 `SET ROLE` 陳述式，即可啟用角色。如需詳細資訊，請參閱 MySQL Reference Manual (MySQL 參考手冊) 中的 activate_all_roles_on_login。
> 不過，當不同使用者呼叫預存程序時，您必須在預存程序開始時明確呼叫 `SET ROLE ALL` 以啟用角色。

請針對 Aurora MySQL 第 2 版使用下列陳述式：

```
GRANT SELECT INTO S3 ON *.* TO 'user'@'domain-or-ip-address'
```

`AWS_SELECT_S3_ACCESS` 角色和 `SELECT INTO S3` 權限是 Amazon Aurora MySQL 特定的，而且不適用於 MySQL 資料庫或 RDS for MySQL 資料庫執行個體。如果您在 Aurora MySQL 資料庫叢集 (複寫來源) 和 MySQL 資料庫 (複寫用戶端) 之間設定複寫，則角色或權限的 GRANT 陳述式會導致複寫停止並產生錯誤。您可以放心略過此錯誤並繼續複寫。若要在 RDS for MySQL 資料庫執行個體上略過此錯誤，請使用 mysql_rds_skip_repl_error 程序。若要略過外部 MySQL 資料庫上的錯誤，請使用 slave_skip_errors 系統變數 (Aurora MySQL 第 2 版) 或 replica_skip_errors 系統變數 (Aurora MySQL 第 3 版)。

指定 Amazon S3 儲存貯體的路徑

指定路徑將資料和資訊清單存放在 Amazon S3 儲存貯體的語法，類似於 `LOAD DATA FROM S3 PREFIX` 陳述式中使用的語法，如下所示。

```
s3-region://bucket-name/file-prefix
```

路徑包含以下值：

- region （選用） – 包含要儲存資料之 Amazon S3 儲存貯體 AWS 的區域。此值是選用的。如果未指定 region 值，Aurora 會將檔案儲存至資料庫叢集所在同一個區域中的 Amazon S3。
- bucket-name – 供儲存資料的 Amazon S3 儲存貯體的名稱。支援表示虛擬資料夾路徑的物件字首。
- file-prefix – Amazon S3 物件字首，指出要儲存在 Amazon S3 中的檔案。

SELECT INTO OUTFILE S3 陳述式建立的資料檔案使用下列路徑，其中 *00000* 代表從零開始的 5 位數整數。

```
s3-region://bucket-name/file-prefix.part_00000
```

例如，假設 SELECT INTO OUTFILE S3 陳述式指定 s3-us-west-2://bucket/prefix 做為路徑來存放資料檔案，並建立三個資料檔案。指定的 Amazon S3 儲存貯體包含下列資料檔案。

- s3-us-west-2://bucket/prefix.part_00000
- s3-us-west-2://bucket/prefix.part_00001
- s3-us-west-2://bucket/prefix.part_00002

建立資訊清單以列出資料檔案

您可以使用 SELECT INTO OUTFILE S3 陳述式搭配 MANIFEST ON 選項，以建立 JSON 格式的資訊清單檔案，其中列出陳述式要建立的文字檔案。LOAD DATA FROM S3 陳述式可以使用資訊清單檔案，將資料檔案載入回 Aurora MySQL 資料庫叢集。如需使用資訊清單檔案從 Amazon S3 將資料檔案載入 Aurora MySQL 資料庫叢集的詳細資訊，請參閱使用資訊清單指定要載入的資料檔案。

SELECT INTO OUTFILE S3 陳述式所建立的資訊清單中包含的資料檔案，依陳述式建立它們的順序列出。例如，假設 SELECT INTO OUTFILE S3 陳述式指定 s3-us-west-2://bucket/prefix 做為路徑來存放資料檔案，並建立三個資料檔案和一個資訊清單檔案。指定的 Amazon S3 儲存貯體包含名為 s3-us-west-2://bucket/prefix.manifest 的資訊清單檔案，其中包含下列資訊。

```
{
  "entries": [
    {
      "url":"s3-us-west-2://bucket/prefix.part_00000"
    },
    {
      "url":"s3-us-west-2://bucket/prefix.part_00001"
```

```
    },
    {
      "url":"s3-us-west-2://bucket/prefix.part_00002"
    }
  ]
}
```

SELECT INTO OUTFILE S3

您可以使用 SELECT INTO OUTFILE S3 陳述式從資料庫叢集查詢資料，然後將資料直接儲存至 Amazon S3 儲存貯體中存放的分隔文字檔案。

不支援壓縮檔案。從 Aurora MySQL 2.09.0 版開始支援加密的檔案。

語法

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
        [HIGH_PRIORITY]
        [STRAIGHT_JOIN]
        [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
        [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
        [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
        [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
         [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
INTO OUTFILE S3 's3_uri'
[CHARACTER SET charset_name]
    [export_options]
    [MANIFEST {ON | OFF}]
    [OVERWRITE {ON | OFF}]
    [ENCRYPTION {ON | OFF | SSE_S3 | SSE_KMS ['cmk_id']}]

export_options:
    [FORMAT {CSV|TEXT} [HEADER]]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
```

```
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
]
```

Parameters

SELECT INTO OUTFILE S3 陳述式會使用下列必要和選用參數，這些參數專供 Aurora 使用。

s3-uri

指定要使用之 Amazon S3 字首的 URI。使用 指定 Amazon S3 儲存貯體的路徑 中描述的語法。

FORMAT {CSV|TEXT} [HEADER]

選擇性地以 CSV 格式儲存資料。

TEXT 選項為預設值，並會產生現有的 MySQL 匯出格式。

CSV 選項會產生逗號分隔的資料值。CSV 格式會遵循 RFC-4180 中的規格。如果您指定選用的關鍵字 HEADER，則輸出檔包含一個標題列。標題列的標籤對應於 SELECT 陳述式中的欄名稱。您可以使用 CSV 檔案來訓練資料模型，以便與 AWS ML 服務搭配使用。如需搭配 AWS ML 服務使用匯出 Aurora 資料的詳細資訊，請參閱 將資料匯出至 Amazon S3 進行 SageMaker AI 模型訓練 (進階)。

MANIFEST {ON | OFF}

指出是否在 Amazon S3 中建立資訊清單檔案。資訊清單檔案是 JavaScrip 物件標記法 (JSON) 檔案，可供 LOAD DATA FROM S3 MANIFEST 陳述式用來將資料載入 Aurora 資料庫叢集。如需有關 LOAD DATA FROM S3 MANIFEST 的詳細資訊，請參閱 從 Amazon S3 儲存貯體中的文字檔案將資料載入 Amazon Aurora MySQL 資料庫叢集。

如果在查詢中指定 MANIFEST ON，則在建立和上傳所有資料檔案之後，就會在 Amazon S3 中建立資訊清單檔案。資訊清單檔案是使用下列路徑來建立：

```
s3-region://bucket-name/file-prefix.manifest
```

如需資訊清單檔案之內容格式的詳細資訊，請參閱建立資訊清單以列出資料檔案。

OVERWRITE {ON | OFF}

指出是否覆寫所指定 Amazon S3 儲存貯體中的現有檔案。如果指定 OVERWRITE ON，當現有檔案與 s3-uri 所指定之 URI 中的檔案字首相符時，就會覆寫檔案。否則會發生錯誤。

ENCRYPTION {ON | OFF | SSE_S3 | SSE_KMS ['*cmk_id*']}

指出是否搭配 Amazon S3 受管金鑰 (SSE-S3) 或 AWS KMS keys (SSE-KMS，包括 AWS 受管金鑰 和 客戶受管金鑰) 使用伺服器端加密。SSE_S3 和 SSE_KMS 設定適用於 Aurora MySQL 3.05 版及更新版本。

您也可以使用 aurora_select_into_s3_encryption_default 工作階段變數，而不是如下列範例所示的 ENCRYPTION 子句。使用 SQL 子句或工作階段變數，但不同時使用兩者。

```
set session set session aurora_select_into_s3_encryption_default={ON | OFF | SSE_S3
 | SSE_KMS};
```

SSE_S3 和 SSE_KMS 設定適用於 Aurora MySQL 3.05 版及更新版本。

當您將 aurora_select_into_s3_encryption_default 設定為下列值時：

- OFF – 會遵循 S3 儲存貯體的預設加密政策。aurora_select_into_s3_encryption_default 的預設值為 OFF。
- ON 或 SSE_S3 - S3 物件會使用 Amazon S3 受管金鑰 (SSE-S3) 進行加密。
- SSE_KMS – S3 物件使用 加密 AWS KMS key。

  在這種情況下，您還須包括工作階段變數 aurora_s3_default_cmk_id，例如：

  ```
  set session aurora_select_into_s3_encryption_default={SSE_KMS};
  set session aurora_s3_default_cmk_id={NULL | 'cmk_id'};
  ```

  - 當 aurora_s3_default_cmk_id 是 NULL，S3 物件會使用 AWS 受管金鑰進行加密。
  - 當 aurora_s3_default_cmk_id 是非空字串 cmk_id，S3 物件會使用客戶受管金鑰進行加密。

    cmk_id 的值不可以是空字串。

當您使用 SELECT INTO OUTFILE S3 命令時，Aurora 會按下列方式確定加密：

- 如果 SQL 命令中存在 ENCRYPTION 子句，則 Aurora 僅依賴 ENCRYPTION 的值，且不會使用工作階段變數。
- 如果 ENCRYPTION 子句不存在，Aurora 會依賴工作階段變數的值。

如需詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的使用伺服器端加密搭配 Amazon S3 受管金鑰 (SSE-S3) 和使用伺服器端加密搭配AWS KMS 金鑰 (SSE-KMS)。

您可以在 MySQL 文件的 SELECT 陳述式和 LOAD DATA 陳述式中找到其他參數的詳細資訊。

考量事項

寫入 Amazon S3 儲存貯體的檔案數目，取決於 SELECT INTO OUTFILE S3 陳述式所選取的資料量和 Aurora MySQL 的檔案大小臨界值。預設的檔案大小臨界值為 6 GB。如果陳述式所選取的資料小於檔案大小臨界值，則只會建立單一檔案，否則會建立多個檔案。關於此陳述式所建立的檔案，其他考量包括：

- Aurora MySQL 保證資料檔案中的列分割不會跨越檔案界限。若為多個檔案，每個資料檔案 (最後一個檔案除外) 的大小通常接近檔案大小臨界值。不過，偶爾低於檔案大小臨界值會導致一列分割在兩個資料檔案中。在此情況下，Aurora MySQL 會建立資料檔案來保持列的完整，但可能大於檔案大小臨界值。
- 因為 Aurora MySQL 中的每個 SELECT 陳述式都以不可分割的交易來執行，如果 SELECT INTO OUTFILE S3 陳述式選取很大的資料集，則執行時可能需要花一些時間。如果陳述式由於任何原因而失敗，您可能需要重新開始發出陳述式。不過，如果陳述式失敗，則已上傳至 Amazon S3 的檔案仍然留在指定的 Amazon S3 儲存貯體中。您可以使用另一個陳述式來上傳剩餘的資料，而不必重新開始。
- 如果要選取的資料量很大 (超過 25 GB)，建議您使用多個 SELECT INTO OUTFILE S3 陳述式將資料儲存至 Amazon S3。每個陳述式應該選取不同的資料部分來儲存，也應該在 file_prefix 參數中指定不同的 s3-uri，以便於儲存資料檔案時使用。使用多個陳述式來分割要選取的資料，可以更輕鬆地從某個陳述式中的錯誤中復原。如果某個陳述式發生錯誤，則只需要重新選取部分資料並上傳至 Amazon S3。使用多個陳述式也有助於避免單一長時間執行的交易，可提升效能。
- 如果多個 SELECT INTO OUTFILE S3 陳述式平行執行來選取資料給 Amazon S3，而且在 file_prefix 參數中使用相同的 s3-uri，則無法確定行為。
- Aurora MySQL 不會將中繼資料上傳至 Amazon S3，例如資料表結構描述或檔案中繼資料。
- 在某些情況下，您可能需要重新執行 SELECT INTO OUTFILE S3 查詢，例如從失敗中復原。在這些情況下，您必須從 Amazon S3 儲存貯體中移除具有相同檔案字首 (在 s3-uri 中指定) 的任何現有資料檔案，或在 OVERWRITE ON 查詢中包含 SELECT INTO OUTFILE S3。

SELECT INTO OUTFILE S3 陳述式會在成功或失敗時傳回一般 MySQL 錯誤號碼和回應。如果您無法存取 MySQL 錯誤號碼和回應，最簡單的方法是在陳述式中指定 MANIFEST ON，即可判斷何時完

成。資訊清單檔案是陳述式寫入的最後一個檔案。換言之，如果您有資訊清單檔案，就表示陳述式已完成。

目前，無法直接監控 SELECT INTO OUTFILE S3 陳述式在執行時的進度。不過，假設您使用此陳述式從 Aurora MySQL 將大量資料寫入 Amazon S3，且知道陳述式所選取的資料大小。在此情況下，您可以監控 Amazon S3 中建立資料檔案的情形，以估計進度。

在作法上，您知道陳述式選取的資料大約每 6 GB，就會在指定的 Amazon S3 儲存貯體中建立一個資料檔案。將選取的資料大小除以 6 GB，即可估計要建立的資料檔案數目。然後，您可以監控陳述式執行時上傳至 Amazon S3 的檔案數目，以估計陳述式的進度。

範例

下列陳述式選取 employees 資料表中的所有資料，並將資料儲存至不是位於 Aurora MySQL 資料庫叢集所在區域中的 Amazon S3 儲存貯體。在此陳述式所建立的資料檔案中，每個欄位的結尾是逗號 (,) 字元，而每一列的結尾是換行 (\n) 字元。如果符合 sample_employee_data 檔案字首的檔案存在於指定的 Amazon S3 儲存貯體中，此陳述式會傳回錯誤。

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n';
```

下列陳述式選取 employees 資料表中的所有資料，並將資料儲存至 Aurora MySQL 資料庫叢集所在同一個區域中的 Amazon S3 儲存貯體。在此陳述式所建立的資料檔案中，每個欄位的結尾是逗號 (,) 字元，而每一列的結尾是換行 (\n) 字元，此外也建立一個資訊清單檔案。如果符合 sample_employee_data 檔案字首的檔案存在於指定的 Amazon S3 儲存貯體中，此陳述式會傳回錯誤。

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    MANIFEST ON;
```

下列陳述式選取 employees 資料表中的所有資料，並將資料儲存至不是位於 Aurora 資料庫叢集所在區域中的 Amazon S3 儲存貯體。在此陳述式所建立的資料檔案中，每個欄位的結尾是逗號 (,) 字元，而每一列的結尾是換行 (\n) 字元。此陳述式會覆寫指定的 Amazon S3 儲存貯體中任何符合 sample_employee_data 檔案字首的現有檔案。

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    OVERWRITE ON;
```

下列陳述式選取 employees 資料表中的所有資料，並將資料儲存至 Aurora MySQL 資料庫叢集所在同一個區域中的 Amazon S3 儲存貯體。在此陳述式所建立的資料檔案中，每個欄位的結尾是逗號 (,) 字元，而每一列的結尾是換行 (\n) 字元，此外也建立一個資訊清單檔案。此陳述式會覆寫指定的 Amazon S3 儲存貯體中任何符合 sample_employee_data 檔案字首的現有檔案。

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    MANIFEST ON
    OVERWRITE ON;
```

從 Amazon Aurora MySQL 資料庫叢集叫用 Lambda 函式

您可以使用原生 AWS Lambda 函數lambda_sync或 從 Amazon Aurora MySQL 相容版本資料庫叢集叫用 函數lambda_async。Aurora 資料庫叢集必須能夠存取 Lambda，您才能從 Aurora MySQL 叫用 Lambda 函式。如需有關授予 Aurora MySQL 存取權的詳細資訊，請參閱授權 Aurora 存取 Lambda。如需 lambda_sync 與 lambda_async 預存函數的相關資訊，請參閱使用 Aurora MySQL 原生函式叫用 Lambda 函式。

您也可以使用預存程序呼叫 AWS Lambda 函數。不過，已棄用預存程序。如果您使用下列其中一個 Aurora MySQL 版本，強烈建議使用 Aurora MySQL 原生函數：

- Aurora MySQL 第 2 版，適用於 MySQL 5.7 相容的叢集。
- Aurora MySQL 3.01 版和更新版本，適用於 MySQL 8.0 相容的叢集。預存程序不適用於 Aurora MySQL 第 3 版。

如需有關授予 Aurora 對 Lambda 的存取權和叫用 Lambda 函數的資訊，請參閱下列主題。

主題

- 授權 Aurora 存取 Lambda
- 使用 Aurora MySQL 原生函式叫用 Lambda 函式

- 使用 Aurora MySQL 預存程序叫用 Lambda 函式 (已棄用)

授權 Aurora 存取 Lambda

在可以從 Aurora MySQL 資料庫叢集叫用 Lambda 函式之前，務必首先授權您的叢集存取 Lambda，。

授權 Aurora MySQL 存取 Lambda

1. 建立 AWS Identity and Access Management (IAM) 政策來提供許可，以允許 Aurora MySQL 資料庫叢集叫用 Lambda 函式。如需說明，請參閱「建立 IAM 政策以存取 AWS Lambda 資源」。
2. 建立 IAM 角色，並將您於建立 IAM 政策以存取 AWS Lambda 資源中建立的 IAM 政策連接至新的 IAM 角色。如需說明，請參閱「建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務」。
3. 將 `aws_default_lambda_role` 資料庫叢集參數設為新的 IAM 角色的 Amazon Resource Name (ARN)。

   如果叢集屬於 Aurora 全球資料庫，請對全球資料庫中的每個 Aurora 叢集套用相同的設定。

   如需資料庫叢集參數的詳細資訊，請參閱 Amazon Aurora 資料庫叢集和資料庫執行個體參數。
4. 若要允許 Aurora MySQL 資料庫叢集的資料庫使用者叫用 Lambda 函式，請將您在建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務中建立的角色與資料庫叢集建立關聯。如需將 IAM 角色與資料庫叢集建立關聯的相關資訊，請參閱將 IAM 角色與 Amazon Aurora MySQL 資料庫叢集建立關聯。

   如果叢集屬於 Aurora 全球資料庫，請將此角色與全球資料庫中的每個 Aurora 叢集建立關聯。
5. 設定 Aurora MySQL 資料庫叢集來允許對外連接至 Lambda。如需說明，請參閱「啟用從 Amazon Aurora 到其他服務的網路通訊 AWS」。

   如果叢集屬於 Aurora 全球資料庫，請對全球資料庫中的每個 Aurora 叢集啟用傳出連線。

使用 Aurora MySQL 原生函式叫用 Lambda 函式

> **Note**
>
> 使用 Aurora MySQL 第 2 版，或 Aurora MySQL 3.01 版和更新版本時，您可以呼叫原生函數 `lambda_sync` 和 `lambda_async`。如需 Aurora MySQL 版本的詳細資訊，請參閱 Amazon Aurora MySQL 的資料庫引擎更新。

您可以透過呼叫原生 AWS Lambda 函數 `lambda_sync`和 ，從 Aurora MySQL 資料庫叢集叫用 函數`lambda_async`。當您想要將 Aurora MySQL 上執行的資料庫與其他 AWS 服務整合時，此方法非常有用。例如，每當資料庫的特定資料表中插入一列時，您可能想要使用 Amazon Simple Notification Service (Amazon SNS) 傳送通知。

內容

- 使用原生函數呼叫 Lambda 函數
  - 在 Aurora MySQL 第 3 版中授予角色
  - 在 Aurora MySQL 第 2 版中授予權限
  - lambda_sync 函數的語法
  - lambda_sync 函數的參數
  - lambda_sync 函數範例
  - lambda_async 函數的語法
  - lambda_async 函數的參數
  - lambda_async 函數範例
  - 叫用觸發條件內的 Lambda 函數

使用原生函數呼叫 Lambda 函數

`lambda_sync` 和 `lambda_async` 函式是內建的原生函式，可同步或非同步叫用 Lambda 函式。如果您在繼續執行另一個動作之前，必須得知 `lambda_sync` 函數的結果，請使用同步函數 Lambda。如果您在繼續執行另一個動作之前，並不需要得知 Lambda 函數的結果，請使用非同步函數 `lambda_async`。

在 Aurora MySQL 第 3 版中授予角色

在 Aurora MySQL 第 3 版中，必須將 `AWS_LAMBDA_ACCESS` 角色授予叫用原生函數的使用者。若要將此角色授予使用者，請以管理使用者身分連接至資料庫執行個體，然後執行下列陳述式。

```
GRANT AWS_LAMBDA_ACCESS TO user@domain-or-ip-address
```

您可以執行下列陳述式來撤銷此角色。

```
REVOKE AWS_LAMBDA_ACCESS FROM user@domain-or-ip-address
```

> **Tip**
>
> 在 Aurora MySQL 第 3 版中使用角色技術時，您也可以使用 `SET ROLE role_name` 或 `SET ROLE ALL` 陳述式啟用角色。如果您不熟悉 MySQL 8.0 角色系統，您可以在角色型權限模型進一步了解。如需詳細資訊，請參閱《MySQL 參考手冊》中的使用角色。
> 此僅適用於目前的作用中工作階段。重新連接時，您必須再次執行 `SET ROLE` 陳述式以授予權限。如需詳細資訊，請參閱 MySQL Reference Manual (MySQL 參考手冊) 中的 SET ROLE 陳述式。
> 您可以使用 `activate_all_roles_on_login` 資料庫叢集參數，在使用者連線至資料庫執行個體時自動啟動所有角色。設定此參數後，您通常不需要明確呼叫 `SET ROLE` 陳述式，即可啟用角色。如需詳細資訊，請參閱 MySQL Reference Manual (MySQL 參考手冊) 中的 activate_all_roles_on_login。
> 不過，當不同使用者呼叫預存程序時，您必須在預存程序開始時明確呼叫 `SET ROLE ALL` 以啟用角色。

如果在嘗試叫用 Lambda 函數時出現如下錯誤，請執行 `SET ROLE` 陳述式。

```
SQL Error [1227] [42000]: Access denied; you need (at least one of) the Invoke Lambda
 privilege(s) for this operation
```

請確定您將角色授予正確的使用者，如 `mysql.users` 資料表項目所示。可能有多個使用者具有相同名稱，但在不同的主機上。根據哪個應用程式或主機正在叫用 `lambda_sync` 函數，MySQL 會根據 `host` 資料欄項目選取最相符的使用者。

在 Aurora MySQL 第 2 版中授予權限

在 Aurora MySQL 第 2 版中，必須將 `INVOKE LAMBDA` 權限授予叫用原生函數的使用者。若要將此權限授予使用者，請以管理使用者身分連接至資料庫執行個體，然後執行下列陳述式。

```
GRANT INVOKE LAMBDA ON *.* TO user@domain-or-ip-address
```

您可以執行下列陳述式來撤銷此權限。

```
REVOKE INVOKE LAMBDA ON *.* FROM user@domain-or-ip-address
```

lambda_sync 函數的語法

您可以指定 lambda_sync 呼叫類型，以同步呼叫 RequestResponse 函數。此函數會在 JSON 承載中傳回 Lambda 呼叫結果。此函數的語法如下。

```
lambda_sync (
  lambda_function_ARN,
  JSON_payload
)
```

lambda_sync 函數的參數

此lambda_sync 函數具有下列參數。

lambda_function_ARN

要呼叫之 Lambda 函數的 Amazon Resource Name (ARN)。

JSON_payload

所呼叫之 Lambda 函數的承載 (JSON 格式)。

> **Note**
>
> Aurora MySQL 第 3 版支援來自 MySQL 8.0 的 JSON 解析函數。不過，Aurora MySQL 第 2 版不包含這些函數。當 Lambda 函數傳回最小單位值時，例如數字或字串，就不需要 JSON 剖析。

lambda_sync 函數範例

下列以 lambda_sync 為基礎的查詢使用函式 ARN，以同步叫用 Lambda 函式 BasicTestLambda。函數的承載為 {"operation": "ping"}。

```
SELECT lambda_sync(
    'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
    '{"operation": "ping"}');
```

lambda_async 函數的語法

您可以指定 lambda_async 呼叫類型，以非同步呼叫 Event 函數。此函數會在 JSON 承載中傳回 Lambda 呼叫結果。此函數的語法如下。

```
lambda_async (
  lambda_function_ARN,
  JSON_payload
)
```

lambda_async 函數的參數

此lambda_async 函數具有下列參數。

lambda_function_ARN

要呼叫之 Lambda 函數的 Amazon Resource Name (ARN)。

JSON_payload

所呼叫之 Lambda 函數的承載 (JSON 格式)。

> **Note**
>
> Aurora MySQL 第 3 版支援來自 MySQL 8.0 的 JSON 解析函數。不過，Aurora MySQL 第 2 版不包含這些函數。當 Lambda 函數傳回最小單位值時，例如數字或字串，就不需要 JSON 剖析。

lambda_async 函數範例

下列以 lambda_async 為基礎的查詢使用函式 ARN，以非同步叫用 Lambda 函式 BasicTestLambda。函數的承載為 {"operation": "ping"}。

```
SELECT lambda_async(
    'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
    '{"operation": "ping"}');
```

叫用觸發條件內的 Lambda 函數

您可以在資料修改陳述式上使用觸發條件來呼叫 Lambda。下列範例會使用 lambda_async 原生函數，並將結果存放在變數中。

```
mysql>SET @result=0;
mysql>DELIMITER //
mysql>CREATE TRIGGER myFirstTrigger
```

```
      AFTER INSERT
          ON Test_trigger FOR EACH ROW
      BEGIN
      SELECT lambda_async(
          'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
          '{"operation": "ping"}')
          INTO @result;
      END; //
mysql>DELIMITER ;
```

> **Note**
>
> 觸發條件不是針對每一 SQL 陳述式來執行一次，而是每修改一列就執行一次，而且一次只針對一列。觸發程序執行時，處理程序是同步的。只有在觸發程序完成時，才會傳回資料修改陳述式。
>
> 從經歷高寫入流量的資料表觸發叫用 AWS Lambda 函數時，請小心。INSERT、 UPDATE和 DELETE觸發會依資料列啟用。具有 INSERT、 UPDATE或 DELETE觸發條件之資料表上的大量寫入工作負載會導致對 AWS Lambda 函數的大量呼叫。

使用 Aurora MySQL 預存程序叫用 Lambda 函式 (已棄用)

您可以透過呼叫 `mysql.lambda_async` 程序，從 Aurora MySQL 資料庫叢集叫用 AWS Lambda 函數。當您想要將 Aurora MySQL 上執行的資料庫與其他 AWS 服務整合時，此方法非常有用。例如，每當資料庫的特定資料表中插入一列時，您可能想要使用 Amazon Simple Notification Service (Amazon SNS) 傳送通知。

內容

- Aurora MySQL 版本考量
- 使用 mysql.lambda_async 程序呼叫 Lambda 函數 (已棄用)
  - 語法
  - Parameters
  - 範例

Aurora MySQL 版本考量

從 Aurora MySQL 第 2 版開始，您可以使用原生函數方法，而不是這些預存程序來叫用 Lambda 函數。如需原生函數的詳細資訊，請參閱使用原生函數呼叫 Lambda 函數。

Aurora MySQL 第 2 版不再支援預存程序 mysql.lambda_async。強烈建議改用原生 Lambda 函數。

在 Aurora MySQL 第 3 版中，無法使用預存程序。

使用 mysql.lambda_async 程序呼叫 Lambda 函數 (已棄用)

mysql.lambda_async 程序是內建的預存程序，可非同步叫用 Lambda 函式。資料庫使用者必須有 EXECUTE 預存程序的 mysql.lambda_async 權限，才能使用此程序。

語法

mysql.lambda_async 程序的語法如下。

```
CALL mysql.lambda_async (
  lambda_function_ARN,
  lambda_function_input
)
```

Parameters

mysql.lambda_async 程序具有下列參數。

lambda_function_ARN

要呼叫之 Lambda 函數的 Amazon Resource Name (ARN)。

lambda_function_input

所呼叫之 Lambda 函數的輸入字串 (JSON 格式)。

範例

在最佳實務上，建議您將 mysql.lambda_async 程序的呼叫包裝在預存程序中，即可供不同來源呼叫，例如觸發條件或用戶端程式碼。此方法有助於避免阻抗不符問題，讓您輕鬆呼叫 Lambda 函數。

> **Note**
>
> 在經歷高寫入流量的資料表上叫用 AWS Lambda 函數時，請小心。INSERT、 UPDATE和 DELETE觸發會依資料列啟用。具有 INSERT、UPDATE 或 DELETE 觸發條件之資料表上的大量寫入工作負載，會導致對 AWS Lambda 函數的大量呼叫。

雖然呼叫 mysql.lambda_async 程序為非同步執行，但觸發條件是同步執行。導致觸發條件大量引發的陳述式不會等待 AWS Lambda 函數呼叫完成，但會等待觸發條件完成才將控制權交回用戶端。

Example範例：叫用 AWS Lambda 函數來傳送電子郵件

下列範例建立預存程序，供您在資料庫程式碼中利用 Lambda 函數來呼叫，以傳送電子郵件。

AWS Lambda 函數

```
import boto3

ses = boto3.client('ses')

def SES_send_email(event, context):

    return ses.send_email(
        Source=event['email_from'],
        Destination={
            'ToAddresses': [
            event['email_to'],
            ]
        },

        Message={
            'Subject': {
            'Data': event['email_subject']
            },
            'Body': {
                'Text': {
                    'Data': event['email_body']
                }
            }
        }
    )
```

預存程序

```
DROP PROCEDURE IF EXISTS SES_send_email;
DELIMITER ;;
  CREATE PROCEDURE SES_send_email(IN email_from VARCHAR(255),
```

```
                                    IN email_to VARCHAR(255),
                                    IN subject VARCHAR(255),
                                    IN body TEXT) LANGUAGE SQL
  BEGIN
    CALL mysql.lambda_async(
         'arn:aws:lambda:us-west-2:123456789012:function:SES_send_email',
         CONCAT('{"email_to" : "', email_to,
             '", "email_from" : "', email_from,
             '", "email_subject" : "', subject,
             '", "email_body" : "', body, '"}')
     );
  END
  ;;
DELIMITER ;
```

叫用預存程序來叫用 AWS Lambda 函式

```
mysql> call SES_send_email('example_from@amazon.com', 'example_to@amazon.com', 'Email
 subject', 'Email content');
```

Example範例：叫用 AWS Lambda 函數從觸發程序發佈事件

下列範例建立預存程序來使用 Amazon SNS 發佈事件。當資料表中新增一列時，程式碼會從觸發條件呼叫程序。

AWS Lambda 函數

```
import boto3

sns = boto3.client('sns')

def SNS_publish_message(event, context):

    return sns.publish(
        TopicArn='arn:aws:sns:us-west-2:123456789012:Sample_Topic',
        Message=event['message'],
        Subject=event['subject'],
        MessageStructure='string'
    )
```

預存程序

```
DROP PROCEDURE IF EXISTS SNS_Publish_Message;
```

```
DELIMITER ;;
CREATE PROCEDURE SNS_Publish_Message (IN subject VARCHAR(255),
                                      IN message TEXT) LANGUAGE SQL
BEGIN
  CALL mysql.lambda_async('arn:aws:lambda:us-
west-2:123456789012:function:SNS_publish_message',
     CONCAT('{ "subject" : "', subject,
            '", "message" : "', message, '" }')
     );
END
;;
DELIMITER ;
```

資料表

```
CREATE TABLE 'Customer_Feedback' (
  'id' int(11) NOT NULL AUTO_INCREMENT,
  'customer_name' varchar(255) NOT NULL,
  'customer_feedback' varchar(1024) NOT NULL,
  PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

觸發條件

```
DELIMITER ;;
CREATE TRIGGER TR_Customer_Feedback_AI
  AFTER INSERT ON Customer_Feedback
  FOR EACH ROW
BEGIN
  SELECT CONCAT('New customer feedback from ', NEW.customer_name),
 NEW.customer_feedback INTO @subject, @feedback;
  CALL SNS_Publish_Message(@subject, @feedback);
END
;;
DELIMITER ;
```

在資料表中插入一列以觸發通知

```
mysql> insert into Customer_Feedback (customer_name, customer_feedback) VALUES ('Sample
 Customer', 'Good job guys!');
```

將 Amazon Aurora MySQL 日誌發佈至 Amazon CloudWatch Logs

您可以設定 Aurora MySQL 資料庫叢集將一般、緩慢、稽核和錯誤日誌資料發佈至 Amazon CloudWatch Logs 中的日誌群組。使用 CloudWatch Logs，您可以執行日誌資料的即時分析，並使用 CloudWatch 來建立警示和檢視指標。您可以使用 CloudWatch Logs 將日誌記錄存放在高耐用性的儲存裝置中。

若要將日誌發佈至 CloudWatch Logs，必須啟用各自的日誌。依預設會啟用錯誤日誌，但您必須明確啟用其他類型的日誌。如需在 MySQL 中啟用日誌的相關資訊，請參閱 MySQL 文件中的選取一般查詢和緩慢查詢日誌輸出目的地。如需啟用 Aurora MySQL 稽核日誌的詳細資訊，請參閱 啟用進階稽核。

Note

- 如果已停止匯出日誌資料，Aurora 不會刪除現有的日誌群組或日誌串流。如果已停止匯出日誌資料，現有的日誌資料會繼續存在於 CloudWatch Logs 中，視日誌保留期間而定，而存放的稽核日誌資料仍會產生費用。您可以使用 CloudWatch Logs 主控台、AWS CLI 或 CloudWatch Logs API 來刪除日誌串流和日誌群組。
- 將稽核日誌發佈至 CloudWatch Logs 的另一種方法是啟用進階稽核，然後建立自訂資料庫叢集參數群組，並將 `server_audit_logs_upload` 參數設為 `1`。`server_audit_logs_upload` 資料庫叢集參數的預設值為 `0`。如需啟用進階稽核的資訊，請參閱 使用進階稽核與 Amazon Aurora MySQL 資料庫叢集搭配。

  若您使用此替代方法，則必須具有 IAM 角色來存取 CloudWatch Logs，並將 `aws_default_logs_role` 叢集等級參數設為此角色的 ARN。如需有關建立角色的詳細資訊，請參閱 設定 IAM 角色以存取 AWS 服務。不過，如果您具有 `AWSServiceRoleForRDS` 服務連結角色，則此角色會提供對 CloudWatch Logs 的存取權限，並複寫任何自定義的角色。如需 Amazon RDS 之服務連結角色的相關資訊，請參閱 使用 Amazon Aurora 的服務連結角色。
- 如果您不要將稽核日誌匯出至 CloudWatch Logs，請確定所有匯出稽核日誌的方法都已停用。這些方法包括 AWS 管理主控台、AWS CLI、RDS API 和 `server_audit_logs_upload` 參數。
- Aurora Serverless v1 資料庫叢集的程序與具有佈建或 Aurora Serverless v2 執行個體的資料庫叢集略有不同。Aurora Serverless v1 資料庫叢集會自動上傳您透過組態參數啟用的所有日誌。

  因此，透過在資料庫叢集參數群組中開啟和關閉不同的日誌類型，您可以開啟或關閉 Aurora Serverless v1 資料庫叢集的日誌上傳。您不會透過 AWS 管理主控台、AWS CLI 或 RDS

API 來修改叢集本身的設定。如需開啟和關閉 Aurora Serverless v1 叢集 MySQL 日誌的詳細資訊，請參閱 Aurora Serverless v1 的參數群組。

主控台

您可以透過主控台推送佈建的叢集 Aurora MySQL 日誌到 CloudWatch Logs。

從主控台發佈 Aurora MySQL 日誌

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您要發佈日誌資料的 Aurora MySQL 資料庫叢集。
4. 選擇 Modify (修改)。
5. 在 Log exports (日誌匯出) 區段中，選擇您要開始發佈至 CloudWatch Logs 的日誌。
6. 選擇 Continue (繼續)，然後在摘要頁面上選擇 Modify DB Cluster (修改資料庫叢集)。

AWS CLI

您可以透過 AWS CLI 推送佈建的叢集 Aurora MySQL 日誌。如要執行此動作，您可以搭配下列選項執行 modify-db-cluster AWS CLI 命令：

- `--db-cluster-identifier`—資料庫叢集識別符
- `--cloudwatch-logs-export-configuration` — 資料庫叢集的哪些日誌類型要匯出至 CloudWatch Logs 的組態設定。

您也可以執行下列其中一個 AWS CLI 命令來發佈 Aurora MySQL 日誌：

- create-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

執行上述其中一個 AWS CLI 命令並指定下列選項：

- `--db-cluster-identifier`—資料庫叢集識別符

- `--engine` — 資料庫引擎。
- `--enable-cloudwatch-logs-exports` — 資料庫叢集的哪些日誌類型要匯出至 CloudWatch Logs 的組態設定。

視您執行的 AWS CLI 命令而定，可能需要其他選項。

Example

下列命令修改現有的 Aurora MySQL 資料庫叢集，以將日誌檔案發佈至 CloudWatch Logs。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["error","general","slowquery","audit","instance"]}'
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["error","general","slowquery","audit","instance"]}'
```

Example

下列命令建立 Aurora MySQL 資料庫叢集，以將日誌檔案發佈至 CloudWatch Logs。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine aurora \
    --enable-cloudwatch-logs-exports
 '["error","general","slowquery","audit","instance"]'
```

在 Windows 中：

```
aws rds create-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine aurora ^
    --enable-cloudwatch-logs-exports
 '["error","general","slowquery","audit","instance"]'
```

RDS API

您可以透過 RDS API 推送佈建的叢集 Aurora MySQL 日誌。若要執行此操作，您可以執行 ModifyDBCluster 操作並指定下列選項：

- `DBClusterIdentifier`—資料庫叢集識別符
- `CloudwatchLogsExportConfiguration` — 資料庫叢集的哪些日誌類型要匯出至 CloudWatch Logs 的組態設定。

您也可以執行下列其中一個 RDS API 操作，以利用 RDS API 來發佈 Aurora MySQL 日誌：

- CreateDBCluster
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime

請搭配下列參數執行 RDS API 操作：

- `DBClusterIdentifier`—資料庫叢集識別符
- `Engine` — 資料庫引擎。
- `EnableCloudwatchLogsExports` — 資料庫叢集的哪些日誌類型要匯出至 CloudWatch Logs 的組態設定。

視您執行的 AWS CLI 命令而定，可能需要其他參數。

在 Amazon CloudWatch 中監控日誌事件

啟用 Aurora MySQL 日誌事件之後，您就可以在 Amazon CloudWatch Logs 中監控事件。針對 Aurora 資料庫叢集，自動會在下列字首下方建立新的日誌群組，其中 *`cluster-name`* 代表資料庫叢集名稱，*`log_type`* 代表日誌類型。

```
/aws/rds/cluster/cluster-name/log_type
```

例如，若您設定匯出功能來包含 mydbcluster 資料庫叢集的緩慢查詢日誌，則緩慢查詢資料會存放在 /aws/rds/cluster/mydbcluster/slowquery 日誌群組中。

來自叢集中所有執行個體的事件，將會使用不同的記錄串流推送至記錄群組。該行為取決於下列哪個條件為真：

- 具指定名稱的記錄群組存在。

  Aurora 會使用現有的記錄群組來匯出叢集的記錄資料。如要以預先定義的記錄保留期間、指標篩選條件和客戶存取權來建立記錄群組，您可使用自動化組態 (例如 AWS CloudFormation)。
- 具指定名稱的記錄群組不存在。

  在執行個體的記錄檔中偵測到相符的記錄項目時，Aurora MySQL 會自動在 CloudWatch Logs 中建立新的記錄群組。記錄群組會使用 Never Expire (永不過期) 的預設記錄保留期間。

  如要變更記錄保留期間，請使用 CloudWatch Logs 主控台、AWS CLI 或 CloudWatch Logs API。如需在 CloudWatch Logs 中變更日誌保留期間的詳細資訊，請參閱在 CloudWatch Logs 中變更日誌資料保留期間。

如要於日誌事件內搜尋資料庫叢集的資訊，請使用 CloudWatch Logs 主控台、AWS CLI 或 CloudWatch Logs API。如需搜尋和篩選日誌資料的詳細資訊，請參閱搜尋和篩選日誌資料。

Amazon Aurora MySQL 實驗室模式

> **Important**
>
> Aurora MySQL 第 2 版中引入了實驗室模式，以啟用快速 DDL 最佳化，從而提高了某些 DDL 操作的效率。在 Aurora MySQL 第 3 版中，實驗室模式已移除，而快速 DDL 已由名為即時 DDL 的 MySQL 8.0 功能取代。

Aurora 實驗室模式的用途，是啟用目前 Aurora 資料庫版本已提供但預設未啟用的 Aurora 功能。雖然生產資料庫叢集不建議使用 Aurora 實驗室模式功能，但您可以使用 Aurora 實驗室模式，為開發與測試環境中的資料庫叢集啟用這些功能。如需進一步了解 Aurora 實驗室模式啟用時，有哪些 Aurora 功能可以使用，請參閱 Aurora 實驗室模式功能。

`aurora_lab_mode` 參數為執行個體層級參數，位於預設的參數群組中。預設參數群組中，此參數設為 `0` (停用)。若要啟用 Aurora 實驗室模式，請建立自訂參數群組，接著將自訂參數群組中的 `aurora_lab_mode` 參數設為 `1` (已啟用)，並修改 Aurora 叢集中一或多個資料庫執行個體來使其使用自訂參數群組。接著連接至適當的執行個體端點來嘗試實驗室模式功能。如需修改資料庫參數群組的相關資訊，請參閱 修改 Amazon Aurora 中的資料庫參數群組中的參數 一文。如需參數群組與 Amazon Aurora 的相關資訊，請參閱 Aurora MySQL 組態參數 一文。

Aurora 實驗室模式功能

Aurora 實驗室模式啟用時，目前可用的 Aurora 功能如下所示。您必須啟用 Aurora 實驗室模式，才能使用表中的任何功能。

快速 DDL

此功能可讓您幾乎立即執行 `ALTER TABLE` *`tbl_name`* `ADD COLUMN` *`col_name column_definition`* 作業。此操作不需要複製資料表即可完成，且對其他 DML 陳述式沒有實質影響。由於它不會耗用複製資料表所需的暫存儲存體，即使是小型執行個體類別上的大型資料表，DDL 陳述式依然很實用。

快速 DDL 目前僅支援在資料表結尾新增沒有預設值，且可為 Null 的資料欄。如需使用此功能的詳細資訊，請參閱 使用快速 DDL 更改 Amazon Aurora 中的資料表 一文。

Amazon Aurora MySQL 的最佳實務

此主題包含有關使用資料或將資料遷移至 Amazon Aurora MySQL 資料庫叢集的最佳實務和選項的資訊。本主題中的資訊摘要說明並重申您可以在 管理 Amazon Aurora 資料庫叢集 中找到的部分指導方針和程序。

內容

- 判斷您連接的資料庫執行個體
- Aurora MySQL 效能和擴展的最佳實務
  - 使用 T 執行個體類別進行開發和測試
  - 使用非同步索引鍵預先提取，將 Amazon Aurora MySQL 索引聯結查詢最佳化
    - 啟用非同步索引鍵預先提取
    - 非同步索引鍵預先提取的最佳化查詢
  - 使用雜湊聯結，將大型 Aurora MySQL 聯結查詢最佳化
    - 啟用雜湊聯結
    - 最佳化雜湊聯結的查詢
  - 使用 Amazon Aurora 為 MySQL 資料庫擴展讀取
  - 最佳化時間戳記操作
  - 虛擬索引 ID 溢出錯誤
- Aurora MySQL 高可用性的最佳實務
  - 使用 Amazon Aurora 搭配 MySQL 資料庫進行災難復原
  - 從 MySQL 遷移至 Amazon Aurora MySQL 時減少停機時間
  - 避免 Aurora MySQL 資料庫執行個體效能變慢、自動重新啟動和容錯移轉
- Aurora MySQL 中 MySQL 功能的建議
  - 在 Aurora MySQL 中使用多執行緒複寫
  - 使用原生 MySQL AWS Lambda 函數叫用函數
  - 避免搭配 Amazon Aurora MySQL 使用 XA 交易
  - DML 陳述式期間保持外部索引鍵的開啟狀態
  - 設定日誌緩衝區的排清頻率
  - 減少和疑難排解 Aurora MySQL 的死結情況
    - 減少 InnoDB 死結情況
    - 監控 InnoDB 死結情況

- 使用 Amazon CloudWatch 指標評估 Aurora MySQL 的資料庫執行個體用量

判斷您連接的資料庫執行個體

若要判斷連線所連接的是 Aurora MySQL 資料庫叢集中的哪個資料庫執行個體，請檢查 `innodb_read_only` 全域變數，如以下範例所示。

```
SHOW GLOBAL VARIABLES LIKE 'innodb_read_only';
```

如果您連線到讀取器資料庫執行個體，則 `innodb_read_only` 變數設定為 `ON`。如果您連線至寫入器資料庫執行個體 (例如已佈建叢集中的主要執行個體)，則此設定為 `OFF`。

如果要將邏輯新增至您的應用程式碼，以平衡工作負載或確保寫入操作使用的是正確的連接，則此方法很實用。

Aurora MySQL 效能和擴展的最佳實務

您可以應用以下最佳實務，改善 Aurora MySQL 叢集的效能和可擴展性。

主題

- 使用 T 執行個體類別進行開發和測試
- 使用非同步索引鍵預先提取，將 Amazon Aurora MySQL 索引聯結查詢最佳化
- 使用雜湊聯結，將大型 Aurora MySQL 聯結查詢最佳化
- 使用 Amazon Aurora 為 MySQL 資料庫擴展讀取
- 最佳化時間戳記操作
- 虛擬索引 ID 溢出錯誤

使用 T 執行個體類別進行開發和測試

對於不支援需時較長的高工作負載應用程式，使用 `db.t2`、`db.t3` 或 `db.t4g` 資料庫執行個體類別的 Amazon Aurora MySQL 執行個體是完美之選。T 執行個體旨在提供適度的基準效能和容量，可視您的工作負載需要大幅提升效能。它們適用非經常或持續使用整個 CPU，但偶爾需要高載的工作負載。建議您在開發、測試伺服器或其他非生產伺服器時，僅使用 T 資料庫執行個體類別。如需 T 執行個體類別的詳細資訊，請參閱爆量效能執行個體。

如果您的 Aurora 叢集大於 40 TB，請勿使用 T 執行個體類別。當您的資料庫有大量資料時，管理結構描述物件的記憶體額外負荷可能會超過 T 執行個體的容量。

請不要在 Amazon Aurora MySQL T 執行個體上啟用 MySQL Performance Schema (MySQL 效能結構描述)。如果已啟用 Performance Schema (效能結構描述)，執行個體可能會用盡記憶體。

> **Tip**
>
> 若您的資料庫有時處於閒置狀態，但有時工作負載相當大，則可使用 Aurora Serverless v2 作為 T 執行個體的替代方案。利用 Aurora Serverless v2，您可定義容量範圍，Aurora 會根據目前的工作負載自動擴展或縮小資料庫。如需用量詳細資料，請參閱 使用 Aurora Serverless v2。若需可與 Aurora Serverless v2 搭配使用的資料庫引擎版本，請參閱 Aurora Serverless v2 的要求與限制。

當您在 Aurora MySQL 資料庫叢集中，使用 T 執行個體作為資料庫執行個體時，我們建議下列作法：

- 在您的資料庫叢集中，所有執行個體請皆使用相同的資料庫執行個體類別。例如：如果您的寫入器執行個體使用 `db.t2.medium`，則我們也建議讀取器執行個體使用 `db.t2.medium`。
- 請不要調整任何與記憶體相關的組態設定，例如 `innodb_buffer_pool_size`。Aurora 可針對 T 執行個體上的記憶體緩衝區使用一組高度調整的預設值。Aurora 需要這些特殊的預設值，才能在記憶體受限的執行個體上執行。如果您在 T 執行個體上變更任何與記憶體相關的設定，即使變更旨在增加緩衝區大小，也很可能會遇到記憶體不足的情況。
- 監控您的 CPU 點數餘額 (`CPUCreditBalance`) 以確保它處於可永續使用的層級。也就是說，CPU 點數的累積速率與使用時速率相同。

  某個執行個體用盡 CPU 點數時，您會看到可用 CPU 立即下降，以及看到執行個體的讀寫延遲增加。此情況會造成執行個體整體效能嚴重降低。

  如果您的 CPU 點數餘額未處於可永續使用的層級，建議您修改資料庫執行個體，以使用其中一個支援的 R 資料庫執行個體類別 (擴展運算)。

  如需監控指標的詳細資訊，請參閱在 Amazon RDS 主控台中檢視指標。
- 監控寫入器執行個體與讀取器執行個體之間的複本延遲 (`AuroraReplicaLag`)。

  如果讀取器執行個體先於寫入器執行個體耗盡 CPU 點數，則產生的延遲可能會導致讀取器執行個體頻繁重新啟動。當應用程式分散在讀取器執行個體之間的讀取操作負載繁重，而同時寫入器執行個體有少量寫入操作時，此結果很常見。

  如果您看到複本延遲持續增加，請確定資料庫叢集中讀取器執行個體的 CPU 點數餘額尚未用盡。

如果您的 CPU 點數餘額未處於可永續使用的層級，建議您修改資料庫執行個體，以使用其中一個支援的 R 資料庫執行個體類別 (擴展運算)。

- 針對已啟用二進位日誌的資料庫叢集，將每個交易的插入數保持在 100 萬個以下。

  如果資料庫叢集的資料庫叢集參數群組的 `binlog_format` 參數設為 `OFF` 以外的值，那麼，如果資料庫叢集收到的交易包含超過 100 萬個要插入的資料列，資料庫叢集可能會遇到記憶體不足情況。您可以監控可釋放記憶體 (`FreeableMemory`) 指標，以判斷資料庫叢集是否用盡可用的記憶體。然後可以檢查寫入操作 (`VolumeWriteIOPS`) 指標，以查看寫入器執行個體接收的寫入器操作負載是否繁重。若是這種情況，建議更新您的應用程式，將一個交易中的插入數限制在 100 萬個以下。或者，您可以修改執行個體，以使用其中一個支援的 R 資料庫執行個體類別 (擴展運算)。

使用非同步索引鍵預先提取，將 Amazon Aurora MySQL 索引聯結查詢最佳化

Aurora MySQL 可以使用非同步索引鍵預先提取 (AKP) 功能來改善聯結索引間資料表之查詢的效能。在 JOIN 查詢要求使用批次索引鍵存取 (BKA) 聯結演算法和多範圍讀取 (MRR) 最佳化功能的情況下，透過預期執行查詢所需的資料列，此功能可藉此改善效能。如需 BKA 和 MRR 的詳細資訊，請參閱 MySQL 文件中的封鎖巢狀迴圈和批次索引鍵存取聯結和多範圍讀取最佳化。

若要利用 AKP 功能，查詢必須同時使用 BKA 和 MRR。一般來說，當查詢的 JOIN 子句使用次要索引，但也需要來自主要索引的一些資料欄時，會發生這類查詢。例如，當 JOIN 子句代表小型外部資料表與大型內部資料表間索引值的對等聯結，並且較大資料表上的索引具有高度選擇性時，您可以使用 AKP。AKP 可與 BKA 和 MRR 共同合作，以在 JOIN 子句的評估期間執行次要至主要索引查詢。AKP 會識別在 JOIN 子句的評估期間執行查詢所需的資料列。然後使用背景執行緒，在執行查詢之前，將包含那些資料列的頁面非同步載入至記憶體。

AKP 適用於 Aurora MySQL 第 2.10 版以上及第 3 版。如需 Aurora MySQL 版本的詳細資訊，請參閱 Amazon Aurora MySQL 的資料庫引擎更新。

啟用非同步索引鍵預先提取

您可以透過將 MySQL 伺服器變數 `aurora_use_key_prefetch` 設定為 `on` 來啟用 AKP 功能。依預設，此值是設為 `on`。不過，在您也啟用 BKA 聯結演算法並停用成本型 MRR 功能之前，無法啟用 AKP。若要這麼做，您必須設定 MySQL 伺服器變數 `optimizer_switch` 的下列值：

- 將 `batched_key_access` 設定為 `on`。此值可控制 BKA 聯結演算法的使用。依預設，此值是設為 `off`。
- 將 `mrr_cost_based` 設定為 `off`。此值可控制成本型 MRR 功能的使用。依預設，此值是設為 `on`。

目前，您只能在工作階段層級設定這些值。下列範例說明如何透過執行 SET 陳述式來設定這些值，來為目前的工作階段啟用 AKP。

```
mysql> set @@session.aurora_use_key_prefetch=on;
mysql> set @@session.optimizer_switch='batched_key_access=on,mrr_cost_based=off';
```

相同地，您可以使用 SET 陳述式來停用 AKP 和 BKA 聯結演算法，以及為目前的工作階段重新啟用成本型 MRR 功能，如以下範例所示。

```
mysql> set @@session.aurora_use_key_prefetch=off;
mysql> set @@session.optimizer_switch='batched_key_access=off,mrr_cost_based=on';
```

如需 batched_key_access 和 mrr_cost_based 最佳化器切換參數的詳細資訊，請參閱 MySQL 文件中的可切換的最佳化。

非同步索引鍵預先提取的最佳化查詢

您可以確認查詢是否可利用 AKP 功能。若要這麼做，請使用 EXPLAIN 陳述式在執行查詢之前描繪查詢。EXPLAIN 陳述式提供要針對指定查詢使用之執行計劃的相關資訊。

在 EXPLAIN 陳述式的輸出中，Extra 資料欄說明執行計劃隨附的其他資訊。如果 AKP 功能適用於查詢中使用的資料表，此資料欄會包括以下其中一個值：

- `Using Key Prefetching`
- `Using join buffer (Batched Key Access with Key Prefetching)`

下列範例示範使用 EXPLAIN 來檢視可利用 AKP 之查詢的執行計劃。

```
mysql> explain select sql_no_cache
    ->     ps_partkey,
    ->     sum(ps_supplycost * ps_availqty) as value
    -> from
    ->     partsupp,
    ->     supplier,
    ->     nation
    -> where
    ->     ps_suppkey = s_suppkey
    ->     and s_nationkey = n_nationkey
```

```
    ->     and n_name = 'ETHIOPIA'
    -> group by
    ->     ps_partkey having
    ->         sum(ps_supplycost * ps_availqty) > (
    ->             select
    ->                 sum(ps_supplycost * ps_availqty) * 0.0000003333
    ->             from
    ->                 partsupp,
    ->                 supplier,
    ->                 nation
    ->             where
    ->                 ps_suppkey = s_suppkey
    ->                 and s_nationkey = n_nationkey
    ->                 and n_name = 'ETHIOPIA'
    ->         )
    -> order by
    ->     value desc;
+----+-------------+----------+------+-----------------------+--------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
| id | select_type | table    | type | possible_keys         | key           | key_len
 | ref                               | rows | filtered | Extra
                            |
+----+-------------+----------+------+-----------------------+--------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
|  1 | PRIMARY     | nation   | ALL  | PRIMARY               | NULL          | NULL
 | NULL                              |   25 |   100.00 | Using where; Using temporary;
 Using filesort                |
|  1 | PRIMARY     | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5
 | dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
                            |
|  1 | PRIMARY     | partsupp | ref  | i_ps_suppkey          | i_ps_suppkey  | 4
 | dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key
 Access with Key Prefetching) |
|  2 | SUBQUERY    | nation   | ALL  | PRIMARY               | NULL          | NULL
 | NULL                              |   25 |   100.00 | Using where
                            |
|  2 | SUBQUERY    | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5
 | dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
                            |
|  2 | SUBQUERY    | partsupp | ref  | i_ps_suppkey          | i_ps_suppkey  | 4
 | dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key
 Access with Key Prefetching) |
```

```
+----+-------------+----------+------+----------------------+--------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
6 rows in set, 1 warning (0.00 sec)
```

如需 EXPLAIN 輸出格式的詳細資訊，請參閱 MySQL 文件中的延伸 EXPLAIN 輸出格式。

使用雜湊聯結，將大型 Aurora MySQL 聯結查詢最佳化

需要使用對等聯結來聯結大量資料時，雜湊聯結可以改善查詢效能。您可以為 Aurora MySQL 啟用雜湊聯結。

雜湊聯結資料欄可以是任何複雜的表達式。在雜湊聯結資料欄中，您可以利用下列方式來比較各個資料類型：

- 您可以比較各類別的精確數值資料類型項目，例如 `int`、`bigint`、`numeric` 和 `bit`。
- 您可以比較各類別的近似數值資料類型項目，例如 `float` 和 `double`。
- 您可以比較各字串類型的項目，以得知字串類型是否有相同的字元集和定序。
- 您可以比較項目的日期和時間戳記資料類型，以得知類型是否相同。

> **Note**
>
> 您無法在不同類別中比較資料類型。

下列限制適用 Aurora MySQL 的雜湊聯結：

- Aurora MySQL 第 2 版不支援左右外部連接，但第 3 版支援。
- 不支援半聯結 (例如子查詢)，除非先將子查詢具體化。
- 不支援多資料表更新或刪除。

> **Note**
>
> 支援單一資料表更新或刪除。

- BLOB 和空間資料類型資料欄不可為雜湊聯結中的聯結資料欄。

啟用雜湊聯結

若要啟用雜湊聯結：

- Aurora MySQL 第 2 版 — 將資料庫參數或資料庫叢集參數 aurora_disable_hash_join 設為 0。若關閉 aurora_disable_hash_join，optimizer_switch 的值將為 hash_join=on。
- Aurora MySQL 第 3 版 — 將 MySQL 伺服器參數 optimizer_switch 設為 block_nested_loop=on。

雜湊聯結在 Aurora MySQL 第 3 版中會預設開啟，在 Aurora MySQL 第 2 版則預設關閉。下列範例說明如何為 Aurora MySQL 第 3 版啟用雜湊聯結。您可以發出陳述式 select @@optimizer_switch，以查看哪些其他設定存在於 SET 參數字串中。更新 optimizer_switch 參數中的某個設定不會清除或修改其他設定。

```
mysql> SET optimizer_switch='block_nested_loop=on';
```

> **Note**
>
> 對於 Aurora MySQL 第 3 版，雜湊聯結支援適用於所有次要版本，預設為開啟。
> 對於 Aurora MySQL 第 2 版，雜湊聯結支援適用於所有次要版本。在 Aurora MySQL 第 2 版中，雜湊聯結功能一律由 aurora_disable_hash_join 值控制。

利用此設定，最佳化器會根據成本、查詢特質和資源可用性選擇使用雜湊聯結。如果成本估算不正確，您可以強制最佳化器選擇某個雜湊聯結。您可以透過將 MySQL 伺服器變數 hash_join_cost_based 設定為 off 來執行此動作。下列範例說明如何強制最佳化器選擇雜湊聯結。

```
mysql> SET optimizer_switch='hash_join_cost_based=off';
```

> **Note**
>
> 此設定會覆寫成本類型最佳化工具的決策。雖然該設定對於測試和開發非常有用，但建議您不要在生產環境中使用它。

最佳化雜湊聯結的查詢

若要了解查詢是否可利用雜湊聯結，請先使用 EXPLAIN 陳述式來描繪查詢。EXPLAIN 陳述式提供要針對指定查詢使用之執行計劃的相關資訊。

在 EXPLAIN 陳述式的輸出中，Extra 資料欄說明執行計劃隨附的其他資訊。如果雜湊聯結適用於查詢中所用的資料表，此資料欄會包括類似以下的值：

- `Using where; Using join buffer (Hash Join Outer table table1_name)`
- `Using where; Using join buffer (Hash Join Inner table table2_name)`

下列範例示範使用 EXPLAIN 來檢視雜湊聯結查詢的執行計劃。

```
mysql> explain SELECT sql_no_cache * FROM hj_small, hj_big, hj_big2
    ->     WHERE hj_small.col1 = hj_big.col1 and hj_big.col1=hj_big2.col1 ORDER BY 1;
+----+-------------+----------+------+---------------+------+---------+------+------
+-------------------------------------------------------------------+
| id | select_type | table    | type | possible_keys | key  | key_len | ref  | rows |
 Extra                                                              |
+----+-------------+----------+------+---------------+------+---------+------+------
+-------------------------------------------------------------------+
|  1 | SIMPLE      | hj_small | ALL  | NULL          | NULL | NULL    | NULL |    6 |
 Using temporary; Using filesort                                    |
|  1 | SIMPLE      | hj_big   | ALL  | NULL          | NULL | NULL    | NULL |   10 |
 Using where; Using join buffer (Hash Join Outer table hj_big)      |
|  1 | SIMPLE      | hj_big2  | ALL  | NULL          | NULL | NULL    | NULL |   15 |
 Using where; Using join buffer (Hash Join Inner table hj_big2)     |
+----+-------------+----------+------+---------------+------+---------+------+------
+-------------------------------------------------------------------+
3 rows in set (0.04 sec)
```

在輸出中，Hash Join Inner table 是用來建置雜湊資料表的資料表，而 Hash Join Outer table 是用來探測雜湊資料表的資料表。

如需延伸 EXPLAIN 輸出格式的詳細資訊，請參閱 MySQL 產品文件中的延伸 EXPLAIN 輸出格式。

在 Aurora MySQL 2.08 及更高版本中，您可以使用 SQL 提示來影響查詢是否使用雜湊聯結，以及要使用哪些資料表來建置和聯結探測端。如需詳細資訊，請參閱Aurora MySQL 提示。

使用 Amazon Aurora 為 MySQL 資料庫擴展讀取

您可以使用 Amazon Aurora 搭配 MySQL 資料庫執行個體來利用 Amazon Aurora 的讀取擴展功能，並為 MySQL 資料庫執行個體擴展讀取工作負載。若要使用 Aurora 來讀取擴展 MySQL 資料庫執行個體，請建立 Aurora MySQL 資料庫叢集，並讓它成為您 MySQL DB 執行個體的讀取複本。然後連線至 Aurora MySQL 叢集以處理讀取查詢。該來源資料庫可以是 RDS for MySQL 資料庫執行個體，也可以是在 Amazon RDS 外部執行的 MySQL 資料庫。如需詳細資訊，請參閱使用 Amazon Aurora 為 MySQL 資料庫擴展讀取。

最佳化時間戳記操作

當系統變數 `time_zone` 的值設為 `SYSTEM` 時，每個需要時區計算的 MySQL 函數呼叫都會進行系統程式庫呼叫。當您執行以高並行方式傳回或變更這類 `TIMESTAMP` 值的 SQL 陳述式時，可能會遇到延遲、鎖定爭用和 CPU 使用率增加的情況。如需詳細資訊，請參閱 MySQL 文件中的 time_zone。

若要避免此行為，建議您將 `time_zone` 資料庫叢集參數的值變更為 `UTC`。如需詳細資訊，請參閱在 Amazon Aurora 中修改資料庫叢集參數群組中的參數。

儘管 `time_zone` 參數是動態的 (不需要重新啟動資料庫伺服器)，但新值僅用於新連線。若要確保所有連線都更新為使用新的 `time_zone` 值，建議您在更新資料庫叢集參數之後回收應用程式連線。

虛擬索引 ID 溢出錯誤

Aurora MySQL 會將虛擬索引 ID 的值限制為 8 位元，以防止 MySQL 中復原格式造成的問題。如果索引超過虛擬索引 ID 限制，您的叢集可能無法使用。當索引接近虛擬索引 ID 限制，或當您嘗試建立超過虛擬索引 ID 限制的索引時，RDS 可能會擲出錯誤代碼 63955 或警告代碼 63955。若要解決虛擬索引 ID 限制錯誤，建議您使用邏輯傾印和還原重新建立資料庫。

如需 Amazon Aurora MySQL 邏輯傾印和還原的詳細資訊，請參閱使用 MyDumper 和 MyLoader 將非常大型的資料庫遷移至 Amazon Aurora MySQL。如需在 Amazon Aurora 中存取錯誤日誌的詳細資訊，請參閱 監控 Amazon Aurora 日誌檔案。

Aurora MySQL 高可用性的最佳實務

您可以應用以下最佳實務，改善 Aurora MySQL 叢集的可用性。

主題

- 使用 Amazon Aurora 搭配 MySQL 資料庫進行災難復原
- 從 MySQL 遷移至 Amazon Aurora MySQL 時減少停機時間

- 避免 Aurora MySQL 資料庫執行個體效能變慢、自動重新啟動和容錯移轉

使用 Amazon Aurora 搭配 MySQL 資料庫進行災難復原

您可以使用 Amazon Aurora 搭配 MySQL 資料庫執行個體來建立用於災難復原的離站備份。若要使用 Aurora 進行 MySQL 資料庫執行個體的災難復原，請建立 Amazon Aurora 資料庫叢集，並讓它成為 MySQL 資料庫執行個體的讀取複本。這可套用至 RDS for MySQL 資料庫執行個體，或在 Amazon RDS 外部執行的 MySQL 資料庫。

> **Important**
>
> 設定 MySQL 資料庫執行個體與 Amazon Aurora MySQL 資料庫叢集之間的複寫時，您應該監控此複寫以確保其維持正常運作，並在必要時加以修復。

如需如何建立 Amazon Aurora MySQL 資料庫叢集，並讓它成為 MySQL 資料庫執行個體讀取複本的指示，請遵循使用 Amazon Aurora 為 MySQL 資料庫擴展讀取中的程序。

如需災難復原模型的詳細資訊，請參閱 How to choose the best disaster recovery option for your Amazon Aurora MySQL cluster (如何為您的 Amazon Aurora MySQL 叢集選擇最佳災難復原選項)。

從 MySQL 遷移至 Amazon Aurora MySQL 時減少停機時間

從支援線上應用程式的 MySQL 資料庫將資料匯入至 Amazon Aurora MySQL 資料庫叢集時，您可能希望能減少遷移時服務中斷的時間。為此，您可以使用《Amazon Relational Database Service 使用者指南》中減少將資料匯入 Amazon RDS for MySQL 資料庫執行個體時的停機時間所記載的程序。如果您使用的是超大型資料庫，這個程序特別有用。您可以使用此程序來透過降低經過網路傳輸至 AWS的資料量，藉此減少匯入的成本。

此程序列出將資料庫資料的複本傳輸至 Amazon EC2 執行個體，並將資料匯入新的 RDS for MySQL 資料庫執行個體的步驟。因為 Amazon Aurora 與 MySQL 相容，您可以改為對目標 Amazon RDS MySQL 資料庫執行個體使用 Amazon Aurora 資料庫叢集。

避免 Aurora MySQL 資料庫執行個體效能變慢、自動重新啟動和容錯移轉

如果您正在執行繁重的工作負載或工作負載超出資料庫執行個體配置的資源，則可能會在執行應用程式和 Aurora 資料庫時耗盡資源。若要取得資料庫執行個體的指標 (例如 CPU 使用率、記憶體使用量和使用的資料庫連線數目)，您可以參考 Amazon CloudWatch 提供的指標、Performance Insights 和增強型監控。如需如何監控資料庫執行個體的詳細資訊，請參閱 在 Amazon Aurora 叢集中監控指標。

如果您的工作負載耗盡了您正在使用的資源，您的資料庫執行個體可能會變慢、重新啟動，甚至容錯移轉到其他資料庫執行個體。若要避免這種情況，請監控資源使用率、檢查資料庫執行個體上執行的工作負載，並在必要時進行最佳化。如果最佳化無法改善執行個體指標並減緩資源耗盡，請考慮在達到其限制之前縱向擴展資料庫執行個體。如需可用資料庫執行個體類別及其規格的詳細資訊，請參閱 Amazon Aurora 資料庫執行個體類別。

Aurora MySQL 中 MySQL 功能的建議

以下功能可在與 MySQL 相容的 Aurora MySQL 中使用。但是，這些功能在 Aurora 環境中存在效能、可擴展性、穩定性或相容性的問題。因此，建議您在使用這些功能時遵循某些指導方針。例如，建議您不要將特定功能用於生產 Aurora 部署。

主題

- 在 Aurora MySQL 中使用多執行緒複寫
- 使用原生 MySQL AWS Lambda 函數叫用函數
- 避免搭配 Amazon Aurora MySQL 使用 XA 交易
- DML 陳述式期間保持外部索引鍵的開啟狀態
- 設定日誌緩衝區的排清頻率
- 減少和疑難排解 Aurora MySQL 的死結情況

在 Aurora MySQL 中使用多執行緒複寫

使用多執行緒二進位日誌複寫，SQL 執行緒會從轉送日誌讀取事件，並將它們排入佇列，以供 SQL 工作者執行緒套用。SQL 工作者執行緒是由協調器執行緒管理。可能的話，會平行套用二進位日誌事件。

Aurora MySQL 第 3 版和 Aurora MySQL 2.12.1 版及更新版本支援多執行緒複寫。

對於低於 3.04 版 Aurora MySQL，當您將 Aurora MySQL 資料庫叢集用來做為二進位日誌複寫的讀取複本時，Aurora 預設會使用單一執行緒複寫。

Aurora MySQL 第 2 版的早期版本已從 MySQL Community Edition 繼承數個有關多執行緒複寫的問題。對於這些版本，建議您不要在生產中使用多執行緒複寫。

如果您要使用多執行緒複寫，建議您徹底進行測試。

如需在 Amazon Aurora 中使用複寫的詳細資訊，請參閱以 Amazon Aurora 進行複寫。如需 Aurora MySQL 中多執行緒複寫的詳細資訊，請參閱 多執行緒二進位日誌複寫。

使用原生 MySQL AWS Lambda 函數叫用函數

建議您使用原生 MySQL 函數 `lambda_sync` 和 `lambda_async`，以叫用 Lambda 函數。

如果您使用已棄用的 `mysql.lambda_async` 程序，建議在預存程序中包裝對 `mysql.lambda_async` 程序的呼叫。您可以從不同的來源 (例如觸發程式或用戶端程式碼) 呼叫此預存程序。此方法有助於避免阻抗不符問題，並讓您的資料庫程式設計人員能夠輕鬆呼叫 Lambda 函數。

如需從 Amazon Aurora 叫用 Lambda 函數的詳細資訊，請參閱從 Amazon Aurora MySQL 資料庫叢集叫用 Lambda 函式。

避免搭配 Amazon Aurora MySQL 使用 XA 交易

建議您不要使用 eXtended Architecture (XA) 交易搭配 Aurora MySQL，因為如果 XA 處於 PREPARED 狀態，它們可能造成過長的復原時間。如果您必須使用 XA 交易搭配 Aurora MySQL，請遵循這些最佳實務：

- 請勿讓 XA 交易保持開啟在 PREPARED 狀態。
- 盡可能讓 XA 交易越小越好。

如需使用 XA 交易搭配 MySQL 的詳細資訊，請參閱 MySQL 文件中的 XA 交易。

DML 陳述式期間保持外部索引鍵的開啟狀態

在將 `foreign_key_checks` 變數設為 `0` (關閉) 時，強烈建議您不要執行任何資料定義語言 (DDL) 陳述式。

如果您需要插入或更新需要暫時違反外部索引鍵的資料列，請遵循這些步驟：

1. 將 `foreign_key_checks` 設定為 `0`。
2. 進行您的資料處理語言 (DML) 變更。
3. 確定您完成的變更未違反任何外部索引鍵限制。
4. 將 `foreign_key_checks` 設為 1 (開啟)。

此外，遵循用於外部索引鍵限制的這些其他最佳實務：

- 確定您的用戶端應用程式未隨著 `foreign_key_checks` 變數將 `0` 變數設為 `init_connect`。

- 如果從邏輯備份 (例如 mysqldump) 的還原失敗或未完成，在相同工作階段中開始任何其他操作之前，請確定將 foreign_key_checks 設為 1。邏輯備份會在開始時將 foreign_key_checks 設為 0。

設定日誌緩衝區的排清頻率

在 MySQL Community Edition 中，若要使交易可以耐久，InnoDB 日誌緩衝區必須排清到耐久儲存。您可以使用 innodb_flush_log_at_trx_commit 參數來設定日誌緩衝區排清到磁碟的頻率。

當您將 innodb_flush_log_at_trx_commit 參數設為預設值 1 時，日誌緩衝區會在每次交易認可時進行排清。此設定有助於保持資料庫 ACID 合規。建議您保留預設設定 1。

將 innodb_flush_log_at_trx_commit 變更為非預設值可協助減少資料操作語言 (DML) 延遲，但會犧牲日誌記錄的耐久性。一旦缺乏耐久性，就會使資料庫 ACID 不合規。建議您的資料庫必須是 ACID 合規，以避免在伺服器重新啟動時發生資料遺失的風險。如需此參數的詳細資訊，請參閱 MySQL 文件中的 innodb_flush_log_at_trx_commit。

在 Aurora MySQL 中，重做日誌處理會卸載至儲存層，因此資料庫執行個體上不會排清至日誌檔。發出寫入時，重做日誌會從寫入器資料庫執行個體直接傳送至 Aurora 叢集磁碟區。唯一跨網路的寫入是重做日誌記錄。始終不會從資料庫層寫入任何頁面。

根據預設，每個認可交易的執行緒都會等待 Aurora 叢集磁碟區的確認。此確認指出已寫入此記錄和所有先前的重做日誌記錄，都已寫入並達到仲裁。不論是透過自動認可還是明確認可，持續保留日誌記錄並達到仲裁，都能使交易耐久。如需 Aurora 儲存架構的詳細資訊，請參閱 Amazon Aurora 儲存解密。

Aurora MySQL 不會如 MySQL Community Edition 一樣將日誌排清到資料檔案。不過，在將重做日誌記錄寫入至 Aurora 叢集磁碟區時，您可以使用 innodb_flush_log_at_trx_commit 參數來放鬆耐久性限制。

適用於 Aurora MySQL 第 2 版：

- innodb_flush_log_at_trx_commit = 0 或 2 – 資料庫不會等待重做日誌記錄寫入 Aurora 叢集磁碟區的確認。
- innodb_flush_log_at_trx_commit = 1 – 資料庫會等待確認重做日誌記錄寫入 Aurora 叢集磁碟區。

適用於 Aurora MySQL 第 3 版：

- `innodb_flush_log_at_trx_commit` = 0 – 資料庫不會等待確認重做日誌記錄寫入 Aurora 叢集磁碟區。
- `innodb_flush_log_at_trx_commit` = 1 或 2 – 資料庫會等待確認重做日誌記錄寫入 Aurora 叢集磁碟區。

因此，若要取得 Aurora MySQL 第 3 版中與 Aurora MySQL 第 2 版中值設定為 0 或 2 相同的非預設行為，請將參數設定為 0。

雖然這些設定可以降低用戶端的 DML 延遲，但在發生容錯移轉或重新啟動時，也可能導致資料遺失。因此，建議您將 `innodb_flush_log_at_trx_commit` 參數持續設定為預設值 1。

雖然 MySQL Community Edition 和 Aurora MySQL 都可能發生資料遺失，但由於其架構不同，每個資料庫的行為都會有所不同。這些架構差異可能會導致不同程度的資料遺失。若要確保您的資料庫是 ACID 合規，請將 `innodb_flush_log_at_trx_commit` 一律設定為 1。

> **Note**
>
> 在 Aurora MySQL 第 3 版中，在您將 `innodb_flush_log_at_trx_commit` 變更為 1 以外的值之前，將您必須先將 `innodb_trx_commit_allow_data_loss` 的值變更為 1。如此一來，您就會確認資料遺失的風險。

減少和疑難排解 Aurora MySQL 的死結情況

在相同資料頁面上並行修改記錄時，執行在唯一次要索引或外部索引鍵上經常遇到限制條件違規之工作負載的使用者，可能會遇到更多的死結情況和鎖定等待逾時。這些死結情況和逾時皆源自 MySQL Community Edition 錯誤修正。

此修正包含在 MySQL Community Edition 5.7.26 及更新版本中，並已向後移植至 Aurora MySQL 2.10.3 及更新版本。強制序列化必須有此修正，方法是在 InnoDB 資料表的記錄變更上，針對這些資料處理語言 (DML) 類型執行額外鎖定。此問題是調查先前 MySQL Community Edition 錯誤修正導致的死結問題時發現的。

該項修正變更了 InnoDB 儲存引擎中，元組 (資料列) 更新的部分回復內部處理。在外部索引鍵或唯一次要索引產生限制條件違規的操作會導致部分回復。這包括但不限於並行 `INSERT...ON DUPLICATE KEY UPDATE`、`REPLACE INTO`, 和 `INSERT IGNORE` 陳述式 (upserts)。

在此情況下，部分回復並非應用程式層級交易的回復，而是發生限制條件違規時，對叢集索引的內部 InnoDB 回復變更。例如，在 upsert 操作期間找到重複索引鍵值。

在正常的插入操作中，InnoDB 會自動為每個索引建立叢集和次要索引項目。如果 InnoDB 在 upsert 操作期間偵測到唯一次要索引上的重複值，則叢集索引中插入的項目必須回復 (部分回復)，並將更新套用至現有的重複列。在此內部部分回復步驟中，InnoDB 必須在操作中鎖定每個記錄。此修正會在部分回復之後引入額外鎖定，以確保交易序列化。

減少 InnoDB 死結情況

您可以採取下列方法來減少資料庫執行個體的死結情況發生頻率。MySQL 文件中有更多範例。

1. 為了減少死結情況發生，請在進行一系列相關變更後立即遞交交易。您可以將大型交易 (遞交之間的多列更新) 拆解為較小交易來執行此動作。若您要批次插入行，請盡量減少批次插入的大小，尤其是使用上述提及的 upsert 操作時。

   若要減少部分回復的次數，您可以嘗試下列方法：

   a. 一次插入一行，而非進行批次插入操作。如此便能減少可能發生衝突之交易的鎖定時間。

   b. 請不要使用 `REPLACE INTO`，而是將 SQL 陳述式以多陳述式交易重新寫入，如下所示：

   ```
   BEGIN;
   DELETE conflicting rows;
   INSERT new rows;
   COMMIT;
   ```

   c. 請不要使用 `INSERT...ON DUPLICATE KEY UPDATE`，而是將 SQL 陳述式以多陳述式交易重新寫入，如下所示：

   ```
   BEGIN;
   SELECT rows that conflict on secondary indexes;
   UPDATE conflicting rows;
   INSERT new rows;
   COMMIT;
   ```

2. 避免長時間執行且可能導致鎖定的作用中或閒置交易。其中包括互動式 MySQL 用戶端工作階段，這些工作階段可能會在未遞交的交易中長時間開啟。最佳化交易大小或批次大小時，影響會因並行、重複項目數量和資料表結構等因素而有所不同。任何變更都應根據您的工作負載來執行和測試。

3. 在部分情況下，當兩筆交易嘗試以不同順序存取相同的資料集時，不論資料集是否位於同一個資料表，皆可能發生死結情況。為了防止這種情況，您可以修改交易以相同順序存取資料，進而序列化該存取。例如，建立待完成的交易佇列。當多個交易並行發生時，此方法便能協助避免死結情況。

4. 將謹慎選擇的索引新增到資料表中，可改善選取性並減少存取資料列的需求，從而減少死結情況。

5. 若發生間隙鎖定，您可以將工作階段或交易的交易隔離層級修改為 READ COMMITTED，以避免發生間隙鎖定。如需詳細了解 InnoDB 隔離層級和其行為，請參閱 MySQL 文件的交易隔離層級。

> **Note**
>
> 雖然您可以採取預防措施來減少死結情況，但死結情況是可預期的資料庫行為，仍然可能發生。應用程式應具有必要的邏輯來處理死結情況。例如，在應用程式中執行重試和停止邏輯。最理想的是解決問題根本原因，但是若確實發生死結情況，應用程式仍可以選擇等待並重試。

監控 InnoDB 死結情況

應用程式交易嘗試以導致循環等待的方式取得資料表層級和資料列層級鎖定時，MySQL 中可能發生死結。偶爾的 InnoDB 死結情況不一定是大問題，因為 InnoDB 儲存引擎會立即偵測到狀況，並自動復原交易。若您經常遇到死結情況，建議您檢閱和修改應用程式，以減緩效能問題並避免死結發生。當死結偵測開啟 (預設值) 時，InnoDB 會自動偵測交易死結，並復原交易以解決死結情況。InnoDB 嘗試選擇要復原的小型交易，而交易的大小是由插入、更新或刪除的資料列數而定。

- SHOW ENGINE 陳述式 – SHOW ENGINE INNODB STATUS \G 陳述式包含自上次重啟以來，資料庫最近遇到的死結詳細資訊。
- MySQL 錯誤日誌 – 若您頻繁遇到死結，而 SHOW ENGINE 陳述式的輸出不敷使用，您可以開啟 innodb_print_all_deadlocks 資料庫叢集參數。

  開啟此參數時，InnoDB 使用者交易中所有死結的相關資訊皆會記錄在 Aurora MySQL 錯誤日誌。
- Amazon CloudWatch 指標 – 我們也建議使用 CloudWatch 指標 Deadlocks 主動監控死結情況。如需詳細資訊，請參閱Amazon Aurora 的執行個體層級指標。
- Amazon CloudWatch Logs – 透過 CloudWatch Logs，您可以檢視指標、分析日誌資料，並建立即時警示。如需更多詳細資訊，請參閱使用 Amazon CloudWatch 監控 Amazon Aurora MySQL 和 Amazon RDS for MySQL 的錯誤，並透過 Amazon SNS 傳送通知。

  若在 innodb_print_all_deadlocks 開啟的情況下使用 CloudWatch Logs，您可以設定警示，以便在死結數量超過指定閾值時通知您。若要定義閾值，建議您觀察平時情況，根據您的正常工作負載來定義該值。
- 績效詳情 — 當您使用績效詳情時，您可以監控 innodb_deadlocks 和 innodb_lock_wait_timeout 指標。如需這些指標的詳細資訊，請參閱 Aurora MySQL 的非原生計數器。

使用 Amazon CloudWatch 指標評估 Aurora MySQL 的資料庫執行個體用量

您可以使用 CloudWatch 指標來監控資料庫執行個體輸送量，並判斷資料庫執行個體類別是否為您的應用程式提供足夠的資源。如需您資料庫執行個體類別限制的相關資訊，請參閱 Aurora 資料庫執行個體類別的硬體規格。尋找資料庫執行個體類別的規格，以尋找網路效能。

如果您的資料庫執行個體用量接近執行個體類別限制，效能可能會開始變慢。CloudWatch 指標可以確認這種情況，因此您可以規劃手動縱向擴展到更大的執行個體類別。

結合下列 CloudWatch 指標值，以了解您是否接近執行個體類別限制：

- NetworkThroughput - 用戶端針對 Aurora MySQL 資料庫叢集中各個執行個體所接收和傳輸的網路輸送量。此輸送量值不包含資料庫叢集中的執行個體與叢集磁碟區之間的網路流量。
- StorageNetworkThroughput - Aurora MySQL 資料庫叢集中各個執行個體接收自和傳送至 Aurora 儲存子系統的網路輸送量。

將 NetworkThroughput 新增至 StorageNetworkThroughput，以尋找 Aurora MySQL 資料庫叢集中各個執行個體接收自和傳送至 Aurora 儲存子系統的網路輸送量。執行個體的執行個體類別限制應該大於這兩個合併指標之和。

在傳送和接收時，您可以使用下列指標，來檢閱來自用戶端應用程式之網路流量的其他詳細資訊：

- NetworkReceiveThroughput - Aurora MySQL 資料庫叢集中的每個資料庫執行個體從用戶端接收到的網路輸送量。此輸送量不包含資料庫叢集中的執行個體與叢集磁碟區之間的網路流量。
- NetworkTransmitThroughput - Aurora 資料庫叢集中的各個執行個體傳送至用戶端的網路輸送量。此輸送量不包含資料庫叢集中的執行個體與叢集磁碟區之間的網路流量。
- StorageNetworkReceiveThroughput - 資料庫叢集中的各個執行個體從 Aurora 儲存子系統接收到的網路輸送量。
- StorageNetworkTransmitThroughput - 資料庫叢集中的各個執行個體傳送至 Aurora 儲存子系統的網路輸送量。

將所有這些指標加在一起，以評估您的網路用量與資料庫執行個體類別限制的比較情形。執行個體類別限制應該大於這些個合併指標之和。

儲存的網路限制和 CPU 用量是直接相關的。當網路輸送量增加時，CPU 用量也會增加。監控 CPU 和網路用量可提供資源如何和為何耗盡的相關資訊。

若要協助將網路用量降至最低，您可以考慮下列事項：

- 使用更大的資料庫執行個體類別。
- 批次分割寫入要求，以減少整體交易。
- 將唯讀工作負載導向至唯讀執行個體。
- 刪除任何未使用的索引。

對 Amazon Aurora MySQL 資料庫效能進行故障診斷

本主題著重於一些常見的 Aurora MySQL 資料庫效能問題，以及如何故障診斷或收集資訊以快速修復這些問題。我們將資料庫效能分為兩個類別：

- 伺服器效能 – 整個資料庫伺服器執行速度較慢。
- 查詢效能 – 執行一或多個查詢需要更長的時間。

AWS 監控選項

我們建議您使用下列 AWS 監控選項來協助進行故障診斷：

- Amazon CloudWatch - Amazon CloudWatch 會即時監控您的 AWS 資源，以及您在 AWS 上執行的應用程式。您可以使用 CloudWatch 收集和追蹤指標，這些是您可以為您的資源和應用程式測量的變數。如需詳細資訊，請參閱什麼是 Amazon CloudWatch？

  您可以在 AWS 管理主控台 上檢視資料庫執行個體的所有系統指標和處理程序資訊。您可以設定 Aurora MySQL 資料庫叢集將一般、緩慢、稽核和錯誤日誌資料發佈至 Amazon CloudWatch Logs 中的日誌群組。這可讓您檢視趨勢、在主機受到影響時維護日誌，以及建立「正常」效能的基準，以輕鬆識別異常或變更。如需更多詳細資訊，請參閱 將 Amazon Aurora MySQL 日誌發佈至 Amazon CloudWatch Logs。
- 增強型監控 – 若要啟用 Aurora MySQL 資料庫的其他 Amazon CloudWatch 指標，請開啟增強型監控。當您建立或修改 Aurora 資料庫叢集時，請選取啟用增強型監控。這可讓 Aurora 將效能指標發佈至 CloudWatch。一些可用的關鍵指標包括 CPU 用量、資料庫連線、儲存空間用量和查詢延遲。這些指標可協助判斷效能瓶頸。

  針對資料庫執行個體傳輸的資訊量與針對「增強型監控」定義的精密度成正比。較短的監控時間間隔會導致較頻繁的作業系統指標報告，並增加您的監控成本。若要管理成本，請針對 AWS 帳戶 中的不同執行個體設定不同的精細程度。建立執行個體時的預設精細程度為 60 秒。如需更多詳細資訊，請參閱 增強型監控的成本。
- 績效詳情 – 您可以檢視所有資料庫呼叫指標。這包括資料庫鎖定、等待和已處理的列數，所有這些都可以用於故障診斷。當您建立或修改 Aurora 資料庫叢集時，請選取開啟績效詳情。根據預設，績效詳情具有 7 天的資料保留期，但可以自訂以分析長期的績效趨勢。對於超過 7 天的保留期，您需要升級至付費方案。如需詳細資訊，請參閱績效詳情定價。您可以分別設定每個 Aurora 資料庫執行個體的資料保留期間。如需更多詳細資訊，請參閱 在 Amazon Aurora 上使用績效詳情監控資料庫負載。

Aurora MySQL 資料庫效能問題的最常見原因

您可以使用下列步驟來故障診斷 Aurora MySQL 資料庫中的效能問題。我們會以調查的邏輯順序列出這些步驟，但它們不是線性的。一個探索可能會跨步驟跳躍，允許一系列的調查路徑。

1. 工作負載 – 了解您的資料庫工作負載。
2. 記錄 – 檢閱所有資料庫日誌。
3. 資料庫連線 – 請確定您的應用程式與資料庫之間的連線是可靠的。
4. 查詢效能 – 檢查您的查詢執行計劃，以查看它們是否已變更。程式碼變更可能會導致計劃變更。

針對 Aurora MySQL 資料庫的工作負載問題進行故障診斷

資料庫工作負載可以檢視為讀取和寫入。了解「正常」資料庫工作負載後，您可以調校查詢和資料庫伺服器，以滿足其變更的需求。效能可以變更的原因有很多種，因此第一步是了解變更的內容。

- 是否有主要或次要版本升級？

  主要版本升級包括引擎程式碼的變更，特別是在可變更查詢執行計畫的最佳化工具中。升級資料庫版本時，特別是主要版本，分析資料庫工作負載並進行相應調校非常重要。調校可能涉及最佳化和重寫查詢，或根據測試結果新增和更新參數設定。了解造成影響的原因，可讓您開始專注於該特定區域。

  如需詳細資訊，請參閱 MySQL 文件中的 MySQL 8.0 中的新功能和伺服器和狀態變數，以及 MySQL 8.0 中新增、棄用或移除的選項，以及 比較 Aurora MySQL 第 2 版和 Aurora MySQL 第 3 版。
- 正在處理的資料是否增加 (列計數)？
- 是否有更多同時執行的查詢？
- 是否有結構描述或資料庫變更？
- 是否有程式碼瑕疵或修正？

內容

- 執行個體主機指標
  - CPU 用量
  - 記憶體用量
  - 網路輸送量
- 資料庫指標
- 針對 Aurora MySQL 資料庫的記憶體用量問題進行故障診斷
  - 範例 1：持續高記憶體用量
  - 範例 2：暫時性記憶體峰值
  - 範例 3：可釋放的記憶體持續下降，且不會回收
- 故障診斷 Aurora MySQL 資料庫記憶體不足的問題

執行個體主機指標

監控執行個體主機指標，例如 CPU、記憶體和網路活動，以協助了解工作負載是否已變更。了解工作負載變更有兩個主要概念：

- 使用率 – 裝置的使用率，例如 CPU 或磁碟。它可以是以時間為基礎或以容量為基礎。
  - 時間型 – 資源在特定觀察期間忙碌的時間量。
  - 容量型 – 系統或元件可交付的輸送量，以容量的百分比表示。
- 飽和度 – 資源需要的工作量超過其可處理的程度。當容量型用量達到 100% 時，便無法處理額外工作，且必須排入佇列。

CPU 用量

您可以使用下列工具來識別 CPU 用量和飽和度：

- CloudWatch 提供 `CPUUtilization` 指標。如果達到 100%，則執行個體會飽和。不過，CloudWatch 指標的平均值為 1 分鐘，且缺乏精細程度。

  如需 CloudWatch 指標的相關資訊，請參閱 Amazon Aurora 的執行個體層級指標。
- 增強型監控提供作業系統 `top` 命令傳回的指標。它會顯示負載平均值和下列 CPU 狀態，精細程度為 1 秒：
  - `Idle (%)` = 閒置時間
  - `IRQ (%)` = 軟體中斷
  - `Nice (%)` = 具有良好狀態優先順序之程序的良好時間。
  - `Steal (%)` = 服務其他租用戶所花費的時間 (虛擬化相關)
  - `System (%)` = 系統時間
  - `User (%)` = 使用者時間
  - `Wait (%)` = I/O 等待

  如需增強式監控指標的詳細資訊，請參閱 Aurora 的作業系統指標。

記憶體用量

如果系統處於記憶體壓力下，且資源消耗達到飽和，您應該會觀察高度的頁面掃描、分頁、交換和記憶體不足錯誤。

您可以使用下列工具來識別記憶體用量和飽和度：

CloudWatch 提供 FreeableMemory 指標，顯示透過清除一些作業系統快取和目前的可用記憶體，可以回收多少記憶體。

如需 CloudWatch 指標的相關資訊，請參閱 Amazon Aurora 的執行個體層級指標。

增強型監控提供下列指標，可協助您識別記憶體用量問題：

- Buffers (KB) – 在寫入至儲存裝置之前，用於緩衝 I/O 請求的記憶體數量，以 KB 為單位。
- Cached (KB) – 用來快取檔案系統 I/O 的記憶體數量。
- Free (KB) – 未指派的記憶體數量，以 KB 為單位。
- Swap – 快取、免費和總計。

例如，如果您看到資料庫執行個體使用 Swap 記憶體，則工作負載的記憶體總量會大於執行個體目前可用的記憶體總量。我們建議您增加資料庫執行個體的大小，或調校工作負載以使用較少的記憶體。

如需增強式監控指標的詳細資訊，請參閱 Aurora 的作業系統指標。

如需使用效能結構描述和 sys 結構描述來判斷哪些連線和元件正在使用記憶體的詳細資訊，請參閱 針對 Aurora MySQL 資料庫的記憶體用量問題進行故障診斷。

網路輸送量

CloudWatch 為總網路輸送量提供下列指標，所有指標均為 1 分鐘內平均值：

- NetworkReceiveThroughput – Aurora 資料庫叢集中的各個執行個體從用戶端接收到的網路輸送量。
- NetworkTransmitThroughput – Aurora 資料庫叢集中的各個執行個體傳送至用戶端的網路輸送量。
- NetworkThroughput – Aurora 資料庫叢集中的各個執行個體從用戶端接收及傳送至用戶端的網路輸送量。
- StorageNetworkReceiveThroughput – 資料庫叢集中的各個執行個體從 Aurora 儲存子系統接收到的網路輸送量。
- StorageNetworkTransmitThroughput – Aurora 資料庫叢集中的各個執行個體傳送至 Aurora 儲存子系統的網路輸送量。
- StorageNetworkThroughput – Aurora 資料庫叢集中的各個執行個體從 Aurora 儲存子系統接收和向其傳入的網路輸送量。

如需 CloudWatch 指標的相關資訊，請參閱 Amazon Aurora 的執行個體層級指標。

增強型監控提供 network 的已接收 (RX) 和已傳輸 (TX) 圖形，精細程度高達 1 秒。

如需增強式監控指標的詳細資訊，請參閱 Aurora 的作業系統指標。

資料庫指標

檢查下列 CloudWatch 指標以確認是否有工作負載變更：

- `BlockedTransactions` – 被封鎖的資料庫中的平均每秒交易數量。
- `BufferCacheHitRatio` – 由緩衝區快取提供服務的請求的百分比。
- `CommitThroughput` – 遞交操作的每秒平均數量。
- `DatabaseConnections` – 連線至資料庫執行個體的用戶端網路連線數。
- `Deadlocks` – 資料庫中平均每秒的死鎖數量。
- `DMLThroughput` – 平均每秒插入、更新及刪除的數量。
- `ResultSetCacheHitRatio` – 由查詢快取提供服務的請求的百分比。
- `RollbackSegmentHistoryListLength` – 記錄具有刪除標記記錄之已遞交交易的復原日誌。
- `RowLockTime` – 針對 InnoDB 資料表取得資料列鎖定所花費的時間總計。
- `SelectThroughput` – 平均每秒選取查詢的數量。

如需 CloudWatch 指標的相關資訊，請參閱 Amazon Aurora 的執行個體層級指標。

檢查工作負載時，請考慮下列問題：

1. 資料庫執行個體類別最近是否有變更，例如將執行個體大小從 8xlarge 減少為 4xlarge，或從 db.r5 變更為 db.r6？
2. 您可以建立複製並重現問題，還是僅在該單一執行個體上發生？
3. 是否有伺服器資源耗盡、高 CPU 或記憶體耗盡？ 如果是，這可能表示需要額外的硬體。
4. 一個或多個查詢需要更長的時間嗎？
5. 變更是否由升級造成，尤其是主要版本升級？ 如果是，請比較升級前和升級後指標。
6. 讀取器資料庫執行個體的數量是否有變更？
7. 您是否已啟用一般、稽核或二進位記錄？ 如需更多詳細資訊，請參閱 記錄 Aurora MySQL 資料庫。
8. 您是否啟用、停用或變更二進位記錄 (binlog) 複寫的使用？

9. 是否有任何長時間執行的交易具有大量資料列鎖定？ 檢查 InnoDB 歷史記錄清單長度 (HLL) 是否有長時間執行交易的指示。

   如需詳細資訊，請參閱 InnoDB 歷史記錄清單長度顯著增加 和部落格文章為什麼我的 SELECT 查詢在 Amazon Aurora MySQL 資料庫叢集上緩慢執行？。

   a. 如果大型 HLL 是由寫入交易造成，表示 UNDO 日誌正在累積 (未定期清理)。在大型寫入交易中，此累積可能會快速成長。在 MySQL 中，UNDO 存放在 SYSTEM 資料表空間中。SYSTEM 資料表空間無法縮減。UNDO 日誌可能會導致 SYSTEM 資料表空間成長到數個 GB 甚至 TB。清除後，透過建立資料的邏輯備份 (傾印) 釋放配置的空間，然後將傾印匯入新的資料庫執行個體。

   b. 如果大型 HLL 是由讀取交易 (長時間執行的查詢) 所造成，可能表示查詢使用大量的暫時空間。重新啟動以釋放暫時空間。檢查 Performance Insights 資料庫指標是否有 Temp 區段中的任何變更，例如 `created_tmp_tables`。如需更多詳細資訊，請參閱 在 Amazon Aurora 上使用績效詳情監控資料庫負載。

10 您可以將長時間執行的交易分割成修改較少資料列的較小交易嗎？

11 封鎖的交易是否有任何變更或死結增加？ 用來檢查 `Locks` 區段中狀態變數的任何變更的 Performance Insights 資料庫指標，例如 `innodb_row_lock_time`、`innodb_row_lock_waits` 和 `innodb_dead_locks`。使用 1 分鐘或 5 分鐘的間隔。

12 是否有增加的等待事件？ 使用 1 分鐘或 5 分鐘的間隔檢查 Performance Insights 等待事件和等待類型。分析最久等待事件，並查看它們是否與工作負載變更或資料庫爭用相關聯。例如，`buf_pool mutex` 表示緩衝集區爭用。如需更多詳細資訊，請參閱 使用等待事件調校 Aurora MySQL。

針對 Aurora MySQL 資料庫的記憶體用量問題進行故障診斷

雖然 CloudWatch、增強型監控和 Performance Insights 在作業系統層級提供記憶體用量的良好概觀，例如資料庫程序使用的記憶體用量，但它們不允許您細分引擎內可能導致此記憶體用量的連線或元件。

若要對此進行故障診斷，您可以使用效能結構描述和 `sys` 結構描述。在 Aurora MySQL 第 3 版中，啟用效能結構描述時，預設會啟用記憶體檢測。在 Aurora MySQL 第 2 版中，預設只會啟用效能結構描述記憶體用量的記憶體檢測。如需效能結構描述中可用資料表的相關資訊，以追蹤記憶體用量並啟用效能結構描述記憶體檢測，請參閱 MySQL 文件中的記憶體摘要資料表。如需使用效能結構描述搭配 Performance Insights 的詳細資訊，請參閱 Aurora MySQL 上 Performance Insights 的效能結構描述概觀。

雖然效能結構描述中提供了詳細資訊來追蹤目前的記憶體用量，但 MySQL sys 結構描述具有效能結構描述資料表的檢視，可讓您快速找出記憶體的使用位置。

在 `sys` 結構描述中，下列檢視可用於依連線、元件和查詢來追蹤記憶體用量。

檢視	描述
memory_by_host_by_current_bytes	提供主機引擎記憶體用量的相關資訊。這有助於識別哪些應用程式伺服器或用戶端主機正在耗用記憶體。
memory_by_thread_by_current_bytes	依執行緒 ID 提供引擎記憶體用量的相關資訊。MySQL 中的執行緒 ID 可以是用戶端連線或背景執行緒。您可以使用 sys.processlist 檢視或 performance_schema.threads 資料表，將執行 ID 映射至 MySQL 連線 ID。
memory_by_user_by_current_bytes	依使用者提供引擎記憶體用量的相關資訊。這有助於識別哪些使用者帳戶或用戶端正在耗用記憶體。
memory_global_by_current_bytes	依引擎元件提供引擎記憶體用量的相關資訊。這有助於依引擎緩衝區或元件全域識別記憶體用量。例如，您可能會看到 InnoDB 緩衝集區的 memory/innodb/buf_buf_pool 事件，或預備陳述式的 memory/sql/Prepared_statement::main_mem_root 事件。
memory_global_total	提供資料庫引擎中總追蹤記憶體用量的概觀。

在 Aurora MySQL 3.05 版及更高版本中，您也可以在效能結構描述陳述式摘要表格中，依陳述式摘要追蹤記憶體用量上限。陳述式摘要資料表包含標準化陳述式摘要及其執行的彙總統計資料。MAX_TOTAL_MEMORY 欄可協助您識別自上次重設統計資料後，或資料庫執行個體重新啟動後，查詢摘要所使用的記憶體上限。這有助於識別可能耗用大量記憶體的特定查詢。

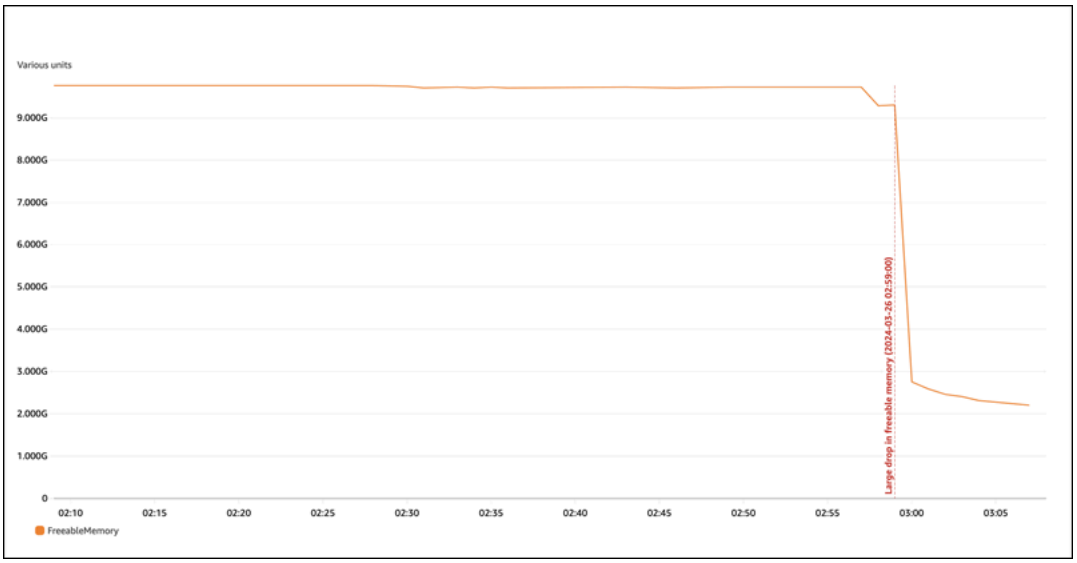
> **Note**
>
> 效能結構描述和 sys 結構描述會顯示伺服器上目前的記憶體使用量，以及每個連線和引擎元件所耗用記憶體的高水位標記。由於效能結構描述會保留在記憶體中，因此資料庫執行個體重新啟動時會重設資訊。若要隨時間維護歷程記錄，建議您在效能結構描述之外設定此資料的擷取和儲存。

主題

- 範例 1：持續高記憶體用量
- 範例 2：暫時性記憶體峰值
- 範例 3：可釋放的記憶體持續下降，且不會回收

範例 1：持續高記憶體用量

在 CloudWatch 中全域查看 `FreeableMemory`，我們可以看到記憶體用量在 2024-03-26 02:59 UTC 大幅增加。



這不會告訴我們整體情況。若要判斷哪個元件使用最多記憶體，您可以登入資料庫並查看 `sys.memory_global_by_current_bytes`。此資料表包含 MySQL 追蹤的記憶體事件清單，以及每個事件的記憶體配置資訊。每個記憶體追蹤事件都以 `memory/%` 開頭，後面接著與事件相關聯之引擎元件/功能的其他資訊。

例如，`memory/performance_schema/%` 適用於與效能結構描述相關的記憶體事件，而 `memory/innodb/%` 適用於 InnoDB，以此類推。如需關於事件命名慣例的詳細資訊，請參閱 MySQL 文件中的效能結構描述檢測命名慣例。

從下列查詢中，我們可以根據 `current_alloc` 找出可能的問題點，但我們也可以看到許多 `memory/performance_schema/%` 事件。

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes LIMIT 10;
```

```
+-----------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| event_name                                                                  |
 current_count | current_alloc | current_avg_alloc | high_count | high_alloc |
 high_avg_alloc |
+-----------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| memory/sql/Prepared_statement::main_mem_root                                |
 512817 | 4.91 GiB      | 10.04 KiB         |     512823 | 4.91 GiB   | 10.04 KiB
 |
| memory/performance_schema/prepared_statements_instances                     |
   252 | 488.25 MiB    | 1.94 MiB          |        252 | 488.25 MiB | 1.94 MiB       |
| memory/innodb/hash0hash                                                     |
     4 | 79.07 MiB     | 19.77 MiB         |          4 | 79.07 MiB  | 19.77 MiB      |
| memory/performance_schema/events_errors_summary_by_thread_by_error          |
  1028 | 52.27 MiB     | 52.06 KiB         |       1028 | 52.27 MiB  | 52.06 KiB      |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name |
     4 | 47.25 MiB     | 11.81 MiB         |          4 | 47.25 MiB  | 11.81 MiB      |
| memory/performance_schema/events_statements_summary_by_digest               |
     1 | 40.28 MiB     | 40.28 MiB         |          1 | 40.28 MiB  | 40.28 MiB      |
| memory/performance_schema/memory_summary_by_thread_by_event_name            |
     4 | 31.64 MiB     | 7.91 MiB          |          4 | 31.64 MiB  | 7.91 MiB       |
| memory/innodb/memory                                                        |
 15227 | 27.44 MiB     | 1.85 KiB          |      20619 | 33.33 MiB  | 1.66 KiB       |
| memory/sql/String::value                                                    |
 74411 | 21.85 MiB     |  307 bytes        |      76867 | 25.54 MiB  |  348 bytes     |
| memory/sql/TABLE                                                            |
  8381 | 21.03 MiB     | 2.57 KiB          |       8381 | 21.03 MiB  | 2.57 KiB       |
+-----------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
10 rows in set (0.02 sec)
```

我們先前提到效能結構描述存放在記憶體中，這表示它也會在 performance_schema 記憶體檢測中受到追蹤。

> **Note**
>
> 如果您發現效能結構描述正在使用大量記憶體，並且想要限制其記憶體用量，您可以根據您的需求調整資料庫參數。如需詳細資訊，請參閱 MySQL 文件中的效能結構描述記憶體配置模型。

為了方便閱讀，您可以重新執行相同的查詢，但排除效能結構描述事件。輸出會顯示下列內容：

- 主要記憶體取用者為 memory/sql/Prepared_statement::main_mem_root。
- current_alloc 欄會通知我們，MySQL 目前已配置 4.91 GiB 給此事件。
- high_alloc column 告訴我們 4.91 GiB 是自上次重設統計資料後或伺服器重新啟動後 current_alloc 的高水位標記。這表示 memory/sql/Prepared_statement::main_mem_root 處於其最高值。

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name NOT LIKE
 'memory/performance_schema/%' LIMIT 10;

+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
| memory/sql/Prepared_statement::main_mem_root  |        512817 | 4.91 GiB      | 10.04
 KiB         |     512823 | 4.91 GiB   | 10.04 KiB      |
| memory/innodb/hash0hash                       |             4 | 79.07 MiB     | 19.77
 MiB         |          4 | 79.07 MiB  | 19.77 MiB      |
| memory/innodb/memory                          |         17096 | 31.68 MiB     | 1.90
 KiB         |      22498 | 37.60 MiB  | 1.71 KiB       |
| memory/sql/String::value                      |        122277 | 27.94 MiB     |  239
 bytes       |     124699 | 29.47 MiB  |  247 bytes     |
| memory/sql/TABLE                              |          9927 | 24.67 MiB     | 2.55
 KiB         |       9929 | 24.68 MiB  | 2.55 KiB       |
| memory/innodb/lock0lock                       |          8888 | 19.71 MiB     | 2.27
 KiB         |       8888 | 19.71 MiB  | 2.27 KiB       |
| memory/sql/Prepared_statement::infrastructure |        257623 | 16.24 MiB     |   66
 bytes       |     257631 | 16.24 MiB  |   66 bytes     |
| memory/mysys/KEY_CACHE                        |             3 | 16.00 MiB     | 5.33
 MiB         |          3 | 16.00 MiB  | 5.33 MiB       |
```

```
| memory/innodb/sync0arr                          |             3 | 7.03 MiB      | 2.34
 MiB          |          3 | 7.03 MiB   | 2.34 MiB       |
| memory/sql/THD::main_mem_root                   |           815 | 6.56 MiB      | 8.24
 KiB          |        849 | 7.19 MiB   | 8.67 KiB       |
+-------------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
10 rows in set (0.06 sec)
```

從事件的名稱，我們可以指出此記憶體正用於預備陳述式。如果您想要查看哪些連線使用此記憶體，您可以檢查 memory_by_thread_by_current_bytes。

在下列範例中，每個連線配置了大約 7 MiB，高水位標記約為 6.29 MiB (current_max_alloc)。這很合理，因為範例使用 sysbench 搭配 80 個資料表和 800 個連線以及預備陳述式。如果您想要在此案例中減少記憶體使用量，您可以最佳化應用程式對預備陳述式的使用量，以減少記憶體使用量。

```
mysql> SELECT * FROM sys.memory_by_thread_by_current_bytes;

+-----------+-------------------------------------------+--------------------
+------------------+------------------+------------------+-----------------+
| thread_id | user                                      | current_count_used |
 current_allocated | current_avg_alloc | current_max_alloc | total_allocated |
+-----------+-------------------------------------------+--------------------
+------------------+------------------+------------------+-----------------+
|        46 | rdsadmin@localhost                        |                405 | 8.47 MiB
          | 21.42 KiB         | 8.00 MiB          | 155.86 MiB      |
|        61 | reinvent@10.0.4.4                         |               1749 | 6.72 MiB
          | 3.93 KiB          | 6.29 MiB          | 14.24 MiB       |
|       101 | reinvent@10.0.4.4                         |               1845 | 6.71 MiB
          | 3.72 KiB          | 6.29 MiB          | 14.50 MiB       |
|        55 | reinvent@10.0.4.4                         |               1674 | 6.68 MiB
          | 4.09 KiB          | 6.29 MiB          | 14.13 MiB       |
|        57 | reinvent@10.0.4.4                         |               1416 | 6.66 MiB
          | 4.82 KiB          | 6.29 MiB          | 13.52 MiB       |
|       112 | reinvent@10.0.4.4                         |               1759 | 6.66 MiB
          | 3.88 KiB          | 6.29 MiB          | 14.17 MiB       |
|        66 | reinvent@10.0.4.4                         |               1428 | 6.64 MiB
          | 4.76 KiB          | 6.29 MiB          | 13.47 MiB       |
|        75 | reinvent@10.0.4.4                         |               1389 | 6.62 MiB
          | 4.88 KiB          | 6.29 MiB          | 13.40 MiB       |
|       116 | reinvent@10.0.4.4                         |               1333 | 6.61 MiB
          | 5.08 KiB          | 6.29 MiB          | 13.21 MiB       |
|        90 | reinvent@10.0.4.4                         |               1448 | 6.59 MiB
          | 4.66 KiB          | 6.29 MiB          | 13.58 MiB       |
```

```
|        98 | reinvent@10.0.4.4                          |              1440 | 6.57 MiB
         | 4.67 KiB          | 6.29 MiB          | 13.52 MiB       |
|        94 | reinvent@10.0.4.4                          |              1433 | 6.57 MiB
         | 4.69 KiB          | 6.29 MiB          | 13.49 MiB       |
|        62 | reinvent@10.0.4.4                          |              1323 | 6.55 MiB
         | 5.07 KiB          | 6.29 MiB          | 13.48 MiB       |
|        87 | reinvent@10.0.4.4                          |              1323 | 6.55 MiB
         | 5.07 KiB          | 6.29 MiB          | 13.25 MiB       |
|        99 | reinvent@10.0.4.4                          |              1346 | 6.54 MiB
         | 4.98 KiB          | 6.29 MiB          | 13.24 MiB       |
|       105 | reinvent@10.0.4.4                          |              1347 | 6.54 MiB
         | 4.97 KiB          | 6.29 MiB          | 13.34 MiB       |
|        73 | reinvent@10.0.4.4                          |              1335 | 6.54 MiB
         | 5.02 KiB          | 6.29 MiB          | 13.23 MiB       |
|        54 | reinvent@10.0.4.4                          |              1510 | 6.53 MiB
         | 4.43 KiB          | 6.29 MiB          | 13.49 MiB       |
.                                                                                .
.                                                                                .
.                                                                                .
|       812 | reinvent@10.0.4.4                          |              1259 | 6.38 MiB
         | 5.19 KiB          | 6.29 MiB          | 13.05 MiB       |
|       214 | reinvent@10.0.4.4                          |              1279 | 6.38 MiB
         | 5.10 KiB          | 6.29 MiB          | 12.90 MiB       |
|       325 | reinvent@10.0.4.4                          |              1254 | 6.38 MiB
         | 5.21 KiB          | 6.29 MiB          | 12.99 MiB       |
|       705 | reinvent@10.0.4.4                          |              1273 | 6.37 MiB
         | 5.13 KiB          | 6.29 MiB          | 13.03 MiB       |
|       530 | reinvent@10.0.4.4                          |              1268 | 6.37 MiB
         | 5.15 KiB          | 6.29 MiB          | 12.92 MiB       |
|       307 | reinvent@10.0.4.4                          |              1263 | 6.37 MiB
         | 5.17 KiB          | 6.29 MiB          | 12.87 MiB       |
|       738 | reinvent@10.0.4.4                          |              1260 | 6.37 MiB
         | 5.18 KiB          | 6.29 MiB          | 13.00 MiB       |
|       819 | reinvent@10.0.4.4                          |              1252 | 6.37 MiB
         | 5.21 KiB          | 6.29 MiB          | 13.01 MiB       |
|        31 | innodb/srv_purge_thread                    |             17810 | 3.14 MiB
         |  184 bytes        | 2.40 MiB          | 205.69 MiB      |
|        38 | rdsadmin@localhost                         |               599 | 1.76 MiB
         | 3.01 KiB          | 1.00 MiB          | 25.58 MiB       |
|         1 | sql/main                                   |              3756 | 1.32 MiB
         |  367 bytes        | 355.78 KiB        | 6.19 MiB        |
```

```
|       854 | rdsadmin@localhost                          |                 46 | 1.08 MiB
          | 23.98 KiB         | 1.00 MiB          | 5.10 MiB        |
|        30 | innodb/clone_gtid_thread                    |               1596 | 573.14
 KiB        |  367 bytes        | 254.91 KiB        | 970.69 KiB      |
|        40 | rdsadmin@localhost                          |                235 | 245.19
 KiB        | 1.04 KiB          | 128.88 KiB        | 808.64 KiB      |
|       853 | rdsadmin@localhost                          |                 96 | 94.63
 KiB         | 1009 bytes        | 29.73 KiB         | 422.45 KiB      |
|        36 | rdsadmin@localhost                          |                 33 | 36.29
 KiB         | 1.10 KiB          | 16.08 KiB         | 74.15 MiB       |
|        33 | sql/event_scheduler                         |                  3 | 16.27
 KiB         | 5.42 KiB          | 16.04 KiB         | 16.27 KiB       |
|        35 | sql/compress_gtid_table                     |                  8 | 14.20
 KiB         | 1.77 KiB          | 8.05 KiB          | 18.62 KiB       |
|        25 | innodb/fts_optimize_thread                  |                 12 | 1.86 KiB
          |  158 bytes        |  648 bytes        | 1.98 KiB        |
|        23 | innodb/srv_master_thread                    |                 11 | 1.23 KiB
          |  114 bytes        |  361 bytes        | 24.40 KiB       |
|        24 | innodb/dict_stats_thread                    |                 11 | 1.23 KiB
          |  114 bytes        |  361 bytes        | 1.35 KiB        |
|         5 | innodb/io_read_thread                       |                  1 |  144
 bytes        |  144 bytes        |  144 bytes        |  144 bytes      |
|         6 | innodb/io_read_thread                       |                  1 |  144
 bytes        |  144 bytes        |  144 bytes        |  144 bytes      |
|         2 | sql/aws_oscar_log_level_monitor             |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|         4 | innodb/io_ibuf_thread                       |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|         7 | innodb/io_write_thread                      |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|         8 | innodb/io_write_thread                      |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|         9 | innodb/io_write_thread                      |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        10 | innodb/io_write_thread                      |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        11 | innodb/srv_lra_thread                       |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        12 | innodb/srv_akp_thread                       |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        18 | innodb/srv_lock_timeout_thread              |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |  248 bytes      |
|        19 | innodb/srv_error_monitor_thread             |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
```

```
|        20 | innodb/srv_monitor_thread                 |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        21 | innodb/buf_resize_thread                  |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        22 | innodb/btr_search_sys_toggle_thread       |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        32 | innodb/dict_persist_metadata_table_thread |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        34 | sql/signal_handler                        |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
+-----------+-------------------------------------------+-------------------
+-------------------+------------------+-------------------+-----------------+
831 rows in set (2.48 sec)
```

如前所述，此處的執行緒 ID (thd_id) 值可以參考伺服器背景執行緒或資料庫連線。如果您想要將執行緒 ID 值映射至資料庫連線 ID，您可以使用 performance_schema.threads 資料表或 sys.processlist 檢視，其中 conn_id 是連線 ID。

```
mysql> SELECT thd_id,conn_id,user,db,command,state,time,last_wait FROM sys.processlist
 WHERE user='reinvent@10.0.4.4';

+--------+---------+------------------+----------+---------+---------------+------
+-------------------------------------------------+
| thd_id | conn_id | user             | db       | command | state         | time |
 last_wait                                        |
+--------+---------+------------------+----------+---------+---------------+------
+-------------------------------------------------+
|    590 |     562 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                           |
|    578 |     550 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                             |
|    579 |     551 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                           |
|    580 |     552 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                        |
|    581 |     553 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                        |
|    582 |     554 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                             |
|    583 |     555 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                             |
|    584 |     556 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                        |
```

```
|    585 |     557 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
|    586 |     558 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                       |
|    587 |     559 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
.
                                                 .
.
                                                 .
.
                                                 .
|    323 |     295 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                            |
|    324 |     296 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                       |
|    325 |     297 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
|    326 |     298 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                       |
|    438 |     410 | reinvent@10.0.4.4 | sysbench | Execute | System lock    |    0 |
 wait/lock/table/sql/handler                     |
|    280 |     252 | reinvent@10.0.4.4 | sysbench | Sleep   | starting       |    0 |
 wait/io/socket/sql/client_connection            |
|     98 |      70 | reinvent@10.0.4.4 | sysbench | Query   | freeing items  |    0 |
 NULL                                            |
+--------+---------+-------------------+----------+---------+----------------+------
+-------------------------------------------------+
804 rows in set (5.51 sec)
```

現在，我們停止 sysbench 工作負載，這會關閉連線並釋放記憶體。再次檢查事件，我們可以確認記憶體已釋出，但 high_alloc 仍告訴我們什麼是高水位標記。high_alloc 欄在識別記憶體用量中的短峰值時非常有用，其中您可能無法立即識別來自 current_alloc 的用量，這只會顯示目前配置的記憶體。

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name='memory/sql/
Prepared_statement::main_mem_root' LIMIT 10;

+-----------------------------------------------+---------------+---------------
+-------------------+------------+------------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
```

```
+-----------------------------------------------+--------------+--------------
+------------------+-----------+-----------+----------------+
| memory/sql/Prepared_statement::main_mem_root |            17 | 253.80 KiB    | 14.93
 KiB         |     512823 | 4.91 GiB   | 10.04 KiB      |
+-----------------------------------------------+--------------+--------------
+------------------+-----------+-----------+----------------+
1 row in set (0.00 sec)
```

如果您想要重設 high_alloc，您可以截斷 performance_schema 記憶體摘要資料表，但這會重設所有記憶體檢測。如需詳細資訊，請參閱 MySQL 文件中的效能結構描述一般資料表特性。

在下列範例中，我們可以看到 high_alloc 會在截斷後重設。

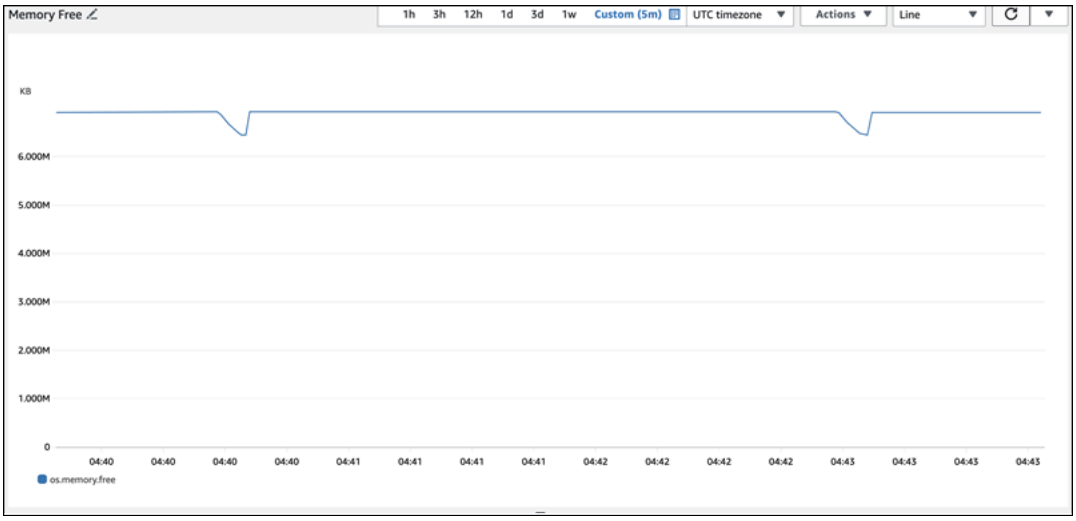
```
mysql> TRUNCATE `performance_schema`.`memory_summary_global_by_event_name`;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name='memory/sql/
Prepared_statement::main_mem_root' LIMIT 10;

+-----------------------------------------------+--------------+--------------
+------------------+-----------+-----------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+-----------------------------------------------+--------------+--------------
+------------------+-----------+-----------+----------------+
| memory/sql/Prepared_statement::main_mem_root |            17 | 253.80 KiB    | 14.93
 KiB         |         17 | 253.80 KiB | 14.93 KiB      |
+-----------------------------------------------+--------------+--------------
+------------------+-----------+-----------+----------------+
1 row in set (0.00 sec)
```

範例 2：暫時性記憶體峰值

另一個常見情況是資料庫伺服器上記憶體用量的短暫峰值。這些可能是在 sys.memory_global_by_current_bytes 中使用 current_alloc 時難以故障診斷的可用記憶體中定期下降，因為記憶體已釋放。

> **Note**
>
> 如果已重設效能結構描述統計資料，或資料庫執行個體已重新啟動，此資訊將無法在 sys 或 performance_schema 中使用。若要保留此資訊，建議您設定外部指標集合。

增強型監控中 os.memory.free 指標的下圖顯示記憶體用量的短暫 7 秒峰值。增強型監控可讓您以最短 1 秒的間隔進行監控，這非常適合用於捕捉像這樣的暫時性峰值。



為了協助診斷此處記憶體用量的原因，我們可以在 sys 記憶體摘要檢視和效能結構描述陳述式摘要資料表中使用 high_alloc 的組合，嘗試識別違規的工作階段和連線。

如預期，由於記憶體使用量目前不高，因此在 current_alloc 下的 sys 結構描述檢視中看不到任何主要違規項目。

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes LIMIT 10;

+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| event_name                                                                    |
 current_count | current_alloc | current_avg_alloc | high_count | high_alloc |
 high_avg_alloc |
+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| memory/innodb/hash0hash                                                       |
     4 | 79.07 MiB     | 19.77 MiB         |          4 | 79.07 MiB  | 19.77 MiB      |
| memory/innodb/os0event                                                        |
 439372 | 60.34 MiB     |  144 bytes        |     439372 | 60.34 MiB  |  144 bytes
 |
| memory/performance_schema/events_statements_summary_by_digest                |
     1 | 40.28 MiB     | 40.28 MiB         |          1 | 40.28 MiB  | 40.28 MiB      |
| memory/mysys/KEY_CACHE                                                        |
     3 | 16.00 MiB     | 5.33 MiB          |          3 | 16.00 MiB  | 5.33 MiB       |
```

```
| memory/performance_schema/events_statements_history_long                    |
     1 | 14.34 MiB     | 14.34 MiB         |          1 | 14.34 MiB  | 14.34 MiB      |
| memory/performance_schema/events_errors_summary_by_thread_by_error          |
   257 | 13.07 MiB     | 52.06 KiB         |        257 | 13.07 MiB  | 52.06 KiB      |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name |
     1 | 11.81 MiB     | 11.81 MiB         |          1 | 11.81 MiB  | 11.81 MiB      |
| memory/performance_schema/events_statements_summary_by_digest.digest_text   |
     1 | 9.77 MiB      | 9.77 MiB          |          1 | 9.77 MiB   | 9.77 MiB       |
| memory/performance_schema/events_statements_history_long.digest_text        |
     1 | 9.77 MiB      | 9.77 MiB          |          1 | 9.77 MiB   | 9.77 MiB       |
| memory/performance_schema/events_statements_history_long.sql_text           |
     1 | 9.77 MiB      | 9.77 MiB          |          1 | 9.77 MiB   | 9.77 MiB       |
+------------------------------------------------------------------------------
+---------------+---------------+------------------+------------+------------
+----------------+
10 rows in set (0.01 sec)
```

展開檢視以依 high_alloc 排序，我們現在可以在這裡看到 memory/temptable/physical_ram 元件是很好的候選項目。其最高耗用 515.00 MiB。

顧名思義，memory/temptable/physical_ram 會在 MySQL 8.0 中引入的 MySQL 中檢測 TEMP 儲存引擎的記憶體用量。如需有關 MySQL 如何使用暫存資料表的詳細資訊，請參閱 MySQL 文件中的 MySQL 中的內部暫存資料表使用。

> **Note**
>
> 我們在此範例中使用 sys.x$memory_global_by_current_bytes 檢視。

```
mysql> SELECT event_name, format_bytes(current_alloc) AS "currently allocated",
 sys.format_bytes(high_alloc) AS "high-water mark"
FROM sys.x$memory_global_by_current_bytes ORDER BY high_alloc DESC LIMIT 10;

+------------------------------------------------------------------------------
+---------------------+-----------------+
| event_name                                                                   |
 currently allocated | high-water mark |
+------------------------------------------------------------------------------
+---------------------+-----------------+
| memory/temptable/physical_ram                                                | 4.00
 MiB            | 515.00 MiB      |
```

```
| memory/innodb/hash0hash                                                    | 79.07
 MiB           | 79.07 MiB       |
| memory/innodb/os0event                                                     | 63.95
 MiB           | 63.95 MiB       |
| memory/performance_schema/events_statements_summary_by_digest              | 40.28
 MiB           | 40.28 MiB       |
| memory/mysys/KEY_CACHE                                                     | 16.00
 MiB           | 16.00 MiB       |
| memory/performance_schema/events_statements_history_long                   | 14.34
 MiB           | 14.34 MiB       |
| memory/performance_schema/events_errors_summary_by_thread_by_error         | 13.07
 MiB           | 13.07 MiB       |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name | 11.81
 MiB           | 11.81 MiB       |
| memory/performance_schema/events_statements_summary_by_digest.digest_text   | 9.77
 MiB            | 9.77 MiB        |
| memory/performance_schema/events_statements_history_long.sql_text          | 9.77
 MiB            | 9.77 MiB        |
+----------------------------------------------------------------------------
+--------------------+-----------------+
10 rows in set (0.00 sec)
```

在 範例 1：持續高記憶體用量 中，我們檢查了每個連線目前的記憶體用量，以判斷哪些連線負責使用有問題的記憶體。在此範例中，記憶體已釋放，因此檢查目前連線的記憶體用量並不實用。

為了深入了解並尋找違規的陳述式、使用者和主機，我們會使用效能結構描述。效能結構描述包含多個由不同維度分割的陳述式摘要資料表，例如事件名稱、陳述式摘要、主機、執行緒和使用者。每個檢視都可讓您深入了解特定陳述式的執行位置及其執行方式。本節著重於 MAX_TOTAL_MEMORY，但您可以在效能結構描述陳述式摘要資料表文件中，找到所有可用資料欄的詳細資訊。

```
mysql> SHOW TABLES IN performance_schema LIKE 'events_statements_summary_%';

+------------------------------------------------------------+
| Tables_in_performance_schema (events_statements_summary_%) |
+------------------------------------------------------------+
| events_statements_summary_by_account_by_event_name         |
| events_statements_summary_by_digest                        |
| events_statements_summary_by_host_by_event_name            |
| events_statements_summary_by_program                       |
| events_statements_summary_by_thread_by_event_name          |
| events_statements_summary_by_user_by_event_name            |
| events_statements_summary_global_by_event_name             |
+------------------------------------------------------------+
```

```
7 rows in set (0.00 sec)
```

首先，我們會檢查 events_statements_summary_by_digest 以查看 MAX_TOTAL_MEMORY。

在這裡，我們可以看到以下內容：

- 具有摘要 20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a 的查詢似乎適合此記憶體用量。MAX_TOTAL_MEMORY 是 537450710，符合我們在 sys.x$memory_global_by_current_bytes 中看到 memory/temptable/physical_ram 事件的高水位標記。
- 它已執行四次 (COUNT_STAR)，第一個是 2024-03-26 04:08:34.943256，最後一個是 2024-03-26 04:43:06.998310。

```
mysql> SELECT SCHEMA_NAME,DIGEST,COUNT_STAR,MAX_TOTAL_MEMORY,FIRST_SEEN,LAST_SEEN
FROM performance_schema.events_statements_summary_by_digest ORDER BY MAX_TOTAL_MEMORY
 DESC LIMIT 5;

+-------------+----------------------------------------------------------------
+------------+------------------+----------------------------
+-----------------------------+
| SCHEMA_NAME | DIGEST                                                          |
 COUNT_STAR | MAX_TOTAL_MEMORY | FIRST_SEEN                 | LAST_SEEN
  |
+-------------+----------------------------------------------------------------
+------------+------------------+----------------------------
+-----------------------------+
| sysbench    | 20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a |
     4 |        537450710 | 2024-03-26 04:08:34.943256 | 2024-03-26 04:43:06.998310 |
| NULL        | f158282ea0313fefd0a4778f6e9b92fc7d1e839af59ebd8c5eea35e12732c45d |
     4 |          3636413 | 2024-03-26 04:29:32.712348 | 2024-03-26 04:36:26.269329 |
| NULL        | 0046bc5f642c586b8a9afd6ce1ab70612dc5b1fd2408fa8677f370c1b0ca3213 |
     2 |          3459965 | 2024-03-26 04:31:37.674008 | 2024-03-26 04:32:09.410718 |
| NULL        | 8924f01bba3c55324701716c7b50071a60b9ceaf17108c71fd064c20c4ab14db |
     1 |          3290981 | 2024-03-26 04:31:49.751506 | 2024-03-26 04:31:49.751506 |
| NULL        | 90142bbcb50a744fcec03a1aa336b2169761597ea06d85c7f6ab03b5a4e1d841 |
     1 |          3131729 | 2024-03-26 04:15:09.719557 | 2024-03-26 04:15:09.719557 |
+-------------+----------------------------------------------------------------
+------------+------------------+----------------------------
+-----------------------------+
5 rows in set (0.00 sec)
```

現在我們知道違規摘要，我們可以取得更多詳細資訊，例如查詢文字、執行該摘要的使用者，以及執行該摘要的位置。根據傳回的摘要文字，我們可以看到這是一個常見的資料表運算式 (CTE)，可建立四個暫存資料表並執行四個資料表掃描，這非常沒有效率。

```
mysql> SELECT
 SCHEMA_NAME,DIGEST_TEXT,QUERY_SAMPLE_TEXT,MAX_TOTAL_MEMORY,SUM_ROWS_SENT,SUM_ROWS_EXAMINED,SUM
FROM performance_schema.events_statements_summary_by_digest
WHERE DIGEST='20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a'\G;

*************************** 1. row ***************************
           SCHEMA_NAME: sysbench
           DIGEST_TEXT: WITH RECURSIVE `cte` ( `n` ) AS ( SELECT ? FROM `sbtest1` UNION
 ALL SELECT `id` + ? FROM `sbtest1` ) SELECT * FROM `cte`
     QUERY_SAMPLE_TEXT: WITH RECURSIVE cte (n) AS (   SELECT 1  from sbtest1 UNION ALL
  SELECT id + 1 FROM sbtest1) SELECT * FROM cte
      MAX_TOTAL_MEMORY: 537450710
         SUM_ROWS_SENT: 80000000
     SUM_ROWS_EXAMINED: 80000000
SUM_CREATED_TMP_TABLES: 4
     SUM_NO_INDEX_USED: 4
1 row in set (0.01 sec)
```

如需 `events_statements_summary_by_digest` 資料表和其他效能結構描述陳述式摘要資料表的詳細資訊，請參閱 MySQL 文件中的陳述式摘要資料表。

您也可以執行 EXPLAIN 或 EXPLAIN ANALYZE 陳述式，以查看更多詳細資訊。

> **Note**
>
> EXPLAIN ANALYZE 可以提供比 EXPLAIN 更多的資訊，但它也會執行查詢，因此請注意。

```
-- EXPLAIN
mysql> EXPLAIN WITH RECURSIVE cte (n) AS (SELECT 1  FROM sbtest1 UNION ALL SELECT id +
 1 FROM sbtest1) SELECT * FROM cte;

+----+-------------+------------+------------+-------+---------------+------+---------
+------+----------+----------+-------------+
| id | select_type | table      | partitions | type  | possible_keys | key  | key_len |
 ref  | rows     | filtered | Extra       |
+----+-------------+------------+------------+-------+---------------+------+---------
+------+----------+----------+-------------+
```

```
|  1 | PRIMARY     | <derived2> | NULL       | ALL   | NULL          | NULL | NULL    |
 NULL | 19221520 |   100.00 | NULL        |
|  2 | DERIVED     | sbtest1    | NULL       | index | NULL          | k_1  | 4       |
 NULL |  9610760 |   100.00 | Using index |
|  3 | UNION       | sbtest1    | NULL       | index | NULL          | k_1  | 4       |
 NULL |  9610760 |   100.00 | Using index |
+----+-------------+------------+------------+-------+---------------+------+---------
+------+----------+----------+-------------+
3 rows in set, 1 warning (0.00 sec)

-- EXPLAIN format=tree
mysql> EXPLAIN format=tree WITH RECURSIVE cte (n) AS (SELECT 1 FROM sbtest1 UNION ALL
 SELECT id + 1 FROM sbtest1) SELECT * FROM cte\G;

*************************** 1. row ***************************
EXPLAIN: -> Table scan on cte  (cost=4.11e+6..4.35e+6 rows=19.2e+6)
    -> Materialize union CTE cte  (cost=4.11e+6..4.11e+6 rows=19.2e+6)
        -> Index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
        -> Index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
1 row in set (0.00 sec)

-- EXPLAIN ANALYZE
mysql> EXPLAIN ANALYZE WITH RECURSIVE cte (n) AS (SELECT 1 from sbtest1 UNION ALL
 SELECT id + 1 FROM sbtest1) SELECT * FROM cte\G;

*************************** 1. row ***************************
EXPLAIN: -> Table scan on cte  (cost=4.11e+6..4.35e+6 rows=19.2e+6) (actual
 time=6666..9201 rows=20e+6 loops=1)
    -> Materialize union CTE cte  (cost=4.11e+6..4.11e+6 rows=19.2e+6) (actual
 time=6666..6666 rows=20e+6 loops=1)
        -> Covering index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
 (actual time=0.0365..2006 rows=10e+6 loops=1)
        -> Covering index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
 (actual time=0.0311..2494 rows=10e+6 loops=1)
1 row in set (10.53 sec)
```

但誰執行了查詢？ 我們可以在效能結構描述中看到 destructive_operator 使用者擁有 537450710 的 MAX_TOTAL_MEMORY，這再次符合先前的結果。

> **Note**
>
> 效能結構描述存放在記憶體中，因此不應依賴而作為稽核的唯一來源。如果您需要維護陳述式執行的歷程記錄，以及從哪些使用者執行，我們建議您啟用 Aurora 進階稽核。如果您還需要維護記憶體用量的相關資訊，建議您設定監控以匯出和儲存這些值。

```
mysql> SELECT USER,EVENT_NAME,COUNT_STAR,MAX_TOTAL_MEMORY FROM
 performance_schema.events_statements_summary_by_user_by_event_name
ORDER BY MAX_CONTROLLED_MEMORY DESC LIMIT 5;

+----------------------+---------------------------+------------+------------------+
| USER                 | EVENT_NAME                | COUNT_STAR | MAX_TOTAL_MEMORY |
+----------------------+---------------------------+------------+------------------+
| destructive_operator | statement/sql/select      |          4 |        537450710 |
| rdsadmin             | statement/sql/select      |       4172 |          3290981 |
| rdsadmin             | statement/sql/show_tables |          2 |          3615821 |
| rdsadmin             | statement/sql/show_fields |          2 |          3459965 |
| rdsadmin             | statement/sql/show_status |         75 |          1914976 |
+----------------------+---------------------------+------------+------------------+
5 rows in set (0.00 sec)

mysql> SELECT HOST,EVENT_NAME,COUNT_STAR,MAX_TOTAL_MEMORY FROM
 performance_schema.events_statements_summary_by_host_by_event_name
WHERE HOST != 'localhost' AND COUNT_STAR>0 ORDER BY MAX_CONTROLLED_MEMORY DESC LIMIT 5;

+------------+----------------------+------------+------------------+
| HOST       | EVENT_NAME           | COUNT_STAR | MAX_TOTAL_MEMORY |
+------------+----------------------+------------+------------------+
| 10.0.8.231 | statement/sql/select |          4 |        537450710 |
+------------+----------------------+------------+------------------+
1 row in set (0.00 sec)
```

範例 3：可釋放的記憶體持續下降，且不會回收

InnoDB 資料庫引擎針對不同的元件採用一系列專門的記憶體追蹤事件。這些特定事件允許精細追蹤索引鍵 InnoDB 子系統中的記憶體用量，例如：

- `memory/innodb/buf0buf` – 專用於監控 InnoDB 緩衝集區的記憶體配置。
- `memory/innodb/ibuf0ibuf` – 專門追蹤與 InnoDB 變更緩衝區相關的記憶體變更。

為了識別記憶體的最高取用者，我們可以查詢 sys.memory_global_by_current_bytes：

```
mysql> SELECT event_name,current_alloc FROM sys.memory_global_by_current_bytes LIMIT
 10;

+-----------------------------------------------------------------+---------------+
| event_name                                                      | current_alloc |
+-----------------------------------------------------------------+---------------+
| memory/innodb/memory                                            | 5.28 GiB      |
| memory/performance_schema/table_io_waits_summary_by_index_usage | 495.00 MiB    |
| memory/performance_schema/table_shares                          | 488.00 MiB    |
| memory/sql/TABLE_SHARE::mem_root                                | 388.95 MiB    |
| memory/innodb/std                                               | 226.88 MiB    |
| memory/innodb/fil0fil                                           | 198.49 MiB    |
| memory/sql/binlog_io_cache                                      | 128.00 MiB    |
| memory/innodb/mem0mem                                           | 96.82 MiB     |
| memory/innodb/dict0dict                                         | 96.76 MiB     |
| memory/performance_schema/rwlock_instances                      | 88.00 MiB     |
+-----------------------------------------------------------------+---------------+
10 rows in set (0.00 sec)
```

結果顯示 memory/innodb/memory 是目前配置記憶體中使用 5.28 GiB 的最高取用者。此事件可做為與更具體等待事件無關之各種 InnoDB 元件的記憶體配置類別，如先前 memory/innodb/buf0buf 所述。

在確定 InnoDB 元件是記憶體的主要取用者之後，我們可以使用下列 MySQL 命令深入探討細節：

```
SHOW ENGINE INNODB STATUS \G;
```

SHOW ENGINE INNODB STATUS 命令提供 InnoDB 儲存引擎的完整狀態報告，包括不同 InnoDB 元件的詳細記憶體用量統計資料。它有助於識別哪些特定的 InnoDB 結構或操作耗用最多的記憶體。如需詳細資訊，請參閱 MySQL 文件中的 InnoDB 記憶體內結構。

分析 InnoDB 狀態報告的 BUFFER POOL AND MEMORY 區段時，我們看到 5,051,647,748 個位元組 (4.7 GiB) 配置到字典物件快取，該快取佔了 memory/innodb/memory 所追蹤記憶體的 89%。

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total large memory allocated 0
Dictionary memory allocated 5051647748
```

```
Buffer pool size 170512
Free buffers 142568
Database pages 27944
Old database pages 10354
Modified db pages 6
Pending reads 0
```

字典物件快取是共用的全域快取，可將先前存取的資料字典物件存放在記憶體中，以啟用物件重複使用並改善效能。字典物件快取的高記憶體配置會建議資料字典快取中的大量資料庫物件。

現在我們知道資料字典快取是主要取用者，我們繼續檢查資料字典快取是否有開啟的資料表。若要尋找資料表定義快取中的資料表數目，請查詢全域狀態變數 open_table_definitions。

```
mysql> show global status like 'open_table_definitions';

+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Open_table_definitions | 20000 |
+-------------------------+-------+
1 row in set (0.00 sec)
```

如需詳細資訊，請參閱 MySQL 文件中的 MySQL 如何開啟及關閉資料表。

您可以透過限制資料庫叢集或資料庫執行個體參數群組中的 table_definition_cache 參數，來限制資料字典快取中的資料表定義數目。對於 Aurora MySQL，此值可作為資料表定義快取中資料表數量的軟性限制。預設值取決於執行個體類別，並設定為以下內容：

```
LEAST({DBInstanceClassMemory/393040}, 20000)
```

當資料表數量超過 table_definition_cache 限制時，至少最近最少使用 (LRU) 機制會從快取中移出和移除資料表。不過，涉及外部索引鍵關係的資料表不會放置在 LRU 清單中，以防止其移除。

在我們目前的案例中，我們會執行 FLUSH TABLES 來清除資料表定義快取。此動作會導致 Open_table_definitions 全域狀態變數從 20,000 大幅降至 12，如下所示：

```
mysql> show global status like 'open_table_definitions';

+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
```

```
| Open_table_definitions | 12    |
+------------------------+-------+
1 row in set (0.00 sec)
```

儘管如此減少，我們仍觀察到 memory/innodb/memory 的記憶體配置在 5.18 GiB 維持很高，而配置的字典記憶體也保持不變。這在下列查詢結果中很明顯：

```
mysql> SELECT event_name,current_alloc FROM sys.memory_global_by_current_bytes LIMIT
 10;

+-----------------------------------------------------------------+---------------+
| event_name                                                      | current_alloc |
+-----------------------------------------------------------------+---------------+
| memory/innodb/memory                                            | 5.18 GiB      |
| memory/performance_schema/table_io_waits_summary_by_index_usage | 495.00 MiB    |
| memory/performance_schema/table_shares                          | 488.00 MiB    |
| memory/sql/TABLE_SHARE::mem_root                                | 388.95 MiB    |
| memory/innodb/std                                               | 226.88 MiB    |
| memory/innodb/fil0fil                                           | 198.49 MiB    |
| memory/sql/binlog_io_cache                                      | 128.00 MiB    |
| memory/innodb/mem0mem                                           | 96.82 MiB     |
| memory/innodb/dict0dict                                         | 96.76 MiB     |
| memory/performance_schema/rwlock_instances                      | 88.00 MiB     |
+-----------------------------------------------------------------+---------------+
10 rows in set (0.00 sec)
```

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total large memory allocated 0
Dictionary memory allocated 5001599639
Buffer pool size 170512
Free buffers 142568
Database pages 27944
Old database pages 10354
Modified db pages 6
Pending reads 0
```

這種持續高的記憶體用量可以歸因於涉及外部索引鍵關係的資料表。這些資料表不會放在 LRU 清單中進行移除，這解釋了即使在清除資料表定義快取之後，記憶體配置仍保持很高的原因。

解決此問題：

1. 檢閱和最佳化資料庫結構描述，特別是外部索引鍵關係。
2. 請考慮移至具有更多記憶體的大型資料庫執行個體類別，以容納您的字典物件。

透過遵循這些步驟並了解記憶體配置模式，您可以更好地管理 Aurora MySQL 資料庫執行個體中的記憶體用量，並防止因記憶體壓力導致的潛在效能問題。

故障診斷 Aurora MySQL 資料庫記憶體不足的問題

Aurora MySQL `aurora_oom_response` 執行個體層級參數可啟用資料庫執行個體監控系統記憶體，並估計各種陳述式和連線耗用的記憶體。如果系統執行時記憶體不足，它可以執行一連串的動作來嘗試釋放記憶體。它會這樣做以嘗試避免因記憶體不足 (OOM) 導致資料庫重新啟動。此執行個體層級參數會使用一個字串，字串內容是以逗號分隔的動作，當記憶體不足時，資料庫執行個體即會採取這些動作。Aurora MySQL 第 2 版和第 3 版支援 `aurora_oom_response` 參數。

下列值及其組合可用於 `aurora_oom_response` 參數。空字串表示不採取任何動作，並有效地關閉此功能，讓資料庫容易重新啟動 OOM。

- `decline` – 一旦資料庫執行個體記憶體不足，便拒絕新的查詢。
- `kill_connect` – 關閉耗用大量記憶體的資料庫連線，並結束目前的交易和資料定義語言 (DDL) 陳述式。Aurora MySQL 第 2 版不支援此回應。

  如需詳細資訊，請參閱 MySQL 文件中的 KILL 陳述式。
- `kill_query` – 依記憶體消耗的遞減順序結束查詢，直到執行個體記憶體高於低閾值。DDL 陳述式不會結束。

  如需詳細資訊，請參閱 MySQL 文件中的 KILL 陳述式。
- `print` – 只會列印耗用大量記憶體的查詢。
- `tune` – 調整內部資料表快取，以釋放部分記憶體給系統。Aurora MySQL 會在記憶體不足情況下減少用於快取 (例如 `table_open_cache` 和 `table_definition_cache`) 的記憶體。最終，當系統不再記憶體不足時，Aurora MySQL 會將記憶體使用量設回正常情況。

  如需詳細資訊，請參閱 MySQL 文件中的 table_open_cache 和 table_definition_cache。
- `tune_buffer_pool` – 減少緩衝集區的大小以釋出一些記憶體，並讓資料庫伺服器能夠處理連線。Aurora MySQL 3.06 版及更新版本支援此回應。

  您必須在 `aurora_oom_response` 參數值中與 `kill_query` 或 `kill_connect` 配對 `tune_buffer_pool`。如果沒有，即使您將 `tune_buffer_pool` 包含在參數值中，也不會發生緩衝集區調整大小。

在低於 3.06 的 Aurora MySQL 版本中，對於記憶體小於或等於 4 GiB 的資料庫執行個體類別，當執行個體處於記憶體壓力下時，預設動作包括 `print`、`tune`、`decline` 和 `kill_query`。如果資料庫執行個體的記憶體大於 4 GiB，則參數值依預設為空白 (停用)。

在 Aurora MySQL 3.06 版及更高版本中，對於記憶體小於或等於 4 GiB 的資料庫執行個體類別，Aurora MySQL 也會關閉最耗用記憶體的連線 (`kill_connect`)。對於記憶體大於 4 GiB 的資料庫執行個體類別，預設參數值為 `print`。

在 Aurora MySQL 3.09 版和更新版本中，對於記憶體大於 4 GiB 的資料庫執行個體類別，預設參數值為 `print,decline,kill_connect`。

如果您經常遇到記憶體不足問題，則可以在啟用 `performance_schema` 時，使用記憶體摘要資料表來監控記憶體用量。

如需與 OOM 相關的 Amazon CloudWatch 指標，請參閱 Amazon Aurora 的執行個體層級指標。如需與 OOM 相關的全域狀態變數，請參閱 Aurora MySQL 全域狀態變數。

記錄 Aurora MySQL 資料庫

Aurora MySQL 日誌提供有關資料庫活動和錯誤的基本資訊。透過啟用這些日誌，您可以識別和故障診斷問題、了解資料庫效能，以及稽核資料庫活動。建議您為所有 Aurora MySQL 資料庫執行個體啟用這些日誌，以確保資料庫的最佳效能和可用性。您可以啟用下列類型的記錄。每個日誌都包含可能導致發現資料庫處理影響的特定資訊。

- 錯誤 – 只有在啟動、當機以及發生錯誤時，Aurora MySQL 才會寫入錯誤日誌。資料庫執行個體可在未寫入新項目到錯誤日誌的情況下持續執行數小時或數日。若您沒有看到最近的項目，這是因為伺服器未遇到需寫入日誌項目的錯誤。錯誤記錄預設為啟用。如需更多詳細資訊，請參閱 Aurora MySQL 錯誤日誌。
- 一般 – 一般日誌提供有關資料庫活動的詳細資訊，包括資料庫引擎執行的所有 SQL 陳述式。如需啟用一般記錄和設定記錄參數的詳細資訊，請參閱 MySQL 文件中的 Aurora MySQL 慢查詢與一般查詢 和一般查詢日誌。

  > **Note**
  >
  > 一般日誌可能會成長為非常大，並消耗您的儲存體。如需更多詳細資訊，請參閱 Aurora MySQL 的日誌輪換與保留。

- 緩慢查詢日誌 — 包含執行時間超過 long_query_time 秒數，且至少有 min_examined_row_limit 列需要檢查之 SQL 陳述式的緩慢查詢日誌。您可以使用緩慢查詢日誌來尋找需要很長時間才能執行的查詢，因此是最佳化的候選項目。

  `long_query_time` 的預設值為 10 秒。我們建議您從高值開始，以識別最慢的查詢，然後著手進行微調。

  您也可以使用相關參數，例如 `log_slow_admin_statements` 和 `log_queries_not_using_indexes`。比較 `rows_examined` 與 `rows_returned`。如果 `rows_examined` 明顯大於 `rows_returned`，則這些查詢可能會遭到封鎖。

  在 Aurora MySQL 第 3 版中，您可以讓 `log_slow_extra` 取得更多詳細資訊。如需詳細資訊，請參閱 MySQL 文件中的緩慢查詢日誌內容。您也可以在工作階段層級修改 `long_query_time`，以互動方式偵錯查詢執行，而如果 `log_slow_extra` 全域啟用，這特別有用。

  如需啟用緩慢查詢記錄和設定記錄參數的詳細資訊，請參閱 MySQL 文件中的 Aurora MySQL 慢查詢與一般查詢和緩慢查詢日誌。

- 稽核 – 稽核日誌會監控和記錄資料庫活動。Aurora MySQL 的稽核日誌稱為進階稽核。若要啟用進階稽核，請設定某些資料庫叢集參數。如需更多詳細資訊，請參閱 使用進階稽核與 Amazon Aurora MySQL 資料庫叢集搭配。
- 二進位 – 二進位日誌 (binlog) 包含描述資料庫變更的事件，例如資料表建立操作和資料表資料的變更。它還包含可能已進行變更之陳述式的事件 (例如，符合沒有資料列的 DELETE)，除非使用資料列型記錄。二進位日誌也包含有關每個陳述式使用該更新資料多久的資訊。

  在啟用二進位記錄的情況下執行伺服器會使效能稍微變慢。不過，二進位登入的好處可讓您設定複寫和還原操作，通常超過此輕微效能降低的缺點。

  > **Note**
  >
  > Aurora MySQL 不需要二進位記錄即可進行還原操作。

  如需啟用二進位記錄和設定二進位日誌格式的詳細資訊，請參閱 MySQL 文件中的 為單一可用區資料庫設定 Aurora MySQL 二進位記錄 和二進位日誌。

您可以將錯誤、一般、緩慢、查詢和稽核日誌發佈至 Amazon CloudWatch Logs。如需更多詳細資訊，請參閱 將資料庫日誌發佈至 Amazon CloudWatch Logs。

另一個用於摘要緩慢、一般和二進位日誌檔案的實用工具是 pt-query-digest。

針對 Aurora MySQL 資料庫的連線問題進行疑難排解

確保應用程式與 RDS 資料庫執行個體之間的可靠連線，對於工作負載的順暢操作至關重要。不過，連線問題可能會因為各種因素而產生，例如網路組態、驗證問題或資源限制。本指南旨在提供全方位方法來針對 Aurora MySQL 的連線問題進行疑難排解。

內容

- 識別 Aurora MySQL 的資料庫連線問題
- 收集 Aurora MySQL 連線問題的資料
- 監控 Aurora MySQL 的資料庫連線
  - Aurora MySQL 的其他監控
- Aurora MySQL 的連線錯誤代碼
- Aurora MySQL 的參數調校建議
- 針對 Aurora MySQL 資料庫連線問題進行疑難排解的範例
  - 範例 1：針對失敗的連線嘗試進行疑難排解
  - 範例 2：針對異常用戶端中斷連線進行疑難排解
  - 範例 3：對 IAM 失敗的連線嘗試進行故障診斷

識別 Aurora MySQL 的資料庫連線問題

識別連線問題的特定類別有助於縮小潛在原因範圍，並引導疑難排解程序。每個類別可能需要不同的診斷和解決方法和技術。資料庫連線問題可廣泛分類為下列類別。

連線錯誤和例外狀況

連線錯誤和例外狀況可能因各種原因而發生，例如不正確的連線字串、驗證失敗、網路中斷或資料庫伺服器問題。原因可能包括設定錯誤的連線參數、無效的憑證、網路中斷或資料庫伺服器當機或重新啟動。設定錯誤的安全群組、虛擬私有雲端 (VPC) 設定、網路存取控制清單 (ACL) 以及與子網路相關聯的路由表也可能導致連線問題。

達到連線限制

當資料庫伺服器的並行連線數目超過允許的上限時，就會發生此問題。資料庫伺服器通常具有由叢集和執行個體參數群組中的參數 max_connections 定義的可設定連線上限。透過強加連線限制，資料庫伺服器可確保其有足夠的資源 (例如記憶體、CPU 和檔案控點) 可有效率地處理現有連線並提供可接受的效能。原因可能包括應用程式中的連線洩漏、連線集區效率低下，或連線請求意外突增。

連線逾時。

當用戶端應用程式無法在指定的逾時期間內與資料庫伺服器建立連線時，就會發生連線逾時。常見原因包括網路問題、伺服器過載、防火牆規則和設定錯誤的連線設定。

閒置連線逾時

資料庫伺服器可能會自動關閉長時間處於非作用中狀態的閒置連線，以節省資源。此逾時通常可使用 `wait_timeout` 和 `interactive_timeout parameters` 來設定，且應根據應用程式的連線使用模式進行調整。原因可能包括讓連線長時間閒置的應用程式邏輯，或不當的連線管理。

現有連線的間歇中斷連線

此錯誤類別是指用戶端應用程式與資料庫之間已建立的連線，即使處於作用中狀態和使用中，仍會以不規律的間隔意外終止或中斷連線的情況。這些中斷連線會間歇性發生，這表示它們會以不規律的間隔發生，而不是一致的間隔。原因可包含下列項目：

- 資料庫伺服器問題，例如重新啟動或容錯移轉
- 不當的應用程式連線處理
- 負載平衡和代理問題
- 網路不穩定
- 連線路徑中涉及的第三方元件或中介軟體問題
- 查詢執行逾時
- 伺服器端或用戶端的資源限制

透過全面監控、記錄和分析來識別根本原因至關重要，同時實作適當的錯誤處理、連線集區和重試機制，有助於緩解這些間歇性中斷連線對應用程式功能和使用者體驗的影響。

收集 Aurora MySQL 連線問題的資料

收集與應用程式、資料庫、網路和基礎設施元件相關的完整資料，對於有效針對應用程式與 Aurora MySQL 資料庫之間的連線問題進行疑難排解至關重要。透過收集相關的日誌、組態和診斷資訊，您可以獲得寶貴的洞察，以協助識別連線問題的根本原因，並引導您進行適當的解決方案。

網路日誌和組態，例如安全群組規則、VPC 設定和路由表，對於識別可能使應用程式無法與資料庫建立成功連線的潛在網路相關瓶頸或組態錯誤至關重要。透過分析這些網路元件，您可以確保必要的連接埠已開啟、允許 IP 地址，以及正確設定路由組態。

時間戳記

發生連線問題時，請記錄確切的時間戳記。這有助於識別模式，或將問題與其他事件或活動建立關聯。

資料庫引擎日誌

除了一般資料庫日誌之外，請檢閱資料庫引擎日誌 (例如 MySQL 錯誤日誌和慢查詢日誌) 是否有任何相關資訊或可能與間歇性連線問題相關的錯誤。如需詳細資訊，請參閱記錄 Aurora MySQL 資料庫。

用戶端應用程式日誌

從連線至資料庫的用戶端應用程式收集詳細日誌。應用程式日誌可從應用程式的角度提供連線嘗試、錯誤和任何相關資訊的可見性，這可能會顯示與連線字串、驗證憑證或應用程式層級連線處理相關的問題。

另一方面，資料庫日誌提供對資料庫端錯誤、慢速查詢或可能導致連線問題的事件的洞察。如需詳細資訊，請參閱記錄 Aurora MySQL 資料庫。

用戶端環境變數

檢查用戶端的任何環境變數或組態設定是否可能影響資料庫連線，例如代理設定、SSL/TLS 設定或任何其他相關變數。

用戶端程式庫版本

確定用戶端使用用於資料庫連線的任何資料庫驅動程式、程式庫或架構的最新版本。過期版本可能會有已知問題或相容性問題。

用戶端網路擷取

使用 Wireshark 或 `tcpdump` 等工具在發生連線問題期間，在用戶端執行網路擷取。這有助於識別用戶端上任何網路相關問題或異常。

用戶端網路拓撲

了解用戶端的網路拓撲，包括任何防火牆、負載平衡器或其他元件，例如 RDS Proxy 或 Proxy SQL，這些元件正在與資料庫進行連線，而不是用戶端直接進行連線。

用戶端作業系統設定

檢查可能影響網路連線的用戶端作業系統設定，例如防火牆規則、網路轉接器設定，以及任何其他相關設定。

連線集區組態

如果您在應用程式中使用連線集區機制，請檢閱組態設定並監控集區指標 (例如作用中連線、閒置連線和連線逾時)，以確保集區正常運作。同時檢閱集區設定，例如集區大小上限、集區大小下限和連線驗證設定，以確保設定正確。

連接字串

連線字串通常包含主機名稱或端點、連接埠號碼、資料庫名稱和驗證憑證等參數。分析連線字串有助於識別可能導致連線問題的潛在錯誤組態或不正確的設定。例如，不正確的主機名稱或連接埠號碼可能會導致用戶端無法連線到資料庫執行個體，而無效的驗證憑證可能會導致驗證失敗和連線遭拒。此外，連線字串可能會顯示與連線集區、逾時或其他可能導致連線問題之連線特定設定相關的問題。提供用戶端應用程式使用的完整連線字串，有助於精確找出用戶端上的任何錯誤組態。

資料庫指標

在發生連線問題期間監控資料庫指標，例如 CPU 用量、記憶體用量和磁碟 I/O。這些項目可協助識別資料庫執行個體是否遇到資源爭用或效能問題。

DB engine version (資料庫引擎版本)

請注意 Aurora MySQL 資料庫引擎版本。 AWS 會定期發行更新，以解決已知問題、安全漏洞並引入效能增強功能。因此，強烈建議您升級至最新的可用版本，因為這些更新通常包含與連線能力、效能和穩定性相關的錯誤修正和改進。提供資料庫版本資訊以及其他收集的詳細資訊，可協助 支援 有效診斷和解決連線問題。

網路指標

在發生連線問題期間收集網路指標，例如延遲、封包遺失和輸送量。`ping`、`traceroute` 和網路監控工具等工具可協助收集此資料。

來源和用戶端詳細資訊

判斷正在啟動資料庫連線的應用程式伺服器、負載平衡器或任何其他元件的 IP 地址。這可以是單一 IP 地址或 IP 地址範圍 (CIDR 表示法)。如果來源是 Amazon EC2 執行個體，也有助於檢閱與執行個體相關聯的執行個體類型、可用區域、子網路 ID 和安全群組，以及私有 IP 地址和公有 IP 地址等網路介面詳細資訊。

透過徹底分析收集的資料，您可以識別錯誤組態、資源限制、網路中斷或其他導致間歇性或持久性連線問題的基礎問題。此資訊可讓您採取有針對性的動作，例如調整組態、解決網路問題，或解決應用程式層級連線處理。

監控 Aurora MySQL 的資料庫連線

若要監控連線問題並進行疑難排解，您可以使用下列指標和功能。

CloudWatch 指標

- `CPUUtilization` – 資料庫執行個體上的高 CPU 用量可能會導致查詢執行緩慢，進而導致連線逾時或遭拒。
- `DatabaseConnections` – 監控資料庫執行個體的作用中連線數目。接近已設定最大值的高連線數量，可能表示潛在的連線問題或連線集區耗盡。
- `FreeableMemory` – 由於資源限制，可用記憶體不足可能會導致效能問題和連線問題。
- `NetworkReceiveThroughput` 和 `NetworkTransmitThroughput` – 網路輸送量異常峰值或下降可能表示連線問題或網路瓶頸。

績效詳情指標

若要使用 Performance Insights 針對 Aurora MySQL 中的連線問題進行疑難排解，請分析資料庫指標，如下所示：

- Aborted_clients
- Aborted_connects
- 連線
- max_connections
- Threads_connected
- Threads_created
- Threads_running

這些指標可協助您識別連線瓶頸、偵測網路或驗證問題、最佳化連線集區，並確保有效的執行緒管理。如需詳細資訊，請參閱Aurora MySQL 的績效詳情計數器。

Performance Insights 功能

- 資料庫負載 – 視覺化一段時間內的資料庫負載，並將其與連線問題或效能降級建立關聯。
- SQL 統計資料 – 分析 SQL 統計資料，以識別可能造成連線問題的低效率查詢或資料庫操作。
- 熱門查詢 – 識別和分析資源最密集的查詢，這有助於識別可能導致連線問題的潛在效能瓶頸或長時間執行的查詢。

透過監控這些指標並利用 Performance Insights，您可以了解資料庫執行個體的效能、資源用量，以及可能導致連線問題的潛在瓶頸。例如：

- 接近上限的高 DatabaseConnections 可能表示連線集區耗盡或不當的連線處理，導致連線問題。
- 高 CPUUtilization 或低 FreeableMemory 可能表示資源限制，這可能會導致查詢執行緩慢和連線逾時或遭拒。
- 分析熱門查詢和 SQL 統計資料有助於識別可能導致連線問題的效率低下或資源密集型查詢。

此外，監控 CloudWatch Logs 和設定警示可協助您在連線問題升級之前主動識別和回應。

請務必注意，雖然這些指標和工具可以提供寶貴的洞察，但應該與其他疑難排解步驟搭配使用。透過檢閱網路組態、安全群組規則和應用程式層級連線處理，您可以全面診斷和解決 Aurora MySQL 資料庫執行個體的連線問題。

Aurora MySQL 的其他監控

CloudWatch 指標

- AbortedClients – 追蹤未正確關閉的用戶端連線數目。
- AuroraSlowConnectionHandleCount – 追蹤緩慢連線處理操作的數量，指出潛在的連線問題或效能瓶頸。
- AuroraSlowHandshakeCount – 測量緩慢交握操作的數量，這也可能是連線問題的指標。
- ConnectionAttempts – 測量對 Aurora MySQL 資料庫執行個體進行的連線嘗試次數。

全域狀態變數

Aurora_external_connection_count – 顯示資料庫執行個體的資料庫連線數目，排除用於資料庫運作狀態檢查的 RDS 服務連線。

透過監控這些指標和全域狀態變數，您可以了解可能導致 Amazon Aurora MySQL 執行個體連線問題的連線模式、錯誤和潛在瓶頸。

例如，大量 AbortedClients 或 AuroraSlowConnectionHandleCount 可能表示連線問題。

此外，設定 CloudWatch 警示和通知可協助您在連線問題升級並影響應用程式效能之前，主動識別和回應連線問題。

Aurora MySQL 的連線錯誤代碼

以下是 Aurora MySQL 資料庫的一些常見連線錯誤，以及其錯誤代碼和說明。

錯誤代碼 1040：太多連線

當用戶端嘗試建立的連線數量超過資料庫伺服器允許的上限時，就會發生此錯誤。可能的子句包括：

- 連線集區錯誤組態 – 如果使用連線集區機制，請確定集區大小上限未設定得太高，且連線已正確釋放回集區。
- 資料庫執行個體組態 – 驗證資料庫執行個體允許的連線設定上限，並視需要設定 `max_connections` 參數進行調整。
- 高並行 – 如果多個用戶端或應用程式同時連線到資料庫，可能會達到允許的連線上限。

錯誤代碼 1045：拒絕使用者 '...'@'...' 的存取 (使用密碼：是/否)

此錯誤表示嘗試連線至資料庫時發生驗證失敗。可能的子句包括：

- 驗證外掛程式相容性 – 檢查用戶端使用的驗證外掛程式是否與資料庫伺服器的驗證機制相容。
- 不正確的使用者名稱和密碼 – 確認在連線字串或驗證機制中使用正確的使用者名稱和密碼。
- 使用者許可 – 確定使用者具有從指定主機或網路連線至資料庫執行個體的必要許可。

錯誤代碼 1049：不明資料庫 '...'

此錯誤表示用戶端嘗試連線到伺服器上不存在的資料庫。可能的子句包括：

- 未建立資料庫 – 確定已在資料庫伺服器上建立指定的資料庫。
- 資料庫名稱不正確 – 再次檢查在連線字串或查詢中使用的資料庫名稱是否正確。
- 使用者許可 – 確認使用者具有存取指定資料庫的必要許可。

錯誤代碼 1153：擁有大於 'max_allowed_packet' 位元組的封包

當用戶端嘗試傳送或接收的資料超過資料庫伺服器允許的封包大小上限時，就會發生此錯誤。可能的子句包括：

- 大型查詢或結果集 – 如果執行涉及大量資料的查詢，可能會超過封包大小限制。
- 設定錯誤的封包大小設定 – 檢查資料庫伺服器上的 `max_allowed_packet` 設定，並視需要進行調整。
- 網路組態問題 – 確定網路組態 (例如 MTU 大小) 允許所需的封包大小。

錯誤代碼 1226：使用者 '...' 已超過 'max_user_connections' 資源 (目前值：...)

此錯誤表示使用者已超過資料庫伺服器允許的並行連線數目上限。可能的原因包括下列項目：

- 連線集區錯誤組態 – 如果使用連線集區機制，請確定集區大小上限針對使用者的連線限制未設定得太高。

- 資料庫執行個體組態 – 驗證資料庫執行個體的 max_user_connections 設定，並視需要進行調整。
- 高並行 – 如果多個用戶端或應用程式同時使用相同使用者連線到資料庫，可能會達到使用者特定連線限制。

錯誤代碼 2003：無法在 '...' 上連線至 MySQL 伺服器 (10061)

此錯誤通常發生在用戶端無法與資料庫伺服器建立 TCP/IP 連線時。其可能由各種問題造成，例如下列項目：

- 資料庫執行個體狀態 – 確定資料庫執行個體處於 available 狀態，而且不會進行任何維護或備份操作。
- 防火牆規則 – 檢查是否有任何防火牆 (作業系統、網路或安全群組) 封鎖指定連接埠上的連線 (針對 MySQL 通常是 3306)。
- 不正確的主機名稱或端點 – 請確定在連線字串中使用的主機名稱或端點正確且符合資料庫執行個體。
- 網路連線問題 – 確認用戶端機器可以透過網路連線資料庫執行個體。檢查是否有任何網路中斷、路由問題，或 VPC 或子網路錯誤組態。

錯誤代碼 2005：不明 MySQL 伺服器主機 '...' (11001)

當用戶端無法將資料庫伺服器的主機名稱或端點解析為 IP 地址時，就會發生此錯誤。可能的子句包括：

- DNS 解析問題 – 確認用戶端機器可以使用 DNS 正確解析主機名稱。檢查 DNS 設定、DNS 快取，並嘗試使用 IP 地址而非主機名稱。
- 不正確的主機名稱或端點 – 再次檢查在連線字串中使用的主機名稱或端點是否正確。
- 網路組態問題 – 確定用戶端的網路組態 (例如 VPC、子網路和路由表) 允許 DNS 解析和連線至資料庫執行個體。

錯誤代碼 2026：SSL 連線錯誤

當連線嘗試期間發生 SSL/TLS 組態或憑證驗證問題時，就會發生此錯誤。可能的子句包括：

- 憑證過期 – 檢查伺服器使用的 SSL/TLS 憑證是否已過期且需要續約。
- 憑證驗證問題 – 驗證用戶端是否能夠正確驗證伺服器的 SSL/TLS 憑證，以及憑證是否可信任。
- 網路組態問題 – 確定網路組態允許 SSL/TLS 連線，而且不會封鎖或干擾 SSL/TLS 交握程序。
- SSL/TLS 組態不符 – 確定用戶端和伺服器上的 SSL/TLS 設定 (例如，密碼套件和通訊協定版本) 相容。

透過了解每個錯誤代碼的詳細說明和潛在原因，您可以在使用 Aurora MySQL 資料庫時更好地疑難排解和解決連線問題。

Aurora MySQL 的參數調校建議

連線數上限

調整這些參數有助於防止因達到允許的最大連線限制而導致的連線問題。請確定根據應用程式的並行要求和資源限制，適當地設定這些值。

- max_connections – 此參數指定允許的資料庫執行個體並行連線數目上限。
- max_user_connections – 您可以在使用者建立和修改期間指定此參數，並設定針對特定使用者帳戶允許的並行連線數目上限。

網路緩衝區大小

增加這些值可以改善網路效能，尤其是涉及大型資料傳輸或結果集的工作負載。不過請小心謹慎，因為較大的緩衝區大小可能會耗用更多記憶體。

- net_buffer_length – 此參數會設定用戶端連線和結果緩衝區的初始大小，平衡記憶體用量與查詢效能。
- max_allowed_packet – 此參數指定資料庫執行個體可傳送或接收的單一網路封包大小上限。

網路壓縮 (用戶端)

啟用網路壓縮可以減少網路頻寬用量，但可能會增加用戶端和伺服器端的 CPU 額外負荷。

- compress – 此參數會啟用或停用用戶端/伺服器通訊的網路壓縮。
- compress_protocol – 此參數指定用於網路通訊的壓縮通訊協定。

網路效能調校

調整這些逾時有助於管理閒置連線並防止資源耗盡，但請小心謹慎，因為低值可能會導致連線提早終止。

- interactive_timeout – 此參數指定伺服器在互動式連線上關閉活動之前等待的秒數。
- wait_timeout – 此參數決定伺服器在非互動式連線上關閉活動之前等待的秒數。

網路逾時設定

調整這些逾時有助於解決與緩慢或無回應連線相關的問題。但請小心不要將其設得太低，因為這可能會導致連線過早失敗。

- net_read_timeout – 此參數指定在結束讀取操作之前等待來自連線之更多資料的秒數。
- net_write_timeout – 此參數決定在結束寫入操作之前等待區塊寫入連線的秒數。

針對 Aurora MySQL 資料庫連線問題進行疑難排解的範例

下列範例示範如何識別 Aurora MySQL 的資料庫連線問題並進行疑難排解。

範例 1：針對失敗的連線嘗試進行疑難排解

連線嘗試可能會因為多種原因而失敗，包括驗證失敗、SSL/TLS 交握失敗、達到 `max_connections` 限制，以及資料庫執行個體的資源限制。

您可以從 Performance Insights 或使用下列命令來追蹤失敗連線的數量。

```
mysql> show global status like 'aborted_connects';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| Aborted_connects | 7     |
+------------------+-------+
1 row in set (0.00 sec)
```

如果 `Aborted_connects` 的數量隨時間增加，則應用程式可能會有間歇性連線問題。

您可以使用 Aurora 進階稽核來記錄與用戶端連線的連線和中斷連線。您可以在資料庫叢集參數群組中設定下列參數來執行此操作：

- `server_audit_logging` = 1
- `server_audit_events` = CONNECT

以下是失敗登入之稽核日誌的擷取。

```
1728498527380921,auora-mysql-node1,user_1,172.31.49.222,147189,0,FAILED_CONNECT,,,1045
1728498527380940,auora-mysql-node1,user_1,172.31.49.222,147189,0,DISCONNECT,,,0
```

其中：

- 1728498527380921 – 失敗登入發生時的 epoch 時間戳記
- `aurora-mysql-node1` – 連線失敗之 Aurora MySQL 叢集節點的執行個體識別符
- `user_1` – 登入失敗的資料庫使用者名稱
- `172.31.49.222` – 建立連線之用戶端的私有 IP 地址
- 147189 – 失敗登入的連線 ID

- FAILED_CONNECT – 表示連線失敗。
- 1045 – 傳回碼。非零值表示錯誤。在此情況下，1045 對應至存取遭拒。

如需詳細資訊，請參閱 MySQL 文件中的伺服器錯誤代碼和用戶端錯誤代碼。

您也可以檢查 Aurora MySQL 錯誤日誌是否有任何相關的錯誤訊息，例如：

```
2024-10-09T19:26:59.310443Z 220 [Note] [MY-010926] [Server] Access denied for user
 'user_1'@'172.31.49.222' (using password: YES) (sql_authentication.cc:1502)
```

範例 2：針對異常用戶端中斷連線進行疑難排解

您可以從 Performance Insights 或使用下列命令來追蹤異常用戶端中斷連線的數量。

```
mysql> show global status like 'aborted_clients';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| Aborted_clients | 9     |
+-----------------+-------+
1 row in set (0.01 sec)
```

如果 Aborted_clients 的數量隨著時間增加，則應用程式無法正確關閉與資料庫的連線。如果連線未正確關閉，可能會導致資源洩漏和潛在的效能問題。不必要地讓連線保持開啟可能會耗用系統資源，例如記憶體和檔案描述項，最終可能導致應用程式或伺服器變得無回應或重新啟動。

您可以使用下列查詢來識別未正確關閉連線的帳戶。它會擷取使用者帳戶名稱、使用者連線的主機、未關閉的連線數，以及未關閉的連線百分比。

```
SELECT
    ess.user,
    ess.host,
    (a.total_connections - a.current_connections) - ess.count_star AS not_closed,
    (((a.total_connections - a.current_connections) - ess.count_star) * 100) /
 (a.total_connections - a.current_connections) AS pct_not_closed
FROM
    performance_schema.events_statements_summary_by_account_by_event_name AS ess
    JOIN performance_schema.accounts AS a ON (ess.user = a.user AND ess.host = a.host)
WHERE
    ess.event_name = 'statement/com/quit'
```

```
    AND (a.total_connections - a.current_connections) > ess.count_star;

+----------+---------------+------------+----------------+
| user     | host          | not_closed | pct_not_closed |
+----------+---------------+------------+----------------+
| user1    | 172.31.49.222 |          1 |        33.3333 |
| user1    | 172.31.93.250 |       1024 |        12.1021 |
| user2    | 172.31.93.250 |         10 |        12.8551 |
+----------+---------------+------------+----------------+
3 rows in set (0.00 sec)
```

識別未關閉連線的使用者帳戶和主機之後，您可以繼續檢查未正常關閉連線的程式碼。

例如，使用 Python 中的 MySQL 連接器，使用連線物件的 `close()` 方法來關閉連線。以下是建立資料庫連線、執行查詢及關閉連線的範例函數：

```
import mysql.connector

def execute_query(query):
    # Establish a connection to the database
    connection = mysql.connector.connect(
        host="your_host",
        user="your_username",
        password="your_password",
        database="your_database"
    )

    try:
        # Create a cursor object
        cursor = connection.cursor()

        # Execute the query
        cursor.execute(query)

        # Fetch and process the results
        results = cursor.fetchall()
        for row in results:
            print(row)

    finally:
        # Close the cursor and connection
        cursor.close()
        connection.close()
```

在此範例中，會在 `finally` 區塊中呼叫 `connection.close()` 方法，以確保連線已關閉，無論是否發生例外狀況。

範例 3：對 IAM 失敗的連線嘗試進行故障診斷

與 AWS Identity and Access Management (IAM) 使用者的連線可能會失敗，原因如下：

- 不正確的 IAM 政策組態
- 過期的安全登入資料
- 網路連線問題
- 資料庫許可不相符

若要疑難排解這些身分驗證錯誤，請在 Amazon Relational Database Service (RDS) 或 Aurora 資料庫中啟用`iam-db-auth-error`日誌匯出功能。這可讓您在 Amazon RDS 或 Amazon Aurora 叢集的 CloudWatch Log 群組中檢視詳細的身分驗證錯誤訊息。

啟用後，您可以檢閱這些日誌，以識別和解決 IAM 身分驗證失敗的特定原因。

例如：

```
2025-09-22T12:02:30,806 [ERROR] Failed to authorize the connection request for
 user 'user_1' due to an internal IAM DB Auth error. (Status Code: 500, Error Code:
 InternalError)
```

以及

```
2025-09-22T12:02:51,954 [ERROR] Failed to authenticate the connection request for user
 'user_2' because the provided token is malformed or otherwise invalid. (Status Code:
 400, Error Code: InvalidToken)
```

如需疑難排解指引，請參閱 IAM 資料庫身分驗證的 Aurora 疑難排解指南。

針對 Aurora MySQL 資料庫的查詢效能進行故障診斷

MySQL 透過系統變數提供查詢最佳化工具控制，這些變數會影響查詢計劃的評估方式、可切換最佳化、最佳化工具與索引提示，以及最佳化工具成本模型。這些資料點不僅可以比較不同的 MySQL 環境，還可以比較先前的查詢執行計劃與目前的執行計劃，以及了解 MySQL 查詢在任何時間點的整體執行。

查詢效能取決於許多因素，包括執行計劃、資料表結構描述和大小、統計資料、資源、索引和參數組態。查詢調校需要識別瓶頸並最佳化執行路徑。

- 尋找查詢的執行計劃，並檢查查詢是否使用適當的索引。您可以使用 EXPLAIN 並檢閱每個計劃的詳細資訊，以最佳化查詢。
- Aurora MySQL 第 3 版 (與 MySQL 8.0 Community Edition 相容) 使用 EXPLAIN ANALYZE 陳述式。EXPLAIN ANALYZE 陳述式是一種分析工具，可顯示 MySQL 在查詢上花費的時間和原因。透過 EXPLAIN ANALYZE，Aurora MySQL 會規劃、準備和執行查詢，同時計數資料列並測量在執行計劃的各個點所花費的時間。查詢完成時，EXPLAIN ANALYZE 會列印計劃及其測量結果，而不是查詢結果。
- 使用 ANALYZE 陳述式來更新您的結構描述統計資料。由於統計資料過時，查詢最佳化工具有時可以選擇不佳的執行計劃。這可能會導致查詢效能不佳，因為資料表和索引的基數預估不正確。innodb_table_stats 資料表的 `last_update` 欄會顯示您的結構描述統計資料上次更新的時間，這是「過時」的良好指標。
- 可能會發生其他問題，例如未考慮資料表基數的資料分佈扭曲。如需詳細資訊，請參閱 MySQL 文件中的估算 InnoDB 資料表的 ANALYZE TABLE 複雜性和 MySQL 中的直方圖統計資料。

了解查詢所花費的時間

以下是判斷查詢所花費時間的方法：

- 分析
- 效能結構描述
- 查詢最佳化工具

分析

根據預設，分析已停用。啟用分析，然後執行緩慢查詢並檢閱其設定檔。

```
SET profiling = 1;
```

```
Run your query.
SHOW PROFILE;
```

1. 識別花費最多時間的階段。根據 MySQL 文件中的一般執行緒狀態，讀取和處理 SELECT 陳述式的資料列通常是指定查詢生命週期中執行時間最長的狀態。您可以使用 EXPLAIN 陳述式來了解 MySQL 如何執行此查詢。
2. 檢閱緩慢查詢日誌以評估 rows_examined 和 rows_sent，以確保工作負載在每個環境中都類似。如需詳細資訊，請參閱記錄 Aurora MySQL 資料庫。
3. 針對屬於已識別查詢一部分的資料表執行下列命令：

   ```
   SHOW TABLE STATUS\G;
   ```

4. 在每個環境上執行查詢之前和之後擷取下列輸出：

   ```
   SHOW GLOBAL STATUS;
   ```

5. 在每個環境上執行下列命令，以查看是否有任何其他查詢/工作階段會影響此範例查詢的效能。

   ```
   SHOW FULL PROCESSLIST;

   SHOW ENGINE INNODB STATUS\G;
   ```

   有時，當伺服器上的資源忙碌時，它會影響伺服器上的所有其他操作，包括查詢。您也可以在執行查詢或設定 cron 任務時定期擷取資訊，以有用的間隔擷取資訊。

效能結構描述

效能結構描述提供有關伺服器執行時期效能的實用資訊，同時對該效能的影響最小。這與提供資料庫執行個體相關結構描述資訊的 information_schema 不同。如需詳細資訊，請參閱Aurora MySQL 上 Performance Insights 的效能結構描述概觀。

查詢最佳化工具追蹤

若要了解為何選擇特定查詢計劃執行，您可以設定 optimizer_trace 來存取 MySQL 查詢最佳化工具。

執行最佳化工具追蹤，以顯示最佳化工具及其選擇可用的所有路徑的廣泛資訊。

```
SET SESSION OPTIMIZER_TRACE="enabled=on";
SET optimizer_trace_offset=-5, optimizer_trace_limit=5;

-- Run your query.
```

```
SELECT * FROM table WHERE x = 1 AND y = 'A';

-- After the query completes:
SELECT * FROM information_schema.OPTIMIZER_TRACE;
SET SESSION OPTIMIZER_TRACE="enabled=off";
```

檢閱查詢最佳化工具設定

相較於 Aurora MySQL 第 2 版 (與 MySQL 5.7 Community Edition 相容)，Aurora MySQL 第 3 版 (與 MySQL 8.0 Community Edition 相容) 有許多最佳化工具相關的變更。如果您有 optimizer_switch 的一些自訂值，建議您檢閱預設值的差異，並設定最適合工作負載的 optimizer_switch 值。我們也建議您測試 Aurora MySQL 第 3 版可用的選項，以檢查您的查詢如何執行。

> **Note**
>
> Aurora MySQL 第 3 版針對 innodb_stats_persistent_sample_pages 參數使用社群預設值 20。

您可以使用下列命令來顯示 optimizer_switch 值：

```
SELECT @@optimizer_switch\G;
```

下表顯示 Aurora MySQL 第 2 版和第 3 版的預設 optimizer_switch 值。

設定	Aurora MySQL 第 2 版	Aurora MySQL 第 3 版
batched_key_access	off	off
block_nested_loop	on	on
condition_fanout_filter	on	on
derived_condition_pushdown	–	on
derived_merge	on	on
duplicateweedout	on	on
engine_condition_pushdown	on	on

設定	Aurora MySQL 第 2 版	Aurora MySQL 第 3 版
firstmatch	on	on
hash_join	off	on
hash_join_cost_based	on	–
hypergraph_optimizer	–	off
index_condition_pushdown	on	on
index_merge	on	on
index_merge_intersection	on	on
index_merge_sort_union	on	on
index_merge_union	on	on
loosescan	on	on
materialization	on	on
mrr	on	on
mrr_cost_based	on	on
prefer_ordering_index	on	on
semijoin	on	on
skip_scan	–	on
subquery_materialization_cost_based	on	on
subquery_to_derived	–	off
use_index_extensions	on	on
use_invisible_indexes	–	off

如需詳細資訊，請參閱 MySQL 文件中的切換最佳化 (MySQL 5.7) 和切換最佳化 (MySQL 8.0)。

Amazon Aurora MySQL 參考

此參考包括 Aurora MySQL 參數、狀態變數和一般 SQL 擴充功能或社群 MySQL 資料庫引擎差異的相關資訊。

主題

- Aurora MySQL 組態參數
- Aurora MySQL 全域狀態變數
- Aurora MySQL 等待事件
- Aurora MySQL 執行緒狀態
- Aurora MySQL 隔離層級
- Aurora MySQL 提示
- Aurora MySQL 預存程序參考
- Aurora MySQL 特定的 information_schema 資料表

Aurora MySQL 組態參數

您可以採用管理其他 Amazon RDS 資料庫執行個體的相同方式來管理您的 Amazon Aurora MySQL 資料庫叢集，方法是在資料庫參數群組中使用參數。Amazon Aurora 與其他資料庫引擎的差異在於，您會有包含多個資料庫執行個體的資料庫叢集。因此，您用來管理 Aurora MySQL 資料庫叢集的部分參數會套用至整個叢集。其他參數只套用至資料庫叢集的特定資料庫執行個體。

若要管理叢集層級參數，請使用資料庫叢集參數群組。若要管理執行個體層級參數，請使用資料庫參數群組。每個 Aurora MySQL 資料庫叢集中的資料庫執行個體都與 MySQL 資料庫引擎相容。不過，您可以在叢集層級套用一些 MySQL 資料庫引擎參數，而您可使用資料庫叢集參數群組來管理這些參數。您無法在 Aurora 資料庫叢集的執行個體資料庫參數群組中找到叢集層級參數。本主題稍後將提供叢集層級參數清單。

您可以使用 AWS 管理主控台、AWS CLI 或 Amazon RDS API 同時管理叢集層級和執行個體層級參數。您會對管理叢集層級參數和執行個體層級參數使用不同的命令。例如，您可以使用 modify-db-cluster-parameter-group CLI 命令來管理資料庫叢集參數群組中的叢集層級參數。您可以使用 modify-db-parameter-group CLI 命令，以在資料庫叢集中的資料庫執行個體中，管理資料庫參數群組中的執行個體層級參數。

您可以在主控台中，或使用 CLI 或 RDS API 來檢視叢集層級和執行個體層級參數。例如，您可以使用 modify-db-cluster-parameter-group AWS CLI 命令來檢視資料庫叢集參數群組中的叢集層級參數。您

可以使用 modify-db-parameter-group CLI 命令，以在資料庫叢集中的資料庫執行個體中，檢視資料庫參數群組中的執行個體層級參數。

> **Note**
>
> 每個預設參數群組包含參數群組中所有參數的預設值。如果參數具有此值的「引擎預設值」，請參閱特定版本的 MySQL 或 PostgreSQL 文件以取得實際預設值。
> 除非另有說明，否則下表所列參數對 Aurora MySQL 第 2 版和第 3 版有效。

如需資料庫參數群組的詳細資訊，請參閱 Amazon Aurora 的參數群組。如需 Aurora Serverless v1 叢集的規則與限制，請參閱 Aurora Serverless v1 的參數群組。

主題

- 叢集層級參數
- 執行個體層級參數
- 不適用於 Aurora MySQL 的 MySQL 參數

叢集層級參數

下表顯示套用至整個 Aurora MySQL 資料庫叢集的所有參數。

參數名稱	可修改	備註
aurora_binlog_read_buffer_size	是	只會影響使用二進位日誌 (binlog) 複寫的叢集。如需 binlog 複寫的資訊，請參閱 Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。已從 Aurora MySQL 第 3 版中移除。
aurora_binlog_replication_max_yield_seconds	是	只會影響使用二進位日誌 (binlog) 複寫的叢集。如需 binlog 複寫的資訊，請參閱 Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。

參數名稱	可修改	備註
aurora_binlog_replication_sec_index_parallel_workers	是	設定在複寫具有多個次要索引的大型資料表的交易時，可用於套用次要索引變更的平行執行緒總數。依預設，參數會設為 0 (停用)。 此參數適用於 Aurora MySQL 3.06 版及更新版本。如需更多詳細資訊，請參閱最佳化 Aurora MySQL 的二進位日誌複寫。
aurora_binlog_use_large_read_buffer	是	只會影響使用二進位日誌 (binlog) 複寫的叢集。如需 binlog 複寫的資訊，請參閱 Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。已從 Aurora MySQL 第 3 版中移除。
aurora_disable_hash_join	是	將此參數設為 ON，以在 Aurora MySQL 2.09 或更新版本中關閉雜湊聯結最佳化功能。第 3 版不支援它。如需更多詳細資訊，請參閱 Amazon Aurora MySQL 的平行查詢。
aurora_enable_replica_log_compression	是	如需更多詳細資訊，請參閱 Amazon Aurora MySQL 複寫的效能考量。不套用到屬於 Aurora 全球資料庫的叢集。已從 Aurora MySQL 第 3 版中移除。
aurora_enable_repl_bin_log_filtering	是	如需更多詳細資訊，請參閱 Amazon Aurora MySQL 複寫的效能考量。不套用到屬於 Aurora 全球資料庫的叢集。已從 Aurora MySQL 第 3 版中移除。
aurora_enable_staggered_replica_restart	是	此設定可在 Aurora MySQL 版本 3 中使用，但未使用。

參數名稱	可修改	備註
aurora_enable_zdr	是	此設定在 Aurora MySQL 2.10 及更高版本中預設為開啟。
aurora_in_memory_relaylog	是	設定內部記憶體轉送日誌模式。您可以在 binlog 複本上使用此功能來改善二進位日誌複寫效能。若要關閉此功能，請將參數設定為 OFF。若要開啟此功能，請將參數設定為 ON。
aurora_enhanced_binlog	是	將此參數的值設定為 1，以在 Aurora MySQL 3.03.1 版及更新版本中開啟增強型 Binlog。如需更多詳細資訊，請參閱設定 Aurora MySQL 的增強型 binlog。
aurora_full_double_precision_in_json	是	設定此參數的值，以完全精確度在 JSON 文件中啟用浮點數的剖析。
aurora_jemalloc_background_thread	是	使用此參數可啟用背景執行緒來執行記憶體維護操作。允許的值為 0 (停用) 和 1 (啟用)。預設值為 0。 此參數適用於 Aurora MySQL 3.05 版及更新版本。
aurora_jemalloc_dirty_decay_ms	是	使用此參數可保留釋放的記憶體一段特定時間 (以毫秒為單位)。保留記憶體可讓您更快速重複使用。允許的值為 0–18446744073709551615 。預設值為 10000 (10 秒)。 您可以使用較短的延遲來協助避免記憶體不足的問題，但代價是效能較慢。 此參數適用於 Aurora MySQL 3.05 版及更新版本。

參數名稱	可修改	備註
aurora_jemalloc_tcache_enabled	是	使用此參數可在執行緒本機快取中提供小型記憶體請求 (最多 32 KiB)，繞過記憶體區域。允許的值為 0 (停用) 和 1 (啟用)。預設值為 1。 此參數適用於 Aurora MySQL 3.05 版及更新版本。
aurora_load_from_s3_role	是	如需更多詳細資訊，請參閱 從 Amazon S3 儲存貯體中的文字檔案將資料載入 Amazon Aurora MySQL 資料庫叢集。目前不適用於 Aurora MySQL 第 3 版。請使用 aws_default_s3_role 。
aurora_mask_password_hashes_type	是	此設定在 Aurora MySQL 2.11 及更高版本中預設為開啟。 使用此設定可在慢速查詢和稽核日誌中遮罩 Aurora MySQL 密碼雜湊。有效值為 0 和 1 (預設值)。設為 1 時，密碼會記錄為 <secret>。設為 0 時，密碼會記錄為雜湊 (#) 值。
aurora_select_into_s3_role	是	如需更多詳細資訊，請參閱 將來自 Amazon Aurora MySQL 資料庫叢集的資料儲存至 Amazon S3 儲存貯體中的文字檔案。目前不適用於 Aurora MySQL 第 3 版。請使用 aws_default_s3_role 。

參數名稱	可修改	備註
authentication_kerberos_caseins_cmp	是	控制 authentication_kerberos 外掛程式的不區分大小寫使用者名稱比較。將其設為 true 進行不區分大小寫的比較。根據預設，使用區分大小寫的比較 (false)。如需更多詳細資訊，請參閱針對 Aurora MySQL 使用 Kerberos 身分驗證。 此參數適用於 Aurora MySQL 3.03 版及更新版本。
auto_increment_increment	是	無
auto_increment_offset	是	無
aws_default_lambda_role	是	如需更多詳細資訊，請參閱從 Amazon Aurora MySQL 資料庫叢集叫用 Lambda 函式。
aws_default_s3_role	是	從資料庫叢集呼叫 LOAD DATA FROM S3、LOAD XML FROM S3 或 SELECT INTO OUTFILE S3 陳述式時使用。 在 Aurora MySQL 第 2 版中，如果未針對適當陳述式的 aurora_load_from_s3_role 或 aurora_select_into_s3_role 指定 IAM 角色，則會使用此參數中指定的 IAM 角色。 在 Aurora MySQL 第 3 版中，一律使用針對此參數指定的 IAM 角色。 如需更多詳細資訊，請參閱將 IAM 角色與 Amazon Aurora MySQL 資料庫叢集建立關聯。

參數名稱	可修改	備註
binlog_backup	是	將此參數的值設定為 0，以在 Aurora MySQL 3.03.1 版及更新版本中開啟增強型 Binlog。僅在使用增強型 Binlog 時，才能關閉此參數。如需更多詳細資訊，請參閱 設定 Aurora MySQL 的增強型 binlog。
binlog_checksum	是	如果未設定此參數，AWS CLI 和 RDS API 會報告 None 的值。在這種情況下，Aurora MySQL 會使用引擎預設值，即 CRC32。這與 NONE 的明確設定不同，該設定會關閉檢查總和。
binlog-do-db	是	此參數適用於 Aurora MySQL 第 3 版。
binlog_format	是	如需更多詳細資訊，請參閱 Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。
binlog_group_commit_sync_delay	是	此參數適用於 Aurora MySQL 第 3 版。
binlog_group_commit_sync_no_delay_count	是	此參數適用於 Aurora MySQL 第 3 版。
binlog-ignore-db	是	此參數適用於 Aurora MySQL 第 3 版。
binlog_replication_globaldb	是	將此參數的值設定為 0，以在 Aurora MySQL 3.03.1 版及更新版本中開啟增強型 Binlog。僅在使用增強型 Binlog 時，才能關閉此參數。如需更多詳細資訊，請參閱 設定 Aurora MySQL 的增強型 binlog。
binlog_row_image	否	無
binlog_row_metadata	是	此參數適用於 Aurora MySQL 第 3 版。

參數名稱	可修改	備註
binlog_row_value_options	是	此參數適用於 Aurora MySQL 第 3 版。
binlog_rows_query_log_events	是	無
binlog_transaction_compression	是	此參數適用於 Aurora MySQL 第 3 版。
binlog_transaction_compression_level_zstd	是	此參數適用於 Aurora MySQL 第 3 版。
binlog_transaction_dependency_history_size	是	此參數會設定儲存在記憶體中的資料列雜湊數目上限，並用於查詢上次修改指定資料列的交易。達到此雜湊數目之後，會清除歷史記錄。 此參數適用於 Aurora MySQL 2.12 版及更新版本以及 3 版。
binlog_transaction_dependency_tracking	是	此參數適用於 Aurora MySQL 第 3 版。
character-set-client-handshake	是	無
character_set_client	是	無
character_set_connection	是	無
character_set_database	是	預設資料庫使用的字元集。預設值為 utf8mb4。
character_set_filesystem	是	無
character_set_results	是	無
character_set_server	是	無

參數名稱	可修改	備註
collation_connection	是	無
collation_server	是	無
completion_type	是	無
default_storage_engine	否	Aurora MySQL 叢集使用 InnoDB 儲存引擎來存放您的所有資料。
enforce_gtid_consistency	有時候	在 Aurora MySQL 第 2 版及更新的版本中可修改。
event_scheduler	是	指示事件排程器的狀態。 僅可在 Aurora MySQL 第 3 版中的叢集層級進行修改。
gtid-mode	有時候	在 Aurora MySQL 第 2 版及更新的版本中可修改。
information_schema_stats_expiry	是	MySQL 資料庫伺服器從儲存引擎擷取資料並取代快取中的資料之後的秒數。允許的值為 0–31536000。 此參數適用於 Aurora MySQL 第 3 版。

參數名稱	可修改	備註
init_connect	是	要由伺服器針對每個連線的用戶端執行的命令。針對設定使用雙引號 (")，以避免連線失敗，例如： `SET optimizer_switch="hash_join =off"` 在 Aurora MySQL 第 3 版中，此參數不適用於具有 CONNECTION_ADMIN 權限的使用者。這包括 Aurora 主要使用者。如需更多詳細資訊，請參閱 角色型權限模型。
innodb_adaptive_hash_index	是	您可以在 Aurora MySQL 第 2 版和第 3 版中的資料庫叢集層級修改此參數。 讀取器資料庫執行個體不支援自適應雜湊索引。
innodb_aurora_instant_alter_column_allowed	是	控制 INSTANT 演算法是否可用於全域層級的 ALTER COLUMN 操作。允許值如下： • 0 - INSTANT 演算法不允許用於 ALTER COLUMN 操作 (OFF)。還原至其他演算法。 • 1 - INSTANT 演算法允許用於 ALTER COLUMN 操作 (ON)。這是預設值。 如需詳細資訊，請參閱 MySQL 文件中的欄操作。 此參數適用於 Aurora MySQL 3.05 版及更新版本。
innodb_autoinc_lock_mode	是	無

參數名稱	可修改	備註
innodb_checksums	否	已從 Aurora MySQL 第 3 版中移除。
innodb_cmp_per_index_enabled	是	無
innodb_commit_concurrency	是	無
innodb_data_home_dir	否	Aurora MySQL 會在您無法直接存取檔案系統時使用受管執行個體。
innodb_deadlock_detect	是	此選項用於停用 Aurora MySQL 2.11 和更新版本以及 3 版中的死結偵測。 在高度並行系統上，當多個執行緒等待同一個結時，死結偵測可能會導致速度變慢。如需此參數的詳細資訊，請參閱 MySQL 文件。
innodb_default_row_format	是	此參數定義 InnoDB 資料表的預設資料列格式 (包括使用者建立的 InnoDB 暫存資料表)。它適用於 Aurora MySQL 第 2 版和第 3 版。 其值可以是 DYNAMIC、COMPACT 或 REDUNDANT。 。
innodb_file_per_table	是	此參數會影響資料表儲存的組織方式。如需更多詳細資訊，請參閱 儲存體擴展。

參數名稱	可修改	備註
innodb_flush_log_at_trx_commit	是	建議您使用預設值 1。 在 Aurora MySQL 第 3 版中，您必須先將 innodb_trx_commit_allow_data_loss 的值設定為 1，才能將此參數設定為 1 以外的值。 如需更多詳細資訊，請參閱 設定日誌緩衝區的排清頻率。
innodb_ft_max_token_size	是	無
innodb_ft_min_token_size	是	無
innodb_ft_num_word_optimize	是	無
innodb_ft_sort_pll_degree	是	無
innodb_online_alter_log_max_size	是	無
innodb_optimize_fulltext_only	是	無
innodb_page_size	否	無
innodb_print_all_deadlocks	是	開啟時，InnoDB 所有死結的資訊皆會記錄在 Aurora MySQL 錯誤日誌。如需更多詳細資訊，請參閱 減少和疑難排解 Aurora MySQL 的死結情況。
innodb_purge_batch_size	是	無
innodb_purge_threads	是	無
innodb_rollback_on_timeout	是	無

參數名稱	可修改	備註
innodb_rollback_segments	是	無
innodb_spin_wait_delay	是	無
innodb_strict_mode	是	無
innodb_support_xa	是	已從 Aurora MySQL 第 3 版中移除。
innodb_sync_array_size	是	無
innodb_sync_spin_loops	是	無
innodb_stats_include_delete_marked	是	啟用此參數時，InnoDB 會在計算永久性最佳化程式統計資料時包含已刪除標記的記錄。 此參數適用於 Aurora MySQL 2.12 版及更新版本以及 3 版。
innodb_table_locks	是	無
innodb_trx_commit_allow_data_loss	是	在 Aurora MySQL 第 3 版中，將此參數的值設定為 1，以便您可以變更 innodb_flush_log_at_trx_commit 的值。 innodb_trx_commit_allow_data_loss 的預設值為 0。 如需更多詳細資訊，請參閱 設定日誌緩衝區的排清頻率。
innodb_undo_directory	否	Aurora MySQL 會在您無法直接存取檔案系統時使用受管執行個體。

參數名稱	可修改	備註
internal_tmp_disk_storage_engine	是	控制內部暫存資料表所要使用的記憶體內儲存引擎。允許的值為 INNODB 和 MYISAM。 此參數適用於 Aurora MySQL 第 2 版。
internal_tmp_mem_storage_engine	是	控制內部暫存資料表所要使用的記憶體內儲存引擎。允許的值為 MEMORY 和 TempTable 。 此參數適用於 Aurora MySQL 第 3 版。
key_buffer_size	是	MyISAM 資料表的金鑰快取。如需詳細資訊，請參閱 keycache->cache_lock mutex。
lc_time_names	是	無
log_error_suppression_list	是	指定未記錄在 MySQL 錯誤日誌中的錯誤代碼清單。這可讓您忽略某些非關鍵錯誤情況，以協助保持錯誤日誌的乾淨。如需詳細資訊，請參閱 MySQL 文件中的 log_error_suppression_list。 此參數適用於 Aurora MySQL 3.03 版及更新版本。
low_priority_updates	是	INSERT、UPDATE、DELETE、和 LOCK TABLE WRITE 操作會等待到沒有擱置的 SELECT 操作為止。此參數只會影響僅使用資料表層級鎖定 (MyISAM、MEMORY、MERGE) 的儲存引擎。 此參數適用於 Aurora MySQL 第 3 版。

參數名稱	可修改	備註
lower_case_table_names	是 (Aurora MySQL 第 2 版) 僅在建立叢集時 (Aurora MySQL 第 3 版)	在 Aurora MySQL 2.10 及更新的 2.x 版中，請務必在變更此設定並重新啟動寫入器執行個體之後，重新啟動所有讀取器執行個體。如需詳細資訊，請參閱 使用讀取可用性功能重新啟動 Aurora 叢集。 在 Aurora MySQL 第 3 版中，此參數的值會在建立叢集時永久設定。如果您對此選項使用非預設值，請在升級之前設定 Aurora MySQL 第 3 版自訂參數群組，並在建立第 3 版叢集的快照還原作業期間指定參數群組。 使用以 Aurora MySQL 為基礎的 Aurora 全域資料庫時，若啟用 lower_case_table_names 參數，即無法從 Aurora MySQL 第 2 版就地升級至第 3 版。如需詳細了解您可以使用的方法，請參閱 主要版本升級。
master-info-repository	是	已從 Aurora MySQL 第 3 版中移除。
master_verify_checksum	是	Aurora MySQL 第 2 版。在 Aurora MySQL 第 3 版中使用 source_verify_checksum 。
max_delayed_threads	是	設定要處理 INSERT DELAYED 陳述式的執行緒數量上限。 此參數適用於 Aurora MySQL 第 3 版。
max_error_count	是	要儲存用於顯示的錯誤訊息、警告和備註訊息數量上限。 此參數適用於 Aurora MySQL 第 3 版。

參數名稱	可修改	備註
max_execution_time	是	執行 SELECT 陳述式的逾時，以毫秒為單位。值可以介於 0–18446744073709551615 之間。設為 0 時，不會發生逾時。 如需詳細資訊，請參閱 MySQL 文件中的 max_execution_time。
min_examined_row_limit	是	使用此參數可防止所檢查資料列未達指定數量的查詢。
partial_revokes	否	此參數適用於 Aurora MySQL 第 3 版。
preload_buffer_size	是	預先載入索引時所配置的緩衝區大小。 此參數適用於 Aurora MySQL 第 3 版。
query_cache_type	是	已從 Aurora MySQL 第 3 版中移除。

參數名稱	可修改	備註
read_only	是	此參數開啟時，伺服器不允許任何更新，除了由複本執行緒執行的更新。 對於 Aurora MySQL 第 2 版，有效值如下： • 0 – OFF • 1 – ON • {TrueIfReplica} – ON 適用於僅供讀取複本。這是預設值。 • {TrueIfClusterReplica} – ON 適用於複本叢集，例如跨區域僅供讀取複本、Aurora 全球資料庫中的次要叢集，以及藍/綠部署。 對於 Aurora MySQL 第 3 版，有效值如下： • 0 – OFF。這是預設值。 • 1 – ON • {TrueIfClusterReplica} – ON 適用於複本叢集，例如跨區域僅供讀取複本、Aurora 全球資料庫中的次要叢集，以及藍/綠部署。 在 Aurora MySQL 第 3 版中，此參數不適用於具有 CONNECTION_ADMIN 權限的使用者。這包括 Aurora 主要使用者。如需更多詳細資訊，請參閱 角色型權限模型。
relay-log-space-limit	是	此參數適用於 Aurora MySQL 第 3 版。

參數名稱	可修改	備註
replica_parallel_type	是	此參數會啟用在所有處於準備階段的未遞交執行緒複本上平行執行，而不違反一致性。它適用於 Aurora MySQL 第 3 版。 在 Aurora MySQL 3.03.* 及較低版本中，預設值是 DATABASE。在 Aurora MySQL 3.04 及較高版本中，預設值是 LOGICAL_CLOCK。
replica_preserve_commit_order	是	此參數適用於 Aurora MySQL 第 3 版。
replica_transaction_retries	是	此參數適用於 Aurora MySQL 第 3 版。
replica_type_conversions	是	此參數會決定複本上所使用的類型轉換。允許的值為 ALL_LOSSY 、ALL_NON_LOSSY 、ALL_SIGNED 和 ALL_UNSIGNED 。如需詳細資訊，請參閱 MySQL 文件中的在來源和複本上使用不同資料表定義進行複寫。 此參數適用於 Aurora MySQL 第 3 版。
replicate-do-db	是	此參數適用於 Aurora MySQL 第 3 版。
replicate-do-table	是	此參數適用於 Aurora MySQL 第 3 版。
replicate-ignore-db	是	此參數適用於 Aurora MySQL 第 3 版。
replicate-ignore-table	是	此參數適用於 Aurora MySQL 第 3 版。
replicate-wild-do-table	是	此參數適用於 Aurora MySQL 第 3 版。
replicate-wild-ignore-table	是	此參數適用於 Aurora MySQL 第 3 版。

參數名稱	可修改	備註
require_secure_transport	是	此參數適用於 Aurora MySQL 第 2 版和第 3 版。如需更多詳細資訊，請參閱 Aurora MySQL 資料庫叢集的 TLS 連線。
rpl_read_size	是	此參數適用於 Aurora MySQL 第 3 版。
server_audit_cw_upload	否	此參數在 Aurora MySQL 中已棄用。請使用 server_audit_logs_upload 。 如需更多詳細資訊，請參閱 將 Amazon Aurora MySQL 日誌發佈至 Amazon CloudWatch Logs。
server_audit_events	是	如需更多詳細資訊，請參閱 使用進階稽核與 Amazon Aurora MySQL 資料庫叢集搭配。
server_audit_excl_users	是	如需更多詳細資訊，請參閱 使用進階稽核與 Amazon Aurora MySQL 資料庫叢集搭配。
server_audit_incl_users	是	如需更多詳細資訊，請參閱 使用進階稽核與 Amazon Aurora MySQL 資料庫叢集搭配。
server_audit_logging	是	如需上傳日誌到 Amazon CloudWatch Logs 的說明，請參閱 將 Amazon Aurora MySQL 日誌發佈至 Amazon CloudWatch Logs。

參數名稱	可修改	備註
server_audit_logs_upload	是	您可以透過啟用進階稽核並將此參數設定為 1，將稽核日誌發佈至 CloudWatch Logs。server_audit_logs_upload 參數的預設值為 0。 如需更多詳細資訊，請參閱 將 Amazon Aurora MySQL 日誌發佈至 Amazon CloudWatch Logs。
server_id	否	無
skip-character-set-client-handshake	是	無
skip_name_resolve	否	無
slave-skip-errors	是	僅適用於 Aurora MySQL 第 2 版叢集，與 MySQL 5.7 相容。
source_verify_checksum	是	此參數適用於 Aurora MySQL 第 3 版。
sync_frm	是	已從 Aurora MySQL 第 3 版中移除。
thread_cache_size	是	要快取的執行緒數目。此參數適用於 Aurora MySQL 第 2 版和第 3 版。
time_zone	是	Aurora 資料庫叢集的時區預設為國際標準時間 (UTC)。您可以將您的資料庫叢集中執行個體的時區改為設定成應用程式的本機時區。如需更多詳細資訊，請參閱 Amazon Aurora 資料庫叢集的本機時區。
tls_version	是	如需更多詳細資訊，請參閱 適用於 Aurora MySQL 的 TLS 版本。

執行個體層級參數

下表顯示套用至 Aurora MySQL 資料庫叢集中特定資料庫執行個體的所有參數。

參數名稱	可修改	備註
activate_all_roles_on_login	是	此參數適用於 Aurora MySQL 第 3 版。
allow-suspicious-udfs	否	無
aurora_disable_hash_join	是	將此參數設為 ON，以在 Aurora MySQL 2.09 或更新版本中關閉雜湊聯結最佳化功能。第 3 版不支援它。如需更多詳細資訊，請參閱 Amazon Aurora MySQL 的平行查詢。
aurora_lab_mode	是	如需更多詳細資訊，請參閱 Amazon Aurora MySQL 實驗室模式。已從 Aurora MySQL 第 3 版中移除。
aurora_oom_response	是	Aurora MySQL 第 2 版和第 3 版支援此參數。如需更多詳細資訊，請參閱 故障診斷 Aurora MySQL 資料庫記憶體不足的問題。
aurora_parallel_query	是	設為 ON 以在 Aurora MySQL 2.09 版或更新版本中開啟平行查詢。舊 aurora_pq 參數不在這些版本中使用。如需更多詳細資訊，請參閱 Amazon Aurora MySQL 的平行查詢。
aurora_pq	是	設定為 OFF 以關閉 2.09 之前 Aurora MySQL 版本中特定資料庫執行個體的平行查詢。在 2.09 或更新版本中，請改用 aurora_parallel_query 開啟和關閉平行查詢。如需更多詳細資訊，請參閱 Amazon Aurora MySQL 的平行查詢。

參數名稱	可修改	備註
aurora_read_replica_read_committed	是	對 Aurora 複本啟用 READ COMMITTED 隔離層級，並變更隔離行為，以縮短長時間執行的查詢帶來的清除延遲時間。只在您了解行為變更及其如何影響查詢結果時，才啟用此設定。例如，此設定使用的隔離不像 MySQL 預設值那麼嚴格。啟用此設定時，長時間執行的查詢可能看到同一列有多個副本，因為在查詢執行的同時，Aurora 重組資料表資料。如需更多詳細資訊，請參閱 Aurora MySQL 隔離層級。
aurora_tmptable_enable_per_table_limit	是	確定 tmp_table_size 參數是否控制 Aurora MySQL 3.04 版及更新版本中的 TempTable 儲存體引擎建立記憶體內暫時資料表的大小上限。 如需更多詳細資訊，請參閱 限制記憶體內部暫存資料表的大小。
aurora_use_vector_instructions	是	啟用此參數時，Aurora MySQL 會使用現代 CPU 提供的最佳化向量處理指令來改善 I/O 密集型工作負載的效能。 此設定在 Aurora MySQL 3.05 及更高版本中預設為啟用。
autocommit	是	無
automatic_sp_privileges	是	無
back_log	是	無
basedir	否	Aurora MySQL 會在您無法直接存取檔案系統時使用受管執行個體。
binlog_cache_size	是	無

參數名稱	可修改	備註
binlog_max_flush_queue_time	是	無
binlog_order_commits	是	無
binlog_stmt_cache_size	是	無
binlog_transaction_compression	是	此參數適用於 Aurora MySQL 第 3 版。
binlog_transaction_compression_level_zstd	是	此參數適用於 Aurora MySQL 第 3 版。
bulk_insert_buffer_size	是	無
concurrent_insert	是	無
connect_timeout	是	無
core-file	否	Aurora MySQL 會在您無法直接存取檔案系統時使用受管執行個體。
datadir	否	Aurora MySQL 會在您無法直接存取檔案系統時使用受管執行個體。
default_authentication_plugin	否	此參數適用於 Aurora MySQL 第 3 版。
default_time_zone	否	無
default_tmp_storage_engine	是	暫存資料表的預設儲存引擎。
default_week_format	是	無
delay_key_write	是	無
delayed_insert_limit	是	無
delayed_insert_timeout	是	無

參數名稱	可修改	備註
delayed_queue_size	是	無
div_precision_increment	是	無
end_markers_in_json	是	無
eq_range_index_dive_limit	是	無
event_scheduler	有時候	指示事件排程器的狀態。 僅可在 Aurora MySQL 第 3 版中的叢集層級進行修改。
explicit_defaults_for_timestamp	是	無
flush	否	無
flush_time	是	無
ft_boolean_syntax	否	無
ft_max_word_len	是	無
ft_min_word_len	是	無
ft_query_expansion_limit	是	無
ft_stopword_file	是	無
general_log	是	如需上傳日誌到 CloudWatch Logs 的說明，請參閱 將 Amazon Aurora MySQL 日誌發佈至 Amazon CloudWatch Logs。
general_log_file	否	Aurora MySQL 會在您無法直接存取檔案系統時使用受管執行個體。
group_concat_max_len	是	無

參數名稱	可修改	備註
host_cache_size	是	無
init_connect	是	要由伺服器針對每個連線的用戶端執行的命令。針對設定使用雙引號 (")，以避免連線失敗，例如： `SET optimizer_switch="hash_join =off"` 在 Aurora MySQL 第 3 版中，此參數不適用於具有 CONNECTION_ADMIN 權限的使用者，也不適用於 Aurora 主要使用者。如需更多詳細資訊，請參閱 角色型權限模型。
innodb_adaptive_hash_index	是	您可以在 Aurora MySQL 第 2 版中的資料庫執行個體層級修改此參數。僅可在 Aurora MySQL 第 3 版中的資料庫叢集層級進行修改。 讀取器資料庫執行個體不支援自適應雜湊索引。
innodb_adaptive_max_sleep_delay	是	修改此參數不會有任何作用，因為對 Aurora 而言，innodb_thread_concurrency 永遠為 0。

參數名稱	可修改	備註
innodb_aurora_max_partitions_for_range	是	在某些無法使用持續性統計資料的情況下，您可以使用此參數來改善分割資料表上的資料列計數估算效能。 您可以將其設定為介於 0-8192 之間的值，該值會決定資料列計數估算期間所要檢查的分割區數量。預設值為 0，會使用所有分割區進行估算，與預設 MySQL 行為一致。 此參數適用於 Aurora MySQL 第 3.03.1 版及更高版本。
innodb_autoextend_increment	是	無
innodb_buffer_pool_dump_at_shutdown	否	無
innodb_buffer_pool_dump_now	否	無
innodb_buffer_pool_filename	否	無
innodb_buffer_pool_load_abort	否	無
innodb_buffer_pool_load_at_startup	否	無
innodb_buffer_pool_load_now	否	無

參數名稱	可修改	備註
innodb_buffer_pool_size	是	預設值以公式表示。如需如何在公式中計算 DBInstanceClassMemory 值的詳細資訊，請參閱 資料庫參數公式變數。
innodb_change_buffer_max_size	否	Aurora MySQL 完全不會使用 InnoDB 變更緩衝。
innodb_compression_failure_threshold_pct	是	無
innodb_compression_level	是	無
innodb_compression_pad_pct_max	是	無
innodb_concurrency_tickets	是	修改此參數不會有任何作用，因為對 Aurora 而言，innodb_thread_concurrency 永遠為 0。
innodb_deadlock_detect	是	此選項用於停用 Aurora MySQL 2.11 和更新版本以及 3 版中的死結偵測。 在高度並行系統上，當多個執行緒等待同一個結時，死結偵測可能會導致速度變慢。如需此參數的詳細資訊，請參閱 MySQL 文件。
innodb_file_format	是	已從 Aurora MySQL 第 3 版中移除。
innodb_flushing_avg_loops	否	無
innodb_force_load_corrupted	否	無
innodb_ft_aux_table	是	無
innodb_ft_cache_size	是	無

參數名稱	可修改	備註
innodb_ft_enable_stopword	是	無
innodb_ft_server_stopword_table	是	無
innodb_ft_user_stopword_table	是	無
innodb_large_prefix	是	已從 Aurora MySQL 第 3 版中移除。
innodb_lock_wait_timeout	是	無
innodb_log_compressed_pages	否	無
innodb_lru_scan_depth	是	無
innodb_max_purge_lag	是	無
innodb_max_purge_lag_delay	是	無
innodb_monitor_disable	是	無
innodb_monitor_enable	是	無
innodb_monitor_reset	是	無
innodb_monitor_reset_all	是	無
innodb_old_blocks_pct	是	無
innodb_old_blocks_time	是	無
innodb_open_files	是	無
innodb_print_all_deadlocks	是	開啟時，InnoDB 所有死結的資訊皆會記錄在 Aurora MySQL 錯誤日誌。如需更多詳細資訊，請參閱 減少和疑難排解 Aurora MySQL 的死結情況。

參數名稱	可修改	備註
innodb_random_read_ahead	是	無
innodb_read_ahead_threshold	是	無
innodb_read_io_threads	否	無
innodb_read_only	否	Aurora MySQL 會根據叢集的類型管理資料庫執行個體的唯讀和讀寫狀態。例如，佈建叢集具有一個讀寫資料庫執行個體 (primary instance (主要執行個體))，其他叢集內的任何執行個體都是唯讀 (Aurora 複本)。
innodb_replication_delay	是	無
innodb_sort_buffer_size	是	無
innodb_stats_auto_recalc	是	無
innodb_stats_method	是	無
innodb_stats_on_metadata	是	無
innodb_stats_persistent	是	無
innodb_stats_persistent_sample_pages	是	無
innodb_stats_transient_sample_pages	是	無
innodb_thread_concurrency	否	無
innodb_thread_sleep_delay	是	修改此參數不會有任何作用，因為對 Aurora 而言，innodb_thread_concurrency 永遠為 0。

參數名稱	可修改	備註
interactive_timeout	是	Aurora 會評估 interactive_timeout 和 wait_timeout 的最小值。然後使用該最小值作為逾時來結束所有閒置工作階段，包括互動式和非互動的工作階段。
internal_tmp_disk_storage_engine	是	控制內部暫存資料表所要使用的記憶體內儲存引擎。允許的值為 INNODB 和 MYISAM。 此參數適用於 Aurora MySQL 第 2 版。
internal_tmp_mem_storage_engine	有時候	控制內部暫存資料表所要使用的記憶體內儲存引擎。寫入器資料庫執行個體的允許值為 MEMORY 和 TempTable 。 對於讀取器資料庫執行個體，此參數設定為 TempTable 且無法修改。 此參數適用於 Aurora MySQL 第 3 版。
join_buffer_size	是	無
keep_files_on_create	是	無
key_buffer_size	是	MyISAM 資料表的金鑰快取。如需詳細資訊，請參閱 keycache->cache_lock mutex。
key_cache_age_threshold	是	無
key_cache_block_size	是	無
key_cache_division_limit	是	無
local_infile	是	無
lock_wait_timeout	是	無

參數名稱	可修改	備註
log-bin	否	將 binlog_format 設定為 STATEMENT 、MIXED 或 ROW，會自動將 log-bin 設定為 ON。將 binlog_format 設定為 OFF，會自動將 log-bin 設定為 OFF。如需更多詳細資訊，請參閱 Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。
log_bin_trust_function_creators	是	無
log_bin_use_v1_row_events	是	已從 Aurora MySQL 第 3 版中移除。
log_error	否	無
log_error_suppression_list	是	指定未記錄在 MySQL 錯誤日誌中的錯誤代碼清單。這可讓您忽略某些非關鍵錯誤情況，以協助保持錯誤日誌的乾淨。如需詳細資訊，請參閱 MySQL 文件中的 log_error_suppression_list。 此參數適用於 Aurora MySQL 3.03 版及更新版本。
log_output	是	無
log_queries_not_using_indexes	是	無
log_slave_updates	否	Aurora MySQL 第 2 版。在 Aurora MySQL 第 3 版中使用 log_replica_updates 。
log_replica_updates	否	Aurora MySQL 第 3 版

參數名稱	可修改	備註
log_throttle_queries_not_using_indexes	是	無
log_warnings	是	已從 Aurora MySQL 第 3 版中移除。
long_query_time	是	無
low_priority_updates	是	INSERT、UPDATE、DELETE、和 LOCK TABLE WRITE 操作會等待到沒有擱置的 SELECT 操作為止。此參數只會影響僅使用資料表層級鎖定 (MyISAM、MEMORY、MERGE) 的儲存引擎。 此參數適用於 Aurora MySQL 第 3 版。
max_allowed_packet	是	無
max_binlog_cache_size	是	無
max_binlog_size	否	無
max_binlog_stmt_cache_size	是	無
max_connect_errors	是	無
max_connections	是	預設值以公式表示。如需如何在公式中計算 DBInstanceClassMemory 值的詳細資訊，請參閱 資料庫參數公式變數。有關取決於執行個體類別的預設值，請參閱 對 Aurora MySQL 資料庫執行個體的連線數上限。
max_delayed_threads	是	設定要處理 INSERT DELAYED 陳述式的執行緒數量上限。 此參數適用於 Aurora MySQL 第 3 版。

參數名稱	可修改	備註
max_error_count	是	要儲存用於顯示的錯誤訊息、警告和備註訊息數量上限。 此參數適用於 Aurora MySQL 第 3 版。
max_execution_time	是	執行 SELECT 陳述式的逾時，以毫秒為單位。值可以介於 0–18446744073709551615 之間。設為 0 時，不會發生逾時。 如需詳細資訊，請參閱 MySQL 文件中的 max_execution_time。
max_heap_table_size	是	無
max_insert_delayed_threads	是	無
max_join_size	是	無
max_length_for_sort_data	是	已從 Aurora MySQL 第 3 版中移除。
max_prepared_stmt_count	是	無
max_seeks_for_key	是	無
max_sort_length	是	無
max_sp_recursion_depth	是	無
max_tmp_tables	是	已從 Aurora MySQL 第 3 版中移除。
max_user_connections	是	無
max_write_lock_count	是	無
metadata_locks_cache_size	是	已從 Aurora MySQL 第 3 版中移除。

參數名稱	可修改	備註
min_examined_row_limit	是	使用此參數可防止所檢查資料列未達指定數量的查詢。 此參數適用於 Aurora MySQL 第 3 版。
myisam_data_pointer_size	是	無
myisam_max_sort_file_size	是	無
myisam_mmap_size	是	無
myisam_sort_buffer_size	是	無
myisam_stats_method	是	無
myisam_use_mmap	是	無
net_buffer_length	是	無
net_read_timeout	是	無
net_retry_count	是	無
net_write_timeout	是	無
old-style-user-limits	是	無
old_passwords	是	已從 Aurora MySQL 第 3 版中移除。
optimizer_prune_level	是	無
optimizer_search_depth	是	無
optimizer_switch	是	如需使用此參數之 Aurora MySQL 功能的相關資訊，請參閱 Amazon Aurora MySQL 的最佳實務。
optimizer_trace	是	無

參數名稱	可修改	備註
optimizer_trace_features	是	無
optimizer_trace_limit	是	無
optimizer_trace_max_mem_size	是	無
optimizer_trace_offset	是	無
performance-schema-consumer-events-waits-current	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance-schema-consumer-events-waits-current 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance-schema-instrument	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance-schema-instrument 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema	是	如果來源欄設定為 Modified，則 Performance Insights 會自動管理效能結構描述。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。

參數名稱	可修改	備註
performance_schema_accounts_size	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_accounts_size。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_consumer_global_instrumentation	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_consumer_global_instrumentation。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_consumer_thread_instrumentation	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_consumer_thread_instrumentation。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_consumer_events_stages_current	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_consumer_events_stages_current。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。

參數名稱	可修改	備註
performance_schema_consumer_events_stages_history	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_consumer_events_stages_history 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_consumer_events_stages_history_long	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_consumer_events_stages_history_long 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_consumer_events_statements_current	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_consumer_events_statements_current 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_consumer_events_statements_history	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_consumer_events_statements_history 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。

參數名稱	可修改	備註
performance_schema_consumer_events_statements_history_long	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_consumer_events_statements_history_long 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_consumer_events_waits_history	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_consumer_events_waits_history 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_consumer_events_waits_history_long	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_consumer_events_waits_history_long 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_consumer_statements_digest	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_consumer_statements_digest 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。

參數名稱	可修改	備註
performance_schema_digests_size	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_digests_size 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_events_stages_history_long_size	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_events_stages_history_long_size 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_events_stages_history_size	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_events_stages_history_size 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_events_statements_history_long_size	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_events_statements_history_long_size 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。

參數名稱	可修改	備註
performance_schema_events_statements_history_size	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_events_statements_history_size 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_events_transactions_history_long_size	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_events_transactions_history_long_size 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_events_transactions_history_size	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_events_transactions_history_size 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_events_waits_history_long_size	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_events_waits_history_long_size 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。

參數名稱	可修改	備註
performance_schema_events_waits_history_size	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_events_waits_history_size 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_hosts_size	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_hosts_size 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_cond_classes	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_cond_classes 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_cond_instances	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_cond_instances 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。

參數名稱	可修改	備註
performance_schema_max_digest_length	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_digest_length 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_file_classes	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_file_classes 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_file_handles	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_file_handles 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_file_instances	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_file_instances 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。

參數名稱	可修改	備註
performance_schema_max_index_stat	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_index_stat 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_memory_classes	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_memory_classes 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_metadata_locks	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_metadata_locks 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_mutex_classes	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_mutex_classes 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。

參數名稱	可修改	備註
performance_schema_max_mutex_instances	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_mutex_instances 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_prepared_statements_instances	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_prepared_statements_instances 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_program_instances	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_program_instances 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_rwlock_classes	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_rwlock_classes 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。

參數名稱	可修改	備註
performance_schema_max_rwlock_instances	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_rwlock_instances 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_socket_classes	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_socket_classes 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_socket_instances	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_socket_instances 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_sql_text_length	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_sql_text_length 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。

參數名稱	可修改	備註
performance_schema_max_stage_classes	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_stage_classes 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_statement_classes	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_statement_classes 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_statement_stack	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_statement_stack 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_table_handles	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_table_handles 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。

參數名稱	可修改	備註
performance_schema_max_table_instances	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_table_instances 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_table_lock_stat	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_table_lock_stat 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_thread_classes	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_thread_classes 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_max_thread_instances	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_max_thread_instances 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。

參數名稱	可修改	備註
performance_schema_session_connect_attrs_size	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_session_connect_attrs_size 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_setup_actors_size	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_setup_actors_size 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_setup_objects_size	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_setup_objects_size 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。

參數名稱	可修改	備註
performance_schema_show_processlist	是	此參數決定使用那些 SHOW PROCESSLIST 方案： • 預設方案在持有全局互斥體的同時，從執行緒管理器中迴圈存取使用中的執行緒。這可能會導致效能降低，尤其是當系統繁忙時。 • SHOW PROCESSLIST 替代方案是根據 Performance Schema processlist 資料表。此方案從 Performance Schema 而不是執行緒管理器上查詢使用中的執行緒資料，並且不會導致互斥。 此參數適用於 Aurora MySQL 2.12 版及更新版本以及 3 版。 如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_show_processlist 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
performance_schema_users_size	是	如果參數 performance_schema 的來源欄設定為 Modified，則效能結構描述會使用參數 performance_schema_users_size 。如需啟用效能結構描述的相關資訊，請參閱 決定績效詳情是否管理效能結構描述。
pid_file	否	無

參數名稱	可修改	備註
plugin_dir	否	Aurora MySQL 會在您無法直接存取檔案系統時使用受管執行個體。
port	否	Aurora MySQL 會管理連線屬性，並在叢集內的所有資料庫執行個體上強制實行一致的設定。
preload_buffer_size	是	預先載入索引時所配置的緩衝區大小。 此參數適用於 Aurora MySQL 第 3 版。
profiling_history_size	是	無
query_alloc_block_size	是	無
query_cache_limit	是	已從 Aurora MySQL 第 3 版中移除。
query_cache_min_res_unit	是	已從 Aurora MySQL 第 3 版中移除。
query_cache_size	是	預設值以公式表示。如需如何在公式中計算 DBInstanceClassMemory 值的詳細資訊，請參閱 資料庫參數公式變數。 已從 Aurora MySQL 第 3 版中移除。
query_cache_type	是	已從 Aurora MySQL 第 3 版中移除。
query_cache_wlock_invalidate	是	已從 Aurora MySQL 第 3 版中移除。
query_prealloc_size	是	無
range_alloc_block_size	是	無
read_buffer_size	是	無

參數名稱	可修改	備註
read_only	是	此參數開啟時，伺服器不允許任何更新，除了由複本執行緒執行的更新。 對於 Aurora MySQL 第 2 版，有效值如下： • 0 – OFF • 1 – ON • {TrueIfReplica} – ON 適用於僅供讀取複本。這是預設值。 • {TrueIfClusterReplica} – ON 適用於複本叢集中的執行個體，例如跨區域僅供讀取複本、Aurora 全球資料庫中的次要叢集，以及藍/綠部署。 建議您使用 Aurora MySQL 第 2 版的資料庫叢集參數群組，以確保在容錯移轉時，read_only 參數有套用至新的寫入器執行個體。 **Note** 讀取器執行個體一律是唯讀，因為 Aurora MySQL 會在所有讀取器上將 innodb_read_only 設為 1。因此，在讀取器執行個體上 read_only 是備援的。 已在 Aurora MySQL 第 3 版的執行個體層級中移除。
read_rnd_buffer_size	是	無
relay-log	否	無

參數名稱	可修改	備註
relay_log_info_repository	是	已從 Aurora MySQL 第 3 版中移除。
relay_log_recovery	否	無
replica_checkpoint_group	是	Aurora MySQL 第 3 版
replica_checkpoint_period	是	Aurora MySQL 第 3 版
replica_parallel_workers	是	Aurora MySQL 第 3 版
replica_pending_jobs_size_max	是	Aurora MySQL 第 3 版
replica_skip_errors	是	Aurora MySQL 第 3 版
replica_sql_verify_checksum	是	Aurora MySQL 第 3 版
safe-user-create	是	無
secure_auth	是	此參數在 Aurora MySQL 第 2 版中始終開啟。嘗試將其關閉會產生錯誤。 已從 Aurora MySQL 第 3 版中移除。
secure_file_priv	否	Aurora MySQL 會在您無法直接存取檔案系統時使用受管執行個體。
show_create_table_verbosity	是	啟用此變數會導致 SHOW_CREATE_TABLE 顯示 ROW_FORMAT ，無論其是否為預設格式。 此參數適用於 Aurora MySQL 2.12 版及更新版本以及 3 版。
skip-slave-start	否	無
skip_external_locking	否	無

參數名稱	可修改	備註
skip_show_database	是	無
slave_checkpoint_group	是	Aurora MySQL 第 2 版。在 Aurora MySQL 第 3 版中使用 replica_checkpoint_group 。
slave_checkpoint_period	是	Aurora MySQL 第 2 版。在 Aurora MySQL 第 3 版中使用 replica_checkpoint_period 。
slave_parallel_workers	是	Aurora MySQL 第 2 版。在 Aurora MySQL 第 3 版中使用 replica_parallel_workers 。
slave_pending_jobs_size_max	是	Aurora MySQL 第 2 版。在 Aurora MySQL 第 3 版中使用 replica_pending_jobs_size_max 。
slave_sql_verify_checksum	是	Aurora MySQL 第 2 版。在 Aurora MySQL 第 3 版中使用 replica_sql_verify_checksum 。
slow_launch_time	是	無
slow_query_log	是	如需上傳日誌到 CloudWatch Logs 的說明，請參閱 將 Amazon Aurora MySQL 日誌發佈至 Amazon CloudWatch Logs。
slow_query_log_file	否	Aurora MySQL 會在您無法直接存取檔案系統時使用受管執行個體。
socket	否	無
sort_buffer_size	是	無
sql_mode	是	無

參數名稱	可修改	備註
sql_select_limit	是	無
stored_program_cache	是	無
sync_binlog	否	無
sync_master_info	是	無
sync_source_info	是	此參數適用於 Aurora MySQL 第 3 版。
sync_relay_log	是	已從 Aurora MySQL 第 3 版中移除。
sync_relay_log_info	是	無
sysdate-is-now	是	無
table_cache_element_entry_ttl	否	無
table_definition_cache	是	預設值以公式表示。如需如何在公式中計算 DBInstanceClassMemory 值的詳細資訊，請參閱 資料庫參數公式變數。
table_open_cache	是	預設值以公式表示。如需如何在公式中計算 DBInstanceClassMemory 值的詳細資訊，請參閱 資料庫參數公式變數。
table_open_cache_instances	是	無
temp-pool	是	已從 Aurora MySQL 第 3 版中移除。
temptable_max_mmap	是	此參數適用於 Aurora MySQL 第 3 版。如需詳細資訊，請參閱 Aurora MySQL 第 3 版的新暫時資料表行為。

參數名稱	可修改	備註
temptable_max_ram	是	此參數適用於 Aurora MySQL 第 3 版。如需詳細資訊，請參閱 Aurora MySQL 第 3 版的新暫時資料表行為。
temptable_use_mmap	是	此參數適用於 Aurora MySQL 第 3 版。如需詳細資訊，請參閱 Aurora MySQL 第 3 版的新暫時資料表行為。
thread_cache_size	是	要快取的執行緒數目。此參數適用於 Aurora MySQL 第 2 版和第 3 版。
thread_handling	否	無
thread_stack	是	無
timed_mutexes	是	無
tmp_table_size	是	定義 MEMORY 儲存體引擎在 Aurora MySQL 3 版中建立內部記憶體暫時資料表的大小上限。 在 Aurora MySQL 3.04 及更新版本中，定義 TempTable 記憶體引擎建立的內部記憶體中暫時資料表的大小上限當 aurora_tmptable_enable_per_table_limit 時為 ON。 如需更多詳細資訊，請參閱 限制記憶體內部暫存資料表的大小。
tmpdir	否	Aurora MySQL 會在您無法直接存取檔案系統時使用受管執行個體。
transaction_alloc_block_size	是	無

參數名稱	可修改	備註
transaction_isolation	是	此參數適用於 Aurora MySQL 第 3 版。其會取代 tx_isolation 。
transaction_prealloc_size	是	無
tx_isolation	是	已從 Aurora MySQL 第 3 版中移除。其會遭 transaction_isolation 取代。
updatable_views_with_limit	是	無
validate-password	否	無
validate_password_dictionary_file	否	無
validate_password_length	否	無
validate_password_mixed_case_count	否	無
validate_password_number_count	否	無
validate_password_policy	否	無
validate_password_special_char_count	否	無
wait_timeout	是	Aurora 會評估 interactive_timeout 和 wait_timeout 的最小值。然後使用該最小值作為逾時來結束所有閒置工作階段，包括互動式和非互動的工作階段。

不適用於 Aurora MySQL 的 MySQL 參數

由於 Aurora MySQL 與 MySQL 之間的架構不同，某些 MySQL 參數不適用於 Aurora MySQL。

以下 MySQL 參數不適用於 Aurora MySQL。這不是完整清單。

- `activate_all_roles_on_login` – 此參數不適用於 Aurora MySQL 第 2 版。其可在 Aurora MySQL 第 3 版中使用。
- `big_tables`
- `bind_address`
- `character_sets_dir`
- `innodb_adaptive_flushing`
- `innodb_adaptive_flushing_lwm`
- `innodb_buffer_pool_chunk_size`
- `innodb_buffer_pool_instances`
- `innodb_change_buffering`
- `innodb_checksum_algorithm`
- `innodb_data_file_path`
- `innodb_dedicated_server`
- `innodb_doublewrite`
- `innodb_flush_log_at_timeout` – 此參數不適用於 Aurora MySQL。如需更多詳細資訊，請參閱 設定日誌緩衝區的排清頻率。
- `innodb_flush_method`
- `innodb_flush_neighbors`
- `innodb_io_capacity`
- `innodb_io_capacity_max`
- `innodb_log_buffer_size`
- `innodb_log_file_size`
- `innodb_log_files_in_group`
- `innodb_log_spin_cpu_abs_lwm`
- `innodb_log_spin_cpu_pct_hwm`
- `innodb_log_writer_threads`
- `innodb_max_dirty_pages_pct`

- innodb_numa_interleave
- innodb_page_size
- innodb_redo_log_capacity
- innodb_redo_log_encrypt
- innodb_undo_log_encrypt
- innodb_undo_log_truncate
- innodb_undo_logs
- innodb_undo_tablespaces
- innodb_use_native_aio
- innodb_write_io_threads

Aurora MySQL 全域狀態變數

Aurora MySQL 包含社群 MySQL 的狀態變數，以及 Aurora 獨有的變數。您可以檢查這些變數，以了解資料庫引擎內發生的情況。如需社群 MySQL 中狀態變數的詳細資訊，請參閱社群 MySQL 8.0 文件中的伺服器狀態變數。

您可以使用下列陳述式來尋找 Aurora MySQL 全域狀態變數目前的值：

```
show global status like '%aurora%';
```

> **Note**
>
> 資料庫引擎重新啟動時會清除全域狀態變數。

下列資料表說明 Aurora MySQL 使用的全域狀態變數。

名稱	描述
AuroraDb_commits	自上次重新啟動以來的提交總數。
AuroraDb_commit_latency	自上次重新啟動以來的提交延遲的彙總。
AuroraDb_ddl_stmt_duration	自上次重新啟動以來的 DDL 延遲的彙總。

名稱	描述
AuroraDb_select_stmt_duration	自上次重新啟動以來的 SELECT 陳述式延遲的彙總。
AuroraDb_insert_stmt_duration	自上次重新啟動以來的 INSERT 陳述式延遲的彙總。
AuroraDb_update_stmt_duration	自上次重新啟動以來的 UPDATE 陳述式延遲的彙總。
AuroraDb_delete_stmt_duration	自上次重新啟動以來的 DELETE 陳述式延遲的彙總。
Aurora_binlog_io_cache_allocated	配置給 Binlog 輸入/輸出快取記憶體的位元組數目。
Aurora_binlog_io_cache_read_requests	對 binlog 輸入/輸出快取發起讀取請求的次數。
Aurora_binlog_io_cache_reads	從 binlog 輸入/輸出快取發起讀取請求的次數。
Aurora_enhanced_binlog	指示此資料庫執行個體是啟用還是停用增強型 binlog。如需詳細資訊，請參閱設定 Aurora MySQL 的增強型 binlog。
Aurora_external_connection_count	資料庫執行個體的資料庫連線數目，不包括用於資料庫運作狀態檢查的 RDS 服務連線。
Aurora_fast_insert_cache_hits	成功擷取並驗證快取游標時會遞增的計數器。如需快速插入快取的詳細資訊，請參閱Amazon Aurora MySQL 效能增強功能。
Aurora_fast_insert_cache_misses	快取游標不再有效，且 Aurora 執行正常索引周遊時會遞增的計數器。如需快速插入快取的詳細資訊，請參閱Amazon Aurora MySQL 效能增強功能。

名稱	描述
Aurora_fts_cache_memory_used	InnoDB 全文搜尋系統正在使用的位元組記憶體數量。此變數適用於 Aurora MySQL 3.07 版和更高版本。
Aurora_fwd_master_dml_stmt_count	轉送至此寫入器資料庫執行個體的 DML 陳述式總數。此變數適用於 Aurora MySQL 2 版。
Aurora_fwd_master_dml_stmt_duration	轉送至此寫入器資料庫執行個體的 DML 陳述式總持續時間。此變數適用於 Aurora MySQL 2 版。
Aurora_fwd_master_errors_rpc_timeout	無法在寫入器上建立轉送連線的次數。
Aurora_fwd_master_errors_session_limit	由於寫入器上 session full 原因而拒絕轉送查詢的次數。
Aurora_fwd_master_errors_session_timeout	轉送工作階段由於寫入器逾時而結束的次數。
Aurora_fwd_master_open_sessions	寫入器資料庫執行個體上轉送的工作階段數目。此變數適用於 Aurora MySQL 2 版。
Aurora_fwd_master_select_stmt_count	轉送至此寫入器資料庫執行個體的 SELECT 陳述式總數。此變數適用於 Aurora MySQL 2 版。
Aurora_fwd_master_select_stmt_duration	轉送至此寫入器資料庫執行個體的 SELECT 陳述式總持續時間。此變數適用於 Aurora MySQL 2 版。
Aurora_fwd_writer_dml_stmt_count	轉送至此寫入器資料庫執行個體的 DML 陳述式總數。此變數適用於 Aurora MySQL 3 版。
Aurora_fwd_writer_dml_stmt_duration	轉送至此寫入器資料庫執行個體的 DML 陳述式總持續時間。此變數適用於 Aurora MySQL 3 版。

名稱	描述
Aurora_fwd_writer_errors_rpc_timeout	無法在寫入器上建立轉送連線的次數。
Aurora_fwd_writer_errors_session_limit	由於寫入器上 session full 原因而拒絕轉送查詢的次數。
Aurora_fwd_writer_errors_session_timeout	轉送工作階段由於寫入器逾時而結束的次數。
Aurora_fwd_writer_open_sessions	寫入器資料庫執行個體上轉送的工作階段數目。此變數適用於 Aurora MySQL 3 版。
Aurora_fwd_writer_select_stmt_count	轉送至此寫入器資料庫執行個體的 SELECT 陳述式總數。此變數適用於 Aurora MySQL 3 版。
Aurora_fwd_writer_select_stmt_duration	轉送至此寫入器資料庫執行個體的 SELECT 陳述式總持續時間。此變數適用於 Aurora MySQL 3 版。
Aurora_lockmgr_buffer_pool_memory_used	Aurora MySQL 鎖定管理員正在使用的緩衝集區記憶體大小 (以位元組為單位)。
Aurora_lockmgr_memory_used	Aurora MySQL 鎖定管理員正在使用的記憶體大小 (以位元組為單位)。
Aurora_ml_actual_request_cnt	Aurora MySQL 從 Aurora 機器學習服務接收的彙總請求計數，涵蓋資料庫執行個體的使用者執行的所有查詢。如需詳細資訊，請參閱將 Amazon Aurora Machine Learning 與 Aurora MySQL 搭配使用。
Aurora_ml_actual_response_cnt	Aurora MySQL 從 Aurora Machine Learning 服務接收的彙總回應計數，涵蓋資料庫執行個體的使用者執行的所有查詢。如需詳細資訊，請參閱將 Amazon Aurora Machine Learning 與 Aurora MySQL 搭配使用。

名稱	描述
Aurora_ml_cache_hit_cnt	Aurora MySQL 從 Aurora Machine Learning 服務接收的彙總內部快取命中計數，涵蓋資料庫執行個體的使用者執行的所有查詢。如需詳細資訊，請參閱將 Amazon Aurora Machine Learning 與 Aurora MySQL 搭配使用。
Aurora_ml_logical_request_cnt	自上次重新設定狀態後，資料庫執行個體已評估傳送至 Aurora 機器學習服務的邏輯請求數目。視是否使用批次處理而定，此值可能會大於 Aurora_ml_actual_request_cnt 。如需詳細資訊，請參閱將 Amazon Aurora Machine Learning 與 Aurora MySQL 搭配使用。
Aurora_ml_logical_response_cnt	Aurora MySQL 從 Aurora Machine Learning 服務接收的彙總回應計數，涵蓋資料庫執行個體的使用者執行的所有查詢。如需詳細資訊，請參閱將 Amazon Aurora Machine Learning 與 Aurora MySQL 搭配使用。
Aurora_ml_retry_request_cnt	自上次重新設定狀態後，資料庫執行個體傳送至 Aurora 機器學習服務的重試請求數目。如需詳細資訊，請參閱將 Amazon Aurora Machine Learning 與 Aurora MySQL 搭配使用。
Aurora_ml_single_request_cnt	由非批次模式評估的 Aurora Machine Learning 函數的彙總計數，涵蓋資料庫執行個體的使用者執行的所有查詢。如需詳細資訊，請參閱將 Amazon Aurora Machine Learning 與 Aurora MySQL 搭配使用。

名稱	描述
aurora_oom_avoidance_recovery_state	指出此資料庫執行個體的 Aurora 記憶體不足 (OOM) 迴避復原是否處於 ACTIVE 或 INACTIVE 狀態。 此變數適用於 Aurora MySQL 3.06.0 版和更高版本。
aurora_oom_reserved_mem_enter_kb	代表在 Aurora 的 OOM 處理機制中進入 RESERVED 狀態的閾值。 當伺服器上可用的記憶體低於此閾值時，aurora_oom_status 會變更為 RESERVED，表示伺服器即將接近記憶體用量的臨界層級。 此變數適用於 Aurora MySQL 3.06.0 版和更高版本
aurora_oom_reserved_mem_exit_kb	代表在 Aurora 的 OOM 處理機制中結束 RESERVED 狀態的閾值。 當伺服器上的可用記憶體超過此閾值時，aurora_oom_status 會還原到 NORMAL，表示伺服器已返回到具有足夠記憶體資源的更穩定狀態。 此變數適用於 Aurora MySQL 3.06.0 版和更高版本。

名稱	描述
aurora_oom_status	代表此資料庫執行個體目前的 OOM 狀態。當值為 NORMAL 時，表示有足夠的記憶體資源。 如果值變更為 RESERVED，則表示伺服器的可用記憶體不足。根據 aurora_oom_response 參數組態採取動作。 如需詳細資訊，請參閱故障診斷 Aurora MySQL 資料庫記憶體不足的問題。 此變數適用於 Aurora MySQL 3.06.0 版和更高版本。
Aurora_pq_bytes_returned	在平行查詢期間已傳輸至前端節點之 Tuple 資料結構的位元組數目。除以 16,384 以針對 Aurora_pq_pages_pushed_down 進行比較。
Aurora_pq_max_concurrent_requests	可以同時在此 Aurora 資料庫執行個體上執行之平行查詢工作階段的數目上限。這是取決於 AWS 資料庫執行個體類別的固定數字。
Aurora_pq_pages_pushed_down	資料頁面的數目 (每個頁面的固定大小為 16 KiB)，在這些資料頁面中平行查詢已避免透過網路將資料傳輸至前端節點。
Aurora_pq_request_attempted	已請求的平行查詢工作階段數目。此值可能代表每個查詢多個工作階段，取決於 SQL 建構，例如子查詢和聯結。
Aurora_pq_request_executed	已成功執行的平行查詢工作階段數目。

名稱	描述
Aurora_pq_request_failed	已傳回錯誤至用戶端的平行查詢工作階段數目。在某些情況下，平行查詢的請求可能失敗，例如，因為儲存層中發生問題。在這些情況下，會使用非平行查詢機制來重試失敗的查詢部分。如果重試的查詢也失敗，則錯誤會傳回至用戶端，而且此計數器會遞增。
Aurora_pq_request_in_progress	目前進行中的平行查詢工作階段數目。此數目適用於您已連線的特定 Aurora 資料庫執行個體，但不適用於整個 Aurora 資料庫叢集。若要查看資料庫執行個體是否接近並行限制，請將此值與 Aurora_pq_max_concurrent_requests 比較。
Aurora_pq_request_not_chosen	未選擇平行查詢以滿足查詢的次數。此值是數個其他更精細計數器的總和。EXPLAIN 陳述式可以增加此計數器，即使查詢實際上並未執行。
Aurora_pq_request_not_chosen_below_min_rows	由於資料表中的資料列數而未選擇平行查詢的次數。EXPLAIN 陳述式可以增加此計數器，即使查詢實際上並未執行。
Aurora_pq_request_not_chosen_column_bit	使用非平行查詢處理路徑，因為投影的資料欄清單中不支援的資料類型的平行查詢要求數目。
Aurora_pq_request_not_chosen_column_geometry	因為 GEOMETRY 資料表具有資料類型的資料行，所以使用非平行查詢處理路徑的平行查詢要求數目。如需移除此限制之 Aurora MySQL 版本的相關資訊，請參閱將平行查詢叢集升級至 Aurora MySQL 第 3 版。

名稱	描述
Aurora_pq_request_not_chosen_column_lob	使用非平行查詢處理路徑的平行查詢要求數目，因為資料表具有 LOB 資料類型的 VARCHAR 資料欄，或因宣告長度而儲存在外部的資料欄。如需移除此限制之 Aurora MySQL 版本的相關資訊，請參閱將平行查詢叢集升級至 Aurora MySQL 第 3 版。
Aurora_pq_request_not_chosen_column_virtual	因為資料表包含虛擬資料欄，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_custom_charset	因為資料表具有自訂字元集的資料欄，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_fast_ddl	使用非平行查詢處理路徑的平行查詢要求數目，因為資料表目前正在變更快速的 DDL ALTER 陳述式。
Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool	即使小於 95% 的資料表資料在緩衝集區中，也未選擇平行查詢的次數，因為沒有足夠的未置於緩衝的資料表資料，讓平行查詢值得執行。
Aurora_pq_request_not_chosen_full_text_index	因為資料表具有全文檢索索引，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_high_buffer_pool_pct	因為高百分比的資料表資料 (目前，大於 95%) 已在緩衝集區中，所以未選擇平行查詢的次數。在這些情況下，最佳化器判定從緩衝集區讀取資料最有效率。EXPLAIN 陳述式可以增加此計數器，即使查詢實際上並未執行。
Aurora_pq_request_not_chosen_index_hint	因為查詢包含索引提示，所以會使用非平行查詢處理路徑的平行查詢要求數目。

名稱	描述
Aurora_pq_request_not_chosen_innodb_table_format	因為資料表使用不支援的 InnoDB 資料列格式，所以會使用非平行查詢處理路徑的平行查詢要求數目。Aurora 平行查詢只適用於 COMPACT、REDUNDANT 和 DYNAMIC 資料列格式。
Aurora_pq_request_not_chosen_long_trx	由於在長時間執行的交易內啟動查詢，而使用非平行查詢處理路徑的平行查詢請求數目。EXPLAIN 陳述式可以增加此計數器，即使查詢實際上並未執行。
Aurora_pq_request_not_chosen_no_where_clause	因為查詢不包含任何 WHERE 子句，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_range_scan	因為查詢在索引上使用範圍掃描，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_row_length_too_long	因為所有資料欄的總合長度太長，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_small_table	由於資料表中的整體大小 (由資料列數和平均資料列長度決定) 而未選擇平行查詢的次數。EXPLAIN 陳述式可以增加此計數器，即使查詢實際上並未執行。
Aurora_pq_request_not_chosen_temporary_table	因為查詢參考使用不支援 MyISAM 或 memory 資料表類型的暫存資料表，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_tx_isolation	因為查詢使用不支援的交易隔離層級，所以會使用非平行查詢處理路徑的平行查詢要求數目。在讀取器資料庫執行個體上，平行查詢僅適用於 REPEATABLE READ 和 READ COMMITTED 隔離層級。

名稱	描述
Aurora_pq_request_not_chosen_update_delete_stmts	因為查詢是 UPDATE 或 DELETE 陳述式的一部分，所以會使用非平行查詢處理路徑的平行查詢要求數目。
Aurora_pq_request_not_chosen_unsupported_access	因為 WHERE 子句不符合平行查詢的條件，所以使用非平行查詢處理路徑的平行查詢請求數目。如果查詢不需要資料密集掃描，或如果查詢是 DELETE 或 UPDATE 陳述式，則會發生此結果。
Aurora_pq_request_not_chosen_unsupported_storage_type	由於 Aurora MySQL 資料庫叢集未使用支援的 Aurora 叢集儲存組態，因此平行查詢數目請求是使用非平行查詢處理路徑。如需詳細資訊，請參閱限制。 此參數適用於 Aurora MySQL 3.04 版及更新版本。
Aurora_pq_request_throttled	由於已在特定 Aurora 資料庫執行個體上執行的並行平行查詢數目已達到上限，而未選擇平行查詢的次數。
Aurora_repl_bytes_received	自上次重新啟動後，複寫至 Aurora MySQL 讀取器資料庫執行個體的位元組數目。如需詳細資訊，請參閱以 Amazon Aurora MySQL 進行複寫。
Aurora_reserved_mem_exceeded_incidents	自上次重新啟動後，引擎超出保留記憶體限制的次數。如果已設定 aurora_oom_response ，此閾值定義何時觸發記憶體不足 (OOM) 避免活動。如需 Aurora MySQL OOM 回應的詳細資訊，請參閱故障診斷 Aurora MySQL 資料庫記憶體不足的問題。
aurora_temptable_max_ram_allocation	自上次重新啟動後，內部暫存資料表隨時使用的記憶體數量上限，以位元組為單位。

名稱	描述
aurora_temptable_ram_allocation	內部暫存資料表使用的目前記憶體數量，以位元組為單位。
Aurora_in_memory_relaylog_status	內部記憶體轉送日誌功能的目前狀態，值可以是 ENABLED 或 DISABLED。
Aurora_in_memory_relaylog_disabled_reason	顯示內部記憶體轉送日誌功能狀態的目前原因，如果功能已停用，則會顯示說明功能停用原因的訊息。
Aurora_in_memory_relaylog_fallback_count	顯示內部記憶體轉送日誌功能至持久性轉送日誌模式 (舊版) 的備用總數。備用可能是由於單一事件大於快取大小 (目前為 128MB) 或交易重試超過複本交易重試限制 replica_transaction_retries 所致。
Aurora_in_memory_relaylog_recovery_count	顯示自動執行的內部記憶體轉送日誌復原的總數。此計數包括備用總數，以及在暫時備用之後，自動模式切換回內部記憶體轉送日誌模式的次數。
Aurora_thread_pool_thread_count	Aurora 執行緒集區中目前的執行緒數目。如需 Aurora MySQL 執行緒集區回應的詳細資訊，請參閱執行緒集區。
Aurora_tmz_version	表示資料庫叢集使用時區資訊的目前版本。這些值會遵循網際網路號碼分配機構 (IANA) 格式：YYYYsuffix ，例如 2022a 和 2023c。 此參數適用於 Aurora MySQL 2.12 版及更新版本以及第 3.04 版及更新版本。
Aurora_zdr_oom_threshold	代表 Aurora 資料庫執行個體啟動零停機時間重新啟動 (ZDR)，從潛在記憶體相關問題中復原的記憶體閾值，以 KB 為單位。

名稱	描述
server_aurora_das_running	指示此資料庫執行個體上是啟用還是停用資料庫活動串流 (DAS)。如需詳細資訊，請參閱使用資料庫活動串流來監控 Amazon Aurora。

不適用於 Aurora MySQL 的 MySQL 狀態變數

由於 Aurora MySQL 與 MySQL 之間的架構不同，某些 MySQL 狀態變數不適用於 Aurora MySQL。

以下 MySQL 狀態變數不適用於 Aurora MySQL。這不是完整清單。

- innodb_buffer_pool_bytes_dirty
- innodb_buffer_pool_pages_dirty
- innodb_buffer_pool_pages_flushed

Aurora MySQL 第 3 版會移除 Aurora MySQL 第 2 版中的下列狀態變數：

- AuroraDb_lockmgr_bitmaps0_in_use
- AuroraDb_lockmgr_bitmaps1_in_use
- AuroraDb_lockmgr_bitmaps_mem_used
- AuroraDb_thread_deadlocks
- available_alter_table_log_entries
- Aurora_lockmgr_memory_used
- Aurora_missing_history_on_replica_incidents
- Aurora_new_lock_manager_lock_release_cnt
- Aurora_new_lock_manager_lock_release_total_duration_micro
- Aurora_new_lock_manager_lock_timeout_cnt
- Aurora_total_op_memory
- Aurora_total_op_temp_space
- Aurora_used_alter_table_log_entries
- Aurora_using_new_lock_manager
- Aurora_volume_bytes_allocated

- Aurora_volume_bytes_left_extent
- Aurora_volume_bytes_left_total
- Com_alter_db_upgrade
- Compression
- External_threads_connected
- Innodb_available_undo_logs
- Last_query_cost
- Last_query_partial_plans
- Slave_heartbeat_period
- Slave_last_heartbeat
- Slave_received_heartbeats
- Slave_retried_transactions
- Slave_running
- Time_since_zero_connections

這些 MySQL 狀態變數可在 Aurora MySQL 第 2 版中使用，但它們不可在 Aurora MySQL 第 3 版中使用：

- Innodb_redo_log_enabled
- Innodb_undo_tablespaces_total
- Innodb_undo_tablespaces_implicit
- Innodb_undo_tablespaces_explicit
- Innodb_undo_tablespaces_active

Aurora MySQL 等待事件

以下是 Aurora MySQL 的一些常見等候事件。

> **Note**
>
> 如需詳細了解使用等待事件調整 Aurora MySQL 效能，請參閱 使用等待事件調校 Aurora MySQL。

如需關於用於 MySQL 等候事件的命名慣例資訊，請參閱 MySQL 文件中的效能結構描述工具命名慣例。

cpu

準備好執行的作用中連線數量一致高於 vCPU 的數量。如需詳細資訊，請參閱cpu。

io/aurora_redo_log_flush

工作階段正在將資料保留至 Aurora 儲存體。一般而言，此等候事件是供 Aurora MySQL 的寫入輸入/輸出操作使用的。如需詳細資訊，請參閱io/aurora_redo_log_flush。

io/aurora_respond_to_client

查詢處理已完成，並將結果傳回至下列 Aurora MySQL 版本的應用程式用戶端：2.10.2 及更高的 2.10 版本、2.09.3 及更高的 2.09 版本，以及 2.07.7 及更高的 2.07 版本。將資料庫執行個體類別的網路頻寬與傳回的結果集大小進行比較。此外，請檢查用戶端回應時間。如果用戶端沒有回應，而且無法處理 TCP 封包，則可能會發生封包捨棄和 TCP 重新傳輸。這種情況會對網路頻寬造成負面影響。在低於 2.10.2、2.09.3 和 2.07.7 的版本中，等待事件錯誤地包含閒置時間。若要了解如何在此等待重要時調整資料庫，請參閱 io/aurora_respond_to_client。

io/file/csv/data

執行緒正在以逗號分隔值 (CSV) 格式寫入資料表。檢查您的 CSV 資料庫使用量。此事件通常是因為資料表上設定 `log_output`。

io/file/sql/binlog

執行緒正在等待將寫入至磁碟的二進位日誌 (binlog) 檔案。

io/redo_log_flush

工作階段正在將資料保留至 Aurora 儲存體。一般而言，此等候事件是供 Aurora MySQL 的寫入輸入/輸出操作使用的。如需詳細資訊，請參閱io/redo_log_flush。

io/socket/sql/client_connection

`mysqld` 程式忙於建立執行緒來處理傳入的新用戶端連線。如需詳細資訊，請參閱io/socket/sql/client_connection。

io/table/sql/handler

引擎正在等待存取資料表。無論是在緩衝集區中快取資料，還是在磁碟上存取資料，都會發生此事件。如需詳細資訊，請參閱io/table/sql/handler。

lock/table/sql/handler

此等候事件是一個資料表鎖定事件處理常式。如需效能結構描述中原子和分子事件的相關資訊，請參閱 MySQL 文件中的效能結構描述原子與分子事件。

synch/cond/innodb/row_lock_wait

多個資料處理語言 (DML) 陳述式正在同時存取相同的資料庫資料列。如需詳細資訊，請參閱synch/cond/innodb/row_lock_wait。

synch/cond/innodb/row_lock_wait_cond

多個 DML 陳述式正在同時存取相同的資料庫資料列。如需詳細資訊，請參閱synch/cond/innodb/row_lock_wait_cond。

synch/cond/sql/MDL_context::COND_wait_status

執行緒正在等待資料表中繼資料鎖定。引擎使用這種類型的鎖定來管理資料庫結構描述的並行存取，並確保資料一致性。如需詳細資訊，請參閱 MySQL 文件中的最佳化鎖定操作。若要了解如何在此事件重要時調整資料庫，請參閱 synch/cond/sql/MDL_context::COND_wait_status。

synch/cond/sql/MYSQL_BIN_LOG::COND_done

您已開啟二進位日誌。可能有很高的遞交輸送量、大量交易遞交，或複本讀取 Binlog。考慮使用多列陳述式或將陳述式綁定成一個交易。在 Aurora 中，使用全域資料庫而非二進位日誌複寫，或使用 `aurora_binlog_*` 參數。

synch/mutex/innodb/aurora_lock_thread_slot_futex

多個 DML 陳述式正在同時存取相同的資料庫資料列。如需詳細資訊，請參閱synch/mutex/innodb/aurora_lock_thread_slot_futex。

synch/mutex/innodb/buf_pool_mutex

緩衝集區不夠大，無法容納工作資料集。或者，工作負載會從特定資料表存取頁面，這會導致緩衝集區中的爭用。如需詳細資訊，請參閱synch/mutex/innodb/buf_pool_mutex。

synch/mutex/innodb/fil_mutex

程序正在等待存取資料表空間記憶體快取。如需詳細資訊，請參閱synch/mutex/innodb/fil_mutex。

synch/mutex/innodb/fil_mutex

作業是以一致或控制方式在 InnoDB 中檢查、更新、刪除或新增交易 ID。這些作業需要 `trx_sys` 互斥呼叫，這是由效能結構描述偵測來追蹤。作業包括在資料庫啟動或關閉時管理交易系統、回

復、復原清除、資料列讀取存取，以及緩衝集區載入。具有大量交易的高資料庫負載會導致此等待事件頻繁出現。如需詳細資訊，請參閱synch/mutex/innodb/fil_mutex。

synch/mutex/mysys/KEY_CACHE::cache_lock

`keycache->cache_lock` 互斥鎖定可控制對 MyISAM 資料表之索引鍵快取的存取。雖然 Aurora MySQL 不允許使用 MyISAM 資料表來存放持久性資料，但它們是用來存放內部暫存資料表。請考慮檢查 `created_tmp_tables` 或 `created_tmp_disk_tables` 狀態計數器，因為在某些情況下，暫存資料表會在不再容納於記憶體時寫入至磁碟。

synch/mutex/sql/file _A_table:: LOCK_Offset

開啟或建立資料表中繼資料檔案時，引擎會取得此互斥。當這個等待事件過於頻繁發生時，表示正在建立或開啟的資料表數目已爆增。

synch/mutex/sql/FILE_AS_TABLE::LOCK_shim_lists

在追蹤所開啟資料表的內部結構上執行 `reset_size`、`detach_contents` 或 `add_contents` 之類的作業時，引擎會取得此互斥。互斥會同步對清單內容的存取。當這個等待事件頻繁發生時，它表示先前存取的資料表集突然變更。引擎需要存取新的資料表或放下與先前存取之資料表相關的內容。

synch/mutex/sql/LOCK_open

工作階段正在開啟的資料表數目超過資料表定義快取或資料表開啟快取的大小。增加這些快取的大小。如需詳細資訊，請參閱 MySQL 如何開啟及關閉資料表。

synch/mutex/sql/LOCK_table_cache

工作階段正在開啟的資料表數目超過資料表定義快取或資料表開啟快取的大小。增加這些快取的大小。如需詳細資訊，請參閱 MySQL 如何開啟及關閉資料表。

synch/mutex/sql/LOG

在此等候事件中，執行緒正在等待日誌鎖定。例如，一個執行緒可能會等待鎖定寫入至慢查詢日誌檔案。

synch/mutex/sql/MYSQL_BIN_LOG::LOCK_commit

在此等候事件中，有一個執行緒正在等候取得欲提交至二進位日誌的鎖定。在具有非常高度變動率的資料庫上可能會發生二進位記錄爭用。依照您的 MySQL 版本，某些鎖定會用於保護二進位日誌的一致性和耐用性。在 RDS for MySQL 中，二進位日誌用於複寫與自動備份處理。在 Aurora MySQL 中，原生複寫或備份並不需要二進位日誌。他們依預設為停用，但可以啟用並用於外部複寫或變更資料擷取。如需詳細資訊，請參閱 MySQL 文件中的二進位日誌。

sync/mutex/sql/MYSQL_BIN_LOG::LOCK_dump_thread_metrics_collection

如果開啟了二進位記錄，則在引擎將作用中傾印執行緒指標列印至引擎錯誤記錄日誌和內部作業映射時，其會取得此互斥。

sync/mutex/sql/MYSQL_BIN_LOG::LOCK_inactive_binlogs_map

如果開啟了二進位記錄，則在引擎新增至其中、從中刪除，或搜尋最新清單後面的完整 Binlog 檔案清單時，其會取得此互斥。

sync/mutex/sql/MYSQL_BIN_LOG::LOCK_io_cache

如果開啟了二進位記錄，引擎會在 Aurora Binlog IO 快取作業期間取得此互斥：配置、調整大小、釋放、寫入、讀取、清除和存取快取資訊。如果此事件經常發生，引擎會存取 Binlog 事件存放所在的快取。若要減少等待時間，請減少遞交。嘗試將多個陳述式分組成單一交易。

synch/mutex/sql/MYSQL_BIN_LOG::LOCK_log

您已開啟二進位日誌。可能有很高的遞交輸送量、許多交易遞交，或複本讀取 Binlog。考慮使用多列陳述式或將陳述式綁定成一個交易。在 Aurora 中，使用全域資料庫而非二進位日誌複寫，或使用 `aurora_binlog_*` 參數。

synch/mutex/sql/SERVER_THREAD::LOCK_sync

互斥 `SERVER_THREAD::LOCK_sync` 是在排程、處理或啟動執行緒以進行檔案寫入期間取得的。過度發生此等待事件表示資料庫中的寫入活動增加。

synch/mutex/sql/TABLESPACES:lock

引擎會在下列資料表空間作業期間取得 `TABLESPACES:lock` 互斥：建立、刪除、截斷和延伸。過度發生此等待事件表示資料表空間作業頻率很高。範例是將大量資料載入至資料庫。

synch/rwlock/innodb/dict

在此等候事件中，執行緒在 InnoDB 資料字典保存的 rwlock (讀取寫入鎖定) 上進行等待。

synch/rwlock/innodb/dict_operation_lock

在此等候事件中，執行緒正在 InnoDB 資料字典操作上持有鎖定。

synch/rwlock/innodb/dict sys RW lock

同時觸發資料定義語言程式碼 (DDL) 中的大量並行資料控制語言陳述式 (DCL)。在一般應用程式活動期間，減少應用程式對 DDL 的相依性。

synch/rwlock/innodb/index_tree_rw_lock

大量類似的資料處理語言 (DML) 陳述式正在同時存取相同的資料庫物件。嘗試使用多列陳述式。此外，將工作負載分散到不同的資料庫物件上。例如，實作分割。

synch/sxlock/innodb/dict_operation_lock

同時觸發資料定義語言程式碼 (DDL) 中的大量並行資料控制語言陳述式 (DCL)。在一般應用程式活動期間，減少應用程式對 DDL 的相依性。

synch/sxlock/innodb/dict_sys_lock

同時觸發資料定義語言程式碼 (DDL) 中的大量並行資料控制語言陳述式 (DCL)。在一般應用程式活動期間，減少應用程式對 DDL 的相依性。

synch/sxlock/innodb/hash_table_locks

工作階段找不到緩衝集區中的頁面。引擎需要讀取檔案或修改緩衝集區的最近最少使用 (LRU) 清單。考慮增加緩衝區快取大小，並改善相關查詢的存取路徑。

synch/sxlock/innodb/index_tree_rw_lock

許多類似的資料處理語言 (DML) 陳述式正在同時存取相同的資料庫物件。嘗試使用多列陳述式。此外，將工作負載分散到不同的資料庫物件上。例如，實作分割。

synch/mutex/innodb/temp_pool_manager_mutex

當工作階段正在等待取得互斥以管理工作階段暫存資料表空間集區時，會發生此等待事件。

如需同步等待事件疑難排解的詳細資訊，請參閱 Why is my MySQL DB instance showing a high number of active sessions waiting on SYNCH wait events in Performance Insights? (為什麼我的 MySQL 資料庫執行個體在績效詳情中會顯示等待 SYNCH 等待事件的大量作用中工作階段？)。

Aurora MySQL 執行緒狀態

以下是 Aurora MySQL 的一些常見執行緒狀態。

檢查許可

執行緒正在檢查伺服器是否具有執行陳述式所需的權限。

檢查查詢快取中是否有查詢

伺服器正在檢查目前的查詢是否存在於查詢快取中。

已清除

這是連線的最終狀態，表示其工作已完成，但用戶端尚未將其關閉。最好的解決方案是明確關閉程式碼中的連線。或者，您可以在參數群組中為 wait_timeout 設定較低的值。

關閉資料表

執行緒正在將已變更的資料表資料清空到磁碟並關閉已使用的資料表。如果這不是快速作業，請針對執行個體類別網路頻寬來驗證網路頻寬耗用指標。另外，請檢查 table_open_cache 和 table_definition_cache 參數的參數值是否允許足夠的資料表同時開啟，以便引擎不需要經常開啟和關閉資料表。這些參數會影響執行個體上的記憶體耗用量。

將 HEAP 轉換為 MyISAM

查詢正在將暫時資料表從記憶體內轉換為磁碟。這種轉換是必要的，因為在查詢處理的中繼步驟中由 MySQL 建立的暫時資料表對於記憶體來說增長太大。檢查 tmp_table_size 和 max_heap_table_size 的值。在更新的版本中，這個執行緒狀態名稱是 converting HEAP to ondisk。

將 HEAP 轉換為 ondisk

執行緒正在將內部暫時資料表從記憶體內資料表轉換為磁碟上資料表。

複製到 tmp 資料表

執行緒正在處理 ALTER TABLE 陳述式。在建立了具有新結構的資料表之後，但在將資料列複製到其中之前，會發生此狀態。對於處於此狀態的執行緒，您可以使用效能結構描述來取得複製作業進度的相關資訊。

建立排序索引

Aurora MySQL 正在執行排序，因為它不能使用現有的索引來滿足查詢的 ORDER BY 或 GROUP BY子句。如需更多詳細資訊，請參閱 正在建立排序索引。

建立資料表

執行緒正在建立永久或暫時資料表。

延遲遞交已順利完成

Aurora MySQL 中的非同步遞交已收到確認，且已完成。

延遲遞交已順利啟動

Aurora MySQL 執行緒已啟動非同步遞交程序，但正在等待確認。這通常是交易的真正遞交時間。

延遲傳送已順利完成

當回應傳送至用戶端時，可以釋放繫結至連線的 Aurora MySQL 工作者執行緒。執行緒可以開始其他工作。狀態 `delayed send ok` 表示對用戶端的非同步確認已完成。

延遲傳送已順利啟動

Aurora MySQL 工作者執行緒已非同步地將回應傳送至用戶端，現在可以自由地為其他連線執行工作。交易已啟動尚未確認的非同步遞交程序。

執行中

執行緒已開始執行陳述式。

釋放項目

執行緒已執行命令。某些在此狀態期間完成的項目釋放涉及查詢快取。這種狀態通常伴隨著清除。

init

這種狀態在初始化 `ALTER TABLE`、`DELETE`、`INSERT`、`SELECT` 或 `UPDATE` 陳述式之前發生。處於此狀態的動作包括清空二進位日誌或 InnoDB 日誌，以及查詢快取的部分清除。

來源已將所有 binlog 傳送至複本；正在等待更多更新

主節點已完成其複寫部分。執行緒正在等待更多的查詢執行，以便它可以寫入至二進位日誌 (Binlog)。

開啟表格

執行緒正在嘗試開啟資料表。此作業速度很快，除非 `ALTER TABLE` 或 `LOCK TABLE` 陳述式需要完成，否則它會超過 `table_open_cache` 的值。

最佳化

伺服器正在執行查詢的初始最佳化。

準備中

此狀態發生在查詢最佳化期間。

查詢結束

在處理查詢之後，但在釋放項目狀態之前，會發生此狀態。

移除重複項目

Aurora MySQL 無法在查詢的早期階段最佳化 `DISTINCT` 作業。Aurora MySQL 必須在將結果傳送至用戶端之前移除所有重複的資料列。

搜尋資料列以進行更新

執行緒正在尋找所有相符的資料列，然後再更新它們。如果 UPDATE 正在變更引擎用來尋找資料列的索引，這個階段是必要的。

將 Binlog 事件傳送到從屬節點

執行緒已從二進位日誌讀取事件，並正在將其傳送到複本。

將快取的結果傳送到用戶端

伺服器正在從查詢快取中取得查詢的結果，並將其傳送到用戶端。

傳送資料

執行緒正在讀取和處理 SELECT 陳述式的資料列，但尚未開始將資料傳送至用戶端。此程序正在識別哪些頁面包含滿足查詢所需的結果。如需更多詳細資訊，請參閱 正在傳送資料。

傳送至用戶端

伺服器正在將封包寫入至用戶端。在早期的 MySQL 版本中，此等待事件被標記為 `writing to net`。

開始

這是在陳述式執行開始的第一個階段。

統計資訊

伺服器正在計算統計資料以開發查詢執行計劃。如果執行緒長時間處於這種狀態，伺服器可能在執行其他工作時受到磁碟限制。

將結果存放在查詢快取中

伺服器正在將查詢的結果存放在查詢快取中。

系統鎖定

執行緒已呼叫 `mysql_lock_tables`，但執行緒狀態自呼叫以來未更新過。發生這種一般狀態的原因很多。

update

執行緒正在準備開始更新資料表。

更新

執行緒正在搜尋資料列並更新它們。

使用者鎖定

執行緒已發出 GET_LOCK 呼叫。執行緒已請求一個建議鎖定並正在等待它，或者正在規劃請求它。

等待更多更新

主節點已完成其複寫部分。執行緒正在等待更多的查詢執行，以便它可以寫入至二進位日誌 (Binlog)。

等待結構描述中繼資料鎖定

這表示等待中繼資料鎖定。

等待預存函數中繼資料鎖定

這表示等待中繼資料鎖定。

等待預存程序中繼資料鎖定

這表示等待中繼資料鎖定。

等待資料表清空

執行緒正在執行 FLUSH TABLES 並等待所有執行緒關閉其資料表。或者，執行緒收到了資料表的基礎結構已變更的通知，因此它必須重新開啟資料表以取得新結構。若要重新開啟資料表，執行緒必須等到所有其他執行緒關閉了資料表。如果另一個執行緒已在資料表上使用下列其中一個陳述式，就會發生此通知：FLUSH TABLES、ALTER TABLE、RENAME TABLE、REPAIR TABLE、ANALYZE TABLE 或 OPTIMIZE TABLE。

等待資料表層級鎖定

一個工作階段對資料表保有鎖定，而另一個工作階段嘗試對同一資料表取得相同的鎖定。

等待執行緒中繼資料鎖定

Aurora MySQL 使用中繼資料鎖定來管理資料庫物件的並行存取，並確保資料一致性。在此等待事件中，一個工作階段對資料表保有中繼資料鎖定，而另一個工作階段嘗試對同一資料表取得相同的鎖定。啟用效能結構描述時，此執行緒狀態會報告為等待事件 synch/cond/sql/MDL_context::COND_wait_status。

寫入至網路

伺服器正在將封包寫入至網路。在更新的 MySQL 版本中，此等待事件被標記為 Sending to client。

Aurora MySQL 隔離層級

了解 Aurora MySQL 叢集的資料庫執行個體如何實作資料庫的隔離屬性。此主題解釋 Aurora MySQL 預設行為如何在嚴格一致性與高效能之間取得平衡點。您可以根據工作負載的特性，使用此資訊協助您決定何時變更預設設定。

寫入器執行個體可用的隔離層級

您可以在 Aurora MySQL 資料庫叢集的主要執行個體上使用隔離等級 `REPEATABLE READ`、`READ COMMITTED`、`READ UNCOMMITTED` 和 `SERIALIZABLE`。這些隔離層級在 Aurora MySQL 和 RDS for MySQL 中以同樣的方式運作。

讀取器執行個體的 REPEATABLE READ 隔離層級

根據預設，設定成唯讀 Aurora 複本的 Aurora MySQL 資料庫執行個體一律使用 `REPEATABLE READ` 隔離層級。這些資料庫執行個體忽略任何 `SET TRANSACTION ISOLATION LEVEL` 陳述式，並繼續使用 `REPEATABLE READ` 隔離層級。

讀取器執行個體的 READ COMMITTED 隔離層級

如果應用程式在主要執行個體上有密集寫入的工作負載，而在 Aurora 複本上有長時間執行的查詢，您可能會感受到嚴重的清除延遲。如果長時間執行的查詢阻礙內部垃圾收集，此即所謂的清除延遲。您看見的徵狀是在 `history list length` 命令的輸出中，`SHOW ENGINE INNODB STATUS` 的值很高。您可以在 CloudWatch 中使用 `RollbackSegmentHistoryListLength` 指標監控此值。嚴重的清除延遲會降低次要索引的效率，導致整體查詢效能下降和儲存空間浪費。

如果遇到這種問題，您可以設定 Aurora MySQL 工作階段層級的組態設定 `aurora_read_replica_read_committed`，在 Aurora 複本上使用 `READ COMMITTED` 隔離層級。在交易修改資料表的同時，長時間執行的查詢可能導致速度變慢和浪費空間，套用此設定有助於避免這種情形。

使用此設定前，建議您了解 Aurora MySQL 在 `READ COMMITTED` 隔離情況下的具體行為。Aurora 複本 `READ COMMITTED` 行為符合 ANSI SQL 標準。然而，此隔離不像您所熟悉的 MySQL `READ COMMITTED` 獨特行為那麼嚴格。因此，就同樣的查詢而言，在 Aurora MySQL 僅供讀取複本的 `READ COMMITTED` 情況下，以及在 Aurora MySQL 主要執行個體或 RDS for MySQL 的 `READ COMMITTED` 情況下，您所見的查詢結果可能不同。在某些案例中，例如綜合報告掃描巨大的資料庫，您可以考慮使用 `aurora_read_replica_read_committed` 設定。反之，如果準確性和可重複性很重要，以較短的查詢來產生較小的結果集可避免此情況。

READ COMMITTED 隔離層級不適用於在使用寫入轉送功能的 Aurora 全域資料庫中次要叢集內的工作階段。如需寫入轉送的資訊，請參閱 在 Amazon Aurora 全域資料庫中使用寫入轉送。

對讀取器使用 READ COMMITTED

若要對 Aurora 複本使用 READ COMMITTED 隔離層級，請將 aurora_read_replica_read_committed 組態設定為 ON。請於連線至特定 Aurora 複本時，在工作階段層級使用此設定。若要這樣做，請執行下列 SQL 命令：

```
set session aurora_read_replica_read_committed = ON;
set session transaction isolation level read committed;
```

您可以暫時使用此組態設定，以執行互動的臨時性一次查詢。您也可以執行報告或資料分析應用程式來善用 READ COMMITTED 隔離層級，但對於其他應用程式維持預設設定不變。

啟用 aurora_read_replica_read_committed 設定時，請使用 SET TRANSACTION ISOLATION LEVEL 命令為適當的交易指定隔離層級。

```
set transaction isolation level read committed;
```

Aurora 複本上 READ COMMITTED 行為的差異

aurora_read_replica_read_committed 設定可讓 Aurora 複本使用 READ COMMITTED 隔離層級，其一致性行為最適合長時間執行的交易。Aurora 複本上的 READ COMMITTED 隔離層級，不像 Aurora 主要執行個體上的隔離那麼嚴格。因此，在 Aurora 複本上，只在您知道查詢能接受可能有某種不一致結果時，才應該啟用此設定。

啟用 aurora_read_replica_read_committed 設定時，查詢可能會遇到某種讀取異常狀況。在應用程式碼中，特別需要了解和處理兩種異常。當查詢執行時有另一個交易遞交，此即所謂的不可重複讀取。長時間執行的查詢在查詢開始和結束時所見的資料不同。當查詢執行時有其他交易導致現有的列重組，使得查詢讀取一或多列兩次，此即所謂的幻影讀取。

幻影讀取可能導致查詢看到不一致的列計數。查詢也可能因為不可重複讀取而傳回不完整或不一致的結果。例如，假設聯結操作所參考的資料表有 SQL 陳述式正在修改，例如 INSERT 或 DELETE。在此情況下，聯結從一個資料表中讀取的列，可能不是另一個資料表中相應的列。

ANSI SQL 標準在 READ COMMITTED 隔離層級下允許這兩種行為。不過，這些行為不同於 MySQL 典型的 READ COMMITTED 實作。因此，在啟用 aurora_read_replica_read_committed 設定之前，請檢查任何現有的 SQL 程式碼，確認在較寬鬆一致性模式下的運作符合預期。

啟用此設定時，在 READ COMMITTED 隔離層級下，列計數和其他結果可能不那麼一致。因此，通常只在執行分析查詢來彙總大量資料，且不需要絕對精準時，您才啟用此設定。如果您沒有這種長時間執行的查詢，也沒有密集寫入的工作負載，可能就不需要 aurora_read_replica_read_committed 設定。如果既沒有長時間執行的查詢，也沒有寫入密集的工作負載，就不太可能遭遇歷史記錄清單過長的問題。

Example顯示 Aurora 複本上 READ COMMITTED 隔離行為的查詢

下列範例顯示 Aurora 複本上的 READ COMMITTED 查詢在同時有交易修改相關聯資料表時，如何傳回不可重複的結果。在任何查詢開始前，資料表 BIG_TABLE 包含 1 百萬列。查詢執行時有其他資料操作語言 (DML) 陳述式新增、移除或變更列。

在 READ COMMITTED 隔離層級下，Aurora 主要執行個體上的查詢產生可預測的結果。但是，為了替每個長時間執行的查詢自始至終維持一致的讀取檢視，所付出的成本將導致後來垃圾收集的代價更高。

在 READ COMMITTED 隔離層級下，Aurora 複本上的查詢最能將此垃圾收集成本降到最低。根據查詢擷取的列是否為查詢執行當時遞交的交易所新增、移除或重組而定，結果可能不同，而這就是代價。查詢可以考慮這幾列，但並非必要。為了示範，查詢只使用 COUNT(*) 函數檢查資料表的列數。

時間	Aurora 主要執行個體上的 DML 陳述式	Aurora 主要執行個體上搭配 READ COMMITTED 的查詢	Aurora 複本上搭配 READ COMMITTED 的查詢
T1	`INSERT INTO big_table SELECT * FROM other_table LIMIT 1000000; COMMIT;`		
T2		Q1: `SELECT COUNT(*) FROM big_table;`	Q2: `SELECT COUNT(*) FROM big_table;`
T3	`INSERT INTO big_table (c1, c2) VALUES (1, 'one more row'); COMMIT;`		

時間	Aurora 主要執行個體上的 DML 陳述式	Aurora 主要執行個體上搭配 READ COMMITTED 的查詢	Aurora 複本上搭配 READ COMMITTED 的查詢
T4		如果 Q1 現在完成，則結果為 1,000,000。	如果 Q2 現在完成，則結果為 1,000,000 或 1,000,001。
T5	`DELETE FROM big_table LIMIT 2; COMMIT;`		
T6		如果 Q1 現在完成，則結果為 1,000,000。	如果 Q2 現在完成，則結果為 1,000,000 或 1,000,001 或 999,999 或 999,998。
T7	`UPDATE big_table SET c2 = CONCAT(c2 ,c2,c2); COMMIT;`		
T8		如果 Q1 現在完成，則結果為 1,000,000。	如果 Q2 現在完成，則結果為 1,000,000 或 1,000,001 或 999,999，或者，可能是某個更大的數字。
T9		Q3: `SELECT COUNT(*) FROM big_table;`	Q4: `SELECT COUNT(*) FROM big_table;`
T10		如果 Q3 現在完成，則結果為 999,999。	如果 Q4 現在完成，則結果為 999,999。

時間	Aurora 主要執行個體上的 DML 陳述式	Aurora 主要執行個體上搭配 READ COMMITTED 的查詢	Aurora 複本上搭配 READ COMMITTED 的查詢
T11		Q5: `SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;`	Q6: `SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;`
T12	`INSERT INTO parent_table (id, s) VALUES (1000, 'hello'); INSERT INTO child_table (id, s) VALUES (1000, 'world'); COMMIT;`		
T13		如果 Q5 現在完成，則結果為 0。	如果 Q6 現在完成，則結果為 0 或 1。

如果查詢快速完成，在其他任何交易執行 DML 陳述式和遞交之前，結果是在主要執行個體與 Aurora 複本之間是可預測且相同。從第一個查詢開始詳細檢查行為的差異。

Q1 的結果很容易預測，因為主要執行個體上的 `READ COMMITTED` 使用類似於 `REPEATABLE READ` 隔離層級的強大一致性模式。

隨著查詢執行當時交易是遞交什麼而定，Q2 的結果可能不同。例如，假設查詢執行當時，有其他交易執行 DML 陳述式並遞交。在此情況下，Aurora 複本上搭配 `READ COMMITTED` 隔離層級的查詢可能考量或不考量變更。列計數不能像在 `REPEATABLE READ` 隔離層級下那樣預測。也不像在主要執行個體或 RDS for MySQL 執行個體上於 `READ COMMITTED` 隔離層級下執行的查詢那麼可預測。

T7 上的 `UPDATE` 陳述式實際上不變更資料表的列數。不過，此陳述式能夠變更可變長度欄的長度，導致列在內部重組。長時間執行的 `READ COMMITTED` 交易可能看到某一列的舊版本，而後來在相同查詢

內，可能又看到那同一列的新版本。該查詢也可以同時略過此列的新舊版本，因此列數可能與預期不同。

Q5 和 Q6 的結果可能完全相同或稍微不同。Aurora 複本上在 `READ COMMITTED` 下的查詢 Q6 能夠看到 (但不必一定看到) 查詢執行當時遞交的新列。也可能看到只在一個資料表而不在另一個資料表中的列。如果聯結查詢在兩個資料表中找不到相符一列，則傳回計數為零。如果查詢在 `PARENT_TABLE` 和 `CHILD_TABLE` 中都找到新列，查詢傳回計數為 1。在長時間執行的查詢中，可能間隔很久才查閱聯結的資料表。

> **Note**
>
> 這些行為差異取決於交易遞交的時間，以及查詢何時處理基礎資料表列。因此，在耗費數分鐘或數小時，且在同時處理 OLTP 交易的 Aurora 叢集上所執行的報告查詢中，最可能看到這些差異。Aurora 複本上的 `READ COMMITTED` 隔離層級最有益於這幾種混合工作負載。

Aurora MySQL 提示

您可以將 SQL 提示與 Aurora MySQL 查詢搭配使用，以微調效能。您也可以使用提示來防止重要查詢的執行計劃因不可預期的情況而變更。

> **Tip**
>
> 若要驗證提示對查詢的影響，請檢查 `EXPLAIN` 陳述式所產生的查詢計劃。比較有和沒有提示時的查詢計劃。

在 Aurora MySQL 第 3 版中，您可以使用 MySQL Community Edition 8.0 中提供的所有提示。如需這些提示的詳細資訊，請參閱《MySQL 參考手冊》中的最佳化工具提示。

下列提示可在 Aurora MySQL 第 2 版中取得。這些提示適用於使用 Aurora MySQL 第 2 版中雜湊聯結功能的查詢，尤其是使用平行查詢最佳化的查詢。

PQ，NO_PQ

指定是否強制最佳化工具以每資料表或每查詢為基礎，使用平行查詢。

`PQ` 會強制最佳化工具對指定資料表或整個查詢 (區塊) 使用平行查詢。`NO_PQ` 則會防止最佳化工具對指定資料表或整個查詢 (區塊) 使用平行查詢。

此提示可在 Aurora MySQL 2.11 及更高版本中取得。下列範例示範如何使用此提示。

> **Note**
>
> 若指定資料表名稱，會強制最佳化工具僅在所選資料表套用 PQ/NO_PQ 提示。若未指定資料表名稱，會強制執行受查詢區塊影響的所有資料表之 PQ/NO_PQ 提示。

```
EXPLAIN SELECT /*+ PQ() */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ PQ(t1) */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ PQ(t1,t2) */ f1, f2
    FROM num1 t1, num1 t2 WHERE t1.f1 = t2.f21;

EXPLAIN SELECT /*+ NO_PQ() */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ NO_PQ(t1) */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ NO_PQ(t1,t2) */ f1, f2
    FROM num1 t1, num1 t2 WHERE t1.f1 = t2.f21;
```

HASH_JOIN、NO_HASH_JOIN

開啟或關閉平行查詢最佳化工具，以選擇是否要針對查詢使用雜湊聯結最佳化方法的功能。如果該機制更有效率，則 HASH_JOIN 允許最佳化器使用雜湊聯結。NO_HASH_JOIN 可防止最佳化器使用雜湊聯結進行查詢。此提示可在 Aurora MySQL 2.08 及更高版本中取得。它在 Aurora MySQL 第 3 版中無效。

下列範例示範如何使用此提示。

```
EXPLAIN SELECT/*+ HASH_JOIN(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ NO_HASH_JOIN(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

HASH_JOIN_PROBING、NO_HASH_JOIN_PROBING

在雜湊聯結查詢中，指定是否要使用指定的資料表作為聯結的探查端。查詢會測試建置資料表中的資料欄位值是否存在於探測資料表中，而不是讀取探測資料表的完整內容。您可以使用 HASH_JOIN_PROBING 和 HASH_JOIN_BUILDING 指定處理雜湊聯結查詢的方式，而不需重新排序查詢文字中的資料表。此提示可在 Aurora MySQL 2.08 及更高版本中取得。它在 Aurora MySQL 第 3 版中無效。

下列範例示範如何使用此提示。指定資料表 HASH_JOIN_PROBING 的 T2 提示與指定資料表 NO_HASH_JOIN_PROBING 的 T1 效果相同。

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_PROBING(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_PROBING(t1) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

HASH_JOIN_BUILDING、NO_HASH_JOIN_BUILDING

在雜湊聯結查詢中，指定是否要使用指定的資料表作為聯結的建置端。查詢會處理此資料表中的所有資料列，以建立與其他資料表交互參照的資料欄位值清單。您可以使用 HASH_JOIN_PROBING 和指 HASH_JOIN_BUILDING 定處理雜湊聯結查詢的方式，而不需重新排序查詢文字中的資料表。此提示可在 Aurora MySQL 2.08 及更高版本中取得。它在 Aurora MySQL 第 3 版中無效。

下列範例示範如何使用此提示。指定資料表 HASH_JOIN_BUILDING 的 T2 提示與指定資料表 NO_HASH_JOIN_BUILDING 的 T1 效果相同。

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_BUILDING(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_BUILDING(t1) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

JOIN_FIXED_ORDER

指定查詢中的資料表是根據它們在查詢中列出的順序來聯結。它在涉及三個或更多資料表的查詢時很有用。它旨在替代 MySQL STRAIGHT_JOIN 提示，並等同於 MySQL JOIN_FIXED_ORDER 提示。此提示可在 Aurora MySQL 2.08 及更高版本中取得。

下列範例示範如何使用此提示。

```
EXPLAIN SELECT /*+ JOIN_FIXED_ORDER() */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_ORDER

指定查詢中資料表的聯結順序。它在涉及三個或更多資料表的查詢時很有用。它相當於 MySQL JOIN_ORDER 提示。此提示可在 Aurora MySQL 2.08 及更高版本中取得。

下列範例示範如何使用此提示。

```
EXPLAIN SELECT /*+ JOIN_ORDER (t4, t2, t1, t3) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_PREFIX

指定要在聯結順序中放第一個的資料表。它在涉及三個或更多資料表的查詢時很有用。它相當於 MySQL JOIN_PREFIX 提示。此提示可在 Aurora MySQL 2.08 及更高版本中取得。

下列範例示範如何使用此提示。

```
EXPLAIN SELECT /*+ JOIN_PREFIX (t4, t2) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_SUFFIX

指定要放在聯結順序最後一個的資料表。它在涉及三個或更多資料表的查詢時很有用。它相當於 MySQL JOIN_SUFFIX 提示。此提示可在 Aurora MySQL 2.08 及更高版本中取得。

下列範例示範如何使用此提示。

```
EXPLAIN SELECT /*+ JOIN_SUFFIX (t1) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

若要取得有關使用雜湊聯結查詢的資訊，請參閱 使用雜湊聯結，將大型 Aurora MySQL 聯結查詢最佳化。

Aurora MySQL 預存程序參考

您可以呼叫內建預存程序，以管理您的 Aurora MySQL 資料庫叢集。

主題

- 收集和維護全域狀態歷史記錄
- 設定、啟動和停止二進位日誌 (binlog) 複寫
- 結束工作階段或查詢
- 使用 GTID 複寫交易
- 輪換查詢日誌
- 設定和顯示二進位日誌組態

收集和維護全域狀態歷史記錄

Amazon RDS 會提供一組程序，將在一段時間內快照狀態變數的值，並將它們以及自從上次快照後的任何變更寫入至資料表。此基礎設施稱為「全域狀態歷史記錄」。如需更多詳細資訊，請參閱管理全域狀態歷史記錄。

下列預存程序會管理收集及維護「全域狀態歷史記錄」的方式。

主題

- mysql.rds_collect_global_status_history
- mysql.rds_disable_gsh_collector
- mysql.rds_disable_gsh_rotation
- mysql.rds_enable_gsh_collector
- mysql.rds_enable_gsh_rotation
- mysql.rds_rotate_global_status_history
- mysql.rds_set_gsh_collector
- mysql.rds_set_gsh_rotation

mysql.rds_collect_global_status_history

隨需建立全域狀態歷史記錄的快照。

語法

```
CALL mysql.rds_collect_global_status_history;
```

mysql.rds_disable_gsh_collector

停用全域狀態歷史記錄建立的快照。

語法

```
CALL mysql.rds_disable_gsh_collector;
```

mysql.rds_disable_gsh_rotation

關閉 `mysql.global_status_history` 表格的輪換。

語法

```
CALL mysql.rds_disable_gsh_rotation;
```

mysql.rds_enable_gsh_collector

啟用全域狀態歷史記錄來依據 rds_set_gsh_collector 指定的間隔建立預設快照。

語法

```
CALL mysql.rds_enable_gsh_collector;
```

mysql.rds_enable_gsh_rotation

啟用依 mysql.global_status_history 指定的間隔將 mysql.global_status_history_old 資料表的內容輪換至 rds_set_gsh_rotation。

語法

```
CALL mysql.rds_enable_gsh_rotation;
```

mysql.rds_rotate_global_status_history

隨需將 mysql.global_status_history 資料表的內容輪換至 mysql.global_status_history_old。

語法

```
CALL mysql.rds_rotate_global_status_history;
```

mysql.rds_set_gsh_collector

指定全域狀態歷史記錄建立快照之間的間隔 (以分鐘為單位)。

語法

```
CALL mysql.rds_set_gsh_collector(intervalPeriod);
```

參數

intervalPeriod

快照之間的間隔 (以分鐘為單位)。預設值為 5。

mysql.rds_set_gsh_rotation

指定 mysql.global_status_history 資料表輪換之間的間隔 (以天為單位)。

語法

```
CALL mysql.rds_set_gsh_rotation(intervalPeriod);
```

參數

intervalPeriod

資料表輪換之間的間隔 (以天為單位)。預設值為 7。

設定、啟動和停止二進位日誌 (binlog) 複寫

您可以在連線至 Aurora MySQL 叢集中的主要執行個體時呼叫以下預存程序。這些程序控制了外部資料庫的交易複製到 Aurora MySQL 中，或從 Aurora MySQL 複製到外部資料庫的方式。

主題

- mysql.rds_disable_session_binlog (Aurora MySQL 2 版)
- mysql.rds_enable_session_binlog (Aurora MySQL 2 版)
- mysql.rds_import_binlog_ssl_material
- mysql.rds_next_master_log (Aurora MySQL 版本 2)
- mysql.rds_next_source_log (Aurora MySQL 第 3 版)
- mysql.rds_remove_binlog_ssl_material
- mysql.rds_reset_external_master (Aurora MySQL 第 2 版)
- mysql.rds_reset_external_source (Aurora MySQL 第 3 版)
- mysql.rds_set_binlog_source_ssl (Aurora MySQL 第 3 版)
- mysql.rds_set_external_master (Aurora MySQL 第 2 版)
- mysql.rds_set_external_source (Aurora MySQL 第 3 版)
- mysql.rds_set_external_master_with_auto_position (Aurora MySQL 第 2 版)
- mysql.rds_set_external_source_with_auto_position (Aurora MySQL 第 3 版)
- mysql.rds_set_master_auto_position (Aurora MySQL 第 2 版)
- mysql.rds_set_read_only (Aurora MySQL 第 3 版)
- mysql.rds_set_session_binlog_format (Aurora MySQL 2 版)
- mysql.rds_set_source_auto_position (Aurora MySQL 第 3 版)
- mysql.rds_skip_repl_error
- mysql.rds_start_replication
- mysql.rds_start_replication_until(Aurora MySQL 第 3 版)
- mysql.rds_stop_replication

mysql.rds_disable_session_binlog (Aurora MySQL 2 版)

透過將 `sql_log_bin` 變數設定為 `OFF` 以關閉目前工作階段的二進位記錄。

語法

```
CALL mysql.rds_disable_session_binlog;
```

Parameters

無

使用須知

針對 Aurora MySQL 資料庫叢集，您可在連線至主要執行個體時呼叫此預存程序。

針對 Aurora，Aurora MySQL 2.12 版和更新的相容版本 MySQL 5.7 支援此程序。

> **Note**
>
> 在 Aurora MySQL 第 3 版中，如果您有 SESSION_VARIABLES_ADMIN 權限，您可以使用下列命令來停用目前工作階段的二進制記錄：
>
> ```
> SET SESSION sql_log_bin = OFF;
> ```

mysql.rds_enable_session_binlog (Aurora MySQL 2 版)

透過將 sql_log_bin 變數設定為 ON 以開啟目前工作階段的二進位記錄。

語法

```
CALL mysql.rds_enable_session_binlog;
```

Parameters

無

使用須知

針對 Aurora MySQL 資料庫叢集，您可在連線至主要執行個體時呼叫此預存程序。

針對 Aurora，Aurora MySQL 2.12 版和更新的相容版本 MySQL 5.7 支援此程序。

> **Note**
>
> 在 Aurora MySQL 第 3 版中，如果您有 SESSION_VARIABLES_ADMIN 權限，您可以使用下列命令來啟用目前工作階段的二進制記錄：
>
> ```
> SET SESSION sql_log_bin = ON;
> ```

mysql.rds_import_binlog_ssl_material

將憑證授權單位憑證、用戶端憑證和用戶端金鑰匯入 Aurora MySQL 資料庫叢集。SSL 通訊和加密複寫需要這些資訊。

> **Note**
>
> 目前，下列 Aurora MySQL 版本支援此程序：第二版：2.09.2、2.10.0、2.10.1 和 2.11.0；以及第 3 版：3.01.1 及更新版本。

語法

```
CALL mysql.rds_import_binlog_ssl_material (
  ssl_material
);
```

參數

ssl_material

JSON 承載，其中包含 MySQL 用戶端的以下 .pem 格式檔案的內容：

- "ssl_ca" : "*********"
- "ssl_cert" : "*****"
- "ssl_key" : "*****"

使用須知

執行此程序之前為加密複寫做好準備：

- 如果您在外部 MySQL 來源資料庫執行個體上未啟用 SSL，也沒有準備用戶端金鑰和用戶端憑證，請在 MySQL 資料庫伺服器上啟用 SSL，並產生所需的用戶端金鑰和用戶端憑證。
- 如果外部來源資料庫執行個體上已啟用 SSL，請為 Aurora MySQL 資料庫叢集提供用戶端金鑰和憑證。如果您沒有這些資料，請為 Aurora MySQL 資料庫叢集產生新的金鑰和憑證。若要簽署用戶端憑證，您必須有用於外部 MySQL 來源資料庫執行個體上設定 SSL 的憑證授權單位金鑰。

如需詳細資訊，請參閱 MySQL 文件中的使用 openssl 建立 SSL 憑證和金鑰。

> **Important**
>
> 為加密複寫做好準備之後，使用 SSL 連線來執行此程序。不可透過不安全的連線來傳送用戶端金鑰。

此程序將外部 MySQL 資料庫中的 SSL 資訊匯入 Aurora MySQL 資料庫叢集。SSL 資訊是 .pem 格式檔案，其中包含 Aurora MySQL 資料庫叢集的 SSL 資訊。在加密複寫期間，Aurora MySQL 資料庫叢集充當 MySQL 資料庫伺服器的用戶端。Aurora MySQL 用戶端的憑證和金鑰位於 .pem 格式的檔案中。

您可以將這些檔案中的資訊複製到正確 JSON 承載中的 `ssl_material` 參數。若要支援加密複寫，請將此 SSL 資訊匯入 Aurora MySQL 資料庫叢集。

JSON 承載必須採用下列格式。

```
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
ssl_ca_pem_body_code
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
ssl_cert_pem_body_code
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
ssl_key_pem_body_code
-----END RSA PRIVATE KEY-----\n"}'
```

範例

下列範例將 SSL 資訊匯入 Aurora MySQL。在 .pem 格式檔案中，內文程式碼通常比範例所示的內文程式碼更長。

```
call mysql.rds_import_binlog_ssl_material(
```

```
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

mysql.rds_next_master_log (Aurora MySQL 版本 2)

將來源資料庫執行個體日誌位置變更為來源資料庫執行個體上下一個二進位日誌的開頭。只有當您在僅供讀取複本上收到複寫輸入/輸出錯誤 1236 時，才使用此程序。

語法

```
CALL mysql.rds_next_master_log(
curr_master_log
);
```

參數

curr_master_log

目前主控端日誌檔案的索引。例如，若目前檔案的名稱是 mysql-bin-changelog.012345，則索引為 12345。若要查明目前主控端日誌檔案名稱，請執行 SHOW REPLICA STATUS 命令並檢視 Master_Log_File 欄位。

使用須知

主要使用者必須執行 mysql.rds_next_master_log 程序。

> **Warning**
>
> 只有當複寫來源的異地同步備份資料庫執行個體在容錯移轉之後複寫失敗時，且 `mysql.rds_next_master_log` 的 `Last_IO_Errno` 欄位報告輸入/輸出錯誤 1236，才呼叫 `SHOW REPLICA STATUS`。
> 在容錯移轉事件發生之前，如果來源執行個體中的交易未寫入磁碟上的二進位日誌，則呼叫 `mysql.rds_next_master_log` 會導致僅供讀取複本遺失資料。

範例

假設 Aurora MySQL 僅供讀取複本上的複寫失敗。在僅供讀取複本上執行 `SHOW REPLICA STATUS \G` 將傳回下列結果：

```
*************************** 1. row ***************************
             Replica_IO_State:
                  Source_Host: myhost.XXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Source_Log_Pos: 30223232
              Relay_Log_Space: 5248928866
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Source_SSL_Allowed: No
```

```
           Source_SSL_CA_File:
           Source_SSL_CA_Path:
              Source_SSL_Cert:
            Source_SSL_Cipher:
               Source_SSL_Key:
        Seconds_Behind_Master: NULL
Source_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from master when reading data from
 binary log: 'Client requested master to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Source_Server_Id: 67285976
```

Last_IO_Errno 欄位顯示執行個體收到輸入/輸出錯誤 1236。Master_Log_File 欄位顯示檔案名稱是 mysql-bin-changelog.012345，這表示日誌檔案索引為 12345。若要解決錯誤，您可以呼叫 mysql.rds_next_master_log 並指定下列參數：

```
CALL mysql.rds_next_master_log(12345);
```

mysql.rds_next_source_log (Aurora MySQL 第 3 版)

將來源資料庫執行個體日誌位置變更為來源資料庫執行個體上下一個二進位日誌的開頭。只有當您在僅供讀取複本上收到複寫輸入/輸出錯誤 1236 時，才使用此程序。

語法

```
CALL mysql.rds_next_source_log(
curr_source_log
);
```

Parameters

curr_source_log

目前來源日誌檔案的索引。例如，若目前檔案的名稱是 mysql-bin-changelog.012345，則索引為 12345。若要查明目前來源日誌檔案名稱，請執行 SHOW REPLICA STATUS 命令並檢視 Source_Log_File 欄位。

使用須知

管理使用者必須執行 mysql.rds_next_source_log 程序。

> **Warning**
>
> 只有當複寫來源的異地同步備份資料庫執行個體在容錯移轉之後複寫失敗時，且 mysql.rds_next_source_log 的 Last_IO_Errno 欄位報告輸入/輸出錯誤 1236，才呼叫 SHOW REPLICA STATUS。
>
> 在容錯移轉事件發生之前，如果來源執行個體中的交易未寫入磁碟上的二進位日誌，則呼叫 mysql.rds_next_source_log 會導致僅供讀取複本遺失資料。您可以將來源執行個體參數 sync_binlog 和 innodb_support_xa 設為 1，以降低此情況發生的機率，雖然這麼做會降低效能。

範例

假設 Aurora MySQL 僅供讀取複本上的複寫失敗。在僅供讀取複本上執行 SHOW REPLICA STATUS \G 將傳回下列結果：

```
*************************** 1. row ***************************
             Replica_IO_State:
                  Source_Host: myhost.XXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
```

```
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Source_Log_Pos: 30223232
              Relay_Log_Space: 5248928866
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Source_SSL_Allowed: No
           Source_SSL_CA_File:
           Source_SSL_CA_Path:
              Source_SSL_Cert:
            Source_SSL_Cipher:
               Source_SSL_Key:
        Seconds_Behind_Source: NULL
Source_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from source when reading data from
 binary log: 'Client requested source to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Source_Server_Id: 67285976
```

Last_IO_Errno 欄位顯示執行個體收到輸入/輸出錯誤 1236。Source_Log_File 欄位顯示檔案名稱是 mysql-bin-changelog.012345，這表示日誌檔案索引為 12345。若要解決錯誤，您可以呼叫 mysql.rds_next_source_log 並指定下列參數：

```
CALL mysql.rds_next_source_log(12345);
```

mysql.rds_remove_binlog_ssl_material

移除用於 SSL 通訊和加密複寫的憑證授權單位憑證、用戶端憑證和用戶端金鑰。此資訊是利用 mysql.rds_import_binlog_ssl_material 來匯入。

語法

```
CALL mysql.rds_remove_binlog_ssl_material;
```

mysql.rds_reset_external_master (Aurora MySQL 第 2 版)

將 Aurora MySQL 資料庫執行個體重新設定為不再是 MySQL 執行個體 (在 Amazon RDS 外部執行) 的僅供讀取複本。

> **Important**
>
> 若要執行此程序，必須啟用 `autocommit`。若要啟用它，請將 `autocommit` 參數設定為 1。如需修改參數的相關資訊，請參閱修改 Amazon Aurora 中的資料庫參數群組中的參數。

語法

```
CALL mysql.rds_reset_external_master;
```

使用須知

主要使用者必須執行 `mysql.rds_reset_external_master` 程序。此程序必須在要做為 MySQL 執行個體 (在 Amazon RDS 外部執行) 之僅供讀取複本的將被移除的 MySQL 資料庫執行個體上執行。

> **Note**
>
> 我們提供這些預存程序主要是為了對 Amazon RDS 外部執行的 MySQL 執行個體啟用複寫。如果可能，建議您使用 Aurora 複本來管理 Aurora MySQL 資料庫叢集內的複寫。如需 Aurora MySQL 資料庫叢集中管理複寫的相關資訊，請參閱 使用 Aurora 複本。

如需使用複寫從 Aurora MySQL 外部執行的 MySQL 執行個體匯入資料的詳細資訊，請參閱 Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。

mysql.rds_reset_external_source (Aurora MySQL 第 3 版)

將 Aurora MySQL 資料庫執行個體重新設定為不再是 MySQL 執行個體 (在 Amazon RDS 外部執行) 的僅供讀取複本。

> **Important**
>
> 若要執行此程序，必須啟用 `autocommit`。若要啟用它，請將 `autocommit` 參數設定為 1。如需修改參數的相關資訊，請參閱修改 Amazon Aurora 中的資料庫參數群組中的參數。

語法

```
CALL mysql.rds_reset_external_source;
```

使用須知

管理使用者必須執行 `mysql.rds_reset_external_source` 程序。此程序必須在要做為 MySQL 執行個體 (在 Amazon RDS 外部執行) 之僅供讀取複本的將被移除的 MySQL 資料庫執行個體上執行。

> **Note**
>
> 我們提供這些預存程序主要是為了對 Amazon RDS 外部執行的 MySQL 執行個體啟用複寫。如果可能，建議您使用 Aurora 複本來管理 Aurora MySQL 資料庫叢集內的複寫。如需 Aurora MySQL 資料庫叢集中管理複寫的相關資訊，請參閱 使用 Aurora 複本。

mysql.rds_set_binlog_source_ssl (Aurora MySQL 第 3 版)

針對 binlog 複寫啟用 SOURCE_SSL 加密。如需詳細資訊，請參閱 MySQL 文件中的 CHANGE REPLICATION SOURCE TO 陳述式。

語法

```
CALL mysql.rds_set_binlog_source_ssl(mode);
```

Parameters

##

指出是否啟用 SOURCE_SSL 加密的值：

- `0` – SOURCE_SSL 加密已停用。預設值為 `0`。
- `1` – SOURCE_SSL 加密已啟用。您可以使用 SSL 或 TLS 設定加密。

使用須知

Aurora MySQL 3.04 版及更新版本支援此程序。

mysql.rds_set_external_master (Aurora MySQL 第 2 版)

將 Aurora MySQL 資料庫執行個體設定為 MySQL 執行個體 (在 Amazon RDS 外部執行) 的僅供讀取複本。

`mysql.rds_set_external_master` 程序已棄用，且會在未來版本中將其移除。請改用 `mysql.rds_set_external_source`。

> **Important**
>
> 若要執行此程序，必須啟用 `autocommit`。若要啟用它，請將 `autocommit` 參數設定為 1。如需修改參數的相關資訊，請參閱修改 Amazon Aurora 中的資料庫參數群組中的參數。

語法

```
CALL mysql.rds_set_external_master (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

參數

host_name

要成為來源資料庫執行個體之 MySQL 執行個體 (在 Amazon RDS 外部執行) 的主機名稱或 IP 地址。

host_port

要設定為來源資料庫執行個體之 MySQL 執行個體 (在 Amazon RDS 外部執行) 所使用的連線埠。如果網路組態包含會轉換連線埠號碼的安全殼層 (SSH) 連線埠複寫，請指定 SSH 所公開的連線埠號碼。

replication_user_name

在 Amazon RDS 外部執行的 MySQL 執行個體上具有 REPLICATION CLIENT 和 REPLICATION SLAVE 許可的使用者 ID。我們建議您提供單獨用於外部執行個體複寫的帳戶。

replication_user_password

replication_user_name 中指定之使用者 ID 的密碼。

mysql_binary_log_file_name

來源資料庫執行個體上包含複寫資訊之二進位日誌的名稱。

mysql_binary_log_file_location

複寫在 mysql_binary_log_file_name 二進位日誌中開始讀取複寫資訊的位置。

您可以藉由在來源資料庫執行個體上執行 SHOW MASTER STATUS 來判斷 binlog 檔案名稱和位置。

ssl_encryption

此值指定在複寫連線上是否使用 Secure Socket Layer (SSL) 加密。1 指定使用 SSL 加密，0 指定不使用加密。預設為 0。

> **Note**
>
> 不支援 MASTER_SSL_VERIFY_SERVER_CERT 選項。此選項設定為 0，表示連線已加密，但憑證未經過驗證。

使用須知

主要使用者必須執行 mysql.rds_set_external_master 程序。此程序必須在要設定為 MySQL 執行個體 (在 Amazon RDS 外部執行) 之僅供讀取複本的 MySQL 資料庫執行個體上執行。

在執行 mysql.rds_set_external_master 之前，您必須將 Amazon RDS 外部執行的 MySQL 執行個體設定為來源資料庫執行個體。若要連線至 Amazon RDS 外部執行的 MySQL 執行個體，您必須指定 replication_user_name 和 replication_user_password 值，以指出在外部 MySQL 執行個體上具有 REPLICATION CLIENT 和 REPLICATION SLAVE 許可的複寫使用者。

將外部 MySQL 執行個體設定為來源資料庫執行個體

1. 使用您選擇的 MySQL 用戶端，連線至外部 MySQL 執行個體，並建立用於複寫的使用者帳戶。下列是 範例。

   MySQL 5.7

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
   ```

   MySQL 8.0

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED WITH mysql_native_password BY
    'password';
   ```

   > **Note**
   >
   > 指定此處所顯示提示以外的密碼，作為安全最佳實務。

2. 在外部 MySQL 執行個體上，將 REPLICATION CLIENT 和 REPLICATION SLAVE 權限授予複寫使用者。下列範例將所有資料庫上的 REPLICATION CLIENT 和 REPLICATION SLAVE 權限授予您網域中的 'repl_user' 使用者。

   MySQL 5.7

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
    IDENTIFIED BY 'password';
   ```

   MySQL 8.0

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
   ```

若要使用加密複寫，請將來源資料庫執行個體設定為使用 SSL 連線。此外，使用 mysql.rds_import_binlog_ssl_material 程序，將憑證認證機構憑證、用戶端憑證和用戶端金鑰匯入資料庫執行個體或資料庫叢集。

> **Note**
>
> 我們提供這些預存程序主要是為了對 Amazon RDS 外部執行的 MySQL 執行個體啟用複寫。如果可能，建議您使用 Aurora 複本來管理 Aurora MySQL 資料庫叢集內的複寫。如需 Aurora MySQL 資料庫叢集中管理複寫的相關資訊，請參閱 使用 Aurora 複本。

呼叫 `mysql.rds_set_external_master` 將 Amazon RDS 資料庫執行個體設定為僅供讀取複本之後，您可以在僅供讀取複本上呼叫 mysql.rds_start_replication 來啟動複寫程序。您可以呼叫 mysql.rds_reset_external_master (Aurora MySQL 第 2 版) 來移除僅供讀取複本組態。

呼叫 `mysql.rds_set_external_master` 時，Amazon RDS 將時間、使用者和 `set master` 的動作記錄在 `mysql.rds_history` 和 `mysql.rds_replication_status` 資料表中。

範例

在 MySQL 資料庫執行個體上執行時，下列範例會將資料庫執行個體設定為 MySQL 執行個體 (在 Amazon RDS 外部執行) 的僅供讀取複本。

```
call mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  1);
```

mysql.rds_set_external_source (Aurora MySQL 第 3 版)

將 Aurora MySQL 資料庫執行個體設定為 MySQL 執行個體 (在 Amazon RDS 外部執行) 的僅供讀取複本。

> **Important**
>
> 若要執行此程序，必須啟用 `autocommit`。若要啟用它，請將 `autocommit` 參數設定為 1。如需修改參數的相關資訊，請參閱修改 Amazon Aurora 中的資料庫參數群組中的參數。

語法

```
CALL mysql.rds_set_external_source (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

參數

host_name

要成為來源資料庫執行個體之 MySQL 執行個體 (在 Amazon RDS 外部執行) 的主機名稱或 IP 地址。

host_port

要設定為來源資料庫執行個體之 MySQL 執行個體 (在 Amazon RDS 外部執行) 所使用的連線埠。如果網路組態包含會轉換連線埠號碼的安全殼層 (SSH) 連線埠複寫，請指定 SSH 所公開的連線埠號碼。

replication_user_name

在 Amazon RDS 外部執行的 MySQL 執行個體上具有 REPLICATION CLIENT 和 REPLICATION SLAVE 許可的使用者 ID。我們建議您提供單獨用於外部執行個體複寫的帳戶。

replication_user_password

replication_user_name 中指定之使用者 ID 的密碼。

mysql_binary_log_file_name

來源資料庫執行個體上包含複寫資訊之二進位日誌的名稱。

mysql_binary_log_file_location

複寫在 mysql_binary_log_file_name 二進位日誌中開始讀取複寫資訊的位置。

您可以藉由在來源資料庫執行個體上執行 SHOW MASTER STATUS 來判斷 binlog 檔案名稱和位置。

ssl_encryption

此值指定在複寫連線上是否使用 Secure Socket Layer (SSL) 加密。1 指定使用 SSL 加密，0 指定不使用加密。預設值為 0。

> **Note**
>
> 您必須使用 mysql.rds_import_binlog_ssl_material 匯入自訂 SSL 憑證，才能啟用此選項。如果您尚未匯入自訂 SSL 憑證，請將此參數設為 0，並使用 mysql.rds_set_binlog_source_ssl (Aurora MySQL 第 3 版) 以針對二進制日誌複寫啟用 SSL。
> 不支援 SOURCE_SSL_VERIFY_SERVER_CERT 選項。此選項設定為 0，表示連線已加密，但憑證未經過驗證。

使用須知

管理使用者必須執行 mysql.rds_set_external_source 程序。此程序必須在要設定為 MySQL 執行個體 (在 Amazon RDS 外部執行) 之僅供讀取複本的 Aurora MySQL 資料庫執行個體上執行。

在執行 mysql.rds_set_external_source 之前，您必須將 Amazon RDS 外部執行的 MySQL 執行個體設定為來源資料庫執行個體。若要連線至 Amazon RDS 外部執行的 MySQL 執行個體，您必須指定 replication_user_name 和 replication_user_password 值，以指出在外部 MySQL 執行個體上具有 REPLICATION CLIENT 和 REPLICATION SLAVE 許可的複寫使用者。

將外部 MySQL 執行個體設定為來源資料庫執行個體

1. 使用您選擇的 MySQL 用戶端，連線至外部 MySQL 執行個體，並建立用於複寫的使用者帳戶。下列是 範例。

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
```

> **Note**
>
> 指定此處所顯示提示以外的密碼，作為安全最佳實務。

2. 在外部 MySQL 執行個體上，將 REPLICATION CLIENT 和 REPLICATION SLAVE 權限授予複寫使用者。下列範例將所有資料庫上的 REPLICATION CLIENT 和 REPLICATION SLAVE 權限授予您網域中的 'repl_user' 使用者。

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
```

若要使用加密複寫，請將來源資料庫執行個體設定為使用 SSL 連線。此外，使用 mysql.rds_import_binlog_ssl_material 程序，將憑證授權單位憑證、用戶端憑證和用戶端金鑰匯入資料庫執行個體或資料庫叢集。

> **Note**
>
> 我們提供這些預存程序主要是為了對 Amazon RDS 外部執行的 MySQL 執行個體啟用複寫。如果可能，建議您使用 Aurora 複本來管理 Aurora MySQL 資料庫叢集內的複寫。如需 Aurora MySQL 資料庫叢集中管理複寫的相關資訊，請參閱 使用 Aurora 複本。

呼叫 `mysql.rds_set_external_source` 將 Aurora MySQL 資料庫執行個體設定為僅供讀取複本之後，您可以在僅供讀取複本上呼叫 mysql.rds_start_replication 來啟動複寫程序。您可以呼叫 mysql.rds_reset_external_source (Aurora MySQL 第 3 版) 來移除僅供讀取複本組態。

呼叫 `mysql.rds_set_external_source` 時，Amazon RDS 將時間、使用者和 `set master` 的動作記錄在 `mysql.rds_history` 和 `mysql.rds_replication_status` 資料表中。

範例

在 Aurora MySQL 資料庫執行個體上執行時，下列範例會將資料庫執行個體設定為 MySQL 執行個體 (在 Amazon RDS 外部執行) 的僅供讀取複本。

```
call mysql.rds_set_external_source(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  1);
```

mysql.rds_set_external_master_with_auto_position (Aurora MySQL 第 2 版)

設定 Aurora MySQL 主要執行個體以接受從外部 MySQL 執行個體傳入的複寫。此程序也會依據全域交易識別符 (GTID) 設定複寫。

此程序不會設定延遲複寫，因為 Aurora MySQL 不支援延遲複寫。

語法

```
CALL mysql.rds_set_external_master_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
);
```

參數

host_name

要成為複寫來源之 MySQL 執行個體 (在 Aurora 外部執行) 的主機名稱或 IP 地址。

host_port

要設定為複寫來源之 MySQL 執行個體 (在 Aurora 外部執行) 所使用的連線埠。如果網路組態包含會轉換連線埠號碼的安全殼層 (SSH) 連線埠複寫，請指定 SSH 所公開的連線埠號碼。

replication_user_name

在 Aurora 外部執行的 MySQL 執行個體上具有 `REPLICATION CLIENT` 和 `REPLICATION SLAVE` 許可的使用者 ID。我們建議您提供單獨用於外部執行個體複寫的帳戶。

replication_user_password

`replication_user_name` 中指定之使用者 ID 的密碼。

ssl_encryption

此選項目前尚未實作。預設值為 0。

使用須知

針對 Aurora MySQL 資料庫叢集，您可在連線至主要執行個體時呼叫此預存程序。

主要使用者必須執行 `mysql.rds_set_external_master_with_auto_position` 程序。主要使用者會在做為複寫目標的 Aurora MySQL 資料庫叢集主要執行個體上執行此程序。這可能是外部 MySQL 資料庫執行個體或 Aurora MySQL 資料庫叢集的複本目標。

Aurora MySQL 第 2 版支援此程序。對於 Aurora MySQL 第 3 版，請改用 mysql.rds_set_external_source_with_auto_position (Aurora MySQL 第 3 版)。

在您執行 mysql.rds_set_external_master_with_auto_position 前，請將外部 MySQL 資料庫執行個體設定為複寫來源。如要連線至外部 MySQL 執行個體，請指定 replication_user_name 和 replication_user_password 的值。這些值必須指明具有 MySQL 外部執行個體 REPLICATION CLIENT 和 REPLICATION SLAVE 許可的複寫使用者。

將外部 MySQL 執行個體設定為複寫來源

1. 使用您選擇的 MySQL 用戶端，連線至外部 MySQL 執行個體，並建立用於複寫的使用者帳戶。以下是範例。

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
   ```

2. 若為外部 MySQL 執行個體，請將 REPLICATION CLIENT 和 REPLICATION SLAVE 權限授予您的複寫使用者。下列範例將所有資料庫上的 REPLICATION CLIENT 和 REPLICATION SLAVE 權限授予您網域中的 'repl_user' 使用者。

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
   IDENTIFIED BY 'SomePassW0rd'
   ```

當您呼叫 mysql.rds_set_external_master_with_auto_position 時，Amazon RDS 會記錄特定資訊。此資訊為時間、使用者，以及 "set master" 和 mysql.rds_history 資料表中的 mysql.rds_replication_status 動作。

若要略過已知會導致災難的特定 GTID 型交易，可以使用 mysql.rds_skip_transaction_with_gtid(Aurora MySQL 第 2 版和第 3 版） 預存程序。如需有關依據 GTID 複寫的詳細資訊，請參閱使用 GTID 式複寫。

範例

在 Aurora 主要執行個體上執行時，下列範例會將 Aurora 叢集設定為 MySQL 執行個體 (在 Aurora 外部執行) 的僅供讀取複本。

```
call mysql.rds_set_external_master_with_auto_position(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
```

```
  'SomePassW0rd');
```

mysql.rds_set_external_source_with_auto_position (Aurora MySQL 第 3 版)

設定 Aurora MySQL 主要執行個體以接受從外部 MySQL 執行個體傳入的複寫。此程序也會依據全域交易識別符 (GTID) 設定複寫。

語法

```
CALL mysql.rds_set_external_source_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
);
```

參數

host_name

要成為複寫來源之 MySQL 執行個體 (在 Aurora 外部執行) 的主機名稱或 IP 地址。

host_port

要設定為複寫來源之 MySQL 執行個體 (在 Aurora 外部執行) 所使用的連線埠。如果網路組態包含會轉換連線埠號碼的安全殼層 (SSH) 連線埠複寫，請指定 SSH 所公開的連線埠號碼。

replication_user_name

在 Aurora 外部執行的 MySQL 執行個體上具有 REPLICATION CLIENT 和 REPLICATION SLAVE 許可的使用者 ID。我們建議您提供單獨用於外部執行個體複寫的帳戶。

replication_user_password

replication_user_name 中指定之使用者 ID 的密碼。

ssl_encryption

此選項目前尚未實作。預設值為 0。

> **Note**
>
> 使用 mysql.rds_set_binlog_source_ssl (Aurora MySQL 第 3 版) 以針對二進制日誌複寫啟用 SSL。

使用須知

針對 Aurora MySQL 資料庫叢集，您可在連線至主要執行個體時呼叫此預存程序。

管理使用者必須執行 mysql.rds_set_external_source_with_auto_position 程序。管理使用者會在做為複寫目標的 Aurora MySQL 資料庫叢集主要執行個體上執行此程序。這可能是外部 MySQL 資料庫執行個體或 Aurora MySQL 資料庫叢集的複本目標。

Aurora MySQL 第 3 版支援此程序。此程序不會設定延遲複寫，因為 Aurora MySQL 不支援延遲複寫。

在您執行 mysql.rds_set_external_source_with_auto_position 前，請將外部 MySQL 資料庫執行個體設定為複寫來源。如要連線至外部 MySQL 執行個體，請指定 replication_user_name 和 replication_user_password 的值。這些值必須指明具有 MySQL 外部執行個體 REPLICATION CLIENT 和 REPLICATION SLAVE 許可的複寫使用者。

將外部 MySQL 執行個體設定為複寫來源

1. 使用您選擇的 MySQL 用戶端，連線至外部 MySQL 執行個體，並建立用於複寫的使用者帳戶。以下是範例。

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
   ```

2. 若為外部 MySQL 執行個體，請將 REPLICATION CLIENT 和 REPLICATION SLAVE 權限授予您的複寫使用者。下列範例將所有資料庫上的 REPLICATION CLIENT 和 REPLICATION SLAVE 權限授予您網域中的 'repl_user' 使用者。

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
   IDENTIFIED BY 'SomePassW0rd'
   ```

當您呼叫 mysql.rds_set_external_source_with_auto_position 時，Amazon RDS 會記錄特定資訊。此資訊為時間、使用者，以及 "set master" 和 mysql.rds_history 資料表中的 mysql.rds_replication_status 動作。

若要略過已知會導致災難的特定 GTID 型交易，可以使用 mysql.rds_skip_transaction_with_gtid(Aurora MySQL 第 2 版和第 3 版） 預存程序。如需有關依據 GTID 複寫的詳細資訊，請參閱使用 GTID 式複寫。

範例

在 Aurora 主要執行個體上執行時，下列範例會將 Aurora 叢集設定為 MySQL 執行個體 (在 Aurora 外部執行) 的僅供讀取複本。

```
call mysql.rds_set_external_source_with_auto_position(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'SomePassW0rd');
```

mysql.rds_set_master_auto_position (Aurora MySQL 第 2 版)

將複寫模式設為依據二進制日誌檔案位置或全域交易識別符 (GTID)。

語法

```
CALL mysql.rds_set_master_auto_position (
auto_position_mode
);
```

參數

auto_position_mode

此值指示要使用日誌檔案位置複寫或是 GTID 複寫：

- 0 – 使用依據二進制日誌檔案位置的複寫模式。預設值為 0。
- 1 – 使用依據 GTID 的複寫方法。

使用須知

主要使用者必須執行 mysql.rds_set_master_auto_position 程序。

Aurora MySQL 第 2 版支援此程序。

mysql.rds_set_read_only (Aurora MySQL 第 3 版)

針對資料庫執行個體全域開啟或關閉 read_only 模式。

語法

```
CALL mysql.rds_set_read_only(mode);
```

Parameters

##

指出針對資料庫執行個體全域開啟或關閉 read_only 模式的值：

- 0 – OFF。預設值為 0。
- 1 – ON

使用須知

mysql.rds_set_read_only 預存程序只會修改 read_only 參數。innodb_read_only 參數無法在讀取器資料庫執行個體上變更。

read_only 參數變更不會在重新啟動時保留。若要對 read_only 進行永久變更，您必須使用 read_only 資料庫叢集參數。

Aurora MySQL 3.04 版及更新版本支援此程序。

mysql.rds_set_session_binlog_format (Aurora MySQL 2 版)

設定目前工作階段的二進位日誌格式。

語法

```
CALL mysql.rds_set_session_binlog_format(format);
```

Parameters

format

指示目前工作階段的二進位日誌格式的值：

- STATEMENT – 複寫來源會根據 SQL 陳述式將事件寫入二進位日誌。

- ROW – 複寫來源會將事件寫入二進位日誌，指示個別資料表資料列的變更。
- MIXED – 日誌記錄通常以 SQL 陳述式為基礎，但在特定條件下會切換至資料列。如需詳細資訊，請參閱 MySQL 文件中的混合式二進位日誌格式。

使用須知

針對 Aurora MySQL 資料庫叢集，您可在連線至主要執行個體時呼叫此預存程序。

若要使用這個預存程序，您必須為目前的工作階段設定二進位日誌。

針對 Aurora，Aurora MySQL 2.12 版和更新的相容版本 MySQL 5.7 支援此程序。

mysql.rds_set_source_auto_position (Aurora MySQL 第 3 版)

將複寫模式設為依據二進制日誌檔案位置或全域交易識別符 (GTID)。

語法

```
CALL mysql.rds_set_source_auto_position (auto_position_mode);
```

參數

auto_position_mode

此值指示要使用日誌檔案位置複寫或是 GTID 複寫：

- 0 – 使用依據二進制日誌檔案位置的複寫模式。預設值為 0。
- 1 – 使用依據 GTID 的複寫方法。

使用須知

針對 Aurora MySQL 資料庫叢集，您可在連線至主要執行個體時呼叫此預存程序。

管理使用者必須執行 mysql.rds_set_source_auto_position 程序。

mysql.rds_skip_repl_error

略過和刪除 MySQL 資料庫僅供讀取複本上的複寫錯誤。

語法

```
CALL mysql.rds_skip_repl_error;
```

使用須知

主要使用者必須在僅供讀取複本上執行 mysql.rds_skip_repl_error 程序。如需有關此程序的詳細資訊，請參閱略過目前的複寫錯誤。

若要判斷是否有錯誤，執行 MySQL SHOW REPLICA STATUS\G 命令。如果複寫錯誤不嚴重，您可以執行 mysql.rds_skip_repl_error 來略過錯誤。如果有多個錯誤，mysql.rds_skip_repl_error 會刪除第一個錯誤，然後警告還有其他錯誤。然後，您可以使用 SHOW REPLICA STATUS\G，以針對下一個錯誤判斷正確的行動步驟。如需傳回值的相關資訊，請參閱 MySQL 文件中的 SHOW REPLICA STATUS 陳述式。

如需有關解決 Aurora MySQL 複寫錯誤的詳細資訊，請參閱 診斷和解決 MySQL 讀取複寫失敗。

複寫已停止錯誤

當您呼叫 mysql.rds_skip_repl_error 程序時，可能會收到錯誤訊息，指出複本已關閉或停用。

如果您在主要執行個體而非僅供讀取複本上執行程序，此錯誤訊息就會出現。您必須在僅供讀取複本上執行此程序，程序才能運作。

如果您在僅供讀取複本上執行程序，但複寫無法成功重新啟動，此錯誤訊息也可能出現。

如果您需要略過大量錯誤，複寫延遲可能增加至超出二進位日誌(binlog) 檔案的預設保留期間。在此情況下，由於在清除 binlog 檔案之前已在僅供讀取複本上重播該檔案，您可能會遇到嚴重錯誤。此清除動作會導致複寫停止，而您將無法再呼叫 mysql.rds_skip_repl_error 命令來略過複寫錯誤。

透過增加 binlog 檔案在來源資料庫執行個體上保留的小時數，即可以減輕此問題。在延長二進位記錄檔保留時間之後，您可以重新啟動複寫，並視需要呼叫 mysql.rds_skip_repl_error 命令。

若要設定 binlog 保留時間，請使用 mysql.rds_set_configuration 程序，並指定 'binlog retention hours'組態參數加上資料庫叢集上保留 binlog 檔案的時數。下列範例會將 binlog 檔案的保留期間設定為 48 小時。

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

mysql.rds_start_replication

從 Aurora MySQL 資料庫叢集起始複寫。

> **Note**
>
> 您可使用 mysql.rds_start_replication_until(Aurora MySQL 第 3 版) 或 mysql.rds_start_replication_until_gtid(Aurora MySQL 第 3 版) 預存程序從 Aurora MySQL 資料庫執行個體來啟動複寫，並從特定的二進位日誌檔案位置停止複寫。

語法

```
CALL mysql.rds_start_replication;
```

使用須知

主要使用者必須執行 mysql.rds_start_replication 程序。

如果要從 Amazon RDS 外部的 MySQL 執行個體匯入資料，請呼叫 mysql.rds_set_external_master (Aurora MySQL 第 2 版) 或 mysql.rds_set_external_source (Aurora MySQL 第 3 版) 來建置複寫組態，再呼叫僅供讀取複本上的 mysql.rds_start_replication 來啟動複寫程序。如需詳細資訊，請參閱Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。

若要將資料匯出至 Amazon RDS 外部的 MySQL 執行個體，請在僅供讀取複本上呼叫 mysql.rds_start_replication 和 mysql.rds_stop_replication 來控制某些複寫動作，例如清除二進位日誌。如需詳細資訊，請參閱Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。

您也可以對僅供讀取複本呼叫 mysql.rds_start_replication，以重新啟動您先前呼叫 mysql.rds_stop_replication 所停止的任何複寫程序。如需詳細資訊，請參閱複寫已停止錯誤。

mysql.rds_start_replication_until(Aurora MySQL 第 3 版)

從 Aurora MySQL 資料庫叢集啟動複寫，並從特定的二進位日誌檔案位置停止複寫。

語法

```
CALL mysql.rds_start_replication_until (
replication_log_file
  , replication_stop_point
);
```

參數

replication_log_file

來源資料庫執行個體上包含複寫資訊之二進位日誌的名稱。

replication_stop_point

在 replication_log_file 二進位日誌中的複寫將停止的位置。

使用須知

主要使用者必須執行 mysql.rds_start_replication_until 程序。

Aurora MySQL 3.04 版及更新版本支援此程序。

受管複寫不支援 mysql.rds_start_replication_until 預存程序，其中包括下列項目：

- 跨 AWS 區域 複寫 Amazon Aurora MySQL 資料庫叢集
- 使用 Aurora 讀取複本，從 RDS for MySQL 資料庫執行個體將資料遷移至 Amazon Aurora MySQL 資料庫叢集

replication_log_file 參數的指定檔名必須與來源資料庫執行個體 binlog 檔案的名稱相同。

當 replication_stop_point 參數指定了過去的一個停止位置，複寫即會立即停止。

範例

以下範例會啟動複寫並複寫變更，直到達到 120 二進位日誌檔案中的位置 mysql-bin-changelog.000777 為止。

```
call mysql.rds_start_replication_until(
  'mysql-bin-changelog.000777',
  120);
```

mysql.rds_stop_replication

從 MySQL 資料庫執行個體停止複寫。

語法

```
CALL mysql.rds_stop_replication;
```

使用須知

主要使用者必須執行 mysql.rds_stop_replication 程序。

如果您要將複寫設定為從 Amazon RDS 外部執行的 MySQL 執行個體匯入資料，在匯入完成之後，請在僅供讀取複本上呼叫 mysql.rds_stop_replication 來停止複寫程序。如需詳細資訊，請參閱Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。

如果您要設定複寫將資料匯出至 Amazon RDS 外部的 MySQL 執行個體，請在僅供讀取複本上呼叫 mysql.rds_start_replication 和 mysql.rds_stop_replication 來控制某些複寫動作，例如清除二進位日誌。如需詳細資訊，請參閱Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。

受管複寫不支援 mysql.rds_stop_replication 預存程序，其中包括下列項目：

- 跨 AWS 區域 複寫 Amazon Aurora MySQL 資料庫叢集
- 使用 Aurora 讀取複本，從 RDS for MySQL 資料庫執行個體將資料遷移至 Amazon Aurora MySQL 資料庫叢集

結束工作階段或查詢

下列預存程序會結束工作階段或查詢。

主題

- mysql.rds_kill
- mysql.rds_kill_query

mysql.rds_kill

結束 MySQL 伺服器的連線。

語法

```
CALL mysql.rds_kill(processID);
```

參數

processID

要結束之連線執行緒的身分。

使用須知

MySQL 伺服器的每個連線都在個別執行緒中執行。若要結束連線，請使用 `mysql.rds_kill` 程序並傳入該連線的執行緒 ID。若要取得執行緒 ID，請使用 MySQL SHOW PROCESSLIST 命令。

範例

下列範例結束執行緒 ID 為 4243 的連線：

```
CALL mysql.rds_kill(4243);
```

mysql.rds_kill_query

結束對 MySQL 伺服器執行的查詢。

語法

```
CALL mysql.rds_kill_query(processID);
```

參數

processID

正在執行要結束之查詢的處理序或執行緒的身分。

使用須知

若要停止對 MySQL 伺服器執行的查詢，請使用 mysql_rds_kill_query 程序並傳入執行查詢之執行緒的連線 ID。然後程序就會終止連線。

若要取得 ID，請查詢 MySQL INFORMATION_SCHEMA PROCESSLIST 資料表或使用 MySQL SHOW PROCESSLIST 命令。SHOW PROCESSLIST 或 SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST 中 ID 欄的值為 *processID*。

範例

下列範例會停止查詢執行緒 ID 為 230040 的查詢：

```
CALL mysql.rds_kill_query(230040);
```

使用 GTID 複寫交易

下列預存程序控制如何使用全域交易識別符 (GTID) 搭配 Aurora MySQL 複寫交易。如要了解透過 Aurora MySQL 使用以 GTID 為基礎的複寫使用方式，請參閱 使用 GTID 式複寫。

主題

- mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL 第 3 版)
- mysql.rds_gtid_purged (Aurora MySQL 3 版)
- mysql.rds_skip_transaction_with_gtid(Aurora MySQL 第 2 版和第 3 版)
- mysql.rds_start_replication_until_gtid(Aurora MySQL 第 3 版)

mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL 第 3 版)

配置 `CHANGE REPLICATION SOURCE TO` 陳述式的 `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` 選項。它會使複寫通道將 GTID 指派給沒有 GTID 的複寫交易。如此一來，您就可以將二進位日誌從不使用 GTID 型複寫的來源複寫到使用該複寫的複本。如需詳細資訊，請參閱《MySQL 參考手冊》中的 CHANGE REPLICATION SOURCE TO 陳述式和從沒有 GTID 的來源複寫到具有 GTID 的複本。

語法

```
CALL mysql.rds_assign_gtids_to_anonymous_transactions(gtid_option);
```

Parameters

gtid_option

字串值。允許的值為 `OFF`、`LOCAL` 或指定的 UUID。

使用須知

此程序的效果與在社群 MySQL 中發出陳述式 `CHANGE REPLICATION SOURCE TO ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS = gtid_option` 相同。

GTID 必須轉換為 `ON`，*gtid_option* 才能設定為 `LOCAL` 或特定的 UUID。

預設為 `OFF`，表示不使用該功能。

`LOCAL` 會指派 GTID，其中包含複本自己的 UUID (`server_uuid` 設定)。

傳遞的參數是 UUID，則會指派一個 UUID，其中包含指定的 GTID，例如複寫來源伺服器的 `server_uuid` 設定。

範例

若要關閉此功能：

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('OFF');
+-------------------------------------------------------------+
| Message  |
+-------------------------------------------------------------+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: OFF |
+-------------------------------------------------------------+
1 row in set (0.07 sec)
```

若要使用複本自己的 UUID：

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('LOCAL');
+--------------------------------------------------------------+
| Message  |
+--------------------------------------------------------------+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: LOCAL |
+--------------------------------------------------------------+
1 row in set (0.07 sec)
```

若要使用指定的 UUID：

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('317a4760-
f3dd-3b74-8e45-0615ed29de0e');
+-----------------------------------------------------------------------------------------------------------
+
| Message |
+-----------------------------------------------------------------------------------------------------------
+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: 317a4760-
f3dd-3b74-8e45-0615ed29de0e |
+-----------------------------------------------------------------------------------------------------------
+
1 row in set (0.07 sec)
```

mysql.rds_gtid_purged (Aurora MySQL 3 版)

將系統變數 gtid_purged 的全域值設定為特定全域交易識別碼 (GTID) 設定。gtid_purged 系統變數是一個 GTID 集，由伺服器上已提交但不存在於伺服器上的任何二進位日誌檔中的所有 GTID 交易組成。

為了與 MySQL 8.0 相容，有兩種方法可以設定 gtid_purged 的值：

- 將 gtid_purged 的值取代為指定的 GTID 設定。
- 將指定的 GTID 集附加至已包含 gtid_purged 的 GTID 設定。

語法

若要使用指定的 GTID 設定取代 gtid_purged 的值：

```
CALL mysql.rds_gtid_purged (gtid_set);
```

若要將指定的 GTID 設定附加至 gtid_purged 的值：

```
CALL mysql.rds_gtid_purged (+gtid_set);
```

Parameters

gtid_set

gtid_set 的值必須是目前 gtid_purged 值的超集，且不能與 gtid_subtract(gtid_executed,gtid_purged) 相交。也就是說，新的 GTID 集必須包含已在 gtid_purged 中的任何 GTID，並且不能在 gtid_executed 中包含任何 GTID 尚未清除的項目。*gtid_set* 參數也不能包含全域中的任何 GTID gtid_owned 設定，目前正在伺服器上處理交易的 GTID。

使用須知

主要使用者必須執行 mysql.rds_gtid_purged 程序。

Aurora MySQL 3.04 版及更新版本支援此程序。

範例

下列範例會指派 GTID 3E11FA47-71CA-11E1-9E33-C80AA9429562:23 到 gtid_purged 全域變數。

```
CALL mysql.rds_gtid_purged('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_skip_transaction_with_gtid(Aurora MySQL 第 2 版和第 3 版)

略過 Aurora 主要執行個體上具有指定全域交易識別碼 (GTID) 之交易的複寫。

若已知特定 GTID 交易導致錯誤，可以使用此程序進行災難復原。使用此預存程序來略過有問題的交易。有問題的交易範例包括停用複寫、刪除重要資料或導致資料庫執行個體無法使用的交易。

語法

```
CALL mysql.rds_skip_transaction_with_gtid (
gtid_to_skip
);
```

Parameters

gtid_to_skip

要略過的複寫交易的 GTID。

使用須知

主要使用者必須執行 mysql.rds_skip_transaction_with_gtid 程序。

Aurora MySQL 第 2 版和第 3 版支援此程序。

範例

下列範例會略過使用 GTID 3E11FA47-71CA-11E1-9E33-C80AA9429562:23 進行交易的複寫。

```
CALL mysql.rds_skip_transaction_with_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_start_replication_until_gtid(Aurora MySQL 第 3 版)

從 Aurora MySQL 資料庫叢集啟動複寫，並在指定的全域交易識別碼 (GTID) 之後立即停止複寫。

語法

```
CALL mysql.rds_start_replication_until_gtid(gtid);
```

Parameters

gtid

在此 GTID 後停止複寫。

使用須知

主要使用者必須執行 mysql.rds_start_replication_until_gtid 程序。

Aurora MySQL 3.04 版及更新版本支援此程序。

受管複寫不支援 mysql.rds_start_replication_until_gtid 預存程序，其中包括下列項目：

- 跨 AWS 區域 複寫 Amazon Aurora MySQL 資料庫叢集
- 使用 Aurora 讀取複本，從 RDS for MySQL 資料庫執行個體將資料遷移至 Amazon Aurora MySQL 資料庫叢集

當 gtid 參數指定了複本已經執行的交易時，複寫會立即停止。

範例

以下範例會啟動複寫並複寫變更，直到達到 GTID 3E11FA47-71CA-11E1-9E33-C80AA9429562:23 為止。

```
call mysql.rds_start_replication_until_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

輪換查詢日誌

下列預存程序會將 MySQL 日誌輪換為備份資料表。如需更多詳細資訊，請參閱 Aurora MySQL資料庫日誌檔案。

主題

- mysql.rds_rotate_general_log
- mysql.rds_rotate_slow_log

mysql.rds_rotate_general_log

將 mysql.general_log 資料表輪換至備份資料表。

語法

```
CALL mysql.rds_rotate_general_log;
```

使用須知

您可以呼叫 mysql.general_log 程序，將 mysql.rds_rotate_general_log 資料表輪換至備份資料表。輪換日誌資料表時，目前日誌資料表會複製到備份日誌資料表，並移除目前日誌資料表中的項目。如果備份日誌資料表已存在，則其會在目前日誌資料表複製到備份之前遭到刪除。如有需要，您可以查詢備份日誌資料表。mysql.general_log 資料表的備份日誌資料表名為 mysql.general_log_backup。

當 log_output 參數設定為 TABLE 時，您僅可執行此程序。

mysql.rds_rotate_slow_log

將 mysql.slow_log 資料表輪換至備份資料表。

語法

```
CALL mysql.rds_rotate_slow_log;
```

使用須知

您可以呼叫 mysql.slow_log 程序，將 mysql.rds_rotate_slow_log 資料表輪換至備份資料表。輪換日誌資料表時，目前日誌資料表會複製到備份日誌資料表，並移除目前日誌資料表中的項目。如果備份日誌資料表已存在，則其會在目前日誌資料表複製到備份之前遭到刪除。

如有需要，您可以查詢備份日誌資料表。mysql.slow_log 資料表的備份日誌資料表名為 mysql.slow_log_backup。

設定和顯示二進位日誌組態

下列預存程序會設定並顯示組態參數，例如二進位日誌檔案保留。

主題

- mysql.rds_set_configuration
- mysql.rds_show_configuration

mysql.rds_set_configuration

指定保留二進位日誌的小時數，或延遲複寫的秒數。

語法

```
CALL mysql.rds_set_configuration(name,value);
```

Parameters

name

要設定之組態參數的名稱。

value

組態參數的值。

使用須知

`mysql.rds_set_configuration` 程序支援下列組態參數：

- binlog 保留 (小時)

組態參數會永久存放，且在任何資料庫執行個體重新啟動或容錯移轉後依然存在。

binlog 保留 (小時)

`binlog retention hours` 參數用於指定保留二進位日誌檔的小時數。Amazon Aurora 通常會儘快清除二進位日誌，但複寫 Aurora 外部的 MySQL 資料庫時可能仍需要二進位日誌。

`binlog retention hours` 的預設值為 NULL。對於 Aurora MySQL，NULL 意味著二進位日誌被延遲清理。Aurora MySQL 二進位日誌可能會在系統中保留一段時間，通常不會超過一天。

若要指定在資料庫叢集上保留二進位日誌的時數，請使用 mysql.rds_set_configuration 預存程序，並指定讓複寫有足夠時間進行的期間，如下列範例所示。

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

> **Note**
>
> 不可針對 binlog retention hours 使用值 0。

對於 Aurora MySQL 2.11.0 版和更新版本，以及第 3 版資料庫叢集，最大 binlog retention hours 值是 2160 (90 天)。

設定保留期間之後，請監控資料庫執行個體的儲存體用量，確定保留的二進位日誌沒有佔用太多儲存體。

mysql.rds_show_configuration

保留二進位日誌的時數。

語法

```
CALL mysql.rds_show_configuration;
```

使用須知

若要驗證 Amazon RDS 保留二進位日誌的時數，請使用 mysql.rds_show_configuration 預存程序。

範例

下列範例顯示保留期間：

```
call mysql.rds_show_configuration;
                name                          value     description
                binlog retention hours        24        binlog retention hours specifies
 the duration in hours before binary logs are automatically deleted.
```

Aurora MySQL 特定的 information_schema 資料表

Aurora MySQL 具有 Aurora 專屬的特定 information_schema 資料表。

information_schema.aurora_global_db_instance_status

`information_schema.aurora_global_db_instance_status` 資料表包含全球資料庫主要和次要資料庫叢集中所有資料庫執行個體狀態的相關資訊。下列資料表顯示您可以使用的資料欄。其餘的資料欄僅供 Aurora 內部使用。

> **Note**
>
> Aurora MySQL 3.04.0 版及更新版本的全球資料庫才能使用此資訊結構描述表。

資料行	資料類型	描述
SERVER_ID	varchar(100)	資料庫執行個體的識別符。
SESSION_ID	varchar(100)	目前工作階段的唯一識別符。`MASTER_SESSION_ID` 值可識別寫入器 (主) 資料庫執行個體。
AWS_REGION	varchar(100)	此全球資料庫執行個體執行所在的 AWS 區域。如需區域清單，請參閱 區域可用性。
DURABLE_LSN	bigint unsigned	可長期儲存的日誌序號 (LSN)。記錄序號 (LSN) 是一組獨特的序號，可用來識別資料庫交易日誌中的記錄。系統會排序 LSN，而 LSN 越大，就表示交易發生時間越後面。
HIGHEST_LSN_RCVD	bigint unsigned	資料庫執行個體從寫入器資料庫執行個體接收的最高 LSN。
OLDEST_READ_VIEW_TRX_ID	bigint unsigned	寫入器資料庫執行個體可清除到最舊的交易 ID。
OLDEST_READ_VIEW_LSN	bigint unsigned	資料庫執行個體從儲存體中讀取資料時所使用的最舊 LSN。

資料行	資料類型	描述
VISIBILITY_LAG_IN_MSEC	浮點 (10,0) 不帶正負號	對於主要資料庫叢集中的讀取器，此資料庫執行個體延遲於寫入器資料庫執行個體的時間 (以毫秒為單位)。對於次要資料庫叢集中的讀取器，此資料庫執行個體延遲於次要磁碟區的時間 (以毫秒為單位)。

information_schema.aurora_global_db_status

`information_schema.aurora_global_db_status` 資料表包含有關 Aurora 全球資料庫延遲各方面的資訊，特別是基礎 Aurora 儲存的延遲 (稱為持久性延遲) 及復原點目標 (RPO) 之間的延遲。下列資料表顯示您可以使用的資料欄。其餘的資料欄僅供 Aurora 內部使用。

> **Note**
>
> Aurora MySQL 3.04.0 版及更新版本的全球資料庫才能使用此資訊結構描述表。

資料行	資料類型	描述
AWS_REGION	varchar(100)	此全球資料庫執行個體執行所在的 AWS 區域。如需區域清單，請參閱 區域可用性。
HIGHEST_LSN_WRITTEN	bigint unsigned	目前存在於此資料庫叢集上的最高日誌序號 (LSN)。記錄序號 (LSN) 是一組獨特的序號，可用來識別資料庫交易日誌中的記錄。系統會排序 LSN，而 LSN 越大，就表示交易發生時間越後面。
DURABILITY_LAG_IN_MILLISECONDS	浮點 (10,0) 不帶正負號	次要資料庫叢集上的 HIGHEST_LSN_WRITTE

資料行	資料類型	描述
		N 與主要資料庫叢集上的 HIGHEST_LSN_WRITTEN 之間的時間戳記值差異。在 Aurora 全球資料庫的主要資料庫叢集上，此值始終為 0。
RPO_LAG_IN_MILLISECONDS	浮點 (10,0) 不帶正負號	復原點目標 (RPO) 延遲。RPO 延遲是指最近使用者交易 COMMIT 在儲存於 Aurora 全域資料庫的主資料庫叢集之後，將其儲存於次要資料庫叢集上所需的時間。在 Aurora 全球資料庫的主要資料庫叢集上，此值始終為 0。 簡言之，此指標會計算 Aurora 全球資料庫中每個 Aurora MySQL 資料庫叢集的復原點目標，亦即，若有中斷發生，可能會遺失的資料量。與延遲一樣，RPO 是以時間來衡量。
LAST_LAG_CALCULATION_TIMESTAMP	datetime	指出上次為 DURABILITY_LAG_IN_MILLISECONDS 和 RPO_LAG_IN_MILLISECONDS 計算值的時間戳記。例如 1970-01-01 00:00:00+00 之類的時間值表示此為主要資料庫叢集。
OLDEST_READ_VIEW_TRX_ID	bigint unsigned	寫入器資料庫執行個體可清除到最舊的交易 ID。

information_schema.replica_host_status

`information_schema.replica_host_status` 資料表包含複寫資訊。下表中顯示您可以使用的資料欄。其餘的資料欄僅供 Aurora 內部使用。

資料行	資料類型	描述
CPU	double	複本主機的 CPU 使用率。
IS_CURRENT	tinyint	複本是否為最新的。
LAST_UPDATE_TIMESTAMP	datetime(6)	最新更新發生的時間。用來判斷記錄是否過時。
REPLICA_LAG_IN_MILLISECONDS	double	複本延遲 (毫秒)
SERVER_ID	varchar(100)	資料庫伺服器的 ID。
SESSION_ID	varchar(100)	資料庫工作階段的 ID。用來判斷資料庫執行個體是寫入器還是讀取器執行個體。

> **Note**
>
> 當複本執行個體落後時，從其 `information_schema.replica_host_status` 資料表查詢的資訊可能已過期。在此情況下，建議您改從寫入器執行個體進行查詢。
> 雖然 `mysql.ro_replica_status` 資料表具有類似的資訊，但不建議您使用它。

information_schema.aurora_forwarding_processlist

`information_schema.aurora_forwarding_processlist`資料表包含有關寫入轉送涉及的程序資訊。

只有在已開啟全域或叢集內寫入轉送的資料庫叢集的寫入器資料庫執行個體上，才能看到此資料表的內容。讀取器資料庫執行個體上會傳回空白結果。

欄位	資料類型	描述
ID	bigint	寫入器資料庫執行個體上連線的識別碼。此識別符與 SHOW PROCESSLIST 陳述式中 Id 欄位顯示的值相同，並由執行緒中的 CONNECTION_ID() 函數傳回。
USER	varchar(32)	發出陳述式的 MySQL 使用者。
HOST	varchar(255)	發出陳述式的 MySQL 用戶端。對於轉寄的陳述式，此欄位會顯示在轉送讀取器資料庫執行個體上建立連線的應用程式用戶端主機位址。
DB	varchar(64)	執行緒的預設資料庫。
命令	varchar(16)	如果工作階段處於閒置狀態，執行緒代表用戶端執行的命令類型，或 Sleep。如需執行緒命令的說明，請參閱 MySQL 文件上的 MySQL 文件中的執行緒命令值。
TIME	int	執行緒處於目前狀態的時間 (以秒為單位)。
STATE	varchar(64)	動作、事件或狀態，指出執行緒正在執行什麼動作。如需狀態值的描述，請參閱 MySQL 文件中的一般執行續狀態。
INFO	longtext	如果沒有執行陳述式，則執行緒正在執行的陳述式，或 NULL。陳述式可能是傳送至伺服器的陳述式，或者如果陳述式執行其他陳述式，則是最內層的陳述式。
IS_FORWARDED	bigint	指示執行緒是否從讀取器資料庫執行個體轉送。
REPLICA_SESSION_ID	bigint	Aurora 複本上的連線識別碼。此識別碼與轉送 Aurora 讀取器資料庫執行個體 SHOW PROCESSLIST 陳述式 Id 欄位顯示的值相同。
REPLICA_INSTANCE_IDENTIFIER	varchar(64)	轉送執行緒的資料庫執行個體識別符。

欄位	資料類型	描述
REPLICA_CLUSTER_NAME	varchar(64)	轉送執行緒的資料庫叢集識別符。對於叢集內的寫入轉送，此識別碼與寫入器資料庫執行個體的資料庫叢集相同。
REPLICA_REGION	varchar(64)	AWS 區域 轉送執行續的來源。對於叢集內的寫入轉送，此區域與寫入器資料庫執行個體的 AWS 區域 相同。

Amazon Aurora MySQL 的資料庫引擎更新

Amazon Aurora 版本會定期更新。更新會在系統維護時段套用到 Aurora 資料庫叢集。套用更新的時間取決於資料庫叢集的區域和維護時段，以及更新類型。

Amazon Aurora 版本在多天時間內可供所有 AWS 區域使用。某些區域可能會暫時顯示其他區域尚無法使用的引擎版本。

更新會同時套用到資料庫叢集的所有執行個體。更新時，資料庫叢集的所有執行個體上需要重新啟動資料庫，因此您會經歷 20 到 30 秒的停機時間，之後就可以繼續使用資料庫叢集。您可以從 AWS 管理主控台檢視或變更您的維護時段設定。

有關 Amazon Aurora 支援的 Aurora MySQL 版本的詳細內容，請參閱 Aurora MySQL 版本備註。

以下，您可以了解如何為叢集選擇正確的 Aurora MySQL 版本、如何在建立或升級叢集時指定版本，以及如何以最少的中斷將叢集從某個版本升級到另一個版本的程序。

主題

- 檢查 Aurora MySQL 版本編號
- Amazon Aurora MySQL 的長期支援 (LTS) 和 Beta 版
- 為 Amazon Aurora MySQL 相容版本第 2 版標準支援結束做好準備
- 為 Amazon Aurora MySQL 相容版本第 1 版結束生命週期做好準備
- 升級 Amazon Aurora MySQL 資料庫叢集
- Amazon Aurora MySQL 的資料庫引擎更新和修正

檢查 Aurora MySQL 版本編號

雖然 Aurora MySQL 相容版本 與 MySQL 資料庫引擎相容，但 Aurora MySQL 包含 Aurora MySQL 版本特有的功能和錯誤修正。應用程式開發人員可以使用 SQL 檢查其應用程式中的 Aurora MySQL 版本。資料庫管理員可以在建立或升級 Aurora MySQL 資料庫叢集和資料庫執行個體時檢查和指定 Aurora MySQL 版本。

主題

- 透過 AWS 檢查或指定 Aurora MySQL 引擎版本
- 使用 SQL 檢查 Aurora MySQL 版本

透過 AWS 檢查或指定 Aurora MySQL 引擎版本

當您使用 AWS 管理主控台、AWS CLI 或 RDS API 執行管理工作時，請以描述性英數字元格式指定 Aurora MySQL 版本。

從 Aurora MySQL 第 2 版開始，Aurora 引擎版本具有下列語法。

```
mysql-major-version.mysql_aurora.aurora-mysql-version
```

mysql-major-version- 部分為 5.7 或 8.0。此值代表用戶端通訊協定的版本，以及對應 Aurora MySQL 版本的 MySQL 功能支援的一般等級。

aurora-mysql-version 是有三個部分的虛線值：Aurora MySQL 主要版本、Aurora MySQL 次要版本及修補程式層級。主要版本為 2 或 3。這些值代表 Aurora MySQL 分別與 MySQL 5.7 或 8.0 相容。次要版本代表 2.x 或 3.x 系列中的功能版本。每個次要版本的修補程式等級會從 0 開始，並代表套用至次要版本的後續錯誤修正。有時候，新功能會併入次要版本，但不會立即顯示。在這些情況下，功能會進行微調，並在稍後的修補程式等級中公開。

所有 2.x Aurora MySQL 引擎版本都和 Community MySQL 5.7.12 或更新版本具備線路相容性。所有 3.x Aurora MySQL 引擎版本都和 MySQL 8.0.23 或更新版本具備線路相容性。您可參考特定 3.x 版的版本備註，找出對應的 MySQL 相容版本。

例如，Aurora MySQL 3.04.0 和 2.11.2 的引擎版本如下所示。

```
8.0.mysql_aurora.3.04.0
5.7.mysql_aurora.2.11.2
```

> **Note**
>
> 社群 MySQL 版本與 Aurora MySQL 2.x 版本之間沒有一對一的對應關係。對於 Aurora MySQL 第 3 版，有一個更直接的映射。若要檢查特定 Aurora MySQL 版本中有哪些錯誤修正和新功能，請參閱 Aurora MySQL 版本備註中的 Amazon Aurora MySQL 第 3 版的資料庫引擎更新和 Amazon Aurora MySQL 第 2 版的資料庫引擎更新。如需按時間順序排列的新功能和版本清單，請參閱 文件歷史記錄。若要檢查安全性相關修正所需的最低版本，請參閱 Aurora MySQL 版本備註中的 Aurora MySQL 中修復的安全漏洞。

在某些 AWS CLI 命令和 RDS API 操作中，請指定 Aurora MySQL 引擎版本。例如，當您執行 `--engine-version` 命令 create-db-cluster 和 modify-db-cluster 時，請指定 AWS CLI 選項。當您執行 RDS API 操作 CreateDBCluster 和 ModifyDBCluster 時，請指定 `EngineVersion` 參數。

在 Aurora MySQL 第 2 版和更新版本中，AWS 管理主控台 中的引擎版本也會包含 Aurora 版本。升級叢集會變更顯示的值。這項變更可協助您指定並檢查精確的 Aurora MySQL 版本，而不需要連線到叢集或執行任何 SQL 命令。

> **Tip**
>
> 對於透過 CloudFormation 所管理的 Aurora 叢集，`EngineVersion` 設定中的這項變更會觸發 CloudFormation 執行動作。如需有關 CloudFormation 如何處理 `EngineVersion` 設定變更的資訊，請參閱 CloudFormation 文件。

使用 SQL 檢查 Aurora MySQL 版本

您可以使用 SQL 查詢在應用程式中擷取的 Aurora 版本編號會使用格式 *<major version>*.*<minor version>*.*<patch version>*。透過查詢 `AURORA_VERSION` 系統變數，您可以取得 Aurora MySQL 叢集中任何資料庫執行個體的此版本編號。若要取得此版本編號，請使用下列查詢之一。

```
select aurora_version();
select @@aurora_version;
```

這些查詢會產生類似下列的輸出。

```
mysql> select aurora_version(), @@aurora_version;
+------------------+------------------+
| aurora_version() | @@aurora_version |
+------------------+------------------+
| 3.05.2           | 3.05.2           |
+------------------+------------------+
```

主控台、CLI 和 RDS API 使用 透過 AWS 檢查或指定 Aurora MySQL 引擎版本 中所述技術傳回的版本編號通常更具描述性。

Amazon Aurora MySQL 的長期支援 (LTS) 和 Beta 版

Aurora MySQL 為某些 Aurora MySQL 引擎版本提供長期支援 (LTS) 和 Beta 版。

主題

- Aurora MySQL 長期支援 (LTS) 版本
- Aurora MySQL Beta 版

Aurora MySQL 長期支援 (LTS) 版本

當您建立或升級資料庫叢集時，每個新的 Aurora MySQL 版本皆可為您保留特定的時間供使用。在這段期間過後，您必須升級該叢集使用的版本。您可以在支援期間結束前手動升級叢集，或 Aurora 可幫您在 Aurora MySQL 版本不再支援時自動升級。

Aurora 指定特定的 Aurora MySQL 版本做為長期支援 (LTS) 版本。使用 LTS 版本的資料庫叢集相較於非 LTS 版本，可以在相同版本上停留更長的時間，且可以進行幾次升級週期。使用 LTS 版本的資料庫叢集可以保留在相同的次要版本上至少三年，或直到主要版本的標準支援結束，以先到者為準。當 LTS 版本上的資料庫叢集需要升級時，Aurora 會將其升級至下一個 LTS 版本。透過這種方式，叢集版本不需要花費長時間再次升級。

在 Aurora MySQL LTS 版本存留期之間，會發布新的修補層級修正重要的問題。修補層級不會包含任何新的功能。您可以選擇是否套用修補程式至執行 LTS 版本的資料庫叢集執行。我們建議執行 LTS 版本的客戶至少每年升級到 LTS 次要版本的最新修補程式版本一次，以利用高嚴重性的安全性和操作修正。針對特定的關鍵修正，Amazon 可能會在相同的 LTS 版本中執行受管升級達到某個修補層級。此受管升級在叢集維護時段內自動執行。所有 Aurora MySQL 版本 (包含 LTS 和非 LTS 版本) 都經過廣泛的穩定性和操作測試。選取次要版本會指定為 LTS 版本，讓客戶可以在不升級至較新的次要版本的情況下，更長的時間保留在這些次要版本上。

針對大部分 Aurora MySQL 叢集，我們建議您升級至最新版本，而不是使用 LTS 版本。如此，可利用 Aurora 做為管理服務並提供您存取最新版本的功能和錯誤修正。LTS 版本適用於具有下列特性的叢集：

- 除了極少發生的關鍵修補程式之外，您無法承擔 Aurora MySQL 應用程式為了升級而停機所帶來的後果。
- 每次 Aurora MySQL 資料庫引擎更新時，您的叢集和應用程式相關連的測試週期都需要較長的時間。
- 您的 Aurora MySQL 叢集資料庫版本有所有應用程式需要的資料庫引擎功能和錯誤修正。

Aurora MySQL 的目前 LTS 發行版本如下：

- Aurora MySQL 版本 3.10.*。

- Aurora MySQL 版本 3.04.*。

如需 LTS 版本的詳細資訊，請參閱《Aurora MySQL 版本備註》中的 Aurora MySQL 第 3 版的資料庫引擎更新。

> **Note**
>
> 建議您停用 LTS 版本的自動次要版本升級。將 `AutoMinorVersionUpgrade` 參數設定為 `false`，或清除 AWS 管理主控台上的啟用自動次要版本升級核取方塊。
> 如果您不停用它，您的資料庫叢集可能會升級至非 LTS 版本。

Aurora MySQL Beta 版

Aurora MySQL Beta 版是有限數量的早期、僅安全性修正版本 AWS 區域。這些修正之後將隨下一個修補程式版本更廣泛部署至所有區域。

Beta 版的編號類似於 Aurora MySQL 次要版本，但具有額外的第四位數，例如 2.12.0.1 或 3.05.0.1。

如需詳細資訊，請參閱 Aurora MySQL 版本備註中的 Amazon Aurora MySQL 第 2 版的資料庫引擎更新和 Amazon Aurora MySQL 第 3 版的資料庫引擎更新

為 Amazon Aurora MySQL 相容版本第 2 版標準支援結束做好準備

Amazon Aurora MySQL 相容版本第 2 版 (與 MySQL 5.7 相容) 計劃於 2024 年 10 月 31 日結束標準支援。我們建議您在 Aurora MySQL 第 2 版的標準支援期結束之前，將執行 Aurora MySQL 第 2 版的所有叢集都升級到預設的 Aurora MySQL 第 3 版 (與 MySQL 8.0 相容) 或更高版本。在 2024 年 10 月 31 日，Amazon RDS 會自動將您的資料庫註冊到 Amazon RDS 延長支援。如果您在 Aurora Serverless 第 1 版叢集中執行 Amazon Aurora MySQL 第 2 版 (與 MySQL 5.7 相容)，則這不適用於您。如果您想要將 Aurora Serverless 第 1 版叢集升級至 Aurora MySQL 第 3 版，請參閱 Aurora Serverless v1 資料庫叢集的升級路徑。

您可以在 Aurora MySQL 主要版本的發佈行事曆中找到 Aurora MySQL 主要版本的近期終止支援日期。

如果您有執行 Aurora MySQL 第 2 版的叢集，標準支援結束日期快到時，您會定期收到通知，其中包含有關如何進行升的最新資訊。我們將定期更新此頁面以提供最新資訊。

標準支援結束時間表

1. 即日起至 2024 年 10 月 31 日 – 您可以將叢集從 Aurora MySQL 第 2 版 (與 MySQL 5.7 相容) 升級至 Aurora MySQL 第 3 版 (與 MySQL 8.0 相容)。
2. 2024 年 10 月 31 日 – 在此日期，Aurora MySQL 第 2 版將終止標準支援，Amazon RDS 會自動將您的叢集註冊到 Amazon RDS 延長支援。

我們將自動為您註冊 RDS 延長支援。如需詳細資訊，請參閱使用 Amazon Aurora 的 Amazon RDS 延伸支援。

尋找受此生命週期結束程序影響的叢集

若要尋找受此生命週期結束程序影響的叢集，請使用以下程序。

> **Important**
>
> 請務必在 AWS 帳戶 資源所在的每個 AWS 區域 和 中執行這些指示。

主控台

尋找 Aurora MySQL 第 2 版叢集

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 在依資料庫篩選方塊中，輸入 5.7。
4. 在引擎列中檢查 Aurora MySQL。

AWS CLI

若要使用 尋找受此end-of-life程序影響的叢集 AWS CLI，請呼叫 describe-db-clusters 命令。您可以使用以下指令範例。

Example

```
aws rds describe-db-clusters --include-share --query 'DBClusters[?(Engine==`aurora-
mysql` && contains(EngineVersion,`5.7.mysql_aurora`))].{EngineVersion:EngineVersion,
```

```
DBClusterIdentifier:DBClusterIdentifier, EngineMode:EngineMode}' --output table
--region us-east-1

       +-----------------------------------------------------------------+
       |                       DescribeDBClusters                        |
       +---------------------+---------------+-------------------------+
       |         DBCI        |      EM       |          EV             |
       +---------------------+---------------+-------------------------+
       |    aurora-mysql2    |  provisioned  | 5.7.mysql_aurora.2.11.3 |
       | aurora-serverlessv1 |   serverless  | 5.7.mysql_aurora.2.11.3 |
       +---------------------+---------------+-------------------------+
```

RDS API

若要尋找執行 Aurora MySQL 第 2 版的 Aurora MySQL 資料庫叢集，請使用 RDS DescribeDBClusters API 操作和以下必要參數：

- `DescribeDBClusters`
  - Filters.Filter.N
    - 名稱
      - engine
    - Values.Value.N
      - ['aurora']

Amazon RDS 延長支援

您可以免費透過社群 MySQL 5.7 使用 Amazon RDS 延長支援，直到支援結束日期 2024 年 10 月 31 日為止。在 2024 年 10 月 31 日，Amazon RDS 會自動將您的資料庫註冊到 Aurora MySQL 第 2 版的 RDS 延長支援。適用於 Aurora 的 RDS 延長支援是付費服務，可為 Aurora MySQL 第 2 版提供額外長達 28 個月的支援，直到 2027 年 2 月的 RDS 延長支援結束為止。RDS 延長支援僅適用於 Aurora MySQL 次要版本 2.11 和 2.12。若要在標準支援結束後使用 Amazon Aurora MySQL 第 2 版，請規劃在 2024 年 10 月 31 日之前在其中一個次要版本上執行您的資料庫。

如需 RDS 延長支援的詳細資訊，例如費用和其他考量，請參閱 使用 Amazon Aurora 的 Amazon RDS 延伸支援。

執行升級

在主要版本之間進行升級，需要比次要版本更廣泛的規劃和測試。該程序可能需要大量時間。我們要把升級視為包含三步驟的流程，包括升級前、升級中和升級後的活動。

升級前：

在升級之前，我們建議您檢查升級後叢集的應用程式相容性、效能、維護程序和類似的考量，以確認升級後您的應用程式能如預期運作。以下五項建議有助於為您提供更好的升級體驗。

- 首先，了解 Aurora MySQL 主要版本就地升級的運作方式 至關重要。
- 接下來，探索 從 Aurora MySQL 第 2 版升級到第 3 版 時可用的升級技術。
- 為了協助您決定正確的升級時間和方法，您可以了解 Aurora MySQL 第 3 版與目前環境搭配 比較 Aurora MySQL 第 2 版和 Aurora MySQL 第 3 版 之間的差異：
- 在您決定了方便且效果最佳的選項後，請使用 規劃 Aurora MySQL 叢集的主要版本升級，嘗試在複製的叢集上模擬就地升級。
- 請參閱Aurora MySQL 的主要版本升級預先檢查。升級預先檢查程式可以執行並確定您的資料庫是否可以成功升級、升級後是否存在任何應用程式不相容問題，以及效能、維護程序和類似的考量。
- 並非所有種類或版本的 Aurora MySQL 叢集都可以使用就地升級機制。如需詳細資訊，請參閱Aurora MySQL 主要版本升級路徑。

如果您有任何問題或疑慮，可在社群論壇和 Premium Support 上取得 AWS 支援團隊。

進行升級：

您可以使用下列其中一項升級技術：您的系統將經歷的停機時間取決於所選擇的技術。

- 藍綠部署 – 對於首要目標是減少應用程式停機時間的情況，您可以使用 Amazon RDS 藍/綠部署，在佈建的 Amazon Aurora 資料庫叢集中執行主要版本升級。藍/綠部署會建立一個複製生產環境的預備環境。您可以對綠色 (預備) 環境中的 Aurora 資料庫叢集進行某些變更，而不會影響生產工作負載。轉換通常只需不到一分鐘的時間，不會遺失資料。如需詳細資訊，請參閱Amazon Aurora 藍/綠部署概觀。這樣可將停機時間降到最低，但需要您在執行升級時執行其他資源。
- 就地升級 – 您可以執行就地升級，Aurora 會自動為您執行預先檢查程序、將叢集離線、備份叢集、執行升級並使叢集重新連線。只要按幾下滑鼠即可執行就地主要版本升級，不需要與其他叢集進行其他協調或容錯移轉，但確實會涉及停機時間。如需詳細資訊，請參閱就地升級執行方式
- 快照還原 – 您可以透過從 Aurora MySQL 第 2 版快照還原到 Aurora MySQL 第 3 版叢集，來升級 Aurora MySQL 第 2 版叢集。為此，您應該遵循拍攝快照並從中還原的過程進行操作。這個程序涉及資料庫中斷，因為您是從快照還原。

升級後：

升級之後，您需要密切監控您的系統 (應用程式和資料庫)，並在必要時進行微調。嚴格遵循升級前步驟，可將所需的變更降到最低。如需詳細資訊，請參閱對 Amazon Aurora MySQL 資料庫效能進行故障診斷。

若要進一步了解 Aurora MySQL 主要版本升級的方法、計畫、測試和疑難排解，請務必徹底閱讀 升級 Amazon Aurora MySQL 資料庫叢集的主要版本，包含 Aurora MySQL 就地升級疑難排解。另請注意，Aurora MySQL 第 3 版不支援某些執行個體類型。如需詳細資訊，請參閱Amazon Aurora 資料庫執行個體類別。

Aurora Serverless v1 資料庫叢集的升級路徑

在主要版本之間進行升級，需要比次要版本更廣泛的規劃和測試。該程序可能需要大量時間。我們要把升級視為包含三步驟的流程，包括升級前、升級中和升級後的活動。

Aurora MySQL 第 2 版 (與 MySQL 5.7 相容) 將繼續獲得 Aurora Serverless v1 叢集的標準支援。

如果您想要升級到 Amazon Aurora MySQL 3 (與 MySQL 8.0 相容) 並繼續執行 Aurora Serverless，您可以使用 Amazon Aurora Serverless v2。若要了解 Aurora Serverless v1 和 Aurora Serverless v2 之間的差異，請參閱 比較 Aurora Serverless v2 和 Aurora Serverless v1。

升級到 Aurora Serverless v2：您可以將 Aurora Serverless v1 叢集升級到 Aurora Serverless v2。如需詳細資訊，請參閱從 Aurora Serverless v1 叢集升級至 Aurora Serverless v2。

為 Amazon Aurora MySQL 相容版本第 1 版結束生命週期做好準備

Amazon Aurora MySQL 相容版本第 1 版 (與 MySQL 5.6 相容) 預計將於 2023 年 2 月 28 日結束生命週期。Amazon 建議您將執行 Aurora MySQL 第 1 版的所有叢集 (已佈建的和 Aurora Serverless) 升級至 Aurora MySQL 第 2 版 (與 MySQL 5.7 相容) 或 Aurora MySQL 第 3 版 (與 MySQL 8.0 相容)。在 Aurora MySQL 第 1 版結束支援前執行此操作。

若為 Aurora 佈建的資料庫叢集，您可以透過多種方法完成從 Aurora MySQL 第 1 版升級至 Aurora MySQL 第 2 版。如需就地升級機制的相關說明，請參閱 就地升級執行方式。另一種完成升級的方法，是擷取 Aurora MySQL 第 1 版叢集的快照，然後將此快照還原至 Aurora MySQL 第 2 版叢集。您也以遵循並排執行新舊叢集的多步驟程序。如需每種方法的詳細資訊，請參閱 升級 Amazon Aurora MySQL 資料庫叢集的主要版本。

若為 Aurora Serverless v1 資料庫叢集，您可以從 Aurora MySQL 第 1 版就地升級至 Aurora MySQL 第 2 版。如需這種方法的詳細資訊，請參閱 修改 Aurora Serverless v1 資料庫叢集。

若為 Aurora 佈建的資料庫叢集，您可以使用兩階段升級程序，完成從 Aurora MySQL 第 1 版升級至 Aurora MySQL 第 3 版：

1. 使用上述方法，從 Aurora MySQL 第 1 版升級至 Aurora MySQL 第 2 版。
2. 從 Aurora MySQL 第 2 版升級到 Aurora MySQL 第 3 版，其升級方法與從第 1 版升級到第 2 版的方法相同。如需詳細資訊，請參閱從 Aurora MySQL 第 2 版升級到第 3 版。請記下 Aurora MySQL 第 2 版與第 3 版之間的功能差異。

如需了解 Aurora 主要版本即將到來之生命週期結束日期，請參閱 Amazon Aurora 版本。Amazon 會自動升級您未在生命週期結束日期前自行升級的任何叢集。在生命週期結束日期之後，系統會在叢集的排程維護時段內進行這些自動升級至後續主版本的工作。

以下是升級即將結束生命週期之 Aurora MySQL 第 1 版叢集 (已佈建的和 Aurora Serverless) 的其他里程碑。每次升級的開始時間是國際標準時間 (UTC) 00:00。

1. 從現在到 2023 年 2 月 28 日：您隨時可以開始將 Aurora MySQL 第 1 版 (與 MySQL 5.6 相容) 叢集升級至 Aurora MySQL 第 2 版 (與 MySQL 5.7 相容)。若為 Aurora MySQL 第 2 版，您可以進一步為 Aurora 佈建的資料庫叢集升級至 Aurora MySQL 第 3 版 (與 MySQL 8.0 相容)。
2. 2023 年 1 月 16 日：在此時間之後，您便無法從 AWS 管理主控台 或 AWS Command Line Interface (AWS CLI) 建立新的 Aurora MySQL 第 1 版叢集或執行個體。您也無法將多個次要區域新增至 Aurora 全域資料庫。這可能會影響您從意外停機中復原的能力 (如 從計劃外中斷復原 Amazon Aurora 全域資料庫 所述)，因為此後您便無法完成步驟 5 和 6。您還將無法建立執行 Aurora MySQL 版本 1 的新跨區域僅供讀取複本。在 2023 年 2 月 28 日之前，您仍可對現有的 Aurora MySQL 第 1 版叢集執行以下操作：
   - 將擷取的 Aurora MySQL 第 1 版叢集快照還原至與原始快照叢集相同的版本。
   - 新增僅供讀取複本 (不適用於 Aurora Serverless 資料庫叢集)。
   - 變更執行個體組態。
   - 執行時間點還原。
   - 建立現有第 1 版叢集的複本。
   - 建立執行 Aurora MySQL 版本 2 或更高版本的新跨區域僅供讀取複本。
3. 2023 年 2 月 28 日：在此時間之後，我們預計會在隨後的排程維護時段內自動將 Aurora MySQL 第 1 版叢集升級至 Aurora MySQL 第 2 版的預設版本。還原 Aurora MySQL 第 1 版資料庫快照，會導致將還原的叢集自動升級至當時 Aurora MySQL 第 2 版的預設版本。

在主要版本之間進行升級，需要比次要版本更廣泛的規劃和測試。該程序可能需要大量時間。

對於首要目標是減少停機時間的情況，您也可以使用藍/綠部署，在佈建的 Amazon Aurora 資料庫叢集中執行主要版本升級。藍/綠部署會建立一個複製生產環境的預備環境。您可以對綠色 (預備) 環境中的 Aurora 資料庫叢集進行變更，而不會影響生產工作負載。切換通常只需不到一分鐘的時間，不會遺失資料，也不需要變更應用程式。如需更多詳細資訊，請參閱 Amazon Aurora 藍/綠部署概觀。

升級完成後，您還可能需要進行後續工作。例如，由於 SQL 相容性、某些 MySQL 相關功能的工作方式或新舊版本之間的參數設定存在差異，您可能需要進行後續工作。

若要進一步了解 Aurora MySQL 主要版本升級的方法、計畫、測試和疑難排解，請務必徹底閱讀 升級 Amazon Aurora MySQL 資料庫叢集的主要版本。

尋找受此生命週期結束程序影響的叢集

若要尋找受此生命週期結束程序影響的叢集，請使用以下程序。

> **Important**
>
> 請務必在您的資源所在的每個 AWS 區域 和 AWS 帳戶 中執行這些指示。

主控台

尋找 Aurora MySQL 第 1 版叢集

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 在 Filter by databases (依資料庫篩選) 方塊中，輸入 5.6。
4. 在引擎列中檢查 Aurora MySQL。

AWS CLI

若要使用 AWS CLI 尋找受此生命週期結束程序影響的叢集，呼叫 describe-db-clusters 命令。您可以使用以下指令範例。

Example

```
aws rds describe-db-clusters --include-share --query 'DBClusters[?Engine==`aurora`].
{EV:EngineVersion, DBCI:DBClusterIdentifier, EM:EngineMode}' --output table --region
 us-east-1
```

```
+-----------------------------------------+
|            DescribeDBClusters           |
+---------------+--------------+----------+
|     DBCI      |     EM       |    EV    |
+---------------+--------------+----------+
|  my-database-1|  serverless  |  5.6.10a |
+---------------+--------------+----------+
```

RDS API

若要尋找執行 Aurora MySQL 第 1 版的 Aurora MySQL 資料庫叢集，請使用 RDS DescribeDBClusters API 操作和以下必要參數：

- `DescribeDBClusters`
  - Filters.Filter.N
    - 名稱
      - engine
    - Values.Value.N
      - ['aurora']

升級 Amazon Aurora MySQL 資料庫叢集

您可以升級 Aurora MySQL 資料庫叢集以取得錯誤修正、新的 Aurora MySQL 功能，或變更為全新版本的基礎資料庫引擎。下列各節說明如何執行此操作。

> **Note**
>
> 您進行的升級類型取決於您可以為叢集提供多少停機時間、計劃進行多少驗證測試、特定錯誤修正或新功能對您的使用案例有多重要，以及您是否計劃經常進行小型升級，還是偶爾會略過若干中間版本。針對每次升級，您可以變更叢集的主要版本、次要版本和修補程式層級。如果您不熟悉 Aurora MySQL 主要版本、次要版本和修補程式層級之間的區別，可以在此閱讀背景資訊 檢查 Aurora MySQL 版本編號。

> **Tip**
>
> 您可以使用藍/綠部署，將資料庫叢集升級所需的停機時間降至最低。如需更多詳細資訊，請參閱 使用 Amazon Aurora 藍/綠部署進行資料庫更新。

主題

- 升級 Aurora MySQL 資料庫叢集的次要版本或修補程式層級
- 升級 Amazon Aurora MySQL 資料庫叢集的主要版本

升級 Aurora MySQL 資料庫叢集的次要版本或修補程式層級

您可以使用下列方法來升級資料庫叢集的次要版本或修補資料庫叢集：

- 透過修改引擎版本升級 Aurora MySQL (適用於 Aurora MySQL 第 2 版和第 3 版)
- 啟用次要 Aurora MySQL 版本之間的自動升級

如需零停機時間修補如何在升級程序期間減少中斷的相關資訊，請參閱使用零停機時間修補。

如需為 Aurora MySQL 資料庫叢集執行次要版本升級的相關資訊，請參閱下列主題。

主題

- 執行次要版本升級之前
- Aurora MySQL 的次要版本升級預先檢查
- 透過修改引擎版本升級 Aurora MySQL
- 啟用次要 Aurora MySQL 版本之間的自動升級
- 使用零停機時間修補
- 替代藍/綠升級技術

執行次要版本升級之前

建議您執行下列動作，以減少次要版本升級期間的停機時間：

- Aurora 資料庫叢集維護應在低流量期間執行。使用 Performance Insights 來識別這些時段，以正確設定維護時段。如需 Performance Insights 的詳細資訊，請參閱在 Amazon RDS 上使用

Performance Insights 監控資料庫負載。如需資料庫叢集維護時段的詳細資訊，請參閱 調整偏好的資料庫叢集維護時段。

- 最佳實務是支援指數退避和抖動 AWS SDKs。如需詳細資訊，請參閱指數退避和抖動。

Aurora MySQL 的次要版本升級預先檢查

當您開始次要版本升級時，Amazon Aurora 會自動執行預先檢查。

系統會強制執行這些前置檢查，您無法選擇略過這些檢查。前置檢查提供以下優勢：

- 升級期間可避免非預期的停機時間。
- 出現不相容情況時，Amazon Aurora 即會防止系統進行升級，並提供相關日誌讓您了解。如此，您就可以使用這些日誌來減少不相容情況，為資料庫做好升級的準備。如需移除不相容的詳細資訊，請參閱 MySQL 文件中的準備您的安裝進行升級。

前置檢查會在系統將資料庫執行個體停止以進行升級前執行，意即前置檢查執行期間不會造成任何停機時間。如果預先檢查找到不相容，Aurora 會在資料庫執行個體停止前自動取消升級。Aurora 也會為不相容產生事件。如需 Amazon Aurora 事件的詳細資訊，請參閱 使用 Amazon RDS 事件通知。

Aurora 會在日誌檔案 `PrePatchCompatibility.log` 中記錄每個不相容的相關詳細資訊。在多數情況下，日誌項目包含修正不相容的 MySQL 文件連結。如需檢視日誌檔案的詳細資訊，請參閱檢視並列出資料庫日誌檔案。

根據前置檢查的特性，這些檢查作業會分析資料庫中的物件。此分析會耗用資源，並增加升級完成的時間。

透過修改引擎版本升級 Aurora MySQL

升級 Aurora MySQL 資料庫叢集的次要版本會將額外的修正程式和新功能套用至現有叢集。

此類升級適用於原始版本和升級版本皆有相同 Aurora MySQL 主要版本 2 或 3 的 Aurora MySQL 叢集。此程序既快速又直接，因為它不涉及 Aurora MySQL 中繼資料的任何轉換或資料表資料的重組。

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 修改資料庫叢集的引擎版本，以執行此類升級。例如，如果您的叢集正在執行 Aurora MySQL 3.x，請選擇較高的 3.x 版本。

如果要針對 Aurora 全球資料庫執行次要升級，請先升級所有次要叢集，再升級主要叢集。

> **Note**
>
> 若要執行次要版本升級至 Aurora MySQL 3.04.* 版及更新版本，或 2.12 * 版，請遵循下列操作：
>
> 1. 從全域叢集移除所有次要區域。請遵循 從 Amazon Aurora 全域資料庫中移除叢集 中的步驟。
> 2. 將主要區域的引擎版本升級至 3.04.* 或更新版本，或 2.12.* 版 (如適用)。請遵循 To modify the engine version of a DB cluster 中的步驟。
> 3. 將次要區域新增至全域叢集。請遵循 將 AWS 區域 新增到 Amazon Aurora 全域資料庫 中的步驟。

修改資料庫叢集的引擎版本

- 使用主控台 – 修改叢集的屬性。在 Modify DB cluster (修改資料庫叢集) 視窗中，變更 DB engine version (資料庫引擎版本) 方塊中的 Aurora MySQL 引擎版本。如果您不熟悉修改叢集的一般程序，請遵循 使用主控台、CLI 和 API 修改資料庫叢集 中的指示。
- 透過使用 AWS CLI – 呼叫 modify-db-cluster AWS CLI 命令，並在 `--db-cluster-identifier` 選項中指定資料庫叢集的名稱，在 `--engine-version` 選項中指定引擎版本。

  例如，若要升級到 Aurora MySQL 3.04.1 版，請將 `--engine-version` 選項設為 `8.0.mysql_aurora.3.04.1`。指定 `--apply-immediately` 選項以立即更新資料庫叢集的引擎版本。
- 使用 RDS API – 呼叫 ModifyDBCluster API 操作，並在 `DBClusterIdentifier` 參數中指定資料庫叢集的名稱，在 `EngineVersion` 參數中指定引擎版本。將 `ApplyImmediately` 參數設為 `true`，以立即更新資料庫叢集的引擎版本。

啟用次要 Aurora MySQL 版本之間的自動升級

對於 Amazon Aurora MySQL 資料庫叢集，您可以指定 Aurora 會自動將資料庫叢集升級為新的次要版本。您可以透過設定資料庫叢集的 `AutoMinorVersionUpgrade` 屬性 (AWS 管理主控台 中的自動次要版本升級) 來執行此操作。

自動升級會在維護時段期間進行。如果資料庫叢集中的個別資料庫執行個體維護時段不同於叢集維護時段，則會優先考慮叢集維護時段。

自動次要版本升級不適用於下列類型的 Aurora MySQL 叢集：

- 屬於 Aurora 全域資料庫一部分的叢集
- 具有跨區域複本的叢集

中斷持續時間視工作負載、叢集大小、二進位記錄資料的數量，以及 Aurora 是否可使用零停機時間修補 (ZDP) 功能而定。Aurora 會重新啟動資料庫叢集，因此在繼續使用叢集之前，您可能會遇到短暫無法使用的狀況。特別是二進位日誌資料的數量會影響復原時間。資料庫執行個體會在復原期間處理二進位記錄資料。因此，大量的二進位記錄資料會增加復原時間。

> **Note**
>
> Aurora 僅在資料庫叢集中的所有資料庫執行個體皆已啟用 AutoMinorVersionUpgrade 設定時才會執行自動升級。如需了解如何設定它，以及套用於叢集和執行個體層級時如何運作，請參閱 Aurora 資料庫叢集的自動次要版本升級。
> 然後，如果資料庫叢集執行個體的升級路徑存在，而次要資料庫引擎版本將 AutoUpgrade 設為 true，則會進行升級。AutoUpgrade 設定是動態的，並由 RDS 設定。
> 自動次要版本的升級會執行至預設次要版本。

您可以使用下列 CLI 命令來檢查 Aurora MySQL 叢集中所有資料庫執行個體的AutoMinorVersionUpgrade升級設定狀態。

```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

此命令會產生類似下列的輸出：

```
[
  {
      "DBInstanceIdentifier": "db-t2-medium-instance",
      "DBClusterIdentifier": "cluster-57-2020-06-03-6411",
      "AutoMinorVersionUpgrade": true
  },
  {
      "DBInstanceIdentifier": "db-t2-small-original-size",
      "DBClusterIdentifier": "cluster-57-2020-06-03-6411",
      "AutoMinorVersionUpgrade": false
  },
  {
```

```
      "DBInstanceIdentifier": "instance-2020-05-01-2332",
      "DBClusterIdentifier": "cluster-57-2020-05-01-4615",
      "AutoMinorVersionUpgrade": true
  },
... output omitted ...
```

在此範例中，資料庫叢集 `cluster-57-2020-06-03-6411` 的啟用自動次要版本升級已關閉，因為叢集的其中一個資料庫執行個體已關閉此功能。

使用零停機時間修補

執行資料 Aurora MySQL 資料庫叢集的升級牽涉到資料庫關閉和升級時發生中斷的可能性。根據預設，如果您在資料庫忙碌時開始升級，則會遺失資料庫叢集正在處理的所有連線和交易。如果您要等到資料庫閒置才執行升級，就可能必須等待很長時間。

零停機時間修補 (ZDP) 功能以最佳作法為基礎，試圖在整個 Aurora MySQL 升級過程維持用戶端正常連線。如果 ZDP 成功完成，即可保留應用程式工作階段，資料庫引擎也會在升級期間重新啟動。資料庫引擎重新啟動可能會導致輸送量下降，持續時間為數秒到一分鐘左右。

ZDP 不適用於下列項目：

- 作業系統 (OS) 修補程式與升級
- 主要版本升級

ZDP 適用於所有支援的 Aurora MySQL 版本和資料庫執行個體類別。

Aurora Serverless v1 或 Aurora 全球資料庫不支援 ZDP。

> **Note**
>
> 建議您在開發、測試伺服器或其他非生產伺服器時，僅使用 T 資料庫執行個體類別。如需詳細了解 T 執行個體類別，請參閱 使用 T 執行個體類別進行開發和測試。

您可以在 MySQL 錯誤日誌中查看 ZDP 期間重要屬性的指標。您也可以在 AWS 管理主控台 中的 Events (事件) 頁面上，查看 Aurora MySQL 何時使用 ZDP 或選擇不使用 ZDP 的相關資訊。

在 Aurora MySQL 中，不論是否啟用二進位日誌複寫功能，Aurora 皆可執行零停機時間修補。若啟用二進位日誌複寫功能，Aurora MySQL 會在 ZDP 操作期間自動中斷與 binlog 目標的連線。Aurora MySQL 會在重新啟動完成後，自動重新連線到 binlog 目標並繼續複寫。

ZDP 也可與 Aurora MySQL 中的重新開機增強功能搭配使用。修補寫入器資料庫執行個體會同時自動修補讀取器。執行修補程式之後，Aurora 會恢復寫入器和讀取器資料庫執行個體上的連線。

若有以下情況，ZDP 便可能無法成功完成：

- 長時間執行的查詢或交易正在進行中 如果 Aurora 可以在此情況下執行 ZDP，則會取消任何開啟的交易，但其連線會保留。
- 暫存資料表、使用者鎖定或資料表鎖定正在使用中，例如資料定義語言 (DDL) 陳述式執行時。Aurora 會捨棄這些連線。
- 存在擱置中的參數變更。

若因為一或多個上述條件而無執行 ZDP 的合適時段，修補作業會還原至標準行為。

雖然在成功的 ZDP 操作之後連線仍保持不變，但某些變數和功能會重新初始化。下列類型的資訊在零停機修補所造成的重新啟動時不會保留：

- 全域變數。Aurora 會恢復工作階段變數，但它不會在重新啟動後恢復全域變數。
- 狀態變數。特別是，引擎狀態報告的正常執行時間值會在使用 ZDR 或 ZDP 機制的重新啟動之後重設。
- `LAST_INSERT_ID`.
- 資料表的記憶體內 `auto_increment` 狀態。記憶體內的自動增量狀態會重新初始化。如需自動增量值的詳細資訊，請參閱 MySQL 參考手冊。
- 來自 `INFORMATION_SCHEMA` 和 `PERFORMANCE_SCHEMA` 資料表的診斷資訊。這項診斷資訊也會出現在 `SHOW PROFILE` 和 `SHOW PROFILES` 等命令的輸出中。

Events (事件) 頁面會報告下列與零停機時間重新啟動相關的活動：

- 嘗試在零停機時間的情況下升級資料庫。
- 嘗試在零停機時間的情況下升級資料庫已完成。事件會報告程序所花費的時間。此事件也會報告重新啟動期間保留的連線數目，以及已中斷的連線數目。您可以查閱資料庫錯誤日誌，瞭解重新啟動期間所發生情況的相關詳細資訊。

替代藍/綠升級技術

在某些情況下，您的首要目標是立即從舊叢集切換至升級的叢集。在這類情況下，您也可以使用多步驟程序，並排執行新舊叢集。在這裡，您會將舊叢集的資料複寫到新叢集，直至您準備好接管新叢集。如需詳細資訊，請參閱 使用 Amazon Aurora 藍/綠部署進行資料庫更新。

升級 Amazon Aurora MySQL 資料庫叢集的主要版本

在 Aurora MySQL 版本編號 (例如 3.04.1) 中，3 表示主要版本。Aurora MySQL 第 2 版與 MySQL 5.7 相容。Aurora MySQL 第 3 版與 MySQL 8.0 相容。

在主要版本之間進行升級，需要比次要版本更廣泛的規劃和測試。該程序可能需要大量時間。升級完成後，您還可能需要進行後續工作。例如，發生這種情況可能是由於 SQL 相容性的差異或某些 MySQL 相關功能的運作方式所致。或者，可能是因為舊版本和新版本之間的參數設定不同而發生。

內容

- 從 Aurora MySQL 第 2 版升級到第 3 版
- Aurora MySQL 主要版本升級路徑
- Aurora MySQL 主要版本就地升級的運作方式
- 規劃 Aurora MySQL 叢集的主要版本升級
  - 透過複製資料庫叢集來模擬升級
  - 藍/綠部署
- Aurora MySQL 的主要版本升級預先檢查
  - Aurora MySQL 的預先檢查程序
  - Aurora MySQL 的預先檢查日誌格式
  - Aurora MySQL 的預先檢查日誌輸出範例
  - Aurora MySQL 的預先檢查效能
  - Community MySQL 升級預先檢查摘要
  - Aurora MySQL 升級預先檢查摘要
  - Aurora MySQL 的預先檢查描述參考
    - 錯誤
      - 報告錯誤的 MySQL 預先檢查
      - Aurora MySQL 會預先檢查報告錯誤
    - 警告

        - 報告警告的 MySQL 預先檢查
        - 報告警告的 Aurora MySQL 預先檢查
    - 注意
    - 錯誤、警告或通知
- 就地升級執行方式
    - 就地升級如何影響叢集的參數群組
    - 在 Aurora MySQL 版本之間對叢集屬性的變更
    - 全域資料庫的就地主要升級
    - 使用跨區域僅供讀取複本進行資料庫叢集就地升級
- Aurora MySQL 會就地升級教學
- 尋找 Aurora MySQL 主要版本升級失敗的原因
- Aurora MySQL 就地升級疑難排解
- Aurora MySQL 第 3 版的升級後清除
    - 空間索引

從 Aurora MySQL 第 2 版升級到第 3 版

如果您有與 MySQL 576 相容的叢集，並且希望將其升級至與 MySQL 8.0 相容的叢集，可以在叢集本身執行升級程序來實現此操作。這種升級為就地升級，與之形成對照的是，建立新叢集時所做的升級。這項技術會保留相同的端點和叢集的其他特性。升級速度相對較快，因為不需要將所有資料複製到新的叢集磁碟區。此穩定性有助於將應用程式中的任何組態變更降至最低。此外，還有助於減少已升級叢集的測試量。這是因為資料庫執行個體及其執行個體類別的數目都保持不變。

就地升級機制在操作時涉及關閉您的資料庫叢集。Aurora 會執行正常關閉並完成未完成的操作，例如交易回復和還原清除。如需詳細資訊，請參閱Aurora MySQL 主要版本就地升級的運作方式。

就地升級方法非常便捷，因為執行相關應用程式的組態變更非常簡單，並且可將組態變更降至最低。例如，就地升級會保留叢集的端點和資料庫執行個體集。不過，就地升級所需的時間可能會因結構描述的屬性，以及叢集的忙碌程度而有所差異。因此，根據叢集的需求，您可以在各種升級技術之間進行選擇。

- 就地升級
- 藍/綠部署
- 快照還原

> **Note**
>
> 如果您使用 AWS CLI 或 RDS API 進行快照還原升級方法，則必須執行後續操作，以在還原的資料庫叢集中建立寫入器資料庫執行個體。

如需 Aurora MySQL 第 3 版及其新功能的一般資訊，請參閱 Aurora MySQL 第 3 版與 MySQL 8.0 相容。

如需有關規劃升級的詳細資訊，請參閱 規劃 Aurora MySQL 叢集的主要版本升級 和 就地升級執行方式。

Aurora MySQL 主要版本升級路徑

並非所有種類或版本的 Aurora MySQL 叢集都可以使用就地升級機制。您可以參閱下表，了解每個 Aurora MySQL 叢集的適當升級路徑。

Aurora MySQL 資料庫叢集的類型	是否可以採用就地升級？	Action
Aurora MySQL 佈建叢集，第 2 版	是	與 5.7 相容的 Aurora MySQL 叢集支援就地升級。 如需升級至 Aurora MySQL 第 3 版的相關資訊，請參閱規劃 Aurora MySQL 叢集的主要版本升級和就地升級執行方式。
Aurora MySQL 佈建叢集，第 3 版	不適用	使用次要版本升級程序，在 Aurora MySQL 第 3 版之間進行升級。
Aurora Serverless v1 叢集	不適用	僅在第 2 版上，Aurora MySQL 才會支援 Aurora Serverless v1。
Aurora Serverless v2 叢集	不適用	僅在第 3 版上，Aurora MySQL 才會支援 Aurora Serverless v2。
Aurora 全域資料庫中的叢集	是	若要將 Aurora MySQL 第 2 版升級至第 3 版，請遵循 Aurora 全域資料庫中叢集的進行就地升級的程序。在全域

Aurora MySQL 資料庫叢集的類型	是否可以採用就地升級？	Action
		叢集上執行升級。Aurora 會同時升級全域資料庫中的主要叢集和所有次要叢集。 如果您使用 AWS CLI 或 RDS API，請呼叫 `modify-global-cluster` 命令或 `ModifyGlobalCluster` 操作，而不是 `modify-db-cluster` 或 `ModifyDBCluster` 。 只有在 `lower_case_table_names` 參數設為預設值且重新啟動全球資料庫時，您才能從 Aurora MySQL 第 2 版就地升級至第 3 版。如需詳細資訊，請參閱主要版本升級。
平行查詢叢集	是	您可以執行就地升級。
二進位日誌複寫目標的叢集	也許可以	如果二進位日誌複寫來自 Aurora MySQL 叢集，您可以執行就地升級。如果二進位日誌複寫來自 RDS MySQL 或內部部署 MySQL 資料庫執行個體，則無法執行升級。在這種情況下，您可以使用快照還原機制進行升級。
具有零資料庫執行個體的叢集	否	您可以使用 AWS CLI 或 RDS API，在沒有連接任何資料庫執行個體的情況下建立 Aurora MySQL 叢集。同樣，您也可以從 Aurora MySQL 叢集移除所有資料庫執行個體，同時保持叢集磁碟區中的資料完好。雖然叢集具有零資料庫執行個體，但您無法執行就地升級。 升級機制需要叢集中的寫入器執行個體，才能對系統資料表、資料檔案等執行轉換。在此情況下，請使用 AWS CLI 或 RDS API 為叢集建立寫入器執行個體。然後可以執行就地升級。
啟用恢復的叢集	是	您可以針對使用恢復功能的 Aurora MySQL 叢集，執行就地升級。不過升級之後，您無法將叢集恢復到升級前的時間。

Aurora MySQL 主要版本就地升級的運作方式

Aurora MySQL 會將主要版本升級作為多階段程序執行。您可以檢查升級的目前狀態。某些升級步驟也會提供進度資訊。每個階段開始時，Aurora MySQL 會記錄事件。您可以在事件發生時，在 RDS 主控台的 Events (事件) 頁面上檢查事件。如需使用事件的資訊，請參閱使用 Amazon RDS 事件通知。

> **Important**
>
> 程序開始後即會執行，直至升級成功或失敗為止。升級進行中時，無法取消升級。如果升級失敗，Aurora 會復原所有變更，且您的叢集具有相同的引擎版本、中繼資料等，與之前一樣。

升級程序包含三個階段：

1. Aurora 會在開始升級程序之前執行一系列預先檢查。Aurora 在執行這些檢查時，叢集會持續執行。例如，叢集不能有任何 XA 交易處於準備狀態，或者正在處理任何資料定義語言 (DDL) 陳述式。例如，您可能需要關閉提交特定類型的 SQL 陳述式的應用程式。或者，您可以直接等到某些長時間執行的陳述式完成。然後再次嘗試升級。某些檢查會測試無法阻止升級，但可能會使升級需要很長時間的條件。

   如果 Aurora 偵測到不符合任何必要條件，請修改事件詳細資訊中識別的條件。請遵循 Aurora MySQL 就地升級疑難排解 中的指引。如果 Aurora 偵測到可能導致升級緩慢的條件，請規劃長期監控升級。
2. Aurora 會使叢集離線。然後 Aurora 會執行類似的一組測試，如上一階段，以確認關閉程序中沒有出現任何新問題。如果此時 Aurora 偵測到任何可能阻止升級的條件，Aurora 會取消升級並使叢集恢復線上狀態。在這種情況下，請確認條件何時不再適用，然後再次開始升級。
3. Aurora 會建立叢集磁碟區的快照。假設您在升級完成後，發現相容性或其他類型的問題。或者，假設您想要同時使用原始叢集和升級的叢集來執行測試。在這種情況下，您可以從此快照還原，以建立具有原始引擎版本和原始資料的新叢集。

   > **Tip**
   >
   > 此快照為手動快照。不過，即使您已達到手動快照的配額，Aurora 也可以建立快照並繼續進行升級程序。此快照會永久保留 (如有需要)，直至您將其刪除。完成所有升級後測試之後，您可以刪除此快照，將儲存費用降至最低。

4. Aurora 會複製您的叢集磁碟區。複製是一個快速操作，不涉及複製實際的資料表資料。如果在升級期間 Aurora 遇到問題，它會從複製的叢集磁碟區還原為原始資料，並使叢集恢復線上狀態。升級期間暫時複製的磁碟區不受單一叢集磁碟區複製數目的一般限制。
5. Aurora 會執行寫入器資料庫執行個體的正常關閉。在正常關閉期間，進行下列操作的進度事件每 15 分鐘記錄一次。您可以在事件發生時，在 RDS 主控台的 Events (事件) 頁面上檢查事件。
   - Aurora 會清除舊版本資料列的還原記錄。
   - Aurora 會復原任何未遞交的交易。
6. Aurora 會升級寫入器資料庫執行個體上的引擎版本：
   - Aurora 會在寫入器資料庫執行個體上安裝新引擎版本的二進位檔。
   - Aurora 會使用寫入器資料庫執行個體，將您的資料升級為 MySQL 5.7 相容的格式。在此階段，Aurora 會修改系統資料表，並執行其他會影響叢集磁碟區中資料的轉換。特別是，Aurora 會將系統資料表中的分割區中繼資料，升級為與 MySQL 5.7 分割區格式相容。如果叢集中的資料表具有大量的分割區，此階段可能需要很長時間。

     如果在此階段發生任何錯誤，您可以在 MySQL 錯誤日誌中找到詳細資訊。在此階段開始之後，如果升級程序因任何原因失敗，Aurora 會從複製的叢集磁碟區還原該原始資料。
7. Aurora 會升級讀取器資料庫執行個體上的引擎版本。
8. 升級程序已完成。Aurora 會記錄最終事件，以指示升級程序已順利完成。現在，您的資料庫叢集正在執行新的主要版本。

規劃 Aurora MySQL 叢集的主要版本升級

為了協助您決定正確的升級時間和方法，您可以了解 Aurora MySQL 第 3 版與目前 Aurora 和 MySQL 環境之間的差異：

- 如果您是從 RDS for MySQL 8.0 或 MySQL 8.0 社群版轉換，請參閱 比較 Aurora MySQL 第 3 版與 MySQL 8.0 社群版。
- 如果您是從 Aurora MySQL 第 2 版、RDS for MySQL 5.7 或社群 MySQL 5.7 升級，請參閱 比較 Aurora MySQL 第 2 版和 Aurora MySQL 第 3 版。
- 為任何自訂參數群組建立新的 MySQL 8.0 相容版本。將任何必要的自訂參數值套用至新的參數群組。請參閱 Aurora MySQL 第 3 版的參數變更以了解參數變更。
- 在升級之前，請檢閱您的 Aurora MySQL 第 2 版資料庫結構描述和物件定義，取得 MySQL 8.0 Community Edition 中引入的新保留關鍵字的使用方式。在升級之前執行此操作。如需詳細資訊，請參閱 MySQL 文件中的 MySQL 8.0 New Keywords and Reserved Words (MySQL 8.0 新關鍵字與保留字)。

您也可以在《MySQL 參考手冊》中的 MySQL 8.0 中的變更，尋找其他 MySQL 特定升級考量和秘訣。例如，您可以使用命令 `mysqlcheck --check-upgrade` 來分析您現有的 Aurora MySQL 資料庫，並找出潛在的升級問題。

> **Note**
>
> 使用就地升級或快照還原技術升級至 Aurora MySQL 第 3 版時，建議使用較大的資料庫執行個體類別。範例為 db.r5.24xlarge 和 db.r6g.16xlarge。這有助於使用資料庫執行個體上的大多數可用 CPU 容量，加快升級程序的完成速度。您可以在主要版本升級完成後變更為所需的資料庫執行個體類別。

完成升級本身後，您可以按照 Aurora MySQL 第 3 版的升級後清除 中的升級後程序來操作。最後，測試應用程式的功能和效能。

如果您是從 RDS for MySQL 或社群 MySQL 轉換，請遵循將資料遷移至 Amazon Aurora MySQL 資料庫叢集中說明的遷移程序。在某些情況下，您可能會使用二進位日誌複寫，將資料與 Aurora MySQL 第 3 版叢集同步，做為遷移的一部分。若是如此，來源系統必須執行與您的目標資料庫叢集相容的版本。

若要確保在主要版本之間升級叢集之後，您的應用程式和管理程序可以順利運作，請進行一些預先規劃和準備。若要查看要更新 AWS CLI 指令碼或 RDS API 型應用程式的管理程式碼，請參閱 就地升級如何影響叢集的參數群組。另請參閱在 Aurora MySQL 版本之間對叢集屬性的變更。

若要了解升級期間可能會遇到的問題，請參閱 Aurora MySQL 就地升級疑難排解。對於可能導致升級需要很長時間的問題，您可以預先測試這些條件並進行更正。

> **Note**
>
> 就地升級在操作時涉及關閉您的資料庫叢集。Aurora MySQL 會執行正常關閉並完成未完成的操作，例如還原清除。如果有許多復原記錄需要清除，則升級可能需要很長的時間。我們建議只在歷史記錄清單長度 (HLL) 很低之後才執行升級。HLL 的一般可接受值為 100,000 或更少。如需詳細資訊，請參閱這篇部落格文章。

透過複製資料庫叢集來模擬升級

您可以針對升級的叢集檢查應用程式相容性、效能、維護程序和類似考量。若要這樣做，您可以在進行真正的升級之前執行升級的模擬。此技術對於生產叢集特別有用。在此，務必將停機時間降至最低，並在升級完成後立即準備好升級的叢集。

使用下列步驟：

1. 建立原始叢集的複製。請遵循 複製 Amazon Aurora 資料庫叢集的一個磁碟區 中的程序。
2. 設定與原始叢集相似的一組寫入器和讀取器資料庫執行個體。
3. 執行複製叢集的就地升級。請遵循 就地升級執行方式 中的程序。

   建立複製後立即開始升級。如此一來，叢集磁碟區仍與原始叢集的狀態相同。如果複製在進行升級之前閒置，Aurora 會在後台執行資料庫清除程序。在這種情況下，複製的升級並非正確的升級原始叢集模擬。
4. 使用複製的叢集測試應用程式相容性、效能、管理程序等。
5. 如果您遇到任何問題，請調整升級計劃以解決這些問題。例如，調整任何應用程式的程式碼，以便與更高版本的功能集相容。根據叢集中的資料量，估計升級可能需要多長時間。您還可以選擇在叢集不忙碌的時間排程升級。
6. 在您對應用程式和工作負載適當地使用測試叢集感到滿意之後，即可執行生產叢集的就地升級。
7. 在主要版本升級期間，努力將叢集的總停機時間降到最低。若要這樣做，請確定叢集上的工作負載在升級時很低或為零。特別是，請確定在您啟動升級時，沒有長時間執行的交易正在進行。

藍/綠部署

在某些情況下，您的首要目標是立即從舊叢集切換至升級的叢集。在這類情況下，您也可以使用多步驟程序，並排執行新舊叢集。在這裡，您會將舊叢集的資料複寫到新叢集，直至您準備好接管新叢集。如需詳細資訊，請參閱 使用 Amazon Aurora 藍/綠部署進行資料庫更新。

Aurora MySQL 的主要版本升級預先檢查

從一個主要版本升級 MySQL 到另一個主要版本，例如從 MySQL 5.7 到 MySQL 8.0，涉及一些需要仔細規劃和準備的重大架構變更。與主要著重於更新資料庫引擎軟體的次要版本升級，以及在某些情況下的系統資料表不同，主要 MySQL 升級通常會對資料庫存放和管理其中繼資料的方式帶來重大變更。

為了協助您識別這種不相容，從 Aurora MySQL 第 2 版升級至第 3 版時，Aurora 會自動執行升級相容性檢查 (預先檢查)，以檢查資料庫叢集中的物件，並識別已知可能會防止升級繼續進行的不相容。如需 Aurora MySQL 預先檢查的詳細資訊，請參閱 Aurora MySQL 的預先檢查描述參考。Aurora 預先檢查會在 Community MySQL 升級檢查程式公用程式執行的檢查以外執行。

系統會強制執行這些前置檢查，您無法選擇略過這些檢查。前置檢查提供以下優勢：

- 它們可以降低發生升級失敗的可能性，這可能會導致延長停機時間。
- 出現不相容情況時，Amazon Aurora 即會防止系統繼續升級，並提供相關日誌讓您了解。如此，您就可以使用這些日誌來減少不相容情況，為資料庫做好升級至第 3 版 的準備。如需解決不相容問題的詳細資訊，請參閱 MySQL 文件中的準備您的安裝進行升級，以及 MySQL 部落格上的升級至 MySQL 8.0？ 這裡是您需要知道的事項...。

  如需升級至 MySQL 8.0 的詳細資訊，請參閱 MySQL 文件中的 Upgrading MySQL (升級 MySQL)。

預先檢查會在資料庫叢集離線以進行主要版本升級之前執行。如果預先檢查找到不相容，Aurora 會在資料庫執行個體停止前自動取消升級。Aurora 也會為不相容產生事件。如需 Amazon Aurora 事件的詳細資訊，請參閱 使用 Amazon RDS 事件通知。

完成預先檢查後，Aurora 會記錄 `upgrade-prechecks.log` 檔案中每個不相容的詳細資訊。在多數情況下，日誌項目包含修正不相容的 MySQL 文件連結。如需檢視日誌檔案的詳細資訊，請參閱檢視並列出資料庫日誌檔案。

> **Note**
>
> 根據前置檢查的特性，這些檢查作業會分析資料庫中的物件。此分析會耗用資源，並增加升級完成的時間。如需預先檢查效能考量的詳細資訊，請參閱 Aurora MySQL 的預先檢查程序。

內容

- Aurora MySQL 的預先檢查程序
- Aurora MySQL 的預先檢查日誌格式

- Aurora MySQL 的預先檢查日誌輸出範例
- Aurora MySQL 的預先檢查效能
- Community MySQL 升級預先檢查摘要
- Aurora MySQL 升級預先檢查摘要
- Aurora MySQL 的預先檢查描述參考
  - 錯誤
    - 報告錯誤的 MySQL 預先檢查
    - Aurora MySQL 會預先檢查報告錯誤
  - 警告
    - 報告警告的 MySQL 預先檢查
    - 報告警告的 Aurora MySQL 預先檢查
  - 注意
  - 錯誤、警告或通知

Aurora MySQL 的預先檢查程序

如前所述，Aurora MySQL 升級程序涉及在資料庫上執行相容性檢查 (預先檢查)，然後才能繼續主要版本升級。

對於就地升級，預先檢查會在寫入器資料庫執行個體上線時對其執行。如果預先檢查成功，升級會繼續進行。如果發現錯誤，則會記錄在 `upgrade-prechecks.log` 檔案中，並取消升級。再次嘗試升級之前，請先解決 `upgrade-prechecks.log` 檔案中傳回的任何錯誤。

對於快照還原升級，預先檢查會在還原程序期間執行。如果成功，您的資料庫將升級至新的 Aurora MySQL 版本。如果發現錯誤，則會記錄在 `upgrade-prechecks.log` 檔案中，並取消升級。再次嘗試升級之前，請先解決 `upgrade-prechecks.log` 檔案中傳回的任何錯誤。

如需詳細資訊，請參閱尋找 Aurora MySQL 主要版本升級失敗的原因及Aurora MySQL 的預先檢查描述參考。

若要監控預先檢查狀態，您可以在資料庫叢集上檢視下列事件。

預先檢查狀態	事件訊息	Action
已開始	升級準備進行中：開始線上升級預先檢查。	無

預先檢查狀態	事件訊息	Action
失敗	資料庫叢集處於無法升級的狀態：升級預先檢查失敗。如需詳細資訊，請參閱 upgrade-prechecks.log 檔案。 如需故障診斷升級失敗原因的詳細資訊，請參閱 https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/AuroraMySQL.Upgrading.Troubleshooting.html	檢閱錯誤的 `upgrade-prechecks.log` 。 修正錯誤。 重試升級。
Succeeded	升級準備進行中：已完成線上升級預先檢查。	預先檢查成功，未傳回錯誤。 檢閱警告和通知的 `upgrade-prechecks.log` 。

如需檢視事件的詳細資訊，請參閱 檢視 Amazon RDS 事件。

Aurora MySQL 的預先檢查日誌格式

升級相容性檢查 (預先檢查) 完成後，您可以檢閱 `upgrade-prechecks.log` 檔案。日誌檔案包含每個預先檢查的結果、受影響的物件和修正資訊。

錯誤會封鎖升級。您必須先解決這些錯誤，才能重試升級。

警告和通知較不重要，但仍建議您仔細檢閱，以確保應用程式工作負載沒有相容性問題。盡快解決任何已識別的問題。

日誌檔案的格式如下：

- `targetVersion` – Aurora MySQL 升級的 MySQL 相容版本。
- `auroraServerVersion` – 執行預先檢查的 Aurora MySQL 版本。
- `auroraTargetVersion` – 您要升級的 Aurora MySQL 版本。
- `checksPerformed` – 包含已執行的預先檢查清單。

- id – 正在執行的預先檢查名稱。
- title – 正在執行的預先檢查描述。
- status – 這不表示預先檢查成功或失敗，但會顯示預先檢查查詢的狀態：
  - OK – 預先檢查查詢已成功執行並完成。
  - ERROR – 預先檢查查詢無法執行。這可能是因為資源限制、意外執行個體重新啟動或相容性預先檢查查詢中斷等問題而發生。

    如需詳細資訊，請參閱此範例。
- description – 不相容的一般描述，以及如何修正問題。
- documentationLink – 如適用，此處會記下相關 Aurora MySQL 或 MySQL 文件的連結。如需詳細資訊，請參閱Aurora MySQL 的預先檢查描述參考。
- detectedProblems – 如果預先檢查傳回錯誤、警告或通知，這會顯示不相容的詳細資訊，以及適用的不相容物件：
  - level – 預先檢查偵測到的不相容層級。有效層級如下：
    - Error – 您必須先解決不相容的問題，才能繼續升級。
    - Warning – 升級可以繼續，但偵測到已棄用的物件、語法或組態。請仔細檢閱警告，並盡快解決，以避免未來版本發生問題。
    - Notice – 升級可以繼續，但偵測到已棄用的物件、語法或組態。請仔細檢閱通知，並盡快解決，以避免未來版本發生問題。
  - dbObject – 偵測到不相容的資料庫物件名稱。
  - description – 不相容的詳細描述，以及如何修正問題。
- errorCount – 偵測到的不相容錯誤數目。這些會封鎖升級。
- warningCount – 偵測到的不相容警告數量。這些不會封鎖升級，但請盡快解決，以避免未來版本發生問題。
- noticeCount – 偵測到的不相容通知數量。這些不會封鎖升級，請盡快解決，以避免未來版本發生問題。
- Summary – 預先檢查相容性錯誤、警告和通知計數的摘要。

Aurora MySQL 的預先檢查日誌輸出範例

下列範例顯示您可能會看到的預先檢查日誌輸出。如需執行中預先檢查的詳細資訊，請參閱 Aurora MySQL 的預先檢查描述參考。

預先檢查狀態正常，未偵測到不相容

預先檢查查詢已成功完成。未偵測到不相容。

```
{
  "id": "auroraUpgradeCheckIndexLengthLimitOnTinytext",
  "title": "Check for the tables with indexes defined with prefix length greater
 than 255 bytes on tiny text columns",
  "status": "OK",
  "detectedProblems": []
},
```

預先檢查狀態正常，偵測到錯誤

預先檢查查詢已成功完成。偵測到一個錯誤。

```
{
  "id": "auroraUpgradeCheckForPrefixIndexOnGeometryColumns",
  "title": "Check for geometry columns on prefix indexes",
  "status": "OK",
  "description": "Consider dropping the prefix indexes of geometry columns and
 restart the upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test25.sbtest1",
        "description": "Table `test25`.`sbtest1` has an index `idx_t1` on geometry
 column/s. Mysql 8.0 does not support this type of index on a geometry column
 https://dev.mysql.com/worklog/task/?id=11808. To upgrade to MySQL 8.0, Run 'DROP
 INDEX `idx_t1` ON `test25`.`sbtest1`;"
      },
 }
```

預先檢查狀態正常，偵測到警告

當預先檢查成功或失敗時，可能會傳回警告。

在這裡，預先檢查查詢已成功完成。偵測到兩個警告。

```
{
  "id": "zeroDatesCheck",
  "title": "Zero Date, Datetime, and Timestamp values",
  "status": "OK",
```

```
  "description": "Warning: By default zero date/datetime/timestamp values are no
 longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are included
 in SQL_MODE by default. These modes should be used with strict mode as they will be
 merged with strict mode in a future release. If you do not include these modes in
 your SQL_MODE setting, you are able to insert date/datetime/timestamp values that
 contain zeros. It is strongly advised to replace zero values with valid ones, as
 they may not work correctly in the future.",
  "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "global.sql_mode",
        "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
 which allows insertion of zero dates"
      },
      {
        "level": "Warning",
        "dbObject": "session.sql_mode",
        "description": " of 10 session(s) does not contain either NO_ZERO_DATE or
 NO_ZERO_IN_DATE which allows insertion of zero dates"
      }
    ]
}
```

預先檢查狀態錯誤，未回報不相容

預先檢查查詢失敗並發生錯誤，因此無法確認不相容。

```
{
  "id": "auroraUpgradeCheckForDatafilePathInconsistency",
  "title": "Check for inconsistency related to ibd file path.",
  "status": "ERROR",
  "description": "Can't connect to MySQL server on 'localhost:3306' (111) at
 13/08/2024 12:22:20 UTC. This failure can occur due to low memory available on the
 instance for executing upgrade prechecks. Please check 'FreeableMemory' Cloudwatch
 metric to verify the available memory on the instance while executing prechecks. If
 instance ran out of memory, we recommend to retry the upgrade on a higher instance
 class."
}
```

此失敗可能是因為未預期的執行個體重新啟動，或執行時資料庫上的相容性預先檢查查詢中斷。例如，在較小的資料庫執行個體類別上，當執行個體上的可用記憶體不足時，您可能會遇到這種情況。

您可以使用 FreeableMemory Amazon CloudWatch 指標，在執行預先檢查時確認執行個體上的可用記憶體。如果執行個體記憶體不足，建議您在較大的資料庫執行個體類別上重試升級。在某些情況下，您可以使用藍/綠部署 這可讓預先檢查和升級在與生產工作負載無關的「綠」資料庫叢集上執行，這也會使用系統資源。

如需詳細資訊，請參閱針對 Aurora MySQL 資料庫的記憶體用量問題進行故障診斷。

預先檢查摘要，偵測到一個錯誤和三個警告

相容性預先檢查也包含來源和目標 Aurora MySQL 版本的資訊，以及預先檢查輸出結束時的錯誤、警告和通知計數摘要。

例如，下列輸出顯示嘗試從 Aurora MySQL 2.11.6 升級到 Aurora MySQL 3.07.1。升級傳回一個錯誤、三個警告，而且沒有通知。由於在傳回錯誤時無法繼續升級，您必須解決 routineSyntaxCheck 相容性問題，然後重試升級。

```
{
  "serverAddress": "/tmp%2Fmysql.sock",
  "serverVersion": "5.7.12 - MySQL Community Server (GPL)",
  "targetVersion": "8.0.36",
  "auroraServerVersion": "2.11.6",
  "auroraTargetVersion": "3.07.1",
  "outfilePath": "/rdsdbdata/tmp/PreChecker.log",
  "checksPerformed": [{
      ... output for each individual precheck ...
      .
      .
      {
        "id": "oldTemporalCheck",
        "title": "Usage of old temporal type",
        "status": "OK",
          "detectedProblems": []
      },
      {
        "id": "routinesSyntaxCheck",
        "title": "MySQL 8.0 syntax check for routine-like objects",
        "status": "OK",
        "description": "The following objects did not pass a syntax check with
 the latest MySQL 8.0 grammar. A common reason is that they reference names that
 conflict with new reserved keywords. You must update these routine definitions and
 `quote` any such references before upgrading.",
        "documentationLink": "https://dev.mysql.com/doc/refman/en/keywords.html",
        "detectedProblems": [{
```

```
            "level": "Error",
            "dbObject": "test.select_res_word",
            "description": "at line 2,18: unexpected token 'except'"
        }]
      },
      .
      .
      .
      {
        "id": "zeroDatesCheck",
        "title": "Zero Date, Datetime, and Timestamp values",
        "status": "OK",
        "description": "Warning: By default zero date/datetime/timestamp values
 are no longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are
 included in SQL_MODE by default. These modes should be used with strict mode as
 they will be merged with strict mode in a future release. If you do not include
 these modes in your SQL_MODE setting, you are able to insert date/datetime/
timestamp values that contain zeros. It is strongly advised to replace zero values
 with valid ones, as they may not work correctly in the future.",
        "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
        "detectedProblems": [{
            "level": "Warning",
            "dbObject": "global.sql_mode",
            "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
 which allows insertion of zero dates"
            },
            {
            "level": "Warning",
            "dbObject": "session.sql_mode",
            "description": " of 8 session(s) does not contain either NO_ZERO_DATE or
 NO_ZERO_IN_DATE which allows insertion of zero dates"
            }
          ]
       },
       .
       .
       .
  }],
  "errorCount": 1,
  "warningCount": 3,
  "noticeCount": 0,
  "Summary": "1 errors were found. Please correct these issues before upgrading to
 avoid compatibility issues."
```

```
}
```

Aurora MySQL 的預先檢查效能

相容性預先檢查會在資料庫執行個體離線以進行升級之前執行，因此在一般情況下，不會在執行時導致資料庫執行個體停機時間。不過，它們可能會影響寫入器資料庫執行個體上執行的應用程式工作負載。預先檢查會透過 information_schema 資料表存取資料字典，如果有許多資料庫物件，可能會很慢。請考慮下列因素：

- 預先檢查持續時間會因資料表、資料欄、常式和限制條件等資料庫物件數量而有所不同。具有大量物件的資料庫叢集可能需要更長的時間來執行。

  例如，removedFunctionsCheck 可能需要更長的時間，並根據儲存的物件數量使用更多資源。
- 對於就地升級，使用較大的資料庫執行個體類別 (例如 db.r5.24xlarge 或 db.r6g.16xlarge) 可以透過使用更多 CPU 協助更快完成升級。您可以在升級後縮減大小。
- 跨多個資料庫對 `information_schema` 的查詢可能會很慢，尤其是在許多物件和較小的資料庫執行個體上。在這種情況下，請考慮使用複製、快照還原或藍/綠部署進行升級。
- 預先檢查資源用量 (CPU、記憶體) 可能會隨著更多物件而增加，導致較小資料庫執行個體上的執行時間更長。在這種情況下，請考慮使用複製、快照還原或藍/綠部署進行測試以進行升級。

  如果預先檢查因資源不足而失敗，您可以使用狀態輸出在預先檢查日誌中偵測到此問題：

```
"status": "ERROR",
```

如需詳細資訊，請參閱Aurora MySQL 主要版本就地升級的運作方式及規劃 Aurora MySQL 叢集的主要版本升級。

Community MySQL 升級預先檢查摘要

以下是 MySQL 5.7 和 8.0 之間不相容的一般清單：

- 您的 MySQL 5.7 相容資料庫叢集不得使用 MySQL 8.0 中不支援的功能。

  如需詳細資訊，請參閱 MySQL 文件中的 Features Removed in MySQL 8.0 (MySQL 8.0 中移除的功能)。
- 不能違反關鍵字或保留字的規定。MySQL 8.0 中可能會保留一些先前未保留的關鍵字。

  如需詳細資訊，請參閱 MySQL 文件中的 Keywords and Reserved Words (關鍵字與保留字)。

- 運用 Unicode 增強支援時，請考慮將使用 utf8mb3 字元集的物件轉換成使用 utf8mb4 字元集，因為 utf8mb3 字元集已棄用。另外，utf8mb4 目前是 utf8 字元集的別名，因此請考慮使用 utf8 做為字元集參考，而不是 utf8mb3。

  如需詳細資訊，請參閱 MySQL 文件中的 The utf8mb3 character set (3-byte UTF-8 unicode encoding)。
- 不得有非預設資料列格式的 InnoDB 資料表。
- 不得有 ZEROFILL 或 display 長度類型屬性。
- 分割資料表所使用的儲存引擎皆需提供原生分割支援。
- MySQL 5.7 mysql 系統資料庫中的資料表名稱不得與 MySQL 8.0 資料字典所使用的資料表名稱相同。
- 資料表不能使用過時的資料類型或函數。
- 外部索引鍵的限制條件名稱不得超過 64 個字元。
- 不能在 sql_mode 系統變數設定中定義過時的 SQL 模式。
- 資料表或預存程序的個別 ENUM 或 SET 資料欄元素長度皆不得超過 255 個字元。
- 共用 InnoDB 資料表空間中不能存在資料表分割區。
- 資料表空間資料檔案路徑中不得有循環參考。
- 查詢和預存程式定義皆不得對 ASC 子句使用 DESC 或 GROUP BY 限定詞。
- 不得移除任何系統變數，且系統變數必須使用 MySQL 8.0 的新預設值。
- 日期、日期時間或時間戳記值不得為零 (0)。
- 不得有因檔案移除或損毀所導致的結構描述不一致
- 不得有任何包含 FTS 字元字串的資料表名稱。
- 不得有任何屬於不同引擎的 InnoDB 資料表。
- 不得有任何資料表或結構描述名稱對 MySQL 5.7 無效。

如需執行中預先檢查的詳細資訊，請參閱 Aurora MySQL 的預先檢查描述參考。

如需升級至 MySQL 8.0 的詳細資訊，請參閱 MySQL 文件中的 Upgrading MySQL (升級 MySQL)。如需 MySQL 8.0 中變更的一般說明，請參閱 MySQL 文件中的 MySQL 8.0 中的新功能。

Aurora MySQL 升級預先檢查摘要

從第 2 版升級至第 3 版時，Aurora MySQL 有自己的特定需求，包括下列項目：

- 在檢視、常式、觸發條件和事件中，不得有已棄用的 SQL 語法，例如 SQL_CACHE、SQL_NO_CACHE 和 QUERY_CACHE。
- 任何沒有 FTS 索引的資料表上不得有 FTS_DOC_ID 欄。
- InnoDB 資料字典與實際資料表定義之間不得有任何資料欄定義不相符。
- lower_case_table_names 參數設定為 1 時，所有資料庫和資料表名稱都必須是小寫。
- 事件和觸發條件不能有遺漏或空白的 DEFINER，或是無效的建立內容。
- 資料庫中的所有觸發條件名稱都必須是唯一的。
- Aurora MySQL 第 3 版不支援 DDL 復原和快速 DDL。資料庫中不得有與這些功能相關的成品。
- 具有 REDUNDANT 或 COMPACT 資料列格式的資料表索引不能大於 767 個位元組。
- 在 tiny 文字資料欄上定義的索引字首長度不能超過 255 個位元組。使用 utf8mb4 字元集時，這會將支援的字首長度限制為 63 個字元。

  使用 innodb_large_prefix 參數在 MySQL 5.7 中允許較大的字首長度。此參數已在 MySQL 8.0 中棄用。
- mysql.host 資料表中不得有任何 InnoDB 中繼資料不一致。
- 系統資料表中不得有任何資料欄資料類型不符。
- prepared 狀態中不得有 XA 交易。
- 檢視中的資料欄名稱不能超過 64 個字元。
- 預存程序中的特殊字元不能不一致。
- 資料表不能有資料檔案路徑不一致。

如需執行中預先檢查的詳細資訊，請參閱 Aurora MySQL 的預先檢查描述參考。

Aurora MySQL 的預先檢查描述參考

此處詳細說明 Aurora MySQL 的升級預先檢查。

內容

- 錯誤
  - 報告錯誤的 MySQL 預先檢查
  - Aurora MySQL 會預先檢查報告錯誤
- 警告

  - 報告警告的 MySQL 預先檢查
  - 報告警告的 Aurora MySQL 預先檢查
- 注意
- 錯誤、警告或通知

錯誤

下列預先檢查會在預先檢查失敗且升級無法繼續時產生錯誤。

主題

- 報告錯誤的 MySQL 預先檢查
- Aurora MySQL 會預先檢查報告錯誤

報告錯誤的 MySQL 預先檢查

下列預先檢查來自 Community MySQL：

- checkTableMysqlSchema
- circularDirectoryCheck
- columnsWhichCannotHaveDefaultsCheck
- depreciatedSyntaxCheck
- engineMixupCheck
- enumSetElementLengthCheck
- foreignKeyLengthCheck
- getDuplicateTriggers
- getEventsWithNullDefiner
- getMismatchedMetadata
- getTriggersWithNullDefiner
- getValueOfVariablelower_case_table_names
- groupByAscSyntaxCheck
- mysqlEmptyDotTableSyntaxCheck
- mysqlIndexTooLargeCheck

- mysqlInvalid57NamesCheck
- mysqlOrphanedRoutinesCheck
- mysqlSchemaCheck
- nonNativePartitioningCheck
- oldTemporalCheck
- partitionedTablesInSharedTablespaceCheck
- removedFunctionsCheck
- routineSyntaxCheck
- schemaInconsistencyCheck

checkTableMysqlSchema

預先檢查層級：錯誤

mysql 結構描述的 **check table x for upgrade** 命令所報告的問題

開始升級至 Aurora MySQL 第 3 版之前，check table for upgrade 會在資料庫執行個體 mysql 結構描述中的每個資料表上執行。check table for upgrade 命令會檢查資料表是否有任何在升級至較新版本的 MySQL 期間可能發生的潛在問題。在嘗試升級之前執行此命令有助於事先識別和解決任何不相容，使實際的升級程序更順暢。

此命令會對每個資料表執行各種檢查，如下所示：

- 確認資料表結構和中繼資料是否與目標 MySQL 版本相容
- 檢查資料表使用的任何已棄用或已移除的功能
- 確保資料表可以正確升級，而不會遺失資料

如需詳細資訊，請參閱 MySQL 文件中的 CHECK TABLE 陳述式。

輸出範例：

```
{
  "id": "checkTableMysqlSchema",
  "title": "Issues reported by 'check table x for upgrade' command for mysql
 schema.",
  "status": "OK",
  "detectedProblems": []
```

```
}
```

此預先檢查的輸出取決於遇到的錯誤，以及遇到的時間，因為 `check table for upgrade` 會執行多個檢查。

如果您在此預先檢查中遇到任何錯誤，請向 AWS Support 開立案例，以請求解決中繼資料不一致的問題。或者，您可以執行邏輯傾印，然後還原至新的 Aurora MySQL 第 3 版資料庫叢集，以重試升級。

circularDirectoryCheck

預先檢查層級：錯誤

資料表空間資料檔案路徑中的循環目錄參考

從 MySQL 8.0.17 開始，`CREATE TABLESPACE ... ADD DATAFILE` 子句不再允許循環目錄參考。為避免升級問題，請先從資料表空間資料檔案路徑移除任何循環目錄參考，再升級至 Aurora MySQL 第 3 版。

輸出範例：

```
{
  "id": "circularDirectory",
  "title": "Circular directory references in tablespace data file paths",
  "status": "OK",
  "description": "Error: Following tablespaces contain circular directory references
 (e.g. the reference '/../') in data file paths which as of MySQL 8.0.17 are not
 permitted by the CREATE TABLESPACE ... ADD DATAFILE clause. An exception to the
 restriction exists on Linux, where a circular directory reference is permitted if
 the preceding directory is a symbolic link. To avoid upgrade issues, remove any
 circular directory references from tablespace data file paths before upgrading.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-innodb-changes",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "ts2",
        "description": "circular reference in datafile path: '/home/ec2-user/dbdata/
mysql_5_7_44/../ts2.ibd'",
        "dbObjectType": "Tablespace"
      }
  ]
}
```

如果您收到此錯誤，請使用 file-per-table 資料表空間重建資料表。對所有資料表空間和資料表定義使用預設檔案路徑。

> **Note**
>
> Aurora MySQL 不支援一般資料表空間或 CREATE TABLESPACE 命令。
> 在重建資料表空間之前，請參閱 MySQL 文件中的線上 DDL 操作，以了解鎖定和資料移動對前景交易的影響。

重建後，預先檢查會通過，允許升級繼續進行。

```
{
  "id": "circularDirectoryCheck",
  "title": "Circular directory references in tablespace data file paths",
  "status": "OK",
  "detectedProblems": []
},
```

columnsWhichCannotHaveDefaultsCheck

預先檢查層級：錯誤

無法具有預設值的資料欄

在 MySQL 8.0.13 之前，BLOB、GEOMETRY、TEXT 和 JSON 資料欄不能有預設值。在升級至 Aurora MySQL 第 3 版之前，請移除這些資料欄上的任何預設子句。如需這些資料類型預設處理方式變更的詳細資訊，請參閱 MySQL 文件中的資料類型預設值。

輸出範例：

```
{
  "id": "columnsWhichCannotHaveDefaultsCheck",
  "title": "Columns which cannot have default values",
  "status": "OK",
  "description": "Error: The following columns are defined as either BLOB, TEXT,
 GEOMETRY or JSON and have a default value set. These data types cannot have default
 values in MySQL versions prior to 8.0.13, while starting with 8.0.13, the default
 value must be specified as an expression. In order to fix this issue, please use
 the ALTER TABLE ... ALTER COLUMN ... DROP DEFAULT statement.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/data-type-
defaults.html#data-type-defaults-explicit",
```

```
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.test_blob_default.geo_col",
        "description": "geometry"
      }
  ]
},
```

預先檢查會傳回錯誤，因為 test.test_blob_default 資料表中的 geo_col 欄使用具有指定預設值的 BLOB、GEOMETRY、TEXT 或 JSON 資料類型。

查看資料表定義，我們可以看到 geo_col 欄定義為 geo_col geometry NOT NULL default ''。

```
mysql> show create table test_blob_default\G
*************************** 1. row ***************************
       Table: test_blob_default
Create Table: CREATE TABLE `test_blob_default` (
  `geo_col` geometry NOT NULL DEFAULT ''
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

移除此預設子句以允許預先檢查通過。

> **Note**
>
> 在執行 ALTER TABLE 陳述式或重建資料表空間之前，請參閱 MySQL 文件中的線上 DDL 操作，以了解鎖定和資料移動對前景交易的影響。

```
mysql> ALTER TABLE test_blob_default modify COLUMN geo_col geometry NOT NULL;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show create table test_blob_default\G
*************************** 1. row ***************************
       Table: test_blob_default
Create Table: CREATE TABLE `test_blob_default` (
  `geo_col` geometry NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

預先檢查通過，您可以重試升級。

```
{
  "id": "columnsWhichCannotHaveDefaultsCheck",
  "title": "Columns which cannot have default values",
  "status": "OK",
  "detectedProblems": []
},
```

depreciatedSyntaxCheck

預先檢查層級：錯誤

定義中已棄用關鍵字的使用

MySQL 8.0 已移除查詢快取。因此，某些查詢快取特定的 SQL 語法已移除。如果您的任何資料庫物件包含 QUERY CACHE、SQL_CACHE 或 SQL_NO_CACHE 關鍵字，則會傳回預先檢查錯誤。若要解決此問題，請重新建立這些物件，移除提及的關鍵字。

輸出範例：

```
{
  "id": "depreciatedSyntaxCheck",
  "title": "Usage of depreciated keywords in definition",
  "status": "OK",
  "description": "Error: The following DB objects contain keywords like 'QUERY
 CACHE', 'SQL_CACHE', 'SQL_NO_CACHE' which are not supported in major version 8.0.
 It is recommended to drop these DB objects or rebuild without any of the above
 keywords before upgrade.",
  "detectedProblems": [
      {
"level": "Error",
"dbObject": "test.no_query_cache_check",
"description": "PROCEDURE uses depreciated words in definition"
      }
  ]
}
```

預先檢查會報告 test.no_query_cache_check 預存程序正在使用其中一個已移除的關鍵字。查看程序定義，我們可以看到它使用 SQL_NO_CACHE。

```
mysql> show create procedure test.no_query_cache_check\G
```

```
*************************** 1. row ***************************
           Procedure: no_query_cache_check
            sql_mode:
    Create Procedure: CREATE DEFINER=`reinvent`@`%` PROCEDURE
 `no_query_cache_check`()
BEGIN
    SELECT SQL_NO_CACHE k from sbtest1 where id > 10 and id < 20 group by k asc;
END
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

移除關鍵字。

```
mysql> drop procedure test.no_query_cache_check;
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter //

mysql> CREATE DEFINER=`reinvent`@`%` PROCEDURE `no_query_cache_check`() BEGIN
 SELECT k from sbtest1 where id > 10 and id < 20 group by k asc; END//
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
```

移除關鍵字後，預先檢查會成功完成。

```
{
  "id": "depreciatedSyntaxCheck",
  "title": "Usage of depreciated keywords in definition",
  "status": "OK",
  "detectedProblems": []
}
```

engineMixupCheck

預先檢查層級：錯誤

InnoDB 辨識屬於不同引擎的資料表

與 schemaInconsistencyCheck 類似，此預先檢查會先確認 MySQL 中的資料表中繼資料是否一致，再繼續升級。

如果您在此預先檢查中遇到任何錯誤，請向 AWS Support 開立案例，以請求解決中繼資料不一致的問題。或者，您可以執行邏輯傾印，然後還原至新的 Aurora MySQL 第 3 版資料庫叢集，以重試升級。

輸出範例：

```
{
  "id": "engineMixupCheck",
  "title": "Tables recognized by InnoDB that belong to a different engine",
  "status": "OK",
  "description": "Error: Following tables are recognized by InnoDB engine while the
 SQL layer believes they belong to a different engine. Such situation may happen
 when one removes InnoDB table files manually from the disk and creates e.g. a
 MyISAM table with the same name.\n\nA possible way to solve this situation is to
 e.g. in case of MyISAM table:\n\n1. Rename the MyISAM table to a temporary name
 (RENAME TABLE).\n2. Create some dummy InnoDB table (its definition does not need to
 match), then copy (copy, not move) and rename the dummy .frm and .ibd files to the
 orphan name using OS file commands.\n3. The orphan table can be then dropped (DROP
 TABLE), as well as the dummy table.\n4. Finally the MyISAM table can be renamed
 back to its original name.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "mysql.general_log_backup",
        "description": "recognized by the InnoDB engine but belongs to CSV"
      }
  ]
}
```

enumSetElementLengthCheck

預先檢查層級：錯誤

ENUM 和 SET 欄定義包含超過 255 個字元的元素

資料表和預存程序的 ENUM 或 SET 欄元素不得超過 255 個字元或 1020 位元組。在 MySQL 8.0 之前，合併長度上限為 64K，但 8.0 會將個別元素限制為 255 個字元或 1020 個位元組 (支援多位元組)。如果您收到 enumSetElementLengthCheck 的預先檢查失敗，請在重試升級之前修改任何超過這些新限制的元素。

輸出範例：

```
{
```

```
  "id": "enumSetElementLengthCheck",
  "title": "ENUM/SET column definitions containing elements longer than 255
 characters",
  "status": "OK",
  "description": "Error: The following columns are defined as either ENUM or SET and
 contain at least one element longer that 255 characters. They need to be altered so
 that all elements fit into the 255 characters limit.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/string-type-
overview.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.large_set.s",
        "description": "SET contains element longer than 255 characters"
      }
  ]
},
```

預先檢查會報告錯誤，因為 `test.large_set` 資料表中的 `s` 欄包含大於 255 個字元的 SET 元素。

減少此欄的 SET 大小後，預先檢查會通過，讓升級繼續進行。

```
{
  "id": "enumSetElementLenghtCheck",
  "title": "ENUM/SET column definitions containing elements longer than 255
 characters",
  "status": "OK",
  "detectedProblems": []
},
```

foreignKeyLengthCheck

預先檢查層級：錯誤

外部索引鍵的限制條件名稱超過 64 個字元。

在 MySQL 中，識別符的長度限制為 64 個字元，如 MySQL 文件所述。由於發現外部索引鍵長度可能等於或超過此值的問題，導致升級失敗，因此已實作此預先檢查。如果此預先檢查發生錯誤，您應該變更或重新命名限制條件，使其在重試升級之前少於 64 個字元。

輸出範例：

```
{
  "id": "foreignKeyLength",
  "title": "Foreign key constraint names longer than 64 characters",
  "status": "OK",
  "detectedProblems": []
}
```

getDuplicateTriggers

預先檢查層級：錯誤

資料庫中的所有觸發條件名稱必須是唯一的。

由於資料字典實作的變更，MySQL 8.0 不支援資料庫中區分大小寫的觸發條件。此預先檢查會確認您的資料庫叢集沒有一或多個包含重複觸發條件的資料庫。如需詳細資訊，請參閱 MySQL 文件中的識別符區分大小寫。

輸出範例：

```
{
  "id": "getDuplicateTriggers",
  "title": "MySQL pre-checks that all trigger names in a database are unique or
 not.",
  "status": "OK",
  "description": "Error: You have one or more database containing duplicate
 triggers. Mysql 8.0 does not support case sensitive triggers within a database
 https://dev.mysql.com/doc/refman/8.0/en/identifier-case-sensitivity.html. To
 upgrade to MySQL 8.0, drop the triggers with case-insensitive duplicate names and
 recreate with distinct names.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test",
        "description": "before_insert_product"
      },
      {
        "level": "Error",
        "dbObject": "test",
        "description": "before_insert_PRODUCT"
      }
  ]
}
```

預先檢查會報告錯誤，指出資料庫叢集有兩個具有相同名稱的觸發條件，但使用不同的案例：test.before_insert_product 和 test.before_insert_PRODUCT。

升級之前，請重新命名觸發條件或捨棄，然後使用新名稱重新建立觸發條件。

將 test.before_insert_PRODUCT 重新命名為 test.before_insert_product_2 後，預先檢查會成功。

```
{
  "id": "getDuplicateTriggers",
  "title": "MySQL pre-checks that all trigger names in a database are unique or
 not.",
  "status": "OK",
  "detectedProblems": []
}
```

getEventsWithNullDefiner

預先檢查層級：錯誤

mysql.event 的定義程式資料欄不能為 null 或空白。

DEFINER 屬性會指定擁有預存物件定義的 MySQL 帳戶，例如觸發條件、預存程序或事件。此屬性在您想要控制預存物件執行所在安全性內容的情況下特別有用。建立存放的物件時，如果 DEFINER 未指定 ，則預設值為建立物件的使用者。

升級至 MySQL 8.0 時，您無法在 MySQL 資料字典中有任何具有 null 或空白定義程式的預存物件。如果您有此類預存物件，則會引發預先檢查錯誤。您必須先修正此問題，才能繼續升級。

錯誤範例：

```
{
  "id": "getEventsWithNullDefiner",
  "title": "The definer column for mysql.event cannot be null or blank.",
  "status": "OK",
  "description": "Error: Set definer column in mysql.event to a valid non-null
 definer.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.get_version",
```

```
        "description": "Set definer for event get_version in Schema test"
      }
  ]
}
```

預先檢查會傳回 test.get_version 事件的錯誤，因為它具有 null 定義程式。

若要解決此問題，您可以檢查事件定義。如您所見，定義程式為 null 或空白。

```
mysql> select db,name,definer from mysql.event where name='get_version';
+------+-------------+---------+
| db   | name        | definer |
+------+-------------+---------+
| test | get_version |         |
+------+-------------+---------+
1 row in set (0.00 sec)
```

捨棄或重新建立具有有效定義程式的事件。

> **Note**
>
> 在捨棄或重新定義 DEFINER 之前，請仔細檢閱並檢查您的應用程式和權限需求。如需詳細資訊，請參閱 MySQL 文件中的預存物件存取控制。

```
mysql> drop event test.get_version;
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> delimiter $$
mysql> CREATE EVENT get_version
    ->     ON SCHEDULE
    ->       EVERY 1 DAY
    ->     DO
    ->      ///DO SOMETHING //
    -> $$
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;

mysql> select db,name,definer from mysql.event where name='get_version';
```

```
+------+-------------+------------+
| db   | name        | definer    |
+------+-------------+------------+
| test | get_version | reinvent@% |
+------+-------------+------------+
1 row in set (0.00 sec)
```

現在，預先檢查通過了。

```
{
  "id": "getEventsWithNullDefiner",
  "title": "The definer column for mysql.event cannot be null or blank.",
  "status": "OK",
  "detectedProblems": []},
```

getMismatchedMetadata

預先檢查層級：錯誤

InnoDB 資料字典和實際資料表定義之間的資料欄定義不相符

與 schemaInconsistencyCheck 類似，此預先檢查會先確認 MySQL 中的資料表中繼資料是否一致，再繼續升級。在此情況下，預先檢查會確認 InnoDB 資料字典和 MySQL 資料表定義之間的資料欄定義是否相符。如果偵測到不相符，則升級不會繼續。

如果您在此預先檢查中遇到任何錯誤，請向 AWS Support 開立案例，以請求解決中繼資料不一致的問題。或者，您可以執行邏輯傾印，然後還原至新的 Aurora MySQL 第 3 版資料庫叢集，以重試升級。

輸出範例：

```
{
  "id": "getMismatchedMetadata",
  "title": "Column definition mismatch between InnoDB Data Dictionary and actual
 table definition.",
  "status": "OK",
  "description": "Error: Your database has mismatched metadata. The upgrade to mysql
 8.0 will not succeed until this is fixed.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.mismatchTable",
```

```
        "description": "Table `test/mismatchTable` column names mismatch with InnoDb
 dictionary column names: iD <> id"
      }
  ]
}
```

預先檢查會報告 `test.mismatchTable` 資料表中 `id` 欄的中繼資料不相符。具體而言，MySQL 中繼資料的資料欄名稱為 `iD`，而 InnoDB 的資料欄名稱則為 `id`。

getTriggersWithNullDefiner

預先檢查層級：錯誤

`information_schema.triggers` 的定義程式資料欄不能為 `null` 或空白。

預先檢查會確認您的資料庫沒有使用 `null` 或空白定義程式定義的觸發條件。如需預存物件定義程式需求的詳細資訊，請參閱 getEventsWithNullDefiner。

輸出範例：

```
{
  "id": "getTriggersWithNullDefiner",
  "title": "The definer column for information_schema.triggers cannot be null or
 blank.",
  "status": "OK",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.example_trigger",
        "description": "Set definer for trigger example_trigger in Schema test"
      }
  ]
}
```

預先檢查會傳回錯誤，因為 `test` 結構描述中的 `example_trigger` 觸發條件具有 `null` 定義程式。若要修正此問題，請使用有效的使用者重新建立觸發條件，或捨棄觸發條件，以修正定義程式。如需詳細資訊，請參閱 getEventsWithNullDefiner 中的範例。

> **Note**
>
> 在捨棄或重新定義 `DEFINER` 之前，請仔細檢閱並檢查您的應用程式和權限需求。如需詳細資訊，請參閱 MySQL 文件中的預存物件存取控制。

getValueOfVariablelower_case_table_names

預先檢查層級：錯誤

lower_case_table_names 參數設定為 **1** 時，所有資料庫或資料表名稱都必須是小寫。

在 MySQL 8.0 之前，資料庫名稱、資料表名稱和其他物件會對應至資料目錄中的檔案，例如檔案型中繼資料 (.frm)。lower_case_table_names 系統變數可讓使用者控制伺服器如何處理資料庫物件的識別符區分大小寫，以及儲存此類中繼資料物件。重新啟動後，可以在已初始化的伺服器上變更此參數。

不過，在 MySQL 8.0 中，雖然此參數仍然控制伺服器如何處理識別符大小寫的區分性，但在資料字典初始化後就無法變更。升級或建立 MySQL 8.0 資料庫時，資料字典第一次在 MySQL 上啟動時針對 lower_case_table_names 所設定的值會用於該資料庫的生命週期。此限制是實作單元資料字典實作的一部分，其中資料庫物件會從檔案型中繼資料遷移到 mysql 結構描述中的內部 InnoDB 資料表。

如需詳細資訊，請參閱 MySQL 文件中的資料字典變更。

為了避免在升級期間將檔案型中繼資料更新為新的單元資料字典時發生問題，此預先檢查會驗證在 lower_case_table_names = 1 時，所有資料表都會以小寫儲存在磁碟上。如果沒有，則會傳回預先檢查錯誤，而且您必須先修正中繼資料，才能繼續升級。

輸出範例：

```
{
  "id": "getValueOfVariablelower_case_table_names",
  "title": "MySQL pre-checks that all database or table names are lowercase when the
 lower_case_table_names parameter is set to 1.",
  "status": "OK",
  "description": "Error: You have one or more databases or tables with uppercase
 letters in the names, but the lower_case_table_names parameter is set to 1. To
 upgrade to MySQL 8.0, either change all database or table names to lowercase, or
 set the parameter to 0.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.TEST",
        "description": "Table test.TEST contains one or more capital letters in name
 while lower_case_table_names = 1"
      }
```

```
  ]
}
```

因為資料表 `test.TEST` 包含大寫字母，但 `lower_case_table_names` 設定為 `1`，所以傳回錯誤。

若要解決此問題，您可以重新命名資料表以使用小寫，或修改資料庫叢集上的 `lower_case_table_names` 參數，然後再開始升級。

> **Note**
>
> 仔細測試和檢閱 MySQL 中區分大小寫的文件，以及任何此類變更如何影響您的應用程式。另請檢閱 MySQL 8.0 文件，了解如何在 MySQL 8.0 中以不同方式處理 lower_case_table_names。

groupByAscSyntaxCheck

預先檢查層級：錯誤

已移除 **GROUP BY ASC/DESC** 語法的使用

從 MySQL 8.0.13 開始，已移除 GROUP BY 子句的已棄用 ASC 或 DESC 語法。依賴 GROUP BY 排序的查詢現在可能會產生不同的結果。若要取得特定的排序順序，請改用 ORDER BY 子句。如果資料庫中有任何物件使用此語法，您必須在重試升級之前，使用 ORDER BY 子句重新建立物件。如需更多詳細資訊，請參閱 MySQL 文件中的 SQL 變更。

輸出範例：

```
{
  "id": "groupbyAscSyntaxCheck",
  "title": "Usage of removed GROUP BY ASC/DESC syntax",
  "status": "OK",
  "description": "Error: The following DB objects use removed GROUP BY ASC/DESC
 syntax. They need to be altered so that ASC/DESC keyword is removed from GROUP BY
 clause and placed in appropriate ORDER BY clause.",
  "documentationLink": "https://dev.mysql.com/doc/relnotes/mysql/8.0/en/
news-8-0-13.html#mysqld-8-0-13-sql-syntax",
  "detectedProblems": [
      {
```

```
        "level": "Error",
        "dbObject": "test.groupbyasc",
        "description": "PROCEDURE uses removed GROUP BY ASC syntax",
        "dbObjectType": "Routine"
      }
  ]
}
```

mysqlEmptyDotTableSyntaxCheck

預先檢查層級：錯誤

檢查常式中使用的已棄用 **.<table>** 語法。

在 MySQL 8.0 中，常式無法再包含已棄用的識別符語法 (\".<table>\")。如果任何預存常式或觸發條件包含此類識別符，則升級會失敗。例如，不再允許下列 .dot_table 參考：

```
mysql> show create procedure incorrect_procedure\G
*************************** 1. row ***************************
           Procedure: incorrect_procedure
            sql_mode:
    Create Procedure: CREATE DEFINER=`reinvent`@`%` PROCEDURE
 `incorrect_procedure`()
BEGIN delete FROM .dot_table; select * from .dot_table where 1=1; END
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

重新建立常式和觸發程序以使用正確的識別符語法和逸出之後，預先檢查會通過，而且升級可以繼續進行。如需詳細資訊，請參閱 PostgreSQL 文件中的結構描述物件名稱。

輸出範例：

```
{
  "id": "mysqlEmptyDotTableSyntaxCheck",
  "title": "Check for deprecated '.<table>' syntax used in routines.",
  "status": "OK",
  "description": "Error: The following routines contain identifiers in deprecated
 identifier syntax (\".<table>\"), and should be corrected before upgrade:\n",
  "detectedProblems": [
      {
```

```
        "level": "Error",
        "dbObject": "test.incorrect_procedure",
        "description": " routine body contains deprecated identifiers."
      }
  ]
}
```

預先檢查會傳回 `test` 資料庫中 `incorrect_procedure` 常式的錯誤，因為它包含已棄用的語法。

修正常式後，預先檢查會成功，您可以重試升級。

mysqlIndexTooLargeCheck

預先檢查層級：錯誤

檢查索引是否太大而無法在高於 5.7 的 MySQL 版本上運作

對於精簡或備援資料列格式，不應能夠建立字首大於 767 個位元組的索引。不過，在 MySQL 5.7.35 版之前，這是可能的。如需詳細資訊，請參閱 MySQL 5.7.35 版本備註。

升級到 MySQL 8.0 後，任何受此錯誤影響的索引都將無法存取。此預先檢查會識別必須在允許升級繼續之前重建的違規索引。

```
{
 "id": "mysqlIndexTooLargeCheck",
 "title": "Check for indexes that are too large to work on higher versions of MySQL
Server than 5.7",
 "status": "OK",
 "description": "Error: The following indexes ware made too large for their format
in an older version of MySQL (older than 5.7.34). Normally those indexes within
tables with compact or redundant row formats shouldn't be larger than 767 bytes. To
fix this problem those indexes should be dropped before upgrading or those tables
will be inaccessible.",
 "detectedProblems": [
     {
       "level": "Error",
       "dbObject": "test.table_with_large_idx",
       "description": "IDX_2"
     }
  ]
}
```

預先檢查會傳回錯誤，因為 `test.table_with_large_idx` 資料表包含資料表上的索引，其使用大於 767 個位元組的精簡或備援資料列格式。升級到 MySQL 8.0 後，這些資料表將變得無法存取。在繼續升級之前，請執行下列其中一項操作：

- 捨棄預先檢查中提及的索引。
- 新增預先檢查中提及的索引。
- 變更資料表所使用的資料列格式。

在這裡，我們會重建資料表以解決預先檢查失敗。在重建資料表之前，請確定 innodb_file_format 設定為 `Barracuda`，且 innodb_default_row_format 設定為 `dynamic`。這些是 MySQL 5.7 中的預設值。如需詳細資訊，請參閱 MySQL 文件中的 InnoDB 資料列格式和 InnoDB 檔案格式管理。

> **Note**
>
> 在重建資料表空間之前，請參閱 MySQL 文件中的線上 DDL 操作，以了解鎖定和資料移動對前景交易的影響。

```
mysql > select @@innodb_file_format,@@innodb_default_row_format;
+----------------------+-----------------------------+
| @@innodb_file_format | @@innodb_default_row_format |
+----------------------+-----------------------------+
| Barracuda            | dynamic                     |
+----------------------+-----------------------------+
1 row in set (0.00 sec)

mysql> optimize table table_with_large_idx;
+---------------------------+----------+----------
+-------------------------------------------------------------------+
| Table                     | Op       | Msg_type | Msg_text
                                |
+---------------------------+----------+----------
+-------------------------------------------------------------------+
| test.table_with_large_idx | optimize | note     | Table does not support optimize,
 doing recreate + analyze instead |
| test.table_with_large_idx | optimize | status   | OK
                                |
+---------------------------+----------+----------
+-------------------------------------------------------------------+
2 rows in set (0.02 sec)
```

```
# Verify FILE_FORMAT and ROW_FORMAT
mysql>  select * from information_schema.innodb_sys_tables where name like 'test/
table_with_large_idx';
+----------+---------------------------+------+--------+-------+-------------
+------------+---------------+------------+
| TABLE_ID | NAME                      | FLAG | N_COLS | SPACE | FILE_FORMAT |
 ROW_FORMAT | ZIP_PAGE_SIZE | SPACE_TYPE |
+----------+---------------------------+------+--------+-------+-------------
+------------+---------------+------------+
|       43 | test/table_with_large_idx |   33 |      4 |    26 | Barracuda   |
 Dynamic    |             0 | Single     |
+----------+---------------------------+------+--------+-------+-------------
+------------+---------------+------------+
1 row in set (0.00 sec)
```

重建資料表後，預先檢查會通過，而且升級可以繼續。

```
{
  "id": "mysqlIndexTooLargeCheck",
  "title": "Check for indexes that are too large to work on higher versions of MySQL
 Server than 5.7",
  "status": "OK",
  "detectedProblems": []
},
```

mysqlInvalid57NamesCheck

預先檢查層級：錯誤

檢查 MySQL 5.7 中使用的無效資料表和結構描述名稱

遷移至 MySQL 8.0 中的新資料字典時，您的資料庫執行個體不能包含字首為 #mysql50# 的結構描述或資料表。如果有任何此類物件存在，則升級會失敗。若要解決此問題，請針對傳回的結構描述和資料表執行 mysqlcheck。

> **Note**
>
> 請務必使用 MySQL 5.7 版本的 mysqlcheck，因為 --fix-db-names 和 --fix-table-names 已從 MySQL 8.0 中移除。

輸出範例：

```
{
  "id": "mysqlInvalid57NamesCheck",
  "title": "Check for invalid table names and schema names used in 5.7",
  "status": "OK",
  "description": "The following tables and/or schemas have invalid names. In order
 to fix them use the mysqlcheck utility as follows:\n\n  $ mysqlcheck --check-
upgrade --all-databases\n  $ mysqlcheck --fix-db-names --fix-table-names --all-
databases\n\nOR via mysql client, for eg:\n\n  ALTER DATABASE `#mysql50#lost+found`
 UPGRADE DATA DIRECTORY NAME;",
  "documentationLink": "https://dev.mysql.com/doc/refman/5.7/en/identifier-
mapping.html https://dev.mysql.com/doc/refman/5.7/en/alter-database.html https://
dev.mysql.com/doc/refman/8.0/en/mysql-nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "#mysql50#fix_db_names",
        "description": "Schema name"
      }
  ]
}
```

預先檢查會報告結構描述 #mysql50#fix_db_names 的名稱無效。

修正結構描述名稱後，預先檢查會通過，允許升級繼續進行。

```
{
  "id": "mysqlInvalid57NamesCheck",
  "title": "Check for invalid table names and schema names used in 5.7",
  "status": "OK",
  "detectedProblems": []
},
```

mysqlOrphanedRoutinesCheck

預先檢查層級：錯誤

檢查 5.7 中的孤立常式

遷移至 MySQL 8.0 中的新資料字典時，如果資料庫中有任何預存程序不再存在結構描述，則升級會失敗。此預先檢查會確認資料庫執行個體上預存程序中參考的所有結構描述是否仍然存在。若要允許繼續升級，請捨棄這些預存程序。

輸出範例：

```
{
  "id": "mysqlOrphanedRoutinesCheck",
  "title": "Check for orphaned routines in 5.7",
  "status": "OK",
  "description": "Error: The following routines have been orphaned. Schemas that
 they are referencing no longer exists.\nThey have to be cleaned up or the upgrade
 will fail.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "dropped_db.get_version",
        "description": "is orphaned"
      }
  ]
},
```

預先檢查會報告 dropped_db 資料庫中的 get_version 預存程序是孤立的。

若要清除此程序，您可以重新建立缺少的結構描述。

```
mysql> create database dropped_db;
Query OK, 1 row affected (0.01 sec)
```

重新建立結構描述之後，您可以捨棄該程序，以允許升級繼續進行。

```
{
  "id": "mysqlOrphanedRoutinesCheck",
  "title": "Check for orphaned routines in 5.7",
  "status": "OK",
  "detectedProblems": []
},
```

mysqlSchemaCheck

預先檢查層級：錯誤

mysql 結構描述中的資料表名稱與 MySQL 8.0 中的新資料表衝突

MySQL 8.0 中引入的新單元資料字典會將所有中繼資料存放在 mysql 結構描述中的一組內部 InnoDB 資料表中。在升級期間，系統會在 mysql 結構描述中建立新的內部資料字典資料表。為了避免導致升級失敗的命名衝突，預先檢查會檢查 mysql 結構描述中的所有資料表名稱，以確保沒有任何新的資料表名稱已在使用中。如果是，則會傳回錯誤，而且不允許繼續升級。

輸出範例：

```
{
  "id": "mysqlSchema",
  "title": "Table names in the mysql schema conflicting with new tables in the
 latest MySQL.",
  "status": "OK",
  "description": "Error: The following tables in mysql schema have names that will
 conflict with the ones introduced in the latest version. They must be renamed or
 removed before upgrading (use RENAME TABLE command). This may also entail changes
 to applications that use the affected tables.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrade-before-you-
begin.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "mysql.tablespaces",
        "description": "Table name used in mysql schema.",
        "dbObjectType": "Table"
      }
  ]
}
```

因為 mysql 結構描述中有名為 tablespaces 的資料表，所以傳回錯誤。這是 MySQL 8.0 中新的內部資料字典資料表名稱之一。您必須在升級之前使用 RENAME TABLE 命令重新命名或移除任何此類資料表。

nonNativePartitioningCheck

預先檢查層級：錯誤

使用具有非原生分割之引擎的分割資料表

根據 MySQL 8.0 文件，兩個儲存引擎目前提供原生分割支援：InnoDB 和 NDB。其中，Aurora MySQL 第 3 版僅支援 InnoDB，與 MySQL 8.0 相容。使用任何其他儲存引擎在 MySQL 8.0 中建立分割資料表的任何嘗試都會失敗。此預先檢查會在資料庫叢集中尋找使用非原生分割的資料表。如果傳回任何結果，您必須移除分割或將儲存引擎轉換為 InnoDB。

輸出範例：

```
{
  "id": "nonNativePartitioning",
  "title": "Partitioned tables using engines with non native partitioning",
```

```
  "status": "OK",
  "description": "Error: In the latest MySQL storage engine is responsible for
 providing its own partitioning handler, and the MySQL server no longer provides
 generic partitioning support. InnoDB and NDB are the only storage engines that
 provide a native partitioning handler that is supported in the latest MySQL. A
 partitioned table using any other storage engine must be altered—either to convert
 it to InnoDB or NDB, or to remove its partitioning—before upgrading the server,
 else it cannot be used afterwards.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-configuration-changes",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.partMyisamTable",
         "description": "MyISAM engine does not support native partitioning",
         "dbObjectType": "Table"
      }
  ]
}
```

在這裡，MyISAM 資料表正在使用分割，這需要採取動作才能繼續升級。

oldTemporalCheck

預先檢查層級：錯誤

舊暫時類型的使用

「舊暫時」是指在 MySQL 5.5 版和更低版本中建立的暫時類型資料欄 (例如 TIMESTAMP 和 DATETIME)。在 MySQL 8.0 中，系統會移除對這些舊暫時資料類型的支援，這表示如果資料庫包含這些舊暫時類型，就無法從 MySQL 5.7 就地升級至 8.0。若要修正此問題，您必須先重建任何包含此類舊暫時日期類型的資料表，才能繼續升級。

如需 MySQL 5.7 中舊暫時資料類型棄用的詳細資訊，請參閱此部落格。如需在 MySQL 8.0 中移除舊暫時資料類型的詳細資訊，請參閱此部落格。

> **Note**
>
> 在重建資料表空間之前，請參閱 MySQL 文件中的線上 DDL 操作，以了解鎖定和資料移動對前景交易的影響。

輸出範例：

```
{
  "id": "oldTemporalCheck",
  "title": "Usage of old temporal type",
  "status": "OK",
  "description": "Error: Following table columns use a deprecated and no longer
 supported temporal disk storage format. They must be converted to the new format
 before upgrading. It can by done by rebuilding the table using 'ALTER TABLE
 <table_name> FORCE' command",
  "documentationLink": "https://dev.mysql.com/blog-archive/mysql-8-0-removing-
support-for-old-temporal-datatypes/",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.55_temporal_table.timestamp_column",
        "description": "timestamp /* 5.5 binary format */",
        "dbObjectType": "Column"
      }
  ]
},
```

會針對資料表 timestamp_column 中的資料欄 test.55_temporal_table 報告錯誤，因為它使用不再支援的舊暫時磁碟儲存格式。

若要解決此問題並允許升級繼續進行，請重建資料表，將舊暫時磁碟儲存格式轉換為 MySQL 5.6 中引入的新格式。如需詳細資訊和先決條件，請參閱 MySQL 文件中的在 3 位元組和 4 位元組 Unicode 字元集之間轉換。

執行下列命令來重建此預先檢查中提及的資料表，會將舊暫時類型轉換為較新的格式，精確度為小數秒數。

```
ALTER TABLE ... ENGINE=InnoDB;
```

如需重建資料表的詳細資訊，請參閱 MySQL 文件中的 ALTER TABLE 陳述式。

在重建有問題的資料表並重新啟動升級之後，相容性檢查會通過，讓升級繼續進行。

```
{
  "id": "oldTemporalCheck",
  "title": "Usage of old temporal type",
  "status": "OK",
  "detectedProblems": []
```

```
}
```

partitionedTablesInSharedTablespaceCheck

預先檢查層級：錯誤

在共用資料表空間中使用分割資料表

從 MySQL 8.0.13 開始，已移除在共用資料表空間中放置資料表分割區的支援。升級之前，請將任何此類資料表從共用資料表空間移至 file-per-table 資料表空間。

> **Note**
>
> 在重建資料表空間之前，請參閱 MySQL 文件中的分割操作，以了解鎖定和資料移動對前景交易的影響。

輸出範例：

```
{
  "id": "partitionedTablesInSharedTablespaceCheck",
  "title": "Usage of partitioned tables in shared tablespaces",
  "status": "OK",
  "description": "Error: The following tables have partitions in shared tablespaces.
 They need to be moved to file-per-table tablespace before upgrading. You can do
 this by running query like 'ALTER TABLE table_name REORGANIZE PARTITION X INTO
 (PARTITION X VALUES LESS THAN (30) TABLESPACE=innodb_file_per_table);'",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/mysql-
nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.partInnoDBTable",
        "description": "Partition p1 is in shared tablespace innodb",
        "dbObjectType": "Table"
      }
  ]
}
```

預先檢查失敗，因為來自資料表 `test.partInnoDBTable` 的分割區 `p1` 位於系統資料表空間中。

若要解決此問題，請重建 `test.partInnodbTable` 資料表，將違規分割區放在 file-per-table 資料表空間中的 p1。

```
mysql > ALTER TABLE partInnodbTable REORGANIZE PARTITION p1
    ->   INTO (PARTITION p1 VALUES LESS THAN ('2014-01-01')
 TABLESPACE=innodb_file_per_table);
Query OK, 0 rows affected, 1 warning (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

執行此操作後，預先檢查會通過。

```
{
  "id": "partitionedTablesInSharedTablespaceCheck",
  "title": "Usage of partitioned tables in shared tablespaces",
  "status": "OK",
  "detectedProblems": []
}
```

removedFunctionsCheck

預先檢查層級：錯誤

使用從最新 MySQL 版本中移除的函數

在 MySQL 8.0 中，已移除許多內建函數。此預先檢查會檢查您的資料庫是否有可能使用這些函數的物件。如果找到，則會傳回錯誤。您必須先解決問題，才能重試升級。

移除的大多數函數都是空間函數，已被同等 `ST_*` 函數取代。在這些情況下，您可以修改資料庫物件以使用新的程序命名。如需詳細資訊，請參閱 MySQL 文件中的 Features Removed in MySQL 8.0 (MySQL 8.0 中移除的功能)。

輸出範例：

```
{
  "id": "removedFunctionsCheck",
  "title": "Usage of removed functions",
  "status": "OK",
  "description": "Error: The following DB objects use functions that were removed
 in the latest MySQL version. Please make sure to update them to use supported
 alternatives before upgrade.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/mysql-
nutshell.html#mysql-nutshell-removals",
```

```
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.GetLocationsInPolygon",
        "description": "PROCEDURE uses removed function POLYGONFROMTEXT (consider
 using ST_POLYGONFROMTEXT instead)",
        "dbObjectType": "Routine"
      },
      {
        "level": "Error",
        "dbObject": "test.InsertLocation",
        "description": "PROCEDURE uses removed function POINTFROMTEXT (consider
 using ST_POINTFROMTEXT instead)",
        "dbObjectType": "Routine"
      }
  ]
},
```

預先檢查會報告 `test.GetLocationsInPolygon` 預存程序正在使用兩個已移除的函數：POLYGONFROMTEXT 和 POINTFROMTEXT。它也建議您使用新的 ST_POLYGONFROMTEXT 和 ST_POINTFROMTEXT 來取代。使用建議重新建立程序後，預先檢查會成功完成。

```
{
  "id": "removedFunctionsCheck",
  "title": "Usage of removed functions",
  "status": "OK",
  "detectedProblems": []
},
```

> **Note**
>
> 雖然在大多數情況下，已棄用的函數具有直接取代項目，但請務必測試您的應用程式，並檢閱文件以了解因變更而導致的任何行為變更。

routineSyntaxCheck

預先檢查層級：錯誤

MySQL 語法檢查類似常式的物件

MySQL 8.0 引入了一些先前未保留的預留關鍵字。升級預先檢查程式會評估資料庫物件名稱及其定義和本文中預留關鍵字的使用情況。如果它偵測到資料庫物件中使用的預留關鍵字，例如預存程序、函數、事件和觸發條件，則升級會失敗，並發佈錯誤至 upgrade-prechecks.log 檔案。若要解決此問題，您必須先更新這些物件定義，並在升級之前以單引號 (') 括住任何此類參考。如需在 MySQL 中逸出預留字的詳細資訊，請參閱 MySQL 文件中的字串常值。

或者，您可以將名稱變更為不同的名稱，這可能需要應用程式變更。

輸出範例：

```
{
  "id": "routineSyntaxCheck",
  "title": "MySQL syntax check for routine-like objects",
  "status": "OK",
  "description": "The following objects did not pass a syntax check with the latest
 MySQL grammar. A common reason is that they reference names that conflict with new
 reserved keywords. You must update these routine definitions and `quote` any such
 references before upgrading.",
  "documentationLink": "https://dev.mysql.com/doc/refman/en/keywords.html",
  "detectedProblems": [
      {
         "level": "Error",
         "dbObject": "test.select_res_word",
         "description": "at line 2,18: unexpected token 'except'",
         "dbObjectType": "Routine"
      }
  ]
}
```

若要修正此問題，請檢查常式定義。

```
SHOW CREATE PROCEDURE test.select_res_word\G

*************************** 1. row ***************************
           Procedure: select_res_word
            sql_mode:
 ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_Z
    Create Procedure: CREATE PROCEDURE 'select_res_word'()
BEGIN
    SELECT * FROM except;
END
character_set_client: utf8
collation_connection: utf8_general_ci
```

```
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

程序使用名為 except 的資料表，這是 MySQL 8.0 中的新關鍵字。透過逸出字串常值來重新建立程序。

```
> drop procedure if exists select_res_word;
Query OK, 0 rows affected (0.00 sec)

> DELIMITER $$
 > CREATE PROCEDURE select_res_word()
    -> BEGIN
    ->     SELECT * FROM 'except';
    -> END$$
Query OK, 0 rows affected (0.00 sec)

 > DELIMITER ;
```

預先檢查現在會通過。

```
{
  "id": "routineSyntaxCheck",
  "title": "MySQL syntax check for routine-like objects",
  "status": "OK",
  "detectedProblems": []
}
```

schemaInconsistencyCheck

預先檢查層級：錯誤

因檔案移除或損毀所導致的結構描述不一致

如前所述，MySQL 8.0 推出了單元資料字典，該字典會將所有中繼資料存放在 mysql 結構描述中的一組內部 InnoDB 資料表中。這種新架構提供交易、ACID 合規的方式來管理資料庫中繼資料，從舊的檔案型方法解決「單元 DDL」問題。在 MySQL 8.0 之前，如果 DDL 操作意外中斷，結構描述物件可能會變得孤立。在升級期間將檔案型中繼資料遷移至新的單元資料字典資料表，可確保資料庫執行個體中沒有此類孤立結構描述物件。如果遇到任何孤立的物件，升級會失敗。

為了協助在啟動升級之前偵測這些孤立物件，系統會執行 schemaInconsistencyCheck 預先檢查，以確保所有資料字典中繼資料物件都同步。如果偵測到任何孤立的中繼資料物件，則不會繼續升級。若要繼續升級，請清除這些孤立的中繼資料物件。

如果您在此預先檢查中遇到任何錯誤，請向 AWS Support 開立案例，以請求解決中繼資料不一致的問題。或者，您可以執行邏輯傾印，然後還原至新的 Aurora MySQL 第 3 版資料庫叢集，以重試升級。

輸出範例：

```
{
  "id": "schemaInconsistencyCheck",
  "title": "Schema inconsistencies resulting from file removal or corruption",
  "status": "OK",
  "description": "Error: Following tables show signs that either table datadir
 directory or frm file was removed/corrupted. Please check server logs, examine
 datadir to detect the issue and fix it before upgrade",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.schemaInconsistencyCheck_failure",
        "description": "present in INFORMATION_SCHEMA's INNODB_SYS_TABLES table but
 missing from TABLES table"
      }
  ]
}
```

預先檢查會報告 `test.schemaInconsistencyCheck_failure` 資料表的中繼資料不一致。在此情況下，資料表存在於 InnoDB 儲存引擎中繼資料 (`information_schema.INNODB_SYS_TABLES`) 中，但不存在於 MySQL 中繼資料 (`information_schema.TABLES`) 中。

Aurora MySQL 會預先檢查報告錯誤

下列預先檢查專屬於 Aurora MySQL：

- auroraCheckDDLRecovery
- auroraCheckRdsUpgradePrechecksTable
- auroraFODUpgradeCheck
- auroraGetDanglingFulltextIndex
- auroraUpgradeCheckForDatafilePathInconsistency
- auroraUpgradeCheckForFtsSpaceIdZero
- auroraUpgradeCheckForIncompleteXATransactions

- auroraUpgradeCheckForInstanceLimit
- auroraUpgradeCheckForInternalUsers
- auroraUpgradeCheckForInvalidUtf8mb3CharacterStringInViews
- auroraUpgradeCheckForInvalidUtf8mb3ColumnComments
- auroraUpgradeCheckForInvalidUtf8mb3IndexComments
- auroraUpgradeCheckForInvalidUtf8mb3TableComments
- auroraUpgradeCheckForMasterUser
- auroraUpgradeCheckForPrefixIndexOnGeometryColumns
- auroraUpgradeCheckForSpecialCharactersInProcedures
- auroraUpgradeCheckForSysSchemaObjectTypeMismatch
- auroraUpgradeCheckForViewColumnNameLength
- auroraUpgradeCheckIndexLengthLimitOnTinyColumns
- auroraUpgradeCheckMissingInnodbMetadataForMysqlHostTable

auroraCheckDDLRecovery

預先檢查層級：錯誤

檢查與 Aurora DDL 復原功能相關的成品

作為 Aurora MySQL 中資料定義語言 (DDL) 復原實作的一部分，傳輸中 DDL 陳述式的中繼資料會保留在 `ddl_log_table` 結構描述中的 `ddl_log_md_table` 和 `mysql` 資料表中。第 3 版以後不支援 Aurora 的 DDL 復原實作，因為此功能是 MySQL 8.0 中新單元資料字典實作的一部分。如果您在相容性檢查期間執行任何 DDL 陳述式，此預先檢查可能會失敗。我們建議您在沒有 DDL 陳述式執行時嘗試升級。

如果此預先檢查失敗，而沒有任何執行中的 DDL 陳述式，請向 AWS Support 開立案例，以請求解決中繼資料不一致的問題。或者，您可以執行邏輯傾印，然後還原至新的 Aurora MySQL 第 3 版資料庫叢集，以重試升級。

如果有任何 DDL 陳述式正在執行，預先檢查輸出會列印下列訊息：

```
“There are DDL statements in process. Please allow DDL statements to finish before
 upgrading.”
```

輸出範例：

```
{
  "id": "auroraCheckDDLRecovery",
  "title": "Check for artifacts related to Aurora DDL recovery feature",
  "status": "OK",
  "description": "Aurora implementation of DDL recovery is not supported from 3.x
 onwards. This check verifies that the database do not have artifacts realted to the
 feature",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "mysql.ddl_log_md_table",
        "description": "Table mysql.ddl_log_md_table is not empty. Your database has
 pending DDL recovery operations. Reachout to AWS support for assistance."
      },
      {
        "level": "Error",
        "dbObject": "mysql.ddl_log_table",
        "description": "Table mysql.ddl_log_table is not empty. Your database has
 pending DDL recovery operations. Reachout to AWS support for assistance."
      },
      {
        "level": "Error",
        "dbObject": "information_schema.processlist",
        "description": "There are DDL statements in process. Please allow DDL
 statements to finish before upgrading."
      }
  ]
}
```


由於與相容性檢查同時執行的傳輸中 DDL，預先檢查傳回錯誤。我們建議您在未執行任何 DDL 的情況下重試升級。

auroraCheckRdsUpgradePrechecksTable

預先檢查層級：錯誤

檢查 **mysql.rds_upgrade_prechecks** 資料表是否存在

這是 RDS 服務執行的僅限內部預先檢查。升級時會自動處理任何錯誤，並且可以安全地忽略。

如果您在此預先檢查中遇到任何錯誤，請向 AWS Support 開立案例，以請求解決中繼資料不一致的問題。或者，您可以執行邏輯傾印，然後還原至新的 Aurora MySQL 第 3 版資料庫叢集，以重試升級。

```
{
  "id": "auroraCheckRdsUpgradePrechecksTable",
  "title": "Check existence of mysql.rds_upgrade_prechecks table",
  "status": "OK",
  "detectedProblems": []
}
```

auroraFODUpgradeCheck

預先檢查層級：錯誤

檢查與 Aurora 快速 DDL 功能相關的成品

快速 DDL 最佳化是在 Aurora MySQL 第 2 版的實驗室模式中推出，以提高某些 DDL 操作的效率。在 Aurora MySQL 第 3 版中，實驗室模式已移除，快速 DDL 實作已由名為即時 DDL 的 MySQL 8.0 功能取代。

在升級至 Aurora MySQL 第 3 版之前，任何在實驗室模式中使用快速 DDL 的資料表都必須執行 `OPTIMIZE TABLE` 或 `ALTER TABLE ... ENGINE=InnoDB` 命令來重建，以確保與 Aurora MySQL 第 3 版的相容性。

此預先檢查會傳回任何此類資料表的清單。重建傳回的資料表之後，您可以重試升級。

輸出範例：

```
{
  "id": "auroraFODUpgradeCheck",
  "title": "Check for artifacts related to Aurora fast DDL feature",
  "status": "OK",
  "description": "Aurora fast DDL is not supported from 3.x onwards. This check
 verifies that the database does not have artifacts related to the feature",
  "documentationLink": "https://docs.aws.amazon.com/AmazonRDS/latest/
AuroraUserGuide/AuroraMySQL.Managing.FastDDL.html#AuroraMySQL.Managing.FastDDL-v2",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.test",
        "description": "Your table has pending Aurora fast DDL operations. Run
 'OPTIMIZE TABLE <table name>' for the table to apply all the pending DDL updates.
 Then try the upgrade again."
      }
  ]
```

```
}
```

預先檢查會報告資料表 `test.test` 有待定的快速 DDL 操作。

若要允許繼續升級，您可以重建資料表，然後重試升級。

> **Note**
>
> 在重建資料表空間之前，請參閱 MySQL 文件中的線上 DDL 操作，以了解鎖定和資料移動對前景交易的影響。

```
mysql> optimize table test.test;
+-----------+----------+----------
+-------------------------------------------------------------------+
| Table     | Op       | Msg_type | Msg_text
                  |
+-----------+----------+----------
+-------------------------------------------------------------------+
| test.test | optimize | note     | Table does not support optimize, doing recreate
 + analyze instead |
| test.test | optimize | status   | OK
                  |
+-----------+----------+----------
+-------------------------------------------------------------------+
2 rows in set (0.04 sec)
```

重建資料表後，預先檢查會成功。

```
{
  "id": "auroraFODUpgradeCheck",
  "title": "Check for artifacts related to Aurora fast DDL feature",
  "status": "OK",
  "detectedProblems": []
}
```

auroraGetDanglingFulltextIndex

預先檢查層級：錯誤

具有懸置 **FULLTEXT** 索引參考的資料表

在 MySQL 5.6.26 之前，捨棄全文搜尋索引後，隱藏的 FTS_DOC_ID 和 FTS_DOC_ID_INDEX 資料欄可能會變得孤立。如需詳細資訊，請參閱錯誤 #76012。

如果您在發生這種情況的舊版 MySQL 上建立任何資料表，可能會導致升級至 Aurora MySQL 第 3 版失敗。此預先檢查會確認資料庫叢集上沒有此類孤立或「懸置」的全文索引，然後再升級至 MySQL 8.0。如果此預先檢查失敗，請重建包含此類懸置全文索引的任何資料表。

輸出範例：

```
{
  "id": "auroraGetDanglingFulltextIndex",
  "title": "Tables with dangling FULLTEXT index reference",
  "status": "OK",
  "description": "Error: The following tables contain dangling FULLTEXT index which
 is not supported. It is recommended to rebuild the table before upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.table_with_fts_index",
        "description": "Table `test.table_with_fts_index` contains dangling FULLTEXT
 index. Kindly recreate the table before upgrade."
      }
  ]
},
```

預先檢查會報告 test.table_with_fts_index 資料表的錯誤，因為它包含懸置的全文索引。若要允許繼續升級，請重建資料表以清除全文索引輔助資料表。使用 OPTIMIZE TABLE test.table_with_fts_index 或 ALTER TABLE test.table_with_fts_index, ENGINE=INNODB。

重建資料表後，預先檢查會通過。

```
{
  "id": "auroraGetDanglingFulltextIndex",
  "title": "Tables with dangling FULLTEXT index reference",
  "status": "OK",
  "detectedProblems": []
},
```

auroraUpgradeCheckForDatafilePathInconsistency

預先檢查層級：錯誤

檢查與 **ibd** 檔案路徑相關的不一致

此預先檢查僅適用於 Aurora MySQL 3.03.0 版及更低版本。如果此預先檢查發生錯誤，請升級至 Aurora MySQL 3.04 版或更新版本。

輸出範例：

```
{
  "id": "auroraUpgradeCheckForDatafilePathInconsistency",
  "title": "Check for inconsistency related to ibd file path.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForFtsSpaceIdZero

預先檢查層級：錯誤

檢查空間 ID 為零的全文索引

在 MySQL 中，當您將全文索引新增至 InnoDB 資料表時，會建立多個輔助索引資料表空間。由於舊版 MySQL 在 5.6.20 版中修正的錯誤，這些輔助索引資料表可能在系統資料表空間中建立，而不是在自己的 InnoDB 資料表空間中建立。

如果有任何此類輔助資料表空間存在，升級將會失敗。重新建立預先檢查錯誤中提及的全文索引，然後重試升級。

輸出範例：

```
{
  "id": "auroraUpgradeCheckForFtsSpaceIdZero",
  "title": "Check for fulltext index with space id as zero",
  "status": "OK",
  "description": "The auxiliary tables of FTS indexes on the table are created
 in system table-space. Due to this DDL queries executed on MySQL8.0 shall cause
 database unavailability. To avoid that, drop and recreate all the FTS indexes on
 the table or rebuild the table using ALTER TABLE query before the upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.fts_space_zero_check",
        "description": " The auxiliary tables of FTS indexes on the table
 'test.fts_space_zero_check' are created in system table-space due to https://
```

```
bugs.mysql.com/bug.php?id=72132. In MySQL8.0, DDL queries executed on this table
 shall cause database unavailability. To avoid that, drop and recreate all the
 FTS indexes on the table or rebuild the table using ALTER TABLE query before the
 upgrade."
      }
  ]
},
```

預先檢查會報告 `test.fts_space_zero_check` 資料表的錯誤，因為它在系統資料表空間中有輔助全文搜尋 (FTS) 資料表。

在您捨棄並重新建立與此資料表相關聯的 FTS 索引後，預先檢查會成功。

> **Note**
>
> 在重建資料表空間之前，請參閱 MySQL 文件中的分割操作，以了解鎖定和資料移動對前景交易的影響。

```
{
 "id": "auroraUpgradeCheckForFtsSpaceIdZero",
 "title": "Check for fulltext index with space id as zero",
 "status": "OK",
 "detectedProblems": []
}
```

auroraUpgradeCheckForIncompleteXATransactions

預先檢查層級：錯誤

檢查 XA 交易是否處於預備狀態

執行主要版本升級程序時，Aurora MySQL 第 2 版資料庫執行個體必須進行正常關閉。這可確保所有交易都已遞交或復原，而且 InnoDB 已清除所有復原日誌記錄。由於交易復原是必要的，如果您的資料庫有任何 XA 交易處於預備狀態，它可能會封鎖正常關閉，使其無法繼續。因此，如果偵測到任何預備 XA 交易，在您採取動作遞交或復原之前，升級將無法繼續。

如需使用 `XA RECOVER` 尋找處於預備狀態的 XA 交易的詳細資訊，請參閱 MySQL 文件中的 XA 交易 SQL 陳述式。如需 XA 交易狀態的詳細資訊，請參閱 MySQL 文件中的 XA 交易狀態。

輸出範例：

```
{
  "id": "auroraUpgradeCheckForIncompleteXATransactions",
  "title": "Pre-checks for XA Transactions in prepared state.",
  "status": "OK",
  "description": "Your cluster currently has XA transactions in the prepared state.
 To proceed with the upgrade, commit or rollback these transactions.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "all",
        "description": "Your cluster currently has XA transactions in the prepared
 state. To proceed with the upgrade, commit or rollback these transactions."
      }
  ]
}
```

此預先檢查會報告錯誤，因為有交易處於預備狀態，應遞交或復原。

登入資料庫後，您可以檢查 information_schema.innodb_trx 資料表和 XA RECOVER 輸出以取得詳細資訊。

> **Important**
>
> 在遞交或復原交易之前，我們建議您檢閱 MySQL 文件和應用程式需求。

```
mysql> select trx_started,
    trx_mysql_thread_id,
    trx_id,trx_state,
    trx_operation_state,
    trx_rows_modified,
    trx_rows_locked
from
    information_schema.innodb_trx;
+---------------------+---------------------+---------+-----------
+---------------------+-------------------+-----------------+
| trx_started         | trx_mysql_thread_id | trx_id  | trx_state |
 trx_operation_state | trx_rows_modified | trx_rows_locked |
+---------------------+---------------------+---------+-----------
+---------------------+-------------------+-----------------+
| 2024-08-12 01:09:39 |                   0 | 2849470 | RUNNING   | NULL
    |                 1 |               0 |
```

```
+--------------------+--------------------+---------+-----------
+--------------------+------------------+-----------------+
1 row in set (0.00 sec)

mysql> xa recover;
+----------+--------------+--------------+--------+
| formatID | gtrid_length | bqual_length | data   |
+----------+--------------+--------------+--------+
|        1 |            6 |            0 | xatest |
+----------+--------------+--------------+--------+
1 row in set (0.00 sec)
```

在這種情況下，我們會復原預備交易。

```
mysql> XA ROLLBACK 'xatest';
Query OK, 0 rows affected (0.00 sec)
v
mysql> xa recover;
Empty set (0.00 sec)
```

在復原 XA 交易之後，預先檢查會成功。

```
{
  "id": "auroraUpgradeCheckForIncompleteXATransactions",
  "title": "Pre-checks for XA Transactions in prepared state.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForInstanceLimit

預先檢查層級：錯誤

檢查目前執行個體類別是否支援升級

目前不支援從寫入器資料庫執行個體類別為 db.r6i.32xlarge 的 Aurora MySQL 2.12.0 或 2.12.1 版執行就地升級。在這種情況下，預先檢查會傳回錯誤。若要允許繼續升級，您可以將資料庫執行個體類別變更為 db.r6i.24xlarge 或更小。或者，您可以升級至 Aurora MySQL 2.12.2 版或更新版本，其中 db.r6i.32xlarge 支援就地升級至 Aurora MySQL 第 3 版。

輸出範例：

```
{
  "id": "auroraUpgradeCheckForInstanceLimit",
  "title": "Checks if upgrade is supported on the current instance class",
  "status": "OK",
  "description": "Upgrade from Aurora Version 2.12.0 and 2.12.1 may fail for 32.xl
 and above instance class.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "all",
        "description": "Upgrade is not supported on this instance size for Aurora
 MySql Version 2.12.1. Before upgrading to Aurora MySql 3, please consider either:
 1. Changing the instance class to 24.xl or lower. -or- 2. Upgrading to patch
 version 2.12.2 or higher."
      }
  ]
},
```

預先檢查會傳回錯誤，因為寫入器資料庫執行個體正在使用 db.r6i.32xlarge 執行個體類別，且正在 Aurora MySQL 2.12.1 版上執行。

auroraUpgradeCheckForInternalUsers

預先檢查層級：錯誤

檢查 8.0 內部使用者

此預先檢查僅適用於 Aurora MySQL 3.03.0 版及更低版本。如果此預先檢查發生錯誤，請升級至 Aurora MySQL 3.04 版或更新版本。

輸出範例：

```
{
  "id": "auroraUpgradeCheckForInternalUsers",
  "title": "Check for 8.0 internal users.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForInvalidUtf8mb3CharacterStringInViews

預先檢查層級：錯誤

檢查檢視定義中是否有無效的 utf8mb3 字元

此預先檢查可識別包含無效 utf8mb3 字元編碼註解的檢視。在 MySQL 8.0 中，更嚴格的驗證會套用至中繼資料中的字元編碼，包括檢視註解。如果任何檢視定義包含 utf8mb3 字元集中無效的字元，升級會失敗。

若要解決此問題，請修改檢視定義以移除或取代任何非 BMP 字元，然後再嘗試升級。

輸出範例：

```
{
"id": "auroraUpgradeCheckForInvalidUtf8mb3CharacterStringInViews",
"title": "Check for invalid utf8mb3 character string.",
"status": "OK",
"description": "Definition of following view(s) has/have invalid utf8mb3 character
 string.",
"detectedProblems": [
        {
            "level": "Error",
            "dbObject": "precheck.utf8mb3_invalid_char_view",
            "description": "Definition of view precheck.utf8mb3_invalid_char_view
 contains an invalid utf8mb3 character string. This is due to https://
bugs.mysql.com/bug.php?id=110177. To fix the inconsistency, we recommend you to
 modify the view definition to not use non-BMP characters and try the upgrade
 again."
        }
    ]
},
```

預先檢查會報告 utf8mb3_invalid_char_view 檢視定義在其定義中包含無效的 utf8mb3 字元。

若要解決此問題，請識別包含不支援字元的檢視，並更新註解。首先，檢查檢視結構並識別註解。

```
MySQL> SHOW CREATE VIEW precheck.utf8mb3_invalid_char_view\G
*************************** 1. row ***************************
                View: utf8mb3_invalid_char_view
         Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`admin`@`%` SQL SECURITY
 DEFINER VIEW `utf8mb3_invalid_char_view` AS select 'This row contains a dolphin #'
 AS `message`
character_set_client: utf8
collation_connection: utf8_general_ci
```

```
1 row in set, 1 warning (0.00 sec)
```

識別包含錯誤的檢視後，請將檢視取代為 CREATE OR REPLACE VIEW 陳述式。

```
MySQL> CREATE OR REPLACE VIEW precheck.utf8mb3_invalid_char_view AS select 'This
 view definition to not use non-BMP characters' AS message;
```

更新包含不支援字元的所有檢視定義後，預先檢查會通過，且升級可以繼續進行。

```
{
"id": "auroraUpgradeCheckForInvalidUtf8mb3ColumnComments",
"title": "Check for invalid utf8mb3 column comments.",
"status": "OK",
"detectedProblems": []
}
```

auroraUpgradeCheckForInvalidUtf8mb3ColumnComments

預先檢查層級：錯誤

檢查 utf8mb3 資料欄註解是否無效

此預先檢查會識別包含含有無效 utf8mb3 字元編碼之資料欄註解的資料表。在 MySQL 8.0 中，更嚴格的驗證會套用至中繼資料中的字元編碼，包括資料欄註解。如果任何資料欄註解包含 utf8mb3 字元集中無效的字元，升級將會失敗。

若要解決此問題，您必須在嘗試升級之前修改資料欄註解，以移除或取代任何非 BMP 字元。您可以使用 ALTER TABLE 陳述式來更新資料欄註解。

輸出範例：

```
{
  "id": "auroraUpgradeCheckForInvalidUtf8mb3ColumnComments",
  "title": "Check for invalid utf8mb3 column comments.",
  "status": "OK",
  "description": "Following table(s) has/have invalid utf8mb3 comments on the
 column/columns.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.t2",
```

```
        "description": "Table test.t2 has invalid utf8mb3 comments in it's column/
columns. This is due to non-BMP characters in the comment field. To fix the
 inconsistency, we recommend you to modify comment fields to not use non-BMP
 characters and try the upgrade again."
      }
  ]
}
```

預先檢查會報告 `test.t2` 資料表在一或多個資料欄註解中包含無效的 `utf8mb3` 字元，特別是因為存在非 BMP 字元。

若要解決此問題，您可以識別有問題的資料欄並更新其註解。首先，檢查資料表結構以識別具有註解的資料欄：

```
mysql> SHOW CREATE TABLE test.t2\G
```

使用有問題的註解識別資料欄後，請使用 `ALTER TABLE` 陳述式更新資料欄。例如：

```
mysql> ALTER TABLE test.t2 MODIFY COLUMN column_name data_type COMMENT 'Updated
 comment without non-BMP characters';
```

或者，您可以完全移除註解：

```
mysql> ALTER TABLE test.t2 MODIFY COLUMN column_name data_type COMMENT '';
```

更新所有有問題的資料欄註解後，預先檢查將會通過，而且升級可以繼續：

```
{
  "id": "auroraUpgradeCheckForInvalidUtf8mb3ColumnComments",
  "title": "Check for invalid utf8mb3 column comments.",
  "status": "OK",
  "detectedProblems": []
}
```

> **Note**
>
> 修改資料欄註解之前，請確定任何依賴這些註解的應用程式程式碼或文件都會隨之更新。如果您的應用程式需要非 BMP 字元，請考慮遷移至 utf8mb4 字元集，以獲得更好的 Unicode 支援。

auroraUpgradeCheckForInvalidUtf8mb3IndexComments

預先檢查層級：錯誤

檢查 utf8mb3 索引註解是否無效

此預先檢查會識別包含索引註解且 utf8mb3 字元編碼無效之資料表。在 MySQL 8.0 中，更嚴格的驗證會套用至中繼資料中的字元編碼，包括索引註解。如果任何索引註解包含字元集中無效的 utf8mb3 字元，升級會失敗。

若要解決此問題，您必須在嘗試升級之前修改索引註解，以移除或取代任何非 BMP 字元。

輸出範例：

```
{
    "id": "auroraUpgradeCheckForInvalidUtf8mb3IndexComments",
    "title": "Check for invalid utf8mb3 index comments.",
    "status": "OK",
    "description": "Following table(s) has/have invalid utf8mb3 comments on the
 index.",
    "detectedProblems": [
        {
            "level": "Error",
            "dbObject": "precheck.utf8mb3_tab_index_comment",
            "description": "Table precheck.utf8mb3_tab_index_comment has invalid
 utf8mb3 comments in it's index. This is due to https://bugs.mysql.com/bug.php?
id=110177. To fix the inconsistency, we recommend you to modify comment fields to
 not use non-BMP characters and try the upgrade again."
        }
    ]
},
```

預先檢查會報告 utf8mb3_tab_index_comment 資料表在一或多個資料欄註解中包含無效的 utf8mb3 字元，特別是因為存在非 BMP 字元。

若要解決此問題，請先檢查資料表結構，以識別包含有問題註解的索引。

```
MySQL> SHOW CREATE TABLE precheck.utf8mb3_tab_index_comment\G
*************************** 1. row ***************************
    Table: utf8mb3_tab_index_comment
Create Table: CREATE TABLE `utf8mb3_tab_index_comment` (
`id` int(11) DEFAULT NULL,
```

```
`name` varchar(100) DEFAULT NULL,
KEY `idx_name` (`name`) COMMENT 'Name index #'
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.01 sec)
```

識別包含使用不支援字元之註解的索引後，請捨棄並重新建立索引。

> **Note**
>
> 捨棄和重新建立資料表索引可能會導致停機時間。建議您在維護期間規劃並排程此操作。

```
MySQL> ALTER TABLE precheck.utf8mb3_tab_index_comment DROP INDEX idx_name;
MySQL> ALTER TABLE precheck.utf8mb3_tab_index_comment ADD INDEX idx_name(name);
```

下列範例顯示重新建立索引的另一種方式。

```
MySQL> ALTER TABLE utf8mb3_tab_index_comment DROP INDEX idx_name, ADD INDEX idx_name
 (name) COMMENT 'Updated comment without non-BMP characters';
```

在您移除或更新所有不支援的索引註解後，預先檢查會通過，且升級可以繼續進行。

```
{
"id": "auroraUpgradeCheckForInvalidUtf8mb3IndexComments",
"title": "Check for invalid utf8mb3 index comments.",
"status": "OK",
"detectedProblems": []
},
```

auroraUpgradeCheckForInvalidUtf8mb3TableComments

預先檢查層級：錯誤

檢查資料表定義中是否有無效的 utf8mb3 字元

此預先檢查可識別包含無效 `utf8mb3` 字元編碼註解的資料表。在 MySQL 8.0 中，更嚴格的驗證會套用至中繼資料中的字元編碼，包括資料表註解。如果任何資料表註解包含字元集中無效的 `utf8mb3` 字元，升級會失敗。

若要解決此問題，您必須在嘗試升級之前修改資料表註解，以移除或取代任何非 BMP 字元。

輸出範例：

```
{
    "id": "auroraUpgradeCheckForInvalidUtf8mb3TableComments",
    "title": "Check for invalid utf8mb3 table comments.",
    "status": "OK",
    "description": "Following table(s) has/have invalid utf8mb3 comments.",
    "detectedProblems": [
        {
            "level": "Error",
            "dbObject": "precheck.utf8mb3_table_with_comment",
            "description": "Table precheck.utf8mb3_table_with_comment has invalid
 utf8mb3 comments. This is due to https://bugs.mysql.com/bug.php?id=110177. To fix
 the inconsistency, we recommend you to modify comment fields to not use non-BMP
 characters and try the upgrade again."
        }

    ]
},
```

預先檢查會報告測試資料庫中 utf8mb3_table_with_comment 資料表定義的無效 utf8mb3 註解。

若要解決此問題，請識別包含不支援字元的資料表，並更新註解。首先，檢查資料表結構並識別註解。

```
MySQL> SHOW CREATE TABLE precheck.utf8mb3_table_with_comment\G
*************************** 1. row ***************************
    Table: utf8mb3_table_with_comment
Create Table: CREATE TABLE `utf8mb3_table_with_comment` (
`id` int(11) NOT NULL,
`name` varchar(100) DEFAULT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='This table comment contains flag ##'
1 row in set (0.00 sec)
```

識別包含不支援聊天程式的資料表註解後，請使用 ALTER TABLE 陳述式更新註解。

```
MySQL> ALTER TABLE precheck.utf8mb3_table_with_comment COMMENT='Updated comment
 without non-BMP characters';
```

或者，您可以移除該註解。

```
MySQL> ALTER TABLE precheck.utf8mb3_table_with_comment COMMENT='';
```

從所有資料表註解中移除所有不支援的字元後，預先檢查會成功。

```
{
"id": "auroraUpgradeCheckForInvalidUtf8mb3TableComments",
"title": "Check for invalid utf8mb3 table comments.",
"status": "OK",
"detectedProblems": []
},
```

auroraUpgradeCheckForMasterUser

預先檢查層級：錯誤

檢查 RDS 主要使用者是否存在

MySQL 8 已新增支援角色和動態權限的新權限模型，讓權限管理更容易且更精細。作為此變更的一部分，Aurora MySQL 推出了新的 `rds_superuser_role`，在從 Aurora MySQL 第 2 版升級至第 3 版時，會自動授予資料庫的主要使用者。

如需 Aurora MySQL 中指派給主要使用者的角色和權限的詳細資訊，請參閱 主要使用者帳戶權限。如需 Aurora MySQL 第 3 版中角色型權限模型的詳細資訊，請參閱 角色型權限模型。

此預先檢查會確認主要使用者是否存在於資料庫中。如果主要使用者不存在，預先檢查會失敗。若要允許繼續升級，請重設主要使用者密碼或手動建立使用者，以重新建立主要使用者。然後重試升級。如需重設主要使用者密碼的詳細資訊，請參閱 變更資料庫主要使用者的密碼。

輸出範例：

```
{
  "id": "auroraUpgradeCheckForMasterUser",
  "title": "Check if master user exists",
  "status": "OK",
  "description": "Throws error if master user has been dropped!",
  "documentationLink": "https://docs.aws.amazon.com/AmazonRDS/latest/
AuroraUserGuide/UsingWithRDS.MasterAccounts.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "all",
```

```
        "description": "Your Master User on host '%' has been dropped. To proceed
 with the upgrade, recreate the master user `reinvent` on default host '%'"
      }
  ]
}
```

重設主要使用者密碼後，預先檢查將會通過，而且您可以重試升級。

下列範例使用 AWS CLI 重設密碼。密碼變更會立即套用。

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --master-user-password my-new-password
```

然後，預先檢查成功。

```
{
  "id": "auroraUpgradeCheckForMasterUser",
  title": "Check if master user exists",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForPrefixIndexOnGeometryColumns

預先檢查層級：錯誤

檢查字首索引上的幾何資料欄

從 MySQL 8.0.12 開始，您無法再使用幾何資料類型在資料欄上建立字首索引。如需詳細資訊，請參閱 WL#11808。

如果有任何此類索引存在，您的升級將會失敗。若要解決此問題，請在預先檢查失敗時提及的資料表上捨棄字首 GEOMETRY 索引。

輸出範例：

```
{
  "id": "auroraUpgradeCheckForPrefixIndexOnGeometryColumns",
  "title": "Check for geometry columns on prefix indexs",
  "status": "OK",
  "description": "Consider dropping the prefix Indexes of geometry columns and
 restart the upgrade.",
```

```
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.geom_index_prefix",
        "description": "Table `test`.`geom_index_prefix` has an index `LatLon` on
 geometry column/s. Mysql 8.0 does not support this type of index on a geometry
 column https://dev.mysql.com/worklog/task/?id=11808. To upgrade to MySQL 8.0, Run
 'DROP INDEX `LatLon` ON `test`.`geom_index_prefix`;"
      }
  ]
}
```

預先檢查會報告錯誤，因為 `test.geom_index_prefix` 資料表包含 GEOMETRY 欄上有字首的索引。

捨棄此索引後，預先檢查會成功。

```
{
  "id": "auroraUpgradeCheckForPrefixIndexOnGeometryColumns",
  "title": "Check for geometry columns on prefix indexs",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForSpecialCharactersInProcedures

預先檢查層級：錯誤

檢查預存程序中與特殊字元相關的不一致

在 MySQL 8.0 之前，資料庫名稱、資料表名稱和其他物件會對應至資料目錄中的檔案，也就是檔案型中繼資料。在升級至 MySQL 8.0 的過程中，所有資料庫物件都會遷移至存放在 `mysql` 結構描述中的新內部資料字典資料表，以支援新實作的單元資料字典。在遷移預存程序的過程中，系統會驗證每個程序的程序定義和本文，因為它會擷取到新的資料字典中。

在 MySQL 8 之前，在某些情況下，可以建立預存常式，或直接插入 `mysql.proc` 資料表中包含特殊字元的程序。例如，您可以建立預存程序，其中包含不合規、非中斷空格字元 `\xa0` 的註解。如果遇到任何此類程序，升級會失敗。

此預先檢查會驗證預存程序的本文和定義不包含任何此類字元。若要允許繼續升級，請重新建立這些預存程序，而不含任何隱藏或特殊字元。

輸出範例：

```
{
  "id": "auroraUpgradeCheckForSpecialCharactersInProcedures",
  "title": "Check for inconsistency related to special characters in stored
 procedures.",
  "status": "OK",
  "description": "Following procedure(s) has special characters inconsistency.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "information_schema.routines",
        "description": "Data Dictionary Metadata is inconsistent for the procedure
 `get_version_proc` in the database `test` due to usage of special characters in
 procedure body. To avoid that, drop and recreate the procedure without any special
 characters before proceeding with the Upgrade."
      }
  ]
}
```

預先檢查會報告資料庫叢集包含 get_version_proc 資料庫中名為 test 的程序，其中包含程序本文中的特殊字元。

捨棄並重新建立預存程序後，預先檢查會成功，讓升級繼續進行。

```
{
  "id": "auroraUpgradeCheckForSpecialCharactersInProcedures",
  "title": "Check for inconsistency related to special characters in stored
 procedures.",
  "status": "OK",
  "detectedProblems": []
},
```

auroraUpgradeCheckForSysSchemaObjectTypeMismatch

預先檢查層級：錯誤

檢查 **sys** 結構描述的物件類型不符

sys 結構描述是 MySQL 資料庫中的一組物件和檢視，可協助使用者故障診斷、最佳化和監控其資料庫執行個體。從 Aurora MySQL 第 2 版升級至第 3 版時，sys 結構描述檢視會重新建立並更新為新的 Aurora MySQL 第 3 版定義。

在升級過程中，如果 sys 結構描述中的任何物件是使用儲存引擎 (INFORMATION_SCHEMA.TABLES 中的sys_config/BASE TABLE) 而非檢視來定義，則升級將會失敗。您可以在 information_schema.tables 資料表中找到此類資料表。這不是預期的行為，但在某些情況下，可能會因使用者錯誤而發生。

此預先檢查會驗證所有 sys 結構描述物件，以確保它們使用正確的資料表定義，且檢視不會錯誤地定義為 InnoDB 或 MyISAM 資料表。若要解決問題，請重新命名或捨棄任何傳回的物件，以手動修正這些物件。然後重試升級。

輸出範例：

```
{
  "id": "auroraUpgradeCheckForSysSchemaObjectTypeMismatch",
  "title": "Check object type mismatch for sys schema.",
  "status": "OK",
  "description": "Database contains objects with type mismatch for sys schema.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "sys.waits_global_by_latency",
        "description": "Your object sys.waits_global_by_latency has a type mismatch.
 To fix the inconsistency we recommend to rename or remove the object before
 upgrading (use RENAME TABLE command). "
      }
  ]
}
```

預先檢查會報告 sys 結構描述中的 sys.waits_global_by_latency 檢視的類型不符，導致無法繼續升級。

登入資料庫執行個體後，您可以看到此物件在應該是檢視時定義為 InnoDB 資料表。

```
mysql> show create table sys.waits_global_by_latency\G
*************************** 1. row ***************************
       Table: waits_global_by_latency
Create Table: CREATE TABLE `waits_global_by_latency` (
  `events` varchar(128) DEFAULT NULL,
  `total` bigint(20) unsigned DEFAULT NULL,
  `total_latency` text,
  `avg_latency` text,
  `max_latency` text
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

```
1 row in set (0.00 sec)
```

若要解決此問題，我們可以捨棄並重新建立具有正確定義的檢視，或將其重新命名。在升級過程中，系統會自動使用正確的資料表定義建立它。

```
mysql> RENAME TABLE sys.waits_global_by_latency to sys.waits_global_by_latency_old;
Query OK, 0 rows affected (0.01 sec)
```

執行此操作後，預先檢查會通過。

```
{
  "id": "auroraUpgradeCheckForSysSchemaObjectTypeMismatch",
  "title": "Check object type mismatch for sys schema.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForViewColumnNameLength

預先檢查層級：錯誤

檢查檢視中資料欄名稱的上限

MySQL 中允許的資料欄名稱長度上限為 64 個字元。在 MySQL 8.0 之前，在某些情況下，可以建立資料欄名稱大於 64 個字元的檢視。如果您的資料庫執行個體上存在任何此類檢視，則會傳回預先檢查錯誤，且升級將會失敗。若要允許繼續升級，您必須重新建立有問題的檢視，確保其資料欄長度小於 64 個字元。然後重試升級。

輸出範例：

```
{
  "id": "auroraUpgradeCheckForViewColumnNameLength",
  "title": "Check for upperbound limit related to column name in view.",
  "status": "OK",
  "description": "Following view(s) has column(s) with length greater than 64.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject":
 "test.colname_view_test.col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pa
        "description": "View `test`.`colname_view_test`has column
 `col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad` with
```

```
 invalid column name length. To avoid Upgrade errors, view should be altered by
 renaming the column name so that its length is not 0 and doesn't exceed 64."
      }
  ]
}
```

預先檢查會報告 test.colname_view_test 檢視包含的資料欄 col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad 超過允許的資料欄長度上限 64 個字元。

查看檢視定義，我們可以看到違規的資料欄。

```
mysql> desc `test`.`colname_view_test`;
+-------------------------------------------------------------------+-------------
+------+-----+---------+-------+
| Field                                                             | Type        |
 Null | Key | Default | Extra |
+-------------------------------------------------------------------+-------------
+------+-----+---------+-------+
| col1                                                              | varchar(20) |
 YES  |     | NULL    |       |
| col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad | int(11)     |
 YES  |     | NULL    |       |
+-------------------------------------------------------------------+-------------
+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

若要允許繼續升級，請重新建立檢視，確保資料欄長度不超過 64 個字元。

```
mysql> drop view `test`.`colname_view_test`;
Query OK, 0 rows affected (0.01 sec)

mysql> create view `test`.`colname_view_test`(col1, col2_nopad) as select inf,
 fodcol from test;
Query OK, 0 rows affected (0.01 sec)

mysql> desc `test`.`colname_view_test`;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| col1       | varchar(20) | YES  |     | NULL    |       |
| col2_nopad | int(11)     | YES  |     | NULL    |       |
```

```
+------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

執行此操作後，預先檢查會成功。

```
{
  "id": "auroraUpgradeCheckForViewColumnNameLength",
  "title": "Check for upperbound limit related to column name in view.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckIndexLengthLimitOnTinyColumns

預先檢查層級：錯誤

檢查在小型資料欄上定義字首長度大於 255 個位元組之索引的資料表

在 MySQL 中使用二進位資料類型在資料欄上建立索引時，您必須在索引定義中新增字首長度。在 MySQL 8.0 之前，在某些情況下，您可以指定大於此類資料類型所允許大小上限的字首長度。例如 TINYTEXT 和 TINYBLOB 欄，其中允許的資料大小上限為 255 個位元組，但允許大於此值的索引字首。如需詳細資訊，請參閱 MySQL 文件中的 InnoDB 限制。

如果此預先檢查失敗，請捨棄違規的索引，或將索引的 TINYTEXT 和 TINYBLOB 欄的字首長度減少到少於 255 個位元組。然後重試升級。

輸出範例：

```
{
  "id": "auroraUpgradeCheckIndexLengthLimitOnTinyColumns",
  "title": "Check for the tables with indexes defined with prefix length greater
 than 255 bytes on tiny columns",
  "status": "OK",
  "description": "Prefix length of the indexes defined on tiny columns cannot
 exceed 255 bytes. With utf8mb4 char set, this limits the prefix length supported
 upto 63 characters only. A larger prefix length was allowed in MySQL5.7 using
 innodb_large_prefix parameter. This parameter is deprecated in MySQL 8.0.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/innodb-limits.html,
 https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html",
  "detectedProblems": [
      {
        "level": "Error",
```

```
        "dbObject": "test.tintxt_prefixed_index.col1",
        "description": "Index 'PRIMARY' on tinytext/tinyblob column `col1` of table
 `test.tintxt_prefixed_index` is defined with prefix length exceeding 255 bytes.
 Reduce the prefix length to <=255 bytes depending on character set used. For
 utf8mb4, it should be <=63."
      }
  ]
}
```

預先檢查會報告 test.tintxt_prefixed_index 資料表的錯誤，因為其索引 PRIMARY 的字首在 TINYTEXT 或 TINYBLOB 資料欄上大於 255 個位元組。

查看此資料表的定義，我們可以看到主索引鍵在 TINYTEXT 資料欄 col1 上有 65 的字首。由於資料表是使用 utf8mb4 字元集來定義，每個字元可存放 4 個位元組，因此字首超過 255 個位元組的限制。

```
mysql> show create table `test`.`tintxt_prefixed_index`\G
*************************** 1. row ***************************
       Table: tintxt_prefixed_index
Create Table: CREATE TABLE `tintxt_prefixed_index` (
  `col1` tinytext NOT NULL,
  `col2` tinytext,
  `col_id` tinytext,
  PRIMARY KEY (`col1`(65))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 ROW_FORMAT=DYNAMIC
1 row in set (0.00 sec)
```

使用 utf8mb4 字元集將索引字首修改為 63 將允許繼續升級。

```
mysql> alter table `test`.`tintxt_prefixed_index` drop PRIMARY KEY, ADD  PRIMARY KEY
 (`col1`(63));
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

執行此操作後，預先檢查會成功。

```
{
  "id": "auroraUpgradeCheckIndexLengthLimitOnTinyColumns",
  "title": "Check for the tables with indexes defined with prefix length greater
 than 255 bytes on tiny columns",
  "status": "OK",
  "detectedProblems": []
```

```
}
```

auroraUpgradeCheckMissingInnodbMetadataForMysqlHostTable

預先檢查層級：錯誤

檢查 **mysql.host** 資料表缺少的 InnoDB 中繼資料不一致

這是 RDS 服務執行的僅限內部預先檢查。升級時會自動處理任何錯誤，並且可以安全地忽略。

如果您在此預先檢查中遇到任何錯誤，請向 AWS Support 開立案例，以請求解決中繼資料不一致的問題。或者，您可以執行邏輯傾印，然後還原至新的 Aurora MySQL 第 3 版資料庫叢集，以重試升級。

警告

下列預先檢查會在預先檢查失敗時產生警告，但升級可以繼續。

主題

- 報告警告的 MySQL 預先檢查
- 報告警告的 Aurora MySQL 預先檢查

報告警告的 MySQL 預先檢查

下列預先檢查來自 Community MySQL：

- defaultAuthenticationPlugin
- maxdbFlagCheck
- mysqlDollarSignNameCheck
- reservedKeywordsCheck
- utf8mb3Check
- zeroDatesCheck

defaultAuthenticationPlugin

預先檢查層級：警告

新的預設身分驗證外掛程式考量

在 MySQL 8.0 中，引進了 caching_sha2_password 身分驗證外掛程式，提供比已棄用 mysql_native_password 外掛程式更安全的密碼加密和更好的效能。對於 Aurora MySQL 第 3 版，用於資料庫使用者的預設身分驗證外掛程式是 mysql_native_password 外掛程式。

此預先檢查會警告將會移除此外掛程式，並在未來的主要版本發行中變更預設值。在此變更之前，請考慮評估應用程式用戶端和使用者的相容性。

如需詳細資訊，請參閱 MySQL 文件中的 caching_sha2_password 相容性問題和解決方案。

輸出範例：

```
{
  "id": "defaultAuthenticationPlugin",
  "title": "New default authentication plugin considerations",
  "description": "Warning: The new default authentication plugin
 'caching_sha2_password' offers more secure password hashing than previously
 used 'mysql_native_password' (and consequent improved client connection
 authentication). However, it also has compatibility implications that
 may affect existing MySQL installations. If your MySQL installation
 must serve pre-8.0 clients and you encounter compatibility issues after
 upgrading, the simplest way to address those issues is to reconfigure
 the server to revert to the previous default authentication plugin
 (mysql_native_password). For example, use these lines in the server option file:
\n\n[mysqld]\ndefault_authentication_plugin=mysql_native_password\n\nHowever, the
 setting should be viewed as temporary, not as a long term or permanent solution,
 because it causes new accounts created with the setting in effect to forego the
 improved authentication security.\nIf you are using replication please take time to
 understand how the authentication plugin changes may impact you.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-caching-sha2-password-compatibility-issues\nhttps://
dev.mysql.com/doc/refman/8.0/en/upgrading-from-previous-series.html#upgrade-caching-
sha2-password-replication"
},
```

maxdbFlagCheck

預先檢查層級：警告

使用過時的 **MAXDB sql_mode** 旗標

在 MySQL 8.0 中，移除了一些已棄用的 sql_mode 系統變數選項，其中一個是 MAXDB。此預先檢查會檢查所有目前連線的工作階段，以及常式和觸發條件，以確保沒有任何 sql_mode 設定為任何包含 MAXDB 的組合。

輸出範例：

```
{
  "id": "maxdbFlagCheck",
  "title": "Usage of obsolete MAXDB sql_mode flag",
  "status": "OK",
  "description": "Warning: The following DB objects have the obsolete MAXDB
 option persisted for sql_mode, which will be cleared during the upgrade. It can
 potentially change the datatype DATETIME into TIMESTAMP if it is used inside
 object's definition, and this in turn can change the behavior in case of dates
 earlier than 1970 or later than 2037. If this is a concern, please redefine these
 objects so that they do not rely on the MAXDB flag before running the upgrade.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/mysql-
nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.maxdb_stored_routine",
        "description": "PROCEDURE uses obsolete MAXDB sql_mode",
        "dbObjectType": "Routine"
      }
  ]
}
```

預先檢查會報告 test.maxdb_stored_routine 常式包含不支援的 sql_mode 選項。

登入資料庫後，您可以在 sql_mode 包含 MAXDB 的常式定義中看到。

```
 > SHOW CREATE PROCEDURE test.maxdb_stored_routine\G

*************************** 1. row ***************************
           Procedure: maxdb_stored_routine
            sql_mode:
 PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,MAXDB,NO_KEY_OPTIONS,NO_TABLE_OPTIONS,NO_FIELD_OPT
    Create Procedure: CREATE DEFINER="msandbox"@"localhost" PROCEDURE
 "maxdb_stored_routine"()
BEGIN
    SELECT * FROM test;
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

若要解決此問題，請在用戶端上設定正確的 sql_mode 之後重新建立程序。

> **Note**
>
> 根據 MySQL 文件，MySQL 會存放建立或修改常式時生效的 sql_mode 設定。無論常式啟動時的 sql_mode 設定為何，它一律會使用此設定執行常式。
> 在變更 sql_mode 之前，請參閱 MySQL 文件中的伺服器 SQL 模式。仔細評估對應用程式執行此操作的任何潛在影響。

在沒有不支援的 sql_mode 的情況下重新建立程序。

```
mysql > set sql_mode='PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql > DROP PROCEDURE test.maxdb_stored_routine\G
Query OK, 0 rows affected (0.00 sec)

mysql >
mysql > DELIMITER $$
mysql >
mysql > CREATE PROCEDURE test.maxdb_stored_routine()
    -> SQL SECURITY DEFINER
    -> BEGIN
    ->     SELECT * FROM test;
    -> END$$
Query OK, 0 rows affected (0.00 sec)

mysql >
mysql > DELIMITER ;
mysql > show create procedure test.maxdb_stored_routine\G
*************************** 1. row ***************************
           Procedure: maxdb_stored_routine
            sql_mode: PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE
    Create Procedure: CREATE DEFINER="msandbox"@"localhost" PROCEDURE
 "maxdb_stored_routine"()
BEGIN
    SELECT * FROM test;
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

```
1 row in set (0.00 sec)
```

預先檢查會成功。

```
{
  "id": "maxdbFlagCheck",
  "title": "Usage of obsolete MAXDB sql_mode flag",
  "status": "OK",
  "detectedProblems": []
}
```

mysqlDollarSignNameCheck

預先檢查層級：警告

檢查物件名稱中是否已棄用單一美元符號

從 MySQL 8.0.32 開始，使用美元符號 ($) 作為未引用識別符的第一個字元已棄用。如果您有包含 $ 作為第一個字元的任何結構描述、資料表、檢視、資料欄或常式，此預先檢查會傳回警告。雖然此警告不會阻止升級繼續進行，但我們建議您盡快採取動作來解決此問題。從 MySQL 8.4，任何此類識別符都會傳回語法錯誤，而不是警告。

輸出範例：

```
{
  "id": "mysqlDollarSignNameCheck",
  "title": "Check for deprecated usage of single dollar signs in object names",
  "status": "OK",
  "description": "Warning: The following objects have names with deprecated usage
 of dollar sign ($) at the begining of the identifier. To correct this warning,
 ensure, that names starting with dollar sign, also end with it, similary to quotes
 ($example$). ",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.$deprecated_syntx",
        "description": " name starts with $ sign."
      }
  ]
},
```

預先檢查會報告警告，因為 `test` 結構描述中的 `$deprecated_syntx` 資料表包含 `$` 作為第一個字元。

reservedKeywordsCheck

預先檢查層級：警告

使用名稱與新預留關鍵字衝突的資料庫物件

此檢查類似於 routineSyntaxCheck，因為它會檢查名稱與新預留關鍵字衝突的資料庫物件使用情況。雖然它們不會封鎖升級，但您需要仔細評估警告。

輸出範例：

使用上一個範例搭配名為 `except` 的資料表，預先檢查會傳回警告：

```
{
  "id": "reservedKeywordsCheck",
  "title": "Usage of db objects with names conflicting with new reserved keywords",
  "status": "OK",
  "description": "Warning: The following objects have names that conflict with
 new reserved keywords. Ensure queries sent by your applications use `quotes` when
 referring to them or they will result in errors.",
  "documentationLink": "https://dev.mysql.com/doc/refman/en/keywords.html",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.except",
        "description": "Table name",
        "dbObjectType": "Table"
      }
  ]
}
```

此警告可讓您知道可能有一些應用程式查詢需要檢閱。如果您的應用程式查詢未正確逸出字串常值，則升級到 MySQL 8.0 後可能會發生錯誤。檢閱您的應用程式，以確認、測試第 3 版上執行之 Aurora MySQL 資料庫叢集的複製或快照。

升級後將失敗的非逸出應用程式查詢範例：

```
SELECT * FROM escape;
```

升級後將成功的正確逸出應用程式查詢範例：

```
SELECT * FROM 'escape';
```

utf8mb3Check

預先檢查層級：警告

utf8mb3 字元集的使用

在 MySQL 8.0 中，utf8mb3 字元集已棄用，並將在未來 MySQL 主要版本中移除。如果偵測到使用此字元集的任何資料庫物件，則會實作此預先檢查以引發警告。雖然這不會阻止升級繼續進行，但強烈建議您考慮將資料表遷移到 utf8mb4 字元集，這是 MySQL 8.0 的預設值。如需 utf8mb3 和 utf8mb4 的詳細資訊，請參閱 MySQL 文件中的在 3 位元組和 4 位元組 Unicode 字元集之間轉換。

輸出範例：

```
{
  "id": "utf8mb3",
  "title": "Usage of utf8mb3 charset",
  "status": "OK",
  "description": "Warning: The following objects use the deprecated utf8mb3
 character set. It is recommended to convert them to use utf8mb4 instead, for
 improved Unicode support. The utf8mb3 character is subject to removal in the
 future.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/charset-unicode-
utf8mb3.html",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.t1.col1",
        "description": "column's default character set: utf8",
        "dbObjectType": "Column"
      },
      {
        "level": "Warning",
        "dbObject": "test.t1.col2",
        "description": "column's default character set: utf8",
        "dbObjectType": "Column"
      }
  ]
}
```

若要解決此問題，請重建參考的物件和資料表。如需詳細資訊和先決條件，請參閱 MySQL 文件中的在 3 位元組和 4 位元組 Unicode 字元集之間轉換。

zeroDatesCheck

預先檢查層級：警告

零日期、日期時間和時間戳記值

MySQL 現在會對日期、日期時間和時間戳記資料欄使用零值強制執行更嚴格的規則。我們建議您使用 NO_ZERO_IN_DATE 和 NO_ZERO_DATE SQL 模式搭配 strict 模式，因為它們將在未來 MySQL 版本中與 strict 模式合併。

如果執行預先檢查時任何資料庫連線的 sql_mode 設定不包含這些模式，則預先檢查中會引發警告。使用者仍然可以插入包含零值的日期、日期時間和時間戳記值。不過，我們強烈建議將任何零值取代為有效值，因為其行為可能會在未來變更，而且可能無法正常運作。由於這是警告，因此不會封鎖升級，但建議您開始規劃以採取行動。

輸出範例：

```
{
  "id": "zeroDatesCheck",
  "title": "Zero Date, Datetime, and Timestamp values",
  "status": "OK",
  "description": "Warning: By default zero date/datetime/timestamp values are no
 longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are included
 in SQL_MODE by default. These modes should be used with strict mode as they will be
 merged with strict mode in a future release. If you do not include these modes in
 your SQL_MODE setting, you are able to insert date/datetime/timestamp values that
 contain zeros. It is strongly advised to replace zero values with valid ones, as
 they may not work correctly in the future.",
  "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "global.sql_mode",
        "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
 which allows insertion of zero dates"
      },
      {
        "level": "Warning",
        "dbObject": "session.sql_mode",
        "description": " of 10 session(s) does not contain either NO_ZERO_DATE or
 NO_ZERO_IN_DATE which allows insertion of zero dates"
```

```
      }
  ]
}
```

報告警告的 Aurora MySQL 預先檢查

下列預先檢查專屬於 Aurora MySQL：

- auroraUpgradeCheckForRollbackSegmentHistoryLength
- auroraUpgradeCheckForUncommittedRowModifications

auroraUpgradeCheckForRollbackSegmentHistoryLength

預先檢查層級：警告

檢查叢集的復原區段歷史記錄清單長度是否很高

如 auroraUpgradeCheckForIncompleteXATransactions 中所述，在執行主要版本升級程序時，Aurora MySQL 第 2 版資料庫執行個體必須經過正常關閉。這可確保所有交易都已遞交或復原，而且 InnoDB 已清除所有復原日誌記錄。

如果您的資料庫叢集具有高復原區段歷史記錄清單長度 (HLL)，可能會延長 InnoDB 完成復原日誌記錄清除所需的時間，導致主要版本升級程序期間延長停機時間。如果預先檢查偵測到資料庫叢集上的 HLL 很高，就會引發警告。雖然這不會阻止您的升級繼續進行，但我們建議您密切監控資料庫叢集上的 HLL。將其保持在低水準可減少主要版本升級期間所需的停機時間。如需監控 HLL 的詳細資訊，請參閱 InnoDB 歷史記錄清單長度顯著增加。

輸出範例：

```
{
  "id": "auroraUpgradeCheckForRollbackSegmentHistoryLength",
  "title": "Checks if the rollback segment history length for the cluster is high",
  "status": "OK",
  "description": "Rollback Segment History length is greater than 1M. Upgrade may
 take longer time.",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "information_schema.innodb_metrics",
```

```
        "description": "The InnoDB undo history list length('trx_rseg_history_len')
 is 82989114. Upgrade may take longer due to purging of undo information for old row
 versions."
      }
  ]
}
```

預先檢查會傳回警告，因為它偵測到資料庫叢集 (82989114) 上的 InnoDB 復原 HLL 很高。雖然升級繼續進行，但根據要清除的復原數量，它可能會延長升級程序所需的停機時間。

我們建議您在生產環境中執行升級之前，先調查資料庫叢集上的開啟中交易，以確保您的 HLL 保持在可管理的大小。

auroraUpgradeCheckForUncommittedRowModifications

預先檢查層級：警告

檢查是否有許多未遞交的資料列修改

如 auroraUpgradeCheckForIncompleteXATransactions 中所述，在執行主要版本升級程序時，Aurora MySQL 第 2 版資料庫執行個體必須經過正常關閉。這可確保所有交易都已遞交或復原，而且 InnoDB 已清除所有復原日誌記錄。

如果您的資料庫叢集有已修改大量資料列的交易，則可能會延長 InnoDB 在清除關機程序中完成此交易復原所需的時間。如果預先檢查找到長時間執行的交易，且資料庫叢集上有大量修改的資料列，則會引發警告。雖然這不會阻止您的升級繼續進行，但我們建議您密切監控資料庫叢集上作用中交易的大小。將其保持在低水準可減少主要版本升級期間所需的停機時間。

輸出範例：

```
{
  "id": "auroraUpgradeCheckForUncommittedRowModifications",
  "title": "Checks if there are many uncommitted modifications to rows",
  "status": "OK",
  "description": "Database contains uncommitted row changes greater than 10M.
 Upgrade may take longer time.",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "information_schema.innodb_trx",
        "description": "The database contains 11000000 uncommitted row change(s) in
 1 transaction(s). Upgrade may take longer due to transaction rollback."
      }
```

```
  ]
},
```

預先檢查會報告資料庫叢集包含具有 11,000,000 個未遞交資料列變更的交易，這些變更需要在正常關閉程序期間復原。升級會繼續進行，但為了減少升級程序期間的停機時間，建議您在生產叢集上執行升級之前監控並調查此問題。

若要檢視寫入器資料庫執行個體上的作用中交易，您可以使用 information_schema.innodb_trx 資料表。下列寫入器資料庫執行個體的查詢會顯示資料庫叢集目前的交易、執行時間、狀態和修改的資料列。

```
# Example of uncommitted transaction
mysql> SELECT trx_started,
       TIME_TO_SEC(TIMEDIFF(now(), trx_started)) AS seconds_trx_has_been_running,
       trx_mysql_thread_id AS show_processlist_connection_id,
       trx_id,
       trx_state,
       trx_rows_modified AS rows_modified
FROM information_schema.innodb_trx;
+---------------------+-----------------------------
+---------------------------------+----------+-----------+---------------+
| trx_started         | seconds_trx_has_been_running |
 show_processlist_connection_id | trx_id   | trx_state | rows_modified |
+---------------------+-----------------------------
+---------------------------------+----------+-----------+---------------+
| 2024-08-12 18:32:52 |                          1592 |
 20041 | 52866130 | RUNNING   |      11000000 |
+---------------------+-----------------------------
+---------------------------------+----------+-----------+---------------+
1 row in set (0.01 sec)

# Example of transaction rolling back
mysql> SELECT trx_started,
       TIME_TO_SEC(TIMEDIFF(now(), trx_started)) AS seconds_trx_has_been_running,
       trx_mysql_thread_id AS show_processlist_connection_id,
       trx_id,
       trx_state,
       trx_rows_modified AS rows_modified
FROM information_schema.innodb_trx;
+---------------------+-----------------------------
+---------------------------------+----------+--------------+---------------+
| trx_started         | seconds_trx_has_been_running |
 show_processlist_connection_id | trx_id   | trx_state    | rows_modified |
```

```
+---------------------+------------------------------
+---------------------------------+----------+--------------+---------------+
| 2024-08-12 18:32:52 |                          1719 |
 20041 | 52866130 | ROLLING BACK |      10680479 |
+---------------------+------------------------------
+---------------------------------+----------+--------------+---------------+
1 row in set (0.01 sec)
```

在遞交或復原交易之後，預先檢查不會再傳回警告。在復原任何大型交易之前，請參閱 MySQL 文件和您的應用程式團隊，因為復原可能需要一些時間才能完成，具體取決於交易大小。

```
{
  "id": "auroraUpgradeCheckForUncommittedRowModifications",
  "title": "Checks if there are many uncommitted modifications to rows",
  "status": "OK",
  "detectedProblems": []
},
```

如需有關最佳化 InnoDB 交易管理，以及在 MySQL 資料庫執行個體上執行和復原大型交易的潛在影響的詳細資訊，請參閱 MySQL 文件中的最佳化 InnoDB 交易管理。

注意

下列預先檢查會在預先檢查失敗時產生通知，但升級可以繼續。

sqlModeFlagCheck

預先檢查層級：注意

使用過時的 **sql_mode** 旗標

除了 MAXDB 之外，還移除了許多其他 sql_mode 選項：DB2、MSSQL、MYSQL323、MYSQL40ORACLE、POSTGRESQL、NO_FIELD_OPTIONS、NO_KEY_OP
和 NO_TABLE_OPTIONS。從 MySQL 8.0 開始，這些值都無法指派給 sql_mode 系統變數。如果此預先檢查使用這些 sql_mode 設定尋找任何開啟中工作階段，請確定已更新您的資料庫執行個體和資料庫叢集參數群組，以及用戶端應用程式和組態以停用它們。- 如需詳細資訊，請參閱 MySQL 文件。

輸出範例：

```
{
```

```
  "id": "sqlModeFlagCheck",
  "title": "Usage of obsolete sql_mode flags",
  "status": "OK",
  "detectedProblems": []
}
```

若要解決任何這些預先檢查失敗，請參閱 maxdbFlagCheck。

錯誤、警告或通知

視預先檢查輸出而定，下列預先檢查可能會傳回錯誤、警告或通知。

checkTableOutput

預先檢查層級：錯誤、警告或通知

`check table x for upgrade` 命令報告的問題

開始升級至 Aurora MySQL 第 3 版之前，`check table for upgrade` 會在資料庫叢集的使用者結構描述中的每個資料表上執行。此預先檢查與 checkTableMysqlSchema 不同。

`check table for upgrade` 命令會檢查資料表是否有任何在升級至較新版本的 MySQL 期間可能發生的潛在問題。在嘗試升級之前執行此命令有助於事先識別和解決任何不相容，使實際的升級程序更順暢。

此命令會對每個資料表執行各種檢查，如下所示：

- 確認資料表結構和中繼資料是否與目標 MySQL 版本相容
- 檢查資料表使用的任何已棄用或已移除的功能
- 確保資料表可以正確升級，而不會遺失資料

與其他預先檢查不同，它可能會根據 `check table` 輸出傳回錯誤、警告或通知。如果此預先檢查傳回任何資料表，請在起始升級之前仔細檢閱它們，以及傳回程式碼和訊息。如需詳細資訊，請參閱 MySQL 文件中的 CHECK TABLE 陳述式。

我們在這裡提供錯誤範例和警告範例。

錯誤範例：

```
{
  "id": "checkTableOutput",
  "title": "Issues reported by 'check table x for upgrade' command",
```

```
  "status": "OK",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.parent",
        "description": "Table 'test.parent' doesn't exist"
      }
  ]
},
```

預先檢查會報告 test.parent 資料表不存在的錯誤。

寫入器資料庫執行個體的 mysql-error.log 檔案顯示有外部索引鍵錯誤。

```
2024-08-13T15:32:10.676893Z 62 [Warning] InnoDB: Load table `test`.`parent` failed,
 the table has missing foreign key indexes. Turn off 'foreign_key_checks' and try
 again.
2024-08-13T15:32:10.676905Z 62 [Warning] InnoDB: Cannot open table test/parent from
 the internal data dictionary of InnoDB though the .frm file for the table exists.
Please refer to http://dev.mysql.com/doc/refman/5.7/en/innodb-troubleshooting.html
 for how to resolve the issue.
```

登入寫入器資料庫執行個體並執行 show engine innodb status\G 以取得外部索引鍵錯誤的詳細資訊。

```
mysql> show engine innodb status\G
*************************** 1. row ***************************
  Type: InnoDB
  Name:
Status:
=====================================
2024-08-13 15:33:33 0x14ef7b8a1700 INNODB MONITOR OUTPUT
=====================================
.
.
.
------------------------
LATEST FOREIGN KEY ERROR
------------------------
2024-08-13 15:32:10 0x14ef6dbbb700 Error in foreign key constraint of table test/
child:
there is no index in referenced table which would contain
the columns as the first columns, or the data types in the
```

```
referenced table do not match the ones in table. Constraint:
,
  CONSTRAINT `fk_pname` FOREIGN KEY (`p_name`) REFERENCES `parent` (`name`)
The index in the foreign key in table is p_name_idx
Please refer to http://dev.mysql.com/doc/refman/5.7/en/innodb-foreign-key-
constraints.html for correct foreign key definition.
.
.
```

LATEST FOREIGN KEY ERROR 訊息報告 test.child 資料表中參考 test.parent 資料表的 fk_pname 外部索引鍵限制條件缺少索引或資料類型不相符。有關外部索引鍵限制條件的 MySQL 文件指出外部索引鍵中參考的資料欄必須具有相關聯的索引，且父/子資料欄必須使用相同的資料類型。

若要確認這是否與缺少索引或資料類型不符有關，請登入資料庫，並暫時停用工作階段變數 foreign_key_checks 來檢查資料表定義。這麼做之後，我們可以看到有問題的子限制條件 (fk_pname) 使用 p_name varchar(20) CHARACTER SET latin1 DEFAULT NULL 來參考父資料表 name varchar(20) NOT NULL。父資料表使用 DEFAULT CHARSET=utf8，但子資料表的 p_name 資料欄使用 latin1，因此擲回資料類型不相符錯誤。

```
mysql> show create table parent\G
ERROR 1146 (42S02): Table 'test.parent' doesn't exist

mysql> show create table child\G
*************************** 1. row ***************************
       Table: child
Create Table: CREATE TABLE `child` (
  `id` int(11) NOT NULL,
  `p_name` varchar(20) CHARACTER SET latin1 DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `p_name_idx` (`p_name`),
  CONSTRAINT `fk_pname` FOREIGN KEY (`p_name`) REFERENCES `parent` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

mysql> set foreign_key_checks=0;
Query OK, 0 rows affected (0.00 sec)

mysql> show create table parent\G
*************************** 1. row ***************************
       Table: parent
Create Table: CREATE TABLE `parent` (
```

```
  `name` varchar(20) NOT NULL,
  PRIMARY KEY (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

mysql> show create table child\G
*************************** 1. row ***************************
       Table: child
Create Table: CREATE TABLE `child` (
  `id` int(11) NOT NULL,
  `p_name` varchar(20) CHARACTER SET latin1 DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `p_name_idx` (`p_name`),
  CONSTRAINT `fk_pname` FOREIGN KEY (`p_name`) REFERENCES `parent` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

若要解決此問題，我們可以將子資料表變更為使用與父系相同的字元集，或將父資料表變更為使用與子資料表相同的字元集。在這裡，由於子資料表在 p_name 資料欄定義中明確使用 latin1，因此我們執行 ALTER TABLE 將字元集修改為 utf8。

```
mysql> alter table child modify p_name varchar(20) character set utf8 DEFAULT NULL;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> flush tables;
Query OK, 0 rows affected (0.01 sec)
```

執行此操作後，預先檢查會通過，而且升級可以繼續。

警告範例：

```
{
  "id": "checkTableOutput",
  "title": "Issues reported by 'check table x for upgrade' command",
  "status": "OK",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.orders",
        "description": "Trigger test.orders.delete_audit_trigg does not have CREATED
 attribute."
```

```
      }
  ]
}
```

預先檢查會報告 test.orders 資料表上 delete_audit_trigg 觸發條件的警告，因為它沒有 CREATED 屬性。根據 MySQL 文件中的檢查版本相容性，此訊息是資訊性的，並針對在 MySQL 5.7.2 之前建立的觸發條件進行列印。

由於這是警告，因此不會封鎖升級繼續進行。不過，如果您想要解決問題，您可以重新建立有問題的觸發條件，而預先檢查會成功且沒有警告。

```
{
  "id": "checkTableOutput",
  "title": "Issues reported by 'check table x for upgrade' command",
  "status": "OK",
  "detectedProblems": []
},
```

就地升級執行方式

建議您檢閱Aurora MySQL 主要版本就地升級的運作方式中的背景資料。

執行任何升級前規劃和測試，如 規劃 Aurora MySQL 叢集的主要版本升級 中所述。

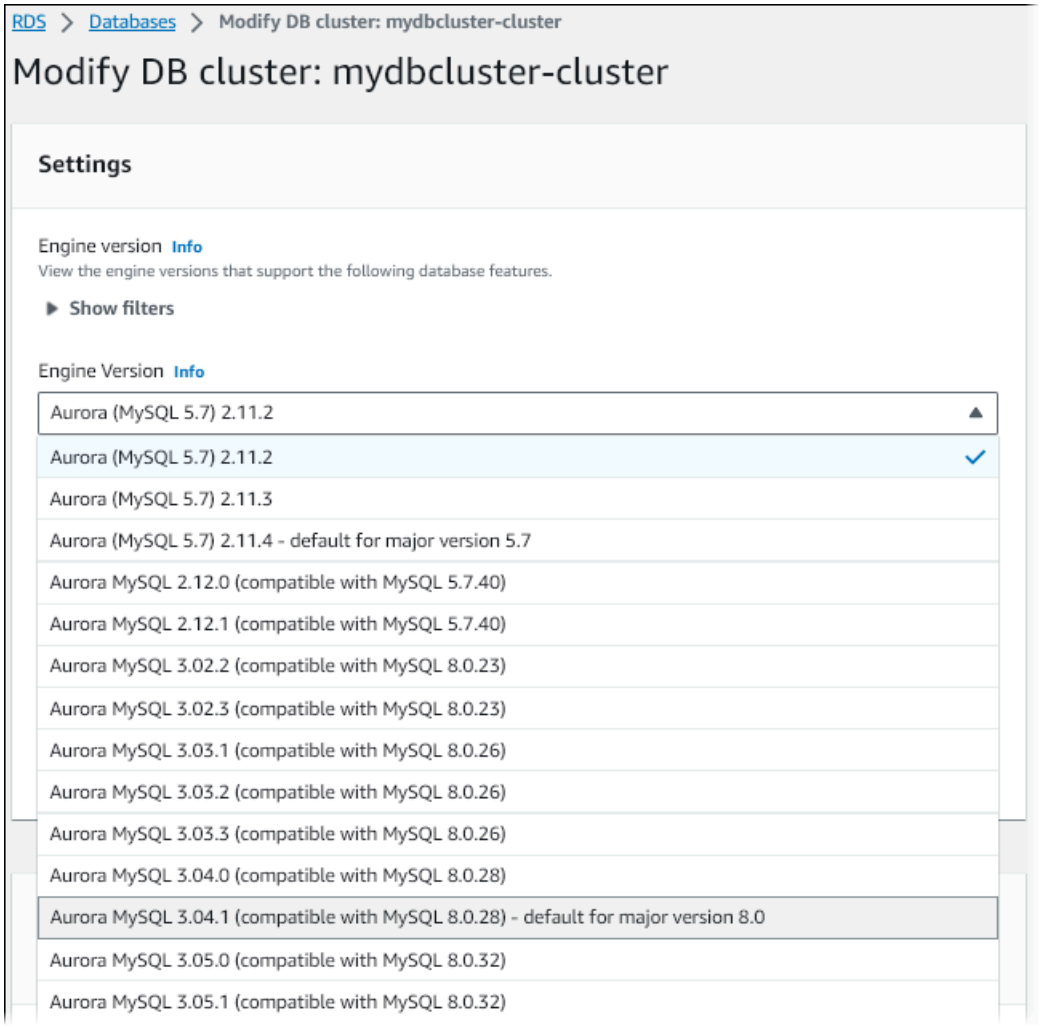
主控台

下列範例會將 `mydbcluster-cluster` 資料庫叢集升級至 Aurora MySQL 3.04.1 版。

升級 Aurora MySQL 資料庫叢集的主要版本

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 如果您已對原始資料庫叢集使用自訂參數群組，請建立與新主要版本相容的對應參數群組。對該新參數群組中的組態參數進行任何必要的調整。如需更多詳細資訊，請參閱 就地升級如何影響叢集的參數群組。
3. 在導覽窗格中，選擇 Databases (資料庫)。
4. 在清單中，選擇您要修改的資料庫叢集。
5. 選擇 Modify (修改)。
6. 對於 Version (版本)，請選擇新的 Aurora MySQL 主要版本。

   我們通常建議使用主要版本的最新次要版本。在這裡，我們選擇目前的預設版本。



7. 選擇 Continue (繼續)。
8. 在下一頁中，指定執行升級的時間。選擇 During the next scheduled maintenance window (在下一個排程的維護期間) 或 Immediately (立即)。
9. (選用) 在升級期間，定期檢查 RDS 主控台中的 Events (事件) 頁面。這樣做可協助您監控升級的進度，並找出任何問題。如果升級發生任何問題，請參閱 Aurora MySQL 就地升級疑難排解 以了解要採取的步驟。
10. 如果您在此程序開始時建立了新的參數群組，請將自訂參數群組與升級的叢集建立關聯。如需更多詳細資訊，請參閱 就地升級如何影響叢集的參數群組。

> **Note**
>
> 執行此步驟需要再次啟動叢集，才能套用新的參數群組。

11. (選用) 完成任何升級後測試後，請刪除升級開始時 Aurora 建立的手動快照。

AWS CLI

若要升級 Aurora MySQL 資料庫叢集的主要版本，請使用具有下列必要參數的 AWS CLI modify-db-cluster 命令：

- `--db-cluster-identifier`
- `--engine-version`
- `--allow-major-version-upgrade`
- `--apply-immediately` 或 `--no-apply-immediately` *

如果您的叢集使用任何自訂參數群組，也請包含下列其中一個或兩個選項：

- `--db-cluster-parameter-group-name` (如果叢集使用自訂叢集參數群組)
- `--db-instance-parameter-group-name` (如果叢集中的任何執行個體使用自訂資料庫參數群組)

下列範例會將 `sample-cluster` 資料庫叢集升級至 Aurora MySQL 3.04.1 版。升級會立即進行，而不是等待下一個維護時段。

Example

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
          --db-cluster-identifier sample-cluster \
          --engine-version 8.0.mysql_aurora.3.04.1 \
          --allow-major-version-upgrade \
          --apply-immediately
```

在 Windows 中：

```
aws rds modify-db-cluster ^
          --db-cluster-identifier sample-cluster ^
          --engine-version 8.0.mysql_aurora.3.04.1 ^
          --allow-major-version-upgrade ^
          --apply-immediately
```

您可以將其他 CLI 命令與 `modify-db-cluster` 合併，建立自動化端對端程序，以執行和驗證升級。如需詳細資訊和範例，請參閱 Aurora MySQL 會就地升級教學。

> **Note**
>
> 如果您的叢集是 Aurora 全域資料庫的一部分，就地升級程序會略有差異。呼叫 modify-global-cluster 命令操作，而不是 `modify-db-cluster`。如需更多詳細資訊，請參閱 全域資料庫的就地主要升級。

RDS API

若要升級 Aurora MySQL 資料庫叢集的主要版本，請搭配下列必要參數使用 RDS API 操作 ModifyDBCluster：

- `DBClusterIdentifier`
- `Engine`
- `EngineVersion`
- `AllowMajorVersionUpgrade`
- `ApplyImmediately` (設定為 `true` 或 `false`)

> **Note**
>
> 如果您的叢集是 Aurora 全域資料庫的一部分，就地升級程序會略有差異。呼叫 ModifyGlobalCluster 操作，而不是 `ModifyDBCluster`。如需更多詳細資訊，請參閱 全域資料庫的就地主要升級。

就地升級如何影響叢集的參數群組

對於與 MySQL 5.7 或 8.0 相容的叢集，Aurora 參數群組具有不同的組態設定集。當您執行就地升級時，升級的叢集及其所有執行個體必須使用對應的叢集和執行個體參數群組。

您的叢集和執行個體可能會使用預設的 5.7 相容參數群組。若是如此，升級的叢集和執行個體會以預設的 8.0 相容參數群組開始。如果您的叢集和執行個體使用任何自訂參數群組，請務必建立對應的 8.0 相容參數群組。此外，亦請務必在升級過程中指定這些項目。

> **Note**
>
> 對於大部分參數設定，您可以在兩點選擇自訂參數群組。這些是您建立叢集或在將參數群組與叢集建立關聯時。

不過，如果您對 lower_case_table_names 參數使用非預設設定，則必須預先使用此設定來設定自訂參數群組。然後，在執行快照還原來建立叢集時指定參數群組。在建立叢集之後，對 lower_case_table_names 參數的任何變更都無效。
當您從 Aurora MySQL 第 2 版升級到第 3 版時，我們建議您針對 lower_case_table_names 使用相同的設定。
使用以 Aurora MySQL 為基礎的 Aurora 全球資料庫時，只有將 lower_case_table_names 參數設為預設值並重新啟動全球資料庫，才能從 Aurora MySQL 第 2 版就地升級至第 3 版。如需詳細了解您可以使用的方法，請參閱 主要版本升級。

在 Aurora MySQL 版本之間對叢集屬性的變更

當您從 Aurora MySQL 第 2 版升級至第 3 版時，請務必確認用於設定或管理 Aurora MySQL 叢集和資料庫執行個體的任何應用程式或指令碼。

此外，變更操作參數群組的程式碼，以考慮預設參數群組名稱對於 5.7 和 8.0 相容叢集不同的事實。Aurora MySQL 第 2 版和第 3 版叢集的預設參數群組名稱分別為 default.aurora-mysql5.7 和 default.aurora-mysql8.0。

例如，在升級之前，可能會有如下所示的程式碼套用至叢集。

```
# Check the default parameter values for MySQL 5.7-compatible clusters.
aws rds describe-db-parameters --db-parameter-group-name default.aurora-mysql5.7 --
region us-east-1
```

升級叢集的主要版本後，如下所示修改該程式碼。

```
# Check the default parameter values for MySQL 8.0-compatible clusters.
aws rds describe-db-parameters --db-parameter-group-name default.aurora-mysql8.0 --
region us-east-1
```

全域資料庫的就地主要升級

若為 Aurora 全域資料庫，您可以升級全域資料庫叢集。Aurora 會自動同時升級所有叢集，並確保其皆執行相同的引擎版本。這項要求是因為對系統資料表、資料檔案格式等所做的任何變更都會自動複寫至所有次要叢集。

請遵循中的說明進行Aurora MySQL 主要版本就地升級的運作方式 在您指定要升級的內容時，請確認已選擇全域資料庫叢集，而非其中包含的其中一個叢集。

若您使用 AWS 管理主控台，請選擇具有 Global database (全域資料庫) 角色的項目。

DB identifier	Role	Engine	Engine version
global-cluster	Global database	Aurora MySQL	5.7.mysql_aurora.2.09.2
cluster1	Primary cluster	Aurora MySQL	5.7.mysql_aurora.2.09.2
dbinstance-1	Writer instance	Aurora MySQL	5.7.mysql_aurora.2.09.2
cluster-2	Secondary cluster	Aurora MySQL	5.7.mysql_aurora.2.09.2
dbinstance-2	Reader instance	Aurora MySQL	5.7.mysql_aurora.2.09.2

如果您使用 AWS CLI 或 RDS API，請呼叫 modify-global-cluster 命令或 ModifyGlobalCluster 操作來啟動升級程序。您會使用其中一個，而不是 `modify-db-cluster` 或 `ModifyDBCluster`。

> **Note**
>
> 在執行 Aurora 全球資料庫的主要版本升級時，無法為全域資料庫叢集指定自訂參數群組。在全域叢集的每個區域中建立您的自訂參數群組。然後，在升級後手動將其套用至區域叢集。

若要透過使用 AWS CLI 來升級 Aurora MySQL 全球資料庫叢集的主要版本，請使用具有下列必要參數的 modify-global-cluster 命令：

- `--global-cluster-identifier`
- `--engine aurora-mysql`
- `--engine-version`
- `--allow-major-version-upgrade`

下列範例會將全球資料庫叢集升級至 Aurora MySQL 3.04.2 版。

Example

對於 Linux、macOS 或 Unix：

```
aws rds modify-global-cluster \
          --global-cluster-identifier global_cluster_identifier \
          --engine aurora-mysql \
          --engine-version 8.0.mysql_aurora.3.04.2 \
          --allow-major-version-upgrade
```

在 Windows 中：

```
aws rds modify-global-cluster ^
          --global-cluster-identifier global_cluster_identifier ^
          --engine aurora-mysql ^
          --engine-version 8.0.mysql_aurora.3.04.2 ^
          --allow-major-version-upgrade
```

使用跨區域僅供讀取複本進行資料庫叢集就地升級

您可以使用就地升級程序升級具有跨區域僅供讀取複本的 Aurora 資料庫叢集，但有一些考量：

- 您必須先升級僅供讀取複本資料庫叢集。如果您嘗試先升級主要叢集，您會收到錯誤訊息，如下所示：

  無法升級資料庫叢集 test-xr-primary-cluster，因為關聯的 Aurora 跨區域複本 test-xr-replica-cluster 尚未修補。請升級 Aurora 跨區域複本，然後再試一次。

  這表示主要資料庫叢集的資料庫引擎版本不能高於複本叢集。
- 升級主要資料庫叢集之前，請先停止寫入工作負載，並停用對主要叢集寫入器資料庫執行個體的任何新連線請求。
- 當您升級主要叢集時，請選擇自訂資料庫叢集參數群組，並將 `binlog_format` 參數設定為支援二進位記錄複寫的值，例如 `MIXED`。

  如需搭配 Aurora MySQL 使用二進位記錄的詳細資訊，請參閱 Aurora 與 MySQL 之間或 Aurora 與另一個 Aurora 資料庫叢集之間的複寫 (二進位複寫)。如需修改 Aurora MySQL 組態參數的詳細資訊，請參閱 Aurora MySQL 組態參數 和 Amazon Aurora 的參數群組。
- 升級複本叢集之後，請勿等待很長的時間才升級主要資料庫叢集。我們建議不要等待超過下一個維護時段。
- 升級主要資料庫叢集之後，請重新啟動其寫入器資料庫執行個體。在寫入器資料庫執行個體重新啟動之前，啟用 binlog 複寫的自訂資料庫叢集參數群組不會生效。
- 在您確認跨區域複寫已重新啟動，且次要 AWS 區域 中的複本延遲為 0 之前，請勿繼續寫入工作負載或啟用寫入器資料庫執行個體的連線。

Aurora MySQL 會就地升級教學

下列 Linux 範例顯示如何使用 AWS CLI 來執行就地升級程序的一般步驟。

這第一個範例會建立執行 2.x 版 Aurora MySQL 的 Aurora 資料庫叢集。叢集包含寫入器資料庫執行個體和讀取器資料庫執行個體。`wait db-instance-available` 命令會暫停，直至寫入器資料庫執行個體可供使用為止。這正是叢集準備好升級之時。

```
aws rds create-db-cluster --db-cluster-identifier mynewdbcluster --engine aurora-mysql
 \
  --db-cluster-version 5.7.mysql_aurora.2.11.2
...
aws rds create-db-instance --db-instance-identifier mynewdbcluster-instance1 \
  --db-cluster-identifier mynewdbcluster --db-instance-class db.t4g.medium --engine
 aurora-mysql
...
aws rds wait db-instance-available --db-instance-identifier mynewdbcluster-instance1
```

您可以升級叢集的 Aurora MySQL 3.x 版本取決於叢集目前執行的 2.x 版本，以及叢集所在的 AWS 區域。使用第一個命令 `--output text`，只是顯示可用的目標版本。第二個命令顯示回應的完整 JSON 輸出。在該回應中，您可以查看詳細資訊，例如用於 `engine` 參數的 `aurora-mysql` 值。您還可以查看升級至 3.04.0 代表主要版本升級的事實。

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].{EngineVersion:EngineVersion}' --output text
5.7.mysql_aurora.2.11.2

aws rds describe-db-engine-versions --engine aurora-mysql --engine-version
 5.7.mysql_aurora.2.11.2 \
  --query '*[].[ValidUpgradeTarget]'
...
{
    "Engine": "aurora-mysql",
    "EngineVersion": "8.0.mysql_aurora.3.04.0",
    "Description": "Aurora MySQL 3.04.0 (compatible with MySQL 8.0.28)",
    "AutoUpgrade": false,
    "IsMajorVersionUpgrade": true,
    "SupportedEngineModes": [
        "provisioned"
    ],
    "SupportsParallelQuery": true,
    "SupportsGlobalDatabases": true,
    "SupportsBabelfish": false,
    "SupportsIntegrations": false
},
...
```

此範例說明，如果您輸入的目標版本編號不是叢集的有效升級目標，Aurora 不會執行升級。Aurora 也不會執行主要版本升級，除非您包含 --allow-major-version-upgrade 參數。如此一來，您就不會意外執行可能需要對應用程式的程式碼進行大量測試和變更的升級。

```
aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 5.7.mysql_aurora.2.09.2 --apply-immediately
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster
 operation: Cannot find upgrade target from 5.7.mysql_aurora.2.11.2 with requested
 version 5.7.mysql_aurora.2.09.2.

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.04.0 --region us-east-1 --apply-immediately
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster
 operation: The AllowMajorVersionUpgrade flag must be present when upgrading to a new
 major version.

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.04.0 --apply-immediately --allow-major-version-
upgrade
{
  "DBClusterIdentifier": "mynewdbcluster",
  "Status": "available",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.11.2"
}
```

叢集和相關資料庫執行個體的狀態需要一段時間才能變更為 upgrading。叢集和資料庫執行個體的版本編號只有在升級完成時才會變更。同樣，您可以使用寫入器資料庫執行個體的 wait db-instance-available 命令，等到升級完成後再繼續進行。

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].[Status,EngineVersion]' --output text
upgrading 5.7.mysql_aurora.2.11.2

aws rds describe-db-instances --db-instance-identifier mynewdbcluster-instance1 \
  --query '*[].
{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceStatus:DBInstanceStatus} | [0]'
{
    "DBInstanceIdentifier": "mynewdbcluster-instance1",
    "DBInstanceStatus": "upgrading"
}
```

```
aws rds wait db-instance-available --db-instance-identifier mynewdbcluster-instance1
```

此時，叢集的版本編號符合針對升級指定的版本編號。

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].[EngineVersion]' --output text

8.0.mysql_aurora.3.04.0
```

上述範例透過指定 `--apply-immediately` 參數立即執行升級。若要讓升級在方便的時間，當叢集不忙碌時發生，您可以指定 `--no-apply-immediately` 參數。這樣做會在叢集的下一個維護時段期間啟動升級。維護時段會定義維護操作可以開始的期間。長時間執行的操作可能無法在維護時段完成。因此，即使您預期升級可能需要很長時間，也不需要定義較大的維護時段。

下列範例會升級最初執行 Aurora MySQL 2.11.2 版的叢集。在 `describe-db-engine-versions` 輸出中，`False` 和 `True` 值表示 `IsMajorVersionUpgrade` 屬性。從 2.11.2 版開始，您可以升級至其他 2.* 版本。這些升級不會視為主要版本升級，因此不需要就地升級。就地升級僅適用於升級至清單中顯示的 3.* 版。

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].{EngineVersion:EngineVersion}' --output text
5.7.mysql_aurora.2.11.2

aws rds describe-db-engine-versions --engine aurora-mysql --engine-version
 5.7.mysql_aurora.2.10.2 \
  --query '*[].[ValidUpgradeTarget]|[0][0]|[*].[EngineVersion,IsMajorVersionUpgrade]'
 --output text

5.7.mysql_aurora.2.11.3 False
5.7.mysql_aurora.2.11.4 False
5.7.mysql_aurora.2.11.5 False
5.7.mysql_aurora.2.11.6 False
5.7.mysql_aurora.2.12.0 False
5.7.mysql_aurora.2.12.1 False
5.7.mysql_aurora.2.12.2 False
5.7.mysql_aurora.2.12.3 False
8.0.mysql_aurora.3.04.0 True
8.0.mysql_aurora.3.04.1 True
8.0.mysql_aurora.3.04.2 True
8.0.mysql_aurora.3.04.3 True
8.0.mysql_aurora.3.05.2 True
8.0.mysql_aurora.3.06.0 True
```

```
8.0.mysql_aurora.3.06.1 True
8.0.mysql_aurora.3.07.1 True

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.04.0 --no-apply-immediately --allow-major-
version-upgrade
...
```

在沒有指定維護時段的情況下建立叢集時，Aurora 會隨機挑選一週中的一天。在這種情況下，modify-db-cluster 命令會在週一提交。因此，我們將維護時段變更為週二上午。所有時間都以 UTC 時區表示。tue:10:00-tue:10:30 時段對應於太平洋時間上午 2:00-2:30。維護時段中的變更會立即生效。

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster --query '*[].
[PreferredMaintenanceWindow]'
[
    [
        "sat:08:20-sat:08:50"
    ]
]

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster --preferred-
maintenance-window tue:10:00-tue:10:30"
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster --query '*[].
[PreferredMaintenanceWindow]'
[
    [
        "tue:10:00-tue:10:30"
    ]
]
```

下列範例顯示如何取得升級產生事件的報告。--duration 引數代表擷取事件資訊的分鐘數。此引數是必要的，因為根據預設，describe-events 僅傳回最後一小時的事件。

```
aws rds describe-events --source-type db-cluster --source-identifier mynewdbcluster --
duration 20160
{
    "Events": [
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "DB cluster created",
```

```
            "EventCategories": [
                "creation"
            ],
            "Date": "2022-11-17T01:24:11.093000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing online pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T22:57:08.450000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing offline pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T22:57:59.519000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-
mynewdbcluster-5-7-mysql-aurora-2-10-2-to-8-0-mysql-aurora-3-02-0-2022-11-18-22-55].",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:00:22.318000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Cloning volume.",
            "EventCategories": [
                "maintenance"
            ],
```

```
            "Date": "2022-11-18T23:01:45.428000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Purging undo records for old row versions.
Records remaining: 164",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:02:25.141000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Purging undo records for old row versions.
Records remaining: 164",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:06:23.036000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Upgrading database objects.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:06:48.208000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster major version has been upgraded",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:10:28.999000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
```

```
        }
    ]
}
```

尋找 Aurora MySQL 主要版本升級失敗的原因

在教學課程中，從 Aurora MySQL 第 2 版升級至第 3 版成功。但是，如果升級失敗，您會想知道原因。

您可以從使用 `describe-events` AWS CLI 命令開始查看資料庫叢集事件。此範例顯示過去 10 小時內 `mydbcluster` 的事件。

```
aws rds describe-events \
    --source-type db-cluster \
    --source-identifier mydbcluster \
    --duration 600
```

在這種情況下，我們發生升級預先檢查失敗。

```
{
    "Events": [
        {
            "SourceIdentifier": "mydbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster engine version upgrade started.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2024-04-11T13:23:22.846000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster"
        },
        {
            "SourceIdentifier": "mydbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster is in a state that cannot be upgraded: Upgrade
 prechecks failed. For more details, see the
             upgrade-prechecks.log file. For more information on troubleshooting the
 cause of the upgrade failure, see
             https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
AuroraMySQL.Upgrading.Troubleshooting.html",
            "EventCategories": [
                "maintenance"
```

```
            ],
            "Date": "2024-04-11T13:23:24.373000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster"
        }
    ]
}
```

若要診斷問題的確切原因，請檢查寫入器資料庫執行個體的資料庫日誌。升級到 Aurora MySQL 第 3 版失敗時，寫入器執行個體會包含一個日誌檔案，名稱為 upgrade-prechecks.log。此範例說明如何偵測該日誌是否存在，然後將其下載為本機檔案進行檢查。

```
aws rds describe-db-log-files --db-instance-identifier mydbcluster-instance \
    --query '*[].[LogFileName]' --output text

error/mysql-error-running.log
error/mysql-error-running.log.2024-04-11.20
error/mysql-error-running.log.2024-04-11.21
error/mysql-error.log
external/mysql-external.log
upgrade-prechecks.log

aws rds download-db-log-file-portion --db-instance-identifier mydbcluster-instance \
    --log-file-name upgrade-prechecks.log \
    --starting-token 0 \
    --output text >upgrade_prechecks.log
```

upgrade-prechecks.log 檔案為 JSON 格式。我們使用 --output text 選項來下載檔案，避免在另一個 JSON 包裝函式中對 JSON 輸出進行編碼。針對 Aurora MySQL 第 3 版升級，此日誌一律包含特定資訊和警告訊息。它只會在升級失敗時包含錯誤訊息。如果升級成功，則根本不會產生日誌檔案。

若要摘要所有錯誤並顯示關聯的物件和描述欄位，您可以對 upgrade-prechecks.log 檔案的內容執行命令 grep -A 2 '"level": "Error"'。這麼做會顯示每個錯誤行及其後兩行。其中包含對應資料庫物件的名稱，以及如何修正問題的指引。

```
$ cat upgrade-prechecks.log | grep -A 2 '"level": "Error"'

"level": "Error",
"dbObject": "problematic_upgrade.dangling_fulltext_index",
"description": "Table `problematic_upgrade.dangling_fulltext_index` contains dangling
 FULLTEXT index. Kindly recreate the table before upgrade."
```

在此範例中，您可以在違規的資料表上執行下列 SQL 命令，嘗試修正問題，也可以在沒有懸置索引的情況下重新建立資料表。

```
OPTIMIZE TABLE problematic_upgrade.dangling_fulltext_index;
```

然後重試升級。

Aurora MySQL 就地升級疑難排解

使用下列提示有助於對 Aurora MySQL 就地升級的問題進行疑難排解。這些提示不適用於 Aurora Serverless 資料庫叢集。

就地升級被取消或緩慢的原因	Effect	允許在維護時段完成就地升級的解決方案
相關聯的 Aurora 跨區域複本尚未修補	Aurora 會取消升級。	請升級 Aurora 跨區域複本，然後再試一次。
叢集的 XA 交易處於準備狀態	Aurora 會取消升級。	遞交或復原所有準備好的 XA 交易。
叢集正在處理任何資料定義語言 (DDL) 陳述式	Aurora 會取消升級。	在所有 DDL 陳述式完成後，請等待並執行升級。
叢集中的多行有未遞交的變更	升級可能需要很長時間。	升級程序會復原未遞交的變更。此條件的指標為 `TRX_ROWS_MODIFIED` 資料表 `INFORMATION_SCHEMA.INNODB_TRX` 中的值。 僅在遞交或復原所有大型交易之後，才考慮執行升級。
叢集具有大量的復原記錄	升級可能需要很長時間。	即使未遞交的交易不會影響大量的資料列，也可能會涉及大量資料。例如，您可能插入大型 BLOB。Aurora 不會自動偵測或產生這種交易活動的事件。此情況的指標為歷史記錄清單長度 (HLL)。升級程序會復原未遞交的變更。 您可以從 `SHOW ENGINE INNODB STATUS` SQL 命令檢查輸出中的 HLL，或直接使用以下 SQL 查詢：

就地升級被取消或緩慢的原因	Effect	允許在維護時段完成就地升級的解決方案
		`SELECT count FROM information_schema .innodb_metrics WHERE name = 'trx_rseg_history_len';` 您也可以在 Amazon CloudWatch 中監控 `RollbackSegmentHistoryListLength` 指標。 僅在 HLL 減小之後，才考慮執行升級。
叢集正在遞交大型二進位日誌交易	升級可能需要很長時間。	升級程序會等到套用二進位日誌變更之後。在此期間，可能會啟動更多交易或 DDL 陳述式，進一步降低升級程序的速度。 在叢集不忙於產生二進位日誌複寫變更時排程升級程序。Aurora 不會自動偵測或產生此條件的事件。
因檔案移除或損毀所導致的結構描述不一致	Aurora 會取消升級。	將暫存資料表的預設儲存引擎從 MyISAM 變更為 InnoDB。執行以下步驟： 1. 將 `default_tmp_storage_engine` 資料庫參數修改為 `InnoDB`。 2. 重新啟動資料庫叢集。 3. 重新啟動後，請確認 `default_tmp_storage_engine` 資料庫參數已設定為 `InnoDB`。使用下列命令： `show global variables like 'default_tmp_storage_engine';` 4. 切勿建立任何使用 MyISAM 儲存引擎的暫存資料表。我們建議您暫停任何資料庫工作負載，且勿建立任何新的資料庫連線，因為即將進行升級。 5. 請再次嘗試就地升級。

就地升級被取消或緩慢的原因	Effect	允許在維護時段完成就地升級的解決方案

主要使用者已刪除	Aurora 會取消升級。	**Important**
		請勿刪除主要使用者。
		
		不過，如果由於某些原因，您應該刪除主要使用者，並使用下列 SQL 命令進行還原：
		
		`CREATE USER 'master_username '@'% IDENTIFIED BY ' master_user_password ' REQUIRE NONE PASSWORD EXPIRE DEFAULT ACCOUNT UNLOCK;`
		
		`GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES, INDEX, ALTER, SHOW DATABASES, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, LOAD FROM S3, SELECT INTO S3, INVOKE LAMBDA, INVOKE SAGEMAKER, INVOKE COMPREHEND ON *.* TO 'master_username '@'% WITH GRANT OPTION;`

如需故障診斷導致升級預先檢查失敗之問題的詳細資訊，請參閱下列部落格：

- Amazon Aurora MySQL 第 2 版 (與 MySQL 5.7 相容) 到第 3 版 (與 MySQL 8.0 相容) 升級檢查清單，第 1 部分
- Amazon Aurora MySQL 第 2 版 (與 MySQL 5.7 相容) 到第 3 版 (與 MySQL 8.0 相容) 升級檢查清單，第 2 部分

您可以使用下列步驟，對以上資料表中的某些條件執行自己的檢查。如此一來，在您知道資料庫處於升級可以順利且快速完成的狀態時，即可排程升級。

- 您可以執行 XA RECOVER 陳述式，來檢查開啟的 XA 交易。您隨後可以遞交或復原 XA 交易，再開始升級。
- 您可以執行 SHOW PROCESSLIST 陳述式，然後在輸出中尋找 CREATE、DROP、ALTER、RENAME 和 TRUNCATE，來檢查 DDL 陳述式。在開始升級之前，允許所有 DDL 陳述式完成。
- 您可以查詢 INFORMATION_SCHEMA.INNODB_TRX 資料表，檢查未遞交列的總數。該資料表包含每項交易的一個資料列。TRX_ROWS_MODIFIED 資料欄包含交易修改或插入的列數。
- 您可以執行 SHOW ENGINE INNODB STATUS SQL 陳述式，然後在輸出中尋找 History list length，來檢查 InnoDB 歷史記錄清單的長度。您還可以直接執行下列查詢來檢查值：

```
SELECT count FROM information_schema.innodb_metrics WHERE name =
 'trx_rseg_history_len';
```

  歷史記錄清單的長度對應於資料庫存放的還原資訊量，以實作多版本並行控制 (MVCC)。

Aurora MySQL 第 3 版的升級後清除

完成將任何 Aurora MySQL 第 2 版叢集升級至 Aurora MySQL 第 3 版之後，您可以執行下列其他清除動作：

- 為任何自訂參數群組建立新的 MySQL 8.0 相容版本。將任何必要的自訂參數值套用至新的參數群組。
- 更新任何 CloudWatch 警示、設定指令碼等，以針對其名稱受到包容性語言變更影響的任何指標使用新名稱。如需這類指標的清單，請參閱 Aurora MySQL 第 3 版的包容性語言變更。
- 更新任何 CloudFormation 範本，針對其名稱受到包容性語言變更影響的任何組態參數使用新名稱。如需這類參數的清單，請參閱 Aurora MySQL 第 3 版的包容性語言變更。

空間索引

升級至 Aurora MySQL 第 3 版之後，請檢查您是否需要捨棄或重新建立與空間索引相關的物件和索引。在 MySQL 8.0 之前，Aurora 可以使用不包含空間資源識別符 (SRID) 的索引來最佳化空間查詢。Aurora MySQL 第 3 版只使用包含 SRID 的空間索引。在升級期間，Aurora 會自動捨棄任何沒有 SRID 的空間索引，並在資料庫日誌中列印警告訊息。如果您觀察到這類警告訊息，請在升級之後使用 SRID 建立新的空間索引。如需 MySQL 8.0 中對空間函數和資料類型之變更的詳細資訊，請參閱《MySQL 參考手冊》中的 MySQL 8.0 中的變更。

Amazon Aurora MySQL 的資料庫引擎更新和修正

您可以在 Amazon Aurora MySQL 相容版本的版本備註中找到以下資訊：

- Amazon Aurora MySQL 第 3 版的資料庫引擎更新
- Amazon Aurora MySQL 第 2 版的資料庫引擎更新
- Amazon Aurora MySQL 第 1 版的資料庫引擎更新 (已棄用)
- Aurora MySQL 資料庫引擎更新所修正的 MySQL 錯誤
- Amazon Aurora MySQL 中修正的安全漏洞

使用 Amazon Aurora PostgreSQL

Amazon Aurora PostgreSQL 是與 PostgreSQL 相容的完全受管且符合 ACID 規範的關聯式資料庫引擎，結合了 Amazon Aurora 的速度、可靠性和可管理性，以及開源資料庫的簡易性和成本效益。Aurora PostgreSQL 是 PostgreSQL 的便利替代方案，以簡單且經濟實惠的方式設定、操作及擴展新的及現有的 PostgreSQL 部署，讓您得以心無旁，專注於業務和應用程式。若要進一步了解 Aurora 的一般情況，請參閱 什麼是 Amazon Aurora？。

除了 Aurora 的優點之外，Aurora PostgreSQL 還提供從 Amazon RDS 遷移至 Aurora 的便利途徑，並提供按鈕式遷移工具，可將您現有的 RDS for PostgreSQL 應用程式轉換為 Aurora PostgreSQL。例行資料庫任務 (例如佈建、修補、備份、復原、故障偵測和修復) 也可以使用 Aurora PostgreSQL 輕鬆進行管理。

Aurora PostgreSQL 可以搭配許多業界標準一起使用。舉例來說，您可以使用 Aurora PostgreSQL 資料庫來建置符合 HIPAA 規範的應用程式及存放醫療保健相關資訊，包括與 AWS簽訂完整的商業夥伴協議 (BAA)，並據此保障的受保護醫療資訊 (PHI)。

Aurora PostgreSQL 符合 FedRAMP HIGH。如需 AWS 和合規工作的詳細資訊，請參閱AWS 合規計劃範圍內的服務。

> **Important**
>
> 如果您的 Aurora PostgreSQL 資料庫叢集發生問題，您的 AWS 支援代理程式可能需要資料庫運作狀態的詳細資訊。目標是確保 AWS Support 在最短的時間內盡可能取得所需的資訊。
> 您可以使用 PG Collector 來協助在合併的 HTML 檔案中收集寶貴的資料庫資訊。如需 PG Collector、如何執行以及如何下載 HTML 報告的詳細資訊，請參閱 PG Collector。
> 成功完成後，除非另有說明，否則指令碼會以可讀的 HTML 格式傳回輸出。該指令碼旨在從 HTML 中排除任何可能危及您業務的資料或安全性詳細資訊。其也不會對您的資料庫或其環境進行任何修改。但是，如果您在 HTML 中發現任何不希望分享的資訊，請在上傳 HTML 之前移除有問題資訊。當您能夠放心分享 HTML 中的資訊時，請使用支援案例的案例詳細資訊中的附件區段來上傳 HTML。

主題

- 使用資料庫預覽環境
- Amazon Aurora PostgreSQL 的安全性

- 將應用程式更新為使用新的 SSL/TLS 憑證來連線至 Aurora PostgreSQL 資料庫叢集
- 搭配 Aurora PostgreSQL 使用 Kerberos 身分驗證
- 將資料遷移至與 PostgreSQL 相容的 Amazon Aurora
- 在 Aurora PostgreSQL 中最佳化查詢效能
- 在 Aurora PostgreSQL 中使用未記錄的資料表
- 在 Amazon Aurora PostgreSQL 上使用 PostgreSQL 自動清空
- 在 Amazon Aurora PostgreSQL 中管理 TOAST OID 爭用
- 使用 Babelfish for Aurora PostgreSQL
- Amazon Aurora PostgreSQL 的效能和擴展
- 調校 Aurora PostgreSQL 的等待事件
- 使用 Amazon DevOps Guru 主動洞察，調校 Aurora PostgreSQL
- Amazon Aurora PostgreSQL 的最佳實務
- 以 Amazon Aurora PostgreSQL 進行複寫
- 在 Aurora PostgreSQL 中使用本機寫入轉送
- 使用 Aurora PostgreSQL 作為 Amazon Bedrock 的知識庫
- 將 Amazon Aurora PostgreSQL 與其他 AWS 服務整合
- 監控 Aurora PostgreSQL 的查詢執行計劃和尖峰記憶體
- 管理 Aurora PostgreSQL 的查詢執行計劃
- 使用擴充功能和外部資料包裝函式
- 使用適用於 PostgreSQL 的受信任語言延伸模組
- Amazon Aurora PostgreSQL 參考
- Amazon Aurora PostgreSQL 的資料庫引擎更新

使用資料庫預覽環境

PostgreSQL 社群每年發行 PostgreSQL 的新主要版本。同樣地，Amazon Aurora 會使 PostgreSQL 主要版本當作預覽版本提供。這可讓您使用預覽版本建立資料庫叢集，並在資料庫預覽環境中測試其功能。

資料庫預覽環境中的 Aurora PostgreSQL 資料庫叢集功能與其他 Aurora PostgreSQL 資料庫叢集類似。不過，您無法使用預覽版本進行生產。

請謹記下列重要限制：

- 所有資料庫執行個體和資料庫叢集在建立後 60 天就會刪除，也會一併刪除任何備份和快照。
- 您只能在以 Amazon VPC 服務為基礎的 Virtual Private Cloud (VPC) 中建立資料庫執行個體。
- 您無法將資料庫執行個體的快照複製到生產環境。

預覽版支援下列選項：

- 您只能使用 r5、r6g、r6i、r7g、r7i、r8g、x2g、t3 和 t4g 執行個體類型建立資料庫執行個體。如需執行個體類別的詳細資訊，請參閱Amazon Aurora 資料庫執行個體類別。
- 您可以同時使用單一可用區域和多可用區域部署。
- 您可以使用標準 PostgreSQL 傾印和載入函數，從資料庫預覽環境匯出資料庫，或匯入資料庫至資料庫預覽環境。

支援的資料庫執行個體類別類型

Amazon Aurora PostgreSQL 在預覽區域中支援下列資料庫執行個體類別：

記憶體最佳化類別

- db.r5 – 記憶體優化執行個體類別
- db.r6g – 採用 AWS Graviton2 處理器的記憶體最佳化執行個體類別
- db.r6i – 記憶體優化執行個體類別
- db.r7g – 採用 AWS Graviton3 處理器的記憶體最佳化執行個體類別
- db.r7i – 記憶體最佳化的執行個體類別
- db.r8g – 採用 AWS Graviton4 處理器的記憶體最佳化執行個體類別
- db.x2g – 採用 AWS Graviton2 處理器的記憶體最佳化執行個體類別

> **Note**
>
> 如需執行個體類別清單的詳細資訊，請參閱 資料庫執行個體類別的類型。

爆量類別

- db.t3.medium
- db.t3.large

- db.t4g.medium
- db.t4g.large

預覽環境中不支援的功能

預覽環境中無法使用下列功能：

- Aurora Serverless v1 和 v2
- 主要版本升級 (MVU)
- 預覽區域中不會發行新的次要版本
- RDS for PostgreSQL 到 Aurora PostgreSQL 輸入複寫
- Amazon RDS 藍/綠部署
- 跨區域快照複製
- Aurora 全球資料庫
- 資料庫活動串流 (DAS)、RDS Proxy 和 AWS DMS
- 自動擴展僅供讀取複本
- AWS Bedrock
- RDS 匯出
- Performance Insights
- 自訂端點
- 複製快照
- 零 ETL
- Babelfish
- 所有擴充功能

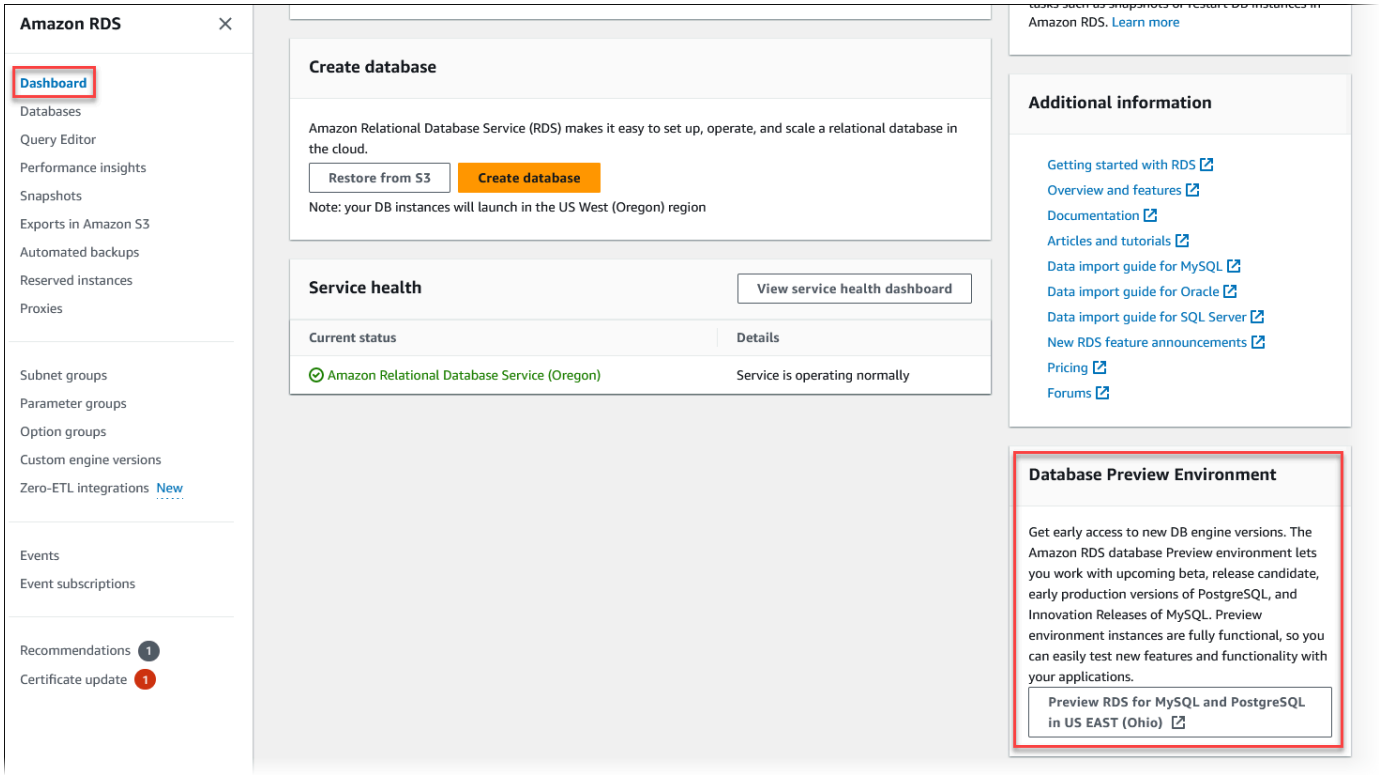
在預覽環境中建立新的資料庫叢集

使用下列程序在預覽環境中建立資料庫叢集。

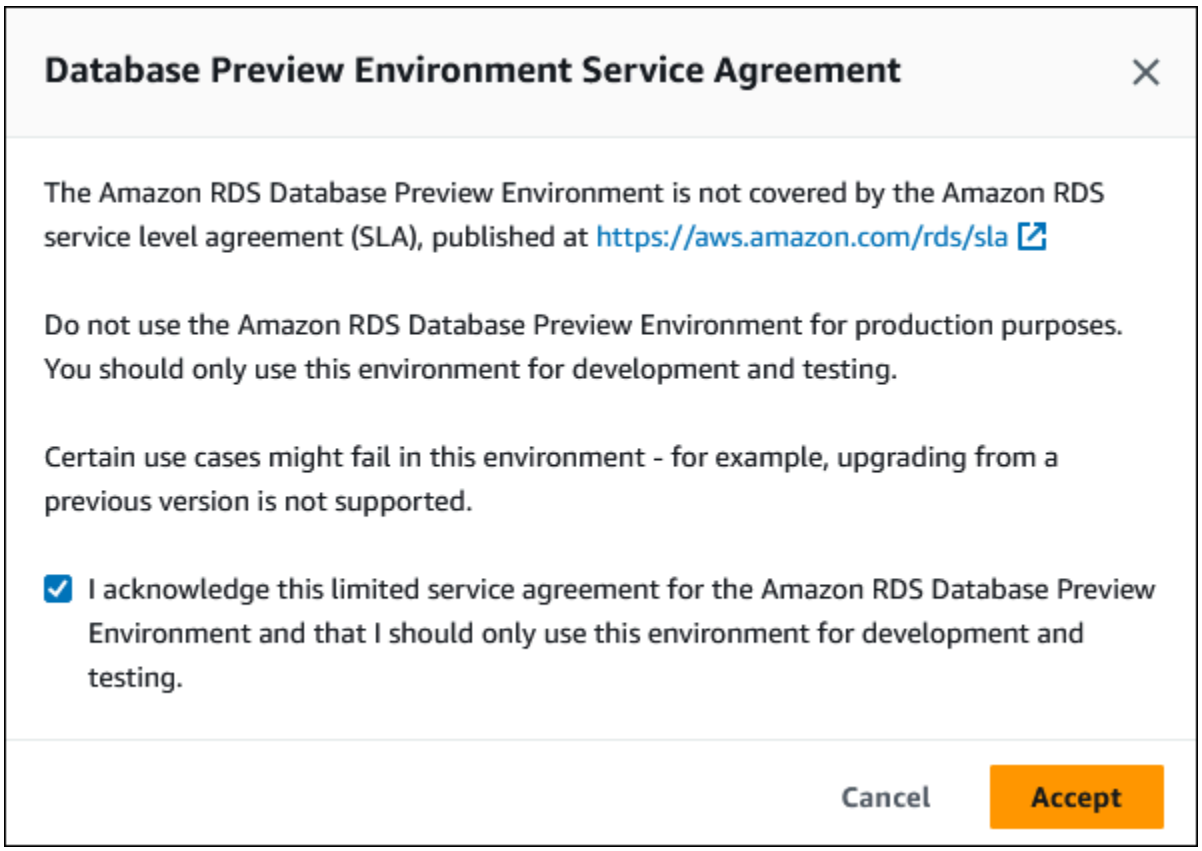
在預覽環境中建立資料庫叢集

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。

2. 從導覽窗格中選擇 Dashboards (儀表板)。
3. 在儀表板頁面中，找出儀表板頁面上的 Database Preview Environment (資料庫預覽環境) 區段，如下圖所示。



您可以直接導覽至資料庫預覽環境。在繼續之前，您必須確認並接受限制。



4. 若要建立 Aurora PostgreSQL 資料庫叢集，請遵循與建立任何 Aurora 資料庫叢集相同的程序。如需詳細資訊，請參閱建立 Amazon Aurora 資料庫叢集。

若要使用 Aurora API 或 在資料庫預覽環境中建立執行個體 AWS CLI，請使用下列端點。

```
rds-preview.us-east-2.amazonaws.com
```

資料庫預覽環境中的 PostgreSQL 第 17 版

⚠ 這是 Aurora PostgreSQL 17 版的預覽文件。內容可能變動。

PostgreSQL 17.0 版現可在 Amazon RDS 資料庫預覽環境中使用。PostgreSQL 第 17 版包含下列 PostgreSQL 文件中所述的數個改善：

- PostgreSQL 17 已發行

如需資料庫預覽環境的資訊，請參閱 使用資料庫預覽環境。若要從主控台存取預覽環境，請選取 https://console.aws.amazon.com/rds-preview/。

Amazon Aurora PostgreSQL 的安全性

如需有關 Aurora 安全性的一般概觀，請參閱 Amazon Aurora 的安全性。您可在幾個不同的層級管理 Amazon Aurora PostgreSQL 的安全性：

- 若要控制誰可以在 Aurora PostgreSQL 資料庫叢集和資料庫執行個體上執行 Amazon RDS 管理動作，請使用 AWS Identity and Access Management (IAM)。IAM 在使用者可存取服務之前處理使用者身分的身分驗證。其還會處理授權，亦即，是否允許使用者去做其想做的事情。IAM 資料庫身分驗證是一種額外的身分驗證方法，您可在建立 Aurora PostgreSQL 資料庫叢集時選擇該方法。如需詳細資訊，請參閱Amazon Aurora 的 Identity and access management。

  若您是將 IAM 與 Aurora PostgreSQL 資料庫叢集搭配使用，請先使用您的 IAM 憑證登入 AWS 管理主控台 ，然後才於 https://console.aws.amazon.com/rds/ 開啟 Amazon RDS 主控台。
- Aurora 資料庫叢集必須在以 Amazon VPC 服務為基礎的虛擬私有雲端 (VPC) 中建立。若要控制哪些裝置和 Amazon EC2 執行個體可以開放對 VPC 中 Aurora 資料庫叢集的資料庫執行個體端點和連接埠的連線，可以使用 VPC 安全群組。您可使用 Secure Sockets Layer (SSL) 對這些端點和連接埠建立連線。此外，貴公司的防火牆規則可控管在公司內執行的裝置是否可開啟與資料庫執行個體的連線。如需 VPC 的詳細資訊，請參閱Amazon VPC 和 Amazon Aurora。

  支援的 VPC 租用取決於您的 Aurora PostgreSQL 資料庫叢集所使用的資料庫執行個體類別。使用 `default` VPC 租用時，資料庫叢集在共用硬體上執行。使用 `dedicated` VPC 租用時，資料庫叢集會在專用硬體執行個體上執行。爆量效能資料庫執行個體類別僅支援預設的 VPC 租用。爆量效能資料庫執行個體類別包括 db.t3 和 db.t4g 資料庫執行個體類別。其他所有 Aurora PostgreSQL 資料庫執行個體類別都支援預設和專用的 VPC 租用。

  如需執行個體類別的詳細資訊，請參閱Amazon Aurora 資料庫執行個體類別。如需 `default` 和 `dedicated` VPC 租用的詳細資訊，請參閱《Amazon Elastic Compute Cloud 使用者指南》中的專用執行個體。
- 如要對 Amazon Aurora 資料庫叢集上執行的 PostgreSQL 資料庫授予權限，您可採取與 PostgreSQL 獨立執行個體相同的通用方法。`CREATE ROLE`、`ALTER ROLE`、`GRANT` 和 `REVOKE` 等命令的運作方式與內部部署資料庫所使用的命令相同，會直接修改資料庫、結構描述，和資料表。

  PostgreSQL 會使用角色來管理權限。`rds_superuser` 角色是 Aurora PostgreSQL 資料庫叢集上權限最高的角色。此角色會自動建立，並授予建立資料庫叢集的使用者 (主要使用者帳戶，依預設為 `postgres`)。如需進一步了解，請參閱 了解 PostgreSQL 角色和許可。

所有可用的 Aurora PostgreSQL 版本 (包括第 10、11、12、13、14 版及更高版本) 支援密碼的 Salted Challenge Response Authentication Mechanism (SCRAM) 功能，作為訊息摘要 (MD5) 的替代方案。建議您使用 SCRAM，因為它比 MD5 更安全。如需詳細資訊，包括如何將資料庫使用者密碼從 MD5 遷移到 SCRAM，請參閱 使用 SCRAM 進行 PostgreSQL 密碼加密。

了解 PostgreSQL 角色和許可

當您使用 建立 Aurora PostgreSQL 資料庫叢集 時 AWS 管理主控台，會同時建立管理員帳戶。依預設，其名稱為 `postgres`，如下列螢幕擷取畫面所示：

您可以選擇其他名稱，而不是接受預設值 (`postgres`)。若是如此，則選擇的名稱必須以字母開頭，且介於 1 到 16 個英數字元之間。為簡單起見，我們在本指南中使用主要使用者帳戶的預設值 (`postgres`)，來指稱主要使用者帳戶。

如果您使用 `create-db-cluster` AWS CLI 而非 AWS 管理主控台，您可以透過使用 `master-username` 參數傳遞使用者名稱來建立使用者名稱。如需詳細資訊，請參閱 步驟 2：建立 Aurora PostgreSQL 資料庫叢集。

無論您是使用 AWS 管理主控台、 AWS CLI或 Amazon RDS API，還是使用預設`postgres`名稱或選擇不同的名稱，這個第一個資料庫使用者帳戶都是 `rds_superuser`群組的成員，並且具有 `rds_superuser` 權限。

主題

- 了解 rds_superuser 角色
- 控制使用者對 PostgreSQL 資料庫的存取
- 委派和控制使用者密碼管理
- 使用 SCRAM 進行 PostgreSQL 密碼加密

了解 rds_superuser 角色

於 PostgreSQL 中，角色可定義使用者、群組或授予群組或使用者對資料庫中各種物件的一組特定權限。CREATE USER 和 CREATE GROUP 的 PostgreSQL 命令已為更為通用的 CREATE ROLE 所取代，其具有可區分資料庫使用者的特定屬性。資料庫使用者可視為具有 LOGIN 權限的角色。

> **Note**
>
> CREATE USER 和 CREATE GROUP 命令仍可使用。如需詳細資訊，請參閱 PostgreSQL 文件中的資料庫角色。

postgres 使用者是 Aurora PostgreSQL 資料庫叢集上具最高權限的資料庫使用者。其具有下列 CREATE ROLE 陳述式所定義的特性。

```
CREATE ROLE postgres WITH LOGIN NOSUPERUSER INHERIT CREATEDB CREATEROLE NOREPLICATION
 VALID UNTIL 'infinity'
```

屬性 NOSUPERUSER、NOREPLICATION、INHERIT，和 VALID UNTIL 'infinity' 為 CREATE ROLE 的預設選項，除非另有指定。

根據預設，postgres 具有授與 rds_superuser 角色的權限，以及建立角色和資料庫的許可。rds_superuser 角色可讓 postgres 使用者執行下列動作：

- 新增可與 Aurora PostgreSQL。如需詳細資訊，請參閱 使用擴充功能和外部資料包裝函式。
- 建立使用者的角色，並授予使用者權限。如需詳細資訊，請參閱 PostgreSQL 文件中的 CREATE ROLE 和 GRANT。
- 建立資料庫。如需詳細資訊，請參閱 PostgreSQL 文件中的 CREATE DATABASE。
- 將 rds_superuser 權限授予並無這些權限的使用者角色，並視需要撤銷這些權限。建議您僅將此角色授予執行超級使用者任務的使用者。換句話說，您可以將此角色授予資料庫管理員 (DBA) 或系統管理員。
- 對不具 rds_replication 角色的資料庫使用者授予 (和撤銷) rds_superuser 角色。
- 對不具 rds_password 角色的資料庫使用者授予 (和撤銷) rds_superuser 角色。
- 使用 pg_stat_activity 檢視，取得有關所有資料庫連線的狀態資訊。如有需要，rds_superuser 可使用 pg_terminate_backend 或 pg_cancel_backend 停止任何連線。

於 CREATE ROLE postgres... 陳述式中，您可看到 postgres 使用者角色明確禁止 PostgreSQL superuser 權限。Aurora PostgreSQL 為受管服務，因此您無法存取主機作業系統，也無法使用 PostgreSQL superuser 帳戶進行連線。許多需要在獨立 PostgreSQL 上進行 superuser 存取的任務是由 Aurora 自動管理。

如需有關授予權限的詳細資訊，請參閱 PostgreSQL 文件中的 GRANT。

rds_superuser 角色是 Aurora PostgreSQL 資料庫叢集中數個預先定義角色的其中一個。

> **Note**
>
> 在 PostgreSQL 13 和更早版本中，預先定義角色稱為預設角色。

於下列清單中，您可以找到為新的 Aurora PostgreSQL 資料庫叢集自動建立的一些其他預先定義角色。預先定義的角色及其權限無法進行變更。您無法為這些預先定義角色停止、重新命名或修改權限。嘗試這麼做會造成錯誤。

- rds_password – 可變更密碼並為資料庫使用者設定密碼約束的角色。依預設授予此角色 rds_superuser 角色，並可授予資料庫使用者該角色。如需詳細資訊，請參閱控制使用者對 PostgreSQL 資料庫的存取。
  - 對於 14 之前的 RDS for PostgreSQL 版本，rds_password 角色可以變更密碼，並為資料庫使用者和具有 rds_superuser 角色的使用者設定密碼限制。從 RDS for PostgreSQL 第 14 版及更新版本，rds_password 角色可以變更密碼，並僅為資料庫使用者設定密碼限制。只有具有 rds_superuser 角色的使用者才能對具有 rds_superuser 角色的其他使用者執行這些動作。
- rdsadmin – 為處理具有 superuser 權限的管理員將在獨立 PostgreSQL 資料庫上執行的許多管理任務而建立的角色。此角色由 Aurora PostgreSQL 在內部用於許多管理任務。

檢視角色及其權限

您可以根據 PostgreSQL 版本，使用不同的命令在 RDS for PostgreSQL 資料庫執行個體中檢視預先定義的角色及其權限。若要查看所有預先定義的角色，您可以連線至 RDS for PostgreSQL 資料庫執行個體，並使用 psql 執行下列命令。

對於 **psql** 第 15 版和更早版本

連線至 RDS for PostgreSQL 資料庫執行個體，並在 psql 中使用 \du 命令：

```
postgres=> \du
```

```
                                                                List of roles
    Role name    |                         Attributes                          |
                 Member of
-----------------+-------------------------------------------------------------
+-----------------------------------------------------
 postgres        | Create role, Create DB                                     +|
 {rds_superuser}
                 | Password valid until infinity                               |
 rds_ad          | Cannot login                                                | {}
 rds_iam         | Cannot login                                                | {}
 rds_password    | Cannot login                                                | {}
 rds_replication | Cannot login                                                | {}
 rds_superuser   | Cannot login                                                |
 {pg_monitor,pg_signal_backend,rds_password,rds_replication}
 rdsadmin        | Superuser, Create role, Create DB, Replication, Bypass RLS+| {}
                 | Password valid until infinity                               |
```

對於 **psql** 第 16 版和更新版本

```
postgres=> \drg+
                            List of role grants
   Role name   |          Member of          |       Options       | Grantor
---------------+-----------------------------+---------------------+----------
 postgres      | rds_superuser               | INHERIT, SET        | rdsadmin
 rds_superuser | pg_checkpoint               | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | pg_monitor                  | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | pg_signal_backend           | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | pg_use_reserved_connections | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | rds_password                | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | rds_replication             | ADMIN, INHERIT, SET | rdsadmin
```

若要在沒有版本相依性的情況下檢查角色成員資格，您可以使用下列 SQL 查詢：

```
SELECT m.rolname AS "Role name", r.rolname AS "Member of"
FROM pg_catalog.pg_roles m
JOIN pg_catalog.pg_auth_members pam ON (pam.member = m.oid)
LEFT JOIN pg_catalog.pg_roles r ON (pam.roleid = r.oid)
LEFT JOIN pg_catalog.pg_roles g ON (pam.grantor = g.oid)
WHERE m.rolname !~ '^pg_'
ORDER BY 1, 2;
```

於輸出中，您可看到 rds_superuser 並非資料庫使用者角色 (無法登入)，但其具有許多其他角色的權限。您還可以看到資料庫使用者 postgres 是 rds_superuser 角色的成員。如前所述，postgres 是 Amazon RDS 主控台 Create database (建立資料庫) 頁面中的預設值。若選擇其他名稱，則該名稱將顯示於角色清單中。

> **Note**
>
> Aurora PostgreSQL 15.2 和 14.7 版導入了限制性的 rds_superuser 角色行為。即使已將 rds_superuser 角色授予 Aurora PostgreSQL 使用者，使用者仍需被授予對應資料庫的 CONNECT 權限才能連線。在 Aurora PostgreSQL 14.7 和 15.2 版之前，只要已將 rds_superuser 角色授予使用者，該使用者就能夠連線到任何資料庫和系統資料表。這種限制行為符合 AWS 和 Amazon Aurora 對持續改善安全性的承諾。
> 如果您的應用程式受到上述增強功能的影響，請更新應用程式中的個別邏輯。

控制使用者對 PostgreSQL 資料庫的存取

PostgreSQL 中的新資料庫會永遠使用資料庫 public 結構描述中的一組預設權限建立，允許所有資料庫使用者和角色建立物件。例如，這些權限可讓資料庫使用者連接至資料庫，並在連線時建立暫存資料表。

為了對您 Aurora PostgreSQL 資料庫叢集主節點上建立之資料庫執行個體的使用者存取進行更好地控制，我們建議您撤消這些預設的 public 權限。完成此作業後，您可更精細地為資料庫使用者授予特定權限，如下列程序所示。

如要設定新資料庫執行個體的角色和權限

假設您正在一個剛建立的 Aurora PostgreSQL 資料庫叢集上設定資料庫，以供多位研究人員使用，其皆需要資料庫的讀寫存取權。

1. 使用 psql (或 pgAdmin) 連接至您 Aurora PostgreSQL 資料庫叢集上的主資料庫執行個體：

   ```
   psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

   出現提示時，輸入您的密碼。psql 用戶端連接並顯示預設管理連接資料庫 postgres=> 作為提示。

2. 如要防止資料庫使用者在 public 結構描述中建立物件，請執行下列動作：

```
postgres=> REVOKE CREATE ON SCHEMA public FROM PUBLIC;
REVOKE
```

3. 接下來，您會建立新的資料庫執行個體：

```
postgres=> CREATE DATABASE lab_db;
CREATE DATABASE
```

4. 從此新資料庫上 PUBLIC 結構描述撤消所有權限。

```
postgres=> REVOKE ALL ON DATABASE lab_db FROM public;
REVOKE
```

5. 建立一個資料庫使用者的角色。

```
postgres=> CREATE ROLE lab_tech;
CREATE ROLE
```

6. 使具有此角色的資料庫使用者可連接至資料庫。

```
postgres=> GRANT CONNECT ON DATABASE lab_db TO lab_tech;
GRANT
```

7. 授予具 lab_tech 角色的所有使用者此資料庫的所有權限。

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_db TO lab_tech;
GRANT
```

8. 建立資料庫使用者，如下所示：

```
postgres=> CREATE ROLE lab_user1 LOGIN PASSWORD 'change_me';
CREATE ROLE
postgres=> CREATE ROLE lab_user2 LOGIN PASSWORD 'change_me';
CREATE ROLE
```

9. 授予這兩個使用者與 lab_tech 角色關聯的權限：

```
postgres=> GRANT lab_tech TO lab_user1;
GRANT ROLE
postgres=> GRANT lab_tech TO lab_user2;
GRANT ROLE
```

至此，lab_user1 和 lab_user2 便可連接 lab_db 資料庫。此範例並未遵循企業使用的最佳實務，其中可能包括建立多個資料庫執行個體、不同的結構描述，及授予有限的權限。如需更多完整資訊和其他方案，請參閱管理 PostgreSQL 使用者和角色。

如需有關 PostgreSQL 資料庫中權限的詳細資訊，請參閱 PostgreSQL 文件中的 GRANT 命令。

委派和控制使用者密碼管理

作為 DBA，您可能想要委派管理使用者密碼。或者，您可能想要防止資料庫使用者變更其密碼或重新設定密碼約束 (例如密碼生命週期)。如要確保只有您選擇的資料庫使用者才可變更密碼設定，您可開啟受限制的密碼管理功能。啟動此功能時，只有那些已被授予 rds_password 角色的資料庫使用者才可管理密碼。

> **Note**
>
> 如要使用受限制的密碼管理，您的 Aurora PostgreSQL 資料庫叢集必須執行 Amazon Aurora PostgreSQL 10.6 或更新版本。

依預設，此功能為 off，如下列所示：

```
postgres=> SHOW rds.restrict_password_commands;
  rds.restrict_password_commands
--------------------------------
 off
(1 row)
```

如要開啟此功能，請使用自訂參數群組，並將 rds.restrict_password_commands 的設定變更為 1。請務必重新啟動您的 Aurora PostgreSQL 的主資料庫執行個體 ，以使設定生效。

啟用此功能後，下列 SQL 命令需要 rds_password 權限：

```
CREATE ROLE myrole WITH PASSWORD 'mypassword';
CREATE ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword';
ALTER ROLE myrole VALID UNTIL '2023-01-01';
ALTER ROLE myrole RENAME TO myrole2;
```

若密碼使用 MD5 雜湊演算法，則重新命名角色 (ALTER ROLE myrole RENAME TO newname) 也會受到限制。

啟用此功能後，嘗試使用任何這些 SQL 命令，而不使用 rds_password 角色權限會產生下列錯誤：

```
ERROR: must be a member of rds_password to alter passwords
```

建議您僅對專用於密碼管理的一些角色授予 rds_password。若您授予不具 rds_superuser 權限之資料庫使用者 rds_password 權限，則還需要授予其 CREATEROLE 屬性。

確定您驗證用戶端上的密碼需求，例如到期時間和需要的複雜度。若您使用自己的用戶端公用程式進行與密碼相關的變更，則該公用程式必須為 rds_password 的成員並具有 CREATE ROLE 權限。

使用 SCRAM 進行 PostgreSQL 密碼加密

Salted Challenge Response Authentication Mechanism (SCRAM) 是 PostgreSQL 預設訊息摘要 (MD5) 演算法的替代選項，用於加密密碼。SCRAM 身分驗證機制被認為比 MD5 更安全。若要進一步了解這兩種不同的密碼保護方法，請參閱 PostgreSQL 文件中的 Password Authentication (密碼身分驗證)。

建議您使用 SCRAM 作為 Aurora PostgreSQL 資料庫叢集的密碼加密配置，而不是使用 MD5。Aurora PostgreSQL 第 10 版和所有更新的主要和次要版本都支援 SCRAM。這是一種加密的挑戰回應機制，使用 scram-sha-256 演算法進行密碼身分驗證和加密。

您可能需要更新程式庫，用戶端應用程式才會支援 SCRAM。例如，42.2.0 之前的 JDBC 版本不支援 SCRAM。如需詳細資訊，請參閱 PostgreSQL JDBC 驅動程式文件中的 PostgreSQL JDBC Driver (PostgreSQL JDBC 驅動程式)。如需其他 PostgreSQL 驅動程式和 SCRAM 支援的清單，請參閱 PostgreSQL 文件中的 List of drivers (驅動程式清單)。

Aurora PostgreSQL 14 版和更新版本預設支援 scram-sha-256 用於新資料庫叢集的密碼加密。對於這些版本，預設資料庫叢集參數群組 (default.aurora-postgresql14) 將其 password_encryption 值設定為 scram-sha-256。Aurora Serverless v1 不支援 SCRAM。

設定 Aurora PostgreSQL 資料庫叢集 以要求 SCRAM

對於 Aurora PostgreSQL 14.3 和更新版本，您可以要求 Aurora PostgreSQL 資料庫叢集僅接受使用 scram-sha-256 演算法的密碼。

> **Important**
>
> 對於 PostgreSQL 資料庫的現有 RDS 代理，如果您將資料庫驗證修改為僅使用 SCRAM，則該代理會變成無法使用，最多持續 60 秒。若要避免發生此問題，請執行下列其中一項：
>
> - 確定資料庫同時允許 SCRAM 和 MD5 身分驗證。

- 若只要使用 SCRAM 身分驗證，請建立新代理、將應用程式流量遷移至新代理，然後刪除先前與資料庫相關聯的代理。

在對系統進行變更之前，請確保您了解完整過程，如下所示：

- 獲得所有資料庫使用者的所有角色與密碼加密相關資訊。
- 針對控制密碼加密的參數，再次檢查 Aurora PostgreSQL 資料庫叢集的參數設定。
- 如果您的 Aurora PostgreSQL 資料庫叢集使用預設參數群組，則需要建立自訂資料庫叢集參數群組，並將其套用到 Aurora PostgreSQL 資料庫叢集，讓您可以在需要時修改參數。如果您的 Aurora PostgreSQL 資料庫叢集使用自訂參數群組，則您稍後可以視需要在過程中修改必要參數。
- 將 `password_encryption` 參數變更為 `scram-sha-256`。
- 通知所有資料庫使用者他們必須更新密碼。針對您的 `postgres` 帳戶進行相同的動作。系統會使用 scram-sha-256 演算法加密與儲存新密碼。
- 驗證確認使用加密類型將所有密碼加密。
- 如果所有密碼都使用 scram-sha-256，您可以將 `rds.accepted_password_auth_method` 參數從 `md5+scram` 變更為 `scram-sha-256`。

> **Warning**
>
> 在您僅將 `rds.accepted_password_auth_method` 變更為 scram-sha-256 後，則任何具有 `md5` 加密密碼的使用者 (角色) 將無法連線。

準備好為您的 Aurora PostgreSQL 資料庫叢集 要求 SCRAM

對 Aurora PostgreSQL 資料庫叢集進行任何變更之前，請檢查所有現有的資料庫使用者帳戶。另外，請檢查用於密碼的加密類型。您可以使用 `rds_tools` 擴充功能執行這些任務。若要查看哪些 PostgreSQL 版本支援 `rds_tools`，請參閱 Amazon RDS for PostgreSQL 的延伸版本。

獲得資料庫使用者 (角色) 與密碼加密方法清單

1. 使用 `psql` 連線至 Aurora PostgreSQL 資料庫叢集上的主要執行個體 ，如下所示。

```
psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

2. 安裝 rds_tools 擴充功能。

```
postgres=> CREATE EXTENSION rds_tools;
CREATE EXTENSION
```

3. 取得角色和加密清單。

```
postgres=> SELECT * FROM
      rds_tools.role_password_encryption_type();
```

您會看到類似下列的輸出。

```
       rolname        | encryption_type
----------------------+-----------------
 pg_monitor           |
 pg_read_all_settings |
 pg_read_all_stats    |
 pg_stat_scan_tables  |
 pg_signal_backend    |
 lab_tester           | md5
 user_465             | md5
 postgres             | md5
(8 rows)
```

建立自訂資料庫叢集參數群組

> **Note**
>
> 如果您的 Aurora PostgreSQL 資料庫叢集已使用自訂參數群組，則不需要建立新的群組。

如需 Aurora 的參數群組概觀，請參閱 在 Amazon Aurora 中建立資料庫叢集參數群組。

用於密碼的密碼加密類型是在某一個參數 (亦即 password_encryption) 中設定的。Aurora PostgreSQL 資料庫叢集允許的加密則是在另一個參數 (亦即 rds.accepted_password_auth_method) 中設定的。若要從這些預設值變更，則需要您建立自訂資料庫叢集參數群組，並套用到您的叢集。

您也可以使用 AWS 管理主控台 或 RDS API 來建立自訂資料庫叢集參數群組。如需詳細資訊，請參閱 在 Amazon Aurora 中建立資料庫叢集參數群組。

您可以使用資料庫執行個體與自訂參數群組建立關聯。

建立自訂資料庫叢集參數群組

1. 使用 create-db-cluster-parameter-group CLI 命令以建立叢集的自訂參數群組。以下範例使用 aurora-postgresql13 作為此自訂參數群組的來源。

   對於 Linux、macOS 或 Unix：

   ```
   aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-lab-scram-passwords' \
     --db-parameter-group-family aurora-postgresql13  --description 'Custom DB cluster parameter group for SCRAM'
   ```

   在 Windows 中：

   ```
   aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-lab-scram-passwords" ^
     --db-parameter-group-family aurora-postgresql13  --description "Custom DB cluster parameter group for SCRAM"
   ```

   現在您可以將自訂參數群組與叢集建立關聯。

2. 使用 modify-db-cluster CLI 命令將此自訂參數群組套用至您的 Aurora PostgreSQL 資料庫叢集。

   對於 Linux、macOS 或 Unix：

   ```
   aws rds modify-db-cluster --db-cluster-identifier 'your-instance-name' \
           --db-cluster-parameter-group-name "docs-lab-scram-passwords
   ```

   在 Windows 中：

   ```
   aws rds modify-db-cluster --db-cluster-identifier "your-instance-name" ^
           --db-cluster-parameter-group-name "docs-lab-scram-passwords
   ```

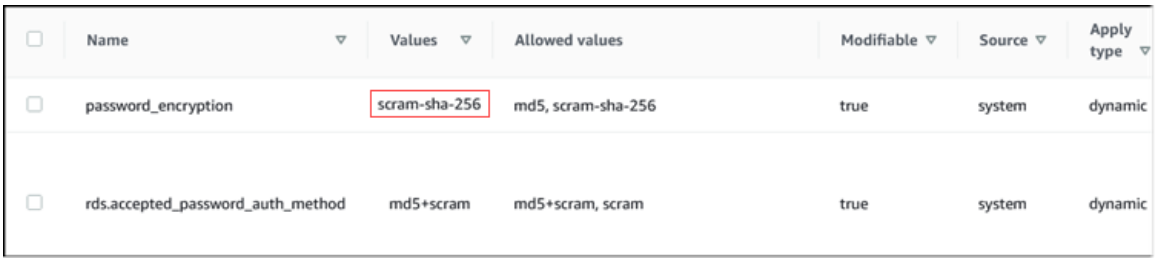
   若要重新同步 Aurora PostgreSQL 資料庫叢集與自訂資料庫叢集參數群組，以及 ，請重新啟動叢集的主要與所有其他執行個體。

設定密碼加密以使用 SCRAM

Aurora PostgreSQL 資料庫叢集使用的密碼加密機制是在資料庫叢集參數群組的 `password_encryption` 參數中設定的。允許的值為未設定、`md5` 或 `scram-sha-256`。預設值視 Aurora PostgreSQL 版本而定，如下所示：

- Aurora PostgreSQL 14 – 預設值為 `scram-sha-256`
- Aurora PostgreSQL 13 – 預設值為 `md5`

透過將自訂資料庫叢集參數群組連接至 Aurora PostgreSQL 資料庫叢集，您可以修改密碼加密參數的值。



Name	Values	Allowed values	Modifiable	Source	Apply type
password_encryption	scram-sha-256	md5, scram-sha-256	true	system	dynamic
rds.accepted_password_auth_method	md5+scram	md5+scram, scram	true	system	dynamic

將密碼加密設定變更為 scram-sha-256

- 將密碼加密的值變更為 scram-sha-256，如下所示。變更會立即套用，因為參數是動態的，因此不需要重新啟動即可使變更生效。

  對於 Linux、macOS 或 Unix：

  ```
  aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name \
    'docs-lab-scram-passwords' --parameters
   'ParameterName=password_encryption,ParameterValue=scram-
  sha-256,ApplyMethod=immediate'
  ```

  在 Windows 中：

  ```
  aws rds modify-db-parameter-group --db-parameter-group-name ^
    "docs-lab-scram-passwords" --parameters
   "ParameterName=password_encryption,ParameterValue=scram-
  sha-256,ApplyMethod=immediate"
  ```

將使用者角色的密碼遷移至 SCRAM

您可以將使用者角色的密碼遷移至 SCRAM，如下說明：

將資料庫使用者 (角色) 密碼從 MD5 遷移至 SCRAM

1. 以管理員使用者身分 (預設使用者名稱 `postgres`) 登入，如下所示。

```
psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

2. 使用下列命令檢查 RDS for PostgreSQL 資料庫執行個體上 `password_encryption` 參數的設定。

```
postgres=> SHOW password_encryption;
 password_encryption
---------------------
 md5
 (1 row)
```

3. 將此參數的值變更為 scram-sha-256。如需詳細資訊，請參閱設定密碼加密以使用 SCRAM。
4. 再次檢查該值以確定它現在已設定為 `scram-sha-256`，如下所示。

```
postgres=> SHOW password_encryption;
 password_encryption
---------------------
 scram-sha-256
 (1 row)
```

5. 通知所有資料庫使用者變更自己的密碼。務必亦為帳戶 `postgres` (具有 `rds_superuser` 權限的資料庫使用者) 變更您自己的密碼。

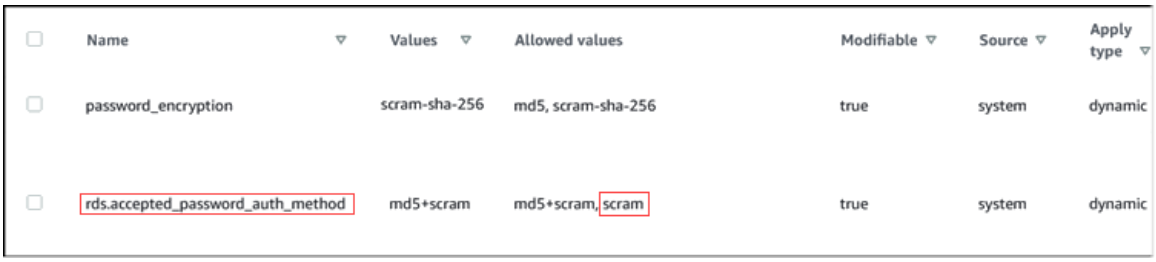
```
labdb=> ALTER ROLE postgres WITH LOGIN PASSWORD 'change_me';
ALTER ROLE
```

6. 對 Aurora PostgreSQL 資料庫叢集上所有的資料庫重複此程序。

變更參數為需要 SCRAM

這是程序中的最後一個步驟。在以下程序中進行變更後，仍然使用 md5 將密碼加密的任何使用者帳戶 (角色) 將無法登入 Aurora PostgreSQL 資料庫叢集。

rds.accepted_password_auth_method 指定 Aurora PostgreSQL 資料庫叢集在登入程序期間接受的使用者密碼加密方法。預設值為 md5+scram，這意味著任何一種方法都接受。在下圖中，您可以找到此參數的預設設定。



Name	Values	Allowed values	Modifiable	Source	Apply type
password_encryption	scram-sha-256	md5, scram-sha-256	true	system	dynamic
rds.accepted_password_auth_method	md5+scram	md5+scram, scram	true	system	dynamic

此參數的允許值為 md5+scram 或僅 scram。將此參數值變更為 scram 會使此成為一個需求。

將參數值變更為需要對密碼進行 SCRAM 身分驗證

1. 確認 Aurora PostgreSQL 資料庫叢集上所有資料庫的所有資料庫使用者密碼使用 scram-sha-256 進行密碼加密。若要這麼做，請向 rds_tools 查詢角色 (使用者) 和加密類型，如下所示。

```
postgres=> SELECT * FROM rds_tools.role_password_encryption_type();
  rolname        | encryption_type
  ---------------------+-----------------
  pg_monitor           |
  pg_read_all_settings |
  pg_read_all_stats    |
  pg_stat_scan_tables  |
  pg_signal_backend    |
  lab_tester           | scram-sha-256
  user_465             | scram-sha-256
  postgres             | scram-sha-256
  ( rows)
```

2. 針對您的 Aurora PostgreSQL 資料庫叢集中所有的資料庫執行個體重複此查詢。

   如果所有密碼都使用 scram-sha-256，您可以繼續進行。

3. 將接受的密碼身分驗證值變更為 scram-sha-256，如下所示。

   對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-lab-scram-passwords' \
```

```
  --parameters
 'ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

在 Windows 中：

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-
lab-scram-passwords" ^
  --parameters
 "ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

使用 SSL/TLS 保護 Aurora PostgreSQL 資料的安全

對於 Aurora PostgreSQL 資料庫叢集，Amazon RDS 支援 Secure Sockets Layer (SSL) 和 Transport Layer Security (TLS) 加密。您可以使用 SSL/TLS，將應用程式與 Aurora PostgreSQL 資料庫叢集之間的連線加密。您也可以強制 Aurora PostgreSQL 資料庫叢集的所有連線都使用 SSL/TLS。Amazon Aurora PostgreSQL 現已支援 Transport Layer Security (TLS) 版本 1.1 和 1.2。建議使用 TLS 1.2 進行加密連線。我們從以下版本的 Aurora PostgreSQL 新增了對 TLSv1.3 的支援：

- 15.3 版和所有更新版本
- 14.8 版和更新的 14 版本
- 13.11 版和更新的 13 版本
- 12.15 版和更新的 12 版
- 11.20 版和更新的 11 版本

如需 SSL/TLS 支援和 PostgreSQL 資料庫的一般資訊，請參閱 PostgreSQL 文件中的 SSL 支援。如需透過 JDBC 使用 SSL/TLS 連線的相關資訊，請參閱 PostgreSQL 文件中的設定用戶端。

主題

- 需要使用 SSL/TLS 連線至 Aurora PostgreSQL 資料庫叢集
- 判斷 SSL/TLS 連線狀態
- 為 Aurora PostgreSQL 資料庫叢集的連線設定密碼套件

Aurora PostgreSQL 的所有 AWS 區域都提供 SSL/TLS 支援。建立資料庫叢集時，Amazon RDS 會為您的 Aurora PostgreSQL 資料庫叢集建立 SSL/TLS 憑證。如果您啟用 SSL/TLS 憑證驗證，則 SSL/TLS 憑證會包含資料庫叢集端點，當作 SSL/TLS 憑證的通用名稱 (CN)，以防範詐騙攻擊。

透過 SSL/TLS 連線至 Aurora PostgreSQL 資料庫叢集

1. 下載憑證。

   如需有關下載憑證的詳細資訊，請參閱使用 SSL/TLS 加密與資料庫叢集的連線。

2. 將憑證匯入作業系統。

3. 透過 SSL/TLS 連線至 Aurora PostgreSQL 資料庫叢集。

   當您使用 SSL/TLS 連線時，用戶端可以選擇是否驗證憑證鏈。如果您的連線參數指定 `sslmode=verify-ca` 或 `sslmode=verify-full`，則用戶端在信任存放區必須有 RDS CA 憑證，或在連線 URL 中必須參考這些憑證。此需求是為了驗證用於簽署資料庫憑證的憑證鏈。

   當用戶端 (例如 psql 或 JDBC) 設有 SSL/TLS 支援時，依預設，用戶端會先嘗試使用 SSL/TLS 連線至資料庫。如果用戶端無法使用 SSL/TLS 連線，則回復為不使用 SSL/TLS 連線。根據預設，JDBC 和 libpq 型用戶端的 `sslmode` 選項設定為 `prefer`。

   使用 `sslrootcert` 參數來參考憑證，例如 `sslrootcert=rds-ssl-ca-cert.pem`。

以下是使用 psql 連接至 Aurora PostgreSQL 資料庫叢集的範例。

```
$ psql -h testpg.cdhmuqifdpib.us-east-1.rds.amazonaws.com -p 5432 \
    "dbname=testpg user=testuser sslrootcert=rds-ca-2015-root.pem sslmode=verify-full"
```

需要使用 SSL/TLS 連線至 Aurora PostgreSQL 資料庫叢集

若需要連至 Aurora PostgreSQL 資料庫叢集的 SSL/TLS 連線，請使用 `rds.force_ssl` 參數。

- 若需要 SSL/TLS 連線，請將 `rds.force_ssl` 參數值設為 1 (開啟)。
- 若要關閉所需的 SSL/TLS 連線，請將 `rds.force_ssl` 參數值設為 0 (關閉)。

此參數的預設值取決於 Aurora PostgreSQL 版本：

- 對於 Aurora PostgreSQL 第 17 版及更新版本：預設值為 1 (開啟)。
- 對於 Aurora PostgreSQL 第 16 版及更舊版本：預設值為 0 (關閉)。

> **Note**
>
> 當您從 Aurora PostgreSQL 第 16 版或更舊版本執行升級至第 17 版或更新版本的主要版本升級時，參數的預設值會從 0 (關閉) 變更為 1 (開啟)。此變更可能會導致未針對 SSL 設定的應用程式連線能力失敗。您可以將此參數設定為 0 (關閉)，以還原至先前的預設行為。

如需處理參數的詳細資訊，請參閱 Amazon Aurora 的參數群組。

更新 `rds.force_ssl` 參數也會將 PostgreSQL `ssl` 參數設為 1 (開啟)，並修改資料庫叢集的 `pg_hba.conf` 檔案，以支援新的 SSL/TLS 組態。

當資料庫叢集的 `rds.force_ssl` 參數設為 1 時，您在連線時會看到類似如下的輸出，表示現在需要使用 SSL/TLS：

```
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql (9.3.12, server 9.4.4)
WARNING: psql major version 9.3, server major version 9.4.
Some psql features might not work.
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

判斷 SSL/TLS 連線狀態

當您連接至資料庫叢集時，登入橫幅中會顯示連線的加密狀態。

```
Password for user master:
psql (9.3.12)
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

您也可以載入 `sslinfo` 擴充功能，然後呼叫 `ssl_is_used()` 函數，以判斷是否正在使用 SSL/TLS。如果連線正在使用 SSL/TLS，此函數會傳回 `t`，否則傳回 `f`。

```
postgres=> create extension sslinfo;
CREATE EXTENSION

postgres=> select ssl_is_used();
 ssl_is_used
---------
t
(1 row)
```

您可以使用 `select ssl_cipher()` 命令來決定 SSL/TLS 密碼：

```
postgres=> select ssl_cipher();
ssl_cipher
--------------------
DHE-RSA-AES256-SHA
(1 row)
```

如果您啟用 `set rds.force_ssl` 並重新啟動資料庫叢集，則非 SSL 連線會遭到拒絕並傳回下列訊息：

```
$ export PGSSLMODE=disable
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql: FATAL: no pg_hba.conf entry for host "host.ip", user "someuser", database
 "postgres", SSL off
$
```

如需 `sslmode` 選項的資訊，請參閱 PostgreSQL 文件中的資料庫連線控制函數。

為 Aurora PostgreSQL 資料庫叢集的連線設定密碼套件

透過使用可設定的密碼套件，您可以更進一步控制資料庫連線的安全性。您可以指定要允許的密碼套件清單，用以保護用戶端與資料庫之間的 SSL/TLS 連線的安全。您可以使用可設定的密碼套件來控制資料庫伺服器接受的連線加密。這樣做有助於避免使用不安全或已作廢的密碼。

Aurora PostgreSQL 11.8 版及更高版本支援可設定的密碼套件。

要指定用於加密連線的許可密碼列表，請修改 `ssl_ciphers` 叢集參數。使用 AWS CLI、 或 RDS API AWS 管理主控台，將 `ssl_ciphers` 參數設定為叢集參數群組中逗號分隔密碼值的字串。要設定叢集參數，請參閱 在 Amazon Aurora 中修改資料庫叢集參數群組中的參數。

下表顯示有效 Aurora PostgreSQL 引擎版本支援的密碼。

Aurora PostgreSQL 引擎版本	受支援的密碼	TLS 1.1	TLS 1.2	TLS 1.3
9.6、10.20 和更低版本，11.15 和更低版本，12.10 和更低版本，13.6 和更低版本	DHE-RSA-AES128-SHA	是	否	否
	DHE-RSA-AES128-SHA256	否	是	否
	DHE-RSA-AES128-GCM-SHA256	否	是	否
	DHE-RSA-AES256-SHA	否	是	否
	DHE-RSA-AES256-SHA256	否	是	否
	DHE-RSA-AES256-GCM-SHA384	否	是	否
	ECDHE-ECDSA-AES256-SHA	否	是	否
	ECDHE-ECDSA-AES256-GCM-SHA384	否	是	否
	ECDHE-RSA-AES256-SHA384	否	是	否
	ECDHE-RSA-AES128-SHA			
	ECDHE-RSA-AES128-SHA256			

Aurora PostgreSQL 引擎版本	受支援的密碼	TLS 1.1	TLS 1.2	TLS 1.3
	ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES256-SHA ECDHE-RSA-AES256-GCM-SHA384			

Aurora PostgreSQL 引擎版本	受支援的密碼	TLS 1.1	TLS 1.2	TLS 1.3
10.21、11.16、12.11、13.7、14.3 和 14.4	ECDHE-RSA-AES128-SHA	是	是	否
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	否	是	否
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	是	是	否
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	是	是	否
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	否	是	否
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	是	是	否
	TLS_ECDHE_RSA_WITH	否	是	否

Aurora PostgreSQL 引擎版本	受支援的密碼	TLS 1.1	TLS 1.2	TLS 1.3
	_AES_256_ GCM_SHA384 TLS_RSA_W ITH_AES_2 56_GCM_SHA384 TLS_RSA_W ITH_AES_2 56_CBC_SHA TLS_RSA_W ITH_AES_1 28_GCM_SHA256 TLS_RSA_W ITH_AES_1 28_CBC_SHA TLS_ECDHE _RSA_WITH _CHACHA20 _POLY1305_SHA256			

Aurora PostgreSQL 引擎版本	受支援的密碼	TLS 1.1	TLS 1.2	TLS 1.3
10.22、11.17、12.12、13.8、14.5 和 15.2	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	是	是	否
		否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	否	是	否
		是	是	否
		否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	是	是	否
		否	是	否
		否	是	否
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	是	是	否
		否	是	否
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	否	是	否
		是	是	否
		否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	是	是	否
		是	是	否
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256			

Aurora PostgreSQL 引擎版本	受支援的密碼	TLS 1.1	TLS 1.2	TLS 1.3
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA TLS_ECDHE _RSA_WITH _AES_256_ GCM_SHA384 TLS_RSA_W ITH_AES_2 56_GCM_SHA384 TLS_RSA_W ITH_AES_2 56_CBC_SHA TLS_RSA_W ITH_AES_1 28_GCM_SHA256 TLS_RSA_W ITH_AES_1 28_CBC_SHA256 TLS_RSA_W ITH_AES_1 28_CBC_SHA TLS_ECDHE _RSA_WITH _CHACHA20 _POLY1305_SHA256			

Aurora PostgreSQL 引擎版本	受支援的密碼	TLS 1.1	TLS 1.2	TLS 1.3
11.20、12.15、13.11、14.8、15.3、16.1 及更新版本	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	是	是	否
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	否	是	否
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	是	是	否
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	是	是	否
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	否	否	是

Aurora PostgreSQL 引擎版本	受支援的密碼	TLS 1.1	TLS 1.2	TLS 1.3
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_256_CBC_SHA TLS_RSA_WITH_AES_128_GCM_SHA256 TLS_RSA_WITH_AES_128_CBC_SHA256 TLS_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256			

Aurora PostgreSQL 引擎版本	受支援的密碼	TLS 1.1	TLS 1.2	TLS 1.3
	TLS_AES_256_GCM_SHA384			

您也可以使用 describe-engine-default-cluster-parameters CLI 命令決定特定參數群組系列目前支援哪些密碼套件。下列範例顯示如何取得適用於 Aurora PostgreSQL 11 的 `ssl_cipher` 叢集參數的允許值。

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-
postgresql11

    ...some output truncated...
 {
  "ParameterName": "ssl_ciphers",
  "Description": "Sets the list of allowed TLS ciphers to be used on secure
 connections.",
  "Source": "engine-default",
  "ApplyType": "dynamic",
  "DataType": "list",
  "AllowedValues": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,
  ECDHE-RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-
SHA256,ECDHE-RSA-AES256-SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-
SHA384,TLS_RSA_WITH_AES_256_GCM_SHA384,

 TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA256,T

 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AE
  "IsModifiable": true,
  "MinimumEngineVersion": "11.8",
  "SupportedEngineModes": [
   "provisioned"
  ]
 },
    ...some output truncated...
```

`ssl_ciphers` 參數預設為所有允許的密碼套件。如需有關密碼的詳細資訊，請參閱 PostgreSQL 文件中的 ssl_cipher 變數。

搭配 Aurora PostgreSQL 使用動態遮罩

動態資料遮罩是一項安全功能，可透過控制資料在查詢時對使用者顯示的方式，保護 Aurora PostgreSQL 資料庫中的敏感資料。Aurora 透過 pg_columnmask延伸模組實作它。pg_columnmask提供資料欄層級資料保護，以補充 PostgreSQL 的原生資料列層級安全性和精細存取控制機制。

使用 pg_columnmask，您可以建立遮罩政策，根據使用者角色判斷資料可見性。當使用者使用遮罩政策查詢資料表時，Aurora PostgreSQL 會根據使用者的角色和政策權重，在查詢時間套用適當的遮罩函數。基礎資料在儲存中保持不變。

pg_columnmask 支援下列功能：

- 內建和自訂遮罩函數 – 針對電子郵件和文字遮罩等常見模式使用預先建置的函數，或建立您自己的自訂函數，透過 SQL 型遮罩政策保護敏感資料 (PII)。
- 多個遮罩策略 – 完全隱藏資訊、以萬用字元取代部分值，或定義自訂遮罩方法。
- 政策優先順序 – 為單一資料欄定義多個政策。使用權重來判斷在將多個政策套用至資料欄時應使用哪些遮罩政策。Aurora PostgreSQL 會根據權重和使用者角色成員資格套用政策。

pg_columnmask 適用於 Aurora PostgreSQL 16.10 版和更新版本，以及 17.6 版和更新版本。可免費使用。

開始使用動態遮罩

若要動態遮罩資料，您可以在資料庫中安裝 pg_columnmask 擴充功能，並為資料表建立遮罩政策。設定程序包含先決條件驗證、延伸安裝、角色組態、政策建立和驗證測試。

延伸模組安裝和組態

使用 RDS 主控台查詢編輯器或 PostgreSQL 用戶端連線至 Aurora PostgreSQL 叢集，例如具有 rds_superuser（主要使用者）憑證的 psql。

執行延伸項目建立命令以啟用 pg_columnmask 功能：

```
CREATE EXTENSION pg_columnmask;
```

此命令會安裝 pg_columnmask 延伸模組、建立必要的目錄資料表，並註冊內建遮罩函數。延伸模組安裝是資料庫特定的，這表示您必須在需要 功能的每個資料庫中分別安裝。

> **Note**
>
> 在安裝此延伸模組之前進行的連線仍會顯示未遮罩的資料。關閉並重新連線以修正此問題。

檢查可用的遮罩函數來驗證延伸模組安裝：

```
SELECT proname FROM pg_proc
    WHERE pronamespace = 'pgcolumnmask'::regnamespace AND proname LIKE 'mask_%';
    proname
--------Output --------
 mask_email
 mask_text
 mask_timestamp
(3 rows)
```

管理資料遮罩政策的程序

您可以使用 pg_columnmask 擴充功能提供的程序來管理遮罩政策。若要建立、修改或捨棄遮罩政策，您必須具有下列其中一個權限：

- 您要建立pg_columnmask政策之資料表的擁有者。
- 的成員rds_superuser。
- pgcolumnmask.policy_admin_rolname 參數設定pg_columnmask的政策管理員角色成員。

下列命令會建立用於後續區段的資料表：

```
CREATE TABLE public.customers (
    id SERIAL PRIMARY KEY,
    name TEXT,
    phone TEXT,
    address TEXT,
    email TEXT
);
```

CREATE_MASKING_POLICY

下列程序會為使用者資料表建立新的遮罩政策：

語法

```
create_masking_policy(
    policy_name,
    table_name,
    masking_expressions,
    roles,
    weight)
```

Arguments (引數)

參數	Datatype	Description
policy_name	NAME	遮罩政策的名稱。每個資料表必須是唯一的。
table_name	REGCLASS	要套用遮罩政策之資料表的合格/不合格名稱或 oid。
masking_expressions	JSONB	包含資料欄名稱和遮罩函數對的 JSON 物件。每個索引鍵都是資料欄名稱，其值是要套用至該資料欄的遮罩表達式。
roles	NAME【】	此遮罩政策套用的角色。預設為 PUBLIC。
weight	INT	遮罩政策的權重。當多個政策適用於指定使用者的查詢時，具有最高權重（較高的整數）的政策將套用至每個遮罩的資料欄。 預設值為 0。資料表上沒有任何兩個遮罩政策可以具有相同的 Wight。

傳回類型

無

Example建立遮罩政策以遮罩test_user角色的電子郵件欄：

```
CALL pgcolumnmask.create_masking_policy(
    'customer_mask',
    'public.customers',
    JSON_OBJECT('{
        "email", "pgcolumnmask.mask_email(email)"
    }')::JSONB,
    ARRAY['test_user'],
    100
);
```

ALTER_MASKING_POLICY

此程序會修改現有的遮罩政策。 ALTER_MASKING_POLICY可以修改政策遮罩表達式、政策套用的一組角色，以及遮罩政策的權重。省略其中一個參數時，政策的對應部分保持不變。

語法

```
alter_masking_policy(
    policy_name,
    table_name,
    masking_expressions,
    roles,
    weight)
```

Arguments (引數)

參數	Datatype	Description
policy_name	NAME	遮罩政策的現有名稱。
table_name	REGCLASS	包含遮罩政策之資料表的合格/不合格名稱 oid。
masking_expressions	JSONB	包含資料欄名稱和遮罩函數對或 NULL 的新 JSON 物件。
roles	NAME【】	套用此遮罩政策的新角色清單，否則為 NULL。

參數	Datatype	Description
weight	INT	遮罩政策或 NULL 的新權重。

傳回類型

無

Example將分析師角色新增至現有遮罩政策，而不變更其他政策屬性。

```
CALL pgcolumnmask.alter_masking_policy(
    'customer_mask',
    'public.customers',
    NULL,
    ARRAY['test_user', 'analyst'],
    NULL
);

-- Alter the weight of the policy without altering other details
CALL pgcolumnmask.alter_masking_policy(
    'customer_mask',
    'customers',
    NULL,
    NULL,
    4
);
```

DROP_MASKING_POLICY

此程序會移除現有的遮罩政策。

語法

```
drop_masking_policy(
        policy_name,
        table_name)
```

Arguments (引數)

參數	Datatype	Description
policy_name	NAME	遮罩政策的現有名稱。
table_name	REGCLASS	包含遮罩政策之資料表的合格/不合格名稱 oid。

傳回類型

無

Example捨棄遮罩政策 customer_mask

```
-- Drop a masking policy
    CALL pgcolumnmask.drop_masking_policy(
        'customer_mask',
        'public.customers',
    );
```

遮罩政策 DDL 程序中的逸出識別符

使用引號識別符建立資料遮罩政策時，需要適當的逸出，以確保正確的物件參考和政策應用程式。若要在pg_columnmask遮罩政策管理程序中使用引號識別符：

- 政策名稱 – 必須以雙引號括住。
- 資料表名稱 – 結構描述名稱和資料表名稱都必須視需要個別以雙引號括住。
- 遮罩表達式 – 遮罩表達式中的資料欄和函數名稱必須以雙引號括住，且必須使用反斜線逸出引號本身。
- 角色 – 會自動引用角色名稱的陣列。角色名稱應該完全符合 中顯示的名稱，pg_roles包括區分大小寫。

Example逸出和引號語法的

此範例顯示為使用混合大小寫名稱或在 Aurora PostgreSQL 中需要引用識別符的資料表、資料欄、函數和角色建立遮罩政策時，適當的逸出和引用語法。

```
-- Create a table and columns with mixed case name
CREATE TABLE public."Employees" (
```

```
    "Name" TEXT,
    "Email" TEXT,
    ssn VARCHAR(20)
);

-- Create a role with mixed case name
CREATE ROLE "Masked_user";

-- Create a function with mixed case name
CREATE OR REPLACE FUNCTION public."MaskEmail"(text)
    RETURNS character varying
    LANGUAGE plpgsql
    IMMUTABLE PARALLEL SAFE
    AS $$ BEGIN
        RETURN 'XXXXXXXX'::text;
    END $$;

-- Now use these objects with mixed case names in
-- masking policy management procedures
CALL pgcolumnmask.create_masking_policy(
    '"Policy1"',  -- policy name should be surrounded with double quotes for quoting
    'public."Employees"', -- table and schema name should be individually
                          -- surrounded with double quotes for quoting
    JSON_OBJECT('{
        "\"Email\"", "\"MaskEmail\"(\"Email\")"
    }')::JSONB, -- masking expression should have double quotes around function names
                -- and columns names etc when needed. Also the double quotes itself
                -- should be escaped using \ (backslash) since this is a JSON string
    ARRAY['Masked_user'], -- Rolename do not need quoting
                          -- (this behaviour may change in future release)
    100
);

SELECT * FROM pgcolumnmask.pg_columnmask_policies
    WHERE tablename = 'Employees';
-[ RECORD 1 ]-----+-------------------------------------
schemaname        | public
tablename         | Employees
policyname        | Policy1
roles             | {Masked_user}
masked_columns    | {Email}
masking_functions | {"(\"MaskEmail\"(\"Email\"))::text"}
weight            | 100
```

管理檢視

您可以使用可公開存取的pgcolumnmask.pg_columnmask_policies管理檢視來檢閱所有pg_columnmask政策。下列資訊可透過此檢視取得。檢視只會傳回目前使用者擁有的遮罩政策。

欄名稱	資料類型	Description
結構描述名稱	NAME	政策所連接關係的結構描述
資料表名稱	NAME	政策所連接關係的名稱
policyname	NAME	遮罩政策的名稱，所有遮罩政策都有唯一的名稱
roles	TEXT【】	政策適用的角色。
masked_columns	TEXT【】	遮罩的資料欄
masking_functions	TEXT【】	遮罩函數
Weight	INT	附加政策的權重

預先定義的資料遮罩函數

pg_columnmask extension 提供以 C 語言寫入的內建公用程式函數（以加快執行速度），可用於pg_columnmask政策的遮罩表達式。

mask_text

使用可設定的可見性選項遮罩文字資料的 函數。

Arguments (引數)

參數	Datatype	Description
input	TEXT	要遮罩的原始文字字串
mask_char	CHAR(1)	用於遮罩的字元（預設：'X')

參數	Datatype	Description
visible_prefix	INT	在輸入文字開頭的字元數將保持非遮罩 （ 預設值：0)
visible_suffix	INT	輸入文字結尾的字元數將保持非遮罩 （ 預設值：0)
use_hash_mask	BOOLEAN	若為 TRUE， 會使用以雜湊為基礎的遮罩，而非 mask_char （ 預設值：FALSE)

Example使用不同的遮罩選項

使用預設的「X」字元遮罩整個輸入字串

```
postgres=> SELECT pgcolumnmask.mask_text('Hello World');
  mask_text
-------------
 XXXXXXXXXXX
```

使用 mask_char 引數來遮罩使用不同字元的文字輸入

```
postgres=> SELECT pgcolumnmask.mask_text('Hello World', '*');
  mask_text
-------------
 ***********
```

使用 visible_prefix和 visible_suffix 參數控制在文字的開頭和結尾保持未遮罩的字元數量

```
postgres=> SELECT pgcolumnmask.mask_text('Hello World', '*', 5, 1);
  mask_text
-------------
 Hello*****d
```

當 use_hash_mask 為 true 時，系統會忽略使用隨機字元mask_char引數遮罩輸入字串，visible_prefix但仍會visible_suffix接受

```
postgres=> SELECT pgcolumnmask.mask_text('Hello World', '*', 2, 2, true);
  mask_text
```

```
-------------
 Hex36d0Hild
```

mask_timestamp

參數	Datatype	Description
ts_to_mask	TIMESTAMP	要遮罩的原始時間戳記
mask_part	TEXT	指定要遮罩的時間戳記部分（預設值：'all') 有效值：'year'、'month'、'day'、'hour'、'minute'、'second'、'all'
mask_value	TIMESTAMP	用於遮罩的時間戳記值（預設值：'1900-01-01 00：00：00')

Example使用 mask_timestamps

這些範例示範對預設值的完整時間戳記遮罩、特定時間戳記元件的部分遮罩（僅限年份），以及使用自訂取代值遮罩。

完全遮罩預設時間戳記的輸入值

```
postgres=> SELECT pgcolumnmask.mask_timestamp('2023-06-15 14:30:00');
   mask_timestamp
---------------------
 1900-01-01 00:00:00
```

僅遮罩範例中的某個部分時間戳記，僅限年份

```
postgres=> SELECT pgcolumnmask.mask_timestamp('2023-06-15 14:30:00', 'year');
   mask_timestamp
---------------------
 1900-06-15 14:30:00
```

若要變更時間戳記的遮罩值，請使用 mask_value引數

```
postgres=> SELECT pgcolumnmask.mask_timestamp('2023-06-15 14:30:00', 'all', '2012-12-12
 12:12:12');
```

```
   mask_timestamp
---------------------
 2012-12-12 12:12:12
```

mask_timestamp

在保留電子郵件結構時遮罩電子郵件地址的 函數。

參數	Datatype	Description
input	TEXT	要遮罩的原始電子郵件地址
mask_char	CHAR(1)	用於遮罩的字元 （預設：'X')
mask_local	BOOLEAN	如果是 TRUE， 會遮罩電子郵件的本機部分 (@ 之前） （預設值：TRUE)
mask_domain	BOOLEAN	如果為 TRUE， 會遮罩電子郵件的網域部分 (@ 之後） （預設：TRUE)

Example使用 mask_email

這些範例示範完整的電子郵件遮罩、自訂遮罩字元，以及電子郵件地址的本機部分或網域部分的選擇性遮罩。

完成遮罩

```
postgres=> SELECT pgcolumnmask.mask_email('user@example.com');
    mask_email
------------------
 XXXX@XXXXXXX.com
```

使用 mask_char 變更用於遮罩的字元

```
postgres=> SELECT pgcolumnmask.mask_email('user@example.com', '*');
    mask_email
------------------
 ****@*******.com
```

使用 mask_local和 mask_domain 控制本機和網域上的遮罩

```
postgres=> SELECT pgcolumnmask.mask_email('user@example.com', '*', true, false);
    mask_email
------------------
 ****@example.com

postgres=> SELECT pgcolumnmask.mask_email('user@example.com', '*', false, true);
    mask_email
------------------
 user@*******.com
```

在end-to-end工作流程中實作 pg_columnmask

本節示範pg_columnmask將範例員工資料表與敏感資料搭配使用的完整實作。您將了解如何建立自訂遮罩函數、為各種角色（互動、支援、分析師）定義具有不同權重層級的多個遮罩政策，以及觀察具有單一或多個角色成員資格的使用者如何看到不同層級的遮罩資料。這些範例也涵蓋具有 RETURNING 子句的 DML 陳述式中的遮罩行為、資料表與檢視的觸發，以及政策管理操作，包括重新命名、變更權重和清除。

1. 使用一些敏感資料建立範例資料表：

```
CREATE SCHEMA hr;

CREATE TABLE hr.employees (
    id INT PRIMARY KEY,
    name TEXT NOT NULL,
    email TEXT,
    ssn TEXT,
    salary NUMERIC(10,2)
 );

INSERT INTO hr.employees VALUES
    (1, 'John Doe', 'john.doe@example.com', '123-45-6789', 50000.00),
    (2, 'Jane Smith', 'jane.smith@example.com', '987-65-4321', 60000.00);
```

2. 建立自訂遮罩函數：

```
CREATE OR REPLACE FUNCTION public.mask_ssn(ssn TEXT)
    RETURNS TEXT AS $$
    BEGIN
        RETURN 'XXX-XX-' || RIGHT(ssn, 4);
```

```
    END;
    $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION public.mask_salary(salary NUMERIC, multiplier NUMERIC
 DEFAULT 0.0)
    RETURNS NUMERIC AS $$
    BEGIN
        RETURN salary * multiplier;
    END;
    $$ LANGUAGE plpgsql;
```

3. 根據使用者角色建立具有不同遮罩層級的多個政策：

```
-- Create different roles
CREATE ROLE analyst_role;
CREATE ROLE support_role;
CREATE ROLE intern_role;

GRANT USAGE ON SCHEMA hr TO analyst_role, support_role, intern_role;
GRANT SELECT ON hr.employees TO analyst_role, support_role, intern_role;
-----------------------------------------------------------------------

-- Low-Weight Policy (Intern)
CALL pgcolumnmask.create_masking_policy(
    'employee_mask_strict',
    'hr.employees',
    JSON_BUILD_OBJECT('name', 'pgcolumnmask.mask_text(name, ''*'')',
                      'email', 'pgcolumnmask.mask_email(email)',
                      'ssn', 'pgcolumnmask.mask_text(ssn, ''*'')',
                      'salary', 'public.mask_salary(salary)')::JSONB,
    ARRAY['intern_role'],
    10  -- Lowest weight
);

-----------------------------------------------------------------------
-- Medium-Weight Policy (Support)
CALL pgcolumnmask.create_masking_policy(
    'employee_mask_moderate',
    'hr.employees',
    JSON_BUILD_OBJECT('email', 'pgcolumnmask.mask_email(email, ''#'')',
                      'ssn', 'public.mask_ssn(ssn)',
                      'salary', 'public.mask_salary(salary)')::JSONB,
    ARRAY['support_role'],
    50   -- Medium weight
```

```
);

-------------------------------------------------------------------------
-- High-Weight Policy (Analyst)
CALL pgcolumnmask.create_masking_policy(
    'employee_mask_light',
    'hr.employees',
    JSON_BUILD_OBJECT('ssn', 'public.mask_ssn(ssn)',
                      'salary', 'public.mask_salary(salary, 0.9)')::JSONB,
    ARRAY['analyst_role'],
    100   -- Highest weight
);
```

4. 下列範例示範不同使用者如何根據其角色成員資格和政策權重來查看資料。

```
-- Create users
CREATE USER sarah_intern;
GRANT intern_role TO sarah_intern;

CREATE USER lisa_support;
GRANT support_role TO lisa_support;

CREATE USER mike_analyst;
GRANT analyst_role TO mike_analyst;

CREATE USER ethan_support_intern;
GRANT support_role, intern_role TO ethan_support_intern;

CREATE USER john_analyst_intern;
GRANT analyst_role, intern_role TO john_analyst_intern;
```

作為內部（最嚴格的遮罩）：

```
SET ROLE sarah_intern;

SELECT * FROM hr.employees;
 id |    name    |         email          |     ssn     | salary
----+------------+------------------------+-------------+--------
  1 | ********   | XXXXXXXX@XXXXXXX.com   | *********** |   0.00
  2 | ********** | XXXXXXXXXX@XXXXXXX.com | *********** |   0.00
```

身為支援使用者（中度遮罩）：

```
SET ROLE lisa_support;

SELECT * FROM hr.employees;
 id |    name    |         email          |     ssn     | salary
----+------------+------------------------+-------------+--------
  1 | John Doe   | ########@#######.com   | XXX-XX-6789 |   0.00
  2 | Jane Smith | ##########@#######.com | XXX-XX-4321 |   0.00
```

身為分析師（最輕量的遮罩）：

```
SET ROLE mike_analyst;

SELECT * FROM hr.employees;
 id |    name    |         email          |     ssn     |  salary
----+------------+------------------------+-------------+----------
  1 | John Doe   | john.doe@example.com   | XXX-XX-6789 | 45000.00
  2 | Jane Smith | jane.smith@example.com | XXX-XX-4321 | 54000.00
```

作為 ethan_support_intern 使用者，該使用者同時是 intern 和支援使用者：

```
SET ROLE ethan_support_intern;

-- masking policies appliable to this user: employee_mask_strict and
 employee_mask_moderate
-- id : unmasked because no masking policy appliable on ethan_support_intern
--            masks these columns
-- name : masked because of employee_mask_strict policy
-- email, ssn, salary : both employee_mask_strict and employee_mask_moderate mask
 these columns
--                      but employee_mask_moderate will be use because of higher
 weight

SELECT * FROM hr.employees;
 id |    name    |         email          |     ssn     | salary
----+------------+------------------------+-------------+--------
  1 | ********   | ########@#######.com   | XXX-XX-6789 |   0.00
  2 | ********** | ##########@#######.com | XXX-XX-4321 |   0.00
```

作為 john_analyst_intern，它是 intern 和分析師：

```
SET ROLE john_analyst_intern;

-- masking policies appliable to this user: employee_mask_strict and
 employee_mask_light
-- id : unmasked because no masking policy appliable on john_analyst_intern
--            masks these columns
-- name, email : masked because of employee_mask_strict
-- ssn, salary : both employee_mask_strict and employee_mask_light mask these
 columns
--               but employee_mask_light will be use because of higher weight

SELECT * FROM hr.employees;
 id |    name    |         email          |     ssn     |  salary
----+------------+------------------------+-------------+----------
  1 | ********   | XXXXXXXX@XXXXXXX.com   | XXX-XX-6789 | 45000.00
  2 | ********** | XXXXXXXXXX@XXXXXXX.com | XXX-XX-4321 | 54000.00
```

了解 DML 操作中的遮罩行為

pg_columnmask 會一致地套用至所有 DML 操作，包括 INSERT、UPDATE、DELETE 和 MERGE 陳述式。當您執行這些操作時，Aurora PostgreSQL 會根據核心原則遮罩資料 – 從儲存體讀取的任何資料都會根據目前使用者的適用政策遮罩。

遮罩會影響下列一些查詢元件，例如：

- WHERE 子句
- JOIN 條件
- 子查詢
- RETURNING 子句

所有這些元件都以遮罩值運作，而不是原始資料。當資料寫入儲存體時未遮罩，使用者只會在回讀時看到其遮罩檢視。

Aurora PostgreSQL 會對實際儲存的值強制執行所有資料庫限制（非 NULL、UNIQUE、CHECK、FOREIGN KEY)，而非遮罩的值。如果未仔細設計遮罩函數，這有時可能會產生明顯的不一致。

遮罩可與資料欄層級許可搭配使用：

- 沒有 SELECT 權限的使用者無法讀取資料欄
- 具有 SELECT 權限的使用者會根據其適用的政策查看遮罩的值

了解觸發函數中的遮罩行為

將pg_columnmask政策套用至資料表時，請務必了解遮罩如何與觸發函數互動。觸發條件是自動執行的資料庫函數，以回應資料表上的某些事件，例如 INSERT、UPDATE 或 DELETE 操作。

根據預設，DDM 會根據觸發類型套用不同的遮罩規則：

資料表觸發條件

未遮罩轉換資料表 – 資料表上的觸發函數可以存取其轉換資料表中舊資料列和新資料列版本的未遮罩資料

資料表擁有者會建立觸發條件並擁有資料，因此他們具有有效管理資料表的完整存取權

檢視觸發 (INSTEAD OF Triggers)

轉換資料表已遮罩 – 檢視上的觸發函數會根據目前使用者的許可查看遮罩的資料

檢視擁有者可能與基礎資料表擁有者不同，因此應遵守基礎資料表上的遮罩政策

兩個伺服器層級組態參數會使用遮罩資料表控制觸發行為。這些只能由 設定rds_superuser：

- 限制遮罩資料表的觸發 – 當遮罩使用者對具有適用遮罩政策的資料表執行 DML 操作時，防止觸發執行。
- 使用遮罩資料表限制檢視的觸發： – 當檢視定義包含適用於目前使用者的遮罩政策的資料表時，防止在檢視上觸發執行。

Example函數應用程式與資料表和檢視之間的差異

下列範例會建立觸發函數來列印舊資料列值和新資料列值，然後示範相同的函數在連接到資料表時與檢視時的行為有何不同。

```
-- Create trigger function
CREATE OR REPLACE FUNCTION print_changes()
    RETURNS TRIGGER AS
    $$
        BEGIN
        RAISE NOTICE 'Old row: name=%, email=%, ssn=%, salary=%',
```

```
            OLD.name, OLD.email, OLD.ssn, OLD.salary;

        RAISE NOTICE 'New row: name=%, email=%, ssn=%, salary=%',
            NEW.name, NEW.email, NEW.ssn, NEW.salary;

        RETURN NEW;
        END;
    $$ LANGUAGE plpgsql;

-- Create trigger
CREATE TRIGGER print_changes_trigger
    BEFORE UPDATE ON hr.employees
    FOR EACH ROW
    EXECUTE FUNCTION print_changes();

-- Grant update to analyst role
GRANT UPDATE ON hr.employees TO analyst_role;

-- Unmasked data must be seen inside trigger even for masked user for the OLD and NEW
-- row passed to trigger function
BEGIN;
SET ROLE mike_analyst;
UPDATE hr.employees SET id = id + 10 RETURNING *;
NOTICE:  Old row: name=John Doe, email=john.doe@example.com, ssn=123-45-6789,
 salary=50000.00
NOTICE:  New row: name=John Doe, email=john.doe@example.com, ssn=123-45-6789,
 salary=50000.00
NOTICE:  Old row: name=Jane Smith, email=jane.smith@example.com, ssn=987-65-4321,
 salary=60000.00
NOTICE:  New row: name=Jane Smith, email=jane.smith@example.com, ssn=987-65-4321,
 salary=60000.00
 id |    name    |         email          |     ssn     |  salary
----+------------+------------------------+-------------+----------
 11 | John Doe   | john.doe@example.com   | XXX-XX-6789 | 45000.00
 12 | Jane Smith | jane.smith@example.com | XXX-XX-4321 | 54000.00
(2 rows)

ROLLBACK;


-- Triggers on views (which are supposed to see masked data for new/old row)
CREATE VIEW hr.view_over_employees AS SELECT * FROM hr.employees;
GRANT UPDATE, SELECT ON hr.view_over_employees TO analyst_role;
```

```
-- Create trigger for this view
CREATE TRIGGER print_changes_trigger
    INSTEAD OF UPDATE ON hr.view_over_employees
    FOR EACH ROW
    EXECUTE FUNCTION print_changes();

-- Masked new and old rows should be passed to trigger if trigger is on view
BEGIN;
SET ROLE mike_analyst;
UPDATE hr.view_over_employees SET id = id + 10 RETURNING *;
NOTICE:  Old row: name=John Doe, email=john.doe@example.com, ssn=XXX-XX-6789,
 salary=45000.00
NOTICE:  New row: name=John Doe, email=john.doe@example.com, ssn=XXX-XX-6789,
 salary=45000.00
NOTICE:  Old row: name=Jane Smith, email=jane.smith@example.com, ssn=XXX-XX-4321,
 salary=54000.00
NOTICE:  New row: name=Jane Smith, email=jane.smith@example.com, ssn=XXX-XX-4321,
 salary=54000.00
 id |    name    |         email          |     ssn     |  salary
----+------------+------------------------+-------------+----------
 11 | John Doe   | john.doe@example.com   | XXX-XX-6789 | 45000.00
 12 | Jane Smith | jane.smith@example.com | XXX-XX-4321 | 54000.00
(2 rows)
ROLLBACK;
```

建議您先檢閱觸發行為，再對遮罩的資料表實作觸發。資料表觸發程序可存取轉換資料表中的未遮罩資料，而檢視觸發程序則會看到遮罩的資料。

Example重新命名遮罩政策

下列範例示範如何使用 `rename_masking_policy` 程序重新命名現有政策。

```
-- Rename the strict policy
CALL pgcolumnmask.rename_masking_policy(
    'employee_mask_strict',
    'hr.employees',
    'intern_protection_policy'
);

-- Verify the rename
SELECT policyname, roles, weight
    FROM pgcolumnmask.pg_columnmask_policies
    WHERE tablename = 'employees'
```

```
    ORDER BY weight DESC;

        policyname        |     roles      | weight
--------------------------+----------------+--------
 employee_mask_light      | {analyst_role} |    100
 employee_mask_moderate   | {support_role} |     50
 intern_protection_policy | {intern_role}  |     10
```

Example更改政策權重

下列範例示範如何變更政策權重以變更其權重。

```
-- Change weight of moderate policy
CALL pgcolumnmask.alter_masking_policy(
    'employee_mask_moderate'::NAME,
    'hr.employees'::REGCLASS,
    NULL,    -- Keep existing masking expressions
    NULL,    -- Keep existing roles
    75       -- New weight
);

-- Verify the changes
SELECT policyname, roles, weight
    FROM pgcolumnmask.pg_columnmask_policies
    WHERE tablename = 'employees'
    ORDER BY weight DESC;
        policyname        |     roles      | weight
--------------------------+----------------+--------
 employee_mask_light      | {analyst_role} |    100
 employee_mask_moderate   | {support_role} |     75
 intern_protection_policy | {intern_role}  |     10
```

Example清除

下列範例示範如何捨棄所有政策、資料表和使用者。

```
-- Drop policies
CALL pgcolumnmask.drop_masking_policy(
    'intern_protection_policy',
    'hr.employees'
);
```

```
CALL pgcolumnmask.drop_masking_policy(
    'employee_mask_moderate',
    'hr.employees'
);

CALL pgcolumnmask.drop_masking_policy(
    'employee_mask_light',
    'hr.employees'
);

-- Drop table and functions
DROP VIEW IF EXISTS hr.view_over_employees;
DROP TABLE IF EXISTS hr.employees;
DROP SCHEMA IF EXISTS hr;
DROP FUNCTION IF EXISTS public.mask_ssn(text);
DROP FUNCTION IF EXISTS public.mask_salary(numeric, numeric);

-- Drop users
DROP USER sarah_intern, lisa_support, mike_analyst,
    ethan_support_intern, john_analyst_intern;
DROP ROLE intern_role, support_role, analyst_role;
```

設定遮罩政策管理角色

PostgreSQL 資料欄遮罩延伸模組 pg_columnmask可讓您將遮罩政策的管理委派給特定角色，而不是要求 rds_superuser或 資料表擁有者權限。這可以更精細地控制誰可以建立、更改和捨棄遮罩政策。

若要設定具有遮罩政策管理權限的角色，請遵循下列步驟：

1. 建立政策管理員角色 – 建立負責管理遮罩政策rds_superuser的新角色：

   ```
   CREATE ROLE mask_admin NOLOGIN;
   ```

2. 設定 PostgreSQL 參數 – 在自訂資料庫叢集參數群組中，將pgcolumnmask.policy_admin_rolname引擎組態參數設定為您建立的角色名稱：

   ```
   pgcolumnmask.policy_admin_rolname = mask_admin
   ```

   此引擎組態參數可以在資料庫叢集參數群組中設定，而且不需要重新啟動執行個體。如需更新參數的詳細資訊，請參閱 在 Amazon Aurora 中修改資料庫叢集參數群組中的參數。

3. 將角色授予使用者 作為 rds_superuser，將mask_admin角色授予應能夠管理遮罩政策的使用者：

```
CREATE USER alice LOGIN;
CREATE USER bob LOGIN;
GRANT mask_admin TO alice, bob;
```

此外，請確保使用者在管理遮罩政策的結構描述上具有 USAGE 權限：

```
GRANT USAGE ON SCHEMA hr TO alice, bob;
```

現在，當使用者alice和 bob 連線到資料庫時，他們可以使用標準pg_columnmask延伸函數，在所有結構描述具有USAGE權限的結構描述中的所有資料表上建立、更改和捨棄遮罩政策。

安全 pg_columnmask 實作的最佳實務

下一節提供pg_columnmask在 Aurora PostgreSQL 環境中實作 的安全最佳實務。請遵循這些建議：

- 建立安全的角色型存取控制架構
- 開發防止安全漏洞的遮罩函數
- 使用遮罩資料了解和控制觸發行為

角色型安全架構

定義角色階層以在資料庫中實作存取控制。Aurora PostgreSQL 透過為這些角色中的精細資料遮罩提供額外的圖層來pg_columnmask增強這些控制。

建立符合組織函數的專用角色，而不是將許可授予個別使用者。此方法可提供更好的可稽核性，並簡化組織結構演進時的許可管理。

Example建立組織角色階層

下列範例會建立具有不同 函數專用角色的組織角色階層，然後將個別使用者指派給適當的角色。在此範例中，會先建立組織角色 (analyst_role、support_role)，然後授與個別使用者這些角色的成員資格。此結構可讓您在角色層級管理許可，而不是針對每個個別使用者。

```
-- Create organizational role hierarchy
CREATE ROLE data_admin_role;
CREATE ROLE security_admin_role;
```

```
CREATE ROLE analyst_role;
CREATE ROLE support_role;
CREATE ROLE developer_role;

-- Specify security_admin_role as masking policy manager in the DB cluster parameter
-- group pgcolumnmask.policy_admin_rolname = 'security_admin_role'

-- Create specific users and assign to appropriate roles
CREATE USER security_manager;
CREATE USER data_analyst1, data_analyst2;
CREATE USER support_agent1, support_agent2;

GRANT security_admin_role TO security_manager;
GRANT analyst_role TO data_analyst1, data_analyst2;
GRANT support_role TO support_agent1, support_agent2;
```

僅授予每個角色所需的最低許可，以實作最低權限原則。避免授予在登入資料洩露時可能遭到利用的廣泛許可。

```
-- Grant specific table permissions rather than schema-wide access
GRANT SELECT ON sensitive_data.customers TO analyst_role;
GRANT SELECT ON sensitive_data.transactions TO analyst_role;
-- Do not grant: GRANT ALL ON SCHEMA sensitive_data TO analyst_role;
```

政策管理員需要管理遮罩政策之結構描述USAGE的權限。遵循最低權限原則，選擇性地授予這些權限。定期審查結構描述存取許可，以確保只有獲得授權的人員才能維護政策管理功能。

政策管理員角色參數組態僅限於資料庫管理員。此參數無法在資料庫或工作階段層級修改，防止無權限使用者覆寫政策管理員指派。此限制可確保遮罩政策控制保持集中且安全。

將政策管理員角色指派給特定個人，而非群組。此目標方法可確保選擇性存取遮罩政策管理，因為政策管理員能夠遮罩資料庫中的所有資料表。

安全遮罩函數開發

使用早期繫結語意開發遮罩函數，以確保適當的相依性追蹤，並防止延遲繫結漏洞，例如在執行期修改搜尋路徑。建議使用 SQL BEGIN ATOMIC 函數的語法來啟用編譯時間驗證（即早期繫結）和相依性管理。

```
-- Example - Secure masking function with early binding
CREATE OR REPLACE FUNCTION secure_mask_ssn(input_ssn TEXT)
    RETURNS TEXT
```

```
    LANGUAGE SQL
    IMMUTABLE PARALLEL SAFE STRICT
    BEGIN ATOMIC
        SELECT CASE
            WHEN input_ssn IS NULL THEN NULL
            WHEN length(input_ssn) < 4 THEN repeat('X', length(input_ssn))
            ELSE repeat('X', length(input_ssn) - 4) || right(input_ssn, 4)
        END;
    END;
```

或者，透過明確建構符合所有物件參考的結構描述，建立可避免搜尋路徑變更的函數，確保不同使用者工作階段的行為一致。

```
-- Function immune to search path changes
CREATE OR REPLACE FUNCTION data_masking.secure_phone_mask(phone_number TEXT)
    RETURNS TEXT
    LANGUAGE SQL
    IMMUTABLE PARALLEL SAFE STRICT
    AS $$
    SELECT CASE
        WHEN phone_number IS NULL THEN NULL
        WHEN public.length(public.regexp_replace(phone_number, '[^0-9]', '', 'g')) < 10
 THEN 'XXX-XXX-XXXX'
        ELSE public.regexp_replace(
            phone_number,
            '([0-9]{3})[0-9]{3}([0-9]{4})',
            public.concat('\1-XXX-\2')
        )
    END;
    $$;
```

在遮罩函數中實作輸入驗證，以處理邊緣案例並防止意外行為。一律包含 NULL 處理和驗證輸入格式，以確保一致的遮罩行為。

```
-- Robust masking function with comprehensive input validation
CREATE OR REPLACE FUNCTION secure_mask_phone(phone_number TEXT)
    RETURNS TEXT
    LANGUAGE SQL
    IMMUTABLE PARALLEL SAFE STRICT
    BEGIN ATOMIC
        SELECT CASE
            WHEN phone_number IS NULL THEN NULL
            WHEN length(trim(phone_number)) = 0 THEN phone_number
```

```
            WHEN length(regexp_replace(phone_number, '[^0-9]', '', 'g')) < 10 THEN
 'XXX-XXX-XXXX'
            ELSE regexp_replace(phone_number, '([0-9]{3})[0-9]{3}([0-9]{4})', '\1-XXX-
\2')
        END;
    END;
```

DML 使用 pg_columnmask 觸發行為

對於資料表觸發，轉換資料表將完全取消遮罩。對於檢視觸發 (IOT)，將根據目前使用者的檢視許可遮罩轉換資料表。

使用 pg_columnmask 的資料表觸發

觸發程序會傳遞轉換表，其中包含由觸發 DML 查詢修改的舊版和新版本資料列。根據觸發時間，Aurora PostgreSQL 會填入新舊資料列。例如，BEFORE INSERT觸發條件只有新版本的資料列和空白的舊版本，因為沒有要參考的舊版本。

pg_columnmask 不會遮罩資料表上觸發條件內的轉換資料表。觸發程序可以在其內文中使用遮罩的資料欄，並查看未遮罩的資料。觸發器建立者應確保如何為使用者執行觸發器。在此情況下，下列範例可正常運作。

```
-- Example for table trigger uses masked column in its definition
-- Create a table and insert some rows
CREATE TABLE public.credit_card_table (
    name TEXT,
    credit_card_no VARCHAR(16),
    is_fraud BOOL
);

INSERT INTO public.credit_card_table (name, credit_card_no, is_fraud)
    VALUES
    ('John Doe', '4532015112830366', false),
    ('Jane Smith', '5410000000000000', true),
    ('Brad Smith', '1234567891234567', true);

-- Create a role which will see masked data and grant it privileges
CREATE ROLE intern_user;
GRANT SELECT, DELETE ON public.credit_card_table TO intern_user;

-- Trigger which will silenty skip delete of non fraudelent credit cards
CREATE OR REPLACE FUNCTION prevent_non_fraud_delete()
    RETURNS TRIGGER AS
```

```
    $$
    BEGIN
        IF OLD.is_fraud = false THEN
            RETURN NULL;
        END IF;
        RETURN OLD;
    END;
    $$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_non_fraud_delete
    BEFORE DELETE ON credit_card_table
    FOR EACH ROW
    EXECUTE FUNCTION prevent_non_fraud_delete();

CREATE OR REPLACE FUNCTION public.return_false()
    RETURNS BOOLEAN
    LANGUAGE SQL
    IMMUTABLE PARALLEL SAFE STRICT
    BEGIN ATOMIC
      SELECT false;
    END;

-- A masking policy that masks both credit card number and is_fraud column.
-- If we apply masking inside trigger then prevent_non_fraud_delete trigger will
-- allow deleting more rows to masked user (even non fraud ones).
CALL pgcolumnmask.create_masking_policy(
    'mask_credit_card_no_&_is_fraud'::NAME,
    'public.credit_card_table'::REGCLASS,
    JSON_BUILD_OBJECT('credit_card_no', 'pgcolumnmask.mask_text(credit_card_no)',
                      'is_fraud', 'public.return_false()')::JSONB,
    ARRAY['intern_user']::NAME[],
    10::INT
);

-- Test trigger behaviour using intern_user
BEGIN;
SET ROLE intern_user;
-- credit card number & is_fraud is completely masked from intern_user
SELECT * FROM public.credit_card_table;
    name    |  credit_card_no  | is_fraud
------------+------------------+----------
 John Doe   | XXXXXXXXXXXXXXXX | f
 Jane Smith | XXXXXXXXXXXXXXXX | f
 Brad Smith | XXXXXXXXXXXXXXXX | f
```

```
(3 rows)

-- The delete trigger lets the intern user delete rows for Jane and Brad even though
-- intern_user sees their is_fraud = false, but the table trigger works with
 original
-- unmasked value
DELETE FROM public.credit_card_table RETURNING *;
    name    |  credit_card_no  | is_fraud
------------+------------------+----------
 Jane Smith | XXXXXXXXXXXXXXXX | f
 Brad Smith | XXXXXXXXXXXXXXXX | f
(2 rows)

COMMIT;
```

如果使用者對其在觸發內文中使用的陳述式不小心，觸發建立者會將未遮罩的資料洩漏給使用者。例如，使用 會將資料欄RAISE NOTICE '%', masked_column;列印到目前使用者。

```
-- Example showing table trigger leaking column value to current user
CREATE OR REPLACE FUNCTION leaky_trigger_func()
    RETURNS TRIGGER AS
    $$
    BEGIN
        RAISE NOTICE 'Old credit card number was: %', OLD.credit_card_no;
        RAISE NOTICE 'New credit card number is %', NEW.credit_card_no;
        RETURN NEW;
    END;
    $$ LANGUAGE plpgsql;

CREATE TRIGGER leaky_trigger
    AFTER UPDATE ON public.credit_card_table
    FOR EACH ROW
    EXECUTE FUNCTION leaky_trigger_func();

-- Grant update on column is_fraud to auditor role
-- auditor will NOT HAVE PERMISSION TO READ DATA
CREATE ROLE auditor;
GRANT UPDATE (is_fraud) ON public.credit_card_table TO auditor;

-- Also add auditor role to existing masking policy on credit card table
CALL pgcolumnmask.alter_masking_policy(
    'mask_credit_card_no_&_is_fraud'::NAME,
```

```
    'public.credit_card_table'::REGCLASS,
    NULL::JSONB,
    ARRAY['intern_user', 'auditor']::NAME[],
    NULL::INT
);

-- Log in as auditor
-- [auditor]
-- Update will fail if trying to read data from the table
UPDATE public.credit_card_table
    SET is_fraud = true
    WHERE credit_card_no = '4532015112830366';
ERROR:  permission denied for table cc_table

-- [auditor]
-- But leaky update trigger will still print the entire row even though
-- current user does not have permission to select from public.credit_card_table
UPDATE public.credit_card_table SET is_fraud = true;
NOTICE:  Old credit_card_no was: 4532015112830366
NOTICE:  New credit_card_no is 4532015112830366
```

使用 pg_columnmask 的檢視觸發（而非觸發）

觸發條件只能在 PostgreSQL 的檢視上建立。它們用於在無法更新的檢視上執行 DML 陳述式。傳輸資料表一律會在內部遮罩，而非觸發 (IOT)，因為檢視和檢視查詢中使用的基底資料表可能會有不同的擁有者。在這種情況下，基礎資料表可能會有一些適用於檢視擁有者的遮罩政策，而且檢視擁有者必須在其觸發條件內查看基礎資料表的遮罩資料。這與資料表上的觸發不同，因為在這種情況下，觸發建立者和資料表內的資料是由同一使用者所擁有，而不是這裡的情況。

```
-- Create a view over credit card table
CREATE OR REPLACE VIEW public.credit_card_view
    AS
    SELECT * FROM public.credit_card_table;

-- Truncate credit card table and insert fresh data
TRUNCATE TABLE public.credit_card_table;
INSERT INTO public.credit_card_table (name, credit_card_no, is_fraud)
    VALUES
    ('John Doe', '4532015112830366', false),
    ('Jane Smith', '5410000000000000', true),
    ('Brad Smith', '1234567891234567', true);

CREATE OR REPLACE FUNCTION public.print_changes()
```

```
    RETURNS TRIGGER AS
    $$
    BEGIN
        RAISE NOTICE 'Old row: name=%, credit card number=%, is fraud=%',
            OLD.name, OLD.credit_card_no, OLD.is_fraud;

        RAISE NOTICE 'New row: name=%, credit card number=%, is fraud=%',
            NEW.name, NEW.credit_card_no, NEW.is_fraud;

    RETURN NEW;
   END;
   $$ LANGUAGE plpgsql;

CREATE TRIGGER print_changes_trigger
    INSTEAD OF UPDATE ON public.credit_card_view
    FOR EACH ROW
    EXECUTE FUNCTION public.print_changes();

GRANT SELECT, UPDATE ON public.credit_card_view TO auditor;

-- [auditor]
-- Login as auditor role
BEGIN;

-- Any data coming out from the table will be masked in instead of triggers
-- according to masking policies applicable to current user
UPDATE public.credit_card_view
    SET name = CONCAT(name, '_new_name')
    RETURNING *;
NOTICE:  Old row: name=John Doe, credit card number=XXXXXXXXXXXXXXXX, is fraud=f
NOTICE:  New row: name=John Doe_new_name, credit card number=XXXXXXXXXXXXXXXX, is
 fraud=f
NOTICE:  Old row: name=Jane Smith, credit card number=XXXXXXXXXXXXXXXX, is fraud=f
NOTICE:  New row: name=Jane Smith_new_name, credit card number=XXXXXXXXXXXXXXXX, is
 fraud=f
NOTICE:  Old row: name=Brad Smith, credit card number=XXXXXXXXXXXXXXXX, is fraud=f
NOTICE:  New row: name=Brad Smith_new_name, credit card number=XXXXXXXXXXXXXXXX, is
 fraud=f
        name         |  credit_card_no  | is_fraud
---------------------+------------------+----------
 John Doe_new_name   | XXXXXXXXXXXXXXXX | f
 Jane Smith_new_name | XXXXXXXXXXXXXXXX | f
 Brad Smith_new_name | XXXXXXXXXXXXXXXX | f
```

```
 -- Any new data going into the table using INSERT or UPDATE command will be
 unmasked
 UPDATE public.credit_card_view
    SET credit_card_no = '9876987698769876'
    RETURNING *;
NOTICE:  Old row: name=John Doe, credit card number=XXXXXXXXXXXXXXXX, is fraud=f
NOTICE:  New row: name=John Doe, credit card number=9876987698769876, is fraud=f
NOTICE:  Old row: name=Jane Smith, credit card number=XXXXXXXXXXXXXXXX, is fraud=f
NOTICE:  New row: name=Jane Smith, credit card number=9876987698769876, is fraud=f
NOTICE:  Old row: name=Brad Smith, credit card number=XXXXXXXXXXXXXXXX, is fraud=f
NOTICE:  New row: name=Brad Smith, credit card number=9876987698769876, is fraud=f
    name    |  credit_card_no  | is_fraud
------------+------------------+----------
 John Doe   | 9876987698769876 | f
 Jane Smith | 9876987698769876 | f
 Brad Smith | 9876987698769876 | f

 COMMIT;
```

控制觸發行為的資料庫/使用者層級 GuCs

兩個組態參數控制具有適用遮罩政策之使用者的觸發執行行為。當需要額外的安全限制時，請使用這些參數來防止觸發程序在遮罩的資料表或檢視上執行。預設會停用這兩個參數，允許觸發程序正常執行。

第一個 GUC：遮罩資料表的觸發條件限制

規格：

- 名稱：pgcolumnmask.restrict_dml_triggers_for_masked_users
- 類型：boolean
- 預設：false（允許執行觸發）

設為 TRUE 時，防止遮罩使用者在遮罩資料表上觸發執行。 pg_columnmask會執行錯誤。

第二個 GUC：對具有遮罩資料表的檢視觸發觸發限制

規格：

- 名稱：pgcolumnmask.restrict_iot_triggers_for_masked_users
- 類型：boolean
- 預設：false（允許執行觸發）

設為 TRUE 時，防止對遮罩使用者在其定義中包含遮罩資料表的檢視觸發執行。

這些參數可獨立運作，並可像標準資料庫組態參數一樣進行設定。

Aurora PostgreSQL pg_columnmask 資料移動案例

pg_columnmask 根據操作是發生在儲存體、邏輯或應用程式層，行為會因不同的資料移動操作而有所不同。儲存層級操作（例如複製）的行為與邏輯操作（例如 pg_dump) 和應用程式層級操作（例如 FDW 查詢）不同。本節說明複寫、備份、匯出和遷移等常見案例的遮罩行為，並說明每個案例的安全影響。

主題

- Aurora 全域資料庫和僅供讀取複本
- 資料庫複製和快照還原
- 邏輯複寫
- 藍/綠部署
- 零 ETL 和 CDC 串流
- AWS Database Migration Service
- 資料匯出
- 檢視和具體化檢視
- 資料傾印和還原
- 外部資料包裝函式

Aurora 全域資料庫和僅供讀取複本

Aurora pg_columnmask政策存放在叢集磁碟區的資料庫系統資料表中。所有複本都會存取相同的政策，並傳回一致的遮罩結果。對於 Aurora Global Database 部署，pg_columnmask政策 AWS 區域會與其他資料庫系統資料表一起複寫到次要 ，以確保跨區域的一致資料保護。在容錯移轉案例期間，所有pg_columnmask政策都保持不變且正常運作。

資料庫複製和快照還原

Aurora Fast Clone 和快照還原操作會保留所有pg_columnmask政策、角色和組態，做為資料庫系統資料表的一部分。複製或還原的資料庫會從來源叢集繼承所有現有的政策。複製或還原之後，每個資料庫叢集都會維護獨立的pg_columnmask政策。

邏輯複寫

在初始同步期間，邏輯複寫會使用標準 SQL COPY 操作，並根據複寫使用者的許可強制執行pg_columnmask政策。在進行中的 CDC （變更資料擷取） 期間，不會套用遮罩政策，並透過 WAL 記錄複寫未遮罩的資料。具有 pg_create_subscription權限的使用者可以透過將複寫設定到他們控制的系統，潛在地洩漏未遮罩的資料。

藍/綠部署

在快照還原期間，會自動包含pg_columnmask政策。綠色環境從藍色環境中所有政策的相同副本開始。從藍色複寫到綠色期間，不會遮罩資料。藍色叢集上的後續遮罩政策變更 (DDL 命令） 不會複寫到綠色叢集，並使 RDS 藍/綠部署失效。

零 ETL 和 CDC 串流

資料複寫不受pg_columnmask政策影響。零 ETL 支援 DDL 複寫，但不複寫 pg_columnmask或 RLS 政策。不會將遮罩政策套用至 Zero-ETL 中的複寫資料。

AWS Database Migration Service

初始資料同步會根據為 DMS 任務選取的使用者進行遮罩或取消遮罩。CDC 資料一律會取消遮罩。雖然pg_columnmask相關的內部 RLS 政策可能會遷移，但它們無法在啟用非 pg_columnmask 的目標上運作。

資料匯出

pg_columnmask 將匯出視為任何其他查詢操作，會根據執行使用者的許可套用遮罩。這適用於 SQL 命令，例如 COPY、SELECT INTO、CREATE TABLE AS 和 Aurora PostgreSQL 的 S3 匯出功能。

> **Note**
>
> 當遮罩使用者匯出資料時，產生的檔案包含遮罩值，這些值可能會在還原時違反資料庫限制。

檢視和具體化檢視

使用檢視時，請記住下列考量事項：

- 一般檢視 – INVOKER 一律使用語意。無論誰建立檢視，目前使用者的遮罩政策都會在查詢檢視時套用。
- 具體化視觀表 – 重新整理時，套用具體化視觀表擁有者的遮罩政策，而非執行重新整理的使用者政策。如果擁有者有遮罩政策，具體化視觀表一律包含遮罩資料。

資料傾印和還原

pg_dump 以一般資料庫使用者身分運作，並根據連線使用者的許可套用遮罩政策。如果遮罩的使用者執行傾印，備份檔案會包含遮罩的資料。 pg_columnmask 政策會包含在傾印中，做為資料庫結構描述的一部分。成功還原需要目標資料庫中存在所有參考的角色，且目標已安裝pg_columnmask延伸模組。

> **Note**
>
> 從 PostgreSQL 18 開始， pg_dump支援從資料庫傾印中排除資料列層級安全性 (RLS) 和pg_columnmask遮罩政策–no-policies的選項。如需詳細資訊，請參閱 pg_dump。

外部資料包裝函式

使用外部資料包裝函式時，遠端資料表上的遮罩政策會根據映射的使用者在來源伺服器上的許可套用，而不是本機查詢使用者的許可，而且雖然您可以透過 FDW 存取遮罩的遠端資料，但您無法直接在本機資料庫中的外部資料表上建立 DDM 或 RLS 政策。

將應用程式更新為使用新的 SSL/TLS 憑證來連線至 Aurora PostgreSQL 資料庫叢集

自 2023 年 1 月 13 日起，Amazon RDS 已發佈新的憑證認證機構 (CA) 憑證，使用 Secure Socket Layer 或 Transport Layer Security (SSL/TLS) 來連線至 Aurora 資料庫叢集。接下來，您可以找到更新應用程式使用新憑證的相關資訊。

本主題可協助您判斷任何用戶端應用程式是否使用 SSL/TLS 連線至您的資料庫叢集。若是如此，您可以進一步檢查那些應用程式是否需要驗證憑證才能連線。

> **Note**
>
> 有些應用程式設定為只有在成功驗證伺服器上的憑證時，才能連線至 Aurora PostgreSQL 資料庫叢集。
> 對於這些應用程式，您必須更新用戶端應用程式信任存放區來包含新的 CA 憑證。

更新用戶端應用程式信任存放區中的 CA 憑證之後，您就可以在資料庫叢集輪換憑證。強烈建議先在開發或預備環境中測試這些步驟，再於生產環境中實作。

如需憑證輪換的詳細資訊，請參閱輪換您的 SSL/TLS 憑證。如需下載憑證的詳細資訊，請參閱使用 SSL/TLS 加密與資料庫叢集的連線。如需對 PostgreSQL 資料庫叢集使用 SSL/TLS 的資訊，請參閱使用 SSL/TLS 保護 Aurora PostgreSQL 資料的安全。

主題

- 判斷任何應用程式是否使用 SSL 連線至 Aurora PostgreSQL 資料庫叢集
- 判斷用戶端是否需要驗證憑證才能連線
- 更新應用程式信任存放區
- 針對不同類型的應用程式使用 SSL/TLS 連線

判斷任何應用程式是否使用 SSL 連線至 Aurora PostgreSQL 資料庫叢集

在資料庫叢集組態中檢查 `rds.force_ssl` 參數的值。依預設，`rds.force_ssl` 參數設為 `0` (關閉)。如果 `rds.force_ssl` 參數設為 `1` (開啟)，則用戶端需要使用 SSL/TLS 進行連線。如需參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。

如果 `rds.force_ssl` 不設為 `1` (開啟)，請查詢 `pg_stat_ssl` 檢視，以檢查使用 SSL 的連線。例如，下列查詢只傳回 SSL 連線和關於使用 SSL 的用戶端的資訊。

```
select datname, usename, ssl, client_addr from pg_stat_ssl inner join pg_stat_activity
 on pg_stat_ssl.pid = pg_stat_activity.pid where ssl is true and usename<>'rdsadmin';
```

只有使用 SSL/TLS 連線的列才會顯示連線的相關資訊。下列為範例輸出。

```
 datname  | usename | ssl | client_addr
----------+---------+-----+-------------
 benchdb  | pgadmin | t   | 53.95.6.13
 postgres | pgadmin | t   | 53.95.6.13
(2 rows)
```

上述查詢只顯示查詢當時的連線。沒有結果不代表沒有應用程式使用 SSL 連線。其他 SSL 連線可能在不同時間建立。

判斷用戶端是否需要驗證憑證才能連線

當用戶端 (例如 psql 或 JDBC) 設有 SSL 支援時，依預設，用戶端會先嘗試以 SSL 連線至資料庫。如果用戶端無法以 SSL 連線，則回復為不以 SSL 來連線。用於 libpq 型用戶端 (例如 psql) 與 JDBC 的

預設 sslmode 模式設定為 prefer。只有當提供 sslrootcert 並搭配 sslmode 設為 verify-ca 或 verify-full 時，才會驗證伺服器上的憑證。如果憑證無效，則擲出錯誤。

使用 PGSSLROOTCERT 搭配 PGSSLMODE 環境變數來驗證憑證，而且 PGSSLMODE 設為 verify-ca 或 verify-full。

```
PGSSLMODE=verify-full PGSSLROOTCERT=/fullpath/ssl-cert.pem psql -h
 pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com -U primaryuser -d postgres
```

使用 sslrootcert 引數搭配連線字串格式的 sslmode 來驗證憑證，而且 sslmode 設為 verify-ca 或 verify-full。

```
psql "host=pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com sslmode=verify-full
 sslrootcert=/full/path/ssl-cert.pem user=primaryuser dbname=postgres"
```

例如，在上述案例中，如果您使用無效根憑證，則在用戶端會看到類似以下的錯誤。

```
psql: SSL error: certificate verify failed
```

更新應用程式信任存放區

如需為 PostgreSQL 應用程式更新信任存放區的資訊，請參閱 PostgreSQL 文件中的使用 SSL 保護 TCP/IP 連線的安全。

> **Note**
>
> 更新信任存放區時，除了新增憑證，您還可以保留舊憑證。

為 JDBC 更新應用程式信任存放區

您可以為使用 JDBC 建立 SSL/TLS 連線的應用程式更新信任存放區。

如需下載根憑證的資訊，請參閱 使用 SSL/TLS 加密與資料庫叢集的連線。

如需匯入憑證的範例指令碼，請參閱 將憑證匯入信任存放區的範例指令碼。

針對不同類型的應用程式使用 SSL/TLS 連線

以下提供針對不同類型的應用程式使用 SSL/TLS 連線的資訊：

- psql

  以連線字串或環境變數指定選項，從命令列叫用此用戶端。若為 SSL/TLS 連線，相關選項為 `sslmode` (環境變數 `PGSSLMODE`)、`sslrootcert` (環境變數 `PGSSLROOTCERT`)。

  如需完整的選項清單，請參閱 PostgreSQL 文件中的參數關鍵字。如需完整的環境變數清單，請參閱 PostgreSQL 文件中的環境變數。

- pgAdmin

  這個以瀏覽器為基礎的用戶端是更方便連線至 PostgreSQL 資料庫的介面。

  如需設定連線的資訊，請參閱 pgAdmin 文件。

- JDBC

  JDBC 可讓 Java 應用程式連線至資料庫。

  如需使用 JDBC 連線至 PostgreSQL 資料庫的一般資訊，請參閱 PostgreSQL 文件中的連線至資料庫。如需使用 SSL/TLS 來連線的資訊，請參閱 PostgreSQL 文件中的設定用戶端。

- Python

  常用來連線至 PostgreSQL 資料庫的 Python 程式庫是 `psycopg2`。

  如需使用 `psycopg2` 的資訊，請參閱 psycopg2 文件。如需如何連線至 PostgreSQL 資料庫的簡短教學課程，請參閱 Psycopg2 教學。您可以在 psycopg2 模組內容中找到 connect 命令接受的選項的相關資訊。

> **Important**
>
> 在確定了資料庫連線使用 SSL/TLS 並已更新應用程式信任存放區之後，您可以將資料庫更新為使用 rds-ca-rsa2048-g1 憑證。如需說明，請參閱透過修改資料庫執行個體來更新憑證授權機構憑證中的步驟 3。

搭配 Aurora PostgreSQL 使用 Kerberos 身分驗證

在使用者連線至執行 PostgreSQL 的資料庫叢集時，您可以使用 Kerberos 驗證用戶身分。若要這樣做，請將資料庫叢集設定為 AWS Directory Service for Microsoft Active Directory 用於 Kerberos 身分驗證。 AWS Directory Service for Microsoft Active Directory 也稱為 AWS Managed Microsoft AD。這

是 提供的功能 Directory Service。若要進一步了解，請參閱《 AWS Directory Service 管理指南》中的什麼是 Directory Service？。

若要開始，請建立 AWS Managed Microsoft AD 目錄來存放使用者登入資料。然後，將 Active Directory 網域和其他資訊提供給 PostgreSQL 資料庫叢集。當使用者向 PostgreSQL 資料庫叢集進行驗證時，身分驗證請求會轉送到 AWS Managed Microsoft AD 目錄。

將您的所有登入資料保留在相同目錄可以節省您的時間和精力。這樣您就有一個集中的位置來存放及管理多個資料庫叢集的登入資料。使用目錄也可以改善您的整體安全性描述檔。

除此之外，您也可以從自己的內部部署 Microsoft Active Directory 存取登入資料。若要執行這項操作，請建立信任網域關聯，讓 AWS Managed Microsoft AD 目錄信任您的內部部署 Microsoft Active Directory。如此一來，使用者就可以透過在存取您內部部署網路的工作負載時的相同 Windows 單一登入 (SSO) 體驗，來存取 PostgreSQL 叢集。

資料庫可以使用 Kerberos、 AWS Identity and Access Management (IAM) 或 Kerberos 和 IAM 身分驗證。不過，因為 Kerberos 和 IAM 身分驗證提供了不同的身分驗證方法，所以特定資料庫使用者只能使用一種或其他身分驗證方法登入資料庫，但不能同時使用兩者。如需 IAM 身分驗證的詳細資訊，請參閱 IAM 資料庫身分驗證。

> **Note**
>
> RDS for PostgreSQL 不支援 Active Directory 群組的 Kerberos 身分驗證。

主題

- 區域和版本可用性
- 對 PostgreSQL 資料庫叢集進行 Kerberos 身分驗證的概觀
- 為 PostgreSQL 資料庫叢集設定 Kerberos 身分驗證
- 在 Active Directory 網域中管理 Aurora PostgreSQL 資料庫叢集
- 使用 Kerberos 身分驗證連線至 PostgreSQL
- 使用 AD 安全群組進行 Aurora PostgreSQL 存取控制

區域和版本可用性

功能可用性和支援會因每個資料庫引擎的特定版本以及 AWS 區域而有所不同。如需可與 Kerberos 身分驗證搭配使用之 Aurora PostgreSQL 版本和區域的詳細資訊，請參閱 使用 Aurora PostgreSQL 進行 Kerberos 身分驗證。

對 PostgreSQL 資料庫叢集進行 Kerberos 身分驗證的概觀

若要為 PostgreSQL 資料庫叢集設定 Kerberos 身分驗證，請遵循下列步驟，稍後會提供更詳細的說明：

1. 使用 AWS Managed Microsoft AD 建立 AWS Managed Microsoft AD 目錄。您可以使用 AWS 管理主控台、 AWS CLI或 Directory Service API 來建立目錄。請務必開啟目錄安全性群組上的相關輸出連接埠，以便目錄可以與 叢集 通訊。
2. 建立提供 Amazon Aurora 存取權的角色，以呼叫您的 AWS Managed Microsoft AD 目錄。若要這樣做，請建立使用 受管 IAM 政策 的 AWS Identity and Access Management (IAM) 角色`AmazonRDSDirectoryServiceAccess`。

   若要讓 IAM 角色允許存取，必須在您 AWS 帳戶的正確 AWS 區域中啟用 AWS Security Token Service (AWS STS) 端點。 AWS STS 端點預設會在所有 中處於作用中狀態 AWS 區域，而且您可以使用它們，而不需要任何進一步的動作。如需詳細資訊，請參閱《IAM 使用者指南》中的AWS STS 在 AWS 區域中啟用和停用 。
3. 使用 Microsoft Active Directory 工具在 AWS Managed Microsoft AD 目錄中建立和設定使用者。如需有關在 Active Directory 中建立使用者的詳細資訊，請參閱《 管理指南》中的管理 AWS Managed Microsoft AD 中的使用者和群組。 Directory Service
4. 如果您打算在不同 AWS 帳戶或 Virtual Private Cloud (VPC) 中尋找目錄和資料庫執行個體，請設定 VPC 互連。如需詳細資訊，請參閱《Amazon VPC Peering Guide》中的什麼是 VPC 互連？。
5. 使用下列其中一種方法，從主控台、CLI 或 RDS API 建立或修改 PostgreSQL 資料庫叢集：
   - 建立 Aurora PostgreSQL 資料庫叢集並與之連線
   - 修改 Amazon Aurora 資料庫叢集
   - 從資料庫叢集快照還原
   - 將資料庫叢集還原至指定時間

   您可以在與目錄相同的 Amazon Virtual Private Cloud (VPC) 或不同的 AWS 帳戶或 VPC 中找到叢集。當您建立或修改 PostgreSQL 資料庫叢集時，請執行下列動作：
   - 請提供您建立目錄時產生的網域識別符 (d-* 識別符)。
   - 請提供所建立的 IAM 角色名稱。
   - 確保資料庫執行個體安全群組可以從目錄的安全群組接收傳入流量。
6. 使用 RDS 主要使用者登入資料來連接至 PostgreSQL 資料庫叢集。然後，在 PostgreSQL 中建立要在外部識別的使用者。外部識別的使用者可以使用 Kerberos 身分驗證來登入 PostgreSQL 資料庫叢集。

為 PostgreSQL 資料庫叢集設定 Kerberos 身分驗證

您可以使用 AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) 來設定 PostgreSQL 資料庫叢集的 Kerberos 身分驗證。若要設定 Kerberos 身分驗證，請遵循下列步驟。

主題

- 步驟 1：使用 建立目錄 AWS Managed Microsoft AD
- 步驟 2：（選用）在內部部署 Active Directory 與 之間建立信任關係 Directory Service
- 步驟 3：為 Amazon Aurora 建立 IAM 角色以存取 Directory Service
- 步驟 4：建立和設定使用者
- 步驟 5：啟用目錄和資料庫執行個體之間的跨 VPC 流量
- 步驟 6：建立或修改 PostgreSQL 資料庫叢集
- 步驟 7：針對您的 Kerberos 主體建立 PostgreSQL 使用者
- 步驟 8：設定 PostgreSQL 用戶端

步驟 1：使用 建立目錄 AWS Managed Microsoft AD

Directory Service 在 AWS 雲端中建立全受管 Active Directory。當您建立 AWS Managed Microsoft AD 目錄時，會為您 Directory Service 建立兩個網域控制站和 DNS 伺服器。目錄伺服器是在 VPC 的不同子網路中建立。此備援有助於確保即使發生故障，您仍然可以存取目錄。

當您建立 AWS Managed Microsoft AD 目錄時，AWS Directory Service 會代表您執行下列任務：

- 在 VPC 內設定 Active Directory。
- 建立含有使用者名稱 `Admin` 與指定密碼的目錄管理員帳戶。您可以使用此帳戶來管理目錄。

> **Important**
>
> 請務必儲存此密碼。 Directory Service 不會儲存此密碼，也無法擷取或重設。

- 建立目錄控制器的安全群組。安全群組必須允許與 PostgreSQL 資料庫叢集進行通訊。

啟動時 AWS Directory Service for Microsoft Active Directory，AWS 會建立組織單位 (OU)，其中包含目錄的所有物件。此 OU 有您在建立目錄時所輸入的 NetBIOS 名稱，位於根網域中。網域根由 擁有和管理 AWS。

使用 AWS Managed Microsoft AD 目錄建立Admin的帳戶具有 OU 最常見管理活動的許可：

- 建立、更新或刪除使用者
- 將資源 (例如檔案或列印伺服器) 新增至您的網域，然後對您 OU 中的使用者指派這些資源的許可
- 建立額外的 OU 和容器
- 委派授權
- 從 Active Directory 資源回收筒還原已刪除的物件
- 在 Active Directory Web 服務上執行適用於 Windows PowerShell 的 Active Directory 和網域名稱服務 (DNS) 模組

Admin 帳戶也有權執行下列全網域活動：

- 管理 DNS 組態 (新增、移除或更新記錄、區域和轉寄站)
- 檢視 DNS 事件日誌
- 檢視安全事件日誌

使用 建立目錄 AWS Managed Microsoft AD

1. 在 Directory Service 主控台中，選擇 Directories (目錄)，然後選擇 Set up directory (設定目錄)。
2. 選擇 AWS Managed Microsoft AD。 AWS Managed Microsoft AD 是目前支援與 Amazon Aurora 搭配使用的唯一選項。
3. 選擇下一步。
4. 在 Enter directory information (輸入目錄資訊) 頁面上，提供下列資訊：

   版本

   選擇滿足您需求的版本。

   目錄 DNS 名稱

   目錄的完全合格名稱，例如 **corp.example.com**。

   目錄 NetBIOS 名稱

   目錄的簡短名稱，例如：CORP。

   目錄描述

   選擇填寫其他目錄說明。

管理員密碼

目錄管理員的密碼。目錄建立程序會建立含有使用者名稱 Admin 與這組密碼的管理者帳戶。

目錄管理員密碼不得包含 "admin" 字組。密碼區分大小寫，長度須為 8 至 64 個字元。至少須有一位字元屬於以下四種類型中的三類：

- 小寫字母 (a–z)
- 大寫字母 (A–Z)
- 數字 (0–9)
- 非英數字元 (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

Confirm password (確認密碼)

重新輸入管理員密碼。

> **Important**
> 請確定您儲存此密碼。 Directory Service 不會儲存此密碼，而且無法擷取或重設。

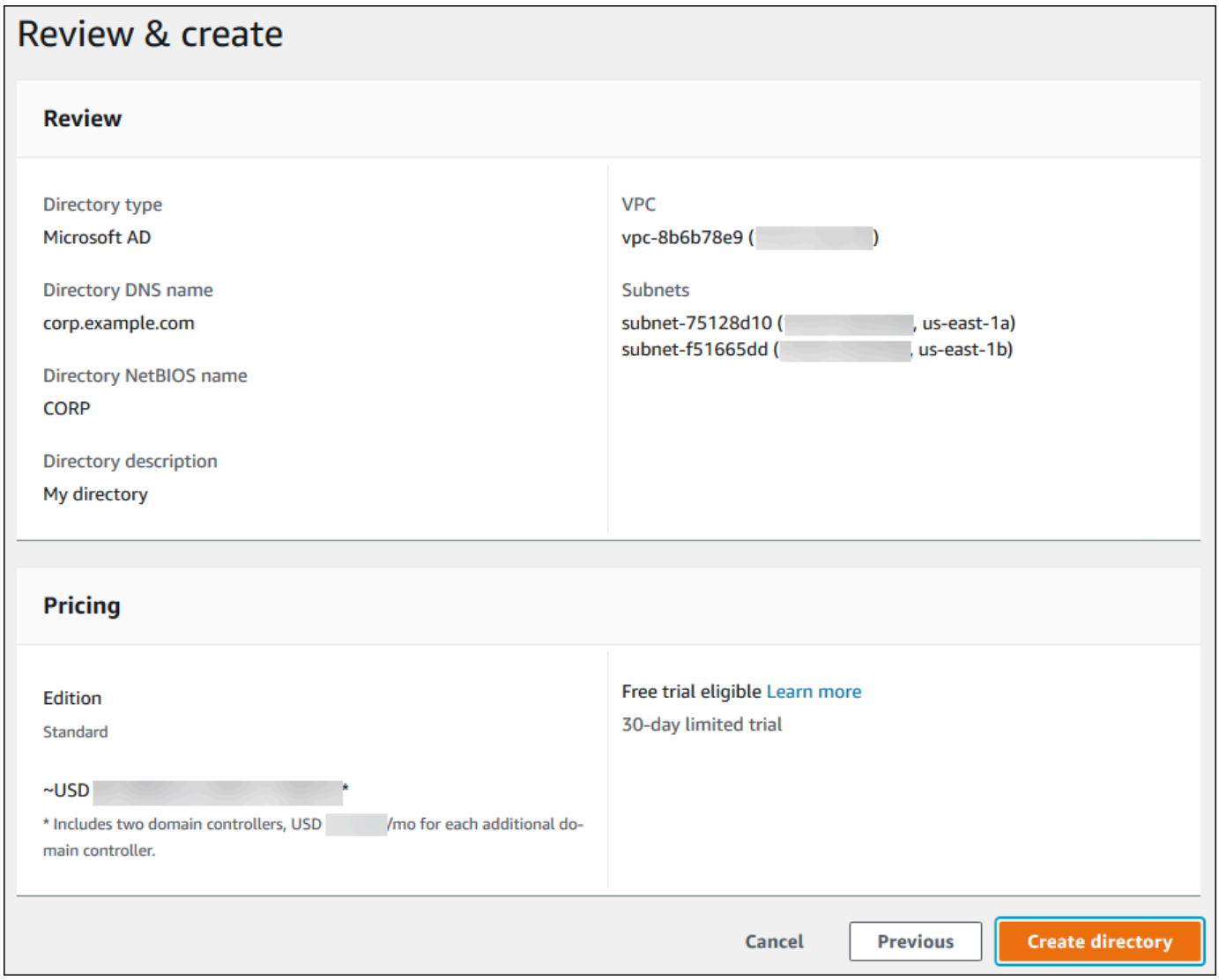
5. 選擇下一步。
6. 在 Choose VPC and subnets (選擇 VPC 和子網路) 頁面上，提供下列資訊：

   VPC

   選擇目錄的 VPC。您可在此相同 VPC 或不同 VPC 中建立 PostgreSQL 資料庫叢集。

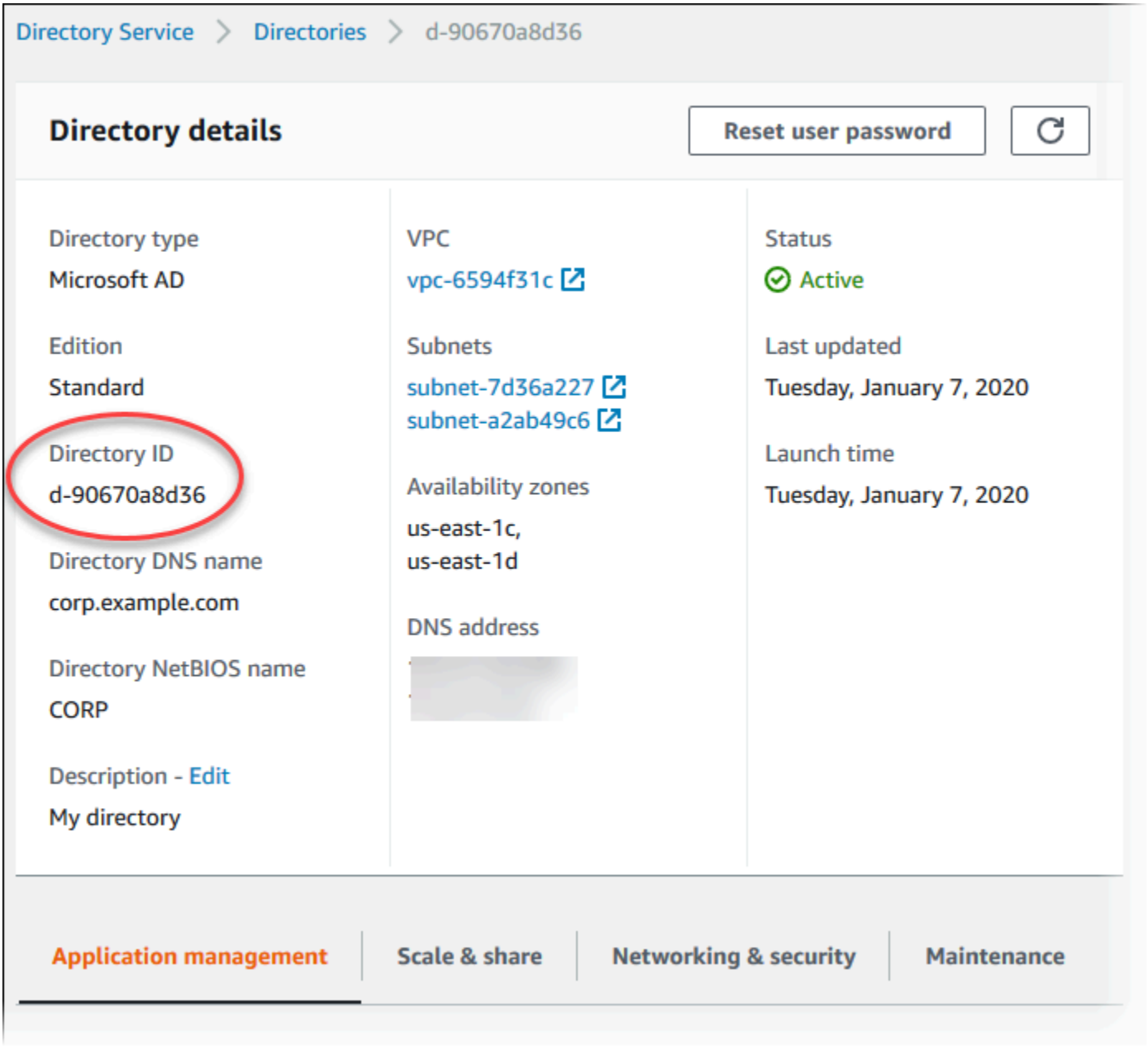
   子網路

   選擇目錄伺服器的子網路。這兩個子網路必須位於不同的可用區域。
7. 選擇 Next (下一步)。
8. 檢閱目錄資訊。如果需要變更，請選擇 Previous (上一步)，然後進行變更。若資訊無誤，請選擇 Create directory (建立目錄)。



建立目錄需要幾分鐘的時間。成功建立時，Status (狀態) 值會變更為 Active (作用中)。

若要查看目錄的資訊，請選擇目錄清單中的目錄 ID。請記下 Directory ID (目錄 ID) 值，建立或修改 PostgreSQL 資料庫執行個體時需要此值。



步驟 2：（選用）在內部部署 Active Directory 與 之間建立信任關係 Directory Service

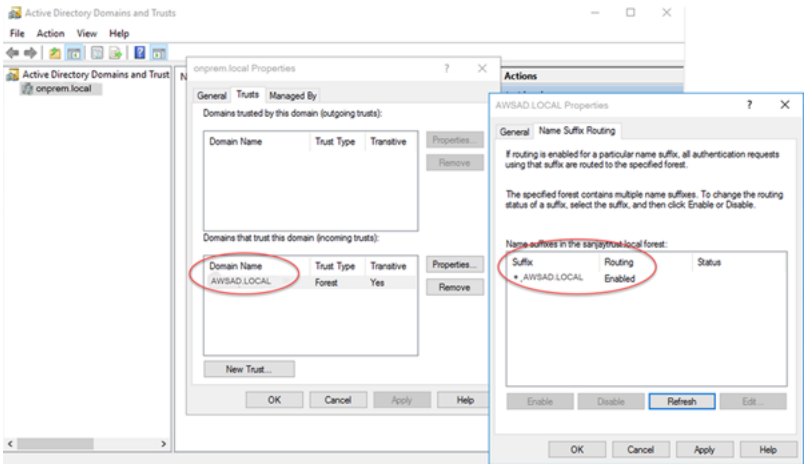
如果您不打算使用自己的內部部署 Microsoft Active Directory，請跳至步驟 3：為 Amazon Aurora 建立 IAM 角色以存取 Directory Service.

若要使用內部部署 Active Directory 取得 Kerberos 身分驗證，您需要使用內部部署 Microsoft Active Directory 與 AWS Managed Microsoft AD 目錄（在 中建立）之間的樹系信任來建立信任網域關係步驟 1：使用 建立目錄 AWS Managed Microsoft AD。信任可以是單向，其中 AWS Managed Microsoft AD 目錄信任內部部署 Microsoft Active Directory。信任也可以是雙向，其中兩個 Active Directory 互相

信任。如需使用 設定信任的詳細資訊 Directory Service，請參閱《 AWS Directory Service 管理指南》中的何時建立信任關係。

> **Note**
>
> 如果您使用內部部署 Microsoft Active Directory：
>
> - Windows 用戶端必須使用端點 Directory Service 中 的網域名稱來連線，而不是 rds.amazonaws.com。如需詳細資訊，請參閱使用 Kerberos 身分驗證連線至 PostgreSQL。
> - Windows 用戶端無法使用 Aurora 自訂端點進行連線。如需詳細資訊，請參閱 Amazon Aurora 端點連線。
> - 針對全球資料庫：
>   - Windows 用戶端只能使用全域資料庫主要 中的執行個體端點或叢集端點 AWS 區域 進行連線。
>   - Windows 用戶端無法使用次要叢集端點進行連線 AWS 區域。

請確定您的內部部署 Microsoft Active Directory 網域名稱包含對應至新建立之信任關係的 DNS 尾碼路由。以下螢幕擷取畫面顯示了一個範例。



步驟 3：為 Amazon Aurora 建立 IAM 角色以存取 Directory Service

若要讓 Amazon Aurora Directory Service 為您呼叫 ， AWS 您的帳戶需要使用 受管 IAM 政策 的 IAM 角色`AmazonRDSDirectoryServiceAccess`。此角色允許 Amazon Aurora 呼叫 Directory Service。（請注意，此存取 的 IAM 角色與用於 的 IAM 角色 Directory Service 不同 IAM 資料庫身分驗證。）

當您使用 建立資料庫執行個體， AWS 管理主控台 且主控台使用者帳戶具有 iam:CreateRole許可時，主控台會自動建立所需的 IAM 角色。在此情況下，角色名稱為 rds-directoryservice-kerberos-access-role。否則，您必須手動建立 IAM 角色。當您建立此 IAM 角色時，請選擇 Directory Service，然後將 AWS 受管政策連接至AmazonRDSDirectoryServiceAccess該角色。

如需為服務建立 IAM 角色的詳細資訊，請參閱《IAM 使用者指南》中的建立角色以將許可委派給 AWS 服務。

> **Note**
>
> 用於 Windows Authentication for RDS for Microsoft SQL Server 的 IAM 角色不可用於 Amazon Aurora。

作為使用 AmazonRDSDirectoryServiceAccess 受管政策的替代方案，您可以建立具有必要許可的政策。在此情況下，IAM 角色必須有以下 IAM 信任政策。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

此角色也須具有下列 IAM 角色政策：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

對於選擇加入 AWS 區域，請在 IAM 角色信任政策中使用區域特定的服務主體。當您為這些區域中的服務建立信任政策時，請在服務主體中指定區域代碼。

下列範例顯示的信任政策，其中包含區域專屬服務主體：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.REGION-CODE.amazonaws.com",
          "rds.REGION-CODE.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

將 REGION-CODE 取代為特定區域的代碼。例如，針對亞太區域 (墨爾本) 使用下列服務主體：

```
"Service": [
  "directoryservice.rds.ap-southeast-4.amazonaws.com",
  "rds.ap-southeast-4.amazonaws.com"
]
```

步驟 4：建立和設定使用者

您可以透過使用 Active Directory 使用者和運算集區來建立使用者。這是 Active Directory Domain Services 和 Active Directory 輕量型目錄服務工具之一。如需詳細資訊，請參閱 Microsoft 文件中的將使用者和電腦新增至 Active Directory 網域。在此情況下，使用者是個人或其他實體，例如其電腦屬於網域，而其身分在目錄中維護。

若要在 Directory Service 目錄中建立使用者，您必須連線到屬於 Directory Service 目錄成員的 Windows 型 Amazon EC2 執行個體。同時，您必須以具有建立使用者之許可的使用者身分來登入。如需詳細資訊，請參閱《AWS Directory Service 管理指南》中的建立使用者。

步驟 5：啟用目錄和資料庫執行個體之間的跨 VPC 流量

如果您打算在相同 VPC 中尋找目錄和資料庫叢集，請略過本步驟，並移至步驟 6：建立或修改 PostgreSQL 資料庫叢集。

如果您打算在不同 VPC 中尋找目錄和資料庫執行個體，請使用 VPC 互連或 AWS Transit Gateway 來設定跨 VPC 流量。

下列程序會使用 VPC 互連來啟用 VPC 之間的流量。請遵循《Amazon Virtual Private Cloud 互連指南》中什麼是 VPC 互連？的指示。

使用 VPC 互連以啟用跨 VPC 流量

1. 設定適當的 VPC 路由規則，以確保網路流量可以雙向對流。
2. 確保資料庫執行個體安全群組可以從目錄的安全群組接收傳入流量。
3. 確保沒有網路存取控制清單 (ACL) 規則來封鎖流量。

如果另一個 AWS 帳戶擁有該目錄，您必須共用該目錄。

在 AWS 帳戶之間共用目錄

1. 遵循 Directory Service 管理指南中的教學課程：共用 AWS Managed Microsoft AD 目錄以實現無縫 EC2 網域加入，以開始與將在其中建立資料庫執行個體 AWS 的帳戶共用目錄。
2. 使用資料庫執行個體的帳戶登入 Directory Service 主控台，並確保網域具有 SHARED 狀態，再繼續。
3. 使用資料庫執行個體的帳戶登入 Directory Service 主控台時，請注意目錄 ID 值。您可以使用此目錄 ID，將資料庫執行個體加入網域。

步驟 6：建立或修改 PostgreSQL 資料庫叢集

建立或修改要搭配目錄使用的 PostgreSQL 資料庫叢集。您可以使用主控台、CLI 或 RDS API，將資料庫叢集與目錄建立關聯。您可採用下列其中一種方式來這麼做：

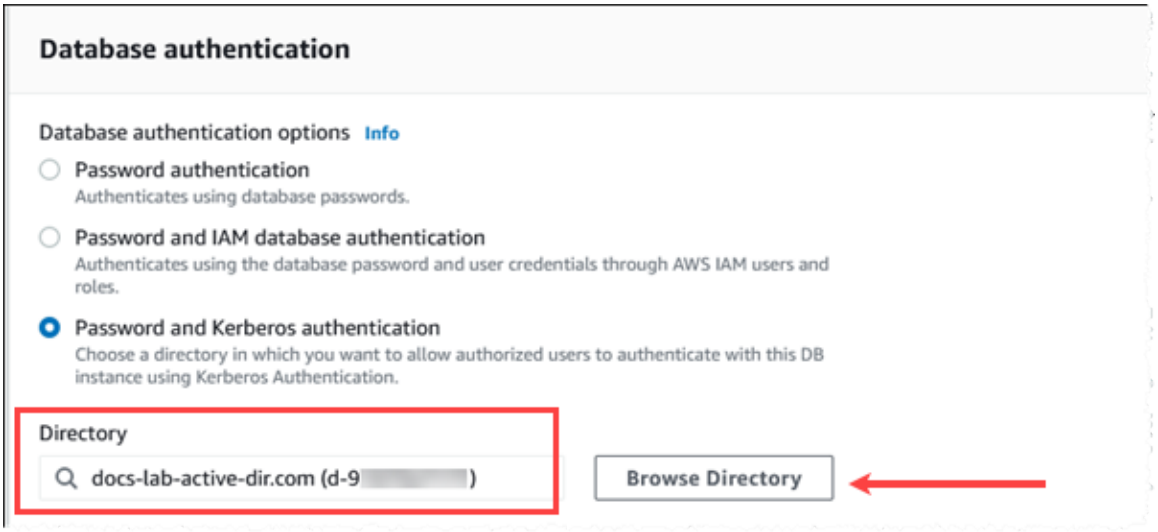
- 使用主控台、create-db-cluster CLI 命令或 CreateDBCluster RDS API 操作，建立新的 PostgreSQL 資料庫叢集。如需說明，請參閱「建立 Aurora PostgreSQL 資料庫叢集並與之連線」。
- 使用主控台、modify-db-cluster CLI 命令或 ModifyDBCluster RDS API 操作，修改現有的 PostgreSQL 資料庫叢集。如需說明，請參閱「修改 Amazon Aurora 資料庫叢集」。
- 使用主控台、restore-db-cluster-from-db-snapshot CLI 命令或 RestoreDBClusterFromDBSnapshot RDS API 操作，從資料庫快照還原 PostgreSQL 資料庫叢集。如需說明，請參閱從資料庫叢集快照還原。
- 使用主控台、 restore-db-instance-to-point-in-time CLI 命令或 RestoreDBClusterToPointInTime RDS API 操作，將 PostgreSQL 資料庫叢集還原至某個時間點。如需說明，請參閱將資料庫叢集還原至指定時間。

僅有 VPC 中的 PostgreSQL 資料庫叢集支援 Kerberos 身分驗證。資料庫叢集可在與目錄相同的 VPC 中，或在不同 VPC 中。資料庫叢集必須使用允許在目錄 VPC 內傳入和傳出的安全群組，如此資料庫叢集才能與目錄通訊。

> **Note**
>
> 從 RDS for PostgreSQL 進行遷移期間，Aurora PostgreSQL 資料庫叢集目前不支援啟用 Kerberos 身分驗證。您只能在獨立 Aurora PostgreSQL 資料庫叢集啟用 Kerberos 身分驗證。

主控台

使用主控台建立、修改或還原資料庫叢集時，請在 Database authentication (資料庫身分驗證) 區段中選擇 Kerberos authentication (Kerberos 身分驗證)。然後選擇 Browse Directory (瀏覽目錄)。選取目錄或選擇 Create a new directory (建立新目錄) 以使用 Directory Service。



AWS CLI

當您使用 時 AWS CLI，資料庫叢集需要下列參數，才能使用您建立的目錄：

- 針對 `--domain` 參數，使用您建立目錄時產生的網域識別碼 ("d-*" 識別碼)。
- 針對 `--domain-iam-role-name` 參數，使用您建立的規則，其會使用受管 IAM 政策 `AmazonRDSDirectoryServiceAccess`。

例如，下列 CLI 命令會修改資料庫叢集來使用目錄。

```
aws rds  modify-db-cluster --db-cluster-identifier mydbinstance --domain d-Directory-ID
 --domain-iam-role-name role-name
```

> **Important**
>
> 如果您修改資料庫叢集，以啟用 Kerberos 身分驗證，請在進行變更後重新啟動資料庫叢集。

步驟 7：針對您的 Kerberos 主體建立 PostgreSQL 使用者

此時，您的 Aurora PostgreSQL 資料庫叢集已加入 AWS Managed Microsoft AD 網域中。您在 步驟 4：建立和設定使用者 的目錄中建立的使用者必須設定為 PostgreSQL 資料庫使用者，並授予登入資料庫的權限。您可以透過以具有 `rds_superuser` 權限的資料庫使用者身分登入來執行此操作。例如，如果您在建立 Aurora PostgreSQL 資料庫叢集時接受預設值，請使用 `postgres`，如下列步驟中所示。

針對 Kerberos 主體建立 PostgreSQL 資料庫使用者

1. 使用 `psql` 搭配 `psql` 以連線至 Aurora PostgreSQL 資料庫叢集端點。下列範例使用 `rds_superuser` 角色的預設 `postgres` 帳戶。

   ```
   psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

2. 針對您想要其有權存取資料庫之每個 Kerberos 主體建立資料庫使用者名稱 (Active Directory 使用者名稱)。針對該使用者名稱使用 Active Directory 執行個體中定義的標準使用者名稱 (身分) (亦即小寫 `alias` (Active Directory 中的使用者名稱))，以及 Active Directory 網域的大寫名稱。Active Directory 使用者名稱是經過外部驗證的使用者，因此請使用引號括住名稱，如下所示。

   ```
   postgres=> CREATE USER "username@CORP.EXAMPLE.COM" WITH LOGIN;
   CREATE ROLE
   ```

3. 將 `rds_ad` 角色授予資料庫使用者。

   ```
   postgres=> GRANT rds_ad TO "username@CORP.EXAMPLE.COM";
   GRANT ROLE
   ```

在您完成了針對 Active Directory 使用者身分建立所有 PostgreSQL 使用者之後，使用者就可以使用其 Kerberos 憑證存取 Aurora PostgreSQL 資料庫叢集。

使用 Kerberos 進行驗證的資料庫使用者，都必須從屬於 Active Directory 網域成員的用戶端機器執行這項操作。

已獲授予 `rds_ad` 角色的資料庫使用者也無法具有 `rds_iam` 角色。這也適用於巢狀成員資格。如需詳細資訊，請參閱 IAM 資料庫身分驗證。

針對不區分大小寫的使用者名稱設定 Aurora PostgreSQL 資料庫叢集

Aurora PostgreSQL 14.5、13.8、12.12 和 11.17 版支援 `krb_caseins_users` PostgreSQL 參數。此參數支援不區分大小寫的 Active Directory 使用者名稱。根據預設，此參數會設為 false，因此 Aurora PostgreSQL 會以區分大小寫的方式解譯使用者名稱。這是 Aurora PostgreSQL 所有舊版本中的預設行為。不過，您可以在自訂資料庫叢集參數群組中將此參數設為 `true`，並允許 Aurora PostgreSQL 資料庫叢集參數群組以區分大小寫的方式解譯使用者名稱。考慮這樣做是為了方便您的資料庫使用者，他們有時可能會在使用 Active Directory 進行驗證時打錯其使用者名稱的大小寫。

若要變更 `krb_caseins_users` 參數，您的 Aurora PostgreSQL 資料庫叢集必須正在使用自訂資料庫叢集參數群組。如需使用自訂資料庫叢集參數群組的相關資訊，請參閱 Amazon Aurora 的參數群組。

您可以使用 AWS CLI 或 AWS 管理主控台 來變更設定。如需詳細資訊，請參閱在 Amazon Aurora 中修改資料庫叢集參數群組中的參數。

步驟 8：設定 PostgreSQL 用戶端

若要設定 PostgreSQL 用戶端，請執行下列步驟：

- 建立可指向網域的 krb5.conf 檔案 (或同等檔案)。
- 確認流量可以在用戶端主機與 之間流動 Directory Service。請使用 Netcat 等網路公用程式檢查以下各項：
  - 確認透過 DNS 傳送至連接埠 53 的流量。
  - 確認透過 TCP/UDP 傳送至連接埠 53 和 Kerberos 的流量，其中包括用於 Directory Service的連接埠 88 和 464。
- 確定流量可透過資料庫連接埠在用戶端主機和資料庫執行個體之間往來。例如，您可以使用 psql 來連接和存取資料庫。

以下是 的範例 krb5 AWS Managed Microsoft AD.conf 內容。

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
[domain_realm]
```

```
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

下列是內部部署 Microsoft Active Directory 的範例 krb5.conf 內容。

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
 ONPREM.COM = {
  kdc = onprem.com
  admin_server = onprem.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .onprem.com = ONPREM.COM
 onprem.com = ONPREM.COM
 .rds.amazonaws.com = EXAMPLE.COM
 .amazonaws.com.cn = EXAMPLE.COM
 .amazon.com = EXAMPLE.COM
```

在 Active Directory 網域中管理 Aurora PostgreSQL 資料庫叢集

您可以使用主控台、CLI 或 RDS API，來管理資料庫叢集，以及其與 Microsoft Active Directory 的關係。例如，您可以使 Active Directory 產生關聯，以啟用 Kerberos 身分驗證。您也可以移除 Active Directory 的關聯，以停用 Kerberos 身分驗證。您可以將要由某個 Microsoft Active Directory 外部識別的資料庫叢集移至另一個 Microsoft Active Directory。

例如，使用 CLI，您可以執行下列動作：

- 若要對失敗的成員資格重新嘗試啟用 Kerberos 身分驗證，請使用 modify-db-cluster CLI 命令。並針對 `--domain` 選項指定目前成員資格的目錄 ID。
- 若要在資料庫執行個體上停用 Kerberos 身分驗證，請使用 modify-db-cluster CLI 命令。並針對 `none` 選項指定 `--domain`。
- 若要將資料庫執行個體從某個網域移至另一個網域，請使用 modify-db-cluster CLI 命令。並針對 `--domain` 選項指定新網域的網域識別符。

了解網域成員資格

在您建立或修改資料庫叢集之後，此資料庫執行個體會成為。您可以在主控台中或執行 describe-db-instances CLI 命令，來檢視網域成員資格狀態。資料庫執行個體的狀態可以是下列其中一個：

- `kerberos-enabled` – 資料庫執行個體已啟用 Kerberos 身分驗證。
- `enabling-kerberos` – AWS 正在在此資料庫執行個體上啟用 Kerberos 身分驗證。
- `pending-enable-kerberos` – 在此資料庫執行個體上擱置 Kerberos 身分驗證的啟用。
- `pending-maintenance-enable-kerberos` – AWS 將在下一個排定的維護時段嘗試在資料庫執行個體上啟用 Kerberos 身分驗證。
- `pending-disable-kerberos` – 在此資料庫執行個體上擱置 Kerberos 身分驗證的停用。
- `pending-maintenance-disable-kerberos` – AWS 將嘗試在下一個排定的維護時段停用資料庫執行個體上的 Kerberos 身分驗證。
- `enable-kerberos-failed` – AWS 無法在資料庫執行個體上啟用 Kerberos 身分驗證的組態問題。請更正問題，然後重新發出命令來修改資料庫執行個體。
- `disabling-kerberos` – AWS 正在停用此資料庫執行個體上的 Kerberos 身分驗證。

由於網路連線問題或 IAM 角色不正確，請求啟用 Kerberos 身分驗證可能失敗。在某些情況下，當您建立或修改資料庫叢集時，嘗試啟用 Kerberos 身分驗證可能會失敗。若是如此，請確定您使用正確的 IAM 角色，然後修改要加入網域的資料庫叢集。

使用 Kerberos 身分驗證連線至 PostgreSQL

您可以使用 pgAdmin 界面或 psql 等命令列界面搭配 Kerberos 身分驗證連接至 PostgreSQL。如需連線的詳細資訊，請參閱 連接至 Amazon Aurora PostgreSQL 資料庫叢集。如需取得端點、連接埠號碼和其他連線所需詳細資訊的資訊，請參閱 檢視 Aurora 叢集的端點。

> **Note**
>
> PostgreSQL 中的 GSSAPI 身分驗證和加密是由 Kerberos 程式庫 `libkrb5.so` 實作。`postgres_fdw` 和 `dblink` 等功能也依賴此相同的程式庫，來使用 Kerberos 身分驗證或加密進行傳出連線。

pgAdmin

若要使用 Kerberos 身分驗證搭配 pgAdmin 連接至 PostgreSQL，請遵循下列步驟：

1. 在您的用戶端電腦上啟動 pgAdmin 應用程式。
2. 在 Dashboard (儀表板) 標籤上，選擇 Add New Server (新增伺服器)。
3. 在 Create - Server (建立 - 伺服器) 對話方塊中，於 General (一般) 標籤上輸入名稱，以識別 pgAdmin 中的伺服器。
4. 在 Connection (連線) 標籤上，輸入來自 Aurora PostgreSQL 資料庫的下列資訊：
   - 對於 Host (託管)，輸入 Aurora PostgreSQL 資料庫叢集的寫入器執行個體的端點。端點看起來類似下列：

     ```
     AUR-cluster-instance.111122223333.aws-region.rds.amazonaws.com
     ```

     若要從 Windows 用戶端連線至內部部署 Microsoft Active Directory，您可以使用 AWS 受管 Active Directory 的網域名稱，而不是在主機端點`rds.amazonaws.com`中。例如，假設 AWS Managed Active Directory 的網域名稱為 `corp.example.com`。然後對於 Host (託管)，端點的指定方式如下：

     ```
     AUR-cluster-instance.111122223333.aws-region.corp.example.com
     ```
   - 針對 Port (連接埠)，輸入指派的連接埠。
   - 針對 Maintenance database (維護資料庫)，輸入用戶端將連接的初始資料庫名稱。
   - 針對 Username (使用者名稱)，輸入您在 步驟 7：針對您的 Kerberos 主體建立 PostgreSQL 使用者 中為 Kerberos 身分驗證輸入的使用者名稱。
5. 選擇儲存。

Psql

若要使用 Kerberos 身分驗證搭配 psql 連接至 PostgreSQL，請遵循下列步驟：

1. 在命令提示中，執行下列命令。

   ```
   kinit username
   ```

   請將 *username* 換成使用者名稱。在提示字元中，輸入 Microsoft Active Directory 中為使用者存放的密碼。
2. 如果 PostgreSQL 資料庫叢集是使用可公開存取的 VPC，請將資料庫叢集端點的 IP 地址放入 EC2 用戶端上的 `/etc/hosts` 檔案。例如，使用下列命令取得 IP 地址，然後將該地址放入 `/etc/hosts` 檔案。

```
% dig +short PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com
;; Truncated, retrying in TCP mode.
ec2-34-210-197-118.AWS-Region.compute.amazonaws.com.
34.210.197.118

% echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com" >> /etc/
hosts
```

如果使用來自 Windows 用戶端的內部部署 Microsoft Active Directory，則需要使用特殊化的端點進行連線。使用 AWS Managed Active Directory 的網域名稱，而不是在主機端點rds.amazonaws.com中使用 Amazon 網域。

例如，假設 AWS Managed Active Directory 的網域名稱為 corp.example.com。則為端點使用 *PostgreSQL-endpoint*.*AWS-Region*.*corp.example.com* 格式並將其放入 /etc/hosts 檔案中。

```
% echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.corp.example.com" >> /etc/
hosts
```

3. 使用下列 psql 命令來登入與 Active Directory 整合的 PostgreSQL 資料庫叢集。使用叢集或執行個體端點。

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
Region.rds.amazonaws.com postgres
```

若要使用內部部署 Active Directory 從 Windows 用戶端登入 PostgreSQL 資料庫叢集，請使用下列 psql 命令搭配上一個步驟的網域名稱 (corp.example.com)：

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
Region.corp.example.com postgres
```

使用 AD 安全群組進行 Aurora PostgreSQL 存取控制

從 Aurora PostgreSQL 14.10 和 15.5 版本，可以使用 AWS Directory Service for Microsoft Active Directory (AD) 安全群組來管理 Aurora PostgreSQL 存取控制。舊版 Aurora PostgreSQL 僅支援個別使用者使用 AD 的 Kerberos 型身分驗證。每個 AD 使用者都必須明確佈建至資料庫叢集才能存取。

您可以利用 AD 安全群組，如下所示，而不是根據業務需求將每個 AD 使用者明確佈建至資料庫叢集：

- AD 使用者是 Active Directory 中各種 AD 安全群組的成員。這些不是由資料庫叢集管理員指定，而是根據業務需求，並由 AD 管理員處理。
- 資料庫叢集管理員會根據業務需求，在資料庫執行個體中建立資料庫角色。這些資料庫角色可能有不同的許可或權限。
- 資料庫叢集管理員會根據每個資料庫叢集，設定從 AD 安全群組到資料庫角色的對應。
- 資料庫使用者可以使用其 AD 憑證來存取資料庫叢集。存取權是以 AD 安全群組成員資格為基礎。AD 使用者會根據其 AD 群組成員資格自動取得或失去存取權。

先決條件

在設定 AD 安全群組的延伸之前，請確定您有下列項目：

- 為 PostgreSQL 資料庫叢集設定 Kerberos 身分驗證。如需詳細資訊，請參閱為 PostgreSQL 資料庫叢集設定 Kerberos 身分驗證。

> **Note**
>
> 對於 AD 安全群組，在此設定程序中略過步驟 7：針對您的 Kerberos 主體建立 PostgreSQL 使用者。

> **Important**
>
> 如果您在已啟用 Kerberos 身分驗證的 Aurora PostgreSQL 叢集上啟用 AD 安全群組，您可能會遇到身分驗證問題。當您將 `pg_ad_mapping` 新增至 `shared_preload_libraries` 參數並重新啟動資料庫時，就會發生這種情況。使用叢集端點時，使用非具有 `rds_ad` 角色之資料庫使用者的 AD 使用者登入嘗試可能會失敗。這也可能會導致引擎當機。若要解決此問題，請予以停用，然後在叢集上重新啟用 Kerberos 身分驗證。現有執行個體需要此解決方法，但不會影響 2025 年 4 月之後建立的執行個體。

- 管理網域中的資料庫叢集 如需詳細資訊，請參閱管理網域中的資料庫叢集。

設定 pg_ad_mapping 延伸模組

Aurora PostgreSQL 現在提供 pg_ad_mapping 延伸模組來管理 Aurora PostgreSQL 叢集中 AD 安全群組和資料庫角色之間的對應。如需 pg_ad_mapping 所提供函數的詳細資訊，請參閱 使用 pg_ad_mapping 延伸模組中的函數。

若要在 Aurora PostgreSQL 資料庫叢集上設定 pg_ad_mapping 延伸模組，您會先針對 Aurora PostgreSQL 資料庫叢集，將 pg_ad_mapping 新增至自訂資料庫叢集參數群組上的共用程式庫。如需建立自訂資料庫叢集參數群組的相關資訊，請參閱 Amazon Aurora 的參數群組。接下來，您會安裝 pg_ad_mapping 延伸模組。本節中的程序展示做法。您可以使用 AWS 管理主控台 或 AWS CLI。

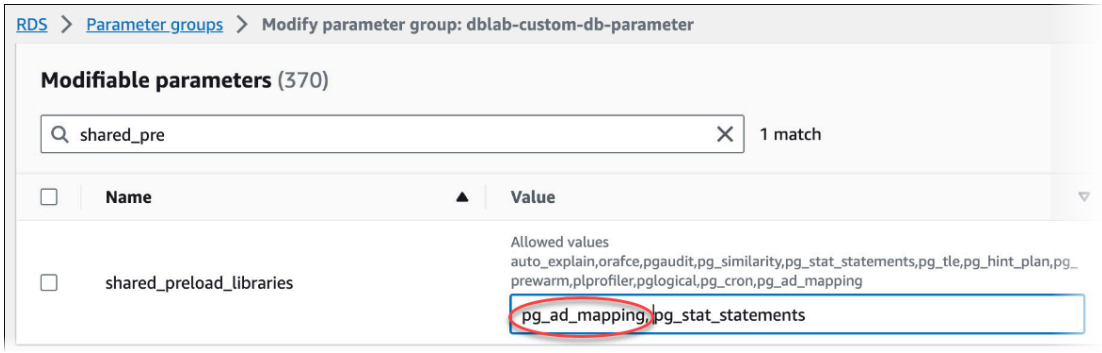
您必須具有做為 rds_superuser 角色的許可，才能執行所有這些任務。

下列步驟假設您的 Aurora PostgreSQL 資料庫叢集與自訂資料庫叢集參數群組相關聯。

主控台

若要設定 **pg_ad_mapping** 延伸模組

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。
3. 針對您的 Aurora PostgreSQL 資料庫叢集的寫入器執行個體開啟組態索引標籤。在執行個體詳細資訊之間，尋找 Parameter group (參數群組) 連結。
4. 選擇連結以開啟與 Aurora PostgreSQL 資料庫叢集相關聯的連結。
5. 在 Parameters (參數) 搜尋欄位中，輸入 shared_pre 以尋找 shared_preload_libraries 參數。
6. 選擇 Edit parameters (編輯參數) 以存取屬性值。
7. 在 Values (值) 欄位中，將 pg_ad_mapping 新增至清單。使用逗號區隔值清單中的項目。



8. 重新啟動 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，以便您對 `shared_preload_libraries` 參數所做的變更生效。
9. 當執行個體可用時，請驗證 pg_ad_mapping 是否已初始化。使用 `psql` 連線至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，然後執行下列命令。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_ad_mapping
(1 row)
```

10. 在初始化 pg_ad_mapping 之後，您現在可以建立延伸模組。您需要在初始化程式庫之後建立延伸模組，才能開始使用此延伸模組提供的函數。

```
CREATE EXTENSION pg_ad_mapping;
```

11. 關閉 `psql` 工作階段。

```
labdb=> \q
```

AWS CLI

若要設定 pg_ad_mapping

若要使用 設定 pg_ad_mapping AWS CLI，您可以呼叫 modify-db-parameter-group 操作，在自訂參數群組中新增此參數，如下列程序所示。

1. 使用下列 AWS CLI 命令將 pg_ad_mapping新增至 `shared_preload_libraries` 參數。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
```

```
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pg_ad_mapping,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. 使用以下 AWS CLI 命令重新啟動 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，以便初始化 pg_ad_mapping。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. 當執行個體可用時，您可以驗證 pg_ad_mapping 是否已初始化。使用 psql 連線至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，然後執行下列命令。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_ad_mapping
(1 row)
```

在初始化 pg_ad_mapping 之後，您現在可以建立延伸模組。

```
CREATE EXTENSION pg_ad_mapping;
```

4. 關閉 psql 工作階段，以便您可以使用 AWS CLI。

```
labdb=> \q
```

在 PowerShell 中擷取 Active Directory 群組 SID

安全識別符 (SID) 用於唯一識別安全主體或安全群組。每當在 Active Directory 中建立安全群組或帳戶時，就會指派 SID 給它。若要從作用中目錄擷取 AD 安全群組 SID，您可以從加入該 Active Directory 網域的 Windows 用戶端機器使用 Get-ADGroup cmdlet。身分參數會指定 Active Directory 群組名稱，以取得對應的 SID。

下列範例會傳回 AD 群組 *adgroup1* 的 SID。

```
C:\Users\Admin> Get-ADGroup -Identity adgroup1 | select SID
```

```
             SID
-----------------------------------------------
S-1-5-21-3168537779-1985441202-1799118680-1612
```

使用 AD 安全群組對應資料庫角色

您需要將資料庫中的 AD 安全群組明確佈建為 PostgreSQL 資料庫角色。屬於至少一個已佈建 AD 安全群組的 AD 使用者將可存取資料庫。您不應授予 `rds_ad role` AD 群組安全型資料庫角色。安全群組的 Kerberos 身分驗證將使用網域名稱尾碼觸發，例如 *`user1@example.com`*。此資料庫角色無法使用密碼或 IAM 身分驗證來存取資料庫。

> **Note**
>
> 在資料庫中具有對應資料庫角色且 `rds_ad` 角色授予給他們的 AD 使用者，無法登入為 AD 安全群組的一部分。他們將以個別使用者身分透過資料庫角色來存取。

例如，accounts-group 是 AD 中的安全群組，您想要在 Aurora PostgreSQL 中將此安全群組佈建為 accounts-role。

AD 安全群組	PosgreSQL 資料庫角色
accounts-group	accounts-role

對應資料庫角色與 AD 安全群組時，您必須確保資料庫角色已設定 LOGIN 屬性，且具有所需登入資料庫的 CONNECT 權限。

```
postgres => alter role accounts-role login;

ALTER ROLE
postgres => grant connect on database accounts-db to accounts-role;
```

管理員現在可以繼續在 AD 安全群組和 PostgreSQL 資料庫角色之間建立對應。

```
admin=>select pgadmap_set_mapping('accounts-group', 'accounts-role', <SID>, <Weight>);
```

如需擷取 AD 安全群組 SID 的資訊，請參閱 在 PowerShell 中擷取 Active Directory 群組 SID。

在某些情況下，AD 使用者可能屬於多個群組，在這種情況下，AD 使用者會繼承以最高權重佈建的資料庫角色權限。如果這兩個角色具有相同的權重，AD 使用者將繼承對應於最近所新增對應的資料庫角色權限。建議是指定反映個別資料庫角色相對許可/權限的權重。資料庫角色的許可或權限越高，應該與對應項目相關聯的權重就越高。這將避免兩個具有相同權重的對應模棱兩可。

下表顯示從 AD 安全群組到 Aurora PostgreSQL 資料庫角色的範例對應。

AD 安全群組	PosgreSQL 資料庫角色	Weight (粗細)
accounts-group	accounts-role	7
sales-group	sales-role	10
dev-group	dev-role	7

在下列範例中，`user1` 會繼承 sales-role 的權限，因為它具有較高的權重，而 `user2` 會繼承 `dev-role` 的權限，因為此角色的對應是在 `accounts-role` 之後建立的，其權重與 `accounts-role` 相同。

使用者名稱	安全群組成員資格
user1	accounts-group sales-group
user2	accounts-group dev-group

用於建立、列出和清除對應的 psql 命令，如下所示。目前，無法修改單一對應項目。需要刪除現有項目，並重新建立對應。

```
admin=>select pgadmap_set_mapping('accounts-group', 'accounts-role', 'S-1-5-67-890',
 7);
admin=>select pgadmap_set_mapping('sales-group', 'sales-role', 'S-1-2-34-560', 10);
admin=>select pgadmap_set_mapping('dev-group', 'dev-role', 'S-1-8-43-612', 7);

admin=>select * from pgadmap_read_mapping();

ad_sid       | pg_role        | weight | ad_grp
```

```
-------------+----------------+--------+---------------
S-1-5-67-890 | accounts-role | 7      | accounts-group
S-1-2-34-560 | sales-role    | 10     | sales-group
S-1-8-43-612 | dev-role      | 7      | dev-group
(3 rows)
```

AD 使用者身分記錄/稽核

使用下列命令來判斷目前或工作階段使用者繼承的資料庫角色：

```
postgres=>select session_user, current_user;

session_user | current_user
-------------+--------------
dev-role     | dev-role

(1 row)
```

若要判斷 AD 安全主體身分，請使用下列命令：

```
postgres=>select principal from pg_stat_gssapi where pid = pg_backend_pid();

 principal
-------------------------
 user1@example.com

(1 row)
```

目前，AD 使用者身分不會顯示在稽核日誌中。您可以啟用 `log_connections` 參數來記錄資料庫工作階段的建立。如需詳細資訊，請參閱 log_connections。此輸出包含 AD 使用者身分，如下所示。與此輸出相關聯的後端 PID 可以協助將動作追溯至實際 AD 使用者。

```
pgrole1@postgres:[615]:LOG: connection authorized: user=pgrole1
 database=postgres application_name=psql GSS (authenticated=yes, encrypted=yes,
 principal=Admin@EXAMPLE.COM)
```

限制

- 不支援稱為 Azure Active Directory 的 Microsoft Entra ID。

使用 **pg_ad_mapping** 延伸模組中的函數

提供的 pg_ad_mapping 延伸模組支援下列函數：

pgadmap_set_mapping

此函數會建立 AD 安全群組與具有相關聯權重的資料庫角色之間的對應。

語法

```
pgadmap_set_mapping(
ad_group,
db_role,
ad_group_sid,
weight)
```

引數

參數	Description
ad_group	AD 群組名稱。值不能為 null 或空字串。
db_role	要對應至指定 AD 群組的資料庫角色。值不能為 null 或空字串。
ad_group_sid	用於唯一識別 AD 群組的安全識別符。值以 'S-1-' 開頭，且不能為 null 或空字串。如需詳細資訊，請參閱在 PowerShell 中擷取 Active Directory 群組 SID。
Weight	與資料庫角色相關聯的權重。當使用者是多個群組的成員時，具有最高權重的角色優先。權重的預設值為 1。

傳回類型

None

使用須知

此函數會將 AD 安全群組的新對應新增至資料庫角色。它只能由具有 rds_superuser 權限的使用者在資料庫叢集的主要資料庫執行個體上執行。

範例

```
postgres=> select pgadmap_set_mapping('accounts-group','accounts-
role','S-1-2-33-12345-67890-12345-678',10);

pgadmap_set_mapping

(1 row)
```

pgadmap_read_mapping

此函數會列出 AD 安全群組與使用 pgadmap_set_mapping 函數所設定資料庫角色之間的對應。

語法

```
pgadmap_read_mapping()
```

引數

None

傳回類型

參數	Description
ad_group_sid	用於唯一識別 AD 群組的安全識別符。值以 'S-1-' 開頭，且不能為 null 或空字串。如需詳細資訊，請參閱 在 PowerShell 中擷取 Active Directory 群組 SID.accounts-role@example.com
db_role	要對應至指定 AD 群組的資料庫角色。值不能為 null 或空字串。
Weight	與資料庫角色相關聯的權重。當使用者是多個群組的成員時，具有最高權重的角色優先。權重的預設值為 1。

參數	Description
ad_group	AD 群組名稱。值不能為 null 或空字串。

使用須知

呼叫此函數，列出 AD 安全群組和資料庫角色之間的所有可用對應。

範例

```
postgres=> select * from pgadmap_read_mapping();

ad_sid                                | pg_role       | weight | ad_grp
--------------------------------------+---------------+--------+------------------
S-1-2-33-12345-67890-12345-678      | accounts-role | 10     | accounts-group
(1 row)

(1 row)
```

pgadmap_reset_mapping

此函數會重設使用 pgadmap_set_mapping 函數設定的一或多個對應。

語法

```
pgadmap_reset_mapping(
ad_group_sid,
db_role,
weight)
```

引數

參數	Description
ad_group_sid	用於唯一識別 AD 群組的安全識別符。
db_role	要對應至指定 AD 群組的資料庫角色。
Weight	與資料庫角色相關聯的權重。

如果未提供引數，則會重設資料庫角色對應的所有 AD 群組。需要提供或不提供所有引數。

傳回類型

None

使用須知

呼叫此函數來刪除資料庫角色對應的特定 AD 群組，或重設所有對應。此函數只能由具有 `rds_superuser` 權限的使用者在資料庫叢集的主要資料庫執行個體上執行。

範例

```
postgres=> select * from pgadmap_read_mapping();

   ad_sid                        | pg_role      | weight      | ad_grp
---------------------------------+--------------+-------------+-------------------
 S-1-2-33-12345-67890-12345-678 | accounts-role| 10          | accounts-group
 S-1-2-33-12345-67890-12345-666 | sales-role   | 10          | sales-group

(2 rows)
postgres=> select pgadmap_reset_mapping('S-1-2-33-12345-67890-12345-678', 'accounts-
role', 10);

pgadmap_reset_mapping
(1 row)

postgres=> select * from pgadmap_read_mapping();

   ad_sid                        | pg_role      | weight      | ad_grp
---------------------------------+--------------+-------------+---------------
 S-1-2-33-12345-67890-12345-666 | sales-role   | 10          | sales-group

(1 row)
postgres=> select pgadmap_reset_mapping();

pgadmap_reset_mapping
(1 row)

postgres=> select * from pgadmap_read_mapping();

   ad_sid                        | pg_role      | weight      | ad_grp
---------------------------------+--------------+-------------+--------------
 (0 rows)
```

將資料遷移至與 PostgreSQL 相容的 Amazon Aurora

若要從現有的資料庫將資料遷移至 Amazon Aurora PostgreSQL 相容版本資料庫叢集，您有幾個選項。遷移選項還取決於要遷移的資料庫以及要遷移的資料大小。以下是您的選項：

使用快照遷移 RDS for PostgreSQL 資料庫執行個體

您可以直接從 RDS for PostgreSQL 資料庫快照將資料遷移至 Aurora PostgreSQL 資料庫叢集。

使用 Aurora 僅供讀取複本遷移 RDS for PostgreSQL 資料庫執行個體

您也可以建立 RDS for PostgreSQL 資料庫執行個體的 Aurora PostgreSQL 僅供讀取複本，從 RDS for PostgreSQL 資料庫執行個體遷移。當 RDS for PostgreSQL 資料庫執行個體和 Aurora PostgreSQL 僅供讀取複本間的複本延遲為零時，您便可以停止複寫。此時，您可以將 Aurora 僅供讀取複本設為獨立 Aurora PostgreSQL 資料庫叢集，進行讀取和寫入。

將資料從 Amazon S3 匯入 Aurora PostgreSQL

您可以透過將資料從 Amazon S3 匯入到屬於 Aurora PostgreSQL 資料庫叢集的資料表中，以遷移資料。

從與 PostgreSQL 不相容的資料庫遷移

您可以使用 AWS Database Migration Service (AWS DMS) 從與 PostgreSQL 不相容的資料庫遷移資料。如需詳細資訊 AWS DMS，請參閱AWS Database Migration Service 《 使用者指南》中的什麼是 AWS Database Migration Service？。

> **Note**
>
> 從 RDS for PostgreSQL 進行遷移期間，Aurora PostgreSQL 資料庫叢集目前不支援啟用 Kerberos 身分驗證。您只能在獨立 Aurora PostgreSQL 資料庫叢集啟用 Kerberos 身分驗證。

如需 Aurora 可用 AWS 區域 位置的清單，請參閱《》中的 Amazon AuroraAWS 一般參考。

> **Important**
>
> 如果您打算在不久的將來將 RDS for PostgreSQL 資料庫執行個體遷移至 Aurora PostgreSQL 資料庫叢集，我們強烈建議您在遷移規劃階段的早期關閉資料庫執行個體的自動次要版本

升級。如果 RDS for PostgreSQL 版本尚未得到 Aurora PostgreSQL 支援，遷移至 Aurora PostgreSQL 可能會延遲。
如需 Aurora PostgreSQL 版本的相關資訊，請參閱 Amazon Aurora PostgreSQL 的引擎版本。

從 RDS for PostgreSQL 資料庫執行個體快照遷移到 Aurora PostgreSQL 資料庫叢集。

若要建立 Aurora PostgreSQL 資料庫叢集，您可以遷移 RDS for PostgreSQL 資料庫執行個體的資料庫快照。新的 Aurora PostgreSQL 資料庫叢集會使用來自原始 RDS for PostgreSQL 資料庫執行個體的資料填入。如需建立資料庫快照的相關資訊，請參閱建立資料庫快照。

在某些情況下，資料庫快照可能不在您要尋找資料的 AWS 區域 中。在這種情況下，請使用 Amazon RDS 主控台將資料庫快照複製到該 AWS 區域。如需複製資料庫快照的相關資訊，請參閱複製資料庫快照。

您可以遷移與指定 AWS 區域中提供的 Aurora PostgreSQL 版本相容的 RDS for PostgreSQL 快照。例如，您可以將 RDS for PostgreSQL 11.1 資料庫執行個體的快照遷移至美國西部 (加利佛尼亞北部) 區域 中的 Aurora PostgreSQL 11.4、11.7、11.8 或 11.9 版。您可以將 RDS for PostgreSQL 10.11 快照遷移至 Aurora PostgreSQL 10.11、10.12、10.13 和 10.14。換言之，RDS for PostgreSQL 快照必須使用與 Aurora PostgreSQL 相同或更低的次要版本。

您也可以選擇使用 AWS KMS key，為新的 Aurora PostgreSQL 資料庫叢集進行靜態加密。這個選項只適用於未加密的資料庫快照。

若要將 RDS for PostgreSQL 資料庫快照遷移至 Aurora PostgreSQL 資料庫叢集，您可以使用 AWS 管理主控台、 AWS CLI或 RDS API。當您使用 時 AWS 管理主控台，主控台會採取建立資料庫叢集和主要執行個體所需的動作。

主控台

使用 RDS 主控台遷移 PostgreSQL 資料庫快照

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 選擇 Snapshots (快照)。
3. 在 Snapshots (快照) 頁面上，選擇您要遷移至 Aurora PostgreSQL 資料庫叢集的 RDS for PostgreSQL 快照。

4. 選擇 Actions (動作)，然後選擇 Migrate snapshot (遷移快照)。
5. 在 Migrate database (遷移資料庫) 頁面上設定下列值：

   - DB engine version (資料庫引擎版本)：選擇要用於新遷移執行個體的資料庫引擎版本。
   - 資料庫執行個體識別符：在 AWS 區域 您選擇的 中，輸入您帳戶唯一的資料庫叢集名稱。此識別符用於資料庫叢集內執行個體的端點位址。您可以選擇將一些智慧新增至名稱，例如包含您選擇的 AWS 區域 和 資料庫引擎，例如 **aurora-cluster1**。

     該資料庫執行個體識別符有下列限制：

     - 他們必須包含 1–63 個英數字元或連字號。
     - 第一個字元必須是字母。
     - 不能以一個連字號結尾或是連續包含兩個連字號。
     - 對每個 AWS 中每個 AWS 區域帳戶的所有資料庫執行個體必須是唯一的。
   - DB instance class (資料庫執行個體類別)：為您的資料庫選擇具有所需儲存體和容量的資料庫執行個體類別，例如 `db.r6g.large`。Aurora 叢集磁碟區會隨著您的資料庫中的資料數量增加自動成長。因此，您只需選擇滿足目前儲存體需求的資料庫執行個體。如需詳細資訊，請參閱Amazon Aurora 儲存體的概觀。
   - Virtual Private Cloud (VPC)：如果具有現有的 VPC，您可以選取 VPC 識別符，例如 `vpc-a464d1c1`，將該 VPC 用於 Aurora PostgreSQL 資料庫叢集。如需有關建立 VPC 的資訊，請參閱 教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4)。

     否則，您可以選擇 Create a new VPC (建立新的 VPC)，選擇由 Amazon RDS 為您建立 VPC。
   - Subnet group (子網路群組)：如果具有現有的子網路群組，您可以選取子網路群組識別符 (例如 `gs-subnet-group1`)，搭配 Aurora PostgreSQL 資料庫叢集使用該子網路群組。
   - Public access (公有存取權)：選擇 No (否)，以指定資料庫叢集內的執行個體只能由 VPC 內的資源存取。選擇 Yes (是)，以指定公有網路上的資源可以存取資料庫叢集內的執行個體。

     > **Note**
     >
     > 生產資料庫叢集可能不必位於公有子網路中，因為只有應用程式伺服器才需要存取資料庫叢集。如果資料庫叢集不必位於公有子網路中，將 Public access (公有存取性) 設為 No (否)。

   - VPC security group (VPC 安全群組)：選擇 VPC 安全群組以允許存取您的資料庫。
   - Availability Zone (可用區域)：選擇可用區域以託管 Aurora PostgreSQL 資料庫叢集的主要執行個體。若要讓 Amazon RDS 為您選擇可用區域，請選擇 No preference (無喜好設定)。

- Database port (資料庫連接埠)：輸入要在連線至 Aurora PostgreSQL 資料庫叢集中的執行個體時使用的預設連接埠。預設值為 5432。

  > **Note**
  >
  > 您可能在公司防火牆的後面，而此防火牆不允許存取預設連接埠，例如 PostgreSQL 預設連接埠 5432。在此情況下，提供公司防火牆允許的連接埠值。稍後在連線至 Aurora PostgreSQL 資料庫叢集時，請記住該連接埠值。

- Enable Encryption (啟用加密)：選擇 Enable Encryption (啟用加密)，對新的 Aurora PostgreSQL 資料庫叢集進行靜態加密。也可以選擇 KMS 金鑰作為 AWS KMS key值。
- Auto minor version upgrade (自動小型版本升級)：選擇 Enable auto minor version upgrade (啟用自動小型版本升級)，讓 Aurora PostgreSQL 資料庫叢集自動接收可用的小型 PostgreSQL 資料庫引擎版本升級。

  Auto minor version upgrade (自動小型版本升級) 選項僅適用於 Aurora PostgreSQL 資料庫叢集的 PostgreSQL 小型引擎版本升級。不適用於為維護系統穩定性而套用的一般修補程式。

6. 選擇 Migrate (遷移) 以遷移您的資料庫快照。
7. 選擇 Databases (資料庫)，以查看新的資料庫叢集。選擇新的資料庫叢集，以監控遷移進度。移轉完成時，叢集的狀態為 Available (可用)。在 Connectivity & security (連線與安全) 標籤上，您可以找到用於連線至資料庫叢集的主要寫入器執行個體的叢集端點。如需連線至 Aurora PostgreSQL 資料庫叢集的詳細資訊，請參閱連接至 Amazon Aurora 資料庫叢集。

AWS CLI

使用 AWS CLI 將 RDS for PostgreSQL 資料庫快照遷移至 Aurora PostgreSQL 需要兩個不同的 AWS CLI 命令。首先，您可以使用 `restore-db-cluster-from-snapshot` AWS CLI 命令建立新的 Aurora PostgreSQL 資料庫叢集。然後使用 `create-db-instance` 命令在新叢集中建立主資料庫執行個體以完成遷移。以下程序使用主要資料庫執行個體建立 Aurora PostgreSQL 資料庫叢集，該資料庫執行個體的組態與用於建立快照的資料庫執行個體的組態相同。

將 RDS for PostgreSQL 資料庫快照遷移到 Aurora PostgreSQL 資料庫叢集

1. 使用 describe-db-snapshots 命令取得有關待遷移資料庫快照的資訊。您可以在命令中指定 `--db-instance-identifier` 參數或 `--db-snapshot-identifier`。如果您沒有指定其中一個參數，便會取得所有快照。

```
aws rds describe-db-snapshots --db-instance-identifier <your-db-instance-name>
```

2. 該命令會傳回從指定資料庫執行個體建立的任何快照的所有組態詳細資訊。在輸出中找到您要遷移的快照，並尋找其 Amazon 資源名稱 (ARN)。如需深入了解 Amazon RDS ARN，請參閱 Amazon Relational Database Service (Amazon RDS)。ARN 類似以下輸出。

```
“DBSnapshotArn": "arn:aws:rds:aws-region:111122223333:snapshot:<snapshot_name>"
```

此外，您也可以在輸出中找到 RDS for PostgreSQL 資料庫執行個體的組態詳細資訊，例如引擎版本、分配的儲存空間、資料庫執行個體是否加密等。

3. 使用 restore-db-cluster-from-snapshot 命令開始遷移作業。指定下列參數：

- `--db-cluster-identifier` – 您要指定給 Aurora PostgreSQL 資料庫叢集的名稱。此 Aurora 資料庫叢集是資料庫快照遷移的目標。
- `--snapshot-identifier` – 要遷移之資料庫快照的 Amazon 資源名稱 (ARN)。
- `--engine` – 指定`aurora-postgresql`Aurora 資料庫叢集引擎。
- `--kms-key-id` – 此選用參數可用來從未加密的資料庫快照建立加密的 Aurora PostgreSQL 資料庫叢集。也可用來為資料庫叢集選擇與用於資料庫快照的金鑰不同的加密金鑰。

> **Note**
>
> 您無法從已加密的資料庫快照來建立未加密的 Aurora PostgreSQL 資料庫叢集。

如果沒有指定 `--kms-key-id` 參數 (如下所示)，restore-db-cluster-from-snapshot AWS CLI 命令會建立一個空的 Aurora PostgreSQL 資料庫叢集，該資料庫叢集是使用與資料庫快照相同的金鑰進行加密，或者如果來源資料庫快照未加密則不進行加密。

對於 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier cluster-name \
    --snapshot-identifier arn:aws:rds:aws-region:111122223333:snapshot:your-
snapshot-name \
    --engine aurora-postgresql
```

在 Windows 中：

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier new_cluster ^
    --snapshot-identifier arn:aws:rds:aws-region:111122223333:snapshot:your-
snapshot-name ^
    --engine aurora-postgresql
```

4. 該命令傳回有關為遷移而建立的 Aurora PostgreSQL 資料庫叢集的詳細資訊。您可以使用 describe-db-clusters AWS CLI 命令來檢查 Aurora PostgreSQL 資料庫叢集的狀態。

```
aws rds describe-db-clusters --db-cluster-identifier cluster-name
```

5. 當資料庫叢集變為「可用」時，您可以使用 create-db-instance 命令，用以 Amazon RDS 資料庫快照為基礎的資料庫執行個體填入 Aurora PostgreSQL 資料庫叢集。指定下列參數：

   - `--db-cluster-identifier` – 上個步驟中所建立新 Aurora PostgreSQL 資料庫叢集名稱。
   - `--db-instance-identifier` – 要指定給資料庫執行個體的名稱。此執行個體將成為 Aurora PostgreSQL 資料庫叢集中的主節點。
   - `----db-instance-class` – 指定要使用的資料庫執行個體類別。選擇您要遷移到 Aurora PostgreSQL 版本支援的哪個資料庫執行個體類別。如需詳細資訊，請參閱 資料庫執行個體類別的類型 及 資料庫執行個體類別的支援資料庫引擎。
   - `--engine` – 為資料庫執行個體指定 `aurora-postgresql`。

   您也可以在命令中傳入適當的選項，以建立具有與來源資料庫快照不同組態的資料庫執行個體`create-db-instance` AWS CLI 。如需詳細資訊，請參閱 create-db-instance 命令。

   對於 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
    --db-cluster-identifier cluster-name \
    --db-instance-identifier --db-instance-class db.instance.class \
    --engine aurora-postgresql
```

   在 Windows 中：

```
aws rds create-db-instance ^
    --db-cluster-identifier cluster-name ^
    --db-instance-identifier --db-instance-class db.instance.class ^
    --engine aurora-postgresql
```

遷移程序完成後，Aurora PostgreSQL 叢集會有一個已填入的主要資料庫執行個體。

使用 Aurora 僅供讀取複本將資料從 RDS for PostgreSQL 資料庫執行個體遷移至 Aurora PostgreSQL 資料庫叢集

您可以將 Aurora 僅供讀取複本用於遷移過程，藉此使用 RDS for PostgreSQL 資料庫執行個體作為新 Aurora PostgreSQL 資料庫叢集的基礎。Aurora 僅供讀取複本選項僅適用於在相同 AWS 區域 和 帳戶中遷移，而且只有在區域為 RDS for PostgreSQL 資料庫執行個體提供相容的 Aurora PostgreSQL 版本時，才能使用。相容是指 Aurora PostgreSQL 版本與 RDS for PostgreSQL 版本相同，或為相同主要版本系列中較高的次要版本。

例如，若要使用此技術來遷移 RDS for PostgreSQL 11.14 資料庫執行個體，則該區域必須提供 PostgreSQL 第 11 版系列中，Aurora PostgreSQL 11.14 版或更高的次要版本。

主題

- 使用 Aurora 僅供讀取複本遷移資料的概觀
- 使用 Aurora 僅供讀取複本準備遷移資料
- 建立 Aurora 僅供讀取複本
- 提升 Aurora 僅供讀取複本

使用 Aurora 僅供讀取複本遷移資料的概觀

從 RDS for PostgreSQL 資料庫執行個體遷移到 Aurora PostgreSQL 資料庫叢集是多步驟程序。首先，您要建立來源 RDS for PostgreSQL 資料庫執行個體的 Aurora 僅供讀取複本。這會啟動從 RDS for PostgreSQL 資料庫執行個體複寫至特殊目的資料庫執行個體叢集 (稱為複本叢集) 的過程。複本叢集僅由 Aurora 僅供讀取複本 (讀取器執行個體) 組成。

> **Note**
>
> 每 TB 資料可能需要花費數小時才能完成遷移。

提升 Aurora PostgreSQL 複本

建立 Aurora PostgreSQL 資料庫叢集之後，請依照下列步驟來提升 Aurora 複本：

1. 停止來源 RDS for PostgreSQL 資料庫執行個體上的所有資料庫寫入工作負載。
2. 從來源 RDS for PostgreSQL 資料庫執行個體取得目前的 `WAL LSN`：

```
SELECT pg_current_wal_lsn();
pg_current_wal_lsn
--------------------
0/F0000318
(1 row)
```

3. 在 Aurora PostgreSQL 複本叢集上，檢查重播的 LSN 是否大於步驟 2 的 LSN：

```
SELECT pg_last_wal_replay_lsn();
pg_last_wal_replay_lsn
------------------------
0/F0000400
(1 row)
```

或者，您可以在來源 RDS for PostgreSQL 資料庫執行個體上使用下列查詢：

```
SELECT restart_lsn FROM pg_replication_slots;
```

4. 提升 Aurora PostgreSQL 複本叢集。

當複寫停止時，複本叢集會提升為獨立的 Aurora PostgreSQL 資料庫叢集，且會將讀取器提升為叢集的寫入器執行個體。此時，您可以將執行個體新增至 Aurora PostgreSQL 資料庫叢集，以根據您的使用案例調整大小。如果您不再需要原始 RDS for PostgreSQL 資料庫執行個體，則可以將其刪除。

如果 RDS for PostgreSQL 資料庫執行個體已具有 Aurora 僅供讀取複本，或已有跨區域僅供讀取複本，您便無法建立 Aurora 僅供讀取複本。

使用 Aurora 僅供讀取複本準備遷移資料

> **Note**
>
> 準備將資料遷移至 Aurora PostgreSQL 時，請務必適當地識別和處理未記錄的資料表。如需詳細資訊，請參閱在遷移期間處理未記錄的資料表。

在使用 Aurora 僅供讀取複本的遷移期間，任何對來源 RDS for PostgreSQL 資料庫執行個體所做的更新皆會以非同步方式複寫至複本叢集的 Aurora 僅供讀取複本。該過程會使用 PostgreSQL 的原生串流複寫功能，該功能會將預寫日誌 (WAL) 區段存放在來源執行個體上。在開始遷移前，請檢查表中所列指標的值，以確認您的執行個體擁有足夠的儲存容量。

指標	描述
`FreeStorageSpace`	可用的儲存空間。 單位：位元組
`OldestReplicationSlotLag`	延遲最久複本中 WAL 資料的延遲大小。 單位：MB
`RDSToAuroraPostgreSQLReplicaLag`	Aurora PostgreSQL 資料庫叢集落後來源 RDS 資料庫執行個體的時間長度 (以秒為單位)。
`TransactionLogsDiskUsage`	交易日誌使用的磁碟空間。 單位：MB

如需監控 RDS 執行個體的詳細資訊，請參閱 Amazon RDS 使用者指南中的監控。

建立 Aurora 僅供讀取複本

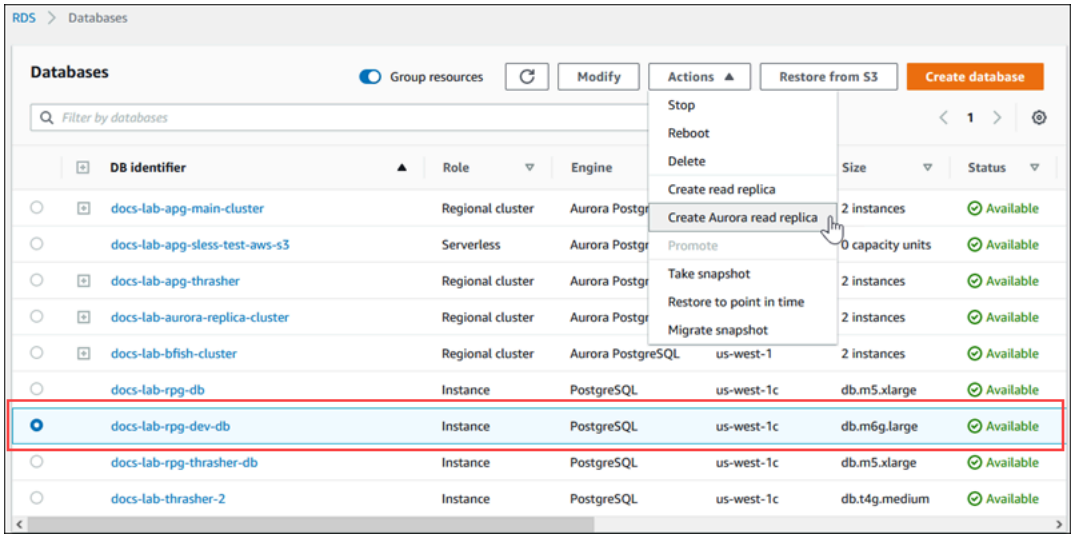
您可以使用 AWS 管理主控台 或 為 RDS for PostgreSQL 資料庫執行個體建立 Aurora 僅供讀取複本 AWS CLI。只有在 AWS 區域 提供相容的 Aurora PostgreSQL 版本時， AWS 管理主控台 才能使用 建立 Aurora 僅供讀取複本的選項。也就是說，僅在 Aurora PostgreSQL 版本與 RDS for PostgreSQL 版本相同，或為相同主要版本系列中較高的次要版本時才適用。

主控台

從來源 PostgreSQL 資料庫執行個體建立 Aurora 僅供讀取複本

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。

3. 選擇要作為 Aurora 僅供讀取複本來源的 RDS for PostgreSQL 資料庫執行個體。在 Actions (動作) 中選擇 Create Aurora read replica (建立僅供讀取複本)。若系統未顯示此選項，則表示相容的 Aurora PostgreSQL 版本在該區域中不適用。



4. 在 Create Aurora read replica settings (建立 Aurora 僅供讀取複本設定) 頁面上，您可以設定 Aurora PostgreSQL 資料庫叢集的屬性，如下表所示。複本資料庫叢集是從來源資料庫執行個體的快照所建立，使用的「主要」使用者名稱和密碼與來源相同，因此您目前無法對其進行更改。

選項	描述
DB instance class (資料庫執行個體類別)	選擇符合資料庫叢集中主要執行個體處理和記憶體要求的資料庫執行個體類別。如需詳細資訊，請參閱 Amazon Aurora 資料庫執行個體類別。
多可用區域部署	遷移期間不適用
DB instance identifier (資料庫執行個體識別碼)：	輸入您要指定給資料庫執行個體的名稱。此識別符用於新資料庫叢集內主要執行個體的端點位址。 該資料庫執行個體識別符有下列限制： • 他們必須包含 1–63 個英數字元或連字號。 • 第一個字元必須是字母。 • 不能以一個連字號結尾或是連續包含兩個連字號。 • 每個 AWS 帳戶的所有資料庫執行個體都必須是唯一的 AWS 區域。

選項	描述
Virtual Private Cloud (VPC)	選擇託管資料庫叢集的 VPC。選擇 Create new VPC (建立新的 VPC) 來讓 Amazon RDS 為您建立 VPC。如需詳細資訊，請參閱資料庫叢集先決條件。
DB subnet group (資料庫子網路群組)	選擇要用於資料庫叢集的資料庫子網路群組。選擇 Create new DB Subnet Group (建立新的資料庫子網路群組)，讓 Amazon RDS 為您建立資料庫子網路群組。如需更多詳細資訊，請參閱 資料庫叢集先決條件。
Public accessibility (公開存取性)	選擇 Yes (是) 以給予資料庫叢集一個公有 IP 地址；否則，請選擇 No (否)。資料庫叢集內的執行個體可以混合公有和私有資料庫執行個體。如需隱藏執行個體而不提供公開存取的詳細資訊，請參閱在 VPC 中的網際網路中隱藏資料庫叢集。
Availability zone (可用區域)	決定您是否要指定特定的可用區域。如需可用區域的詳細資訊，請參閱區域和可用區域。
VPC security groups (VPC 安全群組)	選擇一或多個 VPC 安全群組，保護資料庫叢集的網路存取。選擇 Create new VPC security group (建立新的 VPC 安全群組)，讓 Amazon RDS 為您建立 VPC 安全群組。如需更多詳細資訊，請參閱 資料庫叢集先決條件。
Database port (資料庫連接埠)	指定應用程式和公用程式將用於存取資料庫的連接埠。Aurora PostgreSQL 資料庫叢集預設會使用 PostgreSQL 預設連接埠 5432。某些公司的防火牆會封鎖與此預設連接埠的連線。如果您的公司防火牆會封鎖預設連接埠，請為新的資料庫叢集選擇另一個連接埠。
DB parameter group (資料庫參數群組)	針對 Aurora PostgreSQL 資料庫叢集選擇資料庫參數群組。Aurora 有一個預設資料庫參數群組供您使用，您也可以建立自己的參數群組。如需資料庫參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。

選項	描述
DB cluster parameter group (資料庫叢集參數群組)	針對 Aurora PostgreSQL 資料庫叢集選擇資料庫叢集參數群組。Aurora 有一個預設資料庫叢集參數群組供您使用，您也可以建立自己的資料庫叢集參數群組。如需資料庫叢集參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。
加密	選擇 Enable encryption (啟用加密)，以便對新的 Aurora 資料庫叢集進行靜態加密。如果您選擇 Enable encryption (啟用加密)，則也必須選擇 KMS 加密金鑰作為 AWS KMS key值。
優先順序	選擇資料庫執行個體的容錯移轉優先順序。如果您未選擇值，則預設值為 tier-1 (第一層)。此優先順序決定從主要執行個體失敗中復原時提升 Aurora 複本的順序。如需更多詳細資訊，請參閱 Aurora 資料庫叢集的容錯能力。
Backup retention period (備份保留期間)	選擇從 1–35 天的時間長度，Aurora 會在此時間內保留資料庫備份複本。備份副本可用來進行資料庫的 point-in-time 還原 (PITR) 作業，單位可計算至秒。
Enhanced monitoring (增強型監控)	選擇 Enable enhanced monitoring (啟用增強型監控)，以針對資料庫叢集執行所在的作業系統即時收集指標。如需更多詳細資訊，請參閱 使用增強型監控來監控作業系統指標。
監控角色	只有在選擇 Enable enhanced monitoring (啟用增強型監控) 時才能使用。要用於增強型監控的 AWS Identity and Access Management (IAM) 角色。如需更多詳細資訊，請參閱 設定並啟用增強型監控。
精細程度	只有在選擇 Enable enhanced monitoring (啟用增強型監控) 時才能使用。針對資料庫叢集，設定收集指標之間的間隔 (以秒為單位)。

選項	描述
Auto minor version upgrade (自動次要版本升級)	選擇 Yes (是) 可讓 Aurora PostgreSQL 資料庫叢集自動接收可用的小型 PostgreSQL 資料庫引擎版本升級。 Auto minor version upgrade (自動小型版本升級) 選項僅適用於 Aurora PostgreSQL 資料庫叢集的 PostgreSQL 小型引擎版本升級。不適用於為維護系統穩定性而套用的一般修補程式。
Maintenance window (維護時段)	選擇每週時間範圍，系統維護可在此期間進行。

5. 選擇 Create read replica (建立僅供讀取複本)。

AWS CLI

若要使用 從來源 RDS for PostgreSQL 資料庫執行個體建立 Aurora 僅供讀取複本 AWS CLI，您必須先使用 create-db-cluster CLI 命令來建立空的 Aurora 資料庫叢集。一旦資料庫叢集存在後，您可使用create-db-instance 命令來建立資料庫叢集的主要執行個體。主要執行個體是 Aurora 資料庫叢集中第一個建立的執行個體。於此狀況下，其最初建立為 RDS for PostgreSQL 資料庫執行個體的 Aurora 僅供讀取複本。當此程序結束時，您的 RDS for PostgreSQL 資料庫執行個體已有效遷移至 Aurora PostgreSQL 資料庫叢集。

您無須指定主要使用者帳戶 (通常為 `postgres`)、其密碼或資料庫名稱。Aurora 僅供讀取複本會自動從您叫用 AWS CLI 命令時識別的來源 RDS for PostgreSQL 資料庫執行個體取得這些複本。

您確實需要指定用於 Aurora PostgreSQL 資料庫叢集和資料庫執行個體的引擎版本。您指定的版本應與來源 RDS for PostgreSQL 資料庫執行個體相符。若來源 RDS for PostgreSQL 資料庫執行個體已加密，您還需要指定 Aurora PostgreSQL 資料庫叢集主要執行個體的加密。不支援將加密的執行個體遷移至未加密的 Aurora 資料庫叢集。

下列範例建立名為 `my-new-aurora-cluster` 的 Aurora PostgreSQL 資料庫叢集，此將使用未加密的 RDS 資料庫來源執行個體。您要先呼叫 `create-db-cluster` CLI 命令來建立 Aurora PostgreSQL 資料庫叢集。範例顯示如何使用選用 `--storage-encrypted` 參數來指定應加密的資料庫叢集。由於來源資料庫並未加密，因此 `--kms-key-id` 用來指定要使用的金鑰。如需有關必要參數和選用參數的詳細資訊，請參閱範例後面的清單。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster  \
```

```
    --db-cluster-identifier my-new-aurora-cluster \
    --db-subnet-group-name my-db-subnet
    --vpc-security-group-ids sg-11111111
    --engine aurora-postgresql \
    --engine-version same-as-your-rds-instance-version \
    --replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-
db \
    --storage-encrypted \
    --kms-key-id arn:aws:kms:aws-
region:111122223333:key/11111111-2222-3333-444444444444
```

在 Windows 中：

```
aws rds create-db-cluster  ^
    --db-cluster-identifier my-new-aurora-cluster ^
    --db-subnet-group-name my-db-subnet ^
    --vpc-security-group-ids sg-11111111 ^
    --engine aurora-postgresql ^
    --engine-version same-as-your-rds-instance-version ^
    --replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-
db ^
    --storage-encrypted ^
    --kms-key-id arn:aws:kms:aws-
region:111122223333:key/11111111-2222-3333-444444444444
```

於下列清單中，您可找到顯示於範例中某些選項的詳細資訊。除非另有說明，否則這些參數是必需的。

- `--db-cluster-identifier` – 您需要為新的 Aurora PostgreSQL 資料庫叢集指定一個名稱。
- `--db-subnet-group-name` – 在與來源資料庫執行個體相同的資料庫子網路中建立您的 Aurora PostgreSQL 資料庫叢集。
- `--vpc-security-group-ids` – 指定 Aurora PostgreSQL 資料庫叢集的安全群組。
- `--engine-version` – 指定要用於 Aurora PostgreSQL 資料庫叢集的版本。該版本應與來源 RDS for PostgreSQL 資料庫執行個體所使用的版本相同，或為更高的次要版本。
- `--replication-source-identifier` - 使用其 Amazon 資源名稱 (ARN) 識別 RDS for PostgreSQL 資料庫執行個體。如需 Amazon RDS ARN 的詳細資訊，請參閱您資料庫叢集之 AWS 一般參考 中的 Amazon Relational Database Service (Amazon RDS)。
- `--storage-encrypted` - 選用。僅在需要時使用，依如下方式指定加密：

  - 當來源資料庫執行個體具有加密儲存體時，請使用此參數。若您未將此參數用於具加密儲存體的來源資料庫執行個體，則對 create-db-cluster 的呼叫將會失敗。若要對 Aurora PostgreSQL 資料庫叢集使用與來源資料庫執行個體不同的金鑰，則還需指定 --kms-key-id。
  - 若來源資料庫執行個體的儲存體未加密，但您希望 Aurora PostgreSQL 資料庫叢集使用加密，則使用此選項。若是如此，您還需要標識要與 --kms-key-id 參數一起使用的加密金鑰。
- --kms-key-id - 選用。使用時，您可使用金鑰的 ARN、ID、別名 ARN 或其別名來指定用於儲存加密的金鑰 (--storage-encrypted)。僅於下列狀況時需要此參數：
  - 為 Aurora PostgreSQL 資料庫叢集選擇與來源資料庫執行個體使用之金鑰不同的金鑰。
  - 如要從未加密的來源建立加密叢集。於此情況下，您需要指定 Aurora PostgreSQL 應該用於加密的金鑰。

建立 Aurora PostgreSQL 資料庫叢集後，您可使用 create-db-instance CLI 命令來建立主執行個體，如下列所示：

對於 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
    --db-cluster-identifier my-new-aurora-cluster \
    --db-instance-class db.x2g.16xlarge \
    --db-instance-identifier rpg-for-migration \
    --engine aurora-postgresql
```

在 Windows 中：

```
aws rds create-db-instance ^
    --db-cluster-identifier my-new-aurora-cluster ^
    --db-instance-class db.x2g.16xlarge ^
    --db-instance-identifier rpg-for-migration ^
    --engine aurora-postgresql
```

於下列清單中，您可找到顯示於範例中某些選項的詳細資訊。

- --db-cluster-identifier – 指定您在先前步驟中使用 create-db-instance 命令建立的 Aurora PostgreSQL 資料庫叢集名稱。
- --db-instance-class – 用於主執行個體的資料庫執行個體類別名稱，例如 db.r4.xlarge、db.t4g.medium、db.x2g.16xlarge 等。如需可用的資料庫執行個體類別清單，請參閱 資料庫執行個體類別的類型。

- `--db-instance-identifier` – 指定主要執行個體的名稱。
- `--engine aurora-postgresql` – 指定引擎的 `aurora-postgresql`。

RDS API

要從來源 RDS for PostgreSQL 資料庫執行個體建立 Aurora 僅供讀取複本，請先使用 RDS API 作業 `CreateDBCluster` 來建立從來源 RDS for PostgreSQL 資料庫執行個體所建立 Aurora 僅供讀取複本的新 Aurora 資料庫叢集。當 Aurora PostgreSQL 資料庫叢集可用時，您可以使用 `CreateDBInstance` 來建立 Aurora 資料庫叢集的主執行個體。

您無須指定主要使用者帳戶 (通常為 `postgres`)、其密碼或資料庫名稱。Aurora 僅供讀取複本會自動從使用 `ReplicationSourceIdentifier` 指定的來源 RDS for PostgreSQL 資料庫執行個體取得這些資料。

您確實需要指定用於 Aurora PostgreSQL 資料庫叢集和資料庫執行個體的引擎版本。您指定的版本應與來源 RDS for PostgreSQL 資料庫執行個體相符。若來源 RDS for PostgreSQL 資料庫執行個體已加密，您還需要指定 Aurora PostgreSQL 資料庫叢集主要執行個體的加密。不支援將加密的執行個體遷移至未加密的 Aurora 資料庫叢集。

如要建立 Aurora 僅供讀取複本的 Aurora 資料庫叢集，請使用具有下列參數的 RDS API 作業 `CreateDBCluster`：

- `DBClusterIdentifier` – 要建立的資料庫叢集名稱。
- `DBSubnetGroupName` – 要與此資料庫叢集關聯之資料庫子網路群組的名稱。
- `Engine=aurora-postgresql` – 要使用的引擎名稱。
- `ReplicationSourceIdentifier` – 來源 PostgreSQL 資料庫執行個體的 Amazon 資源名稱 (ARN)。如需 Amazon RDS ARN 的詳細資訊，請參閱 Amazon Web Services 一般參考 中的 Amazon Relational Database Service (Amazon RDS)。若 `ReplicationSourceIdentifier` 標示加密來源，Amazon RDS 將使用您的預設 KMS 金鑰，除非您使用 `KmsKeyId` 選項指定不同的金鑰。
- `VpcSecurityGroupIds` – 與此資料庫叢集關聯的 Amazon EC2 VPC 安全群組清單。
- `StorageEncrypted` – 指出加密的資料庫叢集。當您使用此參數而不指定 `ReplicationSourceIdentifier` 時，Amazon RDS 會使用您的預設 KMS 金鑰。
- `KmsKeyId` – 加密叢集的金鑰。使用時，您可使用金鑰的 ARN、ID、別名 ARN 或其別名來指定用於儲存加密的金鑰。

如需更多資訊，請參閱 Amazon RDS API 參考中的 `CreateDBCluster`。

當 Aurora 資料庫叢集可用後，您可使用 `CreateDBInstance` RDS API 作業，搭配下列參數來建立資料庫叢集的主要執行個體：

- `DBClusterIdentifier` – 資料庫叢集的名稱。
- `DBInstanceClass` – 要用於主要執行個體的執行個體類別名稱。
- `DBInstanceIdentifier` – 主要執行個體的名稱。
- `Engine=aurora-postgresql` – 要使用的引擎名稱。

如需更多資訊，請參閱 Amazon RDS API 參考中的 `CreateDBInstance`。

提升 Aurora 僅供讀取複本

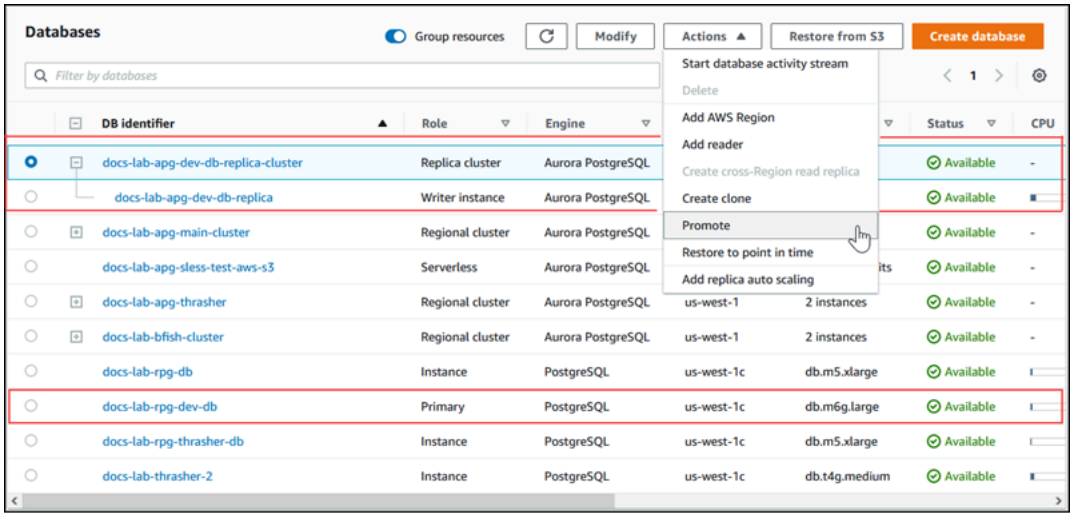
在完整遷移至 Aurora PostgreSQL 之前必須先提升複本叢集，因此目前請勿刪除 RDS for PostgreSQL 來源資料庫執行個體。

在提升複本叢集之前，請確認 RDS for PostgreSQL 資料庫執行個體沒有任何進行中的交易或其他寫入資料庫的活動。在 Aurora 僅供讀取複本上的複本延遲為零 (0) 時，您便可以提升複本叢集。如需有關監控複本延遲的詳細資訊，請參閱 監控 Aurora PostgreSQL 複寫 和 Amazon Aurora 的執行個體層級指標。

主控台

將 Aurora 僅供讀取複本提升為 Aurora 資料庫叢集

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇複本叢集。



4. 針對 Actions (動作)，選擇 Promote (提升)。這可能需要幾分鐘的時間，並可能導致停機。

該過程完成時，Aurora 複本叢集會成為區域 Aurora PostgreSQL 資料庫叢集，具有包含來自 RDS for PostgreSQL 資料庫執行個體資料的寫入器執行個體。

AWS CLI

若要將 Aurora 僅供讀取複本提升為獨立資料庫叢集，請使用 promote-read-replica-db-cluster AWS CLI 命令。

Example

對於 Linux、macOS 或 Unix：

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier myreadreplicacluster
```

在 Windows 中：

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier myreadreplicacluster
```

RDS API

若要將 Aurora 僅供讀取複本提升為獨立資料庫叢集，請使用 RDS API 操作 PromoteReadReplicaDBCluster。

在您提升複本叢集後，便可以透過檢查事件日誌來確認已完成提升，如下所示。

確認 Aurora 複本叢集已提升

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Events (事件)。
3. 請在 Events (事件) 頁面上，找到 Source (來源) 清單中的叢集名稱。每個事件都有來源、類型、時間和訊息。您可以看到在 AWS 區域 中您的帳戶已發生的所有事件。成功提升會產生以下訊息。

```
Promoted Read Replica cluster to a stand-alone database cluster.
```

提升完成後，來源 RDS for PostgreSQL 資料庫執行個體與 Aurora PostgreSQL 資料庫叢集之間的連結便會取消。您可以將用戶端應用程式導向 Aurora 僅供讀取複本的端點。如需 Aurora 端點的詳細資訊，請參閱 Amazon Aurora 端點連線。此時，如果您便可安全地刪除資料庫執行個體。

在 Aurora PostgreSQL 中最佳化查詢效能

最佳化查詢效能至關重要，因為其有助於資料庫更快速、更有效率地執行，同時使用較少的資源，進而提供更好的使用者體驗與降低營運成本。Amazon Aurora PostgreSQL 提供數種功能，可協助最佳化 PostgreSQL 工作負載的查詢效能。

主題

- 使用 Aurora Optimized Reads 改善 Aurora PostgreSQL 的查詢效能
- 最佳化 Aurora PostgreSQL 中的相互關聯子查詢
- 使用調適型聯結改善查詢效能
- 使用共用計劃快取

使用 Aurora Optimized Reads 改善 Aurora PostgreSQL 的查詢效能

您可以使用 Aurora Optimized Reads，為 Aurora PostgreSQL 實現更快的查詢處理。使用 Aurora Optimized Reads 的 Aurora PostgreSQL 資料庫執行個體可為具有大型資料集 (超過資料庫執行個體記憶體容量) 的應用程式提供高達 8 倍的查詢延遲改進，以及高達 30% 的成本節省。

主題

- PostgreSQL 中 Aurora Optimized Reads 的概觀

- 使用 Aurora Optimized Reads
- Aurora Optimized Reads 的使用案例
- 監控使用 Aurora Optimized Reads 的資料庫執行個體
- Aurora Optimized Reads 的最佳實務

PostgreSQL 中 Aurora Optimized Reads 的概觀

當您建立使用 Graviton 型 R6gd, R8gd 和 Intel 型 R6id 執行個體搭配非揮發性記憶體快速 (NVMe) 儲存體的資料庫叢集時，預設可以使用 Aurora Optimized Reads。它可從以下 PostgreSQL 版本取得：

- 適用於 R8gd 執行個體的 14.12 和更新版本、15.7 和更新版本、16.3 和更新版本、17.4 和更新版本
- 14.9 和更新版本、15.4 和更新版本、16.1 和 R6gd 和 R6id 執行個體的所有更新版本

Aurora Optimized Reads 支援兩種功能：階層式快取和暫存物件。

啟用 Optimized Reads 的階層式快取 - 使用階層式快取，您可以將資料庫執行個體快取容量擴充至執行個體記憶體的 5 倍。這會自動維護快取以包含最新的交易一致性資料，讓應用程式免於管理外部結果集型快取解決方案之資料貨幣的額外負荷。它為先前從 Aurora 儲存獲取資料的查詢提供了高達 8 倍的延遲。

在 Aurora 中，預設參數群組中的 `shared_buffers` 值通常設定為可用記憶體的大約 75%。不過，對於 r8gd、r6gd 和 r6id 執行個體類型，Aurora 會將`shared_buffers`空間減少 4.5%，以託管 Optimized Reads 快取的中繼資料。

已啟用 Optimized Reads 的暫存物件 - 使用暫存物件，您可將 PostgreSQL 產生的暫時檔案放置在本機 NVMe 儲存體上，實現更快的查詢處理。如此可減少透過網路傳送至彈性區塊儲存 (EBS) 的流量。對於進階查詢進行排序、聯結或合併不適合資料庫執行個體上可用記憶體容量的大量資料，可提供高達 2 倍的延遲和輸送量。

在 Aurora I/O 最佳化叢集上，Optimized Reads 會在 NVMe 儲存裝置上同時使用階層式快取和暫存物件。透過啟用 Optimized Reads 的階層式快取功能，Aurora 會為暫存物件分配 2 倍的執行個體記憶體、約 10% 的儲存用於內部作業，而剩餘的儲存裝置則分配為階層式快取。在 Aurora 標準叢集上，Optimized Reads 僅會使用暫存物件。

Aurora I/O 最佳化叢集可讓您使用執行個體層級的動態參數 `aurora_temp_space_size`，調整已啟用 Optimized Reads 之臨時物件的配置空間大小。此調整大小功能可從以下 PostgreSQL 版本取得：

- 16.8 版和所有更新版本

- 15.12 版和更新的 15 版本
- 14.17 版和更新的 14 版本

使用此參數，您可以調整容量大小，從 2 倍到最多 6 倍的執行個體記憶體，而不需要重新啟動資料庫引擎。當您擴展臨時物件空間時，無論並行工作負載為何，變更都會立即生效。不過，當您減少空間時，只有在臨時物件中有足夠的未使用空間來容納新的大小請求之後，調整才會完成。調整已啟用 Optimized Reads 的臨時物件大小後，分層快取會自動調整以使用任何可用的空間。

引擎	叢集儲存組態	啟用 Optimized Reads 的暫存物件	啟用 Optimized Reads 的階層式快取	支援的版本
Amazon PostgreSQL-Compatible Edition	標準	是	否	• r6gd 和 r6id 執行個體的 14.9 和更新版本、15.4 和更新版本、16.1 和所有更新版本 • 14.12 和更新版本、15.7 和更新版本、16.3 和更新版本、r8gd 執行個體的 17.4 和更新版本
	I/O 最佳化	是	是	

> **Note**
>
> 在 NVMe 型資料庫執行個體類別上的 IO 最佳化和標準叢集之間進行切換，會立即重新啟動資料庫引擎。

在 Aurora PostgreSQL 中，使用 `temp_tablespaces` 參數設定儲存臨時物件的資料表空間。

若要檢查臨時物件是否已正確設定，請使用下列命令：

```
postgres=> show temp_tablespaces;
temp_tablespaces
---------------------
aurora_temp_tablespace
(1 row)
```

aurora_temp_tablespace 是 Aurora 設定的資料表空間，指向 NVMe 本機儲存體。您無法修改此參數或切換回 Amazon EBS 儲存體。

若要檢查是否已開啟最佳化的讀取快取，請使用下列命令：

```
postgres=> show shared_preload_libraries;
                shared_preload_libraries
-----------------------------------------------------------
rdsutils,pg_stat_statements,aurora_optimized_reads_cache
```

使用 Aurora Optimized Reads

當您使用其中一個 NVMe 型資料庫執行個體佈建 Aurora PostgreSQL 資料庫執行個體時，資料庫執行個體會自動使用 Aurora Optimized Reads。

若要開啟 Aurora Optimized Reads，請執行下列其中一項：

- 使用其中一個 NVMe 型資料庫執行個體類別，建立 Aurora PostgreSQL 資料庫叢集。如需詳細資訊，請參閱建立 Amazon Aurora 資料庫叢集。
- 修改現有的 Aurora PostgreSQL 資料庫叢集，以使用其中一個 NVMe 型資料庫執行個體類別。如需詳細資訊，請參閱修改 Amazon Aurora 資料庫叢集。

Aurora Optimized Reads 適用於支援 AWS 區域 一或多個具有本機 NVMe SSD 儲存體的資料庫執行個體類別的所有 。如需詳細資訊，請參閱Amazon Aurora 資料庫執行個體類別。

若要切換回非 Optimized Reads Aurora 執行個體，請將 Aurora 執行個體的資料庫執行個體類別修改為針對資料庫工作負載沒有 NVMe 暫時性儲存的類似執行個體類別。例如，如果目前的資料庫執行個體類別是 db.r6gd.4xlarge，請選擇 db.r6g.4xlarge 以切換回來。如需詳細資訊，請參閱修改 Aurora 資料庫執行個體。

Aurora Optimized Reads 的使用案例

啟用 Optimized Reads 的階層式快取

下列是一些可從 Optimized Reads 搭配階層式快取受益的使用案例：

- 具有嚴格效能 SLA 的網際網路規模的應用程式，例如支付處理、計費、電子商務。
- 執行數百個點查詢以便收集指標/資料的即時報告儀表板。
- 具有 pgvector 擴充功能的生成式 AI 應用程式可以在數百萬個向量嵌入中搜尋精確或最近的鄰居。

啟用 Optimized Reads 的暫存物件

下列是一些可從 Optimized Reads 搭配暫存物件受益的使用案例：

- 分析查詢，包括一般資料表表達式 (CTE)、衍生資料表和群組操作。
- 處理應用程式未最佳化查詢的僅供讀取複本。
- 具有複雜操作的隨需或動態報告查詢 (例如 GROUP BY 和 ORDER BY)，無法始終使用適當的索引。
- 用於排序的 `CREATE INDEX` 或 `REINDEX` 操作。
- 使用內部暫存資料表的其他工作負載

監控使用 Aurora Optimized Reads 的資料庫執行個體

您可以使用 EXPLAIN 命令監控使用啟用 Optimized Reads 之階層式快取的查詢，如以下範例所示：

```
Postgres=> EXPLAIN (ANALYZE, BUFFERS) SELECT c FROM sbtest15 WHERE id=100000000


QUERY PLAN
------------------------------------------------------------------------------------
 Index Scan using sbtest15_pkey on sbtest15  (cost=0.57..8.59 rows=1 width=121) (actual
 time=0.287..0.288 rows=1 loops=1)
   Index Cond: (id = 100000000)
   Buffers: shared hit=3 read=2 aurora_orcache_hit=2
   I/O Timings: shared/local read=0.264
 Planning:
   Buffers: shared hit=33 read=6 aurora_orcache_hit=6
   I/O Timings: shared/local read=0.607
 Planning Time: 0.929 ms
 Execution Time: 0.303 ms
(9 rows)
Time: 2.028 ms
```

> **Note**
>
> 只有在 Optimized Reads 已開啟且其值大於零時，才會顯示解釋計畫 `Buffers` 區段中的 `aurora_orcache_hit` 和 `aurora_storage_read` 欄位。讀取欄位是 `aurora_orcache_hit` 和 `aurora_storage_read` 欄位的總計。

您可以透過下列 CloudWatch 指標，監控使用 Aurora Optimized Reads 的資料庫執行個體：

- `AuroraOptimizedReadsCacheHitRatio`
- `FreeEphemeralStorage`
- `ReadIOPSEphemeralStorage`
- `ReadLatencyEphemeralStorage`
- `ReadThroughputEphemeralStorage`
- `WriteIOPSEphemeralStorage`
- `WriteLatencyEphemeralStorage`
- `WriteThroughputEphemeralStorage`

這些指標提供可用執行個體儲存體、IOPS 和輸送量的相關資料。如需這些指標的詳細資訊，請參閱 Amazon Aurora 的執行個體層級指標。

您也可以使用 `pg_proctab` 擴充功能來監控 NVMe 儲存體。

```
postgres=>select * from pg_diskusage();

major | minor |       devname       | reads_completed | reads_merged | sectors_read |
 readtime | writes_completed | writes_merged | sectors_written | writetime | current_io
 | iotime  | totaliotime
------+-------+---------------------+-----------------+--------------+--------------
+----------+------------------+---------------+-----------------+-----------
+------------+---------+-------------
      |       | rdstemp             |           23264 |            0 |       191450 |
  11670 |          1750892 |             0 |        24540576 |    819350 |          0 |
 3847580 |      831020
```

```
      |       | rdsephemeralstorage |           23271 |            0 |       193098 |
   2620 |           114961 |             0 |        13845120 |    130770 |          0 |
  215010 |      133410
(2 rows)
```

Aurora Optimized Reads 的最佳實務

請使用 Aurora Optimized Reads 的下列最佳實務：

- 使用 CloudWatch 指標 `FreeEphemeralStorage` 監控執行個體儲存體上的可用儲存空間。如果執行個體儲存體由於資料庫執行個體的工作負載而達到其限制，請調整大量使用暫存物件的並行和查詢，或修改它以使用較大的資料庫執行個體類別。
- 監控 CloudWatch 指標以了解 Optimized Reads 快取命中率。像 VACUUM 這樣的操作可以非常快速地修改大量區塊。這可能會造成命中率暫時下降。`pg_prewarm` 擴充功能可用來將資料載入緩衝區快取，讓 Aurora 主動將其中一些區塊寫入 Optimized Reads 快取。
- 您可以啟用叢集快取管理 (CCM)，預熱第 0 層讀取器上的緩衝區快取和階層式快取，這將用作容錯移轉目標。啟用 CCM 時，會定期掃描緩衝區快取，以便在階層式快取中寫入符合移出資格的頁面。如需 CCM 的詳細資訊，請參閱 Aurora PostgreSQL 的容錯移轉後使用叢集快取管理快速復原。

最佳化 Aurora PostgreSQL 中的相互關聯子查詢

關聯的子查詢會參考外部查詢中的資料表資料欄。它會針對外部查詢傳回的每個資料列評估一次。在下列範例中，子查詢參考資料表 ot 中的資料欄。此資料表不包含在子查詢的 FROM 子句中，但會在外部查詢的 FROM 子句中參考。如果資料表 ot 有 100 萬個資料列，則需要評估 100 萬次子查詢。

```
SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT AVG(it.b) FROM it WHERE it.a = ot.a);
```

> **Note**
>
> - 從 16.8 版開始，Aurora PostgreSQL 提供子查詢轉換和子查詢快取，而 Babelfish for Aurora PostgreSQL 支援 4.2.0 版的這些功能。
> - 從 Babelfish for Aurora PostgreSQL 4.6.0 和 5.2.0 版開始，下列參數控制這些功能：
>   - babelfishpg_tsql.apg_enable_correlated_scalar_transform
>   - babelfishpg_tsql.apg_enable_subquery_cache
>
>   這兩個參數預設會開啟。

使用子查詢轉換改善 Aurora PostgreSQL 查詢效能

Aurora PostgreSQL 可以透過將相互關聯子查詢轉換為同等外部聯結來加速相互關聯子查詢。此最佳化適用於下列兩種相互關聯的子查詢類型：

- 傳回單一彙總值並出現在 SELECT 清單中的子查詢。

```
SELECT ot.a, ot.b, (SELECT AVG(it.b) FROM it WHERE it.a = ot.a) FROM ot;
```

- 傳回單一彙總值並出現在 WHERE 子句中的子查詢。

```
SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT AVG(it.b) FROM it WHERE it.a = ot.a);
```

在子查詢中啟用轉換

若要啟用相互關聯子查詢轉換為同等外部聯結，請將 `apg_enable_correlated_scalar_transform` 參數設定為 `ON`。此參數的預設值為 `OFF`。

您可以修改叢集或執行個體參數群組以設定這些參數。如需詳細資訊，請參閱 Amazon Aurora 的參數群組。

或者，您也可以使用下列命令，為目前工作階段進行設定：

```
SET apg_enable_correlated_scalar_transform TO ON;
```

確認轉換

使用 EXPLAIN 命令來確認相互關聯的子查詢是否已轉換為查詢計劃中的外部聯結。

啟用轉換時，適用的相互關聯子查詢部分會轉換為外部聯結。例如：

```
postgres=> CREATE TABLE ot (a INT, b INT);
CREATE TABLE
postgres=> CREATE TABLE it (a INT, b INT);
CREATE TABLE

postgres=> SET apg_enable_correlated_scalar_transform TO ON;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT
 AVG(it.b) FROM it WHERE it.a = ot.a);
```

```
                          QUERY PLAN
------------------------------------------------------------
 Hash Join
   Hash Cond: (ot.a = apg_scalar_subquery.scalar_output)
   Join Filter: ((ot.b)::numeric < apg_scalar_subquery.avg)
   ->  Seq Scan on ot
   ->  Hash
         ->  Subquery Scan on apg_scalar_subquery
               ->  HashAggregate
                     Group Key: it.a
                     ->  Seq Scan on it
```

當 GUC 參數切換為 OFF 時，不會轉換相同的查詢。計劃不會有外部聯結，而是子計劃。

```
postgres=> SET apg_enable_correlated_scalar_transform TO OFF;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT
 AVG(it.b) FROM it WHERE it.a = ot.a);
               QUERY PLAN
----------------------------------------
 Seq Scan on ot
   Filter: ((b)::numeric < (SubPlan 1))
   SubPlan 1
     ->  Aggregate
           ->  Seq Scan on it
                 Filter: (a = ot.a)
```

限制

- 子查詢必須位於 SELECT 清單或 WHERE 子句中的其中一個條件。否則，將不會進行轉換。
- 子查詢必須傳回彙總函數。不支援使用者定義的彙總函數進行轉換。
- 傳回運算式不是簡單彙總函數的子查詢將不會轉換。
- 子查詢 WHERE 子句中的相互關聯條件應該是簡單的資料欄參考。否則，將不會進行轉換。
- 子查詢中的相互關聯條件，其中子句必須是純等式述詞。
- 子查詢不能包含 HAVING 或 GROUP BY 子句。
- 子查詢中的 WHERE 子句可能包含一或多個與 AND 結合的述詞。

> **Note**
>
> 轉換的效能影響取決於您的結構描述、資料和工作負載。隨著外部查詢產生的資料列數增加，轉換的相互關聯子查詢執行可以大幅改善效能。強烈建議您在非生產環境中使用實際結構描述、資料和工作負載測試此功能，然後再在生產環境中啟用此功能。

使用子查詢快取來改善 Aurora PostgreSQL 查詢效能

Aurora PostgreSQL 支援子查詢快取，以存放相互關聯子查詢的結果。當子查詢結果已在快取中時，此功能會略過重複的相會關聯子查詢執行。

了解子查詢快取

PostgreSQL 的記憶節點是子查詢快取的關鍵部分。記憶節點會在本機快取中維護雜湊資料表，以從輸入參數值映射至查詢結果列。雜湊資料表的記憶體限制是 work_mem 和 hash_mem_multiplier 的乘積。若要進一步了解，請參閱資源耗用。

在查詢執行期間，子查詢快取會使用快取命中率 (CHR) 來估計快取是否改善查詢效能，並在查詢執行時間決定是否繼續使用快取。CHR 是快取命中次數與請求總數的比率。例如，如果相互關聯子查詢需要執行 100 次，而且可以從快取擷取其中 70 個執行結果，則 CHR 為 0.7。

對於每個快取遺漏的 apg_subquery_cache_check_interval 數量，檢查 CHR 是否大於 apg_subquery_cache_hit_rate_threshold 來評估子查詢快取的優點。如果沒有，快取將從記憶體中刪除，而查詢執行將返回原始、未快取的子查詢重新執行。

控制子查詢快取行為的參數

下表列出控制子查詢快取行為的參數。

參數	描述	預設	允許
apg_enable_subquery_cache	啟用相互關聯純量子查詢的快取使用。	OFF	ON、OFF
apg_subquery_cache_check_interval	設定快取未命中次數的頻率，以評估子查詢快取命中率。	500	0–2147483647

參數	描述	預設	允許
apg_subquery_cache_hit_rate_threshold	設定子查詢快取命中率的閾值。	0.3	0.0–1.0

> **Note**
>
> - 較大的 apg_subquery_cache_check_interval 值可能會改善 CHR 型快取效益估算的準確性，但會增加快取負荷，因為快取資料表有 apg_subquery_cache_check_interval 資料列之前，CHR 不會進行評估。
> - 較大的 apg_subquery_cache_hit_rate_threshold 偏差值會捨棄子查詢快取，並返回原始、未快取的子查詢重新執行。

您可以修改叢集或執行個體參數群組以設定這些參數。如需詳細資訊，請參閱 Amazon Aurora 的參數群組。

或者，您也可以使用下列命令，為目前工作階段進行設定：

```
SET apg_enable_subquery_cache TO ON;
```

在 Aurora PostgreSQL 中開啟子查詢快取

啟用子查詢快取時，Aurora PostgreSQL 會套用快取來儲存子查詢結果。然後，查詢計劃在 SubPlan 下會有記憶節點。

例如，下列命令序列顯示沒有子查詢快取之簡單相互關聯子查詢的預估查詢執行計劃。

```
postgres=> SET apg_enable_subquery_cache TO OFF;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT it.b
 FROM it WHERE it.a = ot.a);

             QUERY PLAN
-----------------------------------
 Seq Scan on ot
   Filter: (b < (SubPlan 1))
   SubPlan 1
```

```
     ->  Seq Scan on it
           Filter: (a = ot.a)
```

開啟 apg_enable_subquery_cache 後，查詢計劃會在 SubPlan 節點下包含記憶節點，表示子查詢打算使用快取。

```
postgres=> SET apg_enable_subquery_cache TO ON;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT it.b
 FROM it WHERE it.a = ot.a);

             QUERY PLAN
-----------------------------------
 Seq Scan on ot
   Filter: (b < (SubPlan 1))
   SubPlan 1
     ->  Memoize
           Cache Key: ot.a
           Cache Mode: binary
           ->  Seq Scan on it
                 Filter: (a = ot.a)
```

實際查詢執行計劃包含子查詢快取的更多詳細資訊，包括快取命中和快取遺漏。下列輸出顯示將一些值插入資料表後，上述範例查詢的實際查詢執行計劃。

```
postgres=> EXPLAIN (COSTS FALSE, TIMING FALSE, ANALYZE TRUE) SELECT ot.a, ot.b FROM ot
 WHERE ot.b < (SELECT it.b FROM it WHERE it.a = ot.a);
            QUERY PLAN
-------------------------------------------------------------------------------
 Seq Scan on ot (actual rows=2 loops=1)
   Filter: (b < (SubPlan 1))
   Rows Removed by Filter: 8
   SubPlan 1
     ->  Memoize (actual rows=0 loops=10)
           Cache Key: ot.a
           Cache Mode: binary
           Hits: 4  Misses: 6  Evictions: 0  Overflows: 0  Memory Usage: 1kB
           ->  Seq Scan on it (actual rows=0 loops=6)
                 Filter: (a = ot.a)
                 Rows Removed by Filter: 4
```

快取命中總數為 4，快取未命中總數為 6。如果命中和未命中總數少於記憶節點中的迴圈數目，表示 CHR 評估未通過，且快取已在某個時間點清除和捨棄。子查詢執行接著會返回原始未快取的重新執行。

限制

子查詢快取不支援相互關聯子查詢的特定模式。即使子查詢快取已開啟，仍會在沒有快取的情況下執行這些類型的查詢：

- IN/EXISTS/ANY/ALL 相互關聯子查詢
- 包含非確定性函數的相互關聯子查詢。
- 參考資料類型不支援雜湊或相等操作之外部資料表資料欄的相互關聯子查詢。

使用調適型聯結改善查詢效能

概觀

調適型聯結是 Aurora PostgreSQL 17.4 中的預覽功能，有助於改善查詢效能。此功能預設為停用，但您可以使用全域使用者組態 (GUC) 參數來啟用此功能。由於這是預覽功能，預設參數值可能會變更。啟用時，調適型聯結可透過在執行階段從巢狀迴圈聯結動態切換到雜湊聯結，協助最佳化查詢效能。當 PostgreSQL 最佳化工具因為不正確的基數預估而不正確地選擇巢狀迴圈聯結時，就會發生此切換。

設定調適型聯結

您可以使用這三個 GUC 參數來控制調適型聯結：

調適型聯結組態參數

GUC 參數	Description	預設值和組態選項
apg_adaptive_join_crossover_multiplier	此乘數可與資料列交叉點搭配使用，以判斷何時從巢狀迴圈切換到雜湊聯結。資料列交叉點是 SQL 最佳化工具預估巢狀迴圈和雜湊聯結操作具有相同成本的位置。較高的乘數值可降低調適型聯結切換到雜湊聯結的可能性。	控制是否已啟用調適型聯結 • 預設值：-1 (已停用) • 有效範圍：-1 到 DBL_MAX • 若要啟用：設定為 >= 1
apg_adaptive_join_cost_threshold	此參數會設定最低查詢成本閾值。對於低於此閾值的查詢，調適型聯結會	設定查詢的最低成本閾值

GUC 參數	Description	預設值和組態選項
	自行自動停用。這可避免簡單查詢的效能額外負荷，其中規劃調適型聯結的成本可能會超過從巢狀迴圈切換到雜湊聯結的優勢。	• 預設值：100 • 有效範圍：0 到 DBL_MAX

apg_enable_parameterized_adaptive_join	此參數會在啟用時，將調適型聯結功能延伸至參數化巢狀迴圈聯結。根據預設，調適型聯結僅適用於未參數化巢狀迴圈聯結，因為這些聯結更可能因為切換到雜湊聯結而獲益。參數化巢狀迴圈聯結通常效能更佳，因此切換到雜湊聯結較不重要。	控制巢狀迴圈聯結的調適型聯結行為
		• 預設值：false
		• 有效值：true/false
		  • 當為 False 時：僅適用於未參數化巢狀迴圈聯結
		  • 當為 True 時：適用於參數化和未參數化巢狀迴圈聯結
		Note
		需要先啟用 `apg_adaptive_join_crossover_multiplier`

使用共用計劃快取

概觀

Aurora PostgreSQL 使用process-per-user模型，其中每個用戶端連線都會建立專用的後端程序。每個後端程序都會針對預備陳述式維護自己的本機計劃快取。由於這些快取無法在程序之間共用，因此使用許多預備陳述式的應用程式可能會在不同的後端程序中建立重複的快取，進而增加記憶體用量。

Aurora PostgreSQL 17.6 版和更新版本以及 16.10 版和更新版本推出共用計劃快取功能。當您啟用此功能時，後端程序可以共用一般計劃，從而減少記憶體用量，並透過消除重複的計劃產生來改善效能。

共用計劃快取使用下列元件做為其快取金鑰：

- 查詢字串（包括註解）
- 規劃人員相關的 GUC 參數（包括 `search_path`)
- 使用者 ID
- 資料庫 ID

執行個體重新啟動會重設共用快取。

Parameters

下表說明控制共用計劃快取功能的參數：

參數	描述	預設	允許
`apg_shared_plan_cache.enable`	開啟或關閉共用計劃快取	0（關閉）	0, 1
`apg_shared_plan_cache.max`	快取項目的數量上限	200–1000（視執行個體大小而定）	100–50000
`apg_shared_plan_cache.min_size_per_entry`	在共用快取中存放的計劃大小下限。較小的計劃使用本機快取來最佳化 OLTP 效能。	16 KB	0–32768 (KB)
`apg_shared_plan_cache.max_size_per_entry`	共用快取的計劃大小上限。較大的計劃只會存放成本資訊。	256 KB–4 MB（視執行個體大小而定）	0–32768 (KB)

參數	描述	預設	允許
apg_shared_plan_cache.idle_generic_plan_release_timeout	閒置工作階段發佈本機一般計劃的時間。較低的值可節省記憶體；較高的值可能會改善效能。	10 秒	0–2147483647（毫秒）

> **Note**
>
> 您可以修改所有參數，而無需重新啟動。

監控檢視和函數

- apg_shared_plan_cache() – 顯示詳細的快取項目資訊（命中、有效性、時間戳記）
- apg_shared_plan_cache_stat() – 顯示執行個體層級統計資料（移出、失效）
- apg_shared_plan_cache_reset() – 移除 apg_shared_plan_cache()和 中的所有項目 apg_shared_plan_cache_stat()
- apg_shared_plan_cache_remove(cache_key) – 從項目符合 apg_shared_plan_cache()的項目中移除項目 cache_key

限制

- 僅適用於預備陳述式，不會快取 PL/pgSQL 陳述式
- 不會快取包含暫存資料表或目錄資料表的查詢
- 不會快取依賴 RLS 的查詢（資料列層級安全性）
- 每個複本都會維護自己的快取（無跨複本共用）

在 Aurora PostgreSQL 中使用未記錄的資料表

Amazon Aurora PostgreSQL 支援當機保護的未記錄資料表，即使在寫入器執行個體故障或容錯移轉之後也能維持資料完整性。在標準 PostgreSQL 中，未記錄的資料表會在寫入操作期間略過預寫日誌

(WAL)，進而加快寫入速度。不過，這會導致耐久性降低，因為未記錄的資料表不防當機，而且可能會在系統故障或不正常的關機後遺失資料。這些未記錄的資料表會在當機或不正常的關機後自動截斷。它們的內容和索引也不會複寫到待命伺服器。

相反地，由於 Aurora PostgreSQL 的分散式儲存架構，因此會以不同的方式處理未記錄的資料表。這是因為 Aurora 的儲存系統不依賴傳統的 PostgreSQL WAL 來提供耐久性。不過，在標準 PostgreSQL 中，通常與未記錄資料表相關的效能優勢在 Aurora 中可能不那麼重要。這是因為 Aurora 的分散式儲存架構，相較於標準 PostgreSQL 中使用的本機儲存，可能會帶來額外的額外負荷。

在 Aurora PostgreSQL 中使用未記錄的資料表時，請考慮下列事項：

- 您只能從 Aurora 資料庫叢集中的寫入器節點存取未記錄的資料表。
- 只有在提升為寫入器狀態時，讀取器節點才能存取未記錄的資料表。
- 當您嘗試從讀取器節點存取未記錄的資料表時，將導致下列錯誤：

```
cannot access temporary or unlogged relations during recovery.
```

建立未記錄的資料表

若要在 Aurora PostgreSQL 中建立未記錄的資料表，請在 CREATE TABLE 陳述式中新增 UNLOGGED 關鍵字：

```
CREATE UNLOGGED TABLE staging_sales_data (
    transaction_id bigint,
    customer_id bigint,
    product_id bigint,
    transaction_date date,
    amount NUMERIC
);
```

在遷移期間處理未記錄的資料表

準備將資料遷移至 Aurora PostgreSQL 時，請務必適當地識別和處理未記錄的資料表。未記錄的資料表不會進行 WAL 記錄，而且會從複寫串流中排除，這表示它們不會複寫至 Aurora 僅供讀取複本。

在建立 Aurora 僅供讀取複本之前，如果尚未使用，請將未記錄的資料表轉換為已記錄的資料表或捨棄它們。

若要確認執行個體內每個資料庫中是否存在未記錄的資料表，請使用下列命令：

```
SELECT oid, relfilenode, relname, relpersistence, relkind
FROM pg_class
WHERE relpersistence ='u';
```

若要將未記錄的資料表轉換回記錄的資料表，請使用下列命令：

```
ALTER TABLE table_name SET LOGGED;
```

此操作會重寫整個資料表，並在其上放置獨佔鎖定，直到完成為止。對於大型資料表，這可能會導致大量停機時間。

將未記錄的資料表轉換為已記錄的資料表

當您需要將未記錄的資料表轉換回記錄的資料表時，您可以使用下列命令：

```
ALTER TABLE table_name SET LOGGED;
```

此操作會重寫整個資料表，並在其上放置獨佔鎖定，直到操作完成為止。對於大型資料表，這可能會導致大量的停機時間。

未記錄的資料表和邏輯複寫

邏輯複寫通常不包含未記錄的資料表，因為邏輯複寫依賴 WAL 來擷取和傳輸變更。根據預設，未記錄資料表的變更不會進行 WAL 記錄，且會從複寫串流中排除，因此不適合需要邏輯複寫的使用案例。不過，Aurora PostgreSQL 提供名為 `rds.logically_replicate_unlogged_tables` 的參數，可讓您控制此行為：

- 當 `rds.logically_replicate_unlogged_tables` 設為 0 (關閉) 時，未記錄的資料表會從邏輯複寫中排除。
- 當 `rds.logically_replicate_unlogged_tables` 設為 1 (開啟) 時，未記錄的資料表會包含在邏輯複寫中。

> **Note**
>
> 在 Aurora PostgreSQL 中，`rds.logically_replicate_unlogged_tables` 參數預設為版本 14 和更早版本中的 1 (開啟)，以及版本 15 和更新版本中的 0 (關閉)。

在 Amazon Aurora PostgreSQL 上使用 PostgreSQL 自動清空

強烈建議您使用自動資料清理功能，以維護 PostgreSQL 資料庫執行個體的運作狀態。自動資料清理會自動啟動 VACUUM 和 ANALYZE 指令。此功能會檢查含有大量輸入、更新或刪除元組的資料表。完成檢查後，即會透過從 PostgreSQL 資料庫移除已淘汰的資料或元組回收儲存空間。

預設情況下，使用任何預設 PostgreSQL 資料庫參數群組建立的 Aurora PostgreSQL 資料庫執行個體上，都開啟了自動清空功能。預設情況下，還會設定與自動資料清理功能相關聯的其他組態參數。因為這些預設值相當泛用，針對特定的工作負載調校與自動資料清理功能相關聯的某些參數，對您會有所幫助。

接下來，您可以找到更多有關自動清空功能以及如何在您的 Aurora PostgreSQL 資料庫執行個體上調校該功能一些參數的資訊。

主題

- 配置自動資料清理的記憶體
- 降低交易 ID 包圍的可能性
- 判斷資料庫中的資料表是否需要清理
- 判斷哪些資料表目前適合進行自動資料清理
- 判斷自動資料清理目前是否執行中且執行多久時間
- 執行手動清理凍結
- 在自動資料清理執行時重新為資料表建立索引
- 使用大型索引管理自動清空
- 影響自動資料清理的其他參數
- 設定自動資料清理參數資料表層級
- 記錄清理和自動資料清理活動
- 了解具有無效資料庫的自動清空行為
- 在 Aurora PostgreSQL 中識別並解決積極的清空封鎖程式

配置自動資料清理的記憶體

autovacuum_work_mem 參數是影響自動資料清理效能的最重要參數之一。在 Aurora PostgreSQL 第 14 版及更舊版本中，autovacuum_work_mem 參數會設定為 -1，表示改用 maintenance_work_mem 的設定。對於所有其他版本，autovacuum_work_mem 是由 GREATEST({DBInstanceClassMemory/32768}, 65536) 決定。

手動清空操作一律會使用 maintenance_work_mem 設定，預設設定為 GREATEST({DBInstanceClassMemory/63963136*1024}, 65536)，也可以使用 SET 命令在工作階段層級進行調整，以進行更精準的手動 VACUUM 操作。

autovacuum_work_mem 會決定自動清空的記憶體，以保留無效元組 (pg_stat_all_tables.n_dead_tup) 的識別符來清空索引。

執行計算以判斷 autovacuum_work_mem 參數的值時，請注意下列事項：

- 如果您將參數設得太低，則清理程序可能必須掃描資料表多次才能完成其工作。多次的掃描可能會對效能產生負面影響。對於較大的執行個體，將 maintenance_work_mem或 autovacuum_work_mem 設定為至少 1 GB 可以改善清空具有大量無效元組之資料表的效能。不過，在 PostgreSQL 第 16 版和之前版本中，清空的記憶體用量上限為 1 GB，這足以在單次通過中處理大約 1.79 億個無效元組。如果資料表的無效元組超過此值，清空將需要多次通過資料表的索引，大幅增加所需的時間。從 PostgreSQL 第 17 版開始，沒有 1 GB 的限制，而自動清空可以透過使用基數樹來處理超過 1.79 億個元組。

  元組識別符的大小為 6 個位元組。若要預估清空資料表索引所需的記憶體，請查詢 pg_stat_all_tables.n_dead_tup 以尋找無效元組的數量，然後將此數字乘以 6，以判斷在單次通過中清空索引所需的記憶體。您可以使用下列查詢：

  ```
  SELECT
      relname AS table_name,
      n_dead_tup,
      pg_size_pretty(n_dead_tup * 6) AS estimated_memory
  FROM
      pg_stat_all_tables
  WHERE
      relname = 'name_of_the_table';
  ```

- autovacuum_work_mem 參數會搭配 autovacuum_max_workers 參數運作。autovacuum_max_workers 中的每個背景工作者可以使用您配置的記憶體。如果您有許多小型資料表，請配置較多 autovacuum_max_workers 和較少 autovacuum_work_mem。如果您有

大型資料表 (大於 100 GB)，請配置較多記憶體和較少背景工作者處理程序。您必須配置足夠的記憶體，才能在最大的資料表上順利執行作業。因此，請確保背景工作者處理程序與記憶體的組合等於您想要配置的總記憶體。

降低交易 ID 包圍的可能性

在一些狀況中，自動資料清理相關的參數群組設定可能不夠積極以防止交易 ID 包圍。為了解決此問題，Aurora PostgreSQL 提供自動調整自動清空參數值的機制。自動清空參數彈性調整是 Aurora PostgreSQL 的孤能。PostgreSQL 文件中有詳細的 交易 ID 包圍 的說明。

預設開啟 Aurora PostgreSQL 執行個體的自動清空參數彈性調整功能，且動態參數 `rds.adaptive_autovacuum` 設定為 [開啟]。我們強烈建議您開啟此選項。不過，若要關閉參數彈性調校功能，請將參數 `rds.adaptive_autovacuum` 設定為 0 或 OFF (關閉)。

Aurora 調校自動清空參數時，交易 ID 包圍仍可能發生。我們鼓勵您實施交易 ID 包圍的Amazon CloudWatch 警報。如需詳細資訊，請參閱 AWS 資料庫部落格上的文章在 RDS for PostgreSQL 中實作交易 ID 包裝的提早警告系統。

若開啟自動資料清理參數彈性調校，當 CloudWatch 指標 `MaximumUsedTransactionIDs` 達到 `autovacuum_freeze_max_age` 參數的值或 500,000,000 時 (以較大者為準)，Amazon RDS 將開始調整自動資料清理參數。

如果資料表繼續有交易 ID 包圍的趨勢，則 Amazon RDS 會繼續調整自動資料清理的參數。每次調整都提供更多自動資料清理的資源以避免包圍。Amazon RDS 會更新下列自動資料清理相關參數：

- autovacuum_vacuum_cost_delay
- autovacuum_vacuum_cost_limit
- `autovacuum_work_mem`
- autovacuum_naptime

RDS 只會在新的值能夠使自動資料清理更為積極時修改這些參數。參數在資料庫執行個體上的記憶體中修改。參數群組中的值不會變更。若要檢視目前的使用中記憶體設定，請使用 PostgreSQL SHOW (顯示) SQL 指令。

當 Amazon RDS 修改任何自動資料清理參數時，即會對受影響的資料庫執行個體產生事件。此事件會顯示在 上， AWS 管理主控台 並透過 Amazon RDS API 顯示。`MaximumUsedTransactionIDs` CloudWatch 指標回到閥值以下時，Amazon RDS 就會將記憶體中的自動資料清理參數重設回參數群組中指定的值。系統隨即會產生與此變更相對應的另一個事件。

判斷資料庫中的資料表是否需要清理

您可以使用以下查詢可用來顯示資料庫中未凍結的交易數目。資料庫 datfrozenxid 列的 pg_database 欄是顯示於該資料庫正常交易 ID 的下線。此欄位是資料庫內每個資料表 relfrozenxid 最小的值。

```
SELECT datname, age(datfrozenxid) FROM pg_database ORDER BY age(datfrozenxid) desc
 limit 20;
```

例如，執行上述查詢的結果可能如下所示。

```
datname    | age
mydb       | 1771757888
template0  | 1721757888
template1  | 1721757888
rdsadmin   | 1694008527
postgres   | 1693881061
(5 rows)
```

當交易 ID 包圍的存留期觸及 20 億，則會發生交易 ID 包圍 (XID)，且資料庫將變成唯獨狀態。您可使用此查詢來產生指標且一天執行數次。根據預設，自動資料清理已將交易的存留期設定為不超過 200,000,000 (autovacuum_freeze_max_age)。

範例監視策略可能如下所示：

- 設定 autovacuum_freeze_max_age 值到 2 億交易數目
- 如果資料表達到 5 億的未凍結交易，則觸發低安全性警報。這並非不合理的值，但可以表示該自動資料清理並未保持啟動狀態。
- 如果資料表存留達到 10 億，這應視為採取動作的警報。一般而言，您會基於效能理由，而想讓存留期比較接近 autovacuum_freeze_max_age。我們建議您使用下列建議調查。
- 如果資料表達到 15 億的未清理交易，則觸發高安全性警報。視資料庫使用交易的速度而定，此警報可指出系統來不及執行自動資料清理。在此情況下，我們建議您立即解決此問題。

如果資料表不斷違反這些閾值，請進一步修改自動資料清理參數。依預設，手動使用 VACUUM (已停用或以成本為基礎的延遲) 比使用預設自動資料清理更積極，但總體上對系統的干擾也更加嚴重。

我們建議下列作法：

- 請注意並開啟監控機制，如此您才會注意最舊交易的存留期。

如需建立程序以警告您交易 ID 包裝的相關資訊，請參閱 AWS 資料庫部落格文章在 Amazon RDS for PostgreSQL 中實作交易 ID 包裝的早期警告系統。

- 對於比較忙碌的資料表，除了依賴自動資料清理以外，請在維護時段定期執行手動清理凍結。如需執行手動清理凍結的資訊，請參閱 執行手動清理凍結。

判斷哪些資料表目前適合進行自動資料清理

通常有一或兩個資料表需要清理。表單中 `relfrozenxid` 的值大於 `autovacuum_freeze_max_age` 中交易的值，總是會成為自動資料清理的目標。否則，如果元組數因為最後一個 VACUUM 超出清理閾值而過時，則會清理資料表。

自動資料清理閾值已定義如下：

```
Vacuum-threshold = vacuum-base-threshold + vacuum-scale-factor * number-of-tuples
```

其中 `vacuum base threshold`是 `autovacuum_vacuum_threshold`，`vacuum scale factor`是 `autovacuum_vacuum_scale_factor`，而 `number of tuples`是 `pg_class.reltuples`。

當您連線到資料庫時，執行以下查詢可查看自動資料清理功能認為符合清理條件的資料表清單。

```
WITH vbt AS (SELECT setting AS autovacuum_vacuum_threshold FROM
pg_settings WHERE name = 'autovacuum_vacuum_threshold'),
vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor FROM
pg_settings WHERE name = 'autovacuum_vacuum_scale_factor'),
fma AS (SELECT setting AS autovacuum_freeze_max_age FROM pg_settings WHERE name =
 'autovacuum_freeze_max_age'),
sto AS (select opt_oid, split_part(setting, '=', 1) as param,
split_part(setting, '=', 2) as value from (select oid opt_oid, unnest(reloptions)
 setting from pg_class) opt)
SELECT '"'||ns.nspname||'"."'||c.relname||'"' as relation,
pg_size_pretty(pg_table_size(c.oid)) as table_size,
age(relfrozenxid) as xid_age,
coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
 autovacuum_freeze_max_age,
(coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples)
AS autovacuum_vacuum_tuples, n_dead_tup as dead_tuples FROM
pg_class c join pg_namespace ns on ns.oid = c.relnamespace
```

```
join pg_stat_all_tables stat on stat.relid = c.oid join vbt on (1=1) join vsf on (1=1)
 join fma on (1=1)
left join sto cvbt on cvbt.param = 'autovacuum_vacuum_threshold' and c.oid =
 cvbt.opt_oid
left join sto cvsf on cvsf.param = 'autovacuum_vacuum_scale_factor' and c.oid =
 cvsf.opt_oid
left join sto cfma on cfma.param = 'autovacuum_freeze_max_age' and c.oid = cfma.opt_oid
WHERE c.relkind = 'r' and nspname <> 'pg_catalog'
AND (age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
OR coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) *
c.reltuples <= n_dead_tup)
ORDER BY age(relfrozenxid) DESC LIMIT 50;
```

判斷自動資料清理目前是否執行中且執行多久時間

如果您需要手動清理資料表，請判斷目前是否在執行自動資料清理功能。若是，您可能需要調整參數，使其更有效率地執行，或暫時關閉自動資料清理功能，以便您手動執行 VACUUM。

使用以下查詢來判斷自動資料清理是否執行中、已執行多久時間，以及是否正等待另一個工作階段。

```
SELECT datname, usename, pid, state, wait_event, current_timestamp - xact_start AS
 xact_runtime, query
FROM pg_stat_activity
WHERE upper(query) LIKE '%VACUUM%'
ORDER BY xact_start;
```

在執行此查詢之後，您應會看到類似底下的輸出：

```
 datname | usename  |  pid  | state  | wait_event |      xact_runtime       | query
 --------+----------+-------+--------+------------+-------------------------
+-----------------------------------------------------------------------------------------------
 mydb    | rdsadmin | 16473 | active |            | 33 days 16:32:11.600656 |
 autovacuum: VACUUM ANALYZE public.mytable1 (to prevent wraparound)
 mydb    | rdsadmin | 22553 | active |            | 14 days 09:15:34.073141 |
 autovacuum: VACUUM ANALYZE public.mytable2 (to prevent wraparound)
 mydb    | rdsadmin | 41909 | active |            | 3 days 02:43:54.203349  |
 autovacuum: VACUUM ANALYZE public.mytable3
 mydb    | rdsadmin |   618 | active |            | 00:00:00                |
 SELECT datname, usename, pid, state, wait_event, current_timestamp - xact_start AS
 xact_runtime, query+
```

```
        |          |       |        |            |                          | FROM
pg_stat_activity
          +
        |          |       |        |            |                          | WHERE
query like '%VACUUM%'
         +
        |          |       |        |            |                          | ORDER BY
xact_start;
      +
```

有幾個問題可能造成長時間執行自動資料清理階段 (長達數天)。最常見的問題就是資料表大小或更新率的 maintenance_work_mem 參數值設得太低。

建議您依照下列公式來設定 maintenance_work_mem 參數值。

```
GREATEST({DBInstanceClassMemory/63963136*1024},65536)
```

短時間執行的自動資料清理工作階段也可以指出問題：

- 它可指出您的工作負載沒有足夠的 autovacuum_max_workers。在此情況下，您需要指出工作者數目。
- 它可指出有索引損毀 (自動資料清理會當機並以相同的關聯重新啟動，但沒有任何進度)。在這種情況下，請手動執行 vacuum freeze verbose *table* 來查看確切原因。

執行手動清理凍結

您可以在清理程序已在執行中的資料表上執行手動清理。如果您已辨識出表單的交易將近 20 億 (或超出您所監控的任何閾值) 的資料表，此功能將很實用。

下列步驟是準則，而程序會有多種變化。例如，在測試期間，假設您會發現 maintenance_work_mem 參數值設定的太小而您需要對資料表採取立即行動。不過，或許您當下不想退回執行個體。使用上個區段所列的查詢，您可判斷哪個資料表有問題，並注意長時間執行的自動資料清理工作階段。您知道您必須變更 maintenance_work_mem 參數設定，但您也需要採取立即動作並清理有問題的資料表。以下程序說明在此情況下的處理方式。

手動執行清理凍結

1. 對包含要清理之資料表的資料庫，開啟兩個工作階段。在第二個工作階段中，如果連線中斷，請使用 "screen" 或其他公用程式來維持此工作階段。
2. 在第一個工作階段中，取得在資料表上執行之自動資料清理工作階段的處理程序 ID (PID)。

執行以下查詢來取得自動資料清理工作階段的 PID。

```
SELECT datname, usename, pid, current_timestamp - xact_start
AS xact_runtime, query
FROM pg_stat_activity WHERE upper(query) LIKE '%VACUUM%' ORDER BY
xact_start;
```

3. 在第二個工作階段中，計算您需要用於此作業的記憶體數量。在此範例中，我們判斷我們可以將最多 2 GB 的記憶體使用於此作業，所以將目前工作階段的 maintenance_work_mem 設定為 2 GB。

```
SET maintenance_work_mem='2 GB';
SET
```

4. 在第二個工作階段中，對表格發佈 vacuum freeze verbose 指令。詳細資訊設定很實用，因為即使 PostgreSQL 中目前沒有此工作階段的進度報告，您仍可查看活動。

```
\timing on
Timing is on.
vacuum freeze verbose pgbench_branches;
```

```
INFO:  vacuuming "public.pgbench_branches"
INFO:  index "pgbench_branches_pkey" now contains 50 row versions in 2 pages
DETAIL:  0 index row versions were removed.
0 index pages have been deleted, 0 are currently reusable.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:  index "pgbench_branches_test_index" now contains 50 row versions in 2 pages
DETAIL:  0 index row versions were removed.
0 index pages have been deleted, 0 are currently reusable.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:  "pgbench_branches": found 0 removable, 50 nonremovable row versions
     in 43 out of 43 pages
DETAIL:  0 dead row versions cannot be removed yet.
There were 9347 unused item pointers.
0 pages are entirely empty.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
VACUUM
Time: 2.765 ms
```

5. 在第一個工作階段，如果自動清空封鎖清空工作階段，pg_stat_activity 會顯示清空工作階段的等待為 T。在此情況下，請按如下方式結束自動清空程序。

```
SELECT pg_terminate_backend('the_pid');
```

> **Note**
>
> 某些較低版本的 Amazon Aurora 無法使用上述命令終止自動清空程序，並失敗並顯示下列錯誤：`ERROR: 42501: must be a superuser to terminate superuser process LOCATION: pg_terminate_backend, signalfuncs.c:227`。若要尋找已修補的 PostgreSQL 版本，請在 Amazon Aurora PostgreSQL 更新中搜尋下列項目符號：
>
> ```
> Allow rds_superuser to terminate backends which are not explicitly
>  associated with a role
> ```

此時，您的工作階段會開始。自動清空會立即重新啟動，因為此資料表可能排在其工作清單的最前面。

6. 在第二個工作階段中啟動您的 `vacuum freeze verbose` 命令，然後在第一個工作階段中結束自動資料清理程序。

在自動資料清理執行時重新為資料表建立索引

如果索引毀損，自動資料清理會繼續處理資料表並且失敗。如果您在此情況下嘗試手動清理，則會收到如下錯誤訊息。

```
postgres=>  vacuum freeze pgbench_branches;
ERROR: index "pgbench_branches_test_index" contains unexpected
   zero page at block 30521
HINT: Please REINDEX it.
```

當索引毀損且自動資料清理嘗試在資料表上執行時，您全力應付已在執行中的自動資料清理工作階段。當您發佈 REINDEX 指令，您開啟表單上的獨佔鎖定。寫入操作遭到封鎖，也使用特定索引讀取操作。

在資料表上執行自動資料清理時重新為資料表建立索引

1. 對包含要清理之資料表的資料庫，開啟兩個工作階段。在第二個工作階段中，如果連線中斷，請使用 "screen" 或其他公用程式來維持此工作階段。

2. 在第一個工作階段中，取得在資料表上執行之自動資料清理工作階段的 PID。

   執行以下查詢來取得自動資料清理工作階段的 PID。

```
SELECT datname, usename, pid, current_timestamp - xact_start
AS xact_runtime, query
FROM pg_stat_activity WHERE upper(query) like '%VACUUM%' ORDER BY
xact_start;
```

3. 在第二個工作階段中，發出 reindex 命令。

```
\timing on
Timing is on.
reindex index pgbench_branches_test_index;
REINDEX
  Time: 9.966 ms
```

4. 在第一個工作階段中，如果自動資料清理封鎖處理程序，您會在 pg_stat_activity 中看見清理工作階段的等待為 "T"。在此情況下，您就結束自動資料清理程序。

```
SELECT pg_terminate_backend('the_pid');
```

   此時，您的工作階段會開始。請特別注意，自動資料清理會立即重新啟動，因為此資料表可能排在其工作清單的最前面。

5. 在第二個工作階段中啟動您的命令，然後在第一個工作階段中結束自動資料清理程序。

使用大型索引管理自動清空

作為其操作的一部分，自動清空會在資料表上執行時執行數個清空階段。在清除資料表之前，首先會清空其所有索引。移除多個大型索引時，此階段會耗用大量的時間和資源。因此，最佳實務是務必控制資料表上的索引數目，並清除未使用的索引。

在此程序中，首先檢查整體索引大小。然後，判斷是否有可以移除的潛在未用索引，如下列範例所示。

檢查資料表及其索引的大小

```
postgres=> select pg_size_pretty(pg_relation_size('pgbench_accounts'));
pg_size_pretty
6404 MB
(1 row)
```

```
postgres=> select pg_size_pretty(pg_indexes_size('pgbench_accounts'));
pg_size_pretty
11 GB
(1 row)
```

在此範例中，索引的大小大於資料表。這種差異可能會導致性能問題，因為索引膨脹或未使用，這會影響自動清空以及插入操作。

檢查是否有未使用的索引

使用 pg_stat_user_indexes 檢視，您可以檢查索引與 idx_scan 資料欄搭配使用的頻率。在下列範例中，未使用的索引的 idx_scan 值為 0。

```
postgres=> select * from pg_stat_user_indexes where relname = 'pgbench_accounts' order
 by idx_scan desc;

relid  | indexrelid | schemaname | relname          | indexrelname          | idx_scan
 | idx_tup_read | idx_tup_fetch
-------+------------+------------+------------------+-----------------------+----------
+--------------+---------------
16433  | 16454      | public     | pgbench_accounts | index_f               | 6
 | 6            | 0
16433  | 16450      | public     | pgbench_accounts | index_b               | 3
 | 199999       | 0
16433  | 16447      | public     | pgbench_accounts | pgbench_accounts_pkey | 0
 | 0            | 0
16433  | 16452      | public     | pgbench_accounts | index_d               | 0
 | 0            | 0
16433  | 16453      | public     | pgbench_accounts | index_e               | 0
 | 0            | 0
16433  | 16451      | public     | pgbench_accounts | index_c               | 0
 | 0            | 0
16433  | 16449      | public     | pgbench_accounts | index_a               | 0
 | 0            | 0
(7 rows)
```

```
postgres=> select schemaname, relname, indexrelname, idx_scan from pg_stat_user_indexes
 where relname = 'pgbench_accounts' order by idx_scan desc;

schemaname  | relname          | indexrelname          | idx_scan
------------+------------------+-----------------------+----------
```

```
public      | pgbench_accounts | index_f               | 6
public      | pgbench_accounts | index_b               | 3
public      | pgbench_accounts | pgbench_accounts_pkey | 0
public      | pgbench_accounts | index_d               | 0
public      | pgbench_accounts | index_e               | 0
public      | pgbench_accounts | index_c               | 0
public      | pgbench_accounts | index_a               | 0
(7 rows)
```

> **Note**
>
> 這些統計資訊是從統計資訊重設時開始累計的。假設您有僅在營業季度結束時使用的索引，或僅用於特定報告的索引。自統計資訊重設以來，可能尚未使用此索引。如需詳細資訊，請參閱統計資訊函數。用來強制執行唯一性的索引不會執行掃描，也不應將其識別為未使用的索引。若要識別未使用的索引，您應該對應用程式及其查詢有深入的理解。

若要檢查上次何時重設資料庫的統計資訊，請使用 pg_stat_database

```
postgres=> select datname, stats_reset from pg_stat_database where datname =
 'postgres';

datname   | stats_reset
----------+-------------------------------
postgres  | 2022-11-17 08:58:11.427224+00
(1 row)
```

盡快清空資料表

RDS for PostgreSQL 12 和更新版本

如果大型資料表中有太多索引，您的資料庫執行個體可能接近交易 ID 包圍 (XID)，也就是當 XID 計數器包圍至零時。若保持取消核取的狀態，此情況可能會導致資料遺失。不過，您可以在不清除索引的情況下快速清空資料表。在 RDS for PostgreSQL 12 及更新版本中，您可以使用 VACUUM 搭配 INDEX_CLEANUP 子句。

```
postgres=> VACUUM (INDEX_CLEANUP FALSE, VERBOSE TRUE) pgbench_accounts;

INFO: vacuuming "public.pgbench_accounts"
```

```
INFO: table "pgbench_accounts": found 0 removable, 8 nonremovable row versions in 1 out
 of 819673 pages
DETAIL: 0 dead row versions cannot be removed yet, oldest xmin: 7517
Skipped 0 pages due to buffer pins, 0 frozen pages.
CPU: user: 0.01 s, system: 0.00 s, elapsed: 0.01 s.
```

如果自動清空工作階段已在執行中，您必須終止它才能開始手動 VACUUM。如需執行手動清空凍結的相關資訊，請參閱 執行手動清理凍結。

> **Note**
>
> 定期略過索引清除可能會導致索引膨脹，這會降低掃描效能。索引會保留無效資料列，而資料表會保留無效資料行指標。因此，pg_stat_all_tables.n_dead_tup 會增加，直到自動清空或手動 VACUUM 搭配索引清除執行為止。作為最佳實務，只使用此程序來防止交易 ID 包圍。

RDS for PostgreSQL 11 和更舊版本

不過，在 RDS for PostgreSQL 11 和更舊版本中，允許清空更快完成的唯一方法是減少資料表上的索引數目。捨棄索引可能會影響查詢計畫。我們建議您先捨棄未使用的索引，然後在 XID 包圍非常接近時捨棄索引。在清空程序完成之後，您可以重新建立這些索引。

影響自動資料清理的其他參數

以下查詢顯示會直接影響自動資料清理及其行為的某些參數值。PostgreSQL 文件會完整說明自動資料清理參數。

```
SELECT name, setting, unit, short_desc
FROM pg_settings
WHERE name IN (
'autovacuum_max_workers',
'autovacuum_analyze_scale_factor',
'autovacuum_naptime',
'autovacuum_analyze_threshold',
'autovacuum_analyze_scale_factor',
'autovacuum_vacuum_threshold',
'autovacuum_vacuum_scale_factor',
'autovacuum_vacuum_threshold',
'autovacuum_vacuum_cost_delay',
```

```
'autovacuum_vacuum_cost_limit',
'vacuum_cost_limit',
'autovacuum_freeze_max_age',
'maintenance_work_mem',
'vacuum_freeze_min_age');
```

雖然這些全都會影響自動資料清理，其中最重要的參數如下：

- maintenance_work_mem
- autovacuum_freeze_max_age
- autovacuum_max_workers
- autovacuum_vacuum_cost_delay
- autovacuum_vacuum_cost_limit

設定自動資料清理參數資料表層級

您可以在資料表層級設定自動資料清理相關的儲存參數，這比改變整個資料庫的行為更理想。對於大型資料表，您可能需要設定積極的設定值，而且不想讓自動資料清理以那種方式對待所有的資料表。

以下查詢顯示哪些資料表目前已備妥資料表層級選項。

```
SELECT relname, reloptions
FROM pg_class
WHERE reloptions IS NOT null;
```

在比您其餘的資料表大許多的資料表範例上，此查詢可能很實用。假設您有一個 300 GB 的表單，及 30 個少於 1 GB 的其他表單。在此情況下，您可以為大型資料表設定一些特定參數，您就不會改變整個系統的行為。

```
ALTER TABLE mytable set (autovacuum_vacuum_cost_delay=0);
```

這麼做會針對此資料表關閉成本型自動資料清理延遲，代價是在您系統上使用更多資源。通常情況下，每次達到 `autovacuum_cost_limit` 時，自動資料清理都會暫停 `autovacuum_vacuum_cost_delay`。如需詳細資訊，請參閱 PostgreSQL 文件中的成本型清理。

記錄清理和自動資料清理活動

有關自動資料清理活動的資訊將根據 `rds.force_autovacuum_logging_level` 參數中指定的層級傳送到 `postgresql.log`。以下是此參數允許的值，以及預設為該值的 PostgreSQL 版本：

- disabled (PostgreSQL 10、PostgreSQL 9.6)
- debug5, debug4, debug3, debug2, debug1
- info (PostgreSQL 12、PostgreSQL 11)
- notice
- warning (PostgreSQL 13 及以上)
- error、日誌、fatal、panic

rds.force_autovacuum_logging_level 使用 log_autovacuum_min_duration 參數。log_autovacuum_min_duration 參數的值是記錄自動資料清理動作的閾值 (以毫秒為單位)。設定為 -1，表示不會記錄任何內容；設定為 0，則會記錄所有動作。如同 rds.force_autovacuum_logging_level，log_autovacuum_min_duration 的預設值取決於版本，如下所示：

- 10000 ms：PostgreSQL 14、PostgreSQL 13、PostgreSQL 12 和 PostgreSQL 11
- (empty)：PostgreSQL 10 和 PostgreSQL 9.6 沒有預設值

建議您將 rds.force_autovacuum_logging_level 設定為 WARNING。我們也建議您將 log_autovacuum_min_duration 設定為 1000 到 5000 之間的值。設定為 5000，表示會記錄時間超過 5,000 毫秒的活動。如果鎖定衝突或同時刪除關係導致跳過自動資料清理動作，-1 以外的任何設定也會記錄訊息。如需詳細資訊，請參閱 PostgreSQL 文件中的自動資料清理。

若要解決問題，您可以將 rds.force_autovacuum_logging_level 參數變更為其中一個除錯等級 (從 debug1 至 debug5)，以取得詳盡資訊。我們建議您在短時間內使用除錯設定，並且僅用於疑難排解目的。如需進一步了解，請參閱 PostgreSQL 文件中的何時記錄。

> **Note**
>
> PostgreSQL 可讓 rds_superuser 帳戶檢視 pg_stat_activity 中的自動資料清理工作階段。例如，您可以找出並終止會阻擋命令執行的自動資料清理工作階段，或是執行速度比手動發出的清理命令還要慢的自動資料清理工作階段。

了解具有無效資料庫的自動清空行為

系統會將新值 -2 引入 pg_database 目錄中的 datconnlimit 欄，以指出在 DROP DATABASE 操作中間中斷的資料庫無效。

此新值可從下列 Aurora PostgreSQL 版本取得：

- 15.4 版和所有更新版本
- 14.9 和更新版本
- 13.12 和更新版本
- 12.16 和更新版本
- 11.21 和更新版本

無效的資料庫不會影響自動清空凍結有效資料庫功能的能力。自動清空會忽略無效的資料庫。因此，定期清空操作將繼續對 PostgreSQL 環境中的所有有效資料庫正常運作且有效率地運作。

主題

- 監控交易 ID
- 調整監控查詢
- 解決無效的資料庫問題

監控交易 ID

`age(datfrozenxid)` 函數通常用於監控資料庫的交易 ID (XID) 存留期，以防止交易 ID 包圍。

由於自動清空會排除無效的資料庫，因此其交易 ID (XID) 計數器可以達到 2 `billion` 的最大值、包裝至 `-` 2 `billion`，並無限期地繼續此週期。用於監控交易 ID 包圍的典型查詢可能如下所示：

```
SELECT max(age(datfrozenxid)) FROM pg_database;
```

不過，隨著引入 `datconnlimit` 的 -2 值，無效的資料庫可能會扭曲此查詢的結果。由於這些資料庫無效，且不應成為定期維護檢查的一部分，因此可能會導致誤報，造成您認為 `age(datfrozenxid)` 高於實際值。

調整監控查詢

為了確保準確的監控，您應該調整監控查詢以排除無效的資料庫。遵循以下建議的查詢：

```
SELECT
    max(age(datfrozenxid))
FROM
```

```
    pg_database
WHERE
    datconnlimit <> -2;
```

此查詢可確保 age(datfrozenxid) 計算中只會考慮有效的資料庫，以提供 PostgreSQL 環境中交易 ID 存留期的真實反映。

解決無效的資料庫問題

嘗試連線至無效的資料庫時，您可能會遇到類似以下的錯誤訊息：

```
postgres=> \c db1
connection to server at "mydb.xxxxxxxxxx.us-west-2.rds.amazonaws.com" (xx.xx.xx.xxx),
 port xxxx failed: FATAL:  cannot connect to invalid database "db1"
HINT:  Use DROP DATABASE to drop invalid databases.
Previous connection kept
```

此外，如果 log_min_messages 參數設定為 DEBUG2 或更高版本，您可能會注意到下列日誌項目，指出自動清空程序正在略過無效的資料庫：

```
2024-07-30 05:59:00 UTC::@:[32000]:DEBUG:  autovacuum: skipping invalid database "db6"
2024-07-30 05:59:00 UTC::@:[32000]:DEBUG:  autovacuum: skipping invalid database "db1"
```

若要解決問題，請遵循連線嘗試期間提供的 HINT。使用 RDS 主帳戶或具有 rds_superuser 角色的資料庫帳戶 (RDS 主帳戶) 連線到任何有效的資料庫，並捨棄無效的資料庫。

```
SELECT
    'DROP DATABASE ' || quote_ident(datname) || ';'
FROM
    pg_database
WHERE
    datconnlimit = -2 \gexec
```

在 Aurora PostgreSQL 中識別並解決積極的清空封鎖程式

在 PostgreSQL 中，清空對於確保資料庫在回收儲存體時的運作狀態，並防止交易 ID 包圍問題至關重要。不過，有時候可以視需要防止清空操作，這可能會導致效能降低、儲存膨脹，甚至影響資料庫執行個體透過交易 ID 包圍的可用性。因此，識別和解決這些問題對於最佳的資料庫效能和可用性至關重要。閱讀了解 Amazon RDS for PostgreSQL 環境中的自動清空功能以進一步了解自動清空功能。

`postgres_get_av_diag()` 函數有助於識別防止或延遲積極清空進度的問題。提供建議，其中可能包括用於解決可辨識問題的命令，或用於無法辨識問題之進一步診斷的指引。當存留期超過 RDS 的彈性自動清空閾值 5 億筆交易 ID 時，就會報告積極的清空封鎖程式。

交易 ID 的存留期為何？

交易 ID 的 `age()` 函數會計算自資料庫 (`pg_database.datfrozenxid`) 或資料表 (`pg_class.relfrozenxid`) 最舊的未凍結交易 ID 以來發生的交易數量。此值表示自上次積極清空操作以來的資料庫活動，並強調近期 VACUUM 程序的可能工作負載。

什麼是積極清空？

積極的 VACUUM 操作會全面掃描資料表中的所有頁面，包括在一般 VACUUM 期間通常會略過的頁面。此徹底掃描旨在「凍結」接近其最長存留期的交易 ID，有效防止稱為交易 ID 包圍的情況。

若要讓 `postgres_get_av_diag()` 報告封鎖程式，封鎖程式必須至少有 5 億筆舊的交易。

主題

- 在 Aurora PostgreSQL 中安裝自動清空監控和診斷工具
- Aurora PostgreSQL 中 postgres_get_av_diag() 的函數
- 解決 Aurora PostgreSQL 中可識別的清空封鎖程式
- 解決 Aurora PostgreSQL 中無法識別的清空封鎖程式
- 解決 Aurora PostgreSQL 中的清空效能問題
- Aurora PostgreSQL 中的通知訊息說明

在 Aurora PostgreSQL 中安裝自動清空監控和診斷工具

`postgres_get_av_diag()` 函數目前可在下列 Aurora PostgreSQL 版本中使用：

- 17.4 版和更新的 17 版本
- 16.7 版和更新的 16 版本

- 15.11 版和更新的 15 版本
- 14.16 版和更新的 14 版本
- 13.19 版和更新的 13 版本

若要使用 postgres_get_av_diag()，請建立 rds_tools 延伸模組。

```
postgres=> CREATE EXTENSION rds_tools ;
CREATE EXTENSION
```

確認已安裝延伸模組。

```
postgres=> \dx rds_tools
             List of installed extensions
   Name    | Version |  Schema   |                    Description
 ----------+---------+-----------
+-------------------------------------------------------------
 rds_tools |   1.9   | rds_tools | miscellaneous administrative functions for RDS
 PostgreSQL
 1 row
```

確認已建立函數。

```
postgres=> SELECT
    proname function_name,
    pronamespace::regnamespace function_schema,
    proowner::regrole function_owner
FROM
    pg_proc
WHERE
    proname = 'postgres_get_av_diag';
    function_name     | function_schema | function_owner
----------------------+-----------------+----------------
 postgres_get_av_diag | rds_tools       | rds_superuser
(1 row)
```

Aurora PostgreSQL 中 postgres_get_av_diag() 的函數

postgres_get_av_diag() 函數會擷取在 Aurora PostgreSQL 資料庫中封鎖或落後之自動清空程序的診斷資訊。查詢需要在具有最舊交易 ID 的資料庫中執行，以獲得準確的結果。如需使用具有最舊交易 ID 的資料庫的詳細資訊，請參閱未連線至具有最舊交易 ID 存留期的資料庫

```
SELECT
    blocker,
    DATABASE,
    blocker_identifier,
    wait_event,
    TO_CHAR(autovacuum_lagging_by, 'FM9,999,999,999') AS autovacuum_lagging_by,
    suggestion,
    suggested_action
FROM (
    SELECT
        *
    FROM
        rds_tools.postgres_get_av_diag ()
    ORDER BY
        autovacuum_lagging_by DESC) q;
```

`postgres_get_av_diag()` 函數會傳回具有下列資訊的資料表。

封鎖程式

指定封鎖清空的資料庫活動類別。

- 作用中陳述式
- 交易閒置
- 備妥交易
- 邏輯複寫槽
- 讀取器執行個體
- 暫時資料表

資料庫

指定適用且支援的資料庫名稱。這是活動正在進行並封鎖或將封鎖自動清空的資料庫。這是您需要連線並採取動作的資料庫。

blocker_identifier

指定封鎖或將封鎖自動清空之活動的識別符。識別符可以是程序 ID 以及 SQL 陳述式、備妥交易、僅供讀取複本的 IP 位址，以及複寫槽的名稱，可以是邏輯或實體。

wait_event

指定封鎖工作階段的等待事件，並適用於下列封鎖程式：

- 作用中陳述式

- 交易閒置

autovacum_lagging_by

指定自動清空在每個類別的待處理項目工作中落後的交易數量。

建議

指定解決封鎖程式的建議。這些指示包括適用時活動所在的資料庫名稱、適用時工作階段的程序 ID (PID)，以及要採取的動作。

suggested_action

建議解決封鎖程式所需採取的動作。

解決 Aurora PostgreSQL 中可識別的清空封鎖程式

自動清空會執行積極的清空，並將交易 IDs 的存留期降至低於 RDS 執行個體 `autovacuum_freeze_max_age` 參數指定的閾值。您可以使用 Amazon CloudWatch 指標來追蹤此存留期。`MaximumUsedTransactionIDs`

若要尋找 Amazon RDS 執行個體的 `autovacuum_freeze_max_age` 設定 (預設為 2 億筆交易 ID)，您可以使用下列查詢：

```
SELECT
    TO_CHAR(setting::bigint, 'FM9,999,999,999') autovacuum_freeze_max_age
FROM
    pg_settings
WHERE
    name = 'autovacuum_freeze_max_age';
```

請注意，只有在存留期超過 Amazon RDS 的 5 億筆交易 ID 的彈性自動清空閾值時，`postgres_get_av_diag()` 才會檢查積極的清空封鎖程式。若要讓 `postgres_get_av_diag()` 偵測封鎖程式，封鎖程式必須至少有 5 億筆舊的交易。

`postgres_get_av_diag()` 函數會識別下列類型的封鎖程式：

主題

- 作用中陳述式
- 交易閒置
- 備妥交易
- 邏輯複寫槽

- 讀取器執行個體
- 暫時資料表

作用中陳述式

在 PostgreSQL 中，作用中陳述式是目前正在由資料庫執行的 SQL 陳述式。這包括查詢、交易或任何進行中的操作。透過 pg_stat_activity 監控時，狀態欄會指出具有對應 PID 的程序處於作用中狀態。

該 postgres_get_av_diag() 函數會在識別為作用中陳述式的陳述式時顯示類似下列的輸出。

```
blocker               | Active statement
database              | my_database
blocker_identifier    | SELECT pg_sleep(20000);
wait_event            | Timeout:PgSleep
autovacuum_lagging_by | 568,600,871
suggestion            | Connect to database "my_database", review carefully and you
 may consider terminating the process using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"SELECT pg_terminate_backend (29621);"}
```

建議動作

遵循 suggestion 欄中的指引，使用者可以連線到存在作用中陳述式的資料庫，並且如 suggested_action 欄中所指定，建議您仔細檢閱終止工作階段的選項。如果終止是安全的，您可以使用 pg_terminate_backend() 函數來終止工作階段。此動作可由管理員 (例如 RDS 主帳戶） 或具有必要pg_terminate_backend() 權限的使用者執行。

> **Warning**
>
> 終止的工作階段將復原 (ROLLBACK) 其所做的變更。根據您的需求，您可能想要重新執行陳述式。不過，建議只在自動清空程序完成其積極的清空操作之後才執行此操作。

交易閒置

交易陳述式的閒置是指已開啟明確交易 (例如透過發出 BEGIN 陳述式）、已執行一些工作，且現在正在等待用戶端透過發出 COMMIT、ROLLBACK 或 END (這會導致隱含 COMMIT) 傳遞更多工作或發出交易結束訊號的任何工作階段。

將 idle in transaction 陳述式識別為封鎖程式時，postgres_get_av_diag() 函數會顯示類似以下的輸出。

```
blocker               | idle in transaction
database              | my_database
blocker_identifier    | INSERT INTO tt SELECT * FROM tt;
wait_event            | Client:ClientRead
autovacuum_lagging_by | 1,237,201,759
suggestion            | Connect to database "my_database", review carefully and you
 may consider terminating the process using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"SELECT pg_terminate_backend (28438);"}
```

建議動作

如 suggestion 欄中所示，您可以連線到交易工作階段中存在閒置的資料庫，並使用 pg_terminate_backend() 函數終止工作階段。使用者可以是您的管理員 (RDS 主帳戶） 使用者或具有 pg_terminate_backend() 權限的使用者。

> **Warning**
>
> 終止的工作階段將復原 (ROLLBACK) 其所做的變更。根據您的需求，您可能想要重新執行陳述式。不過，建議只在自動清空程序完成其積極的清空操作之後才執行此操作。

備妥交易

PostgreSQL 允許屬於兩個階段遞交策略的交易，稱為備妥交易。這些是透過將 max_prepared_transactions 參數設定為非零值來啟用。備妥交易旨在確保交易耐用性，即使在資料庫當機、重新啟動或用戶端中斷連線後仍然可用。與一般交易一樣，它們會獲指派交易 ID，並可能影響自動清空。如果保持備妥狀態，則自動清空無法執行凍結，並可能導致交易 ID 包圍。

當交易無限期準備而不由交易管理員解決時，它們會變成孤立的備妥交易。修正此問題的唯一方法是分別使用 COMMIT PREPARED 或 ROLLBACK PREPARED 命令遞交或轉返交易。

> **Note**
>
> 請注意，在準備交易期間進行的備份在還原後仍會包含該交易。請參閱下列有關如何尋找和關閉此類交易的資訊。

該 postgres_get_av_diag() 函數會在識別為備妥交易的封鎖程式時顯示下列輸出。

```
blocker               | Prepared transaction
database              | my_database
blocker_identifier    | myptx
wait_event            | Not applicable
autovacuum_lagging_by | 1,805,802,632
suggestion            | Connect to database "my_database" and consider either COMMIT
 or ROLLBACK the prepared transaction using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"COMMIT PREPARED 'myptx';",[OR],"ROLLBACK PREPARED 'myptx';"}
```

建議動作

如建議欄中所述，連線至備妥交易所在的資料庫。根據 suggested_action 欄，仔細檢閱是否執行 COMMIT 或 ROLLBACK，以及相應動作。

若要監控一般備妥交易，PostgreSQL 提供稱為 pg_prepared_xacts 的目錄檢視。您可以使用下列查詢來尋找準備好的交易。

```
SELECT
    gid,
    prepared,
    owner,
    database,
    transaction AS oldest_xmin
FROM
    pg_prepared_xacts
ORDER BY
    age(transaction) DESC;
```

邏輯複寫槽

複寫槽的目的是保留未使用的變更，直到這些變更複寫到目標伺服器為止。如需詳細資訊，請參閱 PostgreSQL 的邏輯複寫。

邏輯複寫槽有兩種類型。

非作用中邏輯複寫槽

複寫終止時，無法移除未使用的交易日誌，且複寫槽會變成非作用中。雖然訂閱者目前未使用非作用中的邏輯複寫槽，但其會保留在伺服器上，導致 WAL 檔案的保留，並防止移除舊的交易日誌。這可能會

增加磁碟使用量，尤其是封鎖自動清空清除內部目錄資料表，因為系統必須保留 LSN 資訊以免遭到覆寫。如果未解決，這可能會導致目錄膨脹、效能降低，以及包圍清空的風險增加，進而可能導致交易停機時間。

作用中但緩慢的邏輯複寫槽

有時候，由於邏輯複寫的效能降低，移除目錄的無效元組會延遲。此複寫延遲會減緩更新 `catalog_xmin` 的速度，並可能導致目錄膨脹和包圍清空。

當 `postgres_get_av_diag()` 函數找到邏輯複寫插槽作為封鎖程式時，會顯示類似下列的輸出。

```
blocker               | Logical replication slot
database              | my_database
blocker_identifier    | slot1
wait_event            | Not applicable
autovacuum_lagging_by | 1,940,103,068
suggestion            | Ensure replication is active and resolve any lag for the slot
 if active. If inactive, consider dropping it using the command in suggested_action.
 For more information, see Working with PostgreSQL autovacuum in the Amazon RDS User
 Guide.
suggested_action      | {"SELECT pg_drop_replication_slot('slot1') FROM
 pg_replication_slots WHERE active = 'f';"}
```

建議動作

若要解決此問題，請檢查複寫組態是否有可能終止套用程序之目標結構描述或資料的問題。最常見的原因如下：

- 遺失資料欄
- 不相容的資料類型
- 資料不符
- 遺失資料表

如果問題與基礎設施問題有關：

- 網路問題 - 如何解決網路狀態不相容的 Amazon RDS 資料庫問題？
- 由於下列原因，資料庫或資料庫執行個體無法使用：
  - 複本執行個體的用盡儲存體 - 檢閱 Amazon RDS 資料庫執行個體用盡儲存體，以取得新增儲存體的相關資訊。

- 不相容參數 - 檢閱如何修正停滯在不相容參數狀態的 Amazon RDS 資料庫執行個體？以取得如何解決問題的詳細資訊。

如果您的執行個體不在 AWS 網路或 AWS EC2 上，請洽詢您的管理員，了解如何解決可用性或基礎設施相關問題。

捨棄非作用中插槽

> **Warning**
>
> 注意：捨棄複寫槽之前，請仔細確認它沒有進行中的複寫、處於非作用中狀態，而且處於無法復原的狀態。提早捨棄插槽可能會中斷複寫或導致資料遺失。

確認不再需要複寫槽後，請將其捨棄以允許自動清空繼續。條件 `active = 'f'` 可確保只會捨棄非作用中的插槽。

```
SELECT pg_drop_replication_slot('slot1') WHERE active ='f'
```

讀取器執行個體

啟用 hot_standby_feedback 設定時，可防止寫入器執行個體上的自動清空移除讀取器執行個體上執行的查詢仍可能需要的無效資料列。此行為是必要的，因為在讀取器執行個體上執行的查詢 (也適用於 Aurora 全球資料庫中的讀取器執行個體） 需要這些資料列在寫入器執行個體上保持可用，以防止查詢衝突和取消。

> **Note**
>
> `hot_standby_feedback` 預設會在 Aurora PostgreSQL 中啟用且無法修改。

當找到具有實體複寫槽作為封鎖程式的讀取複本時，`postgres_get_av_diag()` 函數會顯示類似以下的輸出。

```
blocker               | Oldest query running on aurora reader
database              | Not applicable
blocker_identifier    | my-aurora-reader-2
wait_event            | Not applicable
```

```
autovacuum_lagging_by | 540,122,859
suggestion            | Run the following query on the reader "my-aurora-reader-2" to
 find the long running query:

                      | SELECT * FROM pg_catalog.pg_stat_activity WHERE
 backend_xmin::text::bigint = 523476310;

                      | Review carefully and you may consider terminating the query on
 reader using suggested_action.
suggested_action      | {"SELECT pg_terminate_backend(pid) FROM
 pg_catalog.pg_stat_activity WHERE backend_xmin::text::bigint = 523476310;","

                      | [OR]


                      | ","Delete the reader if not needed"}
```

如 suggested_action 欄中的建議，請仔細檢閱這些選項以解除封鎖自動清空。

- 終止查詢 – 根據建議欄中的指引，您可以連線到讀取複本，如 suggested_action 欄中所指定，建議您仔細檢閱終止工作階段的選項。如果終止被視為安全，您可以使用 pg_terminate_backend( ) 函數來終止工作階段。此動作可由管理員 (例如 RDS 主帳戶) 或具有必要 pg_terminate_backend() 權限的使用者執行。

  您可以在讀取複本上執行下列 SQL 命令，以終止導致主要資料庫上的清空無法清除舊資料列的查詢。backend_xmin 的值會在函數的輸出中報告：

  ```
  SELECT
      pg_terminate_backend(pid)
  FROM
      pg_catalog.pg_stat_activity
  WHERE
      backend_xmin::text::bigint = backend_xmin;
  ```

- 如果不需要，請刪除讀取器執行個體 – 如果不再需要讀取器執行個體，您可以將其刪除。這將移除相關聯的複寫額外負荷，並允許主要資料庫回收交易日誌，而不會被執行個體保留。

暫時資料表

使用 TEMPORARY 關鍵字建立的暫時資料表位於臨時結構描述中，例如 pg_temp_xxx，並且只能由建立它們的工作階段存取。暫時資料表會在工作階段結束時捨棄。不過，PostgreSQL 的自動清空程序看

不到這些資料表，而且必須由建立它們的工作階段手動清空。嘗試從另一個工作階段清空暫時資料表沒有效果。

在異常情況下，暫時資料表存在，而沒有作用中的工作階段擁有它。如果擁有工作階段由於嚴重損毀、網路問題或類似事件意外結束，則可能無法清除暫時資料表，將其保留為「孤立」資料表。當 PostgreSQL 自動清空程序偵測到孤立的暫時資料表時，它會記錄下列訊息：

```
LOG: autovacuum: found orphan temp table \"%s\".\"%s\" in database \"%s\"
```

postgres_get_av_diag() 函數將暫存資料表識別為封鎖程式時，會顯示類似下列的輸出。為了讓函數正確顯示與暫時資料表相關的輸出，它需要在存在這些資料表的相同資料庫中執行。

```
blocker               | Temporary table
database              | my_database
blocker_identifier    | pg_temp_14.ttemp
wait_event            | Not applicable
autovacuum_lagging_by | 1,805,802,632
suggestion            | Connect to database "my_database". Review carefully, you
 may consider dropping temporary table using command in suggested_action. For more
 information, see Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"DROP TABLE ttemp;"}
```

建議動作

遵循輸出 suggestion 欄中提供的指示，以識別和移除防止自動清空執行的暫時資料表。使用以下命令捨棄 postgres_get_av_diag() 報告的暫時資料表。根據 postgres_get_av_diag() 函數提供的輸出取代資料表名稱。

```
DROP TABLE my_temp_schema.my_temp_table;
```

下列查詢可用來識別暫時資料表：

```
SELECT
    oid,
    relname,
    relnamespace::regnamespace,
    age(relfrozenxid)
FROM
    pg_class
WHERE
relpersistence = 't'
```

```
ORDER BY
    age(relfrozenxid) DESC;
```

解決 Aurora PostgreSQL 中無法識別的清空封鎖程式

本節會探索其他可能阻止清空進行的原因。postgres_get_av_diag() 函數目前無法直接識別這些問題。

主題

- 索引不一致
- 異常高的交易速率

索引不一致

邏輯上不一致的索引會防止自動清空進行。在索引的清空階段或 SQL 陳述式存取索引時，系統會記錄下列錯誤或類似錯誤。

```
ERROR: right sibling's left-link doesn't match:block 5 links to 10 instead of expected
 2 in index ix_name
```

```
ERROR: failed to re-find parent key in index "XXXXXXXXXX" for deletion target page XXX
CONTEXT:  while vacuuming index index_name of relation schema.table
```

指引

在手動 VACUUM FREEZE 上使用 INDEX_CLEANUP 重建索引或略過索引。

- 使用 CONCURRENTLY 選項 – 在 PostgreSQL 第 12 版之前，重建索引需要獨佔資料表鎖定，以限制對資料表的存取。透過 PostgreSQL 第 12 版和更新版本，CONCURRENTLY 選項允許資料列層級鎖定，以大幅改善資料表的可用性。以下是命令：

  ```
  REINDEX INDEX ix_name CONCURRENTLY;
  ```

  雖然 CONCURRENTLY 干擾性較低，但在忙碌的資料表上可能會變慢。如果可能，請考慮在低流量期間建立索引。如需詳細資訊，請參閱 PostgreSQL 文件中的 REINDEX。
- 使用 INDEX_CLEANUP FALSE 選項 – 如果索引較大且估計需要大量時間才能完成，您可以在排除索引的同時執行手動 VACUUM FREEZE 來解除封鎖自動清空。此功能可在 PostgreSQL 第 12 版及更新版本中使用。

略過索引可讓您略過不一致索引的清空程序，並減輕包圍問題。不過，這無法解決基本的無效頁面問題。若要完全處理和解決無效的頁面問題，您仍然需要重建索引。

異常高的交易速率

在 PostgreSQL 中，高交易速率可能會大幅影響自動清空的效能，導致清除無效元組的速度變慢，並提高交易 ID 包圍的風險。您可以透過測量兩個時段之間的 `max(age(datfrozenxid))` 差異來監控交易速率，通常是每秒。此外，您可以使用 RDS Performance Insights 的下列計數器指標來測量交易速率 (xact_commit 和 xact_rollback 的總和)，這是交易的總數。

計數器	類型	單位	指標
xact_commit	交易	每秒遞交數	db.Transactions.xact_commit
xact_rollback	交易	每秒轉返數	db.Transactions.xact_rollback

快速增加表示交易負載很高，可能會自動清空不堪負荷，導致膨脹、鎖定爭用和潛在的效能問題。這可能會在幾個方面對自動清空程序產生負面影響：

- 資料表活動：正在清空的特定資料表可能遇到大量交易，因而導致延遲。
- 系統資源整體系統可能會超載，導致自動清空難以存取必要的資源以有效率地運作。

請考慮下列策略，以允許自動清空更有效地運作並跟上其任務：

1. 如果可能，請降低交易速率。考慮在可行的情況下批次處理或分組類似的交易。
2. 以經常更新的資料表為目標，在離峰時段內每夜、每週或每兩週手動 `VACUUM FREEZE` 操作一次。
3. 考慮擴展執行個體類別以配置更多系統資源，以處理高交易量和自動清空。

解決 Aurora PostgreSQL 中的清空效能問題

本節討論通常會導致降低清空效能的因素，以及如何解決這些問題。

主題

- 清空大型索引

- 要清空的資料表或資料庫過多
- 積極清空 (以防止包圍) 正在執行

清空大型索引

VACUUM 會執行循序階段：初始化、堆積掃描、索引和堆積清空、索引清除、堆積截斷和最終清除。在堆積掃描期間，程序會刪除頁面、重組並凍結頁面。完成堆積掃描後，VACUUM 會清除索引、將空白頁面傳回至作業系統，並執行最終清除任務，例如清空可用空間貼圖和更新統計資料。

當 maintenance_work_mem (或 autovacuum_work_mem) 不足以處理索引時，索引清空可能需要多次通過。在 PostgreSQL 16 和更早版本中，用於儲存無效元組的 ID 1 GB 記憶體限制通常會強制在大型索引上多次通過。PostgreSQL 17 推出 TidStore，可動態配置記憶體，而不是使用單一配置陣列。這可消除 1 GB 限制、更有效率地使用記憶體，並減少每個索引進行多次索引掃描的需求。

如果可用的記憶體無法同時容納整個索引處理，大型索引仍可能需要 PostgreSQL 17 中的多次通過。一般而言，較大的索引包含更多需要多次通過的無效元組。

偵測緩慢清空操作

postgres_get_av_diag() 函數可以偵測清空操作何時因為記憶體不足而緩慢執行。如需該功能的詳細資訊，請參閱在 Aurora PostgreSQL 中安裝自動清空監控和診斷工具。

當可用的記憶體不足以在單次通過中完成索引清空時， postgres_get_av_diag() 函數會發出下列通知。

rds_tools 1.9

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound and
 it might be slow.
```

```
NOTICE: The current setting of autovacuum_work_mem is XX might not be sufficient.
 Consider increasing the setting to XXX, and if necessary, scaling up the RDS instance
 class for more
        memory. The suggested value is an estimate based on the current number of dead
 tuples for the table being vacuumed, which might not fully reflect the latest state.
 Additionally, review the possibility of manual
        vacuum with exclusion of indexes using (VACUUM (INDEX_CLEANUP FALSE, VERBOSE
 TRUE) table_name;). For more information, see

        Working with PostgreSQL autovacuum in the Amazon Aurora User Guide.
```

> **Note**
>
> postgres_get_av_diag() 函數依賴 pg_stat_all_tables.n_dead_tup 來估計索引清空所需的記憶體量。

當 postgres_get_av_diag() 函數識別由於 autovacuum_work_mem 不足而需要多個索引掃描的緩慢清空操作時，會產生下列訊息：

```
NOTICE: Your vacuum is performing multiple index scans due to insufficient
 autovacuum_work_mem:XXX for index vacuuming.
       For more information, see Working with PostgreSQL autovacuum in the Amazon
 Amazon RDS User Guide.
```

指引

您可以使用手動 VACUUM FREEZE 來套用下列解決方法，以加速凍結資料表。

增加清空的記憶體

如 postgres_get_av_diag() 函數所建議，建議增加 autovacuum_work_mem 參數以解決執行個體層級的潛在記憶體限制。雖然 autovacuum_work_mem 是動態參數，但請務必注意，若要讓新的記憶體設定生效，自動清空精靈需要重新啟動其工作者。若要完成此操作：

1. 確認新的設定已就緒。
2. 終止目前正在執行自動清空的程序。

此方法可確保調整後的記憶體配置套用至新的自動清空操作。

如需更立即的結果，請考慮在工作階段中手動執行增加 maintenance_work_mem 設定的 VACUUM FREEZE 操作：

```
SET maintenance_work_mem TO '1GB';
VACUUM FREEZE VERBOSE table_name;
```

如果您使用 Amazon RDS 並發現需要額外的記憶體以支援 maintenance_work_mem 或 autovacuum_work_mem 的更高值，請考慮升級至具有更多記憶體的執行個體類別。這可提供必要的資源來增強手動和自動清空操作，進而改善整體清空和資料庫效能。

停用 INDEX_CLEANUP

PostgreSQL 第 12 版和更新版本中的手動 VACUUM 允許略過索引清除階段，而 PostgreSQL 第 14 版和更新版本中的緊急自動清空會根據 vacuum_failsafe_age 參數自動執行此操作。

> **Warning**
>
> 略過索引清除可能會導致索引膨脹，並對查詢效能產生負面影響。若要緩解這種情況，請考慮在維護時段重新編製索引或清空受影響的索引。

如需處理大型索引的其他指引，請參閱 使用大型索引管理自動清空 上的文件。

平行索引清空

從 PostgreSQL 13 開始，根據預設可以使用手動 VACUUM 平行清空和清理索引，並為每個索引指派一個清空工作者程序。不過，對於 PostgreSQL 判斷清空操作是否符合平行執行的資格，必須符合特定條件：

- 必須至少有兩個索引。
- 應將 max_parallel_maintenance_workers 參數設定為至少 2。
- 索引大小必須超過 min_parallel_index_scan_size 限制，預設為 512KB。

您可以根據 Amazon RDS 執行個體上可用的 vCPU 數目和資料表上的索引數目來調整 max_parallel_maintenance_workers 設定，以最佳化清空周轉時間。

如需詳細資訊，請參閱 Amazon RDS for PostgreSQL 和 Amazon Aurora PostgreSQL 中的平行清空。

要清空的資料表或資料庫過多

如 PostgreSQL 的自動清空精靈文件所述，自動清空精靈會透過多個程序運作。這包括持久性自動清空啟動器，負責為系統中的每個資料庫啟動自動清空工作者程序。啟動器會排程這些工作者，每個資料庫大約每 autovacuum_naptime 秒啟動一次。

使用「N」個資料庫時，新工作者大約每 [autovacuum_naptime/N 秒] 開始一次。不過，並行工作者的總數會受到 autovacuum_max_workers 設定的限制。如果需要清空的資料庫或資料表數量超過此限制，則在工作者一變成可用時，就會立即處理下一個資料庫或資料表。

當許多大型資料表或資料庫需要同時清空時，所有可用的自動清空工作者可能會長時間佔用，進而延遲其他資料表和資料庫的維護。在交易速率較高的環境中，此瓶頸可能會快速提升，並可能導致 Amazon RDS 執行個體中的包圍清空問題。

當 postgres_get_av_diag() 偵測到大量資料表或資料庫時，會提供下列建議：

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound and
 it might be slow.
```

```
NOTICE: The current setting of autovacuum_max_workers:3 might not be sufficient.
 Consider increasing the setting and, if necessary, consider scaling up the Amazon RDS
 instance class for more workers.
```

指引

增加 autovacuum_max_workers

為了加速清空，建議您調整 autovacuum_max_workers 參數，以允許更多並行自動清空工作者。如果效能瓶頸持續存在，請考慮將 Amazon RDS 執行個體擴展到具有更多 vCPU 的類別，這可以進一步改善平行處理功能。

積極清空 (以防止包圍) 正在執行

PostgreSQL 中資料庫 (MaximumUsedTransactionIDs) 的存留期只會在積極清空 (防止包圍) 成功完成時減少。在此清空完成之前，存留期會根據交易速率繼續增加。

當 postgres_get_av_diag() 函數偵測到積極的清空時，會產生下列 NOTICE。不過，它只會在清空作用中至少兩分鐘後觸發此輸出。

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound,
 monitor autovacuum performance.
```

如需積極清空的詳細資訊，請參閱積極清空已執行時。

您可以使用下列查詢來確認積極清空是否正在進行中：

```
SELECT
    a.xact_start AS start_time,
    v.datname "database",
    a.query,
    a.wait_event,
    v.pid,
    v.phase,
    v.relid::regclass,
    pg_size_pretty(pg_relation_size(v.relid)) AS heap_size,
```

```
    (
        SELECT
            string_agg(pg_size_pretty(pg_relation_size(i.indexrelid)) || ':' ||
 i.indexrelid::regclass || chr(10), ', ')
        FROM
            pg_index i
        WHERE
            i.indrelid = v.relid
    ) AS index_sizes,
    trunc(v.heap_blks_scanned * 100 / NULLIF(v.heap_blks_total, 0)) AS step1_scan_pct,
    v.index_vacuum_count || '/' || (
        SELECT
            count(*)
        FROM
            pg_index i
        WHERE
            i.indrelid = v.relid
    ) AS step2_vacuum_indexes,
    trunc(v.heap_blks_vacuumed * 100 / NULLIF(v.heap_blks_total, 0)) AS
 step3_vacuum_pct,
    age(CURRENT_TIMESTAMP, a.xact_start) AS total_time_spent_sofar
FROM
    pg_stat_activity a
    INNER JOIN pg_stat_progress_vacuum v ON v.pid = a.pid;
```

您可以透過檢查輸出中的查詢欄來判斷是否為積極的清空 (以防止包圍)。片語「防止包圍」表示它是積極的清空。

```
query                 | autovacuum: VACUUM public.t3 (to prevent wraparound)
```

例如，假設您有交易存留期為 10 億的封鎖程式和需要積極清空的資料表，以防止在相同的交易存留期進行包圍。此外，有另一個交易存留期為 7.5 億的封鎖程式。在交易存留期 10 億時清除封鎖程式後，交易存留期不會立即降至 7.5 億。它將保持很高，直到需要積極清空的資料表或任何存留期超過 7.5 億的交易完成為止。在此期間，PostgreSQL 叢集的交易存留期將繼續增加。清空程序完成後，交易存留期將降至 7.5 億，但會再次開始增加，直到進一步清空完成為止。只要這些條件持續存在，此週期就會繼續，直到交易存留期最終降至由 `autovacuum_freeze_max_age` 所指定為 Amazon RDS 執行個體設定的層級為止。

Aurora PostgreSQL 中的通知訊息說明

`postgres_get_av_diag()` 函數提供下列通知訊息：

當存留期尚未達到監控閾值時

用於 postgres_get_av_diag() 識別封鎖程式的監控閾值預設為 5 億筆交易。如果 postgres_get_av_diag()產生下列通知，則表示交易存留期尚未達到此閾值。

```
NOTICE: postgres_get_av_diag() checks for blockers that prevent aggressive vacuums
 only, it does so only after exceeding dvb_threshold which is 500,000,000 and age of
 this PostgreSQL cluster is currently at 2.
```

未連線至具有最舊交易 ID 存留期的資料庫

postgres_get_av_diag() 函數會在連接至具有最舊交易 ID 存留期的資料庫時，提供最準確的輸出。postgres_get_av_diag() 所報告之交易 ID 存留期最早的資料庫，將與您案例中的「my_database」不同。如果您未連線到正確的資料庫，會產生下列通知：

```
NOTICE: You are not connected to the database with the age of oldest transaction
 ID. Connect to my_database database and run postgres_get_av_diag() for accurate
 reporting.
```

連線具有最舊交易存留期的資料庫非常重要，原因如下：

- 識別暫時資料表封鎖程式：由於暫時資料表的中繼資料專屬於每個資料庫，因此它們通常會在建立它們的資料庫中找到。不過，如果暫存資料表剛好是最熱門的封鎖程式，並且位於具有最舊交易的資料庫中，則可能會誤導。連線到正確的資料庫可確保正確識別暫時資料表封鎖程式。
- 診斷緩慢清空：索引中繼資料和資料表計數資訊是資料庫特定的，且為診斷緩慢清空問題的必要項目。

具有依存留期最舊交易的資料庫位於 rdsadmin 或 template0 資料庫

在某些情況下，rdsadmin或 template0 資料庫可能會識別為具有最舊交易 ID 存留期的資料庫。如果發生這種情況，postgres_get_av_diag() 會發出下列通知：

```
NOTICE: The database with the age of oldest transaction ID is rdsadmin or template0,
 reach out to support if the reported blocker is in rdsadmin or template0.
```

確認列出的封鎖程式不是來自這兩個資料庫。如果報告封鎖程式出現在 rdsadmin或 template0 中，請聯絡支援，因為這些資料庫無法供使用者存取且需要介入。

rdsadmin 或 template0 資料庫不太可能包含最熱門的封鎖程式。

當積極清空已在執行時

postgres_get_av_diag() 函數旨在報告積極清空程序何時執行，但只會在清空作用中至少 1 分鐘後觸發此輸出。此刻意延遲有助於降低誤報的機會。透過等待，函數可確保僅報告有效且重要的清空，進而更準確且可靠地監控清空活動。

當 postgres_get_av_diag() 函數偵測到一或多個進行中的積極清空時，會產生下列通知。

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound,
 monitor autovacuum performance.
```

如通知所示，繼續監控清空的效能。如需積極清空的詳細資訊，請參閱 積極清空 (以防止包圍) 正在執行

當自動清空關閉時

如果資料庫執行個體上已停用自動清空，postgres_get_av_diag() 函數會產生下列通知：

```
NOTICE: Autovacuum is OFF, we strongly recommend to enable it, no restart is
 necessary.
```

自動清空是 Aurora PostgreSQL 資料庫執行個體的重要功能，可確保順暢的資料庫操作。它會自動移除舊的資料列版本、回收儲存空間，並防止資料表膨脹，有助於保持資料表和索引的效率，以獲得最佳效能。此外，它可以防止交易 ID 包圍，這會停止 Amazon RDS 執行個體上的交易。停用自動清空可能會導致資料庫效能和穩定性的長期下降。我們建議您隨時將其保留。如需詳細資訊，請參閱了解 Aurora PostgreSQL 環境中的自動清空。

> **Note**
>
> 關閉自動清空不會停止積極的清空。這些仍會在您的資料表達到 autovacuum_freeze_max_age 閾值時發生。

剩餘的交易數量非常低

當包圍清空即將關閉時，postgres_get_av_diag() 函數會產生下列通知。當您的 Amazon RDS 執行個體有 1 億筆交易且不可能拒絕新交易時，就會發出此通知。

```
WARNING: Number of transactions remaining is critically low, resolve issues with
 autovacuum or perform manual VACUUM FREEZE before your instance stops accepting
 transactions.
```

您需要立即採取動作，以避免資料庫停機時間。您應該密切監控您的清空操作，並考慮在受影響的資料庫上手動啟動 `VACUUM FREEZE`，以防止交易失敗。

在 Amazon Aurora PostgreSQL 中管理 TOAST OID 爭用

TOAST（超大型屬性儲存技術）是一種 PostgreSQL 功能，旨在處理超過一般 8KB 資料庫區塊大小的大型資料值。PostgreSQL 不允許實體資料列跨越多個區塊。區塊大小可做為資料列大小的上限。TOAST 透過將大型欄位值分割為較小的區塊來克服此限制。它將它們分別存放在連結到主資料表的專用 TOAST 資料表中。如需詳細資訊，請參閱 PostgreSQL TOAST 儲存機制和實作文件。

主題

- 了解 TOAST 操作
- 識別效能挑戰
- 建議
- 監控

了解 TOAST 操作

TOAST 會執行壓縮，並離線存放大型欄位值。TOAST 會將唯一的 OID（物件識別符）指派給儲存在 TOAST 資料表中的每個超大型資料區塊。主資料表會將 TOAST 值 ID 和關聯 ID 存放在頁面上，以參考 TOAST 資料表中對應的資料列。這可讓 PostgreSQL 有效率地尋找和管理這些 TOAST 區塊。不過，隨著 TOAST 資料表的成長，系統可能會耗盡可用的 OIDs，導致效能降低和因 OID 耗盡而可能停機。

TOAST 中的物件識別符

物件識別符 (OID) 是 PostgreSQL 用於參考資料庫物件的全系統唯一識別符，例如資料表、索引和函數。這些識別符在 PostgreSQL 的內部操作中扮演重要角色，可讓資料庫有效率地尋找和管理物件。

對於具有合格資料集的資料表，PostgreSQL 會指派 OIDs，以唯一識別存放在相關聯 TOAST 資料表中的每個超大型資料區塊。系統會將每個區塊與 建立關聯`chunk_id`，這有助於 PostgreSQL 在 TOAST 資料表中有效率地組織和尋找這些區塊。

識別效能挑戰

PostgreSQL 的 OID 管理倚賴全域 32 位元計數器，以便在產生 40 億個唯一值之後進行包裝。當資料庫叢集共用此計數器時，OID 配置會在 TOAST 操作期間涉及兩個步驟：

- 用於配置的全域計數器 – 全域計數器會跨叢集指派新的 OID。
- 本機搜尋衝突 – TOAST 資料表可確保新的 OID 不會與該特定資料表中已使用的現有 OIDs 衝突。

發生下列情況時，可能會發生效能降低：

- TOAST 資料表具有高分段或密集的 OID 用量，導致指派 OID 時發生延遲。
- 系統經常在具有大量使用 TOAST 的高資料流失或寬資料表的環境中配置和重複使用 OIDs。

如需詳細資訊，請參閱 PostgreSQL TOAST 資料表大小限制和 OID 配置文件：

全域計數器會產生 OIDs 並包圍每 40 億個值，因此系統偶爾會再次產生已使用的值。PostgreSQL 偵測到 ，並使用下一個 OID 再次嘗試。如果使用過的 OID 值執行很長，且在 TOAST 資料表中沒有間隙，則可能會發生緩慢的 INSERT。隨著 OID 空間填滿，這些挑戰變得更加明顯，導致插入和更新速度變慢。

識別問題

- 簡單INSERT陳述式需要比平常更長的時間，方式不一致且隨機。
- 只有涉及 TOAST 操作的 INSERT和 UPDATE陳述式才會發生延遲。
- 當系統難以在 TOAST 資料表中尋找可用的 OIDs時，以下日誌項目會出現在 PostgreSQL 日誌中：

```
LOG: still searching for an unused OID in relation "pg_toast_20815"
DETAIL: OID candidates have been checked 1000000 times, but no unused OID has been
 found yet.
```

- 績效詳情指出與 LWLock:buffer_io和LWLock:OidGenLock等待事件相關聯的大量平均作用中工作階段 (AAS)。

  您可以執行下列 SQL 查詢，以識別具有等待事件的長期執行 INSERT 交易：

```
SELECT
    datname AS database_name,
    usename AS database_user,
    pid,
    now() - pg_stat_activity.xact_start AS transaction_duration,
    concat(wait_event_type, ':', wait_event) AS wait_event,
    substr(query, 1, 30) AS TRANSACTION,
    state
FROM
```

```
    pg_stat_activity
WHERE (now() - pg_stat_activity.xact_start) > INTERVAL '60 seconds'
    AND state IN ('active', 'idle in transaction', 'idle in transaction (aborted)',
 'fastpath function call', 'disabled')
    AND pid <> pg_backend_pid()
AND lower(query) LIKE '%insert%'
ORDER BY
    transaction_duration DESC;
```

顯示具有延長等待時間的 INSERT 操作的查詢結果範例：

```
 database_name |  database_user  |  pid  | transaction_duration |     wait_event
 |          transaction           | state
---------------+-----------------+-------+----------------------
+---------------------+---------------------------------+--------
 postgres       | db_admin_user| 70965 | 00:10:19.484061      | LWLock:buffer_io    |
 INSERT INTO "products" (......... | active
 postgres       | db_admin_user| 69878 | 00:06:14.976037      | LWLock:buffer_io    |
 INSERT INTO "products" (......... | active
 postgres       | db_admin_user| 68937 | 00:05:13.942847      | :                   |
 INSERT INTO "products" (......... | active
```

隔離問題

- 測試小型插入 – 插入小於`toast_tuple_target`閾值的記錄。請記住，壓縮會在 TOAST 儲存體之前套用。如果此操作沒有效能問題，則問題與 TOAST 操作有關。
- 測試新資料表 – 建立具有相同結構的新資料表，並插入大於 的記錄`toast_tuple_target`。如果這樣做沒有問題，問題會本地化為原始資料表的 OID 配置。

建議

下列方法有助於解決 TOAST OID 爭用問題。

- 資料清理和封存 – 檢閱和刪除任何過時或不必要的資料，以釋出 OIDs 以供日後使用，或封存資料。考量下列限制：
  - 可擴展性有限，因為未來清理可能並不總是可行。
  - 可能長時間執行的 VACUUM 操作，以移除產生的無效元組。

- 寫入新資料表 – 為未來的插入建立新的資料表，並使用UNION ALL檢視來合併查詢的舊資料和新資料。此檢視會顯示來自舊資料表和新資料表的合併資料，允許查詢以單一資料表的形式存取它們。考量下列限制：
  - 舊資料表的更新仍可能導致 OID 耗盡。
- 分割區或碎片 – 分割資料表或碎片資料，以獲得更佳的可擴展性和效能。考量下列限制：
  - 查詢邏輯和維護的複雜性更高，可能需要應用程式變更才能正確處理分割的資料。

監控

使用系統資料表

您可以使用 PostgreSQL 的系統資料表來監控 OID 用量的成長。

> **Warning**
>
> 根據 TOAST 資料表中的 OIDs 數量，可能需要一些時間才能完成。建議您在非上班時間排定監控，將影響降至最低。

下列匿名區塊會計算每個 TOAST 資料表中使用的不同 OIDs 數目，並顯示父資料表資訊：

```
DO $$
DECLARE
    r record;
    o bigint;
    parent_table text;
    parent_schema text;
BEGIN
    SET LOCAL client_min_messages TO notice;
    FOR r IN
    SELECT
        c.oid,
        c.oid::regclass AS toast_table
    FROM
        pg_class c
    WHERE
        c.relkind = 't'
        AND c.relowner != 10 LOOP
            -- Fetch the number of distinct used OIDs (chunk IDs) from the TOAST table
```

```
            EXECUTE 'SELECT COUNT(DISTINCT chunk_id) FROM ' || r.toast_table INTO o;
            -- If there are used OIDs, find the associated parent table and its schema
            IF o <> 0 THEN
                SELECT
                    n.nspname,
                    c.relname INTO parent_schema,
                    parent_table
                FROM
                    pg_class c
                    JOIN pg_namespace n ON c.relnamespace = n.oid
                WHERE
                    c.reltoastrelid = r.oid;
                -- Raise a concise NOTICE message
                RAISE NOTICE 'Parent schema: % | Parent table: % | Toast table: %
 | Number of used OIDs: %', parent_schema, parent_table, r.toast_table, TO_CHAR(o,
 'FM9,999,999,999,999');
            END IF;
        END LOOP;
END
$$;
```

依 TOAST 資料表顯示 OID 用量統計資料的範例輸出：

```
NOTICE:  Parent schema: public | Parent table: my_table | Toast table:
 pg_toast.pg_toast_16559 | Number of used OIDs: 45,623,317
NOTICE:  Parent schema: public | Parent table: my_table1 | Toast table:
 pg_toast.pg_toast_45639925 | Number of used OIDs: 10,000
NOTICE:  Parent schema: public | Parent table: my_table2 | Toast table:
 pg_toast.pg_toast_45649931 | Number of used OIDs: 1,000,000
DO
```

下列匿名區塊會擷取每個非空白 TOAST 資料表的指派 OID 上限：

```
DO $$
DECLARE
    r record;
    o bigint;
    parent_table text;
    parent_schema text;
BEGIN
    SET LOCAL client_min_messages TO notice;
    FOR r IN
    SELECT
```

```
        c.oid,
        c.oid::regclass AS toast_table
    FROM
        pg_class c
    WHERE
        c.relkind = 't'
        AND c.relowner != 10 LOOP
            -- Fetch the max(chunk_id) from the TOAST table
            EXECUTE 'SELECT max(chunk_id) FROM ' || r.toast_table INTO o;
            -- If there's at least one TOASTed chunk, find the associated parent table
 and its schema
            IF o IS NOT NULL THEN
                SELECT
                    n.nspname,
                    c.relname INTO parent_schema,
                    parent_table
                FROM
                    pg_class c
                    JOIN pg_namespace n ON c.relnamespace = n.oid
                WHERE
                    c.reltoastrelid = r.oid;
                -- Raise a concise NOTICE message
                RAISE NOTICE 'Parent schema: % | Parent table: % | Toast table:
 % | Max chunk_id: %', parent_schema, parent_table, r.toast_table, TO_CHAR(o,
 'FM9,999,999,999,999');
            END IF;
        END LOOP;
END
$$;
```

顯示 TOAST 資料表最大區塊 IDs的範例輸出：

```
NOTICE:  Parent schema: public | Parent table: my_table | Toast table:
 pg_toast.pg_toast_16559 | Max chunk_id: 45,639,907
NOTICE:  Parent schema: public | Parent table: my_table1 | Toast table:
 pg_toast.pg_toast_45639925 | Max chunk_id: 45,649,929
NOTICE:  Parent schema: public | Parent table: my_table2 | Toast table:
 pg_toast.pg_toast_45649931 | Max chunk_id: 46,649,935
DO
```

使用 Performance Insights

在需要指派新物件識別符 (OIDs) 的操作期間，等待事件`LWLock:buffer_io`和`LWLock:OidGenLock`會出現在績效詳情中。這些事件的高平均作用中工作階段 (AAS) 通常會在 OID 指派和資源管理期間指向爭用。這在具有高資料流失、大量大數據用量或頻繁建立物件的環境中特別常見。

LWLock：buffer_io

`LWLock:buffer_io` 是當 PostgreSQL 工作階段正在等待共用緩衝區上的 I/O 操作完成時發生的等待事件。當資料庫將資料從磁碟讀取到記憶體，或將修改過的頁面從記憶體寫入磁碟時，通常會發生這種情況。`BufferIO` 等待事件會防止多個程序在 I/O 操作進行時存取或修改相同的緩衝區，以確保一致性。此等待事件的頻繁出現可能表示資料庫工作負載中的磁碟瓶頸或過多的 I/O 活動。

在 TOAST 操作期間：

- PostgreSQL 會為大型物件配置 OIDs，並透過掃描 TOAST 資料表的索引來確保其唯一性。
- 大型 TOAST 索引可能需要存取多個頁面來驗證 OID 唯一性。這會導致磁碟 I/O 增加，特別是當緩衝集區無法快取所有必要的頁面時。

索引的大小會直接影響在這些操作期間需要存取的緩衝區頁面數目。即使索引未膨脹，其透薄大小也會增加緩衝區 I/O，特別是在高並行或高流失環境中。如需詳細資訊，請參閱 LWLock：BufferIO 等待事件疑難排解指南。

LWLock:OidGenLock

`OidGenLock` 是當 PostgreSQL 工作階段正在等待配置新的物件識別符 (OID) 時發生的等待事件。此鎖定可確保循序且安全地產生 OIDs，一次只允許一個程序產生 OIDs。

在 TOAST 操作期間：

- TOAST 資料表中區塊的 OID 配置 – PostgreSQL 會將 OIDs 指派給 TOAST 資料表中的區塊。每個 OID 都必須是唯一的，以防止系統目錄中的衝突。
- 高並行 – 由於對 OID 產生器的存取是循序的，因此當多個工作階段同時建立需要 OIDs物件時，`OidGenLock`可能會發生 的爭用。這會增加工作階段等待 OID 配置完成的可能性。
- 相依於系統目錄存取 – 配置 OIDs 需要更新共用系統目錄資料表，例如 `pg_class`和 `pg_type`。如果這些資料表遇到繁重的活動（由於頻繁的 DDL 操作），可能會增加 的鎖定爭用`OidGenLock`。
- 高 OID 分配需求 – 具有大型資料記錄的 TOAST 繁重工作負載需要持續的 OID 分配，從而增加爭用。

增加 OID 爭用的其他因素：

- 頻繁建立物件 – 經常建立和捨棄物件的工作負載，例如暫存資料表，在全域 OID 計數器上擴增爭用。
- 全域計數器鎖定 – 全域 OID 計數器會循序存取，以確保唯一性，在高並行環境中建立單一爭用點。

使用 Babelfish for Aurora PostgreSQL

Babelfish for Aurora PostgreSQL 會擴展您的 Aurora PostgreSQL 資料庫叢集，使其能夠接受來自 SQL Server 用戶端的資料庫連線。透過 Babelfish，原本為 SQL Server 建置的應用程式可以直接使用 Aurora PostgreSQL，相較於傳統遷移，程式碼只需稍微變更，不需要變更資料庫驅動程式。如需遷移的詳細資訊，請參閱將 SQL Server 資料庫遷移至 Babelfish for Aurora PostgreSQL。

Babelfish 為 Aurora PostgreSQL 資料庫叢集提供額外的端點，使之瞭解 SQL Server 線路層級通訊協定和常用的 SQL Server 陳述式。使用表格式資料串流 (TDS) 線路通訊協定的用戶端應用程式能夠以原生方式連線至 Aurora PostgreSQL 上的 TDS 接聽程式連接埠。若要進一步了解 TDS，請參閱 Microsoft 網站上的 [MS-TDS]：表格式資料串流通訊協定。

> **Note**
>
> Babelfish for Aurora PostgreSQL 支援 TDS 第 7.1 版到第 7.4 版。

Babelfish 也會使用 PostgreSQL 連線來存取資料。依預設，Babelfish 支援的兩種 SQL 方言都可以透過其在下列連接埠的原生接線通訊協定取得：

- SQL Server 方言 (T-SQL)，用戶端需要連線至連接埠 1433。
- PostgreSQL 方言 (PL/pgSQL)，用戶端會連線至連接埠 5432。

Babelfish 會執行 Transact-SQL (T-SQL) 語言，但有一些不同。如需更多詳細資訊，請參閱 Babelfish for Aurora PostgreSQL 與 SQL Server 之間的差異。

在下列各節，您可以找到設定與使用 Babelfish for Aurora PostgreSQL 資料庫執行個體的相關資訊。

主題

- Babelfish 限制
- 了解 Babelfish 架構和組態
- 建立 Babelfish for Aurora PostgreSQL DB 叢集
- 將 SQL Server 資料庫遷移至 Babelfish for Aurora PostgreSQL
- 使用 Babelfish for Aurora PostgreSQL 進行資料庫身分驗證
- 連線至 Babelfish 資料庫叢集
- 使用 Babelfish

- Babelfish 疑難排解
- 停用 Babelfish
- 管理 Babelfish for Aurora PostgreSQL 版本更新
- Babelfish for Aurora PostgreSQL 參考

Babelfish 限制

下列限制目前適用於 Babelfish for Aurora PostgreSQL：

- Babelfish 不支援下列 Aurora 功能：
  - AWS Identity and Access Management
  - 資料庫活動串流 (DAS)
  - RDS 資料 API 搭配 Aurora PostgreSQL Serverless 第 2 版與佈建
  - 搭配 RDS for SQL Server 的 RDS Proxy
  - Salted 挑戰回應身分驗證機制 (SCRAM)
  - 查詢編輯器
  - 零 ETL 整合
- Babelfish 不提供下列用戶端驅動程式 API 支援：
  - 不支援連線屬性與 Microsoft Distributed Transaction Coordinator (MSDTC) 相關的 API 要求。這些包括 SQL 伺服器 JDBC 驅動程式中 SQLServerXAResource 類別的 XA 呼叫。
- Babelfish 目前不支援下列 Aurora PostgreSQL 延伸模組：
  - `bloom`
  - `btree_gin`
  - `btree_gist`
  - `citext`
  - `cube`
  - `hstore`
  - `hypopg`
  - 使用 `pglogical` 進行邏輯複寫
  - `ltree`
  - `pgcrypto`
  - 使用 `apg_plan_mgmt` 的查詢計劃管理

  若要進一步了解 PostgreSQL 延伸模組，請參閱使用擴充功能和外部資料包裝函式。
- 不支援設計為 Microsoft JDBC 驅動程式之替代方案的開放原始碼 JTDS 驅動程式。

了解 Babelfish 架構和組態

您可以如同管理任何 Aurora 資料庫叢集一樣地管理執行 Babelfish 的 Aurora PostgreSQL 相容版本資料庫叢集。也就是說，您可享有 Aurora 資料庫叢集提供的可擴展性、因支援容錯移轉而有的高可用性，及內建複本所帶來的益處。若要進一步了解這些功能，請參閱 管理 Aurora 資料庫叢集的效能和擴展、Amazon Aurora 的高可用性 及 以 Amazon Aurora 進行複寫。您還可以存取許多其他 AWS 工具和公用程式，包括以下項目：

- Amazon CloudWatch是一項監控和可觀察性服務，為您提供資料和可行的洞察。如需更多詳細資訊，請參閱 使用 Amazon CloudWatch 監控 Amazon Aurora 指標。
- 績效詳情是一項資料庫效能調校和監視功能，可協助您快速評估資料庫的負載。如需詳細資訊，請參閱 在 Amazon Aurora 上使用績效詳情監控資料庫負載。
- Aurora 全域資料庫橫跨多個 AWS 區域，可實現低延遲全域讀取，並從可能影響整個 AWS 區域 的罕見中斷提供快速復原。如需更多詳細資訊，請參閱 使用 Amazon Aurora 全球資料庫。
- 自動軟體修補使用最新發佈的安全性和功能修補程式，隨時更新資料庫。
- Amazon RDS 事件透過電子郵件或簡訊，通知您重要的資料庫事件，例如自動容錯移轉。如需更多詳細資訊，請參閱 監控 Amazon Aurora 事件。

在下文中，您可以了解 Babelfish 架構，以及 Babelfish 如何處理您遷移的 SQL Server 資料庫。建立 Babelfish 資料庫叢集時，您需要預先針對單資料庫或多資料庫、定序和其他詳細資料做出一些決策。

主題

- Babelfish 架構
- Babelfish 的資料庫叢集參數群組設定
- 了解 Babelfish for Aurora PostgreSQL 中的定序
- 使用逃生艙管理 Babelfish 錯誤處理

Babelfish 架構

建立啟用 Babelfish 的 Aurora PostgreSQL 叢集時，Aurora 會在叢集上佈建名為 `babelfish_db` 的 PostgreSQL 資料庫。所有遷移的 SQL Server 物件和結構都位於此資料庫。

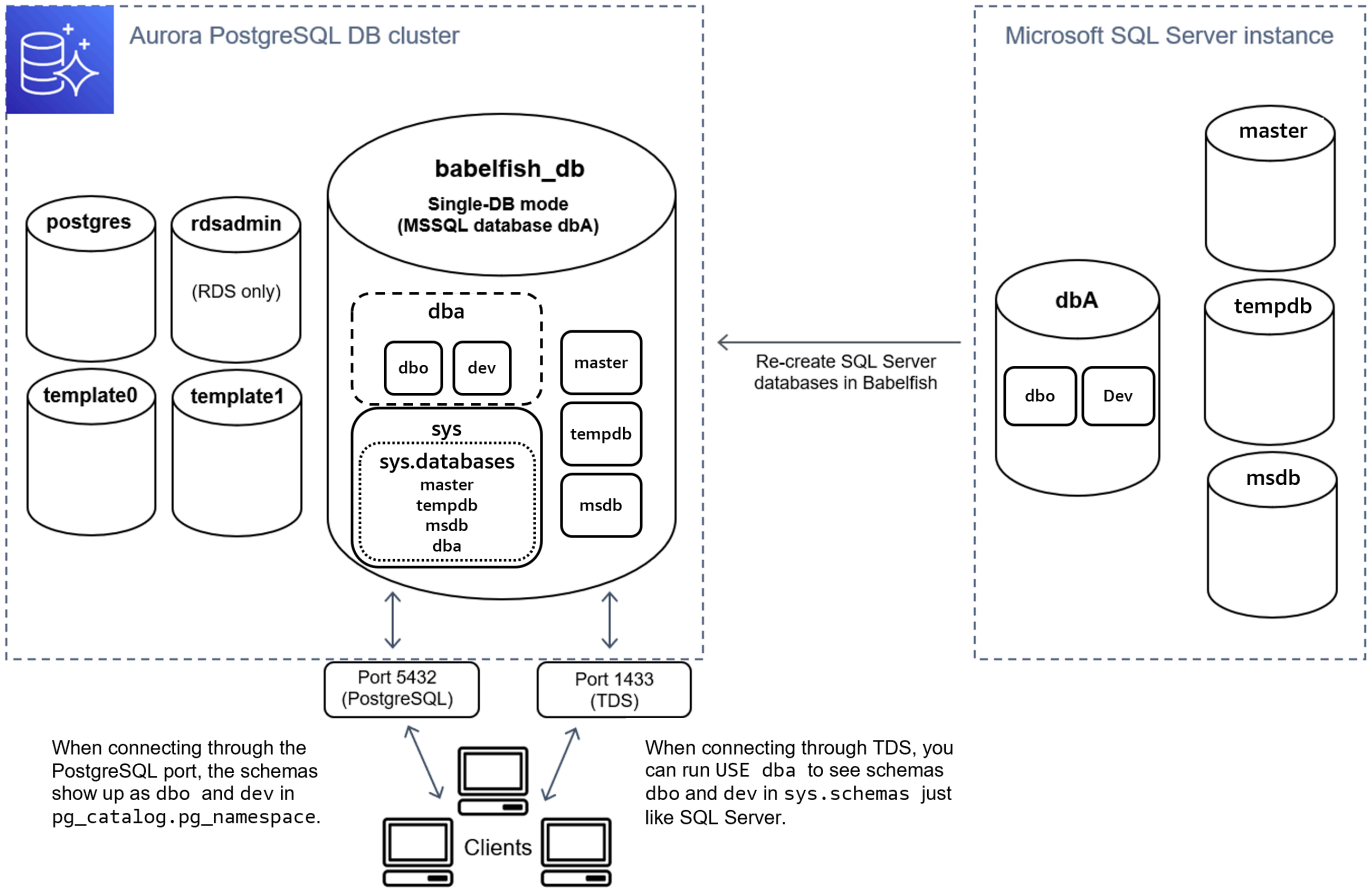
> **Note**
>
> 在 Aurora PostgreSQL 叢集中，為 Babelfish 保留了 babelfish_db 資料庫名稱。若自行在 Babelfish 資料庫叢集上建立 "babelfish_db" 資料庫，可能會導致 Aurora 無法成功佈建 Babelfish。

連線至 TDS 連接埠時，工作階段就放在 babelfish_db 資料庫中。就 T-SQL 而言，結構看似連線至 SQL Server 執行個體。您會看到 master、msdb 和 tempdb 資料庫，以及 sys.databases 目錄。您可以建立其他使用者資料庫，並使用 USE 陳述式來切換資料庫。您建立的 SQL Server 使用者資料庫會扁平化為 babelfish_db PostgreSQL 資料庫。您的資料庫保留同於或類似於 SQL Server 提供的跨資料庫語法和語意。

搭配單一資料庫或多個資料庫來使用 Babelfish

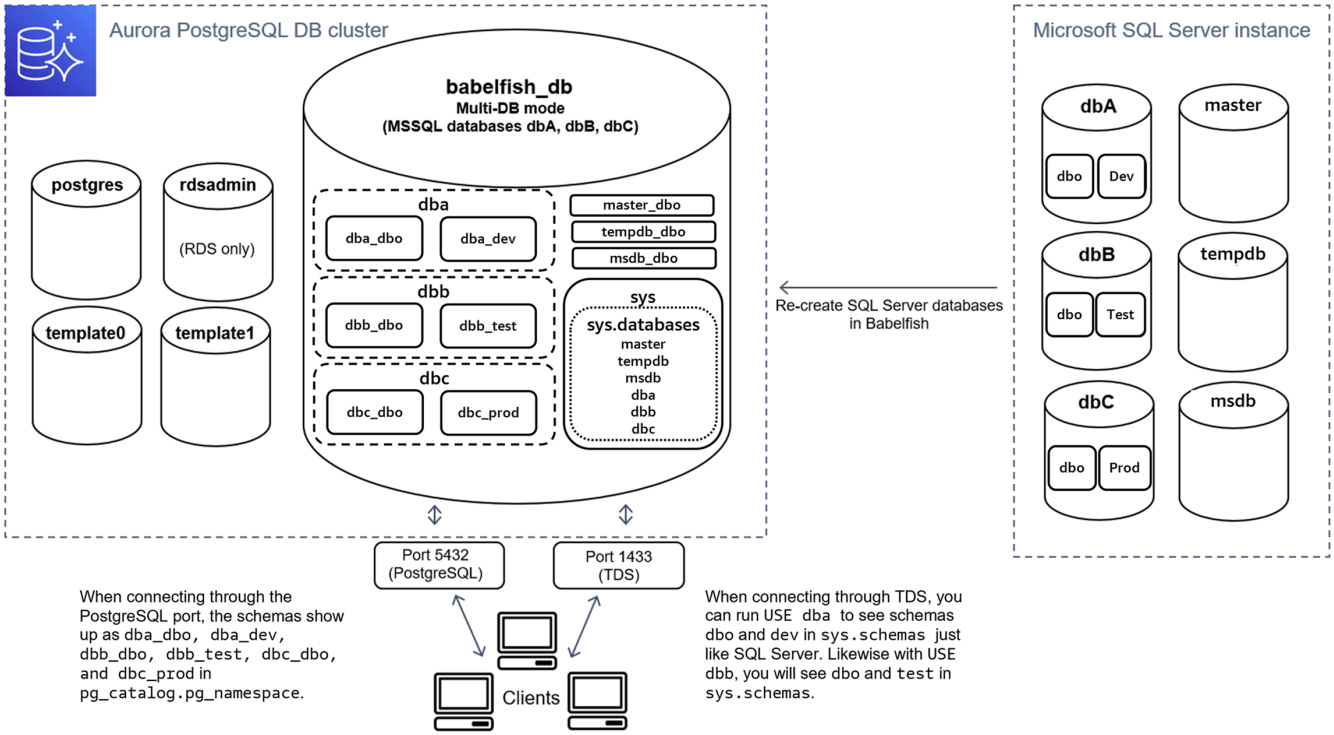
建立適用於 Babelfish 的 Aurora PostgreSQL 叢集時，您可以選擇單獨使用一個 SQL Server 資料庫，或同時使用多個 SQL Server 資料庫。您的選擇會影響 babelfish_db 資料庫內的 SQL Server 結構描述名稱如何出現在 Aurora PostgreSQL 中。遷移模式存放在 migration_mode 參數中。您不得在建立叢集之後變更此參數，因為您可能會失去先前建立之所有 SQL 物件的存取權。

在單一資料庫模式中，SQL Server 資料庫的結構描述名稱在 PostgreSQL 的 babelfish_db 資料庫中仍保持相同。如果您選擇只遷移單一資料庫，則可以利用 SQL Server 中使用的同一名稱，在 PostgreSQL 中參考已遷移之使用者資料庫的結構描述名稱。例如，dbo 和 smith 結構描述位於 dbA 資料庫內。



透過 TDS 連線時，您可以執行 USE dba 來查看架構描述 dbo 和 dev，就像在 SQL Server 中一樣。從 PostgreSQL 也可以看到未變更的結構描述名稱。

在多資料庫模式中，若從 PostgreSQL 存取，使用者資料庫的結構描述名稱變成 dbname_schemaname。若從 T-SQL 存取，結構描述名稱保持不變。



如圖所示，當透過 TDS 連接埠連線及使用 T-SQL 時，多資料庫模式和單資料庫模式與 SQL Server 相同。例如：USE dbA 會列出結構描述 dbo 和 dev，就如同在 SQL 伺服器中一樣。從 PostgreSQL 可以看到對應的結構描述名稱，例如 dba_dbo 和 dba_dev。

每個資料庫仍然包含您的結構描述。SQL Server 結構描述名稱的前面加上每個資料庫的名稱，以底線為分隔符號，例如：

- dba 包含 dba_dbo 和 dba_dev。
- dbb 包含 dbb_dbo 和 dbb_test。
- dbc 包含 dbc_dbo 和 dbc_prod。

在 babelfish_db 資料庫內，T-SQL 使用者仍需執行 USE dbname 來切換資料庫環境，外觀和風格上仍類似於 SQL Server。

選擇遷移模式

每個遷移模式各有優缺點。請根據您的使用者資料庫數目和遷移計畫來選擇遷移模式。在建立與 Babelfish 搭配使用的叢集之後，您不得變更遷移模式，因為可能失去所有先前建立之 SQL 物件的存取權。選擇移轉模式時，請考量使用者資料庫和用戶端的需求。

建立適用於 Babelfish 的叢集時，Aurora PostgreSQL 會建立系統資料庫 `master` 和 `tempdb`。如果您在系統資料庫中 (`master` 或 `tempdb`) 建立或修改物件，請務必在新叢集中重新建立這些物件。與 SQL Server 不同，在 `tempdb` 叢集重新啟動後，Babelfish 不會重新初始化。

在下列情況下，請使用單一資料庫遷移模式：

- 如果您遷移單一 SQL Server 資料庫。在單一資料庫模式中，從 PostgreSQL 存取的遷移結構描述名稱與原始 SQL Server 結構描述名稱相同。若您想要最佳化現有 SQL 查詢，以便透過 PostgreSQL 連線執行，這會減少對現有 SQL 查詢的程式碼變更。
- 如果最終目標是完整遷移至原生 Aurora PostgreSQL。在遷移之前，請將結構描述合併為單一結構描述 (dbo)，然後遷移至單一叢集，以減少必要的變更。

在下列情況下，請使用多資料庫遷移模式：

- 若您想要在同一個執行個體中擁有多個使用者資料庫的預設 SQL Server 體驗。
- 若需要一起移轉多個使用者資料庫。

Babelfish 的資料庫叢集參數群組設定

當您建立 Aurora PostgreSQL 資料庫叢集並選擇 Turn on Babelfish (開啟 Babelfish) 時，如果您選擇 Create new (新建)，則會自動為您建立資料庫叢集參數群組。此資料庫叢集參數群組以針對為此安裝所選 Aurora PostgreSQL 版本 (例如 Aurora PostgreSQL 第 14 版) 的 Aurora PostgreSQL 資料庫叢集參數群組為基礎。它使用下列一般模式命名：

```
custom-aurora-postgresql14-babelfish-compat-3
```

您可以在叢集建立過程中變更以下設定，但其中一些設定一旦儲存在自訂參數群組後就無法變更，因此請謹慎選擇：

- 單一資料庫或多個資料庫
- 預設定序地區設定
- 定序名稱
- DB parameter group (資料庫參數群組)

若要使用現有的 Aurora PostgreSQL 資料庫叢集第 13 版或更新版本的參數群組，請編輯該資料庫，並將 `babelfish_status` 參數設定為 on。請在建立 Aurora PostgreSQL 叢集之前指定任何 Babelfish 選項。如需詳細資訊，請參閱 Amazon Aurora 的參數群組。

下列參數控制 Babelfish 偏好設定。除非在「說明」中另有說明，否則參數是可修改的。預設值包含在描述中。若要查看任何參數的允許值，請執行以下操作：

> **Note**
>
> 當您建立新資料庫參數群組與資料庫執行個體的關聯時，只有在資料庫執行個體重新開機之後，才會套用修改過的靜態參數和動態參數。不過，如果您在將資料庫參數群組與資料庫執行個體建立關聯之後修改該群組中的動態參數，則會立即套用這些變更，而不需重新開機。

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 從導覽功能表中，選擇 Parameter groups (參數群組)。
3. 從清單中選擇 `default.aurora-postgresql14` 資料庫叢集參數群組。
4. 在搜尋欄位中輸入參數的名稱。例如，在搜尋欄位中輸入 `babelfishpg_tsql.default_locale` 以顯示此參數，以及其預設值和允許的設定。

> **Note**
>
> 只有在那些區域中開啟下列參數時，Babelfish for Aurora PostgreSQL 全域資料庫才會在次要區域中運作。

參數	Description	套用類型	是可修改的
babelfishpg_tsql.apg_enable_correlated_scalar_transform	讓規劃器能夠轉換 Babelfish 中的相關純量子查詢。(預設值：開啟) (允許值：開啟、關閉)	動態	true
babelfishpg_tsql.apg_enable_subquery_cache	啟用 Babelfish 中相關純量子查詢的快取使用。(預設值：開啟) (允許值：開啟、關閉)	動態	true
babelfishpg_tds.tds_default_numeric_scale	針對要在 TDS 資料欄中繼資料中傳送的數值類型，設定預設小數位數 (如果引擎未指定)。(預設值：8) (允許值：0–38)	動態	true
babelfishpg_tds.tds_default_numeric_precision	此整數會針對要在 TDS 資料欄中繼資料中傳送的數值類型，設定預設精確度 (如果引擎未指定)。(預設值：38) (允許值：1–38)	動態	true
babelfishpg_tds.tds_default_packet_size	此整數設定用於連線 SQL Server 用戶端的預設封包大小。(預設	動態	true

參數	Description	套用類型	是可修改的
	值：4096) (允許值：512–32767)		
babelfishpg_tds.tds_default_protocol_version	此整數可設定用於連線用戶端的預設 TDS 通訊協定版本。(預設值：DEFAULT) (允許的值：TDSv7.0、TDSv7.1、TDSv7.1.1、TDSv7.2、TDSv7.3A、TDSv7.3B、TDSv7.4、DEFAULT)	動態	true
babelfishpg_tds.default_server_name	此字串可識別 Babelfish 伺服器的預設名稱。(預設值：Microsoft SQL Server) (允許值：null)	動態	true
babelfishpg_tds.tds_debug_log_level	此整數設定 TDS 中的記錄層級；0 會關閉日誌。(預設值：1) (允許值：0、1、2、3)	動態	true
babelfishpg_tds.listen_addresses	此字串可設定用來接聽 TDS 的主機名稱或一個或多個 IP 地址。Babelfish 資料庫叢集建立之後，便無法變更此參數。(預設值：*) (允許值：null)	–	false

參數	Description	套用類型	是可修改的
babelfishpg_tds.port	此整數在 SQL Server 語法中指定用於請求的 TCP 連接埠。(預設值：1433) (允許值：1–65535)	靜態	true
babelfishpg_tds.tds_ssl_encrypt	此布林值會針對周遊 TDS 接聽程式連接埠的資料開啟 (0) 或關閉 (1) 加密。如需有關使用 SSL 進行用戶端連線的詳細資訊，請參閱Babelfish SSL 設定與用戶端連線。(預設值：0) (允許值：0、1)	動態	true
babelfishpg_tds.tds_ssl_max_protocol_version	此字串指定用於 TDS 工作階段的最高 SSL/TLS 通訊協定版本。(預設值：'TLSv1.2') (允許值：'TLSv1'、'TLSv1.1'、'TLSv1.2')	動態	true

參數	Description	套用類型	是可修改的
babelfishpg_tds.tds_ssl_min_protocol_version	此字串指定用於 TDS 工作階段的最低 SSL/TLS 通訊協定版本。(預設值：Aurora PostgreSQL 第 16 版中的 'TLSv1.2'，'TLSv1' 適用於 Aurora PostgreSQL 第 16 版之前的版本) (允許值：'TLSv1'、'TLSv1.1'、'TLSv1.2')	動態	true
babelfishpg_tds.unix_socket_directories	此字串識別 TDS 伺服器 Unix 通訊端目錄。Babelfish 資料庫叢集建立之後，便無法變更此參數。(預設值：/tmp) (允許值：null)	–	false
babelfishpg_tds.unix_socket_group	此字串識別 TDS 伺服器 Unix 通訊端群組。Babelfish 資料庫叢集建立之後，便無法變更此參數。(預設值：rdsdb) (允許值：null)	–	false

參數	Description	套用類型	是可修改的
babelfishpg_tsql.default_locale	此字串指定用於 Babelfish 定序的預設地區設定。預設地區設定只是地區設定，不含任何限定詞。 請在佈建 Babelfish 資料庫叢集時設定此參數。佈建資料庫叢集之後，會忽略對此參數的變更。(預設值：en_US) (允許值：請參閱資料表)	靜態	true
babelfishpg_tsql.migration_mode	此不可修改的清單指定支援單個或多個使用者資料庫。請在佈建 Babelfish 資料庫叢集時設定此參數。佈建資料庫叢集之後，您無法修改此參數的值。(預設值：Aurora PostgreSQL 第 16 版的 multi-db，single-db 適用於 Aurora PostgreSQL 第 16 版之前的版本) (允許值：Single-db、multi-db、null)	靜態	true

參數	Description	套用類型	是可修改的
babelfishpg_tsql.server_collation_name	此字串指定用於伺服器層級動作的定序名稱。請在佈建 Babelfish 資料庫叢集時設定此參數。佈建資料庫叢集之後，請勿修改此參數的值。(預設值：bbf_unicode_general_ci_as) (允許值：請參閱資料表)	靜態	true
babelfishpg_tsql.version	此字串設定 @@VERSION 變數的輸出。請勿對 Aurora PostgreSQL 資料庫叢集修改此值。(預設值：null) (允許值：default)	動態	true
rds.babelfish_status	此字串設定 Babelfish 功能的狀態。此參數設定為 `datatypes only` 時，Babelfish 會停用，但 SQL Server 資料類型仍然可用。(預設值：off) (允許值：on、off、datatypesonly)	靜態	true

參數	Description	套用類型	是可修改的
unix_socket_permissions	此整數設定 TDS 伺服器 Unix 通訊端許可。Babelfish 資料庫叢集建立之後，便無法變更此參數。(預設值：0700) (允許值：0–511)	–	false

Babelfish SSL 設定與用戶端連線

若需要連至 Babelfish for Aurora PostgreSQL 資料庫叢集的 SSL/TLS 連線，請使用 `rds.force_ssl` 參數。

- 若需要 SSL/TLS 連線，請將 `rds.force_ssl` 參數值設為 1 (開啟)。
- 若要關閉所需的 SSL/TLS 連線，請將 `rds.force_ssl` 參數值設為 0 (關閉)。

此參數的預設值取決於 Aurora PostgreSQL 版本：

- 對於 Aurora PostgreSQL 第 17 版及更新版本：預設值為 1 (開啟)。
- 對於 Aurora PostgreSQL 第 16 版及更舊版本：預設值為 0 (關閉)。

> **Note**
>
> 當您從 Aurora PostgreSQL 第 16 版或更舊版本執行升級至第 17 版或更新版本的主要版本升級時，參數的預設值會從 0 (關閉) 變更為 1 (開啟)。此變更可能會導致未針對 SSL 設定的應用程式連線能力失敗。您可以將此參數設定為 0 (關閉)，以還原至先前的預設行為。

如需驅動程式特定的詳細資訊，請參閱連線至 Babelfish 資料庫叢集。

當用戶端連線至 TDS 連接埠 (預設 1433) 時，Babelfish 會比較用戶端交握期間傳送的 Secure Sockets Layer (SSL) 設定與 Babelfish SSL 參數設定 (`tds_ssl_encrypt`)。然後，Babelfish 會決定是否允許連線。如果允許連線，則強制或不強制執行加密行為，視您的參數設定和用戶端提供的加密支援而定。

下表顯示 Babelfish 對每個組合的反應。

用戶端 SSL 設定	Babelfish SSL 設定	rds.force_ssl	允許連線？	傳回給用戶端的值
ENCRYPT_ON	任何	任何	允許，加密整個連線	ENCRYPT_ON
ENCRYPT_OFF	tds_ssl_encrypt=1	任何	允許，加密整個連線	ENCRYPT_REQ

用戶端 SSL 設定	Babelfish SSL 設定	rds.force_ssl	允許連線？	傳回給用戶端的值
ENCRYPT_OFF	tds_ssl_encrypt=0	rds.force_ssl=0	允許，加密登入封包	ENCRYPT_OFF
ENCRYPT_OFF	tds_ssl_encrypt=0	rds.force_ssl=1	否，關閉連線	ENCRYPT_OFF
ENCRYPT_NOT_SUP	tds_ssl_encrypt=0	rds.force_ssl=0	是	ENCRYPT_NOT_SUP
ENCRYPT_NOT_SUP	tds_ssl_encrypt=1	任何	否，關閉連線	ENCRYPT_REQ
ENCRYPT_NOT_SUP	tds_ssl_encrypt=0	rds.force_ssl=1	否，關閉連線	ENCRYPT_NOT_SUP
ENCRYPT_CLIENT_CERT	任何	任何	否，關閉連線	不支援

了解 Babelfish for Aurora PostgreSQL 中的定序

當您使用 Babelfish 建立 Aurora PostgreSQL 資料庫叢集時，您可以為您的資料選擇定序。定序可指定排序順序，及以人類書寫語言產生文字或字元的位元模式。定序中包含規則，可用於比較一組所給定位元模式的資料。定序與當地語系化相關。不同的地區設定會影響字元對應、排序順序等。定序屬性會反映在各種定序的名稱中。如需有關這些屬性的詳細資訊，請參閱 Babelfish collation attributes table。

Babelfish 將 SQL Server 定序映射至 Babelfish 提供的類似定序。Babelfish 預先定義支援文化差異性字串比較和排序順序的 Unicode 定序。Babelfish 也能夠將 SQL Server 資料庫中的定序轉化為最相符的 Babelfish 定序。針對不同語言和區域，已提供地區設定專用的定序。

某些定序指定對應於用戶端編碼的字碼頁。Babelfish 根據每個輸出資料欄的定序，自動將伺服器編碼轉化為用戶端編碼。

Babelfish 支援 Babelfish supported collations table 中所列的定序。Babelfish 將 SQL Server 定序映射至 Babelfish 提供的類似定序。

Babelfish 使用第 153.80 版 International Components for Unicode (ICU) 定序程式庫。如需 ICU 定序的詳細資訊，請參閱 ICU 文件中的定序。若要進一步了解 PostgreSQL 和定序，請參閱 PostgreSQL 文件中的定序支援。

主題

- 可控制定序和地區設定的資料庫叢集參數
- Babelfish 中的確定性和非確定性定序
- 在 Babelfish 中資料庫層級支援的定序
- Babelfish 中的伺服器和物件定序
- Babelfish 中的預設定序行為
- 管理定序
- 定序限制與行為差異

可控制定序和地區設定的資料庫叢集參數

以下參數會影響定序行為。

babelfishpg_tsql.default_locale

> 此參數指定該定序使用的預設地區設定。此參數用於與 Babelfish collation attributes table 中所列的屬性結合，以自訂特定語言和區域的定序。此參數的預設值為 en-US。

預設地區設定會套用至所有以字母 "BBF" 開頭的 Babelfish 定序，也會套用至對應至 Babelfish 定序的所有 SQL Server 定序。在現有 Babelfish 資料庫叢集上變更此參數的設定，並不會影響現有定序的地區設定。如需定序清單，請參閱 Babelfish supported collations table。

babelfishpg_tsql.server_collation_name

此參數會指定伺服器 (Aurora PostgreSQL 資料庫叢集執行個體) 和資料庫的預設定序。預設值為 `sql_latin1_general_cp1_ci_as`。`server_collation_name` 必須是 `CI_AS` 定序，因為在 T-SQL 中，伺服器定序決定如何比較識別符。

當您建立 Babelfish 資料庫叢集時，請從可選取清單中選擇 Collation name (定序名稱)。這些包括 Babelfish supported collations table 中所列的定序。建立 Babelfish 資料庫之後，請勿修改 `server_collation_name`。

您在建立 Babelfish for Aurora PostgreSQL 資料庫叢集時所選擇的設定，會儲存在與這些參數的叢集相關聯的資料庫叢集參數群組中，並設定其定序行為。

Babelfish 中的確定性和非確定性定序

Babelfish 支援確定性和非確定性定序：

- 確定性定序會將位元組序列相同的字元評估為相等。這表示 `x` 和 `X` 在確定性定序中不相等。確定性定序區分大小寫 (CS) 和區分重音 (AS)。
- 非確定性定序不要求完全相符。非確定性定序將 `x` 和 `X` 評估為相同。非確定性定序不區分大小寫 (CI) 也不區分重音 (AI)，或兩者都是。

在下表中，您可以在使用非確定性定序找到 Babelfish 和 PostgreSQL 之間的一些行為差異。

Babelfish	PostgreSQL
支援 CI_AS 定序的 LIKE 子句。	在非確定性定序上不支援 LIKE 子句。
僅支援來自 Babelfish 4.2.0 版中下列 AI 定序的 LIKE 子句： • bbf_unicode_cp1250_ci_ai • bbf_unicode_cp1250_cs_ai • bbf_unicode_cp1257_ci_ai • bbf_unicode_cp1257_cs_ai	在非確定性定序上不支援 LIKE 子句。

Babelfish	PostgreSQL
• bbf_unicode_cp1_ci_ai • bbf_unicode_cp1_cs_ai • estonian_ci_ai • finnish_swedish_ci_ai • french_ci_ai • latin1_general_ci_ai • latin1_general_cs_ai • modern_spanish_ci_ai • polish_ci_ai • sql_latin1_general_cp1_ci_ai • sql_latin1_general_cp1_cs_ai • traditional_spanish_ci_ai	

如需 Babelfish 與 SQL Server 和 PostgreSQL 比較之下的其他限制和行為差異清單，請參閱 定序限制與行為差異。

Babelfish 和 SQL Server 遵循定序命名慣例來描述定序屬性，如下表所示。

屬性	描述
AI	不區分重音。
AS	區分重音。
BIN2	BIN2 要求以字碼指標順序來儲存資料。Unicode 字碼指標順序與 UTF-8、UTF-16 和 UCS-2 編碼的字元順序相同。字碼指標順序是快速的確定性定序。
CIS	不區分大小寫。
CS	區分大小寫。
PREF	若要將小寫字母排在大寫字母前面，請使用 PREF 定序。如果比較時不區分大小寫，且沒有其他差別，則大寫字母排在小寫字母前置。ICU 程式庫支援 `colCaseFirst=upper` 的大寫偏好設定，但不適用於 CI_AS 定序。

屬性	描述
	PREF 只能套用至 CS_AS 確定性定序。

在 Babelfish 中資料庫層級支援的定序

在 Babelfish 中資料庫層級支援下列定序：

- bbf_unicode_bin2
- bbf_unicode_cp1_ci_ai
- bbf_unicode_cp1_ci_as
- bbf_unicode_cp1250_ci_ai
- bbf_unicode_cp1250_ci_as
- bbf_unicode_cp1257_ci_ai
- bbf_unicode_cp1257_ci_as
- estonian_ci_ai
- estonian_ci_as
- finnish_swedish_ci_ai
- finnish_swedish_ci_as
- french_ci_ai
- french_ci_as
- latin1_general_bin2
- latin1_general_ci_ai
- latin1_general_ci_as
- latin1_general_90_bin2
- latin1_general_100_bin2
- latin1_general_140_bin2
- modern_spanish_ci_ai
- modern_spanish_ci_as
- polish_ci_ai
- polish_ci_as
- sql_latin1_general_cp1_ci_ai

- sql_latin1_general_cp1_ci_as
- sql_latin1_general_cp1250_ci_as
- sql_latin1_general_cp1251_ci_as
- sql_latin1_general_cp1257_ci_as
- traditional_spanish_ci_ai
- traditional_spanish_ci_as

> **Note**
>
> 若要在資料庫層級使用不同的定序，請確定其符合伺服器層級定序。如需詳細資訊，請參閱Babelfish 中的伺服器和物件定序

Babelfish 中的伺服器和物件定序

使用下列定序作為伺服器定序或物件定序。

定序 ID	備註
bbf_unicode_general_ci_as	支援不區分大小寫比較和 LIKE 運算子。
bbf_unicode_cp1_ci_as	也稱為 CP1252 的非確定性定序。
bbf_unicode_CP1250_ci_as	在採用拉丁字母的中歐和東歐語言中用於表示文字的非確定性定序。
bbf_unicode_CP1251_ci_as	採用斯拉夫字母的語言所用的非確定性定序。
bbf_unicode_cp1253_ci_as	用於表示現代希臘文的非確定性定序。
bbf_unicode_cp1254_ci_as	支援土耳其文的非確定性定序。
bbf_unicode_cp1255_ci_as	支援希伯來文的非確定性定序。

定序 ID	備註
bbf_unicode_cp1256_ci_as	書寫採用阿拉伯字母的語言所用的非確定性定序。
bbf_unicode_cp1257_ci_as	用於支援愛沙尼亞、拉脫維亞和立陶宛語言的非確定性定序。
bbf_unicode_cp1258_ci_as	用於書寫越南文字元的非確定性定序。
bbf_unicode_cp874_ci_as	用於書寫泰文字元的非確定性定序。
sql_latin1_general_cp1250_ci_as	用於表示拉丁字元的非確定性單位元組字元編碼。
sql_latin1_general_cp1251_ci_as	支援拉丁字元的非確定性定序。
sql_latin1_general_cp1_ci_as	支援拉丁字元的非確定性定序。
sql_latin1_general_cp1253_ci_as	支援拉丁字元的非確定性定序。
sql_latin1_general_cp1254_ci_as	支援拉丁字元的非確定性定序。
sql_latin1_general_cp1255_ci_as	支援拉丁字元的非確定性定序。
sql_latin1_general_cp1256_ci_as	支援拉丁字元的非確定性定序。

定序 ID	備註
sql_latin1_general_cp1257_ci_as	支援拉丁字元的非確定性定序。
sql_latin1_general_cp1258_ci_as	支援拉丁字元的非確定性定序。
chinese_prc_ci_as	支援簡體中文的非確定性定序。
cyrillic_general_ci_as	支援斯拉夫文的非確定性定序。
finnish_swedish_ci_as	支援芬蘭文的非確定性定序。
french_ci_as	支援法文的非確定性定序。
japanese_ci_as	支援日文的非確定性定序。Babelfish 2.1.0 及更高版本支援。
korean_wansung_ci_as	支援韓文 (字典腔) 的非確定性定序。
latin1_general_ci_as	支援拉丁字元的非確定性定序。
modern_spanish_ci_as	支援現代西班牙文的非確定性定序。
polish_ci_as	支援波蘭文的非確定性定序。
thai_ci_as	支援泰文的非確定性定序。
traditional_spanish_ci_as	支援西班牙文 (傳統腔) 的非確定性定序。
turkish_ci_as	支援土耳其文的非確定性定序。

定序 ID	備註
ukrainian_ci_as	支援烏克蘭文的非確定性定序。
vietnamese_ci_as	支援越南文的非確定性定序。

您可以使用下列定序作為物件定序。

方言	確定性選項	非確定性選項
Arabic	Arabic_CS_AS	Arabic_CI_AS Arabic_CI_AI
阿拉伯字母	BBF_Unicode_CP1256_CS_AS BBF_Unicode_Pref_CP1256_CS_AS	BBF_Unicode_CP1256_CI_AI BBF_Unicode_CP1256_CS_AI
二進位	latin1_general_bin2 BBF_Unicode_BIN2	–
使用拉丁字母的中歐和東歐語言	BBF_Unicode_CP1250_CS_AS BBF_Unicode_Pref_CP1250_CS_AS	BBF_Unicode_CP1250_CI_AI BBF_Unicode_CP1250_CS_AI
中文	Chinese_PRC_CS_AS	Chinese_PRC_CI_AS Chinese_PRC_CI_AI
Cyrillic_General	Cyrillic_General_CS_AS	Cyrillic_General_CI_AS Cyrillic_General_CI_AI

方言	確定性選項	非確定性選項
斯拉夫字母	BBF_Unicode_CP1251_CS_AS BBF_Unicode_Pref_CP1251_CS_AS	BBF_Unicode_CP1251_CI_AI BBF_Unicode_CP1251_CS_AI
Estonian	Estonian_CS_AS	Estonian_CI_AS Estonian_CI_AI
愛沙尼亞文、拉脫維亞文和立陶宛文	BBF_Unicode_CP1257_CS_AS BBF_Unicode_Pref_CP1257_CS_AS	BBF_Unicode_CP1257_CI_AI BBF_Unicode_CP1257_CS_AI
Finnish_Swedish	Finnish_Swedish_CS_AS	Finnish_Swedish_CI_AS Finnish_Swedish_CI_AI
French	French_CS_AS	French_CI_AS French_CI_AI
Greek	Greek_CS_AS	Greek_CI_AS Greek_CI_AI
Hebrew	BBF_Unicode_CP1255_CS_AS BBF_Unicode_Pref_CP1255_CS_AS Hebrew_CS_AS	BBF_Unicode_CP1255_CI_AI BBF_Unicode_CP1255_CS_AI Hebrew_CI_AS Hebrew_CI_AI
日文 (Babelfish 2.1.0 及更新版本)	Japanese_CS_AS	Japanese_CI_AI Japanese_CI_AS

方言	確定性選項	非確定性選項
Korean_Wamsung	Korean_Wamsung_CS_AS	Korean_Wamsung_CI_AS Korean_Wamsung_CI_AI
程式碼頁面 CP1252 的拉丁字元	latin1_general_cs_as BBF_Unicode_General_CS_AS BBF_Unicode_General_Pref_CS_AS BBF_Unicode_Pref_CP1_CS_AS BBF_Unicode_CP1_CS_AS	latin1_general_ci_as latin1_general_ci_ai latin1_general_cs_ai BBF_Unicode_General_CI_AI BBF_Unicode_General_CS_AI BBF_Unicode_CP1_CI_AI BBF_Unicode_CP1_CS_AI
現代希臘文	BBF_Unicode_CP1253_CS_AS BBF_Unicode_Pref_CP1253_CS_AS	BBF_Unicode_CP1253_CI_AI BBF_Unicode_CP1253_CS_AI
Modern_Spanish	Modern_Spanish_CS_AS	Modern_Spanish_CI_AS Modern_Spanish_CI_AI
Mongolian	Mongolian_CS_AS	Mongolian_CI_AS Mongolian_CI_AI
Polish	Polish_CS_AS	Polish_CI_AS Polish_CI_AI

方言	確定性選項	非確定性選項
Thai	BBF_Unicode_CP874_CS_AS BBF_Unicode_Pref_CP874_CS_AS Thai_CS_AS	BBF_Unicode_CP874_CI_AI BBF_Unicode_CP874_CS_AI Thai_CI_AS、Thai_CI_AI
Traditional_Spanish	Traditional_Spanish_CS_AS	Traditional_Spanish_CI_AS Traditional_Spanish_CI_AI
Turkish	BBF_Unicode_CP1254_CS_AS BBF_Unicode_Pref_CP1254_CS_AS Turkish_CS_AS	BBF_Unicode_CP1254_CI_AI BBF_Unicode_CP1254_CS_AI Turkish_CI_AS、Turkish_CI_AI
Ukranian	Ukranian_CS_AS	Ukranian_CI_AS Ukranian_CI_AI
越南文	BBF_Unicode_CP1258_CS_AS BBF_Unicode_Pref_CP1258_CS_AS Vietnamese_CS_AS	BBF_Unicode_CP1258_CI_AI BBF_Unicode_CP1258_CS_AI Vietnamese_CI_AS Vietnamese_CI_AI

Babelfish 中的預設定序行為

早期，可定序資料類型的預設定序是 `pg_catalog.default`。資料類型和依賴這些資料類型的物件遵循區分大小寫的定序。此情況可能會影響定序不區分大小寫之資料集的 T-SQL 物件。從 Babelfish 2.3.0 開始，可定序資料類型 (TEXT 和 NTEXT 除外) 的預設定序 與 `babelfishpg_tsql.server_collation_name` 參數中的定序相同。當您升級至 Babelfish 2.3.0 時，系統會在建立資料庫叢集時自動挑選預設定序，這不會產生任何可見的影響。

管理定序

ICU 程式庫提供定序版本追蹤，以確保有新版本的 ICU 可用時，重新編製依賴定序的索引。若要查看目前的資料庫是否有定序需要重新整理，您可以在使用 psql 或 pgAdmin 連線後使用下列查詢：

```
SELECT pg_describe_object(refclassid, refobjid,
    refobjsubid) AS "Collation",
    pg_describe_object(classid, objid, objsubid) AS "Object"
    FROM pg_depend d JOIN pg_collation c ON refclassid = 'pg_collation'::regclass
    AND refobjid = c.oid WHERE c.collversion <> pg_collation_actual_version(c.oid)
    ORDER BY 1, 2;
```

此查詢會傳回如下所示的輸出：

```
 Collation | Object
-----------+--------
(0 rows)
```

在此範例中，沒有任何定序需要重新整理。

若要取得 Babelfish 資料庫中預先定義的定序清單，您可以將 psql 或 pgAdmin 搭配下列查詢使用：

```
SELECT * FROM pg_collation;
```

預先定義的定序會存放在 sys.fn_helpcollations 資料表中。您可以使用下列命令來顯示定序的資訊 (例如 lcid、樣式和定序旗標)。若要使用 sqlcmd 取得所有定序的清單，請連線至 T-SQL 連接埠 (預設為 1433) 並執行下列查詢：

```
1> :setvar SQLCMDMAXVARTYPEWIDTH 40
2> :setvar SQLCMDMAXFIXEDTYPEWIDTH 40
3> SELECT * FROM fn_helpcollations()
4> GO
name                                  description
------------------------------------- ----------------------------------------
arabic_cs_as                          Arabic, case-sensitive, accent-sensitive
arabic_ci_ai                          Arabic, case-insensitive, accent-insensi
arabic_ci_as                          Arabic, case-insensitive, accent-sensiti
bbf_unicode_bin2                      Unicode-General, case-sensitive, accent-
bbf_unicode_cp1250_ci_ai              Default locale, code page 1250, case-ins
bbf_unicode_cp1250_ci_as              Default locale, code page 1250, case-ins
```

```
bbf_unicode_cp1250_cs_ai                Default locale, code page 1250, case-sen
bbf_unicode_cp1250_cs_as                Default locale, code page 1250, case-sen
bbf_unicode_pref_cp1250_cs_as           Default locale, code page 1250, case-sen
bbf_unicode_cp1251_ci_ai                Default locale, code page 1251, case-ins
bbf_unicode_cp1251_ci_as                Default locale, code page 1251, case-ins
bbf_unicode_cp1254_ci_ai                Default locale, code page 1254, case-ins
...
(124 rows affected)
```

範例中顯示的第 1 行和第 2 行會縮小輸出範圍，這僅用於文件可讀性目的。

```
1> SELECT SERVERPROPERTY('COLLATION')
2> GO
serverproperty
------------------------------------------------------------------------
sql_latin1_general_cp1_ci_as

(1 rows affected)
1>
```

定序限制與行為差異

Babelfish 使用 ICU 程式庫來支援定序。PostgreSQL 以特定版本的 ICU 建置，最多只能符合定序的一個版本。不同版本的變化不可避免，就像語言隨著時間演變，也會發生微小的變化。您下列清單中可以找到 Babelfish 定序的已知限制和行為變化：

- 索引與定序類型相依性 – 依據 International Components for Unicode (ICU) 定序程式庫 (Babelfish 使用的程式庫) 的使用者定義類型索引，不會在程式庫版本變更時失效。
- COLLATIONPROPERTY 函數 – 針對支援的 Babelfish BBF 定序實作定序屬性。如需更多資訊，請參閱Babelfish supported collations table。
- Unicode 排序規則差異 – SQL Server 的 SQL 定序將 Unicode 編碼的資料 (`nchar` 和 `nvarchar`) 排序的方式不同於非 Unicode 編碼的資料 (`char` 和 `varchar`)。Babelfish 資料庫一律為 UTF-8 編碼，且不論資料類型為何，一律以一致的方式套用 Unicode 排序規則，因此 `char` 或 `varchar` 的排序順序與 `nchar` 或 `nvarchar` 的相同。
- 第二級相等定序與排序行為 – 預設 ICU Unicode 第二級相等 (`CI_AS`) 定序將標點符號和其他非英數字元排在數值字元前面，並將數值字元排在字母字元前面。但是，標點符號和其他特殊字元的順序不同。
- Tertiary collations, workaround for ORDER BY – SQL 定序 (例如 `SQL_Latin1_General_Pref_CP1_CI_AS`) 支援 `TERTIARY_WEIGHTS` 函數，且能夠將 `CI_AS`

定序中視為同等的字串排序為大寫優先：ABC、ABc、AbC、Abc、aBC、aBc、abC，最後是 abc。因此，DENSE_RANK OVER (ORDER BY column) 分析函數將這些字串評估為相同等級，但在分割區內以大寫優先來排序。

在 Babelfish 中，您可以在指定第三級 CS_AS 定序的 ORDER BY 子句中，新增 COLLATE 子句來指定 @colCaseFirst=upper，以獲得類似的結果。不過，colCaseFirst 修飾詞僅適用於第三級相等的字串 (而不是第二級相等，例如 CI_AS 定序)。因此，您無法使用單一 ICU 定序來模擬第三級 SQL 定序。

為了解決這種情況，建議您將使用 SQL_Latin1_General_Pref_CP1_CI_AS 定序的應用程式修改成優先使用 BBF_SQL_Latin1_General_CP1_CI_AS 定序。然後將 COLLATE BBF_SQL_Latin1_General_Pref_CP1_CS_AS 新增至此資料欄的任何 ORDER BY 子句。

- 字元擴充 – 字元擴充將單一字元視為等同於主要層級的一連串字元。SQL Server 的預設 CI_AS 定序支援字元擴展。ICU 定序僅支援不區分重音定序的字元擴展。

  需要字元擴充時，請使用 AI 定序來做比較。不過，LIKE 運算子目前不支援這種定序。

- char 與 varchar 編碼 – 當 SQL 定序用於 char 或 varchar 資料類型時，ASCII 127 之前的字元由該 SQL 定序的特定字碼頁決定排序順序。使用 SQL 定序時，宣告為 char 或 varchar 的字串與宣告為 nchar 或 nvarchar 的字串，可能以不同方式排序。

  PostgreSQL 使用資料庫編碼將所有字串編碼，因此會將所有字元轉換為 UTF-8，並根據 Unicode 規則來排序。

  因為 SQL 定序使用 Unicode 規則來排序 nchar 和 nvarchar 資料類型，所以 Babelfish 使用 UTF-8 將伺服器上的所有字串編碼。Babelfish 使用 Unicode 規則來排序 nchar 和 nvarchar 字串的方式，同於排序 char 和 varchar。

- 補增字元 – SQL Server 函數 NCHAR、UNICODE 及 LEN 支援 Unicode Basic Multilingual Plane (BMP) 以外字碼指標的字元。反之，非 SC 定序使用代理配對字元來處理增補字元。對於 Unicode 資料類型，SQL Server 可以使用 UCS-2 來表示最多 65,535 個字元，或者，如果使用增補字元，則表示完整的 Unicode 範圍 (1,114,114 個字元)。
- 區分假名 (KS) 定序 –「區分假名」(KS) 定序會將 Hiragana 和 Katakana 日文假名字元視為不同。ICU 支援日文定序標準 JIS X 4061。現在已棄用的 colhiraganaQ [on | off] 地區設定修飾詞可能有同於 KS 定序的功能。不過，Babelfish 目前不支援與 SQL Server 同名的 KS 定序。
- 區分寬度 (WS) 定序 – 將單一位元組字元 (半形) 和以雙位元組字元 (全形) 表示的同一個字元視為不同時，定序就稱為區分寬度 (WS)。不過，Babelfish 目前不支援與 SQL Server 同名的 WS 定序。
- 區分變化選擇器 (VSS) 定序 – 區分變化選擇器 (VSS) 定序會區分日文定序 Japanese_Bushu_Kakusu_140 和 Japanese_XJIS_140 中的表意變化選擇器。變化序列由一

個基本字元加上一個額外的變化選擇器組成。如果您未選取 _VSS 選項，則做比較時不會考慮變化選擇器。

Babelfish 目前不支援 VSS 定序。

- BIN 與 BIN2 定序 – BIN2 定序根據字碼指標順序來排序字元。UTF-8 的逐位元組二進制順序保留 Unicode 字碼指標順序，因此這也可能是表現最佳的定序。如果 Unicode 代碼指標順序適用於應用程式，請考慮使用 BIN2 定序。不過，使用 BIN2 定序可能會導致資料在用戶端以超出文化預期的順序顯示。隨著時間經過，Unicode 中也增加對小寫字元的新映射，因此 LOWER 函數在不同版本的 ICU 上可能有以不同方式運作。這是較常見定序版本控制問題的特殊情況，而不是 BIN2 定序所獨有。

  Babelfish 隨著 Babelfish 發佈而提供 BBF_Latin1_General_BIN2 定序，以依照 Unicode 代碼指標順序來排序。在 BIN 定序中，只有第一個字元排序為 wchar。剩餘的字元逐位元組來排序，實際上是依照符合其編碼的代碼指標順序。這種作法未遵循 Unicode 定序，Babelfish 並不支援。

- 非確定性定序和 CHARINDEX 限制 – 對於早於第 2.1.0 版的 Babelfish 版本，您無法將 CHARINDEX 與非確定性定序搭配使用。預設情況下，Babelfish 使用不區分大小寫 (非確定性) 定序。對舊版 Babelfish 使用 CHARINDEX 會造成下列執行時間錯誤：

```
nondeterministic collations are not supported for substring searches
```

> **Note**
>
> 此限制和解決方法僅適用於 Babelfish 1.x 版 (Aurora PostgreSQL 13.x 版)。Babelfish 2.1.0 及更新版本沒有此問題。

您可以透過以下其中一種方法解決此問題：

- 將表達式明確轉換為區分大小寫的定序，並套用 LOUP 或 UPER 將這兩個參數進行大小寫折疊 (case-fold)。例如，SELECT charindex('x', a) FROM t1 會變成下列：

```
SELECT charindex(LOWER('x'), LOWER(a COLLATE sql_latin1_general_cp1_cs_as)) FROM t1
```

- 建立一個 SQL 函數 f_charindex，並將 CHARINDEX 呼叫替換為對以下函數的呼叫：

```
CREATE function f_charindex(@s1 varchar(max), @s2 varchar(max)) RETURNS int
AS
BEGIN
declare @i int = 1
```

```
WHILE len(@s2) >= len(@s1)
BEGIN
  if LOWER(@s1) = LOWER(substring(@s2,1,len(@s1))) return @i
  set @i += 1
  set @s2 = substring(@s2,2,999999999)
END
return 0
END
go
```

使用逃生艙管理 Babelfish 錯誤處理

Babelfish 盡可能模仿 SQL 的控制流程和交易狀態行為。當 Babelfish 遇到錯誤時，會傳回類似 SQL Server 錯誤代碼的錯誤代碼。如果 Babelfish 無法將錯誤對應至 SQL Server 代碼，則會傳回固定的錯誤代碼 (33557097)，並會根據錯誤類型採取特定動作，如下所示：

- 針對編譯階段錯誤，Babelfish 會回復交易。
- 針對執行階段錯誤，Babelfish 會結束批次並回復交易。
- 對於用戶端和伺服器之間的通訊協定錯誤，不會回復交易。

如果錯誤代碼無法映射至同等代碼，但類似錯誤有可用代碼，則錯誤代碼會映射至替代碼。例如，造成 SQL Server 代碼 8143 和 8144 的行為都映射至 8143。

無法映射的錯誤不遵守 `TRY... CATCH` 建構。

您可以使用 `@@ERROR` 來傳回 SQL Server 錯誤代碼，或使用 `@@PGERROR` 函數來使回 PostgreSQL 錯誤代碼。您也可以使用 `fn_mapped_system_error_list` 函數來傳回映射的錯誤代碼清單。如需 PostgreSQL 錯誤代碼的詳細資訊，請參閱 PostgreSQL 網站。

修改 Babelfish 逃生艙設定

為了處理可能失敗的陳述式，Babelfish 定義幾個稱為逃生艙的選項。逃生艙選項指定 Babelfish 遇到不支援的功能或語法時採取的行為。

您可以使用 `sp_babelfish_configure` 預存程序來控制逃生艙的設定。使用指令碼將逃生艙設定為 `ignore` 或 `strict`。如果設定為 `strict`，Babelfish 會傳回錯誤，您必須更正才能繼續。

將變更套用至目前工作階段和叢集層級，包括 `server` 關鍵字。

用法如下：

- 若要列出所有逃生艙及狀態，還有使用資訊，請執行 `sp_babelfish_configure`。
- 若要列出目前工作階段或整個叢集的特定逃生艙及其值，請執行 `sp_babelfish_configure 'hatch_name'` 命令，其中 *`hatch_name`* 是一個或多個逃生艙的識別符。*`hatch_name`* 可以使用 SQL 萬用字元，例如 '%'。
- 若要將一個或多個逃生艙設定為指定的值，請執行 `sp_babelfish_configure ['hatch_name' [, 'strict'|'ignore' [, 'server']]]`。若要讓設定永久存在於叢集全面層級上，請加上 `server` 關鍵字，如下所示：

```
EXECUTE sp_babelfish_configure 'escape_hatch_unique_constraint', 'ignore', 'server'
```

若只要在目前工作階段中設定，請勿使用 `server`。

- 若要將所有逃生艙重設為其預設值，請執行 `sp_babelfish_configure 'default'` (Babelfish 1.2.0 及更高版本)。

一個逃生艙 (或多個逃生艙) 的識別字串可包含 SQL 萬用字元。例如，以下將 Aurora PostgreSQL 叢集的所有語法逃生艙設定為 `ignore`。

```
EXECUTE sp_babelfish_configure '%', 'ignore', 'server'
```

在下表中，您可以找到 Babelfish 預先定義逃生艙的說明和預設值。

逃生艙	描述	預設
`escape_hatch_checkpoint`	允許在程序碼中使用 CHECKPOINT 陳述式，但目前尚未實作 CHECKPOINT 陳述式。	ignore
`escape_hatch_constraint_name_for_default`	控制預設限制條件名稱相關的 Babelfish 行為。	ignore
`escape_hatch_database_misc_options`	在 CREATE DATABASE 上控制與下列選項相關的 Babelfish 行為：CONTAINMENT、DB_CHAINING、TRUSTWORTHY、PERSISTENT_LOG_BUFFER。	ignore
`escape_hatch_for_replication`	建立或更改資料表時，控制 [NOT] FOR REPLICATION 子句相關的 Babelfish 行為。	strict
`escape_hatch_fulltext`		

逃生艙	描述	預設
	控制 FULLTEXT 功能相關的 Babelfish 行為，例如 CREATE/ALTER DATABASE 中的 DEFAULT_FULLTEXT_LANGUAGE、CREATE FULLTEXT INDEX 或 sp_fulltext_database。	ignore
escape_hatch_ignore_dup_key	控制與 CREATE/ALTER TABLE 和 CREATE INDEX 相關的 Babelfish 行為。當 IGNORE_DUP_KEY=ON，如果設定為 strict (預設值) 將會發生錯誤，或者在設定為 ignore (Babelfish 1.2.0 及更高版本) 時忽略該錯誤。	strict
escape_hatch_index_clustering	控制索引及 PRIMARY KEY 或 UNIQUE 限制條件的 CLUSTERED 或 NONCLUSTERED 關鍵字相關的 Babelfish 行為。忽略 CLUSTERED 時，仍會視為已指定 NONCLUSTERED 來建立索引或限制條件。	ignore
escape_hatch_index_columnstore	控制 COLUMNSTORE 子句相關的 Babelfish 行為。如果指定 ignore，Babelfish 會建立正規 B 型樹狀結構索引。	strict
escape_hatch_join_hints	控制 JOIN 運算子中的關鍵字行為：LOOP、HASH、MERGE、REMOTE、REDUCE、REDISTRIBUTE、REPLICATE。	ignore

逃生艙	描述	預設
escape_hatch_language_non_english	控制英文以外的螢幕訊息語言相關的 Babelfish 行為。Abelfish 目前僅支援 us_english 的螢幕訊息。SET LANGUAGE 可能使用含有語言名稱的變數，因此只能在執行時偵測已設定的實際語言。	strict
escape_hatch_login_hashed_password	忽略時會抑制 CREATE LOGIN 和 ALTER LOGIN 的 HASHED 關鍵字的錯誤。	strict
escape_hatch_login_misc_options	忽略時，除了 CREATE LOGIN 和 ALTER LOGIN 的 HASHED、MUST_CHANGE 、OLD_PASSWORD 及 UNLOCK 之外，還會抑制其他關鍵字的錯誤。	strict
escape_hatch_login_old_password	忽略時會抑制 CREATE LOGIN 和 ALTER LOGIN 的 OLD_PASSWORD 關鍵字的錯誤。	strict
escape_hatch_login_password_must_change	忽略時會抑制 CREATE LOGIN 和 ALTER LOGIN 的 MUST_CHANGE 關鍵字的錯誤。	strict
escape_hatch_login_password_unlock	忽略時會抑制 CREATE LOGIN 和 ALTER LOGIN 的 UNLOCK 關鍵字的錯誤。	strict

逃生艙	描述	預設
escape_hatch_nocheck_add_constraint	控制限制條件的 WITH CHECK 或 NOCHECK 子句相關的 Babelfish 行為。	strict
escape_hatch_nocheck_existing_constraint	控制 FOREIGN KEY 或 CHECK 限制條件相關的 Babelfish 行為。	strict
escape_hatch_query_hints	控制查詢提示相關的 Babelfish 行為。此選項設定為 ignore 時，對於使用 OPTION (...) 子句指定查詢處理事宜的提示，伺服器會忽略提示。例子包括 SELECT FROM ... OPTION(MERGE JOIN HASH, MAXRECURSION 10))。	ignore
escape_hatch_rowversion	控制 ROWVERSION 和 TIMESTAMP 資料類型的行為。如需使用方式的資訊，請參閱 使用具有限制實作的 Babelfish 功能。	strict
escape_hatch_schemabinding_function	控制 WITH SCHEMABINDING 子句相關的 Babelfish 行為。預設會忽略 CREATE 或 ALTER FUNCTION 命令中指定的 WITH SCHEMABINDING 子句。	ignore
escape_hatch_schemabinding_procedure	控制 WITH SCHEMABINDING 子句相關的 Babelfish 行為。預設會忽略 CREATE 或 ALTER PROCEDURE 命令中指定的 WITH SCHEMABINDING 子句。	ignore

逃生艙	描述	預設
escape_hatch_rowguidcol_column	建立或更改資料表時，控制 ROWGUIDCOL 子句相關的 Babelfish 行為。	strict
escape_hatch_schemabinding_trigger	控制 WITH SCHEMABINDING 子句相關的 Babelfish 行為。預設會忽略 CREATE 或 ALTER TRIGGER 命令中指定的 WITH SCHEMABINDING 子句。	ignore
escape_hatch_schemabinding_view	控制 WITH SCHEMABINDING 子句相關的 Babelfish 行為。預設會忽略 CREATE 或 ALTER VIEW 命令中指定的 WITH SCHEMABINDING 子句。	ignore
escape_hatch_session_settings	控制不支援的工作階段層級 SET 陳述式相關的 Babelfish 行為。	ignore
escape_hatch_showplan_all	控制與 SET SHOWPLAN_ALL 與 SET STATISTICS PROFILE 相關的 Babelfish 行為。當設定為 ignore 時，它們的行為類似於 SET BABELFISH_SHOWPLAN_ALL 和 SET BABELFISH_STATISTICS PROFILE；當設定為 strict 時，會無聲地將其忽略。	strict
escape_hatch_storage_on_partition	定義分割時控制 ON partition_scheme column 子句相關的 Babelfish 行為。Babelfish 目前未實作分割。	strict

逃生艙	描述	預設
escape_hatch_storage_options	CREATE、ALTER DATABASE、TABLE、INDEX 中使用的任何儲存選項上的逃生艙 這包括子句 (LOG) ON、TEXTIMGE_ON、FILESTREAM_ON，用於定義資料表、索引、限制條件及資料庫的儲存位置 (分割區、檔案群組)。此逃生艙設定套用至以上所有子句 (包括 ON [PRIMARY] 和 ON "DEFAULT")。但以 ON partition_scheme (column) 指定資料表或索引的分割區時例外。	ignore
escape_hatch_table_hints	控制使用 WITH (...) 子句指定的資料表提示的行為。	ignore
escape_hatch_unique_constraint	設定為嚴格時，SQL Server 和 PostgreSQL 在處理索引欄上的 NULL 值方面的模糊語義差異可能會引發錯誤。語義差異只會出現於不切實際的使用案例中，因此您可將此轉義剖面設定為「忽略」以避免看到錯誤。 已從下列版本移除：3.6.0 及更新版本、4.2.0 及更新版本	strict

建立 Babelfish for Aurora PostgreSQL DB 叢集

Aurora PostgreSQL 13.4 版及更新版本支援 Babelfish for Aurora PostgreSQL。

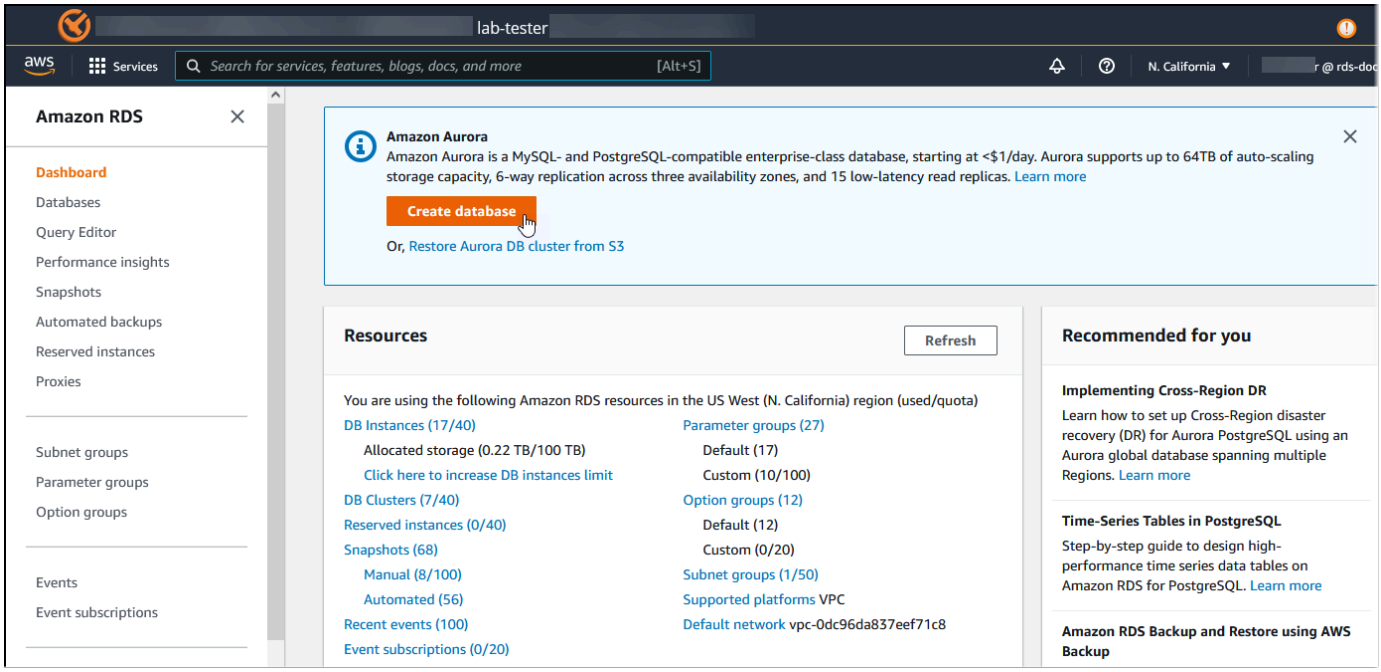
您可以使用 AWS 管理主控台 或 AWS CLI 來建立具有 Babelfish 的 Aurora PostgreSQL 叢集。

> **Note**
>
> 在 Aurora PostgreSQL 叢集中，為 Babelfish 保留了 `babelfish_db` 資料庫名稱。若自行在 Babelfish for Aurora PostgreSQLh 上建立 "babelfish_db" 資料庫，可能會導致 Aurora 無法成功佈建 Babelfish。

主控台

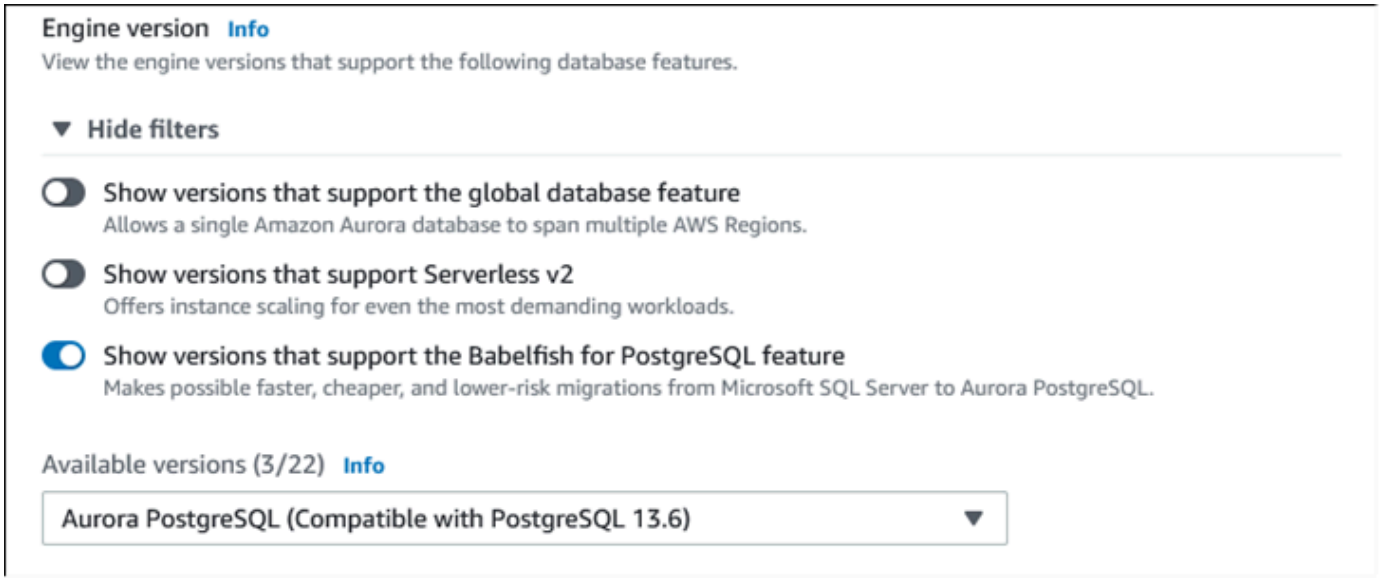
使用執行 的 Babelfish 建立叢集 AWS 管理主控台

1. 開啟 Amazon RDS 主控台 (https://console.aws.amazon.com/rds/)，然後選擇 Create database (建立資料庫)。



2. 針對 Choose a database creation method (選擇資料庫建立方法)，請執行下列其中一個動作：
   - 若要指定詳細的引擎選項，請選擇 Standard create (標準建立)。
   - 若要使用預先設定的選項來支援 Aurora 叢集的最佳實務，請選擇 Easy create (輕鬆建立)。
3. 針對引擎類型，請選擇 Aurora (PostgreSQL 相容)。

4. 選擇 Show filters (顯示篩選條件)，然後選擇 Show versions that support the Babelfish for PostgreSQL feature (顯示支援 Babelfish for PostgreSQL 功能的版本)，以列出支援 Babelfish 的引擎類型。目前 Aurora PostgreSQL 13.4 版及更新版本支援 Babelfish。
5. 針對 Available versions (可用的版本)，選擇一個 Aurora PostgreSQL 版本。若要取得最新的 Babelfish 功能，請選擇最新的 Aurora PostgreSQL 主要版本。

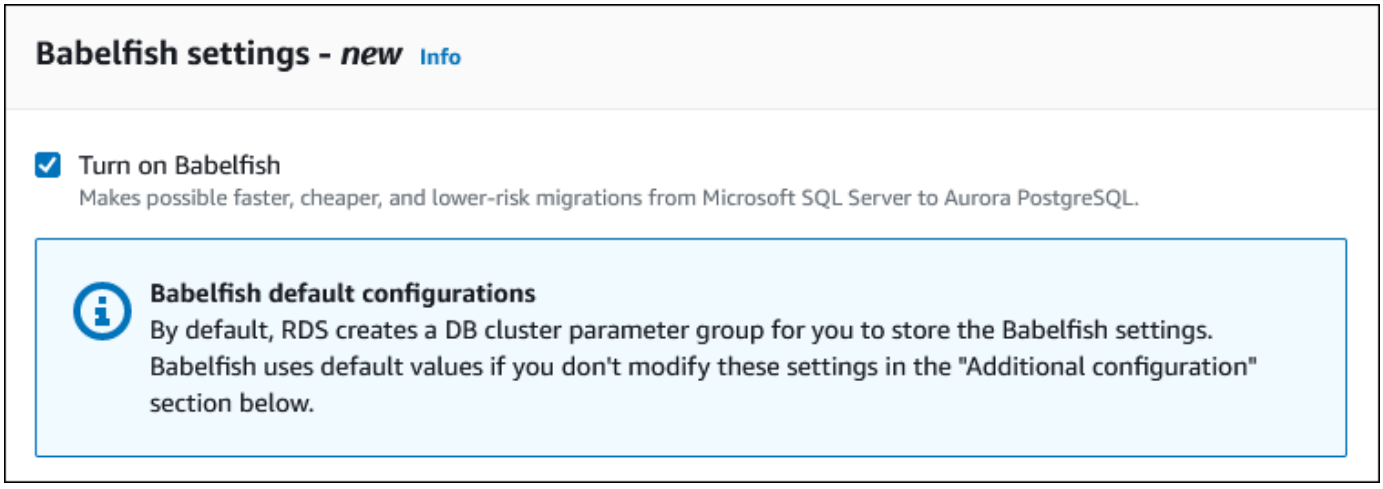


6. 針對 Templates (範本)，選擇您的使用案例適合的範本。
7. 針對 DB cluster identifier (資料庫叢集識別符)，輸入稍後可以在資料庫叢集清單中輕鬆找到的名稱。
8. 針對 Master username (主要使用者名稱)，輸入管理員使用者名稱。Aurora PostgreSQL 的預設值為 `postgres`。您可以接受預設名稱，或選擇不同的名稱。例如，若要遵循 SQL Server 資料庫上使用的命名慣例，您可以輸入 `sa` (系統管理員) 作為主要使用者名稱。

   如果現在不建立名為 `sa` 的使用者，稍後可以在您選擇的用戶端建立。建立使用者後，使用 `ALTER SERVER ROLE` 命令將其新增至叢集的 `sysadmin` 群組 (角色)。

   > **Warning**
   >
   > 主要使用者名稱必須始終使用小寫字元失敗，資料庫叢集無法透過 TDS 連接埠連接到 Babelfish。

9. 針對 Master password (主要密碼)，建立一個強式密碼並確認密碼。
10. 對於下來一直到 Babelfish settings (Babelfish 設定) 區段為止的選項，指定資料庫叢集設定。如需每項設定的相關資訊，請參閱 Aurora 資料庫叢集的設定。
11. 若要啟用 Babelfish 功能，請選取 Turn on Babelfish (開啟 Babelfish) 方塊。



12. 針對 DB cluster parameter group (資料庫叢集參數群組)，請執行下列其中一個動作：

    - 選擇 Create new (新建)，以建立啟用 Babelfish 的新參數群組。
    - 選擇 Choose existing (選擇現有)，以使用現有的參數群組。如果您使用現有群組，請務必在建立叢集之前修改群組，並新增 Babelfish 參數的值。如需 Babelfish 參數的資訊，請參閱Babelfish 的資料庫叢集參數群組設定。

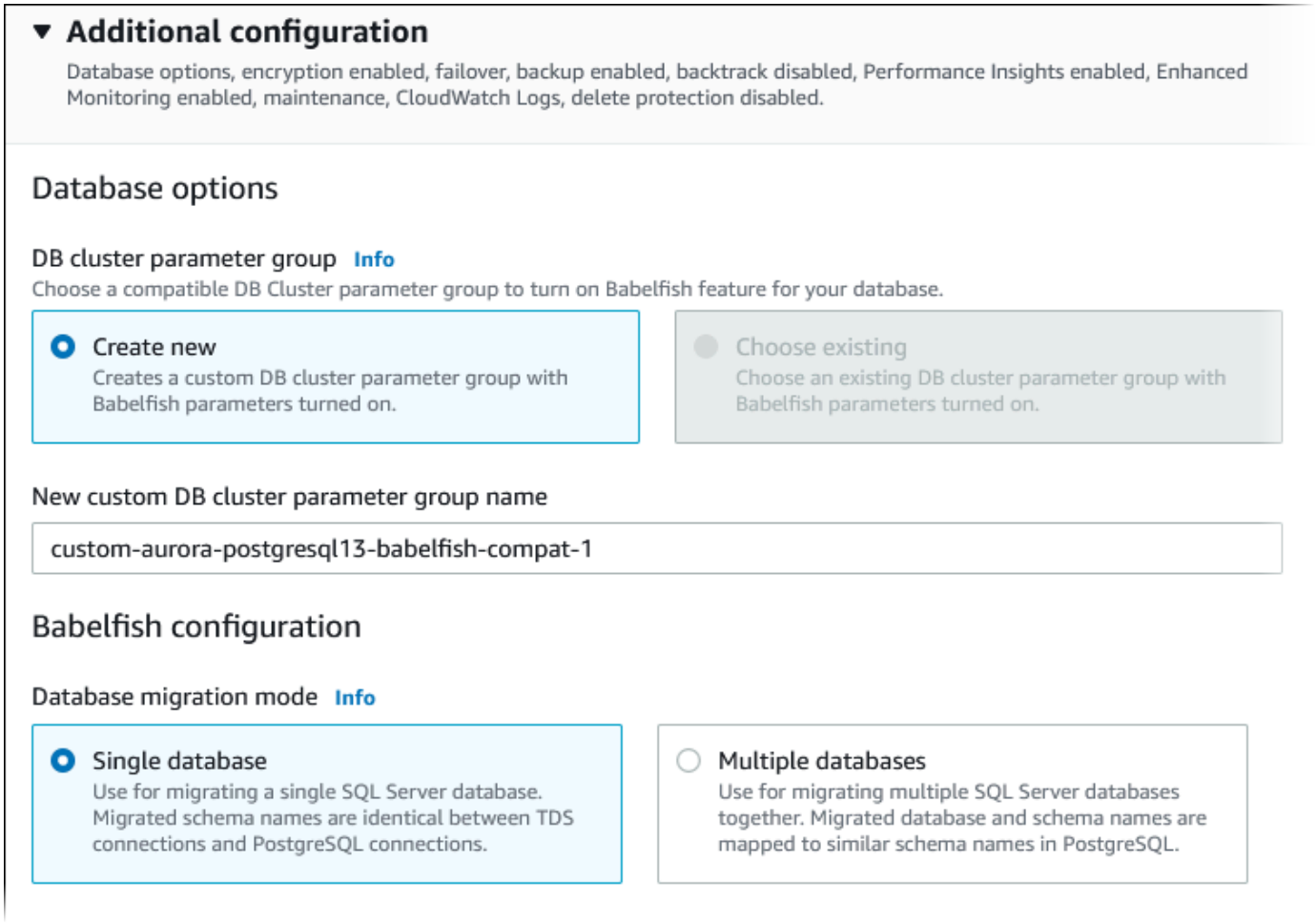
      如果您使用現有群組，請在後面的方塊中提供群組名稱。

13. 針對 Database migration modee (資料庫遷移模式)，選擇下列其中一項：

    - Single database (單一資料庫)，以遷移單一 SQL Server 資料庫。

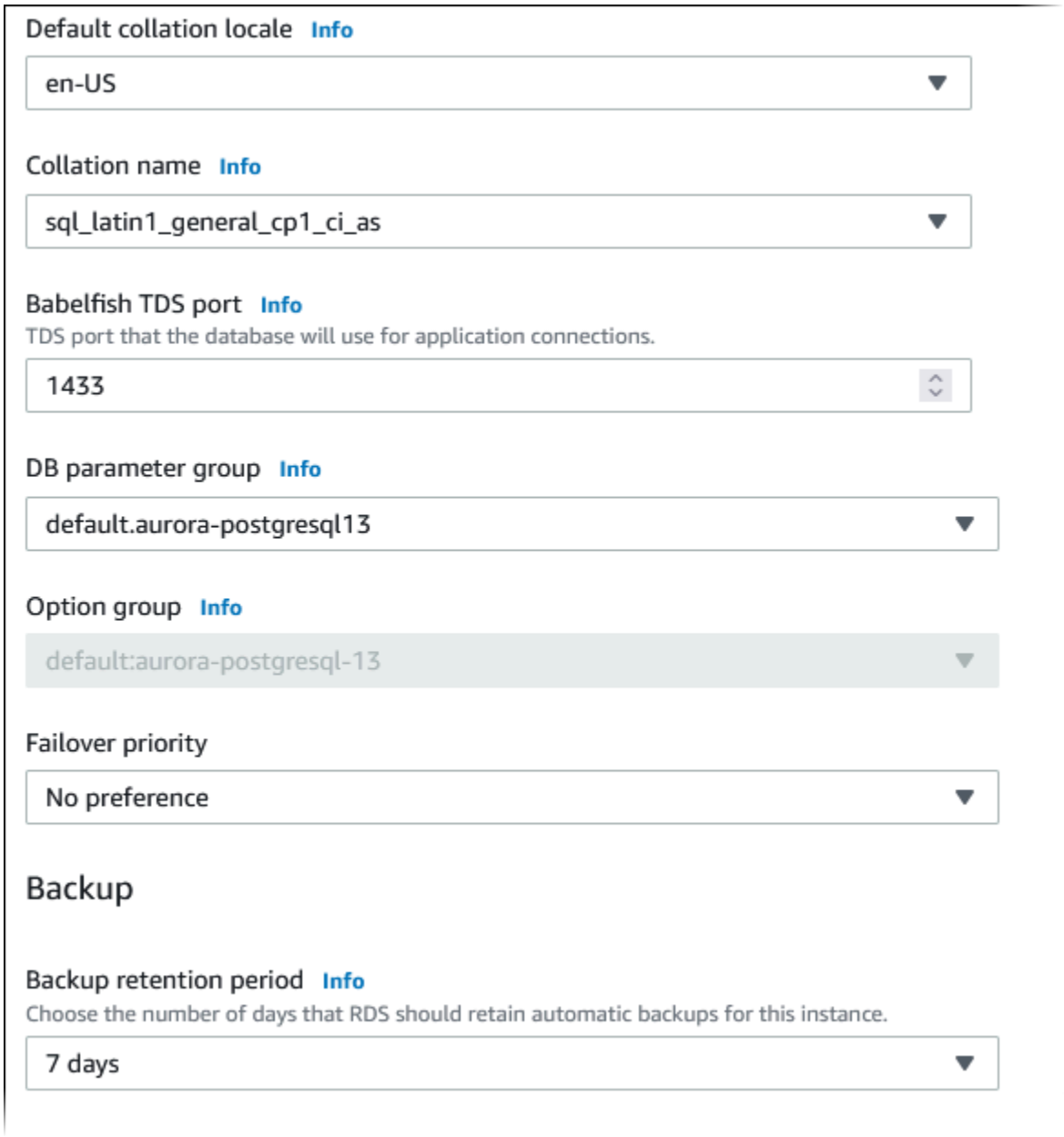
      在某些情況下，多個使用者資料庫可能一起遷移，而您的最終目標是完整遷移至沒有 Babelfish 的原生 Aurora PostgreSQL。如果最終的應用程式需要合併的結構描述 (單一dbo 結構描述)，請務必先將 SQL Server 資料庫合併為單一 SQL Server 資料庫。然後使用 Single database (單一資料庫) 模式來遷移至 Babelfish。
    - Multiple databases (多個資料庫)，以遷移多個 SQL Server 資料庫 (來自單一 SQL Server 安裝)。多資料庫模式不合併不是來自單一 SQL Server 安裝的多個資料庫。如需有關遷移多個資料庫的資訊，請參閱搭配單一資料庫或多個資料庫來使用 Babelfish。

> **Note**
>
> 從 Aurora PostgreSQL 第 16 版，預設會選擇多個資料庫作為資料庫遷移模式。



14. 針對 Default collation locale (預設定序地區設定)，輸入伺服器地區設定。預設值為 `en-US`。如需定序的詳細資訊，請參閱 了解 Babelfish for Aurora PostgreSQL 中的定序。
15. 針對 Collation name (定序名稱) 欄位，輸入預設定序。預設值為 `sql_latin1_general_cp1_ci_as`。如需詳細資訊，請參閱 了解 Babelfish for Aurora PostgreSQL 中的定序。
16. 針對 Babelfish TDS 連接埠，請輸入預設連接埠 1433。目前，Babelfish 僅支援您資料庫叢集的連接埠 1433。
17. 針對 DB parameter group (資料庫參數群組)，選擇參數群組，或讓 Aurora 使用預設設定為您建立新的群組。
18. 針對 Failover priority (容錯移轉優先順序)，選擇執行個體的容錯移轉優先順序。如果不選擇值，則預設為 `tier-1`。此優先順序決定從主要執行個體失敗中復原時提升複本所按照的順序。如需詳細資訊，請參閱Aurora 資料庫叢集的容錯能力。

19. 針對 Backup retention period (備份保留期),選擇 Aurora 保留資料庫備份副本的時間長度 (1 到 35 天)。您可以使用備份副本對資料庫進行時間點復原 (PITR)，可精確至秒。預設保留期間為七天。



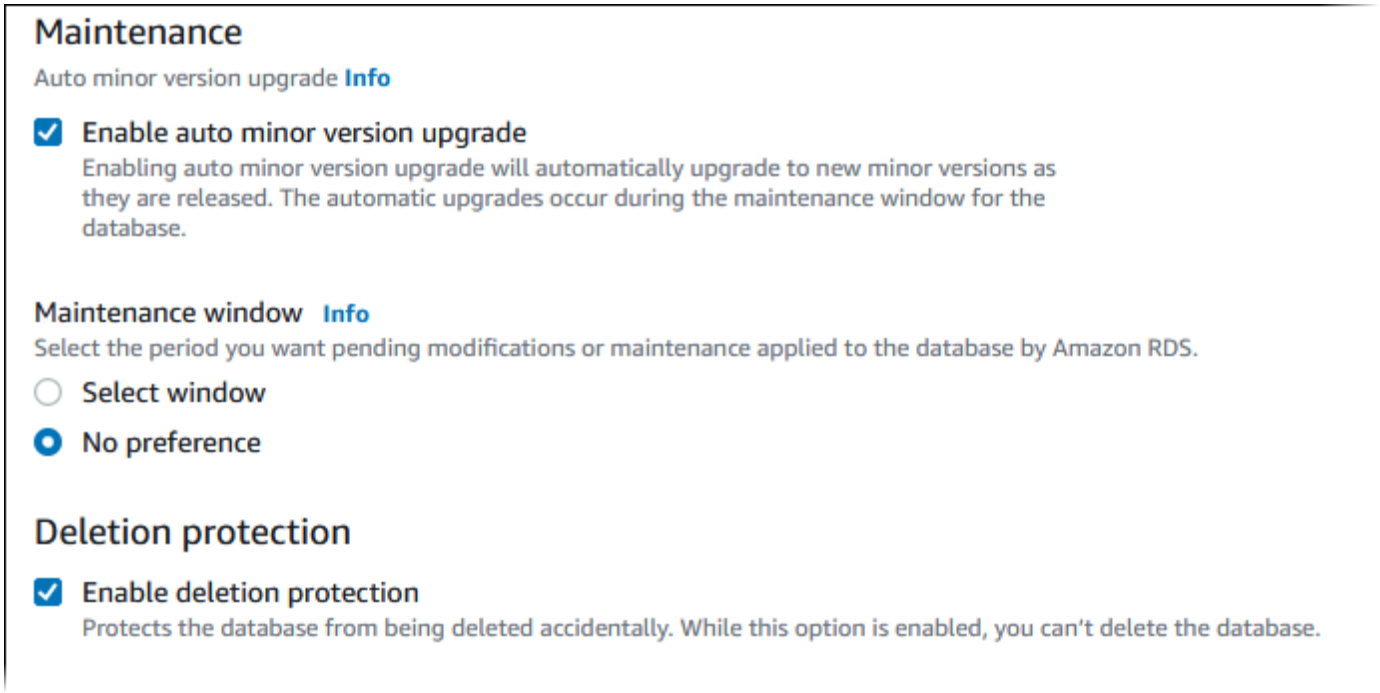
20. 選擇 Copy tags to snapshots (將標籤複製到快照)，在建立快照時將任何資料庫執行個體標籤複製到資料庫快照。

> **Note**
>
> 從快照還原資料庫叢集時，其不會還原為 Babelfish for Aurora PostgreSQL 資料庫叢集。您需要開啟資料庫叢集參數群組中控制 Babelfish 偏好設定的參數，才能再次啟用

Babelfish。如需 Babelfish 參數的詳細資訊，請參閱Babelfish 的資料庫叢集參數群組設定。

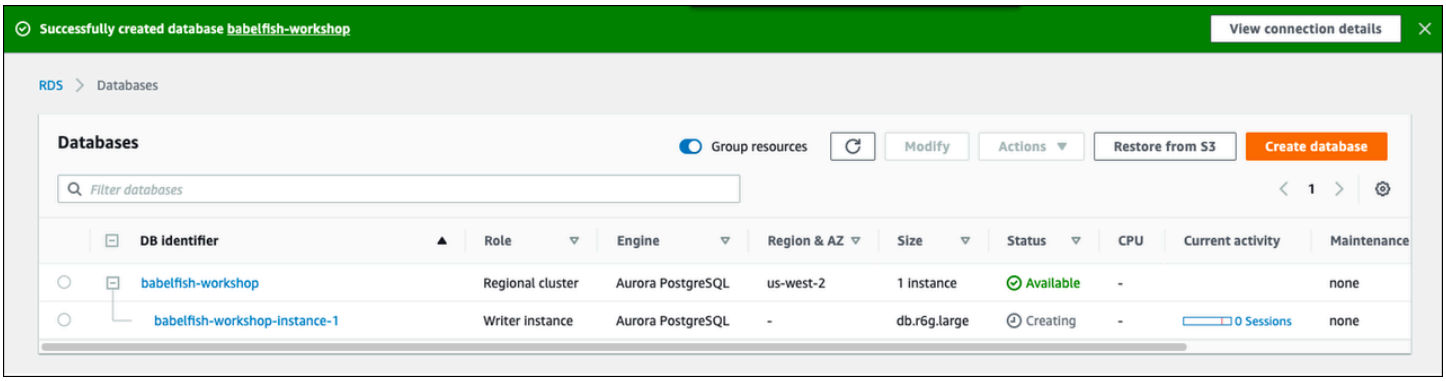
21. 選擇 Enable encryption (啟用加密)，對此資料庫叢集啟用靜態加密 (Aurora 儲存加密)。
22. 選擇 Enable Performance Insights (啟用績效詳情)，以啟用 Amazon RDS 績效詳情。
23. 選擇 Enable Enhanced monitoring (啟用增強型監控)，以針對執行資料庫叢集的作業系統，開始即時收集指標。
24. 選擇 PostgreSQL log (PostgreSQL 日誌)，將日誌檔發佈到 Amazon CloudWatch Logs。
25. 選擇 Enable auto minor version upgrade (啟用自動次要版本升級)，在有次要版本升級可用時自動更新 Aurora 資料庫叢集。
26. 針對 Maintenance window (維護時段)，請執行下列動作：
    - 若要選擇 Amazon RDS 進行修改或執行維護的時間，請選擇 Select window (選取時段)。
    - 若要在非排定時間執行 Amazon RDS 維護，請選擇 No preference (無偏好設定)。
27. 選取 Enable deletion protection (啟用刪除保護) 方塊，以防止意外刪除您的資料庫。

    如果啟用此功能，則無法直接刪除資料庫。相反地，在刪除資料庫之後，您需要修改資料庫叢集並停用此功能。



28. 選擇建立資料庫。

你可以在 Databases (資料庫) 清單中看到為 Babelfish 建立的新資料庫。部署完成時，Status (狀態) 欄會顯示 Available (可用)。



AWS CLI

當您使用 建立 Babelfish for Aurora PostgreSQL 時 AWS CLI，您需要傳遞命令資料庫叢集參數群組的名稱，以用於叢集。如需詳細資訊，請參閱資料庫叢集先決條件。

使用 AWS CLI 建立具有 Babelfish 的 Aurora PostgreSQL 叢集之前，請執行下列動作：

- 從 Amazon Aurora 端點和配額上的服務清單中，選擇您的端點 URL。
- 建立叢集的參數群組。如需參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。
- 修改參數群組，新增參數來啟用 Babelfish。

使用 建立具有 Babelfish 的 Aurora PostgreSQL 資料庫叢集 AWS CLI

下列範例使用預設主要使用者名稱，`postgres`。根據需要取代成以您為資料庫叢集建立的使用者名稱，例如 `sa`，或者如果您不接受預設值，則取代成任何您選擇的使用者名稱。

1. 建立參數群組。

   對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster-parameter-group \
--endpoint-url endpoint-url \
--db-cluster-parameter-group-name parameter-group \
--db-parameter-group-family aurora-postgresql14 \
--description "description"
```

   在 Windows 中：

```
aws rds create-db-cluster-parameter-group ^
--endpoint-url endpoint-URL ^
--db-cluster-parameter-group-name parameter-group ^
--db-parameter-group-family aurora-postgresql14 ^
--description "description"
```

2. 修改參數群組來啟用 Babelfish。

   對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
--endpoint-url endpoint-url \
--db-cluster-parameter-group-name parameter-group \
--parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
```

   在 Windows 中：

```
aws rds modify-db-cluster-parameter-group ^
--endpoint-url endpoint-url ^
--db-cluster-parameter-group-name paramater-group ^
--parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
```

3. 識別新資料庫叢集的資料庫子網路群組和 Virtual Private Cloud (VPC) 安全群組 ID，然後呼叫 create-db-cluster 命令。

   對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
--db-cluster-identifier  cluster-name\
--master-username postgres \
--manage-master-user-password \
--engine aurora-postgresql \
--engine-version 14.3            \
--vpc-security-group-ids security-group \
--db-subnet-group-name subnet-group-name \
--db-cluster-parameter-group-name parameter-group
```

   在 Windows 中：

```
aws rds create-db-cluster ^
--db-cluster-identifier cluster-name ^
--master-username postgres ^
--manage-master-user-password ^
--engine aurora-postgresql ^
--engine-version 14.3 ^
--vpc-security-group-ids security-group ^
--db-subnet-group-name subnet-group ^
--db-cluster-parameter-group-name parameter-group
```

此範例會指定 `--manage-master-user-password` 選項來產生主要使用者密碼，並在 Secrets Manager 中管理該密碼。如需詳細資訊，請參閱使用 Amazon Aurora 和 進行密碼管理 AWS Secrets Manager。或者，您可以使用 `--master-password` 選項，自行指定和管理密碼。

4. 明確為資料庫叢集建立主要執行個體。當您呼叫 create-db-instance 命令時，使用您在步驟 3 中為 `--db-cluster-identifier` 引數建立的叢集名稱，如下所示。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
--db-instance-identifier instance-name \
--db-instance-class db.r6g \
--db-subnet-group-name subnet-group \
--db-cluster-identifier cluster-name \
--engine aurora-postgresql
```

在 Windows 中：

```
aws rds create-db-instance ^
--db-instance-identifier instance-name ^
--db-instance-class db.r6g ^
--db-subnet-group-name subnet-group ^
--db-cluster-identifier cluster-name ^
--engine aurora-postgresql
```

將 SQL Server 資料庫遷移至 Babelfish for Aurora PostgreSQL

您可以使用 Babelfish for Aurora PostgreSQL 將 SQL Server 資料庫遷移至 Amazon Aurora PostgreSQL 資料庫叢集。遷移之前，請檢閱搭配單一資料庫或多個資料庫來使用 Babelfish。

主題

- 遷移程序概觀
- 評估和處理 SQL Server 與 Babelfish 之間的差異
- 用於從 SQL Server 遷移至 Babelfish 的匯入/匯出工具

遷移程序概觀

下列摘要列出成功遷移 SQL Server 應用程式並使其適用於 Babelfish 所需的步驟。如需與可用於匯出與匯入程序的工具相關的資訊，或其他詳細資料，請參閱 用於從 SQL Server 遷移至 Babelfish 的匯入/匯出工具。若要載入資料，建議您使用 AWS DMS 搭配 Aurora PostgreSQL 資料庫叢集做為目標端點。

1. 建立一個 Babelfish 開啟的新 Aurora PostgreSQL 資料庫叢集。若要了解作法，請參閱建立 Babelfish for Aurora PostgreSQL DB 叢集。

   若要匯入從 SQL Server 資料庫匯出的各種 SQL 成品，請使用 SQL Server 工具 (例如 sqlcmd) 來連線至 Babelfish 叢集。如需詳細資訊，請參閱使用 SQL Server 用戶端來連線至資料庫叢集。
2. 在您要遷移的 SQL Server 資料庫上，匯出資料定義語言 (DDL)。DDL 是描述資料庫物件的 SQL 程式碼，這些物件包含使用者資料 (例如資料表、索引和檢視表) 和使用者撰寫的資料庫程式碼 (例如預存程序、使用者定義函數和觸發程序)。

   如需詳細資訊，請參閱使用 SQL Server Management Studio (SSMS) 遷移至 Babelfish。
3. 執行評估工具來評估您可能需要進行任何變更的範圍，讓 Babelfish 能夠有效地支援在 SQL Server 上執行的應用程式。如需詳細資訊，請參閱評估和處理 SQL Server 與 Babelfish 之間的差異。
4. 檢閱 AWS DMS 目標端點限制，並視需要更新 DDL 指令碼。如需詳細資訊，請參閱使用 Babelfish for Aurora PostgreSQL 作為目標中將 PostgreSQL 目標端點與 Babelfish 資料表搭配使用的限制。
5. 在新的 Babelfish 資料庫叢集上，於您指定的 T-SQL 內執行 DDL，以僅建立結構描述、使用者定義的資料類型，以及具有主索引鍵條件約束的資料表。
6. 使用 AWS DMS 將資料從 SQL Server 遷移至 Babelfish 資料表。對於使用 SQL Server 變更資料擷取或 SQL 複寫的連續複寫，請使用 Aurora PostgreSQL 而不是 Babelfish 作為端點。若要這樣做，請參閱使用 Babelfish for Aurora PostgreSQL 作為 AWS Database Migration Service的目標。

7. 資料載入完成時，建立所有支援 Babelfish 叢集上應用程式的剩餘 T-SQL 物件。
8. 將用戶端應用程式重新設定成連線至 Babelfish 端點，而不是 SQL Server 資料庫。如需詳細資訊，請參閱連線至 Babelfish 資料庫叢集。
9. 視需要修改應用程式並重新測試。如需詳細資訊，請參閱Babelfish for Aurora PostgreSQL 與 SQL Server 之間的差異。

您仍然需要評估您的用戶端 SQL 查詢。從 SQL Server 執行個體產生的結構描述僅轉換伺服器端 SQL 程式代碼。建議您採取下列步驟：

- 透過將 SQL Server Profiler 與 TSQL_Replay 預先定義的範本搭配使用，來擷取用戶端查詢。此範本會擷取 T-SQL 陳述式資訊，之後您可以重播這些資訊以進行反覆調校與測試。您可以從 Tools (工具) 功能表啟動 SQL Server Management Studio 內的分析工具。選擇 SQL Server Profiler 開啟分析器，然後選擇 TSQL_Replay 範本。

  若要用於 Babelfish 遷移，請啟動追蹤，然後使用功能測試執行應用程式。分析器會擷取 T-SQL 陳述式。完成測試後，請停止追蹤。將結果連同您的用戶端查詢儲存至 XML 檔案 (File (檔案) > Save as (另存新檔) > Trace XML File for Replay (追蹤用於重播的 XML 檔案))。

  如需詳細資訊，請參閱 Microsoft 文件中的 SQL Server Profiler。如需 TSQL_Replay 範本的詳細資訊，請參閱 SQL Server Profiler 範本。
- 如果應用程式執行複雜的用戶端 SQL 查詢，建議您使用 Babelfish Compass 來分析這些查詢是否相容於 Babelfish。如果分析指出用戶端 SQL 陳述式包含不支援的 SQL 功能，請檢閱用戶端應用程式的 SQL 部分，並視需要修改。
- 您也可以擷取 SQL 查詢作為延伸事件 (.xel 格式)。若要這樣做，請使用 SSMS XEvent Profiler。在產生 .xel 檔案之後，請將 SQL 陳述式擷取到 Compass 隨後可以處理的 .xml 檔案。如需詳細資訊，請參閱 Microsoft 文件中的使用 SSMS XEvent Profler。

如果您對所遷移應用程式需要的所有測試、分析和任何修改都滿意，便可開始將 Babelfish 資料庫用於生產環境。若要這麼做，請停止原始資料庫，並將即時用戶端應用程式轉向使用 Babelfish TDS 連接埠。

> **Note**
>
> AWS DMS 現在支援從 Babelfish 複寫資料。如需詳細資訊，請參閱 AWS DMS 現支援 Babelfish for Aurora PostgreSQL 作為來源。

評估和處理 SQL Server 與 Babelfish 之間的差異

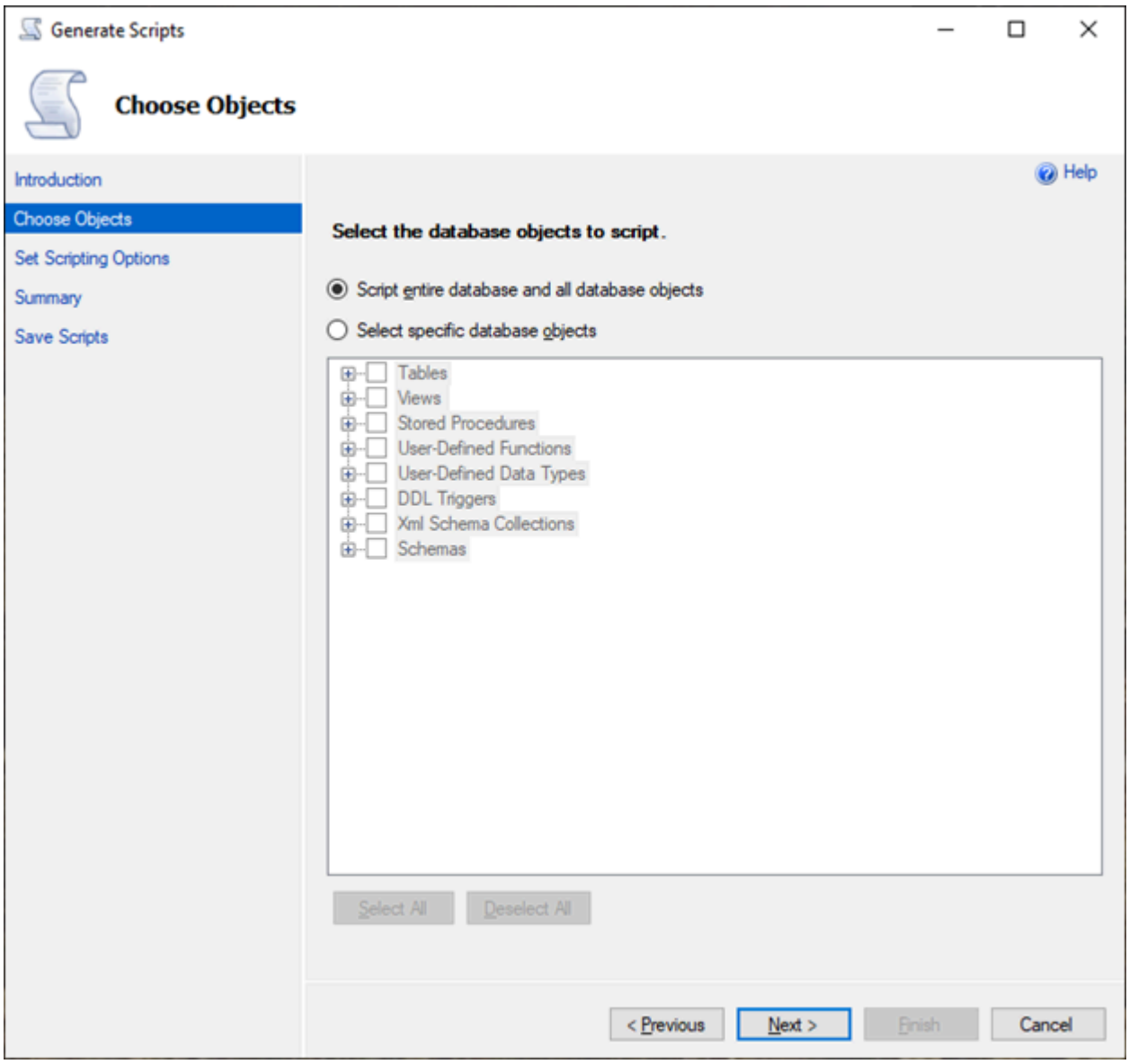
為了獲得最佳結果，建議您先評估產生的 DDL/DML 和用戶端查詢程式碼，再真正地將 SQL Server 資料庫應用程式遷移到 Babelfish。根據 Babelfish 的版本及應用程式實作的 SQL Server 特有功能，您可能需要重構您的應用程式，或使用 Babelfish 中尚未完全支援功能的替代功能。

- 若要評估 SQL Server 應用程式碼，請在產生的 DDL 上使用 Babelfish Compass，來判斷 Babelfish 對 T-SQL 程式碼支援的程度。在 Babelfish 上執行之前，請找出可能需要修改的 T-SQL 程式碼。如需此工具的詳細資訊，請參閱 GitHub 上的 Babelfish Compass 工具。

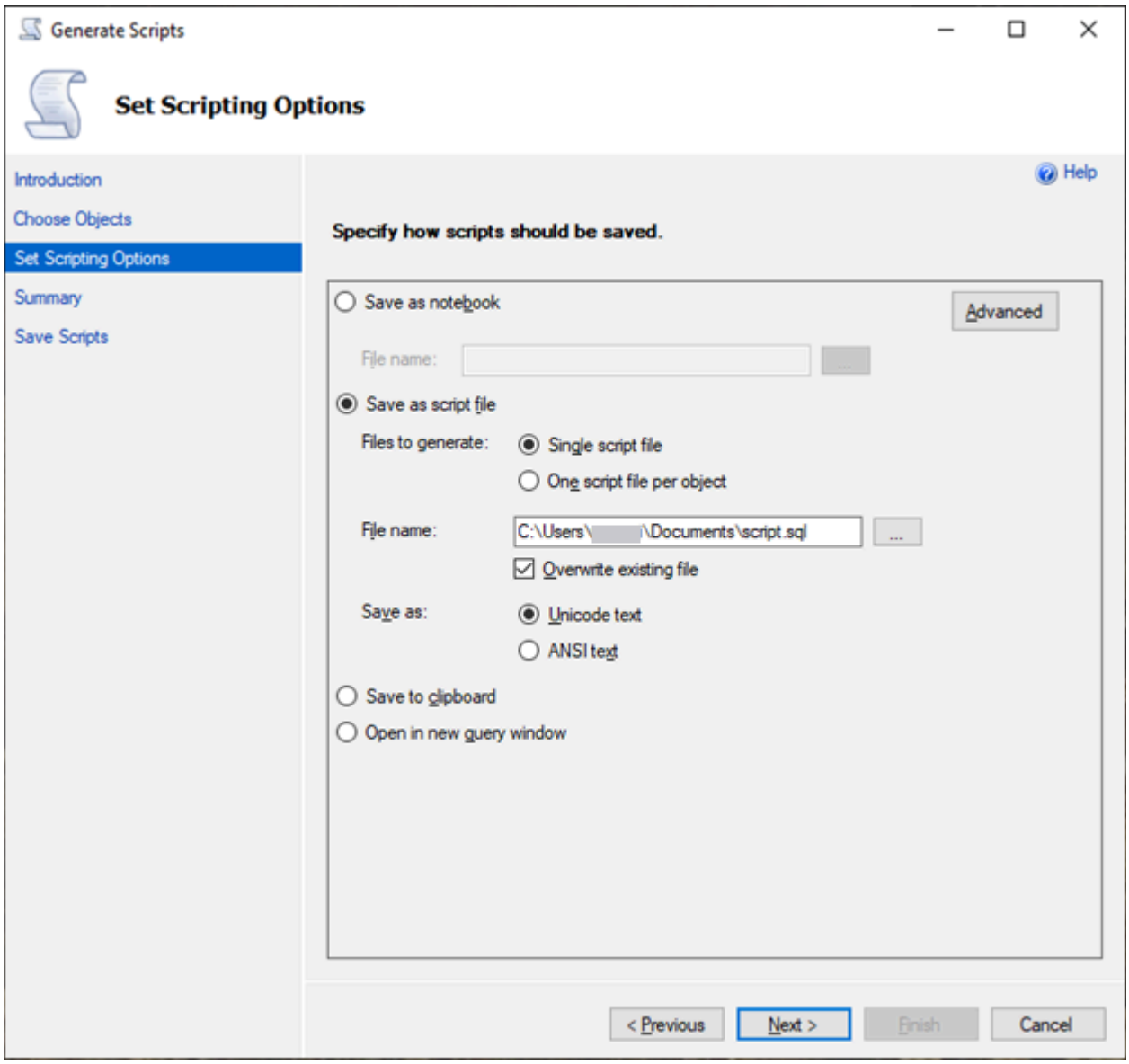
> **Note**
>
> Babelfish Compass 是一種開放原始碼工具。請透過 GitHub 回報 Babelfish Compass 的任何問題，而不是透過 AWS Support。

您可以使用產生指令碼精靈搭配 SQL Server Management Studio (SSMS)，來產生由 Babelfish Compass 或 AWS Schema Conversion Tool CLI 評估的 SQL 檔案。我們建議採取下列步驟來簡化評估。

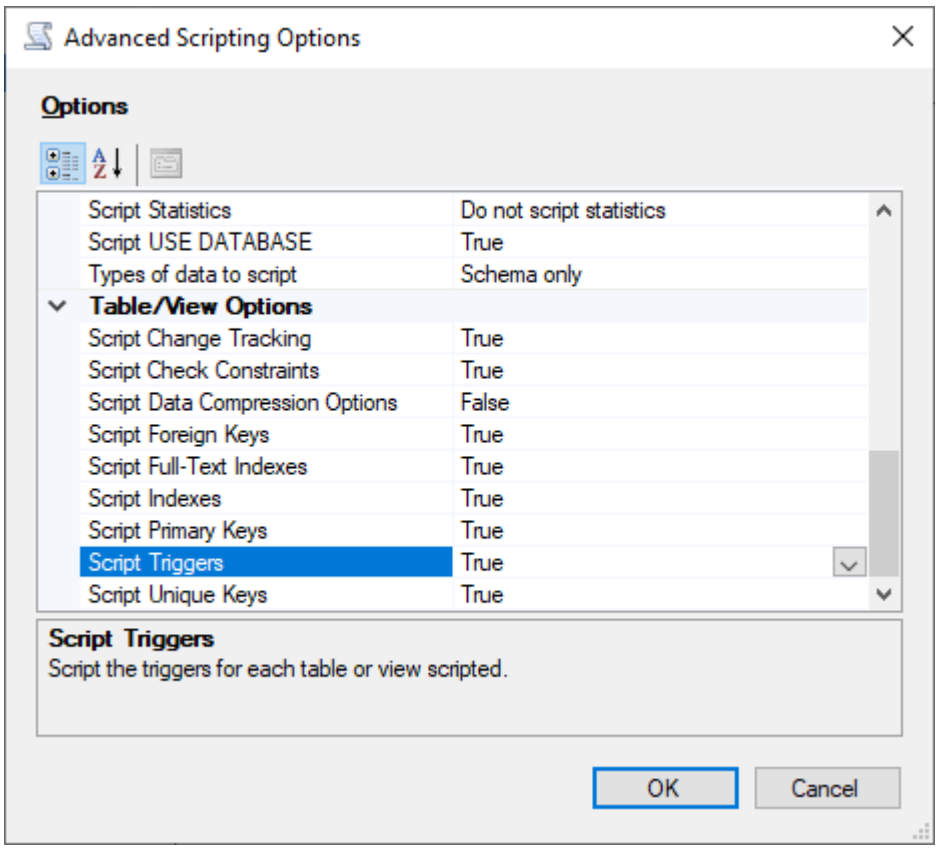
1. 在 Choose Objects (選擇物件) 頁面上，選擇 Script entire database and all database objects (編寫整個資料庫和所有資料庫物件的指令碼)。



2. 針對 Set Scripting Options (設定指令碼選項)，選擇 Save as script file (儲存為指令碼檔案) 作為 Single script file (單一指令碼檔案)。



3. 選擇 Advanced (進階) 來變更預設指令碼選項，以識別通常設定為 false 進行完整評估的功能：
   - Script Change Tracking (指令碼變更追蹤) 設為 True
   - Script Full-Text Indexes (指令碼全文檢索) 設為 True
   - Script Triggers (指令碼觸發條件) 設為 True
   - Script Logins (指令碼登入) 設為 True
   - Script Owner (指令碼擁有者) 設為 True
   - Script Object-Level Permissions (指令碼物件層級許可) 設為 True
   - Script Collations (指令碼定序) 設為 True



4. 執行精靈中的其餘步驟來產生檔案。

用於從 SQL Server 遷移至 Babelfish 的匯入/匯出工具

我們建議您使用 AWS DMS 做為從 SQL Server 遷移至 Babelfish 的主要工具。不過，Babelfish 支援數種其他方式，使用 SQL Server 工具遷移資料，其中包括下列項目。

- SQL Server Integration Services (SSIS)，適用於 Babelfish 的所有版本。如需詳細資訊，請參閱使用 SSIS 和 Babelfish 從 SQL Server 遷移至 Aurora PostgreSQL。
- 針對 Babelfish 2.1.0 及更新版本使用 SSMS 匯入/匯出精靈。此工具可透過 SSMS 取得，但也可作為獨立工具使用。如需詳細資訊，請參閱 Microsoft 文件中的歡迎使用 SQL Server 匯入和匯出精靈。
- Microsoft 大量資料複製程式 (bcp) 公用程式可讓您將 Microsoft SQL Server 執行個體中的資料以您指定的格式複製到資料檔案。如需詳細資訊，請參閱 Microsoft 文件中的 bcp 公用程式。Babelfish 現在支援使用 BCP 用戶端進行資料遷移，而 bcp 公用程式現在支援 `-E`旗標 (用於身分資料欄) 和 `-b` 旗標 (用於批次插入)。不支援某些 bcp 選項，包括 `-C`、`-T`、`-G`、`-K`、`-R`、`-V` 和 `-h`。

使用 SQL Server Management Studio (SSMS) 遷移至 Babelfish

建議針對每個特定物件類型產生個別的檔案。您可以針對每組 DDL 陳述式使用 SSMS 中的產生指令碼精靈，然後以群組的形式修改物件，來修正評估期間找到的任何問題。

執行這些步驟，以使用 AWS DMS 或其他資料遷移方法遷移資料。首先執行這些建立指令碼類型，來取得更好、更快的方法，以在 Aurora PostgreSQL 中的 Babelfish 資料表上載入資料。

1. 執行 CREATE SCHEMA 陳述式。
2. 執行 CREATE TYPE 陳述式來建立使用者定義的資料類型。
3. 執行具有主索引鍵或唯一條件約束的基本 CREATE TABLE 陳述式。

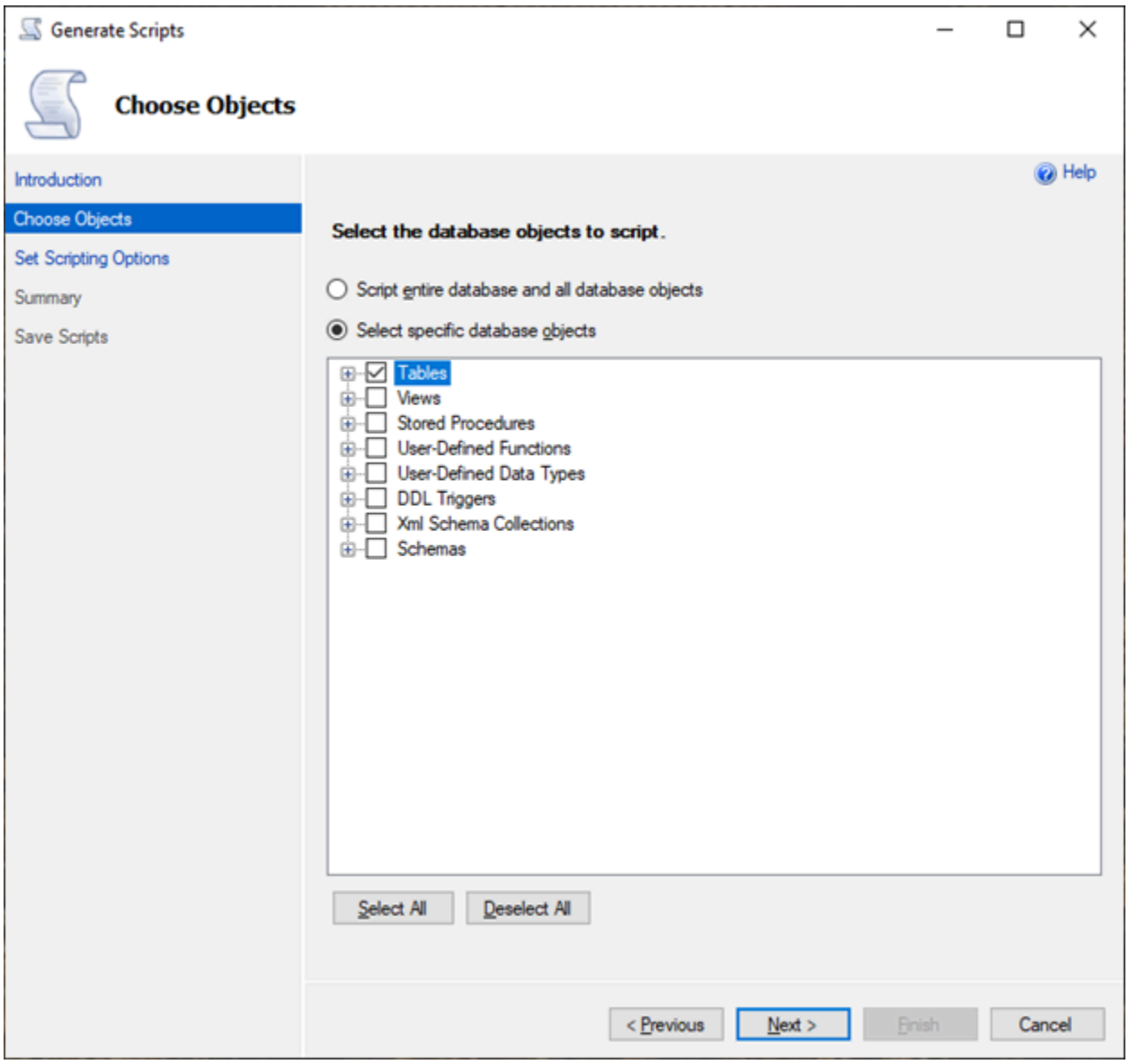
使用建議的匯入/匯出工具執行資料載入。針對下列步驟執行修改的指令碼，以新增剩餘的資料庫物件。您需要建立資料表陳述式來針對條件約束、觸發條件和索引執行這些指令碼。在指令碼產生之後，請刪除建立資料表陳述式。

1. 針對檢查條件約束、外部索引鍵條件約束、預設條件約束執行 ALTER TABLE 陳述式。
2. 執行 CREATE TRIGGER 陳述式。
3. 執行 CREATE INDEX 陳述式。
4. 執行 CREATE VIEW 陳述式。
5. 執行 CREATE STORED PROCEDURE 陳述式。

針對每個物件類型產生指令碼

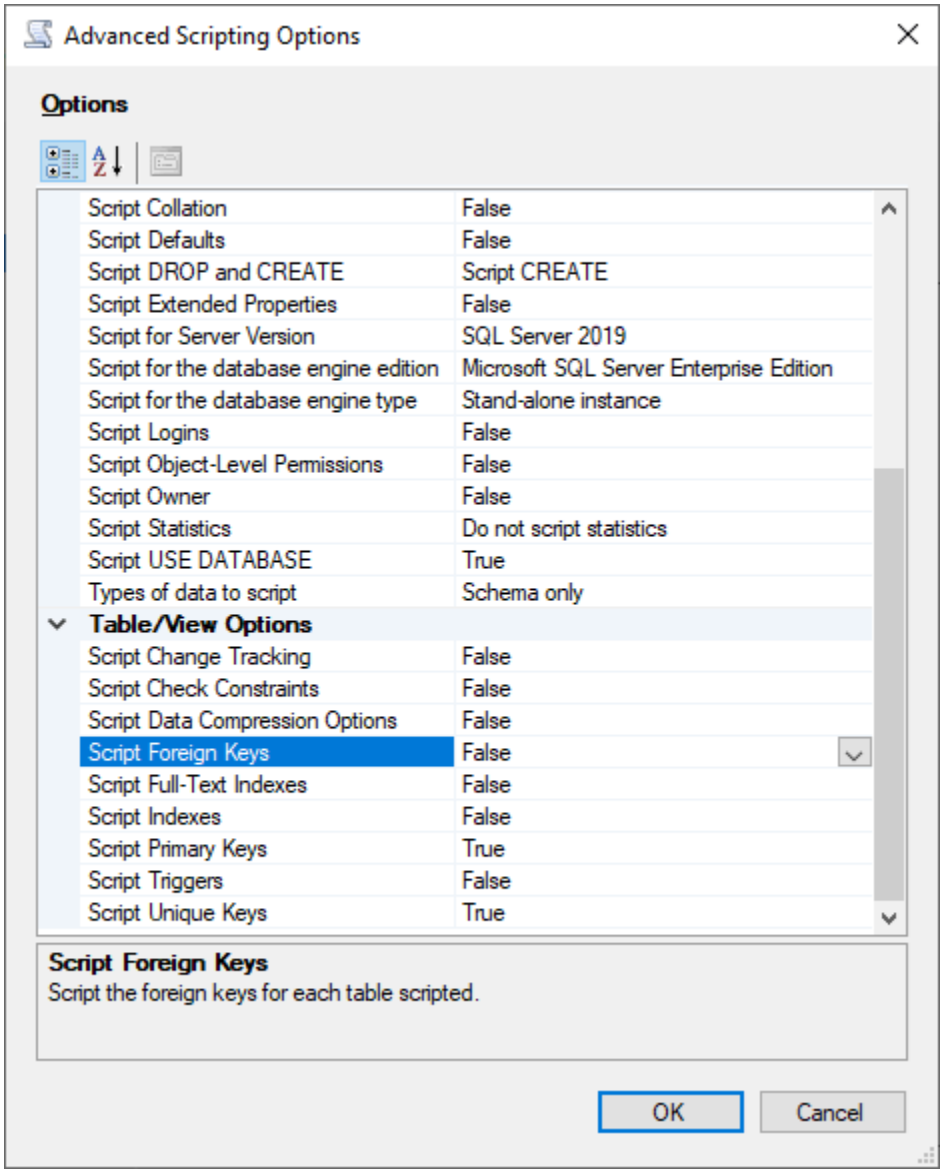
請使用下列步驟，使用 SSMS 中的產生程式碼精靈，來建立基本建立資料表陳述式。遵循相同步驟，針對不同的物件類型產生指令碼。

1. 連線至現有的 SQL Server 執行個體。
2. 在資料庫名稱上開啟內容選單 (按一下滑鼠右鍵)。
3. 選擇 Tasks (任務)，然後選擇 Generate Scripts... (產生指令碼...)。
4. 在 Choose Objects (選擇物件) 窗格中，選擇 Select specific database objects (選取特定資料庫物件)。選擇 Tables (資料表)，然後選取所有資料表。選擇 Next (下一步) 繼續。

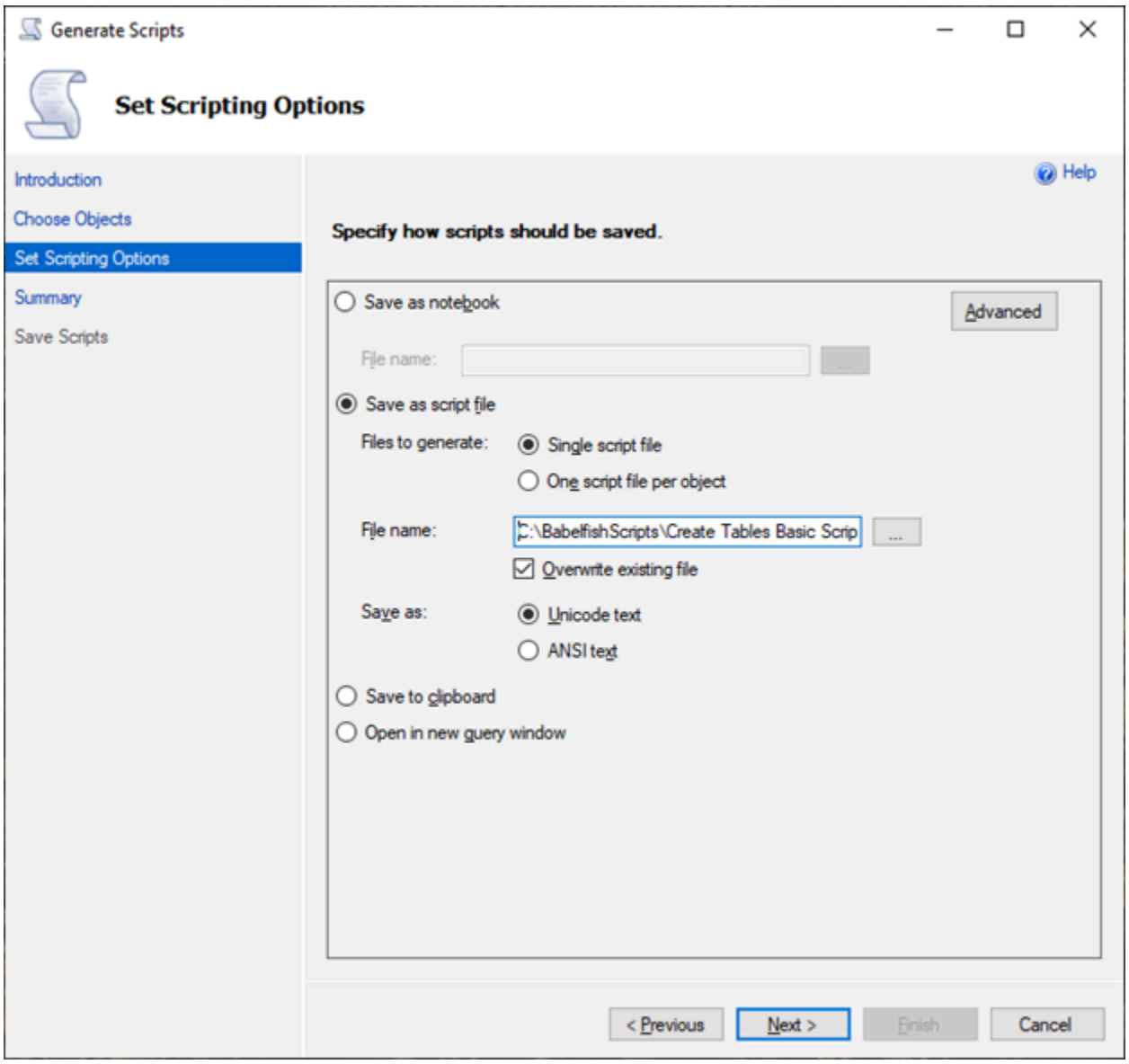


5. 在 Set Scripting Options (設定指令碼選項) 頁面上，選擇 Advanced (進階) 以開啟 Options (選項) 設定。若要產生基本建立資料表陳述式，請變更下列預設值：

   - Script Defaults (指令碼預設值) 變更為 False。
   - Script Extended Properties (指令碼延伸屬性) 變更為 False。Babelfish 不支援延伸屬性。
   - Script Check Constraints (指令碼檢查條件約束) 變更為 False。Script Foreign Keys (指令碼外部索引鍵) 變更為 False。



6. 選擇確定。
7. 在 Set Scripting Options (設定指令碼選項) 頁面上，選擇 Save as script file (儲存為指令碼檔案)，然後選擇 Single script file (單一指令碼檔案)。輸入您的 File name (檔案名稱)。



8. 選擇 Next (下一步) 以檢視 Summary wizard (摘要精靈) 頁面。
9. 選擇 Next (下一步) 以開始產生指令碼。

   您可以繼續針對精靈中的其他物件類型產生指令碼。請選擇三次 Previous (上一步」按鈕，以回到 Choose Objects (選擇物件) 頁面，而不是在儲存檔案之後選擇 Finish (完成)。然後重複精靈中的步驟，針對其他物件類型產生指令碼。

使用 Babelfish for Aurora PostgreSQL 進行資料庫身分驗證

Babelfish for Aurora PostgreSQL 支援兩種驗證資料庫使用者的方式。密碼身分驗證預設適用於所有 Babelfish 資料庫叢集。您也可以為相同的資料庫叢集新增 Kerberos 身分驗證。

主題

- 使用 Babelfish 進行密碼身分驗證
- 使用 Babelfish 進行 Kerberos 身分驗證
- 使用 Active Directory 安全群組為 Babelfish 設定 Kerberos 驗證

使用 Babelfish 進行密碼身分驗證

Babelfish for Aurora PostgreSQL 支援密碼身分驗證。密碼以加密形式存放在磁碟上。如需 Aurora PostgreSQL 叢集上進行身分驗證的詳細資訊，請參閱 Amazon Aurora PostgreSQL 的安全性。

每次連線至 Babelfish 時，可能會要求您輸入憑證。任何遷移至或建立於 Aurora PostgreSQL 的使用者，在 SQL Server 連接埠和 PostgreSQL 連接埠上都可以使用相同的憑證。Babelfish 不強制執行密碼政策，但建議您執行下列動作：

- 需要長度至少 8 個字元的複雜密碼。
- 強制執行密碼過期政策。

若要檢閱完整的資料庫使用者清單，請使用 `SELECT * FROM pg_user;` 命令。

使用 Babelfish 進行 Kerberos 身分驗證

Babelfish for Aurora PostgreSQL 15.2 版支援使用 Kerberos 的資料庫叢集身分驗證。此方法允許您使用 Microsoft Windows 驗證，在使用者連線至您的 Babelfish 資料庫時進行身分驗證。若要這樣做，您必須先將資料庫叢集設定為 AWS Directory Service for Microsoft Active Directory 用於 Kerberos 身分驗證。如需詳細資訊，請參閱《 AWS Directory Service 管理指南》中的什麼是 Directory Service？。

設定 Kerberos 身分驗證

Babelfish for Aurora PostgreSQL 資料庫叢集可以使用兩個不同的連接埠進行連線，但 Kerberos 身分驗證設定是一次性程序。因此，您必須先為資料庫叢集設定 Kerberos 身分驗證。如需詳細資訊，請參閱設定 Kerberos 身分驗證。完成設定之後，請確認您可以使用 Kerberos 連線至 PostgreSQL 用戶端。如需詳細資訊，請參閱使用 Kerberos 身分驗證進行連線。

Babelfish 的登入和使用者佈建

從表格式資料串流 (TDS) 連接埠建立的 Windows 登入可與 TDS 或 PostgreSQL 連接埠搭配使用。首先，可以使用 Kerberos 進行身分驗證的登入必須先從 TDS 連接埠佈建，才能讓 T-SQL 使用者和應用程式連線至 Babelfish 資料庫。建立 Windows 登入時，管理員可以使用 DNS 網域名稱或 NetBIOS 網域名稱來提供登入。一般而言，NetBIOS 網域是 DNS 網域名稱的子網域。例如，如果 DNS 網域名稱是 `CORP.EXAMPLE.COM`，則 NetBIOS 網域可以是 `CORP`。如果針對登入提供 NetBIOS 網域名稱格式，則 DNS 網域名稱必須存在映射。

管理 NetBIOS 網域名稱與 DNS 網域名稱的映射

為了管理 NetBIOS 網域名稱與 DNS 網域名稱之間的映射，Babelfish 提供系統預存程序來新增、移除和截斷這些映射。只有具 `sysadmin` 角色的使用者可以執行這些程序。

若要在 NetBIOS 與 DNS 網域名稱之間建立映射，請使用 Babelfish 提供的系統預存程序 `babelfish_add_domain_mapping_entry`。這兩個引數都必須有一個不是 NULL 的有效值。

Example

```
EXEC babelfish_add_domain_mapping_entry 'netbios_domain_name',
 'fully_qualified_domain_name'
```

下面範例說明如何在 NetBIOS 名稱 CORP 與 DNS 網域名稱 CORP.EXAMPLE.COM 之間建立映射：

Example

```
EXEC babelfish_add_domain_mapping_entry 'corp', 'corp.example.com'
```

若要刪除現有的映射項目，請使用系統預存程序 babelfish_remove_domain_mapping_entry。

Example

```
EXEC babelfish_remove_domain_mapping_entry 'netbios_domain_name'
```

下面範例說明如何移除 NetBIOS 名稱 CORP 的映射。

Example

```
EXEC babelfish_remove_domain_mapping_entry 'corp'
```

若要移除所有現有的映射項目，請使用系統預存程序 babelfish_truncate_domain_mapping_table：

Example

```
EXEC babelfish_truncate_domain_mapping_table
```

若要檢視 NetBIOS 與 DNS 網域名稱之間的所有映射，請使用下列查詢。

Example

```
SELECT netbios_domain_name, fq_domain_name FROM babelfish_domain_mapping;
```

管理登入

建立登入

使用具正確權限的登入，透過 TDS 端點連線至資料庫。若使用者沒有為登入建立資料庫，則登入會映射至訪客使用者。若未啟用訪客使用者，則登入嘗試失敗。

使用下列查詢建立 Windows 登入。FROM WINDOWS 選項允許使用 Active Directory 進行驗證。

```
CREATE LOGIN login_name FROM WINDOWS [WITH DEFAULT_DATABASE=database]
```

Example

下列範例顯示以 db1 的預設資料庫，建立 Active Directory 使用者 [corp\test1] 的登入。

```
CREATE LOGIN [corp\test1] FROM WINDOWS WITH DEFAULT_DATABASE=db1
```

此範例假設 NetBIOS 網域 CORP 與 DNS 網域名稱 CORP.EXAMPLE.COM 之間存在映射。如果沒有映射，則您必須提供 DNS 網域名稱 [CORP.EXAMPLE.COM\test1]。

> **Note**
>
> 根據 Active Directory 使用者的登入會限制為少於 21 個字元的名稱。

捨棄登入

若要捨棄登入，請使用與任何登入相同的語法，如下範例所示：

```
DROP LOGIN [DNS domain name\login]
```

更改登入

若要更改登入，請使用與任何登入相同的語法，如下範例所示：

```
ALTER LOGIN [DNS domain name\login] { ENABLE|DISABLE|WITH DEFAULT_DATABASE=[master] }
```

ALTER LOGIN 命令支援有限的 Windows 登入選項，包括下列項目：

- DISABLE – 停用登入。您不能使用已停用的登入進行驗證。
- ENABLE – 啟用已停用的登入。
- DEFAULT_DATABASE – 變更登入的預設資料庫。

> **Note**
>
> 所有密碼管理都透過 執行 Directory Service，因此 ALTER LOGIN 命令不允許資料庫管理員變更或設定 Windows 登入的密碼。

使用 Kerberos 身分驗證連線至 Babelfish for Aurora PostgreSQL

一般而言，使用 Kerberos 進行驗證的資料庫使用者是從其用戶端機器執行這項操作。這些機器是 Active Directory 網域的一部份。他們使用其用戶端應用程式的 Windows 驗證來存取 TDS 連接埠上的 Babelfish for Aurora PostgreSQL 伺服器。

使用 Kerberos 身分驗證連線至 PostgreSQL 連接埠上的 Babelfish for Aurora PostgreSQL

您可以使用以 TDS 或 PostgreSQL 連接埠，從 TDS 連接埠建立的登入。但是根據預設，PostgreSQL 會針對使用者名稱使用區分大小寫的比較。若要讓 Aurora PostgreSQL 將 Kerberos 使用者名稱解譯為不區分大小寫，您必須在自訂 Babelfish 叢集參數群組中，將 `krb_caseins_users` 參數設為 `true`。此參數預設為 `false`。如需詳細資訊，請參閱設定不區分大小寫的使用者名稱。此外，您必須以 <login@DNS 網域名稱> 格式，從 PostgreSQL 用戶端應用程式指定登入使用者名稱。您不能使用 <DNS 網域名稱\登入> 格式。

常見錯誤

您可以在內部部署 Microsoft Active Directory 與 AWS Managed Microsoft AD之間設定樹系信任關係。如需詳細資訊，請參閱建立信任關係。然後，您必須使用專門的網域特定端點進行連線，而非使用主機端點中的 Amazon 網域 `rds.amazonaws.com`。若不使用正確的網域特定端點，可能會發生下列錯誤：

```
Error: “Authentication method "NTLMSSP" not supported (Microsoft SQL Server, Error:
 514)"
```

TDS 用戶端無法快取所提供端點 URL 的服務票證時，就會發生此錯誤。如需更多詳細資訊，請參閱使用 Kerberos 連線。

使用 Active Directory 安全群組為 Babelfish 設定 Kerberos 驗證

從 Babelfish 4.2.0 版開始，您可以使用 Active Directory 安全群組為 Babelfish 設定 Kerberos 身分驗證。若要使用 Active Directory 設定 Kerberos 身分驗證，以下是要完成的先決條件：

- 您必須遵循使用 Babelfish 進行 Kerberos 身分驗證中提到的所有步驟。
- 確保資料庫執行個體與 Active Directory 相關聯。若要驗證，您可以在 主控台中或透過執行 describe-db-instances AWS CLI 命令來檢視網域成員資格的狀態。

  資料庫執行個體的狀態應為啟用 kerberos 的狀態。如需了解網域成員資格的詳細資訊，請參閱了解網域成員資格。
- 使用以下查詢驗證 NetBIOS 網域名稱與 DNS 網域名稱之間的對應：

  ```
  SELECT netbios_domain_name, fq_domain_name FROM babelfish_domain_mapping;
  ```
- 繼續進行之前，請先驗證使用個別登入的 Kerberos 身分驗證是否如預期運作。以 Active Directory 使用者身分使用 Kerberos 身分驗證的連線應會成功。如果您遇到任何問題，請參閱常見錯誤。

設定 pg_ad_mapping 延伸模組

您必須遵循設定 pg_ad_mapping 延伸模組中提到的所有步驟。若要驗證是否已安裝延伸模組，請從 TDS 端點執行下列查詢：

```
1> SELECT extname, extversion FROM pg_extension where extname like 'pg_ad_mapping';
2> GO
extname       extversion
------------- ----------
pg_ad_mapping 0.1

(1 rows affected)
```

管理群組登入

依照管理登入中所述步驟建立群組登入。我們建議登入名稱與 Active Directory (AD) 安全群組名稱應相同，以便輕鬆維護，但這不是強制性的。例如：

```
CREATE LOGIN [corp\accounts-group] FROM WINDOWS [WITH DEFAULT_DATABASE=database]
```

使用 AD 安全群組對應 T-SQL 群組登入

您需要為每個需要存取資料庫伺服器的 AD 安全群組明確佈建 T-SQL Windows 群組登入。屬於至少一個佈建 AD 安全群組的 AD 使用者將可存取資料庫伺服器。

> **Note**
>
> 此 T-SQL 登入無法再使用密碼型身分驗證進行身分驗證。

例如，accounts-group 是 AD 中的安全群組，如果您想要在 Babelfish 中佈建此安全群組，則必須使用格式 [corp\accounts-group]。

- AD 安全群組：accounts-group
- TSQL 登入：[corp\accounts-group]
- 指定 TSQL 登入的同等 PG 角色：accounts-group@CORP.EXAMPLE.COM

管理員現在可以繼續透過下列 psql 命令，從 PostgreSQL 端點建立 AD 安全群組和 T-SQL 登入之間的對應。如需函數使用的詳細資訊，請參閱使用 pg_ad_mapping 延伸模組中的函數。

> **Note**
>
> 新增對應時，應以 login_name@FQDN 格式指定 T-SQL 登入。透過 TDS 端點進行連線時，會忽略權重。如需權重使用的詳細資訊，請參閱透過 PostgreSQL 連接埠上的 PostgreSQL 端點連線至 Babelfish。

```
postgres=>select pgadmap_set_mapping('accounts-group', 'accounts-
group@CORP.EXAMPLE.COM', <SID>, <Weight>);
```

如需擷取 AD 安全群組 SID 的資訊，請參閱在 PowerShell 中擷取 Active Directory 群組 SID。

下表顯示從 AD 安全群組到 T-SQL 登入的範例對應：

AD 安全群組	TSQL 登入	指定 TSQL 登入的同等 PG 角色	Weight (粗細)
accounts-group	[corp\accounts-group]	accounts-group@CORP.EXAMPLE.COM	7
sales-group	[corp\sales-group]	sales-group@CORP.EXAMPLE.COM	10
dev-group	[corp\dev-group]	dev-group@CORP.EXAMPLE.COM	7

```
postgres=> select admap.ad_sid, admap.ad_grp, lgn.orig_loginname, lgn.rolname,
 admap.weight from pgadmap_read_mapping() as admap, sys.babelfish_authid_login_ext as
 lgn where admap.pg_role = lgn.rolname;
    ad_sid    |     ad_grp     |    orig_loginname   |            rolname
 | weight
--------------+----------------+---------------------+--------------------------------
+--------
```

```
 S-1-5-67-890 | accounts-group | corp\accounts-group  | accounts-group@CORP.EXAMPLE.COM
 |     7
 S-1-2-34-560 | sales-group    | corp\sales-group     | sales-group@CORP.EXAMPLE.COM
 |     10
 S-1-8-43-612 | dev-group      | corp\dev-group       | dev-group@CORP.EXAMPLE.COM
 |     7
 (7 rows)
```

透過 TDS 端點連線至 Babelfish

在下列範例中，user1 是 accounts-group 和 sales-group 的成員，user2 是 accounts-group 和 dev-group 的成員。

使用者名稱	AD 安全群組成員資格
user1	accounts-group、sales-group
user2	accounts-group、dev-group

使用 sqlcmd 公用程式連線至 Babelfish 資料庫伺服器。您可以依照下列範例，使用 Kerberos 驗證使用者 (此範例中的 user1)：

```
1> select principal, gss_authenticated from pg_stat_gssapi where pid =
 pg_backend_pid();
2>  GO
principal               gss_authenticated
----------------------  -----------------
user1@CORP.EXAMPLE.COM  1

((1 rows affected))
1> select suser_name();
2>  GO
suser_name
----------
corp\user1

(1 rows affected)
```

在此範例中，user1 會繼承帳戶群組和銷售群組的權限。您可以使用 `sys.login_token` 系統檢視來驗證群組成員資格。

```
1> SELECT name, type FROM sys.login_token;
2>  GO
name                type
------------------- ----
corp\accounts-group WINDOWS GROUP
corp\sales-group    WINDOWS GROUP

(2 rows affected)
```

稽核和記錄

若要判斷 AD 安全主體身分，請使用下列命令：

```
1> select suser_name();
2> GO
suser_name
----------
corp\user1

(1 rows affected)
```

目前，日誌中看不到 AD 使用者身分。您可以開啟 `log_connections` 參數來記錄資料庫工作階段的建立。如需詳細資訊，請參閱 log_connections。如下列範例所示，輸出包含作為主體的 AD 使用者身分。與此輸出相關聯的後端 PID 可以協助將動作追溯至實際 AD 使用者。

```
bbf_group_ad_login@babelfish_db:[615]:LOG: connection authorized:
 user=bbf_group_ad_login database=babelfish_db application_name=sqlcmd GSS
 (authenticated=yes, encrypted=yes, principal=user1@CORP.EXAMPLE.COM)
```

AD 安全群組成員資格的使用權限

繼承伺服器層級權限

屬於指定 AD 安全群組成員的 AD 使用者會繼承伺服器層級權限，這些權限會授予至對應的 Windows 群組登入。例如，請考慮 `accounts-group` AD 安全群組，此群組會在 Babelfish 中獲授予 `sysadmin` 伺服器角色成員資格。您可以使用下列命令繼承伺服器層級權限：

```
1> ALTER SERVER ROLE sysadmin ADD MEMBER [corp\accounts-group];
```

因此，身為 accounts-group AD 安全群組成員的任何 Active Directory 使用者，都會繼承與該 sysadmin 角色相關聯的伺服器層級權限。這表示像 corp\user1 這樣的使用者 (身為 accounts-group 的成員) 現在將能夠在 Babelfish 中執行伺服器層級操作。

> **Note**
>
> 若要執行伺服器層級 DDL，個別 AD 使用者的 Windows 登入必須存在。如需詳細資訊，請參閱限制。

繼承資料庫層級權限

若要授予資料庫層級權限，必須使用 Windows 群組登入來建立和對應資料庫使用者。屬於指定 AD 安全群組成員的 AD 使用者會繼承資料庫層級權限，這些權限會授予至該資料庫使用者。在下列範例中，您可以查看如何指派 Windows 群組 [corp\accounts-group] 的資料庫層級權限。

```
1> CREATE DATABASE db1;
2> GO
1> USE db1;
2> GO
Changed database context to 'db1'.
1> CREATE TABLE dbo.t1(a int);
2> GO
```

為 Windows 群組登入 [corp\accounts-group] 建立資料庫使用者 [corp\sales-group]。若要執行此步驟，請使用身為 sysadmin 成員的登入透過 TDS 端點進行連線。

```
1> CREATE USER [corp\accounts-group] FOR LOGIN [corp\accounts-group];
2> GO
```

現在，請以 AD 使用者 user1 身分進行連線，以檢查資料表 t1 的存取權。我們尚未授予資料庫層級權限，因此會導致許可遭拒錯誤。

```
1> SELECT * FROM dbo.t1;
2> GO
Msg 33557097, Level 16, State 1, Server db-inst, Line 1
permission denied for table t1
```

將資料表 t1 的 SELECT 授予資料庫使用者 [corp\accounts-group]。若要執行此步驟，請使用身為 sysadmin 成員的登入透過 TDS 端點進行連線。

```
1> GRANT SELECT ON dbo.t1 TO [corp\accounts-group];
2> GO
```

以 AD 使用者 user1 身分進行連線以驗證存取權。

```
1> SELECT * FROM dbo.t1;
2> GO
a
-----------

(0 rows affected)
```

根據預設或明確結構描述處理 DDL 陳述式行為

使用 AD 驗證工作階段時，目前工作階段的預設結構描述取決於下列條件：

- 如果存在個別資料庫使用者，會將使用者的預設結構描述視為目前工作階段的預設結構描述。
- 如果群組資料庫使用者的預設結構描述存在，則會將群組資料庫使用者的預設結構描述視為具有最小主體 ID 的目前工作階段預設結構描述。

了解 CREATE DDL 陳述式行為

如果 CREATE DDL 陳述式中未指定明確的結構描述，則會在目前工作階段的預設結構描述中建立物件。如果無法判斷結構描述是預設還是明確，則 DDL 陳述式會擲出下列錯誤：

```
"Babelfish Unsupported Command : Schema required for CREATE DDLs when connecting with
 Active Directory Group authentication. Assign default schema to group user or specify
 schema in command."
```

Example：Windows 群組使用者的預設結構描述不存在

Windows 群組使用者 [corp\accounts-group] 具有 NULL 預設結構描述，而 AD 使用者 user1 正嘗試執行 DDL，而不明確指定結構描述。由於 user1 不存在個別 Windows 登入和使用者，因此只會取得 Windows 群組使用者 [corp\accounts-group] 的資料庫層級權限。

```
1> create TABLE t2(a int);
2> GO

Msg 33557097, Level 16, State 1, Server db-inst, Line 1
Babelfish Unsupported Command : Schema required for CREATE DDLs when connecting with
 Active Directory Group authentication. Assign default schema to group user or specify
 schema in command.
```

> **Note**
>
> AD 使用者 user1 的個別 Windows 登入和使用者不存在

Example：Windows 群組使用者存在預設結構描述

使用 sysadmin 以預設結構描述建立 [corp\accounts-group] 登入的 Windows 群組使用者。

```
1> CREATE USER [corp\accounts-group] FOR LOGIN [corp\accounts-group] WITH
 DEFAULT_SCHEMA = sch_acc;
2> GO
1> CREATE SCHEMA sch_acc AUTHORIZATION [gad\accounts-group];
2> GO
1> SELECT name, principal_id, default_schema_name FROM sys.database_principals WHERE
 name = 'corp\accounts-group';
2> GO
```

```
name                principal_id default_schema_name
------------------- ------------ -------------------
corp\accounts-group 24162        sch_acc

(1 rows affected)
```

嘗試建立物件，而不使用 AD 使用者 user1 明確指定結構描述。系統將在 [corp\accounts-group] Windows 群組的預設結構描述中建立資料表 t2。此物件的擁有者將與結構描述 sch_acc 的擁有者相同。

```
1> CREATE TABLE t_group(a int);
2> GO
1> SELECT name, schema_name(schema_id) FROM sys.objects WHERE name like 't_group';
2> GO

name    schema_name
------- -----------
t_group sch_acc

(1 rows affected)
```

> **Note**
>
> AD 使用者 user1 的個別 Windows 登入和使用者不存在

Example：AD 使用者的個別資料庫使用者也存在

如果 AD 使用者的個別資料庫使用者也存在，則一律會在與個別資料庫使用者相關聯的結構描述中建立物件。如果資料庫使用者不存在結構描述，則會使用 dbo 結構描述。為 AD 使用者 user1 建立個別 Windows 登入和資料庫使用者。使用 sysadmin 登入透過 TDS 端點進行連線

```
1> CREATE LOGIN [corp\user1] FROM WINDOWS;
2> GO
```

```
1> CREATE USER [corp\user1] FOR LOGIN [corp\user1] WITH DEFAULT_SCHEMA = sch1;
2> GO
1> CREATE SCHEMA sch1 AUTHORIZATION [corp\user1];
2> GO
1> SELECT name, default_schema_name FROM sys.database_principals WHERE name = 'corp
\user1';
2> GO

name      default_schema_name
--------- -------------------
corp\user1 sch1

(1 rows affected)
```

使用 AD 使用者 user1 進行連線，並嘗試建立物件，而不明確指定結構描述。將在結構描述 sch1 中建立資料表 t2。另請注意，此物件的擁有者將與結構描述 sch1 的擁有者相同。

```
1> CREATE TABLE t2(a int);
2> GO
1> SELECT name, schema_name(schema_id) FROM sys.objects WHERE name like 't2';
2> GO

name schema_name
---- -----------
t2   sch1

(1 rows affected)
```

限制

- 傾印/還原公用程式不支援傾印 pg_ad_mapping 延伸模組對應。還原後，您將需要重新建立那些對應。
- 針對使用 pg_ad_mapping 的 Babelfish 和 Aurora PostgreSQL 執行個體，不支援藍綠部署。
- 不支援隱含結構描述的建立。不支援需要隱含結構描述建立的 DDL 陳述式。
- 當個別 Windows 登入不存在時，群組 AD 驗證工作階段中不支援伺服器層級 DDL ALTER AUTHORIZATION ON DATABASE、CREATE DATABASE、CREATE LOGIN、ALTER

LOGIN、ALTER SERVER ROLE、ALTER DATABASE，只有群組 Windows 登入存在。若要解決此限制，建議在密碼驗證工作階段中執行這些操作，或建立個別 Windows 登入。

- 不支援隱含使用者建立。理想的 T-SQL 行為 [Babelfish 中尚不支援]；在某些情況下，例如 DDL 和 GRANT/REVOKE 之類存取控制陳述式，在命令中會指定 AD 使用者名稱，但資料庫中不存在該名稱時，則會隱含地建立名為 AD 使用者的資料庫使用者。
- 對於從 PSQL 端點建立且從群組 AD 驗證工作階段中 TDS 端點執行的 PL/pgSQL 程序或函數中的 DDL：
  - 將支援 ALTER/DROP 陳述式。
  - CREATE TABLE、CREATE VIEW、CREATE INDEX、CREATE FUNCTION/PROC、CREATE TYPE、CREATE SEQUENCE、CREATE TRIGGER、SELECT INTO、CREATE FULLTEXT INDEX、CREATE UNIQUE INDEX 會在未明確提供結構描述且目前工作階段的預設結構描述為 null 時擲出錯誤。
  - CREATE DATABASE、CREATE EXTENSION 和所有其他 CREATE 陳述式 (適用於 PG(不在 T-SQL 中) 特定物件 CREATE 訂閱、CREATE 資料表空間、CREATE 政策、CREATE 轉換的 CREATE 陳述式將不受支援。
- Group AD 驗證工作階段中不支援 PostgreSQL 端點的 DDL。作為解決方法，您可以隨時使用主要使用者或任何其他使用密碼型身分驗證機制的使用者進行連線。
- SUSER_SID()、IS_SRVROLEMEMBER()、IS_MEMBER()、sys.dm_exec_sessions 等系統物件具有下列限制。
  - 提供 AD 使用者或 AD 安全群組時，SUSER_SID() 不會傳回 SID。
  - 如果目前的 AD 使用者從任何 Windows 群組登入的伺服器角色成員資格繼承伺服器角色成員資格，則 IS_SRVROLEMEMBER() 不會考慮角色成員資格。
  - IS_MEMBER() 會針對任何 Windows 群組相關查詢傳回 false。
  - sys.dm_exec_sessions 不會顯示預期值 login_name、nt_user_name 欄。

透過 PostgreSQL 連接埠上的 PostgreSQL 端點連線至 Babelfish

您也可以利用從 TDS 連接埠建立的群組登入，透過 PostgreSQL 連接埠進行連線。若要透過 PostgreSQL 連接埠進行連線，您需要從 PostgreSQL 用戶端應用程式以格式 `<ad_username@FQDN>` 指定 AD 使用者名稱。您不能使用 `<DNS domain name\ad_username>` 格式。

根據預設，PostgreSQL 會針對使用者名稱使用區分大小寫的比較。若要讓 Aurora PostgreSQL 將 Kerberos 使用者名稱解譯為不區分大小寫，您必須在自訂 Babelfish 叢集參數群組中，將 krb_caseins_users 參數設為 true。此參數預設為 false。如需詳細資訊，請參閱針對不區分大小寫的使用者名稱設定 Aurora PostgreSQL 資料庫叢集。

當 AD 使用者屬於多個群組時，T-SQL 和 PostgreSQL 端點的行為差異

假設 AD 使用者 user1 是兩個 AD 安全群組 [corp\accounts-group] 和 [corp\sales-group] 的一部分，且資料庫管理員已透過下列方式設定使用者對應。

```
postgres=> select * from pgadmap_read_mapping();

ad_sid       | pg_role                           | weight | ad_grp
-------------+-----------------------------------+--------+---------------
S-1-5-67-980 | accounts-group@CORP.EXAMPLE.COM | 7      | accounts-group
S-1-2-34-560 | sales-group@CORP.EXAMPLE.COM    | 10     | sales-group
(2 rows)
```

如果使用者從 T-SQL 端點連線，則在授權期間，其會繼承所有相關聯 T-SQL 登入的權限。在此範例中，user1 將從 T-SQL 群組登入繼承權限聯集，且權重將遭到忽略。這符合標準 T-SQL 行為。

不過，如果相同的使用者從 PostgreSQL 端點進行連線，則只能從一個具有最高權重的相關聯 T-SQL 登入繼承權限。如果兩個 T-SQL 群組登入受指派相同的權重，則 AD 使用者將繼承與最近新增的對應相對應的 T-SQL 登入權限。對於 PostgreSQL，建議指定反映個別資料庫角色相對許可/權限的權重，以避免模棱兩可的情況。在下列範例中，user1 透過 PSQL 端點連線，並僅繼承銷售群組權限。

```
babelfish_db=> select session_user, current_user;

   session_user               |   current_user
------------------------------+---------------------------
 sales-group@CORP.EXAMPLE.COM | sales-group@CORP.EXAMPLE.COM
(1 row)


babelfish_db=> select principal, gss_authenticated from pg_stat_gssapi where pid =
 pg_backend_pid();

     principal          | gss_authenticated
------------------------+-------------------
 user1@CORP.EXAMPLE.COM | t
(1 row)
```

連線至 Babelfish 資料庫叢集

若要連線至 Babelfish，請連線至執行 Babelfish 的 Aurora PostgreSQL 叢集端點。您的用戶端可以使用下列其中一個符合 TDS 7.1 到 7.4:版的用戶端驅動程式：

- 開放式資料庫連線 (ODBC)
- OLE DB 驅動程式/MSOLEDBSQL
- Java 資料庫連線 (JDBC) 8.2.2 版 (mssql-jdbc-8.2.2) 及更新版本
- Microsoft SqlClient Data Provider for SQL Server
- .NET Data Provider for SQL Server
- SQL Server Native Client 11.0 (已棄用)
- OLE DB Provider/SQLOLEDB (已棄用)

BabelSQL 可讓您執行如下：

- 在 TDS 連接埠上 (預設為連接埠 1433) 執行 SQL Server 工具、應用程式和語法。
- 在 PostgreSQL 連接埠上 (預設為連接埠 5432) 執行 PostgreSQL 工具、應用程式和語法。

若要進一步了解連線至 Aurora PostgreSQL 的一般資訊，請參閱 連接至 Amazon Aurora PostgreSQL 資料庫叢集。

> **Note**
>
> 不支援使用 SQL Server OLEDB 提供者存取中繼資料的第三方開發人員工具。建議您針對這些工具使用 SQL Server JDBC、ODBC 或 SQL 原生用戶端連線。

從 Babelfish 5.1.0 版開始，預設會強制執行端對端連線加密。若要確保持續連線能力：

- 為連線設定 SSL/TLS 加密。如需詳細資訊，請參閱使用 SSL/TLS 加密與資料庫叢集的連線。
- 在用戶端電腦匯入所需的憑證。如需詳細資訊，請參閱搭配 Microsoft SQL Server 資料庫執行個體使用 SSL。

如果您想要繼續使用舊版 Babelfish (5.1.0 版之前) 的加密設定，您可以在資料庫叢集參數群組中將 `rds.force_ssl` 參數設為 `0`。

主題

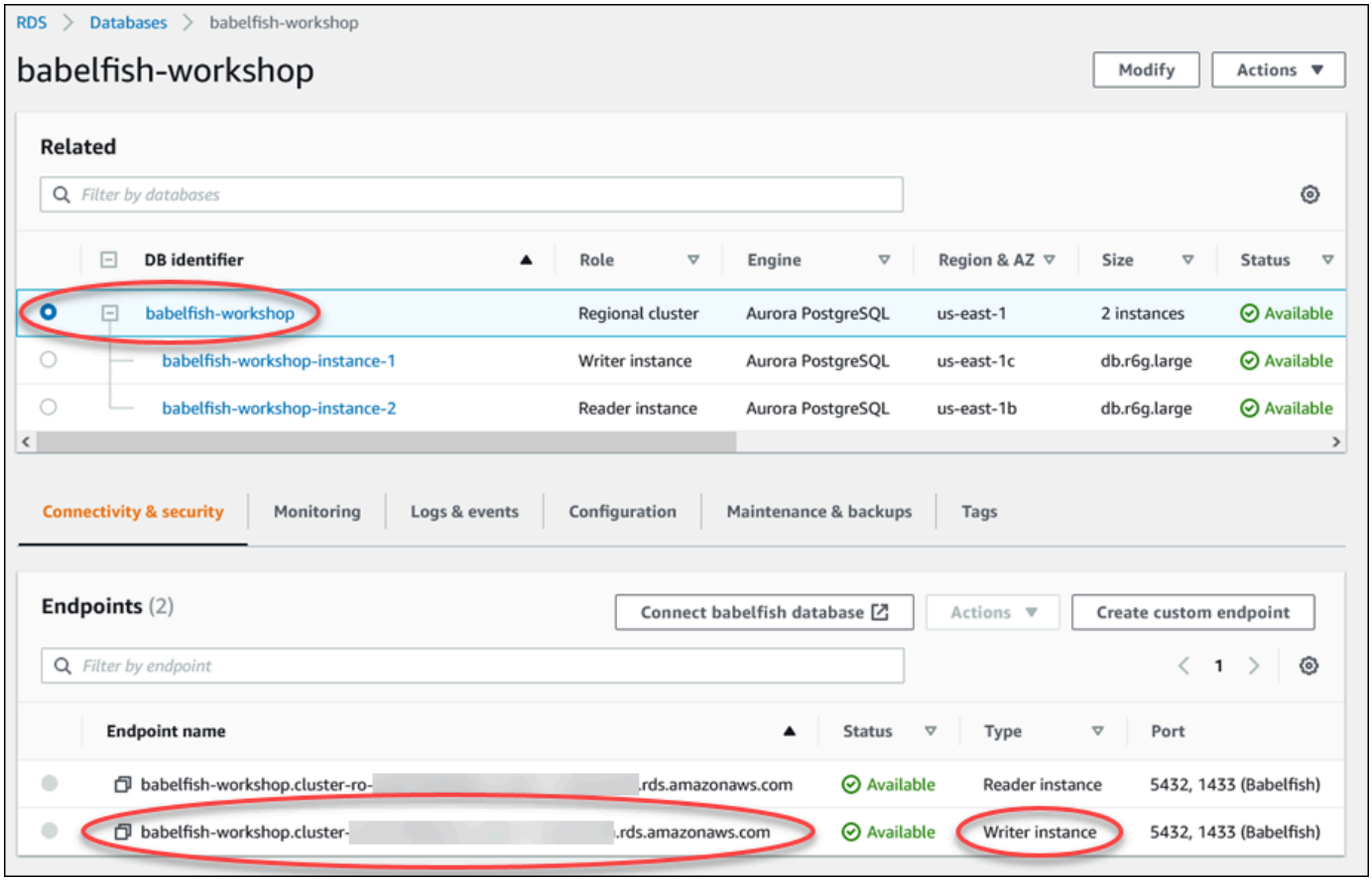
- 尋找寫入器端點和連接埠號碼
- 建立對 Babelfish 的 C# 或 JDBC 用戶端連線
- 使用 SQL Server 用戶端來連線至資料庫叢集
- 使用 PostgreSQL 用戶端來連線至資料庫叢集

尋找寫入器端點和連接埠號碼

若要連線到 Babelfish 資料庫叢集，請使用與資料庫叢集的寫入器 (主要) 執行個體相關聯的端點。執行個體的狀態必須為 Available (可用)。在為 Aurora PostgreSQL 資料庫叢集建立 Babelfish 後，執行個體可能需要 20 分鐘才可供使用。

尋找資料庫端點

1. 開啟 Babelfish 的主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 從列出的項目中選取您的 Babelfish for Aurora PostgreSQL 資料庫叢集，以查看其詳細資料。
4. 在 Connectivity & security (連線與安全性) 索引標籤上，注意可用的叢集 Endpoints (端點) 值。對於執行資料庫寫入或讀取操作的任何應用程式，請在連線字串中使用寫入器執行個體的叢集端點。



如需 Aurora 資料庫叢集的詳細資訊，請參閱 建立 Amazon Aurora 資料庫叢集。

建立對 Babelfish 的 C# 或 JDBC 用戶端連線

在下文中，您可以找到使用 C# 和 JDBC 類別連線至 Babelfish for Aurora PostgreSQL 的一些範例。

Example：使用 C# 程式碼來連線至資料庫叢集

```
string dataSource = 'babelfishServer_11_24';

//Create connection
connectionString = @"Data Source=" + dataSource
    +";Initial Catalog=your-DB-name"
    +";User ID=user-id;Password=password";

// [optional] To validate server certificate during TLS/SSL connection
connectionString = connectionString + ";ServerCertificate=/path/to/certificate.pem";


SqlConnection cnn = new SqlConnection(connectionString);
cnn.Open();
```

Example：使用通用 JDBC API 類別和界面來連線至資料庫叢集

```
String dbServer =
   "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";
String connectionUrl = "jdbc:sqlserver://" + dbServer + ":1433;" +
    "databaseName=your-DB-name;user=user-id;password=password";

// [optional] To validate server certificate during TLS/SSL connection
connectionUrl = connectionUrl + ";serverCertificate=/path/to/certificate.pem";

// Load the SQL Server JDBC driver and establish the connection.
System.out.print("Connecting Babelfish Server ... ");
Connection cnn = DriverManager.getConnection(connectionUrl);
```

Example：使用 SQL Server 特定 JDBC 類別和界面來連線至資料庫叢集

```
// Create datasource.
SQLServerDataSource ds = new SQLServerDataSource();
ds.setUser("user-id");
ds.setPassword("password");
String babelfishServer =
   "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";
```

```
ds.setServerName(babelfishServer);
ds.setPortNumber(1433);
ds.setDatabaseName("your-DB-name");

// [optional] To validate server certificate during TLS/SSL connection
ds.setServerCertificate("/path/to/certificate.pem");

Connection con = ds.getConnection();
```

Important

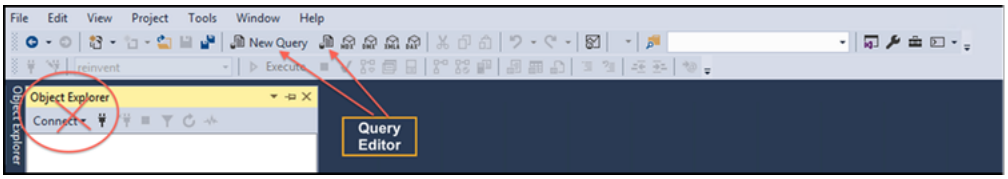
確保憑證符合 AWS 管理主控台上資料庫叢集組態中顯示的憑證授權機構。

使用 SQL Server 用戶端來連線至資料庫叢集

您可以使用 SQL Server 用戶端在 TDS 連接埠上與 Babelfish 連線。自 Babelfish 2.1.0 及更新版本開始，您可以使用 SSMS 物件總管或 SSMS 查詢編輯器連線到您的 Babelfish 叢集。

限制

- 在 Babelfish 2.1.0 及更早版本中，使用 PARSE 檢查 SQL 語法未如預期運作。PARSE 命令並不會在未執行查詢的情況下檢查語法，而是會執行查詢，但不會顯示任何結果。使用 SMSS <Ctrl><F5> 組合鍵檢查語法是否具有相同的異常行為，亦即 Babelfish 意外執行查詢但不提供任何輸出。
- Babelfish 不支援 MARS (多活動結果集)。請確保用於連線到 Babelfish 的任何用戶端應用程式都沒有設定為使用 MARS。
- 對於 Babelfish 1.3.0 及較舊版本，SSMS 僅支援查詢編輯器。若要將 SSMS 搭配 Babelfish 使用，請務必在 SSMS 中開 [Query Editor] (查詢編輯器) 連線對話方塊，而不是物件總管。如果 Object Explorer (物件總管) 對話方塊開啟，請取消該對話方塊，再重新開啟 Query Editor (查詢編輯器)。在下圖中，您可以找到當連線到 Babelfish 1.3.0 或較舊版本時可選擇的功能表選項。



如需 SQL Server 和 Babelfish 之間的互通性和行為差異的詳細資訊，請參閱 Babelfish for Aurora PostgreSQL 與 SQL Server 之間的差異。

使用 sqlcmd 來連線至資料庫叢集

您可以僅使用 SQL Server `sqlcmd` 命令列用戶端 19.1 版或更舊版本，以連線至支援 Babelfish 的 Aurora PostgreSQL 資料庫叢集並與其互動。不支援 SSMS 19.2 版來連線到 Babelfish 叢集。使用下列命令來連線。

```
sqlcmd -S endpoint,port -U login-id -P password -d your-DB-name
```

選項如下：

- `-S` 是資料庫叢集的端點和 (選用) TDS 連接埠。
- `-U` 是使用者的登入名稱。
- `-P` 是與使用者相關聯的密碼。

- -d 是 Babelfish 資料庫的名稱。

連線之後，就可以您在 SQL Server 上同樣使用的許多命令。如需一些範例，請參閱從 Babelfish 系統目錄取得資訊。

使用 SSMS 來連線至資料庫叢集

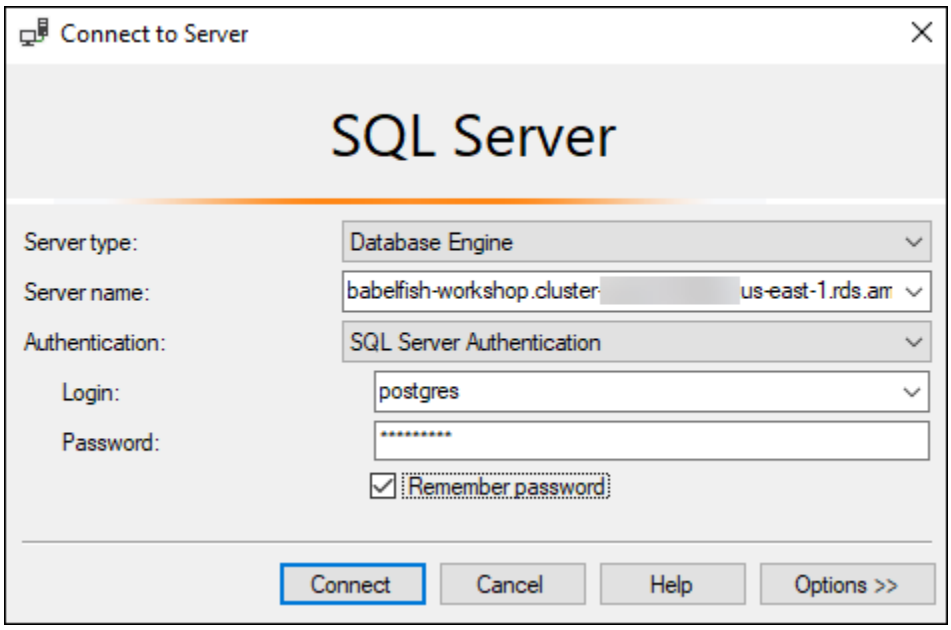
您可以使用 Microsoft SQL Server Management Studio (SSMS) 連線至執行 Babelfish 的 Aurora PostgreSQL 資料庫叢集。SSMS 包括各種工具，包括 SQL Server 匯入與匯出精靈，如 將 SQL Server 資料庫遷移至 Babelfish for Aurora PostgreSQL 中討論。如需 SSMS 的詳細資訊，請參閱 Microsoft 文件中的下載 SQL Server Management Studio (SSMS)。若要設定 SSL/TLS，請參閱搭配 Microsoft SQL Server 資料庫執行個體使用 SSL。

使用 SSMS 來連線至 Babelfish 資料庫

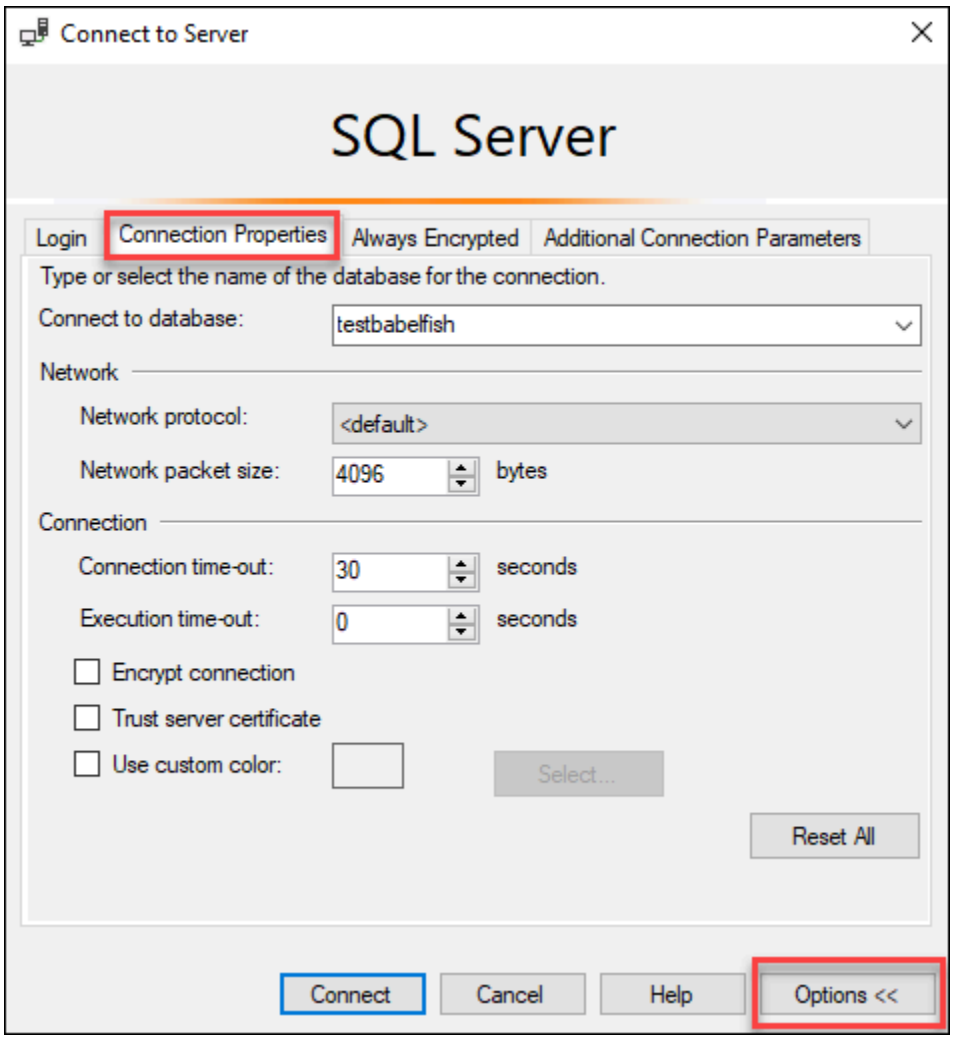
1. 啟動 SSMS。
2. 開啟 Connect to Server (連接至伺服器) 對話方塊。若要繼續連線，請執行以下任一操作：
   - 選擇新增查詢。
   - 如果查詢編輯器已開啟，請選擇查詢、連線、連線。
3. 根據您的資料庫提供下列資訊：
   a. 針對伺服器類型，選擇資料庫引擎。
   b. 針對伺服器名稱，輸入 DNS 名稱。例如，您的伺服器名稱看起來如下。

      ```
      cluster-name.cluster-555555555555.aws-region.rds.amazonaws.com,1433
      ```

   c. 針對驗證，選擇 SQL Server 驗證。
   d. 針對登入，輸入您建立資料庫時選擇的使用者名稱。
   e. 針對密碼，輸入您建立資料庫時選擇的密碼。



4. (選用) 選擇選項，然後選擇連線內容索引標籤。



5. (選用) 針對連線至資料庫，指定要連線的已遷移 SQL Server 資料庫的名稱，然後選擇連線。

   如果出現訊息指出 SSMS 無法套用連線字串，請選擇確定。

   如果您在連接至 Babelfish 時遇到問題，請參閱 連線失敗。

   如需 SQL Server 連線問題的詳細資訊，請參閱《Amazon RDS 使用者指南》中的對您的 SQL Server 資料庫執行個體的連線進行故障診斷。

使用 PostgreSQL 用戶端來連線至資料庫叢集

您可以使用 PostgreSQL 用戶端在 PostgreSQL 連接埠上連線至 Babelfish。從 5.1.0 版開始，Babelfish 伺服器預設會強制執行端對端連線加密。更新應用程式，以使用 SSL/TLS 憑證。如需 SSL/TLS 憑證的設定詳細資訊，請參閱the section called “使用 SSL/TLS 保護 Aurora PostgreSQL 資料的安全”。

使用 psql 來連線至資料庫叢集

您可以從 PostgreSQL 網站下載 PostgreSQL 用戶端。請遵循適用於您作業系統版本的指示，以安裝 psql。

您可以使用 `psql` 命令列用戶端，以查詢支援 Babelfish 的 Aurora PostgreSQL 資料庫叢集。連線時，請使用 PostgreSQL 連接埠 (預設為連接埠 5432)。通常，您不必指定連接埠號碼，除非您將號碼變更成非預設值。從 `psql` 用戶端使用下列命令連線至 Babelfish：

```
psql -h bfish-db.cluster-123456789012.aws-region.rds.amazonaws.com
-p 5432 -U postgres -d babelfish_db
```

參數如下：

- `-h` – 您要存取的資料庫叢集 (叢集端點) 的主機名稱。
- `-p` – 用於連線至資料庫執行個體的 PostgreSQL 連接埠號碼。
- `-d` - 您要連線的資料庫。預設值為 `babelfish_db`。
- `-U` – 您要存取的資料庫使用者帳戶。(此範例顯示預設主要使用者名稱。)

在 psql 用戶端執行 SQL 命令時，請以分號結束命令。例如，下列 SQL 命令查詢 pg_tables 系統檢視表，以傳回資料庫中每個資料表的相關資訊。

```
SELECT * FROM pg_tables;
```

psql 用戶端也有一組內建的中繼命令。中繼命令是一種調整格式的快捷方式，或提供捷徑以簡單易用的格式傳回中繼資料。例如，下列中繼命令傳回類似上一個 SQL 命令的資訊：

```
\d
```

中繼命令不需要以分號 (;) 結束。

若要退出 psql 用戶端，請輸入 `\q`。

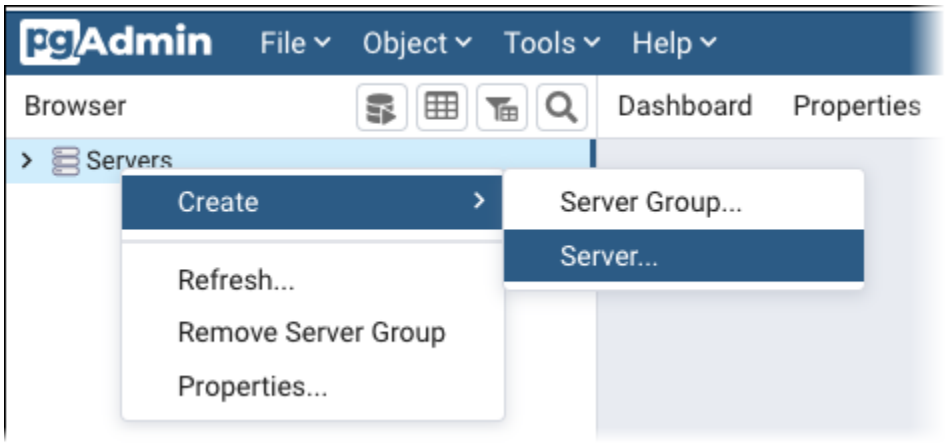
如需使用 psql 用戶端來查詢 Aurora PostgreSQL 叢集的詳細資訊，請參閱 PostgreSQL 文件。

使用 pgAdmin 來連線至資料庫叢集

您可以使用 pgAdmin 用戶端以原生 PostgreSQL 方言存取資料。

使用 pgAdmin 用戶端來連線至叢集

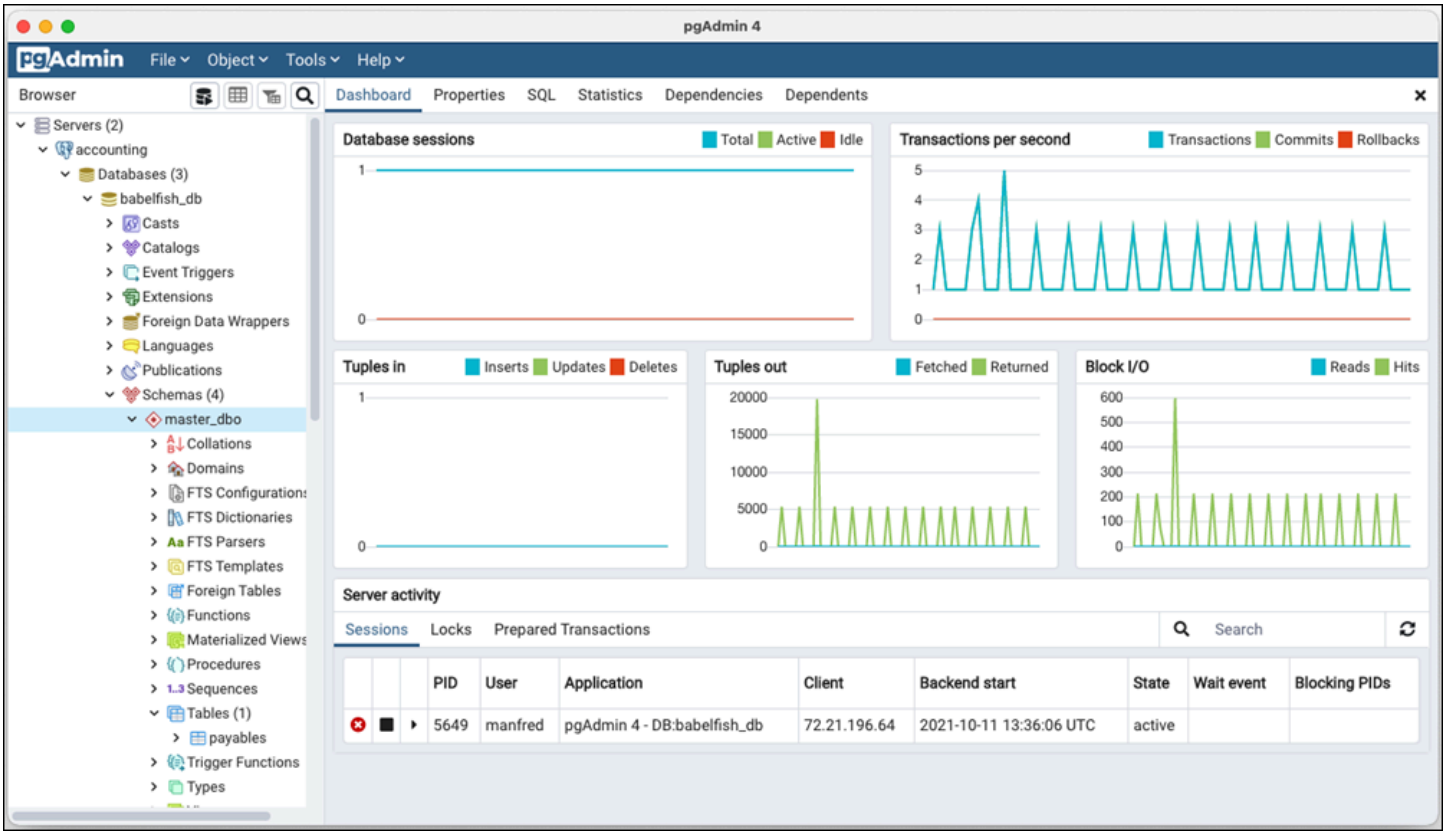
1. 從 pgAdmin 網站下載並安裝 pgAdmin 用戶端。
2. 開啟用戶端並向 pgAdmin 驗證身分。
3. 開啟 Servers (伺服器) 的內容功能表 (按一下滑鼠右鍵)，然後選擇 Create (建立)、Server (伺服器)。



4. 在 Create - Server (建立 - 伺服器) 對話方塊中輸入資訊。

   在 Connection (連線) 索引標籤上，針對 Host (主機)，新增 Aurora PostgreSQL 叢集位址，針對 Port (連接埠)，新增 PostgreSQL 連接埠號碼 (預設為 5432)。提供身分驗證詳細資訊，然後選擇 Save (儲存)。

連線之後，您可以使用 pgAdmin 功能，在 PostgreSQL 連接埠上監控和管理 Aurora PostgreSQL 叢集。



如需進一步了解，請參閱 pgAdmin 網頁。

使用 Babelfish

在下文中，您可以找到 Babelfish 的使用資訊，包括使用 Babelfish 與 SQL Server 之間及使用 Babelfish 與 PostgreSQL 資料庫之間的一些差異。

主題

- 從 Babelfish 系統目錄取得資訊
- 在 Babelfish for Aurora PostgreSQL 中管理許可和存取控制
- Babelfish for Aurora PostgreSQL 與 SQL Server 之間的差異
- 使用具有限制實作的 Babelfish 功能
- 改善 Babelfish 查詢效能
- 搭配 Babelfish 使用 Aurora PostgreSQL 擴充功能
- Babelfish 支援連結的伺服器
- 在 Babelfish 中使用全文搜尋
- Babelfish 支援地理空間資料類型
- 了解 Babelfish 中的分割

從 Babelfish 系統目錄取得資訊

您可以透過查詢許多與 SQL Server 中所使用相同的系統檢視，來取得儲存在 Babelfish 叢集中資料庫物件的相關資訊。Babelfish 的每個新版本都增加對對更多系統檢視的支援。如需目前可用的可用檢視清單，請參閱 SQL Server system catalog views 資料表。

這些系統檢視會提供來自系統目錄 (`sys.schemas`) 的資訊。若為 Babelfish，這些檢視包含 SQL Server 和 PostgreSQL 系統結構描述。若查詢 Babelfish 中的系統目錄資訊，您可以使用 TDS 連接埠或 PostgreSQL 連接埠，如以下範例所示。

- 使用 **`sqlcmd`** 或其他 SQL Server 用戶端查詢 T-SQL 連接埠。

```
1> SELECT * FROM sys.schemas
2> GO
```

此查詢會傳回 SQL Server 和 Aurora PostgreSQL 系統結構描述，如下所示。

```
name
-----------------------------------------------------------
```

```
demographic_dbo
public
sys
master_dbo
tempdb_dbo
...
```

- 使用 **psql** 或 **pgAdmin** 查詢 PostgreSQL 連接埠。此範例使用 psql 清單結構描述中繼命令 (\dn)：

  ```
  babelfish_db=> \dn
  ```

  查詢會傳回與在 T-SQL 連接埠上由 sqlcmd 傳回相同的結果集。

  ```
          List of schemas
               Name
  ------------------------------

   demographic_dbo

   public
   sys
   master_dbo
   tempdb_dbo
  ...
  ```

Babelfish 中可用的 SQL Server 系統目錄

在下表中，您可以找到 Babelfish 中目前實作的 SQL Server 檢視。如需 SQL Server 中系統目錄的詳細資訊，請參閱 Microsoft 文件中的系統目錄檢視 (Transact-SQL)。

檢視表名稱	描述或 Babelfish 限制 (如果有)
sys.all_columns	所有資料表和檢視表中的所有資料欄
sys.all_objects	所有結構描述中的所有物件
sys.all_sql_modules	sys.sql_modules 與 sys.system_sql_modules 的聯集

檢視表名稱	描述或 Babelfish 限制 (如果有)
sys.all_views	所有結構描述中的所有檢視表
sys.columns	使用者定義的資料表和檢視表中的所有資料欄
sys.configurations	Babelfish 支援限於單個唯讀組態。
sys.data_spaces	每個資料空間包含一列。這可以是檔案群組、分割區配置或 FILESTREAM 資料檔案群組。
sys.database_files	此各資料庫檢視中的每一列代表資料庫中的每一個檔案，如同儲存在資料庫本身中。
sys.database_mirroring	如需相關資訊，請參閱 Microsoft Transact-SQL 文件中的 sys.database_mirroring。
sys.database_principals	如需相關資訊，請參閱 Microsoft Transact-SQL 文件中的 sys.database_principals。
sys.database_role_members	如需相關資訊，請參閱 Microsoft Transact-SQL 文件中的 sys.database_role_members。
sys.databases	所有結構描述中的所有資料庫
sys.dm_exec_connections	如需相關資訊，請參閱 Microsoft Transact-SQL 文件中的 sys.dm_exec_connections。
sys.dm_exec_sessions	如需相關資訊，請參閱 Microsoft Transact-SQL 文件中的 sys.dm_exec_sessions。
sys.dm_hadr_database_replica_states	如需相關資訊，請參閱 Microsoft Transact-SQL 文件中的 sys.dm_hadr_database_replica_states。
sys.dm_os_host_info	如需相關資訊，請參閱 Microsoft Transact-SQL 文件中的 sys.dm_os_host_info。
sys.endpoints	如需相關資訊，請參閱 Microsoft Transact-SQL 文件中的 sys.endpoints。

檢視表名稱	描述或 Babelfish 限制 (如果有)
sys.indexes	如需相關資訊，請參閱 Microsoft Transact-SQL 文件中的 sys.indexes。
sys.languages	如需相關資訊，請參閱 Microsoft Transact-SQL 文件中的 sys.languages。
sys.schemas	所有結構描述
sys.server_principals	所有登入和角色
sys.sql_modules	如需相關資訊，請參閱 Microsoft Transact-SQL 文件中的 sys.sql_modules。
sys.sysconfigures	Babelfish 支援限於單個唯讀組態。
sys.syscurconfigs	Babelfish 支援限於單個唯讀組態。
sys.sysprocesses	如需相關資訊，請參閱 Microsoft Transact-SQL 文件中的 sys.sysprocesses。
sys.system_sql_modules	如需相關資訊，請參閱 Microsoft Transact-SQL 文件中的 sys.system_sql_modules。
sys.table_types	如需相關資訊，請參閱 Microsoft Transact-SQL 文件中的 sys.table_types。
sys.tables	所有結構描述中的所有資料表
sys.xml_schema_collections	如需相關資訊，請參閱 Microsoft Transact-SQL 文件中的 sys.xml_schema_collections。

PostgreSQL 實作的系統目錄類似於 SQL Server 物件目錄檢視表。如需完整的系統目錄清單，請參閱 PostgreSQL 文件中的系統目錄。

Babelfish 支援的 DDL 匯出

針對 Babelfish 2.4.0 和 3.1.0 版，Babelfish 支援使用多種工具匯出 DDL。例如，您可以使用 SQL Server Management Studio (SSMS) 的這項功能，為 Babelfish for Aurora PostgreSQL 資料庫的各種

物件生成資料定義指令碼。然後，您可以使用此指令碼中生成的 DDL 命令，在另一個 Babelfish for Aurora PostgreSQL 或 SQL Server 資料庫中建立相同物件。

Babelfish 支援指定版本中，以下物件的 DDL 匯出。

物件清單	2.4.0	3.1.0
使用者資料表	是	是
主索引鍵	是	是
外部索引鍵	是	是
唯一限制條件	是	是
索引	是	是
檢查限制	是	是
檢視	是	是
預存程序	是	是
使用者定義的函數	是	是
資料表值函數	是	是
觸發	是	是
使用者定義的資料類型	否	否
使用者定義的資料表類型	否	否
使用者	否	否
登入	否	否
序列	否	否
角色	否	否

匯出 DDL 的限制

- 使用逃生艙，再以匯出的 DDL 建立物件 – Babelfish 不支援匯出的 DDL 指令碼中所有命令。使用逃生艙，避免在 Babelfish 中從 DDL 命令重新建立物件時造成錯誤。如需有關逃生艙的詳細資訊，請參閱 使用逃生艙管理 Babelfish 錯誤處理。
- 有明確 COLLATE 子句的 CHECK 限制條件之物件 — 具有從 SQL Server 資料庫生成物件的指令碼，具有與 Babelfish 資料庫不同但相等的定序。有些定序是最近的 Windows 定序，例如 sql_latin1_general_cp1_cs_as, sql_latin1_general_cp1251_cs_as 或 latin1_general_cs_as are generated as latin1_general_cs_as。

在 Babelfish for Aurora PostgreSQL 中管理許可和存取控制

在 Babelfish for Aurora PostgreSQL 中，您可以管理資料庫、結構描述和物件的許可和存取控制。下表將概述特定 SQL 命令，這些命令適用於在 Babelfish 中授予許可以實現各種存取控制案例。其中涵蓋可實作的支援使用案例，以及目前不支援案例的解決方法。這可讓您在處理 Babelfish 資料庫時設定適當的許可，以符合安全和合規要求。

支援的使用案例

下表說明 Babelfish 中支援的使用案例。對於每個使用案例，資料表會顯示實現該案例所需的動作和範例 SQL 命令。

使用案例	Action	SQL 命令	說明	Babelfish 版本相容性
允許登入在任何資料庫中執行 SELECT/DML/DDL	將登入新增至 sysadmin 伺服器角色	ALTER SERVER ROLE sysadmin ADD MEMBER `login`	無	所有版本
允許登入在資料庫中執行 SELECT/DML/DDL	讓登入成為資料庫擁有者	ALTER AUTHORIZATION ON DATABASE::`database` TO `login`	資料庫只能有一個擁有者。	3.4 版及更新版本

使用案例	Action	SQL 命令	說明	Babelfish 版本相容性
允許資料庫使用者對結構描述執行 SELECT/DML	將許可授予結構描述上的資料庫使用者	GRANT SELECT/EXECUTE/INSERT/UPDATE/DELETE ON SCHEMA::`schema` TO `user`	無	3.6 版及更新版本，4.2 版及更新版本
允許資料庫使用者對結構描述執行 SELECT/DML	讓資料庫使用者在結構描述建立時成為結構描述擁有者	CREATE SCHEMA `schema` AUTHORIZATION `user`	目前不支援在建立後變更結構描述擁有權。	1.2 版及更新版本
允許資料庫使用者對物件執行 SELECT/DML	將許可授予物件上的資料庫使用者	GRANT SELECT/EXECUTE/INSERT/UPDATE/DELETE ON OBJECT::`object` TO `user`	無	所有版本
允許資料庫使用者在資料庫 (包括捨棄資料庫) 中執行 SELECT/DML/DDL	將使用者新增至 db_owner 固定資料庫角色	ALTER ROLE db_owner ADD MEMBER `user`	只能夠將使用者新增至 db_owner 固定資料庫角色。尚不支援將角色新增至 db_owner 角色。	4.5 版及更新版本，5.1 版及更新版本
允許自訂資料庫角色的使用者或成員只在資料庫中執行 SELECT	將使用者或角色新增至 db_datareader 固定資料庫角色	ALTER ROLE db_datareader ADD MEMBER `user` / `role`	無	4.5 版及更新版本，5.1 版及更新版本

使用案例	Action	SQL 命令	說明	Babelfish 版本相容性
允許自訂資料庫角色的使用者或成員只在資料庫中執行 DML	將使用者或角色新增至 db_datawriter 固定資料庫角色	ALTER ROLE db_datawriter ADD MEMBER `user` / `role`	無	4.5 版及更新版本，5.1 版及更新版本
允許自訂資料庫角色的使用者或成員只在資料庫中執行 DDL	將使用者或角色新增至 db_accessadmin 固定資料庫角色	ALTER ROLE db_accessadmin ADD MEMBER `user` / `role`	無	4.5 版及更新版本，5.1 版及更新版本
允許自訂資料庫角色的使用者或成員僅使用 CREATE/ALTER/DROP 自訂角色、資料庫中物件的 GRANT/REVOKE 許可和/或資料庫中的 CREATE SCHEMA	將使用者或角色新增至 db_securityadmin 固定資料庫角色	ALTER ROLE db_securityadmin ADD MEMBER `user` / `role`	無	4.5 版及更新版本，5.1 版及更新版本

使用案例	Action	SQL 命令	說明	Babelfish 版本相容性
允許自訂資料庫角色的使用者或成員僅對任何使用者進行 CREATE/ALTER/DROP、授予和撤銷資料庫存取權，以及將使用者帳戶對應至登入，和/或在資料庫中進行 CREATE SCHEMA	將使用者或角色新增至 db_accessadmin 固定資料庫角色	ALTER ROLE db_accessadmin ADD MEMBER `user / role`	無	4.5 版及更新版本，5.1 版及更新版本
僅允許登入對任何資料庫進行 CREATE/DROP/ALTER	將登入新增至 dbcreator 固定伺服器角色	ALTER SERVER ROLE dbcreator ADD MEMBER `login`	只能夠變更 dbcreator 登入可以存取的那些資料庫。	4.5 版及更新版本，5.1 版及更新版本
僅允許登入對任何登入進行 CREATE/ALTER/DROP	將登入新增至 securityadmin 固定伺服器角色	ALTER SERVER ROLE securityadmin ADD MEMBER `login`	無	4.5 版及更新版本，5.1 版及更新版本

具有解決方法的不支援使用案例

下表說明 Babelfish 中不支援，但可使用解決方法達成的使用案例。

使用案例	Action	SQL 命令	說明	Babelfish 版本的解決方法相容性
允許使用者對物件/結構描述執行 SELECT/DML，	直接將許可 GRANT 所有其他使用者	GRANT SELECT/EXECUTE/INS	GRANT ... 目前不支援	3.6 版及更新版本，4.2 版及更新版本

使用案例	Action	SQL 命令	說明	Babelfish 版本的解決方法相容性
以及將這些許可 GRANT 其他使用者的選項		ERT/UPDATE/ DELETE ON OBJECT/SC HEMA::`object/sc` TO `user`	WITH GRANT OPTION。	
允許資料庫使用者在資料庫 (包括捨棄資料庫) 中執行 SELECT/DML/DDL	將角色成員新增至 db_owner 固定資料庫角色	ALTER ROLE db_owner ADD MEMBER `user`	目前不支援將角色新增至 db_owner 角色。	4.5 版及更新版本，5.1 版及更新版本

不支援的使用案例

下表說明 Babelfish 中不支援的使用案例。

使用案例	說明
拒絕自訂資料庫角色的使用者或成員在資料庫中執行 SELECT	尚不支援 db_denydatareader 固定資料庫角色
拒絕自訂資料庫角色的使用者或成員在資料庫中執行 DML	目前不支援 db_denydatawriter 固定資料庫角色。
允許登入，以僅對任何資料庫連線進行 KILL	目前不支援 processadmin 固定伺服器角色。
允許登入，以僅新增或移除連結的伺服器	目前不支援 setupadmin 固定伺服器角色。

在升級後解決物件擁有權差異

Babelfish 4.6 版和更新版本以及 5.2 版和更新版本，都包含透過 TDS 端點對物件擁有權處理所做的變更。當您透過 TDS 端點建立新物件時，這些物件現在由結構描述擁有者 (而不是目前使用者) 擁有。當您從較 4.6 或 5.2 更舊的版本升級時，相較於現有物件，此擁有權變更可能會影響新物件的許可行為。

為了解決這些擁有權差異，Babelfish 提供 `sys.generate_alter_ownership_statements()` 函數。此函數會產生 SQL 陳述式，使物件擁有權與結構描述擁有權保持一致。

處理物件擁有權時，請注意下列限制：

- 透過 PostgreSQL 端點獲授予 CREATE 許可的使用者無法透過那些結構描述中的 TDS 端點建立物件。
- 不建議透過 PostgreSQL 端點修改對 T-SQL 物件的許可，這可能會導致不正確的 T-SQL 行為。
- 新舊物件的存取許可可能有所不同，因為其擁有權不相符。例如，請考慮 `sch_own` 擁有的結構描述，其中包含 dbo 擁有的物件。在這種情況下，與升級之後建立的物件 (`sch_own` 擁有) 相比，升級之前建立的物件 (dbo 擁有) 可能具有不同的存取許可。這可能會影響 SELECT 和 INSERT 等操作。

如果資料庫叢集包含在較 4.6 或 5.2 更舊的 Babelfish 版本中建立的物件，請考慮調整其擁有權。

解決物件擁有權差異

1. 使用 PostgreSQL 端點連線至資料庫叢集中的 `babelfish_db` 資料庫。
2. 執行以下命令：

   ```
   SELECT * from sys.generate_alter_ownership_statements();
   ```

   這會產生 SQL 陳述式清單，這些陳述式是為了標準化物件之間的擁有權。
3. 在測試環境執行產生的陳述式，先驗證其效果，再套用到生產環境。

我們建議您執行這些陳述式，以在資料庫中實現一致的物件擁有權模型。

Babelfish for Aurora PostgreSQL 與 SQL Server 之間的差異

Babelfish 是一種不斷發展的 Aurora PostgreSQL 功能，自從 Aurora PostgreSQL 13.4 中的初始產品以來，每個版本中都添加了新功能。Babelfish 旨在使用 TDS 連接埠，透過 T-SQL 方言在 PostgreSQL 的基礎上提供 T-SQL 語意。Babelfish 的每個新版本都會增加功能與 T-SQL 功能和行為更為一致，這些功能與 T-SQL 功能和行為更為一致，如 Babelfish 各版本支援的功能 表所示。為了在使用 Babelfish 時獲得最佳結果，我們建議您了解最新版本 SQL 伺服器支援的 T-SQL 與 Babelfish 之間目前存在的差異。如需詳細資訊，請參閱 Babelfish 中的 T-SQL 差異。

除了 Babelfish 和 SQL 伺服器支援的 T-SQL 之間的差異之外，您可能還需要考慮在 Aurora PostgreSQL 資料庫叢集的環境中，Babelfish 和 PostgreSQL 之間的相互操作性問題。如前所

述，Babelfish 使用 TDS 連接埠，透過 T-SQL 方言在 PostgreSQL 的基礎上支援 T-SQL 語意。同時，Babblefish 資料庫也可以使用 PostgreSQL SQL 陳述式，透過標準 PostgreSQL 通訊埠存取。如果您考慮在生產部署中同時使用 PostgreSQL 和 Babelfish 功能，則需要注意結構描述名稱、識別碼、許可、交易語意、多個結果集、預設定序等之間潛在的相互操作性問題。簡單來說，如果 Babelfish 環境中出現 PostgreSQL 陳述式或 PostgreSQL 存取，當 Babelfish 發行新版本時，PostgreSQL 和 Babelfish 之間可能會發生干擾，並且可能會影響語法、語意和相容性。如需有關所有考量事項的完整資訊和指引，請參閱 Babelfish for PostgreSQL 文件中的 Babelfish 相互操作性指南。

> **Note**
>
> 在同一個應用程式環境中同時使用 PostgreSQL 原生功能和 Babelfish 功能之前，強烈建議您考慮 Babelfish for PostgreSQL 文件中 Babelfish 相互操作性指南中討論的問題。只有當您打算在與 Babelfish 相同的應用程式環境中使用 PostgreSQL 資料庫執行個體時，才會發生這些相互操作性問題 (Aurora PostgreSQL 和 Babelfish)。

主題

- Babelfish 傾印和還原
- Babelfish 中的 T-SQL 差異
- Babelfish 中的交易隔離層級

Babelfish 傾印和還原

從 4.0.0 和 3.4.0 版開始，Babelfish 使用者現在可以利用傾印和還原公用程式來備份和還原其資料庫。如需詳細資訊，請參閱 Babelfish 傾印和還原。此功能是根據 PostgreSQL 傾印和還原公用程式所建置。如需詳細資訊，請參閱 pg_dump 和參閱 pg_restore。為了有效地在 Babelfish 中使用此功能，您需要使用專門針對 Babelfish 調整以 PostgreSQL 為基礎的工具。Babelfish 的備份和還原功能與 SQL Server 的備份和還原功能明顯不同。如需這些差異的詳細資訊，請參閱傾印和還原功能差異：Babelfish 和 SQL Server。Babelfish for Aurora PostgreSQL 為 Amazon Aurora PostgreSQL 資料庫叢集的備份和還原提供額外功能。如需詳細資訊，請參閱備份與還原 Amazon Aurora 資料庫叢集。

Babelfish 中的 T-SQL 差異

您可以在以下內容找到 Babelfish 目前版本支援的 T-SQL 功能的表格，其中包括一些有關行為與 SQL Server 不同的註釋。

如需各種版本的支援詳情，請參閱 Babelfish 各版本支援的功能。如需目前不支援的功能詳情，請參閱 Babelfish 中不支援的功能。

Babelfish 可與 Aurora PostgreSQL 相容版本一起使用。如需 Babelfish 版本的更多資訊，請參閱 Aurora PostgreSQL 版本備註。

功能或語法	行為或差異的描述
\ (行連續字元)	目前不支援字元和十六進位字串的行延續字元 (換行之前的反斜線)。針對字元字串，反斜線換行會被解讀為字串中的字元。針對十六進位字串，反斜槓換行會導致語法錯誤。
@@version	`@@version` 傳回的值與 SQL Server 傳回的值格式略有不同。依賴 `@@version` 格式的程式碼可能無法正常運作。
彙總函數	支援部分彙總函數 (支援 AVG、COUNT、COUNT_BIG、GROUPING、MAX、MIN、STRING_AGG 和 SUM)。如需不支援的彙總函數清單，請參閱 不支援的函數。
ALTER TABLE	僅支援新增或捨棄單個資料欄或限制。
ALTER TABLE..ALTER COLUMN	目前無法指定 NULL 和 NOT NULL。若要變更資料欄的可 null 性，請使用 PostgreSQL 陳述式 ALTER TABLE..{SET|DROP} NOT NULL。

AT TIME ZONE	從日光節約時間 (DST) 轉換到標準時間期間，系統會使用標準時間偏移顯示重疊期間。為釐清此點，請思考下列範例：
	`SELECT CONVERT(DATETIME2(0), '2022-10-30T02:00:00', 126) AT TIME ZONE 'Central European Standard Time';`
	`GO;`
	`Result: 2022-10-30 02:00:00 +01:00`

沒有資料欄別名的空白資料欄名稱	`sqlcmd` 和 `psql` 公用程式以不同方式處理名稱空白的資料欄： • SQL Server `sqlcmd` 傳回空白資料欄名稱。 • PostgreSQL `psql` 傳回產生的資料欄別名。

功能或語法	行為或差異的描述
CHECKSUM 函數	Babelfish 和 SQL 伺服器將不同的雜湊演算法使用於 CHECKSUM 函數。其結果是，由 Babelfish 中的 CHECKSUM 函數產生的雜湊值可能與 SQL Server 中的 CHECKSUM 函數產生的雜湊值不同。
資料欄預設	建立資料欄預設時會忽略限制條件名稱。若要捨棄資料欄預設，請使用下列語法：`ALTER TABLE...ALTER COLUMN..DROP DEFAULT...`
Constraint_name	在 SQL Server 中，限制條件名稱在資料表所屬的結構描述中必須是唯一的。不過，在 Babelfish 中，這僅適用於 PRIMARY KEY 和 UNIQUE 限制條件。其他類型的限制條件不受此限制。
限制條件	PostgreSQL 不支援啟用和停用個別限制條件。陳述式會被忽略，並發出警告。
條件限制搭配 IGNORE_DUP_KEY	建立的條件限制沒有此內容。
CREATE、ALTER、DROP SERVER ROLE	ALTER SERVER ROLE 僅支援 `sysadmin`。不支援其他所有語法。 Babelfish 中的 T-SQL 使用者，可體驗類似 SQL Server 的登入 (伺服器主體)、資料庫和資料庫使用者 (資料庫主體) 等概念。
CREATE、ALTER LOGIN 子句支援的語法有限	CREATE LOGIN... 支援 PASSWORD 子句、...DEFAULT_DATABASE 子句及 ...DEFAULT_LANGUAGE 子句。ALTER LOGIN... 支援 PASSWORD 子句，而不是 ALTER LOGIN... 不支援 OLD_PASSWORD 子句。只有以 sysadmin 成員身分登入才能修改密碼。
CREATE DATABASE 區分大小寫定序	CREATE DATABASE 庫陳述式不支援區分大小寫定序。

功能或語法	行為或差異的描述
CREATE DATABASE 關鍵字和子句	不支援 COLLATE 和 CONTAINMENT=NONE 以外的選項。接受 COLLATE 子句，一律設定為 `babelfishpg_tsql.server_collation_name` 的值。
CREATE SCHEMA... 支援子句	您可以使用 CREATE SCHEMA 命令來建立空的結構描述。使用其他命令來建立結構描述物件。
資料庫 ID 值在 Babelfish 上不同	master 和 tempdb 資料庫不會是資料庫 ID 1 和 2。
支援 FORMAT 日期類型函數，但有以下限制	不支援單一字元子午線。 SQL 伺服器中的 "yyy" 格式對於 1000 以上的年份傳回 4 位數，但對於其他則只傳回 3 位數。 不支援 "g" 和 "R" 格式 "vi-VN" 地區設定轉換略有不同。
超過 63 個字元的識別符	PostgreSQL 支援最多 63 個字元的識別符。Babelfish 會轉換超過 63 個字元的識別符，轉換後的名稱包含原始名稱的雜湊。例如，一個建立為 "AB(ABC123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890" 的資料表可能會被轉換為 "ABC123456789012345678901234567890123456789012345678901234567890"。
IDENTITY 資料欄支援	`IDENTITY` 資料欄支援資料類型 `tinyint`、`smallint`、`int`、`bigint`、`numeric` 及 `decimal`。 SQL Server 在 IDENTITY 資料欄中支援資料類型 `numeric` 和 `decimal` 精確到 38 位數。 PostgreSQL 在 IDENTITY 資料欄中支援資料類型 `numeric` 和 `decimal` 精確到 19 位數。
索引搭配 IGNORE_DUP_KEY	如果語法建立的索引包含 INNORE_DUP_KEY，則視為省略此內容來建立索引。

功能或語法	行為或差異的描述
超過 32 個資料欄的索引	索引不能包含超過 32 個資料欄。在 PostgreSQL 中，包含的索引資料欄計入上限，但在 SQL Server 中不會這樣。
索引 (叢集化)	視為已指定 NONCLUSTERED 來建立叢集索引。
索引子句	忽略下列子句：FILLFACTOR、ALLOW_PAGE_LOCKS、ALLOW_ROW_LOCKS、PAD_INDEX、STATISTICS_NORECOMPUTE、OPTIMIZE_FOR_SEQUENTIAL_KEY、SORT_IN_TEMPDB、DROP_EXISTING、ONLINE、COMPRESSION_DELAY、MAXDOP 和 DATA_COMPRESSION
JSON 支援	不能保證名稱-值對的順序。但是陣列類型仍然不受影響。
LOGIN 物件	支援 LOGIN 物件的所有選項，以下除外：PASSWORD、DEFAULT_DATABASE、DEFAULT_LANGUAGE、ENABLE，DISABLE。
NEWSEQUENTIALID 函數	實作成 NEWID；不保證循序行為。呼叫 `NEWSEQUENTIALID` 時，PostgreSQL 會產生新的 GUID 值。
支援 OUTPUT 子句，但有以下限制	不支援同一個 DML 查詢中有 OUTPUT 和 OUTPUT INTO。不支援 OUTPUT 子句中的 UPDATE 或 DELETE 操作參考非目標資料表。OUTPUT... 不支援同一個查詢中有 DELETED *、INSERTED *。
程序或函數參數限制	Babelfish 支援一個程序或函數最多有 100 個參數。
ROWGUIDCOL	目前忽略此子句。查詢參考 `$GUIDGOL` 會導致語法錯誤。
SEQUENCE 物件支援	SEQUENCE 物件支援資料類型 tinyint、smallint、int、bigint、numeric 及 decimal。 Aurora PostgreSQL 在 SEQUENCE 中支援資料類型 numeric 和 and decimal 精確到 19 位數。

功能或語法	行為或差異的描述
伺服器層級角色	支援 sysadmin 伺服器層級角色。不支援其他伺服器層級角色 (sysadmin 除外)。
db_owner 除外的資料庫層級角色	支援 db_owner 資料庫層級角色和使用者定義的資料庫層級角色。不支援 db_owner 除外的資料庫層級角色。
SQL 關鍵字 SPARSE	接受和忽略關鍵字 SPARSE。
SQL 關鍵字子句 ON filegroup	目前忽略此子句。
索引和限制條件的 SQL 關鍵字 CLUSTERED 和 NONCLUSTERED	Babelfish 接受和忽略 CLUSTERED 和 NONCLUSTERED 關鍵字。
sysdatabases.cmptlevel	sysdatabases.cmptlevel 一律設定為 120。
重新啟動時不會重新初始化 tempdb	資料庫重新啟動時，不會移除 tempdb 中建立的永久物件 (例如資料表和程序)。
TEXTIMAGE_ON filegroup	Babelfish 會忽略 TEXTIMAGE_ON *filegroup* 子句。
時間精確度	Babelfish 支援 6 位數精確度的小數秒。此行為尚無不良影響。
交易隔離層級	READUNCOMMITTED 和 READCOMMITTED 視為相同。
虛擬計算資料欄 (非持久性)	虛擬計算資料欄建立為持久性。
沒有 SCHEMABINDING 子句	函數、程序、觸發程序或檢視表中不支援此子句。視為未指定 WITH SCHEMABINDING 來建立物件。

Babelfish 中的交易隔離層級

Babelfish 支援交易隔離層級 `READ UNCOMMITTED`、`READ COMMITTED` 和 `SNAPSHOT`。從 Babelfish 3.4 版開始，支援其他隔離層級 `REPEATABLE READ` 和 `SERIALIZABLE`。Babelfish 中的所有隔離層級都受 PostgreSQL 中對應隔離層級的行為支援。SQL Server 和 Babelfish 使用不同的基礎機制來實作交易隔離層級 (封鎖並行存取、交易保留的鎖定、錯誤處理等)。此外，並行存取可能適用於不同工作負載的方式有一些細微差異。如需此 PostgreSQL 行為的詳細資訊，請參閱交易隔離。

主題

- 交易隔離層級概觀
- 設定交易隔離層級
- 啟用或停用交易隔離層級
- 比較 Babelfish 和 SQL Server 隔離層級

交易隔離層級概觀

原始 SQL Server 交易隔離層級是根據悲觀鎖定，其中只有一個資料複本存在，而查詢必須在存取列之類的資源之前將其鎖定。稍後，推出了 `READ COMMITTED` 隔離層級的變化。這使得能夠使用列版本透過非封鎖存取，在讀取器和寫入器之間提供更好的並行。此外，也可以使用名為 `SNAPSHOT` 的新隔離層級。其還使用列版本來提供比 `REPEATABLE READ` 隔離層級更好的並行，方法是避免在讀取資料時保留直至交易結束前的共用鎖定。

與 SQL Server 不同，Babelfish 中的所有交易隔離層級都是以樂觀鎖定 (MVCC) 為基礎。無論基礎資料的目前狀態為何，每個交易都會在陳述式 (`READ COMMITTED`) 開頭或交易 (`REPEATABLE READ`、`SERIALIZABLE`) 開頭看到資料的快照。因此，Babelfish 中並行交易的執行行為可能與 SQL Server 不同。

例如，假設具有隔離層級 `SERIALIZABLE` 的交易最初在 SQL Server 中遭到封鎖，但稍後成功。由於序列化與讀取或更新相同列的並行交易發生衝突，最終可能會在 Babelfish 中失敗。在某些情況下，與 SQL Server 相比，執行多個並行交易會在 Babelfish 中產生不同的最終結果。使用隔離層級的應用程式，應針對並行案例進行徹底測試。

SQL Server 中的隔離層級	Babelfish 隔離層級	PostgreSQL 隔離層級	說明
`READ UNCOMMITTED`	`READ UNCOMMITTED`	`READ UNCOMMITTED`	`READ UNCOMMITTED` 與 Babelfish 或

SQL Server 中的隔離層級	Babelfish 隔離層級	PostgreSQL 隔離層級	說明
			PostgreSQL 中的 READ COMMITTED 相同
READ COMMITTED	READ COMMITTED	READ COMMITTED	SQL Server READ COMMITTED 以悲觀鎖定為基礎，而 Babelfish READ COMMITTED 則以快照 (MVCC) 為基礎。
READ COMMITTED SNAPSHOT	READ COMMITTED	READ COMMITTED	兩者都是以快照 (MVCC) 為基礎，但不完全相同。
SNAPSHOT	SNAPSHOT	REPEATABLE READ	完全相同。
REPEATABLE READ	REPEATABLE READ	REPEATABLE READ	SQL Server REPEATABLE READ 以悲觀鎖定為基礎，而 Babelfish REPEATABLE READ 則以快照 (MVCC) 為基礎。
SERIALIZABLE	SERIALIZABLE	SERIALIZABLE	SQL Server SERIALIZABLE 是悲觀隔離，而 Babelfish SERIALIZABLE 是以快照 (MVCC) 為基礎。

> **Note**
>
> 資料表提示目前不受支援，其行為是透過使用 Babelfish 預先定義的逃生艙 `escape_hatch_table_hints` 來控制。

設定交易隔離層級

使用下列命令來設定交易隔離層級：

Example

```
SET TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ |
 SNAPSHOT | SERIALIZABLE }
```

啟用或停用交易隔離層級

Babelfish 預設會停用交易隔離層級 `REPEATABLE READ` 和 `SERIALIZABLE`，您必須使用 `sp_babelfish_configure`，將 `babelfishpg_tsql.isolation_level_serializable` 或 `babelfishpg_tsql.isolation_level_repeatable_read` 逃生艙設為 `pg_isolation` 以明確將其啟用。如需更多詳細資訊，請參閱 使用逃生艙管理 Babelfish 錯誤處理。

以下範例示範如何透過設定各自的逃生艙，啟用或停用目前工作階段中 `REPEATABLE READ` 和 `SERIALIZABLE` 的使用。選擇性地包含 `server` 參數，以設定目前工作階段以及所有後續新工作階段的逃生艙。

僅在目前工作階段中啟用 `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ` 的使用。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'pg_isolation'
```

在目前工作階段和所有後續新工作階段中啟用 `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ` 的使用。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'pg_isolation',
 'server'
```

若要停用目前工作階段和後續新工作階段中 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ 的使用。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'off', 'server'
```

僅在目前工作階段中啟用 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE 的使用。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'pg_isolation'
```

在目前工作階段和所有後續新工作階段中啟用 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE 的使用。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'pg_isolation', 'server'
```

若要停用目前工作階段和後續新工作階段中 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE 的使用。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'off', 'server'
```

比較 Babelfish 和 SQL Server 隔離層級

以下是一些範例，示範 SQL Server 和 Babelfish 如何實作 ANSI 隔離層級的細微差異。

> **Note**
>
> - Babelfish 中的隔離層級 `REPEATABLE READ`和 `SNAPSHOT` 是相同的。
> - Babelfish 中的隔離層級 `READ UNCOMMITTED`和 `READ COMMITTED` 是相同的。

下列範例示範如何為以下提及所有範例建立基礎資料表：

```
CREATE TABLE employee (
    id sys.INT NOT NULL PRIMARY KEY,
    name sys.VARCHAR(255)NOT NULL,
    age sys.INT NOT NULL
);
INSERT INTO employee (id, name, age) VALUES (1, 'A', 10);
INSERT INTO employee (id, name, age) VALUES (2, 'B', 20);
INSERT INTO employee (id, name, age) VALUES (3, 'C', 30);
```

主題

- Babelfish READ UNCOMMITTED 相較於 SQL Server READ UNCOMMITTED 隔離層級
- Babelfish READ COMMITTED 相較於 SQL Server READ COMMITTED 隔離層級
- Babelfish READ COMMITTED 相較於 SQL Server READ COMMITTED SNAPSHOT 隔離層級
- Babelfish REPEATABLE READ 相較於 SQL Server REPEATABLE READ 隔離層級
- Babelfish SERIALIZABLE 相較於 SQL Server SERIALIZABLE 隔離層級

Babelfish **`READ UNCOMMITTED`** 相較於 SQL Server **`READ UNCOMMITTED`** 隔離層級

下表提供執行並行交易時，變更讀取的詳細資訊。與 Babelfish 實作相比，其顯示在 SQL Server 中使用 `READ UNCOMMITTED` 隔離層級時觀察到的結果。

交易 1	交易 2	SQL Server **`READ UNCOMMITTED`**	Babelfish **`READ UNCOMMITTED`**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	無	無
`SET TRANSACTION ISOLATION LEVEL`	`SET TRANSACTION ISOLATION LEVEL`	無	無

交易 1	交易 2	SQL Server **READ UNCOMMITTED**	Babelfish **READ UNCOMMITTED**
`READ UNCOMMITTED;`	`READ UNCOMMITTED;`		
交易閒置	`UPDATE employee SET age=0;`	更新成功。	更新成功。
交易閒置	`INSERT INTO employee VALUES (4, 'D', 40);`	插入成功。	插入成功。
`SELECT * FROM employee;`	交易閒置	交易 1 可以看到來自交易 2 的未遞交變更。	與 Babelfish 中的 `READ COMMITTED` 相同。交易 1 看不到來自交易 2 的未遞交變更。
交易閒置	`COMMIT`	無	無
`SELECT * FROM employee;`	交易閒置	查看交易 2 遞交的變更。	查看交易 2 遞交的變更。

Babelfish **READ COMMITTED** 相較於 SQL Server **READ COMMITTED** 隔離層級

下表提供執行並行交易時的讀寫封鎖行為詳細資訊。與 Babelfish 實作相比，其顯示在 SQL Server 中使用 `READ COMMITTED` 隔離層級時觀察到的結果。

交易 1	交易 2	SQL Server **READ COMMITTED**	Babelfish **READ COMMITTED**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	無	無
`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	無	無

交易 1	交易 2	SQL Server **READ COMMITTED**	Babelfish **READ COMMITTED**
`SELECT * FROM employee;`	交易閒置	無	無
交易閒置	`UPDATE employee SET age=100 WHERE id = 1;`	更新成功。	更新成功。
`UPDATE employee SET age = 0 WHERE age IN (SELECT MAX(age) FROM employee);`	交易閒置	步驟封鎖，直到交易 2 遞交為止。	交易 2 變更尚不可見。更新 id=3 的列。
交易閒置	`COMMIT`	交易 2 已成功遞交。交易 1 現在已解除封鎖，並查看來自交易 2 的更新。	交易 2 已成功遞交。
`SELECT * FROM employee;`	交易閒置	交易 1 更新 id = 1 的列。	交易 1 更新 id = 3 的列。

Babelfish **READ COMMITTED** 相較於 SQL Server **READ COMMITTED SNAPSHOT** 隔離層級

下表提供的詳細資訊，與執行並行交易時新插入之列的封鎖行為有關。與 `READ COMMITTED` Babelfish 實作相比，其顯示在 SQL Server 中使用 `READ COMMITTED SNAPSHOT` 隔離層級時觀察到的結果。

交易 1	交易 2	SQL Server **READ COMMITTED SNAPSHOT**	Babelfish **READ COMMITTED**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	無	無

交易 1	交易 2	SQL Server **READ COMMITTED SNAPSHOT**	Babelfish **READ COMMITTED**
`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	無	無
`INSERT INTO employee VALUES (4, 'D', 40);`	交易閒置	無	無
交易閒置	`UPDATE employee SET age = 99;`	步驟封鎖，直到交易 1 遞交為止。插入的列遭交易 1 鎖定。	已更新三列。新插入的列尚不可見。
`COMMIT`	交易閒置	遞交成功。交易 2 現已解除封鎖。	遞交成功。
交易閒置	`SELECT * FROM employee;`	所有 4 列都具有存留期=99。	id = 4 之列的存留期值為 40，因為更新查詢期間交易 2 看不到此值。其他列會更新為存留期=99。

Babelfish **REPEATABLE READ** 相較於 SQL Server **REPEATABLE READ** 隔離層級

下表提供執行並行交易時的讀寫封鎖行為詳細資訊。與 REPEATABLE READ Babelfish 實作相比，其顯示在 SQL Server 中使用 REPEATABLE READ 隔離層級時觀察到的結果。

交易 1	交易 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	無	無
`SET TRANSACTION ISOLATION`	`SET TRANSACTION ISOLATION`	無	無

交易 1	交易 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
`LEVEL REPEATABLE READ;`	`LEVEL REPEATABLE READ;`		
`SELECT * FROM employee;`	交易閒置	無	無
`UPDATE employee SET name='A_TXN1' WHERE id=1;`	交易閒置	無	無
交易閒置	`SELECT * FROM employee WHERE id != 1;`	無	無
交易閒置	`SELECT * FROM employee;`	交易 2 遭封鎖，直到交易 1 遞交為止。	交易 2 會正常繼續執行。
`COMMIT`	交易閒置	無	無
交易閒置	`SELECT * FROM employee;`	交易 1 的更新是可見的。	交易 1 的更新是不可見的。
`COMMIT`	交易閒置	無	無
交易閒置	`SELECT * FROM employee;`	會看到來自交易 1 的更新。	會看到來自交易 1 的更新。

下表提供執行並行交易時的寫入-寫入封鎖行為詳細資訊。與 `REPEATABLE READ` Babelfish 實作相比，其顯示在 SQL Server 中使用 `REPEATABLE READ` 隔離層級時觀察到的結果。

交易 1	交易 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	無	無
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	無	無
`UPDATE employee SET name='A_TXN1' WHERE id=1;`	交易閒置	無	無
交易閒置	`UPDATE employee SET name='A_TXN2' WHERE id=1;`	交易 2 遭封鎖。	交易 2 遭封鎖。
`COMMIT`	交易閒置	遞交成功，交易 2 已解除封鎖。	遞交成功，交易 2 失敗，但由於並行更新，錯誤無法序列化存取。
交易閒置	`COMMIT`	遞交成功。	交易 2 已中止。
交易閒置	`SELECT * FROM employee;`	id=1 的列名稱為='A_TX2'。	id=1 的列名稱為='A_TX1'。

下表提供執行並行交易時幽靈讀取行為的詳細資訊。與 `REPEATABLE READ` Babelfish 實作相比，其顯示在 SQL Server 中使用 `REPEATABLE READ` 隔離層級時觀察到的結果。

交易 1	交易 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	無	無
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	無	無
`SELECT * FROM employee;`	交易閒置	無	無
交易閒置	`INSERT INTO employee VALUES (4, 'NewRowName', 20);`	交易 2 繼續進行，而沒有任何封鎖。	交易 2 繼續進行，而沒有任何封鎖。
交易閒置	`SELECT * FROM employee;`	新插入的列是可見的。	新插入的列是可見的。
交易閒置	`COMMIT`	無	無
`SELECT * FROM employee;`	交易閒置	交易 2 插入的新列是可見的。	交易 2 插入的新列是不可見的。
`COMMIT`	交易閒置	無	無
`SELECT * FROM employee;`	交易閒置	新插入的列是可見的。	新插入的列是可見的。

下表提供執行並行交易時的詳細資訊，以及與 `REPEATABLE READ` Babelfish 實作相比，在 SQL Server 中使用 `REPEATABLE READ` 隔離層級時的不同最終結果。

交易 1	交易 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	無	無
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	無	無
`UPDATE employee SET age = 100 WHERE age IN (SELECT MIN(age) FROM employee);`	交易閒置	交易 1 會更新具有 id 1 的列。	交易 1 會更新具有 id 1 的列。
交易閒置	`UPDATE employee SET age = 0 WHERE age IN (SELECT MAX(age) FROM employee);`	SELECT 陳述式嘗試讀取交易 1 中由 UPDATE 查詢鎖定的列，因此會封鎖交易 2。	交易 2 會繼續執行，而不會進行任何封鎖，因為讀取從未遭到封鎖，SELECT 陳述式會執行，最後更新 id = 3 的列，因為交易 1 變更尚不可見。
交易閒置	`SELECT * FROM employee;`	此步驟會在交易 1 遞交之後執行。id = 1 的列會由上一步的交易 2 更新，於此處是可見的。	交易 2 會更新 id = 3 的列。
`COMMIT`	交易閒置	交易 2 現已解除封鎖。	遞交成功。
交易閒置	`COMMIT`	無	無

交易 1	交易 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
`SELECT * FROM employee;`	交易閒置	兩個交易都會在 id = 1 的列上執行更新。	交易 1 和 2 會更新不同的列。

Babelfish **SERIALIZABLE** 相較於 SQL Server **SERIALIZABLE** 隔離層級

下表提供執行並行交易時範圍鎖定的詳細資訊。與 SERIALIZABLE Babelfish 實作相比，其顯示在 SQL Server 中使用 SERIALIZABLE 隔離層級時觀察到的結果。

交易 1	交易 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	無	無
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	無	無
`SELECT * FROM employee;`	交易閒置	無	無
交易閒置	`INSERT INTO employee VALUES (4, 'D', 35);`	交易 2 遭封鎖，直到交易 1 遞交為止。	交易 2 繼續進行，而沒有任何封鎖。
交易閒置	`SELECT * FROM employee;`	無	無
`COMMIT`	交易閒置	交易 1 已成功遞交。交易 2 現已解除封鎖。	交易 1 已成功遞交。
交易閒置	`COMMIT`	無	無

交易 1	交易 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
`SELECT * FROM employee;`	交易閒置	新插入的列是可見的。	新插入的列是可見的。

下表提供執行並行交易時的詳細資訊，以及與 SERIALIZABLE Babelfish 實作相比，在 SQL Server 中使用 SERIALIZABLE 隔離層級時的不同最終結果。

交易 1	交易 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	無	無
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	無	無
交易閒置	`INSERT INTO employee VALUES (4, 'D', 40);`	無	無
`UPDATE employee SET age =99 WHERE id = 4;`	交易閒置	交易 1 遭封鎖，直到交易 2 遞交為止。	交易 1 繼續進行，而沒有任何封鎖。
交易閒置	`COMMIT`	交易 2 已成功遞交。交易 1 現已解除封鎖。	交易 2 已成功遞交。
`COMMIT`	交易閒置	無	無
`SELECT * FROM employee;`	交易閒置	新插入的列是可見的，存留期值 = 99。	新插入的列是可見的，存留期值 = 40。

當您 INSERT 進入具有唯一限制條件的資料表時，下表會提供詳細資訊。與 SERIALIZABLE Babelfish 實作相比，其顯示在 SQL Server 中使用 SERIALIZABLE 隔離層級時觀察到的結果。

交易 1	交易 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	無	無
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	無	無
交易閒置	`INSERT INTO employee VALUES (4, 'D', 40);`	無	無
`INSERT INTO employee VALUES ((SELECT MAX(id)+1 FROM employee), 'E', 50);`	交易閒置	交易 1 遭封鎖，直到交易 2 遞交為止。	交易 1 遭封鎖，直到交易 2 遞交為止。
交易閒置	`COMMIT`	交易 2 已成功遞交。交易 1 現已解除封鎖。	交易 2 已成功遞交。因錯誤重複索引鍵值而中止的交易 1 違反唯一限制條件。
`COMMIT`	交易閒置	交易 1 已成功遞交。	交易 1 遞交失敗，因為交易之間的讀取或寫入相依性而無法序列化存取。
`SELECT * FROM employee;`	交易閒置	列 (5、'E'、50) 已插入。	只有 4 個列存在。

在 Babelfish 中，以可序列化隔離層級執行的並行交易，若其執行結果與那些交易執行結果所有可能的序列 (一次一個) 不一致，則會因可序列化異常錯誤而失敗。

下表提供執行並行交易時，序列化異常的詳細資訊。與 SERIALIZABLE Babelfish 實作相比，其顯示在 SQL Server 中使用 SERIALIZABLE 隔離層級時觀察到的結果。

交易 1	交易 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	無	無
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	無	無
`SELECT * FROM employee;`	交易閒置	無	無
`UPDATE employee SET age=5 WHERE age=10;`	交易閒置	無	無
交易閒置	`SELECT * FROM employee;`	交易 2 遭封鎖，直到交易 1 遞交為止。	交易 2 繼續進行，而沒有任何封鎖。
交易閒置	`UPDATE employee SET age=35 WHERE age=30;`	無	無
`COMMIT`	交易閒置	交易 1 已成功遞交。	交易 1 會先遞交，並且能夠成功遞交。
交易閒置	`COMMIT`	交易 2 已成功遞交。	交易 2 遞交失敗並出現序列化錯誤，整個交易已復原。重試交易 2。

交易 1	交易 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
`SELECT * FROM employee;`	交易閒置	兩個交易的變更都是可見的。	交易 2 已復原。只會看到交易 1 變更。

在 Babelfish 中，只有當所有並行交易都在隔離層級 SERIALIZABLE 執行時，才可能發生序列化異常。在下表中，讓我們以上述範例為例，但將交易 2 改設為隔離層級 REPEATABLE READ。

交易 1	交易 2	SQL Server 隔離層級	Babelfish 隔離層級
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	無	無
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	無	無
`SELECT * FROM employee;`	交易閒置	無	無
`UPDATE employee SET age=5 WHERE age=10;`	交易閒置	無	無
交易閒置	`SELECT * FROM employee;`	交易 2 遭封鎖，直到交易 1 遞交為止。	交易 2 繼續進行，而沒有任何封鎖。
交易閒置	`UPDATE employee SET age=35 WHERE age=30;`	無	無
`COMMIT`	交易閒置	交易 1 已成功遞交。	交易 1 已成功遞交。
交易閒置	`COMMIT`	交易 2 已成功遞交。	交易 2 已成功遞交。

交易 1	交易 2	SQL Server 隔離層級	Babelfish 隔離層級
`SELECT * FROM employee;`	交易閒置	兩個交易的變更都是可見的。	兩個交易的變更都是可見的。

使用具有限制實作的 Babelfish 功能

Babelfish 的每個新版本都會增加對功能的支援，這些功能與 T-SQL 功能和行為更為一致。儘管如此，目前的實作中仍存在一些不受支援的功能和差異。您可以在以下內容找到有關 Babelfish 和 T-SQL 之間功能差異的資訊，以及一些解決方法或使用說明。

從 Babelfish 1.2.0 版起，以下功能目前具有限制的實作：

- SQL 伺服器目錄 (系統檢視) — 目錄 `sys.sysconfigures`、`sys.syscurconfigs` 以及 `sys.configurations` 僅支持單一唯讀組態。目前不支援 `sp_configure`。如需 Babelfish 實作的其他 SQL Server 檢視的詳細資訊，請參閱 從 Babelfish 系統目錄取得資訊。
- GRANT 許可 — 支持 GRANT…TO PUBLIC，但目前不支援 GRANT..TO PUBLIC WITH GRANT OPTION。
- SQL Server擁有權鏈和許可機制限制 — 在 Babelfish 中，SQL Server 擁有權鏈適用於檢視，但不適用於預存程序。這代表必須向程序授予對與呼叫程序之相同擁有者所擁有的其他物件的明確存取權。在 SQL Server 中，授予呼叫者在程序上的 EXECUTE 權限足以呼叫同一擁有者所擁有的其他物件。在 Babelfish 中，呼叫者還必須被授予對程序所存取之物件的權限。
- 解析度不合格 (無結構描述名稱) 的物件參考 — 當 SQL 物件 (程序、檢視、函式或觸發程序) 參考物件而不使用結構描述名稱限定物件時，SQL Server 會使用發生參考的 SQL 物件的結構描述名稱來解析該物件的結構描述名稱。目前，Babelfish 使用執行程序的資料庫使用者的預設結構描述，以不同方式解決此問題。
- 預設結構描述變更、工作階段和連接 — 如果使用者將預設結構描述變更為 `ALTER USER...WITH DEFAULT SCHEMA`，此變更會立即在該工作階段中生效。但是，對於屬於同一使用者的其他目前連接的工作階段，計時會有所不同，如下所示：
  - 如果是 SQL Server：— 此變更將立即在此使用者的所有其他連接中生效。
  - 如果是 Babelfish：— 此變更僅對此使用者的新連接生效。
- ROWVERSION 和 TIMESTAMP 資料類型實作和逃生艙設定 — 現在 Babelfish 已支援 ROWVERSION 和 TIMESTAMP 資料類型。若要在 Babelfish 中使用 ROWVERSION 和 TIMESTAMP，您必須將逃生艙的設定 `babelfishpg_tsql.escape_hatch_rowversion` 從預

設值 (嚴格) 變更為 `ignore`。ROWVERSION 和 TIMESTAMP 資料類型的 Babelfish 實作在語義上與 SQL Server 大多相同，但有以下例外：

- 內建 @@DBTS 函數的行為與 SQL Server 類似，但有些微差異。Babelfish 不會傳回 `SELECT @@DBTS` 其上次使用的值，而是產生新的時間戳記，這是因為基礎 PostgreSQL 資料庫引擎及其多版本並行控制 (MVCC) 實作之故。
- 在 SQL Server 中，每個插入或更新的行都會獲得唯一的 ROWVERSION/TIMESTAMP 值。在 Babelfish 中，由同一陳述式更新的每個插入行都會被指派相同的 ROWVERSION/TIMESTAMP 值。

  例如，當 UPDATE 陳述式或 INSERT-SELECT 陳述式影響多行時，在 SQL Server 中，受影響的行在其 ROWVERSION/TIMESTAMP 列中都具有不同的值。在 Babelfish (PostgreSQL) 中，行具有相同的值。
- 在 SQL Server 中，當您使用 SELECT-INTO 建立新資料表時，可以將明確值 (如 NULL) 轉換為要建立的 ROWVERSION/TIMESTAMP 列。當您在 Babelfish 中做同樣的事情時，Babelfish 會將一個實際的 ROWVERSION/TIMESTAMP 值指派給新資料表中的每一行。

ROWVERSION/TIMESTAMP 資料類型中的這些小差異不會對在 Babelfish 上執行的應用程式產生負面影響。

- TOP N PERCENT 子句 – Babelfish 支援具有某些限制的 TOP N PERCENT 子句。不支援具有 TOP N PERCENT 的 UPDATE、DELETE 和 INSERT 操作時，支援 SELECT 操作。也不支援 TOP 子句中的 WITH TIES 選項和子查詢。當表達式值超過 100 時，行為會有所不同：
  - 針對 SQL Server – 擲回錯誤。
  - 對於 Babelfish – 將值視為有效並傳回結果。
- 結構描述建立、擁有權和權限 — 在由非 DBO 使用者 (使用 `CREATE SCHEMA` *`schema name`* `AUTHORIZATION` *`user name`*) 擁有的結構描述中建立和存取物件的許可，不同於 SQL Server 和 Babelfish 非 DBO 使用者，如下表所示：

擁有該結構描述的資料庫使用者 (非 DBO) 可執行下列操作：	SQL Server	Babelfish
在結構描述中建立物件，而不需要 DBO 的額外授權？	否	是
存取 DBO 在結構描述中建立的物件，而不需要額外授權？	是	否

擁有該結構描述的資料庫使用者 (非 DBO) 可執行下列操作：	SQL Server	Babelfish

- CREATE OR ALTER VIEW / ALTER VIEW 語法：Babelfish 中對這些語法的支援有下列限制：
  - 無法在已連接 INSTEAD-OF 觸發條件的檢視上使用這些陳述式。
  - 若某些檢視具有其他檢視 (以此檢視為基礎)，則無法在那些檢視上使用這些陳述式。

改善 Babelfish 查詢效能

您可以使用查詢提示和 PostgreSQL 最佳化工具，在 Babelfish 中實現更快的查詢處理。

主題

- 使用解釋計劃改善 Babelfish 查詢效能
- 使用 T-SQL 查詢提示來改善 Babelfish 查詢效能

您也可以使用 `sp_babelfish_volatility` 程序改善查詢效能。如需更多詳細資訊，請參閱 sp_babelfish_volatility。

您也可以使用子查詢轉換和子查詢快取來改善查詢效能。如需更多詳細資訊，請參閱 最佳化 Aurora PostgreSQL 中的相互關聯子查詢。

使用解釋計劃改善 Babelfish 查詢效能

從第 2.1.0 版開始，Babelfish 包含兩個函數，以透明地使用 PostgreSQL 最佳化工具為 TDS 連接埠上的 T-SQL 查詢產生預估的和實際的查詢計劃。這些函數類似於將 SET STATISTICS PROFILE 或 SET SHOWPLAN_ALL 搭配 SQL Server 資料庫使用，以識別及改善慢速執行的查詢。

> **Note**
>
> 目前不支援從函數、控制流程與游標取得查詢計劃。

在此表中，您可以找到 SQL Server、Babelfish 及 PostgreSQL 之間查詢計劃解釋函數的比較。

SQL Server	Babelfish	PostgreSQL
SHOWPLAN_ALL	BABELFISH_SHOWPLAN_ALL	EXPLAIN
STATISTICS PROFILE	BABELFISH_STATISTICS PROFILE	解釋分析
使用 SQL Server 最佳化工具	使用 PostgreSQL 最佳化工具	使用 PostgreSQL 最佳化工具
SQL Server 輸入與輸出格式	SQL Server 輸入與 PostgreSQL 輸出格式	PostgreSQL 輸入和輸出格式
為工作階段設定	為工作階段設定	套用至特定陳述式
支援以下項目： • SELECT • INSERT • UPDATE • DELETE • CURSOR • CREATE • EXECUTE • EXEC 與函數，包括控制流程 (CASE、WHILE-BREAK-CONTINUE、WAITFOR、BEGIN-END、IF-ELSE 等等)	支援以下項目： • SELECT • INSERT • UPDATE • DELETE • CREATE • EXECUTE • EXEC • RAISEERROR • THROW • PRINT • USE	支援以下項目： • SELECT • INSERT • UPDATE • DELETE • CURSOR • CREATE • EXECUTE

按照下列方式使用 Babelfish 函數：

- SET BABELFISH_SHOWPLAN_ALL [ON|OFF] – 設定 ON 以產生預估的查詢執行計劃。此函數會實作 PostgreSQL `EXPLAIN` 命令的行為。使用此命令可取得給定查詢的解釋計劃。
- SET BABELFISH_STATISTICS PROFILE [ON|OFF] – 設定為 ON 以取得實際查詢執行計劃。這個函數實作 PostgreSQL 其 `EXPLAIN ANALYZE` 命令的行為。

如需有關在 PostgreSQL EXPLAIN 和 EXPLAIN ANALYZE 的詳細資訊，請參閱 PostgreSQL 文件中的 EXPLAIN。

> **Note**
>
> 從 2.2.0 版開始，您可以將 escape_hatch_showplan_all 參數設為 ignore，以避免在 SQL Server 語法中將 BABELFISH_ 字首用於 SHOWPLAN_ALL 和 STATISTICS PROFILE SET 命令。

例如，以下命令順序會開啟查詢計劃，然後傳回 SELECT 陳述式的預估查詢執行計劃，而不執行查詢。此範例透過使用 sqlcmd 命令列工具查詢 TDS 連接埠，以使用 SQL Server 範例 northwind 資料庫：

```
1> SET BABELFISH_SHOWPLAN_ALL ON
2> GO
1> SELECT t.territoryid, e.employeeid FROM
2> dbo.employeeterritories e, dbo.territories t
3> WHERE e.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO

QUERY PLAN


------------------------------------------------------------------------------


Query Text: SELECT t.territoryid, e.employeeid FROM
dbo.employeeterritories e, dbo.territories t
WHERE e.territoryid=e.territoryid ORDER BY t.territoryid
Sort  (cost=6231.74..6399.22 rows=66992 width=10)
  Sort Key: t.territoryid NULLS FIRST
  ->  Nested Loop  (cost=0.00..861.76 rows=66992 width=10)
        ->  Seq Scan on employeeterritories e  (cost=0.00..22.70 rows=1264 width=4)
              Filter: ((territoryid)::"varchar" IS NOT NULL)
        ->  Materialize  (cost=0.00..1.79 rows=53 width=6)
              ->  Seq Scan on territories t  (cost=0.00..1.53 rows=53 width=6)
```

完成檢閱和調整查詢後，您可以關閉該函數，如下所示：

```
1> SET BABELFISH_SHOWPLAN_ALL OFF
```

將 BABELFISH_STATISTICS PROFILE 設定為 ON 的情況下，每個執行的查詢會傳回其一般結果集，接著傳回一個顯示實際查詢執行計劃的附加結果集。Babelfish 會產生查詢計劃，以在呼叫 SELECT 陳述式時提供最快的結果集。

```
1> SET BABELFISH_STATISTICS PROFILE ON
1>
2> GO
1> SELECT e.employeeid, t.territoryid FROM
2> dbo.employeeterritories e, dbo.territories t
3> WHERE t.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO
```

傳回結果集和查詢計劃 (此範例僅顯示查詢計劃)。

```
QUERY PLAN


------------------------------------------------------------------------------
Query Text: SELECT e.employeeid, t.territoryid FROM
dbo.employeeterritories e, dbo.territories t
WHERE t.territoryid=e.territoryid ORDER BY t.territoryid
Sort  (cost=42.44..43.28 rows=337 width=10)
  Sort Key: t.territoryid NULLS FIRST


  ->  Hash Join  (cost=2.19..28.29 rows=337 width=10)
       Hash Cond: ((e.territoryid)::"varchar" = (t.territoryid)::"varchar")
        ->  Seq Scan on employeeterritories e  (cost=0.00..22.70 rows=1270 width=36)
        ->  Hash  (cost=1.53..1.53 rows=53 width=6)
             ->  Seq Scan on territories t  (cost=0.00..1.53 rows=53 width=6)
```

若要進一步了解如何分析您的查詢及 PostgreSQL 最佳化工具傳回的結果，請參閱 explain.depesz.com。如需有關 PostgreSQL EXPLAIN 與 EXPLAIN ANALYZE 的詳細資訊，請參閱 PostgreSQL 文件中的 EXPLAIN。

可控制 Babelfish 解釋選項的參數

您可以使用下表中顯示的參數控制查詢計劃顯示的資訊類型。

參數	描述
babelfishpg_tsql.explain_buffers	此布林值會開啟 (和關閉) 最佳化工具的緩衝區量資訊。(預設值：關閉) (允許值：關閉、開啟)
babelfishpg_tsql.explain_costs	此布林值會開啟 (和關閉) 最佳化工具的預估啟動和總成本資訊。(預設值：開啟) (允許值：關閉、開啟)
babelfishpg_tsql.explain_format	指定 EXPLAIN 計劃的輸出格式。(預設值：文字) (允許值：文字、xml、json、yaml)
babelfishpg_tsql.explain_settings	此布林值會開啟 (或關閉) 在 EXPLAIN 計劃輸出中包含組態參數的相關資訊。(預設值：關閉) (允許值：關閉、開啟)
babelfishpg_tsql.explain_summary	此布林值會開啟 (或關閉) 摘要資訊，例如查詢計劃後的總時間。(預設值：開啟) (允許值：關閉、開啟)
babelfishpg_tsql.explain_timing	此布林值會開啟 (或關閉) 實際啟動時間及在輸出的每個節點中花費的時間。(預設值：開啟) (允許值：關閉、開啟)
babelfishpg_tsql.explain_verbose	此布林值會開啟 (或關閉) 解釋計劃最詳細的版本。(預設值：關閉) (允許值：關閉、開啟)
babelfishpg_tsql.explain_wal	此布林值會開啟 (或關閉) WAL 記錄資訊的產生使其成為解釋計劃的一部分。(預設值：關閉) (允許值：關閉、開啟)

您可以透過使用 PostgreSQL 用戶端或 SQL Server 用户端，來檢查系統上任何 Babelfish 相關參數的值。請執行下列命令，以取得目前的參數值：

```
1> execute sp_babelfish_configure '%explain%';
2> GO
```

在以下輸出中，您可以看到此特定 Babelfish 資料庫叢集上的所有設定均為其預設值。並非所有的輸出都會出現在此範例中。

```
             name                    setting                      short_desc
--------------------------------- --------
 ---------------------------------------------------------
babelfishpg_tsql.explain_buffers   off      Include information on buffer usage
babelfishpg_tsql.explain_costs     on       Include information on estimated startup
 and total cost
babelfishpg_tsql.explain_format    text     Specify the output format, which can be
 TEXT, XML, JSON, or YAML
babelfishpg_tsql.explain_settings  off      Include information on configuration
 parameters
babelfishpg_tsql.explain_summary   on       Include summary information (e.g.,totaled
 timing information) after the query plan
babelfishpg_tsql.explain_timing    on       Include actual startup time and time spent
 in each node in the output
babelfishpg_tsql.explain_verbose   off      Display additional information regarding
 the plan
babelfishpg_tsql.explain_wal       off      Include information on WAL record
 generation

(8 rows affected)
```

您可以使用 sp_babelfish_configure 變更這些參數的設定，如下列範例所示。

```
1> execute sp_babelfish_configure 'explain_verbose', 'on';
2> GO
```

如果您想要讓設定永久存在於整個叢集層級上，請包括關鍵字 server，如下列範例所示。

```
1> execute sp_babelfish_configure 'explain_verbose', 'on', 'server';
2> GO
```

使用 T-SQL 查詢提示來改善 Babelfish 查詢效能

從 2.3.0 版開始，Babelfish 支援使用 pg_hint_plan 來使用查詢提示。在 Aurora PostgreSQL 中，預設會安裝 pg_hint_plan。如需 PostgreSQL 延伸模組 pg_hint_plan 的詳細資訊，請參閱 https://github.com/ossc-db/pg_hint_plan。如需 Aurora PostgreSQL 支援的這個延伸模組版本的詳細資訊，請參閱《Aurora PostgreSQL 版本備註》中的 Amazon Aurora PostgreSQL 延伸模組版本。

查詢最佳化工具是精心設計來為 SQL 陳述式尋找最佳執行計劃。選取計劃時，查詢最佳化工具會同時考慮引擎的成本模型，以及資料欄和資料表統計資料。不過，建議的計劃可能不符合資料集的需求。因此，查詢提示可解決效能問題，以改善執行計劃。query hint 是新增到 SQL 標準的語法，其會指示資料庫引擎如何執行查詢。例如，提示可能會指示引擎遵循循序掃描，並覆寫查詢最佳化工具已選取的任何計劃。

開啟 Babelfish 中的 T-SQL 查詢提示

目前，Babelfish 預設忽略所有 T-SQL 提示。若要套用 T-SQL 提示，請在 enable_pg_hint 為 ON 的情況下執行命令 sp_babelfish_configure。

```
EXECUTE sp_babelfish_configure 'enable_pg_hint', 'on' [, 'server']
```

您可以包括 server 關鍵字，讓設定永久存在於整個叢集層級上。若只要設定目前工作階段的設定，請不要使用 server。

在 enable_pg_hint 為 ON 之後，Babelfish 會套用下列 T-SQL 提示。

- INDEX 提示
- JOIN 提示
- FORCE ORDER 提示
- MAXDOP 提示

例如，下列命令序列會開啟 pg_hint_plan。

```
1> CREATE TABLE t1 (a1 INT PRIMARY KEY, b1 INT);
2> CREATE TABLE t2 (a2 INT PRIMARY KEY, b2 INT);
3> GO
1> EXECUTE sp_babelfish_configure 'enable_pg_hint', 'on';
2> GO
1> SET BABELFISH_SHOWPLAN_ALL ON;
2> GO
1> SELECT * FROM t1 JOIN t2 ON t1.a1 = t2.a2; --NO HINTS (HASH JOIN)
2> GO
```

沒有提示會套用至 SELECT 陳述式。傳回沒有提示的查詢計劃。

```
QUERY PLAN


------------------------------------------------------------------------------
Query Text: SELECT * FROM t1 JOIN t2 ON t1.a1 = t2.a2
Hash Join (cost=60.85..99.39 rows=2260 width=16)
 Hash Cond: (t1.a1 = t2.a2)
 -> Seq Scan on t1 (cost=0.00..32.60 rows=2260 width=8)
 -> Hash (cost=32.60..32.60 rows=2260 width=8)
 -> Seq Scan on t2 (cost=0.00..32.60 rows=2260 width=8)
```

```
1> SELECT * FROM t1 INNER MERGE JOIN t2 ON t1.a1 = t2.a2;
2> GO
```

查詢提示會套用至 SELECT 陳述式。下列輸出顯示傳回了具有合併聯結的查詢計劃。

```
QUERY PLAN


------------------------------------------------------------------------------
Query Text: SELECT/*+ MergeJoin(t1 t2) Leading(t1 t2)*/ * FROM t1 INNER JOIN t2 ON
 t1.a1 = t2.a2
Merge Join (cost=0.31..190.01 rows=2260 width=16)
 Merge Cond: (t1.a1 = t2.a2)
 -> Index Scan using t1_pkey on t1 (cost=0.15..78.06 rows=2260 width=8)
 -> Index Scan using t2_pkey on t2 (cost=0.15..78.06 rows=2260 width=8)
```

```
1> SET BABELFISH_SHOWPLAN_ALL OFF;
2> GO
```

限制

使用查詢提示時，請考慮下列限制：

- 如果在 enable_pg_hint 開啟之前已快取查詢計劃，則不會在相同工作階段中套用提示。它將在新的工作階段中套用。
- 如果明確地提供結構描述名稱，則無法套用提示。您可以使用資料表別名做為解決方法。

- 查詢提示無法套用至檢視和子查詢。
- 提示不適用於搭配 JOIN 的 UPDATE/DELETE 陳述式。
- 系統會忽略不存在索引或資料表的索引提示。
- FORCE ORDER 提示不適用於 HASH JOIN 和非 HASH JOIN。

搭配 Babelfish 使用 Aurora PostgreSQL 擴充功能

Aurora PostgreSQL 提供使用其他服務的擴充 AWS 功能。以下是支援各種使用案例的選用擴充功能，例如使用 Amazon S3 搭配資料庫叢集以匯入或匯出資料。

- 若要將資料從 Amazon S3 儲存貯體匯入至 Babelfish 資料庫叢集，需要設定 `aws_s3` Aurora PostgreSQL 擴充功能。此擴充功能還可讓您將資料從 Aurora PostgreSQL 資料庫叢集匯出至 Amazon S3 儲存貯體。
- AWS Lambda 是一種運算服務，可讓您執行程式碼，而無需佈建或管理伺服器。您可以使用 Lambda 函數執行各種程序，例如處理來自資料庫執行個體的事件通知。若要進一步了解 Lambda，請參閱《AWS Lambda 開發人員指南》中的什麼是 AWS Lambda？。若要從 Babelfish 資料庫叢集叫用 Lambda 函數，需設定 `aws_lambda` Aurora PostgreSQL 擴充功能。

若要為 Babelfish 叢集設定這些擴充功能，首先需要授予載入擴充功能的許可給內部 Babelfish 使用者。授予許可後，就可以載入 Aurora PostgreSQL 擴充功能。

在 Babelfish 資料庫叢集中啟用 Aurora PostgreSQL 擴充功能

授予 Babelfish 資料庫叢集所需的權限後，才能載入 `aws_s3` 或 `aws_lambda` 擴充功能。

以下程序使用 `psql` PostgreSQL 命令列工具連線到資料庫叢集。如需詳細資訊，請參閱使用 psql 來連線至資料庫叢集。您也可以使用 pgAdmin。如需詳細資訊，請參閱 使用 pgAdmin 來連線至資料庫叢集。

此程序會依序載入 `aws_s3` 和 `aws_lambda`。如果您只想使用其中一個擴充功能，就不需要同時載入兩者。各自都需要 `aws_commons` 擴充功能，且會依預設載入，如輸出中所示。

使用 Aurora PostgreSQL 擴充功能的權限設定 Babelfish 資料庫叢集

1. 連線至 Babelfish 資料庫叢集。使用您在建立 Babelfish 資料庫叢集時指定的「主要」使用者 (-U) 名稱。預設值 (`postgres`) 顯示在範例中。

   對於 Linux、macOS 或 Unix：

```
psql -h your-Babelfish.cluster.444455556666-us-east-1.rds.amazonaws.com \
-U postgres \
-d babelfish_db \
-p 5432
```

在 Windows 中：

```
psql -h your-Babelfish.cluster.444455556666-us-east-1.rds.amazonaws.com ^
-U postgres ^
-d babelfish_db ^
-p 5432
```

命令回應會出現提示，要求輸入使用者名稱 (-U) 的密碼。

```
Password:
```

輸入資料庫叢集使用者名稱 (-U) 的密碼。如果成功連線，會出現類似以下輸出。

```
psql (13.4)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
 compression: off)
Type "help" for help.

postgres=>
```

2. 授予內部 Babelfish 使用者建立和載入擴充功能的權限。

```
babelfish_db=> GRANT rds_superuser TO master_dbo;
GRANT ROLE
```

3. 建立並載入 aws_s3 擴充功能。aws_commons 擴充功能為必要項目，且在安裝 aws_s3 時即已自動安裝。

```
babelfish_db=> create extension aws_s3 cascade;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
```

4. 建立並載入 aws_lambda 擴充功能。

```
babelfish_db=> create extension aws_lambda cascade;
```

```
CREATE EXTENSION
babelfish_db=>
```

搭配使用 Babelfish 與 Amazon S3

如果您還沒有 Amazon S3 儲存貯體能與 Babelfish 資料庫叢集搭配使用，可以建立一個儲存貯體。為您想使用的任何 Amazon S3 儲存貯體提供存取權。

嘗試使用 Amazon S3 儲存貯體匯入或匯出資料之前，請完成以下一次性步驟。

為 Babelfish 資料庫執行個體設定 Amazon S3 儲存貯體的存取權

1. 如果需要，可為 Babelfish 執行個體建立 Amazon S3 儲存貯體。若要執行此作業，請依照《Amazon Simple Storage Service 使用者 Guideguide-s3-user;》中建立儲存貯體提供的說明操作。
2. 將檔案上傳至 Amazon S3 儲存貯體。若要執行此作業，請依照《Amazon Simple Storage Service 使用者 Guideguide-s3-user;》中將物件新增到儲存貯體的說明操作。
3. 視需要設定許可：

   - 若要從 Amazon S3 匯入資料，Babelfish 資料庫叢集需要存取儲存貯體的許可。建議您使用 AWS Identity and Access Management (IAM) 角色，並將 IAM 政策連接至叢集的該角色。若要啟用，請依照「使用 IAM 角色存取 Amazon S3 儲存貯體」中的步驟進行。
   - 若要從 Babelfish 資料庫叢集中匯出資料，必須將 Amazon S3 儲存貯體的存取權授予叢集。與匯入一樣，建議使用 IAM 角色和政策。若要啟用，請依照「設定對 Amazon S3 儲存貯體的存取權」中的步驟進行。

您現在可以搭配 Babelfish 資料庫叢集使用 Amazon S3 與 `aws_s3` 擴充功能。

將資料從 Amazon S3 匯入至 Babelfish 以及將 Babelfish 資料匯出至 Amazon S3

1. 搭配 Babelfish 資料庫叢集使用 `aws_s3` 擴充功能。

   執行此操作時，請務必以資料表存在於 PostgreSQL 情境的方式參考資料表。也就是說，如果你想匯入至名稱為 `[database].[schema].[tableA]` 的 Babelfish 資料表，請在 `aws_s3` 函數中以 `database_schema_tableA` 參考該表:

   - 如需使用 `aws_s3` 函數匯入資料的範例，請參閱 將資料從 Amazon S3 匯入 Aurora PostgreSQL 資料庫叢集 。

- 如需使用 aws_s3 函數匯出資料的範例，請參閱 使用 aws_s3.query_export_to_s3 函數匯出查詢資料。

2. 使用 aws_s3 擴充功能和 Amazon S3 時，請務必以 PostgreSQL 命名方式參考 Babelfish 資料表，如下表所示。

Babelfish 資料表	Aurora PostgreSQL 資料表
database.*schema*.*table*	*database_schema_table*

若要進一步了解如何將 Amazon S3 與 Aurora PostgreSQL 搭配使用，請參閱 將資料從 Amazon S3 匯入 Aurora PostgreSQL 資料庫叢集 和 將資料從 Aurora PostgreSQL 資料庫叢集匯出至 Amazon S3。

搭配 使用 Babelfish AWS Lambda

aws_lambda 擴充功能載入 Babelfish 資料庫叢集中之後，但在叫用 Lambda 函數之前，您可以按照此程序為資料庫叢集提供 Lambda 存取權。

為 Babelfish 資料庫叢集設定存取權以搭配使用 Lambda

此程序使用 AWS CLI 建立 IAM 政策和角色，並將其與 Babelfish 資料庫叢集建立關聯。

1. 建立允許從 Babelfish 資料庫叢集存取 Lambda 的 IAM 政策。

```
aws iam create-policy  --policy-name rds-lambda-policy --policy-document '{
 "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "AllowAccessToExampleFunction",
        "Effect": "Allow",
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"
        }
    ]
}'
```

2. 建立政策可在執行時擔任的 IAM 角色。

```
aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}'
```

3. 將政策連接到角色。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \
    --role-name rds-lambda-role --region aws-region
```

4. 將角色連接至 Babelfish 資料庫叢集

```
aws rds add-role-to-db-cluster \
       --db-cluster-identifier my-cluster-name \
       --feature-name Lambda \
       --role-arn  arn:aws:iam::444455556666:role/rds-lambda-role   \
       --region aws-region
```

完成這些任務後，您就可以叫用 Lambda 函數。如需使用 設定 AWS Lambda Aurora PostgreSQL 資料庫叢集的詳細資訊和範例 AWS Lambda，請參閱 步驟 2：為您的 Aurora PostgreSQL 資料庫叢集和 設定 IAM AWS Lambda。

從 Babelfish 資料庫叢集叫用 Lambda 函數

AWS Lambda 支援以 Java、Node.js、Python、Ruby 和其他語言撰寫的函數。如果叫用的函數會傳回文字，則可以從 Babelfish 資料庫叢集中叫用它。以下範例是傳回問候語的預留位置 python 函數。

```
lambda_function.py
import json
def lambda_handler(event, context):
    #TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
```

目前 Babelfish 不支援 JSON。如果函數會傳回 JSON，需使用包裝函式來處理 JSON。例如，假設上方顯示的 `lambda_function.py` 在 Lambda 中會存放為 `my-function`。

1. 使用 `psql` 用戶端 (或 pgAdmin 用戶端) 連線至 Babelfish 資料庫叢集。如需詳細資訊，請參閱使用 psql 來連線至資料庫叢集。
2. 建立包裝函式。此範例針對 SQL 使用 PostgreSQL 的程序語言 PL/pgSQL。如需進一步了解，請參閱 PL/PGSQL-SQL 程序性語言。

```
create or replace function master_dbo.lambda_wrapper()
returns text
language plpgsql
as
$$
declare
   r_status_code integer;
   r_payload text;
begin
   SELECT payload INTO r_payload
     FROM aws_lambda.invoke(  aws_commons.create_lambda_function_arn('my-function',
 'us-east-1')
                           ,'{"body": "Hello from Postgres!"}'::json );
   return r_payload ;
end;
$$;
```

現在可以從 Babelfish TDS 連接埠 (1433) 或 PostgreSQL 連接埠 (5433) 執行該函數。

a. 若要從 PostgreSQL 連接埠叫用 (呼叫) 此函數：

```
SELECT * from aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function', 'us-east-1'), '{"body": "Hello from Postgres!"}'::json );
```

輸出類似以下內容：

```
status_code |                       payload                        |
 executed_version | log_result
-------------+------------------------------------------------------
+------------------+------------
         200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST
         |
```

```
(1 row)
```

b. 若要從 TDS 連接埠叫用 (呼叫) 此函數，請使用 SQL Server `sqlcmd` 命令列用戶端連線到連接埠。如需詳細資訊，請參閱使用 SQL Server 用戶端來連線至資料庫叢集。連線後，請執行以下命令：

```
1> select lambda_wrapper();
2> go
```

此命令會傳回類似以下的輸出：

```
{"statusCode": 200, "body": "\"Hello from Lambda!\""}
```

若要進一步了解如何將 Lambda 與 Aurora PostgreSQL 搭配使用，請參閱 從 Aurora PostgreSQL 資料庫叢集 叫用 AWS Lambda 函數。如需使用 Lambda 函數的詳細資訊，請參閱《AWS Lambda 開發人員指南》中的 Lambda 入門。

在 Babelfish 中使用 pg_stat_statements

Babelfish for Aurora PostgreSQL 從 3.3.0 開始支援 `pg_stat_statements` 擴充功能。若要進一步瞭解，請參閱 pg_stat_statements。

如需 Aurora PostgreSQL 支援的這個擴充功能版本的詳細資訊，請參閱擴充功能版本。

建立 pg_stat_statements 擴充功能

若要開啟 `pg_stat_statements`，您必須開啟查詢識別碼計算。如果 `compute_query_id` 在參數群組中設定為 `on` 或 `auto`，則會自動完成此操作。`compute_query_id` 參數的預設值為 `auto`。您也需要建立此擴充功能，才能開啟此功能。使用下列命令從 T-SQL 端點安裝擴充功能：

```
1>EXEC sp_execute_postgresql 'CREATE EXTENSION pg_stat_statements WITH SCHEMA sys';
```

您可以使用下列查詢來存取查詢統計資料：

```
postgres=>select * from pg_stat_statements;
```

> **Note**
>
> 在安裝過程中，如果您沒有為擴充功能提供結構描述名稱，則預設情況下它將在公開結構描述中建立它。若要存取它，您必須使用帶有結構描述限定詞的方括號，如下所示：

```
postgres=>select * from [public].pg_stat_statements;
```

您也可以從 PSQL 端點建立擴充功能。

授權擴充功能

根據預設，您可以查看在 T-SQL 資料庫中執行之查詢的統計資料，而不需要任何授權。

若要存取其他人所建立的查詢統計資料，您必須擁有 `pg_read_all_stats` PostgreSQL 角色。按照下面提到的步驟建立 GRANT pg_read_all_stats 命令。

1. 在 T-SQL 中，請使用下列查詢來傳回內部 PG 角色名稱。

   ```
   SELECT rolname FROM pg_roles WHERE oid = USER_ID();
   ```

2. 使用 rds_superuser 權限連接到 Babelfish for Aurora PostgreSQL 資料庫並使用以下命令：

   ```
   GRANT pg_read_all_stats TO <rolname_from_above_query>
   ```

範例

從 T-SQL 端點：

```
1>SELECT rolname FROM pg_roles WHERE oid = USER_ID();
2>go
```

```


rolname
-------
master_dbo
(1 rows affected)
```

從 PSQL 端點：

```
babelfish_db=# grant pg_read_all_stats to master_dbo;
```

```
GRANT ROLE
```

您可以使用 pg_stat_statements 檢視來存取查詢統計資料：

```
1>create table t1(cola int);
2>go
1>insert into t1 values (1),(2),(3);
2>go
```

```
(3 rows affected)
```

```
1>select userid, dbid, queryid, query from pg_stat_statements;
2>go
```

```
userid dbid queryid             query
------ ---- -------             -----
37503 34582 6487973085327558478 select * from t1
37503 34582 6284378402749466286 SET QUOTED_IDENTIFIER OFF
37503 34582 2864302298511657420 insert into t1 values ($1),($2),($3)
10    34582 NULL                <insufficient privilege>
37503 34582 5615368793313871642 SET TEXTSIZE 4096
37503 34582 639400815330803392  create table t1(cola int)
(6 rows affected)
```

重設查詢統計

您可以使用 `pg_stat_statements_reset()` 來重設 pg_stat_statements 迄今收集的統計資料。若要進一步瞭解，請參閱 pg_stat_statements。目前僅透過 PSQL 端點支援此功能。使用 `rds_superuser` 權限連接到 Babelfish for Aurora PostgreSQL，使用以下命令：

```
SELECT pg_stat_statements_reset();
```

限制

- 目前，T-SQL 端點不支援 pg_stat_statements()。pg_stat_statements 檢視是收集統計資料的建議方法。
- 某些查詢可能會由 Aurora PostgreSQL 引擎實現的 T-SQL 解析器重寫，pg_stat_statements 檢視將顯示重寫的查詢，而不是原始查詢。

  範例

  ```
  select next value for [dbo].[newCounter];
  ```

  上述查詢在 pg_stat_statements 檢視中重寫如下。

  ```
  select nextval($1);
  ```

- 根據陳述式的執行流程，某些查詢可能不會被 pg_stat_statements 追蹤，且在檢視中不可見。這包括下列陳述式：use dbname、goto、print、raise error、set、throw、declare cursor。
- 對於 CREATE LOGIN 和 ALTER LOGIN 陳述式，不會顯示查詢和 queryid。它將顯示權限不足。
- pg_stat_statements 檢視始終包含以下兩個項目，因為這些是由 sqlcmd 用戶端內部執行。
  - SET QUOTED_IDENTIFIER OFF
  - SET TEXTSIZE 4096

在 Babelfish 中使用 pgvector

pgvector 是一種開放原始碼延伸模組，可讓您直接在 Postgres 資料庫中搜尋類似的資料。Babelfish 現支援從 15.6 和 16.2 版開始的此延伸模組。如需詳細資訊，請參閱 pgvector 開放原始碼文件。

先決條件

若要啟用 pgvector 功能，請使用下列其中一種方法，在 sys 結構描述中安裝延伸模組：

- 在 sqlcmd 用戶端中執行下列命令：

  ```
  exec sys.sp_execute_postgresql 'CREATE EXTENSION vector WITH SCHEMA sys';
  ```

- 連接至 babelfish_db，以及在 psql 用戶端中執行下列命令：

```
CREATE EXTENSION vector WITH SCHEMA sys;
```

> **Note**
>
> 安裝 pgvector 延伸模組後，就只能在您建立的新資料庫連線中使用此向量資料類型。現有的連線無法辨識新的資料類型。

受支援的功能

Babelfish 延伸 T-SQL 功能以支援下列項目：

- 存放

  Babelfish 現支援向量資料類型相容的語法，進而增強其 T-SQL 相容性。若要進一步了解如何使用 pgvector 存放資料，請參閱存放。
- 查詢

  Babelfish 延伸 T-SQL 表達式支援，以包含向量相似性運算子。不過，對於所有其他查詢，仍需要標準 T-SQL 語法。

  > **Note**
  >
  > T-SQL 不支援陣列類型，且資料庫驅動程式沒有任何可處理這些類型的界面。作為解決方法，Babelfish 使用文字字串 (varchar/nvarchar) 來存放向量資料。例如，當您請求向量值 [1,2,3] 時，Babelfish 會傳回字串 '[1,2,3]' 作為回應。您可以根據需求，在應用程式層級剖析和分割此字串。

  若要進一步了解如何使用 pgvector 查詢資料，請參閱查詢。
- 編製索引

  T-SQL `Create Index` 現在支援 `USING INDEX_METHOD` 語法。您現在可以定義相似性搜尋運算子，以在建立索引時用於特定欄。

  文法也會延伸，以支援必要欄的向量相似性操作 (檢查 column_name_list_with_order_for_vector 文法)。

```
CREATE [UNIQUE] [clustered] [COLUMNSTORE] INDEX <index_name> ON <table_name> [USING
 vector_index_method] (<column_name_list_with_order_for_vector>)
Where column_name_list_with_order_for_vector is:
    <column_name> [ASC | DESC] [VECTOR_COSINE_OPS | VECTOR_IP_OPS | VECTOR_L2_OPS]
 (COMMA simple_column_name [ASC | DESC] [VECTOR_COSINE_OPS | VECTOR_IP_OPS |
 VECTOR_L2_OPS])
```

若要進一步了解如何使用 pgvector 為資料編製索引，請參閱編製索引。

- 效能
  - 使用 `SET BABELFISH_STATISTICS PROFILE ON` 從 T-SQL 端點針對查詢計劃進行偵錯。
  - 使用 T-SQL 中支援的 `set_config` 函數來增加 `max_parallel_workers_get_gather`。
  - 使用 `IVFFlat` 進行近似搜尋。如需更多詳細資訊，請參閱 IVFFlat。

  若要改善 pgvector 的效能，請參閱效能。

限制

- Babelfish 不支援混合搜尋的全文搜尋。如需詳細資訊，請參閱混合搜尋。
- Babelfish 目前不支援重新編製索引功能。不過，您仍可以使用 PostgreSQL 端點來重新編製索引。如需詳細資訊，請參閱清空。

將 Amazon Aurora 機器學習功能與 Babelfish 搭配使用

您可以將 Babelfish for Aurora PostgreSQL 資料庫叢集與 Amazon Aurora 機器學習整合，以延伸其功能。這種順暢整合可讓您存取各種功能強大的服務，例如 Amazon Comprehend 或 Amazon SageMaker AI 或 Amazon Bedrock，每個服務都專為滿足不同的機器學習需求而量身打造。

身為 Babelfish 使用者，您可以在使用 Aurora 機器學習時使用現有的 T-SQL 語法和語意知識。遵循 Aurora PostgreSQL AWS 文件中提供的指示。如需詳細資訊，請參閱將 Amazon Aurora Machine Learning 與 Aurora PostgreSQL 搭配使用。

先決條件

- 在嘗試設定 Babelfish for Aurora PostgreSQL 資料庫叢集以使用 Aurora 機器學習之前，請務必了解相關要求和先決條件。如需詳細資訊，請參閱將 Aurora Machine Learning 與 Aurora PostgreSQL 搭配使用的建議。
- 請務必擇一使用 Postgres 端點或 `sp_execute_postgresql` 存放程序來安裝 `aws_ml` 延伸模組。

```
exec sys.sp_execute_postgresql 'Create Extension aws_ml'
```

> **Note**
>
> Babelfish 目前不支援在 Babelfish 中使用 `sp_execute_postgresql` 的層疊操作。`aws_ml` 依賴 `aws_commons`，因此您需要使用 Postgres 端點來單獨安裝此項目。

```
create extension aws_common;
```

使用 **aws_ml** 函數處理 T-SQL 語法和語意

下列範例說明如何將 T-SQL 語法和語意套用至 Amazon ML 服務：

Example: aws_bedrock.invoke_model：使用 Amazon Bedrock 函數的簡單查詢

```
aws_bedrock.invoke_model(
   model_id      varchar,
   content_type  text,
   accept_type   text,
   model_input   text)
Returns Varchar(MAX)
```

下列範例示範如何使用 invoke_model 調用 Bedrock 的 Anthropic Claude 2 模型。

```
SELECT aws_bedrock.invoke_model (
    'anthropic.claude-v2', -- model_id
    'application/json', -- content_type
    'application/json', -- accept_type
    '{"prompt": "\n\nHuman:
    You are a helpful assistant that answers questions directly
    and only using the information provided in the context below.
    \nDescribe the answerin detail.\n\nContext: %s \n\nQuestion:
    %s \n\nAssistant:","max_tokens_to_sample":4096,"temperature"
    :0.5,"top_k":250,"top_p":0.5,"stop_sequences":[]}' -- model_input
);
```

Example: aws_comprehend.detect_sentiment：使用 Amazon Comprehend 函數的簡單查詢

```
aws_comprehend.detect_sentiment(
   input_text varchar,
   language_code varchar,
   max_rows_per_batch int)
Returns table (sentiment varchar, confidence real)
```

下列範例示範如何調用 Amazon Comprehend 服務。

```
select sentiment from aws_comprehend.detect_sentiment('This is great', 'en');
```

Example: aws_sagemaker.invoke_endpoint：使用 Amazon SageMaker 函數的簡單查詢

```
aws_sagemaker.invoke_endpoint(
  endpoint_name varchar,
  max_rows_per_batch int,
  VARIADIC model_input "any") -- Babelfish inherits PG's variadic parameter type
Rerurns Varchar(MAX)
```

model_input 標記為 VARIADIC 且類型為「任何」，因此使用者可以將任意長度和任何資料類型的清單傳遞至函數，該函數將作為此模型的輸入。下列範例顯示如何調用 Amazon SageMaker 服務。

```
SELECT CAST (aws_sagemaker.invoke_endpoint(
    'sagemaker_model_endpoint_name',
    NULL,
    arg1, arg2 -- model inputs are separate arguments )
AS INT) -- cast the output to INT
```

如需如何搭配 Aurora PostgreSQL 使用 Aurora 機器學習的詳細資訊，請參閱將 Amazon Aurora Machine Learning 與 Aurora PostgreSQL 搭配使用。

限制

- 雖然 Babelfish 不允許建立陣列，但仍可以處理代表陣列的資料。當您使用會傳回陣列的 `aws_bedrock.invoke_model_get_embeddings` 之類的函數時，系統會以內含陣列元素的字串傳遞結果。

Babelfish 支援連結的伺服器

Babelfish for Aurora PostgreSQL 使用 3.1.0 版的 PostgreSQL `tds_fdw` 擴充功能來支援連結的伺服器。若要使用連接的伺服器，您必須安裝 `tds_fdw` 擴充功能。如需詳細了解 `tds_fdw` 擴充功能，請參閱 使用 Amazon Aurora PostgreSQL 支援的外部資料包裝函式。

安裝 **tds_fdw** 擴充功能

您可以使用以下方法安裝 `tds_fdw` 擴充功能。

從 PostgreSQL 端點使用 CREATE EXTENSION

1. 連線至 PostgreSQL 連接埠中 Babelfish 資料庫的 PostgreSQL 資料庫執行個體。使用具有 rds_superuser 角色的帳戶。

```
psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --
username=test --dbname=babelfish_db --password
```

2. 安裝 `tds_fdw` 擴充功能。這是一次性安裝程序。您不需要在重啟資料庫叢集時重新安裝。

```
babelfish_db=> CREATE EXTENSION tds_fdw;
CREATE EXTENSION
```

從 TDS 端點呼叫 **sp_execute_postgresql** 預存程序

Babelfish 從 3.3.0 版本開始支援透過呼叫 `sp_execute_postgresql` 程序來安裝 `tds_fdw` 擴充功能。您可以在不結束 T-SQL 連接埠的情況下，從 T-SQL 端點執行 PostgreSQL 陳述式。如需詳細資訊，請參閱使用 Babelfish for Aurora PostgreSQL

1. 連線至 T-SQL 連接埠中 Babelfish 資料庫的 PostgreSQL 資料庫執行個體。

```
sqlcmd -S your-DB-instance.aws-region.rds.amazonaws.com -U test -P password
```

2. 安裝 tds_fdw 擴充功能。

```
1>EXEC sp_execute_postgresql N'CREATE EXTENSION tds_fdw';
2>go
```

受支援的功能

Babelfish 支援新增遠端 RDS for SQL Server 或 Babelfish for Aurora PostgreSQL 端點做為連結的伺服器。您也可以新增其他遠端 SQL Server 執行個體做為連結的伺服器。接著，使用 OPENQUERY() 從這些連結的伺服器擷取資料。從 Babelfish 3.2.0 版開始，也支持四段式名稱。

下列預存程序和目錄檢視皆受到支援，以使用連結的伺服器。

預存程序

- sp_addlinkedserver – Babelfish 不支援 @provstr 參數。
- sp_addlinkedsrvlogin
  - 您必須提供明確的遠端使用者名稱和密碼，才能連線到遠端資料來源。您無法透過使用者自我憑證進行連線。Babelfish 僅支援 @useself = false。
  - Babelfish 不支援 @locallogin 參數，因為設定特定於本機登入的遠端伺服器存取不受支援。
- sp_linkedservers
- sp_helplinkedsrvlogin
- sp_dropserver
- sp_droplinkedsrvlogin – Babelfish 不支援 @locallogin 參數，因為設定特定於本機登入的遠端伺服器存取不受支援。
- sp_serveroption - Babelfish 支援下列伺服器選項：
  - 查詢逾時 (從 Babelfish 3.2.0 版開始)
  - 連線逾時 (從 Babelfish 3.3.0 版開始)
- sp_testlinkedserver (從 Babelfish 3.3.0 版開始)
- sp_enum_oledb_providers (從 Babelfish 3.3.0 版開始)

目錄檢視

- sys.servers

- sys.linked_logins

對連線使用傳輸中加密

從來源 Babelfish for Aurora PostgreSQL 伺服器到目標遠端伺服器的連接使用傳輸中加密 (TLS/SSL)，具體取決於遠端伺服器資料庫組態。如果遠端伺服器未設定為加密，則 Babelfish 伺服器對遠端資料庫發出的請求將回退為未加密。

強制執行連線加密

- 如果目標連結伺服器是 RDS for SQL Server 執行個體，請將目標 SQL Server 執行個體設為 `rds.force_ssl = on`。如需有關 RDS for SQL Server 的 SSL/TLS 組態的詳細資訊，請參閱對 Microsoft SQL Server 資料庫執行個體使用 SSL
- 如果目標連結伺服器是 Babelfish for Aurora PostgreSQL 叢集，請將目標伺服器設為 `babelfishpg_tds.tds_ssl_encrypt = on` 和 `ssl = on`。如需有關 SSL/TLS 的詳細資訊，請參閱 Babelfish SSL 設定與用戶端連線。

將 Babelfish 新增為來自 SQL Server 的連結伺服器

Babelfish for Aurora PostgreSQL 可以新增為來自 SQL Server 的連結伺服器。在 SQL Server 資料庫上，您可以使用適用於 ODBC 的 Microsoft OLE 資料庫提供者，將 Babelfish 新增為連結伺服器：MSDASQL。

有兩種方法可以使用 MSDASQL 提供者，將 Babelfish 設定為來自 SQL Server 的連結伺服器：

- 提供 ODBC 連線字串作為提供者字串。
- 新增連結伺服器時，提供 ODBC 資料來源的系統 DSN。

限制

- OPENQUERY() 僅適用於 SELECT，不適用於 DML。
- 由四個部分組成的物件名稱僅適用於讀取，不適用於修改遠端資料表。UPDATE 可以參考 FROM 子句中的遠端資料表，而不對其進行修改。
- 不支援針對 Babelfish 連結伺服器執行預存程序。
- 若有依賴 `OPENQUERY()` 的物件或透過四段式名稱參考的物件，則 Babelfish 主要版本升級可能無法運作。在進行主要版本升級之前，您必須確定任何參考 `OPENQUERY()` 或四段式名稱的物件都已捨棄。

- 對於遠端 Babelfish 伺服器，下列資料類型無法如預期般運作：`nvarchar(max)`、`varchar(max)`、`varbinary(max)`、`binary(max)` 和 `time`。我們建議使用 CAST 函數將這些轉換為支援的資料類型。

範例

在下列範例中，Babelfish for Aurora PostgreSQL 執行個體連線至雲端中的 RDS for SQL Server 執行個體。

```
EXEC master.dbo.sp_addlinkedserver @server=N'rds_sqlserver', @srvproduct=N'',
 @provider=N'SQLNCLI', @datasrc=N'myserver.CB2XKFSFFMY7.US-WEST-2.RDS.AMAZONAWS.COM';
EXEC master.dbo.sp_addlinkedsrvlogin
 @rmtsrvname=N'rds_sqlserver',@useself=N'False',@locallogin=NULL,@rmtuser=N'username',@rmtpassw
```

當連結的伺服器就緒時，您就可以使用 T-SQL OPENQUERY() 或標準的四部分命名來參考遠端伺服器上的資料表、檢視表或其他支援的物件：

```
SELECT * FROM OPENQUERY(rds_sqlserver, 'SELECT * FROM TestDB.dbo.t1');
SELECT * FROM rds_sqlserver.TestDB.dbo.t1;
```

若要捨棄連結的伺服器和所有相關聯的登入：

```
EXEC master.dbo.sp_dropserver @server=N'rds_sqlserver', @droplogins=N'droplogins';
```

疑難排解

您可以對來源伺服器和遠端伺服器使用相同的安全群組，以允許它們彼此通訊。安全群組應該只允許 TDS 連接埠 (預設為 1433) 上的輸入流量，而安全群組中的來源 IP 可以設定為安全群組 ID 本身。如需如何設定從具有相同安全群組的另一個執行個體連線至執行個體的規則的詳細資訊，請參閱從執行個體使用相同安全群組連線到執行個體的規則。

若未正確設定存取權，當您嘗試查詢遠端伺服器時，會出現類似下列範例的錯誤訊息。

```
TDS client library error: DB #: 20009, DB Msg: Unable to connect: server is unavailable
 or does not exist (mssql2019.aws-region.rds.amazonaws.com), OS #: 110, OS Msg:
 Connection timed out, Level: 9
```

在 Babelfish 中使用全文搜尋

從 4.0.0 版開始，Babelfish 為全文搜尋 (FTS) 提供有限的支援。FTS 是關聯式資料庫中強大的功能，可有效搜尋大量文字資料和為其編製索引。此功能可讓您執行複雜的文字搜尋，並快速擷取相關結果。FTS 對於處理大量文字資料的應用程式特別重要，例如內容管理系統、電子商務平台和文件封存。

了解 Babelfish 全文搜尋支援的功能

Babelfish 支援下列全文搜尋功能：

- CONTAINS 子句：
  - CONTAINS 子句的基本支援。

    ```
    CONTAINS (
         {
            column_name
         }
         , '<contains_search_condition>'
    )
    ```

    > **Note**
    >
    > 目前僅支援英文語言。

  - `simple_term` 搜尋字串的全面處理和轉譯。
- `FULLTEXT INDEX` 子句：
  - 僅支援 `CREATE FULLTEXT INDEX ON table_name(column_name [...n]) KEY INDEX index_name` 陳述式。
  - 支援完整 `DROP FULLTEXT INDEX` 陳述式。

> **Note**
>
> 若要為全文索引重新編製索引，您需要捨棄全文索引，以及在相同的欄上建立新的索引。

- 搜尋條件中的特殊字元：
  - Babelfish 可確保有效處理搜尋字串中出現的單一特殊字元。

    > **Note**
    >
    > 雖然 Babelfish 現在會在搜尋字串中識別特殊字元，但請務必注意取得的結果可能與使用 T-SQL 取得的結果不同。

- column_name 中的資料表別名：
  - 透過資料表別名支援，使用者可以為全文搜尋建立更簡潔、可讀的 SQL 查詢。

Babelfish 全文搜尋中的限制

- 目前，Babelfish 中不支援 `CONTAINS` 子句的下列選項。
  - 不支援英文以外的特殊字元和語言。您將收到不支援字元和語言的一般錯誤訊息

    ```
    Full-text search conditions with special characters or languages other than English
     are not currently supported in Babelfish
    ```

  - `column_list` 之類的多個欄
  - PROPERTY 屬性
  - `prefix_term`, `generation_term`, `generic_proximity_term`, `custom_proximity_term`, 和 `weighted_term`
  - 不支援布林值運算子，使用時您會收到下列錯誤訊息：

    ```
    boolean operators not supported
    ```

  - 不支援帶有句點的識別符名稱。
- 目前，Babelfish 中不支援 `CREATE FULLTEXT INDEX` 子句的下列選項。
  - [TYPE COLUMN type_column_name]
  - [LANGUAGE language_term]
  - [STATISTICAL_SEMANTICS]
  - 目錄檔案群組選項

- 不支援全文目錄的建立。建立全文索引不需要全文目錄。
- CREATE FULLTEXT INDEX 不支援帶有句點的識別符名稱。
- Babelfish 目前不支援在搜尋字串中的連續特殊字元。您會在使用時收到下列錯誤訊息：

```
Consecutive special characters in the full-text search condition are not currently
 supported in Babelfish
```

Babelfish 支援地理空間資料類型

從 3.5.0 和 4.1.0 版開始，Babelfish 包含下列兩種空間資料類型的支援：

- 幾何資料類型：此資料類型適用於存放平面或歐幾里德 (平地球) 資料。
- 地理資料類型：此資料類型適用於存放橢圓或圓地球資料，例如 GPS 緯度和經度座標。

這些資料類型允許空間資料的儲存和處理，但有限制。

了解 Babelfish 中的地理空間資料類型

- 檢視、程序和資料表等各種資料庫物件都支援地理空間資料類型。
- 支援點資料類型，將位置資料存放為經緯度和有效的空間參考系統識別符 (SRID) 定義的點。點可能包含 Z （上升）、M （測量） 值，而且可以是空的。
- 支援由一系列點和連接它們的線段以及有效的空間參考系統識別符 (SRID) 定義的 linestring 資料類型 （從 5.4.0 版開始）。linestring 可能包含具有 Z （上升）、M （測量） 值的點，並且可以是空的。
- 透過 JDBC、ODBC、DOTNET 和 PYTHON 等驅動程式連線至 Babelfish 的應用程式可以利用此地理空間功能。

Babelfish 中支援的幾何資料類型函數

- STGeomFromText (***geometry_tagged_text***、SRID)：使用已知文字 (WKT) 表示法建立幾何執行個體。
- STPointFromText (***point_tagged_text***、SRID)：使用 WKT 表示法建立點執行個體。
- 點 (X、Y、SRID)：使用 x 和 y 座標的浮點數值建立點執行個體。
- <geometry_instance>.STAsText ( )：從幾何執行個體擷取 WKT 表示法。
- <geometry_instance>。STAsBinary () – 從幾何執行個體擷取 WKB 表示法。

- <geometry_instance>。STArea ( ) – 計算幾何執行個體的總表面積。
- <geometry_instance>。STSrid () – 擷取幾何執行個體的空間參考識別碼 (SRID)。
- <geometry_instance>。STDimension () – 擷取幾何執行個體的空間維度。
- <geometry_instance>。STIsEmpty () – 檢查幾何執行個體是否為空。
- <geometry_instance>。STIsClosed () – 檢查幾何執行個體是否關閉。
- <geometry_instance>。STIsValid () – 檢查幾何執行個體是否有效。
- <geometry_instance>.STDistance (other_geometry)：計算兩個幾何執行個體之間的距離。
- <geometry_instance>。STEquals (other_geometry) – 檢查幾何執行個體是否代表與另一個幾何執行個體相同的設定點。
- <geometry_instance>。STContains (other_geometry) – 檢查幾何執行個體是否包含其他_geometry 執行個體。
- <geometry_instance>。STDisjoint (other_geometry) – 檢查兩個幾何執行個體是否沒有共同點。
- <geometry_instance>。STIntersects (other_geometry) – 檢查兩個幾何執行個體是否空間相交。
- <geometry_instance>.STX：擷取幾何執行個體的 X 座標 (經度)。
- <geometry_instance>.STY：擷取幾何執行個體的 Y 座標 (緯度)。

從 4.7.0 和 5.3.0 版開始，Babelfish 包含對下列空間資料函數的支援：

- <geometry_instance>。M – 擷取幾何執行個體的 M 座標。
- <geometry_instance>。Z – 擷取幾何執行個體的 Z 座標。
- <geometry_instance>。HasM – 檢查幾何執行個體是否具有至少一個 M 值。
- <geometry_instance>。HasZ – 檢查幾何執行個體是否具有至少一個 Z 值。

從 5.4.0 版開始，Babelfish 包含對下列空間資料函數的支援：

- STLineFromText (***linestring_tagged_text***、SRID) – 使用 WKT 表示法建立 linestring 執行個體。

Babelfish 中支援的地理資料類型函數

- STGeomFromText (***geography_tagged_text***、SRID)：使用 WKT 表示法建立地理執行個體。
- STPointFromText (***point_tagged_text***、SRID)：使用 WKT 表示法建立點執行個體。
- 點 (緯度、經度、SRID)：使用緯度和經度的浮點數值建立點執行個體。

- <geography_instance>.STAsText ( )：從地理執行個體擷取 WKT 表示法。
- <geography_instance>。STAsBinary () – 從地理位置執行個體擷取 WKB 表示法。
- <geography_instance>。STArea ( ) – 計算地理執行個體的總表面積。
- <geography_instance>。STSrid () – 擷取地理執行個體的空間參考識別碼 (SRID)。
- <geography_instance>。STDimension () – 擷取地理執行個體的空間維度。
- <geography_instance>。STIsEmpty () – 檢查地理位置執行個體是否為空。
- <geography_instance>。STIsClosed () – 檢查地理位置執行個體是否關閉。
- <geography_instance>。STIsValid( ) – 檢查地理位置執行個體是否有效。
- <geography_instance>.STDistance (other_geography)：計算兩個地理執行個體之間的距離。
- <geography_instance>。STEquals (other_geography) – 檢查地理位置執行個體是否代表與另一個地理位置執行個體相同的設定點。
- <geography_instance>。STContains (other_geography) – 檢查地理位置執行個體是否包含其他 _geography 執行個體。
- <geography_instance>。STDisjoint (other_geography) – 檢查兩個地理執行個體是否沒有共同點。
- <geography_instance>。STIntersects (other_geography) – 檢查兩個地理執行個體是否空間相交。
- <geography_instance>.Lat：擷取地理執行個體的緯度值。
- <geography_instance>.Long：擷取地理執行個體的經度值。

從 4.7.0 和 5.3.0 版開始，Babelfish 包含對下列空間資料函數的支援：

- <geography_instance>。M – 擷取地理執行個體的 M 座標。
- <geography_instance>。Z – 擷取地理執行個體的 Z 座標。
- <geography_instance>。HasM – 檢查地理位置執行個體是否具有至少一個 M 值。
- <geography_instance>。HasZ – 檢查地理位置執行個體是否具有至少一個 Z 值。

從 5.4.0 版開始，Babelfish 包含對下列空間資料函數的支援：

- STLineFromText (***linestring_tagged_text***、SRID) – 使用 WKT 表示法建立 linestring 執行個體。

Babelfish 中地理空間資料類型的限制

- 點和 linestring 執行個體以外的幾何和地理類型目前不受支援：

  - CircularString
  - CompoundCurve
  - 多邊形
  - CurvePolygon
  - MultiPoint
  - MultiLineString
  - MultiPolygon
  - GeometryCollection
- 目前，針對地理空間資料類型，不支援空間索引。
- 目前針對這些資料類型僅支援列出的函數。如需詳細資訊，請參閱Babelfish 中支援的幾何資料類型函數及Babelfish 中支援的地理資料類型函數。
- 與 T-SQL 相比，地理資料的 STDistance 函數輸出可能會有細微的精確度變化。這是因為基礎 PostGIS 實作所致。如需詳細資訊，請參閱 ST_Distance
- 與 T-SQL 相比，幾何和地理資料的 STIsValid 函數輸出可能會有一些偏差。因此，在某些情況下 函數 - STDistance、STContains、STInstersects、STDisjoint、STDimension、STArea、STEquals 也可能偏離 T-SQL (傳回輸出而非擲出錯誤)。這是因為基礎 PostGIS 實作所致。如需詳細資訊，請參閱 ST_IsValid。
- 為獲得最佳效能，請使用內建的地理空間資料類型，而無需在 Babelfish 中建立額外的抽象層。
- 在 Babelfish 中，地理空間函數名稱會用作為關鍵字，只有在以預期方式使用時，才會執行空間操作。

了解 Babelfish 中的分割

從 4.3.0 版開始，Babelfish 推出使用有限支援的資料表和索引分割。下列各節提供的詳細資訊，與在 Babelfish 中建立分割區函數、定義分割區結構，以及實作分割資料表和索引有關。

主題

- Babelfish 中的分割簡介
- 限制和解決方法

Babelfish 中的分割簡介

- 分割區函數：

  - CREATE PARTITION FUNCTION：定義如何分割資料表或索引，方法是指定分割欄類型和每個分割區的值範圍。
  - DROP PARTITION FUNCTION：移除現有的分割區函數。
- 分割區結構：
  - CREATE PARTITION SCHEME：定義分割區和檔案群組之間的對應。

    > **Note**
    >
    > 在 Babelfish 中，會將檔案群組視為虛擬物件，檔案群組不代表實體儲存位置。

  - DROP PARTITION SCHEME：移除現有的分割區結構。
- 系統函數：
  - $PARTITION：此系統函數會傳回分割欄中指定值屬於指定分割資料表的分割區編號。
- 分割的資料表和索引：
  - CREATE TABLE ... ON partition_scheme_name (partition_column_name)：根據指定的分割區結構和分割欄建立分割的資料表。
  - CREATE INDEX ... ON partition_scheme_name (partition_column_name)：根據指定的分割區結構和分割欄建立分割的索引。
- 分割中繼資料的系統檢視：

  新增下列系統檢視，以提供與分割相關的中繼資料：
  - sys.destination_data_spaces
  - sys.partitions
  - sys.partition_functions
  - sys.partition_parameters
  - sys.partition_range_values
  - sys.partition_schemes

限制和解決方法

Babelfish 尚不支援下列 SQL Server 分割功能：

- ALTER PARTITION FUNCTION 與 ALTER PARTITION SCHEME。

> **Note**
>
> Babelfish 不支援分割和合併操作。在建立期間定義分割區函數中的所有分割區，因為您稍後無法新增或移除分割區。

- 作為分割欄的運算欄。
- 分割資料表的 `INSERT BULK` 和 BCP 公用程式。
- 分割區函數的 `LEFT` 界限選項。
- 分割區函數的 `SQL_VARIANT` 資料類型。
- `TRUNCATE TABLE ... WITH PARTITION`.
- `ALTER TABLE ... SWITCH PARTITION`.
- 非對齊的分割索引，例如分割區結構以及與分割資料表不同的分割區欄。
- 僅針對分割資料表上的完整載入任務，才支援從 Babelfish 來源進行 DMS 遷移。
- Babelfish 不支援這些語法選項，但提供解決方法：
  - 在 CREATE TABLE 陳述式中使用具有限制條件或索引的分割區結構。
  - ALTER TABLE ... ADD CONSTRAINT ... ON partition_scheme_name (partition_column_name)。

> **Note**
>
> 您可以將 `babelfishpg_tsql.escape_hatch_storage_on_partition` 逃生艙設為忽略。這可讓剖析器忽略與限制條件或索引搭配使用的分割區結構選項，而後端會為每個分割區建立個別限制條件或索引。

Babelfish 疑難排解

以下提供部分 Babelfish 資料庫叢集問題的疑難排解概念和解決方法。

主題

- 連線失敗

連線失敗

無法連線至執行 Babelfish 的新的 Aurora 資料庫叢集時，常見原因包括：

- 安全群組不允許存取 - 如果您無法連線至 Babelfish，請確保您已將 IP 地址新增至預設的 Amazon EC2 安全群組。您可以使用 https://checkip.amazonaws.com/ 來判斷 IP 地址，然後新增至 TDS 連接埠和 PostgreSQL 連接埠的傳入規則。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的將規則新增至安全群組。
- SSL 組態不符 — 如果在 Aurora PostgreSQL 上開啟 `rds.force_ssl` 參數 (設定為 1)，則用戶端必須透過 SSL 連接到 Babelfish。如果您的用戶端設定不正確，您會看到錯誤訊息，例如以下所示：

  ```
  Cannot connect to your-Babelfish-DB-cluster, 1433
  ----------------------
  ADDITIONAL INFORMATION:
  no pg_hba_conf entry for host "256.256.256.256", user "your-user-name",
  "database babelfish_db", SSL off (Microsoft SQL Server, Error: 33557097)
  ...
  ```

  此錯誤表示您的本地用戶端和 Babelfish 資料庫叢集之間可能存在 SSL 組態問題，並且叢集要求用戶端使用 SSL (`rds.force_ssl` 參數設定為 1)。如需設定 SSL 的詳細資訊，請參閱《Amazon RDS 使用者指南》中的將 SSL 與 PostgreSQL 資料庫執行個體搭配使用。

  如果您使用 SQL Server Management Studio (SSMS) 連接至 Babelfish，並且看到此錯誤，您可以在 Connection Properties (連接屬性) 窗格中選擇Encrypt connection (加密連線) 和 Trust server certificate (信任伺服器憑證) 連接選項，然後再試一次。這些設定會處理 SSMS 的 SSL 連接要求。

如需 Aurora 連線問題疑難排解的詳細資訊，請參閱無法連線至 Amazon RDS 資料庫執行個體。

停用 Babelfish

不再需要 Babelfish 時，您可以停用 Babelfish 功能。

一些注意事項：

- 在某些情況下，您可能在完成遷移至 Aurora PostgreSQL 之前停用 Babelfish。如果這樣做，但您的 DDL 依賴於 SQL Server 資料類型，或您在程式碼中使用任何 T-SQL 功能，則程式碼會失敗。
- 佈建 Babelfish 執行個體之後，如果您關閉 Babelfish 延伸，則無法在同一個叢集上再次佈建該資料庫。

若要停用 Babelfish，請修改參數群組，將 `rds.babelfish_status` 設定為 `OFF`。您可以將 `rds.babelfish_status` 設定為 `datatypesonly`，在 Babelfish 停用的情況下繼續使用 SQL Server 資料類型。

如果您在參數群組中停用 Babelfish，則使用該參數群組的所有叢集會失去 Babelfish 功能。

如需有關修改參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。如需 Babelfish 特定參數的資訊，請參閱Babelfish 的資料庫叢集參數群組設定。

管理 Babelfish for Aurora PostgreSQL 版本更新

Babelfish 是適用於 Aurora PostgreSQL 13.4 版及更高版本的選項。Babelfish 的更新適用於 Aurora PostgreSQL 資料庫引擎的某些新版本。如需詳細資訊，請參閱 Aurora PostgreSQL 版本備註。

> **Note**
>
> 在 Aurora PostgreSQL 13 的任何版本上執行的 Babelfish 資料庫叢集無法升級至 Aurora PostgreSQL 14.3、14.4 和 14.5。此外，Babelfish 不支援 13.x 到 15.x 的直接升級。您必須先將 13.x 資料庫叢集升級至 14.6 及更高版本，再升級至 15.x 版本。

如需不同 Babelfish 版本之間支援的功能清單，請參閱 Babelfish 各版本支援的功能。

如需目前不支援的功能清單，請參閱 Babelfish 中不支援的功能。

您可以如下所示，使用 describe-db-engine-versions AWS CLI 命令查詢 AWS 區域，以取得支援 Babelfish 的 Aurora PostgreSQL 版本清單。

對於 Linux、macOS 或 Unix：

```
$ aws rds describe-db-engine-versions --region us-east-1 \
    --engine aurora-postgresql \
    --query '*[]|[?SupportsBabelfish==`true`].[EngineVersion]' \
    --output text
```

如需詳細資訊，請參閱 AWS CLI 命令參考中的 describe-db-engine-versions。

在下列主題中，您可以了解如何識別 Aurora PostgreSQL 資料庫叢集上執行的 Babelfish 版本，以及如何升級至新版本。

內容

- 識別您的 Babelfish 版本
- 將 Babelfish 叢集升級至新版本
  - 將 Babelfish 升級至新的次要版本
  - 將 Babelfish 升級至新的主要版本
    - 將 Babelfish 升級至新的主要版本之前
    - 執行主要版本升級

    - 升級至新的主要版本之後
    - 範例：將 Babelfish 資料庫叢集升級至主要版本
- 使用 Babelfish 產品版本參數
    - 設定 Babelfish 產品版本參數
    - 受影響的查詢和參數
    - 包含 babelfishpg_tsql.version 參數的界面

識別您的 Babelfish 版本

您可以查詢 Babelfish 來尋找 Babelfish 版本、Aurora PostgreSQL 版本，以及相容的 Microsoft SQL Server 版本的詳細資訊。您可以使用 TDS 連接埠或 PostgreSQL 連接埠。

- To use the TDS port to query for version information
- To use the PostgreSQL port to query for version information

使用 TDS 連接埠查詢版本資訊

1. 使用 `sqlcmd` 或 `ssms` 以連線至 Babelfish 資料庫叢集的端點。

```
sqlcmd -S bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com,1433 -U
    login-id -P password -d db_name
```

2. 若要識別 Babelfish 版本，請執行下列查詢：

```
1> SELECT CAST(serverproperty('babelfishversion') AS VARCHAR)
2> GO
```

   此查詢傳回類似以下的結果。

```
serverproperty
------------------------------
3.4.0

(1 rows affected)
```

3. 若要識別 Aurora PostgreSQL 資料庫叢集的版本，請執行下列查詢：

```
1> SELECT aurora_version() AS aurora_version
```

```
2> GO
```

此查詢傳回類似以下的結果。

```
aurora_version


------------------------------------------------
15.5.0

(1 rows affected)
```

4. 若要識別相容的 Microsoft SQL Server 版本，請執行下列查詢：

```
1> SELECT @@VERSION AS version
2> GO
```

此查詢傳回類似以下的結果。

```
Babelfish for Aurora PostgreSQL with SQL Server Compatibility - 12.0.2000.8
Dec 7 2023 09:43:06
Copyright (c) Amazon Web Services
PostgreSQL 15.5 on x86_64-pc-linux-gnu (Babelfish 3.4.0)

(1 rows affected)
```

以顯示 Babelfish 和 Microsoft SQL Server 之間的一個小差異為例，您可以執行以下查詢。在 Babelfish 上，查詢會傳回 1，而在 Microsoft SQL Server 上，查詢會傳回 NULL。

```
SELECT CAST(serverproperty('babelfish') AS VARCHAR) AS runs_on_babelfish
```

您也可以使用 PostgreSQL 連接埠來取得版本資訊，如以下程序所示。

使用 PostgreSQL 連接埠查詢版本資訊

1. 使用 psql 或 pgAdmin 連線至 Babelfish 資料庫叢集的端點。

```
psql host=bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com
     port=5432 dbname=babelfish_db user=sa
```

2. 開啟 psql 的擴充功能 (\x)，增加輸出的可讀性。

```
babelfish_db=> \x
babelfish_db=> SELECT
babelfish_db=> aurora_version() AS aurora_version,
babelfish_db=> version() AS postgresql_version,
babelfish_db=> sys.version() AS Babelfish_compatibility,
babelfish_db=> sys.SERVERPROPERTY('BabelfishVersion') AS Babelfish_Version;
```

此查詢會傳回類似以下的輸出：

```
-[ RECORD 1 ]-----------
+-------------------------------------------------------------------------------------------
aurora_version          | 15.5.0
postgresql_version      | PostgreSQL 15.5 on x86_64-pc-linux-gnu, compiled by
 x86_64-pc-linux-gnu-gcc (GCC) 9.5.0, 64-bit
babelfish_compatibility | Babelfish for Aurora Postgres with SQL Server
 Compatibility - 12.0.2000.8                    +
                         | Dec 7 2023 09:43:06
                                    +
                         | Copyright (c) Amazon Web Services
                                     +
                         | PostgreSQL 15.5 on x86_64-pc-linux-gnu (Babelfish 3.4.0)
babelfish_version       | 3.4.0
```

將 Babelfish 叢集升級至新版本

新版的 Babelfish 可與 13.4 版後的部份新 Aurora PostgreSQL 資料庫引擎版本搭配使用。Babelfish 的每個新版本都有自己的版本編號。與 Aurora PostgreSQL 一樣，Babelfish 會針對版本使用 *major*.*minor*.*patch* 命名方式。例如，第一個 Babelfish 版本 (Babelfish 1.0.0 版) 可用作 Aurora PostgreSQL 13.4.0 的一部分。

Babelfish 不需要個別的安裝程序。如建立 Babelfish for Aurora PostgreSQL DB 叢集中所述，Turn on Babelfish (開啟 Babelfish) 是您在建立 Aurora PostgreSQL 資料庫叢集時選擇的一個選項。

同樣地，您無法獨立於支援的 Aurora 資料庫叢集升級 Babelfish。若要將現有的 Babelfish for Aurora PostgreSQL 資料庫叢集升級至新的 Babelfish 版本，請將 Aurora PostgreSQL 資料庫叢集升級至支援您要使用之 Babelfish 版本的新版本。升級所遵循的程序取決於支援 Babelfish 部署的 Aurora PostgreSQL 版本，如下所示。

主要版本升級

您必須將下列 Aurora PostgreSQL 版本升級至 Aurora PostgreSQL 14.6 和更高版本，才能升級至 Aurora PostgreSQL 15.2 版。

- Aurora PostgreSQL 13.8 和所有更新的版本
- Aurora PostgreSQL 13.7.1 和所有更新的次要版本
- Aurora PostgreSQL 13.6.4 和所有更新的次要版本

您可以將 Aurora PostgreSQL 14.6 和更高版本升級至 Aurora PostgreSQL 15.2 和更高版本。

將 Aurora PostgreSQL 資料庫叢集升級至新的主要版本，涉及數個初步任務。如需更多詳細資訊，請參閱 執行主要版本升級。若要成功升級 Babelfish for Aurora PostgreSQL 資料庫叢集，您需要為新的 Aurora PostgreSQL 版本建立自訂資料庫叢集參數群組。這個新參數群組必須包含與您要升級之叢集相同的 Babelfish 參數設定。如需主要版本升級來源和目標的詳細資訊和資料表，請參閱 將 Babelfish 升級至新的主要版本。

次要版本升級和修補程式

次要版本和修補程式不需要建立新的資料庫叢集參數群組，即可進行升級。次要版本和修補程式可以使用次要版本升級程序，無論是自動套用還是手動套用。如需版本來源和目標的詳細資訊和資料表，請參閱 將 Babelfish 升級至新的次要版本。

> **Note**
>
> 在執行主要或次要升級之前，請將所有擱置中維護任務套用至 Babelfish for Aurora PostgreSQL 叢集。

主題

- 將 Babelfish 升級至新的次要版本
- 將 Babelfish 升級至新的主要版本

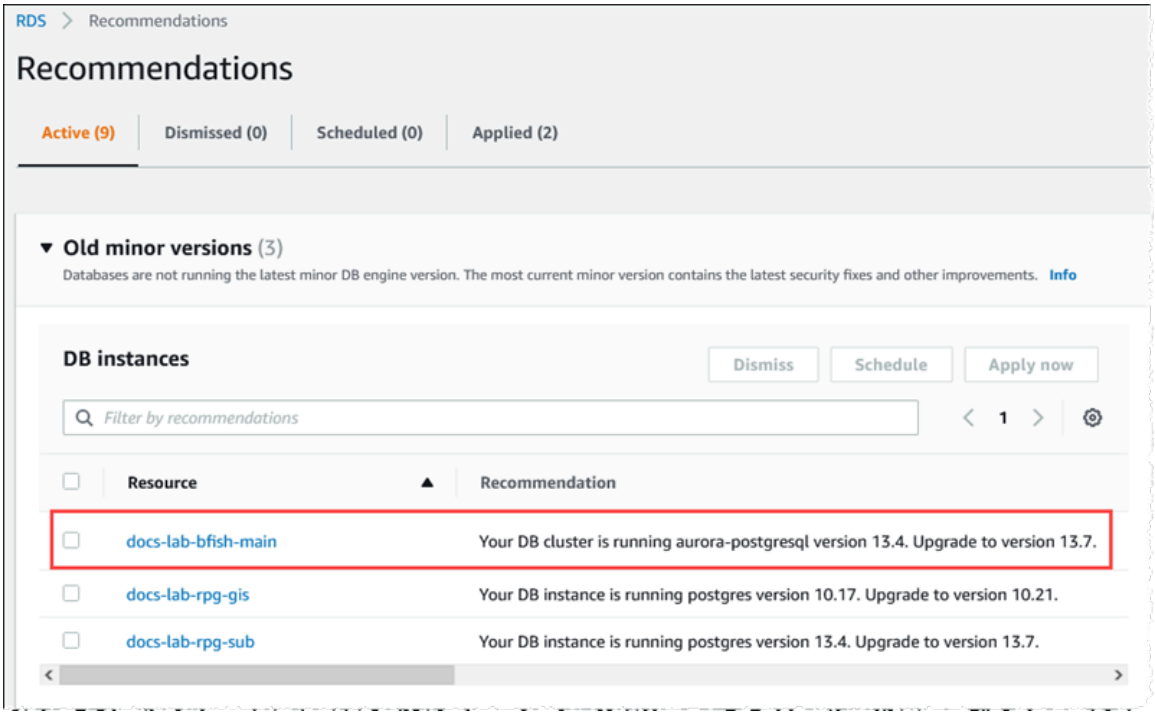
將 Babelfish 升級至新的次要版本

次要版本更新旨在維持回溯相容性。不過，在某些情況下，重要的安全修正或重要的錯誤修正可能不具備完全回溯相容性。修補程式版本中包含其發行後次要版本的重要修正。例如，第一個發行的 Aurora PostgreSQL 13.4 的版本標籤為 Aurora PostgreSQL 13.4.0。至今已發行該次要版本的數個修補程式，

包括 Aurora PostgreSQL 13.4.1、13.4.2 和 13.4.4。您可以在該版本的 Aurora PostgreSQL 版本備註頂端的 Patch releases (修補程式版本清單) 中找到適用於每個 Aurora PostgreSQL 版本的修補程式。如需範例，請參閱《Aurora PostgreSQL 版本備註》中的 PostgreSQL 14.3。

如果您的 Aurora PostgreSQL 資料庫叢集設定為搭配 Auto minor version upgrade (自動次要版本升級) 選項，則您的 Babelfish for Aurora PostgreSQL 資料庫叢集會在叢集的維護時段期間自動升級。若要進一步了解自動次要版本升級 (AmVU) 以及如何使用它，請參閱 Aurora 資料庫叢集的自動次要版本升級。如果您的叢集不是 AmVU，則您可以透過回應維護任務，或修改叢集以使用新版本，手動將 Babelfish for Aurora PostgreSQL 資料庫叢集升級至新的次要版本。

當您選擇要安裝的 Aurora PostgreSQL 版本，以及當您在 中檢視現有的 Aurora PostgreSQL 資料庫叢集時 AWS 管理主控台，版本只會顯示 *major*.*minor* 數字。例如，來自主控台針對具有 Aurora PostgreSQL 13.4 的現有 Babelfish for Aurora PostgreSQL 資料庫叢集的下圖建議將叢集升級至 13.7 版，這是 Aurora PostgreSQL 的新次要版本。



若要取得完整的版本詳細資訊 (包括####等級)，您可以使用 `aurora_version` Aurora PostgreSQL 函數查詢 Aurora PostgreSQL 資料庫叢集。如需詳細資訊，請參閱 Aurora PostgreSQL 函數參考 中的 aurora_version。您可以在To use the PostgreSQL port to query for version information的識別您的 Babelfish 版本程序中尋找使用函數的範例。

下表顯示 Aurora PostgreSQL 和 Babelfish 版本，以及可支援次要版本升級程序的可用目標版本。

目前來源版本	最新升級目標
Aurora PostgreSQL (Babelfish)	Aurora PostgreSQL (Babelfish)
17.4 (5.1)	17.5 (5.2)
16.8 (4.5)	16.9 (4.6)
16.6 (4.4.0)	16.9 (4.6)、16.8 (4.5.0)
16.4 (4.3.0)	16.9 (4.6)、16.8 (4.5.0)、16.6 (4.4.0)
16.3 (4.2.0)	16.9 (4.6)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)
16.2 (4.1.0)	16.9 (4.6)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)、16.3 (4.2.0)
16.1 (4.0.0)	16.9 (4.6)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)、16.3 (4.2.0)、16.2 (4.1.0)
15.12 (3.9.0)	15.13 (3.10)
15.10 (3.8.0)	15.13 (3.10)、15.12 (3.9.0)
15.8 (3.7.0)	15.13 (3.10)、15.12 (3.9.0)、15.10 (3.8.0)
15.7 (3.6.0)	15.13 (3.10)、15.12 (3.9.0)、15.10 (3.8.0)、15.8 (3.7.0)
15.6 (3.5.0)	15.13 (3.10)、15.12 (3.9.0)、15.10 (3.8.0)、15.8 (3.7.0)、15.7 (3.6.0)
15.5 (3.4.0)	15.13 (3.10)、15.12 (3.9.0)、15.10 (3.8.0)、15.8 (3.7.0)、15.7 (3.6.0)、15.6 (3.5.0)
15.4 (3.3.0)	15.13 (3.10)、15.12 (3.9.0)、15.10 (3.8.0)、15.8 (3.7.0)、15.7 (3.6.0)、15.6 (3.5.0)、15.5 (3.4.0)

目前來源版本	最新升級目標
15.12 (3.9.0)、15.3 (3.2.0)	15.13 (3.10)、15.12 (3.9.0)、15.10 (3.8.0)、15.8 (3.7.0)、15.7 (3.6.0)、15.6 (3.5.0)、15.5 (3.4.0)、15.4 (3.3.0)
15.2 (3.1.0)	15.13 (3.10)、15.12 (3.9.0)、15.10 (3.8.0)、15.8 (3.7.0)、15.7 (3.6.0)、15.6 (3.5.0)、15.5 (3.4.0)、15.4 (3.3.0)、15.3 (3.2.0)
14.17 (2.12.0)	14.18 (2.13.0)
14.15 (2.11.0)	14.18 (2.13.0)、14.17 (2.12.0)
14.13 (2.10.0)	14.18 (2.13.0)、14.17 (2.12.0)、14.15 (2.11.0)
14.12 (2.9.0)	14.18 (2.13.0)、14.17 (2.12.0)、14.15 (2.11.0)、14.13 (2.10.0)
14.11 (2.8.0)	14.18 (2.13.0)、14.17 (2.12.0)、14.15 (2.11.0)、14.13 (2.10.0)、14.12 (2.9.0)
14.10 (2.7.0)	14.18 (2.13.0)、14.17 (2.12.0)、14.15 (2.11.0)、14.13 (2.10.0)、14.12 (2.9.0)、14.11 (2.8.0)
14.9 (2.6.0)	14.18 (2.13.0)、14.17 (2.12.0)、14.15 (2.11.0)、14.13 (2.10.0)、14.12 (2.9.0)、14.11 (2.8.0)、14.10 (2.7.0)
14.8 (2.5.0)	14.18 (2.13.0)、14.17 (2.12.0)、14.15 (2.11.0)、14.13 (2.10.0)、14.12 (2.9.0)、14.11 (2.8.0)、14.10 (2.7.0)、14.9 (2.6.0)
14.7 (2.4.0)	14.18 (2.13.0)、14.17 (2.12.0)、14.15 (2.11.0)、14.13 (2.10.0)、14.12 (2.9.0)、14.11 (2.8.0)、14.10 (2.7.0)、14.9 (2.6.0)、14.8 (2.5.0)

目前來源版本	最新升級目標
14.6 (2.3.0)	14.18 (2.13.0)、14.17 (2.12.0)、14.15 (2.11.0)、14.13 (2.10.0)、14.12 (2.9.0)、14.11 (2.8.0)、14.10 (2.7.0)、14.9 (2.6.0)、14.8 (2.5.0)、14.7 (2.4.0)
14.5 (2.2.0)	14.18 (2.13.0)、14.17 (2.12.0)、14.15 (2.11.0)、14.13 (2.10.0)、14.12 (2.9.0)、14.11 (2.8.0)、14.10 (2.7.0)、14.9 (2.6.0)、14.8 (2.5.0)、14.7 (2.4.0)、14.6 (2.3.0)
14.3 (2.1.0)	14.18 (2.13.0)、14.6 (2.3.0)
13.8 (1.4.0)	13.9 (1.5.0)
13.7 (1.3.0)	13.9 (1.5.0)、13.8 (1.4.0)

將 Babelfish 升級至新的主要版本

對於主要版本升級，您必須先將 Babelfish for Aurora PostgreSQL 資料庫叢集升級至支援主要版本升級的版本。若要達成此目的，請將修補程式更新或次要版本升級套用至資料庫叢集。如需詳細資訊，請參閱 將 Babelfish 升級至新的次要版本。

下表顯示可支援主要版本升級的 Aurora PostgreSQL 版本和 Babelfish 版本。

目前來源版本	最新升級目標
Aurora PostgreSQL (Babelfish)	Aurora PostgreSQL (Babelfish)
16.9 (4.6.0)	17.5 (5.2.0)
16.8 (4.5.0)	17.5 (5.2.0)、17.4 (5.1.0)
16.6 (4.4.0)	17.5 (5.2.0)、17.4 (5.1.0)
16.4 (4.3.0)	17.5 (5.2.0)、17.4 (5.1.0)
16.3 (4.2.0)	17.5 (5.2.0)、17.4 (5.1.0)

目前來源版本	最新升級目標
16.2 (4.1.0)	17.5 (5.2.0)、17.4 (5.1.0)
16.1 (4.0.0)	17.5 (5.2.0)、17.4 (5.1.0)
15.13 (3.10)	17.5 (5.2.0) 16.9 (4.6.0)
15.12 (3.9.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)
15.10 (3.8.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)
15.8 (3.7.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)
15.7 (3.6.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)、16.3 (4.2.0)
15.6 (3.5.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)、16.3 (4.2.0)、16.2 (4.1.0)
15.5 (3.4.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)、16.3 (4.2.0)、16.2 (4.1.0)、16.1 (4.0.0)

目前來源版本	最新升級目標
15.4 (3.3.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)、16.3 (4.2.0)、16.2 (4.1.0)、16.1 (4.0.0)
15.3 (3.2.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)、16.3 (4.2.0)、16.2 (4.1.0)、16.1 (4.0.0)
15.2 (3.1.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)、16.3 (4.2.0)、16.2 (4.1.0)、16.1 (4.0.0)
14.18 (2.13.0)	17.5 (5.2.0) 16.9 (4.6.0) 15.13 (3.10.0)
14.17 (2.12.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0) 15.13 (3.10.0)、15.12 (3.9.0)
14.15 (2.11.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0) 15.13 (3.10.0)、15.12 (3.9.0)、15.10 (3.8.0)

目前來源版本	最新升級目標
14.13 (2.10.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0) 15.13 (3.10.0)、15.12 (3.9.0)、15.10 (3.8.0)、15.8 (3.7.0)
14.12 (2.9.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)、16.3 (4.2.0) 15.13 (3.10.0)、15.12 (3.9.0)、15.10 (3.8.0)、15.8 (3.7.0)、15.7 (3.6.0)
14.11 (2.8.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)、16.3 (4.2.0)、16.2 (4.1.0) 15.13 (3.10.0)、15.12 (3.9.0)、15.10 (3.8.0)、15.8 (3.7.0)、15.7 (3.6.0)、15.6 (3.5.0)
14.10 (2.7.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)、16.3 (4.2.0)、16.2 (4.1.0)、16.1 (4.0.0) 15.13 (3.10.0)、15.12 (3.9.0)、15.10 (3.8.0)、15.8 (3.7.0)、15.7 (3.6.0)、15.6 (3.5.0)、15.5 (3.4.0)

目前來源版本	最新升級目標
14.9 (2.6.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)、16.3 (4.2.0)、16.2 (4.1.0)、16.1 (4.0.0) 15.13 (3.10.0)、15.12 (3.9.0)、15.10 (3.8.0)、15.8 (3.7.0)、15.7 (3.6.0)、15.6 (3.5.0)、15.5 (3.4.0)、15.4 (3.3.0)
14.8 (2.5.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)、16.3 (4.2.0)、16.2 (4.1.0)、16.1 (4.0.0) 15.13 (3.10.0)、15.12 (3.9.0)、15.10 (3.8.0)、15.8 (3.7.0)、15.7 (3.6.0)、15.6 (3.5.0)、15.5 (3.4.0)、15.4 (3.3.0)、15.3 (3.2.0)
14.7 (2.4.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)、16.3 (4.2.0)、16.2 (4.1.0)、16.1 (4.0.0) 15.13 (3.10.0)、15.12 (3.9.0)、15.10 (3.8.0)、15.8 (3.7.0)、15.7 (3.6.0)、15.6 (3.5.0)、15.5 (3.4.0)、15.4 (3.3.0)、15.3 (3.2.0)、15.2 (3.1.0)

目前來源版本	最新升級目標
14.6 (2.3.0)	17.5 (5.2.0)、17.4 (5.1.0) 16.9 (4.6.0)、16.8 (4.5.0)、16.6 (4.4.0)、16.4 (4.3.0)、16.3 (4.2.0)、16.2 (4.1.0)、16.1 (4.0.0) 15.13 (3.10.0)、15.12 (3.9.0)、15.10 (3.8.0)、15.8 (3.7.0)、15.7 (3.6.0)、15.6 (3.5.0)、15.5 (3.4.0)、15.4 (3.3.0)、15.3 (3.2.0)、15.2 (3.1.0)
13.9 (1.5.0)	14.6 (2.3.0)
13.8 (1.4.0)	14.6 (2.3.0)
13.7 (1.3.0)	14.6 (2.3.0)

將 Babelfish 升級至新的主要版本之前

升級可能涉及短暫的中斷。因此，建議您在維護時段或其他低用量期間執行或安排升級。

執行主要版本升級之前

1. 使用識別您的 Babelfish 版本中概述的命令，識別現有 Aurora PostgreSQL 資料庫叢集的 Babelfish 版本。Aurora PostgreSQL 版本和 Babelfish 版本資訊是由 PostgreSQL 處理，因此請遵循To use the PostgreSQL port to query for version information程序中的詳細步驟來取得詳細資訊。
2. 驗證您的版本是否支援主要版本升級。如需支援主要版本升級功能的版本清單，請參閱 將 Babelfish 升級至新的次要版本 並執行必要的升級前任務。

   例如，如果您的 Babelfish 版本是在 Aurora PostgreSQL 13.5 資料庫叢集上執行，而您想要升級至 Aurora PostgreSQL 15.2，請先套用所有次要版本和修補程式，將您的叢集升級至 Aurora PostgreSQL 14.6 或更高版本。當您的叢集為 14.6 或更高版本時，請繼續進行主要版本升級程序。
3. 建立目前 Babelfish 資料庫叢集的手動快照做為備份。備份可讓您將叢集還原至其 Aurora PostgreSQL 版本 (Babelfish 版本)，並將所有資料還原至升級前的狀態。如需詳細資訊，請參

閱建立資料庫叢集快照。如果您決定將此叢集還原至升級前的狀態，請務必讓現有的自訂資料庫叢集參數群組可再次使用。如需詳細資訊，請參閱從資料庫叢集快照還原及參數群組考量。

4. 針對目標 PostgreSQL 資料庫版本準備自訂資料庫叢集參數群組。從目前 Babelfish for Aurora PostgreSQL 資料庫叢集複製 Babelfish 參數的設定。若要尋找所有 Babelfish 參數的清單，請參閱 Babelfish 的資料庫叢集參數群組設定。對於主要版本升級，下列參數需要與來源資料庫叢集相同的設定。若要成功升級，所有設定都必須相同。

   - rds.babelfish_status
   - babelfishpg_tds.tds_default_numeric_precision
   - babelfishpg_tds.tds_default_numeric_scale
   - babelfishpg_tsql.database_name
   - babelfishpg_tsql.default_locale
   - babelfishpg_tsql.migration_mode
   - babelfishpg_tsql.server_collation_name

   > **Warning**
   >
   > 如果新 Aurora PostgreSQL 版本的自訂資料庫叢集參數群組中的 Babelfish 參數設定與您要升級之叢集的參數值不符，則 `ModifyDBCluster` 操作會失敗。`InvalidParameterCombination` 錯誤訊息會顯示在 或來自 `modify-db-cluster` AWS CLI 命令的輸出 AWS 管理主控台 中。

5. 使用 AWS 管理主控台 或 AWS CLI 建立自訂資料庫叢集參數群組。針對您要用於升級的 Aurora PostgreSQL 版本選擇適用的 Aurora PostgreSQL 系列。

   > **Tip**
   >
   > 參數群組是在 AWS 區域 層級管理。使用 時 AWS CLI，您可以使用預設區域來設定 ，而不是在 命令`--region`中指定 。若要進一步了解如何使用 AWS CLI，請參閱AWS Command Line Interface 《 使用者指南》中的快速設定。

執行主要版本升級

1. 將 Aurora PostgreSQL 資料庫叢集升級至新的主要版本。如需詳細資訊，請參閱將 Aurora PostgreSQL 引擎升級為新的主要版本。

2. 重新啟動叢集的寫入器執行個體，以便參數設定可以生效。

升級至新的主要版本之後

將主要版本升級至新的 Aurora PostgreSQL 版本之後，具有 IDENTITY 資料欄的資料表中的 IDENTITY 值可能會比升級前的值大 (+32)。結果是，當下一資料列插入到這樣的資料表時，產生的身分資料欄值會跳至 +32 數字，並從那裡開始序列。這種情況不會對 Babelfish 資料庫叢集的函數產生負面影響。不過，如果需要，您可以根據資料欄的最大值重設序列物件。若要這樣做，請使用 sqlcmd 或另一個 SQL Server 用戶端連線至 Babelfish 寫入器執行個體上的 T-SQL 連接埠。如需詳細資訊，請參閱使用 SQL Server 用戶端來連線至資料庫叢集。

```
sqlcmd -S bfish-db.cluster-123456789012.aws-region.rds.amazonaws.com,1433 -U
     sa -P ******** -d dbname
```

連線後，請使用下列 SQL 命令來產生陳述式，您可以將其用來植入相關聯的序列物件。這個 SQL 命令同時適用於單一資料庫和多資料庫 Babelfish 組態。如需這兩個部署模型的詳細資訊，請參閱 搭配單一資料庫或多個資料庫來使用 Babelfish。

```
DECLARE @schema_prefix NVARCHAR(200) = ''
IF current_setting('babelfishpg_tsql.migration_mode') = 'multi-db'
    SET @schema_prefix = db_name() + '_'
SELECT 'SELECT setval(pg_get_serial_sequence(''' + @schema_prefix +
 schema_name(tables.schema_id)
    + '.' + tables.name + ''', ''' + columns.name + '''),(select max(' + columns.name +
 ')
    FROM ' + schema_name(tables.schema_id) + '.' + tables.name + '));
    'FROM sys.tables tables JOIN sys.columns
    columns ON tables.object_id = columns.object_id
    WHERE columns.is_identity = 1
GO
```

查詢會產生一系列 SELECT 陳述式，然後您可以執行這些陳述式來重設最大 IDENTITY 值並關閉任何間隙。以下顯示使用範例 SQL Server 資料庫 (Northwind)，在 Babelfish 叢集上執行時的輸出。

```
-----------------------------------------------------------
SELECT setval(pg_get_serial_sequence('northwind_dbo.categories', 'categoryid'),(select
 max(categoryid)
    FROM dbo.categories));
```

```
SELECT setval(pg_get_serial_sequence('northwind_dbo.orders', 'orderid'),(select
 max(orderid)
    FROM dbo.orders));


SELECT setval(pg_get_serial_sequence('northwind_dbo.products', 'productid'),(select
 max(productid)
    FROM dbo.products));


SELECT setval(pg_get_serial_sequence('northwind_dbo.shippers', 'shipperid'),(select
 max(shipperid)
    FROM dbo.shippers));


SELECT setval(pg_get_serial_sequence('northwind_dbo.suppliers', 'supplierid'),(select
 max(supplierid)
    FROM dbo.suppliers));


(5 rows affected)
```

逐個執行陳述式以重設序列值。

範例：將 Babelfish 資料庫叢集升級至主要版本

在此範例中，您可以找到一系列 AWS CLI 命令，說明如何將執行 Babelfish 1.2.2 版的 Aurora PostgreSQL 13.6.4 資料庫叢集升級至 Aurora PostgreSQL 14.6。首先，針對 Aurora PostgreSQL 14 建立自訂資料庫叢集參數群組。接著，修改參數值以符合 Aurora PostgreSQL 第 13 版來源的參數值。最後，修改來源叢集來執行升級。如需詳細資訊，請參閱Babelfish 的資料庫叢集參數群組設定。在該主題中，您也可以找到使用 AWS 管理主控台 執行升級的相關資訊。

使用 create-db-cluster-parameter-group CLI 命令，針對新版本建立資料庫叢集參數群組。

針對 Linux、macOS 或 Unix：

```
aws rds create-db-cluster-parameter-group \
    --db-cluster-parameter-group-name docs-lab-babelfish-apg-14 \
    --db-parameter-group-family aurora-postgresql14 \
    --description 'New custom parameter group for upgrade to new major version' \
    --region us-west-1
```

當您發出此命令時，系統會在 AWS 區域中建立自訂資料庫叢集參數群組。您會看到類似下列的輸出。

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "docs-lab-babelfish-apg-14",
        "DBParameterGroupFamily": "aurora-postgresql14",
        "Description": "New custom parameter group for upgrade to new major version",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-west-1:111122223333:cluster-
pg:docs-lab-babelfish-apg-14"
    }
}
```

如需詳細資訊，請參閱在 Amazon Aurora 中建立資料庫叢集參數群組。

使用 modify-db-cluster-parameter-group CLI 命令來修改設定，使其與來源叢集相符。

在 Windows 中：

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name docs-lab-
babelfish-apg-14 ^
  --parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tds.tds_default_numeric_precision,ParameterValue=38,ApplyMethod=pen
reboot" ^
 "ParameterName=babelfishpg_tds.tds_default_numeric_scale,ParameterValue=8,ApplyMethod=pending-
reboot" ^
 "ParameterName=babelfishpg_tsql.database_name,ParameterValue=babelfish_db,ApplyMethod=pending-
reboot" ^
 "ParameterName=babelfishpg_tsql.default_locale,ParameterValue=en-
US,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tsql.migration_mode,ParameterValue=single-
db,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tsql.server_collation_name,ParameterValue=sql_latin1_general_cp1_ci
reboot"
```

回應看起來類似以下的內容。

```
{
    "DBClusterParameterGroupName": "docs-lab-babelfish-apg-14"
}
```

使用 modify-db-cluster CLI 命令來修改叢集，以使用新版本和新的自訂資料庫叢集參數群組。您也會指定 `--allow-major-version-upgrade` 引數，如下列範例所示。

```
aws rds modify-db-cluster \
--db-cluster-identifier docs-lab-bfish-apg-14 \
--engine-version 14.6 \
--db-cluster-parameter-group-name docs-lab-babelfish-apg-14 \
--allow-major-version-upgrade \
--region us-west-1 \
--apply-immediately
```

使用 reboot-db-instance CLI 命令，重新啟動叢集的寫入器執行個體，以便參數設定可以生效。

```
aws rds reboot-db-instance \
--db-instance-identifier docs-lab-bfish-apg-14-instance-1\
--region us-west-1
```

使用 Babelfish 產品版本參數

Babelfish 2.4.0 和 3.1.0 版有新的 Grand Unified Configuration (GUC) 參數，稱為 `babelfishpg_tds.product_version`。此參數可讓您將 SQL Server 產品版本號碼設定為 Babelfish 的輸出。

該參數是具有 4 個部分的版本 ID 字串，每個部分都應該用「.」分隔。

語法

```
Major.Minor.Build.Revision
```

- 主要版本：介於 11 到 16 之間的數字。
- 次要版本：介於 0 到 255 之間的數字。
- 建置版本：介於 0 到 65535 之間的數字。
- 修訂版本：0 和任何正數。

設定 Babelfish 產品版本參數

您必須使用叢集參數群組來設定主控台的 babelfishpg_tds.product_version 參數。如需如何修改資料庫叢集參數的詳細資訊，請參閱在 Amazon Aurora 中修改資料庫叢集參數群組中的參數。

若將產品版本參數設為無效值，變更就不會生效。雖然主控台可能會顯示新值，但參數會保留先前的值。請檢查引擎日誌檔案，取得錯誤訊息的詳細資訊。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name mydbparametergroup \
--parameters
 "ParameterName=babelfishpg_tds.product_version,ParameterValue=15.2.4000.1,ApplyMethod=immediat
```

在 Windows 中：

```
aws rds modify-db-cluster-parameter-group ^
--db-cluster-parameter-group-name mydbparametergroup ^
--parameters
 "ParameterName=babelfishpg_tds.product_version,ParameterValue=15.2.4000.1,ApplyMethod=immediat
```

受影響的查詢和參數

查詢/參數	結果	生效時間
選取 @@版本	回傳使用者定義的 SQL Server 版本 (babelfishpg_tsql.version 值 = 預設)	立即
選取 SERVERPROPERTY (「ProductVersion」)	回傳使用者定義的 SQL Server 版本	立即
選取 SERVERPROPERTY (「ProductMajorVersion」)	回傳使用者定義的 SQL Server 版本的主要版本	立即
PRELOGIN 回應訊息中的 VERSION 字符	伺服器以使用者定義的 SQL Server 版本回傳 PRELOGIN 訊息	當使用者建立新的工作階段時生效
使用 JDBC 時，LoginAck 中的 SQLServerVersion	DatabaseMetaData.getDatabaseProductVersion()	當使用者建立新的工作階段時生效

查詢/參數	結果	生效時間
	回傳使用者定義的 SQL Server 版本	

包含 babelfishpg_tsql.version 參數的界面

您可以使用參數 babelfishpg_tsql.version 和 babelfishpg_tds.product_version，來設定 @@版本的輸出。下列範例顯示這兩個參數的界面。

- 當 babelfishpg_tsql.version 參數為「預設」，而 babelfishpg_tds.product_version 為 15.0.2000.8。
  - @@版本的輸出 – 15.0.2000.8。
- 當 babelfishpg_tsql.version 參數設為 13.0.2000.8，且 babelfishpg_tds.product_version 參數為 15.0.2000.8。
  - @@版本的輸出 – 13.0.2000.8。

Babelfish for Aurora PostgreSQL 參考

主題

- Babelfish 各版本支援的功能
- Babelfish 中不支援的功能
- 使用 Babelfish for Aurora PostgreSQL
- Babelfish 支援 XML 資料類型方法

Babelfish 各版本支援的功能

在下表中，您可以找到不同 Babelfish 版本支援的所有功能清單。如需不支援的功能清單，請參閱 Babelfish 中不支援的功能。如需多個 Babelfish 版本的相關資訊，請參閱 Aurora PostgreSQL 版本備註。

Aurora 和 PostgreSQL 功能

在下表中，您可以找到不同 Babelfish 版本支援的 Aurora 和 PostgreSQL 功能。

Aurora 和 PostgreSQL 功能	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
Aurora ML 服務	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
使用 Kerberos 進行資料庫身分驗證 Directory Service	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
傾印和還原	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
pg_stat_statement 延伸模組	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
pgvector																																

| Aurora 和 PostgreSQL 功能 | 5.5.4.4.4.4.4.4.4.3.3.3.3.3.3.3.3.3.2.2.2

T-SQL 功能或語法	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
ALTER 使用者...使用登入	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
ALTER VIEW	是	否	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
CREATE FUNCTION 中的 AS 關鍵字	是	是	是	是	是	是	是	是	否	否	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
AT TIME ZONE 子句	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
作為連結伺服器的 Babelfish 執行個體	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否
比較運算子 !< 和 !>	是	是	是	是	是	是	是	是	否	否	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
CREATE INDEX ... ON partition _scheme_name (partition_column_ name)	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
在 SQL Server 檢視上的 CREATE Instead of 觸發條件 (DML)	是	是	是	是	是	是	是	否	否	否	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
CREATE PARTITION FUNCTION	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否

| T-SQL 功能或語法 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2

T-SQL 功能或語法	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
使用 bcp 用戶端公用程式進行資料遷移	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
資料類型 TIMESTAMP、ROWVERSION (如需用量資訊，請參閱具備受限實作的功能	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
在預存程序和函數之呼叫中的 DEFAULT 關鍵字	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
DBCC CHECKIDENT	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
DROP DATABASE	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
DROP IF EXISTS (適用於 SCHEMA、DATABASE 和 USER 物件)	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
DROP INDEX 索引 ON schema.table	是	是	是	是	是	是	是	是	否	否	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
DROP INDEX schema.table.index	是	是	是	是	是	是	是	是	否	否	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否

T-SQL 功能或語法	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
DROP PARTITION FUNCTION	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
DROP PARTITION SCHEME	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
DROP ROLE	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
ENABLE/DISABLE TRIGGER	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
FULL TEXT SEARCH	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
使用 CONTAINS 子句進行全文搜尋	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
幾何和地理空間資料類型	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
GRANT	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
GRANT 許可 [,...n] ON SCHEMA	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
GUC babelfish pg_tds.product_version	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否

T-SQL 功能或語法	5.5	5.4	4.4	4.4	4.4	4.4	4.4	4.4	4.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.1.0
開頭為點字元的識別符	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
資料表上的 INSTEAD OF 觸發條件	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
檢視上的 INSTEAD OF 觸發條件	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
KILL	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
$PARTITION.partition_function_name(partition_column_value)	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
PIVOT (在 4.4.0 和 3.8.0 的檢視定義中使用，從 3.4.0 的通用資料表運算式，從 3.4.0 聯結)	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
REVOKE	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
REVOKE 許可 [,...n] ON SCHEMA	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
SELECT... OFFSET...FETCH 子句	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否

T-SQL 功能或語法	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
SELECT FOR JSON AUTO	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
將 BABELFISH_SHOWPLAN_ALL 設為開啟 (和 OFF)	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
將 BABELFISH_STATISTICS PROFILE 設為開啟 (關閉)	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
SET CONTEXT_INFO	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	是	是	是	是	是	是	是	否	否	否	否	否	否
SET LOCK_TIMEOUT	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
SET NO_BROWSETABLE	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
SET 列計數	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	是	是	是	是	是	是	是	否	否	否	否	否	否
SET SHOWPLAN_ALL	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
SET STATISTICS IO	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否

T-SQL 功能或語法	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
SET TRANSACTION ISOLATION LEVEL 語法	是	是	是	是	是	是	是	是	否	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
SSMS：與物件總管連線對話方塊連線	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
SSMS：使用匯入/匯出精靈進行資料遷移	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
SSMS：對物件總管的部分支援	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
STDEV	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	是	是	是	是	是	是	否	否	否	否	否	否	否	否
STDEVP	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	是	是	是	是	是	是	否	否	否	否	否	否	否	否
具有多個 DML 動作的觸發條件能夠參考轉換資料表	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是

T-SQL 功能或語法	5.5.4	4.4.4	4.4.3	3.3.3	3.3.2	2.2.2	2.2.1	2.0.1	1.0
T-SQL 提示 (聯結方法、索引用量、MAXDOP)	是	是	是	是	是	是	是	是	否
使用 LIKE 述詞的 T-SQL 方括號語法	是	是	是	是	是	是	是	是	否
UNPIVOT	是	是	是	是	是	是	是	是	否
預存程序呼叫和預設值中未加引號的字串值	是	是	是	是	是	是	是	是	否
VAR	是	是	是	是	是	是	是	是	否
VARP	是	是	是	是	是	是	是	是	否

T-SQL 內建函數

在下表中，您可以找到不同 Babelfish 版本支援的 T-SQL 內建功能。

T-SQL 內建函數	5.5.4	4.4.4	4.4.3	3.3.3	3.3.2	2.2.2	2.2.1	2.0.1	1.0
APP_NAME	是	是	是	是	是	是	是	是	否
ATN2	是	是	是	是	是	是	是	是	否
CHARINDEX									

T-SQL 內建函數	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
CHOOSE	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
COL_LENGTH	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
COL_NAME	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
COLUMNS_UPDATED	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
COLUMNPROPERTY (CharMaxLen，僅限 AllowsNull)	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
CONCAT_WS	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
CONTEXT_INFO	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
CURSOR_STATUS	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
DATABASE_PRINCIPAL_ID	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否

T-SQL 內建函數	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
DATEADD	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	
DATEDIFF	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	
DATEDIFF_BIG	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	
DATEFROMPARTS	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	
DATENAME	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	
DATEPART	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	
DATETIMEFROMPARTS	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	
DATETIME2FROMPARTS	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	
DATETIMEOFFSETFROMPARTS	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	
DATETRUNC	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	是	是	是	是	否	否	否	否	否	否	否	否	否	

T-SQL 內建函數	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
DATE_BUCKET	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	是	是	是	是	是	是	否	否	否	否	否	否	否	否
EOMONTH	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	是	是	是	是	是	是	否	否	否	否	否	否	否	否
EXECUTE AS CALLER	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否
fn_listextendedproperty	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	是	是	是	是	是	是	否	否	否	否	否	否	否	否
適用於 JSON	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
FULLTEXTSERVICEPROPERTY	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
HAS_DBACCESS	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
HAS_PERMS_BY_NAME	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
HOST_NAME	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
HOST_ID	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否

T-SQL 內建函數	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
IDENTITY	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
IS_MEMBER	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
IS_ROLEMEMBER	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
IS_SRVROLEMEMBER	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
ISJSON	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
JSON_MODIFY	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
JSON_QUERY	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
JSON_VALUE	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
NEXT VALUE FOR	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否
OBJECT_DEFINITION	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否

T-SQL 內建函數	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
OBJECT_SCHEMA_NAME	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否
OPENJSON	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
OPENQUERY	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
ORIGINAL_LOGIN	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
PARSENAME	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	是	是	是	是	是	是	是	否	否	否	否	否	否	否
PATINDEX	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
ROWCOUNT_BIG	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	是	是	是	是	是	是	是	否	否	否	否	否	否	否
SCHEMA_NAME	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
SESSION_CONTEXT	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否
SESSION_USER	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是

T-SQL 內建函數	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
SID_BINARY (一律傳回 NULL)	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
SMALLDATE TIMEFROMPARTS	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
SQUARE	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
STR	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
STRING_AGG	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
STRING_SPLIT	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
SUSER_SID	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
SUSER_SNAME	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
SWITCHOFFSET	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
SYSTEM_USER	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否

T-SQL 內建函數	5.5	5.4	4.4	4.3	4.2	4.1	4.0	3.9	3.8	3.7	3.6	3.5	3.4	3.3	3.2	3.1	3.0	2.9	2.8	2.7	2.6	2.5	2.4	2.3	2.2	2.1	2.0	1.2	1.1	1.0
TIMEFROMPARTS	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否
TODATETIMEOFFSET	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否
TO_CHAR	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
TRIGGER_NESTLEVEL (僅限不使用引數的情況)	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
TRY_CONVERT	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
TYPE_ID	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否
TYPE_NAME	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否
UPDATE	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是

T-SQL INFORMATION_SCHEMA 目錄

在下表中，您可以找到不同 Babelfish 版本支援的 T-SQL INFORMATION_SCHEMA 目錄。

| | | |

T-SQL 系統定義的 @@ 變數

在下表中，您可以找到不同 Babelfish 版本支援的 T-SQL 系統定義 @@ 變數。

T-SQL 系統定義的 @@ 變數	5.5.4	4.4.4	4.4.3	3.3.3	3.3.2	2.2.2	2.2.1.0
@@CURSOR_ROWS	是	是	是	是	是	是	是
@@DATEFIRST	是	是	是	是	是	是	是
@@DBTS	是	是	是	是	是	是	是
@@ERROR	是	是	是	是	是	是	是
@@ERROR=213	是	是	是	是	是	是	是
@@FETCH_STATUS	是	是	是	是	是	是	是
@@IDENTITY	是	是	是	是	是	是	是
@@LANGUAGE	是	是	是	是	是	是	是
@@LOCK_TIMEOUT	是	是	是	是	是	是	是

T-SQL 系統定義的 @@ 變數	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
@@MAX_CONNECTIONS	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
@@MAX_PRECISION	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
@@MICROSOFTVERSION	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
@@NESTLEVEL	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
@@PROCID	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
@@ROWCOUNT	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
@@SERVERNAME	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
@@SERVICENAME	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
@@SPID	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
@@TRANCOUNT	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是

| T-SQL 系統定義的 @@ 變數 | 5.5.4 | 4.4.4 | 4.4.4 | 4.4.4 | 4.4.3 | 3.3.3 | 3.3.3 | 3.3.3 | 3.3.2 | 2.2.2 | 2.2.2 | 2.2

| T-SQL 系統預存程序 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2

T-SQL 系統預存程序	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_cursorprepare	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_cursorprepexec	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_cursorunprepare	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_databases	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_datatype_info	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_datatype_info_100	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_describe_cursor	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_describe_first_result_set	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_describe_undeclared_parameters	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_dropextendedproperty	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	是	是	是	是	是	是	否	否	否	否	否	否	否	否

T-SQL 系統預存程序	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_droplinkedsrvlogin	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	
sp_droprole	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	
sp_droprolemember	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	
sp_dropserver	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	
sp_enum_oledb_providers	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	是	是	是	是	是	是	是	否	否	否	否	否	否	
sp_execute	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	
sp_execute_postgresql(CREATE、ALTER、DROP)	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	是	是	是	是	是	是	否	否	否	否	否	否	否	
sp_executesql	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	
sp_fkeys	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	
sp_getapplock	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	

| T-SQL 系統預存程序 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2

T-SQL 系統預存程序	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_prefix	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
sp_prepare	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_procedure_params_100_managed	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
sp_releaseapplock	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_rename	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否
sp_renamedb	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
sp_rename (適用於 object_type 'DATABASE')	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
sp_reset_connection	是	是	是	是	是	否	否	否	否	是	是	否	否	否	否	否	否	否	否	是	是	否	否	否	否	否	否	否	否	否	否	否	否
sp_serveroption(connect_timeout 選項)	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	是	是	是	是	是	是	否	否	否	否	否	否
sp_set_session_context	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否

T-SQL 系統預存程序	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_special_columns	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_sproc_columns	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_sproc_columns_100	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_statistics	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_statistics_100	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_stored_procedures	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_table_privileges	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_tablecollations_100	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_tables	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_testlinkedserver	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否

| T-SQL 系統預存程序 | 5.5 | 5.4 | 4.4 | 4.4 | 4.4 | 4.3 | 3.3 | 3.3 | 3.3 | 3.3 | 3.3 | 3.3 | 3.3 | 3.3 | 3.3 | 2.2 | 2.2 | 2.2

CONNECTIONPROPERTY 系統函數支援的 T-SQL 屬性	5.5.44.44.44.44.44.44.43.33.33.33.33.33.33.33.33.32.22.22.22.22.22.22.22.22.22.22.22.22.2	1.0
	是	
net_transport	是	
protocol_type	是	
physical_net_transport	是	

OBJECTPROPERTY 系統函數支援的 T-SQL 屬性

在下表中，您可找到不同 Babelfish 版本支援的 OBJECTPROPERTY 系統函數上支援的 T-SQL 屬性。

OBJECTPROPERTY 系統函數支援的 T-SQL 屬性	5.5.44.44.44.44.44.44.43.33.33.33.33.33.33.33.33.32.22.22.22.22.22.22.22.22.22.22.22.22.2	1.0
IsInlineFunction	是	
IsScalarFunction	是	
IsTableFunction	是	

SERVERPROPERTY 函數支援的 T-SQL 屬性

在下表中，您可找到不同 Babelfish 版本支援的 SERVERPROPERTY 函數上支援的 T-SQL 屬性。

SERVERPROPERTY 函數支援的 T-SQL 屬性	5.5.0	5.4.0	4.4.0	4.3.3	4.3.2	4.2.2	4.2.1	4.1.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.0.0
Babelfish	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
定序	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
CollationID	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
版本	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
EditionID	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
EngineEdition	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
InstanceName	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
IsAdvancedAnalyticsInstalled	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
IsBigDataCluster																			

SERVERPROPERTY 函數支援的 T-SQL 屬性	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
IsFullTextInstalled	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
IsIntegratedSecurityOnly	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
IsLocalDB	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
IsPolyBaseInstalled	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
IsSingleUser	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
IsXTPSupported	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
Japanese_CI_AI	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
Japanese_CI_AS	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
Japanese_CS_AS	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是

SERVERPROPERTY 函數支援的 T-SQL 屬性	5.5	5.4	4.4	4.3	3.3	3.2	2.2	2.1.0
LicenseType	是	是	是	是	是	是	是	是
MachineName	否	否	否	否	否	否	否	否
ProductLevel	否	否	否	否	否	否	否	否
ProductMajorVersion	是	是	是	是	是	是	是	是
ProductMinorVersion	是	是	是	是	是	是	是	是
ProductUpdateLevel	否	否	否	否	否	否	否	否
ProductVersion	是	是	是	是	是	是	是	是
ServerName	是	是	是	是	是	是	是	是

Babelfish 支援的 SQL Server 檢視

在下表中，您可以找到不同 Babelfish 版本支援的 SQL Server 檢視。

Babelfish 支援的 SQL Server 檢視	5.5.4.4.4.4.4.4.4.3.3.3.3.3.3.3.3.3.2.2.2.2.2.2.2.2.2.2.2.2.2.2.1.0
information_schema.key_column_usage	否
information_schema.routines	否
information_schema.schemata	否
information_schema.sequences	否
sys.all_columns	是
sys.all_objects	是
sys.all_parameters	否
sys.all_sql_modules	是
sys.all_views	是
sys.columns	是

Babelfish 支援的 SQL Server 檢視	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sys.configurations	是
sys.data_spaces	是
sys.database_files	是
sys.database_mirroring	是
sys.database_principals	是
sys.database_role_members	是
sys.databases	是
sys.destination_data_spaces	否
sys.dm_exec_connections	是
sys.dm_exec_sessions	是

Babelfish 支援的 SQL Server 檢視	5.5.4.4.4.4.4.4.4.3.3.3.3.3.3.3.3.3.2.2.2.2.2.2.2.2.2.2.2.2.2.2	1.0
sys.dm_hadr_database_replica_states	是	
sys.dm_os_host_info	是	
sys.dm_os_sys_info	否	
sys.endpoints	是	
sys.extended_properties	否	
sys.indexes	是	
sys.partitions	否	
sys.partition_functions	否	
sys.partition_parameters	否	
sys.partition_range_values	否	

Babelfish 支援的 SQL Server 檢視	5.1.0
sys.partition_schemes	否
sys.schemas	是
sys.server_principals	是
sys.server_role_members	否
sys.sql_modules	是
sys.sysconfigures	是
sys.syscurconfigs	是
sys.syslogins	否
sys.sysprocesses	是
sys.sysusers	否

Babelfish 支援的 SQL Server 檢視	5.5	5.4	4.4	4.3	4.2	4.1	4.0	3.4	3.3	3.2	3.1	3.0	2.11	2.10	2.9	2.8	2.7	2.6	2.5	2.4	2.3	2.2	2.1	2.0	1.0
sys.table_types	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sys.tables	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sys.types	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
sys.xml_schema_collections	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
syslanguages	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sysobjects.crdate	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否

Babelfish 中不支援的功能

下列表格和清單中列出 Babelfish 目前不支援的功能。Babelfish 的更新包含於 Aurora PostgreSQL 版本中。如需詳細資訊，請參閱 Aurora PostgreSQL 版本備註。

主題

- 目前不支援的功能
- 不支援的設定
- 不支援的命令
- 不支援的資料欄名稱或屬性
- 不支援的資料類型
- 不支援的物件類型

- 不支援的函數
- 不支援的語法

目前不支援的功能

在此表格中，您可以找到目前不支援的某些功能的相關資訊。

功能或語法	Description
組件模組和 SQL Common Language Runtime (CLR) 常式	不支援組件模組和 CLR 常式相關的功能。
資料欄屬性	不支援 ROWGUIDCOL、SPARSE、FILESTREAM 和 MASKED。
自主資料庫	不支援在資料庫層級而非伺服器層級驗證登入的自主資料庫。
跨資料庫 DDL	尚不支援在多個資料庫的物件上執行會參考或操作的 DDL 陳述式。
游標 (可更新)	不支援可更新的游標。
游標 (全域)	不支援 GLOBAL 游標。
游標 (擷取行為)	不支援下列游標擷取行為：FETCH PRIOR、FIRST、LAST、ABSOLUTE 和 RELATIVE
游標類型的輸出參數	不支援游標類型的變數和參數的輸出參數 (會引發錯誤)。
游標選項	SCROLL、KEYSET、DYNAMIC、FAST_FORWARD、SCROLL_LOCKS、OPTIMISTIC、TYPE_WARNING 和 FOR UPDATE
資料加密	不支援資料加密。
資料層應用程式 (DAC)	不支援使用 DAC 套件 (.dacpac) 或 DAC 備份 (.bacpac) 檔案進行資料層應用程式 (DAC) 匯入或匯出操作。
DBCC 命令	不支援 Microsoft SQL Server 資料庫主控台命令 (DBCC)。在 Babelfish 3.4.0 及更新版本中支援 DBCC CHECKIDENT。

功能或語法	Description
DROP IF EXISTS	此語法不支援 USER 和 SCHEMA 物件。支援物件 TABLE、VIEW、PROCEDURE、FUNCTION 及 DATABASE。
加密	內建函數和陳述式不支援加密。
ENCRYPT_CLIENT_CERT 連線	不支援用戶端憑證連線。
EXECUTE AS 陳述式	不支援此陳述式。
EXECUTE AS SELF 子句	函數、程序或觸發程序中不支援此子句。
EXECUTE AS USER 子句	函數、程序或觸發程序中不支援此子句。
參考資料庫名稱的外部索引鍵條件限制	不支援參考資料庫名稱的外部索引鍵條件限制。
FORMAT	不支援使用者定義的類型。
超過 100 個參數的函數宣告	不支援包含超過 100 個參數的函數宣告。
以 DEFAULT 做為參數值的函數呼叫	DEFAULT 不是函數呼叫支援的參數值。Babelfish 3.4.0 及更新版本支援 DEFAULT 作為函數呼叫的參數值。
外部定義的函數	不支援外部函數，包括 SQL CLR 函數。
全域暫存資料表 (名稱以 ## 開頭的資料表)	不支援全域暫存資料表。
圖表功能	不支援所有 SQL 圖表功能。
一般擴充預存程序	不支援系統預存程序，這類程序會為各種維護活動提供從 SQL Server 執行個體到外部程式的界面。這包括 `xp_cmdshell` 和其他系統儲存的程序。如需詳細資訊，請參閱一般擴充預存程序。
開頭有多個 @ 字元的標識符 (變數或參數)	不支援以多個 @ 開頭的識別符。

功能或語法	Description
包含 @ 或]] 字元的識別符、資料表或列名	不支援包含 @ 符號或方括號的資料表或資料欄名稱。
內嵌索引	不支援內嵌索引。
叫用以變數代表名稱的程序	不支援使用變數作為程序名稱。
具體化檢視表	不支援具體化檢視表。
NOT FOR REPLICATION 子句	接受並忽略此語法。
ODBC 逸出函數	不支援 ODBC 逸出函數。
以 DEFAULT 為參數值的程序呼叫	DEFAULT 不是支援的參數值。Babelfish 3.4.0 及更新版本支援 DEFAULT 作為函數呼叫的參數值。
超過 100 個參數的程序宣告	不支援包含超過 100 個參數的程序宣告。
外部定義的程序	不支援外部定義的程序，包括 SQL CLR 程序。
程序版本控制	不支援程序版本控制。
程序 WITH RECOMPILE	不支援 WITH RECOMPILE (與 DECLARE 和 EXECUTE 陳述式一起使用時)。
遠端物件參考	不支援針對 Babelfish 連結伺服器執行預存程序。由四個部分組成的物件名稱僅適用於讀取，不適用於修改遠端資料表。UPDATE 可以參考 FROM 子句中的遠端資料表，而不對其進行修改。如需詳細資訊，請參閱Babelfish 支援連結的伺服器。
資料列層級安全性	不支援資料列層級安全性使用 CREATE SECURITY POLICY 和內嵌資料表值函數。
服務代理程式功能	不支援服務代理程式功能。

功能或語法	Description
SESSIONPROPERTY	不支援的內容：ANSI_NULLS、ANSI_PADDING、ANSI_WARNINGS、ARITHABORT、CONCAT_NULL_YIELDS_NULL 和 NUMERIC_ROUNDABORT
SET LANGUAGE	此語法不支援 `english` 或 `us_english` 除外的任何值。
SP_CONFIGURE	不支援此系統預存程序。
SQL 關鍵字 `SPARSE`	接受和忽略關鍵字 SPARSE。
資料表值建構函式語法 (FROM 子句)	不支援的語法適用於使用 FROM 子句建構的衍生資料表。
時態資料表	不支援時態資料表。
不自動捨棄暫存程序	不支援此功能。
外部定義的觸發程序	不支援這些觸發程序，包括 SQL Common Language Runtime (CLR)。

不支援的設定

不支援以下設定：

- SET ANSI_NULL_DFLT_OFF ON
- SET ANSI_NULL_DFLT_ON OFF
- SET ANSI_PADDING OFF
- SET ANSI_WARNINGS OFF
- SET ARITHABORT OFF
- SET ARITHIGNORE ON
- SET CURSOR_CLOSE_ON_COMMIT ON
- SET NUMERIC_ROUNDABORT ON
- SET PARSEONLY ON (命令未如預期運作)
- SET FMTLY ON (命令未如預期運作。它只會抑制 SELECT 陳述式的執行，而不會抑制其他陳述式的執行。)

不支援的命令

某些功能不支援以下命令：

- ADD SIGNATURE
- ALTER DATABASE、ALTER DATABASE SET
- BACKUP/RESTORE DATABASE/LOG
- BACPAC 和 DACPAC FILES RESTORE
- CREATE、ALTER、DROP AUTHORIZATION。針對資料庫物件支援 ALTER AUTHORIZATION。
- CREATE、ALTER、DROP AVAILABILITY GROUP
- CREATE、ALTER、DROP BROKER PRIORITY
- CREATE、ALTER、DROP COLUMN ENCRYPTION KEY
- CREATE、ALTER、DROP DATABASE ENCRYPTION KEY
- CREATE、ALTER、DROP、BACKUP CERTIFICATE
- CREATE AGGREGATE
- CREATE CONTRACT
- CHECKPOINT

不支援的資料欄名稱或屬性

不支援以下資料欄名稱：

- $IDENTITY
- $ROWGUID
- IDENTITYCOL

不支援的資料類型

不支援以下資料類型：

- HIERARCHYID

不支援的物件類型

不支援以下物件類型：

- COLUMN MASTER KEY
- CREATE、ALTER EXTERNAL DATA SOURCE
- CREATE、ALTER、DROP DATABASE AUDIT SPECIFICATION
- CREATE、ALTER、DROP EXTERNAL LIBRARY
- CREATE、ALTER、DROP SERVER AUDIT
- CREATE、ALTER、DROP SERVER AUDIT SPECIFICATION
- CREATE、ALTER、DROP、OPEN/CLOSE SYMMETRIC KEY
- CREATE、DROP DEFAULT
- CREDENTIAL
- CRYPTOGRAPHIC PROVIDER
- DIAGNOSTIC SESSION
- 已編製索引的檢視表
- SERVICE MASTER KEY
- SYNONYM

不支援的函數

不支援以下內建函數：

彙總函數

- APPROX_COUNT_DISTINCT
- CHECKSUM_AGG
- GROUPING_ID
- 使用 WITHIN GROUP 子句的 STRING_AGG

密碼編譯函數

- CERTENCODED 函數
- CERTID 函數
- CERTPROPERTY 函數

中繼資料函數

- COLUMNPROPERTY
- TYPEPROPERTY
- SERVERPROPERTY 函數 — 不支援以下屬性：
  - BuildClrVersion
  - ComparisonStyle
  - ComputerNamePhysicalNetBIOS
  - HadrManagerStatus
  - InstanceDefaultDataPath
  - InstanceDefaultLogPath
  - IsClustered
  - IsHadrEnabled
  - LCID
  - NumLicenses
  - ProcessID
  - ProductBuild
  - ProductBuildType
  - ProductUpdateReference
  - ResourceLastUpdateDateTime
  - ResourceVersion
  - ServerName
  - SqlCharSet
  - SqlCharSetName
  - SqlSortOrder
  - SqlSortOrderName
  - FilestreamShareName
  - FilestreamConfiguredLevel
  - FilestreamEffectiveLevel

安全性函數

- CERTPRIVATEKEY
- LOGINPROPERTY

陳述式、運算子、其他函數

- EVENTDATA 函數
- GET_TRANSMISSION_STATUS
- OPENXML

不支援的語法

不支援以下語法：

- ALTER DATABASE
- ALTER DATABASE SCOPED CONFIGURATION
- ALTER DATABASE SCOPED CREDENTIAL
- ALTER DATABASE SET HADR
- ALTER INDEX
- ALTER PARTITION FUNCTION
- ALTER PARTITION SCHEME
- ALTER SCHEMA
- ALTER SERVER CONFIGURATION
- ALTER SERVICE、BACKUP/RESTORE SERVICE MASTER KEY 子句
- BEGIN CONVERSATION TIMER
- BEGIN DISTRIBUTED TRANSACTION
- BEGIN DIALOG CONVERSATION
- BULK INSERT
- CREATE COLUMNSTORE INDEX
- CREATE EXTERNAL FILE FORMAT
- CREATE EXTERNAL TABLE
- CREATE、ALTER、DROP APPLICATION ROLE

- CREATE、ALTER、DROP ASSEMBLY
- CREATE、ALTER、DROP ASYMMETRIC KEY
- CREATE, ALTER, DROP CREDENTIAL
- CREATE, ALTER, DROP CRYPTOGRAPHIC PROVIDER
- CREATE, ALTER, DROP ENDPOINT
- CREATE、ALTER、DROP EVENT SESSION
- CREATE, ALTER, DROP EXTERNAL LANGUAGE
- CREATE、ALTER、DROP EXTERNAL RESOURCE POOL
- CREATE、ALTER、DROP FULLTEXT CATALOG
- CREATE、ALTER、DROP FULLTEXT INDEX
- CREATE、ALTER、DROP FULLTEXT STOPLIST
- CREATE, ALTER, DROP MESSAGE TYPE
- CREATE、ALTER、DROP、OPEN/CLOSE、BACKUP/RESTORE MASTER KEY
- CREATE、ALTER、DROP QUEUE
- CREATE、ALTER、DROP RESOURCE GOVERNOR
- CREATE, ALTER, DROP RESOURCE POOL
- CREATE、ALTER、DROP ROUTE
- CREATE, ALTER, DROP SEARCH PROPERTY LIST
- CREATE, ALTER, DROP SECURITY POLICY
- CREATE, ALTER, DROP SELECTIVE XML INDEX clause
- CREATE、ALTER、DROP SERVICE
- CREATE, ALTER, DROP SPATIAL INDEX
- CREATE, ALTER, DROP TYPE
- CREATE, ALTER, DROP XML INDEX
- CREATE, ALTER, DROP XML SCHEMA COLLECTION
- CREATE/DROP RULE
- CREATE, DROP WORKLOAD CLASSIFIER
- CREATE、ALTER、DROP WORKLOAD GROUP
- ALTER TRIGGER
- CREATE TABLE... GRANT 子句

- CREATE TABLE... IDENTITY 子句
- CREATE USER – 不支援此語法。PostgreSQL 陳述式 CREATE USER 不建立相當於 SQL Server CREATE USER 語法的使用者。如需詳細資訊，請參閱Babelfish 中的 T-SQL 差異。
- 拒絕
- END, MOVE CONVERSATION
- EXECUTE with AS LOGIN or AT option
- GET CONVERSATION GROUP
- GROUP BY ALL clause
- GROUP BY CUBE clause
- GROUP BY ROLLUP clause
- INSERT... DEFAULT VALUES
- MERGE
- READTEXT
- REVERT
- SELECT TOP... WITH TIES
- SELECT... FOR BROWSE
- SELECT... FOR XML AUTO
- SELECT... FOR XML EXPLICIT
- SELECT... FOR XML PATH
- SEND
- SET DATEFORMAT
- SET DEADLOCK_PRIORITY
- SET FMTONLY
- SET FORCEPLAN
- SET NUMERIC_ROUNDABORT ON
- SET OFFSETS
- SET REMOTE_PROC_TRANSACTIONS
- SET SHOWPLAN_TEXT
- SET SHOWPLAN_XML
- SET STATISTICS
- SET STATISTICS PROFILE

- SET STATISTICS TIME
- SET STATISTICS XML
- SHUTDOWN statement
- UPDATE STATISTICS
- UPDATETEXT
- Using EXECUTE to call a SQL function
- VIEW... CHECK OPTION clause
- VIEW... VIEW_METADATA clause
- WAITFOR DELAY
- WAITFOR TIME
- WAITFOR, RECEIVE
- WITH XMLNAMESPACES construct
- WRITETEXT
- XPATH expressions

使用 Babelfish for Aurora PostgreSQL

概觀

您可以將下列程序用於執行 Babelfish for Aurora PostgreSQL 的 Amazon RDS 資料庫執行個體，以取得更好的查詢效能：

- sp_babelfish_volatility
- sp_execute_postgresql

sp_babelfish_volatility

PostgreSQL 函數波動有助於最佳化工具取得更好的查詢執行，當用於特定子句的某些部分時，這會對查詢效能產生顯著影響。

語法

```
sp_babelfish_volatility 'function_name', 'volatility'
```

引數

function_name (選用)

您可以使用兩段式名稱 (如 `schema_name.function_name`) 或僅使用 `function_name`，來指定此引數的值。如果您僅指定 `function_name`，結構描述名稱就是目前使用者的預設結構描述。

波動 (選用)

波動的有效 PostgreSQL 值為 `stable`、`volatile` 或 `immutable`。如需詳細資訊，請參閱https://www.postgresql.org/docs/current/xfunc-volatility.html

> **Note**
>
> 當使用具有多個定義的 `function_name` 呼叫 `sp_babelfish_volatility` 時，其會擲出一個錯誤。

結果集

如果未提及參數，則結果集會顯示在下列欄位下方：schemaname、functionname、volatility。

使用須知

PostgreSQL 函數波動有助於最佳化工具取得更好的查詢執行，當用於特定子句的某些部分時，這會對查詢效能產生顯著影響。

範例

下列範例展示如何建立簡單函數，稍後說明如何使用不同的方法，在這些函數上使用 sp_babelfish_volatility。

```
1> create function f1() returns int as begin return 0 end
2> go
```

```
1> create schema test_schema
2> go
```

```
1> create function test_schema.f1() returns int as begin return 0 end
2> go
```

下列範例顯示函數的波動：

```
1> exec sp_babelfish_volatility
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           volatile
test_schema f1           volatile
```

下列範例展示如何變更函數的波動：

```
1> exec sp_babelfish_volatility 'f1','stable'
2> go
1> exec sp_babelfish_volatility 'test_schema.f1','immutable'
```

```
2> go
```

當您僅指定 function_name 時，其會顯示結構描述名稱，函數名稱，以及該函數的波動。下列範例顯示函數在變更值之後的波動：

```
1> exec sp_babelfish_volatility 'test_schema.f1'
2> go

schemaname  functionname volatility
----------- ------------ ----------
test_schema f1           immutable
```

```
1> exec sp_babelfish_volatility 'f1'
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           stable
```

當您未指定任何引數時，其會顯示目前資料庫中存在的函數清單 (結構描述名稱、函數名稱、函數的波動)：

```
1> exec sp_babelfish_volatility
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           stable
test_schema f1           immutable
```

sp_execute_postgresql

您可以從 T-SQL 端點執行 PostgreSQL 陳述式。這樣可簡化您的應用程式，因為您不需要結束 T-SQL 連接埠，即可執行這些陳述式。

語法

```
sp_execute_postgresql [ @stmt = ] statement
```

引數

[@stmt] 陳述式

引數屬於資料類型 varchar。此引數接受 PG 方言陳述式。

> **Note**
>
> 您只能傳遞一個 PG 方言陳述式作為引數，否則其會引發下列錯誤。

```
1>exec sp_execute_postgresql 'create extension pg_stat_statements; drop extension
 pg_stat_statements'
2>go
```

```
Msg 33557097, Level 16, State 1, Server BABELFISH, Line 1
expected 1 statement but got 2 statements after parsing
```

使用須知

CREATE EXTENSION

建立新的擴充功能並將其載入至目前資料庫。

```
1>EXEC sp_execute_postgresql 'create extension [ IF NOT EXISTS ] <extension name>
 [ WITH ] [SCHEMA schema_name] [VERSION version]';
2>go
```

下列範例示範如何建立擴充功能：

```
1>EXEC sp_execute_postgresql 'create extension pg_stat_statements with schema sys
 version "1.10"';
2>go
```

請使用下列命令存取擴充功能物件：

```
1>select * from pg_stat_statements;
2>go
```

> **Note**
>
> 如果在擴充功能建立期間未明確提供結構描述名稱，則依預設，擴充功能會安裝在公開結構描述中。您必須提供結構描述限定詞來存取擴充功能物件，如下所述：

```
1>select * from [public].pg_stat_statements;
2>go
```

支援的擴充功能

Aurora PostgreSQL 提供的以下擴充功能可與 Babelfish 搭配使用。

- `pg_stat_statements`
- `tds_fdw`
- `fuzzystrmatch`

限制

- 使用者必須在 T-SQL 上具有 sysadmin 角色，並在 postgres 上具有 rds_superuser，才能安裝擴充功能。
- 擴充功能無法安裝在使用者建立的結構描述中，也不能安裝在 master、tempdb 和 msdb 資料庫的 dbo 和客體結構描述中。
- 不支援 CASCADE 選項。

ALTER EXTENSION

您可以使用 ALTER Extension 升級到新的擴充功能版本。

```
1>EXEC sp_execute_postgresql 'alter extension <extension name> UPDATE TO
 <new_version>';
2>go
```

限制

- 您只能使用 ALTER Extension 陳述式來升級擴充功能的版本。不支援其他操作。

DROP EXTENSION

刪除指定的擴充功能。您也可以使用 `if exists` 或 `restrict` 選項刪除擴充功能。

```
1>EXEC sp_execute_postgresql 'drop extension <extension name>';
2>go
```

限制

- 不支援 CASCADE 選項。

Babelfish 支援 XML 資料類型方法

從 5.4.0 版開始，Babelfish 現在支援預存程序 sp_xml_preparedocument 和 sp_xml_removedocument、列集函數 OPENXML() 和 xml datatype 方法 .VALUE()。透過這些函數和程序，查詢 XML 資料變得更加容易。

了解 XML 程序與方法

- sp_xml_preparedocument – 程序 sp_xml_preparedocument 剖析指定為輸入的 XML 文字，並將控點傳回至本文件。此控制代碼在工作階段期間或直到 sp_xml_removedocument 移除為止有效。
- sp_xml_removedocument – 程序 sp_xml_removedocument 會使程序 sp_xml_preparedocument 建立的控制代碼失效。
- OPENXML() – OPENXML 透過 XML 文件提供資料列集檢視。由於 OPENXML 是資料列集提供者，而且會傳回一組資料列，因此我們可以在 FROM 子句中使用 OPENXML，就像使用任何其他資料表、檢視或資料表值函數一樣。
- VALUE() – XML 資料類型方法 VALUE() 用於從存放在 xml 類型資料欄、參數或變數中的 XML 執行個體擷取值。

Babelfish XML 程序與方法的限制

- Babelfish 僅支援 OPENXML() 的第二個引數 （即 ROWPATTERN) 的 XPATH 1.0 語法。
- OPENXML() 目前不支援中繼屬性和旗標 8。
- Babelfish 僅支援 VALUE() 資料類型方法的第一個引數 （即 XQuery) 的 XPATH 1.0 語法。

Amazon Aurora PostgreSQL 的效能和擴展

下列幾節說明如何管理 Amazon Aurora PostgreSQL 資料庫叢集的效能和擴展功能。也包括其他維護任務的相關資訊。

主題

- 擴展 Aurora PostgreSQL 資料庫執行個體
- 對 Aurora PostgreSQL 資料庫執行個體的連線數上限
- Aurora PostgreSQL 的暫存空間限制
- Aurora PostgreSQL 的巨型分頁
- 使用錯誤注入查詢測試 Amazon Aurora PostgreSQL

- 顯示 Aurora PostgreSQL 資料庫叢集的磁碟區狀態
- 指定 stats_temp_directory 的 RAM 磁碟
- 使用 PostgreSQL 管理暫存檔案

擴展 Aurora PostgreSQL 資料庫執行個體

有兩種方式可以擴展 Aurora PostgreSQL 資料庫執行個體：執行個體擴展和讀取擴展。如需讀取擴展的詳細資訊，請參閱讀取擴展。

您可以修改資料庫叢集中每個資料庫執行個體的資料庫執行個體類別，來擴展 Aurora PostgreSQL 資料庫叢集。Aurora PostgreSQL 支援數個針對 Aurora 最佳化的資料庫執行個體類別。針對大小超過 40 TB 的較大 Aurora 叢集，請勿使用 db.t2 或 db.t3 執行個體類別。

> **Note**
>
> 建議您在開發、測試伺服器或其他非生產伺服器時，僅使用 T 資料庫執行個體類別。如需詳細了解 T 執行個體類別，請參閱 資料庫執行個體類別的類型。

擴展並非即時進行。可能需要 15 分鐘或更長時間，才能完成對其他資料庫執行個體類別做的變更。若您使用此方法修改資料庫執行個體類別，請在下一個排程維護時段 (而非立即) 套用變更，以避免影響使用者。

作為修改資料庫執行個體類別的直接替代方法，您可以使用 Amazon Aurora 的高可用性功能來盡力減少停機時間。首先，請將 Aurora 複本新增至您的叢集。在建立複本時，請選擇要用於叢集的資料庫執行個體類別規模。在 Aurora 複本與叢集同步時，您便可以容錯移轉為新增的複本。如需了解詳細資訊，請參閱 Aurora 複本 和 Amazon Aurora PostgreSQL 的快速容錯移轉。

如需 Aurora PostgreSQL 支援資料庫執行個體類別的詳細規格，請參閱資料庫執行個體類別的支援資料庫引擎。

對 Aurora PostgreSQL 資料庫執行個體的連線數上限

Aurora PostgreSQL 資料庫叢集會根據資料庫執行個體類別及其可用記憶體分配資來源。與資料庫叢集的每個連線都會遞增地佔用這些資源，例如記憶體和 CPU 資源。每個連線佔用的記憶體會因查詢類型、計數及是否使用臨時資料表而有所不同。即使是閒置連線也會佔用記憶體和 CPU。這是因為當查詢在連線上執行時，會為每個查詢分配更多的記憶體，且即使停止處理，也不會將其完全釋放。因此，我們建議您確保您的應用程式不會在閒置連線上等待：每一個閒置連線都會浪費資源並對效能

產生負面影響。如需詳細資訊，請參閱 Resources consumed by idle PostgreSQL connections (閒置 PostgreSQL 連線所佔用的資源)。

允許對 Aurora PostgreSQL 資料庫執行個體建立的連線數上限，由資料庫執行個體的執行個體參數群組中指定的 max_connections 參數決定。max_connections 參數的理想設定是支援應用程式所需的所有用戶端連線，而不會超過未使用的連線，以及至少 3 個支援 AWS 自動化的連線。在修改 max_connections 參數設定之前，建議您考慮下列內容：

- 如果 max_connections 值太低，用戶端嘗試連線時 Aurora PostgreSQL 資料庫執行個體可能會沒有足夠的可用連線。若發生這種情況，則嘗試使用 psql 連線將引發下列錯誤訊息：

```
psql: FATAL: remaining connection slots are reserved for non-replication superuser
 connections
```

- 如果 max_connections 值超過實際需要的連線數，未使用的連線可能會導致效能降低。

max_connections 的預設值是從下列 Aurora PostgreSQL LEAST 函數所衍生的：

```
LEAST({DBInstanceClassMemory/9531392},5000).
```

若您想變更 max_connections 值，您需要建立自訂資料庫叢集參數群組並在其中變更其值。將自訂資料庫參數群組套用至叢集後，請務必重新啟動主執行個體，新的值才會生效。如需詳細資訊，請參閱Amazon Aurora PostgreSQL 參數及在 Amazon Aurora 中建立資料庫叢集參數群組。

> **Tip**
>
> 如果您的應用程式經常開啟和關閉連線，或者保持大量長期連線開啟，我們建議您使用 Amazon RDS Proxy。RDS 代理是個完全受管、高可用性的資料庫代理，其使用連線集區，安全且高效地共用資料庫連線。如要進一步了解 RDS 代理，請參閱 Amazon RDS Proxy for Aurora。

如需 Aurora Serverless v2 執行個體處理此參數的詳細資訊，請參閱 Aurora Serverless v2 的連線數上限。

Aurora PostgreSQL 的暫存空間限制

Aurora PostgreSQL 會將資料表和索引儲存在 Aurora 儲存子系統。Aurora PostgreSQL 對非持久性暫存檔案會使用個別的暫存空間。這包括用於查詢處理期間排序大型資料集或用於索引建置作業等目的的

檔案。如需詳細資訊，請參閱如何在 Aurora PostgreSQL 相容的執行個體中針對本機儲存體問題進行疑難排解？一文。

這些本機儲存磁碟區由 Amazon Elastic Block Store 支援，且可透過使用更大的資料庫執行個體類別來擴充。如需儲存體的詳細資訊，請參閱Amazon Aurora 儲存體。您也可以使用啟用 NVMe 的執行個體類型和啟用 Aurora 最佳化讀取的暫存物件，來增加暫存物件的本機儲存體。如需詳細資訊，請參閱使用 Aurora Optimized Reads 改善 Aurora PostgreSQL 的查詢效能。

> **Note**
>
> 當擴充資料庫執行個體時，例如從 db.r5.2xlarge 擴充到 db.r5.4xlarge，您可能會看見 `storage-optimization` 事件。

下表顯示每個 Aurora PostgreSQL 資料庫執行個體類別可用的暫存空間容量上限。如需 Aurora 之資料庫執行個體類別支援的詳細資訊，請參閱 Amazon Aurora 資料庫執行個體類別。

DB instance class (資料庫執行個體類別)	可用的暫存空間上限 (GiB)
db.x2g.16xlarge	1829
db.x2g.12xlarge	1606
db.x2g.8xlarge	1071
db.x2g.4xlarge	535
db.x2g.2xlarge	268
db.x2g.xlarge	134
db.x2g.large	67
db.r8g.48xlarge	3072
db.r8g.24xlarge	1536
db.r8g.16xlarge	998
db.r8g.12xlarge	749

DB instance class (資料庫執行個體類別)	可用的暫存空間上限 (GiB)
db.r8g.8xlarge	499
db.r8g.4xlarge	250
db.r8g.2xlarge	125
db.r8g.xlarge	63
db.r8g.large	31
db.r7g.16xlarge	1008
db.r7g.12xlarge	756
db.r7g.8xlarge	504
db.r7g.4xlarge	252
db.r7g.2xlarge	126
db.r7g.xlarge	63
db.r7g.large	32
db.r7i.48xlarge	3072
db.r7i.24xlarge	1500
db.r7i.16xlarge	1008
db.r7i.12xlarge	748
db.r7i.8xlarge	504
db.r7i.4xlarge	249
db.r7i.2xlarge	124
db.r7i.xlarge	62

DB instance class (資料庫執行個體類別)	可用的暫存空間上限 (GiB)
db.r7i.large	31
db.r6g.16xlarge	1008
db.r6g.12xlarge	756
db.r6g.8xlarge	504
db.r6g.4xlarge	252
db.r6g.2xlarge	126
db.r6g.xlarge	63
db.r6g.large	32
db.r6i.32xlarge	1829
db.r6i.24xlarge	1500
db.r6i.16xlarge	1008
db.r6i.12xlarge	748
db.r6i.8xlarge	504
db.r6i.4xlarge	249
db.r6i.2xlarge	124
db.r6i.xlarge	62
db.r6i.large	31
db.r5.24xlarge	1500
db.r5.16xlarge	1008
db.r5.12xlarge	748

DB instance class (資料庫執行個體類別)	可用的暫存空間上限 (GiB)
db.r5.8xlarge	504
db.r5.4xlarge	249
db.r5.2xlarge	124
db.r5.xlarge	62
db.r5.large	31
db.r4.16xlarge	960
db.r4.8xlarge	480
db.r4.4xlarge	240
db.r4.2xlarge	120
db.r4.xlarge	60
db.r4.large	30
db.t4g.large	16.5
db.t4g.medium	8.13
db.t3.large	16
db.t3.medium	7.5

> **Note**
>
> 啟用 NVMe 的執行個體類型可將可用暫存空間增加至最高可達 NVMe 的大小總計。如需詳細資訊，請參閱使用 Aurora Optimized Reads 改善 Aurora PostgreSQL 的查詢效能。

您可以使用 `FreeLocalStorage` CloudWatch 指標來監控資料庫執行個體可用的暫存空間，如 Amazon Aurora 的 Amazon CloudWatch 指標 中所述。(這不適用於 Aurora Serverless v2。)

對於某些工作負載，您可以針對執行該操作的處理程序配置更多記憶體，以減少暫存空間容量。若要增加可用於操作的記憶體，請增加 work_mem 或 maintenance_work_mem PostgreSQL 參數的值。

Aurora PostgreSQL 的巨型分頁

巨型分頁是一項記憶體管理功能，可減少資料庫執行個體處理大型連續記憶體區塊 (如共用緩衝區使用的記憶體區塊) 時的額外負荷。所有目前可用的 Aurora PostgreSQL 版本都支援此 PostgreSQL 功能。

對於 t3.medium,db.t3.large、db.t4g.medium、db.t4g.large 類別以外的所有資料庫執行個體類別，預設為開啟 Huge_pages 參數。您無法在 Aurora PostgreSQL 的受支援執行個體類別中變更 huge_pages 參數值或關閉此功能。

在不支援巨型分頁記憶體功能的 Aurora PostgreSQL 資料庫執行個體，特定程序記憶體用量可能會增加，而不會造成對應的工作負載變更。

系統會在伺服器啟動期間分配共用記憶體區段，例如緩衝區快取。當巨型記憶體分頁無法使用時，系統不會向郵件管理員程序收取這些分配的費用。相反地，其包含在共用記憶體區段中第一次存取每個 4KB 頁面的程序中的記憶體。

> **Note**
>
> 無論跨程序追蹤共用記憶體用量的方式為何，作用中連線都會視需要共用分配的記憶體。

使用錯誤注入查詢測試 Amazon Aurora PostgreSQL

您可以使用錯誤注入查詢來測試 Aurora PostgreSQL 資料庫叢集的容錯能力。錯誤注入查詢作為 SQL 命令發出到 Amazon Aurora 執行個體。故障注入查詢可讓您損毀執行個體，以便測試容錯移轉和復原。您也可以模擬 Aurora 複本故障、磁碟故障和磁碟擁塞。所有可用的 Aurora PostgreSQL 版本都支援故障插入查詢，如下所示。

- Aurora PostgreSQL 版本 12、13、14 版和更新版本
- Aurora PostgreSQL 版本 11.7 版和更新版本
- Aurora PostgreSQL 版本 10.11 版和更新版本

主題

- 測試執行個體當機

- 測試 Aurora 複本失敗
- 測試磁碟失敗
- 測試磁碟壅塞

當故障注入查詢指定當機時，它會強制 Aurora PostgreSQL 資料庫執行個體發生當機。其他錯誤注入查詢會造成失敗事件的模擬，但不會造成事件發生。提交錯誤注入查詢時，您也可以指定讓失敗事件模擬發生的時間。

透過連線至 Aurora 複本的端點，您可以將錯誤注入查詢提交給您的其中一個 Aurora 複本執行個體。如需更多詳細資訊，請參閱 Amazon Aurora 端點連線。

測試執行個體當機

您可以使用錯誤注入查詢函數 `aurora_inject_crash()` 來強制讓 Aurora PostgreSQL 執行個體當機。

針對此錯誤注入查詢，將不會發生容錯移轉。如果您要測試容錯移轉，則可以在 RDS 主控台為您的資料庫叢集選擇 Failover (容錯移轉) 執行個體動作，或是使用 failover-db-cluster AWS CLI 命令或 FailoverDBCluster RDS API 操作。

語法

```
SELECT aurora_inject_crash ('instance' | 'dispatcher' | 'node');
```

選項

此錯誤注入查詢需要下列其中一個當機類型。當機類型不區分大小寫：

'instance'

模擬 Amazon Aurora 執行個體之 PostgreSQL 相容資料庫的當機情況。

'發送器'

模擬 Aurora 資料庫叢集主要執行個體上發送器的當機情況。發送器會將更新寫入 Amazon Aurora 資料庫叢集的叢集磁碟區。

'node'

模擬 PostgreSQL 相容資料庫和 Amazon Aurora 執行個體發送器的當機情況。

測試 Aurora 複本失敗

您可以使用錯誤注入查詢函數 `aurora_inject_replica_failure()` 來模擬 Aurora 複本的失敗。

Aurora 複本失敗會根據指定時間間隔的指定百分比，封鎖與資料庫叢集中的 Aurora 複本或所有 Aurora 複本的複寫。當時間間隔完成，受影響的 Aurora 複本將與主要執行個體自動同步。

語法

```
SELECT aurora_inject_replica_failure(
   percentage_of_failure,
   time_interval,
   'replica_name'
);
```

選項

此錯誤注入查詢會使用下列參數：

percentage_of_failure

在失敗事件期間封鎖的複寫百分比。此值可以是介於 0 與 100 之間的雙位數數字。如果指定 0，則不會封鎖任何複寫。如果指定 100，則會封鎖所有複寫。

time_interval

模擬 Aurora 複本失敗的時間量。間隔以秒為單位。例如，如果值為 20，則模擬會執行 20 秒。

> **Note**
>
> 指定 Aurora 複本失敗事件的時間間隔時請注意。如果指定的間隔太長，而您的寫入器執行個體在失敗事件期間寫入大量資料，則 Aurora 資料庫叢集可能會假設 Aurora 複本已當機並加以取代。

replica_name

要在其中注入失敗模擬的 Aurora 複本。指定 Aurora 複本的名稱，以模擬單一 Aurora 複本的失敗。指定空字串以模擬資料庫叢集中所有 Aurora 複本的失敗。

若要識別複本名稱，請參閱 `server_id` 函數中的 `aurora_replica_status()` 欄。例如：

```
postgres=> SELECT server_id FROM aurora_replica_status();
```

測試磁碟失敗

您可以使用錯誤注入查詢函數 `aurora_inject_disk_failure()` 來模擬 Aurora PostgreSQL 資料庫叢集的磁碟失敗。

在磁碟失敗模擬期間，Aurora PostgreSQL 資料庫叢集會隨機將磁碟區段標示為發生錯誤。在模擬期間將封鎖對那些區段的請求。

語法

```
SELECT aurora_inject_disk_failure(
   percentage_of_failure,
   index,
   is_disk,
   time_interval
);
```

選項

此錯誤注入查詢會使用下列參數：

percentage_of_failure

在失敗事件期間標示為發生錯誤的磁碟百分比。此值可以是介於 0 與 100 之間的雙位數數字。如果指定 0，則不會將任何磁碟標示為發生錯誤。如果指定 100，則會將整個磁碟標示為發生錯誤。

index

模擬失敗事件的特定資料邏輯區塊。如果超出可用的資料邏輯區塊或儲存節點資料的範圍，您會收到錯誤，告訴您可以指定的索引值上限。若要避免此錯誤，請參閱 顯示 Aurora PostgreSQL 資料庫叢集的磁碟區狀態。

is_disk

指出注入失敗是否為邏輯區塊或儲存節點。指定 true 代表將注入失敗設定為邏輯區塊。指定 false 代表注入失敗設定為儲存節點。

time_interval

模擬磁碟失敗的時間量。間隔以秒為單位。例如，如果值為 20，則模擬會執行 20 秒。

測試磁碟壅塞

您可以使用錯誤注入查詢函數 `aurora_inject_disk_congestion()` 來模擬 Aurora PostgreSQL 資料庫叢集的磁碟壅塞。

在磁碟擁塞模擬期間，Aurora PostgreSQL 資料庫叢集會隨機將磁碟區段標示為擁塞。對那些區段的請求延遲將介於模擬期間的指定最小和最大延遲時間之間。

語法

```
SELECT aurora_inject_disk_congestion(
   percentage_of_failure,
   index,
   is_disk,
   time_interval,
   minimum,
   maximum
);
```

選項

此錯誤注入查詢會使用下列參數：

percentage_of_failure

在失敗事件期間標示為擁塞的磁碟百分比。這是介於 0 和 100 之間的雙倍值。如果指定 0，則不會將任何磁碟標示為擁塞。如果指定 100，則會將整個磁碟標示為擁塞。

index

用於模擬失敗事件的資料特定邏輯區塊或儲存節點。

如果超出可用的資料邏輯區塊或儲存節點資料的範圍，您會收到錯誤，告訴您可以指定的最大索引值。若要避免此錯誤，請參閱 顯示 Aurora PostgreSQL 資料庫叢集的磁碟區狀態。

is_disk

指出注入失敗是否為邏輯區塊或儲存節點。指定 true 代表將注入失敗設定為邏輯區塊。指定 false 代表注入失敗設定為儲存節點。

time_interval

模擬磁碟壅塞的時間量。間隔以秒為單位。例如，如果值為 20，則模擬會執行 20 秒。

最小值、最大值

最小數量和最大數量擁塞延遲 (以毫秒為單位)。有效值的範圍為 0.0 至 100.0 毫秒。標示為壅塞的磁碟區段將會於模擬期間在最小和最大數量的範圍內延遲隨機的時間。最大值必須大於最小值。

顯示 Aurora PostgreSQL 資料庫叢集的磁碟區狀態

在 Amazon Aurora 中，資料庫叢集磁碟區包含邏輯區塊的集合。每個項目都代表 10 GB 的配置儲存體。這些區塊稱為保護群組。

每個保護群組中的資料會在六個實體儲存體裝置 (稱為儲存節點) 間複寫。這些儲存節點會在資料庫叢集所在區域中的三個可用區域 (AZ) 中配置。每個儲存節點依次包含資料庫叢集磁碟區的一或多個資料邏輯區塊。如需保護群組和儲存節點的詳細資訊，請參閱 AWS 資料庫部落格上的介紹 Aurora 儲存引擎。若要進一步了解 Aurora 叢集磁碟區的一般資訊，請參閱 Amazon Aurora 儲存體。

使用 `aurora_show_volume_status()` 函數傳回下列伺服器狀態變數：

- `Disks`—資料庫叢集磁碟區的邏輯區塊總數。
- `Nodes` —資料庫叢集磁碟區的儲存節點總數。

您可以使用 `aurora_show_volume_status()` 函數，以協助避免在使用錯誤注入函數 `aurora_inject_disk_failure()` 時發生錯誤。`aurora_inject_disk_failure()` 錯誤注入函數會模擬整個儲存節點或儲存節點內單一資料邏輯區塊的失敗。在函數中，您可以指定特定資料邏輯區塊或儲存節點的索引值。不過，如果您指定的索引值大於資料邏輯區塊或資料庫叢集磁碟區使用的儲存節點的數量，陳述式會傳回錯誤。如需錯誤注入查詢的詳細資訊，請參閱 使用錯誤注入查詢測試 Amazon Aurora PostgreSQL。

> **Note**
>
> `aurora_show_volume_status()` 函數適用於 Aurora PostgreSQL 10.11 版。如需 Aurora PostgreSQL 版本的詳細資訊，請參閱 Amazon Aurora PostgreSQL 版本和引擎版本。

語法

```
SELECT * FROM aurora_show_volume_status();
```

範例

```
customer_database=> SELECT * FROM aurora_show_volume_status();
 disks | nodes
-------+-------
    96 |    45
```

指定 stats_temp_directory 的 RAM 磁碟

您可以使用 Aurora PostgreSQL 參數 `rds.pg_stat_ramdisk_size` 指定配置給 RAM 磁碟的系統記憶體，以存放 PostgreSQL `stats_temp_directory`。RAM 磁碟參數僅適用於Aurora PostgreSQL 14 及更低版本。

在某些工作負載下，設定此參數可以提高效能和降低 IO 需求。如需關於 `stats_temp_directory` 的詳細資訊，請參閱 PostgreSQL 文件中的執行時間統計資料。從第 15 版的 PostgreSQL 開始，PostgreSQL 社群改為使用動態共用記憶體。因此，沒有必要設定 `stats_temp_directory`。

若要為 `stats_temp_directory` 啟用 RAM 磁碟，請在資料庫叢集使用的資料庫叢集參數群組中，將 `rds.pg_stat_ramdisk_size` 參數設為非零值。此參數表示 MB，因此您必須使用整數值。表達式、公式和函數對 `rds.pg_stat_ramdisk_size` 參數無效。請務必重新啟動資料庫叢集，變更才會生效。如需有關設定參數的詳細資訊，請參閱 Amazon Aurora 的參數群組。如需重新啟動資料庫叢集的詳細資訊，請參閱 重新啟動 Amazon Aurora 資料庫叢集或 Amazon Aurora 資料庫執行個體。

例如，下列 AWS CLI 命令會將 RAM 磁碟參數設定為 256 MB。

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name db-cl-pg-ramdisk-testing \
    --parameters "ParameterName=rds.pg_stat_ramdisk_size, ParameterValue=256,
 ApplyMethod=pending-reboot"
```

重新啟動資料庫叢集之後，執行以下命令來查看 `stats_temp_directory` 的狀態：

```
postgres=> SHOW stats_temp_directory;
```

此命令應該會傳回下列結果：

```
stats_temp_directory
----------------------------
/rdsdbramdisk/pg_stat_tmp
(1 row)
```

使用 PostgreSQL 管理暫存檔案

在 PostgreSQL 中，複雜查詢可能會同時執行數個排序或雜湊操作，每個操作都使用執行個體記憶體來儲存結果，直到 work_mem 參數中指定的值為止。當執行個體記憶體不足時，會建立暫存檔來儲存結果。這些檔案會寫入磁碟以完成查詢執行。稍後，這些檔案會在查詢完成後自動移除。在 Amazon PostgreSQL 中，這些檔案與其他日誌檔案共用本機儲存體。您可以透過監看 FreeLocalStorage 的 Amazon CloudWatch 指標，來監控 Aurora PostgreSQL 資料庫叢集的本機儲存空間。如需詳細資訊，請參閱疑難排解本機儲存問題。

我們建議針對涉及多個並行查詢的工作負載使用 Aurora Optimized Read 叢集，以增加暫存檔案的使用量。這些叢集會使用本機非揮發性記憶體快速 (NVMe) 為主的固態硬碟 (SSD) 區塊層級儲存體來放置暫存檔案。如需詳細資訊，請參閱 使用 Aurora Optimized Reads 改善 Aurora PostgreSQL 的查詢效能。

您可以使用以下參數和函數來管理執行個體中的暫存檔案。

- **temp_file_limit** – 此參數會取消任何超過 temp_file 大小的查詢 (以 KB 為單位)。此限制可防止任何查詢無休止地執行，並消耗含有暫存檔的磁碟空間。您可以使用 log_temp_files 參數的結果來估計值。最佳實務是檢查工作負載行為並根據估計值設定限制。以下範例顯示查詢超過限制時的取消方式。

```
postgres=>select * from pgbench_accounts, pg_class, big_table;
```

```
ERROR: temporary file size exceeds temp_file_limit (64kB)
```

- **log_temp_files** – 此參數會在移除工作階段的暫存檔案時，將訊息傳送至 postgresql.log。這個參數會在查詢順利完成後產生日誌。因此，它可能無助於疑難排解使用中、長時間執行的查詢。

  下列範例顯示，當查詢順利完成時，這些項目會在清理暫存檔時記錄在 postgresql.log 檔案中。

```
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
 temporary file: path "base/pgsql_tmp/pgsql_tmp31236.5", size 140353536
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
 temporary file: path "base/pgsql_tmp/pgsql_tmp31236.4", size 180428800
```

```
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
```

- **pg_ls_tmpdir** – 此功能可從 RDS for PostgreSQL 13 及以上版本使用，提供目前暫存檔案使用情況的可見性。完成的查詢不會出現在函數的結果中。在下列範例中，您可以檢視此函數的結果。

```
postgres=>select * from pg_ls_tmpdir();
```

```
      name       |    size    |      modification
-----------------+------------+------------------------
 pgsql_tmp8355.1 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.0 | 1072250880 | 2023-02-06 22:54:43+00
 pgsql_tmp8327.0 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.1 |  703168512 | 2023-02-06 22:54:56+00
 pgsql_tmp8355.0 | 1072250880 | 2023-02-06 22:54:00+00
 pgsql_tmp8328.1 |  835031040 | 2023-02-06 22:54:56+00
 pgsql_tmp8328.0 | 1072250880 | 2023-02-06 22:54:40+00
(7 rows)
```

```
postgres=>select query from pg_stat_activity where pid = 8355;

query
-------------------------------------------------------------------------------------------
select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid
(1 row)
```

檔案名稱包含產生暫存檔之工作階段的處理 ID (PID)。較進階的查詢 (例如下列範例) 會針對每個 PID 執行暫存檔案的總和。

```
postgres=>select replace(left(name, strpos(name, '.')-1),'pgsql_tmp','') as pid,
 count(*), sum(size) from pg_ls_tmpdir() group by pid;
```

```
 pid  | count |   sum
```

```
------+------------------
 8355 |     2 | 2144501760
 8351 |     2 | 2090770432
 8327 |     1 | 1072250880
 8328 |     2 | 2144501760
(4 rows)
```

- **pg_stat_statements** – 如果您啟動 pg_stat_statements 參數，則可以檢視每次呼叫的平均暫存檔案使用量。您可以識別查詢的 query_id，並使用它來檢查暫存檔的使用情況，如以下範例所示。

```
postgres=>select queryid from pg_stat_statements where query like 'select a.aid from
 pgbench%';
```

```
       queryid
----------------------
 -7170349228837045701
(1 row)
```

```
postgres=>select queryid, substr(query,1,25), calls, temp_blks_read/calls
 temp_blks_read_per_call, temp_blks_written/calls temp_blks_written_per_call from
 pg_stat_statements where queryid = -7170349228837045701;
```

```
       queryid        |          substr           | calls | temp_blks_read_per_call |
 temp_blks_written_per_call
----------------------+---------------------------+-------+-------------------------
+---------------------------
 -7170349228837045701 | select a.aid from pgbench |    50 |                  239226 |
                    388678
(1 row)
```

- **Performance Insights** – 在績效詳情儀表板中，您可以透過開啟指標 temp_bytes 和 temp_files 來檢視暫存檔的使用情況。然後，您可以查看這兩個指標的平均值，並查看它們如何對應至查詢工作負載。績效詳情中的檢視不會明確顯示產生暫存檔的查詢。不過，當您將績效詳情與為 pg_ls_tmpdir 顯示的查詢結合使用時，您可以疑難排解、分析及判斷查詢工作負載中的變更。

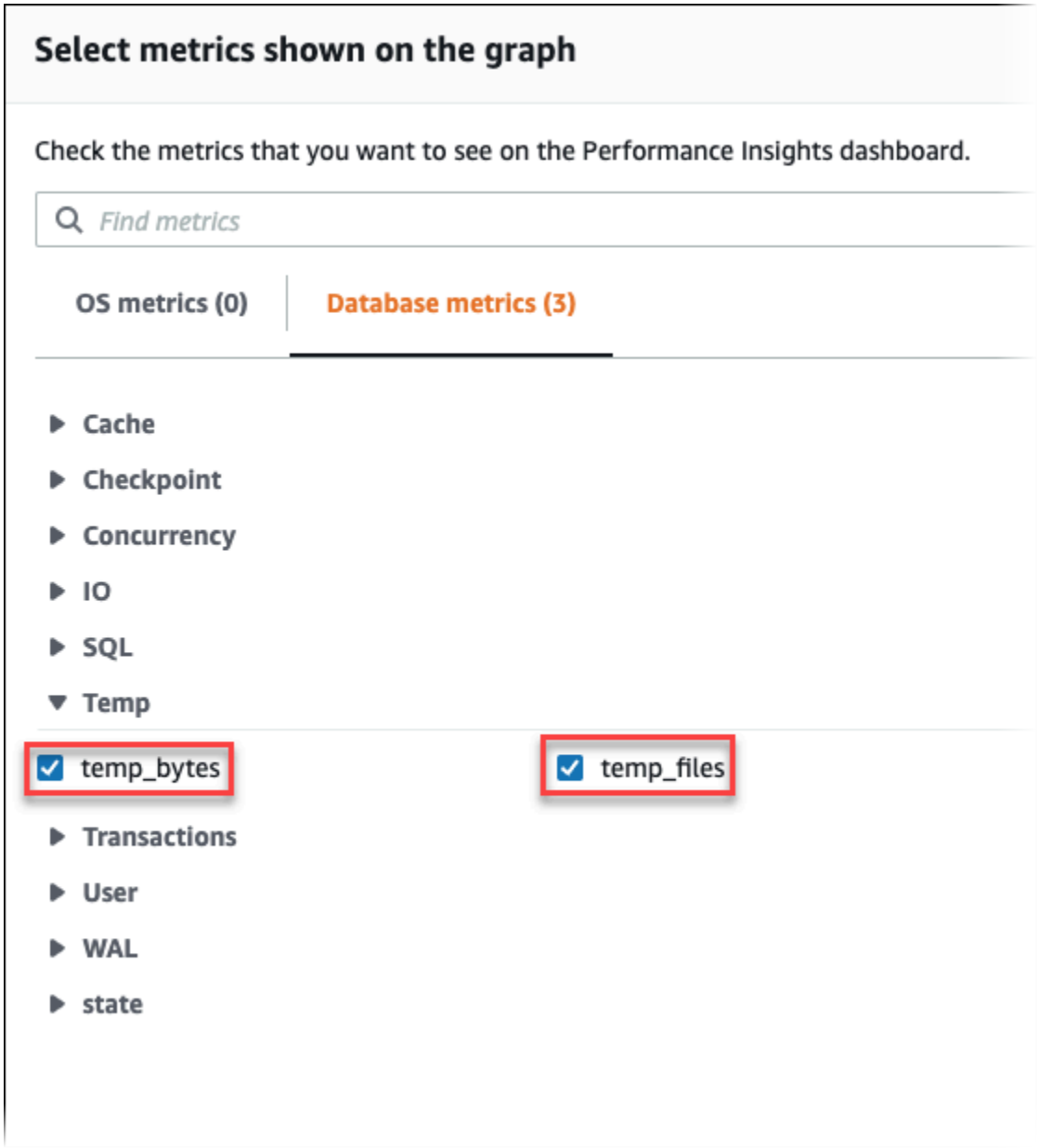
如需如何使用 Performance Insights 來分析指標和查詢的詳細資訊，請參閱 使用績效詳情儀表板來分析指標。

如需使用 Performance Insights 檢視暫存檔案使用量的範例，請參閱 使用 Performance Insights 來檢視暫存檔使用情況

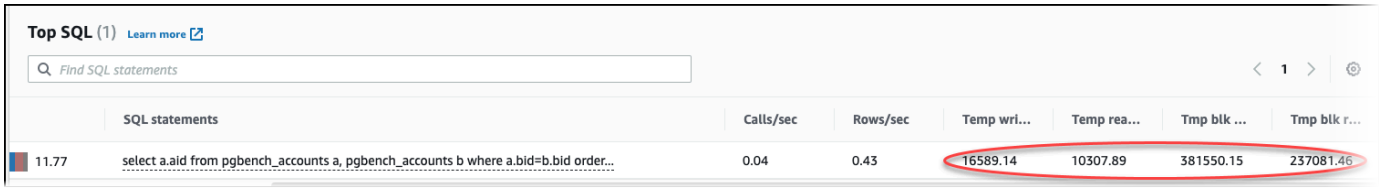
使用 Performance Insights 來檢視暫存檔使用情況

您可以使用 Performance Insights，透過開啟指標 temp_bytes 和 temp_files 來檢視暫存檔的使用情況。Performance Insights 中的檢視不會顯示產生暫存檔的特定查詢，但是，當您將 Performance Insights 與針對 `pg_ls_tmpdir` 顯示的查詢結合時，您可以疑難排解、分析和判斷查詢工作負載中的變更。

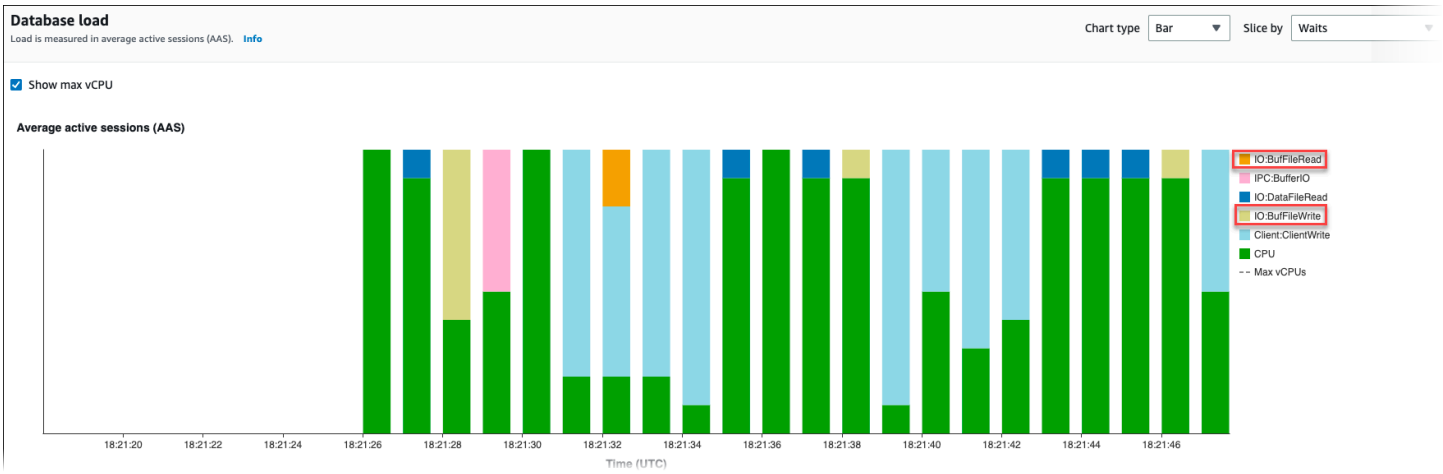
1. 在 [績效詳情] 儀表板中選擇管理指標。
2. 選擇資料庫指標，並選取 temp_bytes 和 temp_files 指標，如下方影像所示。



3. 在最高 SQL索引標籤中，選擇偏好設定圖示。
4. 在偏好設定視窗中，開啟最高 SQL索引標籤中顯示的下列統計資料，然後選擇繼續。
   - Temp writes/sec
   - Temp reads/sec
   - Tmp blk write/call
   - Tmp blk read/call
5. 暫存檔在與針對 `pg_ls_tmpdir` 顯示的查詢組合時會被劃分，如以下範例所示。



當工作負載中最常用的查詢經常建立暫存檔案時，就會發生 `IO:BufFileRead` 和 `IO:BufFileWrite` 事件。您可以使用 Performance Insights，透過檢閱「資料庫負載」和「最高 SQL」區段中的「平均作用中工作階段」(AAS)，找出最常在 `IO:BufFileRead` 和 `IO:BufFileWrite` 上等待的查詢。



如需如何使用 Performance Insights 來分析各等待事件的最常用查詢和負載的詳細資訊，請參閱 最高 SQL 索引標籤概觀。您應找出並調整造成暫存檔案使用量及相關等待事件增加的查詢。如需這些等待事件和修補的詳細資訊，請參閱 IO:BufFileRead 和 IO:BufFileWrite。

> **Note**
>
> `work_mem` 參數可控制排序操作何時用完記憶體，以及將結果寫入暫存檔。我們建議您不要將此參數的設定變更為高於預設值，因為它會允許每個資料庫工作階段耗用更多記憶體。此外，執行複雜聯結和排序的單一工作階段可以執行平行操作，其中每個操作都會耗用記憶體。最佳實務是，當您有具有多個聯結和排序的大型報表時，請使用 `SET work_mem` 命令在工作階段層級設定此參數。然後，變更僅套用於目前工作階段，不會全域變更該值。

調校 Aurora PostgreSQL 的等待事件

等待事件是 Aurora PostgreSQL 的重要調校工具。如果您了解工作階段為何等待資源及其運作情形，就更能夠減少瓶頸。您可以使用本節的資訊來尋找可能的原因和更正動作。深入研究本節之前，我們強烈建議您先了解 Aurora 的基本概念，尤其是下列主題：

- Amazon Aurora 儲存體
- 管理 Aurora 資料庫叢集的效能和擴展

> **Important**
>
> 本節中的等待事件專屬於 Aurora PostgreSQL。本節的資訊僅適用於調校 Amazon Aurora，而不是 RDS for PostgreSQL。
>
> 本節部分等待事件在這些資料庫引擎的開放原始碼版本中沒有對等的事件。其他等待事件與開放原始碼引擎中的事件同名，但行為不同。例如，Amazon Aurora 儲存與開放原始碼儲存的運作方式不同，因此與儲存相關的等待事件表示不同的資源條件。

主題

- Aurora PostgreSQL 調校的基本概念
- Aurora PostgreSQL 等待事件
- Client:ClientRead
- Client:ClientWrite
- CPU
- IO:BufFileRead 和 IO:BufFileWrite
- IO:DataFileRead
- IO:XactSync
- IPC:DamRecordTxAck
- IPC:parallel 等待事件
- IPC:ProcArrayGroupUpdate
- Lock:advisory
- Lock:extend
- Lock:Relation

- Lock:transactionid
- Lock:tuple
- LWLock:buffer_content (BufferContent)
- LWLock:buffer_mapping
- LWLock:BufferIO (IPC:BufferIO)
- LWLock:lock_manager
- LWLock:MultiXact
- LWLock:pg_stat_statements
- Timeout:PgSleep

Aurora PostgreSQL 調校的基本概念

調校 Aurora PostgreSQL 資料庫之前，請務必先瞭解什麼是等待事件及其發生原因。也請檢閱 Aurora PostgreSQL 的基本記憶體和磁碟架構。如需實用的架構圖，請參閱 PostgreSQL 維基教科書。

主題

- Aurora PostgreSQL 等待事件
- Aurora PostgreSQL 記憶體
- Aurora PostgreSQL 程序

Aurora PostgreSQL 等待事件

「等待事件」表示工作階段正在等待的資源。例如，當 Aurora PostgreSQL 等待從用戶端接收資料時會發生等待事件 `Client:ClientRead`。工作階段等待的典型資源包括：

- 透過單一執行緒存取緩衝區，例如，當工作階段嘗試修改緩衝區時
- 另一個工作階段目前鎖定的資料列
- 資料檔讀取
- 日誌檔寫入

例如，為了滿足查詢，工作階段可能執行完整的資料表掃描。如果資料不在記憶體中，工作階段會等待磁碟輸入/輸出完成。將緩衝區讀入記憶體後，工作階段可能需要等待，因為其他工作階段正在存取這些緩衝區。資料庫使用預先定義的等待事件來記錄等待。這些事件分組為多個類別。

等待事件本身不表示有效能問題。例如，如果請求的資料不在記憶體中，則需要從磁碟讀取資料。如果一個工作階段鎖定資料列來更新，則另一個工作階段要等待此資料列解除鎖定才能更新。遞交需要等待寫入日誌檔完成。等待是資料庫正常運作所不可或缺。

大量等待事件通常表示有效能問題。在這種情況下，您可以使用等待事件資料來判斷工作階段將時間花在何處。例如，如果報告通常執行幾分鐘，但現在執行數小時，您可以識別佔總等待時間最多的等待事件。如果您可以查出最常等待事件的原因，通常就能做些改變來改善效能。例如，如果工作階段等待的資料列被另一個工作階段鎖定，您可以結束該鎖定工作階段。

Aurora PostgreSQL 記憶體

Aurora PostgreSQL 記憶體分成共用和本機。

主題

- Aurora PostgreSQL 中的共用記憶體
- Aurora PostgreSQL 中的本機記憶體

Aurora PostgreSQL 中的共用記憶體

Aurora PostgreSQL 在執行個體啟動時配置共用記憶體。共用記憶體分成多個子區域。接下來，您可以看到最重要子區域的描述。

主題

- 共用緩衝區
- 預寫日誌 (WAL) 緩衝區

共用緩衝區

共用緩衝集區是一種 Aurora PostgreSQL 記憶體區域，其中保留應用程式連線正在使用或已使用的所有分頁。分頁是記憶體形式的磁碟區塊。共用緩衝集區快取從磁碟讀取的資料區塊。集區可減少從磁碟重新讀取資料的次數，讓資料庫運作更有效率。

每個資料表和索引儲存為一連串固定大小的分頁。每個區塊包含多個元組，對應於資料列。元組可以存放在任何分頁中。

共用緩衝集區的記憶體有限。如果新請求需要的分頁不在記憶體中，且已沒有更多記憶體，Aurora PostgreSQL 會移出較不常用的分頁來容納請求。移出政策以時鐘掃描演算法實作。

`shared_buffers` 參數決定伺服器專用於快取資料的記憶體數量。

預寫日誌 (WAL) 緩衝區

預寫日誌 (WAL) 緩衝區保留交易資料，供 Aurora PostgreSQL 稍後寫入持久性儲存。Aurora PostgreSQL 可以透過 WAL 機制達成下列目標：

- 在故障後復原資料
- 避免頻繁寫入磁碟以減少磁碟輸入/輸出

當用戶端變更資料時，Aurora PostgreSQL 會將變更寫入 WAL 緩衝區。當用戶端發出 `COMMIT` 時，WAL 寫入器程序會將交易資料寫入 WAL 檔案。

`wal_level` 參數決定將多少資訊寫入 WAL。

Aurora PostgreSQL 中的本機記憶體

每個後端程序會配置本機記憶體來處理查詢。

主題

- 工作記憶體區域
- 維護工作記憶體區域
- 暫時緩衝區域

工作記憶體區域

工作記憶體區域為執行排序和雜湊的查詢保留暫存資料。例如，含有 `ORDER BY` 子句的查詢執行排序。查詢在雜湊聯結和彙總中使用雜湊表。

`work_mem` 參數指定在寫入暫存磁碟檔案之前，供內部排序操作和雜湊表使用的記憶體數量。預設值為 4 MB。多個工作階段可以同時執行，每個工作階段可以平行執行維護操作。因此，使用的總工作記憶體可能是 `work_mem` 設定的倍數。

維護工作記憶體區域

維護工作記憶體區域快取維護操作的資料。這些操作包括清理、建立索引和新增外部索引鍵。

`maintenance_work_mem` 參數指定供維護操作使用的記憶體數量上限。預設值為 64 MB。一個資料庫工作階段一次只能執行一個維護操作。

暫時緩衝區域

暫時緩衝區域快取每個資料庫工作階段的暫存資料表。

每個工作階段視需要配置暫存緩衝區，以您指定的限制為上限。工作階段結束時，伺服器會清除緩衝區。

`temp_buffers` 參數設定每個工作階段使用的暫存緩衝區數目上限。在工作階段內第一次使用暫存資料表之前，您可以變更 `temp_buffers` 值。

Aurora PostgreSQL 程序

Aurora PostgreSQL 使用多個程序。

主題

- 郵件管理員程序
- 後端程序
- 背景程序

郵件管理員程序

郵件管理員程序是您啟動 Aurora PostgreSQL 時啟動的第一個程序。郵件管理員程序有下列主要責任：

- 分叉和監控背景程序
- 接收來自用戶端程序的身分驗證請求，並在驗證請求之後才允許資料庫處理請求

後端程序

如果郵件管理員驗證用戶端請求，郵件管理員會分叉新的後端程序，也稱為 postgres 程序。一個用戶端程序只連線到一個後端程序。用戶端程序和後端程序直接通訊，無須郵件管理員程序介入。

背景程序

郵件管理員程序分叉幾個程序來執行不同的後端任務。一些較重要的程序包括：

- WAL 寫入器

  Aurora PostgreSQL 將 WAL (預寫日誌) 緩衝區中的資料寫入日誌檔。預寫日誌的原則是直到資料庫將描述變更的日誌記錄寫入磁碟之後，資料庫才能將這些變更寫入資料檔。WAL 機制可減少磁碟輸入/輸出，並允許 Aurora PostgreSQL 在故障後使用日誌來復原資料庫。
- 背景寫入器

這個程序定期將已變更 (已修改) 分頁從記憶體緩衝區寫入資料檔。當後端程序在記憶體中修改分頁時，此分頁會變成已變更。

- 自動資料清理常駐程式

  此常駐程式由下列組成：

  - 自動資料清理啟動器
  - 自動資料清理工作者程序

  自動資料清理啟用時會檢查已插入、更新或刪除大量元組的資料表。此常駐程式有下列責任：

  - 復原或重複使用已更新或刪除的資料列所佔用的磁碟空間
  - 更新規劃員使用的統計數字
  - 防止因交易 ID 環繞而遺失舊資料

  自動資料清理功能可自動執行 VACUUM 和 ANALYZE 命令。VACUUM 有以下變體：標準和完整。標準清理與其他資料庫操作平行執行。VACUUM FULL 需要獨佔鎖定其處理的資料表。因此，無法與存取同一個資料表的操作平行執行。VACUUM 會建立大量輸入/輸出流量，可能導致其他作用中工作階段的效能不佳。

Aurora PostgreSQL 等待事件

下表列出 Aurora PostgreSQL 中最常表示有效能問題的等待事件，並概述最常見的原因和更正動作。下列等待事件是 Amazon Aurora PostgreSQL 等待事件中的清單子集。

等待事件	定義
Client:ClientRead	此事件表示 Aurora PostgreSQL 正在等待從用戶端接收資料。
Client:ClientWrite	此事件表示 Aurora PostgreSQL 正在等待將資料寫入用戶端。
CPU	此事件表示執行緒活躍於 CPU 中或正在等待 CPU。
IO:BufFileRead 和 IO:BufFileWrite	這些事件表示 Aurora PostgreSQL 建立暫存檔。

等待事件	定義
IO:DataFileRead	此事件表示連線等待後端程序從儲存讀取必要分頁，因為共用記憶體中沒有此分頁。
IO:XactSync	此事件表示資料庫正在等待 Aurora 儲存子系統確認遞交一般交易，或是遞交或回復備妥交易。
IPC:DamRecordTxAck	此事件表示 Aurora PostgreSQL 在使用資料庫活動串流的工作階段中產生活動串流事件，然後等待該事件變得持久。
Lock:advisory	此事件表示 PostgreSQL 應用程式使用鎖定在多個工作階段之間協調活動。
Lock:extend	此事件表示後端程序正在等待鎖定關聯來延伸，但另一個程序也基於相同目的而鎖定該關係。
Lock:Relation	此事件表示查詢正等待在目前由另一個交易鎖定的資料表或檢視表上取得鎖定。
Lock:transactionid	此事件表示交易正在等待資料列層級鎖定。
Lock:tuple	此事件表示後端程序正等待在元組上取得鎖定。
LWLock:buffer_content (BufferContent)	此事件表示工作階段正等待在記憶體中讀取或寫入資料分頁，但另一個工作階段已鎖定該分頁來寫入。
LWLock:buffer_mapping	此事件表示工作階段正在等待將資料區塊與共用緩衝集區中的緩衝區建立關聯。
LWLock:BufferIO (IPC:BufferIO)	此事件表示 Aurora PostgreSQL 或 RDS for PostgreSQL 與其他程序同時嘗試存取分頁，正在等待其他程序完成輸入/輸出 (輸入/輸出) 操作。
LWLock:lock_manager	此事件表示因為無法執行快速路徑鎖定，Aurora PostgreSQL 引擎維護共用鎖定的記憶體區域來配置、檢查和解除配置鎖定。

等待事件	定義
LWLock:MultiXact	當 Aurora PostgreSQL 將工作階段保持開啟狀態，以完成多個涉及資料表中相同資料列的交易時，就會發生這種類型的事件。等待事件指出多重交易處理的哪個層面正在產生等待事件，亦即 LWLock:MultiXactOffsetSLRU、LWLock:MultiXactOffsetBuffer、LWLock:MultiXactMemberSLRU 或 LWLock:MultiXactMemberBuffer。
Timeout:PgSleep	此事件表示伺服器程序已呼叫 pg_sleep 函數，正在等待睡眠逾時到期。

Client:ClientRead

`Client:ClientRead` 事件表示 Aurora PostgreSQL 正在等待從用戶端接收資料。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因
- 動作

支援的引擎版本

Aurora PostgreSQL 第 10 版及更新版本支援此等待事件資訊。

Context

Aurora PostgreSQL 資料庫叢集正在等待從用戶端接收資料。Aurora PostgreSQL 資料庫叢集必須先從用戶端接收資料，才能將更多資料傳送至用戶端。叢集從用戶端接收資料之前等待的時間是 `Client:ClientRead` 事件。

等待變多的可能原因

`Client:ClientRead` 事件出現在最常等待名單中的常見原因包括：

網路延遲較久

Aurora PostgreSQL 資料庫叢集與用戶端之間的網路延遲可能變長。網路延遲較久會導致資料庫叢集需要更多時間從用戶端接收資料。

用戶端的負載增加

用戶端可能面臨 CPU 壓力或網路飽和。用戶端的負載增加可能延遲將資料從用戶端傳輸至 Aurora PostgreSQL 資料庫叢集。

網路往返太頻繁

如果 Aurora PostgreSQL 資料庫叢集與用戶端之間網路往返太頻繁，可能會延遲將資料從用戶端傳輸至 Aurora PostgreSQL 資料庫叢集。

複製操作龐大

在複製操作期間，資料從用戶端的檔案系統傳輸至 Aurora PostgreSQL 資料庫叢集。如果將大量資料傳送至資料庫叢集，可能會延遲將資料從用戶端傳輸至資料庫叢集。

閒置用戶端連線

Aurora PostgreSQL 資料庫執行個體的連線處於交易閒置狀態，並等待用戶端傳送更多資料或發出命令。此狀態可能導致 `Client:ClientRead` 事件增加。

連線集區使用的 PgBuncer

PgBuncer 有一個低階網路組態設定稱為 `pkt_buf`，預設為 4,096。如果工作負載透過 PgBuncer 傳送大於 4,096 個位元組的查詢封包，建議將 `pkt_buf` 設定增加為 8,192。如果新的設定未能減少 `Client:ClientRead` 事件，建議將 `pkt_buf` 設定增加為更大的值，例如 16,384 或 32,768。如果查詢文字很大，則較大的設定可能特別有用。

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 將用戶端放在與叢集相同的可用區域和 VPC 子網路中
- 擴展用戶端
- 使用最新一代的執行個體
- 增加網路頻寬

- 監控網路效能的最大值
- 監控處於 "idle in transaction" 狀態的交易

將用戶端放在與叢集相同的可用區域和 VPC 子網路中

若要縮短網路延遲並增加網路輸送量，請將用戶端放在與 Aurora PostgreSQL 資料庫叢集相同的可用區域和 Virtual Private Cloud (VPC) 子網路中。確保用戶端的地理位置盡可能靠近資料庫叢集。

擴展用戶端

使用 Amazon CloudWatch 或其他主機指標，判斷用戶端目前是否受限於 CPU、網路頻寬，或同時受限於這兩者。如果用戶端受到限制，請相應地擴展用戶端。

使用最新一代的執行個體

在某些情況下，您使用的資料庫執行個體類別可能不支援 Jumbo Frame。如果您在 Amazon EC2 上執行應用程式，請考慮為用戶端使用最新一代的執行個體。此外，在用戶端作業系統上設定最大傳輸單位 (MTU)。此技術可以減少網路往返次數並增加網路輸送量。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Jumbo Frame (9001 MTU)。

如需資料庫執行個體類別的相關資訊，請參閱 Amazon Aurora 資料庫執行個體類別。若要決定等同於 Amazon EC2 執行個體類型的資料庫執行個體類別，請在 Amazon EC2 執行個體類型名稱前面加上 `db.`。例如，`r5.8xlarge` Amazon EC2 執行個體等同於 `db.r5.8xlarge` 資料庫執行個體類別。

增加網路頻寬

使用 `NetworkReceiveThroughput` 和 `NetworkTransmitThroughput` Amazon CloudWatch 指標來監控資料庫叢集上的傳入和傳出網路流量。這些指標可協助您判斷網路頻寬是否足以應付工作負載。

如果網路頻寬不夠，請增加頻寬。如果 AWS 用戶端或資料庫執行個體達到網路頻寬限制，增加頻寬的唯一方法是增加資料庫執行個體大小。

如需 CloudWatch 指標的詳細資訊，請參閱Amazon Aurora 的 Amazon CloudWatch 指標。

監控網路效能的最大值

如果您使用 Amazon EC2 用戶端，Amazon EC2 提供網路效能指標的最大值，包括傳入和傳出網路總頻寬。也提供連線追蹤來確保如預期傳回封包，還有網域名稱系統 (DNS) 等服務的連結本機服務存取。若要監控這些最大值，請使用目前的增強型網路驅動程式，以監控用戶端的網路效能。

如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的監控 Amazon EC2 執行個體的網路效能和《Amazon EC2使用者指南》中的監控 Amazon EC2 執行個體的網路效能。

監控處於 "idle in transaction" 狀態的交易

檢查 `idle in transaction` 連線是否變多。作法是監控 `pg_stat_activity` 資料表的 `state` 資料欄。您可以執行類似下列的查詢來識別連線來源。

```
select client_addr, state, count(1) from pg_stat_activity
where state like 'idle in transaction%'
group by 1,2
order by 3 desc
```

Client:ClientWrite

`Client:ClientWrite` 事件表示 Aurora PostgreSQL 正在等待將資料寫入用戶端。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因
- 動作

支援的引擎版本

Aurora PostgreSQL 第 10 版及更新版本支援此等待事件資訊。

Context

用戶端程序必須先讀取從 Aurora PostgreSQL 資料庫叢集收到的所有資料，叢集才能傳送更多資料。叢集將更多資料傳送至用戶端之前等待的時間是 `Client:ClientWrite` 事件。

Aurora PostgreSQL 資料庫叢集與用戶端之間的網路輸送量減少可能造成此事件。用戶端的 CPU 壓力和網路飽和也可能造成此事件。CPU 壓力表示 CPU 耗盡，但有任務正在等待 CPU 時間。網路飽和表示資料庫與用戶端之間的網路傳送太多資料，應付不來。

等待變多的可能原因

`Client:ClientWrite` 事件出現在最常等待名單中的常見原因包括：

網路延遲較久

Aurora PostgreSQL 資料庫叢集與用戶端之間的網路延遲可能變長。網路延遲較久會導致用戶端需要更多時間來接收資料。

用戶端的負載增加

用戶端可能面臨 CPU 壓力或網路飽和。用戶端的負載增加會延遲從 Aurora PostgreSQL 資料庫叢集接收資料。

傳送大量資料給用戶端

Aurora PostgreSQL 資料庫叢集可能將大量資料傳送至用戶端。用戶端可能來不及接收叢集傳送的資料。例如，複製大型資料表這種活動可能導致 `Client:ClientWrite` 事件增加。

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 將用戶端放在與叢集相同的可用區域和 VPC 子網路中
- 使用最新一代的執行個體
- 減少傳送至用戶端的資料量
- 擴展用戶端

將用戶端放在與叢集相同的可用區域和 VPC 子網路中

若要縮短網路延遲並增加網路輸送量，請將用戶端放在與 Aurora PostgreSQL 資料庫叢集相同的可用區域和 Virtual Private Cloud (VPC) 子網路中。

使用最新一代的執行個體

在某些情況下，您使用的資料庫執行個體類別可能不支援 Jumbo Frame。如果您在 Amazon EC2 上執行應用程式，請考慮為用戶端使用最新一代的執行個體。此外，在用戶端作業系統上設定最大傳輸單位 (MTU)。此技術可以減少網路往返次數並增加網路輸送量。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Jumbo Frame (9001 MTU)。

如需資料庫執行個體類別的相關資訊，請參閱 Amazon Aurora 資料庫執行個體類別。若要決定等同於 Amazon EC2 執行個體類型的資料庫執行個體類別，請在 Amazon EC2 執行個體類型名稱前面加上 `db.`。例如，`r5.8xlarge` Amazon EC2 執行個體等同於 `db.r5.8xlarge` 資料庫執行個體類別。

減少傳送至用戶端的資料量

可能的話，請調整應用程式，以減少 Aurora PostgreSQL 資料庫叢集傳送至用戶端的資料量。這樣調整可以減輕用戶端的 CPU 和網路爭用情形。

擴展用戶端

使用 Amazon CloudWatch 或其他主機指標，判斷用戶端目前是否受限於 CPU、網路頻寬，或同時受限於這兩者。如果用戶端受到限制，請相應地擴展用戶端。

CPU

此事件表示執行緒活躍於 CPU 中或正在等待 CPU。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因
- 動作

支援的引擎版本

此等待事件資訊與 Aurora PostgreSQL 9.6 版及更新版本有關。

Context

中央處理單元 (CPU) 是執行指令的電腦元件。例如，CPU 指令執行算術運算，並在記憶體中交換資料。如果查詢增加更多指令來透過資料庫引擎執行，則查詢會執行更久。CPU 排程將 CPU 時間撥給程序。排程由作業系統的核心協調。

主題

- 如何得知發生這種等待
- DBLoadCPU 指標
- os.cpuUtilization 指標
- CPU 排程的可能原因

如何得知發生這種等待

此 CPU 等待事件表示後端程序活躍於 CPU 中或正在等待 CPU。當查詢顯示下列資訊時，就表示發生此事件：

- pg_stat_activity.state 欄具有值 active。
- pg_stat_activity 中的 wait_event_type 和 wait_event 欄皆為 null。

若要查看正在使用或等待 CPU 的後端程序，請執行下列查詢。

```
SELECT *
FROM   pg_stat_activity
WHERE  state = 'active'
AND    wait_event_type IS NULL
AND    wait_event IS NULL;
```

DBLoadCPU 指標

CPU 的績效詳情指標為 DBLoadCPU。DBLoadCPU 的值與 Amazon CloudWatch 指標 CPUUtilization 的值可能不同。後者是從 HyperVisor 為資料庫執行個體收集的指標。

os.cpuUtilization 指標

績效詳情作業系統指標提供 CPU 使用率的詳細資訊。例如，您可以顯示下列指標：

- os.cpuUtilization.nice.avg
- os.cpuUtilization.total.avg
- os.cpuUtilization.wait.avg
- os.cpuUtilization.idle.avg

績效詳情以 os.cpuUtilization.nice.avg 報告資料庫引擎的 CPU 使用率。

CPU 排程的可能原因

就作業系統而言，CPU 不執行閒置執行緒時，就處於作用中。CPU 執行運算時就處於作用中，但等待記憶體輸入/輸出時也是處於作用中。在典型資料庫工作負載中，這種輸入/輸出佔多數。

滿足下列條件時，程序可能等待排定使用 CPU：

- CloudWatch `CPUUtilization` 指標接近 100%。
- 平均負載大於 vCPU 數目，表示負載過重。您可以在績效詳情的作業系統指標區段中找到 `loadAverageMinute` 指標。

等待變多的可能原因

CPU 等待事件比平時更常發生時，可能表示有效能問題，典型原因包括下列各項。

主題

- 突然激增的可能原因
- 長期頻繁的可能原因
- 罕見情況

突然激增的可能原因

突然激增最可能的原因如下：

- 應用程式對資料庫同時開啟太多連線。這種情況稱為「連線風暴」。
- 應用程式工作負載出現下列任何變化：
  - 新查詢
  - 資料集變大
  - 維護或建立索引
  - 新函數
  - 新運算子
  - 平行執行查詢增加
- 查詢執行計劃已變更。在某些情況下，變更可能造成緩衝區增加。例如，查詢原本使用索引，但現在使用循序掃描。在此情況下，查詢需要更多 CPU 才能完成相同的目標。

長期頻繁的可能原因

事件長期復發最可能的原因：

- CPU 上同時執行太多後端程序。這些程序可能是平行工作者。
- 查詢表現欠佳，因為需要大量緩衝區。

罕見情況

如果可能原因都不是真正原因，可能表示發生下列情況：

- CPU 正在換入換出程序。
- CPU 環境切換次數變多。
- Aurora PostgreSQL 程式碼遺漏等待事件。

動作

如果 CPU 等待事件在資料庫活動中佔多數，則不見得表示有效能問題。效能降低時才需要回應此事件。

主題

- 調查資料庫是否造成 CPU 使用率上升
- 判斷連線數目是否增加
- 回應工作負載變更

調查資料庫是否造成 CPU 使用率上升

在績效詳情中檢查 `os.cpuUtilization.nice.avg` 指標。如果此值遠低於 CPU 使用率，表示佔用 CPU 以非資料庫的程序為主。

判斷連線數目是否增加

在 Amazon CloudWatch 中檢查 `DatabaseConnections` 指標。您的動作取決於在 CPU 等待事件變多期間，連線數目是增加還是減少。

連線增加

如果連線數目上升，請將耗用 CPU 的後端程序數目與 vCPU 數目做比較。可能的情況如下：

- 耗用 CPU 的後端程序數目小於 vCPU 數目。

  在此情況下，連線數目不是問題。但您仍然可以嘗試降低 CPU 使用率。
- 耗用 CPU 的後端程序數目大於 vCPU 數目。

  在此情況下，請考慮下列選項：

- 將連線至資料庫的後端程序數目減少。例如，實作連線集區解決方案，例如 RDS Proxy。如需詳細資訊，請參閱 Amazon RDS Proxy for Aurora。
- 加大執行個體以取得更多 vCPU。
- 如適用，請將一些唯讀工作負載重新導向至讀取器節點

連線未增加

在績效詳情中檢查 `blks_hit` 指標。尋找 `blks_hit` 增加與 CPU 使用率之間的關聯性。可能的情況如下：

- CPU 使用率與 `blks_hit` 有關。

  在此情況下，請找出與 CPU 使用率最有關的 SQL 陳述式，並尋找計劃變更。您可以使用下列任一技巧：

  - 手動解釋計劃，並與預期的執行計劃做比較。
  - 查明每秒區塊命中數和每秒本機區塊命中數是否增加。在績效詳情儀表板的 Top SQL (常用 SQL) 區段中，選擇 Preferences (偏好設定)。
- CPU 使用率與 `blks_hit` 無關。

  在此情況下，請判斷是否發生下列任何情形：

  - 應用程式與資料庫之間快速連線和斷線。

    請開啟 `log_connections` 和 `log_disconnections`，然後分析 PostgreSQL 日誌，以診斷此行為。考慮使用 `pgbadger` 日誌分析器。如需詳細資訊，請參閱 https://github.com/darold/pgbadger。
  - 作業系統超載。

    在此情況下，績效詳情會指出後端程序耗用 CPU 比平常更久。在績效詳情 `os.cpuUtilization` 指標或 CloudWatch `CPUUtilization` 指標中查證。如果作業系統超載，請檢查增強型監控指標以進一步診斷。具體來說，請檢查程序清單及每個程序耗用的 CPU 百分比。
  - 常用 SQL 陳述式耗用太多 CPU。

    檢查與 CPU 使用率有關的陳述式，了解是否可以少用 CPU。執行 `EXPLAIN` 命令，專注於影響最大的計劃節點上。考慮使用 PostgreSQL execution plan visualizer。若要試用此工具，請參閱 http://explain.dalibo.com/。

回應工作負載變更

如果工作負載已變更，請尋找下列類型的變更：

新查詢

檢查是否需要新的查詢。如果是，請確定其執行計劃和每秒執行次數合理。

資料集變大

決定是否應該分割 (如果尚未分割)。此策略可以減少查詢需要擷取的分頁數。

維護或建立索引

檢查是否需要排定維護。最佳實務是將維護活動排定在尖峰活動之外。

新函數

檢查這些函數在測試期間是否正常執行。具體來說，請檢查每秒執行次數是否合理。

新運算子

檢查在測試期間是否正常執行。

平行執行查詢增加

判斷是否發生下列任何情況：

- 涉及的關聯或索引突然變大，明顯不同於 `min_parallel_table_scan_size` 或 `min_parallel_index_scan_size`。
- 最近變更 `parallel_setup_cost` 或 `parallel_tuple_cost`。
- 最近變更 `max_parallel_workers` 或 `max_parallel_workers_per_gather`。

IO:BufFileRead 和 IO:BufFileWrite

`IO:BufFileRead` 和 `IO:BufFileWrite` 事件表示 Aurora PostgreSQL 建立暫存檔。如果操作需要的記憶體超過工作記憶體參數目前的定義，則會將暫存資料寫入永久性儲存。此操作有時稱為溢出到磁碟。如需暫存檔案及其使用情況的詳細資訊，請參閱 使用 PostgreSQL 管理暫存檔案。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因

- 動作

支援的引擎版本

所有版本的 Aurora PostgreSQL 都支援此等待事件資訊。

Context

IO:BufFileRead 和 IO:BufFileWrite 與工作記憶體區域和維護工作記憶體區域有關。如需這些本機記憶體區域的詳細資訊，請參閱工作記憶體區域和維護工作記憶體區域。

work_mem 的預設值為 4 MB。如果一個工作階段平行執行操作，則負責平行處理的每個工作者會使用 4 MB 的記憶體。因此，請小心設定 work_mem。如果將此值設得太大，則執行許多工作階段的資料庫可能會耗用太多記憶體。如果將此值設得太小，Aurora PostgreSQL 會在本機儲存中建立暫存檔。這些暫存檔的磁碟輸入/輸出可能造成效能降低。

如果您看到下列事件序列，表示資料庫可能正在產生暫存檔：

1. 可用性突然遽降
2. 可用空間迅速復原

您也可能看到「鏈鋸」模式。此模式可能表示資料庫不斷建立小檔案。

等待變多的可能原因

一般而言，這些等待事件起因於操作耗用超過 work_mem 或 maintenance_work_mem參數所配置的記憶體。為了補償，操作寫入暫存檔。IO:BufFileRead 和 IO:BufFileWrite 事件的常見原因包括：

查詢需要比工作記憶體區域中更多的記憶體

具有下列特性的查詢使用工作記憶體區域：

- 雜湊聯結。
- ORDER BY 子句
- GROUP BY 子句
- DISTINCT
- 範圍函數
- CREATE TABLE AS SELECT

- 具體化檢視表重新整理

陳述式需要比維護工作記憶體區域中更多的記憶體

下列陳述式使用維護工作記憶體區域：

- CREATE INDEX
- CLUSTER

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 識別問題
- 檢查聯結查詢
- 檢查 ORDER BY 和 GROUP BY 查詢
- 避免使用 DISTINCT 操作
- 考慮使用視窗函數代替 GROUP BY 函數
- 調查具體化檢視表和 CTAS 陳述式
- 建立索引時使用 pg_repack
- 叢集化資料表時提高 maintenance_work_mem
- 調整記憶體以防止 IO:BufFileRead 和 IO:BufFileWrite

識別問題

您可以直接在績效詳情中檢視暫存檔案用量。如需詳細資訊，請參閱使用 Performance Insights 來檢視暫存檔使用情況。停用績效詳情時，您可能會注意到 IO:BufFileRead和 IO:BufFileWrite操作增加。若要疑難排解，請執行下列動作：

1. 在 Amazon CloudWatch 中檢查 FreeLocalStorage 指標。
2. 尋找鏈鋸模式，即一連串鋸齒尖。

鏈鋸模式表示快速耗用和釋放儲存，通常與暫存檔有關。如果您看到此模式，請開啟績效詳情。使用績效詳情時，您可以識別等待事件何時發生及其相關聯的查詢。您的解決方案取決於導致事件的特定查詢。

或設定 log_temp_files 參數。此參數會記錄產生超過閾值 KB 暫存檔的所有查詢。如果值為 0，Aurora PostgreSQL 會記錄所有暫存檔。如果值為 1024，Aurora PostgreSQL 會記錄產生大於 1 MB 暫存檔的所有查詢。如需 log_temp_files 的詳細資訊，請參閱 PostgreSQL 文件中的錯誤報告和日誌記錄。

檢查聯結查詢

您的應用程式可能使用聯結。例如，下列查詢會聯結四個資料表。

```
SELECT *
       FROM order
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = order.customer_id)
 INNER JOIN customer_address
       ON (customer_address.customer_id = customer.id AND
           order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

暫存檔使用量激增的可能原因在於查詢本身有問題。例如，不標準的子句可能無法正確篩選聯結。請看下列範例中的第二個內部聯結。

```
SELECT *
       FROM order
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = customer.id)
 INNER JOIN customer_address
       ON (customer_address.customer_id = customer.id AND
           order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

上述查詢誤將 customer.id 聯結至 customer.id，導致在每個客戶與每筆訂單之間產生笛卡爾乘積。這種意外聯結會產生大型暫存檔。根據資料表的大小，笛卡爾查詢甚至可能導致填滿儲存。有下列情況時，表示應用程式可能有笛卡爾聯結：

- 您看到儲存可用性大幅遽降，接著迅速復原。
- 未建立索引。
- 未發出 CREATE TABLE FROM SELECT 陳述式。

- 未重新整理具體化檢視表。

若要檢查是否以適當索引鍵聯結資料表，請檢驗查詢和物件關聯式映射指令。切記，不一定會呼叫應用程式的某些查詢，有些查詢是動態產生。

檢查 ORDER BY 和 GROUP BY 查詢

在某些情況下，ORDER BY 子句可能導致產生過多暫存檔。請考量下列準則：

- 只將需要排序的資料欄放入 ORDER BY 子句中。如果查詢傳回數千個資料列，並在 ORDER BY 子句中指定許多資料欄，此準則尤其重要。
- 當 ORDER BY 子句比對的資料欄有相同遞增或遞減順序時，請考慮建立索引以加速執行。最好是局部索引，因為較小。讀取和周遊較小的索引比較快。
- 如果您為可接受空值的資料欄建立索引，請決定要將空值存放在索引結尾還是開頭。

  可能的話，請篩選結果集，以減少需要排序的資料列數。如果您使用 WITH 子句陳述式或子查詢，請記住，內部查詢會產生結果集並傳給外部查詢。查詢篩選掉越多資料列，查詢就越不需要排序。
- 如果不需要取得完整結果集，請使用 LIMIT 子句。例如，如果您只想要前五個資料列，則在查詢中使用 LIMIT 子句就不會一直產生結果。如此，查詢只需要較少的記憶體和暫存檔。

使用 GROUP BY 子句的查詢可能也需要暫存檔。GROUP BY 查詢使用如下函數來彙總值：

- COUNT
- AVG
- MIN
- MAX
- SUM
- STDDEV

若要調校 GROUP BY 查詢，請遵循 ORDER BY 查詢的建議。

避免使用 DISTINCT 操作

可能的話，請避免使用 DISTINCT 操作來移除重複的資料列。查詢傳回不必要和重複的資料列越多，DISTINCT 操作的成本越高。可能的話，請在 WHERE 子句中增加篩選條件，即使不同的資料表使用相同的篩選器也無妨。篩選查詢並正確聯結可改善效能和減少使用資源。還可防止不正確的報告和結果。

如果需要對同一個資料表的多個資料列使用 DISTINCT，請考慮建立複合索引。將多個資料列組合成一個索引，可縮短相異資料列的評估時間。此外，如果您使用 Amazon Aurora PostgreSQL 第 10 版或更新版本，則可以使用 CREATE STATISTICS 命令，將多個資料欄之間的統計數字相互關聯。

考慮使用視窗函數代替 GROUP BY 函數

使用 GROUP BY 時，您變更結果集，然後擷取彙總結果。使用視窗函數時，您彙總資料而不變更結果集。視窗函數使用 OVER 子句來跨查詢所定義的集執行計算，使資料列彼此相互關聯。您在視窗函數中可以使用所有 GROUP BY 函數，但也可使用如下函數：

- RANK
- ARRAY_AGG
- ROW_NUMBER
- LAG
- LEAD

若要盡量減少視窗函數產生的暫存檔，當需要兩個相異彙總時，請移除相同結果集的重複部分。請看下列查詢。

```
SELECT sum(salary) OVER (PARTITION BY dept ORDER BY salary DESC) as sum_salary
     , avg(salary) OVER (PARTITION BY dept ORDER BY salary ASC) as avg_salary
  FROM empsalary;
```

您可以使用 WINDOW 子句重寫查詢，如下所示。

```
SELECT sum(salary) OVER w as sum_salary
         , avg(salary) OVER w as_avg_salary
    FROM empsalary
  WINDOW w AS (PARTITION BY dept ORDER BY salary DESC);
```

根據預設，Aurora PostgreSQL 執行規劃工具會合併相似的節點，以免重複操作。不過，使用視窗區塊的明確宣告可以更輕鬆維護查詢。防止重複也可以改善效能。

調查具體化檢視表和 CTAS 陳述式

具體化檢視表重新整理時會執行查詢。此查詢可以包含 GROUP BY、ORDER BY 或 DISTINCT 等操作。在重新整理期間，您可能看到大量暫存檔及等待事件 IO:BufFileWrite 和

IO:BufFileRead。同樣地，當您根據 SELECT 陳述式建立資料表時，CREATE TABLE 陳述式會執行查詢。若要減少所需的暫存檔，請最佳化查詢。

建立索引時使用 pg_repack

當您建立索引時，引擎會排序結果集。隨著資料表變大，以及索引資料欄的值變得更多樣化，暫存檔需要更多空間。在大多數情況下，除非修改維護工作記憶體區域，否則無法阻止為大型資料表建立暫存檔。如需詳細資訊，請參閱維護工作記憶體區域。

重新建立大型索引時，可能的解決方法是使用 pg_repack 工具。如需詳細資訊，請參閱 pg_repack 文件中的以最少鎖定重組 PostgreSQL 資料庫中的資料表。

叢集化資料表時提高 maintenance_work_mem

CLUSTER 命令根據 index_name 指定的現有索引，以叢集化 table_name 指定的資料表。Aurora PostgreSQL 實際上會重新建立資料表，以符合特定索引的順序。

在磁帶儲存盛行的年代，因為儲存輸送量有限，叢集很普遍。如今 SSD 型儲存很普遍，較不流行叢集。不過，如果將資料表叢集化，還是可稍微提高效能，視資料表大小、索引、查詢等而定。

如果您執行 CLUSTER 命令，然後看到等待事件 IO:BufFileWrite 和 IO:BufFileRead,請調校 maintenance_work_mem。請將記憶體調到很大。較大的值表示引擎可以使用更多記憶體執行叢集操作。

調整記憶體以防止 IO:BufFileRead 和 IO:BufFileWrite

在某些情況下，您需要調整記憶體。目標是平衡下列需求：

- work_mem 值 (請參閱工作記憶體區域)
- 折除 shared_buffers 值之後剩餘的記憶體 (請參閱緩衝集區)
- 已開啟和使用中的連線數上限，受限於 max_connections

增加工作記憶體區域的大小

在某些情況下，您只能選擇增加工作階段所使用的記憶體。如果查詢撰寫無誤，且使用正確的索引鍵來聯結，請考慮提高 work_mem 值。如需詳細資訊，請參閱工作記憶體區域。

若要了解查詢產生多少暫存檔，請將 log_temp_files 設定為 0。如果將 work_mem 值提高到日誌中指出的最大值，就可以防止查詢產生暫存檔。然而，work_mem 會針對每個連線或平行工作者，設

定每個計劃節點的最大值。如果資料庫有 5,000 個連線，且每個連線各使用 256 MiB 的記憶體，則引擎需要 1.2 TiB 的 RAM。因此，執行個體可能記憶體不足。

為共用緩衝集區保留足夠記憶體

資料庫不只使用工作記憶體區域，還使用共用緩衝集區之類的記憶體區域。提高 work_mem 之前，請考慮這些額外記憶體區域的需求。如需緩衝集區的詳細資訊，請參閱緩衝集區。

例如，假設您的 Aurora PostgreSQL 執行個體類別為 db.r5.2xlarge。此類別有 64 GiB 的記憶體。預設會保留 75% 的記憶體給共用緩衝集區。減去配置給共用記憶體區域的數量後，剩下 16,384 MB。請勿將剩餘的記憶體全部配置給工作記憶體區域，因為作業系統和引擎也需要記憶體。

可配置給 work_mem 的記憶體取決於執行個體類別。使用越大的執行個體類別，可用的記憶體越多。不過，在上述範例中，最多只能使用 16 GiB。否則，當記憶體不足時，就無法使用執行個體。為了讓執行個體從無法使用狀態中復原，Aurora PostgreSQL 自動化服務會自動重新啟動。

管理連線數目

假設您的資料庫執行個體有 5,000 個同時連線。每個連線至少使用 4 MiB 的 work_mem。連線耗用大量記憶體可能導致效能降低。因應之道如下：

- 提升為更大的執行個體類別。
- 使用連線代理或集區來減少同時的資料庫連線數。

關於代理，根據您的應用程式而定，請考慮 Amazon RDS Proxy、pgBuncer 或連線集區。此解決方案可減輕 CPU 負載。面臨所有連線都需要工作記憶體區域時，也能降低風險。在只有少數資料庫連線時，您可以提高 work_mem 的值。如此就能減少 IO:BufFileRead 和 IO:BufFileWrite 等待事件。等待工作記憶體區域的查詢也會大幅加速。

IO:DataFileRead

IO:DataFileRead 事件表示因為所需的分頁不在共用記憶體中，連線等待後端程序從儲存讀取該分頁。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因

- 動作

支援的引擎版本

所有版本的 Aurora PostgreSQL 都支援此等待事件資訊。

Context

所有查詢和資料操作 (DML) 作業會存取緩衝集區中的分頁。引起讀取的陳述式包括 SELECT、UPDATE 及 DELETE。例如，UPDATE 可以從資料表或索引讀取分頁。如果所請求或更新的頁面不在共用緩衝集區中，則此讀取可能引起 IO:DataFileRead 事件。

共用緩衝集區有限，可能填滿。在此情況下，請求的分頁不在記憶體中，迫使資料庫從磁碟讀取區塊。如果 IO:DataFileRead 事件經常發生，可能表示共用緩衝集區太小，不足以應付工作負載。這是嚴重問題，因為 SELECT 查詢讀取大量資料列，塞不進緩衝集區。如需緩衝集區的詳細資訊，請參閱緩衝集區。

等待變多的可能原因

IO:DataFileRead 的常見原因包括：

連線尖峰

您可能發現多個連線產生一樣多的 IO:DataFileRead 等待事件。在此情況下，IO:DataFileRead 事件可能出現尖峰 (突然大幅增加)。

SELECT 和 DML 陳述式執行循序掃描

您的應用程式可能執行新的操作。或者，現有的操作可能因為新的執行計劃而變更。在這種情況下，請尋找 seq_scan 值較大的資料表 (特別是大型資料表)。查詢 pg_stat_user_tables 來尋找。若要追蹤哪些查詢產生較多讀取操作，請使用延伸 pg_stat_statements。

大型資料集的 CTAS 和 CREATE INDEX

CTAS 代表 CREATE TABLE AS SELECT 陳述式。如果您以大型資料集為來源執行 CTAS，或在大型資料表上建立索引，可能會發生 IO:DataFileRead 事件。建立索引時，資料庫可能需要使用循序掃描來讀取整個物件。當分頁不在記憶體中時，CTAS 會產生 IO:DataFile 讀取。

多個清理工作者同時執行

清理工作者是手動或自動觸發。建議採取積極清理策略。不過，當資料表更新或刪除許多資料列時，IO:DataFileRead 等待會變多。回收空間後，花在 IO:DataFileRead 的清理時間就會減少。

擷取大量資料

當應用程式擷取大量資料時，ANALYZE 操作可能更頻繁發生。ANALYZE 程序可以由自動資料清理啟動器觸發，或手動叫用。

ANALYZE 操作讀取資料表的子集。將 30 乘以 default_statistics_target 值，即可算出必須掃描的分頁數。如需詳細資訊，請參閱 PostgreSQL 文件。default_statistics_target 參數接受 1 到 10,000 之間的值，預設值為 100。

資源耗盡

如果耗用執行個體網路頻寬或 CPU，IO:DataFileRead 事件可能更頻繁發生。

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 對產生等待的查詢檢查述詞篩選條件
- 將維護操作的影響降至最低
- 因應大量連線

對產生等待的查詢檢查述詞篩選條件

假設您發現特定的查詢正在產生 IO:DataFileRead 等待事件。請利用下列技巧來識別：

- 績效詳情
- 目錄檢視表，例如延伸 pg_stat_statements 提供的檢視表
- 目錄檢視表 pg_stat_all_tables (如果定期指出實體讀取變多)
- pg_statio_all_tables 檢視表 (如果指出 _read 計數器上升)

建議您判斷這些查詢的述詞中 (WHERE 子句) 使用哪些篩選條件。請遵守下列準則：

- 執行 EXPLAIN 命令。在輸出中，識別使用的掃描類型。循序掃描並不一定代表有問題。使用循序掃描的查詢，當然比使用篩選條件的查詢產生更多 IO:DataFileRead 事件。

查明列在 WHERE 子句中的資料欄是否已編成索引。如果不是，請考慮為此資料欄建立索引。這種方法可避免循序掃描，並減少 IO:DataFileRead 事件。如果查詢有嚴格的篩選條件，但仍產生循序掃描，請評估使用的索引是否適當。

- 查明查詢是否存取非常大的資料表。在某些情況下，將資料表分割可以改善效能，讓查詢只讀取必要的分割區。
- 檢查聯結操作中的基數 (總資料列數)。請注意您在 WHERE 子句的過濾條件中傳入的值有多嚴格。可能的話，請調整查詢，以減少在計劃的每個步驟中傳入的資料列數目。

將維護操作的影響降至最低

維護操作很重要，例如 VACUUM 和 ANALYZE。建議不要因為發現這些維護操作相關的 IO:DataFileRead 等待事件而關閉維護。下列方法可以將這些操作的影響降至最低：

- 在離峰時段手動執行維護操作。這項技巧可防止資料庫達到自動化操作的閾值。
- 如果資料表非常大，請考慮分割資料表。這項技巧可減少維護操作的額外負荷。資料庫只會存取需要維護的分割區。
- 擷取大量資料時，請考慮停用自動分析功能。

下列公式成立時會自動對資料表觸發自動資料清理功能。

```
pg_stat_user_tables.n_dead_tup > (pg_class.reltuples x autovacuum_vacuum_scale_factor)
 + autovacuum_vacuum_threshold
```

檢視表 pg_stat_user_tables 和目錄 pg_class 有多個資料列。一個資料列可以對應於資料表中的一個資料列。這個公式假設 reltuples 專用於特定資料表。通常是為整個執行個體來整體設定參數 autovacuum_vacuum_scale_factor (預設為 0.20) 和 autovacuum_vacuum_threshold (預設為 50 個元組)。但您可以針對特定資料表設定不同的值。

主題

- 尋找耗用不必要空間的資料表
- 尋找耗用不必要空間的索引
- 尋找適合自動資料清理的資料表

尋找耗用不必要空間的資料表

若要尋找耗用超過所需空間的資料表，請執行下列查詢。當此查詢是由沒有 rds_superuser 角色的資料庫使用者角色執行時，它只會傳回使用者角色有權讀取之資料表的相關資訊。PostgreSQL 第 12 版及更新版本支援此查詢。

```
WITH report AS (
   SELECT   schemaname
            ,tblname
            ,n_dead_tup
            ,n_live_tup
            ,block_size*tblpages AS real_size
            ,(tblpages-est_tblpages)*block_size AS extra_size
            ,CASE WHEN tblpages - est_tblpages > 0
               THEN 100 * (tblpages - est_tblpages)/tblpages::float
               ELSE 0
             END AS extra_ratio, fillfactor, (tblpages-est_tblpages_ff)*block_size AS
 bloat_size
            ,CASE WHEN tblpages - est_tblpages_ff > 0
               THEN 100 * (tblpages - est_tblpages_ff)/tblpages::float
               ELSE 0
             END AS bloat_ratio
            ,is_na
    FROM (
           SELECT  ceil( reltuples / ( (block_size-page_hdr)/tpl_size ) ) +
 ceil( toasttuples / 4 ) AS est_tblpages
                  ,ceil( reltuples / ( (block_size-page_hdr)*fillfactor/
(tpl_size*100) ) ) + ceil( toasttuples / 4 ) AS est_tblpages_ff
                  ,tblpages
                  ,fillfactor
                  ,block_size
                  ,tblid
                  ,schemaname
                  ,tblname
                  ,n_dead_tup
                  ,n_live_tup
                  ,heappages
                  ,toastpages
                  ,is_na
             FROM (
                    SELECT ( 4 + tpl_hdr_size + tpl_data_size + (2*ma)
                               - CASE WHEN tpl_hdr_size%ma = 0 THEN ma ELSE
 tpl_hdr_size%ma END
```

```
                                - CASE WHEN ceil(tpl_data_size)::int%ma = 0 THEN ma ELSE
 ceil(tpl_data_size)::int%ma END
                            ) AS tpl_size
                            ,block_size - page_hdr AS size_per_block
                            ,(heappages + toastpages) AS tblpages
                            ,heappages
                            ,toastpages
                            ,reltuples
                            ,toasttuples
                            ,block_size
                            ,page_hdr
                            ,tblid
                            ,schemaname
                            ,tblname
                            ,fillfactor
                            ,is_na
                            ,n_dead_tup
                            ,n_live_tup
                           FROM (
                                SELECT  tbl.oid                       AS tblid
                                       ,ns.nspname                    AS schemaname
                                       ,tbl.relname                   AS tblname
                                       ,tbl.reltuples                 AS reltuples
                                       ,tbl.relpages                  AS heappages
                                       ,coalesce(toast.relpages, 0)   AS toastpages
                                       ,coalesce(toast.reltuples, 0)  AS toasttuples
                                       ,psat.n_dead_tup               AS n_dead_tup
                                       ,psat.n_live_tup               AS n_live_tup
                                       ,24                            AS page_hdr
                                       ,current_setting('block_size')::numeric AS
 block_size

 ,coalesce(substring( array_to_string(tbl.reloptions, ' ') FROM
 'fillfactor=([0-9]+)')::smallint, 100) AS fillfactor
                                       ,CASE WHEN version()~'mingw32' OR version()~'64-
bit|x86_64|ppc64|ia64|amd64' THEN 8 ELSE 4 END        AS ma
                                       ,23 + CASE WHEN MAX(coalesce(null_frac,0)) > 0
 THEN ( 7 + count(*) ) / 8 ELSE 0::int END              AS tpl_hdr_size
                                       ,sum( (1-coalesce(s.null_frac, 0)) *
 coalesce(s.avg_width, 1024) )                                  AS tpl_data_size
                                       ,bool_or(att.atttypid =
 'pg_catalog.name'::regtype) OR count(att.attname) <> count(s.attname)        AS is_na
                                  FROM  pg_attribute       AS att
```

```
                                    JOIN  pg_class           AS tbl    ON (att.attrelid =
 tbl.oid)
                                    JOIN  pg_stat_all_tables AS psat   ON (tbl.oid =
 psat.relid)
                                    JOIN  pg_namespace       AS ns     ON (ns.oid =
 tbl.relnamespace)
                               LEFT JOIN  pg_stats           AS s      ON
 (s.schemaname=ns.nspname AND s.tablename = tbl.relname AND s.inherited=false AND
 s.attname=att.attname)
                               LEFT JOIN  pg_class           AS toast  ON
 (tbl.reltoastrelid = toast.oid)
                                   WHERE  att.attnum > 0
                                     AND  NOT att.attisdropped
                                     AND  tbl.relkind = 'r'
                                GROUP BY  tbl.oid, ns.nspname, tbl.relname,
 tbl.reltuples, tbl.relpages, toastpages, toasttuples, fillfactor, block_size, ma,
 n_dead_tup, n_live_tup
                                ORDER BY  schemaname, tblname
                              ) AS s
                  ) AS s2
       ) AS s3
 ORDER BY bloat_size DESC
)
  SELECT *
    FROM report
   WHERE bloat_ratio != 0
 -- AND schemaname = 'public'
 -- AND tblname = 'pgbench_accounts'
;

-- WHERE NOT is_na
--   AND tblpages*((pst).free_percent + (pst).dead_tuple_percent)::float4/100 >= 1
```

您可以在應用程式中檢查資料表和索引膨脹。如需詳細資訊，請參閱診斷資料表和索引膨脹。

尋找耗用不必要空間的索引

若要尋找耗用不必要空間的索引，請執行下列查詢。

```
-- WARNING: run with a nonsuperuser role, the query inspects
-- only indexes on tables you have permissions to read.
-- WARNING: rows with is_na = 't' are known to have bad statistics ("name" type is not
 supported).
-- This query is compatible with PostgreSQL 8.2 and later.
```

```
SELECT current_database(), nspname AS schemaname, tblname, idxname,
 bs*(relpages)::bigint AS real_size,
  bs*(relpages-est_pages)::bigint AS extra_size,
  100 * (relpages-est_pages)::float / relpages AS extra_ratio,
  fillfactor, bs*(relpages-est_pages_ff) AS bloat_size,
  100 * (relpages-est_pages_ff)::float / relpages AS bloat_ratio,
  is_na
  -- , 100-(sub.pst).avg_leaf_density, est_pages, index_tuple_hdr_bm,
  -- maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, sub.reltuples, sub.relpages
  -- (DEBUG INFO)
FROM (
  SELECT coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)/(4+nulldatahdrwidth)::float)), 0
       -- ItemIdData size + computed avg size of a tuple (nulldatahdrwidth)
    ) AS est_pages,
    coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)*fillfactor/
(100*(4+nulldatahdrwidth)::float))), 0
    ) AS est_pages_ff,
    bs, nspname, table_oid, tblname, idxname, relpages, fillfactor, is_na
    -- , stattuple.pgstatindex(quote_ident(nspname)||'.'||quote_ident(idxname)) AS
 pst,
    -- index_tuple_hdr_bm, maxalign, pagehdr, nulldatawidth, nulldatahdrwidth,
 reltuples
    -- (DEBUG INFO)
  FROM (
    SELECT maxalign, bs, nspname, tblname, idxname, reltuples, relpages, relam,
 table_oid, fillfactor,
      ( index_tuple_hdr_bm +
          maxalign - CASE -- Add padding to the index tuple header to align on MAXALIGN
            WHEN index_tuple_hdr_bm%maxalign = 0 THEN maxalign
            ELSE index_tuple_hdr_bm%maxalign
          END
        + nulldatawidth + maxalign - CASE -- Add padding to the data to align on
 MAXALIGN
            WHEN nulldatawidth = 0 THEN 0
            WHEN nulldatawidth::integer%maxalign = 0 THEN maxalign
            ELSE nulldatawidth::integer%maxalign
          END
      )::numeric AS nulldatahdrwidth, pagehdr, pageopqdata, is_na
      -- , index_tuple_hdr_bm, nulldatawidth -- (DEBUG INFO)
    FROM (
      SELECT
```

```
       i.nspname, i.tblname, i.idxname, i.reltuples, i.relpages, i.relam, a.attrelid
AS table_oid,
       current_setting('block_size')::numeric AS bs, fillfactor,
       CASE -- MAXALIGN: 4 on 32bits, 8 on 64bits (and mingw32 ?)
         WHEN version() ~ 'mingw32' OR version() ~ '64-bit|x86_64|ppc64|ia64|amd64'
THEN 8
         ELSE 4
       END AS maxalign,
       /* per page header, fixed size: 20 for 7.X, 24 for others */
       24 AS pagehdr,
       /* per page btree opaque data */
       16 AS pageopqdata,
       /* per tuple header: add IndexAttributeBitMapData if some cols are null-able */
       CASE WHEN max(coalesce(s.null_frac,0)) = 0
         THEN 2 -- IndexTupleData size
         ELSE 2 + (( 32 + 8 - 1 ) / 8)
         -- IndexTupleData size + IndexAttributeBitMapData size ( max num filed per
index + 8 - 1 /8)
       END AS index_tuple_hdr_bm,
       /* data len: we remove null values save space using it fractionnal part from
stats */
       sum( (1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 1024)) AS
nulldatawidth,
       max( CASE WHEN a.atttypid = 'pg_catalog.name'::regtype THEN 1 ELSE 0 END ) > 0
AS is_na
     FROM pg_attribute AS a
       JOIN (
         SELECT nspname, tbl.relname AS tblname, idx.relname AS idxname,
           idx.reltuples, idx.relpages, idx.relam,
           indrelid, indexrelid, indkey::smallint[] AS attnum,
           coalesce(substring(
             array_to_string(idx.reloptions, ' ')
              from 'fillfactor=([0-9]+)')::smallint, 90) AS fillfactor
         FROM pg_index
           JOIN pg_class idx ON idx.oid=pg_index.indexrelid
           JOIN pg_class tbl ON tbl.oid=pg_index.indrelid
           JOIN pg_namespace ON pg_namespace.oid = idx.relnamespace
         WHERE pg_index.indisvalid AND tbl.relkind = 'r' AND idx.relpages > 0
       ) AS i ON a.attrelid = i.indexrelid
       JOIN pg_stats AS s ON s.schemaname = i.nspname
         AND ((s.tablename = i.tblname AND s.attname =
pg_catalog.pg_get_indexdef(a.attrelid, a.attnum, TRUE))
         -- stats from tbl
         OR  (s.tablename = i.idxname AND s.attname = a.attname))
```

```
          -- stats from functionnal cols
        JOIN pg_type AS t ON a.atttypid = t.oid
      WHERE a.attnum > 0
      GROUP BY 1, 2, 3, 4, 5, 6, 7, 8, 9
    ) AS s1
  ) AS s2
    JOIN pg_am am ON s2.relam = am.oid WHERE am.amname = 'btree'
) AS sub
-- WHERE NOT is_na
ORDER BY 2,3,4;
```

尋找適合自動資料清理的資料表

若要尋找適合自動資料清理的資料表，請執行下列查詢。

```
--This query shows tables that need vacuuming and are eligible candidates.
--The following query lists all tables that are due to be processed by autovacuum.
-- During normal operation, this query should return very little.
WITH  vbt AS (SELECT setting AS autovacuum_vacuum_threshold
              FROM pg_settings WHERE name = 'autovacuum_vacuum_threshold')
    , vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor
              FROM pg_settings WHERE name = 'autovacuum_vacuum_scale_factor')
    , fma AS (SELECT setting AS autovacuum_freeze_max_age
              FROM pg_settings WHERE name = 'autovacuum_freeze_max_age')
    , sto AS (SELECT opt_oid, split_part(setting, '=', 1) as param,
                split_part(setting, '=', 2) as value
              FROM (SELECT oid opt_oid, unnest(reloptions) setting FROM pg_class) opt)
SELECT
    '"'||ns.nspname||'"."'||c.relname||'"' as relation
    , pg_size_pretty(pg_table_size(c.oid)) as table_size
    , age(relfrozenxid) as xid_age
    , coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
 autovacuum_freeze_max_age
    , (coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
         coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) *
 c.reltuples)
         as autovacuum_vacuum_tuples
    , n_dead_tup as dead_tuples
FROM pg_class c
JOIN pg_namespace ns ON ns.oid = c.relnamespace
JOIN pg_stat_all_tables stat ON stat.relid = c.oid
JOIN vbt on (1=1)
JOIN vsf ON (1=1)
JOIN fma on (1=1)
```

```
LEFT JOIN sto cvbt ON cvbt.param = 'autovacuum_vacuum_threshold' AND c.oid =
 cvbt.opt_oid
LEFT JOIN sto cvsf ON cvsf.param = 'autovacuum_vacuum_scale_factor' AND c.oid =
 cvsf.opt_oid
LEFT JOIN sto cfma ON cfma.param = 'autovacuum_freeze_max_age' AND c.oid = cfma.opt_oid
WHERE c.relkind = 'r'
AND nspname <> 'pg_catalog'
AND (
    age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
    or
    coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
      coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples
 <= n_dead_tup
    -- or 1 = 1
)
ORDER BY age(relfrozenxid) DESC;
```

因應大量連線

監控 Amazon CloudWatch 時，您可能發現 `DatabaseConnections` 指標激增。此增加表示資料庫的連線數增加。建議採取下列作法：

- 限制應用程式可以對每個執行個體開啟的連線數。如果應用程式有內嵌連線集區功能，請設定合理的連線數目。請以執行個體中的 vCPU 可有效平行處理的數目為準。

  如果應用程式不使用連線集區功能，請考慮使用 Amazon RDS Proxy 或替代方案。這種作法可讓應用程式對負載平衡器開啟多個連線。因此，平衡器就能對資料庫開啟較少的連線。由於平行執行的連線較少，資料庫執行個體在核心中就能減少切換環境。查詢應該會進行得更快，使得等待事件變少。如需詳細資訊，請參閱Amazon RDS Proxy for Aurora。
- 盡可能在 Aurora PostgreSQL 中使用讀取器節點，在 RDS for PostgreSQL 中使用僅供讀取複本。當應用程式執行唯讀操作時，請將這些請求傳送至僅限讀取器端點。這項技巧可將應用程式請求分散到所有讀取器節點，減少寫入器節點的輸入/輸出壓力。
- 考慮擴充資料庫執行個體的規模。容量較大的執行個體類別提供更多記憶體，讓 Aurora PostgreSQL 有較大的共用緩衝集區可保留分頁。越大也讓資料庫執行個體有越多 vCPU 來處理連線。當寫入操作產生 `IO:DataFileRead` 等待事件時，較多 vCPU 會特別有用。

IO:XactSync

`IO:XactSync` 事件表示資料庫正在等待 Aurora 儲存子系統確認遞交一般交易，或是遞交或回復備妥交易。備妥交易屬於 PostgreSQL 的兩階段遞交支援。此事件也可能發生在查詢等待另一個交易遞交

時，特別是在關閉自動遞交的情況下。在這種情況下，即使更新尚未遞交，更新仍可能在 XactSync 上等待。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因
- 動作

支援的引擎版本

所有版本的 Aurora PostgreSQL 都支援此等待事件資訊。

Context

`IO:XactSync` 事件表示執行個體正在花時間等待 Aurora 儲存子系統確認已處理的交易資料。

等待變多的可能原因

`IO:XactSync` 事件比平時更常出現時，可能表示有效能問題，典型原因包括：

網路飽和

用戶端與資料庫執行個體之間的流量，或流向儲存子系統的流量，對網路頻寬而言可能太沉重。

CPU 壓力

工作負載繁重可能導致 Aurora 儲存常駐程式無法取得足夠的 CPU 時間。

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 監控資源
- 擴充 CPU 規模
- 增加網路頻寬
- 減少遞交次數

監控資源

若要查明 IO:XactSync 事件增加的原因，請檢查下列指標：

- WriteThroughput 和 CommitThroughput — 寫入輸送量或遞交輸送量的變化可能表示工作負載增加。
- WriteLatency 和 CommitLatency — 寫入延遲或遞認延遲的變化，可能表示要求儲存子系統執行更多工作。
- CPUUtilization — 如果執行個體的 CPU 使用率超過 90%，表示 Aurora 儲存常駐程式可能無法在 CPU 上取得足夠時間。在此情況下，輸入/輸出 效能會降低。

如需這些指標的相關資訊，請參閱 Amazon Aurora 的執行個體層級指標。

擴充 CPU 規模

若要解決 CPU 不足的問題，請考慮改用有更多 CPU 容量的執行個體類型。關於資料庫執行個體類別的 CPU 容量，如需相關資訊，請參閱 Aurora 資料庫執行個體類別的硬體規格。

增加網路頻寬

若要判斷執行個體是否達到網路頻寬限制，請檢查下列其他等待事件：

- IO:DataFileRead、IO:BufferRead、IO:BufferWrite 及 IO:XactWrite — 使用大量輸入/輸出的查詢可能產生更多這些等待事件。
- Client:ClientRead 和 Client:ClientWrite — 執行大量用戶端通訊的查詢可能產生更多這些等待事件。

如果問題在於網路頻寬，請考慮改為有更多網路頻寬的執行個體類型。關於資料庫執行個體類別的網路效能，如需相關資訊，請參閱 Aurora 資料庫執行個體類別的硬體規格。

減少遞交次數

若要減少遞交次數，請將陳述式合併成交易區塊。

IPC:DamRecordTxAck

IPC:DamRecordTxAck 事件表示在使用資料庫活動串流的工作階段中，Aurora PostgreSQL 產生活動串流事件，然後等待該事件變得持久。

主題

- 相關的引擎版本
- Context
- 原因
- 動作

相關的引擎版本

此等待事件資訊與所有 Aurora PostgreSQL 10.7 和更高的 10 版本、11.4 和更高的 11 版本，以及所有 12 和 13 版本有關。

Context

在同步模式下，活動串流事件的持久性優先於資料庫效能。在等待持久寫入事件時，工作階段會封鎖其他資料庫活動，造成 `IPC:DamRecordTxAck` 等待事件。

原因

如果 `IPC:DamRecordTxAck` 事件出現在最常等待名單中，最常見的原因是資料庫活動串流 (DAS) 功能為全面稽核。高階 SQL 活動會產生需要記錄的活動串流事件。

動作

我們根據等待事件的原因，建議不同的動作：

- 減少 SQL 陳述式或關閉資料庫活動串流。這樣做可減少需要持久寫入的事件數。
- 改為非同步模式。這樣做有助於減少 `IPC:DamRecordTxAck` 等待事件的爭用情形。

  不過，在非同步模式下，DAS 功能不保證每個事件的持久性。

IPC:parallel 等待事件

下列 `IPC:parallel wait events` 指出工作階段正在等待與平行查詢執行操作相關的程序間通訊。

- `IPC:BgWorkerStartup` - 程序正在等待平行工作者程序完成其啟動序列。初始化工作者以進行平行查詢執行時會發生這種情況。
- `IPC:BgWorkerShutdown` - 程序正在等待平行工作者程序完成其關閉順序。這會在平行查詢執行的清除階段期間發生。

- `IPC:ExecuteGather` - 程序正在等待在查詢執行期間從平行工作者程序接收資料。當領導程序需要從工作者收集結果時，就會發生這種情況。
- `IPC:ParallelFinish` - 程序正在等待平行工作者完成其執行並報告其最終結果。這會在平行查詢執行的完成階段發生。

主題

- 支援的引擎版本
- Context
- 等待時間增加的可能原因
- 動作

支援的引擎版本

所有版本的 Aurora PostgreSQL 都支援此等待事件資訊。

Context

PostgreSQL 中的平行查詢執行涉及多個程序共同處理單一查詢。當查詢判斷為適合平行處理時，領導程序會根據 `max_parallel_workers_per_gather` 參數設定，與一或多個平行工作者程序協調。領導程序會將工作分割給工作者，每個工作者都會獨立處理其部分的資料，並將結果收集回領導程序。

> **Note**
>
> 每個平行工作者會以單獨的程序運作，具有與完整使用者工作階段類似的資源需求。這表示與非平行查詢相比，具有 4 個工作者的平行查詢最多可耗用 5 倍的資源 (CPU、記憶體、I/O 頻寬)，因為領導程序和每個工作者程序都會維持自己的資源配置。針對執行個體，例如 `work_mem` 等設定會個別套用到每個工作者，可能會將所有程序之間的總記憶體用量相乘。

平行查詢架構包含三個主要元件：

- 領導程序：啟動平行操作、分割工作負載並與工作者程序協調的主要程序。
- 工作者程序：平行執行部分查詢的背景程序。
- 收集/收集合併：將多個工作者程序的結果合併回領導者的操作

在平行執行期間，程序需要透過程序間通訊 (IPC) 機制彼此通訊。這些 IPC 等待事件會在不同的階段發生：

- 工作者啟動：正在初始化平行工作者時
- 資料交換：工作者處理資料並將結果傳送給領導時
- 工作者關閉：平行執行完成且工作者終止時
- 同步點：當程序需要協調或等待其他程序完成其任務時

了解這些等待事件對於診斷與平行查詢執行相關的效能問題至關重要，尤其是在可能同時執行多個平行查詢的高度並行環境中。

等待時間增加的可能原因

有幾個因素可能會導致平行相關的 IPC 等待事件增加：

平行查詢的高度並行

當許多平行查詢同時執行時，可能會導致資源爭用並增加 IPC 操作的等待時間。這在具有高交易量或分析工作負載的系統中特別常見。

次佳的平行查詢計劃

如果查詢規劃器選擇效率不佳的平行計劃，可能會導致工作者之間不必要的平行化或工作分配不佳。這可能會導致 IPC 等待增加，尤其是 IPC:ExecuteGather 和 IPC:ParallelFinish 事件。這些規劃問題通常來自過時的統計資料和資料表/索引膨脹。

頻繁啟動和關閉平行工作者

頻繁啟動和終止平行工作者的短期查詢可能會導致 `IPC:BgWorkerStartup` 和 `IPC:BgWorkerShutdown` 事件增加。這通常在具有許多小型、可平行化查詢的 OLTP 工作負載中出現。

資源限制

有限的 CPU、記憶體或 I/O 容量可能會導致平行執行的瓶頸，進而增加所有 IPC 事件的等待時間。例如，如果 CPU 飽和，工作者程序可能需要更長的時間才能啟動或處理其工作部分。

複雜的查詢結構

具有多層平行處理的查詢 (例如，平行聯結後接平行彙總) 可能會導致更複雜的 IPC 模式，並可能增加等待時間，尤其是 `IPC:ExecuteGather` 事件。

大型結果集

產生大型結果集的查詢可能會導致 IPC:ExecuteGather 等待時間增加，因為領導程序會花費更多時間收集和處理工作者程序的結果。

了解這些因素有助於診斷和解決與 Aurora PostgreSQL 中平行查詢執行相關的效能問題。

動作

當您看到與平行查詢相關的等待時，這通常表示後端程序正在協調或等待平行工作者程序。這些等待在平行計劃執行期間很常見。您可以透過監控平行工作者用量、檢閱參數設定，以及調校查詢執行和資源配置，來調查和減輕這些等待的影響。

主題

- 分析低效率平行處理的查詢計劃
- 監控平行查詢用量
- 檢閱和調整平行查詢設定
- 最佳化資源配置
- 調查連線管理
- 檢閱和最佳化維護操作
- 利用查詢計劃管理 (QPM)

分析低效率平行處理的查詢計劃

平行查詢執行通常會導致系統不穩定、CPU 峰值和無法預測的查詢效能差異。徹底分析平行處理是否實際改善您的特定工作負載至關重要。使用 EXPLAIN ANALYZE 檢閱平行查詢執行計劃。

暫時停用工作階段層級的平行處理，以比較計劃效率：

```
SET max_parallel_workers_per_gather = 0;
EXPLAIN ANALYZE <your_query>;
```

重新啟用平行處理並比較：

```
RESET max_parallel_workers_per_gather;
```

```
EXPLAIN ANALYZE <your_query>;
```

如果停用平行處理可產生更好或更一致的結果，請考慮使用 SET 命令在工作階段層級針對特定查詢停用。為了獲得更廣泛的影響，您可能想要透過調整叢集或執行個體參數群組中的相關參數，在執行個體層級停用平行處理。如需詳細資訊，請參閱Amazon Aurora PostgreSQL 參數。

監控平行查詢用量

使用下列查詢來取得平行查詢活動和容量的可見性：

檢查作用中的平行工作者程序：

```
SELECT
    COUNT(*)
FROM
    pg_stat_activity
WHERE
    backend_type = 'parallel worker';
```

此查詢會顯示作用中平行工作者程序的數量。高值可能表示您的 `max_parallel_workers` 已設定為高值，建議您考慮將其減少。

檢查並行平行查詢：

```
SELECT
    COUNT(DISTINCT leader_pid)
FROM
    pg_stat_activity
WHERE
    leader_pid IS NOT NULL;
```

此查詢會傳回已啟動平行查詢的不同領導程序數目。此處的高數字表示多個工作階段同時執行平行查詢，這可能會增加對 CPU 和記憶體的需求。

檢閱和調整平行查詢設定

檢閱下列參數，以確保它們與您的工作負載一致：

- max_parallel_workers：所有工作階段的平行工作者總數。
- max_parallel_workers_per_gather：每個查詢的工作者上限。

對於 OLAP 工作負載，增加這些值可以改善效能。對於 OLTP 工作負載，通常偏好較低的值。

```
SHOW max_parallel_workers;
SHOW max_parallel_workers_per_gather;
```

最佳化資源配置

監控 CPU 使用率，如果持續很高，且您的應用程式受益於平行查詢，請考慮調整 vCPU 數量。確保有足夠記憶體可供平行操作使用。

- 使用 Performance Insights 指標來判斷系統是否受限於 CPU。
- 每個平行工作者都會使用自己的 work_mem。確保總記憶體用量在執行個體限制內。

平行查詢可能會耗用比非平行查詢更多的資源，因為每個工作者程序都是完全獨立的程序，對系統的影響與額外的使用者工作階段大致相同。選擇此設定的值時，以及設定控制資源使用率的其他設定時，應考慮這一點，例如 work_mem。如需詳細資訊，請參閱 PostgreSQL 文件。work_mem 等資源限制會個別套用至每個工作者，這表示所有程序的總使用率可能遠高於任何單一程序。

如果您的工作負載高度平行化，請考慮增加 vCPU 或調校記憶體參數。

調查連線管理

如果連線耗盡，請檢閱應用程式連線集區策略。如果尚未使用，請考慮在應用程式層級實作連線集區。

檢閱和最佳化維護操作

協調索引建立和其他維護任務，以防止資源爭用。請考慮在離峰時間排程這些操作。避免在高使用者查詢負載期間排程大量維護 (例如平行索引建置)。這些操作可能會耗用平行工作者，並影響定期查詢的效能。

利用查詢計劃管理 (QPM)

在 Aurora PostgreSQL 中，查詢計劃管理 (QPM) 旨在確保計劃適應性和穩定性，無論資料庫環境的變更是否可能導致查詢計劃迴歸。如需詳細資訊，請參閱 Aurora PostgreSQL 查詢計劃管理的概觀.QPM

提供最佳化工具的一些控制。在 QPM 中檢閱核准的計劃，以確保它們符合目前的平行處理設定。更新或移除可能強制執行次佳平行執行的過時計劃。

您也可以使用 pg_hint_plan 來修正計劃。如需詳細資訊，請參閱使用 pg_hint_plan 修正計劃。您可以使用名為 `Parallel` 的提示來強制執行平行執行。如需詳細資訊，請參閱平行計劃的提示。

IPC:ProcArrayGroupUpdate

當工作階段正在等待群組領導者在該操作結束時更新交易狀態時，就會`IPC:ProcArrayGroupUpdate`發生事件。雖然 PostgreSQL 通常會將 IPC 類型等待事件與平行查詢操作建立關聯，但此特定等待事件並非平行查詢特有的。

主題

- 支援的引擎版本
- Context
- 等待時間增加的可能原因
- 動作

支援的引擎版本

所有 Aurora PostgreSQL 的版本都支援此等待事件資訊。

Context

了解程序陣列 – 程序 (proc) 陣列是 PostgreSQL 中的共用記憶體結構。它保留所有執行中程序的相關資訊，包括交易詳細資訊。在交易完成期間 (`COMMIT` 或 `ROLLBACK`)，需要更新 ProcArray，以反映變更並從陣列清除 transactionID。嘗試完成其交易的工作階段必須在 ProcArray 上取得專屬鎖定。這可防止其他程序取得共用或獨佔鎖定。

群組更新機制 – 執行 COMMIT 或 ROLLBACK 時，如果後端程序無法在獨佔模式中取得 ProcArrayLock，則會更新稱為 ProcArrayGroupMember 的特殊欄位。這會將交易新增至要結束的工作階段清單。此後端程序接著會進入休眠，而其休眠時間會檢測為 ProcArrayGroupUpdate 等待事件。ProcArray with procArrayGroupMember 中的第一個程序稱為領導程序，會以獨佔模式取得 ProcArrayLock。然後，它會清除等待群組 transactionID 清除的程序清單。完成後，領導者會釋出 ProcArrayLock，然後喚醒此清單中的所有程序，通知他們交易已完成。

等待時間增加的可能原因

執行的程序越多，領導者在排他模式下保留至 procArrayLock 的時間就越長。因此，群組更新案例中的寫入交易越多，導致等待`ProcArrayGroupUpdate`事件的程序可能堆積。在 Database Insights 的最高 SQL 檢視中，您會看到 COMMIT 是具有大部分此等待事件的陳述式。這是預期的，但需要對正在執行的特定寫入 SQL 進行更深入的調查，以決定要採取何種適當動作。

動作

根據等待事件的原因，我們會建議不同的動作。使用 Amazon RDS Performance Insights 或查詢 PostgreSQL 系統檢視 來識別`IPC:ProcArrayGroupUpdate`事件`pg_stat_activity`。

主題

- 監控交易遞交和轉返操作
- 減少並行
- 實作連線集區

監控交易遞交和轉返操作

監控遞交和轉返 – 遞交和轉返的數量增加可能會導致 ProcArray 的壓力增加。例如，如果 SQL 陳述式因為增加的重複金鑰違規而開始失敗，您可能會看到回復增加，這可能會增加 ProcArray 爭用和資料表膨脹。

Amazon RDS Database Insights 提供 PostgreSQL 指標`xact_rollback`，`xact_commit`並報告每秒遞交和轉返的數量。

減少並行

批次交易 – 盡可能在單一交易中批次操作，以減少遞交/轉返操作。

限制並行 – 減少並行作用中交易的數量，以減輕 ProcArray 上的鎖定爭用。雖然它需要一些測試，但減少並行連線的總數可以減少爭用和維護輸送量。

實作連線集區

連線集區解決方案 – 使用連線集區有效率地管理資料庫連線，減少後端總數，進而減少 ProcArray 上的工作負載。雖然它需要一些測試，但減少並行連線的總數可以減少爭用和維護輸送量。

如需詳細資訊，請參閱 Aurora PostgreSQL 的連線集區。

減少連線風暴 – 同樣地，經常建立和終止連線的模式會對 ProcArray 造成額外的壓力。透過減少此模式，可減少整體爭用。

Lock:advisory

`Lock:advisory` 事件表示 PostgreSQL 應用程式使用鎖定來協調多個工作階段的活動。

主題

- 相關的引擎版本
- Context
- 原因
- 動作

相關的引擎版本

此等待事件資訊與 Aurora PostgreSQL 9.6 版及更新版本有關。

Context

PostgreSQL 諮詢鎖定是應用程式層級的合作式鎖定，由使用者的應用程式碼明確鎖定和解除鎖定。應用程式可以使用 PostgreSQL 諮詢鎖定來協調多個工作階段的活動。不同於物件層級或資料列層級的一般鎖定，應用程式完全控制鎖定的生命週期。如需詳細資訊，請參閱 PostgreSQL 文件中的諮詢鎖定。

諮詢鎖定可以在交易結束之前釋放，或由工作階段跨交易保留。但系統強制的隱含鎖定不是如此，例如 `CREATE INDEX` 陳述式在資料表上取得的存取獨佔鎖定。

如需用來取得 (鎖定) 和釋放 (解除鎖定) 諮詢鎖定的函數描述，請參閱 PostgreSQL 說明文件中的諮詢鎖定函數。

諮詢鎖定是在一般 PostgreSQL 鎖定系統上實作，在 `pg_locks` 系統檢視表中可見。

原因

明確使用此鎖定類型的應用程式可完全控制鎖定。如果查詢為每個資料列都取得諮詢鎖定，可能會造成鎖定激增或長期累積。

當查詢取得的鎖定比查詢傳回的資料列更多時，就會出現這些現象。應用程式最終必須釋放每個鎖定，但如果是在未傳回的資料列上取得鎖定，則應用程式無法找齊全部的鎖定。

下列範例來自 PostgreSQL 文件中的諮詢鎖定。

```
SELECT pg_advisory_lock(id) FROM foo WHERE id > 12345 LIMIT 100;
```

在此範例中，只有在內部選取資料列並鎖定其 ID 值之後，LIMIT 子句才能停止查詢的輸出。突然發生這種情況表示資料量不斷增加，導致規劃工具選擇另一個未經過開發期間測試的執行計劃。在此情況下，發生累積起因於應用程式對每個已鎖定的 ID 值,明確呼叫 pg_advisory_unlock。但是，在此情況下，找不到在未傳回的資料列上取得的鎖定集。因為是在工作階段層級取得鎖定，交易結束時不會自動釋放鎖定。

鎖定嘗試受阻次數激增的另一個可能原因是意外衝突。在這些衝突中，應用程式的無關聯部分不慎共用相同的鎖定 ID 空間。

動作

檢閱諮詢鎖定的應用程式用量，並詳述在應用程式流程中何處和何時取得和釋放每一種諮詢鎖定。

查明究竟是工作階段取得太多鎖定，還是長時間執行的工作階段未及早釋放鎖定，導致鎖定逐漸堆積。您可以使用 pg_terminate_backend(pid) 結束工作階段，以免工作階段層級鎖定逐漸堆積。

等待諮詢鎖定的用戶端以 wait_event_type=Lock 和 wait_event=advisory 出現在 pg_stat_activity 中。您可以在 pg_locks 系統檢視表中查詢相同的 pid，尋找 locktype=advisory 和 granted=f，以取得特定的鎖定值。

然後，您可以在 pg_locks 中查詢 granted=t 的同一個諮詢鎖定，以識別引起封鎖的工作階段，如下列範例所示。

```
SELECT blocked_locks.pid AS blocked_pid,
         blocking_locks.pid AS blocking_pid,
         blocked_activity.usename AS blocked_user,
         blocking_activity.usename AS blocking_user,
         now() - blocked_activity.xact_start AS blocked_transaction_duration,
         now() - blocking_activity.xact_start AS blocking_transaction_duration,
         concat(blocked_activity.wait_event_type,':',blocked_activity.wait_event) AS
 blocked_wait_event,
         concat(blocking_activity.wait_event_type,':',blocking_activity.wait_event) AS
 blocking_wait_event,
         blocked_activity.state AS blocked_state,
         blocking_activity.state AS blocking_state,
         blocked_locks.locktype AS blocked_locktype,
         blocking_locks.locktype AS blocking_locktype,
         blocked_activity.query AS blocked_statement,
```

```
         blocking_activity.query AS blocking_statement
   FROM pg_catalog.pg_locks blocked_locks
   JOIN pg_catalog.pg_stat_activity blocked_activity ON blocked_activity.pid =
blocked_locks.pid
   JOIN pg_catalog.pg_locks blocking_locks
       ON blocking_locks.locktype = blocked_locks.locktype
       AND blocking_locks.DATABASE IS NOT DISTINCT FROM blocked_locks.DATABASE
       AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
       AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
       AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
       AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
       AND blocking_locks.transactionid IS NOT DISTINCT FROM
blocked_locks.transactionid
       AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
       AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
       AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
       AND blocking_locks.pid != blocked_locks.pid
   JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid =
blocking_locks.pid
   WHERE NOT blocked_locks.GRANTED;
```

所有諮詢鎖定 API 函數都有兩組引數，可能是一個 bigint 引數或兩個 integer 引數：

- 如果 API 函數有一個 bigint 引數，則前段 32 位元在 pg_locks.classid 中，後段 32 位元在 pg_locks.objid 中。
- 如果 API 函數有兩個 integer 引數，則第一個引數為 pg_locks.classid，第二個引數為 pg_locks.objid。

pg_locks.objsubid 值表示使用何種 API 形式：1 表示一個 bigint 引數；2 表示二個 integer 引數。

Lock:extend

Lock:extend 事件表示後端程序正在等待鎖定關聯來延伸，但另一個程序也基於相同目的而鎖定該關聯。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因

- 動作

支援的引擎版本

所有版本的 Aurora PostgreSQL 都支援此等待事件資訊。

Context

Lock:extend 事件表示後端程序正在等待延伸關聯，但另一個後端程序已鎖定該關聯而正在延伸。因為一次只有一個程序可以延伸關聯，所以系統產生 Lock:extend 等待事件。INSERT、COPY 及UPDATE 操作可能產生此事件。

等待變多的可能原因

Lock:extend 事件比平時更常出現時，可能表示有效能問題，典型原因包括：

突增並行插入或更新同一個資料表

以查詢來插入或更新同一個資料表的並行工作階段可能變多。

網路頻寬不足

資料庫執行個體上的網路頻寬可能不足以滿足目前工作負載的儲存通訊需求。這可能會引起儲存延遲，導致 Lock:extend 事件增加。

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 減少並行插入和更新同一個關聯
- 增加網路頻寬

減少並行插入和更新同一個關聯

首先，判斷 tup_inserted 和 tup_updated 指標是否增加，以及此等待事件是否也隨之增加。如果是，請檢查哪些關聯激烈爭用插入和更新操作。若要查明，請在 pg_stat_all_tables 檢視表中查詢 n_tup_ins 和 n_tup_upd 欄位的值。如需 pg_stat_all_tables 檢視表的相關資訊，請參閱 PostgreSQL 文件中的 pg_stat_all_tables。

關於引起封鎖的查詢和被封鎖的查詢，如需詳細資訊，請查詢 pg_stat_activity，如下列範例所示：

```
SELECT
    blocked.pid,
    blocked.usename,
    blocked.query,
    blocking.pid AS blocking_id,
    blocking.query AS blocking_query,
    blocking.wait_event AS blocking_wait_event,
    blocking.wait_event_type AS blocking_wait_event_type
FROM pg_stat_activity AS blocked
JOIN pg_stat_activity AS blocking ON blocking.pid = ANY(pg_blocking_pids(blocked.pid))
where
blocked.wait_event = 'extend'
and blocked.wait_event_type = 'Lock';

   pid  | usename  |            query             | blocking_id |
            blocking_query                          | blocking_wait_event |
 blocking_wait_event_type
  ------+----------+------------------------------+-------------
+-------------------------------------------------------------------
+---------------------+--------------------------
   7143 |  myuser  | insert into tab1 values (1); |        4600 | INSERT INTO tab1 (a)
 SELECT s FROM generate_series(1,1000000) s; | DataFileExtend      | IO
```

在找出導致 Lock:extend 事件增加的關聯之後，請使用下列技巧來減少爭用：

- 查明您是否可以使用分割來減少爭用同一個資料表。將插入或更新的元組分成不同分割區可以減少爭用。如需分割的相關資訊，請參閱使用 pg_partman 擴充功能來管理 PostgreSQL 分割區。
- 如果等待事件主要是由於更新活動，請考慮降低關聯的 fillfactor 值。這樣可以減少在更新期間請求新區塊。fillfactor 是資料表的儲存參數，決定可供壓縮資料表分頁的最大空間。以分頁總空間的百分比表示。如需 fillfactor 參數的詳細資訊，請參閱 PostgreSQL 文件中的 CREATE TABLE。

> **Important**
>
> 如果您變更 fillfactor，強烈建議您測試系統，因為變更此值可能對效能造成負面影響，視工作負載而定。

增加網路頻寬

若要檢查寫入延遲是否變長，請在 CloudWatch 中檢查 `WriteLatency` 指標。如果是，請使用 `WriteThroughput` 和 `ReadThroughput` Amazon CloudWatch 指標來監控資料庫叢集上的儲存相關流量。這些指標可協助您判斷網路頻寬是否足以應付工作負載的儲存活動。

如果網路頻寬不夠，請增加頻寬。如果資料庫執行個體快達到網路頻寬限制，則增加頻寬的唯一辦法就是加大資料庫執行個體。

如需 CloudWatch 指標的詳細資訊，請參閱Amazon Aurora 的 Amazon CloudWatch 指標。關於每個資料庫執行個體類別的網路效能，如需相關資訊，請參閱 Aurora 資料庫執行個體類別的硬體規格。

Lock:Relation

`Lock:Relation` 事件表示查詢正在等待取得鎖定的資料表或檢視表 (關聯)，目前由另一個交易鎖定。

主題

- 支援的引擎版本
- Context
- 等待時間增加的可能原因
- 動作

支援的引擎版本

所有版本的 Aurora PostgreSQL 都支援此等待事件資訊。

Context

大多數 PostgreSQL 命令隱含地使用鎖定來控制並行存取資料表中的資料。您也可以在應用程式碼中使用 `LOCK` 命令，以明確使用這些鎖定。許多鎖定模式彼此不相容，在嘗試存取同一個物件時，可能封鎖交易。發生這種情況時，Aurora PostgreSQL 會產生 `Lock:Relation` 事件。以下是一些常見例子：

- 獨佔鎖定 (例如 `ACCESS EXCLUSIVE`) 會封鎖所有並行存取。資料定義語言 (DDL) 操作 (例如 `DROP TABLE`、`TRUNCATE`、`VACUUM FULL` 及 `CLUSTER`) 隱含地取得 `ACCESS EXCLUSIVE` 鎖定。對於未明確指定模式的 `LOCK TABLE` 陳述式，`ACCESS EXCLUSIVE` 也是預設鎖定模式。
- 在資料表上使用 `CREATE INDEX (without CONCURRENT)` 時，與取得 `ROW EXCLUSIVE` 鎖定的資料處理語言 (DML) 陳述式 `UPDATE`、`DELETE` 及 `INSERT` 會發生衝突。

如需表格層級鎖定和衝突鎖定模式的詳細資訊，請參閱 PostgreSQL 文件中的明確鎖定。

引起封鎖的查詢和交易通常透過下列其中一種方法解除封鎖：

- 引起封鎖的查詢 — 由應用程式取消查詢，或由使用者結束程序。透過工作階段的陳述式逾時或死鎖偵測機制，引擎也可以強制結束查詢。
- 引起封鎖的交易 — 交易執行 ROLLBACK 或 COMMIT 陳述式來停止封鎖。當工作階段由用戶端或因為網路問題而中斷連線時，或工作階段結束時，也會自動復原。資料庫引擎關閉、系統記憶體不足等原因會結束工作階段。

等待時間增加的可能原因

當 Lock:Relation 事件發生頻率高於正常情況時，則可能表示效能問題。典型原因包括：

資料表鎖定發生衝突的並行工作階段變多

以查詢來鎖定同一個資料表但鎖定模式衝突的並行工作階段可能變多。

維護操作

運作狀態維護操作 (例如 VACUUM 和 ANALYZE) 可能大幅增加衝突鎖定的數量。VACUUM FULL 取得 ACCESS EXCLUSIVE 鎖定，ANALYZE 取得 SHARE UPDATE EXCLUSIVE 鎖定。這兩種鎖定都可能引起 Lock:Relation 等待事件。應用程式資料維護操作 (例如重新整理具體化檢視表) 也會使鎖定的查詢和交易增加。

鎖定讀取器執行個體

寫入器和讀取器保有的關係鎖之間可能存在衝突。目前，只有 ACCESS EXCLUSIVE 關係鎖被複製到讀取器執行個體。但是，ACCESS EXCLUSIVE 關係鎖將與讀取器所持有的任何 ACCESS SHARE 關係鎖發生衝突。這可能會造成讀取器上的鎖定關係等待事件增加。

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 減少封鎖 SQL 陳述式的影響
- 將維護操作的影響降至最低
- 檢查讀取器鎖定

減少封鎖 SQL 陳述式的影響

若要減少封鎖 SQL 陳述式的影響，請盡可能修改應用程式碼。以下是減少封鎖的兩種常用技巧：

- 使用 NOWAIT 選項 — 某些 SQL 命令支援此選項，例如 SELECT 和 LOCK 陳述式。如果無法立即獲得鎖定，NOWAIT 指令會取消提出鎖定請求的查詢。這項技巧有助於避免引起封鎖的工作階段背後堆積被封鎖的工作階段。

  例如：假設交易 A 等待的鎖定由交易 B 持有。現在，如果 B 請求鎖定的資料表由交易 C 鎖定，則可能封鎖交易 A，直到交易 C 完成為止。但是，如果交易 B 請求鎖定 C 時使用 NOWAIT，則會很快失敗，以確保交易 A 不必無限期等待。
- 使用 SET lock_timeout - 設定 lock_timeout 值來限制 SQL 陳述式在關聯上取得鎖定所等待的時間。如果在指定的逾時內未獲得鎖定，則會取消提出鎖定請求的交易。請在工作階段層級設定此值。

將維護操作的影響降至最低

維護操作很重要，例如 VACUUM 和 ANALYZE。建議不要因為發現這些維護操作相關的 Lock:Relation 等待事件而關閉維護。下列方法可以將這些操作的影響降至最低：

- 在離峰時段手動執行維護操作。
- 若要減少 Lock:Relation 等待，請執行任何所需的自動資料清理調校。如需調整自動清理的相關資訊，請參閱《Amazon RDS 使用者指南》中的在 Amazon RDS 上使用 PostgreSQL 自動資料清理。

檢查讀取器鎖定

您可以檢查寫入器和讀取器上的並行工作階段是否持有相互封鎖的鎖定。作法是執行傳回鎖類型和關聯的查詢。您可在資料表中找到兩個此類並行工作階段的查詢序列，即寫入器工作階段和讀取器工作階段。

重播過程將在取消讀取器查詢之前等待 max_standby_streaming_delay 持續時間。如範例所示，100ms 的鎖定逾時遠低於預設 max_standby_streaming_delay 的 30 秒。鎖定在出現問題之前逾時。

序列事件	Session (工作階段)	命令或輸出

設定名為 READER 且具有指定值的環境變數，並嘗試使用此端點連線至資料庫執行個體。	讀取器工作階段	CLI 命令：
		`export READER=aurorapg2.12345678910.us-west-1.rds.amazonaws.com`
		`psql -h $READER`
		輸出：
		`psql (15devel, server 10.14)`
		`Type "help" for help.`

設定名為 WRITER 的環境變數，並嘗試使用此端點連線至資料庫執行個體。	寫入器工作階段	CLI 命令：
		`export WRITER=aurorapg1.12345678910.us-west-1.rds.amazonaws.com`
		`psql -h $WRITER`
		輸出：
		`psql (15devel, server 10.14)`
		`Type "help" for help.`

寫入器工作階段在寫入器執行個體上建立 t1 資料表。	寫入器工作階段	PostgreSQL 查詢：
		`postgres=> CREATE TABLE t1(b integer);`
		`CREATE TABLE`

序列事件	Session (工作階段)	命令或輸出
如果寫入器上沒有衝突的查詢，則會立即在寫入器上取得 ACCESS EXCLUSIVE 鎖定。	寫入器工作階段	ACCESS EXCLUSIVE 鎖定已啟用

讀取器工作階段設定 100 毫秒的鎖定逾時間隔。	讀取器工作階段	PostgreSQL 查詢：
		`postgres=> SET lock_timeout=100;`
		`SET`

讀取器工作階段在讀取器執行個體上嘗試從 t1 資料表讀取資料。	讀取器工作階段	PostgreSQL 查詢：
		`postgres=> SELECT * FROM t1;`
		輸出範例：
		`b`
		`---`
		`(0 rows)`

寫入器工作階段捨棄 t1。	寫入器工作階段	PostgreSQL 查詢：
		`postgres=> BEGIN;`
		`BEGIN`
		`postgres=> DROP TABLE t1;`
		`DROP TABLE`
		`postgres=>`

序列事件	Session (工作階段)	命令或輸出

在讀取器上，查詢逾時，已取消。	讀取器工作階段	PostgreSQL 查詢：
		`postgres=> SELECT * FROM t1;`
		輸出範例：
		`ERROR: canceling statement due to lock timeout`
		`LINE 1: SELECT * FROM t1;`
		`^`

為了查明錯誤原因，讀取器工作階段會查詢 pg_locks 和 pg_stat_activity 。	讀取器工作階段	PostgreSQL 查詢：
		`postgres=> SELECT locktype, relation, mode, backend_type`
		`postgres=> FROM pg_locks l, pg_stat_activity t1`
		`postgres=> WHERE l.pid=t1.pid AND relation = 't1'::regclass::oid;`

序列事件	Session (工作階段)	命令或輸出

結果表示 aurora wal replay 程序在資料表 t1 上持有 ACCESS EXCLUSIVE 鎖定。	讀取器工作階段	查詢結果：
		`locktype | relation | mode | backend_type`
		`----------+----------+-------------------+---------------`
		`relation | 68628525 | AccessExclusiveLock | aurora wal replay`
		`(1 row)`

Lock:transactionid

Lock:transactionid 事件表示交易正在等待資料列層級鎖定。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因
- 動作

支援的引擎版本

所有版本的 Aurora PostgreSQL 都支援此等待事件資訊。

Context

Lock:transactionid 事件表示交易嘗試取得的資料列層級鎖定已授予同一時間執行的另一個交易。因為此鎖定，已封鎖出現 Lock:transactionid 等待事件的工作階段。當引起封鎖的交易在 COMMIT 或 ROLLBACK 陳述式中結束時，被封鎖的交易就可以繼續進行。

Aurora PostgreSQL 的多版本並行控制語意保證讀取器不會封鎖寫入器，而寫入器也不會封鎖讀取器。引起封鎖和被封鎖的交易必須發出下列類型的衝突陳述式，才會發生資料列層級衝突：

- UPDATE
- SELECT … FOR UPDATE
- SELECT … FOR KEY SHARE

SELECT … FOR KEY SHARE 陳述式是特殊情況。資料庫使用 FOR KEY SHARE 子句來最佳化參考完整性的效能。如果資料列上有資料列層級鎖定，則會封鎖其他資料表上參考此資料列的 INSERT、UPDATE 及 DELETE 命令。

等待變多的可能原因

此事件比平時更常出現時，通常是因為 UPDATE、SELECT … FOR UPDATE 或SELECT … FOR KEY SHARE 陳述式兼具下列情況。

主題

- 高度並行
- 交易閒置
- 長時間執行的交易

高度並行

Aurora PostgreSQL 可以使用精密的資料列層級鎖定語意。有下列情況時，較可能發生資料列層級衝突：

- 高度並行工作負載爭用相同的資料列。
- 並行增加。

交易閒置

有時候 pg_stat_activity.state 資料欄會顯示 idle in transaction 值。已啟動交易但尚未發出 COMMIT 或 ROLLBACK 的工作階段會出現此值。如果 pg_stat_activity.state 值不是 active，pg_stat_activity 中會顯示最近要完成執行的查詢。因為開啟的交易持有鎖定，引起封鎖的工作階段目前未處理查詢。

如果閒置交易已獲得資料列層級鎖定，可能會阻止其他工作階段取得鎖定。這種情況導致頻繁發生等待事件 Lock:transactionid。若要診斷問題，請檢查 pg_stat_activity 和 pg_locks 的輸出。

長時間執行的交易

長時間執行的交易會長時間持有鎖定。這些長期持有的鎖定可能阻止其他交易執行。

動作

資料列鎖定會造成 UPDATE、SELECT … FOR UPDATE 或 SELECT … FOR KEY SHARE 陳述式之間發生衝突。嘗試解決之前，請查明這些陳述式是否在同一個資料列上執行。使用此資訊來選擇以下各節描述的策略。

主題

- 因應高度並行
- 因應閒置的交易
- 因應長時間執行的交易

因應高度並行

如果問題在於並行，請嘗試下列其中一項技巧：

- 降低應用程式中的並行。例如，減少作用中工作階段的數目。
- 實作連線集區。若要了解如何使用 RDS Proxy 來建立連線集區，請參閱 Amazon RDS Proxy for Aurora。
- 將應用程式或資料模型設計成避免爭用 UPDATE 和 SELECT … FOR UPDATE 陳述式。您也可以減少 SELECT … FOR KEY SHARE 陳述式存取的外部索引鍵數目。

因應閒置的交易

如果 pg_stat_activity.state 顯示 idle in transaction，請使用下列策略：

- 盡可能開啟自動遞交。這種方法可以防止交易在等待 COMMIT 或 ROLLBACK 時封鎖其他交易。
- 搜尋缺少 COMMIT、ROLLBACK 或 END 的程式碼路徑。
- 確保應用程式中的異常處理邏輯一定有路徑通往有效 end of transaction。
- 確保應用程式以 COMMIT 或 ROLLBACK 結束交易後處理查詢結果。

因應長時間執行的交易

如果長時間執行的交易導致頻繁出現 Lock:transactionid，請嘗試下列策略：

- 避免長時間執行的交易使用資料列鎖定。
- 盡可能實作自動遞交來限制查詢的長度。

Lock:tuple

`Lock:tuple` 事件表示後端程序正等待在元組上取得鎖定。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因
- 動作

支援的引擎版本

所有版本的 Aurora PostgreSQL 都支援此等待事件資訊。

Context

`Lock:tuple` 事件表示後端正等待在元組上取得鎖定，但另一個後端在同一個元組上持有衝突鎖定。下表說明工作階段產生 `Lock:tuple` 事件的情節。

時間	工作階段 1	工作階段 2	工作階段 3
t1	開始交易。		
t2	更新資料列 1。		
t3		更新資料列 1。工作階段在元組上取得獨佔鎖定，然後等待工作階段 1 遞交或復原來釋放鎖定。	
t4			更新資料列 1。工作階段等待工作階段 2 釋放元組上的獨佔鎖定。

或者，您可以使用基準化分析工具 pgbench 來模擬此等待事件。使用自訂 SQL 檔案，將大量並行工作階段設定成在資料表中更新相同資料列。

若要進一步了解衝突鎖定模式，請參閱 PostgreSQL 文件中的明確鎖定。若要進一步了解 pgbench，請參閱 PostgreSQL 文件中的 pgbench。

等待變多的可能原因

此事件比平時更常出現時，可能表示有效能問題，典型原因包括：

- 大量並行工作階段執行 UPDATE 或 DELETE 陳述式，嘗試取得同一個元組的衝突鎖定。
- 高度並行工作階段使用 FOR UPDATE 或 FOR NO KEY UPDATE 鎖定模式來執行 SELECT 陳述式。
- 各種因素迫使應用程式或連線集區開啟更多工作階段來執行相同的操作。由於新的工作階段嘗試修改相同的資料行，資料庫負載會激增，並出現 Lock:tuple。

如需詳細資訊，請參閱 PostgreSQL 文件中的資料列層級鎖定。

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 調查應用程式邏輯
- 尋找引起封鎖的工作階段
- 減少高度並行
- 瓶頸疑難排解

調查應用程式邏輯

查明引起封鎖的工作階段是否長時間處於 idle in transaction 狀態。如果是，請考慮結束引起封鎖的工作階段，當作短期的解決辦法。您也可以使用 pg_terminate_backend 函數。如需此函數的詳細資訊，請參閱 PostgreSQL 文件中的伺服器訊號函數。

如需長期解決方案，請執行下列動作：

- 調整應用程式邏輯。

- 使用 idle_in_transaction_session_timeout 參數。任何工作階段中，如果開啟的交易已閒置超過一段指定的時間，此參數會結束工作階段。如需詳細資訊，請參閱 PostgreSQL 文件中的用戶端連線預設值。
- 盡可能使用自動遞交。如需詳細資訊，請參閱 PostgreSQL 文件中的 SET AUTOCOMMIT。

尋找引起封鎖的工作階段

Lock:tuple 等待事件發生時，請查明哪些鎖定彼此相依，以找出引起封鎖和被封鎖的工作階段。如需詳細資訊，請參閱 PostgreSQL Wiki 中的鎖定相依性資訊。若要分析過去的 Lock:tuple 事件，請使用 Aurora 函數 aurora_stat_backend_waits。

下列範例查詢所有工作階段，篩選 tuple 並依據 wait_time 排序。

```
--AURORA_STAT_BACKEND_WAITS
      SELECT a.pid,
             a.usename,
             a.app_name,
             a.current_query,
             a.current_wait_type,
             a.current_wait_event,
             a.current_state,
             wt.type_name AS wait_type,
             we.event_name AS wait_event,
             a.waits,
             a.wait_time
        FROM (SELECT pid,
                     usename,
                     left(application_name,16) AS app_name,
                     coalesce(wait_event_type,'CPU') AS current_wait_type,
                     coalesce(wait_event,'CPU') AS current_wait_event,
                     state AS current_state,
                     left(query,80) as current_query,
                     (aurora_stat_backend_waits(pid)).*
                FROM pg_stat_activity
               WHERE pid <> pg_backend_pid()
                 AND usename<>'rdsadmin') a
NATURAL JOIN aurora_stat_wait_type() wt
NATURAL JOIN aurora_stat_wait_event() we
WHERE we.event_name = 'tuple'
    ORDER BY a.wait_time;
```

```
  pid   | usename | app_name |                 current_query                  |
 current_wait_type | current_wait_event | current_state | wait_type | wait_event |
 waits | wait_time
-------+---------+----------+------------------------------------------------
+-------------------+--------------------+--------------+-----------+------------
+-------+-----------
 32136 | sys     | psql     | /*session3*/ update tab set col=1 where col=1; | Lock
           | tuple              | active        | Lock      | tuple      |     1 |
 1000018
 11999 | sys     | psql     | /*session4*/ update tab set col=1 where col=1; | Lock
           | tuple              | active        | Lock      | tuple      |     1 |
 1000024
```

減少高度並行

Lock:tuple 事件可能持續發生，尤其在忙碌的工作負載時段。在此情況下，對於非常忙碌的資料列，請考慮減少高度並行。通常只有少數資料列控制佇列或布林邏輯，使得這些資料列非常忙碌。

您可以根據商業需求、應用程式邏輯和工作負載類型，使用不同方法來減少並行。例如，您可以執行下列動作：

- 重新設計資料表和資料邏輯來減少高度並行。
- 變更應用程式邏輯來減少資料列層級的高度並行。
- 善用並重新設計含有資料列層級鎖定的查詢。
- 對重試操作使用 NOWAIT 子句。
- 考慮使用樂觀和混合鎖定邏輯並行控制。
- 考慮變更資料庫隔離層級。

瓶頸疑難排解

Lock:tuple 可能發生瓶頸，例如 CPU 不足或 Amazon EBS 頻寬耗盡。若要減少瓶頸，請考慮下列方法：

- 擴充執行個體類別類型的規模。
- 將消耗大量資源的查詢最佳化。
- 變更應用程式邏輯。
- 封存不常存取的資料。

LWLock:buffer_content (BufferContent)

LWLock:buffer_content 事件表示工作階段正等待在記憶體中讀取或寫入資料分頁，但另一個工作階段已鎖定該分頁來寫入。在 Aurora PostgreSQL 13 及更新版本中，此等待事件稱為 BufferContent。

主題

- 支援的引擎版本
- Context
- 等待時間增加的可能原因
- 動作

支援的引擎版本

所有版本的 Aurora PostgreSQL 都支援此等待事件資訊。

Context

為了讀取或操作資料，PostgreSQL 透過共用記憶體緩衝區來存取資料。為了讀取緩衝區，程序以共用模式在緩衝區內容上取得輕量級鎖定 (LWLock)。為了寫入緩衝區，程序以獨佔模式取得該鎖定。共用鎖定允許其他程序同時在該內容上取得共用鎖定。獨佔鎖定阻止其他程序在該內容上得得任何類型的鎖定。

LWLock:buffer_content (BufferContent) 事件表示多個程序正嘗試在特定緩衝區的內容上取得輕量級鎖定 (LWLocks)。

等待時間增加的可能原因

LWLock:buffer_content (BufferContent) 事件比平時更常出現時，可能表示有效能問題，典型原因包括：

更常並行更新相同資料

以查詢來更新相同緩衝區內容的並行工作階段可能變多。在有大量索引的資料表上，這種爭用可能更明顯。

工作負載資料不在記憶體中

當作用中工作負載處理的資料不在記憶體中時，這些等待事件可能增加。這是因為程序在執行磁碟輸入/輸出操作時持有鎖定更久。

過度使用外部索引鍵限制

外部索引鍵限制會延長程序持有緩衝區內容鎖定的時間。這是因為讀取操作在更新參考的索引鍵時，在該索引鍵上需要共用緩衝區內容鎖定。

動作

根據等待事件的原因，我們會建議不同的動作。您可以使用 Amazon RDS 績效詳情或查詢 `LWLock:buffer_content` 檢視表來識別 `BufferContent` (`pg_stat_activity`) 事件。

主題

- 改善記憶體內效率
- 減少使用外部索引鍵限制
- 移除未使用的索引
- 移除重複的索引
- 捨棄或 REINDEX 無效索引
- 使用部分索引
- 移除資料表和索引膨脹

改善記憶體內效率

若要讓作用中工作負載資料更有機會留在記憶體中，請分割資料表或擴充執行個體類別的規模。如需資料庫執行個體類別的相關資訊，請參閱 Amazon Aurora 資料庫執行個體類別。

監控 `BufferCacheHitRatio` 指標會測量資料庫叢集中資料庫執行個體之緩衝區快取所提供的請求百分比。這個指標可讓您深入了解記憶體提供了多少資料量。高命中率表示您的資料庫執行個體有足夠的記憶體可供您的工作資料集使用，而低比率則表示您的查詢經常從儲存體存取資料。

PG Collector 報告記憶體設定區段下的每個資料表的快取讀取命中率和每個索引的快取讀取命中率，可以提供資料表和快取命中率的洞見。

減少使用外部索引鍵限制

調查遇到大量 `LWLock:buffer_content` (`BufferContent`) 等待事件的工作負載如何使用外部引索鍵限制。刪除不必要的外部索引鍵限制。

移除未使用的索引

對於遇到大量 LWLock:buffer_content (BufferContent) 等待事件的工作負載，請識別未使用的索引並移除。

PG Collector 報告的未使用索引區段可以提供資料庫中未使用索引的洞見。

移除重複的索引

識別重複的索引並將其移除。

PG Collector 報告的重複索引區段可以提供資料庫中重複索引的洞見。

捨棄或 REINDEX 無效索引

使用 CREATE INDEX CONCURRENTLY 或 REINDEX CONCURRENTLY 時通常會發生無效的索引，且命令會失敗或中止。

無效的索引無法用於查詢，但仍會更新並佔用磁碟空間。

PG Collector 報告的無效索引區段可以提供資料庫中無效索引的洞見。

使用部分索引

您可以利用部分索引來增強查詢效能並減少索引大小。部分索引是建置在資料表子集上的索引，子集由條件式運算式所定義。如部分索引文件中詳述，部分索引可減少維護索引的額外負荷，因為 PostgreSQL 在所有情況下都不需要更新索引。

移除資料表和索引膨脹

資料表和索引膨脹過多可能會對資料庫效能產生負面影響。膨脹的資料表和索引會增加作用中的工作集大小，降低內部記憶體效率。此外，膨脹會增加儲存成本並減慢查詢執行速度。若要診斷膨脹，請參閱診斷資料表和索引膨脹。此外，PG Collector 報告的分段 (膨脹) 區段可以提供資料表和索引膨脹的洞見。

若要處理資料表和索引膨脹，有幾個選項：

VACUUM FULL

> VACUUM FULL 會建立新的資料表複本，僅複製即時元組，然後以新的資料表取代舊資料表，同時保留 ACCESS EXCLUSIVE 鎖定。這可防止讀取或寫入資料表，這可能會導致中斷。此外，如果資料表很大，VACUUM FULL 將需要更長的時間。

pg_repack

在 VACUUM FULL 可能不適合的情況下，pg_repack 很有用。它會建立新的資料表，其中包含膨脹資料表的資料、追蹤原始資料表的變更，然後將原始資料表取代為新的資料表。在建立新資料表時，它不會鎖定原始資料表以進行讀取或寫入操作。如需如何使用 pg_repack 的詳細資訊，請參閱使用 pg_repack 移除膨脹和 pg_repack。

REINDEX

REINDEX 命令可用來解決索引膨脹問題。REINDEX 會撰寫索引的新版本，而不會有無效頁面或空白或幾乎空白的頁面，進而減少索引的空間耗用。如需 REINDEX 命令的詳細資訊，請參閱 REINDEX 文件。

從資料表和索引移除膨脹後，可能需要增加這些資料表上的自動清空頻率。在資料表層級實作積極的自動清空設定，有助於防止未來發生膨脹。如需詳細資訊，請參閱 Vacuuming and analyzing tables automatically 上的文件。

LWLock:buffer_mapping

此事件表示工作階段正在等待將資料區塊與共用緩衝集區中的緩衝區建立關聯。

> **Note**
>
> 在 Aurora PostgreSQL 第 12 版及更低版本中，此事件顯示為 LWLock:buffer_mapping，在第 13 版及更新版本中，則顯示為 LWLock:BufferMapping。

主題

- 支援的引擎版本
- Context
- 原因
- 動作

支援的引擎版本

此等待事件資訊與 Aurora PostgreSQL 9.6 版及更新版本有關。

Context

共用緩衝集區是一種 Aurora PostgreSQL 記憶體區域，其中保留程序正在使用或已使用的所有分頁。程序需要分頁時會將分頁讀入共用緩衝集區。shared_buffers 參數設定共用緩衝區大小，並保留記憶體區域來存放資料表和索引分頁。如果您變更此參數，請務必重新啟動資料庫。如需詳細資訊，請參閱共用緩衝區。

下列情況會發生 LWLock:buffer_mapping 等待事件：

- 程序在緩衝區資料表中搜尋分頁，並取得共用緩衝區映射鎖定。
- 程序將分頁載入緩衝集區，並取得獨佔緩衝區映射鎖定。
- 程序從集區移除頁面，並取得獨佔緩衝區映射鎖定。

原因

此事件比平時更常出現時，可能表示有效能問題，資料庫正在共用緩衝集區中頁進頁出。典型原因包括：

- 大型查詢
- 膨脹的索引和資料表
- 完整資料表掃描
- 小於工作集的共用集區大小

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 監控緩衝區相關指標
- 評估索引策略
- 減少必須快速配置的緩衝區數目

監控緩衝區相關指標

LWLock:buffer_mapping 等待激增時，請調查緩衝區命中率。您可以使用這些指標，以更加了解緩衝區快取中的情況。檢查下列指標：

BufferCacheHitRatio

此 Amazon CloudWatch 指標測量資料庫叢集中資料庫執行個體的緩衝區快取所處 理的請求百分比。在 LWLock:buffer_mapping 等待事件發生之前，您可能看到此指標下降。

blks_hit

此績效詳情計數器指標指出從共用緩衝集區擷取的區塊數目。在 LWLock:buffer_mapping 等待事件出現之後，您可能發現 blks_hit 激增。

blks_read

此績效詳情計數器指標指出需要將輸入/輸出讀入共用緩衝集區的區塊數目。在 LWLock:buffer_mapping 等待事件發生之前，您可能發現 blks_read 激增。

評估索引策略

若要確認索引策略不會降低效能，請檢查下列各項：

索引膨脹

請確定索引和資料表膨脹不會導致將不必要的分頁讀入共用緩衝區。如果資料表包含未使用的資料列，請考慮封存資料並從資料表中移除資料列。然後，您可以對已調整大小的資料表重建索引。

常用查詢的索引

若要判斷您是否有最佳索引，請在績效詳情中監控資料庫引擎指標。tup_returned 指標顯示讀取的資料列數。tup_fetched 指標顯示傳回給用戶端的資料列數。如果 tup_returned 明顯大於 tup_fetched，表示資料可能未正確編製索引。此外，資料表統計數字可能不是最新。

減少必須快速配置的緩衝區數目

若要減少 LWLock:buffer_mapping 等待事件，請嘗試減少必須快速配置的緩衝區數目。有一種策略是執行較小的批次操作。您可以分割資料表來獲得較小的批次。

LWLock:BufferIO (IPC:BufferIO)

LWLock:BufferIO 事件表示 Aurora PostgreSQL 或 RDS for PostgreSQL 與其他程序同時嘗試存取分頁，正在等待其他程序完成輸入/輸出 (輸入/輸出) 操作。目的是為了將該分頁讀入共用緩衝區。

主題

- 相關的引擎版本

- Context
- 原因
- 動作

相關的引擎版本

此等待事件資訊與所有 Aurora PostgreSQL 版本有關。對於 Aurora PostgreSQL 12 和更早版本，此等待事件命名為 lwlock:buffer_io，而 Aurora PostgreSQL 13 版中，命名為 lwlock:bufferio。從 Aurora PostgreSQL 14 版 BufferIO 等待事件從 `LWLock` 移動至 `IPC` 等待事件類型 (IPC:BufferIO)。

Context

每次必須在共用緩衝集區外擷取區塊 (或分頁) 時，每個共用緩衝區都有與 `LWLock:BufferIO` 等待事件相關聯的輸入/輸出鎖定。

此鎖定用於處理全都需要存取同一個區塊的多個工作階段。必須從 `shared_buffers` 參數定義的共用緩衝集區外讀取此區塊。

在共用緩衝集區內讀取分頁後，就會立刻釋放 `LWLock:BufferIO` 鎖定。

> **Note**
>
> `LWLock:BufferIO` 等待事件在 IO:DataFileRead 等待事件之前發生。從儲存讀取資料時會發生 `IO:DataFileRead` 等待事件。

如需輕量級鎖定的詳細資訊，請參閱鎖定概觀。

原因

`LWLock:BufferIO` 事件出現在最常等待名單中的常見原因包括：

- 多個後端或連線嘗試存取同一個分頁，而此分頁也擱置輸入/輸出操作
- 共用緩衝集區 (由 `shared_buffers` 參數定義) 的大小與目前工作負載所需緩衝區數目之間的比率
- 共用緩衝集區的大小與其他操作移出的分頁數目不太相稱
- 大型或膨脹的索引迫使引擎將過多分頁讀入共用緩衝集區
- 缺少索引迫使資料庫引擎從資料表讀取過多分頁

- 嘗試對同一分頁執行操作的資料庫連線突然激增

動作

我們根據等待事件的原因，建議不同的動作：

- 觀察 Amazon CloudWatch 指標，以瞭解 `BufferCacheHitRatio` 和 `LWLock:BufferIO` 等待事件遽降之間的關聯。這可能表示共用緩衝區設定太小。您可能需要提高此設定，或擴充資料庫執行個體類別的規模。您可以將工作負載分割成更多讀取器節點。
- 驗證是否有未使用的索引，然後移除。
- 使用分割的資料表 (也有分割的索引)。這樣做有助於盡量避免索引重新排序，並減少其影響。
- 避免不必要地編製資料欄的索引。
- 使用連線集區來防止資料庫連線突然激增。
- 在最佳實務上限制資料庫的連線數目上限。

LWLock:lock_manager

此事件表示因為無法執行快速路徑鎖定，Aurora PostgreSQL 引擎維護共用鎖定的記憶體區域來配置、檢查和解除配置鎖定。

主題

- 支援的引擎版本
- Context
- 等待變多的可能原因
- 動作

支援的引擎版本

此等待事件資訊與 Aurora PostgreSQL 9.6 版及更新版本有關。

Context

當您發出 SQL 陳述式時，Aurora PostgreSQL 會記錄鎖定，在並行操作期間保護資料庫的結構、資料和完整性。引擎可以使用快速路徑鎖定或不快速的路徑鎖定來達成此目標。不快速的路徑鎖定比快速路徑鎖定花更多成本，還會產生更多額外負荷。

快速路徑鎖定

如果鎖定經常取得和釋放但很少衝突，為了減少額外負荷，後端程序可以使用快速路徑鎖定。資料庫使用此機制來處理符合下列條件的鎖定：

- 使用 DEFAULT 鎖定方法。
- 代表鎖定資料庫關聯，而不是共用關聯。
- 弱鎖定，不太可能衝突。
- 引擎可以快速確認不可能有衝突鎖定。

有下列任一情況時，引擎無法使用快速路徑鎖定：

- 鎖定不符合上述條件。
- 已無插槽可供後端程序使用。

如需快速路徑鎖定的詳細資訊，請參閱 PostgreSQL 鎖定管理員 README 中的快速路徑和 PostgreSQL 文件中的 pg-locks。

鎖定管理員的擴展問題範例

在此範例中，名為 `purchases` 的資料表存放五年的資料，並按日分割。每個分割區都有兩個索引。發生下列事件序列：

1. 您查詢很多天的資料，使得資料庫需要讀取許多分割區。
2. 資料庫為每個分割區建立鎖定項目。如果分割區索引出現在最佳化工具存取路徑中，則資料庫也為這種索引建立鎖定項目。
3. 對同一個後端程序所請求的鎖定項目數大於 16 時，即 `FP_LOCK_SLOTS_PER_BACKEND` 的值，鎖定管理員會使用不快速的路徑鎖定方法。

現代化應用程式可能有數百個工作階段。如果並行工作階段查詢父項，但沒有適當的分割區剔除，則資料庫可能會建立數百甚至數千個不快速的路徑鎖定。當此並行高於 vCPU 數目時，通常會出現 `LWLock:lock_manager` 等待事件。

> **Note**
>
> LWLock:lock_manager 等待事件與資料庫結構描述中的分割區或索引數目無關。但與資料庫必須控制的不快速路徑鎖定數目有關。

等待變多的可能原因

LWLock:lock_manager 等待事件比平常更常發生時，可能表示有效能問題，突然激增最可能的原因如下：

- 並行作用中工作階段正在執行的查詢未使用快速路徑鎖定。這些工作階段也超過 vCPU 上限。
- 大量並行作用中工作階段正在存取高度分割的資料表。每個分割區有多個索引。
- 資料庫遇到連線風暴。根據預設，當資料庫變慢時，某些應用程式和連線集區軟體會建立更多連線。這種做法使問題變得更糟。請調校連線集區軟體，以免發生連線風暴。
- 大量工作階段查詢父資料表但未剔除分割區。
- 資料定義語言 (DDL)、資料處理語言 (DML) 或維護命令獨佔鎖定忙碌關聯，或是經常存取或修改的元組。

動作

根據等待事件的原因，我們會建議不同的動作。

主題

- 使用分割區剔除
- 移除不必要的索引
- 調校查詢以使用快速路徑鎖定
- 調校其他等待事件
- 降低硬體瓶頸
- 使用連線集區
- 升級 Aurora PostgreSQL 版本

使用分割區剔除

剔除分割區是一種查詢最佳化策略，可將不需要的分割區排除在資料表掃描外，進而改善效能。分割區剔除預設為啟用。如果已停用，請如下啟用。

```
SET enable_partition_pruning = on;
```

當 WHERE 子句包含用於分割的資料欄時，查詢可以利用分割區剔除。如需詳細資訊，請參閱 PostgreSQL 文件中的分割區剔除。

移除不必要的索引

資料庫可能包含未使用或很少使用的索引。若是如此，請考慮刪除這些索引。執行下列任何一項：

- 請參閱 PostgreSQL Wiki 中的未使用的索引，以了解如何尋找不必要的索引。
- 執行 PG Collector。此 SQL 指令碼會收集資料庫資訊，並顯示在合併的 HTML 報告中。請檢查「Unused indexes (未使用的索引)」區段。如需詳細資訊，請參閱 AWS Labs GitHub 儲存庫中的 pg-collector。

調校查詢以使用快速路徑鎖定

若要查明查詢是否使用快速路徑鎖定，請查詢 pg_locks 資料表的 fastpath 資料欄。如果查詢未使用快速路徑鎖定，請嘗試將每個查詢的關聯數量減少到 16 以下。

調校其他等待事件

如果 LWLock:lock_manager 在最常等待名單中排行前兩名，請檢查下列等待事件是否也出現在名單中：

- Lock:Relation
- Lock:transactionid
- Lock:tuple

如果上述事件在名單中排行前幾名，請考慮先調校這些等待事件。這些事件可能引發 LWLock:lock_manager。

降低硬體瓶頸

您可能遇到硬體瓶頸，例如 CPU 不足或 Amazon EBS 頻寬耗盡。在這些情況下，請考慮降低硬體瓶頸。考慮下列動作：

- 擴充執行個體類別的規模。
- 將耗用大量 CPU 和記憶體的查詢最佳化。
- 變更應用程式邏輯。
- 封存資料。

如需 CPU、記憶體和 EBS 網路頻寬的詳細資訊，請參閱 Amazon RDS 執行個體類型。

使用連線集區

如果作用中連線總數超過 vCPU 上限，表示需要 CPU 的作業系統程序超過執行個體類型可支援的數量。在此情況下，請考慮使用或調校連線集區。關於執行個體類型的 vCPU，如需詳細資訊，請參閱 Amazon RDS 執行個體類型。

如需連線集區的詳細資訊，請參閱下列資源：

- Amazon RDS Proxy for Aurora
- pgbouncer
- 《PostgreSQL 文件》中的連線集區和資料來源

升級 Aurora PostgreSQL 版本

如果您目前的 Aurora PostgreSQL 版本低於 12，請升級至第 12 版或更新版本。PostgreSQL 版本 12 和 13 已改善分割機制。如需第 12 版的詳細資訊，請參閱 PostgreSQL 12.0 版本備註。如需升級 Aurora PostgreSQL 的詳細資訊，請參閱 Amazon Aurora PostgreSQL 的資料庫引擎更新。

LWLock:MultiXact

`LWLock:MultiXactMemberBuffer`、`LWLock:MultiXactOffsetBuffer`、`LWLock:MultiXactMembe` 和 `LWLock:MultiXactOffsetSLRU` 等待事件表示工作階段正在等待擷取所指定資料表中修改相同資料列的交易清單。

- `LWLock:MultiXactMemberBuffer` – 程序正在等待簡單的最近最少使用 (SLRU) 緩衝區上的輸入/輸出，以取得 multixact 成員。
- `LWLock:MultiXactMemberSLRU` – 程序正在等待存取簡單的最近最少使用 (SLRU) 快取，以取得 multixact 成員。
- `LWLock:MultiXactOffsetBuffer` – 程序正在等待簡單的最近最少使用 (SLRU) 緩衝區上的輸入/輸出，以取得 multixact 位移。

- `LWLock:MultiXactOffsetSLRU` – 程序正在等待存取簡單的最近最少使用 (SLRU) 快取上，以取得 multixact 位移。

主題

- 支援的引擎版本
- Context
- 等待時間增加的可能原因
- 動作

支援的引擎版本

所有版本的 Aurora PostgreSQL 都支援此等待事件資訊。

Context

multixact 是資料結構，其中存放一份清單，列出修改相同資料表資料列的交易 ID (XID)。當單一交易參考資料表中的資料列時，交易 ID 會存放在資料表標題列中。當多個交易參考資料表中的同一資料列時，交易 ID 清單會存放在 multixact 資料結構中。multixact 等待事件表示工作階段正在從資料結構中擷取參照資料表中指定資料列的交易清單。

等待時間增加的可能原因

使用 multixact 的三個常見原因如下：

- 來自明確儲存點的子交易 – 在交易中明確建立儲存點會為相同資料列產生新的交易。例如，依序使用 `SELECT FOR UPDATE`、`SAVEPOINT`、`UPDATE`。

  某些驅動程式、物件關聯式映射器 (ORM) 和抽象層具有組態選項，用於使用儲存點自動包裝所有操作。這可能會在某些工作負載中產生許多 multixact 等待事件。PostgreSQL JDBC 驅動程式的 `autosave` 選項就是此情況的範例。如需詳細資訊，請參閱 PostgreSQL JDBC 文件中的 pgJDBC。另一個範例是 PostgreSQL ODBC 驅動程式及其 `protocol` 選項。如需詳細資訊，請參閱 PostgreSQL ODBC 驅動程式文件中的 psqlODBC Configuration Options (psqlODBC 組態選項)。
- 來自 PL/pgSQL EXCEPTION 子句的子交易 – 您在 PL/pgSQL 函數或程序中寫入的每個 `EXCEPTION` 子句都會在 `SAVEPOINT` 內部建立 。
- 外部索引鍵 – 多個交易在上層記錄上取得共用鎖定。

當指定資料列包含在多重交易操作中時，處理資料列需要從 `multixact` 清單中擷取交易 ID。如果查詢無法從記憶體快取中取得 multixact，則必須從 Aurora 儲存層讀取資料結構。這個來自儲存體的 I/O 表示查詢可能需要較長的時間。由於大量的多重交易，記憶體快取遺漏可能會在大量使用情況下開始發生。所有這些因素都會導致此等待事件的增加。

動作

根據等待事件的原因，我們會建議不同的動作。其中一些動作有助於立即減少等待事件。但是，其他人可能需要調查和修正才能擴展工作負載。

主題

- 使用此等待事件對資料表執行清空凍結
- 使用此等待事件增加資料表的自動清空頻率
- 增加記憶體參數
- 減少長時間執行的交易
- 長期動作

使用此等待事件對資料表執行清空凍結

如果此等待事件突然激增並影響您的生產環境，您可以使用下列任何暫時方法來減少其計數。

- 在受影響的資料表或資料表分割區上使用 VACUUM FREEZE，以立即解決問題。如需詳細資訊，請參閱 VACUUM。
- 使用 VACUUM (FREEZE、INDEX_CLEANUP FALSE) 子句略過索引來執行快速清空。如需詳細資訊，請參閱盡快清空資料表。

使用此等待事件增加資料表的自動清空頻率

掃描所有資料庫中的所有資料表後，VACUUM 最終會移除 multixact，而其最舊的 multixact 值會向前。如需詳細資訊，請參閱 Multixacts 和 Wraparound。若要將 LWLock：MultiXact 等待事件維持在其最小值，您必須視需要執行 VACUUM。若要這麼做，請確定 Aurora PostgreSQL 資料庫叢集中的 VACUUM 已進行最佳設定。

如果在受影響的資料表或資料表分割區上使用 VACUUM FREEZE 解決等待事件問題，我們建議您使用排程器，例如 `pg_cron` 來執行 VACUUM，而不是在執行個體層級調整自動清空。

若要讓自動清空更頻繁地發生，您可以減少受影響資料表中儲存參數 `autovacuum_multixact_freeze_max_age` 的值。如需詳細資訊，請參閱 autovacuum_multixact_freeze_max_age。

增加記憶體參數

您可以調整下列參數，以最佳化 multixact 快取的記憶體用量。這些設定會控制為這些快取預留多少記憶體，這有助於減少工作負載中的 multixact 等待事件。我們建議您從下列值開始：

對於 Aurora PostgreSQL 17 和更新版本：

- `multixact_offset_buffers` = 128
- `multixact_member_buffers` = 256

對於 Aurora PostgreSQL 16 和更早版本：

- `multixact_offsets_cache_size` = 128
- `multixact_members_cache_size` = 256

> **Note**
>
> 在 Aurora PostgreSQL 17 中，參數名稱從 `multixact_offsets_cache_size` 變更為 `multixact_offset_buffers`，以及從 `multixact_members_cache_size` 變更為 `multixact_member_buffers`，以符合社群 PostgreSQL 17。

您可以在叢集層級設定這些參數，讓叢集中的所有執行個體保持一致。我們建議您測試和調整這些值，以達到最適合您的特定工作負載需求和執行個體類別。為了使參數變更生效，您必須重新啟動寫入器執行個體。

這些參數會以 multixact 快取項目表示。每個快取項目會使用 8 KB 記憶體。若要計算保留的總記憶體，請將每個參數值乘以 8 KB。例如，如果您將參數設定為 128，則預留記憶體總數將為 128 * 8 KB = 1 MB。

減少長時間執行的交易

長時間執行的交易會導致清空程序保留其資訊，直到認可交易或關閉唯讀交易為止。建議您主動監控和管理長時間執行的交易。如需更多詳細資訊，請參閱 資料庫在交易連線中長時間閒置。請嘗試修改您的應用程式，以避免使用或儘量不使用長時間執行的交易。

長期動作

檢查您的工作負載，以探索 multixact 溢出的原因。您必須修正問題，才能擴展工作負載並減少等待事件。

- 您必須分析用來建立資料表的 DDL (資料定義語言)。請確定資料表結構和索引的設計良好。
- 當受影響的資料表具有外部索引鍵時，請判斷是否需要這些索引鍵，或是否有其他方式可強制執行參考完整性。
- 當資料表有大型未使用的索引時，可能會導致自動清空不符合您的工作負載，並可能使其無法執行。若要避免這種情況，請檢查未使用的索引並將其完全移除。如需詳細資訊，請參閱使用大型索引管理自動清空。
- 減少在交易中使用儲存點。

LWLock:pg_stat_statements

當 `pg_stat_statements` 延伸對追蹤 SQL 陳述式的雜湊資料表進行獨佔鎖定時，就會發生 LWLock:pg_stat_statements 等待事件。這會發生在下列案例中：

- 當追蹤的陳述式數量達到設定的 `pg_stat_statements.max` 參數值，而且需要為更多項目騰出空間時，延伸會對呼叫數量執行排序、移除 5% 的最少執行的陳述式，並使用剩餘的項目重新填入雜湊。
- 當 `pg_stat_statements` 對磁碟上的 `pgss_query_texts.stat` 檔案執行 `garbage collection` 操作，並重新寫入檔案。

主題

- 支援的引擎版本
- Context
- 等待時間增加的可能原因
- 動作

支援的引擎版本

所有 Aurora PostgreSQL 的版本都支援此等待事件資訊。

Context

了解 pg_stat_statements 延伸 – pg_stat_statements 延伸會追蹤雜湊資料表中的 SQL 陳述式執行統計資料。延伸會追蹤 SQL 陳述式，最高可達 `pg_stat_statements.max` 參數定義的限制。此參數會決定可追蹤的陳述式數目上限，對應至 pg_stat_statements 檢視中的資料列數目上限。

陳述式統計資料持久性 – 延伸會保留執行個體重新啟動時的陳述式統計資料：

- 將資料寫入名為 pg_stat_statements.stat 的檔案
- 使用 pg_stat_statements.save 參數控制持久性行為

當 pg_stat_statements.save 設定為：

- ON (預設)：統計資料會在關閉時儲存，並在伺服器啟動時重新載入
- OFF：統計資料不會在關機時儲存，也不會在伺服器啟動時重新載入

查詢文字儲存 – 延伸會將追蹤查詢的文字儲存在名為 `pgss_query_texts.stat` 的檔案中。在垃圾回收發生之前，此檔案可以成長為所有追蹤 SQL 陳述式的平均大小的兩倍。在清理操作和重寫 `pgss_query_texts.stat` 檔案期間，延伸需要對雜湊資料表進行獨佔鎖定。

陳述式解除配置程序 – 當追蹤的陳述式數量達到 `pg_stat_statements.max` 限制且需要追蹤新的陳述式時，延伸會：

- 在雜湊資料表上進行獨佔鎖定 (LWLock:pg_stat_statements)。
- 將現有資料載入本機記憶體。
- 根據呼叫次數執行快速排序。
- 移除最少呼叫的陳述式 (底部 5%)。
- 將剩餘項目重新填入雜湊資料表。

監控陳述式解除配置 – 在 PostgreSQL 14 和更新版本中，您可以使用 pg_stat_statements_info 檢視來監控陳述式解除配置。此檢視包含解除配置欄，顯示陳述式被解除配置的次數，以便為新陳述式騰出空間

如果陳述式的解除配置頻繁發生，將導致磁碟上 `pgss_query_texts.stat` 檔案的垃圾回收更頻繁。

等待時間增加的可能原因

增加 LWLock:pg_stat_statements 等待的典型原因包括：

- 應用程式使用的唯一查詢數量增加。
- 與使用的唯一查詢數目相比，pg_stat_statements.max 參數值很小。

動作

根據等待事件的原因，我們會建議不同的動作。您可以使用 Amazon RDS Performance Insights 或查詢檢視 pg_stat_activity 來識別 LWLock:pg_stat_statements 事件。

調整下列 pg_stat_statements 參數以控制追蹤行為，並減少 LWLock:pg_stat_ 陳述式等待事件。

主題

- 停用 pg_stat_statements.track 參數
- 增加 pg_stat_statements.max 參數
- 停用 pg_stat_statements.track_utility 參數

停用 pg_stat_statements.track 參數

如果 LWLock:pg_stat_statements 等待事件對資料庫效能造成負面影響，且在進一步分析 pg_stat_statements 檢視以找出根本原因之前需要快速解決方案，則可以將其設定為 none 來停用 pg_stat_statements.track 參數。這將停用陳述式統計資料的集合。

增加 pg_stat_statements.max 參數

若要減少解除配置並將磁碟上 pgss_query_texts.stat 檔案的垃圾回收降至最低，請增加 pg_stat_statements.max 參數的值。預設值為 5,000。

> **Note**
>
> pg_stat_statements.max 參數設為靜態。您必須重新啟動資料庫執行個體，才能將任何變更套用至此參數。

停用 pg_stat_statements.track_utility 參數

您可以分析 pg_stat_statements 檢視，以判斷哪些公用程式命令耗用 pg_stat_statements 追蹤最多的資源。

pg_stat_statements.track_utility 參數可控制模組是否追蹤公用程式命令，其中包含 SELECT、INSERT、UPDATE、DELETE 和 MERGE 以外的所有命令。根據預設，此參數會設定為 on。

例如，當您的應用程式使用許多原本是唯一的儲存點查詢時，可能會增加陳述式解除配置。若要解決此問題，您可以停用 pg_stat_statements.track_utility 參數以防止 pg_stat_statements 追蹤儲存點查詢。

> **Note**
>
> pg_stat_statements.track_utility 參數是動態參數。您可以變更其值，而無需重新啟動資料庫執行個體。

Example pg_stat_statements 中唯一儲存點查詢的範例

```
                    query                         |       queryid
--------------------------------------------------+---------------------
 SAVEPOINT JDBC_SAVEPOINT_495701                  | -7249565344517699703
 SAVEPOINT JDBC_SAVEPOINT_1320                    | -1572997038849006629
 SAVEPOINT JDBC_SAVEPOINT_26739                   |  54791337410474486
 SAVEPOINT JDBC_SAVEPOINT_1294466                 |  8170064357463507593
 ROLLBACK TO SAVEPOINT JDBC_SAVEPOINT_65016       | -33608214779996400
 SAVEPOINT JDBC_SAVEPOINT_14185                   | -2175035613806809562
 SAVEPOINT JDBC_SAVEPOINT_45837                   | -6201592986750645383
 SAVEPOINT JDBC_SAVEPOINT_1324                    |  6388797791882029332
```

PostgreSQL 17 為公用程式命令追蹤推出數種增強功能：

- 儲存點名稱現在會顯示為常數。
- 兩階段遞交命令的全域交易 IDs(GIDs) 現在會顯示為常數。
- DEALLOCATE 陳述式的名稱會顯示為常數。
- CALL 參數現在會顯示為常數。

Timeout:PgSleep

`Timeout:PgSleep` 事件表示伺服器程序已呼叫 `pg_sleep` 函數，正在等待睡眠逾時到期。

主題

- 支援的引擎版本
- 等待變多的可能原因
- 動作

支援的引擎版本

所有版本的 Aurora PostgreSQL 都支援此等待事件資訊。

等待變多的可能原因

此等待事件表示應用程式、預存函數或使用者發出的 SQL 陳述式呼叫下列其中一一個函數：

- `pg_sleep`
- `pg_sleep_for`
- `pg_sleep_until`

上述函數會延遲執行，直到經過指定的秒數。例如，`SELECT pg_sleep(1)` 會暫停 1 秒。如需詳細資訊，請參閱 PostgreSQL 文件中的延遲執行。

動作

找出執行 `pg_sleep` 函數的陳述式。判斷使用此函數是否適當。

使用 Amazon DevOps Guru 主動洞察，調校 Aurora PostgreSQL

DevOps Guru 主動洞察會偵測可能在 Aurora PostgreSQL 資料庫叢集上造成問題的狀況，並在發生問題前即讓您了解狀況。主動式洞察可以提醒您交易連線中長時間執行的閒置。如需在交易連線中對長時間執行閒置進行疑難排解的詳細資訊，請參閱 資料庫在交易連線中長時間閒置

DevOps Guru 可以執行下列動作：

- 透過交叉檢查一般建議設定與您的資料庫設定，避免許多常見的資料庫問題。
- 警告您機群內的重大問題，若未勾選，可能導致更嚴重的問題。

- 提醒您新發現的問題。

每個主動洞察都包含問題原因分析和修正動作建議。

如需 Amazon DevOps Guru for Amazon RDS 的詳細資訊，請參閱 使用 Amazon DevOps Guru for Amazon RDS 分析 Aurora 效能異常。

資料庫在交易連線中長時間閒置

資料庫的連線已經超過 1800 秒都處在 `idle in transaction` 狀態。

主題

- 支援的引擎版本
- Context
- 造成此問題的可能原因
- 動作
- 相關指標

支援的引擎版本

所有版本的 Aurora PostgreSQL 皆支援此洞察資訊。

Context

`idle in transaction` 狀態的交易可以擁有封鎖其他查詢的鎖定。也可以防止 `VACUUM` (包含自動清空) 清理無效資料列，導致索引或資料表膨脹，或導致交易 ID 包圍。

造成此問題的可能原因

尚未使用 COMMIT、ROLLBACK 或 END 命令，關閉以 BEGIN 或 START TRANSACTION 在互動式工作階段中啟動的交易。這會導致交易移至 `idle in transaction` 狀態。

動作

您可以透過查詢 `pg_stat_activity`，找出閒置的交易。

請在您的 SQL 用戶端中執行下列查詢，以列出 `idle in transaction` 狀態的所有連線，並按持續時間排序：

```
SELECT now() - state_change as idle_in_transaction_duration, now() - xact_start as
 xact_duration,*
FROM  pg_stat_activity
WHERE state  = 'idle in transaction'
AND   xact_start is not null
ORDER BY 1 DESC;
```

根據洞察的原因，我們會建議不同的動作。

主題

- End 交易
- 終止連線
- 設定 idle_in_transaction_session_timeout 參數
- 檢查 AUTOCOMMIT 狀態
- 檢查應用程式程式碼中的交易邏輯

End 交易

使用 BEGIN 或 START TRANSACTION 在互動式工作階段中啟動交易時，該筆交易會移至 `idle in transaction` 狀態。交易會保持在此狀態，直到您發出 COMMIT、ROLLBACK、END 命令結束交易，或完全斷開連線以轉返結束交易。

終止連線

使用以下查詢，終止與閒置交易的連線：

```
SELECT pg_terminate_backend(pid);
```

pid 是連線的程序 ID。

設定 idle_in_transaction_session_timeout 參數

在新的參數群組中設定 `idle_in_transaction_session_timeout` 參數。設定此參數的優點在於，不需要手動介入即可終止長時間閒置的交易。如需此參數的詳細資訊，請參閱 PostgreSQL 文件。

當交易處於 idle_in_transaction 狀態超過指定時間時，PostgreSQL 日誌檔會在連線終止之後報告下列訊息。

```
FATAL: terminating connection due to idle in transaction timeout
```

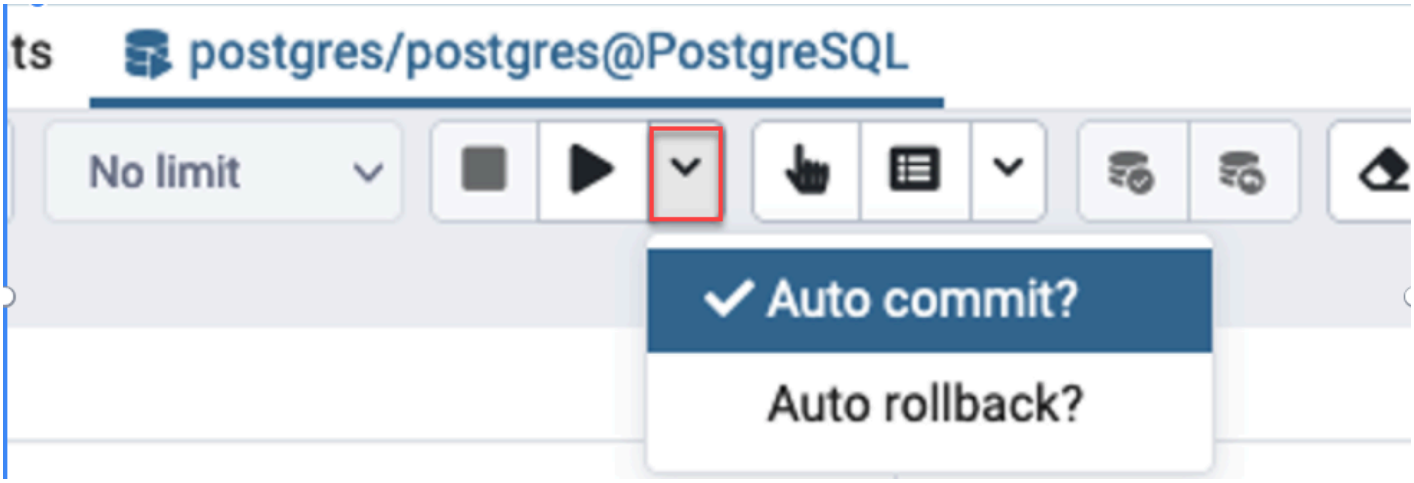
檢查 AUTOCOMMIT 狀態

根據預設，AUTOCOMMIT 為啟用狀態。但是，若客戶端意外將其關閉，請確認重啟。

- 在 psql 用戶端執行下列命令：

```
postgres=> \set AUTOCOMMIT on
```

- 在 pgadmin 中，從向下箭頭選擇 AUTOCOMMIT 選項以將其開啟。



檢查應用程式程式碼中的交易邏輯

調查應用程式邏輯，找出可能的問題。考慮下列動作：

- 檢查 JDBC 自動遞交是否在您的應用程式中設為 true。另外，請考慮在程式碼中使用明確的 COMMIT 命令。
- 檢查錯誤處理邏輯，確認其是否會在發生錯誤後關閉交易。
- 檢查交易開啟時，您的應用程式是否需要長時間處理查詢傳回的資料列。若是如此，請考慮對應用程式進行編碼，在處理資料列之前關閉交易。
- 檢查交易是否包含許多長時間執行的操作。若是如此，請將單一交易分割為多筆交易。

相關指標

下列 PI 指標與此洞察相關：

- idle_in_transaction_count - 處於 `idle in transaction` 狀態的工作階段數量。
- idle_in_transaction_max_time - 處於 `idle in transaction` 狀態的最長時間執行交易。

Amazon Aurora PostgreSQL 的最佳實務

以下提供幾種管理 Amazon Aurora PostgreSQL 資料庫叢集的最佳實務。請務必同時檢閱基本維護工作。如需詳細資訊，請參閱Amazon Aurora PostgreSQL 的效能和擴展。

主題

- 避免 Aurora PostgreSQL 資料庫執行個體效能變慢、自動重新啟動和容錯移轉
- 診斷資料表和索引膨脹
- 在 Amazon Aurora PostgreSQL 中管理高物件計數
- Aurora PostgreSQL 中的記憶體管理已改善
- Amazon Aurora PostgreSQL 的快速容錯移轉
- Aurora PostgreSQL 的容錯移轉後使用叢集快取管理快速復原
- 使用集區管理 Aurora PostgreSQL 連線流失
- PostgreSQL 中的無效連線處理
- 調整 Aurora PostgreSQL 的記憶體參數
- 使用 Amazon CloudWatch 指標來分析 Aurora PostgreSQL 的資源用量
- 使用邏輯複寫來執行 Aurora PostgreSQL 的主要版本升級
- 在 Aurora PostgreSQL 中管理自訂轉換
- Aurora PostgreSQL 中平行查詢的最佳實務
- 針對 Aurora PostgreSQL 中的儲存問題進行故障診斷

避免 Aurora PostgreSQL 資料庫執行個體效能變慢、自動重新啟動和容錯移轉

如果您正在執行繁重的工作負載或工作負載超出資料庫執行個體配置的資源，則可能會在執行應用程式和 Aurora 資料庫時耗盡資源。若要取得資料庫執行個體的指標 (例如 CPU 使用率、記憶體使用量和使用的資料庫連線數目)，您可以參考 Amazon CloudWatch 提供的指標、Performance Insights 和增強型監控。如需如何監控資料庫執行個體的詳細資訊，請參閱 在 Amazon Aurora 叢集中監控指標。

如果您的工作負載耗盡了您正在使用的資源，您的資料庫執行個體可能會變慢、重新啟動，甚至容錯移轉到其他資料庫執行個體。若要避免這種情況，請監控資源使用率、檢查資料庫執行個體上執行的工作負載，並在必要時進行最佳化。如果最佳化無法改善執行個體指標並減緩資源耗盡，請考慮在達到其限制之前縱向擴展資料庫執行個體。如需可用資料庫執行個體類別及其規格的詳細資訊，請參閱 Amazon Aurora 資料庫執行個體類別。

診斷資料表和索引膨脹

您可以使用 PostgreSQL 多版本並行控制 (MVCC) 來協助維護資料的完整性。PostgreSQL MVCC 的運作方式是儲存更新或刪除的資料列 (元組) 內部複本，直到交易提交或復原。使用者看不到這份儲存的內部複本。但是，若 VACUUM 或 AUTOVACUUM 公用程式未定期清理這些隱藏複本，便可能發生資料表膨脹。不選取，資料表膨脹可能會增加儲存成本並降低處理速度。

在許多情況下，Aurora 上 VACUUM 或 AUTOVACUUM 的預設設定足夠處理不需要的資料表膨脹。但是，若您的應用程式遇到以下情況，可能需要檢查膨脹情形：

- 在 VACUUM 程序間的較短時間內處理大量交易。
- 執行效果不佳且儲存空間不足。

若要開始使用，請取得失效元組使用空間量的準確資訊，以及您能透過清理資料表和索引膨脹來復原的空間量。若要這麼做，請使用 `pgstattuple` 擴充功能來取得 Aurora 叢集上的統計資料。如需更多詳細資訊，請參閱 pgstattuple。只有 `pg_stat_scan_tables` 角色和資料庫超級使用者能使用 `pgstattuple` 擴充功能。

若要在 Aurora 上建立 `pgstattuple` 擴充功能，請將用戶端工作階段連線到叢集，例如 psql 或 pgAdmin，然後使用下列命令：

```
CREATE EXTENSION pgstattuple;
```

在您要設定的每個資料庫中建立擴充功能。建立擴充功能之後，請使用命令列界面 (CLI) 來計算您可以收回多少無法使用的空間。在取得統計資料之前，請先將 AUTOVACTURE 設為 0，以修改叢集參數群組。將該值設為 0 可防止 Aurora 自動清除應用程式留下的任何失效元組，進而影響結果準確性。輸入下列命令建立簡易資料表：

```
postgres=> CREATE TABLE lab AS SELECT generate_series (0,100000);
SELECT 100001
```

在下方範例中，我們在啟用 AUTOVACUUM 的情況下執行資料庫叢集查詢。dead_tuple_count 為 0，表示 AUTOVACUUM 已從 PostgreSQL 資料庫刪除過時的資料或元組。

若要以 pgstattuple 來取得資料表相關資訊，請在查詢中指定資料表名稱或物件識別碼 (OID)：

```
postgres=> SELECT * FROM pgstattuple('lab');
```

```
table_len  | tuple_count | tuple_len | tuple_percent | dead_tuple_count |
 dead_tuple_len | dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+----------------+--------------------+------------+--------------
3629056    | 100001      | 2800028   | 77.16         | 0                | 0
  | 0                  | 16616      | 0.46
(1 row)
```

在下方的查詢中，我們關閉 AUTOVACUUM 並輸入命令，從資料表中刪除 25,000 資料列。結果是，dead_tuple_count 增加到 25000。

```
postgres=> DELETE FROM lab WHERE generate_series < 25000;

DELETE 25000
```

```
SELECT * FROM pgstattuple('lab');
```

```
table_len | tuple_count | tuple_len | tuple_percent | dead_tuple_count | dead_tuple_len
 | dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+----------------+--------------------+------------+--------------
3629056 | 75001 | 2100028 | 57.87 | 25000 | 700000 | 19.29 | 16616 | 0.46
(1 row)
```

若要回收這些無效元組，請啟動 VACUUM 程序。

在不中斷應用程式的情況下觀察膨脹情形

Aurora 叢集上的設定經過最佳化，可為多數工作負載提供最佳實務。不過，您可能想最佳化叢集，以更符合您的應用程式和使用模式。在此情況下，您可以使用 pgstattuple 擴充功能，且不會中斷忙碌的應用程式。若要這麼做，請執行下列步驟：

1. 複製您的 Aurora 執行個體。
2. 修改參數檔案，以關閉複製中的 AUTOVACUUM。
3. 執行 pgstattuple 查詢時以範例工作負載或 pgbench 測試複製，pgbench 是在 PostgreSQL 上執行基準測試的程式。如需更多詳細資訊，請參閱 pgbench。

執行應用程式並檢視結果之後，請在還原的複本上使用 pg_repack 或 VACUUM FULL 並比較差異。若 dead_tuple_count、dead_tuple_len 或 dead_tuple_percent 的值顯著下降，請調整生產叢集上的真空排程，盡量減少膨脹。

避免暫存資料表膨脹

若您的應用程式要建立暫存資料表，請確認應用程式會在不需要暫存資料表後進行移除。自動清空程序找不到暫存資料表。保持不選取，暫存資料表可以快速建立資料庫膨脹。此外，膨脹可以擴展到系統資料表，也就是追蹤 PostgreSQL 物件和屬性的內部資料表，例如 pg_attribute 和 pg_depend。

當不再需要暫存資料表時，您可以使用 TRUNCATE 陳述式來清空資料表並釋放空間。然後，手動清空 pg_attribute 和 pg_depend 資料表。清空這些資料表，確保不斷建立和截斷/刪除暫存資料表時，不會新增元組並導致系統膨脹。

您可以在建立暫存資料表時避免這個問題，方法是加入下列語法，在提交內容時刪除新資料列：

```
CREATE TEMP TABLE IF NOT EXISTS table_name(table_description) ON COMMIT DELETE ROWS;
```

遞交交易時，ON COMMIT DELETE ROWS 子句會截斷暫存資料表。

避免索引膨脹

變更資料表的索引欄位時，索引更新會導致該索引出現一或多個失效元組。根據預設，自動清空程序會清除索引中的膨脹，但是該清理程序會佔用大量時間和資源。若要在建立資料表時指定索引清理偏好

設定，請包括 vacuum_index_cleanup 子句。根據預設，建立資料表時，子句會設為 AUTO，表示伺服器會決定索引在清空資料表時是否需要進行清除。您可以將子句設為 ON 以開啟特定資料表的索引清除，或設為 OFF 關閉該資料表的索引清除。請留意，關閉索引清除可能會節省時間，但同時也可能導致索引膨脹。

清空資料表時，您可以在命令列中手動控制索引清除。若要清空資料表並從索引中刪除失效元組，請包括值設為 ON 的 INDEX_CLEANUP 子句和資料表名稱：

```
acctg=> VACUUM (INDEX_CLEANUP ON) receivables;

INFO: aggressively vacuuming "public.receivables"
VACUUM
```

若要在不清除索引的情況下清空資料表，請將值設為 OFF：

```
acctg=> VACUUM (INDEX_CLEANUP OFF) receivables;

INFO: aggressively vacuuming "public.receivables"
VACUUM
```

在 Amazon Aurora PostgreSQL 中管理高物件計數

雖然 PostgreSQL 限制是理論上的，但資料庫中具有極高的物件計數會對各種操作造成顯著的效能影響。本文件涵蓋數種常見的物件類型，當總計數過高時，可能會導致數種影響。

下表提供物件類型及其潛在影響的摘要：

物件類型和潛在影響

物件的類型	自動清空	邏輯複寫	主要版本升級	pg_dump / pg_restore	一般效能	執行個體重新啟動
關係	x		x	x	x	
暫時資料表	x				x	
未記錄的資料表		x				x
分區					x	

物件的類型	自動清空	邏輯複寫	主要版本升級	pg_dump / pg_restore	一般效能	執行個體重新啟動
暫存檔案					x	
序列		x				
大型物件		x	x			

關係

PostgreSQL 資料庫中的資料表數量沒有特定的硬性限制。理論限制非常高，但在資料庫設計期間需要記住其他實際限制。

影響：自動清空落後

與工作量相比，自動清空可能會難以跟上交易 ID 成長或資料表膨脹的情況。

建議的動作：調整自動清空有幾個因素，可以正確跟上指定數量的資料表和指定工作負載。如需如何判斷適當自動清空設定的建議 的最佳實務使用 PostgreSQL autovacuum 的最佳實務。使用 postgres_get_av_diag 公用程式來監控交易 ID 成長的問題。

影響：主要版本升級/pg_dump 和還原

Amazon RDS 在 pg_upgrade 執行期間使用「--link」選項，以避免必須複製資料檔案，結構描述中繼資料仍需要還原至新版本的資料庫。即使使用平行 pg_restore，如果有大量關聯，這也會增加停機時間。

影響：一般效能降低

由於目錄大小而降低的一般效能。每個資料表及其相關聯的資料欄都會新增至 `pg_attribute`，`pg_class`以及常用於一般資料庫操作的`pg_depend`資料表。不會顯示特定的等待事件，但共用緩衝區效率會受到影響。

建議的動作：定期檢查這些特定資料表的資料表膨脹，並偶爾在這些特定資料表`VACUUM FULL`上執行 。請注意，在目錄資料表`VACUUM FULL`上需要`ACCESS EXCLUSIVE`鎖定，這表示在操作完成之前，其他查詢都無法存取它們。

大約閾值：百萬

暫時資料表

使用暫存資料表對於測試資料或中繼結果很有用，而且是許多資料庫引擎中常見的模式。必須了解 PostgreSQL 中大量使用的影響，以避免某些陷阱。每個暫存資料表的建立和捨棄都會將資料列新增至系統目錄資料表，當資料表膨脹時，會導致一般效能問題。

影響：自動清空落後

暫時資料表不會由自動清空清空，但會在交易 IDs存在期間保留，若未移除，可能會導致包裝。

建議的動作：暫時資料表會在建立它們的工作階段期間持續運作，或可以手動捨棄。避免使用暫存資料表長時間執行交易的最佳實務，可防止這些資料表產生最大使用的交易 ID 成長。

影響：一般效能降低

由於目錄大小而降低的一般效能。當工作階段持續建立和捨棄暫存資料表時，它會新增至 `pg_attribute`，`pg_class`以及常用於一般資料庫操作的`pg_depend`資料表。不會顯示特定的等待事件，但共用緩衝區效率會受到影響。

建議的動作：

- 定期檢查這些特定資料表的資料表膨脹，並偶爾在這些特定資料表`VACUUM FULL`上執行 。請注意，在目錄資料表`VACUUM FULL`上需要`ACCESS EXCLUSIVE`鎖定，這表示在操作完成之前，其他查詢都無法存取它們。
- 如果大量使用暫存資料表，在主要版本升級之前，強烈建議使用這些特定目錄資料表`VACUUM FULL`的 ，以減少停機時間。

一般最佳實務：

- 使用常見的資料表表達式來產生中繼結果，以減少暫時資料表的使用。這些有時可能會使所需的查詢複雜化，但會消除上述影響。
- 使用 `TRUNCATE`命令來清除內容，而不是執行捨棄/建立步驟，以重複使用暫存資料表。這也會消除暫時資料表造成交易 ID 成長的問題。

大約閾值：數十萬

未記錄的資料表

未記錄的資料表可以提供效能提升，因為它們不會產生任何 WAL 資訊。必須仔細使用它們，因為它們在資料庫損毀復原期間不會提供任何耐用性，因為它們將被截斷。這是 PostgreSQL 中的昂貴操作，因

為每個未記錄的資料表都是序列截斷的。雖然對少量未記錄的資料表而言，此操作速度很快，但當它們以千為單位編號時，它可以開始在啟動期間新增顯著的延遲。

影響：邏輯複寫

邏輯複寫通常不包含未記錄的資料表，包括藍/綠部署，因為邏輯複寫依賴 WAL 來擷取和傳輸變更。閱讀如何設定 Aurora PostgreSQL 以複寫未記錄資料表的詳細資訊。

影響：讀取器節點

您只能從 Aurora 資料庫叢集中的寫入器節點存取未記錄的資料表。如需完整詳細資訊，請參閱在 Aurora PostgreSQL 中使用未記錄的資料表。

一般最佳實務：

- 未記錄資料表在 Aurora PostgreSQL 中提供有限的效能優勢，因此請評估正常資料表是否更適當。PostgreSQL

大約閾值：數千

分區

分割可以提高查詢效能並提供資料的邏輯組織。在理想情況下，會組織分割區，以便在查詢規劃和執行期間使用分割區剔除。使用太多分割區可能會對查詢效能和資料庫維護產生負面影響。應謹慎選擇如何分割資料表，因為查詢規劃和執行的效能可能會受到設計不佳的負面影響。如需分割的詳細資訊，請參閱 PostgreSQL 文件。

影響：一般效能降低

有時，規劃時間額外負荷會增加，並解釋查詢的計劃會變得更加複雜，導致難以識別調校機會。對於 18 之前的 PostgreSQL 版本，具有高工作負載的許多分割區可能會導致`LWLock:LockManager`等待。

建議動作：判斷可讓您完成資料組織，同時提供效能查詢執行的分割區數量下限。

影響：維護複雜性

非常大量的分割區會導致維護困難，例如建立前和移除。自動清空會將分割區視為一般關係，且必須執行定期清除，因此需要足夠的工作者來完成任務。

建議的動作：

- 請確定您預先建立分割區，以便在需要新分割區（例如，每月型分割區）且舊分割區推出時，不會封鎖工作負載。
- 確保您有足夠的自動清空工作者，可執行所有分割區的正常清除維護。

大約閾值：數百

暫存檔案

與上述的暫存資料表不同，當複雜的查詢可能同時執行多個排序或雜湊操作時，PostgreSQL 會建立暫存檔案，每個操作會使用執行個體記憶體將結果儲存到 `work_mem` 參數中指定的值。當執行個體記憶體不足時，會建立暫存檔來儲存結果。如需檔案的詳細資訊，請參閱管理暫存檔案。如果您的工作負載產生大量這些檔案，可能會有幾項影響。

影響：FreeLocalStorage 耗用量

> 當查詢結果無法容納 work_mem 時，暫存檔案是 PostgreSQL 的必要部分。一般而言，這沒問題。不過，當工作負載廣泛使用時，表示大型查詢會持續執行，而您將觀察到 FreeLocalStorage 減少。如需詳細資訊，請參閱管理暫存檔案。

一般最佳實務：

- 使用 Performance Insights 監控您的暫存檔案用量。
- 調校正在產生重要暫存檔案的查詢，以查看是否可以減少暫存檔案的總數。

大約閾值：數千

序列

序列是用於在 PostgreSQL 中自動遞增資料欄的基礎物件，可提供資料的唯一性和金鑰。這些可用於個別資料表，在正常操作期間不會產生任何後果，但邏輯複寫除外。

在 PostgreSQL 中，邏輯複寫目前不會將序列的目前值複寫至任何訂閱者。若要進一步了解，請參閱 PostgreSQL 文件中的限制頁面。

影響：延長切換時間

> 如果您計劃將 藍/綠部署用於任何類型的組態變更或升級，請務必了解大量序列對切換的影響。切換的最後一個階段之一將同步序列的目前值，如果有數千個，這將增加整體切換時間。

建議的動作：如果您的資料庫工作負載允許使用共用 UUID 而非sequence-per-table的方法，這會在切換期間減少同步步驟。

大約閾值：數千

大型物件

大型物件存放在名為 pg_largeobject 的單一系統資料表中。每個大型物件在系統資料表 pg_largeobject_metadata 中也有一個項目。這些物件的建立、修改和清除與標準關係截然不同。大型物件不會由自動清空處理，必須透過稱為 vacuumlo 的個別程序定期清理。如需管理大型物件的範例，請參閱使用 lo 模組管理大型物件。

影響：邏輯複寫

在邏輯複寫期間，目前不會在 PostgreSQL 中複寫大型物件。若要進一步了解，請參閱 PostgreSQL 文件中的限制頁面。在藍色/綠色組態中，這表示藍色環境中的大型物件不會複寫至綠色環境。

影響：主要版本升級

如果有數百萬個大型物件，且執行個體無法在升級期間處理它們，則升級可能會耗盡記憶體並失敗。PostgreSQL 主要版本升級程序包含兩個大階段：透過 pg_dump 傾印結構描述，並透過 pg_restore 還原結構描述。如果您的資料庫有數百萬個大型物件，您需要確保您的執行個體在升級期間有足夠的記憶體來處理 pg_dump 和 pg_restore，並將其擴展到更大的執行個體類型。

一般最佳實務：

- 定期使用 vacuumlo 公用程式移除您可能擁有的任何孤立大型物件。
- 請考慮使用 BYTEA 資料類型，將大型物件存放在資料庫中。

大約閾值：百萬

大約閾值

本主題中提到的大約閾值僅用於提供特定資源可擴展程度的估計。它們代表一般範圍，其中所描述的影響變得更有可能，但實際行為取決於您的特定工作負載、執行個體大小和組態。雖然可能超過這些預估值，但必須遵守注意和維護，以避免列出的影響。

Aurora PostgreSQL 中的記憶體管理已改善

Aurora PostgreSQL 現在包含進階記憶體管理功能，可在不同的工作負載下最佳化資料庫效能和彈性。這些改善有助於 Aurora PostgreSQL 維持一致的可用性和回應能力，即使在高記憶體需求的期間也是如此。

在下列適用於佈建執行個體的 Aurora PostgreSQL 版本中，此功能依預設會提供並啟用：

- 15.3 及所有更高次要版本
- 14.8 及更高次要版本
- 13.11 及更高次要版本
- 12.15 及更高次要版本
- 11.20 及更高次要版本

在下列適用於 Aurora Serverless 執行個體的 Aurora PostgreSQL 中，此功能依預設會提供並啟用：

- 16.3 及所有更高次要版本
- 15.7 及所有更高次要版本
- 14.12 及更高次要版本
- 13.15 及更高次要版本

當客戶工作負載用完所有可用的可用記憶體時，作業系統可能會重新啟動資料庫來保護資源，導致暫時無法使用。Aurora PostgreSQL 中新的記憶體管理改進會在系統遇到高記憶體壓力時主動取消特定交易，以協助維持資料庫穩定性。

改善記憶體管理的主要功能如下：

- 當系統接近重大記憶體壓力時，取消請求更多記憶體的資料庫交易。
- 當系統耗盡所有實體記憶體並即將耗盡交換記憶體時，一般認為該系統處於重大記憶體壓力之下。在這些情況下，任何請求記憶體的交易將會取消，以試圖降低資料庫執行個體中的記憶體壓力。
- 基本 PostgreSQL 啟動器和背景工作者 (例如自動真空工作者) 始終受到保護。

處理記憶體管理參數

開啟記憶體管理

此功能預設為開啟。當交易由於記憶體不足而取消時，系統會顯示錯誤訊息，如下列範例所示：

```
ERROR: out of memory Detail: Failed on request of size 16777216.
```

關閉記憶體管理

若要關閉此功能，請使用 psql 連線至 Aurora PostgreSQL 資料庫叢集，並針對參數值使用 SET 陳述式，如下所述。

> **Note**
>
> 我們建議您保持啟用記憶體管理。這有助於防止因記憶體耗盡而導致工作負載引發資料庫重新啟動的潛在記憶體不足錯誤。

下表顯示如何關閉不同 Aurora PostgreSQL 版本的記憶體管理功能：

Aurora PostgreSQL 版本	參數	預設	在工作階段層級關閉記憶體管理的命令
11.20、11.21、12.15、12.16、13.11、13.12、14.8、14.9、15.3、15.4	`rds.memory_allocation_guard`	`false`	`SET rds.memory_allocation_guard = true;`
12.17、13.13、14.10、15.5 及更高版本	`rds.enable_memory_management`	`true`	`SET rds.enable_memory_management = false;`

> **Note**
>
> 在 Aurora PostgreSQL 12.17、13.13、14.10、15.5 和更高版本中已棄用 `rds.memory_allocation_guard` 參數。

在資料庫叢集參數群組中設定這些參數的值，可防止查詢遭到取消。如需資料庫叢集參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。

限制

- db.t3.medium 執行個體類別不支援此功能。

Amazon Aurora PostgreSQL 的快速容錯移轉

以下，您可以瞭解如何確保儘可能快速地進行容錯移轉。若要在容錯移轉後快速復原，您可以對 Aurora PostgreSQL 資料庫叢集使用叢集快取管理。如需詳細資訊，請參閱Aurora PostgreSQL 的容錯移轉後使用叢集快取管理快速復原。

您可以採取，以便快速地執行容錯移轉的一些步驟包括下列步驟：

- 以短時間範圍設定傳輸控制通訊協定 (TCP) 保持連線，如果發生故障，則會在讀取逾時到期之前停止較長的執行查詢。
- 積極地設定 Java 網域名稱系統 (DNS) 快取的逾時時間。這麼做有助於確保 Aurora 唯讀端點可在稍後嘗試連線時正確地循環切換唯讀節點。
- 將在 JDBC 連線字串中使用的逾時變數設定為越低越好。對短時間和長時間執行的查詢使用不同的連線物件。
- 使用提供的讀取和寫入 Aurora 端點來連線到叢集。
- 使用 RDS API 操作測試應用程式在伺服器端故障時的回應。此外，使用封包捨棄工具來測試應用程式對用戶端故障的回應。
- 使用 AWS JDBC 驅動程式充分利用 Aurora PostgreSQL 的容錯移轉功能。如需 AWS JDBC 驅動程式的詳細資訊和完整的使用說明，請參閱 Amazon Web Services (AWS) JDBC 驅動程式 GitHub 儲存庫。

下文將詳細介紹這些內容。

主題

- 設定 TCP 保持連線參數
- 為快速容錯移轉設定您的應用程式
- 測試容錯移轉
- 快速容錯移轉 Java 範例

設定 TCP 保持連線參數

設定 TCP 連線時，有一組計時器與連線關聯。當保持連線計時器到達零時，會傳送一個保持連線探測封包給連線端點。如果探測收到回覆，您可以假設該連線仍然正常執行。

開啟 TCP 保持連線參數並積極地設定它們，可確保如果您的用戶端無法連線至資料庫時，即可快速關閉任何作用中的連線。然後，應用程式可以連線到新的端點。

請確認您設定了下列 TCP 保持連線參數：

- `tcp_keepalives_idle` 控制時間 (以秒為單位)，如果通訊端經過此時間後未傳送任何資料，則傳送保持連線封包。ACK 並不視為資料。我們建議您使用下列設定：

  ```
  tcp_keepalives_idle = 1
  ```
- `tcp_keepalives_interval` 控制時間 (以秒為單位)，此時間為傳送初始封包之後傳送後續保持連線封包的時間。請使用 `tcp_keepalives_idle` 參數設定此時間。我們建議您使用下列設定：

  ```
  tcp_keepalives_interval = 1
  ```
- `tcp_keepalives_count` 為通知應用程式之前發生的未確認保持連線探測的數量。我們建議您使用下列設定：

  ```
  tcp_keepalives_count = 5
  ```

這些設定應該在資料庫停止回應的五秒鐘內通知應用程式。如果在應用程式網路內保持連線封包經常遭到捨棄，則可以設定較高的 `tcp_keepalives_count` 值。這麼做雖然會增加偵測實際故障所需的時間，但在較不可靠的網路中可允許更多緩衝。

若要在 Linux 上設定 TCP 保持連線參數

1. 測試如何設定 TCP 保持連線參數。

   我們建議您在命令列中使用下列命令來執行此作業。這個建議的組態是屬於整個系統的。換句話說，它也會影響在 SO_KEEPALIVE 選項開啟的情況下建立連線的所有其他應用程式。

   ```
   sudo sysctl net.ipv4.tcp_keepalive_time=1
   sudo sysctl net.ipv4.tcp_keepalive_intvl=1
   sudo sysctl net.ipv4.tcp_keepalive_probes=5
   ```
2. 找到適合您應用程式的組態之後，請將下列各行 (包括您所做的任何變更) 新增至 `/etc/sysctl.conf`，以持續保留這些設定：

```
tcp_keepalive_time = 1
tcp_keepalive_intvl = 1
tcp_keepalive_probes = 5
```

為快速容錯移轉設定您的應用程式

接下來，您可以找到針對 Aurora PostgreSQL 進行的數個組態變更的討論，這些變更可以進行快速容錯移轉。若要進一步了解 PostgreSQL JDBC 驅動程式設定和組態，請參閱 PostgreSQL JDBC 驅動程式文件。

主題

- 減少 DNS 快取逾時
- 為快速容錯移轉設定 Aurora PostgreSQL 連線字串
- 用於取得主機字串的其他選項

減少 DNS 快取逾時

當您的應用程式嘗試在容錯移轉之後建立連線時，新的 Aurora PostgreSQL 寫入器將會是先前的讀取器。您可以在 DNS 更新完全傳播之前使用 Aurora 唯讀端點來找到它。將 java DNS 存活時間 (TTL) 設定為較低的值 (例如 30 秒)，有助於在後續連線嘗試時切換讀取器節點。

```
// Sets internal TTL to match the Aurora RO Endpoint TTL
java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
// If the lookup fails, default to something like small to retry
java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
```

為快速容錯移轉設定 Aurora PostgreSQL 連線字串

若要使用 Aurora PostgreSQL 快速容錯移轉，請確認您應用程式的連線字串具有主機的清單，而非只有單一主機。以下是可以用來連線至 Aurora PostgreSQL 叢集的範例連線字串。在此範例中，主機以粗體顯示。

```
jdbc:postgresql://myauroracluster.cluster-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
/postgres?user=<primaryuser>&password=<primarypw>&loginTimeout=2
&connectTimeout=2&cancelSignalTimeout=2&socketTimeout=60
```

```
&tcpKeepAlive=true&targetServerType=primary
```

為了獲得最佳可用性，以及避免對 RDS API 的依賴，我們建議您維護要連線的檔案。此檔案包含應用程式在您建立與資料庫的連線時會從中讀取的主機字串。此主機字串具有可供叢集使用的所有 Aurora 端點。如需 Aurora 端點的詳細資訊，請參閱Amazon Aurora 端點連線。

例如，您可以將端點儲存在本機檔案中，如下所示。

```
myauroracluster.cluster-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

您的應用程式會從此檔案讀取，以填入 JDBC 連線字串的主機區段。重新命名資料庫叢集會造成這些端點變更。如果發生這種情況，請確定您的應用程式會處理該情況。

另一個選項是使用資料庫執行個體節點的清單，如下所示。

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node3.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node4.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

此方法的優點是 PostgreSQL JDBC 連線驅動程式會在此清單中重複查看所有節點，以尋找有效的連線。相形之下，當您使用 Aurora 端點時，每次嘗試連線時，只會嘗試兩個節點。但是，使用資料庫執行個體節點有一個缺點。如果從您的叢集新增或移除節點，而執行個體端點的清單變得過時，則連線驅動程式可能永遠找不到要連線的正確主機。

積極地設定下列參數，以協助確保您的應用程式不會在連線至任何主機時等候過久。

- `targetServerType` – 控制驅動程式是否連接至寫入或讀取節點。若要確保您的應用程式只會重新連線至寫入節點，請將 `targetServerType` 值設定為 `primary`。

  `targetServerType` 參數的值包括 `primary`、`secondary`、`any` 和 `preferSecondary`。此 `preferSecondary` 值會先嘗試與讀取器建立連線。如果沒有讀取器連線可以建立，則會連線至寫入器。
- `loginTimeout` – 控制通訊端連線建立之後，您的應用程式等待登入資料庫的時間長短。
- `connectTimeout` – 控制通訊端等候資料庫連線建立的時間長短。

視您希望應用程式的積極程度而定，可以修改其他應用程式參數以加速連線程序。

- cancelSignalTimeout – 在某些應用程式中，您可能希望對已逾時的查詢傳送「最大努力」取消信號。如果此取消信號位於您的容錯移轉路徑中，您應該考慮積極地設定它，以避免將此信號傳送至已停止執行的主機。
- socketTimeout – 此參數可控制通訊端等候讀取操作的時間長短。此參數可用做為全域「查詢逾時」，以確保任何查詢的等候時間均不超過於此值。一個好的做法是擁有兩個連線處理程序。一個連接處理程序運行短期查詢，並將此值設定為較低的值。另一個連線處理常式 (針對長時間執行的查詢) 的設定要高得多。使用此種方法，您可以仰賴 TCP 保持連線參數，當伺服器關閉時停止長時間執行的查詢。
- tcpKeepAlive – 啟用此參數以確保系統會遵守您所設定的 TCP 保持連線參數。
- loadBalanceHosts – 設為 true 時，此參數會將應用程式連接至自候選主機清單選擇的隨機主機。

用於取得主機字串的其他選項

您可以從數個來源 (包括 aurora_replica_status 函數和透過使用 Amazon RDS API) 取得主機字串。

在許多情況下，您需要判斷誰是叢集的寫入器，或尋找叢集中的其他讀取器節點。為了這麼做，您的應用程式可以連線至資料庫叢集中的任何資料庫執行個體，並查詢 aurora_replica_status 函數。您可以使用此函數來減少尋找要連接之主機所花費的時間量。然而在某些網路故障情況下，aurora_replica_status 函數可能會顯示過時或不完整的資訊。

為了確保應用程式可以找到要連線的節點，最好嘗試先連線至叢集寫入器端點，接著再連線至叢集讀取器端點。您可以執行此操作，直到您可以建立可讀連線為止。除非您重新命名資料庫叢集，否則這些端點不會變更。因此一般而言，您可以將它們保留為您應用程式的靜態成員，或存放在您應用程式會從中讀取的資源檔案中。

使用其中一個端點建立連線之後，您可以取得叢集其餘部分的資訊。若要執行此作業，請呼叫 aurora_replica_status 函數。例如，下列命令會使用 aurora_replica_status 來擷取資訊。

```
postgres=> SELECT server_id, session_id, highest_lsn_rcvd, cur_replay_latency_in_usec,
 now(), last_update_timestamp
FROM aurora_replica_status();

server_id | session_id | highest_lsn_rcvd | cur_replay_latency_in_usec | now |
 last_update_timestamp
-----------+--------------------------------------+------------------
+---------------------------+-------------------------------+-----------------------
```

```
mynode-1 | 3e3c5044-02e2-11e7-b70d-95172646d6ca | 594221001 | 201421 | 2017-03-07
 19:50:24.695322+00 | 2017-03-07 19:50:23+00
mynode-2 | 1efdd188-02e4-11e7-becd-f12d7c88a28a | 594221001 | 201350 | 2017-03-07
 19:50:24.695322+00 | 2017-03-07 19:50:23+00
mynode-3 | MASTER_SESSION_ID | | | 2017-03-07 19:50:24.695322+00 | 2017-03-07
 19:50:23+00
(3 rows)
```

例如連線字串的主機區段可以從寫入器和讀取器叢集端點開始，如下所示。

```
myauroracluster.cluster-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

在此案例中，您的應用程式會嘗試與任何節點類型 (主要或次要) 建立連線。應用程式連線之後，先檢查節點的讀取/寫入狀態，這是良好的做法。要做到這一點，請查詢命令 `SHOW transaction_read_only` 的結果。

如果查詢的傳回值為 `OFF`，則您已成功連線至主節點。然而，假設傳回值是 `ON`，而且您的應用程式需要讀取/寫入連線。在此種情況下，您可以呼叫 `aurora_replica_status` 函數來確定 `server_id` 具有 `session_id='MASTER_SESSION_ID'`。此函數提供主要節點的名稱。您可以將其與 `endpointPostfix` 搭配使用，說明如下。

當您連線到具有過時資料的複本時，請確保您曉得此種情況。發生此情況時，`aurora_replica_status` 函數可能會顯示過時的資訊。您可以在應用程式層級設定失效的臨界值。若要檢查這一點，您可以查看伺服器時間與 `last_update_timestamp` 值之間的差異。一般來說，由於 `aurora_replica_status` 函數傳回的資訊會衝突，您的應用程式應該確保避免在兩個主機之間反覆執行。您的應用程式應該先嘗試所有已知主機，而不是遵循 `aurora_replica_status` 函數傳回的資料。

使用 DescribeDBClusters API 操作列出執行個體的 Java 範例

您可以使用 適用於 Java 的 AWS SDK (特別是 DescribeDBClusters API 操作)，以程式設計方式尋找執行個體的清單。

這裡是您在 Java 8 中如何執行此動作的小型範例：

```
AmazonRDS client = AmazonRDSClientBuilder.defaultClient();
DescribeDBClustersRequest request = new DescribeDBClustersRequest()
   .withDBClusterIdentifier(clusterName);
DescribeDBClustersResult result =
rdsClient.describeDBClusters(request);
```

```
DBCluster singleClusterResult = result.getDBClusters().get(0);

String pgJDBCEndpointStr =
singleClusterResult.getDBClusterMembers().stream()
   .sorted(Comparator.comparing(DBClusterMember::getIsClusterWriter)
   .reversed()) // This puts the writer at the front of the list
   .map(m -> m.getDBInstanceIdentifier() + endpointPostfix + ":" +
 singleClusterResult.getPort()))
   .collect(Collectors.joining(","));
```

這裡，pgJDBCEndpointStr 包含端點的格式化清單，如下所示。

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

變數 endpointPostfix 可以是您的應用程式設定的常數。或者您的應用程式可以針對叢集中的單一執行個體查詢 DescribeDBInstances API 操作而取得此資訊。對於個別客戶，此值在 AWS 區域 和內保持不變。因此，它免去了 API 呼叫，只將此常數保留在應用程式從中讀取的資源檔案中。在先前的範例中，它被設定為下列內容。

```
.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com
```

針對高可用性目的，如果 API 未回應或回應時間太長，理想的做法是預設為使用資料庫叢集的 Aurora 端點。在更新 DNS 記錄耗用的時間內會保證這些端點是最新的。使用端點更新 DNS 記錄需要的時間通常不到 30 秒。您可以將端點儲存在應用程式取用的資源檔案中。

測試容錯移轉

在所有情況下，您必須擁有包含兩個或多個資料庫執行個體的資料庫叢集。

從伺服器端，某些 API 操作可能會導致中斷，可用來測試您的應用程式如何回應：

- FailoverDBCluster – 此作業嘗試將資料庫叢集中的新資料庫執行個體提升為寫入器。

  下列程式碼範例顯示如何使用 failoverDBCluster 來造成中斷。如需設定 Amazon RDS 用戶端的詳細資訊，請參閱使用適用於 Java 的 AWS 開發套件。

  ```
  public void causeFailover() {

      final AmazonRDS rdsClient = AmazonRDSClientBuilder.defaultClient();
  ```

```
    FailoverDBClusterRequest request = new FailoverDBClusterRequest();
    request.setDBClusterIdentifier("cluster-identifier");

    rdsClient.failoverDBCluster(request);
}
```

- RebootDBInstance – 此 API 操作不保證容錯移轉。但是，它會關閉寫入器上的資料庫。您可以使用它來測試您的應用程式對連線中斷的回應方式。`ForceFailover` 參數不適用於 Aurora 引擎。反之，您可以使用 `FailoverDBCluster` API 操作。
- ModifyDBCluster – 叢集中的節點開始聆聽新連接埠時，修改 `Port` 參數會造成中斷。一般而言，您的應用程式可以先回應此故障，方法是確保只有您的應用程式會控制連接埠變更。此外，請確保它可以適當地更新所依賴的端點。您可以讓某人在 API 層級進行修改時手動更新連接埠來執行此操作。或者，您也可以使用應用程式中的 RDS API 來判斷連接埠是否已變更。
- ModifyDBInstance – 修改 `DBInstanceClass` 參數會導致中斷。
- DeleteDBInstance – 刪除主要 (寫入器) 執行個體會導致資料庫叢集的新資料庫執行個體被提升為寫入器。

如果您使用的是 Linux，您可以從應用程式或用戶端測試應用程式如何回應突然的封包捨棄。您可以使用 iptables 命令，依據連接埠、主機是否傳送或接收 TCP 保持連線封包來執行此操作。

快速容錯移轉 Java 範例

下列程式碼範例示範應用程式設定 Aurora PostgreSQL 驅動程式管理員的方法。

應用程式會在需要連線時呼叫 `getConnection`。呼叫 `getConnection` 可能無法找到有效的主機。一個範例是，沒有發現寫入器，但 `targetServerType` 參數設定為 `primary` 時。在此種情況下，呼叫應用程式應該直接重試呼叫該函數。

此重試呼叫可以輕鬆包裝於連線集區內，以避免將重試行為推送至應用程式。對於大部分連線集區工具而言，您可以指定 JDBC 連接字串。所以您的應用程式可以呼叫 `getJdbcConnectionString` 並將其傳送給連線集區。這樣做意味著您可以透過 Aurora PostgreSQL 使用更快速的容錯移轉。

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
```

```
import java.util.stream.IntStream;

import org.joda.time.Duration;

public class FastFailoverDriverManager {
   private static Duration LOGIN_TIMEOUT = Duration.standardSeconds(2);
   private static Duration CONNECT_TIMEOUT = Duration.standardSeconds(2);
   private static Duration CANCEL_SIGNAL_TIMEOUT = Duration.standardSeconds(1);
   private static Duration DEFAULT_SOCKET_TIMEOUT = Duration.standardSeconds(5);

   public FastFailoverDriverManager() {
       try {
            Class.forName("org.postgresql.Driver");
       } catch (ClassNotFoundException e) {
            e.printStackTrace();
       }

       /*
         * RO endpoint has a TTL of 1s, we should honor that here. Setting this
 aggressively makes sure that when
         * the PG JDBC driver creates a new connection, it will resolve a new different
 RO endpoint on subsequent attempts
         * (assuming there is > 1 read node in your cluster)
         */
        java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
       // If the lookup fails, default to something like small to retry
       java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
   }

   public Connection getConnection(String targetServerType) throws SQLException {
       return getConnection(targetServerType, DEFAULT_SOCKET_TIMEOUT);
   }

   public Connection getConnection(String targetServerType, Duration queryTimeout)
 throws SQLException {
        Connection conn =
 DriverManager.getConnection(getJdbcConnectionString(targetServerType, queryTimeout));

       /*
         * A good practice is to set socket and statement timeout to be the same thing
 since both
         * the client AND server will stop the query at the same time, leaving no
 running queries
         * on the backend
```

```
         */
        Statement st = conn.createStatement();
        st.execute("set statement_timeout to " + queryTimeout.getMillis());
        st.close();

       return conn;
   }

   private static String urlFormat = "jdbc:postgresql://%s"
           + "/postgres"
           + "?user=%s"
           + "&password=%s"
           + "&loginTimeout=%d"
           + "&connectTimeout=%d"
           + "&cancelSignalTimeout=%d"
           + "&socketTimeout=%d"
           + "&targetServerType=%s"
           + "&tcpKeepAlive=true"
           + "&ssl=true"
           + "&loadBalanceHosts=true";
   public String getJdbcConnectionString(String targetServerType, Duration
 queryTimeout) {
       return String.format(urlFormat,
                getFormattedEndpointList(getLocalEndpointList()),
                CredentialManager.getUsername(),
                CredentialManager.getPassword(),
                LOGIN_TIMEOUT.getStandardSeconds(),
                CONNECT_TIMEOUT.getStandardSeconds(),
                CANCEL_SIGNAL_TIMEOUT.getStandardSeconds(),
                queryTimeout.getStandardSeconds(),
                targetServerType
       );
   }

   private List<String> getLocalEndpointList() {
       /*
         * As mentioned in the best practices doc, a good idea is to read a local
 resource file and parse the cluster endpoints.
         * For illustration purposes, the endpoint list is hardcoded here
         */
        List<String> newEndpointList = new ArrayList<>();
        newEndpointList.add("myauroracluster.cluster-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432");
```

```
        newEndpointList.add("myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432");

        return newEndpointList;
    }

    private static String getFormattedEndpointList(List<String> endpoints) {
        return IntStream.range(0, endpoints.size())
                .mapToObj(i -> endpoints.get(i).toString())
                .collect(Collectors.joining(","));
    }
}
```

Aurora PostgreSQL 的容錯移轉後使用叢集快取管理快速復原

針對您的 Aurora PostgreSQL 叢集發生容錯移轉後快速復原資料庫執行個體寫入器，對 Amazon Aurora PostgreSQL 使用叢集快取管理。叢集快取管理確保應用程式如果發生容錯移轉時維持效能。

在典型的容錯移轉情況中，容錯移轉後您可能會發現效能暫時大幅降低。這樣的降低情形會發生於容錯移轉資料庫執行個體啟動、但緩衝快取處於空的狀態時。空的快取也稱作 冷快取。冷快取會降低效能，因為資料庫執行個體需要從較慢的磁碟讀取，而非利用緩衝快取中儲存的值。

使用叢集快取管理，設定特定讀取器的資料庫執行個體作為容錯移轉的目標。叢集快取管理確保指定讀取器中的快取資料，能夠與資料庫執行個體寫入器的快取資料保持同步。預先填入值的指定讀取器快取也稱作 熱快取。如果容錯轉移發生，指定的讀取器立即使用熱快取的值提升到新的資料庫執行個體寫入器。此方法提供您的應用程式更佳的復原效能。

叢集快取管理要求指定的讀取器執行個體具有與編寫器相同的執行個體類別類型和大小 (例如 `db.r5.2xlarge` 或 `db.r5.xlarge`)。當您建立 Aurora PostgreSQL 資料庫叢集時，請記住這一點，以便叢集可以在容錯移轉期間復原。如需執行個體類別類型和大小的清單，請參閱 Aurora 的資料庫執行個體類別的硬體規格。

> **Note**
>
> 屬於 Aurora 全球資料庫一部分的 Aurora PostgreSQL 次要資料庫叢集不支援叢集快取管理。對於支援此功能的主要叢集，建議不要在指定的層級 0 讀取器執行工作負載。

內容

- 設定叢集快取管理

- 啟用叢集快取管理
- 設定寫入器資料庫執行個體的提取層級優先順序
- 設定讀取器資料庫執行個體的提取層級優先順序。
- 監控緩衝區快取
- 對 CCM 組態進行疑難排解

設定叢集快取管理

若要設定叢集快取管理，請依序執行下列程序。

主題

- 啟用叢集快取管理
- 設定寫入器資料庫執行個體的提取層級優先順序
- 設定讀取器資料庫執行個體的提取層級優先順序。

> **Note**
>
> 在完成這些步驟之後，等待至少 1 分鐘，讓叢集快取管理可以開始正常運作。

啟用叢集快取管理

若要啟用叢集快取管理，請執行以下所述的步驟。

主控台

若要啟用叢集快取管理

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。
3. 在清單中，針對您的 Aurora PostgreSQL 資料庫叢集選擇參數群組。

   資料庫叢集必須使用預設以外的參數群組，因為您無法變更參數群組中的值。
4. 針對 Parameter group actions (參數群組動作)，選擇 Edit (編輯)。
5. 設定叢集參數中 `apg_ccm_enabled` 的值到 `1`。

6. 選擇儲存變更。

AWS CLI

若要啟用 Aurora PostgreSQL 資料庫叢集的叢集快取管理，請使用 AWS CLI `modify-db-cluster-parameter-group`命令搭配下列必要參數：

- `--db-cluster-parameter-group-name`
- `--parameters`

Example

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my-db-cluster-parameter-group \
    --parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

在 Windows 中：

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my-db-cluster-parameter-group ^
    --parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

設定寫入器資料庫執行個體的提取層級優先順序

針對叢集快取管理，請確保 Aurora PostgreSQL 資料庫叢集的寫入器資料庫執行個體提取優先順序為層級 0。提取層級優先順序的值，代表容錯移轉後 Aurora 讀取器提升為寫入器資料庫執行個體的特定順序。有效的值為 0–15，0 代表最高優先順序，15代表最低優先順序。如需提取層級的詳細資訊，請參閱 Aurora 資料庫叢集的容錯能力。

主控台

若要設定資料庫執行個體寫入器提取優先順序為層級 0

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇 Writer (寫入器) Aurora PostgreSQL 資料庫叢集的資料庫執行個體。

4. 選擇 Modify (修改)。Modify DB Instance (修改資料庫執行個體) 頁面隨即出現。
5. 在 Additional configuration (其他組態) 面板上，為 Failover priority (容錯移轉優先順序) 選擇 tier-0 (層級 0)。
6. 選擇 Continue (繼續)，並檢查修改的摘要。
7. 若要在儲存後立即套用變更，選擇 Apply immediately (立即套用)。
8. 選擇 Modify DB Instance (修改資料庫執行個體)，以儲存變更。

AWS CLI

若要使用 將寫入器資料庫執行個體的提升層優先順序設定為 0 AWS CLI，請使用下列必要參數呼叫 modify-db-instance 命令：

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

針對 Linux、macOS 或 Unix：

```
aws rds modify-db-instance \
    --db-instance-identifier writer-db-instance \
    --promotion-tier 0 \
    --apply-immediately
```

在 Windows 中：

```
aws rds modify-db-instance ^
    --db-instance-identifier writer-db-instance ^
    ---promotion-tier 0  ^
    --apply-immediately
```

設定讀取器資料庫執行個體的提取層級優先順序。

您必須為叢集快取管理設定唯一的讀取器資料庫執行個體。若要執行此作業，請從 Aurora PostgreSQL 叢集選擇與資料庫執行個體寫入器相同執行個體類別和大小的讀取器。例如，如果編寫器使用

db.r5.xlarge，請選擇使用此相同執行個體類別類型和大小的讀取器。然後設定提升層級優先順序為 0。

提取層級優先順序的值，代表容錯移轉後 Aurora 讀取器提升為寫入器資料庫執行個體的特定順序。有效的值為 0–15，0 代表最高優先順序，15 代表最低優先順序。

主控台

設定資料庫執行個體讀取器的提取層級優先順序到 0。

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇與資料庫執行個體寫入器相同執行個體類別 Aurora PostgreSQL 資料庫叢集的資料庫執行個體 Reader (讀取器)。
4. 選擇 Modify (修改)。Modify DB Instance (修改資料庫執行個體) 頁面隨即出現。
5. 在 Additional configuration (其他組態) 面板上，為 Failover priority (容錯移轉優先順序) 選擇 tier-0 (層級 0)。
6. 選擇 Continue (繼續)，並檢查修改的摘要。
7. 若要在儲存後立即套用變更，選擇 Apply immediately (立即套用)。
8. 選擇 Modify DB Instance (修改資料庫執行個體)，以儲存變更。

AWS CLI

若要使用 將讀取器資料庫執行個體的提升層優先順序設定為 0 AWS CLI，請使用下列必要參數呼叫 modify-db-instance 命令：

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

針對 Linux、macOS 或 Unix：

```
aws rds modify-db-instance \
```

```
    --db-instance-identifier reader-db-instance \
    --promotion-tier 0 \
    --apply-immediately
```

在 Windows 中：

```
aws rds modify-db-instance ^
    --db-instance-identifier reader-db-instance ^
    ---promotion-tier 0  ^
    --apply-immediately
```

監控緩衝區快取

在設定叢集快取管理後，您可以監控資料庫執行個體寫入器緩衝快取和指定讀取器熱緩衝快取間的同步狀態。若要檢查資料庫執行個體寫入器與指定資料庫執行個體讀取器的熱快取內容，請使用 PostgreSQL pg_buffercache 模組。如需詳細資訊，請參閱 PostgreSQL pg_buffercache 文件。

使用 **aurora_ccm_status** 函數

叢集快取管理也提供 aurora_ccm_status 函數。使用資料庫執行個體寫入器上的 aurora_ccm_status 函數以獲得下列關於指定讀取器上熱快取進度的資訊：

- buffers_sent_last_minute – 最後一分鐘有多少緩衝區送到指定的讀取器。
- buffers_found_last_minute – 過去一分鐘內識別出頻繁存取緩衝區的數量。
- buffers_sent_last_scan – 最後完成緩衝區快取掃描時有多少緩衝區送到指定的讀取器。
- buffers_found_last_scan – 有多少緩衝已經辨識為經常許可並需要在最後完成緩衝快取掃描時送出。緩衝區指定讀取器上未送出的快取。
- buffers_sent_current_scan – 截至目前掃描送出多少緩衝區。
- buffers_found_current_scan – 目前掃描有多少緩衝區已經辨識為經常許可。
- current_scan_progress – 目前的掃描期間迄今已造訪多少緩衝區。

以下範例示範如何使用 aurora_ccm_status 函數將一些輸出轉換為暖比率和暖百分比。

```
SELECT buffers_sent_last_minute*8/60 AS warm_rate_kbps,
   100*(1.0-buffers_sent_last_scan::float/buffers_found_last_scan) AS warm_percent
   FROM aurora_ccm_status();
```

對 CCM 組態進行疑難排解

當您啟用 apg_ccm_enabled 叢集參數時，系統會在寫入器資料庫執行個體的執行個體層級自動開啟叢集快取管理，並在 Aurora PostgreSQL 資料庫叢集自動開啟一個讀取器資料庫執行個體。寫入器和讀取器執行個體應使用相同的執行個體類別類型和大小。其提升層優先順序會設為 0。資料庫叢集中的其他讀取器執行個體應具有非零的提升層，且會針對那些執行個體停用叢集快取管理。

下列原因可能會導致組態發生問題，以及停用叢集快取管理：

- 未將單一讀取器資料庫執行個體設為提升層 0 時。
- 未將寫入器資料庫執行個體設為提升層 0 時。
- 將多個讀取器資料庫執行個體設為提升層 0 時。
- 當具有提升層 0 的寫入器和一個讀取器資料庫執行個體的執行個體大小不同時。

使用集區管理 Aurora PostgreSQL 連線流失

當用戶端應用程式經常連線和中斷連線，以致 Aurora PostgreSQL 資料庫叢集回應時間變慢時，就表示叢集正在經歷連線流失。每個與 Aurora PostgreSQL 資料庫叢集端點的新連線都會消耗資源，因而減少可用於處理實際工作負載的資源。我們建議您遵循以下討論的一些最佳實務來管理連線流失問題。

對於初學者來說，您可以改善連線流失率高的 Aurora PostgreSQL 資料庫叢集的回應時間。為了這麼做，您可以使用連線集區，例如 RDS Proxy。連線集區為用戶端提供立即可用連線的快取。幾乎所有版本的 Aurora PostgreSQL 都支援 RDS 代理。如需詳細資訊，請參閱使用 Aurora PostgreSQL 的 Amazon RDS Proxy。

如果您的特定版本 Aurora PostgreSQL 不支援 RDS Proxy，您可以使用其他與 PostgreSQL 相容的連線集區工具，例如 PgBouncer。如需進一步了解，請參閱 PgBouncer 網站。

如需查看您的 Aurora PostgreSQL 資料庫叢集是否會受益於連線集區，您可以檢查用於連線和連線中斷的 postgresql.log 檔案。您也可以使用 Performance Insights，找出 Aurora PostgreSQL 資料庫叢集遇到多少連線流失。在以下內容中您可以找到這兩個主題的相關資訊。

記錄連線與中斷連線

PostgreSQL log_connections 和 log_disconnections 參數可以擷取 Aurora PostgreSQL 資料庫叢集的寫入器執行個體、的連線和連線中斷。這些參數預設會被關閉。若要開啟這些參數，請使用自訂參數群組，並將值變更為 1 來開啟。如需自訂參數群組的詳細資訊，請參閱 Amazon Aurora 資料庫叢集的資料庫叢集參數群組。若要檢查設定，請依照下列方式使用 psql 和查詢連線至 Aurora PostgreSQL 的資料庫叢集端點。

```
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_connections';
 setting
---------
on
(1 row)
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_disconnections';
setting
---------
on
(1 row)
```

當這兩個參數都開啟時，記錄會擷取所有的新連線和中斷連線。您會看到每個新授權連線的使用者和資料庫。在斷線時，也會記錄工作階段持續時間，如以下範例所示。

```
2022-03-07 21:44:53.978 UTC [16641] LOG: connection authorized: user=labtek
 database=labdb application_name=psql
2022-03-07 21:44:55.718 UTC [16641] LOG: disconnection: session time: 0:00:01.740
 user=labtek database=labdb host=[local]
```

若要檢查您的應用程式是否有連線流失，請開啟這些參數 (如果尚未開啟)。然後以實際的工作負載和時間間隔執行應用程式，收集 PostgreSQL 日誌檔中的資料以進行分析。您可以在 RDS 主控台中檢視日誌檔。選擇 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，然後選擇 Logs & events (日誌和事件) 標籤。如需詳細資訊，請參閱檢視並列出資料庫日誌檔案。

或者，您可以從控制台下載日誌檔並使用以下命令序列。此序列會尋找每分鐘授權和捨棄的連線總數。

```
grep "connection authorized\|disconnection: session time:"
 postgresql.log.2022-03-21-16|\
awk {'print $1,$2}' |\
sort |\
uniq -c |\
sort -n -k1
```

在範例輸出中，您可以看到授權連線中出現尖峰，然後從 16:12:10 開始連線中斷。

```
.....
,.......
.........
5 2022-03-21 16:11:55 connection authorized:
```

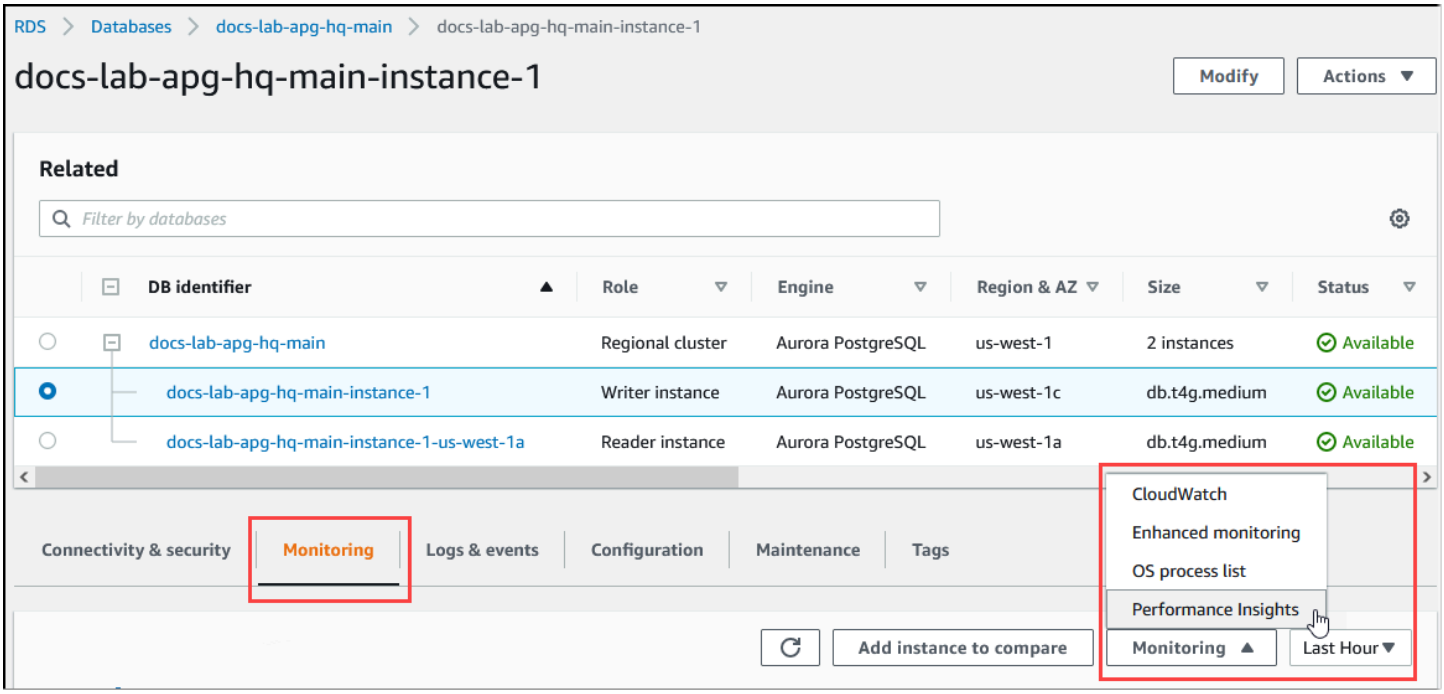
```
9 2022-03-21 16:11:55 disconnection: session
5 2022-03-21 16:11:56 connection authorized:
5 2022-03-21 16:11:57 connection authorized:
5 2022-03-21 16:11:57 disconnection: session
32 2022-03-21 16:12:10 connection authorized:
30 2022-03-21 16:12:10 disconnection: session
31 2022-03-21 16:12:11 connection authorized:
27 2022-03-21 16:12:11 disconnection: session
27 2022-03-21 16:12:12 connection authorized:
27 2022-03-21 16:12:12 disconnection: session
41 2022-03-21 16:12:13 connection authorized:
47 2022-03-21 16:12:13 disconnection: session
46 2022-03-21 16:12:14 connection authorized:
41 2022-03-21 16:12:14 disconnection: session
24 2022-03-21 16:12:15 connection authorized:
29 2022-03-21 16:12:15 disconnection: session
28 2022-03-21 16:12:16 connection authorized:
24 2022-03-21 16:12:16 disconnection: session
40 2022-03-21 16:12:17 connection authorized:
42 2022-03-21 16:12:17 disconnection: session
40 2022-03-21 16:12:18 connection authorized:
40 2022-03-21 16:12:18 disconnection: session
.....
,.......
.........
1 2022-03-21 16:14:10 connection authorized:
1 2022-03-21 16:14:10 disconnection: session
1 2022-03-21 16:15:00 connection authorized:
1 2022-03-21 16:16:00 connection authorized:
```

有了這些資訊，您就可以決定您的工作負載是否可以從連線集區工具中獲益。如需更詳細的分析，您可以使用 Performance Insights。

使用 Performance Insights 偵測連線流失

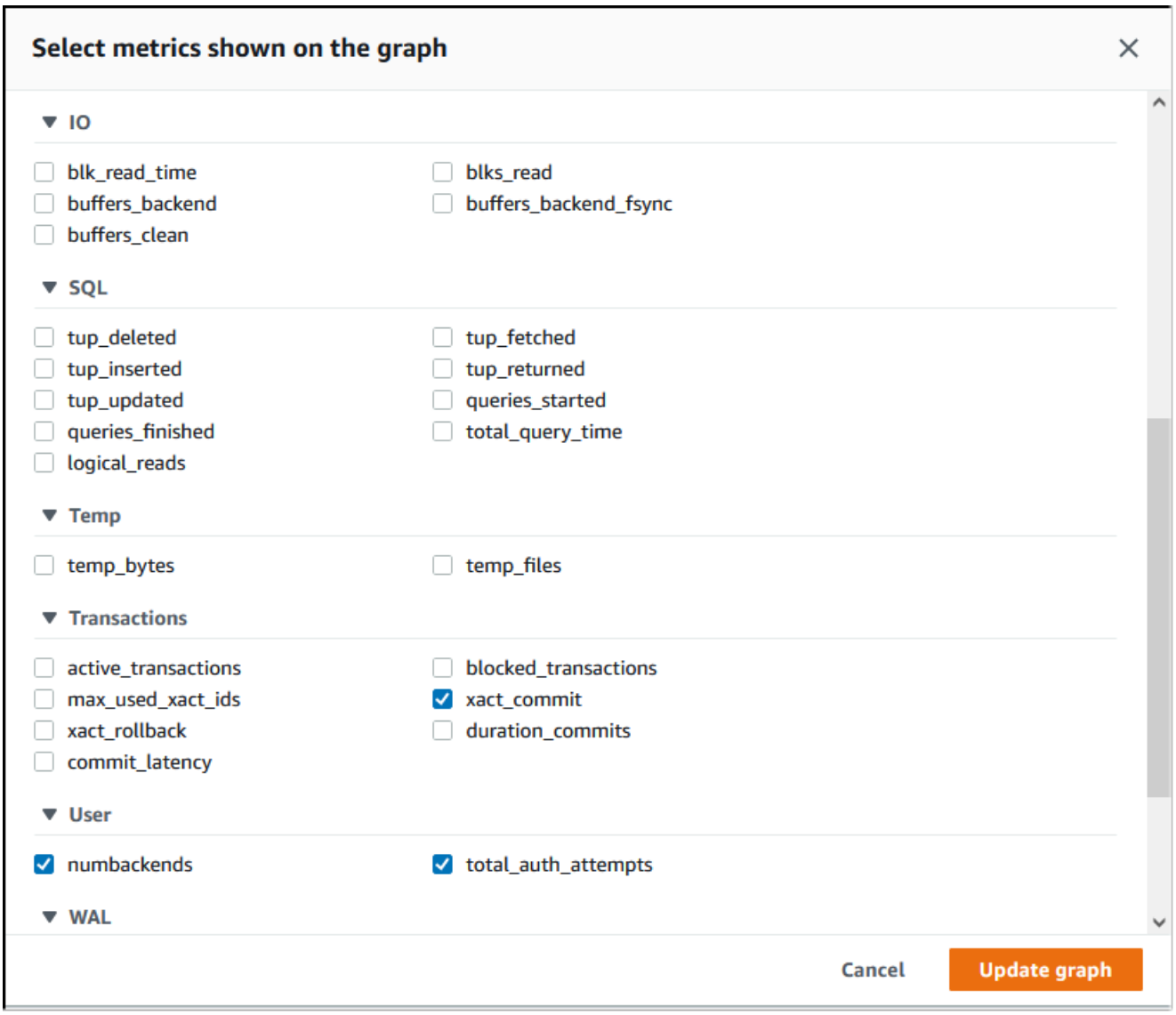
您可以使用 Performance Insights 來評估您 Aurora PostgreSQL 相容版本資料庫叢集的連線流失。當您建立 Aurora PostgreSQL 資料庫叢集時，預設會開啟 Performance Insights 的設定。如果您在建立資料庫叢集時清除此選項，請修改叢集以開啟此功能。如需詳細資訊，請參閱修改 Amazon Aurora 資料庫叢集。

透過在 Aurora PostgreSQL 資料庫叢集上執行的 Performance Insights，您可以選擇要監控的指標。您可以在主控台中的導覽窗格存取效能洞見。您也可以從 Aurora PostgreSQL 資料庫叢集的寫入器執行個體的 Monitoring (監控) 標籤存取 Performance Insights，如下圖所示。



從「Performance Insights」主控台選擇 Manage metrics (管理指標)。若要分析 Aurora PostgreSQL 資料庫叢集的連線和中斷連線動作，請選擇下列指標。這些都是來自 PostgreSQL 的指標。

- `xact_commit` – 已遞交的交易數量。
- `total_auth_attempts` – 每分鐘嘗試驗證的使用者連線數目。
- `numbackends` – 目前連線至資料庫的後端數目。

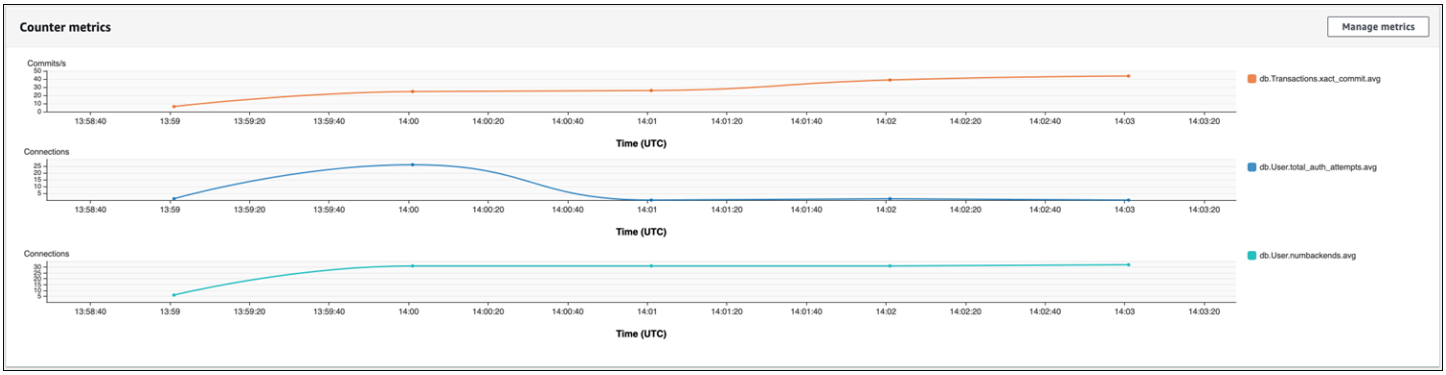


若要儲存設定並顯示連線動作，請選擇 Update graph (更新圖表)。

在下圖中，您可以看到有 100 個使用者時運行 pgbench 的影響。顯示連線位於一致向上斜坡的直線。若要進一步了解 pgbench 以及如何使用，請參閱 PostgreSQL 文件中的 pgbench。

此影像顯示，如果在沒有連線集區的情況下執行最多 100 個使用者的工作負載，可能會導致在整個工作負載處理期間 `total_auth_attempts` 的數量大幅增加。請注意，最好盡可能讓 `total_auth_attempts` 保持接近零。

使用 RDS Proxy 連線集區時，連線嘗試會在工作負載開始時增加。設定連線集區之後，平均值會下降。交易和後端使用所使用的資源在整個工作負載處理過程中保持一致。



如需搭配 Aurora PostgreSQL 資料庫叢集使用 Performance Insights 的詳細資訊，請參閱 在 Amazon Aurora 上使用績效詳情監控資料庫負載。若要分析指標，請參閱 使用績效詳情儀表板來分析指標。

展示連線集區的好處

如前所述，如果您判斷 Aurora PostgreSQL 資料庫叢集有連線流失問題，您可以使用 RDS Proxy 來改善效能。接下來，您可以找到一個範例，其中顯示連線置於集區以及不置於集區時處理工作負載的差異。此範例使用 pgbench 建立交易工作負載的模型。

與 psql 一樣，pgbench 是一個 PostgreSQL 用戶端應用程式，您可以在本地用戶端機器上安裝和運行。您也可以從用來管理 Aurora PostgreSQL 資料庫叢集的 Amazon EC2 執行個體進行安裝和執行。如需詳細資訊，請參閱 PostgreSQL 文件中的 pgbench。

若要逐步執行這個範例，您必須先在資料庫中建立 pgbench 環境。下面的命令是初始化指定資料庫中 pgbench 資料表的基本範本。此範例使用預設的主要使用者帳戶，`postgres`，用於登入。視需要針對您的 Aurora PostgreSQL 資料庫叢集進行變更。您在叢集的寫入器執行個體上的資料庫中建立 pgbench 環境。

> **Note**
>
> pgbench 初始化過程會刪除並重新建立名為 `pgbench_accounts`、`pgbench_branches`、`pgbench_history` 和 `pgbench_tellers` 的資料表。請確定當您初始化 pgbench 時，您選擇用於 *dbname* 的資料庫不會使用這些名稱。

```
pgbench -U postgres -h db-cluster-instance-1.111122223333.aws-region.rds.amazonaws.com
 -p 5432 -d -i -s 50 dbname
```

針對 pgbench，請指定下列參數：

-d

在 pgbench 執行時輸出偵錯報告。

-h

選擇 Aurora PostgreSQL 資料庫叢集的寫入器庫執行個體的端點。

-i

初始化資料庫中的 pgbench 環境，以進行基準測試。

-p

識別用於資料庫連線的連接埠。Aurora PostgreSQL 的預設值通常是 5432 或 5433。

-s

指定用於將資料列填入資料表的擴展係數。預設擴展係數為 1，會在 pgbench_branches 資料表中產生 1 個資料列，在 pgbench_tellers 資料表中產生 10 個資料列，以及在 pgbench_accounts 資料表中產生 10 萬個資料列。

-U

指定 Aurora PostgreSQL 資料庫叢集的寫入器執行個體的使用者帳戶。

設定 pgbench 環境後，您可以使用或不使用連線集區來執行基準測試。預設測試包含每筆交易在指定時間內重複執行的 SELECT、UPDATE 和 INSERT 的一系列五個命令。您可以指定擴展係數、用戶端數量和其他詳細資料，以建立您自己的使用案例的模型。

舉例來說，接下來的命令會以 20 個並行連線 (-c 選項) 執行基準測試 60 秒 (時間為-T 選項)。-C 選項每次都會使用新連線執行測試，而不是每個用戶端工作階段執行一次。此設定可提供連線額外負荷的指示。

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
 postgres -p 5432 -T 60 -c 20 -C labdb
Password:**********
pgbench (14.3, server 13.3)
  starting vacuum...end.
```

```
  transaction type: <builtin: TPC-B (sort of)>
    scaling factor: 50
    query mode: simple
    number of clients: 20
    number of threads: 1
    duration: 60 s
    number of transactions actually processed: 495
    latency average = 2430.798 ms
    average connection time = 120.330 ms
    tps = 8.227750 (including reconnection times)
```

在 Aurora PostgreSQL 資料庫叢集的寫入器執行個體上執行 pgbench，而不重複使用連線，表示每秒只會處理大約 8 筆交易。這在 1 分鐘的測試期間總共提供了 495 筆交易。

如果您重複使用連線，Aurora PostgreSQL 資料庫叢集對使用者數目的回應速度將快上近 20 倍。透過重複使用，系統會處理 9,042 筆交易，而在相同的時間量和相同數量的使用者連線中則處理 495 筆交易。不同之處在於：在以下情況中，每個連線都被重複使用。

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
 postgres -p 5432 -T 60 -c 20 labdb
Password:*********
pgbench (14.3, server 13.3)
      starting vacuum...end.
      transaction type: <builtin: TPC-B (sort of)>
      scaling factor: 50
      query mode: simple
      number of clients: 20
      number of threads: 1
      duration: 60 s
      number of transactions actually processed: 9042
      latency average = 127.880 ms
      initial connection time = 2311.188 ms
      tps = 156.396765 (without initial connection time)
```

這個範例說明，將連線置於集區可以大幅改善回應時間。如需針對 Aurora PostgreSQL 資料庫叢集設定 RDS Proxy 的詳細資訊，請參閱 Amazon RDS Proxy for Aurora。

PostgreSQL 中的無效連線處理

即使用戶端應用程式異常捨棄或終止，資料庫工作階段仍在伺服器上保持作用中狀態時，就會發生無效連線。當用戶端處理當機或意外終止，且未正確關閉其資料庫連線或取消持續請求時，通常會發生這種情況。

當伺服器程序閒置或嘗試將資料傳送至用戶端時，PostgreSQL 可有效識別和清除無效連線。不過，對於閒置、等待用戶端輸入或主動執行查詢的工作階段而言，偵測具有挑戰性。為了處理這些案例，PostgreSQL 提供 `tcp_keepalives_*`、`tcp_user_timeout` 和 `client_connection_check_interval` 參數。

主題

- 了解 TCP 保持連線
- Aurora PostgreSQL 中的金鑰 TCP 保持連線參數
- TCP 保持連線設定的使用案例
- 最佳實務

了解 TCP 保持連線

TCP 保持連線是一種通訊協定層級機制，可協助維護和確認連線完整性。每個 TCP 連線都會維護控制保持連線行為的核心層級設定。當保持連線計時器過期時，系統會執行下列動作：

- 傳送未設定資料和 ACK 旗標的探查封包。
- 根據 TCP/IP 規格，預期來自遠端端點的回應。
- 根據回應或缺乏回應來管理連線狀態。

Aurora PostgreSQL 中的金鑰 TCP 保持連線參數

參數	Description	預設值
`tcp_keepalives_idle`	Specifies number of seconds of inactivity before sending keepalive message.	300
`tcp_keepalives_interval`	Specifies number of seconds between retransmissions of unacknowledged keepalive messages.	30
`tcp_keepalives_count`	Maximum lost keepalive messages before declaring connection dead	2

參數	Description	預設值
tcp_user_timeout	Specifies how long (in Milliseconds) unacknowledged data can remain before forcibly closing the connection.	0
client_connection_check_interval	Sets the interval (in Milliseconds) for checking client connection status during long-running queries. This ensures quicker detection of closed connections.	0

TCP 保持連線設定的使用案例

保持閒置工作階段持續運作

若要防止防火牆或路由器因閒置而終止閒置連線：

- 設定 tcp_keepalives_idle 以定期傳送保持連線封包。

偵測無效連線

若要立即偵測無效連線：

- 調整 tcp_keepalives_idle、tcp_keepalives_interval 和 tcp_keepalives_count。例如，使用 Aurora PostgreSQL 預設值時，偵測無效連線大約需要一分鐘 (2 次探查 × 30 秒）。降低這些值可以加快偵測速度。
- 使用 tcp_user_timeout 指定確認的最長等待時間。

TCP 保持連線設定可協助核心偵測無效連線，但在使用通訊端之前，PostgreSQL 可能不會運作。如果工作階段執行長查詢，只有在查詢完成後才會偵測到無效連線。在 PostgreSQL 14 和更高版本中，client_connection_check_interval 可以在查詢執行期間定期輪詢通訊端，以加速無效連線偵測。

最佳實務

- 設定合理的保持連線間隔：調校 `tcp_user_timeout`、`tcp_keepalives_idle`、`tcp_keepalives_count` 和 `tcp_keepalives_interval`，以平衡偵測速度和資源使用。
- 為您的環境最佳化：使設定與網路行為、防火牆政策和工作階段需求保持一致。
- 利用 PostgreSQL 功能：在 PostgreSQL 14 和更高版本中使用 `client_connection_check_interval`，以進行有效的連線檢查。

調整 Aurora PostgreSQL 的記憶體參數

在 Amazon Aurora PostgreSQL 中，您可以使用數個參數來控制用於各種處理任務的記憶體容量。如果工作佔用的記憶體超過為指定參數設定的量，Aurora PostgreSQL 會使用其他資源進行處理，例如寫入磁碟。這可能會導致您的 Aurora PostgreSQL 資料庫叢集變慢或可能停止，並發生記憶體不足錯誤。

每個記憶體參數的預設設定通常可以處理其預期的處理工作。然而，您也可以調整 Aurora PostgreSQL 資料庫叢集 的記憶體相關參數。您可以執行此調整，以確保為處理特定工作負載配置了足夠的記憶體。

您可以在下方找到有關控制記憶體管理的參數的資訊。您也可以了解如何評估記憶體使用率。

檢查和設定參數值

您可以設定用來管理記憶體和評估 Aurora PostgreSQL 資料庫叢集記憶體使用量的參數如下：

- `work_mem` – 指定 Aurora PostgreSQL 資料庫叢集在寫入暫存磁碟檔案之前，用於內部排序操作和雜湊表的記憶體數量。
- `log_temp_files` – 記錄臨時檔案建立、檔案名稱和大小。此參數開啟時，會為建立的每個暫存檔儲存一個日誌項目。開啟此選項可查看您的 Aurora PostgreSQL 資料庫叢集需要寫入磁碟有多頻繁。收集有關 Aurora PostgreSQL 資料庫叢集產生暫存檔的資訊後，請再次將其關閉，以避免過度記錄。
- `logical_decoding_work_mem` – 指定每個內部重新排序緩衝區在溢出到磁碟之前要使用的記憶體量 (以 KB 為單位)。此記憶體用於邏輯解碼，這是用來建立複本的程序。它會藉由將資料從預寫日誌 (WAL) 檔案轉換為目標所需的邏輯串流輸出來完成。

此參數的值會為每個複寫連線建立指定大小的單一緩衝區。根據預設，其為 65536 KB。此緩衝區填滿後，多餘的內容會以檔案的形式寫入磁碟。若要將磁碟活動最小化，您可以將此參數的值設定為遠遠高於 work_mem。

這些都是動態參數，因此您可以針對目前工作階段變更它們。若要這麼做，請使用 psql 連線到 Aurora PostgreSQL 資料庫叢集 ，並使用 SET 陳述式，如下所示。

```
SET parameter_name TO parameter_value;
```

工作階段設定僅限工作階段期間內有效。工作階段結束時，參數會回復為資料庫叢集參數群組中的設定。 (資料庫參數群組) 在變更任何參數之前，請先查詢 pg_settings 資料表，檢查目前的值，如下所示。

```
SELECT unit, setting, max_val
   FROM pg_settings WHERE name='parameter_name';
```

例如，若要尋找 work_mem 參數的值，請連線至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，然後執行下列查詢。

```
SELECT unit, setting, max_val, pg_size_pretty(max_val::numeric)
  FROM pg_settings WHERE name='work_mem';
unit  | setting  | max_val    | pg_size_pretty
------+----------+------------+----------------
 kB   | 1024     | 2147483647| 2048 MB
(1 row)
```

變更參數設定，使其持續需要使用自訂資料庫叢集參數群組。 (資料庫參數群組) 以這些參數不同的值使用 SET 陳述式來訓練您的 Aurora PostgreSQL 資料庫叢集 後，您可以建立自訂參數群組，並套用至您的 Aurora PostgreSQL 資料庫叢集。如需詳細資訊，請參閱Amazon Aurora 的參數群組。

了解工作記憶參數

工作記憶參數 (work_mem) 指定 Aurora PostgreSQL 可用來處理複雜查詢的最大記憶體容量。複雜的查詢包括涉及排序或分組作業的查詢 - 換句話說，使用下列子句的查詢：

- ORDER BY
- DISTINCT
- GROUP BY

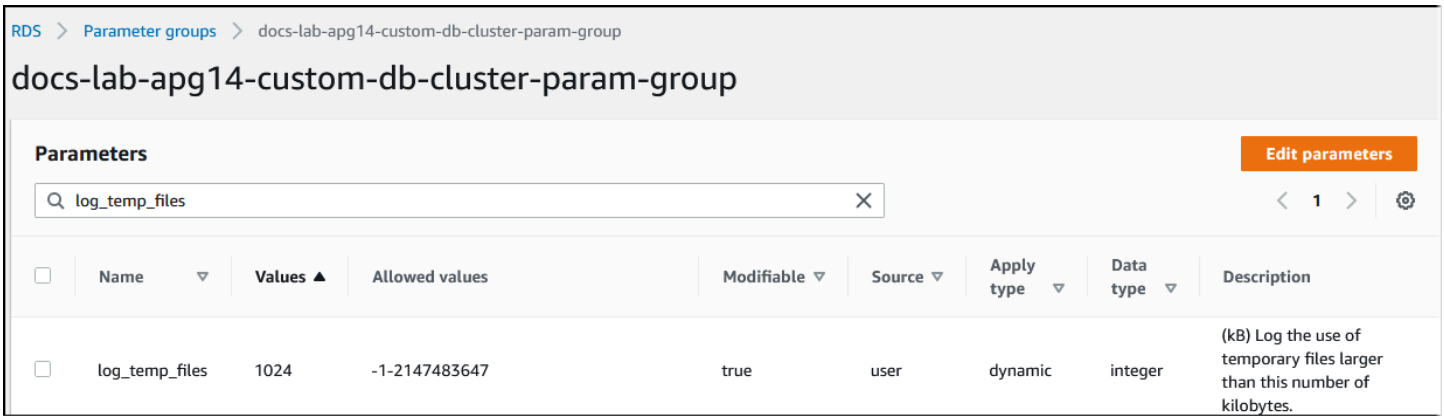
- JOIN (MERGE 和 HASH)

查詢規劃工具會間接影響您的 Aurora PostgreSQL 資料庫叢集使用工作記憶體的方式。查詢規劃程式會產生處理 SQL 陳述式的執行計劃。給定的計劃可能會將複雜的查詢分解為可以平行運行的多個工作單元。如果可能的話，Aurora PostgreSQL 在每個平行程序寫入磁碟之前，會使用在 work_mem 參數中為每個工作階段指定的記憶體的量。

多個資料庫使用者同時執行多項作業，且平行產生多個工作單位，可能會耗盡 Aurora PostgreSQL 資料庫叢集分配的工作記憶體。這可能會導致建立的暫存檔和磁碟 I/O 過多，或者更糟糕的是，可能導致記憶體不足錯誤。

識別暫存檔案的使用

當處理查詢所需的記憶體超過 work_mem 參數中指定的值，工作資料會卸載到磁碟中的暫存檔。您可以開啟 log_temp_files 參數，以查看發生這種情況的頻率。此參數預設會被關閉 (被設定為 -1)。若要擷取所有暫存檔資訊，請將此參數設定為 0。將 log_temp_files 設定為任何其他正整數，以擷取資料量等於或大於該值之檔案 (以 KB 為單位) 的暫存檔資訊。在下圖中，您可以看到來自 的範例 AWS 管理主控台。



設定暫存檔記錄功能後，您可以使用自己的工作負載進行測試，以查看工作記憶體設定是否足夠。您也可以使用 pgbench (PostgreSQL 社群提供的簡單基準測試應用程式) 來模擬工作負載。

以下範例會建立運行測試所需的資料表和資料列，以初始化 (-i) pgbench。在此範例中，擴展係數 (-s 50) 會在 pgbench_branches 資料表中建立 50 個資料列，在 pgbench_tellers 中建立 500 個資料列，以及在 labdb 資料庫的 pgbench_accounts 資料表中建立 5,000,000 資料列。

```
pgbench -U postgres -h your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com
 -p 5432 -i -s 50 labdb
Password:
dropping old tables...
```

```
NOTICE:  table "pgbench_accounts" does not exist, skipping
NOTICE:  table "pgbench_branches" does not exist, skipping
NOTICE:  table "pgbench_history" does not exist, skipping
NOTICE:  table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
5000000 of 5000000 tuples (100%) done (elapsed 15.46 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 61.13 s (drop tables 0.08 s, create tables 0.39 s, client-side generate 54.85
 s, vacuum 2.30 s, primary keys 3.51 s)
```

初始化環境後，您可以針對特定時間 (-T) 和用戶端數量 (-c) 執行基準測試。此範例還會使用當 Aurora PostgreSQL 資料庫叢集處理交易時輸出除錯資訊的 -d 選項。

```
pgbench -h -U postgres your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com
 -p 5432 -d -T 60 -c 10 labdb
Password:*******
pgbench (14.3)
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 50
query mode: simple
number of clients: 10
number of threads: 1
duration: 60 s
number of transactions actually processed: 1408
latency average = 398.467 ms
initial connection time = 4280.846 ms
tps = 25.096201 (without initial connection time)
```

如需 pg_upgrade 的詳細資訊，請參閱 PostgreSQL 文件中的 pgbench。

您可以使用 psql 元資料命令 (\d) 列出 pgbench 建立的關聯，例如資料表、檢視和索引。

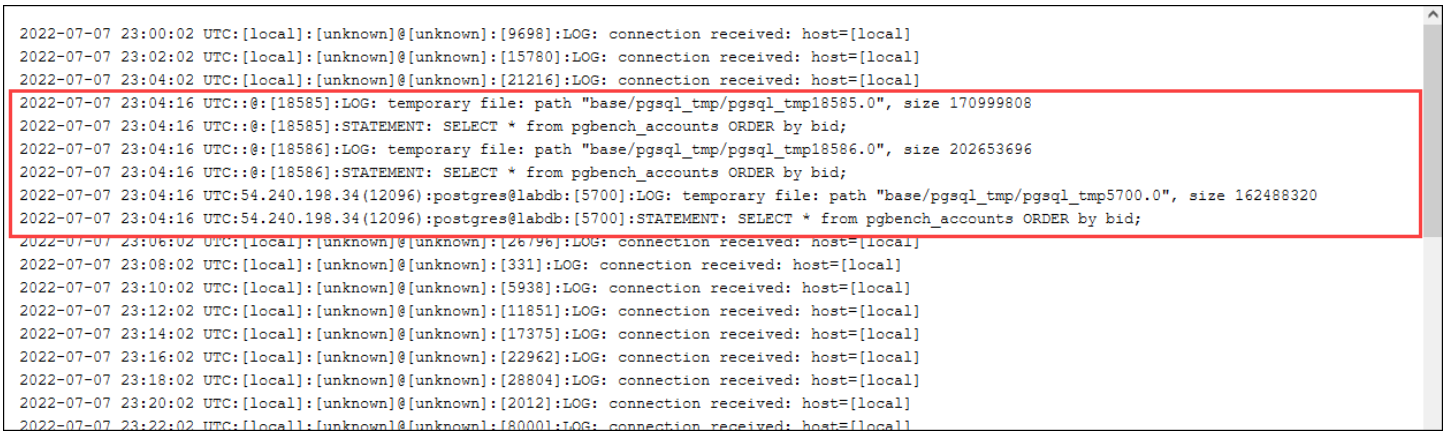
```
labdb=>  \d pgbench_accounts
 Table "public.pgbench_accounts"
  Column  |     Type      | Collation | Nullable | Default
----------+---------------+-----------+----------+---------
 aid      | integer       |           | not null |
 bid      | integer       |           |          |
 abalance | integer       |           |          |
 filler   | character(84) |           |          |
```

```
Indexes:
    "pgbench_accounts_pkey" PRIMARY KEY, btree (aid)
```

如輸出所示，pgbench_accounts 資料表的 aid 資料欄建立了索引。若要確保下一個查詢使用工作記憶體，請查詢任何非索引資料欄，例如下列範例中所示。

```
postgres=> SELECT * FROM pgbench_accounts ORDER BY bid;
```

檢查臨時檔案的日誌。若要這樣做，請開啟 AWS 管理主控台，選擇 Aurora PostgreSQL 資料庫叢集執行個體，然後選擇日誌與事件索引標籤。在主控台中檢視日誌，或下載以供進一步分析。如下圖所示，處理查詢所需的暫存檔大小表示您應該考慮增加針對 work_mem 參數指定的量。



您可以根據您的營運需求，針對個人和群組以不同的方式設定此參數。例如，您可以針對名為 dev_team 的角色將 work_mem 參數設定為 8 GB。

```
postgres=> ALTER ROLE dev_team SET work_mem='8GB';
```

使用此一 work_mem 的設定，屬於 dev_team 角色成員的任何角色都會分配到 8 GB 的工作內存。

使用索引加快回應時間

如果您的查詢傳回結果花費的時間太長，您可以驗證索引是否如預期般使用。首先，請啟用 \timing，psql 元命令，如下所示。

```
postgres=>  \timing on
```

開啟計時後，請使用簡單的 SELECT 陳述式。

```
postgres=> SELECT COUNT(*) FROM
```

```
  (SELECT * FROM pgbench_accounts
  ORDER BY bid)
  AS accounts;
count
-------
5000000
(1 row)
Time: 3119.049 ms (00:03.119)
```

如輸出所示，此查詢需要 3 秒以上才能完成。若要縮短回應時間，請針對 pgbench_accounts 建立索引，如下所示。

```
postgres=> CREATE INDEX ON pgbench_accounts(bid);
CREATE INDEX
```

重新執行查詢，並注意回應時間較快。在這個範例中，完成查詢大約需要半秒鐘，快了大約 5 倍。

```
postgres=>  SELECT COUNT(*) FROM (SELECT * FROM pgbench_accounts ORDER BY bid) AS
 accounts;
 count
-------
 5000000
(1 row)
Time: 567.095 ms
```

調整邏輯解碼的工作記憶體

邏輯複製自從在 PostgreSQL 第 10 版引入以來，已在 Aurora PostgreSQL 的所有版本中使用。設定邏輯複製時，您也可以設定 logical_decoding_work_mem 參數，指定邏輯解碼程序可用於解碼和串流處理程序的記憶體數量。

在邏輯解碼期間，預寫日誌 (WAL) 記錄會轉換為 SQL 陳述式，接著會傳送至另一個目標以進行邏輯複寫或其他工作。當交易寫入 WAL 然後進行轉換時，整個交易必須符合針對 logical_decoding_work_mem 指定的值。此參數預設為 65536 KB。任何溢出都會寫入磁碟。這意味著必須先從磁碟重新讀取，然後才能傳送到目的地，這樣會減緩整個過程。

您可以使用 aurora_stat_file 函數評估目前工作負載在特定時間點的交易溢出量，如以下範例所示。

```
SELECT split_part (filename, '/', 2)
   AS slot_name, count(1) AS num_spill_files,
```

```
   sum(used_bytes) AS slot_total_bytes,
   pg_size_pretty(sum(used_bytes)) AS slot_total_size
   FROM aurora_stat_file()
   WHERE filename like '%spill%'
   GROUP BY 1;
  slot_name | num_spill_files | slot_total_bytes | slot_total_size
------------+-----------------+------------------+-----------------
 slot_name  |       590       |      411600000   | 393 MB
(1 row)
```

呼叫查詢時，此查詢會傳回 Aurora PostgreSQL 資料庫叢集上溢出檔案的計數和大小。執行時間較長的工作負載在磁碟上可能沒有任何溢出檔案。若要針對長時間執行的工作負載進行效能分析，建議您建立一個資料表，以便在工作負載執行時擷取溢出檔案資訊。您可以建立資料表，如下所示。

```
CREATE TABLE spill_file_tracking AS
    SELECT now() AS spill_time,*
    FROM aurora_stat_file()
    WHERE filename LIKE '%spill%';
```

若要查看邏輯複寫期間如何使用溢出檔案，請設定發布者和訂閱者，然後啟動簡單複寫。如需詳細資訊，請參閱針對 Aurora PostgreSQL 資料庫叢集設定邏輯複寫。進行複寫時，您可以建立一項工作，從 `aurora_stat_file()` 溢出檔函數擷取結果集，如下所示。

```
INSERT INTO spill_file_tracking
  SELECT now(),*
  FROM aurora_stat_file()
  WHERE filename LIKE '%spill%';
```

使用以下 psql 命令每秒運行一次作業。

```
\watch 0.5
```

工作執行時，從另一個 psql 工作階段連線到寫入器執行個體。使用以下一系列陳述式來執行超出記憶體組態的工作負載，並讓 Aurora PostgreSQL 建立溢出檔案。

```
labdb=> CREATE TABLE my_table (a int PRIMARY KEY, b int);
CREATE TABLE
labdb=> INSERT INTO my_table SELECT x,x FROM generate_series(0,10000000) x;
INSERT 0 10000001
labdb=> UPDATE my_table SET b=b+1;
```

```
UPDATE 10000001
```

這些陳述式需要幾分鐘的時間來完成。完成後，同時按 Ctrl 鍵和 C 鍵停止監視功能。然後使用下列命令建立資料表，以保存 Aurora PostgreSQL 資料庫叢集溢出檔案使用情況的相關資訊。

```
SELECT spill_time, split_part (filename, '/', 2)
    AS slot_name, count(1)
    AS spills, sum(used_bytes)
    AS slot_total_bytes, pg_size_pretty(sum(used_bytes))
    AS slot_total_size FROM spill_file_tracking
  GROUP BY 1,2 ORDER BY 1;
                  spill_time | slot_name             | spills | slot_total_bytes |
 slot_total_size
-------------------------------+-----------------------+--------+------------------
+-----------------
2022-04-15 13:42:52.528272+00 | replication_slot_name | 1      | 142352280        | 136
 MB
2022-04-15 14:11:33.962216+00 | replication_slot_name | 4      | 467637996        | 446
 MB
2022-04-15 14:12:00.997636+00 | replication_slot_name | 4      | 569409176        | 543
 MB
2022-04-15 14:12:03.030245+00 | replication_slot_name | 4      | 569409176        | 543
 MB
2022-04-15 14:12:05.059761+00 | replication_slot_name | 5      | 618410996        | 590
 MB
2022-04-15 14:12:07.22905+00  | replication_slot_name | 5      | 640585316        | 611
 MB
(6 rows)
```

輸出顯示執行範例會建立五個使用 611 MB 記憶體的溢出檔案。若要避免寫入磁碟，建議將 `logical_decoding_work_mem` 參數設定為下一個最高記憶體大小，亦即 1024。

使用 Amazon CloudWatch 指標來分析 Aurora PostgreSQL 的資源用量

Aurora 每隔 1 分鐘會自動將指標資料傳送至 CloudWatch。您可以使用 CloudWatch 指標來分析 Aurora PostgreSQL 的資源用量。您可以使用指標來評估網路輸送量和網路用量。

使用 CloudWatch 評估網路輸送量

當您的系統用量接近執行個體類型的資源限制時，處理速度可能會變慢。您可以使用 CloudWatch Logs Insights (日誌洞察)，來監控儲存體資源用量，並確保有足夠的資源可用。需要時，您可以將資料庫執行個體修改為更大的執行個體類別。

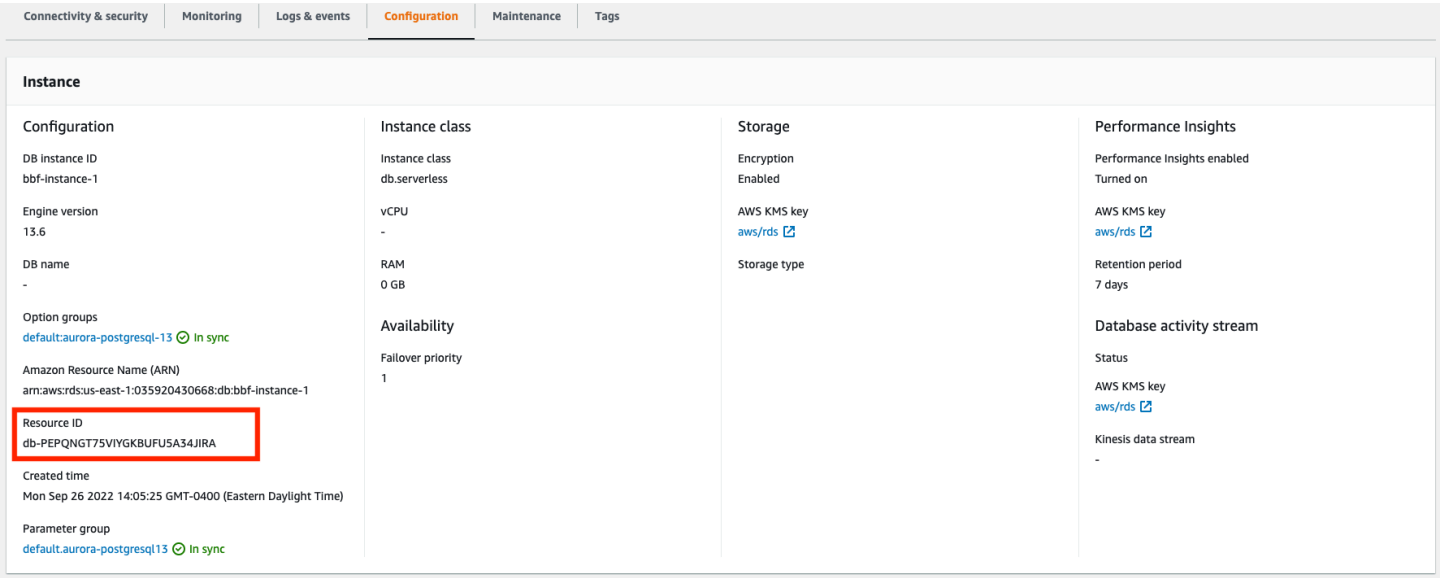
Aurora 儲存體處理速度可能由於下列原因而變慢：

- 用戶端與資料庫執行個體之間的網路頻寬不足。
- 儲存體子系統的網路頻寬不足。
- 對於您的執行個體類型而言很大的工作負載。

您可以查詢 CloudWatch Logs Insights (日誌洞察)，來產生 Aurora 儲存體資源用量的圖形以監控資源。此圖會顯示 CPU 使用率和指標，以協助您決定是否要縱向擴展至較大的執行個體大小。如需 CloudWatch Logs Insights (日誌洞察) 查詢語法的相關資訊，請參閱 CloudWatch Logs Insights 查詢語法。

若要使用 CloudWatch，您需要將 Aurora PostgreSQL 日誌檔匯出到 CloudWatch。您也可以修改現有的叢集，以將日誌匯出到 CloudWatch。如需將日誌匯出至 CloudWatch 的相關資訊，請參閱 開啟將日誌發佈到 Amazon CloudWatch 的選項。

您需要資料庫執行個體的 Resource ID (資源 ID)，才能查詢 CloudWatch Logs Insights (日誌洞察)。Resource ID (資源 ID) 可在主控台的 Configuration (組態) 索引標籤中找到：



若要查詢資源儲存體指標的日誌檔：

1. 透過 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。

   CloudWatch 概觀首頁隨即顯示。
2. 如有需要，請變更 AWS 區域。在導覽列中，選擇 AWS 區域 AWS 資源所在的 。如需詳細資訊，請參閱 Regions and endpoints。

3. 在導覽窗格中，選擇 Logs (日誌)，然後選擇 Logs Insights (日誌洞察)。

   Logs Insights (日誌洞察) 頁面即會出現。

4. 從下拉式清單選取要分析的日誌檔。

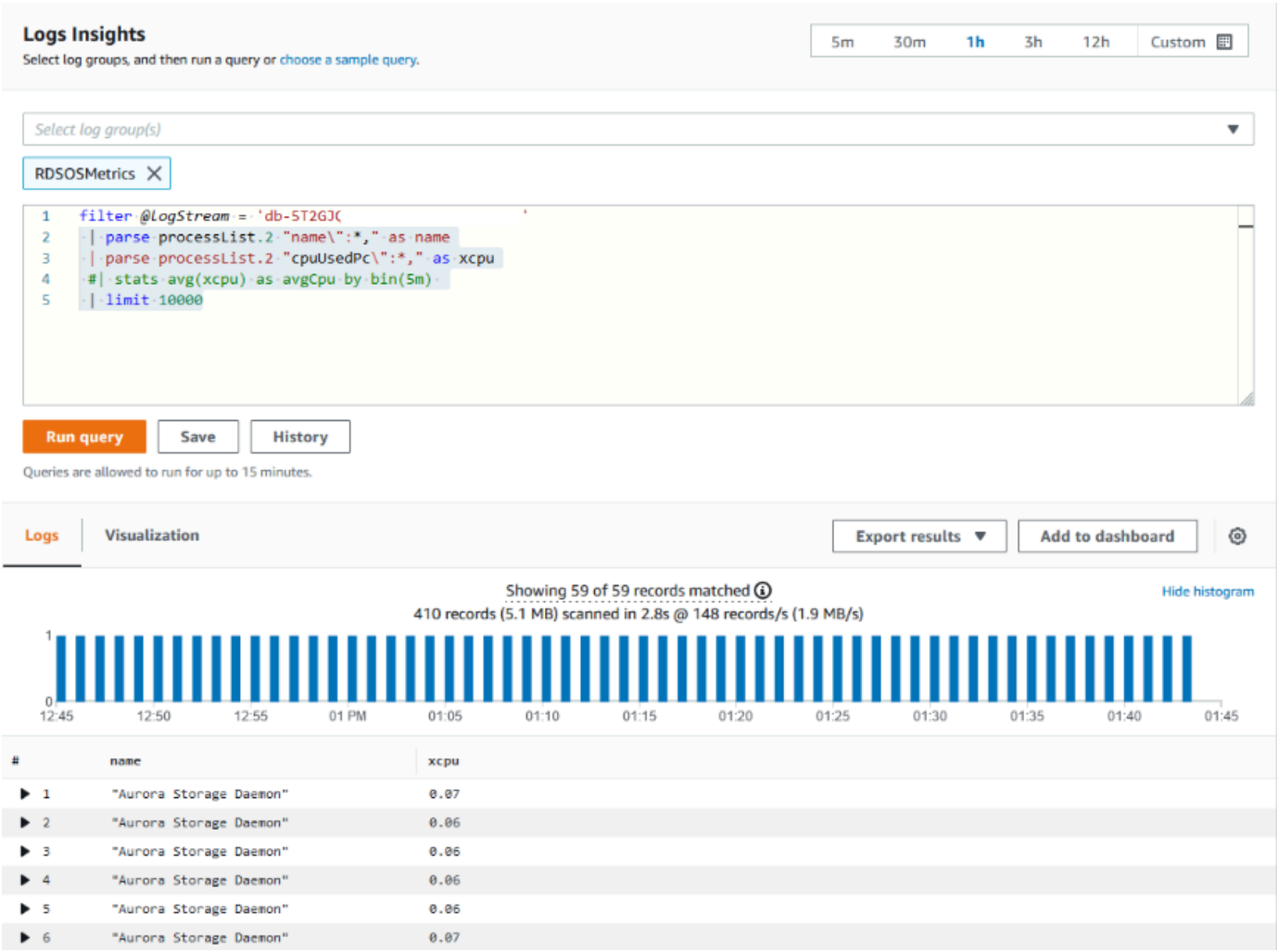
5. 在欄位中輸入下列查詢，將 <resource ID> 取代為資料庫叢集的資源 ID：

```
filter @logStream = <resource ID> | parse @message "\"Aurora Storage Daemon\"*memoryUsedPc\":*,\"cpuUsedPc\":*," as a,memoryUsedPc,cpuUsedPc | display memoryUsedPc,cpuUsedPc #| stats avg(xcpu) as avgCpu by bin(5m) | limit 10000
```

6. 按一下 Run query (執行查詢)。

   即會顯示儲存體使用率圖形。

   下圖提供 Logs Insights (日誌洞察) 頁面和圖形顯示。



使用 CloudWatch 指標評估 Aurora PostgreSQL 的資料庫執行個體用量

您可以使用 CloudWatch 指標來監看執行個體輸送量，並探索執行個體類別是否為您的應用程式提供足夠的資源。如需資料庫執行個體類別限制的相關資訊，請前往 Aurora 資料庫執行個體類別的硬體規格並尋找資料庫執行個體類別的規格，以找出您的網路效能。

如果您的資料庫執行個體用量接近執行個體類別限制，效能可能會開始變慢。CloudWatch 指標可以確認這種情況，因此您可以規劃手動縱向擴展到更大的執行個體類別。

結合下列 CloudWatch 指標值，以了解您是否接近執行個體類別限制：

- NetworkThroughput - 用戶端針對 Aurora MySQL 資料庫叢集中各個執行個體所接收和傳輸的網路輸送量。此輸送量值不包含資料庫叢集中的執行個體與叢集磁碟區之間的網路流量。
- StorageNetworkThroughput - Aurora MySQL 資料庫叢集中各個執行個體接收自和傳送至 Aurora 儲存子系統的網路輸送量。

將 NetworkThroughput 新增至 StorageNetworkThroughput，以尋找 Aurora MySQL 資料庫叢集中各個執行個體接收自和傳送至 Aurora 儲存子系統的網路輸送量。執行個體的執行個體類別限制應該大於這兩個合併指標之和。

在傳送和接收時，您可以使用下列指標，來檢閱來自用戶端應用程式之網路流量的其他詳細資訊：

- NetworkReceiveThroughput - 資料庫叢集中的各個執行個體從用戶端接收到的網路輸送量。此輸送量不包含資料庫叢集中的執行個體與叢集磁碟區之間的網路流量。
- NetworkTransmitThroughput - Aurora 資料庫叢集中的各個執行個體傳送至用戶端的網路輸送量。此輸送量不包含資料庫叢集中的執行個體與叢集磁碟區之間的網路流量。
- StorageNetworkReceiveThroughput - 資料庫叢集中的各個執行個體從 Aurora 儲存子系統接收到的網路輸送量。
- StorageNetworkTransmitThroughput - 資料庫叢集中的各個執行個體傳送至 Aurora 儲存子系統的網路輸送量。

將所有這些指標加在一起，以評估您的網路用量與執行個體類別限制的比較情形。執行個體類別限制應該大於這些個合併指標之和。

儲存體的網絡限制和 CPU 使用率是交互的。當網路輸送量增加時，CPU 使用率也會增加。監控 CPU 和網路用量可提供資源如何和為何耗盡的相關資訊。

若要協助將網路用量降至最低，您可以考慮：

- 使用更大的執行個體類別。
- 使用 pg_partman 分割策略。
- 批次分割寫入要求，以減少整體交易。
- 將唯讀工作負載導向至唯讀執行個體。
- 刪除任何未使用的索引。
- 檢查是否有膨脹的物體和 VACUUM。若有嚴重膨脹，請使用 PostgreSQL 延伸模組 pg_repack。如需 pg_repack 的詳細資訊，請參閱以最少的鎖定重組 PostgreSQL 資料庫中的資料表。

使用邏輯複寫來執行 Aurora PostgreSQL 的主要版本升級

使用邏輯複寫和 Aurora 快速複製，您可使用目前版本 Aurora PostgreSQL 資料庫執行主要版本升級，同時逐漸將變更資料遷移至新的主要版本資料庫。此低停機時間升級程序稱為藍/綠升級。資料庫的目前版本稱為「藍色」環境，而新的資料庫版本則稱為「綠色」環境。

Aurora 快速複製會透過取得來源資料庫的快照，完全載入現有資料。快速複製會使用建置在 Aurora 儲存層之上的寫入時複製通訊協定，這可讓您在短時間內建立資料庫複製。升級到大型資料庫時，此方法非常有效。

PostgreSQL 中的邏輯複寫會追蹤您的資料變更，並將其從初始執行個體傳輸到並行執行的新執行個體，直到您移至更新版本的 PostgreSQL 為止。邏輯複寫使用發佈與訂閱模型。如需有關 Aurora PostgreSQL 邏輯複寫的詳細資訊，請參閱 以 Amazon Aurora PostgreSQL 進行複寫。

> **Tip**
>
> 您可以使用受管 Amazon RDS 藍/綠部署功能，將主要版本升級所需的停機時間降至最低。如需詳細資訊，請參閱使用 Amazon Aurora 藍/綠部署進行資料庫更新。

主題

- 需求
- 限制
- 設定和檢查參數值
- 將 Aurora PostgreSQL 引擎升級至新的主要版本
- 執行升級後任務

需求

您必須符合下列需求，才能執行此低停機時間升級程序：

- 您必須具有 rds_superuser 許可。
- 您要升級的 Aurora PostgreSQL 資料庫叢集必須執行受支援的版本，其可使用邏輯複寫執行主要版本升級。務必將任何次要版本更新和修補程式套用至資料庫叢集。下列版本的 Aurora PostgreSQL 支援此技術中使用的 `aurora_volume_logical_start_lsn` 函數：
  - 15.2 版和更新的 15 版本
  - 14.3 版和更新的 14 版
  - 13.6 版及更新的第 13 版本
  - 12.10 版和更新的第 12 版本
  - 11.15 和更新的第 11 版本
  - 10.20 和更新的第 10 版本

  如需有關 `aurora_volume_logical_start_lsn` 函數的詳細資訊，請參閱 aurora_volume_logical_start_lsn。
- 您的所有資料表都必須具有主索引鍵或包含 PostgreSQL 身分資料欄。
- 針對您的 VPC 設定安全群組，以允許兩個 Aurora PostgreSQL 資料庫叢集 (無論新舊) 之間的傳入和傳出存取。您可以將存取權授與特定無類別域間路由 (CIDR) 範圍，或是授予 VPC 或對等 VPC 中的另一個安全群組。(對等 VPC 需要 VPC 對等互連。)

> **Note**
>
> 如需設定和管理執行中邏輯複寫案例所需許可的詳細資訊，請參閱 PostgreSQL 核心文件。

限制

在對 Aurora PostgreSQL 資料庫叢集執行低停機時間升級以將其升級到新的主要版本時，您使用的是原生 PostgreSQL 邏輯複寫功能。它具有與 PostgreSQL 邏輯複寫相同的功能和限制。如需詳細資訊，請參閱 PostgreSQL 邏輯複寫。

- 不會複寫資料定義語言 (DDL) 命令。
- 複寫不支援即時資料庫中的結構描述變更。在複製過程中，會以其原始形式重新建立結構描述。如果您在複製之後，但在完成升級之前變更結構描述，該結構描述不會反映在升級的執行個體中。

- 不會複製大型物件，但您可以將資料儲存在一般資料表中。
- 只有資料表 (包括分割的資料表) 支援複寫。不支援複寫至其他類型的關係，例如視觀表、具體化視觀表或外部資料表。
- 不會複寫序列資料，且其需要在容錯移轉後手動更新。

> **Note**
>
> 此升級不支援自動編寫指令碼。您應該手動執行所有步驟。

設定和檢查參數值

升級之前，請先設定 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，以充當發佈伺服器。執行個體應該搭配下列設定使用自訂資料庫叢集參數群組：

- `rds.logical_replication` – 將此參數設為 1。`rds.logical_replication` 參數提供的用途與獨立 PostgreSQL 伺服器的 `wal_level` 參數和其他控制預寫日誌檔管理的參數相同。
- `max_replication_slots` – 將此參數值設為您計畫建立的訂閱總數。如果您使用的是 AWS DMS，請將此參數設定為您計劃用於此資料庫叢集變更資料擷取 AWS DMS 的任務數量。
- `max_wal_senders` – 設為並行連線的數目，加上一些額外的數目，以供管理任務和新工作階段使用。如果您使用的是 AWS DMS，max_wal_senders 的數量應該等於並行工作階段的數量，以及在任何指定時間可能運作 AWS DMS 的任務數量。
- `max_logical_replication_workers` – 設為您預期的邏輯複寫工作者和資料表同步工作者的數目。將複寫工作者的數目設為用於 max_wal_senders 的相同值。通常很安全。工作者是從配置給伺服器的背景程序 (max_worker_process) 集區中取得的。
- `max_worker_processes` – 設為伺服器的背景程序數目。此數目應該大到足以配置工作者進行複寫、自動清空程序，以及其他可能同時發生的維護程序。

升級到更新版本的 Aurora PostgreSQL 時，您需要複製已在舊版參數群組中修改的任何參數。這些參數會套用至升級的版本。您可以查詢 `pg_settings` 資料表以取得參數設定清單，以便可在新的 Aurora PostgreSQL 資料庫叢集上重新建立參數設定。

例如，若要取得複寫參數的設定，請執行下列查詢：

```
SELECT name, setting FROM pg_settings WHERE name in
('rds.logical_replication', 'max_replication_slots',
```

```
'max_wal_senders', 'max_logical_replication_workers',
'max_worker_processes');
```

將 Aurora PostgreSQL 引擎升級至新的主要版本

準備發佈者 (藍色)

1. 在以下範例中，來源寫入器執行個體 (藍色) 是執行 PostgreSQL 11.15 版的 Aurora PostgreSQL 資料庫叢集。這是我們複寫案例中的發佈節點。針對此示範，我們的來源寫入器執行個體會管理一個保留一系列值的範例資料表：

   ```
   CREATE TABLE my_table (a int PRIMARY KEY);
   INSERT INTO my_table VALUES (generate_series(1,100));
   ```

2. 若要在來源執行個體上建立發佈，請使用 psql (PostgreSQL 的 CLI) 或使用您選擇的用戶端連線至執行個體的寫入器節點。在每個資料庫中輸入下列命令：

   ```
   CREATE PUBLICATION publication_name FOR ALL TABLES;
   ```

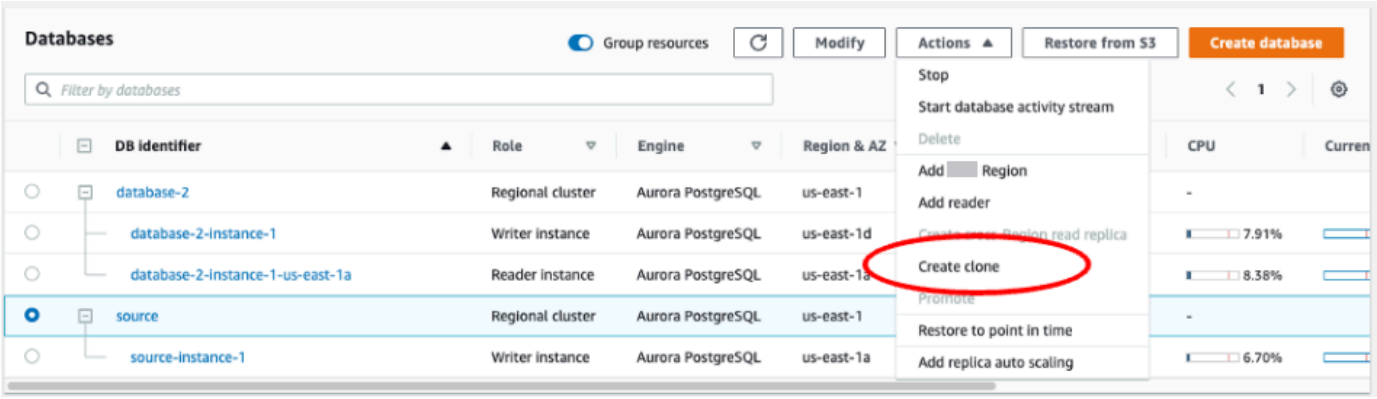
   publication_name 指定發佈的名稱。

3. 您也需要在執行個體上建立複寫槽。下列命令會建立複寫槽並載入 pgoutput 邏輯解碼外掛程式。外掛程式會將從預寫日誌 (WAL) 讀取的內容變更為邏輯複寫通訊協定，並根據發佈規格篩選資料。

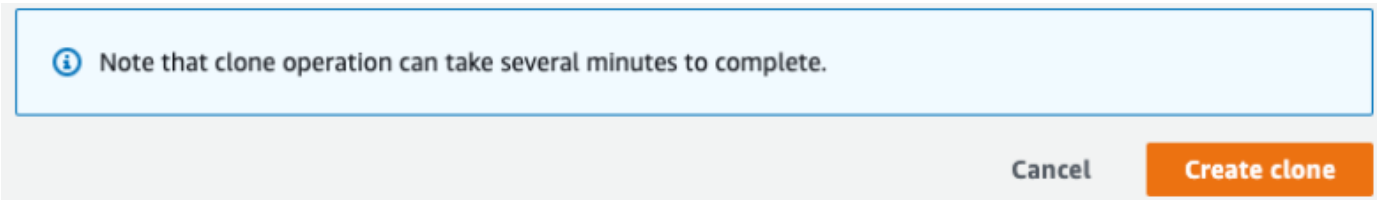
   ```
   SELECT pg_create_logical_replication_slot('replication_slot_name', 'pgoutput');
   ```

複製發佈者

1. 使用 Amazon RDS 主控台建立來源執行個體的複製。反白顯示 Amazon RDS 主控台中的執行個體名稱，然後在 Actions (動作) 功能表中選擇 Create clone (建立複製)。



2. 提供執行個體的唯一名稱。大部分的設定都是來源執行個體中的預設值。當您對新執行個體進行了必要的變更時，請選擇 Create clone (建立複製)。



3. 當目標執行個體啟動時，寫入器節點的 Status (狀態) 資料欄會在 Status (狀態) 資料欄中顯示 Creating (建立中)。當執行個體準備就緒時，狀態會變更為 Available (可用)。

準備複製以進行升級

1. 複製是部署模型中的「綠色」執行個體。它是複寫訂閱節點的主機。當節點變成可用時，請與 psql 連線，並查詢新的寫入器節點，以取得日誌序號 (LSN)。LSN 識別 WAL 串流中記錄的開頭。

```
SELECT aurora_volume_logical_start_lsn();
```

2. 在查詢的回應中，您可以找到 LSN 號碼。您稍後在此程序中需要此號碼，因此請記下它。

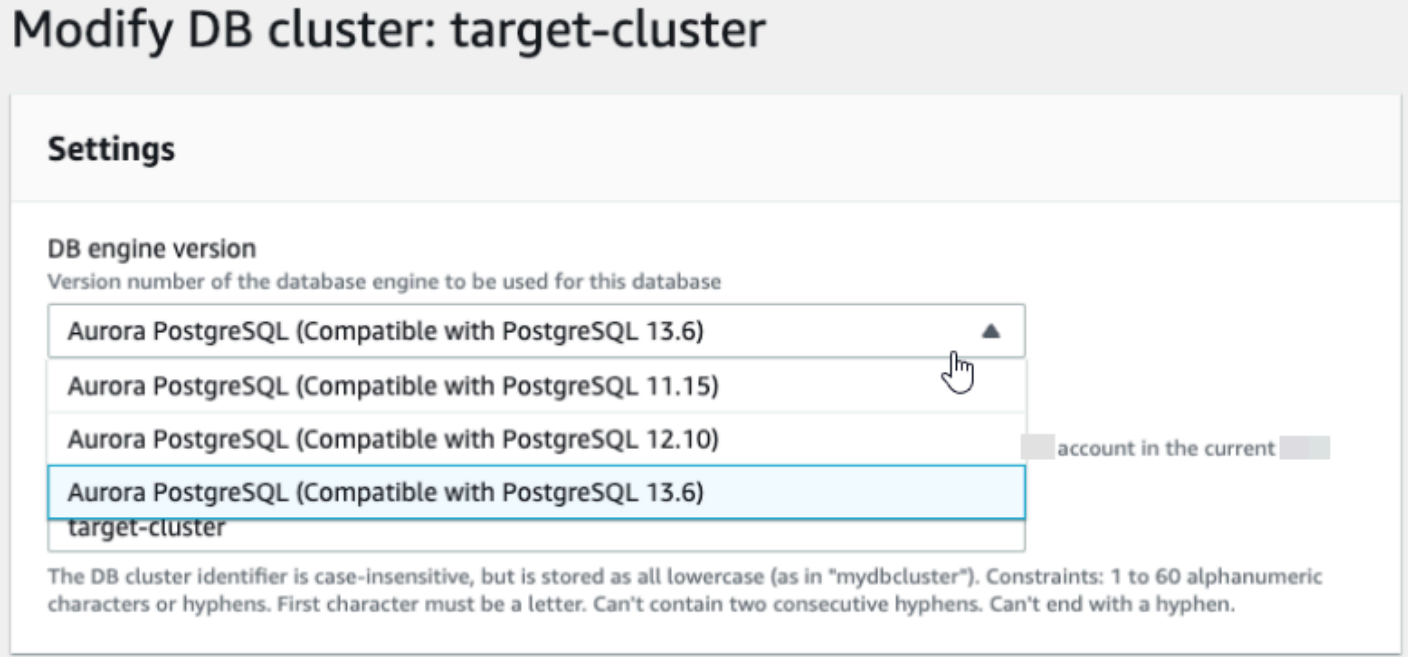
```
postgres=> SELECT aurora_volume_logical_start_lsn();
aurora_volume_logical_start_lsn
---------------
0/402E2F0
(1 row)
```

3. 升級複製之前，請先捨棄複製的複寫槽。

```
SELECT pg_drop_replication_slot('replication_slot_name');
```

將叢集升級至新的主要版本

- 在複製提供者節點之後，請使用 Amazon RDS 主控台，在訂閱節點上啟動主要版本升級。反白顯示 RDS 主控台中的執行個體名稱，然後選取 Modify (修改) 按鈕。選取更新的版本和更新的參數群組，然後立即套用設定以升級目標執行個體。



- 您也可以使用 CLI 執行升級：

```
aws rds modify-db-cluster –db-cluster-identifier $TARGET_Aurora_ID –engine-version
 13.6 –allow-major-version-upgrade –apply-immediately
```

準備訂閱者 (綠色)

1. 升級完成後，當複製變成可用時，請與 psql 連線並定義訂閱。若要這樣做，您需要在 CREATE SUBSCRIPTION 命令中指定下列選項：

   - subscription_name – 訂閱的名稱。
   - admin_user_name – 具有 rds_superuser 許可的管理使用者名稱。
   - admin_user_password – 與管理使用者相關聯的密碼。
   - source_instance_URL – 發佈伺服器執行個體的 URL。
   - database – 訂閱伺服器將與其連線的資料庫。
   - publication_name – 發佈伺服器的名稱。

- replication_slot_name - 複寫槽的名稱。

```
CREATE SUBSCRIPTION subscription_name
CONNECTION 'postgres://admin_user_name:admin_user_password@source_instance_URL/
database' PUBLICATION publication_name
WITH (copy_data = false, create_slot = false, enabled = false, connect = true,
 slot_name = 'replication_slot_name');
```

2. 建立訂閱之後，請查詢 pg_replication_origin 檢視以擷取 roname 值，這是複寫來源的識別符。每個執行個體都有一個 roname：

```
SELECT * FROM pg_replication_origin;
```

例如：

```
postgres=>
SELECT * FROM pg_replication_origin;

roident | roname
---------+----------
1 | pg_24586
```

3. 提供您先前查詢發佈節點時所儲存的 LSN，以及從命令中訂閱節點 [INSTANCE] 傳回的 roname。此命令會使用 pg_replication_origin_advance 函數來指定日誌序列中進行複寫的起點。

```
SELECT pg_replication_origin_advance('roname', 'log_sequence_number');
```

roname 是 pg_replication_origin 檢視傳回的識別符。

log_sequence_number 是先前查詢 aurora_volume_logical_start_lsn 函數所傳回的值。

4. 然後，使用 ALTER SUBSCRIPTION... ENABLE 子句開啟邏輯複寫。

```
ALTER SUBSCRIPTION subscription_name ENABLE;
```

5. 此時，您可以確認複製正在運作中。將一值新增至發佈執行個體，然後確認該值複寫到訂閱節點。

然後，使用下列命令來監控發佈節點上的複寫延遲：

```
SELECT now() AS CURRENT_TIME, slot_name, active, active_pid,
 pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn)) AS diff_size, pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn) AS diff_bytes FROM pg_replication_slots WHERE slot_type =
 'logical';
```

例如：

```
postgres=> SELECT now() AS CURRENT_TIME, slot_name, active, active_pid,
 pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn)) AS diff_size, pg_wal_lsn_diff(pg_current_wal_lsn(),
 confirmed_flush_lsn) AS diff_bytes FROM pg_replication_slots WHERE slot_type =
 'logical';

current_time                   | slot_name             | active | active_pid |
 diff_size | diff_bytes
-------------------------------+-----------------------+--------+------------
+-----------+------------
2022-04-13 15:11:00.243401+00  | replication_slot_name | t      | 21854      | 136
 bytes | 136
(1 row)
```

您可以使用 `diff_size` 和 `diff_bytes` 值來監控複寫延遲。當這些值達到 0，複本即已跟上來源資料庫執行個體。

執行升級後任務

升級完成時，執行個體狀態會在主控台儀表板的 Status (狀態) 資料欄中顯示為 Available (可用)。在新執行個體上，建議您執行下列動作：

- 重新導向您的應用程式以指向寫入器節點。
- 新增讀取器節點以管理工作量，並在寫入器節點發生問題時提供高可用性。
- Aurora PostgreSQL 資料庫叢集偶爾需要作業系統更新。這些更新可能會包含較新版本的 glibc 程式庫。在此類更新期間，建議您遵循 Aurora PostgreSQL 支援的定序 中所述指示。
- 更新新執行個體上的使用者許可來確保存取權。

在新執行個體上測試您的應用程式和資料之後，建議您先對初始執行個體進行最終備份，然後再將其移除。如需在 Aurora 主機上使用邏輯複寫的詳細資訊，請參閱 針對 Aurora PostgreSQL 資料庫叢集設定邏輯複寫。

在 Aurora PostgreSQL 中管理自訂轉換

PostgreSQL 中的類型轉換是將值從一個資料類型轉換為另一個資料類型的程序。PostgreSQL 提供許多常見轉換的內建轉換，但您也可以建立自訂轉換來定義特定類型轉換應如何運作。

轉換指定如何執行從一個資料類型到另一個資料類型的轉換。例如，將文字轉換為`'123'`整數 123，或將數值`45.67`轉換為文字 `'45.67'`。

如需 PostgreSQL 轉換概念和語法的完整資訊，請參閱 PostgreSQL CREATE CAST 文件。

從 Aurora PostgreSQL 13.23、14.20、15.15、16.11、17.7 和 18.1開始，您可以使用 rds_casts 擴充功能為內建類型安裝額外的轉換，同時仍然能夠為自訂類型建立自己的轉換。

主題

- 安裝和使用 rds_casts 延伸模組
- 支援的轉換
- 建立或捨棄轉換
- 使用適當的內容策略建立自訂轉換

安裝和使用 rds_casts 延伸模組

若要建立`rds_casts`擴充功能，請將 RDS for PostgreSQL 資料庫執行個體連接至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，`rds_superuser`並執行下列命令：

```
CREATE EXTENSION IF NOT EXISTS rds_casts;
```

支援的轉換

在您要使用自訂轉換的每個資料庫中建立延伸。建立擴充功能後，請使用下列命令來檢視所有可用的轉換：

```
SELECT * FROM rds_casts.list_supported_casts();
```

此函數會列出可用的轉換組合（來源類型、目標類型、強制內容和轉換函數）。例如，如果您想要text將 建立numeric為 implicit 轉換。您可以使用下列查詢來尋找轉換是否可供建立：

```
SELECT * FROM rds_casts.list_supported_casts()
WHERE source_type = 'text' AND target_type = 'numeric';
 id | source_type | target_type |          qualified_function          |
 coercion_context
----+-------------+-------------+--------------------------------------
+------------------
 10 | text        | numeric     | rds_casts.rds_text_to_numeric_custom | implicit
 11 | text        | numeric     | rds_casts.rds_text_to_numeric_custom | assignment
 13 | text        | numeric     | rds_casts.rds_text_to_numeric_custom | explicit
 20 | text        | numeric     | rds_casts.rds_text_to_numeric_inout  | implicit
 21 | text        | numeric     | rds_casts.rds_text_to_numeric_inout  | assignment
 23 | text        | numeric     | rds_casts.rds_text_to_numeric_inout  | explicit
```

rds_casts 擴充功能為每個轉換提供兩種類型的轉換函數：

- _inout 函數 - 使用 PostgreSQL 的標準 I/O 轉換機制，其行為與使用 INOUT 方法建立的轉換完全相同
- _custom 函數 - 提供處理邊緣案例的增強型轉換邏輯，例如將空字串轉換為 NULL 值以避免轉換錯誤

這些inout函數會複寫 PostgreSQL 的原生轉換行為，而custom函數會透過處理標準 INOUT 轉換無法容納的案例來擴展此功能，例如將空字串轉換為整數。

建立或捨棄轉換

您可以使用兩種方法來建立和捨棄支援的轉換：

投射建立

方法 1：使用原生 CREATE CAST 命令

```
CREATE CAST (text AS numeric)
WITH FUNCTION rds_casts.rds_text_to_numeric_custom
AS IMPLICIT;
```

方法 2：使用 rds_casts.create_cast 函數

```
SELECT rds_casts.create_cast(10);
```

create_cast 函數會從list_supported_casts()輸出取得 ID。此方法更簡單，並確保您使用正確的函數和內容組合。此 ID 保證在不同 Postgres 版本中保持不變。

若要確認已成功建立轉換，請查詢 pg_cast 系統目錄：

```
SELECT oid, castsource::regtype, casttarget::regtype, castfunc::regproc, castcontext,
 castmethod
FROM pg_cast
WHERE castsource = 'text'::regtype AND casttarget = 'numeric'::regtype;
  oid   | castsource | casttarget |               castfunc               | castcontext
 | castmethod
--------+------------+------------+--------------------------------------+-------------
+------------
 356372 | text       | numeric    | rds_casts.rds_text_to_numeric_custom | i
 | f
```

castcontext 資料欄顯示：e適用於 EXPLICIT、a適用於 ASSIGNMENT 或i適用於 IMPLICIT。

捨棄轉換

方法 1：使用 DROP CAST 命令

```
DROP CAST IF EXISTS (text AS numeric);
```

方法 2：使用 rds_casts.drop_cast 函數

```
SELECT rds_casts.drop_cast(10);
```

drop_cast 函數會採用與建立投射時相同的 ID。此方法可確保您捨棄使用對應 ID 建立的確切轉換。

使用適當的內容策略建立自訂轉換

為整數類型建立多個轉換時，如果所有轉換都建立為 IMPLICIT，則可能會發生運算子模棱兩可錯誤。下列範例透過建立兩個隱含從文字轉換為不同整數寬度來示範此問題：

```
-- Creating multiple IMPLICIT casts causes ambiguity
postgres=> CREATE CAST (text AS int4) WITH FUNCTION
 rds_casts.rds_text_to_int4_custom(text) AS IMPLICIT;
CREATE CAST
postgres=> CREATE CAST (text AS int8) WITH FUNCTION
 rds_casts.rds_text_to_int8_custom(text) AS IMPLICIT;
```

```
CREATE CAST

postgres=> CREATE TABLE test_cast(col int);
CREATE TABLE
postgres=> INSERT INTO test_cast VALUES ('123'::text);
INSERT 0 1
postgres=> SELECT * FROM test_cast WHERE col='123'::text;
ERROR:  operator is not unique: integer = text
LINE 1: SELECT * FROM test_cast WHERE col='123'::text;
                                         ^
HINT:  Could not choose a best candidate operator. You might need to add explicit type
 casts.
```

發生錯誤是因為 PostgreSQL 無法判斷在比較整數資料欄與文字值時要使用的隱含轉換。int4 和 int8 隱含轉換都是有效的候選項目，可建立模棱兩可。

若要避免此運算子模棱兩可的情況，請針對較小的整數寬度使用 ASSIGNMENT 內容，針對較大的整數寬度使用 IMPLICIT 內容：

```
-- Use ASSIGNMENT for smaller integer widths
CREATE CAST (text AS int2)
WITH FUNCTION rds_casts.rds_text_to_int2_custom(text)
AS ASSIGNMENT;

CREATE CAST (text AS int4)
WITH FUNCTION rds_casts.rds_text_to_int4_custom(text)
AS ASSIGNMENT;

-- Use IMPLICIT for larger integer widths
CREATE CAST (text AS int8)
WITH FUNCTION rds_casts.rds_text_to_int8_custom(text)
AS IMPLICIT;

postgres=> INSERT INTO test_cast VALUES ('123'::text);
INSERT 0 1
postgres=> SELECT * FROM test_cast WHERE col='123'::text;
 col
-----
 123
(1 row)
```

在此策略中，只有 int8 轉換是隱含的，因此 PostgreSQL 可以明確判斷要使用的轉換。

Aurora PostgreSQL 中平行查詢的最佳實務

平行查詢執行是 PostgreSQL 的一項功能，可讓單一 SQL 查詢分成較小的任務，由多個背景工作者程序同時處理。PostgreSQL 可以在多個 CPU 核心之間分配部分查詢，例如掃描、聯結、彙總或排序，而不是完全在單一後端程序中執行查詢。領導者程序會協調此執行，並從平行工作者收集結果。

不過，對於大多數生產工作負載，特別是高並行 OLTP 系統，我們建議停用自動平行查詢執行。雖然平行處理可以加速分析或報告工作負載中大型資料集的查詢，但它帶來的重大風險通常超過忙碌生產環境中的優勢。

平行執行也會帶來大量的額外負荷。每個平行工作者都是完整的 PostgreSQL 後端程序，這需要程序偽造（複製記憶體結構和初始化程序狀態）和身分驗證（從您的`max_connections`限制使用連線插槽）。每個工作者也會使用自己的記憶體，包括`work_mem`用於排序和雜湊操作，每個查詢有多個工作者，記憶體用量會快速增加（例如，4 個工作者 × 64MB `work_mem` = 每個查詢 256MB)。因此，平行查詢可能會耗用比單一程序查詢更多的系統資源。如果未正確調校， 它們可能會導致 CPU 飽和（多個工作者壓倒可用的處理容量）， 增加內容切換（作業系統經常在許多工作者程序之間切換， 新增額外負荷並降低輸送量）、 或 連線耗盡（因為每個平行工作者都會使用連線插槽， 具有 4 個工作者的單一查詢總共使用 5 個連線， 1 名領導者 + 4 名工作者， 可在高並行下快速耗盡連線集區， 防止新的用戶端連線並導致應用程式失敗）。在高並行工作負載中，這些問題特別嚴重，其中多個查詢可能會同時嘗試平行執行。

PostgreSQL 會根據成本估算決定是否使用平行處理。在某些情況下，規劃器可能會自動切換到平行計劃，即使實際上並不理想，它看起來更便宜。如果索引統計資料已過期，或者如果膨脹使循序掃描看起來比索引查詢更具吸引力，則可能會發生這種情況。由於這種行為，自動平行計畫有時可能會在查詢效能或系統穩定性中引入迴歸。

若要從 Aurora PostgreSQL 的平行查詢中獲得最大效益，請務必根據您的工作負載進行測試和調校、監控系統影響，以及停用自動平行計劃選擇，以有利於查詢層級控制。

組態參數

PostgreSQL 使用數個參數來控制平行查詢的行為和可用性。了解和調校這些對於實現可預測效能至關重要：

參數	描述	預設
`max_parallel_workers`	可以執行的背景工作者程序總數上限	GREATEST($DBInstanceVCPU/2,8)

參數	描述	預設
max_parallel_workers_per_gather	每個查詢計劃節點的工作者數量上限（例如，每個 Gather)	2
parallel_setup_cost	新增啟動平行查詢基礎設施的規劃器成本	1000
parallel_tuple_cost	在平行模式下處理的每元組成本（影響規劃器決策）	0.1
force_parallel_mode	強制規劃器測試平行計劃 (off、on、regress)	off

重要考量事項

- max_parallel_workers 控制平行工作者的總集區。如果設定過低，某些查詢可能會回復到序列執行。
- max_parallel_workers_per_gather 會影響單一查詢可以使用的工作者數量。較高的值會增加並行，但也會增加資源用量。
- parallel_setup_cost 和 parallel_tuple_cost會影響規劃器的成本模型。降低這些值可能會讓平行計劃更容易被選擇。
- force_parallel_mode 適用於測試，但除非必要，否則不應用於生產環境。

> **Note**
>
> 參數的預設值max_parallel_workers會根據執行個體大小使用公式 動態計算GREATEST($DBInstanceVCPU/2, 8)。這表示當您將 Aurora instance擴展到具有更多 vCPUs較大運算大小時，可用的平行工作者數量上限會自動增加。因此，先前連續執行或具有有限平行處理的查詢可能會在擴展操作後突然利用更多平行工作者，這可能會導致連線用量、CPU 使用率和記憶體耗用量意外增加。請務必監控任何運算擴展事件之後的平行查詢行為max_parallel_workers_per_gather，並視需要調整以維持可預測的資源用量。

識別平行查詢用量

查詢可能會根據資料分佈或統計資料轉換為平行計畫。例如：

```
SELECT count(*) FROM customers WHERE last_login < now() - interval '6 months';
```

此查詢可能會針對最近資料使用 索引，但切換到針對歷史資料的平行循序掃描。

您可以透過載入 auto_explain模組來記錄查詢執行計畫。若要進一步了解，請參閱 AWS 知識中心的記錄查詢的執行計畫。

您可以在 Aurora PostgreSQL 資料庫執行個體中監控查詢執行計畫，以偵測造成目前資料庫負載的執行計畫，並使用 aurora_compute_plan_id 參數追蹤執行計畫隨時間的效能統計資料。若要進一步了解，請參閱監控 Aurora PostgreSQL 的查詢執行計畫和尖峰記憶體

您可以監控 CloudWatch Database Insights 的平行查詢相關等待事件。若要進一步了解平行查詢相關的等待事件，請執行 IPC：平行等待事件

從 PostgreSQL 第 18 版，您可以使用 pg_stat_database和 中的新資料欄來監控平行工作者活動pg_stat_statements：

- parallel_workers_to_launch：計劃啟動的平行工作者數量
- parallel_workers_launched：實際啟動的平行工作者數量

這些指標有助於識別計劃和實際平行處理之間的差異，這可能表示資源限制或組態問題。使用下列查詢來監控平行執行：

對於資料庫層級平行工作者指標：

```
SELECT datname, parallel_workers_to_launch, parallel_workers_launched
FROM pg_stat_database
WHERE datname = current_database();
```

對於查詢層級平行工作者指標

```
SELECT query, parallel_workers_to_launch, parallel_workers_launched
FROM pg_stat_statements
ORDER BY parallel_workers_launched;
```

如何控制平行處理

控制查詢平行處理有數種方式，每種方式都專為不同的案例和需求而設計。

若要全域停用自動平行處理，請修改參數群組以設定：

```
max_parallel_workers_per_gather = 0;
```

對於持久性、使用者特定的設定，ALTER ROLE 命令提供設定參數的方法，這些參數將套用至特定使用者未來的所有工作階段。

例如：

`ALTER ROLE username SET max_parallel_workers_per_gather = 4;` 確保每次此使用者連線到資料庫時，其工作階段都會在需要時使用此平行工作者設定。

您可以使用 SET 命令來達成工作階段層級控制，該命令會修改目前資料庫工作階段期間的參數。當您需要暫時調整設定，而不會影響其他使用者或未來的工作階段時，這特別有用。一旦設定，這些參數會保持有效，直到明確重設或工作階段結束為止。命令非常簡單：

```
SET max_parallel_workers_per_gather = 4;
-- Run your queries
RESET max_parallel_workers_per_gather;
```

對於更精細的控制，SET LOCAL 可讓您修改單一交易的參數。當您需要調整交易中特定查詢集的設定時，這很理想，之後設定會自動回復到其先前的值。此方法有助於防止對相同工作階段中的其他操作產生意外影響。

利用查詢計劃管理 (QPM)

在 Aurora PostgreSQL 中，查詢計劃管理 (QPM) 旨在確保計劃適應性和穩定性，無論資料庫環境的變更是否可能導致查詢計劃迴歸。如需詳細資訊，請參閱 Aurora PostgreSQL 查詢計畫管理概觀。QPM 提供了對優化工具的一些控制。在 QPM 中檢閱核准的計劃，以確保它們符合目前的平行處理設定。更新或移除可能強制執行次佳平行執行的過時計劃。

您也可以使用 pg_hint_plan 來修正計劃。如需詳細資訊，請參閱使用 pg_hint_plan 修正計劃。您可以使用名為 `Parallel` 的提示來強制執行平行執行。如需詳細資訊，請參閱平行計劃的提示。

診斷平行查詢行為

使用 `EXPLAIN (ANALYZE, VERBOSE)`確認查詢是否使用平行執行：

- 尋找節點，例如 Gather、 Gather Merge或 Parallel Seq Scan。
- 比較具有和不具有平行處理的計劃。

若要暫時停用平行處理以進行比較：

```
SET max_parallel_workers_per_gather = 0;
EXPLAIN ANALYZE <your_query>;
RESET max_parallel_workers_per_gather;
```

針對 Aurora PostgreSQL 中的儲存問題進行故障診斷

如果排序或索引建立操作所需的工作記憶體數量超出 work_mem 參數所配置的數量，Aurora PostgreSQL 會將多餘的資料寫入至暫存磁碟檔案。當它寫入資料時，Aurora PostgreSQL 會使用其用於存放錯誤和訊息日誌的相同儲存體，亦即，本機儲存體。Aurora PostgreSQL 資料庫叢集中的每個執行個體都有可用的本機儲存量。儲存量是以其資料庫執行個體類別為基礎。若要增加本機儲存量，您需要修改執行個體以使用較大的資料庫執行個體類別。資料庫執行個體類別規格，請參閱 Aurora 資料庫執行個體類別的硬體規格。

您可以透過監看 FreeLocalStorage 的 Amazon CloudWatch 指標，來監控 Aurora PostgreSQL 資料庫叢集的本機儲存空間。此指標回報可供 Aurora 資料庫叢集中每個資料庫執行個體用於暫時資料表與記錄的儲存量。如需詳細資訊，請參閱使用 Amazon CloudWatch 監控 Amazon Aurora 指標。

排序、檢索和分組操作在工作記憶體中開始，但通常必須卸載至本機儲存體。如果您的 Aurora PostgreSQL 資料庫叢集由於這些類型的操作而耗盡本機儲存體，則您可以採取下列其中一個動作來解決此問題。

- 增加工作記憶體數量。這會減少使用本機儲存體的需求。根據預設，PostgreSQL 會針對每個排序、群組和索引操作配置 4 MB。若要檢查 Aurora PostgreSQL 資料庫叢集寫入器執行個體目前的工作記憶體數值，請使用 psql 連線至執行個體，並執行下列命令。

  ```
  postgres=> SHOW work_mem;
  work_mem
  ----------
   4MB
  (1 row)
  ```

  您可以在排序、群組和其他操作之前增加工作階段層級的工作記憶體，如下所示。

```
SET work_mem TO '1 GB';
```

如需為工作記憶體的詳細資訊，請參閱 PostgreSQL 文件中的資源耗用。

- 變更日誌保留期間，以便存放日誌的時間範圍縮短。若要了解作法，請參閱Aurora PostgreSQL 資料庫日誌檔案。

針對大於 40 TB 的 Aurora PostgreSQL 資料庫執行個體，請不要使用 db.t2、db.t3 或 db.t4g 執行個體類別。建議您在開發、測試伺服器或其他非生產伺服器時，僅使用 T 資料庫執行個體類別。如需詳細資訊，請參閱資料庫執行個體類別的類型。

以 Amazon Aurora PostgreSQL 進行複寫

在下文中，您可以找到如何使用 Amazon Aurora PostgreSQL 進行複寫的相關資訊，其中包括如何監控和使用邏輯複寫。

主題

- 使用 Aurora 複本
- 改善 Aurora 複本的讀取可用性
- 監控 Aurora PostgreSQL 複寫
- 以 Aurora 進行 PostgreSQL 邏輯複寫的概觀
- 針對 Aurora PostgreSQL 資料庫叢集設定邏輯複寫
- 關閉邏輯複寫
- 監控 Aurora PostgreSQL 邏輯複寫的全部寫入快取和邏輯槽
- 範例：搭配 Aurora PostgreSQL 資料庫叢集使用邏輯複寫
- 範例：使用 Aurora PostgreSQL 和 進行邏輯複寫 AWS Database Migration Service
- 設定邏輯複寫連線的 IAM 身分驗證

使用 Aurora 複本

Aurora 複本是 Aurora 資料庫叢集中的獨立端點，最適合用於擴展讀取操作和提高可用性。Aurora 資料庫叢集最多可包含 15 個 Aurora 複本，位於 Aurora 資料庫叢集 AWS 區域的可用區域。

資料庫叢集磁碟區由資料庫叢集的多個資料副本組成。然而，叢集磁碟區中的資料是以單一邏輯磁碟區的形式呈現給主要執行個體，以及資料庫叢集中的 Aurora 複本。如需 Aurora 複本的詳細資訊，請參閱 Aurora 複本。

Aurora 複本很適合讀取擴展，因為完全專用於叢集磁碟區上的讀取操作。寫入器資料庫執行個體會管理寫入操作。Aurora PostgreSQL 資料庫叢集中的所有執行個體將共用叢集磁碟區。因此，不需要額外的工作來複寫每個 Aurora 複本的資料複本。

有了 Aurora PostgreSQL，在刪除 Aurora 複本時，隨即會移除其執行個體端點，並且會從讀取器端點移除 Aurora 複本。如果在要刪除的 Aurora 複本上有陳述式正在執行，則會有三分鐘的寬限期。現有陳述式在寬限期期間可以從容地完成。在寬限期結束時，Aurora 複本會關閉並刪除。

Aurora PostgreSQL 資料庫叢集使用 Aurora 全域資料庫，支援不同 AWS 區域中的 Aurora 複本。如需詳細資訊，請參閱使用 Amazon Aurora 全球資料庫。

> **Note**
>
> 透過讀取可用性功能，您必須手動執行才可重新啟動資料庫叢集中的 Aurora 複本。對於在此功能之前建立的資料庫叢集，重新啟動寫入器資料庫執行個體會自動重新啟動 Aurora 複本。自動重新啟動會重新建立進入點，保證資料庫叢集之間讀取/寫入的一致性。

改善 Aurora 複本的讀取可用性

Aurora PostgreSQL 透過在寫入器資料庫執行個體重新啟動時，或在 Aurora 複本無法跟上寫入流量時持續提供讀取請求，從而改善資料庫叢集中的讀取可用性。

根據預設，可在下列版本的 Aurora PostgreSQL 上使用讀取可用性功能：

- 16.1 版和所有更新版本
- 15.2 版和更新的 15 版本
- 14.7 版和更新的 14 版本
- 13.10 版和更新的 13 版本
- 12.14 版和更新的 12 版本

Aurora 全球資料庫支援下列版本中的讀取可用性功能：

- 16.1 版和所有更新版本

- 15.4 版和更新的 15 版本
- 14.9 版和更新的 14 版本
- 13.12 版和更新的 13 版本
- 12.16 版和更新的 12 版本

若要在此啟動之前針對在其中一個版本上建立的資料庫叢集使用讀取可用性功能，請重新啟動資料庫叢集的寫入器執行個體。

修改 Aurora PostgreSQL 資料庫叢集的靜態參數時，您必須重新啟動寫入器執行個體，以便參數變更生效。例如，設定 shared_buffers 的值時，您必須重新啟動寫入器執行個體。使用 Aurora 複本讀取可用性功能時，資料庫叢集能維持更佳的可用性，在寫入器執行個體重新啟動時降低對叢集的影響。讀取器執行個體不會重新啟動，並繼續回應讀取請求。若要套用靜態參數變更，請重新啟動每個個別的讀取器執行個體。

Aurora PostgreSQL 資料庫叢集的 Aurora 複本可從複寫錯誤 (例如寫入器重新啟動、容錯移轉、緩慢複寫和網路問題) 中復原，方法為在與寫入器重新連線之後快速復原至記憶體資料庫狀態。此方法可讓 Aurora 複本執行個體在用戶端資料庫仍然可用時，與最新的儲存體更新達成一致性。

與複寫復原發生衝突的進行中交易可能會收到錯誤，但是在讀取器趕上寫入器之後，用戶端可以重試這些交易。

監控 Aurora 複本

從寫入器中斷連線復原時，您可以監控 Aurora 複本。使用以下指標檢查讀取器執行個體是否有最新的相關資訊，以及追蹤進行中的唯讀交易。

- aurora_replica_status 函數會更新，以在讀取器執行個體仍然連線時，傳回該執行個體的最新資訊。資料列若對應至查詢執行所在的資料庫執行個體，aurora_replica_status 中的最後一個更新時間戳記一律為空白。這表示讀取器執行個體具有最新的資料。
- 當 Aurora 複本中斷與寫入器執行個體的連線並重新連線時，系統會發送下列資料庫事件：

  ```
  Read replica has been disconnected from the writer instance and
  reconnected.
  ```

- 當唯讀查詢由於復原衝突而遭到取消時，您可能會在資料庫錯誤日誌中看到一或多個下列錯誤訊息：

  ```
  Canceling statement due to conflict with recovery.
  ```

  ```
  User query may not have access to page data to replica disconnect.
  ```

```
User query might have tried to access a file that no longer exists.

When the replica reconnects, you will be able to repeat your command.
```

限制

下列限制適用於具有讀取可用性功能的 Aurora 複本：

- 如果在複寫復原期間無法從寫入器執行個體串流資料，則次要資料庫叢集的 Aurora 複本就可能會重新啟動。
- 如果 Aurora 複本已在進行中，則它不支援線上複寫復原，並將重新啟動。
- 當您的資料庫執行個體接近交易 ID 包圍時，Aurora 複本將重新啟動。如需交易 ID 包圍的詳細資訊，請參閱避免交易 ID 包圍失敗。
- 在特定情況下，當複寫程序遭到封鎖時，Aurora 複本可以重新啟動。

監控 Aurora PostgreSQL 複寫

讀取擴展和高可用性取決於最短延遲時間。您可以監控 Amazon CloudWatch `ReplicaLag` 指標，以監控 Aurora 複本落後 Aurora PostgreSQL 資料庫叢集之寫入器資料庫執行個體的程度。因為 Aurora 複本會從與寫入器資料庫執行個體所在的相同叢集磁碟區讀取，`ReplicaLag` 指標對 Aurora PostgreSQL 資料庫叢集有不同的意義。Aurora 複本的 `ReplicaLag` 指標指出 Aurora 複本的頁面快取落後於寫入器資料庫執行個體的頁面快取的程度。

如需監控 RDS 執行個體和 CloudWatch 指標的詳細資訊，請參閱 在 Amazon Aurora 叢集中監控指標。

以 Aurora 進行 PostgreSQL 邏輯複寫的概觀

透過將 PostgreSQL 的邏輯複寫功能與 Aurora PostgreSQL 資料庫叢集搭配使用，您可以複寫和同步個別資料表，而不是整個資料庫執行個體。邏輯複寫會使用發佈與訂閱模型，將來源中的變更複寫到一或多個收件人。其運作方式為使用 PostgreSQL 預寫日誌 (WAL) 中的變更記錄。來源 (或發佈者) 會將所指定資料表的 WAL 資料傳送給一或多個收件人 (訂閱者)，進而複寫變更，並讓訂閱者的資料表與發行者的表格保持同步。來自發佈者的變更集是使用發佈來識別的。訂閱者取得變更的方式為建立訂閱，定義與發佈者資料庫及其發佈的連線。複寫插槽是此結構描述中用來追蹤訂閱進度的機制。

對於 Aurora PostgreSQL 資料庫叢集，WAL 記錄會儲存在 Aurora 儲存體上。在邏輯複寫案例中做為發佈者的 Aurora PostgreSQL 資料庫叢集會從 Aurora 儲存體讀取 WAL 資料、將其解碼，然後將

其傳送給訂閱者，以便可將變更套用至該執行個體上的資料表。發佈者會使用邏輯解碼器來解碼資料以供訂閱者使用。根據預設，Aurora PostgreSQL 資料庫叢集會在傳送資料時使用原生 PostgreSQL `pgoutput` 外掛程式。還有其他邏輯解碼器可供使用。例如，Aurora PostgreSQL 也支援將 WAL 資料轉換為 JSON 的 `wal2json` 外掛程式。

從 Aurora PostgreSQL 14.5、13.8、12.12 和 11.17 版起，Aurora 使用直接寫入式快取來擴增 PostgreSQL 邏輯複寫程序，以提升效能。WAL 交易日誌會在緩衝區中進行本機快取，以減少磁碟 I/O 的數量，亦即在邏輯解碼期間從 Aurora 儲存體進行讀取。每當您針對 Aurora PostgreSQL 資料庫叢集使用邏輯複寫時，預設會使用直接寫入式快取。Aurora 提供數個您可以用來管理快取的功能。如需詳細資訊，請參閱監控 Aurora PostgreSQL 邏輯複寫全部寫入快取。

所有目前可用的 Aurora PostgreSQL 版本都支援邏輯複寫。如需詳細資訊，請參閱《Aurora PostgreSQL 版本資訊》中的 Amazon Aurora PostgreSQL 更新。

從下列版本 Babelfish for Aurora PostgreSQL 支援邏輯複寫：

- 15.7 及更新版本
- 16.3 及更新版本

> **Note**
>
> 除了在 PostgreSQL 10 中引進的原生 PostgreSQL 邏輯複寫功能之外，Aurora PostgreSQL 也支援 `pglogical` 延伸模組。如需詳細資訊，請參閱使用 pglogical 跨執行個體同步資料。

如需 PostgreSQL 邏輯複寫的詳細資訊，請參閱 PostgreSQL 文件中的邏輯複寫和邏輯解碼概念。

> **Note**
>
> PostgreSQL 16 新增支援從僅供讀取複本進行邏輯解碼。Aurora PostgreSQL 不支援此功能。

針對 Aurora PostgreSQL 資料庫叢集設定邏輯複寫

設定邏輯複製需要 `rds_superuser` 權限。您的 Aurora PostgreSQL 資料庫叢集必須設定為使用自訂資料庫叢集參數群組，以便您可以依下列程序所述設定必要的參數。如需詳細資訊，請參閱Amazon Aurora 資料庫叢集的資料庫叢集參數群組。

針對 Aurora PostgreSQL 資料庫叢集設定 PostgreSQL 邏輯複寫

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Aurora PostgreSQL 資料庫叢集。
3. 開啟 Configuration (組態) 索引標籤。在執行個體詳細資訊中找出參數群組連結，並於類型清單中選擇資料庫叢集參數群組。
4. 選擇連結以開啟與 Aurora PostgreSQL 資料庫叢集相關聯的連結。
5. 在 Parameters (參數) 搜尋欄位中，輸入 `rds` 以尋找 `rds.logical_replication` 參數。此參數的預設值為 `0`，表示預設為關閉。
6. 選擇 Edit parameters (編輯參數) 以存取屬性值，然後從選取器中選擇 1 以開啟功能。根據您預期的使用情況，您可能也需要變更下列參數的設定。不過，在許多情況下，預設值就足夠了。

   - `max_replication_slots` – 將此參數設定為至少等於您規劃之邏輯複寫發佈與訂閱總數的值。如果您使用的是 AWS DMS，此參數應至少等於您從叢集規劃的變更資料擷取任務，以及邏輯複寫出版物和訂閱。
   - `max_wal_senders` 和 `max_logical_replication_workers` – 將這些參數設定為至少等於您要作用中的邏輯複寫槽數目，或變更資料擷取的作用中 AWS DMS 任務數目的值。讓邏輯複寫槽處於非作用中會導致清理無法從資料表移除已淘汰的元組，因此建議您監控複寫槽並視需要移除非作用中的複寫槽。
   - `max_worker_processes` – 將此參數設定為至少等於 `max_logical_replication_workers`、`autovacuum_max_workers`、和 `max_parallel_workers` 值總計的值。在小型資料庫執行個體類別上，背景工作者程序可能影響應用程式工作負載，因此，如果您將 `max_worker_processes` 設為高於預設值，請監控資料庫的效能。(預設值是 `GREATEST(${DBInstanceVCPU*2},8}` 的結果，表示依預設，這是 8 或是資料庫執行個體類別對等 CPU 的兩倍，以較大者為準)。

   > **Note**
   >
   > 您可修改客戶建立的資料庫參數群組中的參數值；但無法變更預設資料庫參數群組中的參數值。

7. 選擇儲存變更。

8. 重新啟動 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，以便您的變更生效。在 Amazon RDS 主控台中，選擇叢集的主要資料庫執行個體，然後從 Actions (動作) 中選擇 Reboot (重新啟動)。
9. 當執行個體可用時，您可以驗證邏輯複寫是否已開啟，如下所示。

   a. 使用 psql 連線到 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。

   ```
   psql --host=your-db-cluster-instance-1.aws-region.rds.amazonaws.com --port=5432
    --username=postgres --password --dbname=labdb
   ```

   b. 使用下列命令驗證是否已啟用邏輯複寫。

   ```
   labdb=> SHOW rds.logical_replication;
    rds.logical_replication
   -------------------------
    on
   (1 row)
   ```

   c. 驗證 wal_level 是否已設為 logical。

   ```
   labdb=> SHOW wal_level;
     wal_level
   -----------
    logical
   (1 row)
   ```

如需使用邏輯複寫，讓資料庫表格與來源 Aurora PostgreSQL 資料庫叢集中的變更保持同步的範例，請參閱 範例：搭配 Aurora PostgreSQL 資料庫叢集使用邏輯複寫。

關閉邏輯複寫

完成複寫任務之後，您應該停止複寫程序、捨棄複寫槽，並關閉邏輯複寫。在捨棄複寫槽之前，確定不再需要它們。無法捨棄作用中複寫槽。

關閉邏輯複寫

1. 捨棄所有複寫槽。

   若要捨棄所有複寫槽，請連線到發佈者，然後執行下列 SQL 命令。

```
SELECT pg_drop_replication_slot(slot_name)
  FROM pg_replication_slots
 WHERE slot_name IN (SELECT slot_name FROM pg_replication_slots);
```

執行此命令時，複寫槽不能處於作用中。

2. 如 針對 Aurora PostgreSQL 資料庫叢集設定邏輯複寫 所述，修改與發佈者相關聯的自訂資料庫叢集參數群組，但將 `rds.logical_replication` 參數設為 0。

   如需自訂參數群組的詳細資訊，請參閱 在 Amazon Aurora 中修改資料庫叢集參數群組中的參數。

3. 重新啟動發佈者 Aurora PostgreSQL 資料庫叢集，使 `rds.logical_replication` 參數的變更生效。

監控 Aurora PostgreSQL 邏輯複寫的全部寫入快取和邏輯槽

監控邏輯複寫全部寫入快取並管理邏輯槽，以改善 Aurora PostgreSQL 資料庫叢集的效能。以下請見全部寫入快取和邏輯槽的詳細資訊。

主題

- 監控 Aurora PostgreSQL 邏輯複寫全部寫入快取
- 管理 Aurora PostgreSQL 的邏輯槽

監控 Aurora PostgreSQL 邏輯複寫全部寫入快取

根據預設，Aurora PostgreSQL 14.5、13.8、12.12 和 11.17 版，以及更高版本會使用直接寫入式快取來改善邏輯複寫的效能。若沒有直接寫入式快取，Aurora PostgreSQL 會在實作原生 PostgreSQL 邏輯複寫程序時使用 Aurora 儲存層。作法是將 WAL 資料寫入儲存體，然後從儲存體中讀回資料，以對其進行解碼並傳送 (複寫)到其目標 (訂閱者)。這可能會導致 Aurora PostgreSQL 資料庫叢集進行邏輯複寫期間產生瓶頸。

全部寫入快取可將對 Aurora 儲存層的依賴降至最低。Aurora PostgreSQL 不會持續寫入此層和從中讀取，而是會使用緩衝區來快取邏輯 WAL 串流，以便在複寫過程中使用，進而減少對存取磁碟的需求。此緩衝區是邏輯複寫所使用的原生 PostgreSQL 快取，其會在 Aurora PostgreSQL 資料庫叢集參數中識別為 `rds.logical_wal_cache`。

當使用邏輯複寫搭配 Aurora PostgreSQL 資料庫叢集 (適用於支援直接寫入式快取的版本)時，您可以監控快取命中率，以查看其在您使用案例中的運作有多好。若要這麼做，請使

用 psql 連線至 Aurora PostgreSQL 資料庫叢集的寫入執行個體，然後使用 Aurora 函數 aurora_stat_logical_wal_cache，如下列範例所示。

```
SELECT * FROM aurora_stat_logical_wal_cache();
```

此函數會傳回如下的輸出。

```
name       | active_pid | cache_hit | cache_miss | blks_read | hit_rate |
 last_reset_timestamp
-----------+------------+-----------+------------+-----------+----------+--------------
test_slot1 | 79183      | 24        | 0          | 24        | 100.00%  | 2022-08-05
 17:39...
test_slot2 |            | 1         | 0          |  1        | 100.00%  | 2022-08-05
 17:34...
(2 rows)
```

為了可讀性，last_reset_timestamp 值已縮短。如需此函數狀態的詳細資訊，請參閱 aurora_stat_logical_wal_cache。

Aurora PostgreSQL 提供下列兩個函數，用於監控直接寫入式快取。

- aurora_stat_logical_wal_cache 函數 – 如需參考文件，請參閱 aurora_stat_logical_wal_cache。
- aurora_stat_reset_wal_cache 函數 – 如需參考文件，請參閱 aurora_stat_reset_wal_cache。

如果發現自動調整的 WAL 快取大小無法滿足工作負載，則可以手動變更 rds.logical_wal_cache 的值。考慮下列各項：

- 停用 rds.logical_replication 參數時，rds.logical_wal_cache 會設定為零 (0)。
- 啟用 rds.logical_replication 參數時，rds.logical_wal_cache 的預設值為 16 MB。
- rds.logical_wal_cache 為靜態參數，且需要重新啟動資料庫執行個體，變更才能產生作用。系統會以 8 Kb 區塊為單位定義此參數。請注意，任何小於 32 Kb 的正值都會視為 32 Kb。如需 wal_buffers 的詳細資訊，請參閱 PostgreSQL 文件中的預寫日誌。

管理 Aurora PostgreSQL 的邏輯槽

在 pg_replication_origin_status 檢視中擷取串流活動。若要查看此檢視的內容，您可以使用 pg_show_replication_origin_status() 函數，如下所示：

```
SELECT * FROM pg_show_replication_origin_status();
```

您可以使用下面的 SQL 查詢，取得您的邏輯槽清單。

```
SELECT * FROM pg_replication_slots;
```

若要捨棄邏輯槽，請使用 pg_drop_replication_slot 搭配該槽的名稱，如下列命令中所示。

```
SELECT pg_drop_replication_slot('test_slot');
```

範例：搭配 Aurora PostgreSQL 資料庫叢集使用邏輯複寫

下列程序說明如何在兩個 Aurora PostgreSQL 資料庫叢集之間啟動邏輯複寫。發佈者和訂閱者都必須針對邏輯複寫進行設定，如 針對 Aurora PostgreSQL 資料庫叢集設定邏輯複寫 中所述。

作為指定發佈者的 Aurora PostgreSQL 資料庫叢集也必須允許存取複寫槽。若要這麼做，請根據 Amazon VPC 服務，修改與 Aurora PostgreSQL 資料庫叢集之虛擬公有雲端 (VPC) 相關聯的安全群組。將與訂閱者 VPC 相關聯的安全群組新增至發佈者的安全群組，以允許傳入存取。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的使用安全群組控制到資源的流量。

完成這些初步步驟後，您可以在發佈者上使用 PostgreSQL 命令 CREATE PUBLICATION，以及在訂閱者上使用 CREATE SUBSCRIPTION，如下列程序所述。

在兩個 Aurora PostgreSQL 資料庫叢集之間啟動邏輯複寫程序

這些步驟假設您的 Aurora PostgreSQL 資料庫叢集具有一個寫入器執行個體，其中包含可在其中建立範例資料表的資料庫。

1. 在發佈者 Aurora PostgreSQL 資料庫叢集上

   a. 使用下列 SQL 陳述式建立資料表。

   ```
   CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
   ```

   b. 使用下列 SQL 陳述式，將資料插入發佈者資料庫。

   ```
   INSERT INTO LogicalReplicationTest VALUES (generate_series(1,10000));
   ```

   c. 使用下列 SQL 陳述式，驗證資料是否存在於資料表中。

```
SELECT count(*) FROM LogicalReplicationTest;
```

d. 使用 CREATE PUBLICATION 陳述式建立此資料表的發佈，如下所示。

```
CREATE PUBLICATION testpub FOR TABLE LogicalReplicationTest;
```

2. 在訂閱者 Aurora PostgreSQL 資料庫叢集上

a. 在訂閱者上建立您已在發佈者上建立的相同 LogicalReplicationTest 資料表，如下所示。

```
CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
```

b. 驗證此資料表是否為空的。

```
SELECT count(*) FROM LogicalReplicationTest;
```

c. 建立訂閱以從發佈者取得變更。您必須使用有關發佈者 Aurora PostgreSQL 資料庫叢集的下列詳細資訊。

- host – 發佈者 Aurora PostgreSQL 資料庫叢集的寫入器資料庫執行個體。
- port – 寫入器資料庫執行個體正在聆聽的連接埠。PostgreSQL 的預設值為 5432。
- dbname – 資料庫的名稱。

```
CREATE SUBSCRIPTION testsub CONNECTION
   'host=publisher-cluster-writer-endpoint port=5432 dbname=db-name user=user
 password=password'
   PUBLICATION testpub;
```

> **Note**
>
> 指定此處所顯示提示以外的密碼，作為安全最佳實務。

建立訂閱後，發佈者會建立一個邏輯複寫槽。

d. 對於本範例，若要驗證初始資料已複寫在訂閱者上，請對訂閱者資料庫使用下列 SQL 陳述式。

```
SELECT count(*) FROM LogicalReplicationTest;
```

發佈者發生的任何其他變更會被複寫到訂閱者。

邏輯複寫會影響效能。建議您在複寫任務完成後關閉邏輯複寫。

範例：使用 Aurora PostgreSQL 和 進行邏輯複寫 AWS Database Migration Service

您可以使用 AWS Database Migration Service (AWS DMS) 來複寫資料庫或資料庫的一部分。使用 AWS DMS 將您的資料從 Aurora PostgreSQL 資料庫遷移至另一個開放原始碼或商業資料庫。如需 的詳細資訊 AWS DMS，請參閱AWS Database Migration Service 《使用者指南》。

下列範例示範如何從 Aurora PostgreSQL 資料庫設定邏輯複寫做為發佈者，然後使用 AWS DMS 進行遷移。本範例使用在 範例：搭配 Aurora PostgreSQL 資料庫叢集使用邏輯複寫 建立的同一個發佈者和訂閱者。

若要使用 設定邏輯複寫 AWS DMS，您需要 Amazon RDS 發佈者和訂閱者的詳細資訊。尤其是關於發佈者的寫入器資料庫執行個體，以及訂閱者的資料庫執行個體的詳細資訊。

取得發佈者的寫入器資料庫執行個體的下列資訊：

- Virtual Private Cloud (VPC) 識別符
- 子網路群組
- 可用區域 (AZ)
- VPC 安全群組
- 資料庫執行個體 ID

取得訂閱者的資料庫執行個體的下列資訊：

- 資料庫執行個體 ID
- 來源引擎

使用 與 Aurora PostgreSQL AWS DMS 進行邏輯複寫

1. 準備要使用的發佈者資料庫 AWS DMS。

若要這麼做，PostgreSQL 10.x 和以上版本的資料庫要求您將 AWS DMS 包裝函式套用於發佈者資料庫。如需此步驟和稍後步驟的詳細資訊，請參閱 AWS Database Migration Service 使用者指南中的使用 PostgreSQL 10.x 和以上版本作為 AWS DMS的來源。

2. 登入 AWS 管理主控台 並開啟位於 的 AWS DMS 主控台https://console.aws.amazon.com/dms/v2。在右上角，選擇發佈者和訂閱者所在的相同 AWS 區域。
3. 建立 AWS DMS 複寫執行個體。

   選擇與發佈者的寫入器資料庫執行個體相同的值。這些值包括下列設定：

   - 在 VPC 中，選擇與寫入器資料庫執行個體相同的 VPC。
   - 針對 Replication Subnet Group (複寫子網路群組)，選擇值同於寫入器資料庫執行個體的子網路群組。必要時，建立新的子網路群組。
   - 在 Availability zone (可用區域) 中，選擇與寫入器資料庫執行個體相同的區域。
   - 在 VPC Security Group (VPC 安全群組) 中，請選擇與寫入器資料庫執行個體相同的群組。
4. 建立來源的 AWS DMS 端點。

   使用下列設定，將發佈者指定為來源端點：

   - 在 Endpoint type (端點類型) 中，選擇 Source (來源)。
   - 選擇 Select RDS DB Instance (選取 RDS 資料庫執行個體)。
   - 在 RDS Instance (RDS 執行個體) 中，選擇發佈者的寫入器資料庫執行個體的資料庫標識符。
   - 在 Source engine (來源引擎) 中，選擇 postgres。
5. 建立目標的 AWS DMS 端點。

   使用下列設定，將發佈者指定為目標端點：

   - 在 Endpoint type (端點類型) 中，選擇 Target (目標)。
   - 選擇 Select RDS DB Instance (選取 RDS 資料庫執行個體)。
   - 在 RDS Instance (RDS 執行個體) 中，選擇發佈者資料庫執行個體的資料庫識別符。
   - 在 Source engine (來源引擎) 中選擇一個值。例如，如果訂閱者為一 RDS PostgreSQL 資料庫，則選擇 postgres。如果訂閱者是 Aurora PostgreSQL 資料庫，請選擇 aurora-postgresql。
6. 建立 AWS DMS 資料庫遷移任務。

   您使用資料庫遷移任務指定要遷移哪一個資料庫資料表，使用目標結構描述映射資料，並於目標資料庫上建立新資料表。至少將下列設定使用於 Task configuration (任務組態)：

- 在 Replication instance (複寫執行個體) 中，選擇您在先前步驟中建立的複寫執行個體。
- 在 Source database endpoint (來源資料庫端點) 中，選擇您在先前步驟中建立的發佈者來源。
- 在 Target database endpoint (目標資料庫端點) 中，選擇您在先前步驟中建立的訂閱者目標。

其餘的任務細節端視您的遷移專案而定。如需指定 DMS 任務所有詳細資訊的詳細資訊，請參閱《 使用者指南》中的使用 AWS DMS 任務。 AWS Database Migration Service

在 AWS DMS 建立任務之後，它會開始將資料從發佈者遷移至訂閱者。

設定邏輯複寫連線的 IAM 身分驗證

從 Aurora PostgreSQL 第 11 版及更新版本開始，您可以使用 AWS Identity and Access Management (IAM) 身分驗證進行複寫連線。此功能可讓您使用 IAM 角色而非密碼來管理資料庫存取，以增強安全性。它可在叢集層級運作，並遵循與標準 IAM 身分驗證相同的安全模型。

複寫連線的 IAM 身分驗證是一項選擇加入功能。若要啟用它，請在資料庫叢集`rds.iam_auth_for_replication`參數群組中將 參數設定為 `1`。由於這是動態參數，因此您的資料庫叢集不需要重新啟動，可讓您利用 IAM 身分驗證與現有的工作負載，而無需停機。啟用此功能之前，您必須符合下列先決條件各項。

先決條件

若要針對複寫連線使用 IAM 身分驗證，您需要符合下列所有要求：

- 您的 Aurora PostgreSQL 資料庫叢集必須是 11 版或更新版本。
- 在您的發佈者 Aurora PostgreSQL 資料庫叢集上：
  - 啟用 IAM 資料庫身分驗證。

    如需詳細資訊，請參閱啟用和停用 IAM 資料庫身分驗證。
  - 將 `rds.logical_replication` 參數設定為 `1` 以啟用邏輯複寫。

    如需詳細資訊，請參閱針對 Aurora PostgreSQL 資料庫叢集設定邏輯複寫。

  在邏輯複寫中，發佈者是將資料傳送至訂閱者叢集的來源 Aurora PostgreSQL 資料庫叢集。如需詳細資訊，請參閱使用 Aurora 進行 PostgreSQL 邏輯複寫的概觀。

> **Note**
>
> 您的發佈者 Aurora PostgreSQL 資料庫叢集上必須同時啟用 IAM 身分驗證和邏輯複寫。如果其中一個未啟用，則您無法將 IAM 身分驗證用於複寫連線。

啟用複寫連線的 IAM 身分驗證

完成下列步驟，以啟用複寫連線的 IAM 身分驗證。

1. 確認您的 Aurora PostgreSQL 資料庫叢集符合具有複寫連線的 IAM 身分驗證的所有先決條件。如需詳細資訊，請參閱先決條件。
2. 透過修改資料庫叢集`rds.iam_auth_for_replication`參數群組來設定 參數：
   - 將 `rds.iam_auth_for_replication` 參數設為 1。此動態參數不需要重新啟動。
3. 連線至您的資料庫，並將必要的角色授予您的複寫使用者：

   下列 SQL 命令會授予必要角色，以啟用複寫連線的 IAM 身分驗證：

   ```
   -- Grant IAM authentication role
   GRANT rds_iam TO replication_user_name;
   -- Grant replication privileges
   ALTER USER replication_user_name WITH REPLICATION;
   ```

完成這些步驟後，指定的使用者必須使用 IAM 身分驗證進行複寫連線。

> **Important**
>
> 當您啟用此功能時，同時具有 `rds_iam`和 `rds_replication`角色的使用者必須使用 IAM 身分驗證進行複寫連線。無論角色是直接指派給使用者或透過其他角色繼承，這都會套用。

停用複寫連線的 IAM 身分驗證

您可以使用下列任一方法停用複寫連線的 IAM 身分驗證：

- 將資料庫叢集`rds.iam_auth_for_replication`參數群組`0`中的 參數設定為
- 或者，您可以在 Aurora PostgreSQL 資料庫叢集上停用下列其中一項功能：

- 將 `rds.logical_replication` 參數設定為 以停用邏輯複寫 `0`
- 停用 IAM 身分驗證

當您停用此功能時，複寫連線可以使用資料庫密碼進行身分驗證，如果已設定。

> **Note**
>
> 即使啟用 功能，沒有 `rds_iam`角色的使用者複寫連線也可以使用密碼身分驗證。

限制及考量

將 IAM 身分驗證用於複寫連線時，適用下列限制和考量。

- 複寫連線的 IAM 身分驗證僅適用於 Aurora PostgreSQL 第 11 版及更新版本。
- 發佈者必須支援複寫連線的 IAM 身分驗證。
- IAM 身分驗證字符預設會在 15 分鐘後過期。您可能需要在字符過期之前重新整理長時間執行的複寫連線。

在 Aurora PostgreSQL 中使用本機寫入轉送

本機 (叢集內) 寫入轉送允許您的應用程式直接在 Aurora 複本上發出讀取/寫入交易。然後，寫入命令會轉送到要遞交的寫入器資料庫執行個體。您可以針對偶爾寫入的應用程式使用本機寫入轉送，並要求先寫後讀一致性，也就是能夠讀取交易中最新寫入的能力。

不使用寫入轉送時，您的應用程式必須完全拆分所有讀取和寫入流量，維護兩組資料庫連線，才能將流量傳送到適當的端點。僅供讀取複本會以非同步方式從寫入器執行個體接收更新。此外，由於複寫延遲在僅供讀取複本之間可能不同，因此在所有複本之間實現全域讀取一致性並不容易。您必須處理寫入器資料庫執行個體上需要先寫後讀一致性的任何讀取。或者，您需要開發複雜的自訂應用程式邏輯，才能利用多個僅供讀取複本來實現可擴展性，同時確保一致性。

透過寫入轉送，您可以避免分割這些交易或將它們單獨傳送到寫入器執行個體。您也不必開發複雜的應用程式邏輯，即可實現先寫後讀一致性。

本機寫入轉送在每個提供 Aurora PostgreSQL 的區域均可使用。其支援以下的 Aurora PostgreSQL 版本：

- 16.4 版和更新的 16 版本
- 15.8 版和更新的 15 版本
- 14.13 版和更新的 14 版本

本機寫入轉送會用於轉送區域內複本的寫入。若要轉送來自全域複本的寫入，請參閱 在 Amazon Aurora 全域資料庫中使用寫入轉送。

主題

- Aurora PostgreSQL 中本機寫入轉送的限制和考量
- 設定 Aurora PostgreSQL 以進行本機寫入轉送
- 使用 Aurora PostgreSQL 的本機寫入轉送
- 在 Aurora PostgreSQL 中監控本機寫入轉送

Aurora PostgreSQL 中本機寫入轉送的限制和考量

下列限制目前適用於 Aurora PostgreSQL 中的本機寫入轉送：

- RDS Proxy 不支援本機寫入轉送。
- 當您在具有寫入轉送的 Aurora PostgreSQL 中使用某些陳述式時，系統不允許使用這些陳述式或這些陳述式可能會產生過時的結果。此外，不支援使用者定義的函數和使用者定義的程序。因此，資料庫叢集的 `EnableLocalWriteForwarding` 設定預設為關閉。在開啟此功能之前，請檢查以確定您的應用程式程式碼不受上述任何限制的影響。
- 寫入轉送不支援下列 SQL 陳述式類型：

> **Note**
>
> 您可以在應用程式中隱含地使用這些陳述式，或由 PostgreSQL 通訊協定推斷。例如，PL/SQL 例外狀況處理可能會導致使用 SAVEPOINT，這不是支援的陳述式。

  - `ANALYZE`
  - `CLUSTER`
  - `COPY`
  - 游標 – 不支援游標，因此請務必在使用本機寫入轉送之前將其關閉。
  - 資料定義語言 (DDL) 陳述式

- GRANT|REVOKE|REASSIGN OWNED|SECURITY LABEL
- LISTEN / NOTIFY
- LOCK
- SAVEPOINT
- SELECT INTO
- SET CONSTRAINTS
- 序列更新：nextval()、setval()
- TRUNCATE
- 兩階段遞交命令：PREPARE TRANSACTION、COMMIT PREPARED、ROLLBACK PREPARED
- 使用者定義的函數和使用者定義的程序。
- VACUUM

您可以考慮使用下列 SQL 陳述式搭配寫入轉送：

- DML 陳述式可能包含多個部分，例如 INSERT ... SELECT 陳述式或 DELETE ... WHERE 陳述式。在這種情況下，系統會將整個陳述式轉送到寫入器資料庫執行個體並在該處執行陳述式。
- 資料操作語言 (DML) 陳述式，例如 INSERT、DELETE 和 UPDATE。
- 包含此清單中陳述式的 EXPLAIN 陳述式
- PREPARE 和 EXECUTE 陳述式。
- SELECT FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } 陳述式。

設定 Aurora PostgreSQL 以進行本機寫入轉送

使用下列各節，您可以為 Amazon Aurora PostgreSQL 資料庫叢集啟用本機寫入轉送、設定一致性層級，以及使用寫入轉送管理交易。

啟用本機寫入轉送

根據預設，Aurora PostgreSQL 資料庫叢集不會啟用本機寫入轉送功能。您能夠在叢集層級啟用本機寫入轉送，而非執行個體層級。

主控台

使用 AWS 管理主控台，當您建立或修改資料庫叢集時，選取僅供讀取複本寫入轉送中的開啟本機寫入轉送核取方塊。

AWS CLI

若要使用 AWS CLI 啟用本機寫入轉送，請使用 `--enable-local-write-forwarding` 選項。當您使用 `create-db-cluster` 命令建立新的資料庫叢集時，此選項會起作用。當您使用 `modify-db-cluster` 命令修改現有的資料庫叢集時，此選項也會起作用。您可以使用 `--no-enable-local-write-forwarding` 選項搭配這些相同的 CLI 命令來停用本機寫入轉送。

下列範例會建立啟用本機寫入轉送的 Aurora PostgreSQL 資料庫叢集。

```
aws rds create-db-cluster \
--db-cluster-identifier write-forwarding-test-cluster \
--enable-local-write-forwarding \
--engine aurora-postgresql \
--engine-version 16.4 \
--master-username myuser \
--master-user-password mypassword \
--backup-retention 1
```

然後，您可以建立寫入器和讀取器資料庫執行個體，以便您可以使用寫入轉送。如需更多詳細資訊，請參閱 建立 Amazon Aurora 資料庫叢集。

RDS API

若要使用 Amazon RDS API 啟用本機寫入轉送，請將 `EnableLocalWriteForwarding` 參數設定為 `true`。當您使用 `CreateDBCluster` 操作建立新的資料庫叢集時，此參數會起作用。當您使用 `ModifyDBCluster` 操作修改現有的資料庫叢集時，此選項也會起作用。您可以將 `EnableLocalWriteForwarding` 參數設定為 `false` 來停用本機寫入轉送。

啟用資料庫工作階段的本機寫入轉送

`apg_write_forward.consistency_mode` 參數是啟用寫入轉送的資料庫參數和資料庫叢集參數。您可以指定 `SESSION`、`EVENTUAL`、`GLOBAL` 或 `OFF` 以實現讀取一致性層級。若要進一步了解一致性層級，請參閱Aurora PostgreSQL 中本機寫入轉送的一致性與隔離。

下列規則適用於此參數：

- 預設值為 `SESSION`。
- 只有將 `apg_write_forward.consistency_mode` 設定為 `EVENTUAL`、`SESSION` 或 `GLOBAL` 時，才能使用本機寫入轉送。此參數僅在啟用本機寫入轉送之資料庫叢集的讀取器執行個體中啟用。

- 將值設定為 OFF 會停用工作階段中的本機寫入轉送。

Aurora PostgreSQL 中本機寫入轉送的一致性與隔離

您可以控制僅供讀取複本上的讀取一致性程度。您可以調整讀取一致性層級，以確保在任何後續查詢之前，您都可以在僅供讀取複本中看見工作階段中的所有轉送寫入操作。您也可以使用此設定，確保僅供讀取複本上的查詢永遠會看到寫入器資料庫執行個體的最新更新。即使是由其他工作階段或其他叢集所提交的更新。若要為應用程式指定這種行為類型，請為工作階段層級參數 apg_write_forward.consistency_mode 選擇適當的值。該 apg_write_forward.consistency_mode 參數只會對啟用本機寫入轉送的僅供讀取複本產生影響。

> **Note**
>
> 對於 apg_write_forward.consistency_mode 參數，您可以指定值 SESSION、EVENTUAL、GLOBAL 或 OFF。根據預設，系統會將此值設為 SESSION。將值設定為 OFF 會停用寫入轉送。

當您提高一致性層級時，您的應用程式會花費更多時間，等待變更傳播至僅供讀取複本。您可以選擇在較低延遲之間的平衡，並確保在查詢執行之前，在其他位置所做的變更完全可用。

對於每個可用的一致性模式設定，效果如下：

- SESSION – 僅供讀取複本上使用本機寫入轉送的工作階段會看到該工作階段中所做之所有變更的結果。無論交易是否已遞交，這些變更都是可見的。如有必要，查詢會等待轉送寫入操作的結果複寫到目前的讀取器資料庫執行個體。它不會等待來自寫入操作的更新結果，這些寫入操作的執行位置位於目前資料庫叢集內的其他工作階段。
- EVENTUAL – 僅供讀取複本上使用本機寫入轉送的工作階段可能會看到由於複寫延遲而稍微過時的資料。在寫入器資料庫執行個體上執行寫入操作以及複寫至僅供讀取複本之前，不會顯示相同工作階段中的寫入操作結果。查詢不會等待更新的結果變成可用。因此，它可能會擷取較舊的資料或更新的資料，視陳述式的時間和複寫延遲量而定。
- GLOBAL – 僅供讀取複本上的工作階段會看到該工作階段所做的變更。它也會看到寫入器資料庫執行個體和其他僅供讀取複本的所有遞交變更。每個查詢可能會等待一段時間，長短取決於工作階段的延遲量。自查詢開始的時間起，當僅供讀取複本與寫入器資料庫執行個體中的所有遞交資料都是最新時，查詢就會繼續。

> **Note**
>
> 全域一致性模式會影響工作階段中執行的查詢延遲。即使工作階段未傳送任何寫入查詢，也會執行等待。

- OFF – 本機寫入轉送已停用。

在使用寫入轉送的工作階段中，您可以使用 REPEATABLE READ 和 READ COMMITTED 隔離層級。不過，不支援 SERIALIZABLE 隔離層級。

如需寫入轉送中所含所有參數的詳細資訊，請參閱 寫入轉送的預設參數設定。

具有寫入轉送的交易存取模式

如果交易存取模式設定為唯讀，則不會使用本機寫入轉送。您可以在連線至已啟用本機寫入轉送的資料庫叢集和工作階段時，將存取模式設定為僅讀取寫入。

如需交易存取模式的詳細資訊，請參閱設定交易。

使用 Aurora PostgreSQL 的本機寫入轉送

使用下列各節，您可以檢查資料庫叢集是否已啟用本機寫入轉送、檢視相容性考量，以及查看可設定的參數和身分驗證設定。此資訊可讓您取得詳細資訊，以有效利用 Aurora PostgreSQL 中的本機寫入轉送功能。

> **Note**
>
> 當叢集中使用本機寫入轉送的寫入器執行個體重新啟動時，使用本機寫入轉送的讀取器執行個體上任何作用中的轉送交易和查詢都會自動關閉。當寫入器執行個體再次可用之後，您可以重試這些交易。

檢查資料庫叢集是否已啟用本機寫入轉送

若要確定您是否可以在資料庫叢集中使用本機寫入轉送，請確認該叢集的屬性 LocalWriteForwardingStatus 設定為 enabled。

在 AWS 管理主控台 中，叢集的詳細資訊頁面上的組態標籤，您可以看到本機僅供讀取複本寫入轉送的狀態為啟用。

若要查看所有叢集的本機寫入轉送設定狀態，請執行下列 AWS CLI 命令。

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,LocalWriteForwardingStatus:LocalWriteForwardingStatus}

[
{
"LocalWriteForwardingStatus": "enabled",
"DBClusterIdentifier": "write-forwarding-test-cluster-1"
},
{
"LocalWriteForwardingStatus": "disabled",
"DBClusterIdentifier": "write-forwarding-test-cluster-2"
},
{
"LocalWriteForwardingStatus": "requested",
"DBClusterIdentifier": "test-global-cluster-2"
},
{
"LocalWriteForwardingStatus": "null",
"DBClusterIdentifier": "aurora-postgresql-v2-cluster"
}
]
```

資料庫叢集可以具有以下值 `LocalWriteForwardingStatus`：

- `disabled` – 本機寫入轉送已停用。
- `disabling` – 本機寫入轉送正處於停用的過程。
- `enabled` – 本機寫入轉送已啟用。
- `enabling` – 本機寫入轉送正處於啟用的過程。
- `null` – 本機寫入轉送不適用於此資料庫叢集。
- `requested` – 已請求本機寫入轉送，但尚未作用中。

寫入轉送的預設參數設定

Aurora 叢集參數群組包含本機寫入轉送功能的設定。因為這些是叢集參數，所以每個叢集中的所有資料庫執行個體都有這些變數的相同值。下表列出了有關這些參數的詳細資訊，並在表格後面附有使用注意事項。

參數	範圍	類型	預設值	有效值
apg_write_forward.connect_timeout	Session (工作階段)	秒	30	0–2147483647
apg_write_forward.consistency_mode	工作階段 (Session)	enum	Session (工作階段)	SESSION, EVENTUAL, GLOBAL, 和 OFF
apg_write_forward.idle_in_transaction_session_timeout	Session (工作階段)	毫秒	86400000	0–2147483647
apg_write_forward.idle_session_timeout	Session (工作階段)	毫秒	300000	0–2147483647
apg_write_forward.max_forwarding_connections_percent	全球服務	int	25	1–100

apg_write_forward.max_forwarding_connections_percent 參數是可用於處理從讀取器轉送之查詢的資料庫連線上限。此上限的表示方式是寫入器資料庫執行個體的 max_connections 設定百分比。例如，如果 max_connections 是 800，且 apg_write_forward.max_forwarding_connections_percent 是 10，則寫入器允許最多 80 個同時轉送的工作階段。這些連線來自 max_connections 設定所管理的相同連線集區。此設定僅適用於叢集已啟用本機寫入轉送功能的寫入器資料庫執行個體。

使用下列設定來控制本機寫入轉送請求：

- apg_write_forward.consistency_mode – 工作階段層級參數，用於控制僅供讀取複本的讀取一致性程度。有效值為 SESSION、EVENTUAL、GLOBAL 或 OFF。根據預設，系統會將此值設為 SESSION。將值設定為 OFF 會停用工作階段中的本機寫入轉送。若要進一步了解一致性層級，請參

閱Aurora PostgreSQL 中本機寫入轉送的一致性與隔離。此參數僅在啟用本機寫入轉送的讀取器執行個體中啟用。

- `apg_write_forward.connect_timeout` – 在建立寫入器資料庫執行個體的連線時，僅供讀取複本在放棄之前等待的秒數上限。值 `0` 表示無限期等待。
- `apg_write_forward.idle_in_transaction_session_timeout` – 寫入器資料庫執行個體在關閉讀取器執行個體之前，等待從具有開啟交易之僅供讀取複本轉送的連線上活動的毫秒數。如果在此期間之後工作階段在交易中仍處於閒置狀態，則 Aurora 會終止工作階段。值為 `0` 會停用逾時。
- `apg_write_forward.idle_session_timeout` – 寫入器資料庫執行個體在關閉讀取器執行個體之前，等待從僅供讀取複本轉送的連線上活動的毫秒數。如果工作階段在此期間之後仍處於閒置狀態，則 Aurora 會終止工作階段。值為 `0` 會停用逾時。

rdswriteforwarduser

`rdswriteforwarduser` 是我們用來在僅供讀取複本與寫入器資料庫執行個體之間建立連線的使用者。

> **Note**
>
> `rdswriteforwarduser` 透過 PUBLIC 角色將其 CONNECT 權限繼承至客戶資料庫。如果撤銷 PUBLIC 角色的權限，您將需要針對轉送寫入所需的資料庫 GRANT CONNECT 權限。

在 Aurora PostgreSQL 中監控本機寫入轉送

使用下列各節，您可以監控 Aurora PostgreSQL 叢集中的本機寫入轉送，包括相關的 CloudWatch 指標和等待事件，以追蹤效能並識別潛在問題。

用於寫入轉送的 Amazon CloudWatch 指標和 Aurora PostgreSQL 狀態變數

當您在一或多個僅供讀取複本上使用寫入轉送時，下列 Amazon CloudWatch 指標適用於寫入器資料庫執行個體。

CloudWatch 指標	單位和描述
`AuroraLocal ForwardingWriterDMLThroughput`	計數 (每秒) 此寫入器資料庫執行個體每秒處理的轉送 DML 陳述式數目。

CloudWatch 指標	單位和描述
AuroraLocalForwardingWriterOpenSessions	計數。此寫入器資料庫執行個體處理轉送查詢的開啟工作階段數目。
AuroraLocal ForwardingWriterTotalSessions	計數。此寫入器資料庫執行個體上轉送的工作階段總數。

下列 CloudWatch 指標適用於每個僅供讀取複本。這些指標的測量是在啟用本機寫入轉送的資料庫叢集中，每個讀取器資料庫執行個體上進行。

CloudWatch 指標	單位和描述
AuroraForwardingReplicaCommitThroughput	計數 (每秒) 此複本每秒轉送之工作階段中的遞交數目。
AuroraForwardingReplicaDMLLatency	毫秒。複本上轉送 DML 的平均回應時間 (以毫秒為單位)。
AuroraForwardingReplicaDMLThroughput	計數 (每秒) 每秒在此複本上處理的轉送 DML 陳述式數目。
AuroraForwardingReplicaErrorSessionsLimit	計數。由於達到連線數上限或寫入轉送連線數上限而被寫入器資料庫執行個體拒絕的工作階段數目。
AuroraForwardingReplicaOpenSessions	計數。在複本執行個體上使用本機寫入轉送的工作階段數目。
AuroraForwardingReplicaReadWaitLatency	毫秒。複本等待與寫入器資料庫執行個體的 LSN 保持一致的平均等待時間 (毫秒)。讀取器資料庫執行個體等待的程度取決於 apg_write_forward.consistency_mode 設定。如需有關此設定的詳細資訊，請參閱 the section called “Aurora PostgreSQL 中寫入轉送的組態參數”。

Aurora PostgreSQL 中本機寫入轉送的等待事件

當您搭配 Aurora PostgreSQL 使用寫入轉送時，Amazon Aurora 會產生下列等待事件。

主題

- IPC:AuroraWriteForwardConnect
- IPC:AuroraWriteForwardConsistencyPoint
- IPC:AuroraWriteForwardExecute
- IPC:AuroraWriteForwardGetGlobalConsistencyPoint
- IPC:AuroraWriteForwardXactAbort
- IPC:AuroraWriteForwardXactCommit
- IPC:AuroraWriteForwardXactStart

IPC:AuroraWriteForwardConnect

當僅供讀取複本上的後端程序正在等待寫入器資料庫執行個體的連線開啟時，就會發生 `IPC:AuroraWriteForwardConnect` 事件。

等待時間增加的可能原因

隨著從僅供讀取複本到寫入器節點的連線嘗試次數增加，此事件也會增加。

動作

減少從僅供讀取複本到寫入器節點的同時連線數目。

IPC:AuroraWriteForwardConsistencyPoint

`IPC:AuroraWriteForwardConsistencyPoint` 事件描述來自僅供讀取複本上節點的查詢將等待轉送的寫入操作的結果複寫到目前區域的時間。只有在工作階段層級參數 `apg_write_forward.consistency_mode` 設定為下列其中一項時，才會產生此事件：

- `SESSION` – 僅供讀取複本上的查詢會等待該工作階段中所做之所有變更的結果。
- `GLOBAL` – 僅供讀取複本上的查詢會等待該工作階段所做的變更結果，以及寫入器資料庫執行個體和僅供讀取複本的所有遞交變更。

如需有關 `apg_write_forward.consistency_mode` 參數設定的資訊，請參閱 the section called “Aurora PostgreSQL 中寫入轉送的組態參數”。

等待時間增加的可能原因

等待時間較長的常見原因如下：

- 根據 Amazon CloudWatch ReplicaLag 指標所衡量的複本延遲增加。如需此指標的詳細資訊，請參閱 監控 Aurora PostgreSQL 複寫。
- 增加寫入器資料庫執行個體或僅供讀取複本的負載。

動作

請根據應用程式的需求變更一致性模式。

IPC:AuroraWriteForwardExecute

當僅供讀取複本上的後端程序正在等待轉送的查詢完成，並從資料庫叢集的寫入器節點取得結果時，就會發生 IPC:AuroraWriteForwardExecute 事件。

等待時間增加的可能原因

等待時間增加的常見原因包括：

- 從寫入器節點擷取大量的行。
- 寫入器節點與僅供讀取複本之間的網路延遲增加，會增加僅供讀取複本從寫入器節點接收資料所花費的時間。
- 增加僅供讀取複本的負載可能會延遲查詢請求從僅供讀取複本傳輸到寫入器節點。
- 寫入器節點上的負載增加，可能會延遲將資料從寫入器節點傳輸至僅供讀取複本。

動作

根據等待事件的原因，我們會建議不同的動作。

- 最佳化查詢以僅擷取必要的資料。
- 將資料處理語言 (DML) 操作最佳化，以僅修改必要的資料。
- 如果僅供讀取複本或寫入器節點受到 CPU 或網路頻寬的限制，請考慮將其變更為具有更多 CPU 容量或更多網路頻寬的執行個體類型。

IPC:AuroraWriteForwardGetGlobalConsistencyPoint

當僅供讀取複本上使用全域一致性模式的後端程序在執行查詢之前等待從寫入器節點取得全域一致性點時，就會發生 IPC:AuroraWriteForwardGetGlobalConsistencyPoint 事件。

等待時間增加的可能原因

等待時間增加的常見原因包括：

- 僅供讀取複本與寫入器節點之間的網路延遲增加，會增加僅供讀取複本從寫入器節點接收資料所花費的時間。
- 增加僅供讀取複本的負載可能會延遲查詢請求從僅供讀取複本傳輸到寫入器節點。
- 寫入器節點上的負載增加，可能會延遲將資料從寫入器節點傳輸至僅供讀取複本。

動作

根據等待事件的原因，我們會建議不同的動作。

- 請根據應用程式的需求變更一致性模式。
- 如果僅供讀取複本或寫入器節點受到 CPU 或網路頻寬的限制，請考慮將其變更為具有更多 CPU 容量或更多網路頻寬的執行個體類型。

IPC:AuroraWriteForwardXactAbort

僅供讀取複本上的後端程序正在等待遠端清除查詢的結果時，就會發生 IPC:AuroraWriteForwardXactAbort 事件。在中止寫入轉送的交易之後，會發出清除查詢以將程序返回到適當的狀態。Amazon Aurora 執行這些操作可能是因為發現錯誤，也可能是因為使用者發出了明確的 ABORT 命令或取消了正在執行的查詢。

等待時間增加的可能原因

等待時間增加的常見原因包括：

- 僅供讀取複本與寫入器節點之間的網路延遲增加，會增加僅供讀取複本從寫入器節點接收資料所花費的時間。
- 增加僅供讀取複本的負載可能會延遲將清除查詢請求從僅供讀取複本傳輸到寫入器節點。
- 寫入器節點上的負載增加，可能會延遲將資料從寫入器節點傳輸至僅供讀取複本。

動作

根據等待事件的原因，我們會建議不同的動作。

- 調查中止交易的原因。
- 如果僅供讀取複本或寫入器資料庫執行個體受到 CPU 或網路頻寬的限制，請考慮將其變更為具有更多 CPU 容量或更多網路頻寬的執行個體類型。

IPC:AuroraWriteForwardXactCommit

當僅供讀取複本上的後端程序正在等待轉送的 commit 交易命令的結果時，就會發生 `IPC:AuroraWriteForwardXactCommit` 事件。

等待時間增加的可能原因

等待時間增加的常見原因包括：

- 僅供讀取複本與寫入器節點之間的網路延遲增加，會增加僅供讀取複本從寫入器節點接收資料所花費的時間。
- 增加僅供讀取複本的負載可能會延遲查詢請求從僅供讀取複本傳輸到寫入器節點。
- 寫入器節點上的負載增加，可能會延遲將資料從寫入器節點傳輸至僅供讀取複本。

動作

如果僅供讀取複本或寫入器節點受到 CPU 或網路頻寬的限制，請考慮將其變更為具有更多 CPU 容量或更多網路頻寬的執行個體類型。

IPC:AuroraWriteForwardXactStart

當僅供讀取複本上的後端程序正在等待轉送的 start 交易命令的結果時，就會發生 `IPC:AuroraWriteForwardXactStart` 事件。

等待時間增加的可能原因

等待時間增加的常見原因包括：

- 僅供讀取複本與寫入器節點之間的網路延遲增加，會增加僅供讀取複本從寫入器節點接收資料所花費的時間。
- 增加僅供讀取複本的負載可能會延遲查詢請求從僅供讀取複本傳輸到寫入器節點。
- 寫入器節點上的負載增加，可能會延遲將資料從寫入器節點傳輸至僅供讀取複本。

動作

如果僅供讀取複本或寫入器節點受到 CPU 或網路頻寬的限制，請考慮將其變更為具有更多 CPU 容量或更多網路頻寬的執行個體類型。

使用 Aurora PostgreSQL 作為 Amazon Bedrock 的知識庫

您可以使用 Aurora PostgreSQL 資料庫叢集作為 Amazon Bedrock 的知識庫。如需詳細資訊，請參閱在 Amazon Aurora 中建立向量存放區。知識庫會自動擷取在 Amazon S3 儲存貯體中存放的非結構化文字資料、將其轉換為文字區塊和向量，並將其存放在 PostgreSQL 資料庫中。透過生成式 AI 應用程式，您就可以使用 Amazon Bedrock 代理人，查詢在知識庫中存放的資料，並使用那些查詢的結果來增強基礎模型提供的答案。此工作流程稱為檢索增強生成 (RAG)。如需 RAG 的詳細資訊，請參閱檢索增強生成 (RAG)。

如需使用 Aurora PostgreSQL 透過 RAG 建置生成式 AI 應用程式的詳細資訊，請參閱此部落格文章。

主題

- 先決條件
- 準備將 Aurora PostgreSQL 用作為 Amazon Bedrock 知識庫
- 在 Bedrock 主控台中建立知識庫
- 快速建立 Aurora PostgreSQL Amazon Bedrock 知識庫

先決條件

熟悉下列先決條件，以使用 Aurora PostgreSQL 叢集作為 Amazon Bedrock 的知識庫。您需要以高層次將下列服務設為與 Bedrock 搭配使用：

- 在下列任一版本中建立的 Amazon Aurora PostgreSQL 資料庫叢集：
  - 16.1 版和所有更新版本
  - 15.4 和更新版本
  - 14.9 和更新版本
  - 13.12 和更新版本
  - 12.16 和更新版本

> **Note**
>
> 您必須在目標資料庫中啟用 pgvector 延伸模組，以及使用 0.5.0 版或更新版本。如需詳細資訊，請參閱使用 HNSW 索引的 pgvector 第 0.5.0 版。

- RDS 資料 API
- 在 中受管的使用者 AWS Secrets Manager。如需詳細資訊，請參閱使用 Amazon Aurora 和 進行密碼管理 AWS Secrets Manager。

準備將 Aurora PostgreSQL 用作為 Amazon Bedrock 知識庫

請依照以下各節中說明的步驟，準備將 Aurora PostgreSQL 用作為 Amazon Bedrock 知識庫。

建立和設定 Aurora PostgreSQL

若要使用 Aurora PostgreSQL 資料庫叢集設定 Amazon Bedrock，您必須先建立 Aurora PostgreSQL 資料庫叢集，並記下使用 Amazon Bedrock 進行設定的重要欄位。如需建立 Aurora PostgreSQL 資料庫叢集的詳細資訊，請參閱建立 Aurora PostgreSQL 資料庫叢集並與之連線。

- 在建立 Aurora PostgreSQL 資料庫叢集時啟用資料 API。如需有關支援版本的詳細資訊，請參閱使用 Amazon RDS 資料 API。
- 請務必記下 Aurora PostgreSQL 資料庫叢集的 Amazon Resource Name (ARN)。您需要此資訊來設定資料庫叢集，以便與 Amazon Bedrock 搭配使用。如需詳細資訊，請參閱 Amazon Resource Name (ARN)。

連線至資料庫並安裝 pgvector

您可以使用任何連線公用程式連線至 Aurora PostgreSQL。如需這些公用程式的詳細資訊，請參閱連接至 Amazon Aurora PostgreSQL 資料庫叢集。或者，您可以使用 RDS 主控台查詢編輯器來執行查詢。您需要已啟用 RDS 資料 API 的 Aurora 資料庫叢集，才能使用查詢編輯器。

1. 使用主要使用者登入資料庫並設定 pgvector。如果未安裝延伸模組，請使用下列命令：

   ```
   CREATE EXTENSION IF NOT EXISTS vector;
   ```

   使用支援 HNSW 索引的 pgvector 0.5.0 和更高版本。如需詳細資訊，請參閱使用 HNSW 索引的 pgvector 第 0.5.0 版。

2. 若要檢查已安裝的 pg_vector 版本，請使用下列命令：

```
SELECT extversion FROM pg_extension WHERE extname='vector';
```

設定資料庫物件和權限

1. 建立 Bedrock 可用來查詢資料的特定結構描述。使用以下命令來建立結構描述：

```
CREATE SCHEMA bedrock_integration;
```

2. 建立 Bedrock 可用來查詢資料庫的新角色。使用以下命令來建立新角色：

```
CREATE ROLE bedrock_user WITH PASSWORD 'password' LOGIN;
```

> **Note**
>
> 請記下此密碼，因為稍後您需要此資訊來建立 Secrets Manager 密碼。

如果您使用的是 psql 用戶端，請使用下列命令來建立新角色：

```
CREATE ROLE bedrock_user LOGIN;
\PASSWORD password;
```

3. 授予管理 bedrock_integration 結構描述的 bedrock_user 許可。這將提供在結構描述中建立資料表或索引的功能。

```
GRANT ALL ON SCHEMA bedrock_integration to bedrock_user;
```

4. 以 bedrock_user 身分登入，以及在 bedrock_integration schema 中建立資料表。

```
CREATE TABLE bedrock_integration.bedrock_kb (id uuid PRIMARY KEY, embedding
 vector(n), chunks text, metadata json, custom_metadata jsonb);
```

此命令將使用 Titan 嵌入在 bedrock_integration 結構描述中建立 bedrock_kb 資料表。

將 vector(*n*) 資料類型中的 n 取代為您所使用的嵌入模型適當維度。使用以下建議來協助選取維度：

- 若是 Titan 第 2 版模型，使用 `vector(1024)`、`vector(512)` 或 `vector (256)`。若要進一步了解，請參閱 Amazon Titan 嵌入文字。
- 若是 Titan 第 1.2 版模型，使用 `vector(1536)`。若要進一步了解，請參閱 Amazon Titan 多模態嵌入 G1。
- 若是 Cohere Embed 模型，使用 `vector(1024)`。若要進一步了解，請參閱 Cohere Embed 模型。
- 若是 Cohere Embed Multilingual 第 3 版，使用 `vector(1024)`。

前四欄為必要欄位。對於中繼資料處理，Bedrock 會將中繼資料檔案中的資料寫入 `custom_metadata` 欄。如果您打算使用中繼資料和篩選條件，建議您建立此欄。如果您未建立 `custom_metadata` 欄，請在開始擷取之前，為資料表中的每個中繼資料屬性新增個別欄。如需詳細資訊，請參閱設定和自訂查詢和回應產生。

5. 請依照下列步驟建立 Bedrock 用來查詢資料的必要索引：

   - 使用 bedrock 可用來查詢資料的餘弦運算子建立索引。

   ```
   CREATE INDEX ON bedrock_integration.bedrock_kb USING hnsw (embedding
    vector_cosine_ops);
   ```

   - 對於使用平行索引建置的 pgvector 0.6.0 和更高版本，我們建議您將 `ef_construction` 的值設為 256。

   ```
   CREATE INDEX ON bedrock_integration.bedrock_kb USING hnsw (embedding
    vector_cosine_ops) WITH (ef_construction=256);
   ```

   - 建立索引，Bedrock 可使用此索引來查詢文字資料。

   ```
   CREATE INDEX ON bedrock_integration.bedrock_kb USING gin (to_tsvector('simple',
    chunks));
   ```

   - 如果您建立了自訂中繼資料的欄，請建立索引，Bedrock 可使用此索引來查詢中繼資料。

   ```
   CREATE INDEX ON bedrock_integration.bedrock_kb USING gin (custom_metadata);
   ```

在 Secrets Manager 中建立秘密

Secrets Manager 可讓您存放 Aurora 憑證，以便安全地將這些憑證傳輸到應用程式。如果您在建立 Aurora PostgreSQL 資料庫叢集時未選擇 AWS 秘密管理員選項，現在可以建立秘密。如需建立 AWS Secrets Manager 資料庫秘密的詳細資訊，請參閱 AWS Secrets Manager 資料庫秘密。

在 Bedrock 主控台中建立知識庫

在準備將 Aurora PostgreSQL 用作為知識庫向量存放區時，您必須收集下列詳細資訊，您需要將這些資訊提供給 Amazon Bedrock 主控台。

- Amazon Aurora 資料庫叢集 ARN：資料庫叢集的 ARN。
- 秘密 ARN：資料庫叢集之 AWS Secrets Manager 金鑰的 ARN。
- 資料庫名稱：資料庫名稱。例如，您可以使用預設資料庫 *postgres*。
- 資料表名稱：我們建議您在使用與下列類似的命令建立資料表時，提供結構描述的完整名稱：

```
CREATE TABLE bedrock_integration.bedrock_kb;
```

  此命令將在 bedrock_integration 結構描述中建立 bedrock_kb 資料表。
- 建立資料表時，請務必使用指定的欄和資料類型來設定資料表。您可以使用偏好的欄名稱，而不是資料表中列出的欄名稱。請務必記下您選擇的名稱，以便在知識庫設定期間作為參考。

欄名稱	資料類型	Description
id	UUID 主索引鍵	包含每個記錄的唯一識別碼。
區塊	文字	包含來自資料來源的原始文字區塊。
嵌入項目	向量	包含資料來源的向量嵌入項目。
中繼資料	JSON	包含執行來源屬性及啟用資料擷取和查詢所需的中繼資料。
custom_metadata	JSONB	(選用) 定義 Amazon Bedrock 從資料來源寫入中繼資料詳細資訊的目標欄。

您現在可以透過這些詳細資訊，在 Bedrock 主控台中建立知識庫。如需設定向量索引和建立知識庫資訊的更多詳細資訊，請參閱在 Amazon Aurora 中建立向量存放區和在 Amazon Aurora 中建立向量存放區。

新增 Aurora 作為知識庫後，您現在就可以擷取資料來源以進行搜尋和查詢。如需詳細資訊，請參閱將資料來源擷取至知識庫。

快速建立 Aurora PostgreSQL Amazon Bedrock 知識庫

Amazon Bedrock 的檢索增強生成 (RAG) 工作流程會依賴在 Aurora PostgreSQL 資料庫中存放的向量資料來支援內容擷取。先前，將 Aurora PostgreSQL 設定為 Bedrock 知識庫的向量資料存放區是多步驟的程序，需要跨不同使用者界面進行許多手動動作。這使得資料科學家和開發人員難以將 Aurora 用於 Bedrock 專案。

為了改善使用者體驗，AWS 建立了以 CloudFormation 為基礎的全新快速建立選項，可簡化設定程序。透過 Aurora 快速建立，現在按一下就能夠將預先設定的 Aurora PostgreSQL 資料庫叢集佈建為 Amazon Bedrock 知識庫的向量存放區。

主題

- 支援的區域和 Aurora PostgreSQL 版本
- 了解快速建立程序
- 使用 Aurora 快速建立的優勢
- Aurora 快速建立程序的限制

支援的區域和 Aurora PostgreSQL 版本

您可在支援 Amazon Bedrock 知識庫的所有 AWS 區域中使用 Aurora 快速建立選項。根據預設，其會建立版本為 15.7 的 Aurora PostgreSQL 資料庫叢集。如需支援區域的詳細資訊，請參閱 Amazon Bedrock 知識庫支援的模型和區域。

了解快速建立程序

快速建立程序會自動佈建下列資源，以將 Amazon Aurora PostgreSQL 資料庫設定為 Bedrock 知識庫的向量資料存放區：

您帳戶中的 Aurora PostgreSQL 資料庫叢集 (使用預設設定加以設定)。

- 將 ACU (Aurora 容量單位) 從 0 設為 16。這可讓您將向量存放區在不使用時縮減至零，以節省運算成本。稍後可以在 Amazon RDS 主控台中調整 ACU。

- (Hierarchical Navigable Small World) HNSW 索引使用歐幾里德距離作為在 Aurora 中存放的 Bedrock 向量嵌入的相似量值。
- 資料庫執行個體是無伺服器第 2 版執行個體。
- 叢集與預設 VPC 和子網路相關聯，且已啟用 RDS 資料 API。
- 叢集管理員憑證由 AWS Secrets Manager 管理。

除了預設設定外，還會為您進行下列設定。進行此程序時，您會看到工作流程的說明畫面。

- 使用必要的資料庫物件植入 Aurora 叢集：
  - 建立 Bedrock 知識庫所需的 pgvector 延伸模組、結構描述、角色和資料表。
  - 註冊有限權限的資料庫使用者，讓 Bedrock 能夠與叢集互動。
- 整個資源佈建程序都會顯示進度橫幅，讓您能夠追蹤下列子事件的狀態：
  - Aurora 叢集建立
  - 植入 Aurora 叢集
  - 知識庫建立

  即使您離開頁面並返回，橫幅仍會保持可見狀態，直到知識庫完全建立為止。
- 您可以在進度橫幅按一下 `View details`，即可查看每個步驟的狀態。如需知識庫建立期間事件的詳細資訊，請在檢視詳細資料畫面中選擇 CloudFormation 連結。程序完成後，新的 Bedrock 知識庫即可使用。
- 如果您需要參考，可以在 Bedrock 知識庫的標籤中找到所有快速建立資源的堆疊 ID。

建立 Bedrock 知識庫，同時將新佈建的 Aurora 叢集設為向量存放區。

使用 Aurora 快速建立的優勢

- 以 CloudFormation 為基礎的快速建立程序可大幅減少使用 Aurora 作為向量存放區所需的時間和複雜性。
- Aurora 提供絕佳的效能、向量可擴展性和成本效益，能夠在不使用時將運算費用縮減至零。
- 快速建立程序可簡化端對端體驗，讓您使用 Aurora 即可輕鬆建立和設定 Bedrock 知識庫。
- 客戶可以根據 CloudFormation 範本進行建置，以便使用自己的組態自訂佈建。

Aurora 快速建立程序的限制

- 使用 Aurora 快速建立選項時，會使用預設組態佈建資料庫叢集。不過，這些預設設定可能不符合特定需求或預期使用案例。快速建立不提供在佈建程序期間修改組態的選項。系統會自動設定這些組態，以簡化部署體驗。如果需要自訂 Aurora 資料庫叢集組態，則可以在初始部署之後，運用 Amazon RDS 主控台中的快速建立執行此操作。
- 雖然快速建立流程簡化了設定程序，但 Aurora 資料庫叢集的建立時間仍需約 10 分鐘的時間 (與手動部署相同)。這是因為佈建 Aurora 基礎設施所需的時間。
- 快速建立選項專為實驗和快速設定而設計。透過快速建立而建立的資源可能不適合生產使用，而您將無法將其直接遷移到 VPC 中的生產環境。

將 Amazon Aurora PostgreSQL 與其他 AWS 服務整合

Amazon Aurora 與其他 AWS 服務整合，因此您可以擴展 Aurora PostgreSQL 資料庫叢集，以在 AWS 雲端中使用其他功能。您的 Aurora PostgreSQL 資料庫叢集可以使用 AWS 服務來執行下列動作：

- 使用 Amazon RDS Performance Insights 快速收集、檢視和評估 Aurora PostgreSQL 資料庫執行個體的效能。績效詳情會延伸現有 Amazon RDS 監控功能的基礎，藉此說明資料庫效能，並幫助您分析可能影響效能的任何問題。利用績效詳情儀表板，您可以將資料庫負載視覺化，並依等候、SQL 陳述式、主機或使用者篩選負載。如需 Performance Insights 的詳細資訊，請參閱在 Amazon Aurora 上使用績效詳情監控資料庫負載。
- 設定 Aurora PostgreSQL 資料庫叢集，以將日誌資料發佈至 Amazon CloudWatch Logs。CloudWatch Logs 提供了高耐用性儲存空間，可供您存放日誌資料。使用 CloudWatch Logs，您可以執行日誌資料的即時分析，並使用 CloudWatch 來建立警示和檢視指標。如需詳細資訊，請參閱將 Aurora PostgreSQL 日誌發佈至 Amazon CloudWatch Logs。
- 將資料從 Amazon S3 儲存貯體匯入 Aurora PostgreSQL 資料庫叢集，或將資料從 Aurora PostgreSQL 資料庫叢集匯出至 Amazon S3 儲存貯體。如需詳細資訊，請參閱將資料從 Amazon S3 匯入 Aurora PostgreSQL 資料庫叢集 和將資料從 Aurora PostgreSQL 資料庫叢集匯出至 Amazon S3。
- 使用 SQL 語言，將以機器學習為基礎的預測新增至資料庫應用程式。Aurora Machine Learning 使用 Aurora 資料庫與 AWS 機器學習 (ML) 服務 SageMaker AI 和 Amazon Comprehend 之間的高度最佳化整合。如需詳細資訊，請參閱將 Amazon Aurora Machine Learning 與 Aurora PostgreSQL 搭配使用。

- 從 Aurora PostgreSQL 資料庫叢集叫用 AWS Lambda 函數。若要執行此操作，請使用 Aurora PostgreSQL 提供的 aws_lambda PostgreSQL 擴充功能。如需詳細資訊，請參閱從 Aurora PostgreSQL 資料庫叢集 叫用 AWS Lambda 函數。
- 整合來自 Amazon Redshift 和 Aurora PostgreSQL 的查詢。如需詳細資訊，請參閱《Amazon Redshift 資料庫開發人員指南》中的開始對 PostgreSQL 使用聯合查詢。

將資料從 Amazon S3 匯入 Aurora PostgreSQL 資料庫叢集

您可以將使用 Amazon Simple Storage Service 儲存的資料匯入 Aurora PostgreSQL 資料庫叢集。要執行此操作，您首先安裝 Aurora PostgreSQL aws_s3 擴充功能。此擴充功能提供可用於從 Amazon S3 儲存貯體匯入資料的函數。儲存貯體是物件或檔案的 Amazon S3 容器。資料可以是逗號分隔值 (CSV) 檔案、文字檔案或壓縮 (gzip) 檔案。從下文中，您可以了解如何安裝擴充功能，以及如何將資料從 Amazon S3 匯入資料表。

您的資料庫必須執行 PostgreSQL 10.7 版或更高版本，才能從 Simple Storage Service (Amazon S3) 匯入 。Aurora PostgreSQL。

如果您沒有資料儲存於 Amazon S3 上，您需要先建立儲存貯體並儲存資料。如需詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的下列主題：

- 建立儲存貯體
- 將物件新增到儲存貯體

支援從 Amazon S3 匯入跨帳戶。如需更多詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的授予跨帳戶許可。

從 S3 匯入資料時，您可以使用客戶受管金鑰進行加密。如需詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的 儲存在 AWS KMS 中的 KMS 金鑰。

> **Note**
>
> Aurora Serverless v1 不支援從 Amazon S3 匯入資料。Aurora Serverless v2 則支援。

主題

- 安裝 aws_s3 擴充功能
- 從 Amazon S3 資料匯入資料的概觀

- 設定對 Amazon S3 儲存貯體的存取權
- 將資料從 Amazon S3 匯入 Aurora PostgreSQL 資料庫叢集
- 函數參考

安裝 aws_s3 擴充功能

您需要先安裝 aws_s3 擴充功能，才能將 Amazon S3 與 Aurora PostgreSQL 資料庫叢集 搭配使用。此擴充功能提供從 Amazon S3 匯入資料的函數。它還提供了從 Aurora PostgreSQL 資料庫叢集執行個體中匯出資料到 Amazon S3 儲存貯體的功能。如需更多詳細資訊，請參閱 將資料從 Aurora PostgreSQL 資料庫叢集匯出至 Amazon S3。aws_s3 擴充功能取決於 aws_commons 擴充功能中的一些輔助函數，需要時會自動安裝。

安裝 **aws_s3** 擴充功能

1. 使用 psql (或 pgAdmin) 以具有 rds_superuser 權限的使用者身分連接到 Aurora PostgreSQL 資料庫叢集的寫入器執行個體 。若您在安裝程序期間保留預設名稱，則連接為 postgres。

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. 若要安裝擴充功能，請執行下列命令。

```
postgres=> CREATE EXTENSION aws_s3 CASCADE;
NOTICE: installing required extension "aws_commons"
CREATE EXTENSION
```

3. 若要驗證是否已經安裝擴充功能，可以使用 psql \dx 中繼命令。

```
postgres=> \dx
       List of installed extensions
    Name     | Version |   Schema   |                Description
-------------+---------+------------+-------------------------------------------
 aws_commons | 1.2     | public     | Common data types across AWS services
 aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
 plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

現在可以使用從 Amazon S3 匯入和匯出資料的功能。

從 Amazon S3 資料匯入資料的概觀

將 S3 資料匯入 Aurora PostgreSQL

首先，收集您需要提供給函數的詳細資訊。其中包括 Aurora PostgreSQL 資料庫叢集執行個體上的資料表名稱、以及儲存貯體名稱、檔案路徑、檔案類型，以及 Amazon S3 資料的存放 AWS 區域 位置。如需詳細資訊，請參閱 Amazon Simple Storage Service 使用者指南中的檢視物件。

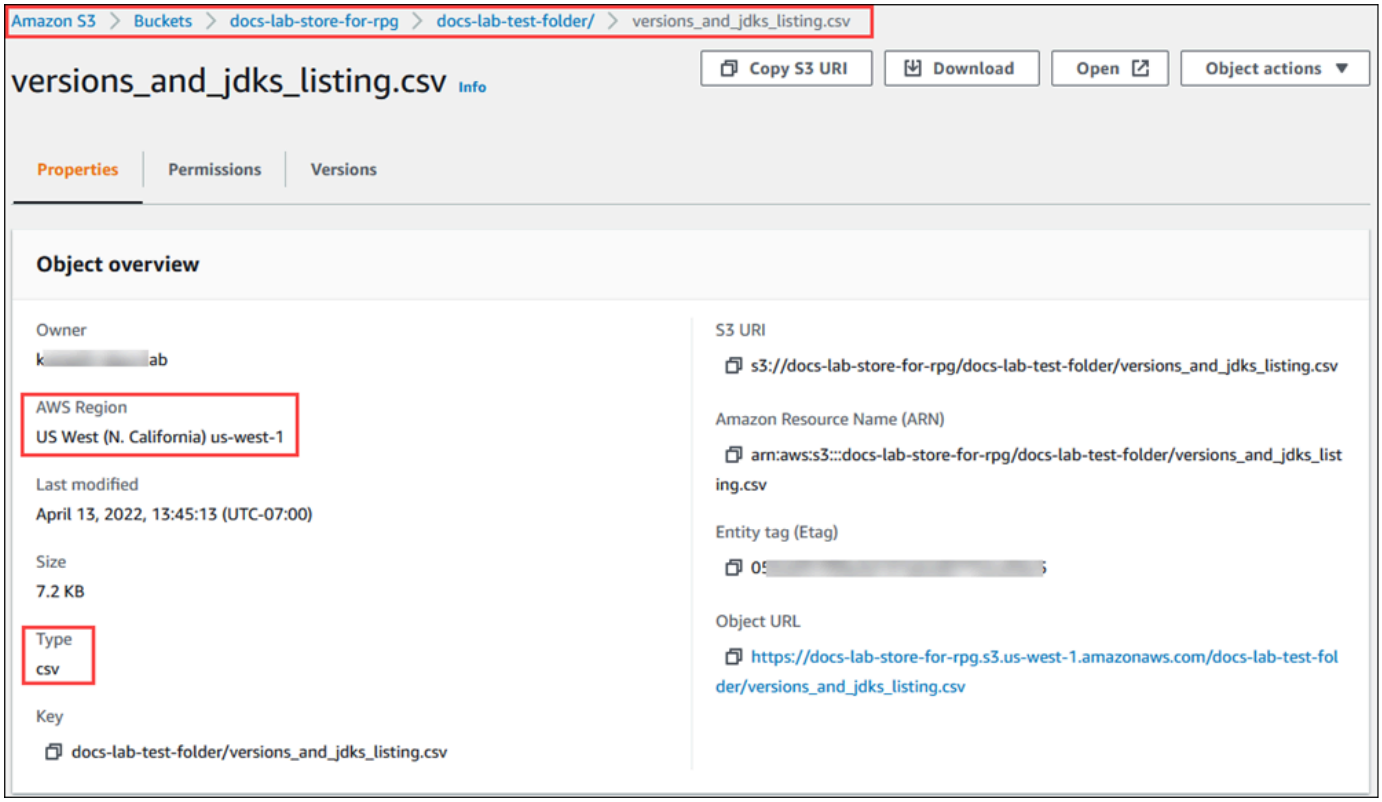
> **Note**
>
> 目前不支援從 Amazon S3 匯入多重部分資料。

1. 取得 `aws_s3.table_import_from_s3` 函數要匯入資料的資料表名稱。舉例來說，下列命令建立的資料表 `t1`，可用於之後的步驟中。

   ```
   postgres=> CREATE TABLE t1
       (col1 varchar(80),
       col2 varchar(80),
       col3 varchar(80));
   ```

2. 取得 Amazon S3 儲存貯體以及要匯入資料的詳細資料。要執行此操作，請在以下網址開啟 Amazon S3 主控台：https://console.aws.amazon.com/s3/，然後選擇 Buckets (儲存貯體)。在清單中尋找包含您資料的儲存貯體。選擇儲存貯體，開啟其物件概觀頁面，然後選擇 Properties (屬性)。

   請記下儲存貯體名稱、路徑 AWS 區域、 和 檔案類型。您稍後需要 Amazon Resource Name (ARN)，以設定透過 IAM 角色對 Amazon S3 的存取權限。如需詳細資訊，請參閱 設定對 Amazon S3 儲存貯體的存取權。下圖顯示範例。



3. 您可以使用 AWS CLI 命令 來驗證 Amazon S3 儲存貯體上資料的路徑`aws s3 cp`。如果資訊正確，此命令會下載 Amazon S3 檔案的副本。

```
aws s3 cp s3://amzn-s3-demo-bucket/sample_file_path ./
```

4. 設定 Aurora PostgreSQL 資料庫叢集 的許可，以允許存取 Amazon S3 儲存貯體上的檔案。若要這樣做，您可以使用 AWS Identity and Access Management (IAM) 角色或安全登入資料。如需詳細資訊，請參閱設定對 Amazon S3 儲存貯體的存取權。

5. 將收集到的路徑和其他 Amazon S3 物件詳細資訊 (請參閱步驟 2) 提供給 `create_s3_uri` 函數，以建構 Amazon S3 URI 物件。若要進一步了解此函數，請參閱 aws_commons.create_s3_uri。以下是在 psql 工作階段中建構此物件的範例。

```
postgres=> SELECT aws_commons.create_s3_uri(
   'docs-lab-store-for-rpg',
   'versions_and_jdks_listing.csv',
   'us-west-1'
) AS s3_uri \gset
```

在下一個步驟中，您將此物件 (`aws_commons._s3_uri_1`) 傳遞到 `aws_s3.table_import_from_s3` 函數，將資料匯入資料表。

6. 調用 `aws_s3.table_import_from_s3` 函數，將資料從 Amazon S3 匯入資料表。如需參考資訊，請參閱 aws_s3.table_import_from_s3。如需範例，請參閱 將資料從 Amazon S3 匯入 Aurora PostgreSQL 資料庫叢集 。

設定對 Amazon S3 儲存貯體的存取權

若要從 Amazon S3 檔案匯入資料，請為 Aurora PostgreSQL 資料庫叢集提供許可，以便存取檔案所在的 Amazon S3 儲存貯體。您可以透過以下兩個方式中的一個提供存取給 Amazon S3 儲存貯體，如下列主題中所述。

主題

- 使用 IAM 角色存取 Amazon S3 儲存貯體
- 使用安全登入資料存取 Amazon S3 儲存貯體
- 對 Amazon S3 的存取進行故障診斷

使用 IAM 角色存取 Amazon S3 儲存貯體

在您由 Amazon S3 檔案載入資料之前，請為 Aurora PostgreSQL 資料庫叢集 提供許可，以便存取檔案所在的 Amazon S3 儲存貯體。這樣您就不必管理額外的登入資料資訊或在 aws_s3.table_import_from_s3 函數呼叫中提供它。

若要執行此動作，請建立可提供 Amazon S3 儲存貯體存取的 IAM 政策。建立 IAM 角色，並將政策連接到該角色。然後將 IAM 角色指派到您的資料庫叢集。

> **Note**
>
> 您無法建立 IAM 角色與 Aurora Serverless v1 資料庫叢集的關聯，因此以下步驟不適用。

透過 IAM 角色將 Aurora PostgreSQL 資料庫叢集 的存取權授予 Amazon S3

1. 建立 IAM 政策。

   此政策可提供儲存貯體及物件許可，讓 Aurora PostgreSQL 資料庫叢集 能夠存取 Amazon S3。

   在政策中納入下列必要動作，以允許由 Amazon S3 儲存貯體傳輸檔案至 Aurora PostgreSQL：

   - `s3:GetObject`

- `s3:ListBucket`

在政策中包含下列資源，以識別 Amazon S3 儲存貯體和儲存貯體中的物件。這會顯示用於存取 Amazon S3 的 Amazon Resource Name (ARN) 格式。

- arn:aws:s3:::*amzn-s3-demo-bucket*
- arn:aws:s3:::*amzn-s3-demo-bucket*/*

如需建立 Aurora PostgreSQL IAM 政策的詳細資訊，請參閱 建立並使用 IAM 政策進行 IAM 資料庫存取。另請參閱《IAM 使用者指南》中的教學：建立和連接您的第一個客戶受管原則。

下列 AWS CLI 命令會使用這些選項建立名為 `rds-s3-import-policy`的 IAM 政策。其將存取權授予名為 *amzn-s3-demo-bucket* 的儲存貯體。

> **Note**
>
> 請記下此命令所傳回政策的 Amazon Resource Name (ARN)。在後續步驟中將政策連接至 IAM 角色時，您會需要此 ARN。

Example

針對 Linux、macOS 或 Unix：

```
aws iam create-policy \
   --policy-name rds-s3-import-policy \
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3import",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::amzn-s3-demo-bucket",
           "arn:aws:s3:::amzn-s3-demo-bucket/*"
```

```
        ]
      }
    ]
  }'
```

在 Windows 中：

```
aws iam create-policy ^
   --policy-name rds-s3-import-policy ^
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3import",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::amzn-s3-demo-bucket",
           "arn:aws:s3:::amzn-s3-demo-bucket/*"
         ]
       }
     ]
   }'
```

2. 建立 IAM 角色。

   這麼做是為了讓 Aurora PostgreSQL 擔任此 IAM 角色，以存取您的 Amazon S3 儲存貯體。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以將許可委派給 IAM 使用者。

   建議您在資源型政策中使用 aws:SourceArn 和 aws:SourceAccount 全域條件內容金鑰，將服務的許可限定於特定資源。這是防止混淆代理人問題最有效的方式。

   如果同時使用這兩個全域條件內容索引鍵，且 aws:SourceArn 值包含帳戶 ID，則在相同政策陳述式中使用 aws:SourceAccount 值和 aws:SourceArn 值中的帳戶時，必須使用相同的帳戶 ID。

   - 如果您想要跨服務存取單一資源，請使用 aws:SourceArn。
   - 如果您想要允許該帳戶中的任何資源與跨服務使用相關聯，請使用 aws:SourceAccount。

在政策中，請務必搭配資源的完整 ARN 來使用 `aws:SourceArn` 全域條件內容金鑰。下列範例示範如何使用 AWS CLI 命令來建立名為 的角色`rds-s3-import-role`。

Example

針對 Linux、macOS 或 Unix：

```
aws iam create-role \
   --role-name rds-s3-import-role \
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:clustername"
                }
             }
       }
     ]
   }'
```

在 Windows 中：

```
aws iam create-role ^
   --role-name rds-s3-import-role ^
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
```

```
          "Action": "sts:AssumeRole",
          "Condition": {
              "StringEquals": {
                 "aws:SourceAccount": "111122223333",
                 "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:clustername"
                 }
              }
       }
     ]
   }'
```

3. 將您建立的 IAM 政策附加至您建立的 IAM 角色。

   下列 AWS CLI 命令會將上一個步驟中建立的政策連接至名為 `rds-s3-import-role` Replace *`your-policy-arn`* with the policy ARN 的角色，該角色是您在先前步驟中記下的政策 ARN。

   Example

   針對 Linux、macOS 或 Unix：

   ```
   aws iam attach-role-policy \
      --policy-arn your-policy-arn \
      --role-name rds-s3-import-role
   ```

   在 Windows 中：

   ```
   aws iam attach-role-policy ^
      --policy-arn your-policy-arn ^
      --role-name rds-s3-import-role
   ```

4. 將 IAM 角色新增至資料庫叢集。

   您可以使用 AWS 管理主控台 或 來執行此操作 AWS CLI，如下所述。

主控台

使用主控台為 PostgreSQL 資料庫叢集新增 IAM 角色

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。

2. 選擇 PostgreSQL 資料庫叢集名稱以顯示其詳細資訊。
3. 在 Connectivity & security (連線能力與安全) 標籤上，在 Manage IAM roles (管理 IAM 角色) 區段中，在 Add IAM roles to this cluster (將 IAM 角色新增到此叢集執行個體) 下方，選擇要新增的角色。
4. 請在 Feature (功能) 下，選擇 s3Import。
5. 選擇 Add role (新增角色)。

AWS CLI

使用 CLI 為 PostgreSQL 資料庫叢集新增 IAM 角色

- 使用下列命令將角色新增至名為 `my-db-cluster` 的 PostgreSQL 資料庫叢集。將 *`your-role-arn`* 替換為您前個步驟記下的角色 ARN。使用 `s3Import` 作為 `--feature-name` 選項的值。

  Example

  針對 Linux、macOS 或 Unix：

```
aws rds add-role-to-db-cluster \
   --db-cluster-identifier my-db-cluster \
   --feature-name s3Import \
   --role-arn your-role-arn   \
   --region your-region
```

  在 Windows 中：

```
aws rds add-role-to-db-cluster ^
   --db-cluster-identifier my-db-cluster ^
   --feature-name s3Import ^
   --role-arn your-role-arn ^
   --region your-region
```

RDS API

若要使用 Amazon RDS API 為 PostgreSQL 資料庫叢集新增 IAM 角色，請呼叫 AddRoleToDBCluster 操作。

使用安全登入資料存取 Amazon S3 儲存貯體

如果想要，可以使用安全登入資料來提供 Amazon S3 儲存貯體的存取，而非使用 IAM 角色提供存取。要執行此操作，您可以指定 aws_s3.table_import_from_s3 函數呼叫中的 credentials 參數。

credentials 參數是類型 的結構aws_commons._aws_credentials_1，其中包含 AWS 登入資料。請使用 aws_commons.create_aws_credentials 函數在 aws_commons._aws_credentials_1 結構設定存取金鑰及秘密金鑰，如下所示。

```
postgres=> SELECT aws_commons.create_aws_credentials(
   'sample_access_key', 'sample_secret_key', '')
AS creds \gset
```

建立 aws_commons._aws_credentials_1 結構之後，請使用 aws_s3.table_import_from_s3 函數搭配 credentials 參數來匯入資料，如下所示。

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't', '', '(format csv)',
   :'s3_uri',
   :'creds'
);
```

或是您可在 aws_commons.create_aws_credentials 函數呼叫之中內嵌 aws_s3.table_import_from_s3 函數呼叫。

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't', '', '(format csv)',
   :'s3_uri',
   aws_commons.create_aws_credentials('sample_access_key', 'sample_secret_key', '')
);
```

對 Amazon S3 的存取進行故障診斷

如果您在嘗試從 Amazon S3 匯入資料時遇到問題，請參閱下文以取得建議：

- 對 Amazon Aurora 身分與存取進行故障診斷
- 《Amazon Simple Storage Service 使用者指南》中的針對 Amazon S3 進行故障診斷。
- 《IAM 使用者指南》中的針對 Amazon S3 和 IAM 進行故障診斷

將資料從 Amazon S3 匯入 Aurora PostgreSQL 資料庫叢集

您可以使用 aws_s3 擴充功能的 `table_import_from_s3` 函數，由 Amazon S3 儲存貯體匯入資料。如需參考資訊，請參閱 aws_s3.table_import_from_s3。

> **Note**
>
> 下列範例使用 IAM 角色方法，允許對 Amazon S3 儲存貯體的存取。因此，`aws_s3.table_import_from_s3` 函數呼叫不包含登入資料參數。

以下顯示一個典型範例。

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't1',
   '',
   '(format csv)',
   :'s3_uri'
);
```

參數如下：

- `t1` – PostgreSQL 資料庫叢集的表格名稱，資料會複製到此表格。
- `''` – 資料庫表格之中選用的欄清單。您可使用此參數指出哪些 S3 資料欄要置於哪些表格欄。如果沒有指定欄，所有欄都會複製到表格中。如需使用欄清單的範例，請參閱 匯入使用自訂分隔符號的 Amazon S3 檔案。
- `(format csv)` – PostgreSQL COPY 引數。複製程序使用 PostgreSQL COPY 命令的引數及格式匯入資料。格式的選擇包括逗號分隔值 (CSV)、文字和二進位。預設為文字。
- `s3_uri` – 包含識別 Amazon S3 檔案資訊的結構。如需使用 aws_commons.create_s3_uri 函數來建立 `s3_uri` 結構的範例，請參閱 從 Amazon S3 資料匯入資料的概觀。

如需此函數狀態的詳細資訊，請參閱 aws_s3.table_import_from_s3。

`aws_s3.table_import_from_s3` 函數傳回文字。若要指定從 Amazon S3 儲存貯體匯入的其他檔案類型，請參閱下列其中一個範例。

> **Note**
>
> 匯入 0 位元組檔案將導致錯誤。

主題

- 匯入使用自訂分隔符號的 Amazon S3 檔案
- 匯入 Amazon S3 壓縮 (gzip) 檔案
- 匯入編碼的 Amazon S3 檔案

匯入使用自訂分隔符號的 Amazon S3 檔案

下列範例顯示如何匯入使用自訂分隔符號的檔案。其中也顯示如何使用 `column_list` 函數的 aws_s3.table_import_from_s3 參數，控制資料放置於資料庫表格的位置。

我們在此範例假設下列資訊整理至 Amazon S3 檔案之中的縱線分隔欄。

```
1|foo1|bar1|elephant1
2|foo2|bar2|elephant2
3|foo3|bar3|elephant3
4|foo4|bar4|elephant4
...
```

匯入使用自訂分隔符號的檔案

1. 在資料庫為匯入資料建立表格。

   ```
   postgres=> CREATE TABLE test (a text, b text, c text, d text, e text);
   ```

2. 使用以下的 aws_s3.table_import_from_s3 函數格式由 Amazon S3 檔案匯入資料。

   您可在 aws_commons.create_s3_uri 函數呼叫之中內嵌 `aws_s3.table_import_from_s3` 函數呼叫以指定檔案。

   ```
   postgres=> SELECT aws_s3.table_import_from_s3(
      'test',
      'a,b,d,e',
      'DELIMITER ''|''',
      aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'pipeDelimitedSampleFile', 'us-
   east-2')
   ```

```
);
```

資料目前位於下列欄的表格中。

```
postgres=> SELECT * FROM test;
a | b | c | d | e
---+------+---+---+------+-----------
1 | foo1 | | bar1 | elephant1
2 | foo2 | | bar2 | elephant2
3 | foo3 | | bar3 | elephant3
4 | foo4 | | bar4 | elephant4
```

匯入 Amazon S3 壓縮 (gzip) 檔案

下列範例顯示如何由以 gzip 壓縮的 Amazon S3 匯入檔案。匯入的檔案需具有以下 Amazon S3 中繼資料：

- 索引鍵：`Content-Encoding`
- 值：`gzip`

如果您使用 上傳檔案 AWS 管理主控台，系統通常會套用中繼資料。如需有關使用 AWS 管理主控台、AWS CLI或 API 將檔案上傳至 Amazon S3 的資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的上傳物件。

如需 Amazon S3 中繼資料的詳細資訊以及系統所提供中繼資料的詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的在 Amazon S3 主控台中編輯物件中繼資料。

請依據以下所示內容，將 gzip 檔案匯入 Aurora PostgreSQL 資料庫叢集 。

```
postgres=> CREATE TABLE test_gzip(id int, a text, b text, c text, d text);
postgres=> SELECT aws_s3.table_import_from_s3(
 'test_gzip', '', '(format csv)',
 'amzn-s3-demo-bucket', 'test-data.gz', 'us-east-2'
);
```

匯入編碼的 Amazon S3 檔案

下列範例顯示如何由採用 Windows-1252 編碼的 Amazon S3 匯入檔案。

```
postgres=> SELECT aws_s3.table_import_from_s3(
```

```
 'test_table', '', 'encoding ''WIN1252''',
 aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'SampleFile', 'us-east-2')
);
```

函數參考

函數

- aws_s3.table_import_from_s3
- aws_commons.create_s3_uri
- aws_commons.create_aws_credentials

aws_s3.table_import_from_s3

將 Amazon S3 資料匯入 Aurora PostgreSQL 表。aws_s3 擴充功能提供 aws_s3.table_import_from_s3 函數。傳回值為文字。

語法

必要的參數為 table_name、column_list 及 options。這些參數可識別資料庫表格，並指定資料要如何複製到表格中。

您也可以使用下列參數：

- s3_info 參數指定要匯入的 Amazon S3 檔案。您使用此參數時，IAM 角色會將 Amazon S3 存取權提供給 PostgreSQL 資料庫叢集。

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   s3_info aws_commons._s3_uri_1
)
```

- credentials 參數指定登入資料以存取 Amazon S3。您使用此項參數時，不必使用 IAM 角色。

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   s3_info aws_commons._s3_uri_1,
```

```
   credentials aws_commons._aws_credentials_1
)
```

Parameters

table_name

必要的文字字串，其中含有要匯入資料的 PostgreSQL 資料庫表格名稱。

column_list

必要的文字字串，其中含有要複製資料的 PostgreSQL 資料庫表格欄選用清單。如果字串為空白，就會使用表格的所有欄。如需範例，請參閱 匯入使用自訂分隔符號的 Amazon S3 檔案。

options

必要的文字字串，含有 PostgreSQL COPY 命令引數。這些引數指定資料要如何複製到 PostgreSQL 表格中。詳細資訊請參閱 PostgreSQL COPY 文件。

s3_info

aws_commons._s3_uri_1 複合類型，含有下列 S3 物件相關資訊：

- bucket – 含有檔案的 Amazon S3 儲存貯體名稱。
- file_path – 包括檔案路徑的 Amazon S3 檔案名稱。
- region – 檔案所在的 AWS 區域。如需 AWS 區域名稱和相關值的清單，請參閱 區域和可用區域。

登入資料

aws_commons._aws_credentials_1 複合類型，含有下列登入資料以用於匯入作業：

- 存取金鑰
- 私密金鑰
- 工作階段字符

如需建立 aws_commons._aws_credentials_1 複合結構的詳細資訊，請參閱 aws_commons.create_aws_credentials。

替代語法

為了協助進行測試，您可使用一組更大的參數取代 s3_info 及 credentials 參數。以下是 aws_s3.table_import_from_s3 函數的額外語法變化：

- 請不要使用 `s3_info` 參數識別 Amazon S3 檔案，而是使用 `bucket`、`file_path` 及 `region` 參數組合進行。使用這種形式的函數，Amazon S3 存取權會由 IAM 角色在 PostgreSQL 資料庫執行個體提供。

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   bucket text,
   file_path text,
   region text
)
```

- 請不要使用 `credentials` 參數識別 Amazon S3 存取，而是使用 `access_key`、`session_key` 及 `session_token` 參數組合進行。

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   bucket text,
   file_path text,
   region text,
   access_key text,
   secret_key text,
   session_token text
)
```

替代參數

bucket

文字字串，其中含有包含檔案的 Amazon S3 儲存貯體名稱。

file_path

包含 Amazon S3 檔案名稱 (包括檔案路徑) 的文字字串。

region

識別檔案 AWS 區域 位置的文字字串。如需 AWS 區域 名稱和相關值的清單，請參閱 區域和可用區域。

access_key

文字字串，其中含有用於匯入作業的存取金鑰。預設值為 NULL。

secret_key

文字字串，其中含有用於匯入作業的秘密金鑰。預設值為 NULL。

session_token

(選用) 文字字串，其中含有用於匯入作業的工作階段金鑰。預設值為 NULL。

aws_commons.create_s3_uri

建立 aws_commons._s3_uri_1 結構以保留 Amazon S3 檔案資訊。使用 aws_commons.create_s3_uri 函數 s3_info 參數的 aws_s3.table_import_from_s3 函數結果。

語法

```
aws_commons.create_s3_uri(
   bucket text,
   file_path text,
   region text
)
```

Parameters

bucket

必要的文字字串，其中含有檔案的 Amazon S3 儲存貯體名稱。

file_path

包含 Amazon S3 檔案名稱 (包括檔案路徑) 的必要文字字串。

region

必要的文字字串 AWS 區域 ，其中包含 檔案所在的 。如需 AWS 區域 名稱和相關值的清單，請參閱 區域和可用區域。

aws_commons.create_aws_credentials

在 aws_commons._aws_credentials_1 結構設定存取金鑰及秘密金鑰。使用 aws_commons.create_aws_credentials 函數 credentials 參數的 aws_s3.table_import_from_s3 函數結果。

語法

```
aws_commons.create_aws_credentials(
   access_key text,
   secret_key text,
   session_token text
)
```

Parameters

access_key

必要的文字字串，其中含有用於匯入 Amazon S3 檔案的存取金鑰。預設值為 NULL。

secret_key

必要的文字字串，其中含有用於匯入 Amazon S3 檔案的秘密金鑰。預設值為 NULL。

session_token

選用的文字字串，其中含有用於匯入 Amazon S3 檔案的工作階段字符。預設值為 NULL。如果您提供選用的 session_token，就可以使用臨時登入資料。

將資料從 Aurora PostgreSQL 資料庫叢集匯出至 Amazon S3

您可以從 Aurora PostgreSQL 資料庫叢集中查詢資料，然後將資料直接匯出至 Amazon S3 儲存貯體中存放的檔案中。若要這麼做，首先要安裝 Aurora PostgreSQL aws_s3 擴充功能。此擴充功能提供函數，可用於匯出資料至 Amazon S3。接著，您可以了解如何安裝擴充功能，以及如何將資料匯出至 Amazon S3。

您僅可從已佈建的或 Aurora Serverless v2 資料庫執行個體匯出。這些步驟不支援 Aurora Serverless v1。

> **Note**
>
> 不支援跨帳戶匯出至 Amazon S3。

所有目前可用的 Aurora PostgreSQL 版本都支援將快照資料匯出至 Amazon 簡單儲存服務。如需詳細的版本資訊，請參閱《Aurora PostgreSQL 版本資訊》中的 Amazon Aurora PostgreSQL 更新。

如果您沒有為匯出設定儲存貯體，請參閱下列主題 Amazon Storage Service 使用者指南。

- 設定 Amazon S3
- 建立儲存貯體

根據預設，從 Aurora PostgreSQL 匯出至 Amazon S3 的資料會使用伺服器端加密 AWS 受管金鑰。或者，您可以使用已建立的客戶受管金鑰。如果您使用儲存貯體加密，Amazon S3 儲存貯體必須使用 AWS Key Management Service (AWS KMS) 金鑰 (SSE-KMS) 加密。目前不支援使用 Amazon S3 受管金鑰 (SSE-S3) 加密的儲存貯體。

> **Note**
>
> 您可以使用 AWS 管理主控台 AWS CLI或 Amazon RDS API，將資料庫和資料庫叢集快照資料儲存至 Amazon S3。如需詳細資訊，請參閱將資料庫叢集快照資料匯出至 Amazon S3。

主題

- 安裝 aws_s3 擴充功能
- 將資料匯出至 Amazon S3 的概觀
- 指定要匯出的 Amazon S3 檔案路徑
- 設定對 Amazon S3 儲存貯體的存取權
- 使用 aws_s3.query_export_to_s3 函數匯出查詢資料
- 函數參考
- 對 Amazon S3 的存取進行故障診斷

安裝 aws_s3 擴充功能

在您可以使用 Amazon Simple Storage Service 搭配 Aurora PostgreSQL 資料庫叢集、 您需要安裝 aws_s3 擴充功能。此擴充功能提供從 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，匯出資料至 Amazon S3 儲存貯體的功能。它還提供可從 Amazon S3 匯入資料的函數。如需詳細資訊，請參閱將資料從 Amazon S3 匯入 Aurora PostgreSQL 資料庫叢集 。aws_s3 擴充功能取決於 aws_commons 擴充功能中的一些輔助函數，需要時會自動安裝。

安裝 **aws_s3** 擴充功能

1. 使用 psql (或 pgAdmin) 以具有 rds_superuser 權限的使用者身分連接到 Aurora PostgreSQL 資料庫叢集的寫入器執行個體 。若您在安裝程序期間保留預設名稱，則連接為 postgres。

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. 若要安裝擴充功能，請執行下列命令。

```
postgres=> CREATE EXTENSION aws_s3 CASCADE;
NOTICE: installing required extension "aws_commons"
CREATE EXTENSION
```

3. 若要驗證是否已經安裝擴充功能，可以使用 psql \dx 中繼命令。

```
postgres=> \dx
       List of installed extensions
    Name     | Version |   Schema   |                Description
-------------+---------+------------+-------------------------------------------
 aws_commons | 1.2     | public     | Common data types across AWS services
 aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
 plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

現在可以使用從 Amazon S3 匯入和匯出資料的功能。

確認您的 Aurora PostgreSQL 版本支援匯出至 Amazon S3

您可以使用 describe-db-engine-versions 命令，驗證您的 Aurora PostgreSQL 版本是否支援匯出至 Amazon S3。下列範例會檢查 10.14 版本是否可以匯出至 Amazon S3。

```
aws rds describe-db-engine-versions --region us-east-1 \
--engine aurora-postgresql --engine-version 10.14 | grep s3Export
```

如果輸出包含字串 "s3Export"，則引擎支援 Amazon S3 匯出。否則，引擎不支援它們。

將資料匯出至 Amazon S3 的概觀

如果要將儲存在 Aurora PostgreSQL 資料庫中的資料匯出至 Amazon S3 儲存貯體，請使用以下程序。

將 Aurora PostgreSQL 資料匯出至 S3

1. 識別用於匯出資料的 Amazon S3 檔案路徑。如需此程序的詳細資訊，請參閱指定要匯出的 Amazon S3 檔案路徑。
2. 提供許可，以存取 Amazon S3 儲存貯體。

   如果要將資料匯出至 Amazon S3 檔案，請為 Aurora PostgreSQL 資料庫叢集提供許可，以便存取匯出將用來儲存的 Amazon S3 儲存貯體。這麼做包括以下步驟：

   1. 建立 IAM 政策來為您要匯出的 Amazon S3 儲存貯體提供存取權。
   2. 建立 IAM 角色。
   3. 請將您建立的政策連接到您建立的角色。
   4. 將此 IAM 角色新增至您的資料庫叢集 。

   如需此程序的詳細資訊，請參閱設定對 Amazon S3 儲存貯體的存取權。
3. 識別資料庫查詢以取得資料。呼叫 `aws_s3.query_export_to_s3` 函數來匯出查詢資料。

   完成上述的準備工作後，請使用 aws_s3.query_export_to_s3 函數將查詢結果匯出至 Amazon S3。如需此程序的詳細資訊，請參閱使用 aws_s3.query_export_to_s3 函數匯出查詢資料。

指定要匯出的 Amazon S3 檔案路徑

指定下列資訊來識別 Amazon S3 中您要匯出資料的位置：

- 儲存貯體名稱 – 儲存貯體是 Amazon S3 物件或檔案的容器。

  如需使用 Amazon S3 存放資料的詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的建立儲存貯體和使用物件。
- 檔案路徑 – 檔案路徑會識別匯出項目儲存在 Amazon S3 儲存貯體中的位置。檔案路徑由以下項目組成：
  - 識別虛擬資料夾路徑的選擇性路徑字首。
  - 識別一或多個要儲存檔案的檔案字首。較大的匯出項目會儲存在多個檔案中，每個檔案的大小上限約為 6 GB。其他檔案名稱具有相同的檔案字首，但會加上 `_part`*XX*。*XX* 代表 2，接著是 3，以此類推

  舉例來說，具有 `exports` 資料夾和 `query-1-export` 檔案字首的檔案路徑為 `/exports/query-1-export`。

- AWS 區域 （選用） – Amazon S3 儲存貯體所在的 AWS 區域。如果您未指定 AWS 區域值，則 Aurora 會將您的檔案儲存在與匯出資料庫叢集相同的 AWS 區域中的 Amazon S3。

  > **Note**
  >
  > 目前， AWS 區域必須與匯出資料庫叢集的區域相同。

  如需 AWS 區域名稱和相關值的清單，請參閱 區域和可用區域。

如果要保留匯出項目儲存之位置的 Amazon S3 檔案資訊，可以使用 aws_commons.create_s3_uri 函數建立 `aws_commons._s3_uri_1` 複合結構，如下所示。

```
psql=> SELECT aws_commons.create_s3_uri(
   'amzn-s3-demo-bucket',
   'sample-filepath',
   'us-west-2'
) AS s3_uri_1 \gset
```

您稍後可以在對 `s3_uri_1` 函數的呼叫中以參數形式提供此 aws_s3.query_export_to_s3 值。如需範例，請參閱「使用 aws_s3.query_export_to_s3 函數匯出查詢資料」。

設定對 Amazon S3 儲存貯體的存取權

如果要將資料匯出至 Amazon S3，請為 PostgreSQL DB 叢集提供許可，以便存取用來放置檔案的 Amazon S3 儲存貯體。

若要執行此操作，請使用下列程序。

透過 IAM 角色授與 PostgreSQL 資料庫叢集的 Amazon S3 存取權

1. 建立 IAM 政策。

   此政策可提供儲存貯體及物件許可，讓 PostgreSQL 資料庫叢集能夠存取 Amazon S3。

   在建立此原則的過程中，請採取下列步驟：

   a. 在政策中納入下列必要動作，以允許從 PostgreSQL 資料庫叢集叢集傳輸檔案至 Amazon S3 儲存貯體：

      - `s3:PutObject`

- s3:AbortMultipartUpload

b. 包含用於識別 Amazon S3 儲存貯體和物件的 Amazon Resource Name (ARN)。存取 Amazon S3 的 ARN 格式為：arn:aws:s3:::*amzn-s3-demo-bucket*/*

如需如何建立 Aurora PostgreSQL IAM 政策的詳細資訊，請參閱建立並使用 IAM 政策進行 IAM 資料庫存取。另請參閱《IAM 使用者指南》中的教學：建立和連接您的第一個客戶受管原則。

下列 AWS CLI 命令會使用這些選項建立名為 rds-s3-export-policy的 IAM 政策。其將存取權授予名為 *amzn-s3-demo-bucket* 的儲存貯體。

> **Warning**
>
> 建議您在設有可存取特定儲存貯體之端點原則的私有 VPC 中設定資料庫。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的對 Amazon S3 使用端點政策。
> 強烈建議您不要建立具有全資源存取權的政策。這個存取權可能會對資料安全構成威脅。如果您建立的原則能為 S3:PutObject 提供使用 "Resource":"*" 存取所有資源的存取權，則具有匯出許可的使用者可將資料匯出至您帳戶中的所有儲存貯體。此外，使用者可以將資料匯出至您 AWS 區域內可公開寫入的所有儲存貯體。

政策建立後，請記下政策的 Amazon Resource Name (ARN)。在後續步驟中將政策附加至 IAM 角色時，您會需要此 ARN。

```
aws iam create-policy  --policy-name rds-s3-export-policy  --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3export",
         "Action": [
           "s3:PutObject*",
           "s3:ListBucket",
           "s3:GetObject*",
           "s3:DeleteObject*",
           "s3:GetBucketLocation",
           "s3:AbortMultipartUpload"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::amzn-s3-demo-bucket/*"
```

```
        ]
      }
    ]
  }'
```

2. 建立 IAM 角色。

   您會執行此動作，使得 Aurora PostgreSQL 可以代表您擔任此 IAM 角色，以存取您的 Amazon S3 儲存貯體。如需更多詳細資訊，請參閱《IAM 使用者指南》中的建立角色以將許可委派給 IAM 使用者。

   建議您在資源型政策中使用 aws:SourceArn 和 aws:SourceAccount 全域條件內容金鑰，將服務的許可限定於特定資源。這是防止混淆代理人問題最有效的方式。

   如果同時使用這兩個全域條件內容索引鍵，且 aws:SourceArn 值包含帳戶 ID，則在相同政策陳述式中使用 aws:SourceAccount 值和 aws:SourceArn 值中的帳戶時，必須使用相同的帳戶 ID。

   - 如果您想要跨服務存取單一資源，請使用 aws:SourceArn。
   - 如果您想要允許該帳戶中的任何資源與跨服務使用相關聯，請使用 aws:SourceAccount。

   在政策中，請務必搭配資源的完整 ARN 來使用 aws:SourceArn 全域條件內容金鑰。下列範例示範如何使用 AWS CLI 命令來建立名為 的角色rds-s3-export-role。

   Example

   針對 Linux、macOS 或 Unix：

```
aws iam create-role  \
    --role-name rds-s3-export-role  \
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
```

```
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:dbname"
                }
            }
        }
      ]
   }'
```

在 Windows 中：

```
aws iam create-role  ^
    --role-name rds-s3-export-role  ^
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:dbname"
                }
            }
        }
      ]
   }'
```

3. 將您建立的 IAM 政策附加至您建立的 IAM 角色。

   下列 AWS CLI 命令會將先前建立的政策連接至名為 `rds-s3-export-role`. Replace 的角色，*your-policy-arn*取代為您在先前步驟中記下的政策 ARN。

```
aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
export-role
```

4. 將 IAM 角色新增至資料庫叢集。您可以使用 AWS 管理主控台 或 來執行此操作 AWS CLI，如下所述。

主控台

使用主控台為 PostgreSQL 資料庫叢集新增 IAM 角色

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 選擇 PostgreSQL 資料庫叢集名稱以顯示其詳細資訊。
3. 在 Connectivity & security (連線能力與安全性) 標籤上的 Manage IAM roles (管理 IAM 角色) 區段中，選擇要在 Add IAM roles to this instance (新增 IAM 角色到此執行個體) 下新增的角色。
4. 請在 Feature (功能) 下，選擇 s3Export。
5. 選擇 Add role (新增角色)。

AWS CLI

使用 CLI 為 PostgreSQL 資料庫叢集新增 IAM 角色

- 使用下列命令將角色新增至名為 `my-db-cluster` 的 PostgreSQL 資料庫叢集。將 *`your-role-arn`* 替換為您前個步驟記下的角色 ARN。使用 `s3Export` 作為 `--feature-name` 選項的值。

  Example

  針對 Linux、macOS 或 Unix：

```
aws rds add-role-to-db-cluster \
   --db-cluster-identifier my-db-cluster \
   --feature-name s3Export \
   --role-arn your-role-arn   \
   --region your-region
```

  在 Windows 中：

```
aws rds add-role-to-db-cluster ^
   --db-cluster-identifier my-db-cluster ^
   --feature-name s3Export ^
   --role-arn your-role-arn ^
```

```
    --region your-region
```

使用 aws_s3.query_export_to_s3 函數匯出查詢資料

呼叫 aws_s3.query_export_to_s3 函數來將 PostgreSQL 資料匯出至 Amazon S3。

主題

- 先決條件
- 正在呼叫 aws_s3.query_export_to_s3
- 匯出至使用自訂分隔符號的 CSV 檔案
- 利用編碼匯出至二進位檔案

先決條件

使用 aws_s3.query_export_to_s3 函數前，請先完成下列必要條件：

- 安裝需要的 PostgreSQL 延伸，如將資料匯出至 Amazon S3 的概觀 中所述。
- 決定要將資料匯出至 Amazon S3 的位置，如指定要匯出的 Amazon S3 檔案路徑中所述。
- 確認資料庫叢集如 設定對 Amazon S3 儲存貯體的存取權 中所述，有對 Amazon S3 的匯出存取權。

下列範例會使用稱為 sample_table 的資料庫資料表。這些範例會將資料匯出至名為 *amzn-s3-demo-bucket* 的儲存貯體。範例資料表和資料會以下列 psql 形式的 SQL 陳述式建立。

```
psql=> CREATE TABLE sample_table (bid bigint PRIMARY KEY, name varchar(80));
psql=> INSERT INTO sample_table (bid,name) VALUES (1, 'Monday'), (2,'Tuesday'), (3,
 'Wednesday');
```

正在呼叫 aws_s3.query_export_to_s3

以下示範呼叫 aws_s3.query_export_to_s3 函數的基本方式。

這些範例會使用變數 s3_uri_1 來識別包含識別 Amazon S3 檔案資訊的結構。使用 aws_commons.create_s3_uri 函數來建立結構。

```
psql=> SELECT aws_commons.create_s3_uri(
   'amzn-s3-demo-bucket',
   'sample-filepath',
```

```
   'us-west-2'
) AS s3_uri_1 \gset
```

雖然下列兩個 aws_s3.query_export_to_s3 函數呼叫的參數有所不同，但這些範例的結果是相同的。sample_table 資料表的所有列都會匯出至名為 *amzn-s3-demo-bucket* 的儲存貯體。

```
psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1');

psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1', options :='format text');
```

參數說明如下：

- 'SELECT * FROM sample_table' – 第一個字串是包含 SQL 查詢的文字字串。PostgreSQL 引擎會執行此查詢。查詢結果會複製到其他參數中識別的 S3 儲存貯體。
- :'s3_uri_1' – 此參數是識別 Amazon S3 檔案的結構。此範例使用變數來識別先前建立的結構。您可以改為透過在 aws_commons.create_s3_uri 函數呼叫中包含內嵌 aws_s3.query_export_to_s3 函數呼叫來建立結構，如下所示。

  ```
  SELECT * from aws_s3.query_export_to_s3('select * from sample_table',
     aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'sample-filepath', 'us-west-2')
  );
  ```

- options :='format text' – options 是包含 PostgreSQL COPY 引數的選用文字字串。複製程序使用 PostgreSQL COPY 命令的引數及格式。

如果指定的檔案不存在於 Amazon S3 儲存貯體內，就會建立。如果檔案已存在，即會遭到覆寫。以下是存取 Amazon S3 中已匯出資料的語法。

```
s3-region://bucket-name[/path-prefix]/file-prefix
```

較大的匯出項目會儲存在多個檔案中，每個檔案的大小上限約為 6 GB。其他檔案名稱具有相同的檔案字首，但會加上 _part*XX*。*XX* 代表 2，接著是 3，以此類推 舉例來說，假設您要指定儲存資料檔案的路徑，如下所示。

```
s3-us-west-2://amzn-s3-demo-bucket/my-prefix
```

如果匯出必須建立三個資料檔案，則 Amazon S3 儲存貯體會包含下列資料檔案。

```
s3-us-west-2://amzn-s3-demo-bucket/my-prefix
s3-us-west-2://amzn-s3-demo-bucket/my-prefix_part2
s3-us-west-2://amzn-s3-demo-bucket/my-prefix_part3
```

如需此函數的完整參考以及其他呼叫方式，請參閱 aws_s3.query_export_to_s3。如需存取 Amazon S3 中檔案的詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的檢視物件。

匯出至使用自訂分隔符號的 CSV 檔案

下列範例示範如何呼叫使用自訂分隔符號來將資料匯入檔案的 aws_s3.query_export_to_s3 函數。範例使用了 PostgreSQL COPY 命令的引數，來指定逗號分隔值 (CSV) 格式和冒號 (:) 分隔符號。

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format csv, delimiter $$:$$');
```

利用編碼匯出至二進位檔案

下列範例示範如何呼叫 aws_s3.query_export_to_s3 函數來將資料匯出至使用 Windows-1253 編碼的二進位檔案。

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format binary, encoding WIN1253');
```

函數參考

函數

- aws_s3.query_export_to_s3
- aws_commons.create_s3_uri

aws_s3.query_export_to_s3

將 PostgreSQL 查詢結果匯出至 Amazon S3 儲存貯體。aws_s3 延伸提供 aws_s3.query_export_to_s3 函數。

兩個必要參數為 query 及 s3_info。這些參數會定義要匯出的查詢，以及要匯出至的 Amazon S3 儲存貯體。名為 options 的選用參數會提供來定義各種匯出參數。如需使用 aws_s3.query_export_to_s3 函數的範例，請參閱 使用 aws_s3.query_export_to_s3 函數匯出查詢資料。

語法

```
aws_s3.query_export_to_s3(
    query text,
    s3_info aws_commons._s3_uri_1,
    options text,
    kms_key text
)
```

輸入參數

query

包含 PostgreSQL 引擎執行之 SQL 查詢的必要文字字串。此查詢的結果會複製到 s3_info 參數中識別的 S3 儲存貯體。

s3_info

aws_commons._s3_uri_1 複合類型，含有下列 S3 物件相關資訊：

- bucket – 包含檔案的 Amazon S3 儲存貯體名稱。
- file_path – Amazon S3 檔案名稱和路徑。
- region – 儲存貯體所在的 AWS 區域。如需 AWS 區域名稱和相關值的清單，請參閱 區域和可用區域。

  目前，此值必須與匯出資料庫叢集 AWS 的區域相同。預設值為匯出資料庫叢集 AWS 的區域。

如果要建立 aws_commons._s3_uri_1 複合結構，請參閱 aws_commons.create_s3_uri 函數。

options

選用的文字字串，含有 PostgreSQL COPY 命令引數。這些引數指定資料要如何在匯出時複製。詳細資訊請參閱 PostgreSQL COPY 文件。

kms_key 文字

選用的文字字串，其中包含要將資料匯出至其中的 S3 儲存貯體之客戶受管 KMS 金鑰。

替代輸入參數

為了協助進行測試，您可使用一組更大的參數取代 s3_info 參數。以下是 aws_s3.query_export_to_s3 函數的其他語法變化。

請不要使用 s3_info 參數識別 Amazon S3 檔案，而是使用 bucket、file_path 及 region 參數組合進行。

```
aws_s3.query_export_to_s3(
    query text,
    bucket text,
    file_path text,
    region text,
    options text,
    kms_key text
)
```

query

包含 PostgreSQL 引擎執行之 SQL 查詢的必要文字字串。此查詢的結果會複製到 s3_info 參數中識別的 S3 儲存貯體。

bucket

必要文字字串，其中含有包含檔案的 Amazon S3 儲存貯體名稱。

file_path

包含 Amazon S3 檔案名稱 (包括檔案路徑) 的必要文字字串。

region

選用的文字字串，其中包含儲存貯體所在的 AWS 區域。如需 AWS 區域名稱和相關值的清單，請參閱 區域和可用區域。

目前，此值必須與匯出資料庫叢集 AWS 的區域相同。預設值為匯出資料庫叢集 AWS 的區域。

options

選用的文字字串，含有 PostgreSQL COPY 命令引數。這些引數指定資料要如何在匯出時複製。詳細資訊請參閱 PostgreSQL COPY 文件。

kms_key 文字

選用的文字字串，其中包含要將資料匯出至其中的 S3 儲存貯體之客戶受管 KMS 金鑰。

輸出參數

```
aws_s3.query_export_to_s3(
```

```
    OUT rows_uploaded bigint,
    OUT files_uploaded bigint,
    OUT bytes_uploaded bigint
)
```

rows_uploaded

指定查詢成功上傳至 Amazon S3 的資料表列數。

files_uploaded

上傳至 Amazon S3 的檔案數。建立的檔案大小約為 6 GB。每個額外建立的檔案，名稱都會加上 _part*XX*。*XX* 代表 2，接著是 3，視需要以此類推。

bytes_uploaded

上傳至 Amazon S3 的總位元組數。

範例

```
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-demo-bucket', 'sample-filepath');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-demo-bucket', 'sample-filepath','us-west-2');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-demo-bucket', 'sample-filepath','us-west-2','format text');
```

aws_commons.create_s3_uri

建立 aws_commons._s3_uri_1 結構以保留 Amazon S3 檔案資訊。您使用 aws_commons.create_s3_uri 函數 s3_info 參數之中的 aws_s3.query_export_to_s3 函數結果。如需使用 aws_commons.create_s3_uri 函數的範例，請參閱 指定要匯出的 Amazon S3 檔案路徑。

語法

```
aws_commons.create_s3_uri(
   bucket text,
   file_path text,
   region text
)
```

輸入參數

bucket

必要的文字字串，其中含有檔案的 Amazon S3 儲存貯體名稱。

file_path

包含 Amazon S3 檔案名稱 (包括檔案路徑) 的必要文字字串。

region

必要的文字字串，其中包含檔案所在的 AWS 區域。如需 AWS 區域名稱和相關值的清單，請參閱區域和可用區域。

對 Amazon S3 的存取進行故障診斷

如果您在嘗試將資料匯出至 Amazon S3 時遇到連線問題，請先確認與資料庫執行個體關聯之 VPC 安全群組的輸出存取規則允許網路連線。明確的說，安全群組必須擁有允許資料庫執行個體傳至連接埠 443 和任何 IPv4 地址 (0.0.0.0/0)。如需更多詳細資訊，請參閱 建立安全群組以存取 VPC 中的資料庫叢集。

另請參閱以下內容中的建議：

- 對 Amazon Aurora 身分與存取進行故障診斷
- 《Amazon Simple Storage Service 使用者指南》中的針對 Amazon S3 進行故障診斷。
- 《IAM 使用者指南》中的針對 Amazon S3 和 IAM 進行故障診斷

從 Aurora PostgreSQL 資料庫叢集 叫用 AWS Lambda 函數

AWS Lambda 是一種事件驅動的運算服務，可讓您執行程式碼，而無需佈建或管理伺服器。它可用於許多 AWS 服務，包括 Aurora PostgreSQL。例如，您可以使用 Lambda 函數處理來自資料庫的事件通知，或在新檔案上傳到 Simple Storage Service (Amazon S3) 時從檔案中載入資料。若要進一步了解 Lambda，請參閱《 開發人員指南》中的什麼是 AWS Lambda？。 AWS Lambda

> **Note**
>
> Aurora PostgreSQL 11.9 及更高版本 （ 包括 Aurora Serverless v2) 支援叫用 AWS Lambda 函數。

設定 Aurora PostgreSQL 以使用 Lambda 函數是涉及 IAM AWS Lambda、VPC 和 Aurora PostgreSQL 資料庫叢集的多步驟程序。以下提供必要步驟的摘要。

如需有關 Lambda 函數的詳細資訊，請參閱《 AWS Lambda 開發人員指南》中的 Lambda 入門和 AWS Lambda 函數。

主題

- 步驟 1：為您的 Aurora PostgreSQL 資料庫叢集 設定傳出連線 AWS Lambda
- 步驟 2：為您的 Aurora PostgreSQL 資料庫叢集 和 設定 IAM AWS Lambda
- 步驟 3：安裝 Aurora PostgreSQL 資料庫叢集 的aws_lambda擴充功能
- 步驟 4：搭配 Aurora PostgreSQL 資料庫叢集使用 Lambda helper 函數 (選用)
- 步驟 5：從 Aurora PostgreSQL 資料庫叢集叫用 Lambda 函數。
- 步驟 6：授予其他使用者呼叫 Lambda 函數的許可權限
- 範例：從 Aurora PostgreSQL 資料庫叢集叫用 Lambda 函數
- Lambda 函數錯誤訊息
- AWS Lambda 函數和參數參考

步驟 1：為您的 Aurora PostgreSQL 資料庫叢集 設定傳出連線 AWS Lambda

Lambda 函數一律會在 AWS Lambda 服務擁有的 Amazon VPC 內執行。Lambda 會將網路存取和安全性規則套用至此 VPC，並自動維護和監控 VPC。您的 Aurora PostgreSQL 資料庫叢集 會將網路流量傳送到 Lambda 服務的 VPC。具體設定取決於 Aurora 資料庫叢集的主要資料庫執行個體是公有的或私有。

- 公有 Aurora PostgreSQL 資料庫叢集 – 如果執行個體位於公有子網路中，且執行個體的「PubliclyAccessible able」屬性為 `true`，則資料庫叢集的主要資料庫執行個體為公有的。若要尋找此屬性的值，您可以使用 describe-db-instances AWS CLI 命令。或者可以使用 AWS 管理主控台開啟 Connectivity & security (連線與安全性) 索引標籤，檢查 Publicly accessible (可公開存取) 是否為 Yes (是)。若要驗證執行個體是否在您的 VPC 公有子網路中，您可以使用 AWS 管理主控台 或 AWS CLI。

  若要設定 Lambda 的存取權，您可以使用 AWS 管理主控台 或 AWS CLI 在您的 VPC 安全群組上建立傳出規則。傳出規則會指定 TCP 可以使用連接埠 443 將封包傳至任何 IPv4 位址 (0.0.0.0/0)。
- 私有 Aurora PostgreSQL 資料庫叢集 – 在此情況下，執行個體的「PubliclyAccessible」屬性為 `false`，或者位於私有子網路中。若要允許執行個體使用 Lambda，您可以使用網路位址轉譯 (NAT) 閘道。如需更多詳細資訊，請參閱 NAT 閘道。或者，您可以使用 Lambda 的 VPC 端點來

設定 VPC。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 VPC 端點。端點將 Aurora PostgreSQL 資料庫叢集 所發出呼叫的回傳至 Lambda 函數。

您的 VPC 現在可以在網路層級與 AWS Lambda VPC 互動。下一步，您需要使用 IAM 來設定許可。

步驟 2：為您的 Aurora PostgreSQL 資料庫叢集 和 設定 IAM AWS Lambda

從 Aurora PostgreSQL 資料庫叢集叫用 Lambda 函數需要某些權限。若要設定必要的權限，建議您建立允許叫用 Lambda 函數的 IAM 政策，將政策指派給一個角色，然後將該角色套用至資料庫叢集。此做法會提供資料庫叢集權限，允許代表您叫用指定的 Lambda 函數。下列步驟說明如何在 AWS CLI中執行此操作。

設定 IAM 許可以搭配 Lambda 使用叢集

1. 使用 create-policy AWS CLI 命令建立 IAM 政策，允許 Aurora PostgreSQL 資料庫叢集叫用指定的 Lambda 函數。(陳述式 ID (Sid) 是政策陳述式的選用描述，不會影響使用。) 此政策為 Aurora 資料庫叢集提供叫用指定 Lambda 函數所需的最低許可。

```
aws iam create-policy  --policy-name rds-lambda-policy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "AllowAccessToExampleFunction",
        "Effect": "Allow",
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"
        }
    ]
}'
```

   或者，您可以使用預先定義的 AWSLambdaRole 政策，該政策允許叫用任何 Lambda 函數。如需詳細資訊，請參閱適用於 Lambda 的身分型 IAM 政策。

2. 使用 create-role AWS CLI 命令來建立政策可在執行時間擔任的 IAM 角色。

```
aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
```

```
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}'
```

3. 使用 attach-role-policy AWS CLI 命令將政策套用至角色。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \
    --role-name rds-lambda-role --region aws-region
```

4. 使用 add-role-to-db-cluster AWS CLI 命令，將角色套用至 Aurora PostgreSQL 資料庫叢集 。最後這個步驟允許資料庫叢集的資料庫使用者叫用 Lambda 函數。

```
aws rds add-role-to-db-cluster \
       --db-cluster-identifier my-cluster-name \
       --feature-name Lambda \
       --role-arn  arn:aws:iam::444455556666:role/rds-lambda-role   \
       --region aws-region
```

完成 VPC 和 IAM 設定後，現在可以安裝 aws_lambda 擴充功能。(請注意，您可以隨時安裝擴充功能，但在設定正確的 VPC 支援和 IAM 權限之前，aws_lambda 擴充功能不會為 Aurora PostgreSQL 資料庫叢集的功能新增任何項目。)

步驟 3：安裝 Aurora PostgreSQL 資料庫叢集 的**aws_lambda**擴充功能

若要 AWS Lambda 搭配 Aurora PostgreSQL 資料庫叢集使用 ，請將 aws_lambda PostgreSQL 延伸模組新增至 Aurora PostgreSQL 資料庫叢集。此擴充功能讓 Aurora PostgreSQL 資料庫叢集能夠從 PostgreSQL 呼叫 Lambda 函數。

在 Aurora PostgreSQL 資料庫叢集 中安裝**aws_lambda**擴充功能

使用 PostgreSQL psql 命令列或 pgAdmin 工具連線到Aurora PostgreSQL 資料庫叢集 。

1. 以具有 rds_superuser 權限的使用者身分連線至 Aurora PostgreSQL 資料庫叢集 執行個體。預設 postgres 使用者顯示於範例中。

```
psql -h cluster-instance.444455556666.aws-region.rds.amazonaws.com -U postgres -p
 5432
```

2. 安裝 aws_lambda 擴充功能。另外也需要 aws_commons 擴充功能。它為 aws_lambda 提供了 helper 函數和 PostgreSQL 的許多其他 Aurora 擴充功能。如果尚未裝在 Aurora PostgreSQL 資料庫叢集 上，會透過 aws_lambda 進行安裝，如下所示。

```
CREATE EXTENSION IF NOT EXISTS aws_lambda CASCADE;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
```

aws_lambda 擴充功能已安裝在 Aurora PostgreSQL 資料庫叢集的主要資料庫執行個體上。您現在可以建立便利的結構，用於叫用 Lambda 函數。

步驟 4：搭配 Aurora PostgreSQL 資料庫叢集使用 Lambda helper 函數 (選用)

您可以在 aws_commons 擴充功能中使用 helper 函數，準備可更輕鬆從 PostgreSQL 叫用的實體。若要執行此操作，您需要以下有關 Lambda 函數的資訊：

- 函數名稱 – Lambda 函數的名稱、Amazon 資源名稱 (ARN)、版本或別名。在 步驟 2：為叢集 和 Lambda 設定 IAM 中建立的 IAM 政策需要 ARN，因此建議您使用函數的 ARN。
- AWS 區域 – （選用） 如果 AWS Lambda 函數不在與 Aurora PostgreSQL 資料庫叢集相同的區域中，則 Lambda 函數所在的區域。

若要保存 Lambda 函數名稱資訊，可使用 aws_commons.create_lambda_function_arn 函數。此 helper 函數會建立一個 aws_commons._lambda_function_arn_1 複合結構，其中包含叫用函數所需的詳細資訊。接下來說明設定此複合結構的三種替代做法。

```
SELECT aws_commons.create_lambda_function_arn(
   'my-function',
   'aws-region'
) AS aws_lambda_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
   '111122223333:function:my-function',
   'aws-region'
) AS lambda_partial_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
   'arn:aws:lambda:aws-region:111122223333:function:my-function'
) AS lambda_arn_1 \gset
```

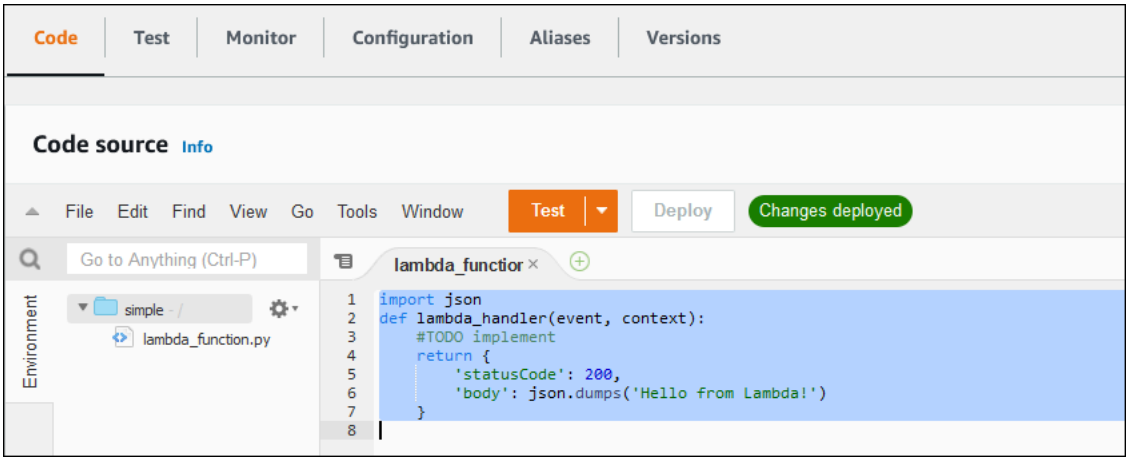
這些值全部都可以用於 aws_lambda.invoke 函數呼叫。如需範例，請參閱 步驟 5：從 Aurora PostgreSQL 資料庫叢集叫用 Lambda 函數。。

步驟 5：從 Aurora PostgreSQL 資料庫叢集叫用 Lambda 函數。

aws_lambda.invoke 函數採同步或異步行為，具體取決於 invocation_type。此參數的兩個替代項目是 RequestResponse (預設值) 和 Event，如下所示：

- **RequestResponse** – 此叫用類型為同步。這是在未指定叫用類型的情況下進行呼叫時的預設行為。回應承載包括 aws_lambda.invoke 函數的結果。如果您的工作流程需要接收 Lambda 函數的結果才能繼續執行，請使用此叫用類型。
- **Event** – 此叫用類型為非同步。回應不包括含有結果的承載。如果您的工作流程不需要 Lambda 函數的結果即可繼續執行，請使用此叫用類型。

如要簡單測試您的設定，可以使用 psql 連線至資料庫執行個體，並從命令列叫用範例函數。假設您在 Lambda 服務上設定了一個基本函數，例如下方螢幕擷取畫面中顯示的簡單 Python 函數。



叫用範例函數

1. 使用 psql 或 pgAdmin 連線至主要資料庫執行個體。

   ```
   psql -h cluster.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
   ```

2. 使用函數的 ARN 叫用函數。

```
SELECT * from
 aws_lambda.invoke(aws_commons.create_lambda_function_arn('arn:aws:lambda:aws-
region:444455556666:function:simple', 'us-west-1'), '{"body": "Hello from
 Postgres!"}'::json );
```

回應如下所示。

```
status_code |                          payload                          |
 executed_version | log_result
-------------+-----------------------------------------------------------
+------------------+------------
         200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST
     |
(1 row)
```

如果您的叫用嘗試未成功，請參閱 Lambda 函數錯誤訊息。

步驟 6：授予其他使用者呼叫 Lambda 函數的許可權限

在程序中的這一點上，只有身為 rds_superuser 的您可以叫用 Lambda 函式。如要允許其他使用者呼叫您建立的任何函數，您需要授予其許可權限。

如要授予叫用 Lambda 函數的許可權限

1. 使用 psql 或 pgAdmin 連線至主要資料庫執行個體。

```
psql -h cluster.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
```

2. 執行下列 SQL 命令：

```
postgres=>  GRANT USAGE ON SCHEMA aws_lambda TO db_username;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA aws_lambda TO db_username;
```

範例：從 Aurora PostgreSQL 資料庫叢集叫用 Lambda 函數

以下提供幾個呼叫 aws_lambda.invoke 函數的範例。大部分範例使用您在 步驟 4：搭配 Aurora PostgreSQL 資料庫叢集使用 Lambda helper 函數 (選用) 中建立的複合結構 aws_lambda_arn_1 來

簡化函數詳細資訊的傳遞。如需非同步叫用的範例，請參閱範例：Lambda 函數的非同步 (Event) 叫用。列出的所有其他範例都使用同步叫用。

若要進一步了解 Lambda 叫用類型，請參閱《AWS Lambda 開發人員指南》中的叫用 Lambda 函數。如需 aws_lambda_arn_1 的相關資訊，請參閱 aws_commons.create_lambda_function_arn。

範例清單

- 範例：Lambda 函數的同步 (RequestResponse) 叫用
- 範例：Lambda 函數的非同步 (Event) 叫用
- 範例：在函數回應中擷取 Lambda 執行日誌
- 範例：在 Lambda 函數中包含用戶端內容
- 範例：叫用特定版本的 Lambda 函數

範例：Lambda 函數的同步 (RequestResponse) 叫用

下面是同步 Lambda 函數叫用的兩個範例。這些 aws_lambda.invoke 函數的結果相同。

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
```

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse');
```

參數說明如下：

- :'aws_lambda_arn_1' – 此參數識別 步驟 4：搭配 Aurora PostgreSQL 資料庫叢集使用 Lambda helper 函數 (選用) 中使用 aws_commons.create_lambda_function_arn helper 函數建立的複合結構。您也可以在 aws_lambda.invoke 呼叫中建立這個結構，如下所示。

  ```
  SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-function',
   'aws-region'),
  '{"body": "Hello from Postgres!"}'::json
  );
  ```

- '{"body": "Hello from PostgreSQL!"}'::json – 要傳遞給 Lambda 函數的 JSON 承載。
- 'RequestResponse' – Lambda 叫用類型。

範例：Lambda 函數的非同步 (Event) 叫用

以下是非同步 Lambda 函數叫用的範例。Event 叫用類型會使用指定的輸入承載，來排程 Lambda 函數叫用並立即傳回。在某些工作流程中，使用不依賴於 Lambda 函數結果的 Event 叫用類型。

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'Event');
```

範例：在函數回應中擷取 Lambda 執行日誌

在 aws_lambda.invoke 函數呼叫中使用 log_type 參數，即可讓函數回應中包含執行日誌的最後 4 KB。此參數預設為 None，但您可指定 Tail 以在回應中擷取 Lambda 執行日誌的結果，如下所示。

```
SELECT *, select convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

將 aws_lambda.invoke log_type 函數的 Tail 參數設定為在回應中包含執行日誌。log_type 參數的預設值為 None。

log_result 傳回的是 base64 編碼字串。您可以使用 decode 和 convert_from PostgreSQL 函數的組合，來解碼內容。

如需 log_type 的相關資訊，請參閱 aws_lambda.invoke。

範例：在 Lambda 函數中包含用戶端內容

aws_lambda.invoke 函數具有 context 參數，可用來獨立於承載之外傳遞資訊，如下所示。

```
SELECT *, convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

若要包含用戶端內容，請使用 JSON 物件作為 aws_lambda.invoke 函數的 context 參數。

如需 context 參數的詳細資訊，請參閱 aws_lambda.invoke 參考。

範例：叫用特定版本的 Lambda 函數

使用 aws_lambda.invoke 呼叫加入 qualifier 參數，即可指定特定版本的 Lambda 函數。以下提供使用 '*custom_version*' 作為版本別名的範例。

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse', 'None', NULL, 'custom_version');
```

也可改為提供含有函數名稱的 Lambda 函數限定詞，如下所示。

```
SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function:custom_version', 'us-west-2'),
'{"body": "Hello from Postgres!"}'::json);
```

如需 `qualifier` 和其他參數的詳細資訊，請參閱 aws_lambda.invoke 參考。

Lambda 函數錯誤訊息

於下列清單中，您可找到有關錯誤訊息的資訊，及可能的原因和解決方案。

- VPC 組態問題

  嘗試連線時，VPC 組態問題可能會引發下列錯誤訊息：

  ```
  ERROR:  invoke API failed
  DETAIL: AWS Lambda client returned 'Unable to connect to endpoint'.
  CONTEXT:  SQL function "invoke" statement 1
  ```

  此錯誤的常見原因是未正確設定 VPC 安全群組。務必在您的 VPC 安全群組連接埠 443 上開啟 TCP 的傳出規則，讓 VPC 可連線至 Lambda VPC。
- 缺乏叫用 Lambda 函式所需的許可權限

  若您看到下列其中一個錯誤訊息，則叫用該函數的使用者 (角色) 並無適當的許可權限。

  ```
  ERROR:  permission denied for schema aws_lambda
  ```

  ```
  ERROR:  permission denied for function invoke
  ```

  使用者 (角色) 必須取得特定授權才可叫用 Lambda 函數 如需詳細資訊，請參閱步驟 6：授予其他使用者呼叫 Lambda 函數的許可權限。
- 不正確處理 Lambda 函數中的錯誤

  如果 Lambda 函數在請求處理期間拋出異常，則 `aws_lambda.invoke` 會失敗並顯示如下所示 PostgreSQL 錯誤。

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
ERROR:  lambda invocation failed
DETAIL:  "arn:aws:lambda:us-west-2:555555555555:function:my-function" returned error
 "Unhandled", details: "<Error details string>".
```

請務必處理 Lambda 函數或 PostgreSQL 應用程式中的錯誤。

AWS Lambda 函數和參數參考

以下是函數用於以 Aurora PostgreSQL 調用 Lambda 之函數和參數的參考。

函數和參數

- aws_lambda.invoke
- aws_commons.create_lambda_function_arn
- aws_lambda 參數

aws_lambda.invoke

為 Aurora PostgreSQL 資料庫叢集 執行 Lambda 函數。

如需有關叫用 Lambda 函數的詳細資訊，請參閱 AWS Lambda 開發人員指南中的叫用。

語法

JSON

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSON,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSON,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

JSONB

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSONB,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSONB,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT
)
```

輸入參數

function_name

識別 Lambda 函數的名稱。該值可以是函數名稱、ARN 或部分 ARN。如需適用格式的清單，請參閱 AWS Lambda 開發人員指南中的 Lambda 函數名稱格式。

payload

Lambda 函數的輸入。格式可以是 JSON 或 JSONB。如需詳細資訊，請參閱 PostgreSQL 文件中的 JSON 類型。

區域

(選用) 函數的 Lambda 區域。根據預設，Aurora 會從 AWS 中的完整 ARN 解析 `function_name` 區域，或使用 Aurora PostgreSQL 資料庫執行個體區域。如果此區域值與 `function_name` ARN 中提供的值衝突，則會引發錯誤。

invocation_type

Lambda 函數的叫用類型。 值會區分大小寫。可能的值包括以下：

- `RequestResponse` – 預設值。Lambda 函數的這種叫用類型是同步的，並在結果中傳回回應承載。若您的工作流程依賴於立即接收 Lambda 函數，請使用 `RequestResponse` 叫用類型。
- `Event` – Lambda 函數這種叫用類型是非同步的，並且在立即傳回時不含承載。若您在工作流程繼續進行之前不需要 Lambda 函數的結果，請使用 `Event` 叫用類型。
- `DryRun` – 這種類型的叫用會測試存取而不執行該 Lambda 函數。

log_type

`log_result` 輸出參數中要傳回的 Lambda 日誌類型。 值會區分大小寫。可能的值包括以下：

- 結尾 – 傳回的 `log_result` 輸出參數會包含執行日誌的最後 4 KB。
- 無 – 沒有傳回 Lambda 日誌資訊。

context

JSON 或 JSONB 格式的用戶端內容。要使用的欄位包括 `custom` 和 `env`。

限定詞

識別待叫用 Lambda 函數版本的限定詞。如果該值與 `function_name` ARN 中提供的值衝突，則會引發錯誤。

輸出參數

status_code

HTTP 狀態回應代碼。如需更多詳細資訊，請參閱 AWS Lambda 開發人員指南中的 Lambda 叫用回應元素。

payload

從執行的 Lambda 函數傳回的資訊。格式是 JSON 或 JSONB。

executed_version

Lambda 函數執行的版本。

log_result

如果 `log_type` 值是叫用 Lambda 函數時的 `Tail`，則會傳回執行日誌資訊。結果包含以 Base64 編碼的執行日誌的最後 4 KB。

aws_commons.create_lambda_function_arn

建立 `aws_commons._lambda_function_arn_1` 結構來保存 Lambda 函數名稱資訊。您可以在 aws_lambda.invoke `aws_commons.create_lambda_function_arn` 函數的 `function_name` 參數中，使用 aws_lambda.invoke 函數的結果。

語法

```
aws_commons.create_lambda_function_arn(
    function_name TEXT,
    region TEXT DEFAULT NULL
    )
    RETURNS aws_commons._lambda_function_arn_1
```

輸入參數

function_name

包含 Lambda 函數名稱的必要文字字串。該值可以是函數名稱、部分 ARN 或完整 ARN。

區域

選用文字字串，其中含有 Lambda 函數所在的 AWS 區域。如需區域名稱和相關聯值的清單，請參閱區域和可用區域。

aws_lambda 參數

在資料表中，您可以尋找與 aws_lambda 函數相關聯的參數。

參數	描述
aws_lambda.connect_timeout_ms	這是動態參數，其會設定連線到 AWS Lambda 時的最長等待時間。預設值為 1000。此參數的允許值為 1 - 900000。
aws_lambda.request_timeout_ms	這是動態參數，其會設定在等待 AWS Lambda 回應時的最長等待時間。預設值為 3000。此參數的允許值為 1 - 900000。
aws_lambda.endpoint_override	指定可用來連線至 AWS Lambda 的端點。空字串會選取區域的預設 AWS Lambda 端點。您必須重新啟動資料庫，此靜態參數變更才能產生作用。

將 Aurora PostgreSQL 日誌發佈至 Amazon CloudWatch Logs

您可以設定 Aurora PostgreSQL 資料庫叢集，將日誌資料定期匯出至 Amazon CloudWatch Logs。當您這麼做時，Aurora PostgreSQL 資料庫叢集的 PostgreSQL 日誌中的事件會以 Amazon CloudWatch Logs 的形式自動發佈至 Amazon CloudWatch。在 CloudWatch 中，您可以在日誌群組中找到 Aurora PostgreSQL 資料庫叢集的匯出日誌資料。日誌群組包含一或多個日誌串流，其中包含來自叢集中每個執行個體 PostgreSQL 日誌的事件。

將日誌發佈到 CloudWatch Logs 可讓您將叢集的 PostgreSQL 日誌記錄保留在高度耐用的儲存裝置中。使用 CloudWatch Logs 中可用的日誌資料，您可以評估並改善叢集的操作。您還可以使用 CloudWatch 建立警示和檢視指標。如需詳細資訊，請參閱 在 Amazon CloudWatch 中監控日誌事件。

> **Note**
>
> 將 PostgreSQL 日誌發佈到 CloudWatch Logs 會消耗儲存空間，而且您需要支付該儲存空間的費用。請務必刪除您不再需要的任何 CloudWatch Logs。

關閉現有 Aurora PostgreSQL 資料庫叢集的匯出日誌選項不會影響任何已存放在 CloudWatch Logs 中的資料。根據您的日誌保留設定，現有的日誌在 CloudWatch Logs 中仍然可以使用。如需 CloudWatch Logs 的詳細資訊，請參閱什麼是 Amazon CloudWatch Logs？

Aurora PostgreSQL 支援將日誌發佈至下列版本的 CloudWatch Logs：

- 16.1 版和所有更新版本
- 15.2 版和更新的 15 版本
- 14.3 版和更新的 14 版本
- 13.3 和更新的 13 版本
- 12.8 版和更新的 12 版本
- 11.9 和更新的 11 版本

如需詳細資訊，了解如何開啟將日誌發佈至 CloudWatch Logs 的選項、在 CloudWatch Logs 中監控日誌事件，以及使用 CloudWatch Logs Insights 分析日誌，請參閱下列主題。

主題

- 開啟將日誌發佈到 Amazon CloudWatch 的選項
- 在 Amazon CloudWatch 中監控日誌事件
- 使用 CloudWatch Logs Insights 分析 PostgreSQL 日誌

開啟將日誌發佈到 Amazon CloudWatch 的選項

若要將 Aurora PostgreSQL 資料庫叢集的 PostgreSQL 日誌發佈到 CloudWatch Logs，請選擇叢集的 Log export (日誌匯出) 選項。您可以在建立 Aurora PostgreSQL 資料庫叢集時選擇日誌匯出設定。或者，您可以稍後修改叢集。當您修改現有叢集時，每個執行個體的 PostgreSQL 日誌會從該時間點開始發佈到 CloudWatch 叢集。對於 Aurora PostgreSQL，PostgreSQL 日誌 (`postgresql.log`) 是唯一發佈到 Amazon CloudWatch 的日誌。

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 開啟 Aurora PostgreSQL 資料庫叢集的日誌匯出功能。

主控台

您可以選擇日誌匯出選項，開始將 PostgreSQL 日誌從您的 Aurora PostgreSQL 資料庫叢集發佈到 CloudWatch Logs。

從主控台開啟日誌匯出功能

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。

2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您要發佈日誌資料到 CloudWatch Logs 的 Aurora PostgreSQL 資料庫叢集。
4. 選擇 Modify (修改)。
5. 在 Log exports (日誌匯出) 區段中，選擇 Postgresql log (Postgresql 日誌)。
6. 選擇 Continue (繼續)，然後在摘要頁面上選擇 Modify cluster (修改叢集)。

AWS CLI

您可以開啟日誌匯出選項，開始使用 AWS CLI 將 Aurora PostgreSQL 日誌發佈到 Amazon CloudWatch Logs。若要執行此動作，您可以搭配下列選項執行 modify-db-cluster AWS CLI 命令：

- `--db-cluster-identifier`—資料庫叢集識別符
- `--cloudwatch-logs-export-configuration`—要設定的日誌類型的組態設定，適用於匯出到資料庫叢集的 CloudWatch Logs。

您也可以執行下列其中一個 AWS CLI 命令來發佈 Aurora PostgreSQL 日誌：

- create-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

執行上述其中一個 AWS CLI 命令並指定下列選項：

- `--db-cluster-identifier`—資料庫叢集識別符
- `--engine` — 資料庫引擎。
- `--enable-cloudwatch-logs-exports` — 資料庫叢集的哪些日誌類型要匯出至 CloudWatch Logs 的組態設定。

視您執行的 AWS CLI 命令而定，可能需要其他選項。

Example

下列命令建立 Aurora PostgreSQL 資料庫叢集，以將日誌檔案發佈至 CloudWatch Logs。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --engine aurora-postgresql \
    --enable-cloudwatch-logs-exports postgresql
```

在 Windows 中：

```
aws rds create-db-cluster ^
    --db-cluster-identifier my-db-cluster ^
    --engine aurora-postgresql ^
    --enable-cloudwatch-logs-exports postgresql
```

Example

下列命令修改現有的 Aurora PostgreSQL 資料庫叢集，以將日誌檔案發佈至 CloudWatch Logs。`--cloudwatch-logs-export-configuration` 值為 JSON 物件。這個物件的金鑰是 `EnableLogTypes`，而其值為 `postgresql` 和 `instance`。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["postgresql","instance"]}'
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier my-db-cluster ^
    --cloudwatch-logs-export-configuration '{\"EnableLogTypes\":[\"postgresql\",
\"instance\"]}'
```

> **Note**
>
> 使用 Windows 命令提示字元時，務必在 JSON 程式碼中的雙引號 (") 開頭加上反斜線 (\)，以逸出雙引號。

Example

下列範例會修改現有的 Aurora PostgreSQL 資料庫執行個體叢集，以停用將日誌檔案發佈至 CloudWatch Logs。`--cloudwatch-logs-export-configuration` 值為 JSON 物件。這個物件的金鑰是 `DisableLogTypes`，而其值為 `postgresql` 和 `instance`。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbinstance \
    --cloudwatch-logs-export-configuration '{"DisableLogTypes":
["postgresql","instance"]}'
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration "{\"DisableLogTypes\":[\"postgresql\",
\"instance\"]}"
```

> **Note**
>
> 使用 Windows 命令提示字元時，您必須在 JSON 程式碼中的雙引號 (") 開頭加上反斜線 (\)，以逸出雙引號。

RDS API

您可以開啟日誌匯出選項，以開始使用 RDS API 來發佈 Aurora PostgreSQL 日誌。若要執行此動作，您可以執行 ModifyDBCluster 操作並指定下列選項：

- `DBClusterIdentifier` – 資料庫叢集識別符。
- `CloudwatchLogsExportConfiguration` – 要啟用的日誌類型的組態設定，適用於匯出到資料庫叢集的 CloudWatch Logs。

您也可以執行下列其中一個 RDS API 操作，以利用 RDS API 來發佈 Aurora PostgreSQL 日誌：

- CreateDBCluster
- RestoreDBClusterFromS3

- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime

執行 RDS API 動作並指定下列參數：

- `DBClusterIdentifier`—資料庫叢集識別符
- `Engine` — 資料庫引擎。
- `EnableCloudwatchLogsExports` — 資料庫叢集的哪些日誌類型要匯出至 CloudWatch Logs 的組態設定。

視您執行的 AWS CLI 命令而定，可能需要其他參數。

在 Amazon CloudWatch 中監控日誌事件

使用 Aurora PostgreSQL 日誌事件發佈並以 Amazon CloudWatch Logs 的形式提供，您可以使用 Amazon CloudWatch 來檢視和監控事件。如需監控的相關詳細資訊，請參閱檢視傳送至 CloudWatch Logs 的日誌資料。

當您開啟日誌匯出時，會使用具有 Aurora PostgreSQL 名稱和日誌類型的前綴 `/aws/rds/cluster/` 自動建立新的日誌群組。

```
/aws/rds/cluster/your-cluster-name/postgresql
```

例如，假設名為 `docs-lab-apg-small` 的 Aurora PostgreSQL 資料庫叢集將日誌匯出至 Amazon CloudWatch Logs。它在 Amazon CloudWatch 的日誌群組名稱如下所示。

```
/aws/rds/cluster/docs-lab-apg-small/postgresql
```

如果存在指定名稱的日誌群組，Aurora 會使用該日誌群組來匯出 Aurora 資料庫叢集的日誌資料。Aurora PostgreSQL 資料庫叢集中的每個資料庫執行個體將其 PostgreSQL 日誌上傳到日誌群組，做為不同的日誌串流。您可以使用 Amazon CloudWatch 中提供的各種圖形和分析工具來檢查日誌群組及其日誌串流。

例如，您可以從 Aurora PostgreSQL 資料庫叢集搜尋日誌事件內的資訊，並使用 CloudWatch Logs 主控台 AWS CLI、 或 CloudWatch Logs API 篩選事件。如需詳細資訊，請參閱《Amazon CloudWatch Logs 使用者指南》中的搜尋和篩選日誌資料。

新日誌群組建立時預設會使用永不過期作為其保留期間。您可以使用 CloudWatch Logs 主控台 AWS CLI、 或 CloudWatch Logs API 來變更日誌保留期間。如需詳細資訊，請參閱《Amazon CloudWatch Logs 使用者指南》中的變更 CloudWatch 日誌中的日誌資料保留期間。

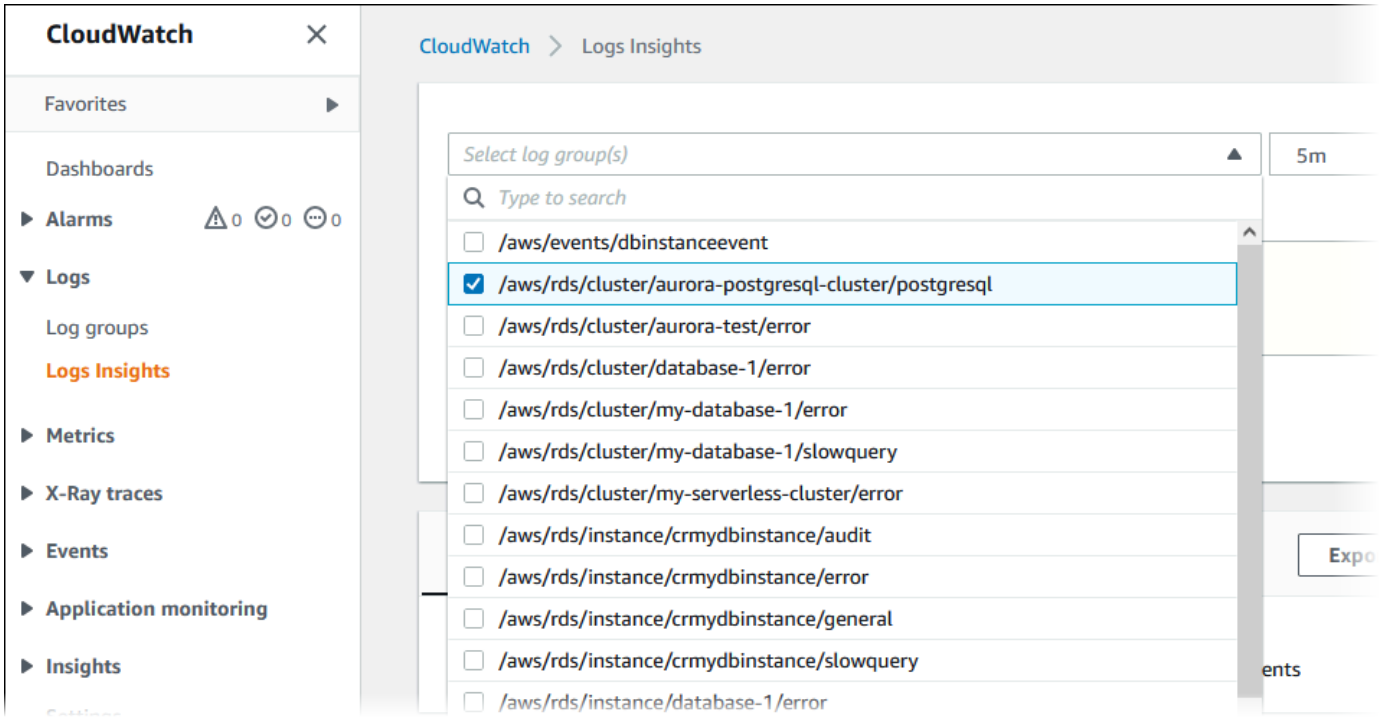
> **Tip**
>
> 您可以使用自動化組態 AWS CloudFormation，例如，建立具有預先定義日誌保留期、指標篩選條件和存取許可的日誌群組。

使用 CloudWatch Logs Insights 分析 PostgreSQL 日誌

透過以 CloudWatch Logs 形式發佈，且來自 Aurora PostgreSQL 資料庫叢集的 PostgreSQL 日誌，您可以使用 CloudWatch Logs Insights，以互動方式搜尋和分析 Amazon CloudWatch Logs 中的日誌資料。CloudWatch Logs Insights 包含查詢語言、範例查詢和其他用於分析日誌資料的工具，讓您可以識別潛在問題並驗證修正程式。如需詳細資訊，請參閱《Amazon CloudWatch Logs 使用者指南》中的使用 CloudWatch Logs Insights 分析日誌資料。

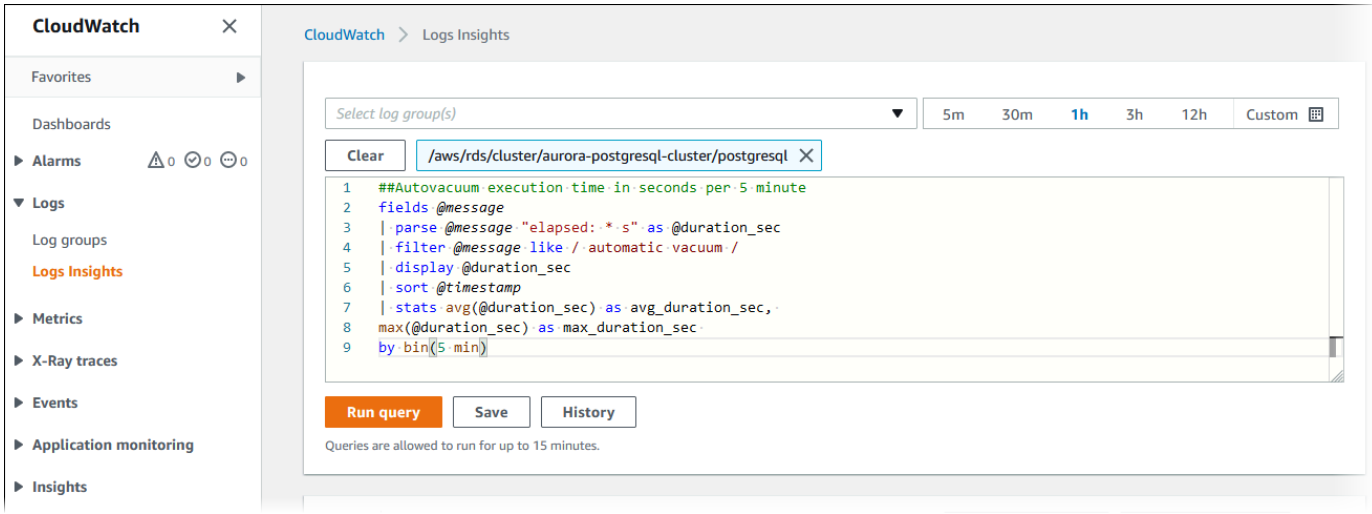
若要使用 CloudWatch Logs Insights 分析 PostgreSQL 日誌

1. 透過 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在導覽窗格中，開啟 Logs (日誌)，然後選擇 Log insights (日誌洞察)。
3. 在 Select log group(s) (選取日誌群組) 中，為您的 Aurora PostgreSQL 資料庫叢集選取日誌群組。

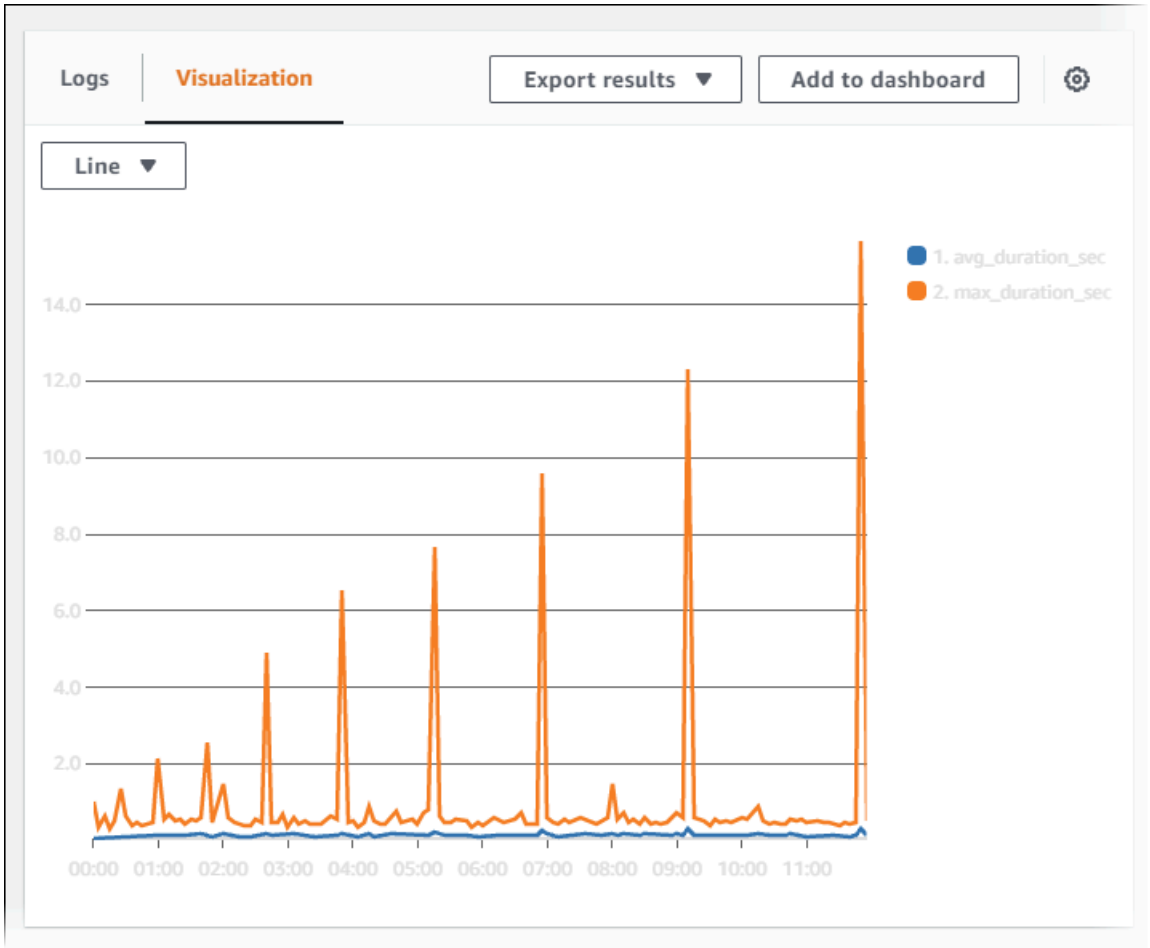


4. 在查詢編輯器中，刪除目前顯示的查詢，然後輸入以下查詢，然後選擇 Run query (執行查詢)。

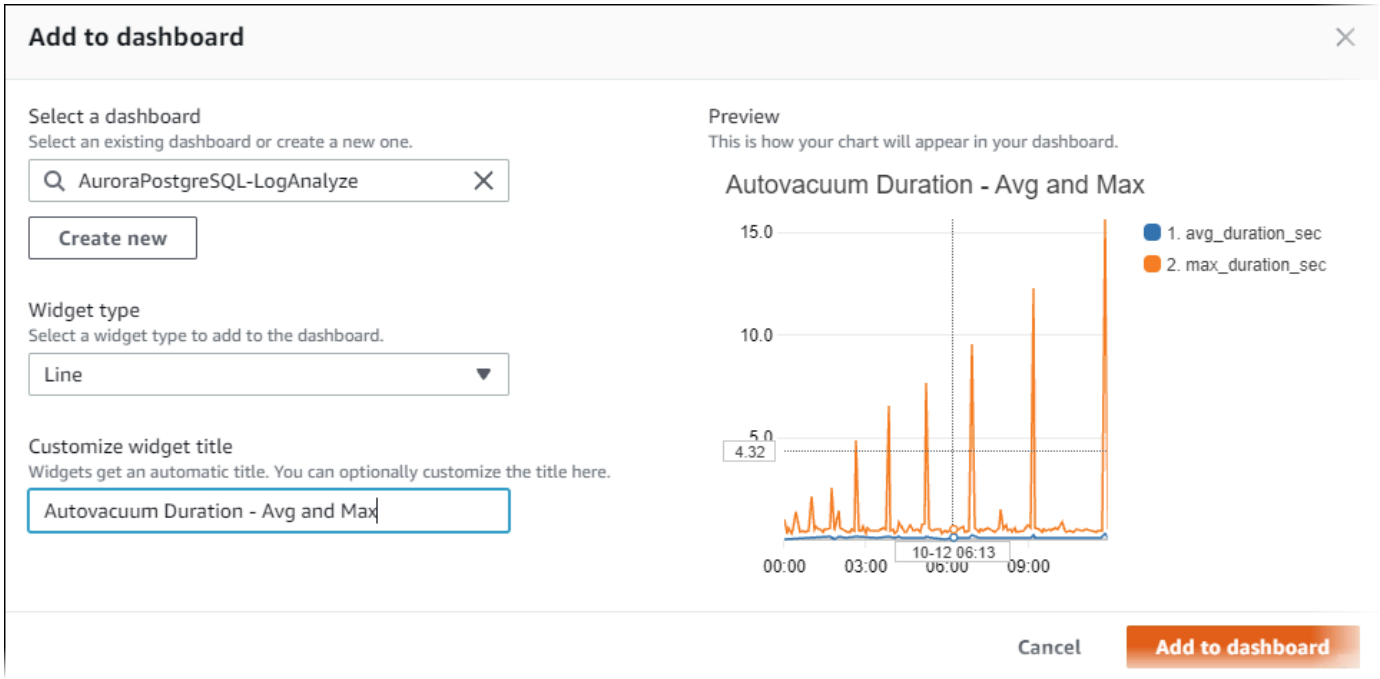
```
##Autovacuum execution time in seconds per 5 minute
fields @message
| parse @message "elapsed: * s" as @duration_sec
| filter @message like / automatic vacuum /
| display @duration_sec
| sort @timestamp
| stats avg(@duration_sec) as avg_duration_sec,
max(@duration_sec) as max_duration_sec
by bin(5 min)
```



5. 選擇 Visualization (視覺化) 標籤。



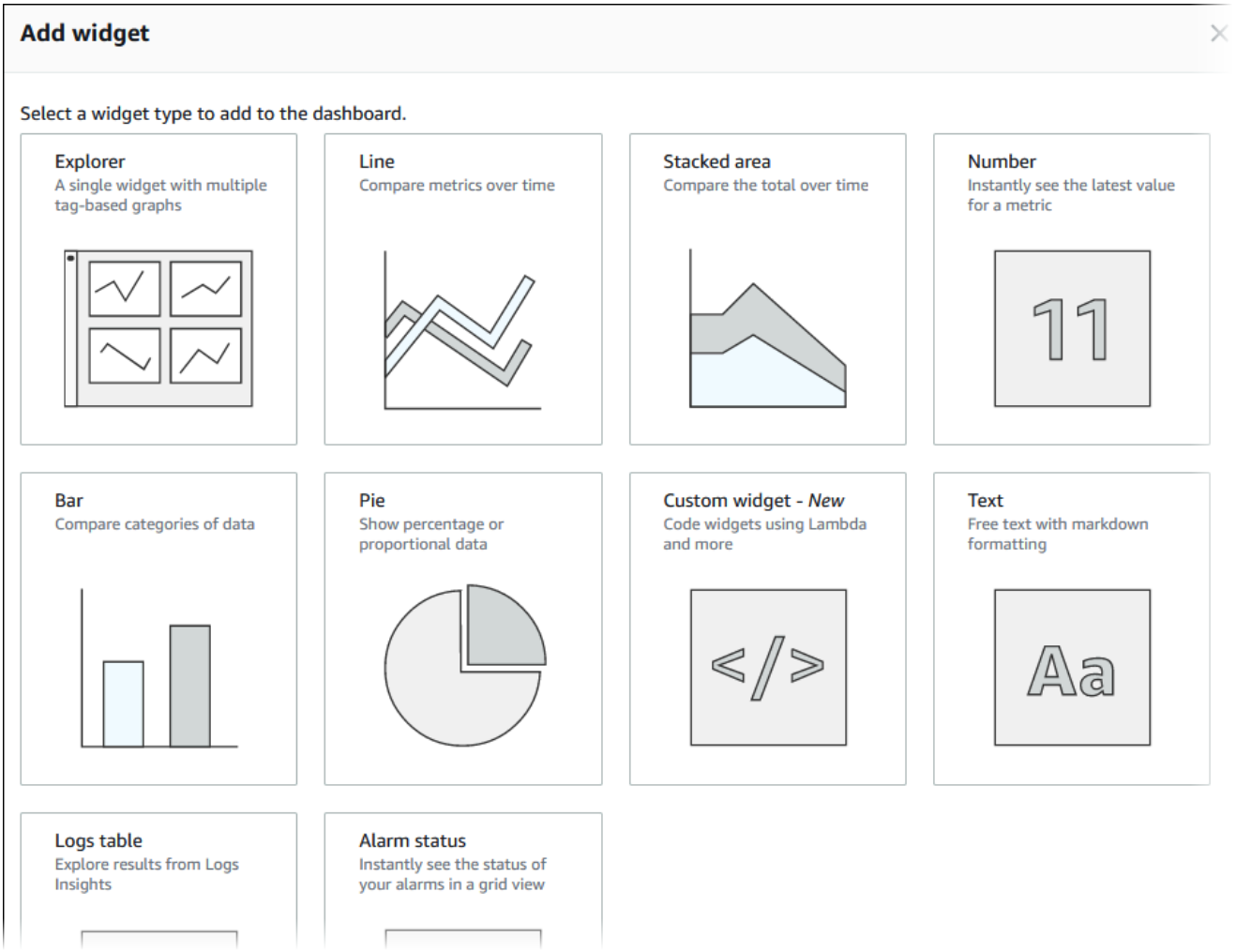
6. 選擇 Add to dashboard (新增至儀表板)。
7. 在 Select a dashboard (選取儀表板) 中，選取儀表板或輸入名稱以建立新儀表板。
8. 在 Widget type (小工具類型) 中，為您的視覺化選擇小工具類型。



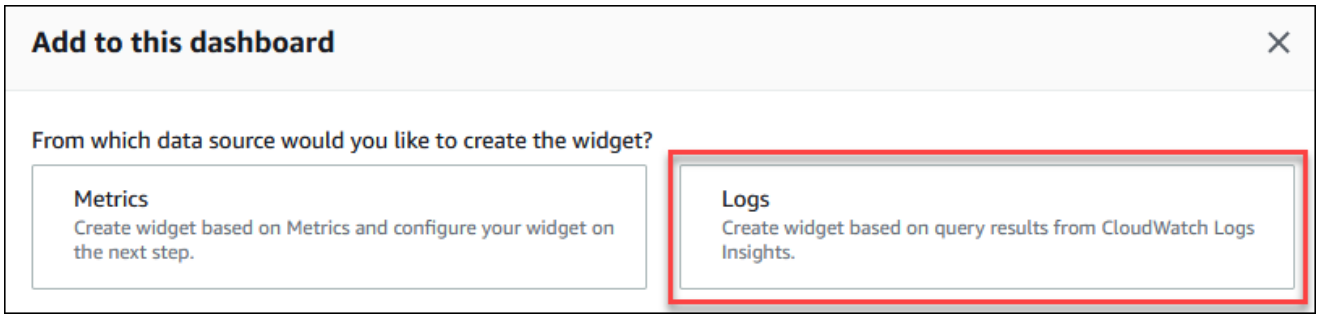
9. (選用) 根據您的日誌查詢結果新增更多小工具。

   a. 選擇 Add widget (新增 widget)。

   b. 選擇小工具類型，例如 Line (線條)。



c. 在 Add to this dashboard (新增至此儀表板) 視窗中，選擇 Logs (日誌)。

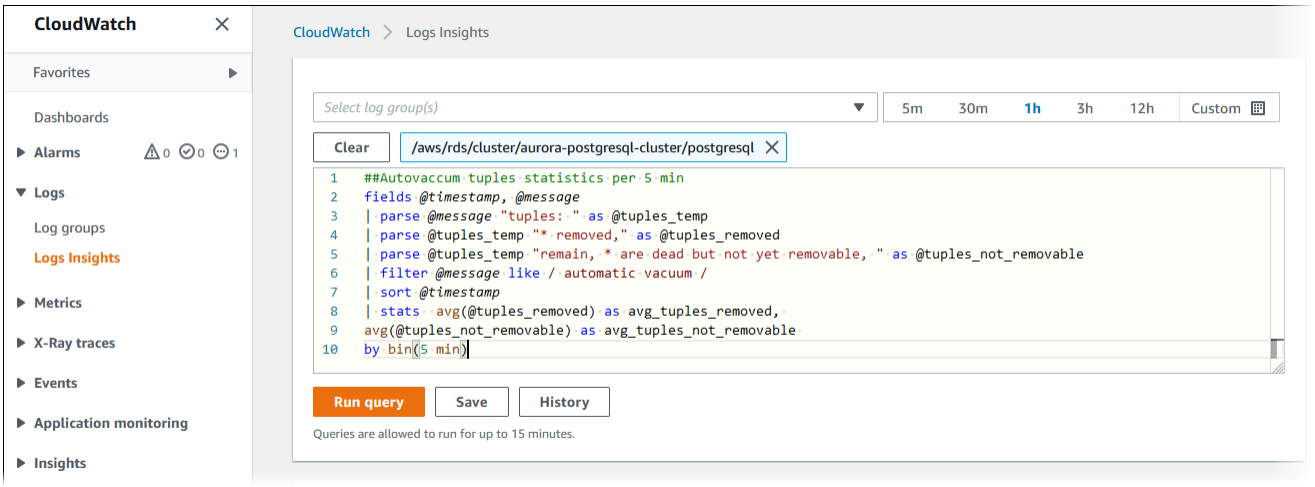


d. 在 Select log group(s) (選取日誌群組) 中，為您的資料庫叢集選取日誌群組。

e. 在查詢編輯器中，刪除目前顯示的查詢，然後輸入以下查詢，然後選擇 Run query (執行查詢)。

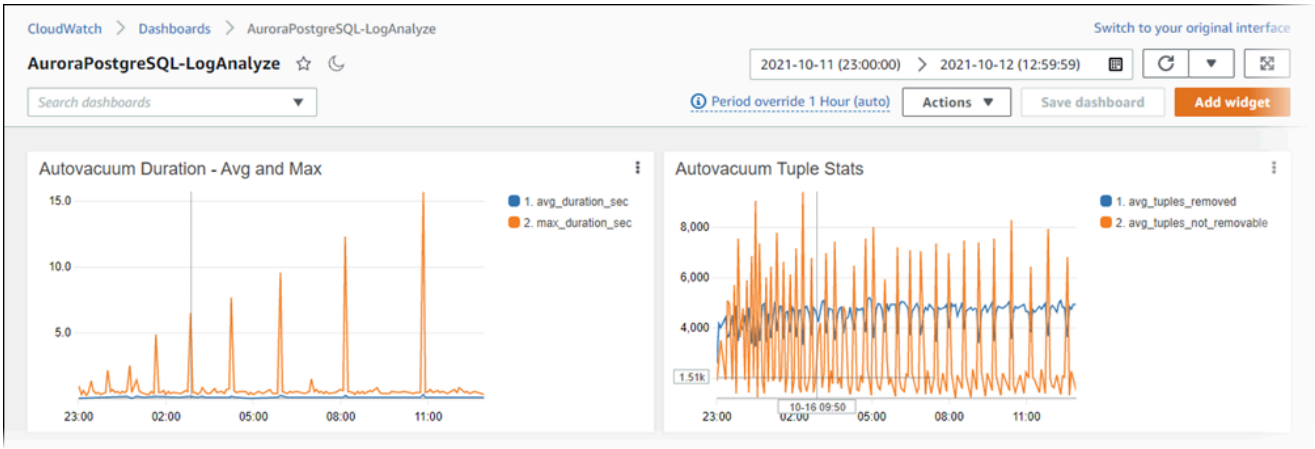
```
##Autovacuum tuples statistics per 5 min
fields @timestamp, @message
| parse @message "tuples: " as @tuples_temp
```

```
| parse @tuples_temp "* removed," as @tuples_removed
| parse @tuples_temp "remain, * are dead but not yet removable, " as
 @tuples_not_removable
| filter @message like / automatic vacuum /
| sort @timestamp
| stats  avg(@tuples_removed) as avg_tuples_removed,
avg(@tuples_not_removable) as avg_tuples_not_removable
by bin(5 min)
```



f. 選擇 Create widget (建立 widget)。

您的儀表板看起來應該會如下圖所示。



監控 Aurora PostgreSQL 的查詢執行計劃和尖峰記憶體

您可以在 Aurora PostgreSQL 資料庫執行個體中監控查詢執行計劃，以偵測造成目前資料庫負載的執行計劃，並使用 `aurora_compute_plan_id` 參數追蹤執行計劃隨時間的效能統計資料。每當查詢執行時，查詢使用的執行計劃都會獲指派識別符，而相同計劃的後續執行也會使用相同的識別符。

根據預設，`aurora_compute_plan_id` 會在 Aurora PostgreSQL 14.10、15.5 版和更新版本的資料庫參數群組中設定為 `OFF`。若要指派計劃識別符，請在參數群組中將 `aurora_compute_plan_id` 設定為 `ON`。

此計劃識別符用於數個提供不同用途的公用程式。

您可以監控資料庫執行個體中的查詢尖峰記憶體用量，以偵測從下列版本導致高資料庫記憶體用量的查詢：

- 16.3 版和所有更新版本
- 15.7 和更新版本
- 14.12 和更新版本

每當查詢執行時，系統都會追蹤查詢所使用的尖峰記憶體。查詢通常會執行多次；每個查詢都可以檢視所有執行的平均、最小和最大記憶體用量值。

主題

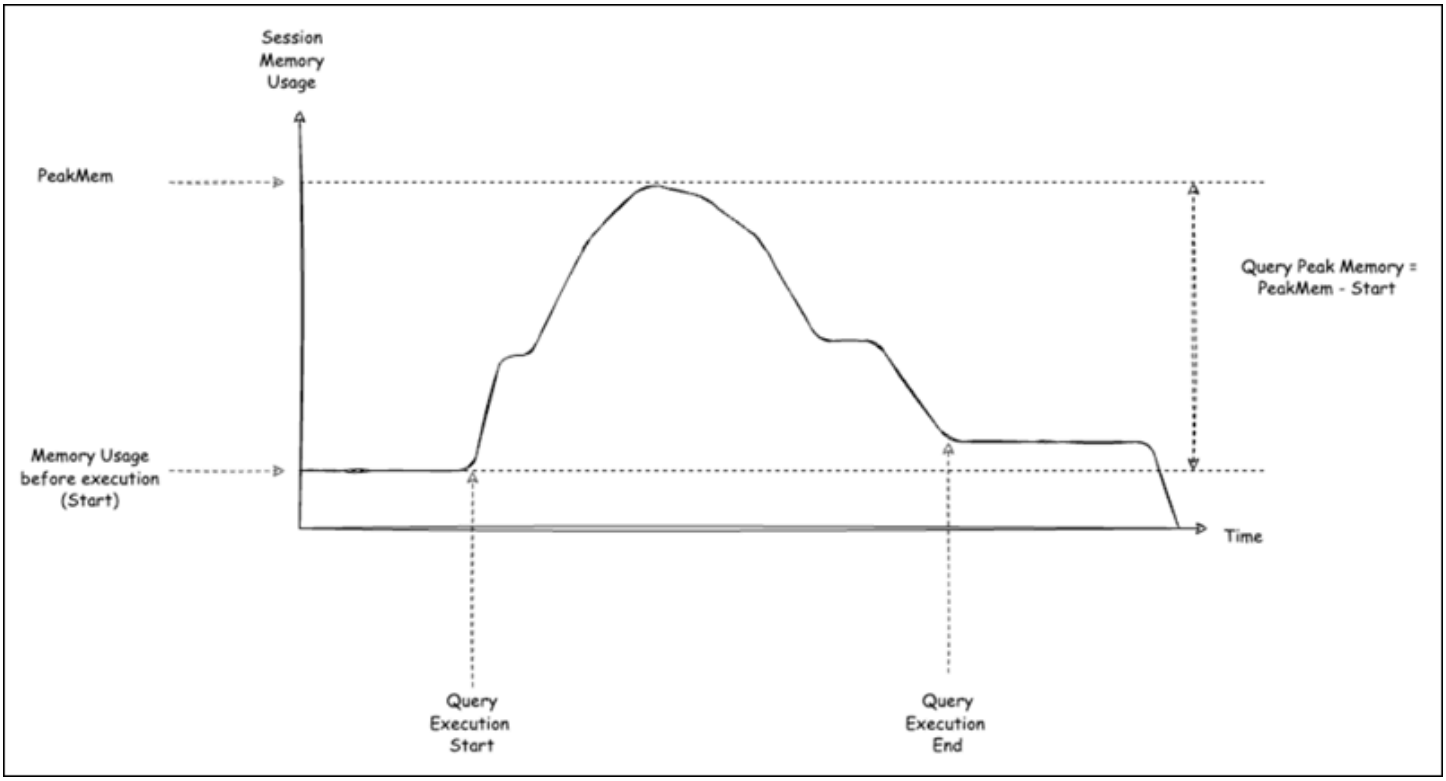
- 使用 Aurora 函數存取查詢執行計劃和尖峰記憶體
- Aurora PostgreSQL 查詢執行計劃的參數參考

使用 Aurora 函數存取查詢執行計劃和尖峰記憶體

透過 `aurora_compute_plan_id`，您可以使用下列函數存取執行計劃：

- aurora_stat_activity
- aurora_stat_plans

查詢尖峰記憶體不包含查詢處理開始之前配置的記憶體。針對每個查詢的規劃和執行階段，系統會分別追蹤和報告尖峰記憶體用量。



您可以使用下列函數來存取查詢尖峰記憶體統計資料。

- aurora_stat_statements
- aurora_stat_plans

如需這些函數的詳細資訊，請參閱 Aurora PostgreSQL 函數參考。

Aurora PostgreSQL 查詢執行計劃的參數參考

您可以使用資料庫參數群組中的下列參數來監控查詢執行計劃。

Parameters

- aurora_compute_plan_id
- aurora_stat_plans.minutes_until_recapture
- aurora_stat_plans.calls_until_recapture
- aurora_stat_plans.with_costs
- aurora_stat_plans.with_analyze
- aurora_stat_plans.with_timing
- aurora_stat_plans.with_buffers

- aurora_stat_plans.with_wal
- aurora_stat_plans.with_triggers

> **Note**
>
> `aurora_stat_plans.with_*` 參數的組態只會對新擷取的計劃生效。

aurora_compute_plan_id

`aurora_compute_plan_id` 是組態參數，可控制是否在查詢執行期間指派計劃識別符。

預設	允許的值	Description
off	0 (關閉)	設定為 `off` 以防止指派計劃識別符。
	1 (開啟)	設定為 `on` 以指派計劃識別符。

aurora_stat_plans.minutes_until_recapture

計劃重新擷取之前要經過的分鐘數。預設為 0，這會停用重新擷取計劃。超過 `aurora_stat_plans.calls_until_recapture` 閾值時，將會重新擷取計劃。

預設	允許的值	Description
0	0-1073741823	設定計劃重新擷取之前要經過的分鐘數。

aurora_stat_plans.calls_until_recapture

重新擷取計劃之前對計劃的呼叫次數。預設值為 0，這會停用在多次呼叫後重新擷取計劃。超過 `aurora_stat_plans.minutes_until_recapture` 閾值時，將會重新擷取計劃。

預設	允許的值	Description
0	0-1073741823	設定重新擷取計劃之前的呼叫次數。

aurora_stat_plans.with_costs

使用預估成本擷取 EXPLAIN 計劃。允許的值為 on 和 off。預設值為 on。

預設	允許的值	Description
on	0 (關閉)	不會顯示每個計劃節點的預估成本和資料列。
	1 (開啟)	顯示每個計劃節點的預估成本和資料列。

aurora_stat_plans.with_analyze

使用 ANALYZE 控制 EXPLAIN 計劃。此模式只會在第一次擷取計劃時使用。允許的值為 on 和 off。預設值為 off。

預設	允許的值	Description
off	0 (關閉)	不包含計劃的實際執行時間統計資料。
	1 (開啟)	包括計劃的實際執行時間統計資料。

aurora_stat_plans.with_timing

使用 ANALYZE 時，計劃時間將在解釋中擷取。預設值為 on。

預設	允許的值	Description
on	0 (關閉)	不包含實際啟動時間和在每個計劃節點中花費的時間。
	1 (開啟)	包括實際啟動時間和在每個計劃節點中花費的時間。

aurora_stat_plans.with_buffers

使用 ANALYZE 時，計劃緩衝區用量統計資料將在解釋中擷取。預設值為 off。

預設	允許的值	Description
off	0 (關閉)	不包含緩衝區用量的相關資訊。
	1 (開啟)	包含緩衝區用量的相關資訊。

aurora_stat_plans.with_wal

使用 ANALYZE 時，計劃 WAL 用量統計資料將在解釋中擷取。預設值為 `off`。

預設	允許的值	Description
off	0 (關閉)	不包含 WAL 記錄產生的資訊。
	1 (開啟)	包含 WAL 記錄產生的資訊。

aurora_stat_plans.with_triggers

使用 `ANALYZE` 時，計劃觸發程序執行統計資料將在解釋中擷取。預設值為 `off`。

預設	允許的值	Description
off	0 (關閉)	不包含觸發程序執行統計資料。
	1 (開啟)	包含觸發程序執行統計資料。

管理 Aurora PostgreSQL 的查詢執行計劃

Aurora PostgreSQL 查詢計劃管理是選用功能，您可以搭配 Amazon Aurora PostgreSQL 相容版本資料庫叢集使用。此功能封裝成您可以將其安裝在 Aurora PostgreSQL 資料庫叢集中的 `apg_plan_mgmt` 延伸模組。查詢計畫管理可讓您管理最佳化工具針對 SQL 應用程式產生的查詢執行計劃。`apg_plan_mgmt` AWS 延伸項目是以 PostgreSQL 資料庫引擎的原生查詢處理功能為基礎。

接下來，您可以尋找 Aurora PostgreSQL 查詢計劃管理功能、如何進行設定，以及如何將其與 Aurora PostgreSQL 資料庫叢集搭配使用的相關資訊。在開始之前，建議您檢閱 Aurora PostgreSQL 版本所適用 `apg_plan_mgmt` 延伸模組之特定版本的任何版本說明。如需詳細資訊，請參閱《Aurora PostgreSQL 版本說明》中的 Aurora PostgreSQL apg_plan_mgmt 延伸模組版本。

主題

- Aurora PostgreSQL 查詢計劃管理的概觀
- Aurora PostgreSQL 查詢計劃管理的最佳實務
- Aurora PostgreSQL 查詢計劃管理
- 擷取 Aurora PostgreSQL 執行計畫
- 使用 Aurora PostgreSQL 受管計劃
- 在 dba_plans 檢視中檢查 Aurora PostgreSQL 查詢計劃
- 改善 Aurora PostgreSQL 查詢計劃
- 刪除 Aurora PostgreSQL 查詢計劃
- 匯出和匯入 Aurora PostgreSQL 的受管計劃
- Aurora PostgreSQL 查詢計劃管理的參數參考
- Aurora PostgreSQL 查詢計劃管理的函數參考
- Aurora PostgreSQL 相容版本的 apg_plan_mgmt.dba_plans 檢視參考
- 查詢計畫管理中的進階功能

Aurora PostgreSQL 查詢計劃管理的概觀

Aurora PostgreSQL 查詢計劃管理旨在確保計劃穩定性，無論資料庫的變更是否可能導致查詢計劃迴歸。當最佳化工具在系統或資料庫變更之後，針對指定的 SQL 陳述式選擇次佳計劃時，就會發生「查詢計劃迴歸」。統計資料、限制條件、環境設定、查詢參數繫結的變更，以及 PostgreSQL 資料庫引擎的升級都可能造成計劃迴歸。

使用 Aurora PostgreSQL 查詢計劃管理時，您可以控制查詢執行計劃如何和何時變更。Aurora PostgreSQL 查詢計劃管理包括下列好處：

- 強制最佳化工具從少數已知的良好計劃中選擇，以提高計劃穩定性。
- 將計劃集中最佳化，然後整體分發最佳計劃。
- 識別未使用的索引，並評估建立或捨棄索引所造成的影響。
- 自動偵測最佳化工具新發現的最低成本計劃。
- 以較低的風險來嘗試新的最佳化工具功能，因為您可以選擇只核准可提高效能的計劃變更。

您可以主動使用查詢計劃管理提供的工具，為某些查詢指定最佳計劃。或者，您可以使用查詢計劃管理來反應不斷變化的情況，並避免計劃回歸。如需詳細資訊，請參閱Aurora PostgreSQL 查詢計劃管理的最佳實務。

主題

- 支援的 SQL 陳述式
- 查詢計劃管理限制
- 查詢計劃管理術語
- Aurora PostgreSQL 查詢計劃管理版本
- 開啟 Aurora PostgreSQL 查詢計劃管理
- 升級 Aurora PostgreSQL 查詢計劃管理
- 關閉 Aurora PostgreSQL 查詢計劃管理

支援的 SQL 陳述式

查詢計劃管理支援下列類型的 SQL 陳述式。

- 任何 SELECT、INSERT、UPDATE 或 DELETE 陳述式，而不論複雜性為何。
- 預備陳述式。如需詳細資訊，請參閱 PostgreSQL 文件中的 PREPARE。
- 動態陳述式，包括以立即模式執行的陳述式。如需詳細資訊，請參閱 PostgreSQL 文件中的 Dynamic SQL 和 EXECUTE IMMEDIATE。
- 嵌入式 SQL 命令和陳述式。如需詳細資訊，請參閱 PostgreSQL 文件中的嵌入式 SQL 命令。
- 具名函數內的陳述式。如需詳細資訊，請參閱 PostgreSQL 文件中的 CREATE FUNCTION。
- 包含暫存資料表的陳述式。
- 程序和 DO 區塊內的陳述式。

您可以在手動模式下搭配 EXPLAIN 使用查詢計畫管理來擷取計劃，而不需實際執行它。如需詳細資訊，請參閱分析最佳化工具的所選擇計劃。若要深入了解查詢計劃管理模式 (手動、自動)，請參閱 擷取 Aurora PostgreSQL 執行計畫。

Aurora PostgreSQL 查詢計劃管理支援所有 PostgreSQL 語言功能，包括分割資料表、繼承、列層級安全性和遞迴一般資料表表達式 (CTE)。若要深入了解這些 PostgreSQL 語言功能，請參閱 PostgreSQL 文件中的資料表分割、資料列安全政策和 WITH 查詢 (通用資料表運算式)，以及其他主題。

如需不同版本之 Aurora PostgreSQL 查詢計畫管理功能的相關資訊，請參閱《Aurora PostgreSQL 版本資訊》中的 Aurora PostgreSQL apg_plan_mgmt 延伸模組版本。

查詢計劃管理限制

現行版本的 Aurora PostgreSQL 查詢計畫管理具有下列限制。

- 不會針對參考系統關係的陳述式擷取計畫 – 不會擷取參考系統關係的陳述式 (例如 pg_class)。這是設計的，以防止擷取內部使用的大量系統產生計劃。這也適用於檢視內的系統資料表。
- Aurora PostgreSQL 資料庫叢集可能需要較大的資料庫執行個體類別 – 根據工作負載，查詢計畫管理可能需要具有 2 個以上 vCPU 的資料庫執行個體類別。max_worker_processes 的數目受資料庫執行個體類別大小限制。2-vCPU 資料庫執行個體類別 (例如 db.t3.medium) 提供的 max_worker_processes 數目可能不夠指定的工作負載使用。建議若您使用查詢計畫管理，請為 Aurora PostgreSQL 資料庫叢集選擇具有 2 個以上 vCPU 的資料庫執行個體類別。

  當資料庫執行個體類別不支援工作負載時，查詢計畫管理會引發如下錯誤訊息。

  ```
  WARNING: could not register plan insert background process
  HINT: You may need to increase max_worker_processes.
  ```

  在這種情況下，您應該將 Aurora PostgreSQL 資料庫叢集縱向擴展到具有更多記憶體的資料庫執行個體類別大小。如需詳細資訊，請參閱資料庫執行個體類別的支援資料庫引擎。
- 已存放在工作階段中的計畫不會受到影響 - 查詢計畫管理提供了一種方式，可在不變更應用程式碼的情況下影響查詢計畫。不過，當一般計畫已存放在現有的工作階段時，而且如果您想要變更其查詢計畫，則必須先在資料庫叢集參數群組中將 plan_cache_mode 設為 force_custom_plan。
- 出現下列情況時，apg_plan_mgmt.dba_plans 和 pg_stat_statements 中的 queryid 可能會出現差異：
  - 物件在存放於 apg_plan_mgmt.dba_plans 之後遭到刪除並重新建立。
  - apg_plan_mgmt.plans 表格是從另一個叢集匯入的。

如需不同版本之 Aurora PostgreSQL 查詢計畫管理功能的相關資訊，請參閱《Aurora PostgreSQL 版本資訊》中的 Aurora PostgreSQL apg_plan_mgmt 延伸模組版本。

查詢計劃管理術語

本主題使用下列術語。

受管陳述式

最佳化工具在查詢計劃管理之下擷取的 SQL 陳述式。受管陳述式會將一或多個查詢執行計劃存放在 `apg_plan_mgmt.dba_plans` 檢視中。

計劃基準

所指定受管陳述式的核准計劃集。亦即，受管陳述式的所有其 `status` 資料欄在 `dba_plan` 檢視中具有「已核准」的計劃。

計劃歷史記錄

所指定受管陳述式的所有擷取計劃集。計劃歷史記錄包含針對陳述式擷取的所有計劃，不論狀態為何。

查詢計劃迴歸

此情況是最佳化工具在對資料庫環境進行指定的變更 (例如新的 PostgreSQL 版本或統計資料的變更) 之前選擇較不理想的計劃。

Aurora PostgreSQL 查詢計劃管理版本

所有目前可用的 Aurora PostgreSQL 版本都支援查詢計劃管理。如需詳細資訊，請參閱《Aurora PostgreSQL 版本資訊》中的 Amazon Aurora PostgreSQL 更新。

當您安裝 `apg_plan_mgmt` 擴充功能時，查詢計劃管理功能會新增至您的 Aurora PostgreSQL 資料庫叢集。不同版本的 Aurora PostgreSQL 支援不同版本的 `apg_plan_mgmt` 擴充功能。建議您將查詢計劃管理擴充功能升級至 Aurora PostgreSQL 的最新版本。

> **Note**
>
> 如需每個 `apg_plan_mgmt` 擴充功能版本的版本備註，請參閱《Aurora PostgreSQL 版本資訊》中的 Aurora PostgreSQL apg_plan_mgmt 擴充功能版本。

您可以使用 psql 和使用中繼命令 \dx 列出擴充功能來連線至執行個體，以識別叢集上執行的版本，如下所示。

```
labdb=> \dx
                        List of installed extensions
     Name      | Version |    Schema     |                            Description
---------------+---------+---------------
+-------------------------------------------------------------------
 apg_plan_mgmt | 1.0     | apg_plan_mgmt | Amazon Aurora with PostgreSQL compatibility
 Query Plan Management
 plpgsql       | 1.0     | pg_catalog    | PL/pgSQL procedural language
(2 rows)
```

輸出顯示此叢集使用的是 1.0 版的擴充功能。只有特定 apg_plan_mgmt 版本適用於指定的 Aurora PostgreSQL 版本。在某些情況下，您可能需要將 Aurora PostgreSQL 資料庫叢集升級至新的次要版本，或套用修補程式，以便您可以升級至最新版的查詢計劃管理。輸出中顯示的 apg_plan_mgmt 1.0 版來自 Aurora PostgreSQL 10.17 版資料庫叢集，該叢集沒有更新版本的 apg_plan_mgmt 可用。在此情況下，Aurora PostgreSQL 資料庫叢集應升級至較新版的 PostgreSQL。

如需將 Aurora PostgreSQL 資料庫叢集升級至新版 PostgreSQL 的詳細資訊，請參閱 Amazon Aurora PostgreSQL 的資料庫引擎更新。

若要了解如何升級 apg_plan_mgmt 擴充功能，請參閱 升級 Aurora PostgreSQL 查詢計劃管理。

開啟 Aurora PostgreSQL 查詢計劃管理

為 Aurora PostgreSQL 資料庫叢集設定查詢計劃管理牽涉及安裝擴充功能，以及變更數個資料庫叢集參數設定。您需要 rds_superuser 許可，才能安裝 apg_plan_mgmt 擴充功能和開啟 Aurora PostgreSQL 資料庫叢集的功能。

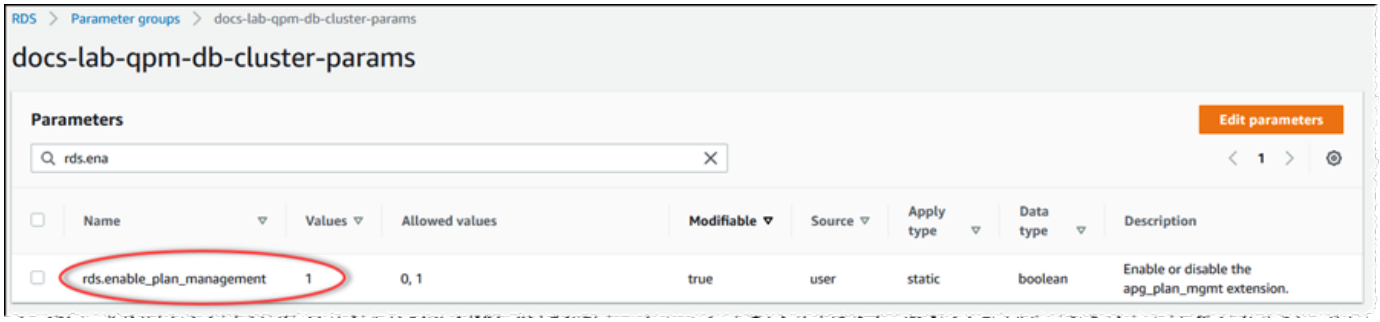
安裝擴充功能會建立新角色 apg_plan_mgmt。此角色可讓資料庫使用者檢視、管理及維護查詢計劃。身為具有 rds_superuser 許可的管理員，請務必視需要將 apg_plan_mgmt 角色授與資料庫使用者。

只有具有 rds_superuser 角色的使用者才能完成下列程序。建立 rds_superuser 延伸及其 apg_plan_mgmt 角色需要 apg_plan_mgmt。使用者必須獲授予 apg_plan_mgmt 角色，才能管理 apg_plan_mgmt 延伸。

開啟 Aurora PostgreSQL 資料庫叢集的查詢計劃管理

下列步驟會針對提交至 Aurora PostgreSQL 資料庫叢集的所有 SQL 陳述式開啟查詢計劃管理。這就是所謂的「自動」模式。若要進一步了解模式之間的差異，請參閱 擷取 Aurora PostgreSQL 執行計畫。

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 針對您的 Aurora PostgreSQL 資料庫叢集建立自訂資料庫叢集參數群組。您需要變更某些參數，才能啟動查詢計劃管理並設定其行為。如需詳細資訊，請參閱在 Amazon Aurora 中建立資料庫參數群組。
3. 開啟自訂資料庫叢集參數群組，並將 `rds.enable_plan_management` 參數設定為 `1`，如下圖所示。



   如需詳細資訊，請參閱在 Amazon Aurora 中修改資料庫叢集參數群組中的參數。
4. 建立您可以用來在執行個體層級設定查詢計劃參數的自訂資料庫參數群組。如需詳細資訊，請參閱在 Amazon Aurora 中建立資料庫叢集參數群組。
5. 修改 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，來使用自訂資料庫參數群組。如需詳細資訊，請參閱修改資料庫叢集中的資料庫執行個體。
6. 修改 Aurora PostgreSQL 資料庫叢集，來使用自訂資料庫叢集參數群組。如需詳細資訊，請參閱使用主控台、CLI 和 API 修改資料庫叢集。
7. 重新啟動資料庫執行個體來啟用自訂參數群組設定。
8. 使用 `psql` 或 pgAdmin 連線至 Aurora PostgreSQL 資料庫叢集的資料庫執行個體端點。下列範例使用 `rds_superuser` 角色的預設 `postgres` 帳戶。

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=my-db
```

9. 為資料庫執行個體建立 `apg_plan_mgmt` 擴充功能，如下所示。

```
labdb=> CREATE EXTENSION apg_plan_mgmt;
CREATE EXTENSION
```

> **Tip**
>
> 在應用程式的範本資料庫中安裝 apg_plan_mgmt 延伸模組。預設範本資料庫命名為 template1。若要進一步了解，請參閱 PostgreSQL 文件中的範本資料庫。

10. 將 apg_plan_mgmt.capture_plan_baselines 參數變更為 automatic。此設定會導致最佳化工具針對每個已計劃或執行兩次或多次的 SQL 陳述式產生計畫。

> **Note**
>
> 查詢計劃管理也有您可以用於特定 SQL 陳述式的「手動」模式。若要進一步了解，請參閱擷取 Aurora PostgreSQL 執行計畫。

11. 將 apg_plan_mgmt.use_plan_baselines 參數的值變更為 "on"。此參數會導致最佳化工具從其計劃基準中選擇陳述式的計劃。如需詳細資訊，請參閱 使用 Aurora PostgreSQL 受管計劃。

> **Note**
>
> 您可以修改工作階段的其中任一動態參數值，而不需要重新啟動執行個體。

當您的查詢計劃管理設定完成時，請務必將 apg_plan_mgmt 角色授與任何需要檢視、管理或維護查詢計劃的資料庫使用者。

升級 Aurora PostgreSQL 查詢計劃管理

建議您將查詢計劃管理擴充功能升級至 Aurora PostgreSQL 的最新版本。

1. 以擁有 rds_superuser 權限的使用者身分，連線至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。如果您在設定執行個體時保留了預設名稱，則以 postgres 連線。這個範例展示如何使用 psql，但您也可以視需要使用 pgAdmin。

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. 執行以下查詢來升級擴充功能。

```
ALTER EXTENSION apg_plan_mgmt UPDATE TO '2.1';
```

3. 使用 apg_plan_mgmt.validate_plans 函數更新所有計劃的雜湊。最佳化工具會驗證所有「已核准」、「未核准」和「已拒絕」計劃，以確保它們仍然是新版擴充功能的可行計劃。

   ```
   SELECT apg_plan_mgmt.validate_plans('update_plan_hash');
   ```

   若要進一步了此函數，請參閱 驗證計劃。

4. 使用 apg_plan_mgmt.reload 函數，搭配來自 dba_plan 檢視的已驗證計劃，重新整理共用記憶體中的任何計劃。

   ```
   SELECT apg_plan_mgmt.reload();
   ```

若要深入了解可用於查詢計劃管理的所有功能，請參閱 Aurora PostgreSQL 查詢計劃管理的函數參考。

關閉 Aurora PostgreSQL 查詢計劃管理

您可以隨時關閉 apg_plan_mgmt.use_plan_baselines 和 apg_plan_mgmt.capture_plan_baselines 來停用查詢計劃管理。

```
labdb=> SET apg_plan_mgmt.use_plan_baselines = off;

labdb=> SET apg_plan_mgmt.capture_plan_baselines = off;
```

Aurora PostgreSQL 查詢計劃管理的最佳實務

查詢計劃管理可讓您控制查詢執行計劃如何和何時變更。作為 DBA，使用 QPM 時的主要目標包括在資料庫發生變更時防止迴歸，以及控制是否允許最佳化工具使用新計畫。下面是一些使用查詢計畫管理的建議最佳實務。主動式和被動式計畫管理方法在如何和何時核准使用新計畫上有所不同。

內容

- 主動式計劃管理有助於避免效能退化
  - 主要版本升級之後確保計畫穩定性
- 被動式計劃管理可偵測並修復效能迴歸

主動式計劃管理有助於避免效能退化

為了防止發生計畫效能迴歸，您可以執行一個程序，將新探索到之計畫的效能與已核准計畫之現有基準的效能進行比較，以發展計畫基準，然後自動核准最快的計畫集作為新基準。如此一來，一段時間後若探索到更快的計畫，計畫的基準也會隨之改善。

1. 在開發環境中，識別會對效能或系統傳輸量造成最大影響的 SQL 陳述式。然後，擷取這些陳述式的計劃，如針對特定 SQL 陳述式來手動擷取計劃 和自動擷取計劃所述。
2. 從開發環境中匯出擷取的計劃，然後匯入生產環境中。如需更多詳細資訊，請參閱 匯出和匯入 Aurora PostgreSQL 的受管計劃。
3. 在生產環境中，執行您的應用程式，並強制使用已核准的受管計劃。如需更多詳細資訊，請參閱 使用 Aurora PostgreSQL 受管計劃。當應用程式執行時，也要新增最佳化工具所發現的新計劃。如需更多詳細資訊，請參閱 自動擷取計劃。
4. 分析未核准的計劃，並核准其中表現良好的計劃。如需更多詳細資訊，請參閱 評估計劃效能。
5. 當您的應用程式繼續執行時，最佳化工具會視情況開始使用新計劃。

主要版本升級之後確保計畫穩定性

PostgreSQL 的每個主要版本都包括旨在提高效能之查詢最佳化工具的增強和變更。不過，最佳化處理程式在舊版中產生的查詢執行計畫，可能會導致較新的升級版本中的效能迴歸。您可使用查詢計畫管理器解決這些效能問題，並確保主要版本升級後的計畫穩定性。

最佳化工具一律使用成本最低的核准計畫，即使同一陳述式存在多個已核准計畫也是如此。升級之後，最佳化工具可能會發現新計畫，但會儲存為未核准的計畫。只有當使用被動式計畫管理和 unapproved_plan_execution_threshold 參數獲得核准時，才會執行這些計畫。您可以使用計畫管理的主動式樣式搭配 evolve_plan_baselines 參數，最大限度地提高計畫穩定性。這會將新計畫的效能與舊計畫進行比較，並核准或拒絕比下一個最佳計畫至少快 10% 的計畫。

升級後，您可使用 `evolve_plan_baselines` 函數，使用查詢參數繫結來比較升級前後的計畫效能。下列步驟假定您在生產環境中使用了核准的託管計劃，詳情請參閱 使用 Aurora PostgreSQL 受管計劃。

1. 升級之前，請在執行查詢計畫管理器的情況下執行應用程式。當應用程式執行時，新增最佳化工具所發現的新計劃。如需詳細資訊，請參閱自動擷取計劃。
2. 評估每個計畫的效能。如需詳細資訊，請參閱評估計劃效能。

3. 升級後，請使用 `evolve_plan_baselines` 函數再次分析核准的計畫。比較使用查詢參數繫結前後的效能。若新計畫速度快，您可將其新增至已核准的計畫中。若其比相同參數繫結的另一個計畫快，則可將較慢的計畫標記為 Rejected (拒絕)。

   如需詳細資訊，請參閱核准較佳計劃。如需此函數的參考資訊，請參閱 apg_plan_mgmt.evolve_plan_baselines。

如需詳細資訊，請參閱 Ensuring consistent performance after major version upgrades with Amazon Aurora PostgreSQL-Compatible Edition Query Plan Management (使用 Amazon Aurora PostgreSQL 相容版本查詢計畫管理確保主要版本升級後的效能一致)。

> **Note**
>
> 當您使用邏輯複寫或 AWS DMS執行主要版本升級時，請務必複寫 `apg_plan_mgmt` 結構描述，以確保將現有的計畫複製到已升級的執行個體。如需邏輯複寫的詳細資訊，請參閱 使用邏輯複寫來執行 Aurora PostgreSQL 的主要版本升級。

被動式計劃管理可偵測並修復效能迴歸

在應用程式執行時對其進行監控，您可偵測導致效能迴歸的計劃。當您偵測到迴歸時，您可以下列這些步驟手動拒絕或修正不良的計劃。

1. 當您的應用程式執行時，請強制使用受管計劃並自動將新發現的計劃新增為未核准。如需更多詳細資訊，請參閱 使用 Aurora PostgreSQL 受管計劃 及 自動擷取計劃。
2. 監控執行中的應用程式是否效能退化。
3. 當您發現計劃退化時，請將計劃的狀態設為 `rejected`。最佳化工具下一次執行 SQL 陳述式時，它會自動忽略已拒絕的計劃，並改用另一個已核准的計劃。如需更多詳細資訊，請參閱 拒絕或停用較慢的計劃。

   在某些情況下，您可能選擇修正不良的計劃，而非拒絕、停用或刪除計劃。使用 `pg_hint_plan` 延伸來試驗改善計劃。在 `pg_hint_plan` 中，您使用特別註解來指示最佳化工具如何更改其平常建立計劃的方式。如需詳細資訊，請參閱使用 pg_hint_plan 修正計劃。

Aurora PostgreSQL 查詢計劃管理

為 Aurora PostgreSQL 資料庫叢集開啟查詢計劃管理後，最佳化工具會針對其處理多次的任何 SQL 陳述式產生並儲存查詢執行計劃。最佳化工具一律會將受管陳述式第一個產生之計劃的狀態設定為 Approved，並將它儲存於 dba_plans 檢視中。

針對受管陳述式儲存的核准計劃集稱為計劃基線。當您的應用程式執行時，最佳化工具可能會針對受管陳述式產生額外計劃。最佳化工具會將額外的擷取計劃設為 Unapproved 狀態。

之後，您可以判斷 Unapproved 計劃是否表現良好，然後將它們變更為 Approved、Rejected 或 Preferred。若要這麼做，您可以使用 apg_plan_mgmt.evolve_plan_baselines 函數或 apg_plan_mgmt.set_plan_status 函數。

當最佳化工具產生 SQL 陳述式的計劃時，查詢計劃管理會將計劃儲存在 apg_plan_mgmt.plans 資料表中。已獲授與 apg_plan_mgmt 角色的資料庫使用者可以透過查詢 apg_plan_mgmt.dba_plans 檢視來查看計劃詳細資訊。例如，下列查詢會針對非生產 Aurora PostgreSQL 資料庫叢集列出檢視中目前計劃的詳細資訊。

- sql_hash – SQL 陳述式的識別碼，其為 SQL 陳述式標準化文字的雜湊值。
- plan_hash – 計劃的唯一識別碼，其為 sql_hash 與計劃雜湊的組合。
- status – 計劃的狀態。最佳化工具可以執行已核准的計劃。
- enabled – 指出計劃已備妥可供使用 (true) 或未備妥 (false)。
- plan_outline – 用來重建實際執行計劃的計劃表示法。樹狀結構中的運算子會對應到 EXPLAIN 輸出中的運算子。

apg_plan_mgmt.dba_plans 檢視還有更資料欄，其中包含計劃的所有詳細資訊，例如上次使用計劃的時間。如需完整詳細資訊，請參閱 Aurora PostgreSQL 相容版本的 apg_plan_mgmt.dba_plans 檢視參考。

標準化和 SQL 雜湊

在 apg_plan_mgmt.dba_plans 檢視中，您可以透過 SQL 雜湊值來識別受管陳述式。SQL 雜湊是根據已去掉某些差異 (例如常值) 的 SQL 陳述式標準化表示法來計算。

每個 SQL 陳述式的標準化程序都會保留空格和大小寫，因此您仍然可以閱讀並了解 SQL 陳述式的要點。標準化會移除或取代下列項目。

- 前導區塊註釋

- EXPLAIN 關鍵字和 EXPLAIN 選項，以及 EXPLAIN ANALYZE
- 尾隨空格
- 所有常值

以下列陳述式為例。

```
/*Leading comment*/ EXPLAIN SELECT /* Query 1 */ * FROM t WHERE x > 7 AND y = 1;
```

查詢計劃管理會將此陳述式標準化，如下所示：

```
SELECT /* Query 1 */ * FROM t WHERE x > CONST AND y = CONST;
```

標準化允許相同的 SQL 雜湊用於類似的 SQL 陳述式，這些陳述式可能僅在其常值或參數值有所不同。換言之，相同的 SQL 雜湊可能存在多個計劃，而不同的計劃在不同條件下成為最佳。

> **Note**
>
> 與不同結構描述搭配使用的單一 SQL 陳述式有不同的計畫，因為它在執行時繫結至特定結構描述。規劃工具會將統計資料用於結構描述繫結來選擇最佳計劃。

若要深入了解最佳化工具如何選擇計劃，請參閱 使用 Aurora PostgreSQL 受管計劃。在該節中，您可以了解如何在實際使用計劃之前使用 EXPLAIN 和 EXPLAIN ANALYZE 預覽此計劃。如需詳細資訊，請參閱分析最佳化工具的所選擇計劃。如需概述選擇計劃之程序的影像，請參閱 最佳化工具如何選擇要執行的計劃。

擷取 Aurora PostgreSQL 執行計畫

Aurora PostgreSQL 查詢計劃管理提供兩種不同的模式 (自動或手動) 來擷取查詢執行計劃。您可以透過將 apg_plan_mgmt.capture_plans_baselines 值設定為 automatic 或 manual 來選擇模式。您可以使用手動計劃擷取，以擷取特定 SQL 陳述式的執行計劃。您也可以使用自動計劃擷取，以擷取當您的應用程式執行時，已執行兩次或更多次的所有 (或最慢) 計劃。

當擷取計劃時，最佳化工具會將針對受管陳述式而擷取的第一個計劃設為 approved 狀態。最佳化工具會將針對受管陳述式而擷取的任何額外計劃設為 unapproved 狀態。然而，有時可能會儲存不止一個具有 approved 狀態的計劃。當有多個計劃針對某一陳述式被並行建立，而且該陳述式的第一個計劃被遞交之前，可能會發生此種情況。

若要控制可擷取和存放在 dba_plans 檢視中的計劃數量上限，請在資料庫執行個體層級參數群組中設定 apg_plan_mgmt.max_plans 參數。變更 apg_plan_mgmt.max_plans 參數之後需要重新啟動資料庫執行個體，新的值才會生效。如需更多詳細資訊，請參閱 apg_plan_mgmt.max_plans 參數。

針對特定 SQL 陳述式來手動擷取計劃

如果您有一組已知的 SQL 陳述式需要管理，請將陳述式放入 SQL 指令碼檔案中，然後手動擷取計劃。以下顯示的 psql 範例示範如何針對一組 SQL 陳述式來手動擷取查詢計劃。

```
psql> SET apg_plan_mgmt.capture_plan_baselines = manual;
psql> \i my-statements.sql
psql> SET apg_plan_mgmt.capture_plan_baselines = off;
```

在擷取每個 SQL 陳述式的計劃之後，最佳化工具會在 apg_plan_mgmt.dba_plans 檢視中新增一列。

我們建議您在 SQL 指令碼檔案中使用 EXPLAIN 或 EXPLAIN EXECUTE 陳述式。請確認您在參數值中容納了足夠的變化，以擷取您關心的所有計劃。

如果您知道比最佳化工具的最低成本計劃更好的計劃，您可以強制最佳化工具使用更好的計劃。若要這麼做，請指定一個或多個最佳化工具提示。如需更多詳細資訊，請參閱 使用 pg_hint_plan 修正計劃。若要比較 unapproved 和 approved 計劃的效能，並核准、拒絕或刪除它們，請參閱評估計劃效能。

自動擷取計劃

在如下的情況中，請使用自動計劃擷取：

- 您不知道您想管理的特定 SQL 陳述式。
- 您有數百或數千個 SQL 陳述式需要管理。
- 您的應用程式使用用戶端 API。例如，JDBC 使用未命名的備妥陳述式或大量模式陳述式，但無法以 psql 表達。

自動擷取計劃

1. 在資料庫執行個體層級參數群組中，將 apg_plan_mgmt.capture_plan_baselines 設為 automatic，以啟用自動計劃擷取。如需詳細資訊，請參閱修改 Amazon Aurora 中的資料庫參數群組中的參數。
2. 當應用程式執行時，最佳化工具會針對至少執行兩次的每個 SQL 陳述式來擷取計劃。

當應用程式以預設查詢計劃管理參數設定來執行時，最佳化工具會針對至少執行兩次的每個 SQL 陳述式來擷取計劃。使用預設值時擷取所有計劃的執行時間額外負荷非常小，適用於生產環境。

停用自動計劃擷取

- 從資料庫執行個體層級參數群組，將 `apg_plan_mgmt.capture_plan_baselines` 參數設為 `off`。

若要測量未核准計劃的效能，並核准、拒絕或刪除它們，請參閱評估計劃效能。

使用 Aurora PostgreSQL 受管計劃

若要讓最佳化工具使用針對受管陳述式而擷取的計劃，請將參數 apg_plan_mgmt.use_plan_baselines 設為 true。以下是本機執行個體範例。

```
SET apg_plan_mgmt.use_plan_baselines = true;
```

當應用程式執行時，此設定會使得最佳化工具針對每個受管陳述式，使用有效且已啟用的最低成本、優先的或已核准的計劃。

分析最佳化工具的所選擇計劃

當 apg_plan_mgmt.use_plan_baselines 參數設為 true 時，您可以使用 EXPLAIN ANALYZE SQL 陳述式，讓最佳化工具顯示它未來執行陳述式時會使用的計劃。下列是 範例。

```
EXPLAIN ANALYZE EXECUTE rangeQuery (1,10000);
```

```
                                                        QUERY PLAN
-----------------------------------------------------------------------------
 Aggregate  (cost=393.29..393.30 rows=1 width=8) (actual time=7.251..7.251 rows=1
 loops=1)
   ->  Index Only Scan using t1_pkey on t1 t  (cost=0.29..368.29 rows=10000 width=0)
 (actual time=0.061..4.859 rows=10000 loops=1)
Index Cond: ((id >= 1) AND (id <= 10000))
         Heap Fetches: 10000
 Planning time: 1.408 ms
 Execution time: 7.291 ms
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1984047223, Plan Hash: 512153379
```

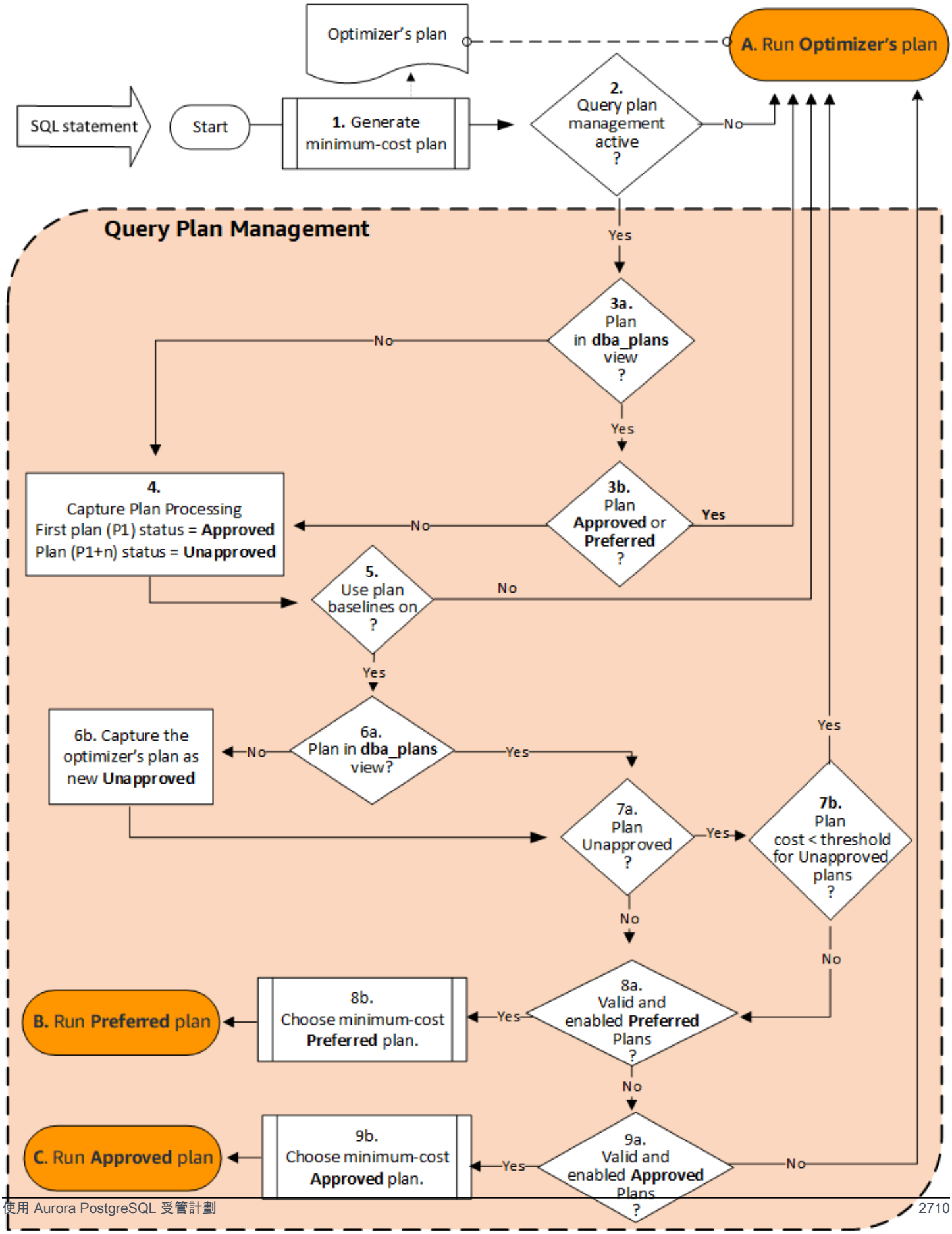
輸出會顯示來自基準且將執行的已核准計劃。不過，輸出也會顯示它找到更低成本的計劃。在此案例中，您啟用自動計劃擷取，如自動擷取計劃所述，以擷取這個新的最低成本計劃。

最佳化工具一律會擷取新計劃作為 Unapproved。使用 apg_plan_mgmt.evolve_plan_baselines 函數來比較計劃，並將它們變更為已核准、已拒絕或已停用。如需詳細資訊，請參閱評估計劃效能。

最佳化工具如何選擇要執行的計劃

執行計劃的成本是最佳化工具為了比較不同計劃而做的預估。計算計劃的成本時，最佳化工具會納入該計劃所需的 CPU 和 I/O 作業等因素。若要了解 PostgreSQL 查詢規劃器成本預估的詳細資訊，請參閱 PostgreSQL 文件中的查詢規劃。

下圖顯示當查詢計劃管理處於或不處於作用中狀態時，如何為指定 SQL 陳述式選擇計劃。


Optimizer's plan
A. Run Optimizer's plan
SQL statement
Start
1. Generate minimum-cost plan
2. Query plan management active ?
No
Query Plan Management
Yes
3a. Plan in dba_plans view ?
No
Yes
4. Capture Plan Processing
First plan (P1) status = Approved
Plan (P1+n) status = Unapproved
3b. Plan Approved or Preferred ?
No
Yes
5. Use plan baselines on ?
No
Yes
6b. Capture the optimizer's plan as new Unapproved
6a. Plan in dba_plans view?
No
Yes
7a. Plan Unapproved ?
Yes
7b. Plan cost < threshold for Unapproved plans ?
Yes
No
No
B. Run Preferred plan
8b. Choose minimum-cost Preferred plan.
Yes
8a. Valid and enabled Preferred Plans ?
No
C. Run Approved plan
9b. Choose minimum-cost Approved plan.
Yes
9a. Valid and enabled Approved Plans ?
No

流程如下所示：

1. 最佳化工具會為 SQL 陳述式產生最低成本計劃。
2. 如果查詢計劃管理未處於作用中狀態，則會立即執行最佳化工具的計劃 (A. 執行最佳化工具的計劃)。當 `apg_plan_mgmt.capture_plan_baselines` 和 `apg_plan_mgmt.use_plan_baselines` 參數均為預設設定 (分別為「off」和「false」) 時，查詢計劃管理會處於非作用中狀態。

   否則，查詢計劃管理會處於作用中狀態。在此情況下，SQL 陳述式和最佳化工具的計劃會在選擇計劃之前進一步評估。

   > **Tip**
   >
   > 具有 `apg_plan_mgmt` 角色的資料庫使用者可以主動比較計劃，更改計劃的狀態，並根據需要強制使用特定計劃。如需詳細資訊，請參閱改善 Aurora PostgreSQL 查詢計劃。

3. SQL 陳述式可能已經有過去由查詢計劃管理儲存的計劃。計劃以及用來建立它們之 SQL 陳述式的相關資訊依同儲存於 `apg_plan_mgmt.dba_plans`。計劃的相關資訊包括其狀態。計劃的狀態可以決定其是否已使用，如下所示。
   a. 如果計劃不在 SQL 陳述式的預存計劃中，表示該特定計畫是第一次由最佳化工具針對指定 SQL 陳述式而產生的。計劃會傳送至「擷取計劃處理」(4)。
   b. 如果計劃位於預存計劃中，且其狀態為「已核准」或「偏好」，則會執行該計劃 (A. 執行最佳化工具的計劃)。

      如果計劃位於預存計劃中，但既不是「已核准」也不是「偏好」，則計劃會傳送至「擷取計劃處理」(4)。
4. 第一次擷取指定 SQL 陳述式的計劃時，計劃的狀態一律設為「已核准 (P1)」。如果最佳化工具隨後針對相同的 SQL 陳述式產生相同的計劃，則該計劃的狀態會變更為「未核准」(P1+n)。

   擷取計劃並更新其狀態後，評估會繼續進行下一個步驟 (5)。
5. 計劃的基準由 SQL 陳述式的歷史記錄及其在各種狀態下的計劃所組成。根據「使用計劃基準」選項是否開啟，查詢計劃管理可以在選擇計劃時將基準列入考量，如下所示。
   - 當 `apg_plan_mgmt.use_plan_baselines` 參數設定為預設值 (`false`) 時，使用計劃基準為「off」。計劃在運行之前不會與基準進行比較 (A. 執行最佳化工具的計劃)。
   - 當 `apg_plan_mgmt.use_plan_baselines` 參數設定為 `true` 時，使用計劃基準為「on」。該計劃會使用基準 (6) 進一步評估。
6. 該計劃會與基準中陳述式的其他計劃進行比較。

a. 如果最佳化工具的計劃位於基準中的計劃，則會檢查其狀態 (7a)。

b. 如果最佳化工具的計劃不在基準的計劃中，該計劃會以新 Unapproved 計劃的形式新增至陳述式的計劃。

7. 系統會檢查計劃的狀態，以決定是否為「未核准」。

a. 如果計劃的狀態為「未核准」，則計劃的預估成本會與針對未核准執行計劃臨界值指定的成本預估進行比較。

- 如果計劃的預估成本低於臨界值，則最佳化工具會使用它，即使它是未核准的計劃 (A. 執行最佳化工具的計劃)。一般而言，最佳化工具不會執行未核准的計劃。然而，當 apg_plan_mgmt.unapproved_plan_execution_threshold 參數指定成本臨界值，最佳化工具會將「未核准」計劃的成本與臨界值進行比較。如果預估成本低於臨界值，最佳化工具會執行該計劃。如需詳細資訊，請參閱apg_plan_mgmt.unapproved_plan_execution_threshold。
- 如果計劃的預估成本不低於臨界值，則會檢查計劃的其他屬性 (8a)。

b. 如果計劃的狀態為「未核准」以外的其他屬性，則會勾選其他屬性 (8a)。

8. 最佳化工具不會使用已停用的計劃。亦即 enable 屬性設定為 'f' (false) 的計劃。最佳化工具也不會使用狀態為「已拒絕」的計劃。

   最佳化工具無法使用任何無效的計劃。隨著時間的推移，當計劃所依賴的物件 (例如索引或資料表) 被移除或刪除時，計劃可能會變成無效。

a. 如果陳述式有任何已啟用且有效的優先計劃，最佳化工具會從為此 SQL 陳述式儲存的「優先計劃」中選擇最低成本計劃。然後，最佳化工具會執行已核准的最低成本計劃。

b. 如果陳述式沒有任何已啟用且有效的偏好計劃，則會在下一個步驟 (9) 中進行評估。

9. 如果陳述式有任何已啟用且有效的已核准計劃，最佳化工具會從為此 SQL 陳述式儲存的「已核准」計劃中選擇最低成本計劃。然後，最佳化工具會執行已核准的最低成本計劃。

   如果陳述式沒有任何有效且已啟用的已核准計劃，最佳化工具會使用最低成本計劃 (A. 執行最佳化工具的計劃)。

在 dba_plans 檢視中檢查 Aurora PostgreSQL 查詢計劃

已獲授與 apg_plan_mgmt 角色的資料庫使用者和管理員可以檢視和管理儲存在 apg_plan_mgmt.dba_plans 中的計劃。Aurora PostgreSQL 資料庫叢集管理員 (具有 rds_superuser 權限的人員) 必須明確地將此角色授與需要使用查詢計劃管理的資料庫使用者。

apg_plan_mgmt 檢視包含每個資料庫之所有受管 SQL 陳述式的計劃歷史記錄，而這些資料庫位於 Aurora PostgreSQL 資料庫叢集的寫入器執行個體上。此檢視可讓您檢查計劃、其狀態、上次使用時間，以及所有其他相關詳細資訊。

如 標準化和 SQL 雜湊 中所討論，每個受管計劃是以結合的 SQL 雜湊值和計劃雜湊值來識別。您可以使用工具 (例如 Amazon RDS Performance Insights) 搭配這些識別符，以追蹤個別計劃效能。如需績效詳情的更多資訊，請參閱使用 Amazon RDS 績效詳情。

列出受管計劃

若要列出受管計劃，請在 apg_plan_mgmt.dba_plans 檢視上使用 SELECT 陳述式。下列範例在 dba_plans 檢視中顯示幾個欄，例如 status，用以識別已核准和未核准的計劃。

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;

 sql_hash   | plan_hash |   status   | enabled | stmt_name
------------+-----------+------------+---------+------------
 1984047223 | 512153379 | Approved   | t       | rangequery
 1984047223 | 512284451 | Unapproved | t       | rangequery
 (2 rows)
```

為了便於閱讀，查詢和顯示的輸出僅列出 dba_plans 檢視中的幾列。如需完整資訊，請參閱 Aurora PostgreSQL 相容版本的 apg_plan_mgmt.dba_plans 檢視參考。

改善 Aurora PostgreSQL 查詢計劃

透過評估計劃效能和修正計劃來改善查詢計劃管理。如需改善查詢計劃的詳細資訊，請參閱下列主題。

主題

- 評估計劃效能
- 使用 pg_hint_plan 修正計劃

評估計劃效能

當最佳化工具將計劃擷取為未核准之後，請使用 apg_plan_mgmt.evolve_plan_baselines 函數，根據實際效能來比較計劃。根據效能試驗的結果而定，您可以將計劃的狀態從未核准變更為已核准或已拒絕。如果計劃不符合您的需要，您可以改為決定使用 apg_plan_mgmt.evolve_plan_baselines 函數來暫時停用計劃。

核准較佳計劃

以下範例示範如何使用 apg_plan_mgmt.evolve_plan_baselines 函數，將受管計劃的狀態變更為已核准。

```
SELECT apg_plan_mgmt.evolve_plan_baselines (
   sql_hash,
   plan_hash,
   min_speedup_factor := 1.0,
   action := 'approve'
)
FROM apg_plan_mgmt.dba_plans WHERE status = 'Unapproved';
```

```
NOTICE:     rangequery (1,10000)
NOTICE:     Baseline   [ Planning time 0.761 ms, Execution time 13.261 ms]
NOTICE:     Baseline+1 [ Planning time 0.204 ms, Execution time 8.956 ms]
NOTICE:     Total time benefit: 4.862 ms, Execution time benefit: 4.305 ms
NOTICE:     Unapproved -> Approved
evolve_plan_baselines
----------------------
0
(1 row)
```

輸出顯示 rangequery 陳述式的效能報告，參數繫結為 1 和 10,000。新的未核准計劃 (Baseline +1) 比先前已核准的最佳計劃 (Baseline) 更好。若要確認新計劃現在 Approved，請查看 apg_plan_mgmt.dba_plans 檢視。

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;
```

```
sql_hash  | plan_hash |  status  | enabled | stmt_name
------------+-----------+----------+---------+------------
1984047223 | 512153379 | Approved | t       | rangequery
1984047223 | 512284451 | Approved | t       | rangequery
(2 rows)
```

受管計劃現在包含兩個已核准的計劃，當作陳述式的計劃基線。您也可以呼叫 apg_plan_mgmt.set_plan_status 函數，直接將計劃的狀態欄位設為 'Approved'、'Rejected'、'Unapproved' 或 'Preferred'。

拒絕或停用較慢的計劃

若要拒絕或停用計劃，請將 'reject' 或 'disable' 當作動作參數傳給 apg_plan_mgmt.evolve_plan_baselines 函數。此範例停用任何已擷取的 Unapproved 計劃，該計劃比陳述式的最佳 Approved 計劃還慢至少 10%。

```
SELECT apg_plan_mgmt.evolve_plan_baselines(
sql_hash,  -- The managed statement ID
plan_hash, -- The plan ID
1.1,       -- number of times faster the plan must be
'disable'  -- The action to take. This sets the enabled field to false.
)
FROM apg_plan_mgmt.dba_plans
WHERE status = 'Unapproved' AND   -- plan is Unapproved
origin = 'Automatic';       -- plan was auto-captured
```

您也可以直接將計劃設為已拒絕或已停用。若要直接將計劃的已啟用欄位設為 true 或 false，請呼叫 apg_plan_mgmt.set_plan_enabled 函數。若要直接將計劃的狀態欄位設為 'Approved'、'Rejected'、'Unapproved' 或 'Preferred'，請呼叫 apg_plan_mgmt.set_plan_status 函數。

若要刪除無效和您預期仍無效的計劃，請使用 apg_plan_mgmt.validate_plans 函數。此函數可讓您刪除或停用無效的計劃。如需詳細資訊，請參閱驗證計劃。

使用 pg_hint_plan 修正計劃

查詢最佳化工具精心設計來為所有陳述式尋找最佳計劃，且最佳化工具在大多數情況下會找到好的計劃。不過，有時您可能知道有一個比最佳化工具所產生的計劃好得多的計劃。有兩個建議方法可讓最佳化工具產生理想的計劃，包括使用 pg_hint_plan 延伸，或在 PostgreSQL 中設定 Grand Unified Configuration (GUC) 變數：

- pg_hint_plan 延伸 – 使用 PostgreSQL 的 pg_hint_plan 延伸來指定「提示」，以修改規劃器的運作方式。若要安裝和進一步了解如何使用 pg_hint_plan 延伸，請參閱 pg_hint_plan 文件。
- GUC 變數 – 覆寫一個或多個成本模型參數或其他最佳化工具參數，例如 from_collapse_limit 或 GEQO_threshold。

當您使用以上其中一項技巧來強制查詢最佳化工具使用計劃時，您也可以使用查詢計劃管理來擷取和強制使用新的計劃。

您可以使用 pg_hint_plan 延伸來變更 SQL 陳述式的聯結順序、聯結方法或存取路徑。您使用 SQL 註解搭配特殊的 pg_hint_plan 語法，以修改最佳化工具建立計劃的方式。例如，假設有問題的 SQL 陳述式具有雙向聯結。

```
SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

再假設最佳化工具選擇聯結順序 (t1, t2)，但您知道聯結順序 (t2, t1) 更快。下列提示強制最佳化工具使用較快的聯結順序 (t2, t1)。包含 EXPLAIN 會讓最佳化工具為 SQL 陳述式產生計劃，但不需執行陳述式。(未顯示輸出。)

```
/*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

下列步驟示範如何使用 pg_hint_plan。

使用 pg_hint_plan 修改最佳化工具所產生的計劃並擷取計劃

1. 啟用手動擷取模式。

   ```
   SET apg_plan_mgmt.capture_plan_baselines = manual;
   ```

2. 針對所關注的 SQL 陳述式來指定提示。

   ```
   /*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
   FROM t1, t2
   WHERE t1.id = t2.id;
   ```

   執行之後，最佳化工具會擷取 apg_plan_mgmt.dba_plans 檢視中的計劃。擷取的計劃不含特殊的 pg_hint_plan 註解語法，因為查詢計劃管理會移除開頭註解而將陳述式標準化。

3. 使用 apg_plan_mgmt.dba_plans 檢視來檢視受管計劃。

   ```
   SELECT sql_hash, plan_hash, status, sql_text, plan_outline
   FROM apg_plan_mgmt.dba_plans;
   ```

4. 將計劃的狀態設定為 Preferred。當最低成本的計劃還不是 Approved 或 Preferred 時，這麼做可以保證最佳化工具選擇執行該計劃，而非從已核准的一組計劃中選擇。

```
SELECT apg_plan_mgmt.set_plan_status(sql-hash, plan-hash, 'preferred' );
```

5. 關閉手動計劃擷取並強制使用受管計劃。

```
SET apg_plan_mgmt.capture_plan_baselines = false;
SET apg_plan_mgmt.use_plan_baselines = true;
```

現在，當原始 SQL 陳述式執行時，最佳化工具會選擇 Approved 或 Preferred 計劃。如果最低成本計劃不是 Approved 也不是 Preferred，最佳化工具會選擇 Preferred 計劃。

刪除 Aurora PostgreSQL 查詢計劃

刪除您未使用或無效的執行計劃。如需有關刪除計劃的詳細資訊，請參閱下列各節。

主題

- 刪除計劃
- 驗證計劃

刪除計劃

如果計劃已經超過一個月 (特別是 32 天) 未使用，系統會自動刪除這些計劃。此為 apg_plan_mgmt.plan_retention_period 參數的預設設定。您可以延長計劃保留期間，或從值 1 開始的較短期間。判斷上次使用計畫後經過的天數，方法是從目前日期減去 last_used 日期。last_used 日期是最佳化工具選擇計劃做為最低成本計劃或執行計劃的最近日期。日期是針對 apg_plan_mgmt.dba_plans 檢視中的計劃存放的。

建議您刪除長時間未使用或沒有用處的計劃。每個計劃都有 last_used 日期，最佳化工具每次執行計劃，或選擇計劃當作陳述式的最低成本計劃時，就會更新此日期。檢查最後 last_used 日期，以識別您可以安全刪除的計劃。

下列查詢會傳回三欄表格，其中包含計劃總數、無法刪除的計劃數，以及已順利刪除的計劃數。它具有巢狀查詢，這是一個範例，展示如何使用 apg_plan_mgmt.delete_plan 函數，來刪除過去 31 天未被選為最低成本計劃且其狀態為 Rejected 的所有計劃。

```
SELECT (SELECT COUNT(*) from apg_plan_mgmt.dba_plans) total_plans,
       COUNT(*) FILTER (WHERE result = -1) failed_to_delete,
       COUNT(*) FILTER (WHERE result = 0) successfully_deleted
```

```
       FROM (
            SELECT apg_plan_mgmt.delete_plan(sql_hash, plan_hash) as result
            FROM apg_plan_mgmt.dba_plans
            WHERE last_used < (current_date - interval '31 days')
            AND status <> 'Rejected'
            ) as dba_plans ;
```

```
 total_plans | failed_to_delete | successfully_deleted
-------------+------------------+----------------------
           3 |                0 |                    2
```

如需詳細資訊，請參閱apg_plan_mgmt.delete_plan。

若要刪除無效和您預期仍無效的計劃，請使用 apg_plan_mgmt.validate_plans 函數。此函數可讓您刪除或停用無效的計劃。如需詳細資訊，請參閱驗證計劃。

> **Important**
>
> 如果您未刪除無關的計劃，最後可能會耗盡已保留給查詢計劃管理的共用記憶體。若要控制受管計劃可用的記憶體，請使用 apg_plan_mgmt.max_plans 參數。在自訂資料庫參數群組中設定此參數，然後重新啟動資料庫執行個體，讓變更生效。如需更多詳細資訊，請參閱 apg_plan_mgmt.max_plans 參數。

驗證計劃

使用 apg_plan_mgmt.validate_plans 函數來刪除或停用無效的計劃。

當計劃所依賴的物件 (例如索引或資料表) 移除時，計劃會變成無效或過時。不過，如果重建已移除的物件，則計劃可能只是暫時無效。如果無效的計劃後來變成有效，您可能會選擇停用無效的計劃，或什麼都不做，而非刪除它。

若要尋找並刪除所有無效且在上週未使用的計劃，請如下使用 apg_plan_mgmt.validate_plans 函數。

```
SELECT apg_plan_mgmt.validate_plans(sql_hash, plan_hash, 'delete')
FROM apg_plan_mgmt.dba_plans
WHERE last_used < (current_date - interval '7 days');
```

若要直接啟用或停用計劃，請使用 apg_plan_mgmt.set_plan_enabled 函數。

匯出和匯入 Aurora PostgreSQL 的受管計劃

您可以匯出受管計劃，再匯入另一個資料庫執行個體中。

匯出受管計劃

獲授權使用者可以將 apg_plan_mgmt.plans 資料表的任何子集複製到另一個資料表，然後使用 pg_dump 命令儲存它。以下是範例。

```
CREATE TABLE plans_copy AS SELECT *
FROM apg_plan_mgmt.plans [ WHERE predicates ] ;
```

```
% pg_dump --table apg_plan_mgmt.plans_copy -Ft mysourcedatabase > plans_copy.tar
```

```
DROP TABLE apg_plan_mgmt.plans_copy;
```

匯入受管計劃

1. 將所匯出受管計劃的 .tar 檔案複製到將還原計劃的系統上。
2. 使用 pg_restore 命令將 tar 檔案複製到新的資料表。

   ```
   % pg_restore --dbname mytargetdatabase -Ft plans_copy.tar
   ```
3. 合併 plans_copy 資料表與 apg_plan_mgmt.plans 資料表，如下列範例所示。

   > **Note**
   >
   > 在某些情況下，您可能會從 apg_plan_mgmt 延伸套件的某個版本傾印，然後還原成不同的版本。在這種情況下，計劃資料表中的資料欄可能不同。如果是這樣的話，請明確命名各欄，而非使用 SELECT *。

   ```
   INSERT INTO apg_plan_mgmt.plans SELECT * FROM plans_copy
    ON CONFLICT ON CONSTRAINT plans_pkey
    DO UPDATE SET
    status = EXCLUDED.status,
    enabled = EXCLUDED.enabled,
    -- Save the most recent last_used date
    --
   ```

```
last_used = CASE WHEN EXCLUDED.last_used > plans.last_used
THEN EXCLUDED.last_used ELSE plans.last_used END,
-- Save statistics gathered by evolve_plan_baselines, if it ran:
--
estimated_startup_cost = EXCLUDED.estimated_startup_cost,
estimated_total_cost = EXCLUDED.estimated_total_cost,
planning_time_ms = EXCLUDED.planning_time_ms,
execution_time_ms = EXCLUDED.execution_time_ms,
total_time_benefit_ms = EXCLUDED.total_time_benefit_ms,
execution_time_benefit_ms = EXCLUDED.execution_time_benefit_ms;
```

4. 將受管計劃重新載入共用記憶體，並移除暫時計劃資料表。

```
SELECT apg_plan_mgmt.reload(); -- refresh shared memory
DROP TABLE plans_copy;
```

Aurora PostgreSQL 查詢計劃管理的參數參考

您可以使用本節中列出的參數設定 apg_plan_mgmt 擴充功能的偏好設定。您可以在自訂資料庫叢集參數以及與 Aurora PostgreSQL 資料庫叢集關聯的資料庫參數群組中使用這些資料庫參數。這些參數會控制查詢計劃管理功能的行為，以及它對最佳化工具的影響。如需設定查詢計劃管理的詳細資訊，請參閱 開啟 Aurora PostgreSQL 查詢計劃管理。如果 apg_plan_mgmt 擴充功能未按照該部分的詳細資訊進行設定，則變更下列參數沒有任何作用。如需修改參數的相關資訊，請參閱 在 Amazon Aurora 中修改資料庫叢集參數群組中的參數 和 Amazon Aurora 資料庫執行個體的資料庫參數群組。

Parameters

- apg_plan_mgmt.capture_plan_baselines
- apg_plan_mgmt.plan_capture_threshold
- apg_plan_mgmt.explain_hashes
- apg_plan_mgmt.log_plan_enforcement_result
- apg_plan_mgmt.max_databases
- apg_plan_mgmt.max_plans
- apg_plan_mgmt.plan_hash_version
- apg_plan_mgmt.plan_retention_period
- apg_plan_mgmt.unapproved_plan_execution_threshold
- apg_plan_mgmt.use_plan_baselines

- auto_explain.hashes

apg_plan_mgmt.capture_plan_baselines

擷取最佳化工具針對每個 SQL 陳述式產生的查詢執行計劃，並將它們儲存於 dba_plans 檢視。根據預設，可儲存的計劃數目上限為 10,000，如 apg_plan_mgmt.max_plans 參數所指定。如需參考資訊，請參閱 apg_plan_mgmt.max_plans。

您可以在自訂資料庫叢集參數群組或自訂資料庫參數群組中設定此參數。變更此參數的值不需要重新開機。

預設	允許的值	Description
off	自動	在工作階段層級或參數群組中套用此設定，以擷取使用兩次以上的計劃。
	手動	在工作階段層級或參數群組中套用此設定，以擷取使用一或多次的計劃。
	off	關閉計劃擷取。

如需詳細資訊，請參閱擷取 Aurora PostgreSQL 執行計畫。

apg_plan_mgmt.plan_capture_threshold

指定一個閾值，以便如果查詢執行計畫的總成本低於閾值，則不會在 apg_plan_mgmt.dba_plans 檢視中擷取該計畫。

變更此參數的值不需要重新開機。

預設	允許的值	Description
0	0 - 1.79769e+308	設定擷取計畫的 apg_plan_mgmt 查詢計畫總執行成本的閾值。

如需詳細資訊，請參閱在 dba_plans 檢視中檢查 Aurora PostgreSQL 查詢計劃。

apg_plan_mgmt.explain_hashes

指定 EXPLAIN [ANALYZE] 是否在其輸出尾端顯示 sql_hash 和 plan_hash。變更此參數的值不需要重新開機。

預設	允許的值	Description
0	0 (關閉)	EXPLAIN 不會顯示沒有雜湊 true 選項的 sql_hash 和 plan_hash。
	1 (開啟)	EXPLAIN 會顯示沒有雜湊 true 選項的 sql_hash 和 plan_hash。

apg_plan_mgmt.log_plan_enforcement_result

指定是否必須記錄結果，以查看是否適當地使用 QPM 受管計畫。使用存放的一般計畫時，日誌檔中不會寫入任何記錄。變更此參數的值不需要重新開機。

預設	允許的值	Description
無	無	不會在日誌檔中顯示任何計畫強制執行結果。
	on_error	只有當 QPM 無法使用受管計畫時，才會在日誌檔中顯示計畫強制執行結果。
	全部	在日誌檔中顯示所有計畫強制執行結果，包括成功和失敗。

apg_plan_mgmt.max_databases

指定 Aurora PostgreSQL 資料庫叢集的寫入器執行個體上可使用查詢計劃管理的資料庫數量上限。預設情況下，查詢計劃管理最多可支援 10 個資料庫。如果執行個體上有 10 個以上的資料庫，您可以變更此設定的值。若要瞭解指定執行個體上有多少資料庫，請使用 psql 連線到該執行個體。然後，使用 psql 元命令，\l，以列出資料庫。

變更此參數的值時，您必須重新啟動執行個體，設定才會生效。

預設	允許的值	Description
10	10-2147483647	執行個體上可使用查詢計畫管理的資料庫數量上限。

您可以在自訂資料庫叢集參數群組或自訂資料庫參數群組中設定此參數。

apg_plan_mgmt.max_plans

設定查詢計畫管理器可以在 `apg_plan_mgmt.dba_plans` 檢視中維護的 SQL 陳述式數量上限。對於所有 Aurora PostgreSQL 版本，我們建議將此參數設定為 `10000` 或更高的值。

您可以在自訂資料庫叢集參數群組或自訂資料庫參數群組中設定此參數。變更此參數的值時，您必須重新啟動執行個體，設定才會生效。

預設	允許的值	Description
10000	10-2147483647	可儲存於 `apg_plan_mgmt.dba_plans` 檢視的計劃數目上限。 Aurora PostgreSQL 第 10 版及較早版本的預設值為 1000。

如需詳細資訊，請參閱在 dba_plans 檢視中檢查 Aurora PostgreSQL 查詢計劃。

apg_plan_mgmt.plan_hash_version

指定 plan_hash 計算旨在涵蓋的使用案例。較高版本的 `apg_plan_mgmt.plan_hash_version` 涵蓋了較低版本的所有功能。例如，第 3 版涵蓋了第 2 版支援的使用案例。

變更此參數的值之後必須接著呼叫 `apg_plan_mgmt.validate_plans('update_plan_hash')`。其會更新每個資料庫 (已安裝 apg_plan_mgmt) 中的 plan_hash，以及計畫資料表中的項目。如需詳細資訊，請參閱驗證計劃

預設	允許的值	Description
1	1	預設 plan_hash 計算。
	2	針對多結構描述支援而修改的 plan_hash 計算。

預設	允許的值	Description
	3	針對多結構描述支援和分割資料表支援而修改的 plan_hash 計算。
	4	plan_hash 計算針對平行運算子進行了修改並支援具體化節點。

apg_plan_mgmt.plan_retention_period

指定 apg_plan_mgmt.dba_plans 檢視中保留計劃的天數，經過此日期後，它們會自動刪除。根據預設，計劃自從上次使用以來經過 32 天時就會被刪除 (apg_plan_mgmt.dba_plans 檢視中的 last_used 資料欄)。您可以將此設定變更為任意數字、1 和以上。

變更此參數的值時，您必須重新啟動執行個體，設定才會生效。

預設	允許的值	Description
32	1-2147483647	上次使用計劃後到該計劃被自動刪除的天數上限。

如需詳細資訊，請參閱在 dba_plans 檢視中檢查 Aurora PostgreSQL 查詢計劃。

apg_plan_mgmt.unapproved_plan_execution_threshold

指定一個成本閾值，低於該閾值時，最佳化工具可以使用未經核准的計畫。預設情況下，閾值為 0，因此最佳化工具不會執行未經核准的計畫。將此參數設為非常低的成本閾值 (例如 100)，可以避免普通計畫的計畫執行開銷。您也可以使用計畫管理的被動式方式，將此參數設定為非常大的值，例如 10000000。這可讓最佳化工具使用所有選定的計畫，不會產生強制實施計畫的開銷。但是，當發現錯誤的計畫時，您可以手動將其標記為「拒絕」，以便下次不再使用。

此參數的值代表執行指定計劃的成本預估值。如果「未核准」的計劃低於該預估成本，最佳化處理程式會將其用於 SQL 陳述式。您可以在 dba_plans 檢視中查看擷取的計劃及其狀態 (已核准、未核准)。如需詳細資訊，請參閱 在 dba_plans 檢視中檢查 Aurora PostgreSQL 查詢計劃。

變更此參數的值不需要重新開機。

預設	允許的值	Description
0	0-2147483647	預估計劃成本，低於此成本時，則使用未核准計劃。

如需詳細資訊，請參閱使用 Aurora PostgreSQL 受管計劃。

apg_plan_mgmt.use_plan_baselines

指定最佳化工具應使用已擷取並儲存於 apg_plan_mgmt.dba_plans 檢視的其中一個「已核准」計劃。根據預設，此參數為 off (false)，導致最佳化工具使用其產生的最低成本計畫，而無需進一步評估。開啟這個參數 (將其設定為 true) 會強制最佳化工具從其計劃基準中選擇陳述式的查詢執行計畫。如需詳細資訊，請參閱使用 Aurora PostgreSQL 受管計劃。若要尋找詳細說明此程序的圖片，請參閱最佳化工具如何選擇要執行的計劃。

您可以在自訂資料庫叢集參數群組或自訂資料庫參數群組中設定此參數。變更此參數的值不需要重新開機。

預設	允許的值	Description
false	true	使用 apg_plan_mgmt.dba_plans 中「已核准」、「偏好」或「未核准」的計劃。如果這些計劃都不符合最佳化工具的所有評估標準，則可以使用自己產生的最低成本計劃。如需詳細資訊，請參閱最佳化工具如何選擇要執行的計劃。
	false	使用最佳化工具產生的最低成本計劃。

您可以視需要評估不同擷取計劃的回應時間，並變更計劃狀態。如需詳細資訊，請參閱改善 Aurora PostgreSQL 查詢計劃。

auto_explain.hashes

指定 auto_explain 輸出是否顯示 sql_hash 和 plan_hash。變更此參數的值不需要重新開機。

預設	允許的值	Description
0 (關閉)	0 (關閉)	auto_explain 結果不顯示 sql_hash 和 plan_hash 。
	1 (開啟)	auto_explain 結果顯示 sql_hash 和 plan_hash 。

Aurora PostgreSQL 查詢計劃管理的函數參考

apg_plan_mgmt 延伸提供下列函數。

函數

- apg_plan_mgmt.copy_outline
- apg_plan_mgmt.delete_plan
- apg_plan_mgmt.evolve_plan_baselines
- apg_plan_mgmt.get_explain_plan
- apg_plan_mgmt.plan_last_used
- apg_plan_mgmt.reload
- apg_plan_mgmt.set_plan_enabled
- apg_plan_mgmt.set_plan_status
- apg_plan_mgmt.update_plans_last_used
- apg_plan_mgmt.validate_plans

apg_plan_mgmt.copy_outline

將指定的 SQL 計畫雜湊和計畫大綱複製到目標 SQL 計畫雜湊和大綱，從而覆寫目標的計畫雜湊和大綱。此函數可在 apg_plan_mgmt 2.3 及更高版本中使用。

語法

```
apg_plan_mgmt.copy_outline(
    source_sql_hash,
    source_plan_hash,
    target_sql_hash,
    target_plan_hash,
    force_update_target_plan_hash
)
```

傳回值

複製成功時傳回 0。引發無效輸入的例外狀況。

參數

參數	Description
source_sql_hash	與要複製到目標查詢之 plan_hash 相關聯的 sql_hash ID。

參數	Description
source_plan_hash	要複製到目標查詢的 plan_hash ID。
target_sql_hash	要使用來源計畫雜湊和大綱更新之查詢的 sql_hash 識別符。
target_plan_hash	要使用來源計畫雜湊和大綱更新之查詢的 plan_hash 識別符。
force_update_target_plan_hash	(選用) 即使 target_sql_hash 的來源計劃無法重現，查詢的 target_plan_hash ID 也會更新。設定為 true 時，函數可用於在關聯名稱和資料欄一致的結構描述之間複製計劃。

使用須知

此函數可讓您將使用提示的計畫雜湊與計畫大綱複製到其他類似的陳述式，因此使您無需在其每次出現在目標陳述式中時都使用內嵌提示陳述式。如果更新的目標查詢產生無效的計畫，則此函數會引發錯誤並復原所嘗試的更新。

apg_plan_mgmt.delete_plan

刪除受管計劃。

語法

```
apg_plan_mgmt.delete_plan(
    sql_hash,
    plan_hash
)
```

傳回值

如果成功刪除，則傳回 0，如果刪除失敗，則傳回 -1。

參數

參數	描述
sql_hash	計劃的受管 SQL 陳述式的 sql_hash ID。
plan_hash	受管計劃的 plan_hash ID。

apg_plan_mgmt.evolve_plan_baselines

驗證已核准的計劃是否較快，或查詢最佳化工具所識別為最低成本計劃的計劃是否較快。

語法

```
apg_plan_mgmt.evolve_plan_baselines(
    sql_hash,
    plan_hash,
    min_speedup_factor,
    action
)
```

傳回值

沒有比最佳已核准的計劃更快的計劃數量。

參數

參數	描述
sql_hash	計劃的受管 SQL 陳述式的 sql_hash ID。
plan_hash	受管計劃的 plan_hash ID。使用 NULL 來表示具有相同 sql_hash ID 值的所有計劃。
min_speedup_factor	最小加速因素可以是一個計劃必須比目前已核准計劃中的最佳計劃快，使其能被核准的倍數。這個因素也可以是一個計劃慢到必須被拒絕或停用的倍數。 這是正浮點值。
action	函數執行的動作。有效值如下。大小寫不重要。

參數	描述
	• 'disable' – 停用不符合最小加速因素的每個相符計劃。 • 'approve' – 啟用符合最小加速因素的每個相符計劃，並將其狀態設定為 approved。 • 'reject' – 對於不符合最小加速因素的每個相符計劃，將其狀態設定為 rejected。 • NULL – 函數只傳回由於不符合最小加速因素而不具效能優點的計劃數量。

使用須知

根據規劃時間加上執行時間比最佳已核准計劃更快的程度是否達到一個可設定的因素，而將指定的計劃設為已核准、已拒絕或已停用。action 參數可能設為 'approve' 或 'reject'，以自動核准或拒絕符合效能條件的計劃。或者，可能設為 '' (空字串) 以執行效能試驗並產生報告，但不採取任何動作。

對於最近已執行過 apg_plan_mgmt.evolve_plan_baselines 函數的計劃，您可以避免漫無目標地重新執行該函數。若要這麼做，請將計劃限制為最近建立，但尚未核准的計劃。或者，您可以避免在有最近 apg_plan_mgmt.evolve_plan_baselines 時間戳記的任何已核准計劃上執行 last_verified 函數。

進行效能試驗，相對於基線中的其他計劃，比較每個計劃的規劃時間加上執行時間。在某些情況下，陳述式只有一個計劃，而且已核准此計劃。在此情況下，請比較該計劃與不使用任何計劃時的規劃時間加上執行時間。

每個計劃增加的優點 (或缺點) 都記錄在 apg_plan_mgmt.dba_plans 檢視的 total_time_benefit_ms 欄。當此值為正數時，有可測量的效能優點足以將此計劃納入基線中。

除了收集每個候選計劃的規劃時間和執行時間，last_verified 檢視的 apg_plan_mgmt.dba_plans 欄也會更新為 current_timestamp。last_verified 時間戳記可用來避免在最近已驗證效能的計劃上再次執行此函數。

apg_plan_mgmt.get_explain_plan

產生指定 SQL 陳述式的 EXPLAIN 陳述式文字。

語法

```
apg_plan_mgmt.get_explain_plan(
```

```
    sql_hash,
    plan_hash,
    [explainOptionList]
)
```

傳回值

傳回指定 SQL 陳述式的執行時間統計資料。不使用 explainOptionList 傳回簡單的 EXPLAIN 計劃。

參數

參數	描述
sql_hash	計劃的受管 SQL 陳述式的 sql_hash ID。
plan_hash	受管計劃的 plan_hash ID。
explainOptionList	以逗號區隔的說明選項清單。有效值包括 'analyze' 、'verbose' 、'buffers' 、'hashes' 及 'format json'。如果 explainOptionList 是 NULL 或空字串 ('')，此函數會產生 EXPLAIN 陳述式，無需任何統計資料。

使用須知

對於 explainOptionList，您可以使用任何與 EXPLAIN 陳述式搭配使用相同的選項。Aurora PostgreSQL 最佳化工具會將您提供給 EXPLAIN 陳述式的選項清單連接起來。

apg_plan_mgmt.plan_last_used

從共用記憶體中傳回指定計劃的 last_used 日期。

> **Note**
>
> 共用記憶體中的值始終是資料庫叢集中主要資料庫執行個體上的最新值。該值只會定期清空到 apg_plan_mgmt.dba_plans 檢視的 last_used 欄。

語法

```
apg_plan_mgmt.plan_last_used(
    sql_hash,
    plan_hash
)
```

傳回值

傳回 last_used 日期。

參數

參數	描述
sql_hash	計劃的受管 SQL 陳述式的 sql_hash ID。
plan_hash	受管計劃的 plan_hash ID。

apg_plan_mgmt.reload

從 apg_plan_mgmt.dba_plans 檢視中將計劃重新載入共用記憶體。

語法

```
apg_plan_mgmt.reload()
```

傳回值

無。

參數

無.

使用須知

在下列情況中，呼叫 reload：

- 用來立即重新整理唯讀複本的共用記憶體，而非等待新的計劃傳播到複本。

- 在匯入受管計劃之後使用。

apg_plan_mgmt.set_plan_enabled

啟用或停用受管計劃。

語法

```
apg_plan_mgmt.set_plan_enabled(
    sql_hash,
    plan_hash,
    [true | false]
)
```

傳回值

如果成功設定，則傳回 0，如果設定失敗，則傳回 -1。

參數

參數	描述
sql_hash	計劃的受管 SQL 陳述式的 sql_hash ID。
plan_hash	受管計劃的 plan_hash ID。
enabled	布林值 true 或 false： • 值為 true 會啟用計劃。 • 值為 false 會停用計劃。

apg_plan_mgmt.set_plan_status

將受管理計劃的狀態設定為 Approved、Unapproved、Rejected 或 Preferred。

語法

```
apg_plan_mgmt.set_plan_status(
    sql_hash,
    plan_hash,
    status
)
```

傳回值

如果成功設定，則傳回 0，如果設定失敗，則傳回 -1。

參數

參數	描述
sql_hash	計劃的受管 SQL 陳述式的 sql_hash ID。
plan_hash	受管計劃的 plan_hash ID。
status	具有下列其中一個數值的字串： • 'Approved' • 'Unapproved' • 'Rejected' • 'Preferred' 您使用的案例並不重要，但狀態值在 apg_plan_mgmt.dba_plans 檢視中會設為初始大寫。如需這些值的詳細資訊，請參閱 status中的 Aurora PostgreSQL 相容版本的 apg_plan_mgmt.dba_plans 檢視參考。

apg_plan_mgmt.update_plans_last_used

立即更新具有共用記憶體中存放之 last_used 日期的計劃表。

語法

```
apg_plan_mgmt.update_plans_last_used()
```

傳回值

無。

參數

無.

使用須知

呼叫 update_plans_last_used 以確保針對 dba_plans.last_used 欄的查詢使用最新的資訊。如果 last_used 日期未立即更新，則背景程序會更新具有 last_used 日期的計畫表，預設為每小時一次。

例如，如果具有特定 sql_hash 的陳述式開始緩慢執行，您可以決定自效能回歸開始以來，已執行該陳述式的哪些計劃。若要執行這項操作，請先將共用記憶體中的資料清空到磁碟，讓 last_used 日期是最新的，然後利用效能回歸查詢陳述式的所有 sql_hash 計畫。在查詢中，確保 last_used 日期大於或等於效能回歸開始的日期。此查詢會識別可能負責效能回歸的計劃或計劃集。您可以使用 apg_plan_mgmt.get_explain_plan 與設為 verbose, hashes 的 explainOptionList 搭配。您也可以使用 apg_plan_mgmt.evolve_plan_baselines，來分析計劃和任何可能表現更好的替代計劃。

update_plans_last_used 函數只會影響資料庫叢集的主要資料庫執行個體。

apg_plan_mgmt.validate_plans

驗證最佳化工具仍可重建計劃。最佳化工具會驗證 Approved、Unapproved 和 Preferred 計劃，而不論計劃已啟用或已停用。不會驗證 Rejected 計劃。您可以選擇性使用 apg_plan_mgmt.validate_plans 函數來刪除或停用無效的計劃。

語法

```
apg_plan_mgmt.validate_plans(
    sql_hash,
    plan_hash,
    action)

apg_plan_mgmt.validate_plans(
    action)
```

傳回值

無效計劃的數量。

參數

參數	描述
sql_hash	計劃的受管 SQL 陳述式的 sql_hash ID。
plan_hash	受管計劃的 plan_hash ID。使用 NULL 來表示具有相同 sql_hash ID 值的所有計劃。
action	函數對無效計劃執行的動作。有效字串值如下。大小寫不重要。 • 'disable' – 停用每個無效計劃。 • 'delete' – 刪除每個無效計劃。 • 'update_plan_hash' – 更新無法完全複製的計劃的 plan_hash ID。此外還會透過重寫 SQL 讓您修正計劃。您也可以接著將良好的計劃註冊為原始 SQL 的 Approved 計劃。 • NULL – 函數僅僅傳回無效計劃的數量。不會執行其他動作。 • '' – 空字串會產生訊息，指出有效和無效計劃的數量。 其他任何值視為空字串。

使用須知

使用 validate_plans(action) 形式來驗證整個 apg_plan_mgmt.dba_plans 檢視中所有受管陳述式的所有受管計劃。

使用 validate_plans(sql_hash, plan_hash, action) 形式來針對 plan_hash 指定的受管陳述式，驗證 sql_hash 指定的受管計劃。

使用 validate_plans(sql_hash, NULL, action) 形式來針對 sql_hash 指定的受管陳述式，驗證所有受管計劃。

Aurora PostgreSQL 相容版本的 apg_plan_mgmt.dba_plans 檢視參考

apg_plan_mgmt.dba_plans 檢視中的計劃資訊欄如下。

dba_plans 欄	描述
cardinality_error	預估基數和實際基數之間的誤差度量。基數是計劃將處理的資料表列數。如果基數誤差很大，則計劃越有可能並未達到最佳。此欄由 apg_plan_mgmt.evolve_plan_baselines 函數填入。
compatibility_level	此參數會顯示查詢計劃上次驗證的時間。在 Aurora PostgreSQL 12.19、13.15、14.12、15.7、16.3 版及更新版本中，它會顯示 Aurora 版本編號。對於舊版，它會顯示功能特定的版本編號。 **Note** 將此參數值保留在其預設設定。Aurora PostgreSQL 會自動設定和更新此值。
created_by	計劃的建立使用者 (已驗證) (session_user)。
enabled	表示計劃已啟用或停用的指標。預設會啟用所有計劃。您可以停用計劃以防止最佳化工具使用它們。若要修改此值，請使用 apg_plan_mgmt.set_plan_enabled 函數。
environment_variables	最佳化工具在擷取計劃時覆寫的 PostgreSQL Grand Unified Configuration (GUC) 參數和值。
estimated_startup_cost	在最佳化工具交付資料表的列之前預估的最佳化工具設定成本。
estimated_total_cost	預估交付最終資料表列的最佳化工具成本。
execution_time_benefit_ms	啟用計劃的執行時間利益 (以毫秒為單位)。此欄由 apg_plan_mgmt.evolve_plan_baselines 函數填入。
execution_time_ms	計劃執行的預估時間 (以毫秒為單位)。此欄由 apg_plan_mgmt.evolve_plan_baselines 函數填入。
has_side_effects	此值表示 SQL 陳述式是包含 VOLATILE 函數的資料操作語言 (DML) 陳述式或 SELECT 陳述式。

dba_plans 欄	描述
last_used	每當計劃執行時，或當計劃是查詢最佳化工具的最低成本計劃時，此值會更新為目前日期。此值存放在共用記憶體中，並定期清空到磁碟。若要取得最新的值，請呼叫函數 apg_plan_mgmt.plan_last_used(sql_hash, plan_hash) 來讀取共用記憶體中的日期，而非讀取 last_used 值。如需其他資訊，請參閱 apg_plan_mgmt.plan_retention_period 參數。
last_validated	最近一次以 apg_plan_mgmt.validate_plans 函數或 apg_plan_mgmt.evolve_plan_baselines 函數來確認可重建計劃的日期和時間。
last_verified	最近一次以 apg_plan_mgmt.evolve_plan_baselines 函數來確認計劃是特定參數的表現最佳計劃的日期和時間。
origin	如何以 apg_plan_mgmt.capture_plan_baselines 參數來擷取計劃。有效值包括以下項目： M – 以手動計劃擷取來擷取計劃。 A – 以自動計劃擷取來擷取計劃。
param_list	傳遞給陳述式 (如果這是備妥陳述式) 的參數值。
plan_created	建立計劃的日期和時間。
plan_hash	計劃識別符。plan_hash 與 sql_hash 的組合可唯一地識別特定計劃。
plan_outline	用來重建實際執行計劃的計劃表示法，與資料庫無關。樹狀目錄中的運算子對應於 EXPLAIN 輸出中出現的運算子。
planning_time_ms	執行規劃器的實際時間 (以毫秒為單位)。此欄由 apg_plan_mgmt.evolve_plan_baselines 函數填入。
queryId	由 pg_stat_statements 延伸計算的陳述式雜湊。這依賴物件識別符 (OID)，所以不是穩定或與資料庫無關的識別符。擷取查詢計畫時，若 compute_query_id 為 off，此值將為 0。

dba_plans 欄	描述
sql_hash	SQL 陳述式文字的雜湊值，已去除常值而標準化。
sql_text	SQL 陳述式的完整文字。
status	計劃的狀態，可決定最佳化工具如何使用計劃。有效值如下。 • Approved – 可供最佳化工具選擇執行的有用計劃。最佳化工具會從受管陳述式的核准計劃集 (基線) 之中執行最低成本計劃。若要將計劃重設為已核准，請使用 apg_plan_mgmt.evolve_plan_baselines 函數。 • Unapproved – 已擷取但尚未確認可用的計劃。如需更多詳細資訊，請參閱 評估計劃效能。 • Rejected – 最佳化工具不會使用的計劃。如需更多詳細資訊，請參閱 拒絕或停用較慢的計劃。 • Preferred – 您已決定優先用於受管陳述式的計劃。 如果最佳化工具的最低成本計劃不是已核准或較偏好的計劃，您可以降低計劃強制實施的開銷。若要這麼做，請產生已核准計劃的子集合 Preferred 。當最佳化工具的最低成本不是 Approved 計劃時，則會優先於 Preferred 計劃而選擇 Approved 計劃。 若要將計劃重設為 Preferred ，請使用 apg_plan_mgmt.set_plan_status 函數。
stmt_name	PREPARE 陳述式之內的 SQL 陳述式的名稱。對於未命名的備妥陳述式，此值為空白字串。對於未備妥陳述式，此值為 NULL。
total_time_benefit_ms	啟用此計劃的總時間利益 (以毫秒為單位)。此值會考量計劃時間和執行時間。 如果此值為負數，則啟用此計劃不利。此欄由 apg_plan_mgmt.evolve_plan_baselines 函數填入。

查詢計畫管理中的進階功能

您可以在下面找到進階 Aurora PostgreSQL 查詢計畫管理 (QPM) 功能的相關資訊：

主題

- 擷取複本中的 Aurora PostgreSQL 執行計畫
- 支援資料表分割

擷取複本中的 Aurora PostgreSQL 執行計畫

QPM (查詢計畫管理) 可讓您擷取 Aurora 複本產生的查詢計畫，並將其儲存在 Aurora 資料庫叢集的主要資料庫執行個體上。您可以從所有 Aurora 複本收集查詢計畫，並在主要執行個體的中央永久性資料表中維護一組最佳計畫。然後，您可以在需要時將這些計畫套用至其他複本。這可協助您維護執行計畫的穩定性，並改善資料庫叢集和引擎版本的查詢效能。

主題

- 先決條件
- 管理 Aurora 複本的計畫擷取
- 故障診斷

先決條件

在 Aurora 複本中開啟 **`capture_plan_baselines parameter`** - 將 `capture_plan_baselines` 參數設定為自動或手動，以在 Aurora 複本中擷取計畫。如需更多詳細資訊，請參閱 apg_plan_mgmt.capture_plan_baselines。

安裝 postgres_fdw 擴充功能 - 您必須安裝 `postgres_fdw` 外部資料包裝函式，才能在 Aurora 複本中擷取計畫。在每個資料庫中執行下列命令，以安裝擴充功能。

```
postgres=> CREATE EXTENSION IF NOT EXISTS postgres_fdw;
```

管理 Aurora 複本的計畫擷取

開啟 Aurora 複本的計畫擷取

您必須擁有在 Aurora 複本中建立或移除計畫擷取的 `rds_superuser` 權限。如需有關使用者角色和權限的詳細資訊，請參閱了解 PostgreSQL 角色和權限。

若要擷取計畫，請在寫入器資料庫執行個體中呼叫函數 apg_plan_mgmt.create_replica_plan_capture，如下所示：

```
postgres=> CALL apg_plan_mgmt.create_replica_plan_capture('endpoint', 'password');
```

- 端點：Aurora 全球資料庫寫入器端點或 cluster_endpoint 為 Aurora 複本中的計劃擷取提供容錯移轉支援。

  如需 Aurora 全球資料庫寫入器端點的詳細資訊，請參閱檢視 Amazon Aurora 全域資料庫的端點。

  如需叢集端點的詳細資訊，請參閱 Amazon Aurora 的叢集端點。
- 密碼 - 我們建議您在建立密碼時遵循以下準則以增強安全性：
  - 必須包含至少 8 個字元。
  - 至少須包含一個大寫字母、一個小寫字母和一個數字。
  - 必須至少有一個特殊字元 (?、!、#、<、>、* 等等)。

> **Note**
>
> 如果您變更端點、密碼或連接埠號碼，則必須使用端點和密碼再次執行 `apg_plan_mgmt.create_replica_plan_capture()`，以重新初始化計劃擷取。如果沒有，從 Aurora 複本擷取計畫將會失敗。

關閉 Aurora 複本的計畫擷取

您可以在 Aurora 複本中關閉 `capture_plan_baselines` 參數，方法是在「參數」群組中將其值設定為 `off`。

移除 Aurora 複本的計畫擷取

您可以完全移除 Aurora 複本中的計畫擷取，但在執行之前請務必先確定。若要移除計畫擷取，請呼叫 `apg_plan_mgmt.remove_replica_plan_capture`，如下所示：

```
postgres=> CALL apg_plan_mgmt.remove_replica_plan_capture();
```

您必須再次呼叫 apg_plan_mgmt.create_replica_plan_capture()，以使用端點和密碼在 Aurora 複本中開啟計劃擷取。

故障診斷

接下來，如果 Aurora 複本未按預期擷取計畫，您可以找到疑難排解的概念和解決方法。

- 參數設定 - 檢查 `capture_plan_baselines` 參數是否設定為適當的值以開啟計畫擷取。
- **`postgres_fdw`** 擴充功能已安裝 - 使用以下查詢來檢查 `postgres_fdw` 是否已安裝。

```
postgres=> SELECT * FROM pg_extension WHERE extname = 'postgres_fdw'
```

- 呼叫 create_replica_plan_capture() - 使用下列命令來檢查使用者映射是否存在。否則，請呼叫 `create_replica_plan_capture()` 以初始化功能。

```
postgres=> SELECT * FROM pg_foreign_server WHERE srvname =
 'apg_plan_mgmt_writer_foreign_server';
```

- 端點和連接埠號碼：檢查端點和連接埠號碼是否適用。如果這些值不正確，將不會顯示任何錯誤訊息。

  使用以下命令來驗證是否在 create() 中是否使用了端點，並檢查它駐留在哪個資料庫中：

```
postgres=> SELECT srvoptions FROM pg_foreign_server WHERE srvname =
 'apg_plan_mgmt_writer_foreign_server';
```

- reload() - 在 Aurora 複本中呼叫 apg_plan_mgmt.delete_plan() 後，必須呼叫 apg_plan_mgmt.reload() 才能使刪除函數生效。這樣可確保變更已成功實施。
- 密碼 - 您必須依照上述指南在 create_replica_plan_capture() 中輸入密碼。否則，您將收到錯誤訊息。如需詳細資訊，請參閱 管理 Aurora 複本的計畫擷取。使用符合要求的另一個密碼。
- 跨區域連線 - Aurora 全球資料庫也支援 Aurora 複本中的計畫擷取，其中寫入器執行個體和 Aurora 複本可以位於不同的區域。確保您使用 Aurora 全球資料庫寫入器端點，在容錯移轉或轉換事件後維持連線能力。如需 Aurora 全球資料庫端點的詳細資訊，請參閱檢視 Amazon Aurora 全球資料庫的端點。寫入器執行個體和跨區域複本必須能夠使用 VPC 對等互連進行通訊。如需詳細資訊，請參閱 VPC 對等互連。如果發生跨區域容錯移轉，您必須將端點重新設定為新的主要資料庫叢集端點。

> **Note**
>
> 使用叢集端點 (而非 Aurora 全球資料庫寫入器端點) 時，您需要在執行全域容錯移轉或轉換操作之後更新叢集端點。

支援資料表分割

Aurora PostgreSQL 查詢計劃管理 (QPM) 支援下列版本中的宣告式資料表分割：

- 15.3 版和更新的 15 版本
- 14.8 版和更新的 14 版本
- 13.11 版和更新的 13 版本

如需詳細資訊，請參閱資料表分割。

主題

- 設定資料表分割
- 擷取資料表分割的計畫
- 強制執行資料表分割計畫
- 命名慣例

設定資料表分割

若要在 Aurora PostgreSQL QPM 中設定資料表分割，請執行以下操作：

1. 在資料庫叢集參數群組中將 `apg_plan_mgmt.plan_hash_version` 設為 3 或以上。
2. 導覽至使用查詢計畫管理且在 `apg_plan_mgmt.dba_plans` 檢視中具有項目的資料庫。
3. 呼叫 `apg_plan_mgmt.validate_plans('update_plan_hash')` 以更新計畫資料表中的 `plan_hash` 值。
4. 針對已啟用查詢計畫管理且在 `apg_plan_mgmt.dba_plans` 檢視中具有項目的所有資料庫重複步驟 2-3。

如需這些參數的詳細資訊，請參閱 Aurora PostgreSQL 查詢計劃管理的參數參考。

擷取資料表分割的計畫

在 QPM 中，不同的計畫依其 `plan_hash` 值區分。若要了解 `plan_hash` 如何變更，你必須首先了解類似的計畫種類。

在附加節點層級累積的存取方法、去除數字索引名稱和去除數字分割區名稱，其組合必須始終如一，才能將計畫視為相同。計畫中存取的特定分割區並不重要。在下列範例中，會建立具有 4 個分割區的資料表 `tbl_a`。

```
postgres=>create table tbl_a(i int, j int, k int, l int, m int) partition by range(i);
CREATE TABLE
postgres=>create table tbl_a1 partition of tbl_a for values from (0) to (1000);
CREATE TABLE
postgres=>create table tbl_a2 partition of tbl_a for values from (1001) to (2000);
CREATE TABLE
postgres=>create table tbl_a3 partition of tbl_a for values from (2001) to (3000);
CREATE TABLE
postgres=>create table tbl_a4 partition of tbl_a for values from (3001) to (4000);
CREATE TABLE
postgres=>create index t_i on tbl_a using btree (i);
CREATE INDEX
postgres=>create index t_j on tbl_a using btree (j);
CREATE INDEX
postgres=>create index t_k on tbl_a using btree (k);
CREATE INDEX
```

下列計畫會視為相同，因為無論查詢查閱的分割區數目為何，都會使用單一掃瞄方法掃描 tbl_a。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 999 and j < 9910 and k > 50;

                        QUERY PLAN
-----------------------------------------------------------------------
Seq Scan on tbl_a1 tbl_a
    Filter: ((i >= 990) AND (i <= 999) AND (j < 9910) AND (k > 50))
SQL Hash: 1553185667, Plan Hash: -694232056
(3 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                        QUERY PLAN
-----------------------------------------------------------------------
Append
    ->  Seq Scan on tbl_a1 tbl_a_1
            Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
    ->  Seq Scan on tbl_a2 tbl_a_2
            Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
    SQL Hash: 1553185667, Plan Hash: -694232056
    (6 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a3 tbl_a_3
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -694232056
(8 rows)
```

下列 3 個計畫也視為相同，因為在父層級，存取方法、去除數字索引名稱和去除數字分割區名稱為 SeqScan tbl_a、IndexScan (i_idx) tbl_a。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                               QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a2_i_idx on tbl_a2 tbl_a_2
         Index Cond: ((i >= 990) AND (i <= 1100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(7 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                               QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
```

```
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(10 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a3 tbl_a_3
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a4_i_idx on tbl_a4 tbl_a_4
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(11 rows)
```

無論子分割區中的順序和出現次數是否不同，存取方法、去除數字索引名稱和去除數字分割區名稱在上述每個計畫的父層級始終如一。

不過，如果滿足以下任何條件，則計畫將視為不同：

- 計畫中使用任何額外的存取方法。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Bitmap Heap Scan on tbl_a3 tbl_a_3
```

```
         Recheck Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a3_i_idx
               Index Cond: ((i >= 990) AND (i <= 2100))
 SQL Hash: 1553185667, Plan Hash: 1134525070
(11 rows)
```

- 計畫中的任何存取方法已不再使用。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 1100 and j < 9910 and k > 50;

                              QUERY PLAN
-----------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -694232056
(6 rows)
```

- 與索引方法相關聯的索引發生變更。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 1100 and j < 9910 and k > 50;

                             QUERY PLAN
-----------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a2_j_idx on tbl_a2 tbl_a_2
         Index Cond: (j < 9910)
         Filter: ((i >= 990) AND (i <= 1100) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993343726
(7 rows)
```

強制執行資料表分割計畫

分割資料表的核准計畫會透過位置關聯強制執行。這些計畫並非分割區特有的，而且可以在原始查詢中所參照計畫以外的分割區上強制執行。這些計畫也讓您可以強制執行查詢，存取與原始核准大綱不同數目的分割區。

例如，如果核准的大綱適用於下列計畫：

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
--------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(10 rows)
```

然後，也可以在參考 2 個、4 個或更多分割區的 SQL 查詢上強制執行此計畫。可從這些案例產生，以進行 2 和 4 分割區存取的可能計畫如下：

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                                QUERY PLAN
--------------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 1100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(8 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a4 tbl_a_4
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(12 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a4_i_idx on tbl_a4 tbl_a_4
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(14 rows)
```

請針對每個分割區考慮另一個具有不同存取方法的核准計畫：

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Bitmap Heap Scan on tbl_a3 tbl_a_3
         Recheck Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a3_i_idx
               Index Cond: ((i >= 990) AND (i <= 2100))
 SQL Hash: 1553185667, Plan Hash: 2032136998
(12 rows)
```

在此情況下，從兩個分割區讀取的任何計畫將無法強制執行。除非來自核准計畫中的所有 (存取方法、索引名稱) 組合都可以使用，否則計畫無法強制執行。例如，下列計劃具有不同的計畫雜湊，而且在這些情況下無法強制執行核准計畫：

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1900 and j < 9910 and k > 50;

                               QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Bitmap Heap Scan on tbl_a1 tbl_a_1
         Recheck Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a1_i_idx
               Index Cond: ((i >= 990) AND (i <= 1900))
   ->  Bitmap Heap Scan on tbl_a2 tbl_a_2
         Recheck Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a2_i_idx
               Index Cond: ((i >= 990) AND (i <= 1900))
  Note: This is not an Approved plan.  No usable Approved plan was found.
  SQL Hash: 1553185667, Plan Hash: -568647260
(13 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1900 and j < 9910 and k > 50;

                              QUERY PLAN
-----------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1900) AND (j < 9910) AND (k > 50))
 Note: This is not an Approved plan.  No usable Approved plan was found.
 SQL Hash: 1553185667, Plan Hash: -496793743
(8 rows)
```

命名慣例

若要讓 QPM 強制執行具有宣告式分割資料表的計劃，您必須遵循父資料表、資料表分割區和索引的特定命名規則：

- 父資料表名稱：這些名稱必須透過字母或特殊字元來區分，而不是只透過數字。例如，對於個別的父資料表，tA，tB 和 tC 是可接受的名稱，而 t1，t2 和 t3 則不是。
- 個別分割區資料表名稱：同一父系的分割區應該僅透過數字彼此區分。例如，可接受的 tA 分割區名稱可以是 tA1、tA2 或 t1A、t2A，或甚至多個數字。

  任何其他差異 (字母、特殊字元) 將不保證計畫強制執行。
- 索引名稱：在分割區資料表階層中，確保所有索引都有唯一的名稱。這表示名稱的非數字部分必須不同。例如，如果您有名為 tA 的分割資料表 (其索引名為 tA_col1_idx1)，則您無法擁有另一個名為 tA_col1_idx2 的索引。不過，您可以有名為 tA_a_col1_idx2 的索引，因為名稱的非數字部分是唯一的。此規則適用於在父資料表和個別分割區資料表上建立的索引。

無法遵守上述命名慣例，可能會導致核准計畫強制執行失敗。下列範例說明此類失敗的強制執行：

```
postgres=>create table t1(i int, j int, k int, l int, m int) partition by range(i);
CREATE TABLE
postgres=>create table t1a partition of t1 for values from (0) to (1000);
CREATE TABLE
postgres=>create table t1b partition of t1 for values from (1001) to (2000);
CREATE TABLE
postgres=>SET apg_plan_mgmt.capture_plan_baselines TO 'manual';
```

```
SET
postgres=>explain (hashes true, costs false) select count(*) from t1 where i > 0;

                            QUERY PLAN
------------------------------------------------------------------------------
 Aggregate
   ->  Append
         ->  Seq Scan on t1a t1_1
               Filter: (i > 0)
         ->  Seq Scan on t1b t1_2
               Filter: (i > 0)
 SQL Hash: -1720232281, Plan Hash: -1010664377
(7 rows)
```

```
postgres=>SET apg_plan_mgmt.use_plan_baselines TO 'on';
SET
postgres=>explain (hashes true, costs false) select count(*) from t1 where i > 1000;

                            QUERY PLAN
------------------------------------------------------------------------------
 Aggregate
   ->  Seq Scan on t1b t1
         Filter: (i > 1000)
 Note: This is not an Approved plan. No usable Approved plan was found.
 SQL Hash: -1720232281, Plan Hash: 335531806
(5 rows)
```

即使兩個計劃看起來可能相同，但由於子資料表的名稱，其 Plan Hash 值也有所不同。資料表名稱因字母字元 (而不只是數字) 的不同，而導致強制執行失敗。

使用擴充功能和外部資料包裝函式

若要將功能擴充到您的 Aurora PostgreSQL 相容版本資料庫叢集，您可以安裝並使用各種「PostgreSQL 擴充功能」。例如，如果您的使用案例要求跨越非常大的資料表進行密集型資料輸入，則可以安裝 pg_partman 擴展功能對資料進行分割，藉此分散工作負載。

> **Note**
>
> 從 Aurora PostgreSQL 14.5 開始，Aurora PostgreSQL 支援 Trusted Language Extensions for PostgreSQL。此功能會實作為延伸模組 pg_tle，您可以將其新增至 Aurora PostgreSQL。透過使用此延伸模組，開發人員可以在安全的環境中建立自己的 PostgreSQL 延伸模組，以簡化設定和組態需求，以及許多針對新延伸模組進行的初步測試。如需詳細資訊，請參閱使用適用於 PostgreSQL 的受信任語言延伸模組。

在某些情況下，您可以將特定模組新增至 Aurora PostgreSQL 資料庫叢集的自訂資料庫叢集參數群組中的 shared_preload_libraries 清單，而不是安裝擴充功能。一般而言，預設資料庫叢集參數群組只會載入 pg_stat_statements，但有數個其他模組可供新增至清單。例如，您可以新增 pg_cron 模組來新增排程功能，如使用 PostgreSQL pg_cron 擴充功能排程維護中所詳述。另一個範例是，您可以載入 auto_explain 模組來記錄查詢執行計劃。若要進一步了解，請參閱 AWS 知識中心的記錄查詢的執行計畫。

提供存取外部資料的擴充功能具體稱為「外部資料包裝函式」(FDW)。例如，oracle_fdw 擴充功能可讓您的 Aurora PostgreSQL 資料庫叢集使用 Oracle 資料庫。

您還可在 rds.allowed_extensions 參數中列出擴充功能，精確指定可在 Aurora PostgreSQL 資料庫執行個體上安裝的擴充功能。如需詳細資訊，請參閱限制安裝 PostgreSQL 擴充功能。

在下文中，您可以找到設定與使用一些可用於 Aurora PostgreSQL 版本的擴充功能、模組和 FDW 的相關資訊。為了簡單起見，這些都被稱為「擴充功能」。如需可與目前可用的 Aurora PostgreSQL 版本搭配使用的擴充功能清單，請參閱《Aurora PostgreSQL 版本資訊》中的 Amazon Aurora PostgreSQL 的擴充功能版本。

- 使用 lo 模組管理大型物件
- 使用 PostGIS 擴充功能管理空間資料
- 使用 pg_partman 擴充功能來管理 PostgreSQL 分割區
- 使用 PostgreSQL pg_cron 擴充功能排程維護

- 使用 PgAudit 記錄資料庫活動
- 使用 pglogical 跨執行個體同步資料
- 使用 oracle_fdw 擴充功能處理 Oracle 資料庫
- 使用 tds_fdw 擴充功能處理 SQL 資料庫

使用 PostgreSQL 的 Amazon Aurora 委派的延伸模組支援

使用 PostgreSQL 的 Amazon Aurora 委派延伸模組支援，您就可以將延伸模組管理委派給不必是 `rds_superuser` 的使用者。系統會透過此委派延伸模組支援，建立名為 `rds_extension` 的新角色，而您必須將此角色指派給使用者以管理其他延伸模組。此角色可以建立、更新和捨棄延伸模組。

您可在 `rds.allowed_extensions` 參數中列出延伸模組，指定可在 Aurora PostgreSQL 資料庫執行個體上安裝的延伸模組。如需詳細資訊，請參閱將 PostgreSQL 延伸模組與 Amazon RDS for PostgreSQL 搭配使用。

您可以使用 `rds.allowed_delegated_extensions` 參數，限制可由具有 `rds_extension` 角色的使用者所管理的可用延伸模組清單。

您可在下列版本中取得委派的延伸模組支援：

- 所有更高版本
- 15.5 版和更新的 15 版本
- 14.10 版和更新的 14 版本
- 13.13 版和更新的 13 版本
- 12.17 版和更新的 12 版本

主題

- 為使用者開啟委派延伸模組支援
- 在 PostgreSQL 的 Aurora 委派延伸模組支援中使用的組態
- 關閉委派延伸模組的支援
- 使用 Amazon Aurora 委派延伸模組支援的優勢
- PostgreSQL 的 Aurora 委派延伸模組支援限制
- 特定延伸模組所需的許可
- 安全考量

- 已停用捨棄延伸模組層疊
- 可使用委派延伸模組支援新增的延伸模組範例

為使用者開啟委派延伸模組支援

您必須執行下列動作，才能將延伸模組支援委派給使用者：

1. 將 **rds_extension** 角色授予使用者：以 rds_superuser 身分連線至資料庫，以及執行下列命令：

```
Postgres => grant rds_extension to user_name;
```

2. 設定可供委派使用者管理的延伸模組清單：rds.allowed_delegated_extensions 可讓您在資料庫叢集參數中使用 rds.allowed_extensions 指定可用延伸模組子集。您可以在下列其中一個層級執行此操作：

   - 在叢集或執行個體參數群組中，透過 AWS 管理主控台 或 API。如需詳細資訊，請參閱Amazon Aurora 的參數群組。
   - 在資料庫層級使用以下命令：

```
alter database database_name set rds.allowed_delegated_extensions =
 'extension_name_1,
                   extension_name_2,...extension_name_n';
```

   - 在使用者層級使用以下命令：

```
alter user user_name set rds.allowed_delegated_extensions = 'extension_name_1,
                   extension_name_2,...extension_name_n';
```

> **Note**
>
> 變更 rds.allowed_delegated_extensions 動態參數後，您就不需要重新啟動資料庫。

3. 允許委派使用者存取在延伸模組建立程序期間建立的物件：某些延伸模組會建立需要授予額外許可的物件，才能讓具有 rds_extension 角色的使用者存取這些物件。rds_superuser 必須向委派的使用者授予對那些物件的存取權。其中一個選項是使用事件觸發，自動將許可授予委派的使用者。

事件觸發範例

如要允許具備 rds_extension 的委派使用者使用延伸模組，而這些延伸模組需要針對延伸模組建立的物件設定許可，您可以自訂下列事件觸發範例，並僅新增您希望委派使用者能夠存取完整功能的延伸模組。系統可以在 template1 (預設範本) 建立此事件觸發，因此從 template1 建立的所有資料庫都會具有該事件觸發。當委派的使用者安裝此延伸模組時，此觸發會針對延伸模組建立的物件自動授予擁有權。

```
CREATE OR REPLACE FUNCTION create_ext()

  RETURNS event_trigger AS $$

DECLARE

  schemaname TEXT;
  databaseowner TEXT;

  r RECORD;

BEGIN

  IF tg_tag = 'CREATE EXTENSION' and current_user != 'rds_superuser' THEN
    RAISE NOTICE 'SECURITY INVOKER';
    RAISE NOTICE 'user: %', current_user;
    FOR r IN SELECT * FROM pg_catalog.pg_event_trigger_ddl_commands()
    LOOP
        CONTINUE WHEN r.command_tag != 'CREATE EXTENSION' OR r.object_type !=
 'extension';

        schemaname = (
            SELECT n.nspname
            FROM pg_catalog.pg_extension AS e
            INNER JOIN pg_catalog.pg_namespace AS n
            ON e.extnamespace = n.oid
            WHERE e.oid = r.objid
        );

        databaseowner = (
            SELECT pg_catalog.pg_get_userbyid(d.datdba)
            FROM pg_catalog.pg_database d
            WHERE d.datname = current_database()
```

```
        );
        RAISE NOTICE 'Record for event trigger %, objid: %,tag: %, current_user: %,
 schema: %, database_owenr: %', r.object_identity, r.objid, tg_tag, current_user,
 schemaname, databaseowner;
        IF r.object_identity = 'address_standardizer_data_us' THEN
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON
 TABLE %I.us_gaz TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON
 TABLE %I.us_lex TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON
 TABLE %I.us_rules TO %I WITH GRANT OPTION;', schemaname, databaseowner);
        ELSIF r.object_identity = 'dict_int' THEN
            EXECUTE pg_catalog.format('ALTER TEXT SEARCH DICTIONARY %I.intdict
 OWNER TO %I;', schemaname, databaseowner);
        ELSIF r.object_identity = 'pg_partman' THEN
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON
 TABLE %I.part_config TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON
 TABLE %I.part_config_sub TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE
 ON TABLE %I.custom_time_partitions TO %I WITH GRANT OPTION;', schemaname,
 databaseowner);
        ELSIF r.object_identity = 'postgis_topology' THEN
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON ALL
 TABLES IN SCHEMA topology TO %I WITH GRANT OPTION;', databaseowner);
            EXECUTE pg_catalog.format('GRANT USAGE, SELECT ON ALL SEQUENCES IN
 SCHEMA topology TO %I WITH GRANT OPTION;', databaseowner);
            EXECUTE pg_catalog.format('GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA
 topology TO %I WITH GRANT OPTION;', databaseowner);
            EXECUTE pg_catalog.format('GRANT USAGE ON SCHEMA topology TO %I WITH
 GRANT OPTION;', databaseowner);
        END IF;
    END LOOP;
  END IF;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE EVENT TRIGGER log_create_ext ON ddl_command_end EXECUTE PROCEDURE
 create_ext();
```

在 PostgreSQL 的 Aurora 委派延伸模組支援中使用的組態

組態名稱	Description	預設值	備註	誰可以修改或授予許可
rds.allowed_delegated_extensions	此參數會限制 rds_extension 角色可在資料庫中管理的延伸模組。其必須是 rds.allowed_extensions 的子集。	空字串	• 根據預設，此參數為空字串，這表示並未將任何延伸模組委派給具有 rds_extension 的使用者。 • 只要使用者具備許可，即可新增任何受支援的延伸模組。若要執行這項操作，請將 rds.allowed_delegated_extensions 參數設定為逗號分隔的延伸模組字串。透過向此參數新增延伸模組清單，您可明確識別具有 rds_extension 角色的使用者可安裝的延伸模組。 • 設為 * 時，表示 rds_allowed_extensions 中列出的所有延伸模組都會委派給具	rds_superuser

組態名稱	Description	預設值	備註	誰可以修改或授予許可
			有 rds_extension 角色的使用者。 若要進一步了解如何設定此參數，請參閱為使用者開啟委派延伸模組支援。	
rds.allowed_extensions	此參數可讓客戶限制 Aurora PostgreSQL 資料庫執行個體中可安裝的延伸模組。如需詳細資訊，請參閱限制安裝 PostgreSQL 延伸模組。	"*"	根據預設，此參數會設為「*」，這表示具有必要權限的使用者都能夠建立 RDS for PostgreSQL 和 Aurora PostgreSQL 上支援的所有延伸模組。 空白表示無法在 Aurora PostgreSQL 資料庫執行個體中安裝任何延伸模組。	管理員

組態名稱	Description	預設值	備註	誰可以修改或授予許可
rds-deleg ated_ nsior ow_dr ascac	此參數會控制具有 rds_extension 的使用者是否能使用層疊選項捨棄延伸模組。	off	根據預設，rds-delegated_extension_allow_drop_cascade 會設定為 off。這表示具有 rds_extension 的使用者無法使用層疊選項捨棄延伸模組。 若要授予該能力，則應將 rds.delegated_extension_allow_drop_cascade 參數設為 on。	rds_superuser

關閉委派延伸模組的支援

部分關閉

委派的使用者無法建立新的延伸模組，但仍可以更新現有的延伸模組。

- 將資料庫叢集參數群組中的 rds.allowed_delegated_extensions 重設為預設值。
- 在資料庫層級使用以下命令：

```
alter database database_name reset rds.allowed_delegated_extensions;
```

- 在使用者層級使用以下命令：

```
alter user user_name reset rds.allowed_delegated_extensions;
```

完全關閉

從使用者撤銷 rds_extension 角色會將使用者還原為標準許可。使用者無法再建立、更新或捨棄延伸模組。

```
postgres => revoke rds_extension from user_name;
```

使用 Amazon Aurora 委派延伸模組支援的優勢

透過使用 PostgreSQL 的 Amazon Aurora 委派延伸模組支援，您就可以將延伸模組管理安全地委派給不具備 rds_superuser 角色的使用者。此功能提供下列優勢：

- 您可以輕鬆地將延伸模組管理委派給所選的使用者。
- 這不需要 rds_superuser 角色。
- 提供支援相同資料庫叢集中不同資料庫不同延伸模組集的能力。

PostgreSQL 的 Aurora 委派延伸模組支援限制

- 在延伸模組建立程序期間建立的物件可能需要額外的權限，延伸模組才能正常運作。

特定延伸模組所需的許可

為了建立、使用或更新下列延伸模組，委派的使用者應具備下列功能、資料表和結構描述的必要權限。

需要擁有權或許可的延伸模組	函式	表格	結構描述	文字搜尋字典	Comment
addre tanda er_da s		us_gaz、us _lex、us_l ex、l.us_rules			
amch	bt_index_ check、bt_ index_par ent_check				
dict_i				intdict	
pg_p n		custom_ti me_partit ions、part _config、p art_config_sub			
pg_st tatem					
Post(	st_tileenvelope	spatial_ref_sys			

需要擁有權或許可的延伸模組	函式	表格	結構描述	文字搜尋字典	Comment
postg aster					
postg opolo		拓撲、層	拓撲		委派使用者必須是資料庫擁有者
log_f	create_fo reign_tab le_for_log_file				
rds_t	role_pass word_encr yption_type				
postg iger_ oder		geocode_s ettings_d efault、ge ocode_settings	tiger		
pg_fr acem	pg_freespace				
pg_vi lity	pg_visibility				

安全考量

請記住，具有 rds_extension 角色的使用者將能夠在其具有連線權限的所有資料庫上管理延伸模組。若需讓委派使用者管理單一資料庫的延伸模組，良好做法是撤銷每個資料庫上公有的所有權限，然後明確將該特定資料庫的連線權限授予委派的使用者。

有多種延伸模組，使用者可透過這些延伸模組從多個資料庫存取資訊。在將這些延伸模組新增至 rds.allowed_delegated_extensions 前，請確定您授予 rds_extension 的使用者具有跨資料庫功能。例如，postgres_fdw 和 dblink 會提供在相同執行個體或遠端執行個體上跨資料庫查詢的功能。log_fdw 會讀取適用於執行個體中所有資料庫的 postgres 引擎日誌檔案，可能包含來自多個資料庫的緩慢查詢或錯誤訊息。pg_cron 會在資料庫執行個體上執行排程背景任務，並可將任務設定為在不同的資料庫中執行。

已停用捨棄延伸模組層疊

具有 rds_extension 角色的使用者是否能捨棄具有層疊選項的延伸模組，由 rds.delegated_extension_allow_drop_cascade 參數所控制。根據預設，rds.delegated_extension_allow_drop_cascade 會設定為 off。如以下查詢所示，這表示具有 rds_extension 角色的使用者不能使用層疊選項捨棄延伸模組。

```
DROP EXTENSION CASCADE;
```

因為這會自動捨棄相依於延伸模組的物件，進而捨棄所有相依於這些物件的物件。嘗試使用層疊選項將導致錯誤。

若要授予該能力，則應將 rds.delegated_extension_allow_drop_cascade 參數設為 on。

變更 rds.delegated_extension_allow_drop_cascade 動態參數不需要重新啟動資料庫。您可以在以下其中一個層級執行此操作：

- 在叢集或執行個體參數群組中，透過 AWS 管理主控台 或 API。
- 在資料庫層級使用下列命令：

```
alter database database_name set rds.delegated_extension_allow_drop_cascade = 'on';
```

- 在使用者層級使用下列命令：

```
alter role tenant_user set rds.delegated_extension_allow_drop_cascade = 'on';
```

可使用委派延伸模組支援新增的延伸模組範例

- rds_tools

```
extension_test_db=> create extension rds_tools;
CREATE EXTENSION
extension_test_db=> SELECT * from rds_tools.role_password_encryption_type() where
 rolname = 'pg_read_server_files';
ERROR: permission denied for function role_password_encryption_type
```

- amcheck

```
extension_test_db=> CREATE TABLE amcheck_test (id int);
CREATE TABLE
extension_test_db=> INSERT INTO amcheck_test VALUES (generate_series(1,100000));
INSERT 0 100000
extension_test_db=> CREATE INDEX amcheck_test_btree_idx ON amcheck_test USING btree
 (id);
CREATE INDEX
extension_test_db=> create extension amcheck;
CREATE EXTENSION
extension_test_db=> SELECT bt_index_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_check
extension_test_db=> SELECT bt_index_parent_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_parent_check
```

- pg_freespacemap

```
extension_test_db=> create extension pg_freespacemap;
CREATE EXTENSION
extension_test_db=> SELECT * FROM pg_freespace('pg_authid');
ERROR: permission denied for function pg_freespace
extension_test_db=> SELECT * FROM pg_freespace('pg_authid',0);
ERROR: permission denied for function pg_freespace
```

- pg_visibility

```
extension_test_db=> create extension pg_visibility;
CREATE EXTENSION
extension_test_db=> select * from pg_visibility('pg_database'::regclass);
ERROR: permission denied for function pg_visibility
```

- postgres_fdw

```
extension_test_db=> create extension postgres_fw;
CREATE EXTENSION
extension_test_db=> create server myserver foreign data wrapper postgres_fdw options
 (host 'foo', dbname 'foodb', port '5432');
ERROR: permission denied for foreign-data wrapper postgres_fdw
```

使用 lo 模組管理大型物件

lo 模組 (擴充功能) 適用於透過 JDBC 或 ODBC 驅動程式使用 PostgreSQL 資料庫的資料庫使用者與開發人員。JDBC 和 ODBC 都期望當對它們的參照變更時，資料庫會處理大型物件的刪除。但是，PostgreSQL 不是這樣運作的。PostgreSQL 不會假設當某個物件其參照變更時應該予以刪除。結果是物件會保留在磁碟上，未被參照。lo 擴充功能包含一個功能，用於在參照變更時視需要觸發該功能以刪除物件。

> **Tip**
>
> 若要判定資料庫是否可以從 lo 擴充功能受益，請使用 `vacuumlo` 公用程式檢查是否有孤立大型物件。若要在不採取任何動作的情況下取得孤立大型物件的計數，請以 `-n` 選項執行該公用程式 (無操作)。若要了解如何操作，請參閱下文中的 vacuumlo utility。

lo 模組適用於 Aurora PostgreSQL 13.7、12.11、11.16、10.21 和更新的次要版本。

若要安裝此模組 (擴充功能)，您需要 `rds_superuser` 權限。安裝 lo 擴充功能會將下列項目新增至資料庫：

- `lo` – 這是一個大型物件 (lo) 資料類型，可用於二進位大型物件 (BLOB) 和其他大型物件。`lo` 資料類型是 `oid` 資料類型的領域。換言之，它是具有選用限制的物件識別碼。如需詳細資訊，請參閱 PostgreSQL 文件中的物件識別碼。簡言之，您可以使用 `lo` 資料類型，來區分保存大型物件參照的資料庫欄與其他物件識別碼 (OID)。
- `lo_manage` – 這是一個函數，您可以在包含大型物件參照的資料表欄上的觸發程序中使用此函數。只要您刪除或修改參照大型物件的值時，觸發程序都會取消該物件 (`lo_unlink`) 與其參照的連結。只有在資料欄是對該大型物件的唯一資料庫參照時，才對該資料欄使用觸發程序。

如需大型物件模組的詳細資訊，請參閱 PostgreSQL 文件中的 lo。

安裝 lo 擴充功能

安裝 lo 擴充功能之前，請確定您已具備 rds_superuser 權限。

安裝擴充功能

1. 使用 psql 連線到 Aurora PostgreSQL 資料庫叢集的主要資料庫執行個體。

   ```
   psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

   出現提示時，輸入您的密碼。psql 用戶端連接並顯示預設管理連接資料庫 postgres=> 作為提示。

2. 安裝擴充功能，如下所示。

   ```
   postgres=> CREATE EXTENSION lo;
   CREATE EXTENSION
   ```

您現在可以使用 lo 資料類型來定義資料表中的資料欄。例如，您可以建立一個資料表 (images)，其中包含點陣影像資料。您可以將 lo 資料類型用於資料欄 raster，如下列建立資料表的範例所示。

```
postgres=> CREATE TABLE images (image_name text, raster lo);
```

使用 lo_Manage 觸發程序函數刪除物件

您可以將 lo_manage 函數用於當更新或刪除 lo 時所要清理 lo 或其他大型物件欄中的觸發程序。

在參照大型物件的資料欄上設定觸發程序

- 執行以下任意一項：
  - 使用引數的資料欄名稱，在每個資料欄上建立 BEFORE UPDATE OR DELETE 觸發程序，以包含對大型物件的唯一參照。

    ```
    postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OR DELETE ON images
        FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
    ```

  - 僅正在更新資料欄時套用觸發程序。

```
postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OF images
    FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
```

lo_manage 觸發程序函數只有在插入或刪除資料欄資料 (視您定義觸發程序的方式而定) 的背景下才會運作。當您執行 DROP 或 TRUNCATE 操作時則不會起作用。這意味著您應先將任何資料表中的物件資料欄刪除,再捨棄資料表，以免產生孤立物件。

例如，假設您想要捨棄包含 images 資料表的資料庫。您如下所示刪除資料欄。

```
postgres=> DELETE FROM images COLUMN raster
```

假設在該資料欄上定義 lo_manage 函數來處理刪除，現在您可以放心地捨棄該資料表。

使用 vacuumlo 移除孤立的大型物件

vacuumlo 公用程式會識別孤立大型物件並從資料庫中移除。此公用程式自 PostgreSQL 9.1.24 起可供使用。如果您的資料庫使用者會例行性地使用大型物件，建議您偶爾執行 vacuumlo 以清理孤立大型物件。

在安裝 lo 擴充功能之前，您可以使用 vacuumlo 來評估您的 Aurora PostgreSQL 資料庫叢集是否會受益。若要這麼做，請將 vacuumlo 搭配 -n 選項 (無操作) 使用，以顯示要刪除的內容，如下所示：

```
$ vacuumlo -v -n -h your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com -
p 5433 -U postgres docs-lab-spatial-db
Password:*****
Connected to database "docs-lab-spatial-db"
Test run: no large objects will be removed!
Would remove 0 large objects from database "docs-lab-spatial-db".
```

如輸出結果所示，孤立大型對象不是此特定資料庫的問題。

如需此公用程式的詳細資訊，請參閱 PostgreSQL 文件中的 vacuumlo。

了解 vacuumlo 的運作方式

vacuumlo 命令會從 PostgreSQL 資料庫移除孤立的大型物件 (LO)，而不會影響您的使用者資料表或與之衝突。

命令運作方式如下所示：

1. vacuumlo 首先會建立暫時資料表，其中包含資料庫中大型物件的所有物件 ID (OID)。
2. vacuumlo 接著會掃描資料庫中使用資料類型 oid 或 lo 的每個欄位。如果 vacuumlo 在這些欄位中找到相符的 OID，則會從暫時資料表中移除 OID。vacuumlo 只會檢查特別命名為 oid 或 lo 的欄位，而不是根據這些類型的網域。
3. 暫時資料表中剩餘的項目代表孤立的 LO，vacuumlo 接著會將其安全地移除。

提升 **vacuumlo** 效能

您也許可以使用 -l 選項增加批次大小，藉此改善 vacuumlo 的效能。這可讓 vacuumlo 一次處理更多 LO。

如果您的系統有足夠的記憶體，而且您可以將暫時資料表完全容納在記憶體中，在資料庫層級增加 temp_buffers 設定可能會改善效能。這可讓資料表完全位於記憶體中，進而增強整體效能。

下列查詢會預估暫時資料表的大小：

```
SELECT
    pg_size_pretty(SUM(pg_column_size(oid))) estimated_lo_temp_table_size
FROM
    pg_largeobject_metadata;
```

大型物件的考量事項

以下是使用大型物件時需要注意的一些重要考量事項：

- Vacuumlo 是唯一的解決方案，因為目前沒有其他方法可以移除孤立的 LO。
- 使用複寫技術 AWS DMS 之 pglogical、原生邏輯複寫和 等工具不支援複寫大型物件。
- 設計資料庫結構描述時，請盡可能避免使用大型物件，並考慮改用像是 bytea 的其他資料類型。
- 至少每週定期執行 vacuumlo 一次，以防止孤立的 LO 發生問題。
- 在存放大型物件的資料表上使用具有 lo_manage 函數的觸發程序，以協助防止建立孤立的 LO。

使用 PostGIS 擴充功能管理空間資料

PostGIS 是 PostgreSQL 的擴充功能，可用於儲存和管理空間資訊。若要進一步了解 PostGIS，請參閱 PostGIS.net。

從 10.5 版開始，PostgreSQL 即支援 PostGIS 用於處理 Mapbox向量圖標資料的 libprotobuf 1.3.0 程式庫。

設定 PostGIS 擴充功能需要 rds_superuser 權限。我們建議您建立一使用者 (角色) 來管理 PostGIS 擴充功能及您的空間資料。PostGIS 擴充功能及其相關元件會為 PostgreSQL 新增數千個函數。若這對您的使用案例有意義，請考慮在其自己的結構描述中建立 PostGIS 擴充功能。下列範例會顯示如何在其自己的資料庫中安裝擴充功能，但這並非必要。

主題

- 步驟 1：建立使用者 (角色) 來管理 PostGIS 擴充功能
- 步驟 2：載入 PostGIS 擴充功能
- 步驟 3：轉移延伸模組結構描述的擁有權
- 步驟 4：轉移 PostGIS 物件的擁有權
- 步驟 5：測試擴充功能
- 步驟 6：升級 PostGIS 擴充功能
- PostGIS 擴充功能版本
- 將 PostGIS 2 升級到 PostGIS 3

步驟 1：建立使用者 (角色) 來管理 PostGIS 擴充功能

首先，以具有 rds_superuser 權限的使用者身分連線至您的 RDS for PostgreSQL 資料庫執行個體。若您在設定執行個體時保留預設名稱，則以 postgres 身分連線：

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --username=postgres
 --password
```

建立一個單獨的角色 (使用者) 來管理 PostGIS 擴充功能。

```
postgres=>  CREATE ROLE gis_admin LOGIN PASSWORD 'change_me';
CREATE ROLE
```

授予此角色 rds_superuser 權限，允許角色安裝擴充功能。

```
postgres=> GRANT rds_superuser TO gis_admin;
GRANT
```

建立用於 PostGIS 成品的資料庫：此為選擇性步驟。或者，您可以在 PostGIS 擴充功能的使用者資料庫中建立結構描述，但這也並非必要。

```
postgres=> CREATE DATABASE lab_gis;
CREATE DATABASE
```

授予 gis_admin 在 lab_gis 資料庫上的所有權限。

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_gis TO gis_admin;
GRANT
```

退出工作階段，並以 gis_admin 身分重新連線至您的 RDS for PostgreSQL 資料庫執行個體。

```
postgres=> psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=gis_admin --password --dbname=lab_gis
Password for user gis_admin:...
lab_gis=>
```

按照後續步驟中的詳細說明，繼續設定擴充功能。

步驟 2：載入 PostGIS 擴充功能

PostGIS 擴充功能包含數個共同運作的相關擴充功能，以提供地理空間的功能。根據您的使用案例，可能不需要在此步驟中建立的所有擴充功能。

使用 CREATE EXTENSION 陳述式載入 PostGIS 擴充。

```
CREATE EXTENSION postgis;
CREATE EXTENSION
CREATE EXTENSION postgis_raster;
CREATE EXTENSION
CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION
CREATE EXTENSION postgis_tiger_geocoder;
CREATE EXTENSION
CREATE EXTENSION postgis_topology;
CREATE EXTENSION
CREATE EXTENSION address_standardizer_data_us;
CREATE EXTENSION
```

您可以執行下例中顯示的 SQL 查詢來確認結果，其中列出擴充功能及其擁有者。

```
SELECT n.nspname AS "Name",
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;
List of schemas
     Name     |   Owner
--------------+-----------
 public       | postgres
 tiger        | rdsadmin
 tiger_data   | rdsadmin
 topology     | rdsadmin
(4 rows)
```

步驟 3：轉移延伸模組結構描述的擁有權

使用 ALTER SCHEMA 陳述式將結構描述的擁有權移轉至 gis_admin 角色。

```
ALTER SCHEMA tiger OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA tiger_data OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA topology OWNER TO gis_admin;
ALTER SCHEMA
```

您可執行下列 SQL 查詢，來確認所有權變更。您也可以使用 \dn 中繼命令和 psql 命令列。

```
SELECT n.nspname AS "Name",
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;

       List of schemas
     Name     |     Owner
--------------+---------------
 public       | postgres
 tiger        | gis_admin
 tiger_data   | gis_admin
 topology     | gis_admin
```

```
(4 rows)
```

步驟 4：轉移 PostGIS 物件的擁有權

> **Note**
>
> 請勿變更 PostGIS 函數的擁有權。PostGIS 的適當操作和未來升級需要這些函數來保留原始擁有權。如需 PostGIS 許可的詳細資訊，請參閱 PostgreSQL 安全性。

使用下列函式將 PostGIS 資料表的擁有權移轉至 `gis_admin` 角色。從 psql 提示字元執行下列陳述式來建立函式。

```
CREATE FUNCTION exec(text) returns text language plpgsql volatile AS $f$ BEGIN EXECUTE
 $1; RETURN $1; END; $f$;
CREATE FUNCTION
```

接下來，執行下列查詢來執行 `exec` 函數，該函數會執行陳述式及變更權限。

```
SELECT exec('ALTER TABLE ' || quote_ident(s.nspname) || '.' || quote_ident(s.relname)
 || ' OWNER TO gis_admin;')
  FROM (
    SELECT nspname, relname
    FROM pg_class c JOIN pg_namespace n ON (c.relnamespace = n.oid)
    WHERE nspname in ('tiger','topology') AND
    relkind IN ('r','S','v') ORDER BY relkind = 'S')
s;
```

步驟 5：測試擴充功能

為避免需要指定結構描述名稱，請使用下列命令將 `tiger` 結構描述新增至您的搜尋路徑。

```
SET search_path=public,tiger;
SET
```

使用下列 SELECT 陳述式來測試 `tiger` 結構描述。

```
SELECT address, streetname, streettypeabbrev, zip
```

```
 FROM normalize_address('1 Devonshire Place, Boston, MA 02109') AS na;
address | streetname | streettypeabbrev |  zip
---------+------------+------------------+-------
       1 | Devonshire | Pl               | 02109
(1 row)
```

要進一步了解此擴充功能，請參閱 PostGIS 文件中的 Tiger Geocoder (Tiger 地理編碼器)。

使用下列 SELECT 陳述式來測試存取 topology 結構描述。這樣會呼叫 createtopology 函數，以使用指定的空間參考識別碼 (26986) 和預設公差 (0.5) 註冊新拓撲物件 (my_new_topo)。如需進一步了解，請參閱 PostgreGIS 文件中的建立拓撲。

```
SELECT topology.createtopology('my_new_topo',26986,0.5);
 createtopology
----------------
              1
(1 row)
```

步驟 6：升級 PostGIS 擴充功能

每個新版本的 PostgreSQL 都支援一個或多個與該版本相容的 PostGIS 擴充功能版本。將 PostgreSQL 引擎升級到新版本並不會自動升級 PostGIS 擴充功能。升級 PostgreSQL 引擎之前，您通常會將 PostGIS 升級到目前 PostgreSQL 版本的最新可用版本。如需詳細資訊，請參閱PostGIS 擴充功能版本。

PostgreSQL 引擎升級之後，接著再次將 PostGIS 擴充功能升級為支援新升級之 PostgreSQL 引擎版本的版本。如需升級 PostgreSQL 資料庫引擎的詳細資訊，請參閱 測試執行生產資料庫叢集升級到新主要版本的程序.

您可以隨時在 Aurora PostgreSQL 資料庫叢集上檢查可用的 PostGIS 擴充功能版本更新。若要這麼做，請執行下列命令。PostGIS 2.5.0 和更新版本可使用此功能。

```
SELECT postGIS_extensions_upgrade();
```

如果您的應用程式不支援最新的 PostGIS 版本，您仍然可以安裝主要版本中可用的舊版 PostGIS，如下所示。

```
CREATE EXTENSION postgis VERSION "2.5.5";
```

如果您想要從舊版本升級到特定的 PostGIS 版本，也可以使用以下命令。

```
ALTER EXTENSION postgis UPDATE TO "2.5.5";
```

視您要從哪個版本升級而定，您可能需要再次執行此函數。第一次執行函數的結果會決定是否需要額外的升級函數。例如，這是從 PostGIS 2 升級到 PostGIS 3 的情況。如需詳細資訊，請參閱將 PostGIS 2 升級到 PostGIS 3。

如果您升級此擴充功能以準備 PostgreSQL 引擎的主要版本升級，您可以繼續執行其他初步工作。如需詳細資訊，請參閱 測試執行生產資料庫叢集升級到新主要版本的程序。

PostGIS 擴充功能版本

我們建議您安裝所有擴充功能的版本，例如在《Aurora PostgreSQL 版本備註》中的 Aurora PostgreSQL 相容擴充功能版本所列出的 PostGIS。若要取得您的發行版本中有哪些可用版本，請使用下列命令。

```
SELECT * FROM pg_available_extension_versions WHERE name='postgis';
```

您也可以在 Aurora PostgreSQL 版本備註的以下各節找到版本資訊：

- Aurora PostgreSQL 14 的擴充功能版本
- Aurora PostgreSQL 相容版本 13 的擴充功能版本
- Aurora PostgreSQL 相容版本 12 的擴充功能版本
- Aurora PostgreSQL 相容版本 11 的擴充功能版本
- Aurora PostgreSQL 相容版本 10 的擴充功能版本
- Aurora PostgreSQL 相容版本 9.6 的擴充功能版本

將 PostGIS 2 升級到 PostGIS 3

從 3.0 版開始，PostGIS 點陣函數現在是一個單獨的擴充功能，`postgis_raster`。此擴充功能具有自己的安裝和升級路徑。如此一來，可以從核心 `postgis` 擴充功能移除點陣影像處理所需的數十種函數、資料類型和其他成品。這意味著，如果您的使用案例不需要點陣處理，則不需要安裝 `postgis_raster` 擴充功能。

在以下升級範例中，第一個升級命令會將點陣函數擷取至 `postgis_raster` 擴充功能。接著便需要第二個升級命令將 `postgis_raster` 升級到新版本。

從 PostGIS 2 升級到 PostGIS 3

1. 識別可用於 PostgreSQL 版本的 PostGIS 預設版本，而其中 PostgreSQL 位於您的 Aurora PostgreSQL 資料庫叢集。若要這麼做，請執行下列查詢。

```
SELECT * FROM pg_available_extensions
    WHERE default_version > installed_version;
  name   | default_version | installed_version |                          comment
---------+-----------------+-------------------
+------------------------------------------------------------
 postgis | 3.1.4           | 2.3.7             | PostGIS geometry and geography
 spatial types and functions
(1 row)
```

2. 識別 Aurora PostgreSQL 資料庫叢集的寫入器執行個體上，每個資料庫中安裝的 PostGIS 版本。換句話說，查詢每個使用者資料庫，如下所示。

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
    AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
  Name   | Version | Schema |                            Description
---------+---------+--------
+------------------------------------------------------------------------
 postgis | 2.3.7   | public | PostGIS geometry, geography, and raster spatial types
 and functions
(1 row)
```

預設版本 (PostGIS 3.1.4) 與安裝的版本 (PostGIS 2.3.7) 之間不符，代表您需要升級 PostGIS 擴充功能。

```
ALTER EXTENSION postgis UPDATE;
```

```
ALTER EXTENSION
WARNING: unpackaging raster
WARNING: PostGIS Raster functionality has been unpackaged
```

3. 執行下列查詢，以確認點陣函數現在位於其自己的套件中。

```
SELECT
    probin,
    count(*)
FROM
    pg_proc
WHERE
    probin LIKE '%postgis%'
GROUP BY
    probin;
          probin          | count
--------------------------+-------
 $libdir/rtpostgis-2.3    | 107
 $libdir/postgis-3        | 487
(2 rows)
```

輸出結果顯示版本之間仍然存在差異。PostGIS 函數是第 3 版 (postgis-3)，而點陣函數 (rtpostgis) 是第 2 版 (rtpostgis-2.3)。若要完成升級，請再次執行升級命令，如下所示。

```
postgres=> SELECT postgis_extensions_upgrade();
```

您可以放心忽略警告訊息。再次執行下列查詢以確認升級是否已完成。當 PostGIS 和所有相關的擴充功能均未標記為需要升級時，則升級完成。

```
SELECT postgis_full_version();
```

4. 使用下列查詢來查看已完成的升級程序和個別套裝的擴充功能，並確認其版本是否相符。

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
```

```
        AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
      Name      | Version | Schema |                            Description
----------------+---------+--------
+------------------------------------------------------------------------
 postgis        | 3.1.5   | public | PostGIS geometry, geography, and raster
 spatial types and functions
 postgis_raster | 3.1.5   | public | PostGIS raster types and functions
(2 rows)
```

輸出結果顯示 PostGIS 2 擴充功能已升級到 PostGIS 3，並且 `postgis` 和現在的個別 `postgis_raster` 擴充功能則為 3.1.5 版。

升級完成後，如果您不打算使用點陣函數，您可以按以下方式卸除擴充功能。

```
DROP EXTENSION postgis_raster;
```

使用 pg_partman 擴充功能來管理 PostgreSQL 分割區

PostgreSQL 表格分割區提供了用於高性能處理資料輸入和報告的框架。對需要非常快速輸入大量資料的資料庫使用分割區。分割區還提供了更快的大型表格查詢。分割區有助於維護資料，而不會影響資料庫執行個體，因為它需要較少的輸入/輸出資源。

透過使用分割區，您可以將資料分割為自訂大小的區塊進行處理。例如，您可以分割時間序列資料，例如每小時、每日、每週、每月、每季、年度、自訂或以上任何組合。對於時間序列資料範例，如果您依小時分割資料表，則每個分割區都將包含一小時的資料。如果您依每日分割時間序列資料表，則每個分割區將保存一天的資料，依此類推。分割區索引鍵控制分割區的大小。

當您在分割表格上使用 `INSERT` 或 `UPDATE` SQL 命令時，資料庫引擎會將資料路由至適當的分割區。儲存資料的 PostgreSQL 表格分割區是主表格的子表格。

在資料庫查詢讀取期間，PostgreSQL 最佳化器會檢查查詢的 `WHERE` 子句，如果可能的話，將資料庫掃描導向僅相關的分割區。

從 10 版開始，PostgreSQL 使用宣告式分割來實作資料表分割區。這也被稱為原生 PostgreSQL 分割區。在 PostgreSQL 10 版之前，您已使用觸發器來實作分割區。

PostgreSQL 表格分割區提供下列功能：

- 隨時建立新的分割區。
- 可變的分割區範圍。
- 使用資料定義語言 (DDL) 陳述式的可分離和可重新連接分割區。

  例如，可拆分的分割區對於從主磁碟分割區移除歷史資料，但保留歷史資料以供分析來說十分實用。
- 新的分割區會繼承父資料庫表格屬性，包括以下各項：
  - 索引
  - 主索引鍵，其中必須包含分割區索引鍵資料欄
  - 外部索引鍵
  - 檢查限制
  - 參考
- 建立完整資料表或每個特定分割區的索引。

您無法變更個別分割區的結構描述。不過，您可以變更父資料表格 (例如新增新資料欄)，此表格會傳播到分割區。

主題

- PostgreSQL pg_partman 擴充功能概述
- 啟用 pg_partman 擴充功能
- 使用 create_parent 函數設定分割區
- 使用 run_maintenance_proc 函數來設定分割區維護

PostgreSQL pg_partman 擴充功能概述

您可以使用 PostgreSQL `pg_partman` 擴充功能，以自動化資料表分割區的建立和維護。如需更多一般資訊，請參閱 `pg_partman` 文件中的 PG 分割區管理員。

> **Note**
>
> Aurora PostgreSQL 版本 12.6 及更新版本支援此 `pg_partman` 擴充功能。

您可以使用下列設定來設定 `pg_partman`，而不必手動建立每個分割區：

- 要分割的表格

- 分割區類型
- 分割區索引鍵
- 分割區間隔
- 分割區預先建立與管理選項

建立 PostgreSQL 分割區資料表之後，您可以透過呼叫 create_parent 函數，使用 pg_partman 進行註冊。這樣做會根據您傳遞給函數的參數來建立必要的分割區。

pg_partman 擴充功能還提供 run_maintenance_proc 函數，您可以按排程呼叫它以自動管理分割區。為了確保視需要建立適當的分割區，可以排程此函數定期執行 (例如每小時)。您還可以確保自動捨棄分割區。

啟用 pg_partman 擴充功能

如果您要管理相同 PostgreSQL 資料庫執行個體內多個資料庫的分割區，則必須分別為每個資料庫啟用 pg_partman 擴充功能。若要啟用特定資料庫的 pg_partman 擴充功能，請建立分割區維護結構描述，然後建立 pg_partman 擴充功能，如下所示。

```
CREATE SCHEMA partman;
CREATE EXTENSION pg_partman WITH SCHEMA partman;
```

> **Note**
>
> 若要建立 pg_partman 擴充功能，請確定您具有 rds_superuser 權限。

如果您收到下列錯誤訊息，請將rds_superuser權限授與帳戶或使用您的超級使用者帳戶。

```
ERROR: permission denied to create extension "pg_partman"
HINT: Must be superuser to create this extension.
```

若要授予 rds_superuser 權限，請連線至您的超級使用者帳戶並執行下列命令。

```
GRANT rds_superuser TO user-or-role;
```

以顯示使用 pg_partman 擴充功能舉例，我們使用下面的範例資料庫表和分割區。此資料庫使用以時間戳記為基礎的分割表格。結構描述data_mart包含一個名為events的表格和一個名為created_at的欄。下列設定包含在events表格中：

- 主索引鍵event_id和created_at，必須有用於引導分割區的欄。
- 強制執行ck_valid_operation表格欄值的檢查約束operation。
- 兩個外鍵，其中一個（fk_orga_membership））指向外部表格organization，另一個（fk_parent_event_id）是自引用的外鍵。
- 兩個索引，其中一個（idx_org_id）用於外鍵，另一個（idx_event_type）用於事件類型。

下列 DDL 陳述式會建立這些物件，這些物件會自動包含在每個分割區上。

```
CREATE SCHEMA data_mart;
CREATE TABLE data_mart.organization ( org_id BIGSERIAL,
        org_name TEXT,
        CONSTRAINT pk_organization PRIMARY KEY (org_id)
    );

CREATE TABLE data_mart.events(
        event_id        BIGSERIAL,
        operation       CHAR(1),
        value           FLOAT(24),
        parent_event_id BIGINT,
        event_type      VARCHAR(25),
        org_id          BIGSERIAL,
        created_at      timestamp,
        CONSTRAINT pk_data_mart_event PRIMARY KEY (event_id, created_at),
        CONSTRAINT ck_valid_operation CHECK (operation = 'C' OR operation = 'D'),
        CONSTRAINT fk_orga_membership
            FOREIGN KEY(org_id)
            REFERENCES data_mart.organization (org_id),
        CONSTRAINT fk_parent_event_id
            FOREIGN KEY(parent_event_id, created_at)
            REFERENCES data_mart.events (event_id,created_at)
    ) PARTITION BY RANGE (created_at);

CREATE INDEX idx_org_id     ON  data_mart.events(org_id);
CREATE INDEX idx_event_type ON  data_mart.events(event_type);
```

使用 create_parent 函數設定分割區

啟用 pg_partman 擴充功能之後，可以使用此 create_parent 函數來設定在分割維護結構描述內的分割區。以下範例使用在 events 中建立的 啟用 pg_partman 擴充功能 資料表範例。如下所示呼叫 create_parent 函數。

```
SELECT partman.create_parent(
 p_parent_table => 'data_mart.events',
 p_control      => 'created_at',
 p_type         => 'range',
 p_interval     => '1 day',
 p_premake      => 30);
```

參數如下：

- p_parent_table – 父項分割表格。此表必須已經存在，並且完全符合資格 (包括結構描述)。
- p_control – 分割區依據的欄。資料類型必須是整數或以時間為基礎。
- p_type – 類型可以是 'range' 或 'list'。
- p_interval – 每個分割區的時間間隔或整數範圍。範例值包括 1 day、1 hour 等。
- p_premake – 預先建立以支援新插入的分割區數目。

如需 create_parent 函數的完整描述，請參閱 pg_partman 文件中的建立函數。

使用 run_maintenance_proc 函數來設定分割區維護

您可以執行分割區維護作業以自動建立新的分割區、分離分割區或移除舊的分割區。分割區維護依賴 pg_partman 擴充功能的 run_maintenance_proc 函數和 pg_cron 擴充功能，它們可以啟動內部排程器。pg_cron排程器會自動執行資料庫中定義的 SQL 陳述式、函數和程序。

下列範例使用中建立於events中的啟用 pg_partman 擴充功能表格範例，將分割區維護作業設定為自動執行。作為必要條件，將 pg_cron 新增至資料庫執行個體的參數群組中的 shared_preload_libraries 參數。

```
CREATE EXTENSION pg_cron;

UPDATE partman.part_config
SET infinite_time_partitions = true,
    retention = '3 months',
    retention_keep_table=true
WHERE parent_table = 'data_mart.events';
SELECT cron.schedule('@hourly', $$CALL partman.run_maintenance_proc()$$);
```

您可以在下面找到上述範例的逐步說明：

1. 修改與資料庫執行個體關聯的參數群組，並新增pg_cron至shared_preload_libraries參數值。此變更需要重新啟動資料庫執行個體才能生效。如需詳細資訊，請參閱 修改 Amazon Aurora 中的資料庫參數群組中的參數。
2. 使用具有 CREATE EXTENSION pg_cron; 許可的帳戶執行命令 rds_superuser。這會啟用 pg_cron 擴充功能。如需詳細資訊，請參閱 使用 PostgreSQL pg_cron 擴充功能排程維護。
3. 執行命令 UPDATE partman.part_config 以調整 data_mart.events 資料表的 pg_partman 設定。
4. 執行 SET . . . 命令 以使用這些子句設定 data_mart.events 資料表：
   a. infinite_time_partitions = true, – 將表格設定為能夠沒有任何限制，自動建立新的分割區。
   b. retention = '3 months', – 將表格設定為最多保留三個月。
   c. retention_keep_table=true – 設定資料表，以便在保留期限到期時，資料表不會自動刪除。相反地，比保留期間還舊的分割區只會從父表格分離。
5. 執行 SELECT cron.schedule . . . 命令 進行 pg_cron 函數呼叫。此呼叫定義排程器執行 pg_partman 維護程序 partman.run_maintenance_proc 的頻率。在此範例中，程序會每小時執行一次。

如需 run_maintenance_proc 函數的完整描述，請參閱 pg_partman 文件中的維護函數。

使用 PostgreSQL pg_cron 擴充功能排程維護

您可以使用 PostgreSQL pg_cron 擴充功能，來排程 PostgreSQL 資料庫內的維護命令。如需擴充功能的詳細資訊，請參閱 pg_cron 文件中的什麼是 pg_cron？。

Aurora PostgreSQL 引擎 12.6 版以及更新版本支援 pg_cron 擴充功能。

若要深入了解如何使用 pg_cron，請參閱在 RDS for PostgreSQL 或 Aurora PostgreSQL 相容版本資料庫上使用 pg_cron 排程任務

> **Note**
>
> pg_cron 延伸版本會在 pg_available_extensions 檢視中顯示為兩個數字的版本，例如 1.6。雖然在某些內容中可能會看到三個數字的版本，例如 1.6.4 或 1.6.5，但您必須在執行延伸升級時指定兩個數字的版本。

主題

- 設定 pg_cron 擴充功能
- 授與資料庫使用者使用 pg_cron 的許可
- 排定 pg_cron 任務
- pg_cron 擴充功能的參考

設定 pg_cron 擴充功能

設定 pg_cron 擴充功能，如下所示：

1. 修改與 PostgreSQL 資料庫執行個體關聯的自訂參數群組，並將 pg_cron 新增至 shared_preload_libraries 參數值。

   重新啟動 PostgreSQL 資料庫執行個體，使參數群組的變更生效。若要深入了解如何使用參數群組，請參閱 Amazon Aurora PostgreSQL 參數。

2. 重新啟動 PostgreSQL 資料庫執行個體之後，請使用具有 rds_superuser 許可的帳戶執行下列命令。例如，如果您在為 Aurora PostgreSQL 資料庫叢集 建立 RDS 時使用了預設設定，請以使用者 postgres 身分連接並建立擴充功能。

   ```
   CREATE EXTENSION pg_cron;
   ```

   pg_cron 排程器設定於名為 postgres 的預設 PostgreSQL 資料庫內。pg_cron 物件會在此 postgres 資料庫中建立，而且所有排程動作都會在此資料庫中執行。

3. 您可以使用預設設定，或將任務排程在 PostgreSQL 資料庫執行個體的其他資料庫中執行。若要將任務排程至 PostgreSQL 資料庫執行個體中的其他資料庫，請參閱 為預設資料庫以外的資料庫排程 cron 任務 中的範例。

授與資料庫使用者使用 pg_cron 的許可

安裝 pg_cron 擴充功能需要 rds_superuser 權限。然而，使用 pg_cron 的許可能夠授予 (由 rds_superuser 群組/角色的成員) 其他資料庫使用者，讓他們可以安排自己的任務。建議您只有在能改善生產環境中的作業時，才視需要授予對 cron 結構描述的許可。

若要授予資料庫使用者對 cron 結構描述的許可，請執行以下命令：

```
postgres=> GRANT USAGE ON SCHEMA cron TO db-user;
```

這樣會給予 *db-user* 存取 cron 結構描述的許可，以針對它們有權存取的物件排程 cron 作業。如果資料庫使用者沒有許可，則作業會在將錯誤訊息發佈至 postgresql.log 檔案後失敗，如下所示：

```
2020-12-08 16:41:00 UTC::@:[30647]:ERROR: permission denied for table table-name
2020-12-08 16:41:00 UTC::@:[27071]:LOG: background worker "pg_cron" (PID 30647) exited
 with exit code 1
```

換句話說，請確保被授予 cron 結構描述許可的資料庫使用者也具有計劃排程之物件 (資料表、結構描述等) 的許可。

Cron 作業及其成功或失敗的詳細資訊也會擷取於 cron.job_run_details 資料表。如需詳細資訊，請參閱用於排程工作和擷取狀態的資料表。

排定 pg_cron 任務

以下各節示範如何使用 pg_cron 任務排程各種管理任務。

> **Note**
>
> 建立 pg_cron 工作時，請檢查 max_worker_processes 設定大於數目 cron.max_running_jobs。如果 pg_cron 任務耗盡背景工作者程序，則會失敗。pg_cron 任務的預設數字為 5。如需詳細資訊，請參閱用於管理 pg_cron 擴充功能的參數。

主題

- 正在清理資料表
- 正在清除 pg_cron 歷史記錄資料表
- 僅將錯誤記錄到 postgresql.log 檔案
- 為預設資料庫以外的資料庫排程 cron 任務

正在清理資料表

自動資料清理功能可處理大多數情況下的清理維護。但是，您可能想要在您選定的時間來排定清理特定的資料表。

以下為使用 cron.schedule 函數來設定一個任務，以在每天 22:00 (GMT) 於特定資料表上使用 VACUUM FREEZE 的範例。

```
SELECT cron.schedule('manual vacuum', '0 22 * * *', 'VACUUM FREEZE pgbench_accounts');
 schedule
----------
1
(1 row)
```

執行上述範例之後，您可以如下所示檢查 cron.job_run_details 資料表中的歷史記錄。

```
postgres=> SELECT * FROM cron.job_run_details;
jobid  | runid | job_pid | database | username | command                        |
 status    | return_message | start_time                    | end_time
-------+-------+---------+----------+----------+--------------------------------
+-----------+----------------+-------------------------------
+--------------------------------
 1     | 1     | 3395    | postgres | adminuser| vacuum freeze pgbench_accounts
 | succeeded | VACUUM         | 2020-12-04 21:10:00.050386+00 | 2020-12-04
 21:10:00.072028+00
(1 row)
```

以下是查詢 cron.job_run_details 資料表以查看失敗的任務。

```
postgres=> SELECT * FROM cron.job_run_details WHERE status = 'failed';
jobid | runid | job_pid | database | username | command                        | status
 | return_message                                | start_time                    |
 end_time
------+-------+---------+----------+----------+--------------------------------+--------
+-----------------------------------------------+-------------------------------
+-------------------------------
 5    | 4     | 30339   | postgres | adminuser| vacuum freeze pgbench_account | failed
 | ERROR: relation "pgbench_account" does not exist | 2020-12-04 21:48:00.015145+00 |
 2020-12-04 21:48:00.029567+00
(1 row)
```

如需詳細資訊，請參閱用於排程工作和擷取狀態的資料表。

正在清除 pg_cron 歷史記錄資料表

cron.job_run_details 資料表包含 cron 任務的歷史記錄，它可能會隨著時間的推移而變得非常大。建議您排程清除此資料表的任務。例如，保留一週的項目可能足以進行疑難排解。

下列範例會使用 cron.schedule 函數來排程每天在午夜執行的任務，以清除 cron.job_run_details 資料表。這項任務只保留過去七天。如下所示使用您的 rds_superuser 帳戶來排程任務。

```
SELECT cron.schedule('0 0 * * *', $$DELETE
    FROM cron.job_run_details
    WHERE end_time < now() - interval '7 days'$$);
```

如需詳細資訊，請參閱用於排程工作和擷取狀態的資料表。

僅將錯誤記錄到 postgresql.log 檔案

若要防止寫入至 cron.job_run_details 資料表，請修改與 PostgreSQL 資料庫執行個體關聯的參數群組，並將 cron.log_run 參數設定為關閉。pg_cron 延伸模組不再寫入至資料表，而且只會將錯誤擷取至 postgresql.log 檔案。如需詳細資訊，請參閱修改 Amazon Aurora 中的資料庫參數群組中的參數。

使用下列命令來檢查 cron.log_run 參數的值。

```
postgres=> SHOW cron.log_run;
```

如需詳細資訊，請參閱用於管理 pg_cron 擴充功能的參數。

為預設資料庫以外的資料庫排程 cron 任務

pg_cron 的中繼資料都保留在名為 postgres 的 PostgreSQL 預設資料庫中。由於使用背景工作者來執行維護 cron 任務，因此您可以在 PostgreSQL 資料庫執行個體內的任何資料庫中排程任務：

> **Note**
>
> 只有具有 rds_superuser 角色或 rds_superuser 權限的使用者才能列出資料庫中的所有 Cron 任務。其他使用者在 cron.job 資料表中只能檢視自己的任務。

1. 在 cron 資料庫中，使用 cron.schedule 如常排程任務。

   ```
   postgres=> SELECT cron.schedule('database1 manual vacuum', '29 03 * * *', 'vacuum
    freeze test_table');
   ```

2. 作為使用 rds_superuser 角色的使用者，請為您剛建立的任務更新資料庫欄，以便在 PostgreSQL 資料庫執行個體內的另一個資料庫中執行。

```
postgres=> UPDATE cron.job SET database = 'database1' WHERE jobid = 106;
```

3. 透過查詢 cron.job 資料表進行驗證。

```
postgres=> SELECT * FROM cron.job;
jobid | schedule    | command                          | nodename  | nodeport |
 database | username  | active | jobname
------+-------------+----------------------------------+-----------+----------
+----------+-----------+--------+-------------------------
106   | 29 03 * * * | vacuum freeze test_table       | localhost | 8192     |
 database1| adminuser | t      | database1 manual vacuum
  1   | 59 23 * * * | vacuum freeze pgbench_accounts | localhost | 8192     |
 postgres | adminuser | t      | manual vacuum
(2 rows)
```

> **Note**
>
> 在某些情況下，您可能會新增一個打算在其他資料庫上執行的 cron 任務。在此情況下，任務可能會嘗試在預設資料庫 (postgres) 中執行，然後再更新正確的資料庫資料欄。如果使用者名稱具有許可，則任務會在預設資料庫中成功執行。

pg_cron 擴充功能的參考

您可以使用下列參數、函數和資料表搭配 pg_cron 擴充功能。如需詳細資訊，請參閱 pg_cron 文件中的什麼是 pg_cron？。

主題

- 用於管理 pg_cron 擴充功能的參數
- 函數參考：cron.schedule
- 函數參考：cron.unschedule
- 用於排程工作和擷取狀態的資料表

用於管理 pg_cron 擴充功能的參數

以下是用來控制 pg_cron 擴充功能行為的參數清單。

參數	Description
cron.database_name	在其中保留 pg_cron 中繼資料的資料庫。
cron.host	要連線至 PostgreSQL 的主機名稱。您無法修改此值。
cron.log_run	記錄 job_run_details 資料表執行的每個任務。值為 on 或 off。如需詳細資訊，請參閱 用於排程工作和擷取狀態的資料表。
cron.log_statement	在執行之前記錄所有 cron 陳述式。值為 on 或 off。
cron.max_running_jobs	可同時執行的任務數量上限。
cron.use_background_workers	使用背景工作者而不是用戶端工作階段。您無法修改此值。

使用下列 SQL 命令來顯示這些參數及其值。

```
postgres=> SELECT name, setting, short_desc FROM pg_settings WHERE name LIKE 'cron.%'
 ORDER BY name;
```

函數參考：cron.schedule

這個函數會排程一個 cron 任務。任務最初是在預設 postgres 資料庫中進行排程。該函數會返回表示任務識別符的 bigint 值。若要將任務排程在 PostgreSQL 資料庫執行個體中的其他資料庫中執行，請參閱 為預設資料庫以外的資料庫排程 cron 任務 中的範例。

該函數有兩種語法格式。

語法

```
cron.schedule (job_name,
    schedule,
    command
);
```

```
cron.schedule (schedule,
    command
);
```

參數

參數	描述
job_name	cron 任務的名稱。
schedule	指出 cron 任務排程的文字。格式為標準 cron 格式。
command	要執行的命令文字。

範例

```
postgres=> SELECT cron.schedule ('test','0 10 * * *', 'VACUUM pgbench_history');
 schedule
----------
      145
(1 row)

postgres=> SELECT cron.schedule ('0 15 * * *', 'VACUUM pgbench_accounts');
 schedule
----------
      146
(1 row)
```

函數參考：cron.unschedule

這個函數會刪除 cron 任務。您可以指定 job_name 或 job_id，兩者之一。政策可確保您是移除任務排程的擁有者。該函數會返回一個表示成功或失敗的布林值。

此函數的語法格示如下。

語法

```
cron.unschedule (job_id);
```

```
cron.unschedule (job_name);
```

參數

參數	描述
job_id	排程 cron 任務時，從 cron.schedule 函數傳回的任務識別符。
job_name	使用 cron.schedule 函數排程的 cron 任務的名稱。

範例

```
postgres=> SELECT cron.unschedule(108);
 unschedule
------------
 t
(1 row)

postgres=> SELECT cron.unschedule('test');
 unschedule
------------
 t
(1 row)
```

用於排程工作和擷取狀態的資料表

下表用於排程 cron 任務以及記錄任務完成的方式。

表格	描述
cron.job	包含每個排程任務的中繼資料。大部分與此資料表的互動應透過使用 cron.schedule 和 cron.unschedule 函數來完成。

表格	描述
	Important 我們建議您不要將更新或插入權限直接提供給此資料表。這樣做將允許使用者更新 username 資料欄，以 rds-superuser 身分執行任務。
cron.job_run_details	包含過去排程任務執行的歷史記錄資訊。這對於調查任務執行的狀態、傳回訊息以及開始和結束時間非常有用。 **Note** 若要防止此資料表無限期增長，請定期清除。如需範例，請參閱 正在清除 pg_cron 歷史記錄資料表。

使用 PgAudit 記錄資料庫活動

金融機構、政府機構和許多產業都需要保留「稽核日誌」，以符合法規要求。透過使用 PostgreSQL 稽核擴充功能 (PgAudit) 搭配 Aurora PostgreSQL 資料庫叢集，您可以擷取稽核人員通常需要的詳細記錄，以符合法規要求。例如，您可以設定 pgAudit 擴充功能來追蹤對特定資料庫和資料表所做的變更、記錄進行變更的使用者，以及許多其他詳細資訊。

pgAudit 擴充功能建置在原生 PostgreSQL 記錄基礎結構的功能之上，以更多的詳細資訊擴充日誌訊息。換言之，您可以使用與檢視任何日誌訊息相同的方法來檢視稽核日誌。如需 PostgreSQL 記錄的詳細資訊，請參閱 Aurora PostgreSQL 資料庫日誌檔案。

pgAudit 擴充功能會從日誌中刪減敏感資料，例如純文字密碼。如果您的 Aurora PostgreSQL 資料庫叢集設定為記錄資料操作語言 (DML) 陳述式 (如開啟 Aurora PostgreSQL 資料庫叢集 的查詢記錄中所述)，您可以使用 PostgreSQL 稽核擴充功能避免純文字密碼問題。

您可以非常具體地設定對資料庫執行個體的稽核。您可以稽核所有資料庫和所有使用者。或者，您可以選擇僅稽核某些資料庫、使用者和其他物件。您也可以明確排除特定使用者和資料庫，使其不受稽核。如需更多詳細資訊，請參閱 從稽核記錄中排除使用者或資料庫。

鑑於可以擷取的詳細資訊量，我們建議您，如果的確使用 pgAudit，請監控您的儲存耗用量。

所有可用的 Aurora PostgreSQL 版本都支援 PgAudit 擴充功能。如需 Aurora PostgreSQL 版本支援的 pgSQL 版本清單，請參閱《Aurora PostgreSQL 版本資訊》中的 Amazon Aurora PostgreSQL 的擴充功能版本。

主題

- 設定 pgAudit 擴充功能
- 稽核資料庫物件
- 從稽核記錄中排除使用者或資料庫
- pgAudit 擴充功能的參考

設定 pgAudit 擴充功能

若要在 Aurora PostgreSQL 資料庫叢集上設定 PgAudit 擴充功能，您必須先將 PgAudit 新增至 Aurora PostgreSQL 資料庫叢集的自訂資料庫叢集參數群組上的共用程式庫。如需建立自訂資料庫參數群組的相關資訊，請參閱 Amazon Aurora 的參數群組。接下來，您會安裝 pgAudit 擴充功能。最後，您會指定要稽核的資料庫和物件。本節中的程序展示做法。您可以使用 AWS 管理主控台 或 AWS CLI。

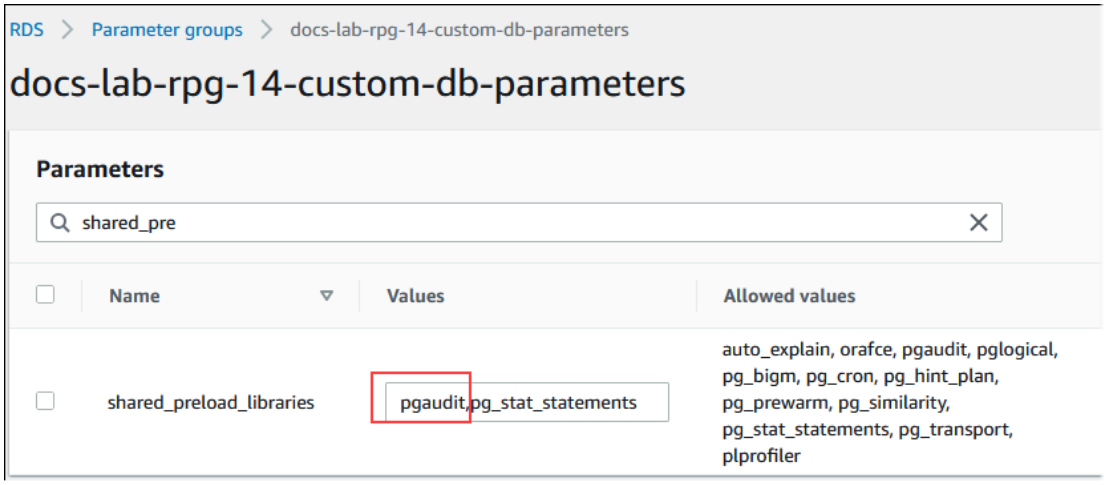
您必須具有做為 `rds_superuser` 角色的許可，才能執行所有這些任務。

以下步驟假設您的 Aurora PostgreSQL 資料庫叢集與自訂資料庫叢集參數群組相關聯。

主控台

設定 pgAudit 擴充功能

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇您的 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。
3. 針對您的 Aurora PostgreSQL 資料庫叢集的寫入器執行個體開啟 Configuration (組態) 索引標籤。在執行個體詳細資訊之間，尋找 Parameter group (參數群組) 連結。
4. 選擇連結以開啟與 Aurora PostgreSQL 資料庫叢集相關聯的自訂參數。
5. 在 Parameters (參數) 搜尋欄位中，輸入 `shared_pre` 以尋找 `shared_preload_libraries` 參數。
6. 選擇 Edit parameters (編輯參數) 以存取屬性值。
7. 在 Values (值) 欄位中，將 `pgaudit` 新增至清單。使用逗號區隔值清單中的項目。



8. 重新啟動 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，以便您對 `shared_preload_libraries` 參數所做的變更生效。
9. 當執行個體可用時，請驗證 pgAudit 是否已初始化。使用 `psql` 連線至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，然後執行下列命令。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

10. 在初始化 pgAudit 之後，您現在可以建立擴充功能。您必須在初始化程式庫之後建立擴充功能，因為 `pgaudit` 擴充功能會安裝事件觸發條件，以稽核資料定義語言 (DDL) 陳述式。

```
CREATE EXTENSION pgaudit;
```

11. 關閉 `psql` 工作階段。

```
labdb=> \q
```

12. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
13. 在清單中尋找 `pgaudit.log` 參數，並設定為適合您使用案例的適當值。例如，將 `pgaudit.log` 參數設定為 `write` (如下圖所示) 會擷取日誌的插入、更新、刪除，以及其他一些類型的變更。

您也可以針對 pgaudit.log 參數選擇下列其中一個值。

- none – 這是預設值。不會記錄任何資料庫變更。
- all – 記錄一切 (read、write、function、role、ddl、misc)。
- ddl – 記錄未包含在 ROLE 類別中的所有資料定義語言 (DDL) 陳述式。
- function – 記錄函數呼叫和 DO 區塊。
- misc – 記錄其他命令，例如 DISCARD、FETCH、CHECKPOINT、VACUUM 和 SET。
- read – 來源為關聯 (例如資料表) 或查詢時，記錄 SELECT 和 COPY。
- role – 記錄與角色和權限相關的陳述式，例如 GRANT、REVOKE、CREATE ROLE、ALTER ROLE 和 DROP ROLE。
- write – 目的地為關聯 (資料表) 時，記錄 INSERT、UPDATE、DELETE、TRUNCATE 和 COPY。

14. 選擇儲存變更。
15. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
16. 在資料庫清單中，選擇您的 Aurora PostgreSQL 資料庫叢集的寫入器執行個體 。

AWS CLI

設定 pgAudit

若要使用 AWS CLI 設定 pgAudit，請呼叫 modify-db-parameter-group 操作，來修改自訂參數群組中的稽核日誌參數，如下列程序所示。

1. 使用下列 AWS CLI 命令，將 pgaudit 新增至 shared_preload_libraries 參數。

```
aws rds modify-db-parameter-group \
```

```
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pgaudit,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. 使用下列 AWS CLI 命令重新啟動 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，以便初始化 pgaudit 程式庫。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. 當執行個體可用時，您可以驗證 pgaudit 是否已初始化。使用 psql 連線至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，然後執行下列命令。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

在初始化 PgAudit 之後，您現在可以建立擴充功能。

```
CREATE EXTENSION pgaudit;
```

4. 關閉 psql 工作階段，以便您可以使用 AWS CLI。

```
labdb=> \q
```

5. 使用下列 AWS CLI 命令，來指定要由工作階段稽核記錄所記錄的陳述式類別。此範例會將 pgaudit.log 參數設定為 write，此參數會擷取日誌的插入、更新和刪除操作。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=pgaudit.log,ParameterValue=write,ApplyMethod=pending-reboot" \
   --region aws-region
```

您也可以針對 pgaudit.log 參數選擇下列其中一個值。

- none – 這是預設值。不會記錄任何資料庫變更。
- all – 記錄一切 (read、write、function、role、ddl、misc)。
- ddl – 記錄未包含在 ROLE 類別中的所有資料定義語言 (DDL) 陳述式。
- function – 記錄函數呼叫和 DO 區塊。
- misc – 記錄其他命令，例如 DISCARD、FETCH、CHECKPOINT、VACUUM 和 SET。
- read – 來源為關聯 (例如資料表) 或查詢時，記錄 SELECT 和 COPY。
- role – 記錄與角色和權限相關的陳述式，例如 GRANT、REVOKE、CREATE ROLE、ALTER ROLE 和 DROP ROLE。
- write – 目的地為關聯 (資料表) 時，記錄 INSERT、UPDATE、DELETE、TRUNCATE 和 COPY。

使用下列 AWS CLI 命令，重新啟動 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。

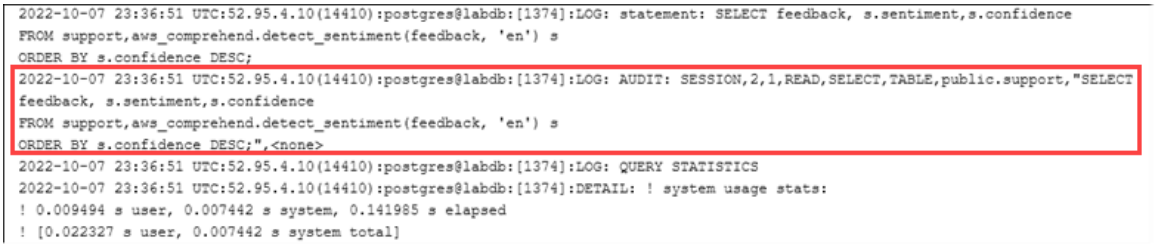
```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

稽核資料庫物件

在您的 Aurora PostgreSQL 資料庫叢集上設定 pgAudit，並針對您的要求進行設定後，會在 PostgreSQL 日誌中擷取更詳細的資訊。例如，雖然預設 PostgreSQL 記錄組態會識別在資料庫資料表中進行變更的日期和時間，但使用 pgAudit 擴充功能時，日誌項目可以包括結構描述、進行變更的使用者，以及其他詳細資訊，視擴充功能參數的設定方式而定。您可以將稽核設定為以下列方式追蹤變更。

- 對於每個工作階段，依使用者。對於工作階段層級，您可以擷取完整的命令文字。
- 對於每個物件，依使用者和資料庫。

在系統上建立 rds_pgaudit 角色，然後將此角色新增至自訂參數群組中的 pgaudit.role 參數時，便會啟用物件稽核功能。根據預設，未設定 pgaudit.role 參數，且唯一允許的值為 rds_pgaudit。下列步驟假設 pgaudit 已初始化，且您已遵循設定 pgAudit 擴充功能中的程序建立 pgaudit 擴充功能。



如此範例所示，「LOG: AUDIT: SESSION」一行提供資料表及其結構描述的相關資訊，以及其他詳細資訊。

設定物件稽核

1. 使用 psql 連線至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。。

```
psql --host=your-instance-name.aws-region.rds.amazonaws.com --port=5432 --
username=postgrespostgres --password --dbname=labdb
```

2. 使用下列命令建立名為 rds_pgaudit 的資料庫角色。

```
labdb=> CREATE ROLE rds_pgaudit;
CREATE ROLE
labdb=>
```

3. 關閉 psql 工作階段。

```
labdb=> \q
```

在接下來的幾個步驟中，使用 AWS CLI 修改自訂參數群組中的稽核日誌參數。

4. 使用下列 AWS CLI 命令，將 pgaudit.role 參數設定為 rds_pgaudit。根據預設，此參數是空的，且 rds_pgaudit 是唯一允許的值。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=pgaudit.role,ParameterValue=rds_pgaudit,ApplyMethod=pending-reboot"
 \
   --region aws-region
```

5. 使用下列 AWS CLI 命令，重新啟動 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，以便您對參數所做的變更生效。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

6. 執行下列命令來確認 pgaudit.role 已設定為 rds_pgaudit。

```
SHOW pgaudit.role;
pgaudit.role
------------------
rds_pgaudit
```

如要測試 pgAudit 記錄功能，您可以執行幾個要稽核的範例命令。例如，您可以執行下列命令。

```
CREATE TABLE t1 (id int);
GRANT SELECT ON t1 TO rds_pgaudit;
SELECT * FROM t1;
id
----
(0 rows)
```

資料庫記錄應包含類似以下的項目。

```
...
2017-06-12 19:09:49 UTC:...:rds_test@postgres:[11701]:LOG: AUDIT:
OBJECT,1,1,READ,SELECT,TABLE,public.t1,select * from t1;
...
```

如需有關檢視日誌的資訊，請參閱 監控 Amazon Aurora 日誌檔案。

若要進一步了解 pgAudit 擴充功能，請參閱 GitHub 上的 pgAudit。

從稽核記錄中排除使用者或資料庫

如Aurora PostgreSQL 資料庫日誌檔案中所述，PostgreSQL 日誌會取用儲存空間。使用 pgAudit 擴充功能會在不同程度上增加日誌中收集的資料量，取決於您追蹤的變更。您可能不需要稽核 Aurora PostgreSQL 資料庫叢集中的每個使用者或資料庫。

若要將對儲存的影響降到最低，並避免不必要地擷取稽核記錄，您可以將使用者和資料庫排除在稽核之外。您也可以在給定的工作階段中變更記錄。下列範例向您展示做法。

> **Note**
>
> 優先處理工作階段層級的參數設定，再處理 Aurora PostgreSQL 資料庫叢集寫入器執行個體的自訂資料庫參數群組中的設定。如果您不想要資料庫使用者略過稽核記錄組態設定，請務必變更其許可。

假設您的 Aurora PostgreSQL 資料庫叢集已設定為稽核所有使用者和資料庫的相同層級活動。然後，您決定不想要稽核使用者 myuser。您可以使用下列 SQL 命令來關閉 myuser 的稽核。

```
ALTER USER myuser SET pgaudit.log TO 'NONE';
```

然後，您可以使用下列查詢來檢查 pgaudit.log 的 user_specific_settings 欄，以確認參數已設定為 NONE。

```
SELECT
    usename AS user_name,
    useconfig AS user_specific_settings
FROM
    pg_user
WHERE
    usename = 'myuser';
```

您會看到如下輸出。

```
 user_name | user_specific_settings
-----------+------------------------
 myuser    | {pgaudit.log=NONE}
(1 row)
```

您可以使用下列命令，在資料庫的工作階段當中關閉特定使用者的記錄。

```
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'none';
```

使用下列查詢，針對特定使用者和資料庫組合檢查 pgaudit.log 的 settings 欄。

```
SELECT
    usename AS "user_name",
```

```
    datname AS "database_name",
    pg_catalog.array_to_string(setconfig, E'\n') AS "settings"
FROM
    pg_catalog.pg_db_role_setting s
    LEFT JOIN pg_catalog.pg_database d ON d.oid = setdatabase
    LEFT JOIN pg_catalog.pg_user r ON r.usesysid = setrole
WHERE
    usename = 'myuser'
    AND datname = 'mydatabase'
ORDER BY
    1,
    2;
```

您會看到類似下列的輸出。

```
  user_name | database_name |     settings
-----------+---------------+------------------
 myuser    | mydatabase    | pgaudit.log=none
(1 row)
```

在關閉 myuser 的稽核之後，您決定不想要追蹤 mydatabase 的變更。您可以使用下列命令來關閉該特定資料庫的稽核。

```
ALTER DATABASE mydatabase SET pgaudit.log to 'NONE';
```

然後，使用下列查詢來檢查 database_specific_settings 欄，以確認 pgaudit.log 已設定為 NONE。

```
SELECT
a.datname AS database_name,
b.setconfig AS database_specific_settings
FROM
pg_database a
FULL JOIN pg_db_role_setting b ON a.oid = b.setdatabase
WHERE
a.datname = 'mydatabase';
```

您會看到如下輸出。

```
 database_name | database_specific_settings
---------------+----------------------------
 mydatabase    | {pgaudit.log=NONE}
```

```
(1 row)
```

若要將設定回復為 myuser 的預設設定，請使用下列命令：

```
ALTER USER myuser RESET pgaudit.log;
```

若要將設定回復為資料庫的預設設定，請使用下列命令。

```
ALTER DATABASE mydatabase RESET pgaudit.log;
```

若要將使用者和資料庫重設為預設設定，請使用下列命令。

```
ALTER USER myuser IN DATABASE mydatabase RESET pgaudit.log;
```

您也可以將 pgaudit.log 設定為 pgaudit.log 參數的其他允許值之一，將特定事件擷取至日誌。如需更多詳細資訊，請參閱 允許的 pgaudit.log 參數設定清單。

```
ALTER USER myuser SET pgaudit.log TO 'read';
ALTER DATABASE mydatabase SET pgaudit.log TO 'function';
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'read,function'
```

pgAudit 擴充功能的參考

您可以變更本節中列出的一或多個參數，針對稽核日誌指定您所需的詳細資訊層級。

控制 pgAudit 行為

您可以變更下表中列出的一或多個參數來控制稽核記錄。

參數	Description
pgaudit.log	指定工作階段稽核記錄將記錄哪些陳述式類別。允許的值包括 ddl、function、misc、read、role、write、none、all。如需詳細資訊，請參閱允許的 pgaudit.log 參數設定清單。
pgaudit.log_catalog	開啟 (設定為 1) 時，如果陳述式中的所有關聯都在 pg_catalog 中，則會將陳述式新增至稽核線索。

參數	Description
pgaudit.log_level	指定要用於日誌項目的日誌層級。允許的值：disabled、debug5、debug4、debug3、debug2、debug1、info、notice、warning、log
pgaudit.log_parameter	開啟 (設定為 1) 時，會在稽核日誌中擷取隨陳述式傳遞的參數。
pgaudit.log_relation	開啟 (設定為 1) 時，工作階段的稽核日誌會為 SELECT 或 DML 陳述式中參考的每個關聯 (TABLE、VIEW 等) 建立個別的日誌項目。
pgaudit.log_statement_once	指定日誌記錄包含的陳述式文字和參數，具有陳述式/子陳述式組合的第一個日誌項目，還是具有每個項目。
pgaudit.role	指定用於物件稽核日誌記錄的主要角色。唯一允許的項目為 rds_pgaudit 。

允許的 **pgaudit.log** 參數設定清單

Value	Description
無	這是預設值。不會記錄任何資料庫變更。
全部	記錄一切 (read、write、function、role、ddl、misc)。
ddl	記錄未包含在 ROLE 類別中的所有資料定義語言 (DDL) 陳述式。
函數	記錄函數呼叫和 DO 區塊。
misc	記錄其他命令，例如 DISCARD、FETCH、CHECKPOINT 、VACUUM 和 SET。
讀取	來源為關聯 (例如資料表) 或查詢時，記錄 SELECT 和 COPY。
role	記錄與角色和權限相關的陳述式，例如 GRANT、REVOKE、CREATE ROLE、ALTER ROLE 和 DROP ROLE。

Value	Description
write	目的地為關聯 (資料表) 時，記錄 INSERT、UPDATE、DELETE、TRUNCATE 和 COPY。

若要使用工作階段稽核功能記錄多個事件類型，請使用逗號分隔清單。若要記錄所有事件類型，請將 pgaudit.log 設定為 ALL。重新啟動資料庫執行個體以套用變更。

您可以透過物件稽核，調整稽核記錄以使用特定關聯。例如，您可以指定您想要針對一或多個資料表上的 READ 操作稽核記錄。

使用 pglogical 跨執行個體同步資料

所有目前可用的 Aurora PostgreSQL 版本都支援 pglogical 延伸模組。pglogical 延伸模組早於 PostgreSQL 在第 10 版中引入的功能，其類似於邏輯複寫功能。如需詳細資訊，請參閱 以 Aurora 進行 PostgreSQL 邏輯複寫的概觀。

pglogical 延伸模組支援在兩個或以上 Aurora PostgreSQL 資料庫叢集之間進行邏輯複寫。它也支援在不同的 PostgreSQL 版本之間，以及在 RDS for PostgreSQL 資料庫執行個體和 Aurora PostgreSQL 資料庫叢集上執行的資料庫之間進行複寫。pglogical 延伸模組會使用發佈-訂閱模型，將資料表和其他物件 (例如序列) 的變更從發佈者複製到訂閱者。它依賴複寫槽，以確保變更從發佈者節點同步到訂閱者節點，其定義如下。

- 發佈者節點是 Aurora PostgreSQL 資料庫叢集，其是要複寫到其他節點的資料來源。發佈者節點定義要在發佈集中複製的資料表。
- 訂閱者節點是 Aurora PostgreSQL 資料庫叢集，其會從發佈者接收 WAL 更新。訂閱者會建立訂閱以連線至發佈者，並取得解碼的 WAL 資料。訂閱者建立訂閱時，系統會在發佈者節點上建立複寫槽。

您可以在以下內容中找到如何設定 pglogical 延伸模組的相關資訊。

主題

- pglogical 延伸模組的需求和限制
- 設定 pglogical 延伸模組
- 針對 Aurora PostgreSQL 資料庫叢集設定邏輯複寫
- 在重大升級之後重新建立邏輯複寫
- 管理 Aurora PostgreSQL 的邏輯複寫槽

- pglogical 延伸模組的參數參考

pglogical 延伸模組的需求和限制

所有目前可用的 Aurora PostgreSQL 版本都支援 `pglogical` 延伸模組。

發佈者節點和訂閱者節點都必須針對邏輯複寫進行設定。

您想要從發佈者複寫到訂閱者的資料表必須具有相同的名稱和相同的結構描述。這些資料表亦須包含相同的資料欄，而且這些資料欄必須使用相同的資料類型。發佈者和訂閱者資料表都必須具有相同的主索引鍵。建議您僅使用 PRIMARY KEY 作為唯一限制條件。

對於 CHECK 限制條件和 NOT NULL 限制條件，訂閱者節點上的資料表與發佈者節點上的資料表相比之下，可以具有更寬鬆的限制條件。

`pglogical` 延伸模組會提供 PostgreSQL (第 10 版及更新版本) 內建邏輯複寫功能不支援的功能，例如雙向複寫。如需詳細資訊，請參閱使用 pglogical 進行 PostgreSQL 雙向複寫。

設定 pglogical 延伸模組

若要在 Aurora PostgreSQL 資料庫叢集上設定 `pglogical` 延伸模組，請將 `pglogical` 新增至下列群組上的共用程式庫：RDS for PostgreSQL 資料庫執行個體的自訂資料庫參數群組。Aurora PostgreSQL 資料庫叢集的自訂資料庫叢集參數群組。您亦需將 `rds.logical_replication` 參數的值設為 1，以開啟邏輯解碼。最後，您可以在資料庫中建立延伸模組。您可以使用 AWS 管理主控台或 AWS CLI 進行這些任務。

您必須具有做為 `rds_superuser` 角色的許可，才能執行這些任務。

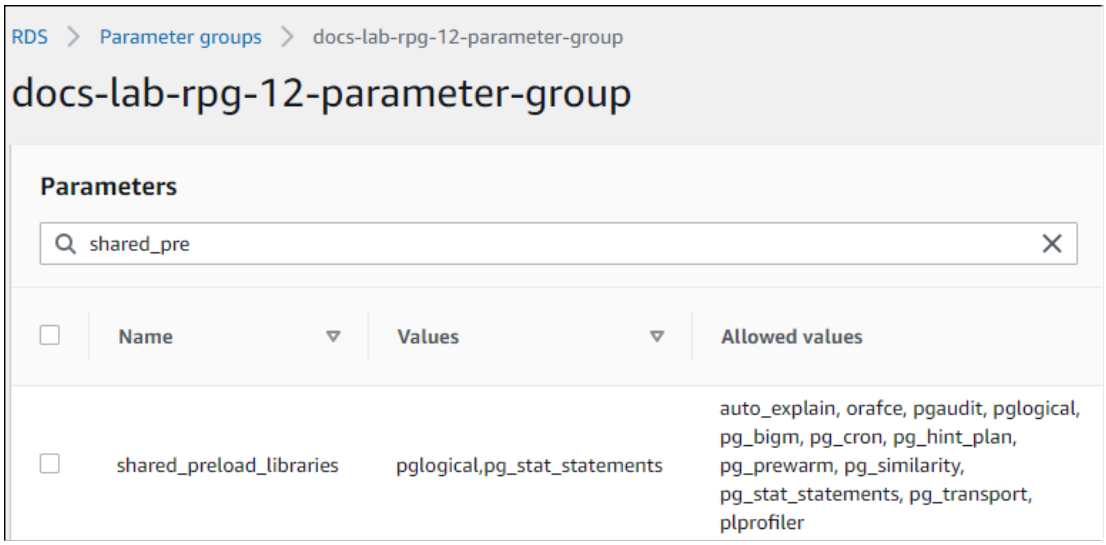
以下步驟假設您的 Aurora PostgreSQL 資料庫叢集與自訂資料庫叢集參數群組相關聯。如需建立自訂資料庫參數群組的相關資訊，請參閱 Amazon Aurora 的參數群組。

主控台

設定 pglogical 延伸模組

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇您的 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。
3. 針對您的 Aurora PostgreSQL 資料庫叢集的寫入器執行個體開啟 Configuration (組態) 索引標籤。在執行個體詳細資訊之間，尋找 Parameter group (參數群組) 連結。

4. 選擇連結以開啟與 Aurora PostgreSQL 資料庫叢集相關聯的自訂參數。
5. 在 Parameters (參數) 搜尋欄位中，輸入 `shared_pre` 以尋找 `shared_preload_libraries` 參數。
6. 選擇 Edit parameters (編輯參數) 以存取屬性值。
7. 在 Values (值) 欄位中，將 `pglogical` 新增至清單。使用逗號區隔值清單中的項目。



8. 尋找 `rds.logical_replication` 參數並將其設為 `1`，以開啟邏輯複寫。
9. 重新啟動 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，讓您的變更生效。
10. 當執行個體可用時，您可以使用 `psql` (或 pgAdmin) 連線至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。。

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

11. 若要驗證 pglogical 是否已初始化，請執行下列命令。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pglogical
(1 row)
```

12. 驗證啟用邏輯解碼的設定，如下所示。

```
SHOW wal_level;
wal_level
-----------
```

```
 logical
(1 row)
```

13. 建立延伸模組，如下所示。

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

14. 選擇儲存變更。
15. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
16. 從資料庫清單中選擇 Aurora PostgreSQL 資料庫叢集的寫入器執行個體以選取它，然後從 Actions (動作) 功能表中選擇 Reboot (重新啟動)。

AWS CLI

設定 pglogical 延伸模組

若要使用 設定 pglogical AWS CLI，您可以呼叫 modify-db-parameter-group 操作來修改自訂參數群組中的特定參數，如下列程序所示。

1. 使用下列 AWS CLI 命令將 `pglogical`新增至 `shared_preload_libraries` 參數。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pglogical,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. 使用下列 AWS CLI 命令將 `rds.logical_replication` 設定為 `1`，以開啟 Aurora PostgreSQL 資料庫叢集寫入器執行個體的邏輯解碼功能。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=rds.logical_replication,ParameterValue=1,ApplyMethod=pending-
reboot" \
   --region aws-region
```

3. 使用以下 AWS CLI 命令重新啟動 Aurora PostgreSQL 資料庫叢集的寫入器執行個體以便初始化 pglogical 程式庫。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

4. 當執行個體可用時，請使用 psql 連線至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。。

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

5. 建立延伸模組，如下所示。

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

6. 使用以下 AWS CLI 命令重新啟動 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

針對 Aurora PostgreSQL 資料庫叢集設定邏輯複寫

下列程序說明如何在兩個 Aurora PostgreSQL 資料庫叢集之間啟動邏輯複寫。這些步驟假設來源 (發佈者) 和目標 (訂閱者) 都已設定 pglogical 延伸模組，如 設定 pglogical 延伸模組 中所述。

> **Note**
>
> 訂閱者節點的 node_name 不能以 rds 開頭。

建立發佈者節點並定義要複製的資料表

這些步驟假設您的 Aurora PostgreSQL 資料庫叢集具有一個內含資料庫的寫入器執行個體，其中包含一或多個您要複寫到另一個節點的資料表。您必須在訂閱者上重新建立來自發佈者的資料表結構，因此必要時先取得資料表結構。您可以使用 psql 中繼命令 \d *tablename*，然後在訂閱者執行個體上建立相同的資料表，以執行該操作。下列程序會在發佈者 (來源) 上建立範例資料表，以供示範之用。

1. 使用 psql 連線至具有資料表的執行個體，您想要將該資料表用作訂閱者的來源。

```
psql --host=source-instance.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

如果您沒有要複寫的現有資料表，則可以建立如下的範例資料表。

a. 使用下列 SQL 陳述式建立範例資料表。

```
CREATE TABLE docs_lab_table (a int PRIMARY KEY);
```

b. 使用下列 SQL 陳述式，將產生的資料填入資料表中。

```
INSERT INTO docs_lab_table VALUES (generate_series(1,5000));
INSERT 0 5000
```

c. 使用下列 SQL 陳述式，驗證資料是否存在於資料表中。

```
SELECT count(*) FROM docs_lab_table;
```

2. 將此 Aurora PostgreSQL 資料庫叢集識別為發佈者節點，如下所示。

```
SELECT pglogical.create_node(
    node_name := 'docs_lab_provider',
    dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
 dbname=labdb');
 create_node
-------------
   3410995529
(1 row)
```

3. 將您要複寫的資料表新增至預設複寫集。如需複寫集的詳細資訊，請參閱 pglogical 文件中的 Replication sets (複寫集)。

```
SELECT pglogical.replication_set_add_table('default', 'docs_lab_table', 'true',
 NULL, NULL);
 replication_set_add_table
  --------------------------
  t
  (1 row)
```

發佈者節點設定完成。您現在可以設定訂閱者節點，從發佈者接收更新。

設定訂閱者節點並建立訂閱來接收更新

這些步驟假設 Aurora PostgreSQL 資料庫叢集已使用 pglogical 延伸模組進行設定。如需詳細資訊，請參閱設定 pglogical 延伸模組。

1. 使用 psql 來連線至您想要從發佈者接收更新的執行個體。

```
psql --host=target-instance.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. 在訂閱者 Aurora PostgreSQL 資料庫叢集上，建立存在於發佈者上的相同資料表。在此範例中，資料表是 docs_lab_table。您可以建立資料表，如下所示。

```
CREATE TABLE docs_lab_table (a int PRIMARY KEY);
```

3. 驗證此資料表是否為空的。

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  0
(1 row)
```

4. 將此 Aurora PostgreSQL 資料庫叢集識別為訂閱者節點，如下所示。

```
SELECT pglogical.create_node(
    node_name := 'docs_lab_target',
    dsn := 'host=target-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=********');
 create_node
-------------
   2182738256
(1 row)
```

5. 建立訂閱。

```
SELECT pglogical.create_subscription(
   subscription_name := 'docs_lab_subscription',
   provider_dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=*******',
   replication_sets := ARRAY['default'],
   synchronize_data := true,
```

```
   forward_origins := '{}' );
 create_subscription
---------------------
1038357190
(1 row)
```

完成此步驟時，會在訂閱者上的資料表中建立來自發佈者上資料表的資料。您可以使用下列 SQL 查詢來驗證是否已發生此情況。

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  5000
(1 row)
```

從此開始，對發佈者上資料表所做的變更會複寫到訂閱者上的資料表。

在重大升級之後重新建立邏輯複寫

在對設為邏輯複寫之發佈者節點的 Aurora PostgreSQL 資料庫叢集執行主要版本升級之前，您必須捨棄所有複寫槽，即使是未作用中的複寫槽也一樣。建議您暫時從發佈者節點轉移資料庫交易、捨棄複寫槽、升級 Aurora PostgreSQL 資料庫叢集，然後重新建立並重新啟動複寫。

複寫槽僅託管於發佈者節點上。邏輯複寫案例中的 Aurora PostgreSQL 訂閱者節點沒有要捨棄的複寫槽。Aurora PostgreSQL 主要版本升級程序支援將訂閱者獨立升級至與發佈者節點不同的新主要 PostgreSQL 版本。不過，升級程序確實會中斷複寫程序，並干擾發佈者節點與訂閱者節點之間的 WAL 資料同步。在升級發佈者、訂閱者或兩者之後，您必須重新建立發佈者與訂閱者之間的邏輯複寫。下列程序說明如何判斷複寫是否已中斷，以及如何解決問題。

判斷邏輯複寫是否已中斷

您可以查詢發佈者節點或訂閱者節點，來判斷複寫程序是否已中斷，如下所示。

檢查發佈者節點

- 使用 `psql` 連線到發佈者節點，然後查詢 `pg_replication_slots` 函數。請注意作用中資料欄中的值。通常，這會傳回 `t` (true)，表明複寫作用中。如果查詢傳回 `f` (false)，表示已停止複寫至訂閱者。

```
SELECT slot_name,plugin,slot_type,active FROM pg_replication_slots;
```

```
                    slot_name               |      plugin      | slot_type | active
-------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_labcb4fa94_docs_lab3de412c | pglogical_output | logical   | f
(1 row)
```

檢查訂閱者節點

在訂閱者節點上，您可以採取三種不同的方式來檢查複寫的狀態。

- 查看訂閱者節點上的 PostgreSQL 日誌，以找出失敗訊息。日誌會以包含結束代碼 1 的訊息識別失敗，如下所示。

```
2022-07-06 16:17:03 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 14610) exited with exit code 1
2022-07-06 16:19:44 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 21783) exited with exit code 1
```

- 查詢 pg_replication_origin 函數。使用 psql 連線至訂閱者節點上的資料庫，然後查詢 pg_replication_origin 函數，如下所示。

```
SELECT * FROM pg_replication_origin;
 roident | roname
---------+--------
(0 rows)
```

  空白結果集表示複寫已中斷。通常，您會看到如下輸出。

```
   roident |                      roname
  ---------+---------------------------------------------------
         1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
  (1 row)
```

- 查詢 pglogical.show_subscription_status 函數，如下列範例所示。

```
SELECT subscription_name,status,slot_name FROM pglogical.show_subscription_status();
     subscription_name | status |              slot_name
---=====----------------+--------+-----------------------------------
 docs_lab_subscription | down   | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

此輸出表明複寫已中斷。其狀態為 down。通常，輸出會將狀態顯示為 replicating。

如果您的邏輯複寫程序已中斷，您可以遵循下列步驟重新建立複寫。

重新建立發佈者與訂閱者節點之間的邏輯複寫

若要重新建立複寫，首先中斷訂閱者與發佈者節點的連線，然後重新建立訂閱，如下列步驟所述。

1. 使用 psql 連線至訂閱者節點，如下所示。

```
psql --host=222222222222.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. 使用 pglogical.alter_subscription_disable 函數停用訂閱。

```
SELECT pglogical.alter_subscription_disable('docs_lab_subscription',true);
 alter_subscription_disable
----------------------------
 t
(1 row)
```

3. 透過查詢 pg_replication_origin 取得發佈者節點的識別符，如下所示。

```
SELECT * FROM pg_replication_origin;
 roident |               roname
---------+--------------------------------------
       1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

4. 使用上一個步驟的回應搭配 pg_replication_origin_create 命令，來指派訂閱可在重新建立時使用的識別符。

```
SELECT pg_replication_origin_create('pgl_labdb_docs_labcb4fa94_docs_lab3de412c');
  pg_replication_origin_create
------------------------------
                             1
(1 row)
```

5. 透過傳送訂閱的名稱來開啟訂閱，其狀態為 true，如下列範例所示。

```
SELECT pglogical.alter_subscription_enable('docs_lab_subscription',true);
```

```
  alter_subscription_enable
--------------------------
 t
(1 row)
```

檢查節點的狀態。其狀態應為 `replicating`，如這個範例所示。

```
SELECT subscription_name,status,slot_name
  FROM pglogical.show_subscription_status();
            subscription_name |   status    |              slot_name
------------------------------+-------------+--------------------------------------
 docs_lab_subscription         | replicating |
 pgl_labdb_docs_lab98f517b_docs_lab3de412c
(1 row)
```

檢查發佈者節點上訂閱者複寫槽的狀態。複寫槽的 `active` 資料欄應傳回 `t` (true)，表示已重新建立複寫。

```
SELECT slot_name,plugin,slot_type,active
  FROM pg_replication_slots;
                    slot_name              |      plugin      | slot_type | active
-------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_lab98f517b_docs_lab3de412c | pglogical_output | logical   | t
(1 row)
```

管理 Aurora PostgreSQL 的邏輯複寫槽

在對充當邏輯複寫案例中發佈者節點的 Aurora PostgreSQL 資料庫叢集寫入器執行個體執行主要版本升級之前，您必須捨棄執行個體上的複寫槽。主要版本升級預先檢查程序會通知您，除非捨棄複寫槽，否則升級無法繼續進行。

若要識別已使用 `pglogical` 延伸模組建立的複寫槽，請登入每個資料庫並取得節點的名稱。查詢訂閱者節點時，您會在輸出中同時取得發佈者和訂閱者節點，如本範例所示。

```
SELECT * FROM pglogical.node;
node_id   |     node_name
------------+------------------
 2182738256 | docs_lab_target
 3410995529 | docs_lab_provider
(2 rows)
```

您可以使用下列查詢，取得有關訂閱的詳細資訊。

```
SELECT sub_name,sub_slot_name,sub_target
  FROM pglogical.subscription;
 sub_name |         sub_slot_name          | sub_target
----------+--------------------------------+------------
  docs_lab_subscription     | pgl_labdb_docs_labcb4fa94_docs_lab3de412c | 2182738256
(1 row)
```

您現在可以捨棄訂閱，如下所示。

```
SELECT pglogical.drop_subscription(subscription_name := 'docs_lab_subscription');
 drop_subscription
-------------------
                 1
(1 row)
```

捨棄訂閱之後，您可以刪除節點。

```
SELECT pglogical.drop_node(node_name := 'docs-lab-subscriber');
 drop_node
-----------
 t
(1 row)
```

您可以驗證節點是否不再存在，如下所示。

```
SELECT * FROM pglogical.node;
 node_id | node_name
---------+-----------
(0 rows)
```

pglogical 延伸模組的參數參考

在資料表中，您可以尋找與 pglogical 延伸模組相關聯的參數。pglogical.conflict_log_level 和 pglogical.conflict_resolution 這類參數用來處理更新衝突。對從發佈者訂閱變更的相同資料表進行本機變更時，可能會出現衝突。在各種案例期間 (例如雙向複寫或從相同發佈者複寫多個訂閱者時)，也會發生衝突。如需詳細資訊，請參閱使用 pglogical 進行 PostgreSQL 雙向複寫。

參數	Description
pglogical.batch_inserts	可能情況下批次插入。依預設不會設定。變更為 '1' 以開啟、變更為 '0' 以關閉。
pglogical.conflict_log_level	設定用於記錄已解決衝突的日誌層級。支援的字串值為 debug5、debug4、debug3、debug2、debug1、info、notice、warning、error、log、fatal、panic。
pglogical.conflict_resolution	設定用來在衝突可解決時解決衝突的方法。支援的字串值為 error、apply_remote、keep_local、last_update_wins、first_update_wins。
pglogical.extra_connection_options	要新增到所有對等節點連線的連線選項。
pglogical.synchronous_commit	pglogical 特定的同步遞交值
pglogical.use_spi	使用 SPI (伺服器程式設計介面) 而不是低階 API 來套用變更。設為 '1' 以開啟、設為 '0' 以關閉。如需 SPI 的詳細資訊，請參閱 PostgreSQL 文件中的 Server Programming Interface (伺服器程式設計介面)。

使用 Amazon Aurora PostgreSQL 支援的外部資料包裝函式

外部資料包裝函式 (FDW) 是一種特定類型的擴充功能，可提供對外部資料的存取。例如，`oracle_fdw` 擴充功能可讓您的 Aurora PostgreSQL 資料庫執行個體 使用 Oracle 資料庫。

在下文中，您可以了解幾個受支援的 PostgreSQL 外部資料包裝函式的資訊。

主題

- 使用 log_fdw 擴充功能存取使用 SQL 的資料庫日誌
- 使用 postgres_fdw 擴充功能存取外部資料
- 使用 mysql_fdw 擴充功能處理 MySQL 資料庫
- 使用 oracle_fdw 擴充功能處理 Oracle 資料庫
- 使用 tds_fdw 擴充功能處理 SQL 資料庫

使用 log_fdw 擴充功能存取使用 SQL 的資料庫日誌

Aurora PostgreSQL 叢集支援 log_fdw 擴充功能，讓您可以使用 SQL 介面存取資料庫引擎日誌。log_fdw 擴充功能推出兩個新函數，可讓您輕鬆為資料庫日誌建立外部資料表：

- list_postgres_log_files – 列出資料庫日誌目錄中的檔案和檔案大小 (以位元組為單位)。
- create_foreign_table_for_log_file(table_name text, server_name text, log_file_name text) – 在目前資料庫中為指定的檔案建立外部資料表。

log_fdw 建立的所有函數皆為 rds_superuser 所擁有。rds_superuser 角色的成員可以將這些函數的存取權授予其他資料庫使用者。

依預設，日誌檔案由 Amazon Aurora 以 stderr (標準錯誤) 格式產生，如 log_destination 參數中所指定。此參數只有兩個選項：stderr 和 csvlog (逗號分隔值，CSV)。若將 csvlog 選項新增至參數，Amazon Aurora 將同時產生 stderr 和 csvlog 日誌。這可能會影響資料庫叢集的儲存容量，因此您需要了解影響日誌處理的其他參數。如需詳細資訊，請參閱設定日誌目標 (stderr、csvlog)。

產生 csvlog 日誌的一個好處為 log_fdw 延伸允許您建置外部資料表，並將資料整齊分割成數個資料欄。為此，您的執行個體需要與自訂資料庫參數群組關聯，則您可變更 log_destination 的設定。如需如何執行作業的資訊，請參閱 Amazon Aurora 的參數群組。

下列範例假設 log_destination 參數包括 cvslog。

使用 log_fdw 擴充功能

1. 安裝 log_fdw 擴充功能。

```
postgres=> CREATE EXTENSION log_fdw;
CREATE EXTENSION
```

2. 建立日誌伺服器做為外部資料包裝函數。

```
postgres=> CREATE SERVER log_server FOREIGN DATA WRAPPER log_fdw;
CREATE SERVER
```

3. 叢日誌檔案清單全選。

```
postgres=> SELECT * FROM list_postgres_log_files() ORDER BY 1;
```

範例回應如下所示。

```
          file_name           | file_size_bytes
------------------------------+-----------------
 postgresql.log.2023-08-09-22.csv |            1111
 postgresql.log.2023-08-09-23.csv |            1172
 postgresql.log.2023-08-10-00.csv |            1744
 postgresql.log.2023-08-10-01.csv |            1102
(4 rows)
```

4. 針對選取的檔案，建立只有單一 'log_entry' 資料欄的資料表。

```
postgres=> SELECT create_foreign_table_for_log_file('my_postgres_error_log',
     'log_server', 'postgresql.log.2023-08-09-22.csv');
```

除了目前存在的資料表之外，回應不提供任何其他詳細資訊。

```
-----------------------------------
(1 row)
```

5. 選取日誌檔案的範例。以下程式碼會擷取日誌時間和錯誤訊息描述。

```
postgres=> SELECT log_time, message FROM my_postgres_error_log ORDER BY 1;
```

範例回應如下所示。

```
             log_time             |                                 message
----------------------------------
+------------------------------------------------------------------------------
Tue Aug 09 15:45:18.172 2023 PDT | ending log output to stderr
Tue Aug 09 15:45:18.175 2023 PDT | database system was interrupted; last known up
 at 2023-08-09 22:43:34 UTC
Tue Aug 09 15:45:18.223 2023 PDT | checkpoint record is at 0/90002E0
Tue Aug 09 15:45:18.223 2023 PDT | redo record is at 0/90002A8; shutdown FALSE
Tue Aug 09 15:45:18.223 2023 PDT | next transaction ID: 0/1879; next OID: 24578
Tue Aug 09 15:45:18.223 2023 PDT | next MultiXactId: 1; next MultiXactOffset: 0
Tue Aug 09 15:45:18.223 2023 PDT | oldest unfrozen transaction ID: 1822, in
 database 1
(7 rows)
```

使用 postgres_fdw 擴充功能存取外部資料

您可以使用 postgres_fdw 擴充功能，存取遠端資料庫伺服器上資料表中的資料。如果您設定來自 PostgreSQL 資料庫執行個體的遠端連線，則也可以存取您的僅供讀取複本。

若要使用 postgres_fdw 來存取遠端資料庫伺服器

1. 安裝 postgres_fdw 擴充功能。

```
CREATE EXTENSION postgres_fdw;
```

2. 使用 CREATE SERVER 建立外部資料伺服器。

```
CREATE SERVER foreign_server
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'xxx.xx.xxx.xx', port '5432', dbname 'foreign_db');
```

3. 建立使用者對應，找出要使用於遠端伺服器的角色。

> **Important**
>
> 若要修訂密碼使其不會在日誌中顯示，請在工作階段層級設定 `log_statement=none`。在參數層級進行設定不會修訂密碼。

```
CREATE USER MAPPING FOR local_user
SERVER foreign_server
OPTIONS (user 'foreign_user', password 'password');
```

4. 建立一個資料表，其對應至遠端伺服器上的資料表。

```
CREATE FOREIGN TABLE foreign_table (
        id integer NOT NULL,
        data text)
SERVER foreign_server
OPTIONS (schema_name 'some_schema', table_name 'some_table');
```

使用 mysql_fdw 擴充功能處理 MySQL 資料庫

若要從 Aurora PostgreSQL 資料庫叢集存取相容於 MySQL 的資料庫，您可以安裝並使用 `mysql_fdw` 擴充功能。此外部資料包裝函數可讓您使用 MySQL、Aurora MySQL、MariaDB 和其他相容於 MySQL 的資料庫的 RDS。從 Aurora PostgreSQL 資料庫叢集到 MySQL 資料庫的連線是在最大努力的基礎上加密的，具體取決於用戶端和伺服器的組態。但是，如有需要，您可以強制執行加密。如需詳細資訊，請參閱搭配此擴充功能使用傳輸中加密。

Amazon Aurora PostgreSQL 版本 15.4、14.9、13.12、12.16 及更高版本支援 `mysql_fdw` 擴充功能。它支援從 RDS for PostgreSQL DB 對相容於 MySQL 的資料庫執行個體上的資料表進行選擇、插入、更新和刪除。

主題

- 設定 Aurora PostgreSQL 資料庫以使用 mysql_fdw 擴充功能
- 範例：使用 Aurora PostgreSQL 中的 Aurora MySQL 資料庫
- 搭配此擴充功能使用傳輸中加密

設定 Aurora PostgreSQL 資料庫以使用 mysql_fdw 擴充功能

在您的 Aurora PostgreSQL 資料庫叢集上設定 `mysql_fdw` 擴充功能包括在資料庫叢集中載入擴充功能，然後建立與 MySQL資料庫執行個體的連線點。針對此任務，您必須有以下關於 MySQL 資料庫執行個體的詳細內容：

- 主機名稱或端點。如果是 Aurora PostgreSQL 資料庫叢集，您可以使用主控台尋找端點。選擇 Connectivity & security (連線和安全) 索引標籤，然後查看「端點和連線埠」區段。
- 連線埠號碼。MySQL 的預設連線埠號為 3306。
- 資料庫的名稱。資料庫識別符。

您也必須提供 MySQL 連線埠 3306 的安全群組或存取控制清單 (ACL) 的存取權限。Aurora PostgreSQL 資料庫叢集和 Aurora MySQL 資料庫叢集 都需要存取連接埠 3306。如果未正確設定存取權限，當您嘗試連線至相容於 MySQL 的資料表時，會看到類似以下的錯誤訊息：

```
ERROR: failed to connect to MySQL: Can't connect to MySQL server on 'hostname.aws-
region.rds.amazonaws.com:3306' (110)
```

在以下程序中，您 (作為 rds_superuser 帳戶) 建立外部伺服器。然後，您將外部伺服器的存取權限授予特定使用者。然後，這些使用者建立自己的映射到適合的 MySQL 使用者帳戶以使用 MySQL 資料庫執行個體。

使用 mysql_fdw 存取 MySQL 資料庫伺服器

1. 使用有 rds_superuser 角色的帳戶連線到您的 PostgreSQL 資料庫執行個體。如果您在建立 Aurora PostgreSQL 資料庫叢集時接受了預設值，則使用者名稱為 postgres，而且您可以使用 psql 命令列工具進行連線，如下所示：

```
psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. 安裝 mysql_fdw 擴充功能，如下所示：

```
postgres=> CREATE EXTENSION mysql_fdw;
CREATE EXTENSION
```

在 Aurora PostgreSQL 資料庫叢集上安裝擴充功能後，您可以設定提供連線至 MySQL 資料庫的外部伺服器。

建立外部伺服器

在 Aurora PostgreSQL 資料庫叢集上執行這些任務。這些步驟假設您以具有 rds_superuser 權限 (例如 postgres) 的使用者進行連線。

1. 在 Aurora PostgreSQL 資料庫叢集中建立外部伺服器：

```
postgres=> CREATE SERVER mysql-db FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'db-
name.111122223333.aws-region.rds.amazonaws.com', port '3306');
CREATE SERVER
```

2. 將適當的使用者存取權限授予外部伺服器。這些使用者應該是非管理員使用者，亦即沒有 rds_superuser 角色。

```
postgres=> GRANT USAGE ON FOREIGN SERVER mysql-db to user1;
GRANT
```

PostgreSQL 使用者透過外部伺服器建立和管理自己的 MySQL 資料庫連線。

範例：使用 Aurora PostgreSQL 中的 Aurora MySQL 資料庫

假設您在 Aurora PostgreSQL 資料庫執行個體上有一個簡單資料表。您的 Aurora PostgreSQL 使用者想要查詢 (SELECT)、INSERT、UPDATE 和 DELETE 該資料表上的項目。假設 `mysql_fdw` 擴充功能已在 RDS for PostgreSQL 資料庫執行個體上建立，詳情如上述程序所述。作為具有 `rds_superuser` 權限的使用者，在您連線到 RDS 的 PostgreSQL 資料庫執行個體後，可繼續執行以下步驟。

1. 在 Aurora PostgreSQL 資料庫執行個體中建立外部伺服器：

```
test=> CREATE SERVER mysqldb FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'your-DB.aws-region.rds.amazonaws.com', port '3306');
CREATE SERVER
```

2. 授與使用量給沒有 `rds_superuser` 許可的使用者，例如 `user1`：

```
test=> GRANT USAGE ON FOREIGN SERVER mysqldb TO user1;
GRANT
```

3. 以 *user1* 身分連線，然後建立一個 MySQL 使用者的映射：

```
test=> CREATE USER MAPPING FOR user1 SERVER mysqldb OPTIONS (username 'myuser',
 password 'mypassword');
CREATE USER MAPPING
```

4. 建立 MySQL 資料表的外部資料表連結。

```
test=> CREATE FOREIGN TABLE mytab (a int, b text) SERVER mysqldb OPTIONS (dbname
 'test', table_name '');
CREATE FOREIGN TABLE
```

5. 對外部資料表執行簡單查詢：

```
test=> SELECT * FROM mytab;
a |   b
---+-------
1 | apple
(1 row)
```

6. 您可以從 MySQL 資料表新增、變更和移除資料。例如：

```
test=> INSERT INTO mytab values (2, 'mango');
```

```
INSERT 0 1
```

再次執行 SELECT 查詢以查看結果：

```
test=> SELECT * FROM mytab ORDER BY 1;
 a |   b
---+-------
1 | apple
2 | mango
(2 rows)
```

搭配此擴充功能使用傳輸中加密

預設情況下，從 Aurora PostgreSQL 到 MySQL 的連線會使用傳輸中加密 (TLS/SSL)。但是，當用戶端和伺服器組態不同時，連線會回退到未加密。您可以在 RDS for MySQL 使用者帳戶上指定 REQUIRE SSL 選項，強制加密所有的對外連線。此方法也適用於 MariaDB 和 Aurora MySQL 使用者帳戶。

當 MySQL 使用者帳戶設定為 REQUIRE SSL，如果無法建立安全連線，連線嘗試將會失敗。

若要強制加密現有 MySQL 資料庫使用者帳戶，可使用 ALTER USER 命令。語法依據 MySQL 版本而異，如下表所示。如需詳細資訊，請參閱《MySQL 參考手冊》中的 ALTER USER。

MySQL 5.7、MySQL 8.0	MySQL 5.6
`ALTER USER 'user'@'%' REQUIRE SSL;`	`GRANT USAGE ON *.* to 'user'@'%' REQUIRE SSL;`

如需 mysql_fdw 擴充功能的詳細資訊，請參閱 mysql_fdw 文件。

使用 oracle_fdw 擴充功能處理 Oracle 資料庫

若要從您的 Aurora PostgreSQL 資料庫叢集 存取 Oracle 資料庫，您可以安裝並使用 oracle_fdw 擴充功能。此擴充功能是 Oracle 資料庫的外部資料包裝函式。若要進一步了解此擴充功能，請參閱 oracle_fdw 文件。

PostgreSQL 12.7 (Amazon Aurora PostgreSQL 版本 4.2) 及更新版本支援 oracle_fdw 擴充功能。

主題

- 開啟 oracle_fdw 擴充功能
- 範例：使用 Amazon RDS for Oracle Database 的外部伺服器連結
- 在傳輸中使用加密
- 了解 pg_user_mappings 檢視和許可權限

開啟 oracle_fdw 擴充功能

如要使用 oracle_fdw 擴充功能，請執行下列程序。

如要開啟 oracle_fdw 擴充功能

- 使用具有 `rds_superuser` 許可的帳戶執行下列命令。

```
CREATE EXTENSION oracle_fdw;
```

範例：使用 Amazon RDS for Oracle Database 的外部伺服器連結

下列範例顯示使用連結至 Amazon RDS for Oracle 資料庫的外部伺服器。

若要建立連結至 RDS for Oracle 資料庫的外部伺服器

1. 請注意以下 RDS for Oracle 資料庫執行個體的事項：
   - 端點
   - 連線埠
   - 資料庫名稱
2. 建立外部伺服器。

```
test=> CREATE SERVER oradb FOREIGN DATA WRAPPER oracle_fdw OPTIONS (dbserver
 '//endpoint:port/DB_name');
CREATE SERVER
```

3. 授予使用權限給沒有 `rds_superuser` 權限的使用者，例如 `user1`。

```
test=> GRANT USAGE ON FOREIGN SERVER oradb TO user1;
GRANT
```

4. 連線為 `user1` 並建立一個對應至 Oracle 使用者的映射。

```
test=> CREATE USER MAPPING FOR user1 SERVER oradb OPTIONS (user 'oracleuser',
 password 'mypassword');
CREATE USER MAPPING
```

5. 建立連結至 Oracle 資料表的外部資料表。

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER oradb OPTIONS (table 'MYTABLE');
CREATE FOREIGN TABLE
```

6. 查詢外部資料表。

```
test=>  SELECT * FROM mytab;
a
---
1
(1 row)
```

如果查詢報告下列錯誤，請檢查您的安全群組和存取控制清單 (ACL)，以確定這兩個執行個體可以通訊。

```
ERROR: connection for foreign table "mytab" cannot be established
DETAIL: ORA-12170: TNS:Connect timeout occurred
```

在傳輸中使用加密

傳輸中的 PostgreSQL-to-Oracle 加密是以從用戶端和伺服器組態參數的組合為依據。如需使用 Oracle 21c 的範例，請參閱 Oracle 文件中的關於溝通加密和完整性的值。在 Amazon RDS 上用於 oracle_fdw 的用戶端已設定為 ACCEPTED，表示加密取決於 Oracle 資料庫伺服器組態，其會使用 Oracle 安全性程式庫 (libnnz) 進行加密。

如果資料庫位於 RDS for Oracle，請參閱 Oracle 原生網路加密以設定加密。

了解 pg_user_mappings 檢視和許可權限

PostgreSQL 目錄 `pg_user_mapping` 儲存從 Aurora PostgreSQL 使用者新增至外部資料 (遠端) 伺服器上使用者的映射。存取目錄受到限制，但您使用 `pg_user_mappings` 檢視查看映射。接下來，您可找到一個範例，其顯示許可權限如何套用於範例 Oracle 資料庫，但此資訊通常適用於任何外部資料包裝函式。

在以下輸出中，您可以找到映射至三個不同範例使用者的角色和權限。使用者 rdssu1 和 rdssu2 是 rds_superuser 角色的成員，user1 則不是。此範例使用 psql 中繼命令 \du 來列出現有角色。

```
test=>  \du
                                                                 List of roles
    Role name    |                          Attributes                          |
                  Member of
-----------------+--------------------------------------------------------------
+------------------------------------------------------------
 rdssu1          |                                                              |
 {rds_superuser}
 rdssu2          |                                                              |
 {rds_superuser}
 user1           |                                                              | {}
```

所有使用者，包括具有 rds_superuser 權限的使用者，可在 pg_user_mappings 資料表中查看自己的使用者映射 (umoptions)。如以下範例所示，當 rdssu1 嘗試獲取所有使用者映射，即使有 rdssu1rds_superuser 權限，仍會引起錯誤：

```
test=> SELECT * FROM pg_user_mapping;
ERROR: permission denied for table pg_user_mapping
```

下列是一些範例。

```
test=> SET SESSION AUTHORIZATION rdssu1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
-------+-------+---------+--------+------------+---------------------------------
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     | {user=oracleuser,password=mypwd}
 16424 | 16411 | oradb   |  16422 | rdssu2     |
 (3 rows)

test=> SET SESSION AUTHORIZATION rdssu2;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
-------+-------+---------+--------+------------+---------------------------------
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     | {user=oracleuser,password=mypwd}
```

```
 (3 rows)

test=> SET SESSION AUTHORIZATION user1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |           umoptions
-------+-------+---------+--------+------------+-------------------------------
 16414 | 16411 | oradb   |  16412 | user1      | {user=oracleuser,password=mypwd}
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     |
 (3 rows)
```

由於實作 `information_schema._pg_user_mappings` 和 `pg_catalog.pg_user_mappings` 存在差異，手動建立的 `rds_superuser` 需要額外的許可才能在 `pg_catalog.pg_user_mappings` 上檢視密碼。

`rds_superuser` 不需要額外許可才能在 `information_schema._pg_user_mappings` 中檢視密碼。

沒有 `rds_superuser` 角色的使用者僅能在以下情況下在 `pg_user_mappings` 中檢視密碼：

- 目前使用者是被映射的使用者，且擁有伺服器或擁有伺服器上 USAGE 的權限。
- 目前使用者是伺服器擁有者，且映射適用於 PUBLIC。

使用 tds_fdw 擴充功能處理 SQL 資料庫

您可以使用 PostgreSQL `tds_fdw` 擴充功能來存取支援表格式資料串流 (TDS) 協議的資料庫，如 Sybase 和 Microsoft SQL Server 資料庫。此外部資料包裝函式可讓您從 Aurora PostgreSQL 資料庫叢集連線到使用 TDS 協議的資料庫，包括 Amazon RDS for Microsoft SQL Server。如需詳細資訊，請參閱 GitHub 上的 tds-fdw/tds_fdw 文件。

Amazon Aurora PostgreSQL 版本 13.6 及更新版本支援 `tds_fdw` 擴充功能。

設定 Aurora PostgreSQL 資料庫以使用 tds_fdw 擴充功能

在以下程序中，您可以找到設定及使用 `tds_fdw` 與 Aurora PostgreSQL 資料庫叢集 的範例。您必須取得執行個體的以下詳細內容，然後才能使用 `tds_fdw` 連線到 SQL Server 資料庫：

- 主機名稱或端點。如果是 RDS for SQL Server 資料庫執行個體，您可以使用主控台找到端點。選擇 Connectivity & security (連線和安全) 索引標籤，然後查看「端點和連線埠」區段。

- 連線埠號碼。Microsoft SQL 伺服器的預設連線埠號是 1433。
- 資料庫的名稱。資料庫識別符。

您也必須提供 SQL 連線埠 1433 的安全群組或存取控制清單 (ACL) 的存取權限。Aurora PostgreSQL 資料庫叢集 和 RDS for SQL 資料庫執行個體都需要存取連線埠 1433。如果存取權限未正確設定，當您嘗試查詢 Microsoft SQL Server 時，您會看到以下錯誤訊息：

```
ERROR: DB-Library error: DB #: 20009, DB Msg: Unable to connect:
Adaptive Server is unavailable or does not exist (mssql2019.aws-
region.rds.amazonaws.com), OS #: 0, OS Msg: Success, Level: 9
```

使用 tds_fdw 連線到 SQL Server 資料庫

1. 使用有 `rds_superuser` 角色的帳戶連線到您的 Aurora PostgreSQL 資料庫叢集的主要執行個體：

   ```
   psql --host=your-cluster-name-instance-1.aws-region.rds.amazonaws.com --port=5432
    --username=test --password
   ```

2. 安裝 `tds_fdw` 擴充功能：

   ```
   test=> CREATE EXTENSION tds_fdw;
   CREATE EXTENSION
   ```

擴充功能安裝到您的 Aurora PostgreSQL 資料庫叢集 之後，您就可以設定外部伺服器。

建立外部伺服器

使用有 `rds_superuser` 權限的帳戶，在 Aurora PostgreSQL 資料庫叢集上執行這些任務。

1. 在 Aurora PostgreSQL 資料庫叢集中建立外部伺服器：

   ```
   test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS
    (servername 'mssql2019.aws-region.rds.amazonaws.com', port '1433', database
    'tds_fdw_testing');
   CREATE SERVER
   ```

   如要存取 SQLServer 端上的非 ASCII 資料，請使用 Aurora PostgreSQL 資料庫叢集 中的 character_set 選項建立伺服器連結：

```
test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS (servername
 'mssql2019.aws-region.rds.amazonaws.com', port '1433', database 'tds_fdw_testing',
 character_set 'UTF-8');
CREATE SERVER
```

2. 將權限授予沒有 rds_superuser 角色權限的使用者，例如 user1：

```
test=> GRANT USAGE ON FOREIGN SERVER sqlserverdb TO user1;
```

3. 以 user1 身分連線，然後建立一個 SQL Server 使用者的映射：

```
test=> CREATE USER MAPPING FOR user1 SERVER sqlserverdb OPTIONS (username
 'sqlserveruser', password 'password');
CREATE USER MAPPING
```

4. 建立連結至 SQL Server 資料表的外部資料表：

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER sqlserverdb OPTIONS (table
 'MYTABLE');
CREATE FOREIGN TABLE
```

5. 查詢外部資料表：

```
test=> SELECT * FROM mytab;
 a
---
 1
(1 row)
```

對連線使用傳輸中加密

從 Aurora PostgreSQL 到 SQL Server 的連線使用傳輸中加密 (TLS/SSL)，具體取決於 SQL Server 資料庫組態。如果 SQL 伺服器未設定為加密，則 RDS for PostgreSQL 用戶端對 SQL Server 資料庫發出的請求將回退為未加密。

您可以透過設定 rds.force_ssl 參數，強制加密 RDS for SQL Server 資料庫執行個體的連線。若要了解作法，請參閱強制使用 SSL 連線至資料庫執行個體。如需有關 RDS for SQL Server 的 SSL/TLS 組態的詳細資訊，請參閱對 Microsoft SQL Server 資料庫執行個體使用 SSL。

使用適用於 PostgreSQL 的受信任語言延伸模組

PostgreSQL 的受信任語言延伸模組是用於建置 PostgreSQL 延伸模組的開放原始碼開發套件。它可讓您建置高效能 PostgreSQL 延伸模組，並在 Aurora PostgreSQL 資料庫叢集上安全地執行它們 透過使用適用於 PostgreSQL 的受信任語言延伸模組 (TLE)，您可以建立 PostgreSQL 延伸模組，遵循記載的方法來擴充 PostgreSQL 功能。如需詳細資訊，請參閱 PostgreSQL 文件中的將相關物件封裝為延伸模組。

TLE 的一個主要優點，就是您可以在未於 PostgreSQL 執行個體之下提供檔案系統存取權的環境中使用它。先前，安裝新的延伸模組需要存取檔案系統。TLE 移除了此條件約束。它提供一個開發環境，可讓您針對任何 PostgreSQL 資料庫 (包括在 Aurora PostgreSQL 資料庫叢集上執行的資料庫) 建立新的延伸模組。

TLE 的設計旨在防止存取對您使用 TLE 建立的延伸模組而言不安全的資源。其執行時間環境會限制任何延伸模組瑕疵對單一資料庫連線的影響。TLE 還可讓資料庫管理員對可以安裝延伸模組的人員進行精細控制，並提供用於執行延伸模組的許可模型。

Aurora PostgreSQL 14.5 版及更新版本支援 TLE。

受信任語言延伸模組開發環境和執行時間會封裝成 `pg_tle` PostgreSQL 延伸模組 1.0.1 版。它支援以 JavaScript、Perl、Tcl、PL/PGSQL 和 SQL 建立延伸模組。您可以採取您安裝其他 PostgreSQL 延伸模組的同一方式，在 Aurora PostgreSQL 資料庫叢集中安裝 `pg_tle` 延伸模組。在設定了 `pg_tle` 之後，開發人員可以使用它，建立新的 PostgreSQL 延伸模組，稱為 TLE 延伸模組。

在下列主題中，您可以找到如何設定受信任語言延伸模組，以及如何開始建立您自己的 TLE 延伸模組的相關資訊。

主題

- 術語
- 使用適用於 PostgreSQL 的受信任語言延伸模組的需求
- 在您的 Aurora PostgreSQL 資料庫叢集中設定受信任語言延伸模組
- 適用於 PostgreSQL 的受信任語言延伸模組概觀
- 針對 Aurora PostgreSQL 建立 TLE 延伸模組
- 從資料庫中捨棄您的 TLE 延伸模組
- 解除安裝適用於 PostgreSQL 的受信任語言延伸模組

- 搭配您的延伸模組使用 PostgreSQL 掛鉤
- 適用於 PostgreSQL 的受信任語言延伸模組的函數參考
- 適用於 PostgreSQL 的受信任語言延伸模組的掛鉤參考

術語

為了協助您更好地了解受信任語言延伸模組，請檢視下列詞彙表以取得本主題中使用的術語。

適用於 PostgreSQL 的受信任語言延伸模組

適用於 PostgreSQL 的受信任語言延伸模組是封裝為 `pg_tle` 延伸模組之開放原始碼開發套件的正式名稱。它可以在任何 PostgreSQL 系統上使用。如需詳細資訊，請參閱 GitHub 上的 aws/pg_tle。

受信任語言延伸模組

受信任語言延伸模組是適用於 PostgreSQL 的受信任語言延伸模組的簡稱。此簡稱及其縮寫 (TLE) 也會在本文件中使用。

受信任語言

受信任語言是具有特定安全性屬性的程式設計或指令碼語言。例如，受信任語言通常會限制對檔案系統的存取，而且也會限制使用指定的聯網屬性。TLE 開發套件專為支援受信任語言而設計。PostgreSQL 支援數種不同的語言，這些語言用來建立受信任或不受信任的延伸模組。如需範例，請參閱 PostgreSQL 文件中的受信任和不受信任的 PL/Perl。當您使用受信任語言延伸模組建立延伸模組時，延伸模組本質上會使用受信任語言機制。

TLE 延伸模組

TLE 延伸模組是已使用受信任語言延伸模組 (TLE) 開發套件所建立的 PostgreSQL 延伸模組。

使用適用於 PostgreSQL 的受信任語言延伸模組的需求

以下是設定和使用 TLE 開發套件時的需求。

- Aurora PostgreSQL 版本 - 僅在 Aurora PostgreSQL 14.5 版 及更新版本上支援受信任語言延伸模組。
  - 如果您需要升級 Aurora PostgreSQL 資料庫叢集， 請參閱 升級 Amazon Aurora PostgreSQL 資料庫叢集。

- 如果您還沒有執行 PostgreSQL 的 Aurora 資料庫叢集 ，則可以建立一個。如需詳細資訊，請參閱建立 Aurora PostgreSQL 資料庫叢集並與之連線。
- 需要 **rds_superuser** 權限 - 若要設定 pg_tle 延伸模組，您的資料庫使用者角色必須具有 rds_superuser 角色的許可。根據預設，會將此角色授予建立 Aurora PostgreSQL 資料庫叢集的 postgres 使用者。
- 需要自訂資料庫參數群組 – 您的 Aurora PostgreSQL 資料庫叢集必須搭配自訂資料庫參數群組進行設定。您會針對 Aurora PostgreSQL 資料庫叢集的寫入器執行個體使用自訂資料庫參數群組。
  - 如果您的 Aurora PostgreSQL 資料庫叢集 未搭配自訂資料庫參數群組進行設定，您應該建立一個，並將其與 Aurora PostgreSQL 資料庫叢集的寫入器執行個體建立關聯。如需步驟的簡短摘要，請參閱 建立並套用自訂資料庫參數群組。
  - 如果您的 Aurora PostgreSQL 資料庫叢集 已使用自訂資料庫參數群組進行設定，您可以設定受信任語言延伸模組。如需詳細資訊，請參閱在您的 Aurora PostgreSQL 資料庫叢集中設定受信任語言延伸模組。

建立並套用自訂資料庫參數群組

請使用下列步驟建立自訂資料庫參數群組，並設定您的 Aurora PostgreSQL 資料庫叢集來使用該群組。

主控台

建立自訂資料庫參數群組，並將其與 Aurora PostgreSQL 資料庫叢集 搭配使用

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 從 Amazon RDS 功能表中，選擇 Parameter groups (參數群組)。
3. 選擇 Create parameter group (建立參數群組)。
4. 在 Parameter group details (參數群組詳細資訊) 頁面中，輸入下列資訊：
   - 針對 Parameter group family (參數群組系列)，選擇 aurora-postgresql14。 。
   - 針對 Type (類型)，選擇 DB Parameter Group (資料庫參數群組)。
   - 針對 Group name (群組名稱)，在操作內容中給與您的參數群組一個有意義的名稱。
   - 針對 Description (描述)，輸入有用的描述，以便您團隊中的其他成員可以輕鬆找到它。
5. 選擇建立。您的自訂資料庫參數群組是在 AWS 區域中建立。您現在可以遵循下列步驟修改 Aurora PostgreSQL 資料庫叢集以使用它。

6. 從 Amazon RDS 功能表中選擇 Databases (資料庫)。
7. 從列出的 Aurora PostgreSQL 資料庫叢集中選擇您想要哪一個與 TLE 搭配使用，然後選擇 Modify (修改)。
8. 在 Modify DB cluster settings (修改資料庫叢集設定) 頁面中，尋找 Database options (資料庫選項)，然後使用選取器選擇您的自訂資料庫參數群組。
9. 選擇 Continue (繼續) 以儲存變更。
10. 選擇 Apply immediately (立即套用)，以便您可以繼續設定 Aurora PostgreSQL 資料庫叢集以使用 TLE。

若要繼續針對受信任語言延伸模組設定您的系統，請參閱 在您的 Aurora PostgreSQL 資料庫叢集中設定受信任語言延伸模組。

如需深入了解如何使用資料庫叢集和資料庫參數群組，請參閱 Amazon Aurora 資料庫叢集的資料庫叢集參數群組。

AWS CLI

您可以避免在使用 CLI 命令時指定 `--region` 引數，方法是使用您的預設 AWS 區域來設定您的 AWS CLI 。如需詳細資訊，請參閱《AWS Command Line Interface 使用者指南》中的組態基礎概念。

建立自訂資料庫參數群組，並將其與 Aurora PostgreSQL 資料庫叢集 搭配使用

1. 使用 create-db-parameter-group AWS CLI 命令，根據 aurora-postgresql14 為您的 建立自訂資料庫參數群組 AWS 區域。請注意，在此步驟中，您會建立資料庫參數群組，以套用至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。

   對於 Linux、macOS 或 Unix：

   ```
   aws rds create-db-parameter-group \
     --region aws-region \
     --db-parameter-group-name custom-params-for-pg-tle \
     --db-parameter-group-family aurora-postgresql14 \
     --description "My custom DB parameter group for Trusted Language Extensions"
   ```

   在 Windows 中：

   ```
   aws rds create-db-parameter-group ^
     --region aws-region ^
     --db-parameter-group-name custom-params-for-pg-tle ^
   ```

```
  --db-parameter-group-family aurora-postgresql14 ^
  --description "My custom DB parameter group for Trusted Language Extensions"
```

您的自訂資料庫參數群組可在 AWS 區域中使用，因此您可以修改 Aurora PostgreSQL 資料庫叢集的寫入器執行個體以使用它。

2. 使用 modify-db-instance AWS CLI 命令，將您的自訂資料庫參數群組套用至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。此命令會立即重新啟動作用中執行個體。

   對於 Linux、macOS 或 Unix：

```
aws rds modify-db-instance \
  --region aws-region \
  --db-instance-identifier your-writer-instance-name \
  --db-parameter-group-name custom-params-for-pg-tle \
  --apply-immediately
```

   在 Windows 中：

```
aws rds modify-db-instance ^
  --region aws-region ^
  --db-instance-identifier your-writer-instance-name ^
  --db-parameter-group-name custom-params-for-pg-tle ^
  --apply-immediately
```

若要繼續針對受信任語言延伸模組設定您的系統，請參閱 在您的 Aurora PostgreSQL 資料庫叢集中設定受信任語言延伸模組。

如需更多詳細資訊，請參閱 Amazon Aurora 資料庫執行個體的資料庫參數群組。

在您的 Aurora PostgreSQL 資料庫叢集中設定受信任語言延伸模組

下列步驟假設您的 Aurora PostgreSQL 資料庫叢集與自訂資料庫叢集參數群組相關聯。您可以針對這些步驟使用 AWS 管理主控台 或 AWS CLI。

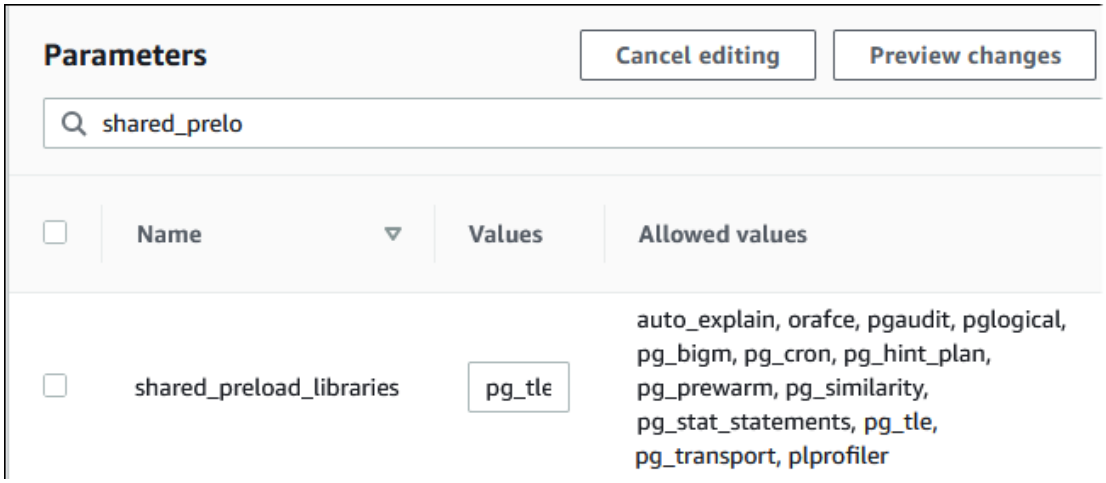
當您在 Aurora PostgreSQL 資料庫叢集中設定受信任語言延伸模組時，可以將其安裝在特定資料庫中，供具有該資料庫許可的資料庫使用者使用。

主控台

設定受信任語言延伸模組

使用屬於 rds_superuser 群組 (角色) 成員的帳戶執行下列步驟。

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇您的 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。
3. 針對您的 Aurora PostgreSQL 資料庫叢集的寫入器執行個體開啟 Configuration (組態) 索引標籤。在執行個體詳細資訊之間，尋找 Parameter group (參數群組) 連結。
4. 選擇連結以開啟與 Aurora PostgreSQL 資料庫叢集相關聯的自訂參數。
5. 在 Parameters (參數) 搜尋欄位中，輸入 shared_pre 以尋找 shared_preload_libraries 參數。
6. 選擇 Edit parameters (編輯參數) 以存取屬性值。
7. 在 Values (值) 欄位中，將 pg_tle 新增至清單。使用逗號區隔值清單中的項目。



8. 重新啟動 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，以便您對 shared_preload_libraries 參數所做的變更生效。
9. 當執行個體可用時，請驗證 pg_tle 是否已初始化。使用 psql 連線至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，然後執行下列命令。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

10. 在初始化 pg_tle 延伸模組之後，您現在可以建立延伸模組。

```
CREATE EXTENSION pg_tle;
```

您可以使用下列 psql 中繼命令，驗證是否已安裝延伸模組。

```
labdb=> \dx
                          List of installed extensions
  Name   | Version |   Schema   |                Description
---------+---------+------------+---------------------------------------------
 pg_tle  | 1.0.1   | pgtle      | Trusted-Language Extensions for PostgreSQL
 plpgsql | 1.0     | pg_catalog | PL/pgSQL procedural language
```

11. 在設定時，將 pgtle_admin 角色授予您已針對 Aurora PostgreSQL 資料庫叢集建立的主要使用者名稱。如果您接受預設值，其為 postgres。

```
labdb=> GRANT pgtle_admin TO postgres;
GRANT ROLE
```

您可以使用 psql 中繼命令來驗證授予是否已發生，如下列範例所示。只有 pgtle_admin 和 postgres 角色會顯示在輸出中。如需詳細資訊，請參閱 了解 PostgreSQL 角色和許可。

```
labdb=> \du
                          List of roles
    Role name    |           Attributes            |               Member of
-----------------+---------------------------------
+-----------------------------------
pgtle_admin     | Cannot login                     | {}
postgres        | Create role, Create DB          +| {rds_superuser,pgtle_admin}
                | Password valid until infinity    |...
```

12. 使用 \q 中繼命令關閉 psql 工作階段。

```
\q
```

若要開始建立 TLE 延伸模組，請參閱 範例：使用 SQL 建立受信任語言延伸模組。

AWS CLI

您可以避免在使用 CLI 命令時指定 `--region` 引數，方法是使用您的預設 AWS 區域來設定您的 AWS CLI。如需詳細資訊，請參閱《AWS Command Line Interface 使用者指南》中的組態基礎概念。

設定受信任語言延伸模組

1. 使用 modify-db-parameter-group AWS CLI 命令，將 `pg_tle` 新增至 `shared_preload_libraries` 參數。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pg_tle,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. 使用 reboot-db-instance AWS CLI 命令重新啟動 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，並初始化 `pg_tle` 程式庫。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. 當執行個體可用時，您可以驗證 `pg_tle` 是否已初始化。使用 `psql` 連線至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，然後執行下列命令。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

   在初始化 `pg_tle` 之後，您現在可以建立延伸模組。

```
CREATE EXTENSION pg_tle;
```

4. 在設定時，將 `pgtle_admin` 角色授予您已針對 Aurora PostgreSQL 資料庫叢集建立的主要使用者名稱。如果您接受預設值，其為 `postgres`。

```
GRANT pgtle_admin TO postgres;
```

```
GRANT ROLE
```

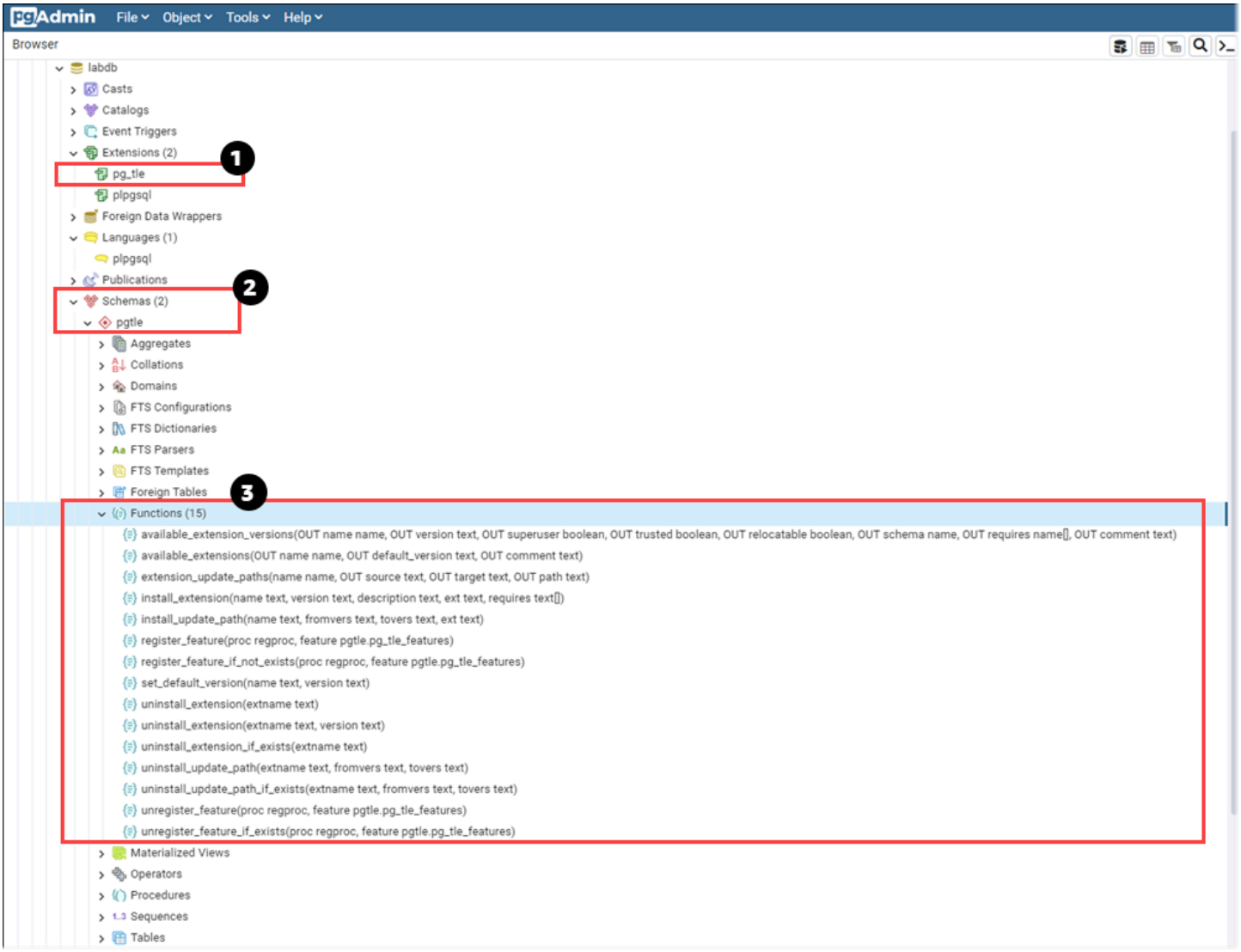
5. 關閉 psql 工作階段，如下所示。

```
labdb=> \q
```

若要開始建立 TLE 延伸模組，請參閱 範例：使用 SQL 建立受信任語言延伸模組。

適用於 PostgreSQL 的受信任語言延伸模組概觀

適用於 PostgreSQL 的受信任語言延伸模組是一種 PostgreSQL 延伸模組，而您使用您設定其他 PostgreSQL 延伸模組的同一方式，將其安裝在 Aurora PostgreSQL 資料庫叢集中。在 pgAdmin 用戶端工具中範例資料庫的下列影像中，您可以檢視構成 pg_tle 延伸模組的一些元件。



您可以查看下列詳細資訊。

1. 適用於 PostgreSQL 的受信任語言延伸模組 (TLE) 開發套件會封裝為 pg_tle 延伸模組。因此，pg_tle 會新增至其安裝所在資料庫的可用延伸模組。
2. TLE 有其自己的結構描述 (pgtle)。此結構描述包含協助程式函數 (3)，用於安裝和管理您建立的延伸模組。
3. TLE 提供了十幾個協助程式函數，用於安裝、註冊和管理您的延伸模組。若要進一步了解這些函數，請參閱適用於 PostgreSQL 的受信任語言延伸模組的函數參考。

pg_tle 延伸套件的其他元件包含下列項目：

- **pgtle_admin** 角色 – 安裝 pg_tle 延伸模組時會建立 pgtle_admin 角色。此角色具有特殊權限，且應如此對待。強烈建議您在將 pgtle_admin 角色授予資料庫使用者時遵循最低權限原則。換句話說，只將 pgtle_admin 角色授予資料庫使用者，允許其建立、安裝和管理新的 TLE 延伸模組，例如 postgres。
- **pgtle.feature_info** 資料表 – pgtle.feature_info 資料表是受保護的資料表，其中包含 TLE、掛鉤，以及其使用的自訂預存程序和函數的相關資訊。如果具有 pgtle_admin 權限，則您可以使用下列受信任語言延伸模組函數，在資料表中新增和更新該資訊。
  - pgtle.register_feature
  - pgtle.register_feature_if_not_exists
  - pgtle.unregister_feature
  - pgtle.unregister_feature_if_exists

針對 Aurora PostgreSQL 建立 TLE 延伸模組

您可以在任何已安裝延伸模組的 Aurora PostgreSQL 資料庫叢集 中安裝使用 TLE 建立的任何 pg_tle 延伸模組。pg_tle 延伸模組的範圍會限定在其安裝所在的 PostgreSQL 資料庫。您使用 TLE 建立的延伸模組，其範圍會限定在相同的資料庫。

使用各種 pgtle 函數來安裝構成 TLE 延伸模組的程式碼。下列受信任語言延伸模組函數全都需要 pgtle_admin 角色。

- pgtle.install_extension
- pgtle.install_update_path
- pgtle.register_feature

- pgtle.register_feature_if_not_exists
- pgtle.set_default_version
- pgtle.uninstall_extension(name)
- pgtle.uninstall_extension(name, version)
- pgtle.uninstall_extension_if_exists
- pgtle.uninstall_update_path
- pgtle.uninstall_update_path_if_exists
- pgtle.unregister_feature
- pgtle.unregister_feature_if_exists

範例：使用 SQL 建立受信任語言延伸模組

下列範例說明如何建立名為 pg_distance 的 TLE 延伸模組，其中包含一些 SQL 函數，用於使用不同的公式計算距離。在清單中，您可以找到用於計算曼哈頓距離的函數，以及計算歐幾里得距離的函數。如需這些公式之間差異的詳細資訊，請參閱 Wikipedia 中的出租車幾何和歐幾里得幾何。

如果您已按照在您的 Aurora PostgreSQL 資料庫叢集中設定受信任語言延伸模組中所述設定 pg_tle 延伸模組，則可以在自己的 Aurora PostgreSQL 資料庫叢集 中使用此範例。

> **Note**
>
> 您必須具有 pgtle_admin 角色的權限才能遵循此程序。

建立範例 TLE 延伸模組

下列步驟會使用名為 labdb 的範例資料庫。此資料庫是由 postgres 主要使用者所擁有。postgres 角色也具有 pgtle_admin 角色的權限。

1. 使用 psql 連線到 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. 複製下列程式碼並將其貼入 psql 工作階段主控台中，來建立名為 pg_distance 的 TLE 延伸模組。

```
SELECT pgtle.install_extension
(
 'pg_distance',
 '0.1',
  'Distance functions for two points',
$_pg_tle_$
    CREATE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8, norm int)
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL;

    CREATE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 1);
    $$ LANGUAGE SQL;

    CREATE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL;
$_pg_tle_$
);
```

您會看到如下輸出。

```
install_extension
---------------
 t
(1 row)
```

構成 pg_distance 延伸模組的成品現在已安裝在您的資料庫中。這些成品包括延伸模組的控制檔和程式碼，這些是必須存在的項目，如此才能使用 CREATE EXTENSION 命令建立延伸模組。換句話說，您仍然需要建立延伸模組，以使其函數可供資料庫使用者使用。

3. 若要建立延伸模組，請使用 CREATE EXTENSION 命令，如您對任何其他延伸模組所做一樣。與其他延伸模組一樣，資料庫使用者需要在資料庫中具有 CREATE 權限。

```
CREATE EXTENSION pg_distance;
```

4. 若要測試 pg_distance TLE 延伸模組，您可以使用它，計算四點之間的曼哈頓距離。

```
labdb=> SELECT manhattan_dist(1, 1, 5, 5);
8
```

若要計算同一組點之間的歐幾里得距離，您可以使用下列命令。

```
labdb=> SELECT euclidean_dist(1, 1, 5, 5);
5.656854249492381
```

pg_distance 延伸模組會在資料庫中載入函數，並使其可供具有資料庫權限的任何使用者使用。

修改 TLE 延伸模組

若要改善此 TLE 延伸模組中封裝之函數的查詢效能，請將下列兩個 PostgreSQL 屬性新增至其規格。

- IMMUTABLE – IMMUTABLE 屬性確保查詢最佳化工具可以使用最佳化來改善查詢回應時間。如需詳細資訊，請參閱 PostgreSQL 文件中的函數波動類別。
- PARALLEL SAFE – PARALLEL SAFE 屬性是允許 PostgreSQL 以平行模式執行函數的另一個屬性。如需詳細資訊，請參閱 PostgreSQL 文件中的 CREATE FUNCTION。

在下列範例中，您可以看到如何使用 pgtle.install_update_path 函數將這些屬性新增到每個函數，以建立 pg_distance TLE 延伸模組的版本 0.2。如需此函數狀態的詳細資訊，請參閱 pgtle.install_update_path。您必須具有 pgtle_admin 角色才能執行此任務。

更新現有 TLE 延伸模組並指定預設版本

1. 使用 psql 或其他用戶端工具 (例如 pgAdmin)，連線至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體 。

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. 複製下列程式碼並將其貼入 psql 工作階段主控台中，來修改現有的 TLE 延伸模組。

```
SELECT pgtle.install_update_path
```

```
(
 'pg_distance',
 '0.1',
 '0.2',
$_pg_tle_$
    CREATE OR REPLACE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8,
 norm int)
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 1);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;
$_pg_tle_$
);
```

您會看到如下回應：

```
install_update_path
--------------------
 t
(1 row)
```

您可以將此版本的延伸模組設為預設版本，這樣資料庫使用者就不必在資料庫中建立或更新延伸模組時指定版本。

3. 若要指定 TLE 延伸模組的修改版本 (版本 0.2) 為預設版本，請使用 `pgtle.set_default_version` 函數，如下列範例所示。

```
SELECT pgtle.set_default_version('pg_distance', '0.2');
```

如需此函數狀態的詳細資訊，請參閱 pgtle.set_default_version。

4. 在程式碼就位之後，您可以使用 ALTER EXTENSION ... UPDATE 命令，以尋常方式更新已安裝的 TLE 延伸模組，如這裡所示：

```
ALTER EXTENSION pg_distance UPDATE;
```

從資料庫中捨棄您的 TLE 延伸模組

您可以採取您對其他 PostgreSQL 延伸模組所用的同一方式，使用 DROP EXTENSION 命令來捨棄 TLE 延伸模組。捨棄延伸模組並不會移除構成延伸模組的安裝檔案，因而允許使用者重新建立延伸模組。若要移除延伸模組及其安裝檔案，請執行下列兩步驟程序。

捨棄 TLE 延伸模組並移除其安裝檔案

1. 使用 psql 或其他用戶端工具，連線至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=dbname
```

2. 捨棄延伸模組，如同您對任何 PostgreSQL 延伸模組所做一般。

```
DROP EXTENSION your-TLE-extension
```

例如，如果您如範例：使用 SQL 建立受信任語言延伸模組所述建立 pg_distance 延伸模組，則可以捨棄延伸模組，如下所示。

```
DROP EXTENSION pg_distance;
```

您會看到輸出，確認已捨棄延伸模組，如下所示。

```
DROP EXTENSION
```

此時，延伸模組在資料庫中不再處於作用中狀態。不過，其安裝檔案和控制檔案仍然可在資料庫中使用，因此資料庫使用者可以再次建立延伸模組 (如果想要的話)。

- 如果想要延伸模組檔案保持不變，以便資料庫使用者可以建立您的 TLE 延伸模組，您可以在此停止。

- 如果想要移除所有構成延伸模組的檔案，請繼續下一個步驟。

3. 若要移除延伸模組的所有安裝檔案，請使用 pgtle.uninstall_extension 函數。此函數會移除延伸模組的所有程式碼和控制檔。

```
SELECT pgtle.uninstall_extension('your-tle-extension-name');
```

例如，若要移除所有 pg_distance 安裝檔案，請使用下列命令。

```
SELECT pgtle.uninstall_extension('pg_distance');
 uninstall_extension
---------------------
 t
(1 row)
```

解除安裝適用於 PostgreSQL 的受信任語言延伸模組

如果您再也不要使用 TLE 建立自己的 TLE 延伸模組，則可以捨棄 pg_tle 延伸模組並移除所有成品。此動作包括捨棄資料庫中的任何 TLE 延伸模組，以及捨棄 pgtle 結構描述。

從資料庫中捨棄 **pg_tle** 延伸模組及其結構描述

1. 使用 psql 或其他用戶端工具，連線至 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=dbname
```

2. 從資料庫中捨棄 pg_tle 延伸模組。如果您自己的 TLE 延伸模組仍在資料庫中執行，您也需要捨棄這些延伸模組。若要這樣做，您可以使用 CASCADE 關鍵字，如下所示。

```
DROP EXTENSION pg_tle CASCADE;
```

如果 pg_tle 延伸模組在資料庫中仍未作用中，則您不需要使用 CASCADE 關鍵字。

3. 捨棄 pgtle 結構描述。此動作會移除資料庫中的所有管理函數。

```
DROP SCHEMA pgtle CASCADE;
```

此命令會在程序完成時傳回下列內容。

```
DROP SCHEMA
```

pg_tle 延伸模組、其結構描述和函數，以及所有成品都會遭到移除。若要使用 TLE 建立新的延伸模組，請再次完成設定程序。如需更多詳細資訊，請參閱 在您的 Aurora PostgreSQL 資料庫叢集中設定受信任語言延伸模組。

搭配您的延伸模組使用 PostgreSQL 掛鉤

掛鉤是 PostgreSQL 中提供的回呼機制，允許開發人員在一般資料庫操作期間呼叫自訂函數或其他常式。TLE 開發套件支援 PostgreSQL 掛鉤，以便您可以在執行時整合自訂函數與 PostgreSQL 行為。例如，您可以使用掛鉤，將身分驗證程序與您自己的自訂程式碼建立關聯，或因應您的特定需求修改查詢規劃和執行程序。

您的 TLE 延伸模組可以使用掛鉤。如果掛鉤在範圍內是全域的，則其在所有資料庫之中都適用。因此，如果您的 TLE 延伸模組使用全域掛鉤，則您需要在使用者可以存取的所有資料庫中建立 TLE 延伸模組。

當使用 pg_tle 延伸模組來建置您自己的受信任語言延伸模組時，您可以使用 SQL API 中可用的掛鉤來建置延伸模組的函數。您應該使用 pg_tle 註冊任何掛鉤。對於某些掛鉤，您可能還需要設定各種組態參數。例如，passcode 檢查掛鉤可以設為開啟、關閉或需要。如需可用 pg_tle 掛鉤之特定需求的詳細資訊，請參閱 適用於 PostgreSQL 的受信任語言延伸模組的掛鉤參考。

範例：建立使用 PostgreSQL 掛鉤的延伸模組

本節中討論的範例使用 PostgreSQL 掛鉤，來檢查在特定 SQL 操作期間提供的密碼，並防止資料庫使用者將其密碼設為 password_check.bad_passwords 資料表中包含的任何密碼。該資料表包含前十大最常用但容易破解的密碼選擇。

若要在您的 Aurora PostgreSQL 資料庫叢集 中設定此範例，您必須已安裝受信任語言延伸模組。如需詳細資訊，請參閱 在您的 Aurora PostgreSQL 資料庫叢集中設定受信任語言延伸模組。

設定密碼檢查掛鉤範例

1. 使用 psql 連線到 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. 從 密碼檢查掛鉤程式碼清單 中複製程式碼，並將其貼入您的資料庫中。

```
SELECT pgtle.install_extension (
  'my_password_check_rules',
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;

  CREATE TABLE password_check.bad_passwords (plaintext) AS
  VALUES
    ('123456'),
    ('password'),
    ('12345678'),
    ('qwerty'),
    ('123456789'),
    ('12345'),
    ('1234'),
    ('111111'),
    ('1234567'),
    ('dragon');
  CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);

  CREATE FUNCTION password_check.passcheck_hook(username text, password text,
 password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
  RETURNS void AS $$
    DECLARE
      invalid bool := false;
    BEGIN
      IF password_type = 'PASSWORD_TYPE_MD5' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE ('md5' || md5(bp.plaintext || username)) = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password
 dictionary';
        END IF;
      ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
```

```
          WHERE bp.plaintext = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password
 dictionary';
        END IF;
      END IF;
    END
  $$ LANGUAGE plpgsql SECURITY DEFINER;

  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

將延伸模組載入資料庫後，您會看到如下輸出。

```
 install_extension
-------------------
 t
(1 row)
```

3. 在仍然連線到資料庫時，您現在可以建立延伸模組。

```
CREATE EXTENSION my_password_check_rules;
```

4. 您可以使用下列 psql 中繼命令，確認已在資料庫中建立延伸模組。

```
\dx
                        List of installed extensions
          Name           | Version |   Schema   |
 Description
-------------------------+---------+------------
+-------------------------------------------------------------
 my_password_check_rules | 1.0     | public     | Prevent use of any of the top-ten
 most common bad passwords
 pg_tle                  | 1.0.1   | pgtle      | Trusted-Language Extensions for
 PostgreSQL
 plpgsql                 | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

5. 開啟另一個要使用 AWS CLI 的終端機工作階段。您需要修改自訂資料庫參數群組，以開啟密碼檢查掛鉤。若要這麼做，請使用 modify-db-parameter-group 命令，如下列範例所示。

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

   可能需要幾分鐘，對參數群組設定所做的變更才會生效。不過，此參數是動態參數，因此您不需要重新啟動 Aurora PostgreSQL 資料庫叢集的寫入器執行個體，設定即可生效。

6. 開啟 psql 工作階段並查詢資料庫，以驗證密碼檢查掛鉤是否已開啟。

```
labdb=> SHOW pgtle.enable_password_check;
pgtle.enable_password_check
----------------------------
on
(1 row)
```

密碼檢查掛鉤現在處於作用中狀態。您可以建立新角色並使用其中一個錯誤密碼來對它進行測試，如下列範例所示。

```
CREATE ROLE test_role PASSWORD 'password';
ERROR:  Cannot use passwords from the common password dictionary
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 21 at RAISE
SQL statement "SELECT password_check.passcheck_hook(
    $1::pg_catalog.text,
    $2::pg_catalog.text,
    $3::pgtle.password_types,
    $4::pg_catalog.timestamptz,
    $5::pg_catalog.bool)"
```

為了方便閱讀，輸出已經過格式化處理。

下列範例顯示 pgsql 互動式中繼命令 \password 行為也受密碼檢查掛鉤的影響。

```
postgres=> SET password_encryption TO 'md5';
SET
```

```
postgres=> \password
Enter new password for user "postgres":*****
Enter it again:*****
ERROR:  Cannot use passwords from the common password dictionary
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 12 at RAISE
SQL statement "SELECT password_check.passcheck_hook($1::pg_catalog.text,
 $2::pg_catalog.text, $3::pgtle.password_types, $4::pg_catalog.timestamptz,
 $5::pg_catalog.bool)"
```

如果需要，您可以捨棄此 TLE 延伸模組並解除安裝其來源檔案。如需更多詳細資訊，請參閱 [從資料庫中捨棄您的 TLE 延伸模組](#)。

密碼檢查掛鉤程式碼清單

此處顯示的範例程式碼定義了 my_password_check_rules TLE 延伸模組的規格。當您複製此程式碼並將其貼入資料庫中時，my_password_check_rules 延伸模組的程式碼會載入至資料庫，而且 password_check 掛鉤會進行註冊以供該延伸模組使用。

```
SELECT pgtle.install_extension (
  'my_password_check_rules',
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;

  CREATE TABLE password_check.bad_passwords (plaintext) AS
  VALUES
    ('123456'),
    ('password'),
    ('12345678'),
    ('qwerty'),
    ('123456789'),
    ('12345'),
    ('1234'),
    ('111111'),
    ('1234567'),
    ('dragon');
  CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);
```

```
  CREATE FUNCTION password_check.passcheck_hook(username text, password text,
 password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
  RETURNS void AS $$
    DECLARE
      invalid bool := false;
    BEGIN
      IF password_type = 'PASSWORD_TYPE_MD5' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE ('md5' || md5(bp.plaintext || username)) = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password dictionary';
        END IF;
      ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE bp.plaintext = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password dictionary';
        END IF;
      END IF;
    END
  $$ LANGUAGE plpgsql SECURITY DEFINER;

  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

適用於 PostgreSQL 的受信任語言延伸模組的函數參考

檢視下列有關適用於 PostgreSQL 的受信任語言延伸模組中提供之函數的參考文件。使用這些函數來安裝、註冊、更新和管理您的 TLE 延伸模組，亦即您使用受信任語言延伸模組開發套件所開發的 PostgreSQL 延伸模組。

函數

- pgtle.available_extensions

- pgtle.available_extension_versions
- pgtle.extension_update_paths
- pgtle.install_extension
- pgtle.install_update_path
- pgtle.register_feature
- pgtle.register_feature_if_not_exists
- pgtle.set_default_version
- pgtle.uninstall_extension(name)
- pgtle.uninstall_extension(name, version)
- pgtle.uninstall_extension_if_exists
- pgtle.uninstall_update_path
- pgtle.uninstall_update_path_if_exists
- pgtle.unregister_feature
- pgtle.unregister_feature_if_exists

pgtle.available_extensions

`pgtle.available_extensions` 函數是一個設定-傳回函數。它會傳回資料庫中所有可用的 TLE 延伸模組。每一傳回的列都包含單一 TLE 延伸模組的相關資訊。

函數原型

```
pgtle.available_extensions()
```

角色

無。

引數

無。

輸出

- name – TLE 延伸模組的名稱。

- default_version – 在未指定版本的情況下呼叫 CREATE EXTENSION 時使用的 TLE 延伸模組版本。
- description – 有關 TLE 延伸模組的詳細描述。

使用範例

```
SELECT * FROM pgtle.available_extensions();
```

pgtle.available_extension_versions

available_extension_versions 函數是一個設定-傳回函數。其會傳回一份清單，列出所有可用的 TLE 延伸模組及其版本。每一列都包含指定 TLE 延伸模組之特定版本的相關資訊，包括其是否需要特定角色。

函數原型

```
pgtle.available_extension_versions()
```

角色

無。

引數

無。

輸出

- name – TLE 延伸模組的名稱。
- version – TLE 延伸模組的版本。
- superuser – 對於 TLE 延伸模組，此值一律為 false。建立 TLE 延伸模組或更新它所需的許可，與在指定資料庫中建立其他物件所需的許可相同。
- trusted – 對於 TLE 延伸模組，此值一律為 false。
- relocatable – 對於 TLE 延伸模組，此值一律為 false。
- schema – 指定其中安裝 TLE 延伸模組的結構描述名稱。
- requires – 一種陣列，其中包含此 TLE 延伸模組所需之其他延伸模組的名稱。

- `description` – TLE 延伸模組的詳細描述。

如需輸出值的詳細資訊，請參閱 PostgreSQL 文件中的將相關物件封裝為延伸模組 > 延伸模組檔案。

使用範例

```
SELECT * FROM pgtle.available_extension_versions();
```

pgtle.extension_update_paths

`extension_update_paths` 函數是一個設定-傳回函數。其會傳回一份清單，列出 TLE 延伸模組的所有可能更新路徑。每一列都包含該 TLE 延伸模組的可用升級或降級。

函數原型

```
pgtle.extension_update_paths(name)
```

角色

無。

引數

`name` – 要從中取得升級路徑的 TLE 延伸模組名稱。

輸出

- `source` – 更新的來源版本。
- `target` – 更新的目標版本。
- `path` – 用來將 TLE 延伸模組從 `source` 版本更新至 `target` 版本的升級路徑，例如 `0.1--0.2`。

使用範例

```
SELECT * FROM pgtle.extension_update_paths('your-TLE');
```

pgtle.install_extension

`install_extension` 函數可讓您在資料庫中安裝構成 TLE 延伸模組的成品，然後您可以使用 `CREATE EXTENSION` 命令來建立它。

函數原型

```
pgtle.install_extension(name text, version text, description text, ext text, requires
 text[] DEFAULT NULL::text[])
```

角色

無。

引數

- name – TLE 延伸模組的名稱。呼叫 CREATE EXTENSION 時會使用此值。
- version – TLE 延伸模組的版本。
- description – 有關 TLE 延伸模組的詳細描述。此描述顯示在 pgtle.available_extensions() 的 comment 欄位中。
- ext – TLE 延伸模組的內容。此值包含函數之類的物件。
- requires – 針對此 TLE 延伸模組指定相依性的選用參數。pg_tle 延伸模組會自動新增為相依性。

其中許多引數與延伸模組控制檔中包含的引數相同，用於在 PostgreSQL 執行個體的檔案系統上安裝 PostgreSQL 延伸模組。如需詳細資訊，請參閱 PostgreSQL 文件中將相關物件封裝為延伸模組中的延伸模組檔案。

輸出

此函數會在成功時傳回 OK，以及在發生錯誤時傳回 NULL。

- OK – TLE 延伸模組已成功安裝在資料庫中。
- NULL – TLE 延伸模組未成功安裝在資料庫中。

使用範例

```
SELECT pgtle.install_extension(
 'pg_tle_test',
 '0.1',
 'My first pg_tle extension',
$_pgtle_$
  CREATE FUNCTION my_test()
```

```
  RETURNS INT
  AS $$
    SELECT 42;
  $$ LANGUAGE SQL IMMUTABLE;
$_pgtle_$
);
```

pgtle.install_update_path

install_update_path 函數會提供 TLE 延伸模組的兩個不同版本之間的更新路徑。此函數允許 TLE 延伸模組的使用者透過使用 ALTER EXTENSION ... UPDATE 語法更新其版本。

函數原型

```
pgtle.install_update_path(name text, fromvers text, tovers text, ext text)
```

角色

pgtle_admin

引數

- name – TLE 延伸模組的名稱。呼叫 CREATE EXTENSION 時會使用此值。
- fromvers – 用於升級之 TLE 延伸模組的來源版本。
- tovers – 用於升級之 TLE 延伸模組的目標版本。
- ext – 更新的內容。此值包含函數之類的物件。

輸出

無。

使用範例

```
SELECT pgtle.install_update_path('pg_tle_test', '0.1', '0.2',
  $_pgtle_$
    CREATE OR REPLACE FUNCTION my_test()
    RETURNS INT
    AS $$
      SELECT 21;
```

```
    $$ LANGUAGE SQL IMMUTABLE;
  $_pgtle_$
);
```

pgtle.register_feature

函數 `register_feature` 會將指定的內部 PostgreSQL 功能新增至 `pgtle.feature_info` 資料表。PostgreSQL 掛鉤是內部 PostgreSQL 功能的範例。受信任語言延伸模組開發套件支援使用 PostgreSQL 掛鉤。目前，此函數支援下列功能。

- `passcheck` – 使用自訂 PostgreSQL 密碼檢查行為的程序或函數註冊密碼檢查掛鉤。

函數原型

```
pgtle.register_feature(proc regproc, feature pg_tle_feature)
```

角色

`pgtle_admin`

引數

- `proc` – 要用於功能的預存程序或函數名稱。
- `feature` – 要向函數註冊的 `pg_tle` 功能 (例如 `passcheck`) 名稱。

輸出

無。

使用範例

```
SELECT pgtle.register_feature('pw_hook', 'passcheck');
```

pgtle.register_feature_if_not_exists

`pgtle.register_feature_if_not_exists` 函數會將指定的 PostgreSQL 功能新增至 `pgtle.feature_info` 資料表，並識別 TLE 延伸模組或其他使用該功能的程序或函數。如需掛鉤和受信任語言延伸模組的詳細資訊，請參閱 搭配您的延伸模組使用 PostgreSQL 掛鉤。

函數原型

```
pgtle.register_feature_if_not_exists(proc regproc, feature pg_tle_feature)
```

角色

pgtle_admin

引數

- proc – 預存程序或函數的名稱，此預存程序或函數包含可用作 TLE 延伸功能之功能的邏輯 (程式碼)。例如，pw_hook 程式碼。
- feature – 要針對 TLE 函數註冊的 PostgreSQL 功能名稱。目前，唯一可用的功能為 passcheck 掛鉤。如需更多詳細資訊，請參閱 密碼檢查掛鉤 (passcheck)。

輸出

在針對指定的延伸模組註冊功能之後傳回 true。如果此功能已註冊，則會傳回 false。

使用範例

```
SELECT pgtle.register_feature_if_not_exists('pw_hook', 'passcheck');
```

pgtle.set_default_version

set_default_version 函數可讓您針對 TLE 延伸模組指定 default_version。您可以使用此函數來定義升級路徑，並將該版本指定為 TLE 延伸模組的預設版本。當資料庫使用者在 CREATE EXTENSION 和 ALTER EXTENSION ... UPDATE 命令中指定您的 TLE 延伸模組時，該版本的 TLE 延伸模組就會在該使用者的資料庫中建立。

成功時此函數會傳回 true。如果 name 引數中指定的 TLE 延伸模組不存在，函數會傳回錯誤。同樣地，如果 TLE 延伸模組的 version 不存在，則其會傳回錯誤。

函數原型

```
pgtle.set_default_version(name text, version text)
```

角色

pgtle_admin

引數

- name – TLE 延伸模組的名稱。呼叫 CREATE EXTENSION 時會使用此值。
- version – 要設定預設值的 TLE 延伸模組版本。

輸出

- true – 設定預設版本成功時，函數會傳回 true。
- ERROR – 如果具有指定名稱或版本的 TLE 延伸模組不存在，則會傳回錯誤訊息。

使用範例

```
SELECT * FROM pgtle.set_default_version('my-extension', '1.1');
```

pgtle.uninstall_extension(name)

uninstall_extension 函數會從資料庫中移除 TLE 延伸模組的所有版本。此函數可防止未來呼叫 CREATE EXTENSION 時安裝 TLE 延伸模組。如果 TLE 延伸模組不存在於資料庫中，則會引發錯誤。

uninstall_extension 函數不會捨棄資料庫中目前作用中的 TLE 延伸模組。若要移除目前作用中的 TLE 延伸模組，您必須明確地呼叫 DROP EXTENSION 才能將其移除。

函數原型

```
pgtle.uninstall_extension(extname text)
```

角色

pgtle_admin

引數

- extname – 要解除安裝的 TLE 延伸模組名稱。此名稱與搭配 CREATE EXTENSION 用來載入 TLE 延伸模組，以用於指定資料庫的名稱相同。

輸出

無。

使用範例

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test');
```

pgtle.uninstall_extension(name, version)

`uninstall_extension(name, version)` 函數會從資料庫中移除 TLE 延伸模組的指定版本。此函數可防止 `CREATE EXTENSION` 和 `ALTER EXTENSION` 將 TLE 延伸模組安裝或更新為指定版本。此函數也會移除 TLE 延伸模組指定版本的所有更新路徑。如果 TLE 延伸模組目前在資料庫中作用中，則此函數無法將其解除安裝。您必須明確地呼叫 `DROP EXTENSION` 才能移除 TLE 延伸模組。若要解除安裝 TLE 延伸模組的所有版本，請參閱 pgtle.uninstall_extension(name)。

函數原型

```
pgtle.uninstall_extension(extname text, version text)
```

角色

`pgtle_admin`

引數

- `extname` – TLE 延伸模組的名稱。呼叫 `CREATE EXTENSION` 時會使用此值。
- `version` – 要從資料庫中解除安裝的 TLE 延伸模組版本。

輸出

無。

使用範例

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test', '0.2');
```

pgtle.uninstall_extension_if_exists

`uninstall_extension_if_exists` 函數會從指定資料庫中移除 TLE 延伸模組的所有版本。如果 TLE 延伸模組不存在，則函數會以無訊息方式返回 (不會引發任何錯誤訊息)。如果指定的延伸模組目前在資料庫內作用中，則此函數不會捨棄它。您必須明確地呼叫 `DROP EXTENSION` 來移除 TLE 延伸模組，然後再使用此函數來解除安裝其成品。

函數原型

```
pgtle.uninstall_extension_if_exists(extname text)
```

角色

pgtle_admin

引數

- extname – TLE 延伸模組的名稱。呼叫 CREATE EXTENSION 時會使用此值。

輸出

在解除安裝指定的延伸模組之後，uninstall_extension_if_exists 函數會傳回 true。如果指定的延伸模組不存在，函數會傳回 false。

- true – 在解除安裝 TLE 延伸模組之後傳回 true。
- false – 當 TLE 延伸模組不存在於資料庫中時傳回 false。

使用範例

```
SELECT * FROM pgtle.uninstall_extension_if_exists('pg_tle_test');
```

pgtle.uninstall_update_path

uninstall_update_path 函數會從 TLE 延伸模組中移除特定的更新路徑。這樣可以防止 ALTER EXTENSION ... UPDATE TO 將其用作更新路徑。

如果 TLE 延伸模組目前正由這個更新路徑上的其中一個版本使用，則其仍會保留在資料庫中。

如果指定的更新路徑不存在，此函數會引發錯誤。

函數原型

```
pgtle.uninstall_update_path(extname text, fromvers text, tovers text)
```

角色

pgtle_admin

引數

- extname – TLE 延伸模組的名稱。呼叫 CREATE EXTENSION 時會使用此值。
- fromvers – 更新路徑上使用之 TLE 延伸模組的來源版本。
- tovers – 更新路徑上使用之 TLE 延伸模組的目標版本。

輸出

無。

使用範例

```
SELECT * FROM pgtle.uninstall_update_path('pg_tle_test', '0.1', '0.2');
```

pgtle.uninstall_update_path_if_exists

uninstall_update_path_if_exists 函數類似於 uninstall_update_path，在這裡其會從 TLE 延伸模組中移除指定的更新路徑。不過，如果更新路徑不存在，則此函數不會引發錯誤訊息。反之，此函數會傳回 false。

函數原型

```
pgtle.uninstall_update_path_if_exists(extname text, fromvers text, tovers text)
```

角色

pgtle_admin

引數

- extname – TLE 延伸模組的名稱。呼叫 CREATE EXTENSION 時會使用此值。
- fromvers – 更新路徑上使用之 TLE 延伸模組的來源版本。
- tovers – 更新路徑上使用之 TLE 延伸模組的目標版本。

輸出

- true – 函數已成功更新 TLE 延伸模組的路徑。
- false – 函數無法更新 TLE 延伸模組的路徑。

使用範例

```
SELECT * FROM pgtle.uninstall_update_path_if_exists('pg_tle_test', '0.1', '0.2');
```

pgtle.unregister_feature

unregister_feature 函數會提供一種方法，來刪除已註冊為使用 pg_tle 功能 (例如掛鉤) 的函數。如需註冊功能的相關資訊，請參閱 pgtle.register_feature。

函數原型

```
pgtle.unregister_feature(proc regproc, feature pg_tle_features)
```

角色

pgtle_admin

引數

- proc – 要向 pg_tle 功能註冊的預存函數名稱。
- feature – 要向函數註冊的 pg_tle 功能名稱。例如，passcheck 是一種功能，可以將其註冊以供您開發的受信任語言延伸模組使用。如需更多詳細資訊，請參閱 密碼檢查掛鉤 (passcheck)。

輸出

無。

使用範例

```
SELECT * FROM pgtle.unregister_feature('pw_hook', 'passcheck');
```

pgtle.unregister_feature_if_exists

unregister_feature 函數會提供一種方法，來刪除已註冊為使用 pg_tle 功能 (例如掛鉤) 的函數。如需更多詳細資訊，請參閱 搭配您的延伸模組使用 PostgreSQL 掛鉤。在成功取消註冊功能之後傳回 true。如果功能未註冊，則會傳回 false。

如需為 TLE 延伸模組註冊 pg_tle 功能的相關資訊，請參閱 pgtle.register_feature。

函數原型

```
pgtle.unregister_feature_if_exists('proc regproc', 'feature pg_tle_features')
```

角色

pgtle_admin

引數

- proc – 已註冊來包含 pg_tle 功能的預存函數名稱。
- feature – 已向受信任語言延伸模組註冊的 pg_tle 功能名稱。

輸出

傳回 true 或 false，如下所示。

- true – 函數已成功從延伸模組中取消註冊功能。
- false – 函數無法從 TLE 延伸模組中取消註冊功能。

使用範例

```
SELECT * FROM pgtle.unregister_feature_if_exists('pw_hook', 'passcheck');
```

適用於 PostgreSQL 的受信任語言延伸模組的掛鉤參考

適用於 PostgreSQL 的受信任語言延伸模組支援 PostgreSQL 掛鉤。掛鉤是開發人員可用於擴充 PostgreSQL 核心功能的內部回呼機制。透過使用掛鉤，開發人員可以實作自己的函數或程序，以在各種資料庫操作期間使用，從而以某種方式修改 PostgreSQL 的行為。例如，您可以使用 passcheck 掛鉤，來自訂 PostgreSQL 在為使用者 (角色) 建立或變更密碼時如何處理所提供的密碼。

檢視下列文件，以了解 TLE 延伸模組可用的 passcheck 勾點。若要進一步了解可用的勾點，包括用戶端驗證勾點，請參閱受信任語言延伸模組勾點。

密碼檢查掛鉤 (passcheck)

passcheck 掛鉤用來針對下列 SQL 命令和 psql 中繼命令自訂密碼檢查過程中的 PostgreSQL 行為。

- CREATE ROLE *username* ...PASSWORD – 如需詳細資訊，請參閱 PostgreSQL 文件中的 CREATE ROLE。
- ALTER ROLE *username*...PASSWORD – 如需詳細資訊，請參閱 PostgreSQL 文件中的 ALTER ROLE。
- \password *username* – 此互動式 psql 中繼命令透過在透明地使用 ALTER ROLE ... PASSWORD 語法之前對密碼進行雜湊處理，來安全地變更所指定使用者的密碼。中繼命令是 ALTER ROLE ... PASSWORD 命令的安全包裝函式，因此掛鉤適用於 psql 中繼命令的行為。

如需範例，請參閱 密碼檢查掛鉤程式碼清單。

內容

- 函數原型
- 引數
- Configuration
- 使用須知

函數原型

```
passcheck_hook(username text, password text, password_type pgtle.password_types,
 valid_until timestamptz, valid_null boolean)
```

引數

passcheck 掛鉤函數採用下列引數：

- username – 設定密碼之角色 (使用者名稱) 的名稱 (以文字形式表示)。
- password – 純文字或雜湊密碼。輸入的密碼應符合 password_type 中指定的類型。
- password_type – 指定密碼的 pgtle.password_type 格式。此 格式可以是下列其中一個選項：
  - PASSWORD_TYPE_PLAINTEXT – 純文字密碼。
  - PASSWORD_TYPE_MD5 – 使用 MD5 (訊息摘要 5) 演算法進行雜湊處理的密碼。
  - PASSWORD_TYPE_SCRAM_SHA_256 – 使用 SCRAM-SHA-256 演算法進行雜湊處理的密碼。
- valid_until – 指定密碼變成無效的時間。此為選用引數。如果使用此引數，請將時間指定為 timestamptz 值。
- valid_null – 如果將此布林值設為 true，則 valid_until 選項會設為 NULL。

Configuration

此函數 pgtle.enable_password_check 可控制 passcheck 掛鉤是否作用中。passcheck 掛鉤有三種可能的設定。

- off – 關閉 passcheck 密碼檢查掛鉤。這是預設值。
- on – 開啟 passcode 密碼檢查掛鉤，以便針對資料表檢查密碼。
- require – 需要定義密碼檢查掛鉤。

使用須知

若要開啟或關閉 passcheck 掛鉤，您需要針對 Aurora PostgreSQL 資料庫叢集的寫入器執行個體修改自訂資料庫參數群組。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

在 Windows 中：

```
aws rds modify-db-parameter-group ^
    --region aws-region ^
    --db-parameter-group-name your-custom-parameter-group ^
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

Amazon Aurora PostgreSQL 參考

在下列主題中，您可以找到 Amazon Aurora PostgreSQL 中與定序、函數、參數和等待事件相關的資訊。

主題

- 適用於 EBCDIC 和其他大型主機遷移的 Aurora PostgreSQL 定序
- Aurora PostgreSQL 支援的定序
- Aurora PostgreSQL 函數參考
- Amazon Aurora PostgreSQL 參數
- Amazon Aurora PostgreSQL 等待事件

適用於 EBCDIC 和其他大型主機遷移的 Aurora PostgreSQL 定序

將大型主機應用程式遷移至新平台 (例如 AWS) 理想地保留應用程式行為。若要在新平台上保留應用程式行為與大型主機完全相同，需要使用相同的定序和排序規則對遷移的資料進行整理。例如，許多 Db2 遷移解決方案將空值轉移到 u0180 (Unicode 位置 0180)，因此這些定序首先對 u0180 進行排序。這是定序與其大型主機來源有所不同，以及為何必須選擇能夠更好地對應至原始 EBCDIC 定序的定序規則的範例。

Aurora PostgreSQL 14.3 及更高版本提供許多 ICU 和 EBCDIC 定序，以支援使用 AWS Mainframe Modernization 服務進行到 AWS 的此種遷移。若要進一步了解此服務，請參閱什麼是 AWSMainframe Modernization？

您可以在下表中找到 Aurora PostgreSQL - 提供的定序。這些定序遵循 EBCDIC 規則，並確保大型主機應用程式在 AWS 上的功能與大型主機環境中相同。定序名稱包含相關字碼頁，(cpnnnn)，以便您可以為您的大型主機來源選擇適當的定序規則。例如，使用 `en-US-cp037-x-icu` 以達到起源於使用字碼頁 037 之大型主機應用程式的 EBCDIC 資料的定序行為。

EBCDIC 定序	AWS Blu Age 定序	AWS Micro Focus 定序
da-DK-cp1142-x-icu	da-DK-cp1142b-x-icu	da-DK-cp1142m-x-icu
da-DK-cp277-x-icu	da-DK-cp277b-x-icu	–
de-DE-cp1141-x-icu	de-DE-cp1141b-x-icu	de-DE-cp1141m-x-icu

EBCDIC 定序	AWS Blu Age 定序	AWS Micro Focus 定序
de-DE-cp273-x-icu	de-DE-cp273b-x-icu	–
en-GB-cp1146-x-icu	en-GB-cp1146b-x-icu	en-GB-cp1146m-x-icu
en-GB-cp285-x-icu	en-GB-cp285b-x-icu	–
en-US-cp037-x-icu	en-US-cp037b-x-icu	–
en-US-cp1140-x-icu	en-US-cp1140b-x-icu	en-US-cp1140m-x-icu
es-ES-cp1145-x-icu	es-ES-cp1145b-x-icu	es-ES-cp1145m-x-icu
es-ES-cp284-x-icu	es-ES-cp284b-x-icu	–
fi-FI-cp1143-x-icu	fi-FI-cp1143b-x-icu	fi-FI-cp1143m-x-icu
fi-FI-cp278-x-icu	fi-FI-cp278b-x-icu	–
fr-FR-cp1147-x-icu	fr-FR-cp1147b-x-icu	fr-FR-cp1147m-x-icu
fr-FR-cp297-x-icu	fr-FR-cp297b-x-icu	–
it-IT-cp1144-x-icu	it-IT-cp1144b-x-icu	it-IT-cp1144m-x-icu
it-IT-cp280-x-icu	it-IT-cp280b-x-icu	–
nl-BE-cp1148-x-icu	nl-BE-cp1148b-x-icu	nl-BE-cp1148m-x-icu
nl-BE-cp500-x-icu	nl-BE-cp500b-x-icu	–

若要進一步了解 AWS Blu Age，請參閱 AWSMainframe Modernization 使用者指南中的教學課程：AWS Blu Age 的受管執行階段。

如需使用 AWS Micro Focus 的詳細資訊，請參閱 AWSMainframe Modernization 使用者指南中的教學課程：Micro Focus 的受管執行階段。

若要進一步了解管理 PostgreSQL 中的定序，請參閱 PostgreSQL 文件中的定序支援。

Aurora PostgreSQL 支援的定序

定序是一組規則，用來決定如何排序和比較存放在資料庫中的字元字串。定序在電腦系統中扮演著基本角色，並包含為作業系統的一部分。隨著新字元加入至語言或排序規則變更時，定序也會變更。

定序程式庫會定義定序的特定規則和演算法。PostgreSQL 內使用的最熱門定序程式庫是 GNU C (glibc)和 Unicode 國際化元件 (ICU)。依預設，Aurora PostgreSQL 會使用 glibc 定序，其中包含多位元組字元序列的 Unicode 字元排序順序。

當您建立新的 Aurora PostgreSQL DB 叢集 時，它會檢查作業系統是否有可用的定序。CREATE DATABASE 命令 LC_COLLATE 和 LC_CTYPE 的 PostgreSQL 參數用來指定一個定序，其作為該資料庫中的預設定序。或者，您也可以使用 CREATE DATABASE 中的 LOCALE 參數來設定這些參數。這會決定資料庫中字元字串的預設定序，以及將字元分類為字母、數字或符號的規則。您也可以選擇要在資料欄、索引或查詢上使用的定序。

Aurora PostgreSQL 取決於作業系統中的 glibc 程式庫是否支援定序。Aurora PostgreSQL 執行個體會定期使用最新版本的作業系統進行更新。這些更新有時會包含較新版本的 glibc 程式庫。較新版本的 glibc 很少會變更某些字元的排序順序或定序，這可能導致資料以不同的方式排序或產生無效的索引項目。如果您在更新期間發現定序的排序順序問題，您可能需要重建索引。

為了 glibc 更新可能造成的影響，Aurora PostgreSQL 現在包含獨立的預設定序程式庫。此定序程式庫可在 Aurora PostgreSQL 14.6、13.9、12.13、11.18 和較新的次要版本中使用。它與 glibc 2.26-59.amzn2 相容，並提供排序順序穩定性以防止出現不正確的查詢結果。

Aurora PostgreSQL 函數參考

您可以在下文找到 Aurora PostgreSQL 函數的清單，這些函數適用於執行 Aurora PostgreSQL 相容版本資料庫引擎的 Aurora 資料庫叢集。除了標準 PostgreSQL 函數之外，還有這些 Aurora PostgreSQL 函數。如需標準 PostgreSQL 函數的詳細資訊，請參閱 PostgreSQL–函數和運算子。

概觀

您可對執行 Aurora PostgreSQL 的 Amazon RDS 資料庫執行個體使用下列函數：

- aurora_db_instance_identifier
- aurora_ccm_status
- aurora_global_db_instance_status
- aurora_global_db_status

- aurora_list_builtins
- aurora_replica_status
- aurora_stat_activity
- aurora_stat_backend_waits
- aurora_stat_bgwriter
- aurora_stat_database
- aurora_stat_dml_activity
- aurora_stat_get_db_commit_latency
- aurora_stat_logical_wal_cache
- aurora_stat_memctx_usage
- aurora_stat_optimized_reads_cache
- aurora_stat_plans
- aurora_stat_reset_wal_cache
- aurora_stat_statements
- aurora_stat_system_waits
- aurora_stat_wait_event
- aurora_stat_wait_type
- aurora_version
- aurora_volume_logical_start_lsn
- aurora_wait_report

aurora_db_instance_identifier

報告您所連線的資料庫執行個體名稱。

語法

```
aurora_db_instance_identifier()
```

引數

無

傳回類型

VARCHAR 字串

使用須知

此函數會顯示 Aurora PostgreSQL 相容版叢集的資料庫執行個體名稱，適用於您的資料庫用戶端或應用程式連線。

此函數從 PostgreSQL 版本 13.7、12.11、11.16、10.21 及所有其他更新的版本開始提供。

範例

下列範例顯示呼叫 `aurora_db_instance_identifier` 函數的結果。

```
=> SELECT aurora_db_instance_identifier();
aurora_db_instance_identifier
------------------------------
 test-my-instance-name
```

您可以將此函數的結果與 `aurora_replica_status` 函數聯結，以取得連線資料庫執行個體的詳細資訊。只有 aurora_replica_status 並不能提供向您顯示您正在使用的資料庫執行個體。下列範例會顯示作法。

```
=> SELECT *
    FROM aurora_replica_status() rt,
         aurora_db_instance_identifier() di
    WHERE rt.server_id = di;
-[ RECORD 1 ]----------------------+-----------------------
server_id                        | test-my-instance-name
session_id                       | MASTER_SESSION_ID
durable_lsn                      | 88492069
highest_lsn_rcvd                 |
current_read_lsn                 |
cur_replay_latency_in_usec       |
active_txns                      |
is_current                       | t
last_transport_error             | 0
last_error_timestamp             |
last_update_timestamp            | 2022-06-03 11:18:25+00
feedback_xmin                    |
feedback_epoch                   |
replica_lag_in_msec              |
```

```
log_stream_speed_in_kib_per_second | 0
log_buffer_sequence_number         | 0
oldest_read_view_trx_id            |
oldest_read_view_lsn               |
pending_read_ios                   | 819
```

aurora_ccm_status

顯示叢集快取管理員的狀態。

語法

```
aurora_ccm_status()
```

引數

無。

傳回類型

SETOF 記錄，包含下列欄：

- buffers_sent_last_minute – 過去一分鐘內傳送至指定讀取器的緩衝區數量。
- buffers_found_last_minute – 過去一分鐘內識別出頻繁存取緩衝區的數量。
- buffers_sent_last_scan – 於緩衝區快取的最後一次完整掃描期間傳送至指定讀取器的緩衝區數量。
- buffers_found_last_scan – 於緩衝區快取的最後一次完整掃描期間傳送的頻繁存取緩衝區數量。已在指定讀取器上快取的緩衝區未傳送。
- buffers_sent_current_scan – 於目前掃描期間傳送的緩衝區數量。
- buffers_found_current_scan – 於目前掃描中識別之頻繁存取緩衝區的數量。
- current_scan_progress – 於目前掃描期間至現在為止所造訪的緩衝區數量。

使用須知

您可使用此功能來檢查和監控叢集快取管理 (CCM) 功能。此函數僅適用於 CCM 在 Aurora PostgreSQL 資料庫叢集上為作用中時。如要使用此函數，請連線至 Aurora PostgreSQL 資料庫叢集上的寫入資料庫執行個體。

您可於叢集的自訂資料庫叢集參數群組中將 apg_ccm_enabled 設定為 1，為 Aurora PostgreSQL 資料庫叢集開啟 CCM。如要瞭解如何作業，請參閱設定叢集快取管理。

當叢集具有 Aurora 讀取器執行個體設定如下時，叢集快取管理在 Aurora PostgreSQL 資料庫叢集為作用中狀態：

- Aurora 讀取器執行個體會使用與叢集寫入器執行個體相同的資料庫執行個體類別類型和大小。
- Aurora 讀取器執行個體會設定為叢集的第 0 層。若叢集具有多個讀取器，則此為其唯一的第 0 層讀取器。

將多個讀取器設定為第 0 層將會停用 CCM。停用 CCM 時，呼叫此函數會回傳下列錯誤訊息：

```
ERROR: Cluster Cache Manager is disabled
```

您還可使用 PostgreSQL Pg_Buffercache 擴充功能來分析緩衝區快取。如需詳細資訊，請參閱 PostgreSQL 文件中的 pg_buffercache。

如需詳細資訊，請參閱 Introduction to Aurora PostgreSQL cluster cache management (Aurora PostgreSQL 叢集快取管理簡介)。

範例

下列範例顯示呼叫 aurora_ccm_status 函數的結果。第一個範例顯示 CCM 統計資料。

```
=> SELECT * FROM aurora_ccm_status();
 buffers_sent_last_minute | buffers_found_last_minute | buffers_sent_last_scan |
 buffers_found_last_scan | buffers_sent_current_scan | buffers_found_current_scan |
 current_scan_progress
--------------------------+---------------------------+------------------------
+-------------------------+---------------------------+----------------------------
+-----------------------
                   2242000 |                   2242003 |               17920442 |
         17923410 |                  14098000 |                  14100964 |
   15877443
```

如需更多完整的詳細資訊，您可使用展開顯示，如下所示：

```
\x
Expanded display is on.
SELECT *  FROM aurora_ccm_status();
[ RECORD 1 ]----------------------+---------
```

```
buffers_sent_last_minute           | 2242000
buffers_found_last_minute          | 2242003
buffers_sent_last_scan             | 17920442
buffers_found_last_scan            | 17923410
buffers_sent_current_scan          | 14098000
buffers_found_current_scan         | 14100964
current_scan_progress              | 15877443
```

此範例說明如何檢查暖率和暖百分比。

```
=> SELECT buffers_sent_last_minute * 8/60 AS warm_rate_kbps,
100 * (1.0-buffers_sent_last_scan/buffers_found_last_scan) AS warm_percent
FROM aurora_ccm_status ();
 warm_rate_kbps | warm_percent
----------------+--------------
 16523 |        100.0
```

aurora_global_db_instance_status

顯示所有 Aurora 執行個體的狀態，包括 Aurora 全域資料庫叢集中的複本。

語法

```
aurora_global_db_instance_status()
```

引數

無

傳回類型

SETOF 記錄，包含下列欄：

- server_id – 資料庫執行個體的識別符。
- session_id – 目前工作階段的唯一識別符。MASTER_SESSION_ID 值可識別寫入器 (主) 資料庫執行個體。
- aws_region – 此全域資料庫執行個體執行所在的 AWS 區域。如需區域清單，請參閱 區域可用性。
- durable_lsn – 可長期儲存的日誌序號 (LSN)。記錄序號 (LSN) 是一組獨特的序號，可用來識別資料庫交易日誌中的記錄。系統會排序 LSN，而 LSN 越大，就表示交易發生時間越後面。

- highest_lsn_rcvd – 資料庫執行個體從寫入器資料庫執行個體接收的最高 LSN。
- feedback_epoch – 資料庫執行個體產生熱待命資訊時所使用的 epoch。熱待命是主資料庫處於復原或待命模式時支援連線和查詢的資料庫執行個體。熱待命資訊包括 epoch (時間點) 及有關用來作為熱待命資料庫執行個體的其他詳細資料。如需詳細資訊，請參閱 PostgreSQL 文件中的 Hot standby (熱待命)。
- feedback_xmin – 資料庫執行個體所使用的最小 (最舊) 作用中交易 ID。
- oldest_read_view_lsn – 資料庫執行個體從儲存體中讀取資料時所使用的最舊 LSN。
- visibility_lag_in_msec – 此資料庫執行個體落後於寫入器資料庫執行個體的時間 (以毫秒為單位)。

使用須知

此函數會顯示 Aurora 資料庫叢集的複寫統計資料。對於叢集中的每個 Aurora PostgreSQL 資料庫執行個體，此函數會顯示一列資料，其中包括全域資料庫組態中的任何跨區域複本。

您可從 Aurora PostgreSQL 資料庫叢集或 Aurora PostgreSQL 全域資料庫中的任何執行個體執行此函數。該函數會回傳所有複本執行個體的延遲相關詳細資料。

若要進一步了解使用此函數 (aurora_global_db_instance_status) 或使用 aurora_global_db_status 監控延遲的相關資訊，請參閱 監控 Aurora PostgreSQL 型全球資料庫。

如需更多 Aurora 全域資料庫的詳細資訊，請參閱 Amazon Aurora 全球資料庫的概觀。

若要開始使用 Aurora 全域資料庫，請參閱 Amazon Aurora 全球資料庫入門，或請參閱 Amazon Aurora 常見問答集。

範例

此範例示範跨區域執行個體統計資料。

```
=> SELECT *
  FROM aurora_global_db_instance_status();
               server_id                |              session_id              |
  aws_region   | durable_lsn | highest_lsn_rcvd | feedback_epoch | feedback_xmin |
 oldest_read_view_lsn | visibility_lag_in_msec
--------------------------------------+--------------------------------------
+--------------+-------------+------------------+----------------+--------------
+----------------------+------------------------
```

```
 db-119-001-instance-01                   | MASTER_SESSION_ID                    | eu-
west-1    |  2534560273 |           [NULL] |         [NULL] |        [NULL] |
     [NULL] |                 [NULL]
 db-119-001-instance-02                   | 4ecff34d-d57c-409c-ba28-278b31d6fc40 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560266 |                      6
 db-119-001-instance-03                   | 3e8a20fc-be86-43d5-95e5-bdf19d27ad6b | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560266 |                      6
 db-119-001-instance-04                   | fc1b0023-e8b4-4361-bede-2a7e926cead6 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560254 |                     23
 db-119-001-instance-05                   | 30319b74-3f08-4e13-9728-e02aa1aa8649 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560254 |                     23
 db-119-001-global-instance-1             | a331ffbb-d982-49ba-8973-527c96329c60 | eu-
central-1 |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                    996
 db-119-001-global-instance-1             | e0955367-7082-43c4-b4db-70674064a9da | eu-
west-2    |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                     14
 db-119-001-global-instance-1-eu-west-2a  | 1248dc12-d3a4-46f5-a9e2-85850491a897 | eu-
west-2    |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                      0
```

此範例示範如何檢查全域複本延遲 (以毫秒為單位)。

```
=> SELECT CASE
          WHEN 'MASTER_SESSION_ID' = session_id THEN 'Primary'
         ELSE 'Secondary'
      END AS global_role,
      aws_region,
      server_id,
      visibility_lag_in_msec
  FROM aurora_global_db_instance_status()
  ORDER BY 1, 2, 3;
   global_role |  aws_region  |               server_id                |
 visibility_lag_in_msec
-------------+--------------+----------------------------------------
+------------------------
 Primary     | eu-west-1    | db-119-001-instance-01                 |
 [NULL]
```

```
 Secondary   | eu-central-1 | db-119-001-global-instance-1            |
     13
 Secondary   | eu-west-1    | db-119-001-instance-02                  |
     10
 Secondary   | eu-west-1    | db-119-001-instance-03                  |
      9
 Secondary   | eu-west-1    | db-119-001-instance-04                  |
      2
 Secondary   | eu-west-1    | db-119-001-instance-05                  |
     18
 Secondary   | eu-west-2    | db-119-001-global-instance-1            |
     14
 Secondary   | eu-west-2    | db-119-001-global-instance-1-eu-west-2a |
     13
```

此範例示範如何從全域資料庫組態檢查每個 AWS 區域 的最小、最大，及平均延遲。

```
=> SELECT 'Secondary' global_role,
       aws_region,
       min(visibility_lag_in_msec) min_lag_in_msec,
       max(visibility_lag_in_msec) max_lag_in_msec,
       round(avg(visibility_lag_in_msec),0) avg_lag_in_msec
  FROM aurora_global_db_instance_status()
 WHERE aws_region NOT IN (SELECT   aws_region
                            FROM aurora_global_db_instance_status()
                           WHERE session_id='MASTER_SESSION_ID')
                        GROUP BY aws_region
UNION ALL
SELECT  'Primary' global_role,
        aws_region,
        NULL,
        NULL,
        NULL
    FROM aurora_global_db_instance_status()
   WHERE session_id='MASTER_SESSION_ID'
ORDER BY 1, 5;
 global_role |  aws_region  | min_lag_in_msec | max_lag_in_msec | avg_lag_in_msec
-------------+--------------+-----------------+-----------------+-----------------
 Primary     | eu-west-1    |          [NULL] |          [NULL] |          [NULL]
 Secondary   | eu-central-1 |             133 |             133 |             133
 Secondary   | eu-west-2    |               0 |             495 |             248
```

aurora_global_db_status

顯示有關 Aurora 全域資料庫延遲各方面的資訊，特別是基礎 Aurora 儲存的延遲 (稱為持久性延遲) 及復原點目標 (RPO) 之間的延遲。

語法

```
aurora_global_db_status()
```

引數

無。

傳回類型

SETOF 記錄，包含下列欄：

- `aws_region` – 此資料庫叢集所在的 AWS 區域。如需依引擎列出的 AWS 區域 完整清單，請參閱區域和可用區域。
- `highest_lsn_written` – 目前存在於此資料庫叢集上的最高日誌序號 (LSN)。記錄序號 (LSN) 是一組獨特的序號，可用來識別資料庫交易日誌中的記錄。系統會排序 LSN，而 LSN 越大，就表示交易發生時間越後面。
- `durability_lag_in_msec` – 次要資料庫叢集上的 `highest_lsn_written` 與主資料庫叢集上的 `highest_lsn_written` 之間的時間戳記值差異。值 -1 識別 Aurora 全域資料庫的主要資料庫叢集。
- `rpo_lag_in_msec` – 復原點目標 (RPO) 延遲。RPO 延遲是指最近使用者交易 COMMIT 在儲存於 Aurora 全域資料庫的主資料庫叢集之後，將其儲存於次要資料庫叢集上所需的時間。值 -1 表示主要資料庫叢集 (因此沒有延遲的問題)。

  簡言之，此指標會計算 Aurora 全域資料庫中每個 Aurora PostgreSQL 資料庫叢集的復原點目標，亦即，若有中斷發生，可能會遺失的資料量。與延遲一樣，RPO 是以時間來衡量。
- `last_lag_calculation_time` – 指出上次為 `durability_lag_in_msec` 和 `rpo_lag_in_msec` 計算值的時間戳記。例如 `1970-01-01 00:00:00+00` 之類的時間值表示此為主要資料庫叢集。
- `feedback_epoch` – 次要資料庫叢集產生熱待命資訊時所使用的 epoch。熱待命是主資料庫為復原或待命模式時支援連線和查詢的資料庫執行個體。熱待命資訊包括 epoch (時間點) 及有關用來作為

熱待命資料庫執行個體的其他詳細資料。如需詳細資訊，請參閱 PostgreSQL 文件中的 Hot standby (熱待命)。

- `feedback_xmin` – 次要資料庫叢集所使用的最小 (最舊) 作用中交易 ID。

使用須知

所有目前可用的 Aurora PostgreSQL 版本都支援此功能。此函數會顯示 Aurora 全域資料庫的複寫統計資料。Aurora PostgreSQL 全域資料庫中每個資料庫叢集會各自顯示為一列。您可從 Aurora PostgreSQL 全域資料庫中的任何執行個體執行此函數。

如要評估 Aurora 全域資料庫複製延遲 (此為可見的資料延遲)，請參閱 aurora_global_db_instance_status。

若要進一步了解使用 `aurora_global_db_status` 和 `aurora_global_db_instance_status` 來監控 Aurora 全域資料庫延遲，請參閱 監控 Aurora PostgreSQL 型全球資料庫。如需更多 Aurora 全域資料庫的詳細資訊，請參閱 Amazon Aurora 全球資料庫的概觀。

範例

此範例示範如何顯示跨區域儲存統計資料。

```
=> SELECT CASE
          WHEN '-1' = durability_lag_in_msec THEN 'Primary'
          ELSE 'Secondary'
       END AS global_role,
       *
  FROM aurora_global_db_status();
 global_role | aws_region | highest_lsn_written | durability_lag_in_msec |
 rpo_lag_in_msec | last_lag_calculation_time  | feedback_epoch | feedback_xmin
-------------+------------+---------------------+------------------------
+-----------------+---------------------------+----------------+---------------
 Primary     | eu-west-1  |           131031557 |                     -1 |
 -1 | 1970-01-01 00:00:00+00     |             0 |              0
 Secondary   | eu-west-2  |           131031554 |                    410 |
  0 | 2021-06-01 18:59:36.124+00 |             0 |         12640
 Secondary   | eu-west-3  |           131031554 |                    410 |
  0 | 2021-06-01 18:59:36.124+00 |             0 |         12640
```

aurora_list_builtins

列出所有可用的 Aurora PostgreSQL 內建函數，以及簡要說明和函數詳細資訊。

語法

```
aurora_list_builtins()
```

引數

無

傳回類型

SETOF 記錄

範例

下列範例會顯示呼叫 aurora_list_builtins 函數的結果。

```
=> SELECT *
FROM aurora_list_builtins();

               Name                | Result data type |                   Argument data
 types                     | Type | Volatility |  Parallel  | Security |
             Description
-----------------------------------+------------------
+-----------------------------------------------------------+------+------------
+------------+----------
+-------------------------------------------------------------------------
 aurora_version                    | text             |
                           | func | stable     | safe       | invoker  | Amazon Aurora
 PostgreSQL-Compatible Edition version string
 aurora_stat_wait_type             | SETOF record     | OUT type_id smallint, OUT
 type_name text                 | func | volatile   | restricted | invoker  | Lists all
 supported wait types
 aurora_stat_wait_event            | SETOF record     | OUT type_id smallint, OUT
 event_id integer, OUT event_na.| func | volatile   | restricted | invoker  | Lists all
 supported wait events
                                   |                  |.me text
                           |      |            |            |          |
 aurora_list_builtins              | SETOF record     | OUT "Name" text, OUT "Result
 data type" text, OUT "Argum.| func | stable     | safe       | invoker  | Lists all
 Aurora built-in functions
                                   |                  |.ent data types" text, OUT
 "Type" text, OUT "Volatility" .|      |            |            |          |
```

```
                                    |                  |.text, OUT "Parallel" text, OUT
"Security" text, OUT "Des.|      |            |            |          |
                                    |                  |.cription" text
                          |      |            |            |          |
.
.
.
aurora_stat_file                    | SETOF record     | OUT filename text, OUT
allocated_bytes bigint, OUT used_.| func | stable     | safe       | invoker  | Lists
all files present in Aurora storage
                                    |                  |.bytes bigint
                          |      |            |            |          |
aurora_stat_get_db_commit_latency | bigint           | oid
                           | func | stable     | restricted | invoker  | Per DB commit
latency in microsecs
```

aurora_replica_status

顯示所有 Aurora PostgreSQL 讀取器節點的狀態。

語法

```
aurora_replica_status()
```

引數

無

傳回類型

SETOF 記錄，包含下列欄：

- server_id – 資料庫執行個體 ID (識別符)。
- session_id – 目前工作階段的唯一識別符，如下所示，為主要執行個體和讀取器執行個體傳回：
  - 若為主要執行個體，session_id 一律為 `MASTER_SESSION_ID’。
  - 若為讀取器執行個體，session_id 一律為讀取器執行個體的 UUID (通用唯一識別符)。
- durable_lsn – 已儲存於儲存體中的日誌序號 (LSN)。
  - 若為主磁碟區，目前作用中的主磁碟區耐用性 LSN (VDL)。
  - 若為任何次要磁碟區，次要已成功套用至之主要的 VDL。

> **Note**
>
> 記錄序號 (LSN) 是一組獨特的序號，可用來識別資料庫交易日誌中的記錄。LSN 依順序排列，LSN 越大，就表示交易發生時間越後面。

- `highest_lsn_rcvd` – 資料庫執行個體從寫入器資料庫執行個體接收的最高 (最新) LSN。
- `current_read_lsn` – 套用至所有讀取器之最新快照的 LSN。
- `cur_replay_latency_in_usec` – 預期在次要設備上重播日誌所需的微秒數。
- `active_txns` – 目前作用中交易的數目。
- `is_current` – 未使用。
- `last_transport_error` – 上次複寫錯誤代碼。
- `last_error_timestamp` – 上次複寫錯誤的時間戳記。
- `last_update_timestamp` – 上次更新複本狀態的時間戳記。從 Aurora PostgreSQL 13.9 開始，您連線的資料庫執行個體的 `last_update_timestamp` 值會設定為 NULL。
- `feedback_xmin` – 複本的熱待命 feedback_xmin。資料庫執行個體所使用的最小 (最舊) 作用中交易 ID。
- `feedback_epoch` – 資料庫執行個體產生熱待命資訊時所使用的 epoch。
- `replica_lag_in_msec`：讀取器執行個體延遲於執行個體的時間 (以毫秒為單位)。
- `log_stream_speed_in_kib_per_second` – 日誌串流輸送量 (以 KB/秒為單位)。
- `log_buffer_sequence_number` – 日誌緩衝區序列號。
- `oldest_read_view_trx_id` – 未使用。
- `oldest_read_view_lsn` – 資料庫執行個體從儲存體中讀取資料時所使用的最舊 LSN。
- `pending_read_ios` – 複本上仍處於待命狀態的未完成頁面讀取。
- `read_ios` – 複本上的頁面讀取總數。
- `iops` – 未使用。
- `cpu`：叢集中每個節點的 Aurora 儲存常駐程式 CPU 用量。如需有關執行個體 CPU 用量的資訊，請參閱 Amazon Aurora 的執行個體層級指標。

使用須知

所有目前可用的 Aurora PostgreSQL 版本都支援此功能。`aurora_replica_status` 函數會從 Aurora PostgreSQL 資料庫叢集的複本狀態管理器傳回值。您可使用此函數以獲取有關 Aurora

PostgreSQL 資料庫叢集上複寫狀態的資訊，包括 Aurora 資料庫叢集中所有資料庫執行個體的指標。例如，您可以執行下列動作：

- 取得有關 Aurora PostgreSQL 資料庫叢集中執行個體類型 (寫入器、讀取器) 的資訊 – 您可檢查下列欄的值以取得此資訊：
  - `server_id` – 包含您在建立執行個體時指定的執行個體名稱。在某些情況下，例如對於主 (寫入器) 執行個體，該名稱通常是將 -instance-1 附加至您為 Aurora PostgreSQL 資料庫叢集建立的名稱來建立的。
  - `session_id` – `session_id` 欄位指出執行個體是讀取器還是寫入器。若為寫入器執行個體，`session_id` 一律設定為 `"MASTER_SESSION_ID"`。若為讀取器執行個體，`session_id` 設定為特定讀取器的 UUID。
- 診斷常見複寫問題，如複本延遲 - 複本延遲是讀取器執行個體頁面快取落後於寫入器執行個體頁面快取的時間 (以毫秒為單位)。出現此延遲的原因是 Aurora 叢集使用非同步複寫，如以 Amazon Aurora 進行複寫 所述。其顯示於此函數傳回結果中的 `replica_lag_in_msec` 欄中。由於與備用伺服器上的復原衝突而取消查詢時，也可能會發生延遲。您可檢查 `pg_stat_database_conflicts()`，以驗證此類衝突是否造成複本延遲。如需相關資訊，請參閱 PostgreSQL 文件中的統計數字收集器。如要進一步了解有關高可用性和複寫的資訊，請參 Amazon Aurora 常見問答集。

  Amazon CloudWatch 隨時間將 `replica_lag_in_msec` 結果儲存為 `AuroraReplicaLag` 指標。如需使用 Aurora 的 CloudWatch 指標相關資訊，請參閱 使用 Amazon CloudWatch 監控 Amazon Aurora 指標。

若要進一步了解 Aurora 僅供讀取複本和重新啟動的疑難排解相關資訊，請參閱 AWS 支援 中心中的為什麼我的 Amazon Aurora 僅供讀取複本落後且重新啟動？。

範例

下列範例顯示如何獲取 Aurora PostgreSQL 資料庫叢集中所有執行個體的複寫狀態：

```
=> SELECT *
FROM aurora_replica_status();
```

下列範例顯示 `docs-lab-apg-main` Aurora PostgreSQL 資料庫叢集中的寫入器執行個體：

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
```

```
        ELSE 'reader'
    END AS instance_role
FROM aurora_replica_status()
WHERE session_id = 'MASTER_SESSION_ID';
        server_id       | instance_role
------------------------+---------------
 db-119-001-instance-01 | writer
```

下列範例會列出叢集中的所有讀取器執行個體：

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role
FROM aurora_replica_status()
WHERE session_id <> 'MASTER_SESSION_ID';
        server_id       | instance_role
------------------------+---------------
db-119-001-instance-02  | reader
db-119-001-instance-03  | reader
db-119-001-instance-04  | reader
db-119-001-instance-05  | reader
(4 rows)
```

下列範例列出了所有執行個體、每個執行個體落後於寫入器的延遲距離，及自上次更新以來的時間：

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role,
    replica_lag_in_msec AS replica_lag_ms,
    round(extract (epoch FROM (SELECT age(clock_timestamp(), last_update_timestamp))) *
 1000) AS last_update_age_ms
FROM aurora_replica_status()
ORDER BY replica_lag_in_msec NULLS FIRST;
       server_id        | instance_role | replica_lag_ms | last_update_age_ms
------------------------+---------------+----------------+--------------------
 db-124-001-instance-03 | writer        |         [NULL] |               1756
 db-124-001-instance-01 | reader        |             13 |               1756
 db-124-001-instance-02 | reader        |             13 |               1756
```

```
(3 rows)
```

aurora_stat_activity

傳回每個伺服器程序的一列，顯示該程序目前活動相關的資訊。

語法

```
aurora_stat_activity();
```

引數

無

傳回類型

每個伺服器程序傳回一列。除了 pg_stat_activity 欄之外，還會新增下列欄位：

- planid：計劃識別符

使用須知

pg_stat_activity 用來傳回相同欄的補充檢視，其中包含附加 plan_id 欄，此欄會顯示目前的查詢執行計劃。

必須啟用 aurora_compute_plan_id，檢視才能傳回 plan_id。

您可從 Aurora PostgreSQL 版本 14.10、15.5 及所有其他更新的版本使用此函數。

範例

以下查詢範例會依 query_id 和 plan_id 彙總最高負載。

```
db1=# select count(*), query_id, plan_id
db1-# from aurora_stat_activity() where state = 'active'
db1-# and pid <> pg_backend_pid()
db1-# group by query_id, plan_id
db1-# order by 1 desc;

count |  query_id             |  plan_id
-------+-----------------------+-------------
 11    | -5471422286312252535 | -2054628807
```

```
 3     | -6907107586630739258 | -815866029
 1     | 5213711845501580017  |  300482084
(3 rows)
```

如果用於 query_id 的計劃有所變更，則 aurora_stat_activity 將報告新的 plan_id。

```
count  |  query_id            |  plan_id
-------+----------------------+-------------
 10    | -5471422286312252535 | 1602979607
 1     | -6907107586630739258 | -1809935983
 1     | -2446282393000597155 | -207532066
(3 rows)
```

aurora_stat_backend_waits

顯示特定後端程序的等待活動統計資料。

語法

```
aurora_stat_backend_waits(pid)
```

引數

pid – 後端程序的 ID。您可使用 pg_stat_activity 檢視取得程序 ID。

傳回類型

SETOF 記錄，包含下列欄：

- type_id – 表示等待事件類型的數字，例如，1 表示輕量型鎖定 (LWLock)，3 表示鎖定，或 6 表示用戶端工作階段。當您將此函數的結果與 aurora_stat_wait_type 函數的欄聯結時，這些值會變得有意義，如 範例 中所示。
- event_id – 等待事件的識別號碼。將此值與 aurora_stat_wait_event 中的欄聯結，以取得有意義的事件名稱。
- waits – 指定程序 ID 的累積等待數目。

- wait_time – 等待時間 (以毫秒為單位)。

使用須知

您可使用此函數分析自連線開啟後發生的特定後端 (工作階段) 等待事件。如要取得有關等待事件名稱和類型更有意義的資訊，您可以使用範例中所示的 JOIN，結合此函數 aurora_stat_wait_type 和 aurora_stat_wait_event。

範例

此範例會示範後端程序 ID 3027 的所有等待、類型和事件名稱。

```
=> SELECT type_name, event_name, waits, wait_time
        FROM aurora_stat_backend_waits(3027)
NATURAL JOIN aurora_stat_wait_type()
NATURAL JOIN aurora_stat_wait_event();
type_name |       event_name       | waits | wait_time
-----------+------------------------+-------+------------
 LWLock    | ProcArrayLock          |     3 |         27
 LWLock    | wal_insert             |   423 |      16336
 LWLock    | buffer_content         | 11840 |    1033634
 LWLock    | lock_manager           | 23821 |    5664506
 Lock      | tuple                  | 10258 |  152280165
 Lock      | transactionid          | 78340 | 1239808783
 Client    | ClientRead             | 34072 |   17616684
 IO        | ControlFileSyncUpdate  |     2 |          0
 IO        | ControlFileWriteUpdate |     4 |         32
 IO        | RelationMapRead        |     2 |        795
 IO        | WALWrite               | 36666 |      98623
 IO        | XactSync               |  4867 |    7331963
```

此範例會示範所有作用中工作階段 (pg_stat_activity state <> 'idle') 目前及累積等待類型和等待事件 (但並無叫用函數 (pid <> pg_backend_pid( )) 的目前工作階段。

```
=> SELECT a.pid,
            a.usename,
            a.app_name,
            a.current_wait_type,
            a.current_wait_event,
            a.current_state,
            wt.type_name AS wait_type,
            we.event_name AS wait_event,
```

```
            a.waits,
            a.wait_time
        FROM (SELECT pid,
                     usename,
                     left(application_name,16) AS app_name,
                     coalesce(wait_event_type,'CPU') AS current_wait_type,
                     coalesce(wait_event,'CPU') AS current_wait_event,
                     state AS current_state,
                     (aurora_stat_backend_waits(pid)).*
                FROM pg_stat_activity
               WHERE pid <> pg_backend_pid()
                 AND state <> 'idle') a
NATURAL JOIN aurora_stat_wait_type() wt
NATURAL JOIN aurora_stat_wait_event() we;
  pid  | usename  | app_name | current_wait_type | current_wait_event | current_state |
 wait_type |       wait_event       | waits | wait_time
-------+----------+----------+-------------------+--------------------+---------------
+-----------+------------------------+-------+-----------
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | wal_insert             |  1937 |     29975
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | buffer_content         | 22903 |    760498
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | lock_manager           | 10012 |    223207
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | tuple                  | 20315 |  63081529
 .
 .
 .
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 IO        | WALWrite               | 93293 |    237440
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 IO        | XactSync               | 13010 |  19525143
 30100 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | ProcArrayLock          |     6 |        53
 30100 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | wal_insert             |  1913 |     25450
 30100 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | buffer_content         | 22874 |    778005
 .
 .
 .
 30109 | postgres | pgbench  | IO                | XactSync           | active        |
 LWLock    | ProcArrayLock          |     3 |        71
```

```
 30109 | postgres | pgbench  | IO               | XactSync          | active        |
 LWLock    | wal_insert             |  1940 |     27741
 30109 | postgres | pgbench  | IO               | XactSync          | active        |
 LWLock    | buffer_content         | 22962 |    776352
 30109 | postgres | pgbench  | IO               | XactSync          | active        |
 LWLock    | lock_manager           |  9879 |    218826
 30109 | postgres | pgbench  | IO               | XactSync          | active        |
 Lock      | tuple                  | 20401 |  63581306
 30109 | postgres | pgbench  | IO               | XactSync          | active        |
 Lock      | transactionid          | 50769 | 211645008
 30109 | postgres | pgbench  | IO               | XactSync          | active        |
 Client    | ClientRead             | 89901 |  44192439
```

此範例會示範所有作用中工作階段 (pg_stat_activity state <> 'idle') 目前及前三 (3) 個累積等待類型和等待事件，目前工作階段 (pid <>pg_backend_pid( )) 除外。

```
=> SELECT top3.*
       FROM (SELECT a.pid,
                    a.usename,
                    a.app_name,
                    a.current_wait_type,
                    a.current_wait_event,
                    a.current_state,
                    wt.type_name AS wait_type,
                    we.event_name AS wait_event,
                    a.waits,
                    a.wait_time,
                    RANK() OVER (PARTITION BY pid ORDER BY a.wait_time DESC)
               FROM (SELECT pid,
                            usename,
                            left(application_name,16) AS app_name,
                            coalesce(wait_event_type,'CPU') AS current_wait_type,
                            coalesce(wait_event,'CPU') AS current_wait_event,
                            state AS current_state,
                            (aurora_stat_backend_waits(pid)).*
                       FROM pg_stat_activity
                      WHERE pid <> pg_backend_pid()
                        AND state <> 'idle') a
       NATURAL JOIN aurora_stat_wait_type() wt
       NATURAL JOIN aurora_stat_wait_event() we) top3
 WHERE RANK <=3;
  pid  | usename  | app_name | current_wait_type | current_wait_event | current_state |
 wait_type |   wait_event    |  waits  | wait_time  | rank
```

```
-------+----------+----------+------------------+--------------------+--------------
+-----------+-----------------+---------+------------+------
 20567 | postgres | psql     | CPU              | CPU                | active       |
 LWLock    | wal_insert      |   25000 |   67512003 |    1
 20567 | postgres | psql     | CPU              | CPU                | active       |
 IO        | WALWrite        | 3071758 |    1016961 |    2
 20567 | postgres | psql     | CPU              | CPU                | active       |
 IO        | BufFileWrite    |   20750 |     184559 |    3
 27743 | postgres | pgbench  | Lock             | transactionid      | active       |
 Lock      | transactionid   |  237350 | 1265580011 |    1
 27743 | postgres | pgbench  | Lock             | transactionid      | active       |
 Lock      | tuple           |   93641 |  341472318 |    2
 27743 | postgres | pgbench  | Lock             | transactionid      | active       |
 Client    | ClientRead      |  417556 |  204796837 |    3
 .
 .
 .
 27745 | postgres | pgbench  | IO               | XactSync           | active       |
 Lock      | transactionid   |  238068 | 1265816822 |    1
 27745 | postgres | pgbench  | IO               | XactSync           | active       |
 Lock      | tuple           |   93210 |  338312247 |    2
 27745 | postgres | pgbench  | IO               | XactSync           | active       |
 Client    | ClientRead      |  419157 |  207836533 |    3
 27746 | postgres | pgbench  | Lock             | transactionid      | active       |
 Lock      | transactionid   |  237621 | 1264528811 |    1
 27746 | postgres | pgbench  | Lock             | transactionid      | active       |
 Lock      | tuple           |   93563 |  339799310 |    2
 27746 | postgres | pgbench  | Lock             | transactionid      | active       |
 Client    | ClientRead      |  417304 |  208372727 |    3
```

aurora_stat_bgwriter

`aurora_stat_bgwriter` 是顯示有關 Optimized Reads 快取寫入資訊的統計資料檢視

語法

```
aurora_stat_bgwriter()
```

引數

無

傳回類型

SETOF 記錄包含所有 pg_stat_bgwriter 欄和以下附加欄。如需 pg_stat_bgwriter 欄的詳細資訊，請參閱 pg_stat_bgwriter。

您可以使用 pg_stat_reset_shared("bgwriter") 重設此函數的統計資料。

- orcache_blks_written - 寫入的最佳化讀取快取資料區塊總數。
- orcache_blk_write_time - 如果啟用 track_io_timing，它會追蹤寫入最佳化讀取快取資料檔案區塊所花費的總時間，以毫秒為單位。如需詳細資訊，請參閱 track_io_timing。

使用須知

此函數適用於下列 Aurora PostgreSQL 版本：

- 15.4 版和更新的 15 版本
- 14.9 版和更新的 14 版本

範例

```
=> select * from aurora_stat_bgwriter();
-[ RECORD 1 ]-----------------+-----------
orcache_blks_written          | 246522
orcache_blk_write_time        | 339276.404
```

aurora_stat_database

它攜帶 pg_stat_database 的所有欄，並在結尾加上新欄。

語法

```
aurora_stat_database()
```

引數

無

傳回類型

SETOF 記錄包含所有 pg_stat_database 欄和以下附加欄。如需 pg_stat_database 欄的詳細資訊，請參閱 pg_stat_database。

- storage_blks_read - 從此資料庫中 Aurora 儲存體讀取的共用區塊總數。
- orcache_blks_hit - 此資料庫中最佳化讀取快取命中的總數。
- local_blks_read - 在此資料庫中讀取的本機區塊總數。
- storage_blk_read_time - 如果啟用 track_io_timing，它會追蹤從 Aurora 儲存讀取資料檔案區塊所花費的總時間 (以毫秒為單位)，否則值為零。如需詳細資訊，請參閱 track_io_timing。
- local_blk_read_time - 如果啟用 track_io_timing，它會追蹤讀取本機資料檔案區塊所花費的總時間 (以毫秒為單位)，否則值為零。如需詳細資訊，請參閱 track_io_timing。
- orcache_blk_read_time - 如果啟用 track_io_timing，它會追蹤從最佳化讀取快取中讀取資料檔案區塊所花費的總時間 (以毫秒為單位)，否則值為零。如需詳細資訊，請參閱 track_io_timing。

> **Note**
>
> blks_read 的值是 storage_blks_read、orcache_blks_hit 和 local_blks_read 的總和。
> blk_read_time 的值是 storage_blk_read_time、orcache_blk_read_time 和 local_blk_read_time 的總和。

使用須知

此函數適用於下列 Aurora PostgreSQL 版本：

- 15.4 版和更新的 15 版本
- 14.9 版和更新的 14 版本

範例

以下範例顯示它如何攜帶所有 pg_stat_database 欄並在結尾附加 6 個新欄：

```
=> select * from aurora_stat_database() where datid=14717;
-[ RECORD 1 ]------------+------------------------------
```

```
datid                     | 14717
datname                   | postgres
numbackends               | 1
xact_commit               | 223
xact_rollback             | 4
blks_read                 | 1059
blks_hit                  | 11456
tup_returned              | 27746
tup_fetched               | 5220
tup_inserted              | 165
tup_updated               | 42
tup_deleted               | 91
conflicts                 | 0
temp_files                | 0
temp_bytes                | 0
deadlocks                 | 0
checksum_failures         |
checksum_last_failure     |
blk_read_time             | 3358.689
blk_write_time            | 0
session_time              | 1076007.997
active_time               | 3684.371
idle_in_transaction_time | 0
sessions                  | 10
sessions_abandoned        | 0
sessions_fatal            | 0
sessions_killed           | 0
stats_reset               | 2023-01-12 20:15:17.370601+00
orcache_blks_hit          | 425
orcache_blk_read_time     | 89.934
storage_blks_read         | 623
storage_blk_read_time     | 3254.914
local_blks_read           | 0
local_blk_read_time       | 0
```

aurora_stat_dml_activity

報告 Aurora PostgreSQL 叢集中資料庫上每種資料處理語言 (DML) 作業類型的累積活動。

語法

```
aurora_stat_dml_activity(database_oid)
```

引數

database_oid

Aurora PostgreSQL 叢集中資料庫的物件 ID (OID)。

傳回類型

SETOF 記錄

使用須知

`aurora_stat_dml_activity` 函數僅適用於與 PostgreSQL 引擎 11.6 及更新版本相容的 Aurora PostgreSQL 3.1 版。

在具有大量資料庫的 Aurora PostgreSQL 叢集上使用此函數，來識別哪些資料庫具有較多或較慢的 DML 活動，或兩者兼具。

`aurora_stat_dml_activity` 函數傳回的作業執行的次數，及 SELECT、INSERT、UPDATE 和 DELETE 作業的累積延遲 (以微秒為單位)。該報告僅包含成功的 DML 作業。

您可使用 PostgreSQL 統計數字存取函數 `pg_stat_reset` 來重置此統計數字。您可使用 `pg_stat_get_db_stat_reset_time` 函數，檢查上次重設此統計數字的時間。如需有關 PostgreSQL 統計數字存取函數的詳細資訊，請參閱 PostgreSQL 文件中的統計數字收集器。

範例

下列範例顯示如何報告已連線資料庫的 DML 活動統計數字。

```
--Define the oid variable from connected database by using \gset
=> SELECT oid,
          datname
     FROM pg_database
    WHERE datname=(select current_database()) \gset
=> SELECT *
     FROM aurora_stat_dml_activity(:oid);
select_count | select_latency_microsecs | insert_count | insert_latency_microsecs |
 update_count | update_latency_microsecs | delete_count | delete_latency_microsecs
--------------+--------------------------+--------------+--------------------------
+--------------+--------------------------+--------------+--------------------------
       178957 |                 66684115 |       171065 |                 28876649 |
    519538 |            1454579206167 |            1 |                    53027
```

```
-- Showing the same results with expanded display on
=> SELECT *
     FROM aurora_stat_dml_activity(:oid);
-[ RECORD 1 ]------------+-------------
select_count             | 178957
select_latency_microsecs | 66684115
insert_count             | 171065
insert_latency_microsecs | 28876649
update_count             | 519538
update_latency_microsecs | 1454579206167
delete_count             | 1
delete_latency_microsecs | 53027
```

下列範例顯示 Aurora PostgreSQL 叢集中所有資料庫的 DML 活動統計數字。此叢集有兩個資料庫：postgres 和 mydb。以逗號分隔的清單會與 select_count、select_latency_microsecs、insert_count、insert_latency_microsecs、up 和 delete_latency_microsecs 欄位相對應。

Aurora PostgreSQL 建立並使用一個名為 rdsadmin 的系統資料庫來支援管理作業，例如備份、還原、運作狀態檢查、複寫等等。這些 DML 作業對您的 Aurora PostgreSQL 叢集並無任何影響。

```
=> SELECT oid,
    datname,
    aurora_stat_dml_activity(oid)
    FROM pg_database;
oid  |    datname     |                   aurora_stat_dml_activity
-------+---------------
+-------------------------------------------------------------------
 14006 | template0      | (,,,,,,,)
 16384 | rdsadmin       | (2346623,1211703821,4297518,817184554,0,0,0,0)
     1 | template1      | (,,,,,,,)
 14007 | postgres       |
 (178961,66716329,171065,28876649,519538,1454579206167,1,53027)
 16401 | mydb           | (200246,64302436,200036,107101855,600000,83659417514,0,0)
```

下列範例顯示所有資料庫的 DML 活動統計數字以資料欄進行編排，提高可讀性。

```
SELECT db.datname,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 1), '()') AS select_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 2), '()') AS select_latency_microsecs,
```

```
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 3), '()') AS insert_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 4), '()') AS insert_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 5), '()') AS update_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 6), '()') AS update_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 7), '()') AS delete_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 8), '()') AS delete_latency_microsecs
FROM  (SELECT datname,
              aurora_stat_dml_activity(oid) AS asdmla
         FROM pg_database
      ) AS db;

    datname     | select_count | select_latency_microsecs | insert_count |
 insert_latency_microsecs | update_count | update_latency_microsecs | delete_count |
 delete_latency_microsecs
---------------+--------------+--------------------------+--------------
+--------------------------+--------------+--------------------------+--------------
+--------------------------
 template0      |              |                          |              |
             |              |                          |              |
 rdsadmin       | 4206523      | 2478812333               | 7009414      | 1338482258
             | 0            | 0                        | 0            | 0
 template1      |              |                          |              |
             |              |                          |              |
 fault_test     | 66           | 452099                   | 0            | 0
             | 0            | 0                        | 0            | 0
 db_access_test | 1            | 5982                     | 0            | 0
             | 0            | 0                        | 0            | 0
 postgres       | 42035        | 95179203                 | 5752         | 2678832898
             | 21157        | 441883182488             | 2            | 1520
 mydb           | 71           | 453514                   | 0            | 0
             | 1            | 190                      | 1            | 152
```

下列範例顯示了資料庫 (具 OID 16401) 之每個 DML 作業的平均累積延遲 (累積延遲除以計數)。

```
=> SELECT select_count,
          select_latency_microsecs,
          select_latency_microsecs/NULLIF(select_count,0) select_latency_per_exec,
          insert_count,
          insert_latency_microsecs,
          insert_latency_microsecs/NULLIF(insert_count,0) insert_latency_per_exec,
          update_count,
          update_latency_microsecs,
          update_latency_microsecs/NULLIF(update_count,0) update_latency_per_exec,
```

```
          delete_count,
          delete_latency_microsecs,
          delete_latency_microsecs/NULLIF(delete_count,0) delete_latency_per_exec
     FROM aurora_stat_dml_activity(16401);
-[ RECORD 1 ]------------+-------------
select_count             | 451312
select_latency_microsecs | 80205857
select_latency_per_exec  | 177
insert_count             | 451001
insert_latency_microsecs | 123667646
insert_latency_per_exec  | 274
update_count             | 1353067
update_latency_microsecs | 200900695615
update_latency_per_exec  | 148478
delete_count             | 12
delete_latency_microsecs | 448
delete_latency_per_exec  | 37
```

aurora_stat_get_db_commit_latency

取得 Aurora PostgreSQL 資料庫的累積遞交延遲 (以微秒為單位)。遞交延遲是指用戶端提交一個遞交請求和其收到遞交確認之間的時間。

語法

```
aurora_stat_get_db_commit_latency(database_oid)
```

引數

database_oid

Aurora PostgreSQL 資料庫的物件 ID (OID)。

傳回類型

SETOF 記錄

使用須知

Amazon CloudWatch 使用此函數來計算平均遞交延遲。如需有關 Amazon CloudWatch 指標及如何對高遞交延遲進行疑難排解的詳細資訊，請參閱 在 Amazon RDS 主控台中檢視指標 和 利用 Amazon CloudWatch 指標對 Amazon RDS 做出更好的決策。

您可使用 PostgreSQL 統計數字存取函數 pg_stat_reset 來重置此統計數字。您可使用 pg_stat_get_db_stat_reset_time 函數，檢查上次重設此統計數字的時間。如需有關 PostgreSQL 統計數字存取函數的詳細資訊，請參閱 PostgreSQL 文件中的統計數字收集器。

範例

下列範例會取得 pg_database 叢集中每個資料庫的累積遞交延遲。

```
=> SELECT oid,
    datname,
    aurora_stat_get_db_commit_latency(oid)
    FROM pg_database;

  oid  |    datname     | aurora_stat_get_db_commit_latency
-------+----------------+-----------------------------------
 14006 | template0      |                                 0
 16384 | rdsadmin       |                         654387789
     1 | template1      |                                 0
 16401 | mydb           |                            229556
 69768 | postgres       |                             22011
```

下列範例會取得目前已連線資料庫的累積遞交延遲。呼叫 aurora_stat_get_db_commit_latency 函數之前，該範例會先使用 \gset 來定義 oid 引數的變數，並從已連線的資料庫設定其值。

```
--Get the oid value from the connected database before calling
 aurora_stat_get_db_commit_latency
=> SELECT oid
     FROM pg_database
    WHERE datname=(SELECT current_database()) \gset
=> SELECT *
     FROM aurora_stat_get_db_commit_latency(:oid);

 aurora_stat_get_db_commit_latency
-----------------------------------
                        1424279160
```

下列範例會取得 pg_database 叢集中 mydb 資料庫的累積遞交延遲。接著，其會使用 pg_stat_reset 函數來重設此統計數字並顯示結果。最後，其可使用 pg_stat_get_db_stat_reset_time 函數來檢查上次重設此統計數字的時間。

```
=> SELECT oid,
```

```
    datname,
    aurora_stat_get_db_commit_latency(oid)
    FROM pg_database
    WHERE datname = 'mydb';

  oid  |  datname  | aurora_stat_get_db_commit_latency
-------+-----------+-----------------------------------
 16427 | mydb      |                           3320370


=> SELECT pg_stat_reset();
 pg_stat_reset
---------------


=> SELECT oid,
          datname,
          aurora_stat_get_db_commit_latency(oid)
     FROM pg_database
    WHERE datname = 'mydb';
  oid  |  datname  | aurora_stat_get_db_commit_latency
-------+-----------+-----------------------------------
 16427 | mydb      |                                 6


=> SELECT *
     FROM pg_stat_get_db_stat_reset_time(16427);

 pg_stat_get_db_stat_reset_time
--------------------------------
 2021-04-29 21:36:15.707399+00
```

aurora_stat_logical_wal_cache

顯示每個插槽的邏輯預寫日誌 (WAL) 快取使用量。

語法

```
SELECT * FROM aurora_stat_logical_wal_cache()
```

引數

無

傳回類型

SETOF 記錄，包含下列欄：

- name - 複寫槽的名稱。
- active_pid – walsender 程序的 ID。
- cache_hit – 自上次重設以來的 WAL 快存命中總數。
- cache_miss – 自上次重設以來的 WAL 快存未命中總數。
- blks_read – WAL 快取讀取請求的總數。
- hit_rate – WAL 快取命中率 (cache_hit / blks_read)。
- last_reset_timestamp – 上次重設計數器的時間。

使用須知

此函數適用於下列 Aurora PostgreSQL 版本。

- 15.2 版和所有更新版本
- 14.7 及更新版本
- 13.8 和更新版本
- 12.12 和更新版本
- 11.17 和更新版本

範例

下列範例顯示了兩個只有一個作用中 aurora_stat_logical_wal_cache 函數的複寫槽。

```
=> SELECT *
     FROM aurora_stat_logical_wal_cache();
    name    | active_pid | cache_hit | cache_miss | blks_read | hit_rate |
 last_reset_timestamp
------------+------------+-----------+------------+-----------+----------
+-------------------------------
 test_slot1 |      79183 |        24 |          0 |        24 | 100.00%  | 2022-08-05
 17:39:56.830635+00
 test_slot2 |            |         1 |          0 |         1 | 100.00%  | 2022-08-05
 17:34:04.036795+00
```

```
(2 rows)
```

aurora_stat_memctx_usage

報告每個 PostgreSQL 程序的記憶體內容使用情況。

語法

```
aurora_stat_memctx_usage()
```

引數

無

傳回類型

SETOF 記錄，包含下列欄：

- pid – 程序的 ID。
- name – 記憶體內容的名稱。
- allocated – 記憶體內容從基礎記憶體子系統取得的位元組數量。
- used – 提交至記憶體內容用戶端的位元組數量。
- instances – 此類型現有內容目前的計數。

使用須知

此函數顯示每個 PostgreSQL 程序的記憶體內容使用情況。有些處理程序會標示 anonymous。程序不會公開，因為它們包含受限制的關鍵字。

此函數從下列 Aurora PostgreSQL 版本開始適用：

- 15.3 版和更新的 15 版本
- 14.8 版和更新的 14 版本
- 13.11 版和更新的 13 版本
- 12.15 版和更新的 12 版
- 11.20 版和更新的 11 版本

範例

下列範例顯示呼叫 aurora_stat_memctx_usage 函數的結果。

```
=> SELECT *
     FROM aurora_stat_memctx_usage();

    pid| name                              | allocated |   used  | instances
-------+-----------------------------------+-----------+---------+-----------
123864 | Miscellaneous                     |     19520 |   15064 |         3
123864 | Aurora File Context               |      8192 |     616 |         1
123864 | Aurora WAL Context                |      8192 |     296 |         1
123864 | CacheMemoryContext                |    524288 |  422600 |         1
123864 | Catalog tuple context             |     16384 |   13736 |         1
123864 | ExecutorState                     |     32832 |   28304 |         1
123864 | ExprContext                       |      8192 |    1720 |         1
123864 | GWAL record construction          |      1024 |     832 |         1
123864 | MdSmgr                            |      8192 |     296 |         1
123864 | MessageContext                    |    532480 |  353832 |         1
123864 | PortalHeapMemory                  |      1024 |     488 |         1
123864 | PortalMemory                      |      8192 |     576 |         1
123864 | printtup                          |      8192 |     296 |         1
123864 | RelCache hash table entries       |      8192 |    8152 |         1
123864 | RowDescriptionContext             |      8192 |    1344 |         1
123864 | smgr relation context             |      8192 |     296 |         1
123864 | Table function arguments          |      8192 |     352 |         1
123864 | TopTransactionContext             |      8192 |     632 |         1
123864 | TransactionAbortContext           |     32768 |     296 |         1
123864 | WAL record construction           |     50216 |   43904 |         1
123864 | hash table                        |     65536 |   52744 |         6
123864 | Relation metadata                 |    191488 |  124240 |        87
104992 | Miscellaneous                     |      9280 |    7728 |         3
104992 | Aurora File Context               |      8192 |     376 |         1
104992 | Aurora WAL Context                |      8192 |     296 |         1
104992 ||Autovacuum Launcher               |      8192 |     296 |         1
104992 | Autovacuum database list          |     16384 |     744 |         2
104992 | CacheMemoryContext                |    262144 |  140288 |         1
104992 | Catalog tuple context             |      8192 |     296 |         1
104992 | GWAL record construction          |      1024 |     832 |         1
104992 | MdSmgr                            |      8192 |     296 |         1
104992 | PortalMemory                      |      8192 |     296 |         1
104992 | RelCache hash table entries       |      8192 |     296 |         1
104992 | smgr relation context             |      8192 |     296 |         1
104992 | Autovacuum start worker (tmp)     |      8192 |     296 |         1
```

```
104992 | TopTransactionContext           |     16384 |     592 |         2
104992 | TransactionAbortContext         |     32768 |     296 |         1
104992 | WAL record construction         |     50216 |   43904 |         1
104992 | hash table                      |     49152 |   34024 |         4
(39 rows)
```

將隱藏一些受限制的關鍵字，輸出結果如下所示：

```
postgres=>SELECT *
     FROM aurora_stat_memctx_usage();

    pid| name                            | allocated |   used   | instances
-------+---------------------------------+-----------+----------+-----------
  5482 | anonymous                       |      8192 |     456 |         1
  5482 | anonymous                       |      8192 |     296 |         1
```

aurora_stat_optimized_reads_cache

此函數顯示階層式快取統計資料。

語法

```
aurora_stat_optimized_reads_cache()
```

引數

無

傳回類型

SETOF 記錄，包含下列欄：

- `total_size` - 總最佳化讀取快取大小。
- `used_size` - 在最佳化讀取快取中使用的頁面大小。

使用須知

此函數適用於下列 Aurora PostgreSQL 版本：

- 15.4 版和更新的 15 版本

- 14.9 版和更新的 14 版本

範例

下列範例會在 r6gd.8xlarge 執行個體中顯示輸出：

```
=> select pg_size_pretty(total_size) as total_size, pg_size_pretty(used_size)
                as used_size from aurora_stat_optimized_reads_cache();
total_size | used_size
-----------+-----------
1054 GB    | 975 GB
```

aurora_stat_plans

傳回每個追蹤執行計劃的列。

語法

```
aurora_stat_plans(
    showtext
)
```

引數

- showtext：顯示查詢和計劃文字。有效值為 NULL、true 或 false。True 會顯示查詢和計劃文字。

傳回類型

傳回每個追蹤計劃的列，其中包含來自 `aurora_stat_statements` 的所有欄和下列額外欄。

- planid：計劃識別符
- explain_plan：解釋計劃文字
- plan_type：
  - `no plan`：未擷取任何計劃
  - `estimate`：以預估成本擷取的計劃
  - `actual`：使用 EXPLAIN ANALYZE 擷取的計劃
- plan_captured_time：計劃的前次擷取時間

使用須知

必須啟用 aurora_compute_plan_id，且 pg_stat_statements 必須位於 shared_preload_libraries，才能追蹤計劃。

可用的計劃數量由 pg_stat_statements.max 參數中設定的值所控制。啟用 aurora_compute_plan_id 時，您可以在 aurora_stat_plans 中追蹤高達此指定值的計劃。

您可從 Aurora PostgreSQL 版本 14.10、15.5 及所有其他更新的版本使用此函數。

範例

在下面的範例中，會擷取查詢識別符 -5471422286312252535 的兩個計劃，以及由 planid 追蹤陳述式統計資料。

```
db1=# select calls, total_exec_time, planid, plan_captured_time, explain_plan
db1-# from aurora_stat_plans(true)
db1-# where queryid = '-5471422286312252535'

calls    |  total_exec_time   |   planid    |      plan_captured_time       |
                explain_plan
---------+--------------------+-------------+-------------------------------
+-------------------------------------------------------------------
 1532632 |  3209846.097107853 |  1602979607 | 2023-10-31 03:27:16.925497+00 | Update on
 pgbench_branches                                      +
         |                    |             |                               |   ->
 Bitmap Heap Scan on pgbench_branches                     +
         |                    |             |                               |
 Recheck Cond: (bid = 76)                                   +
         |                    |             |                               |         -
>  Bitmap Index Scan on pgbench_branches_pkey          +
         |                    |             |                               |
     Index Cond: (bid = 76)
   61365 | 124078.18012200127 | -2054628807 | 2023-10-31 03:20:09.85429+00  | Update on
 pgbench_branches                                      +
         |                    |             |                               |   ->
 Index Scan using pgbench_branches_pkey on pgbench_branches+
         |                    |             |                               |
 Index Cond: (bid = 17)
```

aurora_stat_reset_wal_cache

Resets the counter for logical wal cache.

語法

重設特定槽

```
SELECT * FROM aurora_stat_reset_wal_cache('slot_name')
```

重設所有槽

```
SELECT * FROM aurora_stat_reset_wal_cache(NULL)
```

引數

NULL 或 slot_name

傳回類型

狀態訊息，文字字串

- 重設邏輯 WAL 快取計數器 – 成功訊息。此文字會在函數成功時傳回。
- 找不到複寫槽。請再試一次。- 失敗訊息。此文字會在函數未成功時傳回。

使用須知

此函數適用於下列版本：

- Aurora PostgreSQL 14.5 及更高版本
- Aurora PostgreSQL 13.8 版及更高版本
- Aurora PostgreSQL 12.12 版和更高版本
- Aurora PostgreSQL 11.17 版及更高版本

範例

下列範例會使用 aurora_stat_reset_wal_cache 函數來重設名為 test_results 的槽，然後嘗試重設不存在的槽。

```
=> SELECT *
```

```
     FROM aurora_stat_reset_wal_cache('test_slot');
aurora_stat_reset_wal_cache
------------------------------------
 Reset the logical wal cache counter.
(1 row)
=> SELECT *
     FROM aurora_stat_reset_wal_cache('slot-not-exist');
aurora_stat_reset_wal_cache
-----------------------------------------------
 Replication slot not found. Please try again.
(1 row)
```

aurora_stat_resource_usage

報告即時資源使用率，其中包含所有 Aurora PostgreSQL 後端程序的後端資源指標和 cpu 用量。

語法

```
aurora_stat_resource_usage()
```

引數

無

傳回類型

SETOF 記錄與欄：

- pid：程序識別符
- allocated_memory：依程序分配的記憶體總計 (以位元組為單位)
- used_memory：依程序實際使用的記憶體 (以位元組為單位)
- cpu_usage_percent：程序的 CPU 用量百分比

使用須知

此函數會顯示每個 Aurora PostgreSQL 後端程序的後端資源用量。

此函數從下列 Aurora PostgreSQL 版本開始適用：

- Aurora PostgreSQL 17.5 和更新的 17 版本
- Aurora PostgreSQL 16.9 和更新的 16 版本

- Aurora PostgreSQL 15.13 和更新的 15 版本
- Aurora PostgreSQL 14.18 和更新的 14 版本
- Aurora PostgreSQL 13.21 和更新的 13 版本

範例

下列範例示範 aurora_stat_resource_usage 函數的輸出。

```
=> select * from aurora_stat_resource_usage();
 pid  | allocated_memory | used_memory |   cpu_usage_percent
------+------------------+-------------+-----------------------
  666 |          1074032 |      333544 |   0.00729274882897963
  667 |           787312 |      287360 | 0.0029263928146372746
  668 |          3076776 |     1563488 |  0.006013116835953961
  684 |           803744 |      307480 |  0.002226855426881142
 2401 |          1232992 |      943144 |                     0
  647 |             8000 |         944 |   0.48853387812429855
  659 |           319344 |      243000 | 0.0004135602076683591
  663 |           262000 |      185736 |  0.008181301476644002
  664 |             9024 |        1216 |   0.10992313082653653
(9 rows)
```

aurora_stat_statements

顯示所有 pg_stat_statements 欄，並在結尾附加更多欄。

語法

```
aurora_stat_statements(showtext boolean)
```

引數

showtext 布林值

傳回類型

SETOF 記錄包含所有 pg_stat_statements 欄和以下附加欄。如需 pg_stat_statements 欄的詳細資訊，請參閱 pg_stat_statements。

您可以使用 pg_stat_statements_reset() 重設此函數的統計資料。

- `storage_blks_read` - 此陳述式從 Aurora 儲存中讀取的共用區塊總數。
- `orcache_blks_hit` - 此陳述式最佳化讀取快取命中的總數。
- `storage_blk_read_time`：如果啟用 `track_io_timing`，其會追蹤陳述式從 Aurora 儲存中讀取共用區塊所花費的時間總計 (以毫秒為單位)，否則值為零。如需詳細資訊，請參閱 track_io_timing。
- `local_blk_read_time`：如果啟用 `track_io_timing`，其會追蹤陳述式讀取本機區塊所花費的時間總計 (以毫秒為單位)，否則值為零。如需詳細資訊，請參閱 track_io_timing。
- `orcache_blk_read_time`：如果啟用 `track_io_timing`，其會追蹤陳述式從最佳化讀取快取中讀取共用區塊所花費的時間總計 (以毫秒為單位)，否則值為零。如需詳細資訊，請參閱 track_io_timing。
- `total_plan_peakmem`：此陳述式的所有呼叫在規劃階段期間尖峰記憶體值的總和。若要查看陳述式規劃期間的平均尖峰記憶體，請將此值除以呼叫次數。
- `min_plan_peakmem`：此陳述式的所有呼叫在規劃期間觀察到的最小尖峰記憶體值。
- `max_plan_peakmem`：此陳述式的所有呼叫在規劃期間的最大尖峰記憶體值。
- `total_exec_peakmem`：此陳述式的所有呼叫在執行階段期間的尖峰記憶體值總和。若要查看陳述式執行期間的平均尖峰記憶體，請將此值除以呼叫次數。
- `min_exec_peakmem`：此陳述式的所有呼叫在執行期間觀察的最小尖峰記憶體值 (以位元組為單位)。
- `max_exec_peakmem`：此陳述式的所有呼叫在執行期間觀察的最大尖峰記憶體值 (以位元組為單位)。

> **Note**
>
> `total_plan_peakmen`、`min_plan_peakmem` 和 `max_plan_peakmem` 只會在 `pg_stat_statements.track_planning` 設定開啟時受到監控。

使用須知

若要使用 aurora_stat_statements() 函數，您必須在 `shared_preload_libraries` 參數中包含 `pg_stat_statements` 延伸模組。

此函數適用於下列 Aurora PostgreSQL 版本：

- 15.4 版和更新的 15 版本

- 14.9 版和更新的 14 版本

顯示尖峰記憶體的欄可從下列版本取得：

- 16.3 及更新版本
- 15.7 及更新版本
- 14.12 及更新版本

範例

下列範例示範其如何攜帶所有 pg_stat_statements 欄，並在結尾附加 11 個新欄：

```
=> select * from aurora_stat_statements(true) where query like 'with window_max%';
-[ RECORD 1 ]----------
+----------------------------------------------------------------------------------------------------
userid                 | 16409
dbid                   | 5
toplevel               | t
queryid                | -8347523682669847482
query                  | with window_max as (select custid, max(scratch) over (order by
 scratch rows between $1 preceding
and $2 following) wmax from ts) select sum(wmax), max(custid) from window_max
plans                  | 0
total_plan_time        | 0
min_plan_time          | 0
max_plan_time          | 0
mean_plan_time         | 0
stddev_plan_time       | 0
calls                  | 4
total_exec_time        | 254.105121
min_exec_time          | 57.503164000000005
max_exec_time          | 68.687418
mean_exec_time         | 63.52628025
stddev_exec_time       | 5.150765359979643
rows                   | 4
shared_blks_hit        | 200192
shared_blks_read       | 0
shared_blks_dirtied    | 0
shared_blks_written    | 0
local_blks_hit         | 0
local_blks_read        | 0
```

```
local_blks_dirtied     | 0
local_blks_written     | 0
temp_blks_read         | 0
temp_blks_written      | 0
blk_read_time          | 0
blk_write_time         | 0
temp_blk_read_time     | 0
temp_blk_write_time    | 0
wal_records            | 0
wal_fpi                | 0
wal_bytes              | 0
jit_functions          | 0
jit_generation_time    | 0
jit_inlining_count     | 0
jit_inlining_time      | 0
jit_optimization_count | 0
jit_optimization_time  | 0
jit_emission_count     | 0
jit_emission_time      | 0
storage_blks_read      | 0
orcache_blks_hit       | 0
storage_blk_read_time  | 0
local_blk_read_time    | 0
orcache_blk_read_time  | 0
total_plan_peakmem     | 0
min_plan_peakmem       | 0
max_plan_peakmem       | 0
total_exec_peakmem     | 6356224
min_exec_peakmem       | 1589056
max_exec_peakmem       | 1589056
```

aurora_stat_system_waits

報告 Aurora PostgreSQL 資料庫執行個體的等待事件資訊。

語法

```
aurora_stat_system_waits()
```

引數

無

傳回類型

SETOF 記錄

使用須知

此函數會傳回您目前連線的資料庫執行個體所產生之每個等待事件的累積等待次數和累積等待時間。

傳回的記錄集包含下列欄位：

- type_id - 等待事件類型的 ID。
- event_id - 等待事件的 ID。
- waits - 等待事件發生的次數。
- wait_time - 等待此事件所花費的總時間 (以微秒為單位)。

此函數傳回的統計數字會在資料庫執行個體重新啟動時進行重設。

範例

下列範例會顯示呼叫 aurora_stat_system_waits 函數的結果。

```
=> SELECT *
    FROM aurora_stat_system_waits();
 type_id | event_id  |   waits   |  wait_time
---------+-----------+-----------+--------------
       1 |  16777219 |        11 |        12864
       1 |  16777220 |       501 |       174473
       1 |  16777270 |     53171 |     23641847
       1 |  16777271 |        23 |       319668
       1 |  16777274 |        60 |        12759
.
.
.
      10 | 167772231 |    204596 |    790945212
      10 | 167772232 |         2 |        47729
      10 | 167772234 |         1 |          888
      10 | 167772235 |         2 |           64
```

下列範例顯示如何將此函數與 aurora_stat_wait_event 和 aurora_stat_wait_type 一起使用，以產生更具可讀性的結果。

```
=> SELECT type_name,
             event_name,
             waits,
             wait_time
        FROM aurora_stat_system_waits()
NATURAL JOIN aurora_stat_wait_event()
NATURAL JOIN aurora_stat_wait_type();

 type_name |       event_name        |  waits   |  wait_time
-----------+-------------------------+----------+--------------
 LWLock    | XidGenLock              |       11 |        12864
 LWLock    | ProcArrayLock           |      501 |       174473
 LWLock    | buffer_content          |    53171 |     23641847
 LWLock    | rdsutils                |        2 |        12764
 Lock      | tuple                   |    75686 |   2033956052
 Lock      | transactionid           |  1765147 |  47267583409
 Activity  | AutoVacuumMain          |   136868 |  56305604538
 Activity  | BgWriterHibernate       |     7486 |  55266949471
 Activity  | BgWriterMain            |     7487 |   1508909964
.
.
.
 IO        | SLRURead                |        3 |        11756
 IO        | WALWrite                | 52544463 |    388850428
 IO        | XactSync                |   187073 |    597041642
 IO        | ClogRead                |        2 |        47729
 IO        | OutboundCtrlRead        |        1 |          888
 IO        | OutboundCtrlWrite       |        2 |           64
```

aurora_stat_wait_event

列出 Aurora PostgreSQL 的所有支援等待事件。如需 Aurora PostgreSQL 等待事件的相關資訊，請參閱 Amazon Aurora PostgreSQL 等待事件。

語法

```
aurora_stat_wait_event()
```

引數

無

傳回類型

SETOF 記錄，包含下列欄：

- type_id - 等待事件類型的 ID。
- event_id - 等待事件的 ID。
- type_name – 等待類型名稱
- event_name – 等待事件名稱

使用須知

如要查看具事件類型而非 ID 的事件名稱，請將此函數與其他函數 (例如 aurora_stat_wait_type 和 aurora_stat_system_waits) 一起搭配使用。此函數所傳回的等待事件名稱與 aurora_wait_report 函數傳回的等待事件名稱相同。

範例

下列範例會顯示呼叫 aurora_stat_wait_event 函數的結果。

```
=>  SELECT *
    FROM aurora_stat_wait_event();

 type_id | event_id  |               event_name
---------+-----------+-----------------------------------------
       1 |  16777216 | <unassigned:0>
       1 |  16777217 | ShmemIndexLock
       1 |  16777218 | OidGenLock
       1 |  16777219 | XidGenLock
.
.
.
       9 | 150994945 | PgSleep
       9 | 150994946 | RecoveryApplyDelay
      10 | 167772160 | BufFileRead
      10 | 167772161 | BufFileWrite
      10 | 167772162 | ControlFileRead
.
.
.
      10 | 167772226 | WALInitWrite
      10 | 167772227 | WALRead
```

```
      10 | 167772228 | WALSync
      10 | 167772229 | WALSyncMethodAssign
      10 | 167772230 | WALWrite
      10 | 167772231 | XactSync
.
.
.
      11 | 184549377 | LsnAllocate
```

下列範例連線 aurora_stat_wait_type 和 aurora_stat_wait_event 以傳回類型名稱和事件名稱，可提高可讀性。

```
=> SELECT *
    FROM aurora_stat_wait_type() t
    JOIN aurora_stat_wait_event() e
      ON t.type_id = e.type_id;

 type_id | type_name | type_id | event_id  |               event_name
---------+-----------+---------+-----------+-----------------------------------------
       1 | LWLock    |       1 |  16777216 | <unassigned:0>
       1 | LWLock    |       1 |  16777217 | ShmemIndexLock
       1 | LWLock    |       1 |  16777218 | OidGenLock
       1 | LWLock    |       1 |  16777219 | XidGenLock
       1 | LWLock    |       1 |  16777220 | ProcArrayLock
.
.
.
       3 | Lock      |       3 |  50331648 | relation
       3 | Lock      |       3 |  50331649 | extend
       3 | Lock      |       3 |  50331650 | page
       3 | Lock      |       3 |  50331651 | tuple
.
.
.
      10 | IO        |      10 | 167772214 | TimelineHistorySync
      10 | IO        |      10 | 167772215 | TimelineHistoryWrite
      10 | IO        |      10 | 167772216 | TwophaseFileRead
      10 | IO        |      10 | 167772217 | TwophaseFileSync
.
.
.
      11 | LSN       |      11 | 184549376 | LsnDurable
```

aurora_stat_wait_type

列出 Aurora PostgreSQL 的所有支援等待類型。

語法

```
aurora_stat_wait_type()
```

引數

無

傳回類型

SETOF 記錄，包含下列欄：

- type_id - 等待事件類型的 ID。
- type_name – 等待類型名稱。

使用須知

如要查看具等待事件類型而非 ID 的等待事件名稱，請將此函數與其他函數 (例如 aurora_stat_wait_event 和 aurora_stat_system_waits) 一起搭配使用。此函數所傳回的等待類型名稱與 aurora_wait_report 函數傳回的等待類型名稱相同。

範例

下列範例會顯示呼叫 aurora_stat_wait_type 函數的結果。

```
=> SELECT *
     FROM aurora_stat_wait_type();
 type_id | type_name
---------+-----------
       1 | LWLock
       3 | Lock
       4 | BufferPin
       5 | Activity
       6 | Client
       7 | Extension
       8 | IPC
       9 | Timeout
```

```
      10 | IO
      11 | LSN
```

aurora_version

傳回 Amazon Aurora PostgreSQL 相容版本號的字串值。

語法

```
aurora_version()
```

引數

無

傳回類型

CHAR 或 VARCHAR 字串

使用須知

此函數顯示 Amazon Aurora PostgreSQL 相容版本資料庫引擎的版本。版本號會以字串傳回，格式為*#.##.####*。如需 Aurora PostgreSQL 版本號的詳細資訊，請參閱 Aurora 版本編號。

您可設定 Aurora PostgreSQL 資料庫叢集的維護時段，來選擇套用次要版本升級的時間。如要瞭解如何作業，請參閱維持 為 Amazon Aurora 資料庫叢集。

從發行 PostgreSQL 版本 13.3、12.8、11.13、10.18 及所有其他更新的版本開始，Aurora 版本編號會接著 PostgreSQL 版本編號。如需有關所有 Aurora PostgreSQL 版本的詳細資訊，請參閱 Aurora PostgreSQL 版本備註中的 Amazon Aurora PostgreSQL 更新。

範例

下列範例顯示在執行 PostgreSQL 12.7、Aurora PostgreSQL 版本 4.2 的 Aurora PostgreSQL 資料庫叢集上呼叫 `aurora_version` 函數，然後在執行 Aurora PostgreSQL 版本 13.3 的叢集上執行相同函數的結果。

```
=> SELECT * FROM aurora_version();
aurora_version
----------------
 4.2.2
```

```
SELECT * FROM aurora_version();
aurora_version
----------------
 13.3.0
```

此範例說明如何將函數與各種選項結合使用，以取得有關 Aurora PostgreSQL 版本的更多詳細資料。此範例具有與 PostgreSQL 版本編號不同的 Aurora 版本編號。

```
=> SHOW SERVER_VERSION;
 server_version
----------------
 12.7
(1 row)

=> SELECT * FROM aurora_version();
aurora_version
----------------
 4.2.2
(1 row)

=> SELECT current_setting('server_version') AS "PostgreSQL Compatiblility";
PostgreSQL Compatiblility
--------------------------
 12.7
(1 row)

=> SELECT version() AS "PostgreSQL Compatiblility Full String";
PostgreSQL Compatiblility Full String
-------------------------------------------------------------------------------------------------
 PostgreSQL 12.7 on aarch64-unknown-linux-gnu, compiled by aarch64-unknown-linux-gnu-
gcc (GCC) 7.4.0, 64-bit
(1 row)

=> SELECT 'Aurora: '
    || aurora_version()
    || ' Compatible with PostgreSQL: '
    || current_setting('server_version') AS "Instance Version";
Instance Version
-----------------------------------------------
 Aurora: 4.2.2 Compatible with PostgreSQL: 12.7
(1 row)
```

下一個範例會使用前面範例中具相同選項的函數。此範例並無與 PostgreSQL 版本編號不同的 Aurora 版本編號。

```
=> SHOW SERVER_VERSION;
server_version
----------------
 13.3

=> SELECT * FROM aurora_version();
aurora_version
----------------
 13.3.0
=> SELECT current_setting('server_version') AS "PostgreSQL Compatiblility";
PostgreSQL Compatiblility
--------------------------
 13.3

=> SELECT version() AS "PostgreSQL Compatiblility Full String";
PostgreSQL Compatiblility Full String
-------------------------------------------------------------------------------------------------------
 PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC)
 7.4.0, 64-bit
=> SELECT 'Aurora: '
    || aurora_version()
    || ' Compatible with PostgreSQL: '
    || current_setting('server_version') AS "Instance Version";
Instance Version
-----------------------------------------------------
 Aurora: 13.3.0 Compatible with PostgreSQL: 13.3
```

aurora_volume_logical_start_lsn

傳回記錄序號 (LSN)，用於識別 Aurora 叢集磁碟區邏輯預寫日誌 (WAL) 串流中的記錄開頭。

語法

```
aurora_volume_logical_start_lsn()
```

引數

無

傳回類型

pg_lsn

使用須知

此函數可識別指定 Aurora 叢集磁碟區邏輯 WAL 串流中的記錄開頭。在使用邏輯複寫和 Aurora 快速複製執行主要版本升級時，您可以用此功能判斷擷取快照或資料庫複製的 LSN。然後，對於在 LSN 之後記錄的較新資料，您可以使用邏輯複寫持續進行串流，並將發布者的變更同步至訂閱用戶。

如需使用邏輯複寫進行主要版本升級的詳細資訊，請參閱 使用邏輯複寫來執行 Aurora PostgreSQL 的主要版本升級。

此函數適用於下列版本的 Aurora PostgreSQL：

- 15.2 版和更新的 15 版本
- 14.3 版和更新的 14 版
- 13.6 版及更新的第 13 版本
- 12.10 版和更新的第 12 版本
- 11.15 和更新的第 11 版本
- 10.20 和更新的第 10 版本

範例

您可以使用下列查詢取得日誌序號 (LSN)：

```
postgres=> SELECT aurora_volume_logical_start_lsn();

aurora_volume_logical_start_lsn
---------------
0/402E2F0
(1 row)
```

aurora_wait_report

此函數顯示一段時間內的等待事件活動。

語法

```
aurora_wait_report([time])
```

引數

時間 (選用)

時間 (以秒為單位)。預設為 10 秒。

傳回類型

SETOF 記錄，包含下列欄：

- type_name – 等待類型名稱
- event_name – 等待事件名稱
- wait – 等待次數
- wait_time – 等待時間 (以毫秒為單位)
- ms_per_wait – 等待次數的平均毫秒數
- waits_per_xact – 按一筆交易次數計算的平均等待時間
- ms_per_xact – 交易次數的平均毫秒數

使用須知

此函數從 Aurora PostgreSQL 1.1 版開始使用，並與 PostgreSQL 9.6.6 和更高版本相容。

如要使用此函數，你需要先建立 Aurora PostgreSQL `aurora_stat_utils` 擴充功能，如下所示：

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

如需可用 Aurora PostgreSQL 擴充功能版本的詳細資訊，請參閱 Aurora PostgreSQL 版本備註中的 Amazon Aurora PostgreSQL 的擴充功能版本。

此函數透過比較來自 aurora_stat_system_waits() 函數和 pg_stat_database PostgreSQL 統計數字檢視中的兩個統計資料快照來計算執行個體層級的等待事件。

如需更多 `aurora_stat_system_waits()` 和 `pg_stat_database` 的相關資訊，請參閱 PostgreSQL 文件中的統計數字收集器。

執行時，此函數會拍攝初始快照，等待指定的秒數，然後拍攝第二個快照。函數比較兩個快照並傳回差異。此差異表示執行個體在該時間間隔內的活動。

在寫入器執行個體上，函數還會顯示已提交的交易和 TPS (每秒交易數) 的數量。此函數傳回執行個體層級的資訊，並包括執行個體上的所有資料庫。

範例

此範例說明如何建立 aurora_stat_utils 延伸模組，以使用 aurora_wait_report 函數。

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

此範例說明如何檢查等待報告 10 秒。

```
=> SELECT *
     FROM aurora_wait_report();
NOTICE:  committed 34 transactions in 10 seconds (tps 3)
 type_name |    event_name     | waits | wait_time | ms_per_wait | waits_per_xact |
 ms_per_xact
-----------+-------------------+-------+-----------+-------------+----------------
+-------------
 Client    | ClientRead        |    26 |  30003.00 |    1153.961 |           0.76 |
 882.441
 Activity  | WalWriterMain     |    50 |  10051.32 |     201.026 |           1.47 |
 295.627
 Timeout   | PgSleep           |     1 |  10049.52 |   10049.516 |           0.03 |
 295.574
 Activity  | BgWriterHibernate |     1 |  10048.15 |   10048.153 |           0.03 |
 295.534
 Activity  | AutoVacuumMain    |    18 |   9941.66 |     552.314 |           0.53 |
 292.402
 Activity  | BgWriterMain      |     1 |    201.09 |     201.085 |           0.03 |
   5.914
 IO        | XactSync          |    15 |     25.34 |       1.690 |           0.44 |
   0.745
 IO        | RelationMapRead   |    12 |      0.54 |       0.045 |           0.35 |
   0.016
 IO        | WALWrite          |    84 |      0.21 |       0.002 |           2.47 |
   0.006
 IO        | DataFileExtend    |     1 |      0.02 |       0.018 |           0.03 |
   0.001
```

此範例說明如何檢查等待報告 60 秒。

```
=> SELECT *
     FROM aurora_wait_report(60);
NOTICE:  committed 1544 transactions in 60 seconds (tps 25)
 type_name |       event_name       |  waits  | wait_time | ms_per_wait |
 waits_per_xact | ms_per_xact
-----------+------------------------+---------+-----------+-------------
+----------------+-------------
 Lock      | transactionid          |    6422 | 477000.53 |      74.276 |
 4.16 |     308.938
 Client    | ClientRead             |    8265 | 270752.99 |      32.759 |
 5.35 |     175.358
 Activity  | CheckpointerMain       |       1 |  60100.25 |   60100.246 |
 0.00 |      38.925
 Timeout   | PgSleep                |       1 |  60098.49 |   60098.493 |
 0.00 |      38.924
 Activity  | WalWriterMain          |     296 |  60010.99 |     202.740 |
 0.19 |      38.867
 Activity  | AutoVacuumMain         |     107 |  59827.84 |     559.139 |
 0.07 |      38.749
 Activity  | BgWriterMain           |     290 |  58821.83 |     202.834 |
 0.19 |      38.097
 IO        | XactSync               |    1295 |  55220.13 |      42.641 |
 0.84 |      35.764
 IO        | WALWrite               | 6602259 |  47810.94 |       0.007 |
 4276.07 |      30.966
 Lock      | tuple                  |     473 |  29880.67 |      63.173 |
 0.31 |      19.353
 LWLock    | buffer_mapping         |     142 |   3540.13 |      24.930 |
 0.09 |       2.293
 Activity  | BgWriterHibernate      |     290 |   1124.15 |       3.876 |
 0.19 |       0.728
 IO        | BufFileRead            |    7615 |    618.45 |       0.081 |
 4.93 |       0.401
 LWLock    | buffer_content         |      73 |    345.93 |       4.739 |
 0.05 |       0.224
 LWLock    | lock_manager           |      62 |    191.44 |       3.088 |
 0.04 |       0.124
 IO        | RelationMapRead        |      72 |      5.16 |       0.072 |
 0.05 |       0.003
 LWLock    | ProcArrayLock          |       1 |      2.01 |       2.008 |
 0.00 |       0.001
```

```
IO        | ControlFileWriteUpdate |       2 |      0.03 |       0.013 |
0.00 |       0.000
IO        | DataFileExtend         |       1 |      0.02 |       0.018 |
0.00 |       0.000
IO        | ControlFileSyncUpdate  |       1 |      0.00 |       0.000 |
0.00 |       0.000
```

Amazon Aurora PostgreSQL 參數

您可以採用管理 Amazon RDS 資料庫執行個體的相同方式來管理您的 Amazon Aurora 資料庫叢集，方法是在資料庫參數群組中使用參數。但是 Amazon Aurora 與 Amazon RDS 的不同之處在於，Aurora 資料庫叢集具有多個資料庫執行個體。您用來管理 Amazon Aurora 資料庫叢集的部分參數適用於整個叢集，而其他參數僅適用資料庫叢集中的指定資料庫執行個體，如下所示：

- 資料庫叢集參數群組 – 資料庫叢集參數群組包含套用至整個 Aurora 資料庫叢集的引擎組態參數集。例如，叢集快取管理是 Aurora 資料庫叢集的一項功能，由屬於資料庫叢集參數群組一部分的 `apg_ccm_enabled` 參數控制。資料庫叢集參數群組還包含組成叢集的資料庫執行個體之資料庫參數群組預設設定。
- 資料庫參數群組 – 資料庫參數群組是一組引擎組態值，會套用至該引擎類型的特定資料庫執行個體。PostgreSQL 資料庫引擎的資料庫參數群組是由 RDS for PostgreSQL 資料庫執行個體 和 Aurora PostgreSQL 資料庫叢集使用。這些群態設定適用於因 Aurora 叢集內資料庫執行個體而有所不同的屬性，如記憶體緩衝的尺寸。

您可以管理資料庫叢集參數群組中的叢集層級參數。也可以管理資料庫參數群組中的執行個體層級參數。您可以使用 Amazon RDS 主控台、 AWS CLI或 Amazon RDS API 來管理參數。管理叢集層級參數和執行個體層級參數有不同的命令。

- 若要管理資料庫叢集參數群組中的叢集層級參數，請使用 modify-db-cluster-parameter-group AWS CLI 命令。
- 若要管理資料庫叢集中資料庫執行個體的資料庫參數群組中的執行個體層級參數，請使用 modify-db-parameter-group AWS CLI 命令。

若要進一步了解 AWS CLI，請參閱AWS Command Line Interface 《 使用者指南》中的使用 AWS CLI 。

如需參數群組的詳細資訊，請參閱Amazon Aurora 的參數群組。

查看 Aurora PostgreSQL 資料庫叢集和資料庫參數

在 AWS 管理主控台中，可以查看 RDS for PostgreSQL 資料庫執行個體及 Aurora PostgreSQL 資料庫叢集的所有可用預設參數群組。會針對每個 AWS 區域列出所有資料庫引擎和資料庫叢集類型和版本的預設參數群組。還會列出所有自訂參數群組。

您也可以使用 AWS CLI 或 Amazon RDS API，列出資料庫叢集參數群組和資料庫參數群組中包含的參數 AWS 管理主控台，而不是在 中檢視。例如，若要列出資料庫叢集參數群組中的參數，您可以使用 describe-db-cluster-parameters AWS CLI 命令，如下所示：

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12
```

該命令會傳回每個參數的詳細 JSON 描述。要減少傳回的資訊量，您可以使用 `--query` 選項指定需要的資訊。例如，您可以取得預設 Aurora PostgreSQL 12 資料庫叢集參數群組的參數名稱、描述和允許值，如下所示：

對於 Linux、macOS 或 Unix：

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12 \
    --query 'Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

在 Windows 中：

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12 ^
    --query "Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Aurora 資料庫叢集參數群組包括資料庫執行個體參數群組和指定 Aurora 資料庫引擎的預設值。您可以透過使用 describe-db-parameters AWS CLI 命令，從相同的預設 Aurora PostgreSQL 預設參數群組取得資料庫參數清單，如下所示。

針對 Linux、macOS 或 Unix：

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 \
    --query 'Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

在 Windows 中：

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 ^
    --query "Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

上述命令會傳回資料庫叢集或資料庫參數群組的參數清單，其中包括查詢中指定的描述和其他詳細資訊。以下是回應範例：

```
[
    [
        {
            "ParameterName": "apg_enable_batch_mode_function_execution",
            "ApplyType": "dynamic",
            "Description": "Enables batch-mode functions to process sets of rows at a
 time.",
            "AllowedValues": "0,1"
        }
    ],
    [
        {
            "ParameterName": "apg_enable_correlated_any_transform",
            "ApplyType": "dynamic",
            "Description": "Enables the planner to transform correlated ANY Sublink
 (IN/NOT IN subquery) to JOIN when possible.",
            "AllowedValues": "0,1"
        }
    ],...
```

下表中包含 Aurora PostgreSQL 14 版的預設資料庫叢集參數值和資料庫參數值。

Aurora PostgreSQL 叢集層級參數

您可以使用 AWS 管理主控台、 AWS CLI 或 Amazon RDS API，檢視特定 Aurora PostgreSQL 版本可用的叢集層級參數。如需在 RDS 主控台中檢視 Aurora PostgreSQL 資料庫叢集參數群組中參數的相關資訊，請參閱 在 Amazon Aurora 中檢視資料庫叢集參數群組的參數值。

部分叢集層級參數並不適用於所有版本，而部分正被棄用。如需檢視特定 Aurora PostgreSQL 版本之參數的相關資訊，請參閱 查看 Aurora PostgreSQL 資料庫叢集和資料庫參數。

例如，下表列出 Aurora PostgreSQL 第 14 版預設資料庫叢集參數群組中可用的參數。如果建立 Aurora PostgreSQL 資料庫叢集時沒有指定自己的自訂資料庫參數群組，系統將使用所選版本的預設 Aurora 資料庫叢集參數群組來建立資料庫叢集，例如`default.aurora-postgresql14`、`default.aurora-postgresql13` 等。

如需相同預設資料庫叢集參數群組的資料庫執行個體參數清單，請參閱 Aurora PostgreSQL 執行個體層級參數。

參數名稱	描述	預設
ansi_constraint_trigger_ordering	將限制觸發程序的觸發順序更改為與 ANSI SQL 標準相容。	–
ansi_force_foreign_key_checks	確保無論動作中存在的各種觸發程序內容如何，參考動作 (如串聯刪除或串聯更新) 一律會執行。	–
ansi_qualified_update_set_target	支援 UPDATE 中的資料表和結構描述限定詞... SET 陳述式。	–
apg_ccm_enabled	啟用或停用叢集的叢集快取管理。	–
apg_enable_batch_mode_function_execution	啟用批次模式函數，以便一次處理一組資料列。	–
apg_enable_correlated_any_transform	允許規劃器盡可能將相關的任何子連結 (IN/NOT IN 子查詢) 轉換為 JOIN。	–
apg_enable_function_migration	允許規劃器將符合條件的純量函數遷移到 FROM 子句。	–

參數名稱	描述	預設
apg_enable_not_in_transform	允許規劃器盡可能將 NOT IN 子查詢轉換為 ANTI JOIN。	–
apg_enable_remove_redundant_inner_joins	允許規劃器移除冗餘的內部聯結。	–
apg_enable_semijoin_push_down	允許使用半聯結篩選器進行雜湊聯結。	–
apg_plan_mgmt.capture_plan_baselines	擷取計畫基準模式。手動 - 啟用任何 SQL 陳述式的計畫擷取，關閉 - 停用計畫擷取，自動 - 為 pg_stat_陳述式中滿足合格條件的陳述式啟用計畫擷取。	off
apg_plan_mgmt.max_databases	設定可使用 apg_plan_mgmt 管理查詢的資料庫數上限。	10
apg_plan_mgmt.max_plans	設定 apg_plan_mgmt 可快取的計畫數上限。	10000
apg_plan_mgmt.plan_retention_period	上次使用計畫後經過幾天自動刪除計畫的天數上限。	32
apg_plan_mgmt.unapproved_plan_execution_threshold	估計總計畫成本，低於此成本將執行未核准的計畫。	0
apg_plan_mgmt.use_plan_baselines	受管陳述式僅使用已核准或固定計畫。	false
application_name	設定要在統計資訊和日誌中報告的應用程式名稱。	–
array_nulls	允許在陣列中輸入 NULL 元素。	–

參數名稱	描述	預設
aurora_compute_plan_id	監控查詢執行計劃，以偵測造成目前資料庫負載的執行計劃，以及追蹤執行計劃隨時間的效能統計資料。如需更多詳細資訊，請參閱監控 Aurora PostgreSQL 的查詢執行計劃。	on
authentication_timeout	(秒) 設定完成用戶端身分驗證的允許時間上限。	–
auto_explain.log_analyze	將 EXPLAIN ANALYZE 用於計畫日誌記錄。	–
auto_explain.log_buffers	日誌緩衝區使用情況。	–
auto_explain.log_format	用於計畫日誌記錄的 EXPLAIN 格式。	–
auto_explain.log_min_duration	設定執行時間下限，超出此時間就會記錄計畫。	–
auto_explain.log_nested_statements	記錄巢狀陳述式。	–
auto_explain.log_timing	收集計時資料而不僅是資料列數。	–
auto_explain.log_triggers	在計畫中包含觸發程序統計數字。	–
auto_explain.log_verbose	將 EXPLAIN VERBOSE 用於計畫日誌記錄。	–
auto_explain.sample_rate	待處理查詢的一部分。	–
autovacuum	啟動自動資料清理子程序。	–

參數名稱	描述	預設
autovacuum_analyze_scale_factor	分析之前插入、更新或刪除的元組數 (成為 reltuple 的一部分)。	0.05
autovacuum_analyze_threshold	分析之前插入、更新或刪除的元組數下限。	–
autovacuum_freeze_max_age	要自動資料清理資料表以防止交易 ID 包圍的存留期。	–
autovacuum_max_workers	設定同時執行自動清空工作者程序的數目上限。	GREATEST(DBInstanceClassMemory/64371566592,3)
autovacuum_multixact_freeze_max_age	自動清空資料表以防止 multixact 迴繞的 multixact 期限。	–
autovacuum_naptime	(秒) 自動清空執行之間的休眠時間。	5
autovacuum_vacuum_cost_delay	(毫秒) 自動清空的清空成本延遲 (以毫秒為單位)。	5
autovacuum_vacuum_cost_limit	自動資料清理在小憩前可用的清理成本金額。	GREATEST(log(DBInstanceClassMemory/21474836480)*600,200)
autovacuum_vacuum_insert_scale_factor	分析之前插入的元組數 (成為 reltuple 的一部分)。	–
autovacuum_vacuum_insert_threshold	清空之前插入的元組數下限，或用 -1 停用插入清空。	–
autovacuum_vacuum_scale_factor	清空之前更新或刪除的元組數 (成為 reltuple 的一小部分)。	0.1
autovacuum_vacuum_threshold	清空之前更新或刪除的元組數下限。	–

參數名稱	描述	預設
autovacuum_work_mem	(kB) 設定每個自動清空工作者程序使用的記憶體上限。	GREATEST(DBInstanceClassMemory/32768,131072)
babelfishpg_tds.default_server_name	預設 Babelfish 伺服器名稱	Microsoft SQL Server
babelfishpg_tds.listen_addresses	設定用來接聽 TDS 的主機名稱或 IP 地址。	*
babelfishpg_tds.port	設定伺服器在哪個 TDS TCP 連接埠上接聽。	1433
babelfishpg_tds.tds_debug_log_level	將 TDS 中的日誌記錄層級設為 0 會停用日誌記錄	1
babelfishpg_tds.tds_default_numeric_precision	針對要在 TDS 資料欄中繼資料中傳送的數值類型，設定預設精確度 (如果引擎未指定)。	38
babelfishpg_tds.tds_default_numeric_scale	針對要在 TDS 資料欄中繼資料中傳送的數值類型，設定預設小數位數 (如果引擎未指定)。	8
babelfishpg_tds.tds_default_packet_size	設定用於待連線 SQL Server 用戶端的預設封包大小。	4096
babelfishpg_tds.tds_default_protocol_version	設定用於所有待連線用戶端的預設 TDS 通訊協定版本。	DEFAULT
babelfishpg_tds.tds_ssl_encrypt	設定 SSL 加密選項	0
babelfishpg_tds.tds_ssl_max_protocol_version	設定 TDS 工作階段使用的最高 SSL/TLS 通訊協定版本。	TLSv1.2

參數名稱	描述	預設
babelfishpg_tds.tds_ssl_min_protocol_version	設定 TDS 工作階段使用的最低 SSL/TLS 通訊協定版本。	來自 Aurora PostgreSQL 第 16 版的 TLSv1.2，TLSv1 適用於早於 Aurora PostgreSQL 第 16 版的版本
babelfishpg_tsql.default_locale	CREATE COLLATION 建立的定序要使用的預設地區設定。	zh-TW
babelfishpg_tsql.migration_mode	定義是否支援多個使用者資料庫	來自 Aurora PostgreSQL 第 16 版的 multi-db，single-db 適用於早於 Aurora PostgreSQL 第 16 版的版本
babelfishpg_tsql.server_collation_name	預設伺服器定序的名稱	sql_latin1_general_cp1_ci_as
babelfishpg_tsql.version	設定 @@VERSION 變數的輸出	預設
backend_flush_after	(8Kb) 幾個分頁後將先前執行的寫入排清到磁碟。	–
backslash_quote	設定字串常值中是否允使用 \\。	–
backtrace_functions	記錄這些函數中錯誤的回溯追蹤。	–
bytea_output	設定 bytea 的輸出格式。	–
check_function_bodies	在 CREATE FUNCTION 執行期間檢查函數本文。	–
client_connection_check_interval	設定執行查詢時檢查是否中斷連線的時間間隔。	–

參數名稱	描述	預設
client_encoding	設定用戶端字元集編碼。	UTF8
client_min_messages	設定傳送給用戶端的訊息層級。	–
compute_query_id	計算查詢識別碼。	auto
config_file	設定伺服器主組態檔。	/rdsdbdata/config/postgresql.conf
constraint_exclusion	讓規劃器能夠使用限制條件來最佳化查詢。	–
cpu_index_tuple_cost	設定規劃器在索引掃描期間處理每個索引項目的成本估算。	–
cpu_operator_cost	設定規劃器處理每個運算子或函數呼叫的成本估算。	–
cpu_tuple_cost	設定規劃器處理每個元組 (資料列) 的成本估算。	–
cron.database_name	將資料庫設定為存放 pg_cron 中繼資料表	postgres
cron.log_run	將所有執行的任務記錄到 job_run_details 資訊表中	on
cron.log_statement	在執行之前記錄所有 cron 陳述式。	off
cron.max_running_jobs	可同時執行的任務數量上限。	5
cron.use_background_workers	允許 pg_cron 的背景工作者	on
cursor_tuple_fraction	設定規劃器對於將擷取之游標的資料列部分估算。	–
data_directory	設定伺服器資料目錄。	/rdsdbdata/db
datestyle	設定日期和時間值的顯示格式。	–

參數名稱	描述	預設
db_user_namespace	允許使用各資料庫的使用者名稱。	–
deadlock_timeout	(毫秒) 設定在檢查死鎖前等待鎖定的時間。	–
debug_pretty_print	將剖析和計劃樹狀顯示縮排。	–
debug_print_parse	記錄每項查詢的剖析樹狀結構。	–
debug_print_plan	記錄每項查詢的執行計畫。	–
debug_print_rewritten	記錄每項查詢的重寫剖析樹狀結構。	–
default_statistics_target	設定預設統計資訊目標。	–
default_tablespace	設定要在其中建立資料表和索引的預設資料表空間。	–
default_toast_compression	設定可壓縮值的預設壓縮方法。	–
default_transaction_deferrable	設定新交易的預設可延遲狀態。	–
default_transaction_isolation	設定每項新交易的交易隔離層級。	–
default_transaction_read_only	設定新交易的預設唯讀狀態。	–
effective_cache_size	(8kB) 設定規劃器對磁碟快取大小的假設。	SUM(DBInstanceClassMemory/12038,-50003)
effective_io_concurrency	磁碟子系統可以有效處理的同時要求數目。	–
enable_async_append	允許規劃器使用非同步附加計畫。	–

參數名稱	描述	預設
enable_bitmapscan	允許規劃器使用點陣圖掃描計畫。	–
enable_gathermerge	允許規劃器使用收集合併計畫。	–
enable_hashagg	允許規劃器使用雜湊彙總計畫。	–
enable_hashjoin	允許規劃器使用雜湊聯結計畫。	–
enable_incremental_sort	允許規劃器使用增量排序步驟。	–
enable_indexonlyscan	允許規劃器使用僅限索引掃描計畫。	–
enable_indexscan	允許規劃器使用索引掃描計畫。	–
enable_material	允許規劃器使用實體化。	–
enable_memoize	允許規劃器使用記憶化	–
enable_mergejoin	允許規劃器使用合併聯結計畫。	–
enable_nestloop	允許規劃器使用巢狀迴圈聯結計畫。	–
enable_parallel_append	允許規劃器使用平行附加計畫。	–
enable_parallel_hash	允許規劃器使用平行雜湊計畫。	–
enable_partition_pruning	啟用計畫時間和執行時間分割區剪除。	–
enable_partitionwise_aggregate	允許分割區彙總和分組。	–
enable_partitionwise_join	允許分割區聯結。	–
enable_seqscan	允許規劃器使用循序掃描計畫。	–

參數名稱	描述	預設
enable_sort	允許規劃器使用明確排序步驟。	–
enable_tidscan	允許規劃器使用 TID 掃描計畫。	–
escape_string_warning	警告一般字串常值中反斜線逸出。	–
exit_on_error	出現任何錯誤時終止工作階段。	–
extra_float_digits	設定針對浮點值顯示的位數。	–
force_parallel_mode	強制使用平行查詢設施。	–
from_collapse_limit	設定 FROM-list 大小，超過此大小就不會收合子查詢。	–
geqo	啟用基因查詢最佳化。	–
geqo_effort	GEQO：作業是用於設定其他 GEQO 參數的預設值。	–
geqo_generations	GEQO：演算法的反覆運算次數。	–
geqo_pool_size	GEQO：人口中的個體數目。	–
geqo_seed	GEQO：隨機路徑選取的種子。	–
geqo_selection_bias	GEQO：人口中的選擇壓力。	–
geqo_threshold	設定 FROM 項目的閾值，超出此閾值時就會使用 GEQO。	–
gin_fuzzy_search_limit	設定 GIN 確切搜尋所允許的結果上限。	–
gin_pending_list_limit	(kB) 設定 GIN 索引待定清單的大小上限。	–
hash_mem_multiplier	用於雜湊表的 work_mem 倍數。	–

參數名稱	描述	預設
hba_file	設定伺服器 hba 組態檔。	/rdsdbdata/config/pg_hba.conf
hot_standby_feedback	允許將回饋從熱待命傳送到主要伺服器，以避免查詢衝突。	on
huge_pages	減少資料庫執行個體處理大型連續記憶體區塊 (如共用緩衝區使用的記憶體區塊) 時的額外負荷。對於 t3.medium,db.t3.large、db.t4g.medium、db.t4g.large 類別以外的所有資料庫執行個體類別，預設為開啟 參數。	on
ident_file	設定伺服器 ident 組態檔。	/rdsdbdata/config/pg_ident.conf
idle_in_transaction_session_timeout	(毫秒) 設定任何閒置交易的允許持續時間上限。	86400000
idle_session_timeout	終止閒置時間已超過指定時間量，但不在所開啟交易中的工作階段 (亦即，等待用戶端查詢)	–
intervalstyle	設定間隔值的顯示格式。	–
join_collapse_limit	設定 FROM-list 大小，超過此大小就不會將 JOIN 結構扁平化。	–
krb_caseins_users	設定 GSSAPI (一般安全服務 API) 使用者名稱是否應該以不區分大小寫 (true) 方式來處理。根據預設，此參數會設為 false，因此 Kerberos 預期使用者名稱區分大小寫。如需詳細資訊，請參閱 PostgreSQL 文件中的 GSSAPI 身分驗證。	false
lc_messages	設定用來顯示訊息的語言。	–
lc_monetary	設定用於格式化貨幣金額的地區設定。	–
lc_numeric	設定用於格式化數字的地區設定。	–

參數名稱	描述	預設
lc_time	設定用於格式化日期和時間值的地區設定。	–
listen_addresses	設定接聽的主機名稱或 IP 地址。	*
lo_compat_privileges	允許回溯相容模式以進行大型物件的權限檢查。	0
log_autovacuum_min_duration	(毫秒) 設定執行時間下限，超出此時間就會記錄自動清空動作。	10000
log_connections	記錄每個成功連線。	–
log_destination	設定伺服器日誌輸出的目的地。	stderr
log_directory	設定日誌檔案的目的地目錄。	/rdsdbdata/log/error
log_disconnections	記錄工作階段的結尾，包括持續時間。	–
log_duration	記錄每個已完成 SQL 陳述式的持續時間。	–
log_error_verbosity	設定已記錄訊息的詳細資訊。	–
log_executor_stats	將執行器效能統計資訊寫入至伺服器日誌。	–
log_file_mode	設定日誌檔案的檔案許可。	0644
log_filename	設定日誌檔案的檔案名稱樣式。	postgresql.log.%Y-%m-%d-%H%M
logging_collector	開始子程序，將 stderr 輸出和/或 csvlog 擷取到日誌檔案中。	1
log_hostname	在連線日誌中記錄主機名稱。	0
logical_decoding_work_mem	(kB) 溢出到磁碟之前，每個內部重新排序緩衝區可以使用的記憶體量。	–
log_line_prefix	控制每個日誌行前綴的資訊。	%t:%r:%u@%d:%p]:
log_lock_waits	記錄長鎖定等待。	–

參數名稱	描述	預設
log_min_duration_sample	(毫秒) 設定執行時間下限，超出此時間就會記錄陳述式樣本。採樣由 log_statement_sample_rate 決定。	–
log_min_duration_statement	(毫秒) 設定執行時間下限，超出此時間就會記錄陳述式。	–
log_min_error_statement	導致所有陳述式在這個層級或以上產生要記錄的錯誤。	–
log_min_messages	設定所記錄的訊息層級。	–
log_parameter_max_length	(B) 記錄陳述式時，將記錄的參數值限制為前 N 個位元組。	–
log_parameter_max_length_on_error	(B) 報告錯誤時，將記錄的參數值限制為前 N 個位元組。	–
log_parser_stats	將剖析器效能統計資訊寫入至伺服器日誌。	–
log_planner_stats	將規劃器效能統計資訊寫入至伺服器日誌。	–
log_replication_commands	記錄每個複寫命令。	–
log_rotation_age	(分) 自動日誌檔案輪換將在 N 分鐘後發生。	60
log_rotation_size	(kB) 自動日誌檔案輪換將在 N KB 後發生。	100000
log_statement	設定已記錄的陳述式類型。	–
log_statement_sample_rate	陳述式超出 log_min_duration_sample 的待記錄部分。	–
log_statement_stats	將累積效能統計資訊寫入至伺服器日誌。	–
log_temp_files	(kB) 記錄使用大於此 KB 數的暫存檔案。	–
log_timezone	設定要在日誌訊息中使用的時區。	UTC

參數名稱	描述	預設
log_transaction_sample_rate	為新交易設定要記錄的交易部分。	–
log_truncate_on_rotation	在日誌輪換期間截斷名稱相同的現有日誌檔案。	0
maintenance_io_concurrency	用於維護作業的 effective_io_concurrency 變體。	1
maintenance_work_mem	(kB) 設定要用於維護作業的記憶體上限。	GREATEST(DBInstanceClassMemory/63963136*1024,65536)
max_connections	設定同時連線的數目上限。	LEAST(DBInstanceClassMemory/9531392,5000)
max_files_per_process	設定每個伺服器程序的同時開啟檔案數目上限。	–
max_locks_per_transaction	設定每項交易的鎖定數目上限。	64
max_logical_replication_workers	邏輯複寫工作者程序數上限。	–
max_parallel_maintenance_workers	設定每項維護作業的平行程序數上限。	–
max_parallel_workers	設定同時處於作用中狀態的平行工作者數上限。	GREATEST($DBInstanceVCPU/2,8)
max_parallel_workers_per_gather	設定每個執行器節點的平行程序數上限。	–

參數名稱	描述	預設
max_pred_locks_per_page	設定每頁的預測鎖定元組數上限。	–
max_pred_locks_per_relation	設定每個關係的預測鎖定分頁和元組數上限。	–
max_pred_locks_per_transaction	設定每項交易的述詞鎖定數目上限。	–
max_prepared_transactions	設定同時備妥交易的數目上限。	0
max_replication_slots	設定伺服器可支援的複寫槽數上限。	20
max_slot_wal_keep_size	(MB) 如果 WAL 占用了磁碟的這麼多空間，複寫槽將標記為失敗，並釋出區段進行刪除或回收。	–
max_stack_depth	(kB) 設定堆疊深度上限 (以 KB 為單位)。	6144
max_standby_streaming_delay	(毫秒) 設定熱待命伺服器處理已串流的 WAL 資料時，取消查詢前的延遲上限。	14000
max_sync_workers_per_subscription	每個訂閱的同步工作者數上限	2
max_wal_senders	設定同時執行 WAL 傳送器程序數上限。	10
max_worker_processes	設定並行工作者程序數上限。	GREATEST($DBInstanceVCPU*2, 8)
min_dynamic_shared_memory	(MB) 啟動時保留的動態共用記憶體數量。	–
min_parallel_index_scan_size	(8kB) 設定平行掃描的索引資料量下限。	–

參數名稱	描述	預設
min_parallel_table_scan_size	(8kB) 設定平行掃描的資料表資料量下限。	–
old_snapshot_threshold	(分) 經過多長時間後快照過時而無法讀取拍攝快照後更改的頁面。	–
orafce.nls_date_format	模擬 Oracle 日期輸出行為。	–
orafce.timezone	指定用於 sysdate 函數的時區。	–
parallel_leader_participation	控制「收集」和「收集合併」是否也執行子計畫。	–
parallel_setup_cost	設定規劃器啟動平行查詢工作者程序的成本估算。	–
parallel_tuple_cost	設定規劃器將每個元組 (資料列) 從工作者傳遞到主要後端的成本估算。	–
password_encryption	加密密碼。	–
pgaudit.log	指定工作階段稽核日誌記錄要記錄哪些陳述式類別。	–
pgaudit.log_catalog	指定在陳述式中的所有關係都在 pg_catalog 中的情況下，應啟用工作階段日誌記錄。	–
pgaudit.log_level	指定要用於日誌項目的日誌層級。	–
pgaudit.log_parameter	指定稽核日誌記錄應包括隨陳述式一起傳遞的參數。	–
pgaudit.log_relation	指定工作階段稽核日誌記錄是否應為 SELECT 或 DML 陳述式中參考的每個關係 (TABLE、VIEW 等) 建立單獨的日誌項目。	–

參數名稱	描述	預設
pgaudit.log_statement_once	指定日誌記錄包含的陳述式文字和參數，具有陳述式/子陳述式組合的第一個日誌項目，還是具有每個項目。	–
pgaudit.role	指定用於物件稽核日誌記錄的主要角色。	–
pg_bigm.enable_recheck	指定是否執行重新檢查 (全文檢索搜尋的內部程序)。	on
pg_bigm.gin_key_limit	指定用於全文檢索搜尋的搜尋關鍵字 2-gram 數上限。	0
pg_bigm.last_update	報告 pg_bigm 模組的上次更新日期。	2013.11.22
pg_bigm.similarity_limit	指定相似性搜尋使用的閾值下限。	0.3
pg_hint_plan.debug_print	記錄提示剖析的結果。	–
pg_hint_plan.enable_hint	強制規劃器使用查詢前的提示註解中指定的計畫。	–
pg_hint_plan.enable_hint_table	強制規劃器不透過使用資料表查閱來取得提示。	–
pg_hint_plan.message_level	偵錯訊息的訊息層級。	–
pg_hint_plan.parse_messages	剖析錯誤的訊息層級。	–
pglogical.batch_inserts	可能情況下批次插入	–
pglogical.conflict_log_level	設定用於記錄已解決衝突的日誌層級。	–

參數名稱	描述	預設
pglogical.conflict_resolution	設定用於可解決衝突的衝突解決方法。	–
pglogical.extra_connection_options	要新增到所有對等節點連線的連線選項	–
pglogical.synchronous_commit	pglogical 特定的同步遞交值	–
pglogical.use_spi	使用 SPI 而不是低層級 API 來套用變更	–
pgtle.clientauth_databases_to_skip	對於 clientauth 功能，要略過的資料庫清單。	–
pgtle.clientauth_db_name	控制用於 clientauth 功能的資料庫。	–
pgtle.clientauth_num_parallel_workers	用於 clientauth 功能的背景工作者數量。	–
pgtle.clientauth_users_to_skip	對於 clientauth 功能，要略過的使用者清單。	–
pgtle.enable_clientauth	啟用 clientauth 功能。	–
pgtle.passcheck_db_name	設定要用於整個叢集 passcheck 功能的資料庫。	–
pg_prewarm.autoprewarm	開始自動預熱工作者。	–
pg_prewarm.autoprewarm_interval	設定共用緩衝區傾印之間的間隔	–
pg_similarity.block_is_normalized	設定結果值是否標準化。	–

參數名稱	描述	預設
pg_similarity.block_threshold	設定 Block 相似性函數使用的閾值。	–
pg_similarity.block_tokenizer	設定 Block 相似性函數的字符化工具。	–
pg_similarity.cosine_is_normalized	設定結果值是否標準化。	–
pg_similarity.cosine_threshold	設定 Cosine 相似性函數使用的閾值。	–
pg_similarity.cosine_tokenizer	設定 Cosine 相似性函數的字符化工具。	–
pg_similarity.dice_is_normalized	設定結果值是否標準化。	–
pg_similarity.dice_threshold	設定 Dice 相似性量值使用的閾值。	–
pg_similarity.dice_tokenizer	設定 Dice 相似性量值的字符化工具。	–
pg_similarity.euclidean_is_normalized	設定結果值是否標準化。	–
pg_similarity.euclidean_threshold	設定 Euclidean 相似性量值使用的閾值。	–
pg_similarity.euclidean_tokenizer	設定 Euclidean 相似性量值的字符化工具。	–
pg_similarity.hamming_is_normalized	設定結果值是否標準化。	–
pg_similarity.hamming_threshold	設定 Block 相似性指標使用的閾值。	–

參數名稱	描述	預設
pg_similarity.jaccard_is_normalized	設定結果值是否標準化。	–
pg_similarity.jaccard_threshold	設定 Jaccard 相似性量值使用的閾值。	–
pg_similarity.jaccard_tokenizer	設定 Jaccard 相似性量值的字符化工具。	–
pg_similarity.jaro_is_normalized	設定結果值是否標準化。	–
pg_similarity.jaro_threshold	設定 Jaro 相似性量值使用的閾值。	–
pg_similarity.jarowinkler_is_normalized	設定結果值是否標準化。	–
pg_similarity.jarowinkler_threshold	設定 Jarowinkler 相似性量值使用的閾值。	–
pg_similarity.levenshtein_is_normalized	設定結果值是否標準化。	–
pg_similarity.levenshtein_threshold	設定 Levenshtein 相似性量值使用的閾值。	–
pg_similarity.matching_is_normalized	設定結果值是否標準化。	–
pg_similarity.matching_threshold	設定匹配係數相似性量值使用的閾值。	–
pg_similarity.matching_tokenizer	設定匹配係數相似性量值的字符化工具。	–
pg_similarity.mongeelkan_is_normalized	設定結果值是否標準化。	–

參數名稱	描述	預設
pg_similarity.mongeelkan_threshold	設定 Monge-Elkan 相似性量值使用的閾值。	–
pg_similarity.mongeelkan_tokenizer	設定 Monge-Elkan 相似性量值的字符化工具。	–
pg_similarity.nw_gap_penalty	設定 Needleman-Wunsch 相似性量值使用的空位罰分。	–
pg_similarity.nw_is_normalized	設定結果值是否標準化。	–
pg_similarity.nw_threshold	設定 Needle-Munsch 相似性量值使用的閾值。	–
pg_similarity.overlap_is_normalized	設定結果值是否標準化。	–
pg_similarity.overlap_threshold	設定重疊係數相似性量值使用的閾值。	–
pg_similarity.overlap_tokenizer	設定重疊係數相似性量值的字符化工具。	–
pg_similarity.qgram_is_normalized	設定結果值是否標準化。	–
pg_similarity.qgram_threshold	設定 Q-Gram 相似性量值使用的閾值。	–
pg_similarity.qgram_tokenizer	設定 Q-Gram 量值的字符化工具。	–
pg_similarity.swg_is_normalized	設定結果值是否標準化。	–
pg_similarity.swg_threshold	設定 Smith-Waterman-Gotoh 相似性量值使用的閾值。	–

參數名稱	描述	預設
pg_similarity.sw_is_normalized	設定結果值是否標準化。	–
pg_similarity.sw_threshold	設定 Smith-Waterman 相似性量值使用的閾值。	–
pg_stat_statements.max	設定 pg_stat_statements 追蹤的陳述式數上限。	–
pg_stat_statements.save	儲存伺服器關閉期間的 pg_stat_list 統計數字。	–
pg_stat_statements.track	選擇 pg_stat_statements 追蹤哪些陳述式。	–
pg_stat_statements.track_planning	選擇 pg_stat_statements 是否追蹤計畫持續時間。	–
pg_stat_statements.track_utility	選擇 pg_stat_statements 是否追蹤實用程序命令。	–
plan_cache_mode	控制自訂計畫或通用計畫的規劃器選擇。	–
port	設定伺服器在哪個 TCP 連接埠上接聽。	EndPointPort
postgis.gdal_enabled_drivers	啟用或停用 Postgres 9.3.5 及更新版本中搭配 PostGIS 使用的 GDAL 驅動程式。	ENABLE_ALL
quote_all_identifiers	產生 SQL 片段時為所有識別符新增引號。	–
random_page_cost	設定規劃器對於非循序擷取磁碟分頁的成本估算。	–
rdkit.dice_threshold	Dice 相似性的較低閾值。相似性低於閾值的分子與 # 操作不相似。	–
rdkit.do_chiral_sss	子結構相符時應考慮立體化學。若為 false，則不會在子結構相符項目中使用立體化學資訊。	–

參數名稱	描述	預設
rdkit.tanimoto_threshold	Tanimoto 相似性的較低閾值。相似性低於閾值的分子與 % 操作不相似。	–
rds.accepted_password_auth_method	使用本機儲存的密碼強制驗證連線。	md5+scram
rds.adaptive_autovacuum	RDS 參數用於啟用/停用調整式自動清空。	1
rds.babelfish_status	RDS 參數用於啟用/停用 Babelfish for Aurora PostgreSQL。	off
rds.enable_plan_management	啟用或停用 apg_plan_mgmt 擴充功能。	0

參數名稱	描述	預設
rds.extensions	RDS 提供的延伸模組清單	address_standardizer、address_standardizer_data_us、apg_plan_mgmt、aurora_stat_utils、amcheck、autoinc、aws_commons、aws_ml、aws_s3、aws_lambda、bool_plperl、bloom、btree_gin、btree_gist、citext、cube、dblink、dict_int、dict_xsyn、earthdistance、fuzzystrmatch、hll、hstore、hstore_plperl、insert_username、intagg、intarray、ip4r、isn、jsonb_plperl、lo、log_fdw、ltree、moddatetime、old_snapshot、oracle_fdw、orafce、pgaudit、pgcrypto、pglogical、pgrouting、pgrowlocks、pgstattuple、pgtap、pg_bigm、pg_buffercache、pg_cron、pg_freespacemap、pg_hint_plan、pg_partm

參數名稱	描述	預設
		an、pg_prewarm、pg_proctab、pg_repack、pg_similarity、pg_stat_statements、pg_trgm、pg_visibility、plcoffee、plls、plperl、plpgsql、plprofiler、pltcl、plv8、postgis、postgis_tiger_geocoder、postgis_raster、postgis_topology、postgres_fdw、prefix、rdkit、rds_tools、refint、sslinfo、tablefunc、tds_fdw、test_parser、tsm_system_rows、tsm_system_time、unaccent、uuid-ossp
rds.force_admin_logging_level	查看客户資料庫中 RDS 管理員使用者動作的日誌訊息。	–
rds.force_autovacuum_logging_level	查看與自動清空作業相關的日誌訊息。	WARNING
rds.force_ssl	強制 SSL 連線。	0

參數名稱	描述	預設
rds.global_db_rpo	(秒) 還原點目標閾值 (以秒為單位)，違反此閾值時會阻止使用者遞交。 **Important** 此參數主要用於 Aurora PostgreSQL 型全球資料庫。對於非全球資料庫，請保留預設值。如需使用此參數的詳細資訊，請參閱 the section called “管理 RPO (Aurora PostgreSQL)”。	–
rds.logical_replication	允許邏輯解碼。	0
rds.logically_replicate_unlogged_tables	未記錄的資料表是以邏輯方式複寫。	1
rds.log_retention_period	Amazon RDS 會刪除超過 N 分鐘的 PostgreSQL 日誌	4320
rds.pg_stat_ramdisk_size	統計資料 Ramdisk 的大小 (以 MB 為單位) 。若為非零值將設定 Ramdisk。此參數僅適用於Aurora PostgreSQL 14 及更低版本。	0
rds.rds_superuser_reserved_connections	設定為 rds_superusers 保留的連線槽數。此參數僅適用於第 15 版和更舊版本。如需詳細資訊，請參閱 PostgreSQL 文件保留的連線。	2
rds.restrict_password_commands	將與密碼相關的命令限制為 rds_password 的成員	–
rds.superuser_variables	僅限進階使用者變數清單，我們提升了 rds_superuser 修改陳述式。	session_replication_role
recovery_init_sync_method	設定在損毀復原之前同步處理資料目錄的方法。	syncfs

參數名稱	描述	預設
remove_temp_files_after_crash	後端當機後移除暫存檔案。	0
restart_after_crash	後端當機後重新初始化伺服器。	–
row_security	啟用資料列安全性。	–
search_path	針對不符合結構描述的名稱設定結構描述搜尋順序。	–
seq_page_cost	設定規劃器對循序擷取磁碟分頁的成本估算。	–
session_replication_role	設定觸發器和重寫規則的工作階段行為。	–
shared_buffers	(8kB) 設定伺服器所用的共用記憶體緩衝區數目。	SUM(DBInstanceClassMemory/12038,-50003)
shared_preload_libraries	列出要預先載入到伺服器的共用程式庫。	pg_stat_statements
ssl	啟用 SSL 連線。	1
ssl_ca_file	SSL 伺服器授權檔案的位置。	/rdsdbdata/rds-metadata/ca-cert.pem
ssl_cert_file	SSL 伺服器憑證檔案的位置。	/rdsdbdata/rds-metadata/server-cert.pem
ssl_ciphers	設定要在安全連線上使用的允許 TLS 密碼清單。	–
ssl_crl_dir	SSL 憑證撤銷清單目錄的位置。	/rdsdbdata/rds-metadata/ssl_crl_dir/
ssl_key_file	SSL 伺服器私有金鑰檔案的位置	/rdsdbdata/rds-metadata/server-key.pem

參數名稱	描述	預設
ssl_max_protocol_version	設定允許的最高 SSL/TLS 通訊協定版本	–
ssl_min_protocol_version	設定允許的最低 SSL/TLS 通訊協定版本	TLSv1.2
standard_conforming_strings	導致 ... 字串逐字地處理反斜線。	–
statement_timeout	(毫秒) 設定任何陳述式允許的持續時間上限。	–
stats_temp_directory	將暫存統計數字檔案寫入指定目錄。	/rdsdbdata/db/pg_stat_tmp
superuser_reserved_connections	設定為進階使用者保留的連線槽數。	3
synchronize_seqscans	啟用已同步的循序掃描。	–
synchronous_commit	設定目前交易的同步層級。	on
tcp_keepalives_count	TCP 保持連線重新傳輸的數量上限	–
tcp_keepalives_idle	(秒)發出 TCP 存留之間的時間。	–
tcp_keepalives_interval	(秒) TCP 存留重新傳輸之間的時間。	–
temp_buffers	(8kB) 設定每個工作階段所用的臨時緩衝區數上限。	–
temp_file_limit	限制指定 PostgreSQL 程序可用於暫存檔案的磁碟空間總量 (以 KB 為單位)，不包括用於明確暫存資料表的空間	-1
temp_tablespaces	設定要用於暫存資料表和排序檔案的資料表空間。	–

參數名稱	描述	預設
timezone	設定可供顯示和解譯時間戳記的時區。	UTC
track_activities	收集有關執行命令的資訊。	–
track_activity_query_size	設定為 pg_stat_activity.current_query 保留的大小 (以位元組為單位)。	4096
track_commit_timestamp	Collects transaction commit time.	–
track_counts	收集資料庫活動的統計資訊。	–
track_functions	收集資料庫活動的功能層級統計資訊。	pl
track_io_timing	收集資料庫輸入/輸出活動的計時統計數字。	1
track_wal_io_timing	收集 WAL I/O 活動的計時統計資料。	–
transform_null_equals	將 expr=NULL 視為 expr IS NULL。	–
update_process_title	更新程序標題以顯示作用中的 SQL 命令。	–
vacuum_cost_delay	(毫秒) 清空成本延遲 (以毫秒為單位)。	–
vacuum_cost_limit	在小憩前可用的清理成本金額。	–
vacuum_cost_page_dirty	清理所變更之頁面的清理成本。	–
vacuum_cost_page_hit	在緩衝區快取中找到之頁面的清理成本。	–
vacuum_cost_page_miss	在緩衝區快取中找不到之頁面的清理成本。	0
vacuum_defer_cleanup_age	應延遲清空和熱清理 (如果有的話) 的交易數目。	–

參數名稱	描述	預設
vacuum_failsafe_age	清空作業應觸發失效安全的存留期，以避免環繞式中斷。	1200000000
vacuum_freeze_min_age	清空作業應凍結資料表列的存留期下限。	–
vacuum_freeze_table_age	清空作業應掃描整個資料表以凍結元組的存留期。	–
vacuum_multixact_failsafe_age	清空作業應觸發失效安全的 Multixact 存留期，以避免環繞式中斷。	1200000000
vacuum_multixact_freeze_min_age	清空作業應凍結資料表列中 MultiXactId 的存留期下限。	–
vacuum_multixact_freeze_table_age	清空作業應掃描整個資料表以凍結元組的 Multixact 存留期。	–
wal_buffers	(8kB) 為 WAL 設定共用記憶體中的磁碟分頁緩衝區數目。	–
wal_receiver_create_temp_slot	設定 WAL 接收機在設定永久槽時是否應建立臨時複寫槽。	0
wal_receiver_status_interval	(秒) 設定向主伺服器報告 WAL 接收器狀態之間的間隔上限。	–
wal_receiver_timeout	(毫秒) 設定從主伺服器接收資料的等待時間上限。	30000
wal_sender_timeout	(毫秒) 設定等待 WAL 複寫的等待時間上限。	–
work_mem	(kB) 設定要用於查詢工作空間的記憶體上限。	–
xmlbinary	設定有多少二進位值要在 XML 中編碼。	–
xmloption	設定要將明確剖析和序列化作業中的 XML 資料視為文件或內容片段。	–

Aurora PostgreSQL 執行個體層級參數

您可以使用 AWS 管理主控台、 AWS CLI 或 Amazon RDS API，檢視特定 Aurora PostgreSQL 版本可用的執行個體層級參數。如需在 RDS 主控台中檢視 Aurora PostgreSQL 資料庫參數群組中參數的相關資訊，請參閱 在 Amazon Aurora 中檢視資料庫參數群組的參數值。

部分執行個體層級參數並不適用於所有版本，而部分正被棄用。如需檢視特定 Aurora PostgreSQL 版本之參數的相關資訊，請參閱 查看 Aurora PostgreSQL 資料庫叢集和資料庫參數。

例如，下表列出套用至 Aurora PostgreSQL 資料庫叢集中特定資料庫執行個體的所有參數。此清單是透過對 `--db-parameter-group-name`值執行 describe-db-parameters AWS CLI 命令`default.aurora-postgresql14`而產生。

如需相同預設資料庫參數群組的資料庫叢集參數清單，請參閱 Aurora PostgreSQL 叢集層級參數。

參數名稱	描述	預設
apg_enable_batch_mode_function_execution	啟用批次模式函數，以便一次處理一組資料列。	–
apg_enable_correlated_any_transform	允許規劃器盡可能將相關的任何子連結 (IN/NOT IN 子查詢) 轉換為 JOIN。	–
apg_enable_function_migration	允許規劃器將符合條件的純量函數遷移到 FROM 子句。	–
apg_enable_not_in_transform	允許規劃器盡可能將 NOT IN 子查詢轉換為 ANTI JOIN。	–
apg_enable_remove_redundant_inner_joins	允許規劃器移除冗餘的內部聯結。	–
apg_enable_semijoin_push_down	允許使用半聯結篩選器進行雜湊聯結。	–
apg_plan_mgmt.capture_plan_baselines	擷取計畫基準模式。手動 - 啟用任何 SQL 陳述式的計畫擷取，關閉 - 停用計畫擷取，自動 - 為 pg_stat_陳述式中滿足合格條件的陳述式啟用計畫擷取。	off

參數名稱	描述	預設
apg_plan_mgmt.max_databases	設定可使用 apg_plan_mgmt 管理查詢的資料庫數上限。	10
apg_plan_mgmt.max_plans	設定 apg_plan_mgmt 可快取的計畫數上限。	10000
apg_plan_mgmt.plan_retention_period	上次使用計畫後經過幾天自動刪除計畫的天數上限。	32
apg_plan_mgmt.unapproved_plan_execution_threshold	估計總計畫成本，低於此成本將執行未核准的計畫。	0
apg_plan_mgmt.use_plan_baselines	受管陳述式僅使用已核准或固定計畫。	false
application_name	設定要在統計資訊和日誌中報告的應用程式名稱。	–
aurora_compute_plan_id	監控查詢執行計劃，以偵測造成目前資料庫負載的執行計劃，以及追蹤執行計劃隨時間的效能統計資料。如需更多詳細資訊，請參閱監控 Aurora PostgreSQL 的查詢執行計劃。	on
aurora_temp_space_size	(MB) 針對具有支援執行個體類別的 Aurora I/O 最佳化叢集上已啟用最佳化讀取的暫存物件，設定分配的空間大小。	DBInstanceClassMemory/524288
authentication_timeout	(秒) 設定完成用戶端身分驗證的允許時間上限。	–
auto_explain.log_analyze	將 EXPLAIN ANALYZE 用於計畫日誌記錄。	–
auto_explain.log_buffers	日誌緩衝區使用情況。	–

參數名稱	描述	預設
auto_explain.log_format	用於計畫日誌記錄的 EXPLAIN 格式。	–
auto_explain.log_min_duration	設定執行時間下限，超出此時間就會記錄計畫。	–
auto_explain.log_nested_statements	記錄巢狀陳述式。	–
auto_explain.log_timing	收集計時資料而不僅是資料列數。	–
auto_explain.log_triggers	在計畫中包含觸發程序統計數字。	–
auto_explain.log_verbose	將 EXPLAIN VERBOSE 用於計畫日誌記錄。	–
auto_explain.sample_rate	待處理查詢的一部分。	–
babelfishpg_tds.listen_addresses	設定用來接聽 TDS 的主機名稱或 IP 地址。	*
babelfishpg_tds.tds_debug_log_level	將 TDS 中的日誌記錄層級設為 0 會停用日誌記錄	1
backend_flush_after	(8Kb) 幾個分頁後將先前執行的寫入排清到磁碟。	–
bytea_output	設定 bytea 的輸出格式。	–
check_function_bodies	在 CREATE FUNCTION 執行期間檢查函數本文。	–
client_connection_check_interval	設定執行查詢時檢查是否中斷連線的時間間隔。	–

參數名稱	描述	預設
client_min_messages	設定傳送給用戶端的訊息層級。	–
config_file	設定伺服器主組態檔。	/rdsdbdata/config/postgresql.conf
constraint_exclusion	讓規劃器能夠使用限制條件來最佳化查詢。	–
cpu_index_tuple_cost	設定規劃器在索引掃描期間處理每個索引項目的成本估算。	–
cpu_operator_cost	設定規劃器處理每個運算子或函數呼叫的成本估算。	–
cpu_tuple_cost	設定規劃器處理每個元組 (資料列) 的成本估算。	–
cron.database_name	將資料庫設定為存放 pg_cron 中繼資料表	postgres
cron.log_run	將所有執行的任務記錄到 job_run_details 資訊表中	on
cron.log_statement	在執行之前記錄所有 cron 陳述式。	off
cron.max_running_jobs	可同時執行的任務數量上限。	5
cron.use_background_workers	允許 pg_cron 的背景工作者	on
cursor_tuple_fraction	設定規劃器對於將擷取之游標的資料列部分估算。	–
db_user_namespace	允許使用各資料庫的使用者名稱。	–
deadlock_timeout	(毫秒) 設定在檢查死鎖前等待鎖定的時間。	–
debug_pretty_print	將剖析和計劃樹狀顯示縮排。	–
debug_print_parse	記錄每項查詢的剖析樹狀結構。	–

參數名稱	描述	預設
debug_print_plan	記錄每項查詢的執行計畫。	–
debug_print_rewritten	記錄每項查詢的重寫剖析樹狀結構。	–
default_statistics_target	設定預設統計資訊目標。	–
default_transaction_deferrable	設定新交易的預設可延遲狀態。	–
default_transaction_isolation	設定每項新交易的交易隔離層級。	–
default_transaction_read_only	設定新交易的預設唯讀狀態。	–
effective_cache_size	(8kB) 設定規劃器對磁碟快取大小的假設。	SUM(DBInstanceClassMemory/12038,-50003
effective_io_concurrency	磁碟子系統可以有效處理的同時要求數目。	–
enable_async_append	允許規劃器使用非同步附加計畫。	–
enable_bitmapscan	允許規劃器使用點陣圖掃描計畫。	–
enable_gathermerge	允許規劃器使用收集合併計畫。	–
enable_hashagg	允許規劃器使用雜湊彙總計畫。	–
enable_hashjoin	允許規劃器使用雜湊聯結計畫。	–
enable_incremental_sort	允許規劃器使用增量排序步驟。	–
enable_indexonlyscan	允許規劃器使用僅限索引掃描計畫。	–
enable_indexscan	允許規劃器使用索引掃描計畫。	–

參數名稱	描述	預設
enable_material	允許規劃器使用實體化。	–
enable_memoize	允許規劃器使用記憶化	–
enable_mergejoin	允許規劃器使用合併聯結計畫。	–
enable_nestloop	允許規劃器使用巢狀迴圈聯結計畫。	–
enable_parallel_append	允許規劃器使用平行附加計畫。	–
enable_parallel_hash	允許規劃器使用平行雜湊計畫。	–
enable_partition_pruning	啟用計畫時間和執行時間分割區剪除。	–
enable_partitionwise_aggregate	允許分割區彙總和分組。	–
enable_partitionwise_join	允許分割區聯結。	–
enable_seqscan	允許規劃器使用循序掃描計畫。	–
enable_sort	允許規劃器使用明確排序步驟。	–
enable_tidscan	允許規劃器使用 TID 掃描計畫。	–
escape_string_warning	警告一般字串常值中反斜線逸出。	–
exit_on_error	出現任何錯誤時終止工作階段。	–
force_parallel_mode	強制使用平行查詢設施。	–
from_collapse_limit	設定 FROM-list 大小，超過此大小就不會收合子查詢。	–
geqo	啟用基因查詢最佳化。	–

參數名稱	描述	預設
geqo_effort	GEQO：作業是用於設定其他 GEQO 參數的預設值。	–
geqo_generations	GEQO：演算法的反覆運算次數。	–
geqo_pool_size	GEQO：人口中的個體數目。	–
geqo_seed	GEQO：隨機路徑選取的種子。	–
geqo_selection_bias	GEQO：人口中的選擇壓力。	–
geqo_threshold	設定 FROM 項目的閾值，超出此閾值時就會使用 GEQO。	–
gin_fuzzy_search_limit	設定 GIN 確切搜尋所允許的結果上限。	–
gin_pending_list_limit	(kB) 設定 GIN 索引待定清單的大小上限。	–
hash_mem_multiplier	用於雜湊表的 work_mem 倍數。	–
hba_file	設定伺服器 hba 組態檔。	/rdsdbdata/config/pg_hba.conf
hot_standby_feedback	允許將回饋從熱待命傳送到主要伺服器，以避免查詢衝突。	on
ident_file	設定伺服器 ident 組態檔。	/rdsdbdata/config/pg_ident.conf
idle_in_transaction_session_timeout	(毫秒) 設定任何閒置交易的允許持續時間上限。	86400000
idle_session_timeout	終止閒置時間已超過指定時間量，但不在所開啟交易中的工作階段 (亦即，等待用戶端查詢)	–
join_collapse_limit	設定 FROM-list 大小，超過此大小就不會將 JOIN 結構扁平化。	–
lc_messages	設定用來顯示訊息的語言。	–

參數名稱	描述	預設
listen_addresses	設定接聽的主機名稱或 IP 地址。	*
lo_compat_privileges	允許回溯相容模式以進行大型物件的權限檢查。	0
log_connections	記錄每個成功連線。	–
log_destination	設定伺服器日誌輸出的目的地。	stderr
log_directory	設定日誌檔案的目的地目錄。	/rdsdbdata/log/error
log_disconnections	記錄工作階段的結尾，包括持續時間。	–
log_duration	記錄每個已完成 SQL 陳述式的持續時間。	–
log_error_verbosity	設定已記錄訊息的詳細資訊。	–
log_executor_stats	將執行器效能統計資訊寫入至伺服器日誌。	–
log_file_mode	設定日誌檔案的檔案許可。	0644
log_filename	設定日誌檔案的檔案名稱樣式。	postgresql.log.%Y-%m-%d-%H%M
logging_collector	開始子程序，將 stderr 輸出和/或 csvlog 擷取到日誌檔案中。	1
log_hostname	在連線日誌中記錄主機名稱。	0
logical_decoding_work_mem	(kB) 溢出到磁碟之前，每個內部重新排序緩衝區可以使用的記憶體量。	–
log_line_prefix	控制每個日誌行前綴的資訊。	%t:%r:%u@%d:%p]:
log_lock_waits	記錄長鎖定等待。	–
log_min_duration_sample	(毫秒) 設定執行時間下限，超出此時間就會記錄陳述式樣本。採樣由 log_statement_sample_rate 決定。	–

參數名稱	描述	預設
log_min_duration_statement	(毫秒) 設定執行時間下限，超出此時間就會記錄陳述式。	–
log_min_error_statement	導致所有陳述式在這個層級或以上產生要記錄的錯誤。	–
log_min_messages	設定所記錄的訊息層級。	–
log_parameter_max_length	(B) 記錄陳述式時，將記錄的參數值限制為前 N 個位元組。	–
log_parameter_max_length_on_error	(B) 報告錯誤時，將記錄的參數值限制為前 N 個位元組。	–
log_parser_stats	將剖析器效能統計資訊寫入至伺服器日誌。	–
log_planner_stats	將規劃器效能統計資訊寫入至伺服器日誌。	–
log_replication_commands	記錄每個複寫命令。	–
log_rotation_age	(分) 自動日誌檔案輪換將在 N 分鐘後發生。	60
log_rotation_size	(kB) 自動日誌檔案輪換將在 N KB 後發生。	100000
log_statement	設定已記錄的陳述式類型。	–
log_statement_sample_rate	陳述式超出 log_min_duration_sample 的待記錄部分。	–
log_statement_stats	將累積效能統計資訊寫入至伺服器日誌。	–
log_temp_files	(kB) 記錄使用大於此 KB 數的暫存檔案。	–
log_timezone	設定要在日誌訊息中使用的時區。	UTC
log_truncate_on_rotation	在日誌輪換期間截斷名稱相同的現有日誌檔案。	0

參數名稱	描述	預設
maintenance_io_concurrency	用於維護作業的 effective_io_concurrency 變體。	1
maintenance_work_mem	(kB) 設定要用於維護作業的記憶體上限。	GREATEST(DBInstanceClassMemory/63963136*1024,65536
max_connections	設定同時連線的數目上限。	LEAST(DBInstanceClassMemory/9531392,5000
max_files_per_process	設定每個伺服器程序的同時開啟檔案數目上限。	–
max_locks_per_transaction	設定每項交易的鎖定數目上限。	64
max_parallel_maintenance_workers	設定每項維護作業的平行程序數上限。	–
max_parallel_workers	設定同時處於作用中狀態的平行工作者數上限。	GREATEST($DBInstanceVCPU/2,8
max_parallel_workers_per_gather	設定每個執行器節點的平行程序數上限。	–
max_pred_locks_per_page	設定每頁的預測鎖定元組數上限。	–
max_pred_locks_per_relation	設定每個關係的預測鎖定分頁和元組數上限。	–
max_pred_locks_per_transaction	設定每項交易的述詞鎖定數目上限。	–

參數名稱	描述	預設
max_slot_wal_keep_size	(MB) 如果 WAL 占用了磁碟的這麼多空間，複寫槽將標記為失敗，並釋出區段進行刪除或回收。	–
max_stack_depth	(kB) 設定堆疊深度上限 (以 KB 為單位)。	6144
max_standby_streaming_delay	(毫秒) 設定熱待命伺服器處理已串流的 WAL 資料時，取消查詢前的延遲上限。	14000
max_worker_processes	設定並行工作者程序數上限。	GREATEST($DBInstanceVCPU*2, 8
min_dynamic_shared_memory	(MB) 啟動時保留的動態共用記憶體數量。	–
min_parallel_index_scan_size	(8kB) 設定平行掃描的索引資料量下限。	–
min_parallel_table_scan_size	(8kB) 設定平行掃描的資料表資料量下限。	–
old_snapshot_threshold	(分) 經過多長時間後快照過時而無法讀取拍攝快照後更改的頁面。	–
parallel_leader_participation	控制「收集」和「收集合併」是否也執行子計畫。	–
parallel_setup_cost	設定規劃器啟動平行查詢工作者程序的成本估算。	–
parallel_tuple_cost	設定規劃器將每個元組 (資料列) 從工作者傳遞到主要後端的成本估算。	–
pgaudit.log	指定工作階段稽核日誌記錄要記錄哪些陳述式類別。	–
pgaudit.log_catalog	指定在陳述式中的所有關係都在 pg_catalog 中的情況下，應啟用工作階段日誌記錄。	–

參數名稱	描述	預設
pgaudit.log_level	指定要用於日誌項目的日誌層級。	–
pgaudit.log_parameter	指定稽核日誌記錄應包括隨陳述式一起傳遞的參數。	–
pgaudit.log_relation	指定工作階段稽核記錄是否應為 SELECT 或 DML 陳述式中參考的每個關係 (TABLE、VIEW 等) 建立單獨的日誌項目。	–
pgaudit.log_statement_once	指定日誌記錄包含的陳述式文字和參數，具有陳述式/子陳述式組合的第一個日誌項目，還是具有每個項目。	–
pgaudit.role	指定用於物件稽核日誌記錄的主要角色。	–
pg_bigm.enable_recheck	指定是否執行重新檢查 (全文檢索搜尋的內部程序)。	on
pg_bigm.gin_key_limit	指定用於全文檢索搜尋的搜尋關鍵字 2-gram 數上限。	0
pg_bigm.last_update	報告 pg_bigm 模組的上次更新日期。	2013.11.22
pg_bigm.similarity_limit	指定相似性搜尋使用的閾值下限。	0.3
pg_hint_plan.debug_print	記錄提示剖析的結果。	–
pg_hint_plan.enable_hint	強制規劃器使用查詢前的提示註解中指定的計畫。	–
pg_hint_plan.enable_hint_table	強制規劃器不透過使用資料表查閱來取得提示。	–
pg_hint_plan.message_level	偵錯訊息的訊息層級。	–

參數名稱	描述	預設
pg_hint_plan.parse_messages	剖析錯誤的訊息層級。	–
pglogical.batch_inserts	可能情況下批次插入	–
pglogical.conflict_log_level	設定用於記錄已解決衝突的日誌層級。	–
pglogical.conflict_resolution	設定用於可解決衝突的衝突解決方法。	–
pglogical.extra_connection_options	要新增到所有對等節點連線的連線選項	–
pglogical.synchronous_commit	pglogical 特定的同步遞交值	–
pglogical.use_spi	使用 SPI 而不是低層級 API 來套用變更	–
pg_similarity.block_is_normalized	設定結果值是否標準化。	–
pg_similarity.block_threshold	設定 Block 相似性函數使用的閾值。	–
pg_similarity.block_tokenizer	設定 Block 相似性函數的字符化工具。	–
pg_similarity.cosine_is_normalized	設定結果值是否標準化。	–
pg_similarity.cosine_threshold	設定 Cosine 相似性函數使用的閾值。	–
pg_similarity.cosine_tokenizer	設定 Cosine 相似性函數的字符化工具。	–

參數名稱	描述	預設
pg_similarity.dice_is_normalized	設定結果值是否標準化。	–
pg_similarity.dice_threshold	設定 Dice 相似性量值使用的閾值。	–
pg_similarity.dice_tokenizer	設定 Dice 相似性量值的字符化工具。	–
pg_similarity.euclidean_is_normalized	設定結果值是否標準化。	–
pg_similarity.euclidean_threshold	設定 Euclidean 相似性量值使用的閾值。	–
pg_similarity.euclidean_tokenizer	設定 Euclidean 相似性量值的字符化工具。	–
pg_similarity.hamming_is_normalized	設定結果值是否標準化。	–
pg_similarity.hamming_threshold	設定 Block 相似性指標使用的閾值。	–
pg_similarity.jaccard_is_normalized	設定結果值是否標準化。	–
pg_similarity.jaccard_threshold	設定 Jaccard 相似性量值使用的閾值。	–
pg_similarity.jaccard_tokenizer	設定 Jaccard 相似性量值的字符化工具。	–
pg_similarity.jaro_is_normalized	設定結果值是否標準化。	–
pg_similarity.jaro_threshold	設定 Jaro 相似性量值使用的閾值。	–

參數名稱	描述	預設
pg_similarity.jaro winkler_is_normalized	設定結果值是否標準化。	–
pg_similarity.jaro winkler_threshold	設定 Jarowinkler 相似性量值使用的閾值。	–
pg_similarity.leve nshtein_is_normalized	設定結果值是否標準化。	–
pg_similarity.leve nshtein_threshold	設定 Levenshtein 相似性量值使用的閾值。	–
pg_similarity.matc hing_is_normalized	設定結果值是否標準化。	–
pg_similarity.matc hing_threshold	設定匹配係數相似性量值使用的閾值。	–
pg_similarity.matc hing_tokenizer	設定匹配係數相似性量值的字符化工具。	–
pg_similarity.mong eelkan_is_normalized	設定結果值是否標準化。	–
pg_similarity.mong eelkan_threshold	設定 Monge-Elkan 相似性量值使用的閾值。	–
pg_similarity.mong eelkan_tokenizer	設定 Monge-Elkan 相似性量值的字符化工具。	–
pg_similarity.nw_g ap_penalty	設定 Needleman-Wunsch 相似性量值使用的空位罰分。	–
pg_similarity.nw_i s_normalized	設定結果值是否標準化。	–
pg_similarity.nw_t hreshold	設定 Needle-Munsch 相似性量值使用的閾值。	–

參數名稱	描述	預設
pg_similarity.overlap_is_normalized	設定結果值是否標準化。	–
pg_similarity.overlap_threshold	設定重疊係數相似性量值使用的閾值。	–
pg_similarity.overlap_tokenizer	設定重疊係數相似性量值的字符化工具。	–
pg_similarity.qgram_is_normalized	設定結果值是否標準化。	–
pg_similarity.qgram_threshold	設定 Q-Gram 相似性量值使用的閾值。	–
pg_similarity.qgram_tokenizer	設定 Q-Gram 量值的字符化工具。	–
pg_similarity.swg_is_normalized	設定結果值是否標準化。	–
pg_similarity.swg_threshold	設定 Smith-Waterman-Gotoh 相似性量值使用的閾值。	–
pg_similarity.sw_is_normalized	設定結果值是否標準化。	–
pg_similarity.sw_threshold	設定 Smith-Waterman 相似性量值使用的閾值。	–
pg_stat_statements.max	設定 pg_stat_statements 追蹤的陳述式數上限。	–
pg_stat_statements.save	儲存伺服器關閉期間的 pg_stat_list 統計數字。	–
pg_stat_statements.track	選擇 pg_stat_statements 追蹤哪些陳述式。	–

參數名稱	描述	預設
pg_stat_statements.track_planning	選擇 pg_stat_statements 是否追蹤計畫持續時間。	–
pg_stat_statements.track_utility	選擇 pg_stat_statements 是否追蹤實用程序命令。	–
postgis.gdal_enabled_drivers	啟用或停用 Postgres 9.3.5 及更新版本中搭配 PostGIS 使用的 GDAL 驅動程式。	ENABLE_ALL
quote_all_identifiers	產生 SQL 片段時為所有識別符新增引號。	–
random_page_cost	設定規劃器對於非循序擷取磁碟分頁的成本估算。	–
rds.enable_memory_management	改善 Aurora PostgreSQL 12.17、13.13、14.10、15.5 及更新版本的記憶體管理功能，可防止因可用記憶體不足而導致的穩定性問題和資料庫重新啟動。如需詳細資訊，請參閱Aurora PostgreSQL 中的記憶體管理已改善。	True
rds.force_admin_logging_level	查看客戶資料庫中 RDS 管理員使用者動作的日誌訊息。	–
rds.log_retention_period	Amazon RDS 會刪除超過 N 分鐘的 PostgreSQL 日誌	4320
rds.memory_allocation_guard	改善 Aurora PostgreSQL 11.21、12.16、13.12、14.9、15.4 和較舊版本的記憶體管理功能，可防止因可用記憶體不足而導致的穩定性問題和資料庫重新啟動。如需詳細資訊，請參閱Aurora PostgreSQL 中的記憶體管理已改善。	False
rds.pg_stat_ramdisk_size	統計資料 Ramdisk 的大小 (以 MB 為單位) 。若為非零值將設定 Ramdisk。	0

參數名稱	描述	預設
rds.rds_superuser_reserved_connections	設定為 rds_superusers 保留的連線槽數。此參數僅適用於第 15 版和更舊版本。如需詳細資訊，請參閱 PostgreSQL 文件保留的連線。	2
rds.superuser_variables	僅限進階使用者變數清單，我們提升了 rds_superuser 修改陳述式。	session_replication_role
remove_temp_files_after_crash	後端當機後移除暫存檔案。	0
restart_after_crash	後端當機後重新初始化伺服器。	–
row_security	啟用資料列安全性。	–
search_path	針對不符合結構描述的名稱設定結構描述搜尋順序。	–
seq_page_cost	設定規劃器對循序擷取磁碟分頁的成本估算。	–
session_replication_role	設定觸發器和重寫規則的工作階段行為。	–
shared_buffers	(8kB) 設定伺服器所用的共用記憶體緩衝區數目。	SUM(DBInstanceClassMemory/12038,-50003)
shared_preload_libraries	列出要預先載入到伺服器的共用程式庫。	pg_stat_statements
ssl_ca_file	SSL 伺服器授權檔案的位置。	/rdsdbdata/rds-metadata/ca-cert.pem
ssl_cert_file	SSL 伺服器憑證檔案的位置。	/rdsdbdata/rds-metadata/server-cert.pem
ssl_crl_dir	SSL 憑證撤銷清單目錄的位置。	/rdsdbdata/rds-metadata/ssl_crl_dir/

參數名稱	描述	預設
ssl_key_file	SSL 伺服器私有金鑰檔案的位置	/rdsdbdata/rds-metadata/server-key.pem
standard_conforming_strings	導致 ... 字串逐字地處理反斜線。	–
statement_timeout	(毫秒) 設定任何陳述式允許的持續時間上限。	–
stats_temp_directory	將暫存統計數字檔案寫入指定目錄。	/rdsdbdata/db/pg_stat_tmp
superuser_reserved_connections	設定為進階使用者保留的連線槽數。	3
synchronize_seqscans	啟用已同步的循序掃描。	–
tcp_keepalives_count	TCP 保持連線重新傳輸的數量上限	–
tcp_keepalives_idle	(秒)發出 TCP 存留之間的時間。	–
tcp_keepalives_interval	(秒) TCP 存留重新傳輸之間的時間。	–
temp_buffers	(8kB) 設定每個工作階段所用的臨時緩衝區數上限。	–
temp_file_limit	限制指定 PostgreSQL 程序可用於暫存檔案的磁碟空間總量 (以 KB 為單位)，不包括用於明確暫存資料表的空間	-1
temp_tablespaces	設定要用於暫存資料表和排序檔案的資料表空間。	–
track_activities	收集有關執行命令的資訊。	–
track_activity_query_size	設定為 pg_stat_activity.current_query 保留的大小 (以位元組為單位)。	4096

參數名稱	描述	預設
track_counts	收集資料庫活動的統計資訊。	–
track_functions	收集資料庫活動的功能層級統計資訊。	pl
track_io_timing	收集資料庫輸入/輸出活動的計時統計數字。	1
transform_–_equals	將 expr=– 視為 IS –。	–
update_process_title	更新程序標題以顯示作用中的 SQL 命令。	–
wal_receiver_status_interval	(秒) 設定向主伺服器報告 WAL 接收器狀態之間的間隔上限。	–
work_mem	(kB) 設定要用於查詢工作空間的記憶體上限。	–
xmlbinary	設定有多少二進位值要在 XML 中編碼。	–
xmloption	設定要將明確剖析和序列化作業中的 XML 資料視為文件或內容片段。	–

Amazon Aurora PostgreSQL 等待事件

以下是 Aurora PostgreSQL 的常見等待事件。若要進一步了解等待事件和調整 Aurora PostgreSQL 資料庫叢集，請參閱 調校 Aurora PostgreSQL 的等待事件。

Activity:ArchiverMain

封存程序正在等待活動。

Activity:AutoVacuumMain

自動資料清理啟動器程序正在等待活動。

Activity:BgWriterHibernate

等待活動時，背景寫入器程序正在休眠。

Activity:BgWriterMain

背景寫入器程序正在等待活動。

Activity:CheckpointerMain

檢查點指標程序正在等待活動。

Activity:LogicalApplyMain

邏輯複寫套用程序正在等待活動。

Activity:LogicalLauncherMain

邏輯複寫啟動器程序正在等待活動。

Activity:PgStatMain

統計資料收集器程序正在等待活動。

Activity:RecoveryWalAll

程序正在等待復原時來自串流的預寫日誌 (WAL)。

Activity:RecoveryWalStream

啟動程序正在等待要在串流復原期間送達的預寫日誌 (WAL)。

Activity:SysLoggerMain

syslogger 程序正在等待活動。

Activity:WalReceiverMain

預寫日誌 (WAL) 接收器程序正在等待活動。

Activity:WalSenderMain

預寫日誌 (WAL) 傳送器程序正在等待活動。

Activity:WalWriterMain

預寫日誌 (WAL) 寫入器程序正在等待活動。

BufferPin:BufferPin

程序正在等待獲取緩衝區上的專用 Pin。

Client:GSSOpenServer

在建立一般安全服務應用程式界面 (GSSAPI) 工作階段時，程序正在等待從用戶端讀取資料。

Client:ClientRead

後端程序正在等待從 PostgreSQL 用戶端接收資料。如需詳細資訊，請參閱Client:ClientRead。

Client:ClientWrite

後端程序正在等待將更多資料傳送到 PostgreSQL 用戶端。如需詳細資訊，請參閱Client:ClientWrite。

Client:LibPQWalReceiverConnect

程序正在預寫日誌 (WAL) 接收器中等待，以建立遠端伺服器的連線。

Client:LibPQWalReceiverReceive

程序正在預寫日誌 (WAL) 接收器中等待，以從遠端伺服器接收資料。

Client:SSLOpenServer

程序在嘗試連線時等待 Secure Sockets Layer (SSL)。

Client:WalReceiverWaitStart

程序正在等待啟動程序傳送初始資料以進行串流複寫。

Client:WalSenderWaitForWAL

程序正在等待 WAL 傳送器程序中清空預寫日誌 (WAL)。

Client:WalSenderWriteData

在處理來自 WAL 傳送器程序中預寫日誌 (WAL) 接收器的回覆時，程序正在等待任何活動。

CPU

後端程序處於作用中或正在等待 CPU。如需詳細資訊，請參閱CPU。

Extension:extension

後端程序正在等待擴充功能或模組定義的條件。

IO:AuroraEnhancedLogicalWALRead

後端程序會從變更資料擷取 (CDC) 磁碟區擷取日誌記錄。

IO:AuroraOptimizedReadsCacheRead

程序正在等待從 Optimized Reads 階層式快取進行讀取，因為該頁面在共享記憶體中不可用。

IO:AuroraOptimizedReadsCacheSegmentTruncate

程序正在等待將 Optimized Reads 階層式快取區段檔案進行截斷。

IO:AuroraOptimizedReadsCacheWrite

背景寫入器程序正在等待寫入 Optimized Reads 階層式快取。

IO:AuroraStorageLogAllocate

工作階段正在配置中繼資料並準備交易日誌寫入。

IO:BufFileRead

當作業需要的記憶體超過工作記憶體參數所定義的數量時，引擎會在磁碟上建立暫存檔案。作業從暫存檔案讀取時，就會發生此等待事件。如需詳細資訊，請參閱IO:BufFileRead 和 IO:BufFileWrite。

IO:BufFileWrite

當作業需要的記憶體超過工作記憶體參數所定義的數量時，引擎會在磁碟上建立暫存檔案。當作業寫入至暫存檔案時，就會發生此等待事件。如需詳細資訊，請參閱IO:BufFileRead 和 IO:BufFileWrite。

IO:ControlFileRead

程序正在等待從 `pg_control` 檔案進行讀取。

IO:ControlFileSync

程序正在等待 `pg_control` 檔案達到持久的儲存。

IO:ControlFileSyncUpdate

程序正在等待 `pg_control` 檔案的更新以達到持久的儲存。

IO: 控制檔案寫入

程序正在等待寫入至 `pg_control` 檔案。

IO:ControlFileWriteUpdate

程序正在等待寫入以更新 `pg_control` 檔案。

IO:CopyFileRead

程序正在檔案複製作業期間等待讀取。

IO:CopyFileWrite

程序正在檔案複製作業期間等待寫入。

IO:DataFileExtend

程序正在等待關聯資料檔案擴充。

IO:DataFileFlush

程序正在等待關聯資料檔案達到持久的儲存。

IO:DataFileImmediateSync

程序正在等待關聯資料檔案立即同步到持久的儲存。

IO:DataFilePrefetch

程序正在等待從關聯資料檔案進行非同步預先擷取。

IO:DataFileSync

程序正在等待關聯資料檔案的變更，以達到持久的儲存。

IO:DataFileRead

後端程序已嘗試在共用緩衝區中尋找一個頁面，但沒有找到它，所以從儲存中讀取它。如需詳細資訊，請參閱IO:DataFileRead。

IO:DataFileTruncate

程序正在等待關聯資料檔案進行截斷。

IO:DataFileWrite

程序正在等待寫入至關聯資料檔案。

IO:DSMFillZeroWrite

程序正在等待將零位元組寫入至動態共用記憶體備份檔案。

IO:LockFileAddToDataDirRead

在將一行新增至資料目錄鎖定檔案時，程序正在等待讀取。

IO:LockFileAddToDataDirSync

在將一行新增到資料目錄鎖定檔案時，程序正在等待資料達到持久的儲存。

IO:LockFileAddToDataDirWrite

在將一行新增至資料目錄鎖定檔案時，程序正在等待寫入。

IO:LockFileCreateRead

建立資料目錄鎖定檔案時，程序正在等待讀取。

IO:LockFileCreateSync

在建立資料目錄鎖定檔案時，程序正在等待資料達到持久的儲存。

IO:LockFileCreateWrite

在建立資料目錄鎖定檔案時，程序正在等待寫入。

IO:LockFileReCheckDataDirRead

在重新檢查資料目錄鎖定檔案期間，程序正在等待讀取。

IO:LogicalRewriteCheckpointSync

在檢查點期間，程序正在等待邏輯重寫映射達到持久的儲存。

IO:LogicalRewriteMappingSync

在邏輯重寫期間，程序正在等待映射資料達到持久的儲存。

IO:LogicalRewriteMappingWrite

在邏輯重寫期間，程序正在等待映射資料的寫入。

IO:LogicalRewriteSync

程序正在等待邏輯重寫映射達到持久的儲存。

IO:LogicalRewriteTruncate

在邏輯重寫期間，程序正在等待映射資料的截斷。

IO:LogicalRewriteWrite

程序正在等待邏輯重寫映射的寫入。

IO:RelationMapRead

程序正在等待關聯映射檔案的讀取。

IO:RelationMapSync

程序正在等待關聯映射檔案達到持久的儲存。

IO:RelationMapWrite

程序正在等待寫入至關聯映射檔案。

IO:ReorderBufferRead

在重新排序緩衝區管理期間，程序正在等待讀取。

IO:ReorderBufferWrite

在重新排序緩衝區管理期間，程序正在等待寫入。

IO:ReorderLogicalMappingRead

在重新排序緩衝區管理期間，程序正在等待邏輯映射的讀取。

IO:ReplicationSlotRead

程序正在等待從複寫插槽控制檔案讀取。

IO:ReplicationSlotRestoreSync

在將複寫插槽控制檔案還原到記憶體時，程序正在等待其達到持久的儲存。

IO:ReplicationSlotSync

程序正在等待複寫插槽控制檔案達到持久的儲存。

IO:ReplicationSlotWrite

程序正在等待寫入至複寫插槽控制檔案。

IO:SLRUFlushSync

在檢查點或資料庫關閉期間，程序正在等待簡單的最近最少使用 (SLRU) 資料達到持久的儲存。

IO:SLRURead

程序正在等待讀取簡單的最近最少使用 (SLRU) 頁面。

IO:SLRUSync

程序正在等待簡單的最近最少使用 (SLRU) 資料，在頁面寫入後達到持久的儲存。

IO:SLRUWrite

程序正在等待寫入簡單的最近最少使用 (SLRU) 頁面。

IO:SnapbuildRead

程序正在等待讀取序列化的歷史目錄快照。

IO:SnapbuildSync

程序正在等待序列化的歷史目錄快照達到持久的儲存。

IO:SnapbuildWrite

程序正在等待寫入序列化的歷史目錄快照。

IO:TimelineHistoryFileSync

程序正在等待透過串流複寫接收的時間軸歷史記錄檔案達到持久的儲存。

IO:TimelineHistoryFileWrite

程序正在等待寫入透過串流複寫接收的時間軸歷史記錄檔案。

IO:TimelineHistoryRead

程序正在等待讀取時間軸歷史記錄檔案。

IO:TimelineHistorySync

程序正在等待新建立的時間軸歷史記錄檔案達到持久的儲存。

IO:TimelineHistoryWrite

程序正在等待寫入新建立的時間軸歷史記錄檔案。

IO:TwophaseFileRead

程序正在等待讀取兩個階段狀態檔案。

IO:TwophaseFileSync

程序正在等待兩個階段狀態檔案達到持久的儲存。

IO:TwophaseFileWrite

程序正在等待寫入兩個階段狀態檔案。

IO:WALBootstrapSync

在自舉期間，程序正在等待預寫日誌 (WAL) 達到持久的儲存。

IO:WALBootstrapWrite

在自舉期間，程序正在等待寫入預寫日誌 (WAL) 頁面。

IO:WALCopyRead

複製現有的區段建立新的預寫日誌 (WAL) 區段時，程序正在等待讀取。

IO:WALCopySync

程序正在等待藉由複製現有區段所建立的新預寫日誌 (WAL) 區段達到持久的儲存。

IO:WALCopyWrite

複製現有的區段建立新的預寫日誌 (WAL) 區段時，程序正在等待寫入。

IO:WALInitSync

程序正在等待新初始化的預寫日誌 (WAL) 檔案達到持久的儲存。

IO:WALInitWrite

初始化新的預寫日誌 (WAL) 檔案時，程序正在等待寫入。

IO:WALRead

程序正在等待從預寫日誌 (WAL) 檔案讀取。

IO:WALSenderTimelineHistoryRead

在 WAL 傳送器時間軸命令期間，程序正在等待從時間軸歷史記錄檔案讀取。

IO:WALSync

程序正在等待預寫日誌 (WAL) 檔案達到持久的儲存。

IO:WALSyncMethodAssign

在指派新的預寫日誌 (WAL) 同步方法時，程序正在等待資料達到持久的儲存。

IO:WALWrite

程序正在等待寫入至預寫日誌 (WAL) 檔案。

IO:XactSync

後端程序正在等待 Aurora 儲存子系統確認一般交易的遞交，或是預備交易的遞交或回復。如需詳細資訊，請參閱IO:XactSync。

IPC:AuroraLogicalSchemaUpdate

兩個後端程序正嘗試將相同的項目插入結構描述快取。一個程序會繼續，而另一個程序會等待其完成。

IPC:AuroraOptimizedReadsCacheWriteStop

程序正等待背景寫入器停止寫入最佳化讀取階層式快取。

IPC:BackupWaitWalArchive

程序正在等待備份所需的預寫日誌 (WAL) 檔案成功封存。

IPC:BgWorkerShutdown

程序正在等待背景工作者關閉。

IPC:BgWorkerStartup

程序正在等待背景工作者啟動。

IPC: 建立頁面

程序正在等待繼續平行 B 型樹狀目錄掃描所需的頁碼變成可用。

IPC:CheckpointDone

程序正在等待檢查點完成。

IPC:CheckpointStart

程序正在等待檢查點啟動。

IPC:ClogGroupUpdate

程序正在等待群組領導者在交易結束時更新交易狀態。

IPC:DamRecordTxAck

後端程序已產生資料庫活動串流事件，而且正在等待事件變成持久的。如需詳細資訊，請參閱IPC:DamRecordTxAck。

IPC:ExecuteGather

在執行蒐集計劃節點時，程序正在等待來自子程序的活動。

IPC:Hash/Batch/Allocating

程序正在等待選定的平行雜湊參與者來配置雜湊表。

IPC:Hash/Batch/Electing

程序正在選擇平行雜湊參與者來配置雜湊表。

IPC:Hash/Batch/Loading

程序正在等待其他平行雜湊參與者完成載入雜湊表。

IPC:Hash/Build/Allocating

程序正在等待選定的平行雜湊參與者來配置初始雜湊表。

IPC:Hash/Build/Electing

程序正在選擇平行雜湊參與者來配置初始雜湊表。

IPC:Hash/Build/HashingInner

程序正在等待其他平行雜湊參與者完成內部關聯的雜湊。

IPC:Hash/Build/HashingOuter

程序正在等待其他平行雜湊參與者完成外部關聯的分割。

IPC:Hash/GrowBatches/Allocating

程序正在等待選定的平行雜湊參與者配置更多批次。

IPC:Hash/GrowBatches/Deciding

程序正在選擇平行雜湊參與者來決定未來的批次增長。

IPC:Hash/GrowBatches/Electing

程序正在選擇平行雜湊參與者來配置更多批次。

IPC:Hash/GrowBatches/Finishing

程序正在等待選定的平行雜湊參與者來決定未來的批次增長。

IPC:Hash/GrowBatches/Repartitioning

程序正在等待其他平行雜湊參與者完成重新分割。

IPC:Hash/GrowBuckets/Allocating

程序正在等待選定的平行雜湊參與者完成配置更多的儲存貯體。

IPC:Hash/GrowBuckets/Electing

程序正在選擇一個平行雜湊參與者來配置更多的儲存貯體。

IPC:Hash/GrowBuckets/Reinserting

程序正在等待其他平行雜湊參與者完成將元組插入到新的儲存貯體。

IPC:HashBatchAllocate

程序正在等待選定的平行雜湊參與者來配置雜湊表。

IPC:HashBatchElect

程序正在等待選擇平行雜湊參與者來配置雜湊表。

IPC:HashBatchLoad

程序正在等待其他平行雜湊參與者完成載入雜湊表。

IPC:HashBuildAllocate

程序正在等待選定的平行雜湊參與者來配置初始雜湊表。

IPC:HashBuildElect

程序正在等待選擇平行雜湊參與者來配置初始雜湊表。

IPC:HashBuildHashInner

程序正在等待其他平行雜湊參與者完成內部關聯的雜湊。

IPC:'HashBuildHashOuter

程序正在等待其他平行雜湊參與者完成外部關聯的分割。

IPC:HashGrowBatchesAllocate

程序正在等待選定的平行雜湊參與者配置更多批次。

IPC:'HashGrowBatchesDecide

程序正在等待選擇平行雜湊參與者來決定未來的批次增長。

IPC:HashGrowBatchesElect

程序正在等待選擇平行雜湊參與者來配置更多批次。

IPC:HashGrowBatchesFinish

程序正在等待選定的平行雜湊參與者來決定未來的批次增長。

IPC:HashGrowBatchesRepartition

程序正在等待其他平行雜湊參與者完成重新分割。

IPC:HashGrowBucketsAllocate

程序正在等待選定的平行雜湊參與者完成配置更多的儲存貯體。

IPC:HashGrowBucketsElect

程序正在等待選擇平行雜湊參與者來配置更多的儲存貯體。

IPC:HashGrowBucketsReinsert

程序正在等待其他平行雜湊參與者完成將元組插入到新的儲存貯體。

IPC:LogicalSyncData

程序正在等待邏輯複寫遠端伺服器傳送資料，以進行初始資料表同步。

IPC:LogicalSyncStateChange

程序正在等待邏輯複寫遠端伺服器變更狀態。

IPC:MessageQueueInternal

程序正在等待另一個程序附加到共用訊息佇列。

IPC:MessageQueuePutMessage

程序正在等待將通訊協定訊息寫入至共用訊息佇列。

IPC:MessageQueueReceive

程序正在等待從共用訊息佇列接收位元組。

IPC:MessageQueueSend

程序正在等待將位元組傳送至共用訊息佇列。

IPC:ParallelBitmapScan

程序正在等待平行點陣圖掃描初始化。

IPC:ParallelCreateIndexScan

程序正在等待 CREATE INDEX 工作者完成堆積掃描。

IPC:ParallelFinish

程序正在等待平行工作者完成運算。

IPC:ProcArrayGroupUpdate

程序正等待群組領導者在交易結束時清除交易 ID。

IPC:ProcSignalBarrier

程序正在等待所有後端處理屏障事件。

IPC:Promote

程序正在等待待命提升。

IPC:RecoveryConflictSnapshot

程序正在等待解決復原衝突以進行清理清除。

IPC:RecoveryConflictTablespace

程序正在等待解決復原衝突，以捨棄資料表空間。

IPC:RecoveryPause

程序正在等待復原繼續進行。

IPC:ReplicationOriginDrop

程序正在等待複寫來源變成非作用中，以便可以將其捨棄。

IPC:ReplicationSlotDrop

程序正在等待複寫插槽變成非作用中，以便可以將其捨棄。

IPC:SafeSnapshot

程序正在等待取得 READ ONLY DEFERRABLE 交易的有效快照。

IPC:SyncRep

在同步複寫期間，程序正在等待遠端伺服器的確認。

IPC:XactGroupUpdate

程序正等待群組領導者在交易結束時更新交易狀態。

Lock:advisory

後端程序已請求建議鎖定，並正在等待它。如需詳細資訊，請參閱Lock:advisory。

Lock:extend

後端程序正在等待釋放鎖定，以便它可以延伸關聯。此鎖定是必要的，因為一次只有一個後端程序可以延伸關聯。如需詳細資訊，請參閱Lock:extend。

Lock:frozenid

程序正在等待更新 `pg_database.datfrozenxid` 和 `pg_database.datminmxid`。

Lock:object

程序正在等待對非關聯式資料庫物件取得鎖定。

Lock:page

程序正在等待對關聯的頁面取得鎖定。

Lock:Relation

後端程序正在等待對另一個交易所鎖定的關聯取得鎖定。如需詳細資訊，請參閱Lock:Relation。

Lock:spectoken

程序正在等待取得推測性插入鎖定。

Lock:speculative token

程序正在等待取得推測性插入鎖定。

Lock:transactionid

交易正在等待資料列層級鎖定。如需詳細資訊，請參閱Lock:transactionid。

Lock:tuple

在另一個後端程序對元組保有衝突鎖定時，後端程序正在等待對相同的元組取得鎖定。如需詳細資訊，請參閱Lock:tuple。

Lock:userlock

程序正在等待取得使用者鎖定。

Lock:virtualxid

程序正在等待取得虛擬交易 ID 鎖定。

LWLock:AddinShmemInit

程序正在等待管理共用記憶體中擴充功能的空間配置。

LWLock:AddinShmemInitLock

程序正在等待管理共用記憶體中的空間配置。

LWLock:async

程序正在等待非同步 (通知) 緩衝區上的輸入/輸出。

LWLock:AsyncCtlLock

程序正在等待讀取或更新共用通知狀態。

LWLock:AsyncQueueLock

程序正在等待讀取或更新通知訊息。

LWLock:AuroraOptimizedReadsCacheMapping

程序正在等待將資料區塊與 Optimized Reads 階層式快取中的頁面建立關聯。

LWLock:AutoFile

程序正在等待更新 `postgresql.auto.conf` 檔案。

LWLock:AutoFileLock

程序正在等待更新 `postgresql.auto.conf` 檔案。

LWLock:Autovacuum

程序正在等待讀取或更新自動資料清理工作者的目前狀態。

LWLock:AutovacuumLock

自動資料清理工作者或啟動器正在等待更新或讀取自動資料清理工作者的目前狀態。

LWLock:AutovacuumSchedule

程序正在等待確保為了自動資料清理而選取的資料表仍然需要清理。

LWLock:AutovacuumScheduleLock

程序正在等待確保其為了清理而選取的資料表仍然需要清理。

LWLock:BackendRandomLock

程序正在等待產生亂數。

LWLock:BackgroundWorker

程序正在等待讀取或更新背景工作者狀態。

LWLock:BackgroundWorkerLock

程序正在等待讀取或更新背景工作者狀態。

LWLock:BtreeVacuum

程序正在等待讀取或更新 B 型樹狀結構索引的清理相關資訊。

LWLock:BtreeVacuumLock

程序正在等待讀取或更新 B 型樹狀結構索引的清理相關資訊。

LWLock:buffer_content

後端程序正在等待對共用記憶體緩衝區的內容取得輕量型鎖定。如需詳細資訊，請參閱LWLock:buffer_content (BufferContent)。

LWLock:buffer_mapping

後端程序正在等待將資料區塊與共用緩衝集區中的緩衝區建立關聯。如需詳細資訊，請參閱LWLock:buffer_mapping。

LWLock:BufferIO

後端程序想要將頁面讀入共用記憶體中。程序正在等待其他程序完成頁面的輸入/輸出。如需詳細資訊，請參閱LWLock:BufferIO (IPC:BufferIO)。

LWLock:Checkpoint

程序正在等待開始檢查點。

LWLock:CheckpointLock

程序正在等待執行檢查點。

LWLock:CheckpointerComm

程序正在等待管理 `fsync` 請求。

LWLock:CheckpointerCommLock

程序正在等待管理 `fsync` 請求。

LWLock:clog

程序正在等待 Clog (交易狀態) 緩衝區上的輸入/輸出。

LWLock:CLogControlLock

程序正在等待讀取或更新交易狀態。

LWLock:CLogTruncationLock

程序正在等待執行 `txid_status` 或更新其可用的最舊交易 ID。

LWLock:commit_timestamp

程序正在等待遞交時間戳記緩衝區上的輸入/輸出。

LWLock:CommitTs

程序正在等待讀取或更新針對交易遞交時間戳記設定的最後一個值。

LWLock:CommitTsBuffer

程序正在等待簡單的最近最少使用 (SLRU) 緩衝區上的輸入/輸出，以取得遞交時間戳記。

LWLock:CommitTsControlLock

程序正在等待讀取或更新交易遞交時間戳記。

LWLock:CommitTsLock

程序正在等待讀取或更新針對交易時間戳記設定的最後一個值。

LWLock:CommitTsSLRU

程序正在等待存取簡單的最近最少使用 (SLRU) 快取，以取得遞交時間戳記。

LWLock:ControlFile

程序正在等待讀取或更新 `pg_control` 檔案，或建立新的預寫日誌 (WAL) 檔案。

LWLock:ControlFileLock

程序正在等待讀取或更新控制檔案，或建立新的預寫日誌 (WAL) 檔案。

LWLock:DynamicSharedMemoryControl

程序正在等待讀取或更新動態共用記憶體配置資訊。

LWLock:DynamicSharedMemoryControlLock

程序正在等待讀取或更新動態共用記憶體狀態。

LWLock:lock_manager

後端程序正在等待新增或檢查後端程序的鎖定。或者它正在等待加入或退出平行查詢所使用的鎖定群組。如需詳細資訊，請參閱LWLock:lock_manager。

LWLock:LockFastPath

程序正在等待讀取或更新程序的快速路徑鎖定資訊。

LWLock:LogicalRepWorker

程序正在等待讀取或更新邏輯複寫工作者的狀態。

LWLock:LogicalRepWorkerLock

程序正在等待邏輯複寫工作者上的動作完成。

LWLock:LogicalSchemaCache

程序已修改結構描述快取。

LWLock:multixact_member

程序正在等待 multixact_member 緩衝區上的輸入/輸出。

LWLock:multixact_offset

程序正在等待 multixact 位移緩衝區上的輸入/輸出。

LWLock:MultiXactGen

程序正在等待讀取或更新共用 multixact 狀態。

LWLock:MultiXactGenLock

程序正在等待讀取或更新共用 multixact 狀態。

LWLock:MultiXactMemberBuffer

程序正在等待簡單的最近最少使用 (SLRU) 緩衝區上的輸入/輸出，以取得 multixact 成員。如需詳細資訊，請參閱LWLock:MultiXact。

LWLock:MultiXactMemberControlLock

程序正在等待讀取或更新 multixact 成員映射。

LWLock:MultiXactMemberSLRU

程序正在等待存取簡單的最近最少使用 (SLRU) 快取，以取得 multixact 成員。如需詳細資訊，請參閱LWLock:MultiXact。

LWLock:MultiXactOffsetBuffer

程序正在等待簡單的最近最少使用 (SLRU) 緩衝區上的輸入/輸出，以取得 multixact 位移。如需詳細資訊，請參閱LWLock:MultiXact。

LWLock:MultiXactOffsetControlLock

程序正在等待讀取或更新 multixact 位移映射。

LWLock:MultiXactOffsetSLRU

程序正在等待存取簡單的最近最少使用 (SLRU) 快取，以取得 multixact 位移。如需詳細資訊，請參閱LWLock:MultiXact。

LWLock:MultiXactTruncation

程序正在等待讀取或截斷 multixact 資訊。

LWLock:MultiXactTruncationLock

程序正在等待讀取或截斷 multixact 資訊。

LWLock:NotifyBuffer

程序正在等待簡單的最近最少使用 (SLRU) 緩衝區上的輸入/輸出，以取得 NOTIFY 通知。

LWLock:NotifyQueue

程序正在等待讀取或更新 NOTIFY 訊息。

LWLock:NotifyQueueTail

程序正在等待更新對 NOTIFY 訊息儲存的限制。

LWLock:NotifyQueueTailLock

程序正在等待更新對通知訊息儲存的限制。

LWLock:NotifySLRU

程序正在等待存取簡單的最近最少使用 (SLRU) 快取，以取得 NOTIFY 訊息。

LWLock:OidGen

程序正在等待配置新的物件 ID (OID)。

LWLock:OidGenLock

程序正在等待配置或指派物件 ID (OID)。

LWLock:oldserxid

程序正在等待 oldserxid 緩衝區上的輸入/輸出。

LWLock:OldSerXidLock

程序正在等待讀取或記錄衝突的序列化交易。

LWLock:OldSnapshotTimeMap

程序正在等待讀取或更新舊的快照控制資訊。

LWLock:OldSnapshotTimeMapLock

程序正在等待讀取或更新舊的快照控制資訊。

LWLock:parallel_append

在平行附加計劃執行期間，程序正在等待選擇下一個子計劃。

LWLock:parallel_hash_join

在平行雜湊計劃執行期間，程序正在等待配置或交換記憶體區塊或更新計數器。

LWLock:parallel_query_dsa

程序正在等待鎖定動態共用記憶體配置，以進行平行查詢。

LWLock:ParallelAppend

在平行附加計劃執行期間，程序正在等待選擇下一個子計劃。

LWLock:ParallelHashJoin

在平行雜湊聯結的計劃執行期間，程序正在等待同步工作者。

Lwlock:ParallelQueryDSA

程序正在等待動態共用記憶體配置，以進行平行查詢。

Lwlock:PerSessionDSA

程序正在等待動態共用記憶體配置，以進行平行查詢。

Lwlock:PerSessionRecordType

程序正在等待存取有關複合類型的平行查詢資訊。

Lwlock:PerSessionRecordTypmod

程序正在等待存取有關識別匿名記錄類型之類型修飾詞的平行查詢資訊。

Lwlock:PerXactPredicateList

程序正在等待存取目前可序列化交易在平行查詢期間保留的述詞鎖定清單。

Lwlock:predicate_lock_manager

程序正在等待新增或檢查述詞鎖定資訊。

Lwlock:PredicateLockManager

程序正在等待存取可序列化交易所使用的述詞鎖定資訊。

Lwlock:proc

程序正在等待讀取或更新快速路徑鎖定資訊。

LWLock:ProcArray

程序正在等待存取共用的每個程序資料結構 (通常，用來取得快照或報告工作階段的交易 ID)。

LWLock:ProcArrayLock

程序正在等待取得快照或在交易結束時清除交易 ID。

LWLock:RelationMapping

程序正在等待讀取或更新 `pg_filenode.map` 檔案 (用來追蹤特定系統目錄的檔案節點指派)。

LWLock:RelationMappingLock

程序正在等待更新用來存放目錄至檔案節點映射的關聯映射檔案。

LWLock:RelCacheInit

程序正在等待讀取或更新 `pg_internal.init` 檔案 (關聯快取初始化檔案)。

LWLock:RelCacheInitLock

程序正在等待讀取或寫入關聯快取初始化檔案。

LWLock:replication_origin

程序正在等待讀取或更新複寫進度。

LWLock:replication_slot_io

程序正在等待複寫插槽上的輸入/輸出。

LWLock:ReplicationOrigin

程序正在等待建立、捨棄或使用複寫來源。

LWLock:ReplicationOriginLock

程序正在等待設定、捨棄或使用複寫來源。

LWLock:ReplicationOriginState

程序正在等待讀取或更新某個複寫來源的進度。

LWLock:ReplicationSlotAllocation

程序正在等待配置或釋出複寫插槽。

LWLock:ReplicationSlotAllocationLock

程序正在等待配置或釋出複寫插槽。

LWLock:ReplicationSlotControl

程序正在等待讀取或更新複寫插槽狀態。

LWLock:ReplicationSlotControlLock

程序正在等待讀取或更新複寫插槽狀態。

LWLock:ReplicationSlotIO

程序正在等待複寫插槽上的輸入/輸出。

LWLock:SerialBuffer

程序正在等待簡單的最近最少使用 (SLRU) 緩衝區上的輸入/輸出，以取得可序列化的交易衝突。

LWLock:SerializableFinishedList

程序正在等待存取已完成的可序列化交易清單。

LWLock:SerializableFinishedListLock

程序正在等待存取已完成的可序列化交易清單。

LWLock:SerializablePredicateList

程序正在等待存取可序列化交易所保留的述詞鎖定清單。

LWLock:SerializablePredicateLockListLock

程序正在等待對可列化交易所保留的鎖定清單執行作業。

LWLock:SerializableXactHash

程序正在等待讀取或更新可序列化交易的相關資訊。

LWLock:SerializableXactHashLock

程序正在等待擷取或存放可序列化交易的相關資訊。

LWLock:SerialSLRU

程序正在等待存取簡單的最近最少使用 (SLRU) 快取，以取得可序列化的交易衝突。

LWLock:SharedTidBitmap

在平行點陣圖索引掃描期間，程序正在等待存取共用的元組識別符 (TID) 點陣圖。

LWLock:SharedTupleStore

程序正在等待在平行查詢期間存取共用的元組存放區。

LWLock:ShmemIndex

程序正在等待尋找或配置共用記憶體中的空間。

LWLock:ShmemIndexLock

程序正在等待尋找或配置共用記憶體中的空間。

LWLock:SInvalRead

程序正在等待從共用目錄失效佇列中擷取訊息。

LWLock:SInvalReadLock

程序正在等待從共用失效佇列中擷取或移除訊息。

LWLock:SInvalWrite

程序正在等待將訊息新增至共用目錄失效佇列。

LWLock:SInvalWriteLock

程序正在等待在共用失效佇列中新增訊息。

LWLock:SyncRep

程序正在等待讀取或更新同步複寫狀態的相關資訊。

LWLock:SyncRepLock

程序正在等待讀取或更新同步複本的相關資訊。

LWLock:SyncScan

程序正在等待選取同步表格掃描的起始位置。

LWLock:SyncScanLock

程序正在等待取得表格上掃描的起始位置，以進行同步掃描。

LWLock:TablespaceCreate

程序正在等待建立或捨棄資料表空間。

LWLock:TablespaceCreateLock

程序正在等待建立或捨棄資料表空間。

LWLock:tbm

程序正在等待樹狀點陣圖 (TBM) 上的共用疊代鎖定。

LWLock:TwoPhaseState

程序正在等待讀取或更新預備交易的狀態。

LWLock:TwoPhaseStateLock

程序正在等待讀取或更新預備交易的狀態。

LWLock:wal_insert

程序正在等待將預寫日誌 (WAL) 插入到記憶體緩衝區。

LWLock:WALBufMapping

程序正在等待取代預寫日誌 (WAL) 緩衝區中的頁面。

LWLock:WALBufMappingLock

程序正在等待取代預寫日誌 (WAL) 緩衝區中的頁面。

LWLock:WALInsert

程序正在等待將預寫日誌 (WAL) 資料插入到記憶體緩衝區。

LWLock:WALWrite

程序正在等待預寫日誌 (WAL) 緩衝區寫入至磁碟。

LWLock:WALWriteLock

程序正在等待預寫日誌 (WAL) 緩衝區寫入至磁碟。

LWLock:WrapLimitsVacuum

程序正在等待更新對交易 ID 和 multixact 耗用的限制。

LWLock:WrapLimitsVacuumLock

程序正在等待更新對交易 ID 和 multixact 耗用的限制。

LWLock:XactBuffer

程序正在等待簡單的最近最少使用 (SLRU) 緩衝區上的輸入/輸出，以取得交易狀態。

LWLock:XactSLRU

程序正在等待存取簡單的最近最少使用 (SLRU) 快取，以取得交易狀態。

LWLock:XactTruncation

程序正在等待執行 pg_xact_status 或更新其可用的最舊交易 ID。

LWLock:XidGen

程序正在等待配置新的交易 ID。

LWLock:XidGenLock

程序正在等待配置或指派交易 ID。

Timeout:BaseBackupThrottle

在基礎備份期間，若調節活動，程序即會等待。

Timeout:PgSleep

後端程序已呼叫 pg_sleep 函數，並且正在等待睡眠逾時到期。如需詳細資訊，請參閱Timeout:PgSleep。

Timeout:RecoveryApplyDelay

由於延遲設定，程序正在等待在復原期間套用預寫日誌 (WAL)。

Timeout:RecoveryRetrieveRetryInterval

當預寫日誌 (WAL) 資料無法從任何來源 (pg_wal、封存或串流) 取得時，程序會在復原期間等待。

Timeout:VacuumDelay

程序正在以成本型清理延遲點等待。

如需 PostgreSQL 等待事件的完整清單，請參閱 PostgreSQL 文件中的統計數字收集器 > 等待事件表。

Amazon Aurora PostgreSQL 的資料庫引擎更新

以下提供 Amazon Aurora PostgreSQL 引擎版本和更新版本的相關資訊。另也提供有關如何升級 Amazon Aurora PostgreSQL 引擎的資訊。如需 Aurora 版本的整體詳細資訊，請參閱 Amazon Aurora 版本。

> **Tip**
>
> 您可以使用藍/綠部署，將資料庫叢集升級所需的停機時間降至最低。如需詳細資訊，請參閱使用藍/綠部署進行資料庫更新。

主題

- 識別 Amazon Aurora PostgreSQL 版本
- Amazon Aurora PostgreSQL 版本和引擎版本
- Amazon Aurora PostgreSQL 的擴充功能版本
- 升級 Amazon Aurora PostgreSQL 資料庫叢集
- 使用 Aurora PostgreSQL 長期支援 (LTS) 版本

識別 Amazon Aurora PostgreSQL 版本

Amazon Aurora 中包含某些 Aurora 通用的功能，且適用於所有 Aurora 資料庫叢集。Aurora 中亦包含其他專屬於 Aurora 所支援特定資料庫引擎的功能。這些功能僅適用於使用該資料庫引擎的 Aurora 資料庫叢集，例如 Aurora PostgreSQL。

Aurora 資料庫一般有兩個版本編號，即資料庫引擎版本編號和 Aurora 版本編號。如果 Aurora PostgreSQL 版本有 Aurora 版本編號，會包含在 Amazon Aurora PostgreSQL 版本和引擎版本 清單中的引擎編號之後。

主題

- Aurora 版本編號
- PostgreSQL 引擎版本編號

Aurora 版本編號

Aurora 版本編號使用「##.##.####」命名規則。Aurora 修補程式版本中包含發行後新增到次要版本的重要錯誤修正。若要進一步了解 Amazon Aurora 主要、次要和修補程式版本，請參閱 Amazon Aurora 主要版本、Amazon Aurora 次要版本 以及 Amazon Aurora 修補程式版本。

您可以使用以下 SQL 查詢找到 Aurora PostgreSQL 資料庫執行個體的 Aurora 版本編號：

```
postgres=> SELECT aurora_version();
```

從發行 PostgreSQL 版本 13.3、12.8、11.13、10.18 及所有其他更新的版本開始，Aurora 版本編號更接近 PostgreSQL 引擎版本。例如，查詢 Aurora PostgreSQL 13.3 資料庫叢集會傳回以下內容：

```
aurora_version
----------------
 13.3.1
(1 row)
```

先前的版本 (如 Aurora PostgreSQL 10.14 資料庫叢集) 傳回的版本編號則類似以下內容：

```
aurora_version
----------------
 2.7.3
(1 row)
```

PostgreSQL 引擎版本編號

從 PostgreSQL 10 開始，PostgreSQL 資料庫引擎版本的所有版本都使用「##.##」編號規則。幾個範例包括 PostgreSQL 10.18、PostgreSQL 12.7 和 PostgreSQL 13.3。

PostgreSQL 10 之前的版本使用「##、##、##」編號規則，其中前兩個數字構成主要版本編號，第三個數字表示次要版本。例如，PostgreSQL 9.6 是主要版本，其次要版本 9.6.21 或 9.6.22 由第三個數字表示。

> **Note**
>
> 不再支援 PostgreSQL 引擎 9.6 版。若要升級，請參閱升級 Amazon Aurora PostgreSQL 資料庫叢集。如需版本政策和發行時間表，請參閱 Amazon Aurora 主要版本可用的時間會維持多久。

您可以透過以下 SQL 查詢找出 PostgreSQL 資料庫引擎版本編號：

```
postgres=> SELECT version();
```

Aurora PostgreSQL 13.3 資料庫叢集的結果如下：

```
version
-----------------------------------------------------------------------------------------------------
 PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC)
 7.4.0, 64-bit
(1 row)
```

Amazon Aurora PostgreSQL 版本和引擎版本

Amazon Aurora PostgreSQL 相容版本會定期更新。更新會在系統維護時段套用到 Aurora PostgreSQL 資料庫叢集。套用更新時，取決於資料庫叢集的更新類型 AWS 區域、 和維護時段設定。列出的許多版本都包含 PostgreSQL 版本編號和 Amazon Aurora 版本編號。不過，從發行 PostgreSQL 版本 13.3、12.8、11.13、10.18 及所有其他更新的版本開始，就不使用 Aurora 版本編號。若要判斷 Aurora PostgreSQL 資料庫的版本編號，請參閱 識別 Amazon Aurora PostgreSQL 版本。

如需關於擴充功能和模組的資訊，請參閱Amazon Aurora PostgreSQL 的擴充功能版本。

> **Note**
>
> 如需 Amazon Aurora 的版本政策和版本時間表的詳細資訊，請參閱 Amazon Aurora 主要版本可用的時間會維持多久。

如需 Amazon Aurora 支援的詳細資訊，請參閱 Amazon RDS 常見問答集。

若要判斷 中有哪些 Aurora PostgreSQL 資料庫引擎版本可用 AWS 區域，請使用 describe-db-engine-versions AWS CLI 命令。例如：

```
aws rds describe-db-engine-versions --engine aurora-postgresql --query '*[].
[EngineVersion]' --output text --region aws-region
```

如需 的清單 AWS 區域，請參閱 Aurora PostgreSQL 區域可用性。

如需有關在 Aurora PostgreSQL 上可用之 PostgreSQL 版本的詳細內容，請參閱 Aurora PostgreSQL 版本備註。

Amazon Aurora PostgreSQL 的擴充功能版本

您可以安裝和設定各種 PostgreSQL 擴充功能，以便搭配使用 Aurora PostgreSQL 資料庫叢集。例如，您可以使用 PostgreSQL `pg_partman` 擴充功能，以自動化資料表分割區的建立和維護。如需進一步了解有關此及其他可用於 Aurora PostgreSQL 的擴充功能的詳細資訊，請參閱 使用擴充功能和外部資料包裝函式。

如需有關 Aurora PostgreSQL 支援的 PostgreSQL 擴充功能的詳細資訊，請參閱 Aurora PostgreSQL 版本備註中的 Amazon Aurora PostgreSQL 擴充功能版本。

升級 Amazon Aurora PostgreSQL 資料庫叢集

Amazon Aurora 使新版 PostgreSQL 資料庫引擎只有在經過廣泛的測試後，才可供用於 AWS 區域。當您所在區域有新版可供使用時，您可以將 Aurora PostgreSQL 資料庫叢集升級到新版本。

升級到新版本可能是次要升級或主要升級，這視資料庫叢集目前正在執行的 Aurora PostgreSQL 版本而定。例如，將 Aurora PostgreSQL 11.15 資料庫叢集升級至 Aurora PostgreSQL 13.6 是主要版本升級。將 Aurora PostgreSQL 13.3 資料庫叢集升級至 Aurora PostgreSQL 13.7 是次要版本升級。在以下主題中，您可以找到如何執行這兩種升級的相關資訊。

內容

- Aurora PostgreSQL 升級程序概觀
- 取得 中可用版本的清單 AWS 區域
- 執行主要版本升級
  - 測試執行生產資料庫叢集升級到新主要版本的程序
  - 升級後建議
  - 將 Aurora PostgreSQL 引擎升級為新的主要版本
    - 全域資料庫的主要升級
- 執行次要版本升級
  - 執行次要版本升級之前
  - 如何執行次要版本升級和套用修補程式
  - 次要版本升級和零停機時間修補
  - 零停機時間修補的限制
  - 將 Aurora PostgreSQL 引擎升級為新的次要版本
- 升級 PostgreSQL 延伸

- 替代藍/綠升級技術

Aurora PostgreSQL 升級程序概觀

主要和次要版本升級之間的差異如下：

次要版本升級和修補程式

次要版本升級和修補程式只包含與現有應用程式回溯相容的變更。次要版本升級和修補程式只有在經過 Aurora PostgreSQL 測試並核准後，才可供您使用。

Aurora 會自動為您套用次要版本升級。建立新 Aurora PostgreSQL 資料庫叢集時，系統預設已啟用啟用次要版本升級選項。除非您手動關閉此選項，否則 Aurora 會在您排程的維護時段期間定期套用自動次要版本升級。如需自動次要版本升級 (AmVU) 選項及如何修改 Aurora 資料庫叢集以使用的詳細資訊，請參閱 Aurora 資料庫叢集的自動次要版本升級。

如果未為 Aurora PostgreSQL 資料庫叢集啟用自動次要版本升級，則 Aurora PostgreSQL 不會自動升級到新的次要版本。相反地，當在您的 AWS 區域 中發行新次要版本且 Aurora PostgreSQL 資料庫叢集正在執行較舊的次要版本時，Aurora 會提示您升級。其做法是將建議新增至叢集的維護任務。

修補程式不視為升級，也不會自動套用。Aurora PostgreSQL 會透過對 Aurora PostgreSQL 資料庫叢集的維護任務新增建議，來提示您套用任何修補程式。如需詳細資訊，請參閱如何執行次要版本升級和套用修補程式。

> **Note**
>
> 此外，也會新增可解決安全問題或其他重大問題的修補程式作為維護任務。但是，這些修補程式是必要的。當安全修補程式在您的待定維護任務中可供使用時，務必套用到您的 Aurora PostgreSQL 資料庫叢集。
> 自動次要版本的升級會執行至預設次要版本。

當叢集中每個執行個體升級到新版本時，升級程序可能會出現短暫中斷。不過，在 Aurora PostgreSQL 14.3.3 版、13.7.3 版、12.11.3 版、11.16.3 版、10.21.3 版，以及這些次要版本的其他更高版本和更新的主要版本之後，升級程序會使用零停機時間修補 (ZDP) 功能。此功能可儘量縮短中斷時間，在大多數情況下能夠完全避免中斷。如需詳細資訊，請參閱次要版本升級和零停機時間修補。如需 ZDP 支援功能和限制的詳細資訊，請參閱 零停機時間修補的限制。

主要版本升級

不同於次要版本升級和修補程式，Aurora PostgreSQL 沒有自動主要版本升級選項。新主要 PostgreSQL 版本可能包含與現有應用程式回溯不相容的資料庫變更。新功能可能導致現有的應用程式停止正確運作。

為了防止出現任何問題，強烈建議您先按照 測試執行生產資料庫叢集升級到新主要版本的程序 中所列的程序進行，再升級 Aurora PostgreSQL 資料庫叢集中的資料庫執行個體。首先，請按照該程序確定您的應用程式可以在新版上執行。然後，您可以手動將 Aurora PostgreSQL 資料庫叢集升級到新版本。

當叢集中所有執行個體都升級到新版本時，升級程序可能會出現短暫中斷。初步規劃過程也需要一些時間。建議您一律在叢集維護時段或作業最少的時候執行升等任務。如需詳細資訊，請參閱執行主要版本升級。

> **Note**
>
> 次要版本升級和主要版本升級可能會造成短暫的中斷。因此，強烈建議您在維護時段或其他低利用率期間執行或安排升級。

Aurora PostgreSQL 資料庫叢集偶爾需要作業系統更新。這些更新可能會包含較新版本的 glibc 程式庫。在此類更新期間，建議您遵循 Aurora PostgreSQL 支援的定序 中所述指示。

取得 中可用版本的清單 AWS 區域

您可以使用 AWS 區域 describe-db-engine-versions AWS CLI 命令查詢 ，以取得可用於 Aurora PostgreSQL 資料庫叢集升級目標的所有引擎版本清單，如下所示。

針對 Linux、macOS 或 Unix：

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version version-number \
  --query 'DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}' \
  --output text
```

在 Windows 中：

```
aws rds describe-db-engine-versions ^
```

```
  --engine aurora-postgresql ^
  --engine-version version-number ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
  --output text
```

例如，若要識別 Aurora PostgreSQL 12.10 版資料庫叢集的有效升級目標，請執行下列 AWS CLI 命令：

針對 Linux、macOS 或 Unix：

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version 12.10 \
  --query 'DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}' \
  --output text
```

在 Windows 中：

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version 12.10 ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
  --output text
```

在下表中，您可以找到適用於不同 Aurora PostgreSQL 資料庫版本的主要與次要版本升級目標。為了維護相容性，並非所有版本都會提供做為升級目標。Aurora PostgreSQL 會在每次每季次要版本發行時推出新功能和錯誤修正。如需 Aurora PostgreSQL 次要版本的詳細資訊，請參閱 Aurora PostgreSQL 版本備註。

目前來源版本	升級目標
17.7	無
17.6	17.7
17.5	17.7、17.6
17.4	17.7、17.6、17.5
16.11	無

目前來源版本	升級目標
16.10	17.7 16.11
16.9	17.7 16.11、17.6、16.10、17.5
16.8	17.7 16.11、17.6、16.10、17.5、17.4、16.9
16.6	17.7 16.11、17.6、16.10、17.5、17.4、16.9、16.8
16.4	17.7 16.11、17.6、16.10、17.5、17.4、16.9、16.8、16.6
16.3	17.7 16.11、17.6、16.10、17.5、17.4、16.9、16.8、16.6、16.4
16.2	17.7 16.11、17.6、16.10、17.5、17.4、16.9、16.8、16.6、16.4、16.3
16.1	17.7 16.11、17.6、16.10、17.5、17.4、16.9、16.8、16.6、16.4、16.3、16.2
15.15	無
15.14	17.7 16.11 15.15

目前來源版本	升級目標
15.13	17.7 16.11、15.15、17.6、16.10、15.14、17.5、16.9
15.12	17.7 16.11、15.15、17.6、16.10、15.14、15.13
15.10	17.7 16.11、15.15、17.6、16.10、15.14、17.5、17.4、16.9、16.8、15.13
15.11	17.7 16.11、15.15、17.6、16.10、15.14、17.5、17.4、16.9、16.8、16.6、15.13、
14.20	無

對於您正在考慮的任何版本，請務必檢查您叢集資料庫執行個體類別的可用性。例如，Aurora PostgreSQL 13 不支援 db.r4。如果您的 Aurora PostgreSQL 資料庫叢集目前使用 db.r4 執行個體類別，您必須先將其修改為使用支援的資料庫執行個體類別，才能升級至 Aurora PostgreSQL 13。如需可用 Aurora PostgreSQL 資料庫執行個體類別的詳細資訊，包括哪些是Graviton2-based 和哪些是基於 Intel 的執行個體類別，請參閱Amazon Aurora 資料庫執行個體類別。

執行主要版本升級

主要版本升級可能包含與舊版資料庫不相容的資料庫變更。新版本中的新功能可能導致現有的應用程式停止正確運作。若要避免發生問題，Amazon Aurora 不會自動套用主要版本升級。相反，建議您按照以下步驟謹慎規劃主要版本升級：

1. 從表格中為您的版本列出的可用目標清單中，選擇您要的主要版本。您可以使用 取得 AWS 區域 目前版本在 中可用的版本精確清單 AWS CLI。如需詳細資訊，請參閱取得 中可用版本的清單 AWS 區域。
2. 確認您的應用程式在新版本的試用部署中能如預期運作。如需完整程序的相關資訊，請參閱 測試執行生產資料庫叢集升級到新主要版本的程序。
3. 當確認您的應用程式在試用部署中能如按預期運作後，您便可以升級叢集。如需詳細資訊，請參閱將 Aurora PostgreSQL 引擎升級為新的主要版本。

> **Note**
>
> 您可以執行從 Babelfish for Aurora PostgreSQL 13 型版本 (從 13.6 開始) 至 Aurora PostgreSQL 14 型版本 (從 14.6 開始) 的主要版本升級。Babelfish for Aurora PostgreSQL 13.4 和 13.5 不支援主要版本升級。

您可以使用 AWS 區域 describe-db-engine-versions AWS CLI 命令查詢 ，以取得可用於 Aurora PostgreSQL 資料庫叢集主要版本升級目標的引擎版本清單，如下所示。

針對 Linux、macOS 或 Unix：

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version version-number \
  --query 'DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}' \
  --output text
```

在 Windows 中：

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version version-number ^
  --query "DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}" ^
  --output text
```

在某些情況下，您要升級到的版本不是目前版本的目標。在這種情況下，請使用 versions table 中的資訊執行次要版本升級，直到叢集的版本在其目標列中具有您選擇的目標為止。

測試執行生產資料庫叢集升級到新主要版本的程序

每個新的主要版本都包括旨在提高效能之查詢最佳化工具的增強功能。但是，您的工作負載可能包括導致新版本中計畫效能變差的查詢。這就是為什麼我們建議您在升級生產之前測試和檢閱效能。您可使用查詢計劃管理 (QPM) 擴充功能管理跨版本管理查詢計劃的穩定性，詳情請參閱 主要版本升級之後確保計畫穩定性。

將生產 Aurora PostgreSQL 資料庫叢集升級至新的主要版本之前，強烈建議您測試該升級，以確認您所有的應用程式皆可正確運作：

1. 準備好版本相容的參數群組。

   如果您使用的是自訂資料庫執行個體或資料庫叢集參數群組，您會有兩個選擇：

   a. 指定預設資料庫執行個體、資料庫叢集參數群組，或同時為新資料庫引擎版本指定這兩者。

   b. 為新資料庫引擎版本建立自己的自訂參數群組。

2. 檢查是否存在無效資料庫並捨棄任何存在的無效資料庫。

   pg_database 目錄中的 datconnlimit 欄包含值 -2，用於將 DROP DATABASE 操作期間中斷的任何資料庫標記為無效。使用以下查詢來檢查是否存在無效的資料庫。

   ```
   SELECT
       datname
   FROM
       pg_database
   WHERE
       datconnlimit = - 2;
   ```

   如果查詢傳回資料庫名稱，則這些資料庫無效。使用 DROP DATABASE invalid_db_name 陳述式捨棄無效的資料庫。您可以使用下列動態陳述式來捨棄所有無效的資料庫。

   ```
   SELECT
       'DROP DATABASE ' || quote_ident(datname) || ';'
   FROM
       pg_database
   WHERE
       datconnlimit = -2 \gexec
   ```

3. 檢查是否有不支援的使用：

   - 嘗試升級之前，遞交或轉返所有開啟的備妥交易。您可以使用下列查詢，以確認在執行個體上沒有開啟的備妥交易：

     ```
     SELECT count(*) FROM pg_catalog.pg_prepared_xacts;
     ```

   - 嘗試升級之前，請移除所有 reg* 資料類型的使用。除了 regtype 和 regclass，您無法升級 reg* 資料類型。pg_upgrade 公用程式 (由 Amazon Aurora 用於進行升級) 無法保留此資料類型。若要進一步了解此公用程式，請參閱 PostgreSQL 文件中的 pg_upgrade。

     您可以使用下列查詢，確認在每個資料庫中未使用不支援的 reg* 資料類型：

```
SELECT count(*) FROM pg_catalog.pg_class c, pg_catalog.pg_namespace n,
 pg_catalog.pg_attribute a
  WHERE c.oid = a.attrelid
      AND NOT a.attisdropped
      AND a.atttypid IN ('pg_catalog.regproc'::pg_catalog.regtype,
                         'pg_catalog.regprocedure'::pg_catalog.regtype,
                         'pg_catalog.regoper'::pg_catalog.regtype,
                         'pg_catalog.regoperator'::pg_catalog.regtype,
                         'pg_catalog.regconfig'::pg_catalog.regtype,
                         'pg_catalog.regdictionary'::pg_catalog.regtype)
      AND c.relnamespace = n.oid
      AND n.nspname NOT IN ('pg_catalog', 'information_schema');
```

- 如果您要升級安裝了 pgRouting 擴充功能的 Aurora PostgreSQL 版本 10.18 或更高版本的資料庫叢集，請在升級到版本 12.4 或更高版本之前刪除該擴充功能。

  如果您要升級安裝了 pg_repack 1.4.3 版的 Aurora PostgreSQL 10.x 版本，請在升級到更高版本之前刪除該擴充功能。

4. 檢查 template1 和 template0 資料庫。

   如需成功升級，template1 和 template0 資料庫必須存在，且應列為範本。若要檢查此情況，請使用下列命令：

```
SELECT datname, datistemplate FROM pg_database;

datname    | datistemplate
-----------+---------------
template0  | t
rdsadmin   | f
template1  | t
postgres   | f
```

   在命令輸出中，template1 和 template0 資料庫的 datistemplate 值應該是 t。

5. 刪除邏輯複寫槽。

   如果 Aurora PostgreSQL 資料庫叢集正在使用任何邏輯複寫槽，則升級程序將無法繼續。邏輯複寫槽通常用於短期資料遷移任務，例如使用 AWS DMS 或 將資料表從資料庫複寫到資料湖、BI 工具或其他目標。升級之前，請確保您知道任何現有邏輯複寫槽的用途，並確認可以刪除它們。您可以使用下列查詢來檢查邏輯複寫槽：

```
SELECT * FROM pg_replication_slots;
```

如果邏輯複寫槽仍在使用，則不應刪除它們，也無法繼續升級。但是，如果不需要邏輯複寫槽，可以使用以下 SQL 加以刪除：

```
SELECT pg_drop_replication_slot(slot_name);
```

使用 pglogical 延伸模組的邏輯複寫案例也必須具有已從發佈者節點捨棄的插槽，才能在該節點上順利進行主要版本升級。不過，您可以在升級之後，從訂閱者節點重新啟動複寫程序。如需詳細資訊，請參閱在重大升級之後重新建立邏輯複寫。

6. 執行備份。

   升級程序會在升級期間建立資料庫叢集的資料庫叢集快照。如果您也想在升級程序之前執行手動備份，請參閱 建立資料庫叢集快照 以取得詳細資訊。

7. 在執行主要版本升級之前，請將某些擴充功能套件升級至最新可用版本。需更新的擴充功能包含下列：

   - pgRouting
   - postgis_raster
   - postgis_tiger_geocoder
   - postgis_topology
   - address_standardizer
   - address_standardizer_data_us

   針對目前已安裝的每個擴充功能執行下列命令。

   ```
   ALTER EXTENSION PostgreSQL-extension UPDATE TO 'new-version';
   ```

   如需詳細資訊，請參閱升級 PostgreSQL 延伸。若要深入了解 PostGIS 升級，請參閱 步驟 6：升級 PostGIS 擴充功能。

8. 若要升級至第 11.x 版，請先捨棄不支援的擴充功能套件，然後再執行主要版本升級。需捨棄的擴充功能套件包括：

   - chkpass
   - tsearch2

9. 根據您的目標版本捨棄 unknown 資料類型。

PostgreSQL 10 版不支援 unknown 資料類型。如果第 9.6 版資料庫使用 unknown 資料類型，升級至版本 10 會顯示錯誤訊息，如下所示。

```
Database instance is in a state that cannot be upgraded: PreUpgrade checks failed:
The instance could not be upgraded because the 'unknown' data type is used in user
 tables.
Please remove all usages of the 'unknown' data type and try again."
```

若要尋找資料庫中的 unknown 資料類型，以便移除這類資料欄或將其變更為支援的資料類型，請對每個資料庫使用下列 SQL 程式碼。

```
SELECT n.nspname, c.relname, a.attname
    FROM pg_catalog.pg_class c,
    pg_catalog.pg_namespace n,
    pg_catalog.pg_attribute a
    WHERE c.oid = a.attrelid AND NOT a.attisdropped AND
    a.atttypid = 'pg_catalog.unknown'::pg_catalog.regtype AND
    c.relkind IN ('r','m','c') AND
    c.relnamespace = n.oid AND
    n.nspname !~ '^pg_temp_' AND
    n.nspname !~ '^pg_toast_temp_' AND n.nspname NOT IN ('pg_catalog',
 'information_schema');
```

10執行停用試轉升級。

強烈建議您先在生產資料庫的複本上測試主要版本升級，然後再在生產資料庫上嘗試此升級。您可以監控複本測試執行個體上的執行計劃，以查看任何可能的執行計劃迴歸，並評估其效能。若要建立複本測試執行個體，您可以從最近快照還原資料庫或複製您的資料庫。如需更多詳細資訊，請參閱「從快照還原」或「複製 Amazon Aurora 資料庫叢集的一個磁碟區」。

如需更多詳細資訊，請參閱 將 Aurora PostgreSQL 引擎升級為新的主要版本。

11升級您的生產執行個體。

當試轉主要版本升級成功時，您應該就能放心升級生產資料庫。如需詳細資訊，請參閱將 Aurora PostgreSQL 引擎升級為新的主要版本。

> **Note**
>
> 在升級程序期間，Aurora PostgreSQL 會取得資料庫叢集快照。在此升級期間，您無法進行叢集的時間點還原。之後，您可以執行時間點還原，以便還原至升級開始之前的時間，以及執行個體的自動快照已完成之後的時間。但是，您無法執行時間點還原至先前的次要版本。

如需有關升級進行中的資訊，您可以使用 Amazon RDS 查看 pg_upgrade 公用程式產生的兩個記錄。這兩個記錄是 `pg_upgrade_internal.log` 和 `pg_upgrade_server.log`。Amazon Aurora 會將時間戳記附加至這些日誌的檔案名稱。您可以如同檢視任何其他日誌一般檢視這些日誌。如需更多詳細資訊，請參閱 監控 Amazon Aurora 日誌檔案。

12. 升級 PostgreSQL 擴充功能套件。PostgreSQL 升級程序並不會升級任何 PostgreSQL 擴充功能套件。如需詳細資訊，請參閱升級 PostgreSQL 延伸。

升級後建議

完成主要版本升級後，建議您執行下列動作：

- 執行 ANALYZE 操作以重新整理 `pg_statistic` 資料表。您應為所有 PostgreSQL 資料庫執行個體上的每個資料庫執行此操作。主要版本升級期間不會傳輸最佳化工具統計數字，因此您需要重新產生所有統計數字以避免效能問題。在沒有任何參數的情況下執行命令，以產生目前資料庫中所有一般資料表的統計數字，如下所示：

```
ANALYZE VERBOSE;
```

  您可選用 `VERBOSE` 旗標，但使用時會顯示進度。如需詳細資訊，請參閱 PostgreSQL 說明文件中的 ANALYZE。

  分析特定資料表而非使用 ANALYZE VERBOSE 時，請對每個資料表執行 ANALYZE 命令，如下所示：

```
ANALYZE table_name;
```

  對於分割資料表，一律分析父資料表。此程序：

  - 自動取樣所有分割區的資料列
  - 以遞迴方式更新每個分割區的統計資料

- 在父層級維護基本的規劃統計資料

  雖然父資料表不會存放實際資料，但分析它們對於查詢最佳化至關重要。僅在個別分割區上執行 ANALYZE 可能會導致查詢效能不佳，因為最佳化工具不會擁有有效跨分割區規劃所需的完整統計資料。

  > **Note**
  >
  > 升級後在系統上執行 ANALYZE，以避免效能問題。

- 如果您升級到 PostgreSQL 第 10 版，請在您擁有的任何雜湊索引上執行 REINDEX。雜湊索引在第 10 版中已變更並且必須重建。若要定位無效的雜湊索引，請針對每個包含雜湊索引的資料庫執行下列 SQL。

```
SELECT idx.indrelid::regclass AS table_name,
   idx.indexrelid::regclass AS index_name
FROM pg_catalog.pg_index idx
   JOIN pg_catalog.pg_class cls ON cls.oid = idx.indexrelid
   JOIN pg_catalog.pg_am am ON am.oid = cls.relam
WHERE am.amname = 'hash'
AND NOT idx.indisvalid;
```

- 建議您使用類似的工作負載，在升級後的資料庫上測試您的應用程式，以驗證一切是否依預期運作。確認升級之後，您可以刪除該測試執行個體。

將 Aurora PostgreSQL 引擎升級為新的主要版本

啟動新主要版本的升級程序時，Aurora PostgreSQL 會先擷取 Aurora 資料庫叢集的快照，再對叢集進行任何變更。此快照僅針對主要版本升級而建立，而不是針對次要版本升級。升級程序完成後，您可以在 RDS 主控台的 Snapshots (快照) 下方所列手動快照之中找到此快照。快照名稱包含作為其字首的 `preupgrade`、您 Aurora PostgreSQL 資料庫叢集的名稱、來源版本、目標版本，以及日期與時間戳記，如以下範例所示。

```
preupgrade-docs-lab-apg-global-db-12-8-to-13-6-2022-05-19-00-19
```

升級完成後，如有需要，您可以使用 Aurora 建立並儲存在手動快照清單中的快照，將資料庫叢集還原至其先前的版本。

> **Tip**
>
> 一般來說，快照提供許多方法可將 Aurora 資料庫叢集還原到各種時間點。如需了解詳細資訊，請參閱 從資料庫叢集快照還原 和 將資料庫叢集還原至指定時間。但是，Aurora PostgreSQL 不支援使用快照還原至先前的次要版本。

在主要版本升級程序中，Aurora 會配置磁碟區，並複製來源 Aurora PostgreSQL 資料庫叢集。如果升級因任何原因而失敗，Aurora PostgreSQL 會使用複製來復原升級。在配置某個來源磁碟區的複製項超過 15 個時，後續的複製項會變成完整副本，且需要更長的時間。這可能也會導致升級程序花費更長的時間。如果 Aurora PostgreSQL 復原升級，請注意下列事項：

- 您可能會看到原始磁碟區和升級期間配置的複製磁碟區的計費項目和指標。Aurora PostgreSQL 會在叢集備份保留期間超過升級時間後清理額外的磁碟區。
- 來自此叢集的下一個跨區域快照複製將是完整副本，而非遞增複本。

為了安全地將構成叢集的資料庫執行個體升級，Aurora PostgreSQL 使用 pg_upgrade 公用程式。寫入器升級完成後，每個讀取器執行個體都會發生短暫的中斷，而它會升級為新的主要版本。若要進一步了解此 PostgreSQL 公用程式，請參閱 PostgreSQL 文件中的 pg_upgrade。

您可以使用 AWS 管理主控台 AWS CLI、 或 RDS API，將 Aurora PostgreSQL 資料庫叢集升級至新版本。

主控台

升級資料庫叢集的引擎版本

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中選擇 Databases (資料庫)，然後選擇您要升級的資料庫叢集。
3. 選擇 Modify (修改)。Modify DB cluster (修改資料庫叢集) 頁面隨即出現。
4. 在 Engine version (引擎版本) 中，選擇新版本。
5. 選擇 Continue (繼續)，並檢查修改的摘要。
6. 若要立即套用變更，請選擇 Apply immediately (立即套用)。在某些情況下，選擇此選項會導致停機。如需更多詳細資訊，請參閱 修改 Amazon Aurora 資料庫叢集。
7. 在確認頁面上，檢閱您的變更。如果都正確，請選擇 Modify cluster (修改叢集) 以儲存您的變更。

或者，選擇 Back (上一步) 以編輯變更，或是選擇 Cancel (取消) 以取消變更。

AWS CLI

若要升級資料庫叢集的引擎版本，請使用 modify-db-cluster AWS CLI 命令。指定下列參數：

- `--db-cluster-identifier` – 資料庫叢集的名稱。
- `--engine-version` – 會以此資料庫引擎版本編號為目標進行升級。如需有效引擎版本的資訊，請使用 AWS CLI describe-db-engine-versions 命令。
- `--allow-major-version-upgrade` – 此為當 `--engine-version` 參數是不同於資料庫叢集目前主要版本的主要版本時的必要旗標。
- `--no-apply-immediately` – 下個維護時段再套用變更。若要立即套用變更，請使用 `--apply-immediately`。

Example

針對 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine-version new_version \
    --allow-major-version-upgrade \
    --no-apply-immediately
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine-version new_version ^
    --allow-major-version-upgrade ^
    --no-apply-immediately
```

RDS API

若要升級資料庫叢集的引擎版本，請使用 ModifyDBCluster 操作。指定下列參數：

- `DBClusterIdentifier` – 資料庫叢集的名稱，例如 *mydbcluster*。

- `EngineVersion` – 會以此資料庫引擎版本編號為目標進行升級。如需有效引擎版本的資訊，請使用 DescribeDBEngineVersions 操作。
- `AllowMajorVersionUpgrade` – 此為當 `EngineVersion` 參數是不同於資料庫叢集目前主要版本的主要版本時的必要旗標。
- `ApplyImmediately` – 指出要立即套用變更，或等到下個維護時段再套用。若要立即套用變更，請將值設為 `true`。若要在下一次維護時段套用變更，請將值設為 `false`。

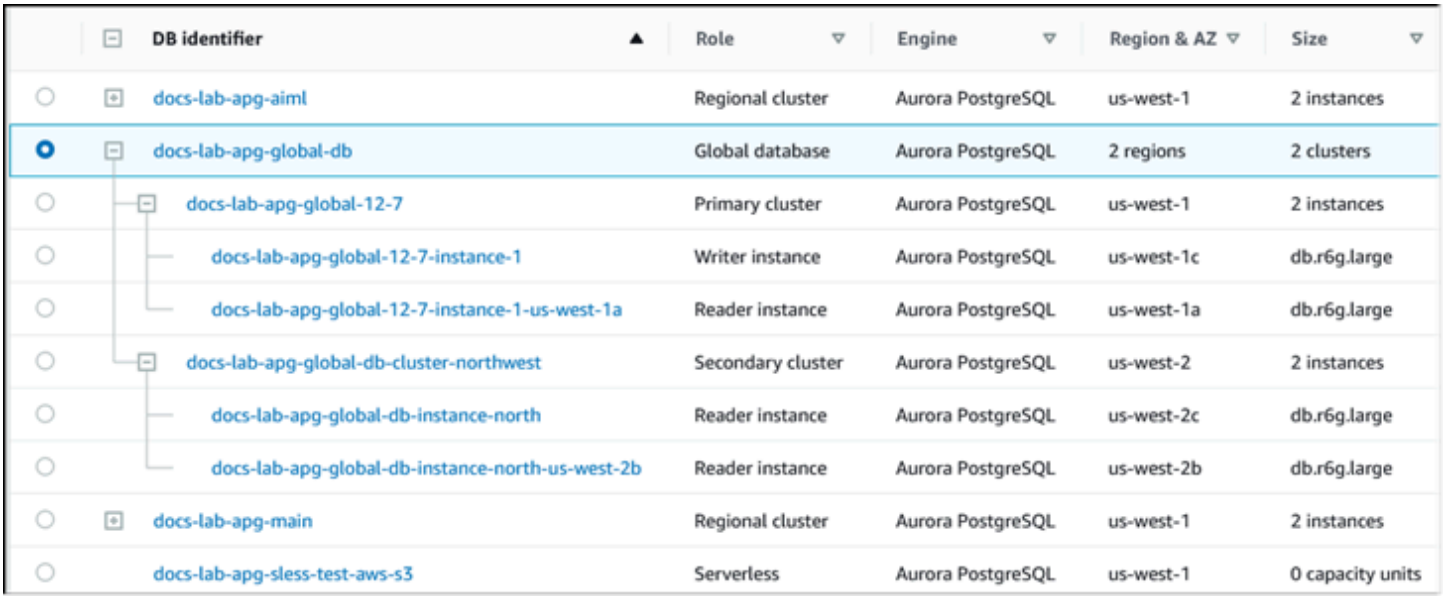
全域資料庫的主要升級

對於 Aurora 全域資料庫叢集，此升級程序會同時升級構成 Aurora 全域資料庫的所有資料庫叢集。這麼做是為了確保每個資料庫叢集都會執行相同的 Aurora PostgreSQL 版本。這也會確定對系統資料表、資料檔案格式等所做的任何變更都會自動複寫至所有次要叢集。

若要將全域資料庫叢集升級成 Aurora PostgreSQL 的新要主版本，建議您在升級版本上測試您的應用程式，詳情請參閱 測試執行生產資料庫叢集升級到新主要版本的程序。在升級之前，請務必為 AWS 區域 Aurora 全域資料庫中的每個 準備資料庫叢集參數群組和資料庫參數群組設定，如 step 1. 的 中所述測試執行生產資料庫叢集升級到新主要版本的程序。

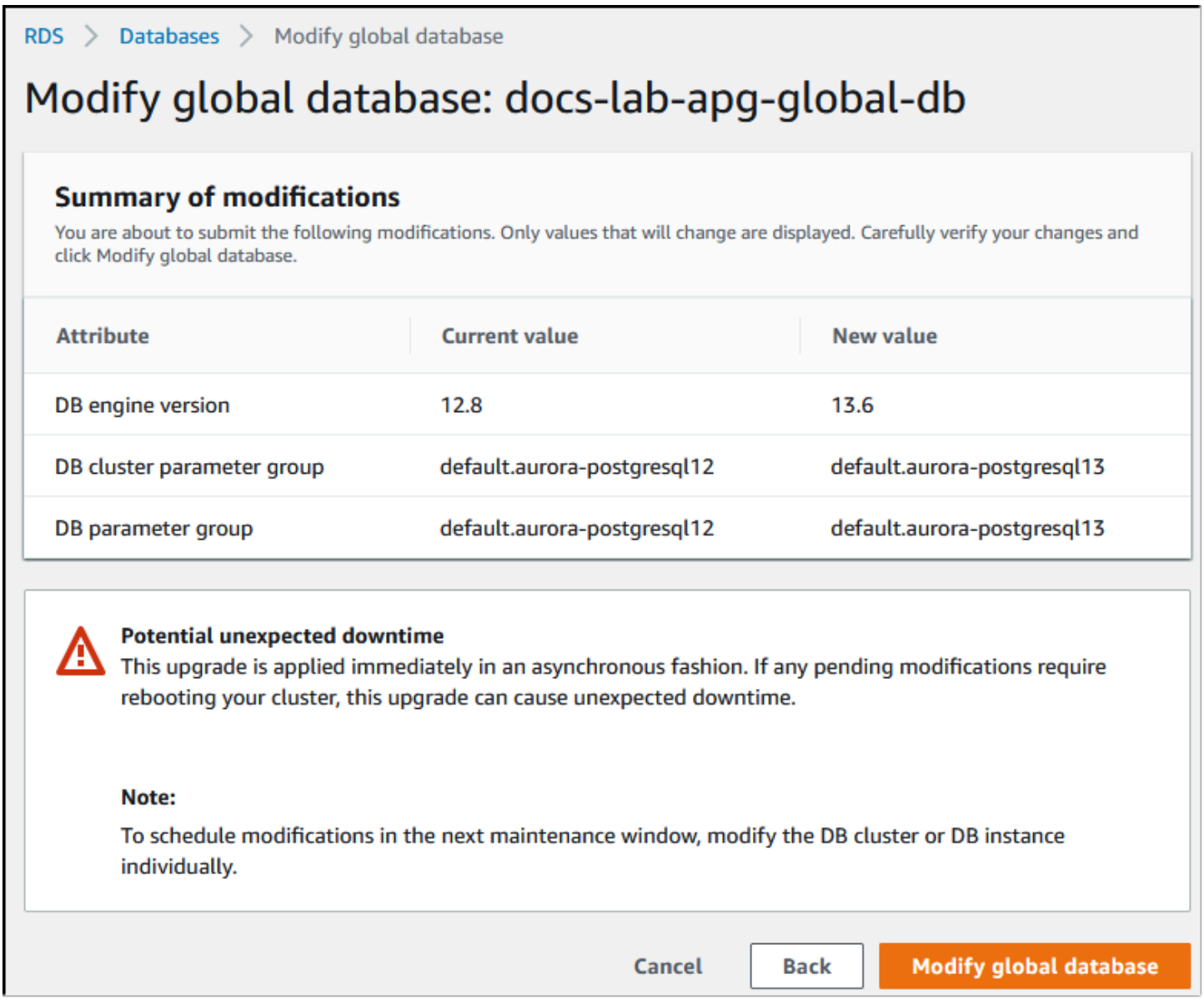
如果 Aurora PostgreSQL 全域資料庫叢集已為其 `rds.global_db_rpo` 參數設定復原點目標 (RPO)，在升級前請務必先重設該參數。如果 RPO 已開啟，則主要版本升級程序不會運作。依預設，此參數為關閉。如需 Aurora PostgreSQL 全域資料庫和 RPO 的詳細資訊，請參閱 管理 Aurora PostgreSQL – 全域資料庫的 RPO。

如果您確認您的應用程式在新版本的試用部署中可以如預期執行，則可以啟動升級程序。若要這麼做，請參閱將 Aurora PostgreSQL 引擎升級為新的主要版本。請務必從 RDS 主控台的 Databases (資料庫) 清單中選擇最頂層項目，亦即 Global database (全域資料庫)，如下圖所示。



DB identifier	Role	Engine	Region & AZ	Size
docs-lab-apg-aiml	Regional cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-global-db	Global database	Aurora PostgreSQL	2 regions	2 clusters
docs-lab-apg-global-12-7	Primary cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-global-12-7-instance-1	Writer instance	Aurora PostgreSQL	us-west-1c	db.r6g.large
docs-lab-apg-global-12-7-instance-1-us-west-1a	Reader instance	Aurora PostgreSQL	us-west-1a	db.r6g.large
docs-lab-apg-global-db-cluster-northwest	Secondary cluster	Aurora PostgreSQL	us-west-2	2 instances
docs-lab-apg-global-db-instance-north	Reader instance	Aurora PostgreSQL	us-west-2c	db.r6g.large
docs-lab-apg-global-db-instance-north-us-west-2b	Reader instance	Aurora PostgreSQL	us-west-2b	db.r6g.large
docs-lab-apg-main	Regional cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-sless-test-aws-s3	Serverless	Aurora PostgreSQL	us-west-1	0 capacity units

與任何修改一樣，當出現提示時，您可以確認讓此程序繼續進行。



您不是使用主控台啟動升級程序，而是使用 AWS CLI 或 RDS API。如同主控台，您是對 Aurora 全域資料庫叢集上進行操作，而不是對其任何組成部分，如下所示：

- 使用 modify-global-cluster AWS CLI 命令，透過使用 開始 Aurora 全域資料庫的升級 AWS CLI。
- 使用 ModifyGlobalCluster API 啟動升級。

執行次要版本升級

您可以使用下列方法來升級資料庫叢集的次要版本或修補資料庫叢集：

主題

- 執行次要版本升級之前
- 如何執行次要版本升級和套用修補程式
- 次要版本升級和零停機時間修補
- 零停機時間修補的限制
- 將 Aurora PostgreSQL 引擎升級為新的次要版本

執行次要版本升級之前

建議您執行下列動作，以減少次要版本升級期間的停機時間：

- Aurora 資料庫叢集維護應在低流量期間執行。使用 Performance Insights 來識別這些時段，以正確設定維護時段。如需 Performance Insights 的詳細資訊，請參閱在 Amazon RDS 上使用 Performance Insights 監控資料庫負載。如需資料庫叢集維護時段的詳細資訊，請參閱 調整偏好的資料庫叢集維護時段。
- 最佳實務是支援指數退避和抖動 AWS SDKs。如需詳細資訊，請參閱指數退避和抖動。

如何執行次要版本升級和套用修補程式

AWS 區域 只有在嚴格測試之後， 才會提供次要版本升級和修補程式。在發佈升級和修補程式之前，Aurora PostgreSQL 會進行測試，以確保在次要社群版本發佈後出現的已知安全問題、錯誤和其他問題，不會破壞 Aurora PostgreSQL 機群的穩定性。

如果您開啟啟用自動次要版本升級，Aurora PostgreSQL 會在指定維護時段期間定期升級您的資料庫叢集。確定已對 Aurora PostgreSQL 資料庫叢集中的所有執行個體開啟啟用自動次要版本升級選項。如需了解如何設定自動次要版本升級，以及該設定套用於叢集和執行個體層級時如何運作，請參閱 Aurora 資料庫叢集的自動次要版本升級。

使用 describe-db-instances AWS CLI 命令，檢查所有 Aurora PostgreSQL 資料庫叢集的啟用自動次要版本升級選項的值。

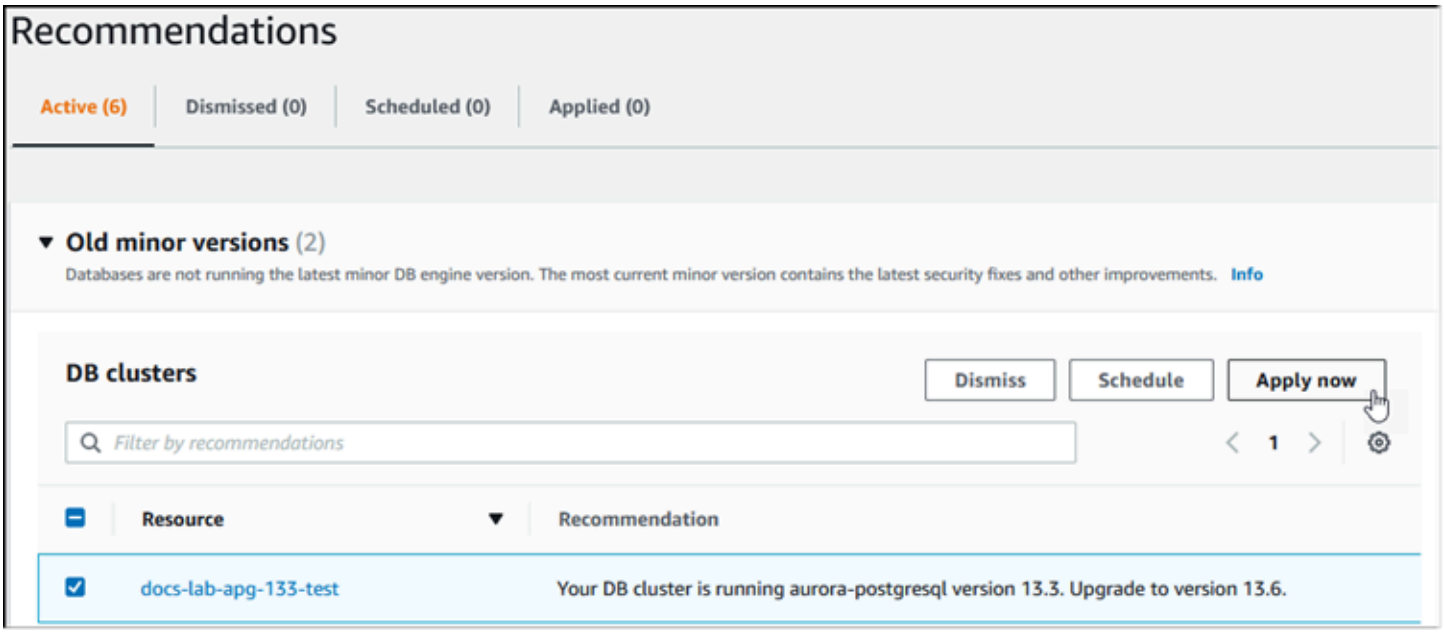
```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

此查詢會傳回一個清單，其中列出所有的 Aurora 資料庫叢集，以及 `AutoMinorVersionUpgrade` 設定的狀態值為 `true` 或 `false` 的執行個體。命令假設您已將 AWS CLI 設定為以預設值為目標 AWS 區域。

如需 AmVU 選項及如何修改 Aurora 資料庫叢集以使用的詳細資訊，請參閱 Aurora 資料庫叢集的自動次要版本升級。

您可以透過回應維護任務，或修改叢集以使用新版本，將 Aurora PostgreSQL 資料庫叢集升級為新的次要版本。

您可以識別 Aurora PostgreSQL 資料庫叢集適用的任何升級或修補程式，方法是使用 RDS 主控台，然後開啟 Recommendations (建議) 功能表。在此，您可以找到各種維護問題的清單，例如 Old minor versions (舊次要版本)。視您的生產環境而定，您可以選擇 Schedule (排程) 升級，或選擇 Apply now (立即套用) 立即採取動作，如下所示。



若要進一步了解如何維護 Aurora 資料庫叢集，包括如何手動套用修補程式和次要版本升級，請參閱 維持 為 Amazon Aurora 資料庫叢集。

次要版本升級和零停機時間修補

升級 Aurora PostgreSQL 資料庫叢集可能會造成中斷。在升級過程中，資料庫在升級期間會關閉。如果您在資料庫忙碌時開始升級，則會遺失資料庫叢集正在處理的所有連線和交易。如果您要等到資料庫閒置才執行升級，就可能必須等待很長時間。

零停機時間修補 (ZDP) 功能改進了升級程序。透過 ZDP，可讓套用次要版本升級和修補程式對 Aurora PostgreSQL 資料庫叢集造成的影響最小。將修補程式或更新的次要版本升級套用至 Aurora PostgreSQL 版本以及這些次要版本的其他更新版本和更新的主要版本時，會使用 ZDP。也就是說，從上述任何一個版本之後的版本升級至新的次要版本都會使用 ZDP。

下列資料表顯示可使用 ZDP 的 Aurora PostgreSQL 版本和資料庫執行個體類別：

版本	db.r* 執行個體類別	db.t* 執行個體類別	db.x* 執行個體類別	db.serverless 執行個體類別
10.21 和更新版本	是	是	是	N/A
11.16 和更新版本	是	是	是	N/A
11.17 和更新版本	是	是	是	N/A
12.11 和更新版本	是	是	是	N/A
12.12 和更新版本	是	是	是	N/A
13.7 和更新版本	是	是	是	N/A
13.8 和更新版本	是	是	是	是
14.3 和更新版本	是	是	是	N/A
14.4 和更新版本	是	是	是	N/A
14.5 和更新版本	是	是	是	是
15.3 和更新版本	是	是	是	是
16.1 和更新版本	是	是	是	是

在使用 ZDP 進行升級程序期間，資料庫引擎會尋找安靜點以暫停所有新事務。此動作可在修補程式和升級期間保護資料庫。為了確保應用程式在事務暫停期間順利執行，建議您將重試邏輯整合到程式碼中。此方法可確保系統能夠管理任何短暫的停機時間而不致失效，且可在升級後重試新的事務。

當 ZDP 成功完成時，會保持應用程式工作階段 (但中斷連線的工作階段除外)，而且資料庫引擎也會重新啟動，但同時間升級仍在進行中。雖然資料庫引擎重新啟動可能會導致輸送量暫時下降，但一般只會持續數秒，最多持續一分鐘左右。

在某些情況下，零停機時間修補 (ZDP) 可能無法成功。例如，在 Aurora PostgreSQL 資料庫叢集或其執行個體上狀態為 pending 的參數變更會干擾 ZDP。

您可以在主控台的 Events (事件) 頁面中找到 ZDP 操作的指標與事件。事件包括 ZDP 升級的開始和升級的完成。在這種情況下，您可以找到此過程所花的時間，以及重新啟動期間發生的保留和捨棄連線數量。您可以在資料庫錯誤日誌中找到詳細資訊。

零停機時間修補的限制

下列限制適用於零停機時間修補：

- ZDP 嘗試在整個 Aurora PostgreSQL 升級程序中，保留與您的 Aurora PostgreSQL 寫入器執行個體的目前用戶端連線。不過，在下列情況下，將會中斷連線，讓 ZDP 完成：
  - 長時間執行的查詢或交易正在進行中。
  - 資料定義語言 (DDL) 陳述式執行中。
  - 暫存資料表或資料表鎖定正在使用中。
  - 所有工作階段都在通知通道上接聽。
  - 處於 'WITH HOLD' 狀態的游標正在使用中。
  - TLSv1.1 連線正在使用中。從 16.1、15.3、14.8、13.11、12.15 和 11.20 之後的 Aurora PostgreSQL 版本開始，TLSv1.3 連線支援 ZDP。
- 下列情況不支援 ZDP：
  - 當 Aurora PostgreSQL 資料庫叢集設定為 Aurora Serverless v1 時。
  - 在任何 Aurora 讀取器執行個體升級期間。
  - 在次要區域中屬於 Aurora 全球資料庫叢集的任何 Aurora 讀取器執行個體升級期間。
  - 在作業系統修補程式和作業系統升級期間

將 Aurora PostgreSQL 引擎升級為新的次要版本

您可以使用 主控台 AWS CLI、 或 RDS API，將 Aurora PostgreSQL 資料庫叢集升級至新的次要版本。在執行升級之前，我們建議您遵循與主要版本升級建議的相同最佳作法。與新的主要版本一樣，新的次要版本也有優化器改進 (如修復)，這可能會導致查詢計劃迴歸。為了確保計劃穩定，我們建議您使用查詢管理計畫 (QPM) 擴充功能，如 主要版本升級之後確保計畫穩定性 中詳細說明。

主控台

升級 Aurora PostgreSQL 資料庫叢集的引擎版本

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中選擇 Databases (資料庫)，然後選擇您要升級的資料庫叢集。
3. 選擇 Modify (修改)。Modify DB cluster (修改資料庫叢集) 頁面隨即出現。
4. 在 Engine version (引擎版本) 中，選擇新版本。
5. 選擇 Continue (繼續)，並檢查修改的摘要。
6. 若要立即套用變更，請選擇 Apply immediately (立即套用)。在某些情況下，選擇此選項會導致停機。如需更多詳細資訊，請參閱 修改 Amazon Aurora 資料庫叢集。
7. 在確認頁面上，檢閱您的變更。如果都正確，請選擇 Modify cluster (修改叢集) 以儲存您的變更。

   或者，選擇 Back (上一步) 以編輯變更，或是選擇 Cancel (取消) 以取消變更。

AWS CLI

若要升級資料庫叢集的引擎版本，請將 modify-db-cluster AWS CLI 命令搭配以下參數使用：

- `--db-cluster-identifier` –Aurora PostgreSQL 資料庫叢集的名稱。
- `--engine-version` – 會以此資料庫引擎版本編號為目標進行升級。如需有效引擎版本的資訊，請使用 AWS CLI describe-db-engine-versions 命令。
- `--no-apply-immediately` – 下個維護時段再套用變更。若要立即套用變更，請改用 `--apply-immediately`。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine-version new_version \
    --no-apply-immediately
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
```

```
    --engine-version new_version ^
    --no-apply-immediately
```

RDS API

若要升級資料庫叢集的引擎版本，請使用 ModifyDBCluster 操作。指定下列參數：

- `DBClusterIdentifier` – 資料庫叢集的名稱，例如 *`mydbcluster`*。
- `EngineVersion` – 會以此資料庫引擎版本編號為目標進行升級。如需有效引擎版本的資訊，請使用 DescribeDBEngineVersions 操作。
- `ApplyImmediately` – 指出要立即套用變更，或等到下個維護時段再套用。若要立即套用變更，請將值設為 `true`。若要在下一次維護時段套用變更，請將值設為 `false`。

升級 PostgreSQL 延伸

將 Aurora PostgreSQL 資料庫叢集升級至新的主要或次要版本不會同時升級 PostgreSQL 延伸模組。對於大多數的延伸模組，您可以在主要或次要版本升級完成後升級延伸模組。但在某些情況下，在升級 Aurora PostgreSQL 資料庫引擎之前要先升級擴充功能。如需詳細資訊，請參閱 測試執行生產資料庫叢集升級到新主要版本的程序 中的 list of extensions to update。

安裝 PostgreSQL 擴充功能需要 `rds_superuser` 權限。通常，`rds_superuser` 會將特定擴充功能的許可委派給相關使用者 (角色)，以促進管理給定的延伸模組。這意味著升級 Aurora PostgreSQL 資料庫叢集中所有擴充功能的這項任務會涉及許多不同的使用者 (角色)。如果要使用指令碼自動執行升級程序，請特別記得這一點。如需 PostgreSQL 權限和角色的詳細資訊，請參閱 Amazon Aurora PostgreSQL 的安全性。

> **Note**
>
> 如需如何更新 PostGIS 擴充功能的相關資訊，請參閱 使用 PostGIS 擴充功能管理空間資料 (步驟 6：升級 PostGIS 擴充功能)。
> 若要更新 `pg_repack` 擴充功能，請捨棄該擴充功能，然後在升級的資料庫執行個體中建立新版本。如需詳細資訊，請參閱 `pg_repack` 說明文件中的 pg_repack 安裝。

若要在引擎升級後更新擴充功能，請使用 `ALTER EXTENSION UPDATE` 命令。

```
ALTER EXTENSION extension_name UPDATE TO 'new_version';
```

若要列出目前已安裝的擴充功能，請在下列命令中使用 PostgreSQL pg_extension 目錄。

```
SELECT * FROM pg_extension;
```

若要檢視您的安裝可用的特定擴充功能版本的清單，請在下列命令中使用 PostgreSQL pg_available_extension_versions 檢視。

```
SELECT * FROM pg_available_extension_versions;
```

替代藍/綠升級技術

在某些情況下，您的首要目標是立即從舊叢集切換至升級的叢集。在這類情況下，您也可以使用多步驟程序，並排執行新舊叢集。在這裡，您會將舊叢集的資料複寫到新叢集，直至您準備好接管新叢集。如需詳細資訊，請參閱使用藍/綠部署進行資料庫更新。

使用 Aurora PostgreSQL 長期支援 (LTS) 版本

當您建立或升級資料庫叢集時，每個新的 Aurora PostgreSQL 版本皆可為您保留特定的時間供使用。在這段期間過後，您必須升級該叢集使用的版本。您可以在支援期間結束前手動升級叢集，或 Aurora 可幫您在 Aurora PostgreSQL 版本不再支援時自動升級。

Aurora 指定特定的 Aurora PostgreSQL 版本做為長期支援 (LTS) 版本。使用 LTS 版本的資料庫叢集相較於非 LTS 版本，可以在相同版本上停留更長的時間，且可以進行幾次升級週期。LTS 次要版本僅包含關鍵穩定性和安全問題的錯誤修正（透過修補程式版本）；LTS 版本不包含引進後發行的新功能。

每年一次，在 LTS 次要版本上執行的資料庫叢集會修補至 LTS 版本的最新修補程式版本。我們會執行此修補作業，以協助確保您從累積的安全性和穩定性修正中受益。如果需要套用關鍵修復 (例如安全性)，我們可能會更頻繁地修補 LTS 次要版本。

> **Note**
>
> 若要在 LTS 次要版本的生命週期內繼續使用，請務必停用資料庫執行個體的自動次要版本升級。為避免自動從 LTS 次要版本升級您的資料庫叢集，請清除 Aurora 叢集中所有資料庫執行個體上的啟用自動次要版本升級核取方塊。

針對大部分 Aurora PostgreSQL 叢集，我們建議您升級至最新版本，而不是使用 LTS 版本。如此，可利用 Aurora 做為管理服務並提供您存取最新版本的功能和錯誤修正。LTS 版本適用於具有下列特性的叢集：

- 除了極少發生的關鍵修補程式之外，您無法承擔 Aurora PostgreSQL 應用程式為了升級而停機所帶來的後果。
- 每次 Aurora PostgreSQL 資料庫引擎更新時，您的叢集和應用程式相關連的測試週期都需要較長的時間。
- 您的 Aurora PostgreSQL 叢集資料庫版本有所有應用程式需要的資料庫引擎功能和錯誤修正。

Aurora PostgreSQL 的目前 LTS 版本如下：

- PostgreSQL 16.8。已於 2025 年 4 月 7 日發行。如需詳細資訊，請參閱 Aurora PostgreSQL 版本備註中的 PostgreSQL 16.8。 PostgreSQL
- PostgreSQL 15.10。其於 2024 年 12 月 27 日發行。如需詳細資訊，請參閱《Aurora PostgreSQL 版本備註》中的 PostgreSQL 15.10。
- PostgreSQL 14.6。於 2023 年 1 月 20 日發佈。如需詳細資訊，請參閱《Aurora PostgreSQL 版本備註》中的 PostgreSQL 14.6。
- PostgreSQL 13.9。於 2023 年 1 月 20 日發佈。如需詳細資訊，請參閱《Aurora PostgreSQL 版本備註》中的 PostgreSQL 13.9。
- PostgreSQL 12.9。其已於 2022 年 2 月 25 日發佈。如需詳細資訊，請參閱 Aurora PostgreSQL 版本備註中的 PostgreSQL 12.9。
- PostgreSQL 11.9 (Aurora PostgreSQL 3.4 版)。它於 2020 年 12 月 11 日發行。如需有關此版本的詳細資訊，請參閱 Aurora PostgreSQL 版本備註中的 PostgreSQL 11.9、Aurora PostgreSQL 3.4 版。

如需 LTS 版本的支援時間表和發布週期的詳細資訊，請參閱 Aurora PostgreSQL 的發佈行事曆。

如需有關如何識別 Aurora 和資料庫引擎版本的資訊，請參閱 識別 Amazon Aurora PostgreSQL 版本。

使用 Amazon Aurora PostgreSQL Limitless Database

Amazon Aurora PostgreSQL Limitless Database 提供自動化水平擴展，每秒處理數百萬筆寫入交易以及管理 PB 級資料，同時維持在單一資料庫中操作的簡單性。使用 Aurora PostgreSQL Limitless Database，您就可以專注於建置大規模應用程式，無需為跨多個資料庫執行個體擴展資料以支援工作負載，而費心建置和維護複雜的解決方案。

主題

- Aurora PostgreSQL Limitless Database 架構
- Aurora PostgreSQL Limitless Database 入門
- Aurora PostgreSQL Limitless Database 要求和考量事項
- 使用 Aurora PostgreSQL Limitless Database 的先決條件
- 建立資料庫叢集，此叢集會使用 Aurora PostgreSQL Limitless Database
- 使用資料庫碎片群組
- 建立 Aurora PostgreSQL Limitless Database 資料表
- 將資料載入 Aurora PostgreSQL Limitless Database
- 查詢 Aurora PostgreSQL Limitless Database
- 管理 Aurora PostgreSQL Limitless Database
- 監控 Aurora PostgreSQL Limitless Database
- 執行個體特定的效能和資源監控
- 備份和還原 Aurora PostgreSQL Limitless Database
- 升級 Amazon Aurora PostgreSQL Limitless Database
- Aurora PostgreSQL Limitless Database 參考

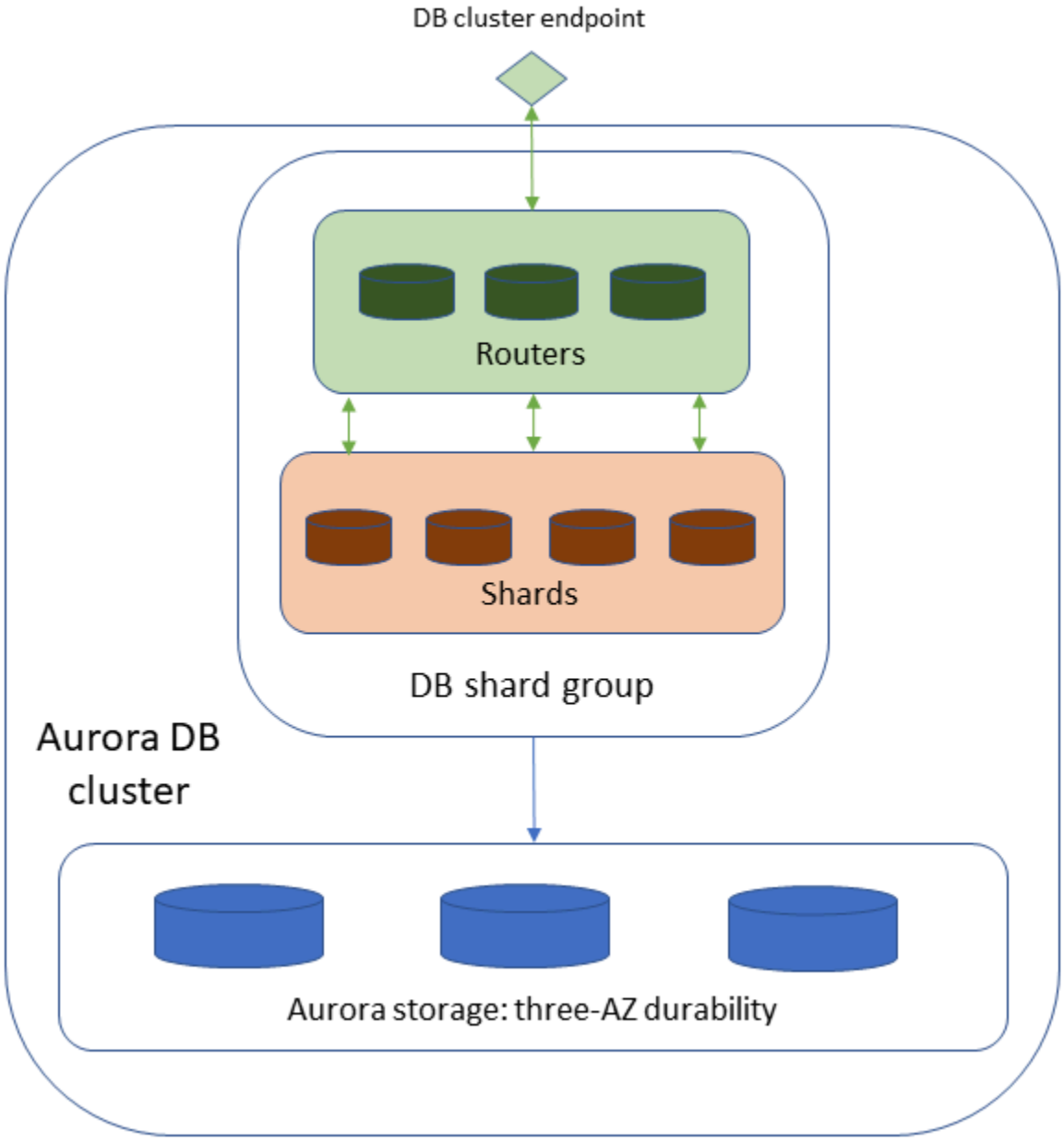
Aurora PostgreSQL Limitless Database 架構

Limitless Database 使用由多個資料庫節點組成的雙層架構來實現擴展。節點可以是路由器或碎片。

- 碎片是 Aurora PostgreSQL 資料庫執行個體，每個執行個體都會存放資料庫的資料子集，允許同時處理以達到更高的寫入輸送量。
- 路由器會管理資料庫的分散式性質，並向資料庫用戶端呈現單一資料庫映像。路由器會維護有關資料存放位置的中繼資料、剖析傳入的 SQL 命令，並將那些命令傳送至碎片。然後，其會彙總碎片中的資料，將單一結果傳回用戶端，以及管理分散式交易，以在整個分散式資料庫中維持一致性。

Aurora PostgreSQL Limitless Database 與標準 Aurora 資料庫叢集 不同之處在於具有資料庫碎片群組 (而不是寫入器資料庫執行個體和讀取器資料庫執行個體)。資料庫碎片群組包含組成 Limitless Database 架構的所有節點。資料庫碎片群組中的個別碎片和路由器不會出現在 AWS 帳戶 中。您可以使用資料庫叢集端點來存取 Limitless Database。

下圖顯示 Aurora PostgreSQL Limitless Database 的高階架構。



如需 Aurora PostgreSQL Limitless Database 架構及如何使用的詳細資訊，請參閱 YouTube 上與 AWS 事件管道相關的使用 Amazon Aurora PostgreSQL Limitless Database 實現擴展。

如需標準 Aurora 資料庫叢集架構的詳細資訊，請參閱 Amazon Aurora 資料庫叢集。

Aurora PostgreSQL Limitless Database 的關鍵術語

資料庫碎片群組

Limitless Database 節點 (碎片和路由器) 的容器。

路由器

接受來自用戶端的 SQL 連線、將 SQL 命令傳送至碎片、維持整個系統的一致性，以及將結果傳回用戶端的節點。

碎片

存放碎片資料表子集、參考資料表完整副本和標準資料表的節點。接受來自路由器的查詢，但用戶端無法直接連線至其中。

碎片資料表

資料表，其資料分割至多個碎片中。

碎片索引鍵

碎片資料表中的欄或一組欄，用於判斷碎片之間的分割。

共置的資料表

共用相同碎片索引鍵且明確宣告為共置的兩個碎片資料表。相同碎片索引鍵值的所有資料都會傳送至相同的碎片。

參考資料表

每個碎片上完整複製其資料的資料表。

標準資料表

Limitless Database 的預設資料表類型。您可以將標準資料表轉換為碎片和參考資料表。

所有標準資料表都存放在系統選取的相同碎片，如此就可以在單一碎片中執行標準資料表之間的聯結。不過，標準資料表受到碎片容量上限 (128 TiB) 的限制。此碎片也會存放碎片和參考資料表的資料，因此標準資料表的有效限制會低於 128 TiB。

Aurora PostgreSQL Limitless Database 的資料表類型

Aurora PostgreSQL Limitless Database 支援三種的資料表：碎片、參考和標準。

碎片資料表會將其資料分佈到資料庫碎片群組中的所有碎片。Limitless Database 會使用碎片索引鍵自動執行此操作，該索引鍵是您分割資料表時指定的欄或一組欄。具有相同碎片金鑰值的所有資料都會傳送至相同的碎片。碎片是以雜湊為基礎，而非以範圍或清單為基礎。

以下是碎片資料表的良好使用案例：

- 應用程式適用於不同的資料子集。
- 資料表非常大。
- 此資料表的成長速度可能比其他資料表快。

碎片資料表可以共置，這表示其共用相同的碎片索引鍵，而且來自具有相同碎片索引鍵值之兩個資料表的所有資料都會傳送至相同的碎片。如果您共置資料表且使用碎片索引鍵加以聯結，則可以在單一碎片執行聯結，因為該碎片上存在所有必要的資料。

參考資料表具有資料庫碎片群組中每個碎片所有資料的完整副本。參考資料表通常會用於寫入量較低的較小資料表，但仍需要經常聯結，而且不適合進行碎片處理。參考資料表的範例包括日期資料表，以及州、城市和郵遞區號等地理資料的資料表。

標準資料表是 Aurora PostgreSQL Limitless Database 中的預設資料表類型。其不是分散式資料表。Aurora PostgreSQL Limitless Database 支援標準資料表與標準、碎片和參考資料表之間的聯結。

Aurora PostgreSQL Limitless Database 的帳單

如需如何向您收取 Aurora PostgreSQL Limitless Database 費用的資訊，請參閱 Aurora 資料庫執行個體計費。

如需 Aurora 定價資訊，請參閱 Aurora 定價頁面。

Aurora PostgreSQL Limitless Database 入門

您可以執行下列動作來開始使用 Aurora PostgreSQL Limitless Database：

1. 建立 Aurora PostgreSQL 資料庫叢集和 Limitless Database 的資料庫碎片群組。如需更多詳細資訊，請參閱 建立資料庫叢集，此叢集會使用 Aurora PostgreSQL Limitless Database。
2. 在資料庫碎片群組中建立碎片和參考資料表。如需更多詳細資訊，請參閱 建立 Aurora PostgreSQL Limitless Database 資料表。
3. 變更資料庫碎片群組、分割碎片和新增路由器的容量。如需更多詳細資訊，請參閱 使用資料庫碎片群組。
4. 將資料載入資料庫碎片群組。如需更多詳細資訊，請參閱 將資料載入 Aurora PostgreSQL Limitless Database。
5. 在資料庫碎片群組執行查詢和其他 SQL 陳述式。如需更多詳細資訊，請參閱 查詢 Aurora PostgreSQL Limitless Database。
6. 監控 Limitless Database 的效能。如需更多詳細資訊，請參閱 監控 Aurora PostgreSQL Limitless Database。

Aurora PostgreSQL Limitless Database 要求和考量事項

Aurora PostgreSQL Limitless Database 有下列要求和考量事項。

主題

- Aurora PostgreSQL Limitless Database 的要求
- Aurora PostgreSQL Limitless Database 的考量事項
- Aurora PostgreSQL Limitless Database 中不支援的功能

Aurora PostgreSQL Limitless Database 的要求

請務必遵循 Aurora PostgreSQL Limitless Database 的這些要求。

- Aurora PostgreSQL 無限資料庫適用於所有 ，但亞太區域 （ 台北 ） AWS 區域 除外。

> **Note**
>
> 如果您在美國東部 (維吉尼亞北部) 建立 Aurora PostgreSQL Limitless Database 叢集，則請勿在資料庫子網路群組中包含 `us-east-1e` 可用區域 (AZ)。由於資源限制，`us-east-1e` AZ 中不支援 Aurora Serverless v2 (因此也不支援 Aurora PostgreSQL Limitless Database)。

- Aurora PostgreSQL Limitless Database 僅支援 Aurora I/O-Optimized 資料庫叢集儲存組態。如需詳細資訊，請參閱Amazon Aurora 資料庫叢集的儲存組態。
- Aurora PostgreSQL 無限資料庫針對 Aurora PostgreSQL 無限資料庫使用特殊的 Aurora PostgreSQL 資料庫引擎版本：`16.X-limitless`。另請參閱可用的 Aurora PostgreSQL 無限資料庫版本
- 資料庫叢集不能有任何寫入器或讀取器資料庫執行個體。
- 您必須使用增強型監控和 Performance Insights。Performance Insights 保留時間必須至少為 1 個月 (31 天)。
- 您必須將 PostgreSQL 日誌匯出至 Amazon CloudWatch Logs。

> **Note**
>
> 增強型監控、Performance Insights 和 CloudWatch Logs 等某些必要功能會產生額外費用。如需 Aurora 定價資訊，請參閱 Aurora 定價頁面。

Aurora PostgreSQL Limitless Database 的考量事項

下列考量事項適用於 Aurora PostgreSQL Limitless Database 中的資料庫碎片群組：

- 每個資料庫叢集只能有一個資料庫碎片群組。
- 每個資料庫碎片群組最多可有五個 AWS 區域。

  因此，每個 最多可以有五個 Aurora PostgreSQL 無限制資料庫資料庫叢集 AWS 區域。如需詳細資訊，請參閱Amazon Aurora 中的配額。
- 您可以將資料庫碎片群組的容量上限設為 16–6144 個 ACU。如需容量限制高於 6144 個 ACUs，請聯絡 AWS。

  路由器和碎片初始數量取決於您在建立資料庫碎片群組時設定的容量上限。如需詳細資訊，請參閱將資料庫碎片群組容量上限與建立的路由器和碎片數量建立關聯。
- 當您修改資料庫碎片群組的容量上限時，路由器和碎片的數量不會變更。
- 請確定您建立資料庫碎片群組的資料庫子網路有足夠的可用 IP 地址，以便與資料庫碎片群組連線。每個路由器都需要一個 IP 位址，資料庫碎片群組中每個碎片最多需要三個 IP 位址。

  如需建立資料庫碎片群組時所建立之路由器數量的詳細資訊，請參閱將資料庫碎片群組容量上限與建立的路由器和碎片數量建立關聯。
- 如果您將資料庫碎片群組設為可公開存取，請務必在 VPC 中設定網際網路閘道。
- 您可以使用 SQL 函數，來分割碎片與新增路由器。
- 不支援合併碎片。
- 您無法刪除個別碎片和路由器。
- 您無法以任何方式修改碎片索引鍵 (對其執行 UPDATE 操作)，包括在資料表列中變更其值。

  若要變更碎片索引鍵，請加以刪除與重新建立。
- 支援可重複讀取、已遞交讀取和未遞交讀取的隔離層級。您無法將隔離層級設為可序列化。
- 不支援某些 SQL 命令。如需詳細資訊，請參閱Aurora PostgreSQL Limitless Database 參考。
- 不支援所有 PostgreSQL 延伸模組。如需詳細資訊，請參閱延伸模組。

- 建立碎片群組或新增碎片群組節點 (碎片或路由器) 時，系統會在資料庫叢集可用的其中一個可用區域 (AZ) 中建立那些節點。您無法為個別節點選擇特定的可用區域。
- 如果您使用 2 的運算備援 (資料庫碎片群組兩個運算待命)，請確定資料庫子網路群組至少具有三個可用 AZ。
- Aurora PostgreSQL Limitless Database 支援最多 54 個字元的碎片資料表名稱。

下列考量事項適用於 Aurora PostgreSQL Limitless Database 資料庫叢集：

- 我們建議您使用 AWS 受管政策，將資料庫和應用程式的許可限制在客戶使用案例所需的許可。如需詳細資訊，請參閱政策最佳實務。
- 建立 Aurora PostgreSQL Limitless Database 資料庫叢集時，您只能設定資料庫碎片群組的擴展參數。
- 如果需要刪除資料庫叢集，您必須先刪除資料庫碎片群組。
- Aurora PostgreSQL Limitless Database 不能是複寫來源。

Aurora PostgreSQL Limitless Database 中不支援的功能

Aurora PostgreSQL Limitless Database 中不支援下列 Aurora PostgreSQL 功能：

- Active Directory (Kerberos) 身分驗證
- Amazon DevOps Guru
- Amazon ElastiCache
- Amazon RDS 藍/綠部署
- Amazon RDS Proxy
- Aurora Auto Scaling (自動將讀取器執行個體新增至資料庫叢集)
- Aurora 全域資料庫
- Aurora Machine Learning
- Aurora 建議
- Aurora Serverless v1
- Aurora 零 ETL 整合
- AWS 備份
- AWS Lambda 整合
- AWS Secrets Manager
- Babelfish for Aurora PostgreSQL
- 複製資料庫叢集
- 自訂端點
- 資料庫活動串流
- 僅供讀取複本
- RDS 資料 API

使用 Aurora PostgreSQL Limitless Database 的先決條件

若要使用 Aurora PostgreSQL Limitless Database，您必須先執行下列任務。

主題

- 啟用資料庫碎片群組操作

啟用資料庫碎片群組操作

您必須先啟用資料庫碎片群組操作，才能建立資料庫碎片群組。

- 將下列區段新增至使用者的 IAM 角色的 IAM 政策，該使用者會存取 Aurora PostgreSQL Limitless Database：

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDBShardGroup",
            "Effect": "Allow",
            "Action": [
                "rds:CreateDBShardGroup",
                "rds:DescribeDBShardGroups",
                "rds:DeleteDBShardGroup",
                "rds:ModifyDBShardGroup",
                "rds:RebootDBShardGroup"
            ],
            "Resource": [
                "arn:aws:rds:*:*:shard-group:*",
                "arn:aws:rds:*:*:cluster:*"
            ]
        }
    ]
}
```

建立資料庫叢集，此叢集會使用 Aurora PostgreSQL Limitless Database

您可以建立使用 Aurora PostgreSQL 的 Limitless Database 版本的新 Aurora 資料庫叢集，以及將資料庫碎片群組新增至叢集。新增資料庫碎片群組時，請以 Aurora 容量單位 (ACU) 指定整個資料庫碎片群組的運算容量上限 (所有路由器和碎片的容量總和)。每個 ACU 是大約 2 GiB 的記憶體、對應的 CPU 和聯網的組合。視應用程式工作負載而定 (類似於 Aurora Serverless v2 的運作方式)，擴展可增加或減少資料庫碎片群組的容量。

主題

- 將資料庫碎片群組容量上限與建立的路由器和碎片數量建立關聯
- 建立資料庫叢集

將資料庫碎片群組容量上限與建立的路由器和碎片數量建立關聯

資料庫碎片群組中的路由器和碎片初始數量取決於您在建立資料庫碎片群組時設定的容量上限。容量上限越高，在資料庫碎片群組中建立的路由器和碎片數量就越高。

每個節點 (碎片或路由器) 都有自己的目前容量值，該值同樣以 ACU 為單位進行量測。

- 當節點目前的容量太低而無法處理負載時，Limitless Database 會將節點擴展到更高的容量。不過，當容量總和達到上限時，節點不會向上擴展。
- 當節點目前容量高於所需時，Limitless Database 會將節點縮減到更低的容量。不過，當容量總和達到下限時，節點不會縮減規模。

下表顯示 Aurora 容量單位 (ACU) 資料庫碎片群組容量上限與建立的節點 (路由器和碎片) 數量之間的關聯性。

> **Note**
>
> 這些值可能會有所變更。
> 如果您將運算備援設定為零以外的值，碎片總數將加倍或變成三倍。這會產生額外費用。
> 運算待命中的節點會向上和向下擴展到與寫入器相同的容量。您不會單獨為待命設定容量範圍。

節點總數	路由器	碎片	預設容量下限 (ACU)	容量範圍上限 (ACU)
4	2	2	16	16–400
5	2	3	20	401–500
6	2	4	24	501–600
7	3	4	28	601–700
8	3	5	32	701–800
9	3	6	36	801–900
10	4	6	40	901–1000
11	4	7	44	1001–1100
12	4	8	48	1101–1200
13	5	8	52	1201–1300
14	5	9	56	1301–1400
15	5	10	60	1401–1500
16	6	10	64	1501–1600
17	6	11	68	1601–1700
18	6	12	72	1701–1800
19	7	12	76	1801–1900
20	7	13	80	1901–2000
21	7	14	84	2001–2100
22	8	14	88	2101–2200
23	8	15	92	2201–2300

節點總數	路由器	碎片	預設容量下限 (ACU)	容量範圍上限 (ACU)
24	8	16	96	2301–6144

資料庫碎片群組的容量型動態組態上限只能在建立期間使用。修改容量上限時，路由器和碎片的數量會保持不變。如需更多詳細資訊，請參閱 變更資料庫碎片群組的容量。

您可以使用 SQL 命令，將碎片和路由器新增至資料庫碎片群組。如需詳細資訊，請參閱下列內容：

- 在資料庫碎片群組中分割碎片
- 將路由器新增至資料庫碎片群組

> **Note**
>
> 您無法刪除碎片或路由器。

建立資料庫叢集

使用下列程序建立 Aurora PostgreSQL 資料庫叢集，此叢集會使用 Aurora PostgreSQL Limitless Database。

您可以使用 AWS 管理主控台 或 AWS CLI，來建立使用 Aurora PostgreSQL Limitless Database 的資料庫叢集。您可以建立主要資料庫叢集和資料庫碎片群組。

主控台

當您使用 AWS 管理主控台 建立主要資料庫叢集時，系統也會在相同的程序中建立資料庫碎片群組。

使用主控台建立資料庫叢集

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 選擇 Create database (建立資料庫)。

   隨即顯示建立資料庫頁面。
3. 針對引擎類型，請選擇 Aurora (PostgreSQL 相容)。
4. 針對版本，選擇下列其中一項：

   - 具有 Limitless Database 的 Aurora PostgreSQL (與 PostgreSQL 16.4 相容)
   - 具有 Limitless Database 的 Aurora PostgreSQL (與 PostgreSQL 16.6 相容)
5. 對於 Aurora PostgreSQL Limitless Database：

a. 輸入資料庫碎片群組識別符。

> **Important**
>
> 建立資料庫碎片群組之後，您就無法變更資料庫叢集識別符或資料庫碎片群組識別符。

b. 針對資料庫碎片群組容量範圍：

i. 輸入容量下限 (ACU)。使用至少為 16 個 ACU 的值。

對於開發環境，預設值為 16 個 ACU。對於生產環境，預設值為 24 個 ACU。

ii. 輸入容量上限 (ACU)。使用至少為 16 個 ACU 或最多為 6144 個 ACU 的值。

對於開發環境，預設值為 64 個 ACU。對於生產環境，預設值為 384 個 ACU。

如需更多詳細資訊，請參閱 [將資料庫碎片群組容量上限與建立的路由器和碎片數量建立關聯]。

c. 對於資料庫碎片群組部署，選擇是否要為資料庫碎片群組建立待命：

- 無運算備援：為每個碎片建立沒有待命的資料庫碎片群組。這是預設值。

- 使用單一容錯移轉目標的運算備援：在不同的可用區域 (AZ) 中建立具有單一運算待命的資料庫碎片群組。
- 具有兩個容錯移轉目標的運算備援：在兩個不同的 AZ 中建立具有兩個運算待命的資料庫碎片群組。

> **Note**
>
> 如果您將運算備援設定為零以外的值，碎片資料庫執行個體總數將加倍或變成三倍。這些額外的資料庫執行個體是運算待命，可擴展和縮減至與寫入器執行個體相同的容量。您不會單獨為待命設定容量範圍。因此，ACU 用量和帳單會相應地加倍和變成三倍。若要了解運算備援產生的確切 ACU 用量，請參閱 DBShardGroup 指標中的 `DBShardGroupComputeRedundancyCapacity` 指標。

d. 選擇是否要使資料庫碎片群組變成可公開存取。

> **Note**
>
> 您無法在建立資料庫碎片群組之後修改此設定。

6. 對於連線：

a. (選用) 選取連線至 EC2 運算資源，然後選擇現有的 EC2 執行個體或建立新的執行個體。

> **Note**
>
> 如果您連線到 EC2 執行個體，則無法使資料庫碎片群組變成可公開存取。

b. 針對網路類型，選擇 IPv4 或雙堆疊模式。

c. 選擇虛擬私有雲端 (VPC) 和資料庫子網路群組，或使用預設設定。

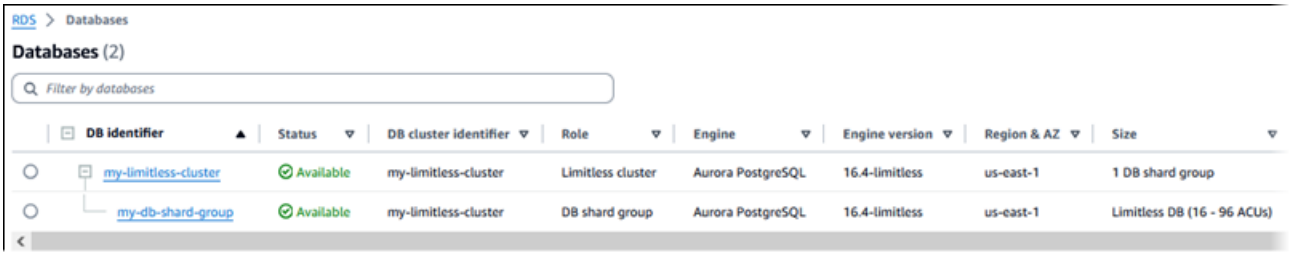
> **Note**
>
> 如果您在美國東部 (維吉尼亞北部) 區域中建立 Limitless Database 資料庫叢集，則請勿在資料庫子網路群組中包含 `us-east-1e` 可用區域 (AZ)。由於資源限制，`us-east-1e` AZ 中不支援 Aurora Serverless v2，因此也不支援 Limitless Database。

d. 選擇 VPC 安全群組 (防火牆)，或使用預設設定。

7. 對於資料庫身分驗證，選擇密碼身分驗證或密碼和 IAM 資料庫身分驗證。
8. 對於監控，請確定已選取開啟績效詳情和啟用增強型監控核取方塊。

   對於 Performance Insights，選擇至少 1 個月的保留時間。
9. 展開頁面上最後一項其他組態。
10. 對於日誌匯出，請確定已選取 PostgreSQL 日誌核取方塊。
11. 視需要指定其他設定。如需更多詳細資訊，請參閱 Aurora 資料庫叢集的設定。
12. 選擇 Create database (建立資料庫)。

建立主要資料庫叢集和資料庫碎片群組之後，其會在資料庫頁面顯示。



CLI

當您使用 AWS CLI 建立使用 Aurora PostgreSQL Limitless Database 的資料庫叢集時，請執行下列任務：

1. 建立主要資料庫叢集.
2. 建立資料庫碎片群組.

建立主要資料庫叢集

若要建立資料庫叢集，需要下列參數：

- `--db-cluster-identifier` – 資料庫叢集的名稱。
- `--engine`：資料庫叢集必須使用 `aurora-postgresql` 資料庫引擎。
- `--engine-version`：資料庫叢集必須使用其中一個資料庫引擎版本：
  - `16.4-limitless`
  - `16.6-limitless`
- `--storage-type`：資料庫叢集必須使用 `aurora-iopt1` 資料庫叢集儲存組態。
- `--cluster-scalability-type`：指定 Aurora 資料庫叢集的可擴展性模式。設為 `limitless` 時，叢集會以 Aurora PostgreSQL Limitless Database 的形式運作。設為 `standard` (預設值) 時，叢集會使用一般資料庫執行個體建立。

> **Note**
>
> 您無法在建立資料庫叢集之後修改此設定。

- `--master-username`：資料庫叢集主要使用者的名稱。
- `--master-user-password`：主要使用者密碼。
- `--enable-performance-insights`：您必須啟用 Performance Insights。
- `--performance-insights-retention-period`：Performance Insights 保留期間必須至少為 31 天。
- `--monitoring-interval`：為資料庫叢集收集增強型監控指標點之間的間隔 (秒)。此值不能是 `0`。
- `--monitoring-role-arn`：允許 RDS 將增強型監控指標傳送到 Amazon CloudWatch Logs 之 IAM 角色的 Amazon Resource Name (ARN)。

- `--enable-cloudwatch-logs-exports`：您必須將 `postgresql` 日誌匯出至 CloudWatch Logs。

下列是選用參數：

- `--db-subnet-group-name`：要與此資料庫叢集關聯之資料庫子網路群組。這也會決定與資料庫叢集相關聯的 VPC。

> **Note**
>
> 如果您在美國東部 (維吉尼亞北部) 區域中建立 Limitless Database 資料庫叢集，則請勿在資料庫子網路群組中包含 `us-east-1e` 可用區域 (AZ)。由於資源限制，`us-east-1e` AZ 中不支援 Aurora Serverless v2，因此也不支援 Limitless Database。

- `--vpc-security-group-ids`：與此資料庫叢集關聯的 VPC 安全群組清單。
- `--performance-insights-kms-key-id`：用於 Performance Insights 資料加密的 AWS KMS key 識別符。如果您未指定 KMS 金鑰，則會使用 AWS 帳戶 的預設金鑰。
- `--region`：您建立資料庫叢集所在的 AWS 區域。其必須是支援 Aurora PostgreSQL Limitless Database 的資料庫。

若要使用預設 VPC 和 VPC 安全群組，請省略 `--db-subnet-group-name` 和 `--vpc-security-group-ids` 選項。

建立主要資料庫叢集

-

```
aws rds create-db-cluster \
    --db-cluster-identifier my-limitless-cluster \
    --engine aurora-postgresql \
    --engine-version 16.6-limitless \
    --storage-type aurora-iopt1 \
    --cluster-scalability-type limitless \
    --master-username myuser \
    --master-user-password mypassword \
    --db-subnet-group-name mysubnetgroup \
    --vpc-security-group-ids sg-c7e5b0d2 \
    --enable-performance-insights \
    --performance-insights-retention-period 31 \
    --monitoring-interval 5 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/EMrole \
```

```
    --enable-cloudwatch-logs-exports postgresql
```

如需詳細資訊，請參閱 create-db-cluster。

建立資料庫碎片群組

接著，在剛建立的資料庫叢集中建立資料庫碎片群組。下列是必要參數：

- `--db-shard-group-identifier`：資料庫碎片群組的名稱。

  資料庫碎片群組識別符有下列限制：

  - 其在 AWS 帳戶 和您建立之的 AWS 區域 中必須是唯一的。
  - 必須包含 1–63 個字母、數字或連字號。
  - 第一個字元必須是字母。
  - 不能以一個連字號結尾或是連續包含兩個連字號。
  -

    Important

    建立資料庫碎片群組之後，您就無法變更資料庫叢集識別符或資料庫碎片群組識別符。
- `--db-cluster-identifier`：您要在其中建立資料庫碎片群組的資料庫叢集名稱。
- `--max-acu`：資料庫碎片群組的最大容量。其必須介於 16–6144 個 ACU 之間。如需高於 6144 個 ACU 的容量限制，請聯絡 AWS。

  路由器和碎片初始數量取決於您在建立資料庫碎片群組時設定的容量上限。容量上限越高，在資料庫碎片群組中建立的路由器和碎片數量就越高。如需更多詳細資訊，請參閱 [將資料庫碎片群組容量上限與建立的路由器和碎片數量建立關聯]。

下列是選用參數：

- `--compute-redundancy`：是否要為資料庫碎片群組建立待命。此參數可以有下列值：
  - `0`：為每個碎片建立沒有待命的資料庫碎片群組。這是預設值。
  - `1`：在不同的可用區域 (AZ) 中建立具有一個運算待命的資料庫碎片群組。
  - `2`：在兩個不同的 AZ 建立具有兩個運算待命的資料庫碎片群組。

  Note

  如果您將運算備援設定為零以外的值，碎片總數將加倍或變成三倍。這會產生額外費用。運算待命中的節點會向上和向下擴展到與寫入器相同的容量。您不會單獨為待命設定容量範圍。
- `--min-acu`：資料庫碎片群組的容量下限。其必須至少為 16 個 ACU，這是預設值。

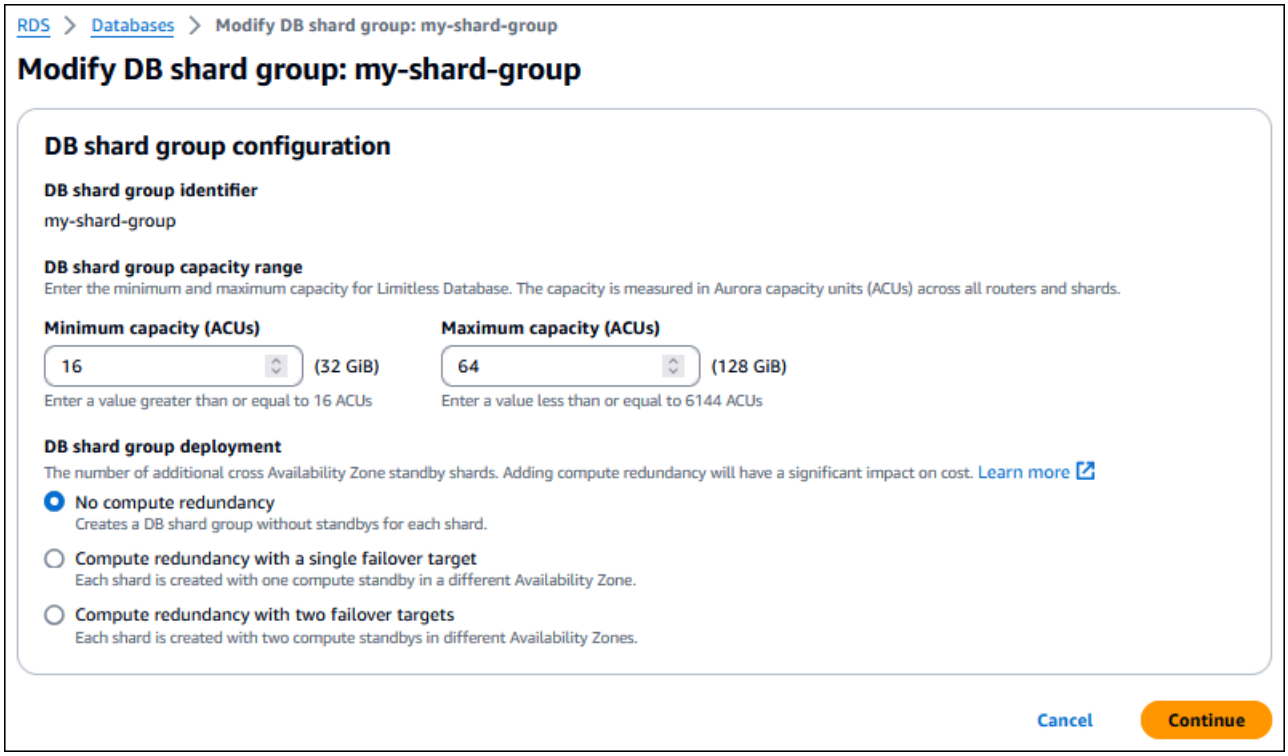
- `--publicly-accessible|--no-publicly-accessible`：是否將可公開存取的 IP 地址指派給資料庫碎片群組。資料庫碎片群組的存取權是由叢集使用的安全群組所控制。

  預設值為 `--no-publicly-accessible`。

  > **Note**
  >
  > 您無法在建立資料庫碎片群組之後修改此設定。

建立資料庫碎片群組

- ```
  aws rds create-db-shard-group \
      --db-shard-group-identifier my-db-shard-group \
      --db-cluster-identifier my-limitless-cluster \
      --max-acu 1000
  ```
  ```

使用資料庫碎片群組

您可以執行下列任務，來新增和管理 Aurora PostgreSQL Limitless Database 的資料庫碎片群組。

主題

- 連線至 Aurora PostgreSQL Limitless Database 資料庫叢集
- 尋找資料庫碎片群組中的路由器和碎片數量
- 描述資料庫碎片群組
- 重新啟動資料庫碎片群組
- 變更資料庫碎片群組的容量
- 在資料庫碎片群組中分割碎片
- 將路由器新增至資料庫碎片群組
- 刪除資料庫碎片群組
- 將資料庫碎片群組新增至現有的 Aurora PostgreSQL Limitless Database 資料庫叢集

連線至 Aurora PostgreSQL Limitless Database 資料庫叢集

若要使用 Aurora PostgreSQL Limitless Database，請連線至叢集寫入器或讀取器端點。您可以使用 `psql` 或與 PostgreSQL 搭配使用的任何其他連線公用程式：

```
$ psql -h DB_cluster_endpoint -p port_number -U database_username -d postgres_limitless
```

下列範例使用您在 CLI 中建立之資料庫叢集的端點。

```
$ psql -h my-limitless-cluster.cluster-ckifpdyyyxxx.us-east-1.rds.amazonaws.com -p 5432
 -U postgres -d postgres_limitless
```

> **Note**
>
> Aurora PostgreSQL Limitless Database 中資料庫碎片群組的預設資料庫為 `postgres_limitless`。

使用無限連線外掛程式

連線至 Aurora PostgreSQL Limitless Database 時，用戶端會使用叢集端點連線，並由 Amazon Route 53 路由至交易路由器。不過，Route 53 的負載平衡能力受到限制，而且可能在交易路由器允許不均勻的工作負載。AWS JDBC 驅動程式的無限連線外掛程式透過執行用戶端負載平衡與負載感知來解決這個問題。如需 AWS JDBC 驅動程式的詳細資訊，請參閱 使用 Amazon Web Services (AWS) JDBC 驅動程式連線至 Aurora PostgreSQL。

尋找資料庫碎片群組中的路由器和碎片數量

您可以使用下列查詢來尋找路由器和碎片的數量：

```
SELECT * FROM rds_aurora.limitless_subclusters;

 subcluster_id | subcluster_type
---------------+-----------------
 1             | router
 2             | router
 3             | shard
 4             | shard
 5             | shard
 6             | shard
```

描述資料庫碎片群組

使用 `describe-db-shard-groups` AWS CLI 命令來描述資料庫碎片群組。以下參數為選用：

- `--db-shard-group-identifier`：資料庫碎片群組的名稱。

下列範例說明特定的資料庫碎片群組。

```
aws rds describe-db-shard-groups --db-shard-group-identifier my-db-shard-group
```

此輸出看起來會與以下範例相似。

```
{
    "DBShardGroups": [
        {
            "DBShardGroupResourceId": "shardgroup-8986d309a93c4da1b1455add17abcdef",
            "DBShardGroupIdentifier": "my-shard-group",
            "DBClusterIdentifier": "my-limitless-cluster",
            "MaxACU": 1000.0,
            "ComputeRedundancy": 0,
            "Status": "available",
            "PubliclyAccessible": false,
            "Endpoint": "my-limitless-cluster.limitless-ccetp2abcdef.us-
east-1.rds.amazonaws.com"
        }
    ]
```

```
}
```

重新啟動資料庫碎片群組

有時候，例如當 `max_connections` 參數因為容量變更上限而變更時，您必須重新啟動資料庫碎片群組。

您可以使用 AWS 管理主控台 或 AWS CLI 來變更資料庫碎片群組的容量。

主控台

使用下列程序。

登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。

1. 導覽至 Databases (資料庫) 頁面。
2. 選取您要重新啟動的資料庫碎片群組。
3. 針對 Actions (動作)，選擇 Reboot (重新啟動)。
4. 選擇確認。

CLI

若要重新啟動資料庫碎片群組，請使用 `reboot-db-shard-group` AWS CLI 命令並搭配以下參數：

- `--db-shard-group-identifier`：資料庫碎片群組的名稱。

下列範例會重新啟動資料庫碎片群組。

```
aws rds reboot-db-shard-group --db-shard-group-identifier my-db-shard-group
```

變更資料庫碎片群組的容量

您可以使用 AWS 管理主控台 或 AWS CLI 來變更資料庫碎片群組的容量。

主控台

使用下列程序。

登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。

1. 導覽至 Databases (資料庫) 頁面。
2. 選取您要修改的資料庫碎片群組。
3. 在 Actions (動作) 中，選擇 Modify (修改)。

   隨即顯示修改資料庫碎片群組頁面。



4. 輸入新的容量下限 (ACU) 值，例如 **100**。
5. 輸入新的容量上限 (ACU) 值，例如 **1000**。
6. 選擇繼續。

   確認頁面隨即顯示，其中包含您的變更摘要。
7. 檢閱變更，然後選擇修改資料庫碎片群組。

CLI

使用 `modify-db-shard-group` AWS CLI 命令搭配下列參數：

- `--db-shard-group-identifier`：資料庫碎片群組的名稱。
- `--max-acu`：資料庫碎片群組的新容量上限。您可以將資料庫碎片群組的最大容量設為 16–6144 個 ACU。如需高於 6144 個 ACU 的容量限制，請聯絡 AWS。

  路由器和碎片的數量不會變更。
- `--min-acu`：資料庫碎片群組的新容量下限。其必須至少為 16 個 ACU，這是預設值。

下列 CLI 範例會將資料庫碎片群組的容量範圍變更為 100–1000 個 ACU。

```
aws rds modify-db-shard-group \
    --db-shard-group-identifier my-db-shard-group \
    --min-acu 100 \
    --max-acu 1000
```

在資料庫碎片群組中分割碎片

您可以手動將資料庫碎片群組中的碎片分割成兩個較小的碎片。這稱為使用者啟動的碎片分割。

Aurora PostgreSQL Limitless Database 也可以在碎片具有非常大量資料或使用量非常高時分割碎片。這稱為系統啟動的碎片分割。

主題

- 先決條件
- 分割碎片
- 追蹤碎片分割
- 完成碎片分割
- 取消碎片分割

先決條件

使用者啟動的碎片分割具有下列先決條件：

- 您必須擁有資料庫碎片群組。
- 資料庫碎片群組不能為空：其必須至少包含一個碎片資料表。
- 使用者必須具有 `rds_aurora_limitless_cluster_admin` 權限。`rds_superuser` 具有此權限，因此主要使用者也具有此權限。`rds_superuser` 可以將權限授予其他使用者：

```
/* Logged in as the master user or a user with rds_superuser privileges */
CREATE USER username;
GRANT rds_aurora_limitless_cluster_admin to username;
```

- 您必須知道要分割之碎片的子叢集 (節點) ID。您可以使用下列查詢取得 ID：

```
SELECT * FROM rds_aurora.limitless_subclusters;

 subcluster_id | subcluster_type
---------------+-----------------
 1             | router
 2             | router
 3             | shard
 4             | shard
 5             | shard
```

```
6             | shard
```

若要啟用系統啟動的碎片分割，請在與資料庫叢集相關聯的自訂資料庫叢集參數群組中設定下列資料庫叢集參數：

參數	Value
rds_aurora.limitless_enable_auto_scale	on
rds_aurora.limitless_auto_scale_options	split_shard 或 add_router,split_shard 。
rds_aurora.limitless_finalize_split_shard_mode	此參數決定如何完成系統啟動的碎片分割。值可為下列其中之一： • user_initiated ：您可以決定何時完成碎片分割。這是預設值。 • immediate ：碎片分割會立即完成。 如需更多詳細資訊，請參閱 完成碎片分割。 **Note** 此參數僅適用於系統啟動的碎片分割。

如需更多詳細資訊，請參閱 Amazon Aurora 資料庫叢集的資料庫叢集參數群組。

分割碎片

若要分割資料庫碎片群組中的碎片，請使用 rds_aurora.limitless_split_shard 函數。此函數會啟動以非同步方式執行的碎片分割任務。

```
SELECT rds_aurora.limitless_split_shard('subcluster_id');
```

成功提交任務後，請等待任務 ID 傳回，例如：

```
SELECT rds_aurora.limitless_split_shard('3');

    job_id
---------------
 1691300000000
(1 row)
```

> **Note**
>
> 不支援並行碎片分割操作。在啟動另一個新增項目操作之前，依序執行與完成每個操作。

追蹤碎片分割

您可以使用任務 ID 來追蹤碎片分割任務。若要描述特定任務並取得更多相關詳細資訊，請執行下列查詢：

```
SELECT * FROM rds_aurora.limitless_list_shard_scale_jobs(job_id);
```

例如：

```
SELECT * FROM rds_aurora.limitless_list_shard_scale_jobs(1691300000000);

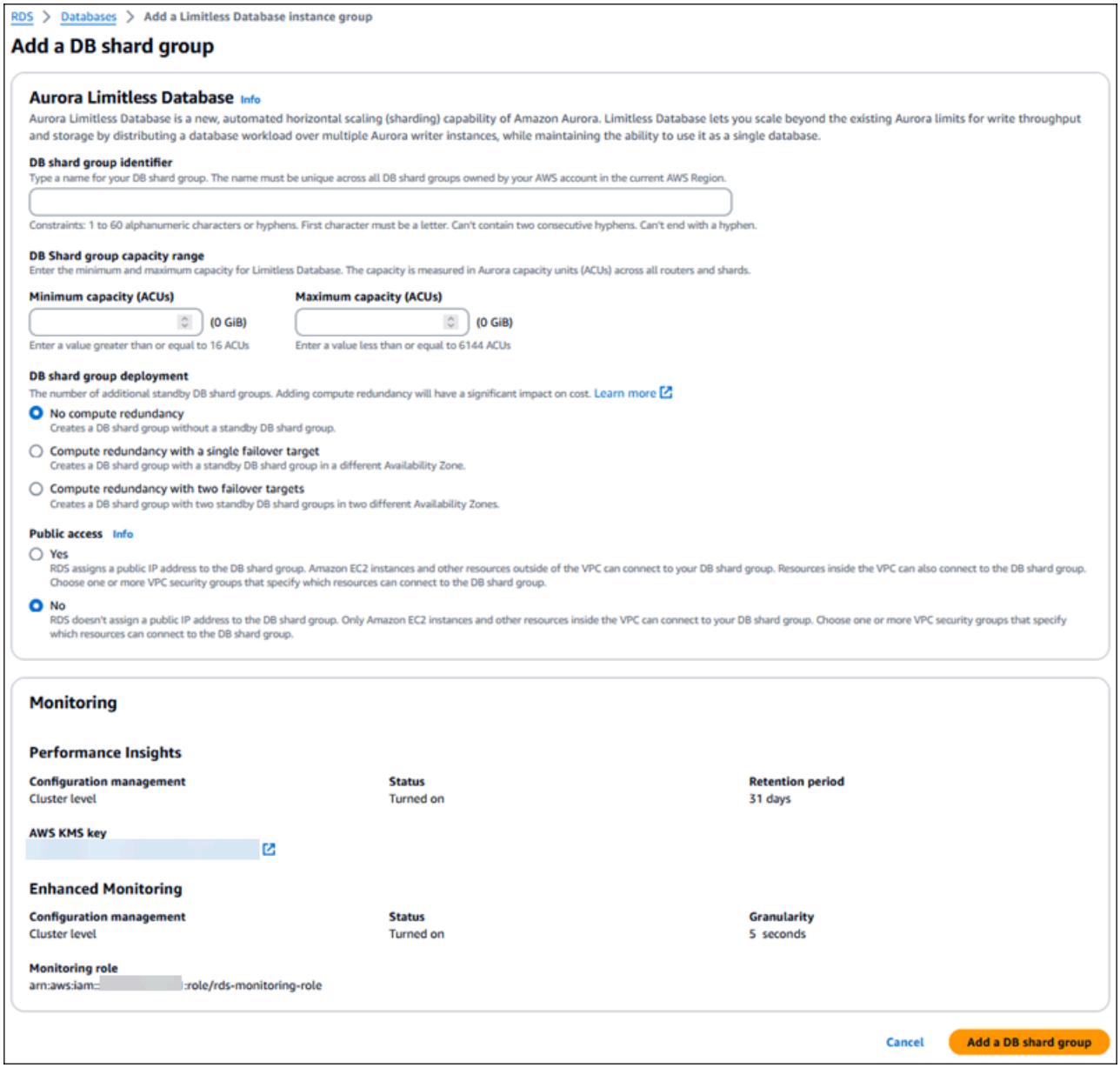
    job_id     |    action   |      job_details      | status  |    submission_time
 |                 message
---------------+-------------+-----------------------+---------
+------------------------+--------------------------------------------
 1691300000000 | SPLIT_SHARD | Split Shard 3 by User | SUCCESS | 2023-08-06 05:33:20+00
 | Scaling job succeeded.                 +
               |             |                       |         |
 | New shard instance with ID 7 was created.
(1 row)
```

當您傳遞不存在的任務作為輸入時，查詢會傳回錯誤。

```
SELECT * from rds_aurora.limitless_list_shard_scale_jobs(1691300000001);

ERROR:  no job found with the job ID provided
```

您可以使用沒有任務 ID 的相同查詢，來追蹤所有碎片分割任務的狀態，例如：

```
SELECT * FROM rds_aurora.limitless_list_shard_scale_jobs();

    job_id     |   action    |  job_details          |   status    |    submission_time
     |                  message
---------------+-------------+-----------------------+-------------
+------------------------
+-------------------------------------------------------------
 1691200000000 | SPLIT_SHARD | Split Shard 3 by User | IN_PROGRESS | 2023-08-05
 01:46:40+00 |
 1691300000000 | SPLIT_SHARD | Split Shard 4 by User | SUCCESS     | 2023-08-06
 05:33:20+00 | Scaling job succeeded. +
               |             |                       |             |
     | New shard instance with ID 7 was created.
 1691400000000 | SPLIT_SHARD | Split Shard 5 by User | FAILED      | 2023-08-07
 09:20:00+00 | Error occurred for the add shard job 1691400000000.
               |             |                       |             |
     | Retry the command. If the issue persists, contact AWS Support.
 1691500000000 | SPLIT_SHARD | Split Shard 5 by User | CANCELED    | 2023-08-07
 09:20:00+00 | Scaling job was cancelled.
(4 rows)
```

任務狀態可以是下列其中之一：

- IN_PROGRESS：碎片分割任務已提交且正在進行中。您一次只能有一個進行中的任務。
- PENDING：碎片分割任務正在等待您完成。如需更多詳細資訊，請參閱 完成碎片分割。
- CANCELLATION_IN_PROGRESS：使用者正在取消碎片分割任務。
- CANCELED：使用者或系統已成功取消碎片分割任務。
- SUCCESS：碎片分割任務已成功完成。message 欄位包含新碎片的執行個體 ID。
- FAILED：碎片分割任務失敗。message 欄位包含失敗的詳細資訊，以及可作為失敗任務後續工作的任何動作。

完成碎片分割

完成是碎片分割程序的最後一步。其會導致一些停機時間。如果您啟動碎片分割任務，則完成會在任務成功完成後立即完成。

當您使用 rds_aurora.limitless_enable_auto_scale 參數啟用系統啟動的碎片分割時，系統有時會根據工作負載分割碎片。

在這種情況下，您可以選擇是否立即完成，還是在您選擇的時間完成。您可以使用 rds_aurora.limitless_finalize_split_shard_mode 資料庫叢集參數來選擇何時發生：

- 如果您將值設為 immediate，則會立即發生。
- 如果您將值設為 user_initiated，則必須手動完成碎片分割任務。

  系統會將 RDS 事件傳送給您，以及將碎片分割任務的狀態設為 PENDING。

設定為 user_initiated 時，您可以使用 rds_aurora.limitless_finalize_split_shard 函數來完成碎片分割任務：

```
SELECT * FROM rds_aurora.limitless_finalize_split_shard(job_id);
```

> **Note**
>
> 此函數僅適用於由系統啟動的碎片分割，而非由您啟動的碎片分割。

取消碎片分割

您可以取消狀態為 IN_PROGRESS 或 PENDING 的使用者啟動或系統啟動碎片分割。您需要任務 ID 才能將其取消。

```
SELECT * from rds_aurora.limitless_cancel_shard_scale_jobs(job_id);
```

除非發生錯誤，否則不會傳回任何輸出。您可以使用任務追蹤查詢來追蹤取消。

將路由器新增至資料庫碎片群組

您可以將路由器新增至資料庫碎片群組。

主題

- 先決條件
- 新增路由器
- 追蹤路由器新增項目
- 取消路由器新增項目

先決條件

新增路由器有下列先決條件：

- 您必須擁有資料庫碎片群組。
- 使用者必須具有 rds_aurora_limitless_cluster_admin 權限。rds_superuser 具有此權限，因此主要使用者也具有此權限。rds_superuser 可以將權限授予其他使用者：

```
/* Logged in as the master user or a user with rds_superuser privileges */
CREATE USER username;
GRANT rds_aurora_limitless_cluster_admin to username;
```

> **Note**
>
> 如果您在建立資料庫碎片群組後變更 AWS 帳戶 的預設 CA 憑證，新路由器將使用新的 CA 憑證 (此憑證與現有路由器的 CA 憑證不同)。根據信任存放區，某些連線可能會失敗。

- 若要啟用系統啟動的路由器新增，請在與資料庫叢集相關聯的自訂資料庫叢集參數群組中設定下列資料庫叢集參數：

參數	Value
rds_aurora.limitless_enable_auto_scale	on
rds_aurora.limitless_auto_scale_options	add_router 或 add_router,split_shard 。

如需更多詳細資訊，請參閱 Amazon Aurora 資料庫叢集的資料庫叢集參數群組。

新增路由器

若要新增路由器，請使用 `rds_aurora.limitless_add_router` 函數。此函數會啟動以非同步方式執行的路由器新增項目任務。

```
SELECT rds_aurora.limitless_add_router();
```

成功提交任務後，請等待任務 ID 傳回，例如：

```
    job_id
---------------
 1691300000000
(1 row)
```

> **Note**
>
> 不支援並行路由器新增項目操作。在啟動另一個新增項目操作之前，依序執行操作並完成每個操作。

追蹤路由器新增項目

您可以使用任務 ID 來追蹤路由器新增項目任務。若要描述特定任務並取得更多相關詳細資訊，請執行下列查詢：

```
SELECT * FROM rds_aurora.limitless_list_router_scale_jobs(job_id);
```

例如：

```
SELECT * FROM rds_aurora.limitless_list_router_scale_jobs(1691300000000);

    job_id     |   action   |        job_details        | status  |    submission_time
   |                  message
---------------+------------+---------------------------+---------
+------------------------+--------------------------------------------
```

```
 1691300000000 | ADD_ROUTER | Add 1 new Router by User | SUCCESS | 2023-08-06
 05:33:20+00 | Scaling job succeeded.                  +
               |            |                          |         |
   | New router instance with ID 7 was created.
(1 row)
```

當您傳遞不存在的任務作為輸入時，查詢會傳回錯誤。

```
SELECT * from rds_aurora.limitless_list_router_scale_jobs(1691300000001);

ERROR:  no job found with the job ID provided
```

您可以使用不含任務 ID 的相同查詢，來追蹤所有路由器新增項目任務的狀態，例如：

```
SELECT * FROM rds_aurora.limitless_list_router_scale_jobs();

    job_id     |   action   |        job_details       |   status    |
 submission_time     |                  message
---------------+------------+--------------------------+-------------
+------------------------+-------------------------------------------
 1691200000000 | ADD_ROUTER | Add 1 new Router by User | IN_PROGRESS | 2023-08-05
 01:46:40+00 |
 1691300000000 | ADD_ROUTER | Add 1 new Router by User | SUCCESS     | 2023-08-06
 05:33:20+00 | Scaling job succeeded.                 +
               |            |                          |             |
       | New router instance with ID 7 was created.
 1691400000000 | ADD_ROUTER | Add 1 new Router by User | FAILED      | 2023-08-07
 09:20:00+00 | Error occurred for the add router job 1691400000000.
               |            |                          |             |
       | Retry the command. If the issue persists, contact AWS Support.
 1691500000000 | ADD_ROUTER | Add 1 new Router by User | CANCELED    | 2023-08-07
 09:20:00+00 | Scaling job was cancelled.
(4 rows)
```

任務狀態可以是下列其中之一：

- IN_PROGRESS：路由器新增項目任務已提交且正在進行中。您一次只能有一個進行中的任務。
- CANCELLATION_IN_PROGRESS：使用者正在取消路由器新增項目任務。
- CANCELED：使用者或系統已成功取消路由器新增項目任務。
- SUCCESS：路由器新增項目任務已成功完成。message 欄位包含新路由器的執行個體 ID。

- FAILED：路由器新增項目任務失敗。message 欄位包含失敗的詳細資訊，以及可作為失敗任務後續工作的任何動作。

> **Note**
>
> 不需要完成路由器新增項目，因此沒有 PENDING 狀態。其不會造成停機時間。

取消路由器新增項目

您可以取消 IN_PROGRESS 的路由器新增項目。您需要任務 ID 才能將其取消。

```
SELECT * from rds_aurora.limitless_cancel_router_scale_jobs(job_id);
```

除非發生錯誤，否則不會傳回任何輸出。您可以使用任務追蹤查詢來追蹤取消。

刪除資料庫碎片群組

您可以視需要刪除資料庫碎片群組。刪除資料庫碎片群組會刪除運算節點 (碎片和路由器)，但不會刪除儲存體。

主控台

登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。

1. 導覽至 Databases (資料庫) 頁面。
2. 選取您要刪除的資料庫碎片群組。
3. 對於 Actions (動作)，請選擇 Delete (刪除)。
4. 在方塊中輸入 **`delete me`**，然後選擇刪除。

資料庫碎片群組已刪除。

AWS CLI

若要刪除資料庫碎片群組，請使用 `delete-db-shard-group` AWS CLI 命令搭配下列參數：

- `--db-shard-group-identifier`：資料庫碎片群組的名稱。

下列範例會刪除您先前建立之 Aurora PostgreSQL 資料庫叢集中的資料庫碎片群組。

```
aws rds delete-db-shard-group --db-shard-group-identifier my-db-shard-group
```

將資料庫碎片群組新增至現有的 Aurora PostgreSQL Limitless Database 資料庫叢集

例如，如果您正在還原資料庫叢集或已刪除資料庫碎片群組，您可以在現有的資料庫叢集中建立資料庫碎片群組。

如需主要資料庫叢集和資料庫碎片群組要求的詳細資訊，請參閱 Aurora PostgreSQL Limitless Database 要求和考量事項。

> **Note**
>
> 每個叢集只能有一個資料庫碎片群組。
> Limitless Database 資料庫叢集必須處於 `available` 狀態，然後才能建立資料庫碎片群組。

主控台

您可以使用 AWS 管理主控台，將資料庫碎片群組新增至現有的資料庫叢集。

新增資料庫碎片群組

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 導覽至 Databases (資料庫) 頁面。
3. 選取您要新增資料庫碎片群組的 Limitless Database 叢集。
4. 針對動作，選擇新增資料庫碎片群組。



5. 輸入資料庫碎片群組識別符。

> **Important**
>
> 建立資料庫碎片群組之後，您就無法變更資料庫叢集識別符或資料庫碎片群組識別符。

6. 輸入容量下限 (ACU)。使用至少為 16 個 ACU 的值。
7. 輸入容量上限 (ACU)。使用 16–6144 個 ACU 的值。

   如需更多詳細資訊，請參閱 [將資料庫碎片群組容量上限與建立的路由器和碎片數量建立關聯](#)。
8. 對於資料庫碎片群組部署，選擇是否要為資料庫碎片群組建立待命：

- 無運算備援：為每個碎片建立沒有待命的資料庫碎片群組。這是預設值。
- 使用單一容錯移轉目標的運算備援：在不同的可用區域 (AZ) 中建立具有單一運算待命的資料庫碎片群組。
- 具有兩個容錯移轉目標的運算備援：在兩個不同的 AZ 中建立具有兩個運算待命的資料庫碎片群組。

9. 選擇是否要使資料庫碎片群組變成可公開存取。

> **Note**
>
> 您無法在建立資料庫碎片群組之後修改此設定。

10. 選擇新增資料庫碎片群組。

AWS CLI

使用 `create-db-shard-group` AWS CLI 命令以建立資料庫碎片群組。

下列是必要參數：

- `--db-cluster-identifier`：資料庫碎片群組所屬的資料庫叢集。
- `--db-shard-group-identifier`：資料庫碎片群組的名稱。

  資料庫碎片群組識別符有下列限制：

  - 其在 AWS 帳戶 和您建立之的 AWS 區域 中必須是唯一的。
  - 必須包含 1–63 個字母、數字或連字號。
  - 第一個字元必須是字母。
  - 不能以一個連字號結尾或是連續包含兩個連字號。

  > **Important**
  > 建立資料庫碎片群組之後，您就無法變更資料庫叢集識別符或資料庫碎片群組識別符。

- `--max-acu`：資料庫碎片群組的容量上限。使用 16–6144 個 ACU 的值。

下列是選用參數：

- `--compute-redundancy`：是否要為資料庫碎片群組建立待命。此參數可以有下列值：
  - `0`：為每個碎片建立沒有待命的資料庫碎片群組。這是預設值。
  - `1`：在不同的可用區域 (AZ) 中建立具有一個運算待命的資料庫碎片群組。
  - `2`：在兩個不同的 AZ 建立具有兩個運算待命的資料庫碎片群組。

  > **Note**
  > 如果您將運算備援設定為零以外的值，節點總數將加倍或變成三倍。這會產生額外費用。

- `--min-acu`：資料庫碎片群組的容量下限。其必須至少為 16 個 ACU，這是預設值。
- `--publicly-accessible|--no-publicly-accessible`：是否將可公開存取的 IP 地址指派給資料庫碎片群組。資料庫碎片群組的存取權是由叢集使用的安全群組所控制。

  預設值為 `--no-publicly-accessible`。

> **Note**
>
> 您無法在建立資料庫碎片群組之後修改此設定。

下列範例會在 Aurora PostgreSQL 資料庫叢集中建立資料庫碎片群組。

```
aws rds create-db-shard-group \
    --db-cluster-identifier my-db-cluster \
    --db-shard-group-identifier my-new-shard-group \
    --max-acu 1000
```

此輸出看起來會與以下範例相似。

```
{
    "Status": "CREATING",
    "Endpoint": "my-db-cluster.limitless-ckifpdyyyxxx.us-east-1.rds.amazonaws.com",
    "PubliclyAccessible": false,
    "DBClusterIdentifier": "my-db-cluster",
    "MaxACU": 1000.0,
    "DBShardGroupIdentifier": "my-new-shard-group",
    "DBShardGroupResourceId": "shardgroup-8986d309a93c4da1b1455add17abcdef",
    "ComputeRedundancy": 0
}
```

建立 Aurora PostgreSQL Limitless Database 資料表

在 Aurora PostgreSQL Limitless Database 中，有三種包含您資料的資料表：

- 標準：這是 Aurora PostgreSQL Limitless Database 中的預設資料表類型。您可以使用 CREATE TABLE 命令建立標準資料表，並可以對其執行資料描述語言 (DDL) 和資料處理語言 (DML) 操作。

  標準資料表不是分散式資料表。其會存放在系統內部選擇的其中一個碎片。
- 碎片：這些資料表會分佈在多個碎片中。系統會根據資料表中指定欄的值，在碎片之間分割資料。這組欄稱為碎片索引鍵。
- 參考：系統會在所有碎片複寫這些資料表。其用於不常修改的參考資料，例如產品目錄和郵遞區號。

  參考和碎片資料表之間的聯結查詢可以在碎片執行，進而消除碎片和路由器之間不必要的資料移動。

無限資料表的建立方式有兩種：

- 使用變數建立無限資料表：當您想要建立新的碎片和參考資料表時，請使用此方法。
- 將標準資料表轉換為無限資料表：當您想要將現有的標準資料表轉換為碎片和參考資料表時，請使用此方法。

我們也提供 Aurora PostgreSQL Limitless Database 的範例結構描述。

使用變數建立無限資料表

您可以設定資料表建立模式，使用變數來建立碎片和參考資料表。然後，您建立的資料表將使用此模式，直到設定不同的模式為止。

使用下列變數，來建立碎片和參考資料表：

- `rds_aurora.limitless_create_table_mode`：將此工作階段變數設為 `sharded` 或 `reference`。此變數的預設值是 `standard`。
- `rds_aurora.limitless_create_table_shard_key`：將此工作階段變數設為欄名稱陣列，以用作為碎片索引鍵。當 `rds_aurora.limitless_create_table_mode` 不是 `sharded` 時，會忽略此變數。

  將值格式化為 `untyped array literal`，類似於將常值插入陣列欄。如需詳細資訊，請參閱 PostgreSQL 文件中的陣列。
- `rds_aurora.limitless_create_table_collocate_with`：將此工作階段變數設為特定資料表名稱，以將新建立的資料表與該資料表共置。

  如果使用相同的碎片索引鍵分割二或多個資料表，您可以將那些資料表明確對齊 (共置)。將二或多個資料表共置時，會將具有相同碎片索引鍵值的資料表中的列放置在相同的碎片。共置有助於將某些操作限制為單一碎片，進而獲得更好的效能。

> **Note**
>
> 所有主要和唯一索引鍵都必須包含碎片索引鍵。這表示碎片索引鍵是主要或唯一索引鍵的子集。
> 無限資料表有一些限制。如需更多詳細資訊，請參閱 Aurora PostgreSQL Limitless Database 的 DDL 限制和其他資訊。

主題

- 使用變數建立無限資料表的範例
- Aurora PostgreSQL Limitless Database 資料表檢視

使用變數建立無限資料表的範例

下列範例示範如何使用這些變數來建立碎片和參考資料表。

使用碎片索引鍵 id 建立名為 items 的碎片資料表。

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='sharded';
SET LOCAL rds_aurora.limitless_create_table_shard_key='{"id"}';
CREATE TABLE items(id int, val int, item text);
COMMIT;
```

使用由 item_id 和 item_cat 欄組成的碎片索引鍵，建立名為 items 的碎片資料表。

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='sharded';
SET LOCAL rds_aurora.limitless_create_table_shard_key='{"item_id", "item_cat"}';
CREATE TABLE items(item_id int, item_cat varchar, val int, item text);
COMMIT;
```

使用由 item_id 和 item_cat 欄組成的碎片索引鍵，建立名為 item_description 的碎片資料表，並將其與上一個範例中的 items 資料表共置。

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='sharded';
SET LOCAL rds_aurora.limitless_create_table_shard_key='{"item_id", "item_cat"}';
SET LOCAL rds_aurora.limitless_create_table_collocate_with='items';
CREATE TABLE item_description(item_id int, item_cat varchar, color_id int);
COMMIT;
```

建立名為 colors 的參考資料表。

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='reference';
CREATE TABLE colors(color_id int primary key, color varchar);
COMMIT;
```

若要將 rds_aurora.limitless_create_table_mode 工作階段變數重設為 standard，請使用下列陳述式：

```
RESET rds_aurora.limitless_create_table_mode;
```

重設此變數之後，系統會將資料表建立為標準資料表，這是預設值。如需標準資料表的詳細資訊，請參閱將標準資料表轉換為無限資料表。

Aurora PostgreSQL Limitless Database 資料表檢視

您可以使用下列檢視，來尋找 Limitless Database 資料表的相關資訊。

rds_aurora.limitless_tables

`rds_aurora.limitless_tables` 檢視包含有關無限資料表及其類型的資訊。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_tables;

 table_gid | local_oid | schema_name | table_name  | table_status | table_type  |
 distribution_key
-----------+-----------+-------------+-------------+--------------+-------------
+------------------
         5 |     18635 | public      | standard    | active       | standard    |
         6 |     18641 | public      | ref         | active       | reference   |
         7 |     18797 | public      | orders      | active       | sharded     |
 HASH (order_id)
         2 |     18579 | public      | customer    | active       | sharded     |
 HASH (cust_id)
(4 rows)
```

rds_aurora.limitless_table_collocations

`rds_aurora.limitless_table_collocations` 檢視包含共置碎片資料表的相關資訊。例如，`orders` 和 `customers` 資料表是共置的，而且具有相同的 `collocation_id`。`users` 和 `followers` 資料表是共置的，而且具有相同的 `collocation_id`。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_table_collocations ORDER BY
 collocation_id;

 collocation_id | schema_name | table_name
----------------+-------------+------------
          16002 | public      | orders
          16002 | public      | customers
          16005 | public      | users
          16005 | public      | followers
(4 rows)
```

rds_aurora.limitless_table_collocation_distributions

`rds_aurora.limitless_table_collocation_distributions` 會顯示每個共置的金鑰分佈。

```
postgres_limitless=> SELECT * FROM
 rds_aurora.limitless_table_collocation_distributions ORDER BY collocation_id,
 lower_bound;

 collocation_id | subcluster_id |     lower_bound      |     upper_bound
----------------+---------------+----------------------+----------------------
          16002 |             6 | -9223372036854775808 | -4611686018427387904
          16002 |             5 | -4611686018427387904 |                    0
          16002 |             4 |                    0 |  4611686018427387904
          16002 |             3 |  4611686018427387904 |  9223372036854775807
          16005 |             6 | -9223372036854775808 | -4611686018427387904
          16005 |             5 | -4611686018427387904 |                    0
          16005 |             4 |                    0 |  4611686018427387904
          16005 |             3 |  4611686018427387904 |  9223372036854775807
(8 rows)
```

將標準資料表轉換為無限資料表

您可以將標準資料表轉換為碎片或參考資料表。在轉換期間，資料會從標準資料表移至分散式資料表，然後來源標準資料表會遭到刪除。使用 INSERT INTO SELECT FROM 命令移動資料。

內容

- 建立碎片資料表
- 建立共置資料表
- 建立參考資料表

建立碎片資料表

您可以在標準資料表執行 rds_aurora.limitless_alter_table_type_sharded 程序來建立碎片資料表。此程序會採用標準資料表和欄清單，然後使用欄清單作為碎片索引鍵來分配指定的資料表。程序會同步執行，以及在資料表取得 ACCESS EXCLUSIVE 鎖定。

程序成功完成後，系統會刪除來源標準資料表，而同名的碎片資料表會變成可用。

rds_aurora.limitless_alter_table_type_sharded 程序會使用以下語法：

```
postgres=> CALL rds_aurora.limitless_alter_table_type_sharded('schema.table',
 ARRAY['shard_key1', 'shard_key2', ... 'shard_keyn']);
```

此程序需要下列參數：

- schema：包含要碎片處理之資料表的資料庫結構描述。如果未指定結構描述，則程序會使用 search_path。
- table：要碎片處理的資料表。
- shard_key*n*：要用作碎片索引鍵的資料表欄陣列。

  碎片索引鍵值是字串常值，因此區分大小寫。如果碎片索引鍵包含單引號 (')，請使用另一個單引號來加以逸出。例如，如果資料表欄名為 customer's id，請使用 customer''s id 作為碎片索引鍵。不需要逸出反斜線 (\) 和雙引號 (")。

> **Note**
>
> 所有主要和唯一索引鍵都必須包含碎片索引鍵。這表示碎片索引鍵是主要或唯一索引鍵的子集。

在碎片資料表中，CHECK 限制條件不支援表達式。
如需更多詳細資訊，請參閱 限制。

建立碎片資料表

下列範例示範如何使用碎片索引鍵 customer_id 建立 customer 碎片資料表。

1. 建立標準資料表。

```
CREATE TABLE customer (customer_id INT PRIMARY KEY NOT NULL, zipcode INT, email
 VARCHAR);
```

2. 將標準資料表轉換為碎片資料表。

```
postgres=> CALL rds_aurora.limitless_alter_table_type_sharded('public.customer',
 ARRAY['customer_id']);

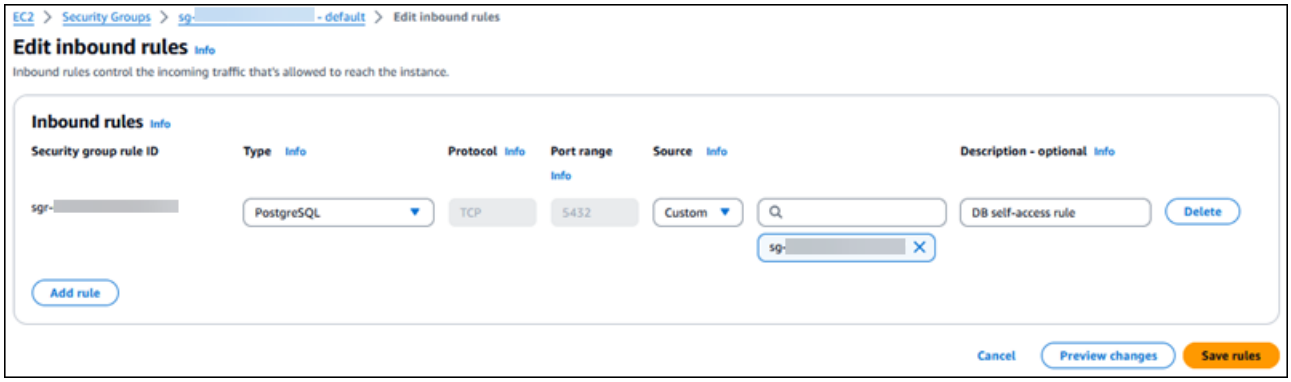
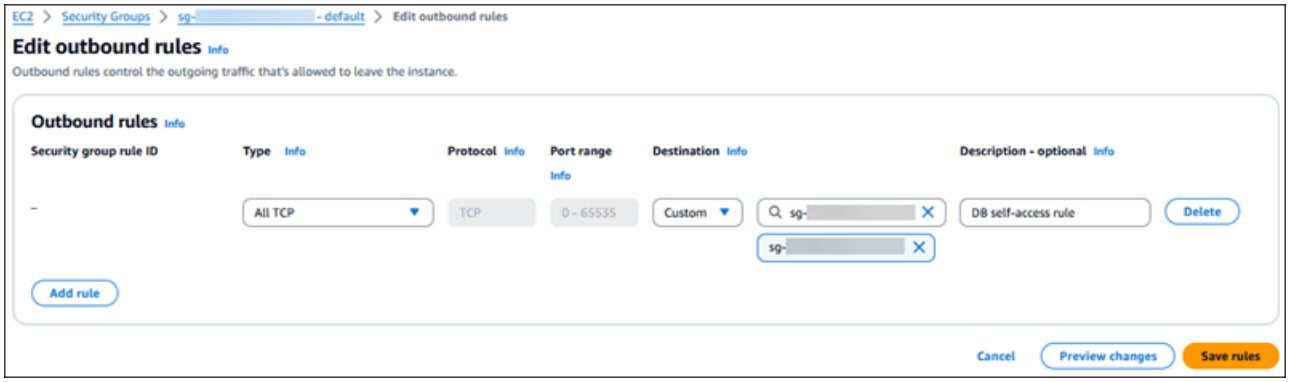
postgres=> \d

                   List of relations
 Schema |     Name     |       Type        |       Owner
--------+--------------+-------------------+--------------------
 public | customer     | partitioned table | postgres_limitless
 public | customer_fs1 | foreign table     | postgres_limitless
 public | customer_fs2 | foreign table     | postgres_limitless
 public | customer_fs3 | foreign table     | postgres_limitless
 public | customer_fs4 | foreign table     | postgres_limitless
 public | customer_fs5 | foreign table     | postgres_limitless
(6 rows)
```

建立共置資料表

如果使用相同的碎片索引鍵分割二或多個資料表，您可以將那些資料表明確對齊 (共置)。將二或多個資料表共置時，會將具有相同碎片索引鍵值的資料表中的列放置在相同的碎片。共置有助於將某些操作限制為單一碎片，進而獲得更好的效能。

您使用 rds_aurora.limitless_alter_table_type_sharded 程序搭配以下語法：

```
postgres=> CALL
 rds_aurora.limitless_alter_table_type_sharded('schema.collocated_table',
 ARRAY['shard_key1', 'shard_key2', ... 'shard_keyn'], 'schema.sharded_table');
```

此程序需要下列參數：

- schema：資料庫結構描述，其中包含要共置的資料表。如果未指定結構描述，則程序會使用 search_path。
- collocated_table：要共置的資料表。
- shard_key*n*：要用作碎片索引鍵的資料表欄陣列。

  您必須使用與原始碎片資料表相同的碎片索引鍵，包括相同的欄名稱和欄類型。
- sharded_table：您要共置 collocated_table 時所用的碎片資料表。

建立共置資料表

1. 遵循建立碎片資料表中的程序建立第一個碎片資料表。
2. 建立共置資料表的標準資料表。

   ```
   CREATE TABLE mytable2 (customer_id INT PRIMARY KEY NOT NULL, column1 INT, column2
    VARCHAR);
   ```
3. 將標準資料表轉換為共置資料表。

   ```
   postgres=> CALL rds_aurora.limitless_alter_table_type_sharded('public.mytable2',
   ARRAY['customer_id'], 'public.customer');

   postgres=> \d

                       List of relations
    Schema |     Name     |       Type        |       Owner
   --------+--------------+-------------------+--------------------
    public | customer     | partitioned table | postgres_limitless
    public | customer_fs1 | foreign table     | postgres_limitless
    public | customer_fs2 | foreign table     | postgres_limitless
    public | customer_fs3 | foreign table     | postgres_limitless
    public | customer_fs4 | foreign table     | postgres_limitless
    public | customer_fs5 | foreign table     | postgres_limitless
    public | mytable2     | partitioned table | postgres_limitless
    public | mytable2_fs1 | foreign table     | postgres_limitless
   ```

```
 public | mytable2_fs2 | foreign table     | postgres_limitless
 public | mytable2_fs3 | foreign table     | postgres_limitless
 public | mytable2_fs4 | foreign table     | postgres_limitless
 public | mytable2_fs5 | foreign table     | postgres_limitless
(12 rows)
```

建立參考資料表

您可以在標準資料表執行 rds_aurora.limitless_alter_table_type_reference 程序來建立參考資料表。此程序會將指定的資料表複寫至資料庫碎片群組中的所有碎片，並將資料表類型變更為參考。程序會同步執行，以及在資料表取得 ACCESS EXCLUSIVE 鎖定。

程序成功完成後，系統會刪除來源標準資料表，而同名的參考資料表會變成可用。

rds_aurora.limitless_alter_table_type_reference 程序會使用以下語法：

```
postgres=> CALL rds_aurora.limitless_alter_table_type_reference('schema.table');
```

預存程序需要下列參數：

- schema：包含要複寫之資料表的資料庫結構描述。如果未指定結構描述，則程序會使用 search_path。
- table：要複寫的資料表。

> **Note**
>
> 您從中建立參考資料表的標準資料表必須具有主索引鍵。
> 在參考資料表中，CHECK 限制條件不支援表達式。
> 先前的函數 (limitless_table_alter_type_reference) 已遭棄用。

建立參考資料表

下列範例示範如何建立 zipcodes 參考資料表。

1. 建立標準資料表。

```
CREATE TABLE zipcodes (zipcode INT PRIMARY KEY, details VARCHAR);
```

2. 將標準資料表轉換為參考資料表。

```
CALL rds_aurora.limitless_alter_table_type_reference('public.zipcodes');

postgres=> \d

                    List of relations
 Schema |     Name     |       Type        |       Owner
--------+--------------+-------------------+--------------------
 public | customer     | partitioned table | postgres_limitless
 public | customer_fs1 | foreign table     | postgres_limitless
 public | customer_fs2 | foreign table     | postgres_limitless
 public | customer_fs3 | foreign table     | postgres_limitless
 public | customer_fs4 | foreign table     | postgres_limitless
 public | customer_fs5 | foreign table     | postgres_limitless
 public | zipcodes     | foreign table     | postgres_limitless
(7 rows)
```

輸出會顯示 `customer` 碎片資料表和 `zipcodes` 參考資料表。

Aurora PostgreSQL Limitless Database 範例結構描述

我們為 Aurora PostgreSQL Limitless Database 提供下列範例結構描述：

- 無限電子商務範例結構描述
- 無限 pgbench

您可以使用這些結構描述快速建立範例資料庫，以及將資料載入 Aurora PostgreSQL Limitless Database 資料表。更多詳細資訊，請參閱 GitHub 儲存庫。

將資料載入 Aurora PostgreSQL Limitless Database

您可以使用 COPY 命令或使用資料載入公用程式，將資料載入 Aurora PostgreSQL Limitless Database 資料表。

> **Note**
>
> 您可以將資料載入標準、碎片和參考資料表。

內容

- 使用 COPY 命令搭配 Aurora PostgreSQL Limitless Database
  - 使用 COPY 命令將資料載入 Aurora PostgreSQL Limitless Database
    - 將資料分割為多個檔案
  - 使用 COPY 命令將 Limitless Database 資料複製到檔案
- 使用 Aurora PostgreSQL Limitless Database 資料載入公用程式
  - 限制
  - 先決條件
  - 準備來源資料庫
  - 準備目的地資料庫
  - 建立資料庫憑證
    - 建立來源資料庫憑證
    - 建立目的地資料庫憑證
  - 使用指令碼設定資料庫身分驗證和資源存取權
    - 資料載入公用程式的設定指令碼
    - 資料載入公用程式設定指令碼中的輸出
    - 清除失敗的資源
  - 手動設定資料庫身分驗證和資源存取
    - 建立客戶受管AWS KMS key
    - 建立資料庫秘密
    - 建立 IAM 角色
    - 更新客戶受管AWS KMS key
    - 新增 IAM 角色許可政策

- 載入來自 Aurora PostgreSQL 資料庫叢集或 RDS for PostgreSQL 資料庫執行個體的資料
- 監控資料載入
  - 列出資料載入任務
  - 使用任務 ID 檢視資料載入任務的詳細資訊
  - 監控 Amazon CloudWatch 日誌群組
  - 監控 RDS 事件
- 取消資料載入

使用 COPY 命令搭配 Aurora PostgreSQL Limitless Database

您可以使用 psql 公用程式中的 \copy 功能，將資料匯入 Aurora PostgreSQL Limitless Database 並從中匯出資料

使用 COPY 命令將資料載入 Aurora PostgreSQL Limitless Database

Aurora PostgreSQL Limitless Database 與 psql 公用程式中的 \copy 功能相容，可用於匯入資料。

在 Limitless Database 中 (如同在 Aurora PostgreSQL 中)，不支援下列項目：

- 直接 SSH 存取資料庫執行個體：您無法將資料檔案 (例如 .csv 格式) 複製到資料庫執行個體主機，並從檔案執行 COPY。
- 在資料庫執行個體上使用本機檔案：使用 COPY ... FROM STDIN 和 COPY ... TO STDOUT。

PostgreSQL 中的 COPY 命令具有使用本機檔案 (FROM/TO) 以及使用用戶端與伺服器 (STDIN/STDOUT) 之間連線傳輸資料的選項。如需詳細資訊，請參閱 PostgreSQL 文件中的 COPY。

PostgreSQL psql 公用程式中的 \copy 命令適用於執行 psql 用戶端所在電腦上的本機檔案。其會在您連線的遠端 (例如 Limitless Database) 伺服器上調用個別 COPY ... FROM STDIN或 COPY ... FROM STDOUT 命令。其會從本機檔案將資料讀取至 STDIN 或從 STDOUT 寫入至其中。

將資料分割為多個檔案

資料會存放在 Aurora PostgreSQL Limitless Database 中的多個碎片。若要使用 \copy 加快資料載入速度，您可以將資料分割成多個檔案。然後，透過平行執行個別 \copy 命令，為每個資料檔案獨立匯入。

例如，您有 CSV 格式的輸入資料檔案，其中包含 300 萬個要匯入的列。您可以將檔案分割成每個保留 200,000 列 (15 個區塊) 的區塊：

```
split -l200000 data.csv data_ --additional-suffix=.csv -d
```

這會透過 data_14.csv 產生檔案 data_00.csv。然後，您可以使用 15 個平行 \copy 命令匯入資料，例如：

```
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table from '/tmp/data_00.csv';" postgres_limitless &
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table FROM '/tmp/data_01.csv';" postgres_limitless &
```

```
...
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table FROM '/tmp/data_13.csv';" postgres_limitless &
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table FROM '/tmp/data_14.csv';" postgres_limitless
```

使用此技術，與使用單一 \copy 命令相比，匯入相同數量的資料大約快 10 倍。

使用 COPY 命令將 Limitless Database 資料複製到檔案

如下列範例所示，您可以使用 \copy 命令，將資料從無限制資料表複製到檔案：

```
postgres_limitless=> \copy test_table TO '/tmp/test_table.csv' DELIMITER ',' CSV
 HEADER;
```

使用 Aurora PostgreSQL Limitless Database 資料載入公用程式

Aurora 提供公用程式，可將資料直接從 Aurora PostgreSQL 資料庫叢集或 RDS for PostgreSQL 資料庫執行個體載入 Limitless Database。

您可以執行下列步驟來使用資料載入公用程式：

1. 先決條件
2. 準備來源資料庫
3. 準備目的地資料庫
4. 建立資料庫憑證
5. 下列其中一項：
   - 使用指令碼設定資料庫身分驗證和資源存取權 (建議)
   - 手動設定資料庫身分驗證和資源存取
6. 載入來自 Aurora PostgreSQL 資料庫叢集或 RDS for PostgreSQL 資料庫執行個體的資料

限制

資料載入公用程式有下列限制：

- 不支援以下資料類型：enum、ARRAY、BOX、CIRCLE、LINE、LSEG、PATH、PG_LSN、PG_SNAPSHOT、POLYGON、TSQUE 和 TXID_SNAPSHOT。
- 會在載入期間將前導零 (0) 從 VARBIT 資料類型中去除。
- 當目的地資料表有外部索引鍵時，資料遷移會失敗。
- Limitless Data Utility 支援 Amazon RDS for PostgreSQL 多可用區域資料庫叢集的下列來源組態：
  - 主要執行個體
    - 支援的模式：快照、快照_then_cdc
  - 複本執行個體
    - 支援的模式：僅限快照
      - 要求：必須啟用 hot_standby_feedback
    - 不支援：快照_then_cdc

先決條件

資料載入公用程式有下列先決條件：

- 來源資料庫使用 Aurora PostgreSQL 或 RDS for PostgreSQL 11.x 版及更新版本。
- 來源資料庫與 AWS 區域 目的地資料庫碎片群組位於相同 AWS 帳戶 和 。
- 來源資料庫叢集或資料庫執行個體處於 `available` 狀態。
- 來源資料庫和無限資料庫上的資料表具有相同的資料表名稱、欄名稱和欄資料類型。
- 來源和目的地資料表具有使用相同欄和欄順序的主索引鍵。
- 您必須擁有可連線至無限資料庫的環境，才能執行資料載入命令。可用的命令如下：
  - `rds_aurora.limitless_data_load_start`
  - `rds_aurora.limitless_data_load_cancel`
- 對於 CDC：
  - 來源資料庫和目的地資料庫碎片群組都必須使用相同的資料庫子網路群組、VPC 安全群組和資料庫連接埠。這些設定適用於與來源資料庫和資料庫碎片群組中路由器的網路連線。
  - 您必須在來源資料庫啟用邏輯複寫。來源資料庫使用者必須具有讀取邏輯複寫的權限。

準備來源資料庫

若要存取來源資料庫進行資料載入，您必須允許傳入其中的網路流量。執行以下步驟。

允許來源資料庫的網路流量

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/ec2/：// 開啟 Amazon EC2 主控台。
2. 瀏覽至安全群組頁面。
3. 針對來源資料庫叢集或執行個體所使用的安全群組，選擇安全群組 ID。

   例如，其安全群組 ID 為 `sg-056a84f1712b77926`。
4. 在傳入規則索引標籤：

   a. 選擇 Edit inbound Rules (編輯傳入規則)。

   b. 為來源資料庫叢集或執行個體新增傳入規則：

      - 連接埠範圍：來源資料庫的資料庫連接埠，通常為 5432
      - 安全群組 ID：在此範例中為 `sg-056a84f1712b77926`



5. 在傳出規則索引標籤：

   a. 選擇編輯傳出規則。

   b. 為來源資料庫叢集或執行個體新增傳出規則：

      - 資料庫連接埠：`All traffic` (包括連接埠 `0-65535`)
      - 安全群組 ID：在此範例中為 `sg-056a84f1712b77926`



6. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/vpc/：// 開啟 Amazon VPC 主控台。
7. 瀏覽至網路 ACL 頁面。
8. 如預設網路 ACL 中所述，新增預設網路 ACL 組態。

準備目的地資料庫

遵循 建立 Aurora PostgreSQL Limitless Database 資料表 中的程序，在資料庫碎片群組中建立目的地資料表。

目的地資料表必須具有與來源資料表相同的結構描述、資料表名稱和主索引鍵。

建立資料庫憑證

您必須在來源和目的地資料庫中建立資料庫使用者，並將必要的權限授予使用者。如需詳細資訊，請參閱 PostgreSQL 文件中的 CREATE USER 和 GRANT。

建立來源資料庫憑證

來源資料庫使用者會在命令中傳遞以開始載入。此使用者必須具有從來源資料庫執行複寫的權限。

1. 使用資料庫主要使用者 (或其他具有 rds_superuser 角色的使用者)，來建立具有 LOGIN 權限的來源資料庫使用者。

```
CREATE USER source_db_username WITH PASSWORD 'source_db_user_password';
```

2. 將 rds_superuser 角色授予來源資料庫使用者。

```
GRANT rds_superuser to source_db_username;
```

3. 如果您使用 full_load_and_cdc 模式，請將 rds_replication 角色授予來源資料庫使用者。授權來管理邏輯槽和利用邏輯槽來串流資料的 rds_replication 角色

```
GRANT rds_replication to source_db_username;
```

建立目的地資料庫憑證

目的地資料庫使用者必須具有寫入資料庫碎片群組中目的地資料表的許可。

1. 使用資料庫主要使用者 (或其他具有 rds_superuser 角色的使用者)，來建立具有 LOGIN 權限的目的地資料庫使用者。

```
CREATE USER destination_db_username WITH PASSWORD 'destination_db_user_password';
```

2. 將 rds_superuser 角色授予目的地資料庫使用者。

```
GRANT rds_superuser to destination_db_username;
```

使用指令碼設定資料庫身分驗證和資源存取權

設定指令碼會建立一個客戶受管 AWS KMS key、一個 AWS Identity and Access Management (IAM) 角色和兩個 AWS Secrets Manager 秘密。

請執行以下步驟來使用設定指令碼：

1. 請確定您已使用 AWS 帳戶 登入資料 AWS CLI 安裝並設定 。
2. 安裝 `jq` 命令列 JSON 處理器。如需詳細資訊，請參閱 jqlang/jq。
3. 將 data_loading_script.zip 檔案複製到您的電腦，並從中解壓縮 `data_load_aws_setup_script.sh` 檔案。
4. 編輯指令碼，將預留位置變數取代為下列項目的適當值：
   - 您的 AWS 帳戶
   - 的 AWS 區域
   - 來源資料庫憑證
   - 目的地資料庫憑證
5. 請開啟電腦上的新終端機，然後執行下列命令：

```
bash ./data_load_aws_setup_script.sh
```

資料載入公用程式的設定指令碼

我們在此處提供 `data_load_aws_setup_script.sh` 檔案的文字以供參考。

```
#!/bin/bash
# Aurora Limitless data loading - AWS resources setup script #
# Set up the account credentials in advance. #
# Update the following script variables. #

###################################
#### Start of variable section ####

ACCOUNT_ID="12-digit_AWS_account_ID"
REGION="AWS_Region"
DATE=$(date +'%m%d%H%M%S')
RANDOM_SUFFIX="${DATE}"
SOURCE_SECRET_NAME="secret-source-${DATE}"
SOURCE_USERNAME="source_db_username"
```

```
SOURCE_PASSWORD="source_db_password"
DESTINATION_SECRET_NAME="secret-destination-${DATE}"
DESTINATION_USERNAME="destination_db_username"
DESTINATION_PASSWORD="destination_db_password"
DATA_LOAD_IAM_ROLE_NAME="aurora-data-loader-${RANDOM_SUFFIX}"
TMP_WORK_DIR="./tmp_data_load_aws_resource_setup/"

#### End of variable section ####
################################

# Main logic start
echo "DATE - [${DATE}]"
echo "RANDOM_SUFFIX - [${RANDOM_SUFFIX}]"
echo 'START!'

mkdir -p $TMP_WORK_DIR

# Create the symmetric KMS key for encryption and decryption.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_key_response.txt"
aws kms create-key --region $REGION | tee $TMP_FILE_PATH
KMS_KEY_ARN=$(cat $TMP_FILE_PATH | jq -r '.KeyMetadata.Arn')
aws kms create-alias \
    --alias-name alias/"${DATA_LOAD_IAM_ROLE_NAME}-key" \
    --target-key-id $KMS_KEY_ARN \
    --region $REGION

# Create the source secret.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_source_secret_response.txt"
aws secretsmanager create-secret \
    --name $SOURCE_SECRET_NAME \
    --kms-key-id $KMS_KEY_ARN \
    --secret-string "{\"username\":\"$SOURCE_USERNAME\",\"password\":\"$SOURCE_PASSWORD
\"}" \
    --region $REGION \
    | tee $TMP_FILE_PATH
SOURCE_SECRET_ARN=$(cat $TMP_FILE_PATH | jq -r '.ARN')

# Create the destination secret.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_destination_secret_response.txt"
aws secretsmanager create-secret \
    --name $DESTINATION_SECRET_NAME \
    --kms-key-id $KMS_KEY_ARN \
    --secret-string "{\"username\":\"$DESTINATION_USERNAME\",\"password\":
\"$DESTINATION_PASSWORD\"}" \
```

```
    --region $REGION \
    | tee $TMP_FILE_PATH
DESTINATION_SECRET_ARN=$(cat $TMP_FILE_PATH | jq -r '.ARN')

# Create the RDS trust policy JSON file.
# Use only rds.amazonaws.com for RDS PROD use cases.
TRUST_POLICY_PATH="${TMP_WORK_DIR}rds_trust_policy.json"
echo '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "rds.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}' > $TRUST_POLICY_PATH

# Create the IAM role.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_iam_role_response.txt"
aws iam create-role \
    --role-name $DATA_LOAD_IAM_ROLE_NAME \
    --assume-role-policy-document "file://${TRUST_POLICY_PATH}" \
    --tags Key=assumer,Value=aurora_limitless_table_data_load \
    --region $REGION \
    | tee $TMP_FILE_PATH
IAM_ROLE_ARN=$(cat $TMP_FILE_PATH | jq -r '.Role.Arn')

# Create the permission policy JSON file.
PERMISSION_POLICY_PATH="${TMP_WORK_DIR}data_load_permission_policy.json"
permission_json_policy=$(cat &lt;&lt;EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Ec2Permission",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeNetworkInterfaces",
                "ec2:CreateNetworkInterface",
```

```
                "ec2:DeleteNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfacePermissions",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeRegions",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeNetworkAcls"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SecretsManagerPermissions",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": [
                "$SOURCE_SECRET_ARN",
                "$DESTINATION_SECRET_ARN"
            ]
        },
        {
            "Sid": "KmsPermissions",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "$KMS_KEY_ARN"
        },
        {
            "Sid": "RdsPermissions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeDBClusters",
                "rds:DescribeDBInstances"
            ],
```

```
            "Resource": "*"
        }
    ]
}
EOF
)
echo $permission_json_policy > $PERMISSION_POLICY_PATH

# Add the inline policy.
aws iam put-role-policy \
    --role-name $DATA_LOAD_IAM_ROLE_NAME \
    --policy-name aurora-limitless-data-load-policy \
    --policy-document "file://${PERMISSION_POLICY_PATH}" \
    --region $REGION

# Create the key policy JSON file.
KEY_POLICY_PATH="${TMP_WORK_DIR}data_load_key_policy.json"
key_json_policy=$(cat &lt;&lt;EOF
{
    "Id": "key-aurora-limitless-data-load-$RANDOM_SUFFIX",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::$ACCOUNT_ID:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "$IAM_ROLE_ARN"
            },
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "*"
        }
```

```
    ]
}
EOF
)
echo $key_json_policy > $KEY_POLICY_PATH

# Add the key policy.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_put_key_policy_response.txt"
sleep 10 # sleep 10 sec for IAM role ready
aws kms put-key-policy \
    --key-id $KMS_KEY_ARN \
    --policy-name default \
    --policy "file://${KEY_POLICY_PATH}" \
    --region $REGION \
    | tee $TMP_FILE_PATH

echo 'DONE!'

echo "ACCOUNT_ID : [${ACCOUNT_ID}]"
echo "REGION : [${REGION}]"
echo "RANDOM_SUFFIX : [${RANDOM_SUFFIX}]"
echo "IAM_ROLE_ARN : [${IAM_ROLE_ARN}]"
echo "SOURCE_SECRET_ARN : [${SOURCE_SECRET_ARN}]"
echo "DESTINATION_SECRET_ARN : [${DESTINATION_SECRET_ARN}]"

# Example of a successful run:
# ACCOUNT_ID : [012345678912]
# REGION : [ap-northeast-1]
# RANDOM_SUFFIX : [0305000703]
# IAM_ROLE_ARN : [arn:aws:iam::012345678912:role/aurora-data-loader-0305000703]
# SOURCE_SECRET_ARN : [arn:aws:secretsmanager:ap-
northeast-1:012345678912:secret:secret-source-0305000703-yQDtow]
# DESTINATION_SECRET_ARN : [arn:aws:secretsmanager:ap-
northeast-1:012345678912:secret:secret-destination-0305000703-5d5Jy8]

# If you want to manually clean up failed resource,
# please remove them in the following order:
# 1. IAM role.
 # aws iam delete-role-policy --role-name Test-Role --policy-name ExamplePolicy --
region us-east-1
# aws iam delete-role --role-name Test-Role --region us-east-1
# 2. Source and destination secrets.
# aws secretsmanager delete-secret --secret-id MyTestSecret --force-delete-without-
recovery --region us-east-1
```

```
# 3. KDM key.
# aws kms schedule-key-deletion --key-id arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab --pending-window-in-days 7
 --region us-east-1
```

資料載入公用程式設定指令碼中的輸出

下列範例示範成功執行指令碼的輸出。

```
% bash ./data_load_aws_setup_script.sh
DATE - [0305000703]
RANDOM_SUFFIX - [0305000703]
START!
{
    "KeyMetadata": {
        "AWSAccountId": "123456789012",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Arn": "arn:aws:kms:ap-
northeast-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "CreationDate": "2024-03-05T00:07:49.852000+00:00",
        "Enabled": true,
        "Description": "",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "Origin": "AWS_KMS",
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "KeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ],
        "MultiRegion": false
    }
}
{
    "ARN": "arn:aws:secretsmanager:ap-northeast-1:123456789012:secret:secret-
source-0305000703-yQDtow",
    "Name": "secret-source-0305000703",
    "VersionId": "a017bebe-a71b-4220-b923-6850c2599c26"
}
{
    "ARN": "arn:aws:secretsmanager:ap-northeast-1:123456789012:secret:secret-
destination-0305000703-5d5Jy8",
    "Name": "secret-destination-0305000703",
```

```
    "VersionId": "32a1f989-6391-46b1-9182-f65d242f5eb6"
}
{
    "Role": {
        "Path": "/",
        "RoleName": "aurora-data-loader-0305000703",
        "RoleId": "AROAYPX63ITQOYORQSC6U",
        "Arn": "arn:aws:iam::123456789012:role/aurora-data-loader-0305000703",
        "CreateDate": "2024-03-05T00:07:54+00:00",
        "AssumeRolePolicyDocument": {
           "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": [
                            "rds.amazonaws.com"
                        ]
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "Tags": [
            {
                "Key": "assumer",
                "Value": "aurora_limitless_table_data_load"
            }
        ]
    }
}
DONE!
ACCOUNT_ID : [123456789012]
REGION : [ap-northeast-1]
RANDOM_SUFFIX : [0305000703]
IAM_ROLE_ARN : [arn:aws:iam::123456789012:role/aurora-data-loader-0305000703]
SOURCE_SECRET_ARN : [arn:aws:secretsmanager:ap-northeast-1:123456789012:secret:secret-
source-0305000703-yQDtow]
DESTINATION_SECRET_ARN : [arn:aws:secretsmanager:ap-
northeast-1:123456789012:secret:secret-destination-0305000703-5d5Jy8]
```

清除失敗的資源

如果您想要手動清除失敗的資源，請依照下列順序將其移除：

1. IAM 角色，例如：

```
aws iam delete-role-policy \
--role-name Test-Role \
--policy-name ExamplePolicy

aws iam delete-role \
--role-name Test-Role
```

2. 來源和目的地秘密，例如：

```
aws secretsmanager delete-secret \
--secret-id MyTestSecret \
--force-delete-without-recovery
```

3. KMS 金鑰，例如：

```
aws kms schedule-key-deletion \
--key-id arn:aws:kms:us-west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab
 \
--pending-window-in-days 7
```

然後，您可以重試指令碼。

手動設定資料庫身分驗證和資源存取

設定資料庫身分驗證和資源存取的手動程序包含下列步驟：

1. 建立客戶受管AWS KMS key
2. 新增 IAM 角色許可政策
3. 建立資料庫秘密
4. 建立 IAM 角色
5. 更新客戶受管AWS KMS key

此為選用程序，執行的任務與 使用指令碼設定資料庫身分驗證和資源存取權 中相同。我們建議您使用指令碼。

建立客戶受管AWS KMS key

遵循建立對稱加密金鑰中的程序，來建立客戶受管 KMS 金鑰。如果現有金鑰符合需求，您可以使用該金鑰。

建立客戶管理的 KMS 金鑰

1. 登入 AWS 管理主控台並在 https：//https://console.aws.amazon.com/kms 開啟 AWS KMS主控台。
2. 瀏覽至客戶受管金鑰頁面。
3. 選擇建立金鑰。
4. 在設定金鑰頁面上：
   a. 對於金鑰類型，選取對稱。
   b. 對於金鑰用途，選取加密和解密。
   c. 選擇下一步。
5. 在新增標籤頁面，輸入別名 (例如 **`limitless`**)，然後選擇下一步。
6. 在定義金鑰管理許可頁面上，確定已選取允許金鑰管理員刪除此金鑰核取方塊，然後選擇下一步。
7. 在定義金鑰使用許可頁面上，選擇下一步。
8. 在 Review (檢閱) 頁面上，選擇 Finish (完成)。

   您稍後會更新金鑰政策。

記錄要在 新增 IAM 角色許可政策 中使用之 KMS 金鑰的 Amazon Resource Name (ARN)。

如需使用 AWS CLI建立客戶受管 KMS 金鑰的資訊，請參閱 create-key 和 create-alias。

建立資料庫秘密

若要允許資料載入公用程式存取來源和目的地資料庫資料表，您可以在 中建立兩個秘密AWS Secrets Manager：一個用於來源資料庫，另一個用於目的地資料庫。這些秘密會存放用於存取來源和目的地資料庫的使用者名稱和密碼。

遵循建立 AWS Secrets Manager 秘密中的程序來建立金鑰/值對秘密。

建立資料庫秘密

1. 前往以下位置開啟機密管理員控制台：https://console.aws.amazon.com/secretsmanager/。
2. 選擇 Store a new secret (存放新機密)。
3. 在選擇秘密類型頁面：
   a. 針對秘密類型，選取其他類型的秘密。
   b. 針對金鑰/值對，選擇純文字索引標籤。
   c. 輸入下列 JSON 程式碼，其中 *sourcedbreader* 和 *sourcedbpassword* 是來自 建立來源資料庫憑證 的來源資料庫使用者的憑證。

      ```
      {
          "username":"sourcedbreader",
          "password":"sourcedbpassword"
      }
      ```

   d. 針對加密金鑰，選擇您在 建立客戶受管AWS KMS key 中建立的 KMS 金鑰，例如 `limitless`。
   e. 選擇下一步。
4. 在設定秘密頁面上，輸入設定秘密 (例如 **`source_DB_secret`**)，然後選擇下一步。
5. 在設定輪換：選用頁面上，選擇下一步。
6. 在 Review (檢閱) 頁面，選擇 Store (存放)。
7. 針對目的地資料庫秘密重複此程序：
   a. 輸入下列 JSON 程式碼，其中 *destinationdbwriter* 和 *destinationdbpassword* 是來自 建立目的地資料庫憑證 的目的地資料庫使用者的憑證。

```
{
    "username":"destinationdbwriter",
    "password":"destinationdbpassword"
}
```

b. 設定秘密名稱，例如 **destination_DB_secret**。

記錄要在 新增 IAM 角色許可政策 中使用的秘密 ARN。

建立 IAM 角色

資料載入需要您提供 AWS資源的存取權。若要提供存取權，您可以遵循建立角色以將許可委派給 IAM 使用者中的程序來建立 aurora-data-loader IAM 角色。

建立 IAM 角色

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 瀏覽至角色頁面。
3. 選擇建立角色。
4. 在選取信任的實體頁面：

   a. 對於信任的實體類型，選取自訂信任政策。

   b. 輸入下列自訂信任政策的 JSON 程式碼：

      JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "rds.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
```

```
}
```

c. 選擇下一步。

5. 在 Add permissions (新增許可) 頁面上，選擇 Next (下一步)。
6. 在命名、檢閱和建立頁面：

a. 對於角色名稱，輸入 **aurora-data-loader** 或您偏好的另一個名稱。

b. 選擇新增標籤，然後輸入下列標籤：

- 索引鍵：**assumer**
- Value (值)：**aurora_limitless_table_data_load**

> **Important**
>
> Aurora PostgreSQL Limitless Database 只能擔任具有此標籤的 IAM 角色。

c. 選擇建立角色。

更新客戶受管AWS KMS key

請遵循變更金鑰政策中的程序，將 IAM 角色新增至 `aurora-data-loader` 預設金鑰政策。

將 IAM 角色新增至金鑰政策

1. 登入 AWS 管理主控台並在 https：//https://console.aws.amazon.com/kms 開啟 AWS KMS主控台。
2. 瀏覽至客戶受管金鑰頁面。
3. 選擇您在 建立客戶受管AWS KMS key 中建立的 KMS 金鑰，例如 `limitless`。
4. 在金鑰政策索引標籤，針對金鑰使用者，選擇新增。
5. 在新增金鑰使用者視窗中，選取您在 建立 IAM 角色 中建立的 IAM 角色名稱，例如 aurora-data-loader。
6. 選擇新增。

新增 IAM 角色許可政策

您必須將許可政策新增至您建立的 IAM 角色。這可讓 Aurora PostgreSQL Limitless Database 資料載入公用程式存取相關 AWS 資源，以建置網路連線並擷取來源和目的地資料庫憑證秘密。

如需詳細資訊，請參閱修改角色。

新增許可政策

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 瀏覽至角色頁面。
3. 選擇您在 建立 IAM 角色 中建立的 IAM 角色，例如 aurora-data-loader。
4. 在許可索引標籤，針對許可政策選擇新增許可，然後選擇建立內嵌政策。
5. 在指定許可頁面，選擇 JSON 編輯器。
6. 將下列範本複製並貼到 JSON 編輯器，將預留位置替換為資料庫秘密和 KMS 金鑰的 ARN。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Ec2Permission",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeNetworkInterfaces",
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfacePermissions",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeRegions",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeNetworkAcls"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SecretsManagerPermissions",
            "Effect": "Allow",
```

```
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:source_DB_secret-ABC123",
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:destination_DB_secret-456DEF"
            ]
        },        {
            "Sid": "KmsPermissions",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-east-1:123456789012:key/aa11bb22-
####-####-####-fedcba123456"
        },
        {
            "Sid": "RdsPermissions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeDBClusters",
                "rds:DescribeDBInstances"
            ],
            "Resource": "*"
        }
    ]
}
```

7. 檢查錯誤並進行更正。
8. 選擇下一步。
9. 在檢閱與建立頁面輸入政策名稱 (例如 **data_loading_policy**)，接著選擇建立政策。

載入來自 Aurora PostgreSQL 資料庫叢集或 RDS for PostgreSQL 資料庫執行個體的資料

完成資源和身分驗證設定後，請連線至叢集端點，然後從無限資料庫 (例如 `postgres_limitless`) 呼叫 `rds_aurora.limitless_data_load_start` 預存程序。無限資料庫是您要將資料遷移至其中的資料庫碎片群組上的資料庫。

此函數會在背景以非同步方式連線至命令中指定的來源資料庫、從來源讀取資料，以及將資料載入碎片。為了獲得更好的效能，系統會使用平行執行緒載入資料。函數會執行 `SELECT` 命令來讀取命令中提供之資料表的資料，以擷取時間點資料表快照。

您可以將資料載入碎片、參考和標準資料表。

您可以在 `rds_aurora.limitless_data_load_start` 呼叫中載入資料庫、結構描述或資料表層級的資料。

- 資料庫：您可以在每次呼叫中一次載入一個資料庫，資料庫中的結構描述或資料表計數沒有限制。
- 結構描述：每個呼叫最多可以載入 15 個結構描述，每個結構描述中的資料表計數沒有限制。
- 資料表：每個呼叫中最多可以載入 15 個資料表。

> **Note**
>
> 此功能不使用 Amazon RDS 快照或資料庫的時間點隔離。為了確保資料表之間的一致性，我們建議複製來源資料庫，然後指向複製的資料庫作為來源。

預存程序會使用以下語法：

```
CALL rds_aurora.limitless_data_load_start('source_type',
    'source_DB_cluster_or_instance_ID',
    'source_database_name',
    'streaming_mode',
    'data_loading_IAM_role_arn',
    'source_DB_secret_arn',
    'destination_DB_secret_arn',
    'ignore_primary_key_conflict_boolean_flag',
    'is_dry_run',
    (optional parameter) schemas/tables => ARRAY['name1', 'name2', ...]);
```

輸入參數如下：

- source_type：來源類型：aurora_postgresql 或 rds_postgresql
- source_DB_cluster_or_instance_ID：來源 Aurora PostgreSQL 資料庫叢集識別符或 RDS for PostgreSQL 資料庫執行個體識別符
- source_database_name：來源資料庫名稱，例如 *postgres*
- streaming_mode：是否包含變更資料擷取 (CDC)：full_load 或 full_load_and_cdc
- data_loading_IAM_role_arn：適用於 aurora-data-loader 的 IAM 角色 Amazon Resource Name (ARN)
- source_DB_secret_arn：來源資料庫秘密 ARN
- destination_DB_secret_arn：目的地資料庫秘密 ARN
- ignore_primary_key_conflict_boolean_flag：如果發生主索引鍵衝突，是否繼續：
  - 如果設定為 true，資料載入會忽略具有主索引鍵衝突之列的新變更。
  - 如果設定為 false，資料載入會在遇到主索引鍵衝突時覆寫目的地資料表上的現有列。
- is_dry_run：是否測試資料載入任務是否可以連線至來源和目的地資料庫：
  - 如果設為 true，則會測試連線而不載入資料
  - 如果設為 false，會載入資料
- (選用) schemas 或 tables：要載入的結構描述或資料表陣列。您可以指定下列任一項：
  - 格式為 tables => ARRAY['*schema1*.*table1*', '*schema1*.*table2*', '*schema2*.*table1*', ...] 的資料表清單
  - 格式為 schemas => ARRAY['*schema1*', '*schema2*', ...] 的結構描述清單

  如果您不包含此參數，則會遷移整個指定的來源資料庫。

輸出參數是具有訊息的任務 ID。

下列範例示範如何使用 `rds_aurora.limitless_data_load_start` 預存程序從 Aurora PostgreSQL 資料庫叢集載入資料。

```
CALL rds_aurora.limitless_data_load_start('aurora_postgresql',
    'my-db-cluster',
    'postgres',
    'full_load_and_cdc',
    'arn:aws:iam::123456789012:role/aurora-data-loader-8f2c66',
    'arn:aws:secretsmanager:us-east-1:123456789012:secret:secret-source-8f2c66-EWrr0V',
    'arn:aws:secretsmanager:us-east-1:123456789012:secret:secret-destination-8f2c66-
d04fbD',
    'true',
    'false',
    tables => ARRAY['public.customer', 'public.order', 'public.orderdetails']);

INFO: limitless data load job id 1688761223647 is starting.
```

監控資料載入

Aurora PostgreSQL Limitless Database 提供多種監控資料載入任務的方法：

- 列出資料載入任務
- 使用任務 ID 檢視資料載入任務的詳細資訊
- 監控 Amazon CloudWatch 日誌群組
- 監控 RDS 事件

列出資料載入任務

您可以連線至叢集端點，以及使用 `rds_aurora.limitless_data_load_jobs` 檢視來列出資料載入任務。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_data_load_jobs LIMIT 6;

    job_id     |  status   | message |     source_db_identifier      | source_db_name |
 full_load_complete_time |
   progress_details                                                                 |
     start_time       |   last_updated_time    |  streaming_mode   | source_engine_type
 | ignore_primary_key_conflict | is_dryrun
--------------+-----------+---------+-------------------------------+---------------
+-------------------------
+-------------------------------------------------------------------------------------------------------
+------------------------+-------------------------+------------------
+--------------------+-----------------------------+-----------
 1725697520693 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 08:48:15+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "9
 of 9 tables loaded", "COMPLETED_AT": "2024/09/07 08:48:15+00", "RECORDS_MIGRATED":
 600003}} | 2024-09-07 08:47:13+00 | 2024-09-07 08:48:15+00 | full_load         |
 aurora_postgresql  | t                          | f
 1725696114225 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 08:24:20+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "3
 of 3 tables loaded", "COMPLETED_AT": "2024/09/07 08:24:20+00", "RECORDS_MIGRATED":
 200001}} | 2024-09-07 08:23:56+00 | 2024-09-07 08:24:20+00 | full_load         |
 aurora_postgresql  | t                          | f
 1725696067630 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 08:23:45+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "6
 of 6 tables loaded", "COMPLETED_AT": "2024/09/07 08:23:45+00", "RECORDS_MIGRATED":
 400002}} | 2024-09-07 08:23:10+00 | 2024-09-07 08:23:45+00 | full_load         |
 aurora_postgresql  | t                          | f
```

```
 1725694221753 | CANCELED  |         | persistent-kdm-auto-source-01 | postgres
    |                         | {}

            | 2024-09-07 07:31:18+00 | 2024-09-07 07:51:49+00 | full_load_and_cdc |
 aurora_postgresql  | t                           | f
 1725691698210 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 07:10:51+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "1
 of 1 tables loaded", "COMPLETED_AT": "2024/09/07 07:10:51+00", "RECORDS_MIGRATED":
 100000}} | 2024-09-07 07:10:42+00 | 2024-09-07 07:10:52+00 | full_load         |
 aurora_postgresql  | t                           | f
 1725691695049 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 07:10:48+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "1
 of 1 tables loaded", "COMPLETED_AT": "2024/09/07 07:10:48+00", "RECORDS_MIGRATED":
 100000}} | 2024-09-07 07:10:41+00 | 2024-09-07 07:10:48+00 | full_load         |
 aurora_postgresql  | t                           | f
(6 rows)
```

任務記錄會在 90 天後刪除。

使用任務 ID 檢視資料載入任務的詳細資訊

如果知道任務 ID，您就可以連線至叢集端點，以及使用 `rds_aurora.limitless_data_load_job_details` 檢視來查看該資料載入任務的詳細資訊 (包括資料表名稱、任務狀態和載入的列數)。您可以在對資料載入開始函數的回應中，或從 `rds_aurora.limitless_data_load_jobs` 檢視取得任務 ID。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_data_load_job_details WHERE
 job_id='1725696114225';

job_id        | destination_table_name | destination_schema_name | start_time
      | status    | full_load_rows | full_load_total_rows | full_load_complete_time |
 cdc_insert | cdc_update | cdc_delete
--------------+------------------------+-------------------------
+-------------------------+-----------+----------------+---------------------
+-------------------------+------------+------------+------------
1725696114225 | standard_1             | public                  | 2024-09-07
 08:23:57+00 | COMPLETED | 100000         | 100000               | 2024-09-07
 08:24:08+00  | 0          | 0          | 0
1725696114225 | standard_2             | public                  | 2024-09-07
 08:24:08+00 | COMPLETED | 100000         | 100000               | 2024-09-07
 08:24:17+00  | 0          | 0          | 0
```

```
1725696114225 | standard_3             | public                  | 2024-09-07
 08:24:18+00 | COMPLETED | 1             | 1                       | 2024-09-07
 08:24:20+00  | 0          | 0          | 0
1725696114225 | standard_4             | public                  | 2024-09-07
 08:23:58+00 | PENDING   | 0             | 0                       |
    | 0          | 0          | 0
(4 rows)
```

任務記錄會在 90 天後刪除。

監控 Amazon CloudWatch 日誌群組

資料載入任務狀態變更為 RUNNING 後，您就可以使用 Amazon CloudWatch Logs 檢查執行時期進度。

監控 CloudWatch 日誌串流

簽署 AWS 管理主控台 並開啟位於 https://console.aws.amazon.com/cloudwatch/ 的 CloudWatch 主控台。

1. 瀏覽至日誌，然後日誌群組。
2. 選擇 /aws/rds/aurora-limitless-database 日誌群組。
3. 依 job_id 搜尋資料載入任務的日誌串流。

   日誌串流具有 Data-Load-Job-***job_id*** 模式。
4. 選擇日誌串流，即可查看日誌事件。

每個日誌串流都會顯示事件，其中包含任務狀態和載入 Aurora PostgreSQL Limitless Database 目的地資料表的列數。如果資料載入任務失敗，也會建立錯誤日誌，其中顯示失敗狀態和原因。

任務記錄會在 90 天後刪除。

監控 RDS 事件

資料載入任務也會發佈 RDS 事件，例如當任務成功、失敗或取消時。您可以從目的地資料庫檢視事件。

如需更多詳細資訊，請參閱 資料庫碎片群組事件。

取消資料載入

若要取消資料載入任務，請呼叫 rds_aurora.limitless_data_load_cancel 預存程序，然後將任務 ID 當作輸入參數。您可以在啟動特定資料載入任務的資料庫碎片群組中從相同資料庫呼叫此預存程序。例如：

```
CALL rds_aurora.limitless_data_load_cancel(12345);

INFO: limitless data load job with id 12345 is canceling without rollback.
```

您無法取消不存在或未在相同資料庫碎片群組中執行的資料載入任務。

如回應所示，Aurora PostgreSQL Limitless Database 資料載入公用程式會將載入的資料保留在目的地資料表中，而不會復原。如果您不想保留載入的資料，我們建議您截斷目的地資料表。

查詢 Aurora PostgreSQL Limitless Database

Aurora PostgreSQL Limitless Database 與查詢的 PostgreSQL 語法相容。您可以使用 `psql` 或任何其他可與 PostgreSQL 搭配使用的連線公用程式，來查詢 Limitless Database。如 連線至 Aurora PostgreSQL Limitless Database 資料庫叢集 所示，若要執行查詢，您可以連線到無限端點。

Aurora PostgreSQL Limitless Database 中支援所有 PostgreSQL `SELECT` 查詢。不過，查詢會在兩層執行：

1. 用戶端將查詢傳送至其中的路由器
2. 資料所在的碎片

效能取決於資料庫的查詢方式，讓資料庫能夠同時在不同碎片實現不同查詢的高度同時處理。會先在分散式交易層 (路由器) 中剖析查詢。在規劃查詢執行之前，有一個分析階段來識別參與查詢之所有關係的位置。如果所有關係都是在相同碎片或參考資料表具有篩選碎片索引鍵的碎片資料表，則會略過路由器層的查詢規劃，並完全向下推送至碎片以進行規劃和執行。此程序可減少不同節點 (路由器和碎片) 之間的往返次數，並在大多數情況產生更好的效能。如需更多詳細資訊，請參閱 Aurora PostgreSQL Limitless Database 中的單一碎片查詢。

> **Note**
>
> 在笛卡爾乘積 (交叉聯結) 之類的某些情況，透過從碎片分開擷取資料查詢效能會更好。

如需查詢執行計劃的詳細資訊，請參閱 Aurora PostgreSQL Limitless Database 參考中的 EXPLAIN。如需查詢的一般資訊，請參閱 PostgreSQL 文件中的查詢。

主題

- Aurora PostgreSQL Limitless Database 中的單一碎片查詢
- Aurora PostgreSQL Limitless Database 中的分散式查詢
- Aurora PostgreSQL Limitless Database 中 PostgreSQL 日誌中的分散式查詢追蹤
- Aurora PostgreSQL Limitless Database 中的分散式死鎖

Aurora PostgreSQL Limitless Database 中的單一碎片查詢

單一碎片查詢是可以直接在碎片執行的查詢，同時維護 SQL ACID 語意。當路由器上的查詢規劃器遇到這類查詢時，規劃器會偵測到該查詢，並繼續將整個 SQL 查詢向下推送至對應的碎片。

此最佳化可減少從路由器到碎片的網路往返次數，進而改善效能。目前針對 INSERT、SELECT、UPDATE 和 DELETE 查詢執行此最佳化。

主題

- 單一碎片查詢範例
- 單一碎片查詢的限制
- 完整 (明確) 聯結
- 設定作用中碎片索引鍵

單一碎片查詢範例

在下列範例中，我們有碎片資料表 customers (其中碎片索引鍵為 customer_id 和參考資料表為 zipcodes)。

SELECT

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) SELECT * FROM customers WHERE
 customer_id = 100;

                        QUERY PLAN
-----------------------------------------------------------
 Foreign Scan
   Output: customer_id, other_id, customer_name, balance
   Remote SQL:  SELECT customer_id,
     other_id,
     customer_name,
     balance
    FROM public.customers
   WHERE (customer_id = 100)
 Single Shard Optimized
(9 rows)
```

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) SELECT * FROM orders
    LEFT JOIN zipcodes ON orders.zipcode_id = zipcodes.zipcode_id
    WHERE customer_id = 11;

                                              QUERY PLAN

-----------------------------------------------------------------------------------------------
 Foreign Scan
```

```
   Output: customer_id, order_id, zipcode_id, customer_name, balance,
 zipcodes.zipcode_id, zipcodes.city
   Remote SQL:  SELECT orders.customer_id,
     orders.order_id,
     orders.zipcode_id,
     orders.customer_name,
     orders.balance,
     zipcodes.zipcode_id,
     zipcodes.city
    FROM (public.orders
      LEFT JOIN public.zipcodes ON ((orders.zipcode_id = zipcodes.zipcode_id)))
   WHERE (orders.customer_id = 11)
 Single Shard Optimized
(13 rows)
```

INSERT

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) INSERT INTO customers
    (customer_id, other_id, customer_name, balance)
    VALUES (1, 10, 'saikiran', 1000);

                      QUERY PLAN
------------------------------------------------------
 Insert on public.customers
   ->  Result
         Output: 1, 10, 'saikiran'::text, '1000'::real
 Single Shard Optimized
(4 rows)
```

UPDATE

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) UPDATE orders SET balance = balance +
 100
    WHERE customer_id = 100;

                                        QUERY PLAN

-----------------------------------------------------------------------------------------
 Update on public.orders
   Foreign Update on public.orders_fs00002 orders_1
   ->  Foreign Update
         Remote SQL:  UPDATE public.orders SET balance = (balance + (100)::double
 precision)
```

```
   WHERE (customer_id = 100)
 Single Shard Optimized
(6 rows)
```

DELETE

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) DELETE FROM orders
    WHERE customer_id = 100 and balance = 0;

                              QUERY PLAN
-----------------------------------------------------------------------
 Delete on public.orders
   Foreign Delete on public.orders_fs00002 orders_1
   ->  Foreign Delete
         Remote SQL:  DELETE FROM public.orders
   WHERE ((customer_id = 100) AND (balance = (0)::double precision))
 Single Shard Optimized
(6 rows)
```

單一碎片查詢的限制

單一碎片查詢有下列限制：

函數

如果單一碎片查詢包含函數，則只有在下列其中一個條件適用時，查詢才符合單一碎片最佳化的資格：

- 函數是不可變的。如需更多詳細資訊，請參閱 函數波動。
- 函數是可變的，但已在 rds_aurora.limitless_distributed_functions 檢視中註冊。如需更多詳細資訊，請參閱 函數分佈。

檢視

如果查詢包含一或多個檢視，如果查詢有下列其中一個條件，則會停用查詢的單一碎片最佳化：

- 任何檢視都有 security_barrier 屬性。
- 查詢中使用的物件需要多個使用者權限。例如，查詢包含兩個檢視，而檢視是在兩個不同的使用者下執行。

```
CREATE VIEW v1 AS SELECT customer_name FROM customers c WHERE c.customer_id =  1;
```

```
CREATE VIEW v2 WITH (security_barrier) AS SELECT customer_name FROM customers c
 WHERE c.customer_id =  1;

postgres_limitless=> EXPLAIN VERBOSE SELECT * FROM v1;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
   Output: customer_name
   Remote Plans from Shard postgres_s3:
         Seq Scan on public.customers_ts00001 c  (cost=0.00..24.12 rows=6 width=32)
           Output: c.customer_name
           Filter: (c.customer_id = 1)
         Query Identifier: -6005737533846718506
   Remote SQL:  SELECT customer_name
    FROM ( SELECT c.customer_name
             FROM public.customers c
            WHERE (c.customer_id = 1)) v1
 Query Identifier: -5754424854414896228
(12 rows)


postgres_limitless=> EXPLAIN VERBOSE SELECT * FROM v2;
                                       QUERY PLAN
-----------------------------------------------------------------------------------------
 Foreign Scan on public.customers_fs00001 c  (cost=100.00..128.41 rows=7 width=32)
   Output: c.customer_name
   Remote Plans from Shard postgres_s3:
         Seq Scan on public.customers_ts00001 customers  (cost=0.00..24.12 rows=6
 width=32)
           Output: customers.customer_name
           Filter: (customers.customer_id = 1)
         Query Identifier: 4136563775490008117
   Remote SQL: SELECT customer_name FROM public.customers WHERE ((customer_id = 1))
 Query Identifier: 5056054318010163757
(9 rows)
```

PREPARE 和 EXECUTE 陳述式

Aurora PostgreSQL Limitless Database 支援預備 SELECT、UPDATE 和 DELETE 陳述式的單一碎片最佳化。

不過，如果您針對 PREPARE 和 EXECUTE 使用預備陳述式，並將 plan_cache_mode 設為 'force_generic_plan'，則查詢規劃器會拒絕該查詢的單一碎片最佳化。

PL/pgSQL

具有 PL/pgSQL 變數的查詢會以隱含預備的陳述式執行。如果查詢包含任何 PL/pgSQL 變數，則查詢規劃器會拒絕單一碎片最佳化。

如果陳述式不包含任何 PL/pgSQL 變數，則 PL/pgSQL 區塊中會支援最佳化。

完整 (明確) 聯結

單一碎片最佳化是以分割區消除為基礎。PostgreSQL 最佳化工具會根據常數條件來消除分割區。如果 Aurora PostgreSQL Limitless Database 發現所有剩餘的分割區和資料表都位於相同的碎片，則會標記符合單一碎片最佳化資格的查詢。所有篩選條件都必須明確，分割區消除才能運作。Aurora PostgreSQL Limitless Database 無法在陳述式中每個碎片資料表的碎片索引鍵上，消除不含一或多個聯結述詞或篩選述詞的分割區。

假設我們已根據 `customer_id` 欄分割 `customers`、`orders` 和 `order_details` 資料表。在此結構描述中，應用程式會嘗試將客戶的所有資料保留在單一碎片。

請考處下列查詢：

```
SELECT * FROM
    customers c, orders o, order_details od
WHERE c.customer_id = o.customer_id
    AND od.order_id = o.order_id
    AND c.customer_id = 1;
```

此查詢會擷取客戶 (`c.customer_id = 1`) 的所有資料。此客戶的資料位於單一碎片，但 Aurora PostgreSQL Limitless Database 不會將此查詢視為單一碎片查詢。查詢的最佳化工具程序如下所示：

1. 根據下列條件，最佳化工具可以消除 `customers` 和 `orders` 的分割區：

   ```
   c.customer_id = 1
   c.customer_id = o.customer_id
   o.customer_id =  1 (transitive implicit condition)
   ```

2. 最佳化工具無法消除 `order_details` 的任何分割區，因為資料表上沒有任何常數條件。
3. 最佳化工具的結論是已從 `order_details` 讀取所有分割區。因此，查詢無法符合單一碎片最佳化的資格。

為了將此設為單一碎片查詢，我們新增下列明確聯結條件：

```
o.customer_id = od.customer_id
```

變更的查詢如下所示：

```
SELECT * FROM
    customers c, orders o,  order_details od
WHERE c.customer_id = o.customer_id
     AND o.customer_id = od.customer_id
     AND od. order_id = o. order_id
 AND c.customer_id =  1;
```

現在最佳化工具可以消除 order_details 的分割區。新查詢會成為單一碎片查詢，並符合最佳化的資格。

設定作用中碎片索引鍵

此功能可讓您在查詢資料庫時設定單一碎片索引鍵，導致所有 SELECT 和 DML 查詢都附加在碎片索引鍵作為常數述詞。如果您已遷移至 Aurora PostgreSQL Limitless Database，並將碎片索引鍵新增至資料表以將結構描述去正規化，則此功能非常有用。

您可以自動將碎片索引鍵述詞附加至現有的 SQL 邏輯，而無需變更查詢的語意。附加作用中碎片索引鍵述詞只會針對相容的資料表執行。

作用中碎片索引鍵功能會使用 rds_aurora.limitless_active_shard_key 變數，其具有下列語法：

```
SET [session | local] rds_aurora.limitless_active_shard_key = '{"col1_value",
 "col2_value", ...}';
```

作用中碎片索引鍵和外部索引鍵的一些考量：

- 如果父資料表和子資料表共置，且外部索引鍵是碎片索引鍵的超集，則碎片資料表可能會有外部索引鍵限制條件。
- 碎片資料表對參考資料表可能有外部索引鍵限制條件。
- 參考資料表對另一個參考資料表可能有外部索引鍵限制條件。

假設我們有 customers 資料表，該資料表是根據 customer_id 欄經過碎片處理。

```
BEGIN;
SET local rds_aurora.limitless_create_table_mode='sharded';
SET local rds_aurora.limitless_create_table_shard_key='{"customer_id"}';
CREATE TABLE customers(customer_id int PRIMARY KEY, name text , email text);
COMMIT;
```

使用作用中碎片索引鍵集時，查詢具有下列轉換。

SELECT

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
SELECT * FROM customers;

-- This statement is changed to:
SELECT * FROM customers WHERE customer_id = '123'::int;
```

INSERT

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
INSERT INTO customers(name, email) VALUES('Alex', 'alex@example.com');

-- This statement is changed to:
INSERT INTO customers(customer_id, name, email) VALUES('123'::int, 'Alex',
 'alex@example.com');
```

UPDATE

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
UPDATE customers SET email = 'alex_new_email@example.com';

-- This statement is changed to:
UPDATE customers SET email = 'alex_new_email@example.com' WHERE customer_id =
 '123'::int;
```

DELETE

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
DELETE FROM customers;

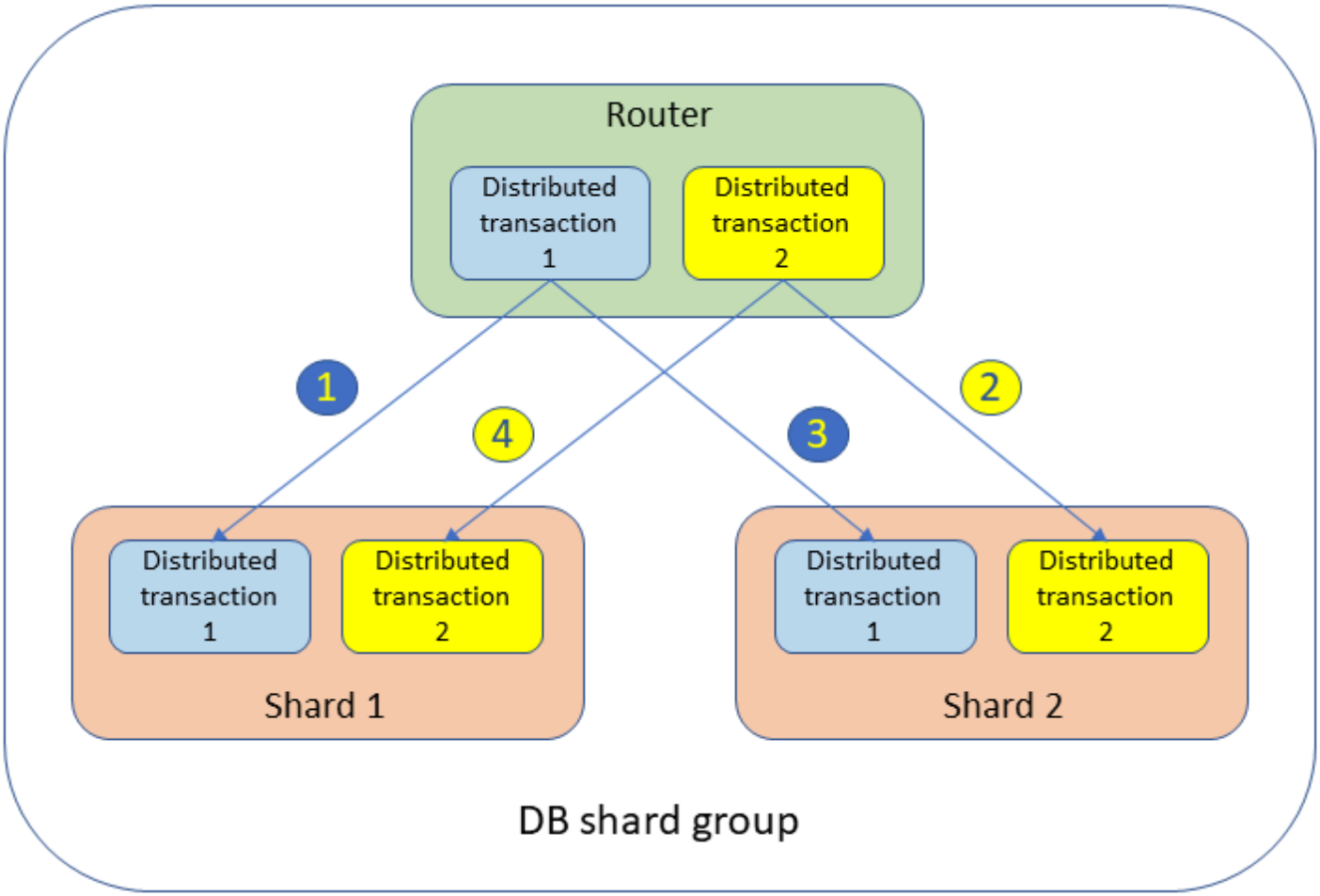
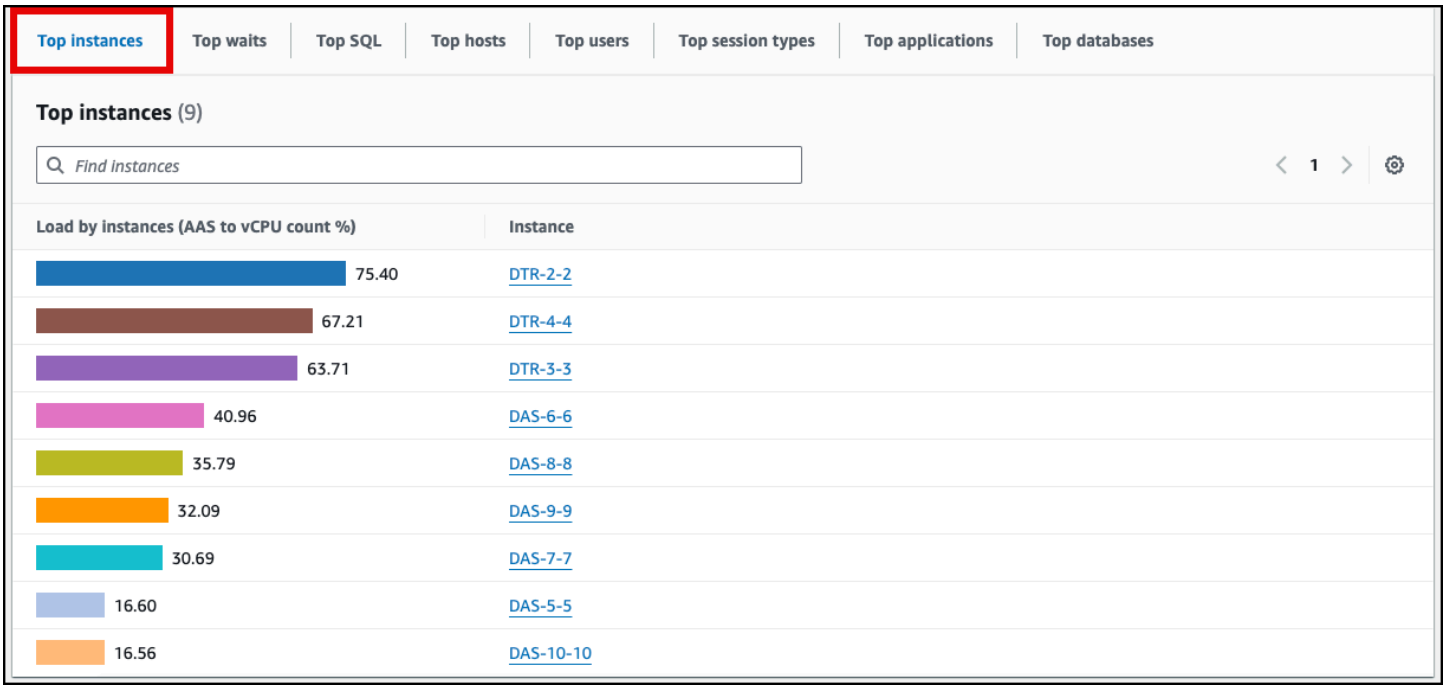
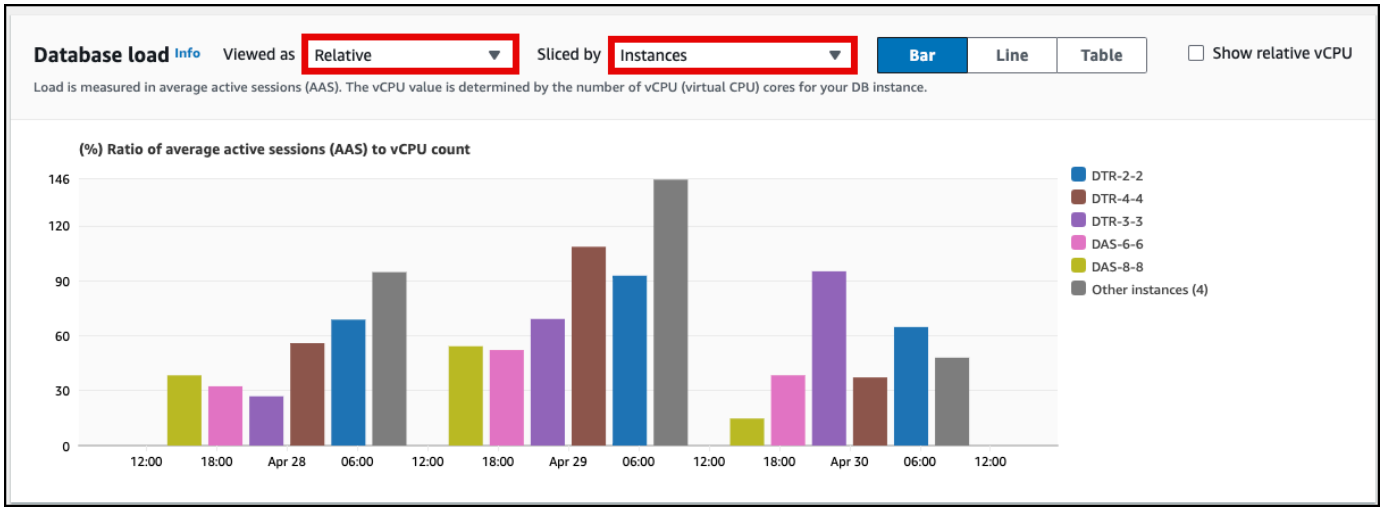
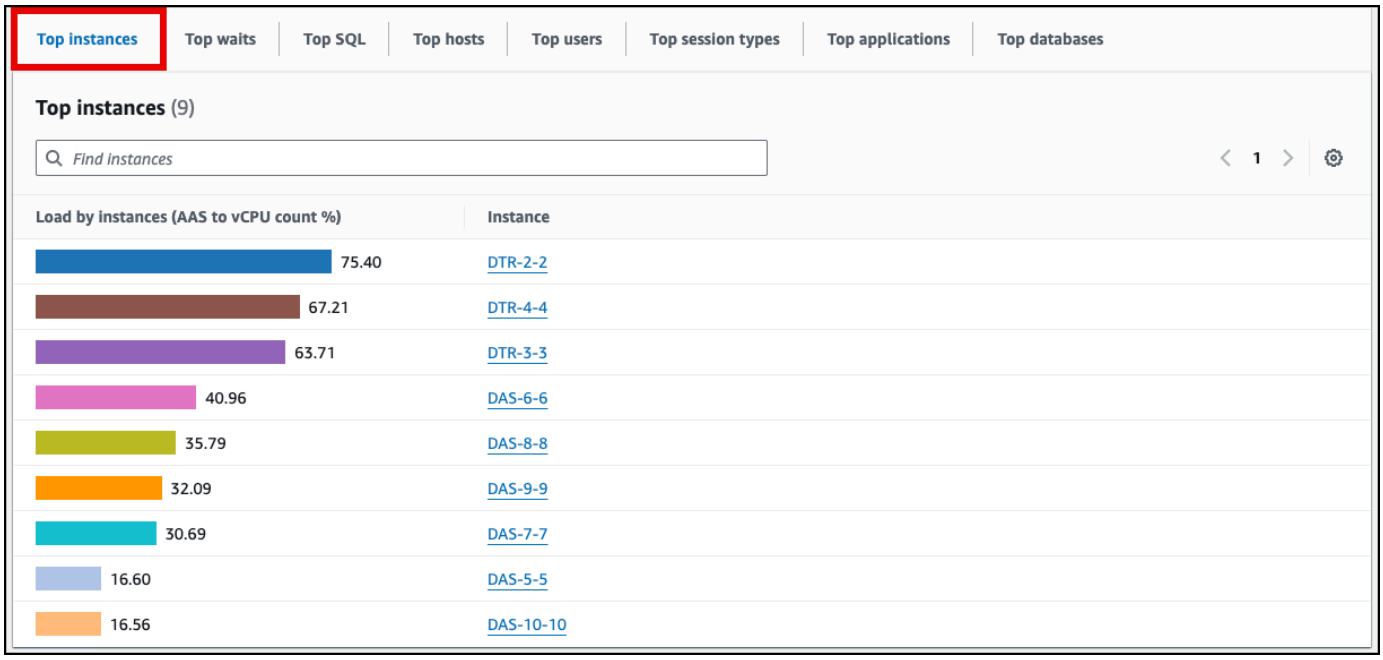
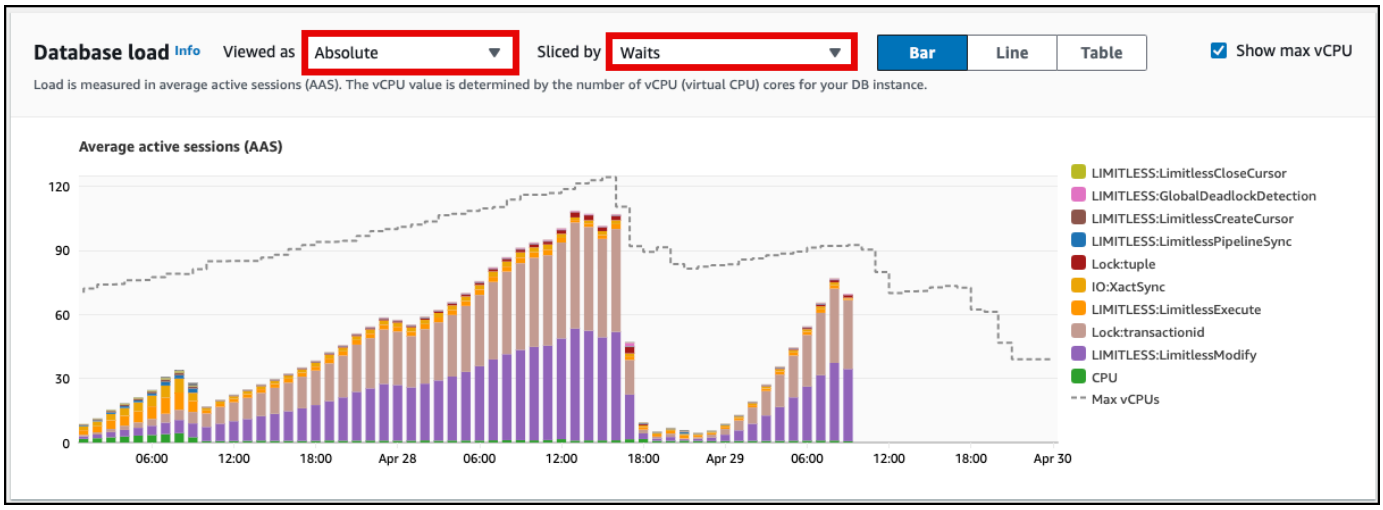
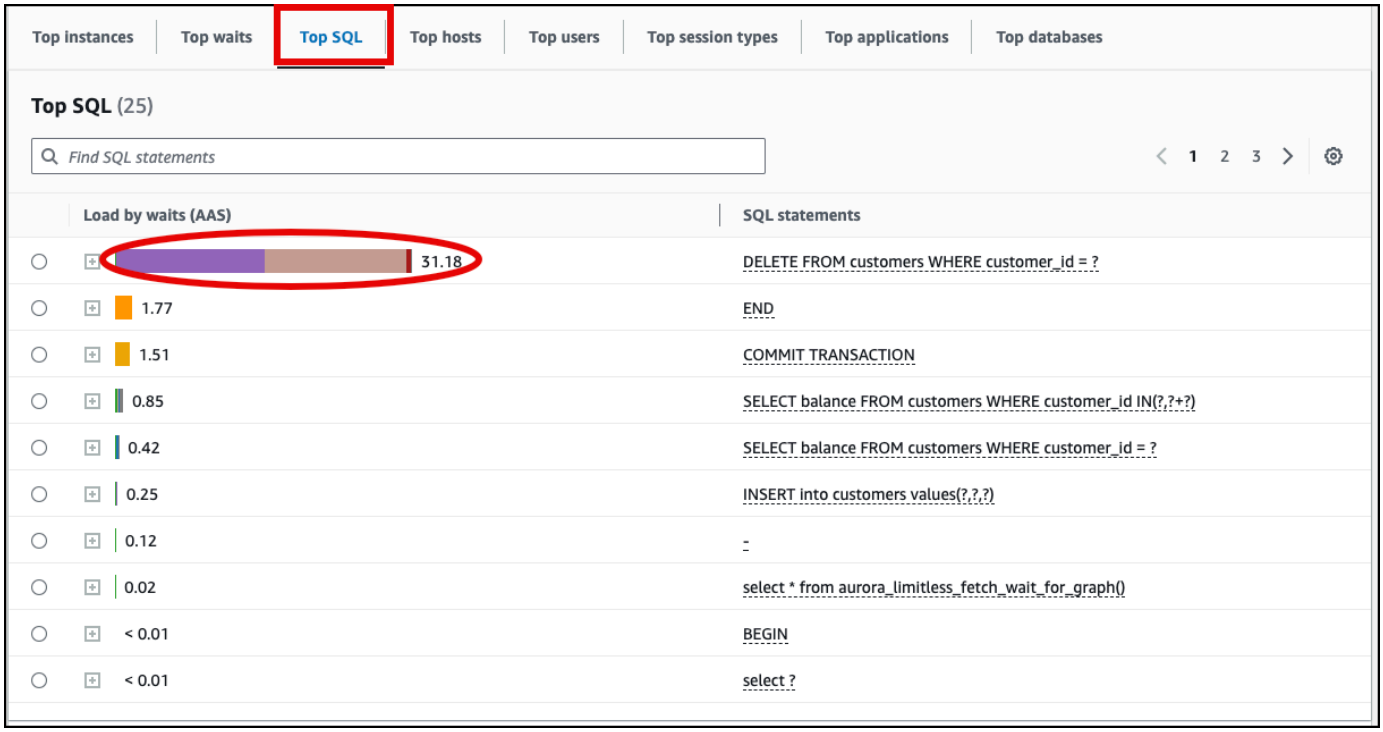
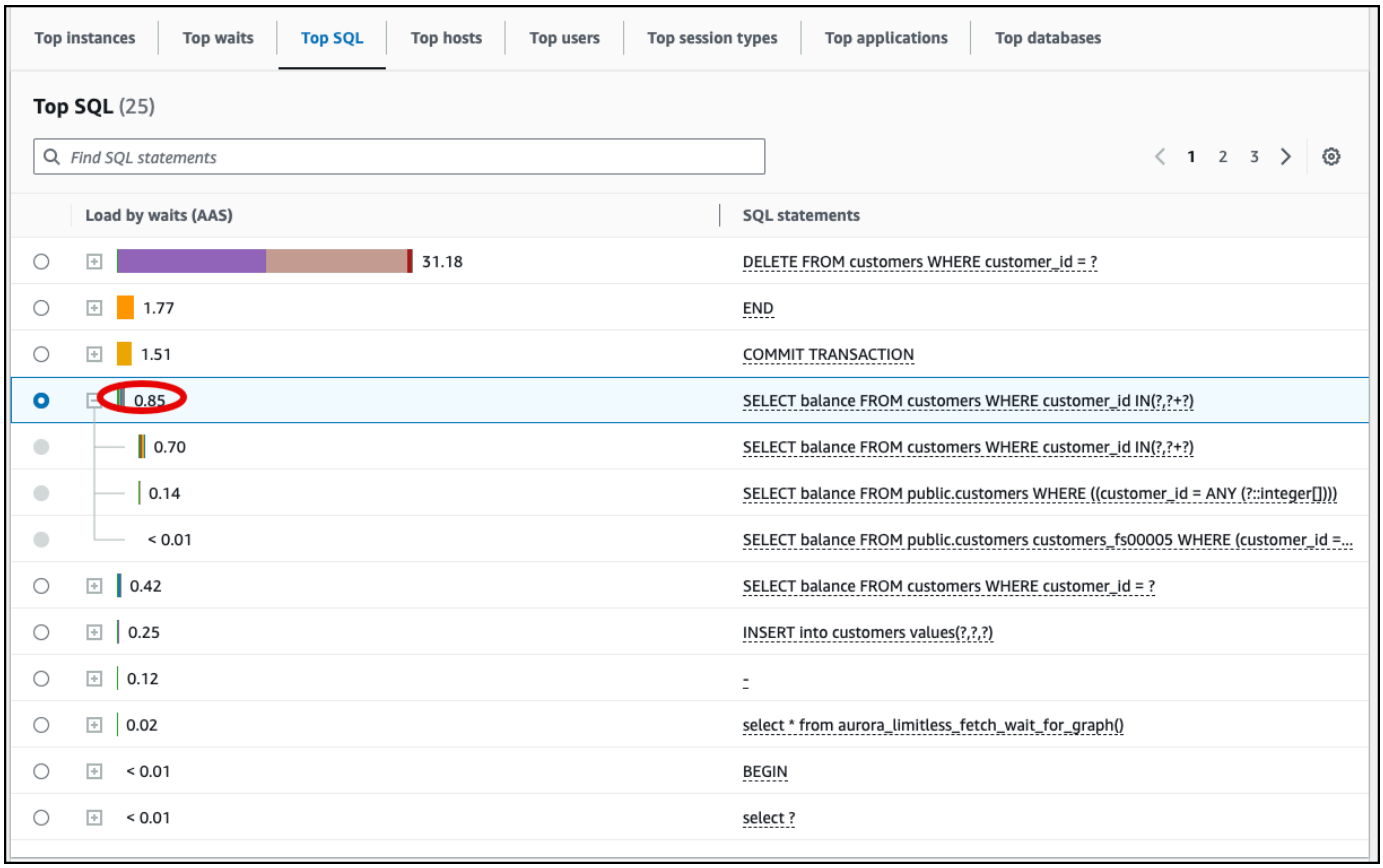
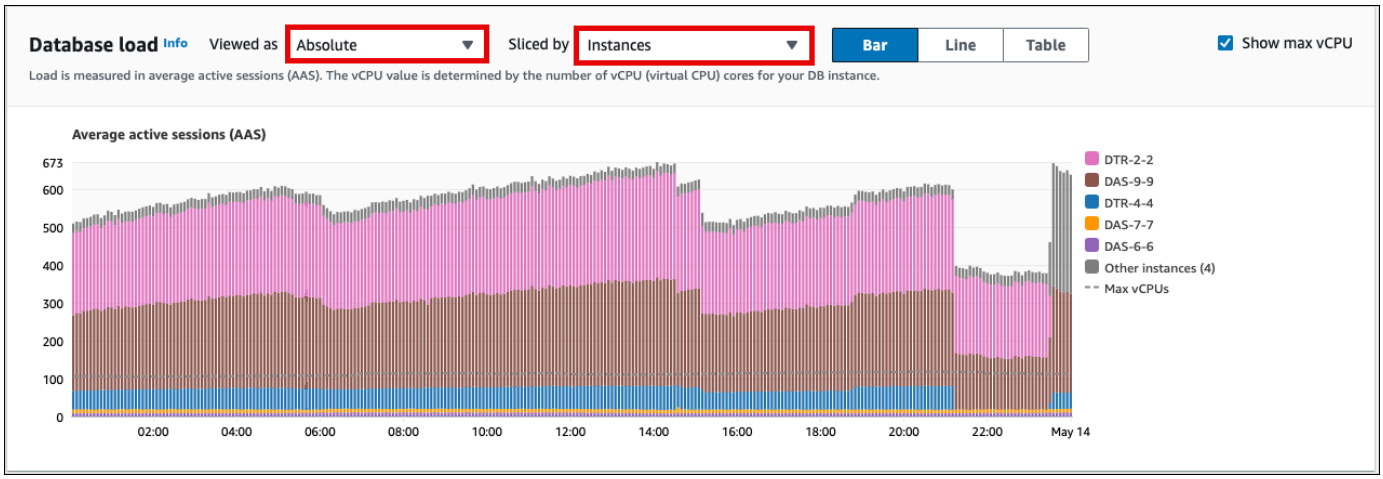
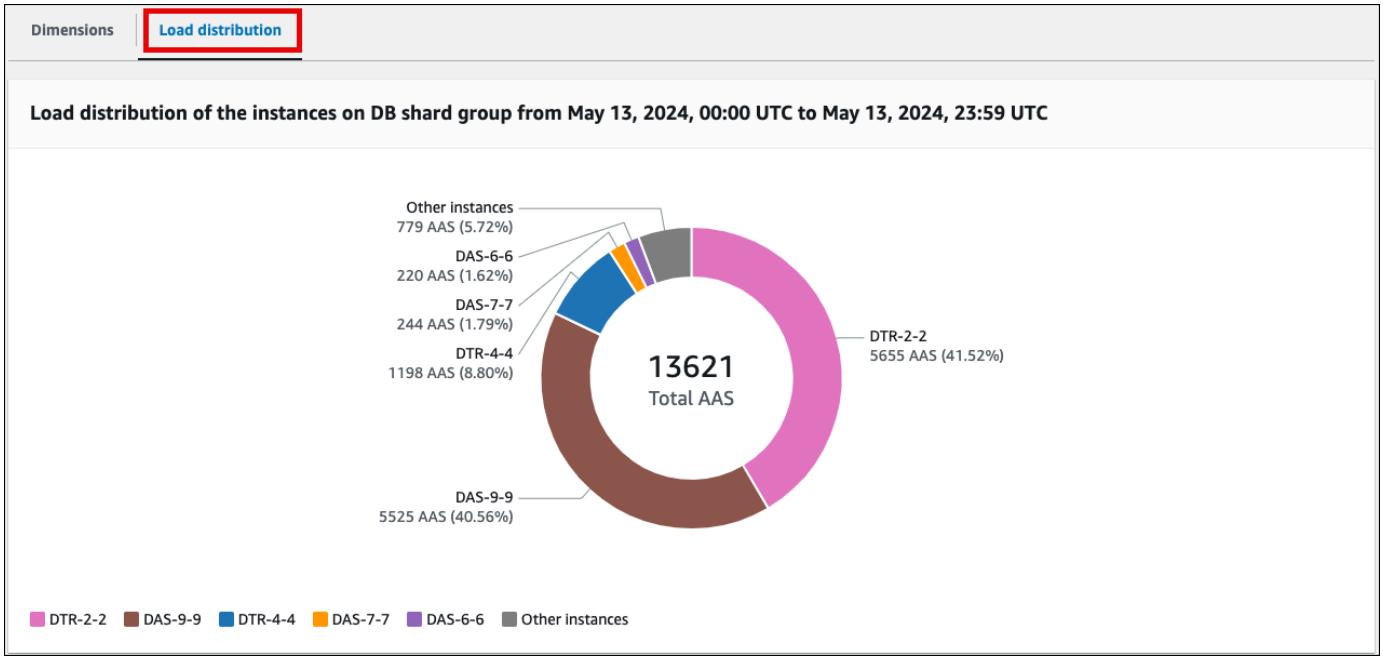
-- This statement is changed to:
DELETE FROM customers WHERE customer_id = '123'::int;
```

聯結

在具有作用中碎片索引鍵的資料表執行聯結操作時，碎片索引鍵述詞會自動新增至聯結中涉及的所有資料表。只有在查詢中的所有資料表都屬於相同的共置群組時，才會自動新增碎片索引鍵述詞。如果查詢涉及來自不同共置群組的資料表，則會改為引發錯誤。

假設我們也有與 customers 資料表共置的 orders 和 order_details 資料表。

```
SET local rds_aurora.limitless_create_table_mode='sharded';
SET local rds_aurora.limitless_create_table_collocate_with='customers';
SET local rds_aurora.limitless_create_table_shard_key='{"customer_id"}';
CREATE TABLE orders (id int , customer_id int, total_amount int, date date);
CREATE TABLE order_details (id int , order_id int, customer_id int, product_name
 VARCHAR(100), price int);
COMMIT;
```

針對客戶 ID 為 10 的客戶，擷取最後 10 張訂單發票。

```
SET rds_aurora.limitless_active_shard_key = '{"10"}';
SELECT * FROM customers, orders, order_details WHERE
    orders.customer_id = customers.customer_id AND
    order_details.order_id = orders.order_id AND
    customers.customer_id = 10
    order by order_date limit 10;
```

此查詢會轉換為以下項目：

```
SELECT * FROM customers, orders, order_details WHERE
    orders.customer_id = customers.customer_id AND
    orders.order_id = order_details.order_id AND
    customers.customer_id = 10 AND
    order_details.customer_id = 10 AND
    orders.customer_id = 10 AND
    ORDER BY "order_date" LIMIT 10;
```

作用中碎片索引鍵相容資料表

碎片索引鍵述詞只會新增至與作用中碎片索引鍵相容的資料表。如果資料表的碎片索引鍵中有與 rds_aurora.limitless_active_shard_key 變數中指定的相同數量欄，則該資料表會被視為相容。如果查詢涉及與作用中碎片索引鍵不相容的資料表，系統會引發錯誤，而不是繼續進行查詢。

例如：

```
-- Compatible table
SET rds_aurora.limitless_active_shard_key = '{"10"}';

-- The following query works because the customers table is sharded on one column.
SELECT * FROM customers;

-- Incompatible table
SET rds_aurora.limitless_active_shard_key = '{"10","20"}';

-- The following query raises a error because the customers table isn't sharded on
 two columns.
 SELECT * FROM customers;
```

Aurora PostgreSQL Limitless Database 中的分散式查詢

分散式查詢會在路由器和多個碎片執行。其中一個路由器會收到此查詢。路由器會建立和管理分散式交易，並將其傳送至參與的碎片。這些碎片會使用路由器提供的內容建立本機交易，並執行查詢。

遞交交易時，路由器會視需要使用最佳化的兩階段遞交通訊協定，並以時間為基礎的多版本並行控制 (MVCC)，在分散式資料庫系統中提供 ACID 語意。

以時間為基礎的 MVCC 會記錄每筆交易的遞交時間，並使用交易開始時間來產生資料快照時間。為了識別指定讀取器快照是否遞交交易 (可見)，資料庫會比較其遞交時間與快照時間。如果其遞交時間小於讀取器的快照時間，則其是可見的，否則為不可見。在此通訊協定下，您將永遠預期在 Aurora PostgreSQL Limitless Database 看到強式一致的資料。

Aurora PostgreSQL Limitless Database 中 PostgreSQL 日誌中的分散式查詢追蹤

分散式查詢追蹤是跨 Aurora PostgreSQL Limitless Database 追蹤和關聯 PostgreSQL 日誌中查詢的工具。在 Aurora PostgreSQL 中，您可以使用交易 ID 來識別交易。但在 Aurora PostgreSQL Limitless Database 中，交易 ID 可以在各種路由器之間重複。因此，我們建議您在 Limitless Database 中使用追蹤 ID。

下列為若干關鍵使用案例：

- 使用 `rds_aurora.limitless_get_last_trace_id()` 函數尋找在目前工作階段中上次查詢執行的唯一追蹤 ID。然後使用該追蹤 ID，在 Amazon CloudWatch Logs 中搜尋資料庫叢集日誌群組，以尋找所有相關日誌。

  您可以同時使用 `log_min_messages` 和 `log_min_error_statement` 參數來控制列印的日誌量，以及列印包含追蹤 ID 的陳述式。
- 使用 `log_min_duration_statement` 參數，來判斷所有查詢列印其執行持續時間和追蹤 ID 的執行時間。然後，您可以在 CloudWatch Logs 中的資料庫叢集日誌群組中搜尋此執行時間，以協助判斷瓶頸節點和規劃器最佳化工作。

  無論 `log_min_messages` 和 `log_min_error_statement` 參數的值為何，`log_min_duration_statement` 參數都會啟用所有節點的追蹤 ID。

主題

- 追蹤 ID
- 使用查詢追蹤
- 日誌範例

追蹤 ID

此功能的核心是稱為追蹤 ID 的唯一識別符。追蹤 ID 是附加至 PostgreSQL 日誌 STATEMENT 日誌行的 31 位數字串，可作為與特定查詢執行相關日誌的精確識別符。範例包括 `11262533757194085040000000000011` 和 `11262533757194084950000000000090`。

追蹤 ID 由下列項目組成：

- 交易識別符：前 20 位數，用來唯一識別交易。

- 命令識別符：前 30 位數，表示交易中的個別查詢。

  如果在明確的交易區塊內執行超過 4,294,967,294 個查詢，則命令識別符會繞回至 1。發生這種情況時，您會在 PostgreSQL 日誌中收到下列 LOG 訊息的通知：

  ```
  wrapping around the tracing ID back to 1 after running 4294967294 (4.2 billion or
   2^32-2) queries inside of an explicit transaction block
  ```

- 節點類型識別符：最後一個數字，指出節點是作為協調器路由器 (1) 還是參與者節點 (0)。

下列範例說明追蹤 ID 的元件：

- 11262533757194085040000000000011:
  - 交易識別符：1126253375719408504
  - 命令識別符：11262533757194085040000000000001 指出交易區塊中的第一個命令
  - 節點類型識別符：1 表示協調器路由器
- 11262533757194084950000000000090:
  - 交易識別符：1126253375719408495
  - 命令識別符：11262533757194084950000000000009 指出交易區塊中的第九個命令
  - 節點類型識別符：0 表示參與者節點

使用查詢追蹤

執行下列任務以使用查詢追蹤：

1. 確定已啟用追蹤。

   您可以使用下列命令來檢查：

   ```
   SHOW rds_aurora.limitless_log_distributed_trace_id;
   ```

   依預設會啟用此功能 (on)。如果未啟用，請使用下列命令進行設定：

   ```
   SET rds_aurora.limitless_log_distributed_trace_id = on;
   ```

2. 透過設定日誌嚴重性層級，來控制列印的日誌數量。

日誌數量由 log_min_messages 參數控制。log_min_error_statement 參數用於列印具有追蹤 ID 的 STATEMENT 行。預設會將兩者設為 ERROR。您可以使用下列命令來檢查：

```
SHOW log_min_messages;
SHOW log_min_error_statement;
```

若要更新嚴重性層級並列印目前工作階段的 STATEMENT 行，請使用下列命令搭配其中一個嚴重性層級：

```
SET log_min_messages = 'DEBUG5 | DEBUG4 | DEBUG3 | DEBUG2 | DEBUG1 | INFO | NOTICE
 | WARNING | ERROR | LOG | FATAL | PANIC';
SET log_min_error_statement = 'DEBUG5 | DEBUG4 | DEBUG3 | DEBUG2 | DEBUG1 | INFO
 | NOTICE | WARNING | ERROR | LOG | FATAL | PANIC';
```

例如：

```
SET log_min_messages = 'WARNING';
SET log_min_error_statement = 'WARNING';
```

3. 啟用在超過特定執行時間後，在日誌中列印追蹤 ID 的功能。

   log_min_duration_statement 參數可變更為查詢的最短執行時間，當查詢執行時間超過此值時，系統將列印日誌行 (內含執行持續時間與資料庫叢集中的追蹤 ID)。此參數預設為 -1，這表示其已停用。您可以使用下列命令來檢查：

   ```
   SHOW log_min_duration_statement;
   ```

   將其變更為 0 會列印資料庫叢集中每個查詢日誌中的持續時間和追蹤 ID。您可以使用下列命令，將目前工作階段的此項目設為 0：

   ```
   SET log_min_duration_statement = 0;
   ```

4. 取得追蹤 ID。

   執行查詢後 (即使在明確交易區塊內)，請呼叫 rds_aurora.limitless_get_last_trace_id 函數以取得上次查詢執行的追蹤 ID：

   ```
   SELECT * FROM rds_aurora.limitless_get_last_trace_id();
   ```

此函數會傳回交易識別符和命令識別符。其不會傳回節點類型識別符。

```
=> SELECT * FROM customers;
 customer_id | fname | lname
-------------+-------+-------
(0 rows)

=> SELECT * FROM rds_aurora.limitless_get_last_trace_id();
 transaction_identifier |       command_identifier
------------------------+--------------------------------
 10104661421959001813   | 101046614219590018130000000001
(1 row)
```

函數會傳回非分散式查詢的空白行，因為其沒有追蹤 ID。

```
=> SET search_path = public;
SET

=> SELECT * FROM rds_aurora.limitless_get_last_trace_id();
 transaction_identifier | command_identifier
------------------------+--------------------
                        |
(1 row)
```

> **Note**
>
> 對於 VACUUM 和 ANALYZE 查詢，不會使用追蹤 ID 記錄持續時間陳述式。因此，`limitless_get_last_trace_id()` 不會傳回追蹤 ID。如果 VACUUM 或 ANALYZE 是長時間執行的操作，您可以使用下列查詢來取得該操作的追蹤 ID：
>
> ```
> SELECT * FROM rds_aurora.limitless_stat_activity
> WHERE distributed_tracing_id IS NOT NULL;
> ```
>
> 如果伺服器在您可以找到最後一個追蹤 ID 前停止，您必須手動搜尋 PostgreSQL 日誌，以便在失敗之前從日誌中找到追蹤 ID。

5. 使用 CloudWatch 在資料庫叢集日誌中搜尋追蹤 ID。

如下列範例所示，使用 CloudWatch Insights 查詢資料庫叢集的日誌群組。

- 查詢特定交易識別符，以及其中執行的所有命令：

```
fields @timestamp, @message
| filter @message like /10104661421959001813/
| sort @timestamp desc
```

- 查詢特定命令識別符：

```
fields @timestamp, @message
| filter @message like /10104661421959001813000000000001/
| sort @timestamp desc
```

6. 檢查分散式查詢所產生之資料庫叢集中的所有日誌。

日誌範例

下列範例示範查詢追蹤的使用。

將容易出錯的查詢日誌進行關聯

在此範例中，當 customers 資料表不存在時，TRUNCATE 命令會在該資料表上執行。

沒有查詢追蹤

協調器路由器上的 PostgreSQL 日誌檔案：

```
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:ERROR: failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:DETAIL: relation
 "public.customers" does not exist
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:CONTEXT: remote
 SQL command: truncate public.customers;
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:STATEMENT:
 truncate customers;
```

參與者碎片上的 PostgreSQL 日誌檔案：

```
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[ 27503]:ERROR: failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[ 27503]:STATEMENT:
 truncate customers;
```

這些是典型的日誌。其缺少在資料庫叢集之間輕鬆關聯查詢所需的精確識別符。

使用查詢追蹤

協調器路由器上的 PostgreSQL 日誌檔案：

```
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:ERROR: failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:DETAIL: relation
 "public.customers" does not exist
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:CONTEXT: remote SQL
 command: truncate public.customers;
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:STATEMENT: /* tid =
 1126253375719408502700000000011 */ truncate customers;
```

參與者碎片上的 PostgreSQL 日誌檔案：

```
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:ERROR:  failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:STATEMENT:  /* tid
 = 1126253375719408502700000000010 */ truncate customers;
```

在查詢追蹤的情況，每個日誌行都會附加 31 位數的唯一識別符。在這裡，1126253375719408502700000000011 和 1126253375719408502700000000010 分別代表協調器和參與者節點的追蹤 ID。

- 交易識別符：11262533757194085027
- 命令識別符：1126253375719408502700000000001
- 節點類型識別符：最後一個位數 (1 或 0) 分別表示協調器路由器和參與者節點。

關聯日誌以尋找各種節點上的查詢執行時間

在此範例中，log_min_duration_statement 參數已更新為 0，以列印所有查詢的持續時間。

沒有查詢追蹤

```
2024-01-15 07:28:46 UTC:[local]:postgres@postgres_limitless:[178322]:LOG: duration:
 12.779 ms statement: select * from customers;
```

使用查詢追蹤

協調器路由器上的 PostgreSQL 日誌檔案：

```
2024-01-15 07:32:08 UTC:[local]:postgres@postgres_limitless:[183877]:LOG: duration:
 12.618 ms statement: /* tid = 045766956624049708840000000011 */ select * from
 customers;
```

參與者碎片上的 PostgreSQL 日誌檔案：

```
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.279 ms statement: /* tid = 045766956624049708840000000010 */ START
 TRANSACTION ISOLATION LEVEL READ COMMITTED
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.249 ms parse <unnamed>: SELECT customer_id, fname, lname FROM
 public.customers
```

```
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.398 ms bind <unnamed>/c1: SELECT customer_id, fname, lname FROM
 public.customers
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.019 ms execute <unnamed>/c1: SELECT customer_id, fname, lname FROM
 public.customers
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.073 ms statement: /* tid = 045766956624049708840000000010 */ COMMIT
 TRANSACTION
```

Aurora PostgreSQL Limitless Database 中的分散式死鎖

在資料庫碎片群組中，分佈在不同路由器和碎片之間的交易之間可能會發生死鎖。例如，如下圖所示，會執行跨越兩個碎片的兩個並行分散式交易。



如下所示，這些交易會鎖定資料表，以及在兩個碎片中建立等待事件：

1. 分散式交易 1：

```
UPDATE table SET value = 1 WHERE key = 'shard1_key';
```

   這會在碎片 1 上保留鎖定。

2. 分散式交易 2：

```
UPDATE table SET value = 2 WHERE key = 'shard2_key';
```

這會在碎片 2 上保留鎖定。

3. 分散式交易 1：

```
UPDATE table SET value = 3 WHERE key = 'shard2_key';
```

分散式交易 1 正在等待碎片 2。

4. 分散式交易 2：

```
UPDATE table SET value = 4 WHERE key = 'shard1_key';
```

分散式交易 2 正在等待碎片 1。

在此案例中，碎片 1 和碎片 2 都看不到問題：交易 1 正在等待碎片 2 上的交易 2，而交易 2 正在等待碎片 1 上的交易 1。透過全域檢視，交易 1 正在等待交易 2，交易 2 正在等待交易 1。在這種情況下，兩個不同碎片上的兩個交易正在彼此等待 (稱為分散式死鎖)。

Aurora PostgreSQL Limitless Database 可以自動偵測和解決分散式死鎖。資料庫碎片群組中的路由器會在交易等待太久而無法取得資源時收到通知。接收通知的路由器會開始從資料庫碎片群組中的所有路由器和碎片收集必要的資訊。然後，路由器會繼續將參與分散式死鎖的交易加以結束，直到資料庫碎片群組中的其餘交易可以繼續，而不會彼此封鎖。

當交易是分散式死鎖的一部分，接著由路由器結束時，您會收到下列錯誤：

```
ERROR: aborting transaction participating in a distributed deadlock
```

`rds_aurora.limitless_distributed_deadlock_timeout` 資料庫叢集參數會設定每個交易等待資源的時間，之後才會通知路由器檢查分散式死鎖。如果工作負載較不容易發生死鎖情況，您可以提高參數值。預設值為 `1000` 毫秒 (1 秒)。

找到並解析跨節點死鎖時，分散式死鎖週期會發佈至 PostgreSQL 日誌。作為死鎖一部分之每個程序的相關資訊包括下列項目：

- 啟動交易的協調器節點
- 協調器節點上交易的虛擬交易 ID (xid) (格式為 *backend_id*/*backend_local_xid*)
- 交易的分散式工作階段 ID

管理 Aurora PostgreSQL Limitless Database

下列主題說明如何管理 Aurora PostgreSQL Limitless Database 資料庫叢集。

主題

- 資料庫和資料表大小考量事項
- 透過清空回收儲存空間

資料庫和資料表大小考量事項

對於 Aurora PostgreSQL Limitless Database，在每個碎片上，碎片資料表會分為數個資料表配量，根據資料庫碎片群組中可用的碎片數量而有所不同。每個資料表配量最多可成長至 32 TiB，但每個碎片的容量上限為 128 TiB。對於整個資料庫碎片群組，參考資料表的大小限制為 32 TiB。

> **Note**
>
> 每個節點 (路由器或碎片) 的容量上限為 128 TiB，因為這是 Aurora PostgreSQL 資料庫叢集的容量上限。

Aurora PostgreSQL 和 Aurora PostgreSQL Limitless Database 中每個資料庫 (包括資料表、檢視和索引) 的關係數目上限為 1,431,650,303。

如需詳細資訊，請參閱 PostgreSQL 文件中的附錄 K. PostgreSQL 限制和 Amazon Aurora 大小限制。

透過清空回收儲存空間

PostgreSQL 多版本並行控制 (MVCC) 可透過儲存更新或刪除列的內部副本，直到交易遞交或復原為止，協助保持資料完整性。如果未定期清除，稱為元組的這些副本可能會導致資料表膨脹。PostgreSQL 執行個體會依交易 ID 排序交易，而 PostgreSQL 會使用交易 ID 型 MVCC 來控制元組可見性並提供交易隔離。每個交易都會建立資料的快照，而且每個元組都有版本。快照和版本都是以交易 ID 為基礎。

若要清除資料，VACUUM 公用程式會在 PostgreSQL 中執行四個關鍵函數：

- VACUUM：移除過期的列版本，讓空間可供重複使用。
- VACUUM FULL：透過移除無效列版本與壓縮資料表、減少大小與提高效率，來提供完整的重組。
- VACUUM FREEZE：將較舊的列版本標記為凍結，以防止交易 ID 包裝問題。
- VACUUM ANALYZE：移除無效列版本與更新資料庫的查詢規劃統計資料。這是 VACUUM 和 ANALYZE 函數的組合。如需 ANALYZE 如何在 Aurora PostgreSQL Limitless Database 中運作的詳細資訊，請參閱 ANALYZE。

如同 MVCC，Aurora PostgreSQL 中的清空是以交易 ID 為基礎。如果在清空開始時有進行中的交易，則不會移除該交易仍然可見的列。

如需 VACUUM 公用程式的詳細資訊，請參閱 PostgreSQL 文件中的 VACUUM。如需 Aurora PostgreSQL Limitless Database 中 VACUUM 支援的詳細資訊，請參閱 VACUUM。

主題

- AUTOVACUUM
- Aurora PostgreSQL Limitless Database 中以時間為基礎的清空
- 使用資料庫統計資料進行清空
- Aurora PostgreSQL 和 Aurora PostgreSQL Limitless Database 之間清空行為的差異

AUTOVACUUM

Aurora PostgreSQL 使用 VACUUM 和 AUTOVACUUM 公用程式來移除不需要的元組。AUTOVACUUM 和手動 VACUUM 的基礎機制相同；唯一的差異是自動化。

AUTOVACUUM 在 Aurora PostgreSQL 和 Aurora PostgreSQL Limitless Database 中是 VACUUM 和 ANALYZE 公用程式的組合。AUTOVACUUM 會根據預先定義的規則，決定要清除哪些資料庫和資料表，例如無效元組的百分比和插入次數。

例如，AUTOVACUUM 會定期「喚醒」以執行清除。間隔由 autovacuum_naptime 參數控制。預設值為 1 分鐘。Aurora PostgreSQL Limitless Database 的 AUTOVACUUM 和 VACUUM 組態參數預設值與 Aurora PostgreSQL 相同。

如果啟用 AUTOVACUUM 常駐程式，則每當資料表的內容有足夠變更時，該常駐程式會自動發出 ANALYZE 命令。在 Aurora PostgreSQL Limitless Database 中，AUTOVACUUM 都會在路由器和碎片發出 ANALYZE。

如需與 AUTOVACUUM 相關聯的 AUTOVACUUM 常駐程式和資料表儲存參數的詳細資訊，請參閱 PostgreSQL 文件中的自動清空常駐程式和儲存參數。

Aurora PostgreSQL Limitless Database 中以時間為基礎的清空

Aurora PostgreSQL Limitless Database 是分散式系統，這表示交易中可能涉及多個執行個體。因此，以交易 ID 為基礎的可見性就不適用。反之，Aurora PostgreSQL Limitless Database 會使用以時間為基礎的可見性，因為跨執行個體的交易 ID 不會「統一」，但跨執行個體的時間可以「統一」。每個交易快照和每個元組版本都遵守時間 (而不是交易 ID)。更具體而言，交易快照具有快照開始時間，而元組具有建立時間 (當 INSERT或 UPDATE 發生時) 和刪除時間 (當 DELETE 發生時)。

為了維持資料庫碎片群組中執行個體中的資料一致性，Aurora PostgreSQL Limitless Database 必須確保清空不會移除資料庫碎片群組中任何作用中交易仍然可見的任何元組。因此，Aurora PostgreSQL Limitless Database 中的清空也是以時間為基礎。VACUUM 的其他方面保持不變，包括在特定資料表上執行 VACUUM 的方面，使用者必須能夠存取該資料表。

> **Note**
>
> 強烈建議您不要讓交易長時間處於未結狀態。
> 相較於以交易 ID 為基礎的清空，以時間為基礎的清空耗用的記憶體更多。

下列範例說明以時間為基礎的清空的運作方式。

1. 客戶資料表會分佈到四個碎片。
2. 交易 1 會從可重複讀取開始，並僅鎖定一個碎片 (碎片 1)。此交易保持未結狀態。

   交易 1 比之後啟動的任何其他交易更舊。
3. 交易 2 稍後啟動，並從資料表中刪除所有元組，然後遞交。
4. 如果 AUTOVACUUM 或手動 VACUUM 嘗試清除無效元組 (因交易 2 而失效)，則不會移除任何項目。

這不僅適用於碎片 1，也適用於碎片 2–4，因為交易 1 仍可能需要存取這些元組。由於 MVCC，交易 1 仍然可以看到這些項目。

最後一步是透過同步來達成，因此所有碎片都知道交易 1，即使交易 1 未觸及所有碎片。

使用資料庫統計資料進行清空

若要取得您可能需要清除的元組資訊，請使用 limitless_stat_all_tables 檢視，其運作方式類似於 pg_stat_all_tables。以下範例會查詢此檢視。

```
SELECT * FROM rds_aurora.limitless_stat_all_tables WHERE relname LIKE '%customer%';
```

同樣地，對於資料庫統計資料，請使用 limitless_stat_database (而不是 pg_stat_database)，以及 limitless_stat_activity (而不是 pg_stat_activity)。

若要檢查資料表磁碟用量，請使用 limitless_stat_relation_sizes 函數，其運作方式類似於 pg_relation_size。下列範例會查詢此函數。

```
SELECT * FROM rds_aurora.limitless_stat_relation_sizes('public','customer');
```

若要追蹤 Aurora PostgreSQL Limitless Database 上 VACUUM 操作的進度，請使用 limitless_stat_progress_vacuum 檢視 (而非 pg_stat_progress_vacuum)。以下範例會查詢此檢視。

```
SELECT * FROM rds_aurora.limitless_stat_progress_vacuum;
```

如需詳細資訊，請參閱Aurora PostgreSQL Limitless Database 檢視及Aurora PostgreSQL Limitless Database 函數。

Aurora PostgreSQL 和 Aurora PostgreSQL Limitless Database 之間清空行為的差異

Aurora PostgreSQL 和 Aurora PostgreSQL Limitless Database 在清空運作方式的一些其他差異如下：

- Aurora PostgreSQL 會對交易 ID 執行 VACUUM 操作，直到最舊的進行中交易為止。如果資料庫中沒有持續的交易，VACUUM 會執行操作，直到上次交易為止。
- Aurora PostgreSQL Limitless Database 每 10 秒同步一次最舊的時間快照。因此，VACUUM 可能不會對過去 10 秒內執行的任何交易執行操作。

如需在 Aurora PostgreSQL Limitless Database 中支援 VACUUM 的資訊，請參閱 VACUUM中的 Aurora PostgreSQL Limitless Database 參考。

監控 Aurora PostgreSQL Limitless Database

您可以使用 Amazon CloudWatch、增強型監控和 Performance Insights 來監控 Aurora PostgreSQL Limitless Database。也有適用於 Aurora PostgreSQL Limitless Database 的新統計資料函數和檢視，以及等待事件 (可用於監控和診斷)。

主題

- 使用 Amazon CloudWatch 監控 Aurora PostgreSQL Limitless Database
- 使用 CloudWatch Database Insights 監控 Aurora PostgreSQL Limitless Database
- 使用 Amazon CloudWatch Logs 監控 Aurora PostgreSQL Limitless Database
- 使用增強型監控來監控 Aurora PostgreSQL Limitless Database
- 使用 Performance Insights 監控 Aurora PostgreSQL Limitless Database
- 使用 Amazon GuardDuty RDS 保護監控 Aurora PostgreSQL Limitless Database
- Aurora PostgreSQL Limitless Database 的函數和檢視
- Aurora PostgreSQL Limitless Database 的等待事件
- 使用 函數建置以提高效率

使用 Amazon CloudWatch 監控 Aurora PostgreSQL Limitless Database

Aurora PostgreSQL Limitless Database 的 CloudWatch 指標會以下列維度報告：

- DBShardGroup
- DBShardGroupRouterAggregation
- DBShardGroupInstance
- DBClusterIdentifier

如需 CloudWatch 指標的相關資訊，請參閱 使用 Amazon CloudWatch 監控 Amazon Aurora 指標。

DBShardGroup 指標

若要在 CloudWatch 主控台中查看 Aurora PostgreSQL Limitless Database 的 `DBShardGroup` 指標，請選擇 RDS，然後選擇 DBShardGroup。

您可以追蹤下列 CloudWatch 指標：

- `DBShardGroupACUUtilization`：Aurora 容量單位 (ACU) 使用量，以百分比計算 (透過將 `DBShardGroupCapacity` 除以 `DBShardGroupMaxACU`)。
- `DBShardGroupCapacity`：資料庫碎片群組的寫入器執行個體所耗用的 ACU 數量。
- `DBShardGroupComputeRedundancyCapacity`：資料庫碎片群組的待命執行個體所耗用的 ACU 數量。
- `DBShardGroupMaxACU`：資料庫碎片群組設定的 ACU 數目上限。
- `DBShardGroupMinACU`：資料庫碎片群組所需的 ACU 數目下限。

`DBShardGroupIdentifier` 維度索引鍵可用於彙總 `DBShardGroup` 指標。

DBShardGroupRouterAggregation 指標

若要在 CloudWatch 主控台中查看 Aurora PostgreSQL Limitless Database 的 `DBShardGroupRouterAggregation` 指標，請選擇 RDS，然後選擇 DBShardGroupRouterAggregation。

您可以追蹤下列 CloudWatch 指標：

- `CommitThroughput`：資料庫碎片群組中所有路由器節點的每秒遞交操作平均數量。

- DatabaseConnections：資料庫碎片群組中所有路由器節點的所有連線總和。

DBShardGroupInstance 指標

DBShardGroupInstance 是每個碎片或路由器子叢集中的個別資料庫執行個體。

若要在 CloudWatch 主控台中查看 Aurora PostgreSQL Limitless Database 的 DBShardGroupInstance 指標，請選擇 RDS，然後選擇 DBShardGroupInstance。

您可以追蹤下列 CloudWatch 指標：

- ACUUtilization：以 ServerlessDatabaseCapacity 指標除以子叢集的指派 ACU 值上限所計算的百分比。
- AuroraReplicaLag：對於啟用運算備援的無限叢集，這是從子叢集中主要執行個體複寫更新時的延遲量。
- AuroraReplicaLagMaximum：對於啟用運算備援的無限叢集，這是從子叢集中主要執行個體複寫更新時的最大延遲量。刪除或重新命名僅供讀取複本時，當舊資源經過回收時，複寫延遲可能會暫時遽增。使用此指標，來尋找是否因其中一個讀取器的高複寫延遲而導致容錯移轉。
- AuroraReplicaLagMinimum：對於啟用運算備援的無限叢集，這是從子叢集中主要執行個體複寫更新時的最小延遲量。
- BufferCacheHitRatio：從執行個體記憶體快取提供的資料和索引百分比 (相對於儲存磁碟區)。
- CommitLatency：引擎和儲存體完成特定節點 (路由器或碎片) 遞交操作的平均持續時間。
- CommitThroughput：遞交操作的每秒平均數量。
- CPUUtilization：CPU 使用量佔子叢集最大指派 ACU 值的百分比。
- FreeableMemory：當碎片群組擴展至其最大容量時，可用的未使用記憶體量。這取決於碎片群組的指派 ACU。若為目前容量低於最大容量的每個 ACU，此值增加約 2 GiB。因此，在資料庫碎片群組擴展至上限之前，此指標不會接近零。
- MaximumUsedTransactionIDs：最舊的未清理交易 ID 的存留期 (以交易為單位)。如果此值達到 2,146,483,648 (2^31 - 1,000,000)，資料庫將強制進入唯讀模式，以避免交易 ID 包圍。如需詳細資訊，請參閱 PostgreSQL 文件中的避免交易 ID 包圍失敗。
- NetworkReceiveThroughput：資料庫碎片群組中的各個執行個體從用戶端接收到的網路輸送量。此輸送量不包含資料庫碎片群組中的執行個體與叢集磁碟區之間的網路流量。
- NetworkThroughput：資料庫碎片群組中用戶端和路由器，以及路由器和碎片之間的彙總網路輸送量 (傳輸和接收)。此輸送量不包含資料庫碎片群組中的執行個體與叢集磁碟區之間的網路流量。

- NetworkTransmitThroughput：資料庫碎片群組中的各個執行個體傳送至用戶端的網路輸送量。此輸送量不包含資料庫碎片群組中的執行個體與叢集磁碟區之間的網路流量。
- ReadIOPS：磁碟讀取每秒讀寫次數 (IOPS) 的平均次數。
- ReadLatency：每次磁碟讀取輸入/輸出 (I/O) 操作耗用的時間。
- ReadThroughput：平均每秒從磁碟讀取的位元組數目。
- ServerlessDatabaseCapacity：資料庫碎片群組中資料庫碎片或路由器子叢集的目前容量。
- StorageNetworkReceiveThroughput：資料庫碎片群組中各個執行個體從 Aurora 儲存子系統接收到的網路輸送量。
- StorageNetworkThroughput：資料庫碎片群組中每個執行個體從 Aurora 儲存子系統傳輸和接收的彙總網路輸送量。
- StorageNetworkTransmitThroughput：資料庫碎片群組中各個執行個體傳送至 Aurora 儲存子系統的網路輸送量。
- SwapUsage：資料庫碎片群組使用的交換空間量。
- TempStorageIOPS：在連接至資料庫執行個體的本機儲存體上執行的 I/O 操作平均數量。其中包含讀取和寫入 I/O 操作。

  TempStorageIOPS 可與 TempStorageThroughput 搭配使用，以診斷於資料庫執行個體和本機儲存體間進行傳輸之網路活動造成容量意外增加的罕見狀況。
- TempStorageThroughput：傳入和傳出與路由器或碎片相關聯之本機儲存體的資料量。
- WriteIOPS：磁碟寫入 IOPS 的平均數量。
- WriteLatency：平均每次磁碟寫入輸入/輸出操作耗用的時間。
- WriteThroughput：平均每秒寫入磁碟的位元組數目。

下列維度金鑰可用於彙總 DBShardGroupInstance 指標：

- DBClusterIdentifier：Aurora PostgreSQL 資料庫叢集。
- DBShardGroupIdentifier：執行個體所屬的資料庫碎片群組。
- DBShardGroupSubClusterType：節點類型，可以是 Distributed Transaction Router (路由器) 或 Data Access Shard (碎片)。
- DBShardGroupSubClusterIdentifier：執行個體所屬的路由器或碎片名稱。

以下是彙總 CloudWatch 指標的範例：

- 資料庫碎片群組中屬於特定碎片或路由器的所有執行個體總計 `CPUUtilization`。
- 資料庫碎片群組中所有執行個體的總計 `CPUUtilization`。

DBClusterIdentifier 指標

若要在 CloudWatch 主控台查看 Aurora PostgreSQL Limitless Database 的 `DBClusterIdentifier` 指標，請選擇 RDS，然後選擇 DBClusterIdentifier。

當您使用 Aurora PostgreSQL Limitless Database 時，可能會有比 Aurora 資料庫叢集更多的輸入/輸出 (I/O) 操作。您可以追蹤 Limitless Database 叢集的下列 CloudWatch 指標：

- `VolumeReadIops`：叢集磁碟區的計費讀取 I/O 操作次數，每隔 5 分鐘回報一次。
- `VolumeWriteIops`：叢集磁碟區的寫入磁碟 I/O 操作次數，每隔 5 分鐘回報一次。

Aurora PostgreSQL Limitless Database 使用 Aurora I/O-Optimized 叢集儲存組態。您可以使用 Aurora I/O-Optimized，為所有 I/O 操作支付單一每月價格，而不是為每一百萬個 I/O 請求付費。如需更多詳細資訊，請參閱 Amazon Aurora 資料庫叢集的儲存組態。

您也可以使用比 Aurora 資料庫叢集更多的儲存空間。您可以追蹤下列 CloudWatch 儲存指標：

- `BackupRetentionPeriodStorageUsed`：Aurora PostgreSQL Limitless Database 叢集的計費連續備份儲存使用量總計。
- `SnapshotStorageUsed`：Aurora PostgreSQL Limitless Database 叢集的計費快照儲存使用量總計。
- `TotalBackupStorageBilled`：自動備份保留和資料庫叢集快照的成本總和。

  如需備份儲存成本的詳細資訊，請參閱了解 Amazon Aurora 備份儲存體用量。
- `VolumeBytesUsed`：Aurora PostgreSQL Limitless Database 叢集使用的儲存量，每隔 5 分鐘回報一次。

使用 CloudWatch Database Insights 監控 Aurora PostgreSQL Limitless Database

啟用 Aurora PostgreSQL Limitless Database 時，需要 Database Insights 的標準模式。您可以使用此模式，來即時監控 Limitless Database 資料庫執行個體的資料庫負載 (資料庫負載)。資料庫負載會測量資料庫中工作階段活動的層級。您可以使用 Database Insights，大規模分析 Aurora PostgreSQL Limitless Database 資料庫執行個體的效能和進行疑難排解。

如需 CloudWatch Database Insights 的詳細資訊，請參閱下列主題。

- 使用 CloudWatch Database Insights 監控 Amazon Aurora 資料庫
- Amazon CloudWatch 使用者指南中的 CloudWatch Database Insights
- Amazon CloudWatch 使用者指南中的開始使用 CloudWatch Database Insights
- 使用 Amazon Aurora 的 Database Insights 設定資料庫以監控慢速 SQL 查詢

如需有關開啟 Database Insights 進階模式或標準模式的資訊，請參閱下列主題。

主題

- 開啟 Aurora PostgreSQL Limitless Database 的 Database Insights 進階模式
- 開啟 Aurora PostgreSQL Limitless Database 的 Database Insights 標準模式

開啟 Aurora PostgreSQL Limitless Database 的 Database Insights 進階模式

若要為 Aurora PostgreSQL Limitless Database 開啟 Database Insights 的進階模式，請使用下列程序。

為 Aurora PostgreSQL Limitless Database 建立資料庫叢集時，開啟 Database Insights 的進階模式

為 Aurora PostgreSQL Limitless Database 建立資料庫時，開啟 Database Insights 的進階模式。

Console

在此主控台中，您可以在建立資料庫叢集時開啟 Database Insights 的進階模式。Database Insights 的設定適用於資料庫叢集中的所有資料庫執行個體。

使用主控台建立資料庫叢集時，開啟 Database Insights 的進階模式

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 選擇 Databases (資料庫)。
3. 選擇建立資料庫。
4. 在 Database Insights 區段中，選取進階模式。然後選擇下列選項：
   - Retention (保留) – 保留 Performance Insights 資料的時間。Database Insights 進階模式的保留期間必須為 15-24 個月。
   - AWS KMS key – 指定您的 KMS 金鑰。績效詳情使用您的 KMS 金鑰來加密所有可能的敏感資料。將會對傳輸中與靜態資料進行加密。如需更多詳細資訊，請參閱 加密 Amazon Aurora 資源。
5. 選擇建立資料庫。

AWS CLI

若要在建立資料庫叢集時開啟 Database Insights 的進階模式，請呼叫 create-db-cluster AWS CLI 命令並提供下列值：

- `--db-cluster-identifier`：資料庫叢集的識別符。
- `--database-insights-mode advanced` 以開啟 Database Insights 的進階模式。
- `--engine`：資料庫叢集必須使用 `aurora-postgresql` 資料庫引擎。
- `--engine-version`：資料庫叢集必須使用其中一個資料庫引擎版本：
  - `16.4-limitless`
  - `16.6-limitless`
- `--storage-type`：資料庫叢集必須使用 `aurora-iopt1` 資料庫叢集儲存組態。
- `--cluster-scalability-type`：指定 Aurora 資料庫叢集的可擴展性模式。設為 `limitless` 時，叢集會以 Aurora PostgreSQL Limitless Database 的形式運作。設為 `standard` (預設值) 時，叢集會使用一般資料庫執行個體建立。

> **Note**
>
> 您無法在建立資料庫叢集之後修改此設定。

- `--master-username`：資料庫叢集主要使用者的名稱。
- `--master-user-password`：主要使用者密碼。
- `--enable-performance-insights` 以開啟 Database Insights 的 Performance Insights。
- `--performance-insights-retention-period`：資料庫叢集資料的保留期間。若要開啟 Database Insights ，保留期必須至少為 465 天。
- `--monitoring-interval`：為資料庫叢集收集增強型監控指標點之間的間隔 (秒)。此值不能是 `0`。
- `--monitoring-role-arn`：允許 RDS 將增強型監控指標傳送到 Amazon CloudWatch Logs 之 IAM 角色的 Amazon Resource Name (ARN)。
- `--enable-cloudwatch-logs-exports`：您必須將 `postgresql` 日誌匯出至 CloudWatch Logs。

下列範例會在建立資料庫叢集時啟用 Database Insights 的進階模式。

針對 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
--db-cluster-identifier my-limitless-cluster \
--database-insights-mode advanced \
--engine aurora-postgresql \
--engine-version 16.6-limitless \
--storage-type aurora-iopt1 \
--cluster-scalability-type limitless \
--master-username myuser \
--master-user-password mypassword \
--enable-performance-insights \
--performance-insights-retention-period 465 \
--monitoring-interval 5 \
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole \
--enable-cloudwatch-logs-exports postgresql
```

針對 Windows：

```
aws rds create-db-cluster ^
--db-cluster-identifier my-limitless-cluster ^
--database-insights-mode advanced ^
--engine aurora-postgresql ^
--engine-version 16.6-limitless ^
```

```
--storage-type aurora-iopt1 ^
--cluster-scalability-type limitless ^
--master-username myuser ^
--master-user-password mypassword ^
--enable-performance-insights ^
--performance-insights-retention-period 465 ^
--monitoring-interval 5 ^
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole ^
--enable-cloudwatch-logs-exports postgresql
```

RDS API

若要在建立資料庫叢集時開啟 Database Insights 的進階模式，請為 CreateDBCluster Amazon RDS API 操作指定下列參數。

- `DatabaseInsightsMode` 至 `advanced`
- `Engine` 至 `aurora-postgresql`
- `EngineVersion` 至適用於 Limitless Database 的可用引擎版本
- `StorageType` 至 `aurora-iopt1`
- `ClusterScalabilityType` 至 `limitless`
- `MasterUsername`
- `MasterUserPassword`
- `EnablePerformanceInsights` 至 `True`
- `PerformanceInsightsRetentionPeriod` 到至少 `465` 天
- `MonitoringInterval` 到非 `0` 的值
- `MonitoringRoleArn` 到允許 RDS 將增強型監控指標傳送到 Amazon CloudWatch Logs 之 IAM 角色的 Amazon Resource Name (ARN)

為 Aurora PostgreSQL Limitless Database 修改資料庫叢集時，開啟 Database Insights 的進階模式

為 Aurora PostgreSQL Limitless Database 修改資料庫時，開啟 Database Insights。

> **Note**
>
> 若要啟用 Database Insights ，資料庫叢集中每個資料庫執行個體都必須具有相同的 Performance Insights 和增強型監控設定。

Console

在主控台中，您可以在修改資料庫叢集時開啟 Database Insights 的進階模式。Database Insights 的設定適用於資料庫叢集中的所有資料庫執行個體。

使用主控台修改資料庫叢集時，開啟 Database Insights 的進階模式

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 選擇 Databases (資料庫)。
3. 選擇資料庫叢集，然後選擇修改。
4. 在 Database Insights 區段中，選取進階模式。然後選擇下列選項：
   - Retention (保留) – 保留 Performance Insights 資料的時間。Database Insights 進階模式的保留期間必須為 15-24 個月。
   - AWS KMS key – 指定您的 KMS 金鑰。績效詳情使用您的 KMS 金鑰來加密所有可能的敏感資料。將會對傳輸中與靜態資料進行加密。如需更多詳細資訊，請參閱 加密 Amazon Aurora 資源。
5. 選擇 Continue (繼續)。
6. 在 Scheduling of Modifications (修改排程) 中，選擇 Apply immediately (立即套用)。如果您選擇在下一個排定的維護時段套用，則資料庫會忽略此設定，並立即開啟 Database Insights 的進階模式。
7. 選擇修改叢集。

AWS CLI

若要在修改資料庫叢集時開啟 Database Insights 的進階模式，請呼叫 modify-db-cluster AWS CLI 命令並提供下列值：

- `--database-insights-mode advanced` 以開啟 Database Insights 的進階模式。
- `--db-cluster-identifier`：資料庫叢集的識別符。
- `--enable-performance-insights` 以開啟 Database Insights 的 Performance Insights。
- `--performance-insights-retention-period`：資料庫叢集的資料保留期間。若要開啟 Database Insights 的進階模式，保留期必須至少為 465 天。

下列範例會在修改資料庫叢集時啟用 Database Insights 的進階模式。

針對 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --database-insights-mode advanced \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 465
```

針對 Windows：

```
aws rds modify-db-cluster ^
    --database-insights-mode advanced ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

若要在修改資料庫叢集時開啟 Database Insights 的進階模式，請為 ModifyDBCluster Amazon RDS API 操作指定下列參數。

- `DatabaseInsightsMode` 至 `advanced`
- `EnablePerformanceInsights` 至 `True`
- `PerformanceInsightsRetentionPeriod` 到至少 `465` 天

開啟 Aurora PostgreSQL Limitless Database 的 Database Insights 標準模式

若要為 Aurora PostgreSQL Limitless Database 開啟 Database Insights 的標準模式，請使用下列程序。

為 Aurora PostgreSQL Limitless Database 建立資料庫叢集時，開啟 Database Insights 的標準模式

為 Aurora PostgreSQL Limitless Database 建立資料庫時，開啟 Database Insights 的標準模式。

Console

在此主控台中，您可以在建立資料庫叢集時開啟 Database Insights 的標準模式。Database Insights 的設定適用於資料庫叢集中的所有資料庫執行個體。

使用主控台建立資料庫叢集時，開啟 Database Insights 的標準模式

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 選擇 Databases (資料庫)。
3. 選擇建立資料庫。
4. 在 Database Insights 區段中，選取標準模式。然後選擇下列選項：
   - Retention (保留) – 保留績效詳情資料的時間。若要為 Aurora PostgreSQL Limitless Database 建立資料庫叢集，保留期間必須至少為 31 天。
   - AWS KMS key – 指定您的 KMS 金鑰。績效詳情使用您的 KMS 金鑰來加密所有可能的敏感資料。將會對傳輸中與靜態資料進行加密。如需更多詳細資訊，請參閱 加密 Amazon Aurora 資源。
5. 選擇建立資料庫。

AWS CLI

若要在建立資料庫叢集時開啟 Database Insights 的標準模式，請呼叫 create-db-cluster AWS CLI 命令並提供下列值：

- `--db-cluster-identifier`：資料庫叢集的識別符。
- `--database-insights-mode standard` 以開啟 Database Insights 的標準模式。
- `--engine`：資料庫叢集必須使用 `aurora-postgresql` 資料庫引擎。
- `--engine-version`：資料庫叢集必須使用其中一個資料庫引擎版本：
  - `16.4-limitless`
  - `16.6-limitless`
- `--storage-type`：資料庫叢集必須使用 `aurora-iopt1` 資料庫叢集儲存組態。
- `--cluster-scalability-type`：指定 Aurora 資料庫叢集的可擴展性模式。設為 `limitless` 時，叢集會以 Aurora PostgreSQL Limitless Database 的形式運作。設為 `standard` (預設值) 時，叢集會使用一般資料庫執行個體建立。

> **Note**
>
> 您無法在建立資料庫叢集之後修改此設定。

- `--master-username`：資料庫叢集主要使用者的名稱。
- `--master-user-password`：主要使用者密碼。
- `--enable-performance-insights` 以開啟 Database Insights 的 Performance Insights。
- `--performance-insights-retention-period`：資料庫叢集資料的保留期間。若要為 Aurora PostgreSQL Limitless Database 建立資料庫叢集，保留期間必須至少為 31 天。
- `--monitoring-interval`：為資料庫叢集收集增強型監控指標點之間的間隔 (秒)。此值不能是 `0`。
- `--monitoring-role-arn`：允許 RDS 將增強型監控指標傳送到 Amazon CloudWatch Logs 之 IAM 角色的 Amazon Resource Name (ARN)。
- `--enable-cloudwatch-logs-exports`：您必須將 `postgresql` 日誌匯出至 CloudWatch Logs。

下列範例會在建立資料庫叢集時啟用 Database Insights 的標準模式。

針對 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
--db-cluster-identifier my-limitless-cluster \
--database-insights-mode standard \
--engine aurora-postgresql \
--engine-version 16.6-limitless \
--storage-type aurora-iopt1 \
--cluster-scalability-type limitless \
--master-username myuser \
--master-user-password mypassword \
--enable-performance-insights \
--performance-insights-retention-period 31 \
--monitoring-interval 5 \
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole \
--enable-cloudwatch-logs-exports postgresql
```

針對 Windows：

```
aws rds create-db-cluster ^
--db-cluster-identifier my-limitless-cluster ^
--database-insights-mode standard ^
--engine aurora-postgresql ^
--engine-version 16.6-limitless ^
```

```
--storage-type aurora-iopt1 ^
--cluster-scalability-type limitless ^
--master-username myuser ^
--master-user-password mypassword ^
--enable-performance-insights ^
--performance-insights-retention-period 31 ^
--monitoring-interval 5 ^
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole ^
--enable-cloudwatch-logs-exports postgresql
```

RDS API

若要在建立資料庫叢集時開啟 Database Insights 的標準模式，請為 CreateDBCluster Amazon RDS API 操作指定下列參數。

- `DatabaseInsightsMode` 至 `standard`
- `Engine` 至 `aurora-postgresql`
- `EngineVersion` 至適用於 Limitless Database 的可用引擎版本
- `StorageType` 至 `aurora-iopt1`
- `ClusterScalabilityType` 至 `limitless`
- `MasterUsername`
- `MasterUserPassword`
- `EnablePerformanceInsights` 至 `True`
- `PerformanceInsightsRetentionPeriod` 到至少 `31` 天
- `MonitoringInterval` 到非 `0` 的值
- `MonitoringRoleArn` 到允許 RDS 將增強型監控指標傳送到 Amazon CloudWatch Logs 之 IAM 角色的 Amazon Resource Name (ARN)

為 Aurora PostgreSQL Limitless Database 修改資料庫叢集時，開啟 Database Insights 的標準模式

為 Aurora PostgreSQL Limitless Database 修改資料庫時，開啟 Database Insights。

> **Note**
>
> 若要啟用 Database Insights，資料庫叢集中的每個資料庫執行個體都必須具有相同的 Performance Insights 和增強型監控設定。

Console

在此主控台中，您可以在建立資料庫叢集時開啟 Database Insights 的標準模式。Database Insights 的設定適用於資料庫叢集中的所有資料庫執行個體。

使用主控台修改資料庫叢集時，開啟 Database Insights 的標準模式

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 選擇 Databases (資料庫)。
3. 選擇資料庫叢集，然後選擇修改。
4. 在 Database Insights 區段中，選取標準模式。然後選擇下列選項：
   - Retention (保留) – 保留 Performance Insights 資料的時間。若要為 Aurora PostgreSQL Limitless Database 建立資料庫叢集，保留期間必須至少為 31 天。
   - AWS KMS key – 指定您的 KMS 金鑰。績效詳情使用您的 KMS 金鑰來加密所有可能的敏感資料。將會對傳輸中與靜態資料進行加密。如需更多詳細資訊，請參閱 加密 Amazon Aurora 資源。
5. 選擇 Continue (繼續)。
6. 在 Scheduling of Modifications (修改排程) 中，選擇 Apply immediately (立即套用)。如果您選擇在下一個排定的維護時段套用，則資料庫會忽略此設定，並立即開啟 Database Insights 的標準模式。
7. 選擇修改叢集。

AWS CLI

若要在修改資料庫叢集時開啟 Database Insights 的標準模式，請呼叫 modify-db-cluster AWS CLI 命令並提供下列值：

- `--db-cluster-identifier`：資料庫叢集的識別符。
- `--database-insights-mode standard` 以開啟 Database Insights 的標準模式。
- `--enable-performance-insights` 以開啟 Database Insights 的 Performance Insights。
- `--performance-insights-retention-period`：資料庫叢集資料的保留期間。若要開啟 Database Insights 的標準模式，保留期必須至少為 31 天。

下列範例會在修改資料庫叢集時啟用 Database Insights 的標準模式。

針對 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --database-insights-mode standard \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 31
```

針對 Windows：

```
aws rds modify-db-cluster ^
    --database-insights-mode standard ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 31
```

RDS API

若要在修改資料庫叢集時開啟 Database Insights 的標準模式，請為 ModifyDBCluster Amazon RDS API 操作指定下列參數。

- `DatabaseInsightsMode` 至 `standard`
- `EnablePerformanceInsights` 至 `True`
- `PerformanceInsightsRetentionPeriod` 到至少 31 天

使用 Amazon CloudWatch Logs 監控 Aurora PostgreSQL Limitless Database

在啟用 Aurora PostgreSQL Limitless Database 時，需要將 PostgreSQL 日誌匯出至 CloudWatch Logs。您可以在 CloudWatch Logs Insights 中存取和分析這些日誌，類似於存取標準 Aurora PostgreSQL 資料庫叢集的 PostgreSQL 日誌。如需更多詳細資訊，請參閱 使用 CloudWatch Logs Insights 分析 PostgreSQL 日誌。

資料庫叢集的日誌群組名稱與 Aurora PostgreSQL 中的名稱相同：

```
/aws/rds/cluster/DB_cluster_ID/postgresql
```

資料庫碎片群組的日誌群組名稱採用下列形式：

```
/aws/rds/cluster/DB_cluster_ID/DB_shard_group_ID/postgresql
```

每個節點 (路由器或碎片) 都有日誌串流。其名稱格式如下：

```
[DistributedTransactionRouter|
DataAccessShard]/node_cluster_serial_ID-node_instance_serial_ID/n
```

例如：

- 路由器：`DistributedTransactionRouter/6-6.2`
- 碎片：`DataAccessShard/22-22.0`

> **Note**
>
> 您無法如在資料庫叢集所做一樣，直接在 RDS 主控台、AWS CLI 或 RDS API 中檢視資料庫碎片群組的 PostgreSQL 日誌檔案。您必須使用 CloudWatch Logs Insights 來檢視這些檔案。

使用增強型監控來監控 Aurora PostgreSQL Limitless Database

啟用 Aurora PostgreSQL Limitless Database 時需要增強型監控。您可以使用此功能來即時監控 Limitless Database 資料庫執行個體的作業系統。

Aurora 在 CloudWatch Logs 中發佈增強型監控指標。一些可用的關鍵指標包括資料庫連線、儲存空間用量和查詢延遲。這些指標可協助判斷效能瓶頸。

如需增強式監控指標的詳細資訊，請參閱 Aurora 的作業系統指標。

使用 Performance Insights 監控 Aurora PostgreSQL Limitless Database

使用 Performance Insights 來監控 Aurora PostgreSQL Limitless Database 叢集。Performance Insights 在 Aurora PostgreSQL Limitless Database 的運作方式與標準 Aurora 資料庫叢集的運作方式類似。不過，您可以在 Aurora PostgreSQL Limitless Database 的碎片群組層級追蹤指標。

要追蹤的兩個主要 Performance Insights 指標如下：

- 資料庫負載：測量資料庫中的活動程度。績效詳情中的關鍵指標為 DBLoad，每秒收集一次。

  Performance Insights 中 DBLoad 指標的單位是平均作用中工作階段 (AAS)。為了取得平均作用中工作階段，績效詳情會取樣並行執行查詢的工作階段數目。AAS 是將特定時段內的工作階段總數除以樣本總數。如需 DBLoad 和 AAS 的詳細資訊，請參閱資料庫負載。
- CPU 上限：資料庫可用的運算能力上限。若要查看作用中的工作階段是否超過最大 CPU，請查看其與 Max vCPU 數線的關係。Max vCPU 數值由資料庫執行個體的 vCPU (虛擬 CPU) 核心數目決定。如需 Max vCPU 的詳細資訊，請參閱最大 CPU。

此外，您可以將 DBLoad 指標「分割」為維度，這是指標的子類別。最有用的維度如下：

- 熱門執行個體：以遞減順序顯示執行個體 (碎片和路由器) 的相對資料庫負載。
- 等待事件：導致 SQL 陳述式等待特定事件發生後，才能繼續執行。等待事件指出工作受阻的位置。
- 熱門 SQL：顯示哪些查詢對資料庫負載的貢獻最多。

如需 Performance Insights 維度的詳細資訊，請參閱維度。

下圖顯示資料庫碎片群組的熱門執行個體維度。



主題

- 使用 Performance Insights 儀表板分析 Aurora PostgreSQL Limitless Database 的資料庫負載

使用 Performance Insights 儀表板分析 Aurora PostgreSQL Limitless Database 的資料庫負載

使用 Performance Insights，您可以在碎片群組層級和執行個體層級追蹤 Aurora PostgreSQL Limitless Database 的指標。分析 Aurora PostgreSQL Limitless Database 的資料庫負載時，建議您將每個碎片和路由器的資料庫負載與最大 vCPU 進行比較。

> **Note**
>
> Aurora PostgreSQL Limitless Database 一律啟用 Performance Insights 和增強型監控。Limitless Database 的 Performance Insights 資料的最短保留期間為 31 天 (1 個月)。

絕對檢視會顯示平均作用中工作階段 (AAS) 的數量和預估 vCPU。相對檢視會顯示 AAS 與預估 vCPU 的比率。

主題

- 使用 Performance Insights 儀表板分析 Aurora PostgreSQL Limitless Database 的相對資料庫負載
- 使用 Performance Insights 儀表板分析 Aurora PostgreSQL Limitless Database 的按等待資料庫負載
- 使用 Performance Insights 儀表板分析 Aurora PostgreSQL Limitless Database 的負載分配

使用 Performance Insights 儀表板分析 Aurora PostgreSQL Limitless Database 的相對資料庫負載

您可能想要透過追蹤相對資料庫負載，改善 Aurora PostgreSQL Limitless Database 的效能。若要按執行個體來分析 Aurora PostgreSQL Limitless Database 的相對資料庫負載，請使用下列程序。

使用主控台分析 Aurora PostgreSQL Limitless Database 的相對資料庫負載

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇 Aurora PostgreSQL Limitless Database。Performance Insights 儀表板會針對該 Aurora PostgreSQL Limitless Database 顯示。
4. 在資料庫負載區段中，選擇配量依據旁邊的執行個體。若要查看 Aurora PostgreSQL Limitless Database 中所有執行個體的平均作用中工作階段 (AAS) 與 vCPU 核心的比率，請針對檢視為選擇相對。

   平均作用中工作階段圖表顯示 Aurora PostgreSQL Limitless Database 中執行個體的資料庫負載。



5. 若要檢視熱門執行個體，請選擇熱門執行個體索引標籤。

   在下列範例中，資料庫負載最高的執行個體為 DTR-2-2。



6. (選用) 若要分析 Aurora PostgreSQL Limitless Database 中執行個體的資料庫負載，請在執行個體欄中選擇執行個體名稱。若要檢視 DTR-2-2 的資料庫負載，請在執行個體欄中選擇 DTR-2-2。

> **Note**
>
> 您只能針對 Aurora PostgreSQL Limitless Database 中的執行個體檢視 Performance Insights 指標。

使用 Performance Insights 儀表板分析 Aurora PostgreSQL Limitless Database 的按等待資料庫負載

您可能想要透過追蹤等待事件，改善 Aurora PostgreSQL Limitless Database 的效能。若要按等待事件來分析 Aurora PostgreSQL Limitless Database 的資料庫負載，請使用下列程序。

使用主控台按等待分析 Aurora PostgreSQL Limitless Database 的資料庫負載

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇 Aurora PostgreSQL Limitless Database。Performance Insights 儀表板會針對該 Aurora PostgreSQL Limitless Database 顯示。
4. 在資料庫負載區段中，針對配量依據選擇等待。若要檢視 AAS 的數量和估計 vCPU，請針對檢視為選擇絕對。

   平均作用中工作階段圖表顯示 Aurora PostgreSQL Limitless Database 中執行個體的資料庫負載。



5. 向下捲動至 Top SQL (最高 SQL) 索引標籤。

   在下列範例中，按等待之負載最高的 SQL 陳述式是 DELETE 陳述式。



6. 選擇 SQL 陳述式，展開到其元件陳述式中。

   在下列範例中，SELECT 陳述式有 3 個元件陳述式。



使用 Performance Insights 儀表板分析 Aurora PostgreSQL Limitless Database 的負載分配

您可能想要平衡 Aurora PostgreSQL Limitless Database 上執行個體的負載分配。若要分析 Aurora PostgreSQL Limitless Database 上執行個體的負載分配，請使用下列程序。

使用主控台分析 Aurora PostgreSQL Limitless Database 上執行個體的負載分配

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇 Aurora PostgreSQL Limitless Database。Performance Insights 儀表板會針對該 Aurora PostgreSQL Limitless Database 顯示。
4. 在資料庫負載區段中，選擇配量依據旁邊的執行個體。若要檢視 Aurora PostgreSQL Limitless Database 中所有執行個體的 AAS 數量和估計 vCPU，請針對檢視為選擇絕對。

   平均作用中工作階段圖表顯示 Aurora PostgreSQL Limitless Database 中執行個體的資料庫負載。



5. 若要查看 Aurora PostgreSQL Limitless Database 中執行個體負載分配的圖表，請選擇負載分配索引標籤。

   在下列範例中，資料庫負載最高的執行個體為 DTR-2-2。


Dimensions
Load distribution
Load distribution of the instances on DB shard group from May 13, 2024, 00:00 UTC to May 13, 2024, 23:59 UTC
Other instances
779 AAS (5.72%)
DAS-6-6
220 AAS (1.62%)
DAS-7-7
244 AAS (1.79%)
DTR-4-4
1198 AAS (8.80%)
13621
Total AAS
DTR-2-2
5655 AAS (41.52%)
DAS-9-9
5525 AAS (40.56%)
DTR-2-2
DAS-9-9
DTR-4-4
DAS-7-7
DAS-6-6
Other instances

使用 Amazon GuardDuty RDS 保護監控 Aurora PostgreSQL Limitless Database

Amazon GuardDuty 是一種威脅偵測服務，可協助保護您的帳戶、容器、工作負載和 AWS 環境中的資料。GuardDuty 使用機器學習 (ML) 模型，以及異常和威脅偵測功能，持續監控不同的日誌來源和執行時期活動，以識別環境中的潛在安全風險和惡意活動和排定其優先順序。

GuardDuty RDS Protection 會分析和描述登入事件，以找出 Amazon Aurora 資料庫的潛在存取威脅。當您開啟 RDS Protection 時，GuardDuty 會從您的 Aurora 資料庫取用 RDS 登入事件。RDS Protection 會監控這些事件並描述它們，以找出潛在的內部威脅或外部演員。

如需 Aurora 中 GuardDuty RDS 保護的詳細資訊，請參閱使用 Amazon GuardDuty RDS Protection for Amazon Aurora監控威脅。

如需了解如何啟用 GuardDuty RDS Protection，請參閱《Amazon GuardDuty 使用者指南》中的 GuardDuty RDS Protection。

Aurora PostgreSQL Limitless Database 的函數和檢視

Aurora PostgreSQL Limitless Database 已新增函數和檢視。其是以對應的 Aurora PostgreSQL 函數和檢視為基礎。

> **Note**
>
> 如果有進行中的交易，某些統計資料可能會傳回不一致的結果。

主題

- Aurora PostgreSQL Limitless Database 函數
- Aurora PostgreSQL Limitless Database 檢視

Aurora PostgreSQL Limitless Database 函數

下表顯示 Aurora PostgreSQL Limitless Database 的新函數。

> **Note**
>
> 此資料表中列出的函數位於 `rds_aurora` 結構描述中。使用 Limitless Database 函數時，請務必包含完整的物件名稱：`rds_aurora`*`object_name`*。

Aurora PostgreSQL Limitless Database 函數	對應的 Aurora PostgreSQL 函數
limitless_backend_dsid	pg_backend_pid
limitless_cancel_session	pg_cancel_backend
limitless_stat_clear_snapshot	pg_stat_clear_snapshot
limitless_stat_database_size	pg_database_size
limitless_stat_get_snapshot_timestamp	pg_stat_get_snapshot_timestamp
limitless_stat_prepared_xacts	pg_prepared_xacts

Aurora PostgreSQL Limitless Database 函數	對應的 Aurora PostgreSQL 函數
limitless_stat_relation_sizes	pg_indexes_size、pg_relation_size、pg_table_size、pg_total_relation_size
limitless_stat_reset	pg_stat_reset
limitless_stat_statements_reset	pg_stat_statements_reset
limitless_stat_system_waits	aurora_stat_system_waits
limitless_terminate_session	pg_terminate_backend
limitless_wait_report	aurora_wait_report

下列範例提供 Aurora PostgreSQL Limitless Database 函數的詳細資訊。如需 PostgreSQL 函數的詳細資訊，請參閱 PostgreSQL 文件中的函數和運算子。

limitless_backend_dsid

`limitless_backend_dsid` 函數會傳回目前工作階段的分散式工作階段 ID。分散式工作階段會在資料庫碎片群組中的路由器執行，以及涉及資料庫碎片群組中一或多個碎片的後端程序。

下列範例示範如何使用 `limitless_backend_dsid` 函數。

```
SELECT rds_aurora.limitless_backend_dsid();

limitless_backend_dsid
------------------------
8CACD7B04D0FC2A5
(1 row)
```

limitless_cancel_session

`limitless_cancel_session` 函數的運作方式類似於 `pg_cancel_backend`，但會嘗試透過傳送 SIGINT (中斷訊號)，來取消與提供的分散式工作階段 ID 相關的所有後端程序。

輸入參數如下：

- `distributed_session_id` (文字)：要取消的分散式工作階段 ID。

輸出參數如下：

- subcluster_id (文字)：此程序所屬的子叢集 ID。
- pid (文字)：後端程序 ID。
- success (布林值)：取消是否成功。

下列範例示範如何使用 limitless_cancel_session 函數。

```
SELECT * FROM rds_aurora.limitless_cancel_session('940CD5C81E3C796B');

 subcluster_id |  pid  | success
---------------+-------+---------
             1 | 26920 | t
(1 row)
```

limitless_stat_clear_snapshot

limitless_stat_clear_snapshot 函數會捨棄所有節點上的目前統計資料快照或快取資訊。

下列範例示範如何使用 limitless_stat_clear_snapshot 函數。

```
SELECT rds_aurora.limitless_stat_clear_snapshot();
```

limitless_stat_database_size

limitless_stat_database_size 函數會傳回資料庫碎片群組中資料庫的大小。

輸入參數如下：

- dbname (名稱)：要取得其大小的資料庫。

輸出參數如下：

- subcluster_id (文字)：此程序所屬的子叢集 ID。
- subcluster_type (文字)：此程序所屬的子叢集類型：router 或 shard。
- db_size：此子叢集中的資料庫大小 (以位元組為單位)。

下列範例示範如何使用 limitless_stat_database_size 函數。

```
SELECT * FROM rds_aurora.limitless_stat_database_size('postgres_limitless');

 subcluster_id | subcluster_type | db_size
---------------+-----------------+----------
```

```
             1 | router          |  8895919
             2 | router          |  8904111
             3 | shard           | 21929391
             4 | shard           | 21913007
             5 | shard           | 21831087
(5 rows)
```

limitless_stat_get_snapshot_timestamp

`limitless_stat_get_snapshot_timestamp` 函數會傳回目前統計資料快照的時間戳記，或如果未拍攝統計資料快照則傳回 NULL。如果 `stats_fetch_consistency` 設定為 `snapshot`，則會在交易中第一次存取累積統計資料時拍攝快照。傳回來自所有節點的快照時間戳記合併檢視。`subcluster_id` 和 `subcluster_type` 欄會顯示資料來自哪個節點。

下列範例示範如何使用 `limitless_stat_get_snapshot_timestamp` 函數。

```
SELECT * FROM rds_aurora.limitless_stat_get_snapshot_timestamp();

 subcluster_id | subcluster_type | snapshot_timestamp
---------------+-----------------+--------------------
             1 | router          |
             2 | router          |
             3 | shard           |
             4 | shard           |
             5 | shard           |
(5 rows)
```

limitless_stat_prepared_xacts

`limitless_stat_prepared_xacts` 函數會傳回目前準備進行兩階段遞交之所有節點的交易相關資訊。如需詳細資訊，請參閱 PostgreSQL 文件中的 pg_prepared_xacts。

下列範例示範如何使用 `limitless_stat_prepared_xacts` 函數。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_prepared_xacts;

 subcluster_id | subcluster_type | transaction_id |             gid              |
         prepared            |  owner_id  |    database_id
---------------+-----------------+----------------+------------------------------
+-------------------------------+------------+--------------------
 8             | shard           |        5815978 | 7_4599899_postgres_limitless |
 2024-09-03 15:51:17.659603+00 | auroraperf | postgres_limitless
```

```
 12            | shard           |        4599138 | 7_4599899_postgres_limitless |
 2024-09-03 15:51:17.659637+00 | auroraperf | postgres_limitless
(2 rows)
```

limitless_stat_relation_sizes

`limitless_stat_relation_sizes` 函數會傳回資料庫碎片群組中資料表的不同大小。

輸入參數如下：

- `relnspname` (名稱)：內含資料表的結構描述名稱。
- `relname` (名稱)：資料表的名稱。

輸出參數如下：

- `subcluster_id` (文字)：此程序所屬的子叢集 ID。
- `subcluster_type` (文字)：此程序所屬的子叢集類型：`router` 或 `shard`。
- `main_size`：此節點中主要資料分支的大小 (以位元組為單位)。
- `fsm_size`：此節點中資料表可用空間映射大小 (以位元組為單位)。
- `vm_size`：此節點中資料表可見度映射大小 (以位元組為單位)。
- `init_size`：此節點中資料表初始化的大小 (以位元組為單位)。
- `toast_size`：此分支中與資料表相關聯的快顯資料表大小 (以位元組為單位)。
- `index_size`：此節點中資料表所有索引的大小 (以位元組為單位)。
- `total_size`：此節點中資料表所有區段的大小 (以位元組為單位)。

下列範例示範如何使用 `limitless_stat_relation_sizes` 函數 (部分欄已省略)。

```
SELECT * FROM rds_aurora.limitless_stat_relation_sizes('public','customers');

 subcluster_id | subcluster_type | main_size | fsm_size | vm_size | toast_size |
 table_size | total_size
---------------+-----------------+-----------+----------+---------+------------
+------------+------------
             1 | router          |         0 |        0 |       0 |          0 |
   0 |          0
             2 | router          |         0 |        0 |       0 |          0 |
   0 |          0
             3 | shard           |   4169728 |  4177920 | 1392640 |    1392640 |
 11132928 |   11132928
```

```
             4 | shard           |   4169728 |  4177920 | 1392640 |    1392640 |
 11132928 |   11132928
             5 | shard           |   3981312 |  4227072 | 1409024 |    1409024 |
 11026432 |   11026432
(5 rows)
```

limitless_stat_reset

`limitless_stat_reset` 函數會將目前資料庫的所有統計資料計數器重設為零 (0)。如果已啟用 `track_functions`，`limitless_stat_database` 中的 `stats_reset` 欄會顯示上次重設資料庫統計資料的時間。根據預設，`limitless_stat_reset` 的執行只能由超級使用者進行。可以使用 EXECUTE 權限向其他使用者授予許可。

下列範例示範如何使用 `limitless_stat_reset` 函數。

```
SELECT tup_inserted, tup_deleted FROM pg_stat_database
WHERE datname = 'postgres_limitless';

 tup_inserted | tup_deleted
--------------+-------------
          896 |           0
(1 row)

SELECT rds_aurora.limitless_stat_reset();

limitless_stat_reset
----------------------
(1 row)

SELECT tup_inserted, tup_deleted FROM pg_stat_database
WHERE datname = 'postgres_limitless';

tup_inserted | tup_deleted
-------------+-------------
           0 |           0
(1 row)
```

limitless_stat_statements_reset

`limitless_stat_statements_reset` 函數會捨棄 `limitless_stat_statements` 目前為止收集的統計資料 (對應於指定的 `username`、`dbname`、`distributed_query_id`和 `queryid` 參數)。如果未指定任何參數，則會針對每個參數使用預設值 "" 或 0 (無效)，以及重設與其他參數相

符的統計資料。如果未指定參數，或所有指定的參數都是 "" 或 0 (無效)，則函數會捨棄所有統計資料。如果 limitless_stat_statements 檢視中的所有統計資料已遭捨棄，則函數也會重設 limitless_stat_statements_info 檢視中的統計資料。

輸入參數如下：

- username (名稱)：查詢陳述式的使用者。
- dbname (名稱)：執行查詢的資料庫。
- distributed_query_id (bigint)：協調器節點中父查詢的查詢 ID。如果這是父查詢，則此欄為 NULL。協調器節點會將分散式查詢 ID 向下推送至參與者節點。因此，對於參與者節點，分散式查詢 ID 和查詢 ID 的值會有所不同。
- queryid (bigint)：陳述式的查詢 ID。

下列範例示範如何使用 limitless_stat_statements_reset 函數來重設 limitless_stat_statements 收集的所有統計資料。

```
SELECT rds_aurora.limitless_stat_statements_reset();
```

limitless_stat_system_waits

limitless_stat_system_waits 函數會從 aurora_stat_system_waits 傳回等待事件資料的合併檢視，該檢視會報告來自所有節點中執行個體中全系統等待活動。subcluster_id 和 subcluster_type 欄會顯示資料來自哪個節點。

下列範例示範如何使用 limitless_stat_system_waits 函數。

```
postgres_limitless=> SELECT *
FROM rds_aurora.limitless_stat_system_waits() lssw, pg_catalog.aurora_stat_wait_event()
 aswe
WHERE lssw.event_id=aswe.event_id and aswe.event_name='LimitlessTaskScheduler';

 subcluster_id | subcluster_type | type_id | event_id  | waits  |  wait_time   |
 event_name
---------------+-----------------+---------+-----------+--------+--------------
+------------------------
             1 | router          |      12 | 201326607 | 677068 | 616942216307 |
 LimitlessTaskScheduler
             2 | router          |      12 | 201326607 | 678586 | 616939897111 |
 LimitlessTaskScheduler
             3 | shard           |      12 | 201326607 | 756640 | 616965545172 |
 LimitlessTaskScheduler
```

```
             4 | shard           |      12 | 201326607 | 755184 | 616958057620 |
 LimitlessTaskScheduler
             5 | shard           |      12 | 201326607 | 757522 | 616963183539 |
 LimitlessTaskScheduler
(5 rows)
```

limitless_terminate_session

limitless_terminate_session 函數的運作方式類似於 pg_terminate_backend，但會嘗試透過傳送 SIGTERM (結束訊號)，來結束與提供的分散式工作階段 ID 相關的所有後端程序。

輸入參數如下：

- distributed_session_id (文字)：要結束的分散式工作階段 ID。

輸出參數如下：

- subcluster_id (文字)：此程序所屬的子叢集 ID。
- pid (文字)：後端程序 ID。
- success (布林值)：程序是否已成功結束。

下列範例示範如何使用 limitless_terminate_session 函數。

```
SELECT * FROM rds_aurora.limitless_terminate_session('940CD5C81E3C796B');

 subcluster_id |  pid  | success
---------------+-------+---------
             1 | 26920 | t
(1 row)
```

limitless_wait_report

limitless_wait_report 函數會從所有節點傳回一段時間內的等待事件活動。subcluster_id 和 subcluster_type 欄會顯示資料來自哪個節點。

輸出參數如下：

- subcluster_id (文字)：此程序所屬的子叢集 ID。
- subcluster_type (文字)：此程序所屬的子叢集類型：router 或 shard。

其餘欄與 aurora_wait_report 中的欄相同。

下列範例示範如何使用 limitless_wait_report 函數。

```
postgres_limitless=> select * from rds_aurora.limitless_wait_report();

 subcluster_id | subcluster_type | type_name | event_name | waits | wait_time |
 ms_per_wait | waits_per_xact | ms_per_xact
---------------+-----------------+-----------+------------+-------+-----------
+-------------+--------------- +-------------
             1 | router          | Client    | ClientRead |    57 | 741550.14 |
 13009.652 |           0.19 |    2505.237
             5 | shard           | Client    | ClientRead |    54 | 738897.68 |
 13683.290 |           0.18 |    2496.276
             4 | shard           | Client    | ClientRead |    54 | 738859.53 |
 13682.584 |           0.18 |    2496.147
             2 | router          | Client    | ClientRead |    53 | 719223.64 |
 13570.257 |           0.18 |    2429.810
             3 | shard           | Client    | ClientRead |    54 | 461720.40 |
 8550.378 |           0.18 |    1559.86
```

Aurora PostgreSQL Limitless Database 檢視

下表顯示 Aurora PostgreSQL Limitless Database 的新檢視。

> **Note**
>
> 此資料表中列出的檢視位於 `rds_aurora` 結構描述中。使用 Limitless Database 檢視時，請務必包含完整的物件名稱：`rds_aurora`*`object_name`*。

Aurora PostgreSQL Limitless Database 檢視	對應的 Aurora PostgreSQL 檢視
limitless_database	pg_database
limitless_locks	pg_locks
limitless_stat_activity	pg_stat_activity
limitless_stat_all_indexes	pg_stat_all_indexes
limitless_stat_all_tables	pg_stat_all_tables
limitless_stat_database	pg_stat_database
limitless_stat_progress_vacuum	pg_stat_progress_vacuum
limitless_stat_statements	pg_stat_statements
limitless_stat_subclusters	無
limitless_stat_statements_info	pg_stat_statements_info
limitless_statio_all_indexes	pg_statio_all_indexes
limitless_statio_all_tables	pg_statio_all_tables
limitless_tables	pg_tables
limitless_table_collocations	無
limitless_table_collocation_distributions	無

下列範例提供 Aurora PostgreSQL Limitless Database 檢視的詳細資訊。如需 PostgreSQL 檢視的詳細資訊，請參閱 PostgreSQL 文件中的檢視統計資料。

> **Note**
>
> 如果您有進行中的交易，某些統計資料檢視可能會傳回不一致的結果。

limitless_database

此檢視包含資料庫碎片群組中可用資料庫的相關資訊。例如：

```
postgres_limitless=> SELECT subcluster_id, subcluster_type, oid, datname, datacl
 FROM rds_aurora.limitless_database;

 subcluster_id | subcluster_type |  oid  |      datname       |
                                 datacl

---------------+-----------------+-------+--------------------
+-------------------------------------------------------------------------------------------
 2             | router          |     4 | template0          | {=c/
rdsadmin,rdsadmin=CTc/rdsadmin}
 2             | router          |     5 | postgres           |
 2             | router          | 16384 | rdsadmin           |
 {rdsadmin=CTc/rdsadmin,rds_aurora_limitless_metadata_admin=c/
rdsadmin,rds_aurora_limitless_heat_mgmt_admin=c/rdsadmin}
 2             | router          | 16477 | postgres_limitless |
 2             | router          |     1 | template1          | {=c/
rdsadmin,rdsadmin=CTc/rdsadmin}
 6             | shard           |     4 | template0          | {=c/
rdsadmin,rdsadmin=CTc/rdsadmin}
```

輸出參數如下：

- `subcluster_id` (文字)：子叢集 (節點) 的 ID
- `subcluster_type` (文字)：子叢集 (節點)、路由器或碎片的類型

其餘欄與 `pg_database` 中的欄相同。

limitless_locks

此檢視包含每個節點每個程序一個列。其提供鎖定相關資訊的存取權，該鎖定由資料庫伺服器中作用中程序所保留。

Example使用兩個交易建立鎖定

在此範例中，我們在兩個路由器同時執行兩個交易。

```
# Transaction 1 (run on router 1)
BEGIN;
SET search_path = public;
SELECT * FROM customers;
INSERT INTO customers VALUES (400,'foo','bar');

# Transaction 2 (run on router 2)
BEGIN;
SET search_path = public;
ALTER TABLE customers ADD COLUMN phone VARCHAR;
```

執行第一個交易。後續交易必須等到第一個交易完成。因此，系統會使用鎖定封鎖第二個交易。若要檢查其根本原因，我們會透過使用 limitless_stat_activity 聯結 limitless_locks 來執行命令。

```
# Run on router 2
SELECT distributed_session_id, state, usename, query, query_start
FROM rds_aurora.limitless_stat_activity
WHERE distributed_session_id in (
SELECT distributed_session_id
FROM rds_aurora.limitless_locks
WHERE relname = 'customers'
);

 distributed_session_id | state               | usename                 | query
                                  | query_start
------------------------+---------------------+-------------------------
+---------------------------------- -------------+------------------------------
 47BDE66E9A5E8477       | idle in transaction | limitless_metadata_admin | INSERT INTO
 customers VALUES (400,'foo','bar'); | 2023-04-13 17:44:45.152244+00
 2AD7F370202D0FA9       | active              | limitless_metadata_admin | ALTER TABLE
 customers ADD COLUMN phone VARCHAR; | 2023-04-13 17:44:55.113388+00
 47BDE66E9A5E8477       |                     | limitless_auth_admin     |
 <insufficient privilege>                       |
 2AD7F370202D0FA9       |                     | limitless_auth_admin     |
 <insufficient privilege>                       |
 47BDE66E9A5E8477       |                     | limitless_auth_admin     |
 <insufficient privilege>                       |
```

```
 2AD7F370202D0FA9       |                     | limitless_auth_admin     |
 <insufficient privilege>                        |
(6 rows)
```

Example明確建立鎖定

在此範例中，我們會明確建立鎖定，然後使用 limitless_locks 檢視來查看鎖定 (部分欄已省略)。

```
BEGIN;
SET search_path = public;
LOCK TABLE customers IN ACCESS SHARE MODE;
SELECT * FROM rds_aurora.limitless_locks WHERE relname = 'customers';

 subcluster_id | subcluster_type | distributed_session_id | locktype |      datname
   | relnspname |  relname  | virtualtransaction |  pid  |      mode
---------------+-----------------+------------------------+----------
+--------------------+------------+ ----------+--------------------+-------
+-----------------
             1 | router          | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 28/600787          | 59564 |
 AccessShareLock
             2 | router          | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 28/600405          | 67130 |
 AccessShareLock
             3 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 15/473401          | 27735 |
 AccessShareLock
             4 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 13/473524          | 27734 |
 AccessShareLock
             5 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 13/472935          | 27737 |
 AccessShareLock
             6 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 13/473015          | 48660 |
 AccessShareLock
(6 rows)
```

limitless_stat_activity

此檢視包含每個節點每個程序一個列。其可用於追蹤需要長時間的整體系統運作狀態和分類程序。例如：

```
postgres=# SELECT
    subcluster_id,
    subcluster_type,
    distributed_session_id,
    distributed_session_state,
    datname,
    distributed_query_id,
    is_sso_query
FROM
    rds_aurora.limitless_stat_activity
WHERE
    distributed_session_id in ('D2470C97E3D07E06', '5A3CD7B8E5FD13FF')
    order by  distributed_session_id;

 subcluster_id | subcluster_type | distributed_session_id | distributed_session_state |
      datname       | distributed_query_id | is_sso_query
---------------+-----------------+------------------------+---------------------------
+--------------------+----------------------+--------------
 2             | router          | 5A3CD7B8E5FD13FF       | coordinator               |
 postgres_limitless |                      | f
 3             | shard           | 5A3CD7B8E5FD13FF       | participant               |
 postgres_limitless |  6808291725541680947 |
 4             | shard           | 5A3CD7B8E5FD13FF       | participant               |
 postgres_limitless |  6808291725541680947 |
 2             | router          | D2470C97E3D07E06       | coordinator               |
 postgres_limitless |                      | t
 3             | shard           | D2470C97E3D07E06       | participant               |
 postgres_limitless |  4058400544464210222 |
(5 rows)
```

輸出參數如下：

- subcluster_id (文字)：此程序所屬的子叢集 ID。
- subcluster_type (文字)：此程序所屬的子叢集類型：router 或 shard。
- distributed_session_id (文字)：此程序所屬的分散式工作階段 ID。
- distributed_session_state (文字)：這是協調器、參與者還是獨立/非分散式的程序 (如 NULL 所顯示)。
- datname (文字)：此程序所連接的資料庫。
- distributed_query_id (bigint)：協調器節點中父查詢的查詢 ID。如果這是父查詢，則此欄為 NULL。協調器節點會將分散式查詢 ID 向下推送至參與者節點。因此，對於參與者節點，分散式查詢 ID 和查詢 ID 的值會有所不同。

- is_sso_query (文字)：這可讓我們知道查詢是否為單一碎片最佳化。

其餘欄與 pg_stat_activity 中的欄相同。

limitless_stat_all_indexes

此檢視包含資料庫碎片群組中索引的使用量統計資料。例如：

```
postgres_limitless=> SELECT schemaname, relname, indexrelname, idx_scan
  FROM rds_aurora.limitless_stat_all_indexes
  WHERE relname LIKE 'orders_ts%' ORDER BY indexrelname LIMIT 10;

 schemaname |    relname     |    indexrelname     | idx_scan
------------+----------------+---------------------+----------
 ec_sample  | orders_ts00001 | orders_ts00001_pkey |   196801
 ec_sample  | orders_ts00002 | orders_ts00002_pkey |   196703
 ec_sample  | orders_ts00003 | orders_ts00003_pkey |   196376
 ec_sample  | orders_ts00004 | orders_ts00004_pkey |   197966
 ec_sample  | orders_ts00005 | orders_ts00005_pkey |   195301
 ec_sample  | orders_ts00006 | orders_ts00006_pkey |   195673
 ec_sample  | orders_ts00007 | orders_ts00007_pkey |   194475
 ec_sample  | orders_ts00008 | orders_ts00008_pkey |   191694
 ec_sample  | orders_ts00009 | orders_ts00009_pkey |   193744
 ec_sample  | orders_ts00010 | orders_ts00010_pkey |   195421
(10 rows)
```

limitless_stat_all_tables

此檢視包含資料庫碎片群組中目前資料庫中所有資料表的統計資料。這在追蹤清空操作和資料處理語言 (DML) 操作時非常有用。例如：

```
postgres_limitless=> SELECT subcluster_id, subcluster_type, relname,
 n_ins_since_vacuum, n_tup_ins, last_vacuum
  FROM rds_aurora.limitless_stat_all_tables
  WHERE relname LIKE 'orders_ts%' ORDER BY relname LIMIT 10;

 subcluster_id | subcluster_type |    relname     | n_ins_since_vacuum | n_tup_ins |
 last_vacuum
---------------+-----------------+----------------+--------------------+-----------
+-------------
 5             | shard           | orders_ts00001 |              34779 |    196083 |
 5             | shard           | orders_ts00002 |              34632 |    194721 |
 5             | shard           | orders_ts00003 |              34950 |    195965 |
```

```
 5             | shard           | orders_ts00004 |              34745 |    197283 |
 5             | shard           | orders_ts00005 |              34879 |    195754 |
 5             | shard           | orders_ts00006 |              34340 |    194605 |
 5             | shard           | orders_ts00007 |              33779 |    192203 |
 5             | shard           | orders_ts00008 |              33826 |    191293 |
 5             | shard           | orders_ts00009 |              34660 |    194117 |
 5             | shard           | orders_ts00010 |              34569 |    195560 |
(10 rows)
```

輸出參數如下：

- subcluster_id (文字)：此程序所屬的子叢集 ID。
- subcluster_type (文字)：此程序所屬的子叢集類型：router 或 shard。
- relname (名稱)：資料表的名稱。

其餘欄與 pg_stat_all_tables 中的欄相同。

limitless_stat_database

此檢視包含資料庫碎片群組中所有資料庫的相關統計資料。每個節點每個資料庫傳回一個列。例如：

```
postgres_limitless=> SELECT
    subcluster_id,
    subcluster_type,
    datname,
    blks_read,
    blks_hit
FROM
    rds_aurora.limitless_stat_database
WHERE
    datname='postgres_limitless';
 subcluster_id | subcluster_type |      datname       | blks_read | blks_hit
---------------+-----------------+--------------------+-----------+----------
             1 | router          | postgres_limitless |       484 | 34371314
             2 | router          | postgres_limitless |       673 | 33859317
             3 | shard           | postgres_limitless |      1299 | 17749550
             4 | shard           | postgres_limitless |      1094 | 17492849
             5 | shard           | postgres_limitless |      1036 | 17485098
             6 | shard           | postgres_limitless |      1040 | 17437257
(6 rows)
```

輸出參數如下：

- subcluster_id (文字)：此程序所屬的子叢集 ID。
- subcluster_type (文字)：此程序所屬的子叢集類型：router 或 shard。
- datname (名稱)：資料庫的名稱。

其餘欄與 pg_stat_database 中的欄相同。

limitless_stat_progress_vacuum

此檢視包含持續清空操作的相關資訊。例如：

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_progress_vacuum;

-[ RECORD 1 ]----------+------------------
subcluster_id          | 3
subcluster_type        | shard
distributed_session_id | A56D96E2A5C9F426
pid                    | 5270
datname                | postgres
nspname                | public
relname                | customer_ts2
phase                  | vacuuming heap
heap_blks_total        | 130500
heap_blks_scanned      | 100036
heap_blks_vacuumed     | 0
index_vacuum_count     | 0
max_dead_tuples        | 11184810
num_dead_tuples        | 0

-[ RECORD 2 ]----------+------------------
subcluster_id          | 3
subcluster_type        | shard
distributed_session_id | 56DF26A89EC23AB5
pid                    | 6854
datname                | postgres
nspname                | public
relname                | sales_ts1
phase                  | vacuuming heap
heap_blks_total        | 43058
heap_blks_scanned      | 24868
heap_blks_vacuumed     | 0
index_vacuum_count     | 0
max_dead_tuples        | 8569523
num_dead_tuples        | 0
```

輸出參數如下：

- subcluster_id (文字)：此程序所屬的子叢集 ID。
- subcluster_type (文字)：此程序所屬的子叢集類型：router 或 shard。
- distributed_session_id (文字)：啟動清空操作之工作階段的識別符。
- datname (名稱)：正在執行清空的資料庫。
- nspname (名稱)：正在清空之資料表的結構描述名稱。如果要清空的資料表與使用者所連線的資料表不在相同的資料庫中，則為 null。
- relname (名稱)：正在清空的資料表名稱。如果要清空的資料表與使用者所連線的資料表不在相同的資料庫中，則為 null。

其餘欄與 pg_stat_progress_vacuum 中的欄相同。

limitless_stat_statements

此檢視提供一種方法，可用來追蹤所有節點上執行之所有 SQL 陳述式的規劃和執行統計資料。

> **Note**
>
> 您必須安裝 pg_stat_statements 延伸模組才能使用 limitless_stat_statements 檢視。
>
> ```
> -- CREATE EXTENSION must be run by a superuser
> CREATE EXTENSION pg_stat_statements;
>
> -- Verify that the extension is created on all nodes in the DB shard group
> SELECT distinct node_id
>     FROM rds_aurora.limitless_stat_statements
>     LIMIT 10;
> ```

下列範例示範 limitless_stat_statements 檢視的使用方式。

```
postgres_limitless=> SELECT
 subcluster_id,
 subcluster_type,
 distributedqueryid,
 username,
 dbname,
 sso_calls
```

```
FROM
 rds_aurora.limitless_stat_statements;

 subcluster_id | subcluster_type |  distributedqueryid  |              username
       |       dbname       | sso_calls
---------------+-----------------+----------------------
+--------------------------------------+--------------------+-----------
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         1
 3             | shard           | -7975178695405682176 | postgres
       | postgres_limitless |
[...]
```

輸出參數如下：

- subcluster_id (文字)：此程序所屬的子叢集 ID。
- subcluster_type (文字)：此程序所屬的子叢集類型：router 或 shard。
- distributedqueryid (bigint)：協調器節點中父查詢的查詢 ID。如果這是父查詢，則此欄為 NULL。協調器節點會將分散式查詢 ID 向下推送至參與者節點。因此，對於參與者節點，分散式查詢 ID 和查詢 ID 的值會有所不同。
- username (名稱)：查詢陳述式的使用者。
- dbname (名稱)：執行查詢的資料庫。
- sso_calls (名稱)：陳述式經過單一碎片最佳化的次數。

其餘的欄與 pg_stat_statements 中的欄相同。

limitless_stat_statements_info

此檢視包含 limitless_stat_statements 檢視的統計資料。每一個列都包含來自每個節點的 pg_stat_statements_info 檢視資料。subcluster_id 欄可識別每個節點。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_statements_info;
```

```
 subcluster_id | subcluster_type | dealloc |          stats_reset
---------------+-----------------+---------+-------------------------------
             1 | router          |       0 | 2023-06-30 21:22:09.524781+00
             2 | router          |       0 | 2023-06-30 21:21:40.834111+00
             3 | shard           |       0 | 2023-06-30 21:22:10.709942+00
             4 | shard           |       0 | 2023-06-30 21:22:10.740179+00
             5 | shard           |       0 | 2023-06-30 21:22:10.774282+00
             6 | shard           |       0 | 2023-06-30 21:22:10.808267+00
(6 rows)
```

輸出參數如下：

- subcluster_id (文字)：此程序所屬的子叢集 ID。

其餘的欄與 pg_stat_statements_info 中的欄相同。

limitless_stat_subclusters

此檢視包含路由器和其他節點之間的網路統計資料。其包含每對路由器和其他節點的列，例如：

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_subclusters;

 orig_subcluster | orig_instance_az | dest_subcluster | dest_instance_az |
 latency_us |       latest_collection       | failed_requests | received_bytes |
 sent_bytes | same_az_requests | cross_az_requests |     stat_reset_timestamp
-----------------+------------------+-----------------+------------------
+------------+-------------------------------+-----------------+---------------
+------------+------------------+-------------------+------------------------------
 3               | us-west-2b       | 2               | us-west-2a       |
 847 | 2024-10-07 17:25:39.518617+00 |              0 |       35668633 |   92090171
 |               0 |           302787 | 2024-10-05 12:39:55.239675+00
 3               | us-west-2b       | 4               | us-west-2b       |
 419 | 2024-10-07 17:25:39.546376+00 |              0 |      101190464 |  248795719
 |           883478 |                 0 | 2024-10-05 12:39:55.231218+00
 3               | us-west-2b       | 5               | us-west-2c       |
    1396 | 2024-10-07 17:25:39.52122+00  |              0 |       72864849 |
 172086292 |               0 |           557726 | 2024-10-05 12:39:55.196412+00
 3               | us-west-2b       | 6               | us-west-2c       |
 729 | 2024-10-07 17:25:39.54828+00  |              0 |       35668584 |   92090171
 |               0 |           302787 | 2024-10-05 12:39:55.247334+00
 3               | us-west-2b       | 7               | us-west-2a       |
    1702 | 2024-10-07 17:25:39.545307+00 |              0 |       71699576 |
 171634844 |               0 |           556278 | 2024-10-05 12:39:52.715168+00
```

```
2               | us-west-2a       | 3               | us-west-2b       |
868 | 2024-10-07 17:25:40.293927+00 |               0 |       35659611 |   92011872
|                0 |           302817 | 2024-10-05 12:39:54.420758+00
2               | us-west-2a       | 4               | us-west-2b       |
786 | 2024-10-07 17:25:40.296863+00 |               0 |      102437253 |  251838024
|                0 |           895060 | 2024-10-05 12:39:54.404081+00
2               | us-west-2a       | 5               | us-west-2c       |
   1232 | 2024-10-07 17:25:40.292021+00 |               0 |       71990027 |
168828110 |                0 |           545453 | 2024-10-05 12:39:36.769549+00
```

輸出參數如下：

- orig_subcluster (文字)：通訊源自的路由器 ID
- orig_subcluster_az (文字)：始發者路由器的可用區域 (AZ)
- dest_subcluster (文字)：目的地節點的 ID
- dest_subcluster_az (文字)：目的地節點上次收集的 AZ
- latency_us (bigint)：節點之間上次收集的網路延遲 (以微秒為單位)。如果節點無法連線，則值為 0。
- latest_collection (時間戳記)：目的地節點最新 AZ 集合和延遲的時間戳記
- failed_requests (bigint)：失敗的內部請求累積計數
- received_bytes (bigint)：從此節點收到的估計累積位元組數
- sent_bytes (bigint)：傳送至此節點的估計累積位元組數
- same_az_requests (bigint)：當內部資料庫請求與始發者路由器位於相同的可用區域時，對此節點的累積請求數量
- cross_az_requests (bigint)：當內部資料庫請求與始發者路由器位於不同的可用區域時，對此節點的累積請求數量
- stat_reset_timestamp (時間戳記)：此檢視的累積統計資料上次重設時的時間戳記

limitless_statio_all_indexes

此檢視包含資料庫碎片群組中所有索引的輸入/輸出 (I/O) 統計資料。例如：

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_statio_all_indexes WHERE
 relname like'customers_ts%';

 subcluster_id | subcluster_type | schemaname |      relname      |
 indexrelname             | idx_blks_read | idx_blks_hit
---------------+-----------------+------------+-------------------
+------------------------------------+ --------------+--------------
```

```
             3 | shard           | public     | customers_ts00002 |
 customers_ts00002_customer_name_idx |             1 |           0
             3 | shard           | public     | customers_ts00001 |
 customers_ts00001_customer_name_idx |             1 |           0
             4 | shard           | public     | customers_ts00003 |
 customers_ts00003_customer_name_idx |             1 |           0
             4 | shard           | public     | customers_ts00004 |
 customers_ts00004_customer_name_idx |             1 |           0
             5 | shard           | public     | customers_ts00005 |
 customers_ts00005_customer_name_idx |             1 |           0
             5 | shard           | public     | customers_ts00006 |
 customers_ts00006_customer_name_idx |             1 |           0
             6 | shard           | public     | customers_ts00007 |
 customers_ts00007_customer_name_idx |             1 |           0
             6 | shard           | public     | customers_ts00008 |
 customers_ts00008_customer_name_idx |             1 |           0
(8 rows)
```

limitless_statio_all_tables

此檢視包含資料庫碎片群組中所有資料表的輸入/輸出 (I/O) 統計資料。例如：

```
postgres_limitless=> SELECT
    subcluster_id,
    subcluster_type,
    schemaname,
    relname,
    heap_blks_read,
    heap_blks_hit
FROM
    rds_aurora.limitless_statio_all_tables
WHERE
    relname LIKE 'customers_ts%';

 subcluster_id | subcluster_type | schemaname |      relname      | heap_blks_read |
 heap_blks_hit
---------------+-----------------+------------+-------------------+----------------
+---------------
             3 | shard           | public     | customers_ts00002 |            305 |
      57780
             3 | shard           | public     | customers_ts00001 |            300 |
      56972
             4 | shard           | public     | customers_ts00004 |            302 |
      57291
```

```
             4 | shard           | public     | customers_ts00003 |            302 |
      57178
             5 | shard           | public     | customers_ts00006 |            300 |
      56932
             5 | shard           | public     | customers_ts00005 |            302 |
      57386
             6 | shard           | public     | customers_ts00008 |            300 |
      56881
             6 | shard           | public     | customers_ts00007 |            304 |
      57635
(8 rows)
```

limitless_tables

此檢視包含 Aurora PostgreSQL Limitless Database 中資料表的相關資訊。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_tables;

 table_gid | local_oid | schema_name | table_name  | table_status | table_type  |
 distribution_key
-----------+-----------+-------------+-------------+--------------+-------------
+------------------
         5 |     18635 | public      | placeholder | active       | placeholder |
         6 |     18641 | public      | ref         | active       | reference   |
         7 |     18797 | public      | orders      | active       | sharded     | HASH
 (order_id)
         2 |     18579 | public      | customer    | active       | sharded     | HASH
 (cust_id)
(4 rows)
```

limitless_table_collocations

此檢視包含共置碎片資料表的相關資訊。

在下列範例中，orders 和 customers 資料表會共置，而 users 和 followers 資料表會共置。共置的資料表具有相同的 collocation_id。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_table_collocations ORDER BY
 collocation_id;

 collocation_id | schema_name | table_name
----------------+-------------+------------
              2 | public      | orders
```

```
              2 | public      | customers
              5 | public      | users
              5 | public      | followers
(4 rows)
```

limitless_table_collocation_distributions

此檢視會顯示每個共置的金鑰分佈。

```
postgres_limitless=> SELECT * FROM
 rds_aurora.limitless_table_collocation_distributions ORDER BY collocation_id,
 lower_bound;

 collocation_id | subcluster_id |     lower_bound      |     upper_bound
----------------+---------------+----------------------+----------------------
              2 |             6 | -9223372036854775808 | -4611686018427387904
              2 |             5 | -4611686018427387904 |                    0
              2 |             4 |                    0 |  4611686018427387904
              2 |             3 |  4611686018427387904 |  9223372036854775807
              5 |             6 | -9223372036854775808 | -4611686018427387904
              5 |             5 | -4611686018427387904 |                    0
              5 |             4 |                    0 |  4611686018427387904
              5 |             3 |  4611686018427387904 |  9223372036854775807
(8 rows)
```

Aurora PostgreSQL Limitless Database 的等待事件

Aurora PostgreSQL 中的等待事件表示工作階段正在等待的資源，例如輸入/輸出 (I/O) 和鎖定。等待事件有助於找出工作階段正在等待資源的原因，以及識別瓶頸。如需更多詳細資訊，請參閱 Aurora PostgreSQL 等待事件。

Aurora PostgreSQL Limitless Database 有自己的等待事件，這些事件與路由器和碎片相關。其中許多適用於等待碎片完成任務的路由器。碎片等待事件包含執行中任務的詳細資訊。

主題

- 查詢等待事件
- Limitless Database 等待事件

查詢等待事件

如下列範例所示，您可以使用 limitless_stat_activity 檢視來查詢等待事件。

```
SELECT wait_event FROM rds_aurora.limitless_stat_activity WHERE
 wait_event_type='AuroraLimitless';

      wait_event
----------------------
 RemoteStatementSetup
 RemoteStatementSetup
(2 rows)
```

您也可以如下列範例所示，使用 aurora_stat_system_waits 函數列出每次等待事件所花費的等待次數和總時間。

```
postgres_limitless=> SELECT type_name,event_name,waits,wait_time
    FROM aurora_stat_system_waits()
    NATURAL JOIN aurora_stat_wait_event()
    NATURAL JOIN aurora_stat_wait_type()
    WHERE type_name='AuroraLimitless'
    ORDER BY wait_time DESC;

    type_name    |       event_name          |  waits  |  wait_time
-----------------+---------------------------+---------+-------------
 AuroraLimitless | RemoteStatementSetup      |    7518 | 75236507897
 AuroraLimitless | RemoteStatementExecution  |      40 |      132986
```

```
 AuroraLimitless | Connect                    |       5 |        1453
(3 rows)
```

Limitless Database 等待事件

下列等待事件適用於 Aurora PostgreSQL Limitless Database。您可以監控這些等待事件，以識別 Aurora PostgreSQL Limitless Database 處理的瓶頸。

主題

- IO:TwophaseFilePoolWrite 等待事件
- IO:TwophaseFilePoolRead 等待事件
- AuroraLimitless:Connect 等待事件
- AuroraLimitless:AsyncConnect 等待事件
- AuroraLimitless:RemoteStatementSetup 等待事件
- AuroraLimitless:RemoteDDLExecution 等待事件
- AuroraLimitless:RemoteStatementExecution 等待事件
- AuroraLimitless:FetchRemoteResults 等待事件
- AuroraLimitless:AsyncGetInitialResponse 等待事件
- AuroraLimitless:AsyncGetNextResponse 等待事件
- AuroraLimitless:AbortedCommandCleanup 等待事件
- AuroraLimitless:DistributedCommitPrepare 等待事件
- AuroraLimitless:DistributedCommit 等待事件
- AuroraLimitless:DistributedCommitPrepareThrottle 等待事件
- AuroraLimitless:PreparedTransactionResolution 等待事件
- AuroraLimitless:SendPreparedTransactionOutcome 等待事件
- AuroraLimitless:CommitClockBarrier 等待事件
- AuroraLimitless:SnapshotClockBarrier 等待事件
- AuroraLimitless:ReaderSnapshotClockBarrier 等待事件
- AuroraLimitless:GatherDistributedDeadlockGraph 等待事件
- AuroraLimitless:DistributedDeadlockDetection 等待事件
- AuroraLimitless:DistributedDeadlockAbort 等待事件
- AuroraLimitless:GatherRemoteStats 等待事件
- AuroraLimitless:GlobalSequenceRefresh 等待事件

- AuroraLimitless:GlobalVacuumTimeExchange 等待事件
- AuroraLimitless:DistributedTransactionMonitorGather 等待事件
- AuroraLimitlessActivity:AdminTaskSchedulerMain 等待事件
- AuroraLimitlessActivity:AdminTaskExecutorMain 等待事件
- AuroraLimitlessActivity:AdminTaskMonitorMain 等待事件
- AuroraLimitlessActivity:DatabaseCleanupMonitorMain 等待事件
- AuroraLimitlessActivity:TopologyCleanupMonitorMain 等待事件
- AuroraLimitlessActivity:ToplogyChangeMonitorMain 等待事件
- AuroraLimitlessActivity:DistributedTransactionMonitorMain 等待事件
- AuroraLimitlessActivity:GlobalVacuumMonitorMain 等待事件

IO:TwophaseFilePoolWrite 等待事件

等待在兩階段狀態檔案集區中寫入兩階段狀態檔案。這是 Aurora 特定的事件。

原因

PREPARED TRANSACTION 命令的執行程序 (包括 Limitless Database 分散式交易中的參與者)，都必須保留兩階段檔案中的交易狀態。Aurora 使用檔案集區來改善此操作的效能。

動作

這是同步寫入 I/O 操作，因此在此事件中的高延遲具有與 IO:XactSync 相似的原因，並且可以以相同的方式接受調查。如果使用 Limitless Database，您可能需要減少執行中的分散式交易數量。

IO:TwophaseFilePoolRead 等待事件

等待讀取兩階段狀態檔案集區中的兩階段狀態檔案。

原因

針對先前預備的交易執行 COMMIT PREPARED 命令的程序 (包括 Limitless Database 分散式交易的參與者)，都可能需要從兩階段檔案中讀取先前持續的交易狀態。Aurora 使用檔案集區來改善此操作的效能。

動作

這是讀取 I/O 操作。因此，此事件中的高延遲具有與 IO:DataFileRead 類似的原因，而且可以以相同的方式接受調查。如果使用 Limitless Database，您可能需要減少執行中的分散式交易數量。

AuroraLimitless:Connect 等待事件

程序正在等待與叢集中另一個節點的連線建立完成。

原因

在程序和遠端節點之間建立連線，以執行查詢、分散式交易和執行 DDL。

動作

減少叢集的同時連線數，或調校跨碎片查詢的使用。

AuroraLimitless:AsyncConnect 等待事件

此事件類似於 Connect，但代表等待與一組節點建立平行連線的程序。

原因

平行連線建立最常在執行 DDL 陳述式時完成。

動作

減少 DDL 陳述式的數量，或在同一工作階段中合併多個 DDL，以改善連線重複使用。

AuroraLimitless:RemoteStatementSetup 等待事件

程序正等待遠端查詢執行設定，例如游標開啟、關閉或預備陳述式建立。

原因

此等待事件會隨著碎片資料表上的掃描次數增加，其中陳述式無法進行單一碎片最佳化。

動作

最佳化查詢，以減少掃描操作的次數或提高單一碎片最佳化的資格。

AuroraLimitless:RemoteDDLExecution 等待事件

程序正等待遠端 DDL 命令完成。

原因

當您在資料庫碎片群組發出 DDL 命令時，必須先將其分佈至其他路由器和碎片節點，才能確認操作。有些 DDL 操作可能會長時間執行，因為資料必須適應結構描述變更。

動作

識別長時間執行的 DDL 命令，讓您可以將其最佳化。

AuroraLimitless:RemoteStatementExecution 等待事件

程序正等待遠端命令完成。

原因

SQL 命令正在遠端節點執行。此事件會經常因內部通訊而出現，例如 auto_analyze 和活動訊號檢查。

動作

識別使用 limitless_stat_statements 檢視的長時間執行命令。在許多情況下，這是預期事件 (對於背景工作者或內部程序尤其如此)，因此不需要採取任何動作。

AuroraLimitless:FetchRemoteResults 等待事件

程序正等待從遠端節點擷取列。

原因

從遠端資料表 (例如碎片或參考資料表) 擷取大量列時，此等待事件可能會增加。

動作

使用 limitless_stat_statements 檢視識別未最佳化的 SELECT 查詢。最佳化查詢以僅擷取必要的資料。您不能調校 rds_aurora.limitless_maximum_adaptive_fetch_size 參數。

AuroraLimitless:AsyncGetInitialResponse 等待事件

在查詢執行中使用管道模式時，程序正等待初始回應。

原因

對於具有單一碎片資料放置的查詢，這通常會在路由器進行碎片執行期間出現，並且這是正常執行的預期部分。

動作

無需採取進一步動作。

AuroraLimitless:AsyncGetNextResponse 等待事件

在查詢執行中使用管道模式時，程序正等待額外的回應。

原因

對於具有單一碎片資料放置的查詢，這通常會在路由器進行碎片執行期間出現，並且這是正常執行的預期部分。

動作

無需採取進一步動作。

AuroraLimitless:AbortedCommandCleanup 等待事件

程序正等待遠端清除查詢的結果。會將清除查詢發行給碎片節點，以在分散式交易結束時將其恢復至適當的狀態。

原因

交易因為發現錯誤或使用者發出明確的 ABORT 命令或取消執行中的查詢而中止時，就會完成交易清除。

動作

調查交易的取消原因。

AuroraLimitless:DistributedCommitPrepare 等待事件

程序正在遞交分散式交易，並等待所有參與者確認準備命令。

原因

修改多個節點的交易必須執行分散式遞交。在 DistributedCommitPrepare 中長時間等待可能是由於參與節點上 IO:TwophaseFilePoolWrite 事件長時間等待所致。

動作

減少交易數量，這些交易會修改多個節點上的資料。調查叢集其他節點中的 IO:TwophaseFilePoolWrite 事件。

AuroraLimitless:DistributedCommit 等待事件

程序正在遞交分散式交易，並等待主要參與者確認遞交命令。

原因

修改多個節點的交易必須執行分散式遞交。在 DistributedCommit 中長時間等待可能是由於主要參與者上 IO:XactSync 事件長時間等待所致。

動作

減少交易數量，這些交易會修改多個節點上的資料。調查叢集其他節點中的 IO:XactSync 事件。

AuroraLimitless:DistributedCommitPrepareThrottle 等待事件

程序正嘗試準備分散式交易，而且由於現有的預備交易而受到限流。

原因

修改多個節點的交易必須執行分散式遞交。在遞交通訊協定的過程中，這些交易中的參與者必須執行準備操作。Aurora 會限制並行準備的數量，如果超過此限制，程序將在 DistributedCommitPrepareThrottle 事件中等待。

動作

減少交易數量，這些交易會修改多個節點上的資料。在那些事件中因為時間增加而調查 IO:TwophaseFilePoolWrite 事件，可能會導致現有的預備交易累積，進而導致新的準備嘗試受到限流。

AuroraLimitless:PreparedTransactionResolution 等待事件

程序遇到元組，處於預備狀態的分散式交易會修改該元組。程序必須判斷分散式交易是否會出現在其快照中。

原因

修改多個節點的交易必須執行內含準備階段的分散式遞交。大量分散式交易或分散式遞交的延遲增加可能會導致其他程序遇到 PreparedTransactionResolution 等待事件。

動作

減少交易數量，這些交易會修改多個節點上的資料。調查分散式遞交相關事件，因為在那些事件中的時間增加，可能會延長分散式交易遞交路徑的延遲。您也可以調查網路和 CPU 負載。

AuroraLimitless:SendPreparedTransactionOutcome 等待事件

程序正在協調分散式交易的節點上執行，而另一個程序已查詢該交易的狀態，或該程序已遞交分散式交易，以及將成果傳送給參與者。

原因

遇到 PreparedTransactionResolution 等待事件的程序將查詢交易協調器。交易協調器上的回應會遇到 SendPreparedTransactionOutcome。

動作

減少交易數量，這些交易會修改多個節點上的資料。調查分散式遞交相關事件 IO:TwophaseFilePoolWrite 和 IO:TwophaseFilePoolRead 事件，因為那些事件可能會延長分散式交易遞交路徑的延遲。您也可以調查網路和 CPU 負載。

AuroraLimitless:CommitClockBarrier 等待事件

程序正在遞交交易而且必須等待，以確保叢集中所有節點的指派遞交時間都必定是過去的時間。

原因

CPU 或網路飽和度可能會導致時鐘偏離增加，導致在此等待事件中花費的時間增加。

動作

調查叢集中的 CPU 或網路飽和度。

AuroraLimitless:SnapshotClockBarrier 等待事件

程序已收到來自另一個節點的快照時間，該節點未來有時鐘，並正等待自己的時鐘到達該時間。

原因

這通常發生在程序從下推到碎片的函數接收結果後，且節點之間存在時鐘偏離時。CPU 或網路飽和度可能會導致時鐘偏離增加，導致在此等待事件中花費的時間增加。

動作

調查叢集中的 CPU 或網路飽和度。

AuroraLimitless:ReaderSnapshotClockBarrier 等待事件

此事件會在讀取節點發生。此程序正等待讀取節點重播寫入串流，以便套用程序快照時間之前發生的所有寫入。

原因

Aurora 複本延遲中的增加可能會導致此事件的等待時間增加。

動作

調查 Aurora 複本延遲。

AuroraLimitless:GatherDistributedDeadlockGraph 等待事件

程序正與其他節點通訊，以收集鎖定圖形作為分散式死鎖偵測的一部分。

原因

當程序等待鎖定時，會在等待超過 `rds_aurora.limitless_distributed_deadlock_timeout` 之後執行分散式死鎖檢查。

動作

調查應用程式中鎖定爭用的原因，並考慮調校 `rds_aurora.limitless_distributed_deadlock_timeout`。

AuroraLimitless:DistributedDeadlockDetection 等待事件

程序正與其他節點通訊，以偵測分散式死鎖。

原因

當程序等待鎖定時，會在等待超過 `rds_aurora.limitless_distributed_deadlock_timeout` 之後執行分散式死鎖檢查。

動作

調查應用程式中鎖定爭用的原因，並考慮調校 `rds_aurora.limitless_distributed_deadlock_timeout`。

AuroraLimitless:DistributedDeadlockAbort 等待事件

程序正與其他節點通訊，以中止選擇為分散式死鎖中受害者的工作階段。

原因

應用程式模式會產生分散式死鎖。

動作

調查導致分散式死鎖的應用程式模式。

AuroraLimitless:GatherRemoteStats 等待事件

此程序正從叢集中的其他節點收集統計資料。

原因

監控或活動查詢和檢視 (例如 `limitless_stat_activity`)，將從其他節點擷取統計資料。

動作

無需採取進一步動作。

AuroraLimitless:GlobalSequenceRefresh 等待事件

此程序正在產生新的序列值，必須向全域序列請求新的區塊。

原因

如果 `rds_aurora.limitless_sequence_chunk_size` 不足，產生高速率的序列值可能會導致在此事件中停滯。

動作

這是正常情況。如果您在此事件中看到時間過長，請考慮調校 `rds_aurora.limitless_sequence_chunk_size`。請參閱 Limitless Database 中序列的相關文件。

AuroraLimitless:GlobalVacuumTimeExchange 等待事件

程序正在交換快照資料以支援清空。

原因

Limitless Database 中的節點與其他節點交換最舊的作用中快照時間資料，以計算清空執行的正確截止時間。

動作

無需採取進一步動作。

AuroraLimitless:DistributedTransactionMonitorGather 等待事件

此程序是從其他節點收集交易中繼資料，以支援分散式交易清除。

原因

Limitless Database 中的節點與其他節點交換交易中繼資料，以判斷何時可以清除分散式交易狀態。

動作

無需採取進一步動作。

AuroraLimitlessActivity:AdminTaskSchedulerMain 等待事件

在任務排程器程序的主迴圈中等待。

AuroraLimitlessActivity:AdminTaskExecutorMain 等待事件

在任務執行器程序的主迴圈中等待。

AuroraLimitlessActivity:AdminTaskMonitorMain 等待事件

在任務監控程序的主迴圈中等待。

AuroraLimitlessActivity:DatabaseCleanupMonitorMain 等待事件

在資料庫清除監控程序的主迴圈中等待。

AuroraLimitlessActivity:TopologyCleanupMonitorMain 等待事件

在拓撲清理監控程序的主迴圈中等待。

AuroraLimitlessActivity:ToplogyChangeMonitorMain 等待事件

在拓撲變更監控程序的主迴圈中等待。

AuroraLimitlessActivity:DistributedTransactionMonitorMain 等待事件

在分散式交易監控程序的主迴圈中等待。

AuroraLimitlessActivity:GlobalVacuumMonitorMain 等待事件

等待全域清空監控程序的主迴圈。

使用 函數建置以提高效率

使用者定義的函數預設不會進行單一碎片最佳化，但可以設定為執行單一碎片操作。函數可以封裝邏輯，並確保以單一碎片最佳化的方式執行。

為什麼單一碎片操作很重要

資源使用率對於效能和成本效益至關重要。相較於跨碎片操作，單一碎片操作使用的資源明顯較少。例如，執行函數以插入一百萬個資料列時，單一碎片執行會使用大約 90.5 ACUs，相較於 126.5 ACUs 進行跨碎片執行，資源效率提升 35%。

單一碎片執行也提供：

- 輸送量比跨碎片操作高出 35%
- 更可預測的回應時間
- 隨著資料成長而提高可擴展性

單一碎片操作和函數

當符合下列任一先決條件時，函數會在碎片上執行：

- 函數會建立為不可變，並包含在單一碎片最佳化查詢中
- 函數由使用者分發

在碎片上執行的函數會執行和擴展更佳，因為它們會執行資料所在的位置。

函數和波動

若要檢查函數的波動，請在 PostgreSQL 的系統資料表上使用此查詢：

```
SELECT DISTINCT nspname, proname, provolatile
FROM pg_proc PRO
JOIN pg_namespace NSP ON PRO.pronamespace = NSP.oid
WHERE proname IN ('random', 'md5');
```

輸出範例：

```
  nspname   | proname | provolatile
------------+---------+-------------
```

```
 pg_catalog | md5     | i
 pg_catalog | random  | v
(2 rows)
```

在此範例中， md5() 是不可變的，並且random()是揮發性的。這表示包含 的單一碎片最佳化陳述式md5()會保持單一碎片最佳化，而包含 的陳述式random()則不會。

具有不可變函數的範例：

```
EXPLAIN ANALYZE
SELECT pg_catalog.md5('123')
FROM s1.t1
WHERE col_a = 776586194
  AND col_b = 654849524
  AND col_c = '3ac2f2affb02987159ccd6ebd23e1ae5';
```

```
                          QUERY PLAN
-----------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
               (actual time=3.409..3.409 rows=1 loops=1)
 Single Shard Optimized
 Planning Time: 0.313 ms
 Execution Time: 4.253 ms
(4 rows)
```

使用揮發性函數的範例：

```
EXPLAIN ANALYZE
SELECT pg_catalog.random()
FROM s1.t1
WHERE col_a = 776586194
  AND col_b = 654849524
  AND col_c = '3ac2f2affb02987159ccd6ebd23e1ae5';
```

```
                          QUERY PLAN
-----------------------------------------------------
 Foreign Scan on t1_fs00001 t1
   (cost=100.00..15905.15 rows=1 width=8)
   (actual time=0.658..0.658 rows=1 loops=1)
 Planning Time: 0.263 ms
 Execution Time: 2.892 ms
```

```
(3 rows)
```

輸出顯示 md5() 已下推並執行為單一碎片最佳化，而 random() 則否。

分發 函數

僅在一個碎片上存取資料的函數應該在該碎片上執行，以獲得效能優勢。必須分發函數，且函數簽章必須包含完整的碎片索引鍵 - 碎片索引鍵中的所有資料欄都必須作為參數傳遞至函數。

函數範例：

```
CREATE OR REPLACE FUNCTION s1.func1(
    param_a bigint,
    param_b bigint,
    param_c char(100)
)
RETURNS int AS $$
DECLARE
    res int;
BEGIN
    SELECT COUNT(*) INTO res
    FROM s1.t1
    WHERE s1.t1.col_a = param_a
      AND s1.t1.col_b = param_b
      AND s1.t1.col_c = param_c;

    RETURN res;
END
$$ LANGUAGE plpgsql;
```

在分佈之前，函數不是單一碎片最佳化：

```
EXPLAIN ANALYZE
SELECT * FROM s1.func1(776586194, 654849524, '3ac2f2affb02987159ccd6ebd23e1ae5');
```

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Function Scan on func1  (cost=0.25..0.26 rows=1 width=4)
                         (actual time=37.503..37.503 rows=1 loops=1)
 Planning Time: 0.901 ms
 Execution Time: 51.647 ms
(3 rows)
```

若要分發函數：

```
SELECT rds_aurora.limitless_distribute_function(
    's1.func1(bigint,bigint,character)',
    ARRAY['param_a','param_b','param_c'],
    's1.t1'
);
```

分佈之後，函數會進行單一碎片最佳化：

```
EXPLAIN ANALYZE
SELECT * FROM s1.func1(776586194, 654849524, '3ac2f2affb02987159ccd6ebd23e1ae5');
```

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
               (actual time=4.332..4.333 rows=1 loops=1)
 Single Shard Optimized
 Planning Time: 0.857 ms
 Execution Time: 5.116 ms
(4 rows)
```

您可以檢查 中的sso_calls資料欄，以確認單一碎片最佳化rds_aurora.limitless_stat_statements：

```
subcluster_id | subcluster_type | calls | sso_calls |                query
--------------+-----------------+-------+-----------
+--------------------------------------
 2            | router          |     2 |         1 | SELECT * FROM s1.func1( $1, $2,
 $3 )
 3            | router          |     1 |         1 | SELECT * FROM s1.func1( $1, $2,
 $3 )
(2 rows)
```

函數和效率模式

執行靠近資料的邏輯更有效率，而且函數在達成此目標時扮演重要角色。使用 函數提高效率有兩種主要使用案例：

1. 從複雜資料擷取碎片金鑰，以叫用個別的單一碎片最佳化函數
2. 透過將跨碎片邏輯與單一碎片最佳化陳述式分開，將跨碎片工作負載轉換為單一碎片最佳化

從複雜資料擷取碎片索引鍵

考慮具有執行數個資料庫操作s3.func3(p_json_doc json)之簽章的函數。這些操作將在交易中跨越所有碎片的所有碎片中執行。如果 JSON 文件包含碎片索引鍵，您可以建置單一碎片最佳化函數來執行資料庫操作。

原始模式：

```
s3.func3(p_json_doc json)
    database operation 1;
    database operation 2;
    database operation 3;
```

最佳化模式：

```
s3.func3(p_json_doc json)
DECLARE
    v_a bigint;
BEGIN
    v_a := (p_json_doc->>'field_a')::bigint;
    SELECT s3.func3_INNER(v_a, p_json_doc);
END;
```

內部函數的執行位置：

```
s3.func3_INNER(p_a, p_json_doc)
    database operation 1 WHERE shard_key = p_a;
    database operation 2 WHERE shard_key = p_a;
    database operation 3 WHERE shard_key = p_a;
```

在此模式中，碎片金鑰會封裝在複雜的資料類型中，或從其他參數將其刪除。邏輯、資料存取和函數可以判斷、擷取或建構碎片金鑰，然後調用單一碎片最佳化函數，以僅執行有關單一碎片的操作。由於應用程式界面不會變更，因此最佳化比較容易測試。

從其他函數或資料延遲碎片索引鍵

當邏輯或資料存取計算或決定碎片索引鍵時，會套用另一個設計模式。這在大多數調用可在單一碎片上執行函數時非常有用，但偶爾需要跨碎片執行。

原始模式：

```
NEWORD(INTEGER, …) RETURNS NUMERIC
```

```
DECLARE
    all_whid_local := true;
    LOOP through the order lines
        Generate warehouse ID;
        IF generated warehouse ID == input warehouse ID
        THEN
            ol_supply_whid := input warehouse ID;
        ELSE
            all_whid_local := false;
            ol_supply_whid := generated warehouse ID;
        END IF;
        …
    END LOOP;
    …
    RETURN no_s_quantity;
```

具有不同函數的最佳化模式：

```
CREATE OR REPLACE FUNCTION NEWORD_sso(no_w_id INTEGER, …)
RETURNS NUMERIC
…
    RETURN no_s_quantity;
    …
END;
LANGUAGE 'plpgsql';

SELECT rds_aurora.limitless_distribute_function(
    'NEWORD_sso(int,…)',
    ARRAY['no_w_id'],
    'warehouse'
);

CREATE OR REPLACE FUNCTION NEWORD_crosshard(no_w_id INTEGER, …)
RETURNS NUMERIC
…
    RETURN no_s_quantity;
    …
END;
LANGUAGE 'plpgsql';
```

然後，讓主要函數呼叫單一碎片最佳化或跨碎片版本：

```
IF all_whid_local THEN
```

```
    SELECT NEWORD_sso(…) INTO no_s_quantity;
ELSE
    SELECT NEWORD_crosshard(…) INTO no_s_quantity;
END IF;
```

此方法可讓大多數調用受益於單一碎片最佳化，同時針對需要跨碎片執行的案例維持正確的行為。

檢查單一碎片操作

使用 EXPLAIN來驗證陳述式是否為單一碎片最佳化。輸出會明確報告「單一碎片最佳化」以進行最佳化操作。

分佈前的跨碎片調用：

```
                        QUERY PLAN
-----------------------------------------------------------------------
 Function Scan on func1  (cost=0.25..0.26 rows=1 width=4)
                          (actual time=59.622..59.623 rows=1 loops=1)
 Planning Time: 0.925 ms
 Execution Time: 60.211 ms
```

分佈後的單一碎片叫用：

```
                        QUERY PLAN
-----------------------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
               (actual time=4.576..4.577 rows=1 loops=1)
 Single Shard Optimized
 Planning Time: 1.483 ms
 Execution Time: 5.404 ms
```

執行時間的差異顯示了單一碎片最佳化的效能優勢。

執行個體特定的效能和資源監控

在執行個體層級上監控是了解連線扭曲、工作負載扭曲和資料扭曲的關鍵，以及何時新增路由器或分割碎片，以利用保留延遲來擴展更高輸送量。

概觀

當您的應用程式對 發出查詢時，該請求會在傳回結果之前周遊複雜的分散式系統。看似簡單的SELECT陳述式可能會觸動多個資料庫執行個體，每個執行個體在處理您的請求時都扮演不同的角色。了解此旅程，以及驅動它的執行個體，會改變您設計應用程式、解譯監控資料和診斷效能問題的方式。

本指南提供執行個體架構的深入技術洞見：

- 無限架構重新整理程式、路由器和碎片
- 何時及如何擴展每個執行個體類型以符合您的效能和容量需求
- 如何監控、疑難排解和最佳化執行個體層級效能
- 有效利用分散式架構的應用程式設計最佳實務

執行個體架構基本概念

透過跨兩種特殊執行個體類型的功能分離來實現水平可擴展性：

- 路由器執行個體提供協調層 - 它們接受用戶端連線、分析查詢、協調分散式操作和彙總結果。路由器是無狀態的，表示它們不會存放資料，而且可以在沒有資料遷移的情況下新增或移除。
- 碎片執行個體提供資料和運算層 - 它們存放資料表資料、對本機資料執行查詢，以及處理交易。碎片具有狀態，每個碎片都擁有由一致性雜湊決定的特定資料子集。

此區隔可讓 根據您的工作負載特性獨立擴展連線處理、查詢協調和資料儲存。

路由器和碎片比較

特性	路由器執行個體	碎片執行個體
主要角色	查詢協調和分佈	資料儲存和查詢執行
State	無狀態（無資料儲存）	狀態（擁有資料）

特性	路由器執行個體	碎片執行個體
可擴展性	立即新增/移除	需要重新平衡資料
資源焦點	用於協調的 CPU；中等記憶體	查詢的 CPU；快取的高記憶體
擴展觸發	高連線計數、分散式 txn 速率	高 CPU、資料磁碟區、查詢輸送量

監控執行個體效能

了解執行個體層級效能對於有效運作至關重要。執行個體特定的監控會顯示會影響效能的分佈模式：連線扭曲、工作負載扭曲和資料扭曲。

偵測扭曲

在理想的部署中，工作負載和資源會在執行個體之間平均分配。實際上，應用程式經常遇到扭曲，不均勻的分佈會集中在特定執行個體上的負載。

要監控的三種扭曲類型：

- 連線扭曲：跨路由器的用戶端連線分佈不平均
- 工作負載扭曲：由於熱碎片索引鍵，跨碎片的查詢負載不均勻
- 資料扭曲：由於碎片金鑰頻率，跨碎片的資料量不平均

Database Insights 負載分佈

評估執行個體層級運作狀態的最快方法是 Database Insights 的負載分佈檢視，可讓您立即了解 Active Sessions 如何在執行個體間分佈。

若要存取負載分佈：

1. 導覽至 RDS 主控台 → 您的無限叢集
2. 選取「績效詳情」索引標籤
3. 按一下「載入分佈」區段

運作狀態良好模式：負載在執行個體間相對平均分佈

- 路由器可能顯示稍微高於碎片的 AAS （協調開銷）
- 在彼此 20% 內的碎片 AAS 值表示平衡良好

關注模式：在特定執行個體上大幅集中

- 路由器負載 >70% 的一個路由器 → 連線扭曲
- 碎片負載 >50% 的一個碎片 → 工作負載或資料扭曲
- 碎片之間的大差異 → 調查碎片金鑰分佈

CloudWatch 指標

針對 Database Insights 以外的深入分析，CloudWatch 提供執行個體特定的指標，可顯示資源使用率模式。

具有維度的`ServerlessDatabaseCapacity`指標`DBShardGroupInstance`會顯示每個執行個體的 ACU 耗用量，提供資源使用率的最直接檢視。

調查時機：

- 路由器 ACU 變異數 >30% → 連線扭曲或跨碎片工作負載集中
- 碎片 ACU 變異 >40% → 資料或工作負載扭曲
- 任何執行個體一致達到最大 ACU → 容量限制

路由器監控和故障診斷

路由器可能會遇到兩個主要原因的效能問題：連線分佈不均和跨碎片工作負載集中。

分佈不均勻的工作階段

徵狀：一個路由器處理不成比例的連線份額

根本原因：DNS 快取會導致多個連線請求解析為相同的路由器端點。

最常見的期間：

- 使用 pgbench 等工具進行基準測試
- 連線集區初始化 （許多連線快速建立）

- 應用程式伺服器重新啟動

補救措施：

- 請務必使用主控台中指定的無限端點
- 手動平衡：擷取路由器端點，並將不同的應用程式連接到不同的路由器
- 對於 libpq 應用程式，請使用 功能 LOADBALANCEHOSTS
- 對於 JDBC 應用程式，請使用無限連線外掛程式
- 使用 NLB 來管理工作階段和分佈

碎片監控和疑難排解

碎片遇到三個主要原因的效能問題：資源限制、資料扭曲和工作負載扭曲。

碎片資源使用率

具有熱門碎片索引鍵的碎片將擁有更多資料和更高的工作負載。這將資訊清單顯示為資源使用率，即執行個體將耗用更多 ACUs。

修復策略：

1. 重新評估碎片金鑰選擇：檢閱碎片金鑰基數和存取模式。請考慮複合碎片索引鍵，以獲得更好的分佈。
2. 分割碎片：將負載分散到更多碎片執行個體

何時分割碎片：

- 單一碎片一致地達到 >80% 的最大 ACU
- 受單一碎片容量限制的查詢輸送量

碎片資料磁碟區

使用 SQL 函數查詢資料磁碟區：

```
SELECT subcluster_id, subcluster_type, pg_size_pretty(db_size)
FROM rds_aurora.limitless_stat_database_size('postgres_limitless')
```

```
ORDER BY 1;
```

若要檢視每個資料表和每個碎片資料：

```
SELECT * FROM rds_aurora.limitless_stat_relation_sizes('public', 'table_name');
```

解決不均勻的使用率

當工作負載或資料扭曲集中在特定碎片上時，分割碎片會將負載重新分配到更多執行個體。

重要考量：

- 無法控制要移動的碎片索引鍵
- 無法復原分割而不復原分割之前所拍攝的手動快照
- 所有執行個體，包括新的碎片，在閒置時都會使用執行個體最低 ACU

分割碎片允許進一步擴展，而連續碎片分割是通往更高輸送量和進一步擴展的路徑，同時保持低延遲。

限制

請注意這些操作限制：

路由器限制：

- 無法移除路由器 - 新增後，路由器會保留在叢集中
- 仔細規劃路由器新增，以避免不必要的基準成本

碎片限制：

- 碎片無法合併 - 碎片分割是單向操作
- 僅限復原選項：從分割前拍攝的快照還原

緩解策略：

- 從最低可行執行個體計數開始

- 視需要遞增新增容量
- 在主要拓撲變更之前拍攝快照
- 隨著叢集成長，監控基準成本

備份和還原 Aurora PostgreSQL Limitless Database

您可以備份和還原資料庫叢集，此叢集會使用 Aurora PostgreSQL Limitless Database。

內容

- 備份資料庫叢集，此叢集會使用 Aurora PostgreSQL Limitless Database
  - 建立資料庫叢集快照
- 還原資料庫叢集，此叢集會使用 Aurora PostgreSQL Limitless Database
  - 從資料庫快照還原資料庫叢集
  - 使用時間點復原還原資料庫叢集
- 不支援 PostgreSQL 備份和還原公用程式

備份資料庫叢集，此叢集會使用 Aurora PostgreSQL Limitless Database

與備份標準 Aurora 資料庫叢集相比，使用 Aurora PostgreSQL Limitless Database 備份資料庫叢集的功能有相似性和差異。

- 當您取得使用 Limitless Database 之 Aurora 資料庫叢集的手動資料庫叢集快照時，快照會包含來自資料庫碎片群組的資料。
- 連續備份包括來自資料庫碎片群組的資料。
- 自動化的每日快照包含來自資料庫碎片群組的資料。
- 支援複製資料庫叢集快照。如需更多詳細資訊，請參閱 資料庫叢集快照複製。
- 支援共用資料庫叢集快照。如需更多詳細資訊，請參閱 共享資料庫叢集快照。
- 您無法使用 `pg_dump` 或 `pg_dumpall` 公用程式來備份資料庫碎片群組中的資料庫。
- Aurora PostgreSQL Limitless Database 支援在刪除資料庫叢集時擷取最終快照。
- Aurora PostgreSQL Limitless Database 不支援在刪除資料庫叢集時保留自動備份。

建立資料庫叢集快照

如下列 AWS CLI 範例所示，您可以使用與標準 Aurora 資料庫叢集相同的方式建立 Aurora PostgreSQL Limitless Database 資料庫叢集快照：

```
aws rds create-db-cluster-snapshot \
    --db-cluster-identifier my-db-cluster \
    --db-cluster-snapshot-identifier my-db-cluster-snapshot
```

如需備份資料庫叢集的詳細資訊，請參閱備份與還原 Aurora 資料庫叢集的概觀。

還原資料庫叢集，此叢集會使用 Aurora PostgreSQL Limitless Database

與還原標準 Aurora 資料庫叢集相比，使用 Aurora PostgreSQL Limitless Database 還原資料庫叢集的功能有相似性和差異。

- 您只能從使用與 Limitless Database 相容的資料庫引擎版本的來源資料庫叢集 (例如 `16.4-limitless`)，還原 Limitless Database 資料庫叢集。
- 當您從使用 Limitless Database 的資料庫叢集手動快照還原資料庫叢集時，系統會還原整個資料庫叢集儲存體。這包括資料庫碎片群組的儲存體。

  您必須建立資料庫碎片群組，才能存取 Limitless Database 的儲存體。
- 您可以使用時間點復原 (PITR)，將資料庫叢集還原至保留期間內的任何時間點。還原的資料庫叢集包含資料庫碎片群組的儲存體。

  您必須建立資料庫碎片群組，才能存取 Limitless Database 的儲存體。
- 針對刪除的 Aurora PostgreSQL Limitless Database 資料庫叢集，不支援 PITR。
- 當您從自動每日快照還原資料庫叢集時，系統也會還原資料庫碎片群組的儲存體。
- 當您還原 Aurora PostgreSQL Limitless Database 資料庫叢集時，您必須啟用增強型監控和 Performance Insights。請務必包含 Performance Insights KMS 金鑰 ID。

還原 Aurora PostgreSQL Limitless Database 資料庫叢集之後，請務必在叢集執行查詢來驗證其功能。

從資料庫快照還原資料庫叢集

下列 AWS CLI 範例示範如何從資料庫叢集快照還原 Aurora PostgreSQL Limitless Database 資料庫叢集。

您必須使用 `16.4-limitless` 資料庫引擎版本。

若要從資料庫叢集快照還原 Limitless Database 資料庫叢集

1. 還原資料庫叢集：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier my-new-db-cluster \
    --snapshot-identifier my-db-cluster-snapshot \
    --engine aurora-postgresql \
    --engine-version 16.4-limitless \
    --enable-performance-insights \
    --performance-insights-retention-period 31 \
    --performance-insights-kms-key-id arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab \
    --monitoring-interval 5 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/EMrole
```

2. 建立資料庫碎片群組：

```
aws rds create-db-shard-group \
    --db-cluster-identifier my-new-db-cluster \
    --db-shard-group-identifier my-new-DB-shard-group \
    --max-acu 1000
```

   如需更多詳細資訊，請參閱 [將資料庫碎片群組新增至現有的 Aurora PostgreSQL Limitless Database 資料庫叢集]。

如需從資料庫叢集快照還原 Aurora 資料庫叢集的詳細資訊，請參閱[從資料庫叢集快照還原]。

使用時間點復原還原資料庫叢集

下列 AWS CLI 範例示範如何使用時間點復原 (PITR) 還原 Aurora PostgreSQL Limitless Database 資料庫叢集。

使用 PITR 還原 Limitless Database 資料庫叢集

1. 還原資料庫叢集：

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-db-cluster \
    --db-cluster-identifier my-new-db-cluster \
    --use-latest-restorable-time \
    --enable-performance-insights \
    --performance-insights-retention-period 31 \
    --performance-insights-kms-key-id arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab \
    --monitoring-interval 5 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/EMrole
```

2. 建立資料庫碎片群組：

```
aws rds create-db-shard-group \
    --db-cluster-identifier my-new-db-cluster \
    --db-shard-group-identifier my-new-DB-shard-group \
    --max-acu 1000
```

   如需更多詳細資訊，請參閱 將資料庫碎片群組新增至現有的 Aurora PostgreSQL Limitless Database 資料庫叢集。

如需 PITR 的詳細資訊，請參閱 將資料庫叢集還原至指定時間。

不支援 PostgreSQL 備份和還原公用程式

針對主要資料庫叢集或資料庫碎片群組，不支援下列 PostgreSQL 公用程式：

- `pg_dump`
- `pg_dumpall`
- `pg_restore`

雖然您可以藉由開放原始碼二進位檔或其他方法來加以使用，但這樣做可能會導致結果不一致。

升級 Amazon Aurora PostgreSQL Limitless Database

下列適用於升級 Aurora PostgreSQL Limitless Database：

- 支援次要版本升級。
- 支援修補 Aurora PostgreSQL Limitless Database。修補程式會顯示為待定維護，以便在維護時段期間套用。

升級使用 Amazon Aurora PostgreSQL Limitless Database 的資料庫叢集

您可以透過修改資料庫叢集以及選擇新的資料庫引擎版本，來升級該叢集。您可以使用 AWS 管理主控台 或 AWS CLI。

主控台

升級 Limitless Database 資料庫叢集

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 導覽至 Databases (資料庫) 頁面。
3. 選取您要升級的 Limitless Database 資料庫叢集。
4. 選擇 Modify (修改)。

   隨即顯示修改資料庫叢集頁面。
5. 對於資料庫引擎版本，請選擇新的資料庫引擎版本，例如具有 Limitless Database 的 Aurora PostgreSQL (與 PostgreSQL 16.6 相容)。
6. 選擇繼續。
7. 在摘要頁面，選取要立即套用變更或是要在下一個排定的維護時段期間套用變更，然後選擇修改叢集。

CLI

下列範例顯示的是使用 modify-db-cluster AWS CLI 命令。

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-sv2-cluster \
    --engine-version 16.6-limitless \
    --apply-immediately
```

Aurora PostgreSQL Limitless Database 參考

我們為 Aurora PostgreSQL Limitless Database 提供下列參考主題。

主題

- 支援和不支援的資料定義語言 (DDL) SQL 命令
- Aurora PostgreSQL Limitless Database 的 DDL 限制和其他資訊
- 支援和不支援的資料處理語言 (DML) 和查詢處理 SQL 命令
- Aurora PostgreSQL Limitless Database 的 DML 限制和其他資訊
- Aurora PostgreSQL Limitless Database 中的變數
- Aurora PostgreSQL Limitless Database 中的資料庫叢集參數

支援和不支援的資料定義語言 (DDL) SQL 命令

下表列出 Aurora PostgreSQL Limitless Database 支援和不支援的 DDL 命令，並參考限制或更多資訊。

Command	支援？	限制或更多資訊
ALTER AGGREGATE	否	不適用
ALTER COLLATION	是	無
ALTER CONVERSION	是	無
ALTER DATABASE	否	不適用
ALTER DEFAULT PRIVILEGES	否	不適用
ALTER DOMAIN	否	不適用
ALTER EVENT TRIGGER	否	不適用
ALTER EXTENSION	是	延伸模組
ALTER FOREIGN DATA WRAPPER	否	不適用

Command	支援？	限制或更多資訊
ALTER FOREIGN TABLE	否	不適用
ALTER FUNCTION	是	函數
ALTER GROUP	是	無
ALTER INDEX	是	無
ALTER LANGUAGE	否	不適用
ALTER LARGE OBJECT	否	不適用
ALTER MATERIALIZED VIEW	否	不適用
ALTER OPERATOR	是	無
ALTER OPERATOR CLASS	是	無
ALTER OPERATOR FAMILY	是	無
ALTER POLICY	否	不適用
ALTER PROCEDURE	是	無
ALTER PUBLICATION	否	不適用
ALTER ROLE	是	無
ALTER ROUTINE	否	不適用
ALTER RULE	否	不適用
ALTER SCHEMA	是	無
ALTER SEQUENCE	是	序列
ALTER SERVER	否	不適用
ALTER STATISTICS	否	不適用

Command	支援？	限制或更多資訊
ALTER SUBSCRIPTION	否	不適用
ALTER SYSTEM	否	不適用
ALTER TABLE	是	ALTER TABLE
ALTER TABLESPACE	否	不適用
ALTER TEXT SEARCH CONFIGURATION	否	不適用
ALTER TEXT SEARCH DICTIONARY	否	不適用
ALTER TEXT SEARCH PARSER	否	不適用
ALTER TEXT SEARCH TEMPLATE	否	不適用
ALTER TRIGGER	否	不適用
ALTER TYPE	是	無
ALTER USER	是	無
ALTER USER MAPPING	否	不適用
ALTER VIEW	是	無
COMMENT	否	不適用
CREATE ACCESS METHOD	否	不適用
CREATE AGGREGATE	否	不適用
CREATE CAST	是	無
CREATE COLLATION	是	無

Command	支援？	限制或更多資訊
CREATE CONVERSION	是	無
CREATE DATABASE	是	CREATE DATABASE
CREATE DOMAIN	否	不適用
CREATE EVENT TRIGGER	否	不適用
CREATE EXTENSION	是	延伸模組
CREATE FOREIGN DATA WRAPPER	否	不適用
CREATE FOREIGN TABLE	否	不適用
CREATE FUNCTION	是	函數
CREATE GROUP	是	無
CREATE INDEX	是	CREATE INDEX
CREATE LANGUAGE	否	不適用
CREATE MATERIALIZED VIEW	否	不適用
CREATE OPERATOR	是	無
CREATE OPERATOR CLASS	是	無
CREATE OPERATOR FAMILY	是	無
CREATE POLICY	是	無
CREATE PROCEDURE	是	無
CREATE PUBLICATION	否	不適用
CREATE ROLE	是	無

Command	支援？	限制或更多資訊
CREATE RULE	否	不適用
CREATE SCHEMA	是	CREATE SCHEMA
CREATE SEQUENCE	是	序列
CREATE SERVER	否	不適用
CREATE STATISTICS	否	不適用
CREATE SUBSCRIPTION	否	不適用
CREATE TABLE	是	CREATE TABLE
CREATE TABLE AS	是	CREATE TABLE AS
CREATE TABLESPACE	否	不適用
CREATE TEMPORARY TABLE	否	不適用
CREATE TEMPORARY TABLE AS	否	不適用
CREATE TEXT SEARCH CONFIGURATION	否	不適用
CREATE TEXT SEARCH DICTIONARY	否	不適用
CREATE TEXT SEARCH PARSER	否	不適用
CREATE TEXT SEARCH TEMPLATE	否	不適用
CREATE TRANSFORM	否	不適用
CREATE TRIGGER	否	不適用

Command	支援？	限制或更多資訊
CREATE TYPE	是	無
CREATE USER	是	無
CREATE USER MAPPING	否	不適用
CREATE VIEW	是	無
DROP ACCESS METHOD	否	不適用
DROP AGGREGATE	是	無
DROP CAST	是	無
DROP COLLATION	是	無
DROP CONVERSION	是	無
DROP DATABASE	是	DROP DATABASE
DROP DOMAIN	否	不適用
DROP EVENT TRIGGER	否	不適用
DROP EXTENSION	是	延伸模組
DROP FOREIGN DATA WRAPPER	否	不適用
DROP FOREIGN TABLE	否	不適用
DROP FUNCTION	是	函數
DROP GROUP	是	無
DROP INDEX	是	無
DROP LANGUAGE	否	不適用
DROP MATERIALIZED VIEW	否	不適用

Command	支援？	限制或更多資訊
DROP OPERATOR	是	無
DROP OPERATOR CLASS	是	無
DROP OPERATOR FAMILY	是	無
DROP OWNED	否	不適用
DROP POLICY	否	不適用
DROP PROCEDURE	是	無
DROP PUBLICATION	否	不適用
DROP ROLE	是	無
DROP ROUTINE	否	不適用
DROP ROLE	否	不適用
DROP SCHEMA	是	無
DROP SEQUENCE	是	無
DROP SERVER	否	不適用
DROP STATISTICS	否	不適用
DROP SUBSCRIPTION	否	無
DROP TABLE	是	無
DROP TABLESPACE	否	不適用
DROP TEXT SEARCH CONFIGURATION	否	不適用
DROP TEXT SEARCH DICTIONARY	否	不適用

Command	支援？	限制或更多資訊
DROP TEXT SEARCH PARSER	否	不適用
DROP TEXT SEARCH TEMPLATE	否	不適用
DROP TRANSFORM	否	不適用
DROP TRIGGER	否	不適用
DROP TYPE	是	無
DROP USER	是	無
DROP USER MAPPING	否	不適用
DROP VIEW	是	無
GRANT	是	無
REASSIGN OWNED	否	不適用
REVOKE	是	無
SECURITY LABEL	否	不適用
SELECT INTO	是	SELECT INTO
SET	是	無
SET CONSTRAINTS	否	不適用
SET ROLE	是	無
SET SESSION AUTHORIZATION	是	無
SET TRANSACTION	是	無
TRUNCATE	是	無

Aurora PostgreSQL Limitless Database 的 DDL 限制和其他資訊

下列主題說明限制，或提供 Aurora PostgreSQL Limitless Database 中 DDL SQL 命令的詳細資訊。

主題

- ALTER TABLE
- CREATE DATABASE
- CREATE INDEX
- CREATE SCHEMA
- CREATE TABLE
- CREATE TABLE AS
- DROP DATABASE
- SELECT INTO
- 限制
- 預設值
- 延伸模組
- 外部索引鍵
- 函數
- 序列

ALTER TABLE

Aurora PostgreSQL Limitless Database 中通常支援 ALTER TABLE 命令。如需詳細資訊，請參閱 PostgreSQL 文件中的 ALTER TABLE。

限制

ALTER TABLE 對於支援的選項有下列限制。

移除欄

- 在碎片資料表，您無法移除屬於碎片索引鍵一部分的欄。
- 在參考資料表，您無法移除主索引鍵欄。

變更資料欄的資料類型

- 不支援 USING 運算式。
- 在碎片資料表，您無法變更屬於碎片索引鍵的欄類型。

新增或移除限制條件

如需不支援之項目的詳細資訊，請參閱限制。

變更欄的預設值

支援預設值。如需更多詳細資訊，請參閱 預設值。

不支援的選項

不支援某些選項，因為其與不支援的功能相依，例如觸發。

不支援 ALTER TABLE 的下列資料表層級選項：

- ALL IN TABLESPACE
- ATTACH PARTITION
- DETACH PARTITION
- ONLY 標記
- RENAME CONSTRAINT

不支援 ALTER TABLE 的下列欄層級選項：

- ADD GENERATED

- DROP EXPRESSION [IF EXISTS]
- DROP IDENTITY [IF EXISTS]
- RESET
- RESTART
- SET
- SET COMPRESSION
- SET STATISTICS

CREATE DATABASE

在 Aurora PostgreSQL Limitless Database 中，僅支援無限資料庫。

執行 CREATE DATABASE 時，在一或多個節點中成功建立的資料庫可能會在其他節點中失敗，因為資料庫建立是非交易操作。在這種情況下，成功建立的資料庫物件會在預定時間內自動從所有節點中移除，以保持資料庫碎片群組中的一致性。在此期間，以相同名稱重新建立資料庫可能會導致錯誤，指出資料庫已存在。

支援下列選項：

- 定序：

```
CREATE DATABASE name WITH
    [LOCALE = locale]
    [LC_COLLATE = lc_collate]
    [LC_CTYPE = lc_ctype]
    [ICU_LOCALE = icu_locale]
    [ICU_RULES = icu_rules]
    [LOCALE_PROVIDER = locale_provider]
    [COLLATION_VERSION = collation_version];
```

- CREATE DATABASE WITH OWNER:

```
CREATE DATABASE name WITH OWNER = user_name;
```

不支援下列選項：

- CREATE DATABASE WITH TABLESPACE:

```
CREATE DATABASE name WITH TABLESPACE = tablespace_name;
```

- CREATE DATABASE WITH TEMPLATE:

```
CREATE DATABASE name WITH TEMPLATE = template;
```

CREATE INDEX

針對碎片資料表，支援 CREATE INDEX CONCURRENTLY：

```
CREATE INDEX CONCURRENTLY index_name ON table_name(column_name);
```

針對所有資料表類型，支援 CREATE UNIQUE INDEX：

```
CREATE UNIQUE INDEX index_name ON table_name(column_name);
```

不支援 CREATE UNIQUE INDEX CONCURRENTLY：

```
CREATE UNIQUE INDEX CONCURRENTLY index_name ON table_name(column_name);
```

如需詳細資訊，請參閱 UNIQUE。如需建立索引的一般資訊，請參閱 PostgreSQL 文件中的 CREATE INDEX。

顯示索引

當您使用 \d *table_name* 或類似命令時，並非所有索引都會在路由器上可見。反之，請如下列範例所示，使用 pg_catalog.pg_indexes 檢視來取得索引。

```
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"id"}';
CREATE TABLE items (id int PRIMARY KEY, val int);
CREATE INDEX items_my_index on items (id, val);

postgres_limitless=> SELECT * FROM pg_catalog.pg_indexes WHERE tablename='items';

 schemaname | tablename |   indexname    | tablespace |
   indexdef
------------+-----------+----------------+------------
+-------------------------------------------------------------------------
```

```
 public     | items     | items_my_index |            | CREATE INDEX items_my_index
 ON ONLY public.items USING btree (id, val)
 public     | items     | items_pkey     |            | CREATE UNIQUE INDEX
 items_pkey ON ONLY public.items USING btree (id)
(2 rows)
```

CREATE SCHEMA

不支援具有結構描述元素的 CREATE SCHEMA：

```
CREATE SCHEMA my_schema CREATE TABLE (column_name INT);
```

此會產生如下的錯誤：

```
ERROR: CREATE SCHEMA with schema elements is not supported
```

CREATE TABLE

不支援 CREATE TABLE 陳述式中的關係，例如：

```
CREATE TABLE orders (orderid int, customerId int, orderDate date) WITH
 (autovacuum_enabled = false);
```

不支援 IDENTITY 欄，例如：

```
CREATE TABLE orders (orderid INT GENERATED ALWAYS AS IDENTITY);
```

Aurora PostgreSQL Limitless Database 支援最多 54 個字元的碎片資料表名稱。

CREATE TABLE AS

若要使用 CREATE TABLE AS 建立資料表，您必須使用 rds_aurora.limitless_create_table_mode 變數。對於碎片資料表，您還必須使用 rds_aurora.limitless_create_table_shard_key 變數。如需更多詳細資訊，請參閱 使用變數建立無限資料表。

```
-- Set the variables.
SET rds_aurora.limitless_create_table_mode='sharded';
```

```
SET rds_aurora.limitless_create_table_shard_key='{"a"}';

CREATE TABLE ctas_table AS SELECT 1 a;

-- "source" is the source table whose columns and data types are used to create the new
 "ctas_table2" table.
CREATE TABLE ctas_table2 AS SELECT a,b FROM source;
```

您無法使用 CREATE TABLE AS 建立參考資料表，因為其需要主索引鍵限制條件。CREATE TABLE AS 不會將主索引鍵傳播到新資料表。

如需一般資訊，請參閱 PostgreSQL 文件中的 CREATE TABLE AS。

DROP DATABASE

您可以捨棄已建立的資料庫。

DROP DATABASE 命令會在背景中以非同步方式執行。執行時，如果您嘗試建立同名的新資料庫，則會收到錯誤。

SELECT INTO

SELECT INTO 的功能類似於 CREATE TABLE AS。您必須使用 rds_aurora.limitless_create_table_mode 變數。對於碎片資料表，您還必須使用 rds_aurora.limitless_create_table_shard_key 變數。如需更多詳細資訊，請參閱 使用變數建立無限資料表。

```
-- Set the variables.
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"a"}';

-- "source" is the source table whose columns and data types are used to create the new
 "destination" table.
SELECT * INTO destination FROM source;
```

目前，SELECT INTO 操作是透過路由器 (而不是直接透過碎片) 執行。因此，效能可能會很慢。

如需一般資訊，請參閱 PostgreSQL 文件中的 SELECT INTO。

限制

下列限制適用於 Aurora PostgreSQL Limitless Database 中的限制條件。

CHECK

支援涉及比較運算子與常值的簡單限制條件。如下列範例所示，不支援需要函數驗證的更複雜運算式和限制條件。

```
CREATE TABLE my_table (
    id  INT CHECK (id > 0)                                          -- supported
  , val INT CHECK (val > 0 AND val < 1000)                          -- supported
  , tag TEXT CHECK (length(tag) > 0)                                -- not supported:
 throws "Expression inside CHECK constraint is not supported"
  , op_date TIMESTAMP WITH TIME ZONE CHECK (op_date <= now())  -- not supported:
 throws "Expression inside CHECK constraint is not supported"
);
```

您可以如下列範例所示，提供限制條件明確名稱。

```
CREATE TABLE my_table (
    id  INT CONSTRAINT positive_id  CHECK (id > 0)
  , val INT CONSTRAINT val_in_range CHECK (val > 0 AND val < 1000)
);
```

您可以如下列範例所示，將資料表層級限制條件語法與 CHECK 限制條件搭配使用。

```
CREATE TABLE my_table (
    id INT CONSTRAINT positive_id  CHECK (id > 0)
  , min_val INT CONSTRAINT min_val_in_range CHECK (min_val > 0 AND min_val < 1000)
  , max_val INT
  , CONSTRAINT max_val_in_range CHECK (max_val > 0 AND max_val < 1000 AND max_val >
 min_val)
);
```

EXCLUDE

Aurora PostgreSQL Limitless Database 中不支援排除限制條件。

FOREIGN KEY

如需更多詳細資訊，請參閱 外部索引鍵。

NOT NULL

支援 NOT NULL 限制條件，沒有限制。

PRIMARY KEY

主索引鍵表示唯一限制條件，因此對唯一限制條件的相同限制適用於主索引鍵。這表示：

- 如果將資料表轉換為碎片資料表，碎片索引鍵必須是主索引鍵的子集。也就是說，主索引鍵包含碎片索引鍵的所有欄。
- 如果將資料表轉換為參考資料表，則其必須有主索引鍵。

下列範例說明主索引鍵的使用方式。

```
-- Create a standard table.
CREATE TABLE public.my_table (
    item_id INT
  , location_code INT
  , val INT
  , comment text
);

-- Change the table to a sharded table using the 'item_id' and 'location_code'
 columns as shard keys.
CALL rds_aurora.limitless_alter_table_type_sharded('public.my_table',
 ARRAY['item_id', 'location_code']);
```

嘗試新增不包含碎片索引鍵的主索引鍵：

```
-- Add column 'item_id' as the primary key.
-- Invalid because the primary key doesnt include all columns from the shard key:
-- 'location_code' is part of the shard key but not part of the primary key
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id); -- ERROR

-- add column "val" as primary key
-- Invalid because primary key does not include all columns from shard key:
--  item_id and location_code iare part of shard key but not part of the primary key
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id); -- ERROR
```

嘗試新增包含碎片索引鍵的主索引鍵：

```
-- Add the 'item_id' and 'location_code' columns as the primary key.
-- Valid because the primary key contains the shard key.
```

```
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id, location_code); -- OK

-- Add the 'item_id', 'location_code', and 'val' columns as the primary key.
-- Valid because the primary key contains the shard key.
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id, location_code, val); -- OK
```

將標準資料表變更為參考資料表。

```
-- Create a standard table.
CREATE TABLE zipcodes (zipcode INT PRIMARY KEY, details VARCHAR);

-- Convert the table to a reference table.
CALL rds_aurora.limitless_alter_table_type_reference('public.zipcode');
```

如需建立碎片和參考資料表的詳細資訊，請參閱建立 Aurora PostgreSQL Limitless Database 資料表。

UNIQUE

在碎片資料表中，唯一索引鍵必須包含碎片索引鍵，也就是說，碎片索引鍵必須是唯一索引鍵的子集。將資料表類型變更為碎片時會檢查此項目。在參考資料表中沒有限制。

```
CREATE TABLE customer (
    customer_id INT NOT NULL
  , zipcode INT
  , email TEXT UNIQUE
);
```

如下列範例所示，支援資料表層級 UNIQUE 限制條件。

```
CREATE TABLE customer (
    customer_id INT NOT NULL
  , zipcode INT
  , email TEXT
  , CONSTRAINT zipcode_and_email UNIQUE (zipcode, email)
);
```

下列範例示範同時使用主索引鍵和唯一索引鍵。這兩個索引鍵都必須包含碎片索引鍵。

```
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"p_id"}';
```

```
CREATE TABLE t1 (
p_id BIGINT NOT NULL,
c_id BIGINT NOT NULL,
PRIMARY KEY (p_id),
UNIQUE (p_id, c_id)
);
```

如需詳細資訊，請參閱 PostgreSQL 文件中的限制條件。

預設值

Aurora PostgreSQL Limitless Database 支援預設值中的運算式。

下列範例示範預設值的使用方式。

```
CREATE TABLE t (
    a INT DEFAULT 5,
    b TEXT DEFAULT 'NAN',
    c NUMERIC
);

CALL rds_aurora.limitless_alter_table_type_sharded('t', ARRAY['a']);
INSERT INTO t DEFAULT VALUES;
SELECT * FROM t;

 a |  b  | c
---+-----+---
 5 | NAN |
(1 row)
```

如下列範例所示，支援運算式。

```
CREATE TABLE t1 (a NUMERIC DEFAULT random());
```

下列範例會新增新欄，該欄為 NOT NULL 且具有預設值。

```
ALTER TABLE t ADD COLUMN d BOOLEAN NOT NULL DEFAULT FALSE;
SELECT * FROM t;

 a |  b  | c | d
---+-----+---+---
```

```
 5 | NAN |   | f
(1 row)
```

下列範例會變更具有預設值的現有欄。

```
ALTER TABLE t ALTER COLUMN c SET DEFAULT 0.0;
INSERT INTO t DEFAULT VALUES;
SELECT * FROM t;

 a |  b  | c   |  d
---+-----+-----+-----
 5 | NAN |     | f
 5 | NAN | 0.0 | f
(2 rows)
```

下列範例會捨棄預設值。

```
ALTER TABLE t ALTER COLUMN a DROP DEFAULT;
INSERT INTO t DEFAULT VALUES;
SELECT * FROM t;

 a |  b  | c   |  d
---+-----+-----+-----
 5 | NAN |     | f
 5 | NAN | 0.0 | f
   | NAN | 0.0 | f
(3 rows)
```

如需的詳細資訊，請參閱 PostgreSQL 文件中的預設值。

延伸模組

Aurora PostgreSQL Limitless Database 支援下列 PostgreSQL 延伸模組：

- `aurora_limitless_fdw`：已預先安裝此延伸模組。您無法將其捨棄。
- `aws_s3`：此延伸模組可在 Aurora PostgreSQL Limitless Database 中運作，運作方式與在 Aurora PostgreSQL 中類似。

  您可以將資料從 Amazon S3 儲存貯體匯入 Aurora PostgreSQL 資料庫叢集，或將資料從 Aurora PostgreSQL Limitless Database 資料庫叢集匯出至 Amazon S3 儲存貯體。如需詳細資訊，請參閱將資料從 Amazon S3 匯入 Aurora PostgreSQL 資料庫叢集 及將資料從 Aurora PostgreSQL 資料庫叢集匯出至 Amazon S3。
- `btree_gin`
- `citext`
- `ip4r`
- `pg_buffercache`：此延伸模組在 Aurora PostgreSQL Limitless Database 中的行為與在社群 PostgreSQL 不同。如需更多詳細資訊，請參閱 Aurora PostgreSQL Limitless Database 中的 pg_buffercache 差異。
- `pg_stat_statements`
- `pg_trgm`
- `pgcrypto`
- `pgstattuple`：此延伸模組在 Aurora PostgreSQL Limitless Database 中的行為與在社群 PostgreSQL 不同。如需更多詳細資訊，請參閱 Aurora PostgreSQL Limitless Database 中的 pgstattuple 差異。
- `pgvector`
- `plpgsql`：已預先安裝此延伸模組，但您可以將其捨棄。
- `PostGIS`：不支援長交易和資料表管理函數。不支援對空間參考資料表進行修改。
- `unaccent`
- `uuid`

Aurora PostgreSQL Limitless Database 目前不支援大多數 PostgreSQL 延伸模組。不過，您仍然可以使用 shared_preload_libraries (SPL) 組態設定，將延伸項目載入 Aurora PostgreSQL 主要資料庫叢集。其也會載入 Aurora PostgreSQL Limitless Database，但可能無法正常運作。

例如，您可以載入 `pg_hint_plan` 延伸模組，但載入延伸模組並不保證使用查詢註解中傳遞的提示。

> **Note**
>
> 您無法修改與 pg_stat_statements 延伸模組相關聯的物件。如需安裝 pg_stat_statements 的資訊，請參閱 limitless_stat_statements。

您可以使用 pg_available_extensions 和 pg_available_extension_versions 函數，來尋找 Aurora PostgreSQL Limitless Database 中支援的延伸模組。

針對以下延伸模組，支援以下 DDL：

CREATE EXTENSION

您可以如 PostgreSQL 中所示建立延伸模組。

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
    [ WITH ] [ SCHEMA schema_name ]
             [ VERSION version ]
             [ CASCADE ]
```

如需詳細資訊，請參閱 PostgreSQL 文件中的 CREATE EXTENSION。

ALTER EXTENSION

支援的 DDL 如下：

```
ALTER EXTENSION name UPDATE [ TO new_version ]

ALTER EXTENSION name SET SCHEMA new_schema
```

如需詳細資訊，請參閱 PostgreSQL 文件中的 ALTER EXTENSION。

DROP EXTENSION

您可以如 PostgreSQL 中所示捨棄延伸模組。

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

如需詳細資訊，請參閱 PostgreSQL 文件中的 DROP EXTENSION。

針對以下延伸模組，不支援以下 DDL：

ALTER EXTENSION

您無法從延伸模組新增或捨棄成員物件。

```
ALTER EXTENSION name ADD member_object

ALTER EXTENSION name DROP member_object
```

Aurora PostgreSQL Limitless Database 中的 pg_buffercache 差異

在 Aurora PostgreSQL Limitless Database 中，當您安裝 pg_buffercache 延伸模組並使用 pg_buffercache 檢視時，您只會從目前連線的節點接收緩衝區相關資訊：路由器。同樣地，使用函數 pg_buffercache_summary 或 pg_buffercache_usage_counts 僅提供來自連線節點的資訊。

您可以擁有許多節點，而且可能需要從任何節點存取緩衝區資訊，才能有效地診斷問題。因此，Limitless Database 提供下列函數：

- rds_aurora.limitless_pg_buffercache(*subcluster_id*)
- rds_aurora.limitless_pg_buffercache_summary(*subcluster_id*)
- rds_aurora.limitless_pg_buffercache_usage_counts(*subcluster_id*)

透過輸入任何節點的子叢集 ID (無論是路由器還是碎片)，您都可以輕鬆存取該節點特定的緩衝區資訊。當您在無限資料庫中安裝 pg_buffercache 延伸模組時，即可直接使用這些函數。

> **Note**
>
> Aurora PostgreSQL Limitless Database 支援 1.4 版及更新版本的 pg_buffercache 延伸模組的這些函數。

limitless_pg_buffercache 檢視中顯示的欄與 pg_buffercache 檢視中的欄略有不同：

- bufferid：與 pg_buffercache 保持不變。
- relname：與 pg_buffercache 中顯示的檔案節點編號不同，limitless_pg_buffercache 會呈現目前資料庫或共用系統目錄中可用的相關 relname，否則會顯示 NULL。
- parent_relname：此新欄在 pg_buffercache 中不存在，如果 relname 欄中的值代表分割的資料表 (如果是碎片資料表)，則會顯示父項 relname。否則，會顯示 NULL。

- spcname：limitless_pg_buffercache 會顯示資料表空間名稱，而不是在 pg_buffercache 中顯示資料表空間物件識別符 (OID)。
- datname：limitless_pg_buffercache 會顯示資料庫名稱，而不是在 pg_buffercache 中顯示資料庫 OID。
- relforknumber：與 pg_buffercache 保持不變。
- relblocknumber：與 pg_buffercache 保持不變。
- isdirty：與 pg_buffercache 保持不變。
- usagecount：與 pg_buffercache 保持不變。
- pinning_backends：與 pg_buffercache 保持不變。

limitless_pg_buffercache_summary 和 limitless_pg_buffercache_usage_counts 檢視中的欄分別與一般 pg_buffercache_summary 和 pg_buffercache_usage_counts 檢視中的欄相同。

透過使用這些函數，您可以存取 Limitless Database 環境中所有節點的詳細緩衝區快取資訊，從而更有效地診斷和管理資料庫系統。

Aurora PostgreSQL Limitless Database 中的 pgstattuple 差異

在 Aurora PostgreSQL 中，pgstattuple 延伸模組目前不支援外部資料表、分割資料表或分割索引。不過，在 Aurora PostgreSQL Limitless Database 中，使用者建立的物件通常是這些不支援的類型之一。雖然有一般資料表和索引 (例如，目錄資料表及其索引)，但大多數物件都位於外部節點，使其成為路由器的外部物件。

我們了解此延伸模組對於取得元組層級統計資料的重要性，這對於排除膨脹和收集診斷資訊等任務至關重要。因此，Aurora PostgreSQL Limitless Database 支援無限資料庫中的 pgstattuple 延伸模組。

Aurora PostgreSQL Limitless Database 在 rds_aurora 結構描述中包含下列函數：

元組層級統計資料函數

rds_aurora.limitless_pgstattuple(*relation_name*)

- 目的：擷取標準資料表及其索引的元組層級統計資料
- 輸入：relation_name (文字)：關係的名稱
- 輸出：欄與 Aurora PostgreSQL 中的 pgstattuple 函數傳回的欄一致

rds_aurora.limitless_pgstattuple(*relation_name*, *subcluster_id*)

- 目的：擷取參考資料表、碎片資料表、目錄資料表及其索引的元組層級統計資料
- 輸入:
  - relation_name (文字)：關係的名稱
  - subcluster_id (文字)：要擷取統計資料之節點的子叢集 ID
- 輸出：
  - 對於參考和目錄資料表 (包括其索引)，欄與 Aurora PostgreSQL 中的欄一致。
  - 對於碎片資料表，統計資料僅代表位於指定子叢集上碎片資料表的分割區。

索引統計資料函數

rds_aurora.limitless_pgstatindex(*relation_name*)

- 目的：擷取標準資料表上 B 型樹狀索引的統計資料
- 輸入：relation_name (文字)：B 型樹狀索引的名稱
- 輸出：傳回除 root_block_no 以外的所有欄。傳回的欄與 Aurora PostgreSQL 中的 pgstatindex 函數一致。

rds_aurora.limitless_pgstatindex(*relation_name*, *subcluster_id*)

- 目的：擷取參考資料表、碎片資料表和目錄資料表上 B 型樹狀索引的統計資料。
- 輸入:
  - relation_name (文字)：B 型樹狀索引的名稱
  - subcluster_id (文字)：要擷取統計資料之節點的子叢集 ID
- 輸出：
  - 對於參考和目錄資料表索引，會傳回所有欄 (root_block_no 除外)。傳回的欄與 Aurora PostgreSQL 一致。
  - 對於碎片資料表，統計資料僅代表位於指定子叢集上碎片資料表索引的分割區。tree_level 欄會顯示請求子叢集上所有資料表配量中的平均值。

rds_aurora.limitless_pgstatginindex(*relation_name*)

- 目的：擷取標準資料表一般化反轉索引 (GIN) 統計資料
- 輸入：relation_name (文字)：GIN 的名稱
- 輸出：欄與 Aurora PostgreSQL 中的 pgstatginindex 函數傳回的欄一致

rds_aurora.limitless_pgstatginindex(*relation_name*, *subcluster_id*)

- 目的：擷取參考資料表、碎片資料表和目錄資料表上 GIN 索引的統計資料。

- 輸入:
  - relation_name (文字)：索引的名稱
  - subcluster_id (文字)：要擷取統計資料之節點的子叢集 ID
- 輸出：
  - 對於參考和目錄資料表 GIN 索引，欄與 Aurora PostgreSQL 中的欄一致。
  - 對於碎片資料表，統計資料僅代表位於指定子叢集上碎片資料表索引的分割區。

rds_aurora.limitless_pgstathashindex(*relation_name*)

- 目的：擷取標準資料表上雜湊索引的統計資料
- 輸入：relation_name (文字)：雜湊索引的名稱
- 輸出：欄與 Aurora PostgreSQL 中的 pgstathashindex 函數傳回的欄一致

rds_aurora.limitless_pgstathashindex(*relation_name*, *subcluster_id*)

- 目的：擷取參考資料表、碎片資料表和目錄資料表上雜湊索引的統計資料。
- 輸入:
  - relation_name (文字)：索引的名稱
  - subcluster_id (文字)：要擷取統計資料之節點的子叢集 ID
- 輸出：
  - 對於參考和目錄資料表雜湊索引，欄與 Aurora PostgreSQL 一致。
  - 對於碎片資料表，統計資料僅代表位於指定子叢集上碎片資料表索引的分割區。

頁面計數函數

rds_aurora.limitless_pg_relpages(*relation_name*)

- 目的：擷取標準資料表及其索引的頁面計數
- 輸入：relation_name (文字)：關係的名稱
- 輸出：指定關係的頁面計數

rds_aurora.limitless_pg_relpages(*relation_name*, *subcluster_id*)

- 目的：擷取參考資料表、碎片資料表和目錄資料表 (包括其索引) 的頁面計數
- 輸入:
  - relation_name (文字)：關係的名稱
  - subcluster_id (文字)：要擷取頁面計數之節點的子叢集 ID
- 輸出：對於碎片資料表，頁面計數是指定子叢集上所有資料表配量中的頁面總和。

約略元組層級統計資料函數

`rds_aurora.limitless_pgstattuple_approx(relation_name)`

- 目的：擷取標準資料表及其索引的約略元組層級統計資料
- 輸入：`relation_name` (文字)：關係的名稱
- 輸出：欄與 Aurora PostgreSQL 中 pgstattuple_approx 函數傳回的欄一致

`rds_aurora.limitless_pgstattuple_approx(relation_name, subcluster_id)`

- 目的：擷取參考資料表、碎片資料表和目錄資料表 (包括其索引) 的約略元組層級統計資料
- 輸入:
  - `relation_name` (文字)：關係的名稱
  - `subcluster_id` (文字)：要擷取統計資料之節點的子叢集 ID
- 輸出：
  - 對於參考和目錄資料表 (包括其索引)，欄與 Aurora PostgreSQL 中的欄一致。
  - 對於碎片資料表，統計資料僅代表位於指定子叢集上碎片資料表的分割區。

> **Note**
>
> 目前，Aurora PostgreSQL Limitless Database 不支援具體化視觀表、TOAST 資料表或暫存資料表上的 `pgstattuple` 延伸模組。
> 在 Aurora PostgreSQL Limitless Database 中，您必須以文字形式提供輸入，但 Aurora PostgreSQL 支援其他格式。

外部索引鍵

支援外部索引鍵 (FOREIGN KEY) 限制條件，但有一些限制：

- 僅針對標準資料表，支援 CREATE TABLE 搭配 FOREIGN KEY。若要使用 FOREIGN KEY 建立碎片或參考資料表，請先建立沒有外部索引鍵限制條件的資料表。然後使用下列陳述式進行修改：

```
ALTER TABLE ADD CONSTRAINT;
```

- 當資料表具有外部索引鍵限制條件時，不支援將標準資料表轉換為碎片或參考資料表。捨棄限制條件，然後在轉換後加以新增。
- 下列限制適用於外部索引鍵限制條件的資料表類型：
  - 標準資料表可以對另一個標準資料表有外部索引鍵限制條件。
  - 如果父資料表和子資料表共置，且外部索引鍵是碎片索引鍵的超集，則碎片資料表可能會有外部索引鍵限制條件。
  - 碎片資料表對參考資料表可能有外部索引鍵限制條件。
  - 參考資料表對另一個參考資料表可能有外部索引鍵限制條件。

主題

- 外部索引鍵選項
- 範例

外部索引鍵選項

對於某些 DDL 選項，Aurora PostgreSQL Limitless Database 中支援外部索引鍵。下表列出 Aurora PostgreSQL Limitless Database 資料表之間支援和不支援的選項。

DDL 選項	參考到參考	碎片到碎片 (共置)	碎片到參考	標準到標準
DEFERRABLE	是	是	是	是
INITIALLY DEFERRED	是	是	是	是

DDL 選項	參考到參考	碎片到碎片 (共置)	碎片到參考	標準到標準
INITIALLY IMMEDIATE	是	是	是	是
MATCH FULL	是	是	是	是
MATCH PARTIAL	否	否	否	否
MATCH SIMPLE	是	是	是	是
NOT DEFERRABLE	是	是	是	是
NOT VALID	是	否	否	是
ON DELETE CASCADE	是	是	是	是
ON DELETE NO ACTION	是	是	是	是
ON DELETE RESTRICT	是	是	是	是
ON DELETE SET DEFAULT	否	否	否	否
ON DELETE SET NULL	是	否	否	是
ON UPDATE CASCADE	否	否	否	是
ON UPDATE NO ACTION	是	是	是	是

DDL 選項	參考到參考	碎片到碎片 (共置)	碎片到參考	標準到標準
ON UPDATE RESTRICT	是	是	是	是
ON UPDATE SET DEFAULT	否	否	否	否
ON UPDATE SET NULL	是	否	否	是

範例

- 標準到標準：

```
set rds_aurora.limitless_create_table_mode='standard';

CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);

SELECT constraint_name, table_name, constraint_type
FROM information_schema.table_constraints WHERE constraint_type='FOREIGN KEY';

 constraint_name         | table_name  | constraint_type
-------------------------+-------------+-----------------
 orders_product_no_fkey  | orders      | FOREIGN KEY
(1 row)
```

- 碎片到碎片 (共置)：

```
set rds_aurora.limitless_create_table_mode='sharded';
```

```
set rds_aurora.limitless_create_table_shard_key='{"product_no"}';
CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

set rds_aurora.limitless_create_table_shard_key='{"order_id"}';
set rds_aurora.limitless_create_table_collocate_with='products';
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    quantity integer
);

ALTER TABLE orders ADD CONSTRAINT order_product_fk FOREIGN KEY (product_no)
 REFERENCES products (product_no);
```

- **碎片到參考**：

```
set rds_aurora.limitless_create_table_mode='reference';
CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

set rds_aurora.limitless_create_table_mode='sharded';
set rds_aurora.limitless_create_table_shard_key='{"order_id"}';
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    quantity integer
);

ALTER TABLE orders ADD CONSTRAINT order_product_fk FOREIGN KEY (product_no)
 REFERENCES products (product_no);
```

- **參考到參考**：

```
set rds_aurora.limitless_create_table_mode='reference';
CREATE TABLE products(
    product_no integer PRIMARY KEY,
```

```
    name text,
    price numeric
);
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    quantity integer
);

ALTER TABLE orders ADD CONSTRAINT order_product_fk FOREIGN KEY (product_no)
 REFERENCES products (product_no);
```

函數

Aurora PostgreSQL Limitless Database 中支援函數。

針對函數，以下是支援的 DDL：

CREATE FUNCTION

您可以像在 Aurora PostgreSQL 中一樣建立函數，在取代時變更其波動除外。

如需詳細資訊，請參閱 PostgreSQL 文件中的 CREATE FUNCTION。

ALTER FUNCTION

您可以像 Aurora PostgreSQL 一樣變更函數，但變更其波動除外。

如需詳細資訊，請參閱 PostgreSQL 文件中的 ALTER FUNCTION。

DROP FUNCTION

您可以如 Aurora PostgreSQL 中所示捨棄函數。

```
DROP FUNCTION [ IF EXISTS ] name [ ( [ [ argmode ] [ argname ] argtype [, ...] ] ) ]
 [, ...]
    [ CASCADE | RESTRICT ]
```

如需詳細資訊，請參閱 PostgreSQL 文件中的 DROP FUNCTION。

主題

- 函數分佈
- 函數波動

函數分佈

當函數的所有陳述式都以單一碎片為目標時，將整個函數向下推送到目標碎片是有益的。然後，結果會傳播回路由器 (而不是在路由器本身顯示函數)。函數和預存程序下推功能對於想要更接近資料來源 (也就是碎片) 的位置執行函數或預存程序的客戶而言相當實用。

若要分佈函數，請先建立函數，然後呼叫 `rds_aurora.limitless_distribute_function` 程序來分佈函數。此函數使用以下語法：

```
SELECT rds_aurora.limitless_distribute_function('function_prototype',
 ARRAY['shard_key'], 'collocating_table');
```

此函數採用下列參數：

- *function_prototype*：要分佈的函數。僅提及輸入引數，而非任何輸出引數。

  如果將任何引數定義為 OUT 參數，請勿在 function_prototype 的引數中包含其類型。
- ARRAY['*shard_key*']：識別為函數碎片索引鍵的函數引數清單。
- *collocating_table*：包含目標碎片上資料範圍的碎片資料表。

若要識別要下推此函數以執行的碎片，系統會採用 ARRAY['*shard_key*'] 引數、對其進行雜湊，並從 *collocating_table* 中找到主控內含此雜湊值之範圍的碎片。

限制

當您分佈函數或程序時，其只會處理該碎片中碎片索引鍵範圍所限制的資料。如果函數或程序嘗試從不同的碎片存取資料，則分散式函數或程序傳回的結果將與非分散式的函數或程序的結果不同。

例如，您建立一個函數，其中包含會觸動多個碎片的查詢，然後呼叫 rds_aurora.limitless_distribute_function 程序來加以分佈。當您提供碎片索引鍵的引數來調用此函數時，執行該函數的結果可能會受到該碎片中存在的值所限制。這些結果與在不分佈函數的情況下產生的結果不同。

範例

請考慮以下函數 func，其中我們有具有碎片索引鍵 customer_id 的碎片資料表 customers。

```
postgres_limitless=> CREATE OR REPLACE FUNCTION func(c_id integer, sc integer)
 RETURNS int
    language SQL
    volatile
    AS $$
    UPDATE customers SET score = sc WHERE customer_id = c_id RETURNING score;
    $$;
```

現在我們分佈此函數：

```
SELECT rds_aurora.limitless_distribute_function('func(integer, integer)',
 ARRAY['c_id'], 'customers');
```

以下是範例查詢計劃。

```
EXPLAIN(costs false, verbose true) SELECT func(27+1,10);

                    QUERY PLAN
 ------------------------------------------------
  Foreign Scan
    Output: (func((27 + 1), 10))
    Remote SQL:  SELECT func((27 + 1), 10) AS func
  Single Shard Optimized
 (4 rows)
```

```
EXPLAIN(costs false, verbose true)
 SELECT * FROM customers,func(customer_id, score) WHERE customer_id=10 AND score=27;

                          QUERY PLAN
 ------------------------------------------------------------------------
  Foreign Scan
    Output: customer_id, name, score, func
    Remote SQL:  SELECT customers.customer_id,
      customers.name,
      customers.score,
      func.func
     FROM public.customers,
      LATERAL func(customers.customer_id, customers.score) func(func)
    WHERE ((customers.customer_id = 10) AND (customers.score = 27))
  Single Shard Optimized
 (10 rows)
```

下列範例示範具有 IN 和 OUT 參數作為引數的程序。

```
CREATE OR REPLACE FUNCTION get_data(OUT id INTEGER, IN arg_id INT)
    AS $$
    BEGIN
        SELECT customer_id,
        INTO id
        FROM customer
        WHERE customer_id = arg_id;
    END;
    $$ LANGUAGE plpgsql;
```

下列範例只會使用 IN 參數來分佈程序。

```
EXPLAIN(costs false, verbose true) SELECT * FROM get_data(1);

             QUERY PLAN
 -----------------------------------
  Foreign Scan
    Output: id
    Remote SQL:  SELECT customer_id
     FROM get_data(1) get_data(id)
  Single Shard Optimized
 (6 rows)
```

函數波動

您可以檢查 pg_proc 檢視中的 provolatile 值，來判斷函數是不可變、穩定還是具有波動性。provolatile 值指出函數的結果是否僅取決於其輸入引數，還是受外部因素影響。

值可以是下列其中一項：

- i：不可變函數，一律為相同的輸入提供相同的結果
- s：穩定函數，其結果 (用於固定輸入) 在掃描中不會變更
- v：揮發性函數，其結果可能隨時變更。也請將 v 用於具有副作用的函數，因此無法最佳化對這些函數的呼叫。

下列範例顯示波動性函數。

```
SELECT proname, provolatile FROM pg_proc WHERE proname='pg_sleep';

 proname  | provolatile
----------+-------------
 pg_sleep | v
(1 row)

SELECT proname, provolatile FROM pg_proc WHERE proname='uuid_generate_v4';

     proname      | provolatile
------------------+-------------
 uuid_generate_v4 | v
(1 row)

SELECT proname, provolatile FROM pg_proc WHERE proname='nextval';
```

```
 proname | provolatile
---------+-------------
 nextval | v
(1 row)
```

Aurora PostgreSQL Limitless Database 中不支援變更現有函數的波動。如下列範例所示，這同時適用於 ALTER FUNCTION 和 CREATE OR REPLACE FUNCTION 命令。

```
-- Create an immutable function
CREATE FUNCTION immutable_func1(name text) RETURNS text language plpgsql
AS $$
BEGIN
    RETURN name;
END;
$$IMMUTABLE;

-- Altering the volatility throws an error
ALTER FUNCTION immutable_func1 STABLE;

-- Replacing the function with altered volatility throws an error
CREATE OR REPLACE FUNCTION immutable_func1(name text) RETURNS text language plpgsql
AS $$
BEGIN
    RETURN name;
END;
$$VOLATILE;
```

強烈建議您將正確的波動指派給函數。例如，如果函數從多個資料表使用 SELECT 或參考資料庫物件，請勿將其設為 IMMUTABLE。如果資料表內容變更，不變性就會中斷。

Aurora PostgreSQL 允許不可變函數中的 SELECT，但結果可能不正確。Aurora PostgreSQL Limitless Database 可能會同時傳回錯誤和不正確的結果。如需函數波動的詳細資訊，請參閱 PostgreSQL 文件中的函數波動類別。

序列

具名序列是以遞增或遞減順序產生唯一數字的資料庫物件。CREATE SEQUENCE 會建立新的序號產生器。序列值保證是唯一的。

當您在 Aurora PostgreSQL Limitless Database 中建立具名序列時，系統會建立分散式序列物件。然後，Aurora PostgreSQL Limitless Database 會將非重疊的序列值區塊分佈到所有分散式交易路由器 (路由器)。區塊表示為路由器上的本機序列物件；因此，nextval 和 currval 等序列操作會在本機執行。路由器會獨立運作，並在需要時向分散式序列請求新的區塊。

如需序列的詳細資訊，請參閱 PostgreSQL 文件中的 CREATE SEQUENCE。

主題

- 請求新的區塊
- 限制
- 不支援的選項
- 範例
- 序列檢視
- 針對序列問題進行疑難排解

請求新的區塊

您可以使用 rds_aurora.limitless_sequence_chunk_size 參數設定在路由器上分配的區塊大小。預設值為 250000。每個路由器最初擁有兩個區塊：作用中和預留。作用中區塊會用於設定本機序列物件 (設定 minvalue 和 maxvalue)，而預留區塊會存放在內部目錄資料表中。當作用中區塊達到最小值或最大值時，會以預留區塊取代之。若要這樣做，系統會在內部使用 ALTER SEQUENCE，表示已取得 AccessExclusiveLock。

背景工作者每 10 秒在路由器節點執行一次，以掃描已使用預留區塊的序列。如果找到使用過的區塊，工作者會從分散式序列請求新的區塊。請務必設定夠大的區塊大小，讓背景工作者有足夠的時間來請求新的區塊。遠端請求永遠不會發生在使用者工作階段的情況中，這表示您無法直接請求新的序列。

限制

下列限制適用於 Aurora PostgreSQL Limitless Database 中的序列：

- pg_sequence 目錄、pg_sequences 函數和 SELECT * FROM *sequence_name* 陳述式都只顯示本機序列狀態 (而不是分散式狀態)。

- 序列值保證是唯一的，而且在工作階段中保證是單調的。但是，如果那些工作階段連接到其他路由器，則在其他工作階段中執行的 nextval 陳述式可能會不按照順序。
- 請確定序列大小 (可用值的數量) 夠大，可以分佈到所有路由器。使用 rds_aurora.limitless_sequence_chunk_size 參數來設定 chunk_size。(每個路由器都有兩個區塊。)
- 支援 CACHE 選項，但快取必須小於 chunk_size。

不支援的選項

Aurora PostgreSQL Limitless Database 中的序列不支援下列選項。

序列處理函式

不支援 setval 函數。如需詳細資訊，請參閱 PostgreSQL 文件中的序列操作函數。

CREATE SEQUENCE

不支援下列選項。

```
CREATE [{ TEMPORARY | TEMP} | UNLOGGED] SEQUENCE
    [[ NO ] CYCLE]
```

如需詳細資訊，請參閱 PostgreSQL 文件中的 CREATE SEQUENCE。

ALTER SEQUENCE

不支援下列選項。

```
ALTER SEQUENCE
    [[ NO ] CYCLE]
```

如需詳細資訊，請參閱 PostgreSQL 文件中的 ALTER SEQUENCE。

ALTER TABLE

針對序列，不支援 ALTER TABLE 命令。

範例

CREATE/DROP SEQUENCE

```
postgres_limitless=> CREATE SEQUENCE s;
```

```
CREATE SEQUENCE

postgres_limitless=> SELECT nextval('s');

 nextval
---------
       1
(1 row)

postgres_limitless=> SELECT * FROM pg_sequence WHERE seqrelid='s'::regclass;

 seqrelid | seqtypid | seqstart | seqincrement | seqmax | seqmin | seqcache |
 seqcycle
----------+----------+----------+--------------+--------+--------+----------
+----------
    16960 |       20 |        1 |            1 |  10000 |      1 |        1 | f
(1 row)

% connect to another router
postgres_limitless=> SELECT nextval('s');

 nextval
---------
   10001
(1 row)

postgres_limitless=> SELECT * FROM pg_sequence WHERE seqrelid='s'::regclass;

 seqrelid | seqtypid | seqstart | seqincrement | seqmax | seqmin | seqcache |
 seqcycle
----------+----------+----------+--------------+--------+--------+----------
+----------
    16959 |       20 |    10001 |            1 |  20000 |  10001 |        1 | f
(1 row)

postgres_limitless=> DROP SEQUENCE s;
DROP SEQUENCE
```

ALTER SEQUENCE

```
postgres_limitless=> CREATE SEQUENCE s;
CREATE SEQUENCE
```

```
postgres_limitless=> ALTER SEQUENCE s RESTART 500;
ALTER SEQUENCE

postgres_limitless=> SELECT nextval('s');

 nextval
---------
     500
(1 row)

postgres_limitless=> SELECT currval('s');

 currval
---------
     500
(1 row)
```

序列處理函式

```
postgres=# CREATE TABLE t(a bigint primary key, b bigint);
CREATE TABLE

postgres=# CREATE SEQUENCE s minvalue 0 START 0;
CREATE SEQUENCE

postgres=# INSERT INTO t VALUES (nextval('s'), currval('s'));

INSERT 0 1

postgres=# INSERT INTO t VALUES (nextval('s'), currval('s'));
INSERT 0 1

postgres=# SELECT * FROM t;

 a | b
---+---
 0 | 0
 1 | 1
(2 rows)

postgres=# ALTER SEQUENCE s RESTART 10000;
ALTER SEQUENCE
```

```
postgres=# INSERT INTO t VALUES (nextval('s'), currval('s'));

INSERT 0 1

postgres=# SELECT * FROM t;

   a   |   b
-------+-------
     0 |     0
     1 |     1
 10000 | 10000
(3 rows)
```

序列檢視

Aurora PostgreSQL Limitless Database 提供下列序列檢視。

rds_aurora.limitless_distributed_sequence

此檢視會顯示分散式序列狀態和組態。minvalue、maxvalue、start、inc 和 cache 欄具有與 pg_sequences 檢視中相同的意義，以及顯示序列建立時所用的選項。lastval 欄會顯示分散式序列物件中最新分配或預留的值。這並不表示值已有人使用，因為路由器會在本機保留序列區塊。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_distributed_sequence WHERE
 sequence_name='test_serial_b_seq';

 schema_name |   sequence_name   | lastval | minvalue |  maxvalue  | start | inc |
 cache
-------------+-------------------+---------+----------+------------+-------+-----
+-------
 public      | test_serial_b_seq | 1250000 |        1 | 2147483647 |     1 |   1 |
   1
(1 row)
```

rds_aurora.limitless_sequence_metadata

此檢視會顯示分散式序列中繼資料，並從叢集節點彙總序列中繼資料。其會使用下列欄：

- subcluster_id：擁有區塊的叢集節點 ID。
- 作用中區塊：使用中的序列區塊 (active_minvalue、active_maxvalue)。
- 預留區塊：接下來將使用的本機區塊 (reserved_minvalue、reserved_maxvalue)。
- local_last_value：來自本機序列的最後一個觀察值。
- chunk_size：區塊的大小 (如建立時所設定)。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_sequence_metadata WHERE
 sequence_name='test_serial_b_seq' order by subcluster_id;

 subcluster_id |   sequence_name   | schema_name | active_minvalue | active_maxvalue
 | reserved_minvalue | reserved_maxvalue | chunk_size | chunk_state |
 local_last_value
---------------+-------------------+-------------+-----------------
+-----------------+-------------------+-------------------+------------
+-------------+------------------
```

```
 1             | test_serial_b_seq | public      |          500001 |          750000
 |           1000001 |           1250000 |     250000 |           1 |
 550010
 2             | test_serial_b_seq | public      |          250001 |          500000
 |            750001 |           1000000 |     250000 |           1 |

(2 rows)
```

針對序列問題進行疑難排解

序列可能發生下列問題。

區塊大小不夠大

如果區塊大小設定不夠大且交易速率很高，背景工作者可能沒有足夠的時間，無法在作用中區塊用盡前請求新的區塊。這可能會導致爭用和等待事件，例如 LIMITLESS:AuroraLimitlessSequenceReplace、LWLock:LockManager、Lockrelation 和 LWlock:bufferscontent。

增加 rds_aurora.limitless_sequence_chunk_size 參數的值。

序列快取設定過高

在 PostgreSQL 中，序列快取會在工作階段層級發生。每個工作階段會在一次存取序列物件期間分配連續的序列值，並相應地增加序列物件的 last_value。然後，下次使用該工作階段中的 nextval 只會傳回預先分配的值 (而不會接觸序列物件)。

當工作階段結束時，工作階段中分配但未使用的任何數字都會遺失，導致序列中出現「洞」。這可以快速使用 sequence_chunk，並導致爭用和等待事件，例如 LIMITLESS:AuroraLimitlessSequenceReplace、LWLock:LockManager、Lockrelation 和 LWlock:bufferscontent。

減少序列快取設定。

下圖顯示序列問題引起的等待事件。

Database load Info
Viewed as
Absolute
Sliced by
Waits
Bar
Line
Table
Show estimated vCPU
Load is measured in average active sessions (AAS). The vCPU value is determined by the number of vCPU (virtual CPU) cores for your DB instance.
Average active sessions (AAS)
2,000
1,500
1,000
500
0
05:42:55
05:43
05:43:05
05:43:10
05:43:15
05:43:20
05:43:25
05:43:30
05:43:35
05:43:40
05:43:45
05:43:50
05:43:55
05:44
05:44:05
05:44:10
05:44:15
05:44:20
IPC:ProcArrayGroupUpdate
LIMITLESS:LimitlessConnect
LIMITLESS:LimitlessExecute
IO:XactSync
LIMITLESS:LimitlessModify
Lock:relation
LWLock:LockManager
LIMITLESS:AuroraLimitlessSequenceRe
LWLock:BufferContent
CPU
Dimensions
Load distribution
Top instances
Top waits
Top SQL
Top hosts
Top users
Top session types
Top applications
Top databases
Top SQL (25)
Find SQL statements
1 2 3
Load by waits (AAS)
SQL statements
441.03
select nextval(?) as order_id
2.50
INSERT INTO ec_sample.orderdetails (customer_id,order_id,orderline_id,product_id...
1.39
INSERT INTO ec_sample.orders (customer_id,order_id,order_date,order_status,updat...

支援和不支援的資料處理語言 (DML) 和查詢處理 SQL 命令

下表列出 Aurora PostgreSQL Limitless Database 支援和不支援的 DML 命令，並參考限制或更多資訊。

命令	支援？	限制或更多資訊
ABORT	是	無
ANALYZE	是	ANALYZE
BEGIN	是	無
CALL	是	無
CHECKPOINT	是	無
CLOSE	是	無
CLUSTER	是	CLUSTER
COMMIT	是	無
COMMIT PREPARED	否	不適用
COPY	是	無
DEALLOCATE	是	無
DECLARE	是	無
DELETE	是	無
DISCARD	是	無
DO	是	無
結束	是	無
EXECUTE	是	無
EXPLAIN	是	EXPLAIN

命令	支援？	限制或更多資訊
FETCH	是	無
IMPORT FOREIGN SCHEMA	否	不適用
INSERT	是	INSERT
LISTEN	否	不適用
LOCK	是	無
MERGE	否	不適用
MOVE	是	無
NOTIFY	否	不適用
OPEN	是	無
PREPARE	是	無
PREPARE TRANSACTION	否	不適用
REFRESH MATERIALIZED VIEW	否	不適用
REINDEX	否	不適用
RELEASE SAVEPOINT	是	無
ROLLBACK	是	無
ROLLBACK PREPARED	否	不適用
ROLLBACK TO SAVEPOINT	是	無
SAVEPOINT	是	無
SELECT	是	無
SELECT INTO	是	無

命令	支援？	限制或更多資訊
SHOW	是	無
START TRANSACTION	是	無
UNLISTEN	否	無
UPDATE	是	UPDATE
UPDATE … WHERE CURRENT OF	否	不適用
VACUUM	是	VACUUM
VALUES	是	無

Aurora PostgreSQL Limitless Database 的 DML 限制和其他資訊

下列主題說明限制，或提供 Aurora PostgreSQL Limitless Database 中 DML 和查詢處理 SQL 命令的詳細資訊。

主題

- ANALYZE
- CLUSTER
- EXPLAIN
- INSERT
- UPDATE
- VACUUM

ANALYZE

ANALYZE 命令會收集資料庫中資料表內容的統計資料。之後，查詢規劃器會使用這些統計資料，來協助判斷最有效率的查詢執行計劃。如需詳細資訊，請參閱 PostgreSQL 說明文件中的 ANALYZE。

在 Aurora PostgreSQL Limitless Database 中，ANALYZE 命令會在執行時收集所有路由器和碎片中的資料表統計資料。

為了防止在 ANALYZE 執行期間計算每個路由器的統計資料，系統會在其中一個路由器計算資料表統計資料，然後複製到對等路由器。

CLUSTER

CLUSTER 命令會根據索引實際重新排序資料表。必須已在資料表上定義索引。在 Aurora PostgreSQL Limitless Database 中，叢集僅對於存在於每個碎片的索引部分有效。

如需詳細資訊，請參閱 PostgreSQL 文件中的 CLUSTER。

EXPLAIN

您可以使用下列參數，來設定來自 EXPLAIN 命令的輸出：

- `rds_aurora.limitless_explain_options`：EXPLAIN 輸出中要包含的內容。預設值為 `single_shard_optimization`：會顯示計劃是否為單一碎片最佳化，但不包含碎片計劃。

在此範例中，EXPLAIN 輸出不會顯示碎片中的計劃。

```
postgres_limitless=> EXPLAIN SELECT * FROM employees where id =25;

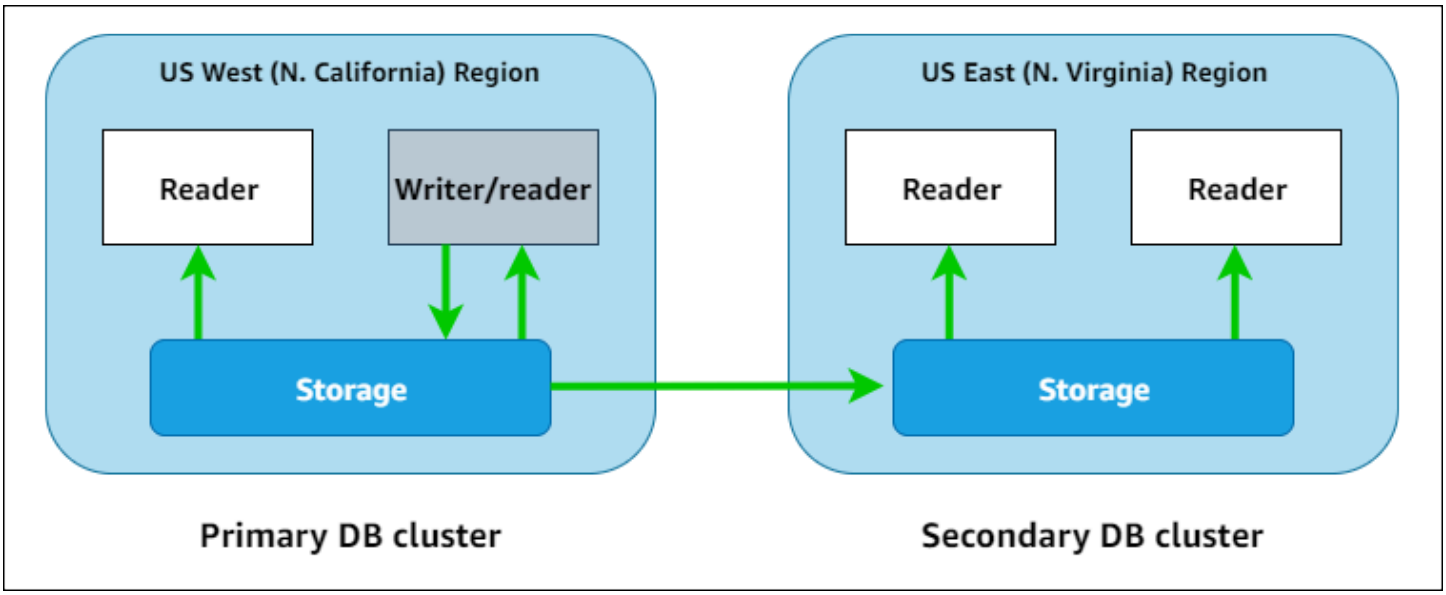
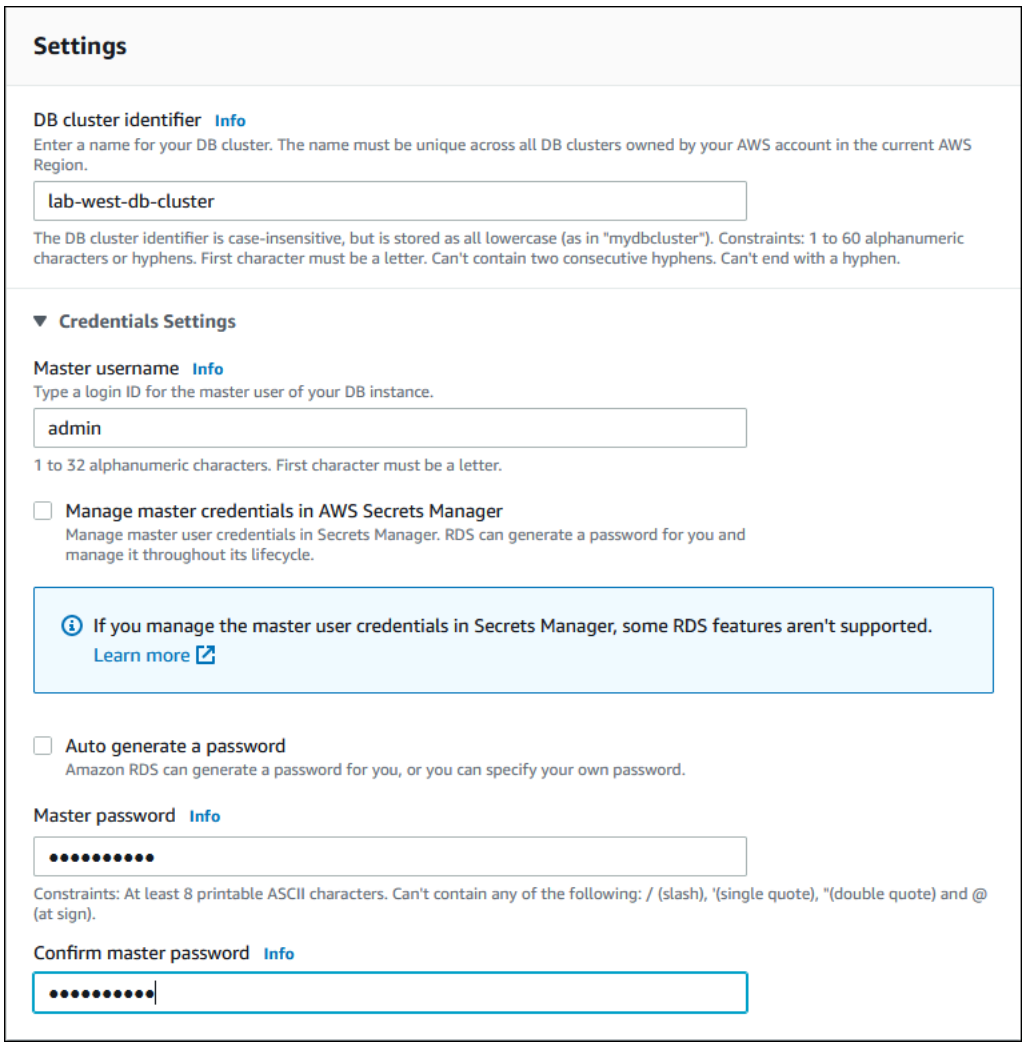
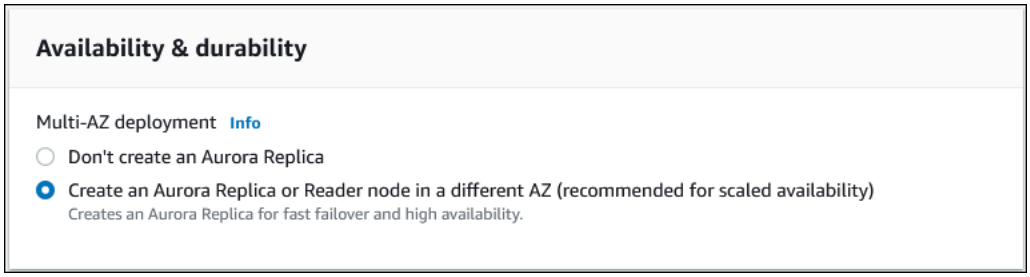
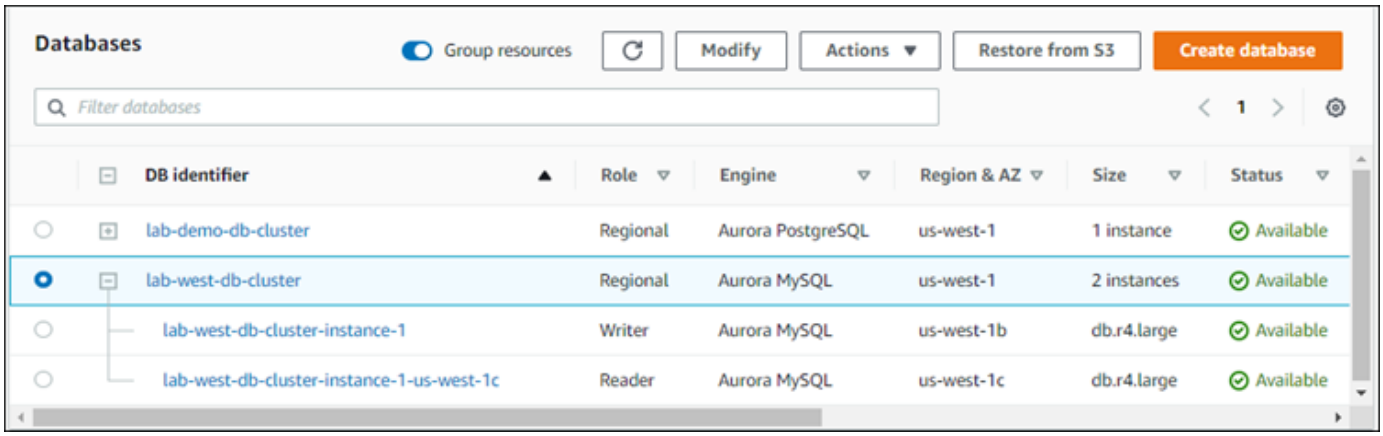
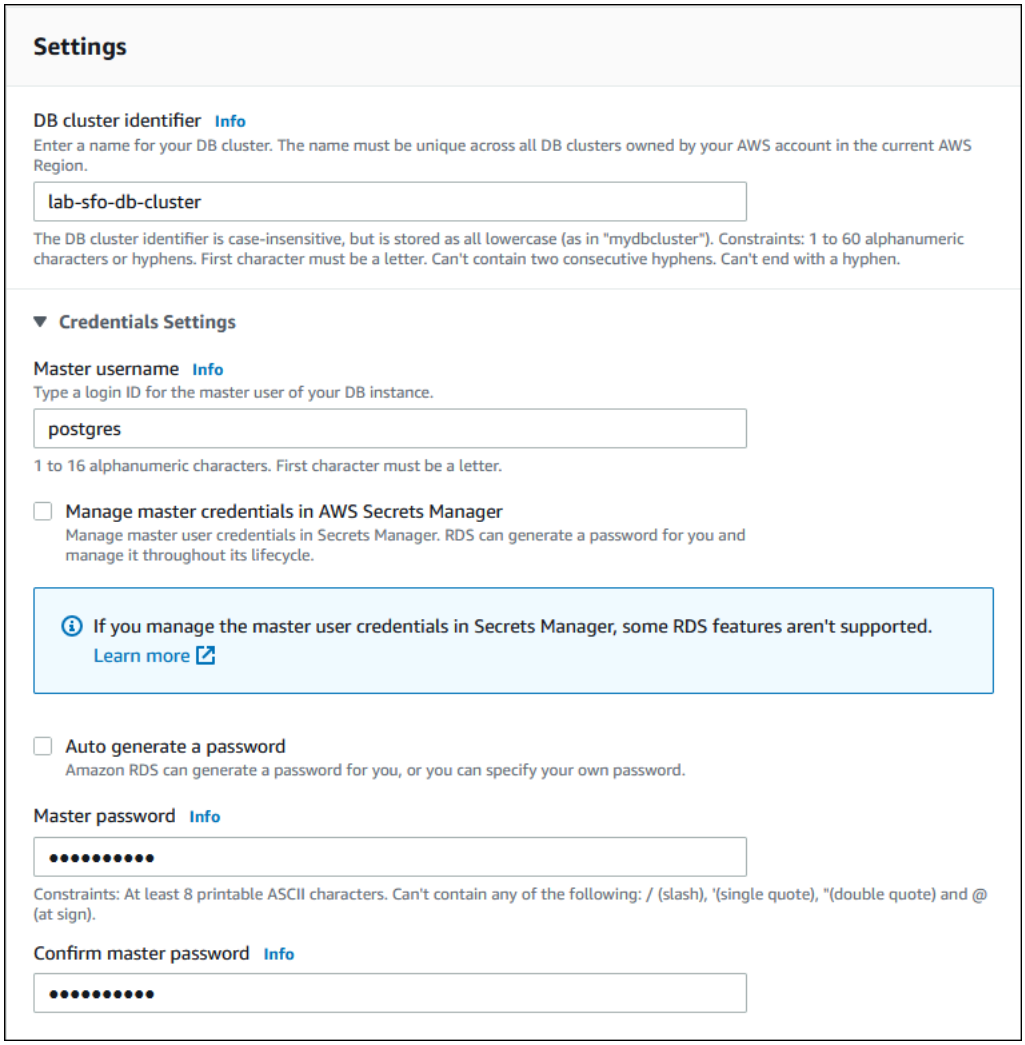
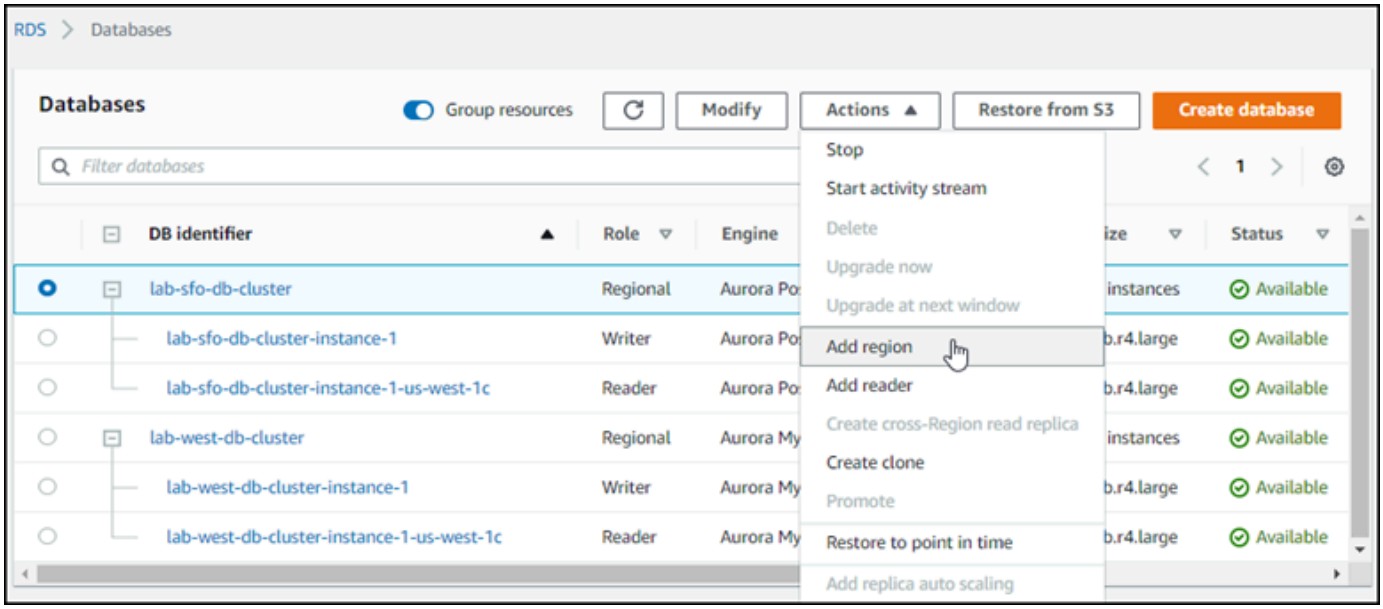
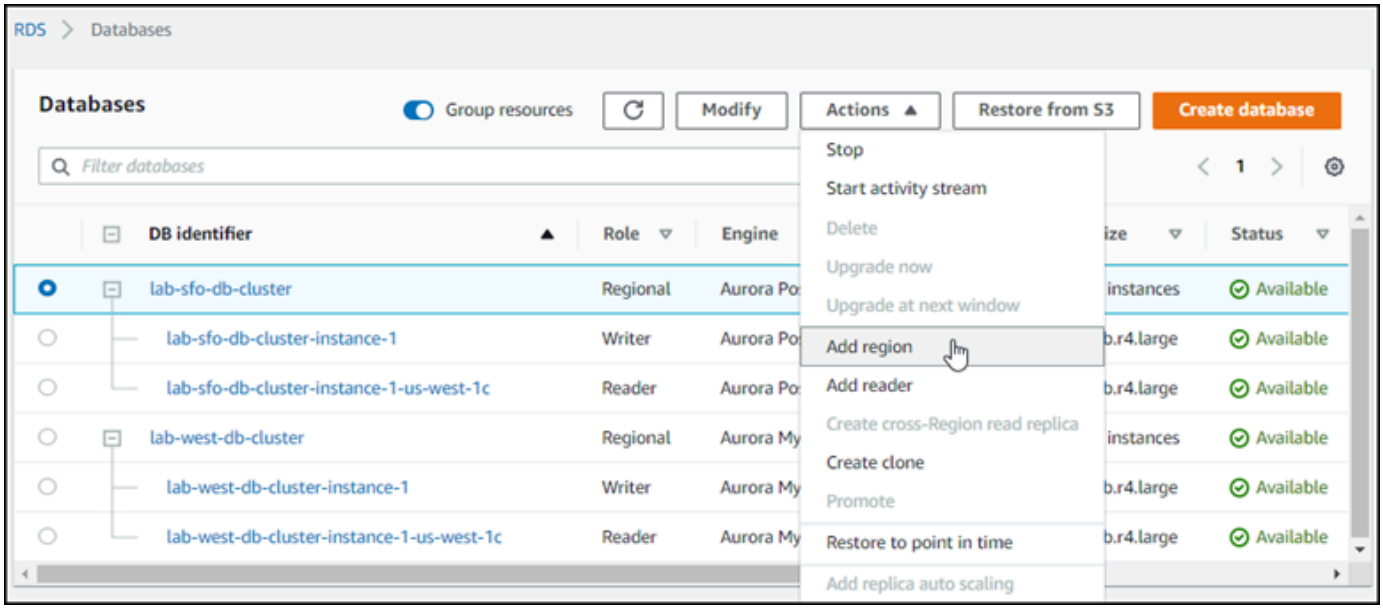
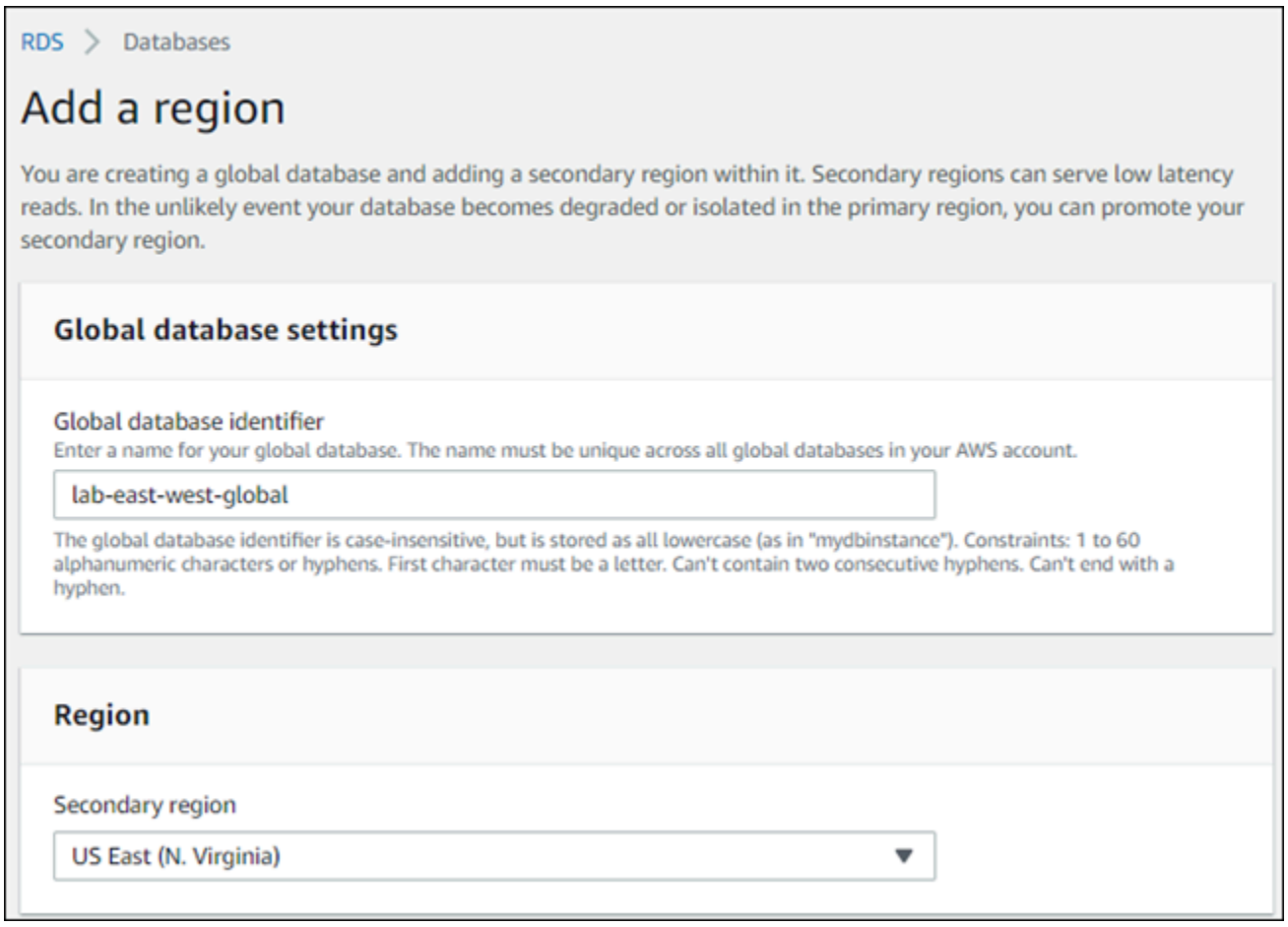
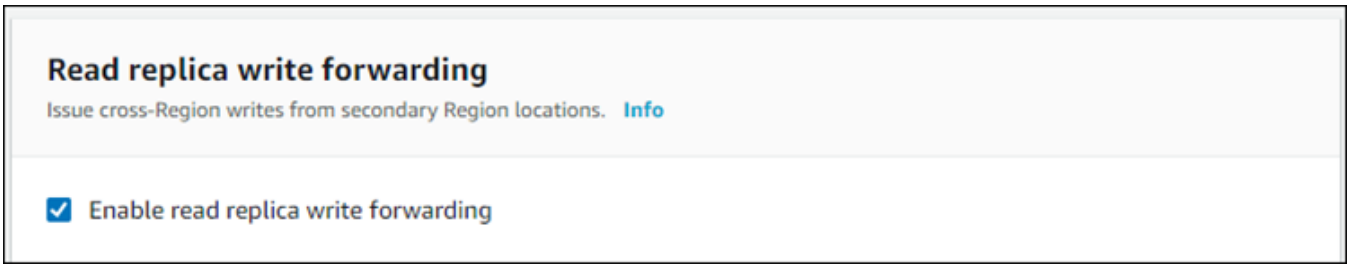
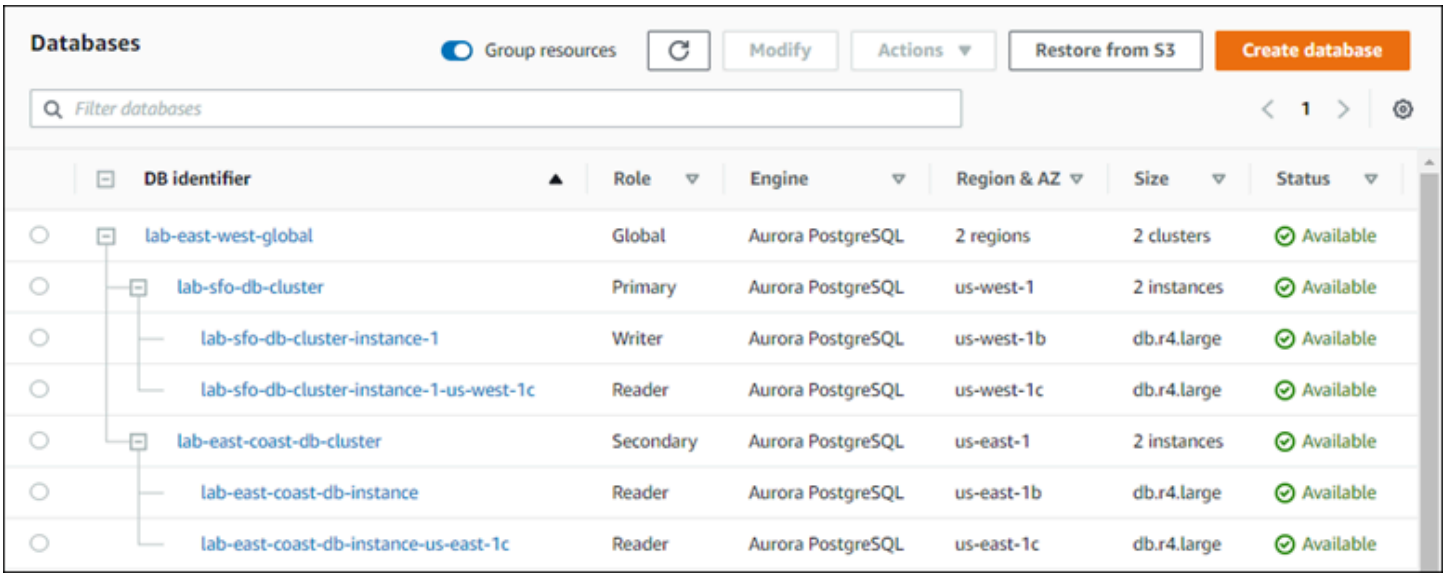
                      QUERY PLAN
------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
 Single Shard Optimized
(2 rows)
```

現在，我們將 `rds_aurora.limitless_explain_options` 設定為包含 `shard_plans` 和 `single_shard_optimization`。我們可以檢視路由器和碎片上陳述式的執行計劃。此外，我們會停用 `enable_seqscan` 參數，強制在碎片層使用索引掃描。

```
postgres_limitless=> SET rds_aurora.limitless_explain_options = shard_plans,
 single_shard_optimization;
SET

postgres_limitless=> SET enable_seqscan = OFF;
SET

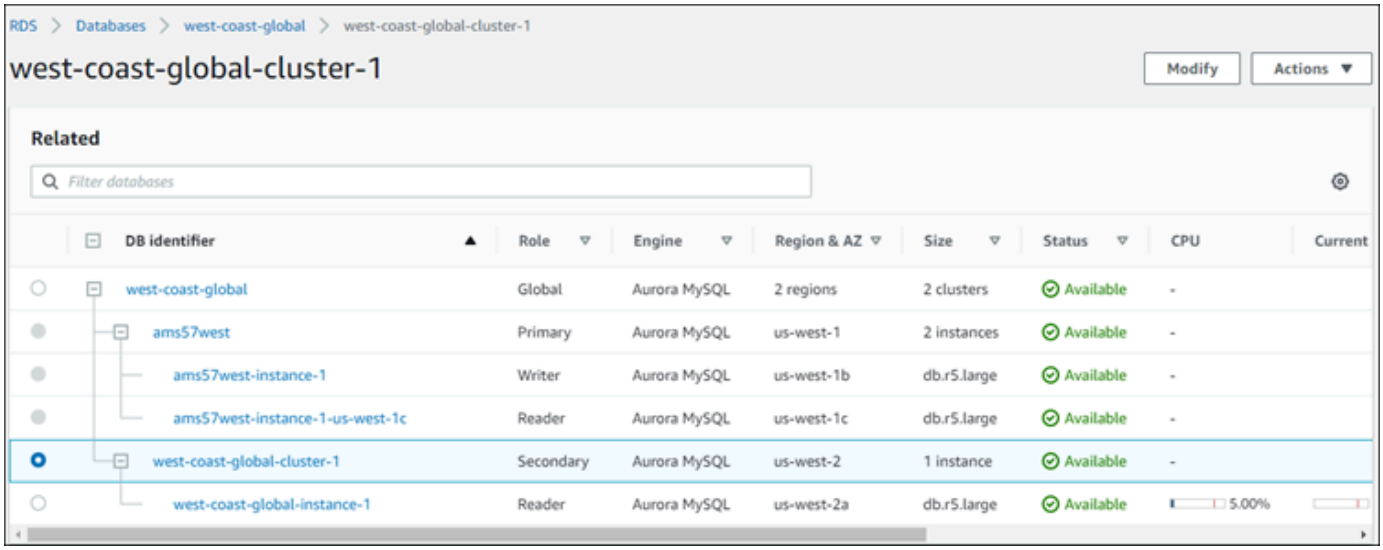
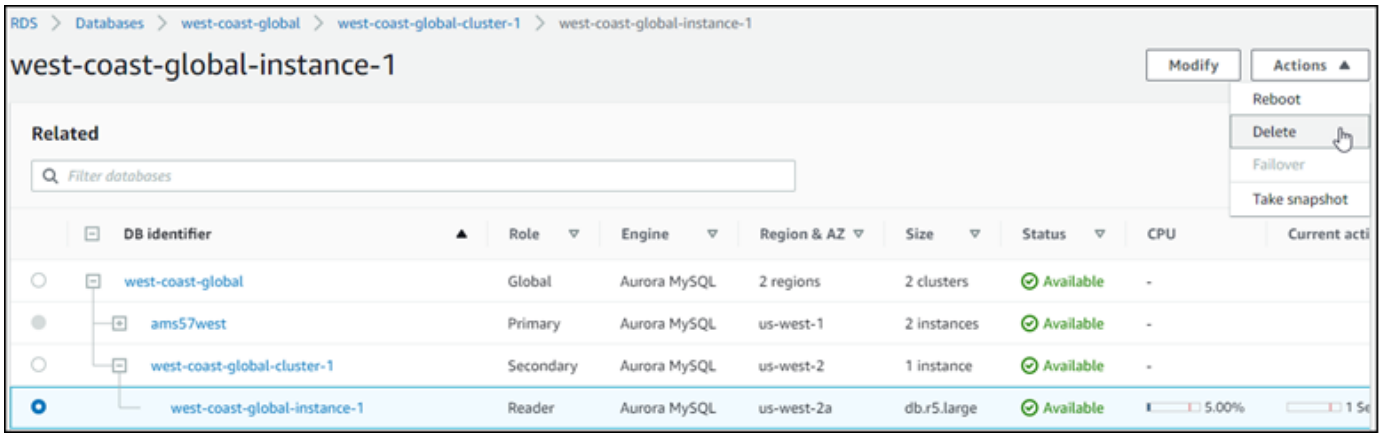
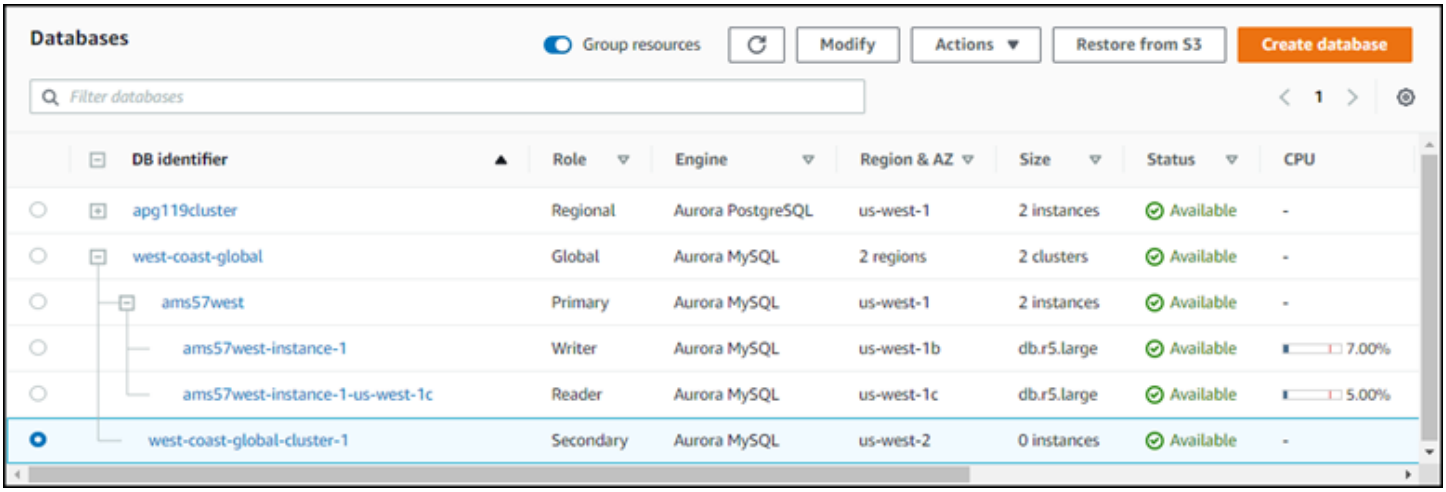
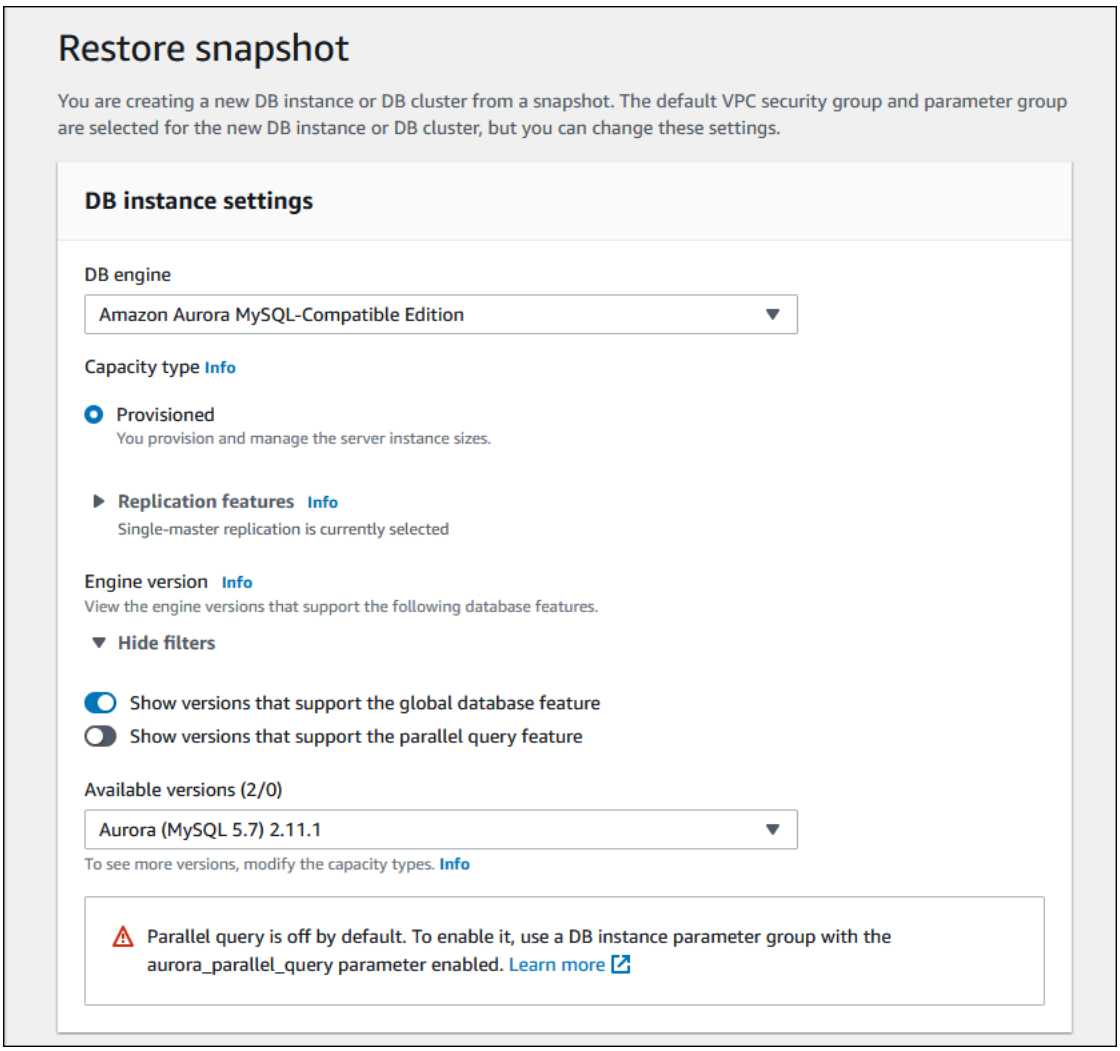
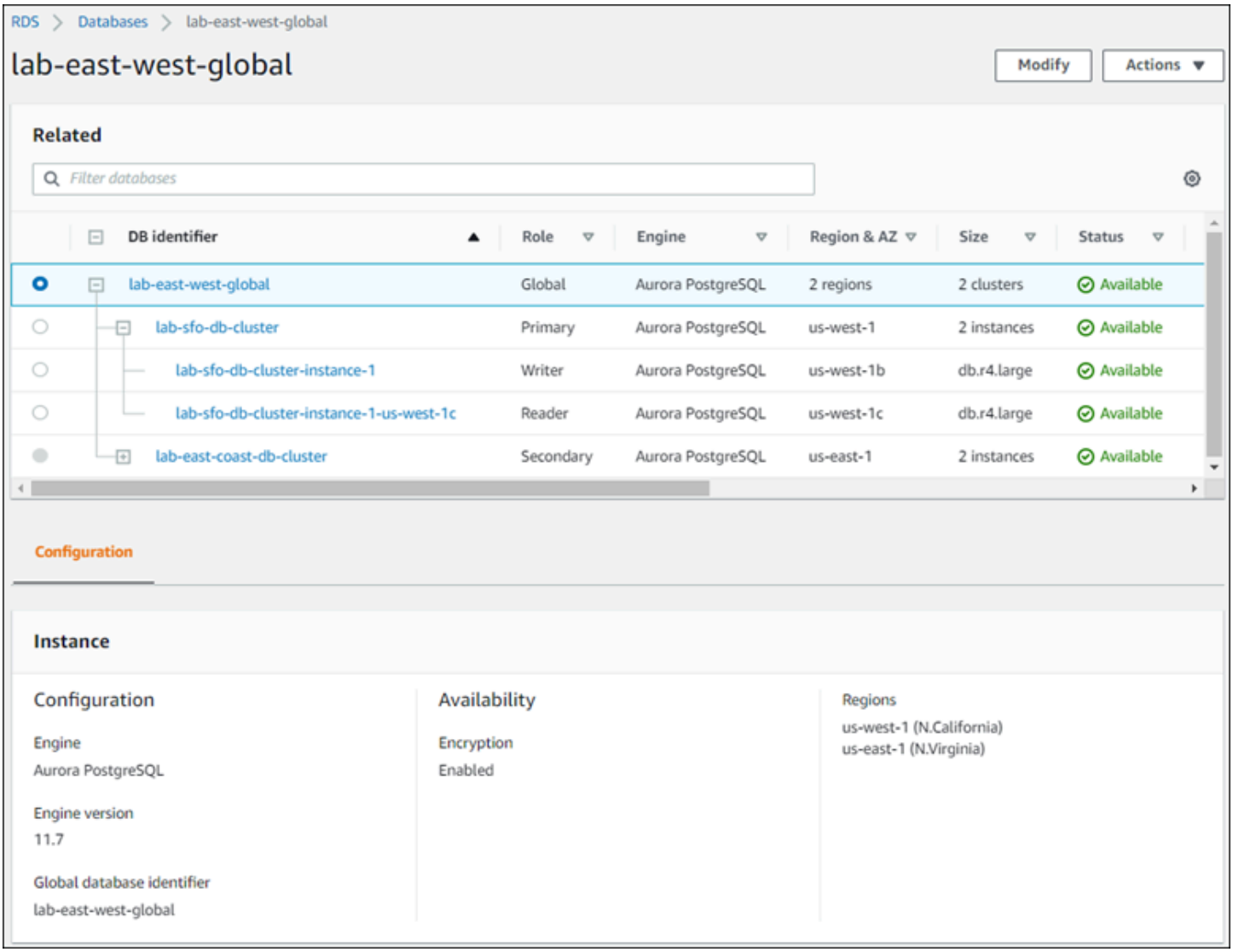
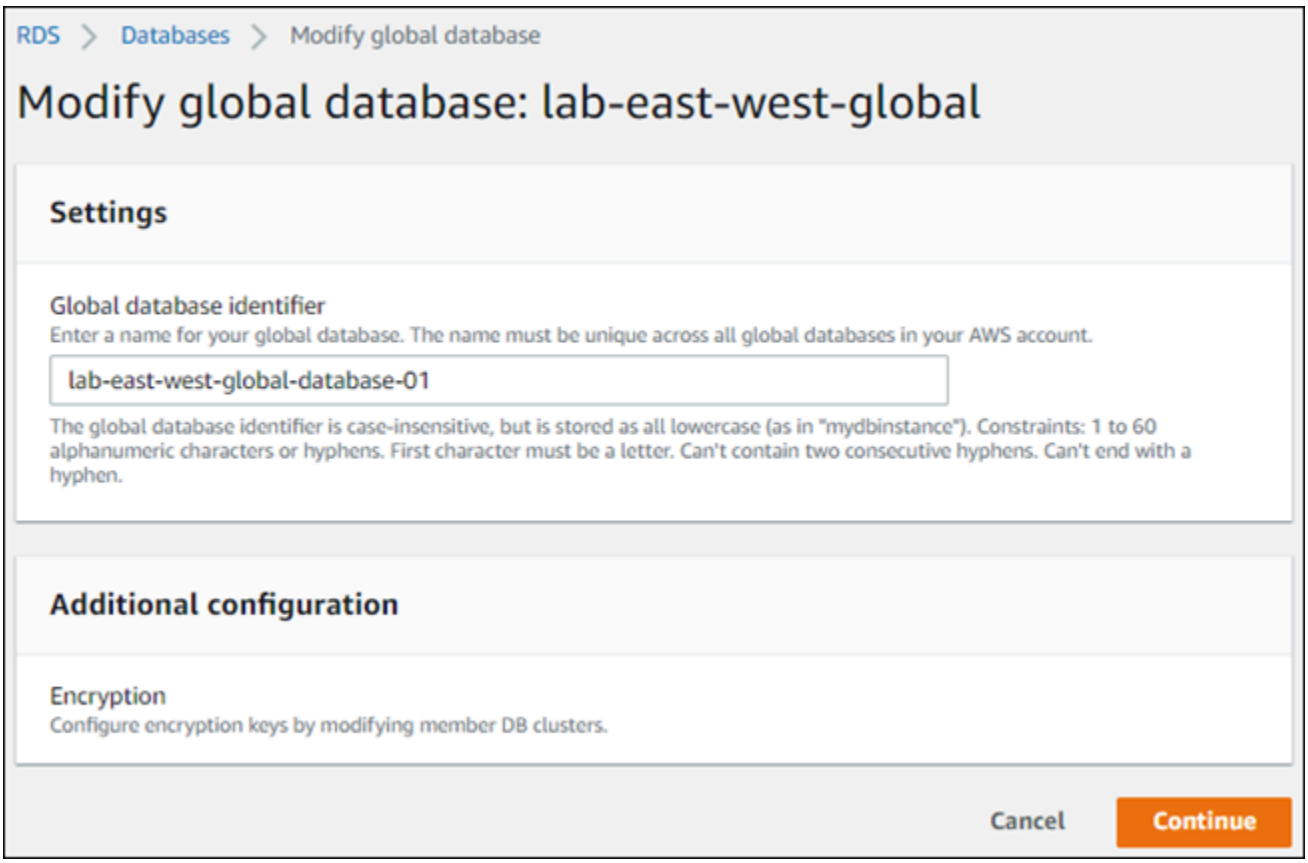
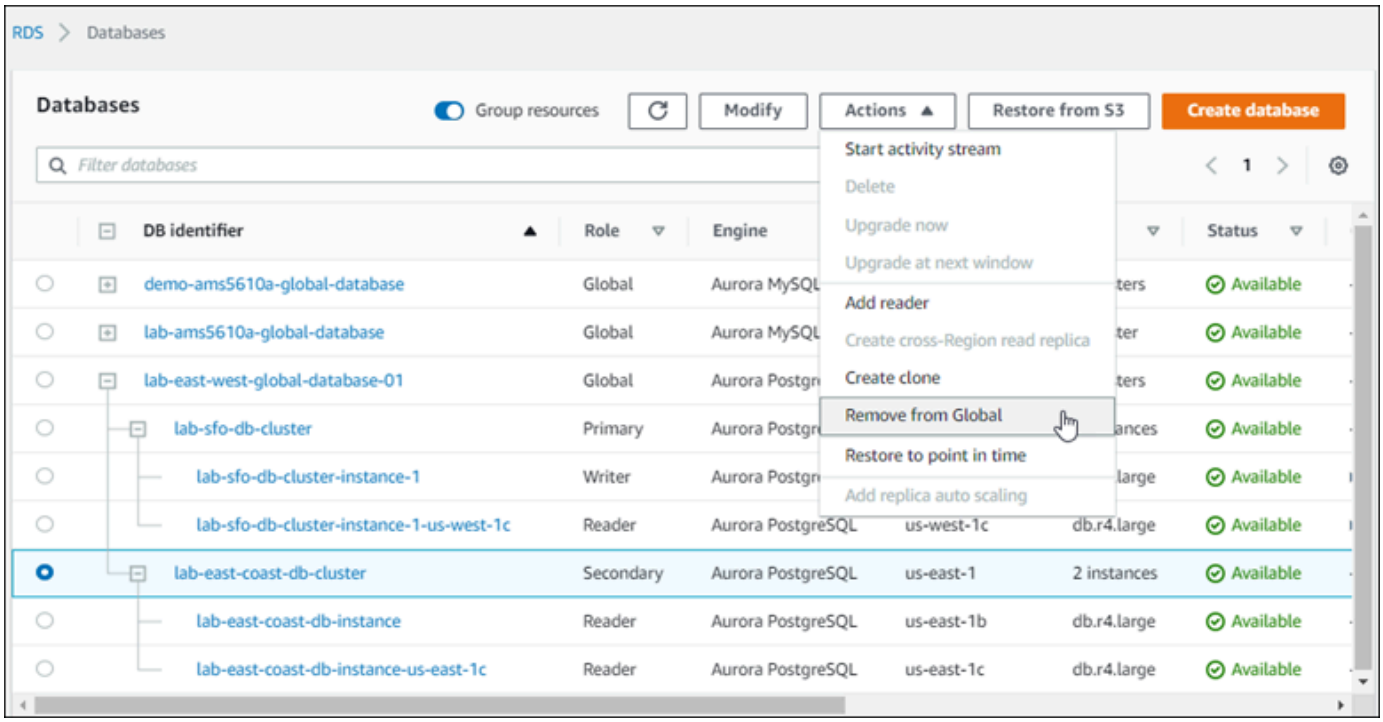
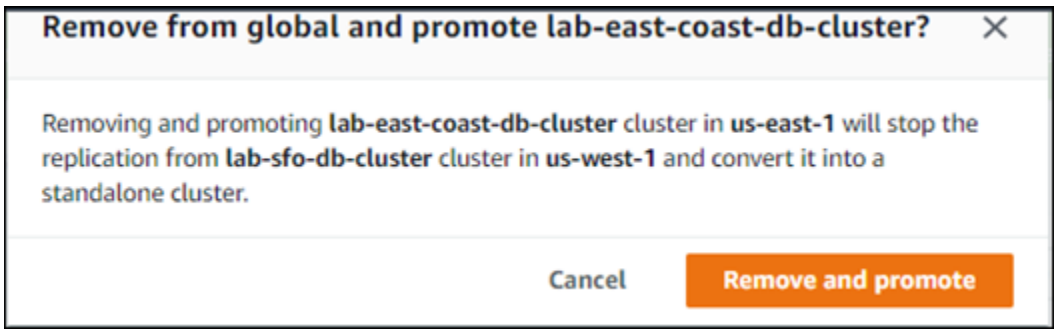
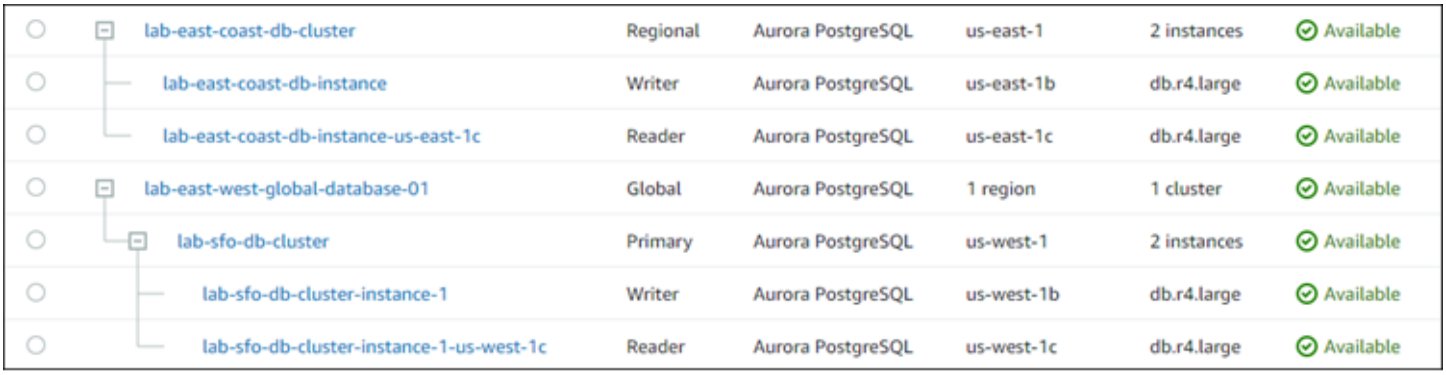
postgres_limitless=> EXPLAIN SELECT * FROM employees WHERE id = 25;

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
   Remote Plans from Shard postgres_s4:
         Index Scan using employees_ts00287_id_idx on employees_ts00287
 employees_fs00003  (cost=0.14..8.16 rows=1 width=15)
           Index Cond: (id = 25)
 Single Shard Optimized
(5 rows)
```

如需 EXPLAIN 命令的詳細資訊，請參閱 PostgreSQL 文件中的 EXPLAIN。

INSERT

Aurora PostgreSQL Limitless Database 中支援大多數 INSERT 命令。

PostgreSQL 沒有明確的 UPSERT 命令，但支援 INSERT ... ON CONFLICT 陳述式。

如果衝突動作具有子查詢或可變函數，則不支援 INSERT ... ON CONFLICT：

```
-- RANDOM is a mutable function.
INSERT INTO sharded_table VALUES (1, 100) ON CONFLICT (id) DO UPDATE SET other_id =
 RANDOM();

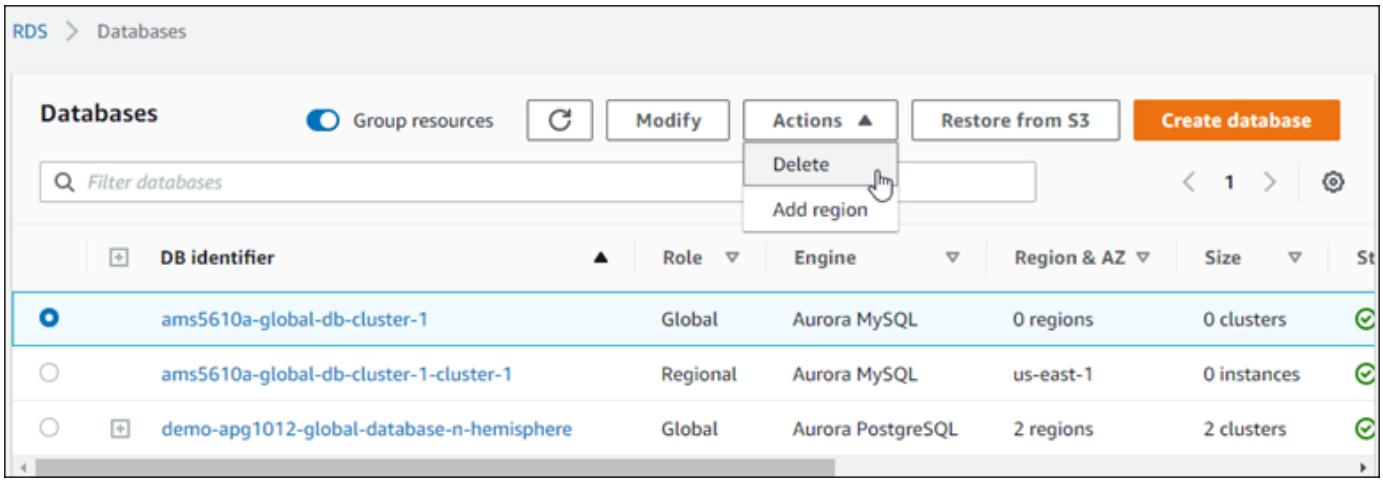
ERROR: Aurora Limitless Tables doesn't support pushdown-unsafe functions with DO UPDATE
 clauses.
```

如需 INSERT 命令的詳細資訊，請參閱 PostgreSQL 文件中的 INSERT。

UPDATE

不支援更新碎片索引鍵。例如，您有一個名為 customers 的碎片資料表 (內含碎片索引鍵 customer_id)。下列 DML 陳述式會導致錯誤：

```
postgres_limitless=> UPDATE customers SET customer_id = 11 WHERE customer_id =1;
ERROR:  Shard key column update is not supported

postgres_limitless=> UPDATE customers SET customer_id = 11 WHERE customer_name='abc';
ERROR:  Shard key column update is not supported
```

若要更新碎片索引鍵，首先您必須 DELETE 具有碎片索引鍵的列，然後 INSERT 具有更新碎片索引鍵值的新列。

如需 UPDATE 命令的詳細資訊，請參閱 PostgreSQL 文件中的更新資料。

VACUUM

您可以在碎片和參考資料表執行清空。Aurora PostgreSQL Limitless Database 中完全支援下列 VACUUM 函數：

- VACUUM
- ANALYZE
- DISABLE_PAGE_SKIPPING

- FREEZE
- FULL
- INDEX_CLEANUP
- PARALLEL
- PROCESS_TOAST
- TRUNCATE
- 詳細

Aurora PostgreSQL Limitless Database 上的 VACUUM 具有下列限制：

- 不支援 pg_visibility_map 延伸模組。
- 不支援使用 pg_stat_all_indexes 檢視檢查未使用的索引。
- 未實作 pg_stat_user_indexes、pg_class 和 pg_stats 的合併檢視。

如需 VACUUM 命令的詳細資訊，請參閱 PostgreSQL 文件中的 VACUUM。如需如何在 Aurora PostgreSQL Limitless Database 中進行清空的詳細資訊，請參閱透過清空回收儲存空間。

Aurora PostgreSQL Limitless Database 中的變數

您可以使用下列變數來設定 Aurora PostgreSQL Limitless Database。

rds_aurora.limitless_active_shard_key

在查詢資料庫時設定單一碎片索引鍵，導致所有 SELECT 和 DML 查詢都附加在碎片索引鍵作為常數述詞。如需更多詳細資訊，請參閱 設定作用中碎片索引鍵。

rds_aurora.limitless_create_table_collocate_with

將此變數設為特定資料表名稱，以將新建立的資料表與該資料表共置。如需更多詳細資訊，請參閱 使用變數建立無限資料表。

rds_aurora.limitless_create_table_mode

設定資料表建立模式。如需更多詳細資訊，請參閱 使用變數建立無限資料表。

rds_aurora.limitless_create_table_shard_key

將此變數設為欄名稱的陣列，以用作碎片索引鍵。如需更多詳細資訊，請參閱 使用變數建立無限資料表。

rds_aurora.limitless_explain_options

EXPLAIN 輸出中要包含的內容。如需更多詳細資訊，請參閱 EXPLAIN。

Aurora PostgreSQL Limitless Database 中的資料庫叢集參數

您可以使用下列資料庫叢集參數來設定 Aurora PostgreSQL Limitless Database。

rds_aurora.limitless_adaptive_fetch_size

增強批次預先擷取。設為 `true` 時，此參數允許自我調整 (適應性) 擷取大小以進行預先擷取。設為 `false` 時，擷取大小為常數。

rds_aurora.limitless_auto_scale_options

設定可用於在資料庫碎片群組中新增路由器或分割碎片的選項。此值可以為 `add_router` 或 `split_shard` (或兩者)。

如需詳細資訊，請參閱將路由器新增至資料庫碎片群組及在資料庫碎片群組中分割碎片。

rds_aurora.limitless_distributed_deadlock_timeout

檢查是否有分散式死鎖條件之前等待鎖定的時間 (以毫秒為單位)。預設值為 `1000` (1 秒)。

如需更多詳細資訊，請參閱 Aurora PostgreSQL Limitless Database 中的分散式死鎖。

rds_aurora.limitless_enable_auto_scale

啟用在資料庫碎片群組中路由器的新增和碎片的分割。

如需詳細資訊，請參閱將路由器新增至資料庫碎片群組及在資料庫碎片群組中分割碎片。

rds_aurora.limitless_finalize_split_shard_mode

決定如何完成系統啟動的碎片分割。如需更多詳細資訊，請參閱 在資料庫碎片群組中分割碎片。

rds_aurora.limitless_maximum_adaptive_fetch_size

設定自適應擷取大小的上限。範圍介於 1 和 `INT_MAX` 之間。預設值為 `1000`。

使用 Amazon Aurora 全球資料庫

使用 Amazon Aurora 全球資料庫功能，您可以設定跨越多個 AWS 區域的多個 Aurora 資料庫叢集。Aurora 會自動將所有在主要資料庫叢集中所做的變更同步至一或多個次要叢集。Aurora 全球資料庫在一個區域中具有主要資料庫叢集，以及在不同區域中具有最多 10 個次要資料庫叢集。此多區域組態可從可能影響整個 AWS 區域的罕見中斷提供快速復原。在多個地理位置擁有您所有資料的完整副本，也能讓從世界各地廣泛分開位置連線的應用程式進行低延遲讀取操作。

主題

- Amazon Aurora 全球資料庫的概觀
- Amazon Aurora 全球資料庫的優點
- 區域和版本可用性
- Amazon Aurora 全球資料庫的限制
- Amazon Aurora 全球資料庫入門
- 管理 Amazon Aurora 全域資料庫
- 連線至 Amazon Aurora 全域資料庫
- 在 Amazon Aurora 全域資料庫中使用寫入轉送
- 在 Amazon Aurora 全球資料庫中使用切換或容錯移轉
- 監控 Amazon Aurora 全域資料庫
- 針對 Amazon Aurora Global Database 使用藍/綠部署
- 將 Amazon Aurora 全域資料庫與其他 AWS 服務搭配使用
- 升級 Amazon Aurora 全域資料庫

Amazon Aurora 全球資料庫的概觀

透過使用 Amazon Aurora 全球資料庫功能，您可以使用跨多個 AWS 區域的單一 Aurora 資料庫來執行全域分散式應用程式。

Aurora 全域資料庫包含一個寫入資料的主要 AWS 區域 資料庫，以及最多 10 個唯讀次要 AWS 區域資料庫。在主要 AWS 區域中，您將寫入操作發佈至主要資料庫叢集。最方便的方式是連線到 Aurora 全球資料庫寫入器端點，即使切換或容錯移轉至不同的 AWS 區域，它仍會指向主要資料庫叢集。進行任何寫入操作後，Aurora AWS 區域 會使用專用基礎設施將資料複寫至次要 ，延遲通常不到一秒。

在下圖中，您可以找到跨越兩個 的 Aurora 全域資料庫範例 AWS 區域。



您可以透過新增一或多個 Aurora 讀取器複本來提供唯讀工作負載，以獨立向上擴展每個次要叢集。您可以將 Aurora Serverless v2 用於讀取器執行個體，以實現更精細且靈活的擴展。

只有主要叢集會執行寫入操作。執行寫入操作的用戶端會連線至 Aurora 全球資料庫寫入器端點，該端點一律指向主要叢集的寫入器資料庫執行個體。如圖所示，Aurora 會使用叢集儲存磁碟區，而非資料庫引擎進行快速且低額外負載的複寫。如需詳細資訊，請參閱 Amazon Aurora 儲存體的概觀。

Aurora 全球資料庫專為擁有全球足跡的應用程式所設計。多個 中的唯讀次要資料庫叢集 AWS 區域 有助於最佳化更接近應用程式使用者的讀取操作。使用寫入轉送功能，您也可以設定您的全球資料庫，讓次要叢集將寫入請求傳送至主要叢集。如需詳細資訊，請參閱在 Amazon Aurora 全域資料庫中使用寫入轉送。

Aurora 全球資料庫支援兩種變更主要資料庫叢集區域的操作，可視情況使用：Aurora 全球資料庫切換和 Aurora 全球資料庫容錯移轉。

- 對於計劃的操作程序，例如區域輪換，請使用切換機制 (舊稱為「受管計畫容錯移轉」)。使用此功能，您可以將運作狀態良好的 Aurora 全球資料庫的主要叢集，重新放置到其中一個次要區域，而不會遺失資料。如需詳細資訊，請參閱 針對 Amazon Aurora 全球資料庫執行轉換。
- 若要在主要區域發生中斷後復原 Aurora 全球資料庫，請使用容錯移轉機制。透過此功能，您可將主要資料庫叢集執行容錯移轉至另一個區域 (跨區域容錯移轉)。如需詳細資訊，請參閱 針對 Aurora 全球資料庫執行受管容錯移轉。

Amazon Aurora 全球資料庫的優點

Aurora 全球資料庫所提供的優點如下：

- 以本機延遲提供全域讀取 – 如果您在世界各地設有辦公室，可以使用 Aurora 全球資料庫，在主要 AWS 區域中保持主要資訊來源的最新狀態。其他區域的辦公室可以在本地延遲的情況下存取自己區域中的資訊。
- 可擴展次要 Aurora 資料庫叢集 – 您可以將更多唯讀執行個體新增至次要 AWS 區域來擴展次要叢集。次要叢集是唯讀的，因此它最多可支援 16 個唯讀資料庫執行個體，而不是單一 Aurora 叢集通常的 15 個複本限制。
- 從主要資料庫叢集快速複寫到次要 Aurora 資料庫叢集 – Aurora 全球資料庫執行的複寫對主要資料庫叢集的效能影響不大。資料庫執行個體的資源完全投入處理應用程式讀取與寫入工作負載。
- 從全區域中斷中復原 – 次要叢集可讓您建立可用於新的主要 AWS 區域 的 Aurora 全球資料庫，相較於傳統複寫解決方案，這個方法更快 (較低的 RTO)，且資料遺失率較低 (較低的 RPO)。

區域和版本可用性

功能可用性和支援會因每個 Aurora 資料庫引擎的特定版本以及 AWS 區域而有所不同。如需 Aurora 全球資料庫版本和區域可用性的詳細資訊，請參閱 Aurora 全球資料庫的支援區域和資料庫引擎。

Amazon Aurora 全球資料庫的限制

Aurora 全球資料庫目前有下列限制：

- Aurora Global Database 適用於特定 AWS 區域 Aurora MySQL 和 Aurora PostgreSQL 版本。如需詳細資訊，請參閱Aurora 全球資料庫的支援區域和資料庫引擎。
- Aurora 全球資料庫對支援的 Aurora 資料庫執行個體類別、 AWS 區域數目上限等具有特定的組態需求。如需詳細資訊，請參閱Amazon Aurora Global Database 的組態需求。
- 對於與 MySQL 5.7 相容的 Aurora MySQL，Aurora 全球資料庫切換需要 2.09.1 版或更新版本的次要版本。
- 只有在主要和次要資料庫叢集具有相同的主要和次要引擎版本時，您才能在 Aurora 全球資料庫上執行受管跨區域轉換或容錯移轉。根據引擎和引擎版本，修補層級可能需要相同，或者修補層級可能不同。如需允許在具有不同修補層級的主要和次要叢集之間進行這些操作的引擎和引擎版本清單，請參閱 受管跨區域轉換和容錯移轉的修補程式等級相容性。如果您的引擎版本需要相同的修補層級，您可以依照 針對 Aurora 全球資料庫執行手動容錯移轉 中的步驟手動執行容錯移轉。

- Aurora 全球資料庫目前不支援下列 Aurora 功能：
  - Aurora Serverless v1
  - Aurora 中的回溯功能
- 如需使用 RDS 代理功能搭配 Aurora 全球資料庫的限制，請參閱 RDS Proxy 搭配全域資料庫的限制。
- 次要版本自動升級不適用於全球資料庫的 Aurora MySQL 和 Aurora PostgreSQL 叢集。請注意，您可以為全球資料庫叢集的資料庫執行個體指定此設定，但設定沒有任何作用。
- Aurora 全球資料庫目前不支援次要資料庫叢集的 Aurora Auto Scaling。
- 若要在執行 Aurora MySQL 5.7 的 Aurora 全球資料庫上使用資料庫活動串流 (DAS)，引擎版本必須為 2.08 版或更新版本。如需 DAS 的相關資訊，請參閱 使用資料庫活動串流來監控 Amazon Aurora。
- 下列限制目前適用於升級 Aurora 全球資料庫：
  - 在執行 Aurora 全域資料庫的主要版本升級時，無法為全域資料庫叢集套用自訂參數群組。您可於全域叢集的每個區域中建立自訂參數群組，然後在升級後手動將其套用至區域叢集。
  - 使用以 Aurora MySQL 為基礎的 Aurora 全域資料庫時，若啟用 `lower_case_table_names` 參數，即無法從 Aurora MySQL 第 2 版就地升級至第 3 版。如需詳細了解您可以使用的方法，請參閱 主要版本升級。
  - 使用 Aurora 全球資料庫時，如果啟用復原點目標 (RPO) 功能，就無法執行 Aurora PostgreSQL 資料庫引擎的主要版本升級。如需 RPO 功能的相關資訊，請參閱 管理 Aurora PostgreSQL – 全域資料庫的 RPO。
  - 使用 Aurora 全球資料庫時，無法使用標準程序，將 Aurora MySQL 3.01 或 3.02 版執行次要版本升級至 3.03 版或更高版本。如需此程序的詳細資訊，請參閱 透過修改引擎版本升級 Aurora MySQL。

  如需升級 Aurora 全球資料庫相關資訊，請參閱 升級 Amazon Aurora 全域資料庫。
- 您無法單獨停止或啟動全球資料庫中的 Aurora 資料庫叢集。如需詳細資訊，請參閱 停用和啟動 Amazon Aurora 資料庫叢集。
- 在某些情況下，連接到次要 Aurora 資料庫叢集的 Aurora 讀取器資料庫執行個體可以重新啟動。如果主要 AWS 區域的寫入器資料庫執行個體發生重新啟動或容錯移轉，次要區域中的讀取器資料庫執行個體也會重新啟動。然後，次要叢集將無法使用，直到其所有執行個體都與主要資料庫叢集的寫入器執行個體重新同步為止。重新啟動或容錯移轉期間，主要叢集的行為與單一非全球資料庫叢集的行為相同。如需詳細資訊，請參閱以 Amazon Aurora 進行複寫。

  在對主要資料庫叢集進行變更之前，請您務必了解對 全域資料庫的影響。如需詳細資訊，請參閱 從計劃外中斷復原 Amazon Aurora 全域資料庫。

- 當 Amazon Aurora 無法存取資料庫叢集的 AWS KMS 金鑰時，Aurora Global Database 目前不支援 `inaccessible-encryption-credentials-recoverable` 狀態。在這類情況下，加密的資料庫叢集會直接進入終端 `inaccessible-encryption-credentials` 狀態。如需這些狀態的詳細資訊，請參閱 檢視資料庫叢集狀態。
- Secrets Manager 不支援 Aurora 全球資料庫。當您將區域新增至全球資料庫時，您必須先關閉資料庫執行個體的 Secrets Manager 整合。
- 使用 Aurora 全球資料庫的 Aurora PostgreSQL 型資料庫叢集具有以下限制：
  - 屬於 Aurora 全球資料庫一部分的 Aurora PostgreSQL 次要資料庫叢集不支援叢集快取管理。
  - 如果全球資料庫的主要資料庫叢集是以 Amazon RDS PostgreSQL 執行個體的複本為基礎，則無法建立次要叢集。請勿嘗試使用 AWS CLI、 AWS 管理主控台或 `CreateDBCluster` API 操作從該叢集建立次要 。嘗試執行這項操作逾時，無法建立次要叢集。

建議您使用與主要資料庫相同版本的 Aurora 資料庫引擎，建立全球資料庫的次要資料庫叢集。如需詳細資訊，請參閱建立 Amazon Aurora 全域資料庫。

Amazon Aurora 全球資料庫入門

要開始使用 Aurora 全球資料庫，首先決定要使用哪個 Aurora 資料庫引擎，以及在哪個 AWS 區域。只有某些 AWS 區域 的特定版本 Aurora MySQL 和 Aurora PostgreSQL 資料庫引擎支援 Aurora 全球資料庫。如需完整清單，請參閱Aurora 全球資料庫的支援區域和資料庫引擎。

您可以使用下列其中一種方式建立 Aurora 全球資料庫：

- 使用新的 Aurora 資料庫叢集和 Aurora 資料庫執行個體建立新的 Aurora 全球資料庫 – 您可以依照 建立 Amazon Aurora 全域資料庫 中的步驟執行此操作。建立主要 Aurora 資料庫叢集之後，請依照 將 AWS 區域 新增到 Amazon Aurora 全域資料庫 中的步驟新增次要 AWS 區域。
- 使用支援 Aurora 全球資料庫功能的現有 Aurora 資料庫叢集，並向其新增 AWS 區域 – 只有當您現有的 Aurora 資料庫叢集使用支援 Aurora 全域模式的資料庫引擎版本或全域相容時，才能執行此操作。

  選取 Aurora 資料庫叢集後，在 AWS 管理主控台 上確認是否可以針對 Action (動作) 選擇 Add region (新增區域)。如果可以，則可將該 Aurora 資料庫叢集用於您的 Aurora 全域叢集。如需更多詳細資訊，請參閱 將 AWS 區域 新增到 Amazon Aurora 全域資料庫。

建立 Aurora 全球資料庫之前，建議您先了解所有組態需求。

主題

- Amazon Aurora Global Database 的組態需求
- 建立 Amazon Aurora 全域資料庫
- 將 AWS 區域 新增到 Amazon Aurora 全域資料庫
- 在次要區域中建立無周邊 Aurora 資料庫叢集
- 從 Aurora 或 Amazon RDS 快照建立 Amazon Aurora 全球資料庫

Amazon Aurora Global Database 的組態需求

開始使用全球資料庫之前，請先規劃您要使用的叢集和執行個體名稱、AWS 區域、執行個體數量和執行個體類別。請確定您的選擇符合下列組態要求。

Aurora 全域資料庫至少跨越兩個 AWS 區域。主要 AWS 區域 支援具有一個寫入器 Aurora 資料庫執行個體的 Aurora 資料庫叢集。次要 AWS 區域 執行完全由 Aurora 複本組成的唯讀 Aurora 資料庫叢集。至少需要一個次要 AWS 區域，但 Aurora 全球資料庫最多可以有 10 個次要 AWS 區域。此資料表列出了 Aurora 全域資料庫中允許的 Aurora 資料庫叢集、Aurora 資料庫執行個體和 Aurora 複本上限。

描述	主要 AWS 區域	次要 AWS 區域
Aurora 資料庫叢集	1	10 個 (上限)
寫入器執行個體	1	0
每個 Aurora 資料庫叢集的唯讀執行個體 (Aurora 複本)	15 (上限)	16 (總計)
唯讀執行個體 (允許的上限，給定的次要區域實際數量)	15 - s	s = 次要 AWS 區域 的總數

組成 Aurora 全域資料庫的 Aurora 資料庫叢集具有下列特定需求：

- 資料庫執行個體類別需求 – Aurora 全域資料庫需要針對記憶體密集型應用程式最佳化的資料庫執行個體類別。如需有關記憶體最佳化的資料庫執行個體類別的資訊，請參閱 資料庫執行個體類別的類型。建議您使用 db.r5 或更高版本執行個體類別。
- AWS 區域 需求 – Aurora 全域資料庫需要一個 AWS 區域 中的主要 Aurora 資料庫叢集，而在不同區域中至少有一個次要 Aurora 資料庫叢集。您最多可以建立 10 個次要 (唯讀) Aurora 資料庫叢集，而

且每個叢集必須位於不同的區域。換句話說，Aurora 全域資料庫中沒有兩個 Aurora 資料庫叢集可以在同一個 AWS 區域。

如需可與 Aurora 全球資料庫搭配使用之 Aurora 資料庫引擎、引擎版本和 AWS 區域 組合的相關資訊，請參閱 Aurora 全球資料庫的支援區域和資料庫引擎。

- 命名要求 – 您為每個 Aurora 資料庫叢集選擇的名稱在所有 AWS 區域 中必須是唯一的。即使它們位於不同的區域，您也無法為不同的 Aurora 資料庫叢集使用相同的名稱。
- Aurora Serverless v2 的容量需求 - 對於採用 Aurora Serverless v2 的全球資料庫，主要 AWS 區域中針對資料庫叢集建議的最小容量是 8 個 ACU。

您需要 AWS 帳戶，才能遵循本節中的程序。完成使用 Amazon Aurora 的設定任務。如需更多詳細資訊，請參閱 設定您的 Amazon Aurora 環境。您還需要完成其他初步步驟以建立任何 Aurora 資料庫叢集。如需進一步了解，請參閱建立 Amazon Aurora 資料庫叢集。

當您準備好設定全球資料庫時，請參閱 建立 Amazon Aurora 全域資料庫 以取得建立所有必要資源的程序。您也可以遵循 將 AWS 區域 新增到 Amazon Aurora 全域資料庫 中的程序，使用現有的 Aurora 叢集作為主要叢集來建立全球資料庫。

建立 Amazon Aurora 全域資料庫

若要使用 AWS 管理主控台、 AWS CLI或 RDS API 建立 Aurora 全域資料庫及其相關資源，請使用下列步驟。

> **Note**
>
> 如果您有現有的 Aurora 資料庫叢集執行全球相容的 Aurora 資料庫引擎，您可以使用此程序的簡短形式。在這種情況下，您可以將另一個 新增至現有的資料庫叢集 AWS 區域 ，以建立 Aurora 全域資料庫。若要這麼做，請參閱將 AWS 區域 新增到 Amazon Aurora 全域資料庫。

主控台

建立 Aurora 全域資料庫的步驟從登入 AWS 區域 支援 Aurora 全域資料庫功能的 開始。如需完整清單，請參閱 Aurora 全球資料庫的支援區域和資料庫引擎。

以下其中一個步驟是根據您的 Aurora 資料庫叢集的 Amazon VPC 選擇 Virtual Private Cloud (VPC)。若要使用您自己的 VPC，我們建議您事先建立它，以便您進行選擇。同時，建立任何相關子網路，並視需建立子網路群組和安全群組。若要了解作法，請參閱教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4)。

如需建立 Aurora 資料庫叢集的一般資訊，請參閱建立 Amazon Aurora 資料庫叢集。

建立 Aurora 全域資料庫

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 選擇 Create database (建立資料庫)。在 Create database (建立資料庫) 頁面上，執行下列動作：
   - 若為資料庫建立方法，請選擇 Standard Create (標準建立)。(請勿選擇 Easy Create (輕鬆建立)。)
   - 針對引擎選項區段的 `Engine type`，請選擇適用的引擎類型：Aurora (MySQL 相容) 或 Aurora (PostgreSQL 相容)。
3. 使用下列程序中的步驟繼續建立您的 Aurora 全域資料庫。

使用 Aurora MySQL 建立全域資料庫

下列步驟適用於 Aurora MySQL 的所有版本。

使用 Aurora MySQL 建立 Aurora 全域資料庫

完成 Create database (建立資料庫) 頁面。

1. 對於 Engine options (引擎選項)，請選擇下列項目：
   - 在 Engine Version (引擎版本) 中，選擇您要用於 Aurora 全域資料庫的 Aurora MySQL 版本。
2. 在 Templates (範本) 中，選擇 Production (生產)。或者，如果適合您的應用案例，您可以選擇 Dev/Test (開發/測試)。不要在生產環境中使用開發/測試。
3. 在 Settings (設定) 下，執行以下操作：
   a. 為資料庫叢集識別符輸入有意義的名稱。當您完成建立 Aurora 全域資料庫時，此名稱會識別主要資料庫叢集。
   b. 為資料庫執行個體輸入您自己的 `admin` 使用者帳戶密碼，或讓 Aurora 為您產生密碼。如果您選擇自動產生密碼，您會看到複製密碼的選項。



4. 若為 DB instance class (資料庫執行個體類別)，請選擇 `db.r5.large` 或其他記憶體最佳化資料庫執行個體類別。建議您使用 db.r5 或更高版本執行個體類別。

5. 對於 Availability & durability (可用性和耐用性)，我們建議您選擇在不同的可用區域 (AZ) 中讓 Aurora 建立 Aurora 複本。如果您現在沒有建立 Aurora 複本，則需要稍後再執行。



6. 對於 Connectivity (連線)，請根據定義此資料庫執行個體之虛擬聯網環境的 Amazon VPC 選擇 Virtual Private Cloud (VPC)。您可以選擇預設值來簡化此任務。

7. 完成 Database authentication (資料庫驗證) 設定。若要簡化程序，您可以立即選擇 Password authentication (密碼身分驗證)，稍後再設定 AWS Identity and Access Management (IAM)。
8. 對於 Additional configuration (其他設定)，請執行下列動作：

   a. 輸入 Initial database name (初始資料庫名稱) 的名稱，以建立此叢集的主要 Aurora 資料庫執行個體。這是 Aurora 主要資料庫叢集的寫入器節點。

      保留為資料庫叢集參數群組和資料庫參數群組選取的預設值，除非您有自己想要使用的自訂參數群組。

   b. 清除 Enable backtrack (啟用恢復功能) 核取方塊 (如果已選取)。Aurora 全域資料庫不支援恢復。否則，接受 Additional configuration (額外組態) 的其他預設設定。
9. 選擇 Create database (建立資料庫)。

   Aurora 完成建立 Aurora 資料庫執行個體、其 Aurora 複本和 Aurora 資料庫叢集的程序可能需要幾分鐘的時間。您可以透過 Aurora 資料庫叢集的狀態，判斷該資料庫叢集已準備好用作 Aurora 全域資料庫中的主要資料庫叢集。如果是這樣，其狀態以及寫入器和複本節點的狀態為 Available (可用)，如下所示。



當您的主要資料庫叢集可用時，請透過向其新增次要叢集來建立 Aurora 全域資料庫。如要執行此操作，請依照 將 AWS 區域 新增到 Amazon Aurora 全域資料庫 中的步驟進行。

使用 Aurora PostgreSQL 建立全域資料庫

使用 Aurora PostgreSQL 建立 Aurora 全域資料庫

完成 Create database (建立資料庫) 頁面。

1. 對於 Engine options (引擎選項)，請選擇下列項目：

- 對於 Engine Version (引擎版本)，請選擇您要用於 Aurora 全域資料庫的 Aurora PostgreSQL 版本。

2. 在 Templates (範本) 中，選擇 Production (生產)。或者，您可以選擇 Dev/Test (開發/測試)。不要在生產環境中使用開發/測試。
3. 在 Settings (設定) 下，執行以下操作：

   a. 為資料庫叢集識別符輸入有意義的名稱。當您完成建立 Aurora 全域資料庫時，此名稱會識別主要資料庫叢集。

   b. 為資料庫叢集的預設管理員帳戶輸入您自己的密碼，或讓 Aurora 為您產生密碼。如果您選擇 Auto generate a password (自動產生密碼)，您會看到複製密碼的選項。



4. 若為 DB instance class (資料庫執行個體類別)，請選擇 `db.r5.large` 或其他記憶體最佳化資料庫執行個體類別。建議您使用 db.r5 或更高版本執行個體類別。

5. 針對 Availability & durability (可用性和耐用性)，我們建議您選擇在不同的可用區域中讓 Aurora 建立 Aurora 複本。如果您現在沒有建立 Aurora 複本，則需要稍後再執行。
6. 對於 Connectivity (連線)，請根據定義此資料庫執行個體之虛擬聯網環境的 Amazon VPC 選擇 Virtual Private Cloud (VPC)。您可以選擇預設值來簡化此任務。
7. (選擇性) 完成 Database authentication (資料庫身分驗證) 設定。密碼驗證會一直啟用。若要簡化程序，您可以跳過此部分，稍後再設定 IAM 或密碼和 Kerberos 身分驗證。
8. 對於 Additional configuration (其他設定)，請執行下列動作：

   a. 輸入 Initial database name (初始資料庫名稱) 的名稱，以建立此叢集的主要 Aurora 資料庫執行個體。這是 Aurora 主要資料庫叢集的寫入器節點。

      保留為資料庫叢集參數群組和資料庫參數群組選取的預設值，除非您有自己想要使用的自訂參數群組。

   b. 接受 Additional configuration (其他組態) 的所有其他預設設定，例如加密、日誌匯出等。
9. 選擇 Create database (建立資料庫)。

   Aurora 完成建立 Aurora 資料庫執行個體、其 Aurora 複本和 Aurora 資料庫叢集的程序可能需要幾分鐘的時間。當叢集可供使用時，Aurora 資料庫叢集及其寫入器和複本節點都會顯示 Available (可用) 狀態。新增次要資料庫叢集之後，這將成為 Aurora 全域資料庫的主要資料庫叢集。



當主要資料庫叢集可供使用時，請依照將 AWS 區域 新增到 Amazon Aurora 全域資料庫中的步驟來建立一或多個次要叢集。

AWS CLI

下列程序中的 AWS CLI 命令會完成下列任務：

1. 建立 Aurora 全域資料庫，命名並指定您計劃使用的 Aurora 資料庫引擎類型。
2. 為 Aurora 全域資料庫建立 Aurora 資料庫叢集。
3. 為叢集建立 Aurora 資料庫執行個體。這是全域資料庫的主要 Aurora 資料庫叢集。
4. 為 Aurora 資料庫叢集建立第二個資料庫執行個體。這是完成 Aurora 資料庫叢集的讀取器。
5. 在另一個區域中建立第二個 Aurora 資料庫叢集，然後依照將 AWS 區域 新增到 Amazon Aurora 全域資料庫中的步驟將其新增至 Aurora 全域資料庫。

遵循您的 Aurora 資料庫引擎的程序。

使用 Aurora MySQL 建立全域資料庫

使用 Aurora MySQL 建立 Aurora 全域資料庫

1. 使用 `create-global-cluster` CLI 命令，傳遞 AWS 區域的名稱、Aurora 資料庫引擎和版本。

   針對 Linux、macOS 或 Unix：

   ```
   aws rds create-global-cluster --region primary_region \
       --global-cluster-identifier global_database_id \
       --engine aurora-mysql \
       --engine-version version # optional
   ```

   在 Windows 中：

   ```
   aws rds create-global-cluster ^
       --global-cluster-identifier global_database_id ^
       --engine aurora-mysql ^
       --engine-version version # optional
   ```

   這將建立一個「空」的 Aurora 全域資料庫，只有一個名稱 (識別符) 和 Aurora 資料庫引擎。可能需要幾分鐘的時間才能使用 Aurora 全域資料庫。在進行下一個步驟之前，請使用 `describe-global-clusters` CLI 命令來查看它是否可用。

```
aws rds describe-global-clusters --region primary_region --global-cluster-
identifier global_database_id
```

當 Aurora 全域資料庫可用時，您可以建立其主要 Aurora 資料庫叢集。

2. 若要建立主要 Aurora 資料庫叢集，請使用 create-db-cluster CLI 命令。使用 --global-cluster-identifier 參數來包含 Aurora 全球資料庫的名稱。

   對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
  --region primary_region \
  --db-cluster-identifier primary_db_cluster_id \
  --master-username userid \
  --master-user-password password \
  --engine aurora-mysql \
  --engine-version version \
  --global-cluster-identifier global_database_id
```

   在 Windows 中：

```
aws rds create-db-cluster ^
  --region primary_region ^
  --db-cluster-identifier primary_db_cluster_id ^
  --master-username userid ^
  --master-user-password password ^
  --engine aurora-mysql ^
  --engine-version version ^
  --global-cluster-identifier global_database_id
```

   使用 describe-db-clusters AWS CLI 命令來確認 Aurora 資料庫叢集已就緒。要單出一個特定的 Aurora 資料庫叢集，請使用 --db-cluster-identifier 參數。或者，您可以在命令中省略 Aurora 資料庫叢集名稱，以獲取有關給定區域中所有 Aurora 資料庫叢集的詳細資訊。

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier primary_db_cluster_id
```

   當叢集的回應顯示 "Status": "available" 時，便可以使用。

3. 為主要 Aurora 資料庫叢集建立資料庫執行個體。若要執行這項操作，請使用 `create-db-instance` CLI 命令。為您的 Aurora 資料庫叢集命令指定名稱，並指定執行個體的組態詳細資訊。您不需要在命令中傳遞 `--master-username` 和 `--master-user-password` 參數，因為它會從 Aurora 資料庫叢集獲取這些參數。

   若為 `--db-instance-class`，您只能使用那些來自記憶體最佳化類別，例如 `db.r5.large`。建議您使用 db.r5 或更高版本執行個體類別。如需這些類別的相關資訊，請參閱 資料庫執行個體類別的類型。

   對於 Linux、macOS 或 Unix：

   ```
   aws rds create-db-instance \
     --db-cluster-identifier primary_db_cluster_id \
     --db-instance-class instance_class \
     --db-instance-identifier db_instance_id \
     --engine aurora-mysql \
     --engine-version version \
     --region primary_region
   ```

   在 Windows 中：

   ```
   aws rds create-db-instance ^
     --db-cluster-identifier primary_db_cluster_id ^
     --db-instance-class instance_class ^
     --db-instance-identifier db_instance_id ^
     --engine aurora-mysql ^
     --engine-version version ^
     --region primary_region
   ```

   `create-db-instance` 作業可能需要一些時間才能完成。在繼續之前，請檢查狀態以查看 Aurora 資料庫執行個體是否可用。

   ```
   aws rds describe-db-clusters --db-cluster-identifier primary_db_cluster_id
   ```

   當命令傳回 `available` 的狀態時，您可以為主要資料庫叢集建立另一個 Aurora 資料庫執行個體。這是 Aurora 資料庫叢集的讀取器執行個體 (Aurora 複本)。

4. 若要為叢集建立另一個 Aurora 資料庫執行個體，請使用 `create-db-instance` CLI 命令。

   對於 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier replica_db_instance_id \
  --engine aurora-mysql
```

在 Windows 中：

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier replica_db_instance_id ^
  --engine aurora-mysql
```

當資料庫執行個體可用時，會從寫入器節點開始複寫到複本。繼續之前，請檢查資料庫執行個體是否可以使用 describe-db-instances CLI 命令。

此時，您有一個 Aurora 全域資料庫，其主要 Aurora 資料庫叢集包含寫入器資料庫執行個體和 Aurora 複本。您現在可以在不同區域中新增唯讀 Aurora 資料庫叢集，以完成 Aurora 全域資料庫。若要啟用，請依照「將 AWS 區域 新增到 Amazon Aurora 全域資料庫」中的步驟進行。

使用 Aurora PostgreSQL 建立全域資料庫

當您使用下列命令建立 Aurora 全域資料庫的 Aurora 物件時，每個命令可能需要幾分鐘的時間才能使用。我們建議您在完成任何命令之後，檢查特定 Aurora 物件的狀態，以確定狀態可用。

若要執行這項操作，請使用 describe-global-clusters CLI 命令。

```
aws rds describe-global-clusters --region primary_region
    --global-cluster-identifier global_database_id
```

使用 Aurora PostgreSQL建立 Aurora 全域資料庫

1. 使用 create-global-cluster CLI 命令。

   對於 Linux、macOS 或 Unix：

   ```
   aws rds create-global-cluster --region primary_region \
       --global-cluster-identifier global_database_id \
   ```

```
    --engine aurora-postgresql \
    --engine-version version # optional
```

在 Windows 中：

```
aws rds create-global-cluster ^
    --global-cluster-identifier global_database_id ^
    --engine aurora-postgresql ^
    --engine-version version # optional
```

當 Aurora 全域資料庫可用時，您可以建立其主要 Aurora 資料庫叢集。

2. 若要建立主要 Aurora 資料庫叢集，請使用 create-db-cluster CLI 命令。使用 `--global-cluster-identifier` 參數來包含 Aurora 全球資料庫的名稱。

   對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
  --region primary_region \
  --db-cluster-identifier primary_db_cluster_id \
  --master-username userid \
  --master-user-password password \
  --engine aurora-postgresql \
  --engine-version version \
  --global-cluster-identifier global_database_id
```

   在 Windows 中：

```
aws rds create-db-cluster ^
  --region primary_region ^
  --db-cluster-identifier primary_db_cluster_id ^
  --master-username userid ^
  --master-user-password password ^
  --engine aurora-postgresql ^
  --engine-version version ^
  --global-cluster-identifier global_database_id
```

   檢查 Aurora 資料庫叢集是否已準備就緒。當下列命令的回應為 Aurora 資料庫叢集顯示 `"Status": "available"` 時，您可以繼續執行。

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier primary_db_cluster_id
```

3. 為主要 Aurora 資料庫叢集建立資料庫執行個體。若要執行這項操作，請使用 create-db-instance CLI 命令。

   使用 --db-cluster-identifier 參數傳遞 Aurora 資料庫叢集的名稱。

   您不需要在命令中傳遞 --master-username 和 --master-user-password 參數，因為它會從 Aurora 資料庫叢集獲取這些參數。

   若為 --db-instance-class，您只能使用那些來自記憶體最佳化類別，例如 db.r5.large。建議您使用 db.r5 或更高版本執行個體類別。如需這些類別的相關資訊，請參閱 資料庫執行個體類別的類型。

   對於 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier db_instance_id \
  --engine aurora-postgresql \
  --engine-version version \
  --region primary_region
```

   在 Windows 中：

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier db_instance_id ^
  --engine aurora-postgresql ^
  --engine-version version ^
  --region primary_region
```

4. 請先檢查 Aurora 資料庫執行個體的狀態，然後再繼續。

```
aws rds describe-db-clusters --db-cluster-identifier primary_db_cluster_id
```

如果回應顯示 Aurora 資料庫執行個體狀態為 `available`，您可以為主要資料庫叢集建立另一個 Aurora 資料庫執行個體。

5. 若要建立 Aurora 資料庫叢集的 Aurora 複本，請使用 `create-db-instance` CLI 命令。

   對於 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier replica_db_instance_id \
  --engine aurora-postgresql
```

   在 Windows 中：

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier replica_db_instance_id ^
  --engine aurora-postgresql
```

當資料庫執行個體可用時，會從寫入器節點開始複寫到複本。繼續之前，請檢查資料庫執行個體是否可以使用 `describe-db-instances` CLI 命令。

您的 Aurora 全域資料庫存在，但它只有其主要區域，其中包含由寫入器資料庫執行個體和 Aurora 複本所組成的 Aurora 資料庫叢集。您現在可以在不同區域中新增唯讀 Aurora 資料庫叢集，以完成 Aurora 全域資料庫。若要啟用，請依照「將 AWS 區域 新增到 Amazon Aurora 全域資料庫」中的步驟進行。

RDS API

若要以 RDS API 建立 Aurora 全球資料庫，請執行 CreateGlobalCluster 操作。

將 AWS 區域 新增到 Amazon Aurora 全域資料庫

您可以使用下列程序，將額外的次要叢集新增至現有的全域資料庫。您也可以使用此程序新增第一個次要 AWS 區域，從獨立 Aurora 資料庫叢集建立全域資料庫。

Aurora 全域資料庫至少需要一個與主要 Aurora 資料庫叢集在不同 AWS 區域 中的次要 Aurora 資料庫叢集。您最多可以將 10 個次要資料庫叢集連線至 Aurora 全域資料庫。為每個新的次要資料庫叢集重

複下列程序。對於新增至 Aurora 全域資料庫的每個次要資料庫叢集，請將允許給主要資料庫叢集的 Aurora 複本數目減少一個。

例如，如果您的 Aurora 全域資料庫有 10 個次要區域，您的主要資料庫叢集只能有 5 個 (而不是 15 個) Aurora 複本。如需更多詳細資訊，請參閱 Amazon Aurora Global Database 的組態需求。

主要資料庫叢集中的 Aurora 複本 (讀取器執行個體) 數量決定了您可以新增的次要資料庫叢集數量。主要資料庫叢集加上次要叢集中的讀取器執行個體總數不能超過 15 個。例如，如果主要資料庫叢集和 1 個次要叢集中的讀取器執行個體總數合計為 14，就無法將另一個次要叢集新增至全域資料庫。

> **Note**
>
> 對於 Aurora MySQL 第 3 版，請在建立次要叢集時確保 `lower_case_table_names` 的值與主要叢集中的值相符。此設定是資料庫參數，會影響伺服器處理識別符區分大小寫的方式。如需這資料庫參數的相關資訊，請參閱 Amazon Aurora 的參數群組。
> 建議您在建立次要叢集時，對主要和次要叢集使用相同的資料庫引擎版本。如有必要，請將主要叢集升級至與次要叢集相同的版本。如需更多詳細資訊，請參閱 受管跨區域轉換和容錯移轉的修補程式等級相容性。

主控台

將 AWS 區域 新增至 Aurora 全域資料庫

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在 AWS 管理主控台 導覽窗格中，選擇 Databases (資料庫)。
3. 選擇需要次要 Aurora 資料庫叢集的 Aurora 全域資料庫。確定主要 Aurora 資料庫叢集是 `Available`。
4. 對於動作，請選擇新增 AWS 區域。



5. 在 Add a region (新增區域) 頁面，選擇次要 AWS 區域。

   您無法針對相同的 Aurora 全域資料庫選擇已有次要 Aurora 資料庫叢集的 AWS 區域。此外，不能是與主要 Aurora 資料庫叢集相同的區域。

> **Note**
>
> 只有在控制 Babelfish 偏好設定的參數在這些區域中開啟時，Babelfish for Aurora PostgreSQL 全域資料庫才能在次要區域中運作。如需詳細資訊，請參閱Babelfish 的資料庫叢集參數群組設定



6. 對於新的 AWS 區域中的次要 Aurora 叢集，填寫剩餘的欄位。這些組態選項與任何 Aurora 資料庫叢集執行個體的組態選項相同，除了下列僅適用於 Aurora MySQL– Aurora 全域資料庫的選項：

   - 啟用僅供讀取複本寫入轉送 – 此選用設定可讓您的 Aurora 全域資料庫的次要資料庫叢集將寫入操作轉送至主要叢集。如需更多詳細資訊，請參閱 在 Amazon Aurora 全域資料庫中使用寫入轉送。

     

7. 選擇新增 AWS 區域。

完成將區域新增到 Aurora 全域資料庫後，您可以在 AWS 管理主控台 的 Databases (資料庫) 清單中看到它，如圖所示的螢幕擷取畫面。



AWS CLI

將次要 AWS 區域 新增至 Aurora 全域資料庫

若要使用 CLI 將次要叢集新增至全域資料庫，您必須已有全域叢集容器物件。如果您尚未執行 `create-global-cluster` 命令，請參閱 建立 Amazon Aurora 全域資料庫 中的 CLI 程序。

1. 將 `create-db-cluster` CLI 命令與 Aurora 全域資料庫的名稱 (`--global-cluster-identifier`) 搭配使用。對於其他參數，請執行下列動作：
2. 對於 `--region`，請選擇與 Aurora 主要區域不同的 AWS 區域。
3. 請選擇 `--engine` 和 `--engine-version` 參數的特定值。這些值與 Aurora 全域資料庫中主要 Aurora 資料庫叢集的值相同。
4. 對於加密叢集，請指定主要 AWS 區域 作為要加密的 `--source-region`。

下面的範例建立了一個新的 Aurora 資料庫叢集，並將其連線到 Aurora 全域資料庫做為唯讀次要 Aurora 資料庫叢集。在最後一個步驟中，Aurora 資料庫執行個體會新增至新的 Aurora 資料庫叢集。

對於 Linux、macOS 或 Unix：

```
aws rds --region secondary_region \
  create-db-cluster \
    --db-cluster-identifier secondary_cluster_id \
    --global-cluster-identifier global_database_id \
```

```
    --engine aurora-mysql | aurora-postgresql \
    --engine-version version

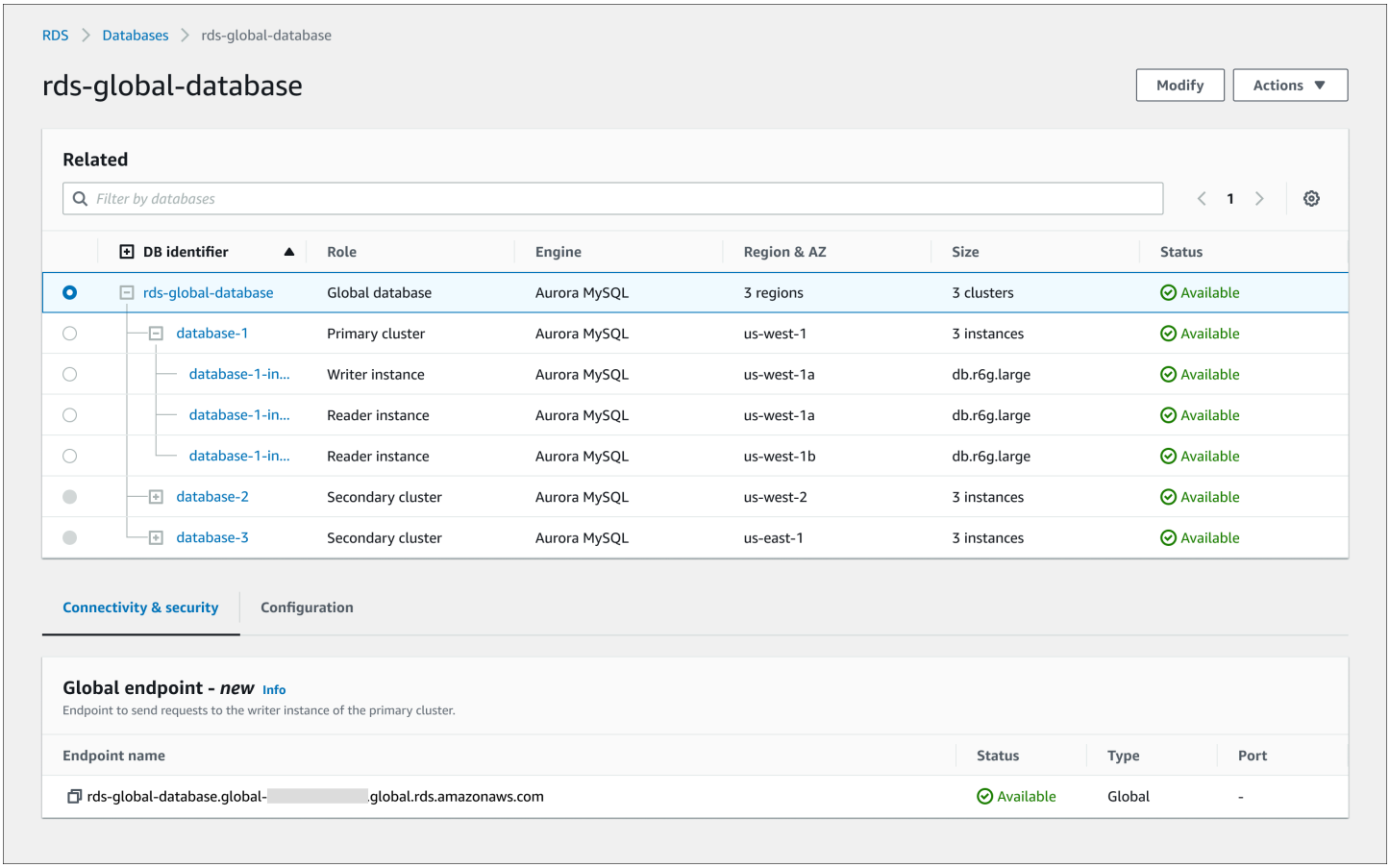
aws rds --region secondary_region \
  create-db-instance \
    --db-instance-class instance_class \
    --db-cluster-identifier secondary_cluster_id \
    --db-instance-identifier db_instance_id \
    --engine aurora-mysql | aurora-postgresql
```

在 Windows 中：

```
aws rds --region secondary_region ^
  create-db-cluster ^
    --db-cluster-identifier secondary_cluster_id ^
    --global-cluster-identifier global_database_id_id ^
    --engine aurora-mysql | aurora-postgresql ^
    --engine-version version

aws rds --region secondary_region ^
  create-db-instance ^
    --db-instance-class instance_class ^
    --db-cluster-identifier secondary_cluster_id ^
    --db-instance-identifier db_instance_id ^
    --engine aurora-mysql | aurora-postgresql
```

RDS API

若要用 RDS API 將新的 AWS 區域 新增到 Aurora 全域資料庫，請執行 CreateDBCluster 操作。使用 `GlobalClusterIdentifier` 參數來指定現有全域資料庫的識別符。

在次要區域中建立無周邊 Aurora 資料庫叢集

雖然 Aurora 全域資料庫在與主要資料庫不同的 AWS 區域 中至少需要一個次要 Aurora 資料庫叢集，但是您可以針對次要叢集使用無周邊設定。無周邊次要 Aurora 資料庫叢集是沒有資料庫執行個體的叢集。這種類型的組態可以降低 Aurora 全域資料庫的費用。在 Aurora 資料庫叢集中，運算和儲存體會解偶。若沒有資料庫執行個體，則無需支付運算費用，只需支付儲存費用。如果設定正確，無周邊次要資料庫的儲存磁碟區會與主要 Aurora 資料庫叢集保持同步。

您可以在建立 Aurora 全域資料庫時，照常新增次要叢集。如果您要建立全域資料庫中的所有叢集，請遵循 建立 Amazon Aurora 全域資料庫 中的程序。如果您已有資料庫叢集可用作主要叢集，請遵循 將 AWS 區域 新增到 Amazon Aurora 全域資料庫 中的程序。

主要 Aurora 資料庫叢集開始複寫到次要資料庫叢集之後，您會從次要 Aurora 資料庫叢集刪除 Aurora 唯讀資料庫執行個體。此次要叢集現在被視為「無周邊」，因為它不再具有資料庫執行個體。即使次要叢集中沒有任何資料庫執行個體，Aurora 仍會讓儲存磁碟區與主要 Aurora 資料庫叢集保持同步。

> **Warning**
>
> 有了 Aurora PostgreSQL，若要在次要 AWS 區域 中建立無周邊叢集，可使用 AWS CLI 或 RDS API 來新增次要 AWS 區域。跳過為次要叢集建立讀取器資料庫執行個體的步驟。目前，RDS 主控台中不支援建立無周邊叢集。如需要使用的 CLI 和 API 程序，請參閱將 AWS 區域 新增到 Amazon Aurora 全域資料庫。
>
> 如果您的全球資料庫使用的 Aurora PostgreSQL 引擎版本低於 13.4、12.8 版或 11.13 版，則在次要區域中建立讀取器資料庫執行個體並隨後刪除該執行個體時，可能會導致主要區域的寫入器資料庫執行個體發生 Aurora PostgreSQL 真空問題。如果您遇到這個問題，請在刪除次要區域的讀取器資料庫執行個體之後，重新啟動主要區域的寫入器資料庫執行個體。

將無周邊次要 Aurora 資料庫叢集新增至 Aurora 全域資料庫

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在 AWS 管理主控台 導覽窗格中，選擇 Databases (資料庫)。
3. 選擇需要次要 Aurora 資料庫叢集的 Aurora 全域資料庫。確定主要 Aurora 資料庫叢集是 `Available`。
4. 對於動作，請選擇新增 AWS 區域。
5. 在 Add a region (新增區域) 頁面，選擇次要 AWS 區域。

   您無法針對相同的 Aurora 全域資料庫選擇已有次要 Aurora 資料庫叢集的 AWS 區域。此外，不能是與主要 Aurora 資料庫叢集相同的區域。
6. 對於新的 AWS 區域 區域中的次要 Aurora 叢集，填寫剩餘的欄位。這些組態選項與任何 Aurora 資料庫叢集執行個體的組態選項相同。

   對於 Aurora MySQL– Aurora 全域資料庫，請忽略 Enable read replica write forwarding (啟用僅供讀取複本寫入轉送) 選項。刪除讀取器執行個體之後，此選項將沒有任何功能。
7. 選擇新增 AWS 區域。完成將區域新增到 Aurora 全域資料庫後，您可以在 AWS 管理主控台 的 Databases (資料庫) 清單中看到它，如圖所示的螢幕擷取畫面。



8. 在繼續之前，請先使用 AWS 管理主控台 或 AWS CLI 檢查次要 Aurora 資料庫叢集及其讀取器執行個體的狀態。例如：

```
$ aws rds describe-db-clusters --db-cluster-identifier secondary-cluster-id --query
 '*[].[Status]' --output text
```

新增的次要 Aurora 資料庫叢集狀態可能需要幾分鐘時間，才能從「creating」變更為「available」。當 Aurora 資料庫叢集可用時，您可以刪除讀取器執行個體。

9. 在次要 Aurora 資料庫叢集中選取讀取器執行個體，然後選擇 Delete (刪除)。



刪除讀取器執行個體之後，次要叢集仍然是 Aurora 全域資料庫的一部分。其沒有與之關聯的執行個體，如下所示。



如果發生此類中斷，您可以使用此無周邊次要 Aurora 資料庫叢集，[從主要 AWS 區域 的非計劃中斷手動復原您的 Amazon Aurora 全域資料庫](#)。

從 Aurora 或 Amazon RDS 快照建立 Amazon Aurora 全球資料庫

您可以還原 Aurora 資料庫叢集的快照或從 Amazon RDS 資料庫執行個體還原，以用做 Aurora 全域資料庫的起點。您可以還原快照並同時建立新 Aurora 佈建的資料庫叢集。然後將另一個 AWS 區域 新增至還原的資料庫叢集，從而將其轉換為 Aurora 全域資料庫。您以這種方式使用快照建立的任何 Aurora 資料庫叢集都會成為 Aurora 全域資料庫的主要叢集。

您使用的快照可以來自 `provisioned` 或來自 `serverless` Aurora 資料庫叢集。

在還原程序期間，請選擇與快照相同的資料庫引擎類型。例如，假設您想要從執行 Aurora PostgreSQL 的 Aurora Serverless 資料庫叢集中還原建立的快照。在此情況下，您可以使用相同的 Aurora PostgreSQL 資料庫引擎和版本，來建立 Aurora 資料庫叢集。

當您將 AWS 區域 新增至其中時，還原的資料庫叢集會擔任 Aurora 全域資料庫主要叢集的角色。此主要叢集中包含的所有資料都會複寫到您新增至 Aurora 全域資料庫的任何次要叢集。



管理 Amazon Aurora 全域資料庫

您會在構成 Aurora 全域資料庫的個別叢集上執行大部分的管理操作。當您在主控台的 Databases (資料庫) 頁面上選擇 Group related resources (組合相關資源) 時，您會看到主要叢集和次要叢集在相關聯的全球資料庫下分組。若要尋找正在執行全域資料庫之資料庫叢集的 AWS 區域、其 Aurora 資料庫引擎和版本，以及其識別符，請使用其 Configuration (組態) 標籤。

跨區域資料庫容錯移轉程序僅適用於 Aurora 全球資料庫，不適用於單一 Aurora 資料庫叢集。如需詳細資訊，請參閱 在 Amazon Aurora 全球資料庫中使用切換或容錯移轉。

若要從其主要區域的計劃外中斷復原 Aurora 全域資料庫，請參閱從計劃外中斷復原 Amazon Aurora 全域資料庫。

主題

- 修改 Amazon Aurora 全域資料庫

- 修改 Aurora 全域資料庫的參數
- 從 Amazon Aurora 全域資料庫中移除叢集
- 刪除 Amazon Aurora 全域資料庫
- 標記 Amazon Aurora 全球資料庫資源

修改 Amazon Aurora 全域資料庫

中資料庫頁面會 AWS 管理主控台 列出所有 Aurora 全域資料庫，顯示每個資料庫的主要叢集和次要叢集。Aurora 全域資料庫擁有其自身的組態設定。具體而言，它與其主要和次要叢集 AWS 區域 相關聯，如以下螢幕擷取畫面所示。



當您對 Aurora 全域資料庫進行變更時，您有機會取消更改，如下面的螢幕擷取畫面。



當您選擇 Continue (繼續) 時，您可以確認變更。

修改 Aurora 全域資料庫的參數

您可以為 Aurora 全域資料庫內的每個 Aurora 叢集，單獨地設定 Aurora 資料庫叢集參數群組。大部分參數的作用就如同用在其他類型的 Aurora 叢集一樣。建議您將全域資料庫中所有叢集的設定保持一致。如果您將次要叢集提升為主要叢集，這麼做有助於避免非預期的行為變更。

例如，讓時區和字元集使用相同設定，以避免不同叢集接管成為主要業集時發生不一致的行為。

`aurora_enable_repl_bin_log_filtering` 和 `aurora_enable_replica_log_compression` 組態設定沒有效果。

從 Amazon Aurora 全域資料庫中移除叢集

您可以根據幾個不同的原因，從 Aurora 全域資料庫中移除 Aurora 資料庫叢集。例如，如果主要叢集降級或隔離，您可能想要從 Aurora 全域資料庫移除 Aurora 資料庫叢集。接著它會變為獨立佈建的 Aurora 資料庫叢集，可用於建立新的 Aurora 全域資料庫。如需進一步了解，請參閱[從計劃外中斷復原 Amazon Aurora 全域資料庫]。

您也可能想要移除 Aurora 資料庫叢集，因為您想要刪除不再需要的 Aurora 全域資料庫。您無法刪除 Aurora 全域資料庫，直至刪除 (分離) 所有關聯的 Aurora 資料庫叢集之後，將主要資料庫叢集設定為最後刪除。如需更多詳細資訊，請參閱 刪除 Amazon Aurora 全域資料庫。

當 Aurora 資料庫叢集從 Aurora 全域資料庫分離時，它不再與主要資料庫叢集保持同步。它會成為具有完整讀取/寫入功能的獨立佈建 Aurora 資料庫叢集。

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 從 Aurora 全域資料庫移除 Aurora 資料庫叢集。

主控台

從 Aurora 全球資料庫中移除 Aurora 叢集

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在 Databases (資料庫) 頁面上選擇叢集。
3. 對於 Actions (動作)，選擇 Remove from Global (從全球移除)。



您會看到一個提示，以確認您要從 Aurora 全域資料庫卸離次要叢集。



4. 選擇 Remove and promote (移除並升級)，即可從全域資料庫移除叢集。

Aurora 資料庫叢集不再用作 Aurora 全域資料庫中的次要叢集，且不再與主要資料庫叢集同步。它是一個獨立的 Aurora 資料庫叢集，具有完整的讀取/寫入功能。



移除或刪除所有次要叢集之後，您就可以用同樣的方式移除主要叢集。在移除所有次要叢集之前，您無法從 Aurora 全域資料庫中卸離 (移除) 主要 Aurora 資料庫叢集。

Aurora 全域資料庫可能會保留在 Databases (資料庫) 清單中，其中包含零個區域和可用區域。如果您不想再使用此 Aurora 全域資料庫，您可以刪除。如需更多詳細資訊，請參閱 刪除 Amazon Aurora 全域資料庫。

AWS CLI

若要從 Aurora 全域資料庫移除 Aurora 叢集，請使用下列參數執行 remove-from-global-cluster CLI 命令：

- `--global-cluster-identifier` – Aurora 全域資料庫的名稱 (識別符)。
- `--db-cluster-identifier` – 要從 Aurora 全域資料庫中移除的每個 Aurora 資料庫叢集的名稱。移除主要資料庫叢集之前，請先移除所有次要 Aurora 資料庫叢集

下列範例從 Aurora 全域資料庫中先移除次要叢集，然後移除主要叢集。

對於 Linux、macOS 或 Unix：

```
aws rds --region secondary_region \
  remove-from-global-cluster \
    --db-cluster-identifier secondary_cluster_ARN \
    --global-cluster-identifier global_database_id

aws rds --region primary_region \
  remove-from-global-cluster \
    --db-cluster-identifier primary_cluster_ARN \
    --global-cluster-identifier global_database_id
```

針對 Aurora 全域資料庫中的每個次要 AWS 區域，重複此 `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN` 命令。

在 Windows 中：

```
aws rds --region secondary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier secondary_cluster_ARN ^
    --global-cluster-identifier global_database_id

aws rds --region primary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier primary_cluster_ARN ^
    --global-cluster-identifier global_database_id
```

針對 Aurora 全域資料庫中的每個次要 AWS 區域，重複此 `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN` 命令。

RDS API

若要使用 RDS API 從 Aurora 全球資料庫中移除 Aurora 叢集，請執行 RemoveFromGlobalCluster 動作。

刪除 Amazon Aurora 全域資料庫

因為 Aurora 全域資料庫通常包含關鍵業務資料，您不能在單一步驟中刪除全域資料庫和相關聯的叢集。若要刪除 Aurora 全域資料庫，請執行下列動作：

- 從 Aurora 全域資料庫移除所有次要資料庫叢集。每個叢集都會變成獨立 Aurora 資料庫叢集。若要瞭解如何操作，請參閱從 Amazon Aurora 全域資料庫中移除叢集。

- 從每個獨立 Aurora 資料庫叢集中，刪除所有 Aurora 複本。
- 從 Aurora 全域資料庫移除主要資料庫叢集。這會成為獨立 Aurora 資料庫叢集。
- 從 Aurora 主要資料庫叢集中，先刪除所有 Aurora 複本，然後刪除寫入器資料庫執行個體。

從新獨立的 Aurora 資料庫叢集刪除寫入器執行個體，通常也會移除 Aurora 資料庫叢集和 Aurora 全域資料庫。

如需一般詳細資訊，請參閱從 Aurora 個體資料庫叢集刪除資料庫執行個體。

若要刪除 Aurora 全域資料庫，您可以使用 AWS 管理主控台、AWS CLI 或 RDS API。

主控台

刪除 Aurora 全域資料庫

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 選擇 Databases (資料庫)，然後在清單中尋找您要刪除的 Aurora 全域資料庫。
3. 請確認已從 Aurora 全域資料庫中移除所有叢集。Aurora 全域資料庫應顯示 0 個區域和可用區域，以及大小為 0 的叢集。

   如果 Aurora 全域資料庫包含任何 Aurora 資料庫叢集，則無法將其刪除。如有必要，請從 Aurora 全域資料庫中卸離主要和次要 Aurora 資料庫叢集。如需更多詳細資訊，請參閱 從 Amazon Aurora 全域資料庫中移除叢集。
4. 在清單中選擇您的 Aurora 全球資料庫，然後從動作功能表中選擇刪除。



AWS CLI

若要刪除 Aurora 全域資料庫，請使用 AWS 區域 名稱和 Aurora 全域資料庫識別符執行 delete-global-cluster CLI 命令，如下列範例所示。

對於 Linux、macOS 或 Unix：

```
aws rds --region primary_region delete-global-cluster \
   --global-cluster-identifier global_database_id
```

在 Windows 中：

```
aws rds --region primary_region delete-global-cluster ^
   --global-cluster-identifier global_database_id
```

RDS API

若要刪除屬於 Aurora 全域資料庫的叢集，請執行 DeleteGlobalCluster API 操作。

標記 Amazon Aurora 全球資料庫資源

使用 Aurora 全球資料庫功能，您可以將 RDS 標籤套用至全球資料庫中不同層級的資源。如果您不熟悉如何將標籤與 AWS 或 Aurora 資源搭配使用，請在全球資料庫中套用標籤之前參閱 標記 Amazon Aurora 和 Amazon RDS 資源。

> **Note**
>
> 由於 AWS 會在成本報告機制中處理標籤資料，因此請勿在標籤名稱或值中包含任何敏感資料或個人身分識別資訊 (PII)。

您可以將標籤套用至全球資料庫中的下列資源類型：

- 整個全球資料庫的容器物件。此資源稱為全域叢集。

  在主控台中執行新增 AWS 區域操作來建立全域叢集之後，您可以使用全域叢集的詳細資訊頁面來新增標籤。在全域叢集詳細資訊頁面的標籤索引標籤上，您可以選擇管理標籤來新增、移除或修改標籤及其相關聯的值。

  使用 AWS CLI 和 RDS API，您可以同時將標籤新增至全域叢集。您也可以新增、移除或修改現有全域叢集的標籤。

- 主要叢集。您可以在此處使用與獨立 Aurora 叢集相同的標籤程序。您可以在將原始 Aurora 叢集切換為全球資料庫之前設定標籤。您也可以在資料庫叢集詳細資訊頁面的標籤索引標籤上選擇管理標籤，以新增、移除或修改標籤及其相關聯的值。
- 任何次要叢集。您可以在此處使用與獨立 Aurora 叢集相同的標籤程序。您可以使用主控台中的新增 AWS 區域動作，在建立次要 Aurora 叢集的同時設定標籤。您也可以在資料庫叢集詳細資訊頁面的標籤索引標籤上選擇管理標籤，以新增、移除或修改標籤及其相關聯的值。
- 主要或次要叢集中的個別資料庫執行個體。您可以在此處使用與 Aurora 或 RDS 資料庫執行個體相同的標籤程序。您可以使用主控台中的新增讀取器動作，在將新的資料庫執行個體新增至 Aurora 叢集的同時設定標籤。您也可以在資料庫執行個體詳細資訊頁面的標籤索引標籤上選擇管理標籤，以新增、移除或修改標籤及其相關聯的值。

以下是您可能在全球資料庫中指派之標籤類型的一些範例：

- 您可以將標籤新增至全域叢集，以記錄應用程式的整體資訊，例如代表組織中擁有者和聯絡人的匿名識別符。您可以使用標籤來表示使用全球資料庫的應用程式屬性。
- 您可以將標籤新增至主要叢集和次要叢集，以在 AWS 區域層級追蹤應用程式的成本。如需該程序的詳細資訊，請參閱 AWS 帳單如何在 Amazon RDS 中使用標籤。
- 您可以使用 Aurora 叢集將標籤新增至特定資料庫執行個體，以指出其特殊用途。例如，在主要叢集中，您可能有一個具有低容錯移轉優先順序的讀取器執行個體，專門用於產生報告。標籤可以區分此特殊用途資料庫執行個體與主要叢集中專用於高可用性的其他執行個體。
- 您可以在全球資料庫資源的所有層級使用標籤，以透過 IAM 政策控制存取。如需詳細資訊，請參閱《AWS Identity and Access Management 使用者指南》中的控制 AWS 資源的存取權。

> **Tip**
>
> 在 AWS 管理主控台 中，您可以在建立全域叢集容器之後，作為個別步驟將標籤新增至全域叢集容器。如果您想要避免全域叢集存在而沒有存取控制標籤的任何時間間隔，您可以在 `CreateGlobalCluster` 操作期間透過 AWS CLI、RDS API 或 CloudFormation 範本建立該資源來套用標籤。

- 您可以在叢集層級或全域叢集使用標籤，記錄應用程式品質保證和測試的相關資訊。例如，您可以在資料庫叢集上指定標籤，以記錄您上次執行切換到該叢集的時間。您可以在全域叢集上指定標籤，以記錄整個應用程式上次災難復原演練的時間。

全球資料庫的標記 AWS CLI 範例

下列 AWS CLI 範例示範如何指定和檢查全球資料庫中所有類型 Aurora 資源的標籤。

您可以使用 `create-global-cluster` 命令指定全域叢集容器的標籤。下列範例會建立全域叢集，並為其指派兩個標籤。標籤具有索引鍵 `tag1` 和 `tag2`。

```
$  aws rds create-global-cluster --global-cluster-identifier my_global_cluster_id \
  --engine aurora-mysql --tags Key=tag1,Value=val1 Key=tag2,Value=val2
```

您可以使用 `describe-global-clusters` 命令列出全域叢集容器上的標籤。使用標籤時，您通常會先執行此命令，以擷取全域叢集的 Amazon Resource Name (ARN)。您可以在後續命令中使用 ARN 作為參數，以搭配使用標籤。下列命令會在 `TagList` 屬性中顯示標籤資訊。它也會顯示在稍後範例中用作參數的 ARN。

```
$  aws rds describe-global-clusters --global-cluster-identifier my_global_cluster_id
{
    "GlobalClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "GlobalClusterArn": "my_global_cluster_arn",
            ...
            "TagList": [
                {
                    "Value": "val1",
                    "Key": "tag1"
                },
                {
                    "Value": "val2",
                    "Key": "tag2"
                }
            ]
        }
    ]
}
```

您可以使用 `add-tags-to-resource` 命令新增標籤。使用此命令，您可以指定全域叢集的 Amazon Resource Name (ARN)，而不是其識別符。新增與現有標籤具有相同名稱的標籤，將會覆寫該標籤的值。如果您在標籤值中包含空格或特殊字元，請根據您的作業系統或命令 shell 引用適當的值。下列範

例會修改先前範例中全域叢集的標籤。最初，叢集具有包含索引鍵 tag1 和 tag2 的標籤。命令完成後，全域叢集具有包含索引鍵 tag3 的新標籤，而包含索引鍵 tag1 的標籤具有不同的值。

```
$  aws rds add-tags-to-resource --resource-name my_global_cluster_arn \
  --tags Key=tag1,Value="new value for tag1" Key=tag3,Value="entirely new tag"

$  aws rds describe-global-clusters --global-cluster-identifier my_global_cluster_id
{
    "GlobalClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            ...
            "TagList": [
                {
                    "Value": "new value for tag1",
                    "Key": "tag1"
                },
                {
                    "Value": "val2",
                    "Key": "tag2"
                },
                {
                    "Value": "entirely new tag",
                    "Key": "tag3"
                }
            ]
        }
    ]
}
```

您可以使用 remove-tags-from-resource 命令從全域叢集刪除標籤。使用此命令，您只需指定一組標籤索引鍵，不需要任何標籤值。下列範例會修改先前範例中全域叢集的標籤。最初，叢集具有包含索引鍵 tag1、tag2 和 tag3 的標籤。命令完成後，只會保留具有索引鍵 tag1 的標籤。

```
$  aws rds remove-tags-from-resource --resource-name my_global_cluster_arn --tag-keys
 tag2 tag3

$  aws rds describe-global-clusters --global-cluster-identifier my_global_cluster_id
{
    "GlobalClusters": [
        {
```

```
            "Status": "available",
            "Engine": "aurora-mysql",
            ...
            "TagList": [
                {
                    "Value": "new value for tag1",
                    "Key": "tag1"
                }
            ]
        }
    ]
}
```

連線至 Amazon Aurora 全域資料庫

每個 Aurora 全域資料庫都有一個寫入器端點，Aurora 會自動更新該端點，將請求路由到主要資料庫叢集的目前寫入器執行個體。使用寫入器端點時，使用受管 Aurora 全域資料庫切換和容錯移轉功能變更主要區域的位置之後，您不需要修改連線字串。若要進一步了解如何使用寫入器端點以及 Aurora 全域資料庫切換和容錯移轉，請參閱 在 Amazon Aurora 全球資料庫中使用切換或容錯移轉。如需使用 RDS Proxy 連線至 Aurora 全域資料庫的資訊，請參閱搭配使用 RDS Proxy 與 Aurora 全域資料庫。

主題

- 選擇符合您應用程式需求的端點
- 檢視 Amazon Aurora 全域資料庫的端點
- 使用全域寫入器端點的考量

選擇符合您應用程式需求的端點

連線至 Aurora 全球資料庫取決於您需要從資料庫讀取或寫入，以及您要路由請求的 AWS 區域。以下是一些典型的使用案例：

- 將請求路由至寫入器執行個體：如果您需要執行資料處理語言 (DML) 和資料定義語言 (DDL) 陳述式，或如果您在讀取和寫入之間需要高度一致性，請連線至 Aurora 全域資料庫寫入器端點。該端點會將請求路由到您全球資料庫主要叢集中的寫入器執行個體。此端點會自動更新，以將請求路由至寫入器執行個體，無需在每次變更全域叢集中的寫入器位置時更新應用程式。您也可以使用全域端點，將跨區域讀取/寫入請求傳送至寫入器。

> **Note**
>
> 如果您在 Aurora 全域資料庫寫入器端點可用之前設定全域資料庫，您的應用程式可能會連線到主要叢集的叢集端點。在此情況下，建議您切換連線設定以改用全域寫入器端點。如此一來，就不需要在每次 Aurora 全域資料庫切換或容錯移轉後變更連線設定。
> 寫入器端點名稱的第一部分是 Aurora 全域資料庫的名稱。因此，如果您重新命名 Aurora 全域資料庫、寫入器端點名稱變更，以及使用它的任何程式碼，都必須更新為新的名稱。

- 擴展讀取更接近您應用程式的區域：若要擴展與您應用程式相同或附近的 AWS 區域中的唯讀請求，請連線至主要或次要 Aurora 叢集的讀取器端點。
- 使用偶爾的跨區域寫入擴展讀取：對於偶爾的 DML 陳述式，例如用於維護和資料清理，請連線至已啟用寫入轉送的次要叢集的讀取器端點。透過寫入轉送，Aurora 會自動將寫入陳述式轉送至 Aurora 全域資料庫主要區域中的寫入器。寫入轉送提供下列優點：
  - 您不需要進行繁重工作，即可在次要叢集和主要叢集之間建立連線，以傳送跨區域寫入。
  - 您不需要分割應用程式中的讀取和寫入請求。
  - 您不需要開發複雜的邏輯來管理 read-after-write 請求的一致性。

  不過，使用寫入轉送時，您需要更新應用程式程式碼或組態，才能在執行跨區域容錯移轉或切換後連線到新提升的主要區域的讀取器端點。我們建議您監控透過寫入轉送完成的操作延遲，以檢查處理寫入請求的額外負荷。最後，寫入轉送不支援某些 MySQL 或 PostgreSQL 操作，例如進行資料定義語言 (DDL) 變更或`SELECT FOR UPDATE`陳述式。

  若要進一步了解如何在 AWS 區域間使用寫入轉送，請參閱 在 Amazon Aurora 全域資料庫中使用寫入轉送。

如需不同類型 Aurora 端點的詳細資訊，請參閱 連接至 Amazon Aurora 資料庫叢集。

檢視 Amazon Aurora 全域資料庫的端點

當您在主控台中檢視 Aurora 全域資料庫時，可以看到與其所有叢集相關聯的所有端點。下圖顯示您在檢視主要資料庫叢集的詳細資訊時看到的端點類型範例：

- 全域寫入器 – 單一讀取/寫入端點，一律指向全域資料庫叢集的目前寫入器資料庫執行個體。
- 寫入器 – 全域資料庫叢集中主要資料庫叢集讀取/寫入請求的連線端點。
- 讀取器 – 全域資料庫叢集中主要或次要資料庫叢集的唯讀請求連線端點。若要將延遲降至最低，請選擇所在 AWS 區域 或最接近之 AWS 區域 中的讀取器端點。



主控台

若要檢視全域資料庫的端點

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 在清單中，選擇全域資料庫，或您要檢視其端點的主要或次要資料庫叢集。
4. 選擇連線和安全索引標籤以查看端點詳細資訊。顯示的端點取決於您選取的叢集類型，如下所示：
   - 全域資料庫 – 全域寫入器端點。
   - 主要資料庫叢集 – 全域寫入器端點，以及主要叢集的叢集端點和讀取器端點。
   - 次要資料庫叢集 – 次要叢集的叢集端點和讀取器端點。在次要叢集上，叢集端點會顯示非作用中狀態，因為它不會處理寫入請求。您仍然可以連線至叢集端點，但僅適用於讀取查詢。

AWS CLI

若要檢視全域叢集的寫入器端點，請使用 AWS CLI describe-global-clusters 命令，如下列範例所示。

```
aws rds describe-global-clusters --region aws_region
{
    "GlobalClusters": [
        {
            "GlobalClusterIdentifier": "global_cluster_id",
            "GlobalClusterResourceId": "cluster-unique_string",
            "GlobalClusterArn": "arn:aws:rds::123456789012:global-
cluster:global_cluster_id",
            "Status": "available",
            "Engine": "aurora-mysql",
            "EngineVersion": "5.7.mysql_aurora.2.11.2",
            "GlobalClusterMembers": [

              ...
            ],
            "Endpoint":
 "global_cluster_id.global-unique_string.global.rds.amazonaws.com"
        }
    ]
}
```

若要檢視全域叢集成員資料庫叢集的叢集和讀取器端點，請使用 AWS CLI describe-db-clusters 命令，如下列範例所示。針對 Endpoint 和 ReaderEndpoint 傳回的值分別是叢集和讀取器端點。

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier db_cluster_id
{
    "DBClusters": [
        {
            "AllocatedStorage": 1,
            "AvailabilityZones": [
                "az_1",
                "az_2",
                "az_3"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "db_cluster_id",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "DBSubnetGroup": "default",
```

```
            "Status": "available",
            "EarliestRestorableTime": "2023-08-01T18:21:11.301Z",
            "Endpoint":
 "db_cluster_id.cluster-unique_string.primary_region.rds.amazonaws.com",
            "ReaderEndpoint": "db_cluster_id.cluster-
ro-unique_string.primary_region.rds.amazonaws.com",
            "MultiAZ": false,
            "Engine": "aurora-mysql",
            "EngineVersion": "5.7.mysql_aurora.2.11.2",

            "ReadReplicaIdentifiers": [
                "arn:aws:rds:secondary_region:123456789012:cluster:db_cluster_id"
            ],
            "DBClusterMembers": [
                {
                    "DBInstanceIdentifier": "db_instance_id",
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "PromotionTier": 1
                }
            ],

            ...
            "TagList": [],
            "GlobalWriteForwardingRequested": false
        }
    ]
}
```

RDS API

若要檢視全域叢集的寫入器端點，請使用 RDS API DescribeGlobalClusters 操作。若要檢視全域叢集成員資料庫叢集的叢集和讀取器端點，請使用 RDS API DescribeDBClusters 操作。

使用全域寫入器端點的考量

您可以遵循下列準則和最佳實務，有效使用 Aurora 全域資料庫寫入器端點：

- 若要將跨區域容錯移轉或切換後的中斷降至最低，您可以在應用程式運算與主要和次要 AWS 區域之間設定 VPC 連線。例如，假設您的應用程式或用戶端系統在與主要叢集相同的 VPC 中執行。如果次要叢集提升，全域寫入器端點會自動變更為指向該叢集。雖然全域寫入器端點可讓您避免變更應用程式的連線設定，但在您設定兩個 VPC 之間的聯網之前，您的應用程式無法存取新提升主要 AWS

區域 VPC 中的 IP 位址。請參閱 Amazon VPC-to-Amazon VPC 連線選項，以評估設定此連線的不同選項。

- 全域資料庫容錯移轉或切換之後的全域寫入器端點更新可能需要很長的時間，具體取決於您的網域名稱服務 (DNS) 快取持續時間。如需進一步了解，請參閱 Amazon Aurora MySQL 資料庫管理員手冊。當 Aurora 全域資料庫在全域寫入器端點上看到 DNS 變更時，會發出 RDS 事件。您可以使用事件來設計策略，以確保 DNS 快取不會超過事件產生後的時間。如需更多詳細資訊，請參閱 資料庫叢集事件。
- Aurora 全域資料庫會以非同步方式複寫資料。跨區域容錯移轉方法可能會導致一些寫入交易資料在容錯移轉啟動之前未複寫到選擇的次要區域。雖然 Aurora 會盡力封鎖原始主要 AWS 區域中的寫入，但容錯移轉可能會容易發生核心分裂問題。將資料遺失和核心分裂風險降至最低的考量也適用於 Aurora 全域資料庫寫入器端點。如需更多詳細資訊，請參閱 針對 Aurora 全球資料庫執行受管容錯移轉。

在 Amazon Aurora 全域資料庫中使用寫入轉送

您可以使用寫入轉送，減少針對 Aurora 全域資料庫上執行之應用程式所需管理的端點數目。啟用寫入轉送，可讓 Aurora 全球資料庫中的次要叢集將執行寫入操作的 SQL 陳述式轉送至主要叢集。主要叢集會更新來源，然後將產生的變更傳播回所有次要 AWS 區域。

寫入轉送組態使您不必實作自己的機制，即可將寫入操作從次要 AWS 區域傳送至主要區域。Aurora 會處理跨區域聯網設定。Aurora 還為每個陳述句傳輸所有必要的工作階段和交易內容。資料的變更一律會先在主要叢集上發生，然後複寫到 Aurora 全域資料庫的次要叢集。如此一來，主要叢集是真實的來源，並永遠擁有最新的所有資料副本。

主題

- 在 Aurora MySQL 全球資料庫中使用寫入轉送
- 在 Aurora PostgreSQL 全域資料庫中使用寫入轉送

在 Aurora MySQL 全球資料庫中使用寫入轉送

主題

- Aurora MySQL 中寫入轉發的區域和版本可用性
- 在 Aurora MySQL 中啟用寫入轉送
- 在 Aurora MySQL 中檢查次要叢集是否已啟用寫入轉送
- Aurora MySQL 中應用程式和 SQL 與寫入轉送的相容性

- Aurora MySQL 中寫入轉送的隔離與一致性
- 在 Aurora MySQL 中使用寫入轉送執行多部分陳述式
- Aurora MySQL 中具有寫入轉送的交易
- Aurora MySQL 中寫入轉送的組態參數
- Aurora MySQL 中寫入轉送的 Amazon CloudWatch 指標
- 用於寫入轉送的 Aurora MySQL 狀態變數

Aurora MySQL 中寫入轉發的區域和版本可用性

在每個可使用 Aurora MySQL 全球資料庫的區域中，Aurora MySQL 2.08.1 及更高版本都支援寫入轉送。

如需 Aurora MySQL 全球資料庫的版本和區域可用性的詳細資訊，請參閱 使用 Aurora MySQL 的 Aurora 全域資料庫。

在 Aurora MySQL 中啟用寫入轉送

根據預設，當您將次要叢集新增至 Aurora 全域資料庫時，不會啟用寫入轉送。

若要使用 AWS 管理主控台 啟用寫入轉送，請在為全球資料庫新增區域時，選取僅供讀取複本寫入轉送中的開啟全域寫入轉送選項。對於現有的次要叢集，請將叢集修改為開啟全域寫入轉送。若要關閉寫入轉送，請在新增區域或修改次要叢集時，清除開啟全域寫入轉送核取方塊。

若要使用 AWS CLI 啟用寫入轉送，請使用 `--enable-global-write-forwarding` 選項。當您使用 `create-db-cluster` 命令建立新的次要叢集時，此選項會起作用。當您使用 `modify-db-cluster` 命令修改現有的次要叢集時，此選項也會起作用。其要求全域資料庫使用支援寫入轉送的 Aurora 版本。您可以使用 `--no-enable-global-write-forwarding` 選項搭配這些相同的 CLI 命令來關閉寫入轉送。

若要使用 Amazon RDS API 啟用寫入轉送，請將 `EnableGlobalWriteForwarding` 參數設定為 `true`。當您使用 `CreateDBCluster` 操作建立新的次要叢集時，此參數會起作用。當您使用 `ModifyDBCluster` 操作修改現有的次要叢集時，此選項也會起作用。其要求全域資料庫使用支援寫入轉送的 Aurora 版本。您可以將 `EnableGlobalWriteForwarding` 參數設定為 `false` 來關閉寫入轉送。

> **Note**
>
> 若要讓資料庫執行階段使用寫入轉送，請指定 aurora_replica_read_consistency 組態參數的設定。在使用寫入轉送功能的每個工作階段中執行這項操作。如需此參數的相關資訊，請參閱 Aurora MySQL 中寫入轉送的隔離與一致性。
> RDS 代理功能不支援 aurora_replica_read_consistency 變數的 SESSION 值。設定此值可能會導致未預期的行為。

下列 CLI 範例顯示如何設定已啟用或停用寫入轉送的 Aurora 全域資料庫。反白顯示的項目代表在設定 Aurora 全域資料庫的基礎設施時，要保持一致的重要指定指令和選項。

下列範例會建立 Aurora 全域資料庫、主要叢集和啟用寫入轉送的次要叢集。將您自己的選擇替換為使用者名稱、密碼以及主要和次要 AWS 區域。

```
# Create overall global database.
aws rds create-global-cluster --global-cluster-identifier write-forwarding-test \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

# Create primary cluster, in the same AWS Region as the global database.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-1 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --master-username user_name --master-user-password password \
  --region us-east-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \
  --db-instance-identifier write-forwarding-test-cluster-1-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \
  --db-instance-identifier write-forwarding-test-cluster-1-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

# Create secondary cluster, in a different AWS Region than the global database,
# with write forwarding enabled.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
```

```
  --db-cluster-identifier write-forwarding-test-cluster-2 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2 \
  --enable-global-write-forwarding

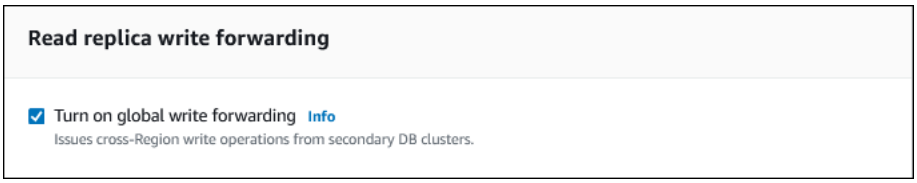
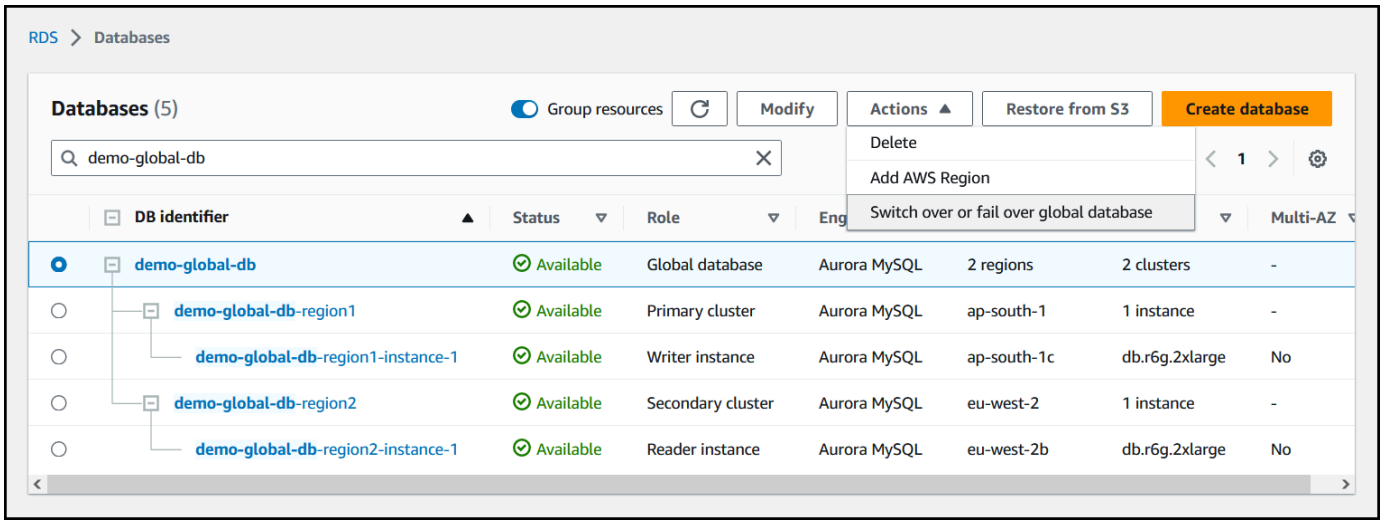
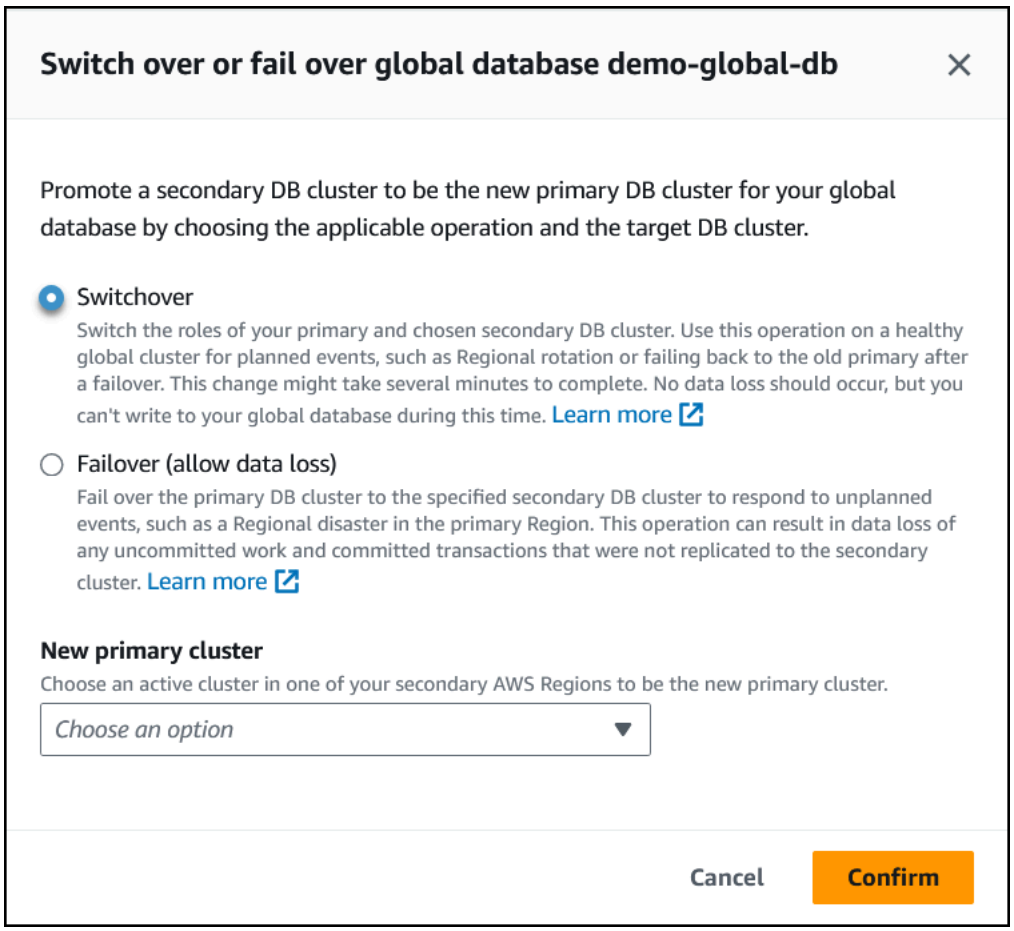
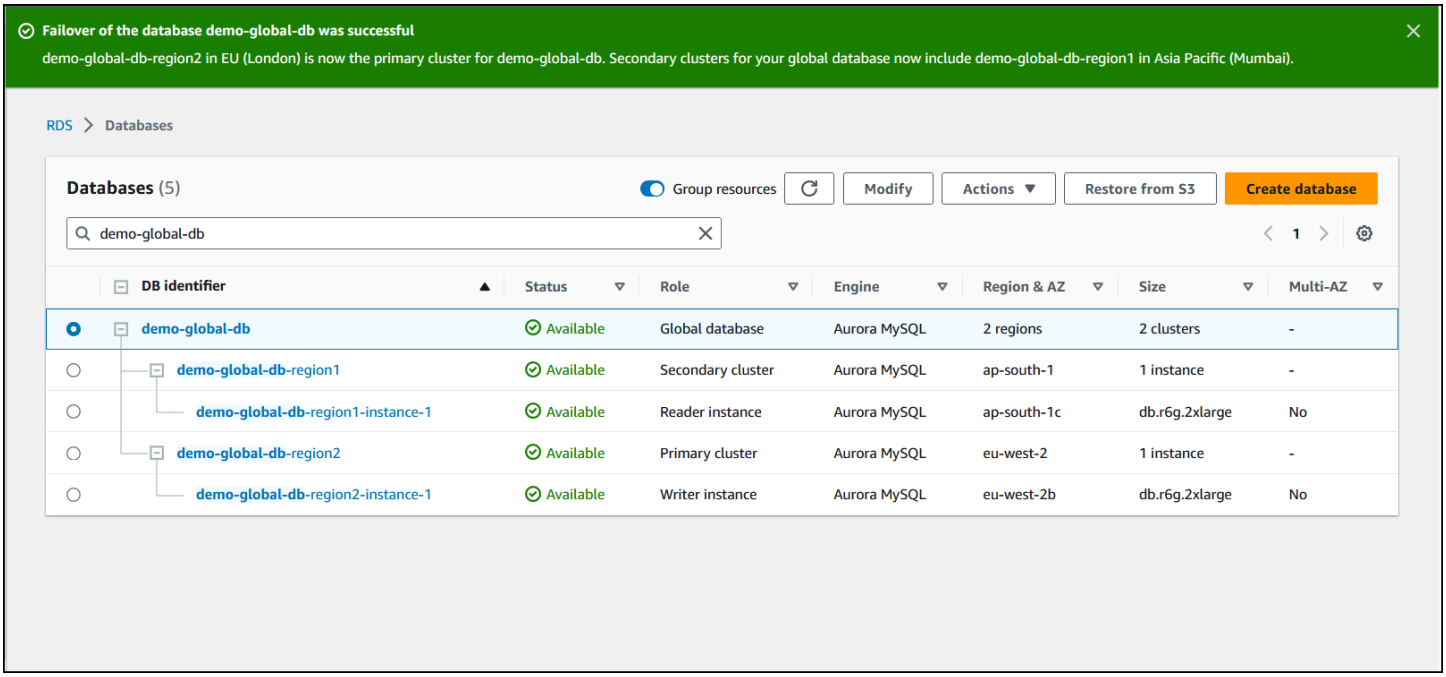
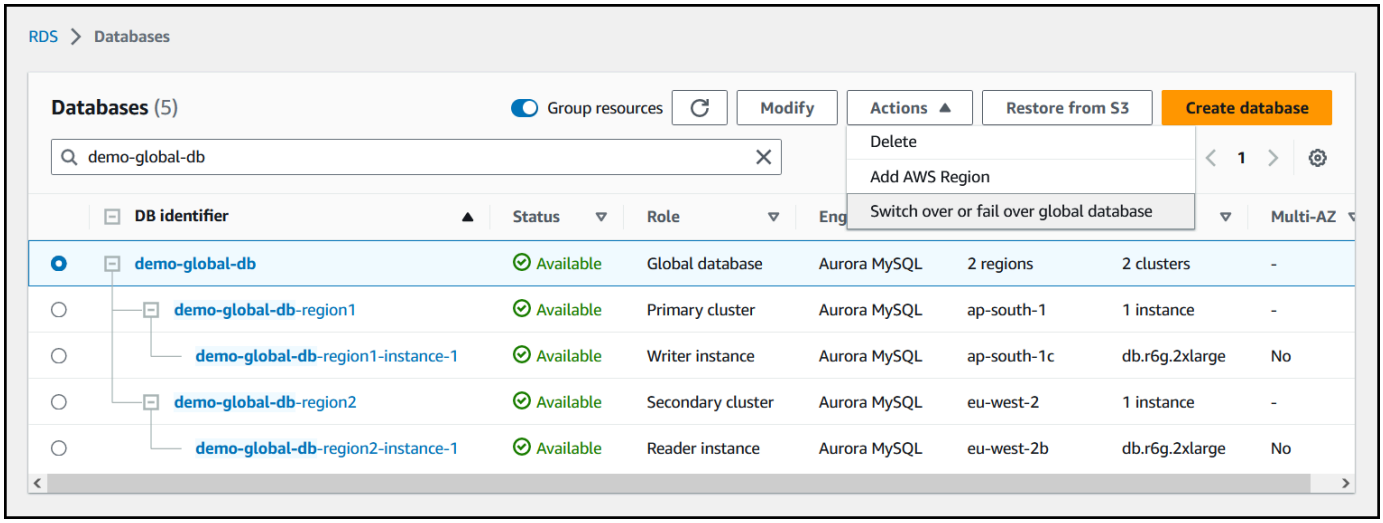
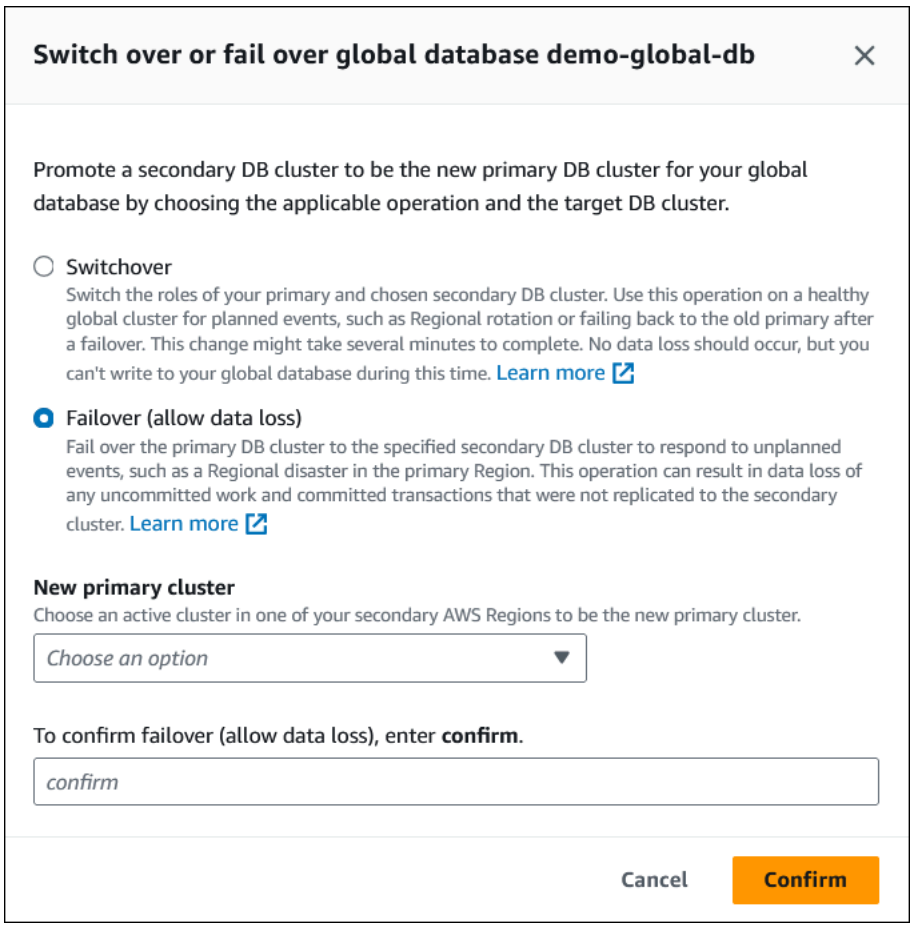
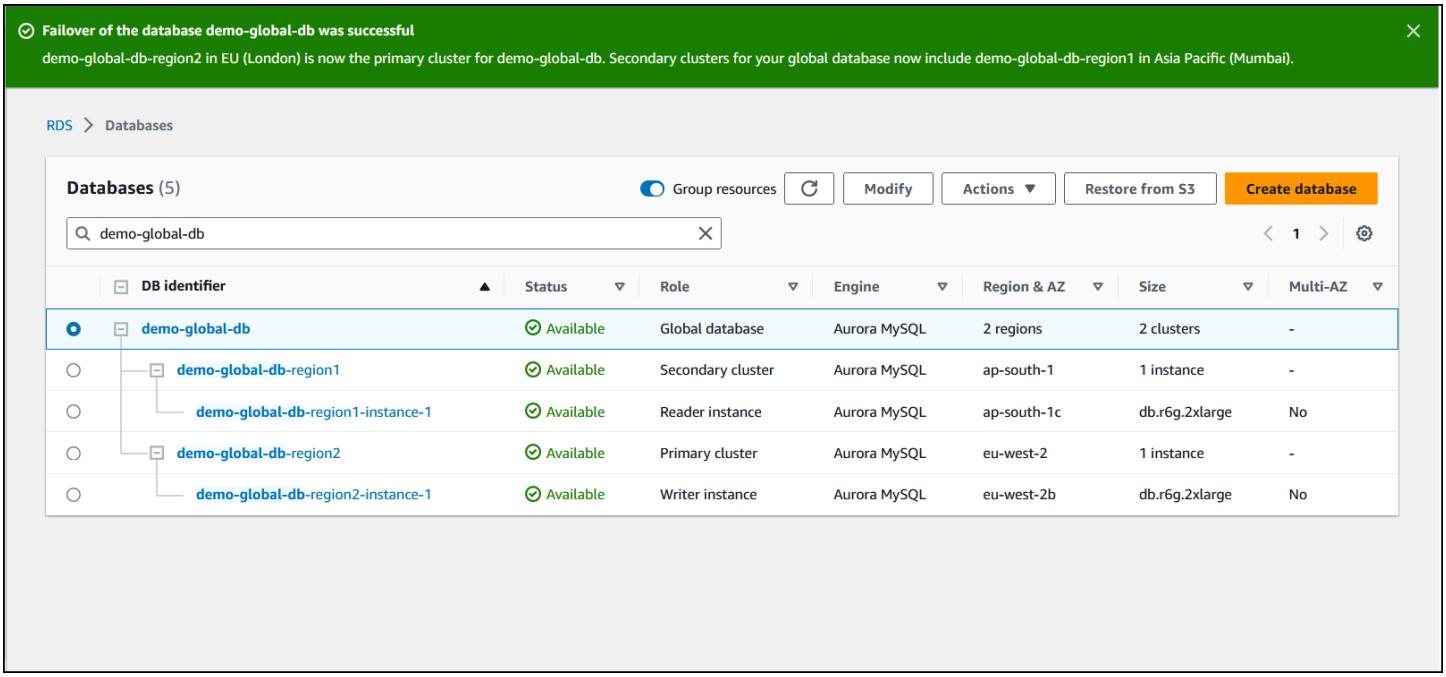
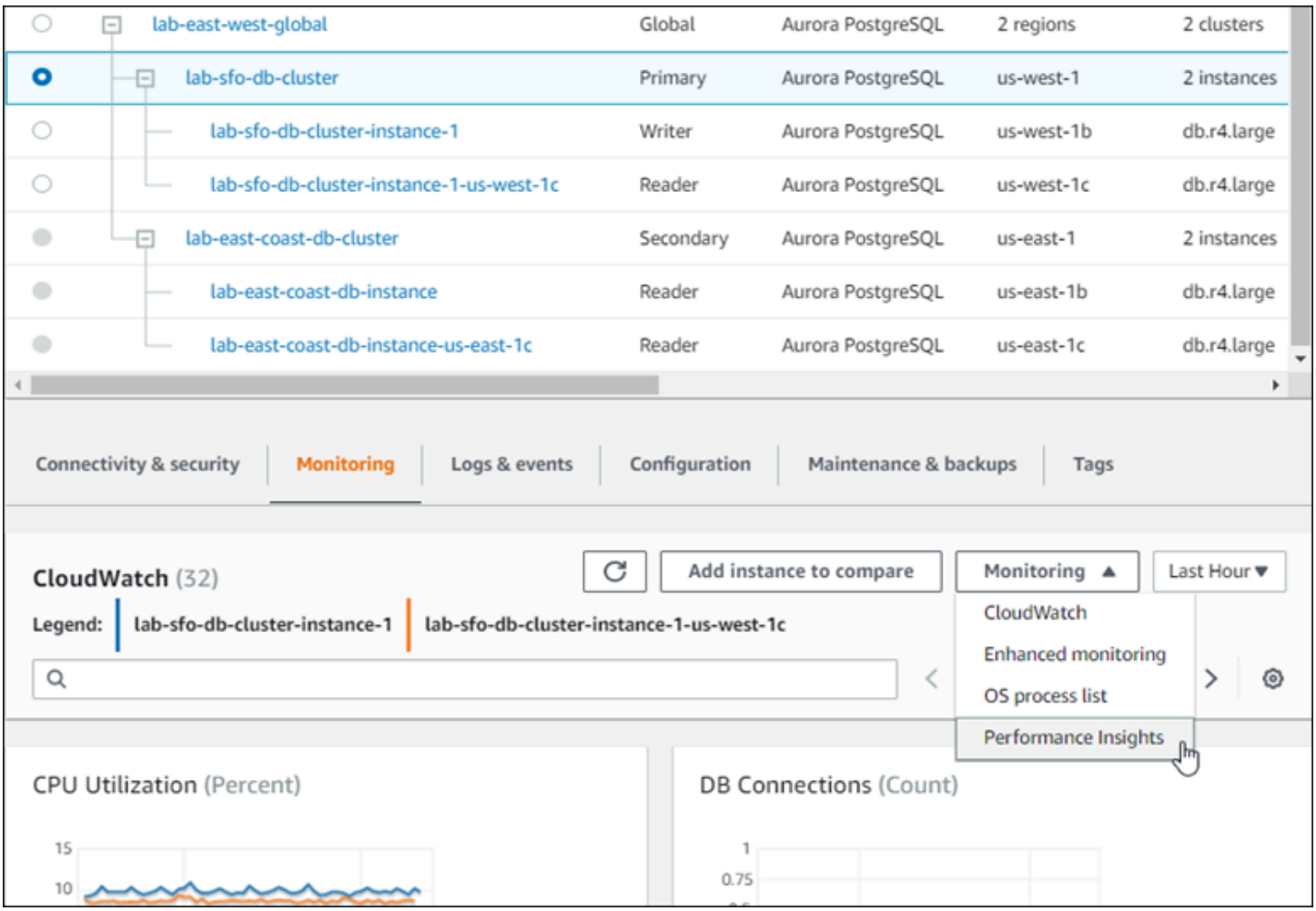
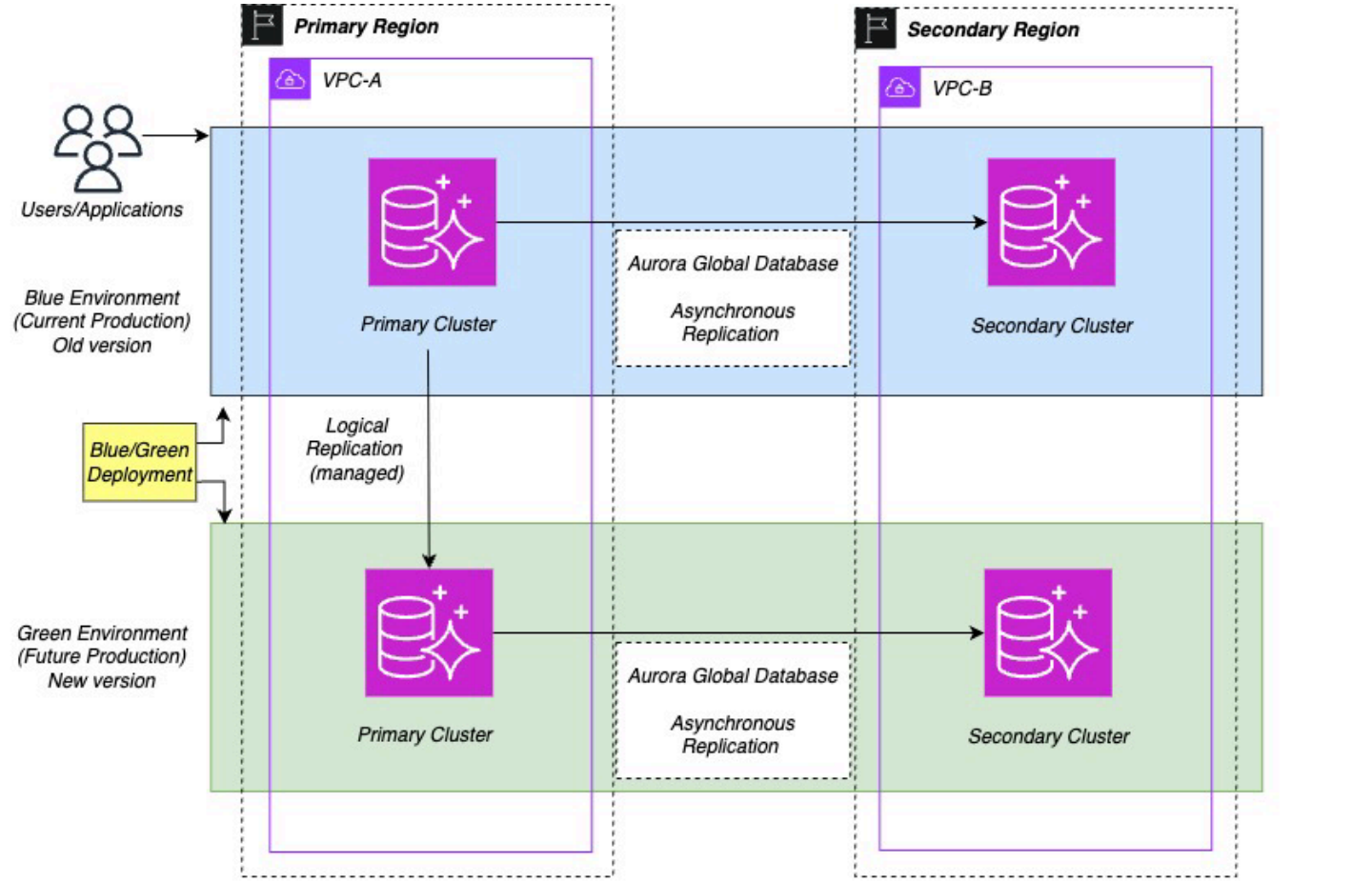
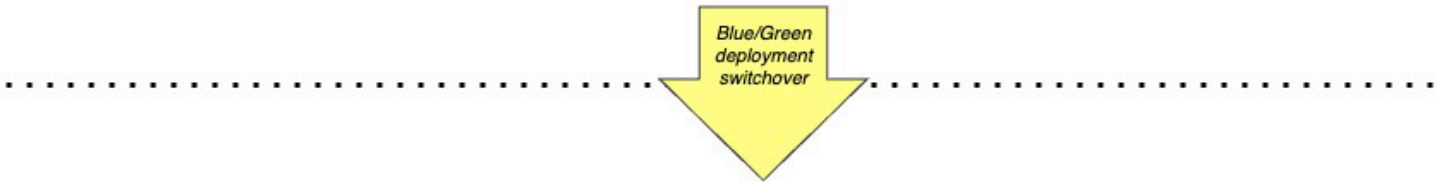
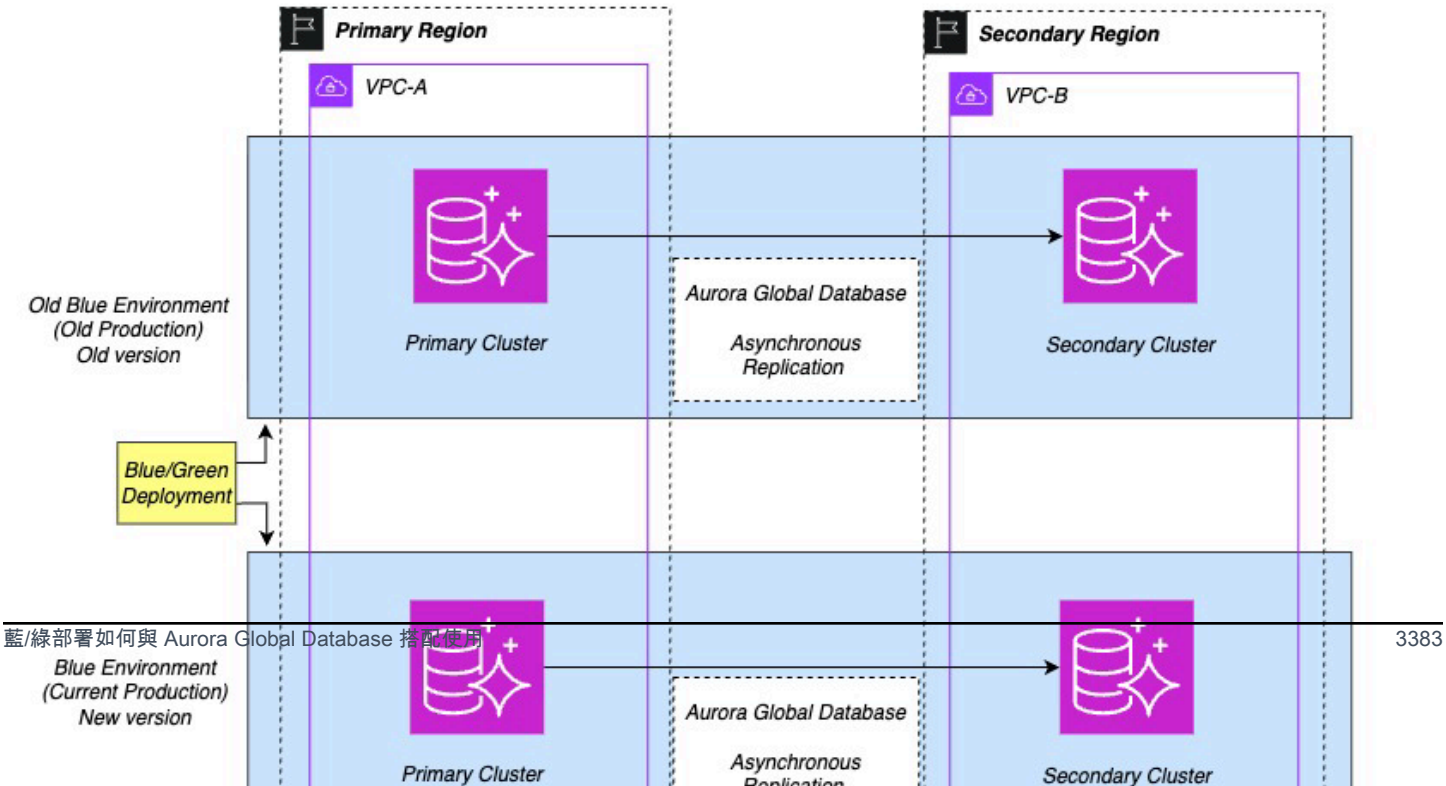
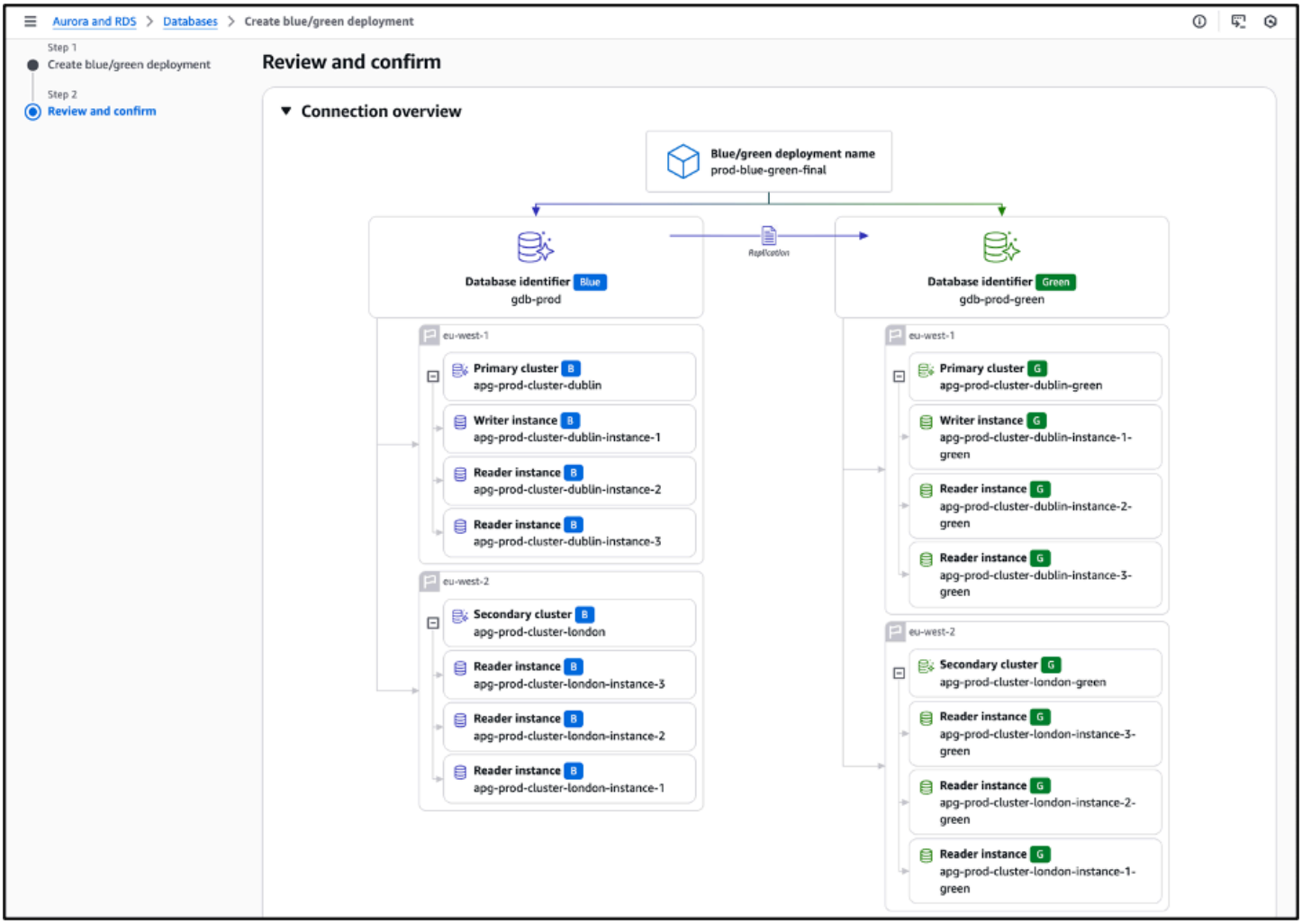
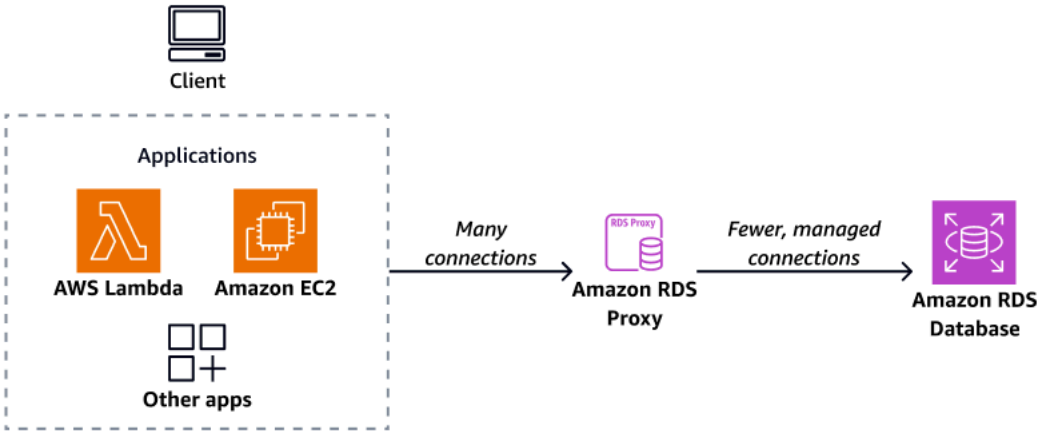
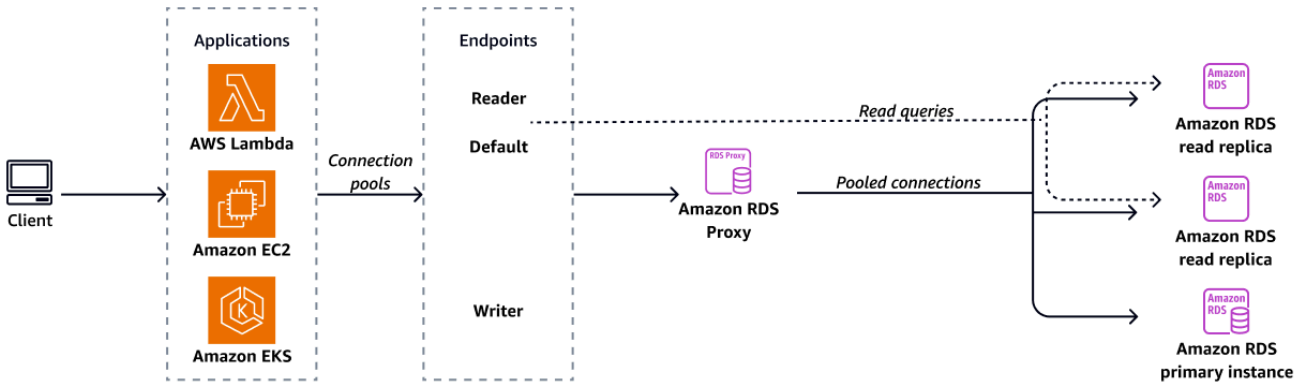
aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2
```

下列範例會從前一個範例繼續。此範例會建立一個沒有啟用寫入轉送的次要叢集，然後啟用寫入轉送。完成此範例之後，全域資料庫中的所有次要叢集都已啟用寫入轉送。

```
# Create secondary cluster, in a different AWS Region than the global database,
# without write forwarding enabled.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-2 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds modify-db-cluster --db-cluster-identifier write-forwarding-test-cluster-2 \
  --region us-east-2 \
  --enable-global-write-forwarding
```

在 Aurora MySQL 中檢查次要叢集是否已啟用寫入轉送

若要判斷是否可以從次要叢集使用寫入轉送，您可以檢查叢集是否具有屬性 `"GlobalWriteForwardingStatus": "enabled"`。

在 AWS 管理主控台 中，叢集的詳細資訊頁面上的組態標籤，您可以看到僅供讀取全域複本寫入轉送的狀態為啟用。

若要查看所有叢集的全域寫入轉送設定狀態，請執行下列 AWS CLI 命令。

次要叢集會顯示值 `"enabled"` 或 `"disabled"`，指出寫入轉送是開啟或關閉。`null` 值表示該叢集無法使用寫入轉送。此叢集不屬於全域資料庫，或是主要叢集，而是次要叢集。如果寫入轉送處於開啟或關閉的程序中，此值也可以 `"enabling"` 是 `"disabling"`。

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu

[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    },
    {
        "GlobalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-2"
    },
    {
        "GlobalWriteForwardingStatus": null,
        "DBClusterIdentifier": "non-global-cluster"
    }
]
```

若要尋找已啟用全域寫入轉送的次要叢集，請執行下列命令。此命令也會傳回叢集的讀取者端點。當您使用寫入轉送從 Aurora 全域資料庫的次要叢集到主要叢集時，您可以使用次要叢集的讀取者端點。

Example

```
aws rds describe-db-clusters --query 'DBClusters[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
 | [?GlobalWriteForwardingStatus == `enabled`]'
```

```
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "ReaderEndpoint": "aurora-write-forwarding-test-replica-1.cluster-ro-
cnpexample.us-west-2.rds.amazonaws.com",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    }
]
```

Aurora MySQL 中應用程式和 SQL 與寫入轉送的相容性

您可以使用以下類型的 SQL 陳述式搭配寫入轉送：

- 資料操作語言 (DML) 陳述式，例如 INSERT、DELETE 和 UPDATE。這些陳述式的屬性有一些限制，您可以將這些屬性與寫入轉送搭配使用，如下所述。
- SELECT ... LOCK IN SHARE MODE 和 SELECT FOR UPDATE 陳述式。
- PREPARE 和 EXECUTE 陳述式。

當您在具有寫入轉送的全域資料庫中使用某些陳述式時，系統不允許使用這些陳述式或這些陳述式可能會產生過時的結果。因此，次要叢集的 EnableGlobalWriteForwarding 設定預設為關閉。在開啟此功能之前，請檢查以確定您的應用程式程式碼不受上述任何限制的影響。

下列限制適用於您與寫入轉送搭配使用的 SQL 陳述式。在某些情況下，您可以在叢集層級，在啟用寫入轉送的次要叢集上使用陳述式。如果在工作階段中的寫入轉送開啟方式不是透過 aurora_replica_read_consistency 組態參數，則此方法有效。在不允許此方法時嘗試使用陳述式，因為寫入轉送會導致以下格式的錯誤訊息。

```
ERROR 1235 (42000): This version of MySQL doesn't yet support 'operation with write
 forwarding'.
```

資料定義語言 (DDL)

連線至主要叢集以執行 DDL 陳述式。您無法從讀取器資料庫執行個體執行。

使用臨時資料表中的資料更新永久資料表

您可以在啟用寫入轉送的次要叢集上使用臨時資料表。但是，如果陳述式參照臨時資料表，則無法使用 DML 陳述式來修改永久資料表。例如，您不能使用從臨時資料表取得資料的 INSERT ... SELECT 陳述式。臨時資料表存在於次要叢集上，且當陳述式在主要叢集上執行時無法使用該資料表。

XA 交易

在工作階段中開啟寫入轉送時，您無法在次要叢集上使用下列陳述式。您可以在未啟用寫入轉送的次要叢集上，或在 aurora_replica_read_consistency 設定為空的工作階段中使用這些陳述式。在工作階段中開啟寫入轉送之前，請檢查您的程式碼是否使用這些陳述式。

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER [CONVERT XID]
```

永久資料表的 LOAD 陳述式

您無法在啟用寫入轉送的次要叢集上使用下列陳述式。

```
LOAD DATA INFILE 'data.txt' INTO TABLE t1;
        LOAD XML LOCAL INFILE 'test.xml' INTO TABLE t1;
```

您可以將資料載入次要叢集上的臨時資料表。不過，請確定您執行的任何 LOAD 陳述式只參照主要叢集上的永久資料表。

外掛程式陳述式

您無法在啟用寫入轉送的次要叢集上使用下列陳述式。

```
INSTALL PLUGIN example SONAME 'ha_example.so';
UNINSTALL PLUGIN example;
```

儲存點陳述式

在工作階段中開啟寫入轉送時，您無法在次要叢集上使用下列陳述式。您可以在未啟用寫入轉送的次要叢集上，或在 aurora_replica_read_consistency 設定為空的工作階段中使用這些陳述式。在工作階段中開啟寫入轉送之前，請檢查您的程式碼是否使用這些陳述式。

```
SAVEPOINT t1_save;
ROLLBACK TO SAVEPOINT t1_save;
RELEASE SAVEPOINT t1_save;
```

Aurora MySQL 中寫入轉送的隔離與一致性

在使用寫入轉送的工作階段中，您只能使用 REPEATABLE READ 隔離層級。雖然您也可以在次要 READ COMMITTED 區域中使用具唯讀叢集的 AWS 隔離層級，但該隔離層級不適用於寫入轉送。如需 REPEATABLE READ 和 READ COMMITTED 隔離層級的相關資訊，請參閱 Aurora MySQL 隔離層級。

您可以控制次要叢集上的讀取一致性程度。讀取一致性層級會決定次要叢集在每次讀取操作之前的等待時間，以確保從主要叢集複寫部分或全部變更。您可以調整讀取一致性層級，以確保在任何後續查詢之前，您都可以在次要叢集中看見工作階段中的所有轉送寫入操作。您也可以使用此設定，確保次要叢集上的查詢永遠會看到主要叢集的最新更新。即使是由其他工作階段或其他叢集所提交的更新。若要為應用程式指定這種行為類型，請選擇工作階段層級參數 aurora_replica_read_consistency 的值。

> **Important**
>
> 務必為您要轉送寫入的任何工作階段設定 aurora_replica_read_consistency 參數。否則，Aurora 不會啟用該工作階段的寫入轉送。此參數預設為空值，因此當您使用此參數時，請選擇特定值。該 aurora_replica_read_consistency 參數只會對啟用寫入轉送的次要叢集產生影響。
>
> 針對 Aurora MySQL 2 版和低於 3.04 的 3 版，請使用 aurora_replica_read_consistency 作為工作階段變數。針對 Aurora MySQL 3.04 版及更新版本，您可以使用 aurora_replica_read_consistency 作為工作階段變數或作為資料庫叢集參數。

對於 aurora_replica_read_consistency 參數，您可以指定值 EVENTUAL、SESSION 和 GLOBAL。

當您提高一致性層級時，您的應用程式會花費更多時間，等待在 AWS 區域之間傳播變更。您可以選擇在快速回應時間之間的平衡，並確保在查詢執行之前，在其他位置所做的變更完全可用。

將讀取一致性設定為 EVENTUAL 後，使用寫入轉送之次要 AWS 區域中的查詢可能會看到由於複寫延遲而稍微過時的資料。在主要區域上執行寫入操作並複寫到目前的區域之前，不會顯示相同工作階段中寫入操作的結果。查詢不會等待更新的結果變成可用。因此，它可能會擷取較舊的資料或更新的資料，視陳述式的時間和複寫延遲量而定。

將讀取一致性設為 SESSION 後，次要 AWS 區域中使用寫入轉送的所有查詢都會看到該工作階段中所做之所有變更的結果。無論交易是否已遞交，這些變更都是可見的。如有必要，查詢會等待轉送寫入操

作的結果複寫到目前的區域。其不會等待來自寫入操作的更新結果，這些寫入操作的執行位置位於其他區域或目前區域內其他工作階段。

將讀取一致性設為 GLOBAL 後，次要 AWS 區域中的工作階段會看到該工作階段所做的變更。其也可以看到來自主要 AWS 區域和其他次要 AWS 區域的所有已遞交變更。每個查詢可能會等待一段時間，長短取決於工作階段的延遲量。自查詢開始的時間起，當次要叢集與主要叢集中的所有遞交資料都是最新時，查詢就會繼續。

如需寫入轉送中所含所有參數的詳細資訊，請參閱 Aurora MySQL 中寫入轉送的組態參數。

使用寫入轉送的範例

這些範例使用 aurora_replica_read_consistency 作為工作階段變數。針對 Aurora MySQL 3.04 版及更新版本，您可以使用 aurora_replica_read_consistency 作為工作階段變數或作為資料庫叢集參數。

在以下範例中，主要叢集位於美國東部 (維吉尼亞北部) 區域。次要叢集位於美國東部 (俄亥俄) 區域中。此範例顯示在執行 INSERT 陳述式後接著執行 SELECT 陳述式的效果。視 aurora_replica_read_consistency 設定的值而定，結果可能會因陳述式的時間而有所不同。為了實現更高的一致性，在發出 SELECT 陳述式之前，您可能需要稍作等待。或者，Aurora 可以自動等待結果完成複寫，然後再繼續進行 SELECT。

在此範例中，具有讀取一致性設定 eventual。立即執行 INSERT 陳述式，然後執行 SELECT 陳述式，仍會傳回 COUNT(*) 的值。該值反映插入新列之前的列數。稍後再次執行 SELECT 會傳回更新的資料列計數。這些 SELECT 陳述式不會等待。

```
mysql> set aurora_replica_read_consistency = 'eventual';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
mysql> insert into t1 values (6); select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
mysql> select count(*) from t1;
```

```
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)
```

如果讀取一致性設定為 session，則在 SELECT 陳述式中的變更顯示前，INSERT 陳述式會在 INSERT 等待後立即執行。後續的 SELECT 陳述式不會等待。

```
mysql> set aurora_replica_read_consistency = 'session';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.01 sec)
mysql> insert into t1 values (6); select count(*) from t1; select count(*) from t1;
Query OK, 1 row affected (0.08 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.37 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.00 sec)
```

將讀取一致性設定仍設為 session 後，執行 INSERT 陳述式後稍等一下，讓更新的資料列計數可在下一個 SELECT 陳述式執行時使用。

```
mysql> insert into t1 values (6); select sleep(2); select count(*) from t1;
Query OK, 1 row affected (0.07 sec)
+----------+
| sleep(2) |
+----------+
|        0 |
```

```
+----------+
1 row in set (2.01 sec)
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

將讀取一致性設定設為 global 後，每個 SELECT 陳述式都會等待，以確保自陳述式開始時間起的所有資料變更都可見，然後再執行查詢。等待每個 SELECT 陳述式的時間會根據主要和次要叢集之間的複寫延遲量而有所不同。

```
mysql> set aurora_replica_read_consistency = 'global';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.75 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.37 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.66 sec)
```

在 Aurora MySQL 中使用寫入轉送執行多部分陳述式

DML 陳述式可能包含多個部分，例如 INSERT ... SELECT 陳述式或 DELETE ... WHERE 陳述式。在這種情況下，系統會將整個陳述式轉送到主要叢集並在該處執行陳述式。

Aurora MySQL 中具有寫入轉送的交易

交易是否被轉送到主要叢集取決於交易的存取模式。您可以使用 SET TRANSACTION 陳述式或 START TRANSACTION 陳述式，來指定交易的存取模式。您也可以透過變更 transaction_read_only 工作階段變數的值，來指定交易存取模式。您只能在連線至已啟用寫入轉送的資料庫叢集時，變更此工作階段值。

如果長時間執行的交易經過很長一段時間內都未發出任何陳述式，則可能會超過閒置逾時期間。此期間的預設值為一分鐘。您最多可以增加一天。超過閒置逾時的交易會被主要叢集取消。您提交的下一個後續陳述式會收到逾時錯誤。然後 Aurora 會復原交易。

當寫入轉送變成無法使用時，可能會發生這種類型的錯誤。例如，如果您重新啟動主要叢集或關閉寫入轉送組態設定，Aurora 就會取消任何使用寫入轉送的交易。

Aurora MySQL 中寫入轉送的組態參數

Aurora 叢集參數群組包含寫入轉送功能的設定。因為這些是叢集參數，所以每個叢集中的所有資料庫執行個體都有這些變數的相同值。下表列出了有關這些參數的詳細資訊，並在表格後面附有使用注意事項。

名稱	範圍	類型	預設值	有效值
aurora_fwd_master_idle_timeout (Aurora MySQL 第 2 版)	全球服務	不帶正負號整數	60	1–86,400
aurora_fwd_master_max_connections_pct (Aurora MySQL 第 2 版)	全球服務	不帶正負號長整數	10	0–90
aurora_fwd_writer_idle_timeout (Aurora MySQL 第 3 版)	全球服務	不帶正負號整數	60	1–86,400
aurora_fwd_writer_max_connections_pct (Aurora MySQL 第 3 版)	全球服務	不帶正負號長整數	10	0–90
aurora_replica_read_consistency	工作階段適用於低於 3.04 的第 2	列舉	" (null)	EVENTUAL, SESSION, GLOBAL

名稱	範圍	類型	預設值	有效值
	版和第 3 版工作階段，全域適用於第 3.04 版及更高版本			

若要控制次要叢集的傳入寫入請求，請在主要叢集上使用這些設定：

- aurora_fwd_master_idle_timeout、aurora_fwd_writer_idle_timeout：主要叢集在關閉從次要叢集轉送的連線前等待活動的秒數。如果工作階段在此期間之後仍處於閒置狀態，則 Aurora 會取消工作階段。
- aurora_fwd_master_max_connections_pct、aurora_fwd_writer_max_connections_pct：可在寫入器資料庫執行個體上，用來處理從讀取器轉送之查詢的資料庫連線上限。此上限的表示方式是主要叢集中寫入器資料庫執行個體的 max_connections 設定百分比。例如，如果 max_connections 是 800，且 aurora_fwd_master_max_connections_pct 或 aurora_fwd_writer_max_connections_pct 是 10，則寫入器允許最多 80 個同時轉送的工作階段。這些連線來自 max_connections 設定所管理的相同連線集區。

  當一或多個次要叢集已啟用寫入轉送時，此設定僅適用於主要叢集。如果您減少此值，現有的連線不會受到影響。Aurora 在嘗試從次要叢集建立新連線時，會考慮設定的新值。預設值為 10，代表該 max_connections 值的 10%。如果您在任何次要叢集上啟用查詢轉送，此設定必須為非零值，才能成功從次要叢集進行寫入操作。如果值為零，則寫入操作會收到含有訊息 ER_CON_COUNT_ERROR 的錯誤碼 Not enough connections on writer to handle your request。

aurora_replica_read_consistency 參數會啟用寫入轉送。您可以在每個工作階段中使用。您可以指定 EVENTUAL SESSION 或 GLOBAL 以實現讀取一致性層級。若要進一步了解一致性層級，請參閱Aurora MySQL 中寫入轉送的隔離與一致性。下列規則適用於此參數：

- 預設值為 '' (空白)。
- 只有在 aurora_replica_read_consistency 設定為 EVENTUAL、SESSION 或 GLOBAL 時，才能在工作階段中使用寫入轉送。此參數僅在啟用寫入轉送之次要叢集的讀取器執行個體中，且該次要叢集位於 Aurora 全域資料庫中才有意義。

- 您無法在多陳述式交易中設定此變量 (當為空時) 或取消設定 (當已經設定時)。但是，您可以在此類交易期間將其從一個有效值 (EVENTUAL、SESSION、或 GLOBAL) 變更為另一個有效值 (EVENTUAL、SESSION、或 GLOBAL)。
- 當次要叢集上未啟用寫入轉送時，此變數不得是 SET。

Aurora MySQL 中寫入轉送的 Amazon CloudWatch 指標

當您在一或多個次要叢集上使用寫入轉送時，下列 Amazon CloudWatch 指標適用於主要叢集。這些指標都是在主要叢集中寫入器資料庫執行個體上測量的。

CloudWatch 指標	單位	描述
AuroraDMLRejectedMasterFull	計數	由於寫入器資料庫執行個體上的工作階段已滿，因此拒絕的轉送查詢數量。 適用於 Aurora MySQL 第 2 版。
AuroraDMLRejectedWriterFull	計數	由於寫入器資料庫執行個體上的工作階段已滿，因此拒絕的轉送查詢數量。 適用於 Aurora MySQL 3 版。
ForwardingMasterDMLLatency	毫秒	處理寫入器資料庫執行個體上每個轉送 DML 陳述式的平均時間。 它不包括次要叢集轉送寫入請求的時間，或將變更複寫至次要叢集的時間。 適用於 Aurora MySQL 第 2 版。
ForwardingMasterDMLThroughput	每秒計數	此寫入器資料庫執行個體每秒處理的轉送 DML 陳述式數目。

CloudWatch 指標	單位	描述
		適用於 Aurora MySQL 第 2 版。
ForwardingMasterOpenSessions	計數	寫入器資料庫執行個體上轉送的工作階段數目。 適用於 Aurora MySQL 第 2 版。
ForwardingWriterDMLLatency	毫秒	處理寫入器資料庫執行個體上每個轉送 DML 陳述式的平均時間。 它不包括次要叢集轉送寫入請求的時間，或將變更複寫至次要叢集的時間。 適用於 Aurora MySQL 3 版。
ForwardingWriterDMLThroughput	每秒計數	此寫入器資料庫執行個體每秒處理的轉送 DML 陳述式數目。 適用於 Aurora MySQL 3 版。
ForwardingWriterOpenSessions	計數	寫入器資料庫執行個體上轉送的工作階段數目。 適用於 Aurora MySQL 3 版。

下列 CloudWatch 指標適用於每個次要叢集。這些指標的測量是在啟用寫入轉送的次要叢集中，每個讀取器資料庫執行個體上進行。

CloudWatch 指標	單位	描述
ForwardingReplicaDMLLatency	毫秒	複本上轉送 DML 的平均回應時間。

CloudWatch 指標	單位	描述
ForwardingReplicaDMLThroughput	每秒計數	每秒處理的轉送 DML 陳述式數目。
ForwardingReplicaOpenSessions	計數	在讀取器資料庫執行個體上使用寫入轉送的工作階段數目。
ForwardingReplicaReadWaitLatency	毫秒	讀取器資料庫執行個體上的 SELECT 陳述式等待以追上主要叢集的平均等待時間。 讀取器資料庫執行個體在處理查詢之前等待的程度取決於 aurora_replica_read_consistency 設定。
ForwardingReplicaReadWaitThroughput	每秒計數	轉寄寫入的所有工作階段中每秒處理的 SELECT 陳述式總數。
ForwardingReplicaSelectLatency	毫秒	轉送的 SELECT 延遲，平均計算監控期間內所有轉送的 SELECT 陳述式。
ForwardingReplicaSelectThroughput	每秒計數	在監控期間內平均每秒轉送的 SELECT 輸送量。

用於寫入轉送的 Aurora MySQL 狀態變數

當您在一或多個次要叢集上使用寫入轉送時，下列 Aurora MySQL 狀態變數適用於主要叢集。這些指標都是在主要叢集中寫入器資料庫執行個體上測量的。

Aurora MySQL 狀態變數	單位	描述
Aurora_fwd_master_dml_stmt_count	計數	轉送至此寫入器資料庫執行個體的 DML 陳述式總數。

Aurora MySQL 狀態變數	單位	描述
		適用於 Aurora MySQL 第 2 版。
Aurora_fwd_master_dml_stmt_duration	微秒	轉送至此寫入器資料庫執行個體的 DML 陳述式總持續時間。 適用於 Aurora MySQL 第 2 版。
Aurora_fwd_master_open_sessions	計數	寫入器資料庫執行個體上轉送的工作階段數目。 適用於 Aurora MySQL 第 2 版。
Aurora_fwd_master_select_stmt_count	計數	轉送至此寫入器資料庫執行個體的 SELECT 陳述式總數。 適用於 Aurora MySQL 第 2 版。
Aurora_fwd_master_select_stmt_duration	微秒	轉送至此寫入器資料庫執行個體的 SELECT 陳述式總持續時間。 適用於 Aurora MySQL 第 2 版。
Aurora_fwd_writer_dml_stmt_count	計數	轉送至此寫入器資料庫執行個體的 DML 陳述式總數。 適用於 Aurora MySQL 3 版。
Aurora_fwd_writer_dml_stmt_duration	微秒	轉送至此寫入器資料庫執行個體的 DML 陳述式總持續時間。
Aurora_fwd_writer_open_sessions	計數	寫入器資料庫執行個體上轉送的工作階段數目。 適用於 Aurora MySQL 3 版。

Aurora MySQL 狀態變數	單位	描述
Aurora_fwd_writer_select_stmt_count	計數	轉送至此寫入器資料庫執行個體的 SELECT 陳述式總數。適用於 Aurora MySQL 3 版。
Aurora_fwd_writer_select_stmt_duration	微秒	轉送至此寫入器資料庫執行個體的 SELECT 陳述式總持續時間。 適用於 Aurora MySQL 3 版。

下列 Aurora MySQL 狀態變數適用於每個次要叢集。這些指標的測量是在啟用寫入轉送的次要叢集中，每個讀取器資料庫執行個體上進行。

Aurora MySQL 狀態變數	單位	描述
Aurora_fwd_replica_dml_stmt_count	計數	從此讀取器資料庫執行個體轉送的 DML 陳述式總數。
Aurora_fwd_replica_dml_stmt_duration	微秒	從此讀取器資料庫執行個體轉送的所有 DML 陳述式總持續時間。
Aurora_fwd_replica_errors_session_limit	計數	主要叢集因下列其中一個錯誤狀況拒絕的工作階段數目： • 寫入器已滿 • 處理過多的轉送陳述式。
Aurora_fwd_replica_open_sessions	計數	在讀取器資料庫執行個體上使用寫入轉送的工作階段數目。
Aurora_fwd_replica_read_wait_count	計數	此讀取器資料庫執行個體上先寫後讀的等待總數。

Aurora MySQL 狀態變數	單位	描述
Aurora_fwd_replica_read_wait_duration	微秒	由於此讀取器資料庫執行個體的讀取一致性設定，而造成的等待總持續時間。
Aurora_fwd_replica_select_stmt_count	計數	從此讀取器資料庫執行個體轉送的 SELECT 陳述式總數。
Aurora_fwd_replica_select_stmt_duration	微秒	從此讀取器資料庫執行個體轉送的 SELECT 陳述式總持續時間。

在 Aurora PostgreSQL 全域資料庫中使用寫入轉送

主題

- Aurora PostgreSQL 中寫入轉送的區域和版本可用性
- 在 Aurora 中啟用寫入轉送
- 檢查 Aurora PostgreSQL 中的次要叢集是否已啟用寫入轉送
- Aurora PostgreSQL 中的應用程式和 SQL 與寫入轉送的相容性
- Aurora PostgreSQL 中寫入轉送的隔離與一致性
- 具有寫入轉送的交易存取模式
- 在 Aurora PostgreSQL 中使用寫入轉送執行多部分陳述式
- Aurora PostgreSQL 中寫入轉送的組態參數
- Aurora PostgreSQL 中寫入轉送的 Amazon CloudWatch 指標
- Aurora PostgreSQL 中寫入轉送的等待事件

Aurora PostgreSQL 中寫入轉送的區域和版本可用性

在 Aurora PostgreSQL 第 16 版和更新的主要版本中，所有次要版本都支援全域寫入轉送。對於舊版的 Aurora PostgreSQL，全域寫入轉送支援 15.4 版和更新的次要版本，以及 14.9 版和更新的次要版本。寫入轉送適用於可使用 Aurora PostgreSQL 型全域資料庫的每個 AWS 區域。

如需 Aurora PostgreSQL 全域資料庫版本和區域可用性的詳細資訊，請參閱 使用 Aurora PostgreSQL 的 Aurora 全域資料庫。

在 Aurora 中啟用寫入轉送

根據預設，當您將次要叢集新增至 Aurora 全域資料庫時，不會啟用寫入轉送。您可以在建立次要資料庫叢集時或建立後的任何時間啟用寫入轉送功能。如有需要，您可以稍後停用它。啟用或停用寫入轉送不會造成停機或重新開機。

> **Note**
>
> 您可以針對偶爾寫入的應用程式使用本機寫入轉送，並要求先寫後讀一致性，也就是能夠讀取交易中最新寫入的能力。

主控台

在主控台中，您可以在建立或修改次要資料庫叢集時啟用寫入轉送。

在建立次要資料庫叢集時啟用或停用寫入轉送

當您建立新的次要資料庫叢集時，您可以選取僅供讀取複本寫入轉送下的開啟全域寫入轉送核取方塊來啟用寫入轉送。或者清除核取方塊以停用它。若要建立次要資料庫叢集，請遵循 建立 Amazon Aurora 資料庫叢集 中的資料庫引擎指示。

下列螢幕擷取畫面顯示已選取開啟全域寫入轉送核取方塊的僅供讀取複本寫入轉送區段。



在修改次要資料庫叢集時啟用或停用寫入轉送

在主控台中，您可以修改次要資料庫叢集以啟用或停用寫入轉送。

使用主控台啟用或停用次要資料庫叢集的寫入轉送

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 選擇 Databases (資料庫)。
3. 選擇次要資料庫叢集，然後選擇修改。
4. 在僅供讀取複本寫入轉送區段中，勾選或清除開啟全域寫入轉送核取方塊。

5. 選擇繼續。
6. 在排程修改中，選擇立即套用。如果您選擇在下一個排定的維護時段套用，Aurora 會忽略此設定，並立即開啟寫入轉送。
7. 選擇修改叢集。

AWS CLI

若要使用 啟用寫入轉送 AWS CLI，請使用 `--enable-global-write-forwarding`選項。當您使用 create-db-cluster 命令建立新的次要叢集時，此選項會起作用。當您使用 modify-db-cluster 命令修改現有的次要叢集時，此選項也會起作用。其要求全域資料庫使用支援寫入轉送的 Aurora 版本。您可以使用 `--no-enable-global-write-forwarding` 選項搭配這些相同的 CLI 命令來停用寫入轉送。

下列程序說明如何使用 AWS CLI來啟用或停用全域叢集中次要資料庫叢集的寫入轉送。

為現有次要資料庫叢集啟用或停用寫入轉送

- 呼叫 modify-db-cluster AWS CLI 命令並提供下列值：
  - `--db-cluster-identifier` – 資料庫叢集的名稱。
  - `--enable-global-write-forwarding` 用來開啟，`--no-enable-global-write-forwarding` 用來關閉。

  下列範例會啟用資料庫叢集 `sample-secondary-db-cluster` 的寫入轉送。

  對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-secondary-db-cluster \
    --enable-global-write-forwarding
```

  在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-secondary-db-cluster ^
    --enable-global-write-forwarding
```

RDS API

若要使用 Amazon RDS API 啟用寫入轉送，請將 `EnableGlobalWriteForwarding` 參數設定為 `true`。當您使用 CreateDBCluster 操作建立新的次要叢集時，此參數會起作用。當您使用 ModifyDBCluster 操作修改現有的次要叢集時，此選項也會起作用。其要求全域資料庫使用支援寫入轉送的 Aurora 版本。您可以將 `EnableGlobalWriteForwarding` 參數設定為 `false` 來停用寫入轉送。

檢查 Aurora PostgreSQL 中的次要叢集是否已啟用寫入轉送

若要判斷是否可以從次要叢集使用寫入轉送，您可以檢查叢集是否具有屬性 `"GlobalWriteForwardingStatus": "enabled"`。

在 中 AWS 管理主控台，您會`Read replica write forwarding`在叢集詳細資訊頁面的組態索引標籤上看到 。若要查看所有叢集的全域寫入轉送設定狀態，請執行下列 AWS CLI 命令。

次要叢集會顯示值 `"enabled"` 或 `"disabled"`，指出寫入轉送是開啟或關閉。`null` 值表示該叢集無法使用寫入轉送。此叢集不屬於全域資料庫，或是主要叢集，而是次要叢集。如果寫入轉送處於開啟或關閉的程序中，此值也可以 `"enabling"` 是 `"disabling"`。

Example

```
aws rds describe-db-clusters --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    },
    {
        "GlobalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-2"
    },
    {
        "GlobalWriteForwardingStatus": null,
        "DBClusterIdentifier": "non-global-cluster"
    }
]
```

若要尋找已啟用全域寫入轉送的次要叢集，請執行下列命令。此命令也會傳回叢集的讀取者端點。當您使用寫入轉送從 Aurora 全域資料庫的次要叢集到主要叢集時，您可以使用次要叢集的讀取者端點。

Example

```
aws rds describe-db-clusters --query 'DBClusters[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
 | [?GlobalWriteForwardingStatus == `enabled`]'
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "ReaderEndpoint": "aurora-write-forwarding-test-replica-1.cluster-ro-
cnpexample.us-west-2.rds.amazonaws.com",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    }
]
```

Aurora PostgreSQL 中的應用程式和 SQL 與寫入轉送的相容性

當您在具有寫入轉送的全域資料庫中使用某些陳述式時，系統不允許使用這些陳述式或這些陳述式可能會產生過時的結果。此外，不支援使用者定義的函數和使用者定義的程序。因此，次要叢集的 `EnableGlobalWriteForwarding` 設定預設為關閉。在開啟此功能之前，請檢查以確定您的應用程式程式碼不受上述任何限制的影響。

您可以使用以下類型的 SQL 陳述式搭配寫入轉送：

- 資料操作語言 (DML) 陳述式，例如 `INSERT`、`DELETE` 和 `UPDATE`
- `SELECT FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE }` 陳述式
- `PREPARE` 和 `EXECUTE` 陳述式
- 包含此清單中陳述式的 `EXPLAIN` 陳述式

寫入轉送不支援下列 SQL 陳述式類型：

- 資料定義語言 (DDL) 陳述式
- `ANALYZE`
- `CLUSTER`
- `COPY`
- 游標 – 不支援游標，因此請務必在使用寫入轉送之前將其關閉。
- `GRANT|REVOKE|REASSIGN OWNED|SECURITY LABEL`
- `LOCK`

- SAVEPOINT 陳述式
- SELECT INTO
- SET CONSTRAINTS
- TRUNCATE
- VACUUM

Aurora PostgreSQL 中寫入轉送的隔離與一致性

在使用寫入轉送的工作階段中，您可以使用 REPEATABLE READ 和 READ COMMITTED 隔離層級。不過，不支援 SERIALIZABLE 隔離層級。

您可以控制次要叢集上的讀取一致性程度。讀取一致性層級會決定次要叢集在每次讀取操作之前的等待時間，以確保從主要叢集複寫部分或全部變更。您可以調整讀取一致性層級，以確保在任何後續查詢之前，您都可以在次要叢集中看見工作階段中的所有轉送寫入操作。您也可以使用此設定，確保次要叢集上的查詢永遠會看到主要叢集的最新更新。即使是由其他工作階段或其他叢集所提交的更新。若要為應用程式指定這種行為類型，請為 apg_write_forward.consistency_mode 參數選擇適當的值。該 apg_write_forward.consistency_mode 參數只會對啟用寫入轉送的次要叢集產生影響。

> **Note**
>
> 對於 apg_write_forward.consistency_mode 參數，您可以指定值 SESSION、EVENTUAL、GLOBAL 或 OFF。根據預設，系統會將此值設為 SESSION。將值設定為 OFF 會停用工作階段中的寫入轉送。

隨著一致性層級的提高，您的應用程式會花費更多時間等待 AWS 區域之間傳播變更。您可以選擇在快速回應時間之間的平衡，並確保在查詢執行之前，在其他位置所做的變更完全可用。

對於每個可用的一致性模式設定，效果如下：

- SESSION – 次要 AWS 區域中使用寫入轉送的所有查詢都會查看該工作階段中所有變更的結果。無論交易是否已遞交，這些變更都是可見的。如有必要，查詢會等待轉送寫入操作的結果複寫到目前的區域。其不會等待來自寫入操作的更新結果，這些寫入操作的執行位置位於其他區域或目前區域內其他工作階段。
- EVENTUAL – 使用寫入轉送的次要 AWS 區域中的查詢可能會看到由於複寫延遲而稍微過時的資料。在主要區域上執行寫入操作並複寫到目前的區域之前，不會顯示相同工作階段中寫入操作的結果。查

詢不會等待更新的結果變成可用。因此，它可能會擷取較舊的資料或更新的資料，視陳述式的時間和複寫延遲量而定。

- GLOBAL – 次要區域中的工作階段會 AWS 看到該工作階段所做的變更。它也會查看主要 AWS 區域和其他次要 AWS 區域的所有遞交變更。每個查詢可能會等待一段時間，長短取決於工作階段的延遲量。自查詢開始的時間起，當次要叢集與主要叢集中的所有遞交資料都是最新時，查詢就會繼續。
- OFF – 已停用工作階段中的寫入轉送。

如需寫入轉送中所含所有參數的詳細資訊，請參閱 Aurora PostgreSQL 中寫入轉送的組態參數。

具有寫入轉送的交易存取模式

如果交易存取模式設定為唯讀，則不會使用寫入轉送。您可以在連線至已啟用寫入轉送的資料庫叢集和工作階段時，將存取模式設定為僅讀取寫入。

如需交易存取模式的詳細資訊，請參閱設定交易。

在 Aurora PostgreSQL 中使用寫入轉送執行多部分陳述式

DML 陳述式可能包含多個部分，例如 `INSERT ... SELECT` 陳述式或 `DELETE ... WHERE` 陳述式。在這種情況下，系統會將整個陳述式轉送到主要叢集並在該處執行陳述式。

Aurora PostgreSQL 中寫入轉送的組態參數

Aurora 叢集參數群組包含寫入轉送功能的設定。因為這些是叢集參數，所以每個叢集中的所有資料庫執行個體都有這些變數的相同值。下表列出了有關這些參數的詳細資訊，並在表格後面附有使用注意事項。

名稱	範圍	類型	預設值	有效值
apg_write_forward.connect_timeout	Session (工作階段)	秒	30	0–2147483647
apg_write_forward.consistency_mode	工作階段 (Session)	enum	Session (工作階段)	SESSION, EVENTUAL, GLOBAL, OFF
apg_write_forward.idle_in_transaction_session_timeout	Session (工作階段)	毫秒	86400000	0–2147483647

名稱	範圍	類型	預設值	有效值
apg_write_forward.idle_session_timeout	Session (工作階段)	毫秒	300000	0–2147483647
apg_write_forward.max_forwarding_connections_percent	全域	int	25	1–100

apg_write_forward.max_forwarding_connections_percent 參數是可用於處理從讀取器轉送之查詢的資料庫連線上限。此上限的表示方式是主要叢集中寫入器資料庫執行個體的 max_connections 設定百分比。例如，如果 max_connections 是 800，且 apg_write_forward.max_forwarding_connections_percent 是 10，則寫入器允許最多 80 個同時轉送的工作階段。這些連線來自 max_connections 設定所管理的相同連線集區。當至少一個次要叢集已啟用寫入轉送時，此設定僅適用於主要叢集。

在次要叢集上使用下列設定：

- apg_write_forward.consistency_mode – 工作階段層級參數，用於控制次要叢集的讀取一致性程度。有效值為 SESSION、EVENTUAL、GLOBAL 或 OFF。根據預設，系統會將此值設為 SESSION。將值設定為 OFF 會停用工作階段中的寫入轉送。若要進一步了解一致性層級，請參閱Aurora PostgreSQL 中寫入轉送的隔離與一致性。此參數僅在啟用寫入轉送之次要叢集的讀取器執行個體中，且該次要叢集位於 Aurora 全域資料庫中才有意義。
- apg_write_forward.connect_timeout – 次要叢集在放棄與主要叢集建立連線之前等待的秒數上限。值 0 表示無限期等待。
- apg_write_forward.idle_in_transaction_session_timeout – 主要叢集在關閉連線之前，等待從具有開啟交易之次要叢集轉送的連線上活動的毫秒數。如果在此期間之後工作階段在交易中仍處於閒置狀態，則 Aurora 會終止工作階段。值為 0 會停用逾時。
- apg_write_forward.idle_session_timeout – 主要叢集在關閉從次要叢集轉送的連線前等待活動的毫秒數。如果工作階段在此期間之後仍處於閒置狀態，則 Aurora 會終止工作階段。值為 0 會停用逾時。

Aurora PostgreSQL 中寫入轉送的 Amazon CloudWatch 指標

當您在一或多個次要叢集上使用寫入轉送時，下列 Amazon CloudWatch 指標適用於主要叢集。這些指標都是在主要叢集中寫入器資料庫執行個體上測量的。

CloudWatch 指標	單位和描述
AuroraForwardingWriterDMLThroughput	計數 (每秒) 此寫入器資料庫執行個體每秒處理的轉送 DML 陳述式數目。
AuroraForwardingWriterOpenSessions	計數。此寫入器資料庫執行個體處理轉送查詢的開啟工作階段數目。
AuroraForwardingWriterTotalSessions	計數。此寫入器資料庫執行個體上轉送的工作階段總數。

下列 CloudWatch 指標適用於每個次要叢集。這些指標的測量是在啟用寫入轉送的次要叢集中，每個讀取器資料庫執行個體上進行。

CloudWatch 指標	單位和描述
AuroraForwardingReplicaCommitThroughput	計數 (每秒) 此複本每秒轉送之工作階段中的遞交數目。
AuroraForwardingReplicaDMLLatency	毫秒。複本上轉送 DML 的平均回應時間 (以毫秒為單位)。
AuroraForwardingReplicaDMLThroughput	計數 (每秒) 每秒在此複本上處理的轉送 DML 陳述式數目。
AuroraForwardingReplicaErrorSessionsLimit	計數。由於達到連線數上限或寫入轉送連線數上限而被主要叢集拒絕的工作階段數目。
AuroraForwardingReplicaOpenSessions	計數。在複本執行個體上使用寫入轉送的工作階段數目。
AuroraForwardingReplicaReadWaitLatency	毫秒。複本等待與主要叢集的 LSN 保持一致的平均等待時間 (毫秒)。讀取器資料庫執行個體等待的程度取決於 apg_write_forward.consistency_mode 設定。如需有關此設定的詳細資訊，請參閱 the section called “Aurora PostgreSQL 中寫入轉送的組態參數”。

Aurora PostgreSQL 中寫入轉送的等待事件

當您搭配 Aurora PostgreSQL 使用寫入轉送時，Amazon Aurora 會產生下列等待事件。

主題

- IPC:AuroraWriteForwardConnect
- IPC:AuroraWriteForwardConsistencyPoint
- IPC:AuroraWriteForwardExecute
- IPC:AuroraWriteForwardGetGlobalConsistencyPoint
- IPC:AuroraWriteForwardXactAbort
- IPC:AuroraWriteForwardXactCommit
- IPC:AuroraWriteForwardXactStart

IPC:AuroraWriteForwardConnect

次要資料庫叢集上的後端程序正在等待開啟與主要資料庫叢集之寫入器節點的連線時，就會發生 `IPC:AuroraWriteForwardConnect` 事件。

等待時間增加的可能原因

隨著從次要區域的讀取器節點到主要資料庫叢集之寫入器節點的連線嘗試次數增加，此事件也會增加。

動作

減少從次要節點到主要區域的寫入器節點的同時連線數目。

IPC:AuroraWriteForwardConsistencyPoint

`IPC:AuroraWriteForwardConsistencyPoint` 事件描述來自次要資料庫叢集上節點的查詢將等待轉送的寫入操作的結果複寫到目前區域的時間。只有在工作階段層級參數 `apg_write_forward.consistency_mode` 設定為下列其中一項時，才會產生此事件：

- `SESSION` – 次要節點上的查詢會等待該工作階段中所做之所有變更的結果。
- `GLOBAL` – 次要節點上的查詢會等待該工作階段所做的變更結果，以及來自全域叢集中主要區域和其他次要區域的所有已遞交變更。

如需有關 `apg_write_forward.consistency_mode` 參數設定的資訊，請參閱 the section called “Aurora PostgreSQL 中寫入轉送的組態參數”。

等待時間增加的可能原因

等待時間較長的常見原因如下：

- 根據 Amazon CloudWatch ReplicaLag 指標所衡量的複本延遲增加。如需此指標的詳細資訊，請參閱 監控 Aurora PostgreSQL 複寫。
- 主要區域寫入器節點或次要節點上的負載增加。

動作

請根據應用程式的需求變更一致性模式。

IPC:AuroraWriteForwardExecute

當次要資料庫叢集上的後端程序正在等待轉送的查詢完成，並從主要資料庫叢集的寫入器節點取得結果時，就會發生 IPC:AuroraWriteForwardExecute 事件。

等待時間增加的可能原因

等待時間增加的常見原因包括：

- 從主要區域的寫入器節點擷取大量的行。
- 次要節點與主要區域寫入器節點之間的網路延遲增加，會增加次要節點從寫入器節點接收資料所花費的時間。
- 次要節點上的負載增加，可能會延遲從次要節點到主要區域寫入器節點之查詢要求的傳輸。
- 主要區域寫入器節點的負載增加，可能會延遲將資料從寫入器節點傳輸至次要節點。

動作

根據等待事件的原因，我們會建議不同的動作。

- 最佳化查詢以僅擷取必要的資料。
- 將資料處理語言 (DML) 操作最佳化，以僅修改必要的資料。
- 如果次要節點或主要區域的寫入器節點受到 CPU 或網路頻寬的限制，請考慮將其變更為具有更多 CPU 容量或更多網路頻寬的執行個體類型。

IPC:AuroraWriteForwardGetGlobalConsistencyPoint

當次要資料庫叢集上使用全域一致性模式的後端程序在執行查詢之前等待從寫入器節點取得全域一致性點時，就會發生 IPC:AuroraWriteForwardGetGlobalConsistencyPoint 事件。

等待時間增加的可能原因

等待時間增加的常見原因包括：

- 次要節點與主要區域寫入器節點之間的網路延遲增加，會增加次要節點從寫入器節點接收資料所花費的時間。
- 次要節點上的負載增加，可能會延遲從次要節點到主要區域寫入器節點之查詢要求的傳輸。
- 主要區域寫入器節點的負載增加，可能會延遲將資料從寫入器節點傳輸至次要節點。

動作

根據等待事件的原因，我們會建議不同的動作。

- 請根據應用程式的需求變更一致性模式。
- 如果次要節點或主要區域的寫入器節點受到 CPU 或網路頻寬的限制，請考慮將其變更為具有更多 CPU 容量或更多網路頻寬的執行個體類型。

IPC:AuroraWriteForwardXactAbort

次要資料庫叢集上的後端程序正在等待遠端清除查詢的結果時，就會發生 IPC:AuroraWriteForwardXactAbort 事件。在中止寫入轉送的交易之後，會發出清除查詢以將程序返回到適當的狀態。Amazon Aurora 執行這些操作可能是因為發現錯誤，也可能是因為使用者發出了明確的 ABORT 命令或取消了正在執行的查詢。

等待時間增加的可能原因

等待時間增加的常見原因包括：

- 次要節點與主要區域寫入器節點之間的網路延遲增加，會增加次要節點從寫入器節點接收資料所花費的時間。
- 次要節點上的負載增加，可能會延遲從次要節點到主要區域寫入器節點之清除查詢要求的傳輸。
- 主要區域寫入器節點的負載增加，可能會延遲將資料從寫入器節點傳輸至次要節點。

動作

根據等待事件的原因，我們會建議不同的動作。

- 調查中止交易的原因。
- 如果次要節點或主要區域的寫入器節點受到 CPU 或網路頻寬的限制，請考慮將其變更為具有更多 CPU 容量或更多網路頻寬的執行個體類型。

IPC:AuroraWriteForwardXactCommit

當次要資料庫叢集上的後端程序正在等待轉送的 commit 交易命令的結果時，就會發生 `IPC:AuroraWriteForwardXactCommit` 事件。

等待時間增加的可能原因

等待時間增加的常見原因包括：

- 次要節點與主要區域寫入器節點之間的網路延遲增加，會增加次要節點從寫入器節點接收資料所花費的時間。
- 次要節點上的負載增加，可能會延遲從次要節點到主要區域寫入器節點之查詢要求的傳輸。
- 主要區域寫入器節點的負載增加，可能會延遲將資料從寫入器節點傳輸至次要節點。

動作

如果次要節點或主要區域的寫入器節點受到 CPU 或網路頻寬的限制，請考慮將其變更為具有更多 CPU 容量或更多網路頻寬的執行個體類型。

IPC:AuroraWriteForwardXactStart

當次要資料庫叢集上的後端程序正在等待轉送的 start 交易命令的結果時，就會發生 `IPC:AuroraWriteForwardXactStart` 事件。

等待時間增加的可能原因

等待時間增加的常見原因包括：

- 次要節點與主要區域寫入器節點之間的網路延遲增加，會增加次要節點從寫入器節點接收資料所花費的時間。
- 次要節點上的負載增加，可能會延遲從次要節點到主要區域寫入器節點之查詢要求的傳輸。
- 主要區域寫入器節點的負載增加，可能會延遲將資料從寫入器節點傳輸至次要節點。

動作

如果次要節點或主要區域的寫入器節點受到 CPU 或網路頻寬的限制，請考慮將其變更為具有更多 CPU 容量或更多網路頻寬的執行個體類型。

在 Amazon Aurora 全球資料庫中使用切換或容錯移轉

Aurora 全球資料庫功能提供的業務持續性和災難復原 (BCDR) 保護，比 Aurora 資料庫叢集在單一 AWS 區域 中提供的標準高可用性更強大。您可以使用 Aurora 全球資料庫擬訂計劃，以迅速從罕見、意外的區域性災難或服務完全中斷的情況復原。

您可以使用 Aurora 全球資料庫功能，參閱下列指引和程序來規劃、測試和實作 BCDR 策略。

主題

- 規劃業務持續性和災難復原
- 針對 Amazon Aurora 全球資料庫執行轉換
- 從計劃外中斷復原 Amazon Aurora 全域資料庫
- 管理 Aurora PostgreSQL – 全域資料庫的 RPO
- 全球資料庫次要叢集的跨區域彈性

規劃業務持續性和災難復原

為了規劃您的業務持續性和災難復原策略，了解下列產業術語以及這些術語與 Aurora 全球資料庫功能的關係會很有幫助。

災難復原通常是由以下兩個業務目標推動：

- 復原時間點目標 (RTO) - 系統在災難發生或服務中斷後傳回工作狀態所需的時間。換言之，RTO 會測量停機時間。對於 Aurora 全球資料庫，RTO 可以按分鐘來排序。
- 復原點目標 (RPO) - 在災難發生或服務中斷後可能遺失的資料量 (以時間為單位)。此資料遺失通常是因為非同步複寫延遲所致。對於 Aurora 全域資料庫，RPO 通常以秒為單位進行測量。使用 Aurora PostgreSQL– 全域資料庫，您可以使用 `rds.global_db_rpo` 參數來設定和追蹤 RPO 上限，但這麼做可能會影響主要叢集寫入器節點上的交易處理。如需更多詳細資訊，請參閱 管理 Aurora PostgreSQL – 全域資料庫的 RPO。

使用 Aurora 全球資料庫執行切換或容錯移轉需要將次要資料庫叢集提升為主要資料庫叢集。「區域中斷」一詞通常用來描述各種故障情況。最壞的情況可能是影響數百平方英里的災難性事件導致大規模中

斷。不過，大多數中斷都較侷限在本地，只會影響一小部分的雲端服務或客戶系統。請考量完整的中斷範圍，以確定跨區域容錯移轉是適當的解決方案，並選擇適當的容錯移轉方法來因應此情況。應該使用容錯移轉或轉換方法，取決於特定中斷案例：

- 容錯移轉 - 使用此方法從意外中斷的情況復原。使用此方法時，您會以 Aurora 全球資料庫中的次要資料庫叢集之一為目標，進行跨區域容錯移轉。此方法的 RPO 通常是以秒為單位測量的非零值。資料遺失量取決於發生故障時，整個 AWS 區域 的 Aurora 全球資料庫複寫延遲。如需詳細資訊，請參閱 從計劃外中斷復原 Amazon Aurora 全域資料庫。
- 切換 - 此操作的舊稱為「受管計劃容錯移轉」。將此方法用於受控案例，例如操作維護和其他計劃的操作程序，其中與其互動的所有 Aurora 叢集和其他服務都處於良好狀態。由於此功能會將次要資料庫叢集與主要資料庫叢集同步處理，再做出任何其他變更，因此 RPO 為 0 (不會遺失資料)。如需詳細資訊，請參閱 針對 Amazon Aurora 全球資料庫執行轉換。

> **Note**
>
> 在可以切換或容錯移轉到無周邊次要 Aurora 資料庫叢集之前，您必須先在其中新增資料庫執行個體。如需無周邊資料庫叢集的詳細資訊，請參閱 在次要區域中建立無周邊 Aurora 資料庫叢集。

針對 Amazon Aurora 全球資料庫執行轉換

> **Note**
>
> 切換的舊稱為受管計劃容錯移轉。

您可以使用切換來例行變更主要叢集的區域。此方法適用於受控情況，例如操作維護及其他計劃內操作程序。

有三種常見的轉換使用案例。

- 對於特定行業強制實施的「區域輪換」需求。例如，金融服務法規可能希望第 0 層系統切換到不同的區域數個月時間，以確保定期演練災難復原程序。
- 對於多區域「全天候」應用程式。例如，某家公司可能希望根據不同時區的營業時間，在不同區域提供較低的延遲寫入。
- 可作為在容錯移轉後，容錯回復至原始主要區域的零資料遺失方法。

> **Note**
>
> 切換旨在用於 Aurora 全球資料庫，其中與其互動的所有 Aurora 叢集和其他服務都處於良好狀態。若要從意外中斷復原，請依照 從計劃外中斷復原 Amazon Aurora 全域資料庫 中適當的程序進行。
>
> 只有在主要和次要資料庫叢集具有相同的主要和次要引擎版本時，您才能在 Aurora 全球資料庫上執行受管跨區域切換。根據引擎和引擎版本，修補層級可能需要相同，或者修補層級可以不同。如需允許在具有不同修補層級的主要和次要叢集之間進行這些操作的引擎和引擎版本清單，請參閱 受管跨區域轉換和容錯移轉的修補程式等級相容性。開始轉換之前，請先檢查全域叢集中的引擎版本，以確定它們可支援受管跨區域轉換，並視需要進行升級。

在切換期間，Aurora 會讓所選次要區域中的叢集成為您的主要叢集。切換機制會維護您全球資料庫的現有複寫拓撲：它在相同區域中仍有相同數量的 Aurora 叢集。在 Aurora 開始切換程序之前，它會等待目標次要區域叢集與主要區域叢集完全同步。然後，主要區域中的資料庫叢集會變成唯讀。選擇的次要叢集會將其中一個唯讀節點升級為完整寫入器狀態，這會允許該叢集擔任主要叢集的角色。由於目標次要叢集都在處理程序開始時與主要叢集同步，因此新的主要叢集會繼續執行 Aurora 全球資料庫的操作，而不會遺失任何資料。您的資料庫在短時間內無法使用，因為同時間主要叢集和選取的次要叢集會承擔其新角色。

> **Note**
>
> 若要在執行切換後管理 Aurora PostgreSQL 的複寫槽，請參閱 管理 Aurora PostgreSQL 的邏輯槽。

若要最佳化應用程式可用性，建議您在使用此功能之前先執行下列動作：

- 在非尖峰時段或在寫入主要資料庫叢集最少時，執行此操作。
- 檢查 Aurora 全域資料庫中所有次要 Aurora 資料庫叢集的延遲時間。對於所有 Aurora PostgreSQL 型全球資料庫及 Aurora MySQL 型全球資料庫，從 3.04.0 及更高版本或 2.12.0 及更高版本引擎開始，請使用 Amazon CloudWatch 檢視所有次要資料庫叢集的 `AuroraGlobalDBRPOLag` 指標。對於較低的次要版本 Aurora MySQL 型全球資料庫，請改為檢視 `AuroraGlobalDBReplicationLag` 指標。這些指標顯示複寫至次要資料庫叢集與複寫至主要資料庫叢集之間的差距 (以毫秒為單位)。此值與 Aurora 完成轉換所需的時間成正比。因此，延遲值越大，轉換所需的時間越長。當您檢查這些指標時，請從目前的主要叢集執行此操作。

  如需 Aurora 的 CloudWatch 指標相關詳細資訊，請參閱Amazon Aurora 的叢集層級指標。

- 在切換期間提升的次要資料庫叢集可能具有與舊主要資料庫叢集不同的組態設定。我們建議您在 Aurora 全球資料庫叢集中的所有叢集中保持以下類型的組態設定一致。這樣做有助於將切換後的效能問題、工作負載不相容和其他異常行為降至最低。
  - 視需為新的主要資料庫叢集設定 Aurora 資料庫叢集參數群組 – 因此，當您提升次要資料庫叢集以接管主要角色時，其參數群組可能會設定與主要資料庫叢集不同的參數群組。如果是這樣，請修改提升的次要資料庫叢集的參數群組，以符合主要叢集的設定。若要瞭解如何操作，請參閱修改 Aurora 全域資料庫的參數。
  - 設定監控工具和選項，例如 Amazon CloudWatch Events 和警示 – 根據全域資料庫的需要，為提升的資料庫叢集設定相同的日誌記錄功能、警示等。與參數群組一樣，在轉換程序期間，這些功能的組態不會從主要叢集繼承。某些 CloudWatch 指標 (例如複寫延遲) 僅適用於次要區域。因此，轉換會改變檢視這些指標並對其設定警示的方式，而且可能需要變更任何預先定義的儀表板。如需 Aurora 資料庫叢集和監控的詳細資訊，請參閱 使用 Amazon CloudWatch 監控 Amazon Aurora 指標。
  - 設定與其他 AWS 服務的整合 - 如果您的 Aurora 全球資料庫與 AWS Secrets Manager、AWS Identity and Access Management、Amazon S3 和 AWS Lambda 等 AWS 服務整合，則務必視需要設定與這些服務的整合。如需整合 Aurora 全域資料庫與 IAM、Amazon S3 和 Lambda 的詳細資訊，請參閱 將 Amazon Aurora 全域資料庫與其他 AWS 服務搭配使用。若要進一步了解 Secrets Manager，請參閱如何在各 AWS 區域 的 AWS Secrets Manager 中自動複寫密碼。

如果您使用的是 Aurora 全球資料庫寫入器端點，則不需要變更應用程式中的連線設定。確認 DNS 變更已傳播，而且您可以在新的主要叢集上連線和執行寫入操作。然後，您可以繼續應用程式的完整操作。

假設您的應用程式連線使用舊主要叢集的叢集端點，而不是全域寫入器端點。在這種情況下，請務必變更您的應用程式連線設定，以使用新主要叢集的叢集端點。如果您在建立 Aurora 全域資料庫時接受提供的名稱，則可透過從應用程式中提升叢集的端點字串移除 `-ro` 來變更端點。例如，當該叢集提升為主要叢集時，次要叢集的端點 `my-global.cluster-ro-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` 會變成 `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com`。

如果您使用 RDS 代理，請務必將應用程式的寫入操作重新導向至與新主要叢集相關聯之代理的適當讀取/寫入端點。此代理端點可能是預設端點或自訂讀取/寫入端點。如需更多資訊，請參閱RDS Proxy 端點如何使用全域資料庫。

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 來執行 Aurora 全球資料庫切換。

主控台

若要在 Aurora 全球資料庫上執行轉換

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 選擇資料庫，然後尋找您要執行切換的 Aurora 全球資料庫。
3. 從動作功能表選擇轉換或容錯移轉全球資料庫。



4. 選擇轉換。



5. 對於新的主要叢集，選擇其中一個次要 AWS 區域 中的使用中叢集，使其成為新的主要叢集。
6. 選擇確認。

轉換完成後，您可以在資料庫清單中看到 Aurora 資料庫叢集及其目前角色，如下圖所示。



AWS CLI

若要在 Aurora 全球資料庫上執行轉換

使用 `switchover-global-cluster` CLI 命令執行 Aurora 全球資料庫的切換。使用命令，傳遞下列參數的值。

- `--region` – 指定執行 Aurora 全域資料庫之主要資料庫叢集的 AWS 區域。
- `--global-cluster-identifier` – 指定 Aurora 全域資料庫的名稱。
- `--target-db-cluster-identifier` – 指定要提升為 Aurora 全域資料庫之 Aurora 主要資料庫叢集的 Amazon Resource Name (ARN)。

對於 Linux、macOS 或 Unix：

```
aws rds --region region_of_primary \
   switchover-global-cluster --global-cluster-identifier global_database_id \
  --target-db-cluster-identifier arn_of_secondary_to_promote
```

在 Windows 中：

```
aws rds --region region_of_primary ^
   switchover-global-cluster --global-cluster-identifier global_database_id ^
  --target-db-cluster-identifier arn_of_secondary_to_promote
```

RDS API

若要執行 Aurora 全球資料庫的切換，請執行 SwitchoverGlobalCluster API 操作。

從計劃外中斷復原 Amazon Aurora 全域資料庫

在罕見情況下，您的 Aurora 全球資料庫可能會在其主要 AWS 區域 發生非預期中斷。如果發生此情況，您的主要 Aurora 資料庫叢集及其寫入器節點將無法使用，而主要和次要資料庫叢集之間的複寫也會停止。若要將停機時間 (RTO) 和資料遺失 (RPO) 減至最低，您可以快速執行跨區域容錯移轉。

Aurora 全球資料庫有兩種容錯移轉方法，可用於災難復原情況：

- 受管容錯移轉 - 建議使用此方法進行災難復原。使用此方法時，當舊的主要區域再次可用時，Aurora 會自動將它新增回全球資料庫做為次要區域。如此便能維持全域叢集的原始拓撲。若要了解如何使用此方法，請參閱 針對 Aurora 全球資料庫執行受管容錯移轉。
- 手動容錯移轉 - 當無法選擇受管容錯移轉時，可以使用此替代方法，例如，當主要和次要區域執行不相容的引擎版本時。若要了解如何使用此方法，請參閱 針對 Aurora 全球資料庫執行手動容錯移轉。

> **Important**
>
> 這兩種容錯移轉方法都可能導致未在容錯移轉事件發生之前複寫至所選次要叢集的寫入交易資料遺失。不過，當復原程序將所選次要資料庫叢集上的資料庫執行個體提升為主要主要寫入器資料庫執行個體時，就能保證資料在交易上處於一致狀態。容錯移轉也容易發生核心分裂問題。

針對 Aurora 全球資料庫執行受管容錯移轉

此方法的目的是在發生實際區域災難或服務完全中斷的情況下，提供業務持續性。

在受管容錯移轉期間，所選次要區域中的次要叢集會成為新的主要叢集。選擇的次要叢集會將其中一個唯讀節點提升為完整寫入器狀態。此步驟可讓叢集擔任主要叢集的角色。當此叢集擔任這個新角色時，您的資料庫在短時間內無法使用。一旦該舊的主要區域正常運作且再次可用，Aurora 就會自動將它新增回全域叢集作為次要區域。因此，系統會維護 Aurora 全球資料庫的現有複寫拓撲。

> **Note**
>
> 若要在執行容錯移轉後管理 Aurora PostgreSQL 的複寫槽，請參閱 管理 Aurora PostgreSQL 的邏輯槽。

> **Note**
>
> 只有在主要和次要資料庫叢集具有相同的主要和次要引擎版本時，您才能在 Aurora 全球資料庫上執行受管跨區域容錯移轉。根據引擎和引擎版本，修補層級可能需要相同，或者修補層級可以不同。如需允許在具有不同修補層級的主要和次要叢集之間進行這些操作的引擎和引擎版本清單，請參閱 受管跨區域轉換和容錯移轉的修補程式等級相容性。開始容錯移轉之前，請先檢查全域叢集中的引擎版本，以確定它們可支援受管跨區域切換，並視需要進行升級。如果您的引擎版本需要相同的修補程式層級，但正在執行不同的修補程式層級，您可以依照 針對 Aurora 全球資料庫執行手動容錯移轉 中的步驟手動執行容錯移轉。

受管容錯移轉不會等待所選次要區域與目前主要區域之間的資料同步。由於 Aurora 全球資料庫會以非同步方式複寫資料，因此在提升至接受完整讀取/寫入功能之前，可能並非所有交易都會複寫至所選的次要 AWS 區域。

為了確保資料處於一致狀態，Aurora 會在復原後為舊的主要區域建立新的儲存磁碟區。在 AWS 區域中建立新的儲存磁碟區之前，Aurora 會嘗試在故障點拍攝舊儲存磁碟區的快照。如此一來，您就可以還原快照，並從中復原任何遺失的資料。如果此操作成功，Aurora 會將名為 `rds:unplanned-global-failover-`*`name-of-old-primary-DB-cluster-timestamp`* 的此快照放置在 AWS 管理主控台 的快照區段中。您也可以使用 `describe-db-cluster-snapshots` AWS CLI 命令或 `DescribeDBClusterSnapshots` API 操作來查看快照的詳細資訊。

當您啟動受管容錯移轉時，Aurora 也會嘗試透過高可用性的 Aurora 儲存層停止寫入流量。我們將此機制稱為「寫入隔離」。如果程序成功，Aurora 會發出 RDS 事件，讓您知道寫入已停止。萬一某個區域發生多個 AZ 故障，寫入隔離程序可能無法及時成功。在這種情況下，Aurora 會發出 RDS 事件，通知您停止寫入的程序已逾時。如果可以在網路上存取舊的主要叢集，Aurora 會在該處記錄這些事件。如果沒有，Aurora 會在新的主要叢集上記錄事件。若要進一步了解這些事件，請參閱 資料庫叢集事件。由於隔離寫入是最佳嘗試，因此可能會在舊的主要區域中暫時接受寫入，從而導致核心分裂問題。

建議您先完成下列任務，再使用 Aurora 全球資料庫執行容錯移轉。這樣做可將核心分裂問題的可能性降至最低，或從舊主要叢集的快照中復原未複寫的資料。

- 若要防止寫入傳送至 Aurora 全球資料庫的主要叢集，請讓應用程式離線。
- 請確定連線至主要資料庫叢集的任何應用程式都使用全域寫入器端點。即使新的區域因為切換或容錯移轉而成為主要叢集，此端點的值也會保持不變。Aurora 會實作額外的保護措施，將透過全域端點提交的寫入操作遺失資料的可能性降至最低。如需全域寫入器端點的詳細資訊，請參閱 連線至 Amazon Aurora 全域資料庫。

- 如果您使用的是全域寫入器端點，且應用程式或聯網層快取 DNS 值，請將 DNS 快取的存留時間 (TTL) 降低為 5 秒等低值。如此一來，您的應用程式就能快速向全域寫入器端點註冊 DNS 變更。雖然 Aurora 嘗試封鎖舊主要區域中的寫入，但動作不保證成功。縮短 DNS 快取持續時間可進一步降低核心分裂問題的可能性。或者，您可以檢查 RDS 事件，在 Aurora 觀察到全域寫入器端點的 DNS 變更時通知您。如此一來，您可以在重新開始應用程式寫入流量之前，驗證您的應用程式是否也已註冊 DNS 變更。
- 檢查 Aurora 全球資料庫中所有次要 Aurora 資料庫叢集的延遲時間。選擇複寫延遲最低的次要區域，即可將目前失敗的主要區域的資料遺失降到最低。

  對於所有版本的 Aurora PostgreSQL 型全球資料庫及 Aurora MySQL 型全球資料庫，從 3.04.0 及更高版本或 2.12.0 及更高版本引擎開始，請使用 Amazon CloudWatch 檢視所有次要資料庫叢集的 `AuroraGlobalDBRPOLag` 指標。對於較低的次要版本 Aurora MySQL 型全球資料庫，請改為檢視 `AuroraGlobalDBReplicationLag` 指標。這些指標顯示複寫至次要資料庫叢集與複寫至主要資料庫叢集之間的差距 (以毫秒為單位)。

  如需 Aurora 的 CloudWatch 指標相關詳細資訊，請參閱Amazon Aurora 的叢集層級指標。

在受管容錯移轉期間，選擇的次要資料庫叢集會提升為作為主要資料庫叢集的新角色。但是，它不會繼承主要資料庫叢集的各種組態選項。組態不相符可能會導致效能問題、工作負載不相容，以及其他異常行為。若要避免此類問題，建議您針對下列各項解決 Aurora 全域資料庫叢集之間的差異：

- 視需為新的主要資料庫叢集設定 Aurora 資料庫叢集參數群組 – 您可以為 Aurora 全域資料庫中的每個 Aurora 叢集獨立設定 Aurora 資料庫叢集參數群組。因此，當您提升次要資料庫叢集以接管主要角色時，其參數群組可能會設定與主要資料庫叢集不同的參數群組。如果是這樣，請修改提升的次要資料庫叢集的參數群組，以符合主要叢集的設定。若要瞭解如何操作，請參閱修改 Aurora 全域資料庫的參數。
- 設定監控工具和選項，例如 Amazon CloudWatch Events 和警示 – 根據全域資料庫的需要，為提升的資料庫叢集設定相同的日誌記錄功能、警示等。與參數群組一樣，在容錯移轉程序期間，這些功能的組態不會從主要叢集繼承。某些 CloudWatch 指標 (例如複寫延遲) 僅適用於次要區域。因此，容錯移轉會改變檢視這些指標並對其設定警示的方式，而且可能需要變更任何預先定義的儀表板。如需監控 Aurora 資料庫叢集的詳細資訊，請參閱 使用 Amazon CloudWatch 監控 Amazon Aurora 指標。
- 設定與其他 AWS 服務的整合 – 如果您的 Aurora 全球資料庫與 AWS Secrets Manager、AWS Identity and Access Management、Amazon S3 和 AWS Lambda 等其他 AWS 服務整合，您需要確定這些服務完成必要設定，才能從任何次要區域存取。如需整合 Aurora 全域資料庫與 IAM、Amazon S3 和 Lambda 的詳細資訊，請參閱 將 Amazon Aurora 全域資料庫與其他 AWS

服務搭配使用。若要進一步了解 Secrets Manager，請參閱如何在各 AWS 區域 的 AWS Secrets Manager 中自動複寫密碼。

通常，選擇的次要叢集會在幾分鐘內擔任主要角色。一旦新的主要區域的寫入器資料庫執行個體可用，您就可以將應用程式連線到該節點並恢復工作負載。Aurora 提升新的主要叢集之後，會自動重建所有其他次要區域叢集。

由於 Aurora 全球資料庫使用非同步複寫，因此每個次要區域的複寫延遲可能有所不同。Aurora 會重建這些次要區域，使其具有與新主要區域叢集完全相同的時間點資料。整個重建工作的期間可能持續幾分鐘到數小時，取決於儲存磁碟區的大小和區域之間的距離。當次要區域叢集從新的主要區域完成重建時，便可供讀取存取使用。

一旦新的主要寫入器進行提升且可用，新的主要區域的叢集就可以處理 Aurora 全球資料庫的讀取和寫入操作。

如果您使用的是全域端點，則不需要變更應用程式中的連線設定。確認 DNS 變更已傳播，而且您可以在新的主要叢集上連線和執行寫入操作。然後，您可以繼續應用程式的完整操作。

如果您未使用全域端點，請務必變更應用程式的端點，以使用新提升的主要資料庫叢集的叢集端點。如果您在建立 Aurora 全域資料庫時接受提供的名稱，則可透過從應用程式中提升叢集的端點字串移除 `-ro` 來變更端點。

例如，當該叢集提升為主要叢集時，次要叢集的端點 `my-global.cluster-**ro**-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` 會變成 `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com`。

如果您使用 RDS 代理，請務必將應用程式的寫入操作重新導向至與新主要叢集相關聯之代理的適當讀取/寫入端點。此代理端點可能是預設端點或自訂讀取/寫入端點。如需更多資訊，請參閱RDS Proxy 端點如何使用全域資料庫。

為了還原全球資料庫叢集的原始拓撲，Aurora 會監控舊主要區域的可用性。一旦該區域正常運作且再次可用，Aurora 就會自動將它新增回全域叢集作為次要區域。在舊的主要區域中建立新的儲存磁碟區之前，Aurora 會嘗試在故障點拍攝舊儲存磁碟區的快照。這樣做是為了讓您用它來復原任何遺失的資料。如果此操作成功，Aurora 會建立名為 `rds:unplanned-global-failover-*name-of-old-primary-DB-cluster-timestamp*` 的快照。您可以在 AWS 管理主控台 的快照區段中找到此快照。您也會在 DescribeDBClusterSnapshots API 操作傳回的資訊中看到此快照。

> **Note**
>
> 舊儲存磁碟區的快照是系統快照，會受到舊的主要叢集上設定的備份保留期限的限制。若要在保留期間之外保留此快照，您可以複製該快照並將它儲存為手動快照。若要進一步了解如何複製快照，包括定價在內，請參閱 資料庫叢集快照複製。

原始拓撲還原之後，您可以在最適合您的業務和工作負載時執行轉換操作，將全球資料庫容錯回復至原始主要區域。若要啟用，請依照「針對 Amazon Aurora 全球資料庫執行轉換」中的步驟進行。

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API，來執行 Aurora 全球資料庫的容錯移轉。

主控台

若要在 Aurora 全球資料庫上執行受管容錯移轉

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 選擇資料庫，然後尋找您要執行容錯移轉的 Aurora 全球資料庫。
3. 從動作功能表選擇轉換或容錯移轉全球資料庫。



4. 選擇容錯移轉 (允許資料遺失)。



5. 對於新的主要叢集，選擇其中一個次要 AWS 區域 中的使用中叢集，使其成為新的主要叢集。
6. 輸入 **`confirm`**，然後選擇確認。

容錯移轉完成後，您可以在資料庫清單中檢視 Aurora 資料庫叢集及其目前狀態，如下圖所示。



AWS CLI

若要在 Aurora 全球資料庫上執行受管容錯移轉

使用 `failover-global-cluster` CLI 命令執行 Aurora 全球資料庫的容錯移轉。使用命令，傳遞下列參數的值。

- `--region` - 指定 AWS 區域，要成為 Aurora 全球資料庫的新主要資料庫叢集的次要資料庫叢集在此執行。
- `--global-cluster-identifier` – 指定 Aurora 全域資料庫的名稱。
- `--target-db-cluster-identifier` - 指定要提升為 Aurora 全球資料庫的新 Aurora 主要資料庫叢集的 Amazon Resource Name (ARN)。
- `--allow-data-loss` - 明確使其成為容錯移轉操作，而不是轉換操作。如果非同步複寫元件尚未完成將所有複寫的資料傳送至次要區域，容錯移轉操作就可能導致部分資料遺失。

對於 Linux、macOS 或 Unix：

```
aws rds --region region_of_selected_secondary \
   failover-global-cluster --global-cluster-identifier global_database_id \
   --target-db-cluster-identifier arn_of_secondary_to_promote \
   --allow-data-loss
```

在 Windows 中：

```
aws rds --region region_of_selected_secondary ^
   failover-global-cluster --global-cluster-identifier global_database_id ^
   --target-db-cluster-identifier arn_of_secondary_to_promote ^
   --allow-data-loss
```

RDS API

若要使用 Aurora 全球資料庫執行容錯移轉，請執行 FailoverGlobalCluster API 操作。

針對 Aurora 全球資料庫執行手動容錯移轉

在某些情況下，您可能無法使用受管容錯移轉程序。舉例來說，如果您的主要和次要資料庫叢集未執行相容的引擎版本，就是這種情況。在這種情況下，您可以依照此手動程序執行容錯移轉至目標次要區域。

> **Tip**
>
> 我們建議您在使用此程序之前先了解該程序。準備計劃，以便在出現第一個全區域問題跡象時快速執行。您可以使用 Amazon CloudWatch 定期追蹤次要叢集的延遲時間，藉此找出最低複寫延遲的次要區域。請務必測試您的計畫，以檢查程序是否完整且準確，以及員工在真正發生容錯移轉之前接受過災難復原容錯移轉的培訓。

在主要區域發生計劃外中斷後執行手動容錯移轉至次要叢集

1. 停止發出 DML 陳述式和其他寫入操作給發生中斷 AWS 區域 中的主要 Aurora 資料庫叢集。
2. 從次要 AWS 區域 識別要用作新的主要資料庫叢集的 Aurora 資料庫叢集。如果您的 Aurora 全球資料庫中有兩個 (或更多) 次要 AWS 區域，請選擇具有最低複寫延遲的次要叢集。
3. 從 Aurora 全域資料庫中分離您選擇的次要資料庫叢集。

   若從 Aurora 全域資料庫移除次要資料庫叢集，會立即停止從主要資料庫叢集複寫到該次要資料庫叢集，並將其提升為具有完整讀取/寫入功能的獨立佈建 Aurora 資料庫叢集。與發生中斷區域的主要叢集關聯的任何其他次要 Aurora 資料庫叢集仍然可用，並且可以接受應用程式的呼叫。它們也會取用資源。因您正在重新建立 Aurora 全域資料庫，請先移除其他次要資料庫叢集，再在後續步驟中建立新的 Aurora 全域資料庫。這樣可以避免 Aurora 全域資料庫的資料庫叢集之間出現資料不一致的情況 (稱為核心分裂問題)。

   如需分離的詳細步驟，請參閱從 Amazon Aurora 全域資料庫中移除叢集。

4. 重新設定您的應用程式，以使用其新端點將所有寫入操作傳送至這個目前獨立的 Aurora 資料庫叢集。如果您在建立 Aurora 全域資料庫時接受提供的名稱，則可透過從應用程式中叢集的端點字串移除 `-ro` 來變更端點。

   例如，當該叢集從 Aurora 全域資料庫分離時，次要叢集的端點 `my-global.cluster-ro-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` 會變成 `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com`。

   當您開始在下一個步驟中新增區域時，此 Aurora 資料庫叢集會成為新的 Aurora 全域資料庫的主要叢集。

   如果您使用 RDS 代理，請務必將應用程式的寫入操作重新導向至與新主要叢集相關聯之代理的適當讀取/寫入端點。此代理端點可能是預設端點或自訂讀取/寫入端點。如需更多資訊，請參閱RDS Proxy 端點如何使用全域資料庫。
5. 將 AWS 區域 新增至資料庫叢集。當您執行這項操作時，從主要到次要的複寫程序即會開始。如需新增區域的詳細步驟，請參閱將 AWS 區域 新增到 Amazon Aurora 全域資料庫。
6. 視需新增更多 AWS 區域，以重新建立支援您的應用程式所需的拓撲。

請確定應用程式寫入會在做出這些變更之前、期間和之後，傳送至正確的 Aurora 資料庫叢集。這樣可以避免 Aurora 全域資料庫的資料庫叢集之間出現資料不一致的情況 (稱為核心分裂問題)。

如果您已重新設定以回應 AWS 區域 中的中斷，則可以在解決了該中斷之後，使 AWS 區域 再次成為主要區域。若要這麼做，請將舊的 AWS 區域新增至新的全球資料庫，然後使用轉換程序來切換其角色。您的 Aurora 全球資料庫必須使用支援轉換的 Aurora PostgreSQL 或 Aurora MySQL 版本。如需更多詳細資訊，請參閱 針對 Amazon Aurora 全球資料庫執行轉換。

管理 Aurora PostgreSQL – 全域資料庫的 RPO

透過 Aurora PostgreSQL 型全球資料庫，您就可以使用 `rds.global_db_rpo` 參數來管理 Aurora 全球資料庫的復原點目標 (RPO)。RPO 表示發生中斷時可能遺失的最大資料量。

當您為 Aurora PostgreSQL – 全域資料庫設定 RPO 時，Aurora 會監控所有次要叢集的 RPO 延遲時間，以確定至少有一個次要叢集保留在目標 RPO 視窗。RPO 延遲時間是另一個以時間為基礎的指標。

容錯移轉完成之後，資料庫會在新的 AWS 區域 中使用 RPO 來恢復操作。Aurora 會評估 RPO 和 RPO 延遲時間，以在主要叢集上遞交 (或封鎖) 交易，如下所示：

- 如果至少有一個次要資料庫叢集的 RPO 延遲時間小於 RPO，則會遞交交易。

- 如果所有次要資料庫叢集的 RPO 延遲時間大於 RPO，則會封鎖交易。它還會將事件記錄到 PostgreSQL 日誌檔案，並發出顯示已封鎖工作階段的「等待」事件。

換言之，如果所有次要叢集都落後於目標 RPO，Aurora 會暫停主要叢集上的交易，直至至少有一個次要叢集追趕上來。至少一個次要資料庫叢集的延遲時間變得小於 RPO 時，暫停的交易即會恢復並進行遞交。因此，在滿足 RPO 之前，沒有任何可以遞交的交易。

此 `rds.global_db_rpo` 參數是動態的。如果您決定不要讓所有寫入交易都停止，直到延遲降低到足夠的程度，那麼您可以快速重設它。在這種情況下，Aurora 會在經過短暫延遲後，辨識並實作變更。

> **Important**
>
> 在只有兩個 AWS 區域的全球資料庫中，我們建議在次要區域的參數群組中保留 `rds.global_db_rpo` 參數的預設值。否則，由於失去主要 AWS 區域而執行容錯移轉時，可能會導致 Aurora 暫停交易。請務必等到 Aurora 在舊的故障 AWS 區域中完成叢集重建，再變更此參數以強制執行最大 RPO。

如果您按照下列所述設定此參數，還可以監控其產生的指標。您可以使用 `psql` 或其他工具來查詢 Aurora 全域資料庫的主要資料庫叢集，並取得有關您的 Aurora PostgreSQL – 全域資料庫操作的詳細資訊。若要瞭解如何操作，請參閱監控 Aurora PostgreSQL 型全球資料庫。

主題

- 設定復原點目標
- 檢視復原點目標
- 停用復原點目標

設定復原點目標

該 `rds.global_db_rpo` 參數可控制 PostgreSQL 資料庫的 RPO 設定。Aurora PostgreSQL 支援此參數。`rds.global_db_rpo` 的有效值範圍為 20 秒至 2,147,483,647 秒 (68 年)。選擇符合您的業務需求和應用案例的實際值。例如，您可能想要允許最多 10 分鐘的 RPO，在此情況下，您可以將值設定為 600。

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API，為 Aurora PostgreSQL 型全域資料庫設定該值。

主控台

設定 RPO

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 選擇 Aurora 全域資料庫的主要叢集，然後開啟 Configuration (組態) 標籤，以尋找其資料庫叢集參數群組。例如，執行 Aurora PostgreSQL 11.7 的主要資料庫叢集的預設參數群組為 `default.aurora-postgresql11`。

   無法直接編輯參數群組。而是執行下列動作：

   - 使用適當的預設參數群組作為起點，建立自訂資料庫叢集參數群組。例如，根據 `default.aurora-postgresql11` 建立自訂資料庫叢集參數群組。
   - 在您的自訂資料庫參數群組上，設定 rds.global_db_rpo 參數的值，以符合您的應用案例。有效值範圍為 20 秒至最大整數值 2,147,483,647 (68 年)。
   - 將修改後的資料庫叢集參數群組套用至您的 Aurora 資料庫叢集。

如需更多詳細資訊，請參閱 在 Amazon Aurora 中修改資料庫叢集參數群組中的參數。

AWS CLI

若要設定 `rds.global_db_rpo` 參數，請使用 modify-db-cluster-parameter-group CLI 命令。在命令中，指定主要叢集參數群組的名稱和 RPO 參數的值。

下列範例會針對名稱為 `my_custom_global_parameter_group` 的主要資料庫叢集參數群組，將 RPO 設定為 600 秒 (10 分鐘)。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my_custom_global_parameter_group \
    --parameters
 "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

在 Windows 中：

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my_custom_global_parameter_group ^
```

```
    --parameters
 "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

RDS API

若要修改 rds.global_db_rpo 參數，請使用 Amazon RDS ModifyDBClusterParameterGroup API 操作。

檢視復原點目標

對於每個資料庫叢集，全域資料庫的復原點目標 (RPO) 會存放在 rds.global_db_rpo 參數中。您可以連線至要檢視之次要叢集的端點，並使用 psql 來查詢執行個體的該值。

```
db-name=>show rds.global_db_rpo;
```

如果未設定此參數，則查詢會傳回下列項目：

```
rds.global_db_rpo
-------------------
 -1
(1 row)
```

此下一個回應來自具有 1 分鐘 RPO 設定的次要資料庫叢集。

```
rds.global_db_rpo
-------------------
 60
(1 row)
```

您還可以使用 CLI 來獲取值，來尋找 rds.global_db_rpo 是否在任何 Aurora 資料叢集上啟用，方法是透過使用 CLI 來獲取叢集所有 user 參數的值。

對於 Linux、macOS 或 Unix：

```
aws rds describe-db-cluster-parameters \
 --db-cluster-parameter-group-name lab-test-apg-global \
 --source user
```

在 Windows 中：

```
aws rds describe-db-cluster-parameters ^
 --db-cluster-parameter-group-name lab-test-apg-global *
 --source user
```

該命令會傳回所有 user 參數類似於下列各項的輸出。這不是 default-engine 或 system 資料庫叢集參數。

```
{
    "Parameters": [
        {
            "ParameterName": "rds.global_db_rpo",
            "ParameterValue": "60",
            "Description": "(s) Recovery point objective threshold, in seconds, that
 blocks user commits when it is violated.",
            "Source": "user",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "20-2147483647",
            "IsModifiable": true,
            "ApplyMethod": "immediate",
            "SupportedEngineModes": [
                "provisioned"
            ]
        }
    ]
}
```

若要進一步了解如何檢視叢集參數群組的參數，請參閱在 Amazon Aurora 中檢視資料庫叢集參數群組的參數值。

停用復原點目標

若要停用 RPO，請重設 rds.global_db_rpo 參數。您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 來重設該參數。

主控台

停用 RPO

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。

2. 在導覽窗格中，選擇 Parameter groups (參數群組)。
3. 在清單中，選擇主要資料庫叢集參數群組。
4. 選擇 Edit parameters (編輯參數)。
5. 選擇 rds.global_db_rpo 參數旁的方塊。
6. 選擇 Reset (重設)。
7. 當畫面顯示 Reset parameters in DB parameter group (重設資料庫參數群組中的參數) 時，選擇 Reset parameters (重設參數)。

如需如何使用主控台重設參數的詳細資訊，請參閱在 Amazon Aurora 中修改資料庫叢集參數群組中的參數。

AWS CLI

若要重設 `rds.global_db_rpo` 參數，請使用 reset-db-cluster-parameter-group 命令。

對於 Linux、macOS 或 Unix：

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name global_db_cluster_parameter_group \
    --parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

在 Windows 中：

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name global_db_cluster_parameter_group ^
    --parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

RDS API

若要重設 `rds.global_db_rpo` 參數，請使用 Amazon RDS API ResetDBClusterParameterGroup 操作。

全球資料庫次要叢集的跨區域彈性

Aurora PostgreSQL 版本 16.6、15.10、14.15、13.18、12.22 或更高版本，以及 Aurora MySQL 版本 3.09 或更高版本包含可用性改進，可讓次要區域僅供讀取複本在意外事件期間維持服務持續性，例如硬體故障、跨 AWS 區域網路中斷、叢集之間的大量資料傳輸等。

雖然僅供讀取複本仍可用於您的應用程式請求，但複寫延遲可能會繼續增加，直到意外事件解決為止。您可以使用 AuroraGlobalDBProgressLag CloudWatch 指標監控主要和次要叢集之間的延遲。若要測量端對端延遲，包括叢集磁碟區與次要叢集資料庫執行個體之間的任何延遲，請新增 AuroraGlobalDBProgressLag 和 AuroraReplicaLag CloudWatch 指標的值。如需指標的詳細資訊，請參閱 Amazon Aurora 的指標參考。

在此類意外事件期間，Aurora MySQL 和舊版 Aurora PostgreSQL 的全球資料庫讀取可用性可能會受到影響。

如需 Aurora PostgreSQL 16.6、15.10、14.15、13.18 和 12.22 中新功能的詳細資訊，請參閱 Aurora PostgreSQL 版本備註中的 PostgreSQL 16.6。

如需 Aurora MySQL 3.09 版和更新版本中新功能的詳細資訊，請參閱 Aurora MySQL 版本備註中的 Amazon Aurora MySQL 第 3 版的資料庫引擎更新。

監控 Amazon Aurora 全域資料庫

當您建立組成 Aurora 全域資料庫的 Aurora 資料庫叢集時，您可以選擇許多選項，讓您監控資料庫叢集的效能。這些選項包括下列項目：

- Amazon RDS 績效詳情 – 可在基礎 Aurora 資料庫引擎中啟用效能結構描述。若要深入了解績效詳情與 Aurora 全域資料庫，請參閱利用 Amazon RDS 績效詳情來監控 Amazon Aurora 全域資料庫。
- 增強監控 – 產生 CPU 上處理程序或執行緒使用率的指標。若要瞭解增強型監控，請參閱 使用增強型監控來監控作業系統指標。
- Amazon CloudWatch Logs – 將指定的日誌類型發佈至 CloudWatch Logs。預設會發佈錯誤日誌，但您可以選擇 Aurora 資料庫引擎特定的其他日誌。
  - 對於 Aurora MySQL – 型 Aurora 資料庫叢集，您可以匯出稽核日誌、一般日誌和慢查詢日誌。
  - 對於 Aurora PostgreSQL 型 Aurora 資料庫叢集，您可以匯出 PostgreSQL 日誌。
- 對於 Aurora MySQL – 型全球資料庫，您可以查詢指定 information_schema 資料表來檢查 Aurora 全球資料庫及其執行個體的狀態。若要了解作法，請參閱監控 Aurora MySQL 型全球資料庫。
- 對於 Aurora PostgreSQL – 型全球資料庫，您可以使用指定函數來檢查 Aurora 全球資料庫及其執行個體的狀態。若要瞭解如何操作，請參閱監控 Aurora PostgreSQL 型全球資料庫。

下面的螢幕擷取畫面顯示了一些在 Aurora 全域資料庫中的主要 Aurora 資料庫叢集的 Monitoring (監控) 索引標籤上可用的選項。



如需詳細資訊，請參閱在 Amazon Aurora 叢集中監控指標。

利用 Amazon RDS 績效詳情來監控 Amazon Aurora 全域資料庫

您可以為 Aurora 全域資料庫使用 Amazon RDS 績效詳情。您可以為 Aurora 全域資料庫中的每個 Aurora 資料庫叢集個別啟用此功能。若要執行這項操作，您可以在 Create database (建立資料庫) 頁面的 Additional configuration (其他組態) 區段中選擇 Enable Performance Insights (啟用績效詳情)。或者，您可以修改您的 Aurora 資料庫叢集，以在其啟動並執行之後使用這項功能。對於每個屬於 Aurora 全域資料庫的叢集，您都可以啟用或關閉績效詳情。

績效詳情所建立的報告適用於全域資料庫中的每個叢集。當您將新的次要 AWS 區域 新增至已使用績效詳情的 Aurora 全域資料庫時，請務必在新增的叢集中啟用績效詳情。這並不會從現有的全球資料庫中繼承績效詳情的設定。

您可以在檢視連接到全域資料庫之資料庫執行個體的績效詳情頁面 AWS 區域 時切換。不過，切換後可能不會立即看到效能資訊 AWS 區域。雖然每個資料庫執行個體的名稱可能相同 AWS 區域，但每個

資料庫執行個體相關聯的績效詳情 URL 都不同。切換後 AWS 區域，請在績效詳情導覽窗格中再次選擇資料庫執行個體的名稱。

針對資料庫執行個體連結與全域資料庫的關聯，在每個 AWS 區域影響效能的因素可能不同。例如，每個 中的資料庫執行個體 AWS 區域 可能具有不同的容量。

若要深入了解如何使用 Performance Insights (績效詳情)，請參閱在 Amazon Aurora 上使用績效詳情監控資料庫負載。

使用資料庫活動串流來監控 Aurora 全域資料庫

透過使用資料庫活動串流功能，您可以在全域資料庫的資料庫叢集中監控和設定稽核活動的警示。您會在每個資料庫叢集上分別啟動資料庫活動串流。每個叢集在其自己的 AWS 區域中將稽核資料傳送至其自己的 Kinesis 串流。如需詳細資訊，請參閱使用資料庫活動串流來監控 Amazon Aurora。

監控 Aurora MySQL 型全球資料庫

若要檢視以 Aurora MySQL 型全球資料庫的狀態，請查詢 information_schema.aurora_global_db_status 和 information_schema.aurora_global_db_instance_status 資料表。

> **Note**
>
> Aurora MySQL 3.04.0 版及更新版本全球資料庫才能使用 `information_schema.aurora_global_db_status` 和 `information_schema.aurora_global_db_instance_status` 資料表。

監控 Aurora MySQL 型全球資料庫

1. 使用 MySQL 用戶端連線至全球資料庫主要叢集端點。如需如何連線的詳細資訊，請參閱 連線至 Amazon Aurora 全域資料庫。
2. 在 mysql 命令中查詢 `information_schema.aurora_global_db_status` 資料表，以列出主要和次要磁碟區。此查詢會傳回全球資料庫次要資料庫叢集的延遲時間，如下列範例所示。

```
mysql> select * from information_schema.aurora_global_db_status;
```

```
AWS_REGION | HIGHEST_LSN_WRITTEN | DURABILITY_LAG_IN_MILLISECONDS |
 RPO_LAG_IN_MILLISECONDS | LAST_LAG_CALCULATION_TIMESTAMP | OLDEST_READ_VIEW_TRX_ID
```

```
-----------+---------------------+--------------------------------
+------------------------+---------------------------------
+------------------------
us-east-1  |           183537946 |                              0   |
      0 |  1970-01-01 00:00:00.000000     |               0
us-west-2  |           183537944 |                              428 |
      0 |  2023-02-18 01:26:41.925000     |               20806982
(2 rows)
```

輸出包含全球資料庫中每個資料庫叢集的資料列，其中包括以下欄位：

- AWS_REGION – AWS 區域 此資料庫叢集所在的 。如需 AWS 區域 依引擎列出的資料表，請參閱 區域可用性。
- HIGHEST_LSN_WRITTEN – 目前寫入此資料庫叢集的最大記錄序號 (LSN)。

  記錄序號 (LSN) 是一組獨特的序號，可用來識別資料庫交易日誌中的記錄。系統會排序 LSN，而 LSN 越大，就表示交易發生時間越後面。
- DURABILITY_LAG_IN_MILLISECONDS – 次要資料庫叢集上的 HIGHEST_LSN_WRITTEN 與主要資料庫叢集上的 HIGHEST_LSN_WRITTEN 之間的時間戳記值差異。在 Aurora 全球資料庫的主要資料庫叢集上，此值始終為 0。
- RPO_LAG_IN_MILLISECONDS – 復原點目標 (RPO) 延遲。RPO 延遲是指最近使用者交易 COMMIT 在儲存於 Aurora 全域資料庫的主資料庫叢集之後，將其儲存於次要資料庫叢集上所需的時間。在 Aurora 全球資料庫的主要資料庫叢集上，此值始終為 0。

  簡言之，此指標會計算 Aurora 全球資料庫中每個 Aurora MySQL 資料庫叢集的復原點目標，亦即，若有中斷發生，可能會遺失的資料量。與延遲一樣，RPO 是以時間來衡量。
- LAST_LAG_CALCULATION_TIMESTAMP – 上次計算 DURABILITY_LAG_IN_MILLISECONDS 和 RPO_LAG_IN_MILLISECONDS 值時指定的時間戳記。例如 1970-01-01 00:00:00+00 之類的時間值表示此為主要資料庫叢集。
- OLDEST_READ_VIEW_TRX_ID – 寫入器資料庫執行個體可清除到最舊的交易 ID。

3. 查詢 information_schema.aurora_global_db_instance_status 資料表列出主要資料庫叢集與次要資料庫叢集的所有次要資料庫執行個體。

```
mysql> select * from information_schema.aurora_global_db_instance_status;
```

```
SERVER_ID            |              SESSION_ID              | AWS_REGION
 | DURABLE_LSN | HIGHEST_LSN_RECEIVED | OLDEST_READ_VIEW_TRX_ID |
 OLDEST_READ_VIEW_LSN | VISIBILITY_LAG_IN_MSEC
```

```
--------------------+--------------------------------------+-----------
+------------+---------------------+------------------------
+---------------------+------------------------
ams-gdb-primary-i2   | MASTER_SESSION_ID                    | us-east-1  |
 183537698   |                   0 |                      0 |
 0 |                      0
ams-gdb-secondary-i1 | cc43165b-bdc6-4651-abbf-4f74f08bf931 | us-west-2  |
 183537689   |           183537692 |                20806928 |
 183537682 |                     0
ams-gdb-secondary-i2 | 53303ff0-70b5-411f-bc86-28d7a53f8c19 | us-west-2  |
 183537689   |           183537692 |                20806928 |
 183537682 |                   677
ams-gdb-primary-i1   | 5af1e20f-43db-421f-9f0d-2b92774c7d02 | us-east-1  |
 183537697   |           183537698 |                20806930 |
 183537691 |                    21
(4 rows)
```

輸出包含全球資料庫中每個資料庫執行個體的資料列，其中包括以下欄位：

- SERVER_ID – 此資料庫執行個體的伺服器識別符。
- SESSION_ID – 目前工作階段的唯一識別符。MASTER_SESSION_ID 值可識別寫入器 (主) 資料庫執行個體。
- AWS_REGION – AWS 區域 此資料庫執行個體所在的 。如需 AWS 區域 依引擎列出的資料表，請參閱 區域可用性。
- DURABLE_LSN – 可長期儲存的 LSN。
- HIGHEST_LSN_RECEIVED – 資料庫執行個體從寫入器資料庫執行個體接收的最大 LSN。
- OLDEST_READ_VIEW_TRX_ID – 寫入器資料庫執行個體可清除到最舊的交易 ID。
- OLDEST_READ_VIEW_LSN – 資料庫執行個體從儲存體中讀取資料時所使用的最舊 LSN。
- VISIBILITY_LAG_IN_MSEC— 對於主要資料庫叢集中的讀取器，此資料庫執行個體延遲於寫入器資料庫執行個體的時間 (以毫秒為單位)。對於次要資料庫叢集中的讀取器，此資料庫執行個體延遲於次要磁碟區的時間 (以毫秒為單位)。

若要查看這些值如何隨時間改變，則可考慮使用以下需要一小時才能插入資料表的交易區塊：

```
mysql> BEGIN;
mysql> INSERT INTO table1 SELECT Large_Data_That_Takes_1_Hr_To_Insert;
mysql> COMMIT;
```

在某些情況下，BEGIN 陳述式後，主要資料庫叢集和次要資料庫叢集之間網路可能會斷線。如果是這樣，則次要資料庫叢集的 DURABILITY_LAG_IN_MILLISECONDS 值會開始增加。INSERT 陳述式結束時，DURABILITY_LAG_IN_MILLISECONDS 值為 1 小時。但是，RPO_LAG_IN_MILLISECONDS 值會是 0，這是因為主要資料庫叢集與次要資料庫叢集之間遞交的所有使用者資料仍是一樣的。只要 COMMIT 陳述式完成，RPO_LAG_IN_MILLISECONDS 的值就會增加。

監控 Aurora PostgreSQL 型全球資料庫

若要檢視 Aurora PostgreSQL 型全球資料庫的狀態，請使用 `aurora_global_db_status` 和 `aurora_global_db_instance_status` 函數。

> **Note**
>
> 只有 Aurora PostgreSQL 支援 `aurora_global_db_status` 和 `aurora_global_db_instance_status` 函數。

監控 Aurora PostgreSQL 型全域資料庫

1. 使用 PostgreSQL 公用程式 (如 psql) 連線至全球資料庫主要叢集端點。如需如何連線的詳細資訊，請參閱 連線至 Amazon Aurora 全域資料庫。
2. 在 psql 命令中使用 `aurora_global_db_status` 函數，以列出主要和次要磁碟區。如此即會顯示全球資料庫次要資料庫叢集的延遲時間。

```
postgres=> select * from aurora_global_db_status();
```

```
aws_region | highest_lsn_written | durability_lag_in_msec | rpo_lag_in_msec |
 last_lag_calculation_time  | feedback_epoch | feedback_xmin
------------+---------------------+------------------------+-----------------
+----------------------------+----------------+---------------
us-east-1  |         93763984222 |                     -1 |              -1 |
 1970-01-01 00:00:00+00     |              0 |              0
us-west-2  |         93763984222 |                    900 |            1090 |
 2020-05-12 22:49:14.328+00 |              2 |    3315479243
(2 rows)
```

輸出包含全球資料庫中每個資料庫叢集的資料列，其中包括以下欄位：

- aws_region – AWS 區域 此資料庫叢集所在的 。如需 AWS 區域 依引擎列出的資料表，請參閱區域可用性。
- highest_lsn_written – 目前寫入此資料庫叢集的最大記錄序號 (LSN)。

  記錄序號 (LSN) 是一組獨特的序號，可用來識別資料庫交易日誌中的記錄。系統會排序 LSN，而 LSN 越大，就表示交易發生時間越後面。
- durability_lag_in_msec – 寫入次要資料庫叢集之最大記錄序號 (highest_lsn_written) 與寫入主要資料庫叢集之 highest_lsn_written 的時間戳記差異。
- rpo_lag_in_msec – 復原點目標 (RPO) 延遲。此延遲是指存放在次要資料庫叢集之最近使用者交易遞交，以及存放在主要資料庫叢集之最近使用者交易遞交間的時間差異。
- last_lag_calculation_time – 上次計算 durability_lag_in_msec 和 rpo_lag_in_msec 值時的時間戳記。
- feedback_epoch – 次要資料庫叢集產生熱待命資訊時所使用的 epoch。

  Hot standby (熱待命) 指的是在伺服器處於復原或待命模式時，資料庫叢集仍可執行連線與查詢操作。熱待命回饋則會提供資料庫叢集在熱待命期間的相關資訊。如需詳細資訊，請參閱 PostgreSQL 文件中的熱待命。
- feedback_xmin – 次要資料庫叢集所使用的最小 (最舊) 作用中交易 ID。

3. 使用 aurora_global_db_instance_status 函數列出主要資料庫叢集與次要資料庫叢集的所有次要資料庫執行個體。

```
postgres=> select * from aurora_global_db_instance_status();
```

```
server_id                                    |              session_id
  | aws_region | durable_lsn | highest_lsn_rcvd | feedback_epoch | feedback_xmin |
 oldest_read_view_lsn | visibility_lag_in_msec
----------------------------------------------+--------------------------------------
+------------+-------------+------------------+----------------+---------------
+----------------------+-----------------------
apg-global-db-rpo-mammothrw-elephantro-1-n1 | MASTER_SESSION_ID
 | us-east-1  | 93763985102 |                  |                |               |
                      |
apg-global-db-rpo-mammothrw-elephantro-1-n2 | f38430cf-6576-479a-b296-dc06b1b1964a
 | us-east-1  | 93763985099 |      93763985102 |              2 |    3315479243 |
        93763985095 |                      10
```

```
apg-global-db-rpo-elephantro-mammothrw-n1   | 0d9f1d98-04ad-4aa4-8fdd-e08674cbbbfe
 | us-west-2  | 93763985095 |      93763985099 |              2 |    3315479243 |
        93763985089 |                   1017
(3 rows)
```

輸出包含全球資料庫中每個資料庫執行個體的資料列，其中包括以下欄位：

- server_id – 此資料庫執行個體的伺服器識別符。
- session_id – 目前工作階段的唯一識別符。
- aws_region – AWS 區域 此資料庫執行個體所在的 。如需 AWS 區域 依引擎列出的資料表，請參閱 區域可用性。
- durable_lsn – 可長期儲存的 LSN。
- highest_lsn_rcvd – 資料庫執行個體從寫入器資料庫執行個體接收的最大 LSN。
- feedback_epoch – 資料庫執行個體產生熱待命資訊時所使用的 epoch。

  熱待命指的是在伺服器處於復原或待命模式時，資料庫執行個體仍可執行連線與查詢操作。熱待命回饋則會提供資料庫執行個體在熱待命期間的相關資訊。如需詳細資訊，請參閱 PostgreSQL 文件以了解有關熱待命。
- feedback_xmin – 資料庫執行個體所使用的最小 (最舊) 作用中交易 ID。
- oldest_read_view_lsn – 資料庫執行個體從儲存體中讀取資料時所使用的最舊 LSN。
- visibility_lag_in_msec – 此資料庫執行個體落後寫入器資料庫執行個體的程度。

若要查看這些值如何隨時間改變，則可考慮使用以下需要一小時才能插入資料表的交易區塊：

```
psql> BEGIN;
psql> INSERT INTO table1 SELECT Large_Data_That_Takes_1_Hr_To_Insert;
psql> COMMIT;
```

在某些情況下，BEGIN 陳述式後，主要資料庫叢集和次要資料庫叢集之間網路可能會斷線。如果是這樣，次要資料庫叢集的 durability_lag_in_msec 值會開始增加。在 INSERT 陳述式的結尾，durability_lag_in_msec 值為 1 小時。但 rpo_lag_in_msec 值會是 0，這是因為主要資料庫叢集與次要資料庫叢集之間遞交的所有使用者資料仍是一樣的。一旦 COMMIT 陳述式完成，rpo_lag_in_msec 值就會增加。

針對 Amazon Aurora Global Database 使用藍/綠部署

Amazon RDS 藍/綠部署提供安全測試資料庫變更的功能。針對您的全球資料庫，藍/綠部署可讓您以最短的停機時間執行升級和維護操作。您可以建立完全受管的預備環境（綠色），以鏡像整個生產環境（藍色），包括主要叢集和跨多個 AWS 區域的所有相關次要區域。預備環境反映您的生產設定，可讓您在切換到新環境之前可靠地測試變更。在整個過程中，藍/綠部署可讓預備環境和生產環境保持同步。準備好將預備環境設為新的生產環境時，請執行藍/綠切換。此操作會將您的主要和所有次要區域轉換為綠色環境，停機時間通常不到一分鐘。此服務會自動重新命名叢集、執行個體和端點以符合原始生產環境，讓您的應用程式能夠存取新的生產環境，而不需要任何組態變更，並將營運開銷降至最低。

主題

- 搭配 Aurora Global Database 使用藍/綠部署的優勢
- 藍/綠部署如何與 Aurora Global Database 搭配使用

搭配 Aurora Global Database 使用藍/綠部署的優勢

- 在 Aurora Global Databases 上執行主要版本、次要版本和系統更新，包括資料庫修補程式和作業系統升級，同時保持最短的停機時間。
- 在全域資料庫的主要和次要區域中建立生產就緒預備（綠色）環境，以測試和實作較新的資料庫功能。
- 根據您的工作負載，使用內建的切換護欄安全地切換，停機時間通常不到一分鐘。
- 在藍/綠轉換過程中維護災難復原功能，允許藍/綠轉換期間進行全域資料庫容錯移轉。
- 您的流量將導向至新的生產環境，而不需要任何應用程式變更。

藍/綠部署如何與 Aurora Global Database 搭配使用

如需如何建立、檢視、切換和刪除藍/綠部署的詳細資訊，請參閱 使用 Amazon Aurora 藍/綠部署進行資料庫更新。您可以使用此功能進行主要或次要版本升級、透過參數更新改善資料庫效能，以及採用新的資料庫功能，並將停機時間降至最低。

下方顯示具有一個次要區域的 Aurora Global Database 藍/綠部署在藍/綠轉換前後的外觀。


Primary Region
VPC-A
Secondary Region
VPC-B
Users/Applications
Blue Environment
(Current Production)
Old version
Primary Cluster
Aurora Global Database
Asynchronous
Replication
Secondary Cluster
Blue/Green
Deployment
Logical
Replication
(managed)
Green Environment
(Future Production)
New version
Primary Cluster
Aurora Global Database
Asynchronous
Replication
Secondary Cluster


Blue/Green
deployment
switchover


Primary Region
VPC-A
Secondary Region
VPC-B
Old Blue Environment
(Old Production)
Old version
Primary Cluster
Aurora Global Database
Asynchronous
Replication
Secondary Cluster
Blue/Green
Deployment
Blue Environment
(Current Production)
New version
Primary Cluster
Aurora Global Database
Asynchronous
Secondary Cluster

您可以從 全球資料庫的主要區域建立藍/綠部署。選取引擎組態，例如綠色環境的主要或次要引擎版本、資料庫參數群組和資料庫叢集參數群組。Amazon RDS 會複製藍色環境的綠色環境拓撲。中的視覺化呈現 AWS 管理主控台 如下所示。



> **Note**
>
> 在藍/綠切換期間支援全域容錯移轉，但在藍/綠切換期間不支援全域切換。

當您在 RDS 藍色/綠色切換期間啟動全域容錯移轉時，目標區域會自動轉返至藍色環境，或在全域容錯移轉發生之前轉返至綠色環境。

如需建立、檢視、切換和刪除藍/綠部署的資訊，請參閱 使用 Amazon Aurora 藍/綠部署進行資料庫更新。遵循全球資料庫的相同工作流程，並在相關步驟中註明特定指示。

將 Amazon Aurora 全域資料庫與其他 AWS 服務搭配使用

您可以使用 Aurora 全域資料庫搭配其他服務 AWS ，例如 Amazon S3 和 AWS Lambda。這樣做需要全域資料庫中的所有 Aurora 資料庫叢集在各自的 AWS 區域中具有相同的權限、外部函數等。由於 Aurora 全域資料庫中的唯讀 Aurora 次要資料庫叢集可以升級為主要角色，因此建議在您計劃與 Aurora 全域資料庫搭配使用的任何服務的所有資料 Aurora 庫叢集上，先設定寫入權限。

下列程序摘要說明每個 要採取的動作 AWS 服務。

從 Aurora 全域資料庫叫用 AWS Lambda 函數

1. 對於組成 Aurora 全球資料庫的所有 Aurora 叢集，請執行從 Amazon Aurora MySQL 資料庫叢集叫用 Lambda 函式中的程序。
2. 針對 Aurora 全域資料庫中的每個叢集，設定新 IAM (IAM) 角色的 (ARN)。
3. 若要允許 Aurora 全球資料庫中的資料庫使用者叫用 Lambda 函數，請將您在建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務中建立的角色與 Aurora 全球資料庫中的每個叢集建立關聯。
4. 設定 Aurora 全球資料庫中的每個叢集來允許對外連線至 Lambda。如需說明，請參閱「啟用從 Amazon Aurora 到其他服務的網路通訊 AWS」。

從 Amazon S3 載入資料

1. 對於組成 Aurora 全球資料庫的所有 Aurora 叢集，請執行從 Amazon S3 儲存貯體中的文字檔案將資料載入 Amazon Aurora MySQL 資料庫叢集中的程序。
2. 對於全球資料庫中的每個 Aurora 叢集，請將 `aurora_load_from_s3_role` 或 `aws_default_s3_role` 資料庫叢集參數設為新的 IAM 角色的 Amazon Resource Name (ARN)。如果 `aurora_load_from_s3_role` 中未指定 IAM 角色，Aurora 會使用 `aws_default_s3_role` 中指定的 IAM 角色。
3. 若要允許 Aurora 全球資料庫中的資料庫使用者存取 S3，請將您在建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務中建立的角色與全球資料庫中的每個 Aurora 叢集建立關聯。
4. 設定 Aurora 全球資料庫中的每個叢集來允許對外連線至 S3。如需說明，請參閱「啟用從 Amazon Aurora 到其他服務的網路通訊 AWS」。

將查詢的資料儲存至 Amazon S3

1. 對於組成 Aurora 全球資料庫的所有 Aurora 叢集，請執行 將來自 Amazon Aurora MySQL 資料庫叢集的資料儲存至 Amazon S3 儲存貯體中的文字檔案 或 將資料從 Aurora PostgreSQL 資料庫叢集匯出至 Amazon S3 中的程序。
2. 對於全球資料庫中的每個 Aurora 叢集，請將 `aurora_select_into_s3_role` 或 `aws_default_s3_role` 資料庫叢集參數設為新的 IAM 角色的 Amazon Resource Name (ARN)。如果 `aurora_select_into_s3_role` 中未指定 IAM 角色，Aurora 會使用 `aws_default_s3_role` 中指定的 IAM 角色。
3. 若要允許 Aurora 全球資料庫中的資料庫使用者存取 S3，請將您在建立 IAM 角色以允許 Amazon Aurora 存取 AWS 服務中建立的角色與全球資料庫中的每個 Aurora 叢集建立關聯。
4. 設定 Aurora 全球資料庫中的每個叢集來允許對外連線至 S3。如需說明，請參閱啟用從 Amazon Aurora 到其他服務的網路通訊 AWS。

搭配 Aurora Global Database 使用 Amazon Application Recovery Controller (ARC)

規劃業務持續性和災難復原策略時，您需要協調跨應用程式堆疊及其相依性的復原。Amazon Application Recovery Controller (ARC) 與 Aurora Global Database 整合，透過 ARC Region Switch 自動化此程序，ARC Region Switch 是自動化多區域應用程式復原的集中式解決方案。區域切換會協調跨 AWS 帳戶和區域的容錯移轉步驟、提供即時復原儀表板，並透過彙整跨資源和帳戶的資料來產生合規報告。進一步了解如何使用適用於 Aurora Global Database 的區域切換。

升級 Amazon Aurora 全域資料庫

升級 Aurora 全域資料庫的程序與升級 Aurora 資料庫叢集的程序相同。但是，以下是在開始該程序之前需要注意的一些重要差異。

建議您將主要和次要資料庫叢集升級至相同版本。只有在主要和次要資料庫叢集具有相同的主要、次要和程式修補等級引擎版本時，您才能在 Aurora 全域資料庫上執行受管跨區域資料庫容錯移轉。但是，修補程式等級可能會有所不同，取決於次要引擎版本。如需詳細資訊，請參閱受管跨區域轉換和容錯移轉的修補程式等級相容性。

主要版本升級

在執行 Amazon Aurora 全域資料庫的主要版本升級時，您將升級全域資料庫叢集，而不是升級其中包含的各個叢集。

若要了解如何將 Aurora PostgreSQL 全域資料庫升級到更高的主要版本，請參閱 全域資料庫的主要升級。

> **Note**
>
> 使用以 Aurora PostgreSQL 為基礎的 Aurora 全域資料庫時，如果啟用復原點目標 (RPO) 功能，就無法執行 Aurora 資料庫引擎的主要版本升級。如需 RPO 功能的相關資訊，請參閱 管理 Aurora PostgreSQL – 全域資料庫的 RPO。

若要了解如何將 Aurora MySQL 全域資料庫升級到更高的主要版本，請參閱 全域資料庫的就地主要升級。

> **Note**
>
> 使用以 Aurora MySQL 為基礎的 Aurora 全球資料庫時，只有將 `lower_case_table_names` 參數設為預設值並重新啟動全球資料庫，才能從 Aurora MySQL 第 2 版就地升級至第 3 版。若要在使用 `lower_case_table_names` 時執行主要版本升級至 Aurora MySQL 第 3 版，請遵循下列操作：
>
> 1. 從全域叢集移除所有次要區域。請遵循 從 Amazon Aurora 全域資料庫中移除叢集 中的步驟。
> 2. 將主要區域的引擎版升級至 Aurora MySQL 第 3 版。請遵循 就地升級執行方式 中的步驟。
> 3. 將次要區域新增至全域叢集。請遵循 將 AWS 區域 新增到 Amazon Aurora 全域資料庫 中的步驟。
>
> 您也可以改用快照還原技術。如需詳細資訊，請參閱從資料庫叢集快照還原。

次要版本升級

您可以使用單一受管操作和最短的停機時間，將 Aurora 全域資料庫升級至跨所有區域的較新的次要引擎版本，無需個別手動升級每個叢集，並減少全域叢集管理的操作開銷。

了解全域資料庫次要版本升級

您可以透過 RDS API AWS CLI或 升級 全域資料庫的次要版本 AWS 管理主控台。此單一操作會協調跨主要叢集和所有次要 （鏡像） 叢集的升級。如果在升級期間發生問題，服務會自動轉返到現有的版本。

> **Note**
>
> 此受管功能目前僅支援 Aurora PostgreSQL 相容引擎。

當您使用 `modify-global-cluster`命令啟動全域資料庫次要版本升級時，您可以指定目標引擎版本，而 服務會協調所有叢集的升級。此升級會立即套用。

針對 Linux、macOS 或 Unix：

```
aws rds modify-global-cluster \
    --global-cluster-identifier global_cluster_identifier \
    --engine-version target_engine_version
```

在 Windows 中：

```
aws rds modify-global-cluster ^
    --global-cluster-identifier global_cluster_identifier ^
    --engine-version target_engine_version
```

次要版本升級的考量事項

規劃全域資料庫的次要版本升級時，請考慮下列事項：

- 受管功能僅適用於次要版本升級。修補程式版本升級會繼續使用現有的系統更新維護動作。
- 受管功能僅支援 Aurora PostgreSQL 全域叢集。

您可以個別升級全域叢集拓撲中的每個叢集。如果您選擇此方法，請在升級主要叢集之前升級所有次要叢集。升級時，請確保主要和次要資料庫叢集升級至相同的次要版本和修補程式層級。若要更新修補程式層級，請在次要叢集上套用所有待定的維護動作。若要了解如何將 Aurora PostgreSQL 全域資料庫升級到更高的次要版本，請參閱 如何執行次要版本升級和套用修補程式。

Aurora MySQL 全域資料庫的次要版本升級

若要了解如何將 Aurora MySQL 全域資料庫升級到更高的次要版本，請參閱 透過修改引擎版本升級 Aurora MySQL。

在執行升級之前，請考慮以下注意事項：

- 升級次要叢集的次要版本不會以任何方式影響主要叢集的可用性或用量。
- 次要叢集必須至少具有一個資料庫執行個體，才能執行次要升級。
- 如果您將 Aurora MySQL 全球資料庫升級至 2.11.* 版，則須將主要和次要資料庫叢集升級至完全相同的版本 (包含修補程式等級)。
- 若要支援受管跨區域切換或容錯移轉，您可能需要將主要和次要資料庫叢集升級至完全相同的版本和修補程式層級。此要求適用於 Aurora MySQL 和某些 Aurora PostgreSQL 版本。如需允許執行不同修補程式層級之叢集之間的切換和容錯移轉的版本清單，請參閱 受管跨區域轉換和容錯移轉的修補程式等級相容性。

受管跨區域轉換和容錯移轉的修補程式等級相容性

若您的 Aurora 全球資料庫執行下列其中一個次要引擎版本時，即使您主要與次要資料庫叢集的修補程式等級不相符，仍然可以執行受管跨區域切換或容錯移轉。若次要引擎版本低於此清單上的版本，您的主要和次要資料庫叢集必須升級至相同的主要、次要和修補程式層級，才能執行受管跨區域切換或容錯移轉。規劃主要叢集和/或次要叢集的升級時，請務必檢閱下表中的版本資訊和備註。

> **Note**
>
> 對於手動跨區域容錯移轉，只要目標次要資料庫叢集與主要資料庫叢集執行的主要和次要引擎版本相同，您就可以執行容錯移轉程序。在此情況下，修補程式等級不需要相符。
> 如果您的引擎版本需要相同的修補層級，您可以依照 針對 Aurora 全球資料庫執行手動容錯移轉 中的步驟手動執行容錯移轉。

資料庫引擎	次要引擎版本	備註
Aurora MySQL	沒有次要版本	若主要與次要資料庫叢集的修補程式層級不同，則所有 Aurora MySQL 次要版本皆不允許受管跨區域切換或容錯移轉。

資料庫引擎	次要引擎版本	備註
Aurora PostgreSQL	• 15 版或更新的主要版本 • 14.5 版或更新的次要版本 • 13.8 版或更新的次要版本 • 12.12 版或更新的次要版本 • 11.17 版或更新的次要版本	透過上一欄所列的引擎版本，您可以從具某一修補程式等級的主要資料庫叢集中，執行受管跨區域切換或容錯移轉至具不同修補程式等級的次要資料庫叢集。 透過低於這些版本的次要版本，若主要與次要資料庫叢集的修補程式等級相符，您只能執行受管跨區域切換或容錯移轉。 **Warning** 當您將全球資料庫中的叢集更新為下列任何修補程式版本時，您將無法執行跨區域切換或容錯移轉，直到全球資料庫中的所有叢集都執行其中一個修補程式版本或更新版本。 • 修補程式版本 16.1.6、16.2.4、16.3.2 和 16.4.2 • 修補程式版本 15.3.8、15.4.9、15.5.6、15.6.4、15.7.2 和 15.8.2 • 修補程式版本 14.8.8、14.9.9、14.10.6、14.11.4、14.12.2 和 14.13.2

Amazon RDS Proxy for Aurora

透過使用 Amazon RDS Proxy，您可以允許應用程式集中和共用資料庫連線，以改善其擴展能力。RDS Proxy 會自動連線至待命資料庫執行個體，同時保留應用程式連線，使應用程式更具有資料庫故障彈性。透過使用 RDS Proxy，您可以針對連線至代理的用戶端強制執行 AWS Identity and Access Management (IAM) 身分驗證，而代理可以使用 IAM 資料庫身分驗證或存放於 中的登入資料來連線至資料庫 AWS Secrets Manager。



使用 RDS Proxy，您可以處理資料庫流量中無法預測的突增情況。否則，這些突增情況可能會因超額訂閱連線或快速建立新連線而導致問題發生。RDS Proxy 會建立資料庫連線集區，並重複使用此集區中的連線。這種方法可以避免每次開啟新資料庫連線的記憶體和 CPU 額外負荷。若要保護資料庫免於過度訂閱，您可以控制建立的資料庫連線數。

RDS Proxy 會將無法立即從連線集區提供服務的應用程式連線排入佇列或節流。雖然延遲可能會增加，但您的應用程式可以繼續擴展，而不會突然失敗或讓資料庫無法負荷。如果連線請求超過了您指定的限制，則 RDS Proxy 會拒絕應用程式連線 (也就是說，它會分散負載)，同時維持 RDS 能以可用容量提供服務的負載的可預測效能。



您可以減少處理登入憑證，並為每個新連線建立安全連線的負荷。RDS Proxy 可以代表資料庫處理其中的一些工作。

RDS Proxy 與其支援的引擎版本完全相容。您可以針對大部分的應用程式啟用 RDS Proxy，而不需要變更程式碼。如需受支援引擎版本的清單，請參閱 Amazon RDS Proxy 的支援區域和 Aurora 資料庫引擎。

主題

- 區域和版本可用性
- RDS Proxy 的配額和限制
- 規劃在哪裡使用 RDS Proxy
- RDS Proxy 概念和術語
- RDS Proxy 入門
- 管理 RDS Proxy
- 使用 Amazon RDS Proxy 端點
- 使用 Amazon CloudWatch 監控 RDS Proxy 指標
- 使用 RDS Proxy 事件
- RDS Proxy 命令列範例
- RDS Proxy 的故障診斷
- 將 RDS Proxy 與 AWS CloudFormation 搭配使用
- 搭配 Aurora 全域資料庫使用 RDS Proxy

區域和版本可用性

如需有關資料庫引擎版本支援和指定 RDS Proxy 可用性的資訊 AWS 區域，請參閱 Amazon RDS Proxy 的支援區域和 Aurora 資料庫引擎。

RDS Proxy 的配額和限制

下列配額和限制適用於 RDS Proxy：

- 每個 AWS 帳戶 ID 限制為 20 個代理。如果您的應用程式需要更多代理，請透過 中的 Service Quotas 頁面要求增加 AWS 管理主控台。在 Service Quotas 頁面中，選取 Amazon Relational

Database Service (Amazon RDS) 並尋找代理以請求增加配額。 AWS 可以自動增加您的配額或等待支援審核您的請求。

- 每個代理最多可以有 200 個關聯的 Secrets Manager 秘密，因此在使用秘密時，限制最多 200 個不同使用者帳戶的連線。
- 每個代理都有預設端點，只會佈建在從代理設定的子網路中選取的兩個可用區域中。如果設定超過兩個AZs子網路，則會選取任兩個。
- 您最多可以為每個代理新增 20 個額外的代理端點。這些額外的端點會佈建在建立期間指定的所有可用區域。您可以建立、檢視、修改和刪除這些端點。
- 在 Aurora 叢集中，所有使用預設代理端點的連線都會由 Aurora 寫入器執行個體處理。若要針對讀取密集型工作負載執行負載平衡，您可以為代理建立唯讀端點。該端點會將連線傳遞至叢集的讀取器端點。這樣，您的代理連線可以利用 Aurora 讀取可擴展性。如需詳細資訊，請參閱 代理端點概觀。
- 您可以將 RDS Proxy 與 Aurora Serverless v2 叢集搭配使用，但不可與 Aurora Serverless v1 叢集搭配使用。
- 您的 RDS Proxy 必須位於與資料庫相同的 Virtual Private Cloud (VPC) 中。代理不可以公開存取，但資料庫可以。例如，如果您在本機主機上設計資料庫的原型，除非您設定必要的網路要求以允許與代理的連線，否則無法連線至您的代理。這是因為您的本機主機位於代理的 VPC 之外。

> **Note**
>
> 對於 Aurora 資料庫叢集，您可以開啟跨 VPC 存取。若要完成這個操作，請為 Proxy 建立額外的端點，並使用該端點指定不同的 VPC、子網路和安全群組。如需詳細資訊，請參閱 跨 VPC 存取 Aurora 資料庫。

- 您無法將 RDS Proxy 與其租用設定為 `dedicated` 的 VPC 搭配使用。
- 您無法在`Enforce Mode`已啟用加密控制的 VPC 中使用 RDS Proxy。
- 對於 IPv6 端點網路類型，請將您的 VPC 和子網路設定為僅支援 IPv6。對於 IPv4 和 IPv6 目標連線網路類型，請將您的 VPC 和子網路設定為支援雙重堆疊模式。
- 如果您使用 RDS Proxy 搭配已啟用 IAM 身分驗證的 Aurora 資料庫叢集，代理可以使用 IAM 身分驗證或存放在 Secrets Manager 中的登入資料連線到資料庫。連線至代理的用戶端必須使用 IAM 登入資料進行驗證。如需詳細的組態說明，請參閱 設定 RDS Proxy 的資料庫登入資料 和 設定 RDS Proxy 的 IAM 身分驗證
- 使用 SSL 主機名稱驗證時，您無法將 RDS Proxy 與自訂 DNS 搭配使用。
- 每個代理都可以與單一目標資料庫叢集相關聯。不過，您可以將多個代理與同一個資料庫叢集建立關聯。

- 文字大小大於 16 KB 的任何陳述式會導致代理將工作階段鎖定至目前的連線。
- 某些區域具有要在建立 Proxy 時考量的可用區域 (AZ) 限制。美國東部 (維吉尼亞北部) 區域不支援 `use1-az3` 可用區域中的 RDS Proxy。美國西部 (加利佛尼亞北部) 區域不支援 `usw1-az2` 可用區域中的 RDS Proxy。在建立 Proxy 時，若選取子網路，請確定您未在上述可用區域中選取子網路。
- 目前，RDS Proxy 不支援任何全域條件內容金鑰。

  如需有關全域條件內容索引鍵的詳細資訊，請參閱《IAM 使用者指南》中的 AWS 全域條件內容索引鍵。
- 您無法將 RDS Proxy 與 RDS Custom for SQL Server 搭配使用。
- 若要反映代理的任何資料庫參數群組修改，即使您選擇立即套用變更，仍需要將執行個體重新開機。對於叢集層級參數，需要全叢集重新開機。
- 當您註冊代理目標時，您的代理會自動建立 `rdsproxyadmin` 資料庫使用者。這是受保護的使用者，對於代理功能至關重要。您應避免竄改任何容量中的 `rdsproxyadmin` 使用者。刪除或修改 `rdsproxyadmin` 使用者或其許可，可能會導致您應用程式的代理完全無法使用。

如需每個資料庫引擎的其他限制，請參閱下列各節：

- Aurora MySQL 的其他限制
- Aurora PostgreSQL 的其他限制

Aurora MySQL 的其他限制

下列其他限制適用於具有 Aurora MySQL 資料庫的 RDS Proxy：

- RDS Proxy 對於 `caching_sha2_password` 身分驗證的支援需要安全 (TLS) 連線。
- RDS Proxy 對於 `caching_sha2_password` 的支援已知與特定 go-sql 驅動程式版本存在相容性問題。
- 使用 MySQL 8.4 C 驅動程式時，如果預備陳述式中的參數計數超過預留位置計數，`mysql_stmt_bind_named_param` API 可能會形成格式錯誤的封包。這會導致不正確的回應。如需詳細資訊，請參閱 MySQL 錯誤報告。
- 目前，所有代理都在 MySQL 的連接埠 3306 上接聽。代理仍會使用您在資料庫設定中指定的連接埠來連線至您的資料庫。
- 您不能將 RDS Proxy 用於 EC2 執行個體中的自我管理 MySQL 資料庫。
- 您無法將 RDS Proxy 與其資料庫參數群組中 `read_only` 參數設定為 1 的 RDS for MySQL 資料庫執行個體搭配使用。

- RDS Proxy 不支援 MySQL 壓縮模式。例如，其不支援 `mysql` 命令的 `--compress` 或 `-C` 選項使所用的壓縮。
- 當 RDS Proxy 重複使用相同的資料庫連線來執行另一個查詢時，處理 `GET DIAGNOSTIC` 命令的資料庫連線可能會傳回不正確的資訊。當 RDS Proxy 多工進行資料庫連線時，可能會發生這種情況。
- 某些 SQL 陳述式和函數 (例如 `SET LOCAL`) 可變更連線狀態，而不會導致鎖定。如需最新的鎖定行為，請參閱 避免鎖定 RDS Proxy。
- 不支援在多陳述式查詢中使用 `ROW_COUNT()` 函數。
- RDS Proxy 不支援無法處理單一 TLS 記錄中多個回應訊息的用戶端應用程式。
- RDS Proxy 不支援 MySQL 雙密碼。
- 當您在 RDS 資料庫參數群組中設定 `init_connect` 參數以設定工作階段狀態變數時，RDS Proxy 可能無法如預期般運作。請改為設定代理的初始化查詢，以便在使用代理連線到資料庫時執行工作階段初始化陳述式。

> **Important**
>
> 對於與 MySQL 資料庫相關聯的代理，請勿在初始化查詢中 `sql_auto_is_null` 將配置參數設置為 `true` 或非零值。這樣做可能會導致應用程式行為不正確。

Aurora PostgreSQL 的其他限制

下列其他限制適用於具有 Aurora PostgreSQL 資料庫的 RDS Proxy：

- RDS Proxy 不支援 PostgreSQL 工作階段鎖定篩選條件。
- 目前，所有代理都在 PostgreSQL 的連接埠 5432 上接聽。
- 針對 PostgreSQL，RDS Proxy 目前不支援藉由發出 `CancelRequest` 取消來自用戶端的查詢。例如，使用 Ctrl+C 在互動式 psql 工作階段中取消長時間執行的查詢時，就是這種情況。
- PostgreSQL 函數 lastval 的結果未必永遠準確。您可以採取的因應措施是，將 INSERT 陳述式與 `RETURNING` 子句搭配使用。
- RDS 代理目前不支援串流複寫模式。
- 預設`postgres`資料庫必須存在於 RDS for PostgreSQL 執行個體上，RDS Proxy 才能運作。請勿刪除此資料庫，即使您的應用程式使用不同的資料庫。

- 如果您使用 ALTER ROLE 來變更具有 SET ROLE 的使用者角色，則當使用者與代理的後續連線遇到鎖定時，可能不會使用此角色設定。為了防止這種情況，使用代理時，請在代理的初始化查詢中使用 SET ROLE。如需詳細資訊，請參閱 為 Amazon Aurora 建立代理 中的初始化查詢。

> **Important**
>
> 對於具有 PostgreSQL 資料庫的現有代理，如果您將資料庫驗身分證修改為僅使用 SCRAM，則該代理會變成無法使用，最多持續 60 秒。若要避免發生此問題，請執行下列其中一項：
>
> - 確定資料庫同時允許 SCRAM 和 MD5 身分驗證。
> - 若只要使用 SCRAM 身分驗證，請建立新代理、將您的應用程式流量遷移至新代理，然後刪除先前與資料庫相關聯的代理。

規劃在哪裡使用 RDS Proxy

您可以決定哪些資料庫執行個體、叢集和應用程式可能因為使用 RDS Proxy 而獲益最大。若要執行這項操作，請考慮下列因素：

- 任何遇到「線數太多」錯誤的資料庫叢集都是與代理關聯的理想候選者。其經常的特徵是具有較高的 ConnectionAttempts CloudWatch 指標值。代理讓應用程式能開啟多個用戶端連線，代理則管理較少數量到資料庫叢集的長時間連線。
- 對於使用較小的 AWS 執行個體類別的資料庫叢集，例如 T2 或 T3，使用代理有助於避免out-of-memory的情況。它也可以協助減少建立連線的 CPU 額外負荷。處理大量連線時，可能會發生這些情況。
- 您可以監控特定 Amazon CloudWatch 指標，以判斷資料庫叢集是否接近特定的限制類型。這些限制是指連線數目以及與連線管理相關聯的記憶體。您也可以監控特定 CloudWatch 指標，以決定資料庫叢集是否正在處理許多短期連線。開啟和關閉這類連線可能會對資料庫造成效能負荷。如需欲監控指標的相關資訊，請參閱 使用 Amazon CloudWatch 監控 RDS Proxy 指標。
- AWS Lambda 函數也可以是使用代理的良好候選項目。這些函數經常進行短暫的資料庫連線，這些連線將受益於 RDS Proxy 提供的連線集區。您可以利用 Lambda 函數中已有的任何 IAM 身分驗證，而不是在 Lambda 應用程式的程式碼中管理資料庫登入資料。
- 通常會大量開啟和關閉的資料庫連線，而且沒有內建的連線集區機制的應用程式，都很適合使用 proxy。

- 長期保持大量連線開啟的應用程式通常是搭配代理使用的理想候選者。軟體即服務 (SaaS) 或電子商務等產業中的應用程式通常會保持連線開啟，將資料庫要求的延遲降至最低。相較於直接連線至資料庫執叢集，應用程式可以透過 RDS Proxy 將更多連線保持開啟。
- 由於為所有資料庫叢集設定此類身分驗證的複雜性，您可能尚未採用 IAM 身分驗證和 Secrets Manager。代理可以針對特定應用程式強制執行用戶端連線的身分驗證政策。您可以利用 Lambda 函數中已有的任何 IAM 身分驗證，而不是在 Lambda 應用程式的程式碼中管理資料庫登入資料。
- RDS Proxy 可協助讓應用程式對資料庫失敗更具彈性且更為透明。RDS Proxy 會略過網域名稱系統 (DNS) 快取，將 Aurora Multi-AZ 資料庫的容錯移轉時間縮短高達 66%。RDS Proxy 也會自動將流量路由到新的資料庫執行個體，同時保留應用程式連線。這使得應用程式的容錯移轉更加透明。

RDS Proxy 概念和術語

您可以使用 RDS Proxy 來簡化 Amazon Aurora 資料庫叢集的連線管理。

RDS Proxy 處理用戶端應用程式與資料庫之間的網路流量。它以主動方式達成此功能，首先了解資料庫協定，然後根據據應用程式的 SQL 操作和資料庫中的結果集調整其行為。

RDS Proxy 減輕了資料庫連線管理的記憶體和 CPU 負荷。當應用程式同時開啟許多連線時，資料庫需要的記憶體和 CPU 資源比較少。它關閉並重新開啟長時間閒置的連線時，也不需要依據應用程式的邏輯。同樣地，當資料庫發生問題時，重新建立連線需要的應用程式邏輯也比較少。

RDS Proxy 的基礎設施具備高度可用性，並可部署到數個可用區域 (AZ) 上。RDS Proxy 的運算、記憶體和儲存體與您的 Aurora 資料庫叢集無關。這種分離有助於降低資料庫伺服器的負荷，使它們可以投入資源來服務資料庫工作負載。RDS Proxy 運算資源可在無伺服器的狀態下執行，根據您的資料庫工作負載自動擴展。

主題

- RDS Proxy 概念概觀
- 連線集區
- RDS Proxy 安全性
- 容錯移轉
- 交易

RDS Proxy 概念概觀

RDS Proxy 運用基礎設施來執行連線集區以及以下各節描述的其他功能。您可以在代理頁面上看到 RDS 主控台中所代表的代理。

每個 Proxy 都會處理單一 Aurora 資料庫叢集的連線。Proxy 會自動決定 Aurora 佈建叢集的目前寫入器執行個體。

由代理保持開放，並可供資料庫應用程式使用的連線形成了連線集區。

根據預設，RDS Proxy 可以在工作階段中的每一筆交易結束之後重新使用連線。此一交易層級重複使用稱為多工。當 RDS Proxy 從連線集區暫時移除連線以重複使用時，該操作就是所謂的借用連線。在這麼做是安全的情況下，RDS Proxy 會將該連線傳回至連線集區。

在某些情況下，RDS Proxy 無法確定在目前工作階段之外重複使用資料庫連線是否安全。這時，它會讓工作階段保持在同一連線上，直到該工作階段結束。這種回復行為稱為鎖定。

代理具有預設端點。當使用 Amazon Aurora 資料庫叢集時，您會連線到此端點。您會這樣做，而不是連線到直接連線至叢集的讀取/寫入端點。Aurora 叢集的特殊用途端點仍可供您使用。對於 Aurora 資料庫叢集，您也可以建立額外的讀取/寫入和唯讀端點。如需詳細資訊，請參閱代理端點概觀。

例如，您仍然可以連線到叢集端點以進行讀取/寫入連線，而不需要連線集區。您仍然可以連線至讀取器端點，以進行唯讀連線的負載平衡。您仍然可以連線到執行個體端點，以便對叢集內的特定資料庫執行個體進行診斷和疑難排解。如果您使用其他 AWS 服務 AWS Lambda ，例如 連接到 RDS 資料庫，請變更其連線設定以使用代理端點。例如，您可以指定代理端點，以允許 Lambda 函數存取您的資料庫，同時利用 RDS Proxy 功能。

每個代理都包含一個目標群組。此目標群組體現了 Proxy 可以連線的 Aurora 資料庫叢集。根據預設，Aurora 叢集的目標群組與該叢集中所有資料庫執行個體相關聯。這樣一來，Proxy 可以連線到叢集中被提升為寫入器執行個體的任何 Aurora 資料庫執行個體。與 Proxy 相關聯的 Aurora 資料庫叢集稱為該 Proxy 的目標。為了方便起見，當您透過主控台建立代理伺服器時，RDS Proxy 也會建立對應的目標群組，並自動註冊關聯的目標。

引擎系列是一組使用相同資料庫協定的相關資料庫引擎。您可以為您建立的每一個代理選擇引擎系列。

連線集區

每個 Proxy 都會針對其相關聯 Aurora 資料庫的寫入器和讀取器執行個體個別執行連線集區。連線集區是一種最佳化，可以減輕開啟和關閉連線，以及保持許多連線同時開啟的相關負荷。此一負荷包

含處理每個新連線所需的記憶體。它還涉及關閉每個連線和開啟新連線的 CPU 額外負荷。範例包括 Transport Layer Security/Secure Sockets Layer (TLS/SSL) 交握、身分驗證、協商功能等。連線集區簡化了應用程式的邏輯。您不需要撰寫應用程式碼，以便同時開啟的連線數降到最低。

每個 Proxy 也會執行連線多工，又稱為連線重用。RDS Proxy 運用「多工」，使用一個基礎資料庫連線執行一筆交易的所有操作。然後，RDS 可以使用不同的連線處理下一筆交易。您可以同時開啟到代理的多個連線，代理則使較少的資料庫執行個體或叢集連線保持開啟。這樣做可以進一步減輕資料庫伺服器上連線的記憶體負荷。此技術也可以降低發生「連線數太多」錯誤的可能性。

RDS Proxy 安全性

RDS Proxy 會使用現有的 RDS 安全機制，例如 TLS/SSL 和 AWS Identity and Access Management (IAM)。如需這些安全功能的一般資訊，請參閱 Amazon Aurora 的安全性。此外，請務必熟悉 Aurora 如何使用身分驗證、授權和其他安全領域。

RDS Proxy 可以充當用戶端應用程式和基礎資料庫之間的額外安全層。例如，即使基礎資料庫執行個體支援舊版 TLS，您仍可使用 TLS 1.3 連線至代理。即使代理使用資料庫使用者和密碼身分驗證方法連線到資料庫，您也可以使用 IAM 角色連線到代理。您可以使用這種技術，強制資料庫應用程式執行嚴格的身分驗證要求，而無需在資料庫執行個體上進行昂貴的遷移工作。

您可以搭配 RDS Proxy 使用下列身分驗證方法：

- 資料庫登入資料
- 標準 IAM 身分驗證
- End-to-end IAM 身分驗證

搭配 RDS Proxy 使用 IAM

RDS Proxy 提供兩種 IAM 身分驗證方法：

- 標準 IAM 身分驗證：在代理使用存放在 Secrets Manager 中的登入資料連線到資料庫時，對代理的連線強制執行 IAM 身分驗證。即使資料庫使用原生密碼身分驗證，這也會強制執行資料庫存取的 IAM 身分驗證。代理會從 Secrets Manager 擷取資料庫憑證，並代表您的應用程式處理對資料庫的身分驗證。
- End-to-end IAM 身分驗證：針對直接從應用程式透過代理連線到資料庫的連線，強制執行 IAM 身分驗證。End-to-end IAM 身分驗證可簡化您的安全組態，並避免在 Secrets Manager 中管理資料庫登入資料。這一層額外的安全層會強制執行從用戶端應用程式到資料庫的 IAM 型存取控制。

若要使用標準 IAM 身分驗證，請將您的代理設定為使用 Secrets Manager 秘密進行身分驗證，並為用戶端連線啟用 IAM 身分驗證。您的應用程式使用 IAM 向代理進行身分驗證，而代理使用從 Secrets Manager 擷取的登入資料向資料庫進行身分驗證。

若要使用end-to-end IAM 身分驗證，請在建立或修改代理時設定預設身分驗證機制時，將代理設定為使用 IAM 身分驗證。

對於end-to-end IAM 身分驗證，您必須更新與代理相關聯的 IAM 角色，以授予 `rds-db:connect`許可。透過end-to-end IAM 身分驗證，您不需要透過 Secrets Manager 秘密向代理註冊個別資料庫使用者。

搭配 RDS Proxy 使用 TLS/SSL

您可以使用 TLS/SSL 通訊協定連線到 RDS Proxy。

> **Note**
>
> RDS Proxy 使用來自 AWS Certificate Manager (ACM) 的憑證。如果您正在使用 RDS Proxy，您不需要下載 Amazon RDS 憑證或更新使用 RDS Proxy 連線的應用程式。

若要針對代理與資料庫之間的所有連線強制執行 TLS，您可以在 AWS 管理主控台中建立或修改代理時，指定需要 Transport Layer Security 設定。

RDS Proxy 也能確保您的工作階段在用戶端與 RDS Proxy 端點之間使用 TLS/SSL。若要讓 RDS Proxy 執行此作業，請在用戶端指定需求。SSL 工作階段變數不會針對使用 RDS Proxy 之資料庫的 SSL 連線進行設定。

- 若為 Aurora MySQL，請在執行 `mysql` 命令時，使用 `--ssl-mode` 參數在用戶端指定需求。
- 若為 和 Aurora PostgreSQL，請在執行 `psql` 命令時，指定 `sslmode=require` 做為 `conninfo` 字串的一部分。

RDS Proxy 支援 TLS 通訊協定 1.0、1.1、1.2 和 1.3 版。您可以使用比基礎資料庫中所使用之 TLS 更新的版本連線至代理。

根據預設，用戶端程式會使用 RDS Proxy 建立加密連線，並且可透過 `--ssl-mode` 選項取得進一步的控制。RDS Proxy 在用戶端支援所有 SSL 模式。

針對用戶端，SSL 模式如下：

PREFERRED

SSL 為第一個選擇，但並非必要。

DISABLED

不允許任何 SSL。

REQUIRED

強制採用 SSL。

VERIFY_CA

強制採用 SSL 並驗證憑證授權單位 (CA)。

VERIFY_IDENTITY

強制採用 SSL 並驗證 CA 和 CA 主機名稱。

在搭配 `--ssl-mode VERIFY_CA` 或 `VERIFY_IDENTITY` 使用用戶端時，請指定指向 `--ssl-ca` 格式 CA 的 `.pem` 選項。對於要使用的 `.pem` 檔案，請從 Amazon Trust Services 下載所有根 CA PEM 並將它們放入單一 `.pem` 檔案。

RDS Proxy 使用萬用字元憑證，此憑證同時套用至網域及其子網域。如果您使用 `mysql` 用戶端來利用 SSL 模式 `VERIFY_IDENTITY` 連接，則目前必須使用 MySQL 8.0 相容的 `mysql` 命令。

容錯移轉

容錯移轉是一種高可用性功能，當原始執行個體無法使用時，用另一個來取代該資料庫執行個體。發生容錯移轉可能是因為資料庫執行個體發生問題。它也可能是正常維護程序的一部分，例如在資料庫升級期間。容錯移轉適用於除了寫入器執行個體之外還有一或多個讀取器執行個體的 Aurora 資料庫叢集。

透過代理進行連線可讓您的應用程式對於資料庫容錯移轉更有彈性。當原始資料庫執行個體無法使用時，RDS Proxy 會連線至待命資料庫，而不會捨棄閒置的應用程式連線。這樣有助於加速並簡化容錯移轉程序。這對應用程式的干擾也比一般的重新啟動或資料庫問題更少。

如果沒有 RDS Proxy，容錯移轉將導致短暫的執行中斷。在中斷期間，您無法對容錯移轉中的資料庫執行寫入操作。任何現有的資料庫連線都會中斷，您的應用程式必須將其重新開啟。當提升唯讀資料庫執行個體，以取代無法使用的執行個體時，該資料庫將可使用於新連線和寫入操作。

在資料庫容錯移轉期間，RDS Proxy 繼續接受位於相同 IP 地址的連線，並自動將連線導向至新的主要資料庫執個體。透過 RDS Proxy 連線的用戶端不會受到下列項目影響：

- 容錯移轉時的網域名稱系統 (DNS) 傳播延遲。
- 本機 DNS 快取。
- 連線逾時。
- 不確定哪個資料庫執行個體是目前的寫入器。
- 等候來自已不可使用，卻未關閉連線的前寫入器的查詢回應。

對於自行維護連線集區的應用程式而言，經歷 RDS Proxy 意味著大部分的連線在容錯移轉或其他中斷期間保持活動狀態。只會取消處於交易或 SQL 陳述式中間的連線。RDS Proxy 會立即接受新的連線。當資料庫寫入器無法使用時，RDS Proxy 會將傳入的請求排入佇列。

對於不自行維護連線集區的應用程式，RDS Proxy 提供了更快的連線速率和更多的開放連線。它免除了從資料庫頻繁重新連接的昂貴負荷。它會重複使用 RDS Proxy 的連線集區中維護的資料庫連線，而達成此功能。此方法對於 TLS 連線來說尤其重要，因為設定成本很大。

交易

單一交易中的所有陳述式總是使用相同的基礎資料庫連線。當交易結束時，連線將可提供給不同的工作階段使用。使用交易做為粒度單位會產生下列後果：

- 開啟 Aurora MySQL `autocommit` 設定時，連線可能會在每一個別陳述式之後重複使用。
- 相反地，`autocommit` 設定關閉時，您在工作階段中發出的第一個陳述式會開始新的交易。例如，假設您輸入一連串的 `SELECT`、`INSERT`、`UPDATE` 和其他資料操作語言 (DML) 陳述式。在此情況下，直到您發出 `COMMIT`、`ROLLBACK` 或以另外方式結束交易之前，都不會重複使用連線。
- 輸入資料定義語言 (DDL) 陳述式將使交易在陳述式完成後結束。

RDS Proxy 透過資料庫用戶端應用程式使用的網路協定偵測交易何時結束。交易偵測並不依賴 SQL 陳述式的文字中出現的 `COMMIT` 或 `ROLLBACK` 等關鍵字。

在某些情況下，RDS Proxy 可能會偵測到資料庫要求，使得將工作階段移到不同的連線變得不切實際。這時，它會在剩餘的工作階段關閉該連線的多工功能。如果 RDS Proxy 無法確定多工在此工作階段中實際可行，則適用相同的規則。此操作稱為鎖定。如需偵測和最小化鎖定的方法，請參閱 避免鎖定 RDS Proxy。

RDS Proxy 入門

使用以下頁面中的資訊來設定和管理 Amazon RDS Proxy for Aurora 並設定相關的安全性選項。安全性選項會控制誰可以存取每個代理，以及每個代理如何連線至資料庫執行個體。

如果您是初次使用 RDS Proxy，建議您依照我們呈現的順序遵循頁面。

主題

- 設定 RDS Proxy 的網路先決條件
- 設定 RDS Proxy 的資料庫登入資料
- 設定 RDS Proxy 的 IAM 身分驗證
- 為 Amazon Aurora 建立代理
- 檢視代理
- 透過 RDS Proxy 連線至資料庫

設定 RDS Proxy 的網路先決條件

使用 RDS Proxy 需要您在 Aurora 資料庫叢集與 RDS Proxy 之間擁有通用的虛擬私有雲端 (VPC)。此 VPC 應至少有兩個位於不同可用區域的子網路。您的帳戶可以擁有這些子網路，或與其他帳戶共用。如需 VPC 共用的詳細資訊，請參閱使用共用 VPC。

對於 IPv6 支援，需要額外的網路組態：

- IPv6 端點網路類型 – 您的 VPC 和子網路必須設定為支援 IPv6。這包括將 IPv6 CIDR 區塊指派給您的 VPC 和子網路。
- 雙堆疊端點網路類型 – 您的 VPC 和子網路必須同時支援 IPv4 和 IPv6 定址。
- IPv6 目標連線網路類型 – 資料庫必須設定為雙堆疊模式，以支援來自代理的 IPv6 連線。

您的用戶端應用程式資源 (例如 Amazon EC2、Lambda 或 Amazon ECS) 可以位於與代理相同的 VPC 中。或者，它們可以位於與代理不同的 VPC 中。如果您已成功連線到任何 Aurora 資料庫叢集，表示您已擁有所需的網路資源。

主題

- 取得子網路的相關資訊
- 規劃 IP 地址容量

取得子網路的相關資訊

如果您剛開始使用 Aurora，您可以依照 設定您的 Amazon Aurora 環境 中的程序了解連線至資料庫的基本知識。您也可以按照 Amazon Aurora 入門 中的教學課程進行操作。

若要建立代理，您必須提供代理操作所在的子網路和 VPC。下列 Linux 範例顯示的 AWS CLI 命令會檢查 擁有VPCs 和子網路 AWS 帳戶。特別是，當您使用 CLI 建立代理時，您會傳遞子網路 ID 做為參數。

```
aws ec2 describe-vpcs
aws ec2 describe-internet-gateways
aws ec2 describe-subnets --query '*[].[VpcId,SubnetId]' --output text | sort
```

下列 Linux 範例顯示 AWS CLI 命令，以判斷與特定 Aurora 資料庫叢集對應的子網路 IDs。

對於 Aurora 叢集，首先您會找到其中一個相關資料庫執行個體的 ID。您可以擷取該資料庫執行個體使用的子網路 ID。若要這樣做，請在資料庫執行個體描述輸出中檢查 DBSubnetGroup 和 Subnets 屬性內的巢狀欄位。設定該資料庫伺服器的代理時，您可以指定部分或全部的子網路 ID。

```
$ # Find the ID of any DB instance in the cluster.
$  aws rds describe-db-clusters --db-cluster-identifier my_cluster_id --query '*[].
[DBClusterMembers]||[0]||[0][*].DBInstanceIdentifier' --output text
```

```
my_instance_id
instance_id_2
instance_id_3
```

在找到資料庫執行個體識別碼之後，請檢查相關聯的 VPC 來尋找其子網路。下列 Linux 範例會顯示做法。

```
$ #From the DB instance, trace through the DBSubnetGroup and Subnets to find the subnet
 IDs.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]||[0]||[0]||[Subnets]||[0]||[*].SubnetIdentifier' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
...
```

```
$ #From the DB instance, find the VPC.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]||[0]||[0].VpcId' --output text
```

```
my_vpc_id
```

```
$ aws ec2 describe-subnets --filters Name=vpc-id,Values=my_vpc_id --query '*[].
[SubnetId]' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
subnet_id_4
subnet_id_5
subnet_id_6
```

規劃 IP 地址容量

RDS Proxy 會根據向它註冊的資料庫執行個體組態，自動調整其容量。對於佈建的執行個體，這取決於執行個體大小，對於Aurora Serverless v2執行個體，這取決於最大 ACU 容量。某些操作可能還需要額外的容量，例如增加已註冊資料庫的大小或內部 RDS Proxy 維護操作。在這些操作期間，您的代理可能需要更多的 IP 地址來佈建額外的容量。這些額外的位址可讓您的代理擴展，而不會影響您的工作負載。子網路中若少了可用的 IP 地址會阻止代理縱向擴展。這可能會導致更高的查詢延遲或用戶端連線失敗。當您的子網路中沒有足夠的可用 IP 地址時，RDS 會透過事件 RDS-EVENT-0243 通知您。如需此金鑰的相關資訊，請參閱 使用 RDS Proxy 事件。

以下是根據資料庫執行個體類別大小，建議在子網路中為您的代理保留可用的 IP 位址數目下限。

DB instance class (資料庫執行個體類別)	可用 IP 地址數目下限
db.*.xlarge 或更小	10
db.*.24xlarge	15
db.*.24xlarge	25
db.*.24xlarge	45
db.*.24xlarge	60
db.*.24xlarge	75
db.*.24xlarge	110

對於 Aurora Serverless v2，請根據最大 ACU 容量，在子網路中為代理保留下列最小數量的可用 IP 地址。

Maximum ACU Capacity	可用 IP 地址數目下限
16 或更小	10
32	15
64	25
96	30
128	40
160	50
192	55
224	65
256	75

> **Note**
>
> RDS Proxy 不會為 VPC 中的每個代理耗用超過 215 個 IP 地址。

RDS Proxy 需要您 Aurora 資料庫至少 10 個 IP 地址。這些建議 IP 地址數量是僅具有預設端點的代理預估值。對於每個額外的自訂端點，我們建議再保留三個 IP 地址。對於每個 Aurora 讀取器執行個體，我們建議您根據該讀取器的目標ACUs 或佈建Aurora Serverless v2目標的資料庫執行個體大小，保留資料表中指定的其他 IP 地址。

若要估計與 Aurora 資料庫叢集相關聯的代理所需的 IP 地址：

- 1 個大小為 的佈建寫入器執行個體`db.r5.8xlarge`和 1 個大小為 的佈建讀取器執行個體`db.r5.2xlarge`。
- 連接到此叢集的代理具有預設端點，以及具有唯讀角色的自訂端點。

在此情況下，代理大約需要 63 個可用 IP 地址 (45 個用於寫入器執行個體，15 個用於讀取器執行個體，3 個用於額外的自訂端點)。

若要估計與具有下列項目的 Aurora 資料庫叢集相關聯的代理所需的 IP 地址：

- 1 個Aurora Serverless v2寫入器執行個體的最大容量為 256 個 ACUs1 個 Serverless v2 讀取器執行個體的最大容量為 192 ACUs。
- 連接到此叢集的代理具有預設端點和 1 個具有唯讀角色的自訂端點。

在此情況下，代理需要大約 133 個可用的 IP 地址（寫入器執行個體 75 個，讀取器執行個體 55 個，其他自訂端點 3 個）。

若要估計與具有下列項目的 Aurora 叢集相關聯的代理所需的 IP 地址：

- 資料庫執行個體大小為 db.r5.4xlarge 的 1 個佈建寫入器執行個體，以及容量上限為 64 個 ACUs 的 1 個 Serverless v2 讀取器執行個體。
- 連接到此叢集的代理具有預設端點和 1 個具有唯讀角色的自訂端點。

在此情況下，代理需要大約 53 個可用的 IP 地址（寫入器執行個體 25 個，讀取器執行個體 25 個，其他自訂端點 3 個）。

若要估計與具有下列項目的 Aurora 資料庫叢集相關聯的代理所需的 IP 地址：

- 1 個大小為 db.r5.24xlarge 的佈建寫入器執行個體，以及 3 個大小為 db.r5.8xlarge 的佈建讀取器執行個體。
- 附加到此資料庫叢集的代理具有預設端點和 1 個具有唯讀角色的自訂端點。

在此情況下，代理需要 215 個可用的 IP 地址。雖然計算顯示 248 IPs (110 + (3*45) + 3)，但 RDS Proxy 對於 VPC 中的每個代理不會耗用超過 215 個 IP 地址。

設定 RDS Proxy 的資料庫登入資料

Amazon RDS 中的 RDS Proxy 會使用 AWS Secrets Manager 來安全地存放和管理資料庫登入資料。您可以將代理與包含必要身分驗證詳細資訊的 Secrets Manager 秘密建立關聯，而不是在應用程式中內嵌憑證。您在 Aurora 資料庫叢集上為代理連線的每個資料庫使用者帳戶建立個別的 Secrets Manager 秘密。

或者，您可以將 RDS Proxy 設定為使用end-to-end IAM 身分驗證，這樣就不需要在 Secrets Manager 中存放資料庫登入資料。RDS Proxy 會針對client-to-proxy連線和 proxy-to-database連線使用 IAM 身分驗證。這提供完全整合的 IAM 型身分驗證解決方案，不需要管理秘密或密碼。如需新增 IAM 資料庫使用者的相關資訊，請參閱 使用 IAM 身分驗證建立資料庫帳戶。

主題

- 建立要與 RDS Proxy 搭配使用的秘密

建立要與 RDS Proxy 搭配使用的秘密

建立代理之前，您必須先建立至少一個存放資料庫憑證的秘密。

主控台

若要建立機密

1. 請開啟位於 https://console.aws.amazon.com/secretsmanager/ 的機密管理員控制台。
2. 選擇 Store a new secret (存放新機密)。
3. 選擇 Amazon RDS 資料庫的憑證。
4. 輸入使用者名稱和密碼。您輸入的憑證必須符合存在於相關聯 RDS 資料庫中資料庫使用者的憑證。RDS Proxy 使用這些憑證來代表應用程式驗證和建立與資料庫的連線。

   如果不相符，您可以更新秘密以符合資料庫密碼。在您更新秘密之前，嘗試使用該秘密透過代理連線會失敗，但使用其他有效秘密的連線仍然可運作。

   > **Note**
   >
   > 對於 RDS for SQL Server，RDS Proxy 需要在 Secrets Manager 中設定區分大小寫的秘密，無論資料庫執行個體定序設定為何。如果您的應用程式允許具有不同大小寫的使用者名稱，例如 "Admin" 和 "admin"，您必須為每個使用者名稱建立個別秘密。RDS Proxy 不支援用戶端和代理之間的不區分大小寫使用者名稱身分驗證。
   > 如需有關 SQL Server 中定序的詳細資訊，請參閱 Microsoft SQL Server 文件。

5. 對於資料庫，請選取秘密將會存取的 Amazon RDS 資料庫。
6. 填入秘密的其他設定，然後選擇儲存。如需完整指示，請參閱《AWS Secrets Manager 使用者指南》中的建立 AWS Secrets Manager 秘密。

AWS CLI

當您透過 建立代理時 AWS CLI，您可以指定對應秘密的 Amazon Resource Name (ARNs)。您可以針對代理可以存取的所有資料庫使用者帳戶執行此操作。在 中 AWS 管理主控台，您可以依秘密的描述性名稱來選擇秘密。

- 若要建立 Secrets Manager 秘密以搭配 RDS Proxy 使用，請使用 create-secret 命令：

```
aws secretsmanager create-secret \
  --name "secret_name" \
  --description "secret_description" \
  --region region_name \
  --secret-string '{"username":"db_user","password":"db_user_password"}'
```

- 您也可以建立自訂金鑰來加密 Secrets Manager 秘密。以下命令會建立範例金鑰。

```
aws kms create-key --description "test-key" --policy '{
  "Id":"kms-policy",
  "Version": "2012-10-17",
  "Statement":
    [
      {
        "Sid":"Enable IAM User Permissions",
        "Effect":"Allow",
        "Principal":{"AWS":"arn:aws:iam::account_id:root"},
        "Action":"kms:*","Resource":"*"
      },
      {
        "Sid":"Allow access for Key Administrators",
        "Effect":"Allow",
        "Principal":
          {
            "AWS":
              ["$USER_ARN","arn:aws:iam:account_id::role/Admin"]
          },
        "Action":
          [
            "kms:Create*",
            "kms:Describe*",
            "kms:Enable*",
            "kms:List*",
            "kms:Put*",
            "kms:Update*",
```

```
            "kms:Revoke*",
            "kms:Disable*",
            "kms:Get*",
            "kms:Delete*",
            "kms:TagResource",
            "kms:UntagResource",
            "kms:ScheduleKeyDeletion",
            "kms:CancelKeyDeletion"
          ],
        "Resource":"*"
      },
      {
        "Sid":"Allow use of the key",
        "Effect":"Allow",
        "Principal":{"AWS":"$ROLE_ARN"},
        "Action":["kms:Decrypt","kms:DescribeKey"],
        "Resource":"*"
      }
    ]
}'
```

例如，下列命令會為兩個資料庫使用者建立 Secrets Manager 秘密：

```
aws secretsmanager create-secret \
  --name secret_name_1 --description "db admin user" \
  --secret-string '{"username":"admin","password":"choose_your_own_password"}'

aws secretsmanager create-secret \
  --name secret_name_2 --description "application user" \
  --secret-string '{"username":"app-user","password":"choose_your_own_password"}'
```

若要建立以自訂 AWS KMS 金鑰加密的秘密，請使用下列命令：

```
aws secretsmanager create-secret \
  --name secret_name_1 --description "db admin user" \
  --secret-string '{"username":"admin","password":"choose_your_own_password"}' \
  --kms-key-id arn:aws:kms:us-east-2:account_id:key/key_id

aws secretsmanager create-secret \
  --name secret_name_2 --description "application user" \
  --secret-string '{"username":"app-user","password":"choose_your_own_password"}' \
```

```
  --kms-key-id arn:aws:kms:us-east-2:account_id:key/key_id
```

若要查看您 AWS 帳戶擁有的秘密，請使用 list-secrets 命令：

```
aws secretsmanager list-secrets
```

當您使用 CLI 建立代理時，您會將一個或多個秘密的 Amazon Resource Names (ARN) 傳遞給 `--auth` 參數。下列範例顯示如何僅使用 AWS 帳戶擁有的每個秘密名稱和 ARN 來準備報告。此範例使用第 2 AWS CLI 版中可用的 `--output table` 參數。如果您使用的是第 1 AWS CLI 版，請`--output text`改用 。

```
aws secretsmanager list-secrets --query '*[].[Name,ARN]' --output table
```

若要確認秘密包含正確格式的正確憑證，請使用 get-secret-value 命令。將 *your_secret_name* 取代為秘密的簡短名稱或 ARN。

```
aws secretsmanager get-secret-value --secret-id your_secret_name
```

輸出包含具有類似下列項目的 JSON 編碼值的行：

```
...
"SecretString": "{\"username\":\"your_username\",\"password\":\"your_password\"}",
...
```

設定 RDS Proxy 的 IAM 身分驗證

若要在 Amazon RDS 中設定 RDS Proxy 的 AWS Identity and Access Management (IAM) 身分驗證，請建立並設定授予必要許可的 IAM 政策。

本主題提供設定 RDS Proxy IAM 身分驗證的步驟，包括建立必要的 IAM 政策並將其連接至 IAM 角色。

> **Tip**
>
> 只有在您想要建立自己的 IAM 角色時，才需要此程序。否則，RDS 可以在設定代理時自動建立必要的角色，因此您可以略過這些步驟。

先決條件

設定 RDS Proxy 的 IAM 身分驗證之前，請確定您有下列項目：

- AWS Secrets Manager – 至少有一個存放的秘密，其中包含資料庫憑證。如需如何建立秘密的指示，請參閱 the section called “設定資料庫憑證”。

  如果您使用end-to-end 身分驗證，則不需要這麼做。
- IAM 許可 – 具有在 AWS Secrets Manager中建立及管理 IAM 政策、角色和秘密之許可的 IAM 角色或使用者。

為end-to-end IAM 身分驗證建立 IAM 政策

使用end-to-end IAM 身分驗證時，RDS Proxy 會使用 IAM 身分驗證連線至資料庫，而不是從 Secrets Manager 擷取登入資料。這需要為要與代理搭配使用的資料庫帳戶設定具有`rds-db:connect`許可的 IAM 角色。

若要使用 IAM 向資料庫驗證 RDS Proxy，請使用授予必要資料庫連線許可的政策來建立 IAM 角色。

主控台

使用代理建立end-to-end IAM 身分驗證的角色

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 建立角色的許可政策。如需一般步驟，請參閱建立 IAM 政策 (主控台)。

   將此政策貼到 JSON 編輯器，並進行以下變更：。

   - 替換為您自己的帳戶 ID。
   - `us-east-2` 以代理必須所在的 取代 。
   - 將資料庫資源 IDs和使用者名稱替換成您要使用的資料庫資源 ID 和使用者名稱。RDS 執行個體和 之間的資源 ID 格式不同Aurora clusters。

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

```
            "Resource": [
                "arn:aws:rds-db:us-
east-2:account_id:dbuser:db_instance_resource_id/db_user_name_1",
                "arn:aws:rds-db:us-
east-2:account_id:dbuser:db_instance_resource_id/db_user_name_2"
            ]
        }
    ]
}
```

3. 建立角色並將許可政策連接至其中。如需一般步驟，請參閱建立角色以將許可委派給 AWS 服務。

   對於信任的實體類型，選擇 AWS 服務。在使用案例底下，選取 RDS，並為使用案例選擇 RDS - 將角色新增至資料庫。

4. 對於許可政策，選擇您建立的政策。
5. 對於選取受信任的實體，輸入角色的下列信任政策：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

AWS CLI

若要使用 建立角色 AWS CLI，請傳送下列請求：

```
aws iam create-role \
  --role-name my_e2e_iam_role_name \

  --assume-role-policy-document '{"Version":"2012-10-17",
     "Statement":[{"Effect":"Allow","Principal":{"Service":
["rds.amazonaws.com"]},"Action":"sts:AssumeRole"}]}'
```

接著，將政策連接至該角色：

```
aws iam put-role-policy \
  --role-name my_e2e_iam_role_name \
  --policy-name e2e_iam_db_connect_policy \
  --policy-document '{

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rds-db:connect",
            "Resource": [
                "arn:aws:rds-db:us-
east-2:account_id:dbuser:db_instance_resource_id/db_user_name_1",
                "arn:aws:rds-db:us-
east-2:account_id:dbuser:db_instance_resource_id/db_user_name_2"
            ]
        }
    ]
}'
```

設定end-to-end IAM 身分驗證的 IAM 角色和許可後，您現在可以建立將 `DefaultAuthScheme` 設為的代理`IAM_AUTH`。此代理會使用 IAM 直接驗證資料庫，而不需要 Secrets Manager 秘密。如需說明，請參閱the section called “建立代理”。

使用end-to-end IAM 身分驗證時，請確定您的資料庫使用者已設定為 IAM 身分驗證，如中所述使用 IAM 身分驗證建立資料庫帳戶。

建立 Secrets Manager 存取的 IAM 政策

若要允許 RDS Proxy 從 Secrets Manager 擷取資料庫憑證，請使用可授予必要許可的政策建立 IAM 角色。

主控台

建立可存取私密的角色以與代理搭配使用

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 建立角色的許可政策。如需一般步驟，請參閱建立 IAM 政策 (主控台)。

將此政策貼到 JSON 編輯器，並進行以下變更：。

- 替換為您自己的帳戶 ID。
- 將 us-east-2 替代為代理所在的區域。
- 將秘密名稱替代為您建立的秘密名稱。如需詳細資訊，請參閱在 IAM 政策陳述式中指定 KMS 金鑰。
- 將 KMS 金鑰 ID 替代為您用來加密 Secrets Manager 秘密的金鑰 ID，可以是預設金鑰或您自己的金鑰。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-
east-2:111122223333:secret:secret_name_1",
                "arn:aws:secretsmanager:us-
east-2:111122223333:secret:secret_name_2"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-2:111122223333:key/key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "secretsmanager.us-
east-2.amazonaws.com"
                }
            }
        }
    ]
}
```

3. 建立角色並將許可政策連接至其中。如需一般步驟，請參閱建立角色以將許可委派給 AWS 服務。

對於信任的實體類型，選擇 AWS 服務。在使用案例底下，選取 RDS，並為使用案例選擇 RDS - 將角色新增至資料庫。

4. 對於許可政策，選擇您建立的政策。
5. 對於選取受信任的實體，輸入角色的下列信任政策：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

AWS CLI

若要使用 建立角色 AWS CLI，請傳送下列請求：

```
aws iam create-role \
  --role-name my_role_name \
  --assume-role-policy-document '{"Version": "2012-10-17",
      "Statement":[{"Effect":"Allow","Principal":{"Service":
["rds.amazonaws.com"]},"Action":"sts:AssumeRole"}]}'
```

接著，將政策連接至該角色：

```
aws iam put-role-policy \
  --role-name my_role_name \
  --policy-name secret_reader_policy \
  --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_1",
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_2"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-2:account_id:key/key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "secretsmanager.us-east-2.amazonaws.com"
                }
            }
        }
    ]
}'
```

設定 IAM 角色和許可後，您現在可以建立代理並將其與此角色建立關聯。這可讓代理從 安全地擷取資料庫憑證， AWS Secrets Manager 並為您的應用程式啟用 IAM 身分驗證。如需說明，請參閱the section called “建立代理”。

為 Amazon Aurora 建立代理

您可以使用 Amazon RDS Proxy，透過更有效率地為連線建立集區和管理資料庫容錯移轉，來改善資料庫應用程式的可擴展性、可用性和安全性。這個主題將逐步說明建立代理的步驟。開始之前，請確定您的資料庫符合必要的先決條件，包括 IAM 許可和 VPC 組態。

您可以將代理關聯至 Aurora MySQL 資料庫執行個體或 Aurora PostgreSQL 資料庫執行個體叢集。

主控台

若要建立代理

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Proxies (代理)。

3. 選擇Create Proxy (建立代理)。
4. 針對您的代理使用以下設定。

設定	說明
引擎系列	在解譯往返資料庫的網路流量時，代理會辨識哪些資料庫網路通訊協定。 **Note** 若要使用 Aurora PostgreSQL，請務必將 `postgres` 資料庫保留在您的執行個體中。請參閱 針對已刪除的 `postgres` 資料庫進行疑難排解。
代理識別符	AWS 帳戶 ID 和目前 AWS 區域中唯一的名稱。
閒置用戶端連線逾時	如果用戶端連線在設定的期間內保持閒置狀態，代理會關閉用戶端連線。根據預設，這是 1,800 秒 (30 分鐘)。完成上一個請求之後若應用程式未在指定時間內提交新請求，連線就會被視為閒置。代理會保持基礎資料庫連線開啟，並將其傳回至連線集區，使其可用於新的用戶端連線。 若要主動移除過時的連線，請減少閒置用戶端連線逾時。若要將工作負載尖峰期間的連線成本降至最低，請增加逾時。
資料庫	要透過此代理存取的 Aurora 資料庫叢集。此清單僅包含具有相容資料庫引擎、引擎版本和其他設定的資料庫執行個體和叢集。如果清單是空的，請建立與 RDS Proxy 相容的新資料庫執行個體或叢集。若要執行此作業，請依照 建立 Amazon Aurora 資料庫叢集 中的程序進行。然後嘗試再次建立代理。
連線集區最大連線數	介於 1 到 100 之間的值，定義 RDS Proxy 可以使用的 `max_connections` 限制百分比。如果您只打算在此資料庫執行個體或叢集中使用一個代理，請將此值設定為 100。如需有關 RDS Proxy 如何使用此設定的詳細資訊，請參閱 the section called “MaxConnectionsPercent”。

設定	說明
工作階段鎖定篩選條件	防止 RDS Proxy 鎖定某些偵測到的工作階段狀態，這會略過多工連線的預設安全措施。目前，PostgreSQL 不支援此設定，且唯一可用的選項是 EXCLUDE_VARIABLE_SETS 。啟用它可能會導致來自一個連線的工作階段變數影響其他連線，如果查詢依賴目前交易外部設定的工作階段變數，則會導致錯誤或正確性問題。只有在確認您的應用程式可以安全地共用資料庫連線之後，才使用此選項。 以下模式可視為安全： • SET 陳述式，其中有效的工作階段變數值沒有變更。換句話說，工作階段變數沒有變更。 • 您會在同一交易中變更工作階段變數值和執行陳述式。 如需詳細資訊，請參閱避免鎖定 RDS Proxy。
連線借用逾時	如果您預期代理使用所有可用的資料庫連線，請設定傳回逾時錯誤之前的等待時間。您最多可以指定五分鐘。只有在代理的連線數已達上限，而且所有連線都已在使用中時，才會套用此設定。

設定	說明
初始化查詢	(選用) 新增初始化查詢，或修改目前的查詢。開啟每個新資料庫連線時，您可為要執行的代理指定一或多個 SQL 陳述式。該設定通常與 SET 陳述式搭配使用，確保每個連線都有相同的設定。請確定您新增的查詢有效。若要在單一 SET 陳述式中包含多個變數，請使用逗號分隔符號。例如： `SET variable1 =value1, variable2 =value2` 對於多個陳述式，請使用分號作為分隔符號。 **Important** 由於您可以在目標群組組態中存取初始化查詢，因此不受身分驗證或密碼編譯方法的保護。有權檢視或管理代理目標群組組態的任何人都可以檢視初始化查詢。您不應將密碼或長期加密金鑰等敏感資料新增至此選項。
AWS Identity and Access Management (IAM) 角色	具有許可可存取 Secrets Manager 秘密的 IAM 角色，代表代理可以使用的資料庫使用者帳戶憑證。或者，您可以從 AWS 管理主控台建立新的 IAM 角色。
Secrets Manager 秘密	建立或選擇代表可使用代理之資料庫使用者帳戶登入資料的 Secrets Manager 秘密。 當預設身分驗證機制設定為無時，此欄位為必要欄位。當預設身分驗證機制設定為 IAM 身分驗證時，此欄位會變成選用欄位，並在主控台中標記為如此。 您可以從下拉式清單中選擇一或多個秘密，或使用建立新的秘密連結建立新的秘密。

設定	說明
用戶端身分驗證類型	代理從用戶端進行連線所使用的身分驗證類型。您的選擇適用於您將其與此代理建立關聯的所有 Secrets Manager 機密。如果您需要為每個秘密指定不同的用戶端身分驗證類型，請改用 AWS CLI 或 API 來建立您的代理。只有當您的用戶端連線使用資料庫登入資料進行身分驗證時，才指定此選項。
IAM 身分驗證	為您的代理連線指定必要或不允許的 IAM 身分驗證。您的選擇適用於您將其與此代理建立關聯的所有 Secrets Manager 機密。如果您需要為每個秘密指定不同的 IAM 身分驗證，請改用 AWS CLI 或 API 來建立您的代理。
預設身分驗證機制	選擇代理用於用戶端連線至代理的預設身分驗證類型，以及從代理連線至基礎資料庫的連線。您有下列選項： • 無（預設）- 代理會從 Secrets Manager 秘密擷取資料庫登入資料。 • IAM 身分驗證 - 代理使用 IAM 身分驗證連線到資料庫，以啟用 end-to-endIAM 身分驗證。 當您選取 IAM 身分驗證時，會顯示資訊提醒，提醒您在目標群組組態中為資料庫啟用 IAM 資料庫身分驗證。 **Note** 此選項僅支援 MySQL、PostgreSQL 和 MariaDB 引擎系列。

設定	說明
IAM 身分驗證的資料庫帳戶	只有在預設身分驗證機制設定為 IAM 身分驗證，且 Identity and Access Management (IAM) 角色設定為建立 IAM 角色時，才會顯示此欄位。 將代理的資料庫使用者帳戶命名為搭配 IAM 身分驗證使用。此為必要欄位。依下列方式指定多個帳戶： • 輸入資料庫使用者名稱以將其新增為標籤 • 使用特定資料庫使用者名稱（例如 db_user、jane_doe) • 針對多個使用者使用萬用字元模式（例如 db_test_* ） 每個帳戶會顯示為可移除的標籤，您可以按一下 X 圖示來刪除該標籤。主控台使用這些值在 IAM 角色政策中建立適當的rds-db:connect 許可。
需要 Transport Layer Security。	對所有用戶端連線強制執行 TLS/SSL。代理針對與基礎資料庫的連線使用相同的加密設定，無論用戶端連線是加密或未加密。
目標連線網路類型	代理用來連線至目標資料庫的 IP 版本。您可以從以下選項中選擇： • IPv4 – 代理會使用 IPv4 位址連線至資料庫。 • IPv6 – 代理會使用 IPv6 位址連線至資料庫。 預設值為 IPv4。若要使用 IPv6，您的資料庫必須支援雙堆疊模式。雙堆疊模式不適用於目標連線。

設定	說明
端點網路類型	用戶端用來連線至代理的代理端點 IP 版本。您可以從以下選項中選擇： • IPv4 – 代理端點僅使用 IPv4 位址。 • IPv6 – 代理端點僅使用 IPv6 位址。 • 雙堆疊 – 代理端點支援 IPv4 和 IPv6 位址。 預設值為 IPv4。若要使用 IPv6 或雙堆疊，您的 VPC 和子網路必須設定為支援選取的網路類型。
子網路	此欄位會預先填入與 VPC 關聯的所有子網路。您可以移除代理不需要的任何子網路，但必須至少保留兩個子網路。對於 IPv6 或雙堆疊端點網路類型，請確定選取的子網路支援所選擇的網路類型。

設定	說明
VPC 安全群組	選擇現有的 VPC 安全群組，或從 AWS 管理主控台建立新的安全群組。設定傳入規則以允許應用程式存取代理和傳出規則，以允許來自資料庫目標的流量。 **Note** 安全群組必須允許從代理至目標資料庫的連線。它會用於從應用程式到代理的輸入，以及從代理到資料庫的輸出。例如，如果您對資料庫和代理使用相同的安全群組，請確定該安全群組內的資源可以互相通訊。 使用共用 VPC 時，請避免使用 VPC 的預設安全群組，或與另一個帳戶相關聯的預設安全群組。而是選取屬於您帳戶的安全群組。如果沒有，請建立一個安全群組。如需詳細資訊，請參閱使用共用 VPC。 RDS 會在多個可用區域上部署代理，以確保高可用性。若要啟用跨可用區域通訊，代理子網路的網路存取控制清單 (ACL) 必須允許引擎連接埠的輸出，以及允許所有連接埠的輸入。如需網路 ACL 的詳細資訊，請參閱使用網路 ACL 控制到子網路的流量。如果您的代理和目標的網路 ACL 相同，您必須新增 TCP 通訊協定輸入規則，並將其中的來源設定為 VPC CIDR。您還必須新增引擎連接埠專屬的 TCP 通訊協定輸出規則，並將其中的目的地設定為 VPC CIDR。
啟用增強型記錄	啟用此設定，以針對代理相容性或效能問題進行疑難排解。啟用時，RDS Proxy 會記錄詳細的效能資訊，以協助您偵錯 SQL 行為或代理連線效能和可擴展性。 僅啟用此設定進行偵錯，並確保採取適當的安全措施來保護日誌中的敏感資訊。為了將額外負荷降至最低，RDS Proxy 會在啟用後 24 小時自動停用此設定。暫時使用此設定以針對特定問題進行疑難排解。

5. 選擇Create Proxy (建立代理)。

AWS CLI

若要使用 建立代理 AWS CLI，請使用下列必要參數呼叫 create-db-proxy 命令：

- `--db-proxy-name`
- `--engine-family`
- `--role-arn`
- `--vpc-subnet-ids`

`--engine-family` 值會區分大小寫。

Example

對於 Linux、macOS 或 Unix：

```
aws rds create-db-proxy \
    --db-proxy-name proxy_name \
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } \
    --role-arn iam_role \
    --vpc-subnet-ids space_separated_list \
    [--default-auth-scheme { NONE | IAM_AUTH }] \
    [--auth ProxyAuthenticationConfig_JSON_string] \
    [--vpc-security-group-ids space_separated_list] \
    [--require-tls | --no-require-tls] \
    [--idle-client-timeout value] \
    [--debug-logging | --no-debug-logging] \
    [--endpoint-network-type { IPV4 | IPV6 | DUAL }] \
    [--target-connection-network-type { IPV4 | IPV6 }] \
    [--tags comma_separated_list]
```

在 Windows 中：

```
aws rds create-db-proxy ^
    --db-proxy-name proxy_name ^
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } ^
    --role-arn iam_role ^
    --vpc-subnet-ids space_separated_list ^
    [--default-auth-scheme { NONE | IAM_AUTH }] ^
    [--auth ProxyAuthenticationConfig_JSON_string] ^
    [--vpc-security-group-ids space_separated_list] ^
```

```
    [--require-tls | --no-require-tls] ^
    [--idle-client-timeout value] ^
    [--debug-logging | --no-debug-logging] ^
    [--endpoint-network-type { IPV4 | IPV6 | DUAL }] ^
    [--target-connection-network-type { IPV4 | IPV6 }] ^
    [--tags comma_separated_list]
```

下列範例是 --auth 選項的 JSON 值。此範例會將不同的用戶端身分驗證類型套用至每個秘密。

```
[
  {
    "Description": "proxy description 1",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:123456789123:secret/1234abcd-12ab-34cd-56ef-1234567890ab",
    "IAMAuth": "DISABLED",
    "ClientPasswordAuthType": "POSTGRES_SCRAM_SHA_256"
  },

  {
    "Description": "proxy description 2",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret/1234abcd-12ab-34cd-56ef-1234567890cd",
    "IAMAuth": "DISABLED",
    "ClientPasswordAuthType": "POSTGRES_MD5"

  },

  {
    "Description": "proxy description 3",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122221111:secret/1234abcd-12ab-34cd-56ef-1234567890ef",
    "IAMAuth": "REQUIRED"
  }

]
```

--endpoint-network-type 參數會指定用戶端用來連線至代理的代理端點 IP 版本。有效的值如下：

- IPV4 – 代理端點僅使用 IPv4 位址 (預設值)。

- IPV6 – 代理端點僅使用 IPv6 位址。
- DUAL – 代理端點同時支援 IPv4 和 IPv6 位址。

`--target-connection-network-type` 參數會指定代理用來連線至目標資料庫的 IP 版本。有效的值如下：

- IPV4 – 代理會使用 IPv4 位址連線至資料庫 (預設值)。
- IPV6 – 代理會使用 IPv6 位址連線至資料庫。

若要使用 IPv6 或雙堆疊端點網路類型，您的 VPC 和子網路必須設定為支援選取的網路類型。若要使用 IPv6 目標連線網路類型，您的資料庫必須支援雙堆疊模式。

> **Tip**
>
> 如果您還不知道要用於 `--vpc-subnet-ids` 參數的子網路 ID，請參閱 設定 RDS Proxy 的網路先決條件 以取得如何尋找它們的範例。

> **Note**
>
> 安全群組必須允許存取代理連線的目標資料庫。相同的安全群組會用於從應用程式到代理的輸入，以及從代理到資料庫的輸出。例如，假設您將同一個安全群組用於您的資料庫和代理。在這種情況下，請務必指定安全群組中的資源能與同一個安全群組中的其他資源進行通訊。
> 使用共用 VPC 時，您無法使用 VPC 的預設安全群組，或屬於另一個帳戶的預設安全群組。選擇屬於您帳戶的安全群組。如果沒有，請先建立一個。如需此限制的詳細資訊，請參閱使用共用的 VPC。

若要建立代理的正確關聯，您也需要使用 register-db-proxy-targets 命令。指定目標群組類型 `default`。RDS Proxy 會在您建立每個代理時，自動建立的目標群組。

```
aws rds register-db-proxy-targets
    --db-proxy-name value
    [--target-group-name target_group_name]
    [--db-instance-identifiers space_separated_list]  # rds db instances, or
    [--db-cluster-identifiers cluster_id]        # rds db cluster (all instances)
```

RDS API

若要建立 RDS 代理，請呼叫 Amazon RDS API 操作 CreateDBProxy。您傳遞具有 AuthConfig 資料結構的參數。

RDS Proxy 在您建立每個代理時，自動建立名為 `default` 的目標群組。您可以呼叫函數 RegisterDBProxyTargets，以將 Aurora 資料庫叢集與目標群組建立關聯。

> **Important**
>
> 當您選取預設身分驗證機制的 IAM 身分驗證時：
>
> - 您必須先在目標資料庫執行個體或叢集上啟用 IAM 資料庫身分驗證，代理才能成功連線。
> - 如果您選擇建立 IAM 角色，IAM 身分驗證的資料庫帳戶欄位為必要欄位。
> - 如果您選取現有的 IAM 角色，主控台不會自動更新具有資料庫連線許可的角色。檢查角色是否具有必要的`rds-db:connect`許可。

檢視代理

建立一或多個 RDS 代理後，您可以在 AWS 管理主控台、AWS CLI 或 RDS API 中進行檢視和管理。您可以檢閱其組態詳細資訊、監控效能，並判斷哪些代理需要修改或刪除。

若要讓資料庫應用程式透過代理路由流量，您必須在連線字串中指定代理端點。

主控台

在主控台檢視代理

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Proxies (代理)。
3. 選取代理名稱以檢視其詳細資訊。
4. 在詳細資訊頁面上，目標群組區段會顯示代理程式如何連結至特定 Aurora 資料庫叢集。您可以導覽至預設目標群組頁面，以深入了解此關聯，包括在建立代理期間定義的組態設定。這些設定包括最大連線百分比、連線借用逾時、引擎系列，以及工作階段鎖定篩選條件。

CLI

若要使用 CLI 檢視代理，請使用 describe-db-proxies 命令。根據預設，請求會傳回您的 AWS 帳戶擁有的所有代理。若要查看單一代理的詳細資料，請使用 `--db-proxy-name` 參數指定其名稱。

```
aws rds describe-db-proxies [--db-proxy-name proxy_name]
```

若要檢視與代理相關聯的其他資訊，請使用下列命令。

```
aws rds describe-db-proxy-target-groups  --db-proxy-name proxy_name

aws rds describe-db-proxy-targets --db-proxy-name proxy_name
```

使用下列命令序列查看與代理關聯的事物的詳細資訊：

1. 若要取得代理清單，請執行 describe-db-proxies。
2. 若要顯示連線參數 (例如代理可以使用的連線百分比上限)，請執行 describe-db-proxy-target-groups `--db-proxy-name`。使用代理的名稱做為參數值。
3. 若要查看與所傳回目標群組相關聯的 Aurora 資料庫叢集的詳細資訊，請執行 describe-db-proxy-targets。

RDS API

若要使用 RDS API 檢視代理，請使用 DescribeDBProxies 操作。它會傳回 DBProxy 資料類型的值。

若要查看代理的連線設定詳細資訊，請將來自此傳回值的代理識別符與 DescribeDBProxyTargetGroups 操作搭配使用。它會傳回 DBProxyTargetGroup 資料類型的值。

若要查看與代理相關聯的 RDS 執行個體或 Aurora 資料庫叢集，請使用 DescribeDBProxyTargets 操作。它會傳回 DBProxyTarget 資料類型的值。

透過 RDS Proxy 連線至資料庫

您可以透過代理連線到 Aurora 資料庫叢集，或使用 Aurora Serverless v2 的叢集，其方式通常與直接連線到資料庫相同。主要區別在於您是指定代理端點，而不是叢集端點。根據預設，所有代理連線都具有讀取/寫入能力，並使用寫入器執行個體。如果您通常使用讀取器端點進行唯讀連線，則可以為代理建立額外的唯讀端點。您可以採取相同的方式使用該端點。如需詳細資訊，請參閱代理端點概觀。

主題

- 使用資料庫登入資料連線至資料庫
- 使用 IAM 身分驗證連線至資料庫
- 連線至 PostgreSQL 的考量事項

使用資料庫登入資料連線至資料庫

使用下列步驟，使用資料庫登入資料連線至代理：

1. 尋找代理端點。在 中 AWS 管理主控台，您可以在對應代理的詳細資訊頁面上找到端點。透過 AWS CLI，您可以使用 describe-db-proxies 命令。下列範例會顯示作法。

```
# Add --output text to get output as a simple tab-separated list.
$ aws rds describe-db-proxies --query '*[*].
{DBProxyName:DBProxyName,Endpoint:Endpoint}'
[
    [
        {
            "Endpoint": "the-proxy.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy"
        },
        {
            "Endpoint": "the-proxy-other-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-other-secret"
        },
        {
            "Endpoint": "the-proxy-rds-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-rds-secret"
        },
        {
            "Endpoint": "the-proxy-t3.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-t3"
        }
    ]
]
```

2. 在用戶端應用程式的連線字串中指定端點做為主機參數。例如，指定代理端點做為 `mysql -h` 選項或 `psql -h` 選項的值。
3. 請提供和平常一樣的資料庫使用者名稱和密碼。

使用 IAM 身分驗證連線至資料庫

當您搭配 RDS Proxy 使用 IAM 身分驗證時，有兩個在用戶端和代理之間進行身分驗證的選項：

- 設定資料庫使用者以使用一般使用者名稱和密碼進行身分驗證。RDS Proxy 會從 Secrets Manager 擷取使用者名稱和密碼登入資料。從 RDS Proxy 連至底層資料庫的連線不會透過 IAM。
- 您也可以使用end-to-end IAM 身分驗證，使用 IAM 透過代理連線到資料庫，而不需要資料庫登入資料。

若要使用 IAM 身分驗證連線到 RDS Proxy，請使用相同的一般連線程序，搭配 Aurora 資料庫叢集進行 IAM 身分驗證。如需使用 IAM 的一般資訊，請參閱 Amazon Aurora 的安全性。如果您使用的是 end-to-end IAM 身分驗證，請提供 IAM 身分驗證外掛程式給您的資料庫使用者。請參閱 使用 IAM 身分驗證建立資料庫帳戶。

RDS Proxy 的 IAM 使用方式的主要差異包含下列項目：

- 使用標準 IAM 身分驗證，資料庫使用者在資料庫中具有一般登入資料。您設定包含這些使用者名稱和密碼的 Secrets Manager 私密，然後授權 RDS Proxy 擷取來自 Secrets Manager 的登入資料。IAM 身分驗證會套用到您用戶端程式與代理之間的連線。接著，代理會使用從 Secrets Manager 中擷取的使用者名稱和密碼登入資料，向資料庫進行身分驗證。
- 透過end-to-end IAM 身分驗證，您不需要為資料庫登入資料設定 Secrets Manager 秘密。IAM 身分驗證適用於用戶端與代理之間的連線，以及代理與資料庫的連線。
- 您會指定代理端點，而非執行個體、叢集或讀取器端點。如需代理端點的詳細資訊，請參閱使用 IAM 身分驗證連接至資料庫叢集。
- 使用 IAM 身分驗證連接至 Proxy 時，請務必使用 Transport Layer Security (TLS)/Secure Sockets Layer (SSL)。

您可以透過修改 IAM 政策，授與特定使用者對代理的存取權。範例如下。

```
"Resource": "arn:aws:rds-db:us-east-2:1234567890:dbuser:prx-ABCDEFGHIJKL01234/db_user"
```

> **Tip**
>
> 設定 RDS Proxy 連線的 IAM 身分驗證時，請遵循下列重要準則以避免連線問題：
>
> - 在維護相同資料庫使用者或`rds_iam`角色的一般密碼身分驗證時，請勿授予角色。

- 請記住，當用戶端使用 IAM 身分驗證連線至 RDS Proxy 時，RDS Proxy 一律會透過 Secrets Manager 使用密碼身分驗證連線至資料庫。
- 如果您經常發生連線終止和重新連線，請從使用者或角色移除任何現有的`rds_iam`授予，並僅使用密碼身分驗證。
- 確保您的密碼政策符合 SCRAM-SHA-256 安全字元要求。

混合相同資料庫使用者的 IAM 和密碼身分驗證方法可能會導致連線不穩定。

連線至 PostgreSQL 的考量事項

如果您建立新的 PostgreSQL 資料庫使用者以連線至 RDS Proxy，請確定您授予使用者對於資料庫的 CONNECT 權限。如果沒有此權限，使用者就無法建立連線。如需詳細資訊，請參閱the section called "使用 RDS Proxy 時，將資料庫使用者新增至 PostgreSQL 資料庫"。

當用戶端開始 PostgreSQL 資料庫的連線時，它會傳送啟動訊息。此訊息包含包含參數名稱和值字串的組合。如需詳細資訊，請參閱 PostgreSQL 文件的 PostgreSQL 訊息格式中的 `StartupMessage`。

透過 RDS 代理連線時，啟動訊息可包含下列目前辨識的參數：

- `user`
- `database`

啟動訊息也可包含下列其他執行時間參數：

- `application_name`
- `client_encoding`
- `DateStyle`
- `TimeZone`
- `extra_float_digits`
- `search_path`

如需 PostgreSQL 傳訊的詳細資訊，請參閱 PostgreSQL 文件中的前端/後端通訊協定。

針對 PostgreSQL，如果您使用的是 JDBC，建議採取下列做法以避免鎖定：

- 將 JDBC 連線參數 `assumeMinServerVersion` 至少設為 `9.0` 以避免鎖定。這樣可防止在執行 `SET extra_float_digits = 3` 時，JDBC 驅動程式在連線啟動期間執行額外的往返。
- 將 JDBC 連線參數 `ApplicationName` 設為 *`any/your-application-name`* 以避免鎖定。這麼做可防止在執行 `SET application_name = "PostgreSQL JDBC Driver"` 時，JDBC 驅動程式在連線啟動期間執行額外的往返。請注意，JDBC 參數為 `ApplicationName`，但 PostgreSQL `StartupMessage` 參數為 `application_name`。

如需詳細資訊，請參閱避免鎖定 RDS Proxy。如需如何使用 JDBC 連線的詳細資訊，請參閱 PostgreSQL 文件中的連線至資料庫。

管理 RDS Proxy

本節提供如何管理 RDS Proxy 操作和組態的資訊。這些程序可協助應用程式以最有效率的方式使用資料庫連線，並達到最大程度的連線重複使用。您越能利用連線重複使用，就越能節省更多 CPU 和記憶體負荷。這樣可以縮短應用程式的延遲，並讓資料庫投入更多資源來處理應用程式要求。

主題

- 修改 RDS Proxy
- 使用 RDS Proxy 時新增資料庫使用者
- 從標準 IAM 身分驗證移至 RDS Proxy end-to-end IAM 身分驗證
- RDS Proxy 連線考量事項
- 避免鎖定 RDS Proxy
- 刪除 RDS Proxy

修改 RDS Proxy

您可以在建立代理之後變更與該代理相關聯的特定設定。您可以修改代理本身、它的關聯目標群組，或兩者都修改。每個代理都有一個關聯的目標群組。

AWS 管理主控台

> **Important**
>
> Client authentication type (用戶端身分驗證類型) 和 IAM authentication (IAM 身分驗證) 欄位中的值適用於與此代理相關聯的所有 Secrets Manager 機密。若要為每個秘密指定不同的值，請改用 AWS CLI 或 API 修改您的代理。

若要修改代理的設定

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Proxies (代理)。
3. 在代理清單中，選擇您要修改其設定的代理伺服器，或移至其詳細資訊頁面。
4. 在 Actions (動作) 中，選擇 Modify (修改)。
5. 輸入或選擇要修改的屬性。您可以修改下列選項：
   - Proxy identifier (Proxy 識別碼) – 輸入新的識別碼，以便將 Proxy 重新命名。
   - Idle client connection timeout (閒置用戶端連線逾時) – 輸入閒置用戶端連線逾時的期間。
   - IAM role (IAM 角色) – 變更從 Secrets Manager 中擷取秘密所使用的 IAM 角色。

     > **Note**
     >
     > 如果您將預設身分驗證機制設定為 IAM 身分驗證，則無法建立新的 IAM 角色。

   - Secrets Manager secrets (Secrets Manager 秘密) – 新增或移除 Secrets Manager 秘密。這些秘密對應於資料庫使用者名稱和密碼。
   - 用戶端身分驗證類型 – 變更用戶端連線至代理的身分驗證類型。
   - IAM authentication (IAM 身分驗證) – 要求或不允許連線至 Proxy 時進行 IAM 身分驗證。
   - 預設身分驗證機制 – 變更代理用於用戶端連線至代理以及從代理連線至基礎資料庫的預設身分驗證機制。
   - Require Transport Layer Security – 開啟或關閉 Transport Layer Security (TLS) 的需求。
   - VPC security group (VPC 安全群組) – 新增或移除 Proxy 要使用的 VPC 安全群組。
   - Enable enhanced logging (啟用增強型日誌) – 啟用或停用增強型日誌。
6. 選擇 Modify (修改)。

如果您找不到您要變更的列出設定，請使用下列程序更新代理的目標群組。與代理相關聯的目標群組可控制與實體資料庫連線相關的設定。每個代理都有一個名為 `default` 的關聯目標群組，這些群組將與代理一起自動建立。您無法重新命名預設目標群組。

您只能從代理詳細資訊頁面修改目標群組，而不能從 代理頁面上的清單修改。

若要修改代理目標群組的設定

1. 在 Proxies (代理) 頁面上，前往代理的詳細資訊頁面。
2. 在目標群組中選擇 `default` 連結。目前，所有的代理都有一個名為 `default` 的目標群組。
3. 在預設目標群組的詳細資訊頁面上，選擇 Modify (修改)。
4. 選擇您可修改之屬性的新設定：

   - 資料庫 – 選擇不同的 Aurora 叢集。
   - Connection pool maximum connections (連線集區最大連線數) – 調整 Proxy 可以使用的最大可用連線數的百分比。
   - Session pinning filters (工作階段鎖定篩選條件) – (選用) 選擇工作階段鎖定篩選條件。這樣做會規避用戶端連線之間多工處理資料庫連線的預設安全措施。目前，PostgreSQL 不支援該設定。唯一的選擇是 `EXCLUDE_VARIABLE_SETS`。

     啟用此設定可能會導致某個連線的工作階段變數影響其他連線。如果您的查詢取決於目前交易之外設定的工作階段變數值，這樣做可能會造成錯誤或正確性問題。請先確認應用程式可安全地在用戶端連線之間共用資料庫連線，再考慮使用此選項。

     以下模式可視為安全：

     - SET 陳述式，其中有效的工作階段變數值沒有變更，也就是工作階段變數沒有變更。
     - 您會在同一交易中變更工作階段變數值和執行陳述式。

     如需詳細資訊，請參閱避免鎖定 RDS Proxy。
   - Connection borrow timeout (連線借用逾時) – 調整連線借用逾時間隔。在已為代理使用連線數上限時，則會套用此設定。此設定會決定代理在傳回逾時錯誤之前等待連線變成可用的時間。
   - 初始化查詢。(選用) 新增初始化查詢，或修改目前的查詢。開啟每個新資料庫連線時，您可為要執行的代理指定一或多個 SQL 陳述式。該設定通常與 SET 陳述式搭配使用，確保每個連線都有相同的設定。請確定您新增的查詢有效。若要在單一 SET 陳述式中包含多個變數，請使用逗號分隔符號。例如：

     ```
     SET variable1=value1, variable2=value2
     ```

對於多個陳述式，請使用分號作為分隔符號。

您無法變更某些屬性，例如目標群組識別碼和資料庫引擎。

5. 選擇 Modify target group (修改目標群組)。

AWS CLI

若要使用 修改代理 AWS CLI，請使用 modify-db-proxy、 modify-db-proxy-target-group、deregister-db-proxy-targets 和 register-db-proxy-targets 命令。

使用 `modify-db-proxy` 命令，您可以變更屬性，如下所示：

- 代理使用的 Secrets Manager 秘密組。
- 是否需要 TLS。
- 閒置用戶端逾時。
- 是否記錄來自 SQL 陳述式的額外資訊以供偵錯。
- 用於擷取 Secrets Manager 秘密的 IAM 角色。
- 代理使用的安全群組。
- 與代理相關聯的預設身分驗證機制。

下列範例顯示如何將現有的代理重新命名。

```
aws rds modify-db-proxy --db-proxy-name the-proxy --new-db-proxy-name the_new_name
```

若要修改連線相關設定或重新命名目標群組，請使用 `modify-db-proxy-target-group` 命令。目前，所有的代理都有一個名為 `default` 的目標群組。使用此目標群組時，您指定代理的名稱，並將目標群組的名稱指定為 `default`。您無法重新命名預設目標群組。

下列範例顯示如何先檢查代理的 `MaxIdleConnectionsPercent` 設定，然後使用目標群組將其變更。

```
aws rds describe-db-proxy-target-groups --db-proxy-name the-proxy

{
    "TargetGroups": [
        {
```

```
            "Status": "available",
            "UpdatedDate": "2019-11-30T16:49:30.342Z",
            "ConnectionPoolConfig": {
                "MaxIdleConnectionsPercent": 50,
                "ConnectionBorrowTimeout": 120,
                "MaxConnectionsPercent": 100,
                "SessionPinningFilters": []
            },
            "TargetGroupName": "default",
            "CreatedDate": "2019-11-30T16:49:27.940Z",
            "DBProxyName": "the-proxy",
            "IsDefault": true
        }
    ]
}

aws rds modify-db-proxy-target-group --db-proxy-name the-proxy --target-group-name
 default --connection-pool-config '
{ "MaxIdleConnectionsPercent": 75 }'

{
    "DBProxyTargetGroup": {
        "Status": "available",
        "UpdatedDate": "2019-12-02T04:09:50.420Z",
        "ConnectionPoolConfig": {
            "MaxIdleConnectionsPercent": 75,
            "ConnectionBorrowTimeout": 120,
            "MaxConnectionsPercent": 100,
            "SessionPinningFilters": []
        },
        "TargetGroupName": "default",
        "CreatedDate": "2019-11-30T16:49:27.940Z",
        "DBProxyName": "the-proxy",
        "IsDefault": true
    }
}
```

使用 `deregister-db-proxy-targets` 和 `register-db-proxy-targets` 命令，您可以透過其目標群組變更 Proxy 與哪一個 Aurora 資料庫叢集相關聯。目前，每個代理都可以連線到一個 Aurora 資料庫叢集。目標群組會追蹤Aurora 叢集中所有資料庫執行個體的連線詳細資訊。

下列範例將從與名為 `cluster-56-2020-02-25-1399` 的 Aurora MySQL 叢集關聯的代理開始。此範例顯示如何變更代理，讓它可以連線到名為 `provisioned-cluster` 的不同叢集。

使用 Aurora 資料庫叢集時，您會指定 `--db-cluster-identifier` 選項。

下列範例會修改 Aurora MySQL 代理。Aurora PostgreSQL 代理具有連接埠 5432。

```
aws rds describe-db-proxy-targets --db-proxy-name the-proxy

{
    "Targets": [
        {
            "Endpoint": "instance-9814.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-9814"
        },
        {
            "Endpoint": "instance-8898.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-8898"
        },
        {
            "Endpoint": "instance-1018.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-1018"
        },
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "cluster-56-2020-02-25-1399"
        },
        {
            "Endpoint": "instance-4330.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-4330"
        }
    ]
}

aws rds deregister-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 cluster-56-2020-02-25-1399

aws rds describe-db-proxy-targets --db-proxy-name the-proxy
```

```
{
    "Targets": []
}

aws rds register-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 provisioned-cluster

{
    "DBProxyTargets": [
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "provisioned-cluster"
        },
        {
            "Endpoint": "gkldje.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "gkldje"
        },
        {
            "Endpoint": "provisioned-1.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "provisioned-1"
        }
    ]
}
```

RDS API

若要使用 RDS API 修改 Proxy，請使用 ModifyDBProxy、ModifyDBProxyTargetGroup、DeregisterDBProxyTargets 和 RegisterDBProxyTargets 操作。

使用 ModifyDBProxy，您可以變更屬性，如下所示：

- 代理使用的 Secrets Manager 秘密組。
- 是否需要 TLS。
- 閒置用戶端逾時。
- 是否記錄來自 SQL 陳述式的額外資訊以供偵錯。

- 用於擷取 Secrets Manager 秘密的 IAM 角色。
- 代理使用的安全群組。

使用 `ModifyDBProxyTargetGroup`，您可以修改連線相關設定。目前，所有的代理都有一個名為 `default` 的目標群組。使用此目標群組時，您指定代理的名稱，並將目標群組的名稱指定為 `default`。您無法重新命名預設目標群組。

使用 `DeregisterDBProxyTargets` 和 `RegisterDBProxyTargets`，您可以透過其目標群組變更 Proxy 與哪一個 Aurora 叢集相關聯。目前，每個 Proxy 都可以連線到一個 Aurora 叢集。目標群組會追蹤Aurora 叢集中資料庫執行個體的連線詳細資訊。

使用 RDS Proxy 時新增資料庫使用者

在某些情況下，您可能會將新的資料庫使用者新增至與 Proxy 相關聯的 Aurora 叢集。根據您是否使用標準身分驗證搭配 Secrets Manager 秘密或end-to-endIAM 身分驗證，繼續進行。

如果您使用標準 IAM 身分驗證，請遵循下列指示：

1. 使用 設定 RDS Proxy 的資料庫登入資料 中描述的程序建立新的 Secrets Manager 秘密。
2. 更新 IAM 角色，以授予 RDS Proxy 存取新 Secrets Manager 私密的權限。若要這麼做，請更新 IAM 角色政策的資源區段。
3. 修改 RDS Proxy，在 Secrets Manager 密碼下新增新的 Secrets Manager 密碼。
4. 如果新使用者取代了現有使用者，請為現有使用者更新儲存於代理的 Secrets Manager 秘密中的登入資料。

如果您使用的是end-to-end IAM 身分驗證，則需要建立資料庫使用者並設定 IAM 許可。若要這樣做，請進行下列步驟：

1. 在資料庫中建立新的資料庫使用者，以符合您要用於身分驗證的 IAM 使用者或角色名稱。
2. 確定資料庫使用者已在資料庫中設定 IAM 身分驗證外掛程式。請參閱 使用 IAM 身分驗證建立資料庫帳戶。
3. 更新 IAM 政策以將`rds-db:connect`許可授予 IAM 使用者或角色，如中所述為end-to-end IAM 身分驗證建立 IAM 政策。
4. 確保您的代理設定為使用 IAM 身分驗證作為預設身分驗證機制。

透過end-to-end IAM 身分驗證，您不需要在 Secrets Manager 秘密中管理資料庫憑證，因為 IAM 憑證用於從用戶端到代理以及從代理到資料庫的身分驗證。

使用 RDS Proxy 時，將資料庫使用者新增至 PostgreSQL 資料庫

將新使用者新增至 PostgreSQL 資料庫時，如果您已執行下列命令：

```
REVOKE CONNECT ON DATABASE postgres FROM PUBLIC;
```

授予 `rdsproxyadmin` 使用者 `CONNECT` 權限，讓使用者可以監控目標資料庫上的連線。

```
GRANT CONNECT ON DATABASE postgres TO rdsproxyadmin;
```

您也可以透過在上述命令中將 `rdsproxyadmin` 變更為資料庫使用者，來允許其他目標資料庫使用者執行運作狀態檢查。

使用 RDS Proxy 時變更資料庫使用者的密碼

在某些情況下，您可能會在與 Proxy 相關聯的 Aurora 叢集中變更資料庫使用者的密碼。如果是這樣，請使用新密碼更新對應的 Secrets Manager 秘密。

如果您使用end-to-end IAM 身分驗證，則不需要更新 Secrets Manager 秘密中的任何密碼。

從標準 IAM 身分驗證移至 RDS Proxy end-to-end IAM 身分驗證

如果您目前使用 RDS Proxy 的標準 IAM 身分驗證，其中用戶端使用 IAM 向代理進行身分驗證，但代理使用秘密連接到資料庫，則可以遷移到用戶端end-to-end代理和proxy-to-database連線都使用 IAM 身分驗證的端對端 IAM 身分驗證。 client-to-proxy

移至end-to-end IAM 身分驗證

1. 更新 RDS Proxy IAM 角色許可

   建立包含 Secrets Manager 和 許可的已更新代理`rds:db-connect`許可政策：

   ```
   # Create updated proxy permission policy
   cat > updated-proxy-policy.json # EOF
   ```

   ```
   {
   ```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GetSecretsValue",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:123456789012:secret:secretName-1234f"
      ]
    },
    {
      "Sid": "RdsDBConnect",
      "Action": [
        "rds-db:connect"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:rds-db:us-east-1:123456789012:dbuser:cluster-ABCDEFGHIJKL01234/
jane_doe"
      ]
    }
  ]
}
```

更新代理您的角色政策：

```
aws iam put-role-policy \
            --role-name RDSProxyRole \
            --policy-name UpdatedProxyPermissions \
            --policy-document file://updated-proxy-policy.json
```

2. 修改 RDS Proxy 以啟用end-to-end IAM 身分驗證

```
aws rds modify-db-proxy \
  --db-proxy-name my-database-proxy \
  --default-auth-scheme IAM_AUTH \
  --region us-east-1
```

確認 RDS Proxy 狀態為可用，並且在繼續以確保遷移期間零停機時間`DefaultAuthScheme`之前`IAM_AUTH`。

```
aws rds describe-db-proxies --db-proxy-name my-database-proxy --region us-east-1
```

預期的輸出結果：

```
{
  "DBProxies": [
    {
      "DBProxyName": "my-database-proxy",
      "DBProxyArn": "arn:aws:rds:us-east-1:123456789012:db-
proxy:prx-0123456789abcdef",
      "Status": "available",
      ...
      "DefaultAuthScheme": "IAM_AUTH"
    }
  ]
}
```

3. 在資料庫上啟用 IAM 身分驗證

```
aws rds modify-db-cluster \
  --db-cluster-identifier my-database-cluster \
  --enable-iam-database-authentication \
  --region us-east-1
```

4. 設定資料庫使用者進行 IAM 身分驗證

對於 Aurora PostgreSQL：

```
GRANT rds_iam TO jane_doe;
```

對於 Aurora MySQL：

```
ALTER USER 'jane_doe' IDENTIFIED WITH AWSAuthenticationPlugin AS 'RDS';
ALTER USER 'jane_doe'@'%' REQUIRE SSL;
```

5. 您的用戶端應用程式程式碼不需要變更。連線程序保持不變：

對於 Aurora PostgreSQL：

```
# Generate authentication token
```

```
export PGPASSWORD=$(aws rds generate-db-auth-token \
  --hostname my-database-proxy.proxy-ABCDEFGHIJKL01234.us-east-1.rds.amazonaws.com
 \
  --port 5432 \
  --username jane_doe \
  --region us-east-1)

# Connect to database through proxy
psql "host=my-database-proxy.proxy-ABCDEFGHIJKL01234.us-east-1.rds.amazonaws.com
 port=5432 user=jane_doe dbname=postgres password=$PGPASSWORD sslmode=require
 sslrootcert=us-east-1-bundle.pem"
```

對於 Aurora MySQL：

```
# Generate authentication token
export MYSQL_PWD=$(aws rds generate-db-auth-token \
  --hostname my-database-proxy.proxy-ABCDEFGHIJKL01234.us-east-1.rds.amazonaws.com
 \
  --port 3306 \
  --username jane_doe \
  --region us-east-1)

# Connect to database through proxy
mysql -h my-database-proxy.proxy-ABCDEFGHIJKL01234.us-east-1.rds.amazonaws.com \
  -P 3306 \
  -u jane_doe \
  --ssl-ca=us-east-1-bundle.pem \
  --enable-cleartext-plugin
```

RDS Proxy 連線考量事項

配置連線設定

若要調整 RDS Proxy 的連線集區，可以修改以下設定：

- IdleClientTimeout
- MaxConnectionsPercent
- MaxIdleConnectionsPercent
- ConnectionBorrowTimeout

IdleClientTimeout

您可以指定 Proxy 關閉用戶端連線之前允許的閒置時間長度。預設值為 1,800 秒 (30 分鐘)。

上一個要求完成後若應用程式未在的指定時間內提交新要求，用戶端連線就會被視為閒置。基礎資料庫連線保持開啟狀態，並傳回至連線集區。因此，它可供新的用戶端連線使用。如果您希望 Proxy 主動移除過時的連線，則降低閒置用戶端連線逾時。如果您的工作負載會經常與 Proxy 建立連線，則提高閒置用戶端連線逾時，以節省建立連線的成本。

此設定以 RDS 主控台中的 Idle client connection timeout (閒置用戶端連線逾時) 欄位，以及 `IdleClientTimeout` 和 API 中的 AWS CLI 設定表示。若要了解如何在 RDS 主控台中變更 Idle client connection timeout (閒置用戶端連線逾時) 欄位的值，請參閱 AWS 管理主控台。若要了解如何變更 `IdleClientTimeout` 設定的值，請參閱 CLI 命令 modify-db-proxy 或 API 操作 ModifyDBProxy。

MaxConnectionsPercent

您可以限制 RDS Proxy 可以與目標資料庫建立的連線數。需以資料庫的可用最大連線數百分比形式指定限制。此設定以 RDS 主控台中的 Connection pool maximum connections (連線集區最大連線數) 欄位，以及 AWS CLI 和 API 中的 `MaxConnectionsPercent` 設定表示。

`MaxConnectionsPercent` 值會以目標群組所使用的 Aurora 資料庫叢集 的 `max_connections` 設定百分比來表示。Proxy 不會事先保留這些連線。此設定可讓 Proxy 在工作負載需要時建立這些連線。

例如，對於 `max_connections` 設定為 1000 且 `MaxConnectionsPercent` 設為 95 的已註冊資料庫目標，RDS Proxy 會將 950 個連線設定為同時連線至該資料庫目標的上限。

工作負載達到允許的資料庫連線數目上限的常見副作用，是會增加整體查詢延遲，以及提高 `DatabaseConnectionsBorrowLatency` 指標。您可以透過比較 `DatabaseConnections` 和 `MaxDatabaseConnectionsAllowed` 指標，來監控目前使用的資料庫連線和允許的總數。

設定此參數時，請注意下列最佳實務：

- 為工作負載模式的變更提供足夠的連線預留空間。建議將參數設定至少比您最近監控的最大使用量高出 30%。由於 RDS Proxy 會在多個節點之間重新分配資料庫連線配額，因此內部容量變更可能需要至少 30% 的成長空間才能進行其他連線，以避免增加借用延遲。
- RDS Proxy 會保留特定數量的連線以進行主動監控，以支援快速容錯移轉、流量路由和內部操作。`MaxDatabaseConnectionsAllowed` 指標不包括這些保留的連線。它代表可用於服務工作負載的連線數，並且可以低於從 `MaxConnectionsPercent` 設定衍生的值。

最低建議 MaxConnectionsPercent 值如下：

- db.t3.small：100
- db.t3.medium：55
- db.t3.large：35
- db.r3.large 或以上：20

如果多個目標執行個體向 RDS Proxy 註冊，例如具有讀取器節點的 Aurora 叢集，請根據最小註冊的執行個體設定最小值。

若要了解如何在 RDS 主控台中變更 Connection pool maximum connections (連線集區最大連線數) 欄位的值，請參閱 AWS 管理主控台。若要了解如何變更 MaxConnectionsPercent 設定的值，請參閱 CLI 命令 modify-db-proxy-target-group 或 API 操作 ModifyDBProxyTargetGroup。

> **Important**
>
> 如果資料庫叢集屬於已開啟寫入轉送的全域資料庫，請依配置給寫入轉送的配額減少代理的 MaxConnectionsPercent 值。寫入轉送配額是在資料庫叢集參數 aurora_fwd_writer_max_connections_pct 中設定的。如需寫入轉送的資訊，請參閱在 Amazon Aurora 全域資料庫中使用寫入轉送。

如需資料庫連線數限制的詳細資訊，請參閱對 Aurora MySQL 資料庫執行個體的連線數上限和對 Aurora PostgreSQL 資料庫執行個體的連線數上限。

MaxIdleConnectionsPercent

您可以控制 RDS Proxy 可在連線集區中保留的閒置資料庫連線數。根據預設，集區中的資料庫連線五分鐘內沒有任何活動時，RDS Proxy 就會將連線視為閒置。

MaxIdleConnectionsPercent 值會以 RDS 資料庫執行個體目標群組的 max_connections 設定百分比來表示。預設值為 MaxConnectionsPercent 的 50%，上限為 MaxConnectionsPercent 的值。例如，如果 MaxConnectionsPercent 為 80，則 MaxIdleConnectionsPercent 的預設值為 40。

值大時，Proxy 會將高百分比的閒置資料庫連線維持在開啟狀態。值小時，Proxy 就會關閉高百分比的閒置資料庫連線。如果您的工作負載無法預測，請考慮為 MaxIdleConnectionsPercent 設定一個高值。這麼做表示 RDS Proxy 可以因應活動中的突增情況，而無需開啟大量新的資料庫連線。

此設定以 AWS CLI 和 API 中 DBProxyTargetGroup 的 MaxIdleConnectionsPercent 設定表示。若要了解如何變更 MaxIdleConnectionsPercent 設定的值，請參閱 CLI 命令 modify-db-proxy-target-group 或 API 操作 ModifyDBProxyTargetGroup。

如需資料庫連線數限制的詳細資訊，請參閱對 Aurora MySQL 資料庫執行個體的連線數上限和對 Aurora PostgreSQL 資料庫執行個體的連線數上限。

ConnectionBorrowTimeout

您可以選擇 RDS Proxy 在傳回逾時錯誤之前，等待連線集區中的資料庫連線變成可用的時間。預設值為 120 秒。此設定適用於連線數達到最大值，因此連線集區中沒有可用的連線時。這也適用於沒有適當的資料庫執行個體可用於處理請求的情況，例如容錯移轉操作進行中時。透過此設定，您可為應用程式設定最佳等待期間，而不必變更應用程式的程式碼中的查詢逾時。

此設定以 RDS 主控台中的 Connection borrow timeout (連線借用逾時) 欄位，或者 AWS CLI 或 API 中 DBProxyTargetGroup 的 ConnectionBorrowTimeout 設定表示。若要了解如何在 RDS 主控台中變更 Connection borrow timeout (連線借用逾時) 欄位的值，請參閱 AWS 管理主控台。若要了解如何變更 ConnectionBorrowTimeout 設定的值，請參閱 CLI 命令 modify-db-proxy-target-group 或 API 操作 ModifyDBProxyTargetGroup。

用戶端與資料庫連線

從應用程式到 RDS Proxy 的連線稱為用戶端連線。從 Proxy 到資料庫的連線為資料庫連線。使用 RDS Proxy 時，用戶端連線會在 Proxy 終止，而資料庫連線則是在 RDS Proxy 內管理。

應用程式端連線集區可提供減少應用程式與 RDS Proxy 之間建立重複連線的好處。

在實作應用程式端連線集區之前，請考慮下列組態層面：

- 用戶端連線生命週期上限：RDS Proxy 會將用戶端連線的生命週期上限強制為 24 小時。此值不可設定。設定連線生命週期上限少於 24 小時的集區，以避免意外的用戶端連線中斷。
- 用戶端連線閒置逾時：RDS Proxy 會強制執行用戶端連線的閒置時間上限。設定閒置連線逾時值低於 RDS Proxy 用戶端連線閒置逾時設定的集區，以避免意外連線中斷。

在應用程式端連線集區中設定的用戶端連線數目上限不必限制為 RDS Proxy 的 max_connections 設定。

用戶端連線集區會延長用戶端連線生命週期。如果您的連線遭遇綁定，則集區用戶端連線可能會降低多工效率。在應用程式端連線集區中綁定但閒置的用戶端連線會繼續保留資料庫連線，並防止其他用戶端連線重複使用資料庫連線。檢閱您的 Proxy 日誌，以檢查您的連線是否遭遇綁定。

> **Note**
>
> RDS Proxy 會在不再使用資料庫連線 24 小時後的某個時間，關閉資料庫連線。無論閒置連線數上限設定的值為何，Proxy 都會執行此動作。

避免鎖定 RDS Proxy

當資料庫請求不依賴先前請求的狀態資訊時，多工會更有效率。在這種情況下，RDS Proxy 可以在每一筆交易結束時重複使用連線。這類狀態資訊的範例包括您可以透過 SET 或 SELECT 陳述式變更的大部分變數和組態參數。根據預設，用戶端連線上的 SQL 交易可以在基礎資料庫連線之間進行多工。

您與代理的連線可能會進入一種稱為鎖定的狀態。連線被鎖定時，每一筆後續交易都會使用相同的基礎資料庫連線，直到工作階段結束為止。其他用戶端連線也無法重複使用該資料庫連線，直到工作階段結束為止。工作階段會在用戶端連線中斷時結束。

在偵測到不適用於其他工作階段的工作階段狀態變更時，RDS Proxy 會自動將用戶端連線鎖定至特定資料庫連線。鎖定會降低連線重複使用的有效性。如果所有或幾乎所有的連線都遭遇鎖定，請考慮修改應用程式的程式碼或工作負載，以減少導致鎖定的情況。

例如，您的應用程式變更了工作階段變數或組態參數。在此情況下，稍後的陳述式可能會依賴新的變數或參數才會生效。因此，當 RDS Proxy 處理請求，而變更工作階段變數或組態設定時，會將該工作階段到資料庫的連線鎖定。這樣一來，工作階段狀態對於同一工作階段中的所有後續交易仍然有效。

對於部分資料庫引擎，此規則不適用於您可以設定的所有參數。RDS Proxy 會追蹤某些陳述式和變數。因此，當您修改這些項目時，RDS Proxy 不會鎖定工作階段。在這種情況下，RDS Proxy 只會重複使用其他工作階段的連線，這些工作階段具有與這些設定相同的值。如需 Aurora MySQL 的追蹤陳述式和變數清單，請參閱 RDS Proxy 針對 Aurora MySQL 資料庫追蹤哪些項目。

RDS Proxy 針對 Aurora MySQL 資料庫追蹤哪些項目

RDS Proxy 會追蹤下列 MySQL 陳述式：

- DROP DATABASE
- DROP SCHEMA
- USE

RDS Proxy 會追蹤下列 MySQL 變數：

- AUTOCOMMIT
- AUTO_INCREMENT_INCREMENT
- CHARACTER SET (or CHAR SET)
- CHARACTER_SET_CLIENT
- CHARACTER_SET_DATABASE
- CHARACTER_SET_FILESYSTEM
- CHARACTER_SET_CONNECTION
- CHARACTER_SET_RESULTS
- CHARACTER_SET_SERVER
- COLLATION_CONNECTION
- COLLATION_DATABASE
- COLLATION_SERVER
- INTERACTIVE_TIMEOUT
- NAMES
- NET_WRITE_TIMEOUT
- QUERY_CACHE_TYPE
- SESSION_TRACK_SCHEMA
- SQL_MODE
- TIME_ZONE
- TRANSACTION_ISOLATION (or TX_ISOLATION)
- TRANSACTION_READ_ONLY (or TX_READ_ONLY)
- WAIT_TIMEOUT

> **Note**
>
> 當您在工作階段範圍設定變數時，RDS Proxy 會追蹤對 TRANSACTION_ISOLATION 和 TRANSACTION_READ_ONLY 變數的變更。不過，如果您在下一個交易範圍設定它們，RDS Proxy 會鎖定連線。無論您使用 SET 陳述式或 SET TRANSACTION 陳述式來設定這些值，此行為都適用。

盡可能減少鎖定

RDS Proxy 的效能調校涉及試圖藉由將鎖定降到最低，讓交易層級的連線重複使用 (多工) 達到最大。

您可以執行下列操作，盡可能減少鎖定：

- 避免可能導致鎖定的不必要資料庫要求。
- 在所有連線中一致地設定變數和組態設定。這樣，稍後的工作階段更有可能重複使用具有這些特定設置的連線。

  但是，對於 PostgreSQL，設定變數會導致工作階段鎖定。
- 若為 MySQL 引擎系列資料庫，將工作階段鎖定篩選條件套用至代理。如果您知道鎖定並不會影響應用程式的正確操作，就可以免除某些類型的操作將工作階段鎖定。
- 監控 Amazon CloudWatch 指標 `DatabaseConnectionsCurrentlySessionPinned`，以查看鎖定發生的頻率。如需有關此 CloudWatch 指標和其他 CloudWatch 指標的資訊，請參閱 使用 Amazon CloudWatch 監控 RDS Proxy 指標。
- 如果您使用 `SET` 陳述式為每個用戶端連線執行相同的初始化，您可以執行此操作，同時保留交易層級的多工。在此情況下，您可以將設定初始工作階段狀態的陳述式移至代理所使用的初始化查詢中。此屬性為一包含一個或多個以分號分隔的 SQL 陳述式的字串。

  例如，您可以為設定特定組態參數的代理定義初始化查詢。然後，每當它為該代理設定新連線時，RDS Proxy 會套用這些設定。您可以從應用程式的程式碼中移除相應的 `SET` 陳述式，這樣它們就不會干擾交易層級的多工。

  如需代理發生鎖定頻率的指標，請參閱 使用 Amazon CloudWatch 監控 RDS Proxy 指標。

導致所有引擎系列鎖定的條件

在下列情況下，多工可能會導致意外行為，代理將工作階段鎖定於目前連線：

- 文字大小大於 16 KB 的任何陳述式會導致代理鎖定工作階段。

導致 Aurora MySQL 鎖定的條件

若為 PostgreSQL，下列互動也會導致鎖定：

- 明確的資料表鎖定陳述式 `LOCK TABLE`、`LOCK TABLES` 或 `FLUSH TABLES WITH READ LOCK` 會導致代理鎖定工作階段。

- 使用 GET_LOCK 建立具名鎖定會導致代理鎖定工作階段。
- 設定使用者或系統變數 (有某些例外) 會將工作階段鎖定至代理。如果這會大幅限制連線重複使用，您可以設定 SET 操作以避免鎖定。若要這樣做，請調整工作階段鎖定篩選屬性。如需詳細資訊，請參閱為 Amazon Aurora 建立代理及修改 RDS Proxy。
- 建立一個暫時資料表會導致代理鎖定工作階段。這樣，無論交易邊界如何，暫時資料表的內容在整個工作階段中都會保留。
- 呼叫 ROW_COUNT 和 FOUND_ROWS 函數有時會導致鎖定。

  這些函數導致鎖定的確切情況，在與 MySQL 5.7 相容的 Aurora MySQL 版本之間可能會有所差異。
- 預備陳述式導致代理鎖定工作階段。無論預備陳述式使用 SQL 文字或二進位協定，此規則都適用。
- 當您使用 SET LOCAL 時，RDS Proxy 不會釘選連線。
- 呼叫存放程序和存放函數不會導致鎖定。RDS Proxy 不會偵測由此類呼叫導致的任何工作階段狀態變更。如果您依賴該工作階段狀態在交易之間持續存在，請確保您的應用程式不會在預存常式內變更工作階段狀態。例如，RDS Proxy 目前與會建立可在所有交易之間持續存在的暫時資料表之預存程序不相容。
- 具有 MySQL （語法 /* ！ ... */) 或 MariaDB （語法 /*M ！ ... */) 可執行註解的查詢會導致鎖定。RDS Proxy 無法剖析這些註解中內嵌的 SQL，以追蹤工作階段狀態變更。

如果您對應用程式行為具有專業知識，您可以略過特定應用程式陳述式的鎖定行為。若要這麼做，請在建立代理時選擇工作階段鎖定篩選條件選項。您目前可以選擇退出工作階段鎖定，以便設定工作階段變數和組態設定。

導致 Aurora PostgreSQL 鎖定的條件

若是 PostgreSQL，下列互動也會導致鎖定：

- 使用 SET 命令。
- 使用 PREPARE、DISCARD、DEALLOCATE 或 EXECUTE 命令來管理預備陳述式。
- 建立暫時序列、資料表或檢視。
- 宣告游標。
- 放棄工作階段狀態。
- 在通知通道上接聽。
- 載入程式庫模組，例如 auto_explain。
- 使用 nextval 和 setval 等函數操作序列。

- 使用 pg_advisory_lock 和 pg_try_advisory_lock 等函數與鎖定互動。

> **Note**
>
> RDS Proxy 不會鎖定交易層級諮詢鎖定，特別是 pg_advisory_xact_lock、pg_advisory_xact_lock_shared、pg_try_advisory_xact_l 和 pg_try_advisory_xact_lock_shared。

- 設定參數或將參數重設為預設值。具體而言，使用 SET 和 set_config 命令將預設值指派給工作階段變數。
- 呼叫存放程序和存放函數不會導致鎖定。RDS Proxy 不會偵測由此類呼叫導致的任何工作階段狀態變更。如果您依賴該工作階段狀態在交易之間持續存在，請確保您的應用程式不會在預存常式內變更工作階段狀態。例如，RDS Proxy 目前與會建立可在所有交易之間持續存在的暫時資料表之預存程序不相容。
- 捨棄工作階段狀態。如果您使用連線集區程式庫，並將 DISCARD ALL 查詢設定為重設查詢，RDS Proxy 會在發行時鎖定您的用戶端連線。這會降低代理的多工效率，並可能導致非預期的結果，因為 DISCARD ALL 命令可能會干擾工作階段狀態管理。

刪除 RDS Proxy

當您不再需要代理時，可以將它刪除。如果您讓與某一代理關聯的資料庫執行個體或叢集停止服務，也可能會刪除該代理。

AWS 管理主控台

若要刪除代理

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Proxies (代理)。
3. 從清單中選擇要刪除的代理。
4. 選擇 Delete Proxy (刪除代理)。

AWS CLI

若要刪除資料庫代理，請使用 AWS CLI 命令 delete-db-proxy。若要移除相關的關聯，請同時使用 deregister-db-proxy-targets 命令。

```
aws rds delete-db-proxy --name proxy_name
```

```
aws rds deregister-db-proxy-targets
    --db-proxy-name proxy_name
    [--target-group-name target_group_name]
    [--target-ids comma_separated_list]       # or
    [--db-instance-identifiers instance_id]       # or
    [--db-cluster-identifiers cluster_id]
```

RDS API

若要刪除資料庫代理，請呼叫 Amazon RDS API 函數 DeleteDBProxy。若要刪除相關項目和關聯，您也可以呼叫函數 DeleteDBProxyTargetGroup 和 DeregisterDBProxyTargets。

使用 Amazon RDS Proxy 端點

RDS Proxy 端點提供彈性且有效率的方式來管理資料庫連線，進而改善可擴展性、可用性和安全性。使用代理端點，您可以：

- 簡化監控和疑難排解 – 使用多個端點來獨立追蹤和管理不同應用程式的連線。
- 增強讀取可擴展性 – 利用具有 Aurora 資料庫叢集的讀取器端點來有效率地分發讀取流量，最佳化大量查詢工作負載的效能。
- 啟用跨 VPC 存取 – 使用跨 VPC 端點跨 VPC 連線資料庫，這允許一個 VPC 中的 Amazon EC2 執行個體等資源存取另一個 VPC 中的資料庫。

主題

- 代理端點概觀
- 代理端點的限制
- 將讀取器端點與 Aurora 叢集搭配使用
- 跨 VPC 存取 Aurora 資料庫
- 建立代理端點
- 檢視代理端點
- 修改代理端點
- 刪除代理端點

代理端點概觀

使用 RDS Proxy 端點涉及與 Aurora 資料庫叢集和讀取器端點相同的程序類型。如果您不熟悉 Aurora 端點，請在 Amazon Aurora 端點連線 中查閱詳細資訊。

當您搭配 Aurora 叢集使用 RDS Proxy 時，預設端點同時支援讀取和寫入操作。這表示其會將所有請求路由至叢集的寫入器執行個體，造成其 `max_connections` 限制。為了更好地分發流量，您可以為代理建立額外的讀取/寫入或唯讀端點，實現更有效率的工作負載管理和改善的可擴展性。

將唯讀端點與代理搭配使用來處理讀取查詢，就像針對 Aurora 佈建叢集使用讀取器端點一樣。此方法透過在一或多個讀取器資料庫執行個體之間分發查詢，將 Aurora 的讀取可擴展性最大化。透過使用唯讀端點並視需要新增更多讀取器執行個體，您可以增加叢集可以處理的同時查詢和連線數量。

> **Tip**
>
> 當您使用 為 Aurora 叢集建立代理時 AWS 管理主控台，您可以讓 RDS Proxy 自動建立讀取器端點。如需讀取器端點優點的相關資訊，請參閱將讀取器端點與 Aurora 叢集搭配使用。

當您建立代理端點時，您可以將其與和代理的 VPC 不同的虛擬私有雲端 (VPC) 建立關聯。這可讓您從另一個 VPC 連線至代理，例如組織內不同應用程式所使用的 VPC。

如需與代理端點關聯的限制相關資訊，請參閱代理端點的限制。

RDS Proxy 日誌中會在每個項目加上相關聯代理端點的名稱作為字首。此名稱可以是您針對使用者定義端點所指定的名稱，也可以是代理的預設讀取/寫入端點的特殊名稱 `default`。

每個代理端點都有自己的 CloudWatch 指標集。監控所有代理端點、特定端點或代理的所有讀取/寫入或唯讀端點的指標。如需詳細資訊，請參閱使用 Amazon CloudWatch 監控 RDS Proxy 指標。

代理端點會使用與相關聯代理相同的身分驗證機制。RDS Proxy 會自動設定使用者定義端點的許可和授權，且與相關聯代理的屬性一致。

若要了解代理端點如何針對 Aurora 全域資料庫中的資料庫叢集運作，請參閱 RDS Proxy 端點如何使用全域資料庫。

代理端點的限制

RDS Proxy 端點有下列限制：

- 無法修改 RDS Proxy 預設端點。
- 代理的使用者定義端點數目上限為 20。因此，代理最多可以有 21 個端點：預設端點，以及您建立的 20 個端點。
- 當您將其他端點與代理建立關聯時，RDS Proxy 會自動確定叢集中要用於每個端點的資料庫執行個體。您無法按照對 Aurora 自訂端點所用的方式來選擇特定的執行個體。
- 對於 IPv6 或雙堆疊端點網路類型，您的 VPC 和子網路必須設定為支援選取的網路類型。

當您建立代理時，RDS 會自動建立 VPC 端點，以在應用程式和資料庫之間進行安全通訊。VPC 端點是可見的，可以從 Amazon VPC 主控台存取。

新增代理端點會佈建 AWS PrivateLink 界面端點。如果您將一或多個端點新增至代理，會產生額外費用。如需詳細資訊，請參閱 RDS Proxy 定價。

將讀取器端點與 Aurora 叢集搭配使用

當您將 RDS Proxy 與 Aurora 叢集搭配使用時，您可以建立並連線至名稱為讀取器端點的唯讀端點。這些讀取器端點有助於改善查詢密集型應用程式的讀取可擴展性。如果叢集中的讀取器資料庫執行個體變得無法使用，讀取器端點也有助於改善連線的可用性。

> **Note**
>
> 當您指定新端點為唯讀時，RDS Proxy 會要求 Aurora 叢集具有一個或多個讀取器資料庫執行個體。在某些情況下，您可能會將代理的目標變更為只包含單一寫入器的 Aurora 叢集。如果這樣做，讀取器端點的任何請求都會因錯誤而失敗。如果代理的目標是 RDS 執行個體而非 Aurora 叢集，請求也會失敗。
>
> 如果 Aurora 叢集具有讀取器執行個體，但這些執行個體無法使用，RDS Proxy會等待傳送請求，而不是立即傳回錯誤。如果連線借用逾時期間內沒有讀取器執行個體變得可用，請求則會失敗並顯示錯誤。

讀取器端點如何協助應用程式變得可用

在某些情況下，叢集中的一個或多個讀取器執行個體可能無法使用。如果是這樣，使用資料庫代理讀取器端點的連線可以比使用 Aurora 讀取者端點更快復原。RDS Proxy 只會將連線路由傳送至叢集中可用的讀取器執行個體。當執行個體變得無法使用時，DNS 快取不會有延遲。

如果連線為多路復用，RDS Proxy 會將後續查詢導向至不同的讀取器資料庫執行個體，而不會中斷您的應用程式。在自動切換至新的讀取器執行個體期間，RDS Proxy 會同時檢查新舊讀取器執行個體的

複寫延遲。RDS Proxy 會確保新讀取器執行個體為最新狀態，且具有與上一個讀取器執行個體相同的變更。這樣，您的應用程式絕不會在 RDS Proxy 從一個讀取器資料庫執行個體切換至另一個時看到過時資料。

如果連線鎖定，則連線的下一個查詢會傳回錯誤。不過，您的應用程式可以立即重新連線至相同端點。RDS Proxy 會將連線路由至不同但為 `available` 狀態的讀取器資料庫執行個體。手動重新連線時，RDS Proxy 不會檢查新舊讀取器執行個體之間的複寫延遲。

如果您的 Aurora 叢集沒有任何可用的讀取器執行個體，RDS Proxy 會檢查此條件是暫時的還是永久的。每種情況下的行為如下：

- 假設您的叢集具有一個或多個讀取器資料庫執行個體，則均未處於 `Available` 狀態。例如，所有讀取器執行個體可能正在重新啟動或發生問題。在這種情況下，連線至讀取器端點的嘗試會等待讀取器執行個體變成可用。如果連線借用逾時期間內沒有讀取器執行個體變得可用，則連線嘗試會失敗。如果讀取器執行個體變得可用，則連線嘗試會成功。
- 假設您的叢集沒有讀取器資料庫執行個體。在這種情況下，如果您嘗試連線至讀取器端點，則 RDS Proxy 會立即傳回錯誤。若要解決此問題，請在連線至讀取器端點之前，將一個或多個讀取器執行個體新增至叢集。

讀取器端點如何協助實現查詢可擴展性

代理的讀取器端點透過以下方式協助實現 Aurora 查詢可擴展性：

- 在您將讀取器執行個體新增至 Aurora 叢集時，RDS Proxy 可以將任何讀取器端點的新連線路由至不同的讀取器執行個體。如此一來，使用一個讀取器端點連線執行的查詢，不會減慢使用另一個讀取器端點連線執行的查詢速度。查詢在單獨的資料庫執行個體上執行。每個資料庫執行個體都有其自己的運算資源、緩衝區快取等。
- 在實際情況下，RDS Proxy 針對使用特定讀取器端點連線的所有查詢問題，使用相同的讀取器資料庫執行個體。如此一來，相同資料表上的一組相關查詢可以利用特定資料庫執行個體上的快取、計劃最佳化等。
- 如果讀取器資料庫執行個體變得無法使用，則對應用程式的影響取決於工作階段是多路復用還是鎖定。如果工作階段是多路復用，RDS Proxy 會將任何後續查詢路由至不同的讀取器資料庫執行個體，而無需執行任何動作。如果工作階段已鎖定，您的應用程式會收到錯誤，並且必須重新連線。您可以立即重新連線至讀取器端點，並且 RDS Proxy 會將連線路由至可用的讀取器資料庫執行個體。如需代理工作階段的多工處理和鎖定的詳細資訊，請參閱RDS Proxy 概念概觀。
- 您在叢集中具有的讀取器資料庫執行個體越多，您可以使用讀取器端點建立的同時連線就越多。例如，假設您的叢集有四個讀取器資料庫執行個體，每個執行個體都設定為支援 200 個同時連線。

也假設您的 Proxy 設定為使用連線數目上限的 50%。在這裡，對於讀取器 1，您可以透過代理中的讀取器端點進行的最大連線數目是 100 (200 的 50%)。對於讀取器 2 也是 100，依此類推，總計為 400。如果您將叢集讀取器資料庫執行個體的數目加倍至八個，則透過讀取器端點的連線數目上限也會加倍至 800。

使用讀取器端點的範例

下列 Linux 範例顯示了，如何確認您已透過讀取器端點連線至 Aurora MySQL 叢集。`innodb_read_only` 組態設定已啟用。嘗試執行寫入操作，例如 `CREATE DATABASE` 陳述式失敗並顯示錯誤。您也可以透過使用 `aurora_server_id` 變數，檢查資料庫執行個體名稱，來確認您已連線至讀取器資料庫執行個體。

> **Tip**
>
> 不要僅依賴檢查資料庫執行個體名稱，來確定連線是讀取/寫入還是唯讀。請記住，Aurora 叢集中的資料庫執行個體，可以在發生容錯移轉時變更寫入器和讀取器之間的角色。

```
$ mysql -h endpoint-demo-reader.endpoint.proxy-demo.us-east-1.rds.amazonaws.com -u
 admin -p
...
mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
mysql> create database shouldnt_work;
ERROR 1290 (HY000): The MySQL server is running with the --read-only option so it
 cannot execute this statement

mysql> select @@aurora_server_id;
+---------------------------------------+
| @@aurora_server_id                    |
+---------------------------------------+
| proxy-reader-endpoint-demo-instance-3 |
+---------------------------------------+
```

下列範例顯示了即使刪除讀取器資料庫執行個體，您與代理讀取器端點的連線仍然可以繼續運作。在此範例中，Aurora 叢集具有兩個讀取器執行個體，`instance-5507` 和 `instance-7448`。讀取器端點

的連線會開始使用其中一個讀取器執行個體。在此範例中，系統會透過 delete-db-instance 命令來刪除此讀取器執行個體。如需後續查詢，RDS Proxy 會切換至不同的讀取器執行個體。

```
$ mysql -h reader-demo.endpoint.proxy-demo.us-east-1.rds.amazonaws.com
  -u my_user -p
...
mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-5507      |
+--------------------+

mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+

mysql> select count(*) from information_schema.tables;
+----------+
| count(*) |
+----------+
|      328 |
+----------+
```

雖然 mysql 工作階段仍在執行中，下列命令仍會刪除讀取器端點連線的讀取器執行個體。

```
aws rds delete-db-instance --db-instance-identifier instance-5507 --skip-final-snapshot
```

mysql 工作階段中的查詢會繼續進行工作，無需重新連線。RDS Proxy 會自動切換至不同的讀取器資料庫執行個體。

```
mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-7448      |
+--------------------+

mysql> select count(*) from information_schema.TABLES;
```

```
+----------+
| count(*) |
+----------+
|      328 |
+----------+
```

跨 VPC 存取 Aurora 資料庫

依預設，RDS 技術堆疊的元件全都在同一個 Amazon VPC 中。例如，假設在 Amazon EC2 執行個體上執行的應用程式連線至 Amazon 資料庫叢集。在此情況下，應用程式伺服器和資料庫必須都位於同一個 VPC 內。

使用 RDS Proxy，您可以從另一個 VPC 的資源 (例如 EC2 執行個體)，設定對一個 VPC 中 Aurora 資料庫叢集的存取。例如，您的組織可能有多個存取相同資料庫資源的應用程式。每個應用程式都可能在其自己的 VPC 中。

若要啟用跨 VPC 存取，您可以為代理建立新的端點。Proxy 本身駐留在與 Aurora 資料庫叢集相同的 VPC 中。然而，跨 VPC 端點駐留在其他 VPC 中，以及 EC2 執行個體等其他資源。跨 VPC 端點與 EC2 和其他資源相同的 VPC 子網路和安全群組關聯。這些關聯可讓您從因 VPC 限制而無法存取資料庫的應用程式連線至端點。

下列步驟說明如何透過 RDS Proxy 來建立和存取跨 VPC 端點：

1. 建立兩個 VPC，或選擇兩個已用於 Aurora 和 工作的 VPC。每個 VPC 都應有其自己的關聯網路資源，例如，網際網路閘道、路由表、子網路和安全群組。如果您只有一個 VPC，可以參閱Amazon Aurora 入門，了解設定另一個 VPC 以成功使用 Aurora 的步驟。您還可以在 Amazon EC2 主控台中查看您現有的 VPC，了解要連線在一起的資源類型。
2. 建立與您要連線的 Aurora 資料庫叢集相關聯的資料庫 Proxy。請遵循 為 Amazon Aurora 建立代理 中的程序。
3. 於 RDS 主控台代理的詳細資訊頁面上，在 Proxy endpoints (代理端點) 區段中，選擇 Create endpoint (建立端點)。請遵循 建立代理端點 中的程序。
4. 選擇要讓跨 VPC 端點讀取/寫入還是唯讀。
5. 不是接受與 Aurora 資料庫叢集相同 VPC 的預設值，而是選擇不同的 VPC。此 VPC 必須與代理所在的 VPC 位於相同的 AWS 區域。
6. 現在，不是接受與 Aurora 資料庫叢集相同 VPC 的子網路和安全群組預設值，而是選擇新的選項。根據您選擇的 VPC 中的子網路和安全群組進行這些設定。

7. 您不需要變更 Secrets Manager 機密的任何設定。相同的登入資料適用於代理的所有端點，無論每個端點所在的 VPC 為何。同樣地，使用 IAM 身分驗證時，即使端點位於不同的 VPCs 中，您的 IAM 組態和許可也會在所有代理端點中一致運作。每個端點不需要額外的 IAM 組態。
8. 等待新端點為可用狀態。
9. 記下完整的端點名稱。這是以 *Region_name*.rds.amazonaws.com 結尾的值，作為資料庫應用程式連線字串的一部分提供。
10. 從與端點相同的 VPC 中的資源存取新端點。測試此程序的簡單方法是，在此 VPC 中建立一個新的 EC2 執行個體。然後，使用連線字串中的端點值，登入 EC2 執行個體並執行 mysql 或 psql 命令來進行連線。

建立代理端點

若要建立代理端點，請按照以下指示：

主控台

若要建立代理伺服器端點

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Proxies (代理)。
3. 按一下您要為其建立新端點的代理名稱。

   該代理的詳細資訊頁面即會出現。
4. 在 Proxy endpoints (代理端點) 區段中，選擇 Create proxy endpoint (建立代理端點)。

   Create proxy endpoint (建立代理端點) 視窗即會出現。
5. 對於 Proxy endpoint name (代理端點名稱)，輸入您選擇的描述性名稱。
6. 對於 Target role (目標角色) 下，選擇要讓端點讀取/寫入還是唯讀。

   使用讀取/寫入端點的連線可以執行任何類型的操作，例如資料定義語言 (DDL) 陳述式、資料處理語言 (DML) 陳述式和查詢。這些端點一律連線至 Aurora 叢集的主要執行個體。當您在應用程式中僅使用單一端點時，您可以使用讀取/寫入端點來進行一般資料庫操作。您還可以將讀取/寫入端點用於管理操作、線上交易處理 (OLTP) 應用程式，以及解壓縮轉換-載入 (ETL) 任務。

   使用唯讀端點的連線只能執行查詢。當 Aurora 叢集中有多個讀取器執行個體時，RDS Proxy 可以針對端點的每個連線使用不同的讀取器執行個體。這樣，查詢密集型應用程式可以利用 Aurora 的

叢集功能。您可以透過新增更多讀取器資料庫執行個體，為叢集新增更多查詢容量。這些唯讀連線不會對叢集的主要執行個體施加任何額外負荷。如此一來，您的報告和分析查詢不會減慢 OLTP 應用程式的寫入操作。

7. 對於虛擬私有雲端 (VPC)，請選擇預設值，從通常用來存取代理或其相關聯資料庫的相同 EC2 執行個體或其他資源來存取端點。若要設定此代理的跨 VPC 存取，請選擇預設值以外的 VPC。如需跨 VPC 存取的詳細資訊，請參閱跨 VPC 存取 Aurora 資料庫。
8. 對於端點網路類型，選擇代理端點的 IP 版本。可用選項為：

   - IPv4 – 代理端點僅使用 IPv4 位址 (預設值)。
   - IPv6 – 代理端點僅使用 IPv6 位址。
   - 雙堆疊 – 代理端點支援 IPv4 和 IPv6 位址。

   若要使用 IPv6 或雙堆疊，您的 VPC 和子網路必須設定為支援選取的網路類型。
9. 對於子網路，RDS Proxy 預設會填入與關聯代理相同的子網路。若要限制對端點的存取，以便只有部分 VPC 地址範圍可以連線至其中，請移除一個或多個子網路。
10. 若為 VPC Security groups (VPC 安全群組)，您可以選擇現有的安全群組，或建立新的安全群組。依據預設，RDS Proxy 會填入與相關聯代理相同的安全群組或群組。如果代理的傳入和傳出規則適用於此端點，則保留預設選項。

    如果您選擇建立新的安全群組，請在此頁面上指定安全群組的名稱。然後稍後從 EC2 主控台編輯安全群組設定。
11. 選擇 Create proxy endpoint (建立代理端點)。

AWS CLI

若要建立代理端點，請使用 AWS CLI create-db-proxy-endpoint 命令。

包含下列必要參數：

- `--db-proxy-name` *value*
- `--db-proxy-endpoint-name` *value*
- `--vpc-subnet-ids` *list_of_ids*. 以空格分隔子網路 ID。您沒有指定 VPC 本身的 ID。

您還可以包含下列選用參數：

- `--target-role { READ_WRITE | READ_ONLY }`此參數預設為 READ_WRITE。READ_ONLY 值只會影響包含一個或多個讀取器資料庫執行個體的 Aurora 佈建叢集。當代理與僅包含寫入器資料庫執行個體的Aurora 叢集相關聯時，您無法指定 READ_ONLY。如需有關預期使用唯讀端點搭配 Aurora 叢集的詳細資訊，請參閱 將讀取器端點與 Aurora 叢集搭配使用 。
- `--vpc-security-group-ids` *value*. 以空格分隔安全群組 ID。如果省略此參數，RDS Proxy 會使用預設的 VPC 安全群組。RDS Proxy 會根據您為 `--vpc-subnet-ids` 參數指定的子網路 ID 來決定 VPC。
- `--endpoint-network-type { IPV4 | IPV6 | DUAL }`。此參數指定代理端點的 IP 版本。預設值為 IPV4。若要使用 IPV6 或 DUAL，您的 VPC 和子網路必須設定為支援選取的網路類型。

Example

下列範例會建立名稱為 my-endpoint 的代理端點

對於 Linux、macOS 或 Unix：

```
aws rds create-db-proxy-endpoint \
  --db-proxy-name my-proxy \
  --db-proxy-endpoint-name my-endpoint \
  --vpc-subnet-ids subnet_id subnet_id subnet_id ... \
  --target-role READ_ONLY \
  --vpc-security-group-ids security_group_id \
  --endpoint-network-type DUAL
```

在 Windows 中：

```
aws rds create-db-proxy-endpoint ^
  --db-proxy-name my-proxy ^
  --db-proxy-endpoint-name my-endpoint ^
  --vpc-subnet-ids subnet_id_1 subnet_id_2 subnet_id_3 ... ^
  --target-role READ_ONLY ^
  --vpc-security-group-ids security_group_id ^
  --endpoint-network-type DUAL
```

RDS API

若要建立代理端點，請使用 RDS API CreateDBProxyEndpoint 動作。

檢視代理端點

若要檢視現有的代理端點，請遵循下列指示：

主控台

若要檢視代理端點的詳細資訊

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Proxies (代理)。
3. 在清單中，選擇您要檢視其端點的代理。按一下代理名稱，以檢視其詳細資訊頁面。
4. 在 Proxy endpoints (代理端點) 區段中，選擇您要檢視的端點。按一下其名稱，以檢視詳細資訊頁面。
5. 檢查您感興趣的參數值。您可以檢查如下所示屬性：
   - 端點是讀取/寫入還是唯讀。
   - 您在資料庫連線字串中使用的端點地址。
   - 與端點關聯的 VPC、子網路和安全群組。

AWS CLI

若要檢視一或多個代理端點，請使用 AWS CLI describe-db-proxy-endpoints 命令。

您可以包含下列任一選用參數：

- `--db-proxy-endpoint-name`
- `--db-proxy-name`

以下範例描述 `my-endpoint` 代理端點。

Example

對於 Linux、macOS 或 Unix：

```
aws rds describe-db-proxy-endpoints \
  --db-proxy-endpoint-name my-endpoint
```

在 Windows 中：

```
aws rds describe-db-proxy-endpoints ^
  --db-proxy-endpoint-name my-endpoint
```

RDS API

若要描述一個或多個代理端點，請使用 RDS API DescribeDBProxyEndpoints 操作。

修改代理端點

若要修改代理端點，請遵循下列指示：

主控台

修改一個或多個代理端點

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Proxies (代理)。
3. 在清單中，選擇您要修改其端點的代理。按一下代理名稱，以檢視其詳細資訊頁面。
4. 在 Proxy endpoints (代理端點) 區段中，選擇您要修改的端點。您可以在清單中選取或按一下其名稱，以檢視詳細資訊頁面。
5. 在代理詳細資訊頁面的 Proxy endpoints (代理端點) 區段中，選擇 Edit (編輯)。或者在代理端點詳細資訊頁面上，針對動作，選擇編輯。
6. 變更您要修改的參數值。
7. 選擇儲存變更。

AWS CLI

若要修改代理端點，請使用 AWS CLI modify-db-proxy-endpoint 命令，其中包含下列必要參數：

- `--db-proxy-endpoint-name`

使用下列一個或多個參數來指定對端點屬性的變更：

- `--new-db-proxy-endpoint-name`

- `--vpc-security-group-ids`. 以空格分隔安全群組 ID。

下列範例會將 `my-endpoint` 代理端點重新命名為 `new-endpoint-name`。

Example

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint \
  --new-db-proxy-endpoint-name new-endpoint-name
```

在 Windows 中：

```
aws rds modify-db-proxy-endpoint ^
  --db-proxy-endpoint-name my-endpoint ^
  --new-db-proxy-endpoint-name new-endpoint-name
```

RDS API

若要修改代理端點，請使用 RDS API ModifyDBProxyEndpoint 操作。

刪除代理端點

若要刪除代理的端點，請遵循下列指示：

> **Note**
>
> 您無法刪除 RDS Proxy 自動為每個代理建立的預設代理端點。
> 當您刪除代理時，RDS Proxy 會自動刪除所有關聯的端點。

主控台

若要使用 AWS 管理主控台 來刪除代理端點

1. 在導覽窗格中，選擇 Proxies (代理)。
2. 在清單中，選擇您要刪除其端點的代理。按一下代理名稱，以檢視其詳細資訊頁面。
3. 在 Proxy endpoints (代理端點) 區段中，選擇您要刪除的端點。您可以在清單中選取一個或多個端點，或者按一下單一端點的名稱，以檢視詳細資訊頁面。

4. 在代理詳細資訊頁面的 Proxy endpoints (代理端點) 區段中，選擇 Delete (刪除)。或者在代理端點詳細資訊頁面上，針對動作，選擇刪除。

AWS CLI

若要刪除代理端點，請執行 delete-db-proxy-endpoint 命令，其中包含下列必要參數：

- `--db-proxy-endpoint-name`

下列命令會刪除名稱為 `my-endpoint` 的代理端點。

對於 Linux、macOS 或 Unix：

```
aws rds delete-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint
```

在 Windows 中：

```
aws rds delete-db-proxy-endpoint ^
  --db-proxy-endpoint-name my-endpoint
```

RDS API

若要使用 RDS API 刪除代理端點，請執行 DeleteDBProxyEndpoint 操作。指定代理端點 `DBProxyEndpointName` 參數的名稱。

使用 Amazon CloudWatch 監控 RDS Proxy 指標

您可以使用 Amazon CloudWatch 來監控 RDS Proxy。CloudWatch 會收集並處理來自代理的原始資料，進而將這些資料轉換為便於讀取且幾近即時的指標。若要在 CloudWatch 主控台中尋找這些指標，請選擇Metrics (指標)，然後選擇 RDS，並選擇 Per-Proxy Metrics (每代理指標)。如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》的使用 Amazon CloudWatch 指標。

> **Note**
>
> RDS 會針對與代理關聯的每個基礎 Amazon EC2 執行個體發佈這些指標。單一代理可能由一個以上的 EC2 執行個體服務。使用 CloudWatch 統計資料彙總所有關聯執行個體之間的代理值。

其中一些指標可能會在 Proxy 第一次成功連線之後才顯示。

在 RDS Proxy 日誌中，每個項目均會以關聯的代理端點名稱作為前綴。此名稱可以是您針對使用者定義端點所指定的名稱，也可以是執行讀取/寫入請求的代理預設端點的特殊名稱 `default`。

所有 RDS Proxy 指標都在群組 `proxy` 中。

每個代理端點都有其自己的 CloudWatch 指標。您可以獨立監控每個代理端點的用量。如需代理端點的詳細資訊，請參閱使用 Amazon RDS Proxy 端點.

您可以使用下列其中一個維度集來彙總每個指標的值。例如，藉由使用 `ProxyName` 維度集，您可以分析特定代理的所有流量。透過使用其他維度集，您可以用不同的方式來分割指標。您可以根據每個代理的不同端點或目標資料庫，或者每個資料庫的讀取/寫入和唯讀流量來分割指標。

- 維度集 1：`ProxyName`
- 維度集 2：`ProxyName`、`EndpointName`
- 維度集 3：`ProxyName`、`TargetGroup`、`Target`
- 維度集 4：`ProxyName`、`TargetGroup`、`TargetRole`

指標	描述	有效期間	CloudWatch 維度集
`AvailabilityPercentage`	目標群組在維度所指出的角色中可供使用的時間百分比。此指標每分鐘報告一次。此指標最實用的統計資料是 Sum。	1 分鐘	Dimension set 4
`ClientConnections`	目前的用戶端連線數。此指標每分鐘報告一次。此指標最實用的統計資料是 Sum。	1 分鐘	Dimension set 1, Dimension set 2
`ClientConnectionsClosed`	已關閉的用戶端連線數。此指標最實用的統計資料是 Sum。	1 分鐘或以上	Dimension set 1, Dimension set 2

指標	描述	有效期間	CloudWatch 維度集
ClientConnectionsInSetup	目前開啟的用戶端連線數量，但尚未完成設定。此指標每分鐘報告一次。此指標最有用的統計資料是「總和」。	1 分鐘	Dimension set 1, Dimension set 2
ClientConnectionsNoTLS	不具有 Transport Layer Security (TLS) 的目前用戶端連線數。此指標每分鐘報告一次。此指標最實用的統計資料是 Sum。	1 分鐘	Dimension set 1, Dimension set 2
ClientConnectionsReceived	接收到的用戶端連線要求數。此指標最實用的統計資料是 Sum。	1 分鐘或以上	Dimension set 1, Dimension set 2
ClientConnectionsSetupFailedAuth	因身分驗證或 TLS 設定錯誤而失敗的用戶端連線嘗試次數。此指標最實用的統計資料是 Sum。	1 分鐘或以上	Dimension set 1, Dimension set 2
ClientConnectionsSetupSucceeded	使用任何具有或不具 TLS 的身分驗證機制成功建立的用戶端連線數。此指標最實用的統計資料是 Sum。	1 分鐘或以上	Dimension set 1, Dimension set 2
ClientConnectionsTLS	具有 TLS 的目前用戶端連線數。此指標每分鐘報告一次。此指標最實用的統計資料是 Sum。	1 分鐘	Dimension set 1, Dimension set 2

指標	描述	有效期間	CloudWatch 維度集
DatabaseConnectionRequests	建立資料庫連線的要求數目。此指標最實用的統計資料是 Sum。	1 分鐘或以上	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionRequestsWithTLS	要透過 TLS 建立資料庫連線的要求數目。此指標最實用的統計資料是 Sum。	1 分鐘或以上	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnections	目前的資料庫連線數。此指標每分鐘報告一次。此指標最實用的統計資料是 Sum。	1 分鐘	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsBorrowLatency	代理為了取得資料庫連線而受到監控所花費的時間 (以毫秒為單位)。此指標最實用的統計資料是 Sum。	1 分鐘或以上	Dimension set 1, Dimension set 2
DatabaseConnectionsCurrentlyBorrowed	目前借用狀態中的資料庫連線數。此指標每分鐘報告一次。此指標最實用的統計資料是 Sum。	1 分鐘	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsCurrentlyInTransaction	交易中的目前資料庫連線數。此指標每分鐘報告一次。此指標最實用的統計資料是 Sum。	1 分鐘	Dimension set 1, Dimension set 3, Dimension set 4

指標	描述	有效期間	CloudWatch 維度集
DatabaseConnectionsCurrentlySessionPinned	由於變更工作階段狀態之用戶端要求中的操作而遭到鎖定的目前資料庫連線數。此指標每分鐘報告一次。此指標最實用的統計資料是 Sum。	1 分鐘	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsSetupFailed	失敗的資料庫連線要求數目。此指標最實用的統計資料是 Sum。	1 分鐘或以上	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsSetupSucceeded	使用或不使用 TLS 成功建立的資料庫連線數。此指標最實用的統計資料是 Sum。	1 分鐘或以上	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsWithTLS	具有 TLS 的目前資料庫連線數。此指標每分鐘報告一次。此指標最實用的統計資料是 Sum。	1 分鐘	Dimension set 1, Dimension set 3, Dimension set 4
MaxDatabaseConnectionsAllowed	允許的資料庫連線數上限。此指標每分鐘報告一次。此指標最實用的統計資料是 Sum。	1 分鐘	Dimension set 1, Dimension set 3, Dimension set 4
QueryDatabaseResponseLatency	資料庫回應查詢已花費的時間 (以毫秒為單位)。此指標最實用的統計資料是 Average。	1 分鐘或以上	Dimension set 1, Dimension set 2, Dimension set 3, Dimension set 4

指標	描述	有效期間	CloudWatch 維度集
QueryRequests	收到的查詢數。包含多個陳述式的查詢會計算為一個查詢。此指標最實用的統計資料是 Sum。	1 分鐘或以上	Dimension set 1, Dimension set 2
QueryRequestsNoTLS	從非 TLS 連線收到的查詢數。包含多個陳述式的查詢會計算為一個查詢。此指標最實用的統計資料是 Sum。	1 分鐘或以上	Dimension set 1, Dimension set 2
QueryRequestsTLS	從 TLS 連線收到的查詢數。包含多個陳述式的查詢會計算為一個查詢。此指標最實用的統計資料是 Sum。	1 分鐘或以上	Dimension set 1, Dimension set 2
QueryResponseLatency	取得查詢請求與代理回應請求之間的時間 (以毫秒為單位)。此指標最實用的統計資料是 Average。	1 分鐘或以上	Dimension set 1, Dimension set 2

您可以在 AWS 管理主控台 中的 CloudWatch 下方找到 RDS Proxy 活動的記錄。每個代理在日誌群組頁面中都有一筆記錄。

> **Important**
>
> 這些日誌檔是供人類使用，用於故障診斷，而非程式設計存取。日誌檔的格式和內容可能會變更。

尤其是，較舊的日誌不包含指示每個請求端點的任何前綴。在較新的日誌中，每個項目均會以關聯的代理端點名稱作為前綴。此名稱可以是您針對使用者定義端點所指定的名稱，也可以是請求的 `default` 特殊名稱 (使用代理的預設端點)。

使用 RDS Proxy 事件

事件指示某個環境的變更，例如 AWS 環境或軟體即服務 (SaaS) 合作夥伴的服務或應用程式。也可以是您自己的自訂應用程式或服務。例如，建立或修改 RDS Proxy 時，Amazon Aurora 會產生一個事件。Amazon Aurora 會近乎即時地將事件傳遞至 Amazon EventBridge。以下列出可訂閱的 Amazon RDS Proxy 事件清單以及 RDS Proxy 事件範例。

如需使用事件的詳細資訊，請參閱下列內容：

- 如需使用 AWS 管理主控台、AWS CLI 或 RDS API 來檢視事件的說明，請參閱 檢視 Amazon RDS 事件。
- 若要了解如何設定 Amazon Aurora 將事件傳送到 EventBridge，請參閱 建立由 Amazon Aurora 事件觸發的規則。

RDS Proxy 事件

下表顯示 RDS Proxy 為來源類型時的事件類別和事件清單。

類別	RDS 事件 ID	訊息	備註
組態變更	RDS-EVENT-0204	RDS 已修改資料庫代理 *name*。	無
組態變更	RDS-EVENT-0207	RDS 已修改資料庫代理 *name* 的端點。	無
組態變更	RDS-EVENT-0213	RDS 偵測到新增了資料庫執行個體，並自動將其新增到資料庫代理 *name* 的目標群組。	無
組態變更	RDS-EVENT-0214	RDS 偵測到刪除了資料庫執行個體 *name*，並自動將	無

類別	RDS 事件 ID	訊息	備註
		其從資料庫代理 *name* 的目標群組 *name* 中移除。	
組態變更	RDS-EVENT-0215	RDS 偵測到刪除了資料庫叢集 *name*，並自動將其從資料庫代理 *name* 的目標群組 *name* 中移除。	無
建立	RDS-EVENT-0203	RDS 已建立資料庫代理 *name*。	無
建立	RDS-EVENT-0206	RDS 已建立資料庫代理 *name* 的端點 *name*。	無
刪除	RDS-EVENT-0205	RDS 已刪除資料庫代理 *name*。	無
刪除	RDS-EVENT-0208	RDS 已刪除資料庫代理 *name* 的端點 *name*。	無
失敗	RDS-EVENT-0243	RDS 無法佈建代理 *name* 的容量，因為您的子網路中沒有足夠的可用 IP 地址：*name*。若要修正此問題，請確定您的子網路具有最低數量的未使用 IP 地址，如 RDS Proxy 文件所建議。	若要確定執行個體類別的建議數量，請參閱 規劃 IP 地址容量。
失敗	RDS-EVENT-0275	RDS 對部分連往資料庫代理##的連線進行限流。從用戶端到代理的同時連線要求數目已超過限制。	無

以下是 JSON 格式的 RDS Proxy 事件範例。事件顯示 RDS 修改了 `my-endpoint` RDS Proxy 的 `my-rds-proxy` 端點。事件 ID 是 RDS-EVENT-0207。

```
{
  "version": "0",
  "id": "68f6e973-1a0c-d37b-f2f2-94a7f62ffd4e",
  "detail-type": "RDS DB Proxy Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-09-27T22:36:43Z",
  "region": "us-east-1",
  "resources": [
     "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy"
  ],
  "detail": {
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PROXY",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy",
    "Date": "2018-09-27T22:36:43.292Z",
    "Message": "RDS modified endpoint my-endpoint of DB Proxy my-rds-proxy.",
    "SourceIdentifier": "my-endpoint",
    "EventID": "RDS-EVENT-0207"
  }
}
```

RDS Proxy 命令列範例

若要查看連線命令和 SQL 陳述式的組合如何與 RDS Proxy 互動，請參閱下列範例。

範例

- Preserving Connections to a MySQL Database Across a Failover
- Adjusting the max_connections Setting for an Aurora DB Cluster

Example透過容錯移轉保留與 MySQL 資料庫的連線

此 MySQL 範例將示範已開啟的連線如何在容錯移轉期間繼續運作。例如，當您重新啟動資料庫，或資料庫因由於有問題而無法使用時。本範例使用一個名為 `the-proxy` 的代理，以及一個具有資料庫執行個體 `instance-8898` 和 `instance-9814` 的 Aurora 資料庫叢集。從 Linux `failover-db-cluster` 命令列執行命令時，代理連線的寫入器執行個體會變更為不同的資料庫執行個體。您可以看到與代理關聯的資料庫執行個體改變了，但連線仍然保持開啟。

```
$ mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p
Enter password:
...

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-9814      |
+--------------------+
1 row in set (0.01 sec)

mysql>
[1]+  Stopped                 mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com
 -u admin_user -p
$ # Initially, instance-9814 is the writer.
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.
$ # Now instance-8898 is the writer.
$ fg
mysql -h the-proxy.proxy-demo.us.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-8898      |
+--------------------+
1 row in set (0.01 sec)

mysql>
[1]+  Stopped                 mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com
 -u admin_user -p
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.
$ # Now instance-9814 is the writer again.
$ fg
mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+--------------------+
```

```
| @@aurora_server_id |
+--------------------+
| instance-9814      |
+--------------------+
1 row in set (0.01 sec)
+---------------+---------------+
| Variable_name | Value         |
+---------------+---------------+
| hostname      | ip-10-1-3-178 |
+---------------+---------------+
1 row in set (0.02 sec)
```

Example 調整 Aurora 資料庫叢集的 max_connections 設定

此範例示範如何調整 Aurora MySQL 資料庫叢集的 max_connections 設定。若要這麼做，請根據 MySQL 5.7 相容叢集的預設參數設定，建立自己的資料庫叢集參數群組。您可以為 max_connections 設定指定一個值，以取代設定預設值的公式。將該資料庫叢集參數群組與資料庫叢集相關聯。

```
export REGION=us-east-1
export CLUSTER_PARAM_GROUP=rds-proxy-mysql-57-max-connections-demo
export CLUSTER_NAME=rds-proxy-mysql-57

aws rds create-db-parameter-group --region $REGION \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-parameter-group-name $CLUSTER_PARAM_GROUP \
  --description "Aurora MySQL 5.7 cluster parameter group for RDS Proxy demo."

aws rds modify-db-cluster --region $REGION \
  --db-cluster-identifier $CLUSTER_NAME \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP

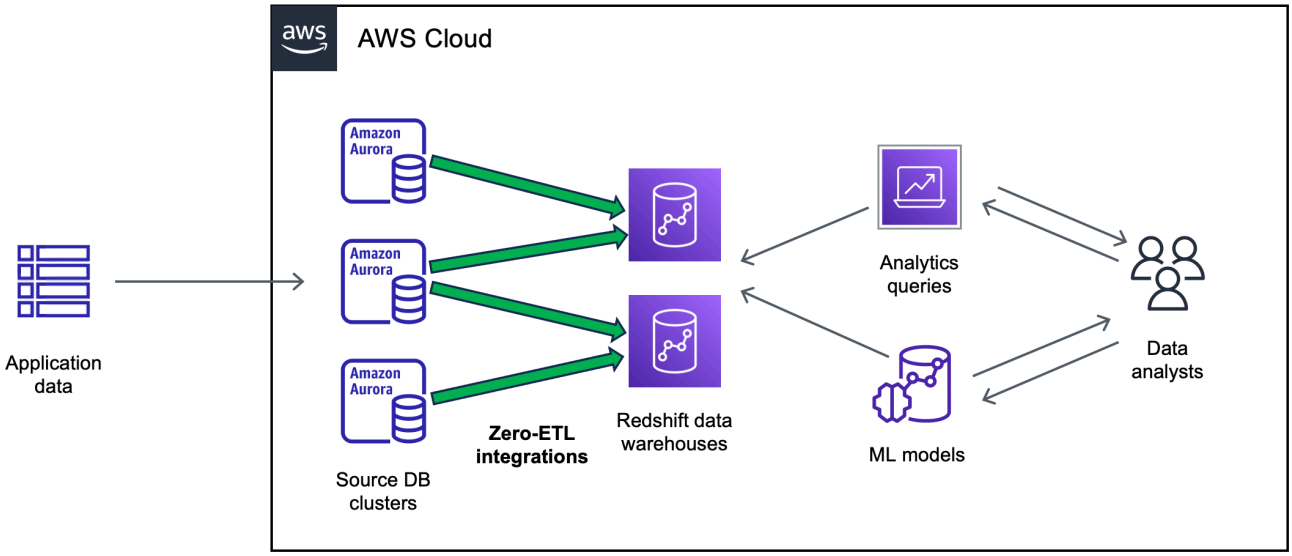
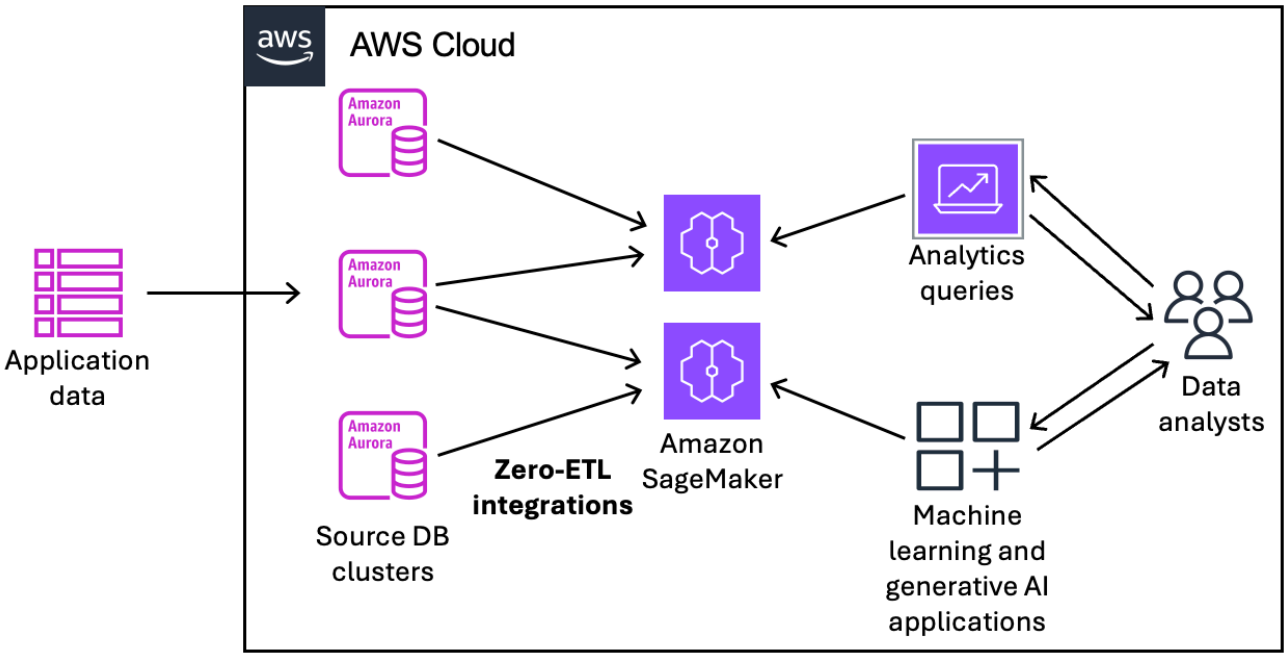
echo "New cluster param group is assigned to cluster:"
aws rds describe-db-clusters --region $REGION \
  --db-cluster-identifier $CLUSTER_NAME \
  --query '*[*].{DBClusterParameterGroup:DBClusterParameterGroup}'

echo "Current value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
  --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
  --output text | grep "^max_connections"
```

```
echo -n "Enter number for max_connections setting: "
read answer

aws rds modify-db-cluster-parameter-group --region $REGION --db-cluster-parameter-
group-name $CLUSTER_PARAM_GROUP \
  --parameters "ParameterName=max_connections,ParameterValue=$
$answer,ApplyMethod=immediate"

echo "Updated value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
  --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
  --output text | grep "^max_connections"
```

RDS Proxy 的故障診斷

您可在下列內容中找到某些常見 RDS Proxy 問題的故障診斷想法，以及 RDS Proxy 的 CloudWatch 日誌的相關資訊。

在 RDS Proxy 日誌中，每個項目均會以關聯的代理端點名稱作為前綴。此名稱可以是您針對使用者定義端點所指定的名稱。也可以是執行讀取/寫入請求之代理預設端點的特殊名稱 `default`。如需代理端點的詳細資訊，請參閱使用 Amazon RDS Proxy 端點.

主題

- 驗證代理的連線能力
- 常見問題與解決方案
- 針對 RDS for MySQL 的 RDS Proxy 問題進行疑難排解
- 針對 RDS for PostgreSQL 的 RDS Proxy 問題進行疑難排解

驗證代理的連線能力

您可以使用下列命令來驗證連線中的所有元件 (例如代理、資料庫和運算執行個體) 可以彼此通訊。

使用 describe-db-proxies 命令檢查代理本身。也請使用 describe-db-proxy-target-groups 命令檢查相關聯的目標群組。檢查目標的詳細資訊是否符合您打算與 Proxy 建立關聯的 Aurora 叢集。使用如下命令。

```
aws rds describe-db-proxies --db-proxy-name $DB_PROXY_NAME
```

```
aws rds describe-db-proxy-target-groups --db-proxy-name $DB_PROXY_NAME
```

若要確認代理可連線至基礎資料庫，請使用 describe-db-proxy-targets 命令檢查目標群組中指定的目標。使用如下命令。

```
aws rds describe-db-proxy-targets --db-proxy-name $DB_PROXY_NAME
```

describe-db-proxy-targets 命令的輸出包含 TargetHealth 欄位。您可以檢查 State 內的 Reason、Description 和 TargetHealth 欄位，以檢查代理是否能與基礎資料庫執行個體進行通訊。

- State 的 AVAILABLE 值表示代理可連線至資料庫執行個體。
- State 的 UNAVAILABLE 值表示暫時或永久的連線問題。在這種情況下，請檢查 Reason 和 Description 欄位。例如，如果 Reason 具有 PENDING_PROXY_CAPACITY 的值，請在代理完成其擴展操作後嘗試再次連線。如果 Reason 具有 UNREACHABLE、CONNECTION_FAILED 或 AUTH_FAILURE 的值，請使用來自 Description 欄位的解說，協助您診斷此問題。
- State 欄位在變更為 REGISTERING 或 AVAILABLE 前，可能短暫具有 UNAVAILABLE 的值。

如果下列 Netcat 命令 (nc) 成功，您可以從 EC2 執行個體或登入的其他系統存取代理端點。如果您與代理和關聯的資料庫不在同一 VPC 中，則此命令會回報失敗。您可能可以直接登入資料庫，而不需要在同一個 VPC 中。然而除非您位於同一個 VPC 中，否則無法登入代理。

```
nc -zx MySQL_proxy_endpoint 3306

nc -zx PostgreSQL_proxy_endpoint 5432
```

您可以使用下列命令確定您的 EC2 執行個體具有需要的屬性。特別是，EC2 執行個體的 VPC 必須與 Proxy 連線的 RDS 資料庫執行個體Aurora 叢集的 VPC 相同。

```
aws ec2 describe-instances --instance-ids your_ec2_instance_id
```

檢查用於代理的 Secrets Manager 秘密。

```
aws secretsmanager list-secrets
aws secretsmanager get-secret-value --secret-id your_secret_id
```

請確定 get-secret-value 所顯示的 SecretString 欄位已編碼為包含 username 和 password 欄位的 JSON 字串。下列範例顯示 SecretString 欄位的格式。

```
{
  "ARN": "some_arn",
  "Name": "some_name",
  "VersionId": "some_version_id",
  "SecretString": '{"username":"some_username","password":"some_password"}',
  "VersionStages": [ "some_stage" ],
  "CreatedDate": some_timestamp
}
```

疑難排解 IAM 身分驗證問題時，請確認下列事項：

- 資料庫已啟用 IAM 身分驗證。
- 代理已設定正確的身分驗證機制。
- 提供給代理的 IAM 角色中的 IAM 政策會將必要的`rds-db:connect`許可授予適當的資料庫及其使用者名稱。
- 對於end-to-end IAM 身分驗證，存在符合 IAM 使用者或角色名稱的資料庫使用者。
- 連線已啟用 SSL/TLS。

常見問題與解決方案

本節說明使用 RDS Proxy 時的一些常見問題和潛在解決方案。

執行 `aws rds describe-db-proxy-targets` CLI 命令之後，如果 `TargetHealth` 描述說明 `Proxy does not have any registered credentials`，請驗證下列事項：

- 有已註冊的憑證，供使用者存取代理。
- 用來存取代理所使用之 Secrets Manager 秘密的 IAM 角色是有效的。

建立或連線至資料庫代理時，您可能會遇到下列 RDS 事件。

類別	RDS 事件 ID	說明
失敗	RDS-EVENT-0243	RDS 無法佈建代理的容量，因為子網路中沒有足夠的可用 IP 地址。若要修正此問題，請確認您的子網路具有最低數目的未使用 IP 地址。若要確定執行

類別	RDS 事件 ID	說明
		個體類別的建議數量，請參閱規劃 IP 地址容量。
失敗	RDS-EVENT-0275	RDS 對部分連往資料庫代理##的連線進行限流。從用戶端到代理的同時連線要求數目已超過限制。

建立新代理或連線至代理時，可能會遇到下列問題。

錯誤	原因或因應措施
`403: The security token included in the request is invalid`	選取現有的 IAM 角色，而不是選擇建立新的 IAM 角色。

針對 RDS for MySQL 的 RDS Proxy 問題進行疑難排解

連線到 MySQL 代理時可能會遇到下列問題。

錯誤	原因或因應措施
`ERROR 1040 (HY000): Connections rate limit exceeded (`*`limit_value`*` )`	從用戶端到代理的連線要求速率已超過限制。
`ERROR 1040 (HY000): IAM authentication`	從用戶端到代理器的 IAM 身分驗證同時請求數目已超過限制。

錯誤	原因或因應措施
`rate limit exceeded`	
`ERROR 1040 (HY000): Number simultaneous connections exceeded (limit_value )`	從用戶端到代理的同時連線要求數目超過限制。
`ERROR 1045 (28000): Access denied for user 'DB_USER'@'%' (usi password: YES)`	代理使用的 Secrets Manager 私密不符合現有資料庫使用者的使用者名稱和密碼。更新 Secrets Manager 私密中的登入資料，或確認資料庫使用者存在，且具有與私密中相同的密碼。
`ERROR 1105 (HY000): Unknown error`	發生未知的錯誤。
`ERROR 1231 (42000): Variable ''character_set_client'' can't be set to the value of value`	為 `character_set_client` 參數設定的值無效。例如，值 `ucs2` 是無效的，因為它可能會讓 MySQL 伺服器當機。
`ERROR 3159 (HY000): This RDS Proxy requires TLS connections.`	您已在代理中啟用需要 Transport Layer Security設定，但您的連線已在 MySQL 用戶端中包含參數 `ssl-mode=DISABLED` 。執行下列任何一項： • 為代理停用需要 Transport Layer Security設定。 • 使用 MySQL 用戶端中 `ssl-mode=REQUIRED` 的最小設定來連線至資料庫。

錯誤	原因或因應措施
`ERROR 2026 (HY000): SSL connection error: Internal Server Error`	TLS 與代理的交握失敗。一些可能的原因包括： • SSL 是必要的，但伺服器不支援它。 • 發生內部伺服器錯誤。 • 發生錯誤的交握。
`ERROR 9501 (HY000): Timed-out waiting to acquire database connection`	代理等候取得資料庫連線逾時。一些可能的原因包括： • 代理無法建立資料庫連線，因為已達到最大連線數 • 代理無法建立資料庫連線，因為資料庫無法使用。

針對 RDS for PostgreSQL 的 RDS Proxy 問題進行疑難排解

連線到 PostgreSQL 代理時可能會遇到下列問題。

錯誤	原因	解決方案
`ERROR 28000: IAM authentication is allowed only with SSL connections.`	使用者嘗試使用 PostgreSQL 用戶端中的 `sslmode=disable` 設定，透過 IAM 身分驗證連線至資料庫。	使用者需要使用 PostgreSQL 用戶端中 `sslmode=require` 的最小設定連接到資料庫。如需詳細資訊，請參閱 PostgreSQL SSL 支援文件。
`ERROR 28000: This RDS proxy has no credentials for the role role_name . Check the credentials for this role and try again.`	這個角色沒有 Secrets Manager 秘密	新增此角色的 Secrets Manager 秘密。如需詳細資訊，請參閱設定 RDS Proxy 的 IAM 身分驗證。
`ERROR 28000: RDS supports only`	用來連線到代理的資料庫用戶端正在使用代理目前不支援的身分驗證機制。	如果您未使用 IAM 身分驗證，請使用 MD5 或 SCRAM 密碼身分驗證。

錯誤	原因	解決方案
`IAM, MD5, or SCRAM authentication.`		
`ERROR 28000: A user name is missing from the connectio n startup packet. Provide a user name for this connection.`	嘗試建立連線時，用來連線至 Proxy 的資料庫用戶端並未傳送使用者名稱。	使用您選擇的 PostgreSQL 用戶端設定與代理連線時，請務必定義使用者名稱。
`ERROR 28000: IAM is allowed only with SSL connections.`	用戶端嘗試使用 IAM 驗證進行連線，但未啟用 SSL。	在用戶端中啟用 SSL。
`ERROR 28000: This RDS Proxy requires TLS connections.`	使用者啟用需要 Transport Layer Security選項，但嘗試使用 PostgreSQL 用戶端中的 `sslmode=disable` 連線。	若要修正此錯誤，請執行下列其中一項操作： • 停用代理的需要 Transport Layer Security選項。 • 使用 PostgreSQL 用戶端中 `sslmode=allow` 的最小設定連線至資料庫。
`ERROR 28P01: IAM authentication failed for user` *`user_name`* `. Check the IAM token for this user and try again.`	此錯誤可能由下列原因造成： • 用戶端提供了不正確的 IAM 使用者名稱。 • 用戶端為使用者提供了不正確的 IAM 授權字符。 • 用戶端使用的 IAM 政策沒有必要許可。 • 用戶端為使用者提供了過期的 IAM 授權字符。	若要修正此錯誤，請執行下列動作： 1. 確認提供的 IAM 使用者存在。 2. 確認 IAM 授權字符屬於所提供的 IAM 使用者。 3. 確認 IAM 政策具有適當的 RDS 許可。 4. 檢查所使用之 IAM 授權字符的有效性。

錯誤	原因	解決方案
`ERROR 28P01: The password that was provided for the role role_name is wrong.`	此角色的密碼與 Secrets Manager 密碼不符。	在 Secrets Manager 檢查此角色的密碼，以查看密碼是否與 PostgreSQL 用戶端中使用的密碼相同。
`ERROR 28P01: The IAM authentication failed for the role role_name . Check the IAM token for this role and try again.`	用於 IAM 驗證的 IAM 符記發生問題。	產生一個新的身分驗證符記，並在新的連線中加以使用。
`ERROR 0A000: Feature not supported: RDS Proxy supports only version 3.0 of the PostgreSQL messaging protocol.`	用於連線到代理的 PostgreSQL 用戶端使用的通訊協定早於 3.0。	使用支援 3.0 訊息通訊協定的較新 PostgreSQL 用戶端。如果您使用的是 PostgreSQL `psql` CLI，請使用大於或等於 7.4 的版本。
`ERROR 0A000: Feature not supported: RDS Proxy currently doesn't support streaming replication mode.`	用來連線至 Proxy 的 PostgreSQL 用戶端正在嘗試使用串流複寫模式，而 RDS Proxy 目前不支援這種模式。	關閉用於連線的 PostgreSQL 用戶端中的串流複寫模式。
`ERROR 0A000: Feature not supported: RDS Proxy currently doesn't support the option option_name .`	透過啟動訊息，用來連線至 Proxy 的 PostgreSQL 用戶端正在要求 RDS Proxy 目前不支援的選項。	在用於連線的 PostgreSQL 用戶端中，關閉上述訊息中顯示為不支援的選項。

錯誤	原因	解決方案
`ERROR 53300: The IAM authentication failed because of too many competing requests.`	從用戶端到代理器的 IAM 身分驗證同時請求數目已超過限制。	降低使用 PostgreSQL 用戶端的 IAM 驗證建立連線的速率。
`ERROR 53300: The maximum number of client connections to the proxy exceeded` *`number_value`* `.`	從用戶端到代理的同時連線要求數目超過限制。	減少從 PostgreSQL 用戶端到此 RDS Proxy 的作用中連線數目。
`ERROR 53300: Rate of connection to proxy exceeded` *`number_value`* `.`	從用戶端到代理的連線要求速率已超過限制。	降低從 PostgreSQL 用戶端建立連線的速率。
`ERROR XX000: Unknown error.`	發生未知的錯誤。	請聯絡 AWS Support 以調查問題。
`ERROR 08000: Timed-out waiting to acquire database connection.`	代理逾時等待在 `ConnectionBorrowTimeout` 設定指定的持續時間內取得資料庫連線。一些可能的原因包括： • 代理無法建立資料庫連線，因為已達到最大連線數 • 代理無法建立資料庫連線，因為資料庫無法使用，或建立資料庫連線的時間超過設定的 `ConnectionBorrowTimeout` 。	可能的解決方案如下： • 避免鎖定代理連線。請參閱避免鎖定 RDS Proxy。 • 檢閱 `ConnectionBorrowTimeout` 和 `MaxConnectionsPercent` 設定。請參閱 RDS Proxy 連線考量事項。 • 檢閱目標可用性。請參閱使用 Amazon CloudWatch 監控 RDS Proxy 指標中的`AvailabilityPercentage` 。

錯誤	原因	解決方案
ERROR XX000: Request returned an error: *database_error* .	從 Proxy 建立的資料庫連線傳回錯誤。	解決方案取決於特定的資料庫錯誤。其中一個範例為 Request returned an error: database "your-database-name" does not exist。這表示指定的資料庫名稱不存在於資料庫伺服器上。或者，它表示用作資料庫名稱的使用者名稱 (如果未指定資料庫名稱) 不存在於伺服器上。
ERROR 53300: The IAM authentication failed because of too many competing requests.	從用戶端到代理器的 IAM 身分驗證同時請求數目已超過限制。	降低使用 PostgreSQL 用戶端的 IAM 驗證建立連線的速率。
ERROR 28000: Enable IAM authentication for the client connection to the proxy and try again.	RDS Proxy 無法連線至資料庫，因為用戶端連線至代理並未啟用 IAM 身分驗證。當代理的DefaultAuthScheme 參數IAM_AUTH設為具有註冊使用者的 ，但用戶端使用密碼身分驗證而非 IAM 身分驗證時，就會發生這種情況。	啟用用戶端連線至代理的 IAM 身分驗證，然後再試一次。
ERROR 28000: Configure IAM authentication as the DefaultAuthScheme in your proxy and try again.	RDS Proxy 無法連線至資料庫，因為 DefaultAuthScheme 未設定為 IAM_AUTH。代理的 DefaultAuthScheme 參數設定為 NONE，但用戶端嘗試使用 IAM 身分驗證。	IAM_AUTH 為您的代理將 DefaultAuthScheme 設定為 ，然後再試一次。

針對已刪除的 **postgres** 資料庫進行疑難排解

如果您錯誤地從執行個體刪除 postgres 資料庫，則需要還原資料庫以還原執行個體的連線。在您的資料庫執行個體內執行下列命令：

```
CREATE DATABASE postgres;
GRANT CONNECT ON DATABASE postgres TO rdsproxyadmin;
```

將 RDS Proxy 與 AWS CloudFormation 搭配使用

您可以將 RDS Proxy 與 AWS CloudFormation 搭配使用。這可協助您建立相關資源的群組。這類群組可以包含可連線至新建立之 Aurora 資料庫叢集的 Proxy。CloudFormation 中的 RDS Proxy 支援包含兩個新的註冊類型：DBProxy 和 DBProxyTargetGroup。

下列清單會顯示 RDS Proxy 的範例 CloudFormation 範本。

```
Resources:
 DBProxy:
   Type: AWS::RDS::DBProxy
   Properties:
     DBProxyName: CanaryProxy
     EngineFamily: MYSQL
     RoleArn:
      Fn::ImportValue: SecretReaderRoleArn
     Auth:
       - {AuthScheme: SECRETS, SecretArn: !ImportValue ProxySecret, IAMAuth: DISABLED}
     VpcSubnetIds:
       Fn::Split: [",", "Fn::ImportValue": SubnetIds]

 ProxyTargetGroup:
   Type: AWS::RDS::DBProxyTargetGroup
   Properties:
     DBProxyName: CanaryProxy
     TargetGroupName: default
     DBInstanceIdentifiers:
       - Fn::ImportValue: DBInstanceName
   DependsOn: DBProxy
```

如需詳細了解此範例中的資源，請參閱 DBProxy 和 DBProxyTargetGroup。

如需有關您可以使用 CloudFormation 建立的資源的詳細資訊，請參閱 RDS 資源類型參考。

搭配 Aurora 全域資料庫使用 RDS Proxy

Aurora 全域資料庫是跨多個 AWS 區域 的單一資料庫，允許低延遲的全域讀取，以及全區域中斷的災難復原。它為您的部署提供內建容錯，因為資料庫執行個體不依賴單一 AWS 區域，而是依賴多個區域和不同的可用區域。如需更多詳細資訊，請參閱 使用 Amazon Aurora 全球資料庫。

您可以搭配 Aurora 全域資料庫中的任何資料庫叢集使用 RDS Proxy。在開始一起使用這些功能之前，請確定您已熟悉下列資訊。

> **Important**
>
> 如果資料庫叢集屬於已開啟寫入轉送的全域資料庫，請依配置給寫入轉送的配額減少代理的 `MaxConnectionsPercent` 值。寫入轉送配額是在資料庫叢集參數 `aurora_fwd_writer_max_connections_pct` 中設定的。如需寫入轉送的資訊，請參閱 在 Amazon Aurora 全域資料庫中使用寫入轉送。

RDS Proxy 搭配全域資料庫的限制

當 Aurora 資料庫叢集已開啟寫入轉送時，RDS Proxy 不支援 `aurora_replica_read_consistency` 變數的 `SESSION` 值。設定此值可能會導致未預期的行為。

RDS Proxy 端點如何使用全域資料庫

當了解 RDS Proxy 端點如何使用全域資料庫時，您可以更妥善地管理搭配這兩個功能使用 Aurora 資料庫的應用程式。

對於將全域資料庫的主要叢集作為註冊目標的代理，代理端點的運作方式與任何 Aurora DB 叢集的運作方式相同。代理的讀取/寫入端點會將所有請求傳送至叢集的寫入器執行個體。代理的唯讀端點會將所有請求傳送至讀取器執行個體。如果讀取器在連線開啟時變成無法使用，RDS Proxy 會將連線上的後續查詢重新導向至另一個讀取器執行個體。對於將次要叢集作為註冊目標的代理，傳送至代理唯讀端點的請求也會傳送至讀取器執行個體。因為叢集沒有寫入器執行個體，所以傳送至讀取/寫入端點的請求會失敗，錯誤為 `"The target group doesn't have any associated read/write instances"`。

全球資料庫轉換和容錯移轉操作都涉及主要和其中一個次要資料庫叢集之間的角色切換。當選取的次要叢集成為新的主要叢集時，其中一個讀取器執行個體會提升為寫入器。此資料庫執行個體現在是全域叢集的新寫入器執行個體。請務必將應用程式的寫入操作重新導向至與新主要叢集相關聯之代理的適當讀取/寫入端點。此代理端點可能是預設端點或自訂讀取/寫入端點。

RDS Proxy 會透過讀取/寫入端點將所有要求排入佇列，並在新主要叢集的寫入器執行個體一旦可用，就會將這些請求傳送至其中。無論轉換或容錯移轉操作是否已完成，都會進行角色切換。在轉換或容錯移轉期間，舊主要叢集的代理預設端點仍會接受寫入操作。不過，該叢集一旦變成次要叢集，所有寫入操作就會失敗。若要了解如何以及何時執行特定的全球轉換域容錯移轉任務，請參閱下列主題：

- 全球資料庫轉換 - 針對 Amazon Aurora 全球資料庫執行轉換
- 全球資料庫容錯移轉 - 從計劃外中斷復原 Amazon Aurora 全域資料庫

Aurora零 ETL 整合

Aurora與 Amazon Redshift 的零 ETL 整合，並使用 Aurora 的資料 Amazon SageMaker AI 啟用近乎即時的分析和機器學習 (ML)。其是一種全受管解決方案，可讓交易資料在寫入 Aurora 資料庫叢集之後，於分析目的地中使用。擷取、轉換和載入 (ETL) 是一種過程，可將來自多個來源的資料合併為大型、中央資料倉儲。

零 ETL 整合可讓 Aurora 資料庫叢集 中的資料近乎即時地在 Amazon Redshift 或 Amazon SageMaker AI 資料湖倉中提供。一旦資料位於目標資料倉儲或資料湖中，您可以使用內建功能來支援分析、ML 和 AI 工作負載，例如機器學習、具體化視觀表、資料共用、對多個資料存放區和資料湖的聯合存取，以及與 Amazon SageMaker AI、Quick 和其他 整合 AWS 服務。

若要建立零 ETL 整合時，請指定 Aurora 資料庫叢集做為來源，以及指定支援資料倉儲或資料湖倉做為目標。整合會將來源資料庫中的資料複寫到目標資料倉儲或資料湖倉。

下圖說明與 Amazon Redshift 進行零 ETL 整合的此功能：



下圖說明與 Amazon SageMaker AI 資料湖倉進行零 ETL 整合的此功能：



整合會監控資料管道的運作狀態，並在可能的情況下從問題中復原。您可以建立從多個 Aurora 資料庫叢集到單一目標資料倉儲或資料湖倉的整合，讓您能夠跨多個應用程式衍生洞察。

如需零 ETL 整合定價的詳細資訊，請參閱 Amazon Aurora 定價和 Amazon Redshift 定價。

主題

- 優勢
- 重要概念
- 限制
- 配額
- 支援的區域
- 開始使用 Aurora 零 ETL 整合
- 建立與 Amazon Redshift 的 Aurora 零 ETL 整合
- 建立與 Amazon SageMaker 資料湖倉的 Aurora 零 ETL 整合
- Aurora 零 ETL 整合的資料篩選
- 將資料新增至來源 Aurora 資料庫叢集並進行查詢
- 檢視和監控 Aurora 零 ETL 整合
- 修改 Aurora 零 ETL 整合
- 刪除 Aurora 零 ETL 整合

- 針對 Aurora 零 ETL 整合進行疑難排解

優勢

Aurora 零 ETL 整合具有下列優勢：

- 協助您從多個資料來源衍生整體洞見。
- 無需建置和維護執行擷取、轉換和載入 (ETL) 操作的複雜資料管道。Zero-ETL 整合會透過為您佈建和管理管道，免除建置和管理這些管道所帶來的挑戰。
- 減少操作負擔和成本，讓您專注於改善應用程式。
- 讓您利用目標目的地的分析和 ML 功能，從交易和其他資料衍生洞見，以有效地回應關鍵、時間敏感的事件。

重要概念

當您開始使用零 ETL 整合時，請考慮下列概念：

整合

全受管資料管道，其會自動將交易資料和結構描述從 Aurora 資料庫叢集複寫到資料倉儲或目錄。

來源資料庫叢集

從中複寫資料的 Aurora 資料庫叢集。您可以指定使用佈建資料庫執行個體或資料庫執行個體做為來源的Aurora Serverless v2資料庫叢集。

Target

將資料複寫至其中的資料倉儲或資料湖倉。資料倉儲有兩種類型：佈建的叢集資料倉儲和無伺服器資料倉儲。佈建的叢集資料倉儲是稱為節點的運算資源集合，這些節點會組織成稱為叢集的群組。無伺服器資料倉儲由存放運算資源的工作群組，以及為資料庫物件和使用者提供空間的命名空間所組成。這兩個資料倉儲都會執行分析引擎，並包含一或多個資料庫。

目標資料湖倉由目錄、資料庫、資料表和檢視組成。如需資料湖倉架構的詳細資訊，請參閱《Amazon SageMaker AI Unified Studio 使用者指南》中的 SageMaker Lakehouse components。

多個來源資料庫叢集可以寫入至相同的目標。

如需詳細資訊，請參閱《Amazon Redshift 開發人員指南》中的資料倉儲系統架構。

限制

以下限制適用於 Aurora 零 ETL 整合。

主題

- 一般限制
- Aurora MySQL 限制
- Aurora PostgreSQL 限制
- Amazon Redshift 限制
- Amazon SageMaker AI Lakehouse 限制

一般限制

- 來源資料庫叢集必須與目標位於相同的區域。
- 您無法重新命名資料庫叢集或其任何執行個體，如果它具有現有整合的話。
- 您無法在相同的來源和目標資料庫之間建立多個整合。
- 您無法刪除具有現有整合的資料庫叢集。您必須先刪除所有相關聯的整合。
- 如果停止來源資料庫叢集，則最後幾個交易可能不會複寫到目標，直到您恢復叢集為止。
- 如果您的叢集是藍/綠部署的來源，則藍色和綠色環境在轉換期間無法具有現有的零 ETL 整合。您必須先刪除整合再進行轉換，然後重新建立該整合。
- 資料庫叢集必須至少包含一個資料庫執行個體，才能成為整合的來源。
- 您無法為跨帳戶複製的來源資料庫叢集建立整合，例如使用 AWS Resource Access Manager () 共用的資料庫叢集AWS RAM。
- 如果來源叢集是 Aurora 全域資料庫中的主要資料庫叢集，且容錯移轉到其中一個次要叢集，則整合會變成非作用中。您必須刪除並重新建立整合。
- 您無法為正在主動建立另一個整合的來源資料庫建立整合。
- 當您一開始建立整合時或當資料表重新同步時，從來源植入目標的資料可能需要 20 到 25 分鐘或更長時間，取決於來源資料庫的大小。此延遲可能會導致複本延遲增加。
- 不支援某些資料類型。如需詳細資訊，請參閱the section called “資料類型差異”。
- 系統資料表、暫存資料表和檢視不會複寫到目標倉儲。
- 在來源資料表上執行 DDL 命令 (例如 ALTER TABLE) 可以觸發資料表重新同步，使得資料表無法在重新同步時進行查詢。如需詳細資訊，請參閱the section called “我的一個或多個 Amazon Redshift 資料表需要重新同步”。

Aurora MySQL 限制

- 您的來源資料庫叢集必須執行支援的 Aurora MySQL 版本。如需支援的版本的清單，請參閱the section called “零 ETL 整合 ”。
- 零 ETL 整合依賴 MySQL 二進位記錄 (binlog)，來擷取持續的資料變更。不要使用 Binlog 型資料篩選，因為其可能會在來源資料庫與目標資料庫之間導致資料不一致。
- 僅針對設定為使用 InnoDB 儲存體引擎的資料庫支援零 ETL 整合。
- 不支援具有預先定義之資料表更新的外部索引鍵參考。具體而言，不支援 `ON DELETE` 和 `ON UPDATE` 規則搭配 `CASCADE`、`SET NULL` 和 `SET DEFAULT` 動作。嘗試建立或更新對另一個資料表具有這類參考的資料表，會將該資料表置於失敗狀態。
- 在來源資料庫叢集上執行的 XA 交易會導致整合進入 `Syncing` 狀態。

Aurora PostgreSQL 限制

- 您的來源資料庫叢集必須執行支援的 Aurora PostgreSQL 版本。如需支援的版本的清單，請參閱the section called “零 ETL 整合 ”。
- 如果您選取 Aurora PostgreSQL 來源資料庫叢集，則必須至少指定一個資料篩選條件模式。模式至少必須包含單一資料庫 (*`database-name`*`.*.*`)，才能複寫至目標倉儲。如需詳細資訊，請參閱the section called “零 ETL 整合的資料篩選”。
- 在來源 Aurora PostgreSQL 資料庫叢集內建立的所有資料庫都必須使用 UTF-8 編碼。
- 使用宣告式分割時，資料表分割區會複寫至 Amazon Redshift。不過，分割的資料表本身不會複寫到 Amazon Redshift。
- 不支援兩階段交易。
- 如果您從整合來源的資料庫叢集刪除所有資料庫執行個體，然後重新新增資料庫執行個體，則來源與目標叢集之間的複寫會中斷。
- 來源資料庫叢集無法使用 Aurora Limitless Database。
- 資料篩選條件中存在的所有資料表都需要主索引鍵。任何沒有主索引鍵的資料表都會進入失敗狀態。

Amazon Redshift 限制

如需與零 ETL 整合相關的 Amazon Redshift 限制清單，請參閱《Amazon Redshift 管理指南》中的將零 ETL 整合與 Amazon Redshift 搭配使用的考量。

Amazon SageMaker AI Lakehouse 限制

以下是湖房零 ETL Amazon SageMaker AI 整合的限制。

- 目錄名稱的長度限制為 19 個字元。

配額

您的帳戶具有與 Aurora 零 ETL 整合相關的下列配額。除非另有說明，否則每個配額都是根據區域而定。

名稱	預設	Description
整合	100	AWS 帳戶內的整合總數。
每個目標的整合	50	將資料傳送至單一目標資料倉儲或資料湖倉的整合數目。
每個來源叢集的整合	5	從單一來源資料庫叢集傳送資料的整合數目。

此外，目標倉儲會對每個資料庫執行個體或叢集節點中允許的資料表數目設定某些限制。如需 Amazon Redshift 配額和限制的詳細資訊，請參閱《Amazon Redshift 管理指南》中的 Amazon Redshift 中的配額和限制。

支援的區域

Aurora零 ETL 整合可在 的子集中使用 AWS 區域。如需支援的區域的清單，請參閱 the section called “零 ETL 整合 ”。

開始使用 Aurora 零 ETL 整合

在建立零 ETL 整合之前，請先使用必要的參數和許可設定 Aurora 資料庫叢集和資料倉儲。安裝期間，您將完成以下步驟：

1. 建立自訂資料庫叢集參數群組。

2. 建立來源資料庫叢集。
3. 為 Amazon Redshift 建立目標資料倉儲或建立目標 Amazon SageMaker Lakehouse。

完成這些任務後，請繼續 the section called “建立與 Amazon Redshift 的零 ETL 整合” 或 the section called “建立與 Amazon SageMaker 資料湖倉的零 ETL 整合”。

您可以使用 AWS SDKs 為您自動化設定程序。如需詳細資訊，請參閱the section called “使用 AWS SDKs 設定整合”。

> **Tip**
>
> 您可以在建立整合時讓 RDS 為您完成這些設定步驟，而不用手動執行這些步驟。若要立即開始建立整合，請參閱 the section called “建立與 Amazon Redshift 的零 ETL 整合”。

對於步驟 3，您可以選擇根據您的需求建立目標資料倉儲 (步驟 3a) 或目標湖房 (步驟 3b)：

- 如果您需要使用 SQL 型分析的傳統資料倉儲功能，請選擇資料倉儲。
- 如果您需要機器學習功能，並想要將 Lakehouse 功能用於資料科學和 ML 工作流程，請選擇 Amazon SageMaker Lakehouse。

步驟 1：建立自訂資料庫叢集參數群組。

Aurora 零 ETL 整合需要控制複寫之資料庫叢集參數的特定值。具體而言，Aurora MySQL 需要增強型 binlog (`aurora_enhanced_binlog`)，而 Aurora PostgreSQL 需要增強型邏輯複寫 (`aurora.enhanced_logical_replication`)。

若要設定二進位記錄或邏輯複寫，您必須先建立自訂資料庫叢集參數群組，然後將其與來源資料庫叢集建立關聯。

Aurora MySQL (aurora-mysql8.0 系列）：

- `aurora_enhanced_binlog=1`
- `binlog_backup=0`
- `binlog_format=ROW`
- `binlog_replication_globaldb=0`
- `binlog_row_image=full`

- `binlog_row_metadata=full`

此外，請確定 `binlog_transaction_compression` 參數未設定為 `ON`，且 `binlog_row_value_options` 參數未設定為 `PARTIAL_JSON`。

如需 Aurora MySQL 增強型 binlog 的詳細資訊，請參閱 the section called “設定增強型 Binlog”。

Aurora PostgreSQL (aurora-postgresql16 系列）：

- `rds.logical_replication=1`
- `aurora.enhanced_logical_replication=1`
- `aurora.logical_replication_backup=0`
- `aurora.logical_replication_globaldb=0`

即使 `REPLICA IDENTITY FULL` 未啟用，啟用增強型邏輯複寫 (`aurora.enhanced_logical_replication`) 仍一律會將所有資料欄值寫入預寫日誌 (WAL)。這可能會增加來源資料庫叢集的 IOPS。

> **Important**
>
> 如果您啟用或停用 `aurora.enhanced_logical_replication` 資料庫叢集參數，主要資料庫執行個體會讓所有邏輯複寫槽失效。這會停止從來源到目標的複寫，而且您必須在主要資料庫執行個體上重新建立複寫槽。為了防止中斷，請在複寫期間保持參數狀態一致。

步驟 2：選取或建立來源資料庫叢集

在建立自訂資料庫叢集參數群組之後，請選擇或建立 Aurora 資料庫叢集。此叢集將是複寫至目標資料倉儲之資料的來源。您可以指定一個資料庫叢集，其會使用佈建的資料庫執行個體或 Aurora Serverless v2 資料庫執行個體做為來源。如需建立資料庫叢集的指示，請參閱 the section called “建立資料庫叢集” 或 the section called “建立 Aurora Serverless v2 資料庫叢集”。

資料庫必須執行支援的資料庫引擎版本。如需支援的版本的清單，請參閱the section called “零 ETL 整合 ”。

當您建立資料庫時，在其他組態下，將預設資料庫叢集參數群組變更為您在上一個步驟中建立的自訂參數群組。

> **Note**
>
> 如果您在已建立叢集之後將參數群組與資料庫叢集建立關聯，則必須將叢集中的主要資料庫執行個體重新開機以套用變更，然後才能建立零 ETL 整合。如需相關指示，請參閱 the section called “重新啟動 Aurora 資料庫叢集或執行個體”。

步驟 3a：建立目標資料倉儲

在建立來源資料庫叢集之後，您必須建立並設定目標資料倉儲。資料倉儲必須符合下列需求：

- 使用具有至少兩個節點的 RA3 節點類型，或 Redshift Serverless。
- 已加密 (如果使用已佈建的叢集)。如需詳細資訊，請參閱 Amazon Redshift 資料庫加密。

如需建立資料倉儲的指示，請參閱建立叢集 (適用於佈建的叢集)，或使用命名空間建立工作群組 (適用於 Redshift Serverless)。

在資料倉儲上啟用區分大小寫

若要成功整合，必須為資料倉儲啟用區分大小寫參數 (`enable_case_sensitive_identifier`)。依預設，所有佈建的叢集和 Redshift Serverless 工作群組上都會停用區分大小寫。

若要啟用區分大小寫，請根據您的資料倉儲類型執行下列步驟：

- 佈建的叢集 – 若要在佈建的叢集上啟用區分大小寫，請建立已啟用 `enable_case_sensitive_identifier` 參數的自訂參數群組。接著，將該參數群組與叢集建立關聯。如需指示，請參閱使用主控台管理參數群組或使用 AWS CLI設定參數值。

  > **Note**
  >
  > 在將自訂參數群組與叢集建立關聯之後，請記得重新啟動該叢集。

- 無伺服器工作群組 - 若要在 Redshift Serverless 工作群組上啟用區分大小寫，您必須使用 AWS CLI。Amazon Redshift 主控台目前不支援修改 Redshift Serverless 參數值。傳送下列 update-workgroup 要求：

```
aws redshift-serverless update-workgroup \
  --workgroup-name target-workgroup \
```

```
 --config-parameters
parameterKey=enable_case_sensitive_identifier,parameterValue=true
```

在修改工作群組的參數值之後，您不需要重新啟動該工作群組。

設定資料倉儲的授權

在建立資料倉儲之後，您必須將來源 Aurora 資料庫叢集設定為授權的整合來源。如需指示，請參閱設定 Amazon Redshift 資料倉儲的授權。

使用 AWS SDKs 設定整合

您可以執行下列 Python 指令碼來為您自動設定所需的資源，而不用手動設定每個資源。程式碼範例使用 適用於 Python (Boto3) 的 AWS SDK 建立來源 Amazon Aurora 資料庫叢集和目標資料倉儲，每個都具有必要的參數值。然後，在資料庫之間建立零 ETL 整合之前，它會等待資料庫可用。您可以根據您需要設定的資源來註解不同的函數。

若要安裝所需的相依性，請執行下列命令：

```
pip install boto3
pip install time
```

在指令碼中，選擇性地修改來源、目標和參數群組的名稱。最終函數會在設定資源後建立名為 `my-integration` 的整合。

Python 程式碼範例

Aurora MySQL

```
import boto3
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default Region.

rds = boto3.client('rds')
redshift = boto3.client('redshift')
sts = boto3.client('sts')

source_cluster_name = 'my-source-cluster' # A name for the source cluster
```

```
source_param_group_name = 'my-source-param-group' # A name for the source parameter
 group
target_cluster_name = 'my-target-cluster' # A name for the target cluster
target_param_group_name = 'my-target-param-group' # A name for the target parameter
 group

def create_source_cluster(*args):
    """Creates a source Aurora MySQL DB cluster"""

    response = rds.create_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        DBParameterGroupFamily='aurora-mysql8.0',
        Description='For Aurora MySQL binary logging'
    )
    print('Created source parameter group: ' + response['DBClusterParameterGroup']
['DBClusterParameterGroupName'])

    response = rds.modify_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        Parameters=[
            {
                'ParameterName': 'aurora_enhanced_binlog',
                'ParameterValue': '1',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_backup',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_format',
                'ParameterValue': 'ROW',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_replication_globaldb',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_row_image',
                'ParameterValue': 'full',
                'ApplyMethod': 'pending-reboot'
```

```
            },
            {
                'ParameterName': 'binlog_row_metadata',
                'ParameterValue': 'full',
                'ApplyMethod': 'pending-reboot'
            }
        ]
    )
    print('Modified source parameter group: ' +
 response['DBClusterParameterGroupName'])

    response = rds.create_db_cluster(
        DBClusterIdentifier=source_cluster_name,
        DBClusterParameterGroupName=source_param_group_name,
        Engine='aurora-mysql',
        EngineVersion='8.0.mysql_aurora.3.05.2',
        DatabaseName='myauroradb',
        MasterUsername='username',
        MasterUserPassword='Password01**'
    )
    print('Creating source cluster: ' + response['DBCluster']
['DBClusterIdentifier'])
    source_arn = (response['DBCluster']['DBClusterArn'])
    create_target_cluster(target_cluster_name, source_arn, target_param_group_name)

    response = rds.create_db_instance(
        DBInstanceClass='db.r6g.2xlarge',
        DBClusterIdentifier=source_cluster_name,
        DBInstanceIdentifier=source_cluster_name + '-instance',
        Engine='aurora-mysql'
    )
    return(response)

def create_target_cluster(target_cluster_name, source_arn, target_param_group_name):
    """Creates a target Redshift cluster"""

    response = redshift.create_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        ParameterGroupFamily='redshift-1.0',
        Description='For Aurora MySQL zero-ETL integrations'
    )
    print('Created target parameter group: ' + response['ClusterParameterGroup']
['ParameterGroupName'])
```

```
    response = redshift.modify_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        Parameters=[
            {
                'ParameterName': 'enable_case_sensitive_identifier',
                'ParameterValue': 'true'
            }
        ]
    )
    print('Modified target parameter group: ' + response['ParameterGroupName'])

    response = redshift.create_cluster(
        ClusterIdentifier=target_cluster_name,
        NodeType='ra3.4xlarge',
        NumberOfNodes=2,
        Encrypted=True,
        MasterUsername='username',
        MasterUserPassword='Password01**',
        ClusterParameterGroupName=target_param_group_name
    )
    print('Creating target cluster: ' + response['Cluster']['ClusterIdentifier'])

    # Retrieve the target cluster ARN
    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Retrieve the current user's account ID
    response = sts.get_caller_identity()
    account_id = response['Account']

    # Create a resource policy specifying cluster ARN and account ID
    response = redshift.put_resource_policy(
        ResourceArn=target_arn,
        Policy='''
        {
            \"Version\":\"2012-10-17\",
            \"Statement\":[
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"Service\":\"redshift.amazonaws.com\"
                },
                \"Action\":[\"redshift:AuthorizeInboundIntegration\"],
```

```
                \"Condition\":{
                    \"StringEquals\":{
                        \"aws:SourceArn\":\"%s\"}
                    }
                },
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"AWS\":\"arn:aws:iam::%s:root\"},
                \"Action\":\"redshift:CreateInboundIntegration\"}
            ]
        }
        ''' % (source_arn, account_id)
    )
    return(response)

def wait_for_cluster_availability(*args):
    """Waits for both clusters to be available"""

    print('Waiting for clusters to be available...')

    response = rds.describe_db_clusters(
        DBClusterIdentifier=source_cluster_name
    )
    source_status = response['DBClusters'][0]['Status']
    source_arn = response['DBClusters'][0]['DBClusterArn']

    response = rds.describe_db_instances(
        DBInstanceIdentifier=source_cluster_name + '-instance'
    )
    source_instance_status = response['DBInstances'][0]['DBInstanceStatus']

    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_status = response['Clusters'][0]['ClusterStatus']
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Every 60 seconds, check whether the clusters are available.
    if source_status != 'available' or target_status != 'available' or
 source_instance_status != 'available':
        time.sleep(60)
        response = wait_for_cluster_availability(
            source_cluster_name, target_cluster_name)
    else:
```

```
        print('Clusters available. Ready to create zero-ETL integration.')
        create_integration(source_arn, target_arn)
        return

def create_integration(source_arn, target_arn):
    """Creates a zero-ETL integration using the source and target clusters"""

    response = rds.create_integration(
        SourceArn=source_arn,
        TargetArn=target_arn,
        IntegrationName='my-integration'
    )
    print('Creating integration: ' + response['IntegrationName'])

def main():
    """main function"""
    create_source_cluster(source_cluster_name, source_param_group_name)
    wait_for_cluster_availability(source_cluster_name, target_cluster_name)

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

Aurora PostgreSQL

```
import boto3
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default Region.

rds = boto3.client('rds')
redshift = boto3.client('redshift')
sts = boto3.client('sts')

source_cluster_name = 'my-source-cluster' # A name for the source cluster
source_param_group_name = 'my-source-param-group' # A name for the source parameter
 group
target_cluster_name = 'my-target-cluster' # A name for the target cluster
target_param_group_name = 'my-target-param-group' # A name for the target parameter
 group

def create_source_cluster(*args):
```

```
    """Creates a source Aurora PostgreSQL DB cluster"""

    response = rds.create_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        DBParameterGroupFamily='aurora-postgresql16',
        Description='For Aurora PostgreSQL logical replication'
    )
    print('Created source parameter group: ' + response['DBClusterParameterGroup']
['DBClusterParameterGroupName'])

    response = rds.modify_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        Parameters=[
            {
                'ParameterName': 'rds.logical_replication',
                'ParameterValue': '1',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'aurora.enhanced_logical_replication',
                'ParameterValue': '1',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'aurora.logical_replication_backup',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'aurora.logical_replication_globaldb',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            }
        ]
    )
    print('Modified source parameter group: ' +
 response['DBClusterParameterGroupName'])

    response = rds.create_db_cluster(
        DBClusterIdentifier=source_cluster_name,
        DBClusterParameterGroupName=source_param_group_name,
        Engine='aurora-postgresql',
        EngineVersion='16.4.aurora-postgresql',
        DatabaseName='mypostgresdb',
```

```
        MasterUsername='username',
        MasterUserPassword='Password01**'
    )
    print('Creating source cluster: ' + response['DBCluster']
['DBClusterIdentifier'])
    source_arn = (response['DBCluster']['DBClusterArn'])
    create_target_cluster(target_cluster_name, source_arn, target_param_group_name)

    response = rds.create_db_instance(
        DBInstanceClass='db.r6g.2xlarge',
        DBClusterIdentifier=source_cluster_name,
        DBInstanceIdentifier=source_cluster_name + '-instance',
        Engine='aurora-postgresql'
    )
    return(response)

def create_target_cluster(target_cluster_name, source_arn, target_param_group_name):
    """Creates a target Redshift cluster"""

    response = redshift.create_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        ParameterGroupFamily='redshift-1.0',
        Description='For Aurora PostgreSQL zero-ETL integrations'
    )
    print('Created target parameter group: ' + response['ClusterParameterGroup']
['ParameterGroupName'])

    response = redshift.modify_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        Parameters=[
            {
                'ParameterName': 'enable_case_sensitive_identifier',
                'ParameterValue': 'true'
            }
        ]
    )
    print('Modified target parameter group: ' + response['ParameterGroupName'])

    response = redshift.create_cluster(
        ClusterIdentifier=target_cluster_name,
        NodeType='ra3.4xlarge',
        NumberOfNodes=2,
        Encrypted=True,
        MasterUsername='username',
```

```
        MasterUserPassword='Password01**',
        ClusterParameterGroupName=target_param_group_name
    )
    print('Creating target cluster: ' + response['Cluster']['ClusterIdentifier'])

    # Retrieve the target cluster ARN
    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Retrieve the current user's account ID
    response = sts.get_caller_identity()
    account_id = response['Account']

    # Create a resource policy specifying cluster ARN and account ID
    response = redshift.put_resource_policy(
        ResourceArn=target_arn,
        Policy='''
        {
            \"Version\":\"2012-10-17\",
            \"Statement\":[
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"Service\":\"redshift.amazonaws.com\"
                },
                \"Action\":[\"redshift:AuthorizeInboundIntegration\"],
                \"Condition\":{
                    \"StringEquals\":{
                        \"aws:SourceArn\":\"%s\"}
                    }
                },
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"AWS\":\"arn:aws:iam::%s:root\"},
                \"Action\":\"redshift:CreateInboundIntegration\"}
            ]
        }
        ''' % (source_arn, account_id)
    )
    return(response)

def wait_for_cluster_availability(*args):
    """Waits for both clusters to be available"""
```

```
    print('Waiting for clusters to be available...')

    response = rds.describe_db_clusters(
        DBClusterIdentifier=source_cluster_name
    )
    source_status = response['DBClusters'][0]['Status']
    source_arn = response['DBClusters'][0]['DBClusterArn']

    response = rds.describe_db_instances(
        DBInstanceIdentifier=source_cluster_name + '-instance'
    )
    source_instance_status = response['DBInstances'][0]['DBInstanceStatus']

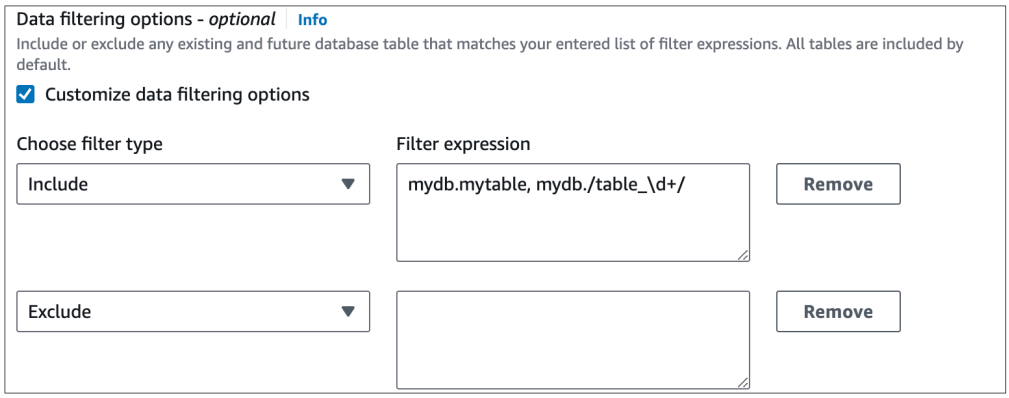
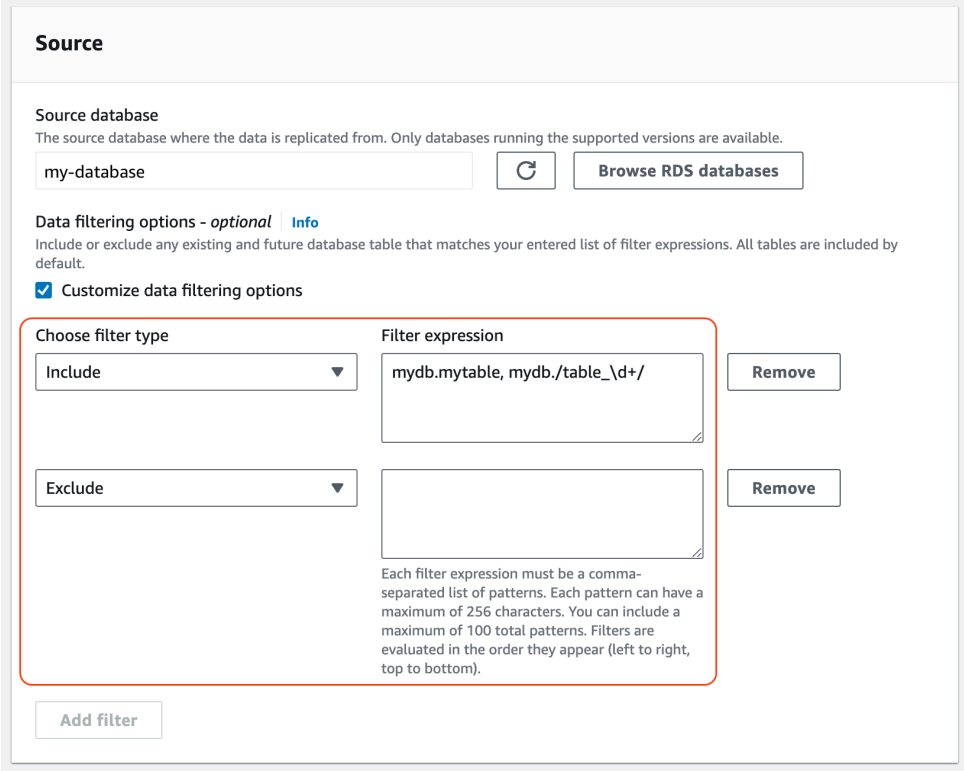
    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_status = response['Clusters'][0]['ClusterStatus']
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Every 60 seconds, check whether the clusters are available.
    if source_status != 'available' or target_status != 'available' or
 source_instance_status != 'available':
        time.sleep(60)
        response = wait_for_cluster_availability(
            source_cluster_name, target_cluster_name)
    else:
        print('Clusters available. Ready to create zero-ETL integration.')
        create_integration(source_arn, target_arn)
        return

def create_integration(source_arn, target_arn):
    """Creates a zero-ETL integration using the source and target clusters"""

    response = rds.create_integration(
        SourceArn=source_arn,
        TargetArn=target_arn,
        IntegrationName='my-integration'
    )
    print('Creating integration: ' + response['IntegrationName'])

def main():
    """main function"""
    create_source_cluster(source_cluster_name, source_param_group_name)
```

```
    wait_for_cluster_availability(source_cluster_name, target_cluster_name)

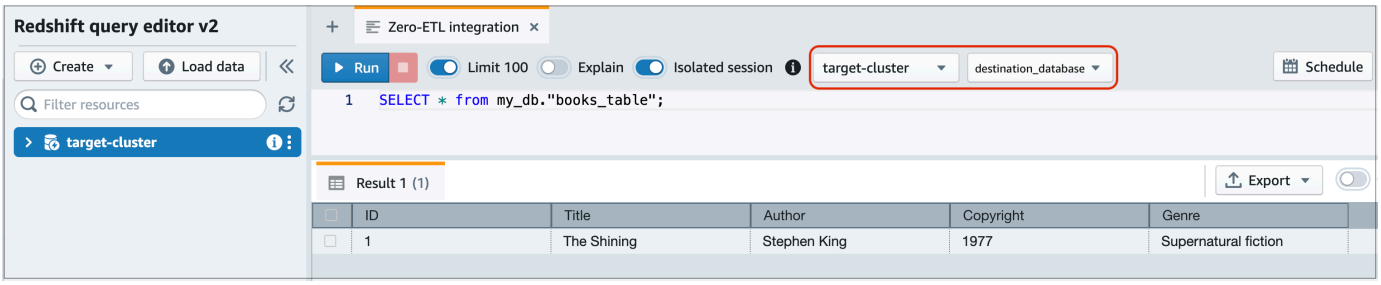
if __name__ == "__main__":
    main()
```

步驟 3b：建立 Amazon SageMaker Lakehouse 零 ETL 整合的 AWS Glue 目錄

與 Amazon SageMaker Lakehouse 建立零 ETL 整合時，您必須在其中建立 AWS Glue 受管目錄 AWS Lake Formation。目標目錄必須是 Amazon Redshift 受管目錄。若要建立 Amazon Redshift 受管目錄，請先建立 `AWSServiceRoleForRedshift` 服務連結角色。在 Lake Formation 主控台中，將 `AWSServiceRoleForRedshift` 新增為唯讀管理員。

如需有關先前任務的詳細資訊，請參閱下列主題。

- 如需有關建立 Amazon Redshift 受管目錄的詳細資訊，請參閱《AWS Lake Formation 開發人員指南》中的在 AWS Glue Data Catalog中建立 Amazon Redshift 受管目錄。
- 如需 Amazon Redshift 服務連結角色的詳細資訊，請參閱《Amazon Redshift 管理指南》中的使用 Amazon Redshift 的服務連結角色。
- 如需 Lake Formation 唯讀管理員許可的詳細資訊，請參閱《AWS Lake Formation 開發人員指南》中的 Lake Formation 角色和 IAM 許可參考。

設定目標 AWS Glue 目錄的許可

在建立零 ETL 整合的目標目錄之前，您必須建立 Lake Formation 目標建立角色和 AWS Glue 資料傳輸角色。使用 Lake Formation 目標建立角色來建立目標目錄。建立目標目錄時，請在從引擎存取區段的 IAM 角色欄位中輸入 Glue 資料傳輸角色。

Lake Formation 目標建立角色

目標建立角色必須是 Lake Formation 管理員，且需要下列許可。

JSON

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

```
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "lakeformation:RegisterResource",
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "s3:PutEncryptionConfiguration",
                "iam:PassRole",
                "glue:CreateCatalog",
                "glue:GetCatalog",
                "s3:PutBucketTagging",
                "s3:PutLifecycleConfiguration",
                "s3:PutBucketPolicy",
                "s3:CreateBucket",
                "redshift-serverless:CreateNamespace",
                "s3:DeleteBucket",
                "s3:PutBucketVersioning",
                "redshift-serverless:CreateWorkgroup"
            ],
            "Resource": [
                "arn:aws:glue:*:111122223333:catalog",
                "arn:aws:glue:*:111122223333:catalog/*",
                "arn:aws:s3:::*",
                "arn:aws:redshift-serverless:*:111122223333:workgroup/*",
                "arn:aws:redshift-serverless:*:111122223333:namespace/*",
                "arn:aws:iam::111122223333:role/GlueDataCatalogDataTransferRole"
            ]
        }
    ]
}
```

目標建立角色必須具有下列信任關係。

JSON

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

```
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        },
        {
          "Effect": "Allow",
          "Principal": {
            "AWS": "arn:aws:iam::111122223333:user/Username"
          },
          "Action": "sts:AssumeRole"
        }
    ]
}
```

Glue 資料傳輸角色

MySQL 目錄操作需要 Glue 資料傳輸角色，且必須具有下列許可。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DataTransferRolePolicy",
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt",
                "glue:GetCatalog",
                "glue:GetDatabase"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Glue 資料傳輸角色必須具有下列信任關係。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "glue.amazonaws.com",
                    "redshift.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

後續步驟

透過來源 Aurora 資料庫叢集和 Amazon Redshift 目標資料倉儲或 Amazon SageMaker Lakehouse，您可以建立零 ETL 整合並複寫資料。如需說明，請參閱the section called “建立與 Amazon Redshift 的零 ETL 整合”。

建立與 Amazon Redshift 的 Aurora 零 ETL 整合

建立 Aurora 零 ETL 整合時，您可以指定來源 Aurora 資料庫叢集和目標 Amazon Redshift 資料倉儲。您也可以自訂加密設定和新增標籤。Aurora 會在來源資料庫叢集與其目標之間建立整合。一旦整合作用中，您插入至來源資料庫叢集的任何資料就會複寫至設定的 Amazon Redshift 目標。

先決條件

在建立零 ETL 整合之前，您必須建立來源資料庫叢集和目標 Amazon Redshift 資料倉儲。您也必須將資料庫叢集新增為授權整合來源，以允許複寫至資料倉儲。

如需完成上述每個步驟的指示，請參閱 the section called “開始使用零 ETL 整合”。

所需的許可

建立零 ETL 整合需要特定 IAM 許可。您至少需要執行下列動作的許可：

- 為來源 Aurora 資料庫叢集建立零 ETL 整合。
- 檢視並刪除所有零 ETL 整合。
- 建立目標資料倉儲的傳入整合。

下列範例政策示範建立和管理整合所需的最低權限許可。如果您的使用者或角色具有更廣泛的許可，例如 `AdministratorAccess` 受管政策，則可能不需要這些確切許可。

> **Note**
>
> Redshift Amazon Resource Name (ARN) 具有下列格式。請注意在無伺服器命名空間 UUID 之前使用正斜線 (`/`)，而不是冒號 (`:`)。
>
> - 佈建的叢集 – `arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid`
> - 無伺服器 – `arn:aws:redshift-serverless:{region}:{account-id}:namespace/namespace-uuid`

Redshift 目標的範例政策

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "CreateIntegration",
      "Effect": "Allow",
      "Action": [
        "rds:CreateIntegration"
      ],
      "Resource": [
        "arn:aws:rds:us-east-1:123456789012:db:source-db",
        "arn:aws:rds:us-east-1:123456789012:integration:*"
```

```
      ]
    },
    {
      "Sid": "DescribeIntegrationDetails",
      "Effect": "Allow",
      "Action": [
        "rds:DescribeIntegrations"
      ],
      "Resource": [
      "arn:aws:rds:us-east-1:123456789012:integration:*"
  ]
    },
    {
      "Sid": "ChangeIntegrationDetails",
      "Effect": "Allow",
      "Action": [
        "rds:DeleteIntegration",
        "rds:ModifyIntegration"
      ],
      "Resource": [
        "arn:aws:rds:us-east-1:123456789012:integration:*"
      ]
    },
    {
      "Sid": "AllowRedShiftIntegration",
      "Effect": "Allow",
      "Action": [
        "redshift:CreateInboundIntegration"
      ],
      "Resource": [
        "arn:aws:redshift:us-east-1:123456789012:namespace:namespace-uuid"
      ]
    }
  ]
}
```

在不同帳戶中選擇目標資料倉儲

如果您打算指定位於另一個 中的目標 Amazon Redshift 資料倉儲 AWS 帳戶，您必須建立角色，允許目前帳戶中的使用者存取目標帳戶中的資源。如需詳細資訊，請參閱AWS 帳戶 為您擁有的另一個 IAM 使用者提供存取權。

角色必須具有下列許可，允許使用者檢視目標帳戶中可用的 Amazon Redshift 佈建叢集和 Redshift 無伺服器命名空間。

必要許可和信任政策

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "redshift:DescribeClusters",
            "redshift-serverless:ListNamespaces"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

角色必須具有下列信任策略，其可指定目標帳戶 ID。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

如需建立角色的指示，請參閱使用自訂信任政策建立角色。

建立零 ETL 整合

您可以使用 AWS 管理主控台 AWS CLI、 或 RDS API 建立零 ETL 整合。

> **Important**
>
> 零 ETL 整合不支援重新整理或重新同步操作。如果您在建立整合之後遇到問題，則必須刪除整合並建立新的整合。

RDS 主控台

建立零 ETL 整合

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在左側導覽窗格中，選擇零 ETL 整合。
3. 選擇建立零 ETL 整合。
4. 對於整合識別符，輸入整合的名稱。此名稱最多可有 63 個英數字元，且可包含連字號。

   > **Important**
   >
   > 目錄名稱的長度限制為 19 個字元。如果整合識別符將用作目錄名稱，請確保其符合此要求。

5. 選擇下一步。
6. 對於來源，請選取資料源自其中的 Aurora 資料庫叢集。

   > **Note**
   >
   > 如果未正確設定資料庫叢集參數，RDS 即會通知您。如果您收到此訊息，您可以選擇為我修正或手動設定它們。如需手動修正它們的指示，請參閱 the section called “步驟 1：建立自訂資料庫叢集參數群組。”。
   > 修改資料庫叢集參數需要重新開機。在可以建立整合之前，必須先完成重新開機，且新參數值必須成功套用至叢集。

7. (選用) 選取自訂資料篩選選項，並將資料篩選條件新增至您的整合。您可以使用資料篩選條件來定義目標資料倉儲的複寫範圍。如需詳細資訊，請參閱the section called “零 ETL 整合的資料篩選”。
8. 順利設定來源資料庫叢集後，即可選擇下一步。
9. 針對目標，執行下列動作：

   1. （選用）若要 AWS 帳戶 對 Amazon Redshift 目標使用不同的 ，請選擇指定不同的帳戶。然後，輸入 IAM 角色的 ARN，該角色擁有顯示您資料倉儲的許可。如需建立 IAM 角色的指示，請參閱 the section called “在不同帳戶中選擇目標資料倉儲”。
   2. 對於 Amazon Redshift 資料倉儲，請從來源資料庫叢集中選取所複寫資料的目標。您可以選擇佈建的 Amazon Redshift 叢集或 Redshift 無伺服器命名空間做為目標。

   > **Note**
   >
   > 如果未正確設定所指定資料倉儲的資源政策或區分大小寫設定，RDS 即會通知您。如果您收到此訊息，您可以選擇為我修正或手動設定它們。如需手動修正它們的指示，請參閱《Amazon Redshift 管理指南》中的開啟資料倉儲的區分大小寫和設定資料倉儲的授權。
   > 修改已佈建 Redshift 叢集的區分大小寫需要重新開機。在可以建立整合之前，必須先完成重新開機，且新參數值必須成功套用至叢集。
   > 如果您選取的來源和目標位於不同的 AWS 帳戶中，則 Amazon RDS 無法為您修正這些設定。您必須導覽至其他帳戶，然後在 Amazon Redshift 中手動修正這些設定。

10. 正確設定了目標資料倉儲後，即可選擇下一步。
11. (選用) 對於標籤，將一或多個標籤新增至整合。如需詳細資訊，請參閱the section called “標記 Aurora 和 RDS 資源”。
12. 對於加密，指定您想要加密整合的方式。根據預設，RDS 會加密與 的所有整合 AWS 擁有的金鑰。若要改為選擇客戶受管金鑰，請啟用自訂加密設定，然後選擇要用於加密的 KMS 金鑰。如需詳細資訊，請參閱the section called “加密 Amazon Aurora 資源”。

    可選擇性地新增加密內容。如需詳細資訊，請參閱 AWS Key Management Service 開發人員指南中的加密內容。

    > **Note**
    >
    > 除了您新增的任何項目之外，Amazon RDS 還會新增下列加密內容對：
    >
    > - `aws:redshift:integration:arn` - `IntegrationArn`

- `aws:servicename:id - Redshift`

這會將您可以新增的整體配對數量從 8 減少為 6，並對授予限制的整體字元限制造成影響。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的使用授予限制條件。

13. 選擇下一步。
14. 檢閱您的整合設定，然後選擇建立零 ETL 整合。

    如果建立失敗，請參閱 the section called “我無法建立零 ETL 整合”，取得疑難排解步驟。

整合建立時的狀態為 `Creating`，而目標 Amazon Redshift 資料倉儲的狀態為 `Modifying`。在此期間，您無法查詢資料倉儲或對其進行任何組態變更。

成功建立整合時，整合和目標 Amazon Redshift 資料倉儲的狀態都會變更為 `Active`。

AWS CLI

若要使用 建立零 ETL 整合 AWS CLI，請使用 create-integration 命令搭配下列選項：

> **Note**
>
> 請記住，目錄名稱限制為 19 個字元。如果整合名稱將用作目錄名稱，請據此選擇整合名稱。

- `--integration-name` - 指定整合的名稱。
- `--source-arn` - 指定 Aurora 資料庫叢集的 ARN，它將會成為整合來源。
- `--target-arn` – 指定 Amazon Redshift 資料倉儲的 ARN，它將會成為整合目標。

Example

針對 Linux、macOS 或 Unix：

```
aws rds create-integration \
    --integration-name my-integration \
    --source-arn arn:aws:rds:{region}:{account-id}:my-db \
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

在 Windows 中：

```
aws rds create-integration ^
    --integration-name my-integration ^
    --source-arn arn:aws:rds:{region}:{account-id}:my-db ^
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

RDS API

若要使用 Amazon RDS API 建立零 ETL 整合，請搭配下列參數使用 CreateIntegration 操作：

> **Note**
>
> 目錄名稱限制為 19 個字元。如果 IntegrationName 參數將用作目錄名稱，請確保其符合此要求。

- `IntegrationName` - 指定整合的名稱。
- `SourceArn` - 指定 Aurora 資料庫叢集的 ARN，它將會成為整合來源。
- `TargetArn` – 指定 Amazon Redshift 資料倉儲的 ARN，它將會成為整合目標。

使用客戶自管金鑰加密整合

如果您在建立整合 AWS 擁有的金鑰 時指定自訂 KMS 金鑰，而不是 ，則金鑰政策必須提供 Amazon Redshift 服務主體對 `CreateGrant`動作的存取權。此外，它必須允許目前的使用者執行 `DescribeKey` 和 `CreateGrant` 動作。

下列範例政策示範如何在金鑰政策中提供必要許可。它包含內容索引鍵，以進一步減少許可範圍。

範例金鑰政策

JSON

```
{
    "Version":"2012-10-17",
    "Id": "Key policy",
    "Statement": [
        {
            "Sid": "Enables IAM user permissions",
            "Effect": "Allow",
```

```
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allows the Redshift service principal to add a grant to a KMS
key",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}": "{context-value}"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "Allows the current user or role to add a grant to a KMS key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/{role-name}"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}": "{context-value}",
                    "kms:ViaService": "rds.us-east-1.amazonaws.com"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
```

```
                    "GenerateDataKey",
                    "CreateGrant"
                ]
            }
        }
    },
    {
        "Sid": "Allows the current uer or role to retrieve information about
 a KMS key",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::111122223333:role/{role-name}"
        },
        "Action": "kms:DescribeKey",
        "Resource": "*"
    }
  ]
}
```

如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的建立金鑰政策。

後續步驟

在成功建立零 ETL 整合之後，您必須在目標 Amazon Redshift 叢集或工作群組內建立目的地資料庫。然後，您就可以開始將資料新增到來源 Aurora 資料庫叢集，並在 Amazon Redshift 中進行查詢。如需指示，請參閱在 Amazon Redshift 中建立目的地資料庫。

建立與 Amazon SageMaker 資料湖倉的 Aurora 零 ETL 整合

當您建立與 Lakehouse 的 Aurora零 ETL Amazon SageMaker 整合時，您可以指定來源 Aurora 資料庫叢集和目標 AWS Glue 受管目錄。您也可以自訂加密設定和新增標籤。Aurora 會在來源資料庫叢集與其目標之間建立整合。一旦整合作用中，您插入至來源資料庫叢集的任何資料就會複寫至設定的目標。

先決條件

在建立與 Lakehouse Amazon SageMaker 的零 ETL 整合之前，您必須建立來源資料庫叢集和目標 AWS Glue 受管目錄。您也必須將資料庫叢集新增為授權整合來源，以允許複寫至目錄。

如需完成上述每個步驟的指示，請參閱 the section called “開始使用零 ETL 整合”。

所需的許可

建立與 Amazon SageMaker 資料湖倉的零 ETL 整合需要特定 IAM 許可。您至少需要執行下列動作的許可：

- 為來源 Aurora 資料庫叢集建立零 ETL 整合。
- 檢視並刪除所有零 ETL 整合。
- 在目標 AWS Glue 受管目錄中建立傳入整合。
- 存取 AWS Glue 受管目錄使用的 Amazon S3 儲存貯體。
- 如果已設定自訂加密，請使用 AWS KMS 金鑰進行加密。
- 向 Lake Formation 註冊資源。
- 在 AWS Glue 受管目錄上放置資源政策，以授權傳入整合。

下列範例政策示範建立和管理與 Amazon SageMaker 資料湖倉的整合所需的最低權限許可。如果您的使用者或角色具有更廣泛的許可，例如 `AdministratorAccess` 受管政策，則可能不需要這些確切許可。

此外，您必須在目標 AWS Glue 受管目錄上設定資源政策，以授權傳入整合。使用下列 AWS CLI 命令來套用資源政策。

用於授權目標目錄上傳入整合的範例 AWS CLI 命令

```
aws glue put-resource-policy \
      --policy-in-json  '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "glue.amazonaws.com"
        },
        "Action": [
            "glue:AuthorizeInboundIntegration"
        ],
        "Resource": ["arn:aws:glue:region:account_id:catalog/catalog_name"],
        "Condition": {
            "StringEquals": {
                "aws:SourceArn": "arn:aws:rds:region:account_id:db:source_name"
            }
```

```
        }
    },
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": "account_id"
        },
        "Action": ["glue:CreateInboundIntegration"],
        "Resource": ["arn:aws:glue:region:account_id:catalog/catalog_name"]
    }
    ]
}' \
      --region region
```

> **Note**
>
> Glue 目錄 Amazon Resource Name (ARN) 具有下列格式：
>
> - Glue 目錄 – `arn:aws:glue:{region}:{account-id}:catalog/catalog-name`

在不同的帳戶中選擇目標 AWS Glue 受管目錄

如果您打算指定位於另一個 的目標 AWS Glue 受管目錄 AWS 帳戶，您必須建立角色，允許目前帳戶中的使用者存取目標帳戶中的資源。如需詳細資訊，請參閱AWS 帳戶 為您擁有的另一個 IAM 使用者提供存取權。

角色必須具有下列許可，允許使用者檢視目標帳戶中可用的 AWS Glue 目錄。

必要許可和信任政策

JSON

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

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

角色必須具有下列信任策略，其可指定目標帳戶 ID。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "AWS": "arn:aws:iam::111122223333:root"
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

如需建立角色的指示，請參閱使用自訂信任政策建立角色。

建立與 Amazon SageMaker 資料湖倉的零 ETL 整合

您可以使用 AWS 管理主控台 AWS CLI、 或 RDS API 來建立與 Lakehouse Amazon SageMaker 的零 ETL 整合。

> **Important**
>
> 與 Amazon SageMaker 資料湖倉的零 ETL 整合不支援重新整理或重新同步操作。如果您在建立整合之後遇到問題，則必須刪除整合並建立新的整合。

RDS 主控台

建立與 Amazon SageMaker 資料湖倉的零 ETL 整合

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在左側導覽窗格中，選擇零 ETL 整合。
3. 選擇建立零 ETL 整合。
4. 對於整合識別符，輸入整合的名稱。此名稱最多可有 63 個英數字元，且可包含連字號。
5. 選擇下一步。
6. 對於來源，請選取資料源自其中的 Aurora 資料庫叢集。

> **Note**
>
> 如果未正確設定資料庫叢集參數，RDS 即會通知您。如果您收到此訊息，您可以選擇為我修正或手動設定它們。如需手動修正它們的指示，請參閱 the section called “步驟 1：建立自訂資料庫叢集參數群組。”。
> 修改資料庫叢集參數需要重新開機。在可以建立整合之前，必須先完成重新開機，且新參數值必須成功套用至叢集。

7. (選用) 選取自訂資料篩選選項，並將資料篩選條件新增至您的整合。您可以使用資料篩選條件來定義目標 Amazon SageMaker 資料湖倉的複寫範圍。如需詳細資訊，請參閱the section called “零 ETL 整合的資料篩選”。
8. 順利設定來源資料庫叢集後，即可選擇下一步。
9. 針對目標，執行下列動作：
   1. （選用） 若要 AWS 帳戶 對湖房目標使用不同的 Amazon SageMaker ，請選擇指定不同的帳戶。然後，輸入具有顯示 AWS Glue 目錄許可的 IAM 角色的 ARN。如需建立 IAM 角色的指示，請參閱 the section called “在不同的帳戶中選擇目標 AWS Glue 受管目錄”。
   2. 對於 AWS Glue 目錄，從來源資料庫叢集選取複寫資料的目標。您可以選擇現有的 AWS Glue 受管目錄做為目標。
   3. 目標 IAM 角色需要描述目標目錄的許可，並且必須具有下列許可：

      JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "glue:GetCatalog",
            "Resource": [
                "arn:aws:glue:us-east-1:111122223333:catalog/*",
                "arn:aws:glue:us-east-1:111122223333:catalog"
            ]
        }
    ]
}
```

目標 IAM 角色必須具有下列信任關係：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

4. 您必須使用在 中建立的 Lake Formation 管理員角色，授予目標 IAM 角色描述目標 AWS Glue 受管目錄的許可步驟 3b：建立 Amazon SageMaker Lakehouse 零 ETL 整合的 AWS Glue 目錄。

> **Note**
>
> 如果指定的 AWS Glue 受管目錄的資源政策或組態設定未正確設定，RDS 會通知您。如果您收到此訊息，您可以選擇為我修正或手動設定它們。
> 如果您選取的來源和目標位於不同的 AWS 帳戶中，則 Amazon RDS 無法為您修正這些設定。您必須導覽至其他帳戶，然後在 SageMaker Unified Studio 中手動修正這些設定。

10. 正確設定目標 AWS Glue 受管目錄後，請選擇下一步。
11. (選用) 對於標籤，將一或多個標籤新增至整合。如需詳細資訊，請參閱the section called “標記 Aurora 和 RDS 資源”。
12. 對於加密，指定您想要加密整合的方式。根據預設，RDS 會加密與 的所有整合 AWS 擁有的金鑰。若要改為選擇客戶受管金鑰，請啟用自訂加密設定，然後選擇要用於加密的 KMS 金鑰。如需詳細資訊，請參閱the section called “加密 Amazon Aurora 資源”。

    可選擇性地新增加密內容。如需詳細資訊，請參閱 AWS Key Management Service 開發人員指南中的加密內容。

    > **Note**
    >
    > 除了您新增的任何項目之外，Amazon RDS 還會新增下列加密內容對：
    >
    > - `aws:glue:integration:arn` - `IntegrationArn`
    > - `aws:servicename:id` - `glue`
    >
    > 這會將您可以新增的整體配對數量從 8 減少為 6，並對授予限制的整體字元限制造成影響。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的使用授予限制條件。

13. 選擇下一步。
14. 檢閱您的整合設定，然後選擇建立零 ETL 整合。

    如果建立失敗，請參閱 the section called “對零 ETL 整合進行疑難排解”，取得疑難排解步驟。

整合建立時的狀態為 `Creating`，而目標 Amazon SageMaker 資料湖倉的狀態為 `Modifying`。在此期間，您無法查詢目錄或對其進行任何組態變更。

成功建立整合時，整合和目標 Amazon SageMaker 資料湖倉的狀態都會變更為 `Active`。

AWS CLI

若要使用 為零 ETL 整合準備目標 AWS Glue 受管目錄 AWS CLI，您必須先使用 create-integration-resource-property 命令搭配下列選項：

- `--resource-arn` – 指定將成為整合目標之 AWS Glue 受管目錄的 ARN。
- `--target-processing-properties` – 指定 IAM 角色的 ARN 以存取目標 AWS Glue 受管目錄

```
aws glue create-integration-resource-property --region us-east-1
 --resource-arn arn:aws:glue:region:account_id:catalog/catalog_name \
 --target-processing-properties '{"RoleArn" : "arn:aws:iam::account_id:role/
TargetIamRole"}'
```

若要使用 建立與 Lakehouse Amazon SageMaker 的零 ETL 整合 AWS CLI，請使用 create-integration 命令搭配下列選項：

- `--integration-name` - 指定整合的名稱。
- `--source-arn` - 指定 Aurora 資料庫叢集的 ARN，它將會成為整合來源。
- `--target-arn` – 指定將成為整合目標之 AWS Glue 受管目錄的 ARN。

Example

針對 Linux、macOS 或 Unix：

```
aws rds create-integration \
    --integration-name my-sagemaker-integration \
    --source-arn arn:aws:rds:{region}:{account-id}:cluster:my-db \
    --target-arn arn:aws:glue:{region}:{account-id}:catalog/catalog-name
```

在 Windows 中：

```
aws rds create-integration ^
    --integration-name my-sagemaker-integration ^
    --source-arn arn:aws:rds:{region}:{account-id}:cluster:my-db ^
    --target-arn arn:aws:glue:{region}:{account-id}:catalog/catalog-name
```

RDS API

若要使用 Amazon RDS API 建立與 Amazon SageMaker 的零 ETL 整合，請搭配下列參數使用 `CreateIntegration` 操作：

> **Note**
>
> 目錄名稱限制為 19 個字元。如果 IntegrationName 參數將用作目錄名稱，請確保其符合此要求。

- IntegrationName - 指定整合的名稱。
- SourceArn - 指定 Aurora 資料庫叢集的 ARN，它將會成為整合來源。
- TargetArn – 指定將成為整合目標之 AWS Glue 受管目錄的 ARN。

使用客戶自管金鑰加密整合

如果您在建立與 的整合 AWS 擁有的金鑰 時指定自訂 KMS 金鑰，而不是 Amazon SageMaker，則金鑰政策必須提供 CreateGrant動作SageMaker Unified Studio的服務主體存取權。此外，它必須允許目前的使用者執行 DescribeKey 和 CreateGrant 動作。

下列範例政策示範如何在金鑰政策中提供必要許可。它包含內容索引鍵，以進一步減少許可範圍。

範例金鑰政策

JSON

```
{
    "Version":"2012-10-17",
    "Id": "Key policy",
    "Statement": [
        {
            "Sid": "EnablesIAMUserPermissions",
            "Effect": "Allow",
            "Principal": {
            "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "GlueServicePrincipalAddGrant",
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}"
                },
```

```
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "AllowsCurrentUserRoleAddGrantKMSKey",
            "Effect": "Allow",
            "Principal": {
            "AWS": "arn:aws:iam::111122223333:role/{role-name}"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}",
                    "kms:ViaService": "rds.us-east-1.amazonaws.com"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "AllowsCurrentUserRoleRetrieveKMSKeyInformation",
            "Effect": "Allow",
            "Principal": {
            "AWS": "arn:aws:iam::111122223333:role/{role-name}"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*"
        }
    ]
}
```

如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的建立金鑰政策。

後續步驟

成功建立與 Amazon SageMaker 的零 ETL 整合後，您可以開始將資料新增至來源 Aurora 資料庫叢集，並在 Amazon SageMaker 資料湖倉中查詢資料。資料會自動複寫，並可用於分析和機器學習工作負載。

Aurora 零 ETL 整合的資料篩選

Aurora 零 ETL 整合支援資料篩選，可讓您控制要將哪些資料從來源 Aurora 資料庫叢集複寫到目標資料倉儲。您可以套用一或多個篩選條件來選擇性地包含或排除特定資料表，而不是複寫整個資料庫。這可協助您透過確保僅傳輸相關資料來最佳化儲存和查詢效能。目前，篩選僅限於資料庫和資料表層級。不支援資料欄層級和資料列層級篩選。

當您想要執行下列操作時，資料篩選很有用：

- 從兩個或多個不同的來源叢集聯結特定資料表，且您不需要任何一個叢集的完整資料。
- 僅使用資料表子集而非整個資料庫機群執行分析，以節省成本。
- 從特定資料表篩選掉敏感資訊，例如電話號碼、地址或信用卡詳細資訊。

您可以使用 AWS 管理主控台、 AWS Command Line Interface (AWS CLI) 或 Amazon RDS API，將資料篩選條件新增至零 ETL 整合。

如果整合具有佈建叢集做為其目標，則叢集必須在修補程式 180 或更高版本，才能使用資料篩選。

主題

- 資料篩選的格式
- 篩選條件邏輯
- 篩選條件優先順序
- Aurora MySQL 範例
- Aurora PostgreSQL 範例
- 將資料篩選條件新增至整合
- 從整合中移除資料篩選條件

資料篩選的格式

您可以為單一整合定義多個篩選條件。每個篩選條件都會包含或排除符合篩選條件運算式中其中一個模式的任何現有和未來資料庫資料表。Aurora 零 ETL 整合使用 Maxwell 篩選條件語法進行資料篩選。

每個篩選條件都有下列元素：

Element	Description
篩選條件類型	`Include` 篩選條件類型包含符合篩選條件運算式中其中一個模式的所有資料表。`Exclude` 篩選條件類型排除符合其中一個模式的所有資料表。
篩選條件表達式	模式的逗號分隔清單。運算式必須使用 Maxwell 篩選條件語法。
模式	Aurora MySQL 的 *`database`*`.`*`table`* 格式篩選條件模式，或 Aurora PostgreSQL 的 *`database`*`.`*`schema`*`.`*`table`* 格式篩選條件模式。您可以指定常值名稱，或定義規則運算式。 **Note** 對於 Aurora MySQL，資料庫和資料表名稱都支援規則運算式。對於 Aurora PostgreSQL，僅結構描述和資料表名稱支援規則運算式，資料庫名稱不支援。 您無法包含資料欄層級篩選條件或拒絕清單。 單一整合最多可有 99 個模式。在主控台中，您可以在單一篩選條件運算式內輸入模式，或將其分散在多個運算式中。單一模式的長度不可超過 256 個字元。

> **Important**
>
> 如果您選取 Aurora PostgreSQL 來源資料庫叢集，則必須至少指定一個資料篩選條件模式。模式至少必須包含單一資料庫 (*database-name*.*.*)，才能複寫至目標資料倉儲。

下圖顯示主控台中 Aurora MySQL 資料篩選條件的結構：



> **Important**
>
> 請勿在您的篩選條件模式中包含個人身分識別、機密或敏感資訊。

中的資料篩選條件 AWS CLI

使用 AWS CLI 新增資料篩選條件時，語法與主控台略有不同。您必須個別為每個模式指派篩選條件類型 (`Include` 或 `Exclude`)，因此您無法在一個篩選條件類型下群組多個模式。

例如，在主控台中，您可以在單一 `Include` 陳述式下群組下列逗號分隔模式：

Aurora MySQL

```
mydb.mytable, mydb./table_\d+/
```

Aurora PostgreSQL

```
mydb.myschema.mytable, mydb.myschema./table_\d+/
```

不過，使用 時 AWS CLI，相同的資料篩選條件必須採用下列格式：

Aurora MySQL

```
'include: mydb.mytable, include: mydb./table_\d+/'
```

Aurora PostgreSQL

```
'include: mydb.myschema.mytable, include: mydb.myschema./table_\d+/'
```

篩選條件邏輯

如果您未在整合中指定任何資料篩選條件，Aurora 會假設 `include:*.*` 的預設篩選條件，這會將所有資料表複寫到目標資料倉儲。不過，如果您新增至少一個篩選條件，預設邏輯會切換為 `exclude:*.*`，預設會排除所有資料表。這可讓您明確定義複寫中要包含哪些資料庫和資料表。

例如，如果您定義下列篩選條件：

```
'include: db.table1, include: db.table2'
```

Aurora 會評估篩選條件，如下所示：

```
'exclude:*.*, include: db.table1, include: db.table2'
```

因此，Aurora 只會從名為 db 的資料庫將 `table1` 和 `table2` 複寫到目標資料倉儲。

篩選條件優先順序

Aurora 會依您指定的順序評估資料篩選條件。在 中 AWS 管理主控台，它會處理從左到右以及從上到下的篩選條件表達式。第二個篩選條件或第一個篩選條件後面的個別模式可以覆寫它。

例如，如果第一個篩選條件是 `Include books.stephenking`，它只會包含 `books` 資料庫中的 `stephenking` 資料表。不過，如果您新增第二個篩選條件 `Exclude books.*`，則會覆寫第一個篩選條件。這可防止 `books` 索引中的任何資料表複寫到目標資料倉儲。

當您指定至少一個篩選條件時，邏輯會從預設假設 `exclude:*.*` 開始，這會自動排除所有資料表進行複寫。最佳實務是定義從最廣泛到最具體的篩選條件。從一或多個 `Include` 陳述式開始，指定要複寫的資料，然後新增 `Exclude` 篩選條件以選擇性地移除特定資料表。

相同的原則適用於您使用 AWS CLI定義的篩選條件。Aurora 會依您指定的順序評估這些篩選條件模式，因此模式可能會覆寫您在之前指定的模式。

Aurora MySQL 範例

下列範例示範資料篩選如何適用於 Aurora MySQL 範例零 ETL 整合：

- 包含所有資料庫和所有資料表：

```
'include: *.*'
```

- 包含 books 資料庫中的所有資料表：

```
'include: books.*'
```

- 排除任何名為 mystery 的資料表：

```
'include: *.*, exclude: *.mystery'
```

- 包含 books 資料庫中的兩個特定資料表：

```
'include: books.stephen_king, include: books.carolyn_keene'
```

- 包含 books 資料庫中的所有資料表，但包含子字串 mystery 的資料表除外：

```
'include: books.*, exclude: books./.*mystery.*/'
```

- 包含 books 資料庫中的所有資料表，但以 mystery 開頭的資料表除外：

```
'include: books.*, exclude: books./mystery.*/'
```

- 包含 books 資料庫中的所有資料表，但以 mystery 結尾的資料表除外：

```
'include: books.*, exclude: books./.*mystery/'
```

- 包含 books 資料庫中以 table_ 開頭的所有資料表，但名為 table_stephen_king 的資料表除外。例如，會複寫 table_movies 或 table_books，但不會複寫 table_stephen_king。

```
'include: books./table_.*/, exclude: books.table_stephen_king'
```

Aurora PostgreSQL 範例

下列範例示範資料篩選如何適用於 Aurora PostgreSQL 零 ETL 整合：

- 包含 books 資料庫中的所有資料表：

```
'include: books.*.*'
```

- 排除 books 資料庫中名為 mystery 的任何資料表：

```
'include: books.*.*, exclude: books.*.mystery'
```

- 包含 books 資料庫的 mystery 結構描述中的一個資料表，以及 employee 資料庫的 finance 結構描述的一個資料表：

```
'include: books.mystery.stephen_king, include: employee.finance.benefits'
```

- 包含 books 資料庫的 science_fiction 結構描述中的所有資料表，但包含子字串 king 的資料表除外：

```
'include: books.science_fiction.*, exclude: books.*./.*king.*/
```

- 包含 books 資料庫中的所有資料表，但具有以 sci 開頭的結構描述名稱的資料表除外：

```
'include: books.*.*, exclude: books./sci.*/.*'
```

- 包含 books 資料庫中的所有資料表，但以 king 結尾的 mystery 結構描述中的資料表除外：

```
'include: books.*.*, exclude: books.mystery./.*king/'
```

- 包含 books 資料庫中以 table_ 開頭的所有資料表，但名為 table_stephen_king 的資料表除外。例如，會複寫 fiction 結構描述中的 table_movies 和 mystery 結構描述中的 table_books，但不會複寫任一結構描述中的 table_stephen_king：

```
'include: books.*./table_.*/, exclude: books.*.table_stephen_king'
```

將資料篩選條件新增至整合

您可以使用 AWS 管理主控台、 AWS CLI或 Amazon RDS API 設定資料篩選。

> **Important**
>
> 如果您在建立整合之後新增篩選條件，Aurora 會將其視為一律存在。它會移除目標資料倉儲中不符合新篩選條件的任何資料，並重新同步所有受影響的資料表。

RDS 主控台

將資料篩選條件新增至零 ETL 整合

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇零 ETL 整合。選取您要新增資料篩選條件的整合，然後選擇修改。
3. 在來源下，新增一或多個 `Include` 和 `Exclude` 陳述式。

   下圖顯示 MySQL 整合的資料篩選條件範例：



4. 當您對變更感到滿意時，請選擇繼續和儲存變更。

AWS CLI

若要使用 將資料篩選條件新增至零 ETL 整合 AWS CLI，請呼叫 modify-integration 命令。除了整合識別符之外，使用逗號分隔的 `Include` 和 `Exclude` Maxwell 篩選條件清單指定 `--data-filter` 參數。

Example

下列範例會將篩選條件模式新增至 `my-integration`。

針對 Linux、macOS 或 Unix：

```
aws rds modify-integration \
    --integration-identifier my-integration \
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

在 Windows 中：

```
aws rds modify-integration ^
    --integration-identifier my-integration ^
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

RDS API

若要使用 RDS API 修改零 ETL 整合，請呼叫 ModifyIntegration 操作。指定整合識別符，並提供以逗號分隔的篩選條件模式清單。

從整合中移除資料篩選條件

當您從整合中移除資料篩選條件時，Aurora 會重新評估剩餘的篩選條件，就好像移除的篩選條件從未存在一樣。然後，它會將目前符合條件的任何先前排除的資料複寫到目標資料倉儲。這會觸發所有受影響資料表的重新同步。

將資料新增至來源 Aurora 資料庫叢集並進行查詢

若要完成建立 零 ETL 整合，將資料從 Amazon Aurora 複製到 Amazon Redshift，您必須在目標目的地中建立資料庫。

對於與 Amazon Redshift 的連線，連線到您的 Amazon Redshift 叢集或工作群組，並建立參考整合識別符的資料庫。然後，您可以將資料新增到來源 Aurora 資料庫叢集，並看到該資料在 Amazon Redshift 或 Amazon SageMaker 中複寫。

主題

- 建立目標資料庫
- 將資料新增至來源資料庫叢集
- 在 Amazon Redshift 中查詢您的 Aurora 資料。
- Aurora 與 Amazon Redshift 資料庫之間的資料類型差異
- Aurora PostgreSQL 的 DDL 操作

建立目標資料庫

在建立整合之後，您必須在目標資料倉儲中建立資料庫，然後才能開始將資料複寫到 Amazon Redshift。此資料庫必須包含整合識別符的參考。您可以使用 Amazon Redshift 主控台或查詢編輯器第 2 版來建立資料庫。

如需建立目的地資料庫的指示，請參閱在 Amazon Redshift 中建立目的地資料庫。

將資料新增至來源資料庫叢集

設定整合之後，您可以將要複寫至資料倉儲的資料填入來源 Aurora 資料庫叢集。

> **Note**
>
> Amazon Aurora 與目標分析倉儲中的資料類型之間存在差異。如需資料類型映射的資料表，請參閱 the section called “資料類型差異”。

首先，使用您選擇的 MySQL 或 PostgreSQL 用戶端連線到來源資料庫叢集。如需說明，請參閱the section called “連接至資料庫叢集”。

然後，建立資料表並插入一列範例資料。

> **Important**
>
> 請確定資料表具有主索引鍵。否則，無法將其複寫到目標資料倉儲。

pg_dump 和 pg_restore PostgreSQL 公用程式一開始會在沒有主索引鍵的情況下建立資料表，之後再新增主索引鍵。如果您使用的是其中一個公用程式，建議您先建立結構描述，然後在單獨的命令中載入資料。

MySQL

下列範例使用 MySQL Workbench 公用程式。

```
CREATE DATABASE my_db;

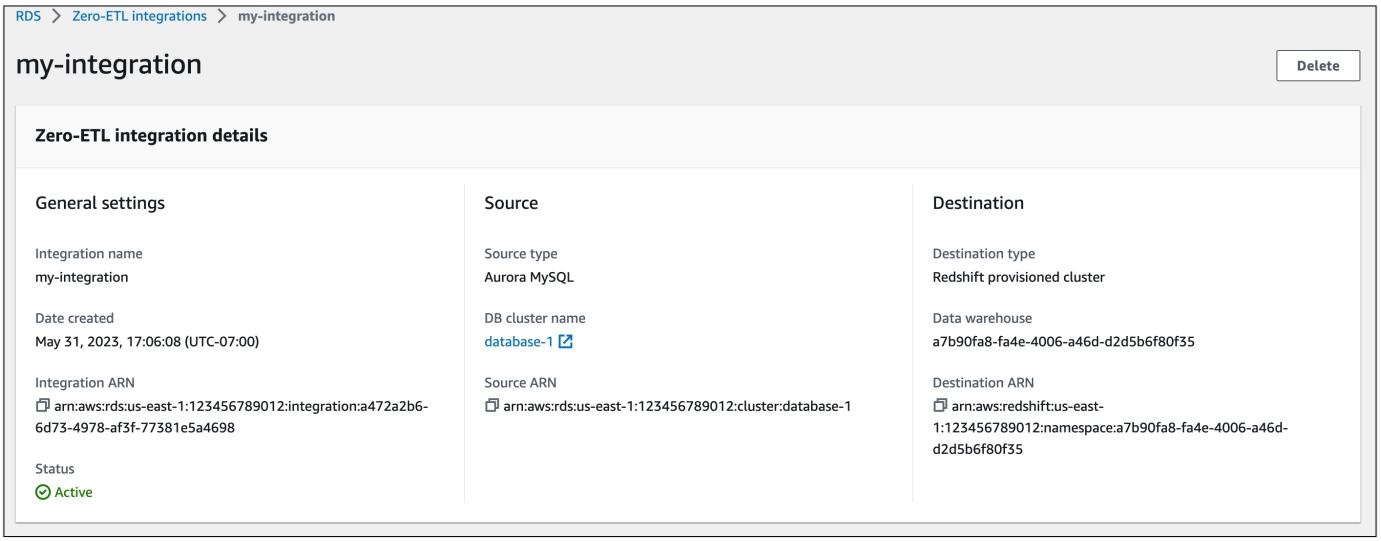
USE my_db;

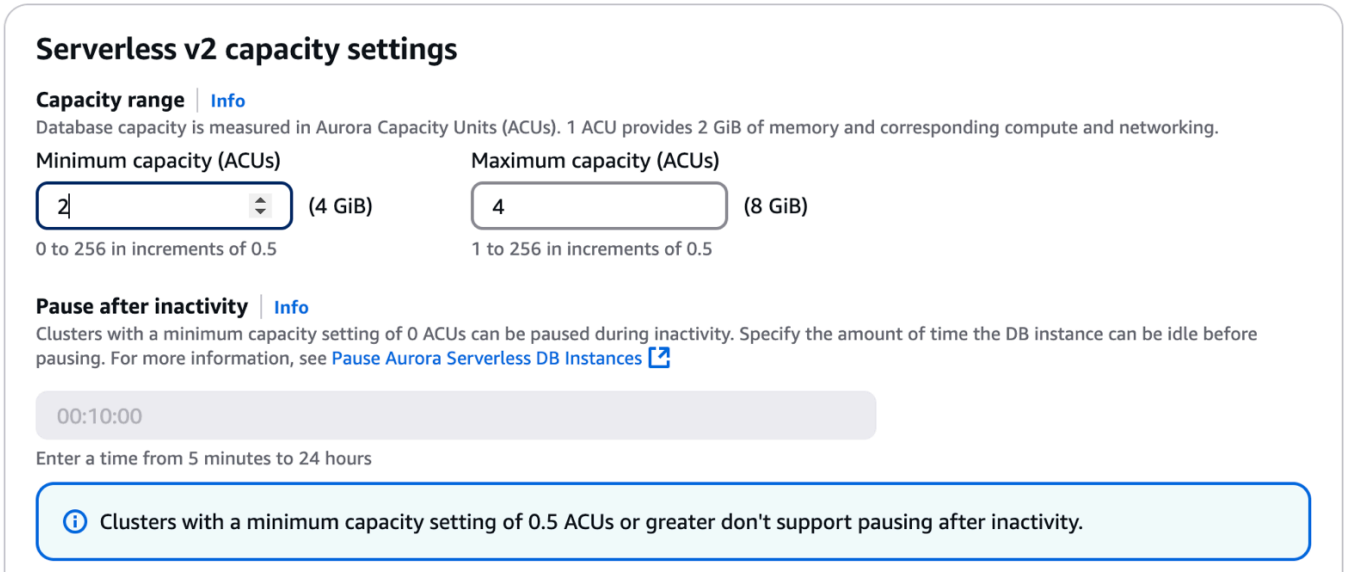
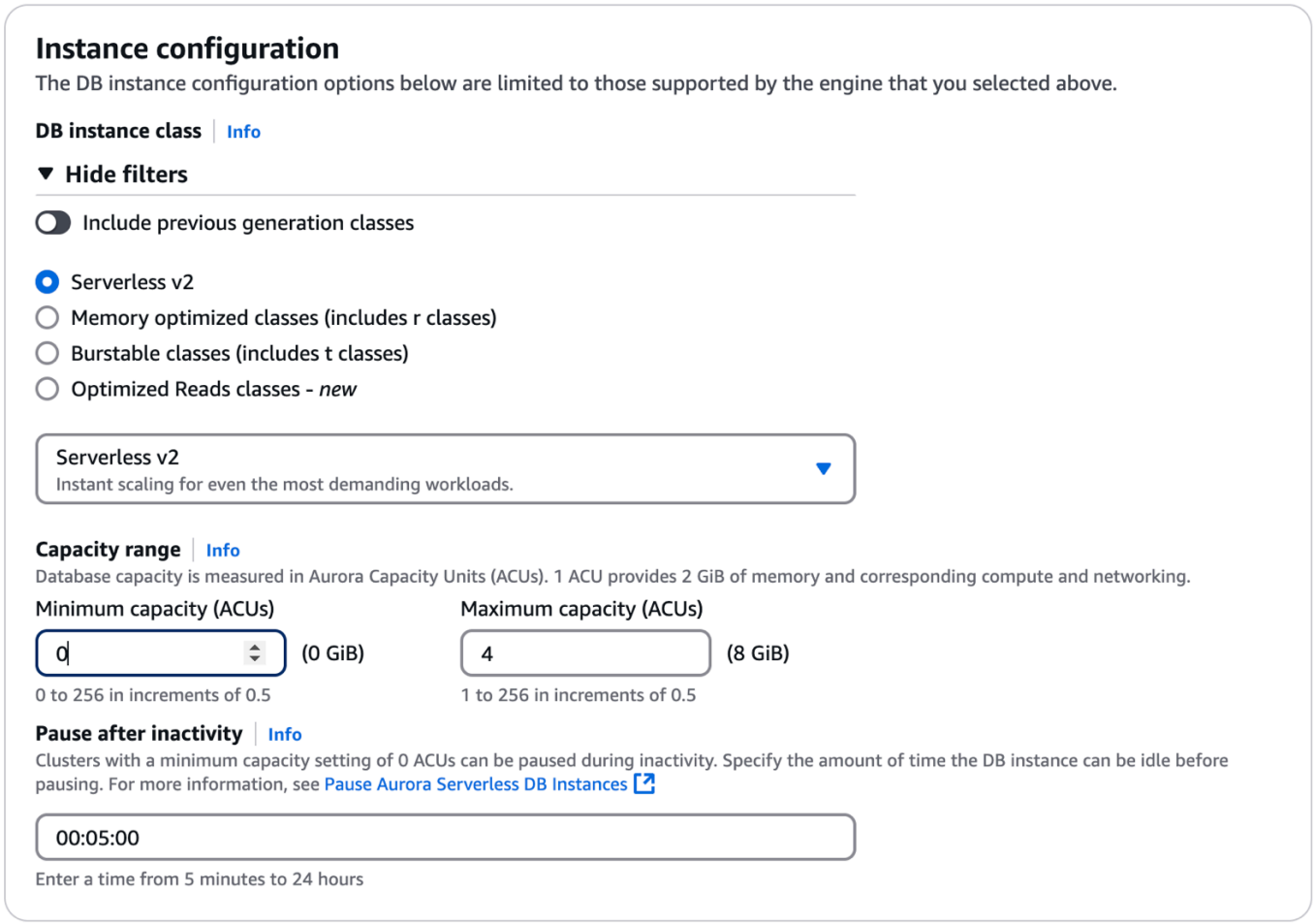
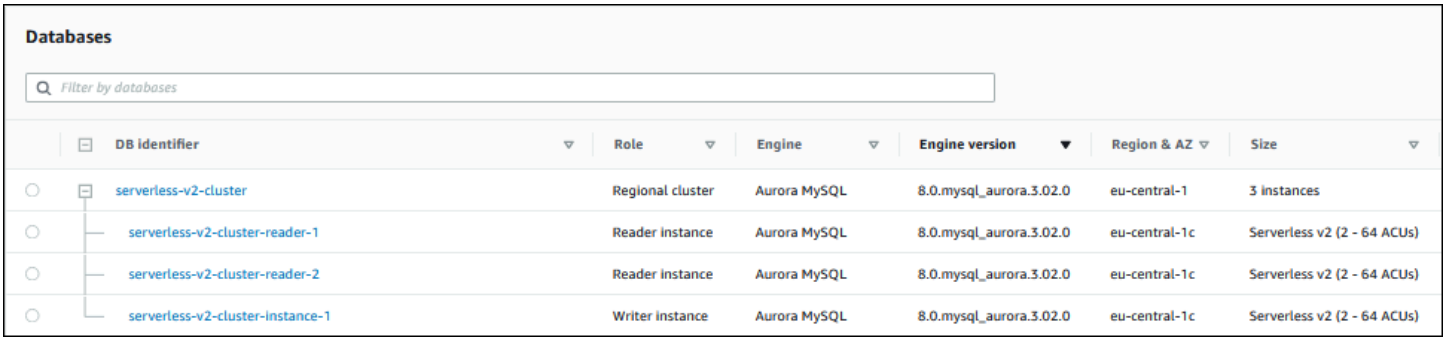
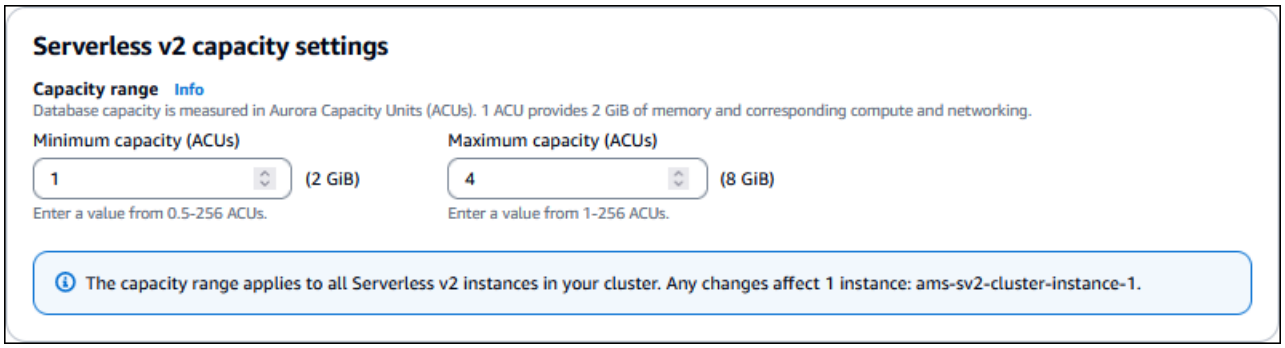
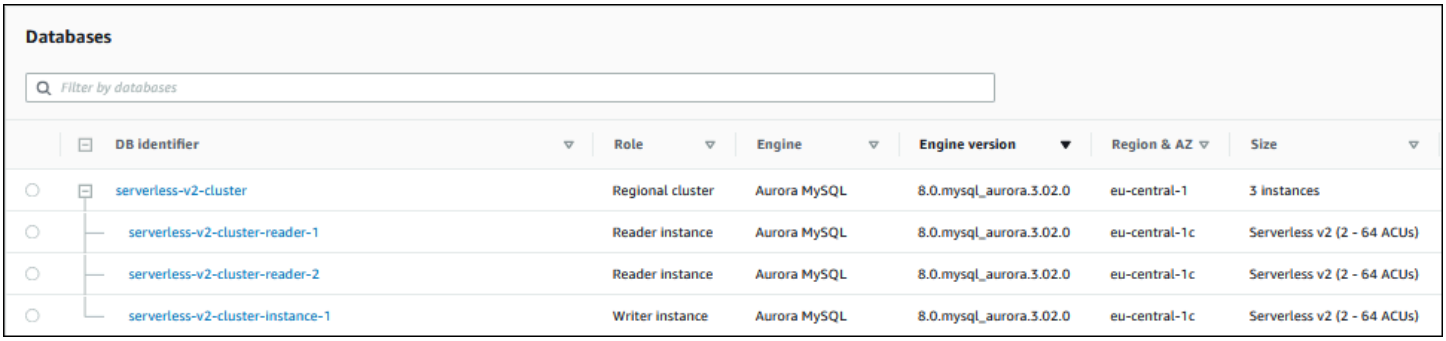
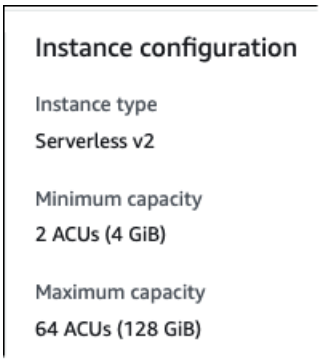
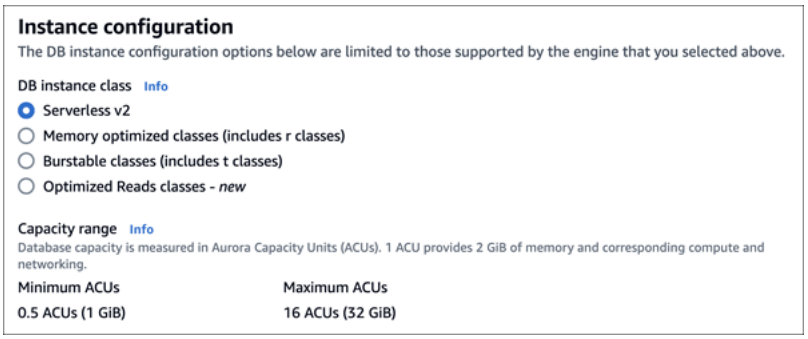
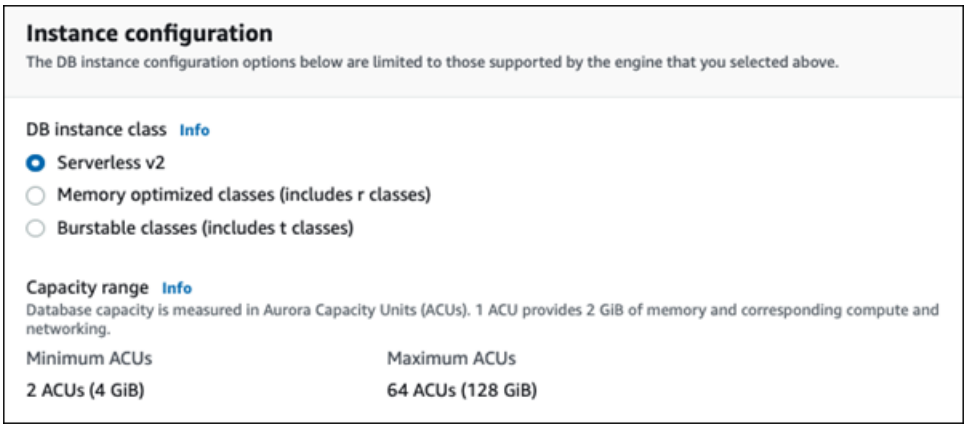
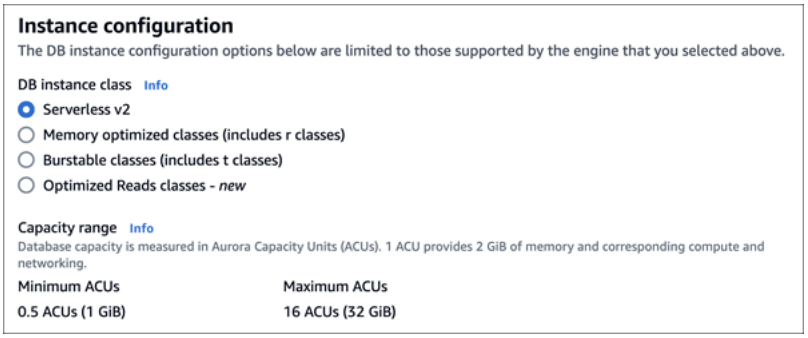
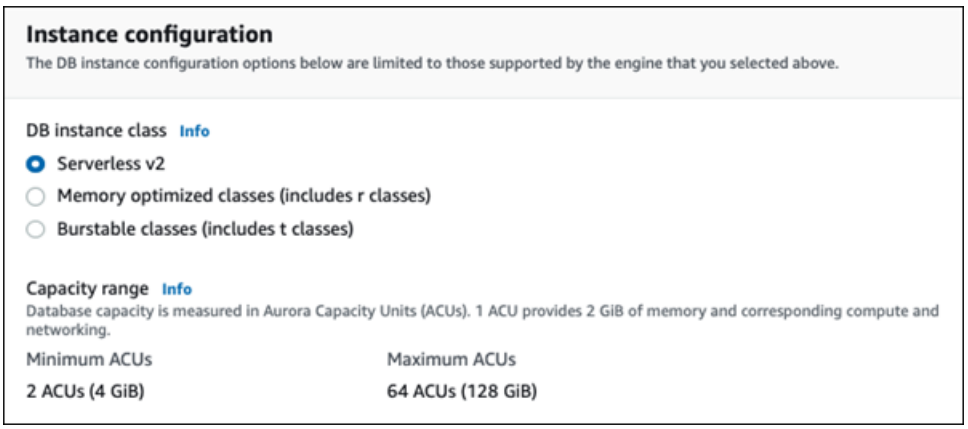
CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL, Author
 VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

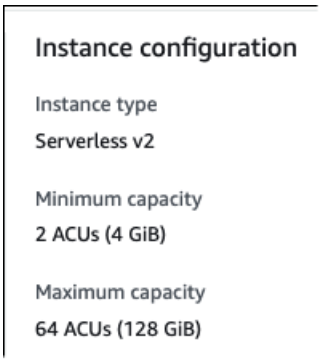
INSERT INTO books_table VALUES (1, 'The Shining', 'Stephen King', 1977, 'Supernatural
 fiction');
```

PostgreSQL

下列範例使用 psql PostgreSQL 互動式終端機。連線至叢集時，請包含您在建立整合時指定的具名資料庫。

```
psql -h mycluster.cluster-123456789012.us-east-2.rds.amazonaws.com -p 5432 -U username
 -d named_db;

named_db=> CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL,
 Author VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

named_db=> INSERT INTO books_table VALUES (1, 'The Shining', 'Stephen King', 1977,
 'Supernatural fiction');
```

在 Amazon Redshift 中查詢您的 Aurora 資料。

將資料新增至 Aurora 資料庫叢集後，資料會複寫至目的地資料庫，並準備好進行查詢。

查詢複製的資料

1. 導覽至 Amazon Redshift 主控台，然後從左側導覽窗格中選擇查詢編輯器第 2 版。
2. 連線到您的叢集或工作群組，然後從下拉式功能表中選擇您已從整合中建立的目的地資料庫 (在此範例中為 destination_database)。如需建立目的地資料庫的指示，請參閱在 Amazon Redshift 中建立目的地資料庫。

3. 使用 SELECT 陳述式查詢您的資料。在此範例中，您可以執行下列命令，從您在來源 Aurora 資料庫叢集中建立的資料表中選取所有資料：

```
SELECT * from my_db."books_table";
```



- *my_db* 是 Aurora 資料庫結構描述名稱。只有 MySQL 資料庫需要此選項。
- *books_table* 是 Aurora 資料表名稱。

您也可以使用命令列用戶端查詢資料。例如：

```
destination_database=# select * from my_db."books_table";

 ID |       Title |        Author |   Copyright |                  Genre |  txn_seq |
 txn_id
----+-------------+---------------+-------------+------------------------+----------
+--------+
  1 | The Shining |  Stephen King |        1977 |   Supernatural fiction |        2 |
 12192
```

> **Note**
>
> 如需區分大小寫，請針對結構描述、資料表和資料欄名稱使用雙引號 (" ")。如需詳細資訊，請參閱 enable_case_sensitive_identifier。

Aurora 與 Amazon Redshift 資料庫之間的資料類型差異

下表顯示 Aurora MySQL 和 Aurora PostgreSQL 的映射資料類型與對應目的地資料類型。Amazon Aurora 目前僅支援這些資料類型進行零 ETL 整合。

如果來源資料庫叢集中的資料表包含不受支援的資料類型，則資料表不同步，而且目的地目標無法取用該資料表。從來源到目標的串流會繼續進行，但是無法使用其中資料類型不受支援的資料表。若要修正資料表並使其可在目標目的地中使用，您必須手動還原重大變更，然後執行 `ALTER DATABASE...INTEGRATION REFRESH` 以重新整理整合。

> **Note**
>
> 您無法重新整理與 Amazon SageMaker 資料湖倉的零 ETL 整合。相反地，請刪除並嘗試再次建立整合。

主題

- Aurora MySQL
- Aurora PostgreSQL

Aurora MySQL

Aurora MySQL 資料類型	目標資料類型	Description	限制
INT	INTEGER	帶正負號的 4 位元組整數	無
SMALLINT	SMALLINT	帶正負號的 2 位元組整數	無
TINYINT	SMALLINT	帶正負號的 2 位元組整數	無
MEDIUMINT	INTEGER	帶正負號的 4 位元組整數	無
BIGINT	BIGINT	帶正負號的 8 位元組整數	無
INT UNSIGNED	BIGINT	帶正負號的 8 位元組整數	無

Aurora MySQL 資料類型	目標資料類型	Description	限制
TINYINT UNSIGNED	SMALLINT	帶正負號的 2 位元組整數	無
MEDIUMINT UNSIGNED	INTEGER	帶正負號的 4 位元組整數	無
BIGINT UNSIGNED	DECIMAL(20,0)	可選擇精確度 (有效位數) 的精確數值	無
DECIMAL(p,s) = NUMERIC(p,s)	DECIMAL(p,s)	可選擇精確度 (有效位數) 的精確數值	不支援大於 38 的精確度和大於 37 的小數位數
DECIMAL(p,s) UNSIGNED = NUMERIC(p,s) UNSIGNED	DECIMAL(p,s)	可選擇精確度 (有效位數) 的精確數值	不支援大於 38 的精確度和大於 37 的小數位數
FLOAT4/REAL	REAL	單精度浮點數	無
FLOAT4/REAL UNSIGNED	REAL	單精度浮點數	無
DOUBLE/REAL/FLOAT8	DOUBLE PRECISION	雙精度浮點數	無
DOUBLE/REAL/FLOAT8 UNSIGNED	DOUBLE PRECISION	雙精度浮點數	無
BIT(n)	VARBYTE(8)	可變長度二進位值	無
BINARY(n)	VARBYTE(n)	可變長度二進位值	無
VARBINARY(n)	VARBYTE(n)	可變長度二進位值	無
CHAR(n)	VARCHAR(n)	可變長度字串值	無

Aurora MySQL 資料類型	目標資料類型	Description	限制
VARCHAR(n)	VARCHAR(n)	可變長度字串值	無
TEXT	VARCHAR(65535)	可變長度字串值 最多 65,535 個字元	無
TINYTEXT	VARCHAR(255)	可變長度字串值 最多 255 個字元	無
MEDIUMTEXT	VARCHAR(65535)	可變長度字串值 最多 65,535 個字元	無
LONGTEXT	VARCHAR(65535)	可變長度字串值 最多 65,535 個字元	無
ENUM	VARCHAR(1020)	可變長度字串值 最多 1,020 個字元	無
SET	VARCHAR(1020)	可變長度字串值 最多 1,020 個字元	無
DATE	DATE	日曆日期 (年、月、日)	無
DATETIME	TIMESTAMP	日期和時間 (未使用時區)	無
TIMESTAMP(p)	TIMESTAMP	日期和時間 (未使用時區)	無
TIME	VARCHAR(18)	可變長度字串值 最多 18 個字元	無

Aurora MySQL 資料類型	目標資料類型	Description	限制
YEAR	VARCHAR(4)	可變長度字串值最多 4 個字元	無
JSON	SUPER	半結構化資料或文件作為值	無

Aurora PostgreSQL

Aurora PostgreSQL 的零 ETL 整合不支援自訂資料類型或擴充功能建立的資料類型。

Aurora PostgreSQL 資料類型	Amazon Redshift 資料類型	Description	限制
陣列	SUPER	半結構化資料或文件作為值	無
bigint	BIGINT	帶正負號的 8 位元組整數	無
bigserial	BIGINT	帶正負號的 8 位元組整數	無
bit varying(n)	VARBYTE(n)	可變長度字串值最多 16,777,216 個位元組	無
bit(n)	VARBYTE(n)	可變長度字串值最多 16,777,216 個位元組	無
bit、bit varying	VARBYTE(16777216)	可變長度字串值最多 16,777,216 個位元組	無
boolean	BOOLEAN	邏輯布林值 (true/false)	無

Aurora PostgreSQL 資料類型	Amazon Redshift 資料類型	Description	限制
bytea	VARBYTE(16777216)	可變長度字串值最多 16,777,216 個位元組	無
char(n)	CHAR(n)	固定長度字元字串值最多 65,535 個位元組	無
char varying(n)	VARCHAR(65535)	可變長度字元字串值最多 65,535 個字元	無
cid	BIGINT	帶正負號的 8 位元組整數	無
cidr	VARCHAR(19)	可變長度字串值最多 19 個字元	無
date	DATE	日曆日期 (年、月、日)	不支援大於 294,276 A.D. 的值
double precision	DOUBLE PRECISION	雙精度浮點數	未完全支援的低於正常值
gtsvector	VARCHAR(65535)	可變長度字串值最多 65,535 個字元	無
inet	VARCHAR(19)	可變長度字串值最多 19 個字元	無
integer	INTEGER	帶正負號的 4 位元組整數	無

Aurora PostgreSQL 資料類型	Amazon Redshift 資料類型	Description	限制
int2vector	SUPER	半結構化資料或文件作為值。	無
間隔	INTERVAL	持續時間	僅支援指定 year to month 或 day to second 限定詞的 INTERVAL 類型。
json	SUPER	半結構化資料或文件作為值	無
jsonb	SUPER	半結構化資料或文件作為值	無
jsonpath	VARCHAR(65535)	可變長度字串值最多 65,535 個字元	無
macaddr	VARCHAR(17)	可變長度字串值最多 17 個字元	無
macaddr8	VARCHAR(23)	可變長度字串值最多 23 個字元	無
money	DECIMAL(20,3)	貨幣金額	無
name	VARCHAR(64)	可變長度字串值最多 64 個字元	無

Aurora PostgreSQL 資料類型	Amazon Redshift 資料類型	Description	限制
numeric(p,s)	DECIMAL(p,s)	使用者定義的固定精確度值	• 不支援 NaN 值 • 精確度和小數位數必須明確定義，且不可大於 38 (精確度) 和 37 (小數位數) • 不支援負小數位數
oid	BIGINT	帶正負號的 8 位元組整數	無
類別向量	SUPER	半結構化資料或文件作為值。	無
pg_brin_bloom_summary	VARCHAR(65535)	可變長度字串值最多 65,535 個字元	無
pg_dependencies	VARCHAR(65535)	可變長度字串值最多 65,535 個字元	無
pg_lsn	VARCHAR(17)	可變長度字串值最多 17 個字元	無
pg_mcv_list	VARCHAR(65535)	可變長度字串值最多 65,535 個字元	無
pg_ndistinct	VARCHAR(65535)	可變長度字串值最多 65,535 個字元	無

Aurora PostgreSQL 資料類型	Amazon Redshift 資料類型	Description	限制
pg_node_tree	VARCHAR(65535)	可變長度字串值最多 65,535 個字元	無
pg_snapshot	VARCHAR(65535)	可變長度字串值最多 65,535 個字元	無
real	REAL	單精度浮點數	未完全支援的低於正常值
refcursor	VARCHAR(65535)	可變長度字串值最多 65,535 個字元	無
smallint	SMALLINT	帶正負號的 2 位元組整數	無
smallserial	SMALLINT	帶正負號的 2 位元組整數	無
serial	INTEGER	帶正負號的 4 位元組整數	無
text	VARCHAR(65535)	可變長度字串值最多 65,535 個字元	無
tid	VARCHAR(23)	可變長度字串值最多 23 個字元	無
不含時區的時間 [(p)]	VARCHAR(19)	可變長度字串值最多 19 個字元	不支援 `Infinity` 和 `-Infinity` 值

Aurora PostgreSQL 資料類型	Amazon Redshift 資料類型	Description	限制
含時區的時間 [(p)]	VARCHAR(22)	可變長度字串值 最多 22 個字元	不支援 `Infinity` 和 `-Infinity` 值
不含時區的時間戳記 [(p)]	TIMESTAMP	日期和時間 (未使用時區)	• 不支援 `Infinity` 和 `-Infinity` 值 • 不支援大於 `9999-12-31` 的值 • 不支援 B.C. 值
含時區的時間戳記 [(p)]	TIMESTAMPTZ	日期和時間 (包含時區)	• 不支援 `Infinity` 和 `-Infinity` 值 • 不支援大於 `9999-12-31` 的值 • 不支援 B.C. 值
tsquery	VARCHAR(65535)	可變長度字串值 最多 65,535 個字元	無
tsvector	VARCHAR(65535)	可變長度字串值 最多 65,535 個字元	無
txid_snapshot	VARCHAR(65535)	可變長度字串值 最多 65,535 個字元	無

Aurora PostgreSQL 資料類型	Amazon Redshift 資料類型	Description	限制
uuid	VARCHAR(36)	可變長度 36 個字元字串	無
xid	BIGINT	帶正負號的 8 位元組整數	無
xid8	DECIMAL(20, 0)	固定精確度小數位數	無
xml	VARCHAR(65535)	可變長度字串值最多 65,535 個字元	無

Aurora PostgreSQL 的 DDL 操作

Amazon Redshift 衍生自 PostgreSQL，因此由於其常見的 PostgreSQL 架構，它與 Aurora PostgreSQL 共用多項功能。零 ETL 整合利用這些相似性，簡化從 Aurora PostgreSQL 到 Amazon Redshift 的資料複寫，依名稱映射資料庫，並使用共用資料庫、結構描述和資料表結構。

管理 Aurora PostgreSQL 零 ETL 整合時，請考慮下列要點：

- 隔離是在資料庫層級管理。
- 複寫會在資料庫層級發生。
- Aurora PostgreSQL 資料庫會依名稱映射至 Amazon Redshift 資料庫，如果原始資料庫重新命名，資料會流向對應的重新命名 Redshift 資料庫。

雖然兩者相似，但 Amazon Redshift 和 Aurora PostgreSQL 仍有重要的差異。下列各節概述常見 DDL 操作的 Amazon Redshift 系統回應。

主題

- 資料庫操作
- 結構描述操作
- 資料表操作

資料庫操作

下表顯示資料庫 DDL 操作的系統回應。

DDL 操作	Redshift 系統回應
CREATE DATABASE	無操作
DROP DATABASE	Amazon Redshift 會捨棄目標 Redshift 資料庫中的所有資料。
RENAME DATABASE	Amazon Redshift 會捨棄原始目標資料庫中的所有資料，並重新同步新目標資料庫中的資料。如果新的資料庫不存在，您必須手動建立資料庫。如需指示，請參閱在 Amazon Redshift 中建立目的地資料庫。

結構描述操作

下表顯示結構描述 DDL 操作的系統回應。

DDL 操作	Redshift 系統回應
CREATE SCHEMA	無操作
DROP SCHEMA	Amazon Redshift 會捨棄原始結構描述。
RENAME SCHEMA	Amazon Redshift 會捨棄原始結構描述，然後重新同步新結構描述中的資料。

資料表操作

下表顯示資料表 DDL 操作的系統回應。

DDL 操作	Redshift 系統回應
CREATE TABLE	Amazon Redshift 會建立資料表。

DDL 操作	Redshift 系統回應
	有些操作會導致資料表建立失敗，例如在沒有主索引鍵的情況下建立資料表，或執行宣告式分割。如需詳細資訊，請參閱the section called “限制”及the section called “對零 ETL 整合進行疑難排解”。
DROP TABLE	Amazon Redshift 會捨棄資料表。
TRUNCATE TABLE	Amazon Redshift 會截斷資料表。
ALTER TABLE (RENAME...)	Amazon Redshift 會重新命名資料表或資料欄。
ALTER TABLE (SET SCHEMA)	Amazon Redshift 會捨棄原始結構描述中的資料表，然後重新同步新結構描述中的資料表。
ALTER TABLE (ADD PRIMARY KEY)	Amazon Redshift 會新增主索引鍵並重新同步資料表。
ALTER TABLE (ADD COLUMN)	Amazon Redshift 會將資料欄新增至資料表。
ALTER TABLE (DROP COLUMN)	如果資料欄不是主索引鍵資料欄，Amazon Redshift 會加以捨棄。否則，它會重新同步資料表。
ALTER TABLE (SET LOGGED/UNLOGGED)	如果您將資料表變更為已記錄，Amazon Redshift 會重新同步資料表。如果您將資料表變更為未記錄，Amazon Redshift 會捨棄資料表。

檢視和監控 Aurora 零 ETL 整合

您可以檢視 Amazon Aurora 零 ETL 整合的詳細資訊，以查看其組態資訊和目前狀態。您也可以透過在 Amazon Redshift 中查詢特定系統檢視來監控整合的狀態。此外，Amazon Redshift 會將某些與整合相關的指標發佈到 Amazon CloudWatch，您可以在 Amazon Redshift 主控台中檢視這些指標。

主題

- 檢視整合

- 使用 Amazon Redshift 的系統資料表監控整合
- 針對 Amazon Redshift 監控與 Amazon EventBridge 的整合

檢視整合

您可以使用 AWS 管理主控台、 AWS CLI或 RDS API 檢視 Aurora零 ETL 整合。

主控台

檢視零 ETL 整合的詳細資訊

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/：// 開啟 Amazon RDS 主控台。
2. 從左側導覽窗格中，選擇零 ETL 整合。
3. 選取整合以檢視其詳細資訊，例如其來源資料庫叢集和目標資料倉儲。



整合可以具有下列狀態：

- `Creating` – 正在建立整合。
- `Active` – 整合正在將交易資料傳送至目標資料倉儲。
- `Syncing` – 整合遇到可復原的錯誤，且正在重新植入資料。在完成重新同步之前，受影響的資料表無法進行查詢。
- `Needs attention` – 整合發生事件或錯誤，需要手動介入才能解決此問題。若要修正問題，請遵循整合詳細資訊頁面上錯誤訊息中的指示。

- `Failed` – 整合發生無法復原的事件或無法修正的錯誤。您必須刪除並重新建立整合。
- `Deleting` – 正在刪除整合。

AWS CLI

若要使用 檢視目前帳戶中的所有零 ETL 整合 AWS CLI，請使用 describe-integrations 命令並指定 `--integration-identifier`選項。

Example

針對 Linux、macOS 或 Unix：

```
aws rds describe-integrations \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

在 Windows 中：

```
aws rds describe-integrations ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

RDS API

若要使用 Amazon RDS API 檢視零 ETL 整合，請搭配 `IntegrationIdentifier` 參數使用 `DescribeIntegrations` 操作：

使用 Amazon Redshift 的系統資料表監控整合

Amazon Redshift 具有系統資料表和檢視，其中包含系統如何運作的相關資訊。您可以使用查詢任何其他資料庫資料表的方式，來查詢這些系統資料表和檢視。如需 Amazon Redshift 中系統資料表和檢視的詳細資訊，請參閱《Amazon Redshift 資料庫開發人員》指南中的系統資料表和檢視參考。

您可以查詢下列系統檢視和表格，以取得 Aurora 零 ETL 整合的相關資訊：

- SVV_INTEGRATION – 提供整合的組態詳細資料。
- SVV_INTEGRATION_TABLE_STATE – 描述整合內每個資料表的狀態。
- SYS_INTEGRATION_TABLE_STATE_CHANGE - 顯示整合的資料表狀態變更日誌。
- SYS_INTEGRATION_ACTIVITY – 提供已完成整合執行的相關資訊。

所有整合相關的 Amazon CloudWatch 指標都源自 Amazon Redshift。如需詳細資訊，請參閱《Amazon Redshift 管理指南》中的零 ETL 整合的指標。目前，Amazon Aurora 不會將任何整合指標發佈至 CloudWatch。

針對 Amazon Redshift 監控與 Amazon EventBridge 的整合

Amazon Redshift 會將整合相關的事件傳送至 Amazon EventBridge。如需事件清單及其對應的事件 ID，請參閱《Amazon Redshift 管理指南》中的搭配 Amazon EventBridge 的零 ETL 整合事件通知。

修改 Aurora 零 ETL 整合

您只能修改支援的資料倉儲中零 ETL 整合的名稱、描述和資料篩選選項。您無法修改用於加密整合的 AWS KMS 金鑰，或是來源或目標資料庫。

如果您將資料篩選條件新增至現有的整合，Aurora 會重新評估篩選條件，彷彿它一直存在一樣。它會移除目前在目標資料倉儲中不符合新篩選條件的任何資料。如果您從整合中移除資料篩選條件，則會將先前不符合篩選條件 (但現在符合) 的任何資料複寫至目標資料倉儲。如需更多詳細資訊，請參閱 the section called “零 ETL 整合的資料篩選”。

您可以使用 AWS 管理主控台、AWS CLI 或 Amazon RDS API 修改零 ETL 整合。

RDS 主控台

修改零 ETL 整合

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇零 ETL 整合，然後選擇您要修改的整合。
3. 選擇修改並修改任何可用的設定。
4. 滿意所有變更時，請選擇修改。

AWS CLI

若要使用 AWS CLI 修改零 ETL 整合，請呼叫 modify-integration 命令。除了 `--integration-identifier` 之外，請指定下列任何選項：

- `--integration-name` - 指定整合的新名稱。
- `--description` - 指定整合的新描述。

- `--data-filter` – 指定整合的資料篩選選項。如需更多詳細資訊，請參閱 the section called “零 ETL 整合的資料篩選”。

Example

下列要求會修改現有的整合。

對於 Linux、macOS 或 Unix：

```
aws rds modify-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 \
    --integration-name my-renamed-integration
```

在 Windows 中：

```
aws rds modify-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 ^
    --integration-name my-renamed-integration
```

RDS API

若要使用 RDS API 修改零 ETL 整合，請呼叫 ModifyIntegration 操作。指定整合識別符，以及您要修改的參數。

刪除 Aurora 零 ETL 整合

當您刪除零 ETL 整合時，Amazon Aurora 會將其從來源 Aurora 資料庫叢集中移除。您的交易資料不會從 Amazon Aurora 或分析目的地中刪除，但 Aurora 不會將新資料傳送到 Amazon Redshift 或 Amazon SageMaker。

只有在整合的狀態為 `Active`、`Failed`、`Syncing` 或 `Needs attention` 時，您才能刪除整合。

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 刪除零 ETL 整合。

主控台

刪除零 ETL 整合

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。

2. 從左側導覽窗格中，選擇零 ETL 整合。
3. 選取您要刪除的零 ETL 整合。
4. 選擇動作、刪除，並確認刪除。

AWS CLI

若要刪除零 ETL 整合，請使用 delete-integration 命令並指定 `--integration-identifier` 選項。

Example

對於 Linux、macOS 或 Unix：

```
aws rds delete-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

在 Windows 中：

```
aws rds delete-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

RDS API

若要使用 Amazon RDS API 刪除零 ETL 整合，請搭配 `IntegrationIdentifier` 參數使用 `DeleteIntegration` 操作：

針對 Aurora 零 ETL 整合進行疑難排解

您可以在分析目的地中查詢 SVV_INTEGRATION 系統資料表，以檢查零 ETL 整合的狀態。如果 `state` 資料欄具有 `ErrorState` 的值，表示有問題。如需詳細資訊，請參閱the section called “使用系統資料表進行監視”。

使用下列資訊，針對 Aurora 零 ETL 整合的常見問題進行疑難排解。

> **Important**
>
> 重新同步和重新整理操作不適用於與 Amazon SageMaker AI 資料湖倉的零 ETL 整合。如果整合發生問題，您必須刪除整合並建立新的整合。您無法重新整理或重新同步現有的整合。

主題

- 我無法建立零 ETL 整合
- 我的整合停滯在 Syncing 狀態
- 我的資料表未複寫至 Amazon Redshift
- 我的一個或多個 Amazon Redshift 資料表需要重新同步
- Amazon SageMaker AI Lakehouse 零 ETL 整合的整合失敗問題
- 在 Aurora PostgreSQL 的 DDL 交易完成之前，DDL 變更在 Amazon Redshift 中

我無法建立零 ETL 整合

如果您無法建立零 ETL 整合，請確定下列情況對於您的來源資料庫是正確的：

- 來源資料庫必須執行支援的資料庫引擎版本。如需支援的版本的清單，請參閱the section called “零 ETL 整合 ”。
- 您已正確設定資料庫參數。如果必要參數設定不正確或未與資料庫相關聯，則建立會失敗。請參閱 the section called “步驟 1：建立自訂資料庫叢集參數群組。”。

此外，請確定下列情況對於您的目標資料倉儲是正確的：

- 已啟用區分大小寫。請參閱開啟資料倉儲的區分大小寫。
- 您已新增正確的授權主體和整合來源。請參閱為您的 Amazon Redshift 資料倉儲設定授權。
- 資料倉儲已加密 (如果是佈建叢集)。請參閱 Amazon Redshift 資料庫加密。

我的整合停滯在 **Syncing** 狀態

如果您變更其中一個必要資料庫參數的值，您的整合可能會持續顯示 Syncing 的狀態。

若要修正此問題，請檢查與來源資料庫叢集相關聯之參數群組中的參數值，並確定其符合必要值。如需詳細資訊，請參閱the section called “步驟 1：建立自訂資料庫叢集參數群組。”。

如果您修改任何參數，請務必將資料庫叢集重新開機以套用變更。

我的資料表未複寫至 Amazon Redshift

如果您沒有看到一或多個資料表反映在 Amazon Redshift 中，您可以執行下列命令來重新同步它們：

```
ALTER DATABASE dbname INTEGRATION REFRESH TABLES table1, table2;
```

如需詳細資訊，請參閱 Amazon Redshift SQL 參考中的 ALTER DATABASE。

您的資料可能不會複寫，因為一或多個來源資料表沒有主索引鍵。Amazon Redshift 中的監控儀表板會將這些資料表的狀態顯示為 `Failed`，而整體零 ETL 整合的狀態會變更為 `Needs attention`。若要解決此問題，您可以識別資料表中可成為主索引鍵的現有索引鍵，也可以新增合成主索引鍵。如需詳細解決方案，請參閱在下列資源：

- 建立與 Amazon Redshift 的 Amazon Aurora MySQL 或 Amazon RDS for MySQL 零 ETL 整合時，在沒有主索引鍵的情況下處理資料表
- 建立與 Amazon Redshift 的 Amazon Aurora PostgreSQL 零 ETL 整合時，在沒有主索引鍵的情況下處理資料表

我的一個或多個 Amazon Redshift 資料表需要重新同步

在來源資料庫上執行某些命令，可能需要重新同步您的資料表。在這些情況下，SVV_INTEGRATION_TABLE_STATE 系統檢視會顯示 `ResyncRequired` 的 `table_state`，這表示整合必須將該特定資料表的資料從 MySQL 完全重新載入至 Amazon Redshift。

當資料表開始重新同步時，其會進入 `Syncing` 的狀態。您不需要採取任何手動動作，即可重新同步資料表。當資料表資料重新同步時，您無法在 Amazon Redshift 中存取該資料。

以下是一些可以將資料表置於 `ResyncRequired` 狀態的範例操作，以及可以考慮的替代方法。

作業	範例	備用

將資料欄新增到特定位置	`ALTER TABLE table_name`
	`  ADD COLUMN column_name  INTEGER`
	`  NOT NULL first;`
		Amazon Redshift 不支援使用 `first` 或 `after` 關鍵字將資料欄新增到特定位置。如果目標資料表中的資料欄順序並不重要，請使

作業	範例	備用
		用更簡單的命令，將資料欄新增至資料表的尾端： `ALTER TABLE table_name ADD COLUMN column_name column_type ;`

作業	範例	備用
新增具有預設 CURRENT_TIMESTAMP 的時間戳記資料欄	`ALTER TABLE table_name ADD COLUMN column_name TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP;`	現有資料表資料列的 CURRENT_TIMESTAMP 值是由 Aurora MySQL 計算，若沒有完整的資料表資料重新同步，就無法在 Amazon Redshift 中進行模擬。 如果可能，請將預設值切換為 2023-01-01 00:00:15 之類的常值常數，避免資料表可用性中的延遲。
在單一命令內執行多個資料欄操作	`ALTER TABLE table_name ADD COLUMN column_1, RENAME COLUMN column_2 TO column_3;`	考慮將命令分成兩個單獨的操作 (ADD 和 RENAME)，這不需要重新同步。

Amazon SageMaker AI Lakehouse 零 ETL 整合的整合失敗問題

如果您在現有的零 ETL Amazon SageMaker AI 整合中遇到與湖房整合的問題，唯一的解決方法是刪除整合並建立新的整合。與其他 AWS 服務不同，零 ETL 整合不支援重新整理或重新同步操作。

若要解決整合問題：

1. 使用主控台、CLI 或 API 刪除有問題的零 ETL 整合。
2. 驗證來源資料庫和目標資料倉儲組態是否正確。
3. 使用相同或更新的組態建立新的零 ETL 整合。

此程序將導致資料管道完全重新初始化，這可能需要一些時間，取決於來源資料庫的大小。

在 Aurora PostgreSQL 的 DDL 交易完成之前，DDL 變更在 Amazon Redshift 中

在 Aurora PostgreSQL 零 ETL 整合中完成 DDL 操作之前，DDL 變更可能會出現在 Amazon Redshift 中。如需詳細資訊，請參閱Aurora PostgreSQL 的 DDL 操作。

使用 Aurora Serverless v2

Aurora Serverless v2 是 Amazon Aurora 的隨需、自動擴展組態。Aurora Serverless v2 有助於自動執行監控工作負載和調整資料庫容量的程序。容量會根據應用程式需求自動調整。您只需支付資料庫叢集消耗的資源費用。因此，Aurora Serverless v2 可協助您保持在預算範圍內，避免支付未使用的電腦資源費用。

這種自動化類型對於多租用戶資料庫、分散式資料庫、開發和測試系統以及其他具有高度變化和不可預測工作負載的環境，尤其有價值。

主題

- Aurora Serverless v2 使用案例
- Aurora Serverless v2 的優點
- Aurora Serverless v2 的運作方式
- Aurora Serverless v2 的要求與限制
- 建立使用 Aurora Serverless v2 的資料庫叢集
- 管理 Aurora Serverless v2 資料庫叢集
- Aurora Serverless v2 的效能和擴展
- 使用 Aurora Serverless v2 的自動暫停和繼續擴展至零個 ACU
- 遷移至 Aurora Serverless v2

Aurora Serverless v2 使用案例

Aurora Serverless v2 支援多種類型的資料庫工作負載。這些範圍從開發和測試環境到具有不可預測工作負載的網站和應用程式，到需要大規模和可用性之高要求的業務關鍵應用程式。

Aurora Serverless v2 在以下使用案例中特別有用：

- 變數工作負載 - 您正在執行的工作負載會有無法預測的突發活動增加。例如一個交通站點，在開始下雨時會看到活動突然增加。另一個是電子商務網站，當您提供銷售或特別促銷活動時，流量會增加。透過 Aurora Serverless v2，您的資料庫會自動擴展容量以滿足應用程式峰值負載需求，並在活動激增結束時縮小規模。透過 Aurora Serverless v2，您不再需要針對峰值或平均容量進行佈建。您可以指定容量上限來處理最壞情況，除非需要，否則不會使用該容量。

Aurora Serverless v2 的擴展精細程度有助於讓容量緊密符合資料庫需求。對於已佈建叢集，擴充規模需要新增全新的資料庫執行個體。對於 Aurora Serverless v1 叢集，擴充規模需要將叢集的 Aurora 容量單元 (ACU) 數量增加一倍，例如從 16 個增加到 32 個，或 32 個增加到 64 個。相較之下，只需要多一點容量時，Aurora Serverless v2 可以新增半個 ACU。根據處理工作負載增加所需的額外容量，它可以新增 0.5、1、1.5、2 或額外的半個 ACU。當工作負載減少且不再需要容量時，它可以移除 0.5、1、1.5、2 或額外的半個 ACU。

- 多租用戶應用程式 – 透過 Aurora Serverless v2，您就無須為機群中的每個應用程式分別管理資料庫容量。Aurora Serverless v2 將為您管理個別資料庫容量。

  您可以為每個租用戶建立叢集。如此一來，您就可以使用複製、快照還原和 Aurora 全球資料庫等功能，來增強高可用性和適合每個租用戶的災難復原。

  根據一天中的時間、一年中的時間、促銷活動等，每個租用戶可能具有特定的繁忙和閒置時段。每個叢集都可能具有廣泛的容量範圍。如此一來，活動量低的叢集會產生最低的資料庫執行個體費用。任何叢集都可以快速擴充規模，以便處理高活動期間。

- 新的應用程式 - 您正在部署新的應用程式，但不確定所需的資料庫執行個體大小。透過使用 Aurora Serverless v2，您可以建立具有一個或多個資料庫執行個體的叢集，並將該資料庫自動擴展至應用程式所需容量。

- 混合用途應用程式 - 假設您有線上交易處理 (OLTP) 應用程式，但您定期遇到查詢流量高峰。透過為叢集中的 Aurora Serverless v2 資料庫執行個體指定提升層，您可以設定叢集，讓讀取器資料庫執行個體可以獨立於寫入器資料庫執行個體進行擴展，以便處理額外的負載。當使用率高峰降低時，讀取器資料庫執行個體會縮小以便符合寫入器資料庫執行個體的容量。

- 容量規劃 - 我們猜想您通常透過修改叢集中所有資料庫執行個體的資料庫執行個體類別，藉以調整資料庫容量，或驗證工作負載的最佳資料庫容量。使用 Aurora Serverless v2，您可以消除此管理開銷。您可以執行工作負載並檢查資料庫執行個體的實際擴展程度，來確定適當的最小和最大容量。

  您可以將現有資料庫執行個體從已佈建修改為 Aurora Serverless v2，或從 Aurora Serverless v2 修改為已佈建。在這種情況下，您不需要建立新的叢集或新的資料庫執行個體。

  使用 Aurora 全球資料庫，次要叢集可能不需要與主要叢集一樣多的容量。您可以在次要叢集中使用 Aurora Serverless v2 資料庫執行個體。如此一來，如果次要區域被提升並接管應用程式的工作負載，叢集容量就可以擴充規模。

- 開發與測試 – 除了執行高要求的應用程式之外，您還可以將 Aurora Serverless v2 用於開發和測試環境。使用 Aurora Serverless v2 時，您可以建立具有較低最小容量的資料庫執行個體，而非使用高載 db.t* 資料庫執行個體類別。您可以設定足夠高的最大容量，讓這些資料庫執行個體仍然可以執行大

量工作負載，而不會面臨記憶體不足。當資料庫未使用時，所有資料庫執行個體都會縮減規模以避免不必要的費用。

> **Tip**
>
> 為了方便在開發和測試環境中使用 Aurora Serverless v2，當您建立新叢集時，AWS 管理主控台提供 Easy create (輕鬆建立) 快速鍵。如果選擇 Dev/Test (開發/測試) 選項，Aurora 會建立具有 Aurora Serverless v2 資料庫執行個體以及開發和測試系統之典型容量範圍的叢集。

將 Aurora Serverless v2 用於現有佈建的工作負載

假設您已在佈建的叢集上執行 Aurora 應用程式。您可以新增一或多個 Aurora Serverless v2 資料庫執行個體到現有叢集做為讀取器資料庫執行個體，來檢查應用程式如何搭配 Aurora Serverless v2 運作。您可以檢查讀取器資料庫執行個體的擴充和縮減規模頻率。您可以使用 Aurora 容錯移轉機制來提升 Aurora Serverless v2 資料庫執行個體成為寫入器，並檢查它如何處理讀/寫工作負載。如此一來，您可以用最短的停機時間進行切換，而無需變更用戶端應用程式使用的端點。如需將現有叢集轉換為 Aurora Serverless v2 的程序細節，請參閱遷移至 Aurora Serverless v2。

Aurora Serverless v2 的優點

Aurora Serverless v2 適用於變動或「尖峰」工作負載。由於這種不可預測的工作負載，您可能難以規劃何時應變更資料庫容量。您也可能無法使用熟悉的機制 (如新增資料庫執行個體或變更資料庫執行個體類別) 進行足夠快速的容量變更。Aurora Serverless v2 提供以下優勢來協助處理此類使用案例：

- 比已佈建的容量管理更簡單 – Aurora Serverless v2 可減輕規劃資料庫執行個體大小和隨著工作負載變化調整資料庫執行個體大小的工作量。它還可以減少為叢集中的所有資料庫執行個體維護一致容量的工作量。
- 在高活動期間更快、更輕鬆地擴展 – Aurora Serverless v2 可根據需要擴展運算與記憶體容量，而不會中斷中斷用戶端交易或整體工作負載。能夠將讀取器資料庫執行個體搭配 Aurora Serverless v2 使用，協助您同時運用水平擴展以及垂直擴展。能夠使用 Aurora 全球資料庫，代表了您可以將 Aurora Serverless v2 讀取工作負載分散到多個 AWS 區域。此功能比已佈建叢集的擴展機制更方便。它也比 Aurora Serverless v1 中的擴展功能更快、更精細。
- 在低活動期間具有成本效益 – Aurora Serverless v2 可協助避免過度佈建資料庫執行個體。Aurora Serverless v2 在資料庫執行個體擴充規模時會以細微增量來新增資源。您僅需為使用的資料庫資源

付費。Aurora Serverless v2 資源使用量是以秒計費。如此一來，當資料庫執行個體縮減規模時，將立即登記減少的資源使用量。

- 搭配已佈建的功能更佳 - 您可以在 Aurora Serverless v2 中使用 Aurora Serverless v1 未提供的許多 Aurora 功能。例如，搭配 Aurora Serverless v2，您可以使用讀取器資料庫執行個體、全球資料庫、AWS Identity and Access Management(IAM) 資料庫身分驗證和績效詳情。您還可以使用比 Aurora Serverless v1 更多的組態參數。

  尤其是，搭配 Aurora Serverless v2 可以善用已佈建叢集的下列功能：

  - 讀取器資料庫執行個體 – Aurora Serverless v2 可以利用讀取器資料庫執行個體來水平擴展。當叢集包含一個或多個讀取器資料庫執行個體時，如果寫入器資料庫執行個體出現問題，叢集可以立即進行容錯移轉。這是 Aurora Serverless v1 不具備的功能。
  - 多可用區域叢集 - 您可以將叢集的 Aurora Serverless v2 資料庫執行個體分散到多個可用區域 (AZ)。設定多可用區域叢集有助於確保業務持續性，即使在遇到會影響整個可用區域的極少數情況下也是如此。這是 Aurora Serverless v1 不具備的功能。
  - 全球資料庫 - 您可以將 Aurora Serverless v2 結合 Aurora 全球資料庫，以便在其他 AWS 區域 中建立叢集的其他唯讀複本以用於災難復原用途。
  - RDS Proxy – 您可使用 Amazon RDS Proxy，可讓應用程式集合並共用資料庫連線，以改善其擴展能力。
- 比 Aurora Serverless v1 更快、更精細、破壞性更小的擴展 – Aurora Serverless v2 可以更快地擴充和縮減規模。擴展可以將容量改變的量壓低至 0.5 個 ACU，而不是將 ACU 的數量加倍或減半。擴展在處理過程中通常不會暫停。與 Aurora Serverless v1 相同，擴展不涉及您必須注意的事件。可在 SQL 陳述式執行中且交易開啟時進行擴展，無需等待安靜點。

Aurora Serverless v2 的運作方式

以下概觀描述 Aurora Serverless v2 如何運作。

主題

- Aurora Serverless v2 概觀
- Aurora 資料庫叢集的組態
- Aurora Serverless v2 容量
- Aurora Serverless v2 擴展
- Aurora Serverless v2 和高可用性
- Aurora Serverless v2 和儲存

- Aurora 叢集的組態參數

Aurora Serverless v2 概觀

Amazon Aurora Serverless v2 適用於高要求、高度可變的工作負載。例如，您的資料庫使用量可能在短時間內很大，然後是長時間的輕度活動或完全沒有活動。某些範例是具有定期促銷活動的零售網站、遊戲或體育網站，以及在需要時產生報告的資料庫。其他則為開發和測試環境，及用量可能會迅速增加的新應用程式。對於這類類似案例和許多其他情況，並不一定可以使用佈建的模型事先正確設定容量。如果您過度佈建且具有未使用的容量，也可能會產生較高的成本。

相比之下，Aurora 佈建的叢集適用於穩定的工作負載。對於已佈建叢集，您可以選擇具有預先定義記憶體量、CPU 功率、I/O 頻寬等的資料庫執行個體類別。若您的工作負載發生變化，您可以手動修改寫入器和讀取器的執行個體類別。當您可以在預期的耗用模式之前調整容量，佈建的模型就能順利運作，並且在您變更集中寫入器和讀取器的執行個體類別時，短暫中斷是可以接受的。

Aurora Serverless v2 的架構完全是為了支援可即時擴展的無伺服器資料庫叢集。Aurora Serverless v2 旨在提供與已佈建寫入器和讀取器相同程度的安全性和隔離性。這些方面在多租戶無伺服器雲端環境中至關重要。動態擴展機制的額外負荷很小，因此可以快速回應資料庫工作負載的變化。其功能強大，足以滿足大幅增加的處理需求。

使用 Aurora Serverless v2，您可以建立 Aurora 資料庫叢集，而不必為每個寫入器和讀取器鎖定特定的資料庫容量。您指定最小和最大的容量範圍。Aurora 在該容量範圍內擴展叢集中的每個 Aurora Serverless v2 寫入器或讀取器。透過使用每個寫入器或讀取器都可以動態擴展的多可用區域叢集，您可以利用動態擴展和高可用性。

Aurora Serverless v2 根據您的最小和最大容量規格自動擴展資料庫資源。擴展速度很快，因為大多數擴展事件作業皆將寫入器或讀取器保留在同一主機上。在極少數狀況下，Aurora Serverless v2 寫入器或讀取器從一個主機移至另一個主機，Aurora Serverless v2 會自動管理連線。您無需變更資料庫用戶端應用程式程式碼或資料庫連線字串。

利用 Aurora Serverless v2，與已佈建叢集一樣，儲存容量和運算容量是獨立的。當我們提到 Aurora Serverless v2 容量和擴展時，總是在增加或減少運算容量。因此，即使 CPU 和記憶體容量縮減規模到低等級，您的叢集也可以包含許多 TB 的資料。

您無需佈建和管理資料庫伺服器，而是指定資料庫容量。如需 Aurora Serverless v2 容量的詳細資訊，請參閱 Aurora Serverless v2 容量。每個 Aurora Serverless v2 寫入器或讀取器的實際容量會隨時間而變化，依您的工作量而定。如需有關該機制的詳細資料，請參閱 Aurora Serverless v2 擴展。

> **Important**
>
> 利用 Aurora Serverless v1，您的叢集具有單一的運算容量度量，可在最小和最大容量值之間擴展。利用 Aurora Serverless v2，您的叢集除了寫入器之外還可包含讀取器。每個 Aurora Serverless v2 寫入器和讀取器可在容量值下限和上限之間進行擴展。因此，Aurora Serverless v2 叢集的總容量取決於您為資料庫叢集定義的容量範圍及叢集中寫入器和讀取器的數量。在任何特定時間點，您只需為您 Aurora 資料庫叢集中正在使用的 Aurora Serverless v2 容量付費。

Aurora 資料庫叢集的組態

對於每個 Aurora 資料庫叢集，您可以選擇 Aurora Serverless v2 容量、已佈建容量或兩者的任意組合。

您可以設定叢集，其中包含 Aurora Serverless v2 和已佈建容量，稱為混合組態叢集。例如，假設您需要比 Aurora Serverless v2 寫入器可用的更多讀取/寫入容量。在這種情況下，您可以使用非常大的已佈建寫入器來設定叢集。在該種情況下，您仍可為讀取器使用 Aurora Serverless v2。或者假設叢集的寫入工作負載有變化，但讀取工作負載穩定。在這種情況下，您可以使用 Aurora Serverless v2 寫入器和一個或多個已佈建讀取器來設定叢集。

您還可以設定一個資料庫叢集，其中所有容量皆由 Aurora Serverless v2 管理。為此，您可以建立一個新叢集,並從一開始就使用 Aurora Serverless v2。或者，您也可以將現有叢集中的所有已佈建容量替換為 Aurora Serverless v2。例如，舊版引擎的某些升級路徑需要從已佈建的寫入器開始，然後將其替換為 Aurora Serverless v2 寫入器。如需有關使用 Aurora Serverless v2 建立新資料庫叢集或將現有資料庫叢集切換至 Aurora Serverless v2 的程序，請參閱 建立 Aurora Serverless v2 資料庫叢集 和 從已佈建叢集切換至 Aurora Serverless v2。

如果您在資料庫叢集中完全不使用 Aurora Serverless v2，則資料庫叢集中的所有寫入器和讀取器都是已佈建的。這是大多數使用者熟悉的最古老、最常見的資料庫叢集。事實上，在 Aurora Serverless 之前，這種 Aurora 資料庫叢集並無特殊名稱。已佈建容量是常數。費用相對容易預測。但是，您必須提前預測您需要多少容量。在某些情況下，您的預測可能不準確，或者您的容量需求可能會發生變化。在這些情況下，您的資料庫叢集可能會變得佈建不足 (比您想要的速度慢) 或過度佈建 (比您想要的成本要高)。

Aurora Serverless v2 容量

Aurora Serverless v2 的測量單位是 Aurora 容量單元 (ACU)。Aurora Serverless v2 容量與您用於已佈建叢集的資料庫執行個體類別無關。

每個 ACU 是大約 2 GiB 的記憶體、對應的 CPU 和聯網的組合。您可以使用此度量單位指定資料庫容量範圍。`ServerlessDatabaseCapacity` 和 `ACUUtilization` 指標可協助您確定資料庫實際使用的容量，及該容量在指定範圍內的位置。

在任何時候，每個 Aurora Serverless v2 資料庫寫入器或讀取器皆有一個容量。容量表示為代表 ACU 的浮點數。每當寫入器或讀取器擴展時，容量都會增加或減少。此值每秒測量一次。對於要使用 Aurora Serverless v2 的每個資料庫叢集，您定義一個容量範圍：每個 Aurora Serverless v2 寫入器或讀取器可在其間擴展的最小和最大容量值。容量範圍對於資料庫叢集中的每個 Aurora Serverless v2 寫入器或讀取器都相同。每個 Aurora Serverless v2 寫入器或讀取器都有自己的能力，落在該範圍內的某個地方。

下表顯示 Aurora MySQL 和 Aurora PostgreSQL 的 Aurora Serverless v2 容量範圍和引擎版本支援。

容量範圍 (ACU)	Aurora MySQL 支援的版本	Aurora PostgreSQL 支援的版本
0.5–128	3.02.0 及更新版本	13.6 版和更新版本、14.3 版和更新版本、15.2 版和更新版本、16.1 版和更新版本
0.5–256	3.06.0 及更新版本	13.13 版和更新版本、14.10 版和更新版本、15.5 版和更新版本、16.1 版和更新版本
0–256	3.08.0 及更新版本	13.15 版和更新版本、14.12 版和更新版本、15.7 版和更新版本、16.3 版和更新版本

下表顯示 Aurora Serverless v2 平台版本及其 ACU 範圍和效能特性。

Aurora Serverless v2 平台版本	ACU 範圍	效能
1	0–128	基準效能
2	0–256	基準效能
3	0–256	與平台版本 2 相比，效能提升高達 30%

> **Note**
>
> 指定叢集的可用擴展範圍取決於引擎版本和平台版本。可以在功能較差的平台版本上執行更強大的引擎版本，反之亦然。擴展範圍取決於功能最低的引擎或平台版本。

您可以在 的執行個體組態區段中 AWS 管理主控台 或透過 API 檢視 DBCluster `ServerlessV2PlatformVersion` 的 來判斷叢集在哪個平台版本上執行。

您可以定義的最小 Aurora Serverless v2 容量為 0 個 ACU，適用於支援自動暫停功能的 Aurora Serverless v2 版本。若其小於或等於最大容量值，您可以指定更大的數字。將最小容量設定為較小的數字可使負載較輕的資料庫叢集消耗最少的運算資源。同時，它們隨時準備好立即接受連線，並在忙碌時擴充規模。

我們建議將最小值設定為允許每個資料庫寫入器或讀取器將應用程式的工作集保存在緩衝集區中的值。這樣，緩衝集區的內容在閒置期間不會被丟棄。如需選擇最小容量值時的所有注意事項，請參閱 選擇叢集的最小值 Aurora Serverless v2 容量設定。如需選擇最大容量值時的所有注意事項，請參閱 選擇叢集的最大值 Aurora Serverless v2 容量設定。

根據您在多可用區域部署中設定讀取器的方式，讀取器的容量可以繫結至寫入器的容量，也可以獨立地調整。如需有關如何執行此作業的詳細資訊，請參閱 Aurora Serverless v2 擴展。

監控 Aurora Serverless v2 涉及隨時間測量資料庫叢集中寫入器和讀取器的容量值。若資料庫未縮減規模至最小容量，則可採取諸如調整最小容量和最佳化資料庫應用程式等作業。如果資料庫持續達到其最大容量，則可採取諸如增加最大值之類的動作。您還可以最佳化資料庫應用程式，並將查詢負載分佈到更多讀取器中。

Aurora Serverless v2 容量的費用是以 ACU 小時計算的。如需有關如何計算 Aurora Serverless v2 費用的資訊，請參閱 Aurora 定價頁。

假設叢集中的寫入器和讀取器總數為 *n*。在這種情況下，當您未執行任何資料庫操作時，叢集大約會使用 *n* x *minimum ACUs*。Aurora 本身可能會執行導致少量負載的監控或維護作業。當資料庫滿容量執行時，該叢集的消耗不超過 *n* x *maximum ACUs*。

如需選擇適當的最小和最大 ACU 值的詳細資訊，請參閱選擇 Aurora 叢集的 Aurora Serverless v2 容量範圍。您指定的最小和最大 ACU 值也會影響某些 Aurora 組態參數對 Aurora Serverless v2 的運作方式。如需容量範圍和組態參數之間互動的詳細資訊，請參閱使用 Aurora Serverless v2 的參數群組。

Aurora Serverless v2 擴展

對於每個 Aurora Serverless v2 寫入器或讀取器，Aurora 會持續追蹤 CPU、記憶體和網路等資源的使用率。這些測量統稱為負載。負載包括應用程式執行的資料庫作業。它還包括資料庫伺服器和 Aurora 管理任務的背景處理。當容量受到上述任何一種限制時，Aurora Serverless v2 擴充規模。Aurora Serverless v2 也會在偵測到這麼做可得到解決的效能問題時進行擴充。您可使用 Aurora Serverless v2 的重要 Amazon CloudWatch 指標 和 使用績效詳情監控 Aurora Serverless v2 效能 中的程序，監控資源使用率及其如何影響 Aurora Serverless v2 擴展。

資料庫叢集中的寫入器和讀取器的負載可能會有所不同。寫入器會處理所有資料定義語言 (DDL) 陳述式，如 `CREATE TABLE`、`ALTER TABLE` 和 `DROP TABLE`。寫入器也會處理所有資料處理語言 (DML) 陳述式，例如 `INSERT` 和 `UPDATE`。讀取器可以處理唯讀陳述式，例如 `SELECT` 查詢。

擴展是增加或減少資料庫 Aurora Serverless v2 容量的操作。搭配 Aurora Serverless v2，每個寫入器和讀取器都有自己的目前容量值，以 ACU 為單位。當 Aurora Serverless v2 目前容量太低而無法處理負載時，會將寫入器或讀取器擴展到更高的容量。當其目前容量高於需要時，它會將寫入器或讀取器縮減到更低的容量。

與每次資料庫叢集達到閾值時透過將容量加倍來進行擴展的 Aurora Serverless v1 不同，Aurora Serverless v2 可逐步增加容量。當您的工作負載需求開始達到寫入器或讀取器的目前資料庫容量時，Aurora Serverless v2會增加該寫入器或讀取器的 ACU 數量。Aurora Serverless v2 會按照所需的增量來調整容量，以便為所耗用的資源提供最佳效能。擴展以小至 0.5 ACU 的增量發生。目前容量越大，擴展增量就越大，因此可以發生更快的擴展。

由於Aurora Serverless v2擴展非常頻繁、精細且不中斷，因此不會 AWS 管理主控台 像Aurora Serverless v1這樣造成離散事件。反之，您可以測量 `ServerlessDatabaseCapacity` 和 `ACUUtilization` 等 Amazon CloudWatch 指標，並隨時間追蹤其最小值、最大值和平均值。如要進一步了解 Aurora 指標，請參閱 在 Amazon Aurora 叢集中監控指標。如需監控 Aurora Serverless v2 的提示，請參閱 Aurora Serverless v2 的重要 Amazon CloudWatch 指標。

您可以選擇與關聯的寫入器同時擴展讀取器，也可以獨立於寫入器進行擴展。您可以透過為該讀取器指定提升層來執行此操作。

- 提升層 0 和 1 中的讀取器與寫入器同時擴展。這種擴展行為使優先順序層 0 和 1 中的讀取器成為可用性的理想選擇。這是因為它們總是調整到適當的容量，以便在發生容錯移轉時從寫入器處接管工作負載。
- 提升層 2 至 15級中的讀取器可獨立於寫入器擴展。每個讀取器保持在您為叢集指定的最小和最大 ACU 值範圍內。當讀取器獨立於關聯的寫入器資料庫進行擴展時，它可能會處於閒置狀態並縮減規

模，而寫入器繼續處理大量交易。如果在較低的提升層中沒有其他讀取器可用，則它仍可用作容錯移轉目標。但是，如果將其提升為寫入器，則可能需要擴充規模以處理寫入器的全部工作負載。

如需提升層的詳細資訊，請參閱選擇 Aurora Serverless v2 讀取器的提升層。

擴展點和來自 Aurora Serverless v1 相關逾時期間的概念不適用於 Aurora Serverless v2。Aurora Serverless v2 擴展可能發生在資料庫連線開啟、SQL 交易正在處理、資料表遭鎖定及臨時表正在使用時。Aurora Serverless v2 不會等待安靜點開始擴展。擴展不會中斷正在進行的任何資料庫作業。

如果您的工作負載需要比單一寫入器和單一讀取器所提供更多的讀取容量，您可以將多個 Aurora Serverless v2 讀取器新增至叢集。每個 Aurora Serverless v2 讀取器皆可在您為資料庫叢集指定的最小和最大容量值範圍內擴展。您可以使用叢集的讀取器端點，將唯讀工作階段導向讀取器，並減少寫入器的負載。

Aurora Serverless v2 是否執行擴展，以及一旦啟動後擴展發生的速度，還取決於叢集的最小和最大 ACU 設定。此外，這取決於讀取器是設定為與寫入器一起擴展還是獨立擴展。如需有關影響 Aurora Serverless v2 擴展因素的詳細資料，請參閱 Aurora Serverless v2 的效能和擴展。

擴展至零

在最近的 Aurora MySQL 和 Aurora PostgreSQL 版本中，Aurora Serverless v2 寫入器和讀取器可以完全縮減到零個 ACU。我們將此功能稱為自動暫停和繼續，或自動暫停。您可以選擇是否允許此行為，方法是指定最小容量的零或非零值。您也可以選擇 Aurora Serverless v2 執行個體暫停前的等待時間。如需哪些版本具有此功能的資訊，請參閱 Aurora Serverless v2 容量。如需如何有效使用此功能的相關資訊，請參閱 使用 Aurora Serverless v2 的自動暫停和繼續擴展至零個 ACU。

在較舊的 Aurora MySQL 和 Aurora PostgreSQL 版本中，閒置 Aurora Serverless v2 寫入器和讀取器可以縮減規模到您為叢集指定的最小 ACU 值，但不是讓 ACU 完全歸零。在這種情況下，當您設定容量範圍時，無法選擇零個 ACU。這種行為與 Aurora Serverless v1 不同，後者可能會在閒置一段時間後暫停，但在開啟新連接時需要一些時間才能恢復。

當一段時間內不需要您的資料庫叢集與 Aurora Serverless v2 容量，您也可以像使用已佈建的資料庫叢集一樣停止和啟動整個叢集。此技術最適合用於開發和測試系統，其中可能一次需要幾個小時，而且繼續叢集的速度並不重要。停止/啟動叢集功能適用於所有 Aurora Serverless v2 版本。如需該功能的詳細資訊，請參閱 停用和啟動 Amazon Aurora 資料庫叢集。

Aurora Serverless v2 和高可用性

為 Aurora 資料庫叢集建立高可用性的方法，是使其成為多可用區域資料庫叢集。多可用區域 Aurora 資料庫叢集在多個可用區域 (AZ) 中始終具有可用的運算容量。該組態保持資料庫的正常執行，即使在

出現嚴重停機的情況下也是如此。Aurora 會在出現影響寫入器甚至整個可用區域的問題時，執行自動容錯移轉。利用 Aurora Serverless v2，您可以選擇待命運算容量隨著寫入器的容量進行縱向擴展和縮減。這樣，第二個可用區域中的運算容量就可以隨時接管目前工作負載。同時，當資料庫閒置時，所有可用區域中的運算容量都可以縮減規模。如需 Aurora 如何使用 AWS 區域 和可用區域的詳細資訊，請參閱 Aurora 資料庫執行個體的高可用性。

Aurora Serverless v2 多可用區域功能除了寫入器，也會使用讀取器。與 Aurora Serverless v1 相比，Aurora Serverless v2 對閱讀器的支援是新的。您可以將最多 15 個 Aurora Serverless v2 讀取器跨 3 個可用區域分散到 Aurora 資料庫叢集。

對於即使發生影響整個叢集或整個 AWS 區域的問題，仍必須保持可用的業務關鍵應用程式，您可以設定 Aurora 全域資料庫。您可於次要叢集中使用 Aurora Serverless v2 容量，以便其準備好在災難復原期間接管。當資料庫不忙碌時，它們還可以縮減規模。如需 Aurora 全球資料庫的詳細資訊，請參閱 使用 Amazon Aurora 全球資料庫。

Aurora Serverless v2 的運作原理類似於為容錯移轉和其他高可用性功能進行佈建。如需詳細資訊，請參閱Amazon Aurora 的高可用性。

假設您要確保 Aurora Serverless v2 叢集的最大可用性。除了寫入器之外，您還可建立讀取器。如果您將讀取器指派到提升層 0 或 1，則寫入器發生擴展時，讀取器也會一併發生。這樣，在發生容錯移轉時，具有相同容量的讀取器一律可以接管寫入器。

假設您希望在叢集繼續處理交易的同時為您的業務執行季度報告。如果新增 Aurora Serverless v2 讀取器至叢集，然後將其指派給提升層 2 到 15，您可以直接連接到該讀取器以執行報告。根據報告查詢的記憶體密集和 CPU 密集程度，該讀取器可以擴充規模以適應工作負載。然後，它可以在報告完成後再次縮減規模。

Aurora Serverless v2 和儲存

每個 Aurora 資料庫叢集的儲存，由分散在三個可用區域中的所有資料的六個副本組成。無論您的資料庫叢集是否包含除寫入器之外的任何讀取器，此內建資料複寫皆適用。如此，即使存在影響叢集運算容量的問題，您的資料也是安全的。

Aurora Serverless v2 儲存具有相同的可靠性和耐久性特性，如Amazon Aurora 儲存體所述。這是因為無論運算容量使用 Aurora Serverless v2 或已佈建，Aurora 資料庫叢集的儲存運作方式都相同。

Aurora 叢集的組態參數

您可以為具有 Aurora Serverless v2 容量的叢集調整所有叢集和資料庫組態參數，方法與已佈建資料庫叢集相同。但是，對於 Aurora Serverless v2，一些與容量相關之參數的處理方式不同。在混合組態叢集中，您為這些與容量相關參數指定的參數值仍會套用至所有已佈建的寫入器和讀取器。

對於 Aurora Serverless v2 寫入器和讀取器，幾乎所有參數的運作方式都與已佈建項目相同。例外狀況是 Aurora 在擴展期間中自動調整的一些參數，及 Aurora 將某些參數保持在取決於最大容量設定的固定值。

例如，為緩衝區快取保留的記憶體量隨著寫入器或讀取器的擴展而增加，並隨著縮小而減少。如此，可在資料庫不忙時釋放記憶體。反之，Aurora 會根據最大容量設定，自動將最大連接數設定為適當的值。這樣，如果負載下降且 Aurora Serverless v2 縮減規模，則作用中連線不會遭到中斷。如需 Aurora Serverless v2 如何處理特定參數的資訊，請參閱 使用 Aurora Serverless v2 的參數群組。

Aurora Serverless v2 的要求與限制

在建立打算使用 Aurora Serverless v2 資料庫執行個體的叢集時，請注意下列要求和限制。

主題

- 區域和版本可用性
- 使用 Aurora Serverless v2 的叢集必須指定容量範圍
- Aurora Serverless v2 中不支援某些佈建功能
- 某些 Aurora Serverless v2 方面與 Aurora Serverless v1 不同

區域和版本可用性

功能可用性和支援會因每個 Aurora 資料庫引擎的特定版本以及 AWS 區域 而有所不同。如需 Aurora 和 Aurora Serverless v2 版本和區域可用性的詳細資訊，請參閱 Aurora Serverless v2 支援的區域和 Aurora 資料庫引擎。

下列範例顯示 AWS CLI 命令,以確認您可以使用 Aurora Serverless v2 用於特定 AWS 區域 的確切引擎值。Aurora Serverless v2 的 `--db-instance-class` 參數一律為 `db.serverless`。`--engine` 參數可為 `aurora-mysql` 或 `aurora-postgresql`。替換適當的`--region` 和 `--engine` 值，以確認您可使用 `--engine-version` 值。若該命令不會產生任何輸出，則 AWS 區域 和資料庫引擎的組合無法使用 Aurora Serverless v2。

```
aws rds describe-orderable-db-instance-options --engine aurora-mysql --db-instance-
class db.serverless \
  --region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output
 text

aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.serverless \
  --region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output
 text
```

使用 Aurora Serverless v2 的叢集必須指定容量範圍

Aurora 叢集必須具有 `ServerlessV2ScalingConfiguration` 屬性，然後才能新增使用 `db.serverless` 資料庫執行個體類別的資料庫執行個體。此屬性指定容量範圍。Aurora Serverless v2 容量範圍從最低 0 個 Aurora 容量單位 (ACU) 到最大 256 個 ACU，增量為 0.5 ACU。允許的最小值取決於 Aurora 版本。每個 ACU 皆提供相當於大約 2 GB (GiB) 的 RAM 以及相關聯的 CPU 和聯網。如需 Aurora Serverless v2 使用容量範圍設定的詳細資料，請參閱 Aurora Serverless v2 的運作方式。

如需各種資料庫引擎版本和平台版本的允許容量範圍，請參閱 Aurora Serverless v2 容量。特定叢集的可用擴展範圍受引擎版本和硬體 (平台版本) 影響。

當您建立叢集和相關聯 Aurora Serverless v2 資料庫執行個體時，可於 AWS 管理主控台 中指定最小及最大 ACU 值。您還可指定 AWS CLI 中的 `--serverless-v2-scaling-configuration` 選項。或者您可搭配 Amazon RDS API 指定 `ServerlessV2ScalingConfiguration` 參數。建立叢集或修改現有叢集時，您可指定此屬性。有關設定容量範圍的程序，請參閱 設定叢集的 Aurora Serverless v2 容量範圍。有關如何選擇最小和最大容量值及這些設定會如何影響某些資料庫參數的詳細討論，請參閱 選擇 Aurora 叢集的 Aurora Serverless v2 容量範圍。

Aurora Serverless v2 中不支援某些佈建功能

Aurora 佈建資料庫執行個體的下列功能目前不適用於 Amazon Aurora Serverless v2：

- 資料庫活動串流 (DAS)。
- Aurora PostgreSQL 的叢集快取管理。`apg_ccm_enabled` 組態參數不適用於 Aurora Serverless v2 資料庫執行個體。

有些 Aurora 功能搭配 Aurora Serverless v2 使用，但若容量範圍低於具有特定工作負載之這些功能的記憶體需求，則可能會造成問題。於該種情況下，您的資料庫可能無法像往常一樣執行，或者可能會遇到記憶體不足錯誤。有關設定適當容量範圍的建議，請參閱 選擇 Aurora 叢集的 Aurora Serverless v2

容量範圍。有關資料庫因容量範圍錯誤設定而造成記憶體不足錯誤時的疑難排解資訊，請參閱 避免記憶體不足錯誤。

不支援 Aurora Auto Scaling。此擴展類型會新增新的讀取器，來處理額外的讀取密集型工作負載 (以 CPU 用量為基礎)。但是，以 CPU 用量為基礎的擴展對 Aurora Serverless v2 不具意義。或者，您可預先建立 Aurora Serverless v2 讀取器資料庫執行個體，並使其縮小至低容量。與動態新增新資料庫執行個體相比，這是一種擴展叢集讀取容量更快、破壞性更低的方法。

某些 Aurora Serverless v2 方面與 Aurora Serverless v1 不同

若您是 Aurora Serverless v1 使用者，而這是您首次使用 Aurora Serverless v2，請參閱Aurora Serverless v2 和 Aurora Serverless v1 要求之間的差異，以了解 Aurora Serverless v1 和 Aurora Serverless v2 之間的要求有何不同。

建立使用 Aurora Serverless v2 的資料庫叢集

若要建立可新增 Aurora Serverless v2 資料庫執行個體的 Aurora 叢集，請遵循建立 Amazon Aurora 資料庫叢集中的相同程序。使用 Aurora Serverless v2，您的叢集可以與已佈建叢集互換。您的叢集中某些資料庫執行個體可以使用 Aurora Serverless v2，而某些資料庫執行個體可以已佈建。

主題

- Aurora Serverless v2 資料庫叢集的設定
- 建立 Aurora Serverless v2 資料庫叢集
- 建立 Aurora Serverless v2 寫入器資料庫執行個體

Aurora Serverless v2 資料庫叢集的設定

請確定叢集的初始設定符合 Aurora Serverless v2 的要求與限制 中列出的要求。指定下列設定，以確定您可以新增 Aurora Serverless v2 資料庫執行個體到叢集：

AWS 區域

在可使用 Aurora Serverless v2 資料庫執行個體的 AWS 區域中建立叢集。如需區域的詳細資訊，請參閱Aurora Serverless v2 支援的區域和 Aurora 資料庫引擎。

DB engine version (資料庫引擎版本)

選擇與 Aurora Serverless v2 相容的引擎版本。如需 Aurora Serverless v2 版本需求的相關資訊，請參閱Aurora Serverless v2 的要求與限制。

資料庫執行個體類別

如果使用AWS 管理主控台建立叢集，您可以同時為寫入器資料庫執行個體選擇資料庫執行個體類別。選擇 Serverless (無伺服器) 資料庫執行個體類別。選擇該資料庫執行個體類別時，還可以指定寫入器資料庫執行個體的容量範圍。該容量範圍適用於您新增至該叢集的所有其他 Aurora Serverless v2 資料庫執行個體。

如果您看不到資料庫執行個體類別的無伺服器選項，請確保您選擇了 Aurora Serverless v2 支援的區域和 Aurora 資料庫引擎 支援的資料庫引擎版本。

當您使用 AWS CLI 或 Amazon RDS API，則為資料庫執行個體類別指定的參數為 `db.serverless`。

容量範圍

填寫套用至叢集中所有資料庫執行個體的最小和最大 Aurora 容量單位 (ACU) 值。當您為資料庫執行個體類別選擇 Serverless (無伺服器) 時，此選項可在Create cluster (建立叢集) 和 Add reader (新增讀取器) 主控台頁面中使用。

如需各種資料庫引擎版本的允許容量範圍，請參閱 Aurora Serverless v2 容量。

如果您沒有看到最小和最大 ACU 欄位，請確定您為寫入器資料庫執行個體選擇了無伺服器資料庫執行個體類別。

如果您最初使用已佈建的資料庫執行個體建立叢集，則不指定最小和最大 ACU。在這種情況下，您可以隨後修改叢集以新增該設定。您也可以新增 Aurora Serverless v2 讀取器資料庫執行個體到叢集中。您可將容量範圍指定為該程序的一部分。

在為叢集指定容量範圍之前，您無法使用 AWS CLI 或 RDS API 新增任何 Aurora Serverless v2 資料庫執行個體到叢集中。如果您嘗試新增 Aurora Serverless v2 資料庫執行個體，您會收到錯誤訊息。在 AWS CLI 或 RDS API 程序中，容量範圍由 `ServerlessV2ScalingConfiguration` 屬性表示。

對於包含多個讀取器資料庫執行個體的叢集，每個 Aurora Serverless v2 讀取器資料庫執行個體的容錯移轉優先順序在該資料庫執行個體如何擴充或縮減規模方面發揮重要作用。您無法在初始建立叢集時指定優先順序。向叢集新增第二個或更多讀取器資料庫執行個體時，請記住此屬性。如需更多詳細資訊，請參閱 選擇 Aurora Serverless v2 讀取器的提升層。

建立 Aurora Serverless v2 資料庫叢集

您可以使用 AWS 管理主控台、AWS CLI 或 RDS API 建立 Aurora Serverless v2 資料庫叢集。

主控台

使用 Aurora Serverless v2 寫入器建立叢集

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇 Create database (建立資料庫)。在顯示的頁面上，選擇下列選項：
   - 針對引擎類型，請選擇 Aurora (MySQL 相容) 或 Aurora (PostgreSQL 相容)。
   - 對於版本，選擇 Aurora Serverless v2 支援的區域和 Aurora 資料庫引擎 支援的其中一個版本。
4. 對於資料庫執行個體類別，選擇 Serverless v2。
5. 對於容量範圍，您可以接受預設範圍。或者，您可以在最小和最大容量單位選擇其他值。您可以從最小 0 個 ACU 到最大 256 個 ACU 中進行選擇，以 0.5 ACU 為單位增加。

   如需 Aurora Serverless v2 容量單位的詳細資訊，請參閱Aurora Serverless v2 容量和Aurora Serverless v2 的效能和擴展。

   視您選擇的引擎和版本而定，上限可能是 128 個 ACU、下限可能是 0.5 個 ACU，或兩者皆是。如需每個 Aurora 引擎和版本組合限制的詳細資訊，請參閱 Aurora Serverless v2 容量。



   選擇最小容量為 0 個 ACU 會啟用 Aurora Serverless v2 自動暫停和繼續功能。在這種情況下，您可以額外選擇 Aurora Serverless v2 資料庫執行個體在沒有資料庫連線的情況下等待一段期間，然後再自動暫停。如需自動暫停和繼續功能的相關資訊，請參閱 使用 Aurora Serverless v2 的自動暫停和繼續擴展至零個 ACU。



6. 選擇任何其他資料庫叢集設定，如 Aurora 資料庫叢集的設定 中所述。
7. 選擇建立資料庫 建立 Aurora 資料庫叢集，並使用 Aurora Serverless v2 資料庫執行個體作為寫入器執行個體 (也稱為主要資料庫執行個體)。

CLI

若要使用 AWS CLI 建立與 Aurora Serverless v2 資料庫執行個體相容的資料庫叢集，請按照建立 Amazon Aurora 資料庫叢集中的 CLI 程序進行。在 `create-db-cluster` 命令中包含以下參數：

- --region *AWS_Region_where_Aurora Serverless v2_instances_are_available*
- --engine-version *serverless_v2_compatible_engine_version*
- --serverless-v2-scaling-configuration MinCapacity=*minimum_capacity*,MaxCapacity=*maximum_capacity*

下列範例顯示 Aurora Serverless v2 資料庫叢集的建立。

```
aws rds create-db-cluster \
    --db-cluster-identifier my-serverless-v2-cluster \
    --region eu-central-1 \
    --engine aurora-mysql \
    --engine-version 8.0.mysql_aurora.3.04.1 \
    --serverless-v2-scaling-configuration MinCapacity=1,MaxCapacity=4 \
    --master-username myuser \
    --manage-master-user-password
```

> **Note**
>
> 使用 AWS CLI 建立 Aurora Serverless v2 資料庫叢集時，引擎模式會在輸出中顯示為 `provisioned` 而不是 `serverless`。`serverless` 引擎模式是指 Aurora Serverless v1。

此範例會指定 `--manage-master-user-password` 選項來產生管理密碼，並在 Secrets Manager 中管理該密碼。如需更多詳細資訊，請參閱 使用 Amazon Aurora 和 進行密碼管理 AWS Secrets Manager。或者，您可以使用 `--master-password` 選項，自行指定和管理密碼。

如需 Aurora Serverless v2 版本需求的相關資訊，請參閱Aurora Serverless v2 的要求與限制。如需容量範圍允許的數字以及這些數字表示意義的相關資訊，請參閱Aurora Serverless v2 容量和Aurora Serverless v2 的效能和擴展。

若要檢查現有叢集是否具有指定的容量設定，請檢查針對 `ServerlessV2ScalingConfiguration` 屬性的 `describe-db-clusters` 命令的輸出。該屬性看起來類似下列內容。

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": 1.5,
    "MaxCapacity": 24.0
}
```

> **Tip**
>
> 如果您在建立叢集時未指定最小和最大 ACU，稍後可以使用 `modify-db-cluster` 命令新增該設定。在您這樣做之前，您無法新增任何 Aurora Serverless v2 資料庫執行個體至叢集。如果您嘗試新增 `db.serverless` 資料庫執行個體，您會收到錯誤訊息。

API

若要使用 RDS API 建立與 Aurora Serverless v2 資料庫執行個體相容的資料庫叢集，請按照建立 Amazon Aurora 資料庫叢集中的 API 程序進行。選擇下列設定。確保您的 `CreateDBCluster` 操作包含下列參數：

```
EngineVersion serverless_v2_compatible_engine_version
ServerlessV2ScalingConfiguration with MinCapacity=minimum_capacity and
 MaxCapacity=maximum_capacity
```

如需 Aurora Serverless v2 版本需求的相關資訊，請參閱Aurora Serverless v2 的要求與限制。如需容量範圍允許的數字以及這些數字表示意義的相關資訊，請參閱Aurora Serverless v2 容量和Aurora Serverless v2 的效能和擴展。

若要檢查現有叢集是否具有指定的容量設定，請檢查 `DescribeDBClusters` 操作的輸出以了解 `ServerlessV2ScalingConfiguration` 屬性。該屬性看起來類似下列內容。

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": 1.5,
    "MaxCapacity": 24.0
}
```

> **Tip**
>
> 如果您在建立叢集時未指定最小和最大 ACU，稍後可以使用 `ModifyDBCluster` 操作新增該設定。在您這樣做之前，您無法新增任何 Aurora Serverless v2 資料庫執行個體至叢集。如果您嘗試新增 `db.serverless` 資料庫執行個體，您會收到錯誤訊息。

建立 Aurora Serverless v2 寫入器資料庫執行個體

雖然您在建立 Aurora 叢集時指定 Aurora Serverless v2 容量範圍，但您可以選擇是否要為叢集中的每個資料庫執行個體使用 Aurora Serverless v2 或佈建。若要立即在您的資料庫叢集中開始使用 Aurora Serverless v2，請新增使用執行個體類別的寫入器資料庫 `db.serverless` 執行個體。在主控台中，您通常會在建立資料庫叢集的工作流程中做出此選擇。因此，當您透過 CLI 執行設定時，此程序大多適用。

主控台

使用AWS 管理主控台建立資料庫叢集時，您可以同時指定寫入器資料庫執行個體的屬性。若要讓寫入器資料庫執行個體使用 Aurora Serverless v2，請選擇 Serverless (無伺服器) 資料庫執行個體類別。

然後，透過指定最小和最大 Aurora 容量單位 (ACU) 值來設定叢集的容量範圍。這些最小和最大值適用於叢集中的每個 Aurora Serverless v2 資料庫執行個體。如需該程序以及最小和最大容量設定的重要性，請參閱 建立 Aurora Serverless v2 資料庫叢集。

如果您首次建立叢集時未建立 Aurora Serverless v2 資料庫執行個體，您可以稍後新增一個或多個 Aurora Serverless v2 資料庫執行個體。若要執行此作業，請依照新增 Aurora Serverless v2 讀取器和將已佈建的寫入器或讀取器轉換為 Aurora Serverless v2中的程序進行。您可以在新增第一個 Aurora Serverless v2 資料庫執行個體到叢集時指定容量範圍。您可以稍後變更容量範圍，方法是遵循設定叢集的 Aurora Serverless v2 容量範圍中的程序。

CLI

使用 AWS CLI 建立 Aurora Serverless v2 資料庫叢集時，您使用 create-db-instance 命令明確地新增寫入器資料庫執行個體。請納入下列參數：

- `--db-instance-class db.serverless`

下列範例顯示建立 Aurora Serverless v2 寫入器資料庫執行個體。

```
aws rds create-db-instance \
    --db-cluster-identifier my-serverless-v2-cluster \
    --db-instance-identifier my-serverless-v2-instance \
    --db-instance-class db.serverless \
    --engine aurora-mysql
```

管理 Aurora Serverless v2 資料庫叢集

使用 Aurora Serverless v2，您的叢集可以與已佈建叢集互換。Aurora Serverless v2 屬性會套用至資料庫叢集中的一或多個資料庫執行個體。因此，建立叢集、修改叢集、建立和恢復快照等程序基本上與其他類型的 Aurora 叢集相同。如需管理 Aurora 叢集和資料庫執行個體的一般程序，請參閱管理 Amazon Aurora 資料庫叢集。

在下列主題中，您可以學習包含 Aurora Serverless v2 資料庫執行個體之叢集的管理考量事項。

主題

- 設定叢集的 Aurora Serverless v2 容量範圍
- 檢查 Aurora Serverless v2 的容量範圍
- 新增 Aurora Serverless v2 讀取器
- 將已佈建的寫入器或讀取器轉換為 Aurora Serverless v2
- 將 Aurora Serverless v2 寫入器或讀取器轉換為已佈建
- 暫停 Aurora Serverless v2 資料庫執行個體
- 選擇 Aurora Serverless v2 讀取器的提升層
- 搭配 Aurora Serverless v2 使用 TLS/SSL
- 檢視 Aurora Serverless v2 寫入器和讀取器
- 為 Aurora Serverless v2 記錄日誌。

設定叢集的 Aurora Serverless v2 容量範圍

若要修改包含 Aurora Serverless v2 資料庫執行個體之叢集的組態參數或其他設定或是資料庫執行個體本身，請遵循與已佈建叢集相同的一般程序。如需詳細資訊，請參閱修改 Amazon Aurora 資料庫叢集。

Aurora Serverless v2 獨有的最重要設定是容量範圍。設定 Aurora 叢集的最小和最大 Aurora 容量單位 (ACU) 值後，您無需主動調整叢集中的 Aurora Serverless v2 資料庫執行個體。Aurora 會為您處理這些工作。此設定是在叢集層級進行管理。相同的最小和最大 ACU 值適用於叢集中的每個 Aurora Serverless v2 資料庫執行個體。

您可以設定下列指定值：

- Minimum ACUs (最小 ACU) – Aurora Serverless v2 資料庫執行個體可以將容量減少到此 ACU 數量。
- Maximum ACUs (最大 ACU) – Aurora Serverless v2 資料庫執行個體可以將容量增加到此 ACU 數量。

Aurora Serverless v2 的可用容量範圍取決於資料庫引擎版本和平台版本。較新的資料庫引擎版本允許最大容量為 256 個 ACU、最小容量為 0 個 ACU，或兩者都允許。如需各種資料庫引擎版本的容量範圍，請參閱 Aurora Serverless v2 容量。

如需透過將最小容量設定為 0 個 ACU 啟用的自動暫停和繼續功能，請參閱 使用 Aurora Serverless v2 的自動暫停和繼續擴展至零個 ACU。

如需確保 Aurora Serverless v2 資料庫叢集可向上擴展至 256 個 ACU 的詳細資訊，請參閱 升級 Aurora Serverless v2 資料庫叢集以允許擴展至 256 個 ACU。

> **Note**
>
> 當您修改 Aurora Serverless v2 資料庫叢集的容量範圍時，無論您選擇立即套用或在下一個排程維護時段套用變更，該變更都會立即發生。

在 Aurora Serverless v2 中，您無法在不修改容量範圍的情況下直接設定目前的容量，因為容量會根據指定範圍內的工作負載動態調整。不過，您可以透過下列方式間接影響容量：

- 調整最小容量 – 暫時降低最小容量，以在工作負載較輕時減少基準容量。
- 暫時暫停擴展 – 若要修正特定值的容量，請將容量下限和上限設定為相同層級。
- 針對高載用量主動擴展 – 如果您預期工作負載高載，請事先增加最小容量，以在高活動期間維持更高的基準。

如需容量範圍的影響以及如何監控和微調其值的詳細資訊，請參閱Aurora Serverless v2 的重要 Amazon CloudWatch 指標和Aurora Serverless v2 的效能和擴展。您的目標是確保叢集的最大容量足以處理尖峰工作負載，並且最小容量足夠低，可在叢集不忙時將成本降至最低。

假設您根據監控，決定叢集的 ACU 範圍應該更高、更低、更寬或更窄。您可以使用 AWS 管理主控台、 AWS CLI或 Amazon RDS API，將 Aurora 叢集的容量設定為特定範圍的 ACUs。此容量範圍適用於叢集中的每個 Aurora Serverless v2 資料庫執行個體。

例如，假設您的叢集容量範圍為 1–16 個 ACU，並且包含兩個 Aurora Serverless v2 資料庫執行個體。則叢集作為整體消耗，將介於 2 個 ACU (閒置時) 和 32 個 ACU (充分利用) 之間。如果您將容量範圍從 8 變更為 40.5 個 ACU，則現在整個叢集在閒置時會消耗 16 個 ACU，充分利用時最多消耗 81 個 ACU。

Aurora 會自動將 Aurora Serverless v2 資料庫執行個體的某些參數值，設定為取決於容量範圍中的最大 ACU 值。如需這類參數的清單，請參閱Aurora Serverless v2 的連線數上限。對於依賴於此類計算的靜態參數，重新開機資料庫執行個體時會再次評估該值。因此，您可以在變更容量範圍後重新開機資料庫執行個體，來更新此類參數的值。若要檢查您是否需要重新開機資料庫執行個體來獲取此類參數變更，請檢查資料庫執行個體的 `ParameterApplyStatus` 屬性。值為 `pending-reboot` 表示重新開機會對某些參數值套用變更。

主控台

您可使用 AWS 管理主控台設定包含 Aurora Serverless v2 資料庫執行個體之叢集的容量範圍。

當您使用主控台時，您可以在新增第一個 Aurora Serverless v2 資料庫執行個體至該叢集時，設定叢集的容量範圍。您在建立叢集時為寫入器資料庫執行個體選擇 Serverless v2 (無伺服器 v2) 資料庫執行個體類別時，也能這麼做。或者，您在新增 Aurora Serverless v2 讀取器資料庫執行個體至叢集時選擇 Serverless (無伺服器) 資料庫執行個體類別時，也能這麼做。或者，您可以在將叢集中的現有已佈建資料庫執行個體轉換為 Serverless (無伺服器) 資料庫執行個體類別時這樣做。如需這些程序的完整版本，請參閱 建立 Aurora Serverless v2 寫入器資料庫執行個體、新增 Aurora Serverless v2 讀取器，以及 將已佈建的寫入器或讀取器轉換為 Aurora Serverless v2

您在叢集層級設定的容量範圍適用於叢集中的所有 Aurora Serverless v2 資料庫執行個體。下圖顯示具有多個 Aurora Serverless v2 讀取器資料庫執行個體的叢集。每個的容量範圍相同，均為 2–64 個 ACU。



修改 Aurora Serverless v2 叢集的容量範圍

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 從清單選擇包含您的 Aurora Serverless v2 資料庫執行個體的叢集。叢集必須至少包含一個 Aurora Serverless v2 資料庫執行個體。否則，Aurora 不會顯示 Capacity range (容量範圍) 區段。
4. 在 Actions (動作) 中，選擇 Modify (修改)。
5. 在 Capacity range (容量範圍) 區段中，選擇下列選項：

    a. 輸入 Minimum ACUs (最小 ACU) 的值。主控台顯示允許的值範圍。您可以選擇從 0 到 256 個 ACU 的最小容量。您可以選擇從 1 到 256 個 ACU 的最大容量。您可以調整容量值，以 0.5 ACU 為單位增加。可用的容量範圍取決於您的資料庫引擎版本和平台版本。

b. 輸入 Maximum ACUs (最大 ACU) 的值。此數值必須等於或大於 Minimum ACUs。主控台顯示允許的值範圍。下圖顯示了該選擇。



6. 選擇 Continue (繼續)。此時系統會顯示 Summary of modifications (修改摘要) 頁面。
7. 選擇 Apply immediately (立即套用)。

   無論您選擇立即套用或在下一個排程維護時段套用容量修改，該容量修改都會立即發生。
8. 選擇 Modify cluster (修改叢集) 以接受修改摘要。您也可以選擇 Back (上一步) 修改變更，或 Cancel (取消) 捨棄變更。

AWS CLI

若要設定您要使用 Aurora Serverless v2 資料庫執行個體的叢集容量 AWS CLI，請執行 modify-db-cluster AWS CLI 命令。指定 `--serverless-v2-scaling-configuration` 選項。叢集可能已包含一個或多個 Aurora Serverless v2 資料庫執行個體，您也可以稍後新增資料庫執行個體。`MinCapacity` 和 `MaxCapacity` 欄位的有效值包括下列項目：

- `0`、`0.5`、`1`、`1.5`、`2` 等，間隔為 0.5，最大可達 256。最小 ACU 值 0 僅適用於支援自動暫停功能的 Aurora 版本。

最大可用容量取決於您的資料庫引擎版本和平台版本。

在此範例中，您將名為 `sample-cluster` 之 Aurora 資料庫叢集的 ACU 範圍設定為最小 `48.5`，最大 64。

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
  --serverless-v2-scaling-configuration MinCapacity=48.5,MaxCapacity=64
```

無論您選擇立即套用或在下一個排程維護時段套用容量修改，該容量修改都會立即發生。

這樣做之後，您可以新增 Aurora Serverless v2 資料庫執行個體到叢集，每個新資料庫執行個體都可以在 48.5 和 64 個 ACU 之間擴展。新的容量範圍也適用於任何已在叢集中的 Aurora Serverless v2 資料庫執行個體。如有必要，資料庫執行個體會擴充或縮減規模以符合新的容量範圍。

如需使用 CLI 設定容量範圍的其他範例，請參閱選擇 Aurora 叢集的 Aurora Serverless v2 容量範圍。

若要使用 Aurora Serverless 修改資料庫叢集的擴展組態 AWS CLI，請執行 modify-db-cluster AWS CLI 命令。指定 `--serverless-v2-scaling-configuration` 選項以設定最小容量和最大容量。有效容量值包括：

- Aurora MySQL：`0`、`0.5`、`1`、`1.5`、`2` 等，以 0.5 ACU 為單位增加，最大可達 256。
- Aurora PostgreSQL：`0`、`0.5`、`1`、`1.5`、`2` 等，以 0.5 ACU 為單位增加，最大可達 256。
- 最小 ACU 值 0 僅適用於支援自動暫停功能的 Aurora 版本。

在下列範例中，您會修改名為 `sample-instance` 之 Aurora Serverless v2 資料庫執行個體 (其屬於名為 `sample-cluster` 之叢集的一部份) 的擴展組態。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

在 Windows 中：

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster ^
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

RDS API

您可使用 ModifyDBCluster API 操作設定 Aurora 資料庫執行個體的容量。指定 `ServerlessV2ScalingConfiguration` 參數。`MinCapacity` 和 `MaxCapacity` 欄位的有效值包括下列項目：

- `0`、`0.5`、`1`、`1.5`、`2` 等，間隔為 0.5，最大可達 256。最小 ACU 值 0 僅適用於支援自動暫停功能的 Aurora 版本。

您可以使用 ModifyDBCluster API 操作來修改包含 Aurora Serverless v2 資料庫執行個體之叢集的擴展組態。指定 `ServerlessV2ScalingConfiguration` 參數以設定最小容量和最大容量。有效容量值包括：

- Aurora MySQL：0、0.5、1、1.5、2 等，以 0.5 ACU 為單位增加，最大可達 256。
- Aurora PostgreSQL：0、0.5、1、1.5、2 等，以 0.5 ACU 為單位增加，最大可達 256。
- 最小 ACU 值 0 僅適用於支援自動暫停功能的 Aurora 版本。

最大可用容量取決於您的資料庫引擎版本和平台版本。

無論您選擇立即套用或在下一個排程維護時段套用容量修改，該容量修改都會立即發生。

升級 Aurora Serverless v2 資料庫叢集以允許擴展至 256 個 ACU

在某些情況下，當您嘗試將 Aurora Serverless v2 資料庫叢集擴展到大於 128 個 ACU 容量時，您會收到錯誤訊息。錯誤訊息會告訴您哪些執行個體與新的擴展範圍不相容。這強調了容量範圍、資料庫引擎版本和平台版本之間的重要關係。

無法擴展大於 128 個 ACU 的原因可能有兩個：

- 較舊的資料庫引擎版本 – 將資料庫引擎版本升級至支援 256 個 ACU 的版本。如需詳細資訊，請參閱Aurora Serverless v2 容量。
- 較舊的平台版本 – 升級 Aurora Serverless v2 資料庫叢集的平台。您可採用下列其中一種方式來這麼做：
  - 停止並重新啟動資料庫叢集。當叢集重新啟動時，它將位於最新的平台版本上，該版本可能具有更高的 ACU 上限。

    不過，停止和啟動資料庫叢集會產生一些停機時間，通常是幾分鐘。如需詳細資訊，請參閱停用和啟動 Amazon Aurora 資料庫叢集。
  - 使用藍/綠部署 如需詳細資訊，請參閱Amazon Aurora 藍/綠部署概觀。
    1. 建立藍/綠部署 如需詳細資訊，請參閱在 Amazon Aurora 中建立藍/綠部署。
    2. 確認您可以將預備 (綠色) 環境的最大容量設定為 256 個 ACU。
    3. 切換到綠色環境。如需詳細資訊，請參閱在 Amazon Aurora 中切換藍/綠部署。
    4. 刪除原始資料庫叢集。

> **Note**
>
> 如果您在 中維護資料庫登入資料 AWS Secrets Manager，則無法使用藍/綠部署。
> Aurora 全球資料庫不支援藍/綠部署。

檢查 Aurora Serverless v2 的容量範圍

檢查 Aurora Serverless v2 叢集容量範圍的程序需要您先設定容量範圍。如果您尚未執行此作業，請遵循設定叢集的 Aurora Serverless v2 容量範圍中的程序。

您在叢集層級設定的容量範圍適用於叢集中的所有 Aurora Serverless v2 資料庫執行個體。下圖顯示具有多個 Aurora Serverless v2 資料庫執行個體的叢集。每個都具有相同的容量範圍。



您也可以檢視叢集中任何 Aurora Serverless v2 資料庫執行個體的詳細資訊頁面。資料庫執行個體的容量範圍顯示在 Configuration (組態) 索引標籤。



您還可以在叢集的 Modify (修改) 頁面上，查看叢集的目前容量範圍。此時，您可以變更容量範圍。如需設定或變更容量範圍的所有方法，請參閱設定叢集的 Aurora Serverless v2 容量範圍。

檢查 Aurora 叢集的目前容量範圍

您可以透過檢查叢集的 `ServerlessV2ScalingConfiguration` 屬性，來檢查為叢集中的 Aurora Serverless v2 資料庫執行個體設定的容量範圍。以下 AWS CLI 範例顯示的叢集最小容量為 0.5 個 Aurora 容量單位 (ACU)，最大容量為 16 個 ACU。

```
$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-64-acu-cluster \
  --query 'DBClusters[*].[ServerlessV2ScalingConfiguration]'
[
    [
```

```
        {
            "MinCapacity": 0.5,
            "MaxCapacity": 16.0
        }
    ]
]
```

新增 Aurora Serverless v2 讀取器

若要新增 Aurora Serverless v2 讀取器資料庫執行個體到您的叢集，您可以按照將 Aurora 複本新增至資料庫叢集中相同的程序進行。為新的資料庫執行個體選擇 Serverless v2 (無伺服器 v2) 執行個體類別。

如果讀取器資料庫執行個體是叢集中的第一個 Aurora Serverless v2 資料庫執行個體，您還可以選擇容量範圍。此設定適用於此讀取器資料庫執行個體和您新增至叢集的任何其他 Aurora Serverless v2 資料庫執行個體。每個 Aurora Serverless v2 資料庫執行個體都能在最小與最大 ACU 值之間進行擴展。

如果叢集中已存在任何其他 Aurora Serverless v2 執行個體，新增讀取器對話方塊會顯示叢集目前的容量範圍。在這種情況下，您無法變更容量。下圖顯示目前叢集容量的報告。



如果您已經新增了任何 Aurora Serverless v2 資料庫執行個體到叢集中，新增另一個 Aurora Serverless v2 讀取器資料庫執行個體會顯示目前的容量範圍。下圖顯示了這些唯讀控制項。



如果您想變更叢集的容量範圍，請遵循設定叢集的 Aurora Serverless v2 容量範圍中的程序。

對於包含多個讀取器資料庫執行個體的叢集，每個 Aurora Serverless v2 讀取器資料庫執行個體的容錯移轉優先順序在該資料庫執行個體如何擴充或縮減規模方面發揮重要作用。您無法在初始建立叢集時指定優先順序。向叢集新增第二個讀取器或其他資料庫執行個體時，請記住此屬性。如需詳細資訊，請參閱選擇 Aurora Serverless v2 讀取器的提升層。

如需查看叢集目前容量範圍的其他方法，請參閱檢查 Aurora Serverless v2 的容量範圍。

將已佈建的寫入器或讀取器轉換為 Aurora Serverless v2

您可以將已佈建的資料庫執行個體轉換為使用 Aurora Serverless v2。若要執行此作業，請依照修改資料庫叢集中的資料庫執行個體中的程序進行。叢集必須符合Aurora Serverless v2 的要求與限制中的要求。例如：Aurora Serverless v2 資料庫執行個體要求叢集執行某些最低引擎版本。

假設您正在轉換執行中的已佈建叢集以便運用 Aurora Serverless v2。在這種情況下，您可以在切換程序的第一個步驟將資料庫執行個體轉換為 Aurora Serverless v2，將停機時間降至最低。如需完整程序，請參閱從已佈建叢集切換至 Aurora Serverless v2。

如果您轉換的資料庫執行個體是叢集中的第一個 Aurora Serverless v2 資料庫執行個體，您可以在 Modify (修改) 操作中選擇叢集的容量範圍。此容量範圍適用於您新增至叢集的每個 Aurora Serverless v2 資料庫執行個體。下圖顯示您在其中指定最小和最大 Aurora 容量單位 (ACU) 的頁面。



如需容量範圍意義的詳細資訊，請參閱Aurora Serverless v2 容量。

如果叢集已包含一或多個 Aurora Serverless v2 資料庫執行個體時，您可以在 Modify (修改) 操作期間查看現有的容量範圍。下圖顯示該資訊面板的範例。



如果您希望在新增更多 Aurora Serverless v2 資料庫執行個體之後變更叢集的容量範圍，請遵循設定叢集的 Aurora Serverless v2 容量範圍中的程序。

將 Aurora Serverless v2 寫入器或讀取器轉換為已佈建

您可以將 Aurora Serverless v2 資料庫執行個體轉換為已佈建的資料庫執行個體。若要執行此作業，請依照修改資料庫叢集中的資料庫執行個體中的程序進行。選擇 Serverless (無伺服器) 以外的資料庫執行個體類別。

如果 Aurora Serverless v2 資料庫執行個體需要比 Aurora Serverless v2 資料庫執行個體的最大 Aurora 容量單位 (ACU) 更大的容量，您可以將其轉換為已佈建。例如，最大的 db.r5 和 db.r6g 資料庫執行個體類別具有比 Aurora Serverless v2 資料庫執行個體擴展上限更大的記憶體容量。

> **Tip**
>
> 某些較舊的資料庫執行個體類別 (如 db.r3 和 db.t2) 不適用於您搭配 Aurora Serverless v2 使用的 Aurora 版本。若要查看在將 Aurora Serverless v2 資料庫執行個體轉換為已佈建的執行個體時可以使用哪些資料庫執行個體類別，請參閱資料庫執行個體類別的支援資料庫引擎。

如果您要將叢集的寫入器資料庫執行個體從 Aurora Serverless v2 轉換為已佈建，則可以按照從已佈建叢集切換至 Aurora Serverless v2中的程序進行，但反過來。將叢集中的其中一個讀取器資料庫執行個體從 Aurora Serverless v2 切換為已佈建。然後執行容錯移轉，以將該已佈建的資料庫執行個體進入寫入器。

您之前為叢集指定的任何容量範圍會持續存在，即使從叢集中移除所有 Aurora Serverless v2 資料庫執行個體也一樣。如果您想要變更容量範圍，您可以依據設定叢集的 Aurora Serverless v2 容量範圍中的說明修改叢集。

暫停 Aurora Serverless v2 資料庫執行個體

您可以設定 Aurora 叢集，以自動暫停和繼續其 Aurora Serverless v2 資料庫執行個體。如需詳細資訊，請參閱使用 Aurora Serverless v2 的自動暫停和繼續擴展至零個 ACU。

選擇 Aurora Serverless v2 讀取器的提升層

對於包含多個 Aurora Serverless v2 資料庫執行個體或混合已佈建和 Aurora Serverless v2 資料庫執行個體的叢集，請注意每個 Aurora Serverless v2 資料庫執行個體的提升層。此設定控制 Aurora Serverless v2 資料庫執行個體的更多行為，而非已佈建的資料庫執行個體。

在 中 AWS 管理主控台，您可以使用建立資料庫、修改執行個體和新增讀取器頁面的其他組態下的容錯移轉優先順序選擇來指定此設定。您可以在 Databases (資料庫) 頁面的選用 Priority tier (優先順序方案) 欄中看到現有資料庫執行個體的這個屬性。您也可以在資料庫叢集或資料庫執行個體的詳細資訊頁面上查看此屬性。

對於已佈建的資料庫執行個體，選擇第 0–15 層僅決定 Aurora 在容錯移轉操作期間選擇將讀取器資料庫執行個體提升為寫入器的順序。對於 Aurora Serverless v2 讀取器資料庫執行個體，層數也會決定資料庫執行個體是否擴充規模以符合寫入器資料庫執行個體的容量，還是根據其自身的工作負載獨立擴展。第 0 層或第 1 層中的 Aurora Serverless v2 讀取器資料庫執行個體會保持在至少與寫入器資料庫執行個體一樣高的最小容量。如此一來，它們可以在發生容錯移轉時接管寫入器資料庫執行個體。如果寫入器資料庫執行個體是已佈建的資料庫執行個體，則 Aurora 會估計同等 Aurora Serverless v2 容量。它會使用該估計值做為 Aurora Serverless v2 讀取器資料庫執行個體的最小容量。

第 2–15 層中的 Aurora Serverless v2 讀取器資料庫執行個體對其最小容量沒有相同的限制。當它們處於閒置狀態時，可以縮減規模到叢集容量範圍中指定的最小 Aurora 容量單位 (ACU) 值。

下列 Linux AWS CLI 範例示範如何檢查叢集中所有資料庫執行個體的提升層。最後一個欄位針對寫入器資料庫執行個體為 `True` 值，針對所有讀取器資料庫執行個體為 `False` 值。

```
$ aws rds describe-db-clusters --db-cluster-identifier promotion-tier-demo \
  --query 'DBClusters[*].DBClusterMembers[*].
[PromotionTier,DBInstanceIdentifier,IsClusterWriter]' \
  --output text

1   instance-192  True
1   tier-01-4840  False
10  tier-10-7425  False
15  tier-15-6694  False
```

下列 Linux AWS CLI 範例示範如何變更叢集中特定資料庫執行個體的提升層。這些命令首先使用新的提升層修改資料庫執行個體。然後，他們等待資料庫執行個體再次可用，並確認資料庫執行個體的新提升層。

```
$ aws rds modify-db-instance --db-instance-identifier instance-192 --promotion-tier 0
$ aws rds wait db-instance-available --db-instance-identifier instance-192
$ aws rds describe-db-instances --db-instance-identifier instance-192 \
  --query '*[].[PromotionTier]' --output text
0
```

如需為不同使用案例指定提升層的詳細指導方針，請參閱Aurora Serverless v2 擴展。

搭配 Aurora Serverless v2 使用 TLS/SSL

Aurora Serverless v2 可使用 Transport Layer Security/Secure Sockets Layer (TLS/SSL) 通訊協定，來加密用戶端與 Aurora Serverless v2 資料庫執行個體之間的通訊。它支援 TLS/SSL 1.0、1.1 和 1.2 版。如需如何將 TLS/SSL 與 Aurora 搭配使用的一般資訊，請參閱Aurora MySQL 資料庫叢集的 TLS 連線。

若要進一步了解如何使用 MySQL 用戶端連線至 Aurora MySQL 資料庫，請參閱連線至執行 MySQL 資料庫引擎的資料庫執行個體。

Aurora Serverless v2 支援 MySQL 用戶端 (`mysql`) 和 PostgreSQL 用戶端 (`psql`) 可用的所有 TLS/SSL 模式，包括下表所列的模式。

TLS/SSL 模式的說明	mysql	psql
不使用 TLS/SSL 連線。	DISABLED	停用
請先嘗試先使用 TLS/SSL 連線，但必要時會回復為 SSL 以外的加密技術。	PREFERRED	偏好 (預設)
強制使用 TLS/SSL。	REQUIRED	require
強制採用 TLS/SSL 並驗證憑證授權單位 (CA)。	VERIFY_CA	verify-ca
強制執行 TLS/SSL、驗證 CA，並驗證 CA 主機名稱。	VERIFY_IDENTITY	verify-full

Aurora Serverless v2 使用萬用字元憑證。如果您在使用 TLS/SSL 時指定「驗證 CA」或「驗證 CA 和 CA 主機名稱」選項，請先從 Amazon 信任服務下載 Amazon 根 CA 1 信任存放區。這樣做之後，您就可以在用戶端命令中識別這個 PEM 格式的檔案。若要使用 PostgreSQL 用戶端執行此作業，請執行以下步驟。

對於 Linux、macOS 或 Unix：

```
psql 'host=endpoint user=user sslmode=require sslrootcert=amazon-root-CA-1.pem
 dbname=db-name'
```

若要進一步了解如何將 Postgres 用戶端與 Aurora PostgreSQL 資料庫搭配使用，請參閱連線至執行 PostgreSQL 資料庫引擎的資料庫執行個體。

如需連線至 Aurora 資料庫叢集的一般資訊，請參閱 連接至 Amazon Aurora 資料庫叢集。

適用於 Aurora Serverless v2 資料庫叢集連線的受支援密碼套件

透過使用可設定的密碼套件，您可以更進一步控制資料庫連線的安全性。您可以指定要允許的密碼套件清單，以保護您資料庫的用戶端 SSL/TLS 連線安全。您可以使用可設定的密碼套件來控制資料庫伺服器接受的連線加密。這麼做可以防止使用不安全或不再使用的密碼。

基於 Aurora MySQL 的 Aurora Serverless v2 資料庫叢集支援與 Aurora MySQL 佈建之資料庫叢集相同的密碼套件。如需這些密碼套件的相關資訊，請參閱 為 Aurora MySQL 資料庫叢集的連線設定密碼套件。

基於 Aurora PostgreSQL 的 Aurora Serverless v2 資料庫叢集支援與 Aurora PostgreSQL 佈建之資料庫叢集相同的密碼套件。如需這些密碼套件的相關資訊，請參閱 為 Aurora PostgreSQL 資料庫叢集的連線設定密碼套件。

檢視 Aurora Serverless v2 寫入器和讀取器

您可以使用檢視已佈建資料庫執行個體的相同方式，來檢視 Aurora Serverless v2 資料庫執行個體的詳細資訊。若要執行此作業，請依照檢視 Amazon Aurora 資料庫叢集中的一般程序進行。叢集可能包含所有 Aurora Serverless v2 資料庫執行個體、所有已佈建資料庫執行個體，或兩者的部分。

在您建立一個或以上的 Aurora Serverless v2 資料庫執行個體後，您可以檢視哪些資料庫執行個體類型為 Serverless (無伺服器)，哪一些為 Instance (執行個體)。您還可以查看 Aurora Serverless v2 資料庫執行個體可以使用的最小和最大 Aurora 容量單位 (ACU)。每一個 ACU 都是處理 (CPU) 與記憶體 (RAM) 容量的組合。此容量範圍適用於叢集中的每個 Aurora Serverless v2 資料庫執行個體。如需檢查叢集或叢集中任何 Aurora Serverless v2 資料庫執行個體之容量範圍的程序，請參閱檢查 Aurora Serverless v2 的容量範圍。

在 Aurora Serverless v2 中 AWS 管理主控台，資料庫執行個體會在資料庫頁面的大小欄下標記。已佈建的資料庫執行個體會顯示資料庫執行個體類別的名稱，例如 r6g.xlarge。Aurora Serverless 資料庫執行個體的資料庫執行個體類別會顯示為 Serverless (無伺服器)，加上資料庫執行個體的最小和最大容量。例如，您可能會看到 Serverless v2 (4–64 ACUs) (無伺服器 v2 (4–64 個 ACU)) 或者 Serverless v2 (1–40 ACUs) (無伺服器 v2 (1–40 個 ACU))。

您可以在主控台的每個 Aurora Serverless v2 資料庫執行個體的 Configuration (組態) 索引標籤中找到相同資訊。例如，您可能會看到 Instance type (執行個體類型) 區段，如下所示。這裡，Instance type (執行個體類型) 值為 Serverless v2 (無伺服器 v2)，Minimum capacity (容量下限) 值為 2 ACUs (4 GiB)，以及 Maximum capacity (容量上限) 值為 64 ACUs (128 GiB)。



您可以監控每個 Aurora Serverless v2 資料庫執行個體一段時間後的容量。透過這種方式，您可以檢查每個資料庫執行個體的最小、最大和平均 ACU。您還可以檢查資料庫執行個體接近其最小或最大容量的程度。若要在 中查看此類詳細資訊 AWS 管理主控台，請檢查資料庫執行個體監控索引標籤上的 Amazon CloudWatch 指標圖形。如需要觀察的指標以及如何解釋它們的詳細資訊，請參閱Aurora Serverless v2 的重要 Amazon CloudWatch 指標。

為 Aurora Serverless v2 記錄日誌。

要開啟資料庫日誌記錄，請使用自訂參數群組中的組態參數指定要啟用的日誌。

對於 Aurora MySQL，您可以啟用下列日誌記錄。

Aurora MySQL	描述
`general_log`	建立一般日誌。設定為 1 可開啟。預設為關閉 (0)。
`log_queries_not_using_indexes`	將任何查詢記錄到不使用索引的慢查詢日誌。預設為關閉 (0)。設定為 1 可開啟此日誌。

Aurora MySQL	描述
long_query_time	防止快速執行的查詢在慢查詢日誌中記錄。可以設定為介於 0 到 31536000 之間的浮點數。預設值為 0 (非作用中)。
server_audit_events	日誌中要擷取的事件清單。支援的值為 CONNECT、QUERY、QUERY_DCL 、QUERY_DDL 、QUERY_DML 和 TABLE。
server_audit_logging	設定為 1 以開啟伺服器稽核日誌記錄。如果開啟此選項，您可以在 server_audit_events 參數中列出稽核事件，以指定要傳送至 CloudWatch 的稽核事件。
slow_query_log	建立慢查詢日誌。設定為 1 以開啟慢查詢日誌。預設為關閉 (0)。

如需詳細資訊，請參閱使用進階稽核與 Amazon Aurora MySQL 資料庫叢集搭配。

對於 Aurora PostgreSQL，您可以在 Aurora Serverless v2 資料庫執行個體上啟用下列日誌記錄。

Aurora PostgreSQL	Description
log_connections	記錄每個成功連線。
log_disconnections	記錄工作階段的結尾，包括持續時間。
log_lock_waits	預設值為 0 (關閉)。設定為 1 可等待日誌鎖定。
log_min_duration_statement	記錄陳述式之前執行的最短持續時間 (以毫秒為單位)。
log_min_messages	設定所記錄的訊息層級。支援的值為 debug5、debug4、debug3、debug2、debug1、inf 若要將效能資料記錄到 postgres 日誌中，則將值設定為 debug1。

Aurora PostgreSQL	Description
log_temp_files	記錄超過指定 KB 的暫存檔案的使用。
log_statement	控制已記錄的特定 SQL 陳述式。支援的值為 none、ddl、mod 和 all。預設值為 none。

主題

- 使用 Amazon CloudWatch 進行記錄
- 在 Amazon CloudWatch 中檢視 Aurora Serverless v2 日誌
- 使用 Amazon CloudWatch 監控容量
- 監控 Aurora Serverless v2 暫停和繼續活動

使用 Amazon CloudWatch 進行記錄

使用為 Aurora Serverless v2 記錄日誌。 中的程序選擇要開啟的資料庫日誌後，您可以選擇要將哪些日誌上傳(「發佈」) 到 Amazon CloudWatch。

您可使用 Amazon CloudWatch 來分析日誌資料、建立警示和檢視指標。預設情況下，Aurora Serverless v2 的錯誤日誌已啟用並自動上傳至 CloudWatch。您也可以從 Aurora Serverless v2 資料庫執行個體上傳其他日誌到 CloudWatch。

然後，您可以使用 中的日誌匯出設定 AWS 管理主控台 或 中的 `--enable-cloudwatch-logs-exports`選項，選擇要上傳至 CloudWatch 的日誌 AWS CLI。

您可以選擇將哪些 Aurora Serverless v2 日誌上傳到 CloudWatch。如需詳細資訊，請參閱使用進階稽核與 Amazon Aurora MySQL 資料庫叢集搭配。

與任何類型的 Aurora 資料庫叢集一樣，您無法修改預設的資料庫叢集參數群組。相反地，請根據資料庫叢集和引擎類型的預設參數，建立您自己的資料庫叢集參數群組。建議您在建立 Aurora Serverless v2 資料庫叢集之前建立自訂的資料庫叢集參數群組，以便在主控台上建立資料庫時可選擇您自訂的群組。

> **Note**
>
> 對於 Aurora Serverless v2，您可以同時建立資料庫叢集和資料庫參數群組。這與 Aurora Serverless v1 相反，在其中您只能建立資料庫叢集參數群組。

在 Amazon CloudWatch 中檢視 Aurora Serverless v2 日誌

使用使用 Amazon CloudWatch 進行記錄中的程序選擇要開啟的資料庫日誌後，就可以查看日誌的內容。

如需搭配 Aurora MySQL 和 Aurora PostgreSQL 日誌使用 CloudWatch 的詳細資訊，請參閱在 Amazon CloudWatch 中監控日誌事件和將 Aurora PostgreSQL 日誌發佈至 Amazon CloudWatch Logs。

檢視 Aurora Serverless v2 資料庫叢集的日誌

1. 透過 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 選擇您的 AWS 區域。
3. 選擇 Log groups (日誌群組)。
4. 從清單中選擇 Aurora Serverless v2 資料庫叢集日誌。日誌命名模式如下。

```
/aws/rds/cluster/cluster-name/log_type
```

> **Note**
>
> 對於 Aurora MySQL 相容的 Aurora Serverless v2 資料庫叢集，即使沒有錯誤，錯誤日誌也會包含緩衝集區擴展事件。

使用 Amazon CloudWatch 監控容量

搭配 Aurora Serverless v2，您可以使用 CloudWatch 來監控叢集中所有 Aurora Serverless v2 資料庫執行個體的容量和使用率。您可以檢視執行個體層級指標，以檢查每個 Aurora Serverless v2 資料庫執行個體在擴充和縮減規模時的容量。您也可以將容量相關指標與其他指標進行比較，以查看工作負載中的變更如何影響資源耗用。例如，您可以將 `ServerlessDatabaseCapacity` 與 `DatabaseUsedMemory`、`DatabaseConnections` 和 `DMLThroughput` 進行比較，來評估資料庫叢集在操作期間如何回應。如需適用於 Aurora Serverless v2 的容量相關指標，請參閱Aurora Serverless v2 的重要 Amazon CloudWatch 指標。

監控 Aurora Serverless v2 資料庫叢集的容量

1. 透過 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。

2. 選擇 Metrics (指標)。所有可用的指標都會顯示為主控台中的卡片，並依服務名稱分組。
3. 選擇 RDS。
4. (選用) 使用 Search (搜尋) 方塊來尋找對於 Aurora Serverless v2 特別重要的指標：`ServerlessDatabaseCapacity`、`ACUUtilization`、`CPUUtilization`，以及`FreeableMemory`。

我們建議您設定 CloudWatch 儀表板，使用容量相關指標來監控 Aurora Serverless v2 資料庫叢集容量。若要了解做法，請參閱使用 CloudWatch 建置儀表板。

若要進一步了解如何將 Amazon CloudWatch 與 Amazon Aurora 搭配使用，請參閱 將 Amazon Aurora MySQL 日誌發佈至 Amazon CloudWatch Logs。

監控 Aurora Serverless v2 暫停和繼續活動

Aurora 會為已啟用自動暫停的 Aurora Serverless v2 資料庫執行個體寫入單獨的日誌檔案。Aurora 會每隔 10 分鐘寫入日誌，而執行個體不會暫停。Aurora 會保留最多 7 個日誌，每日輪換。目前的日誌檔案名為 `instance.log`，而較舊的日誌則使用模式 `instance.`*`YYYY-MM-DD`*`.`*`N`*`.log` 來命名。

根據預設，已啟用自動暫停的 Aurora Serverless v2 資料庫執行個體會啟用此日誌。您可以在 或使用 AWS 管理主控台 或 AWS CLI RDSP API 來檢視此日誌的內容。目前，您無法將此日誌上傳至 CloudWatch。

資料庫執行個體事件 中列出的事件提供暫停和繼續活動的高階概觀，如下所示：

- 當執行個體開始暫停，以及完成暫停時。
- 當執行個體開始繼續，以及完成繼續時。
- 執行個體嘗試暫停，但某些情況防止其暫停的案例。

`instance.log` 提供更詳細的詳細資訊，說明 Aurora Serverless v2 執行個體可能或無法暫停的原因。

日誌可能表示執行個體基於不同原因而繼續：

- 使用者活動：資料庫連線請求。這可能是來自互動式用戶端工作階段、RDS Data API 呼叫，或從執行個體下載日誌的請求。
- 背景活動：廣泛的類別，包含 Aurora 繼續執行個體的所有原因。例如，當對讀取器執行個體的連線請求導致寫入器執行個體繼續時，讀取器的日誌會指出使用者活動，而寫入器的日誌會將該繼續請求

分類為背景活動。如需 Aurora 可能在使用者連線請求以外繼續執行個體的原因，請參閱 [繼續自動暫停的 Aurora Serverless v2 執行個體](#)。

當 Aurora 不知道任何會在自動暫停間隔到期時防止執行個體暫停的情況時，它會定期將資訊性訊息寫入日誌。對於只有單一資料庫執行個體的叢集，日誌包含此訊息：

```
[INFO] No auto-pause blockers registered since time
```

對於具有多個資料庫執行個體的叢集，訊息會略有不同。這是因為寫入器可能因為在任何讀取器執行個體上的活動而無法暫停。如果讀取器上的活動在寫入器上的自動暫停間隔到期之前完成，寫入器可以在預期的時間暫停。

```
[INFO] No auto-pause blockers registered since time.
Database might be required to maintain compute capacity for high availability.
```

如果暫停操作開始，但新的資料庫連線請求在執行個體完成暫停之前到達，則日誌會包含此訊息：

```
[INFO] Unable to pause database due to a new database activity
```

如果 Aurora 知道任何確實防止執行個體暫停的情況，日誌會包含此訊息，列出所有此類條件：

```
[INFO] Auto-pause blockers registered since time: list_of_conditions
```

如此一來，Aurora 就不會防止您開啟複寫、零 ETL 整合、Aurora 全球資料庫等功能，並搭配自動暫停功能。日誌會在使用這類功能可能會防止自動暫停生效時通知您。

以下是 Aurora Serverless v2 執行個體可能超過自動暫停逾時間隔，但無法暫停的原因：

- 自動暫停逾時之前的資料庫活動：資料庫執行個體在逾時間隔到期之前收到連線請求。
- 全球資料庫的成員：如果資料庫叢集是 Aurora 全球資料庫的一部分，則叢集中的 Aurora Serverless v2 執行個體不會暫停。叢集可以從獨立叢集變更為全球資料庫叢集。因此，先前自動暫停的執行個體可能會停止暫停，並在日誌中報告此原因。一旦叢集成為全球資料庫的成員，在您明確分離之前，不會還原至獨立叢集。即使您分離所有次要叢集，主要叢集仍會被視為全球資料庫的一部分。
- 已設定的複寫功能：寫入器資料庫執行個體已啟用引擎特定的複寫，可以是 MySQL 的 binlog 複寫或 PostgreSQL 的邏輯複寫。此條件也可能是由於使用另一個需要開啟複寫的 Aurora 功能所造成，例如零 ETL 整合或資料庫活動串流 (DAS)。

- 持續備份延遲：如果 Aurora 儲存系統尚未完成到目前時間點的儲存變更，寫入器執行個體不會暫停，直到趕上進度為止。此情況只會影響寫入器執行個體，且預期相對短暫。
- 服務或客戶維護動作：如果維護操作開始，則在操作完成之前，資料庫執行個體不會再次暫停。此情況包含您或 Aurora 可能啟動的各種操作，例如升級、複製、變更組態設定、升級、下載日誌檔案等。當您請求刪除執行個體時，也會發生此事件，而 Aurora 會短暫地繼續執行個體，作為刪除機制的一部分。
- 暫時性通訊問題：如果 Aurora 無法判斷擴展組態目前是否具有零個 ACU的最小容量設定，則不會暫停執行個體。這預期是非常罕見的情況。

Aurora Serverless v2 的效能和擴展

下列程序和範例會顯示如何設定 Aurora Serverless v2 羣集及其相關聯資料庫執行個體的容量範圍。您還可使用下列程序，監控資料庫執行個體的繁忙程度。之後，您可使用調查結果來決定是否需要向上或向下調整容量範圍。

在您使用這些程序之前，請確定您熟悉 Aurora Serverless v2 擴展的運作方式。擴展機制與 Aurora Serverless v1 中不同。如需詳細資訊，請參閱 Aurora Serverless v2 擴展。

內容

- 選擇 Aurora 叢集的 Aurora Serverless v2 容量範圍
  - 選擇叢集的最小值 Aurora Serverless v2 容量設定
  - 選擇叢集的最大值 Aurora Serverless v2 容量設定
  - 範例：變更 Aurora MySQL 叢集的 Aurora Serverless v2 容量範圍
  - 範例：變更 Aurora PostgreSQL 叢集的 Aurora Serverless v2 容量範圍
- 使用 Aurora Serverless v2 的參數群組
  - 預設參數值
  - Aurora Serverless v2 的連線數上限
  - Aurora 在 Aurora Serverless v2 放大和縮小時調整的參數
  - Aurora 依據 Aurora Serverless v2 最大容量運算的參數
- 避免記憶體不足錯誤
- Aurora Serverless v2 的重要 Amazon CloudWatch 指標
  - Aurora Serverless v2 指標如何套用至您的 AWS 帳單
  - Aurora Serverless v2 指標的 CloudWatch 命令範例
- 使用績效詳情監控 Aurora Serverless v2 效能

- 針對 Aurora Serverless v2 容量問題進行疑難排解

選擇 Aurora 叢集的 Aurora Serverless v2 容量範圍

利用 Aurora Serverless v2 資料庫執行個體，您可設定套用至資料庫叢集中所有資料庫執行個體的容量範圍，同時新增第一個 Aurora Serverless v2 資料庫執行個體至資料庫叢集。若需執行此作業的程序，請參閱 設定叢集的 Aurora Serverless v2 容量範圍。

您還可變更現有叢集的容量範圍。下列各節會以更詳細的方式討論如何選擇適當的最小值和最大值，及變更容量範圍時會發生的情況。例如，變更容量範圍可修改某些組態參數的預設值。套用所有參數變更可能需要重新啟動每個 Aurora Serverless v2 資料庫執行個體。

主題

- 選擇叢集的最小值 Aurora Serverless v2 容量設定
- 選擇叢集的最大值 Aurora Serverless v2 容量設定
- 範例：變更 Aurora MySQL 叢集的 Aurora Serverless v2 容量範圍
- 範例：變更 Aurora PostgreSQL 叢集的 Aurora Serverless v2 容量範圍

選擇叢集的最小值 Aurora Serverless v2 容量設定

總是為最小 Aurora Serverless v2 容量設定選擇 0.5 是很誘人的。該值可讓資料庫執行個體在完全閒置時縮減規模至最小容量，同時保持作用中狀態。您也可以指定最小容量為 0 個 ACU 來啟用自動暫停行為，如 使用 Aurora Serverless v2 的自動暫停和繼續擴展至零個 ACU 中所述。但是，依據您使用該叢集的方式及您設定的其他設定，不同的最小容量可能是最有效的。在選擇最小容量設定時，請考慮下列因素：

- Aurora Serverless v2 資料庫執行個體的擴展率依其目前容量而定。目前的容量越高，則擴充規模就越快。若您需要資料庫執行個體快速擴展至非常高的容量，請考慮將最小容量設定為擴展率符合您要求的值。
- 若您通常在預期工作負載特別高或低的情況下修改資料庫執行個體的資料庫執行個體類別，則可使用該體驗粗略估計等效的 Aurora Serverless v2 容量範圍。如要決定低流量時要使用的記憶體大小，請參閱 Aurora 資料庫執行個體類別的硬體規格。

  例如，假設您在叢集的工作負載較低時，使用 db.r6g.xlarge 資料庫執行個體類別。該資料庫執行個體類別有 32 GiB 的記憶體。因此，您可將 Aurora 容量單位 (ACU) 的最小設定指定為 16，來設定 Aurora Serverless v2 資料庫執行個體，此可縮小至大約相同的容量。這是因為每個 ACU 對應至大

約 2 GiB 的記憶體。如果您的 db.r6g.xlarge 資料庫執行個體有時並未充分利用，則您可指定一個較低的值，以進一步縮小資料庫執行個體。

- 若資料庫執行個體在緩衝區快取中具有一定數量的資料時，您的應用程式運作效率最高，請考慮指定最小 ACU 設定，其記憶體夠大足以容納頻繁存取的資料。否則，當 Aurora Serverless v2 資料庫執行個體縮減至較低的記憶體大小時。則會從緩衝區快取中移出一些資料。然後，當資料庫執行個體擴展進行備份時，資訊將隨著時間的推移讀回緩衝區快取中。若要將資料帶回緩衝區快取的 I/O 量很大，則選擇較高的最小 ACU 值可能會更有效。
- 若您的 Aurora Serverless v2 資料庫執行個體大部分時間都以特定容量執行，請考慮指定低於該基準但不要太低的最小容量設定。Aurora Serverless v2當目前容量並未明顯低於所需容量時，資料庫執行個體可最有效地估計擴充規模的數量和速度。
- 若您佈建的工作負載對於 T3 或 T4g 等小型資料庫執行個體類別的記憶體要求過高，請選擇提供與 R5 或 R6g 資料庫執行個體相當的記憶體最小 ACU 設定。

  我們特別建議與指定功能一起使用的下列最小容量 (這些建議可能會發生變化)：

  - Performance Insights – 2 個 ACU
  - Aurora 全域資料庫 – 8 個 ACU (僅適用於主要 AWS 區域)
- 在 Aurora 中，複寫發生在儲存層，因此讀取器容量不會直接影響複寫。不過，對於獨立擴展的 Aurora Serverless v2 讀取器資料庫執行個體，請確定最小容量足以在寫入密集期間處理工作負載，以避免查詢延遲。如果提升 2–15 層中的讀取器資料庫執行個體遇到效能問題，請考慮增加叢集的最小容量。如需有關選擇讀取器資料庫執行個體是隨寫入器擴展還是獨立擴展的詳細資訊，請參閱 選擇 Aurora Serverless v2 讀取器的提升層。
- 如果您有具有 Aurora Serverless v2 讀取器資料庫執行個體的資料庫叢集，當讀取器的提升層不是 0 或 1 時，讀取器不會與寫入器資料庫執行個體一起擴展。在這種情況下，設定較低的最小容量可能會導致過多的複寫延遲。這是因為讀取器可能沒有足夠的容量，無法在資料庫繁忙時套用寫入器的變更。我們建議您將最小容量設定為代表與寫入器資料庫執行個體相當之記憶體和 CPU 數量的值。
- Aurora Serverless v2 資料庫執行個體的 `max_connections` 參數值取決於從最大 ACU 衍生的記憶體大小。但是，當您指定 PostgreSQL 相容資料庫執行個體的最小容量為 0 或 0.5 個 ACU 時，`max_connections` 的上限值會設定為 2,000。

  若要將 Aurora PostgreSQL 叢集用於高連線工作負載，請考慮使用最小 1 或更高的 ACU 設定。如需有關 Aurora Serverless v2 處理 `max_connections` 組態參數的方法，請參閱 Aurora Serverless v2 的連線數上限。
- Aurora Serverless v2 資料庫執行個體從最小容量擴展至最大容量所需的時間取決於其最小和最大 ACU 值之間的差異。當資料庫執行個體的目前容量較大時，Aurora Serverless v2 擴充規模的增量較資料庫執行個體從小容量開始的增量更大。因此，若您指定了相對較大的最大容量，且資料庫執行個

體的大部分時間花在該容量附近，請考慮增加最小 ACU 設定。這樣，閒置資料庫執行個體便可更快速地擴展至最大容量。

選擇叢集的最大值 Aurora Serverless v2 容量設定

總是為最大 Aurora Serverless v2 容量設定選擇一些高值是很誘人的。較大的最大容量允許資料庫執行個體在執行密集型工作負載時可以最大程度擴展。較低的值可避免未預期的收費可能性。依據您使用該叢集的方式及您設定的其他設定，最有效的值可能高於或低於您最初想象的值。在選擇最大容量設定時，請考慮下列因素：

- 最大容量必須至少與最小容量相同。您可將最小和最大容量設定為相同。但是，在這種情況下，容量永遠不會擴展或縮小。因此，除了在測試情況下，對最小和最大容量使用相同的值並不合適。
- 最大容量必須高於 0.5 ACU。在大多數情況下，您可將最小和最大容量設定為相同。但是，不可為最小值和最大值指定 0.5。使用 1 或更高的值作為最大容量。
- 若您通常在預期工作負載特別高或低的情況下修改資料庫執行個體的資料庫執行個體類別，則可使用該體驗來估計等效的 Aurora Serverless v2 容量範圍。如要決定高流量時要使用的記憶體大小，請參閱 Aurora 資料庫執行個體類別的硬體規格。

  例如，假設您在叢集的工作負載較高時，使用 db.r6g.4xlarge 資料庫執行個體類別。該資料庫執行個體類別有 128 GiB 的記憶體。因此，您可將最大 ACU 設定指定為 64，來設定 Aurora Serverless v2 資料庫執行個體，此可擴展至大約相同的容量。這是因為每個 ACU 對應至大約 2 GiB 的記憶體。若 db.r6g.4xlarge 資料庫執行個體有時並無足夠的容量來有效處理工作負載，您可指定一個稍高的值，讓資料庫執行個體進一步向上擴展。
- 若您的資料庫用量有預算上限，請選擇一個保持在該上限內的值，即使您的 Aurora Serverless v2 資料庫執行個體一直都以最大容量執行。請記住，當您的叢集中有 n 個 Aurora Serverless v2資料庫執行個體時，依理論，叢集在任何時候可消耗的最大 Aurora Serverless v2 容量為叢集最大 ACU 設定的 n 倍。(實際消耗量可能會減少，例如，若某些讀取器獨立於寫入器擴展。)
- 若您使用 Aurora Serverless v2 讀取器資料庫執行個體從寫入器資料庫執行個體中卸載部分唯讀工作負載，則可選擇較低的最大容量設定。您這麼做是為了反映每個讀取器資料庫執行個體毋需像叢集僅包含單一資料庫執行個體那麼高的擴展。
- 假設您想要防止因資料庫參數設定錯誤或應用程式中的效率低查詢而造成的過度使用。於這種情況下，您可透過選擇低於您可設定的絕對最高容量設定來避免意外過度使用。
- 若因實際使用者活動所引起的峯值很少，但確實發生，您可在選擇最大容量設定時考慮這些情況。若優先順序是讓應用程式以完整的效能和可擴展性持續執行，您可指定一個高於正常用量下所觀察到的最大容量設定。若應用程式可在活動極端高峯期間以縮減的輸送量執行，則可選擇稍低的最大容量設定。確保您選擇的設定仍具有足夠的記憶體和 CPU 資源，以保持應用程式的執行。

- 若您在叢集中啟用了增加每個資料庫執行個體的記憶體用量的設定，請在決定最大 ACU 值時考慮該記憶體。這些設定包括績效詳情、Aurora MySQL 平行查詢、Aurora MySQL 效能結構描述和 Aurora MySQL 二進位日誌複寫的設定。請確保最大 ACU 值允許 Aurora Serverless v2 資料庫執行個體的擴充規模足以在使用這些功能時處理工作負載。如需有關對由低位最大 ACU 設定和利用記憶體額外負荷之 Aurora 功能組合所造成問題進行疑難排解的資訊，請參閱 避免記憶體不足錯誤。

範例：變更 Aurora MySQL 叢集的 Aurora Serverless v2 容量範圍

下列 AWS CLI 範例顯示如何更新現有 Aurora MySQL Aurora Serverless v2 叢集中資料庫執行個體的 ACU 範圍。一開始叢集的容量範圍為 8–32 個 ACU。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

資料庫執行個體處於閒置狀態，並縮小至 8 個 ACU。此時，下列與容量相關的設定適用於資料庫執行個體。為了以易於讀取的單位表示緩衝區集區的大小，我們將其除以 2 的 30 次方，產生以 gibibyte (GiB) 為單位的測量值。這是因為 Aurora 的記憶體相關測量使用以 2次方為基礎，而不是 10 次方。

```
mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              3000 |
+-------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|                9294577664 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----------+
| gibibytes |
```

```
+-----------+
|   8.65625 |
+-----------+
1 row in set (0.00 sec)
```

接下來，我們會變更叢集的容量範圍。在 modify-db-cluster 命令完成後，叢集的 ACU 範圍為 12.5–80。

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

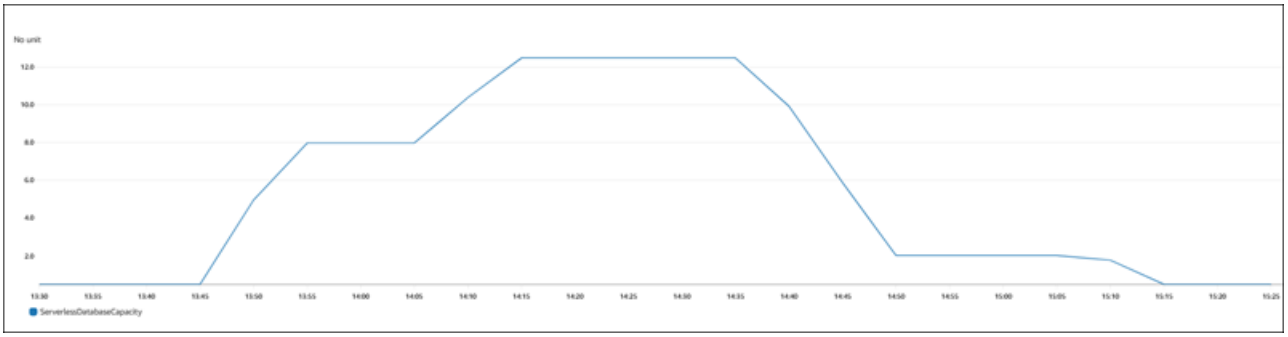
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

變更容量範圍會造成某些組態參數的預設值產生變更。Aurora 可立即套用其中一些新的預設值。但是，某些參數變更僅於重新啟動後生效。pending-reboot 狀態表示需要重新啟動才能套用某些參數的變更。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
}
```

此時，叢集處於閒置狀態，而資料庫執行個體 serverless-v2-instance-1 正在消耗 12.5 個 ACU。innodb_buffer_pool_size 參數已根據資料庫執行個體的目前容量進行調整。max_connections 參數仍然反映來自先前最大容量的值。重新設定該值需要重新啟動資料庫執行個體。

> **Note**
>
> 如果您直接在自訂資料庫參數群組中設定 max_connections 參數，則不需要重新啟動。

```
mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              3000 |
+-------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|               15572402176 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+---------------+
| gibibytes     |
+---------------+
| 14.5029296875 |
+---------------+
1 row in set (0.00 sec)
```

我們現在會重新啟動資料庫執行個體，並等待其再次可用。

```
aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBInstanceStatus": "rebooting"
}

aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

pending-reboot 狀態已清除。該值 in-sync 確認 Aurora 已套用所有待定參數變更。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
```

```
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "in-sync"
        }
    ]
}
```

innodb_buffer_pool_size 參數已增加至其閒置資料庫執行個體的最終大小。已增加 max_connections 參數，來反映從最大 ACU 值衍生出的值。當記憶體大小加倍時，Aurora 用於 max_connections 的公式會增加 1,000。

```
mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|               16139681792 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----------+
| gibibytes |
+-----------+
|  15.03125 |
+-----------+
1 row in set (0.00 sec)

mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              4000 |
+-------------------+
1 row in set (0.00 sec)
```

我們將容量範圍設定為 0.5–128 個 ACU，然後重新啟動資料庫執行個體。閒置資料庫執行個體的緩衝區快取大小現在小於 1 GiB，因此我們以 Mebi 位元組 (MiB) 為單位進行測量。max_connections 值為 5000 衍生自最大容量設定的記憶體大小。

```
mysql> select @@innodb_buffer_pool_size / pow(2,20) as mebibytes, @@max_connections;
+-----------+-------------------+
| mebibytes | @@max_connections |
+-----------+-------------------+
|       672 |              5000 |
+-----------+-------------------+
1 row in set (0.00 sec)
```

範例：變更 Aurora PostgreSQL 叢集的 Aurora Serverless v2 容量範圍

下列 CLI 範例顯示如何更新現有 Aurora PostgreSQL 叢集中 Aurora Serverless v2 資料庫執行個體的 ACU 範圍。

1. 叢集的容量範圍從 0.5–1 個 ACU 開始。
2. 將容量範圍變更為 8–32 個 ACUS。
3. 將容量範圍變更為 12.5–80 個 ACUS。
4. 將容量範圍變更為 0.5–128 個 ACUS。
5. 將容量恢復到 0.5–1 個 ACU 的初始範圍。

下圖顯示 Amazon CloudWatch 中的容量變更。



資料庫執行個體處於閒置狀態，並縮小至 0.5 個 ACU。此時，下列與容量相關的設定適用於資料庫執行個體。

```
postgres=> show max_connections;
 max_connections
```

```
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

接下來，我們會變更叢集的容量範圍。在 modify-db-cluster 命令完成後，叢集的 ACU 範圍為 8.0–32。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

變更容量範圍會造成某些組態參數的預設值產生變更。Aurora 可立即套用其中一些新的預設值。但是，某些參數變更僅於重新啟動後生效。pending-reboot 狀態表示需要重新啟動才能套用某些參數的變更。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
}
```

此時，叢集處於閒置狀態，而資料庫執行個體 serverless-v2-instance-1 正在消耗 8.0 個 ACU。shared_buffers 參數已根據資料庫執行個體的目前容量進行調整。max_connections 參數仍然反映來自先前最大容量的值。重新設定該值需要重新啟動資料庫執行個體。

> **Note**
>
> 如果您直接在自訂資料庫參數群組中設定 max_connections 參數，則不需要重新啟動。

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 1425408
(1 row)
```

我們會重新啟動資料庫執行個體，並等待其再次可用。

```
aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBInstanceStatus": "rebooting"
}

aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

資料庫執行個體現已重新啟動，pending-reboot 處於清除狀態。該值 in-sync 確認 Aurora 已套用所有待定參數變更。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "in-sync"
        }
    ]
```

```
}
```

重新啟動後，max_connections 會顯示來自新最大容量的值。

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 1425408
(1 row)
```

接下來，我們會將叢集的容量範圍變更為 12.5–80 個 ACU。

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

此時，叢集處於閒置狀態，而資料庫執行個體 serverless-v2-instance-1 正在消耗 12.5 個 ACU。shared_buffers 參數已根據資料庫執行個體的目前容量進行調整。max_connections 值仍然是 5000。

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 2211840
```

```
(1 row)
```

我們會再次重新啟動，但參數值保持不變。這是因為 max_connections 對於執行 Aurora PostgreSQL 的 Aurora Serverless v2 資料庫叢集具有最大值 5000。

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 2211840
(1 row)
```

現在，我們會將容量範圍從 0.5 設為 128 個 ACU。資料庫叢集會縮減規模至 10 個 ACU，然後再縮減規模至 2 個。我們會重新啟動資料庫執行個體。

```
postgres=> show max_connections;
 max_connections
-----------------
 2000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

Aurora Serverless v2 資料庫執行個體的 max_connections 值取決於從最大 ACU 衍生的記憶體大小。但是，當您指定 PostgreSQL 相容資料庫執行個體的最小容量為 0 或 0.5 個 ACU 時，max_connections 的上限值會設定為 2,000。

現在，我們將容量恢復到 0.5–1 個 ACU 的初始範圍，然後重新啟動資料庫執行個體。max_connections 參數已返回至原始值。

```
postgres=> show max_connections;
 max_connections
-----------------
```

```
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

使用 Aurora Serverless v2 的參數群組

建立 Aurora Serverless v2 資料庫叢集時，您可以選擇特定的 Aurora 資料庫引擎和相關的資料庫叢集參數群組。若您不熟悉 Aurora 如何使用參數群組，於叢集間一致地套用組態設定，請參閱 Amazon Aurora 的參數群組。所有為參數群組建立、修改、套用及其他動作的程序皆套用至 Aurora Serverless v2。

參數群組功能在佈建叢集和包含Aurora Serverless v2 資料庫執行個體之叢集間的運作方式通常相同：

- 叢集中所有資料庫執行個體的預設參數值由叢集參數群組進行定義。
- 您可為這些資料庫執行個體指定自訂資料庫參數群組，來覆寫特定資料庫執行個體的某些參數。您可於特定資料庫執行個體的偵錯或效能調校過程中執行此作業。例如，假設您有一個包含一些 Aurora Serverless v2 資料庫執行個體和一些佈建資料庫執行個體的叢集。於此狀況下，您可使用自訂資料庫參數群組為佈建的資料庫執行個體指定一些不同的參數。
- 若為 Aurora Serverless v2，您可使用參數群組之 `SupportedEngineModes` 屬性中值為 `provisioned` 的所有參數。於 Aurora Serverless v1 中，您僅可使用 `SupportedEngineModes` 屬性中具有 `serverless` 的參數子集。

主題

- 預設參數值
- Aurora Serverless v2 的連線數上限
- Aurora 在 Aurora Serverless v2 放大和縮小時調整的參數
- Aurora 依據 Aurora Serverless v2 最大容量運算的參數

預設參數值

佈建資料庫執行個體和 Aurora Serverless v2 資料庫執行個體之間的關鍵差別在於，Aurora 會覆寫與資料庫執行個體容量相關之某些參數的任何自訂參數值。自訂參數值仍可套用至叢集中的任何佈建資

料庫執行個體。如需有關 Aurora Serverless v2 資料庫執行個體如何解譯 Aurora 參數群組中參數的更多詳細資訊，請參閱 Aurora 叢集的組態參數。有關 Aurora Serverless v2 覆寫的特定參數，請參閱 Aurora 在 Aurora Serverless v2 放大和縮小時調整的參數 和 Aurora 依據 Aurora Serverless v2 最大容量運算的參數。

使用 describe-db-cluster-parameters CLI 命令並查詢 AWS 區域，您可取得各種 Aurora 資料庫引擎預設參數群組的預設值清單。下列為您可用於與 Aurora Serverless v2 相容之引擎版本的 `--db-parameter-group-family` 和 `-db-parameter-group-name` 選項值。

資料庫引擎和版本	參數群組系列	預設參數群組名稱
Aurora MySQL 第 3 版	`aurora-mysql8.0`	`default.aurora-mysql8.0`
Aurora PostgreSQL 13.x 版	`aurora-postgresql13`	`default.aurora-postgresql13`
Aurora PostgreSQL 14.x 版	`aurora-postgresql14`	`default.aurora-postgresql14`
Aurora PostgreSQL 15.x 版	`aurora-postgresql15`	`default.aurora-postgresql15`
Aurora PostgreSQL 16.x 版	`aurora-postgresql16`	`default.aurora-postgresql16`
Aurora PostgreSQL 17.x 版	`aurora-postgresql17`	`default.aurora-postgresql17`

下列範例會從 Aurora MySQL 第 3 版和 Aurora PostgreSQL 13 的預設資料庫叢集群組取得參數清單。這些是您與 Aurora Serverless v2 一起使用的 Aurora MySQL 和 Aurora PostgreSQL 版本。

對於 Linux、macOS 或 Unix：

```
aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name default.aurora-mysql8.0 \
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' \
```

```
  --output text

aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name default.aurora-postgresql13 \
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' \
  --output text
```

在 Windows 中：

```
aws rds describe-db-cluster-parameters ^
  --db-cluster-parameter-group-name default.aurora-mysql8.0 ^
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' ^
  --output text

aws rds describe-db-cluster-parameters ^
  --db-cluster-parameter-group-name default.aurora-postgresql13 ^
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' ^
  --output text
```

Aurora Serverless v2 的連線數上限

若為 Aurora MySQL 和 Aurora PostgreSQL，Aurora Serverless v2 資料庫執行個體會保持 `max_connections` 參數不變，則在資料庫執行個體縮減規模時，不會中斷連線。此參數的預設值衍生自以資料庫執行個體記憶體大小為基礎的公式。如需有關佈建資料庫執行個體類別之公式和預設值的詳細資訊，請參閱 對 Aurora MySQL 資料庫執行個體的連線數上限 和 對 Aurora PostgreSQL 資料庫執行個體的連線數上限。

Aurora Serverless v2 評估公式時，其會使用依據資料庫執行個體的最大 Aurora 容量單位 (ACU) 的記憶體大小，而非目前的 ACU 值。若變更預設值，我們建議使用公式的變化版而非指定常數值。如此，Aurora Serverless v2 可以使用以最大容量為基礎的適當設定。

變更 Aurora Serverless v2 資料庫叢集的最大容量時，您必須重新啟動 Aurora Serverless v2 資料庫執行個體，才能更新 `max_connections` 值。這是因為 `max_connections` 是 Aurora Serverless v2 的靜態參數。

下表根據最大 ACU 值顯示 Aurora Serverless v2 的 `max_connections` 預設值。

最大 ACU	Aurora MySQL 上的預設連線數上限	Aurora PostgreSQL 上的預設連線數上限
1	90	189
4	135	823
8	1,000	1，669
16	2,000	3，360
32	3,000	5,000
64	4,000	5,000
128	5,000	5,000
192	6,000	5,000
256	6,000	5,000

> **Note**
>
> Aurora Serverless v2 資料庫執行個體的 `max_connections` 值取決於從最大 ACU 衍生的記憶體大小。但是，當您指定 PostgreSQL 相容資料庫執行個體的最小容量為 0 或 0.5 個 ACU 時，`max_connections` 的上限值會設定為 2,000。

如需顯示 `max_connections` 如何隨著最大 ACU 值而變更的特定範例，請參閱 範例：變更 Aurora MySQL 叢集的 Aurora Serverless v2 容量範圍 和 範例：變更 Aurora PostgreSQL 叢集的 Aurora Serverless v2 容量範圍。

Aurora 在 Aurora Serverless v2 放大和縮小時調整的參數

在自動調整規模期間，Aurora Serverless v2 需要可變更每個資料庫執行個體的參數，以最合適的方法使用增加或減少的容量。因此，您無法覆寫與容量相關的某些參數。若為某些可覆寫的參數，請避免對固定值進行硬編碼。下列注意事項適用於與容量相關的這些設定。

若為 Aurora MySQL，Aurora Serverless v2 在擴展過程中會動態地調整一些參數的大小。若為下列參數，Aurora Serverless v2 不會使用您指定的任何自訂參數值：

- innodb_buffer_pool_size
- innodb_purge_threads
- table_definition_cache
- table_open_cache

若為 Aurora PostgreSQL，Aurora Serverless v2 在擴展期間會動態地調整下列參數參數的大小。若為下列參數，Aurora Serverless v2 不會使用您指定的任何自訂參數值：

- shared_buffers

若為此處所列出參數以外的所有參數，Aurora Serverless v2 資料庫執行個體的工作方式與佈建的資料庫執行個體相同。預設參數值繼承自叢集參數群組。您可使用自訂叢集參數群組，修改整個叢集的預設值。或者，您可使用自訂資料庫參數群組，修改某些資料庫執行個體的預設值。動態參數會立即更新。僅於您重新啟動資料庫執行個體後，對靜態參數進行的變更才會生效。

Aurora 依據 Aurora Serverless v2 最大容量運算的參數

若為下列參數，Aurora PostgreSQL 會使用預設值，這些預設值衍生自以最大 ACU 設定為基礎的記憶體大小，與 max_connections 相同：

- autovacuum_max_workers
- autovacuum_vacuum_cost_limit
- autovacuum_work_mem
- effective_cache_size
- maintenance_work_mem

避免記憶體不足錯誤

若您的其中一個 Aurora Serverless v2 資料庫執行個體持續達到其最大容量的限制，Aurora 會將資料庫執行個體的狀態設定為 incompatible-parameters，來指出此狀況。雖然資料庫執行個體的狀態為 incompatible-parameters，但某些作業會遭到封鎖。例如，您無法升級引擎版本。

通常，當資料庫執行個體因記憶體不足錯誤而頻繁重新啟動時，其會進入此狀態。Aurora 會在發生此類型的重新啟動時記錄事件。您可按照 檢視 Amazon RDS 事件 中的步驟檢視該事件。因開啟績效詳情和 IAM 身分驗證等設定的額外負荷，可能會發生異常高的記憶體用量。其還可能來自資料庫執行個體上的繁重工作負載或來自管理與大量結構描述物件相關聯的中繼資料。

若記憶體壓力降低，致使資料庫執行個體無法經常達到最大容量，則 Aurora 會自動將資料庫執行個體狀態變更回 `available`。

如要從此狀況復原，您可進行下列部分或全部動作：

- 變更叢集的最小 Aurora 容量單位 (ACU) 值，增加 Aurora Serverless v2 資料庫執行個體的容量下限。如此可避免閒置資料庫縮減規模至記憶體容量少於叢集中啟用功能所需的容量問題。變更叢集的 ACU 設定後，重新啟動 Aurora Serverless v2 資料庫執行個體。如此可評估 Aurora 是否將狀態重設為 `available`。
- 變更叢集的最大 Aurora 容量單位 (ACU) 值，增加 Aurora Serverless v2 資料庫執行個體的容量上限。如此可避免繁忙的資料庫無法擴充規模至具足夠記憶體的容量，以滿足叢集和資料庫工作負載中所開啟功能的問題。變更叢集的 ACU 設定後，重新啟動 Aurora Serverless v2 資料庫執行個體。如此可評估 Aurora 是否將狀態重設為 `available`。
- 請關閉需要記憶體額外負荷的組態設定。例如，假設您已開啟 AWS Identity and Access Management (IAM)、績效詳情或 Aurora MySQL 二進位日誌複寫等功能，但不使用它們。若是如此，您可將其關閉。或者，您可將叢集的最小和最大容量值調整得更高，以考慮這些功能所使用的記憶體。如需有關選擇最小和最大容量設定的指導方針，請參閱 選擇 Aurora 叢集的 Aurora Serverless v2 容量範圍。
- 縮減資料庫執行個體的工作負載。例如，您可將讀取器資料庫執行個體新增至叢集，以將來自唯讀查詢的負載分散至更多的資料庫執行個體中。
- 調整應用程式所使用的 SQL 程式碼，以使用較少的資源。例如，您可檢查您的查詢計畫、檢查慢查詢日誌或調整資料表上的索引。您還可執行其他傳統類型的 SQL 調校。

Aurora Serverless v2 的重要 Amazon CloudWatch 指標

如要開始使用您 Aurora Serverless v2 資料庫執行個體的 Amazon CloudWatch，請參閱 在 Amazon CloudWatch 中檢視 Aurora Serverless v2 日誌。若要進一步了解如何透過 CloudWatch 監控 Aurora 資料庫叢集，請參閱 在 Amazon CloudWatch 中監控日誌事件。

您可於 CloudWatch 中檢視 Aurora Serverless v2 資料庫執行個體，使用 `ServerlessDatabaseCapacity` 指標監控每個資料庫執行個體耗用的容量。您還可監控所有的標準 Aurora CloudWatch 指標，例如 `DatabaseConnections` 和 `Queries` 等。如需有關您可監控 Aurora

之 CloudWatch 指標的完整清單，請參閱 Amazon Aurora 的 Amazon CloudWatch 指標。指標分為叢集等級和執行個體等級指標，分別位於 Amazon Aurora 的叢集層級指標 和 Amazon Aurora 的執行個體層級指標 中。

下列 CloudWatch 執行個體等級指標對於監控了解您的 Aurora Serverless v2 資料庫執行個體如何擴展和縮減非常重要。所有這些指標每秒計算一次。如此，您便可監控 Aurora Serverless v2 資料庫執行個體的目前狀態。您可設定警報，以在任何 Aurora Serverless v2 資料庫執行個體接近與容量相關之指標的閾值時通知您。您可決定最小和最大容量設定是否合適，或者是否需要進行調整。您可決定將精力集中於最佳化您資料庫效率之處。

- `ServerlessDatabaseCapacity`。作為執行個體等級指標，其報告目前資料庫執行個體容量所代表的 ACU 數量。作為叢集等級指標，其代表叢集中所有 Aurora Serverless v2 資料庫執行個體的 `ServerlessDatabaseCapacity` 值的平均值。該指標只是 Aurora Serverless v1 中的叢集等級指標。於 Aurora Serverless v2 中，其可用於資料庫執行個體等級和叢集等級。
- `ACUUtilization`。該指標是 Aurora Serverless v2 中的新指標。此值以百分比表示。其計算方法為 `ServerlessDatabaseCapacity` 指標的值除以資料庫叢集的最大 ACU 值。請考慮下列指導方針來解譯此指標並採取行動：
  - 若此指標接近 `100.0` 值，則資料庫執行個體已盡可能地擴展。考慮增加叢集的最大 ACU 設定。如此，寫入器和讀取器資料庫執行個體皆可擴展至更高的容量。
  - 假設唯讀工作負載會導致讀取器資料庫執行個體接近 `100.0` 的 `ACUUtilization`，而寫入器資料庫執行個體並未接近其最大容量。於該狀況下，請考慮將額外的讀取器資料庫執行個體新增至叢集。如此，您可將工作負載的唯讀部分分散至更多資料庫執行個體中，進而減少每個讀取器資料庫執行個體的負載。
  - 假設您正在執行生產應用程式，其中效能和可擴展性是主要考慮因素。於該狀況下，您可將叢集的最大 ACU 值設定為較高的數字。您的目標適用於始終低於 `100.0` 的 `ACUUtilization` 指標。使用較高的最大 ACU 值，您可確信有足夠的空間以防資料庫活動出現意外高峰。您僅需為實際使用的資料庫容量付費。
- `CPUUtilization`。此指標在 Aurora Serverless v2 中的解譯與在佈建的資料庫執行個體中不同。若為 Aurora Serverless v2，此值是一個百分比，計算方法為目前使用的 CPU 量除以資料庫叢集最大 ACU 值下可用的 CPU 容量。Aurora 會自動監控此值並在資料庫執行個體持續使用其 CPU 容量的高比例時擴展您的 Aurora Serverless v2 資料庫執行個體。

  若此指標接近 `100.0` 值，則資料庫執行個體已達到其最大 CPU 容量。考慮增加叢集的最大 ACU 設定。若此指標在讀取器資料庫執行個體上接近 `100.0` 的值，請考慮將其他讀取器資料庫執行個體新增至叢集。如此，您可將工作負載的唯讀部分分散至更多資料庫執行個體中，進而減少每個讀取器資料庫執行個體的負載。

- `FreeableMemory`。此值表示當 Aurora Serverless v2 資料庫執行個體擴展至其最大容量時可用的未使用記憶體量。若為目前容量低於最大容量的每個 ACU，此值增加約 2 GiB。因此，在資料庫執行個體盡可能向上擴展之前，該指標不會接近零。

  若此指標接近 0 值，則資料庫執行個體已盡可能地進行擴展，且接近其可用記憶體的限制。考慮增加叢集的最大 ACU 設定。若此指標在讀取器資料庫執行個體上接近 0 的值，請考慮將其他讀取器資料庫執行個體新增至叢集。如此，工作負載的唯讀部分可分佈於更多資料庫執行個體上，進而減少每個讀取器資料庫執行個體上的記憶體用量。

- `TempStorageIOPS`。在附加至資料庫執行個體的本機儲存體上完成的 IOPS 數。其包括讀取和寫入的 IOPS。此指標表示計數，且每秒測量一次。這是 Aurora Serverless v2 的一個新指標。如需詳細資訊，請參閱Amazon Aurora 的執行個體層級指標。

- `TempStorageThroughput`。傳入和傳出與資料庫執行個體相關聯之本機儲存體的資料量。此指標表示位元組，且每秒測量一次。這是 Aurora Serverless v2 的一個新指標。如需詳細資訊，請參閱Amazon Aurora 的執行個體層級指標。

通常，Aurora Serverless v2 資料庫執行個體的大多數擴展是由記憶體用量和 CPU 活動造成的。`TempStorageIOPS` 和 `TempStorageThroughput` 指標可協助您診斷於資料庫執行個體和本機儲存體間進行傳輸之網路活動造成容量意外增加的罕見狀況。如要監控其他網路活動，您可使用以下現有指標：

- `NetworkReceiveThroughput`
- `NetworkThroughput`
- `NetworkTransmitThroughput`
- `StorageNetworkReceiveThroughput`
- `StorageNetworkThroughput`
- `StorageNetworkTransmitThroughput`

您可讓 Aurora 發佈部分或所有資料庫日誌到 Amazon CloudWatch Logs。如需指示，請視您的資料庫引擎而定參閱下列內容：

- the section called “將 Aurora PostgreSQL 日誌發佈至 CloudWatch Logs”
- the section called “將 Aurora MySQL 日誌發佈至 CloudWatch Logs”

Aurora Serverless v2 指標如何套用至您的 AWS 帳單

AWS 帳單上的Aurora Serverless v2費用是根據您可以監控的相同`ServerlessDatabaseCapacity`指標來計算。若您僅於一個小時的一部分內使用 Aurora Serverless v2 容量，計費機制可能與針對此指標計算的 CloudWatch 平均值不同。若系統問題造成 CloudWatch 指標在短時間內無法使用，其亦可能會有所不同。因此，您可能會看到帳單上的 ACU 小時值與您自己依據 `ServerlessDatabaseCapacity` 平均值計算的數字略有不同。

Aurora Serverless v2 指標的 CloudWatch 命令範例

下列 AWS CLI 範例示範如何監控與 相關的最重要 CloudWatch 指標Aurora Serverless v2。於每個案例中，將 `--dimensions` 參數的 `Value=` 字串替換為您自己的 Aurora Serverless v2 資料庫執行個體識別符。

下列 Linux 範例顯示資料庫執行個體的最小、最大和平均容量值，在一小時內每 10 分鐘測量一次。Linux `date` 命令指定相對於目前日期和時間的開始和結束時間。`--query` 參數中的 `sort_by` 函數依據 `Timestamp` 欄位依時間順序對結果進行排序。

```
aws cloudwatch get-metric-statistics --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

下列 Linux 範例展示了監控叢集中每個資料庫執行個體的容量。其測量每個資料庫執行個體的最小、最大和平均容量使用率。在三個小時內每小時進行一次測量。這些範例使用 `ACUUtilization` 指標表示 ACU 上限的百分比，而非 `ServerlessDatabaseCapacity` 表示固定數量的 ACU。如此，您無須知道容量範圍內最小和最大 ACU 值的實際數字。您可看到從 0 到 100 的百分比。

```
aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
  --start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_writer_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table

aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
  --start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_reader_instance \
```

```
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

下列 Linux 範例執行與先前測量類似的測量。於此狀況下，測量適用於 CPUUtilization 指標。在 1 小時內每 10 分鐘進行一次測量。依據資料庫執行個體的最大容量設定的可用 CPU 資源，這些數字表示已使用的可用 CPU 百分比。

```
aws cloudwatch get-metric-statistics --metric-name "CPUUtilization" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

下列 Linux 範例執行與先前測量類似的測量。於此狀況下，測量適用於 FreeableMemory 指標。在 1 小時內每 10 分鐘進行一次測量。

```
aws cloudwatch get-metric-statistics --metric-name "FreeableMemory" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

使用績效詳情監控 Aurora Serverless v2 效能

您可使用績效詳情監控 Aurora Serverless v2 資料庫執行個體的效能。如需有關績效詳情的程序，請參閱 在 Amazon Aurora 上使用績效詳情監控資料庫負載。

下列新的績效詳情計數器適用於 Aurora Serverless v2 資料庫執行個體：

- os.general.serverlessDatabaseCapacity – ACU 中資料庫執行個體的目前容量。該值對應至 ServerlessDatabaseCapacity 資料庫執行個體的 CloudWatch 指標。
- os.general.acuUtilization – 目前容量佔最大設定容量的百分比。該值對應至 ACUUtilization 資料庫執行個體的 CloudWatch 指標。
- os.general.maxConfiguredAcu – 您為此 Aurora Serverless v2 資料庫執行個體設定的最大容量。其以 ACU 為單位進行測量。
- os.general.minConfiguredAcu – 您為此 Aurora Serverless v2 資料庫執行個體設定的最小容量。其以 ACU 為單位進行測量。

如需績效詳情計數器的完整清單，請參閱 Performance Insights 計數器指標。

在績效詳情中顯示資料庫執行個體的 vCPU 值時，這些值會代表依據 Aurora Serverless v2 資料庫執行個體之 ACU 值的估計值。在預設的一分鐘間隔中，任何小數 vCPU 值皆會無條件進位至最接近的非負整數。若為較長的時間間隔，顯示的 vCPU 值為每分鐘整數 vCPU 值的平均值。

針對 Aurora Serverless v2 容量問題進行疑難排解

在某些情況下，即使資料庫沒有負載，Aurora Serverless v2 也不會縮減規模至最小容量。這種情況可能是由於下列原因而發生：

- 某些功能可以增加資源使用量，並防止資料庫縮減規模至最小容量。重要功能如下所示：
  - Aurora 全球資料庫
  - 探索 CloudWatch Logs
  - 在 Aurora PostgreSQL 相容的資料庫叢集上啟用 `pg_audit`
  - Enhanced Monitoring (增強型監控)
  - Performance Insights

  如需詳細資訊，請參閱選擇叢集的最小值 Aurora Serverless v2 容量設定。
- 如果讀取器執行個體未縮減規模至最小值，且維持與寫入器執行個體相同或比其更高的容量，請檢查讀取器執行個體的優先順序方案。方案 0 或方案 1 的 Aurora Serverless v2 讀取器資料庫執行個體保持至少與寫入器資料庫執行個體一樣高的最小容量。將讀取器的優先順序方案變更為 2 或更高，以便其可以獨立縱向擴展和縮減，與寫入器無關。如需詳細資訊，請參閱選擇 Aurora Serverless v2 讀取器的提升層。
- 將影響共用記憶體大小的任何資料庫參數設為其預設值。設定高於預設值的值會增加共用記憶體需求，並防止資料庫縮減規模至最小容量。範例包括 `max_connections` 和 `max_locks_per_transaction`。

  > **Note**
  >
  > 更新共用記憶體參數需要重新啟動資料庫，變更才能生效。

- 繁重的資料庫工作負載會增加資源使用量。
- 大型資料庫磁碟區會增加資源使用量。

Amazon Aurora 使用記憶體和 CPU 資源進行資料庫叢集管理。管理資料庫磁碟區更大的資料庫叢集，Aurora 需要更多 CPU 和記憶體。如果叢集的容量下限低於叢集管理所需的最小容量，則您的叢集將不會縮減至容量下限。

- 背景程序 (例如清除) 也可能會增加資源使用量。
- 平台版本限制可能會影響擴展功能。特定叢集的可用擴展範圍受引擎版本和硬體 (平台版本) 影響。可以在功能較差的平台版本上執行更強大的引擎版本，反之亦然。

如果資料庫仍未縮減規模至設定的最小容量，請停止並重新啟動資料庫，以回收任何可能已隨時間建立的記憶體片段。停止和啟動資料庫會導致停機，因此我們建議您謹慎執行此動作。

使用 Aurora Serverless v2 的自動暫停和繼續擴展至零個 ACU

您可以指定 Aurora Serverless v2 資料庫執行個體縮減規模到零個 ACU，如果使用者活動未在指定期間內啟動任何連線，則會自動暫停。您可以為資料庫叢集指定最小 ACU 值 0 來執行此操作。當執行個體處於暫停狀態時，您不需要支付執行個體容量的費用。為輕度使用或長時間閒置的 Aurora 叢集啟用自動暫停和繼續功能 (自動暫停)，可協助您管理資料庫機群的成本。

> **Note**
>
> 自動暫停功能適用於 Aurora Serverless v2 搭配 Aurora PostgreSQL 和 Aurora MySQL。您可能需要升級 Aurora 資料庫引擎版本，才能利用此功能。如需最低容量為 0 個 ACU的引擎版本，請參閱 Aurora Serverless v2 容量。

主題

- Aurora Serverless v2 自動暫停功能概觀
- Aurora Serverless v2 自動暫停功能的先決條件和限制
- 開啟和關閉自動暫停功能
- Aurora Serverless v2 自動暫停功能的運作方式
- Aurora Serverless v2 自動暫停如何適用於不同類型的 Aurora 叢集
- 監控使用自動暫停的 Aurora 叢集
- Aurora Serverless v2 自動暫停的故障診斷
- Aurora Serverless v2 自動暫停功能的應用程式設計考量

Aurora Serverless v2 自動暫停功能概觀

Aurora Serverless v2 資料庫執行個體可以在沒有使用者連線的期間之後自動暫停，並在連線請求送達時自動繼續。Aurora Serverless v2 自動暫停/繼續功能有助於管理沒有嚴格服務水準目標 (SLO) 的系統成本。例如，您可以為用於開發和測試的叢集啟用此功能，或針對資料庫繼續時可接受短暫暫停的內部應用程式啟用此功能。如果您的工作負載處於閒置期間，並且可以在執行個體繼續時容忍輕微的連線延遲，請考慮針對 Aurora Serverless v2 執行個體使用自動暫停來降低成本。

您可以透過指定叢集中的 Aurora Serverless v2 資料庫執行個體是否可以自動暫停，以及每個執行個體在暫停之前必須閒置多久來控制此行為。若要啟用 Aurora 叢集中所有 Aurora Serverless v2 資料庫執行個體的自動暫停行為，請將叢集的最小容量值設定為零個 ACU。

如果您先前利用在閒置一段時間後擴展到零個 ACU 的 Aurora Serverless v1 功能，您可以升級到 Aurora Serverless v2 並使用其對應的自動暫停功能。

自動暫停功能可節省成本的優點類似於使用停止/啟動叢集功能。Aurora Serverless v2 的自動暫停具有比啟動已停止叢集更快繼續的額外優點，並自動化決定何時暫停和繼續每個資料庫執行個體的程序。

自動暫停功能也提供額外的精細程度，以控制叢集內運算資源的成本。即使叢集中的寫入器執行個體和其他讀取器隨時保持作用中狀態，您仍可讓某些讀取器執行個體暫停。您可以透過將可與其他執行個體獨立暫停的讀取器執行個體指派為 2-15 範圍內的容錯移轉優先順序來執行此操作。

寫入器執行個體的容錯移轉優先順序為 0 和 1 的所有讀取器執行個體一律會同時暫停和繼續。因此，此群組中的執行個體在指定的時間間隔內沒有任何連線後便會暫停。

Aurora 資料庫叢集可以包含寫入器和讀取器資料庫執行個體，以及已佈建和 Aurora Serverless v2 資料庫執行個體的組合。因此，若要有效地使用此功能，了解自動暫停機制的下列層面會很有幫助：

- 資料庫執行個體可能自動暫停的情況。
- 可能防止資料庫執行個體暫停的時間。例如，啟用某些 Aurora 功能或在叢集上執行特定類型的操作可能會防止執行個體暫停，即使對於這些執行個體沒有任何連線也是如此。
- 執行個體暫停時監控和計費的後果。
- 哪些動作會導致資料庫執行個體繼續處理。
- 資料庫執行個體的容量在暫停和繼續事件期間如何變更。
- 如何控制資料庫執行個體暫停之前的閒置間隔。
- 如何編寫應用程式邏輯的程式碼，以處理資料庫執行個體繼續處理的期間。

Aurora Serverless v2 自動暫停功能的先決條件和限制

使用自動暫停功能之前，請檢查要使用的引擎版本。此外，請檢查自動暫停是否可搭配您打算使用的其他 Aurora 功能運作。如果您使用的是不支援它的引擎版本，則無法開啟自動暫停。對於不相容的功能，如果您將它們與自動暫停結合使用，則不會收到任何錯誤。如果叢集使用任何不相容的功能或設定，Aurora Serverless v2 執行個體不會自動暫停。

- 如果您使用的是 Aurora PostgreSQL，資料庫引擎必須至少執行 16.3、15.7、14.12 或 13.15 版。
- 如果您使用的是 Aurora MySQL，資料庫引擎必須執行 3.08.0 版或更新版本。
- 如需可使用此功能之引擎版本和 AWS 區域的完整清單，請參閱 Aurora Serverless v2 支援的區域和 Aurora 資料庫引擎。

- 當 Aurora Serverless v2 執行個體繼續時，其容量可能會低於執行個體暫停時的容量。如需詳細資訊，請參閱 Aurora Serverless v2 和 Aurora Serverless v1 之間自動暫停行為的差異。

特定情況或設定會防止 Aurora Serverless v2 執行個體自動暫停。如需更多詳細資訊，請參閱 Aurora Serverless v2 不會自動暫停的情況。

開啟和關閉自動暫停功能

您可以在叢集層級開啟和關閉自動暫停功能。若要這樣做，您可以使用與調整叢集容量下限和上限時相同的程序。自動暫停功能以最小容量 0 ACU 表示。

主題

- 為叢集中的 Aurora Serverless v2 執行個體開啟自動暫停
- 可設定的 Aurora Serverless v2 自動暫停逾時間隔
- 繼續自動暫停的 Aurora Serverless v2 執行個體
- 關閉叢集中 Aurora Serverless v2 執行個體的自動暫停

為叢集中的 Aurora Serverless v2 執行個體開啟自動暫停

請遵循 設定叢集的 Aurora Serverless v2 容量範圍 中的程序。針對最小容量，請選擇 0 ACU。當您選擇最小容量為 0 ACU 時，您也可以指定執行個體在自動暫停之前閒置的時間長度。

下列 CLI 範例顯示如何在啟用自動暫停功能且自動暫停間隔設定為十分鐘 (600 秒) 的情況下建立 Aurora 叢集。

```
aws rds create-db-cluster \
    --db-cluster-identifier my-serverless-v2-cluster \
    --region eu-central-1 \
    --engine aurora-mysql \
    --engine-version 8.0 \
    --serverless-v2-scaling-configuration
 MinCapacity=0,MaxCapacity=4,SecondsUntilAutoPause=600 \
    --master-username myuser \
    --manage-master-user-password
```

下列 CLI 範例顯示如何開啟現有 Aurora 叢集的自動暫停功能。此範例會將自動暫停間隔設定為一小時 (3600 秒)。

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
```

```
  --serverless-v2-scaling-configuration
 MinCapacity=0,MaxCapacity=80,SecondsUntilAutoPause=3600

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 0,
    "MaxCapacity": 80.0,
    "SecondsUntilAutoPause": 3600
}
```

可設定的 Aurora Serverless v2 自動暫停逾時間隔

逾時間隔會以 Aurora 叢集的 `ServerlessV2ScalingConfiguration` 屬性表示。如果最小容量設定為零個 ACU，您可以在建立或修改 Aurora 叢集時在 AWS 管理主控台 中指定此間隔。您可以在建立或修改 Aurora 叢集時使用 `--serverless-v2-scaling-configuration` 參數，在 AWS CLI 中指定它。您可以在建立或修改 Aurora 叢集時使用 `ServerlessV2ScalingConfiguration` 參數，在 RDS API 中指定它。

您可以設定的最小間隔為 300 秒 (5 分鐘)。如果您未指定間隔，則這是預設值。您可以設定的間隔上限為 86,400 秒 (一天)。

假設您透過將叢集的最小容量變更為非零值來關閉叢集的自動暫停功能。在這種情況下，間隔屬性會從 `ServerlessV2ScalingConfiguration` 屬性中移除。缺少該屬性可額外確認該叢集的自動暫停功能已關閉。如果您稍後重新開啟自動暫停，您可以在此時再次指定任何自訂間隔。

繼續自動暫停的 Aurora Serverless v2 執行個體

- 當您連線到暫停的 Aurora Serverless v2 執行個體時，它會自動繼續並接受連線。
- 未包含有效憑證的連線嘗試仍會導致資料庫執行個體繼續。
- 如果您透過寫入器端點連線，Aurora 會在自動暫停時繼續寫入器資料庫執行個體。同時，Aurora 會繼續容錯移轉優先順序為 0 或 1 的任何自動暫停讀取器執行個體，這表示其容量與寫入器執行個體的容量繫結。
- 如果您透過讀取器端點連線，Aurora 會隨機選擇讀取器執行個體。如果該讀取器執行個體暫停，Aurora 會繼續它。Aurora 也會先繼續寫入器執行個體，因為如果任何讀取器執行個體處於作用中狀態，寫入器執行個體必須一律處於作用中狀態。當 Aurora 繼續該寫入器執行個體時，這也會導致容錯移轉提升層 0 和 1 中的任何讀取器執行個體繼續。
- 如果您透過 RDS Data API 傳送請求至叢集，Aurora 會在暫停時繼續寫入器執行個體。然後，Aurora 會處理資料 API 請求。

- 當您變更資料庫叢集參數群組中組態參數的值時，Aurora 會自動繼續使用該叢集參數群組的所有叢集中任何暫停的 Aurora Serverless v2 執行個體。同樣地，當您變更資料庫參數群組中的參數值時，Aurora 會自動繼續使用該資料庫參數群組的任何暫停 Aurora Serverless v2 執行個體。當您修改叢集以指派不同的叢集參數群組，或是修改執行個體以指派不同的資料庫參數群組時，也會套用相同的自動繼續行為。
- 如果 Aurora Serverless v2 寫入器執行個體暫停，執行繼續請求會自動繼續。Aurora 會在寫入器執行個體繼續後處理恢復請求。您可以恢復到 Aurora Serverless v2 執行個體暫停的時間。
- 建立叢集快照或刪除快照不會導致任何 Aurora Serverless v2 執行個體繼續。
- 建立 Aurora 複製會導致 Aurora 繼續所複製叢集的寫入器執行個體。
- 如果暫停的執行個體在完成繼續之前收到大量的連線請求，則某些工作階段可能無法連線。我們建議對已啟用自動暫停的某些 Aurora Serverless v2 執行個體的 Aurora 叢集的連線實作重試邏輯。例如，您可以重試任何失敗的連線三次。
- Aurora 可以執行某些類型的次要內部維護，而無需喚醒執行個體。不過，在叢集維護時段期間發生的某些維護類型確實需要 Aurora 才能繼續執行個體。維護完成後，如果在指定的間隔後沒有活動，執行個體會自動再次暫停。

> **Note**
>
> Aurora 不會自動繼續引擎特定排程任務的暫停執行個體，例如 PostgreSQL `pg_cron` 延伸模組或 MySQL 事件排程器中的執行個體。為了確保此類任務執行，請在排程時間之前手動啟動與執行個體的連線。Aurora 不會將資料庫執行個體暫停時發生排程時間的任何任務排入佇列。當執行個體稍後繼續時，系統會略過此類任務。

- 如果 Aurora 叢集在 Aurora Serverless v2 執行個體自動暫停時經歷容錯移轉，Aurora 可能會繼續執行個體，然後將該執行個體提升為寫入器。如果一個或多個資料庫執行個體在執行個體暫停時從叢集中移除，則可能會發生相同情況。在此情況下，執行個體會在繼續時立即成為寫入器。
- 變更叢集屬性的操作也會導致任何自動暫停的 Aurora Serverless v2 執行個體繼續。例如，自動暫停的執行個體會繼續執行下列操作：
  - 變更叢集的擴展範圍。
  - 升級叢集的引擎版本
  - 從暫停的執行個體描述或下載日誌檔案。您可以啟用日誌上傳至 CloudWatch 並透過 CloudWatch 分析日誌，以檢查暫停執行個體的歷史日誌資料。

關閉叢集中 Aurora Serverless v2 執行個體的自動暫停

請遵循 設定叢集的 Aurora Serverless v2 容量範圍 中的程序。針對最小容量，選擇 0.5 或更高的值。當您關閉自動暫停功能時，執行個體閒置的間隔會重設。如果您再次開啟自動暫停，您可以指定新的逾時間隔。

下列 CLI 範例顯示如何關閉現有 Aurora 叢集的自動暫停功能。`describe-db-clusters` 輸出顯示當最小容量設定為非零值時，會移除 `SecondsUntilAutoPause` 屬性。

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=2,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 2,
    "MaxCapacity": 80.0
}
```

Aurora Serverless v2 自動暫停功能的運作方式

您可以使用以下資訊來規劃使用自動暫停功能。了解執行個體暫停、繼續或保持作用中狀態的情況，可協助您平衡可用性、回應能力和成本節省之間的權衡。

主題

- Aurora Serverless v2 執行個體暫停時會發生什麼情況
- 自動暫停的 Aurora Serverless v2 執行個體繼續時會發生什麼情況
- 執行個體計費對自動暫停 Aurora Serverless v2 叢集的運作方式
- Aurora Serverless v2 不會自動暫停的情況
- 自動暫停如何搭配叢集停止/啟動功能運作
- 自動暫停 Aurora Serverless v2 叢集的維護和升級如何運作
- Aurora Serverless v2 和 Aurora Serverless v1 之間自動暫停行為的差異

Aurora Serverless v2 執行個體暫停時會發生什麼情況

當 Aurora Serverless v2 資料庫執行個體在一段時間後沒有連線的情況下暫停時：

- 無論執行個體當時有多少 ACU，Aurora 都會在指定的間隔經過且沒有與執行個體的連線後開始暫停執行個體。
- 暫停機制不是即時的。即將自動暫停的 Aurora Serverless v2 執行個體可能會短暫等待，以了解 Aurora 儲存體的所有變更。
- 該執行個體的執行個體費用會保留。當執行個體暫停時，`ServerlessV2Usage` 指標的值為 0。
- 執行個體的狀態值不會變更。狀態仍顯示為「可用」。
- 執行個體會停止寫入資料庫日誌檔案。除了為 `CPUUtilization` 和 `ACUUtilization` 註冊零百分比，以及為 `ServerlessDatabaseCapacity` 註冊零百分比以外，它會停止將指標傳送至 CloudWatch。
- 當 Aurora Serverless v2 資料庫執行個體開始暫停、完成暫停，以及暫停機制中斷或失敗時，Aurora 會發出事件。如需這些事件的詳細資訊，請參閱 資料庫執行個體事件。

自動暫停的 Aurora Serverless v2 執行個體繼續時會發生什麼情況

當 Aurora Serverless v2 資料庫執行個體在自動暫停後繼續時，適用下列情況：

- 執行個體繼續時，會套用 `pending-reboot` 變更中的任何參數變更。
- Aurora 會在每個 Aurora Serverless v2資料庫執行個體開始繼續、完成繼續，以及執行個體因某種原因而無法繼續時發出執行個體層級事件。如需這些事件的詳細資訊，請參閱 資料庫執行個體事件。
- 任何請求的連線會在資料庫執行個體完成繼續後建立。由於繼續的典型時間約為 15 秒，我們建議您將任何用戶端逾時設定調整為超過 15 秒。例如，在 JDBC 驅動程式設定中，您可以將 `connectTimeout` 和 `sslResponseTimeout` 設定的值調整為超過 15 秒。

> **Note**
>
> 如果 Aurora Serverless v2 執行個體保持暫停超過 24 小時，Aurora 可以讓執行個體進入更深層的睡眠，這需要更長的時間才能繼續。在這種情況下，繼續時間可以是 30 秒或更長，大約等同於重新啟動執行個體。如果您的資料庫閒置期間超過一天，建議您將連線逾時設定為 30 秒或以上。

執行個體計費對自動暫停 Aurora Serverless v2 叢集的運作方式

當 Aurora Serverless v2 執行個體自動暫停時，其執行個體費用為零。該 `ServerlessV2Usage` 指標在此期間為零。AWS 仍會針對未繫結至該特定資料庫執行個體的 Aurora 儲存體和叢集的其他層面收取費用。

Aurora Serverless v2 不會自動暫停的情況

- 如果資料庫叢集的最小容量值高於零個 ACU，則叢集中的 Aurora Serverless v2 執行個體不會自動暫停。如果您現有的叢集在自動暫停功能可用之前具有 Aurora Serverless v2 執行個體，最低容量設定為 0.5 ACU。若要對此類叢集使用自動暫停功能，請將最小容量設定修改為零個 ACU。
- 如果任何使用者起始的連線開放給 Aurora Serverless v2 執行個體，執行個體將不會暫停。當修補和升級等活動正在進行時，執行個體也不會暫停。Aurora 用於運作狀態檢查的管理連線不會計入活動，也不會防止執行個體暫停。
- 在已啟用邏輯複寫的 Aurora PostgreSQL 叢集中，或已啟用 binlog 複寫的 Aurora MySQL 叢集中，容錯移轉提升層 0 和 1 中的寫入器和任何讀取器執行個體不會自動暫停。Aurora 會執行持續的最小活動量，以檢查複寫連線的運作狀態。

  對於啟用複寫的叢集，您仍可在自動暫停的來源叢集中擁有 Aurora 讀取器執行個體。若要這樣做，請將容錯移轉優先順序設定為 0 或 1 以外的值。如此一來，它們就可以與寫入器執行個體分開暫停。
- 如果您的 Aurora 叢集具有相關聯的 RDS Proxy，Proxy 會維護叢集中每個資料庫執行個體的開放連線。因此，此類叢集中的任何 Aurora Serverless v2 執行個體都不會自動暫停。
- 如果您的叢集是 Aurora 全球資料庫中的主要叢集，Aurora 不會自動暫停 Aurora Serverless v2 寫入器執行個體。這是因為寫入器執行個體需要持續的活動層級，才能管理全球資料庫中的其他叢集。由於寫入器執行個體保持作用中狀態，任何容錯移轉優先順序為 0 或 1 的 Aurora Serverless v2 讀取器執行個體也不會自動暫停。
- Aurora 全球資料庫次要叢集中的 Aurora Serverless v2 執行個體不會自動暫停。如果資料庫叢集提升為獨立叢集，則自動暫停功能會在符合所有其他情況時生效。
- 在與 Redshift 有關聯的零 ETL 整合的叢集中，寫入器執行個體和容錯移轉提升層 0 和 1 中的任何讀取器執行個體不會自動暫停。
- 除了執行個體主資料庫連接埠上的活動之外，如果 Aurora PostgreSQL 執行個體已啟用 Babelfish 功能，則 T-SQL 連接埠上的任何連線和活動都會防止執行個體自動暫停。

自動暫停如何搭配叢集停止/啟動功能運作

您可以在啟用自動暫停功能時停止和啟動 Aurora 叢集。某些執行個體是否暫停並不重要。當您再次啟動叢集時，任何暫停的 Aurora Serverless v2 執行個體都會自動繼續。

當 Aurora 叢集停止時，任何暫停的 Aurora Serverless v2 執行個體都不會根據連線嘗試自動繼續。再次啟動叢集後，就會套用暫停和繼續 Aurora Serverless v2 執行個體的一般機制。

自動暫停 Aurora Serverless v2 叢集的維護和升級如何運作

- 當 Aurora Serverless v2 執行個體自動暫停時，如果您嘗試升級 Aurora 叢集，Aurora 會繼續執行個體並進行升級。
- Aurora 會定期繼續任何自動暫停的 Aurora Serverless v2 執行個體以執行維護，例如次要版本升級和參數群組等屬性的變更。
- Aurora Serverless v2 執行個體喚醒以進行升級或套用維護等管理操作後，Aurora 會等待至少 20 分鐘，然後再次暫停該執行個體。這是為了允許任何背景操作完成。如果執行個體連續進行多個管理操作，則二十分鐘期間也會避免暫停和繼續執行個體多次。

Aurora Serverless v2 和 Aurora Serverless v1 之間自動暫停行為的差異

- 與 Aurora Serverless v2 相比，Aurora Serverless v1 中恢復時間有所改善。如果執行個體暫停不到 24 小時，則繼續時間通常約為 15 秒。如果執行個體暫停超過 24 小時，則繼續時間可能會更長。
- Aurora Serverless v2 適用於多個可用區叢集的方式表示叢集中的某些資料庫執行個體可能會在其他資料庫執行個體處於作用中狀態時暫停。寫入器執行個體會在任何讀取器執行時繼續，因為需要寫入器才能協調叢集中的特定活動。由於 Aurora Serverless v1 不會使用讀取器執行個體，因此整個叢集一律會暫停或處於作用中狀態。
- 當讀取器端點隨機挑選要連線的讀取器執行個體時，該讀取器執行個體可能已處於作用中狀態或可能已自動暫停。因此，存取讀取器執行個體的時間可能會有所不同，且難以預測。因此，使用 Aurora Serverless v2 和自動暫停的多個可用區叢集可能受益於為特定唯讀使用案例設定自訂端點，而不是將所有唯讀工作階段導向讀取器端點。
- Aurora Serverless v2 執行個體會以與已佈建執行個體相同的頻率進行維護操作。由於 Aurora 會在需要此類維護時自動繼續執行個體，因此您可能會發現 Aurora Serverless v2 執行個體繼續的頻率高於 Aurora Serverless v1 叢集。

Aurora Serverless v2 自動暫停如何適用於不同類型的 Aurora 叢集

自動暫停功能的考量取決於 Aurora 叢集中有多少執行個體、讀取器執行個體的容錯移轉提升層，以及所有執行個體是否是 Aurora Serverless v2 或 Aurora Serverless v2 和已佈建的組合。

主題

- 使用自動暫停時建議的 Aurora 叢集配置
- 資料庫叢集中寫入器執行個體的 Aurora Serverless v2 自動暫停運作方式
- Aurora Serverless v2 自動暫停對多個可用區叢集的運作方式
- Aurora Serverless v2 自動暫停對具有佈建執行個體的叢集的運作方式

使用自動暫停時建議的 Aurora 叢集配置

啟用自動暫停功能時，您可以為 Aurora 叢集安排高可用性、快速回應和可擴展性之間的適當平衡，以符合您的使用案例。您可以透過為叢集中的資料庫執行個體選擇 Aurora Serverless v2 執行個體、已佈建執行個體和容錯移轉提升層的組合來執行此操作。

下列類型的組態示範叢集的高可用性和成本最佳化之間的不同權衡：

- 對於開發和測試系統，您可以設定具有 Aurora Serverless v2 資料庫執行個體的單一可用區域資料庫叢集。單一執行個體提供所有讀取和寫入請求。當叢集在相當時間間隔內未使用時，資料庫執行個體會暫停。此時，叢集的資料庫運算成本也會暫停。
- 對於執行高可用性為優先順序的應用程式，但叢集仍有完全閒置期間的系統，您可以設定多個可用區叢集，其中寫入器和讀取器資料庫執行個體都是 Aurora Serverless v2。將讀取器執行個體設定為容錯移轉優先順序 0 或 1，讓寫入器和讀取器執行個體同時暫停和繼續。現在，您可以在叢集處於作用中狀態時獲得快速容錯移轉的優勢。當叢集保持閒置狀態的時間超過自動暫停閾值時，這兩個執行個體的資料庫執行個體費用都會暫停。當叢集繼續處理時，第一個資料庫工作階段需要短暫的連線時間。
- 假設您的叢集持續處於作用中狀態、活動量極少，且任何連線都需要快速回應。在這種情況下，您可以建立具有多個 Aurora Serverless v2 讀取器執行個體的叢集，並將某些讀取器執行個體的容量從寫入器分離。為寫入器執行個體和一個讀取器執行個體指定容錯移轉優先順序 0 或 1。為其他讀取器執行個體指定大於 1 的優先順序。如此一來，較高優先順序層中的讀取器執行個體可以自動暫停，即使寫入器和其中一個讀取器保持作用中也是如此。

  在這種情況下，您可以採用一些其他技術，以確保叢集在閒置期間仍縮減規模至低容量時持續可用：

  - 您可以針對寫入器和優先順序 0 或 1 讀取器使用佈建的執行個體。如此一來，兩個資料庫執行個體永遠不會自動暫停，且一律可用於提供資料庫流量和執行容錯移轉。

- 您可以設定自訂端點，其中包含較高優先順序層中的 Aurora Serverless v2 執行個體，但無法設定寫入器或提升層 0 或 1 讀取器。如此一來，您可以將對延遲不敏感的唯讀工作階段導向至可能會自動暫停的讀取器。您可以避免對此類請求使用讀取器端點，因為 Aurora 可能會將讀取器端點連線導向始終保持喚醒的讀取器執行個體，或是其中一個自動暫停的執行個體。使用自訂端點可讓您根據對快速回應或額外擴展容量的偏好，將連線導向不同的執行個體群組。

資料庫叢集中寫入器執行個體的 Aurora Serverless v2 自動暫停運作方式

當 Aurora 資料庫叢集只包含單一資料庫執行個體時，自動暫停和繼續資料庫執行個體的機制非常簡單。它僅取決於寫入器執行個體上的活動。對於用於開發和測試的叢集，或用於執行高可用性不重要的應用程式，您可能具有此類組態。請注意，在單一執行個體叢集中，Aurora 會透過讀取器端點將連線導向寫入器資料庫執行個體。因此，對於單一執行個體資料庫叢集，嘗試連線至讀取器端點會導致自動暫停的寫入器資料庫執行個體繼續。

下列其他因素適用於具有多個資料庫執行個體的 Aurora 叢集：

- 在 Aurora 資料庫叢集中，寫入器資料庫執行個體通常會經常性存取。因此，即使一或多個讀取器資料庫執行個體自動暫停，您可能會發現寫入器資料庫執行個體保持作用中狀態。
- 讀取器資料庫執行個體上的某些活動需要提供寫入器資料庫執行個體。因此，在所有讀取器執行個體也暫停之前，寫入器資料庫執行個體都無法暫停。即使您的應用程式未直接存取寫入器執行個體，繼續任何讀取器執行個體也會自動繼續寫入器執行個體。
- 容錯移轉提升層 0 和 1 中的 Aurora Serverless v2 讀取器執行個體會進行擴展，使其容量與寫入器執行個體保持同步。因此，當 Aurora Serverless v2 寫入器執行個體繼續時，任何位於提升層 0 或 1 的 Aurora Serverless v2 讀取器執行個體都會繼續。

Aurora Serverless v2 自動暫停對多個可用區叢集的運作方式

在同時包含寫入器和一或多個讀取器資料庫執行個體的 Aurora 資料庫叢集內，某些 Aurora Serverless v2 資料庫執行個體可能會在其他資料庫執行個體處於作用中狀態時暫停。寫入器執行個體和容錯移轉優先順序為 0 和 1 的任何讀取器執行個體一律會同時暫停和繼續。優先順序不是 0 或 1 的讀取器執行個體可以與其他執行個體分開暫停和繼續。

當您將此功能用於具有多個讀取器執行個體的叢集時，您可以管理成本，而不會犧牲高可用性。寫入器執行個體和另一個一或兩個讀取器執行個體隨時可以保持作用中狀態，而其他讀取器執行個體可以在不需要處理大量讀取流量時暫停。

讀取器執行個體是否可以自動暫停取決於其容量是否可以獨立擴展，或容量是否與寫入器資料庫執行個體的容量繫結。該擴展屬性取決於讀取器資料庫執行個體的容錯移轉優先順序。當讀取器的優先順序為

0 或 1 時，讀取器的容量會追蹤寫入器資料庫執行個體的容量。因此，若要允許讀取器資料庫執行個體在最廣泛的情況下自動暫停，請將優先順序設定為高於 1 的值。

繼續讀取器執行個體的時間可能比繼續寫入器執行個體稍長。如需執行個體可能暫停時的最快回應，請連線至叢集端點。

Aurora Serverless v2 自動暫停對具有佈建執行個體的叢集的運作方式

Aurora 資料庫叢集中的任何已佈建資料庫執行個體都不會自動暫停。只有具有 `db.serverless` 執行個體類別的 Aurora Serverless v2 資料庫執行個體才能使用自動暫停功能。

當您的 Aurora 叢集包含任何佈建的資料庫執行個體時，任何 Aurora Serverless v2 寫入器執行個體都不會自動暫停。這是因為在任何讀取器執行個體作用中時，寫入器執行個體必須保持可用。Aurora Serverless v2 寫入器保持作用中的事實也表示容錯移轉優先順序為 0 和 1 的任何 Aurora Serverless v2 讀取器執行個體都不會在包含任何已佈建執行個體的混合叢集中自動暫停。

監控使用自動暫停的 Aurora 叢集

若要監控 Aurora，您應該已經熟悉 使用 Amazon CloudWatch 監控 Amazon Aurora 指標 中的監控程序，以及 Amazon Aurora 的指標參考 中列出的 CloudWatch 指標。請注意，當您監控使用自動暫停功能的 Aurora 叢集時，有一些特殊考量：

- 因為 Aurora Serverless v2 執行個體已暫停，所以有時候執行個體不會記錄日誌資料和大多數指標。執行個體暫停時傳送到 CloudWatch 的唯一指標為 `CPUUtilization` 和 `ACUUtilization` 的零百分比，以及 `ServerlessDatabaseCapacity` 的零百分比。
- 您可以檢查 Aurora Serverless v2 執行個體暫停的頻率是否高於或低於預期。若要這樣做，請檢查 `ServerlessDatabaseCapacity` 指標從非零值變更為零的頻率，以及它保持零的時間長度。如果執行個體的暫停時間不如預期，則無法節省盡可能多的成本。如果執行個體暫停和繼續的頻率高於您的預期，您的叢集在回應連線請求時可能會有不必要的延遲。如需影響 Aurora Serverless v2 執行個體是否可以暫停以及暫停頻率之因素的相關資訊，請參閱 Aurora Serverless v2 自動暫停功能的先決條件和限制、Aurora Serverless v2 不會自動暫停的情況 和 Aurora Serverless v2 自動暫停的故障診斷。
- 您也可以檢查記錄 Aurora Serverless v2 執行個體自動暫停和繼續操作的日誌檔案。如果執行個體在逾時間隔到期後未暫停，此日誌檔案也包含自動暫停未發生的原因。如需更多詳細資訊，請參閱 監控 Aurora Serverless v2 暫停和繼續活動。

主題

- 檢查 Aurora Serverless v2 執行個體是否暫停
- 自動暫停和自動繼續操作的事件
- 自動暫停如何與 Performance Insights 和增強型監控搭配使用
- Aurora Serverless v2 自動暫停功能如何與 Aurora 指標互動

檢查 Aurora Serverless v2 執行個體是否暫停

若要判斷 Aurora Serverless v2 執行個體是否處於暫停狀態，您可以觀察執行個體的 `ACUUtilization` 指標。執行個體暫停時，該指標的值為零。

當 Aurora Serverless v2 執行個體暫停時，其狀態值仍會列為可用。當暫停的 Aurora Serverless v2 執行個體正在繼續時，同樣適用。這是因為即使連線發生些微延遲，您也可以成功連線到此類執行個體。

任何與 Aurora 執行個體可用性相關的指標都會將執行個體暫停的期間視為執行個體可用的時間。

自動暫停和自動繼續操作的事件

當自動暫停和自動繼續操作開始、完成或取消時，Aurora 會發出 Aurora Serverless v2 執行個體的事件。與自動暫停功能相關的事件是透過 `RDS-EVENT-0374` 的 `RDS-EVENT-0370`。如需這些事件的詳細資訊，請參閱 資料庫執行個體事件。

自動暫停如何與 Performance Insights 和增強型監控搭配使用

當 Aurora Serverless v2 執行個體暫停時，Aurora 不會透過 Performance Insights 或增強型監控來收集該執行個體的監控資訊。執行個體繼續時，在 Aurora 繼續收集此類監控資訊之前，可能會有短暫的延遲。

Aurora Serverless v2 自動暫停功能如何與 Aurora 指標互動

當 Aurora Serverless v2 執行個體暫停時，它不會發出大多數 CloudWatch 指標或將任何資訊寫入其資料庫日誌。執行個體暫停時傳送到 CloudWatch 的唯一指標為 `CPUUtilization` 和 `ACUUtilization` 的零百分比，以及 `ServerlessDatabaseCapacity` 的零百分比。

當 CloudWatch 是執行個體或叢集可用性和執行時間相關的運算統計資料時，Aurora Serverless v2 執行個體會被視為在暫停期間可用。

當您啟動 AWS CLI 或 RDS API 動作來描述或下載暫停 Aurora Serverless v2 執行個體的日誌時，執行個體會自動繼續以提供日誌資訊。

CloudWatch 指標範例

下列 AWS CLI 範例示範如何觀察執行個體容量隨時間變更的情況。在此期間，執行個體會自動暫停，然後繼續。暫停時，ServerlessDatabaseCapacity 指標會報告 0 的值。為了判斷執行個體是否在期間內的任何時間點暫停，我們會檢查該期間內的最小容量是否為零。

下列 Linux 範例代表已自動暫停一段時間的 Aurora Serverless v2 執行個體。我們會在三分鐘的期間內每分鐘取樣 ServerlessDatabaseCapacity 一次。最小 ACU 值 0.0 會確認執行個體已在每分鐘的某個時間點暫停。

```
$ aws cloudwatch get-metric-statistics \
  --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '3 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --statistics Minimum \
  --namespace "AWS/RDS" --dimensions Name=DBInstanceIdentifier,Value=my-autopause-
instance \
  --output text | sort -k 3

ServerlessDatabaseCapacity
DATAPOINTS 0.0 2024-08-02T22:11:00+00:00 None
DATAPOINTS 0.0 2024-08-02T22:12:00+00:00 None
DATAPOINTS 0.0 2024-08-02T22:13:00+00:00 None
```

接下來，我們會嘗試與暫停的 Aurora Serverless v2 執行個體建立連線。在此範例中，我們刻意使用不正確的密碼，讓連線嘗試不會成功。儘管失敗，連線嘗試會導致 Aurora 繼續暫停的執行個體。

```
$ mysql -h my_cluster_endpoint.rds.amazonaws.com -u admin -pwrong-password

ERROR 1045 (28000): Access denied for user 'admin'@'ip_address' (using password: YES)
```

下列 Linux 範例示範暫停的執行個體已繼續、保持閒置約五分鐘，然後回到暫停狀態。執行個體以 2.0 ACU 的容量繼續。然後，它會稍微向上擴展，例如執行一些內部清除。由於它在五分鐘的逾時期間內未收到任何使用者連線嘗試，因此從 4.0 ACU 直接進入暫停狀態。

```
$ aws cloudwatch get-metric-statistics \
  --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '8 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --statistics Minimum \
```

```
  --namespace "AWS/RDS" --dimensions Name=DBInstanceIdentifier,Value=my-autopause-
instance \
  --output text | sort -k 3

ServerlessDatabaseCapacity
DATAPOINTS 0.0 2024-08-02T22:13:00+00:00 None
DATAPOINTS 2.0 2024-08-02T22:14:00+00:00 None
DATAPOINTS 3.0 2024-08-02T22:15:00+00:00 None
DATAPOINTS 3.0 2024-08-02T22:16:00+00:00 None
DATAPOINTS 4.0 2024-08-02T22:17:00+00:00 None
DATAPOINTS 4.0 2024-08-02T22:18:00+00:00 None
DATAPOINTS 4.0 2024-08-02T22:19:00+00:00 None
DATAPOINTS 0.0 2024-08-02T22:20:00+00:00 None
```

如果報告旨在顯示執行個體擴展處理工作負載的速度，我們可能會為統計資料指定 `Maximum` 而非 `Minimum`。對於容量規劃和成本估算，我們可能會指定更長的期間並使用 `Average` 統計資料。如此一來，我們可以判斷在高活動量、低活動量和暫停狀態期間的一般容量值。為了在關於暫停和繼續的精確時間內檢查行為，我們可能會指定一秒的期間，並檢查較短的時間間隔。輸出中的時間戳記值，例如 `2024-08-02T22:13:00+00:00`，示範為 `--start-time` 和 `--end-time` 選項指定精確參數的格式。

Aurora Serverless v2 自動暫停的故障診斷

如果您發現 Aurora Serverless v2 執行個體的暫停頻率不如預期，請檢查下列可能原因：

- 確認您執行的 Aurora 版本支援最低容量為零個 ACU。如需不同 Aurora 版本的容量範圍，請參閱 Aurora Serverless v2 容量。
- 確認叢集的最小容量值設定為零個 ACU。
- 確認有問題的執行個體實際使用 Aurora Serverless v2 執行個體類別 `db.serverless`，而不是其中一個佈建的執行個體類別。
- 確認叢集未使用來自 Aurora Serverless v2 自動暫停功能的先決條件和限制 的任何不相容功能或設定。
- 檢查日誌檔案，顯示 Aurora Serverless v2 執行個體何時暫停、繼續，或 Aurora 因為某些原因而無法暫停或繼續執行個體。如需更多詳細資訊，請參閱 監控 Aurora Serverless v2 暫停和繼續活動。
- 檢查是否有任何用戶端或應用程式長時間保持連線開啟。相反地，請檢查任何使用 RDS 資料 API 或 Lambda 函數的應用程式是否經常傳送請求，讓執行個體永遠不會閒置到足以暫停的時間。您可以檢查 CloudWatch 指標，例如 `ConnectionAttempts` 和 `DatabaseConnections`。如需更多詳細資訊，請參閱 Amazon Aurora 的執行個體層級指標。

- 如果讀取器執行個體很少暫停，請檢查其容錯移轉優先順序。如果讀取器用於讀取擴展，而不是在容錯移轉時作為待命，請將其設定為 2-15 範圍內的優先順序。
- 如果寫入器執行個體很少暫停，請檢查您的讀取器執行個體使用情況。寫入器只能在整個叢集處於閒置狀態時暫停。這是因為連線到任何讀取器執行個體會導致寫入器繼續。
- 如果您的應用程式在 Aurora Serverless v2 執行個體繼續時於連線請求期間收到逾時，請考慮延長應用程式程式碼或基礎資料庫架構所使用的逾時間隔。較長的連線逾時可降低 Aurora Serverless v2 執行個體繼續時連線失敗的可能性。不過，較長的逾時也會讓您的應用程式較慢，難以偵測叢集中的可用性問題。

  相反地，請考慮延長自動暫停間隔，讓應用程式不會經常遇到暫停的執行個體。

  如果自動暫停行為與應用程式的叢集回應能力之間沒有邏輯平衡，則該叢集可能不適合使用自動暫停功能。
- 如果您估計 Aurora Serverless v2 執行個體會暫停多久，請注意，有些因素會使精確預測變得不切實際。
  - 執行個體可能會定期繼續執行維護、次要版本升級，或將變更套用至參數群組。
  - 對於多個可用區叢集，在某些情況下，繼續一個執行個體也會導致其他執行個體繼續。繼續任何讀取器一律會導致寫入器繼續。繼續寫入器一律會導致容錯移轉優先順序為 0 和 1 的任何讀取器繼續。

  我們建議您使用實際的工作負載，測量幾天內的實際暫停時間。然後，使用這些測量來設定您可以預期執行個體暫停的時間比例的基準。
- 您可能會發現 MySQL 清除、MySQL 事件排程器、PostgreSQL 自動清空或透過 `pg_cron` 延伸模組排程的 PostgreSQL 任務等內部操作未執行或未完成。執行個體不會自動繼續執行不涉及使用者與資料庫連線的操作。如果自動暫停逾時到期時正在進行此類內部操作，則會取消 MySQL 清除和 PostgreSQL 自動清空等維護任務。如果 MySQL 事件排程器或 PostgreSQL `pg_cron` 延伸模組中的排程任務在 Aurora 啟動暫停操作時正在進行，也會取消這些任務。

  如果您需要確保執行個體定期喚醒以執行排程的操作，您可以在任務開始時間之前啟動連線以繼續執行個體。您也可以增加自動暫停逾時間隔，以便在使用者活動完成後，自動清空等操作可以執行更長的時間。您也可以使用 Lambda 函數之類的機制，以在必要時自動繼續執行個體的方式排程執行資料庫操作。

Aurora Serverless v2 自動暫停功能的應用程式設計考量

當 Aurora Serverless v2 資料庫執行個體在自動暫停後繼續時，它會從相對較小的容量開始，並從該容量向上擴展。即使資料庫執行個體在自動暫停之前有更高的容量，此起始容量也適用。

將此功能用於在建立連線時可容忍大約 15 秒間隔的應用程式。這會說明 Aurora Serverless v2 執行個體因一個或少量傳入連線而繼續的典型案例。如果執行個體暫停超過 24 小時，則繼續的時間可能會更長。

如果您的應用程式已在使用 Aurora Serverless v1 及其自動暫停功能，您可能已經有此類逾時間隔用於連線嘗試。如果您已經使用 Aurora Serverless v2 搭配 Aurora 停止/啟動叢集功能，自動暫停 Aurora Serverless v2 執行個體的繼續時間通常比啟動已停止叢集的時間短得多。

在應用程式中編碼連線邏輯時，如果第一次嘗試傳回具有暫時性原因的錯誤，請重試連線。(如果錯誤是由於驗證失敗，請在重試之前修正憑證。) 繼續後立即發生的錯誤可能是逾時，或與資料庫限制相關的一些錯誤。在極少數情況下，重試可以處理由於大量同時連線請求而繼續 Aurora Serverless v2 執行個體的問題。在這種情況下，某些連線可能需要比平常更長的時間來處理，或可能超過第一次嘗試時同時連線的限制。

在應用程式開發和偵錯期間，請勿讓用戶端工作階段或具有資料庫連線的程式設計工具保持開啟狀態。如果有任何使用者起始的連線開啟，無論連線是否未執行任何 SQL 陳述式或交易，Aurora 都不會暫停執行個體。當 Aurora 叢集中的一個 Aurora Serverless v2 執行個體無法暫停時，叢集中的其他執行個體也可能無法暫停。如需更多詳細資訊，請參閱 Aurora Serverless v2 不會自動暫停的情況。

當 Aurora Serverless v2 資料庫執行個體開始繼續、完成繼續，以及執行個體因為某些原因而無法繼續時，Aurora 會發出事件。如需這些事件的詳細資訊，請參閱 資料庫執行個體事件。

遷移至 Aurora Serverless v2

如要將現有資料庫叢及轉換為使用 Aurora Serverless v2，您可執行下列作業：

- 從佈建的 Aurora 資料庫叢集升級。
- 從 Aurora Serverless v1 叢集升級。
- 從內部部署資料庫遷移到 Aurora Serverless v2 叢集。

當升級後的叢集執行列於 Aurora Serverless v2 的要求與限制 中適當的引擎版本時，您可開始對其新增 Aurora Serverless v2 資料庫執行個體。您新增至升級叢集的第一個資料庫執行個體必須為佈建的資

料庫執行個體。然後，您可將寫入工作負載、讀取工作負載或兩者的處理切換至 Aurora Serverless v2 資料庫執行個體。

內容

- 升級或切換現有叢集以使用 Aurora Serverless v2
  - 要使用 Aurora Serverless v2 的 MySQL 相容叢集的升級路徑
  - 要使用 Aurora Serverless v2 之 PostgreSQL 相容叢集的升級路徑
- 從已佈建叢集切換至 Aurora Serverless v2
- 比較 Aurora Serverless v2 和 Aurora Serverless v1
  - 比較 Aurora Serverless v2 和 Aurora Serverless v1 要求
  - 比較 Aurora Serverless v2 和 Aurora Serverless v1 擴展和可用性
  - 比較 Aurora Serverless v2 和 Aurora Serverless v1 功能支援
  - 調節Aurora Serverless v1 使用案例為 Aurora Serverless v2
- 從 Aurora Serverless v1 叢集升級至 Aurora Serverless v2
  - Aurora MySQL 相容的資料庫叢集
  - Aurora PostgreSQL 相容的資料庫叢集
- 從內部部署資料庫遷移至 Aurora Serverless v2

> **Note**
>
> 下列主題說明如何轉換現有的資料庫叢集。如需建立新 建立使用 Aurora Serverless v2 的資料庫叢集 資料庫叢集的詳細資訊，請參閱 Aurora Serverless v2。

升級或切換現有叢集以使用 Aurora Serverless v2

若您佈建的叢集具有支援 Aurora Serverless v2 的引擎版本，則切換至不需要升級的 Aurora Serverless v2。於該狀況下，您可將 Aurora Serverless v2 資料庫執行個體新增至您的原始叢集。您可將叢集切換為使用所有的 Aurora Serverless v2 資料庫執行個體。您還可在相同的資料庫叢集中使用 Aurora Serverless v2 和佈建資料庫執行個體的組合。若為支援 Aurora Serverless v2 的 Aurora 引擎版本，請參閱 Aurora Serverless v2 支援的區域和 Aurora 資料庫引擎。

若您執行的是不支援 Aurora Serverless v2 的較低引擎版本，請執行以下一般步驟：

1. 升級叢集。

2. 為升級後的叢集建立佈建的寫入器資料庫執行個體。
3. 修改叢集來使用 Aurora Serverless v2 資料庫執行個體。

> **Important**
>
> 當您使用快照還原或複製，將主要版本升級為 Aurora Serverless v2 相容版本時，您新增至新叢集的第一個資料庫執行個體必須為佈建的資料庫執行個體。此新增會啟動升級過程的最後階段。
>
> 在最後階段發生之前，叢集並無 Aurora Serverless v2 支援所需的基礎設施。因此，這些升級後的叢集會一律以佈建的寫入器資料庫執行個體開始。然後，您可將佈建的資料庫執行個體轉換或故障轉移至 Aurora Serverless v2 執行個體。

從 Aurora Serverless v1 升級至 Aurora Serverless v2 涉及建立佈建叢集作為中間步驟。然後，您會執行與開始佈建叢集時相同的升級步驟。

要使用 Aurora Serverless v2 的 MySQL 相容叢集的升級路徑

若您的原始叢集正在執行 Aurora MySQL，請依據叢集的引擎版本和引擎模式選擇適當的程序。

若您的原始 Aurora MySQL 叢集是這個	請執行此操作以切換至 Aurora Serverless v2
佈建叢集，執行 Aurora MySQL 第 3 版，與 MySQL 8.0 相容	這是來自現有 Aurora MySQL 叢集所有轉換的最後階段。 如有必要，請執行次要版本升級至 3.02.0 或更高版本。為寫入器資料庫執行個體使用已佈建資料庫執行個體。新增一個 Aurora Serverless v2 讀取器資料庫執行個體。執行容錯移轉，以建立寫入器資料庫執行個體。 (選用) 將叢集中其他佈建的資料庫執行個體轉換為 Aurora Serverless v2。或者，新增 Aurora Serverless v2 資料庫執行個體並移除佈建的資料庫執行個體。 如需完整的程序和範例，請參閱 從已佈建叢集切換至 Aurora Serverless v2。

若您的原始 Aurora MySQL 叢集是這個	請執行此操作以切換至 Aurora Serverless v2
佈建叢集，執行 Aurora MySQL 第 2 版，與 MySQL 5.7 相容	執行主要版本升級至 Aurora MySQL 3.02.0 版或更新版本。然後依循 Aurora MySQL 第 3 版的程序，將叢集切換為使用 Aurora Serverless v2 資料庫執行個體。
Aurora Serverless v1 叢集，執行 Aurora MySQL 第 2 版，與 MySQL 5.7 相容	為了協助您規劃從 Aurora Serverless v1 轉換，請先洽詢 比較 Aurora Serverless v2 和 Aurora Serverless v1。 然後遵循 從 Aurora Serverless v1 叢集升級至 Aurora Serverless v2 中的程序。

要使用 Aurora Serverless v2 之 PostgreSQL 相容叢集的升級路徑

若您的原始叢集正在執行 Aurora PostgreSQL，請依據叢集的引擎版本和引擎模式選擇適當的程序。

若您的原始 Aurora PostgreSQL 叢集是這個	請執行此操作以切換至 Aurora Serverless v2
執行 Aurora PostgreSQL 第 13 版的已佈建叢集	這是來自現有 Aurora PostgreSQL 叢集所有轉換的最後階段。 如有必要，請執行次要版本升級至 13.6 版或更高版本。為寫入器資料庫執行個體新增一個已佈建資料庫執行個體。新增一個 Aurora Serverless v2 讀取器資料庫執行個體。執行容錯移轉，以建立 Aurora Serverless v2 執行個體為寫入器資料庫執行個體。 (選用) 將叢集中其他佈建的資料庫執行個體轉換為 Aurora Serverless v2。或者，新增 Aurora Serverless v2 資料庫執行個體並移除佈建的資料庫執行個體。 如需完整的程序和範例，請參閱 從已佈建叢集切換至 Aurora Serverless v2。

若您的原始 Aurora PostgreSQL 叢集是這個	請執行此操作以切換至 Aurora Serverless v2
執行 Aurora PostgreSQL 第 11 或 12 版的已佈建叢集	執行主要版本升級至 Aurora PostgreSQL 13.6 版或更新版本。然後依循 Aurora PostgreSQL 第 13 版的程序，將叢集切換為使用 Aurora Serverless v2 資料庫執行個體。
執行 Aurora PostgreSQL 第 11 或 13 版的 Aurora Serverless v1 叢集	為了協助您規劃從 Aurora Serverless v1 轉換，請先洽詢 比較 Aurora Serverless v2 和 Aurora Serverless v1。 然後遵循 從 Aurora Serverless v1 叢集升級至 Aurora Serverless v2 中的程序。

從已佈建叢集切換至 Aurora Serverless v2

如要使用 Aurora Serverless v2 切換已佈建叢集，請依循些步驟：

1. 檢查已佈建叢集是否需要升級才可與 Aurora Serverless v2 資料庫執行個體一起使用。若為與 Aurora Serverless v2 相容的 Aurora 版本，請參閱 Aurora Serverless v2 的要求與限制。

   若已佈建的叢集執行的引擎版本不適用於 Aurora Serverless v2，請升級叢集的引擎版本：

   - 若您擁有 MySQL 5.7 相容的佈建叢集，請依照 Aurora MySQL 第 3 版的升級說明進行作業。使用 就地升級執行方式 中的程序。
   - 若您有一個 PostgreSQL 相容的佈建叢集，執行 PostgreSQL 第 11 或 12 版，請依照 Aurora PostgreSQL 第 13 版的升級說明進行作業。使用 執行主要版本升級 中的程序。
2. 設定任何其他叢集屬性以符合來自 Aurora Serverless v2 的要求與限制 的 Aurora Serverless v2 要求。
3. 設定叢集的擴展組態。請遵循 設定叢集的 Aurora Serverless v2 容量範圍 中的程序。
4. 將一或多個 Aurora Serverless v2 資料庫執行個體新增至叢集。依循 將 Aurora 複本新增至資料庫叢集 中的一般程序。對於每個新的資料庫執行個體，請在 或 AWS 管理主控台或 AWS CLI Amazon RDS API `db.serverless`中指定特殊資料庫執行個體類別名稱 Serverless。

   在某些狀況下，叢集中可能已有一個或多個已佈建的讀取器資料庫執行個體。若是如此，您可將其中一個讀取器轉換為 Aurora Serverless v2 資料庫執行個體，而不是建立新的資料庫執行個體。若要執行此作業，請依照 將已佈建的寫入器或讀取器轉換為 Aurora Serverless v2 中的程序進行。

5. 執行容錯移轉作業以使其中一個 Aurora Serverless v2 資料庫執行個體作為叢集的寫入器資料庫執行個體。
6. (選用) 將任何已佈建的資料庫執行個體轉換為 Aurora Serverless v2，或從叢集進行移除。遵循 將已佈建的寫入器或讀取器轉換為 Aurora Serverless v2 或 從 Aurora 個體資料庫叢集刪除資料庫執行個體 中的一般程序。

> **Tip**
>
> 移除已佈建的資料庫執行個體並非強制性。您可以設定包含 Aurora Serverless v2 和已佈建的資料庫執行個體的叢集。但是，直到您熟悉 Aurora Serverless v2 資料庫執行個體的效能和擴展特性，我們建議您使用全部相同類型的資料庫執行個體設定叢集。

下列 AWS CLI 範例顯示使用執行 Aurora MySQL 3.02.0 版之佈建叢集的切換程序。叢集已命名為 `mysql-80`。叢集以兩個名為 `provisioned-instance-1` 和 `provisioned-instance-2` 的已佈建資料庫執行個體開始，一個寫入器和一個讀取器。他們都使用 `db.r6g.large` 資料庫執行個體類別。

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' --output text
mysql-80
provisioned-instance-2     False
provisioned-instance-1     True

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-1 \
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-1     db.r6g.large

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-2 \
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-2     db.r6g.large
```

我們建立一個包含一些資料的表格。這樣，我們就可以確認在切換前後叢集的資料和作業是相同的。

```
mysql> create database serverless_v2_demo;
mysql> create table serverless_v2_demo.demo (s varchar(128));
mysql> insert into serverless_v2_demo.demo values ('This cluster started with a
 provisioned writer.');
```

```
Query OK, 1 row affected (0.02 sec)
```

首先，我們將容量範圍新增至叢集。否則，我們在將任何 Aurora Serverless v2 資料庫執行個體新增到叢集時會發生錯誤。如果我們將 AWS 管理主控台 用於此程序，則在新增第一個Aurora Serverless v2 資料庫執行個體時，該步驟會自動執行。

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 \
  --db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-
mysql

An error occurred (InvalidDBClusterStateFault) when calling the CreateDBInstance
 operation:
Set the Serverless v2 scaling configuration on the parent DB cluster before creating a
 Serverless v2 DB instance.

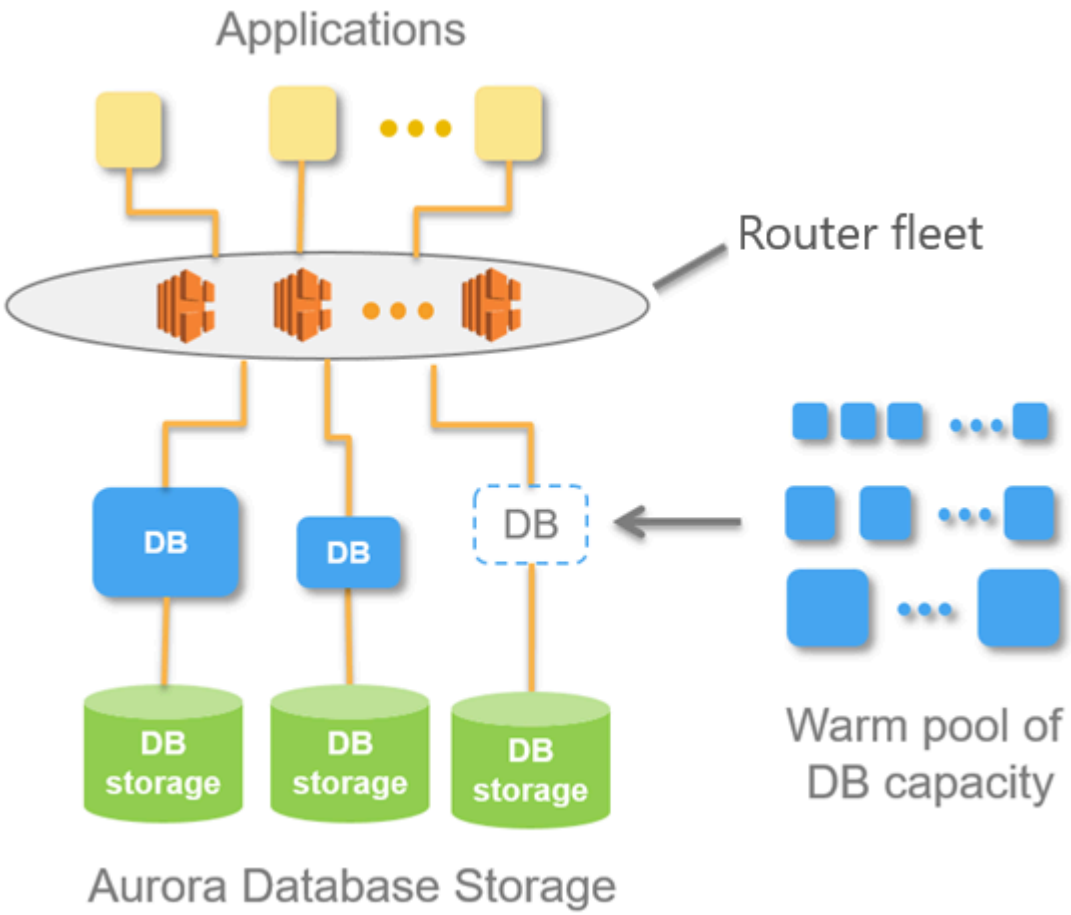
$ # The blank ServerlessV2ScalingConfiguration attribute confirms that the cluster
 doesn't have a capacity range set yet.
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 --query
 'DBClusters[*].ServerlessV2ScalingConfiguration'
[]

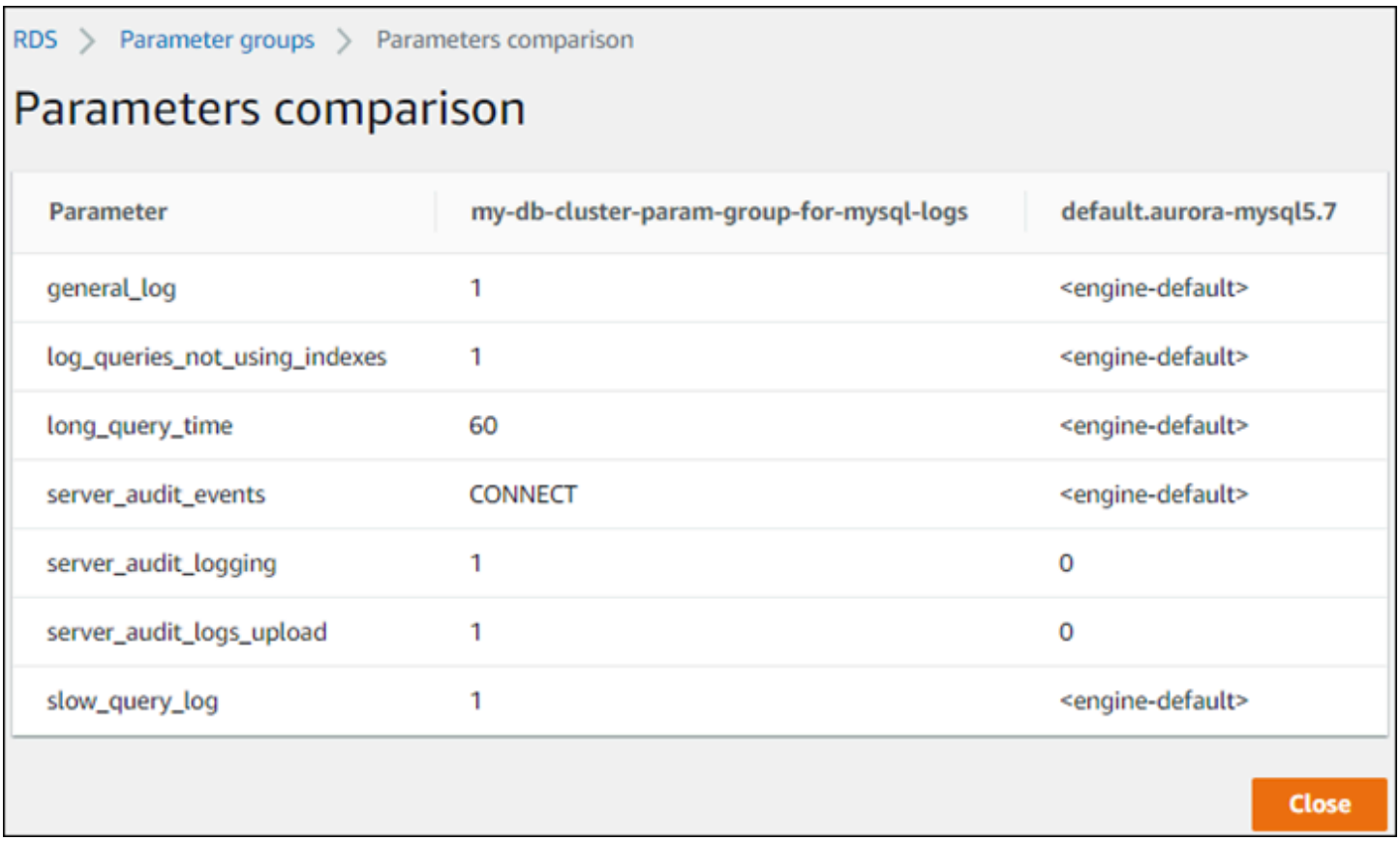
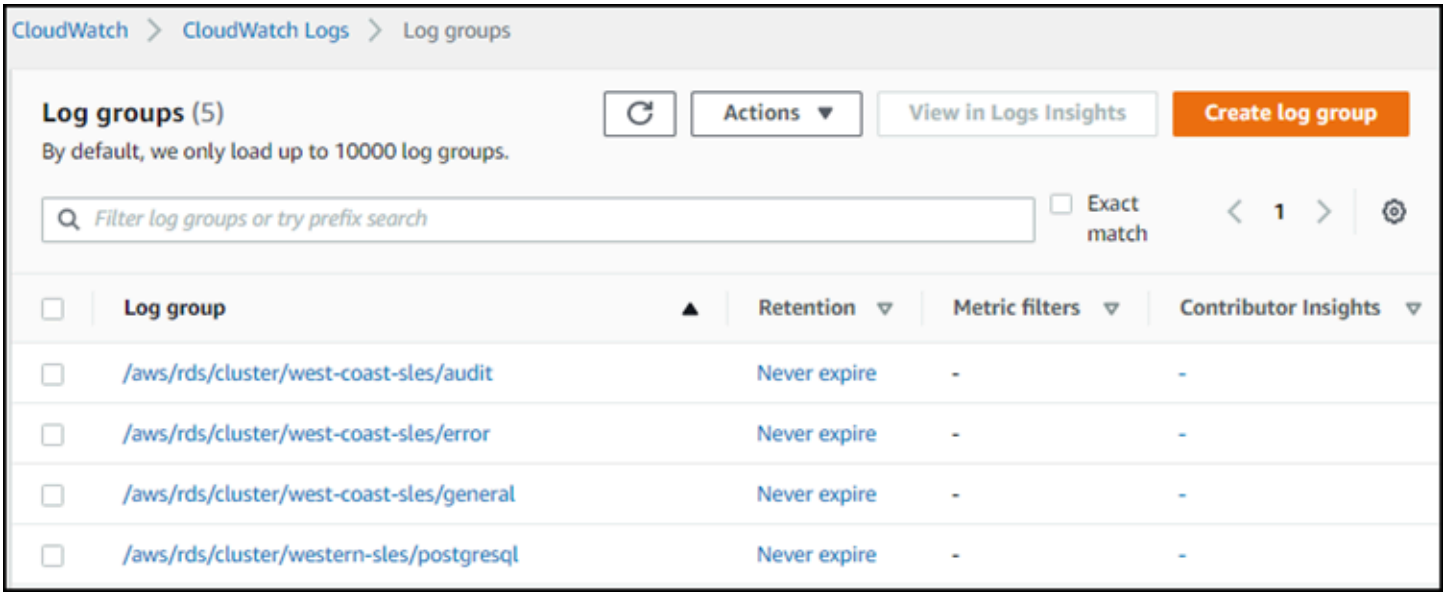
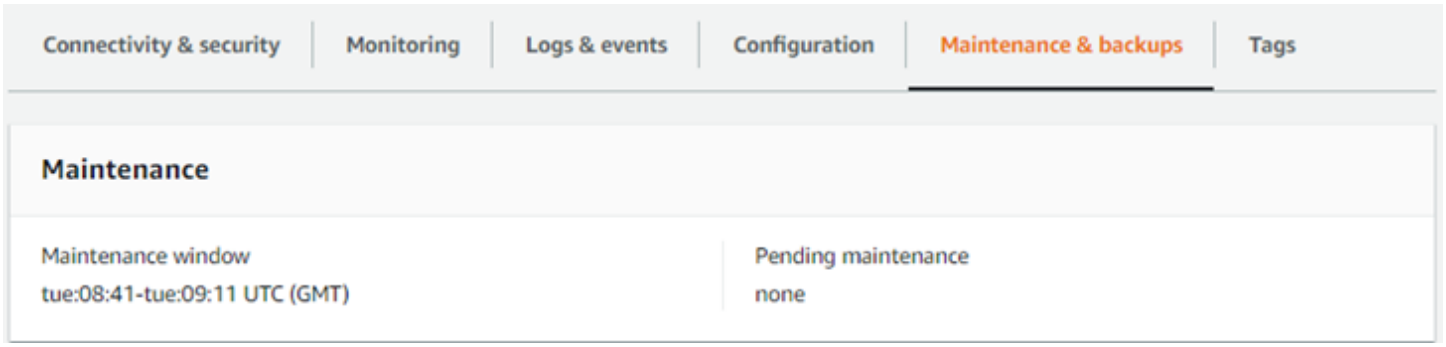
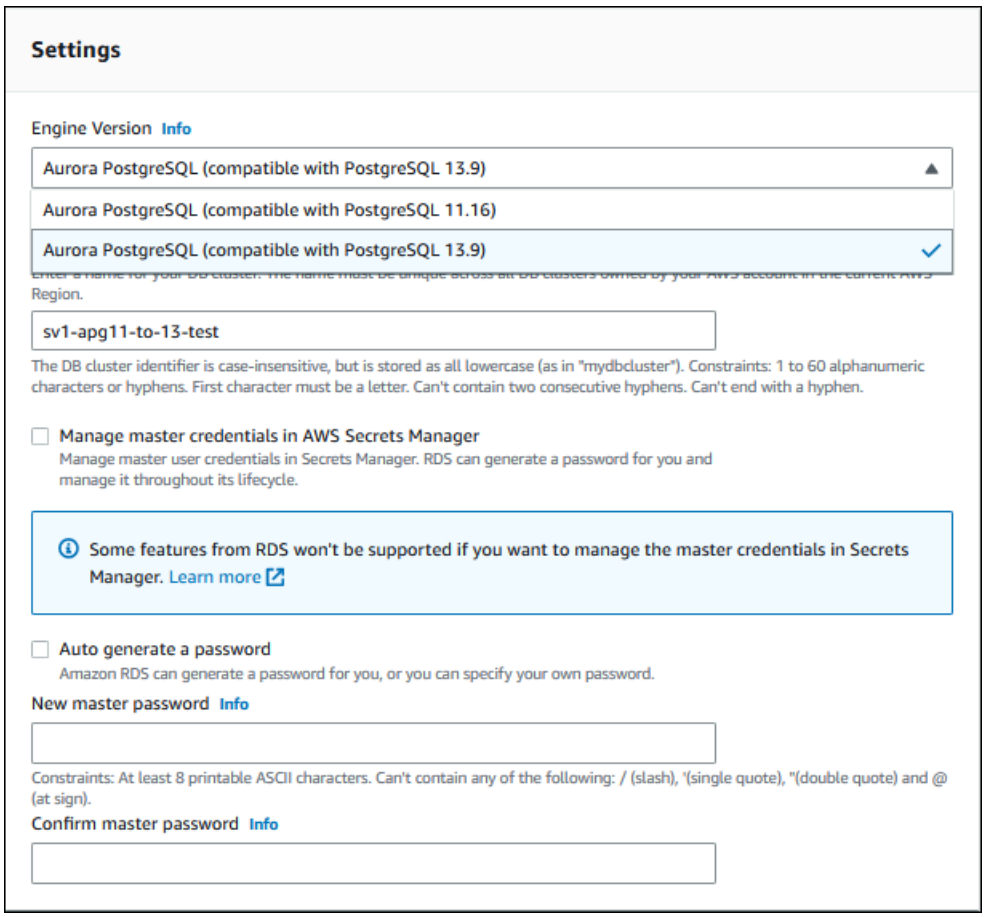
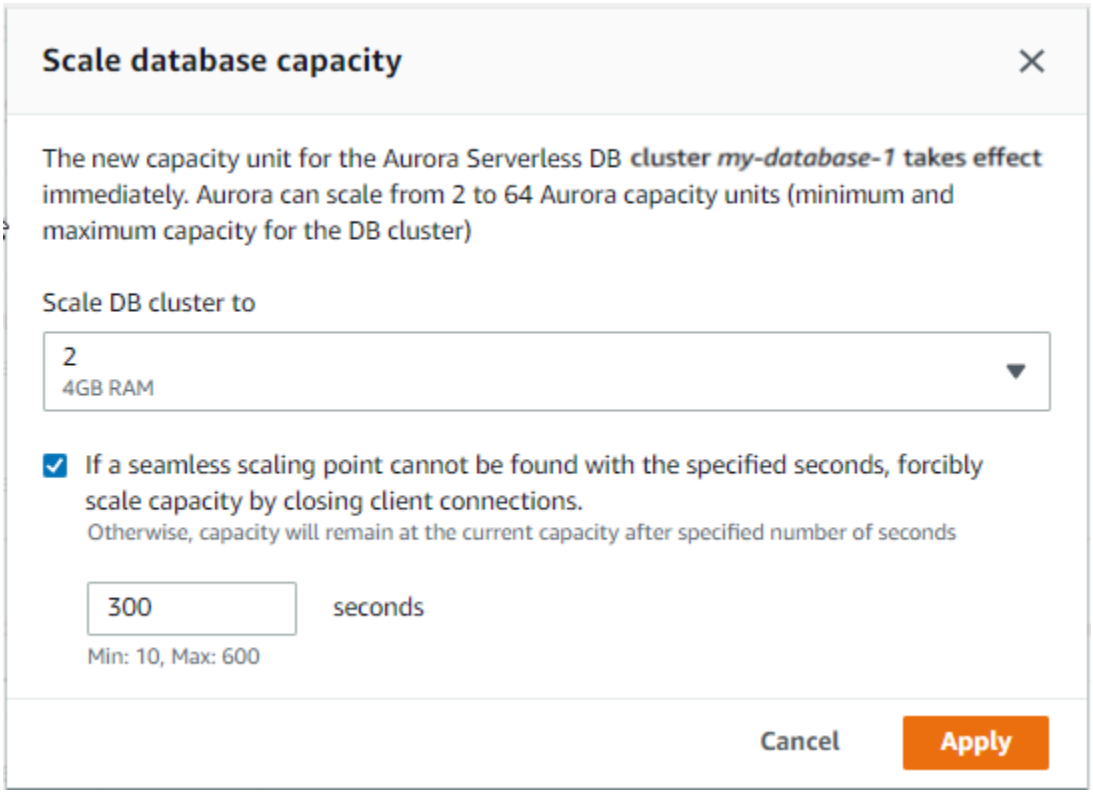
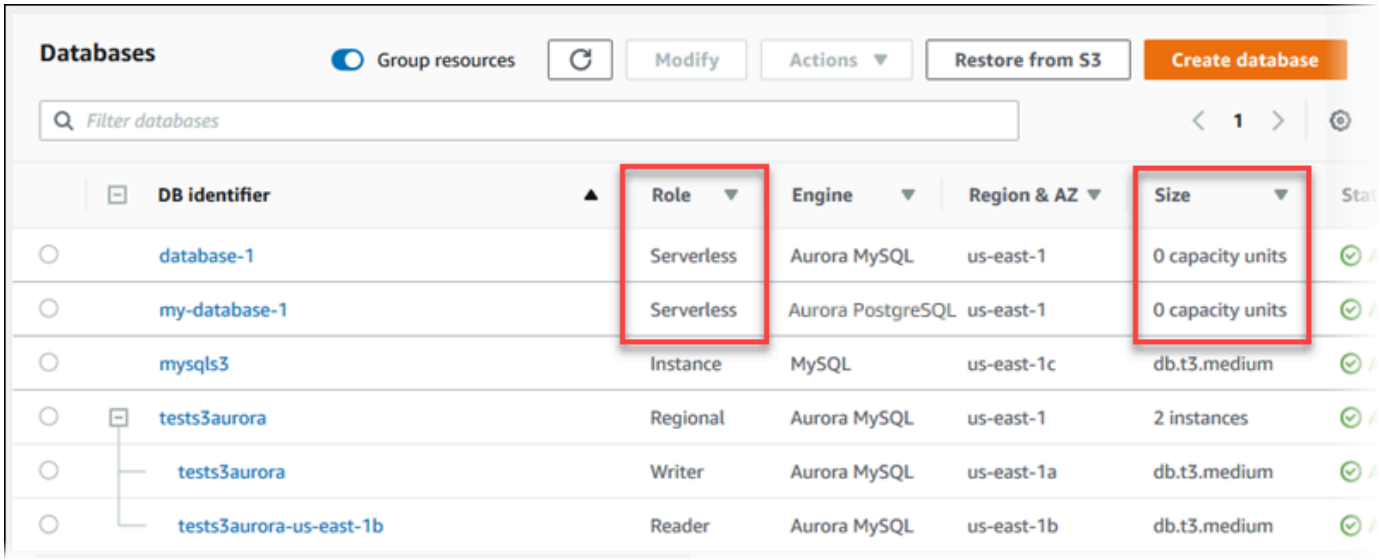
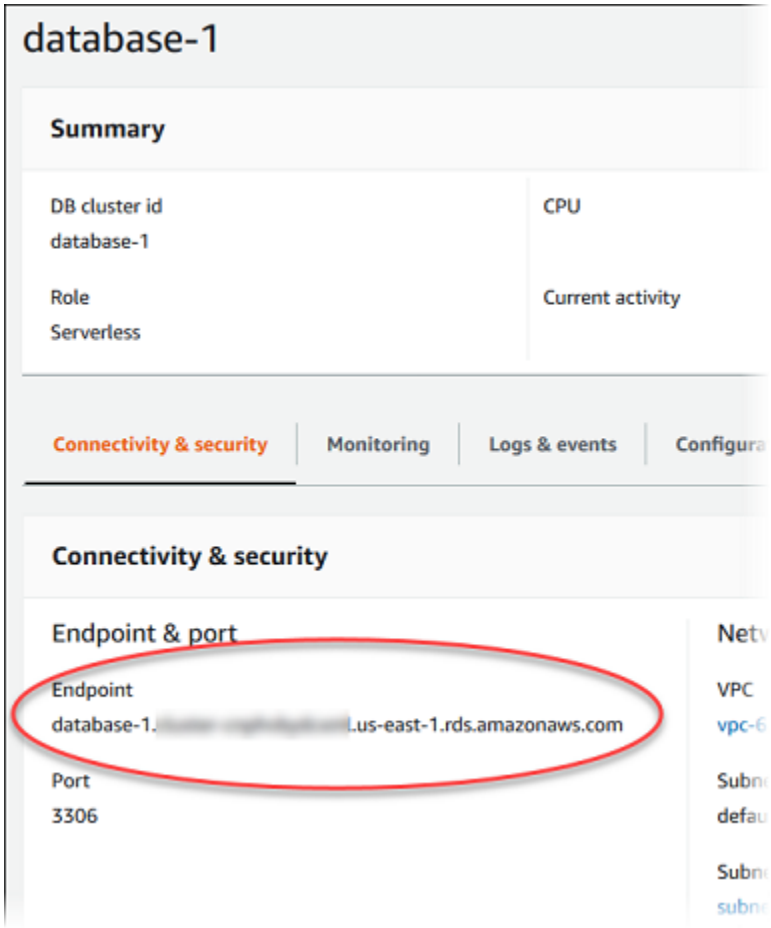
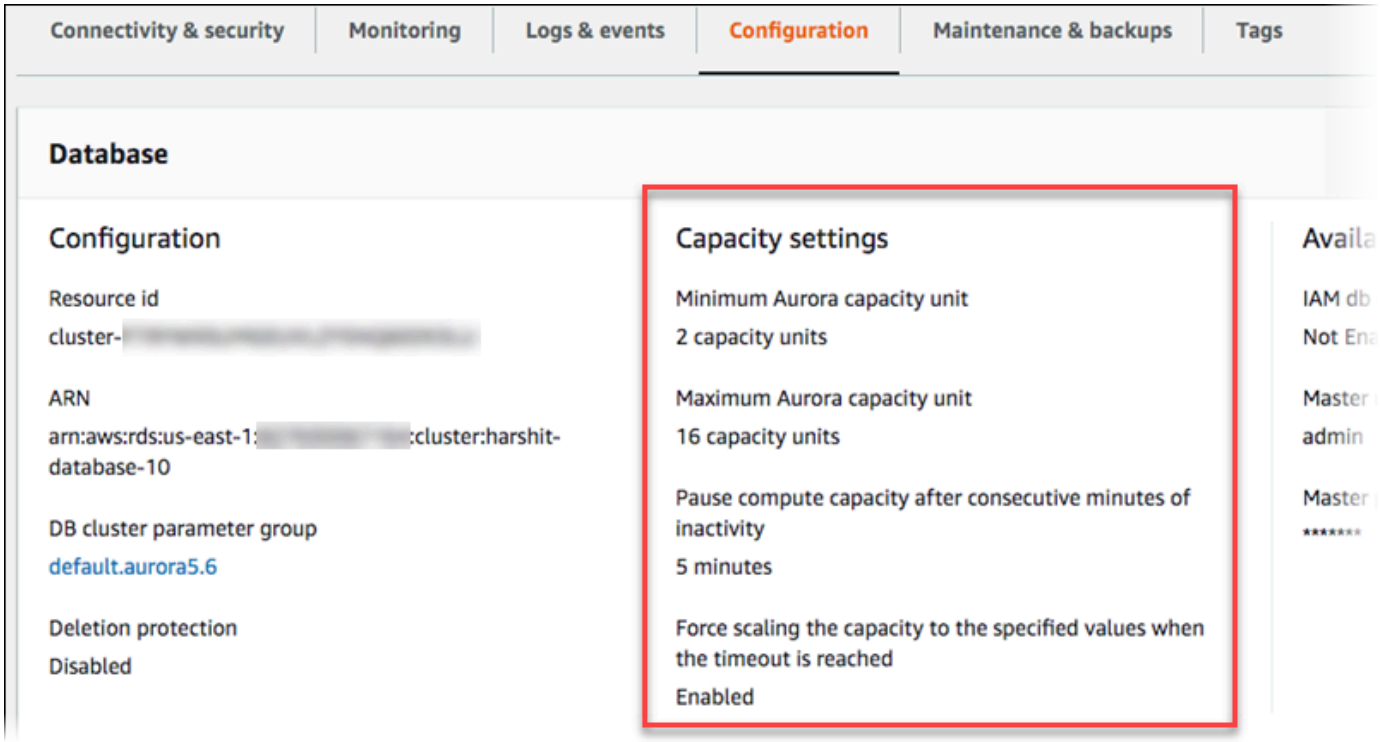
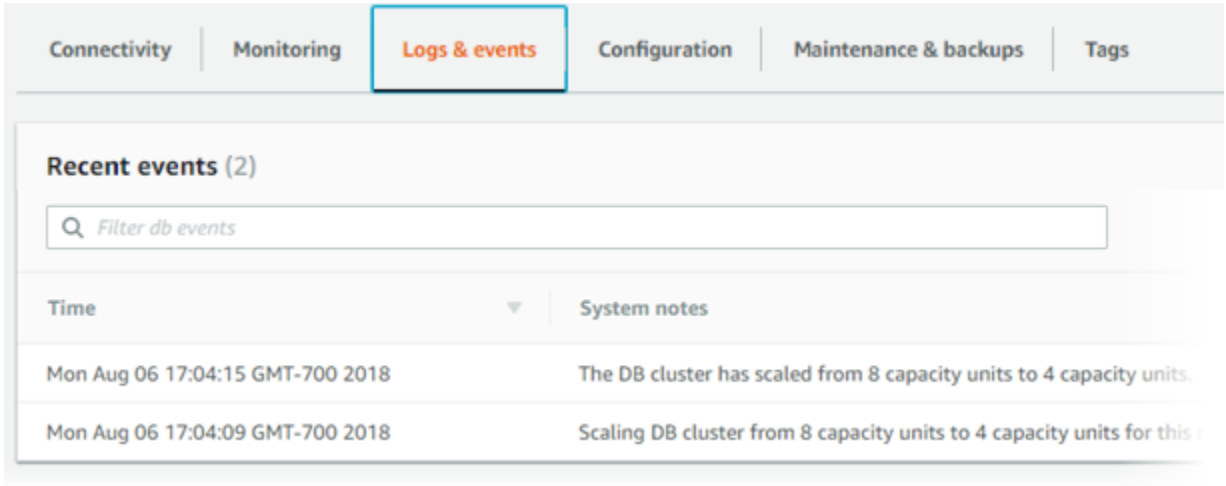
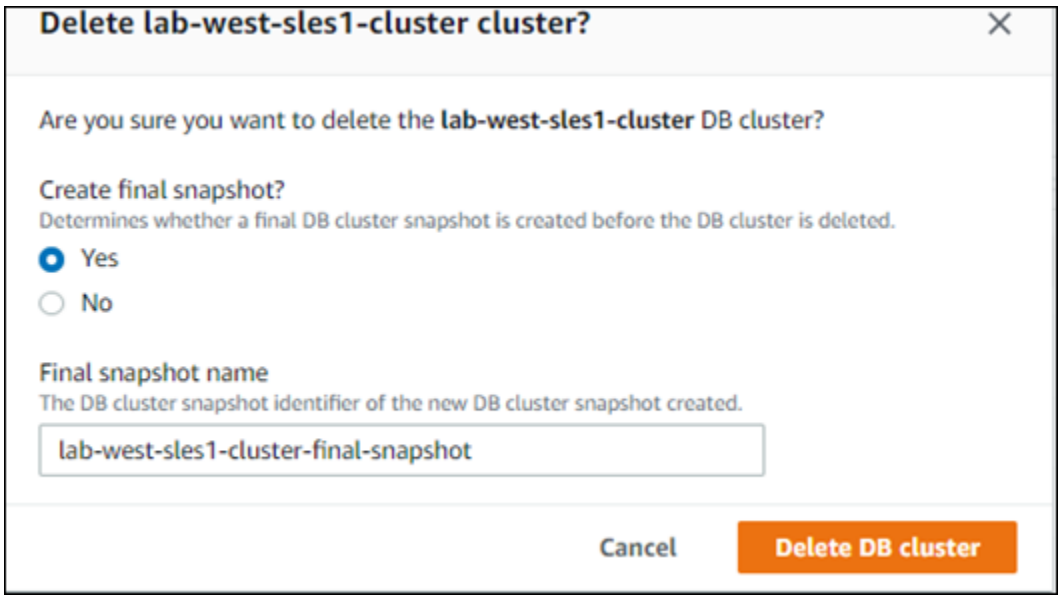
$ aws rds modify-db-cluster --db-cluster-identifier mysql-80 \
  --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16
{
  "DBClusterIdentifier": "mysql-80",
  "ServerlessV2ScalingConfiguration": {
    "MinCapacity": 0.5,
    "MaxCapacity": 16
  }
}
```

我們建立兩個 Aurora Serverless v2 讀取器來取代原始資料庫執行個體。做法是為新的資料庫執行個體指定 db.serverless 資料庫執行個體類別。

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 --db-
cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-mysql
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
  "DBInstanceStatus": "creating"
}
```

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-2 \
  --db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-
mysql
{
  "DBInstanceIdentifier": "serverless-v2-instance-2",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
  "DBInstanceStatus": "creating"
}

$ # Wait for both DB instances to finish being created before proceeding.
$ aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
 && \
  aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-2
```

我們執行容錯移轉以使其中一個 Aurora Serverless v2 資料庫執行個體作為叢集的新寫入器。

```
$ aws rds failover-db-cluster --db-cluster-identifier mysql-80 \
  --target-db-instance-identifier serverless-v2-instance-1
{
  "DBClusterIdentifier": "mysql-80",
  "DBClusterMembers": [
    {
      "DBInstanceIdentifier": "serverless-v2-instance-1",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "serverless-v2-instance-2",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "provisioned-instance-2",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "provisioned-instance-1",
```

```
      "IsClusterWriter": true,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    }
  ],
  "Status": "available"
}
```

變更需要幾秒鐘才會生效。在該點上，我們有一個 Aurora Serverless v2 寫入器和 Aurora Serverless v2 讀取器。因此，我們不需要任何一個原始已佈建的資料庫執行個體。

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' \
  --output text
mysql-80
serverless-v2-instance-1        True
serverless-v2-instance-2        False
provisioned-instance-2          False
provisioned-instance-1          False
```

切換過程的最後一步是刪除這兩個已佈建的資料庫執行個體。

```
$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-2 --skip-
final-snapshot
{
  "DBInstanceIdentifier": "provisioned-instance-2",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceClass": "db.r6g.large"
}

$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-1 --skip-
final-snapshot
{
  "DBInstanceIdentifier": "provisioned-instance-1",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceClass": "db.r6g.large"
}
```

於最後檢查，我們確認原始表格可從 Aurora Serverless v2 寫入器資料庫執行個體存取與寫入。

```
mysql> select * from serverless_v2_demo.demo;
+---------------------------------------------------+
| s                                                 |
+---------------------------------------------------+
| This cluster started with a provisioned writer.   |
+---------------------------------------------------+
1 row in set (0.00 sec)

mysql> insert into serverless_v2_demo.demo values ('And it finished with a Serverless
 v2 writer.');
Query OK, 1 row affected (0.01 sec)

mysql> select * from serverless_v2_demo.demo;
+---------------------------------------------------+
| s                                                 |
+---------------------------------------------------+
| This cluster started with a provisioned writer.   |
| And it finished with a Serverless v2 writer.      |
+---------------------------------------------------+
2 rows in set (0.01 sec)
```

我們也會連線到 Aurora Serverless v2 讀取器資料庫執行個體，並確認新寫入的資料在那裡也可用。

```
mysql> select * from serverless_v2_demo.demo;
+---------------------------------------------------+
| s                                                 |
+---------------------------------------------------+
| This cluster started with a provisioned writer.   |
| And it finished with a Serverless v2 writer.      |
+---------------------------------------------------+
2 rows in set (0.01 sec)
```

比較 Aurora Serverless v2 和 Aurora Serverless v1

若您已使用 Aurora Serverless v1，您可以了解 Aurora Serverless v1 和 Aurora Serverless v2 的主要差異。架構差異 (例如對讀取器資料庫執行個體的支援) 開啟了新的使用案例類型。

您可以使用下列表格，協助了解 Aurora Serverless v2 和 Aurora Serverless v1 最重要的差別。

主題

- 比較 Aurora Serverless v2 和 Aurora Serverless v1 要求
- 比較 Aurora Serverless v2 和 Aurora Serverless v1 擴展和可用性
- 比較 Aurora Serverless v2 和 Aurora Serverless v1 功能支援
- 調節Aurora Serverless v1 使用案例為 Aurora Serverless v2

比較 Aurora Serverless v2 和 Aurora Serverless v1 要求

下表彙總了使用 Aurora Serverless v2 或 Aurora Serverless v1 執行資料庫的不同要求。Aurora Serverless v2 提供比 Aurora Serverless v1 更高版本的 Aurora MySQL 和 Aurora PostgreSQL 資料庫引擎。

功能	Aurora Serverless v2 要求	Aurora Serverless v1 要求
資料庫引擎	Aurora MySQL、Aurora PostgreSQL	Aurora MySQL、Aurora PostgreSQL
支援的 Aurora MySQL 版本	請參閱 Aurora Serverless v2 搭配 Aurora MySQL。	請參閱 Aurora Serverless v1 搭配 Aurora MySQL。
支援的 Aurora PostgreSQL 版本	請參閱 Aurora Serverless v2 搭配 Aurora PostgreSQL。	請參閱 Aurora Serverless v1 搭配 Aurora PostgreSQL。
升級資料庫叢集	與佈建的資料庫叢集類似，您可以手動執行升級，無需等待 Aurora 為您升級資料庫叢集。如需詳細資訊，請參閱修改 Amazon Aurora 資料庫叢集。 **Note** 若要針對 Aurora PostgreSQL 相容的資料庫叢集執行從 13.x 到 14.x 或 15.x 的主要版本升級，叢集的最大	次要版本升級會在可用時自動套用。如需詳細資訊，請參閱Aurora Serverless v1 和 Aurora 資料庫引擎版本。 您可以手動執行主要版本升級。如需詳細資訊，請參閱修改 Aurora Serverless v1 資料庫叢集。

功能	Aurora Serverless v2 要求	Aurora Serverless v1 要求
	容量必須至少為 2 個 ACU。	
從已佈建資料庫叢集轉換	您可以使用下列方式： • 新增一或多個 Aurora Serverless v2 讀取器資料庫執行個體到現有的已佈建叢集。如要使用寫入器的 Aurora Serverless v2，請對其中一個 Aurora Serverless v2 資料庫執行個體執行容錯移轉。若為要使用 Aurora Serverless v2 資料庫執行個體的整個叢集，請在提升 Aurora Serverless v2 資料庫執行個體至寫入器後，移除任何已佈建寫入器。 • 以適當資料庫引擎和引擎版本建立新的叢集。使用任何標準方法。例如，還原叢集快照或建立現有叢集的複製。於新叢集中，為所有或部分資料庫執行個體選擇 Aurora Serverless v2。 如果透過複製建立新叢集，則無法同時升級引擎版本。確保原始集群已經在執行與 Aurora Serverless v2 相容的引擎版本。	還原已佈建叢集的快照，以建立新的 Aurora Serverless v1 叢集。
從 Aurora Serverless v1 叢集轉換	請遵循 從 Aurora Serverless v1 叢集升級至 Aurora Serverless v2 中的程序。	不適用

功能	Aurora Serverless v2 要求	Aurora Serverless v1 要求
可用的資料庫執行個體類別	特殊的資料庫執行個體類別 `db.serverless` 。在 中 AWS 管理主控台，它會標記為無伺服器。	不適用。Aurora Serverless v1 會使用 `serverless` 引擎模式。
連線埠	任何與 MySQL 或 PostgreSQL 相容的連線埠	僅限預設的 MySQL 或 PostgreSQL 連線埠
允許使用公有 IP 地址？	是	否
需要 Virtual private cloud (VPC)？	是	是。每個 Aurora Serverless v1 叢集會佔用分配給 VPC 的 2 個介面和閘道負載平衡器端點。

比較 Aurora Serverless v2 和 Aurora Serverless v1 擴展和可用性

下表彙總可擴展性和可用性的 Aurora Serverless v2 和 Aurora Serverless v1 間的差異。

比起 Aurora Serverless v2 中的擴展，Aurora Serverless v1 擴展的回應性更快、更精細，而且更少中斷。Aurora Serverless v2 可以透過變更資料庫執行個體的大小和向資料庫叢集新增更多資料庫執行個體來擴展。它也可以透過將其他 中的叢集新增至 Aurora 全域資料庫 AWS 區域 來擴展。相比之下，Aurora Serverless v1 只能透過增加或減少寫入器的容量來擴展。Aurora Serverless v1 叢集的所有運算都在單一可用區域和單一 AWS 區域上執行。

擴展和高可用性功能	Aurora Serverless v2 行為	Aurora Serverless v1 行為
Aurora 最小容量單位 (ACU) (Aurora MySQL)	叢集執行時為 0.5，叢集暫停時為 0。	叢集執行時為 1，叢集暫停時為 0。
最低 ACU (Aurora PostgreSQL)	叢集執行時為 0.5，叢集暫停時為 0。	叢集正在執行時為 2，叢集暫停時為 0。
最大 ACU (Aurora MySQL)	256	256

擴展和高可用性功能	Aurora Serverless v2 行為	Aurora Serverless v1 行為
最大 ACU (Aurora PostgreSQL)	256	384
停止叢集	您可以使用與已佈建叢集相同的叢集停止和啟動功能，手動停止和啟動叢集。	叢集在逾時後自動暫停。活動恢復時，需要一些時間才可使用。
資料庫執行個體的擴展	以 0.5 ACU 的最小增量縱向擴展或縮減	將 ACU 增加一倍或減半以進行縱向擴展或縮減
資料庫執行個體的數目	與已佈建叢集相同：1 個寫入器資料庫執行個體，最多 15 個讀取器資料庫執行個體。	1 個同時處理讀取和寫入的資料庫執行個體。
在 SQL 陳述式執行時可以進行擴展嗎？	是。Aurora Serverless v2 不需要等待一個安靜點。	否。例如，擴展等待長時間執行的交易、臨時表和表格鎖的完成。
讀取器資料庫執行個體隨寫入器一起擴展	選用	不適用
最大儲存空間	128 TiB	128 TiB
擴展時保留緩衝區快取	是。緩衝區快取是動態調整大小。	否。緩衝區快取在擴展後再加溫。
容錯移轉	是，與已佈建叢集相同。	僅盡最大努力，視容量可用性而定。比 Aurora Serverless v2 慢。
多可用區域功能	是，與已佈建的相同。多可用區域叢集需要第二個可用區域 (AZ) 中的讀取器資料庫執行個體。若為多可用區域叢集，Aurora 在可用區域故障時執行多可用區容錯移轉。	Aurora Serverless v1 叢集在單一可用區域中執行其所有運算。可用區域故障時的還原僅是盡最大努力，依容量可用性而定。

擴展和高可用性功能	Aurora Serverless v2 行為	Aurora Serverless v1 行為
Aurora 全球資料庫	是	否
根據記憶體壓力進行擴展	是	否
根據 CPU 負載進行擴展	是	是
根據網路流量進行擴展	是，依據網路流量的記憶體和 CPU 額外負荷。`max_connections` 參數保持不變，以避免在縮減規模時中斷連線。	是，根據連線數量。
擴展事件的逾時動作	否	是
透過 新增資料庫執行個體至叢集 AWS Auto Scaling	不適用。您可以在升級層 2–15 中建立 Aurora Serverless v2 讀取器資料庫執行個體，並將其縮減規模到低容量。	否。讀取器資料庫執行個體不可用。

比較 Aurora Serverless v2 和 Aurora Serverless v1 功能支援

下表總結了這些：

- 在 Aurora Serverless v2 中可用，但 Aurora Serverless v1 不可用的功能
- Aurora Serverless v1 和 Aurora Serverless v2 之間運作方式不同的功能
- 目前無法於 Aurora Serverless v2 中使用的功能

Aurora Serverless v2 包含來自已佈建叢集的許多功能，這些功能不適用於 Aurora Serverless v1。

功能	Aurora Serverless v2 支援	Aurora Serverless v1 支援
叢集拓撲	Aurora Serverless v2 是個別資料庫執行個體的屬性。叢集可以包含多個 Aurora Serverless v2 資料庫執行個體，或 Aurora	Aurora Serverless v1 叢集不使用資料庫執行個體的概念。建立叢集後無法變更 Aurora Serverless v1 屬性。

功能	Aurora Serverless v2 支援	Aurora Serverless v1 支援
	Serverless v2 和已佈建資料庫執行個體的組合。	
組態參數	幾乎所有相同的參數皆可修改為與已佈建叢集中一樣。如需詳細資訊，請參閱使用 Aurora Serverless v2 的參數群組。	僅參數的子集可以修改。
參數群組	叢集參數群組和資料庫參數群組。`SupportedEngineModes` 屬性中具 `provisioned` 值的參數可用。這比 Aurora Serverless v1 中的參數多得多。	僅限叢集參數群組。`SupportedEngineModes` 屬性中具 `serverless` 值的參數可用。
叢集磁碟區的加密	選用	必要. Amazon Aurora 加密資料庫叢集的限制 中的限制套用至所有 Aurora Serverless v1 叢集。
跨區域快照	是	快照必須使用您自己的 AWS Key Management Service (AWS KMS) 金鑰加密。
資料庫叢集刪除後保留的自動備份	是	否
TLS/SSL	是。支援與已佈建叢集的支援相同。如需使用方式的資訊，請參閱 搭配 Aurora Serverless v2 使用 TLS/SSL。	是。與對已佈建叢集的 TLS 支援存在一些差異。如需使用方式的資訊，請參閱 搭配 Aurora Serverless v1 使用 TLS/SSL。
複製	僅與 Aurora Serverless v2 相容的開始和結束資料庫引擎版本。您不能使用複製從Aurora Serverless v1或來自早期版本的已佈建叢集。	僅與 Aurora Serverless v1 相容的開始和結束資料庫引擎版本。

功能	Aurora Serverless v2 支援	Aurora Serverless v1 支援
與 Amazon S3 整合	是	是
與 整合 AWS Secrets Manager	是	否
將資料庫叢集快照資料匯出至 S3	是	否
建立 IAM 角色的關聯	是	否
將 日誌上傳到 Amazon CloudWatch	選用。您可以選擇要開啟的日誌及要上傳至 CloudWatch 的日誌。	所有已開啟的日誌都會自動上傳至 CloudWatch。
可使用的資料 API	是	是
可用的查詢編輯器	是	是
Performance Insights	是	否
可用的 Amazon RDS Proxy	是	否
可使用 Babelfish for Aurora PostgreSQL	是。支援同時與 Babelfish 和 Aurora Serverless v2 相容的 Aurora PostgreSQL 版本。	否

調節Aurora Serverless v1 使用案例為 Aurora Serverless v2

根據 Aurora Serverless v1 的使用案例，您可調整該方法以利用 Aurora Serverless v2 功能，如下所示。

假設您有一個負載較輕的 Aurora Serverless v1 叢集，且您的首要任務是保持連續可用性，同時最大限度地降低成本。利用 Aurora Serverless v2，您可以設定 0.5 的最小 ACU 設定，而 Aurora Serverless v1 的最小 ACU 設定為 1。您可以透過建立多可用區域組態來提高可用性，讀取器資料庫執行個體也至少具有 0.5 個 ACU。

假設您有一個在開發和測試案例中使用的 Aurora Serverless v1 叢集。在這種情況下，成本也是高優先順序，但叢集不需要隨時可用。Aurora Serverless v2 可以在完全閒置時自動暫停每個執行個體。您可

以透過為叢集指定 0 個 ACU 的最小容量來執行此操作，如 使用 Aurora Serverless v2 的自動暫停和繼續擴展至零個 ACU 中所述。您也可以在不需要時手動停止叢集，並在下一個測試或開發週期時進行啟動。

假設您有一個工作量很大的 Aurora Serverless v1 叢集。使用 Aurora Serverless v2 的等效叢集可以更細緻地進行擴展。例如：Aurora Serverless v1 透過將容量增加一倍來進行擴展，例如從 64 個 ACU 增加到 128 個 ACU。相反地，您的 Aurora Serverless v2 資料庫執行個體可以 0.5-ACU 增量進行縮減。

假設您的工作負載需要比 Aurora Serverless v1 中更高的總容量。您可以使用多個 Aurora Serverless v2 讀取器資料庫執行個體，從寫入器資料庫執行個體中卸載讀取密集型工作負載部分。您還可以在多個讀取器資料庫執行個體之間分割讀取密集型工作負載。

對於寫入密集型工作負載，您可以將具有大型已佈建資料庫執行個體的叢集設定為寫入器。您可能會針對一或多個 Aurora Serverless v2 讀取器資料庫執行個體這樣做。

從 Aurora Serverless v1 叢集升級至 Aurora Serverless v2

> **Important**
>
> AWS 已宣布 的end-of-lifeAurora Serverless v1：2025 年 3 月 31 日。所有未在 2025 年 3 月 31 日之前遷移的 Aurora Serverless v1 叢集都會在維護時段期間遷移至 Aurora Serverless v2。如果升級失敗，Amazon Aurora 會在維護時段將 Serverless v1 叢集轉換為具有同等引擎版本的佈建叢集。如果適用，Amazon Aurora 將在 Amazon RDS 延長支援中註冊轉換後的已佈建叢集。如需詳細資訊，請參閱RDS 延長支援。

將資料庫叢集從 Aurora Serverless v1 升級至 Aurora Serverless v2 的程序包含多個步驟。這是因為您無法直接從 Aurora Serverless v1 轉換為 Aurora Serverless v2。當中一律包含將 Aurora Serverless v1 資料庫叢集轉換為已佈建叢集的中繼步驟。

Aurora MySQL 相容的資料庫叢集

您可以將 Aurora Serverless v1 資料庫叢集轉換為佈建的資料庫叢集，然後使用藍/綠部署進行升級，並將其轉換為 Aurora Serverless v2 資料庫叢集。建議生產環境使用此程序。如需詳細資訊，請參閱使用 Amazon Aurora 藍/綠部署進行資料庫更新。

若要使用藍/綠部署，請升級執行 Aurora MySQL 第 2 版 (MySQL 5.7 相容) 的 Aurora Serverless v1 叢集

1. 將 Aurora Serverless v1 資料庫叢集轉換為已佈建 Aurora MySQL 第 2 版叢集。請遵循 從 Aurora Serverless v1 轉換為已佈建 中的程序。
2. 建立藍/綠部署 請遵循 在 Amazon Aurora 中建立藍/綠部署 中的程序。
3. 為與 Aurora Serverless v2 相容的綠色叢集選擇 Aurora MySQL 版本，例如 3.04.1。

   如需相容版本，請參閱 Aurora Serverless v2 搭配 Aurora MySQL。
4. 修改綠色叢集的寫入器資料庫執行個體，以使用 Serverless v2 (db.serverless) 資料庫執行個體類別。

   如需詳細資訊，請參閱將已佈建的寫入器或讀取器轉換為 Aurora Serverless v2。
5. 當您升級的 Aurora Serverless v2 資料庫叢集可用時，請從藍色叢集切換到綠色叢集。

Aurora PostgreSQL 相容的資料庫叢集

您可以將 Aurora Serverless v1 資料庫叢集轉換為佈建的資料庫叢集，然後使用藍/綠部署進行升級，並將其轉換為 Aurora Serverless v2 資料庫叢集。建議生產環境使用此程序。如需詳細資訊，請參閱使用 Amazon Aurora 藍/綠部署進行資料庫更新。

若要使用藍/綠部署，請升級執行 Aurora PostgreSQL 第 11 版的 Aurora Serverless v1 叢集

1. 將 Aurora Serverless v1 資料庫叢集轉換為已佈建 Aurora PostgreSQL 叢集。請遵循 從 Aurora Serverless v1 轉換為已佈建 中的程序。
2. 建立藍/綠部署 請遵循 在 Amazon Aurora 中建立藍/綠部署 中的程序。
3. 為與 Aurora Serverless v2 相容的綠色叢集選擇 Aurora PostgreSQL 版本，例如 15.3。

   如需相容版本，請參閱 Aurora Serverless v2 搭配 Aurora PostgreSQL。
4. 修改綠色叢集的寫入器資料庫執行個體，以使用 Serverless v2 (db.serverless) 資料庫執行個體類別。

   如需詳細資訊，請參閱將已佈建的寫入器或讀取器轉換為 Aurora Serverless v2。
5. 當您升級的 Aurora Serverless v2 資料庫叢集可用時，請從藍色叢集切換到綠色叢集。

您也可以將 Aurora Serverless v1 資料庫叢集直接從 Aurora PostgreSQL 第 11 版升級至第 13 版，將其轉換為佈建的資料庫叢集，然後將佈建的叢集轉換為 Aurora Serverless v2 資料庫叢集。

若要升級並轉換執行 Aurora PostgreSQL 第 11 版的 Aurora Serverless v1 叢集

1. 將 Aurora Serverless v1 資料庫叢集轉換為已佈建 Aurora PostgreSQL 叢集。請遵循 從 Aurora Serverless v1 轉換為已佈建 中的程序。
2. 將 Aurora Serverless v1 叢集升級為與 Aurora Serverless v2 相容的 Aurora PostgreSQL 第 13 版 (例如第 13.12 版)。請遵循 升級主要版本 中的程序。

   如需相容版本，請參閱 Aurora Serverless v2 搭配 Aurora PostgreSQL。
3. 新增 Aurora Serverless v2 讀取器資料庫執行個體到叢集中。如需詳細資訊，請參閱新增 Aurora Serverless v2 讀取器。
4. 容錯移轉至 Aurora Serverless v2 資料庫執行個體：

   a. 選取資料庫叢集的寫入器資料庫執行個體。

   b. 針對 Actions (動作)，選擇 Failover (容錯移轉)。

   c. 在確認頁面上，選擇容錯移轉。

對於執行 Aurora PostgreSQL 第 13 版的 Aurora Serverless v1 資料庫叢集，您可以將 Aurora Serverless v1 叢集轉換為佈建的資料庫叢集，然後將佈建的叢集轉換為 Aurora Serverless v2 資料庫叢集。

升級執行 Aurora PostgreSQL 第 13 版的 Aurora Serverless v1 叢集

1. 將 Aurora Serverless v1 資料庫叢集轉換為已佈建 Aurora PostgreSQL 叢集。請遵循 從 Aurora Serverless v1 轉換為已佈建 中的程序。
2. 新增 Aurora Serverless v2 讀取器資料庫執行個體到叢集中。如需詳細資訊，請參閱新增 Aurora Serverless v2 讀取器。
3. 容錯移轉至 Aurora Serverless v2 資料庫執行個體：

   a. 選取資料庫叢集的寫入器資料庫執行個體。

   b. 針對 Actions (動作)，選擇 Failover (容錯移轉)。

   c. 在確認頁面上，選擇容錯移轉。
4. 移除讀取器執行個體。

從內部部署資料庫遷移至 Aurora Serverless v2

您可以將內部部署資料庫遷移到 Aurora Serverless v2，如同佈建 Aurora MySQL 和 Aurora PostgreSQL 一般。

- 對於 MySQL 資料庫，您可以使用 `mysqldump` 命令。如需詳細資訊，請參閱 Amazon Relational Database Service 使用者指南中的將資料匯入至 Amazon RDS for MySQL 資料庫，並減少停機時間。
- 對於 PostgreSQL 資料庫，您可以使用 `pg_dump` 和 `pg_restore` 命令。如需詳細資訊，請參閱部落格貼文 Best practices for migrating PostgreSQL databases to Amazon RDS and Amazon Aurora (將 PostgreSQL 資料庫遷移到 Amazon RDS 和 Amazon Aurora 的最佳實務)。

使用 Amazon Aurora Serverless v1

> **Important**
>
> AWS 已宣布 的end-of-lifeAurora Serverless v1：2025 年 3 月 31 日。所有未在 2025 年 3 月 31 日之前遷移的 Aurora Serverless v1 叢集都會在維護時段期間遷移至 Aurora Serverless v2。如果升級失敗，Amazon Aurora 會在維護時段將 Serverless v1 叢集轉換為具有同等引擎版本的佈建叢集。如果適用，Amazon Aurora 將在 Amazon RDS 延長支援中註冊轉換後的已佈建叢集。如需詳細資訊，請參閱RDS 延長支援。

Amazon Aurora Serverless v1 (Amazon Aurora Serverless 第 1 版) 是適用於 Amazon Aurora 的隨需自動擴展組態。Aurora Serverless v1 資料庫叢集是一種資料庫叢集，可根據應用程式的需求增加和縮減運算容量。這與 Aurora 佈建的資料庫叢集相反，後者的容量管理是手動進行的。Aurora Serverless v1 為頻率較低、間歇性或不可預測的工作負載提供相對簡單、符合成本效益的選項。它可自動啟動、擴展運算容量以符合您的應用程式用量，並在不使用時關閉，因此具有成本效益。

若要進一步了解定價，請參閱 Amazon Aurora pricing 頁面上 MySQL-Compatible Edition (MySQL 相容版本) 或 PostgreSQL-Compatible Edition (PostgreSQL 相容版本) 底下的 無伺服器定價。

Aurora Serverless v1 叢集與已佈建資料庫叢集所使用的高容量、分散式及高可用性儲存磁碟區是同一種。

若為 Aurora Serverless v1 叢集，叢集磁碟區一律加密。您可以選擇加密金鑰，但無法停用加密。這表示您在 Aurora Serverless v1 上執行的操作與您在加密快照上執行的操作是相同的。如需詳細資訊，請參閱Aurora Serverless v1 和快照。

主題

- Aurora Serverless v1 的區域和版本可用性
- Aurora Serverless v1 的優點
- Aurora Serverless v1 的應用案例
- Aurora Serverless v1 的限制
- Aurora Serverless v1 的組態需求
- 搭配 Aurora Serverless v1 使用 TLS/SSL

- Aurora Serverless v1 的運作方式
- 建立 Aurora Serverless v1 資料庫叢集
- 還原 Aurora Serverless v1 資料庫叢集
- 修改 Aurora Serverless v1 資料庫叢集
- 手動擴展 Aurora Serverless v1 資料庫叢集容量
- 檢視 Aurora Serverless v1 資料庫叢集
- 刪除 Aurora Serverless v1 資料庫叢集
- Aurora Serverless v1 和 Aurora 資料庫引擎版本

Aurora Serverless v1 的區域和版本可用性

功能可用性和支援會因每個 Aurora 資料庫引擎的特定版本以及 AWS 區域而有所不同。如需 Aurora 和 Aurora Serverless v1 版本和區域可用性的詳細資訊，請參閱 Aurora Serverless v1。

Aurora Serverless v1 的優點

Aurora Serverless v1 提供下列優點：

- 比佈建簡單 – Aurora Serverless v1 可免除管理資料庫執行個體和容量的大部分複雜性。
- 可擴展性 – Aurora Serverless v1 可在不干擾用戶端連線的情況下，視需要無縫擴展運算與記憶體容量。
- 具成本效益 – 當您使用 Aurora Serverless v1 時，您僅需為使用的資料庫資源付費 (以秒計費)。
- 高可用性儲存 – Aurora Serverless v1 透過六路徑複寫使用相同的容錯、分散式儲存系統做為 Aurora 以防止資料遺失。

Aurora Serverless v1 的應用案例

Aurora Serverless v1 是專為以下應用案例所設計：

- 不常使用的應用程式：您擁有一個每天或每週僅使用數次，且每次僅使用幾分鐘的應用程式 (例如低容量的部落格網站)。使用 Aurora Serverless v1 時，您僅需為使用的資料庫資源付費 (以秒計費)。
- 新的應用程式：您正在部署新的應用程式，但不確定所需的執行個體大小。透過使用 Aurora Serverless v1，您就可以建立資料庫端點，並將該資料庫自動擴展至應用程式所需容量。

- 變數工作負載：您正在執行輕度使用量的應用程式，即每日僅有從 30 分鐘至數小時不等的幾次峰值，或每年僅有數次的使用次數。例如人力資源、預算以及營運報告等應用程式。透過 Aurora Serverless v1，您不再需要針對峰值或平均容量進行佈建。
- 無法預測的工作負載：您正在執行的每日工作負載都會有無法預測的突發活動增加。例如一個交通站點，在開始下雨時會看到活動突然增加。透過 Aurora Serverless v1，您的資料庫會自動擴展容量以滿足應用程式峰值負載需求，並在活動激增結束時縮小規模。
- 開發和測試資料庫：您的開發人員會在上班時間使用資料庫，但不會在夜間與週末使用。透過 Aurora Serverless v1，您的資料庫將在未使用時自動關閉。
- 多租用戶應用程式 – 透過 Aurora Serverless v1，您就無須為機群中的每個應用程式分別管理資料庫容量。Aurora Serverless v1 將為您管理個別資料庫容量。

Aurora Serverless v1 的限制

下列限制適用於 Aurora Serverless v1：

- **Important**

  AWS 已宣布 的end-of-lifeAurora Serverless v1：2025 年 3 月 31 日。所有未在 2025 年 3 月 31 日之前遷移的 Aurora Serverless v1 叢集都會在維護時段期間遷移至 Aurora Serverless v2。如果升級失敗，Amazon Aurora 會在維護時段將 Serverless v1 叢集轉換為具有同等引擎版本的佈建叢集。如果適用，Amazon Aurora 將在 Amazon RDS 延長支援中註冊轉換後的已佈建叢集。如需詳細資訊，請參閱RDS 延長支援。

- Aurora Serverless v1 不支援以下功能：
  - Aurora 全球資料庫
  - Aurora 複本
  - AWS Identity and Access Management (IAM) 資料庫身分驗證
  - Aurora 中的回溯功能
  - 資料庫活動串流
  - Kerberos 身分驗證
  - Performance Insights
  - RDS Proxy
  - 在 中檢視日誌 AWS 管理主控台
- 如果保持開啟的時間超過一天，與 Aurora Serverless v1 資料庫叢集的連線會自動關閉。

- 所有 Aurora Serverless v1 資料庫叢集都有下列限制：
  - 您無法將 Aurora Serverless v1 快照匯出至 Amazon S3 儲存貯體。
  - 您無法搭配資料庫叢集使用 AWS Database Migration Service 和變更資料擷取 Aurora Serverless v1 (CDC)。只有佈建的 Aurora 資料庫叢集支援使用 AWS DMS 做為來源的 CDC。
  - 您無法將資料儲存至 Amazon S3 中的文字檔，也無法將文字檔資料從 S3 載入至 Aurora Serverless v1。
  - 您無法將 IAM 角色附加至 Aurora Serverless v1 資料庫叢集。但是，您可以透過使用具有 Aurora Serverless v1 函數和 `aws_s3` 參數的 `aws_s3.table_import_from_s3` 延伸，將資料從 Amazon S3 載入至 `credentials`。如需詳細資訊，請參閱將資料從 Amazon S3 匯入 Aurora PostgreSQL 資料庫叢集 。
  - 使用查詢編輯器時，會為資料庫憑證建立 Secrets Manager 密碼，以便存取資料庫。如果您從查詢編輯器刪除憑證，相關聯的密碼也會從 Secret Manager 中刪除。刪除此密碼後就無法復原。
- 執行 Aurora Serverless v1、以 Aurora MySQL 為基礎的資料庫叢集不支援以下項目：
  - 從 Aurora MySQL 資料庫叢集內叫用 AWS Lambda 函數。不過， AWS Lambda 函數可以呼叫資料庫Aurora Serverless v1叢集。
  - 從非 Aurora MySQL 或 RDS for MySQL 的資料庫執行個體還原快照。
  - 使用以二進位日誌 (binlog) 為基礎的複寫複寫資料。無論是以 Aurora MySQL 為基礎的資料庫叢集 Aurora Serverless v1 是複寫的來源或目標，這項限制都是存在的。若要將資料從 Aurora 外部的 MySQL 資料庫執行個體 (例如在 Amazon EC2 上執行的執行個體) 複寫到 Aurora Serverless v1 資料庫叢集，建議您考慮使用 AWS Database Migration Service。如需詳細資訊，請參閱「AWS Database Migration Service 使用者指南」。
  - 建立具有以主機為基礎之存取權限的使用者 (`'username'@'IP_address'`)。這是因為 Aurora Serverless v1 使用用戶端和資料庫主機之間的路由器機群進行無縫擴展。Aurora Serverless 資料庫叢集看到的 IP 地址是路由器主機，而不是您的用戶端。如需詳細資訊，請參閱Aurora Serverless v1 架構。

    請改用萬用字元 (`'username'@'%'`)。
- 執行 Aurora Serverless v1、以 Aurora PostgreSQL 為基礎的資料庫叢集有下列限制：
  - 不支援 Aurora PostgreSQL 查詢計劃管理 (`apg_plan_management` 擴充功能)。
  - 不支援 Amazon RDS PostgreSQL 和 Aurora PostgreSQL 中提供的邏輯複寫功能。
  - 不支援輸出通訊，例如由 Amazon RDS for PostgreSQL 延伸所啟用的通訊。例如，您無法使用 `postgres_fdw/dblink` 擴充功能存取外部資料。如需有關 RDS PostgreSQL 延伸的詳細資訊，請參閱 RDS 使用者指南中 Amazon RDS 的 PostgreSQL。

- 目前，不建議使用某些 SQL 查詢和命令。這些包括工作階段層級的建議鎖定、暫時關係、非同步通知 (LISTEN)，以及具有保留 (DECLARE *name* ... CURSOR WITH HOLD FOR *query*) 的游標。此外，NOTIFY 命令會防止擴展，因此不建議使用。

  如需詳細資訊，請參閱Aurora Serverless v1 的自動調整規模。

- 您無法為 Aurora Serverless v1 資料庫叢集設定偏好的自動備份時段。
- 您可以設定 Aurora Serverless v1 資料庫叢集的維護時段。如需詳細資訊，請參閱調整偏好的資料庫叢集維護時段。

Aurora Serverless v1 的組態需求

建立 Aurora Serverless v1 資料庫叢集時，請注意下列需求：

- 為每個資料庫引擎使用這些特定的連接埠號碼：
  - Aurora MySQL：3306
  - Aurora PostgreSQL：5432
- 在以 Amazon VPC 服務為基礎的 Virtual Private Cloud (VPC) 中建立 Aurora Serverless v1 資料庫叢集。在 VPC 中建立 Aurora Serverless v1 資料庫叢集時，您會取用指派給 VPC 的五十 (50) 個界面和 Gateway Load Balancer 端點中的兩 (2) 個。這些端點是自動為您建立的。若要提高配額，您可以聯絡 支援。如需詳細資訊，請參閱 Amazon VPC 配額。
- 您無法為 Aurora Serverless v1 資料庫叢集提供公有 IP 地址。您只能從 VPC 內存取 Aurora Serverless v1 資料庫叢集。
- 在不同的可用區域中，為您用於 Aurora Serverless v1 資料庫叢集的資料庫子網路群組建立子網路。換句話說，您不能在同一個可用區域中有一個以上的子網路。
- 不會將對 Aurora Serverless v1 資料庫叢集使用的子網路群組所做的變更套用到叢集。
- 您可以從 Aurora Serverless v1 存取 資料庫叢集 AWS Lambda。若要這麼做，您必須將 Lambda 函數設定為在與 Aurora Serverless v1 資料庫叢集相同的 VPC 中執行。如需使用 的詳細資訊 AWS Lambda，請參閱《 AWS Lambda 開發人員指南》中的設定 Lambda 函數以存取 Amazon VPC 中的資源。

搭配 Aurora Serverless v1 使用 TLS/SSL

根據預設，Aurora Serverless v1 會使用 Transport Layer Security/Secure Sockets Layer (TLS/SSL) 通訊協定，來加密用戶端與 Aurora Serverless v1 資料庫叢集之間的通訊。它支援 TLS/SSL 1.0、1.1 和 1.2 版。您不需要將 Aurora Serverless v1 資料庫叢集設定為使用 TLS/SSL。

然而，具有下列限制：

- 目前中國（北京）Aurora Serverless v1 不支援資料庫叢集的 TLS/SSL AWS 區域。
- 當您為以 Aurora MySQL 為基礎的 Aurora Serverless v1 資料庫叢集建立資料庫使用者時，請勿將 `REQUIRE` 子句用於 SSL 許可。這樣做會防止使用者連線到 Aurora 資料庫執行個體。
- 對於 MySQL 用戶端和 PostgreSQL 用戶端公用程式，在用戶端和 Aurora Serverless v1 之間使用 TLS/SSL 時，您可能在其他環境中使用的工作階段變數會沒有作用。
- 對於 MySQL 用戶端，當使用 TLS/SSL `VERIFY_IDENTITY` 模式進行連線時，您目前需要使用與 MySQL 8.0 相容的 `mysql` 命令。如需詳細資訊，請參閱連線至執行 MySQL 資料庫引擎的資料庫執行個體 。

視您用來連線到 Aurora Serverless v1 資料庫叢集的用戶端而定，您可能不需要指定 TLS/SSL，即可取得加密連線。例如，若要使用 PostgreSQL 用戶端來連線到執行 Aurora PostgreSQL 相容版本的 Aurora Serverless v1 資料庫叢集，請按照平常的方式進行連線。

```
psql -h endpoint -U user
```

輸入密碼後，PostgreSQL 用戶端會顯示連線詳細資訊，包括 TLS/SSL 版本和密碼。

```
psql (12.5 (Ubuntu 12.5-0ubuntu0.20.04.1), server 10.12)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
 compression: off)
Type "help" for help.
```

> **Important**
>
> 除非用戶端應用程式停用 SSL/TLS，否則預設為 Aurora Serverless v1 使用 Transport Layer Security/Secure Sockets Layer (TLS/SSL) 通訊協定加密連線。TLS/SSL 連線終止於路由器機群。路由器機群與 Aurora Serverless v1 資料庫叢集之間的通訊會在服務的內部網路界限內進行。

您可以檢查用戶端連線的狀態，以檢查與 Aurora Serverless v1 的連線是否為 TLS/SSL 加密。PostgreSQL `pg_stat_ssl` 和 `pg_stat_activity` 資料表及其 `ssl_is_used` 函數不會顯示用戶端應用程式與 Aurora Serverless v1 之間通訊的 TLS/SSL 狀態。同樣地，TLS/SSL 狀態無法衍生自 MySQL `status` 陳述式。
Aurora Serverless v1 之前不支援叢集參數 `force_ssl` for PostgreSQL 和 `require_secure_transport` for MySQL。這些參數現在可用於 Aurora Serverless v1。如需 Aurora 無伺服器 v1 支援的完整參數清單，請呼叫 DescribeEngineDefaultClusterParameters API 作業。如需參數群組和 Aurora 無伺服器 v1 的詳細資訊，請參閱Aurora Serverless v1 的參數群組。

若要使用 MySQL 用戶端來連線到執行 Aurora MySQL 相容版本的 Aurora Serverless v1 資料庫叢集，請在請求中指定 TLS/SSL。下列範例包含從 Amazon 信任服務 (此服務為此連線能成功的必要項目) 下載的 Amazon 根 CA 1 信任存放區。

```
mysql -h endpoint -P 3306 -u user -p --ssl-ca=amazon-root-CA-1.pem --ssl-mode=REQUIRED
```

出現提示時，輸入您的密碼。MySQL 監控器便會開啟。您可以使用 `status` 命令來確認工作階段已加密。

```
mysql> status
--------------
mysql  Ver 14.14 Distrib 5.5.62, for Linux (x86_64) using readline 5.1
Connection id:          19
Current database:
Current user:           ***@*******
SSL:                    Cipher in use is ECDHE-RSA-AES256-SHA
...
```

若要進一步了解如何使用 MySQL 用戶端連線至 Aurora MySQL 資料庫，請參閱連線至執行 MySQL 資料庫引擎的資料庫執行個體 。

Aurora Serverless v1 支援 MySQL 用戶端 (`mysql`) 和 PostgreSQL 用戶端 (`psql`) 可用的所有 TLS/SSL 模式，包括下表所列的模式。

TLS/SSL 模式的說明	mysql	psql
不使用 TLS/SSL 連線。	DISABLED	停用

TLS/SSL 模式的說明	mysql	psql
請先嘗試先使用 TLS/SSL 連線，但必要時會回復為 SSL 以外的加密技術。	PREFERRED	偏好 (預設)
強制使用 TLS/SSL。	REQUIRED	require
強制執行 TLS/SSL 並驗證 CA。	VERIFY_CA	verify-ca
強制執行 TLS/SSL、驗證 CA，並驗證 CA 主機名稱。	VERIFY_IDENTITY	verify-full

Aurora Serverless v1 使用萬用字元憑證。如果您在使用 TLS/SSL 時指定「驗證 CA」或「驗證 CA 和 CA 主機名稱」選項，請先從 Amazon 信任服務下載 Amazon 根 CA 1 信任存放區。這樣做之後，您就可以在用戶端命令中識別這個 PEM 格式的檔案。若要使用 PostgreSQL 用戶端執行這項操作：

對於 Linux、macOS 或 Unix：

```
psql 'host=endpoint user=user sslmode=require sslrootcert=amazon-root-CA-1.pem
 dbname=db-name'
```

若要進一步了解如何將 Postgres 用戶端與 Aurora PostgreSQL 資料庫搭配使用，請參閱連線至執行 PostgreSQL 資料庫引擎的資料庫執行個體。

如需連線至 Aurora 資料庫叢集的一般資訊，請參閱 連接至 Amazon Aurora 資料庫叢集。

適用於 Aurora Serverless v1 資料庫叢集連線的受支援密碼套件

透過使用可設定的密碼套件，您可以更進一步控制資料庫連線的安全性。您可以指定要允許的密碼套件清單，以保護您資料庫的用戶端 SSL/TLS 連線安全。您可以使用可設定的密碼套件來控制資料庫伺服器接受的連線加密。這麼做可以防止使用不安全或不再使用的密碼。

基於 Aurora MySQL 的 Aurora Serverless v1 資料庫叢集支援與 Aurora MySQL 佈建之資料庫叢集相同的密碼套件。如需這些密碼套件的相關資訊，請參閱 為 Aurora MySQL 資料庫叢集的連線設定密碼套件。

基於 Aurora PostgreSQL 的 Aurora Serverless v1 資料庫叢集不支援密碼套件。

Aurora Serverless v1 的運作方式

> **Important**
>
> AWS 已宣布 的end-of-lifeAurora Serverless v1：2025 年 3 月 31 日。所有未在 2025 年 3 月 31 日之前遷移的 Aurora Serverless v1 叢集都會在維護時段期間遷移至 Aurora Serverless v2。如果升級失敗，Amazon Aurora 會在維護時段將 Serverless v1 叢集轉換為具有同等引擎版本的佈建叢集。如果適用，Amazon Aurora 將在 Amazon RDS 延長支援中註冊轉換後的已佈建叢集。如需詳細資訊，請參閱RDS 延長支援。

接下來，您可了解 Aurora Serverless v1 的運作方式。

主題

- Aurora Serverless v1 架構
- Aurora Serverless v1 的自動調整規模
- 容量變更時的逾時動作
- Aurora Serverless v1 的暫停和繼續
- 決定 Aurora Serverless v1 的資料庫連線數量上限
- Aurora Serverless v1 的參數群組
- 為 Aurora Serverless v1 記錄日誌。
- Aurora Serverless v1 和維護
- Aurora Serverless v1 和容錯移轉
- Aurora Serverless v1 和快照

Aurora Serverless v1 架構

下圖顯示 Aurora Serverless v1 架構的概觀。



您將指定 Aurora Capacity Units (ACU)，而不用佈建和管理資料庫伺服器。每個 ACU 是大約 2 GB 的記憶體、對應的 CPU 和聯網的組合。資料庫儲存會以 10 GiB 至 128 tebibytes (TiB) 的範圍自動擴展，與標準的 Aurora 資料庫叢集儲存相同。

您可指定 ACU 的最小值與最大值。最小 Aurora 容量單位是資料庫叢集可縮減到的最低 ACU。最大 Aurora 容量單位是資料庫叢集可擴展到的最高 ACU。Aurora Serverless v1 會根據您的設定自動建立 CPU 使用率、連線數及可用記憶體閾值的規模調整規則。

Aurora Serverless v1 在 中管理資源的暖集區 AWS 區域 ，以將擴展時間降至最低。當 Aurora Serverless v1 將資源新增至 Aurora 資料庫叢集時，它會使用路由器機群，將作用中用戶端連線切換至新資源。在任何特定時間點，您只需為您的 Aurora 資料庫叢集中正在使用的 ACU 付費。

Aurora Serverless v1 的自動調整規模

分配給 Aurora Serverless v1 資料庫叢集的容量，將根據用戶端應用程式產生的負載無縫擴展或縮減。在這裡，負載是 CPU 使用率和連線的數量。當容量受到上述任何一種限制時，Aurora Serverless v1 擴充。Aurora Serverless v1 也會在偵測到這樣做可以得到解決的效能問題時進行擴充。

您可以在 中檢視Aurora Serverless v1叢集的擴展事件 AWS 管理主控台。在自動調整規模期間，Aurora Serverless v1 會重設 `EngineUptime` 指標。重設指標值的值並不意味著無縫規模調整有問題，也不意味著 Aurora Serverless v1 中斷了連線。這只是新容量的正常運作時間的起點。進一步了解指標，請參閱在 Amazon Aurora 叢集中監控指標。

當您的 Aurora Serverless v1 資料庫叢集沒有作用中的連線時，它可以縮小至零容量 (0 ACU)。如需進一步了解，請參閱Aurora Serverless v1 的暫停和繼續。

當它需要執行規模調整操作時，Aurora Serverless v1 首先嘗試識別規模調整點，一個沒有查詢正在處理的時刻。Aurora Serverless v1 可能無法找到規模調整點，原因如下：

- 長時間執行的查詢
- 進行中的交易
- 暫存資料表或資料表鎖定

若要增加 Aurora Serverless v1 資料庫叢集尋找規模調整點時的成功率，建議您避免長時間執行的查詢和長時間執行的交易。如要深入了解封鎖規模調整操作及如何避免這些操作，請參閱使用 Aurora Serverless v1 的最佳實務。

Aurora Serverless v1 預設會嘗試尋找擴展點 5 分鐘 (300 秒)。您可以在建立或修改叢集時，指定其他逾時期間。逾時期間可介於 60 秒到 10 分鐘 (600 秒) 之間。如果 Aurora Serverless v1 在指定的期間內找不到擴展點，則自動擴展逾時。

預設情況下，如果自動調整規模在超時之前找不到規模調整點，Aurora Serverless v1 會將叢集保持在目前的容量。您可以在透過選取 Force the capacity change (強制變更容量) 選項建立或修改 Aurora Serverless v1 資料庫叢集時，變更此預設行為。如需詳細資訊，請參閱 容量變更時的逾時動作。

容量變更時的逾時動作

如果自動擴展在找不到擴展點的情況下逾時，預設情況下 Aurora 會保留目前的容量。您可選取 Force the capacity change (強制變更容量) 選項，選擇讓 Aurora 強制變更。當您建立叢集時，此選項在 Create database (建立資料庫) 頁面的 Autoscaling timeout and action (自動擴展逾時與動作) 區段可供使用。

依預設，不選取 Force the capacity change (強制變更容量) 選項。不勾選此選項，若擴展操作逾時而沒有找到擴展點，可以讓 Aurora Serverless v1 資料庫叢集的容量保持不變。

選取此選項會導致 Aurora Serverless v1 資料庫叢集強制執行容量變更，即使沒有規模調整點也一樣。選取此選項之前，請注意此選項的結果：

- 任何處理中的交易都會中斷，並會出現下列錯誤訊息。

  Aurora MySQL 版本 2 — ERROR 1105 (HY000)：由於無縫擴展，最後一筆交易已終止。請再試一次。

  一旦您的 Aurora Serverless v1 資料庫叢集可供使用，您就可以重新提交交易。

- 暫存資料表和鎖定的連線會被中斷。

  我們建議僅於應用程式可從中斷的連線或未完成的交易復原時，才選取 Force the capacity change (強制變更容量) 選項。

您在建立資料庫叢集 AWS 管理主控台 時在 Aurora Serverless v1 中所做的選擇，會存放在 `ScalingConfigurationInfo` 物件的 `SecondsBeforeTimeout`和 `TimeoutAction` 屬性中。`TimeoutAction` 屬性的值設定為建立叢集時的下列值之一：

- `RollbackCapacityChange` – 當您選取 Roll back the capacity change (回復容量變更) 選項時，會設定此值。這是預設行為。
- `ForceApplyCapacityChange` – 當您選取 Force the capacity change (強制變更容量) 選項時，會設定此值。

您可以使用 describe-db-clusters AWS CLI Aurora Serverless v1 命令，在現有資料庫叢集上取得此屬性的值，如下所示。

針對 Linux、macOS 或 Unix：

```
aws rds describe-db-clusters --region region \
  --db-cluster-identifier your-cluster-name \
  --query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'
```

在 Windows 中：

```
aws rds describe-db-clusters --region region ^
  --db-cluster-identifier your-cluster-name ^
  --query "*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}"
```

例如，下面顯示了美國西部 (加利佛尼亞北部) 區域中名為 `west-coast-sles` 的 Aurora Serverless v1 資料庫叢集的查詢和回應。

```
$ aws rds describe-db-clusters --region us-west-1 --db-cluster-identifier west-coast-
sles
--query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'

[
    {
        "ScalingConfigurationInfo": {
            "MinCapacity": 1,
            "MaxCapacity": 64,
            "AutoPause": false,
            "SecondsBeforeTimeout": 300,
            "SecondsUntilAutoPause": 300,
            "TimeoutAction": "RollbackCapacityChange"
        }
    }
]
```

正如回應所示，此 Aurora Serverless v1 資料庫叢集會使用預設設定。

如需更多詳細資訊，請參閱 建立 Aurora Serverless v1 資料庫叢集。建立 Aurora Serverless v1 後，您可以隨時修改逾時動作和其他容量設定。若要瞭解如何操作，請參閱修改 Aurora Serverless v1 資料庫叢集。

Aurora Serverless v1 的暫停和繼續

在經過一段時間沒有活動後，您可以選擇暫停您的 Aurora Serverless v1 資料庫叢集。您可指定資料庫叢集暫停之前的無活動時間。當您選取此選項時，預設的無活動時間為 5 分鐘，但您可以變更此值。這是選用的設定。

資料庫叢集暫停時不會產生任何運算或記憶體活動，您僅會因為使用儲存空間而付費。如果在 Aurora Serverless v1 資料庫叢集暫停時請求資料庫連線，資料庫叢集會自動繼續並處理連線請求。

當資料庫叢集恢復活動時，它具有與 Aurora 暫停叢集時相同的容量。ACU 的數目取決於在暫停叢集之前 Aurora 擴增或縮減叢集的程度。

> **Note**
>
> 若資料庫叢集已暫停超過七日，該資料庫叢集可能會以快照的方式進行備份。在此情況下，當有連線請求時，Aurora 即會從快照還原資料庫叢集。

決定 Aurora Serverless v1 的資料庫連線數量上限

以下範例適用於與 MySQL 5.7 相容的 Aurora Serverless v1 資料庫叢集。您可以使用 MySQL 用戶端或查詢編輯器 (如果您已設定其存取權)。如需詳細資訊，請參閱在查詢編輯器中執行查詢。

找出資料庫連線數上限

1. 使用 AWS CLI尋找 Aurora Serverless v1 資料庫叢集的容量範圍。

```
aws rds describe-db-clusters \
    --db-cluster-identifier my-serverless-57-cluster \
    --query 'DBClusters[*].ScalingConfigurationInfo|[0]'
```

結果顯示其容量範圍為 1–4 個 ACU。

```
{
    "MinCapacity": 1,
    "AutoPause": true,
    "MaxCapacity": 4,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

2. 執行以下 SQL 查詢以尋找連線數量上限。

```
select @@max_connections;
```

顯示的結果是叢集的最小容量，即 1 個 ACU。

```
@@max_connections
90
```

3. 將叢集擴展到 8–32 個 ACU。

如需擴展的詳細資訊，請參閱修改 Aurora Serverless v1 資料庫叢集。

4. 確認容量範圍。

```
{
    "MinCapacity": 8,
    "AutoPause": true,
    "MaxCapacity": 32,
```

```
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

5. 尋找連線數目上限。

```
select @@max_connections;
```

顯示的結果是叢集的最小容量，即 8 個 ACU。

```
@@max_connections
1000
```

6. 將叢集擴展至可能的上限，256–256 個 ACU。
7. 確認容量範圍。

```
{
    "MinCapacity": 256,
    "AutoPause": true,
    "MaxCapacity": 256,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

8. 尋找連線數目上限。

```
select @@max_connections;
```

顯示的結果是 256 個 ACU。

```
@@max_connections
6000
```

> **Note**
>
> max_connections 值不會隨著 ACU 的數量線性擴縮。

9. 將叢集縮小回 1–4 個 ACU。

```
{
```

```
    "MinCapacity": 1,
    "AutoPause": true,
    "MaxCapacity": 4,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

這一次，max_connections 值為 4 個 ACU。

```
@@max_connections
270
```

10. 將叢集縮小為 2 個 ACU。

```
@@max_connections
180
```

如果您已將叢集設定為在閒置一定的時間後暫停，則叢集會縮小到 0 個 ACU。但是,max_connections 不會降至低於 1 個 ACU 的值。

```
@@max_connections
90
```

Aurora Serverless v1 的參數群組

建立 Aurora Serverless v1 資料庫叢集時，您可以選擇特定的 Aurora 資料庫引擎和相關的資料庫叢集參數群組。不同於已佈建 Aurora 資料庫叢集，Aurora Serverless v1 資料庫叢集具有單一讀取/寫入資料庫執行個體，該執行個體僅設定為資料庫叢集參數群組 — 它沒有單獨的資料庫參數群組。在自動調整規模期間，Aurora Serverless v1 必須能夠變更叢集的參數，才能最適合增加或減少的容量。因此，使用 Aurora Serverless v1 資料庫叢集，您會對特定資料庫引擎類型的參數所做的一些變更可能不適用。

例如，Aurora PostgreSQL–型 Aurora Serverless v1 資料庫叢集無法使用 apg_plan_mgmt.capture_plan_baselines 和可能在已佈建 Aurora PostgreSQL 資料庫叢集上使用以進行查詢計劃管理的其他參數。

藉由使用 describe-engine-default-cluster-parameters CLI 命令並查詢 AWS 區域，您可以取得各種 Aurora 資料庫引擎預設參數群組的預設值清單。下列為您可用於 --db-parameter-group-family 選項的值。

Aurora MySQL 第 2 版	`aurora-mysql5.7`
Aurora PostgreSQL 11 版	`aurora-postgresql11`
Aurora PostgreSQL 第 13 版	`aurora-postgresql13`

建議您 AWS CLI 使用 AWS 存取金鑰 ID 和 AWS 私密存取金鑰來設定 ，並在使用 AWS CLI 命令 AWS 區域 之前設定您的 。將區域提供給 CLI 組態可讓您在執行命令時無法輸入 `--region` 參數。若要進一步了解設定 AWS CLI，請參閱AWS Command Line Interface 《 使用者指南》中的組態基本概念。

下列範例會從 Aurora MySQL 2 的預設資料庫叢集群組取得參數清單。

對於 Linux、macOS 或 Unix：

```
aws rds describe-engine-default-cluster-parameters \
  --db-parameter-group-family aurora-mysql5.7 --query \
  'EngineDefaults.Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, `serverless`) == `true`] | [*].{param:ParameterName}' \
  --output text
```

在 Windows 中：

```
aws rds describe-engine-default-cluster-parameters ^
   --db-parameter-group-family aurora-mysql5.7 --query ^
   "EngineDefaults.Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, 'serverless') == `true`] | [*].{param:ParameterName}" ^
   --output text
```

為 Aurora Serverless v1 修改參數值

正如 Amazon Aurora 的參數群組 所述，無論其類型為何 (DB 叢集參數群組、DB 參數群組)，都無法直接變更預設參數群組中的值。相反地，您可以根據預設的資料庫叢集參數群組為 Aurora 資料庫引擎建立自訂參數群組，並依該參數群組所需變更設定。例如，您可能想要將 Aurora Serverless v1 資料庫叢集的一些設定變更為日誌查詢或上傳資料庫引擎特定日誌到 Amazon CloudWatch。

建立自訂資料庫叢集參數群組

1. 登入 ，AWS 管理主控台 然後開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 選擇 Parameter groups (參數群組)。
3. 選擇 Create parameter group (建立參數群組) 以開啟「參數群組詳細資訊」窗格。
4. 為您要用於 Aurora Serverless v1 資料庫叢集的資料庫引擎，選擇適當的預設資料庫叢集群組。請務必選擇下列選項：

   a. 對於 Parameter group family (參數群組系列)，請為選擇的資料庫引擎選擇適當的系列。請確保您選擇名稱中的字首為 `aurora-`。

   b. 在 Type (類型) 中，選擇 DB Cluster Parameter Group (資料庫叢集參數群組)。

   c. 在 Group name (群組名稱) 和 Description (描述) 中，為您或可能需要使用 Aurora Serverless v1 資料庫叢集及其參數的其他人輸入有意義的名稱。

   d. 選擇建立。

您的自訂資料庫叢集參數群組會新增至您的 AWS 區域中可用的參數群組清單。當您建立新的 Aurora Serverless v1 資料庫叢集時，您可使用自訂的資料庫叢集參數群組。您也可以修改現有的 Aurora Serverless v1 資料庫叢集，以使用自訂的資料庫叢集參數群組。Aurora Serverless v1 資料庫叢集開始使用自訂資料庫叢集參數群組後，您可以使用 AWS 管理主控台 或 變更動態參數的值 AWS CLI。

您也可以使用 Console (主控台) 來檢視與預設資料庫叢集參數群組相比的自訂資料庫叢集參數群組中的值並排比較，如下列螢幕擷取畫面所示。



當您變更作用中資料庫叢集上的參數值時，Aurora Serverless v1 會啟動無縫規模調整，以套用參數變更。若您的 Aurora Serverless v1 資料庫叢集處於暫停狀態，它會恢復並開始規模調整，以便可以進行變更。參數群組的規模調整操作變更會永遠強制變更容量，因此請注意，如果在規模調整期間找不到規模調整點，修改參數可能會導致連線中斷。

為 Aurora Serverless v1 記錄日誌。

預設情況下，Aurora Serverless v1 錯誤日誌已啟用並自動上傳至 Amazon CloudWatch。您還可讓您的 Aurora Serverless v1 資料庫叢集將 Aurora 資料庫引擎特定日誌上傳至 CloudWatch。為此，請在您的自訂資料庫叢集參數群組中啟用組態參數。您的 Aurora Serverless v1 資料庫叢集，然後將所有可用日誌上傳至 Amazon CloudWatch。您可於此時使用 Amazon CloudWatch 來分析日誌資料、建立警示和檢視指標。

對於 Aurora MySQL，下表顯示您可以啟用的日誌。啟用時，它們會自動從您的 Aurora Serverless v1 資料庫叢集上傳到 Amazon CloudWatch。

Aurora MySQL 日誌	Description
general_log	建立一般日誌。設定為 1 可開啟。預設為關閉 (0)。
log_queries_not_using_indexes	將任何查詢記錄到不使用索引的慢查詢日誌。預設為關閉 (0)。設定為 1 可開啟此日誌。
long_query_time	防止快速執行的查詢在慢查詢日誌中記錄。可以設定為介於 0 到 3,1536,000 之間的浮點數。預設值為 0 (非作用中)。
server_audit_events	日誌中要擷取的事件清單。支援的值為 CONNECT、QUERY、QUERY_DCL 、QUERY_DDL 、QUERY_DML 和 TABLE。
server_audit_logging	設定為 1 以開啟伺服器稽核日誌記錄。如果開啟此選項，您可以在 server_audit_events 參數中列出稽核事件，以指定要傳送至 CloudWatch 的稽核事件。
slow_query_log	建立慢查詢日誌。設定為 1 以開啟慢查詢日誌。預設為關閉 (0)。

如需詳細資訊，請參閱使用進階稽核與 Amazon Aurora MySQL 資料庫叢集搭配。

對於 Aurora PostgreSQL，下表顯示您可以啟用的日誌。啟用時，它們會自動從您的 Aurora Serverless v1 資料庫叢集上傳到 Amazon CloudWatch，連同一般錯誤日誌。

Aurora PostgreSQL 日誌	Description
log_connections	依預設為啟用，且無法變更。它會記錄所有新用戶端連線的詳細資訊。
log_disconnections	依預設為啟用，且無法變更。記錄所有用戶端的連線中斷。

Aurora PostgreSQL 日誌	Description
log_hostname	依預設為關閉，且無法變更。主機名稱不會記錄。
log_lock_waits	預設值為 0 (關閉)。設定為 1 可等待日誌鎖定。
log_min_duration_statement	記錄陳述式之前執行的最短持續時間 (以毫秒為單位)。
log_min_messages	設定所記錄的訊息層級。支援的值為 debug5、debug4、debug3、debug2、debug1、inf 若要將效能資料記錄到 postgres 日誌中，則將值設定為 debug1。
log_temp_files	記錄超過指定 KB 的暫存檔案的使用。
log_statement	控制已記錄的特定 SQL 陳述式。支援的值為 none、ddl、mod 和 all。預設值為 none。

為您的 Aurora Serverless v1 資料庫叢集開啟 Aurora MySQL 或 Aurora PostgreSQL 日誌後，您可在 CloudWatch 中檢視日誌。

使用 Amazon CloudWatch 檢視 Aurora Serverless v1 日誌

Aurora Serverless v1 會自動上傳 (「發佈」) 到自訂資料庫叢集參數群組中啟用所有日誌的 Amazon CloudWatch。您不需要選擇或指定日誌類型。只要您啟用日誌組態參數，就會立即開始上傳日誌。如果您稍後停用日誌參數，則會停止後續上傳。不過，所有已發佈至 CloudWatch 的所有日誌會保留直到您刪除為止。

如需搭配 Aurora MySQL 日誌使用 CloudWatch 的詳細資訊，請參閱在 Amazon CloudWatch 中監控日誌事件。

如需 CloudWatch 和 Aurora PostgreSQL 的詳細資訊，請參閱「將 Aurora PostgreSQL 日誌發佈至 Amazon CloudWatch Logs」。

檢視 Aurora Serverless v1 資料庫叢集的日誌

1. 透過 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。

2. 選擇您的 AWS 區域。
3. 選擇 Log groups (日誌群組)。
4. 從清單中選擇 Aurora Serverless v1 資料庫叢集日誌。若是錯誤日誌，命名模式如下。

```
/aws/rds/cluster/cluster-name/error
```

例如，於下列的螢幕擷取畫面中，您可以找到為名為 `western-sles` 的 Aurora PostgreSQL Aurora Serverless v1 資料庫叢集發佈的日誌清單。您還可以找到幾個 Aurora MySQL Aurora Serverless v1 資料庫叢集 `west-coast-sles` 的清單。選擇感興趣的日誌，開始探索其內容。



Aurora Serverless v1 和維護

Aurora Serverless v1 資料庫叢集的維護，例如套用最新功能、修正和安全性更新，都會自動為您執行。 Aurora Serverless v1 有一個維護時段，您可以在資料庫叢集的維護與備份 AWS 管理主控台 中的 Aurora Serverless v1 中檢視。您可以如下圖所示，找到可能執行維護的日期和時間，以及是否有任何正待為 Aurora Serverless v1 資料庫叢集進行的維護。



您可以在建立 Aurora Serverless v1 資料庫叢集時設定維護時段，也可以稍後再修改維護時段。如需詳細資訊，請參閱調整偏好的資料庫叢集維護時段。

維護時段是用於排程的主要版本升級。次要版本升級和修補程式會在擴展期間立即套用。擴展會根據您的 `TimeoutAction` 設定進行：

- `ForceApplyCapacityChange` – 會立即套用變更。
- `RollbackCapacityChange` – Aurora 會在第一次嘗試修補程式的 3 天後強制更新叢集。

就像在沒有適當擴展點的情況下強制執行的任何變更一樣，這可能會中斷您的工作負載。

Aurora Serverless v1 會盡可能地以非中斷式的方式執行維護。需要維護時，Aurora Serverless v1 資料庫叢集會規模調整其容量以處理必要的操作。在縮放之前，Aurora Serverless v1 會尋找縮放點。如有必要，它會長達 3 天。

在找不到規模調整點的每一天結束時，Aurora Serverless v1 就會建立叢集事件。此事件會通知您待處理的維護，以及需要擴展以執行維護的需求。該通知包括 Aurora Serverless v1 可強制資料庫叢集規模調整的日期。

如需詳細資訊，請參閱容量變更時的逾時動作。

Aurora Serverless v1 和容錯移轉

若 Aurora Serverless v1 資料庫叢集的資料庫執行個體變得無法使用或可用區域 (AZ) 失敗，Aurora 會在不同的 AZ 中重新建立資料庫執行個體。但是，Aurora Serverless v1 叢集不是多可用區域叢集。這是因為它是由單一可用區域中的單一資料庫執行個體所組成。因此，這項容錯移轉機制需要的時間比佈建或 Aurora Serverless v2 執行個體的 Aurora 叢集更長。Aurora Serverless v1 容錯移轉的時間未定，因為這取決於特定 AWS 區域內其他可用區域的需求和容量可用性。

因為 Aurora 會區分運算容量和儲存體，叢集的儲存體磁碟區會分布在多個可用區域內。即使中斷影響資料庫執行個體或相關聯的可用區域，您的資料也能維持在可用狀態。

Aurora Serverless v1 和快照

Aurora Serverless v1 叢集的叢集磁碟區一律會加密。您可以選擇加密金鑰，但無法停用加密。如要複製或共用 Aurora Serverless v1 叢集的快照，您可以使用自己的 AWS KMS key加密該快照。如需詳細資訊，請參閱資料庫叢集快照複製。若要進一步了解加密和 Amazon Aurora，請參閱 加密 Amazon Aurora 資料庫叢集

建立 Aurora Serverless v1 資料庫叢集

> **Important**
>
> AWS 已宣布 Aurora Serverless v1 的終止服務日期：2025 年 3 月 31 日。所有未在 2025 年 3 月 31 日之前遷移的 Aurora Serverless v1 叢集都會在維護時段期間遷移至 Aurora Serverless v2。如果升級失敗，Amazon Aurora 會在維護時段將 Serverless v1 叢集轉換為具有同等引擎版本的佈建叢集。如果適用，Amazon Aurora 將在 Amazon RDS 延長支援中註冊轉換後的已佈建叢集。如需更多詳細資訊，請參閱 RDS 延長支援。

下列程序建立一個 Aurora Serverless v1 叢集，不包含任何結構描述物件或資料。若您要建立 Aurora Serverless v1 叢集，該叢集為現有佈建或 Aurora Serverless v1 叢集，您可改為執行快照還原或複製作業。如需這些詳細資訊，請參閱 從資料庫叢集快照還原 和 複製 Amazon Aurora 資料庫叢集的一個磁碟區。您無法將現有的佈建叢集轉換為 Aurora Serverless v1。您也無法將現有的 Aurora Serverless v1 叢集轉換回佈建叢集。

當您建立 Aurora Serverless v1 資料庫叢集時，您可設定叢集的最小與最大容量。每個容量單位相等於一個特定的運算和記憶體組態。Aurora Serverless v1 會建立 CPU 使用率、連線和可用記憶體閾值的擴展規則，並根據您應用程式的需求，順暢地擴展到某個容量單位範圍。如需詳細資訊，請參閱 Aurora Serverless v1 架構。

您可以為您的 Aurora Serverless v1 資料庫叢集設定以下特定值：

- Minimum Aurora capacity unit (最小 Aurora 容量單位) – Aurora Serverless v1 可將容量降低至此容量單位下限。
- Maximum Aurora capacity unit (最大 Aurora 容量單位) – Aurora Serverless v1 可將容量提升至此容量單位上限。

您也可以選擇下列選用的擴展組態選項：

- 達到逾時時，強制將容量調擴展為指定的值 – 如果您希望 Aurora Serverless v1 強制 Aurora Serverless v1 擴展規模 (即使在逾時之前找不到擴展點)，則可選擇此設定。如果您希望 Aurora Serverless v1 在找不到擴展點時取消容量變更，請勿選擇此設定。如需更多詳細資訊，請參閱 容量變更時的逾時動作。
- 在非使用狀態下連續數分鐘後暫停運算容量 – 如果您希望在指定的時間內資料庫叢集沒有任何活動時，Aurora Serverless v1 縮減為零，可以選擇此設定。啟用此設定後，Aurora Serverless v1 資料

庫叢集會自動繼續處理，並在資料庫流量恢復時擴展至必要的容量以處理工作負載。如需進一步了解，請參閱Aurora Serverless v1 的暫停和繼續。

在建立 Aurora Serverless v1 資料庫叢集之前，您需要一個 AWS 帳戶。您也必須完成設定工作，才能使用 Amazon Aurora。如需更多詳細資訊，請參閱 設定您的 Amazon Aurora 環境。您還需要完成其他初步步驟以建立任何 Aurora 資料庫叢集。如需進一步了解，請參閱建立 Amazon Aurora 資料庫叢集。

Aurora Serverless v1 僅適用於特定 AWS 區域，且僅適用於特定 Aurora MySQL 和 Aurora PostgreSQL 版本。如需更多詳細資訊，請參閱 Aurora Serverless v1。

> **Note**
>
> Aurora Serverless v1 叢集的叢集磁碟區一律會加密。建立 Aurora Serverless v1 資料庫叢集時，您就無法關閉加密，但您可以選擇使用自己的加密金鑰。利用 Aurora Serverless v2，您可選擇是否加密叢集磁碟區。

您可以使用 AWS CLI 或 RDS API 建立 Aurora Serverless v1 資料庫叢集。

> **Note**
>
> 如果您在嘗試建立叢集時收到下列錯誤訊息，表示您的帳戶需要額外的權限。
> `Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.`
> 如需詳細資訊，請參閱「使用 Amazon Aurora 的服務連結角色」。

您無法直接連線到 Aurora Serverless v1 資料庫叢集上的資料庫執行個體。若要連線到 Aurora Serverless v1 資料庫叢集，您可以使用資料庫端點。您可以在 Aurora Serverless v1 中叢集的 Connectivity & security (連線與安全性) 索引標籤上找到 AWS 管理主控台 資料庫叢集的端點。如需更多詳細資訊，請參閱 連接至 Amazon Aurora 資料庫叢集。

AWS CLI

若要使用 Aurora Serverless v1 建立新的 AWS CLI 資料庫叢集，請執行 create-db-cluster 命令，並在 `serverless` 選項指定 `--engine-mode`。

您可以選擇指定 `--scaling-configuration` 選項以設定最小容量、最大容量，並在沒有連線時自動暫停。

以下命令範例會透過將 `--engine-mode` 選項設為 `serverless` 來建立新的 Serverless 資料庫叢集。範例也會指定 `--scaling-configuration` 選項的值。

Aurora MySQL 的範例

以下命令會建立新的 Aurora MySQL 相容 Serverless 資料庫叢集。Aurora MySQL 的有效容量值為 1、2、4、8、16、32、64、128 和 256。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
    --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.4 \
    --engine-mode serverless \
    --scaling-configuration
 MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true \
    --master-username username --master-user-password password
```

在 Windows 中：

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
    --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.4 ^
    --engine-mode serverless ^
    --scaling-configuration
 MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true ^
    --master-username username --master-user-password password
```

Aurora PostgreSQL 的範例

以下命令會建立新的 PostgreSQL 13.9 相容 Serverless 資料庫叢集。Aurora PostgreSQL 的有效容量值為 2、4、8、16、32、64、192 和 384。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
    --engine aurora-postgresql --engine-version 13.9 \
    --engine-mode serverless \
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true \
    --master-username username --master-user-password password
```

在 Windows 中：

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
    --engine aurora-postgresql --engine-version 13.9 ^
    --engine-mode serverless ^
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true ^
    --master-username username --master-user-password password
```

RDS API

若要使用 RDS API 建立新的 Aurora Serverless v1 資料庫叢集，請執行 CreateDBCluster 操作並為 `serverless` 參數指定 `EngineMode`。

您可選擇指定 `ScalingConfiguration` 參數以設定最小容量、最大容量，並在沒有連線時自動暫停。有效容量值包括：

- Aurora MySQL：1、2、4、8、16、32、64、128 和 256。
- Aurora PostgreSQL：2、4、8、16、32、64、192 和 384。

還原 Aurora Serverless v1 資料庫叢集

> **Important**
>
> AWS 已宣布 Aurora Serverless v1 的終止服務日期：2025 年 3 月 31 日。所有未在 2025 年 3 月 31 日之前遷移的 Aurora Serverless v1 叢集都會在維護時段期間遷移至 Aurora Serverless v2。如果升級失敗，Amazon Aurora 會在維護時段將 Serverless v1 叢集轉換為具有同等引擎版本的佈建叢集。如果適用，Amazon Aurora 將在 Amazon RDS 延長支援中註冊轉換後的已佈建叢集。如需更多詳細資訊，請參閱 RDS 延長支援。

當您透過 AWS CLI 或 RDS API 還原佈建的資料庫叢集快照時，您可設定 Aurora Serverless v1 資料庫叢集。

在您將快照還原至 Aurora Serverless v1 資料庫叢集時，您可以設定以下指定值：

- Minimum Aurora capacity unit (最小 Aurora 容量單位) – Aurora Serverless v1 可將容量降低至此容量單位下限。

- Maximum Aurora capacity unit (最大 Aurora 容量單位) – Aurora Serverless v1 可將容量提升至此容量單位上限。
- Timeout action (逾時動作) – 當容量修改因找不到擴展點而逾時時，所要採取的動作。Aurora Serverless v1如果設定為 Force scaling the capacity to the specified values... (強制將容量擴展為指定的值...) 選項，則資料庫叢集可以強制將資料庫叢集設定為新的容量設定。或者，如果您沒有選擇此選項，它可以復原容量變更以取消容量變更。如需更多詳細資訊，請參閱 容量變更時的逾時動作。
- Pause after inactivity (在無動作後暫停) – 資料庫經過多久時間沒有流量後，將處理容量縮減為零。當資料庫流量恢復，Aurora 會自動恢復處理容量，並調整規模以應付流量。

如需從快照還原資料庫叢集的一般資訊，請參閱 從資料庫叢集快照還原。

AWS CLI

當您透過 Aurora Serverless、AWS 管理主控台 或 RDS API 還原佈建的資料庫叢集快照時，您可設定 AWS CLI 資料庫叢集。

在您將快照還原至 Aurora Serverless 資料庫叢集時，您可以設定以下指定值：

- Minimum Aurora capacity unit (最小 Aurora 容量單位) – Aurora Serverless 可將容量降低至此容量單位下限。
- Maximum Aurora capacity unit (最大 Aurora 容量單位) – Aurora Serverless 可將容量提升至此容量單位上限。
- Timeout action (逾時動作) – 當容量修改因找不到擴展點而逾時時，所要採取的動作。Aurora Serverless v1如果設定為 Force scaling the capacity to the specified values... (強制將容量擴展為指定的值...) 選項，則資料庫叢集可以強制將資料庫叢集設定為新的容量設定。或者，如果您沒有選擇此選項，它可以復原容量變更以取消容量變更。如需更多詳細資訊，請參閱 容量變更時的逾時動作。
- Pause after inactivity (在無動作後暫停) – 資料庫經過多久時間沒有流量後，將處理容量縮減為零。當資料庫流量恢復，Aurora 會自動恢復處理容量，並調整規模以應付流量。

> **Note**
>
> 資料庫叢集快照的版本必須與 Aurora Serverless v1 相容。如需支援版本的清單，請參閱 Aurora Serverless v1。

若要將快照還原至與 MySQL 5.7 相容的 Aurora Serverless v1 叢集，請包含下列其他參數：

- `--engine` **`aurora-mysql`**
- `--engine-version` **`5.7`**

`--engine` 和 `--engine-version` 參數可讓您從與 MySQL 5.6 相容的 Aurora 或 Aurora Serverless v1 快照，建立與 MySQL 5.7 相容的 Aurora Serverless v1 叢集。以下範例會將快照從名為 *`mydbclustersnapshot`* 的 MySQL 5.6 相容叢集，還原至名為 *`mynewdbcluster`* 的 MySQL 5.7 相容 Aurora Serverless v1 叢集。

對於 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine-mode serverless \
    --engine aurora-mysql \
    --engine-version 5.7
```

在 Windows 中：

```
aws rds restore-db-cluster-from-snapshot ^
    --db-instance-identifier mynewdbcluster ^
    --db-snapshot-identifier mydbclustersnapshot ^
    --engine aurora-mysql ^
    --engine-version 5.7
```

您可以選擇指定 `--scaling-configuration` 選項以設定最小容量、最大容量，並在沒有連線時自動暫停。有效容量值包括：

- Aurora MySQL：1、2、4、8、16、32、64、128 和 256。
- Aurora PostgreSQL：2、4、8、16、32、64、192 和 384。

在以下範例中，您會從先前建立的資料庫叢集快照 (名為 *`mydbclustersnapshot`*) 還原至名為 *`mynewdbcluster`* 的新資料庫叢集。您可以設定 `--scaling-configuration`，以便新的 Aurora Serverless v1 資料庫叢集可視需要從 8 個 ACU 擴展到 64 個 ACU (Aurora 容量單位)，以處理工作負載。處理完成後，在 1000 秒後沒有連線支援的情況下，叢集會關閉，直到連線請求提示重新啟動為止。

對於 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine-mode serverless --scaling-configuration
 MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=10
```

在 Windows 中：

```
aws rds restore-db-cluster-from-snapshot ^
    --db-instance-identifier mynewdbcluster ^
    --db-snapshot-identifier mydbclustersnapshot ^
    --engine-mode serverless --scaling-configuration
 MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=10
```

RDS API

若要使用 RDS API，在您從資料庫叢集進行還原時設定 Aurora Serverless v1 資料庫叢集，請執行 RestoreDBClusterFromSnapshot 操作，並為 `serverless` 參數指定 `EngineMode`。

您可選擇指定 `ScalingConfiguration` 參數以設定最小容量、最大容量，並在沒有連線時自動暫停。有效容量值包括：

- Aurora MySQL：1、2、4、8、16、32、64、128 和 256。
- Aurora PostgreSQL：2、4、8、16、32、64、192 和 384。

修改 Aurora Serverless v1 資料庫叢集

> **Important**
>
> AWS 已宣布 的end-of-lifeAurora Serverless v1：2025 年 3 月 31 日。所有未在 2025 年 3 月 31 日之前遷移的 Aurora Serverless v1 叢集都會在維護時段期間遷移至 Aurora Serverless v2。如果升級失敗，Amazon Aurora 會在維護時段將 Serverless v1 叢集轉換為具有同等引擎版本的佈建叢集。如果適用，Amazon Aurora 將在 Amazon RDS 延長支援中註冊轉換後的已佈建叢集。如需詳細資訊，請參閱RDS 延長支援。

設定資料庫叢集之後，您可以使用 AWS 管理主控台、 Aurora Serverless v1 AWS CLI或 RDS API 修改特定屬性。您可以修改的大部分屬性與其他類型的 Aurora 叢集相同。

以下是與 Aurora Serverless v1 最相關的變更：

- 修改擴展組態
- 升級主要版本
- 從 Aurora Serverless v1 轉換為已佈建

修改 Aurora Serverless v1 資料庫叢集的擴展組態

您可設定資料庫叢集的容量下限和上限。每個容量單位相等於一個特定的運算和記憶體組態。Aurora Serverless 會自動建立 CPU 使用率、連線數及可用記憶體閾值的規模調整規則。您也可以設定 Aurora Serverless 是否在沒有活動時暫停資料庫，並在開始活動時恢復。

您可以為擴展組態設定以下特定值：

- Minimum Aurora capacity unit (最小 Aurora 容量單位) – Aurora Serverless 可將容量降低至此容量單位下限。
- Maximum Aurora capacity unit (最大 Aurora 容量單位) – Aurora Serverless 可將容量提升至此容量單位上限。
- Autoscaling timeout and action (自動擴展逾時和動作) – 此區段可指定 Aurora Serverless 在逾時之前等待尋找擴展點的時間長度。它也可以指定在容量修改因找不到擴展點而逾時時所要採取的動作。Aurora 能夠強制容量變更，以盡快將容量設為指定值。或者，也可復原該容量變更以進行取消。如需詳細資訊，請參閱容量變更時的逾時動作。
- 無動作後暫停 - 使用選用的叢集閒置時將容量調整為 0 個 ACU 設定，在資料庫非作用中時將該資料庫調整為零處理容量。當資料庫流量恢復，Aurora 會自動恢復處理容量，並調整規模以應付流量。

> **Note**
>
> 當您修改 Aurora Serverless 資料庫叢集的容量範圍時，無論您選擇立即套用或在下一個排程維護時段套用變更，該變更都會立即發生。

主控台

您可以使用 AWS 管理主控台來修改 Aurora 資料庫叢集的擴展組態。

修改 Aurora Serverless v1 資料庫叢集

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您要修改的 Aurora Serverless v1 資料庫叢集。
4. 在 Actions (動作) 中，選擇 Modify cluster (修改叢集)。
5. 在 Capacity settings (容量設定) 區段中修改擴展組態。
6. 選擇繼續。
7. 在修改資料庫叢集頁面上檢視您所做的修改，並在套用時選擇。
8. 選擇修改叢集。

AWS CLI

若要使用 Aurora Serverless v1 修改資料庫叢集的擴展組態 AWS CLI，請執行 modify-db-cluster AWS CLI 命令。指定 `--scaling-configuration` 選項以設定最小容量、最大容量，並在沒有連線時自動暫停。有效容量值包括：

- Aurora MySQL：1、2、4、8、16、32、64、128 和 256。
- Aurora PostgreSQL：2、4、8、16、32、64、192 和 384。

在此範例中，您將修改名為 *`sample-cluster`* 的 Aurora Serverless v1 資料庫叢集擴展組態。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange
```

RDS API

您可以使用 ModifyDBCluster API 操作來修改 Aurora 資料庫叢集的擴展組態。指定 `ScalingConfiguration` 參數以設定最小容量、最大容量，並在沒有連線時自動暫停。有效容量值包括：

- Aurora MySQL：1、2、4、8、16、32、64、128 和 256。
- Aurora PostgreSQL：2、4、8、16、32、64、192 和 384。

升級 Aurora Serverless v1 資料庫叢集的主要版本

> **Important**
>
> AWS 已宣布 的end-of-lifeAurora Serverless v1：2025 年 3 月 31 日。所有未在 2025 年 3 月 31 日之前遷移的 Aurora Serverless v1 叢集都會在維護時段期間遷移至 Aurora Serverless v2。如果升級失敗，Amazon Aurora 會在維護時段將 Serverless v1 叢集轉換為具有同等引擎版本的佈建叢集。如果適用，Amazon Aurora 將在 Amazon RDS 延長支援中註冊轉換後的已佈建叢集。如需詳細資訊，請參閱RDS 延長支援。

對於與 PostgreSQL 11 相容的 Aurora Serverless v1 資料庫叢集，您可以將其主要版本升級至對應的 PostgreSQL 13 相容版本。

主控台

您可以使用 AWS 管理主控台來執行 Aurora Serverless v1 資料庫叢集的就地升級。

升級 Aurora Serverless v1 資料庫叢集

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您要升級的 Aurora Serverless v1 資料庫叢集。
4. 在 Actions (動作) 中，選擇 Modify cluster (修改叢集)。
5. 對於版本，請選擇 Aurora PostgreSQL 第 13 版的版本號碼。

   下列範例顯示從 Aurora PostgreSQL 11.16 就地升級至 13.9。



如果您執行主要版本升級，請將所有其他屬性保持不變。若要變更任何一個其他屬性，請在升級完成後執行另一個修改操作。

6. 選擇繼續。
7. 在修改資料庫叢集頁面上檢視您所做的修改，並在套用時選擇。
8. 選擇修改叢集。

AWS CLI

若要從 PostgreSQL 11 相容的 Aurora Serverless v1 資料庫叢集就地升級至 PostgreSQL 13 相容資料庫叢集，請指定其 Aurora PostgreSQL 版本 13 版本編號與 Aurora Serverless v1 相容的 `--engine-version` 參數。亦需包含 `--allow-major-version-upgrade` 參數。

在此範例中，您可以修改 PostgreSQL 11 相容 Aurora Serverless v1 資料庫叢集 (名為 `sample-cluster`) 的主要版本。這麼做會執行就地升級至 PostgreSQL 13 相容的 Aurora Serverless v1 資料庫叢集。

```
aws rds modify-db-cluster \
```

```
    --db-cluster-identifier sample-cluster \
    --engine-version 13.serverless_12 \
    --allow-major-version-upgrade
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --engine-version 13.serverless_12 ^
    --allow-major-version-upgrade
```

RDS API

若要從 PostgreSQL 11 相容的 Aurora Serverless v1 資料庫叢集就地升級至 PostgreSQL 13 相容資料庫叢集，請指定其 Aurora PostgreSQL 版本 13 版本編號與 Aurora Serverless v1 相容的 `EngineVersion` 參數。亦需包含 `AllowMajorVersionUpgrade` 參數。

將 Aurora Serverless v1 資料庫叢集轉換至已佈建資料庫叢集

您可將 Aurora Serverless v1 資料庫叢集轉換為已佈建資料庫叢集。若要執行轉換，請使用 AWS CLI 或 Amazon RDS API 將資料庫執行個體類別變更為佈建。使用下列步驟來修改資料庫執行個體類別。

下列範例示範如何使用 AWS CLI 將 Aurora Serverless v1 資料庫叢集轉換為佈建的叢集。

AWS CLI

若要將Aurora Serverless v1資料庫叢集轉換為佈建叢集，請執行 modify-db-cluster AWS CLI 命令。

下列是必要參數：

- `--db-cluster-identifier` – 您要轉換為已佈建的 Aurora Serverless v1 資料庫叢集。
- `--engine-mode` – 使用值 `provisioned`。
- `--allow-engine-mode-change`
- `--db-cluster-instance-class` – 根據 Aurora Serverless v1 資料庫叢集的容量，選擇已佈建資料庫叢集的資料庫執行個體類別。

在此範例中，請轉換名為 `sample-cluster` 的 Aurora Serverless v1 資料庫叢集，並使用 `db.r5.xlarge` 資料庫執行個體類別。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
--db-cluster-identifier sample-cluster \
--engine-mode provisioned \
--allow-engine-mode-change \
--db-cluster-instance-class db.r5.xlarge
```

在 Windows 中：

```
aws rds modify-db-cluster ^
--db-cluster-identifier sample-cluster ^
--engine-mode provisioned ^
--allow-engine-mode-change ^
--db-cluster-instance-class db.r5.xlarge
```

下列範例示範如何使用 Amazon RDS API 將 Aurora Serverless v1 資料庫叢集轉換為已佈建叢集。

RDS API

若要將 Aurora Serverless v1 資料庫叢集轉換為已佈建叢集，請執行 ModifyDBCluster API 操作。

下列是必要參數：

- `DBClusterIdentifier` – 您要轉換為已佈建的 Aurora Serverless v1 資料庫叢集。
- `EngineMode` – 使用值 `provisioned`。
- `AllowEngineModeChange`
- `DBClusterInstanceClass` – 根據 Aurora Serverless v1 資料庫叢集的容量，選擇已佈建資料庫叢集的資料庫執行個體類別。

從 Aurora Serverless v1 資料庫叢集轉換為已佈建叢集時的考量

Aurora Serverless v1 資料庫叢集轉換為已佈建叢集時，適用下列考量：

- 您可以將此轉換做為將資料庫叢集從 Aurora Serverless v1 升級到 Aurora Serverless v2 的一部分。如需詳細資訊，請參閱從 Aurora Serverless v1 叢集升級至 Aurora Serverless v2。
- 轉換過程會在資料庫叢集中建立讀取器資料庫執行個體、將讀取器執行個體升級為寫入器執行個體，並刪除原始 Aurora Serverless v1 執行個體。轉換資料庫叢集時，無法同時執行任何其他修改，例如變更資料庫引擎版本或資料庫叢集參數群組。系統會立即套用轉換操作且無法還原。

- 轉換期間若發生錯誤，系統會擷取資料庫叢集的備份資料庫叢集快照。資料庫叢集快照的識別符形式為 `pre-modify-engine-mode-DB_cluster_identifier-timestamp`。
- Aurora 會針對已佈建的資料庫叢集使用目前的預設資料庫次要引擎版本。
- 若您沒有為轉換後的資料庫叢集提供資料庫執行個體類別，Aurora 會根據原始 Aurora Serverless v1 資料庫叢集的最大容量，為您建議一個執行個體類別。下表顯示執行個體類別映射的建議容量。

Serverless 最大容量 (ACU)	已佈建的資料庫執行個體類別
1	db.t3.small
2	db.t3.medium
4	db.t3.large
8	db.r5.large
16	db.r5.xlarge
32	db.r5.2xlarge
64	db.r5.4xlarge
128	db.r5.8xlarge
192	db.r5.12xlarge
256	db.r5.16xlarge
384	db.r5.24xlarge

> **Note**
>
> 根據您所選的資料庫執行個體類別和資料庫使用情況，您可能會看到與 Aurora Serverless v1 不同的佈建資料庫叢集成本。
> 若您將 Aurora Serverless v1 資料庫叢集轉換為爆量 (db.t*) 資料庫執行個體類別，使用該資料庫叢集可能會產生額外費用。如需詳細資訊，請參閱資料庫執行個體類別的類型。

手動擴展 Aurora Serverless v1 資料庫叢集容量

> **Important**
>
> AWS 已宣布 Aurora Serverless v1 的終止服務日期：2025 年 3 月 31 日。所有未在 2025 年 3 月 31 日之前遷移的 Aurora Serverless v1 叢集都會在維護時段期間遷移至 Aurora Serverless v2。如果升級失敗，Amazon Aurora 會在維護時段將 Serverless v1 叢集轉換為具有同等引擎版本的佈建叢集。如果適用，Amazon Aurora 將在 Amazon RDS 延長支援中註冊轉換後的已佈建叢集。如需更多詳細資訊，請參閱 RDS 延長支援。

一般而言，Aurora Serverless v1 資料庫叢集會根據工作負載無縫擴展。然而，容量的擴展速度可能不一定能夠如此快速，以滿足突然的極端情況，例如交易的指數型成長。在這種情況下，您可以透過設定新的容量值來手動啟動擴展操作。在您明確設定容量後，Aurora Serverless v1 會自動調整資料庫叢集規模。它是根據調降規模時的冷卻時間來達成的。

您可以使用 Aurora Serverless v1、AWS 管理主控台 或 RDS API，明確地將 AWS CLI 資料庫叢集的容量設定為特定值。

主控台

您可使用 AWS 管理主控台 設定 Aurora 資料庫叢集的容量。

修改 Aurora Serverless v1 資料庫叢集

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您要修改的 Aurora Serverless v1 資料庫叢集。
4. 針對 Actions (動作)，選擇 Set capacity (設定容量)。
5. 在 Scale database capacity (調整資料庫容量) 視窗中，選擇下列項目：

    a. 在 Scale DB cluster to (將資料庫叢集調整為以下大小) 下拉式選擇工具，選擇您要用於資料庫叢集的新容量。

    b. 對於 If a seamless scaling point cannot be found (如果找不到無縫擴展點) 核取方塊中，選擇您想要的 Aurora Serverless v1 資料庫叢集之 `TimeoutAction` 設定的行為，如下所示：

    - 如果您希望 Aurora Serverless v1 在逾時之前找不到擴展點時，將容量保持為不變，請不要勾選此選項。

- 如果您希望 Aurora Serverless v1 資料庫叢集即使在逾時之前找不到擴展點時，仍強制其變更其容量，請選取此選項。此選項可能會導致 Aurora Serverless v1 連線中斷，使其無法找到擴展點。

c. 若為 seconds (秒)，請輸入您希望讓 Aurora Serverless v1 資料庫叢集在逾時之前尋找擴展點的時間。您可以指定從 10 秒到 600 秒 (10 分鐘) 的任何時間。預設值為五分鐘 (300 秒)。下列範例會強制 Aurora Serverless v1 資料庫叢集縮減至 2 個 ACU，即使在五分鐘內找不到擴展點也是如此。



6. 選擇 Apply (套用)。

若要進一步了解擴展點 (`TimeoutAction`) 和冷卻時間，請參閱 Aurora Serverless v1 的自動調整規模。

AWS CLI

若要使用 Aurora Serverless v1 來設定 AWS CLI 資料庫叢集的容量，請執行 modify-current-db-cluster-capacity AWS CLI 命令，並指定 `--capacity` 選項。有效容量值包括：

- Aurora MySQL：1、2、4、8、16、32、64、128 和 256。
- Aurora PostgreSQL：2、4、8、16、32、64、192 和 384。

在此範例中，您將名為 *sample-cluster* 的 Aurora Serverless v1 資料庫叢集容量設為 *64*。

```
aws rds modify-current-db-cluster-capacity --db-cluster-identifier sample-cluster --
capacity 64
```

RDS API

您可使用 ModifyCurrentDBClusterCapacity API 操作設定 Aurora 資料庫叢集容量。指定 `Capacity` 參數。有效容量值包括：

- Aurora MySQL：1、2、4、8、16、32、64、128 和 256。
- Aurora PostgreSQL：2、4、8、16、32、64、192 和 384。

檢視 Aurora Serverless v1 資料庫叢集

> **Important**
>
> AWS 已宣布 Aurora Serverless v1 的終止服務日期：2025 年 3 月 31 日。所有未在 2025 年 3 月 31 日之前遷移的 Aurora Serverless v1 叢集都會在維護時段期間遷移至 Aurora Serverless v2。如果升級失敗，Amazon Aurora 會在維護時段將 Serverless v1 叢集轉換為具有同等引擎版本的佈建叢集。如果適用，Amazon Aurora 將在 Amazon RDS 延長支援中註冊轉換後的已佈建叢集。如需更多詳細資訊，請參閱 RDS 延長支援。

在您建立一個或以上的 Aurora Serverless v1 資料庫叢集後，您就可以檢視哪些資料庫叢集類型為 Serverless (無伺服器)，哪一些為 Instance (執行個體)。您也可以檢視每個 Aurora Serverless v1 資料庫叢集目前正在使用的 Aurora 容量單位 (ACU) 數量。每一個 ACU 都是處理 (CPU) 與記憶體 (RAM) 容量的組合。

檢視 Aurora Serverless v1 資料庫叢集

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在 AWS 管理主控台 的右上角，選擇您要建立 AWS 區域 資料庫叢集的 Aurora Serverless v1。
3. 在導覽窗格中，選擇 Databases (資料庫)。

每個資料庫叢集的資料庫叢集類型顯示於 Role (角色) 之下。Aurora Serverless v1 資料庫叢集顯示類型為 Serverless (無伺服器)。您可以在 Size (大小) 下檢視 Aurora Serverless v1 資料庫叢集的現有容量。



4. 選擇一個 Aurora Serverless v1 資料庫叢集名稱以顯示其詳細資訊。

   在 Connectivity & security (連線與安全性) 索引標籤上，記下資料庫端點。使用此端點連線至 Aurora Serverless v1 資料庫叢集。



選取 Configuration (組態) 索引標籤以檢視容量設定。



當資料庫叢集擴展、縮減、暫停或恢復時，將產生擴展事件。選取 Logs & events (日誌和事件) 索引標籤以查看最近的事件。以下圖片顯示了這些事件的範例。



監控 Aurora Serverless v1 資料庫叢集的容量和擴展事件

您可以檢視 CloudWatch 中的 Aurora Serverless v1 資料庫叢集，以使用 `ServerlessDatabaseCapacity` 指標監控分配給資料庫叢集的容量。您也可以監控所有的標準 Aurora CloudWatch 指標，例如 `CPUUtilization`、`DatabaseConnections`、`Queries` 等。

您可讓 Aurora 發佈部分或所有資料庫日誌到 CloudWatch。您可透過啟用在資料庫叢集參數群組中與 general_log 叢集關聯的組態參數 (例如 slow_query_log 和 Aurora Serverless v1) 選擇要發佈的日誌。與已佈建的叢集不同，Aurora Serverless v1 叢集可讓您無須在資料庫叢集設定中指定要上傳至 CloudWatch 的日誌類別。Aurora Serverless v1 叢集會自動上傳所有可用日誌。當您停用日誌組態參數時，日誌將停止發佈至 CloudWatch。如果已不需要日誌，您也可以在 CloudWatch 中刪除這些日誌。

若要為您的 Aurora Serverless v1 資料庫叢集開始使用 Amazon CloudWatch，請參閱 使用 Amazon CloudWatch 檢視 Aurora Serverless v1 日誌。若要進一步了解如何透過 CloudWatch 監控 Aurora 資料庫叢集，請參閱 在 Amazon CloudWatch 中監控日誌事件。

若要連線至 Aurora Serverless v1 資料庫叢集，請使用資料庫端點。如需更多詳細資訊，請參閱 連接至 Amazon Aurora 資料庫叢集。

> **Note**
>
> 您不能直接連線至 Aurora Serverless v1 資料庫叢集中的特定資料庫執行個體。

刪除 Aurora Serverless v1 資料庫叢集

> **Important**
>
> AWS 已宣布 Aurora Serverless v1 的終止服務日期：2025 年 3 月 31 日。所有未在 2025 年 3 月 31 日之前遷移的 Aurora Serverless v1 叢集都會在維護時段期間遷移至 Aurora Serverless v2。如果升級失敗，Amazon Aurora 會在維護時段將 Serverless v1 叢集轉換為具有同等引擎版本的佈建叢集。如果適用，Amazon Aurora 將在 Amazon RDS 延長支援中註冊轉換後的已佈建叢集。如需更多詳細資訊，請參閱 RDS 延長支援。

視您建立 Aurora Serverless v1 資料庫叢集的方式而定，預設可能會開啟刪除保護。您無法立即刪除已啟用刪除保護的 Aurora Serverless v1 資料庫叢集。若要使用 Aurora Serverless v1 刪除具有刪除保護的 AWS 管理主控台 資料庫叢集，您必須先修改叢集以移除此保護。如需有關使用此任務的 AWS CLI 資訊，請參閱 AWS CLI。

使用 AWS 管理主控台 來停用刪除保護

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。

2. 在導覽窗格中，選擇 DB clusters (資料庫叢集)。
3. 從清單中選擇 Aurora Serverless v1 資料庫叢集。
4. 選擇 Modify (修改) 以開啟資料庫叢集的組態。「修改資料庫叢集」頁面會開啟 Aurora Serverless v1 資料庫叢集的設定、容量設定和其他組態詳細資訊。刪除保護位於 Additional configuration (其他組態) 區段中。
5. 取消勾選 Additional configuration (其他組態) 屬性卡中的 Enable deletion protection (啟用刪除保護) 核取方塊。
6. 選擇 Continue (繼續)。此時系統會顯示 Summary of modifications (修改摘要)。
7. 選擇 Modify cluster (修改叢集) 以接受修改摘要。您也可以選擇 Back (上一步) 修改變更，或 Cancel (取消) 捨棄變更。

在刪除保護不再處於作用中狀態之後，您就可以使用 Aurora Serverless v1 刪除 AWS 管理主控台 資料庫叢集。

主控台

刪除 Aurora Serverless v1 資料庫叢集

1. 登入 AWS 管理主控台，開啟位於 https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在 Resources (資源) 區段中，選擇 DB Clusters (資料庫叢集)。
3. 選擇您要刪除的 Aurora Serverless v1 資料庫叢集。
4. 對於 Actions (動作)，請選擇 Delete (刪除)。系統會提示您確認是否要刪除 Aurora Serverless v1 資料庫叢集。
5. 我們建議您保留預先選取的選項：
   - Create final snapshot? (是否建立最後的快照？) 的 Yes (是)
   - 在 Final snapshot name (最終快照名稱) 的 Aurora Serverless v1 資料庫叢集名稱加上 `-final-snapshot`。但是，您可以在此欄位中變更最終快照的名稱。



如果您為 Create final snapshot? (是否建立最後的快照？) 選擇 No (否)，您就無法使用快照或 point-in-time 恢復來還原資料庫叢集。

6. 選擇 Delete DB cluster (刪除資料庫叢集)。

Aurora Serverless v1 會刪除您的資料庫叢集。如果您選擇擁有最終快照，則會在 Aurora Serverless v1 資料庫叢集刪除之前看到資料庫叢集的狀態變更為「備份」，且不再出現在清單中。

AWS CLI

開始之前，請使用 AWS 存取金鑰 ID、AWS 私密存取金鑰和 Aurora Serverless v1 資料庫叢集所在的 AWS 區域 設定 AWS CLI。如需詳細資訊，請參閱《AWS Command Line Interface 使用者指南》中的組態基礎概念。

您必須先停用使用此選項設定之叢集的刪除保護，才能刪除 Aurora Serverless v1 資料庫叢集。如果您嘗試刪除已啟用此保護選項的叢集，您會看到下列錯誤訊息。

```
An error occurred (InvalidParameterCombination) when calling the DeleteDBCluster
  operation: Cannot delete protected Cluster, please disable deletion protection and
 try again.
```

您可以使用 modify-db-cluster Aurora Serverless v1 命令，變更 AWS CLI 資料庫叢集的刪除保護設定，如下所示：

```
aws rds modify-db-cluster --db-cluster-identifier your-cluster-name --no-deletion-
protection
```

此命令會傳回指定資料庫叢集的修訂屬性。您現在就可以刪除 Aurora Serverless v1 資料庫叢集。

我們建議您每當刪除 Aurora Serverless v1 資料庫叢集時，一律建立最終快照。使用 AWS CLI delete-db-cluster 的下列範例會說明做法。您可以提供資料庫叢集的名稱和快照的名稱。

對於 Linux、macOS 或 Unix：

```
aws rds delete-db-cluster --db-cluster-identifier \
  your-cluster-name --no-skip-final-snapshot \
  --final-db-snapshot-identifier name-your-snapshot
```

在 Windows 中：

```
aws rds delete-db-cluster --db-cluster-identifier ^
  your-cluster-name --no-skip-final-snapshot ^
  --final-db-snapshot-identifier name-your-snapshot
```

Aurora Serverless v1 和 Aurora 資料庫引擎版本

> **Important**
>
> AWS 已宣布 的end-of-lifeAurora Serverless v1：2025 年 3 月 31 日。所有未在 2025 年 3 月 31 日之前遷移的 Aurora Serverless v1 叢集都會在維護時段期間遷移至 Aurora Serverless v2。如果升級失敗，Amazon Aurora 會在維護時段將 Serverless v1 叢集轉換為具有同等引擎版本的佈建叢集。如果適用，Amazon Aurora 將在 Amazon RDS 延長支援中註冊轉換後的已佈建叢集。如需詳細資訊，請參閱RDS 延長支援。

Aurora Serverless v1 僅適用於特定 AWS 區域 Aurora MySQL 和 Aurora PostgreSQL 版本。如需 AWS 區域 支援的目前清單Aurora Serverless v1，以及每個區域中可用的特定 Aurora MySQL 和 Aurora PostgreSQL 版本，請參閱 Aurora Serverless v1。

Aurora Serverless v1 會使用其關聯的 Aurora 資料庫引擎，來識別支援的每個資料庫引擎的特定支援版本，如下所示：

- Aurora MySQL Serverless

- Aurora PostgreSQL Serverless

當資料庫引擎的次要版本變得可供 Aurora Serverless v1 使用時，系統會在推出 AWS 區域 的各個 Aurora Serverless v1 中自動套用這些版本。換句話說，您不需要升級 Aurora Serverless v1 資料庫叢集，就能在叢集資料庫引擎的全新次要版本可供 Aurora Serverless v1 使用時，取得此版本。

Aurora MySQL Serverless

Aurora Serverless v1 僅適用於特定 AWS 區域 和特定 Aurora MySQL 版本。如需 AWS 區域 支援的最新清單，Aurora Serverless v1以及每個區域中可用的特定 Aurora MySQL 版本，請參閱 Aurora Serverless v1 搭配 Aurora MySQL。

若要了解執行 Aurora MySQL 第 2 版的增強功能和錯誤修正，請參閱《Aurora MySQL 版本備註》中的 Amazon Aurora MySQL 第 2 版的資料庫引擎更新。

若要使用較新版本的 Aurora MySQL，您可以使用 Aurora Serverless v2。如需可與 搭配使用的 AWS 區域 和 Aurora MySQL 版本Aurora Serverless v2，請參閱 Aurora Serverless v2 搭配 Aurora MySQL。如需 Aurora Serverless v2 的相關用量資訊，請參閱 使用 Aurora Serverless v2。

Aurora PostgreSQL Serverless

Aurora Serverless v1 僅適用於特定 AWS 區域 和特定 Aurora PostgreSQL 版本。如需 AWS 區域 支援的目前清單，Aurora Serverless v1以及每個區域中可用的特定 Aurora PostgreSQL 版本，請參閱 Aurora Serverless v1 搭配 Aurora PostgreSQL。

如果想要將 Aurora PostgreSQL 用於 Aurora Serverless v1 資料庫叢集，您可以使用 Aurora PostgreSQL 13 相容版本。Aurora PostgreSQL 相容版本 的次要版本僅包含回溯相容的變更。當 Aurora PostgreSQL 次要版本變成可供您 AWS 區域的 Aurora Serverless v1 使用時，Aurora Serverless v1 資料庫叢集就會以透明的方式升級。

若要使用較新版本的 Aurora PostgreSQL，您可以使用 Aurora Serverless v2。如需可與 搭配使用的 AWS 區域 和 Aurora PostgreSQL 版本Aurora Serverless v2，請參閱 Aurora Serverless v2 搭配 Aurora PostgreSQL。如需 Aurora Serverless v2 的相關用量資訊，請參閱 使用 Aurora Serverless v2。

Aurora Serverless v1 的自動次要版本升級

當資料庫引擎的次要版本可供 使用時Aurora Serverless v1，它們會自動套用在 AWS 區域 Aurora Serverless v1 可用的各種 中。換句話說，您不需要升級 Aurora Serverless v1 資料庫叢集，就能在叢集資料庫引擎的全新次要版本可供 Aurora Serverless v1 使用時，取得此版本。

使用 Amazon RDS 資料 API

透過使用 RDS 資料 API (資料 API)，您可以使用網路服務界面來操作 Aurora 資料庫叢集。資料 API 不需要資料庫叢集的持續連線。而是提供安全的 HTTP 端點並與 AWS SDKs整合。您可以使用端點來執行 SQL 陳述式，而不需管理連線。

使用者不需要透過呼叫資料 API 來傳遞登入資料，因為資料 API 會使用存放於 中的資料庫登入資料 AWS Secrets Manager。若要將憑證儲存在 Secrets Manager 中，必須授與使用者使用 Secrets Manager 以及資料 API 的適當許可。如需授權使用者的詳細資訊，請參閱授權 Amazon RDS 資料 API 的存取。

您也可以使用資料 API 將 Amazon Aurora 與其他 AWS 應用程式整合，例如 AWS Lambda AWS AppSync、 和 AWS Cloud9。資料 API 會提供更安全的 AWS Lambda使用方式。這可讓您存取資料庫叢集，而不需將 Lambda 函數設定為在 Virtual Private Cloud (VPC) 存取資源。如需詳細資訊，請參閱AWS Lambda、AWS AppSync及AWS Cloud9。

您可以在建立 Aurora 資料庫叢集時啟用資料 API。您也可以稍後修改組態。如需詳細資訊，請參閱啟用 Amazon RDS 資料 API。

啟用資料 API 之後，您也可以使用查詢編輯器執行臨機操作查詢，而無需設定查詢工具，來存取 VPC 中的 Aurora。如需詳細資訊，請參閱使用 Aurora 查詢編輯器。

主題

- Amazon RDS 資料 API 的區域和版本可用性
- 搭配 Amazon RDS 資料 API 使用 IPv6
- Amazon RDS 資料 API 的限制
- 使用 Aurora Serverless v1 叢集比較 Aurora Serverless v2 和佈建叢集的 Amazon RDS 資料 API 行為
- 授權 Amazon RDS 資料 API 的存取
- 啟用 Amazon RDS 資料 API
- 建立 Amazon RDS 資料 API 的 Amazon VPC 端點 (AWS PrivateLink)
- 呼叫 Amazon RDS 資料 API
- 使用適用於 RDS 資料 API 的 Java 用戶端程式庫

- 以 JSON 格式處理 Amazon RDS 資料 API 查詢結果
- 針對 Amazon RDS 資料 API 進行疑難排解
- 使用 記錄 Amazon RDS Data API 呼叫 AWS CloudTrail
- 使用 Performance Insights 監控 RDS 資料 API 查詢

Amazon RDS 資料 API 的區域和版本可用性

如需資料 API 可用區域和引擎版本的相關資訊，請參閱下列各節。

叢集類型	區域和版本可用性
Aurora PostgreSQL 已佈建和 Serverless 第 2 版	具有 Aurora PostgreSQL Serverless 第 2 版和佈建的資料 API
Aurora MySQL 已佈建和 Serverless 第 2 版	具有 Aurora MySQL Serverless 第 2 版和佈建的資料 API
Aurora PostgreSQL Serverless 第 1 版	具有 Aurora PostgreSQL Serverless 第 1 版的資料 API
Aurora MySQL Serverless 第 1 版	具有 Aurora MySQL Serverless 第 1 版的資料 API

如果您在透過命令列界面或 API 存取資料 API 時，需要 FIPS 140-2 驗證的加密模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-2 概觀。

搭配 Amazon RDS 資料 API 使用 IPv6

Amazon RDS 資料 API 透過雙堆疊端點支援 IPv6 連線。這可讓您使用 IPv6 地址連線至資料 API，同時保持與 IPv4 的回溯相容性。

IPv6 端點支援

資料 API 提供同時支援 IPv4 和 IPv6 連線的雙堆疊端點。這些端點使用 `.aws` 網域 (而非傳統`.amazonaws.com`網域)。

可用的端點類型

公有雙堆疊端點

格式：`rds-data.region.api.aws`

範例：`rds-data.us-east-1.api.aws`

FIPS 雙堆疊端點

格式：`rds-data-fips.region.api.aws`

範例：`rds-data-fips.us-east-1.api.aws`

PrivateLink IPv6 端點

透過支援 IPv6 的 VPC 端點提供

允許 VPC 內的私有 IPv6 連線

舊版僅 IPv4 端點

現有 `.amazonaws.com` 端點繼續支援僅 IPv4 連線：

- `rds-data.region.amazonaws.com`
- `rds-data-fips.region.amazonaws.com`

> **Note**
>
> 舊版端點保持不變，以確保與現有應用程式的回溯相容性。

使用 IPv6 端點

若要搭配資料 API 使用 IPv6，請更新應用程式以使用新的雙堆疊端點。如果可用，應用程式會自動使用 IPv6，或回復為 IPv4。

如需在 VPC 中設定 IPv6 的一般指引，請參閱《Amazon VPC 使用者指南》中的遷移至 IPv6。

您可以透過兩種方式設定 IPv6 端點：

- 使用環境變數：在 IPv6 環境中設定 `AWS_USE_DUALSTACK_ENDPOINT=true`。 AWS CLI AWS SDKs 會自動建構適當的`api.aws`端點，而不需要您手動指定端點 URLs。
- 使用明確的端點 URLs：直接在 AWS CLI 命令或 SDK 組態中指定雙堆疊端點 URL，如以下範例所示。

AWS CLI 組態

透過指定端點 URL，將 AWS CLI 設定為使用 IPv6 端點：

對於 Linux、macOS 或 Unix：

```
aws rds-data execute-statement \
    --endpoint-url https://rds-data.us-east-1.api.aws \
    --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:my-cluster" \
    --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:my-secret" \
    --database "mydb" \
    --sql "SELECT * FROM users LIMIT 10"
```

在 Windows 中：

```
aws rds-data execute-statement ^
    --endpoint-url https://rds-data.us-east-1.api.aws ^
    --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:my-cluster" ^
    --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:my-secret" ^
    --database "mydb" ^
    --sql "SELECT * FROM users LIMIT 10"
```

AWS SDK 組態

AWS SDKs使用雙堆疊端點：

Python

```
import boto3

# Create RDS Data API client with IPv6 dual-stack endpoint
client = boto3.client(
    'rds-data',
    endpoint_url='https://rds-data.us-east-1.api.aws'
)
```

```
# Execute a SQL statement
response = client.execute_statement(
    resourceArn='arn:aws:rds:us-east-1:123456789012:cluster:my-cluster',
    secretArn='arn:aws:secretsmanager:us-east-1:123456789012:secret:my-secret',
    database='mydb',
    sql='SELECT * FROM users LIMIT 10'
)

print(response['records'])
```

Java

```
import software.amazon.awssdk.services.rdsdata.RdsDataClient;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementRequest;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementResponse;
import java.net.URI;

// Create RDS Data API client with IPv6 dual-stack endpoint
RdsDataClient client = RdsDataClient.builder()
    .endpointOverride(URI.create("https://rds-data.us-east-1.api.aws"))
    .build();

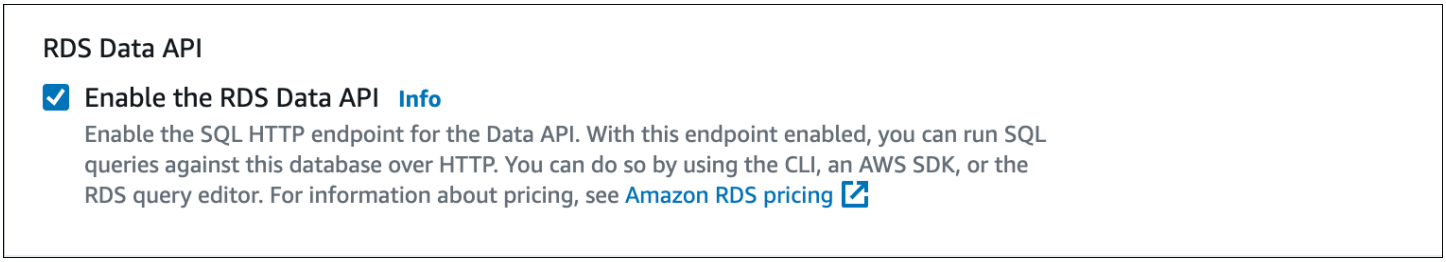
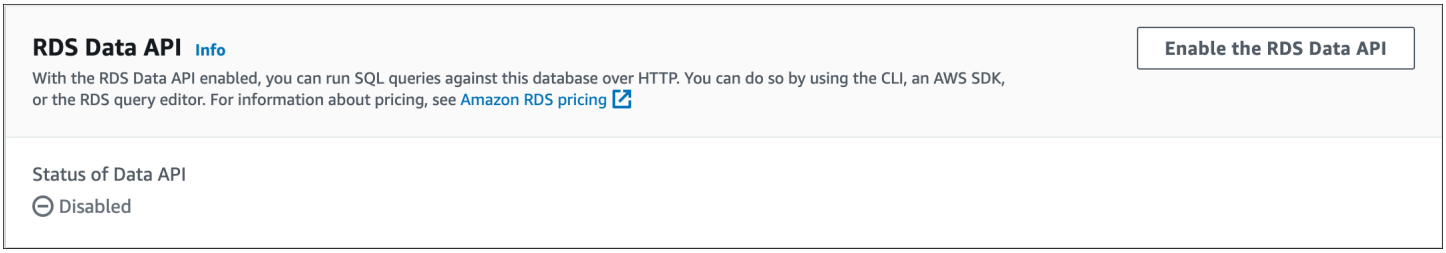
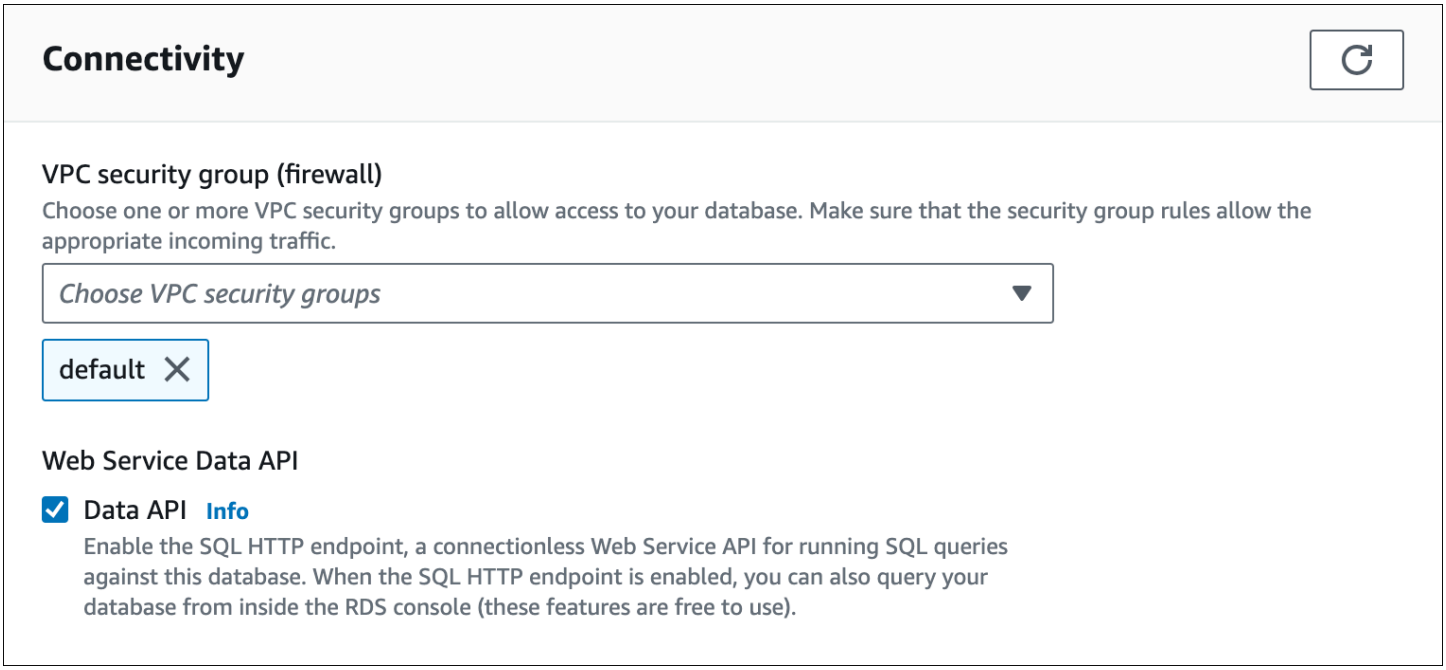
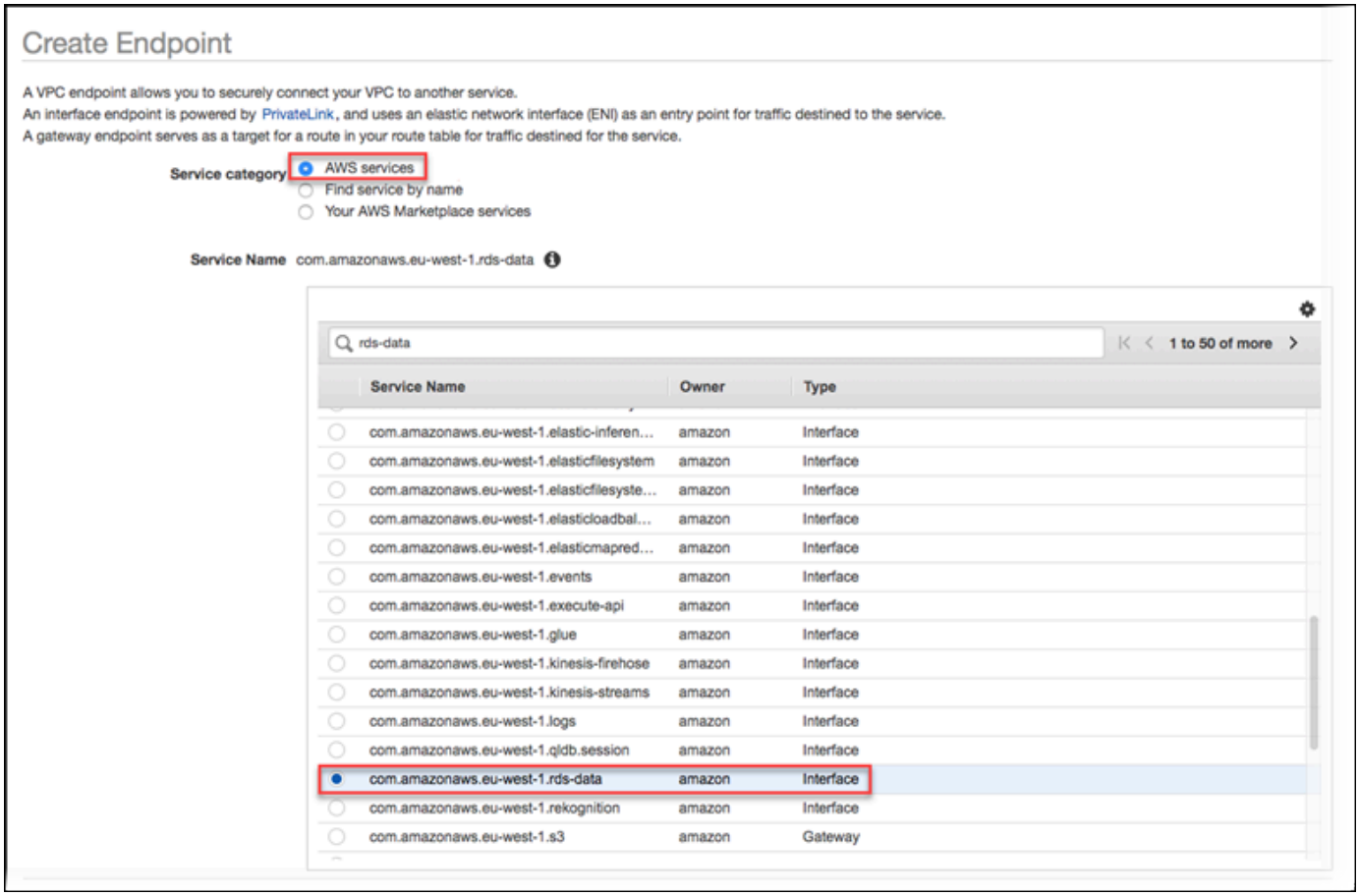
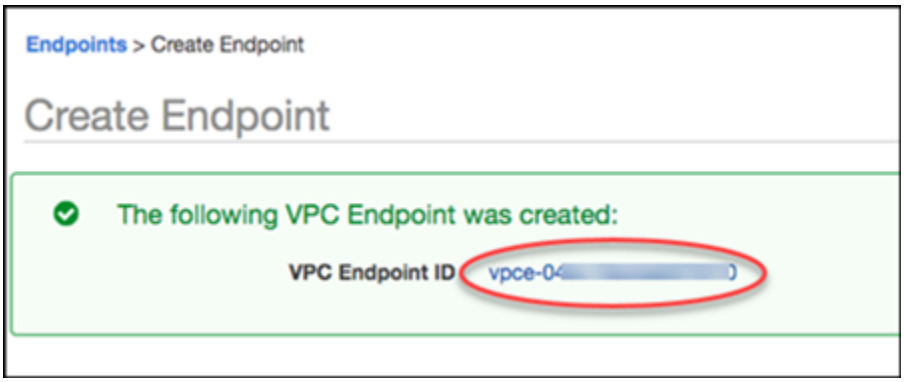
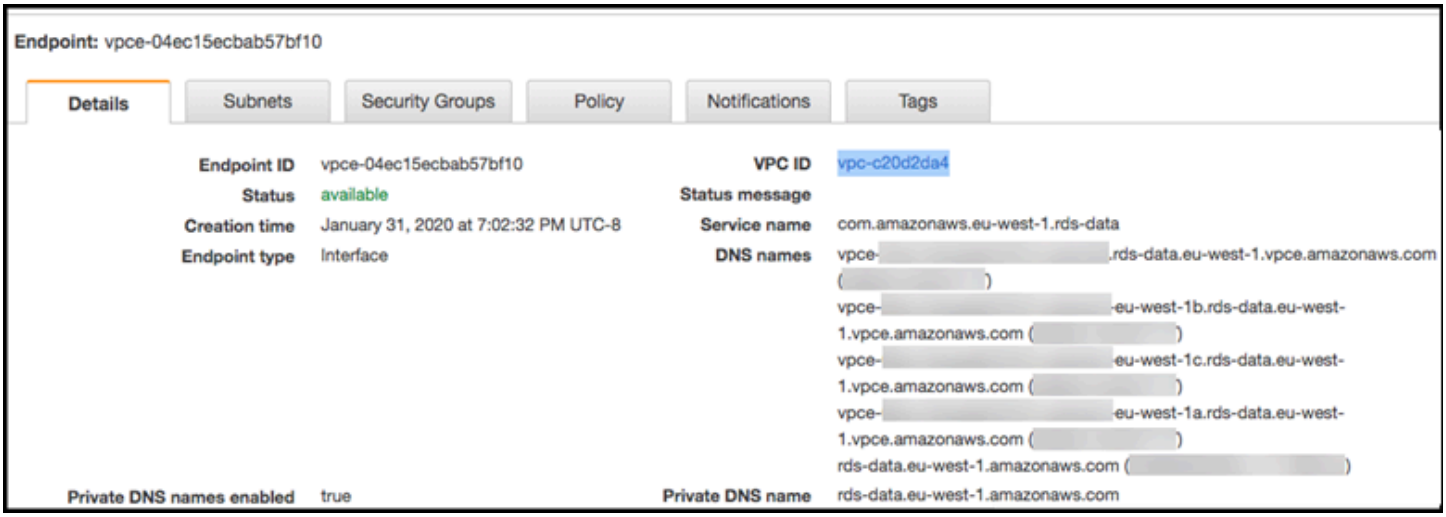
// Execute a SQL statement
ExecuteStatementRequest request = ExecuteStatementRequest.builder()
    .resourceArn("arn:aws:rds:us-east-1:123456789012:cluster:my-cluster")
    .secretArn("arn:aws:secretsmanager:us-east-1:123456789012:secret:my-secret")
    .database("mydb")
    .sql("SELECT * FROM users LIMIT 10")
    .build();

ExecuteStatementResponse response = client.executeStatement(request);
System.out.println(response.records());
```

JavaScript

```
const { RDSDataClient, ExecuteStatementCommand } = require("@aws-sdk/client-rds-
data");

// Create RDS Data API client with IPv6 dual-stack endpoint
const client = new RDSDataClient({
    endpoint: "https://rds-data.us-east-1.api.aws"
});
```

```
// Execute a SQL statement
const command = new ExecuteStatementCommand({
    resourceArn: "arn:aws:rds:us-east-1:123456789012:cluster:my-cluster",
    secretArn: "arn:aws:secretsmanager:us-east-1:123456789012:secret:my-secret",
    database: "mydb",
    sql: "SELECT * FROM users LIMIT 10"
});

const response = await client.send(command);
console.log(response.records);
```

AWS PrivateLink 搭配 IPv6 使用

您可以為支援 VPC 內 IPv6 連線的資料 API 建立 VPC 端點。如需為資料 API 建立 VPC 端點的詳細指示，請參閱建立 Amazon RDS 資料 API 的 Amazon VPC 端點 (AWS PrivateLink)。

為 IPv6 支援建立 VPC 端點時，請確保：

- VPC 和子網路已設定為支援 IPv6
- 安全群組允許在必要連接埠的 IPv6 流量 (HTTPS 通常是 443)
- 網路 ACL 已設定為允許 IPv6 流量

遷移考量事項

遷移至 IPv6 端點時，請考慮下列事項：

- 逐步遷移：您可以透過一次更新一個應用程式的端點 URL 來逐步遷移應用程式。
- 網路相容性：確保網路基礎設施在遷移之前支援 IPv6。
- 安全政策：視需要更新安全群組規則和網路 ACL，以允許 IPv6 流量。
- 監控：更新監控和記錄組態以處理 IPv6 地址。

> **Note**
>
> 資料庫連線地址：使用資料 API 的 IPv6 端點時，基礎資料庫連線和資料庫日誌仍會顯示 IPv4 地址。這是預期的行為，不會影響已啟用 IPv6 的應用程式功能。

針對 IPv6 連線進行疑難排解

如果您遇到 IPv6 連線問題，請檢查下列項目：

網路組態

確認網路支援 IPv6 且 IPv6 路由的設定都正確。

DNS 解析

確保 DNS 解析程式可以解析雙堆疊端點的 AAAA 記錄。

Security groups (安全群組)

更新安全群組規則，以允許在必要連接埠的 IPv6 流量 (HTTPS 通常是 443)。

用戶端程式庫

確認 HTTP 用戶端程式庫支援 IPv6 和雙堆疊連線。

Amazon RDS 資料 API 的限制

RDS 資料 API 具有下列限制：

- 您只能在資料庫叢集中的寫入器執行個體上執行資料 API 查詢。不過，寫入器執行個體可以同時接受寫入和讀取查詢。
- 使用 Aurora 全域資料庫，您就可以在主要和次要資料庫叢集啟用資料 API。不過，次要叢集在提升為主要叢集之前，都不具備寫入器執行個體。資料 API 需要存取寫入器執行個體以進行查詢處理，即使是讀取查詢也是如此。因此，傳送至次要叢集的讀取和寫入查詢會在缺少寫入器執行個體時失敗。一旦次要叢集受到提升並具有可用的寫入器執行個體，該資料庫執行個體上的資料 API 查詢就會成功。
- T 資料庫執行個體類別上不支援資料 API。
- 對於 Aurora Serverless v2 和佈建的資料庫叢集，RDS 資料 API 不支援某些資料類型。如需支援的類型清單，請參閱the section called “與 Aurora Serverless v1 相比具有 Aurora Serverless v2 的資料 API ”。
- 對於 Aurora PostgreSQL 第 14 版及更新版本的資料庫，資料 API 僅支援 `scram-sha-256` 來進行密碼加密。
- 回應大小限制為 1 MiB。若呼叫傳回的回應資料超過 1 MiB，系統就會終止呼叫。
- 對於 Aurora Serverless v1，每秒請求數上限為 1,000。對於所有其他支援的資料庫，沒有限制。

- 針對資料庫傳回結果集中的每個資料列，資料 API 的大小上限為每個資料列 64 KB。請確定結果集的每個列都是 64 KB 或更小。

使用 Aurora Serverless v1 叢集比較 Aurora Serverless v2 和佈建叢集的 Amazon RDS 資料 API 行為

Amazon RDS 資料 API 的最新增強功能可讓使用最新版 PostgreSQL 或 MySQL 引擎的叢集使用資料 API。這些叢集可設定為使用 Aurora Serverless v2 或佈建執行個體類別，例如 `db.r6g` 或 `db.r6i`。

下列各節說明 Aurora Serverless v2 和佈建資料庫叢集和 Aurora Serverless v1 資料庫叢集之間的 Amazon RDS 資料 API 差異。Aurora Serverless v1資料庫叢集使用 `serverless` 引擎模式。佈建的資料庫叢集使用 `provisioned` 引擎模式。Aurora Serverless v2 資料庫叢集也會使用 `provisioned` 引擎模式，且包含一或多個具有 `db.serverless` 執行個體類別的 Aurora Serverless v2 資料庫執行個體。

每秒請求數上限

Aurora Serverless v1

資料 API 每秒最多可提出 1,000 個請求。

Aurora Serverless v2

資料 API 每秒可以提出不限數量的請求。

在現有資料庫啟用或停用 Amazon RDS 資料 API

Aurora Serverless v1

- 使用 Amazon RDS API：使用 `ModifyCluster` 操作，然後指定 `EnableHttpEndpoint` 參數的 `True`或 `False` (如適用)。
- 搭配 AWS CLI：使用 `modify-db-cluster` 操作搭配 `--enable-http-endpoint` 或 `--no-enable-http-endpoint` 選項 (如適用)。

Aurora Serverless v2

- 搭配 Amazon RDS API：使用 `EnableHttpEndpoint` 和 `DisableHttpEndpoint` 操作。
- 搭配 AWS CLI：使用 `enable-http-endpoint` 和 `disable-http-endpoint` 操作。

CloudTrail 事件

Aurora Serverless v1

資料 API 呼叫中的事件是管理事件。根據預設，這些事件會自動包含在線索中。如需更多詳細資訊，請參閱 the section called “將資料 API 事件從 CloudTrail 線索 (僅 Aurora Serverless v1) 中排除”。

Aurora Serverless v2

資料 API 呼叫中的事件是資料事件。預設會在線索中自動排除這些事件。如需更多詳細資訊，請參閱 the section called “在 CloudTrail 線索中包含資料 API 事件”。

多陳述式支援

Aurora Serverless v1

- 對於 Aurora MySQL，不支援多陳述式。
- 對於 Aurora PostgreSQL，多陳述式只會傳回第一個查詢回應。

Aurora Serverless v2

不支援多陳述式。嘗試在單一 API 呼叫中執行多個陳述式會傳回 “`An error occurred (ValidationException) when calling the ExecuteStatement operation: Multistatements aren't supported.`”。若要執行多個陳述式，請分別進行 `ExecuteStatement` API 呼叫或使用 `BatchExecuteStatement` 進行批次處理。

下列範例顯示 API 呼叫所產生的錯誤訊息，此 API 呼叫會嘗試執行多陳述式。

```
aws rds-data execute-statement \
   --resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
   --secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
   --database "your_database" \
   --sql "SELECT * FROM your_table; Select * FROM next_table;

                                "An error occurred (ValidationException) when calling
the ExecuteStatement operation: Multistatements aren't supported.
```

下列範例會使用不同的 `ExecuteStatement` API 呼叫執行多個陳述式。

```
aws rds-data execute-statement \
```

```
    --resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
    --secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
    --database "your_database" \
    --sql "SELECT * FROM your_table;"

aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
    --secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
    --database "your_database" \
    --sql "SELECT * FROM next_table;"
```

相同交易 ID 的並行請求

Aurora Serverless v1

後續請求會等到目前的請求完成。如果等待期間過長，應用程式需要處理逾時錯誤。

Aurora Serverless v2

當資料 API 收到具有相同交易 ID 的多個請求時，其會立即傳回此錯誤：

`DatabaseErrorException: Transaction is still running a query`

此錯誤會在兩種情況下發生：

- 應用程式會使用相同的交易 ID 提出異步請求 (例如 JavaScript 承諾)。
- 具有該交易 ID 的先前請求仍在處理中。

下列範例顯示與 `promise.all()` 平行執行的所有請求。

```
const api_calls = [];
for (let i = 0; i < 10; i++) {
api_calls.push(
    client.send(
    new ExecuteStatementCommand({
        ...params,
        sql: `insert into table_name values (i);`,
        transactionId
    })
    )
);
}
```

```
await Promise.all(api_calls);
```

若要解決此錯誤，請等待目前的請求完成，再傳送另一個具有相同交易 ID 的請求，或移除交易 ID 以允許平行請求。

下列範例顯示的 API 呼叫會使用具有相同交易 ID 的循序執行。

```
 for (let i = 0; i < 10; i++) {
    await client.send(
    new ExecuteStatementCommand({
        ...params,
        sql: `insert into table_name values (i);`,
        transactionId
    })
    ).promise()
);
}
```

BatchExecuteStatement 行為

如需 `BatchExecuteStatement` 的詳細資訊，請參閱 BatchExecuteStatement。

Aurora Serverless v1

更新結果中產生的欄位物件包含插入的值。

Aurora Serverless v2

- 對於 Aurora MySQL，更新結果中產生的欄位物件包含插入的值。
- 對於 Aurora PostgreSQL，產生的欄位物件是空的。

ExecuteSQL 行為

如需 `ExecuteSQL` 的詳細資訊，請參閱 ExecuteSQL。

Aurora Serverless v1

`ExecuteSQL` 操作已棄用。

Aurora Serverless v2

不支援 ExecuteSQL 操作。

ExecuteStatement 行為

如需 ExecuteStatement 的詳細資訊，請參閱 ExecuteStatement。

Aurora Serverless v1

ExecuteStatement 參數支援擷取多維陣列欄和所有進階資料類型。

Aurora Serverless v2

ExecuteStatement 參數不支援多維陣列欄。其也不支援某些 PostgreSQL 資料類型 (包括幾何和貨幣類型)。當資料 API 遇到不支援的資料類型時，其會傳回此錯誤：UnsupportedResultException: The result contains the unsupported data type data_type。

若要解決此問題，請將不支援的資料類型轉換為 TEXT。下列範例會將不支援的資料類型轉換為 TEXT。

```
SELECT custom_type::TEXT FROM my_table;--
ORSELECT CAST(custom_type AS TEXT) FROM my_table;
```

如需每個 Aurora 資料庫引擎支援的資料類型清單，請參閱資料 API 操作參考。

結構描述參數行為

Aurora Serverless v1

不支援 Schema 參數。當您在 API 呼叫中包含 Schema 參數時，資料 API 會忽略此參數。

Aurora Serverless v2

Schema 參數已棄用。當您在 API 呼叫中包含 Schema 參數時，資料 API 會傳回此錯誤：ValidationException: The schema parameter isn't supported。下列範例顯示傳回 ValidationException 錯誤的資料 API 呼叫。

```
aws rds-data execute-statement \
--resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
--secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
```

```
--database "your_database" \
--schema "your_schema" \
--sql "SELECT * FROM your_table LIMIT 10"
```

若要解決此問題，請從 API 呼叫中移除 Schema 參數。

下列範例顯示已移除 Schema 參數的資料 API 呼叫。

```
aws rds-data execute-statement \
--resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
--secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
--database "your_database" \
--sql "SELECT * FROM your_table LIMIT 10"
```

授權 Amazon RDS 資料 API 的存取

使用者獲得授權後才能調用 Amazon RDS 資料 API (資料 API) 作業。您可以連接定義其權限的 AWS Identity and Access Management (IAM) 政策，授予使用者使用資料 API 的許可。如果您使用的是 IAM 角色，也可以將政策連接至角色。 AWS 受管政策 AmazonRDSDataFullAccess包含資料 API 的許可。

此AmazonRDSDataFullAccess政策也包含使用者從中取得秘密值的許可 AWS Secrets Manager。使用者需要使用 Secrets Manager 來儲存可在呼叫資料 API 時使用的秘密。使用秘密表示使用者不需要在呼叫資料 API 時包含其目標資源的資料庫登入資料。資料 API 會透明地呼叫 Secrets Manager，以允許 (或拒絕) 使用者對秘密的請求。如需設定秘密以搭配資料 API 使用的相關資訊，請參閱在 中存放資料庫登入資料 AWS Secrets Manager。

AmazonRDSDataFullAccess 政策提供資源的完整存取 (透過資料 API)。您可以透過定義自己的政策來指定資源的 Amazon Resource Name (ARN)，藉以縮小範圍。

例如，下列政策顯示使用者存取其 ARN 所識別之資料庫叢集之資料 API 的最低所需許可範例。該政策包括存取 Secrets Manager 和取得使用者資料庫執行個體授權所需的許可。

JSON

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

```
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*"
        },
        {
            "Sid": "RDSDataServiceAccess",
            "Effect": "Allow",
            "Action": [
                "rds-data:BatchExecuteStatement",
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:ExecuteStatement",
                "rds-data:RollbackTransaction"
            ],
            "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:prod"
        }
    ]
}
```

我們建議您針對政策陳述式中的「資源」元素使用特定 ARN (如範例所示)，而不是萬用字元 (*)。

使用標籤型授權

RDS 資料 API (資料 API) 和 Secrets Manager 都支援標籤型授權。標籤是用來標示資源 (例如 RDS 叢集) 的索引鍵值配對，並具有額外的字串值，例如：

- `environment:production`
- `environment:development`

您可以將標籤套用至資源，以進行成本配置、作業支援、存取控制和許多其他原因。(如果您的資源上還沒有標籤，而您想要套用標籤，您可至標記 Amazon RDS 資源中進一步了解。) 您可以在政策陳述式中使用標籤來限制對以這些標籤標示之 RDS 叢集的存取。舉例來說，Aurora 資料庫叢集可能具有用來將其環境標識為生產或開發的標籤。

下列範例顯示如何在政策陳述式中使用標籤。此陳述式要求叢集和資料 API 要求中傳遞的秘密都有一個 `environment:production` 標籤。

以下是套用政策的方式：當使用者使用資料 API 進行呼叫時，會將要求傳送至服務。資料 API 首先驗證在請求中傳遞的叢集 ARN 是否標記了 `environment:production`。然後，它會呼叫 Secrets Manager 擷取請求中的使用者秘密值。Secrets Manager 還會驗證使用者的秘密是否被標記為 `environment:production`。如果有，資料 API 接著會使用使用者資料庫密碼的擷取值。最後，如果這也是正確的，則會為使用者成功地呼叫資料 API 請求。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                 "secretsmanager:GetSecretValue"
               ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*",
            "Condition": {
                    "StringEquals": {
                        "aws:ResourceTag/environment": [
                                        "production"
                                       ]
                     }
             }
        },
        {
            "Sid": "RDSDataServiceAccess",
            "Effect": "Allow",
            "Action": [
                  "rds-data:*"
               ],
            "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:*",
            "Condition": {
                    "StringEquals": {
                        "aws:ResourceTag/environment": [
                                        "production"
                                       ]
                     }
             }
         }
     ]
```

```
}
```

此範例顯示適用於資料 API 和 Secrets Manager 之 `rds-data` 和 `secretsmanager` 的個別動作。但是，您可以合併動作並以許多不同的方式定義標籤條件，以支援您的特定使用案例。如需詳細資訊，請參閱針對 Secrets Manager 使用以身分為基礎的政策 (IAM 政策)。

在政策的「條件」元素中，您可以從以下選項中選擇標籤鍵：

- `aws:TagKeys`
- `aws:ResourceTag/${TagKey}`

若要進一步了解資源標籤和如何使用 `aws:TagKeys`，請參閱使用 AWS 資源標籤控制對資源的存取。

> **Note**
>
> 資料 API 和 都會 AWS Secrets Manager 授權使用者。如果您沒有政策中定義之所有動作的許可，您會收到 `AccessDeniedException` 錯誤訊息。

在 中存放資料庫登入資料 AWS Secrets Manager

當您呼叫 Amazon RDS 資料 API (資料 API) 時，您可以在 Secrets Manager 中使用秘密來傳遞Aurora資料庫叢集的憑證。若要這樣傳遞登入資料，請指定秘密的名稱或秘密的 Amazon Resource Name (ARN)。

在秘密中存放資料庫叢集登入資料

1. 使用 Secrets Manager 建立秘密，其中含有 Aurora 資料庫叢集的登入資料。

   如需說明，請參閱《AWS Secrets Manager 使用者指南》中的建立資料庫機密。

2. 使用 Secrets Manager 主控台來檢視您建立之秘密的詳細資訊，或執行 `aws secretsmanager describe-secret` AWS CLI 命令。

   記下秘密的名稱和 ARN。在呼叫資料 API 時即可派上用場。

如需使用 Secrets Manager 的詳細資訊，請參閱 AWS Secrets Manager 使用者指南。

若要了解 Amazon Aurora 如何管理身分識別和存取管理，請參閱 Amazon Aurora 如何搭配 IAM使用。

如需有關建立 IAM 政策的詳細資訊，請參閱IAM 使用者指南中的建立 IAM 政策。如需有關將 IAM 政策新增給使用者的資訊，請參閱 IAM 使用者指南中的新增和移除 IAM 身分許可。

啟用 Amazon RDS 資料 API

若要使用 Amazon RDS 資料 API (資料 API)，請為 Aurora 資料庫叢集將其啟用。您可以在建立或修改資料庫叢集時啟用資料 API。

> **Note**
>
> 叢集是否可使用資料 API，取決於您的 Aurora 版本、資料庫引擎和 AWS 區域。對於較舊的 Aurora 版本，資料 API 僅適用於 Aurora Serverless v1 叢集。對於較新的 Aurora 版本，資料 API 適用於同時使用佈建和 Aurora Serverless v2 執行個體的叢集。若要檢查叢集是否可以使用資料 API，請參閱 RDS 資料 API 的支援區域和 Aurora 資料庫引擎。

主題

- 建立資料庫時啟用 RDS 資料 API
- 在現有資料庫啟用 RDS 資料 API

建立資料庫時啟用 RDS 資料 API

當您建立的是支援 RDS 資料 API (資料 API) 的資料庫時，您可以啟用此功能。下列程序說明如何在使用 AWS 管理主控台 AWS CLI、 或 RDS API 時執行此操作。

主控台

如下列螢幕擷取畫面所示，若要在建立資料庫叢集時啟用資料 API，請選取建立資料庫頁面的連線區段中啟用 RDS 資料 API 核取方塊。



如需如何建立可使用 RDS 資料 API 的 Aurora 資料庫叢集的說明，請參閱下文：

- 對於 Aurora Serverless v2 和佈建的叢集：建立 Amazon Aurora 資料庫叢集
- 對於 Aurora Serverless v1：建立 Aurora Serverless v1 資料庫叢集

AWS CLI

若要在建立 Aurora 資料庫叢集時啟用資料 API，請使用 `--enable-http-endpoint`選項執行 create-db-cluster AWS CLI 命令。

下列範例會建立已啟用資料 API 的 Aurora PostgreSQL 資料庫叢集。

對於 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
    --db-cluster-identifier my_pg_cluster \
    --engine aurora-postgresql \
    --enable-http-endpoint
```

在 Windows 中：

```
aws rds create-db-cluster ^
    --db-cluster-identifier my_pg_cluster ^
    --engine aurora-postgresql ^
    --enable-http-endpoint
```

RDS API

若要在建立 Aurora 資料庫叢集時啟用資料 API，請使用 CreateDBCluster 操作，並將 `EnableHttpEndpoint` 參數的值設為 `true`。

在現有資料庫啟用 RDS 資料 API

您可以修改支援 RDS 資料 API (資料 API) 的資料庫叢集，即可啟用或停用此功能。

主題

- 啟用或停用資料 API (Aurora Serverless v2 和已佈建)
- 啟用或停用資料 API (僅限 Aurora Serverless v1)

啟用或停用資料 API (Aurora Serverless v2 和已佈建)

使用下列程序啟用或停用 Aurora Serverless v2 和佈建資料庫的資料 API。若要啟用或停用在 Aurora Serverless v1 資料庫的資料 API，請使用 the section called “啟用或停用資料 API (僅限 Aurora Serverless v1)” 中的程序。

主控台

您可以使用支援此功能之資料庫叢集的 RDS 主控台，來啟用或停用資料 API。若要這樣做，請開啟您要啟用或停用資料 API 之資料庫的叢集詳細資訊頁面，然後在連線與安全索引標籤，前往 RDS 資料 API 區段。本節顯示資料 API 的狀態，並允許您將其啟用或停用。

下列螢幕擷取畫面顯示未啟用的 RDS 資料 API。



AWS CLI

若要啟用或停用現有資料庫上的資料 API，請執行 enable-http-endpoint 或 disable-http-endpoint AWS CLI 命令，並指定資料庫叢集的 ARN。

下列範例會啟用資料 API。

對於 Linux、macOS 或 Unix：

```
aws rds enable-http-endpoint \
    --resource-arn cluster_arn
```

在 Windows 中：

```
aws rds enable-http-endpoint ^
    --resource-arn cluster_arn
```

RDS API

若要啟用或停用現有資料庫上的資料 API，請使用 EnableHttpEndpoint 和 DisableHttpEndpoint 操作。

啟用或停用資料 API (僅限 Aurora Serverless v1)

使用下列程序來啟用或停用現有 Aurora Serverless v1 資料庫的資料 API。若要在 Aurora Serverless v2 和佈建的資料庫啟用或停用資料 API，請使用 the section called “啟用或停用資料 API” 中的程序。

主控台

修改 Aurora Serverless v1 資料庫叢集時，在 RDS 主控台的連線 區段中啟用資料 API。

下列螢幕擷取畫面顯示修改 Aurora DB 資料庫叢集時啟用的資料 API。



如需如何修改 Aurora Serverless v1 資料庫叢集的指示，請參閱修改 Aurora Serverless v1 資料庫叢集。

AWS CLI

若要啟用或停用資料 API，`--no-enable-http-endpoint`請視需要使用 `--enable-http-endpoint`或 執行 modify-db-cluster AWS CLI 命令。

下列範例會在 `sample-cluster` 啟用資料 API。

對於 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
```

```
    --enable-http-endpoint
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --enable-http-endpoint
```

RDS API

若要啟用資料 API，請使用 ModifyDBCluster 操作，以及視需要將 `EnableHttpEndpoint` 的值設為 `true` 或 `false`。

建立 Amazon RDS 資料 API 的 Amazon VPC 端點 (AWS PrivateLink)

Amazon VPC 可讓您在虛擬私有雲端 (VPC) 中啟動 AWS 資源，例如 Aurora 資料庫叢集和應用程式。AWS PrivateLink 在 Amazon 網路上提供具有高安全性VPCs AWS 和服務之間的私有連線。您可以使用 AWS PrivateLink建立 Amazon VPC 端點，讓您根據 Amazon VPC 連線到不同帳戶和 VPCs 的服務。如需 AWS PrivateLink的相關資訊，請參閱《Amazon Virtual Private Cloud 使用者指南》中的 VPC 端點服務 (AWS PrivateLink)。

您可以使用 Amazon VPC 端點呼叫 RDS 資料 API (資料 API)。使用 Amazon VPC 端點可在 Amazon VPC 中的應用程式與 AWS 網路中的資料 API 之間保留流量，而無需使用公有 IP 地址。Amazon VPC 端點能協助您符合與限制公有網際網路連線相關的合規和法律需求。舉例來說，若使用 Amazon VPC 端點，就能讓在 Amazon EC2 執行個體上執行之應用程式和資料 API 的流量只在包含兩者的 VPC 中傳送。

建立 Amazon VPC 端點之後就能開始使用，而不需要在應用程式中進行任何程式碼或組態變更。

建立資料 API 的 Amazon VPC 端點

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/vpc/：// 開啟 Amazon VPC 主控台。
2. 選擇 Endpoints (端點)，然後選擇 Create Endpoint (建立端點)。
3. 在 Create Endpoint (建立端點) 頁面上，針對 Service category (服務類別) 選擇 AWS services (服務)。針對 Service Name (服務名稱)，選擇 rds-data。



4. 針對 VPC，選擇要在其中建立端點的 VPC。

   選擇包含進行資料 API 呼叫之應用程式的 VPC。

5. 針對子網路，選擇執行您應用程式之 AWS 服務所使用的每個可用區域 (AZ) 的子網路。

   若要建立 Amazon VPC 端點，請指定可存取端點的私有 IP 地址範圍。若要執行此作業，請選擇每個可用區域的子網路。這麼做會將 VPC 端點限制為每個可用區域專屬的私有 IP 地址範圍，並且也會在每個可用區域中建立 Amazon VPC 端點。

6. 針對 Enable DNS name (啟用 DNS 名稱)，選取 Enable for this endpoint (為此端點啟用)。

私有 DNS 會將標準資料 API DNS 主機名稱 (`https://rds-data.region.amazonaws.com`) 解析為與您 Amazon VPC 端點專用 DNS 主機名稱相關的私有 IP 地址。因此，您可以使用 AWS CLI AWS SDKs存取資料 API VPC 端點，而不需要進行任何程式碼或組態變更來更新資料 API 的端點 URL。

7. 針對 Security group (安全群組)，選擇要與 Amazon VPC 端點建立關聯的安全群組。

   選擇允許存取執行您應用程式之 AWS 服務的安全群組。舉例來說，若有 Amazon EC2 執行個體在執行您的應用程式，請選擇要允許存取 Amazon EC2 執行個體的安全群組。安全群組能讓您控制 VPC 中，資源流向 Amazon VPC 端點的流量。

8. 針對 Policy (政策)，請選擇 Full Access (完整存取) 讓 Amazon VPC 中的所有人都能透過此端點存取資料 API。或者選擇 Custom (自訂)，來指定限制存取的政策。

   如果您選擇 Custom (自訂)，請在政策建立工具中輸入政策。

9. 選擇 Create endpoint (建立端點)。

建立端點之後，請選擇 中的連結 AWS 管理主控台 以檢視端點詳細資訊。



端點 Details (詳細資訊) 標籤會顯示建立 Amazon VPC 端點時產生的 DNS 主機名稱。



您可以使用標準端點 (`rds-data.region.amazonaws.com`) 或其中一個 VPC 專用端點，來在 Amazon VPC 中呼叫資料 API。標準資料 API 端點會自動路由至 Amazon VPC 端點。因為私有 DNS 主機名稱在 Amazon VPC 端點建立時已啟用，所以會發生此路由。

當您在資料 API 呼叫中使用 Amazon VPC 端點時，應用程式與資料 API 間的所有流量都會維持在包含兩者的 Amazon VPC 中。您可以使用 Amazon VPC 端點進行任何類型的資料 API 呼叫。如需呼叫資料 API 的詳細資訊，請參閱呼叫 Amazon RDS 資料 API。

呼叫 Amazon RDS 資料 API

在 Aurora 資料庫叢集啟用 Amazon RDS 資料 API (資料 API) 後，您可以使用資料 API 或 AWS CLI，在 Aurora 資料庫叢集執行 SQL 陳述式。資料 API 支援 AWS 開發套件支援的程式設計語言。如需詳細資訊，請參閱在 AWS 上建置的工具。

主題

- Amazon RDS 資料 API 操作參考
- 使用 AWS CLI 呼叫 Amazon RDS 資料 API
- 透過 Python 應用程式呼叫 Amazon RDS 資料 API
- 透過 Java 應用程式呼叫 Amazon RDS 資料 API
- 控制資料 API 逾時行為

Amazon RDS 資料 API 操作參考

Amazon RDS 資料 API 會提供以下操作來執行 SQL 陳述式。

資料 API 操作	AWS CLI 命令	Description
ExecuteStatement	aws rds-data execute-statement	在資料庫上執行 SQL 陳述式。
BatchExecuteStatement	aws rds-data batch-execute-statement	透過對資料陣列執行批次 SQL 陳述式來進行大量更新和插入操作。您可以使用參數集陣列來執行資料處理語言 (DML) 陳述式。批次 SQL 陳述式可針對個別插入和更新陳述式提供明顯的效能改善。

您可以使用任一操作來執行個別 SQL 陳述式或執行交易。針對交易，資料 API 會提供以下操作。

資料 API 操作	AWS CLI 命令	Description
BeginTransaction	aws rds-data begin-transaction	開始 SQL 交易。
CommitTransaction	aws rds-data commit-transaction	結束 SQL 交易並遞交變更。
RollbackTransaction	aws rds-data rollback-transaction	執行交易轉返。

執行 SQL 陳述式和支援交易的操作具有下列常見的資料 API 參數和 AWS CLI 選項。有些操作支援其他參數或選項。

資料 API 操作參數	AWS CLI 命令選項	必要	描述
resourceArn	--resource-arn	是	Aurora 資料庫叢集的 Amazon Resource Name (ARN)。叢集必須與叫用資料 API AWS 帳戶 的 IAM 角色或使用者位於相同的 中。若要存取不同帳戶中的叢集，請擔任該帳戶中的角色。

資料 API 操作參數	AWS CLI 命令選項	必要	描述
secretArn	--secret-arn	是	用來允許存取資料庫叢集之秘密的名稱或 ARN。

RDS 資料 API 支援 Aurora MySQL 的下列資料類型：

- TINYINT(1), BOOLEAN, BOOL
- TINYINT
- SMALLINT [SIGNED | UNSIGNED]
- MEDIUMINT [SIGNED | UNSIGNED]
- INT [SIGNED | UNSIGNED]
- BIGINT [SIGNED | UNSIGNED]
- FLOAT
- DOUBLE
- VARCHAR, CHAR, TEXT, ENUM
- VARBINARY, BINARY, BLOB
- DATE, TIME, DATETIME, TIMESTAMP
- DECIMAL
- JSON
- BIT, BIT(N)

RDS 資料 API 支援下列 Aurora PostgreSQL 純量類型：

- BOOL
- BYTEA
- DATE
- CIDR
- DECIMAL, NUMERIC
- ENUM
- FLOAT8, DOUBLE PRECISION

- INET
- INT, INT4, SERIAL
- INT2, SMALLINT, SMALLSERIAL
- INT8, BIGINT, BIGSERIAL
- JSONB, JSON
- REAL, FLOAT
- TEXT, CHAR(N), VARCHAR, NAME
- TIME
- TIMESTAMP
- UUID
- VECTOR

RDS 資料 API 支援下列 Aurora PostgreSQL 陣列類型：

- BOOL[], BIT[]
- DATE[]
- DECIMAL[], NUMERIC[]
- FLOAT8[], DOUBLE PRECISION[]
- INT[], INT4[]
- INT2[]
- INT8[], BIGINT[]
- JSON[]
- REAL[], FLOAT[]
- TEXT[], CHAR(N)[], VARCHAR[], NAME[]
- TIME[]
- TIMESTAMP[]
- UUID[]

您可以在對 ExecuteStatement和 進行資料 API 呼叫BatchExecuteStatement，以及在執行 AWS CLI 命令execute-statement和 時使用參數batch-execute-statement。若要使用參數，

請在 SqlParameter 資料類型中指定名稱/值對。您使用 Field 資料類型來指定值。下表會將 Java Database Connectivity (JDBC) 資料類型映射至您在資料 API 呼叫中指定的資料類型。

JDBC 資料類型	資料 API 資料類型
INTEGER, TINYINT, SMALLINT, BIGINT	LONG – 、 或 STRING
FLOAT, REAL, DOUBLE	DOUBLE
DECIMAL	STRING
BOOLEAN, BIT	BOOLEAN
BLOB, BINARY, LONGVARBINARY, VARBINARY	BLOB
CLOB	STRING
其他類型 (包含與日期和時間相關的類型)	STRING

> **Note**
>
> 您可以在 LONG 資料 API 呼叫中，針對資料庫傳回的 STRING 值指定 LONG 或資料類型。我們建議您這麼做，以避免失去極大數字的精確度，您在使用 JavaScript 時可能會發生這種情況。

某些類型 (例如 DECIMAL 和 TIME) 需要提示，以便資料 API 以正確的類型將 String 值傳遞給資料庫。若要使用提示，請在 typeHint 資料類型中包含 SqlParameter 的值。typeHint 可能的值如下：

- DATE – 對應的 String 參數值以 DATE 類型的物件傳送至資料庫。接受的格式為 YYYY-MM-DD。
- DECIMAL – 對應的 String 參數值以 DECIMAL 類型的物件傳送至資料庫。
- JSON – 對應的 String 參數值以 JSON 類型的物件傳送至資料庫。
- TIME – 對應的 String 參數值以 TIME 類型的物件傳送至資料庫。接受的格式為 HH:MM:SS[.FFF]。
- TIMESTAMP – 對應的 String 參數值以 TIMESTAMP 類型的物件傳送至資料庫。接受的格式為 YYYY-MM-DD HH:MM:SS[.FFF]。

- `UUID` – 對應的 `String` 參數值以 `UUID` 類型的物件傳送至資料庫。

> **Note**
>
> 目前，資料 API 不支援通用唯一識別符 (UUID) 的陣列。

> **Note**
>
> 對於 Amazon Aurora PostgreSQL，資料 API 始終以 UTC 時區返回 Aurora PostgreSQL 資料類型 `TIMESTAMPTZ`。

使用 AWS CLI 呼叫 Amazon RDS 資料 API

您可以使用 AWS CLI 呼叫 RDS 資料 API(資料 API)。

下列範例會使用 AWS CLI 來呼叫資料 API。如需詳細資訊，請參閱資料 API 的 AWS CLI 參考。

在每個範例中，將資料庫叢集的 Amazon Resource Name (ARN) 取代為 Aurora 資料庫叢集的 ARN。同時也將秘密 ARN 取代為 Secrets Manager 中允許存取資料庫叢集的秘密 ARN。

> **Note**
>
> AWS CLI 可以將回應設為 JSON 格式。

主題

- 開始 SQL 交易
- 執行 SQL 陳述式
- 透過資料陣列來執行批次 SQL 陳述式
- 遞交 SQL 交易
- 復原 SQL 交易

開始 SQL 交易

您可以使用 `aws rds-data begin-transaction` CLI 命令來開始 SQL 交易。此呼叫會傳回交易識別符。

> **Important**
>
> 在資料 API 內，如果三分鐘內沒有使用交易 ID 的任何呼叫，交易就會逾時。如果交易在遞交前就逾時，則資料 API 會自動將其復原。
> 交易中的 MySQL 資料定義語言 (DDL) 陳述式會導致隱含遞交。我們建議您使用 `execute-statement` 選項在個別的 `--continue-after-timeout` 命令中執行每個 MySQL DDL 陳述式。

除了常用選項之外，還將指定 `--database` 選項，其提供資料庫的名稱。

例如，以下 CLI 命令會開始 SQL 交易。

對於 Linux、macOS 或 Unix：

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret"
```

在 Windows 中：

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret"
```

以下是回應的範例。

```
{
    "transactionId": "ABC1234567890xyz"
}
```

執行 SQL 陳述式

您可以使用 `aws rds-data execute-statement` CLI 命令來執行 SQL 陳述式。

您可以透過使用 `--transaction-id` 選項指定交易識別符，來在交易中執行 SQL 陳述式。您可以使用 `aws rds-data begin-transaction` CLI 命令來開始交易。您可以使用 `aws rds-data commit-transaction` CLI 命令來結束和遞交交易。

> **Important**
>
> 如果您沒有指定 `--transaction-id` 選項，系統就會自動遞交呼叫造成的變更。

除了常用的選項，請指定以下選項：

- `--sql` (必要) – 在資料庫叢集上執行的 SQL 陳述式。
- `--transaction-id` (選用) – 使用 `begin-transaction` CLI 命令開始的交易識別符。指定您要在其中包含 SQL 陳述式的交易 ID。
- `--parameters` (選用) – SQL 陳述式的參數。
- `--include-result-metadata | --no-include-result-metadata` (選用) – 此值會指出是否在結果中包含中繼資料。預設值為 `--no-include-result-metadata`。
- `--database` (選用) – 資料庫的名稱。

  當您在先前請求中執行 `--sql "use database_name;"` 之後執行 SQL 陳述式時，`--database` 選項可能無法運作。建議您使用 `--database` 選項，而不是執行 `--sql "use database_name;"` 陳述式。
- `--continue-after-timeout | --no-continue-after-timeout` (選用)：此值會指出在呼叫超過 45 秒的資料 API 逾時間隔後是否繼續執行陳述式。預設值為 `--no-continue-after-timeout`。

  對於資料定義語言 (DDL) 陳述式，我們建議在呼叫逾時後繼續執行陳述式，來避免錯誤和資料結構毀損的可能性。
- `--format-records-as "JSON"|"NONE"` – 選用值，指定是否將結果集格式化為 JSON 字串。預設值為 `"NONE"`。如需處理 JSON 結果集的使用情況資訊，請參閱以 JSON 格式處理 Amazon RDS 資料 API 查詢結果。

資料庫叢集會傳回呼叫的回應。

> **Note**
>
> 回應大小限制為 1 MiB。若呼叫傳回的回應資料超過 1 MiB，系統就會終止呼叫。
> 對於 Aurora Serverless v1，每秒請求數上限為 1,000。對於所有其他支援的資料庫，沒有限制。

例如，以下 CLI 命令會執行單一 SQL 陳述式並省略結果中的中繼資料 (預設值)。

對於 Linux、macOS 或 Unix：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "select * from mytable"
```

在 Windows 中：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "select * from mytable"
```

以下是回應的範例。

```
{
    "numberOfRecordsUpdated": 0,
    "records": [
        [
            {
                "longValue": 1
            },
            {
                "stringValue": "ValueOne"
            }
        ],
        [
            {
                "longValue": 2
            },
            {
                "stringValue": "ValueTwo"
            }
        ],
        [
            {
                "longValue": 3
            },
```

```
            {
                "stringValue": "ValueThree"
            }
        ]
    ]
}
```

以下 CLI 命令會透過指定 `--transaction-id` 選項來在交易中執行單一 SQL 陳述式。

對於 Linux、macOS 或 Unix：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

在 Windows 中：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

以下是回應的範例。

```
{
    "numberOfRecordsUpdated": 1
}
```

以下 CLI 命令會執行含參數的單一 SQL 陳述式。

對於 Linux、macOS 或 Unix：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" --parameters "[{\"name\": \"id\",
 \"value\": {\"longValue\": 1}},{\"name\": \"val\", \"value\": {\"stringValue\":
 \"value1\"}}]"
```

在 Windows 中：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" --parameters "[{\"name\": \"id\",
 \"value\": {\"longValue\": 1}},{\"name\": \"val\", \"value\": {\"stringValue\":
 \"value1\"}}]"
```

以下是回應的範例。

```
{
    "numberOfRecordsUpdated": 1
}
```

以下 CLI 命令會執行資料定義語言 (DDL) SQL 陳述式。DDL 陳述式會將 `job` 欄重新命名為欄 `role`。

> **Important**
>
> 對於 DDL 陳述式，我們建議在呼叫逾時後繼續執行陳述式。當 DDL 陳述式在完成執行前而終止時，可能會發生錯誤且資料結構可能毀損。若要在呼叫超過 45 秒的 RDS 資料 API 逾時間隔後繼續執行陳述式，請指定 `--continue-after-timeout` 選項。

對於 Linux、macOS 或 Unix：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "alter table mytable change column job role varchar(100)" --continue-after-
timeout
```

在 Windows 中：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
```

```
--sql "alter table mytable change column job role varchar(100)" --continue-after-
timeout
```

以下是回應的範例。

```
{
    "generatedFields": [],
    "numberOfRecordsUpdated": 0
}
```

> **Note**
>
> Aurora PostgreSQL 不支援 generatedFields 資料。若要取得所產生欄位的值，請使用 RETURNING 子句。如需詳細資訊，請參閱 PostgreSQL 中的從修改後的資料列傳回資料。

透過資料陣列來執行批次 SQL 陳述式

您可以透過使用 aws rds-data batch-execute-statement CLI 命令，透過資料陣列來執行批次 SQL 陳述式。您可以使用此命令來執行大量匯入或更新操作。

您可以透過使用 --transaction-id 選項指定交易識別符，來在交易中執行 SQL 陳述式。您可以透過使用 aws rds-data begin-transaction CLI 命令來開始交易。您可以透過使用 aws rds-data commit-transaction CLI 命令來結束和遞交交易。

> **Important**
>
> 如果您沒有指定 --transaction-id 選項，系統就會自動遞交呼叫造成的變更。

除了常用的選項，請指定以下選項：

- --sql (必要) – 在資料庫叢集上執行的 SQL 陳述式。

  > **Tip**
  >
  > 對於 MySQL 相容的陳述句，請不要在 --sql 參數結尾包含分號。結尾分號可能會導致語法錯誤。

- `--transaction-id` (選用) – 使用 `begin-transaction` CLI 命令開始的交易識別符。指定您要在其中包含 SQL 陳述式的交易 ID。
- `--parameter-set` (選用) – 批次操作的參數組。
- `--database` (選用) – 資料庫的名稱。

資料庫叢集會傳回呼叫的回應。

> **Note**
>
> 參數組數目並無固定上限。不過，透過資料 API 提交的 HTTP 請求大小上限為 4 MiB。若請求超過此限制，資料 API 會傳回錯誤，不會處理請求。此 4 MiB 限制包括 HTTP 標頭的大小和請求中的 JSON 標記法。因此，您可包含的參數集數目取決於因素組合，例如 SQL 陳述式的大小和每個參數集的大小。
>
> 回應大小限制為 1 MiB。若呼叫傳回的回應資料超過 1 MiB，系統就會終止呼叫。
>
> 對於 Aurora Serverless v1，每秒請求數上限為 1,000。對於所有其他支援的資料庫，沒有限制。

例如，以下 CLI 命令會透過含參數組的資料陣列來執行批次 SQL 陳述式。

對於 Linux、macOS 或 Unix：

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" \
--parameter-sets "[[{\"name\": \"id\", \"value\": {\"longValue\": 1}},{\"name\":
 \"val\", \"value\": {\"stringValue\": \"ValueOne\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 2}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueTwo\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 3}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueThree\"}}]]"
```

在 Windows 中：

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
```

```
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" ^
--parameter-sets "[[{\"name\": \"id\", \"value\": {\"longValue\": 1}},{\"name\":
 \"val\", \"value\": {\"stringValue\": \"ValueOne\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 2}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueTwo\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 3}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueThree\"}}]]"
```

> **Note**
>
> 請勿在 `--parameter-sets` 選項中包含分行符號。

遞交 SQL 交易

您可以使用 `aws rds-data commit-transaction` CLI 命令，來結束您透過 `aws rds-data begin-transaction` 開始的 SQL 交易和遞交變更。

除了常用的選項，請指定以下選項：

- `--transaction-id` (必要) – 使用 `begin-transaction` CLI 命令開始的交易識別符。指定您要結束和遞交的交易 ID。

例如，以下 CLI 命令會結束 SQL 交易和遞交變更。

對於 Linux、macOS 或 Unix：

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

在 Windows 中：

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

以下是回應的範例。

```
{
    "transactionStatus": "Transaction Committed"
}
```

復原 SQL 交易

您可以使用 `aws rds-data rollback-transaction` CLI 命令，來復原您透過 `aws rds-data begin-transaction` 開始的 SQL 交易。復原交易會取消其變更。

> **Important**
>
> 如果交易 ID 已過期，系統會自動復原此交易。在這個情況下，指定此過期交易 ID 的 `aws rds-data rollback-transaction` 命令會傳回錯誤。

除了常用的選項，請指定以下選項：

- `--transaction-id` (必要) – 使用 `begin-transaction` CLI 命令開始的交易識別符。指定您要復原的交易 ID。

例如，以下 AWS CLI 命令會復原 SQL 交易。

對於 Linux、macOS 或 Unix：

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

在 Windows 中：

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

以下是回應的範例。

```
{
    "transactionStatus": "Rollback Complete"
    }
```

透過 Python 應用程式呼叫 Amazon RDS 資料 API

您可以透過 Python 應用程式呼叫 Amazon RDS 資料 API (資料 API)。

以下範例會使用適用於 Python 的 AWS 開發套件 (Boto)。如需 Boto 的更多詳細資訊，請參閱適用於 Python 的 AWS 開發套件 (Boto 3) 文件。

在每個範例中，將資料庫叢集的 Amazon Resource Name (ARN) 取代為 Aurora 資料庫叢集的 ARN。同時也將秘密 ARN 取代為 Secrets Manager 中允許存取資料庫叢集的秘密 ARN。

主題

- 執行 SQL 查詢
- 執行 DML SQL 陳述式
- 執行 SQL 交易

執行 SQL 查詢

您可以執行 SELECT 陳述式並使用 Python 應用程式擷取結果。

以下範例會執行 SQL 查詢。

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

response1 = rdsData.execute_statement(
            resourceArn = cluster_arn,
            secretArn = secret_arn,
            database = 'mydb',
            sql = 'select * from employees limit 3')

print (response1['records'])
[
    [
```

```
        {
            'longValue': 1
        },
        {
            'stringValue': 'ROSALEZ'
        },
        {
            'stringValue': 'ALEJANDRO'
        },
        {
            'stringValue': '2016-02-15 04:34:33.0'
        }
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'DOE'
        },
        {
            'stringValue': 'JANE'
        },
        {
            'stringValue': '2014-05-09 04:34:33.0'
        }
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'STILES'
        },
        {
            'stringValue': 'JOHN'
        },
        {
            'stringValue': '2017-09-20 04:34:33.0'
        }
    ]
]
```

執行 DML SQL 陳述式

您可以執行資料處理語言 (DML) 陳述式來插入、更新或刪除資料庫中的資料。您也可以在 DML 陳述式中使用參數。

> **Important**
>
> 如果某個呼叫不包含 `transactionID` 參數而不屬於某個交易，則系統會自動遞交該呼叫造成的變更。

以下範例會執行插入 SQL 陳述式和使用參數。

```
import boto3

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

rdsData = boto3.client('rds-data')


param1 = {'name':'firstname', 'value':{'stringValue': 'JACKSON'}}
param2 = {'name':'lastname', 'value':{'stringValue': 'MATEO'}}
paramSet = [param1, param2]

response2 = rdsData.execute_statement(resourceArn=cluster_arn,
                                      secretArn=secret_arn,
                                      database='mydb',
                                      sql='insert into employees(first_name, last_name)
 VALUES(:firstname, :lastname)',
                                      parameters = paramSet)

print (response2["numberOfRecordsUpdated"])
```

執行 SQL 交易

您可以開始 SQL 交易、執行一或多個 SQL 陳述式，接著使用 Python 應用程式遞交變更。

> **Important**
>
> 如果三分鐘內沒有使用交易 ID 的任何呼叫，交易就會逾時。如果交易在遞交前就逾時，則系統會自動將其復原。

如果您沒有指定交易 ID，系統就會自動遞交呼叫造成的變更。

以下範例執行的 SQL 交易會在資料表中插入資料列。

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

tr = rdsData.begin_transaction(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     database = 'mydb')

response3 = rdsData.execute_statement(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     database = 'mydb',
     sql = 'insert into employees(first_name, last_name) values('XIULAN', 'WANG')',
     transactionId = tr['transactionId'])

cr = rdsData.commit_transaction(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     transactionId = tr['transactionId'])

cr['transactionStatus']
'Transaction Committed'

response3['numberOfRecordsUpdated']
1
```

Note

如果您執行的是資料定義語言 (DDL) 陳述式，我們建議在呼叫逾時後繼續執行陳述式。當 DDL 陳述式在完成執行前而終止時，可能會發生錯誤且資料結構可能毀損。若要在呼叫超過 45 秒的 RDS 資料 API 逾時間隔後繼續執行陳述式，請將 `continueAfterTimeout` 參數設為 `true`。

透過 Java 應用程式呼叫 Amazon RDS 資料 API

您可以透過 Java 應用程式呼叫 Amazon RDS 資料 API (資料 API)。

以下範例會使用適用於 Java 的 AWS 開發套件。如需詳細資訊，請參閱 適用於 Java 的 AWS SDK 開發人員指南。

在每個範例中，將資料庫叢集的 Amazon Resource Name (ARN) 取代為 Aurora 資料庫叢集的 ARN。同時也將秘密 ARN 取代為 Secrets Manager 中允許存取資料庫叢集的秘密 ARN。

主題

- 執行 SQL 查詢
- 執行 SQL 交易
- 執行批次 SQL 操作

執行 SQL 查詢

您可以執行 SELECT 陳述式並使用 Java 應用程式擷取結果。

以下範例會執行 SQL 查詢。

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementResult;
import com.amazonaws.services.rdsdata.model.Field;

import java.util.List;

public class FetchResultsExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
    AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    ExecuteStatementRequest request = new ExecuteStatementRequest()
```

```
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb")
            .withSql("select * from mytable");

    ExecuteStatementResult result = rdsData.executeStatement(request);

    for (List<Field> fields: result.getRecords()) {
      String stringValue = fields.get(0).getStringValue();
      long numberValue = fields.get(1).getLongValue();

      System.out.println(String.format("Fetched row: string = %s, number = %d",
 stringValue, numberValue));
    }
  }
}
```

執行 SQL 交易

您可以開始 SQL 交易、執行一或多個 SQL 陳述式，接著使用 Java 應用程式遞交變更。

> **Important**
>
> 如果三分鐘內沒有使用交易 ID 的任何呼叫，交易就會逾時。如果交易在遞交前就逾時，則系統會自動將其復原。
> 如果您沒有指定交易 ID，系統就會自動遞交呼叫造成的變更。

以下範例會執行 SQL 交易。

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BeginTransactionRequest;
import com.amazonaws.services.rdsdata.model.BeginTransactionResult;
import com.amazonaws.services.rdsdata.model.CommitTransactionRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;

public class TransactionExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
```

```
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
    AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    BeginTransactionRequest beginTransactionRequest = new BeginTransactionRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb");
    BeginTransactionResult beginTransactionResult =
 rdsData.beginTransaction(beginTransactionRequest);
    String transactionId = beginTransactionResult.getTransactionId();

    ExecuteStatementRequest executeStatementRequest = new ExecuteStatementRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table VALUES ('hello world!')");
    rdsData.executeStatement(executeStatementRequest);

    CommitTransactionRequest commitTransactionRequest = new CommitTransactionRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN);
    rdsData.commitTransaction(commitTransactionRequest);
  }
}
```

> **Note**
>
> 如果您執行的是資料定義語言 (DDL) 陳述式，我們建議在呼叫逾時後繼續執行陳述式。當 DDL 陳述式在完成執行前而終止時，可能會發生錯誤且資料結構可能毀損。若要在呼叫超過 45 秒的 RDS 資料 API 逾時間隔後繼續執行陳述式，請將 `continueAfterTimeout` 參數設為 `true`。

執行批次 SQL 操作

您可以使用 Java 應用程式，透過資料陣列來執行大量插入和更新操作。您可以使用參數集陣列來執行 DML 陳述式。

> **Important**
>
> 如果您沒有指定交易 ID，系統就會自動遞交呼叫造成的變更。

以下範例會執行批次插入操作。

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BatchExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.Field;
import com.amazonaws.services.rdsdata.model.SqlParameter;

import java.util.Arrays;

public class BatchExecuteExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
      AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    BatchExecuteStatementRequest request = new BatchExecuteStatementRequest()
            .withDatabase("test")
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table2 VALUES (:string, :number)")
            .withParameterSets(Arrays.asList(
                    Arrays.asList(
                            new SqlParameter().withName("string").withValue(new
 Field().withStringValue("Hello")),
                            new SqlParameter().withName("number").withValue(new
 Field().withLongValue(1L))
                    ),
                    Arrays.asList(
                            new SqlParameter().withName("string").withValue(new
 Field().withStringValue("World")),
                            new SqlParameter().withName("number").withValue(new
 Field().withLongValue(2L))
```

```
                    )
            ));

    rdsData.batchExecuteStatement(request);
  }
}
```

控制資料 API 逾時行為

對資料 API 的所有呼叫都是同步。假設您執行的是執行 SQL 陳述式的資料 API 操作，例如 INSERT 或 CREATE TABLE。如果資料 API 呼叫成功傳回，則 SQL 處理會在呼叫傳回時完成。

根據預設，如果操作未在 45 秒內完成處理，則資料 API 會取消操作並傳回逾時錯誤。在這種情況下，資料不會插入，資料表也不會建立，以此類推。

您可以使用資料 API，來執行無法在 45 秒內完成的長時間執行操作。如果您預期對大型資料表執行大量 INSERT 或 DDL 操作之類的操作需要超過 45 秒，您可以指定 ExecuteStatement 操作的 continueAfterTimeout 參數。應用程式仍會收到逾時錯誤。不過，操作會繼續執行，不會取消。如需範例，請參閱「執行 SQL 交易」。

如果程式設計語言 AWS 開發套件有自己的 API 呼叫或 HTTP 通訊端連線逾時期間，請確定所有這類逾時期間都超過 45 秒。對於某些 SDK，逾時期間預設為少於 45 秒。我們建議將任何 SDK 特定或用戶端特定逾時期間設定為至少一分鐘。這樣做可避免應用程式在資料 API 操作仍然成功完成時收到逾時錯誤的可能性。如此一來，您就可以確定是否要重試操作。

例如，假設 SDK 將逾時錯誤傳回至應用程式，但資料 API 操作仍在資料 API 逾時間隔內完成。在這種情況下，重試操作可能會插入重複的資料，否則會產生不正確的結果。開發套件可能會自動重試操作，導致資料有誤，而不需要來自應用程式的任何動作。

逾時間隔對於 Java 2 開發套件特別重要。在該開發套件中，API 呼叫逾時和 HTTP 通訊端逾時預設為 30 秒。以下是將這些逾時設為更高值的範例：

```
public RdsDataClient createRdsDataClient() {
    return RdsDataClient.builder()
        .region(Region.US_EAST_1) // Change this to your desired Region
        .overrideConfiguration(createOverrideConfiguration())
        .httpClientBuilder(createHttpClientBuilder())
        .credentialsProvider(defaultCredentialsProvider()) // Change this to your
 desired credentials provider
        .build();
```

```
}

private static ClientOverrideConfiguration createOverrideConfiguration() {
    return ClientOverrideConfiguration.builder()
        .apiCallTimeout(Duration.ofSeconds(60))
        .build();
}

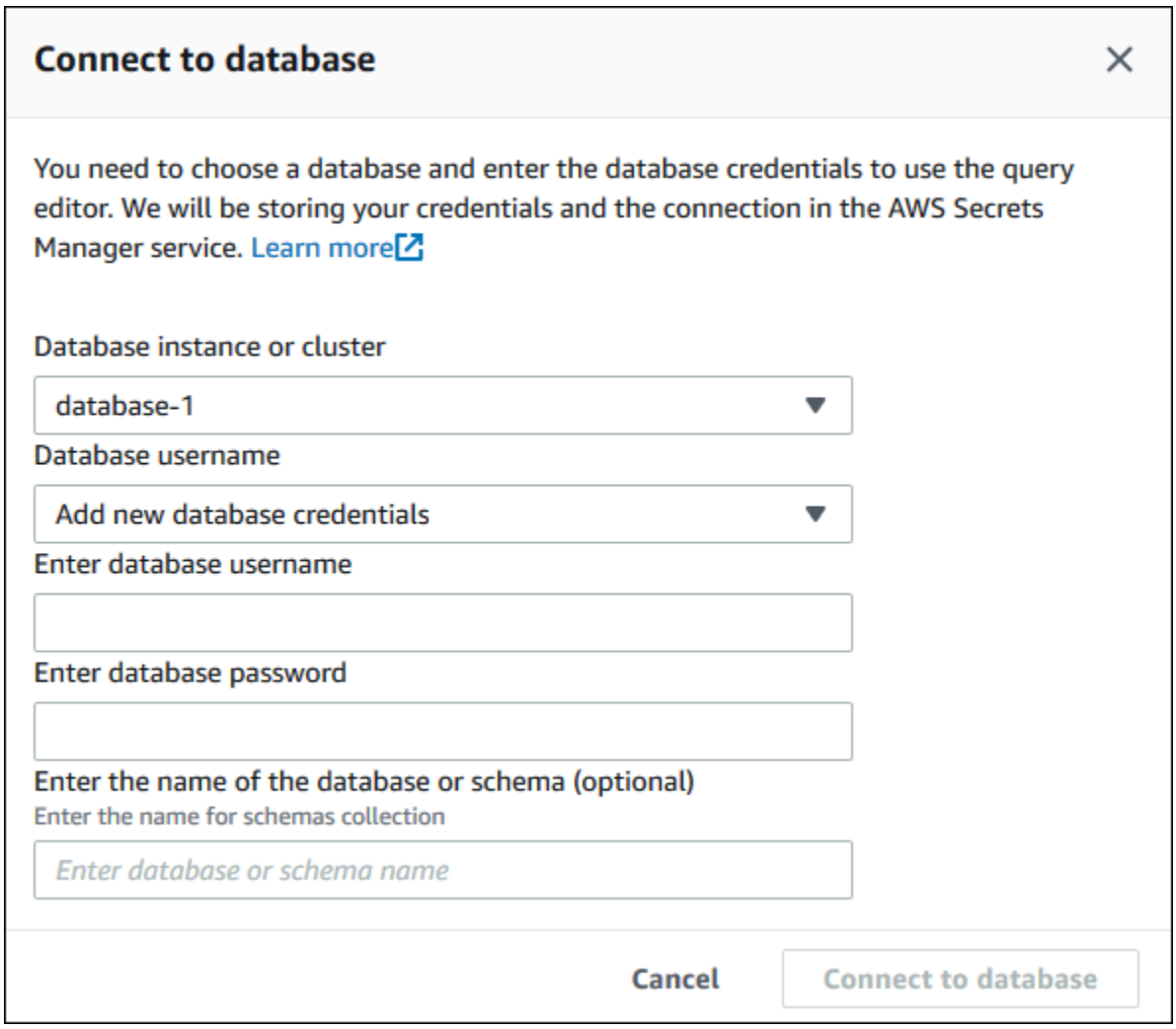
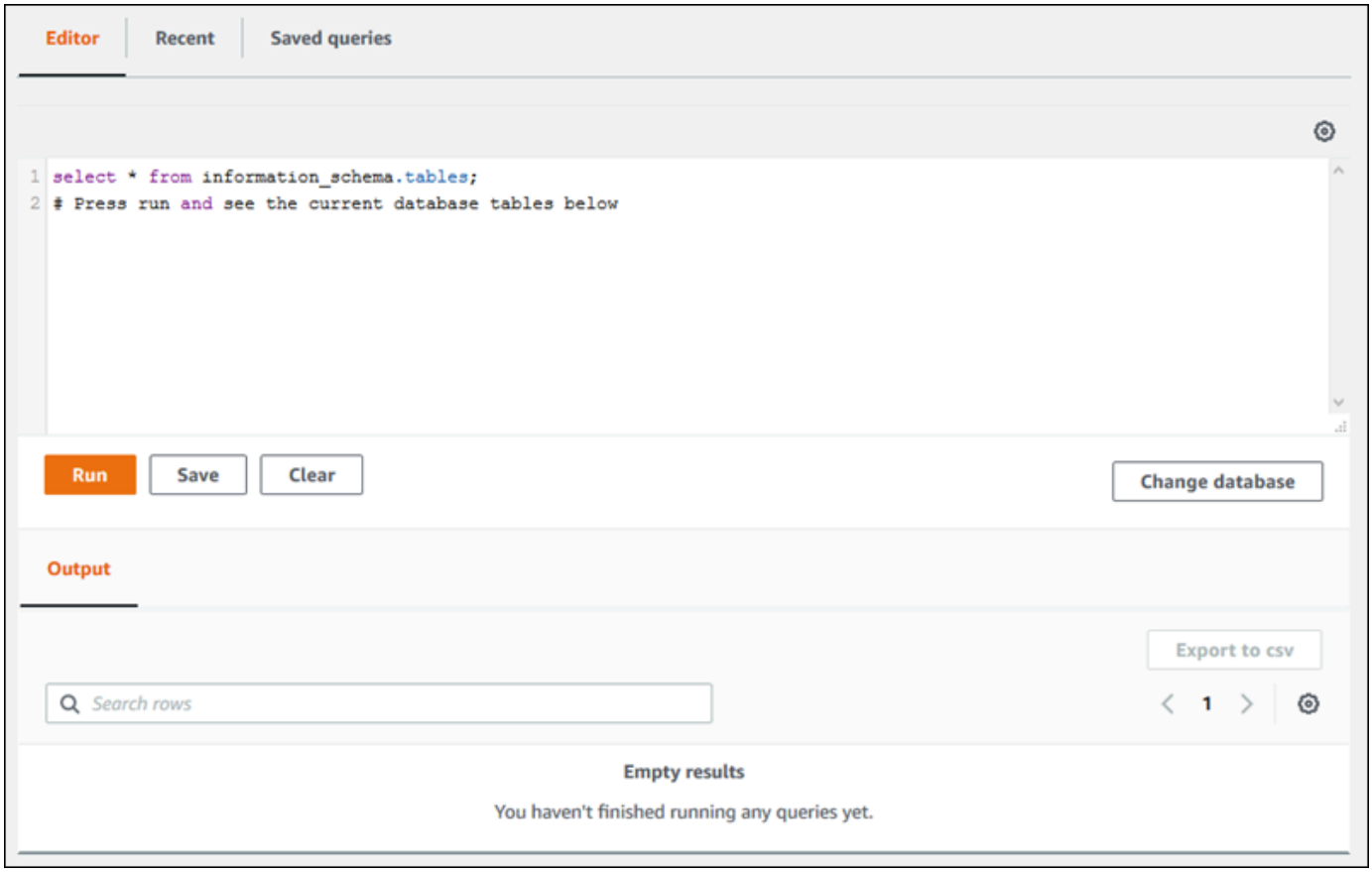
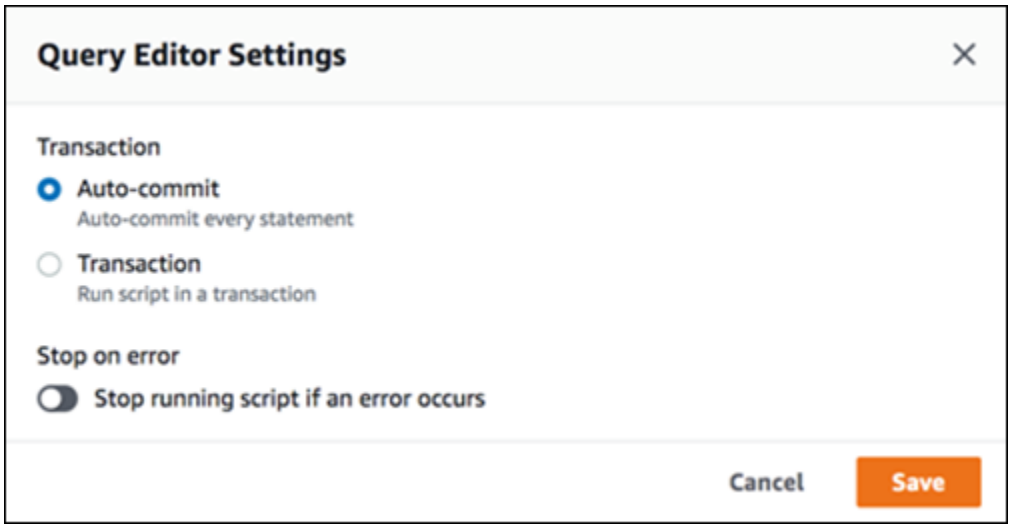
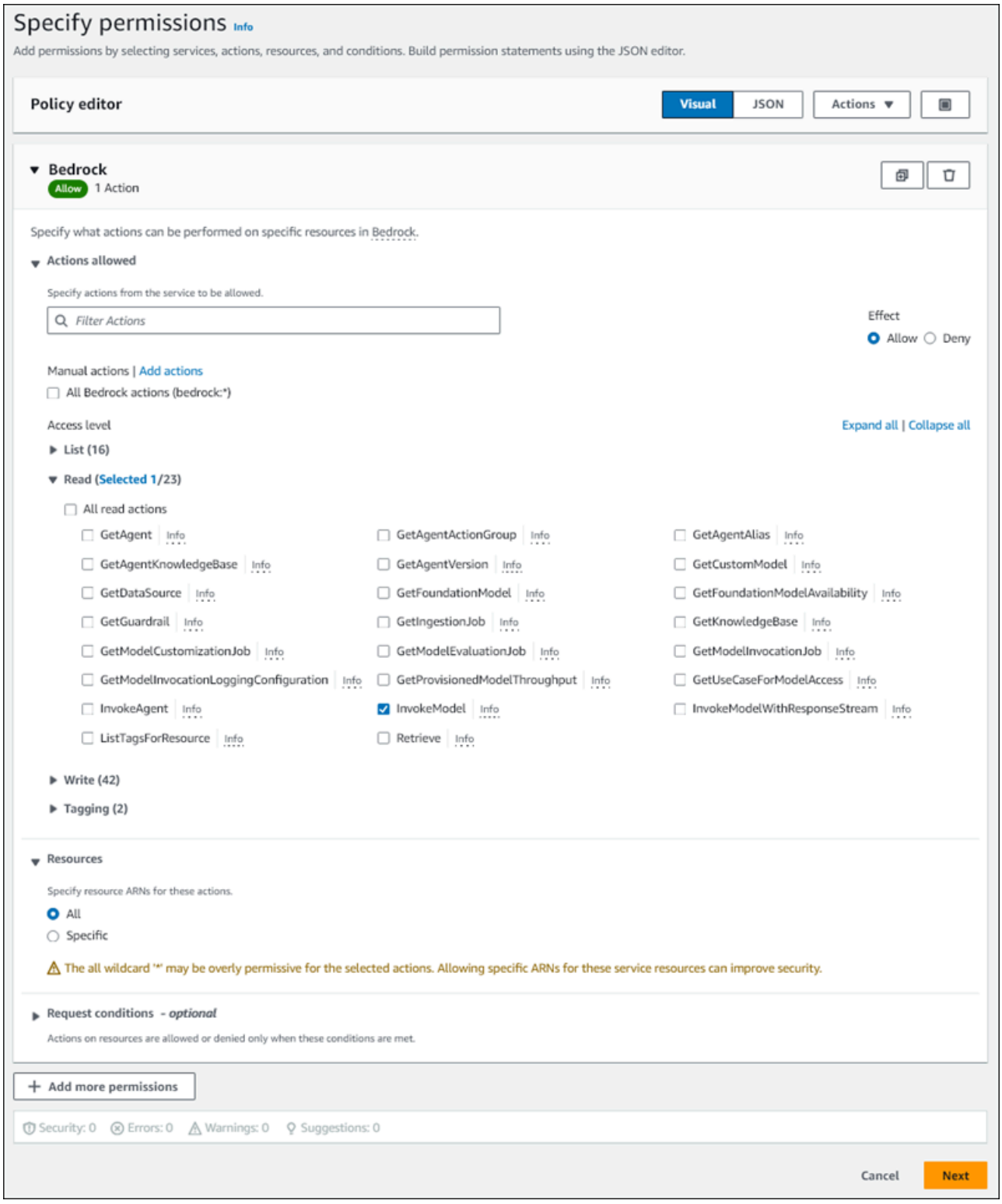
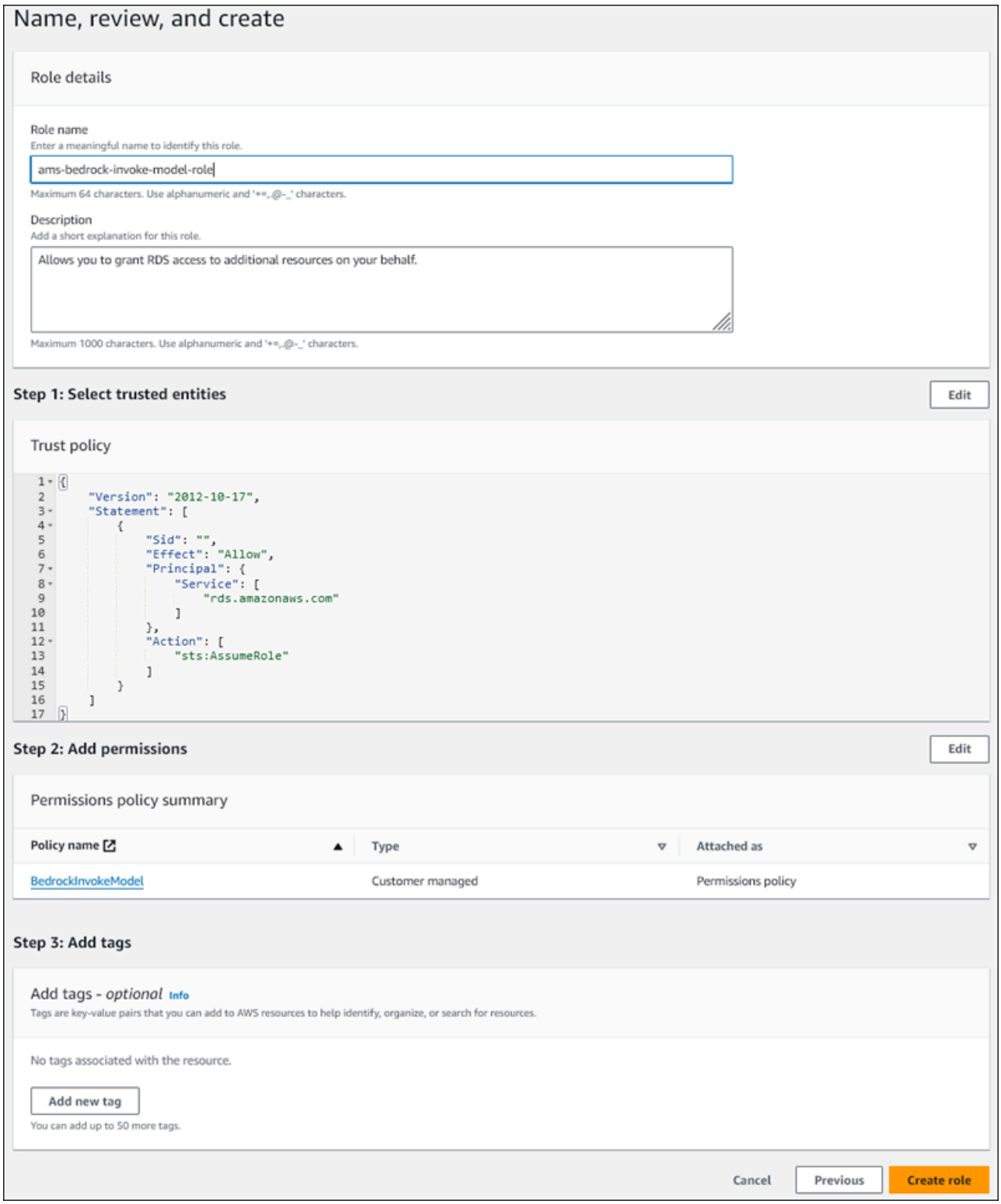
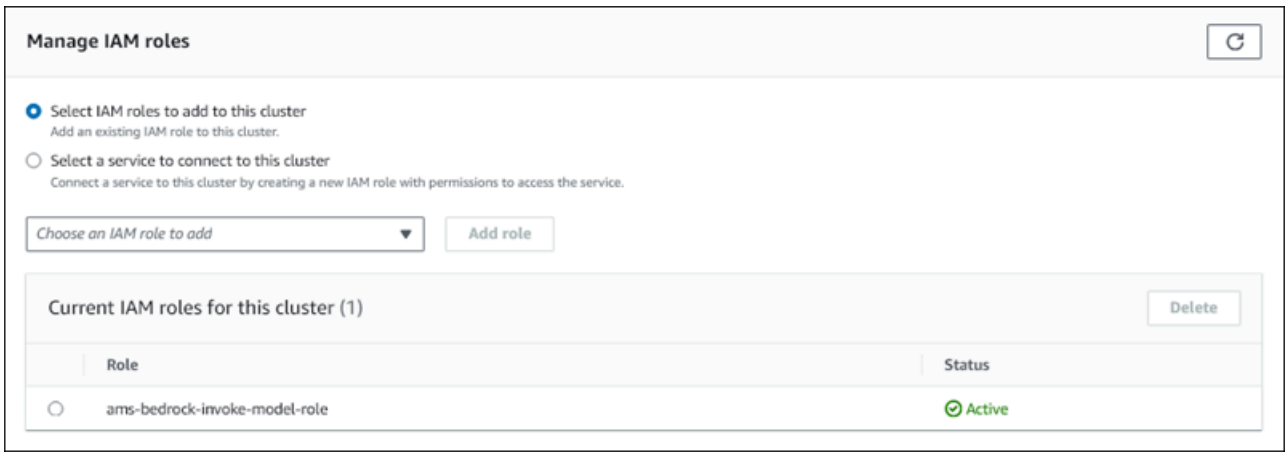
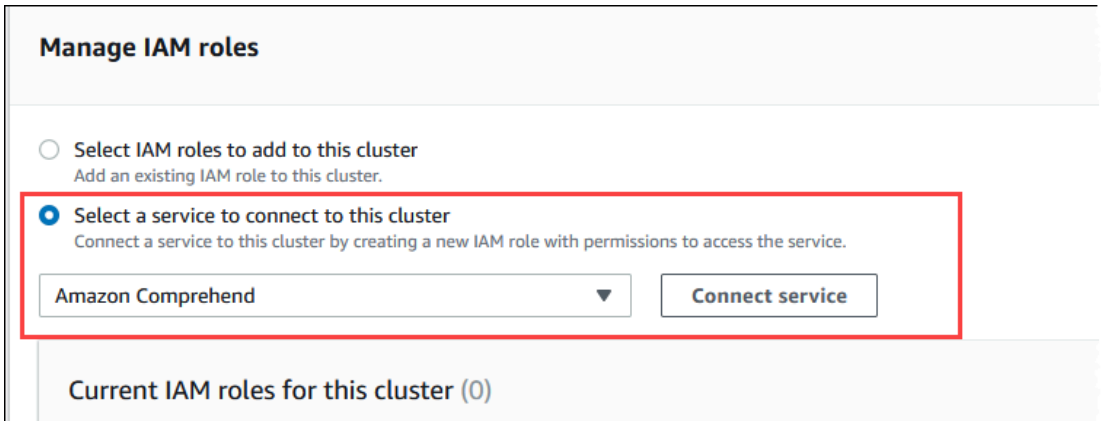
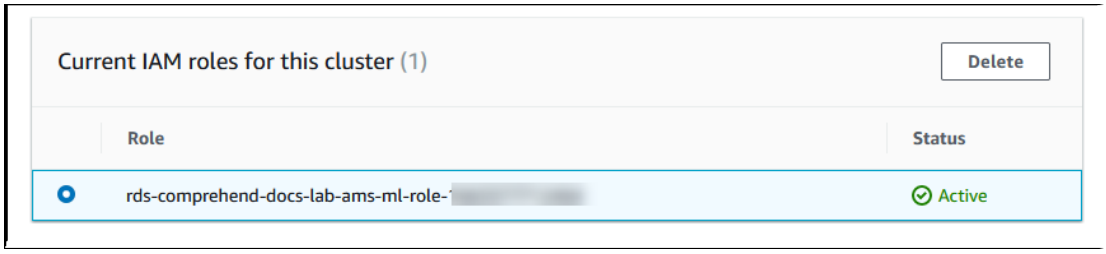
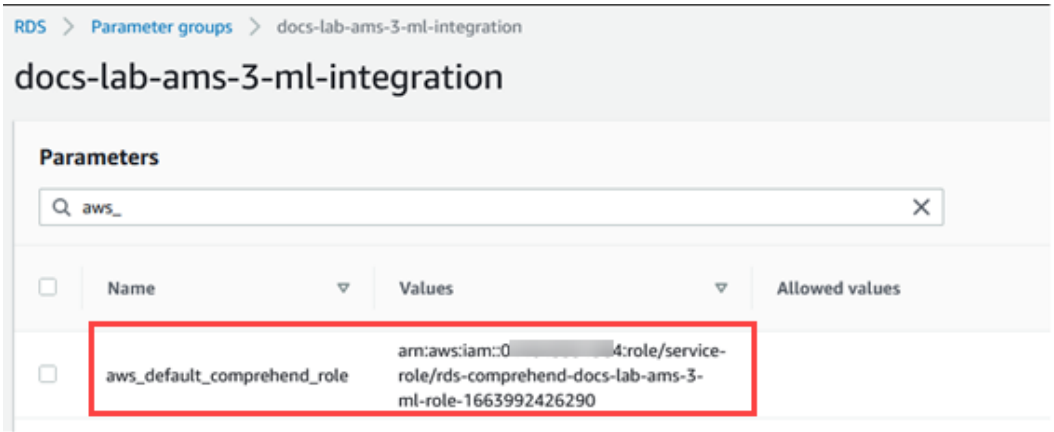
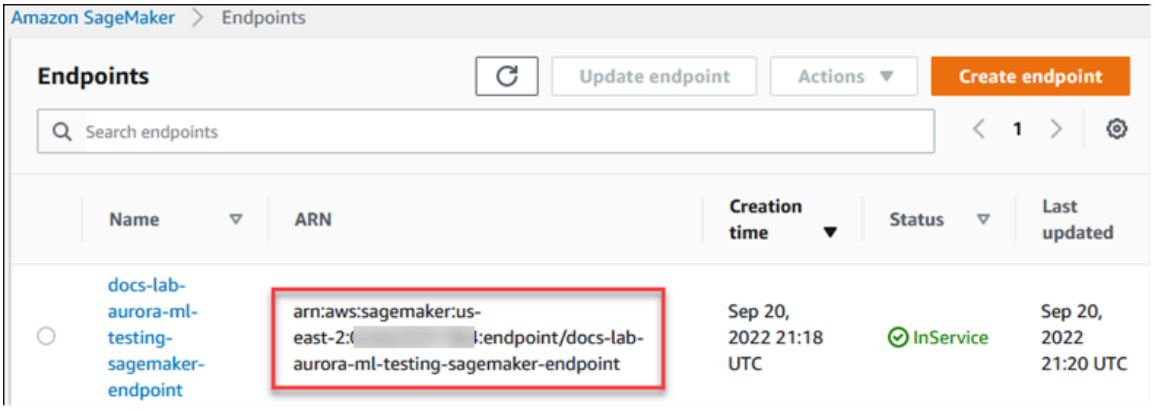
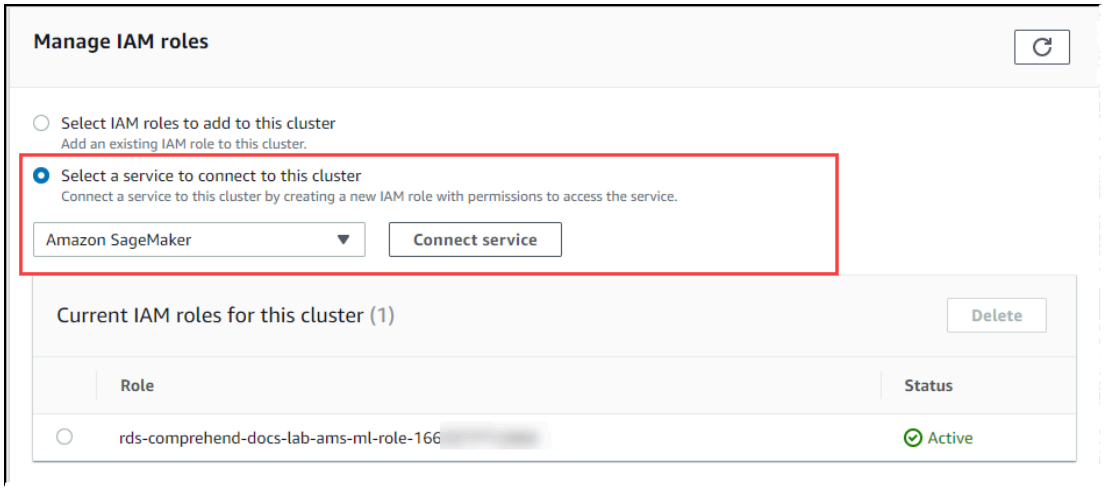
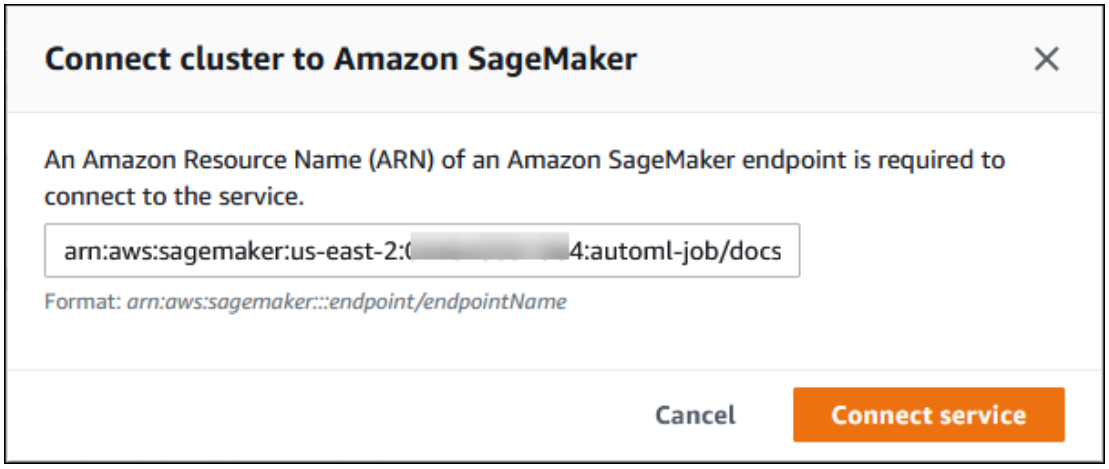
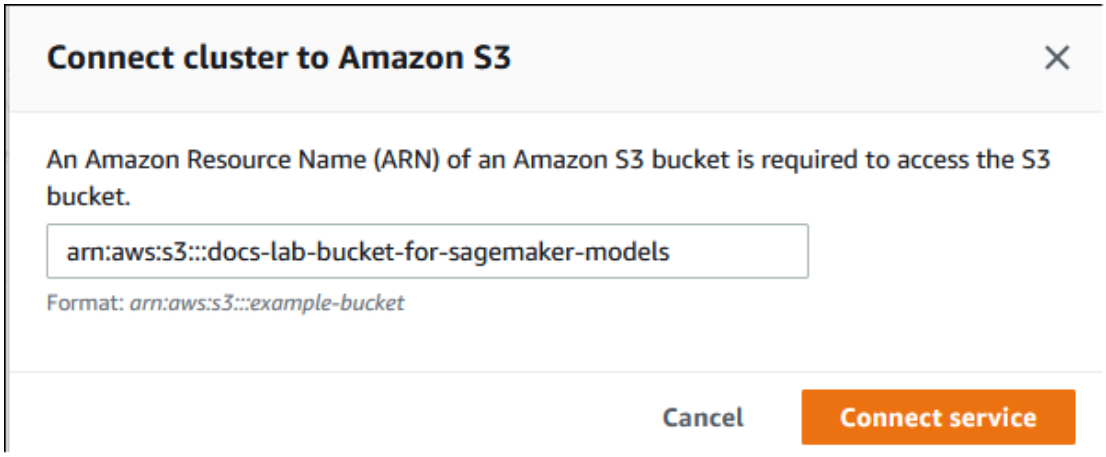
private HttpClientBuilder createHttpClientBuilder() {
    return ApacheHttpClient.builder() // Change this to your desired HttpClient
        .socketTimeout(Duration.ofSeconds(60));
}
```

以下是使用異步資料用戶端的同等範例：

```
public static RdsDataAsyncClient createRdsDataAsyncClient() {
    return RdsDataAsyncClient.builder()
        .region(Region.US_EAST_1) // Change this to your desired Region
        .overrideConfiguration(createOverrideConfiguration())
        .credentialsProvider(defaultCredentialsProvider())  // Change this to your
 desired credentials provider
        .build();
}

private static ClientOverrideConfiguration createOverrideConfiguration() {
    return ClientOverrideConfiguration.builder()
        .apiCallAttemptTimeout(Duration.ofSeconds(60))
        .build();
}

private HttpClientBuilder createHttpClientBuilder() {
    return NettyNioAsyncHttpClient.builder() // Change this to your desired
 AsyncHttpClient
        .readTimeout(Duration.ofSeconds(60));
}
```

使用適用於 RDS 資料 API 的 Java 用戶端程式庫

您可以下載與使用適用於 RDS 資料 API (資料 API) 的 Java 用戶端程式庫。此 Java 用戶端程式庫提供另一種使用資料 API 的方式。您可以使用此程式庫，將用戶端類別映射至資料 API 的要求和回應。此映射支援可以與某些特定的 Java 類型輕鬆整合，例如 `Date`、`Time` 和 `BigDecimal`。

下載適用於資料 API 的 Java 用戶端程式庫

資料 API Java 用戶端程式庫已在 GitHub 開放原始碼，並且可在以下位置取得：

https://github.com/awslabs/rds-data-api-client-library-java

您可以從原始檔案手動建置程式庫，但最佳實務是使用 Apache Maven 相依性管理來取用程式庫。將以下相依性新增到您的 Maven POM 檔案。

對於與 AWS SDK 2.x 相容的 2.x 版，請使用下列項目：

```
<dependency>
   <groupId>software.amazon.rdsdata</groupId>
   <artifactId>rds-data-api-client-library-java</artifactId>
   <version>2.0.0</version>
</dependency>
```

對於與 AWS SDK 1.x 相容的 1.x 版，請使用下列項目：

```
<dependency>
    <groupId>software.amazon.rdsdata</groupId>
    <artifactId>rds-data-api-client-library-java</artifactId>
    <version>1.0.8</version>
</dependency>
```

Java 用戶端程式庫範例

接下來，您可以找到一些關於使用資料 API Java 用戶端程式庫的常見範例。這些範例假設您有一個資料表 `accounts`，而資料表有兩欄：`accountId` 和 `name`。您還有下列資料傳輸物件 (DTO)。

```
public class Account {
    int accountId;
    String name;
    // getters and setters omitted
}
```

用戶端程式可讓您以輸入參數傳遞 DTO。下列範例顯示客戶 DTO 如何映射至輸入參數集。

```
var account1 = new Account(1, "John");
var account2 = new Account(2, "Mary");
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
```

```
        .withParamSets(account1, account2)
        .execute();
```

在某些情況下，以輸入參數處理簡單值比較輕鬆。作法是採用下列語法。

```
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
        .withParameter("accountId", 3)
        .withParameter("name", "Zhang")
        .execute();
```

以下是另一個以輸入參數處理簡單值的範例。

```
client.forSql("INSERT INTO accounts(accountId, name) VALUES(?, ?)", 4, "Carlos")
        .execute();
```

用戶端程式庫可以在結果傳回時自動映射至 DTO。下列範例顯示結果如何映射至 DTO。

```
List<Account> result = client.forSql("SELECT * FROM accounts")
          .execute()
          .mapToList(Account.class);

Account result = client.forSql("SELECT * FROM accounts WHERE account_id = 1")
          .execute()
          .mapToSingle(Account.class);
```

在許多情況下，資料庫結果集只包含單一值。為了簡化擷取這些結果，用戶端程式庫提供以下 API：

```
int numberOfAccounts = client.forSql("SELECT COUNT(*) FROM accounts")
          .execute()
          .singleValue(Integer.class);
```

> **Note**
>
> `mapToList` 函數會將 SQL 結果集轉換為使用者定義的物件清單。我們並不支援在 Java 用戶端程式庫的 `ExecuteStatement` 呼叫中使用 `.withFormatRecordsAs(RecordsFormatType.JSON)` 陳述式，因為其具有相同目的。如需詳細資訊，請參閱以 JSON 格式處理 Amazon RDS 資料 API 查詢結果。

以 JSON 格式處理 Amazon RDS 資料 API 查詢結果

當您呼叫 ExecuteStatement 操作，可以選擇以 JSON 格式的字串形式傳回查詢結果。這樣，您就可以使用程式設計語言的 JSON 剖析功能來解讀和重新格式化結果集。這樣做有助於避免編寫額外的程式碼來對結果集執行迴圈並解讀每個欄值。

若要以 JSON 格式請求結果集，請傳遞選用 formatRecordsAs 參數並搭配 JSON 值。JSON 格式的結果集會在 ExecuteStatementResponse 結構的 formattedRecords 欄位中傳回。

BatchExecuteStatement 動作不會傳回結果集。因此，JSON 選項不適用於該動作。

若要自訂 JSON 雜湊結構中的鍵，請在結果集中定義欄別名。若要這麼做，請在 SQL 查詢的欄清單中使用 AS 子句。

您可以使用 JSON 功能讓結果集更易於閱讀，並將其內容映射到語言特定的架構。由於 ASCII 編碼的結果集的量大於預設表示形式，您可以為傳回大量列或大型欄值的查詢 (這些查詢消耗的記憶體超過應用程式可用的記憶體) 選擇預設表示法。

主題

- 以 JSON 格式擷取查詢結果
- 資料類型映射
- 故障診斷
- 範例

以 JSON 格式擷取查詢結果

若要以 JSON 字串形式接收結果集，請在 ExecuteStatement 呼叫中加入 .withFormatRecordsAs(RecordsFormatType.JSON)。傳回值會以 formattedRecords 欄位中的 JSON 字串傳回。在本案例中，columnMetadata 為 null。欄標籤是表示每列物件的鍵。結果集中的每一列都會重複這些欄名。欄值為帶引號的字串、數字值或表示true、false, 或 null 的特殊值。欄中繼資料 (如長度限制以及數字和字串的精確類型) 不會保留在 JSON 回應中。

如果省略 .withFormatRecordsAs( ) 呼叫或指定 NONE 的參數，則結果集將使用 Records 和 columnMetadata 欄位以二進位格式傳回。

資料類型映射

結果集中的 SQL 值映射至一組較小的 JSON 類型。這些值在 JSON 中表示為字串、數字和一些特殊的常數，例如 `true`、`false`，和 `null`。您可以將這些值轉換為應用程式中的變數，根據您的程式設計語言使用強式或弱式輸入。

JDBC 資料類型	JSON 資料類型
INTEGER, TINYINT, SMALLINT, BIGINT	預設為數字。如果 LongReturnType 選項設定為 STRING 則為字串。
FLOAT, REAL, DOUBLE	數字
DECIMAL	預設為字串。如果 DecimalReturnType 選項設定為 DOUBLE_OR_LONG 則為數字。
STRING	字串
BOOLEAN, BIT	Boolean
BLOB, BINARY, VARBINARY ，LONGVARBINARY	base64 編碼的字串。
CLOB	字串
ARRAY	陣列
NULL	null
其他類型 (包含與日期和時間相關的類型)	字串

故障診斷

JSON 回應限制為 10 MB。若回應大於此限制，您的程式會收到 BadRequestException 錯誤。於此狀況下，您可使用下列其中一種方法解決此錯誤：

- 減少結果集中的列數量。若要執行此作業，請新增 LIMIT 子句。您可使用 LIMIT 和 OFFSET 子句提交多個查詢，將一個較大結果集分割為多個較小的結果集。

如果結果集包含按應用程式邏輯過濾掉的列，則可以透過在 WHERE 子句中新增更多條件來從結果集中移除這些列。

- 減少結果集中的欄數量。若要執行此作業，請從查詢的選取清單中移除項目。
- 在查詢中使用欄別名，來縮短欄標籤。對於結果集中的每一列，每個欄名稱都會在 JSON 字串中重複。因此，具有冗長欄名稱和許多列的查詢結果，可能會超出大小限制。特別是，對複雜的表達式使用欄別名，以避免在 JSON 字串中重複整個表達式。
- 雖然搭配 SQL 可以使用欄別名來產生具有多個相同名稱之欄的結果集，但 JSON 中不允許重複的鍵名稱。如果您請求 JSON 格式的結果集，並且多個欄具有相同的名稱，則 RDS Data API 會傳回錯誤。因此，請確保所有欄標籤都具有唯一的名稱。

範例

以下 Java 範例示範如何呼叫 ExecuteStatement 並搭配 JSON 格式字串的回應，然後解讀結果集。以適當的值來取代 *databaseName*、*secretStoreArn* 以及 *clusterArn* 參數。

以下 Java 範例示範了在結果集中傳回十進位數值的查詢。assertThat 呼叫測試回應的欄位是否具有根據 JSON 結果集規則的預期屬性。

此範例適用於以下結構描述和範例資料：

```
create table test_simplified_json (a float);
insert into test_simplified_json values(10.0);
```

```
public void JSON_result_set_demo() {
    var sql = "select * from test_simplified_json";
    var request = new ExecuteStatementRequest()
      .withDatabase(databaseName)
      .withSecretArn(secretStoreArn)
      .withResourceArn(clusterArn)
      .withSql(sql)
      .withFormatRecordsAs(RecordsFormatType.JSON);
    var result = rdsdataClient.executeStatement(request);
}
```

前面程式中 formattedRecords 欄位的值為：

```
[{"a":10.0}]
```

Records 和 ColumnMetadata 欄位都為 null，因為存在 JSON 結果集。

以下 Java 範例示範了在結果集中傳回整數值的查詢。範例會呼叫 getFormattedRecords 以僅傳回 JSON 格式的字串，並忽略其他空白或 null 的回應欄位。範例會將結果還原序列化為表示記錄清單的結構。每個記錄都具有其名稱對應於結果集中的欄別名的欄位。此技術簡化了用於剖析結果集的程式碼。您的應用程式不必循環對結果集的列和欄執行迴圈，以及將每個值轉換為適當的類型。

此範例適用於以下結構描述和範例資料：

```
create table test_simplified_json (a int);
insert into test_simplified_json values(17);
```

```
public void JSON_deserialization_demo() {
    var sql = "select * from test_simplified_json";
    var request = new ExecuteStatementRequest()
      .withDatabase(databaseName)
      .withSecretArn(secretStoreArn)
      .withResourceArn(clusterArn)
      .withSql(sql)
      .withFormatRecordsAs(RecordsFormatType.JSON);
    var result = rdsdataClient.executeStatement(request)
      .getFormattedRecords();

/* Turn the result set into a Java object, a list of records.
   Each record has a field 'a' corresponding to the column
   labelled 'a' in the result set. */
    private static class Record { public int a; }
    var recordsList = new ObjectMapper().readValue(
        response, new TypeReference<List<Record>>() {
        });
}
```

前面程式中 formattedRecords 欄位的值為：

```
[{"a":17}]
```

若要擷取結果列 0 的 a 欄，則應用程式將參考 recordsList.get(0).a。

相比之下，下面的 Java 範例顯示了當您不使用 JSON 格式時，建置包含結果集的資料結構所需的程式碼類型。在這種情況下，結果集的每一列都包含單一使用者相關資訊的欄位。建置表示結果集的資料結構需要在所有列中執行迴圈。對於每一列，程式碼會擷取每個欄位的值、執行適當的類型轉換，並將結

果指派給代表列之物件中的相應欄位。然後，程式碼將代表每個使用者的物件新增到表示整個結果集的資料結構中。如果查詢變更為對結果集中的欄位進行重新排序、新增或移除，則應用程式的程式碼也必須變更。

```
/* Verbose result-parsing code that doesn't use the JSON result set format */
for (var row: response.getRecords()) {
    var user = User.builder()
      .userId(row.get(0).getLongValue())
      .firstName(row.get(1).getStringValue())
      .lastName(row.get(2).getStringValue())
      .dob(Instant.parse(row.get(3).getStringValue()))
      .build();
    result.add(user);
  }
```

下面的範例值顯示了具有不同欄數、欄別名和欄資料類型之結果集的 formattedRecords 欄位。

如果結果集包含多列，則每一列會表示為陣列元素的物件。結果集中的每一欄都將成為物件中的鍵。結果集中的每一列都會重複這些鍵。因此，對於包含許多列和欄的結果集，您可能需要定義簡短的欄別名，以避免超出整個回應的長度限制。

此範例適用於以下結構描述和範例資料：

```
create table sample_names (id int, name varchar(128));
insert into sample_names values (0, "Jane"), (1, "Mohan"), (2, "Maria"), (3, "Bruce"),
 (4, "Jasmine");
```

```
[{"id":0,"name":"Jane"},{"id":1,"name":"Mohan"},
{"id":2,"name":"Maria"},{"id":3,"name":"Bruce"},{"id":4,"name":"Jasmine"}]
```

如果結果集中的欄被定義為表達式，則表達式的文字將成為 JSON 索引鍵。因此，在查詢的選取清單中為每個表達式定義描述性欄別名通常很方便。例如，以下查詢在其選取清單中包括函數呼叫和算術運算等表達式。

```
select count(*), max(id), 4+7 from sample_names;
```

這些表達式會以索引鍵的形式傳遞給 JSON 結果集。

```
[{"count(*)":5,"max(id)":4,"4+7":11}]
```

新增具有描述性標籤的 AS 欄可讓鍵在 JSON 結果集中更易於解讀。

```
select count(*) as rows, max(id) as largest_id, 4+7 as addition_result from
 sample_names;
```

對於修訂後的 SQL 查詢，AS 子句定義的欄標籤用作鍵名稱。

```
[{"rows":5,"largest_id":4,"addition_result":11}]
```

JSON 字串中每個鍵值對的值都可以是帶引號的字串。字串可能包含 Unicode 字元。如果字串包含逸出序列或 " 或 \ 字元，則這些字元前面會有反斜線逸出字元。以下 JSON 字串範例示範了這些可能性。例如，string_with_escape_sequences 結果包含特殊字元退格、換行字元、歸位字元、tab 鍵、表單摘要和 \。

```
[{"quoted_string":"hello"}]
[{"unicode_string":"####"}]
[{"string_with_escape_sequences":"\b \n \r \t \f \\ '"}]
```

JSON 字串中每個鍵值對的值也可以表示一個數字。數字可以是整數、浮點值、負值或以指數表示法表示的值。以下 JSON 字串範例示範了這些可能性。

```
[{"integer_value":17}]
[{"float_value":10.0}]
[{"negative_value":-9223372036854775808,"positive_value":9223372036854775807}]
[{"very_small_floating_point_value":4.9E-324,"very_large_floating_point_value":1.79769313486231
```

布林值和 null 值用未加引號的特殊關鍵字 true、false 以及 null 來表示。以下 JSON 字串範例示範了這些可能性。

```
[{"boolean_value_1":true,"boolean_value_2":false}]
[{"unknown_value":null}]
```

如果選擇 BLOB 類型的值，則結果將在 JSON 字串中表示為 base64 編碼的值。若要將值轉換回原始表示法，您可以使用應用程式語言中的適當解碼函數。例如，在 Java 中，您可以呼叫函數 Base64.getDecoder().decode()。以下範例輸出顯示選擇 BLOB 值 hello world 並將結果集作為 JSON 字串傳回。

```
[{"blob_column":"aGVsbG8gd29ybGQ="}]
```

以下 Python 範例示範如何從呼叫 Python `execute_statement` 函數的結果中存取值。結果集是欄位 `response['formattedRecords']` 中的字串值。程式碼透過呼叫 `json.loads` 函數，將 JSON 字串轉換為資料結構。然後，結果集的每一列都是資料結構中的一個清單元素，而在每一列中，您可以按名稱參考結果集的每個欄位。

```
import json

result = json.loads(response['formattedRecords'])
print (result[0]["id"])
```

以下 JavaScript 範例示範了如何從呼叫 JavaScript `executeStatement` 函數的結果中存取值。結果集是欄位 `response.formattedRecords` 中的字串值。程式碼透過呼叫 `JSON.parse` 函數，將 JSON 字串轉換為資料結構。然後，結果集的每一列都是資料結構中的一個陣列元素，而在每一列中，您可以按名稱參考結果集的每個欄位。

```
<script>
    const result = JSON.parse(response.formattedRecords);
    document.getElementById("display").innerHTML = result[0].id;
</script>
```

針對 Amazon RDS 資料 API 進行疑難排解

請使用下列標題包含常見錯誤訊息的章節，協助針對您在使用 Amazon RDS 資料 API (資料 API) 時所發生的問題進行疑難排解。

主題

- 找不到交易 <transaction_ID>
- 查詢封包過大
- 資料庫回應超過大小上限
- 叢集 <cluster_ID> 並未啟用 HttpEndpoint
- DatabaseErrorException：交易仍在執行查詢
- 不支援的結果例外狀況
- 不支援多陳述式
- 不支援結構描述參數
- IPv6 連線問題

找不到交易 <transaction_ID>

在此情況下，表示找不到資料 API 呼叫中指定的交易 ID。此問題的原因已附加至錯誤訊息中，且是下列其中一項：

- Transaction may be expired (交易可能已過期)。

  請確認每個交易呼叫都是在前一個呼叫後的三分鐘內執行。

  指定的交易 ID 也可能不是由 BeginTransaction 呼叫所建立的。請確認您的呼叫具有效的交易 ID。
- One previous call resulted in a termination of your transaction (之前的一次呼叫導致您的交易終止)。

  該交易已由您的 `CommitTransaction` 或 `RollbackTransaction` 呼叫終止。
- Transaction has been aborted due to an error from a previous call (由於前次呼叫的錯誤，交易已中止)。

  檢查您之前的呼叫是否發生任何異常。

如需執行交易的資訊，請參閱 呼叫 Amazon RDS 資料 API。

查詢封包過大

在這種情況下，表示為資料列傳回的結果集過大。針對資料庫傳回結果集中的每個資料列，資料 API 的大小上限為每個資料列 64 KB。

若要解決此問題，請確定結果集的每個資料列都是 64 KB 或更小。

資料庫回應超過大小上限

在這種情況下，表示資料庫傳回結果集的大小過大。針對資料庫傳回的結果集，資料 API 的大小上限為 1 MiB。

若要解決此問題，請確認對資料 API 呼叫傳回的資料小於或等於 1 MiB。若您需要傳回超過 1 MiB，您可以在查詢中搭配 `LIMIT` 子句使用多個 `ExecuteStatement` 呼叫。

如需 `LIMIT` 子句的詳細資訊，請參閱 MySQL 文件中的 SELECT 語法。

叢集 <cluster_ID> 並未啟用 HttpEndpoint

檢查此問題的下列潛在原因：

- Aurora 資料庫叢集不支援資料 API。如需有關 RDS 資料 API 支援的資料庫叢集類型的相關資訊，請參閱 the section called “Amazon RDS 資料 API 的區域和版本可用性”。
- Aurora 資料庫叢集並未啟用資料 API。若要搭配 Aurora 資料庫叢集使用資料 API，您必須為資料庫叢集啟用資料 API。如需啟用資料 API 的資訊，請參閱啟用 Amazon RDS 資料 API。
- 資料庫叢集已在為其啟用資料 API 後重新命名。在這種情況下，請關閉該叢集的資料 API，然後再次將其啟用。
- 您指定的 ARN 未精確地符合叢集的 ARN。檢查從其他來源傳回的 ARN 或由程序邏輯建置的 ARN 是否完全符合叢集的 ARN。例如，請確保您使用的 ARN 對於所有字母字元具有正確的字母大小寫。

DatabaseErrorException：交易仍在執行查詢

如果應用程式傳送具有交易 ID 的請求，且該交易目前正在處理另一個請求，資料 API 會立即將此錯誤傳回給應用程式。如果應用程式使用 Javascript 中的「承諾」等機制提出異步請求，則可能會出現此情況。

若要解決此問題，請等待先前的請求完成，然後重試請求。您可以繼續重試，直到不再發生錯誤，或應用程式收到某種不同類型的錯誤。

適用於 Aurora Serverless v2 和佈建執行個體的資料 API 可能會發生此情況。在 Aurora Serverless v1 的資料 API 中，相同交易 ID 的後續請求會自動等待先前的請求完成。不過，由於先前的請求耗時太久，較舊的行為可能會遇到逾時。如果您要移植較舊資料 API 應用程式，且這類應用程式會提出並行請求，請修改例外狀況處理邏輯，以應對這種新的錯誤。

不支援的結果例外狀況

資料 API 不支援所有資料類型。當您執行的是傳回不支援資料類型的查詢時，就會發生此錯誤。

若要解決此問題，請將不支援的資料類型轉換為 TEXT。例如：

```
SELECT custom_type::TEXT FROM my_table;
-- OR
SELECT CAST(custom_type AS TEXT) FROM my_table;
```

不支援多陳述式

Aurora Serverless 第 2 版和佈建叢集的資料 API 不支援多陳述式。嘗試在單一 API 呼叫中執行多個陳述式會導致此錯誤。

若要執行多個陳述式，請使用個別的 `ExecuteStatement` API 呼叫或使用 `BatchExecuteStatement` API 進行批次處理。

不支援結構描述參數

Aurora Serverless 第 1 版以無提示的方式忽略結構描述參數。不過，Aurora Serverless 第 2 版和佈建叢集會明確拒絕內含結構描述參數的 API 呼叫。

若要解決此問題，請在使用 Aurora Serverless 第 2 版或佈建叢集時，從對資料 API 的所有呼叫中移除結構描述參數。

IPv6 連線問題

如果您在使用 IPv6 端點連線至資料 API 時遇到問題，請確認下列潛在原因：

- 網路不支援 IPv6：確認網路基礎設施支援 IPv6，且 IPv6 路由的設定都正確。
- DNS 解析問題：確保 DNS 解析程式可以解析雙堆疊端點的 AAAA 記錄 (例如 `rds-data.us-east-1.api.aws`)。
- 安全群組組態：更新安全群組規則，以允許連接埠 443 (HTTPS) 上的 IPv6 流量。新增 IPv6 CIDR 區塊的規則 (例如，所有 IPv6 地址的 `::/0`)。
- 網路 ACL 組態：確保網路 ACL 允許必要連接埠上的 IPv6 流量。
- 用戶端程式庫相容性：確認您的 HTTP 用戶端程式庫和 AWS SDKs支援 IPv6 和雙堆疊連線。
- VPC 端點組態：如果使用的是 PrivateLink，請確定 VPC 端點已設定為支援 IPv6，且相關聯的子網路已指派 IPv6 CIDR 區塊。

若要針對 IPv6 連線問題進行疑難排解：

1. 使用僅 IPv4 端點 (`.amazonaws.com`) 測試連線，以確認問題是否僅限於 IPv6。
2. 使用網路診斷工具，來驗證雙堆疊端點的 IPv6 連線。
3. 使用 IPv6 端點時，請確認 CloudTrail 日誌是否有任何身分驗證或授權錯誤。
4. 確認應用程式已正確設定為使用新的雙堆疊端點 URL。

使用 記錄 Amazon RDS Data API 呼叫 AWS CloudTrail

RDS Data API （資料 API) 已與 服務整合 AWS CloudTrail，此服務可提供使用者、角色或 AWS 服務在資料 API 中採取之動作的記錄。CloudTrail 將資料 API 的所有 API 呼叫擷取為事件，包括來自

Amazon RDS 主控台的呼叫以及來自對資料 API 操作發出的程式碼呼叫。如果您建立線索，就可以持續將 CloudTrail 事件交付至 Amazon S3 儲存貯體，包括資料 API 的事件。使用 CloudTrail 收集的資訊，您可以判斷眾多資訊。此資訊包括對資料 API 提出的請求、提出請求的 IP 地址、何人提出請求、何時提出請求，以及其他詳細資訊。

若要進一步了解 CloudTrail，請參閱「AWS CloudTrail 使用者指南」。

使用 CloudTrail 中的資料 API 資訊

當您建立 AWS 帳戶時，會在您的帳戶上啟用 CloudTrail。當資料 API 中發生支援的活動（管理事件）時，該活動會與事件歷史記錄中的其他 AWS 服務事件一起記錄在 CloudTrail 事件中。您可以在 AWS 帳戶中檢視、搜尋和下載最近的管理事件。如需詳細資訊，請參閱 AWS CloudTrail 使用者指南中的使用 CloudTrail 事件歷史記錄。

若要持續記錄您 AWS 帳戶中的事件，包括資料 API 的事件，請建立追蹤。線索能讓 CloudTrail 將日誌檔案交付至 Amazon S3 儲存貯體。根據預設，當您在主控台中建立線索時，線索會套用至所有 AWS 區域。線索會記錄 AWS 分割區中所有 AWS 區域的事件，並將日誌檔案交付至您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中所收集的事件資料。如需詳細資訊，請參閱 AWS CloudTrail 使用者指南中的以下主題：

- 建立追蹤的概觀
- CloudTrail 支援的服務和整合
- 設定 CloudTrail 的 Amazon SNS 通知
- 接收多個區域的 CloudTrail 日誌檔案及接收多個帳戶的 CloudTrail 日誌檔案

所有資料 API 操作均由 CloudTrail 記錄，並記載於 Amazon RDS 資料服務 API 參考中。例如，對 `BatchExecuteStatement`、`BeginTransaction`、`CommitTransaction` 和 `ExecuteStatement` 操作的呼叫都會在 CloudTrail 日誌檔案中產生項目。

每一筆事件或日誌項目都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 該請求是否使用根或使用者憑證提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 請求是否由其他 AWS 服務提出。

如需詳細資訊，請參閱 CloudTrail userIdentity 元素。

從 AWS CloudTrail 追蹤中包含和排除資料 API 事件

大多數資料 API 使用者依賴 AWS CloudTrail 追蹤中的事件來提供資料 API 操作的記錄。事件資料不會在資料 API 的請求中顯示資料庫名稱、結構描述名稱或 SQL 陳述式。不過，知道哪個使用者在特定時間對特定資料庫叢集進行某種呼叫，有助於偵測異常存取模式。

在 AWS CloudTrail 追蹤中包含資料 API 事件

對於 Aurora PostgreSQL Serverless v2 和佈建資料庫，下列資料 API 操作會記錄 AWS CloudTrail 為資料事件。資料事件是 CloudTrail 預設不會記錄的大容量資料平面 API 操作。資料事件需支付額外的費用。如需有關 CloudTrail 定價的資訊，請參閱 AWS CloudTrail 定價。

- BatchExecuteStatement
- BeginTransaction
- CommitTransaction
- ExecuteStatement
- RollbackTransaction

您可以使用 CloudTrail 主控台、AWS CLI或 CloudTrail API 操作，來記錄這些資料 API 操作。在 CloudTrail 主控台中，為資料事件類型選擇 RDS 資料 API：資料庫叢集。如需詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的使用 AWS 管理主控台記錄資料事件。

使用 AWS CLI執行 `aws cloudtrail put-event-selectors`命令，為您的線索記錄這些資料 API 操作。若要記錄資料庫叢集上所有資料 API 事件，請為資源類型指定 `AWS::RDS::DBCluster`。下列範例會記錄資料庫叢集上的所有資料 API 事件。如需詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的使用 AWS Command Line Interface記錄資料事件。

```
aws cloudtrail put-event-selectors --trail-name trail_name --advanced-event-selectors \
'{
   "Name": "RDS Data API Selector",
   "FieldSelectors": [
      {
         "Field": "eventCategory",
         "Equals": [
            "Data"
         ]
      },
      {
```

```
          "Field": "resources.type",
          "Equals": [
             "AWS::RDS::DBCluster"
          ]
       }
    ]
}'
```

您可以設定進階事件選擇器，以額外篩選 `readOnly`、`eventName`, 和 `resources.ARN` 欄位。如需這些欄位的詳細資訊，請參閱 AdvancedFieldSelector。

從 AWS CloudTrail 追蹤中排除資料 API 事件 (Aurora Serverless v1僅限)

對於 Aurora Serverless v1，資料 API 事件是管理事件。根據預設，所有資料 API 事件都會包含在 AWS CloudTrail 追蹤中。但是，因為資料 API 會產生大量的事件，您就能夠將這些事件從 CloudTrail 線索中排除。排除 Amazon RDS 資料 API 事件設定會將所有資料 API 事件從線索中排除。您無法排除特定的資料 API 事件。

若要將資料 API 事件從權杖中排除，請執行下列程序：

- 在 CloudTrail 主控台中，當您建立權杖或更新權杖時，選擇 Exclude Amazon RDS Data API events (排除 Amazon RDS 資料 API 事件) 設定。
- 在 CloudTrail API 中，使用 PutEventSelectors 操作。如果使用的是進階事件選擇器，您可以將 `eventSource` 欄位設定為不等於 `rdsdata.amazonaws.com`，即可排除資料 API 事件。如果使用的是基本事件選擇器，您可以將 `ExcludeManagementEventSources` 屬性的值設定為 `rdsdata.amazonaws.com` 來排除資料 API 事件。如需詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的使用 AWS Command Line Interface記錄事件。

> **Warning**
>
> 從 CloudTrail 日誌中排除資料 API 事件可能會遮蔽資料 API 動作。授予委託人執行此操作所需的 `cloudtrail:PutEventSelectors` 許可時時，請務必小心。

您可以變更主控台設定或權杖的事件選擇器，以隨時關閉此排除。然後，權杖即會開始記錄資料 API 事件。但無法復原排除有效期間發生過的資料 API 事件。

當您使用主控台或 API 排除資料 API 事件時，產生的CloudTrail `PutEventSelectors` API 操作也會記錄在 CloudTrail 日誌中。如果資料 API 事件未出現在 CloudTrail 日誌中，

請尋找 ExcludeManagementEventSources 屬性設為 rdsdata.amazonaws.com 的 PutEventSelectors 事件。

如需更多詳細資訊，請參閱 AWS CloudTrail 使用者指南中的記錄權杖的管理事件。

了解資料 API 日誌檔案項目

權杖是一種組態，能讓事件以日誌檔案的形式交付到您指定的 Amazon S3 儲存貯體。CloudTrail 日誌檔案包含一或多個日誌專案。一個事件為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、請求參數等資訊。CloudTrail 日誌檔並非依公有 API 呼叫的堆疊追蹤排序，因此不會以任何特定順序出現。

Aurora PostgreSQL Serverless 第 2 版和已佈建

下列範例示範 CloudTrail 日誌項目，此項目會示範 Aurora PostgreSQL Serverless 第 2 版和已佈建資料庫的 ExecuteStatement 操作。對於這些資料庫，所有資料 API 事件都是事件來源為 rdsdataapi.amazonaws.com 且事件類型為 Rds Data Service 的資料事件。


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T00:49:34Z",
    "eventSource": "rdsdataapi.amazonaws.com",
    "eventName": "ExecuteStatement",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.16.102 Python/3.7.2 Windows/10 botocore/1.12.92",
    "requestParameters": {
        "continueAfterTimeout": false,
        "database": "**********",
        "includeResultMetadata": false,
        "parameters": [],
        "resourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-database-1",
        "schema": "**********",
```

```
        "secretArn": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:dataapisecret-ABC123",
        "sql": "**********"
    },
    "responseElements": null,
    "requestID": "6ba9a36e-b3aa-4ca8-9a2e-15a9eada988e",
    "eventID": "a2c7a357-ee8e-4755-a0d0-aed11ed4253a",
    "eventType": "Rds Data Service",
    "recipientAccountId": "123456789012"
}
```

Aurora Serverless v1

下列範例示範上述範例 CloudTrail 日誌項目在 Aurora Serverless v1 會如何顯示。對於 Aurora Serverless v1，所有事件都是管理事件，其中事件來源為 rdsdata.amazonaws.com，而事件類型為 AwsApiCall。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T00:49:34Z",
    "eventSource": "rdsdata.amazonaws.com",
    "eventName": "ExecuteStatement",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.16.102 Python/3.7.2 Windows/10 botocore/1.12.92",
    "requestParameters": {
        "continueAfterTimeout": false,
        "database": "**********",
        "includeResultMetadata": false,
        "parameters": [],
        "resourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-database-1",
        "schema": "**********",
        "secretArn": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:dataapisecret-ABC123",
```

```
        "sql": "**********"
    },
    "responseElements": null,
    "requestID": "6ba9a36e-b3aa-4ca8-9a2e-15a9eada988e",
    "eventID": "a2c7a357-ee8e-4755-a0d0-aed11ed4253a",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

使用 Performance Insights 監控 RDS 資料 API 查詢

如果 Aurora 叢集正在執行 Aurora Serverless v2 或佈建執行個體，您可以使用 Performance Insights 搭配 RDS 資料 API。

如需如何搭配 Aurora 使用 Performance Insights 的詳細資訊，請參閱 在 Amazon Aurora 上使用績效詳情監控資料庫負載。

Performance Insights 中 RDS 資料 API 查詢的呈現方式

透過資料 API，Aurora 叢集會根據您從應用程式提交的資料 API 呼叫來處理查詢。資料 API 也會執行一些 SQL 陳述式作為其內部工作的一部分，例如將超過逾時閾值的查詢取消。這兩種 SQL 操作都會顯示在 Performance Insights 統計資料和圖表中。

- 對於您提交至 Aurora 叢集的資料 API 查詢，PI 儀表板中的主機欄位會標記為 RDS 資料 API。對於 Aurora PostgreSQL，application_name 欄位的值為 `rds-data-api`。當您使用最高主機或最高應用程式作為維度來分析資料庫負載時，請尋找這些標籤。
- 資料 API 執行以管理資料庫層面的所有內部查詢 (例如連線集區和查詢逾時)，都會以字首 RDS 資料 API 標註。範例：當您依最高 SQL 作為維度分析資料庫負載時，`/* RDS Data API */ select * from my_table;` 會尋找這些字首。系統會以 `/* RDS Data API */` SQL 註解標註陳述式。

使用 Aurora 查詢編輯器

Aurora 查詢編輯器可讓您透過 AWS 管理主控台 在 Aurora 資料庫叢集上執行 SQL 陳述式。您可以執行 SQL 查詢、資料處理 (DML) 陳述式和資料定義 (DDL) 陳述式。使用主控台介面可讓您執行資料庫維護、產生報告，以及執行 SQL 實驗。您可以避免設定網路組態，從 EC2 執行個體或筆記型電腦等個別用戶端系統連線至資料庫叢集。

查詢編輯器需要有已啟用 RDS 資料 API (資料 API) 的 Aurora 資料庫叢集。如需支援資料 API 的資料庫叢集及其啟用方式的詳細資訊，請參閱 使用 Amazon RDS 資料 API。您可以執行的 SQL 受限於資料 API 限制。如需詳細資訊，請參閱the section called “限制”。

查詢編輯器的可用性

查詢編輯器適用於使用支援資料 API 的 Aurora MySQL 和 Aurora PostgreSQL 引擎版本的 Aurora 資料庫叢集，以及可使用資料 API AWS 區域的 。如需詳細資訊，請參閱RDS 資料 API 的支援區域和 Aurora 資料庫引擎。

授權存取查詢編輯器

使用者必須經過授權，才能在查詢編輯器中執行查詢。您可以將預先定義的 AWS Identity and Access Management(IAM) `AmazonRDSDataFullAccess`政策政策新增至該使用者，以授權使用者在查詢編輯器中執行查詢。

> **Note**
>
> 當您建立 IAM 使用者時，請務必使用與您為資料庫使用者使用相同的使用者名稱與密碼，例如管理使用者名稱和密碼。如需詳細資訊，請參閱《AWS Identity and Access Management 使用者指南》中的在您的 AWS 帳戶 中建立 IAM 使用者。

您也可以建立 IAM 政策來授予存取查詢編輯器。建立政策後，將它新增給每一位需要存取查詢編輯器的使用者。

下列政策提供使用者存取查詢編輯器所需的最低許可。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "QueryEditor0",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutResourcePolicy",
                "secretsmanager:PutSecretValue",
                "secretsmanager:DeleteSecret",
                "secretsmanager:DescribeSecret",
                "secretsmanager:TagResource"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*"
        },
        {
            "Sid": "QueryEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "tag:GetResources",
                "secretsmanager:CreateSecret",
                "secretsmanager:ListSecrets",
                "dbqms:CreateFavoriteQuery",
                "dbqms:DescribeFavoriteQueries",
                "dbqms:UpdateFavoriteQuery",
                "dbqms:DeleteFavoriteQueries",
                "dbqms:GetQueryString",
                "dbqms:CreateQueryHistory",
                "dbqms:UpdateQueryHistory",
                "dbqms:DeleteQueryHistory",
                "dbqms:DescribeQueryHistory",
                "rds-data:BatchExecuteStatement",
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:ExecuteStatement",
                "rds-data:RollbackTransaction"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

如需有關建立 IAM 政策的詳細資訊，請參閱 AWS Identity and Access Management 使用者指南中的建立 IAM 政策。

如需有關將 IAM 政策新增給使用者的資訊，請參閱 AWS Identity and Access Management 使用者指南中的新增和移除 IAM 身分許可。

在查詢編輯器中執行查詢

您可以在查詢編輯器中對 Aurora 資料庫叢集執行 SQL 陳述式。您可以執行的 SQL 受限於資料 API 限制。如需詳細資訊，請參閱the section called “限制”。

在查詢編輯器中執行查詢

1. 登入 AWS 管理主控台並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在 的右上角AWS 管理主控台，選擇您在AWS 區域其中建立您要查詢之 Aurora 資料庫叢集的 。
3. 在導覽窗格中，選擇 Databases (資料庫)。
4. 選擇您想對其執行 SQL 查詢的 Aurora 資料庫叢集。
5. 對於 Actions (動作)，選擇 Query (查詢)。如果您之前沒有連接到資料庫，則會開啟 Connect to database (連接至資料庫) 頁面。



6. 輸入下列資訊：

   a. 對於資料庫執行個體或叢集，選擇您想用來執行 SQL 查詢的 Aurora 資料庫叢集。

   b. 對於 Database username (資料庫使用者名稱)，選擇連線時要使用的資料庫使用者的使用者名稱，或選擇 Add new database credentials (新增資料庫登入資料)。如果您選擇 Add new database credentials (新增資料庫登入資料)，請在 Enter database username (輸入資料庫使用者名稱) 中輸入新資料庫登入資料的使用者名稱。

   c. 對於 Enter database password (輸入資料庫密碼)，針對您已選擇的資料庫使用者輸入密碼。

   d. 在最後的方塊中，輸入您要用於 Aurora 資料庫叢集的資料庫或結構描述的名稱。

   e. 選擇 Connect to database (連接至資料庫)。

> **Note**
>
> 如果連線成功，則連線和身分驗證資訊會存放在 AWS Secrets Manager 中。您不需要再次輸入連線資訊。

7. 在查詢編輯器中，輸入您要對資料庫執行的 SQL 查詢。



每個 SQL 陳述式可以自動遞交，或者您可以在交易過程中使用指令碼執行 SQL 陳述式。若要控制此行為，選擇查詢視窗上方的齒輪圖示。

Query Editor Settings (查詢編輯器設定) 視窗即會顯示。



如果您選擇 Auto-commit (自動遞交)，則每個 SQL 陳述式即會自動遞交。如果您選擇 Transaction (交易)，則可以在指令碼中執行一組陳述式。陳述式會在指令碼結尾自動遞交，除非在此之前明確遞交或復原。您也可以透過啟用 Stop on error (錯誤時停止)，選擇在錯誤發生時停止執行中的指令碼。

> **Note**
>
> 在一組陳述式中，資料定義語言 (DDL) 陳述式可能會導致先前的資料處理語言 (DML) 陳述式進行遞交。您也可以使用指令碼，將 `COMMIT` 和 `ROLLBACK` 陳述式包含在一組陳述式中。

在 Query Editor Settings (查詢編輯器設定) 視窗中進行選擇後，請選擇 Save (儲存)。

8. 選擇 Run (執行) 或按下 Ctrl+Enter，然後查詢編輯器將顯示您查詢的結果。

   執行查詢之後，透過選擇 Save (儲存)，將查詢儲存到 Saved queries (儲存的查詢)。

   透過選擇 Export to csv (匯出至 csv)，以試算表格式匯出查詢結果。

您可以尋找、編輯或傳回先前的查詢。若要這麼做，請選擇 Recent (最近的項目) 標籤或 Saved queries (儲存的查詢) 標籤，選擇查詢文字，然後選擇 Run (執行)。

若要變更資料庫，請選擇 Change database (變更資料庫)。

Database Query Metadata Service (DBQMS) API 參考

Database Query Metadata Service (dbqms) 是僅內部服務。它為多個 AWS 服務 (包括 Amazon RDS) 的 AWS 管理主控台 查詢編輯器提供最近和已儲存的查詢。

支援的 DBQMS 動作

- CreateFavoriteQuery
- CreateQueryHistory
- CreateTab
- DeleteFavoriteQueries
- DeleteQueryHistory
- DeleteTab
- DescribeFavoriteQueries
- DescribeQueryHistory
- DescribeTabs
- GetQueryString

- UpdateFavoriteQuery
- UpdateQueryHistory
- UpdateTab

CreateFavoriteQuery

儲存新的最愛查詢。每位 使用者最多可建立 1000 個已儲存的查詢。此限制將來可能會有所變更。

CreateQueryHistory

儲存新的查詢歷史記錄項目。

CreateTab

儲存新的查詢索引標籤。每位 使用者最多可以建立 10 個查詢索引標籤。

DeleteFavoriteQueries

刪除一個或多個已儲存的查詢。

DeleteQueryHistory

刪除查詢歷史記錄項目。

DeleteTab

刪除查詢索引標籤項目。

DescribeFavoriteQueries

列出使用者在指定帳戶中建立的已儲存查詢。

DescribeQueryHistory

列出查詢歷史記錄項目。

DescribeTabs

列出使用者在指定帳號中建立的查詢索引標籤。

GetQueryString

從查詢 ID 擷取完整的查詢文字。

UpdateFavoriteQuery

更新查詢字串、描述、名稱或過期日。

UpdateQueryHistory

更新查詢歷史記錄的狀態。

UpdateTab

更新查詢索引標籤名稱和查詢字串。

使用 Amazon Aurora 機器學習

透過使用 Amazon Aurora 機器學習，您可以根據需求，將 Aurora 資料庫叢集與下列其中一項 AWS 機器學習服務整合。其各自支援特定的機器學習使用案例。

Amazon Bedrock

Amazon Bedrock 是一項全受管服務，可透過 API 提供來自 AI 公司的頂尖基礎模型，以及提供開發人員工具，以協助建置和擴展生成式 AI 應用程式。若使用 Amazon Bedrock ，您需要付費才能在任何第三方基礎模型上執行推論。定價取決於輸入記號和輸出記號數量，以及您是否已為模型購買佈建的輸送量。如需詳細資訊，請參閱 Amazon Bedrock 使用者指南中的什麼是 Amazon Bedrock？。

Amazon Comprehend

Amazon Comprehend 是一種受管自然語言處理 (NLP) 服務，用來從文件擷取洞見。使用 Amazon Comprehend，您可以藉由分析實體、關鍵片語、語言和其他功能，來推斷以文件內容為基礎的觀點。若要進一步了解，請參閱《Amazon Comprehend 開發人員指南》中的什麼是 Amazon Comprehend？。

SageMaker AI

Amazon SageMaker AI 是全管理式機器學習服務。資料科學家和開發人員會使用 Amazon SageMaker AI，針對各種推論任務 (例如詐騙偵測和產品建議) 建置、訓練和測試機器學習模型。當機器學習模型已準備好可用於生產環境時，可以將其部署到 Amazon SageMaker AI 託管環境。如需詳細資訊，請參閱《Amazon SageMaker AI 開發人員指南》中的什麼是 Amazon SageMaker AI？。

搭配 Aurora 資料庫叢集使用 Amazon Comprehend 比使用 SageMaker AI 具有更少的初步設定。如果您不熟悉 AWS 機器學習，建議您從探索 Amazon Comprehend 開始。

主題

- 將 Amazon Aurora Machine Learning 與 Aurora MySQL 搭配使用
- 將 Amazon Aurora Machine Learning 與 Aurora PostgreSQL 搭配使用

將 Amazon Aurora Machine Learning 與 Aurora MySQL 搭配使用

透過將 Amazon Aurora 機器學習與 Aurora MySQL 資料庫叢集搭配使用，您可以根據需求使用 Amazon Bedrock、Amazon Comprehend 或 Amazon SageMaker AI。其各自支援不同的機器學習使用案例。

內容

- 將 Aurora Machine Learning 與 Aurora MySQL 搭配使用的建議
- 區域和版本可用性
- Aurora Machine Learning 搭配 Aurora MySQL 支援的功能和限制
- 設定 Aurora MySQL 資料庫叢集來使用 Aurora Machine Learning
  - 設定 Aurora MySQL 資料庫叢集來使用 Amazon Bedrock
  - 設定您的 Aurora MySQL 資料庫叢集來使用 Amazon Comprehend
  - 設定 Aurora MySQL 資料庫叢集來使用 SageMaker AI
    - 設定 Aurora MySQL 資料庫叢集來使用 Amazon S3 for SageMaker AI (選用)
  - 授與資料庫使用者存取 Aurora Machine Learning 的權限
    - 授與 Amazon Bedrock 函數的存取權
    - 授與 Amazon Comprehend 函數的存取權
    - 授與 SageMaker AI 函數的存取權
- 搭配 Aurora MySQL 資料庫叢集使用 Amazon Bedrock
- 搭配 Aurora MySQL 資料庫叢集使用 Amazon Comprehend
- 搭配 Aurora MySQL 資料庫叢集使用 SageMaker AI
  - 傳回字串的 SageMaker AI 函數的字元集需求
  - 將資料匯出至 Amazon S3 進行 SageMaker AI 模型訓練 (進階)
- 搭配 Aurora MySQL 使用 Aurora Machine Learning 的效能考量
  - 模型和提示
  - 查詢快取
  - Aurora Machine Learning 函數呼叫的批次最佳化
- 監控 Aurora Machine Learning

將 Aurora Machine Learning 與 Aurora MySQL 搭配使用的建議

AWS 機器學習服務是在自己的生產環境中設定和執行的受管服務。Aurora 機器學習支援與 Amazon Bedrock、Amazon Comprehend 和 SageMaker AI 的整合。在嘗試設定 Aurora MySQL 資料庫叢集以使用 Aurora Machine Learning 之前，請務必瞭解下列要求和先決條件。

- 機器學習服務必須在與 Aurora MySQL 資料庫叢集 AWS 區域 相同的 中執行。您無法從不同區域的 Aurora MySQL 資料庫叢集使用機器學習服務。
- 如果 Aurora MySQL 資料庫叢集位於與 Amazon Bedrock、Amazon Comprehend 和 SageMaker AI 服務不同的虛擬公有雲 (VPC)，則 VPC 的安全群組需要允許目標 Aurora 機器學習服務的輸出連線。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的使用安全群組控制到 AWS 資源的流量。
- 如果您想要將 Aurora Machine Learning 與該叢集搭配使用，則可以將執行較低版本的 Aurora MySQL 的 Aurora 叢集升級至支援的較高版本。如需詳細資訊，請參閱Amazon Aurora MySQL 的資料庫引擎更新。
- Aurora MySQL 資料庫叢集必須使用自訂資料庫叢集參數群組。在設定您要使用之每個 Aurora Machine Learning 服務的程序結束時，您可以新增針對該服務所建立之相關聯 IAM 角色的 Amazon Resource Name (ARN)。我們建議您事先為 Aurora MySQL 建立自訂資料庫叢集參數群組，並將 Aurora MySQL 資料庫叢集設定為使用它，以便在安裝程序結束時您就可以加以修改。
- 對於 SageMaker AI：
  - 您要用於推論的機器學習元件必須已設定並準備好使用。在設定 Aurora MySQL 資料庫叢集的過程中，您必須具有 SageMaker AI 端點的 ARN 可供使用。您團隊中的資料科學家可能最能夠處理如何使用 SageMaker，以準備模型並處理其他這類任務。若要開始使用 Amazon SageMaker AI，請參閱開始使用 Amazon SageMaker AI。如需推論和端點的詳細資訊，請參閱即時推論。
  - 若要搭配您自己的訓練資料使用 SageMaker AI，您需要設定 Amazon S3 儲存貯體，作為 Aurora 機器學習的 Aurora MySQL 組態一部分。若要這樣做，請遵循與設定 SageMaker AI 整合相同的一般程序。如需此選用設定程序的摘要，請參閱 設定 Aurora MySQL 資料庫叢集來使用 Amazon S3 for SageMaker AI (選用)。
- 對於 Aurora 全域資料庫，您可以設定要在 AWS 區域 組成 Aurora 全域資料庫的所有 中使用的 Aurora 機器學習服務。例如，如果您想要針對 Aurora 全域資料庫使用 Aurora 機器學習搭配 SageMaker AI，請針對每個 AWS 區域中的每個 Aurora MySQL 資料庫叢集執行以下操作：
  - 使用相同的 SageMaker AI 訓練模型和端點設定 Amazon SageMaker AI 服務。這些也必須使用相同的名稱。
  - 建立 IAM 角色，如設定 Aurora MySQL 資料庫叢集來使用 Aurora Machine Learning中所述。

- 將 IAM 角色的 ARN 新增至每個 AWS 區域中每個 Aurora MySQL 資料庫叢集的自訂資料庫叢集參數群組。

這些任務需要 Aurora Machine Learning 可用於組成 Aurora 全域資料庫的所有 中的 Aurora MySQL AWS 區域 版本。

區域和版本可用性

功能可用性和支援會因每個 Aurora 資料庫引擎的特定版本以及 AWS 區域而有所不同。

- 如需 Amazon Comprehend 和 Amazon SageMaker AI 搭配 Aurora MySQL 的版本和區域可用性資訊，請參閱將機器學習與 Aurora MySQL 搭配使用。
- 僅在 Aurora MySQL 3.06 版及更新版本支援 Amazon Bedrock。

  如需 Amazon Bedrock 區域可用性的相關資訊，請參閱《Amazon Bedrock 使用者指南》中的依 AWS 區域的模型支援。

Aurora Machine Learning 搭配 Aurora MySQL 支援的功能和限制

將 Aurora MySQL 與 Aurora 機器學習搭配使用時，適用下列限制：

- Aurora 機器學習延伸模組不支援向量界面。
- 在觸發條件中使用時，不支援 Aurora 機器學習整合。
- Aurora 機器學習函數與二進位記錄 (binlog) 複寫不相容。
  - 對於 Aurora Machine Learning 函數的呼叫，設定 `--binlog-format=STATEMENT` 會擲出例外狀況。
  - Aurora Machine Learning 函數是非確定性函數，而且非確定性預存函數與 binlog 格式不相容。

  如需詳細資訊，請參閱 MySQL 文件中的二進位日誌格式。
- 不支援呼叫具有一律產生的資料行之資料欄的預存函數。這適用於任何 Aurora MySQL 預存函數。若要進一步了解此資料欄類型，請參閱 MySQL 文件中的 CREATE TABLE和產生的資料欄。
- Amazon Bedrock 函數不支援 `RETURNS JSON`。如有需要，您可以使用 `CONVERT` 或 `CAST` 從 `TEXT` 轉換為 `JSON`。
- Amazon Bedrock 不支援批次請求。
- Aurora MySQL 支援任何 SageMaker AI 端點，這些端點可透過 `ContentType` 的 `text/csv` 讀取和寫入逗號分隔值 (CSV) 格式。下列內建 SageMaker AI 演算法可接受此格式：

- Linear Learner
- Random Cut Forest
- XGBoost

若要進一步了解這些演算法，請參閱《Amazon SageMaker AI 開發人員指南》中的選擇演算法。

設定 Aurora MySQL 資料庫叢集來使用 Aurora Machine Learning

在下列主題中，您可以找到其中每個 Aurora Machine Learning 服務的個別設定程序。

主題

- 設定 Aurora MySQL 資料庫叢集來使用 Amazon Bedrock
- 設定您的 Aurora MySQL 資料庫叢集來使用 Amazon Comprehend
- 設定 Aurora MySQL 資料庫叢集來使用 SageMaker AI
  - 設定 Aurora MySQL 資料庫叢集來使用 Amazon S3 for SageMaker AI (選用)
- 授與資料庫使用者存取 Aurora Machine Learning 的權限
  - 授與 Amazon Bedrock 函數的存取權
  - 授與 Amazon Comprehend 函數的存取權
  - 授與 SageMaker AI 函數的存取權

設定 Aurora MySQL 資料庫叢集來使用 Amazon Bedrock

Aurora Machine Learning 依賴 AWS Identity and Access Management (IAM) 角色和政策，以允許 Aurora MySQL 資料庫叢集存取和使用 Amazon Bedrock 服務。下列程序會建立 IAM 許可政策和角色，讓資料庫叢集可與 Amazon Bedrock 整合。

若要建立 IAM 政策

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在導覽窗格中選擇政策。
3. 選擇 Create a policy (建立政策)。
4. 在指定許可頁面，針對選取服務，選擇 Bedrock。

   Amazon Bedrock 許可隨即顯示。
5. 展開讀取，然後選取 InvokeModel。

6. 針對資源選取全部。

   指定許可頁面應類似於下圖。



7. 選擇下一步。

8. 在檢閱和建立頁面，輸入政策名稱 (例如 **`BedrockInvokeModel`**)。
9. 檢閱政策，然後選擇建立政策。

接著，建立使用 Amazon Bedrock 許可政策的 IAM 角色。

建立 IAM 角色

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在導覽窗格中，選擇 Roles (角色)。
3. 選擇建立角色。
4. 在選取信任的實體頁面中，對於使用案例選擇 RDS。
5. 選取 RDS：將角色新增至資料庫，然後選擇下一步。
6. 在新增許可頁面，針對許可政策，選取您建立的 IAM 政策，然後選擇下一步。
7. 在命名、檢閱和建立頁面，輸入角色名稱 (例如 **`ams-bedrock-invoke-model-role`**)。

   角色應與下圖類似。



8. 檢閱角色，然後選擇建立角色。

接下來，您將 Amazon Bedrock IAM 角色與資料庫叢集建立關聯。

將 IAM 角色與資料庫叢集產生關聯

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您要連線至 Amazon Bedrock 服務的 Aurora MySQL 資料庫叢集。
4. 選擇 Connectivity & security (連線和安全) 索引標籤。
5. 在管理 IAM 角色區段中，選擇選取 IAM 以新增至此叢集。
6. 選擇您建立的 IAM，然後選擇新增角色。

   IAM 角色與資料庫叢集相關聯，一開始的狀態為待定，接著是作用中。程序完成時，您可以在 Current IAM roles for this cluster (此叢集的目前 IAM 角色) 清單中找到該角色。



您必須將此 IAM 角色的 ARN 新增至與 Aurora MySQL 資料庫叢集相關聯之自訂資料庫叢集參數群組中的 `aws_default_bedrock_role` 參數。如果您的 Aurora MySQL 資料庫叢集不使用自訂資料庫叢集參數群組，您必須建立一個群組與 Aurora MySQL 資料庫叢集搭配使用，才能完成整合。如需詳細資訊，請參閱Amazon Aurora 資料庫叢集的資料庫叢集參數群組。

設定資料庫叢集參數

1. 在 Amazon RDS 主控台中，開啟 Aurora MySQL 資料庫叢集的 Configuration (組態) 索引標籤。
2. 找到為叢集設定的資料庫叢集參數群組。選擇連結以開啟自訂資料庫叢集參數群組，然後選擇編輯。
3. 在您的自訂資料庫叢集參數群組中尋找 `aws_default_bedrock_role` 參數。
4. 在值欄位中輸入 IAM 角色的 ARN。
5. 選擇 Save Changes (儲存變更) 來儲存設定。

6. 重新啟動 Aurora MySQL 資料庫叢集的主要執行個體，以便此參數設定生效。

Amazon Bedrock 的 IAM 整合已完成。繼續設定 Aurora MySQL 資料庫叢集，以透過 授與資料庫使用者存取 Aurora Machine Learning 的權限 使用 Amazon Bedrock。

設定您的 Aurora MySQL 資料庫叢集來使用 Amazon Comprehend

Aurora Machine Learning 倚賴 AWS Identity and Access Management 角色和政策，讓您的 Aurora MySQL 資料庫叢集存取和使用 Amazon Comprehend 服務。下列程序會自動為您的叢集建立 IAM 角色和政策，以便其可以使用 Amazon Comprehend。

設定您的 Aurora MySQL 資料庫叢集來使用 Amazon Comprehend

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您要連線至 Amazon Comprehend 服務的 Aurora MySQL 資料庫叢集。
4. 選擇 Connectivity & security (連線和安全) 索引標籤。
5. 對於管理 IAM 角色區段中，然後選擇選取服務以連線至這個叢集。
6. 從功能表中選擇 Amazon Comprehend，然後選擇連線服務。



7. Connect cluster to Amazon Comprehend (將叢集連線到 Amazon Comprehend) 對話方塊不需要任何其他資訊。不過，您可能會看到一則訊息，通知您 Aurora 與 Amazon Comprehend 之間的整合目前處於預覽狀態。請務必先閱讀訊息，然後再繼續。如果您不想繼續，則可以選擇取消。
8. 選擇 Connect service (連線服務) 以完成整合程序。

Aurora 即會建立 IAM 角色。其也會建立允許 Aurora MySQL 資料庫叢集使用 Amazon Comprehend 服務的政策，並將該政策附加到這個角色。程序完成時，您可以在下圖中顯示的 Current IAM roles for this cluster (此叢集的目前 IAM 角色) 清單中找到該角色。



您需要將此 IAM 角色的 ARN 新增至與 Aurora MySQL 資料庫叢集相關聯之自訂資料庫叢集參數群組中的 `aws_default_comprehend_role` 參數。如果您的 Aurora MySQL 資料庫叢集不使用自訂資料庫叢集參數群組，您必須建立一個群組與 Aurora MySQL 資料庫叢集搭配使用，才能完成整合。如需詳細資訊，請參閱Amazon Aurora 資料庫叢集的資料庫叢集參數群組。

在建立自訂資料庫叢集參數群組並將其與 Aurora MySQL 資料庫叢集建立關聯之後，您可以繼續遵循下列步驟。

如果您的叢集使用自訂資料庫叢集參數群組，請執行如下操作。

a. 在 Amazon RDS 主控台中，開啟 Aurora MySQL 資料庫叢集的 Configuration (組態) 索引標籤。

b. 找到為叢集設定的資料庫叢集參數群組。選擇連結以開啟自訂資料庫叢集參數群組，然後選擇編輯。

c. 在您的自訂資料庫叢集參數群組中尋找 `aws_default_comprehend_role` 參數。

d. 在值欄位中輸入 IAM 角色的 ARN。

e. 選擇 Save Changes (儲存變更) 來儲存設定。在下圖中，您可以找到一個範例。



重新啟動 Aurora MySQL 資料庫叢集的主要執行個體，以便此參數設定生效。

Amazon Comprehend 的 IAM 整合已完成。透過將存取權授與適當的資料庫使用者，繼續設定 Aurora MySQL 資料庫叢集以使用 Amazon Comprehend。

設定 Aurora MySQL 資料庫叢集來使用 SageMaker AI

下列程序會自動為您的 Aurora MySQL 資料庫叢集建立 IAM 角色和政策，以便其可以使用 SageMaker AI。在嘗試遵循此程序之前，請確定您具有可用的 SageMaker AI 端點，以便可在需要時輸入它。通常，您團隊中的資料科學家會執行這項工作，以產生您可以從 Aurora MySQL 資料庫叢集使用的端點。您可以在 SageMaker AI 主控台中找到此類端點。在導覽窗格中，開啟 Inference (推論) 功能表，然後選擇 Endpoints (端點)。在下圖中，您可以找到一個範例。



設定 Aurora MySQL 資料庫叢集來使用 SageMaker AI

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 從 Amazon RDS 導覽功能表選擇資料庫，然後選擇您要連線至 SageMaker AI 服務的 Aurora MySQL 資料庫叢集。
3. 選擇 Connectivity & security (連線和安全) 索引標籤。
4. 捲動至 Manage IAM roles (管理 IAM 角色) 區段中，然後選擇 Select a service to connect to this cluster (選取服務以連線至這個叢集)。從選擇器中選擇 SageMaker AI。



5. 選擇 Connect service (連線服務)。
6. 在將叢集連線至 SageMaker 對話方塊中，輸入 SageMaker AI 端點的 ARN。



7. Aurora 即會建立 IAM 角色。其也會建立允許 Aurora MySQL 資料庫叢集使用 SageMaker AI 服務的政策，並將該政策附加到這個角色。程序完成時，您可以在 Current IAM roles for this cluster (此叢集的目前 IAM 角色) 清單中找到該角色。
8. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
9. 從 AWS Identity and Access Management 導覽功能表的 Access management (存取管理) 區段中選擇 Roles (角色)。
10. 在列出的角色之中尋找該角色。它的名稱會使用以下模式。

```
rds-sagemaker-your-cluster-name-role-auto-generated-digits
```

11. 開啟角色的 Summary (摘要) 頁面並找出 ARN。請注意 ARN 或使用複製 Widget 將其複製。
12. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
13. 選擇您的 Aurora MySQL 資料庫叢集，然後選擇其 Configuration (組態) 索引標籤。

14. 找出資料庫叢集參數群組，然後選擇連結來開啟您的自訂資料庫叢集參數群組。尋找 `aws_default_sagemaker_role` 參數並在 Value (值) 欄位中輸入 IAM 角色的 ARN，然後儲存設定。
15. 重新啟動 Aurora MySQL 資料庫叢集的主要執行個體，以便此參數設定生效。

IAM 設定現已完成。透過將存取權授與適當的資料庫使用者，繼續設定 Aurora MySQL 資料庫叢集以使用 SageMaker AI。

如果您想要使用 SageMaker AI 模型進行訓練，而不是使用預先建置的 SageMaker AI 元件，您還需要將 Amazon S3 儲存貯體新增至 Aurora MySQL 資料庫叢集，如後續的設定 Aurora MySQL 資料庫叢集來使用 Amazon S3 for SageMaker AI (選用) 中所述。

設定 Aurora MySQL 資料庫叢集來使用 Amazon S3 for SageMaker AI (選用)

若要搭配您自己的模型使用 SageMaker AI，而不是使用 SageMaker AI 提供的預先建置元件，您需要設定 Amazon S3 儲存貯體，供 Aurora MySQL 資料庫叢集使用。如需有關建立 Amazon S3 儲存貯體的詳細資訊，請參閱 Amazon Simple Storage Service 使用者指南中的建立儲存貯體。

設定 Aurora MySQL 資料庫叢集來使用 SageMaker AI 的 Amazon S3 儲存貯體

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 從 Amazon RDS 導覽功能表選擇資料庫，然後選擇您要連線至 SageMaker AI 服務的 Aurora MySQL 資料庫叢集。
3. 選擇 Connectivity & security (連線和安全) 索引標籤。
4. 捲動至 Manage IAM roles (管理 IAM 角色) 區段中，然後選擇 Select a service to connect to this cluster (選取服務以連線至這個叢集)。從選擇器中選擇 Amazon S3。
5. 選擇 Connect service (連線服務)。
6. 在將叢集連線至 Amazon S3 對話方塊中，輸入 Amazon S3 儲存貯體的 ARN，如下圖所示。



7. 選擇 Connect service (連線服務) 以完成此程序。

如需搭配 SageMaker AI 使用 Amazon S3 儲存貯體的詳細資訊，請參閱《Amazon SageMaker AI 開發人員指南》中的指定 Amazon S3 儲存貯體以上傳訓練資料集和存放輸出資料。若要進一步了解如何使用 SageMaker AI，請參閱《Amazon SageMaker AI 開發人員指南》中的開始使用 Amazon SageMaker AI 筆記本執行個體。

授與資料庫使用者存取 Aurora Machine Learning 的權限

資料庫使用者必須獲授與調用 Aurora 機器學習函數的許可。授與許可的方式取決於您用於 Aurora MySQL 資料庫叢集的 MySQL 版本，如下所述。如何執行此操作取決於 Aurora MySQL 資料庫叢集使用的 MySQL 版本。

- 若為 Aurora MySQL 第 3 版 (與 MySQL 8.0 相容)，資料庫使用者必須獲授與適當的資料庫角色。如需詳細資訊，請參閱《MySQL 8.0 參考手冊》中的使用角色。
- 若為 Aurora MySQL 第 2 版 (與 MySQL 5.7 相容)，資料庫使用者會獲授與權限。如需詳細資訊，請參閱《MySQL 5.7 參考手冊》中的存取控制和帳戶管理。

下表顯示資料庫使用者使用機器學習函數所需的角色和權限。

Aurora MySQL 第 3 版 (角色)	Aurora MySQL 第 2 版 (權限)
AWS_BEDROCK_ACCESS	–
AWS_COMPREHEND_ACCESS	INVOKE COMPREHEND
AWS_SAGEMAKER_ACCESS	INVOKE SAGEMAKER

授與 Amazon Bedrock 函數的存取權

若要讓資料庫使用者存取 Amazon Bedrock 函數，請使用下列 SQL 陳述式：

```
GRANT AWS_BEDROCK_ACCESS TO user@domain-or-ip-address;
```

對於您為了使用 Amazon Bedrock 而建立的函數，資料庫使用者也需要獲授與 EXECUTE 許可：

```
GRANT EXECUTE ON FUNCTION database_name.function_name TO user@domain-or-ip-address;
```

最後，資料庫使用者的角色必須設為 AWS_BEDROCK_ACCESS：

```
SET ROLE AWS_BEDROCK_ACCESS;
```

Amazon Bedrock 函數現在可供使用。

授與 Amazon Comprehend 函數的存取權

若要讓資料庫使用者能夠存取 Amazon Comprehend 函數，請針對您的 Aurora MySQL 版本使用適當的陳述式。

- Aurora MySQL 第 3 版 (與 MySQL 8.0 相容)

  ```
  GRANT AWS_COMPREHEND_ACCESS TO user@domain-or-ip-address;
  ```

- Aurora MySQL 第 2 版 (與 MySQL 5.7 相容)

  ```
  GRANT INVOKE COMPREHEND ON *.* TO user@domain-or-ip-address;
  ```

Amazon Comprehend 函數現在可供使用。如需使用範例，請參閱 搭配 Aurora MySQL 資料庫叢集使用 Amazon Comprehend。

授與 SageMaker AI 函數的存取權

若要讓資料庫使用者能夠存取 SageMaker AI 函數，請針對 Aurora MySQL 版本使用適當的陳述式。

- Aurora MySQL 第 3 版 (與 MySQL 8.0 相容)

  ```
  GRANT AWS_SAGEMAKER_ACCESS TO user@domain-or-ip-address;
  ```

- Aurora MySQL 第 2 版 (與 MySQL 5.7 相容)

```
GRANT INVOKE SAGEMAKER ON *.* TO user@domain-or-ip-address;
```

對於您為了使用 SageMaker AI 而建立的函數，資料庫使用者也需要獲授與 EXECUTE 許可。假設您建立了兩個函數 (db1.anomoly_score 和 db2.company_forecasts) 來調用 SageMaker AI 端點的服務。您會如下範例所示授與執行權限。

```
GRANT EXECUTE ON FUNCTION db1.anomaly_score TO user1@domain-or-ip-address1;
GRANT EXECUTE ON FUNCTION db2.company_forecasts TO user2@domain-or-ip-address2;
```

SageMaker AI 函數現在可供使用。如需使用範例，請參閱 搭配 Aurora MySQL 資料庫叢集使用 SageMaker AI。

搭配 Aurora MySQL 資料庫叢集使用 Amazon Bedrock

若要使用 Amazon Bedrock，您可以在調用模型的 Aurora MySQL 資料庫中建立使用者定義函數 (UDF)。如需詳細資訊，請參閱《Amazon Bedrock 使用者指南》中的 Amazon Bedrock 中支援的模型。

UDF 使用以下語法：

```
CREATE FUNCTION function_name (argument type)
        [DEFINER = user]
    RETURNS mysql_data_type
    [SQL SECURITY {DEFINER | INVOKER}]
    ALIAS AWS_BEDROCK_INVOKE_MODEL
    MODEL ID 'model_id'
    [CONTENT_TYPE 'content_type']
    [ACCEPT 'content_type']
    [TIMEOUT_MS timeout_in_milliseconds];
```

- Amazon Bedrock 函數不支援 RETURNS JSON。如有需要，您可以使用 CONVERT 或 CAST 從 TEXT 轉換為 JSON。
- 如未指定 CONTENT_TYPE 或 ACCEPT，預設值將會是 application/json。
- 如未指定 TIMEOUT_MS，將會使用 aurora_ml_inference_timeout 值。

例如，下列 UDF 會調用 Amazon Titan Text 快捷版模型：

```
CREATE FUNCTION invoke_titan (request_body TEXT)
    RETURNS TEXT
    ALIAS AWS_BEDROCK_INVOKE_MODEL
    MODEL ID 'amazon.titan-text-express-v1'
    CONTENT_TYPE 'application/json'
    ACCEPT 'application/json';
```

若要允許資料庫使用者使用此函數，請使用下列 SQL 命令：

```
GRANT EXECUTE ON FUNCTION database_name.invoke_titan TO user@domain-or-ip-address;
```

然後，如下列範例所示，使用者可以像任何其他函數一樣呼叫 invoke_titan。請務必根據 Amazon Titan 文字模型格式化請求內文。

```
CREATE TABLE prompts (request varchar(1024));
INSERT INTO prompts VALUES (
'{
    "inputText": "Generate synthetic data for daily product sales in various categories
 - include row number, product name, category, date of sale and price. Produce output
 in JSON format. Count records and ensure there are no more than 5.",
    "textGenerationConfig": {
        "maxTokenCount": 1024,
        "stopSequences": [],
        "temperature":0,
        "topP":1
    }
}');

SELECT invoke_titan(request) FROM prompts;

{"inputTextTokenCount":44,"results":[{"tokenCount":296,"outputText":"
```tabular-data-json
{
    "rows": [
        {
            "Row Number": "1",
            "Product Name": "T-Shirt",
            "Category": "Clothing",
            "Date of Sale": "2024-01-01",
            "Price": "$20"
        },
        {
```
```

```
            "Row Number": "2",
            "Product Name": "Jeans",
            "Category": "Clothing",
            "Date of Sale": "2024-01-02",
            "Price": "$30"
        },
        {
            "Row Number": "3",
            "Product Name": "Hat",
            "Category": "Accessories",
            "Date of Sale": "2024-01-03",
            "Price": "$15"
        },
        {
            "Row Number": "4",
            "Product Name": "Watch",
            "Category": "Accessories",
            "Date of Sale": "2024-01-04",
            "Price": "$40"
        },
        {
            "Row Number": "5",
            "Product Name": "Phone Case",
            "Category": "Accessories",
            "Date of Sale": "2024-01-05",
            "Price": "$25"
        }
    ]
}
```","completionReason":"FINISH"}]}

對於您使用的其他模型，請務必為其適當地格式化請求內文。如需詳細資訊，請參閱《Amazon Bedrock 使用者指南》中的基礎模型的推論參數。

搭配 Aurora MySQL 資料庫叢集使用 Amazon Comprehend

對於 Aurora MySQL，Aurora Machine Learning 提供下列兩個內建函數，用於使用 Amazon Comprehend 和您的文字資料。您會提供要分析的文字 (`input_data`) 並指定語言 (`language_code`)。

aws_comprehend_detect_sentiment

此函數會將文字識別為具有正面、負面、中性或混合的情緒狀態。此函數的參考文件如下。

```
aws_comprehend_detect_sentiment(
  input_text,
  language_code
  [,max_batch_size]
)
```

若要進一步了解，請參閱《Amazon 開發人員指南》中的情緒。

aws_comprehend_detect_sentiment_confidence

此函數會測量針對指定文字偵測到之情緒的信賴等級。它會傳回一值 (類型 double)，指出由 aws_comprehend_detect_sentiment 函數指派給文字的情緒信賴度。信賴度是介於 0 和 1 之間的統計指標。信賴等級越高，您可以給與結果的權重就越多。函數文件的摘要如下。

```
aws_comprehend_detect_sentiment_confidence(
  input_text,
  language_code
  [,max_batch_size]
)
```

在這兩個函數
(aws_comprehend_detect_sentiment_confidence、aws_comprehend_detect_sentiment)
中，max_batch_size 會使用預設值 25，如果未指定任何值的話。批次大小應該始終大於 0。您可以使用 max_batch_size 來調整 Amazon Comprehend 函數呼叫的效能。較大的批次大小犧牲較快的效能，以換取 Aurora MySQL 資料庫叢集上較大的記憶體用量。如需詳細資訊，請參閱搭配 Aurora MySQL 使用 Aurora Machine Learning 的效能考量。

如需 Amazon Comprehend 中情緒偵測函數的參數和傳回類型的詳細資訊，請參閱 DetectSentiment

Example範例：使用 Amazon Comprehend 的簡單查詢

以下是簡單查詢的範例，該查詢會叫用這兩個函數，以查看您的客戶對支援團隊的滿意程度。假設您有一個資料庫資料表 (support)，其會在每次請求協助之後存放客戶意見回饋。這個範例查詢會將兩個內建函數套用至資料表的 feedback 資料欄中的文字，並輸出結果。函數傳回的信賴度值為介於 0.0 和 1.0 之間的倍準數。為了獲得更易讀取的輸出，此查詢將結果四捨五入為 6 個小數點。為了更輕鬆地進行比較，這個查詢也會先從具有最高信賴度的結果中，以遞減順序排序結果。

```
SELECT feedback AS 'Customer feedback',
    aws_comprehend_detect_sentiment(feedback, 'en') AS Sentiment,
    ROUND(aws_comprehend_detect_sentiment_confidence(feedback, 'en'), 6)
```

```
    AS Confidence FROM support
    ORDER BY Confidence DESC;
+-----------------------------------------------------------+-----------+------------+
| Customer feedback                                         | Sentiment | Confidence |
+-----------------------------------------------------------+-----------+------------+
| Thank you for the excellent customer support!             | POSITIVE  |   0.999771 |
| The latest version of this product stinks!                | NEGATIVE  |   0.999184 |
| Your support team is just awesome! I am blown away.       | POSITIVE  |   0.997774 |
| Your product is too complex, but your support is great.   | MIXED     |   0.957958 |
| Your support tech helped me in fifteen minutes.           | POSITIVE  |   0.949491 |
| My problem was never resolved!                            | NEGATIVE  |   0.920644 |
| When will the new version of this product be released?    | NEUTRAL   |   0.902706 |
| I cannot stand that chatbot.                              | NEGATIVE  |   0.895219 |
| Your support tech talked down to me.                      | NEGATIVE  |   0.868598 |
| It took me way too long to get a real person.             | NEGATIVE  |   0.481805 |
+-----------------------------------------------------------+-----------+------------+
 10 rows in set (0.1898 sec)
```

Example範例：判斷文字高於特定信賴等級的平均情緒

一般 Amazon Comprehend 查詢尋找情緒等於特定值且信心水準大於特定數字的列。例如，下列查詢顯示如何判斷資料庫中文件的平均情緒。此查詢只考量評估信心至少達 80% 的文件。

```
SELECT AVG(CASE aws_comprehend_detect_sentiment(productTable.document, 'en')
  WHEN 'POSITIVE' THEN 1.0
  WHEN 'NEGATIVE' THEN -1.0
  ELSE 0.0 END) AS avg_sentiment, COUNT(*) AS total
FROM productTable
WHERE productTable.productCode = 1302 AND
    aws_comprehend_detect_sentiment_confidence(productTable.document, 'en') >= 0.80;
```

搭配 Aurora MySQL 資料庫叢集使用 SageMaker AI

若要從 Aurora MySQL 資料庫叢集使用 SageMaker AI 功能，您需要建立預存函數，將呼叫嵌入至 SageMaker AI 端點及其推論功能。您通常會使用 MySQL 的 `CREATE FUNCTION` 執行此操作，其方式與您針對 Aurora MySQL 資料庫叢集上其他處理任務所做的方式相同。

若要使用於 SageMaker AI 部署的模型來推論，請使用您在預存函數中 MySQL 資料定義語言 (DDL) 陳述式，建立使用者定義的函數。每個預存函數代表託管模型的 SageMaker AI 端點。當您定義這類函數時，您指定模型的輸入參數、要調用的特定 SageMaker AI 端點，以及傳回類型。將模型套用至輸入參數後，此函數會傳回 SageMaker AI 端點計算的推論。

所有 Aurora Machine Learning 預存函數傳回數值類型或 VARCHAR。您可以使用 BIT 除外的任何數值類型。不允許其他類型，例如 JSON、BLOB、TEXT 和 DATE。

下列範例顯示使用 SageMaker AI 的 CREATE FUNCTION 語法。

```
CREATE FUNCTION function_name (
     arg1 type1,
     arg2 type2, ...)
     [DEFINER = user]
  RETURNS mysql_type
  [SQL SECURITY { DEFINER | INVOKER } ]
  ALIAS AWS_SAGEMAKER_INVOKE_ENDPOINT
  ENDPOINT NAME 'endpoint_name'
  [MAX_BATCH_SIZE max_batch_size];
```

這是一般 CREATE FUNCTION DDL 陳述式的延伸。在定義 SageMaker AI 函數的 CREATE FUNCTION 陳述式中，您不指定函數主體。反之，在通常出現函數主體的地方，您指定關鍵字 ALIAS。目前，對於此延伸語法，Aurora Machine Learning 僅支援 aws_sagemaker_invoke_endpoint。您必須指定 endpoint_name 參數。針對每個模型，SageMaker AI 端點可能有不同的特性。

> **Note**
>
> 如需 CREATE FUNCTION 的詳細資訊，請參閱《MySQL 8.0 參考手冊》中的 CREATE PROCEDURE 和 CREATE FUNCTION 陳述式。

max_batch_size 為選用參數。根據預設，批次大小上限為 10,000。您可以在函數中使用此參數，來限制 SageMaker AI 批次請求中處理的輸入數目上限。max_batch_size 參數有助於避免太大的輸入所造成的錯誤，或讓 SageMaker AI 更快傳回回應。此參數會影響 SageMaker AI 請求處理使用的內部緩衝區大小。指定太大的 max_batch_size 值可能造成資料庫執行個體耗用大量記憶體。

建議將 MANIFEST 設定保留為預設值 OFF。雖然您可以使用 MANIFEST ON 選項，但有些 SageMaker AI 功能無法直接使用以此選項匯出的 CSV。資訊清單格式與 SageMaker AI 預期的資訊清單格式不相容。

您為每個 SageMaker AI 模型建立個別的預存函數。因為端點與特定模型相關聯，而且每個模型接受的參數不同，函數至模型的這種對應有其必要。針對模型輸入和模型輸出類型使用 SQL 類型，有助於避免在 AWS 服務之間來回傳遞資料的類型轉換錯誤。您可以控制誰可以套用模型。您也可以指定參數來代表批次大小上限，以控制執行時間特性。

目前，所有 Aurora Machine Learning 函數都有 NOT DETERMINISTIC 屬性。如果您不明確指定屬性，Aurora 會自動設定 NOT DETERMINISTIC。有此需求是因為 SageMaker AI 模型可能變更，但完全不通知資料庫。如果發生此狀況，在單一交易內呼叫 Aurora Machine Learning 函數時，即使是相同輸入，也可能傳回不同結果。

在 CONTAINS SQL 陳述式中，您不能使用特性 NO SQL、READS SQL DATA、MODIFIES SQL DATA 或 CREATE FUNCTION。

以下是調用 SageMaker AI 端點來偵測異常的使用範例。有一個 SageMaker AI 端點 random-cut-forest-model。對應的模型已由 random-cut-forest 演算法訓練。對於每個輸入，模型傳回異常分數。此範例顯示資料點的分數比平均分數還大 3 個標準差 (大約第 99.9 百分位數)。

```
CREATE FUNCTION anomaly_score(value real) returns real
  alias aws_sagemaker_invoke_endpoint endpoint name 'random-cut-forest-model-demo';

set @score_cutoff = (select avg(anomaly_score(value)) + 3 * std(anomaly_score(value))
 from nyc_taxi);

select *, anomaly_detection(value) score from nyc_taxi
  where anomaly_detection(value) > @score_cutoff;
```

傳回字串的 SageMaker AI 函數的字元集需求

對於傳回字串值的 SageMaker AI 函數，建議指定字元集 utf8mb4 當作傳回類型。如果這不切實際，請對傳回類型使用夠大的字串長度，以保存 utf8mb4 字元集所代表的值。下列範例顯示如何為函數宣告 utf8mb4 字元集。

```
CREATE FUNCTION my_ml_func(...) RETURNS VARCHAR(5) CHARSET utf8mb4 ALIAS ...
```

目前，每個會傳回字串的 SageMaker AI 函數都以 utf8mb4 作為傳回值的字元集。即使 SageMaker AI 函數隱含或明確宣告其傳回類型為不同字元集，傳回值還是使用此字元集。如果 SageMaker AI 函數宣告傳回值為不同字元集，假設傳回的資料存放在不夠長的資料表欄，則可能自動截斷。例如，搭配 DISTINCT 子句的查詢會導致建立臨時資料表。因此，由於查詢期間內部處理字串的方式，SageMaker AI 函數結果可能截斷。

將資料匯出至 Amazon S3 進行 SageMaker AI 模型訓練 (進階)

我們建議您使用一些提供的演算法，開始使用 Aurora 機器學習和 SageMaker AI，並建議團隊中的資料科學家會為您提供可與 SQL 程式碼搭配使用的 SageMaker AI 端點。在下文中，您可以找到有關將

自己的 Amazon S3 儲存貯體與您自己的 SageMaker AI 模型和 Aurora MySQL 資料庫叢集搭配使用的基本資訊。

機器學習包括兩個主要步驟：訓練和推論。如要訓練 SageMaker AI 模型，您可以將資料匯出到 Amazon S3 儲存貯體。部署模型之前，Jupyter SageMaker AI 筆記本執行個體使用 Amazon S3 儲存貯體來訓練模型。您可以使用 SELECT INTO OUTFILE S3 陳述式從 Aurora MySQL 資料庫叢集查詢資料，然後將資料直接儲存至 Amazon S3 儲存貯體中存放的文字檔案。然後，筆記本執行個體從 Amazon S3 儲存貯體取用資料進行訓練。

Aurora Machine Learning 延伸 Aurora MySQL 中現有的 SELECT INTO OUTFILE 語法，將資料匯出為 CSV 格式。在訓練用途上需要此格式的模型，可以直接取用產生的 CSV 檔案。

```
SELECT * INTO OUTFILE S3 's3_uri' [FORMAT {CSV|TEXT} [HEADER]] FROM table_name;
```

此延伸支援標準 CSV 格式。

- 格式 TEXT 與現有的 MySQL 匯出格式相同。此為預設格式。
- 格式 CSV 是新推出的格式，遵守 RFC-4180 的規格。
- 如果您指定選用的關鍵字 HEADER，則輸出檔包含一個標題列。標題列的標籤對應於 SELECT 陳述式中的欄名稱。
- 您仍可使用關鍵字 CSV 和 HEADER 當作識別符。

SELECT INTO 延伸的語法和文法現在如下所示：

```
INTO OUTFILE S3 's3_uri'
[CHARACTER SET charset_name]
[FORMAT {CSV|TEXT} [HEADER]]
[{FIELDS | COLUMNS}
  [TERMINATED BY 'string']
  [[OPTIONALLY] ENCLOSED BY 'char']
  [ESCAPED BY 'char']
]
[LINES
  [STARTING BY 'string']
  [TERMINATED BY 'string']
]
```

搭配 Aurora MySQL 使用 Aurora Machine Learning 的效能考量

當透過 Aurora 機器學習函數調用時，Amazon Bedrock、Amazon Comprehend 和 SageMaker AI 服務會執行大部分工作。這表示您可以視需要獨立擴展這些資源。對於您的 Aurora MySQL 資料庫叢集，您可以盡可能有效地進行函數呼叫。接下來，您可以找到使用 Aurora Machine Learning 時需要注意的一些效能考量。

模型和提示

使用 Amazon Bedrock 時的效能高度取決於您所使用的模型和提示。選擇最適合您使用案例的模型和提示。

查詢快取

Aurora MySQL 查詢快取不適用於 Aurora Machine Learning 函數。Aurora MySQL 不將查詢結果存放在查詢快取中任何呼叫 Aurora Machine Learning 函數的 SQL 陳述式之中。

Aurora Machine Learning 函數呼叫的批次最佳化

您從 Aurora 叢集可影響的主要 Aurora Machine Learning 效能方面，是 Aurora Machine Learning 預存函數呼叫的批次模式設定。機器學習函數通常需要大量的額外負荷，因此為每個資料列單獨呼叫外部服務並不實用。Aurora Machine Learning 可以將許多資料列對外部 Aurora Machine Learning 服務的呼叫合併成單一批次，來減少這項額外負荷。Aurora Machine Learning 會收到所有輸入資料列的回應，然後一次一列將回應傳給執行中的查詢。此最佳化改善 Aurora 查詢的傳輸量和延遲，而不變更結果。

建立會連線至 SageMaker AI 端點的 Aurora 預存函數時，您會定義批次大小參數。此參數影響每次在幕後呼叫 SageMaker AI 時傳輸多少列。如果查詢處理許多列，針對每一列來個別呼叫 SageMaker AI 會產生極大的額外負荷。預存程序處理的資料集越大，批次大小就應該越大。

如果批次模式最佳化可套用至 SageMaker AI 函數，您可以檢查 `EXPLAIN PLAN` 陳述式產生的查詢計劃來辨別。在此情況下，執行計劃的 `extra` 欄包含 `Batched machine learning`。下列範例顯示對 SageMaker AI 函數的呼叫 (使用批次模式)。

```
mysql> CREATE FUNCTION anomaly_score(val real) returns real alias
 aws_sagemaker_invoke_endpoint endpoint name 'my-rcf-model-20191126';
Query OK, 0 rows affected (0.01 sec)

mysql> explain select timestamp, value, anomaly_score(value) from nyc_taxi;
```

```
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+-------------------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len |
 ref  | rows | filtered | Extra                     |
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+-------------------------+
|  1 | SIMPLE      | nyc_taxi | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL |   48 |   100.00 | Batched machine learning |
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+-------------------------+
1 row in set, 1 warning (0.01 sec)
```

當您呼叫其中一個內建的 Amazon Comprehend 函數時，您可以指定選用的 `max_batch_size` 參數，以控制批次大小。此參數限制每個批次中處理的 `input_text` 值數量上限。一次傳送多個項目可減少 Aurora 與 Amazon Comprehend 之間的來回次數。在查詢搭配 `LIMIT` 子句的情況下，限制批次大小很有用。`max_batch_size` 使用較小的值可避免叫用 Amazon Comprehend 的次數超過您輸入文字的次數。

評估 Aurora Machine Learning 函數時的批次最佳化適用於下列情況：

- 在 select 清單或 `SELECT` 陳述式的 `WHERE` 子句內的函數呼叫
- 在 `INSERT` 和 `REPLACE` 陳述式 `VALUES` 清單中的函數呼叫
- 在 `UPDATE` 陳述式的 `SET` 值中的 SageMaker AI 函數：

```
INSERT INTO MY_TABLE (col1, col2, col3) VALUES
  (ML_FUNC(1), ML_FUNC(2), ML_FUNC(3)),
  (ML_FUNC(4), ML_FUNC(5), ML_FUNC(6));
UPDATE MY_TABLE SET col1 = ML_FUNC(col2), SET col3 = ML_FUNC(col4) WHERE ...;
```

監控 Aurora Machine Learning

如下範例所示，您可以查詢數個全域變數來監控 Aurora 機器學習批次作業。

```
show status like 'Aurora_ml%';
```

您可以使用 `FLUSH STATUS` 陳述式來重設狀態變數。因此，所有數據代表上次重設變數以來的總計、平均等。

`Aurora_ml_logical_request_cnt`

自上次重新設定狀態後，資料庫執行個體已評估傳送至 Aurora 機器學習服務的邏輯請求數目。視是否使用批次處理而定，此值可能會大於 `Aurora_ml_actual_request_cnt`。

`Aurora_ml_logical_response_cnt`

Aurora MySQL 從 Aurora Machine Learning 服務接收的彙總回應計數，涵蓋資料庫執行個體的使用者執行的所有查詢。

`Aurora_ml_actual_request_cnt`

Aurora MySQL 從 Aurora 機器學習服務接收的彙總請求計數，涵蓋資料庫執行個體的使用者執行的所有查詢。

`Aurora_ml_actual_response_cnt`

Aurora MySQL 從 Aurora Machine Learning 服務接收的彙總回應計數，涵蓋資料庫執行個體的使用者執行的所有查詢。

`Aurora_ml_cache_hit_cnt`

Aurora MySQL 從 Aurora Machine Learning 服務接收的彙總內部快取命中計數，涵蓋資料庫執行個體的使用者執行的所有查詢。

`Aurora_ml_retry_request_cnt`

自上次重新設定狀態後，資料庫執行個體傳送至 Aurora 機器學習服務的重試請求數目。

`Aurora_ml_single_request_cnt`

由非批次模式評估的 Aurora Machine Learning 函數的彙總計數，涵蓋資料庫執行個體的使用者執行的所有查詢。

如需詳細資訊，了解如何監控從 Aurora 機器學習函數呼叫之 SageMaker AI 操作的效能，請參閱監控 Amazon SageMaker AI。

將 Amazon Aurora Machine Learning 與 Aurora PostgreSQL 搭配使用

透過將 Amazon Aurora 機器學習與 Aurora PostgreSQL 資料庫叢集搭配使用，您可以根據需求使用 Amazon Comprehend 或 Amazon SageMaker AI 或 Amazon Bedrock。這些服務各支援特定的機器學習使用案例。

某些 AWS 區域 和特定版本的 Aurora PostgreSQL 僅支援 Aurora Machine Learning。在嘗試設定 Aurora Machine Learning 之前，請先檢查 Aurora PostgreSQL 版本和區域的可用性。如需詳細資訊，請參閱將機器學習與 Aurora PostgreSQL 搭配使用。

主題

- 將 Aurora Machine Learning 與 Aurora PostgreSQL 搭配使用的建議
- Aurora Machine Learning 搭配 Aurora PostgreSQL 支援的功能和限制
- 設定 Aurora PostgreSQL 資料庫叢集來使用 Aurora Machine Learning
- 搭配 Aurora PostgreSQL 資料庫叢集使用 Amazon Bedrock
- 搭配 Aurora PostgreSQL 資料庫叢集使用 Amazon Comprehend
- 搭配 Aurora PostgreSQL 資料庫叢集使用 SageMaker AI
- 將資料匯出至 Amazon S3 進行 SageMaker AI 模型訓練 (進階)
- 搭配 Aurora PostgreSQL 使用 Aurora Machine Learning 的效能考量
- 監控 Aurora Machine Learning

將 Aurora Machine Learning 與 Aurora PostgreSQL 搭配使用的建議

AWS 機器學習服務是在自己的生產環境中設定和執行的受管服務。Aurora 機器學習支援與 Amazon Comprehend、SageMaker AI 和 Amazon Bedrock 的整合。在嘗試設定 Aurora PostgreSQL 資料庫叢集以使用 Aurora Machine Learning 之前，請務必了解下列要求和先決條件。

- Amazon Comprehend、SageMaker AI 和 Amazon Bedrock 服務必須在與 Aurora PostgreSQL 資料庫叢集 AWS 區域 相同的 中執行。您無法從不同區域的 Aurora PostgreSQL 資料庫叢集使用 Amazon Comprehend 或 SageMaker AI 或 Amazon Bedrock 服務。
- 如果 Aurora PostgreSQL 資料庫叢集位於以 Amazon VPC 服務為基礎的虛擬公有雲端 (VPC) (而非 Amazon Comprehend 和 SageMaker AI 服務)，則 VPC 的安全群組需要允許目標 Aurora 機器學習服務的輸出連線。如需詳細資訊，請參閱啟用從 Amazon Aurora 到其他服務的網路通訊 AWS。
- 對於 SageMaker AI，您要用於推論的機器學習元件必須已設定並準備好使用。在設定 Aurora PostgreSQL 資料庫叢集的過程中，您必須具有 SageMaker AI 端點的 Amazon Resource Name (ARN) 可供使用。您團隊中的資料科學家可能最能夠處理如何使用 SageMaker，以準備模型並處理其他這類任務。若要開始使用 Amazon SageMaker AI，請參閱開始使用 Amazon SageMaker AI。如需推論和端點的詳細資訊，請參閱即時推論。
- 對於 Amazon Bedrock，您需要擁有 Bedrock 模型的模型 ID，您想要將此模型用於 Aurora PostgreSQL 資料庫叢集組態程序期間的可用推論。團隊中的資料科學家可能最適合使用 Bedrock 來

決定要使用的模型，並視需要微調模型，以及處理其他此類任務。若要開始使用 Amazon Bedrock，請參閱如何設定 Bedrock。

- Amazon Bedrock 使用者必須先要求模型存取權，才能使用模型。如果您想要新增用於文字、聊天和影像產生的其他模型，則需要請求存取 Amazon Bedrock 中的模型。如需詳細資訊，請參閱模型存取。

Aurora Machine Learning 搭配 Aurora PostgreSQL 支援的功能和限制

Aurora 機器學習支援任何可透過 `ContentType` 的 `text/csv` 值讀取和寫入逗號分隔值 (CSV) 格式的 SageMaker AI 端點。目前接受此格式的內建 SageMaker AI 演算法如下。

- Linear Learner
- Random Cut Forest
- XGBoost

若要進一步了解這些演算法，請參閱《Amazon SageMaker AI 開發人員指南》中的選擇演算法。

將 Amazon Bedrock 與 Aurora 機器學習搭配使用時，適用下列限制：

- 使用者定義的函數 (UDF) 提供與 Amazon Bedrock 互動的原生方式。UDF 沒有特定的請求或回應需求，因此可以使用任何模型。
- 您可以使用 UDF 來建置任何所需的工作流程。例如，您可以結合類似 `pg_cron` 的基本概念，來執行查詢、擷取資料、產生推論，以及寫入至資料表以直接提供查詢。
- UDF 不支援批次或平行呼叫。
- Aurora 機器學習延伸模組不支援向量界面。作為延伸模組的一部分，函數可用於輸出模型回應的嵌入 (格式為 `float8[]`)，以將這些嵌入存放在 Aurora 中。如需 `float8[]` 使用的詳細資訊，請參閱搭配 Aurora PostgreSQL 資料庫叢集使用 Amazon Bedrock。

設定 Aurora PostgreSQL 資料庫叢集來使用 Aurora Machine Learning

若要讓 Aurora Machine Learning 與您的 Aurora PostgreSQL 資料庫叢集搭配使用，您需要為要使用的每個服務建立 AWS Identity and Access Management (IAM) 角色。IAM 角色可讓您的 Aurora PostgreSQL 資料庫叢集代表叢集使用 Aurora Machine Learning 服務。您也需要安裝 Aurora Machine Learning 延伸模組。在下列主題中，您可以找到其中每個 Aurora Machine Learning 服務的設定程序。

主題

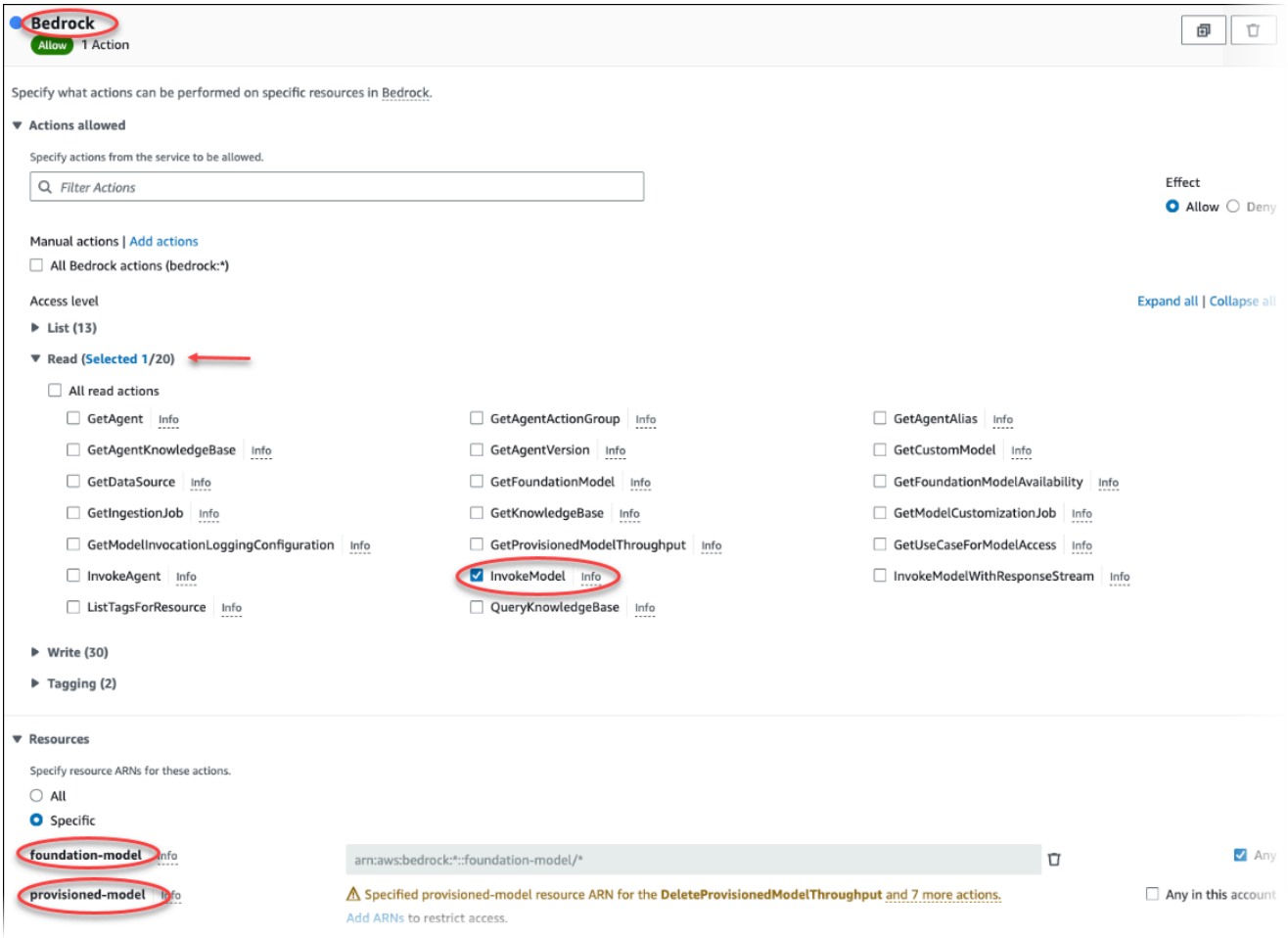
- 設定 Aurora PostgreSQL 來使用 Amazon Bedrock
- 設定 Aurora PostgreSQL 來使用 Amazon Comprehend
- 設定 Aurora PostgreSQL 來使用 Amazon SageMaker AI
  - 設定 Aurora PostgreSQL 來使用 Amazon S3 for SageMaker AI (進階)
- 安裝 Aurora Machine Learning 延伸模組

設定 Aurora PostgreSQL 來使用 Amazon Bedrock

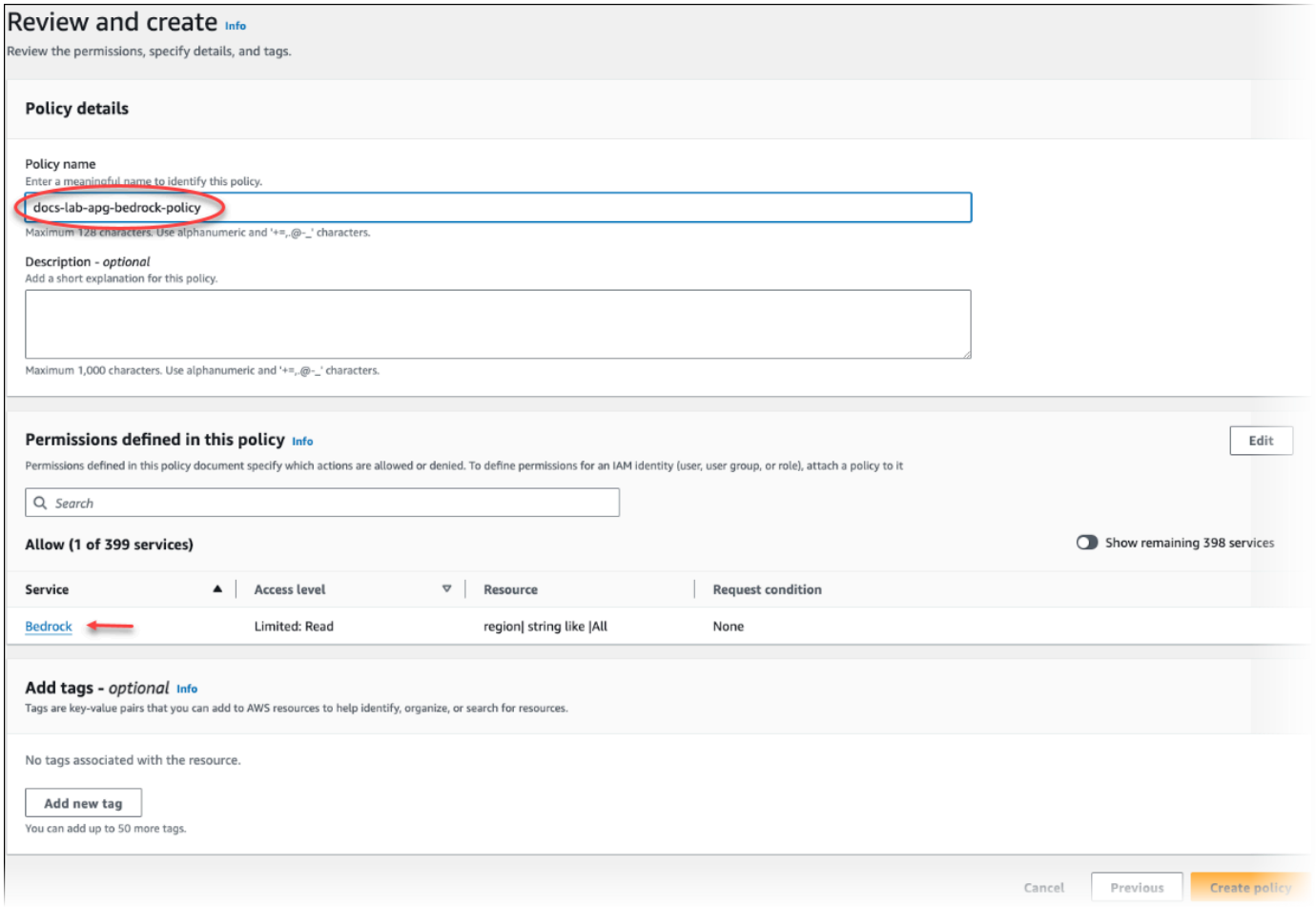
在以下程序中，您首先建立 IAM 角色和政策，授與 Aurora PostgreSQL 許可來代表叢集使用 Amazon Bedrock。然後，您將此政策連接至 IAM 角色，Aurora PostgreSQL 資料庫叢集會使用此角色來使用 Amazon Bedrock。為了簡單起見，此程序使用 AWS 管理主控台 來完成所有任務。

設定 Aurora PostgreSQL 資料庫叢集來使用 Amazon Bedrock

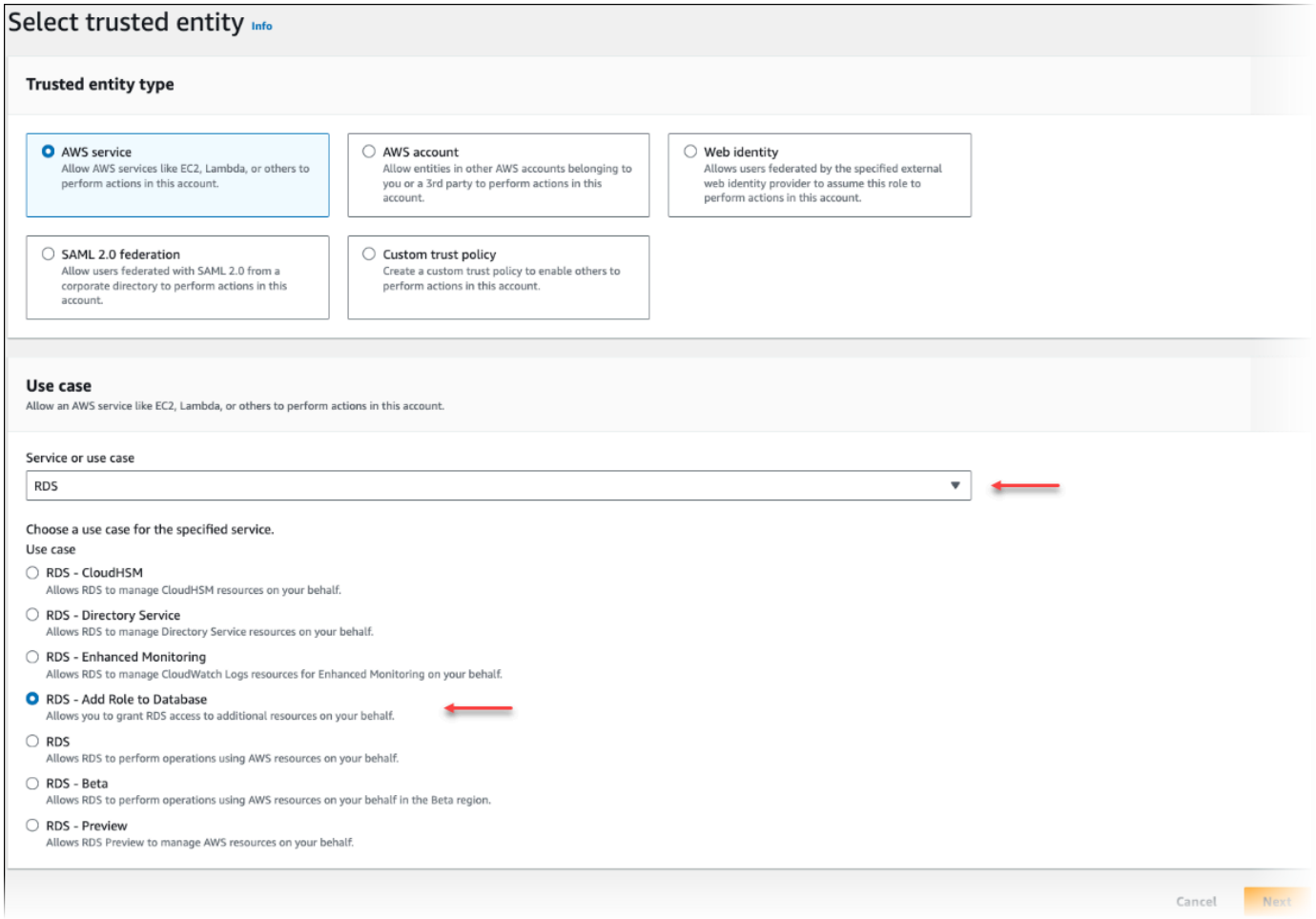
1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
3. 在 (IAM) 主控台功能表中選擇政策 AWS Identity and Access Management （ 在存取管理下 ）。
   a. 選擇建立政策。在視覺化編輯器頁面中，選擇服務，然後在選取服務欄位中輸入 Bedrock。展開讀取存取層級。從 Amazon Bedrock 讀取設定中選擇 InvokeModel。
   b. 選擇您想要透過政策授予讀取存取權的基礎/佈建模型。



4. 選擇 Next: Tags (下一步：標籤) 並定義任何標籤 (這是選用的)。選擇下一步：檢閱。輸入政策的名稱和描述，如圖所示。

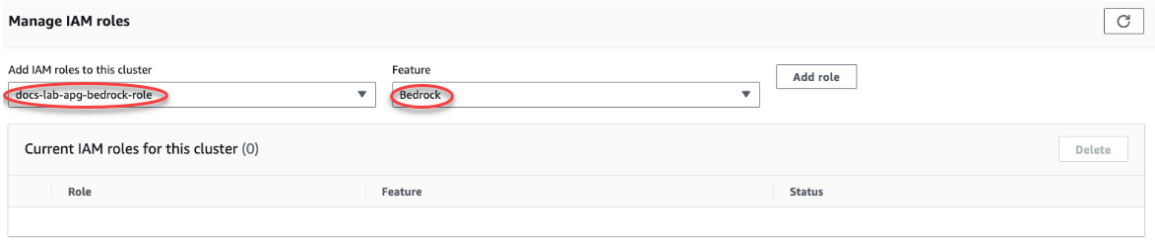


5. 選擇建立政策。儲存策略後，主控台即會顯示警示。您可以在政策清單中找到它。
6. 在 IAM 主控台功能表上選擇 Roles (角色) (在 Access management (存取管理) 之下)。
7. 選擇建立角色。
8. 在 Select trusted entity (選取信任的實體) 頁面上，選擇 AWS service (AWS 服務) 磚，然後選擇 RDS 以開啟選擇器。
9. 選擇 RDS – Add Role to Database (RDS - 將角色新增至資料庫)。



10. 選擇下一步。在 Add permissions (新增許可) 頁面上，尋找您已在先前步驟中建立的政策，並從列出的政策之中進行選擇。選擇下一步。
11. Next: Review (下一步：檢閱)。輸入 IAM 角色的名稱和描述。
12. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
13. 導覽至 AWS 區域 Aurora PostgreSQL 資料庫叢集所在的 。
14. 在導覽窗格中選擇資料庫，然後選擇您要與 Bedrock 搭配使用的 Aurora PostgreSQL 資料庫叢集。
15. 選擇 Connectivity & security (連線和安全) 索引標籤，並捲動至頁面的 Manage IAM roles (管理 IAM 角色) 區段。從 Add IAM roles to this cluster (將 IAM 角色新增至此叢集) 選擇器中，選擇您已在先前步驟中建立的角色。在功能選擇器中，選擇 Bedrock，然後選擇新增角色。

角色 (及其政策) 即會與 Aurora PostgreSQL 資料庫叢集相關聯。程序完成時，角色會列示在此叢集清單的目前 AM 角色中，如下所示。



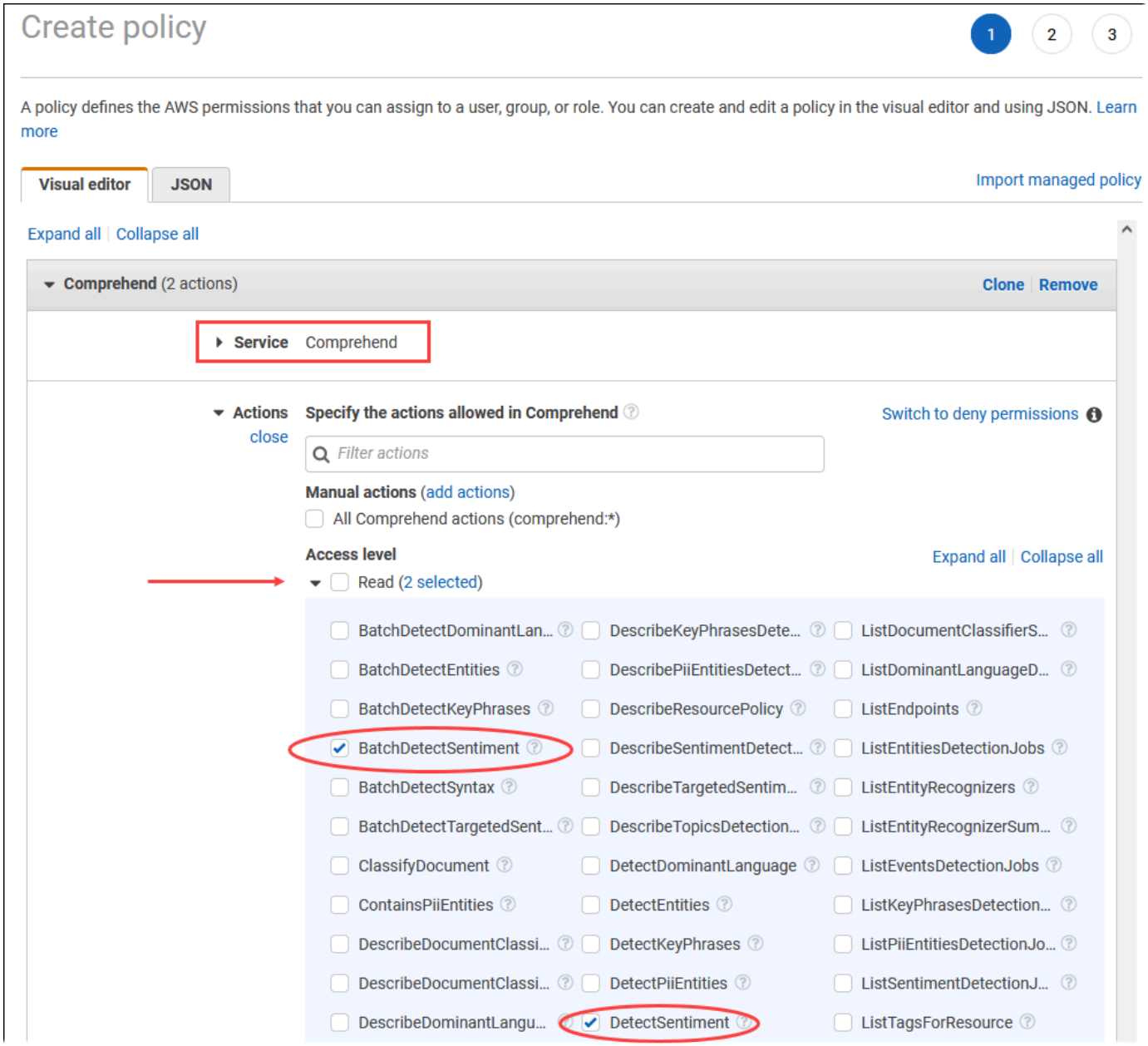
Amazon Bedrock 的 IAM 設定已完成。透過安裝延伸模組，繼續設定 Aurora PostgreSQL 以使用 Aurora Machine Learning，如安裝 Aurora Machine Learning 延伸模組中所述

設定 Aurora PostgreSQL 來使用 Amazon Comprehend

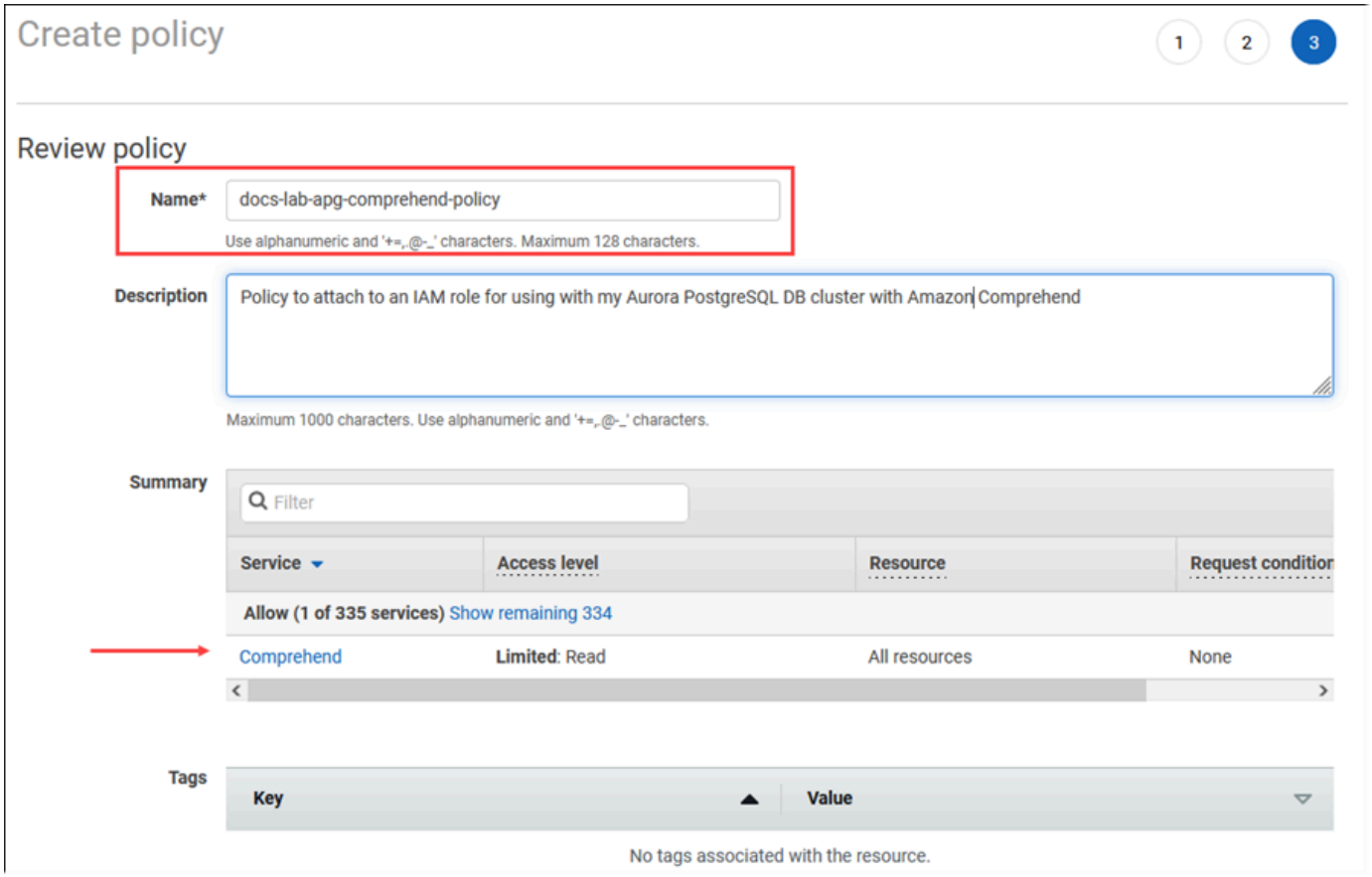
在以下程序中，您首先建立 IAM 角色和政策，授與 Aurora PostgreSQL 許可來代表叢集使用 Amazon Comprehend。然後，您可以將該政策附加到 Aurora PostgreSQL 資料庫叢集用來使用 Amazon Comprehend 的 IAM 角色，為了簡單起見，此程序會使用 AWS 管理主控台 來完成所有任務。

設定您的 Aurora PostgreSQL 資料庫叢集來使用 Amazon Comprehend

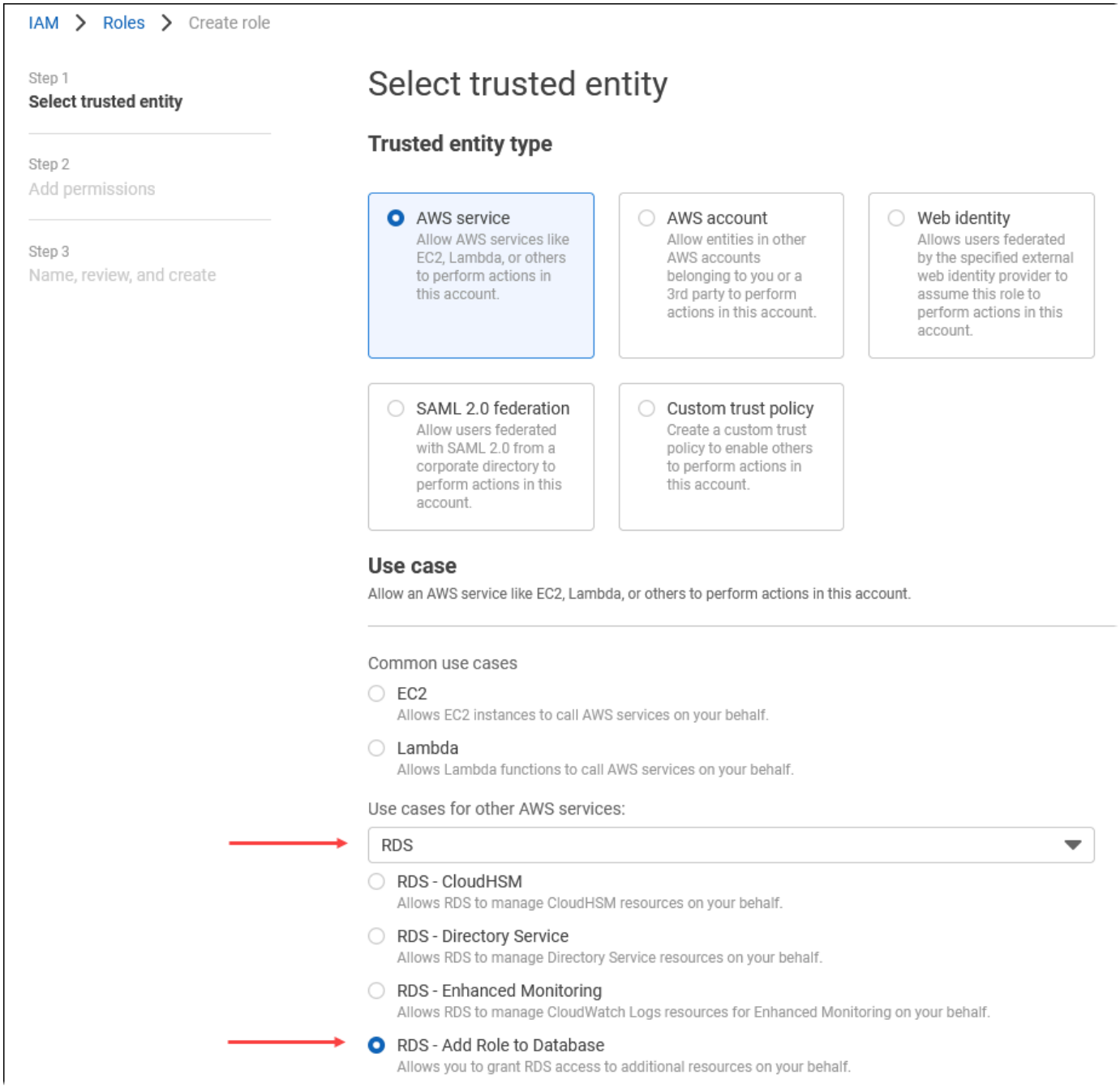
1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
3. 在 (IAM) 主控台功能表中選擇政策 AWS Identity and Access Management （ 在存取管理下 ）。



4. 選擇建立政策。在視覺化編輯器頁面中，選擇 Service (服務)，然後在 Select a service (選取服務) 欄位中輸入 Comprehend。展開讀取存取層級。從 Amazon Comprehend 讀取設定中選擇 BatchDetectSentiment 和 DetectSentiment 。
5. 選擇 Next: Tags (下一步：標籤) 並定義任何標籤 (這是選用的)。選擇下一步：檢閱。輸入政策的名稱和描述，如圖所示。



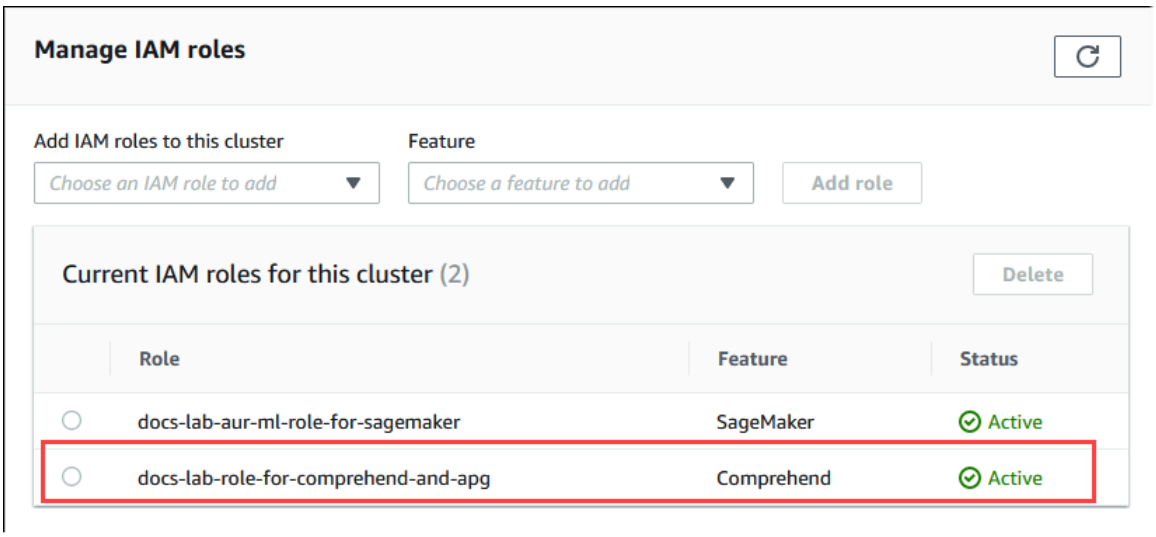
6. 選擇建立政策。儲存策略後，主控台即會顯示警示。您可以在政策清單中找到它。
7. 在 IAM 主控台功能表上選擇 Roles (角色) (在 Access management (存取管理) 之下)。
8. 選擇建立角色。
9. 在 Select trusted entity (選取信任的實體) 頁面上，選擇 AWS service (AWS 服務) 磚，然後選擇 RDS 以開啟選擇器。
10. 選擇 RDS – Add Role to Database (RDS - 將角色新增至資料庫)。



11. 選擇下一步。在 Add permissions (新增許可) 頁面上，尋找您已在先前步驟中建立的政策，並從列出的政策之中進行選擇。選擇 Next (下一步)
12. Next: Review (下一步：檢閱)。輸入 IAM 角色的名稱和描述。
13. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
14. 導覽至 AWS 區域 Aurora PostgreSQL 資料庫叢集所在的 。
15. 在導覽窗格中選擇 Databases (資料庫)，然後選擇您要與 Amazon Comprehend 搭配使用的 Aurora PostgreSQL 資料庫叢集。

16. 選擇 Connectivity & security (連線和安全) 索引標籤，並捲動至頁面的 Manage IAM roles (管理 IAM 角色) 區段。從 Add IAM roles to this cluster (將 IAM 角色新增至此叢集) 選擇器中，選擇您已在先前步驟中建立的角色。在功能選擇器中，選擇 Comprehend，然後選擇新增角色。

角色 (及其政策) 即會與 Aurora PostgreSQL 資料庫叢集相關聯。程序完成時，角色會列示在此叢集清單的目前 AM 角色中，如下所示。



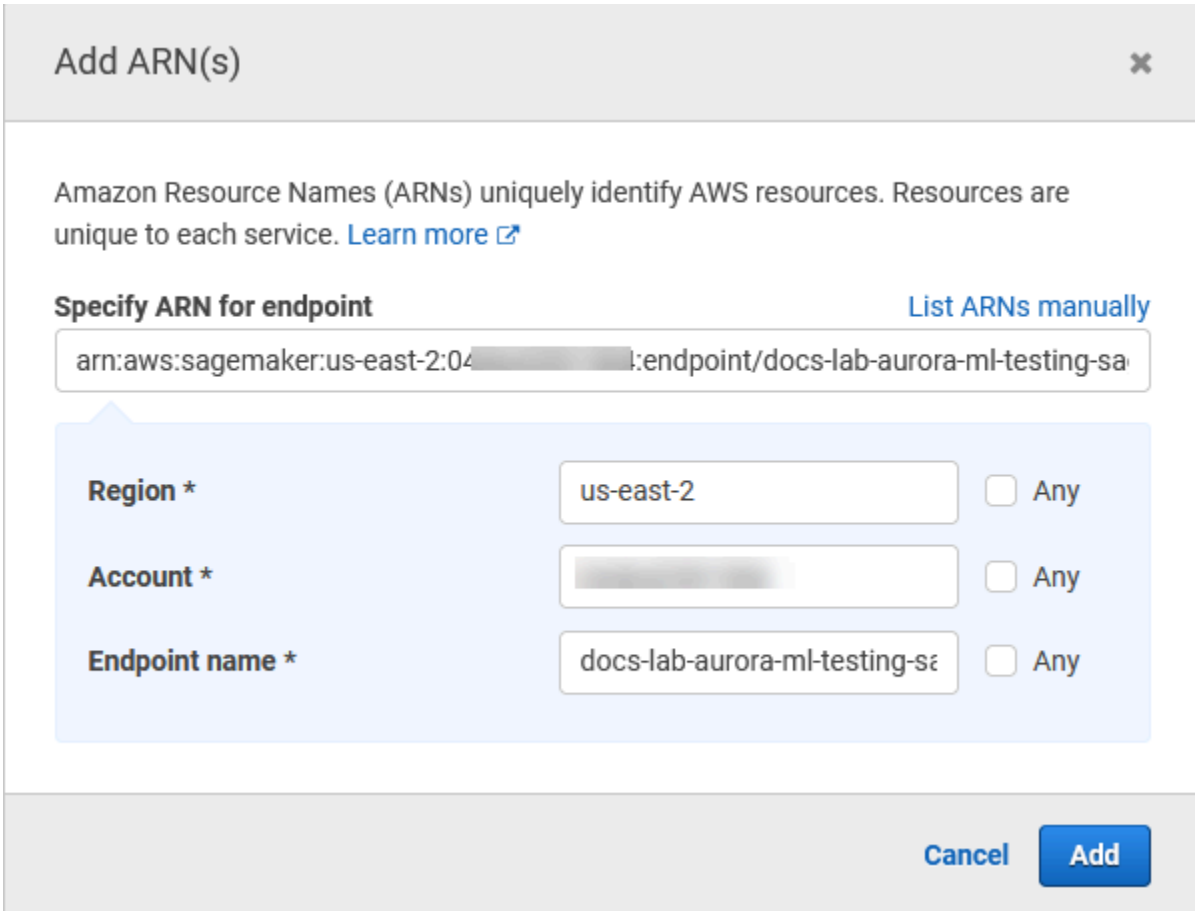
Amazon Comprehend 的 IAM 設定已完成。透過安裝延伸模組，繼續設定 Aurora PostgreSQL 以使用 Aurora Machine Learning，如安裝 Aurora Machine Learning 延伸模組中所述

設定 Aurora PostgreSQL 來使用 Amazon SageMaker AI

在為 Aurora PostgreSQL 資料庫叢集建立 IAM 政策和角色之前，您必須先設定 SageMaker AI 模型，並讓端點可用。

設定 Aurora PostgreSQL 資料庫叢集來使用 SageMaker AI

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在 (IAM) 主控台功能表中選擇政策 AWS Identity and Access Management （ 在存取管理下 ），然後選擇建立政策。在視覺化編輯器中，針對服務選擇 SageMaker。針對動作，開啟讀取選擇器 (在存取層級下)，然後選擇 InvokeEndpoint。當執行此操作時，會顯示警告圖示。
3. 開啟資源選擇器，然後在 InvokeEndpoint 動作的 Specify endpoint resource ARN (指定端點資源 ARN) 下選擇Add ARN to restrict access (新增 ARN 以限制存取) 連結。
4. 輸入 AWS 區域 SageMaker AI 資源的 和端點的名稱。 AWS 您的帳戶已預先填入。



5. 選擇 Add (新增) 以儲存。選擇 Next: Tags (下一步：標籤) 和 Next: Review (下一步：檢閱)，以移至原則建立程序的最後一頁。
6. 輸入此政策的名稱和描述，然後選擇 Create policy (建立政策)。即時會建立策略，並將其新增至政策清單。發生這種情況時，您會在主控台中看到警示。
7. 在 IAM 主控台上，選擇 Roles (角色)。
8. 選擇建立角色。
9. 在 Select trusted entity (選取信任的實體) 頁面上，選擇 AWS service (AWS 服務) 磚，然後選擇 RDS 以開啟選擇器。
10. 選擇 RDS – Add Role to Database (RDS - 將角色新增至資料庫)。
11. 選擇下一步。在 Add permissions (新增許可) 頁面上，尋找您已在先前步驟中建立的政策，並從列出的政策之中進行選擇。選擇 Next (下一步)
12. Next: Review (下一步：檢閱)。輸入 IAM 角色的名稱和描述。
13. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
14. 導覽至 AWS 區域 Aurora PostgreSQL 資料庫叢集所在的 。

15. 在導覽窗格中選擇資料庫，然後選擇您要與 SageMaker AI 搭配使用的 Aurora PostgreSQL 資料庫叢集。
16. 選擇 Connectivity & security (連線和安全) 索引標籤，並捲動至頁面的 Manage IAM roles (管理 IAM 角色) 區段。從 Add IAM roles to this cluster (將 IAM 角色新增至此叢集) 選擇器中，選擇您已在先前步驟中建立的角色。在功能選擇器中，選擇 SageMaker AI，然後選擇新增角色。

角色 (及其政策) 即會與 Aurora PostgreSQL 資料庫叢集相關聯。程序完成時，角色會列示在此叢集清單的目前 AM 角色中。

SageMaker AI 的 IAM 設定已完成。透過安裝延伸模組，繼續設定 Aurora PostgreSQL 以使用 Aurora Machine Learning，如安裝 Aurora Machine Learning 延伸模組中所述。

設定 Aurora PostgreSQL 來使用 Amazon S3 for SageMaker AI (進階)

若要搭配您自己的模型使用 SageMaker AI，而不是使用 SageMaker AI 提供的預先建置元件，您需要設定 Amazon Simple Storage Service (Amazon S3) 儲存貯體，供 Aurora PostgreSQL 資料庫叢集使用。這是進階主題，並未在此《Amazon Aurora 使用者指南》中完整記載。一般程序與整合 SageMaker AI 支援相同，如下所示。

1. 為 Amazon S3 建立 IAM 政策和角色。
2. 在 Aurora PostgreSQL 資料庫叢集的 Connectivity & security (連線和安全) 索引標籤上，新增 IAM 角色和 Amazon S3 匯入或匯出做為功能。
3. 將角色的 ARN 新增至 Aurora 資料庫叢集的自訂資料庫叢集參數群組。

如需基本使用方式的資訊，請參閱 將資料匯出至 Amazon S3 進行 SageMaker AI 模型訓練 (進階)。

安裝 Aurora Machine Learning 延伸模組

Aurora 機器學習延伸模組 `aws_ml 1.0` 提供兩種函數，您可用於調用 Amazon Comprehend、SageMaker AI 服務，且 `aws_ml 2.0` 並提供兩種額外的函數，您可用於調用 Amazon Bedrock 服務。在 Aurora PostgreSQL 資料庫叢集上安裝這些延伸模組也會建立功能的管理角色。

> **Note**
>
> 使用這些功能取決於是否需要 Aurora 機器學習服務 (Amazon Comprehend、SageMaker AI、Amazon Bedrock) 完成 IAM 設定，如設定 Aurora PostgreSQL 資料庫叢集來使用 Aurora Machine Learning中所述。

- aws_comprehend.detect_sentiment – 您可以使用此函數，將情緒分析套用至存放在 Aurora PostgreSQL 資料庫叢集上資料庫中的文字。
- aws_sagemaker.invoke_endpoint：您可以在 SQL 程式碼中使用此函數，與來自叢集的 SageMaker AI 端點進行通訊。
- aws_bedrock.invoke_model：您可以在 SQL 程式碼中使用此函數，與來自叢集的 Bedrock 模型進行通訊。此函數的回應將採用 TEXT 格式，因此，如果模型以 JSON 內文格式回應，則此函數的輸出將以字串格式轉送給最終使用者。
- aws_bedrock.invoke_model_get_embeddings：您可以在 SQL 程式碼中使用此函數來調用 Bedrock 模型，以在 JSON 回應中傳回輸出嵌入。當您想要擷取直接與 json-key 相關聯的嵌入，以簡化任何自我管理工作流程的回應時，可以利用此功能。

在 Aurora PostgreSQL 資料庫叢集中安裝 Aurora Machine Learning 延伸模組

- 使用 `psql` 連線到 Aurora PostgreSQL 資料庫叢集的寫入器執行個體。連線到要在其中安裝 `aws_ml` 延伸模組的特定資料庫。

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=labdb
```

```
labdb=> CREATE EXTENSION IF NOT EXISTS aws_ml CASCADE;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
labdb=>
```

安裝 `aws_ml` 延伸模組也會建立 `aws_ml` 系統管理角色和三個新結構描述，如下所示。

- `aws_comprehend` – Amazon Comprehend 服務的結構描述和 `detect_sentiment` 函數的來源 (`aws_comprehend.detect_sentiment`)。
- `aws_sagemaker`：SageMaker AI 服務的結構描述和 `invoke_endpoint` 函數的來源 (`aws_sagemaker.invoke_endpoint`)。
- `aws_bedrock`：Amazon Bedrock 服務的結構描述和 `invoke_model(aws_bedrock.invoke_model)` 和 `invoke_model_get_embeddings(aws_bedrock.invoke_model_get_embeddings)` 函數的來源。

rds_superuser 角色獲授與 aws_ml 管理角色，並由這三個 Aurora 機器學習結構描述的 OWNER 組成。若要允許其他資料庫使用者存取 Aurora Machine Learning 函數，rds_superuser 需要授與 Aurora Machine Learning 函數的 EXECUTE 權限。根據預設，會在兩個 Aurora Machine Learning 結構描述中的函數上撤銷 PUBLIC 的 EXECUTE 權限。

在多租用戶資料庫組態中，您可以在要保護的特定 Aurora Machine Learning 結構描述上使用 REVOKE USAGE，防止租用戶存取 Aurora Machine Learning 函數。

搭配 Aurora PostgreSQL 資料庫叢集使用 Amazon Bedrock

對於 Aurora PostgreSQL，Aurora 機器學習提供下列 Amazon Bedrock 函數，用於與文字資料搭配使用。只有在您安裝 aws_ml 2.0 延伸模組並完成所有設定程序之後，才能使用此函數。如需詳細資訊，請參閱設定 Aurora PostgreSQL 資料庫叢集來使用 Aurora Machine Learning。

aws_bedrock.invoke_model

此函數採用 JSON 格式的文字作為輸入、處理 Amazon Bedrock 上託管的各種模型，以及從模型傳回 JSON 文字回應。此回應可能包含文字、影像或嵌入。函數文件的摘要如下。

```
aws_bedrock.invoke_model(
   IN model_id      varchar,
   IN content_type  text,
   IN accept_type   text,
   IN model_input   text,
   OUT model_output varchar)
```

此函數的輸入和輸出如下。

- model_id：模型的識別符。
- content_type：Bedrock 模型的請求類型。
- accept_type：從 Bedrock 模型預期的回應類型。通常適用於大多數模型的應用程式/JSON。
- model_input：提示；以 content_type 指定的格式對模型輸入的特定集合。如需模型接受之請求格式/結構的詳細資訊，請參閱基礎模型的推論參數。
- model_output：文字形式的 Bedrock 模型輸出。

下列範例示範如何使用 invoke_model 調用 Bedrock 的 Anthropic Claude 2 模型。

Example範例：使用 Amazon Bedrock 函數的簡單查詢

```
SELECT aws_bedrock.invoke_model (
    model_id    := 'anthropic.claude-v2',
    content_type:= 'application/json',
    accept_type := 'application/json',
    model_input := '{"prompt": "\n\nHuman: You are a helpful assistant that answers
 questions directly and only using the information provided in the context below.
\nDescribe the answer
    in detail.\n\nContext: %s \n\nQuestion: %s \n
\nAssistant:","max_tokens_to_sample":4096,"temperature":0.5,"top_k":250,"top_p":0.5,"stop_seque
[]}'
);
```

aws_bedrock.invoke_model_get_embeddings

在某些情況下，模型輸出可能會指向向量嵌入。由於每個模型的回應有所不同，可以利用另一個函數 invoke_model_get_embeddings，其運作方式與 invoke_model 完全相同，但會透過指定適當的 json-key 來輸出嵌入。

```
aws_bedrock.invoke_model_get_embeddings(
   IN model_id      varchar,
   IN content_type  text,
   IN json_key      text,
   IN model_input   text,
   OUT model_output float8[])
```

此函數的輸入和輸出如下。

- `model_id`：模型的識別符。
- `content_type`：Bedrock 模型的請求類型。在這裡，accept_type 會設為預設值 `application/json`。
- `model_input`：提示；以 content_type 指定的格式對模型輸入的特定集合。如需模型接受之請求格式/結構的詳細資訊，請參閱基礎模型的推論參數。
- `json_key`：參考欄位以從中擷取嵌入。如果嵌入模型有所變更，這可能會不同。
- `model_output`：Bedrock 模型的輸出，作為具有 16 位元十進位的嵌入陣列。

下列範例顯示如何針對 PostgreSQL I/O 監控檢視一詞使用 Titan 嵌入 G1 (文字嵌入模型) 產生嵌入。

Example範例：使用 Amazon Bedrock 函數的簡單查詢

```
SELECT aws_bedrock.invoke_model_get_embeddings(
   model_id      := 'amazon.titan-embed-text-v1',
   content_type  := 'application/json',
   json_key      := 'embedding',
   model_input   := '{ "inputText": "PostgreSQL I/O monitoring views"}') AS embedding;
```

搭配 Aurora PostgreSQL 資料庫叢集使用 Amazon Comprehend

對於 Aurora PostgreSQL，Aurora Machine Learning 提供下列 Amazon Comprehend 函數，用於使用您的文字資料。只有在您安裝 aws_ml 延伸模組並完成所有設定程序之後，才能使用此函數。如需詳細資訊，請參閱設定 Aurora PostgreSQL 資料庫叢集來使用 Aurora Machine Learning。

aws_comprehend.detect_sentiment

此函數會以文字做為輸入，並評估文字是否具有正面、負面、中性或混合的情緒狀態。它會輸出此情緒以及評估的信賴等級。函數文件的摘要如下。

```
aws_comprehend.detect_sentiment(
   IN input_text varchar,
   IN language_code varchar,
   IN max_rows_per_batch int,
   OUT sentiment varchar,
   OUT confidence real)
```

此函數的輸入和輸出如下。

- input_text – 要評估和指派情緒 (負面、正面、中性、混合) 的文字。
- language_code – 使用 2 字母的 ISO 639-1 識別碼搭配地區子標籤 (視需要) 或 ISO 639-2 三字母代碼 (視情況而定) 所識別之 input_text 的語言。例如，en 是英文代碼，而 zh 是簡體中文的代碼。如需詳細資訊，請參閱《Amazon 開發人員指南》中的支援的語言。
- max_rows_per_batch – 批次模式處理中每個批次的資料列數量上限。如需詳細資訊，請參閱了解批次模式和 Aurora Machine Learning 函數。
- sentiment – 輸入文字的情緒，識別為 POSITIVE、NEGATIVE、NEUTRAL 或 MIXED。
- confidence – 所在指定 sentiment 之準確性中的信賴程度。值的範圍從 0.0 到 1.0。

在下列內容中，您可以找到如何使用此函數的範例。

Example範例：使用 Amazon Comprehend 的簡單查詢

以下是簡單查詢的範例，該查詢會叫用此函數，以評估客戶對支援團隊的滿意度。假設您有一個資料庫資料表 (`support`)，其會在每次請求協助之後存放客戶意見回饋。此範例查詢會將 `aws_comprehend.detect_sentiment` 函數套用至資料表的 `feedback` 資料欄中的文字，並輸出情緒以及該情緒的信賴等級。此查詢也會以遞減順序輸出結果。

```
SELECT feedback, s.sentiment,s.confidence
    FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
    ORDER BY s.confidence DESC;
 feedback                          | sentiment | confidence
 ------------------------------------------------------------+-----------+------------
 Thank you for the excellent customer support!              | POSITIVE  |   0.999771
 The latest version of this product stinks!                 | NEGATIVE  |   0.999184
 Your support team is just awesome! I am blown away.        | POSITIVE  |   0.997774
 Your product is too complex, but your support is great.    | MIXED     |   0.957958
 Your support tech helped me in fifteen minutes.            | POSITIVE  |   0.949491
 My problem was never resolved!                             | NEGATIVE  |   0.920644
 When will the new version of this product be released?     | NEUTRAL   |   0.902706
 I cannot stand that chatbot.                               | NEGATIVE  |   0.895219
 Your support tech talked down to me.                       | NEGATIVE  |   0.868598
 It took me way too long to get a real person.              | NEGATIVE  |   0.481805

 (10 rows)
```

為了避免針對每個資料表資料列的情緒分析產生費用超過一次，您可以將結果具體化。請在您感興趣的資料列上執行此作業。例如，臨床醫師的筆記正在更新，以便只有法文 (`fr`) 的筆記使用情緒偵測函數。

```
UPDATE clinician_notes
SET sentiment = (aws_comprehend.detect_sentiment (french_notes, 'fr')).sentiment,
    confidence = (aws_comprehend.detect_sentiment (french_notes, 'fr')).confidence
WHERE
    clinician_notes.french_notes IS NOT NULL AND
    LENGTH(TRIM(clinician_notes.french_notes)) > 0 AND
    clinician_notes.sentiment IS NULL;
```

如需最佳化您函數呼叫的詳細資訊，請參閱 搭配 Aurora PostgreSQL 使用 Aurora Machine Learning 的效能考量。

搭配 Aurora PostgreSQL 資料庫叢集使用 SageMaker AI

如設定 Aurora PostgreSQL 來使用 Amazon SageMaker AI中所述，在設定 SageMaker AI 環境並與 Aurora PostgreSQL 整合之後，您可以使用 aws_sagemaker.invoke_endpoint 函數調用這些操作。aws_sagemaker.invoke_endpoint 函數只會連線到相同 AWS 區域中的模型端點。如果資料庫執行個體具有多個複本， AWS 區域 必須確保您會設定每個 SageMaker AI 模型，並將其部署到每個 AWS 區域。

對 aws_sagemaker.invoke_endpoint 的呼叫進行身分驗證，方法為使用您設定為將 Aurora PostgreSQL 資料庫叢集與 SageMaker AI 服務和您在設定過程中所提供的端點建立關聯的 IAM 角色。SageMaker AI 模型端點的範圍僅限個別帳戶，而非公有。所以 endpoint_name 網址中不包含帳戶 ID。SageMaker AI 會從資料庫執行個體 SageMaker AI IAM 角色提供的身分驗證字符中判斷帳戶 ID。

aws_sagemaker.invoke_endpoint

此函數會採取 SageMaker AI 端點作為輸入，以及採取應作為批次處理的列數目。其也會作為 SageMaker AI 模型端點所預期之各種參數的輸入。此函數的參考文件如下。

```
aws_sagemaker.invoke_endpoint(
  IN endpoint_name varchar,
  IN max_rows_per_batch int,
  VARIADIC model_input "any",
  OUT model_output varchar
  )
```

此函數的輸入和輸出如下。

- endpoint_name – AWS 區域獨立端點 URL。
- max_rows_per_batch – 批次模式處理中每個批次的資料列數量上限。如需詳細資訊，請參閱了解批次模式和 Aurora Machine Learning 函數。
- model_input – 模型的一或多個輸入參數。這些參數可以是 SageMaker AI 模型所需的任何資料類型。PostgreSQL 允許您為函數指定最多 100 個輸入參數。陣列資料類型必須是一維陣列，但可以包含 SageMaker AI 模型預期的任意數量元素。對 SageMaker AI 模型的輸入數僅受限於 SageMaker AI 6 MB 的訊息大小限制。
- model_output：作為文字的 SageMaker AI 模型輸出。

建立使用者定義函數來調用 SageMaker AI 模型

建立一個單獨的使用者定義函數，呼叫您每個 SageMaker AI 模型的 `aws_sagemaker.invoke_endpoint`。使用者定義函數代表託管模型的 SageMaker AI 端點。`aws_sagemaker.invoke_endpoint` 函數會在使用者定義函數內執行。使用者定義函數提供許多優點：

- 您可以自行為 SageMaker AI 模型命名，而非將所有 SageMaker AI 模型稱呼為 `aws_sagemaker.invoke_endpoint`。
- 您可以在您 SQL 應用程式程式碼中的同一個位置指定模型端點 URL。
- 您可以單獨控制每個 Aurora Machine Learning 函數的 `EXECUTE` 權限。
- 您可以使用 SQL 類型宣告模型輸入和輸出類型。SQL 會強制您傳遞至 SageMaker AI 模型的引數數量和類型，並視需要執行類型轉換。使用 SQL 類型也會將 SQL `NULL` 轉換成 SageMaker AI 模型預期的適當預設值。
- 如果您希望更快地傳回前幾個資料列，您可以減少批次大小上限。

如要指定使用者定義函數，請使用 SQL 資料定義語言 (DDL) 陳述式 `CREATE FUNCTION`。在您定義函數時，您可以指定以下內容：

- 對模型的輸入參數。
- 要調用的特定 SageMaker AI 端點。
- 傳回類型。

使用者定義函數會在針對輸入參數執行模型後，傳回 SageMaker AI 端點運算的推論。以下範例會為 SageMaker AI 模型建立使用者定義函數，並包含兩個輸入參數。

```
CREATE FUNCTION classify_event (IN arg1 INT, IN arg2 DATE, OUT category INT)
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', NULL,
        arg1, arg2                          -- model inputs are separate arguments
        )::INT                              -- cast the output to INT
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

注意下列事項：

- `aws_sagemaker.invoke_endpoint` 函數輸入可以是任何資料類型的一或多個參數。

- 此範例使用 INT 輸出類型。如果您將 varchar 類型的輸出轉換成不同類型，則必須將其轉換成 PostgreSQL 內建的純量類型，例如 INTEGER、REAL、FLOAT 或 NUMERIC。如需這些類型的詳細資訊，請參閱 PostgreSQL 文件的 Data Types。
- 指定 PARALLEL SAFE 來啟用平行查詢處理。如需更多詳細資訊，請參閱 使用平行查詢處理改善回應時間。
- 指定 COST 5000 來可預估執行函數的成本。使用正數來表示函數的預估執行成本 (單位為 cpu_operator_cost)。

將陣列作為輸入傳遞至 SageMaker AI 模型

aws_sagemaker.invoke_endpoint 函數可以擁有最多 100 個輸入參數。這項限制是 PostgreSQL 函數的限制。如果 SageMaker AI 模型需要超過 100 個相同類型的參數，請將模型參數作為陣列傳遞。

以下範例會定義一個函數，將陣列作為輸入傳遞至 SageMaker AI 迴歸模型。輸出會轉換為 REAL 值。

```
CREATE FUNCTION regression_model (params REAL[], OUT estimate REAL)
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
      'sagemaker_model_endpoint_name',
      NULL,
      params
      )::REAL
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

在調用 SageMaker AI 模型時指定批次大小

以下範例會為 SageMaker AI 模型建立使用者定義函數，將批次大小預設值設為 NULL。該函數也允許您在呼叫時提供不同的批次大小。

```
CREATE FUNCTION classify_event (
    IN event_type INT, IN event_day DATE, IN amount REAL, -- model inputs
    max_rows_per_batch INT DEFAULT NULL,  -- optional batch size limit
    OUT category INT)                     -- model output
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', max_rows_per_batch,
        event_type, event_day, COALESCE(amount, 0.0)
        )::INT              -- casts output to type INT
```

```
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

注意下列事項：

- 使用選用的 max_rows_per_batch 參數來控制批次模式函數呼叫的資料列數。如果您使用 NULL 值，則查詢最佳化工具會自動選擇最大的批次大小。如需詳細資訊，請參閱了解批次模式和 Aurora Machine Learning 函數。
- 根據預設，傳遞 NULL 作為參數值會先轉換成空白字串，再傳遞給 SageMaker AI。針對此範例，輸入具有不同的類型。
- 如果您有非文字的輸入，或是需要預設為空白字串以外值的文字輸入，請使用 COALESCE 陳述式。在對 COALESCE 的呼叫中，使用 aws_sagemaker.invoke_endpoint 將 NULL 轉換成所需的 null 取代值。針對此範例中的 amount 參數，NULL 值會轉換成 0.0。

呼叫具有多個輸出的 SageMaker AI 模型

以下範例會為 SageMaker AI 模型建立使用者定義函數，傳回多個輸出。您的函數需要將 aws_sagemaker.invoke_endpoint 函數的輸出轉換成對應的資料類型。例如，您可以針對 (x,y) 對使用內建的 PostgreSQL 點類型，或是使用者定義的複合類型。

這個使用者定義函數會針對輸出，使用複合類型從傳回多個輸出的模型傳回值。

```
CREATE TYPE company_forecasts AS (
    six_month_estimated_return real,
    one_year_bankruptcy_probability float);
CREATE FUNCTION analyze_company (
    IN free_cash_flow NUMERIC(18, 6),
    IN debt NUMERIC(18,6),
    IN max_rows_per_batch INT DEFAULT NULL,
    OUT prediction company_forecasts)
AS $$
    SELECT (aws_sagemaker.invoke_endpoint('endpt_name',
       max_rows_per_batch,free_cash_flow, debt))::company_forecasts;


$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

針對複合類型，請根據欄位在模型中的順序，以相同順序使用欄位，並將 aws_sagemaker.invoke_endpoint 的輸出轉換成您的複合類型。發起人可以透過名稱或 PostgreSQL ".*" 表示法擷取個別欄位。

將資料匯出至 Amazon S3 進行 SageMaker AI 模型訓練 (進階)

我們建議您使用提供的演算法和範例，熟悉 Aurora 機器學習和 SageMaker AI，而不是嘗試訓練自己的模型。如需詳細資訊，請參閱開始使用 Amazon SageMaker AI

如要訓練 SageMaker AI 模型，您可以將資料匯出到 Amazon S3 儲存貯體。SageMaker AI 會使用 Amazon S3 儲存貯體來訓練您的模型，之後再進行部署。您可以從 Aurora PostgreSQL 資料庫叢集查詢資料，並直接將這些資料儲存到存放在 Amazon S3 儲存貯體中的文字檔案。SageMaker AI 接著會使用 Amazon S3 儲存貯體的資料進行訓練。如需 SageMaker AI 模型訓練的詳細資訊，請參閱使用 Amazon SageMaker AI 訓練模型。

> **Note**
>
> 當您建立 S3 儲存貯體進行 SageMaker AI 模型訓練或批次評分時，請在 S3 儲存貯體名稱中使用 `sagemaker`。如需詳細資訊，請參閱《Amazon SageMaker AI 開發人員指南》中的指定 Amazon S3 儲存貯體以上傳訓練資料集和存放輸出資料。

如需匯出您資料的詳細資訊，請參閱 將資料從 Aurora PostgreSQL 資料庫叢集匯出至 Amazon S3。

搭配 Aurora PostgreSQL 使用 Aurora Machine Learning 的效能考量

當透過 Aurora 機器學習函數調用時，Amazon Comprehend 和 SageMaker AI 服務會執行大部分工作。這表示您可以視需要獨立擴展這些資源。對於您的 Aurora PostgreSQL 資料庫叢集，您可以盡可能有效地進行函數呼叫。接下來，您可以找到從 Aurora PostgreSQL 使用 Aurora Machine Learning 時需要注意的一些效能考量。

主題

- 了解批次模式和 Aurora Machine Learning 函數
- 使用平行查詢處理改善回應時間
- 使用具體化檢視和具體化資料行

了解批次模式和 Aurora Machine Learning 函數

通常，PostgreSQL 會一次針對單一資料列執行函數。Aurora Machine Learning 可以採用稱為「批次模式執行」的方法，透過將許多資料列對外部 Aurora Machine Learning 服務的呼叫合併成批次，來減少這項額外負荷。在批次模式中，Aurora Machine Learning 會收到輸入資料列批次的回應，然後一次

針對單一資料列將回應交付回正在執行的查詢。這項最佳化可以改善您 Aurora 查詢的輸送量，而無須限制 PostgreSQL 查詢最佳化工具。

如果函數是從 SELECT 清單、WHERE 子句或 HAVING 子句進行推論，Aurora 會自動使用批次模式。請注意，最上層的簡易 CASE 表達式適用批次模式執行。如果第一個 CASE 子句是包含批次模式函數呼叫的簡易述詞，則最上層的搜尋 WHEN 表達式也適用批次模式執行。

您的使用者定義函數必須是 LANGUAGE SQL 函數，且應指定 PARALLEL SAFE 和 COST 5000。

從 SELECT 陳述式到 FROM 子句的函數遷移

通常，Aurora 會自動將適用批次模式執行的 aws_ml 函數遷移至 FROM 子句。

您可以透過每個查詢層級，手動檢查適用批次模式函數遷移至 FROM 子句的過程。若要執行此作業，您可以使用 EXPLAIN 陳述式 (以及 ANALYZE 和 VERBOSE) 來在每個批次模式 Function Scan 的下方尋找「批次處理」資訊。您也可以使用 EXPLAIN (搭配 VERBOSE)，而無須執行查詢。您接著可以觀察對函數的呼叫在並未於原始陳述式中指定的巢狀迴圈聯結下方，是否顯示為 Function Scan。

在以下範例中，計劃中的巢狀迴圈聯結運算子會顯示 Aurora 遷移了 anomaly_score 函數。Aurora 會在函數適用批次模式執行時，將此函數從 SELECT 清單遷移至 FROM 子句。

```
EXPLAIN (VERBOSE, COSTS false)
SELECT anomaly_score(ts.R.description) from ts.R;
                          QUERY PLAN
---------------------------------------------------------------
 Nested Loop
   Output: anomaly_score((r.description)::text)
   ->  Seq Scan on ts.r
         Output: r.id, r.description, r.score
   ->  Function Scan on public.anomaly_score
         Output: anomaly_score.anomaly_score
         Function Call: anomaly_score((r.description)::text)
```

如要停止批次模式執行，請將 apg_enable_function_migration 參數設為 false。這可以防止將 aws_ml 函數從 SELECT 遷移至 FROM 子句。以下顯示作法。

```
SET apg_enable_function_migration = false;
```

apg_enable_function_migration 參數是 Grand Unified Configuration (GUC) 參數。Aurora PostgreSQL apg_plan_mgmt 延伸會針對查詢計劃管理識別此參數。如要停用工作階段中的函數

遷移，請使用查詢計劃管理來將結果計劃儲存為 approved 計劃。在執行時間，查詢計劃管理會透過其 approved 設定，強制實行 apg_enable_function_migration 計劃。這項強制實行無論 apg_enable_function_migration GUC 參數設定為何都會發生。如需更多詳細資訊，請參閱 管理 Aurora PostgreSQL 的查詢執行計劃。

使用 max_rows_per_batch 參數

aws_comprehend.detect_sentiment 和 aws_sagemaker.invoke_endpoint 函數都有一個 max_rows_per_batch 參數。此參數會指定可傳送至 Aurora Machine Learning 服務的資料列數目。您的函數處理的資料集越大，您的批次大小就越大。

批次模式函數會透過建置資料列批次，將 Aurora Machine Learning 函數呼叫的成本分散到大量的資料列，來改善效率。但是，如果 SELECT 陳述式因為 LIMIT 子句而提前完成，則可以針對比查詢所使用資料列數更多的資料列建構批次。這種方法可能會對您的帳戶產生額外費用 AWS 。如要利用批次模式執行的優點，同時避免建置過大的批次，請在您的函數呼叫中針對 max_rows_per_batch 參數使用較小的值。

如果您執行的查詢 EXPLAIN (VERBOSE、ANALYZE) 使用批次模式執行，您會看到位於巢狀迴圈聯結下方的 FunctionScan 運算子。EXPLAIN 報告的迴圈數等於您從 FunctionScan 運算子擷取資料列的次數。如果陳述式使用 LIMIT 子句，則擷取數將會是一致的。如要最佳化批次大小，請將 max_rows_per_batch 參數設為這個值。但是，如果在 WHERE 子句或 HAVING 子劇中的述詞內參考批次模式函數，您便可能無法事先得知擷取數。在此情況下，請使用迴圈做為準則並針對 max_rows_per_batch 進行實驗，尋找最佳化效能的設定。

驗證批次模式執行

要查看函數是否以批次處理模式執行，請使用 EXPLAIN ANALYZE。如果使用了批次模式執行，則查詢計劃會在「批次處理」區段包含資訊。

```
EXPLAIN ANALYZE SELECT user-defined-function();
 Batch Processing: num batches=1 avg/min/max batch size=3333.000/3333.000/3333.000
                                 avg/min/max batch call time=146.273/146.273/146.273
```

在此範例中，有 1 個包含了 3,333 個資料列的批次，其處理的時間為 146.273 毫秒。「批次處理」區段會顯示以下內容：

- 此函數掃描操作有多少批次
- 批次的平均大小、最小值和最大值
- 批次的平均執行時間、最小值和最大值

通常最後一個批次會比其餘批次小，並且通常會導致比平均小許多的批次大小下限。

如要更快地傳回前幾個資料列，請將 max_rows_per_batch 參數設為較小的值。

如要在您於使用者定義函數中使用 LIMIT 時減少對 ML 服務的批次模式呼叫數，請將 max_rows_per_batch 參數設為較小的值。

使用平行查詢處理改善回應時間

若要從大量資料列中盡快取得結果，您可以將平行查詢處理與批次模式處理結合。您可以針對 SELECT、CREATE TABLE AS SELECT 和 CREATE MATERIALIZED VIEW 陳述式使用平行查詢處理。

> **Note**
>
> PostgreSQL 尚未支援資料處理語言 (DML) 陳述式的平行查詢。

平行查詢處理會在資料庫和 ML 服務中進行。資料庫執行個體類別中的核心數會限制執行查詢時所能使用的平行處理程度。資料庫伺服器可以建構平行查詢執行計劃，分割一組平行工作者中的任務。然後，這些工作者都可以各自建置批次請求，其中包含了成千上萬個資料列 (或是每個服務允許的最大數量)。

所有平行工作者的批次請求都會傳送到 SageMaker AI 端點。端點可支援的平行處理程度受制於支援它的執行個體數目和類型。對於 K 程度的平行處理，您需要至少具有 K 個核心的資料庫執行個體類別。您也需要設定模型的 SageMaker AI 端點，讓其擁有 K 個效能夠高執行個體類別的初始執行個體。

若要使用平行查詢處理，您可以設定包含您計劃傳遞資料資料表的 parallel_workers 儲存參數。您可以將 parallel_workers 設為批次模式函數，例如 aws_comprehend.detect_sentiment。如果最佳化工具選擇平行查詢計畫，則可以批次和平行呼叫 AWS ML 服務。

您可以搭配 aws_comprehend.detect_sentiment 函數使用下列參數，取得具備四向平行處理的計劃。如果變更下列兩個參數之一，您必須重新啟動資料庫執行個體，變更才會生效

```
-- SET max_worker_processes to 8;  -- default value is 8
-- SET max_parallel_workers to 8;  -- not greater than max_worker_processes
SET max_parallel_workers_per_gather to 4;  -- not greater than max_parallel_workers

-- You can set the parallel_workers storage parameter on the table that the data
```

```
-- for the Aurora machine learning function is coming from in order to manually
 override the degree of
-- parallelism that would otherwise be chosen by the query optimizer
--
ALTER TABLE yourTable SET (parallel_workers = 4);

-- Example query to exploit both batch-mode execution and parallel query
EXPLAIN (verbose, analyze, buffers, hashes)
SELECT aws_comprehend.detect_sentiment(description, 'en')).*
FROM yourTable
WHERE id < 100;
```

如需控制平行查詢的詳細資訊，請參閱 PostgreSQL 文件中的 Parallel Plans。

使用具體化檢視和具體化資料行

當您從資料庫叫用 SageMaker AI 或 Amazon Comprehend 等 AWS 服務時，會根據該服務的定價政策向您的帳戶收費。若要將帳戶的費用降至最低，您可以將呼叫 AWS 服務的結果具體化為具體化資料欄，以便每個輸入列呼叫 AWS 服務不超過一次。如果需要，您可以新增 `materializedAt` 時間戳記資料行，記錄具體化資料行的時間。

一般單一資料列 `INSERT` 陳述式的延遲通常會比呼叫批次模式函數的延遲要小得多。因此，如果您針對您應用程式執行 `INSERT` 的每個單一資料列呼叫批次模式函數，您便可以無法滿足應用程式的延遲需求。為了具體化將 AWS 服務呼叫具體化資料欄的結果，高效能應用程式通常需要填入具體化資料欄。為了執行此作業，這些應用程式通常會發出 `UPDATE` 陳述式，同時在大型的資料列批次上運作。

`UPDATE` 會採用資料列層級鎖定，可能影響正在執行的應用程式。因此，您可能需要使用 `SELECT ... FOR UPDATE SKIP LOCKED`，或是使用 `MATERIALIZED VIEW`。

在大量資料列上即時運作的分析查詢可以將批次模式具體化與即時處理合併。為了執行此作業，這些查詢會將預先具體化結果的 `UNION ALL` 與尚未包含具體化結果的資料列查詢結合。在某些情況下，許多位置都需要該 `UNION ALL`，或是由第三方應用程式產生查詢。若是這種情況，您可以建立 `VIEW` 來封裝 `UNION ALL` 操作，避免向 SQL 應用程式的其餘部分公開這項詳細資訊。

您可以使用具體化檢視來及時具體化快照中任意 `SELECT` 陳述式的結果。您也可以使用具體化檢視來在未來隨時重新整理具體化檢視。目前 PostgreSQL 不支援增量重新整理，因此每次重新整理具體化檢視時，都會重新運算具體化檢視。

您可以使用 `CONCURRENTLY` 選項重新整理具體化檢視，更新具體化檢視的內容，而無須採取獨佔鎖定。這樣做可以讓 SQL 應用程式在重新整理具體化檢視時讀取具體化檢視。

監控 Aurora Machine Learning

您可以將自訂資料庫叢集參數群組中的 `track_functions` 參數設定為 `all` 來監控 `aws_ml` 函數。根據預設，此參數會設定為 `pl`，表示只會追蹤程序語言函數。藉由將其變更為 `all`，也會追蹤 `aws_ml` 函數。如需詳細資訊，請參閱 PostgreSQL 文件中的執行時間統計資料。

如需資訊，了解如何監控從 Aurora 機器學習函數呼叫之 SageMaker AI 操作的效能，請參閱《Amazon SageMaker AI 開發人員指南》中的監控 Amazon SageMaker AI。

將 `track_functions` 設為 `all` 後，您可以查詢 `pg_stat_user_functions` 檢視，以取得有關您定義並用來叫用 Aurora Machine Learning 服務之函數的統計資料。對於每個函數，檢視會提供 `calls`、`total_time` 和 `self_time` 的數目。

若要檢視 `aws_sagemaker.invoke_endpoint` 和 `aws_comprehend.detect_sentiment` 函數的統計資料，您可以使用下列查詢依結構描述名稱篩選結果。

```
SELECT * FROM pg_stat_user_functions
    WHERE schemaname
    LIKE 'aws_%';
```

若要清除統計資料，請執行以下操作。

```
SELECT pg_stat_reset();
```

您可以查詢 PostgreSQL `pg_proc` 系統目錄，取得呼叫 `aws_sagemaker.invoke_endpoint` 函數的 SQL 函數名稱。此目錄存放函數、程序等的相關資訊。如需詳細資訊，請參閱 PostgreSQL 文件中的 pg_proc。以下是查詢資料表以取得函數 (`proname`) 的名稱，這些名稱的來源 (`prosrc`) 包括文字 invoke_endpoint。

```
SELECT proname FROM pg_proc WHERE prosrc LIKE '%invoke_endpoint%';
```

Aurora AWS SDKs的程式碼範例

下列程式碼範例示範如何使用 Aurora 搭配 AWS 軟體開發套件 (SDK)。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

程式碼範例

- Aurora AWS SDKs的基本範例
  - Hello Aurora
  - 使用 AWS SDK 了解 Aurora 的基本概念
  - Aurora 使用 AWS SDKs的動作
    - CreateDBCluster 搭配 AWS SDK 使用
    - CreateDBClusterParameterGroup 搭配 AWS SDK 使用
    - CreateDBClusterSnapshot 搭配 AWS SDK 使用
    - CreateDBInstance 搭配 AWS SDK 使用
    - DeleteDBCluster 搭配 AWS SDK 使用
    - DeleteDBClusterParameterGroup 搭配 AWS SDK 使用
    - DeleteDBInstance 搭配 AWS SDK 使用
    - DescribeDBClusterParameterGroups 搭配 AWS SDK 使用
    - DescribeDBClusterParameters 搭配 AWS SDK 使用
    - DescribeDBClusterSnapshots 搭配 AWS SDK 使用
    - DescribeDBClusters 搭配 AWS SDK 使用
    - DescribeDBEngineVersions 搭配 AWS SDK 使用
    - DescribeDBInstances 搭配 AWS SDK 使用
    - DescribeOrderableDBInstanceOptions 搭配 AWS SDK 或 CLI 使用
    - ModifyDBClusterParameterGroup 搭配 AWS SDK 使用

- Aurora AWS SDKs的案例
  - 建立出借圖書館 REST API
  - 建立 Aurora 無伺服器工作項目追蹤器

Aurora AWS SDKs的基本範例

下列程式碼範例示範如何搭配使用 Amazon Aurora 的基本功能與 AWS SDK。

範例

- Hello Aurora
- 使用 AWS SDK 了解 Aurora 的基本概念
- Aurora 使用 AWS SDKs的動作
  - CreateDBCluster 搭配 AWS SDK 使用
  - CreateDBClusterParameterGroup 搭配 AWS SDK 使用
  - CreateDBClusterSnapshot 搭配 AWS SDK 使用
  - CreateDBInstance 搭配 AWS SDK 使用
  - DeleteDBCluster 搭配 AWS SDK 使用
  - DeleteDBClusterParameterGroup 搭配 AWS SDK 使用
  - DeleteDBInstance 搭配 AWS SDK 使用
  - DescribeDBClusterParameterGroups 搭配 AWS SDK 使用
  - DescribeDBClusterParameters 搭配 AWS SDK 使用
  - DescribeDBClusterSnapshots 搭配 AWS SDK 使用
  - DescribeDBClusters 搭配 AWS SDK 使用
  - DescribeDBEngineVersions 搭配 AWS SDK 使用
  - DescribeDBInstances 搭配 AWS SDK 使用
  - DescribeOrderableDBInstanceOptions 搭配 AWS SDK 或 CLI 使用
  - ModifyDBClusterParameterGroup 搭配 AWS SDK 使用

Hello Aurora

下列程式碼範例示範如何開始使用 Aurora。

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
using Amazon.RDS;
using Amazon.RDS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace AuroraActions;

public static class HelloAurora
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the
        // Amazon Relational Database Service (Amazon RDS).
        // Use your AWS profile name, or leave it blank to use the default
 profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
            ).Build();

        // Now the client is available for injection. Fetching it directly here
 for example purposes only.
        var rdsClient = host.Services.GetRequiredService<IAmazonRDS>();

        // You can use await and any of the async methods to get a response.
        var response = await rdsClient.DescribeDBClustersAsync(new
 DescribeDBClustersRequest { IncludeShared = true });
        Console.WriteLine($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        if (response.DBClusters == null)
```

```
        {
            Console.WriteLine($"\tNo clusters found.");
        }
        else
        {
            foreach (var cluster in response.DBClusters)
            {
                Console.WriteLine(
                    $"\tCluster: database: {cluster.DatabaseName} identifier:
 {cluster.DBClusterIdentifier}.");
            }
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeDBClusters。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

CMakeLists.txt CMake 檔案的程式碼。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS rds)

# Set this project's name.
project("hello_aurora")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
```

```
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_aurora.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_aurora.cpp 來源檔案的程式碼。

```
#include <aws/core/Aws.h>
#include <aws/rds/RDSClient.h>
#include <aws/rds/model/DescribeDBClustersRequest.h>
#include <iostream>

/*
```

```
 *  A "Hello Aurora" starter application which initializes an Amazon Relational
 Database Service (Amazon RDS) client
 *  and describes the Amazon Aurora (Aurora) clusters.
 *
 *  main function
 *
 *  Usage: 'hello_aurora'
 *
 */
int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::RDS::RDSClient rdsClient(clientConfig);

        Aws::String marker; // Used for pagination.
        std::vector<Aws::String> clusterIds;
        do {
            Aws::RDS::Model::DescribeDBClustersRequest request;

            Aws::RDS::Model::DescribeDBClustersOutcome outcome =
                    rdsClient.DescribeDBClusters(request);

            if (outcome.IsSuccess()) {
                for (auto &cluster: outcome.GetResult().GetDBClusters()) {
                    clusterIds.push_back(cluster.GetDBClusterIdentifier());
                }
                marker = outcome.GetResult().GetMarker();
            } else {
                result = 1;
                std::cerr << "Error with Aurora::GDescribeDBClusters. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
        } while (!marker.empty());
```

```
        std::cout << clusterIds.size() << " Aurora clusters found." << std::endl;
        for (auto &clusterId: clusterIds) {
            std::cout << "  clusterId " << clusterId << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DescribeDBClusters。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
package main

import (
 "context"
 "fmt"

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

// main uses the AWS SDK for Go V2 to create an Amazon Aurora client and list up
 to 20
// DB clusters in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
```

```
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 auroraClient := rds.NewFromConfig(sdkConfig)
 const maxClusters = 20
 fmt.Printf("Let's list up to %v DB clusters.\n", maxClusters)
 output, err := auroraClient.DescribeDBClusters(
  ctx, &rds.DescribeDBClustersInput{MaxRecords: aws.Int32(maxClusters)})
 if err != nil {
  fmt.Printf("Couldn't list DB clusters: %v\n", err)
  return
 }
 if len(output.DBClusters) == 0 {
  fmt.Println("No DB clusters found.")
 } else {
  for _, cluster := range output.DBClusters {
   fmt.Printf("DB cluster %v has database %v.\n", *cluster.DBClusterIdentifier,
    *cluster.DatabaseName)
  }
 }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 DescribeDBClusters。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.paginators.DescribeDBClustersIterable;

public class DescribeDbClusters {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        describeClusters(rdsClient);
        rdsClient.close();
    }

    public static void describeClusters(RdsClient rdsClient) {
        DescribeDBClustersIterable clustersIterable =
 rdsClient.describeDBClustersPaginator();
        clustersIterable.stream()
                .flatMap(r -> r.dbClusters().stream())
                .forEach(cluster -> System.out
                        .println("Database name: " + cluster.databaseName() + "
 Arn = " + cluster.dbClusterArn()));
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DescribeDBClusters。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import boto3
```

```
# Create an RDS client
rds = boto3.client("rds")

# Create a paginator for the describe_db_clusters operation
paginator = rds.get_paginator("describe_db_clusters")

# Use the paginator to get a list of DB clusters
response_iterator = paginator.paginate(
    PaginationConfig={
        "PageSize": 50,  # Adjust PageSize as needed
        "StartingToken": None,
    }
)

# Iterate through the pages of the response
clusters_found = False
for page in response_iterator:
    if "DBClusters" in page and page["DBClusters"]:
        clusters_found = True
        print("Here are your RDS Aurora clusters:")
        for cluster in page["DBClusters"]:
            print(
                f"Cluster ID: {cluster['DBClusterIdentifier']}, Engine:
 {cluster['Engine']}"
            )

if not clusters_found:
    print("No clusters found!")
```

- 如需 API 詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 DescribeDBClusters。

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
require 'aws-sdk-rds'

# Creates an Amazon RDS client for the AWS Region
rds = Aws::RDS::Client.new

puts 'Listing clusters in this AWS account...'

# Calls the describe_db_clusters method to get information about clusters
resp = rds.describe_db_clusters(max_records: 20)

# Checks if any clusters are found and prints the appropriate message
if resp.db_clusters.empty?
  puts 'No clusters found!'
else
  # Loops through the array of cluster objects and prints the cluster identifier
  resp.db_clusters.each do |cluster|
    puts "Cluster identifier: #{cluster.db_cluster_identifier}"
  end
end
```

- 如需 API 詳細資訊，請參閱《適用於 Ruby 的 AWS SDK API 參考》中的 DescribeDBClusters。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
use aws_sdk_rds::Client;

#[derive(Debug)]
struct Error(String);
impl std::fmt::Display for Error {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}
impl std::error::Error for Error {}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);

    let describe_db_clusters_output = client
        .describe_db_clusters()
        .send()
        .await
        .map_err(|e| Error(e.to_string()))?;
    println!(
        "Found {} clusters:",
        describe_db_clusters_output.db_clusters().len()
    );
    for cluster in describe_db_clusters_output.db_clusters() {
        let name = cluster.database_name().unwrap_or("Unknown");
        let engine = cluster.engine().unwrap_or("Unknown");
        let id = cluster.db_cluster_identifier().unwrap_or("Unknown");
        let class = cluster.db_cluster_instance_class().unwrap_or("Unknown");
        println!("\tDatabase: {name}",);
```

```
        println!("\t  Engine: {engine}",);
        println!("\t      ID: {id}",);
        println!("\tInstance: {class}",);
    }

    Ok(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 DescribeDBClusters。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 了解 Aurora 的基本概念

下列程式碼範例示範如何：

- 建立自訂 Aurora 資料庫叢集參數群組並設定參數值。
- 建立使用該參數群組的資料庫叢集。
- 建立包含該資料庫的資料庫執行個體。
- 拍攝該資料庫叢集的快照，並清理資源。

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在命令提示中執行互動式案例。

```
using Amazon.RDS;
using Amazon.RDS.Model;
```

```
using AuroraActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace AuroraScenario;

/// <summary>
/// Scenario for Amazon Aurora examples.
/// </summary>
public class AuroraScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
    1.  Return a list of the available DB engine families for Aurora MySql using
 the DescribeDBEngineVersionsAsync method.
    2.  Select an engine family and create a custom DB cluster parameter group
 using the CreateDBClusterParameterGroupAsync method.
    3.  Get the parameter group using the DescribeDBClusterParameterGroupsAsync
 method.
    4.  Get some parameters in the group using the
 DescribeDBClusterParametersAsync method.
    5.  Parse and display some parameters in the group.
    6.  Modify the auto_increment_offset and auto_increment_increment parameters
        using the ModifyDBClusterParameterGroupAsync method.
    7.  Get and display the updated parameters using the
 DescribeDBClusterParametersAsync method with a source of "user".
    8.  Get a list of allowed engine versions using the
 DescribeDBEngineVersionsAsync method.
    9.  Create an Aurora DB cluster that contains a MySql database and uses the
 parameter group.
        using the CreateDBClusterAsync method.
    10. Wait for the DB cluster to be ready using the DescribeDBClustersAsync
 method.
    11. Display and select from a list of instance classes available for the
 selected engine and version
        using the paginated DescribeOrderableDBInstanceOptions method.
    12. Create a database instance in the cluster using the CreateDBInstanceAsync
 method.
```

```
   13. Wait for the DB instance to be ready using the DescribeDBInstances
method.
   14. Display the connection endpoint string for the new DB cluster.
   15. Create a snapshot of the DB cluster using the
CreateDBClusterSnapshotAsync method.
   16. Wait for DB snapshot to be ready using the
DescribeDBClusterSnapshotsAsync method.
   17. Delete the DB instance using the DeleteDBInstanceAsync method.
   18. Delete the DB cluster using the DeleteDBClusterAsync method.
   19. Wait for DB cluster to be deleted using the DescribeDBClustersAsync
methods.
   20. Delete the cluster parameter group using the
DeleteDBClusterParameterGroupAsync.
   */

   private static readonly string sepBar = new('-', 80);
   private static AuroraWrapper auroraWrapper = null!;
   private static ILogger logger = null!;
   private static readonly string engine = "aurora-mysql";
   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon Relational Database Service
(Amazon RDS).
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
           .ConfigureServices((_, services) =>
               services.AddAWSService<IAmazonRDS>()
                   .AddTransient<AuroraWrapper>()
           )
           .Build();

       logger = LoggerFactory.Create(builder =>
       {
           builder.AddConsole();
       }).CreateLogger<AuroraScenario>();

       auroraWrapper = host.Services.GetRequiredService<AuroraWrapper>();

       Console.WriteLine(sepBar);
```

```
        Console.WriteLine(
            "Welcome to the Amazon Aurora: get started with DB clusters
example.");
        Console.WriteLine(sepBar);

        DBClusterParameterGroup parameterGroup = null!;
        DBCluster? newCluster = null;
        DBInstance? newInstance = null;

        try
        {
            var parameterGroupFamily = await ChooseParameterGroupFamilyAsync();

            parameterGroup = await
CreateDBParameterGroupAsync(parameterGroupFamily);

            var parameters = await
DescribeParametersInGroupAsync(parameterGroup.DBClusterParameterGroupName,
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });

            await
ModifyParametersAsync(parameterGroup.DBClusterParameterGroupName, parameters);

            await
DescribeUserSourceParameters(parameterGroup.DBClusterParameterGroupName);

            var engineVersionChoice = await
ChooseDBEngineVersionAsync(parameterGroupFamily);

            var newClusterIdentifier = "Example-Cluster-" + DateTime.Now.Ticks;

            newCluster = await CreateNewCluster
            (
                parameterGroup,
                engine,
                engineVersionChoice.EngineVersion,
                newClusterIdentifier
            );

            var instanceClassChoice = await ChooseDBInstanceClass(engine,
engineVersionChoice.EngineVersion);

            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;
```

```
            newInstance = await CreateNewInstance(
                newClusterIdentifier,
                engine,
                engineVersionChoice.EngineVersion,
                instanceClassChoice.DBInstanceClass,
                newInstanceIdentifier
            );

            DisplayConnectionString(newCluster!);
            await CreateSnapshot(newCluster!);
            await CleanupResources(newInstance, newCluster, parameterGroup);

            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
        }
        catch (Exception ex)
        {
            await CleanupResources(newInstance, newCluster, parameterGroup);
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// Choose the Aurora DB parameter group family from a list of available
options.
    /// </summary>
    /// <returns>The selected parameter group family.</returns>
    public static async Task<string> ChooseParameterGroupFamilyAsync()
    {
        Console.WriteLine(sepBar);
        // 1. Get a list of available engines.
        var engines = await
auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine);

        Console.WriteLine($"1. The following is a list of available DB parameter
group families for engine {engine}:");

        var parameterGroupFamilies =
            engines.GroupBy(e => e.DBParameterGroupFamily).ToList();
        for (var i = 1; i <= parameterGroupFamilies.Count; i++)
        {
            var parameterGroupFamily = parameterGroupFamilies[i - 1];
            // List the available parameter group families.
```

```
            Console.WriteLine(
                $"\t{i}. Family: {parameterGroupFamily.Key}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
        {
            Console.WriteLine("2. Select an available DB parameter group family
 by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        var parameterGroupFamilyChoice = parameterGroupFamilies[choiceNumber -
 1];
        Console.WriteLine(sepBar);
        return parameterGroupFamilyChoice.Key;
    }

    /// <summary>
    /// Create and get information on a DB parameter group.
    /// </summary>
    /// <param name="dbParameterGroupFamily">The DBParameterGroupFamily for the
 new DB parameter group.</param>
    /// <returns>The new DBParameterGroup.</returns>
    public static async Task<DBClusterParameterGroup>
 CreateDBParameterGroupAsync(string dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
 {dbParameterGroupFamily}:");

        var parameterGroup = await
 auroraWrapper.CreateCustomClusterParameterGroupAsync(
            dbParameterGroupFamily,
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            "New example parameter group");

        var groupInfo =
            await
 auroraWrapper.DescribeCustomDBClusterParameterGroupAsync(parameterGroup.DBClusterParamet

        Console.WriteLine(
            $"3. New DB parameter group created: \n\t{groupInfo?.Description}, \n
\tARN {groupInfo?.DBClusterParameterGroupName}");
```

```
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</
param>
    /// <param name="parameterNames">Optional specific names of parameters to
 describe.</param>
    /// <returns>The list of requested parameters.</returns>
    public static async Task<List<Parameter>>
 DescribeParametersInGroupAsync(string parameterGroupName, List<string>?
 parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
        Console.WriteLine(sepBar);

        var parameters =
            await
 auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName);

        var matchingParameters =
            parameters.Where(p => parameterNames == null ||
 parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);

        return matchingParameters;
    }

    /// <summary>
    /// Modify a parameter from a DBParameterGroup.
    /// </summary>
```

```
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameters">The parameters to modify.</param>
    /// <returns>Async task.</returns>
    public static async Task ModifyParametersAsync(string parameterGroupName,
 List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        await
 auroraWrapper.ModifyIntegerParametersInGroupAsync(parameterGroupName,
 parameters);

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Describe the user source parameters in the group.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</
param>
    /// <returns>Async task.</returns>
    public static async Task DescribeUserSourceParameters(string
 parameterGroupName)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("7. Describe updated user source parameters in the
 group.");

        var parameters =
            await
 auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName,
 "user");

        parameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);
    }
```

```
    /// <summary>
    /// Choose a DB engine version.
    /// </summary>
    /// <param name="dbParameterGroupFamily">DB parameter group family for engine
 choice.</param>
    /// <returns>The selected engine version.</returns>
    public static async Task<DBEngineVersion> ChooseDBEngineVersionAsync(string
 dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine,
 dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group
 family {dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
        {
            Console.WriteLine(
                $"\t{i}. {version.DBEngineVersionDescription}.");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by
 entering a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Create a new RDS DB cluster.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB cluster.</
param>
```

```
    /// <param name="engineName">Engine to use for the DB cluster.</param>
    /// <param name="engineVersion">Engine version to use for the DB cluster.</
param>
    /// <param name="clusterIdentifier">Cluster identifier to use for the DB
 cluster.</param>
    /// <returns>The new DB cluster.</returns>
    public static async Task<DBCluster?> CreateNewCluster(DBClusterParameterGroup
 parameterGroup,
        string engineName, string engineVersion, string clusterIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"9. Create a new DB cluster with identifier
 {clusterIdentifier}.");

        DBCluster newCluster;
        var clusters = await auroraWrapper.DescribeDBClustersPagedAsync();
        var isClusterCreated = clusters.Any(i => i.DBClusterIdentifier ==
 clusterIdentifier);

        if (isClusterCreated)
        {
            Console.WriteLine("Cluster already created.");
            newCluster = clusters.First(i => i.DBClusterIdentifier ==
 clusterIdentifier);
        }
        else
        {
            Console.WriteLine("Enter an admin username:");
            var username = Console.ReadLine();

            Console.WriteLine("Enter an admin password:");
            var password = Console.ReadLine();

            newCluster = await auroraWrapper.CreateDBClusterWithAdminAsync(
                "ExampleDatabase",
                clusterIdentifier,
                parameterGroup.DBClusterParameterGroupName,
                engineName,
                engineVersion,
                username!,
                password!
            );

            Console.WriteLine("10. Waiting for DB cluster to be ready...");
```

```
            while (newCluster.Status != "available")
            {
                Console.Write(".");
                Thread.Sleep(5000);
                clusters = await
 auroraWrapper.DescribeDBClustersPagedAsync(clusterIdentifier);
                newCluster = clusters.First();
            }
        }

        Console.WriteLine(sepBar);
        return newCluster;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
    public static async Task<OrderableDBInstanceOption>
 ChooseDBInstanceClass(string engine, string engineVersion)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await
 auroraWrapper.DescribeOrderableDBInstanceOptionsPagedAsync(engine,
 engineVersion);

        Console.WriteLine($"Available DB instance classes for engine {engine} and
 version {engineVersion}:");
        int i = 1;

        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\t{i}. Instance class: {instance.DBInstanceClass} (storage type
 {instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
```

```
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("11. Select an available DB instance class by
 entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new DB instance.
    /// </summary>
    /// <param name="engineName">Engine to use for the DB instance.</param>
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
    /// <param name="instanceIdentifier">Instance identifier to use for the DB
 instance.</param>
    /// <returns>The new DB instance.</returns>
    public static async Task<DBInstance?> CreateNewInstance(
        string clusterIdentifier,
        string engineName,
        string engineVersion,
        string instanceClass,
        string instanceIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"12. Create a new DB instance with identifier
 {instanceIdentifier}.");
        bool isInstanceReady = false;
        DBInstance newInstance;
        var instances = await auroraWrapper.DescribeDBInstancesPagedAsync();
        isInstanceReady = instances.FirstOrDefault(i =>
            i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
 "available";

        if (isInstanceReady)
        {
            Console.WriteLine("Instance already created.");
```

```
            newInstance = instances.First(i => i.DBInstanceIdentifier ==
 instanceIdentifier);
        }
        else
        {
            newInstance = await auroraWrapper.CreateDBInstanceInClusterAsync(
                clusterIdentifier,
                instanceIdentifier,
                engineName,
                engineVersion,
                instanceClass
            );

            Console.WriteLine("13. Waiting for DB instance to be ready...");
            while (!isInstanceReady)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                instances = await
 auroraWrapper.DescribeDBInstancesPagedAsync(instanceIdentifier);
                isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
 "available";
                newInstance = instances.First();
            }
        }

        Console.WriteLine(sepBar);
        return newInstance;
    }

    /// <summary>
    /// Display a connection string for an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">The DB cluster to use to get a connection string.</
param>
    public static void DisplayConnectionString(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Display the connection string.
        Console.WriteLine("14. New DB cluster connection string: ");
        Console.WriteLine(
            $"\n{engine} -h {cluster.Endpoint} -P {cluster.Port} "
            + $"-u {cluster.MasterUsername} -p [YOUR PASSWORD]\n");
```

```
        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">DB cluster to use when creating a snapshot.</param>
    /// <returns>The snapshot object.</returns>
    public static async Task<DBClusterSnapshot> CreateSnapshot(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"15. Creating snapshot from DB cluster
{cluster.DBClusterIdentifier}.");
        var snapshot = await
auroraWrapper.CreateClusterSnapshotByIdentifierAsync(
            cluster.DBClusterIdentifier,
            "ExampleSnapshot-" + DateTime.Now.Ticks);

        // Wait for the snapshot to be available.
        bool isSnapshotReady = false;

        Console.WriteLine($"16. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            Console.Write(".");
            Thread.Sleep(5000);
            var snapshots =
                await
auroraWrapper.DescribeDBClusterSnapshotsByIdentifierAsync(cluster.DBClusterIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBClusterSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Clean up resources from the scenario.
    /// </summary>
```

```
    /// <param name="newInstance">The instance to clean up.</param>
    /// <param name="newCluster">The cluster to clean up.</param>
    /// <param name="parameterGroup">The parameter group to clean up.</param>
    /// <returns>Async Task.</returns>
    private static async Task CleanupResources(
        DBInstance? newInstance,
        DBCluster? newCluster,
        DBClusterParameterGroup? parameterGroup)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (newInstance is not null && GetYesNoResponse($"\tClean up instance
{newInstance.DBInstanceIdentifier}? (y/n)"))
        {
            // Delete the DB instance.
            Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
            await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
        }

        if (newCluster is not null && GetYesNoResponse($"\tClean up cluster
{newCluster.DBClusterIdentifier}? (y/n)"))
        {
            // Delete the DB cluster.
            Console.WriteLine($"18. Deleting the DB cluster
{newCluster.DBClusterIdentifier}.");
            await
auroraWrapper.DeleteDBClusterByIdentifierAsync(newCluster.DBClusterIdentifier);

            // Wait for the DB cluster to delete.
            Console.WriteLine($"19. Waiting for the DB cluster to delete...");
            bool isClusterDeleted = false;

            while (!isClusterDeleted)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                var cluster = await auroraWrapper.DescribeDBClustersPagedAsync();
                isClusterDeleted = cluster.All(i => i.DBClusterIdentifier !=
newCluster.DBClusterIdentifier);
            }
```

```
            Console.WriteLine("DB cluster deleted.");
        }

        if (parameterGroup is not null && GetYesNoResponse($"\tClean up parameter
 group? (y/n)"))
        {
            Console.WriteLine($"20. Deleting the DB parameter group
 {parameterGroup.DBClusterParameterGroupName}.");
            await
 auroraWrapper.DeleteClusterParameterGroupByNameAsync(parameterGroup.DBClusterParameterGr
            Console.WriteLine("Parameter group deleted.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
```

案例呼叫用以管理 Aurora 動作的包裝函式方式。

```
using Amazon.RDS;
using Amazon.RDS.Model;

namespace AuroraActions;

/// <summary>
/// Wrapper for the Amazon Aurora cluster client operations.
```

```
/// </summary>
public class AuroraWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public AuroraWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }

    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family
 name.</param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
 DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }

    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</
param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
```

```
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await
_amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }

    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await
_amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }
```

```
    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</
param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
 DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }

    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</
param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string
 groupName, List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable.GetValueOrDefault() && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the
 allowed values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
```

```
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await
_amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }


    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string
engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }

    /// <summary>
```

```
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string
groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await
_amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new cluster and database.
    /// </summary>
    /// <param name="dbName">The name of the new database.</param>
    /// <param name="clusterIdentifier">The identifier of the cluster.</param>
    /// <param name="parameterGroupName">The name of the parameter group.</param>
    /// <param name="dbEngine">The engine to use for the new cluster.</param>
    /// <param name="dbEngineVersion">The version of the engine to use.</param>
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
```

```
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }

    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }

    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
cluster.</param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
```

```
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            if (response.DBClusters != null)
            {
                results.AddRange(response.DBClusters);
            }
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action
DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name
or size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
```

```
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }

    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }

    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
```

```
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }

    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });
```

```
        return response.DBInstance;
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的下列主題。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
//! Routine which creates an Amazon Aurora DB cluster and demonstrates several
 operations
//! on that cluster.
/*!
 \sa gettingStartedWithDBClusters()
 \param clientConfiguration: AWS client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::gettingStartedWithDBClusters(
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::RDS::RDSClient client(clientConfig);

    printAsterisksLine();
    std::cout << "Welcome to the Amazon Relational Database Service (Amazon
 Aurora)"
              << std::endl;
    std::cout << "get started with DB clusters demo." << std::endl;
    printAsterisksLine();

    std::cout << "Checking for an existing DB cluster parameter group named '" <<
              CLUSTER_PARAMETER_GROUP_NAME << "'." << std::endl;
    Aws::String dbParameterGroupFamily("Undefined");
    bool parameterGroupFound = true;
    {
        // 1. Check if the DB cluster parameter group already exists.
        Aws::RDS::Model::DescribeDBClusterParameterGroupsRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsOutcome outcome =
                client.DescribeDBClusterParameterGroups(request);

        if (outcome.IsSuccess()) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' already exists." <<
 std::endl;
            dbParameterGroupFamily =
 outcome.GetResult().GetDBClusterParameterGroups()
[0].GetDBParameterGroupFamily();
        }
        else if (outcome.GetError().GetErrorType() ==
                 Aws::RDS::RDSErrors::D_B_PARAMETER_GROUP_NOT_FOUND_FAULT) {
            std::cout << "DB cluster parameter group named '" <<
```

```
                    CLUSTER_PARAMETER_GROUP_NAME << "' does not exist." <<
std::endl;
            parameterGroupFound = false;
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    if (!parameterGroupFound) {
        Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

        // 2. Get available parameter group families for the specified engine.
        if (!getDBEngineVersions(DB_ENGINE, NO_PARAMETER_GROUP_FAMILY,
                                 engineVersions, client)) {
            return false;
        }

        std::cout << "Getting available parameter group families for " <<
DB_ENGINE
                  << "."
                  << std::endl;
        std::vector<Aws::String> families;
        for (const Aws::RDS::Model::DBEngineVersion &version: engineVersions) {
            Aws::String family = version.GetDBParameterGroupFamily();
            if (std::find(families.begin(), families.end(), family) ==
                families.end()) {
                families.push_back(family);
                std::cout << "  " << families.size() << ": " << family <<
std::endl;
            }
        }

        int choice = askQuestionForIntRange("Which family do you want to use? ",
1,
                                            static_cast<int>(families.size()));
        dbParameterGroupFamily = families[choice - 1];
    }
    if (!parameterGroupFound) {
        // 3.  Create a DB cluster parameter group.
        Aws::RDS::Model::CreateDBClusterParameterGroupRequest request;
```

```
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetDBParameterGroupFamily(dbParameterGroupFamily);
        request.SetDescription("Example cluster parameter group.");

        Aws::RDS::Model::CreateDBClusterParameterGroupOutcome outcome =
                client.CreateDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
created."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    printAsterisksLine();
    std::cout << "Let's set some parameter values in your cluster parameter
group."
              << std::endl;

    Aws::Vector<Aws::RDS::Model::Parameter> autoIncrementParameters;
    // 4.  Get the parameters in the DB cluster parameter group.
    if (!getDBClusterParameters(CLUSTER_PARAMETER_GROUP_NAME,
AUTO_INCREMENT_PREFIX,
                                NO_SOURCE,
                                autoIncrementParameters,
                                client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    Aws::Vector<Aws::RDS::Model::Parameter> updateParameters;

    for (Aws::RDS::Model::Parameter &autoIncParameter: autoIncrementParameters) {
        if (autoIncParameter.GetIsModifiable() &&
            (autoIncParameter.GetDataType() == "integer")) {
            std::cout << "The " << autoIncParameter.GetParameterName()
                      << " is described as: " <<
                      autoIncParameter.GetDescription() << "." << std::endl;
```

```
            if (autoIncParameter.ParameterValueHasBeenSet()) {
                std::cout << "The current value is "
                          << autoIncParameter.GetParameterValue()
                          << "." << std::endl;
            }
            std::vector<int> splitValues = splitToInts(
                    autoIncParameter.GetAllowedValues(), '-');
            if (splitValues.size() == 2) {
                int newValue = askQuestionForIntRange(
                        Aws::String("Enter a new value between ") +
                        autoIncParameter.GetAllowedValues() + ": ",
                        splitValues[0], splitValues[1]);
                autoIncParameter.SetParameterValue(std::to_string(newValue));
                updateParameters.push_back(autoIncParameter);

            }
            else {
                std::cerr << "Error parsing " <<
autoIncParameter.GetAllowedValues()
                          << std::endl;
            }
        }
    }

    {
        // 5.  Modify the auto increment parameters in the DB cluster parameter
group.
        Aws::RDS::Model::ModifyDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetParameters(updateParameters);

        Aws::RDS::Model::ModifyDBClusterParameterGroupOutcome outcome =
                client.ModifyDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
modified."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::ModifyDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

```
    }

    std::cout
            << "You can get a list of parameters you've set by specifying a
source of 'user'."
            << std::endl;

    Aws::Vector<Aws::RDS::Model::Parameter> userParameters;
    // 6.  Display the modified parameters in the DB cluster parameter group.
    if (!getDBClusterParameters(CLUSTER_PARAMETER_GROUP_NAME, NO_NAME_PREFIX,
"user",
                                userParameters,
                                client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    for (const auto &userParameter: userParameters) {
        std::cout << "  " << userParameter.GetParameterName() << ", " <<
                  userParameter.GetDescription() << ", parameter value - "
                  << userParameter.GetParameterValue() << std::endl;
    }

    printAsterisksLine();
    std::cout << "Checking for an existing DB Cluster." << std::endl;

    Aws::RDS::Model::DBCluster dbCluster;
    // 7.  Check if the DB cluster already exists.
    if (!describeDBCluster(DB_CLUSTER_IDENTIFIER, dbCluster, client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    Aws::String engineVersionName;
    Aws::String engineName;
    if (dbCluster.DBClusterIdentifierHasBeenSet()) {
        std::cout << "The DB cluster already exists." << std::endl;
        engineVersionName = dbCluster.GetEngineVersion();
        engineName = dbCluster.GetEngine();

    }
    else {
        std::cout << "Let's create a DB cluster." << std::endl;
        const Aws::String administratorName = askQuestion(
```

```
                "Enter an administrator username for the database: ");
        const Aws::String administratorPassword = askQuestion(
                "Enter a password for the administrator (at least 8 characters):
");
        Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

        // 8.  Get a list of engine versions for the parameter group family.
        if (!getDBEngineVersions(DB_ENGINE, dbParameterGroupFamily,
engineVersions,
                                 client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
            return false;
        }

        std::cout << "The available engines for your parameter group family are:"
                  << std::endl;

        int index = 1;
        for (const Aws::RDS::Model::DBEngineVersion &engineVersion:
engineVersions) {
            std::cout << "   " << index << ": " <<
engineVersion.GetEngineVersion()
                      << std::endl;
            ++index;
        }
        int choice = askQuestionForIntRange("Which engine do you want to use? ",
1,

static_cast<int>(engineVersions.size()));
        const Aws::RDS::Model::DBEngineVersion engineVersion =
engineVersions[choice -
                                                                              1];

        engineName = engineVersion.GetEngine();
        engineVersionName = engineVersion.GetEngineVersion();
        std::cout << "Creating a DB cluster named '" << DB_CLUSTER_IDENTIFIER
                  << "' and database '" << DB_NAME << "'.\n"
                  << "The DB cluster is configured to use your custom cluster
parameter group '"
                  << CLUSTER_PARAMETER_GROUP_NAME << "', and \n"
                  << "selected engine version " <<
engineVersion.GetEngineVersion()
                  << ".\nThis typically takes several minutes." << std::endl;
```

```
        Aws::RDS::Model::CreateDBClusterRequest request;
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetEngine(engineName);
        request.SetEngineVersion(engineVersionName);
        request.SetMasterUsername(administratorName);
        request.SetMasterUserPassword(administratorPassword);

        Aws::RDS::Model::CreateDBClusterOutcome outcome =
                client.CreateDBCluster(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBCluster. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
            return false;
        }
    }

    std::cout << "Waiting for the DB cluster to become available." << std::endl;

    int counter = 0;
    // 11. Wait for the DB cluster to become available.
    do {
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 900) {
            std::cerr << "Wait for cluster to become available timed out ofter "
                      << counter
                      << " seconds." << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, "", client);
            return false;
        }

        dbCluster = Aws::RDS::Model::DBCluster();
        if (!describeDBCluster(DB_CLUSTER_IDENTIFIER, dbCluster, client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, "", client);
```

```
            return false;
        }

        if ((counter % 20) == 0) {
            std::cout << "Current DB cluster status is '"
                      << dbCluster.GetStatus()
                      << "' after " << counter << " seconds." << std::endl;
        }
    } while (dbCluster.GetStatus() != "available");

    if (dbCluster.GetStatus() == "available") {
        std::cout << "The DB cluster has been created." << std::endl;
    }

    printAsterisksLine();
    Aws::RDS::Model::DBInstance dbInstance;
    // 11.  Check if the DB instance already exists.
    if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER, "",
                         client);
        return false;
    }

    if (dbInstance.DbInstancePortHasBeenSet()) {
        std::cout << "The DB instance already exists." << std::endl;
    }
    else {
        std::cout << "Let's create a DB instance." << std::endl;

        Aws::String dbInstanceClass;
        // 12.  Get a list of instance classes.
        if (!chooseDBInstanceClass(engineName,
                                   engineVersionName,
                                   dbInstanceClass,
                                   client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
""",
                             client);
            return false;
        }

        std::cout << "Creating a DB instance named '" << DB_INSTANCE_IDENTIFIER
                  << "' with selected DB instance class '" << dbInstanceClass
                  << "'.\nThis typically takes several minutes." << std::endl;
```

```
        // 13. Create a DB instance.
        Aws::RDS::Model::CreateDBInstanceRequest request;
        request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetEngine(engineName);
        request.SetDBInstanceClass(dbInstanceClass);

        Aws::RDS::Model::CreateDBInstanceOutcome outcome =
                client.CreateDBInstance(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB instance creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::CreateDBInstance. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
"",
                             client);
            return false;
        }
    }

    std::cout << "Waiting for the DB instance to become available." << std::endl;

    counter = 0;
    // 14. Wait for the DB instance to become available.
    do {
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 900) {
            std::cerr << "Wait for instance to become available timed out ofter "
                      << counter
                      << " seconds." << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
            return false;
        }

        dbInstance = Aws::RDS::Model::DBInstance();
```

```
        if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
            return false;
        }

        if ((counter % 20) == 0) {
            std::cout << "Current DB instance status is '"
                      << dbInstance.GetDBInstanceStatus()
                      << "' after " << counter << " seconds." << std::endl;
        }
    } while (dbInstance.GetDBInstanceStatus() != "available");

    if (dbInstance.GetDBInstanceStatus() == "available") {
        std::cout << "The DB instance has been created." << std::endl;
    }

    // 15. Display the connection string that can be used to connect a 'mysql'
shell to the database.
    displayConnection(dbCluster);

    printAsterisksLine();

    if (askYesNoQuestion(
            "Do you want to create a snapshot of your DB cluster (y/n)? ")) {
        Aws::String snapshotID(DB_CLUSTER_IDENTIFIER + "-" +
                               Aws::String(Aws::Utils::UUID::RandomUUID()));
        {
            std::cout << "Creating a snapshot named " << snapshotID << "." <<
std::endl;
            std::cout << "This typically takes a few minutes." << std::endl;

            // 16. Create a snapshot of the DB cluster. (CreateDBClusterSnapshot)
            Aws::RDS::Model::CreateDBClusterSnapshotRequest request;
            request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::CreateDBClusterSnapshotOutcome outcome =
                    client.CreateDBClusterSnapshot(request);

            if (outcome.IsSuccess()) {
                std::cout << "Snapshot creation has started."
                          << std::endl;
```

```
            }
            else {
                std::cerr << "Error with Aurora::CreateDBClusterSnapshot. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }
        }

        std::cout << "Waiting for the snapshot to become available." <<
std::endl;

        Aws::RDS::Model::DBClusterSnapshot snapshot;
        counter = 0;
        do {
            std::this_thread::sleep_for(std::chrono::seconds(1));
            ++counter;
            if (counter > 600) {
                std::cerr << "Wait for snapshot to be available timed out ofter "
                          << counter
                          << " seconds." << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }

            // 17. Wait for the snapshot to become available.
            Aws::RDS::Model::DescribeDBClusterSnapshotsRequest request;
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBClusterSnapshotsOutcome outcome =
                    client.DescribeDBClusterSnapshots(request);

            if (outcome.IsSuccess()) {
                snapshot = outcome.GetResult().GetDBClusterSnapshots()[0];
            }
            else {
                std::cerr << "Error with Aurora::DescribeDBClusterSnapshots. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
```

```
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
 client);
                return false;
            }

            if ((counter % 20) == 0) {
                std::cout << "Current snapshot status is '"
                          << snapshot.GetStatus()
                          << "' after " << counter << " seconds." << std::endl;
            }
        } while (snapshot.GetStatus() != "available");

        if (snapshot.GetStatus() != "available") {
            std::cout << "A snapshot has been created." << std::endl;
        }
    }

    printAsterisksLine();

    bool result = true;
    if (askYesNoQuestion(
            "Do you want to delete the DB cluster, DB instance, and parameter
 group (y/n)? ")) {
        result = cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                  DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
                                  client);
    }

    return result;
}

//! Routine which gets a DB cluster description.
/*!
 \sa describeDBCluster()
 \param dbClusterIdentifier: A DB cluster identifier.
 \param clusterResult: The 'DBCluster' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBCluster(const Aws::String &dbClusterIdentifier,
                                       Aws::RDS::Model::DBCluster &clusterResult,
                                       const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBClustersRequest request;
```

```
    request.SetDBClusterIdentifier(dbClusterIdentifier);

    Aws::RDS::Model::DescribeDBClustersOutcome outcome =
            client.DescribeDBClusters(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        clusterResult = outcome.GetResult().GetDBClusters()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_CLUSTER_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::GDescribeDBClusters. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB cluster does not exist.
        // Instead, clusterResult is set to empty.
    else {
        clusterResult = Aws::RDS::Model::DBCluster();
    }

    return result;

}


//! Routine which gets DB parameters using the 'DescribeDBClusterParameters' api.
/*!
 \sa getDBCLusterParameters()
 \param parameterGroupName: The name of the cluster parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBCLusterParameters(const Aws::String
 &parameterGroupName,
                                             const Aws::String &namePrefix,
                                             const Aws::String &source,

 Aws::Vector<Aws::RDS::Model::Parameter> &parametersResult,
                                             const Aws::RDS::RDSClient &client) {
```

```
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeDBClusterParametersRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
            request.SetSource(source);
        }

        Aws::RDS::Model::DescribeDBClusterParametersOutcome outcome =
                client.DescribeDBClusterParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
                }
                else {
                    parametersResult.push_back(parameter);
                }
            }

            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}


//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
```

```
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBEngineVersions(const Aws::String &engineName,
                                         const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                         const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
    }

    engineVersionsResult.clear();
    Aws::String marker; // The marker is used for pagination.
    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
                client.DescribeDBEngineVersions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersions =
                    outcome.GetResult().GetDBEngineVersions();

            engineVersionsResult.insert(engineVersionsResult.end(),
                                        engineVersions.begin(),
 engineVersions.end());
            marker  = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBEngineVersionsRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

```
    } while (!marker.empty());

    return true;
}


//! Routine which gets a DB instance description.
/*!
 \sa describeDBCluster()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBInstance(const Aws::String &dbInstanceIdentifier,
                                        Aws::RDS::Model::DBInstance
 &instanceResult,
                                        const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
            client.DescribeDBInstances(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::DescribeDBInstances. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
    }

    return result;
}
```

```
//! Routine which gets available DB instance classes, displays the list
//! to the user, and returns the user selection.
/*!
 \sa chooseDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::chooseDBInstanceClass(const Aws::String &engine,
                                           const Aws::String &engineVersion,
                                           Aws::String &dbInstanceClass,
                                           const Aws::RDS::RDSClient &client) {
    std::vector<Aws::String> instanceClasses;
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
        request.SetEngine(engine);
        request.SetEngineVersion(engineVersion);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
                client.DescribeOrderableDBInstanceOptions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
 &options =
                    outcome.GetResult().GetOrderableDBInstanceOptions();
            for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
 options) {
                const Aws::String &instanceClass = option.GetDBInstanceClass();
                if (std::find(instanceClasses.begin(), instanceClasses.end(),
                              instanceClass) == instanceClasses.end()) {
                    instanceClasses.push_back(instanceClass);
                }
            }
            marker = outcome.GetResult().GetMarker();
        }
        else {
```

```
            std::cerr << "Error with Aurora::DescribeOrderableDBInstanceOptions.
 "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << "The available DB instance classes for your database engine
 are:"
              << std::endl;
    for (int i = 0; i < instanceClasses.size(); ++i) {
        std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
    }

    int choice = askQuestionForIntRange(
            "Which DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
    return true;
}

//! Routine which deletes resources created by the scenario.
/*!
\sa cleanUpResources()
\param parameterGroupName: A parameter group name, this may be empty.
\param dbInstanceIdentifier: A DB instance identifier, this may be empty.
\param client: 'RDSClient' instance.
\return bool: Successful completion.
*/
bool AwsDoc::Aurora::cleanUpResources(const Aws::String &parameterGroupName,
                                      const Aws::String &dbClusterIdentifier,
                                      const Aws::String &dbInstanceIdentifier,
                                      const Aws::RDS::RDSClient &client) {
    bool result = true;
    bool instanceDeleting = false;
    bool clusterDeleting = false;
    if (!dbInstanceIdentifier.empty()) {
        {
            // 18. Delete the DB instance.
            Aws::RDS::Model::DeleteDBInstanceRequest request;
            request.SetDBInstanceIdentifier(dbInstanceIdentifier);
            request.SetSkipFinalSnapshot(true);
            request.SetDeleteAutomatedBackups(true);
```

```
            Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                    client.DeleteDBInstance(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB instance deletion has started."
                          << std::endl;
                instanceDeleting = true;
                std::cout
                        << "Waiting for DB instance to delete before deleting the
parameter group."
                        << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBInstance. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
    }

    if (!dbClusterIdentifier.empty()) {
        {
            // 19. Delete the DB cluster.
            Aws::RDS::Model::DeleteDBClusterRequest request;
            request.SetDBClusterIdentifier(dbClusterIdentifier);
            request.SetSkipFinalSnapshot(true);

            Aws::RDS::Model::DeleteDBClusterOutcome outcome =
                    client.DeleteDBCluster(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB cluster deletion has started."
                          << std::endl;
                clusterDeleting = true;
                std::cout
                        << "Waiting for DB cluster to delete before deleting the
parameter group."
                        << std::endl;
                std::cout << "This may take a while." << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBCluster. "
```

```
                        << outcome.GetError().GetMessage()
                        << std::endl;
                result = false;
            }
        }
    }
    int counter = 0;

    while (clusterDeleting || instanceDeleting) {
        // 20. Wait for the DB cluster and instance to be deleted.
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 800) {
            std::cerr << "Wait for instance to delete timed out ofter " <<
counter
                      << " seconds." << std::endl;
            return false;
        }

        Aws::RDS::Model::DBInstance dbInstance = Aws::RDS::Model::DBInstance();
        if (instanceDeleting) {
            if (!describeDBInstance(dbInstanceIdentifier, dbInstance, client)) {
                return false;
            }
            instanceDeleting = dbInstance.DBInstanceIdentifierHasBeenSet();
        }

        Aws::RDS::Model::DBCluster dbCluster = Aws::RDS::Model::DBCluster();
        if (clusterDeleting) {
            if (!describeDBCluster(dbClusterIdentifier, dbCluster, client)) {
                return false;
            }

            clusterDeleting = dbCluster.DBClusterIdentifierHasBeenSet();
        }

        if ((counter % 20) == 0) {
            if (instanceDeleting) {
                std::cout << "Current DB instance status is '"
                          << dbInstance.GetDBInstanceStatus() << "." <<
std::endl;
            }

            if (clusterDeleting) {
```

```
                std::cout << "Current DB cluster status is '"
                          << dbCluster.GetStatus() << "." << std::endl;
            }
        }
    }

    if (!parameterGroupName.empty()) {
        // 21. Delete the DB cluster parameter group.
        Aws::RDS::Model::DeleteDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(parameterGroupName);

        Aws::RDS::Model::DeleteDBClusterParameterGroupOutcome outcome =
                client.DeleteDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB parameter group was successfully deleted."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::DeleteDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
    }

    return result;
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的下列主題。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters

- DescribeDBClusterSnapshots
- DescribeDBClusters
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeOrderableDBInstanceOptions
- ModifyDBClusterParameterGroup

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在命令提示中執行互動式案例。

```
import (
 "aurora/actions"
 "context"
 "fmt"
 "log"
 "slices"
 "sort"
 "strconv"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/google/uuid"
)
```

```
// GetStartedClusters is an interactive example that shows you how to use the AWS
 SDK for Go
// with Amazon Aurora to do the following:
//
// 1. Create a custom DB cluster parameter group and set parameter values.
// 2. Create an Aurora DB cluster that is configured to use the parameter group.
// 3. Create a DB instance in the DB cluster that contains a database.
// 4. Take a snapshot of the DB cluster.
// 5. Delete the DB instance, DB cluster, and parameter group.
type GetStartedClusters struct {
 sdkConfig  aws.Config
 dbClusters actions.DbClusters
 questioner demotools.IQuestioner
 helper     IScenarioHelper
 isTestRun  bool
}

// NewGetStartedClusters constructs a GetStartedClusters instance from a
 configuration.
// It uses the specified config to get an Amazon Relational Database Service
 (Amazon RDS)
// client and create wrappers for the actions used in the scenario.
func NewGetStartedClusters(sdkConfig aws.Config, questioner
 demotools.IQuestioner,
 helper IScenarioHelper) GetStartedClusters {
 auroraClient := rds.NewFromConfig(sdkConfig)
 return GetStartedClusters{
  sdkConfig:  sdkConfig,
  dbClusters: actions.DbClusters{AuroraClient: auroraClient},
  questioner: questioner,
  helper:     helper,
 }
}

// Run runs the interactive scenario.
func (scenario GetStartedClusters) Run(ctx context.Context, dbEngine string,
 parameterGroupName string,
 clusterName string, dbName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
  }
 }()
```

```
 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon Aurora DB Cluster demo.")
 log.Println(strings.Repeat("-", 88))

 parameterGroup := scenario.CreateParameterGroup(ctx, dbEngine,
 parameterGroupName)
 scenario.SetUserParameters(ctx, parameterGroupName)
 cluster := scenario.CreateCluster(ctx, clusterName, dbEngine, dbName,
 parameterGroup)
 scenario.helper.Pause(5)
 dbInstance := scenario.CreateInstance(ctx, cluster)
 scenario.DisplayConnection(cluster)
 scenario.CreateSnapshot(ctx, clusterName)
 scenario.Cleanup(ctx, dbInstance, cluster, parameterGroup)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}

// CreateParameterGroup shows how to get available engine versions for a
 specified
// database engine and create a DB cluster parameter group that is compatible
 with a
// selected engine family.
func (scenario GetStartedClusters) CreateParameterGroup(ctx context.Context,
 dbEngine string,
 parameterGroupName string) *types.DBClusterParameterGroup {

 log.Printf("Checking for an existing DB cluster parameter group named %v.\n",
  parameterGroupName)
 parameterGroup, err := scenario.dbClusters.GetParameterGroup(ctx,
 parameterGroupName)
 if err != nil {
  panic(err)
 }
 if parameterGroup == nil {
  log.Printf("Getting available database engine versions for %v.\n", dbEngine)
  engineVersions, err := scenario.dbClusters.GetEngineVersions(ctx, dbEngine, "")
  if err != nil {
   panic(err)
  }

  familySet := map[string]struct{}{}
```

```
  for _, family := range engineVersions {
   familySet[*family.DBParameterGroupFamily] = struct{}{}
  }
  var families []string
  for family := range familySet {
   families = append(families, family)
  }
  sort.Strings(families)
  familyIndex := scenario.questioner.AskChoice("Which family do you want to use?
\n", families)
  log.Println("Creating a DB cluster parameter group.")
  _, err = scenario.dbClusters.CreateParameterGroup(
   ctx, parameterGroupName, families[familyIndex], "Example parameter group.")
  if err != nil {
   panic(err)
  }
  parameterGroup, err = scenario.dbClusters.GetParameterGroup(ctx,
 parameterGroupName)
  if err != nil {
   panic(err)
  }
 }
 log.Printf("Parameter group %v:\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tName: %v\n", *parameterGroup.DBClusterParameterGroupName)
 log.Printf("\tARN: %v\n", *parameterGroup.DBClusterParameterGroupArn)
 log.Printf("\tFamily: %v\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tDescription: %v\n", *parameterGroup.Description)
 log.Println(strings.Repeat("-", 88))
 return parameterGroup

}

// SetUserParameters shows how to get the parameters contained in a custom
 parameter
// group and update some of the parameter values in the group.
func (scenario GetStartedClusters) SetUserParameters(ctx context.Context,
 parameterGroupName string) {
 log.Println("Let's set some parameter values in your parameter group.")
 dbParameters, err := scenario.dbClusters.GetParameters(ctx, parameterGroupName,
 "")
 if err != nil {
  panic(err)
 }
 var updateParams []types.Parameter
```

```
 for _, dbParam := range dbParameters {
  if strings.HasPrefix(*dbParam.ParameterName, "auto_increment") &&
   *dbParam.IsModifiable && *dbParam.DataType == "integer" {
   log.Printf("The %v parameter is described as:\n\t%v",
    *dbParam.ParameterName, *dbParam.Description)
   rangeSplit := strings.Split(*dbParam.AllowedValues, "-")
   lower, _ := strconv.Atoi(rangeSplit[0])
   upper, _ := strconv.Atoi(rangeSplit[1])
   newValue := scenario.questioner.AskInt(
    fmt.Sprintf("Enter a value between %v and %v:", lower, upper),
    demotools.InIntRange{Lower: lower, Upper: upper})
   dbParam.ParameterValue = aws.String(strconv.Itoa(newValue))
   updateParams = append(updateParams, dbParam)
  }
 }
 err = scenario.dbClusters.UpdateParameters(ctx, parameterGroupName,
 updateParams)
 if err != nil {
  panic(err)
 }
 log.Println("You can get a list of parameters you've set by specifying a source
 of 'user'.")
 userParameters, err := scenario.dbClusters.GetParameters(ctx,
 parameterGroupName, "user")
 if err != nil {
  panic(err)
 }
 log.Println("Here are the parameters you've set:")
 for _, param := range userParameters {
  log.Printf("\t%v: %v\n", *param.ParameterName, *param.ParameterValue)
 }
 log.Println(strings.Repeat("-", 88))
}

// CreateCluster shows how to create an Aurora DB cluster that contains a
 database
// of a specified type. The database is also configured to use a custom DB
 cluster
// parameter group.
func (scenario GetStartedClusters) CreateCluster(ctx context.Context, clusterName
 string, dbEngine string,
 dbName string, parameterGroup *types.DBClusterParameterGroup) *types.DBCluster {

 log.Println("Checking for an existing DB cluster.")
```

```
 cluster, err := scenario.dbClusters.GetDbCluster(ctx, clusterName)
 if err != nil {
  panic(err)
 }
 if cluster == nil {
  adminUsername := scenario.questioner.Ask(
   "Enter an administrator user name for the database: ", demotools.NotEmpty{})
  adminPassword := scenario.questioner.Ask(
   "Enter a password for the administrator (at least 8 characters): ",
 demotools.NotEmpty{})
  engineVersions, err := scenario.dbClusters.GetEngineVersions(ctx, dbEngine,
 *parameterGroup.DBParameterGroupFamily)
  if err != nil {
   panic(err)
  }
  var engineChoices []string
  for _, engine := range engineVersions {
   engineChoices = append(engineChoices, *engine.EngineVersion)
  }
  log.Println("The available engines for your parameter group are:")
  engineIndex := scenario.questioner.AskChoice("Which engine do you want to use?
\n", engineChoices)
  log.Printf("Creating DB cluster %v and database %v.\n", clusterName, dbName)
  log.Printf("The DB cluster is configured to use\nyour custom parameter group %v
\n",
   *parameterGroup.DBClusterParameterGroupName)
  log.Printf("and selected engine %v.\n", engineChoices[engineIndex])
  log.Println("This typically takes several minutes.")
  cluster, err = scenario.dbClusters.CreateDbCluster(
   ctx, clusterName, *parameterGroup.DBClusterParameterGroupName, dbName,
 dbEngine,
   engineChoices[engineIndex], adminUsername, adminPassword)
  if err != nil {
   panic(err)
  }
  for *cluster.Status != "available" {
   scenario.helper.Pause(30)
   cluster, err = scenario.dbClusters.GetDbCluster(ctx, clusterName)
   if err != nil {
    panic(err)
   }
   log.Println("Cluster created and available.")
  }
 }
```

```
 log.Println("Cluster data:")
 log.Printf("\tDBClusterIdentifier: %v\n", *cluster.DBClusterIdentifier)
 log.Printf("\tARN: %v\n", *cluster.DBClusterArn)
 log.Printf("\tStatus: %v\n", *cluster.Status)
 log.Printf("\tEngine: %v\n", *cluster.Engine)
 log.Printf("\tEngine version: %v\n", *cluster.EngineVersion)
 log.Printf("\tDBClusterParameterGroup: %v\n", *cluster.DBClusterParameterGroup)
 log.Printf("\tEngineMode: %v\n", *cluster.EngineMode)
 log.Println(strings.Repeat("-", 88))
 return cluster
}

// CreateInstance shows how to create a DB instance in an existing Aurora DB
 cluster.
// A new DB cluster contains no DB instances, so you must add one. The first DB
 instance
// that is added to a DB cluster defaults to a read-write DB instance.
func (scenario GetStartedClusters) CreateInstance(ctx context.Context, cluster
 *types.DBCluster) *types.DBInstance {
 log.Println("Checking for an existing database instance.")
 dbInstance, err := scenario.dbClusters.GetInstance(ctx,
 *cluster.DBClusterIdentifier)
 if err != nil {
  panic(err)
 }
 if dbInstance == nil {
  log.Println("Let's create a database instance in your DB cluster.")
  log.Println("First, choose a DB instance type:")
  instOpts, err := scenario.dbClusters.GetOrderableInstances(
   ctx, *cluster.Engine, *cluster.EngineVersion)
  if err != nil {
   panic(err)
  }
  var instChoices []string
  for _, opt := range instOpts {
   instChoices = append(instChoices, *opt.DBInstanceClass)
  }
  slices.Sort(instChoices)
  instChoices = slices.Compact(instChoices)
  instIndex := scenario.questioner.AskChoice(
   "Which DB instance class do you want to use?\n", instChoices)
  log.Println("Creating a database instance. This typically takes several
 minutes.")
  dbInstance, err = scenario.dbClusters.CreateInstanceInCluster(
```

```
    ctx, *cluster.DBClusterIdentifier, *cluster.DBClusterIdentifier,
  *cluster.Engine,
    instChoices[instIndex])
   if err != nil {
    panic(err)
   }
   for *dbInstance.DBInstanceStatus != "available" {
    scenario.helper.Pause(30)
    dbInstance, err = scenario.dbClusters.GetInstance(ctx,
  *cluster.DBClusterIdentifier)
    if err != nil {
     panic(err)
    }
   }
  }
  log.Println("Instance data:")
  log.Printf("\tDBInstanceIdentifier: %v\n", *dbInstance.DBInstanceIdentifier)
  log.Printf("\tARN: %v\n", *dbInstance.DBInstanceArn)
  log.Printf("\tStatus: %v\n", *dbInstance.DBInstanceStatus)
  log.Printf("\tEngine: %v\n", *dbInstance.Engine)
  log.Printf("\tEngine version: %v\n", *dbInstance.EngineVersion)
  log.Println(strings.Repeat("-", 88))
  return dbInstance
}

// DisplayConnection displays connection information about an Aurora DB cluster
  and tips
// on how to connect to it.
func (scenario GetStartedClusters) DisplayConnection(cluster *types.DBCluster) {
  log.Println(
   "You can now connect to your database using your favorite MySql client.\n" +
    "One way to connect is by using the 'mysql' shell on an Amazon EC2 instance\n"
  +
    "that is running in the same VPC as your database cluster. Pass the endpoint,
\n" +
    "port, and administrator user name to 'mysql' and enter your password\n" +
    "when prompted:")
  log.Printf("\n\tmysql -h %v -P %v -u %v -p\n",
   *cluster.Endpoint, *cluster.Port, *cluster.MasterUsername)
  log.Println("For more information, see the User Guide for Aurora:\n" +
   "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora
  log.Println(strings.Repeat("-", 88))
}
```

```
// CreateSnapshot shows how to create a DB cluster snapshot and wait until it's
 available.
func (scenario GetStartedClusters) CreateSnapshot(ctx context.Context,
 clusterName string) {
 if scenario.questioner.AskBool(
  "Do you want to create a snapshot of your DB cluster (y/n)? ", "y") {
  snapshotId := fmt.Sprintf("%v-%v", clusterName, scenario.helper.UniqueId())
  log.Printf("Creating a snapshot named %v. This typically takes a few minutes.
\n", snapshotId)
  snapshot, err := scenario.dbClusters.CreateClusterSnapshot(ctx, clusterName,
 snapshotId)
  if err != nil {
   panic(err)
  }
  for *snapshot.Status != "available" {
   scenario.helper.Pause(30)
   snapshot, err = scenario.dbClusters.GetClusterSnapshot(ctx, snapshotId)
   if err != nil {
    panic(err)
   }
  }
  log.Println("Snapshot data:")
  log.Printf("\tDBClusterSnapshotIdentifier: %v\n",
 *snapshot.DBClusterSnapshotIdentifier)
  log.Printf("\tARN: %v\n", *snapshot.DBClusterSnapshotArn)
  log.Printf("\tStatus: %v\n", *snapshot.Status)
  log.Printf("\tEngine: %v\n", *snapshot.Engine)
  log.Printf("\tEngine version: %v\n", *snapshot.EngineVersion)
  log.Printf("\tDBClusterIdentifier: %v\n", *snapshot.DBClusterIdentifier)
  log.Printf("\tSnapshotCreateTime: %v\n", *snapshot.SnapshotCreateTime)
  log.Println(strings.Repeat("-", 88))
 }
}

// Cleanup shows how to clean up a DB instance, DB cluster, and DB cluster
 parameter group.
// Before the DB cluster parameter group can be deleted, all associated DB
 instances and
// DB clusters must first be deleted.
func (scenario GetStartedClusters) Cleanup(ctx context.Context, dbInstance
 *types.DBInstance, cluster *types.DBCluster,
 parameterGroup *types.DBClusterParameterGroup) {
```

```
 if scenario.questioner.AskBool(
  "\nDo you want to delete the database instance, DB cluster, and parameter group
 (y/n)? ", "y") {
  log.Printf("Deleting database instance %v.\n",
 *dbInstance.DBInstanceIdentifier)
  err := scenario.dbClusters.DeleteInstance(ctx,
 *dbInstance.DBInstanceIdentifier)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleting database cluster %v.\n", *cluster.DBClusterIdentifier)
  err = scenario.dbClusters.DeleteDbCluster(ctx, *cluster.DBClusterIdentifier)
  if err != nil {
   panic(err)
  }
  log.Println(
   "Waiting for the DB instance and DB cluster to delete. This typically takes
 several minutes.")
  for dbInstance != nil || cluster != nil {
   scenario.helper.Pause(30)
   if dbInstance != nil {
    dbInstance, err = scenario.dbClusters.GetInstance(ctx,
 *dbInstance.DBInstanceIdentifier)
    if err != nil {
     panic(err)
    }
   }
   if cluster != nil {
    cluster, err = scenario.dbClusters.GetDbCluster(ctx,
 *cluster.DBClusterIdentifier)
    if err != nil {
     panic(err)
    }
   }
  }
  log.Printf("Deleting parameter group %v.",
 *parameterGroup.DBClusterParameterGroupName)
  err = scenario.dbClusters.DeleteParameterGroup(ctx,
 *parameterGroup.DBClusterParameterGroupName)
  if err != nil {
   panic(err)
  }
 }
}
```

```
// IScenarioHelper abstracts the function from a scenario so that it
// can be mocked for unit testing.
type IScenarioHelper interface {
 Pause(secs int)
 UniqueId() string
}
type ScenarioHelper struct{}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}

// UniqueId returns a new UUID.
func (helper ScenarioHelper) UniqueId() string {
 return uuid.New().String()
}
```

定義案例所呼叫的函數以管理 Amazon 動作。

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}


// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
```

```
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  ctx, &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBClusterParameterGroups[0], err
 }
}


// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBClusterParameterGroup, error) {

 output, err := clusters.AuroraClient.CreateDBClusterParameterGroup(ctx,
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}



// DeleteParameterGroup deletes the named DB cluster parameter group.
```

```
func (clusters *DbClusters) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(ctx,
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}



// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(ctx context.Context, parameterGroupName
 string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}
```

```
// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(ctx context.Context,
 parameterGroupName string, params []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(ctx,
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}



// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(ctx context.Context, clusterName string)
 (*types.DBCluster, error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(ctx,
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
```

```
// The newly created DB cluster contains a database that uses the specified
 engine and
// engine version.
func (clusters *DbClusters) CreateDbCluster(ctx context.Context, clusterName
 string, parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(ctx,
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(ctx context.Context, clusterName
 string) error {
 _, err := clusters.AuroraClient.DeleteDBCluster(ctx, &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}
```

```
// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(ctx context.Context,
 clusterName string, snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(ctx,
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBClusterSnapshot, nil
 }
}



// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(ctx context.Context, snapshotName
 string) (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(ctx,
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(ctx context.Context,
 clusterName string, instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
```

```
 output, err := clusters.AuroraClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}



// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(ctx context.Context, instanceName string)
 (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(ctx context.Context, instanceName
 string) error {
```

```
 _, err := clusters.AuroraClient.DeleteDBInstance(ctx,
 &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(ctx context.Context, engine string,
 parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {
```

```
 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
  }
 }
 return instances, err
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的下列主題。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances

- DescribeOrderableDBInstanceOptions
- ModifyDBClusterParameterGroup

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/**
 * Before running this Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * This example requires an AWS Secrets Manager secret that contains the
 * database credentials. If you do not create a
 * secret, this example will not work. For details, see:
 *
 * https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html
 *
 * This Java example performs the following tasks:
 *
 * 1. Gets available engine families for Amazon Aurora MySQL-Compatible Edition
 * by calling the DescribeDbEngineVersions(Engine='aurora-mysql') method.
 * 2. Selects an engine family and creates a custom DB cluster parameter group
 * by invoking the describeDBClusterParameters method.
 * 3. Gets the parameter groups by invoking the describeDBClusterParameterGroups
 * method.
 * 4. Gets parameters in the group by invoking the describeDBClusterParameters
 * method.
 * 5. Modifies the auto_increment_offset parameter by invoking the
 * modifyDbClusterParameterGroupRequest method.
```

```
 * 6. Gets and displays the updated parameters.
 * 7. Gets a list of allowed engine versions by invoking the
 * describeDbEngineVersions method.
 * 8. Creates an Aurora DB cluster database cluster that contains a MySQL
 * database.
 * 9. Waits for DB instance to be ready.
 * 10. Gets a list of instance classes available for the selected engine.
 * 11. Creates a database instance in the cluster.
 * 12. Waits for DB instance to be ready.
 * 13. Creates a snapshot.
 * 14. Waits for DB snapshot to be ready.
 * 15. Deletes the DB cluster.
 * 16. Deletes the DB cluster group.
 */
public class AuroraScenario {
    public static long sleepTime = 20;
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws InterruptedException {
        final String usage = "\n" +
                "Usage:\n" +
                "    <dbClusterGroupName> <dbParameterGroupFamily>
 <dbInstanceClusterIdentifier> <dbInstanceIdentifier> <dbName>
 <dbSnapshotIdentifier><secretName>"
                +
                "Where:\n" +
                "    dbClusterGroupName - The name of the DB cluster parameter
 group. \n" +
                "    dbParameterGroupFamily - The DB cluster parameter group
 family name (for example, aurora-mysql5.7). \n"
                +
                "    dbInstanceClusterIdentifier - The instance cluster
 identifier value.\n" +
                "    dbInstanceIdentifier - The database instance identifier.\n"
 +
                "    dbName - The database name.\n" +
                "    dbSnapshotIdentifier - The snapshot identifier.\n" +
                "    secretName - The name of the AWS Secrets Manager secret that
 contains the database credentials\"\n";
        ;

        if (args.length != 7) {
            System.out.println(usage);
```

```
            System.exit(1);
        }

        String dbClusterGroupName = args[0];
        String dbParameterGroupFamily = args[1];
        String dbInstanceClusterIdentifier = args[2];
        String dbInstanceIdentifier = args[3];
        String dbName = args[4];
        String dbSnapshotIdentifier = args[5];
        String secretName = args[6];

        // Retrieve the database credentials using AWS Secrets Manager.
        Gson gson = new Gson();
        User user = gson.fromJson(String.valueOf(getSecretValues(secretName)),
User.class);
        String username = user.getUsername();
        String userPassword = user.getPassword();

        Region region = Region.US_WEST_2;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon Aurora example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("1. Return a list of the available DB engines");
        describeDBEngines(rdsClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Create a custom parameter group");
        createDBClusterParameterGroup(rdsClient, dbClusterGroupName,
dbParameterGroupFamily);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Get the parameter group");
        describeDbClusterParameterGroups(rdsClient, dbClusterGroupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
```

```
        System.out.println("4. Get the parameters in the group");
        describeDbClusterParameters(rdsClient, dbClusterGroupName, 0);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Modify the auto_increment_offset parameter");
        modifyDBClusterParas(rdsClient, dbClusterGroupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Display the updated parameter value");
        describeDbClusterParameters(rdsClient, dbClusterGroupName, -1);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Get a list of allowed engine versions");
        getAllowedEngines(rdsClient, dbParameterGroupFamily);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Create an Aurora DB cluster database");
        String arnClusterVal = createDBCluster(rdsClient, dbClusterGroupName,
dbName, dbInstanceClusterIdentifier,
                username, userPassword);
        System.out.println("The ARN of the cluster is " + arnClusterVal);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Wait for DB instance to be ready");
        waitForInstanceReady(rdsClient, dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Get a list of instance classes available for the
selected engine");
        String instanceClass = getListInstanceClasses(rdsClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Create a database instance in the cluster.");
        String clusterDBARN = createDBInstanceCluster(rdsClient,
dbInstanceIdentifier, dbInstanceClusterIdentifier,
                instanceClass);
        System.out.println("The ARN of the database is " + clusterDBARN);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Wait for DB instance to be ready");
        waitDBInstanceReady(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("13. Create a snapshot");
        createDBClusterSnapshot(rdsClient, dbInstanceClusterIdentifier,
dbSnapshotIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Wait for DB snapshot to be ready");
        waitForSnapshotReady(rdsClient, dbSnapshotIdentifier,
dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Delete the DB instance");
        deleteDatabaseInstance(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("15. Delete the DB cluster");
        deleteCluster(rdsClient, dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("16. Delete the DB cluster group");
        deleteDBClusterGroup(rdsClient, dbClusterGroupName, clusterDBARN);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The Scenario has successfully completed.");
        System.out.println(DASHES);
        rdsClient.close();
    }

    private static SecretsManagerClient getSecretClient() {
        Region region = Region.US_WEST_2;
        return SecretsManagerClient.builder()
                .region(region)
```

```
.credentialsProvider(EnvironmentVariableCredentialsProvider.create())
                .build();
   }

   private static String getSecretValues(String secretName) {
       SecretsManagerClient secretClient = getSecretClient();
       GetSecretValueRequest valueRequest = GetSecretValueRequest.builder()
               .secretId(secretName)
               .build();

       GetSecretValueResponse valueResponse =
secretClient.getSecretValue(valueRequest);
       return valueResponse.secretString();
   }

   public static void deleteDBClusterGroup(RdsClient rdsClient, String
dbClusterGroupName, String clusterDBARN)
           throws InterruptedException {
       try {
           boolean isDataDel = false;
           boolean didFind;
           String instanceARN;

           // Make sure that the database has been deleted.
           while (!isDataDel) {
               DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
               List<DBInstance> instanceList = response.dbInstances();
               int listSize = instanceList.size();
               didFind = false;
               int index = 1;
               for (DBInstance instance : instanceList) {
                   instanceARN = instance.dbInstanceArn();
                   if (instanceARN.compareTo(clusterDBARN) == 0) {
                       System.out.println(clusterDBARN + " still exists");
                       didFind = true;
                   }
                   if ((index == listSize) && (!didFind)) {
                       // Went through the entire list and did not find the
database ARN.
                       isDataDel = true;
                   }
                   Thread.sleep(sleepTime * 1000);
```

```
                    index++;
                }
            }

            DeleteDbClusterParameterGroupRequest clusterParameterGroupRequest =
DeleteDbClusterParameterGroupRequest
                    .builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .build();


rdsClient.deleteDBClusterParameterGroup(clusterParameterGroupRequest);
            System.out.println(dbClusterGroupName + " was deleted.");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void deleteCluster(RdsClient rdsClient, String
dbInstanceClusterIdentifier) {
        try {
            DeleteDbClusterRequest deleteDbClusterRequest =
DeleteDbClusterRequest.builder()
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .skipFinalSnapshot(true)
                    .build();

            rdsClient.deleteDBCluster(deleteDbClusterRequest);
            System.out.println(dbInstanceClusterIdentifier + " was deleted!");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void deleteDatabaseInstance(RdsClient rdsClient, String
dbInstanceIdentifier) {
        try {
            DeleteDbInstanceRequest deleteDbInstanceRequest =
DeleteDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
```

```
                    .deleteAutomatedBackups(true)
                    .skipFinalSnapshot(true)
                    .build();

            DeleteDbInstanceResponse response =
rdsClient.deleteDBInstance(deleteDbInstanceRequest);
            System.out.println("The status of the database is " +
response.dbInstance().dbInstanceStatus());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void waitForSnapshotReady(RdsClient rdsClient, String
dbSnapshotIdentifier,
            String dbInstanceClusterIdentifier) {
        try {
            boolean snapshotReady = false;
            String snapshotReadyStr;
            System.out.println("Waiting for the snapshot to become available.");

            DescribeDbClusterSnapshotsRequest snapshotsRequest =
DescribeDbClusterSnapshotsRequest.builder()
                    .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .build();

            while (!snapshotReady) {
                DescribeDbClusterSnapshotsResponse response =
rdsClient.describeDBClusterSnapshots(snapshotsRequest);
                List<DBClusterSnapshot> snapshotList =
response.dbClusterSnapshots();
                for (DBClusterSnapshot snapshot : snapshotList) {
                    snapshotReadyStr = snapshot.status();
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true;
                    } else {
                        System.out.println(".");
                        Thread.sleep(sleepTime * 5000);
                    }
                }
            }
```

```
            System.out.println("The Snapshot is available!");

        } catch (RdsException | InterruptedException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void createDBClusterSnapshot(RdsClient rdsClient, String
dbInstanceClusterIdentifier,
            String dbSnapshotIdentifier) {
        try {
            CreateDbClusterSnapshotRequest snapshotRequest =
CreateDbClusterSnapshotRequest.builder()
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                    .build();

            CreateDbClusterSnapshotResponse response =
rdsClient.createDBClusterSnapshot(snapshotRequest);
            System.out.println("The Snapshot ARN is " +
response.dbClusterSnapshot().dbClusterSnapshotArn());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void waitDBInstanceReady(RdsClient rdsClient, String
dbInstanceIdentifier) {
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbInstancesRequest instanceRequest =
DescribeDbInstancesRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .build();

            String endpoint = "";
            while (!instanceReady) {
```

```
                DescribeDbInstancesResponse response =
rdsClient.describeDBInstances(instanceRequest);
                List<DBInstance> instanceList = response.dbInstances();
                for (DBInstance instance : instanceList) {
                    instanceReadyStr = instance.dbInstanceStatus();
                    if (instanceReadyStr.contains("available")) {
                        endpoint = instance.endpoint().address();
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database instance is available! The connection
endpoint is " + endpoint);

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static String createDBInstanceCluster(RdsClient rdsClient,
            String dbInstanceIdentifier,
            String dbInstanceClusterIdentifier,
            String instanceClass) {
        try {
            CreateDbInstanceRequest instanceRequest =
CreateDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .engine("aurora-mysql")
                    .dbInstanceClass(instanceClass)
                    .build();

            CreateDbInstanceResponse response =
rdsClient.createDBInstance(instanceRequest);
            System.out.print("The status is " +
response.dbInstance().dbInstanceStatus());
            return response.dbInstance().dbInstanceArn();

        } catch (RdsException e) {
            System.err.println(e.getMessage());
```

```
            System.exit(1);
        }
        return "";
    }

    public static String getListInstanceClasses(RdsClient rdsClient) {
        try {
            DescribeOrderableDbInstanceOptionsRequest optionsRequest =
DescribeOrderableDbInstanceOptionsRequest
                    .builder()
                    .engine("aurora-mysql")
                    .maxRecords(20)
                    .build();

            DescribeOrderableDbInstanceOptionsResponse response = rdsClient
                    .describeOrderableDBInstanceOptions(optionsRequest);
            List<OrderableDBInstanceOption> instanceOptions =
response.orderableDBInstanceOptions();
            String instanceClass = "";
            for (OrderableDBInstanceOption instanceOption : instanceOptions) {
                instanceClass = instanceOption.dbInstanceClass();
                System.out.println("The instance class is " +
instanceOption.dbInstanceClass());
                System.out.println("The engine version is " +
instanceOption.engineVersion());
            }
            return instanceClass;

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }

    // Waits until the database instance is available.
    public static void waitForInstanceReady(RdsClient rdsClient, String
dbClusterIdentifier) {
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbClustersRequest instanceRequest =
DescribeDbClustersRequest.builder()
```

```
                    .dbClusterIdentifier(dbClusterIdentifier)
                    .build();

            while (!instanceReady) {
                DescribeDbClustersResponse response =
rdsClient.describeDBClusters(instanceRequest);
                List<DBCluster> clusterList = response.dbClusters();
                for (DBCluster cluster : clusterList) {
                    instanceReadyStr = cluster.status();
                    if (instanceReadyStr.contains("available")) {
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database cluster is available!");

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static String createDBCluster(RdsClient rdsClient, String
dbParameterGroupFamily, String dbName,
            String dbClusterIdentifier, String userName, String password) {
        try {
            CreateDbClusterRequest clusterRequest =
CreateDbClusterRequest.builder()
                    .databaseName(dbName)
                    .dbClusterIdentifier(dbClusterIdentifier)
                    .dbClusterParameterGroupName(dbParameterGroupFamily)
                    .engine("aurora-mysql")
                    .masterUsername(userName)
                    .masterUserPassword(password)
                    .build();

            CreateDbClusterResponse response =
rdsClient.createDBCluster(clusterRequest);
            return response.dbCluster().dbClusterArn();

        } catch (RdsException e) {
```

```
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
        return "";
    }

    // Get a list of allowed engine versions.
    public static void getAllowedEngines(RdsClient rdsClient, String
dbParameterGroupFamily) {
        try {
            DescribeDbEngineVersionsRequest versionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .engine("aurora-mysql")
                    .build();

            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(versionsRequest);
            List<DBEngineVersion> dbEngines = response.dbEngineVersions();
            for (DBEngineVersion dbEngine : dbEngines) {
                System.out.println("The engine version is " +
dbEngine.engineVersion());
                System.out.println("The engine description is " +
dbEngine.dbEngineDescription());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    // Modify the auto_increment_offset parameter.
    public static void modifyDBClusterParas(RdsClient rdsClient, String
dClusterGroupName) {
        try {
            Parameter parameter1 = Parameter.builder()
                    .parameterName("auto_increment_offset")
                    .applyMethod("immediate")
                    .parameterValue("5")
                    .build();

            List<Parameter> paraList = new ArrayList<>();
            paraList.add(parameter1);
```

```
            ModifyDbClusterParameterGroupRequest groupRequest =
ModifyDbClusterParameterGroupRequest.builder()
                    .dbClusterParameterGroupName(dClusterGroupName)
                    .parameters(paraList)
                    .build();

            ModifyDbClusterParameterGroupResponse response =
rdsClient.modifyDBClusterParameterGroup(groupRequest);
            System.out.println(
                    "The parameter group " +
response.dbClusterParameterGroupName() + " was successfully modified");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
        try {
            DescribeDbClusterParametersRequest dbParameterGroupsRequest;
            if (flag == 0) {
                dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                        .dbClusterParameterGroupName(dbCLusterGroupName)
                        .build();
            } else {
                dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                        .dbClusterParameterGroupName(dbCLusterGroupName)
                        .source("user")
                        .build();
            }

            DescribeDbClusterParametersResponse response = rdsClient
                    .describeDBClusterParameters(dbParameterGroupsRequest);
            List<Parameter> dbParameters = response.parameters();
            String paraName;
            for (Parameter para : dbParameters) {
                // Only print out information about either auto_increment_offset
or
                // auto_increment_increment.
                paraName = para.parameterName();
```

```
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
        try {
            DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .maxRecords(20)
                    .build();

            List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                    .dbClusterParameterGroups();
            for (DBClusterParameterGroup group : groups) {
                System.out.println("The group name is " +
group.dbClusterParameterGroupName());
                System.out.println("The group ARN is " +
group.dbClusterParameterGroupArn());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
```

```
    }

    public static void createDBClusterParameterGroup(RdsClient rdsClient, String
dbClusterGroupName,
            String dbParameterGroupFamily) {
        try {
            CreateDbClusterParameterGroupRequest groupRequest =
CreateDbClusterParameterGroupRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .description("Created by using the AWS SDK for Java")
                    .build();

            CreateDbClusterParameterGroupResponse response =
rdsClient.createDBClusterParameterGroup(groupRequest);
            System.out.println("The group name is " +
response.dbClusterParameterGroup().dbClusterParameterGroupName());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDBEngines(RdsClient rdsClient) {
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .engine("aurora-mysql")
                    .defaultOnly(true)
                    .maxRecords(20)
                    .build();

            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();

            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
the database engine is "
                        + engineOb.dbParameterGroupFamily());
                System.out.println("The name of the database engine " +
engineOb.engine());
```

```
                System.out.println("The version number of the database engine " +
 engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/**
Before running this Kotlin code example, set up your development environment,
 including your credentials.

For more information, see the following documentation topic:

https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

This example requires an AWS Secrets Manager secret that contains the database
 credentials. If you do not create a
secret, this example will not work. For more details, see:

https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html

This Kotlin example performs the following tasks:

1. Returns a list of the available DB engines.
2. Creates a custom DB parameter group.
3. Gets the parameter groups.
4. Gets the parameters in the group.
5. Modifies the auto_increment_increment parameter.
6. Displays the updated parameter value.
7. Gets a list of allowed engine versions.
8. Creates an Aurora DB cluster database.
9. Waits for DB instance to be ready.
10. Gets a list of instance classes available for the selected engine.
11. Creates a database instance in the cluster.
12. Waits for the database instance in the cluster to be ready.
13. Creates a snapshot.
14. Waits for DB snapshot to be ready.
```

```
15. Deletes the DB instance.
16. Deletes the DB cluster.
17. Deletes the DB cluster group.
 */

var slTime: Long = 20

suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <dbClusterGroupName> <dbParameterGroupFamily>
 <dbInstanceClusterIdentifier> <dbName> <dbSnapshotIdentifier> <secretName>
        Where:
            dbClusterGroupName - The database group name.
            dbParameterGroupFamily - The database parameter group name.
            dbInstanceClusterIdentifier - The database instance identifier.
            dbName -  The database name.
            dbSnapshotIdentifier - The snapshot identifier.
            secretName - The name of the AWS Secrets Manager secret that contains
 the database credentials.
    """

    if (args.size != 7) {
        println(usage)
        exitProcess(1)
    }

    val dbClusterGroupName = args[0]
    val dbParameterGroupFamily = args[1]
    val dbInstanceClusterIdentifier = args[2]
    val dbInstanceIdentifier = args[3]
    val dbName = args[4]
    val dbSnapshotIdentifier = args[5]
    val secretName = args[6]

    val gson = Gson()
    val user = gson.fromJson(getSecretValues(secretName).toString(),
 User::class.java)
    val username = user.username
    val userPassword = user.password

    println("1. Return a list of the available DB engines")
    describeAuroraDBEngines()
```

```
    println("2. Create a custom parameter group")
    createDBClusterParameterGroup(dbClusterGroupName, dbParameterGroupFamily)

    println("3. Get the parameter group")
    describeDbClusterParameterGroups(dbClusterGroupName)

    println("4. Get the parameters in the group")
    describeDbClusterParameters(dbClusterGroupName, 0)

    println("5. Modify the auto_increment_offset parameter")
    modifyDBClusterParas(dbClusterGroupName)

    println("6. Display the updated parameter value")
    describeDbClusterParameters(dbClusterGroupName, -1)

    println("7. Get a list of allowed engine versions")
    getAllowedClusterEngines(dbParameterGroupFamily)

    println("8. Create an Aurora DB cluster database")
    val arnClusterVal = createDBCluster(dbClusterGroupName, dbName,
dbInstanceClusterIdentifier, username, userPassword)
    println("The ARN of the cluster is $arnClusterVal")

    println("9. Wait for DB instance to be ready")
    waitForClusterInstanceReady(dbInstanceClusterIdentifier)

    println("10. Get a list of instance classes available for the selected
engine")
    val instanceClass = getListInstanceClasses()

    println("11. Create a database instance in the cluster.")
    val clusterDBARN = createDBInstanceCluster(dbInstanceIdentifier,
dbInstanceClusterIdentifier, instanceClass)
    println("The ARN of the database is $clusterDBARN")

    println("12. Wait for DB instance to be ready")
    waitDBAuroraInstanceReady(dbInstanceIdentifier)

    println("13. Create a snapshot")
    createDBClusterSnapshot(dbInstanceClusterIdentifier, dbSnapshotIdentifier)

    println("14. Wait for DB snapshot to be ready")
    waitSnapshotReady(dbSnapshotIdentifier, dbInstanceClusterIdentifier)
```

```
    println("15. Delete the DB instance")
    deleteDBInstance(dbInstanceIdentifier)

    println("16. Delete the DB cluster")
    deleteCluster(dbInstanceClusterIdentifier)

    println("17. Delete the DB cluster group")
    if (clusterDBARN != null) {
        deleteDBClusterGroup(dbClusterGroupName, clusterDBARN)
    }
    println("The Scenario has successfully completed.")
}

@Throws(InterruptedException::class)
suspend fun deleteDBClusterGroup(
    dbClusterGroupName: String,
    clusterDBARN: String,
) {
    var isDataDel = false
    var didFind: Boolean
    var instanceARN: String

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        // Make sure that the database has been deleted.
        while (!isDataDel) {
            val response = rdsClient.describeDbInstances()
            val instanceList = response.dbInstances
            val listSize = instanceList?.size
            isDataDel = false
            didFind = false
            var index = 1
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceARN = instance.dbInstanceArn.toString()
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        println("$clusterDBARN still exists")
                        didFind = true
                    }
                    if (index == listSize && !didFind) {
                        // Went through the entire list and did not find the
 database ARN.
                        isDataDel = true
                    }
                    delay(slTime * 1000)
```

```
                    index++
                }
            }
        }
        val clusterParameterGroupRequest =
            DeleteDbClusterParameterGroupRequest {
                dbClusterParameterGroupName = dbClusterGroupName
            }

        rdsClient.deleteDbClusterParameterGroup(clusterParameterGroupRequest)
        println("$dbClusterGroupName was deleted.")
    }
}

suspend fun deleteCluster(dbInstanceClusterIdentifier: String) {
    val deleteDbClusterRequest =
        DeleteDbClusterRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            skipFinalSnapshot = true
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        rdsClient.deleteDbCluster(deleteDbClusterRequest)
        println("$dbInstanceClusterIdentifier was deleted!")
    }
}

suspend fun deleteDBInstance(dbInstanceIdentifierVal: String) {
    val deleteDbInstanceRequest =
        DeleteDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            deleteAutomatedBackups = true
            skipFinalSnapshot = true
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
    }
}

suspend fun waitSnapshotReady(
    dbSnapshotIdentifier: String?,
```

```
    dbInstanceClusterIdentifier: String?,
) {
    var snapshotReady = false
    var snapshotReadyStr: String
    println("Waiting for the snapshot to become available.")

    val snapshotsRequest =
        DescribeDbClusterSnapshotsRequest {
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
            dbClusterIdentifier = dbInstanceClusterIdentifier
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!snapshotReady) {
            val response = rdsClient.describeDbClusterSnapshots(snapshotsRequest)
            val snapshotList = response.dbClusterSnapshots
            if (snapshotList != null) {
                for (snapshot in snapshotList) {
                    snapshotReadyStr = snapshot.status.toString()
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true
                    } else {
                        println(".")
                        delay(slTime * 5000)
                    }
                }
            }
        }
    }
    println("The Snapshot is available!")
}

suspend fun createDBClusterSnapshot(
    dbInstanceClusterIdentifier: String?,
    dbSnapshotIdentifier: String?,
) {
    val snapshotRequest =
        CreateDbClusterSnapshotRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterSnapshot(snapshotRequest)
```

```
        println("The Snapshot ARN is
 ${response.dbClusterSnapshot?.dbClusterSnapshotArn}")
    }
}

suspend fun waitDBAuroraInstanceReady(dbInstanceIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")
    val instanceRequest =
        DescribeDbInstancesRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
        }

    var endpoint = ""
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbInstances(instanceRequest)
            response.dbInstances?.forEach { instance ->
                instanceReadyStr = instance.dbInstanceStatus.toString()
                if (instanceReadyStr.contains("available")) {
                    endpoint = instance.endpoint?.address.toString()
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database instance is available! The connection endpoint is
 $endpoint")
}

suspend fun createDBInstanceCluster(
    dbInstanceIdentifierVal: String?,
    dbInstanceClusterIdentifierVal: String?,
    instanceClassVal: String?,
): String? {
    val instanceRequest =
        CreateDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            dbClusterIdentifier = dbInstanceClusterIdentifierVal
            engine = "aurora-mysql"
```

```
            dbInstanceClass = instanceClassVal
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
        return response.dbInstance?.dbInstanceArn
    }
}

suspend fun getListInstanceClasses(): String {
    val optionsRequest =
        DescribeOrderableDbInstanceOptionsRequest {
            engine = "aurora-mysql"
            maxRecords = 20
        }
    var instanceClass = ""
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeOrderableDbInstanceOptions(optionsRequest)
        response.orderableDbInstanceOptions?.forEach { instanceOption ->
            instanceClass = instanceOption.dbInstanceClass.toString()
            println("The instance class is ${instanceOption.dbInstanceClass}")
            println("The engine version is ${instanceOption.engineVersion}")
        }
    }
    return instanceClass
}

// Waits until the database instance is available.
suspend fun waitForClusterInstanceReady(dbClusterIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")

    val instanceRequest =
        DescribeDbClustersRequest {
            dbClusterIdentifier = dbClusterIdentifierVal
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbClusters(instanceRequest)
            response.dbClusters?.forEach { cluster ->
```

```
                instanceReadyStr = cluster.status.toString()
                if (instanceReadyStr.contains("available")) {
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database cluster is available!")
}

suspend fun createDBCluster(
    dbParameterGroupFamilyVal: String?,
    dbName: String?,
    dbClusterIdentifierVal: String?,
    userName: String?,
    password: String?,
): String? {
    val clusterRequest =
        CreateDbClusterRequest {
            databaseName = dbName
            dbClusterIdentifier = dbClusterIdentifierVal
            dbClusterParameterGroupName = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
            masterUsername = userName
            masterUserPassword = password
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbCluster(clusterRequest)
        return response.dbCluster?.dbClusterArn
    }
}

// Get a list of allowed engine versions.
suspend fun getAllowedClusterEngines(dbParameterGroupFamilyVal: String?) {
    val versionsRequest =
        DescribeDbEngineVersionsRequest {
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
        }
```

```
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(versionsRequest)
        response.dbEngineVersions?.forEach { dbEngine ->
            println("The engine version is ${dbEngine.engineVersion}")
            println("The engine description is ${dbEngine.dbEngineDescription}")
        }
    }
}

// Modify the auto_increment_offset parameter.
suspend fun modifyDBClusterParas(dClusterGroupName: String?) {
    val parameter1 =
        Parameter {
            parameterName = "auto_increment_offset"
            applyMethod = ApplyMethod.fromValue("immediate")
            parameterValue = "5"
        }

    val paraList = ArrayList<Parameter>()
    paraList.add(parameter1)
    val groupRequest =
        ModifyDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dClusterGroupName
            parameters = paraList
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.modifyDbClusterParameterGroup(groupRequest)
        println("The parameter group ${response.dbClusterParameterGroupName} was
 successfully modified")
    }
}

suspend fun describeDbClusterParameters(
    dbCLusterGroupName: String?,
    flag: Int,
) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest =
        if (flag == 0) {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
            }
        } else {
```

```
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
                source = "user"
            }
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}

suspend fun describeDbClusterParameterGroups(dbClusterGroupName: String?) {
    val groupsRequest =
        DescribeDbClusterParameterGroupsRequest {
            dbClusterParameterGroupName = dbClusterGroupName
            maxRecords = 20
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbClusterParameterGroups(groupsRequest)
        response.dbClusterParameterGroups?.forEach { group ->
            println("The group name is ${group.dbClusterParameterGroupName}")
            println("The group ARN is ${group.dbClusterParameterGroupArn}")
        }
    }
}
```

```
suspend fun createDBClusterParameterGroup(
    dbClusterGroupNameVal: String?,
    dbParameterGroupFamilyVal: String?,
) {
    val groupRequest =
        CreateDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dbClusterGroupNameVal
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            description = "Created by using the AWS SDK for Kotlin"
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterParameterGroup(groupRequest)
        println("The group name is
 ${response.dbClusterParameterGroup?.dbClusterParameterGroupName}")
    }
}

suspend fun describeAuroraDBEngines() {
    val engineVersionsRequest =
        DescribeDbEngineVersionsRequest {
            engine = "aurora-mysql"
            defaultOnly = true
            maxRecords = 20
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(engineVersionsRequest)
        response.dbEngineVersions?.forEach { engineOb ->
            println("The name of the DB parameter group family for the database
 engine is ${engineOb.dbParameterGroupFamily}")
            println("The name of the database engine ${engineOb.engine}")
            println("The version number of the database engine
 ${engineOb.engineVersion}")
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Kotlin API 參考》中的下列主題。
  - CreateDBCluster

- CreateDBClusterParameterGroup
- CreateDBClusterSnapshot
- CreateDBInstance
- DeleteDBCluster
- DeleteDBClusterParameterGroup
- DeleteDBInstance
- DescribeDBClusterParameterGroups
- DescribeDBClusterParameters
- DescribeDBClusterSnapshots
- DescribeDBClusters
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeOrderableDBInstanceOptions
- ModifyDBClusterParameterGroup

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在命令提示中執行互動式案例。

```
class AuroraClusterScenario:
    """Runs a scenario that shows how to get started using Aurora DB clusters."""

    def __init__(self, aurora_wrapper):
        """
        :param aurora_wrapper: An object that wraps Aurora DB cluster actions.
        """
        self.aurora_wrapper = aurora_wrapper
```

```
    def create_parameter_group(self, db_engine, parameter_group_name):
        """
        Shows how to get available engine versions for a specified database
engine and
        create a DB cluster parameter group that is compatible with a selected
engine family.

        :param db_engine: The database engine to use as a basis.
        :param parameter_group_name: The name given to the newly created
parameter group.
        :return: The newly created parameter group.
        """
        print(
            f"Checking for an existing DB cluster parameter group named
{parameter_group_name}."
        )
        parameter_group =
self.aurora_wrapper.get_parameter_group(parameter_group_name)
        if parameter_group is None:
            print(f"Getting available database engine versions for {db_engine}.")
            engine_versions = self.aurora_wrapper.get_engine_versions(db_engine)
            families = list({ver["DBParameterGroupFamily"] for ver in
engine_versions})
            family_index = q.choose("Which family do you want to use? ",
families)
            print(f"Creating a DB cluster parameter group.")
            self.aurora_wrapper.create_parameter_group(
                parameter_group_name, families[family_index], "Example parameter
group."
            )
            parameter_group = self.aurora_wrapper.get_parameter_group(
                parameter_group_name
            )
        print(f"Parameter group
{parameter_group['DBClusterParameterGroupName']}:")
        pp(parameter_group)
        print("-" * 88)
        return parameter_group

    def set_user_parameters(self, parameter_group_name):
        """
        Shows how to get the parameters contained in a custom parameter group and
        update some of the parameter values in the group.
```

```
        :param parameter_group_name: The name of the parameter group to query and
modify.
        """
        print("Let's set some parameter values in your parameter group.")
        auto_inc_parameters = self.aurora_wrapper.get_parameters(
            parameter_group_name, name_prefix="auto_increment"
        )
        update_params = []
        for auto_inc in auto_inc_parameters:
            if auto_inc["IsModifiable"] and auto_inc["DataType"] == "integer":
                print(f"The {auto_inc['ParameterName']} parameter is described
as:")
                print(f"\t{auto_inc['Description']}")
                param_range = auto_inc["AllowedValues"].split("-")
                auto_inc["ParameterValue"] = str(
                    q.ask(
                        f"Enter a value between {param_range[0]} and
{param_range[1]}: ",
                        q.is_int,
                        q.in_range(int(param_range[0]), int(param_range[1])),
                    )
                )
                update_params.append(auto_inc)
        self.aurora_wrapper.update_parameters(parameter_group_name,
update_params)
        print(
            "You can get a list of parameters you've set by specifying a source
of 'user'."
        )
        user_parameters = self.aurora_wrapper.get_parameters(
            parameter_group_name, source="user"
        )
        pp(user_parameters)
        print("-" * 88)

    def create_cluster(self, cluster_name, db_engine, db_name, parameter_group):
        """
        Shows how to create an Aurora DB cluster that contains a database of a
specified
        type. The database is also configured to use a custom DB cluster
parameter group.

        :param cluster_name: The name given to the newly created DB cluster.
        :param db_engine: The engine of the created database.
```

```
        :param db_name: The name given to the created database.
        :param parameter_group: The parameter group that is associated with the
DB cluster.
        :return: The newly created DB cluster.
        """
        print("Checking for an existing DB cluster.")
        cluster = self.aurora_wrapper.get_db_cluster(cluster_name)
        if cluster is None:
            admin_username = q.ask(
                "Enter an administrator user name for the database: ",
q.non_empty
            )
            admin_password = q.ask(
                "Enter a password for the administrator (at least 8 characters):
",
                q.non_empty,
            )
            engine_versions = self.aurora_wrapper.get_engine_versions(
                db_engine, parameter_group["DBParameterGroupFamily"]
            )
            engine_choices = [
                ver["EngineVersionDescription"] for ver in engine_versions
            ]
            print("The available engines for your parameter group are:")
            engine_index = q.choose("Which engine do you want to use? ",
engine_choices)
            print(
                f"Creating DB cluster {cluster_name} and database {db_name}.\n"
                f"The DB cluster is configured to use\n"
                f"your custom parameter group
{parameter_group['DBClusterParameterGroupName']}\n"
                f"and selected engine {engine_choices[engine_index]}.\n"
                f"This typically takes several minutes."
            )
            cluster = self.aurora_wrapper.create_db_cluster(
                cluster_name,
                parameter_group["DBClusterParameterGroupName"],
                db_name,
                db_engine,
                engine_versions[engine_index]["EngineVersion"],
                admin_username,
                admin_password,
            )
            while cluster.get("Status") != "available":
```

```
                wait(30)
                cluster = self.aurora_wrapper.get_db_cluster(cluster_name)
            print("Cluster created and available.\n")
        print("Cluster data:")
        pp(cluster)
        print("-" * 88)
        return cluster

    def create_instance(self, cluster):
        """
        Shows how to create a DB instance in an existing Aurora DB cluster. A new
DB cluster
        contains no DB instances, so you must add one. The first DB instance that
is added
        to a DB cluster defaults to a read-write DB instance.

        :param cluster: The DB cluster where the DB instance is added.
        :return: The newly created DB instance.
        """
        print("Checking for an existing database instance.")
        cluster_name = cluster["DBClusterIdentifier"]
        db_inst = self.aurora_wrapper.get_db_instance(cluster_name)
        if db_inst is None:
            print("Let's create a database instance in your DB cluster.")
            print("First, choose a DB instance type:")
            inst_opts = self.aurora_wrapper.get_orderable_instances(
                cluster["Engine"], cluster["EngineVersion"]
            )
            inst_choices = list(
                {
                    opt["DBInstanceClass"] + ", storage type: " +
opt["StorageType"]
                    for opt in inst_opts
                }
            )
            inst_index = q.choose(
                "Which DB instance class do you want to use? ", inst_choices
            )
            print(
                f"Creating a database instance. This typically takes several
minutes."
            )
            db_inst = self.aurora_wrapper.create_instance_in_cluster(
                cluster_name,
```

```
                cluster_name,
                cluster["Engine"],
                inst_opts[inst_index]["DBInstanceClass"],
            )
            while db_inst.get("DBInstanceStatus") != "available":
                wait(30)
                db_inst = self.aurora_wrapper.get_db_instance(cluster_name)
        print("Instance data:")
        pp(db_inst)
        print("-" * 88)
        return db_inst

    @staticmethod
    def display_connection(cluster):
        """
        Displays connection information about an Aurora DB cluster and tips on
 how to
        connect to it.

        :param cluster: The DB cluster to display.
        """
        print(
            "You can now connect to your database using your favorite MySql
 client.\n"
            "One way to connect is by using the 'mysql' shell on an Amazon EC2
 instance\n"
            "that is running in the same VPC as your database cluster. Pass the
 endpoint,\n"
            "port, and administrator user name to 'mysql' and enter your password
\n"
            "when prompted:\n"
        )
        print(
            f"\n\tmysql -h {cluster['Endpoint']} -P {cluster['Port']} -u
 {cluster['MasterUsername']} -p\n"
        )
        print(
            "For more information, see the User Guide for Aurora:\n"
            "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora
        )
        print("-" * 88)

    def create_snapshot(self, cluster_name):
```

```
        """
        Shows how to create a DB cluster snapshot and wait until it's available.

        :param cluster_name: The name of a DB cluster to snapshot.
        """
        if q.ask(
            "Do you want to create a snapshot of your DB cluster (y/n)? ",
q.is_yesno
        ):
            snapshot_id = f"{cluster_name}-{uuid.uuid4()}"
            print(
                f"Creating a snapshot named {snapshot_id}. This typically takes a
few minutes."
            )
            snapshot = self.aurora_wrapper.create_cluster_snapshot(
                snapshot_id, cluster_name
            )
            while snapshot.get("Status") != "available":
                wait(30)
                snapshot = self.aurora_wrapper.get_cluster_snapshot(snapshot_id)
            pp(snapshot)
            print("-" * 88)

    def cleanup(self, db_inst, cluster, parameter_group):
        """
        Shows how to clean up a DB instance, DB cluster, and DB cluster parameter
group.
        Before the DB cluster parameter group can be deleted, all associated DB
instances and
        DB clusters must first be deleted.

        :param db_inst: The DB instance to delete.
        :param cluster: The DB cluster to delete.
        :param parameter_group: The DB cluster parameter group to delete.
        """
        cluster_name = cluster["DBClusterIdentifier"]
        parameter_group_name = parameter_group["DBClusterParameterGroupName"]
        if q.ask(
            "\nDo you want to delete the database instance, DB cluster, and
parameter "
            "group (y/n)? ",
            q.is_yesno,
        ):
```

```
            print(f"Deleting database instance
{db_inst['DBInstanceIdentifier']}.")

self.aurora_wrapper.delete_db_instance(db_inst["DBInstanceIdentifier"])
            print(f"Deleting database cluster {cluster_name}.")
            self.aurora_wrapper.delete_db_cluster(cluster_name)
            print(
                "Waiting for the DB instance and DB cluster to delete.\n"
                "This typically takes several minutes."
            )
            while db_inst is not None or cluster is not None:
                wait(30)
                if db_inst is not None:
                    db_inst = self.aurora_wrapper.get_db_instance(
                        db_inst["DBInstanceIdentifier"]
                    )
                if cluster is not None:
                    cluster = self.aurora_wrapper.get_db_cluster(
                        cluster["DBClusterIdentifier"]
                    )
            print(f"Deleting parameter group {parameter_group_name}.")
            self.aurora_wrapper.delete_parameter_group(parameter_group_name)

    def run_scenario(self, db_engine, parameter_group_name, cluster_name,
db_name):
        print("-" * 88)
        print(
            "Welcome to the Amazon Relational Database Service (Amazon RDS) get
started\n"
            "with Aurora DB clusters demo."
        )
        print("-" * 88)

        parameter_group = self.create_parameter_group(db_engine,
parameter_group_name)
        self.set_user_parameters(parameter_group_name)
        cluster = self.create_cluster(cluster_name, db_engine, db_name,
parameter_group)
        wait(5)
        db_inst = self.create_instance(cluster)
        self.display_connection(cluster)
        self.create_snapshot(cluster_name)
        self.cleanup(db_inst, cluster, parameter_group)
```

```
        print("\nThanks for watching!")
        print("-" * 88)


if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    try:
        scenario = AuroraClusterScenario(AuroraWrapper.from_client())
        scenario.run_scenario(
            "aurora-mysql",
            "doc-example-cluster-parameter-group",
            "doc-example-aurora",
            "docexampledb",
        )
    except Exception:
        logging.exception("Something went wrong with the demo.")
```

定義案例所呼叫的函數以管理 Amazon 動作。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameter_group(self, parameter_group_name):
        """
        Gets a DB cluster parameter group.
```

```
        :param parameter_group_name: The name of the parameter group to retrieve.
        :return: The requested parameter group.
        """
        try:
            response = self.rds_client.describe_db_cluster_parameter_groups(
                DBClusterParameterGroupName=parameter_group_name
            )
            parameter_group = response["DBClusterParameterGroups"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
                logger.info("Parameter group %s does not exist.",
parameter_group_name)
            else:
                logger.error(
                    "Couldn't get parameter group %s. Here's why: %s: %s",
                    parameter_group_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return parameter_group


    def create_parameter_group(
        self, parameter_group_name, parameter_group_family, description
    ):
        """
        Creates a DB cluster parameter group that is based on the specified
parameter group
        family.

        :param parameter_group_name: The name of the newly created parameter
group.
        :param parameter_group_family: The family that is used as the basis of
the new
                                       parameter group.
        :param description: A description given to the parameter group.
        :return: Data about the newly created parameter group.
        """
        try:
            response = self.rds_client.create_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
```

```
                Description=description,
            )
        except ClientError as err:
            logger.error(
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to delete.
        :return: Data about the parameter group.
        """
        try:
            response = self.rds_client.delete_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
        Gets the parameters that are contained in a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
filtered
```

```
                            to contain only parameters that start with this
prefix.
        :param source: When specified, only parameters from this source are
retrieved.
                       For example, a source of 'user' retrieves only parameters
that
                       were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBClusterParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator =
self.rds_client.get_paginator("describe_db_cluster_parameters")
            for page in paginator.paginate(**kwargs):
                parameters += [
                    p
                    for p in page["Parameters"]
                    if p["ParameterName"].startswith(name_prefix)
                ]
        except ClientError as err:
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return parameters


    def update_parameters(self, parameter_group_name, update_parameters):
        """
        Updates parameters in a custom DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to update.
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_cluster_parameter_group(
```

```
                DBClusterParameterGroupName=parameter_group_name,
                Parameters=update_parameters,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def get_db_cluster(self, cluster_name):
        """
        Gets data about an Aurora DB cluster.

        :param cluster_name: The name of the DB cluster to retrieve.
        :return: The retrieved DB cluster.
        """
        try:
            response = self.rds_client.describe_db_clusters(
                DBClusterIdentifier=cluster_name
            )
            cluster = response["DBClusters"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBClusterNotFoundFault":
                logger.info("Cluster %s does not exist.", cluster_name)
            else:
                logger.error(
                    "Couldn't verify the existence of DB cluster %s. Here's why:
%s: %s",
                    cluster_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return cluster


    def create_db_cluster(
```

```
        self,
        cluster_name,
        parameter_group_name,
        db_name,
        db_engine,
        db_engine_version,
        admin_name,
        admin_password,
    ):
        """
        Creates a DB cluster that is configured to use the specified parameter
group.
        The newly created DB cluster contains a database that uses the specified
engine and
        engine version.

        :param cluster_name: The name of the DB cluster to create.
        :param parameter_group_name: The name of the parameter group to associate
with
                                     the DB cluster.
        :param db_name: The name of the database to create.
        :param db_engine: The database engine of the database that is created,
such as MySql.
        :param db_engine_version: The version of the database engine.
        :param admin_name: The user name of the database administrator.
        :param admin_password: The password of the database administrator.
        :return: The newly created DB cluster.
        """
        try:
            response = self.rds_client.create_db_cluster(
                DatabaseName=db_name,
                DBClusterIdentifier=cluster_name,
                DBClusterParameterGroupName=parameter_group_name,
                Engine=db_engine,
                EngineVersion=db_engine_version,
                MasterUsername=admin_name,
                MasterUserPassword=admin_password,
            )
            cluster = response["DBCluster"]
        except ClientError as err:
            logger.error(
                "Couldn't create database %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
```

```
                err.response["Error"]["Message"],
            )
            raise
        else:
            return cluster


    def delete_db_cluster(self, cluster_name):
        """
        Deletes a DB cluster.

        :param cluster_name: The name of the DB cluster to delete.
        """
        try:
            self.rds_client.delete_db_cluster(
                DBClusterIdentifier=cluster_name, SkipFinalSnapshot=True
            )
            logger.info("Deleted DB cluster %s.", cluster_name)
        except ClientError:
            logger.exception("Couldn't delete DB cluster %s.", cluster_name)
            raise


    def create_cluster_snapshot(self, snapshot_id, cluster_id):
        """
        Creates a snapshot of a DB cluster.

        :param snapshot_id: The ID to give the created snapshot.
        :param cluster_id: The DB cluster to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_cluster_snapshot(
                DBClusterSnapshotIdentifier=snapshot_id,
DBClusterIdentifier=cluster_id
            )
            snapshot = response["DBClusterSnapshot"]
        except ClientError as err:
            logger.error(
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                cluster_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
```

```
            raise
        else:
            return snapshot


    def get_cluster_snapshot(self, snapshot_id):
        """
        Gets a DB cluster snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_cluster_snapshots(
                DBClusterSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBClusterSnapshots"][0]
        except ClientError as err:
            logger.error(
                "Couldn't get DB cluster snapshot %s. Here's why: %s: %s",
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot


    def create_instance_in_cluster(
        self, instance_id, cluster_id, db_engine, instance_class
    ):
        """
        Creates a database instance in an existing DB cluster. The first database
that is
        created defaults to a read-write DB instance.

        :param instance_id: The ID to give the newly created DB instance.
        :param cluster_id: The ID of the DB cluster where the DB instance is
created.
        :param db_engine: The database engine of a database to create in the DB
instance.
                          This must be compatible with the configured parameter
group
```

```
                          of the DB cluster.
        :param instance_class: The DB instance class for the newly created DB
instance.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBInstanceIdentifier=instance_id,
                DBClusterIdentifier=cluster_id,
                Engine=db_engine,
                DBInstanceClass=instance_class,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst


    def get_engine_versions(self, engine, parameter_group_family=None):
        """
        Gets database engine versions that are available for the specified engine
        and parameter group family.

        :param engine: The database engine to look up.
        :param parameter_group_family: When specified, restricts the returned
list of
                                       engine versions to those that are
compatible with
                                       this parameter group family.
        :return: The list of database engine versions.
        """
        try:
            kwargs = {"Engine": engine}
            if parameter_group_family is not None:
                kwargs["DBParameterGroupFamily"] = parameter_group_family
            response = self.rds_client.describe_db_engine_versions(**kwargs)
            versions = response["DBEngineVersions"]
```

```
        except ClientError as err:
            logger.error(
                "Couldn't get engine versions for %s. Here's why: %s: %s",
                engine,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return versions


    def get_orderable_instances(self, db_engine, db_engine_version):
        """
        Gets DB instance options that can be used to create DB instances that are
        compatible with a set of specifications.

        :param db_engine: The database engine that must be supported by the DB
instance.
        :param db_engine_version: The engine version that must be supported by
the DB instance.
        :return: The list of DB instance options that can be used to create a
compatible DB instance.
        """
        try:
            inst_opts = []
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return inst_opts
```

```
    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.

        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return db_inst


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
        :return: Data about the deleted DB instance.
        """
        try:
            response = self.rds_client.delete_db_instance(
                DBInstanceIdentifier=instance_id,
                SkipFinalSnapshot=True,
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
```

```
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- 如需 API 詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的下列主題。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

一種程式庫，其中包含 Aurora 案例的案例特定功能。

```
use phf::{phf_set, Set};
use secrecy::SecretString;
use std::{collections::HashMap, fmt::Display, time::Duration};

use aws_sdk_rds::{
    error::ProvideErrorMetadata,

 operation::create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
    types::{DbCluster, DbClusterParameterGroup, DbClusterSnapshot, DbInstance,
 Parameter},
};
use sdk_examples_test_utils::waiter::Waiter;
use tracing::{info, trace, warn};

const DB_ENGINE: &str = "aurora-mysql";
const DB_CLUSTER_PARAMETER_GROUP_NAME: &str =
 "RustSDKCodeExamplesDBParameterGroup";
const DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION: &str =
    "Parameter Group created by Rust SDK Code Example";
const DB_CLUSTER_IDENTIFIER: &str = "RustSDKCodeExamplesDBCluster";
const DB_INSTANCE_IDENTIFIER: &str = "RustSDKCodeExamplesDBInstance";

static FILTER_PARAMETER_NAMES: Set<&'static str> = phf_set! {
    "auto_increment_offset",
    "auto_increment_increment",
};

#[derive(Debug, PartialEq, Eq)]
struct MetadataError {
    message: Option<String>,
```

```
    code: Option<String>,
}

impl MetadataError {
    fn from(err: &dyn ProvideErrorMetadata) -> Self {
        MetadataError {
            message: err.message().map(String::from),
            code: err.code().map(String::from),
        }
    }
}

impl Display for MetadataError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        let display = match (&self.message, &self.code) {
            (None, None) => "Unknown".to_string(),
            (None, Some(code)) => format!("({code})"),
            (Some(message), None) => message.to_string(),
            (Some(message), Some(code)) => format!("{message} ({code})"),
        };
        write!(f, "{display}")
    }
}

#[derive(Debug, PartialEq, Eq)]
pub struct ScenarioError {
    message: String,
    context: Option<MetadataError>,
}

impl ScenarioError {
    pub fn with(message: impl Into<String>) -> Self {
        ScenarioError {
            message: message.into(),
            context: None,
        }
    }

    pub fn new(message: impl Into<String>, err: &dyn ProvideErrorMetadata) ->
 Self {
        ScenarioError {
            message: message.into(),
            context: Some(MetadataError::from(err)),
        }
```

```
    }
}

impl std::error::Error for ScenarioError {}
impl Display for ScenarioError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match &self.context {
            Some(c) => write!(f, "{}: {}", self.message, c),
            None => write!(f, "{}", self.message),
        }
    }
}

// Parse the ParameterName, Description, and AllowedValues values and display
 them.
#[derive(Debug)]
pub struct AuroraScenarioParameter {
    name: String,
    allowed_values: String,
    current_value: String,
}

impl Display for AuroraScenarioParameter {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(
            f,
            "{}: {} (allowed: {})",
            self.name, self.current_value, self.allowed_values
        )
    }
}

impl From<aws_sdk_rds::types::Parameter> for AuroraScenarioParameter {
    fn from(value: aws_sdk_rds::types::Parameter) -> Self {
        AuroraScenarioParameter {
            name: value.parameter_name.unwrap_or_default(),
            allowed_values: value.allowed_values.unwrap_or_default(),
            current_value: value.parameter_value.unwrap_or_default(),
        }
    }
}

pub struct AuroraScenario {
    rds: crate::rds::Rds,
```

```
    engine_family: Option<String>,
    engine_version: Option<String>,
    instance_class: Option<String>,
    db_cluster_parameter_group: Option<DbClusterParameterGroup>,
    db_cluster_identifier: Option<String>,
    db_instance_identifier: Option<String>,
    username: Option<String>,
    password: Option<SecretString>,
}

impl AuroraScenario {
    pub fn new(client: crate::rds::Rds) -> Self {
        AuroraScenario {
            rds: client,
            engine_family: None,
            engine_version: None,
            instance_class: None,
            db_cluster_parameter_group: None,
            db_cluster_identifier: None,
            db_instance_identifier: None,
            username: None,
            password: None,
        }
    }

    // Get available engine families for Aurora MySql.
 rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
 'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
    pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
 ScenarioError> {
        let describe_db_engine_versions =
 self.rds.describe_db_engine_versions(DB_ENGINE).await;
        trace!(versions=?describe_db_engine_versions, "full list of versions");

        if let Err(err) = describe_db_engine_versions {
            return Err(ScenarioError::new(
                "Failed to retrieve DB Engine Versions",
                &err,
            ));
        };

        let version_count = describe_db_engine_versions
            .as_ref()
            .map(|o| o.db_engine_versions().len())
```

```
            .unwrap_or_default();
        info!(version_count, "got list of versions");

        // Create a map of engine families to their available versions.
        let mut versions = HashMap::<String, Vec<String>>::new();
        describe_db_engine_versions
            .unwrap()
            .db_engine_versions()
            .iter()
            .filter_map(
                |v| match (&v.db_parameter_group_family, &v.engine_version) {
                    (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
                    _ => None,
                },
            )
            .for_each(|(family, version)|
versions.entry(family).or_default().push(version));

        Ok(versions)
    }

    pub async fn get_instance_classes(&self) -> Result<Vec<String>,
ScenarioError> {
        let describe_orderable_db_instance_options_items = self
            .rds
            .describe_orderable_db_instance_options(
                DB_ENGINE,
                self.engine_version
                    .as_ref()
                    .expect("engine version for db instance options")
                    .as_str(),
            )
            .await;

        describe_orderable_db_instance_options_items
            .map(|options| {
                options
                    .iter()
                    .filter(|o| o.storage_type() == Some("aurora"))
                    .map(|o|
o.db_instance_class().unwrap_or_default().to_string())
                    .collect::<Vec<String>>()
            })
```

```
            .map_err(|err| ScenarioError::new("Could not get available instance
classes", &err))
    }

    // Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
        self.engine_family = Some(engine.to_string());
        self.engine_version = Some(version.to_string());
        let create_db_cluster_parameter_group = self
            .rds
            .create_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
                engine,
            )
            .await;

        match create_db_cluster_parameter_group {
            Ok(CreateDbClusterParameterGroupOutput {
                db_cluster_parameter_group: None,
                ..
            }) => {
                return Err(ScenarioError::with(
                    "CreateDBClusterParameterGroup had empty response",
                ));
            }
            Err(error) => {
                if error.code() == Some("DBParameterGroupAlreadyExists") {
                    info!("Cluster Parameter Group already exists, nothing to
do");
                } else {
                    return Err(ScenarioError::new(
                        "Could not create Cluster Parameter Group",
                        &error,
                    ));
                }
            }
            _ => {
                info!("Created Cluster Parameter Group");
            }
        }
```

```
        Ok(())
    }

    pub fn set_instance_class(&mut self, instance_class: Option<String>) {
        self.instance_class = instance_class;
    }

    pub fn set_login(&mut self, username: Option<String>, password:
Option<SecretString>) {
        self.username = username;
        self.password = password;
    }

    pub async fn connection_string(&self) -> Result<String, ScenarioError> {
        let cluster = self.get_cluster().await?;
        let endpoint = cluster.endpoint().unwrap_or_default();
        let port = cluster.port().unwrap_or_default();
        let username = cluster.master_username().unwrap_or_default();
        Ok(format!("mysql -h {endpoint} -P {port} -u {username} -p"))
    }

    pub async fn get_cluster(&self) -> Result<DbCluster, ScenarioError> {
        let describe_db_clusters_output = self
            .rds
            .describe_db_clusters(
                self.db_cluster_identifier
                    .as_ref()
                    .expect("cluster identifier")
                    .as_str(),
            )
            .await;
        if let Err(err) = describe_db_clusters_output {
            return Err(ScenarioError::new("Failed to get cluster", &err));
        }

        let db_cluster = describe_db_clusters_output
            .unwrap()
            .db_clusters
            .and_then(|output| output.first().cloned());

        db_cluster.ok_or_else(|| ScenarioError::with("Did not find the cluster"))
    }

    // Get the parameter group. rds.DescribeDbClusterParameterGroups
```

```
   // Get parameters in the group. This is a long list so you will have to
paginate. Find the auto_increment_offset and auto_increment_increment parameters
(by ParameterName). rds.DescribeDbClusterParameters
   // Parse the ParameterName, Description, and AllowedValues values and display
them.
   pub async fn cluster_parameters(&self) ->
Result<Vec<AuroraScenarioParameter>, ScenarioError> {
       let parameters_output = self
           .rds
           .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
           .await;

       if let Err(err) = parameters_output {
           return Err(ScenarioError::new(
               format!("Failed to retrieve parameters for
{DB_CLUSTER_PARAMETER_GROUP_NAME}"),
               &err,
           ));
       }

       let parameters = parameters_output
           .unwrap()
           .into_iter()
           .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
           .filter(|p|
FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
           .map(AuroraScenarioParameter::from)
           .collect::<Vec<_>>();

       Ok(parameters)
   }

   // Modify both the auto_increment_offset and auto_increment_increment
parameters in one call in the custom parameter group. Set their ParameterValue
fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
   pub async fn update_auto_increment(
       &self,
       offset: u8,
       increment: u8,
   ) -> Result<(), ScenarioError> {
       let modify_db_cluster_parameter_group = self
           .rds
           .modify_db_cluster_parameter_group(
               DB_CLUSTER_PARAMETER_GROUP_NAME,
```

```
                vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value(format!("{offset}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value(format!("{increment}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ],
            )
            .await;

        if let Err(error) = modify_db_cluster_parameter_group {
            return Err(ScenarioError::new(
                "Failed to modify cluster parameter group",
                &error,
            ));
        }

        Ok(())
    }

    // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
    // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
    // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
    // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
    pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
        if self.password.is_none() {
            return Err(ScenarioError::with(
                "Must set Secret Password before starting a cluster",
```

```
            ));
        }
        let create_db_cluster = self
            .rds
            .create_db_cluster(
                DB_CLUSTER_IDENTIFIER,
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
```

```
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
```

```
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
```

```
   // Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until
Status == 'available'.
   pub async fn snapshot(&self, name: &str) -> Result<DbClusterSnapshot,
ScenarioError> {
       let id = self.db_cluster_identifier.as_deref().unwrap_or_default();
       let snapshot = self
           .rds
           .snapshot_cluster(id, format!("{id}_{name}").as_str())
           .await;
       match snapshot {
           Ok(output) => match output.db_cluster_snapshot {
               Some(snapshot) => Ok(snapshot),
               None => Err(ScenarioError::with("Missing Snapshot")),
           },
           Err(err) => Err(ScenarioError::new("Failed to create snapshot",
&err)),
       }
   }

   pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
       let mut clean_up_errors: Vec<ScenarioError> = vec![];

       // Delete the instance. rds.DeleteDbInstance.
       let delete_db_instance = self
           .rds
           .delete_db_instance(
               self.db_instance_identifier
                   .as_deref()
                   .expect("instance identifier"),
           )
           .await;
       if let Err(err) = delete_db_instance {
           let identifier = self
               .db_instance_identifier
               .as_deref()
               .unwrap_or("Missing Instance Identifier");
           let message = format!("failed to delete db instance {identifier}");
           clean_up_errors.push(ScenarioError::new(message, &err));
       } else {
           // Wait for the instance to delete
           let waiter = Waiter::default();
           while waiter.sleep().await.is_ok() {
               let describe_db_instances =
self.rds.describe_db_instances().await;
```

```
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;
```

```
        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
```

```
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
 rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
 DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }
}

#[cfg(test)]
pub mod tests;
```

在 RDS 用戶端包裝函式使用自動模擬對程式庫進行測試。

```
use crate::rds::MockRdsImpl;
```

```
use super::*;

use std::io::{Error, ErrorKind};

use assert_matches::assert_matches;
use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
 CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::DeleteDbClusterOutput,
        delete_db_cluster_parameter_group::DeleteDbClusterParameterGroupOutput,
        delete_db_instance::DeleteDbInstanceOutput,
        describe_db_cluster_endpoints::DescribeDbClusterEndpointsOutput,
        describe_db_cluster_parameters::{
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
        },
        describe_db_clusters::{DescribeDBClustersError,
 DescribeDbClustersOutput},
        describe_db_engine_versions::{
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
 DescribeDbInstancesOutput},

 describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError,
 ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{
        error::DbParameterGroupAlreadyExistsFault, DbClusterEndpoint,
 DbEngineVersion,
        OrderableDbInstanceOption,
    },
};
use aws_smithy_runtime_api::http::{Response, StatusCode};
use aws_smithy_types::body::SdkBody;
use mockall::predicate::eq;
```

```
use secrecy::ExposeSecret;

#[tokio::test]
async fn test_scenario_set_engine() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert_eq!(set_engine, Ok(()));
    assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
    assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
 Ok(CreateDbClusterParameterGroupOutput::builder().build()));
```

```
    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(

 CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                ),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_get_engines() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Ok(DescribeDbEngineVersionsOutput::builder()
                .db_engine_versions(
```

```
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1a")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1b")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f2")
                        .engine_version("f2a")
                        .build(),
                )
                .db_engine_versions(DbEngineVersion::builder().build())
                .build())
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;

    assert_eq!(
        versions_map,
        Ok(HashMap::from([
            ("f1".into(), vec!["f1a".into(), "f1b".into()]),
            ("f2".into(), vec!["f2a".into()])
        ]))
    );
}

#[tokio::test]
async fn test_scenario_get_engines_failed() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
```

```
                    ErrorKind::Other,
                    "describe_db_engine_versions error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;
    assert_matches!(
        versions_map,
        Err(ScenarioError { message, context: _ }) if message == "Failed to
 retrieve DB Engine Versions"
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Ok(vec![
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora-iopt1")
                    .build(),
```

```
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t2")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t3")
                    .storage_type("aurora")
                    .build(),
            ])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario
        .set_engine("aurora-mysql", "aurora-mysql8.0")
        .await
        .expect("set engine");

    let instance_classes = scenario.get_instance_classes().await;

    assert_eq!(
        instance_classes,
        Ok(vec!["t1".into(), "t2".into(), "t3".into()])
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(

 DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });
```

```
    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_family = Some("aurora-mysql".into());
    scenario.engine_version = Some("aurora-mysql8.0".into());

    let instance_classes = scenario.get_instance_classes().await;

    assert_matches!(
        instance_classes,
        Err(ScenarioError {message, context: _}) if message == "Could not get
 available instance classes"
    );
}

#[tokio::test]
async fn test_scenario_get_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert!(cluster.is_ok());
}

#[tokio::test]
async fn test_scenario_get_cluster_missing_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
```

```
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| Ok(DescribeDbClustersOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if
 message == "Did not find the cluster");
}

#[tokio::test]
async fn test_scenario_get_cluster_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_clusters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
```

```
    let cluster = scenario.get_cluster().await;

    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if
 message == "Failed to get cluster");
}

#[tokio::test]
async fn test_scenario_connection_string() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .endpoint("test_endpoint")
                        .port(3306)
                        .master_username("test_username")
                        .build(),
                )
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let connection_string = scenario.connection_string().await;

    assert_eq!(
        connection_string,
        Ok("mysql -h test_endpoint -P 3306 -u test_username -p".into())
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
```

```
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("d").build())
                .build()])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

    let params = scenario.cluster_parameters().await.expect("cluster params");
    let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
    assert_eq!(
        names,
        vec!["auto_increment_offset", "auto_increment_increment"]
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
```

```
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let params = scenario.cluster_parameters().await;
    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message
 == "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}

#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .withf(|name, params| {
            assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(
                params,
                &vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value("10")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value("20")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ]
            );
            true
        })
        .return_once(|_, _|
 Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

    let scenario = AuroraScenario::new(mock_rds);

    scenario
        .update_auto_increment(10, 20)
        .await
        .expect("update auto increment");
}
```

```
#[tokio::test]
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
            Err(SdkError::service_error(

 ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let update = scenario.update_auto_increment(10, 20).await;
    assert_matches!(update, Err(ScenarioError { message, context: _}) if message
 == "Failed to modify cluster parameter group");
}

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
```

```
.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
```

```
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
```

```
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));
```

```
    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
```

```
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
```

```
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()
```

```
 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
```

```
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );
```

```
    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
```

```
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
```

```
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _})
 if message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_snapshot() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
```

```
        .times(1)
        .return_once(|_, _| {
            Ok(CreateDbClusterSnapshotOutput::builder()
                .db_cluster_snapshot(
                    DbClusterSnapshot::builder()
                        .db_cluster_identifier("MockCluster")

 .db_cluster_snapshot_identifier("MockCluster_MockSnapshot")
                        .build(),
                )
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert!(create_snapshot.is_ok());
}

#[tokio::test]
async fn test_scenario_snapshot_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _| {
            Err(SdkError::service_error(
                CreateDBClusterSnapshotError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create snapshot error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
 message == "Failed to create snapshot");
}
```

```
#[tokio::test]
async fn test_scenario_snapshot_invalid() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _|
 Ok(CreateDbClusterSnapshotOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
 message == "Missing Snapshot");
}
```

一種二進位檔案，其會使用查詢器從前端執行案例，以便使用者可以做出某些決策。

```
use std::fmt::Display;

use anyhow::anyhow;
use aurora_code_examples::{
    aurora_scenario::{AuroraScenario, ScenarioError},
    rds::Rds as RdsClient,
};
use aws_sdk_rds::Client;
use inquire::{validator::StringValidator, CustomUserError};
use secrecy::SecretString;
use tracing::warn;

#[derive(Default, Debug)]
struct Warnings(Vec<String>);

impl Warnings {
    fn new() -> Self {
        Warnings(Vec::with_capacity(5))
    }
```

```
    fn push(&mut self, warning: &str, error: ScenarioError) {
        let formatted = format!("{warning}: {error}");
        warn!("{formatted}");
        self.0.push(formatted);
    }

    fn is_empty(&self) -> bool {
        self.0.is_empty()
    }
}

impl Display for Warnings {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        writeln!(f, "Warnings:")?;
        for warning in &self.0 {
            writeln!(f, "{: >4}- {warning}", "")?;
        }
        Ok(())
    }
}

fn select(
    prompt: &str,
    choices: Vec<String>,
    error_message: &str,
) -> Result<String, anyhow::Error> {
    inquire::Select::new(prompt, choices)
        .prompt()
        .map_err(|error| anyhow!("{error_message}: {error}"))
}

// Prepare the Aurora Scenario. Prompt for several settings that are optional to
 the Scenario, but that the user should choose for the demo.
// This includes the engine, engine version, and instance class.
async fn prepare_scenario(rds: RdsClient) -> Result<AuroraScenario,
 anyhow::Error> {
    let mut scenario = AuroraScenario::new(rds);

    // Get available engine families for Aurora MySql.
 rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
 'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
    let available_engines = scenario.get_engines().await;
    if let Err(error) = available_engines {
        return Err(anyhow!("Failed to get available engines: {}", error));
```

```
    }
    let available_engines = available_engines.unwrap();

    // Select an engine family and create a custom DB cluster parameter group.
 rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    let engine = select(
        "Select an Aurora engine family",
        available_engines.keys().cloned().collect::<Vec<String>>(),
        "Invalid engine selection",
    )?;

    let version = select(
        format!("Select an Aurora engine version for {engine}").as_str(),
        available_engines.get(&engine).cloned().unwrap_or_default(),
        "Invalid engine version selection",
    )?;

    let set_engine = scenario.set_engine(engine.as_str(),
 version.as_str()).await;
    if let Err(error) = set_engine {
        return Err(anyhow!("Could not set engine: {}", error));
    }

    let instance_classes = scenario.get_instance_classes().await;
    match instance_classes {
        Ok(classes) => {
            let instance_class = select(
                format!("Select an Aurora instance class for {engine}").as_str(),
                classes,
                "Invalid instance class selection",
            )?;
            scenario.set_instance_class(Some(instance_class))
        }
        Err(err) => return Err(anyhow!("Failed to get instance classes for
 engine: {err}")),
    }

    Ok(scenario)
}

// Prepare the cluster, creating a custom parameter group overriding some group
 parameters based on user input.
async fn prepare_cluster(scenario: &mut AuroraScenario, warnings: &mut Warnings)
 -> Result<(), ()> {
```

```
   show_parameters(scenario, warnings).await;

   let offset = prompt_number_or_default(warnings, "auto_increment_offset", 5);
   let increment = prompt_number_or_default(warnings,
"auto_increment_increment", 3);

   // Modify both the auto_increment_offset and auto_increment_increment
parameters in one call in the custom parameter group. Set their ParameterValue
fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
   let update_auto_increment = scenario.update_auto_increment(offset,
increment).await;

   if let Err(error) = update_auto_increment {
       warnings.push("Failed to update auto increment", error);
       return Err(());
   }

   // Get and display the updated parameters. Specify Source of 'user' to get
just the modified parameters. rds.DescribeDbClusterParameters(Source='user')
   show_parameters(scenario, warnings).await;

   let username = inquire::Text::new("Username for the database (default
'testuser')")
       .with_default("testuser")
       .with_initial_value("testuser")
       .prompt();

   if let Err(error) = username {
       warnings.push(
           "Failed to get username, using default",
           ScenarioError::with(format!("Error from inquirer: {error}")),
       );
       return Err(());
   }
   let username = username.unwrap();

   let password = inquire::Text::new("Password for the database (minimum 8
characters)")
       .with_validator(|i: &str| {
           if i.len() >= 8 {
               Ok(inquire::validator::Validation::Valid)
           } else {
               Ok(inquire::validator::Validation::Invalid(
                   "Password must be at least 8 characters".into(),
```

```
                ))
            }
        })
        .prompt();

    let password: Option<SecretString> = match password {
        Ok(password) => Some(SecretString::from(password)),
        Err(error) => {
            warnings.push(
                "Failed to get password, using none (and not starting a DB)",
                ScenarioError::with(format!("Error from inquirer: {error}")),
            );
            return Err(());
        }
    };

    scenario.set_login(Some(username), password);

    Ok(())
}

// Start a single instance in the cluster,
async fn run_instance(scenario: &mut AuroraScenario) -> Result<(), ScenarioError>
 {
    // Create an Aurora DB cluster database cluster that contains a MySql
 database and uses the parameter group you created.
    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
 for DBInstanceStatus == 'available'.
    scenario.start_cluster_and_instance().await?;

    let connection_string = scenario.connection_string().await?;

    println!("Database ready: {connection_string}",);

    let _ = inquire::Text::new("Use the database with the connection string. When
 you're finished, press enter key to continue.").prompt();

    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
    // Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until
 Status == 'available'.
    let snapshot_name = inquire::Text::new("Provide a name for the snapshot")
        .prompt()
        .unwrap_or(String::from("ScenarioRun"));
```

```
    let snapshot = scenario.snapshot(snapshot_name.as_str()).await?;
    println!(
        "Snapshot is available: {}",
        snapshot.db_cluster_snapshot_arn().unwrap_or("Missing ARN")
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), anyhow::Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);
    let rds = RdsClient::new(client);
    let mut scenario = prepare_scenario(rds).await?;

    // At this point, the scenario has things in AWS and needs to get cleaned up.
    let mut warnings = Warnings::new();

    if prepare_cluster(&mut scenario, &mut warnings).await.is_ok() {
        println!("Configured database cluster, starting an instance.");
        if let Err(err) = run_instance(&mut scenario).await {
            warnings.push("Problem running instance", err);
        }
    }

    // Clean up the instance, cluster, and parameter group, waiting for the
 instance and cluster to delete before moving on.
    let clean_up = scenario.clean_up().await;
    if let Err(errors) = clean_up {
        for error in errors {
            warnings.push("Problem cleaning up scenario", error);
        }
    }

    if warnings.is_empty() {
        Ok(())
    } else {
        println!("There were problems running the scenario:");
        println!("{warnings}");
        Err(anyhow!("There were problems running the scenario"))
    }
}
```

```
#[derive(Clone)]
struct U8Validator {}
impl StringValidator for U8Validator {
    fn validate(&self, input: &str) -> Result<inquire::validator::Validation,
 CustomUserError> {
        if input.parse::<u8>().is_err() {
            Ok(inquire::validator::Validation::Invalid(
                "Can't parse input as number".into(),
            ))
        } else {
            Ok(inquire::validator::Validation::Valid)
        }
    }
}

async fn show_parameters(scenario: &AuroraScenario, warnings: &mut Warnings) {
    let parameters = scenario.cluster_parameters().await;

    match parameters {
        Ok(parameters) => {
            println!("Current parameters");
            for parameter in parameters {
                println!("\t{parameter}");
            }
        }
        Err(error) => warnings.push("Could not find cluster parameters", error),
    }
}

fn prompt_number_or_default(warnings: &mut Warnings, name: &str, default: u8) ->
 u8 {
    let input = inquire::Text::new(format!("Updated {name}:").as_str())
        .with_validator(U8Validator {})
        .prompt();

    match input {
        Ok(increment) => match increment.parse::<u8>() {
            Ok(increment) => increment,
            Err(error) => {
                warnings.push(
                    format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                    ScenarioError::with(format!("{error}")),
```

```
                );
                default
            }
        },
        Err(error) => {
            warnings.push(
                format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                ScenarioError::with(format!("{error}")),
            );
            default
        }
    }
}
```

Amazon RDS 服務的包裝函式，其允許自動模擬以進行測試。

```
use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
 CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::{DeleteDBClusterError, DeleteDbClusterOutput},
        delete_db_cluster_parameter_group::{
            DeleteDBClusterParameterGroupError,
 DeleteDbClusterParameterGroupOutput,
        },
        delete_db_instance::{DeleteDBInstanceError, DeleteDbInstanceOutput},
        describe_db_cluster_endpoints::{
            DescribeDBClusterEndpointsError, DescribeDbClusterEndpointsOutput,
        },
        describe_db_cluster_parameters::{
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
        },
        describe_db_clusters::{DescribeDBClustersError,
 DescribeDbClustersOutput},
        describe_db_engine_versions::{
```

```
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
 DescribeDbInstancesOutput},

 describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError,
 ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{OrderableDbInstanceOption, Parameter},
    Client as RdsClient,
};
use secrecy::{ExposeSecret, SecretString};

#[cfg(test)]
use mockall::automock;

#[cfg(test)]
pub use MockRdsImpl as Rds;
#[cfg(not(test))]
pub use RdsImpl as Rds;

pub struct RdsImpl {
    pub inner: RdsClient,
}

#[cfg_attr(test, automock)]
impl RdsImpl {
    pub fn new(inner: RdsClient) -> Self {
        RdsImpl { inner }
    }

    pub async fn describe_db_engine_versions(
        &self,
        engine: &str,
    ) -> Result<DescribeDbEngineVersionsOutput,
 SdkError<DescribeDBEngineVersionsError>> {
        self.inner
            .describe_db_engine_versions()
            .engine(engine)
            .send()
            .await
```

```
    }

    pub async fn describe_orderable_db_instance_options(
        &self,
        engine: &str,
        engine_version: &str,
    ) -> Result<Vec<OrderableDbInstanceOption>,
SdkError<DescribeOrderableDBInstanceOptionsError>>
    {
        self.inner
            .describe_orderable_db_instance_options()
            .engine(engine)
            .engine_version(engine_version)
            .into_paginator()
            .items()
            .send()
            .try_collect()
            .await
    }

    pub async fn create_db_cluster_parameter_group(
        &self,
        name: &str,
        description: &str,
        family: &str,
    ) -> Result<CreateDbClusterParameterGroupOutput,
SdkError<CreateDBClusterParameterGroupError>>
    {
        self.inner
            .create_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .description(description)
            .db_parameter_group_family(family)
            .send()
            .await
    }

    pub async fn describe_db_clusters(
        &self,
        id: &str,
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
        self.inner
            .describe_db_clusters()
            .db_cluster_identifier(id)
```

```
            .send()
            .await
    }

    pub async fn describe_db_cluster_parameters(
        &self,
        name: &str,
    ) -> Result<Vec<DescribeDbClusterParametersOutput>,
SdkError<DescribeDBClusterParametersError>>
    {
        self.inner
            .describe_db_cluster_parameters()
            .db_cluster_parameter_group_name(name)
            .into_paginator()
            .send()
            .try_collect()
            .await
    }

    pub async fn modify_db_cluster_parameter_group(
        &self,
        name: &str,
        parameters: Vec<Parameter>,
    ) -> Result<ModifyDbClusterParameterGroupOutput,
SdkError<ModifyDBClusterParameterGroupError>>
    {
        self.inner
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .set_parameters(Some(parameters))
            .send()
            .await
    }

    pub async fn create_db_cluster(
        &self,
        name: &str,
        parameter_group: &str,
        engine: &str,
        version: &str,
        username: &str,
        password: SecretString,
    ) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
        self.inner
```

```
            .create_db_cluster()
            .db_cluster_identifier(name)
            .db_cluster_parameter_group_name(parameter_group)
            .engine(engine)
            .engine_version(version)
            .master_username(username)
            .master_user_password(password.expose_secret())
            .send()
            .await
    }

    pub async fn create_db_instance(
        &self,
        cluster_name: &str,
        instance_name: &str,
        instance_class: &str,
        engine: &str,
    ) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
        self.inner
            .create_db_instance()
            .db_cluster_identifier(cluster_name)
            .db_instance_identifier(instance_name)
            .db_instance_class(instance_class)
            .engine(engine)
            .send()
            .await
    }

    pub async fn describe_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner
            .describe_db_instances()
            .db_instance_identifier(instance_identifier)
            .send()
            .await
    }

    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
```

```
    ) -> Result<CreateDbClusterSnapshotOutput,
SdkError<CreateDBClusterSnapshotError>> {
        self.inner
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
            .send()
            .await
    }

    pub async fn describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }

    pub async fn describe_db_cluster_endpoints(
        &self,
        cluster_identifier: &str,
    ) -> Result<DescribeDbClusterEndpointsOutput,
SdkError<DescribeDBClusterEndpointsError>> {
        self.inner
            .describe_db_cluster_endpoints()
            .db_cluster_identifier(cluster_identifier)
            .send()
            .await
    }

    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

    pub async fn delete_db_cluster(
        &self,
        cluster_identifier: &str,
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
```

```
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

    pub async fn delete_db_cluster_parameter_group(
        &self,
        name: &str,
    ) -> Result<DeleteDbClusterParameterGroupOutput,
 SdkError<DeleteDBClusterParameterGroupError>>
    {
        self.inner
            .delete_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .send()
            .await
    }
}
```

在此案例中使用具有相依項的 Cargo.toml。

```
[package]
name = "aurora-code-examples"
authors = [
  "David Souther <dpsouth@amazon.com>",
]
edition = "2021"
version = "0.1.0"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/
reference/manifest.html

[dependencies]
anyhow = "1.0.75"
assert_matches = "1.5.0"
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-smithy-types = { version = "1.0.1" }
aws-smithy-runtime-api = { version = "1.0.1" }
aws-sdk-rds = { version = "1.3.0" }
```

```
inquire = "0.6.2"
mockall = "0.11.4"
phf = { version = "0.11.2", features = ["std", "macros"] }
sdk-examples-test-utils = { path = "../../test-utils" }
secrecy = "0.8.0"
tokio = { version = "1.20.1", features = ["full", "test-util"] }
tracing = "0.1.37"
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Rust API reference 中的下列主題。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

Aurora 使用 AWS SDKs的動作

下列程式碼範例示範如何使用 AWS SDKs 執行個別 Aurora 動作。每個範例均包含 GitHub 的連結，您可以在連結中找到設定和執行程式碼的相關說明。

這些摘錄會呼叫 Aurora API，是必須在內容中執行之大型程式的程式碼摘錄。您可以在 Aurora AWS SDKs的案例 中查看內容中的動作。

下列範例僅包含最常使用的動作。如需完整清單，請參閱 Amazon Aurora API 參考。

範例

- CreateDBCluster 搭配 AWS SDK 使用
- CreateDBClusterParameterGroup 搭配 AWS SDK 使用
- CreateDBClusterSnapshot 搭配 AWS SDK 使用
- CreateDBInstance 搭配 AWS SDK 使用
- DeleteDBCluster 搭配 AWS SDK 使用
- DeleteDBClusterParameterGroup 搭配 AWS SDK 使用
- DeleteDBInstance 搭配 AWS SDK 使用
- DescribeDBClusterParameterGroups 搭配 AWS SDK 使用
- DescribeDBClusterParameters 搭配 AWS SDK 使用
- DescribeDBClusterSnapshots 搭配 AWS SDK 使用
- DescribeDBClusters 搭配 AWS SDK 使用
- DescribeDBEngineVersions 搭配 AWS SDK 使用
- DescribeDBInstances 搭配 AWS SDK 使用
- DescribeOrderableDBInstanceOptions 搭配 AWS SDK 或 CLI 使用
- ModifyDBClusterParameterGroup 搭配 AWS SDK 使用

`CreateDBCluster` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `CreateDBCluster`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Create a new cluster and database.
    /// </summary>
    /// <param name="dbName">The name of the new database.</param>
    /// <param name="clusterIdentifier">The identifier of the cluster.</param>
    /// <param name="parameterGroupName">The name of the parameter group.</param>
    /// <param name="dbEngine">The engine to use for the new cluster.</param>
    /// <param name="dbEngineVersion">The version of the engine to use.</param>
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
```

```
        return response.DBCluster;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateDBCluster。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region (overrides config file).
    // clientConfig.region = "us-east-1";

Aws::RDS::RDSClient client(clientConfig);

    Aws::RDS::Model::CreateDBClusterRequest request;
    request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
    request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
    request.SetEngine(engineName);
    request.SetEngineVersion(engineVersionName);
    request.SetMasterUsername(administratorName);
    request.SetMasterUserPassword(administratorPassword);

    Aws::RDS::Model::CreateDBClusterOutcome outcome =
            client.CreateDBCluster(request);

    if (outcome.IsSuccess()) {
        std::cout << "The DB cluster creation has started."
                  << std::endl;
    }
    else {
        std::cerr << "Error with Aurora::CreateDBCluster. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
```

```
            return false;
        }
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 CreateDBCluster。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
// The newly created DB cluster contains a database that uses the specified
 engine and
// engine version.
func (clusters *DbClusters) CreateDbCluster(ctx context.Context, clusterName
 string, parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
```

```
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(ctx,
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 CreateDBCluster。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   public static String createDBCluster(RdsClient rdsClient, String
dbParameterGroupFamily, String dbName,
           String dbClusterIdentifier, String userName, String password) {
       try {
           CreateDbClusterRequest clusterRequest =
CreateDbClusterRequest.builder()
                   .databaseName(dbName)
                   .dbClusterIdentifier(dbClusterIdentifier)
```

```
                    .dbClusterParameterGroupName(dbParameterGroupFamily)
                    .engine("aurora-mysql")
                    .masterUsername(userName)
                    .masterUserPassword(password)
                    .build();

            CreateDbClusterResponse response =
 rdsClient.createDBCluster(clusterRequest);
            return response.dbCluster().dbClusterArn();

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
        return "";
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 CreateDBCluster。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun createDBCluster(
    dbParameterGroupFamilyVal: String?,
    dbName: String?,
    dbClusterIdentifierVal: String?,
    userName: String?,
    password: String?,
): String? {
    val clusterRequest =
        CreateDbClusterRequest {
            databaseName = dbName
            dbClusterIdentifier = dbClusterIdentifierVal
            dbClusterParameterGroupName = dbParameterGroupFamilyVal
```

```
            engine = "aurora-mysql"
            masterUsername = userName
            masterUserPassword = password
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbCluster(clusterRequest)
        return response.dbCluster?.dbClusterArn
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 CreateDBCluster。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)
```

```
   def create_db_cluster(
       self,
       cluster_name,
       parameter_group_name,
       db_name,
       db_engine,
       db_engine_version,
       admin_name,
       admin_password,
   ):
       """
       Creates a DB cluster that is configured to use the specified parameter
group.
       The newly created DB cluster contains a database that uses the specified
engine and
       engine version.

       :param cluster_name: The name of the DB cluster to create.
       :param parameter_group_name: The name of the parameter group to associate
with
                                    the DB cluster.
       :param db_name: The name of the database to create.
       :param db_engine: The database engine of the database that is created,
such as MySql.
       :param db_engine_version: The version of the database engine.
       :param admin_name: The user name of the database administrator.
       :param admin_password: The password of the database administrator.
       :return: The newly created DB cluster.
       """
       try:
           response = self.rds_client.create_db_cluster(
               DatabaseName=db_name,
               DBClusterIdentifier=cluster_name,
               DBClusterParameterGroupName=parameter_group_name,
               Engine=db_engine,
               EngineVersion=db_engine_version,
               MasterUsername=admin_name,
               MasterUserPassword=admin_password,
           )
           cluster = response["DBCluster"]
       except ClientError as err:
           logger.error(
               "Couldn't create database %s. Here's why: %s: %s",
```

```
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return cluster
```

- 如需 API 詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 CreateDBCluster。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
        if self.password.is_none() {
            return Err(ScenarioError::with(
```

```
                "Must set Secret Password before starting a cluster",
            ));
        }
        let create_db_cluster = self
            .rds
            .create_db_cluster(
                DB_CLUSTER_IDENTIFIER,
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
```

```
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
```

```
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }
```

```
    pub async fn create_db_cluster(
        &self,
        name: &str,
        parameter_group: &str,
        engine: &str,
        version: &str,
        username: &str,
        password: SecretString,
    ) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
        self.inner
            .create_db_cluster()
            .db_cluster_identifier(name)
            .db_cluster_parameter_group_name(parameter_group)
            .engine(engine)
            .engine_version(version)
            .master_username(username)
            .master_user_password(password.expose_secret())
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
```

```
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });
```

```
    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
```

```
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
```

```
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}
```

```
#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });
```

```
    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 CreateDBCluster。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`CreateDBClusterParameterGroup` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `CreateDBClusterParameterGroup`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</
param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await
 _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateDBClusterParameterGroup。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetDBParameterGroupFamily(dbParameterGroupFamily);
        request.SetDescription("Example cluster parameter group.");

        Aws::RDS::Model::CreateDBClusterParameterGroupOutcome outcome =
                client.CreateDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
created."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 CreateDBClusterParameterGroup。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBClusterParameterGroup, error) {

 output, err := clusters.AuroraClient.CreateDBClusterParameterGroup(ctx,
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 CreateDBClusterParameterGroup。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   public static void createDBClusterParameterGroup(RdsClient rdsClient, String
dbClusterGroupName,
           String dbParameterGroupFamily) {
       try {
           CreateDbClusterParameterGroupRequest groupRequest =
CreateDbClusterParameterGroupRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .dbParameterGroupFamily(dbParameterGroupFamily)
                   .description("Created by using the AWS SDK for Java")
                   .build();

           CreateDbClusterParameterGroupResponse response =
rdsClient.createDBClusterParameterGroup(groupRequest);
           System.out.println("The group name is " +
response.dbClusterParameterGroup().dbClusterParameterGroupName());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 CreateDBClusterParameterGroup。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun createDBClusterParameterGroup(
    dbClusterGroupNameVal: String?,
    dbParameterGroupFamilyVal: String?,
) {
    val groupRequest =
        CreateDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dbClusterGroupNameVal
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            description = "Created by using the AWS SDK for Kotlin"
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterParameterGroup(groupRequest)
        println("The group name is
 ${response.dbClusterParameterGroup?.dbClusterParameterGroupName}")
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 CreateDBClusterParameterGroup。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_parameter_group(
        self, parameter_group_name, parameter_group_family, description
    ):
        """
        Creates a DB cluster parameter group that is based on the specified
 parameter group
        family.

        :param parameter_group_name: The name of the newly created parameter
 group.
        :param parameter_group_family: The family that is used as the basis of
 the new
                                       parameter group.
        :param description: A description given to the parameter group.
        :return: Data about the newly created parameter group.
        """
        try:
            response = self.rds_client.create_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
                Description=description,
            )
        except ClientError as err:
            logger.error(
```

```
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- 如需 API 詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 CreateDBClusterParameterGroup。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   // Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
   pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
       self.engine_family = Some(engine.to_string());
       self.engine_version = Some(version.to_string());
       let create_db_cluster_parameter_group = self
           .rds
           .create_db_cluster_parameter_group(
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
               engine,
           )
           .await;

       match create_db_cluster_parameter_group {
           Ok(CreateDbClusterParameterGroupOutput {
```

```
                db_cluster_parameter_group: None,
                ..
            }) => {
                return Err(ScenarioError::with(
                    "CreateDBClusterParameterGroup had empty response",
                ));
            }
            Err(error) => {
                if error.code() == Some("DBParameterGroupAlreadyExists") {
                    info!("Cluster Parameter Group already exists, nothing to
 do");
                } else {
                    return Err(ScenarioError::new(
                        "Could not create Cluster Parameter Group",
                        &error,
                    ));
                }
            }
            _ => {
                info!("Created Cluster Parameter Group");
            }
        }

        Ok(())
    }

    pub async fn create_db_cluster_parameter_group(
        &self,
        name: &str,
        description: &str,
        family: &str,
    ) -> Result<CreateDbClusterParameterGroupOutput,
 SdkError<CreateDBClusterParameterGroupError>>
    {
        self.inner
            .create_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .description(description)
            .db_parameter_group_family(family)
            .send()
            .await
    }

#[tokio::test]
```

```
async fn test_scenario_set_engine() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert_eq!(set_engine, Ok(()));
    assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
    assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
 Ok(CreateDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(

 CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                ),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 CreateDBClusterParameterGroup。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
TRY.
    DATA(lo_output) = lo_rds->createdbclusterparamgroup(
      iv_dbclusterparamgroupname = iv_param_group_name
      iv_dbparametergroupfamily = iv_param_group_family
      iv_description = iv_description
    ).
    oo_result = lo_output->get_dbclusterparametergroup( ).
  CATCH /aws1/cx_rdsdbparmgralrexfault.
    " Re-raise exception - parameter group already exists
    RAISE EXCEPTION TYPE /aws1/cx_rdsdbparmgralrexfault.
  CATCH /aws1/cx_rdsdbprmgrquotaexcd00.
    " Re-raise exception - quota exceeded
    RAISE EXCEPTION TYPE /aws1/cx_rdsdbprmgrquotaexcd00.
ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 CreateDBClusterParameterGroup。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`CreateDBClusterSnapshot` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `CreateDBClusterSnapshot`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateDBClusterSnapshot。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::CreateDBClusterSnapshotRequest request;
            request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::CreateDBClusterSnapshotOutcome outcome =
                    client.CreateDBClusterSnapshot(request);

            if (outcome.IsSuccess()) {
                std::cout << "Snapshot creation has started."
                          << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::CreateDBClusterSnapshot. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 CreateDBClusterSnapshot。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(ctx context.Context,
 clusterName string, snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(ctx,
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBClusterSnapshot, nil
 }
```

```
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 CreateDBClusterSnapshot。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   public static void createDBClusterSnapshot(RdsClient rdsClient, String
dbInstanceClusterIdentifier,
           String dbSnapshotIdentifier) {
       try {
           CreateDbClusterSnapshotRequest snapshotRequest =
CreateDbClusterSnapshotRequest.builder()
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                   .build();

           CreateDbClusterSnapshotResponse response =
rdsClient.createDBClusterSnapshot(snapshotRequest);
           System.out.println("The Snapshot ARN is " +
response.dbClusterSnapshot().dbClusterSnapshotArn());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 CreateDBClusterSnapshot。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun createDBClusterSnapshot(
    dbInstanceClusterIdentifier: String?,
    dbSnapshotIdentifier: String?,
) {
    val snapshotRequest =
        CreateDbClusterSnapshotRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterSnapshot(snapshotRequest)
        println("The Snapshot ARN is
 ${response.dbClusterSnapshot?.dbClusterSnapshotArn}")
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 CreateDBClusterSnapshot。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_cluster_snapshot(self, snapshot_id, cluster_id):
        """
        Creates a snapshot of a DB cluster.

        :param snapshot_id: The ID to give the created snapshot.
        :param cluster_id: The DB cluster to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_cluster_snapshot(
                DBClusterSnapshotIdentifier=snapshot_id,
 DBClusterIdentifier=cluster_id
            )
            snapshot = response["DBClusterSnapshot"]
        except ClientError as err:
            logger.error(
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                cluster_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- 如需 API 詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 CreateDBClusterSnapshot。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
```

```
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
```

```
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }
```

```
            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
    ) -> Result<CreateDbClusterSnapshotOutput,
SdkError<CreateDBClusterSnapshotError>> {
        self.inner
```

```
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
```

```
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
```

```
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));
```

```
    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
```

```
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
```

```
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
```

```
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
```

```
    let _ = assertions.await;
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 CreateDBClusterSnapshot。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`CreateDBInstance` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `CreateDBInstance`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action
DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
```

```
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name
or size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateDBInstance。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBInstanceRequest request;
        request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetEngine(engineName);
        request.SetDBInstanceClass(dbInstanceClass);

        Aws::RDS::Model::CreateDBInstanceOutcome outcome =
                client.CreateDBInstance(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB instance creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::CreateDBInstance. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
"",
                             client);
            return false;
        }
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 CreateDBInstance。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import (
```

```
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(ctx context.Context,
 clusterName string, instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 CreateDBInstance。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   public static String createDBInstanceCluster(RdsClient rdsClient,
           String dbInstanceIdentifier,
           String dbInstanceClusterIdentifier,
           String instanceClass) {
       try {
           CreateDbInstanceRequest instanceRequest =
CreateDbInstanceRequest.builder()
                   .dbInstanceIdentifier(dbInstanceIdentifier)
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .engine("aurora-mysql")
                   .dbInstanceClass(instanceClass)
                   .build();

           CreateDbInstanceResponse response =
rdsClient.createDBInstance(instanceRequest);
           System.out.print("The status is " +
response.dbInstance().dbInstanceStatus());
           return response.dbInstance().dbInstanceArn();

       } catch (RdsException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
       return "";
   }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 CreateDBInstance。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun createDBInstanceCluster(
    dbInstanceIdentifierVal: String?,
    dbInstanceClusterIdentifierVal: String?,
    instanceClassVal: String?,
): String? {
    val instanceRequest =
        CreateDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            dbClusterIdentifier = dbInstanceClusterIdentifierVal
            engine = "aurora-mysql"
            dbInstanceClass = instanceClassVal
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
        return response.dbInstance?.dbInstanceArn
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 CreateDBInstance。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_instance_in_cluster(
        self, instance_id, cluster_id, db_engine, instance_class
    ):
        """
        Creates a database instance in an existing DB cluster. The first database
 that is
        created defaults to a read-write DB instance.

        :param instance_id: The ID to give the newly created DB instance.
        :param cluster_id: The ID of the DB cluster where the DB instance is
 created.
        :param db_engine: The database engine of a database to create in the DB
 instance.
```

```
                          This must be compatible with the configured parameter
group
                          of the DB cluster.
        :param instance_class: The DB instance class for the newly created DB
instance.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBInstanceIdentifier=instance_id,
                DBClusterIdentifier=cluster_id,
                Engine=db_engine,
                DBInstanceClass=instance_class,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- 如需 API 詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 CreateDBInstance。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
           )
           .await;
       if let Err(err) = create_db_cluster {
           return Err(ScenarioError::new(
               "Failed to create DB Cluster with cluster group",
               &err,
           ));
       }

       self.db_cluster_identifier = create_db_cluster
           .unwrap()
```

```
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
```

```
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
```

```
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn create_db_instance(
        &self,
        cluster_name: &str,
        instance_name: &str,
        instance_class: &str,
        engine: &str,
    ) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
        self.inner
            .create_db_instance()
            .db_cluster_identifier(cluster_name)
            .db_instance_identifier(instance_name)
            .db_instance_class(instance_class)
            .engine(engine)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
```

```
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });
```

```
    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
```

```
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
```

```
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
```

```
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
```

```
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 CreateDBInstance。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DeleteDBCluster` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `DeleteDBCluster`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DeleteDBCluster。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DeleteDBClusterRequest request;
            request.SetDBClusterIdentifier(dbClusterIdentifier);
            request.SetSkipFinalSnapshot(true);

            Aws::RDS::Model::DeleteDBClusterOutcome outcome =
                    client.DeleteDBCluster(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB cluster deletion has started."
                          << std::endl;
                clusterDeleting = true;
                std::cout
                        << "Waiting for DB cluster to delete before deleting the
parameter group."
                        << std::endl;
                std::cout << "This may take a while." << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBCluster. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DeleteDBCluster。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(ctx context.Context, clusterName
 string) error {
 _, err := clusters.AuroraClient.DeleteDBCluster(ctx, &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 DeleteDBCluster。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   public static void deleteCluster(RdsClient rdsClient, String
dbInstanceClusterIdentifier) {
       try {
           DeleteDbClusterRequest deleteDbClusterRequest =
DeleteDbClusterRequest.builder()
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .skipFinalSnapshot(true)
                   .build();

           rdsClient.deleteDBCluster(deleteDbClusterRequest);
           System.out.println(dbInstanceClusterIdentifier + " was deleted!");

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DeleteDBCluster。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun deleteCluster(dbInstanceClusterIdentifier: String) {
    val deleteDbClusterRequest =
        DeleteDbClusterRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            skipFinalSnapshot = true
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        rdsClient.deleteDbCluster(deleteDbClusterRequest)
        println("$dbInstanceClusterIdentifier was deleted!")
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 DeleteDBCluster。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
```

```
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_db_cluster(self, cluster_name):
        """
        Deletes a DB cluster.

        :param cluster_name: The name of the DB cluster to delete.
        """
        try:
            self.rds_client.delete_db_cluster(
                DBClusterIdentifier=cluster_name, SkipFinalSnapshot=True
            )
            logger.info("Deleted DB cluster %s.", cluster_name)
        except ClientError:
            logger.exception("Couldn't delete DB cluster %s.", cluster_name)
            raise
```

- 如需 API 詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 DeleteDBCluster。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
```

```
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
```

```
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
```

```
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
 DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_cluster(
        &self,
        cluster_identifier: &str,
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
```

```
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));
```

```
    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
```

```
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
```

```
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _})
if message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Instances
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 DeleteDBCluster。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DeleteDBClusterParameterGroup` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `DeleteDBClusterParameterGroup`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
```

```
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string
groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await
_amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DeleteDBClusterParameterGroup。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::DeleteDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(parameterGroupName);

        Aws::RDS::Model::DeleteDBClusterParameterGroupOutcome outcome =
                client.DeleteDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB parameter group was successfully deleted."
```

```
                    << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::DeleteDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DeleteDBClusterParameterGroup。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteParameterGroup deletes the named DB cluster parameter group.
```

```
func (clusters *DbClusters) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(ctx,
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 DeleteDBClusterParameterGroup。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   public static void deleteDBClusterGroup(RdsClient rdsClient, String
dbClusterGroupName, String clusterDBARN)
           throws InterruptedException {
       try {
           boolean isDataDel = false;
           boolean didFind;
           String instanceARN;

           // Make sure that the database has been deleted.
           while (!isDataDel) {
               DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
```

```
                List<DBInstance> instanceList = response.dbInstances();
                int listSize = instanceList.size();
                didFind = false;
                int index = 1;
                for (DBInstance instance : instanceList) {
                    instanceARN = instance.dbInstanceArn();
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        System.out.println(clusterDBARN + " still exists");
                        didFind = true;
                    }
                    if ((index == listSize) && (!didFind)) {
                        // Went through the entire list and did not find the
database ARN.
                        isDataDel = true;
                    }
                    Thread.sleep(sleepTime * 1000);
                    index++;
                }
            }

            DeleteDbClusterParameterGroupRequest clusterParameterGroupRequest =
DeleteDbClusterParameterGroupRequest
                    .builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .build();


rdsClient.deleteDBClusterParameterGroup(clusterParameterGroupRequest);
            System.out.println(dbClusterGroupName + " was deleted.");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DeleteDBClusterParameterGroup。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
@Throws(InterruptedException::class)
suspend fun deleteDBClusterGroup(
    dbClusterGroupName: String,
    clusterDBARN: String,
) {
    var isDataDel = false
    var didFind: Boolean
    var instanceARN: String

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        // Make sure that the database has been deleted.
        while (!isDataDel) {
            val response = rdsClient.describeDbInstances()
            val instanceList = response.dbInstances
            val listSize = instanceList?.size
            isDataDel = false
            didFind = false
            var index = 1
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceARN = instance.dbInstanceArn.toString()
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        println("$clusterDBARN still exists")
                        didFind = true
                    }
                    if (index == listSize && !didFind) {
                        // Went through the entire list and did not find the
 database ARN.
                        isDataDel = true
                    }
                    delay(slTime * 1000)
                    index++
```

```
                }
            }
        }
        val clusterParameterGroupRequest =
            DeleteDbClusterParameterGroupRequest {
                dbClusterParameterGroupName = dbClusterGroupName
            }

        rdsClient.deleteDbClusterParameterGroup(clusterParameterGroupRequest)
        println("$dbClusterGroupName was deleted.")
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 DeleteDBClusterParameterGroup。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
```

```
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to delete.
        :return: Data about the parameter group.
        """
        try:
            response = self.rds_client.delete_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- 如需 API 詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 DeleteDBClusterParameterGroup。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
```

```
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
```

```
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
```

```
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_cluster_parameter_group(
        &self,
        name: &str,
    ) -> Result<DeleteDbClusterParameterGroupOutput,
 SdkError<DeleteDBClusterParameterGroupError>>
    {
        self.inner
            .delete_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
```

```
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
```

```
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
```

```
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _})
 if message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 DeleteDBClusterParameterGroup。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
TRY.
    lo_rds->deletedbclusterparamgroup(
      iv_dbclusterparamgroupname = iv_param_group_name
    ).
  CATCH /aws1/cx_rdsdbprmgrnotfndfault.
    " Re-raise exception - parameter group not found
    RAISE EXCEPTION TYPE /aws1/cx_rdsdbprmgrnotfndfault.
  CATCH /aws1/cx_rdsinvdbprmgrstatef00.
    " Re-raise exception - invalid state
    RAISE EXCEPTION TYPE /aws1/cx_rdsinvdbprmgrstatef00.
ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 DeleteDBClusterParameterGroup。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DeleteDBInstance` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `DeleteDBInstance`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DeleteDBInstance。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DeleteDBInstanceRequest request;
            request.SetDBInstanceIdentifier(dbInstanceIdentifier);
            request.SetSkipFinalSnapshot(true);
            request.SetDeleteAutomatedBackups(true);

            Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                    client.DeleteDBInstance(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB instance deletion has started."
                          << std::endl;
                instanceDeleting = true;
                std::cout
                        << "Waiting for DB instance to delete before deleting the
parameter group."
                        << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBInstance. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DeleteDBInstance。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(ctx context.Context, instanceName
 string) error {
 _, err := clusters.AuroraClient.DeleteDBInstance(ctx,
 &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
```

```
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 DeleteDBInstance。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   public static void deleteDatabaseInstance(RdsClient rdsClient, String
dbInstanceIdentifier) {
       try {
           DeleteDbInstanceRequest deleteDbInstanceRequest =
DeleteDbInstanceRequest.builder()
                   .dbInstanceIdentifier(dbInstanceIdentifier)
                   .deleteAutomatedBackups(true)
                   .skipFinalSnapshot(true)
                   .build();

           DeleteDbInstanceResponse response =
rdsClient.deleteDBInstance(deleteDbInstanceRequest);
           System.out.println("The status of the database is " +
response.dbInstance().dbInstanceStatus());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DeleteDBInstance。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun deleteDBInstance(dbInstanceIdentifierVal: String) {
    val deleteDbInstanceRequest =
        DeleteDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            deleteAutomatedBackups = true
            skipFinalSnapshot = true
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 DeleteDBInstance。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class AuroraWrapper:
```

```
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
        :return: Data about the deleted DB instance.
        """
        try:
            response = self.rds_client.delete_db_instance(
                DBInstanceIdentifier=instance_id,
                SkipFinalSnapshot=True,
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- 如需 API 詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 DeleteDBInstance。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
```

```
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;
```

```
        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
```

```
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }
```

```
#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
```

```
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
```

```
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _})
if message == "Failed to check instance state during deletion");
```

```
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 DeleteDBInstance。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DescribeDBClusterParameterGroups` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `DescribeDBClusterParameterGroups`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</
param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
 DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeDBClusterParameterGroups。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsOutcome outcome =
```

```
                client.DescribeDBClusterParameterGroups(request);

        if (outcome.IsSuccess()) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' already exists." <<
 std::endl;
            dbParameterGroupFamily =
 outcome.GetResult().GetDBClusterParameterGroups()
[0].GetDBParameterGroupFamily();
        }

        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DescribeDBClusterParameterGroups。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
```

```
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  ctx, &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBClusterParameterGroups[0], err
 }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 DescribeDBClusterParameterGroups。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
       try {
           DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .maxRecords(20)
                   .build();

           List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                   .dbClusterParameterGroups();
           for (DBClusterParameterGroup group : groups) {
               System.out.println("The group name is " +
group.dbClusterParameterGroupName());
               System.out.println("The group ARN is " +
group.dbClusterParameterGroupArn());
           }

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DescribeDBClusterParameterGroups。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun describeDbClusterParameterGroups(dbClusterGroupName: String?) {
    val groupsRequest =
        DescribeDbClusterParameterGroupsRequest {
            dbClusterParameterGroupName = dbClusterGroupName
            maxRecords = 20
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbClusterParameterGroups(groupsRequest)
        response.dbClusterParameterGroups?.forEach { group ->
            println("The group name is ${group.dbClusterParameterGroupName}")
            println("The group ARN is ${group.dbClusterParameterGroupArn}")
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 DescribeDBClusterParameterGroups。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameter_group(self, parameter_group_name):
        """
        Gets a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to retrieve.
        :return: The requested parameter group.
        """
        try:
            response = self.rds_client.describe_db_cluster_parameter_groups(
                DBClusterParameterGroupName=parameter_group_name
            )
            parameter_group = response["DBClusterParameterGroups"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
                logger.info("Parameter group %s does not exist.",
 parameter_group_name)
            else:
                logger.error(
                    "Couldn't get parameter group %s. Here's why: %s: %s",
                    parameter_group_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
```

```
        else:
            return parameter_group
```

- 如需 API 詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 DescribeDBClusterParameterGroups。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
TRY.
    DATA(lo_output) = lo_rds->describedbclusterparamgroups(
      iv_dbclusterparamgroupname = iv_param_group_name
    ).
    DATA(lt_param_groups) = lo_output->get_dbclusterparametergroups( ).
    IF lines( lt_param_groups ) > 0.
      oo_result = lt_param_groups[ 1 ].
    ENDIF.
  CATCH /aws1/cx_rdsdbprmgrnotfndfault.
ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 DescribeDBClusterParameterGroups。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DescribeDBClusterParameters` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `DescribeDBClusterParameters`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await
_amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
```

```
        }
        while (response.Marker is not null);

        return paramList;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeDBClusterParameters。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets DB parameters using the 'DescribeDBClusterParameters' api.
/*!
 \sa getDBCLusterParameters()
 \param parameterGroupName: The name of the cluster parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBCLusterParameters(const Aws::String
 &parameterGroupName,
                                            const Aws::String &namePrefix,
                                            const Aws::String &source,
```

```
 Aws::Vector<Aws::RDS::Model::Parameter> &parametersResult,
                                            const Aws::RDS::RDSClient &client) {
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeDBClusterParametersRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
            request.SetSource(source);
        }

        Aws::RDS::Model::DescribeDBClusterParametersOutcome outcome =
                client.DescribeDBClusterParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
                }
                else {
                    parametersResult.push_back(parameter);
                }
            }

            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DescribeDBClusterParameters。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(ctx context.Context, parameterGroupName
 string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
```

```
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 DescribeDBClusterParameters。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
```

```
                        .build();
            } else {
                dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                        .dbClusterParameterGroupName(dbCLusterGroupName)
                        .source("user")
                        .build();
            }

            DescribeDbClusterParametersResponse response = rdsClient
                    .describeDBClusterParameters(dbParameterGroupsRequest);
            List<Parameter> dbParameters = response.parameters();
            String paraName;
            for (Parameter para : dbParameters) {
                // Only print out information about either auto_increment_offset
or
                // auto_increment_increment.
                paraName = para.parameterName();
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DescribeDBClusterParameters。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun describeDbClusterParameters(
    dbCLusterGroupName: String?,
    flag: Int,
) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest =
        if (flag == 0) {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
            }
        } else {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
                source = "user"
            }
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
```

```
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 DescribeDBClusterParameters。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)
```

```
    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
        Gets the parameters that are contained in a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
filtered
                            to contain only parameters that start with this
prefix.
        :param source: When specified, only parameters from this source are
retrieved.
                       For example, a source of 'user' retrieves only parameters
that
                       were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBClusterParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator =
self.rds_client.get_paginator("describe_db_cluster_parameters")
            for page in paginator.paginate(**kwargs):
                parameters += [
                    p
                    for p in page["Parameters"]
                    if p["ParameterName"].startswith(name_prefix)
                ]
        except ClientError as err:
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return parameters
```

- 如需 API 詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 DescribeDBClusterParameters。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   // Get the parameter group. rds.DescribeDbClusterParameterGroups
   // Get parameters in the group. This is a long list so you will have to
paginate. Find the auto_increment_offset and auto_increment_increment parameters
(by ParameterName). rds.DescribeDbClusterParameters
   // Parse the ParameterName, Description, and AllowedValues values and display
them.
   pub async fn cluster_parameters(&self) ->
Result<Vec<AuroraScenarioParameter>, ScenarioError> {
       let parameters_output = self
           .rds
           .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
           .await;

       if let Err(err) = parameters_output {
           return Err(ScenarioError::new(
               format!("Failed to retrieve parameters for
{DB_CLUSTER_PARAMETER_GROUP_NAME}"),
               &err,
           ));
       }

       let parameters = parameters_output
           .unwrap()
           .into_iter()
           .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
           .filter(|p|
FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
           .map(AuroraScenarioParameter::from)
```

```
            .collect::<Vec<_>>();

        Ok(parameters)
    }

    pub async fn describe_db_cluster_parameters(
        &self,
        name: &str,
    ) -> Result<Vec<DescribeDbClusterParametersOutput>,
 SdkError<DescribeDBClusterParametersError>>
    {
        self.inner
            .describe_db_cluster_parameters()
            .db_cluster_parameter_group_name(name)
            .into_paginator()
            .send()
            .try_collect()
            .await
    }

#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("d").build())
                .build()])
```

```
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

    let params = scenario.cluster_parameters().await.expect("cluster params");
    let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
    assert_eq!(
        names,
        vec!["auto_increment_offset", "auto_increment_increment"]
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let params = scenario.cluster_parameters().await;
    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message
 == "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 DescribeDBClusterParameters。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
TRY.
    DATA lv_marker TYPE /aws1/rdsstring VALUE ''.
    DATA lt_all_parameters TYPE /aws1/cl_rdsparameter=>tt_parameterslist.

    DO.
      DATA(lo_output) = lo_rds->describedbclusterparameters(
        iv_dbclusterparamgroupname = iv_param_group_name
        iv_source = iv_source
        iv_marker = lv_marker
      ).

      LOOP AT lo_output->get_parameters( ) INTO DATA(lo_param).
        IF iv_name_prefix IS INITIAL OR
           lo_param->get_parametername( ) CP |{ iv_name_prefix }*|.
          APPEND lo_param TO lt_all_parameters.
        ENDIF.
      ENDLOOP.

      lv_marker = lo_output->get_marker( ).
      IF lv_marker IS INITIAL.
        EXIT.
      ENDIF.
    ENDDO.

    ot_parameters = lt_all_parameters.
  CATCH /aws1/cx_rdsdbprmgrnotfndfault.
    " Re-raise exception - parameter group not found
    RAISE EXCEPTION TYPE /aws1/cx_rdsdbprmgrnotfndfault.
ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 DescribeDBClusterParameters。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DescribeDBClusterSnapshots` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `DescribeDBClusterSnapshots`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
```

```
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeDBClusterSnapshots。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DescribeDBClusterSnapshotsRequest request;
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBClusterSnapshotsOutcome outcome =
                    client.DescribeDBClusterSnapshots(request);

            if (outcome.IsSuccess()) {
                snapshot = outcome.GetResult().GetDBClusterSnapshots()[0];
            }
```

```
            else {
                std::cerr << "Error with Aurora::DescribeDBClusterSnapshots. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
 client);
                return false;
            }
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DescribeDBClusterSnapshots。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(ctx context.Context, snapshotName
 string) (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(ctx,
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 DescribeDBClusterSnapshots。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   public static void waitForSnapshotReady(RdsClient rdsClient, String
dbSnapshotIdentifier,
           String dbInstanceClusterIdentifier) {
       try {
           boolean snapshotReady = false;
           String snapshotReadyStr;
           System.out.println("Waiting for the snapshot to become available.");

           DescribeDbClusterSnapshotsRequest snapshotsRequest =
DescribeDbClusterSnapshotsRequest.builder()
                   .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
```

```
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .build();

            while (!snapshotReady) {
                DescribeDbClusterSnapshotsResponse response =
rdsClient.describeDBClusterSnapshots(snapshotsRequest);
                List<DBClusterSnapshot> snapshotList =
response.dbClusterSnapshots();
                for (DBClusterSnapshot snapshot : snapshotList) {
                    snapshotReadyStr = snapshot.status();
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true;
                    } else {
                        System.out.println(".");
                        Thread.sleep(sleepTime * 5000);
                    }
                }
            }

            System.out.println("The Snapshot is available!");

        } catch (RdsException | InterruptedException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DescribeDBClusterSnapshots。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun waitSnapshotReady(
    dbSnapshotIdentifier: String?,
    dbInstanceClusterIdentifier: String?,
) {
    var snapshotReady = false
    var snapshotReadyStr: String
    println("Waiting for the snapshot to become available.")

    val snapshotsRequest =
        DescribeDbClusterSnapshotsRequest {
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
            dbClusterIdentifier = dbInstanceClusterIdentifier
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!snapshotReady) {
            val response = rdsClient.describeDbClusterSnapshots(snapshotsRequest)
            val snapshotList = response.dbClusterSnapshots
            if (snapshotList != null) {
                for (snapshot in snapshotList) {
                    snapshotReadyStr = snapshot.status.toString()
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true
                    } else {
                        println(".")
                        delay(slTime * 5000)
                    }
                }
            }
        }
    }
    println("The Snapshot is available!")
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 DescribeDBClusterSnapshots。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_cluster_snapshot(self, snapshot_id):
        """
        Gets a DB cluster snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_cluster_snapshots(
                DBClusterSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBClusterSnapshots"][0]
        except ClientError as err:
```

```
            logger.error(
                "Couldn't get DB cluster snapshot %s. Here's why: %s: %s",
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- 如需 API 詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 DescribeDBClusterSnapshots。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DescribeDBClusters` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `DescribeDBClusters`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
```

```
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
cluster.</param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            if (response.DBClusters != null)
            {
                results.AddRange(response.DBClusters);
            }
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeDBClusters。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

//! Routine which gets a DB cluster description.
/*!
 \sa describeDBCluster()
 \param dbClusterIdentifier: A DB cluster identifier.
 \param clusterResult: The 'DBCluster' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBCluster(const Aws::String &dbClusterIdentifier,
                                       Aws::RDS::Model::DBCluster &clusterResult,
                                       const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBClustersRequest request;
    request.SetDBClusterIdentifier(dbClusterIdentifier);

    Aws::RDS::Model::DescribeDBClustersOutcome outcome =
            client.DescribeDBClusters(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        clusterResult = outcome.GetResult().GetDBClusters()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_CLUSTER_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::GDescribeDBClusters. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB cluster does not exist.
        // Instead, clusterResult is set to empty.
    else {
        clusterResult = Aws::RDS::Model::DBCluster();
    }

    return result;

}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DescribeDBClusters。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(ctx context.Context, clusterName string)
 (*types.DBCluster, error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(ctx,
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
```

```
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 DescribeDBClusters。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .build();
           } else {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .source("user")
```

```
                        .build();
            }

            DescribeDbClusterParametersResponse response = rdsClient
                    .describeDBClusterParameters(dbParameterGroupsRequest);
            List<Parameter> dbParameters = response.parameters();
            String paraName;
            for (Parameter para : dbParameters) {
                // Only print out information about either auto_increment_offset
or
                // auto_increment_increment.
                paraName = para.parameterName();
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DescribeDBClusters。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun describeDbClusterParameters(
    dbCLusterGroupName: String?,
    flag: Int,
) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest =
        if (flag == 0) {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
            }
        } else {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
                source = "user"
            }
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
```

```
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 DescribeDBClusters。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)
```

```
    def get_db_cluster(self, cluster_name):
        """
        Gets data about an Aurora DB cluster.

        :param cluster_name: The name of the DB cluster to retrieve.
        :return: The retrieved DB cluster.
        """
        try:
            response = self.rds_client.describe_db_clusters(
                DBClusterIdentifier=cluster_name
            )
            cluster = response["DBClusters"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBClusterNotFoundFault":
                logger.info("Cluster %s does not exist.", cluster_name)
            else:
                logger.error(
                    "Couldn't verify the existence of DB cluster %s. Here's why:
%s: %s",
                    cluster_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return cluster
```

- 如需 API 詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 DescribeDBClusters。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
           )
           .await;
       if let Err(err) = create_db_cluster {
           return Err(ScenarioError::new(
               "Failed to create DB Cluster with cluster group",
               &err,
           ));
       }

       self.db_cluster_identifier = create_db_cluster
           .unwrap()
```

```
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
```

```
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
```

```
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn describe_db_clusters(
        &self,
        id: &str,
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
        self.inner
            .describe_db_clusters()
            .db_cluster_identifier(id)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
```

```
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
```

```
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

```
#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
```

```
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
```

```
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
```

```
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 DescribeDBClusters。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DescribeDBEngineVersions` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `DescribeDBEngineVersions`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family
name.</param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeDBEngineVersions。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBEngineVersions(const Aws::String &engineName,
                                         const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                         const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
    }

    engineVersionsResult.clear();
    Aws::String marker; // The marker is used for pagination.
    do {
```

```
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
                client.DescribeDBEngineVersions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersions =
                    outcome.GetResult().GetDBEngineVersions();

            engineVersionsResult.insert(engineVersionsResult.end(),
                                        engineVersions.begin(),
 engineVersions.end());
            marker  = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBEngineVersionsRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    } while (!marker.empty());

    return true;
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DescribeDBEngineVersions。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(ctx context.Context, engine string,
 parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 DescribeDBEngineVersions。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   public static void describeDBEngines(RdsClient rdsClient) {
       try {
           DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                   .engine("aurora-mysql")
                   .defaultOnly(true)
                   .maxRecords(20)
                   .build();

           DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
           List<DBEngineVersion> engines = response.dbEngineVersions();

           // Get all DBEngineVersion objects.
           for (DBEngineVersion engineOb : engines) {
               System.out.println("The name of the DB parameter group family for
the database engine is "
                       + engineOb.dbParameterGroupFamily());
               System.out.println("The name of the database engine " +
engineOb.engine());
               System.out.println("The version number of the database engine " +
engineOb.engineVersion());
           }

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DescribeDBEngineVersions。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Get a list of allowed engine versions.
suspend fun getAllowedClusterEngines(dbParameterGroupFamilyVal: String?) {
    val versionsRequest =
        DescribeDbEngineVersionsRequest {
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(versionsRequest)
        response.dbEngineVersions?.forEach { dbEngine ->
            println("The engine version is ${dbEngine.engineVersion}")
            println("The engine description is ${dbEngine.dbEngineDescription}")
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 DescribeDBEngineVersions。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_engine_versions(self, engine, parameter_group_family=None):
        """
        Gets database engine versions that are available for the specified engine
        and parameter group family.

        :param engine: The database engine to look up.
        :param parameter_group_family: When specified, restricts the returned
 list of
                                       engine versions to those that are
 compatible with
                                       this parameter group family.
        :return: The list of database engine versions.
        """
```

```
        try:
            kwargs = {"Engine": engine}
            if parameter_group_family is not None:
                kwargs["DBParameterGroupFamily"] = parameter_group_family
            response = self.rds_client.describe_db_engine_versions(**kwargs)
            versions = response["DBEngineVersions"]
        except ClientError as err:
            logger.error(
                "Couldn't get engine versions for %s. Here's why: %s: %s",
                engine,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return versions
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 DescribeDBEngineVersions。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   // Get available engine families for Aurora MySql.
rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
   pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
ScenarioError> {
       let describe_db_engine_versions =
self.rds.describe_db_engine_versions(DB_ENGINE).await;
       trace!(versions=?describe_db_engine_versions, "full list of versions");
```

```
        if let Err(err) = describe_db_engine_versions {
            return Err(ScenarioError::new(
                "Failed to retrieve DB Engine Versions",
                &err,
            ));
        };

        let version_count = describe_db_engine_versions
            .as_ref()
            .map(|o| o.db_engine_versions().len())
            .unwrap_or_default();
        info!(version_count, "got list of versions");

        // Create a map of engine families to their available versions.
        let mut versions = HashMap::<String, Vec<String>>::new();
        describe_db_engine_versions
            .unwrap()
            .db_engine_versions()
            .iter()
            .filter_map(
                |v| match (&v.db_parameter_group_family, &v.engine_version) {
                    (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
                    _ => None,
                },
            )
            .for_each(|(family, version)|
versions.entry(family).or_default().push(version));

        Ok(versions)
    }

    pub async fn describe_db_engine_versions(
        &self,
        engine: &str,
    ) -> Result<DescribeDbEngineVersionsOutput,
SdkError<DescribeDBEngineVersionsError>> {
        self.inner
            .describe_db_engine_versions()
            .engine(engine)
            .send()
            .await
    }
```

```
#[tokio::test]
async fn test_scenario_get_engines() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Ok(DescribeDbEngineVersionsOutput::builder()
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1a")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1b")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f2")
                        .engine_version("f2a")
                        .build(),
                )
                .db_engine_versions(DbEngineVersion::builder().build())
                .build())
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;

    assert_eq!(
        versions_map,
        Ok(HashMap::from([
            ("f1".into(), vec!["f1a".into(), "f1b".into()]),
            ("f2".into(), vec!["f2a".into()])
        ]))
    );
}
```

```
#[tokio::test]
async fn test_scenario_get_engines_failed() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_engine_versions error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;
    assert_matches!(
        versions_map,
        Err(ScenarioError { message, context: _ }) if message == "Failed to
 retrieve DB Engine Versions"
    );
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 DescribeDBEngineVersions。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    " iv_engine                = 'mysql'
    " iv_dbparametergroupfamily = 'mysql8.0' (optional - filters by parameter
group family)
    TRY.
        oo_result = lo_rds->describedbengineversions(
          iv_engine                = iv_engine
          iv_dbparametergroupfamily = iv_dbparametergroupfamily ).
        DATA(lv_version_count) = lines( oo_result->get_dbengineversions( ) ).
        MESSAGE |Retrieved { lv_version_count } engine versions.| TYPE 'I'.
    ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 DescribeDBEngineVersions。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DescribeDBInstances` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `DescribeDBInstances`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
```

```
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeDBInstances。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);
```

```
//! Routine which gets a DB instance description.
/*!
 \sa describeDBCluster()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBInstance(const Aws::String &dbInstanceIdentifier,
                                        Aws::RDS::Model::DBInstance
 &instanceResult,
                                        const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
            client.DescribeDBInstances(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::DescribeDBInstances. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
    }

    return result;
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DescribeDBInstances。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(ctx context.Context, instanceName string)
 (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
```

```
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 DescribeDBInstances。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   // Waits until the database instance is available.
   public static void waitForInstanceReady(RdsClient rdsClient, String
dbClusterIdentifier) {
       boolean instanceReady = false;
       String instanceReadyStr;
       System.out.println("Waiting for instance to become available.");
       try {
           DescribeDbClustersRequest instanceRequest =
DescribeDbClustersRequest.builder()
                   .dbClusterIdentifier(dbClusterIdentifier)
                   .build();

           while (!instanceReady) {
               DescribeDbClustersResponse response =
rdsClient.describeDBClusters(instanceRequest);
               List<DBCluster> clusterList = response.dbClusters();
               for (DBCluster cluster : clusterList) {
                   instanceReadyStr = cluster.status();
                   if (instanceReadyStr.contains("available")) {
```

```
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database cluster is available!");

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DescribeDBInstances。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun waitDBAuroraInstanceReady(dbInstanceIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")
    val instanceRequest =
        DescribeDbInstancesRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
        }

    var endpoint = ""
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
```

```
            val response = rdsClient.describeDbInstances(instanceRequest)
            response.dbInstances?.forEach { instance ->
                instanceReadyStr = instance.dbInstanceStatus.toString()
                if (instanceReadyStr.contains("available")) {
                    endpoint = instance.endpoint?.address.toString()
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database instance is available! The connection endpoint is
 $endpoint")
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 DescribeDBInstances。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client
```

```
    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.

        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return db_inst
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 DescribeDBInstances。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
```

```
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
```

```
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
```

```
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
 DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
```

```
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));
```

```
    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
```

```
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
```

```
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _})
if message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
Describe Cluster
    tokio::time::resume();
```

```
    let _ = assertions.await;
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 DescribeDBInstances。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DescribeOrderableDBInstanceOptions` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `DescribeOrderableDBInstanceOptions`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string
engineVersion)
    {
```

```
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeOrderableDBInstanceOptions。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets available DB instance classes, displays the list
//! to the user, and returns the user selection.
```

```
/*!
 \sa chooseDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::chooseDBInstanceClass(const Aws::String &engine,
                                           const Aws::String &engineVersion,
                                           Aws::String &dbInstanceClass,
                                           const Aws::RDS::RDSClient &client) {
    std::vector<Aws::String> instanceClasses;
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
        request.SetEngine(engine);
        request.SetEngineVersion(engineVersion);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
                client.DescribeOrderableDBInstanceOptions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
 &options =
                    outcome.GetResult().GetOrderableDBInstanceOptions();
            for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
 options) {
                const Aws::String &instanceClass = option.GetDBInstanceClass();
                if (std::find(instanceClasses.begin(), instanceClasses.end(),
                              instanceClass) == instanceClasses.end()) {
                    instanceClasses.push_back(instanceClass);
                }
            }
            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeOrderableDBInstanceOptions.
 "
                      << outcome.GetError().GetMessage()
                      << std::endl;
```

```
            return false;
        }
    } while (!marker.empty());

    std::cout << "The available DB instance classes for your database engine
 are:"
              << std::endl;
    for (int i = 0; i < instanceClasses.size(); ++i) {
        std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
    }

    int choice = askQuestionForIntRange(
            "Which DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
    return true;
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DescribeOrderableDBInstanceOptions。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
```

```
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
  }
 }
 return instances, err
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 DescribeOrderableDBInstanceOptions。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   public static void describeDBEngines(RdsClient rdsClient) {
       try {
           DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                   .engine("aurora-mysql")
                   .defaultOnly(true)
                   .maxRecords(20)
                   .build();

           DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
           List<DBEngineVersion> engines = response.dbEngineVersions();

           // Get all DBEngineVersion objects.
           for (DBEngineVersion engineOb : engines) {
               System.out.println("The name of the DB parameter group family for
the database engine is "
                       + engineOb.dbParameterGroupFamily());
               System.out.println("The name of the database engine " +
engineOb.engine());
               System.out.println("The version number of the database engine " +
engineOb.engineVersion());
           }

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DescribeOrderableDBInstanceOptions。

PowerShell

Tools for PowerShell V4

範例 1：此範例列出了支援 AWS 區域中特定資料庫執行個體類別的資料庫引擎版本。

```
$params = @{
  Engine = 'aurora-postgresql'
  DBInstanceClass = 'db.r5.large'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

範例 2：此範例列出 AWS 區域中特定資料庫引擎版本支援的資料庫執行個體類別。

```
$params = @{
  Engine = 'aurora-postgresql'
  EngineVersion = '13.6'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 DescribeOrderableDBInstanceOptions。

Tools for PowerShell V5

範例 1：此範例列出了支援 AWS 區域中特定資料庫執行個體類別的資料庫引擎版本。

```
$params = @{
  Engine = 'aurora-postgresql'
  DBInstanceClass = 'db.r5.large'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

範例 2：此範例列出 AWS 區域中特定資料庫引擎版本支援的資料庫執行個體類別。

```
$params = @{
```

```
  Engine = 'aurora-postgresql'
  EngineVersion = '13.6'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 DescribeOrderableDBInstanceOptions。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_orderable_instances(self, db_engine, db_engine_version):
        """
        Gets DB instance options that can be used to create DB instances that are
```

```
        compatible with a set of specifications.

        :param db_engine: The database engine that must be supported by the DB
instance.
        :param db_engine_version: The engine version that must be supported by
the DB instance.
        :return: The list of DB instance options that can be used to create a
compatible DB instance.
        """
        try:
            inst_opts = []
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return inst_opts
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 DescribeOrderableDBInstanceOptions。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   pub async fn get_instance_classes(&self) -> Result<Vec<String>,
ScenarioError> {
       let describe_orderable_db_instance_options_items = self
           .rds
           .describe_orderable_db_instance_options(
               DB_ENGINE,
               self.engine_version
                   .as_ref()
                   .expect("engine version for db instance options")
                   .as_str(),
           )
           .await;

       describe_orderable_db_instance_options_items
           .map(|options| {
               options
                   .iter()
                   .filter(|o| o.storage_type() == Some("aurora"))
                   .map(|o|
o.db_instance_class().unwrap_or_default().to_string())
                   .collect::<Vec<String>>()
           })
           .map_err(|err| ScenarioError::new("Could not get available instance
classes", &err))
   }

   pub async fn describe_orderable_db_instance_options(
       &self,
       engine: &str,
       engine_version: &str,
   ) -> Result<Vec<OrderableDbInstanceOption>,
SdkError<DescribeOrderableDBInstanceOptionsError>>
   {
       self.inner
           .describe_orderable_db_instance_options()
           .engine(engine)
           .engine_version(engine_version)
           .into_paginator()
           .items()
           .send()
           .try_collect()
           .await
   }
```

```
#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Ok(vec![
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora-iopt1")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t2")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t3")
                    .storage_type("aurora")
                    .build(),
            ])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario
        .set_engine("aurora-mysql", "aurora-mysql8.0")
        .await
        .expect("set engine");
```

```
    let instance_classes = scenario.get_instance_classes().await;

    assert_eq!(
        instance_classes,
        Ok(vec!["t1".into(), "t2".into(), "t3".into()])
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(

 DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_family = Some("aurora-mysql".into());
    scenario.engine_version = Some("aurora-mysql8.0".into());

    let instance_classes = scenario.get_instance_classes().await;

    assert_matches!(
        instance_classes,
        Err(ScenarioError {message, context: _}) if message == "Could not get
 available instance classes"
    );
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 DescribeOrderableDBInstanceOptions。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    " iv_engine        = 'mysql'
    " iv_engineversion = '8.0.35'
    TRY.
        oo_result = lo_rds->descrorderabledbinstoptions(
          iv_engine        = iv_engine
          iv_engineversion = iv_engineversion ).
        DATA(lv_option_count) = lines( oo_result-
>get_orderabledbinstoptions( ) ).
        MESSAGE |Retrieved { lv_option_count } orderable DB instance options.|
 TYPE 'I'.
    ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 DescribeOrderableDBInstanceOptions。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`ModifyDBClusterParameterGroup` 搭配 AWS SDK 使用

下列程式碼範例示範如何使用 `ModifyDBClusterParameterGroup`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</
param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string
 groupName, List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable.GetValueOrDefault() && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the
 allowed values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
```

```
            DBClusterParameterGroupName = groupName,
        };

        var result = await
_amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 ModifyDBClusterParameterGroup。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::ModifyDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetParameters(updateParameters);

        Aws::RDS::Model::ModifyDBClusterParameterGroupOutcome outcome =
                client.ModifyDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
modified."
                      << std::endl;
        }
        else {
```

```
            std::cerr << "Error with Aurora::ModifyDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 ModifyDBClusterParameterGroup。

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(ctx context.Context,
 parameterGroupName string, params []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(ctx,
  &rds.ModifyDBClusterParameterGroupInput{
```

```
    DBClusterParameterGroupName: aws.String(parameterGroupName),
    Parameters:                  params,
   })
  if err != nil {
   log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
   return err
  } else {
   return nil
  }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的 ModifyDBClusterParameterGroup。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
       try {
           DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .maxRecords(20)
                   .build();

           List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                   .dbClusterParameterGroups();
           for (DBClusterParameterGroup group : groups) {
               System.out.println("The group name is " +
group.dbClusterParameterGroupName());
```

```
                System.out.println("The group ARN is " +
 group.dbClusterParameterGroupArn());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 ModifyDBClusterParameterGroup。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
// Modify the auto_increment_offset parameter.
suspend fun modifyDBClusterParas(dClusterGroupName: String?) {
    val parameter1 =
        Parameter {
            parameterName = "auto_increment_offset"
            applyMethod = ApplyMethod.fromValue("immediate")
            parameterValue = "5"
        }

    val paraList = ArrayList<Parameter>()
    paraList.add(parameter1)
    val groupRequest =
        ModifyDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dClusterGroupName
            parameters = paraList
        }
```

```
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.modifyDbClusterParameterGroup(groupRequest)
        println("The parameter group ${response.dbClusterParameterGroupName} was
 successfully modified")
    }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 ModifyDBClusterParameterGroup。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def update_parameters(self, parameter_group_name, update_parameters):
```

```
        """
        Updates parameters in a custom DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to update.
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                Parameters=update_parameters,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- 如需 API 詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的 ModifyDBClusterParameterGroup。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    // Modify both the auto_increment_offset and auto_increment_increment
parameters in one call in the custom parameter group. Set their ParameterValue
fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
```

```
    pub async fn update_auto_increment(
        &self,
        offset: u8,
        increment: u8,
    ) -> Result<(), ScenarioError> {
        let modify_db_cluster_parameter_group = self
            .rds
            .modify_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value(format!("{offset}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value(format!("{increment}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ],
            )
            .await;

        if let Err(error) = modify_db_cluster_parameter_group {
            return Err(ScenarioError::new(
                "Failed to modify cluster parameter group",
                &error,
            ));
        }

        Ok(())
    }

    pub async fn modify_db_cluster_parameter_group(
        &self,
        name: &str,
        parameters: Vec<Parameter>,
    ) -> Result<ModifyDbClusterParameterGroupOutput,
SdkError<ModifyDBClusterParameterGroupError>>
    {
        self.inner
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
```

```
            .set_parameters(Some(parameters))
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .withf(|name, params| {
            assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(
                params,
                &vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value("10")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value("20")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ]
            );
            true
        })
        .return_once(|_, _|
 Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

    let scenario = AuroraScenario::new(mock_rds);

    scenario
        .update_auto_increment(10, 20)
        .await
        .expect("update auto increment");
}

#[tokio::test]
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
            Err(SdkError::service_error(

 ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let update = scenario.update_auto_increment(10, 20).await;
    assert_matches!(update, Err(ScenarioError { message, context: _}) if message
 == "Failed to modify cluster parameter group");
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 ModifyDBClusterParameterGroup。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    TRY.
        oo_result = lo_rds->modifydbclusterparamgroup(
          iv_dbclusterparamgroupname = iv_param_group_name
          it_parameters = it_update_parameters
        ).
```

```
      CATCH /aws1/cx_rdsdbprmgrnotfndfault.
        " Re-raise exception - parameter group not found
        RAISE EXCEPTION TYPE /aws1/cx_rdsdbprmgrnotfndfault.
      CATCH /aws1/cx_rdsinvdbprmgrstatef00.
        " Re-raise exception - invalid state
        RAISE EXCEPTION TYPE /aws1/cx_rdsinvdbprmgrstatef00.
    ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 ModifyDBClusterParameterGroup。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

Aurora AWS SDKs的案例

下列程式碼範例示範如何在 Aurora AWS SDKs中實作常見案例。這些案例會向您示範如何呼叫 Aurora 中的多個函數，或與其他 AWS 服務合併以完成特定任務。每個案例均包含完整原始碼的連結，您可在連結中找到如何設定和執行程式碼的相關指示。

案例的目標是獲得中等水平的經驗，協助您了解內容中的服務動作。

範例

- 建立出借圖書館 REST API
- 建立 Aurora 無伺服器工作項目追蹤器

建立出借圖書館 REST API

下列程式碼範例顯示如何使用 Amazon Aurora 資料庫支援的 REST API 來建立出借圖書館，讓贊助人可以借書與還書。

Python

適用於 Python 的 SDK (Boto3)

示範如何使用 適用於 Python (Boto3) 的 AWS SDK 搭配 Amazon Relational Database Service (Amazon RDS) API 和 AWS Chalice 來建立由 Amazon Aurora 資料庫支援的 REST API。Web 服務是完全無伺服器的，表示這是一種贊助人可以借書與還書的簡單出借圖書館。了解如何：

- 建立與管理無伺服器的 Aurora 資料庫叢集。
- 使用 AWS Secrets Manager 管理資料庫登入資料。
- 實作資料儲存層，該層使用 Amazon RDS 將資料移入和移出資料庫。
- 使用 AWS Chalice 將無伺服器 REST API 部署至 Amazon API Gateway 和 AWS Lambda。
- 使用 Request 套件來將請求傳送到 Web 服務。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- API Gateway
- Aurora
- Lambda
- Secrets Manager

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

建立 Aurora 無伺服器工作項目追蹤器

說明如何建立 Web 應用程式追蹤 Amazon Aurora Serverless 資料庫中的工作項目，並且使用 Amazon Simple Email Service (Amazon SES) 傳送報告。

.NET

適用於 .NET 的 SDK

示範如何使用 適用於 .NET 的 AWS SDK 建立 Web 應用程式，以使用 Amazon Simple Email Service (Amazon SES) 追蹤 Amazon Aurora 資料庫中的工作項目和電子郵件報告。這個範例使用以 React.js 建置的前端與 RESTful .NET 後端互動。

- 將 React Web 應用程式與 AWS 服務整合。
- 列出、新增、更新和刪除 Aurora 資料表中的項目。
- 使用 Amazon SES 傳送篩選工作項目的電子郵件報告。
- 使用隨附的 AWS CloudFormation 指令碼部署和管理範例資源。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- Aurora
- Amazon RDS
- Amazon RDS 資料服務
- Amazon SES

C++

適用於 C++ 的 SDK

說明如何建立可追蹤和報告存放在 Amazon Aurora Serverless 資料庫中的工作項目的 Web 應用程式。

如需完整的原始碼以及如何設定 C++ REST API 以查詢 Amazon Aurora Serverless 資料並供 React 應用程式使用的說明，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- Aurora
- Amazon RDS
- Amazon RDS 資料服務
- Amazon SES

Java

適用於 Java 2.x 的 SDK

說明如何建立可追蹤和報告存放在 Amazon RDS 資料庫中的工作項目的 Web 應用程式。

如需完整的原始碼以及如何設定 Spring REST API 以查詢 Amazon Aurora 無伺服器資料並供 React 應用程式使用的說明，請參閱 GitHub 上的完整範例。

如需完整的原始碼和如何設定及執行使用 JDBC API 之範例的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- Aurora
- Amazon RDS

- Amazon RDS 資料服務
- Amazon SES

JavaScript

適用於 JavaScript (v3) 的 SDK

示範如何使用 適用於 JavaScript 的 AWS SDK (v3) 建立 Web 應用程式，以使用 Amazon Simple Email Service (Amazon SES) 追蹤 Amazon Aurora 資料庫中的工作項目和電子郵件報告。這個範例使用以 React.js 建置的前端與 Express Node.js 後端互動。

- 將 React.js Web 應用程式與 整合。 AWS 服務
- 列出、新增和更新 Aurora 資料表中的項目。
- 使用 Amazon SES 傳送篩選工作項目的電子郵件報告。
- 使用隨附的 AWS CloudFormation 指令碼部署和管理範例資源。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- Aurora
- Amazon RDS
- Amazon RDS 資料服務
- Amazon SES

Kotlin

SDK for Kotlin

說明如何建立可追蹤和報告存放在 Amazon RDS 資料庫中的工作項目的 Web 應用程式。

如需完整的原始碼以及如何設定 Spring REST API 以查詢 Amazon Aurora 無伺服器資料並供 React 應用程式使用的說明，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- Aurora
- Amazon RDS
- Amazon RDS 資料服務

- Amazon SES

PHP

適用於 PHP 的 SDK

示範如何使用 適用於 PHP 的 AWS SDK 建立 Web 應用程式，以使用 Amazon Simple Email Service (Amazon SES) 追蹤 Amazon RDS 資料庫中的工作項目和電子郵件報告。這個範例使用以 React.js 建置的前端與 RESTful PHP 後端互動。

- 將 React.js Web 應用程式與 AWS 服務整合。
- 列出、新增、更新和刪除 Amazon RDS 資料表中的項目。
- 使用 Amazon SES 傳送篩選工作項目的電子郵件報告。
- 使用隨附的 AWS CloudFormation 指令碼部署和管理範例資源。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- Aurora
- Amazon RDS
- Amazon RDS 資料服務
- Amazon SES

Python

適用於 Python 的 SDK (Boto3)

示範如何使用 適用於 Python (Boto3) 的 AWS SDK 建立 REST 服務，以使用 Amazon Simple Email Service (Amazon SES) 追蹤 Amazon Aurora Serverless 資料庫中的工作項目和電子郵件報告。這個範例使用 Flask Web 框架來處理 HTTP 路由，並與 React 網頁整合以呈現功能完整的 Web 應用程式。

- 建置與 整合的 Flask REST 服務 AWS 服務。
- 讀取、寫入和更新儲存在 Aurora 無伺服器資料庫中的工作項目。
- 建立包含資料庫登入資料的 AWS Secrets Manager 秘密，並使用它來驗證對資料庫的呼叫。
- 使用 Amazon SES 傳送工作項目的電子郵件報告。

如需完整的原始碼和如何設定及執行的指示，請參閱 GitHub 上的完整範例。

此範例中使用的服務

- Aurora
- Amazon RDS
- Amazon RDS 資料服務
- Amazon SES

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

Amazon Aurora 的最佳實務

下方包含使用資料或將資料移轉至 Amazon Aurora DB 叢集的一般最佳實務和選項的詳細資訊。

部分 Amazon Aurora 最佳實務為特定資料庫引擎專屬。如需特定資料庫引擎專屬的 Aurora 最佳實務詳細資訊，請參閱以下內容：

資料庫引擎	最佳實務
Amazon Aurora MySQL	請參閱 Amazon Aurora MySQL 的最佳實務
Amazon Aurora PostgreSQL	請參閱 Amazon Aurora PostgreSQL 的最佳實務

> **Note**
>
> 如需 Aurora 的常用建議，請參閱 Amazon Aurora 的建議。

主題

- Amazon Aurora 的基本操作準則
- 資料庫執行個體 RAM 建議
- AWS 資料庫驅動程式
- 監控 Amazon Aurora
- 使用資料庫參數群組和資料庫叢集參數群組
- Amazon Aurora 最佳實務影片

Amazon Aurora 的基本操作準則

下列是使用 Amazon Aurora 時，每個使用者應遵循的基本操作準則。Amazon RDS 服務水準協議需要您遵循這些準則。

- 監控您的記憶體、CPU 和儲存體用量。您可以設定 Amazon CloudWatch 在使用模式變更或接近部署容量時通知您。如此一來，您就可以維護系統效能和可用性。

- 如果您的用戶端應用程式正在快取資料庫執行個體的網域名稱服務 (DNS) 資料，請設定少於 30 秒的存活期 (TTL) 值。資料庫執行個體的基礎 IP 位址可能會在容錯移轉後變更。因此，如果您的應用程式嘗試連線到不再服務的 IP 位址，快取 DNS 資料一段時間可能會導致連線失敗。在使用讀取器端點進行連線，且其中一個僅供讀取複本執行個體處於維護模式或遭刪除的情況下，有多個僅供讀取複本的 Aurora 資料庫叢集也會遇到連線失敗的問題。
- 測試 DB 叢集的容錯移轉，以了解您的使用案例執行此程序需時多長。測試容錯移轉可協助您確保存取資料庫叢集的應用程式可以在容錯移轉後，自動連線到新的資料庫叢集。

資料庫執行個體 RAM 建議

為了最佳化效能，請配置足夠的 RAM，使得您的工作集幾乎能完全在記憶體中。若要判斷您的工作集是否完全在記憶體中，請檢查 Amazon CloudWatch 中的下列指標：

- `VolumeReadIOPS` – 此指標會衡量從叢集磁碟區中讀取 I/O 操作的平均次數，以 5 分鐘為間隔進行報告。`VolumeReadIOPS` 的值應該很小並且穩定。在某些情況下，您可能會發現讀取 I/O 突然增加或比平常高。如果是這樣，請調查資料庫叢集中的資料庫執行個體，以查看哪些資料庫執行個體造成 I/O 增加。

  > **Tip**
  >
  > 如果您的 Aurora MySQL 叢集使用的是平行查詢，您可能會看到 `VolumeReadIOPS` 值增加。平行查詢不會使用緩衝集區。因此，雖然查詢速度很快，但這種最佳化處理可能會導致讀取操作和相關費用增加。

- `BufferCacheHitRatio` – 此指標會測量資料庫叢集中資料庫執行個體之緩衝區快取所提供的請求百分比。透過這個指標，您可以深入了解記憶體提供了多少資料量。

  高命中率表示您的資料庫執行個體有足夠的可用記憶體。低命中率表示您對此資料庫執行個體的查詢經常移至磁碟。請調查您的工作負載，以查看哪些查詢導致此行為。

如果您在調查工作負載後發現需要更多記憶體，請考慮將資料庫執行個體類別擴展至擁有更多 RAM 的類別。這麼做以後，您可以調查前述指標並繼續根據需要進行擴展。如果您的 Aurora 叢集大於 40 TB，請勿使用 db.t2、db.t3 或 db.t4g 執行個體類別。

如需詳細資訊，請參閱Amazon Aurora 的 Amazon CloudWatch 指標。

AWS 資料庫驅動程式

我們建議使用 驅動程式 AWS 套件進行應用程式連線。這些驅動程式旨在支援更快的切換和容錯移轉時間，以及使用 AWS Secrets Manager AWS Identity and Access Management (IAM) 和聯合身分進行身分驗證。 AWS 驅動程式依賴監控資料庫叢集狀態，並注意叢集拓撲以判斷新的寫入器。相較於開放原始碼驅動程式的數十秒，此方法可將切換和容錯移轉時間縮短為短短幾秒。

隨著推出新的服務功能，驅動程式 AWS 套件的目標是為這些服務功能提供內建支援。

如需詳細資訊，請參閱使用 AWS 驅動程式連線至 Aurora 資料庫叢集。

監控 Amazon Aurora

Amazon Aurora 提供各種您可以監控的指標和洞見，以判斷 Aurora 資料庫叢集的運作狀態和效能。您可以使用 AWS 管理主控台 AWS CLI、 和 CloudWatch API 等各種工具來檢視 Aurora 指標。您可以在績效詳情儀表板中檢視組合的績效詳情和 CloudWatch 指標，並監控資料庫執行個體。若要使用此監控檢視，必須為您的資料庫執行個體開啟績效詳情。如需此監控檢視的相關資訊，請參閱 使用 Performance Insights 儀表板檢視組合指標。

您可以針對特定時間區間建立效能分析報告，並檢視所識別出的洞見和解決問題的建議。如需詳細資訊，請參閱 在 Performance Insights 中建立效能分析報告。

使用資料庫參數群組和資料庫叢集參數群組

建議您先在測試資料庫叢集上嘗試進行資料庫參數群組和資料庫叢集參數群組變更，再將參數群組變更套用至生產資料庫叢集。資料庫引擎參數設定不當，可能產生各種意外影響，包括降低效能和系統不穩定。

修改資料庫引擎參數時請務必謹慎，在修改資料庫參數群組之前，請備份您的資料庫叢集。如需備份資料庫叢集的詳細資訊，請參閱備份與還原 Amazon Aurora 資料庫叢集。

Amazon Aurora 最佳實務影片

YouTube 上的 AWS 線上技術講座頻道包含有關建立和設定 Amazon Aurora 資料庫叢集更安全且高可用性之最佳實務的影片簡報。請參閱高可用性的 Amazon Aurora 最佳實務。

透過 Amazon Aurora 執行概念驗證

下列內容將說明如何設定並執行 Aurora 的概念驗證。概念驗證是一種調查，可用來確認 Aurora 是否適合您的應用程式。概念驗證可助您了解資料庫應用程式中的 Aurora 功能，以及 Aurora 與您目前資料庫環境的比較。此功能也會呈現您移動資料、轉換 SQL 程式碼、調整效能與目前的管理程序所需的心力。

此主題提供本功能的概觀，以及執行概念驗證牽涉的高階程序與決策的步驟概述 (詳列如下)。如需詳細說明，您可點選特定主題的連結，閱讀完整文件。

Aurora 概念驗證概觀

執行 Amazon Aurora 概念驗證時，您將了解將現有資料和 SQL 應用程式轉換至 Aurora 所需耗費的心力。您會使用一定量的資料和活動來代表您的生產環境，大規模執行 Aurora 的重要面向。目標是要確認 Aurora 的優勢，能夠完美解決您不想再使用舊有資料庫基礎設施的問題。在概念驗證的結尾，您將取得堅實的計畫，能夠執行更大規模的效能基準與應用程式測試。此時，您會了解生產部署過程中最大的作業項目。

下列最佳實務的建議有助您避開基準測試期間造成問題的常見錯誤。不過，此主題並未涵蓋執行基準測試和調整效能的逐步流程。相關程序會取決於您的工作負載，以及您使用的 Aurora 功能。如需詳細資訊，請參考效能相關文件，例如 管理 Aurora 資料庫叢集的效能和擴展、Amazon Aurora MySQL 效能增強功能、Amazon Aurora PostgreSQL 的效能和擴展 及在 Amazon Aurora 上使用績效詳情監控資料庫負載。

此主題資訊主要適用的應用程式，必須由您的組織撰寫程式碼及設計結構描述，而且要支援 MySQL 和 PostgreSQL 開放原始碼資料庫引擎。若您測試的商用應用程式或程式碼係由應用程式架構所產生，您可能會無法靈活應用所有準則。此時，請諮詢您的 AWS 代表，確認是否有適用您應用程式類型的 Aurora 最佳實務或案例研究。

1. 確認您的目標

評估將 Aurora 用於概念驗證時，您會選擇要訂定何種衡量方式，以及如何評估執行是否成功。

您必須確認應用程式的所有功能均與 Aurora 相容。由於 Aurora 主要版本直接相容於對應的 MySQL 和 PostgreSQL 主要版本，因此針對這些引擎開發的大多數應用程式也相容於 Aurora。不過，您仍必須依據每個應用程式，確認其相容性。

例如，您設定 Aurora 叢集時所選擇的部分組態，會影響您是否能夠/應該採用特定資料庫功能。您可從最一般用途的 Aurora 叢集著手，也就是佈建。接著，您可能要判斷特定組態 (如無伺服器或平行查詢) 能否為您的工作負載帶來效益。

詢問下列問題有助您確認目標並加以量化：

- Aurora 是否支援您工作負載使用案例的所有功能呢？
- 您想要多大的資料集大小或負載程度？ 您能否擴展至該程度？
- 您有何特定的查詢輸送量或延遲需求？ 您能否滿足這些需求？
- 您工作負載可承受的意外或非意外停機時間下限為多少？ 您能否達成這個目標？
- 操作效率的必要指標有哪些？ 您能否精準監控這些指標？
- Aurora 是否支援您特定的業務目標，例如降低成本、增加部署或佈建速度？ 您是否有辦法將這些目標量化？
- 您的工作負載能否滿足所有安全與合規要求？

請花些時間認識 Aurora 資料庫引擎和平台功能，同時檢閱服務文件，並記下所有能夠幫助您取得滿意結果的功能。其中一項實用功能或許為工作負載整合，請閱讀 AWS 資料庫部落格針對整合工作負載，如何依據 MySQL 的相容性規劃 Amazon Aurora 並將之優化相關文章。另一個以需求為基礎的擴展功能或許也很實用，請參閱 Amazon Aurora 使用者指南中的使用 Amazon Aurora 自動擴展搭配 Aurora 複本。其他功能則包括提升效能，或者簡化資料庫的操作。

2. 了解您的工作負載特性

依據您預期的使用案例來評估 Aurora。Aurora 是線上交易處理 (OLTP) 工作負載的理想選擇。您也可在留有即時 OLTP 資料的叢集上執行報告，無須佈建另一個資料倉儲叢集。您可檢視您使用案例是否具備下列特性，確認該案例是否屬於類似情境：

- 高度並行需求，同時間用戶端數量可達數十、數百或數千。
- 大量低延遲查詢 (數毫秒至數秒)。
- 短而即時的交易。
- 高選擇性的查詢模式，搭配索引式查閱。
- 如為 HTAP，能夠善加運用 Aurora 平行查詢的分析查詢。

選擇資料庫的關鍵因素之一就是資料的速度。高速表示資料插入與更新的次數相當頻繁。此類系統可能有數千個連線及數十萬個同時查詢，都會針對資料庫進行讀取和寫入。高速系統的查詢通常會影響相對少數的資料列，而且會存取相同資料列內的多個欄。

Aurora 係針對處理高速資料而設計，搭配單一 r4.16xlarge 資料庫執行個體的 Aurora 叢集會視其工作負載，每秒可能要處理超過 60 萬筆 SELECT 陳述式。同樣地，取決於工作負載而定，這樣的叢集每秒可以處理 200,000 個 INSERT、UPDATE，以及DELETE 陳述式。Aurora 為列儲存區資料庫，非常適合用在大量、高輸送量及高度平行的 OLTP 工作負載。

Aurora 也可在處理 OLTP 工作負載的相同叢集上執行報告查詢。Aurora 支援多達 15 個複本，每個複本平均會在主要執行個體的 10—20 毫秒內。分析師可即時查詢 OLTP 資料，無須將資料複製到另一個資料倉儲叢集。透過 Aurora 叢集使用平行查詢功能，您可將多數處理、篩選和彙總作業，卸載至廣泛分散的 Aurora 儲存子系統。

運用此規畫階段熟悉 Aurora、其他 AWS 服務、AWS 管理主控台 和 AWS CLI 的功能。此外，請確認如何在概念驗證中，將這些功能與您預計要使用的其他工具搭配使用。

3. 透過 AWS 管理主控台或 AWS CLI 來練習

在下一步中，請透過 AWS 管理主控台 或 AWS CLI 練習，熟悉這些工具及 Aurora。

透過 AWS 管理主控台來練習

以下主要為 Aurora 資料庫叢集的初始活動，讓您熟悉 AWS 管理主控台 環境，並練習設定與修改 Aurora 叢集。若您使用 MySQL 相容或 PostgreSQL 相容的資料庫引擎來搭配 Amazon RDS，則使用 Aurora 時須仰賴相關知識。

善用 Aurora 的共用儲存模型和功能 (如複寫和快照)，可將整個資料庫叢集視為另一種您可以自由掌控的物件。在概念驗證期間，您可頻繁設定、拆卸和變更 Aurora 叢集的容量，不用因為最初的容量、資料庫設定和實際資料配置選擇而綁手綁腳。

若要開始使用，請設置空的 Aurora 叢集。請為您的最初測試選擇 provisioned (佈建的) 容量類型，以及 regional (區域性) 位置。

使用用戶端程式 (如 SQL 命令列應用程式) 來連接至該叢集。首先，您使用叢集端點來連接。您要連接至該端點，才能執行寫入操作，例如資料定義語言 (DDL) 陳述式和擷取、轉換、載入 (ETL) 流程。稍後在概念驗證中，您會使用讀取器端點來連接至查詢密集的工作階段，此端點會將查詢工作負載分散至叢集內的多個資料庫執行個體。

新增更多 Aurora 複本來擴展叢集，相關程序請參閱以 Amazon Aurora 進行複寫。擴展或縮減資料庫執行個體的方法，是改變 AWS 執行個體類別。請了解 Aurora 如何簡化這類操作，若您對系統容量的初估值不正確，即可知道如何調整，無須重新來過。

建立快照，並將其還原至不同的叢集。

檢查叢集指標來查看各個時間的活動，並確認指標如何套用至叢集內的資料庫執行個體。

若能在一開始就熟悉如何透過 AWS 管理主控台來執行這些操作，會非常有用。了解 Aurora 的功能後，即可進展至使用 AWS CLI，將這些操作自動化。下一節會進一步說明概念驗證期間，這些活動的程序和最佳實務詳細資訊。

透過 AWS CLI來練習

即使是在概念驗證設定中，我們也建議將部署和管理程序自動化，方法是透過您已經熟悉的 AWS CLI。若您使用 MySQL 相容或 PostgreSQL 相容的資料庫引擎來搭配 Amazon RDS，則使用 Aurora 時須仰賴相關知識。

Aurora 通常會牽涉到叢集內安排的資料庫執行個體群組。因此，許多操作會判斷與叢集相關聯的資料庫執行個體，然後重複在所有執行個體執行管理操作。

例如，您自動化的步驟可能包括建立 Aurora 叢集、透過更大的執行個體類別或額外的資料庫執行個體分別將之垂直、橫向擴展。如此一來，即可在概念驗證中的任何階段重複動作，並探索不同類或不同組態的 Aurora 叢集的模擬情境。

請認識基礎設施部署工具 (如 AWS CloudFormation) 的功能與限制。部分在概念驗證情境中的活動，可能並不適合用於生產。例如，CloudFormation 的修改行為會建立新的執行個體，並刪除現有執行個體與其中資料。如需此行為的詳細資訊，請參閱 AWS CloudFormation 使用者指南中的更新堆疊資源的行為。

4. 建立 Aurora 叢集

透過 Aurora，您可將資料庫執行個體新增至叢集，或者將資料庫執行個體擴展為更為強大的執行個體類別，藉此探索模擬情境。您也可用不同的組態設定來建立叢集，同時執行相同的工作負載。有了 Aurora，您將享受更多彈性來設置、拆卸或重新設定資料庫叢集。有鑑於此，在概念驗證的早期階段練習這些技巧就十分實用。如需建立 Aurora 叢集的一般程序，請參閱建立 Amazon Aurora 資料庫叢集。

如可行，請使用下列設定來著手。若您心中有特定使用案例，請略過此步驟。例如，若您的使用案例需要特殊類別的 Aurora 叢集，則可略過此步驟；或者，若您需要特定資料庫引擎和版本的組合，也可略過此步驟。

- 關閉 Easy create (輕鬆建立)。在概念驗證中，建議您注意所選擇的所有設定，之後即可建立相同或略為不同的叢集。
- 使用最新的資料庫引擎版本。這些資料庫引擎和版本組合廣泛相容於其他 Aurora 功能，相當多客戶都選擇普遍用於生產應用程式中。
  - Aurora MySQL 第 3.x 版 (與 MySQL 8.0 相容)
  - Aurora PostgreSQL 15.x 或 16.x 版
- 選擇 Dev/Test (開發/測試) 範本。此選項對於概念驗證活動並不重要。
- 若為 DB instance class (資料庫執行個體類別)，請選擇 Memory optimized classes (記憶體最佳化類別)，及其中一種 xlarge 執行個體類別。您稍後可上調或縮減此執行個體類別。
- 在 Multi-AZ Deployment (異地同步備份部署) 之下，請選擇 Create an Aurora Replica or Reader node in a different AZ (在不同 AZ 中建立 Aurora 複本或讀取器節點)。許多 Aurora 最實用的功能會牽涉多個資料庫執行個體的叢集，因此，新的叢集從至少兩個資料庫執行個體著手，都是合理做法。第二個資料庫執行個體請選擇不同可用區域，如此有助測試不同的高可用性情境。
- 為資料庫執行個體命名時，請使用通用的命名慣例。不要將任何叢集資料庫執行個體稱為「寫入器」，因為不同資料庫執行個體會視需要擔任這些角色。建議您使用 `clustername-az-serialnumber` 這類名稱，如 `myprodappdb-a-01`，如此即可明確辨識資料庫執行個體及其配置。
- 為 Aurora 叢集設定高備份保留期。保留期間長，您執行的時間點復原 (PITR) 期間最長可為 35 天。執行 DDL 和資料處理語言 (DML) 陳述式後，您可將資料庫重設回已知狀態。若您意外刪除或變更資料，也可加以還原。
- 在建立叢集時，開啟其他還原、記錄和監控功能。請開啟恢復、績效詳情、監控及日誌匯出底下的所有可用選項。啟用這些功能後，即可測試恢復、增強型監控或績效詳情等功能是否適合用於您的工作負載。在概念驗證期間，您也可輕鬆調查效能並執行故障診斷。

5. 設定您的結構描述

在 Aurora 叢集上，為您的應用程式設定資料庫、資料表、索引、外部索引鍵和其他結構描述物件。若您正遷離 MySQL 相容或 PostgreSQL 相容的資料庫系統，這個階段將十分簡單而直接。您要使用相同 SQL 語法和命令列或其他熟悉的用戶端應用程式，來搭配您的資料庫引擎。

如要在叢集上發出 SQL 陳述式，請尋找其叢集端點，並提供該值做為連接至您的用戶端應用程式的連線參數。在叢集詳細資訊頁面的 Connectivity (連線功能) 索引標籤中，即可找到叢集端點。叢集端點會標記 Writer (寫入器)，另一個標記為 Reader (讀取器) 的端點就是您可提供給最終使用者的唯讀連線，讓他們執行報告或其他唯讀查詢。如需連接至叢集的問題協助，請參閱連接至 Amazon Aurora 資料庫叢集。

若您要從不同的資料庫系統轉換結構描述和資料，此時必須略為修改結構描述，這些變更是為了要符合 SQL 語法及 Aurora 內的功能。您可以保留部分欄、限制、觸發條件或其他結構描述物件不變，這樣做可能很有用，因為這些物件對您的概念驗證目標而言可能並不重要，而且未來或許需要搭配 Aurora 相容性重新處理。

若您要遷離的資料庫系統使用 Aurora 以外的基礎引擎，請考慮使用 AWS Schema Conversion Tool (AWS SCT) 來簡化流程。如需詳細資訊，請參閱 AWS Schema Conversion Tool 使用者指南。如需遷移和移植活動的一般詳細資訊，請參閱 AWS 白皮書《將資料庫遷移至 Amazon Aurora》。

此時您可評估結構描述設定是否有效率不彰的問題，例如索引策略或其他資料表結構 (如分割資料表)。將應用程式部署在使用多個資料庫執行個體且工作負載繁重的叢集時，這類效率問題可能會放大。請考慮是否現在就微調這方面的效能層面，或者在稍後的活動 (如完整基準測試) 中再來微調。

6. 匯入資料

概念驗證期間，您會將資料或代表範本，遷離舊有的資料庫系統。如可行，請在每個資料表內至少設定一些資料，如此有助測試所有資料類型和結構描述功能的相容性。練習基本的 Aurora 功能後，請擴展資料量。完成概念驗證時，您應該要測試 ETL 工具、查詢和整體工作負載，其中的資料集要足以取得準確的結論。

您可使用數個技術，將實體資料或邏輯備份資料匯入 Aurora。如需詳細資訊，請參閱將資料遷移至 Amazon Aurora MySQL 資料庫叢集或將資料遷移至與 PostgreSQL 相容的 Amazon Aurora (視您在概念驗證中使用的資料庫引擎而一)。

測試您考慮要使用的 ETL 工具和技術，並選出最能符合需求的。請同時考量輸送量和彈性，例如，部分 ETL 工具會執行一次性傳輸，其他則會持續從舊的系統複寫至 Aurora。

從 MySQL 相容的系統遷移至 Aurora MySQL 時，您可使用原生資料傳輸工具，從 PostgreSQL 相容的系統遷移至 Aurora PostgreSQL 也是如此。若您要遷離的資料庫系統使用的基礎引擎不同於 Aurora，您可透過 AWS Database Migration Service (AWS DMS) 進行測試。如需 AWS DMS 的詳細資訊，請參閱 AWS Database Migration Service 使用者指南。

如需遷移和移植活動的詳細資訊，請參閱 AWS 白皮書 Aurora 遷移手冊。

7. 移植 SQL 程式碼

測試 SQL 及相關聯應用程式，在不同案例需要不同程度的心力，遷離 MySQL 或 PostgreSQL 相容的系統或其他類系統更是有差別。

- 若您正遷離 RDS for MySQL 或 RDS for PostgreSQL，SQL 所需的變動不大，您甚至可在 Aurora 測試原始 SQL 程式碼，並手動帶入必要變更。
- 同樣地，若您遷離的現場部署資料庫與 MySQL 或 PostgreSQL 相容，您也可測試原始 SQL 程式碼，並手動帶入變更。
- 若您正從商用資料庫遷離，則 SQL 就需要大幅變動，此時請考慮使用 AWS SCT。

此時您可評估結構描述設定是否有效率不彰的問題，例如索引策略或其他資料表結構 (如分割資料表)。請考慮是否現在就微調這方面的效能層面，或者在稍後的活動 (如完整基準測試) 中再來微調。

您可在應用程式中驗證資料庫連線邏輯。為了善加運用 Aurora 的分散式處理能力，讀取和寫入操作可能需要不同的連線，而查詢操作則使用相對短的工作階段。如需連線的詳細資訊，請參閱 9. 連線到 Aurora。

請確認您的生產資料庫是否有所犧牲或權衡以應付問題，並在概念驗證期間排入多一點時間，以改善結構描述設計和查詢。如要判斷您是否成功兼顧效能、營運成本和擴充能力，請在不同 Aurora 叢集上同時測試原始和修改後的應用程式。

如需遷移和移植活動的詳細資訊，請參閱 AWS 白皮書 Aurora 遷移手冊。

8. 指定組態設定

在 Aurora 概念驗證練習中，您亦可檢閱資料庫組態參數。您的 MySQL 或 PostgreSQL 組態設定，可能已經針對目前環境的效能和擴充能力進行調整。Aurora 儲存子系統會針對分散式雲端環境和高速儲存子系統進行調整，因此許多舊有的資料庫引擎設定並不適用。建議您透過預設 Aurora 組態設定來進行初始測試。只有當效能和擴充能力出現瓶頸時，才重新套用目前環境的設定。如有興趣，您可前往 AWS 資料庫部落格，在介紹 Aurora Storage Engine 中深入了解此主題。

對於特定應用或使用案例，Aurora 可輕鬆重複使用最佳組態設定。您會管理指派至整個叢集或特定資料庫執行個體的參數組，無須針對每個資料庫執行個體個別編輯組態檔案。例如，時區設定會套用至叢集內的所有資料庫執行個體，而且您可針對每個資料庫執行個體調整頁面快取大小設定。

請從一組預設參數組著手，再將變更套用至您需要微調的參數。如需使用參數群組的詳細資訊，請參閱 Amazon Aurora 資料庫叢集和資料庫執行個體參數。如需 Aurora 叢集適用或不適用的組態設定資訊，請參閱 Aurora MySQL 組態參數或 Amazon Aurora PostgreSQL 參數 (視您的資料庫引擎而異)。

9. 連線到 Aurora

初次設定結構描述與資料並執行範例查詢時，您會發現您可連線至 Aurora 叢集內的不同端點。所使用的端點取決於該操作為讀取 (如 SELECT 陳述式) 或者寫入 (如 CREATE 或 INSERT 陳述式)。在 Aurora 叢集上增加工作負載並測試 Aurora 功能時，您的應用程式務必將每個操作指派給適當的端點。

針對寫入操作使用叢集端點，您會一律連線至叢集內具有讀/寫能力的資料庫執行個體。根據預設，Aurora 叢集內只有一個資料庫執行個體具備讀/寫能力。這個執行個體稱為主要執行個體。若原始主要執行個體變得無法使用，Aurora 會啟動容錯移轉機制，另一個資料庫執行個體會成為主要執行個體。

同樣地，將 SELECT 陳述式指向讀取器端點，即可分散叢集內資料庫執行個體處理查詢的作業。系統會使用輪詢 DNS 解析，將每個讀取器連線指派給不同的資料庫執行個體。在唯讀資料庫 Aurora 複本上執行多數查詢作業，可減少主要執行個體上的負載，釋出資源來處理 DDL 和 DML 陳述式。

使用這些端點會減少對硬編碼主機名稱的依賴，有助應用程式更快速從資料庫執行個體的故障中恢復。

> **Note**
>
> Aurora 也可讓您自訂端點，不過概念驗證期間通常不需要這些端點。

Aurora 複本會產生複本延遲，不過通常僅 10 至 20 毫秒。您可監控複寫延遲，並判斷該延遲是否仍在您資料一致性所需的範圍內。在部分案例中，您的讀取查詢可能需要更強大的讀取一致性 (先寫後讀一致性)，此時可沿用叢集端點，不要改用讀取器端點。

如要善加運用 Aurora 的功能來達成分散式平行執行，您可能需要變更連線邏輯，目標是避免將所有讀取請求傳送至主要執行個體。唯讀 Aurora 複本及其中所有資料會準備好來處理 SELECT 陳述式。將應用程式的邏輯，撰寫成針對各種操作使用適當的端點。請遵守這些一般準則：

- 所有資料庫工作階段避免使用單一硬編碼連線字串。
- 如可行，請將寫入操作 (如 DDL 和 DML 陳述式) 納入用戶端應用程式碼的函數中。如此一來，不同操作就會使用各自的連線。
- 為查詢操作制定不同函數。Aurora 會將新的讀取者端點連線指派給不同的 Aurora 複本，以平衡讀取密集型應用程式的負載。
- 若是牽涉多組查詢的操作，請在每組相關查詢完成後，關閉並重新開啟讀取器端點的連線。若您的軟體堆疊具備此功能，請使用連接共用。將查詢導向不同的連線，有助 Aurora 分散叢集內資料庫執行個體的讀取工作負載。

如需 Aurora 連線管理和端點的一般資訊，請參閱連接至 Amazon Aurora 資料庫叢集。若要深入了解此主題，請參閱 Aurora MySQL 資料庫管理員手冊 – 連線管理。

10. 執行工作負載

結構描述、資料和組態設定就緒後，即可執行工作負載，開始演練叢集。在概念驗證中使用的工作負載，要能反映生產工作負載的主要面向。建議一律使用實際測試結果和工作負載來制定效能方面的決策，避免使用合成基準 (如 Sysbench 和 TPC-C)。如可行，請依據自己的結構描述、查詢模式和使用量來收集測量值。

並盡可能複製出應用程式執行所在的實際情況。例如，您在其上執行應用程式程式碼的 Amazon EC2 執行個體，通常要與 Aurora 叢集位於相同 AWS 區域和 Virtual Private Cloud (VPC)。若您的生產應用程式在多個橫跨不同可用區域的 EC2 執行個體上執行，請以相同方式設定您的概念驗證環境。如需 AWS 區域的詳細資訊，請參閱 Amazon RDS 使用者指南中的區域與可用區域。如需進一步了解 Amazon VPC 服務，請參閱《Amazon VPC 使用者指南》中的什麼是 Amazon VPC？。

確認應用程式運作的基本功能並能夠透過 Aurora 存取資料後，您可以演練 Aurora 叢集的各個面向。建議您試用的功能包括搭配負載平衡的並行連線，以及自動複寫。

此時您應已熟悉資料傳輸機制，因此可以運用大量範例資料來執行測試。

在這個階段，您就能看見組態設定變動的效果，例如記憶體限制和連線限制。請重複 8. 指定組態設定中的程序。

您亦可演練其他機制，例如建立並還原快照。舉例而言，您建立的叢集，可搭配不同 AWS 執行個體類別、AWS 複本數量等等。然後在每個叢集上，還原內含您的結構描述和所有資料的相同快照。如需此階段的詳細資訊，請參閱建立資料庫叢集快照及從資料庫叢集快照還原。

11. 測量效能

這部份的最佳實務能夠確保所有工具和流程均正確設定，可在工作負載操作期間快速隔離異常行為，也能讓您可靠辨識任何問題的合理原因。

您可在 Monitoring (監控) 索引標籤查看叢集的目前狀態，或者隨時間檢查趨勢。每個 Aurora 叢集或資料庫執行個體的主控台詳細資訊頁面，都會提供此索引標籤，其中會以圖表形式呈現 Amazon CloudWatch 監控服務的指標。您可依據名稱、資料庫執行個體和時間區間篩選指標。

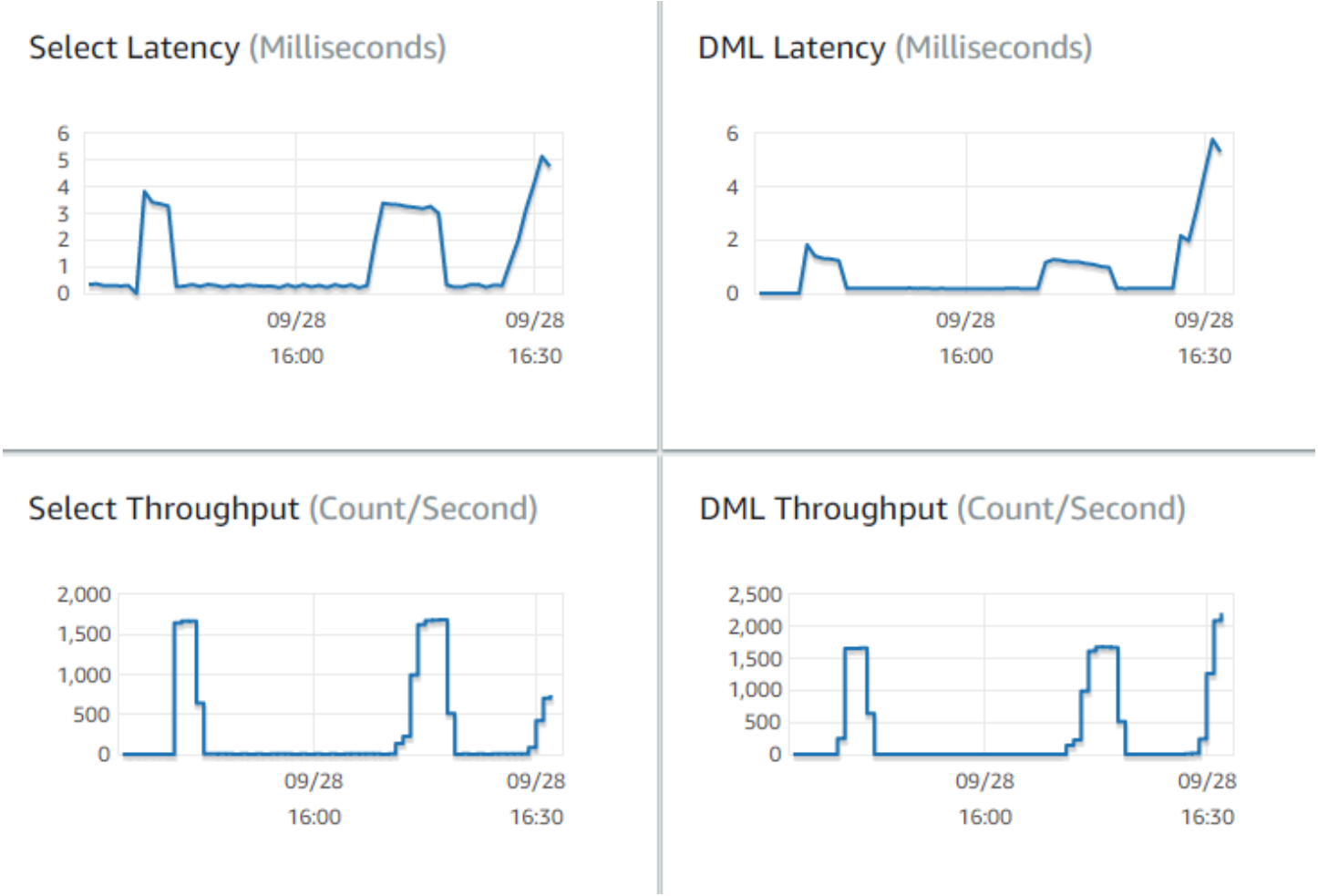
如需 Monitoring (監控) 索引標籤的更多選項，請在叢集設定內啟用增強型監控及績效詳情。若設定叢集時未選擇啟用這些選項，您也可以稍後再進行設定。

如要測量效能，您主要會仰賴呈現整個 Aurora 叢集的活動的圖表。您可確認 Aurora 複本是否有類似負載和回應時間，並查看作業是如何分割到讀/寫主要執行個體及唯讀 Aurora 複本。若資料庫執行個體之間並不平衡，或者某個問題僅影響一個資料庫執行個體，您可在 Monitoring (監控) 索引標籤內檢查該特定執行個體的情形。

模擬生產應用程式的環境和實際工作負載設定完成後，您可測量 Aurora 的效能。最重要的問題如下：

- Aurora 每秒處理幾個查詢？ 您可檢查 Throughput (傳輸量) 指標，查看不同類型作業的數據。
- Aurora 處理一個查詢平均要花多少時間？ 您可檢查 Latency (延遲) 指標，查看不同類型作業的數據。

若要檢視輸送量和延遲指標，請在 Amazon RDS 主控台中檢查指定 Aurora 叢集的監控索引標籤。下列螢幕擷取畫面顯示監控標籤上選取延遲、DML 延遲、選取輸送量和 DML 輸送量指標的範例。



若可以，請在您目前環境為這些指標建立基準值。若無法，請執行等同於生產應用程式的工作負載，藉此在 Aurora 上建構基準。例如，執行您的 Aurora 工作負載，其中同時上線使用者和查詢數量採用類似數量。接著，觀察這些值如何依據您測試不同的執行個體類別、叢集大小、組態設定等而變化。

若輸送量數據低於預期，請進一步調查影響工作負載資料庫效能的因素。同樣地，若延遲數據高於預期，請進一步調查。方法是：監控資料庫伺服器的次要指標 (CPU、記憶體等)，檢查資料庫執行個體是否接近這些限制值。您也可看到資料庫執行個體有多少多餘的容量，可處理更多並行查詢、對更大資料表的查詢等等。

> **Tip**
>
> 如要偵測落出預期範圍的指標值，請設定 CloudWatch 警示。

評估理想 Aurora 叢集大小和容量時，您找到的組態須能夠處理應用程式的尖峰效能，而且不會過度佈建資源。其中一個重要的因素在於在 Aurora 叢集中，為資料庫執行個體找到適當大小。首先選取的執行個體大小，要與您目前生產環境的 CPU 與記憶體容量類似，然後收集工作負載在該執行個體大小中的輸送量和延遲數據。接著，將執行個體擴展至下一個更大的大小，檢查輸送量和延遲數據是否改善。同樣地，縮減執行個體大小，並檢查延遲和輸送量數據是否不變。您的目標是在最小的執行個體上，取得最高的輸送量及最低的延遲。

> **Tip**
>
> 請調整 Aurora 叢集與相關聯資料庫執行個體的大小，確保現有容量足以處理突發、無法預測的流量高峰。以關鍵任務資料庫而言，請保留至少 20% 的備用 CPU 和記憶體容量。

執行效能測試的時間要夠長，才能測量資料庫在穩定的預備狀態下的效能。您可能要執行工作負載數分鐘、甚至數小時，才能到達穩定狀態。執行初期出現些許變動十分正常，變動出現的原因在於每個 Aurora 複本，會依據其處理的 SELECT 查詢來讓快取準備就緒。

Aurora 在處理交易式工作負載的效能最佳，其中涉及多個同時使用者和查詢。如要確保您啟動的負載足以實現最佳效能，請執行多執行緒的基準，或在效能測試中同時執行多個執行個體。請運用數百甚至數千個同時用戶端執行緒來測量效能，並模擬生產環境預期會同時出現的執行緒數量。您可能也要以更多執行緒執行其他壓力測試，藉以測量 Aurora 的擴充能力。

12. 演練 Aurora 高可用性

Aurora 的許多主要功能均與高可用性有關。這些功能包括自動複寫、自動容錯移轉、自動時間點還原備份，而且能夠在叢集中新增資料庫執行個體。這類功能提供的安全與可靠性，對於關鍵任務應用程式十分重要。

評估這些功能需要特定思維。在早期活動 (如效能測量) 中，您會觀察到系統在一切運作順利下的效能如何，而測試高可用性必須全面考慮最差情況的行為。您必須考量各種故障類型，即使該情況非常少見也不能忽略。建議您故意引發一些問題，確認系統可正確而快速地回復。

> **Tip**
>
> 在概念驗證中，將 Aurora 叢集內的所有資料庫執行個體都以相同 AWS 執行個體類別來設定。如此一來，即可測試 Aurora 的可用性功能，將資料庫執行個體離線以模擬故障時，無須大幅變更效能和擴充能力。

我們建議在每個 Aurora 叢集內使用至少兩個執行個體，一個 Aurora 叢集內的資料庫執行個體可橫跨至多三個可用區域 (AZ)，請將前兩個或三個資料庫執行個體放入不同 AZ。開始使用更大的叢集時，請將資料庫執行個體分散至您 AWS 區域內的所有可用區域。如此可增加容錯能力。即使一個問題影響整個 AZ，Aurora 可容錯移轉至不同 AZ 的資料庫執行個體。若您執行的叢集有超過三個執行個體，請將資料庫執行個體盡可能平均分散至所有三個 AZ。

> **Tip**
>
> Aurora 叢集的儲存體會獨立於資料庫執行個體，而每個 Aurora 叢集的儲存體都會橫跨三個 AZ。
>
> 測試高可用性功能時，您在測試叢集內使用的資料庫執行個體一律要具備相同的容量，在資料庫執行個體互相接管時，這樣就可避免效能、延遲等出現無法預測的變更。

如要進一步了解如何模擬故障情形來測試高可用性功能，請參閱使用錯誤注入查詢測試 Amazon Aurora MySQL。

在概念驗證演練中，一個目標是要找到資料庫執行個體的理想數量，以及最適合這些資料庫執行個體的執行個體類別。要達到這個目標，您必須兼顧高可用性及效能的各自需求。

以 Aurora 而言，叢集內有愈多資料庫執行個體，高可用性的效益就愈大。擁有的資料庫執行個體數較多，也會改善讀取密集型應用程式的可擴展性。Aurora 可以在唯讀 Aurora 複本中分佈多個連接，來進行 SELECT 查詢。

另一方面，限制資料庫執行個體的數量，會減少從主要節點的複寫流量。複寫流量會消耗網路頻寬，此為整體效能和擴充能力必須考量的另一面向。因此，對於寫入密集型 OLTP 應用程式而言，使用較少量的大型資料庫執行個體是更好的選擇，而非大量的小型資料庫執行個體。

在典型 Aurora 叢集中，一個資料庫執行個體 (主要執行個體) 會處理所有 DDL 和 DML 陳述式，其他資料庫執行個體 (Aurora 複本) 則僅處理 SELECT 陳述式。雖然資料庫執行個體的工作量不一定相同，我們建議叢集內的所有資料庫執行個體都採用相同的執行個體類別。如此一來，若故障發生使得 Aurora 將一個唯讀資料庫執行個體提升為新的主要執行個體，則主要執行個體就可保有相同容量。

若同一叢集內須使用不同容量的資料庫執行個體，請為這些資料庫執行個體設定容錯移轉方案。這些方案會決定容錯移轉機制提升 Aurora 複本的順序。將容量明顯較大或較小的資料庫執行個體放到更低的容錯移轉方案，如此才會最後才選擇它們進行提升。

演練 Aurora 的資料復原功能，例如自動時間點還原、手動快照及還原，以及叢集恢復。如適用，請將快照複製到其他 AWS 區域，並還原至其他 AWS 區域來模擬 DR 情境。

請調查您組織的復原時間目標 (RTO)、復原點目標 (RPO) 和地區備援的要求。多數組織將這些事項納入整體災難復原類別底下。請評估本節的 Aurora 高可用性功能在您災難復原流程情境中的運作情形，以確保符合您的 RTO 和 RPO 要求。

13. 後續作業

概念驗證流程成功後，您會確認 Aurora 解決方案是否適合您預期的工作負載。透過上述流程，您已檢查 Aurora 在實際操作環境中的運作情形，同時測量其是否符合您的成功條件。

透過 Aurora 設定完資料庫環境並開始運行後，接著您可開始執行更詳細的評估作業，最後才會遷移並進行生產部署。概念驗證流程或許會/不會納入這些其他步驟，而這取決於您的情況。如需遷移和移植活動的詳細資訊，請參閱 AWS 白皮書 Aurora 遷移手冊。

另一項後續作業則是，請考量工作負載相關的安全組態，並將其設計為滿足您生產環境的安全要求。請計劃要採取哪些控管措施，以保護 Aurora 叢集主要使用者登入資料的存取。請定義資料庫使用者的角色與責任，以控制對存放於 Aurora 叢集內資料的存取。請考量應用程式、指令碼和第三方工具或服務的資料庫存取要求。請探索 AWS 服務和功能，例如 AWS Secrets Manager 及 AWS Identity and Access Management (IAM) 身分驗證。

此時，您應已理解透過 Aurora 執行基準測試的程序和最佳實務，因此可能會需要其他效能調校。如需詳細資訊，請參閱管理 Aurora 資料庫叢集的效能和擴展、Amazon Aurora MySQL 效能增強功能、Amazon Aurora PostgreSQL 的效能和擴展 和在 Amazon Aurora 上使用績效詳情監控資料庫負載。若需要其他調校作業，請確認您已熟悉您在概念驗證期間收集的指標。在後續作業中，您建立的新叢集可能會選擇不同的組態設定、資料庫引擎和資料庫版本。或者，您可能會建立特殊類別的 Aurora 叢集，以滿足特定使用案例的需求。

例如，您可探索 Aurora 平行查詢叢集，供混合式交易/分析處理 (HTPA) 應用程式使用。為了災難復原或縮減延遲而需要分散的地理分布，您可探索 Aurora 全球資料庫。若您的工作負載為間歇性，或者您正在開發/測試情境中使用 Aurora，您可探索 Aurora Serverless 叢集。

您的生產叢集可能也需要處理大量傳入連線。若要了解這些技術，請參閱 AWS 白皮書 Aurora MySQL 資料庫管理員手冊 – 連線管理。

概念驗證之後，若您覺得 Aurora 不適合您的使用案例，請考慮其他 AWS 服務：

- 對於純粹的分析使用案例，工作負載受益於單欄式儲存格式和其他更適合 OLAP 工作負載的功能。處理此類使用案例的 AWS 服務包括下列各項：
  - Amazon Redshift
  - Amazon EMR
  - Amazon Athena
- 許多工作負載會受益於 Aurora 及這些一個或多個服務的結合。您可使用這些服務，將資料在服務之間移動：
  - AWS Glue
  - AWS DMS
  - 從 Amazon S3 匯入，如 Amazon Aurora 使用者指南內所述
  - 匯出至 Amazon S3，如 Amazon Aurora 使用者指南內所述
  - 許多其他熱門的 ETL 工具

Amazon Aurora 的安全性

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

安全性是 AWS 與您之間的共同責任。共同責任模型 將此描述為雲端的安全和雲端內的安全：

- 雲端的安全性 – AWS 負責保護在 Cloud AWS 中執行 AWS 服務的基礎設施。 AWS 也為您提供可安全使用的服務。在 AWS 合規計畫中，第三方稽核員會定期測試並驗證我們的安全功效。若要了解適用於 Amazon Aurora (Aurora) 的合規計劃，請參閱合規計劃的AWS 服務範圍。
- 雲端的安全性 – 您的責任取決於您使用 AWS 的服務。您也必須對其他因素負責，包括資料的敏感度、您組織的需求和適用的法律及法規。

本文件有助於您了解如何在使用 Amazon Aurora 時套用共同責任模型。下列主題說明如何設定 Amazon Aurora 以達到您的安全和合規目標。您也會了解如何使用其他 AWS 服務來協助您監控和保護 Amazon Aurora 資源。

您可以管理對 Amazon Aurora 資源以及資料庫叢集上資料庫的存取權限。您用來管理存取權限的方法取決於使用者需要利用 Amazon Aurora 執行何種類型的任務：

- 在以 Amazon VPC 服務為基礎的 virtual private cloud (VPC) 中執行您的資料庫叢集，以盡可能地取得最大的網路存取控制能力。如需在 VPC 中建立資料庫叢集的詳細資訊，請參閱 Amazon VPC 和 Amazon Aurora。
- 使用 AWS Identity and Access Management (IAM) 政策來指派許可，以決定允許誰管理 Amazon Aurora 資源。例如，您可以使用 IAM 來決定誰可以建立、描述、修改和刪除資料庫叢集、標記資源，或修改安全群組。

  若要檢閱 IAM 政策範例，請參閱 Amazon Aurora 以身分為基礎的政策範例。
- 使用安全群組來控制哪些 IP 位址或 Amazon EC2 執行個體可以連線至資料庫叢集上的資料庫。當您第一次建立資料庫叢集時，其防火牆會防止任何資料庫存取，但透過相關聯的安全群組所指定規則進行的資料庫存取除外。
- 對執行 Aurora MySQL 或 Aurora PostgreSQL 的資料庫叢集使用 Secure Socket Layer (SSL) 或 Transport Layer Security (TLS) 連線。如需搭配資料庫叢集使用 SSL/TLS 的詳細資訊，請參閱 使用 SSL/TLS 加密與資料庫叢集的連線。

- 使用 Amazon Aurora 加密來保護資料庫叢集和靜態快照。 Amazon Aurora 加密資料庫執行個體叢集會使用業界標準 AES-256 加密演算法，來加密託管資料庫叢集伺服器上的資料。如需詳細資訊，請參閱 加密 Amazon Aurora 資源。
- 使用資料庫引擎的安全功能，來控制誰可以登入資料庫叢集上的資料庫。這項功能的運作方式就好像資料庫位在您的本機網路上。

  如需 Aurora MySQL 安全的詳細資訊，請參閱 Amazon Aurora MySQL 的安全性。如需 Aurora PostgreSQL 安全的詳細資訊，請參閱 Amazon Aurora PostgreSQL 的安全性。

Aurora 是受管資料庫服務 Amazon Relational Database Service (Amazon RDS) 的一部分。Amazon RDS 是一項 Web 服務，可以讓雲端中關聯式資料庫的設定、操作和擴展更加簡單。如果您尚未熟悉 Amazon RDS，請參閱《Amazon RDS 使用者指南》。

Aurora 包括高效能的儲存子系統。其 MySQL 和 PostgreSQL 相容的資料庫引擎會自訂為善用該快速分散式儲存。Aurora 也會自動化並標準化資料庫叢集和複寫，這通常是資料庫設定和管理中最具挑戰性的層面。

對於 Amazon RDS 和 Aurora，您可以透過程式設計方式存取 RDS API，而且您可以使用 AWS CLI 以互動方式存取 RDS API。有些 RDS API 操作和 AWS CLI 命令同時適用於 Amazon RDS 和 Aurora，其他的則僅適用於 Amazon RDS 或 Aurora。如需 RDS API 操作的資訊，請參閱 Amazon RDS API 參考。如需 的詳細資訊 AWS CLI，請參閱 AWS Command Line Interface Amazon RDS 的參考。

> **Note**
>
> 您只須針對您的使用案例設定安全。您不需要為 Amazon Aurora 管理的程序設定安全存取。這些包括建立備份、自動容錯移轉以及其他程序。

如需管理對 Amazon Aurora 資源及在您資料庫叢集上資料庫存取權限的詳細資訊，請參閱下列主題。

主題

- 使用 Amazon Aurora 進行資料庫身分驗證
- 使用 Amazon Aurora 和 進行密碼管理 AWS Secrets Manager
- Amazon RDS 中的資料保護
- Amazon Aurora 的 Identity and access management
- 在 Amazon Aurora 中記錄和監控
- Amazon Aurora 的合規驗證

- Amazon Aurora 的復原功能
- Amazon Aurora 中的基礎設施安全
- Amazon RDS API 和界面 VPC 端點 (AWS PrivateLink)
- Amazon Aurora 的安全最佳實務
- 使用安全群組控制存取
- 主要使用者帳戶權限
- 使用 Amazon Aurora 的服務連結角色
- Amazon VPC 和 Amazon Aurora

使用 Amazon Aurora 進行資料庫身分驗證

Amazon Aurora 支援驗證資料庫使用者的數種方式。

預設情況下，所有資料庫叢集都可以使用密碼身分驗證。對於 Aurora MySQL 和 Aurora PostgreSQL，您也可以為相同的資料庫叢集新增 IAM 資料庫身分驗證和 Kerberos 身分驗證。

密碼、Kerberos 和 IAM 資料庫身分驗證會對資料庫使用不同的身分驗證方法。因此，特定使用者只能使用一種身分驗證方法登入資料庫。

對於 PostgreSQL，請僅針對特定資料庫的使用者使用下列其中一個角色設定：

- 若要使用 IAM 資料庫身分驗證，請將 `rds_iam` 角色指派給使用者。
- 若要使用 Kerberos 身分驗證，請將 `rds_ad` 角色指派給使用者。
- 若要使用密碼身分驗證，請勿指派 `rds_iam` 或 `rds_ad` 角色給使用者。

不要同時將 `rds_iam` 和 `rds_ad` 角色指派給 PostgreSQL 資料庫的使用者，無論是直接或間接透過巢狀授權存取。如果 `rds_iam` 角色新增至主要使用者，則 IAM 身分驗證優先於密碼身分驗證，因此主要使用者必須以 IAM 使用者身分登入。

> **Important**
>
> 我們強烈建議您不要直接在您的應用程式中使用主要使用者。而是遵循最佳實務，使用以應用程式所需的最低權限建立的資料庫使用者。

主題

- 密碼身分驗證
- IAM 資料庫身分驗證
- Kerberos 身分驗證

密碼身分驗證

使用密碼身分驗證，您的資料庫會執行使用者帳戶的所有管理。您可以使用 SQL 陳述式 (例如 `CREATE USER`) 搭配資料庫引擎指定密碼所需的適當子句，來建立使用者。例如，在 MySQL 中，陳述式為 `CREATE USER` *`name`* `IDENTIFIED BY` *`password`*，而在 PostgreSQL 中，陳述式為 `CREATE USER` *`name`* `WITH PASSWORD` *`password`*。

透過密碼身分驗證，您的資料庫可控制並驗證使用者帳戶。如果資料庫引擎具有強大的密碼管理功能，它們可以增強安全性。當您擁有較小的使用者社群時，使用密碼身分驗證來管理資料庫身分驗證可能會更容易。由於在此情況下會產生純文字密碼，因此與 整合 AWS Secrets Manager 可以增強安全性。

如需有關將 Secrets Manager 與 Amazon Aurora 搭配使用的詳細資訊，請參閱 《AWS Secrets Manager 使用者指南》中的建立基本秘密和輪換支援 Amazon RDS 資料庫的秘密。如需以程式設計方式擷取自訂應用程式中的密碼的相關資訊，請參閱 AWS Secrets Manager 使用者指南中的擷取密碼值。

IAM 資料庫身分驗證

您可以使用 AWS Identity and Access Management (IAM) 資料庫身分驗證來驗證資料庫叢集。透過此身分驗證方法，您在連接至資料庫叢集時就不需要使用密碼。而是改用身分驗證字符。

如需有關 IAM 資料庫身分驗證的詳細資訊，包括特定資料庫引擎可用性的相關資訊，請參閱 IAM 資料庫身分驗證。

Kerberos 身分驗證

Amazon Aurora 支援透過 Kerberos 和 Microsoft Active Directory 對資料庫使用者進行外部身分驗證。Kerberos 是網路身分驗證通訊協定，使用票證和對稱式金鑰加密技術，免除透過網路傳輸密碼的需要。Kerberos 已內建至 Active Directory，旨在驗證網路資源 (例如資料庫) 的使用者身分。

Amazon Aurora 對 Kerberos 和 Active Directory 的支援，提供了資料庫使用者的單一登入和集中式身分驗證優點。您可以在 Active Directory 中保留您的使用者登入資料。Active Directory 提供集中位置，可存放及管理多個資料庫叢集的憑證。

若要使用自我管理 Active Directory 中的登入資料，您需要設定與加入資料庫叢集之 Directory Service Microsoft Active Directory 的信任關係。

Aurora PostgreSQL 和 Aurora MySQL 支援使用整個樹系身分驗證或選擇性身分驗證的單向和雙向樹系信任關係。

在某些情況下，您可以透過外部信任關係設定 Kerberos 身分驗證。這需要自我管理的 Active Directory 具有其他設定。這包括但不限於 Kerberos 樹系搜尋順序。

Aurora 支援適用於 Aurora MySQL 和 Aurora PostgreSQL 資料庫叢集的 Kerberos 身分驗證。如需詳細資訊，請參閱針對 Aurora MySQL 使用 Kerberos 身分驗證及搭配 Aurora PostgreSQL 使用 Kerberos 身分驗證。

使用 Amazon Aurora 和 進行密碼管理 AWS Secrets Manager

Amazon Aurora 與 Secrets Manager 整合，以管理資料庫叢集的主要使用者密碼。

主題

- 區域和版本可用性
- Secrets Manager 與 Amazon Aurora 整合的限制
- 使用 管理主要使用者密碼的概觀 AWS Secrets Manager
- 使用 Secrets Manager 管理主要使用者密碼的優點
- Secrets Manager 整合所需的許可
- 在 中強制執行主要使用者密碼的 Aurora管理 AWS Secrets Manager
- 使用 Secrets Manager 管理資料庫叢集的主要使用者密碼
- 輪換資料庫叢集的主要使用者密碼秘密
- 檢視資料庫叢集秘密的詳細資訊

區域和版本可用性

功能可用性和支援會因每個資料庫引擎的特定版本以及 AWS 區域而有所不同。如需 Secrets Manager 與 Amazon Aurora 整合之版本和區域可用性的詳細資訊，請參閱 Secrets Manager 整合的支援區域和 Aurora 資料庫引擎。

Secrets Manager 與 Amazon Aurora 整合的限制

下列功能不支援使用 Secrets Manager 管理主要使用者密碼：

- Amazon RDS 藍/綠部署
- 屬於 Aurora 全域資料庫的資料庫叢集。
- Aurora Serverless v1 資料庫叢集
- 跨區域僅供讀取複本
- 二進位日誌外部複寫

使用 管理主要使用者密碼的概觀 AWS Secrets Manager

使用 AWS Secrets Manager時，您可以使用 API 呼叫 Secrets Manager 以程式設計方式擷取秘密，取代程式碼中的硬式編碼登入資料，包括資料庫密碼。如需 Secrets Manager 的詳細資訊，請參閱 AWS Secrets Manager 使用者指南。

當您將資料庫秘密存放在 Secrets Manager 時， AWS 帳戶 會產生費用。如需關於定價的資訊，請參閱AWS Secrets Manager 定價。

當您執行下列其中一項操作時，您可以指定 Aurora，在 Secrets Manager 中管理 Amazon Aurora 資料庫叢集的主要使用者密碼：

- 建立資料庫叢集
- 修改資料庫叢集
- 從 Amazon S3 還原資料庫叢集 (僅限 Aurora MySQL)

當您指定 Aurora 在 Secrets Manager 中管理主要使用者密碼時，Aurora 會產生密碼並將其存放在 Secrets Manager 中。您可以直接與機密互動，以擷取主要使用者的憑證。您也可以指定客戶受管金鑰來加密機密，或使用 Secrets Manager 提供的 KMS 金鑰。

Aurora 會管理機密的設定，並依預設每七天輪換一次密碼。您可以修改某些設定，例如輪換排程。如果您刪除在 Secrets Manager 中管理密碼的資料庫叢集，秘密及其相關聯的中繼資料也會一併刪除。

若要使用機密中的憑證連線至資料庫叢集，您可以從 Secrets Manager 擷取機密。如需詳細資訊，請參閱AWS Secrets Manager 《 使用者指南》中的從 擷取秘密 AWS Secrets Manager，並使用 AWS Secrets Manager 秘密中的登入資料連線至 SQL 資料庫。

使用 Secrets Manager 管理主要使用者密碼的優點

使用 Secrets Manager 管理 Aurora 主要使用者密碼可提供下列優點：

- Aurora 會自動產生資料庫憑證。
- Aurora會自動在其中存放和管理資料庫登入資料 AWS Secrets Manager。
- Aurora 會定期輪換資料庫憑證，而無需變更應用程式。
- Secrets Manager 會保護資料庫憑證免於人類存取和純文字檢視。
- Secrets Manager 允許擷取機密中用於資料庫連線的資料庫憑證。
- Secrets Manager 允許使用 IAM 對機密中資料庫憑證的存取進行細微控制。

- 您可以選擇性地使用不同 KMS 金鑰將資料庫加密與憑證加密分開。
- 您可以消除資料庫憑證的手動管理和輪換。
- 您可以使用 AWS CloudTrail 和 Amazon CloudWatch 輕鬆監控資料庫登入資料。

如需 Secrets Manager 優點的詳細資訊，請參閱《AWS Secrets Manager 使用者指南》https://docs.aws.amazon.com/secretsmanager/latest/userguide/。

Secrets Manager 整合所需的許可

使用者必須具有必要的權限，才能執行與 Secrets Manager 整合相關的操作。您可以建立 IAM 政策，授與對其需要的指定資源執行特定 API 操作的許可。然後，您可以將這些政策連接至需要這些許可的 IAM 許可集或角色。如需詳細資訊，請參閱Amazon Aurora 的 Identity and access management。

對於建立、修改或還原操作，指定 Aurora 在 Secrets Manager 中管理主要使用者密碼的使用者必須具有執行下列操作的許可：

- `kms:DescribeKey`
- `secretsmanager:CreateSecret`
- `secretsmanager:TagResource`

需要 `kms:DescribeKey` 許可才能存取 `MasterUserSecretKmsKeyId` 的客戶受管金鑰，並描述 `aws/secretsmanager`。

對於建立、修改或還原操作，指定客戶受管金鑰以加密 Secrets Manager 中機密的使用者必須具有執行下列操作的許可：

- `kms:Decrypt`
- `kms:GenerateDataKey`
- `kms:CreateGrant`

對於修改操作，在 Secrets Manager 中輪換主要使用者密碼的使用者必須具有執行下列操作的許可：

- `secretsmanager:RotateSecret`

在 中強制執行主要使用者密碼的 Aurora管理 AWS Secrets Manager

您可以使用 IAM 條件金鑰，在 AWS Secrets Manager中強制執行主要使用者密碼的 Aurora 管理。除非主要使用者密碼是在 Secrets Manager 中由 Aurora 管理，否則下列政策不允許使用者建立或還原資料庫執行個體或資料庫叢集，。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": ["rds:CreateDBInstance", "rds:CreateDBCluster",
 "rds:RestoreDBInstanceFromS3", "rds:RestoreDBClusterFromS3"],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "rds:ManageMasterUserPassword": false
                }
            }
        }
    ]
}
```

> **Note**
>
> 此政策會在建立 AWS Secrets Manager 時強制執行 中的密碼管理。不過，您仍然可以停用 Secrets Manager 整合，並透過修改叢集個體手動設定主要密碼。
> 為了防止這種情況，請在政策的動作區塊中包含 `rds:ModifyDBInstance`、`rds:ModifyDBCluster`。請注意，這可防止使用者對未啟用 Secrets Manager 整合的現有叢集套用任何進一步的修改。

如需在 IAM 政策中使用條件金鑰的詳細資訊，請參閱 Aurora 的政策條件金鑰 和 範例政策：使用條件金鑰。

使用 Secrets Manager 管理資料庫叢集的主要使用者密碼

執行下列動作時，您可以在 Secrets Manager 中設定主要使用者密碼的 Aurora 管理：

- 建立 Amazon Aurora 資料庫叢集
- 修改 Amazon Aurora 資料庫叢集
- 從外部 MySQL 資料庫將資料遷移至 Amazon Aurora MySQL 資料庫叢集

您可以使用 RDS 主控台 AWS CLI、 或 RDS API 來執行這些動作。

主控台

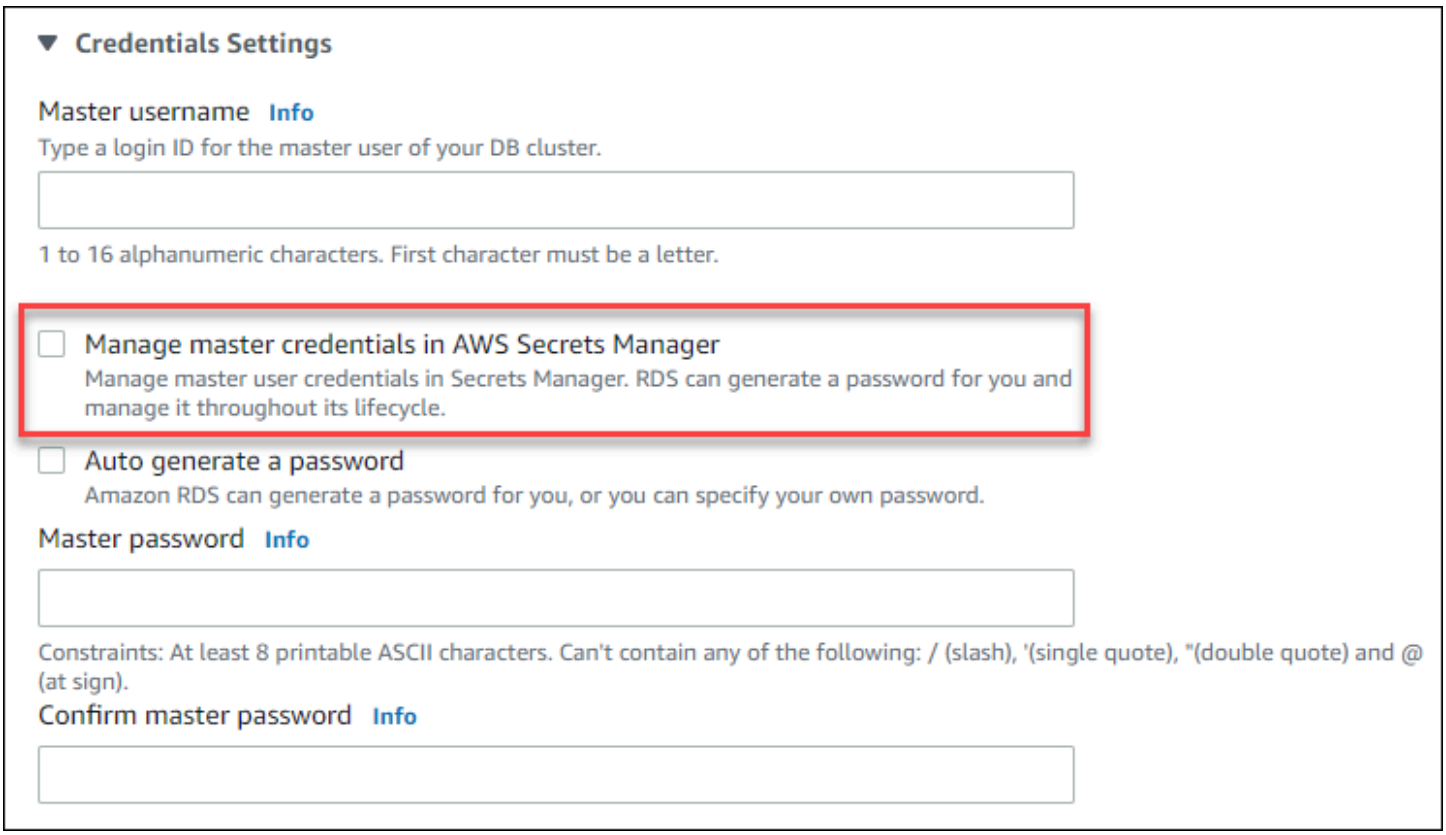
請遵循使用 RDS 主控台建立或修改資料庫叢集的指示：

- 建立資料庫叢集
- 修改資料庫叢集中的資料庫執行個體

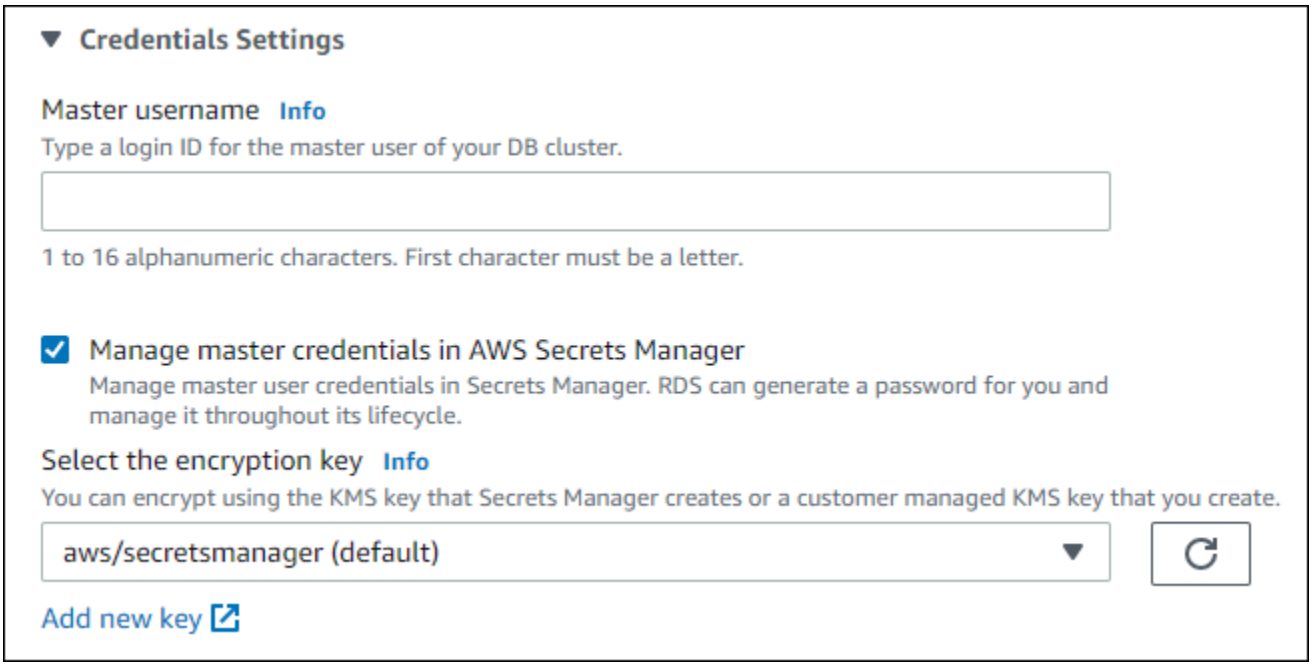
  在 RDS 主控台中，您可以修改任何資料庫執行個體，以指定整個資料庫叢集的主要使用者密碼管理設定。
- 從 Amazon S3 儲存貯體還原 Amazon Aurora MySQL 資料庫叢集

當使用 RDS 主控台來執行其中一項操作時，您可以指定主要使用者密碼是由 Secrets Manager 中的 Aurora 管理。若要在建立或還原資料庫叢集時這樣做，請在 Credential settings (憑證設定) 中選取 Manage master credentials in AWS Secrets Manager (在 AWS Secrets Manager中管理主要憑證)。修改資料庫叢集時，請在 Settings (設定) 中選取 Manage master credentials in AWS Secrets Manager (在 AWS Secrets Manager中管理主要憑證)。

以下影像是建立或還原資料庫叢集時，Manage master credentials in AWS Secrets Manager (在 AWS Secrets Manager中管理主要憑證) 設定的範例。



當您選取此選項時，Aurora 會產生主要使用者密碼，並在 Secrets Manager 中於整個生命週期管理該密碼。



您可以選擇使用 Secrets Manager 提供的 KMS 金鑰，或您建立的客戶受管金鑰來加密機密。在 Aurora 管理資料庫叢集的資料庫憑證之後，您就無法變更用來加密機密的 KMS 金鑰。

您可以選擇其他設定來符合您的需求。

如需當您建立資料庫叢集時可用設定的詳細資訊，請參閱 Aurora 資料庫叢集的設定。如需當您修改資料庫叢集時可用設定的詳細資訊，請參閱 Amazon Aurora的設定。

AWS CLI

若要指定 Aurora 在 Secrets Manager 中管理主要使用者密碼，請在下列其中一個命令中指定 `--manage-master-user-password` 選項：

- create-db-cluster
- modify-db-cluster
- restore-db-cluster-from-s3

當您在這些命令中指定 `--manage-master-user-password` 選項時，Aurora 會產生主要使用者密碼，並在 Secrets Manager 中於整個生命週期管理該密碼。

若要加密機密，您可以指定客戶受管金鑰或使用 Secrets Manager 提供的預設 KMS 金鑰。使用 `--master-user-secret-kms-key-id` 選項來指定客戶受管金鑰。 AWS KMS 金鑰識別符是 KMS 金鑰的金鑰 ARN、金鑰 ID、別名 ARN 或別名名稱。若要在不同的 中使用 KMS 金鑰 AWS 帳戶，請指定金鑰 ARN 或別名 ARN。在 Aurora 管理資料庫叢集的資料庫認證之後，您無法變更用來加密機密的 KMS 金鑰。

您可以選擇其他設定來符合您的需求。

如需當您建立資料庫叢集時可用設定的詳細資訊，請參閱 Aurora 資料庫叢集的設定。如需當您修改資料庫叢集時可用設定的詳細資訊，請參閱 Amazon Aurora的設定。

此範例會建立資料庫叢集，並指定 Aurora 在 Secrets Manager 中管理密碼。機密會使用 Secrets Manager 所提供的 KMS 金鑰進行加密。

Example

針對 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
     --db-cluster-identifier sample-cluster \
     --engine aurora-mysql \
     --engine-version 8.0 \
     --master-username admin \
```

```
    --manage-master-user-password
```

在 Windows 中：

```
aws rds create-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --engine aurora-mysql ^
    --engine-version 8.0 ^
    --master-username admin ^
    --manage-master-user-password
```

RDS API

若要指定 Aurora 在 Secrets Manager 中管理主要使用者密碼，請在下列其中一個操作中將 `ManageMasterUserPassword` 參數設為 `true`：

- CreateDBCluster
- ModifyDBCluster
- RestoreDBClusterFromS3

當您在其中一個操作中將 `ManageMasterUserPassword` 參數設為 `true` 時，Aurora 會產生主要使用者密碼，並在 Secrets Manager 中於整個生命週期管理該密碼。

若要加密機密，您可以指定客戶受管金鑰或使用 Secrets Manager 提供的預設 KMS 金鑰。使用 `MasterUserSecretKmsKeyId` 參數指定客戶受管金鑰。 AWS KMS 金鑰識別符是 KMS 金鑰的金鑰 ARN、金鑰 ID、別名 ARN 或別名名稱。若要在不同的 AWS 帳戶中使用 KMS 金鑰，請指定金鑰 ARN 或別名 ARN。在 Aurora 管理資料庫叢集的資料庫認證之後，您無法變更用來加密機密的 KMS 金鑰。

輪換資料庫叢集的主要使用者密碼秘密

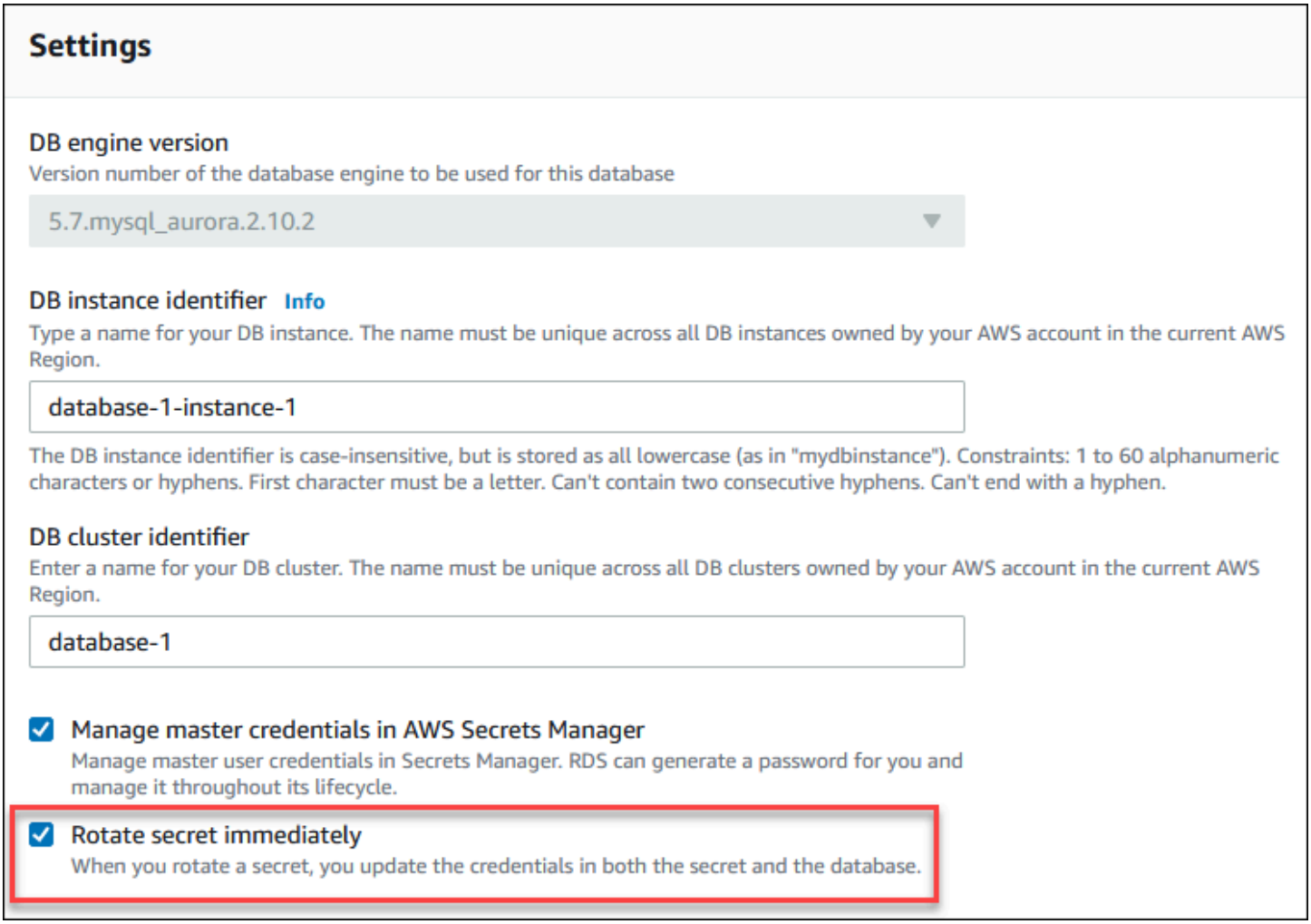
當 Aurora 輪換主要使用者密碼機密時，Secrets Manager 會針對現有機密產生新的機密版本。新版本的機密包含新的主要使用者密碼。Aurora 會變更資料庫叢集的主要使用者密碼，以符合新秘密版本的密碼。

您可以立即輪換機密，而無需等待排程的輪換。若要在 Secrets Manager 中輪換主要使用者密碼機密，請修改資料庫叢集。如需修改資料庫叢集的詳細資訊，請參閱 修改 Amazon Aurora 資料庫叢集。

您可以使用 RDS 主控台 AWS CLI、 或 RDS API 立即輪換主要使用者密碼秘密。新密碼長度一律為 28 個字元，且至少包含一個大寫和小寫字元、一個數字和一個標點符號。

主控台

若要使用 RDS 主控台輪換主要使用者密碼機密，請修改資料庫叢集，然後在 Settings (設定) 中選取 Rotate secret immediately (立即輪換機密)。



遵循使用 使用主控台、CLI 和 API 修改資料庫叢集 中的 RDS 主控台修改資料庫叢集的指示。您必須在確認頁面上選擇 Apply immediately (立即套用)。

AWS CLI

若要使用 輪換主要使用者密碼秘密 AWS CLI，請使用 modify-db-cluster 命令並指定 `--rotate-master-user-password`選項。您必須在輪換主要密碼時指定 `--apply-immediately` 選項。

此範例會輪換主要使用者密碼機密。

Example

針對 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --rotate-master-user-password \
    --apply-immediately
```

在 Windows 中：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --rotate-master-user-password ^
    --apply-immediately
```

RDS API

您可以使用 ModifyDBCluster 操作，並將 RotateMasterUserPassword 參數設為 true，來輪換主要使用者密碼機密。輪換主要密碼時，必須將 ApplyImmediately 參數設為 true。

檢視資料庫叢集秘密的詳細資訊

您可以使用 主控台 (https://console.aws.amazon.com/secretsmanager/：//) 或 AWS CLI (get-secret-value Secrets Manager 命令） 擷取秘密。

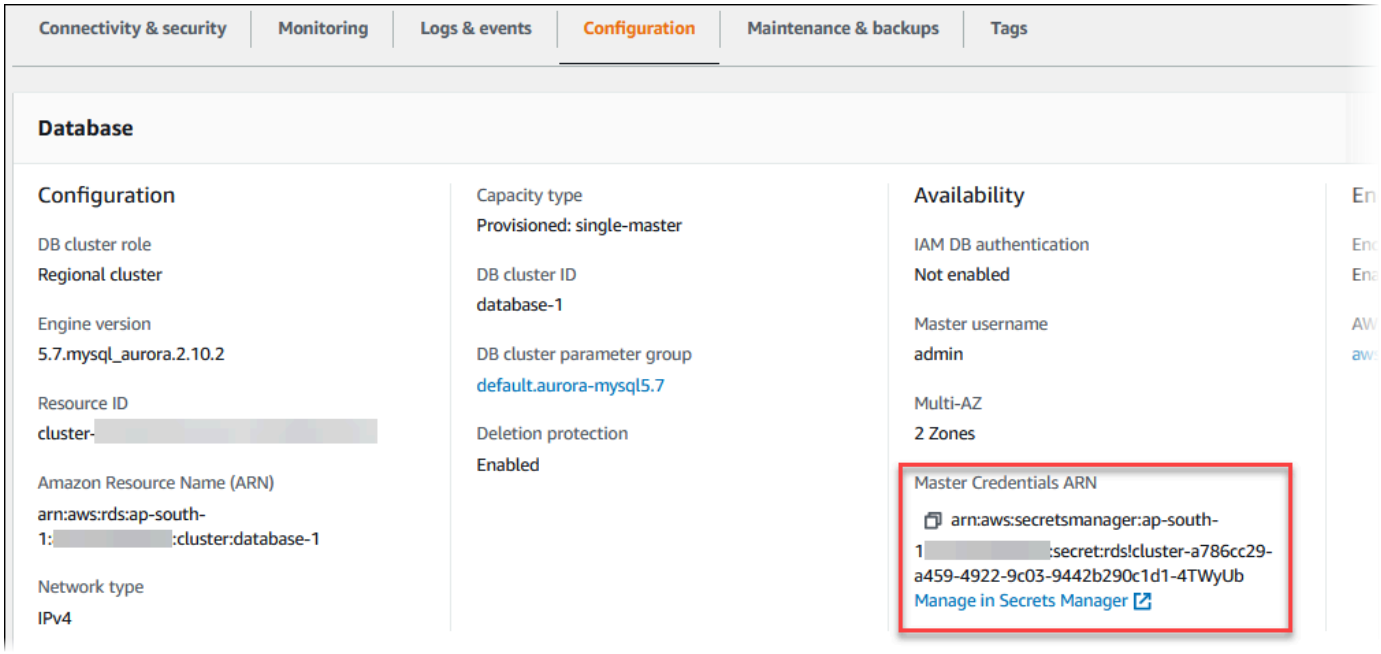
您可以使用 主控台 AWS CLI、 或 RDS API，在 Secrets Manager 中找到由 RDS Aurora 管理的秘密的 Amazon Resource Name (ARN)。

主控台

在 Secrets Manager 中檢視由 Aurora 管理之秘密的詳細資訊

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇資料庫叢集的名稱，以顯示其詳細資訊。
4. 選擇 Configuration (組態) 索引標籤。

   在 Master Credentials ARN (主要憑證 ARN) 中，您可以檢視機密 ARN。



您可以遵循 Manage in Secrets Manager (在 Secrets Manager 中管理) 連結，在 Secret Manager 主控台中檢視和管理機密。

AWS CLI

您可以使用 RDS AWS CLI describe-db-clusters 命令來尋找 Secrets Manager 中 Aurora 所管理秘密的下列相關資訊：

- `SecretArn` – 機密的 ARN
- `SecretStatus` – 機密的狀態

  可能的狀態值包括下列項目：

  - `creating` – 正在建立機密。
  - `active` – 機密可供正常使用和輪換。
  - `rotating` – 正在輪換機密。
  - `impaired` – 機密可以用來存取資料庫憑證，但無法將其輪換。例如，如果變更許可，以便 RDS 再也無法取機密或秘密的 KMS 金鑰，則秘密可能會具有此狀態。

    當密碼具有此狀態時，您可以更正導致狀態的條件。如果您更正了導致狀態的條件，則狀態仍會保留 `impaired`，直到下一次輪換為止。或者，您可以修改資料庫叢集，以關閉資料庫憑證的自動管理，然後再次修改資料庫叢集，以開啟資料庫憑證的自動管理。若要修改資料庫叢集，請在 modify-db-cluster 命令中使用 `--manage-master-user-password` 選項。

- `KmsKeyId` – 用來加密機密之 KMS 金鑰的 ARN

指定 `--db-cluster-identifier` 選項來顯示特定資料庫叢集的輸出。此範例顯示資料庫叢集所使用之機密的輸出。

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydbcluster
```

下列範例顯示機密的輸出：

```
"MasterUserSecret": {
                "SecretArn": "arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx",
                "SecretStatus": "active",
                "KmsKeyId": "arn:aws:kms:eu-
west-1:123456789012:key/0987dcba-09fe-87dc-65ba-ab0987654321"
            }
```

具有機密 ARN 時，您可以使用 get-secret-value Secrets Manager CLI 命令，檢視機密的詳細資訊。

此範例顯示前一個範例輸出中機密的詳細資訊。

Example

針對 Linux、macOS 或 Unix：

```
aws secretsmanager get-secret-value \
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

在 Windows 中：

```
aws secretsmanager get-secret-value ^
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

RDS API

您可以檢視 Secrets Manager 中由 Aurora 管理之機密的 ARN、狀態和 KMS 金鑰，方法為使用 DescribeDBClusters RDS 操作，並將 `DBClusterIdentifier` 參數設為資料庫叢集識別符。機密的詳細資訊包含在輸出中。

具有機密 ARN 時，您可以使用 gGetSecretValue Secrets Manager 操作，檢視機密的詳細資訊。

Amazon RDS 中的資料保護

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

基於資料保護目的，建議您使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 保護 AWS 帳戶 憑證，並設定個人使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用 AWS CloudTrail 設定 API 和使用者活動日誌記錄。如需使用 CloudTrail 追蹤擷取 AWS 活動的相關資訊，請參閱「AWS CloudTrail 使用者指南」中的使用 CloudTrail 追蹤。
- 使用 AWS 加密解決方案，以及 AWS 服務 內的所有預設安全控制項。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在透過命令列介面或 API 存取 AWS 時，需要 FIPS 140-3 驗證的加密模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-3。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用 Amazon RDS 或其他使用主控台、API、AWS CLI 或 AWS SDK 的 AWS 服務 時。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

主題

- 使用加密保護資料

- 網際網路流量隱私權

使用加密保護資料

Amazon Aurora 會在儲存層加密資料庫資源。您也可以加密資料庫叢集的連線。

主題

- 加密 Amazon Aurora 資源
- AWS KMS key 管理
- 使用 SSL/TLS 加密與資料庫叢集的連線
- 輪換您的 SSL/TLS 憑證

加密 Amazon Aurora 資源

無論在內部部署用戶端和 Amazon Aurora 之間移動，還是在 Amazon Aurora 和其他 AWS 資源之間移動，Amazon Aurora 都會保護您的靜態和傳輸中資料。Amazon Aurora 會加密 Amazon Aurora 資料庫叢集中的所有使用者資料，包括日誌、自動備份和快照。

資料加密後， Amazon Aurora 會以透明的方式處理資料的存取和解密身分驗證，且對效能的影響最小。您不需要修改資料庫用戶端應用程式即可使用加密。

> **Note**
>
> 對於加密和未加密的資料庫叢集，即使跨 AWS 區域複寫，來源和僅供讀取複本之間傳輸的資料也會加密。

主題

- Amazon Aurora 資源中的加密概觀
- 加密 Amazon Aurora 資料庫叢集
- 決定是否為資料庫叢集開啟加密
- Amazon Aurora 加密的可用性
- 傳輸中加密
- Amazon Aurora 加密資料庫叢集的限制

Amazon Aurora 資源中的加密概觀

Amazon Aurora 加密資料庫叢集可以保護您的資料，避免基礎儲存體受到未經授權人員的存取，為資料提供另一層保護。在 Amazon Aurora 中於 2026 年 2 月 18 或之後建立的所有新資料庫叢集，都會使用業界標準的 AES-256 加密進行靜態加密。此加密會在背景自動執行，保護您的資料，而不需要您採取任何動作。它還有助於減少保護敏感資料涉及的操作負擔和複雜性。透過靜態加密，您可以保護對合規敏感和安全至關重要的應用程式，防範意外和惡意威脅，同時符合法規要求。

Amazon Aurora 使用 AWS Key Management Service 金鑰來加密這些資源。 AWS KMS 結合安全、高可用性的硬體和軟體，以提供針對雲端擴展的金鑰管理系統。建立新的資料庫叢集時，Amazon Aurora 預設會使用 伺服器端加密 (SSE) 搭配 AWS擁有的金鑰。不過，您可以根據您的安全和合規需求，從三種加密類型中選擇：

- AWS 擁有的金鑰 (SSE-RDS) – 您無法檢視或管理的完全 AWS控制加密金鑰，由 Aurora 自動用於預設加密。
- AWS 受管金鑰 (AMK) – 此金鑰是由 建立和管理 AWS ，並且會顯示在您的帳戶中，但無法自訂。不收取月費，但需支付 AWS KMS API 費用。
- 客戶受管金鑰 (CMK) – 金鑰存放在您的帳戶中，由您建立、擁有和管理。您可以完全控制 KMS 金鑰（需支付AWS KMS 費用）。

AWS 受管金鑰是舊版加密選項，可保持回溯相容性。根據預設，Amazon Aurora 會使用擁有 AWS的金鑰來加密您的資料，提供強大的安全性保護，而不會產生額外費用或管理開銷。對於大多數使用案例，我們建議您使用預設 AWS擁有的金鑰來簡化和提高成本效益，如果您需要完全控制加密金鑰，則建議使用客戶受管金鑰 (CMK)。如需金鑰類型的詳細資訊，請參閱客戶受管金鑰和 AWS 受管金鑰。

> **Note**
>
> 重要：對於 2026 年 2 月 18 日之前建立的來源資料庫執行個體或叢集，如果您未選擇加入從這些來源建立的加密、快照、複製和 Amazon Aurora 複本（讀取執行個體），則不會加密。不過，Amazon Aurora 叢集之外的還原操作和邏輯複寫會產生加密的執行個體。

針對 Amazon Aurora 加密資料庫叢集，所有資料庫執行個體、日誌、備份和快照都會加密。如需加密可用性和限制的詳細資訊，請參閱 Amazon Aurora 加密的可用性 和 Amazon Aurora 加密資料庫叢集的限制。

當您建立加密的資料庫叢集時，您可以選擇客戶受管金鑰或 Amazon Aurora AWS 受管金鑰 的 來加密資料庫叢集，如果您未指定客戶受管金鑰的金鑰識別符，Amazon Aurora 會將 AWS 受管金鑰 用於新

的資料庫叢集。Amazon Aurora 會為 AWS 您的帳戶建立 AWS 受管金鑰 適用於 Amazon Aurora 的 。每個區域的 AWS 受管金鑰 Amazon Aurora AWS AWS 帳戶都有不同的 。

若要管理用於加密和解密 Amazon Aurora 資源的客戶受管金鑰，您可以使用 AWS Key Management Service (AWS KMS)。

使用 AWS KMS，您可以建立客戶受管金鑰並定義政策來控制這些客戶受管金鑰的使用。 AWS KMS 支援 CloudTrail，因此您可以稽核 KMS 金鑰用量，以確認客戶受管金鑰是否適當使用。您可以使用客戶受管金鑰搭配 Amazon Aurora 和支援 AWS 的服務，例如 Amazon S3、Amazon EBS 和 Amazon Redshift。如需與 整合的服務清單 AWS KMS，請參閱AWS 服務整合。使用 KMS 金鑰的一些考量：

- 建立加密的資料庫執行個體後，您就無法變更該執行個體所使用的 KMS 金鑰。在建立加密的資料庫執行個體之前，請務必判斷您的 KMS 金鑰需求。如果您需要變更資料庫叢集的加密金鑰，請遵循下列步驟：
  - 建立叢集的手動快照。
  - 在還原操作期間，還原快照並使用所需的 KMS 金鑰啟用加密。
- 如果您還原未加密快照並選擇不加密，則建立的資料庫叢集將使用預設靜態加密 （ 擁有AWS的金鑰 ） 進行加密。
- 您無法共用已使用共用快照 AWS 受管金鑰 之 AWS 帳戶的 加密的快照。
- 資料庫叢集中的每個資料庫執行個體都會共用使用相同 KMS 金鑰加密的相同儲存體。

> **Important**
>
> 當您停用 KMS 金鑰時，Amazon Aurora 可能會失去對資料庫叢集 KMS 金鑰的存取權。在這些情況下，加密的資料庫叢集會進入 `inaccessible-encryption-credentials-recoverable` 狀態。資料庫叢集會保持此狀態七天，在此期間執行個體會停止。在此期間對資料庫叢集發出的 API 呼叫可能無法成功。若要復原資料庫叢集，請啟用 KMS 金鑰並重新啟動此資料庫叢集。您可以從 AWS 管理主控台 AWS CLI、 或 RDS API 啟用 KMS 金鑰。使用 AWS CLI 命令 start-db-cluster 或 重新啟動資料庫叢集 AWS 管理主控台。
> `inaccessible-encryption-credentials-recoverable` 狀態僅適用於可停止的資料庫叢集。此可復原狀態不適用於無法停止的執行個體，例如具有跨區域讀取複本的叢集。如需詳細資訊，請參閱the section called “限制”。
> 如果資料庫叢集未在七天內復原，它會進入終端 `inaccessible-encryption-credentials` 狀態。在此狀態下，資料庫叢集不再可用，您只能從備份還原資料庫叢集。我們強烈建議您一律開啟備份，以防止資料庫中的資料遺失。

在建立資料庫叢集期間，Aurora 會檢查呼叫主體是否可以存取 KMS 金鑰，並從其在資料庫叢集的整個生命週期內使用的 KMS 金鑰產生授予。撤銷呼叫主體對 KMS 金鑰的存取權不會影響執行中的資料庫。在跨帳戶案例中使用 KMS 金鑰時，例如將快照複製到另一個帳戶，則需要與其他帳戶共用 KMS 金鑰。如果您從快照建立資料庫叢集，但未指定不同的 KMS 金鑰，則新叢集會使用來源帳戶的 KMS 金鑰。在您建立資料庫叢集之後撤銷對金鑰的存取權不會影響叢集。不過，停用金鑰會影響使用該金鑰進行加密的所有資料庫叢集。若要防止這種情況，請在快照複製操作期間指定不同的金鑰。

如需 KMS 金鑰的詳細資訊，請參閱《AWS Key Management Service 開發人員指南》和 AWS KMS key 管理 中的 AWS KMS keys。

加密 Amazon Aurora 資料庫叢集

根據預設，在 2026 年 2 月 18 日當天或之後建立的所有新資料庫叢集都會使用 AWS 擁有的金鑰加密。

若要使用 AWS 受管金鑰 或客戶受管金鑰加密新的資料庫叢集，請選擇 主控台上的 選項。如需建立資料庫叢集的詳細資訊，請參閱建立 Amazon Aurora 資料庫叢集。

如果您使用 create-db-cluster AWS CLI 命令來建立加密的資料庫叢集，請設定 `--storage-encrypted` 參數。若您使用 CreateDBCluster API 操作，請將 `StorageEncrypted` 參數設為 true。

一旦建立了加密的資料庫叢集之後，您就無法變更該資料庫叢集使用的 KMS 金鑰。因此，請務必在建立加密的資料庫叢集前，先決定您的 KMS 金鑰要求。

如果您使用 AWS CLI `create-db-cluster` 命令來建立具有客戶受管金鑰的加密資料庫叢集，請將 `--kms-key-id` 參數設定為 KMS 金鑰的任何金鑰識別符。如果您使用 Amazon RDS API `CreateDBInstance` 作業，請將 `KmsKeyId` 參數設定為 KMS 金鑰的任何金鑰識別碼。若要在不同帳戶中使用客戶受管金鑰 AWS ，請指定金鑰 ARN 或別名 ARN。

決定是否為資料庫叢集開啟加密

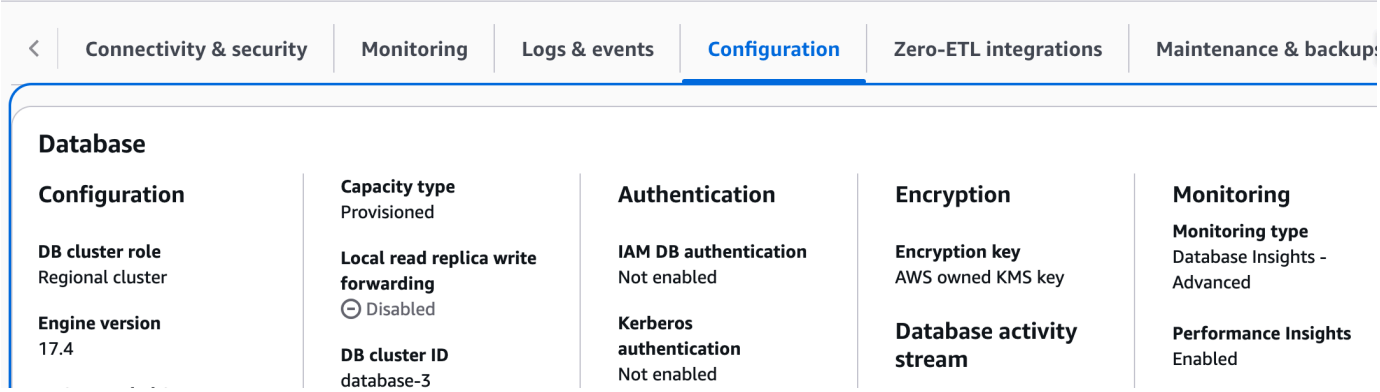
您可以使用 AWS 管理主控台 AWS CLI或 RDS API 來判斷資料庫叢集是否開啟靜態加密。

主控台

決定是否為資料庫叢集開啟靜態加密。

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/rds/ 的 Amazon RDS 主控台。

2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇您要檢查的資料庫叢集名稱，以檢視其詳細資訊。
4. 選擇 Configuration (組態) 索引標籤，然後勾選 Encryption (加密) 值。



AWS CLI

若要使用 判斷資料庫叢集是否開啟靜態加密 AWS CLI，請使用下列選項呼叫 describe-db-clusters 命令：

- `--db-cluster-identifier` – 資料庫叢集的名稱。

下列範例會使用查詢，為 mydb 資料庫叢集傳回關於靜態加密的 TRUE 或 FALSE。

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydb --query "*[].
{StorageEncrypted:StorageEncrypted}" --output text
```

RDS API

若要使用 Amazon RDS API 來決定是否為資料庫叢集開啟靜態加密，請呼叫 DescribeDBClusters 作業，搭配下列參數：

- `DBClusterIdentifier` – 資料庫叢集的名稱。

Amazon Aurora 加密的可用性

所有資料庫引擎和儲存體類型目前都可以使用 Amazon Aurora 加密。

> **Note**
>
> db.t2.micro 資料庫執行個體類別無法使用 Amazon Aurora 加密。

傳輸中加密

在實體層加密

在離開 AWS 安全設施之前， AWS 區域 透過 AWS 全球網路周遊的所有資料都會在實體層自動加密。AZ 之間的所有流量都會加密。額外的加密層 (包括本節所列的加密層) 可能會提供額外的保護。

Amazon VPC 對等互連和 Transit Gateway 跨區域對等互連提供的加密

所有使用 Amazon VPC 和 Transit Gateway 對等互連的跨區域流量都會在離開區域時自動進行批量加密。在離開 AWS 安全設施之前，會在實體層自動為所有流量提供另一層加密。

執行個體間的加密

AWS 在所有類型的資料庫執行個體之間提供安全和私有連線。此外，某些執行個體類型使用基礎 Nitro System 硬體的卸載功能，以自動加密執行個體之間的傳輸中流量。此加密機制使用帶有關聯資料的認證加密 (AEAD) 演算法 (採用 256 位元加密)。這對網路效能沒有影響。若要支援執行個體之間額外的傳輸中流量加密，必須符合下列要求：

- 執行個體使用下列執行個體類型：
  - 一般用途：M6i、M6id、M6in、M6idn、M7g
  - 記憶體最佳化：R6i、R6id、R6in、R6idn、R7g、X2idn、X2iedn、X2iezn
- 這些執行個體位於相同的 AWS 區域中。
- 這些執行個體位於相同的 VPC 或對等 VPC 中，且流量不會經過虛擬網路裝置或服務，例如負載平衡器或傳輸閘道。

Amazon Aurora 加密資料庫叢集的限制

Amazon Aurora 加密資料庫叢集具有下列限制：

- 您無法在加密的資料庫叢集上關閉加密。
- 如果您有現有的未加密叢集，從該叢集建立的所有快照也會未加密。若要從未加密的叢集建立加密快照，您必須在複製操作期間複製快照並指定客戶受管金鑰。如果未指定客戶受管金鑰，您無法從未加密的快照建立加密的快照。

- 您無法建立未加密資料庫的加密快照。
- 加密資料庫叢集的快照必須使用與資料庫叢集相同的 KMS 金鑰進行加密。
- 您無法將未加密的資料庫叢集轉換為已加密的叢集。但是，您可以將未加密的快照還原至加密的 Aurora 資料庫叢集。若要執行此作業，請在從未加密的快照還原時指定 KMS 金鑰。
- 如果您有現有的未加密叢集，從該叢集建立的任何 Amazon Aurora 複本（讀取執行個體）也會未加密。若要從未加密的叢集建立加密的叢集，您需要還原資料庫叢集。還原的叢集預設會在還原操作後加密。
- 若要將加密快照從一個 AWS 區域複製到另一個 AWS 區域，您必須在目的地區域中指定 KMS 金鑰。這是因為 KMS 金鑰專屬於其建立所在的 AWS 區域。

  在整個複製過程中來源快照仍會保持加密狀態。Amazon Aurora 會在複製過程中使用信封加密來保護資料。如需信封加密的詳細資訊，請參閱 AWS Key Management Service 開發人員指南中的信封加密。
- 您無法解密加密的資料庫叢集。但是，您可以從加密的資料庫叢集 匯出資料，然後將資料匯入未加密的資料庫叢集。

AWS KMS key 管理

Amazon Aurora 會自動與 AWS Key Management Service (AWS KMS) 整合以進行金鑰管理。Amazon Aurora 會使用信封加密。如需信封加密的詳細資訊，請參閱 AWS Key Management Service 開發人員指南中的信封加密。

您可以使用兩種類型的 AWS KMS 金鑰來加密資料庫叢集。

- 如果想要完全控制 KMS 金鑰，您必須建立客戶受管金鑰。如需客戶受管金鑰的詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的客戶受管金鑰。
- AWS 受管金鑰 是帳戶中的 KMS 金鑰，由與 整合的 AWS 服務代表您建立、管理和使用 AWS KMS。根據預設，RDS AWS 受管金鑰 (`aws/rds`) 用於加密。您無法管理、輪換或刪除 RDS AWS 受管金鑰。如需 的詳細資訊 AWS 受管金鑰，請參閱《 AWS Key Management Service 開發人員指南AWS 受管金鑰》中的 。

若要管理用於 Amazon Aurora 加密資料庫叢集的 KMS 金鑰，請在 AWS KMS 主控台、 AWS CLI或 AWS KMS API 中使用 AWS Key Management Service (AWS KMS)。若要檢視透過 AWS 受管或客戶受管金鑰採取的每項動作稽核日誌，請使用 AWS CloudTrail。如需有關金鑰轉換的詳細資訊，請參閱轉換 AWS KMS 金鑰。

授權使用客戶受管金鑰

當 Aurora 在密碼編譯操作中使用客戶受管金鑰時，它會代表建立或變更 Aurora 資源的使用者。

若要使用客戶受管金鑰建立 Aurora 資源，使用者必須具備呼叫客戶受管金鑰下列作業的許可：

- `kms:CreateGrant`
- `kms:DescribeKey`

您可以在金鑰政策或在 IAM 政策中指定這些必要的許可 (如果金鑰政策允許)。

> **Important**
>
> 當您使用 Amazon RDS 等受管服務之 AWS KMS 金鑰政策中所有資源 (*) 的明確拒絕陳述式時，您必須指定允許資源擁有帳戶的條件。如果沒有此條件，操作可能會失敗，即使拒絕規則包含 IAM 使用者的例外狀況。

> **Tip**
>
> 若要遵循最低權限原則人，請勿允許 `kms:CreateGrant` 的完整存取。反之，使用 kms:ViaService 條件金鑰，只允許使用者在由 AWS 服務代使用者建立授予時，才在 KMS 金鑰上建立授予。

您可以透過各種方式使 IAM 政策更加嚴格。例如，若要允許客戶受管金鑰僅用於源自 Aurora 的請求，您可以搭配 `rds.<region>.amazonaws.com` 值使用 kms:ViaService 條件金鑰。您也可以使用 Amazon RDS 加密內容 中的金鑰或值作為條件，以便將客戶受管金鑰用於加密。

如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的允許其他帳戶使用者使用 KMS 金鑰和 AWS KMS中的金鑰政策。

Amazon RDS 加密內容

當 Aurora 使用您的 KMS 金鑰時，或者當 Amazon EBS 代表 Aurora 使用 KMS 金鑰時，服務會指定加密內容。加密內容是額外的驗證資料 (AAD)， AWS KMS 用於確保資料完整性。為加密操作指定加密內容時，此服務必須為解密操作指定相同的加密內容。否則，解密會失敗。加密內容也會寫入 AWS CloudTrail 日誌檔，以協助您了解為何使用指定的 KMS 金鑰。您的 CloudTrail 日誌可能包含多

個項目來描述 KMS 金鑰使用情形，但每個日誌項目中的加密內容可協助您判斷該特定使用情形的原因。

至少， Aurora 一律使用資料庫叢集 ID 做為加密內容，如下列 JSON 格式範例：

```
{ "aws:rds:dbc-id": "cluster-CQYSMDPBRZ7BPMH7Y3RTDG5QY" }
```

這個加密內容可協助您識別您的 KMS 金鑰所用的資料庫叢集。

當您的 KMS 金鑰用於特定的資料庫叢集和特定的 Amazon EBS 磁碟區，資料庫叢集 ID 和 Amazon EBS 磁碟區 ID 都會用於加密內容，如下列 JSON 格式範例：

```
{
  "aws:rds:dbc-id": "cluster-BRG7VYS3SVIFQW7234EJQOM5RQ",
  "aws:ebs:id": "vol-ad8c6542"
}
```

使用 SSL/TLS 加密與資料庫叢集的連線

您可以從應用程式使用 Secure Socket Layer (SSL) 或 Transport Layer Security (TLS) 來加密對執行 Aurora MySQL 或 Aurora PostgreSQL 之資料庫叢集的連線。

SSL/TLS 連線會對用戶端與資料庫叢集之間移動的資料進行加密，藉此提供多一層安全性。或者，您的 SSL/TLS 連線可以藉由驗證安裝在資料庫上的伺服器憑證來執行伺服器身分驗證。若需要伺服器身分驗證，請依照此一般程序進行：

1. 為您的資料庫選擇簽署資料庫伺服器憑證的憑證認證機構 (CA)。如需有關憑證認證機構的詳細資訊，請參閱 憑證授權單位。
2. 下載憑證套件，以便在連線至資料庫時使用。若要下載憑證套件，請參閱 憑證套件依據 AWS 區域。

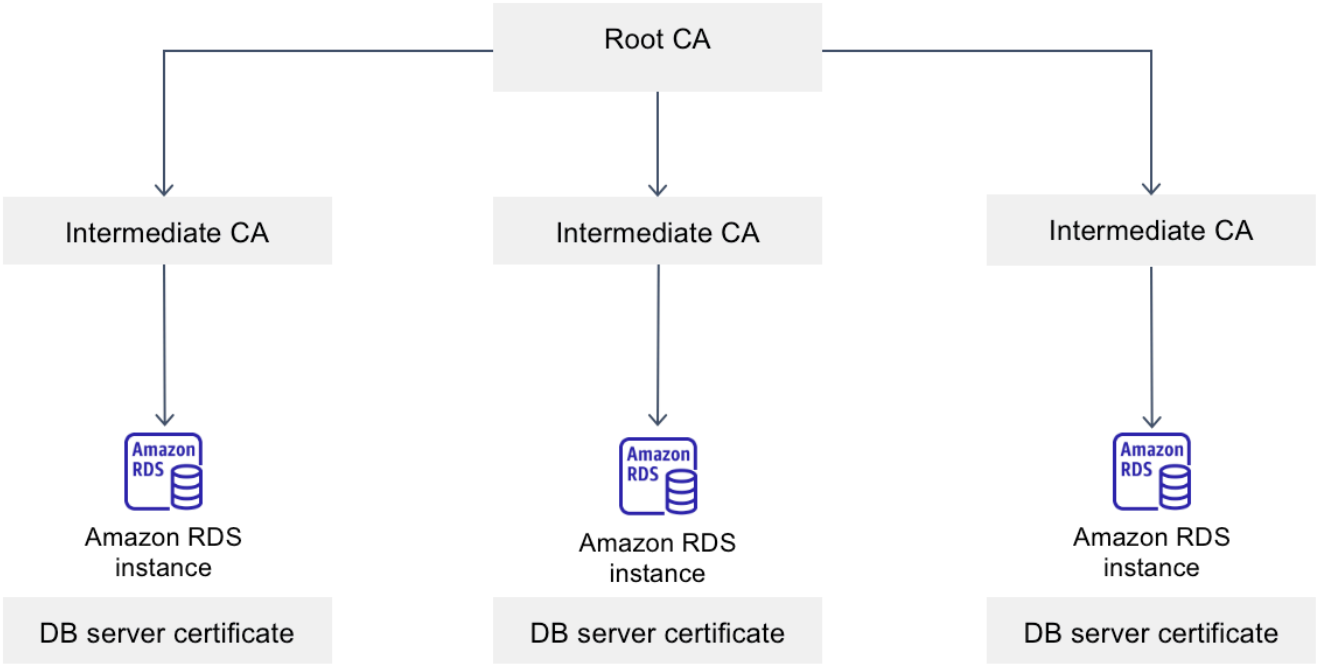
   > **Note**
   >
   > 所有憑證都只能使用 SSL/TLS 連線來下載。

3. 使用資料庫引擎實作 SSL/TLS 連線的程序連線至資料庫。每個資料庫引擎都有自己實作 SSL/TLS 的程序。若要了解如何為您的資料庫實作 SSL/TLS，請瀏覽對應您的資料庫引擎的連結：
   - Amazon Aurora MySQL 的安全性

- Amazon Aurora PostgreSQL 的安全性

憑證授權單位

憑證認證機構 (CA) 是在憑證鏈頂端識別根 CA 的憑證。CA 會簽署安裝在每個資料庫執行個體上的資料庫執行個體憑證。資料庫伺服器憑證會將資料庫執行個體識別為受信任伺服器。



Amazon RDS 提供下列 CA 來簽署資料庫的資料庫伺服器憑證。

憑證授權單位 (CA)	Description	Common name (CN) (通用名稱 (CN))
rds-ca-rsa2048-g1	在大多數 AWS 區域中，使用搭配 RSA 2048 私密金鑰演算法和 SHA256 簽署演算法的憑證認證機構。 在 中 AWS GovCloud (US) Regions，此 CA 使用憑證授權機構搭配 RSA 2048 私有金鑰演算法和 SHA384 簽署演算法。	Amazon RDS ######根 CA RSA2048 G1

憑證授權單位 (CA)	Description	Common name (CN) (通用名稱 (CN))
	此 CA 支援自動伺服器憑證輪換。	
rds-ca-rsa4096-g1	使用搭配 RSA 4096 私密金鑰演算法和 SHA384 簽署演算法的憑證認證機構。此 CA 支援自動伺服器憑證輪換。	Amazon RDS ######根 CA RSA4096 G1
rds-ca-ecc384-g1	使用搭配 ECC 384 私密金鑰演算法和 SHA384 簽署演算法的憑證認證機構。此 CA 支援自動伺服器憑證輪換。	Amazon RDS ######根 CA ECC384 G1

> **Note**
>
> 如果您使用的是 AWS CLI，您可以使用 describe-certificates 來查看上述憑證授權單位的有效性。

這些 CA 憑證包含在區域和全域憑證套件中。當您搭配資料庫使用 rds-ca-rsa2048-g1、rds-ca-rsa4096-g1 或 rds-ca-ecc384-g1 CA 時，RDS 會管理資料庫上的資料庫伺服器憑證。RDS 會在資料庫伺服器憑證到期之前自動進行輪換。

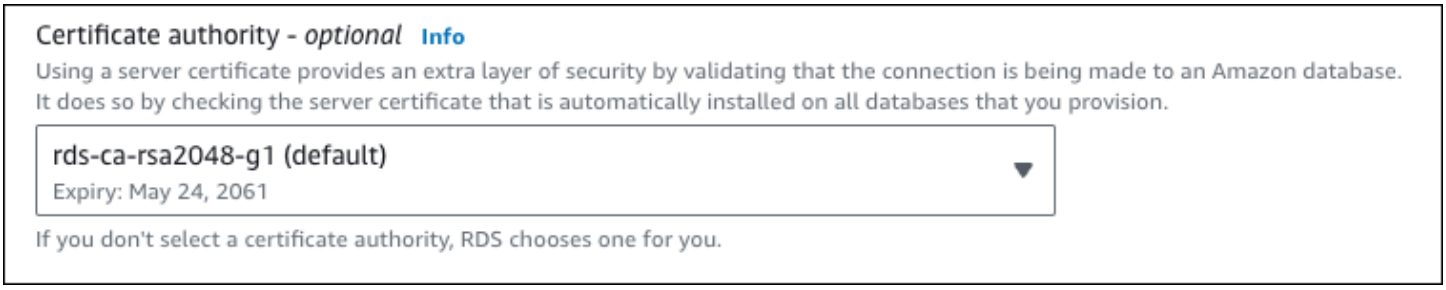
設定資料庫的 CA

執行下列任務時，您可以設定資料庫的 CA：

- 建立 Aurora 資料庫叢集 – 當您使用 AWS CLI 或 RDS API 在資料庫叢集中建立第一個資料庫執行個體時，可以為 Aurora 叢集中的資料庫執行個體設定 CA。當使用 RDS 主控台建立資料庫叢集時，您無法針對資料庫叢集的資料庫執行個體設定 CA。如需說明，請參閱 建立 Amazon Aurora 資料庫叢集。
- 修改資料庫執行個體 - 您可以修改資料庫執行個體，針對資料庫叢集的資料庫執行個體設定 CA。如需說明，請參閱 修改資料庫叢集中的資料庫執行個體。

> **Note**
>
> 預設 CA 設定為 rds-ca-rsa2048-g1。您可以使用 AWS 帳戶 modify-certificates 命令覆寫 的預設 CA。

可用的 CA 取決於資料庫引擎和資料庫引擎版本。使用 時 AWS 管理主控台，您可以使用憑證授權單位設定選擇 CA，如下圖所示。



主控台只會顯示資料庫引擎和資料庫引擎版本可用的 CA。如果您使用的是 AWS CLI，則可以使用 create-db-instance 或 modify-db-instance 命令來設定資料庫執行個體的 CA。

如果您使用的是 AWS CLI，您可以使用 describe-certificates 命令查看帳戶的可用 CAs。此命令也會在輸出中顯示 `ValidTill` 中每個 CA 的到期日。您可以使用 describe-db-engine-versions 命令，尋找適用於特定資料庫引擎和資料庫引擎版本的 CA。

下列範例顯示適用於預設 RDS for PostgreSQL 資料庫引擎版本的 CA。

```
aws rds describe-db-engine-versions --default-only --engine postgres
```

輸出類似如下。可用的 CA 列示在 `SupportedCACertificateIdentifiers` 中。輸出也會顯示 DB 引擎版本是否支援輪換憑證，而不需在 `SupportsCertificateRotationWithoutRestart` 中重新啟動。

```
{
    "DBEngineVersions": [
        {
            "Engine": "postgres",
            "MajorEngineVersion": "13",
            "EngineVersion": "13.4",
            "DBParameterGroupFamily": "postgres13",
            "DBEngineDescription": "PostgreSQL",
            "DBEngineVersionDescription": "PostgreSQL 13.4-R1",
```

```
            "ValidUpgradeTarget": [],
            "SupportsLogExportsToCloudwatchLogs": false,
            "SupportsReadReplica": true,
            "SupportedFeatureNames": [
                "Lambda"
            ],
            "Status": "available",
            "SupportsParallelQuery": false,
            "SupportsGlobalDatabases": false,
            "SupportsBabelfish": false,
            "SupportsCertificateRotationWithoutRestart": true,
            "SupportedCACertificateIdentifiers": [
                "rds-ca-rsa2048-g1",
                "rds-ca-ecc384-g1",
                "rds-ca-rsa4096-g1"
            ]
        }
    ]
}
```

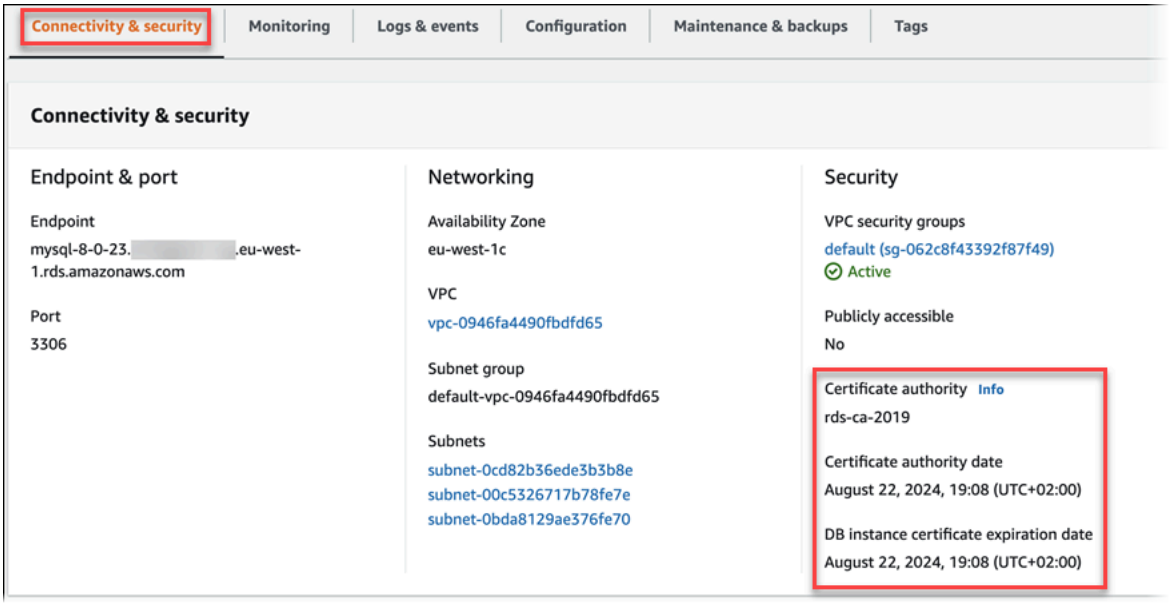
資料庫伺服器憑證有效期

資料庫伺服器憑證的有效期取決於資料庫引擎和資料庫引擎版本。如果資料庫引擎版本支援在不重新啟動的情況下輪換憑證，則資料庫伺服器憑證的有效期為 1 年。若不支援的話，則有效期為 3 年。

如需資料庫伺服器憑證輪換的詳細資訊，請參閱 自動伺服器憑證輪換。

檢視資料庫執行個體的 CA

您可以檢視主控台中的連線和安全索引標籤，來檢視資料庫 CA 的詳細資訊，如下圖所示。



如果您使用的是 AWS CLI，您可以使用 describe-db-instances 命令來檢視資料庫執行個體的 CA 詳細資訊。

下載適用於 Aurora 的憑證套件

當您使用 SSL 或 TLS 連線到資料庫時，資料庫執行個體需要來自 Amazon RDS 的信任憑證。在下表中選取適當的連結，以下載與您託管資料庫所在的 AWS 區域 對應的套件。

憑證套件依據 AWS 區域

所有 AWS 區域 和 GovCloud (US) 區域的憑證套件都包含下列根 CA 憑證：

- `rds-ca-rsa2048-g1`
- `rds-ca-rsa4096-g1`
- `rds-ca-ecc384-g1`

`rds-ca-rsa4096-g1` 和 `rds-ca-ecc384-g1` 憑證不適用於下列區域：

- 亞太地區 (孟買)
- 亞太地區 (墨爾本)
- 加拿大西部 (卡加利)
- 歐洲 (蘇黎世)
- 歐洲 (西班牙)
- 以色列 (特拉維夫)

您的應用程式信任存放區只需要註冊根 CA 憑證。請勿將中繼 CA 憑證註冊到您的信任存放區，因為這可能會在 RDS 自動輪換資料庫伺服器憑證時造成連線問題。

> **Note**
>
> Amazon RDS Proxy 和 Aurora Serverless v1 使用 來自 AWS Certificate Manager (ACM) 的憑證。如果您是使用 RDS Proxy，您不需要下載 Amazon RDS 憑證或更新使用 RDS Proxy 連線的應用程式。如需詳細資訊，請參閱 搭配 RDS Proxy 使用 TLS/SSL。
> 如果您使用 Aurora Serverless v1，則不需要下載 Amazon RDS 憑證。如需詳細資訊，請參閱 搭配 Aurora Serverless v1 使用 TLS/SSL。

若要下載 的憑證套件 AWS 區域，請在下表中選取 AWS 區域 託管資料庫的 連結。

AWS 區域	憑證套件 (PEM)	憑證套件 (PKCS7)
任何商業 AWS 區域	global-bundle.pem	global-bundle.p7b
美國東部 (維吉尼亞北部)	us-east-1-bundle.pem	us-east-1-bundle.p7b
美國東部 (俄亥俄)	us-east-2-bundle.pem	us-east-2-bundle.p7b
美國西部 (加利佛尼亞北部)	us-west-1-bundle.pem	us-west-1-bundle.p7b
美國西部 (奧勒岡)	us-west-2-bundle.pem	us-west-2-bundle.p7b
非洲 (開普敦)	af-south-1-bundle.pem	af-south-1-bundle.p7b
亞太區域 (香港)	ap-east-1-bundle.pem	ap-east-1-bundle.p7b
亞太地區 (海德拉巴)	ap-south-2-bundle.pem	ap-south-2-bundle.p7b
亞太地區 (雅加達)	ap-southeast-3-bundle.pem	ap-southeast-3-bundle.p7b
亞太地區 (馬來西亞)	ap-southeast-5-bundle.pem	ap-southeast-5-bundle.p7b
亞太地區 (墨爾本)	ap-southeast-4-bundle.pem	ap-southeast-4-bundle.p7b
亞太地區 (孟買)	ap-south-1-bundle.pem	ap-south-1-bundle.p7b
亞太區域 (大阪)	ap-northeast-3-bundle.pem	ap-northeast-3-bundle.p7b

AWS 區域	憑證套件 (PEM)	憑證套件 (PKCS7)
亞太區域 (泰國)	ap-southeast-7-bundle.pem	ap-southeast-7-bundle.p7b
亞太地區 (東京)	ap-northeast-1-bundle.pem	ap-northeast-1-bundle.p7b
亞太區域 (首爾)	ap-northeast-2-bundle.pem	ap-northeast-2-bundle.p7b
亞太區域 (新加坡)	ap-southeast-1-bundle.pem	ap-southeast-1-bundle.p7b
亞太區域 (雪梨)	ap-southeast-2-bundle.pem	ap-southeast-2-bundle.p7b
加拿大 (中部)	ca-central-1-bundle.pem	ca-central-1-bundle.p7b
加拿大西部 (卡加利)	ca-west-1-bundle.pem	ca-west-1-bundle.p7b
歐洲 (法蘭克福)	eu-central-1-bundle.pem	eu-central-1-bundle.p7b
歐洲 (愛爾蘭)	eu-west-1-bundle.pem	eu-west-1-bundle.p7b
歐洲 (倫敦)	eu-west-2-bundle.pem	eu-west-2-bundle.p7b
歐洲 (米蘭)	eu-south-1-bundle.pem	eu-south-1-bundle.p7b
歐洲 (巴黎)	eu-west-3-bundle.pem	eu-west-3-bundle.p7b
歐洲 (西班牙)	eu-south-2-bundle.pem	eu-south-2-bundle.p7b
歐洲 (斯德哥爾摩)	eu-north-1-bundle.pem	eu-north-1-bundle.p7b
歐洲 (蘇黎世)	eu-central-2-bundle.pem	eu-central-2-bundle.p7b
以色列 (特拉維夫)	il-central-1-bundle.pem	il-central-1-bundle.p7b
墨西哥 (中部)	mx-central-1-bundle.pem	mx-central-1-bundle.p7b
Middle East (Bahrain)	me-south-1-bundle.pem	me-south-1-bundle.p7b
中東 (阿拉伯聯合大公國)	me-central-1-bundle.pem	me-central-1-bundle.p7b
南美洲 (聖保羅)	sa-east-1-bundle.pem	sa-east-1-bundle.p7b

AWS 區域	憑證套件 (PEM)	憑證套件 (PKCS7)
任何 AWS GovCloud (US) Region	global-bundle.pem	global-bundle.p7b
AWS GovCloud （美國東部）	us-gov-east-1-bundle.pem	us-gov-east-1-bundle.p7b
AWS GovCloud （美國西部）	us-gov-west-1-bundle.pem	us-gov-west-1-bundle.p7b

檢視 CA 憑證的內容

若要查看 CA 憑證套件的內容，請使用下列命令：

```
keytool -printcert -v -file global-bundle.pem
```

輪換您的 SSL/TLS 憑證

Amazon RDS 憑證授權機構憑證 rds-ca-2019 已於 2024 年 8 月到期。如果您使用或計劃使用 Secure Sockets Layer (SSL) 或 Transport Layer Security (TLS) 搭配憑證驗證連線至 RDS 資料庫執行個體或，請考慮使用新的 CA 憑證 rds-ca-rsa2048-g1、rds-ca-rsa4096-g1 或 rds-ca-ecc384-g1。如果您目前未將 SSL/TLS 與憑證驗證搭配使用，則可能仍有過期的 CA 憑證，而且如果您計劃將 SSL/TLS 與憑證驗證搭配使用以連線至 RDS 資料庫，則必須將其更新為新的 CA 憑證。

Amazon RDS 提供新的 CA 憑證做為 AWS 安全最佳實務。如需新憑證和支援 AWS 區域的相關資訊，請參閱 使用 SSL/TLS 加密與資料庫叢集的連線 。

若要更新資料庫的 CA 憑證，請使用下列方法：

- 透過修改資料庫執行個體來更新憑證授權機構憑證
- 透過套用維護來更新憑證授權機構憑證

在更新您的資料庫執行個體以使用新的 CA 憑證之前，請確定您已更新連線至 RDS 資料庫的用戶端或應用程式。

輪換憑證的考量

在輪換憑證之前，請考量下列情況：

- Amazon RDS Proxy 和 Aurora Serverless v1 使用來自 AWS Certificate Manager (ACM) 的憑證。如果您使用 RDS Proxy，當您輪換 SSL/TLS 憑證時，不需要更新使用 RDS Proxy 連線的應用程式。如需詳細資訊，請參閱 搭配 RDS Proxy 使用 TLS/SSL。
- 如果您使用 Aurora Serverless v1，則不需要下載 Amazon RDS 憑證。如需詳細資訊，請參閱 搭配 Aurora Serverless v1 使用 TLS/SSL。
- 如果您使用的 Go 1.15 版應用程式是搭配在 2020 年 7 月 28 日前建立或更新為 rds-ca-2019 憑證的資料庫執行個體，則您必須再次更新憑證。根據您的引擎，將憑證更新為 rds-ca-rsa2048-g1、rds-ca-rsa4096-g1 或 rds-ca-ecc384-g1。

  使用新的 CA 憑證識別符，將 `modify-db-instance` 命令。您可以使用 `describe-db-engine-versions` 命令，尋找適用於特定資料庫引擎和資料庫引擎版本的 CA。

  如果您是在 2020 年 7 月 28 日之後建立資料庫或更新其憑證，則不需要執行任何動作。如需詳細資訊，請參閱 Go GitHub 問題編號 39568。

透過修改資料庫執行個體來更新憑證授權機構憑證

以下範例會將您的 CA 憑證從 rds-ca-2019 更新為 rds-ca-rsa2048-g1。您可以選擇不同的憑證。如需詳細資訊，請參閱 憑證授權單位。

更新您的應用程式信任存放區，以減少與更新 CA 憑證相關聯的任何停機時間。如需與 CA 憑證輪換相關聯重新啟動的詳細資訊，請參閱 自動伺服器憑證輪換。

透過修改資料庫執行個體來更新 CA 憑證

1. 請遵循 使用 SSL/TLS 加密與資料庫叢集的連線 所述，下載新的 SSL/TLS 憑證。
2. 更新您的應用程式，以使用新的 SSL/TLS 憑證。

   更新應用程式 SSL/TLS 憑證的方法取決於您特定的應用程式。與應用程式開發人員合作更新應用程式的 SSL/TLS 憑證。

   如需針對每個資料庫引擎檢查 SSL/TLS 連線和更新應用程式的資訊，請參閱下列主題：

   - 將應用程式更新為使用新的 TLS 憑證來連線至 Aurora MySQL 資料庫叢集
   - 將應用程式更新為使用新的 SSL/TLS 憑證來連線至 Aurora PostgreSQL 資料庫叢集

   關於為 Linux 作業系統更新信任存放區的範例指令碼，請參閱 將憑證匯入信任存放區的範例指令碼。

> **Note**
>
> 憑證套件同時包含舊 CA 和新 CA 的憑證，讓您可以安全地更新應用程式，並在轉換期間維護連線。如果您使用 AWS Database Migration Service 將資料庫遷移至資料庫叢集，我們建議您使用憑證套件來確保遷移期間的連線。

3. 修改資料庫執行個體或，以將 CA 從 rds-ca-2019 變更為 rds-ca-rsa2048-g1。若要檢查資料庫是否需要重新啟動才能更新 CA 憑證，請使用 describe-db-engine-versions 命令並查看 `SupportsCertificateRotationWithoutRestart` 旗標。

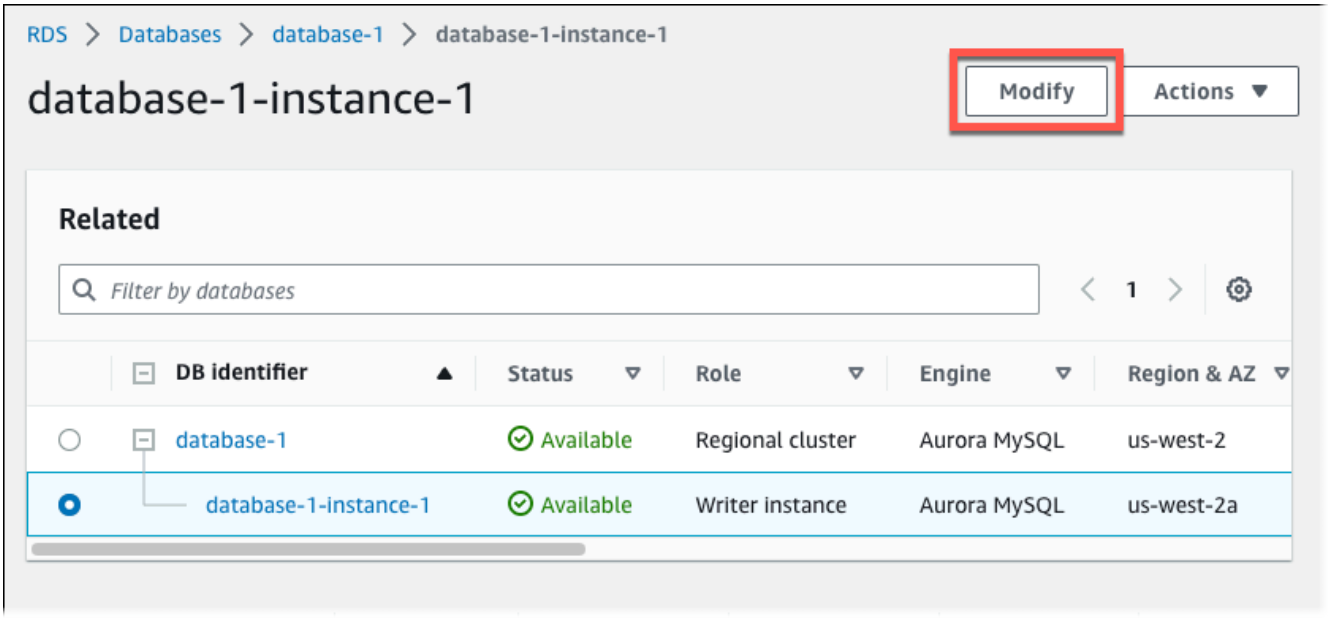
> **Note**
>
> 修改後將 Babelfish 叢集重新開機以更新 CA 憑證。

> **Important**
>
> 如果您在憑證到期後發生連線問題，請在主控台中指定 Apply immediately (立即套用)，或使用 `--apply-immediately` 來指定 AWS CLI選項。依預設，此操作排定在下一個維護時段執行。
> 若要為您的叢集 CA 設定與預測 RDS CA 不同的覆寫，請使用 modify-certificates CLI 命令。

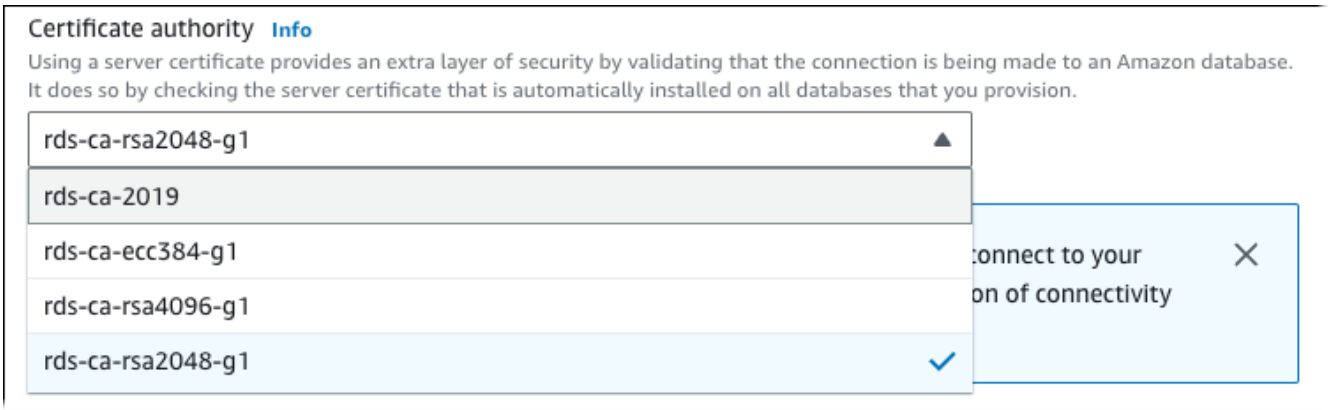
您可以使用 AWS 管理主控台 或 AWS CLI ，將資料庫執行個體的 CA 憑證從 rds-ca-2019 變更為 rds-ca-rsa2048-g1。

Console

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ : // 開啟 Amazon RDS 主控台。
2. 在導覽窗格中選擇資料庫，然後選擇您要修改的資料庫執行個體。
3. 選擇 Modify (修改)。



4. 在連線區段中，選擇 rds-ca-rsa2048-g1。



5. 選擇 Continue (繼續)，並檢查修改的摘要。
6. 若要立即套用變更，請選擇 Apply immediately (立即套用)。
7. 在確認頁面上，檢閱您的變更。如果都正確，請選擇修改資料庫執行個體，以儲存您的變更。

> **Important**
>
> 排定此操作時，請確定您事先已更新用戶端信任存放區。

或者，選擇 Back (上一步) 以編輯變更，或是選擇 Cancel (取消) 以取消變更。

AWS CLI

若要使用 AWS CLI 將資料庫執行個體的 CA 從 rds-ca-2019 變更為 rds-ca-rsa2048-g1，請呼叫 modify-db-instance 或 modify-db-cluster 命令。指定資料庫執行個體識別符和 `--ca-certificate-identifier` 選項。

使用 `--apply-immediately` 參數以立即套用更新。依預設，此操作排定在下一個維護時段執行。

> **Important**
>
> 排定此操作時，請確定您事先已更新用戶端信任存放區。

Example

以下範例會將 CA 憑證設定為 `rds-ca-rsa2048-g1`，藉此修改 `mydbinstance`。

針對 Linux、macOS 或 Unix：

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

在 Windows 中：

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

> **Note**
>
> 如果您的執行個體需要重新開機，可以使用 modify-db-instance CLI 命令並指定 `--no-certificate-rotation-restart` 選項。

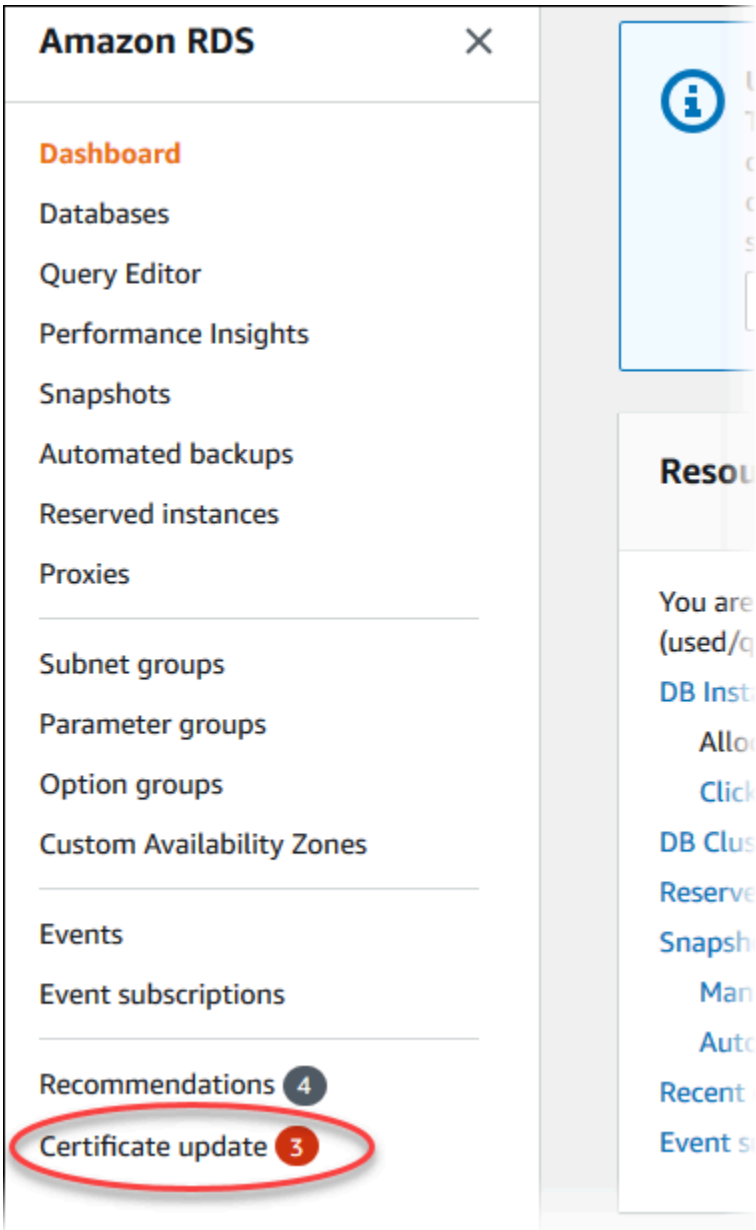
透過套用維護來更新憑證授權機構憑證

透過套用維護，執行下列步驟來套用 CA 憑證。

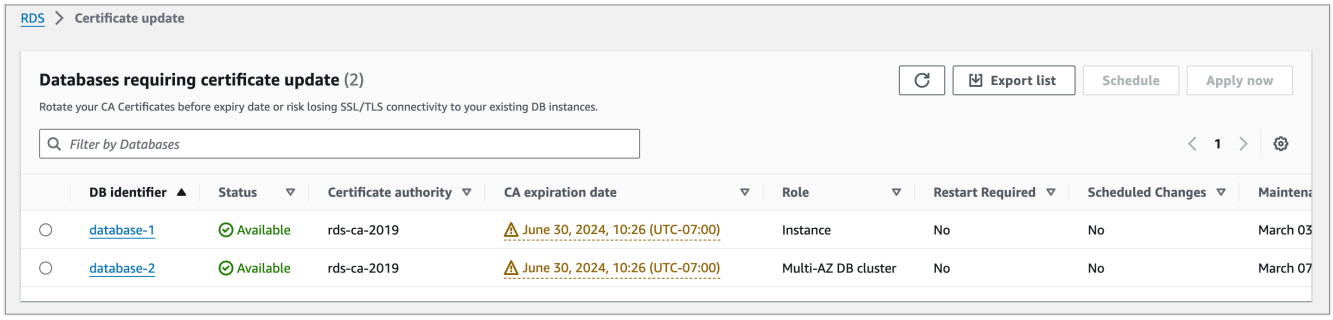
Console

透過套用維護來更新 CA 憑證

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇憑證更新。



需要更新憑證的資料庫頁面隨即顯示。



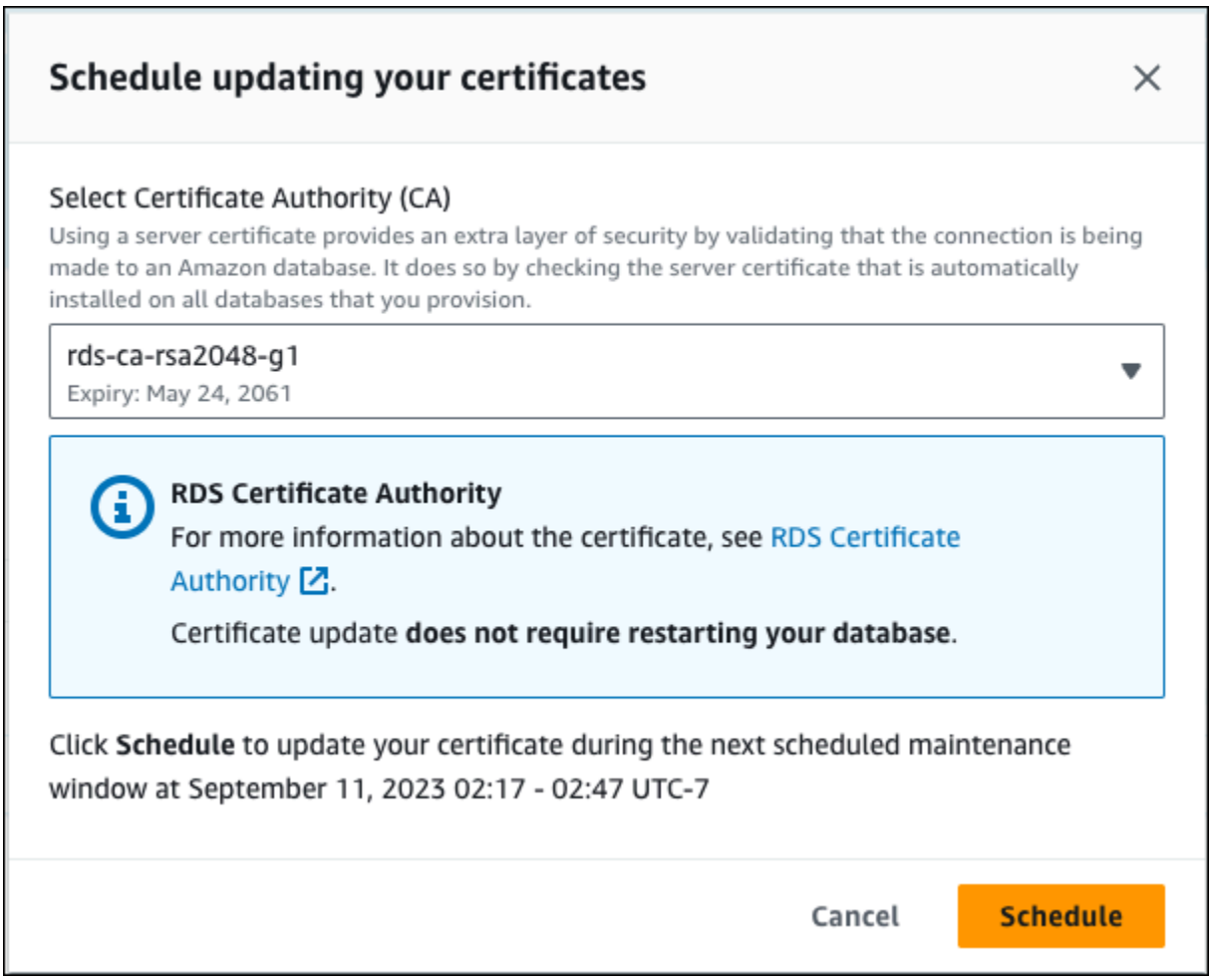
> **Note**
>
> 此頁面僅顯示目前 的資料庫執行個體 AWS 區域。如果您在多個資料庫中有資料庫 AWS 區域，請檢查每個資料庫的此頁面 AWS 區域 ，以查看具有舊 SSL/TLS 憑證的所有資料庫執行個體。

3. 選擇您要更新的資料庫執行個體。

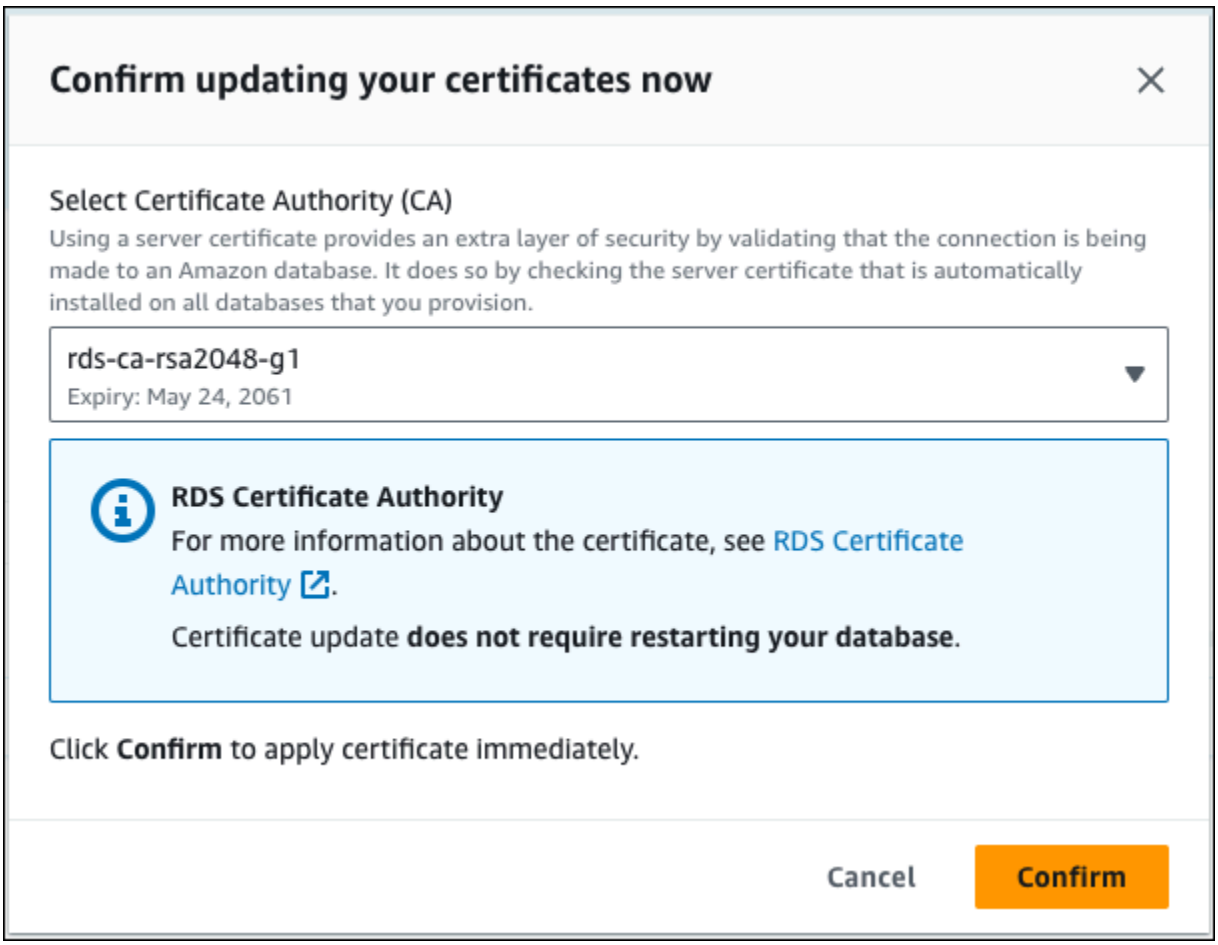
   您可以選擇排程，以排定下一個維護時段的憑證輪換。您可以選擇立即套用以立即套用輪換。

> **Important**
>
> 如果在憑證到期後發生連線問題，請使用立即套用選項。

4. a. 如果您選擇排程，系統會提示您確認 CA 憑證輪換。此提示也會指出排定的更新時段。



b. 如果您選擇立即套用，系統會提示您確認 CA 憑證輪換。



> **Important**
>
> 在資料庫上排定 CA 憑證輪換之前，請先更新使用 SSL/TLS 的任何用戶端應用程式和要連結伺服器憑證。這些更新專屬於您的資料庫引擎。更新這些用戶端應用程式之後，您可以確認 CA 憑證輪換。

若要繼續，請選擇核取方塊，然後選擇 Confirm (確認)。

5. 針對您要更新的每個資料庫執行個體重複步驟 3 和 4。

自動伺服器憑證輪換

如果您的根 CA 支援自動伺服器憑證輪換，RDS 會自動處理資料庫伺服器憑證的輪換。RDS 會使用相同的根 CA 進行此自動輪換，因此您不需要下載新的 CA 套件。請參閱 憑證授權單位。

資料庫伺服器憑證的輪換和有效期取決於資料庫引擎：

- 如果您的資料庫引擎支援在不重新啟動的情況下輪換，則 RDS 會自動輪換資料庫伺服器憑證，而不需要您採取任何動作。RDS 會嘗試在資料庫伺服器憑證有效期過半時，在您偏好的維護時段內輪換您的資料庫伺服器憑證。新的資料庫伺服器憑證有效期為 12 個月。
- 如果您的資料庫引擎不支援在未重新啟動的情況下輪換，Amazon RDS 會透過 Describe-pending-maintenance-actions API、憑證的一半生命週期，或在過期前至少 3 個月顯示`server-certificate-rotation`待定維護動作。您可以使用 apply-pending-maintenance-action API 來套用輪換。新的資料庫伺服器憑證有效期為 36 個月。

使用 describe-db-engine-versions 命令並檢查 `SupportsCertificateRotationWithoutRestart` 旗標，以確定資料庫引擎版本是否支援在不重新啟動的情況下輪換憑證。如需詳細資訊，請參閱 設定資料庫的 CA。

將憑證匯入信任存放區的範例指令碼

以下是範例 Shell 指令碼，會將憑證套件匯入信任存放區。

每個範例 Shell 指令碼都使用 keytool，其是 Java 開發套件 (JDK) 的一部分。如需安裝 JDK 的詳細資訊，請參閱《JDK 安裝指南》。

Linux

以下範例 Shell 指令碼將憑證套件匯入 Linux 作業系統上的信任存放區。

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi truststore=${mydir}/rds-truststore.jks storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem">
 ${mydir}/global-bundle.pem
awk 'split_after == 1 {n++;split_after=0} /-----END CERTIFICATE-----/
 {split_after=1}{print > "rds-ca-" n+1 ".pem"}' < ${mydir}/global-bundle.pem

for CERT in rds-ca-*; do alias=$(openssl x509 -noout -text -in $CERT | perl -ne
 'next unless /Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
```

```
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias |
 cut -d " " -f3- | while read alias
do expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -
alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

macOS

以下是範例 Shell 指令碼，會將憑證套件匯入 macOS 上的信任存放區。

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi truststore=${mydir}/rds-truststore.jks storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem">
 ${mydir}/global-bundle.pem
split -p "-----BEGIN CERTIFICATE-----" ${mydir}/global-bundle.pem rds-ca-

for CERT in rds-ca-*; do alias=$(openssl x509 -noout -text -in $CERT | perl -ne
 'next unless /Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias |
 cut -d " " -f3- | while read alias
do expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -
alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
```

```
   echo " Certificate ${alias} expires in '$expiry'"
done
```

若要進一步了解將 SSL 與 Amazon RDS 搭配使用的最佳實務，請參閱與 Amazon RDS for Oracle 成功進行 SSL 連線的最佳實務。

網際網路流量隱私權

Amazon Aurora 和內部部署應用程式之間，以及 Amazon Aurora 和相同 AWS 區域內的其他 AWS 資源之間的連線都會受到保護。

服務和內部部署用戶端與應用程式之間的流量。

您的私有網路與 之間有兩個連線選項 AWS：

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

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

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

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

Amazon Aurora 的 Identity and access management

AWS Identity and Access Management (IAM) 是 AWS 服務 ，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可控制哪些人員可進行身分驗證 (登入) 並獲得授權 (具有許可) 以使用 Amazon RDS 資源。IAM 是您可以免費使用 AWS 服務 的 。

主題

- 目標對象
- 使用身分來驗證
- 使用政策管理存取權
- Amazon Aurora 如何搭配 IAM 運作
- Amazon Aurora 以身分為基礎的政策範例
- AWS Amazon RDS 的 受管政策
- Amazon RDS 的 AWS 受管政策更新
- 防止跨服務混淆代理人問題
- IAM 資料庫身分驗證
- 對 Amazon Aurora 身分與存取進行故障診斷

目標對象

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

服務使用者 – 若您使用 Aurora 來執行任務，您的管理員可以提供您需要的登入資料和許可。隨著您為了執行作業而使用的 Aurora 功能數量變多，您可能會需要額外的許可。了解存取的管理方式可協助您向管理員請求正確的許可。若您無法存取 Aurora 中的某項功能，請參閱對 Amazon Aurora 身分與存取進行故障診斷。

服務管理員 – 若您在公司負責管理 Aurora 資源，您應該具備 Aurora 的完整存取權限。您的任務是判斷員工應存取的 Aurora 功能及資源。您接著必須將請求提交給您的 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解您公司可搭配 Aurora 使用 IAM 的方式，請參閱 Amazon Aurora 如何搭配 IAM 運作。

管理員 - 如果您是管理員，請了解如何撰寫政策來管理 Aurora 存取權的詳細資訊。若要檢視您可以在 IAM 中使用的範例 Aurora 身分類型政策，請參閱 Amazon Aurora 以身分為基礎的政策範例。

使用身分來驗證

身分驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者、IAM 使用者或擔任 IAM 角色身分進行身分驗證。

您可以使用身分來源的登入資料，例如 AWS IAM Identity Center (IAM Identity Center)、單一登入身分驗證或 Google/Facebook 登入資料，以聯合身分的形式登入。如需有關登入的詳細資訊，請參閱《AWS 登入 使用者指南》中的如何登入您的 AWS 帳戶。

對於程式設計存取， AWS 提供 SDK 和 CLI 以密碼編譯方式簽署請求。如需詳細資訊，請參閱《IAM 使用者指南》中的 API 請求的AWS 第 4 版簽署程序。

AWS 帳戶根使用者

當您建立 時 AWS 帳戶，您會從一個名為 AWS 帳戶 theroot 使用者的登入身分開始，該身分可完整存取所有 AWS 服務 和 資源。強烈建議您不要以根使用者處理日常任務。有關需要根使用者憑證的任務，請參閱《IAM 使用者指南》中的需要根使用者憑證的任務。

聯合身分

最佳實務是要求人類使用者使用聯合身分提供者，以 AWS 服務 使用臨時憑證存取 。

聯合身分是您企業目錄、Web 身分提供者的使用者，或使用身分來源的 AWS 服務 憑證存取 Directory Service 。聯合身分會擔任提供臨時憑證的角色。

對於集中式存取權管理，我們建議使用 AWS IAM Identity Center。如需詳細資訊，請參閱 AWS IAM Identity Center 使用者指南中的什麼是 IAM Identity Center？。

IAM 使用者和群組

IAM 使用者是一種身分具備單人或應用程式的特定許可權。我們建議使用臨時憑證，而不是具有長期使用權憑證的 IAM 使用者。如需詳細資訊，請參閱《IAM 使用者指南》中的要求人類使用者使用聯合身分提供者來 AWS 使用臨時憑證存取 。

IAM 群組會指定 IAM 使用者集合，使管理大量使用者的許可權更加輕鬆。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 使用者的使用案例。

您可以使用 IAM 資料庫身分驗證來驗證您的資料庫叢集。

IAM 資料庫身分驗證使用 Aurora。如需使用 IAM 驗證資料庫叢集身分的詳細資訊，請參閱 IAM 資料庫身分驗證。

IAM 角色

IAM 角色是 中具有特定許可 AWS 帳戶 的身分。它與使用者相似，但是不會與特定人員建立關聯。您可以切換角色，暫時在 中擔任 IAM AWS 管理主控台 角色。您可以透過呼叫 AWS CLI 或 AWS API 操作或使用自訂 URL 來擔任角色。如需使用角色的方法詳細資訊，請參閱 IAM 使用者指南中的使用 IAM 角色。

使用暫時憑證的 IAM 角色在下列情況中非常有用：

- 暫時使用者許可 - 使用者可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需有關聯合角色的相關資訊，請參閱《IAM 使用者指南》中的為第三方身分提供者 (聯合) 建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱《AWS IAM Identity Center 使用者指南》中的許可集。
- 跨帳戶存取權：您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的主體) 存取您帳戶的資源。角色是授予跨帳戶存取權的主要方式。不過，對於某些 AWS 服務，您可以將政策直接連接到資源（而不是使用角色做為代理）。如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱IAM 使用者指南中的 IAM 角色與資源類型政策的差異。
- 跨服務存取 – 有些 AWS 服務 使用其他 中的功能 AWS 服務。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉送存取工作階段 – 轉送存取工作階段 (FAS) 使用呼叫 的委託人許可 AWS 服務，並結合 AWS 服務 請求向下游服務提出請求。如需提出 FAS 請求時的政策詳細資訊，請參閱轉發存取工作階段。
  - 服務角色 – 服務角色是服務擔任的 IAM 角色，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以委派許可權給 AWS 服務。
  - 服務連結角色 – 服務連結角色是一種連結至 的服務角色類型 AWS 服務。服務可以擔任代表您執行動作的角色。服務連結角色會出現在您的 中 AWS 帳戶 ，並由服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon EC2 上執行的應用程式 – 您可以使用 IAM 角色來管理在 EC2 執行個體上執行之應用程式的臨時登入資料，以及提出 AWS CLI 或 AWS API 請求。這是在 EC2 執行個體內儲存存取金鑰的較好方式。若要將 AWS 角色指派給 EC2 執行個體並將其提供給其所有應用程式，您可以建立連接

至執行個體的執行個體描述檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得臨時憑證。如需詳細資訊，請參閱 IAM 使用者指南中的使用 IAM 角色來授予許可權給 Amazon EC2 執行個體上執行的應用程式。

若要了解是否要使用 IAM 角色，請參閱《IAM 使用者指南》中的何時建立 IAM 角色 (而不是使用者)。

使用政策管理存取權

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

管理員可以使用政策來指定誰可以存取 AWS 資源，以及他們可以對這些資源執行哪些動作。每個 IAM 實體 (許可集或角色) 在開始時都沒有許可。換句話說，根據預設，使用者什麼都不做，甚至無法變更自己的密碼。若要授予使用者執行動作的許可，管理員必須將許可政策連接到使用者。或者，管理員可以將使用者新增到具備預定許可的群組。管理員將許可給予群組時，該群組中的所有使用者都會獲得那些許可。

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

身分型政策

身分類型政策是您可以連接到身分 (例如許可集或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分類型政策，請參閱 IAM 使用者指南中的建立 IAM 政策。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一許可集或角色。受管政策是獨立的政策，您可以連接到 AWS 帳戶中的多個許可集和角色。受管政策包括 AWS 受管政策和客戶受管政策。如需了解如何在受管政策及內嵌政策間選擇，請參閱 IAM 使用者指南中的在受管政策和內嵌政策間選擇。

如需 Amazon Aurora 特定 AWS 受管政策的相關資訊，請參閱 AWS Amazon RDS 的 受管政策。

其他政策類型

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

- 許可界限：許可界限是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (許可集或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 `Principal` 欄位中指定許可集或角色的資源型政策則不會受到許可界限限制。任何這些政策中的明確拒絕都會覆寫允許。如需許可界限的詳細資訊，請參閱《IAM 使用者指南》中的 IAM 實體許可界限。
- 服務控制政策 SCPs) – SCPs是 JSON 政策，可指定 中組織或組織單位 (OU) 的最大許可 AWS Organizations。 AWS Organizations 是一種服務，用於分組和集中管理您企業擁有的多個 AWS 帳戶。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 會限制成員帳戶中實體的許可，包括每個實體 AWS 帳戶根使用者。如需 Organizations 和 SCP 的詳細資訊，請參閱 AWS Organizations 使用者指南中的 SCP 運作方式。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合使用者的暫時工作階段時，作為參數傳遞。所產生工作階段的許可會是許可集或角色身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請參《IAM 使用者指南》中的工作階段政策。

多種政策類型

當多種類型的政策適用於請求時，產生的許可會更複雜而無法理解。若要了解如何 AWS 在涉及多個政策類型時決定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

Amazon Aurora 如何搭配 IAM 運作

在您使用 IAM 管理 Amazon Aurora 的存取權前，建議您先了解可搭配 Aurora 使用的 IAM 功能有哪些。

下表列出您可以搭配 Amazon Aurora 使用的 IAM 功能：

IAM 功能	Amazon Aurora 支援
身分型政策	是
資源型政策	否
政策動作	是
政策資源	是
政策條件索引鍵 (服務特定)	是

IAM 功能	Amazon Aurora 支援
ACL	否
以屬性為基礎的存取控制 (ABAC)	是
臨時憑證	是
轉寄存取工作階段	是
服務角色	是
服務連結角色	是

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

主題

- Aurora 以身分為基礎的政策
- Aurora 內以資源為基礎的政策
- Aurora 的政策動作
- Aurora 的政策資源
- Aurora 的政策條件金鑰
- Aurora 中的存取控制清單 (ACL)
- 政策中具有 Aurora 標籤的以屬性為基礎存取控制 Attribute-based access control (ABAC)
- 將暫時登入資料與 Aurora 搭配使用
- Aurora 的轉寄存取工作階段
- Aurora 的服務角色
- Aurora 的服務連結角色

Aurora 以身分為基礎的政策

支援身分型政策：是。

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

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。如要了解您在 JSON 政策中使用的所有元素，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素參考。

Aurora 以身分為基礎的政策範例

若要檢視 Aurora 身分類型政策的範例，請參閱 Amazon Aurora 以身分為基礎的政策範例。

Aurora 內以資源為基礎的政策

支援資源型政策：否。

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

如需啟用跨帳戶存取權，您可以在其他帳戶內指定所有帳戶或 IAM 實體作為資源型政策的主體。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的快帳戶資源存取。

Aurora 的政策動作

支援政策動作：是。

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

JSON 政策的 `Action` 元素描述您可以用來允許或拒絕政策中存取的動作。政策會使用動作來授予執行相關聯動作的許可。

Aurora 中的政策動作會在動作前使用以下前綴：`rds:`。例如，若要授予某人使用 Amazon RDS `DescribeDBInstances` API 操作描述資料庫執行個體的許可，請在其政策中包含 `rds:DescribeDBInstances` 動作。政策陳述式必須包含 `Action` 或 `NotAction` 元素。Aurora 會定義自己的一組動作，描述您可以使用此服務執行的任務。

若要在單一陳述式中指定多個 動作，請用逗號分隔，如下所示。

```
"Action": [
      "rds:action1",
      "rds:action2"
```

您也可以使用萬用字元 (*) 來指定多個動作。例如，如需指定開頭是 Describe 文字的所有動作，請包含以下動作：

```
"Action": "rds:Describe*"
```

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

Aurora 的政策資源

支援政策資源：是。

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

Resource JSON 政策元素可指定要套用動作的物件。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。若動作不支援資源層級許可，使用萬用字元 (*) 表示該陳述式適用於所有資源。

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

資料庫執行個體資源具有以下 Amazon Resource Name (ARN)。

```
arn:${Partition}:rds:${Region}:${Account}:{ResourceType}/${Resource}
```

如需 ARNs 格式的詳細資訊，請參閱 Amazon Resource Name (ARNs AWS 和服務命名空間。

例如，若要在陳述式中指定 dbtest 資料庫執行個體，請使用以下 ARN。

```
"Resource": "arn:aws:rds:us-west-2:123456789012:db:dbtest"
```

若要指定屬於特定帳戶的所有資料庫執行個體，請使用萬用字元 (*)。

```
"Resource": "arn:aws:rds:us-east-1:123456789012:db:*"
```

有些 RDS API 操作 (例如用來建立資源的操作)，無法在特定資源上執行。在這些情況下，請使用萬用字元 (*)。

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

許多 Amazon RDS API 操作都涉及多個資源。例如，`CreateDBInstance` 會建立資料庫執行個體。建立資料庫執行個體時，您可指定 使用者必須使用特定的安全群組和參數群組。若要在單一陳述式中指定多項資源，請使用逗號分隔 ARN。

```
"Resource": [
      "resource1",
      "resource2"
```

若要查看 Aurora 資源類型及其 ARN 的詳細資訊，請參閱服務授權參考中的 Amazon RDS 定義的資源。若要了解您可以使用哪些動作指定每個資源的 ARN，請參閱 Amazon RDS 定義的動作。

Aurora 的政策條件金鑰

支援服務特定政策條件金鑰：是。

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

`Condition` 元素會根據定義的條件，指定陳述式的執行時機。您可以建立使用條件運算子的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。若要查看所有 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的AWS 全域條件內容索引鍵。

Aurora 會定義自己的一組條件金鑰，也支援一些全域條件金鑰的使用。若要查看所有 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的AWS 全域條件內容索引鍵。

所有 RDS API 操作均支援 `aws:RequestedRegion` 條件金鑰。

若要查看 Aurora 條件金鑰清單, 請參閱服務授權參考中的 Amazon RDS 的條件金鑰。若要了解您可以搭配哪些動作和資源使用條件金鑰，請參閱 Amazon RDS 定義的動作。

Aurora 中的存取控制清單 (ACL)

支援存取控制清單 (ACL)：否

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

政策中具有 Aurora 標籤的以屬性為基礎存取控制 Attribute-based access control (ABAC)

在政策中支援屬性型存取控制 (ABAC) 標籤：是

屬性型存取控制 (ABAC) 是一種授權策略，依據稱為標籤的屬性來定義許可。您可以將標籤連接至 IAM 實體 AWS 和資源，然後設計 ABAC 政策，以便在委託人的標籤符合資源上的標籤時允許操作。

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

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

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

如需標記 Aurora 資源的詳細資訊，請參閱指定條件：使用自訂標籤。若要檢視身分型原則範例，以根據該資源上的標籤來限制存取資源，請參閱利用具有兩個不同值的特定標籤，對資源上的動作授予許可。

將暫時登入資料與 Aurora 搭配使用

支援臨時憑證：是。

臨時登入資料提供 AWS 資源的短期存取權，當您使用聯合或切換角色時，會自動建立。 AWS 建議您動態產生臨時登入資料，而不是使用長期存取金鑰。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的臨時安全憑證與可與 IAM 搭配運作的AWS 服務。

Aurora 的轉寄存取工作階段

支援轉寄存取工作階段：是。

轉送存取工作階段 (FAS) 使用呼叫 的委託人許可 AWS 服務，結合 AWS 服務 請求向下游服務提出請求。如需提出 FAS 請求時的政策詳細資訊，請參閱轉發存取工作階段。

Aurora 的服務角色

支援服務角色：是。

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

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

Aurora 的服務連結角色

支援服務連結角色：是。

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

如需使用 Aurora 服務連結角色的詳細資訊，請參閱使用 Amazon Aurora 的服務連結角色。

Amazon Aurora 以身分為基礎的政策範例

根據預設，許可集和角色不具備建立或修改 Aurora 資源的許可。他們也無法使用 AWS 管理主控台 AWS CLI、 或 AWS API 執行任務。IAM 管理員必須建立 IAM 政策，授予許可集和角色在指定資源上執行特定 API 操作的所需許可。管理員接著必須將這些政策連接至需要這些許可的許可集或角色。

若要了解如何使用這些範例 JSON 政策文件建立 IAM 身分型政策，請參閱《IAM 使用者指南》中的在 JSON 標籤上建立政策。

主題

- 政策最佳實務
- 使用 Aurora 主控台
- 使用主控台所需的許可
- 允許使用者檢視他們自己的許可
- 在 Aurora 中建立、修改和刪除資源的許可政策
- 範例政策：使用條件金鑰
- 指定條件：使用自訂標籤

- 在建立期間授予標籤 Aurora 資源的許可

政策最佳實務

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

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予您的使用者和工作負載，請使用將許可授予許多常見使用案例的 AWS 受管政策。它們可在您的 中使用 AWS 帳戶。我們建議您定義特定於使用案例 AWS 的客戶受管政策，以進一步減少許可。如需更多資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策或任務職能的AWS 受管政策。
- 套用最低權限許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱《IAM 使用者指南》中的 IAM 中的政策和許可。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。如果透過特定 等使用服務動作 AWS 服務，您也可以使用條件來授予其存取權 CloudFormation。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您撰寫安全且實用的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM Access Analyzer 驗證政策。
- 需要多重要素驗證 (MFA) – 如果您的案例需要 IAM 使用者或 中的根使用者 AWS 帳戶，請開啟 MFA 以提高安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的透過 MFA 的安全 API 存取。

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

使用 Aurora 主控台

若要存取 Amazon Aurora 主控台，您必須擁有最低的一組許可。這些許可必須允許您列出和檢視 中 Amazon Aurora 資源的詳細資訊 AWS 帳戶。如果您建立比最基本必要許可更嚴格的身分型政策，則對於具有該政策的實體 (使用者或角色) 而言，主控台就無法如預期運作。

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

為了確保這些實體仍然可以使用 Aurora主控台，請將下列 AWS 受管政策連接至實體。

```
AmazonRDSReadOnlyAccess
```

如需詳細資訊，請參閱《IAM 使用者指南》中的新增許可到使用者。

使用主控台所需的許可

針對使用主控台的使用者，該使用者必須擁有一組符合最低限制的許可。這些許可允許使用者描述其 AWS 帳戶的 Amazon Aurora 資源，並提供其他相關資訊，包括 Amazon EC2 安全性和網路資訊。

如果您建立比最基本必要許可更嚴格的 IAM 政策，則對於採取該 IAM 政策的使用者而言，主控台就無法如預期運作。為確保這些使用者仍可使用主控台，也請將 AmazonRDSReadOnlyAccess 受管政策連接至使用者，如使用政策管理存取權所述。

對於僅呼叫 AWS CLI 或 Amazon RDS API 的使用者，您不需要允許其最基本主控台許可。

下列政策授予根 AWS 帳戶所有 Amazon Aurora 資源的完整存取權：

```
AmazonRDSFullAccess
```

允許使用者檢視他們自己的許可

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

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
```

```
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

在 Aurora 中建立、修改和刪除資源的許可政策

下列各節提供授予和限制資源存取權的許可政策範例：

允許使用者在 AWS 帳戶中建立資料庫執行個體

以下是允許 ID 為您的帳戶為 AWS 您的帳戶123456789012建立資料庫執行個體的範例政策。此政策需要新資料庫執行個體的名稱以 `test` 開頭。新的資料庫執行個體也須使用 MySQL 資料庫引擎和 `db.t2.micro` 資料庫執行個體類別。此外，新的資料庫執行個體還須使用選項群組，以及以 `default` 開頭的資料庫參數群組，而且它還須使用 `default` 子網路群組。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "AllowCreateDBInstanceOnly",
         "Effect": "Allow",
         "Action": [
            "rds:CreateDBInstance"
```

```
            ],
            "Resource": [
               "arn:aws:rds:*:123456789012:db:test*",
               "arn:aws:rds:*:123456789012:og:default*",
               "arn:aws:rds:*:123456789012:pg:default*",
               "arn:aws:rds:*:123456789012:subgrp:default"
            ],
            "Condition": {
               "StringEquals": {
                  "rds:DatabaseEngine": "mysql",
                  "rds:DatabaseClass": "db.t2.micro"
               }
            }
         }
      ]
}
```

政策包括單一陳述式，指定 使用者的下列許可：

- 此政策允許帳戶使用 CreateDBInstance API 操作建立資料庫執行個體 （這也適用於 create-db-instance AWS CLI 命令和 AWS 管理主控台)。
- Resource 元素指定使用者可對資源或搭配資源執行動作。您可以使用 Amazon Resource Name (ARN) 來指定資源。此 ARN 包含資源所屬的服務名稱 (rds)、區域 AWS (* 表示此範例中的任何區域）、 AWS 帳戶號碼 (123456789012 是此範例中的帳戶號碼），以及資源類型。如需建立 ARN 的詳細資訊，請參閱Amazon RDS 中的 Amazon Resource Name (ARN)。

  範例中的 Resource 元素對使用者的資源指定下列政策限制：

  - 新資料庫執行個體的資料庫執行個體識別符必須以 test開頭 (例如，testCustomerData1、test-region2-data)。
  - 新資料庫執行個體的選項群組必須以default 開頭。
  - 新資料庫執行個體的資料庫參數群組必須以default 開頭。
  - 新資料庫執行個體的子網路群組必須是default 子網路群組。
- Condition 元素指定資料庫引擎必須是 MySQL，而且資料庫執行個體類別必須是 db.t2.micro。Condition 元素指定政策應該生效時的條件。您可以使用 Condition 元素來新增其他許可或限制。如需指定條件的詳細資訊，請參閱Aurora 的政策條件金鑰。此範例指定 rds:DatabaseEngine 和 rds:DatabaseClass 條件。如需 rds:DatabaseEngine 有效條件值的相關資訊，請參閱 CreateDBInstance 中的 Engine 參數下的清單。如需 rds:DatabaseClass 的有效條件值的相關資訊，請參閱 資料庫執行個體類別的支援資料庫引擎。

此政策不指定 Principal 元素，因為您不會在以身分為基礎的政策中，指定取得許可的主體。當您將政策連接至使用者時，這名使用者即為隱含主體。當您將許可政策連接至 IAM 角色，該角色的信任政策中所識別的主體即取得許可。

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

允許使用者對任何 RDS 資源執行任何描述動作

下列許可政策會授予使用者執行開頭為 Describe 之所有動作的許可。這些動作會顯示 RDS 資源 (如資料庫執行個體) 的相關資訊。Resource 元素中的萬用字元 (*) 表示可對帳戶擁有的所有 Amazon Aurora 資源執行動作。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "AllowRDSDescribe",
         "Effect": "Allow",
         "Action": "rds:Describe*",
         "Resource": "*"
      }
   ]
}
```

允許使用者建立可使用指定資料庫參數群組和子網路群組的資料庫執行個體

以下許可政策授予許可，以允許使用者只能建立一個必須使用 mydbpg 資料庫參數群組和 mydbsubnetgroup 資料庫子網路群組的資料庫執行個體。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": "rds:CreateDBInstance",
```

```
         "Resource": [
            "arn:aws:rds:*:*:pg:mydbpg",
            "arn:aws:rds:*:*:subgrp:mydbsubnetgroup"
         ]
      }
   ]
}
```

利用具有兩個不同值的特定標籤，對資源上的動作授予許可

您可以在身分類型政策中使用條件，根據標籤來控制存取 Aurora 資源。下列政策允許在 stage 標籤設為 development 或 test 的資料庫執行個體上執行 CreateDBSnapshot API 操作的許可。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAnySnapshotName",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
      },
      {
         "Sid":"AllowDevTestToCreateSnapshot",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
```

```
   ]
}
```

下列政策允許在 stage 標籤設為 development 或 test 的資料庫執行個體上執行 ModifyDBInstance API 操作的許可。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowChangingParameterOptionSecurityGroups",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource": [
            "arn:aws:rds:*:123456789012:pg:*",
            "arn:aws:rds:*:123456789012:secgrp:*",
            "arn:aws:rds:*:123456789012:og:*"
         ]
      },
      {
         "Sid":"AllowDevTestToModifyInstance",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

防止使用者刪除資料庫執行個體

以下許可政策授予許可，以防止使用者刪除特定的資料庫執行個體。例如，您可能想拒絕給予任何非管理員使用者刪除生產資料庫執行個體的能力。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "DenyDelete1",
         "Effect": "Deny",
         "Action": "rds:DeleteDBInstance",
         "Resource": "arn:aws:rds:us-west-2:123456789012:db:my-mysql-instance"
      }
   ]
}
```

拒絕對資源的所有存取

您可以明確拒絕對資源的存取權。拒絕政策優先於允許政策。下列政策明確拒絕使用者管理資源的能力：

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Deny",
         "Action": "rds:*",
         "Resource": "arn:aws:rds:us-east-1:123456789012:db:mydb"
      }
   ]
```

```
}
```

範例政策：使用條件金鑰

以下是如何在 Amazon Aurora IAM 許可政策中使用條件金鑰的範例。

範例 1：授予許可來建立一個使用特定資料庫引擎且不是多個可用區的資料庫執行個體

下列政策會使用 RDS 條件金鑰，並允許使用者只建立使用 MySQL 資料庫引擎，而不使用 MultiAZ 的資料庫執行個體。Condition 元素表示資料庫引擎是 MySQL 的需求。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "AllowMySQLCreate",
         "Effect": "Allow",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseEngine": "mysql"
            },
            "Bool": {
               "rds:MultiAz": false
            }
         }
      }
   ]
}
```

範例 2：明確拒絕許可，不得為特定資料庫執行個體類別建立資料庫執行個體，以及不得建立使用佈建 IOPS 的資料庫執行個體

下列政策明確拒絕許可，不得建立使用資料庫執行個體類別 r3.8xlarge 和 m4.10xlarge 的資料庫執行個體，因為它們是最大且最昂貴的資料庫執行個體類別。此政策也會防止使用者建立使用佈建 IOPS 的資料庫執行個體，因為它們會產生額外的成本。

明確拒絕許可會取代任何其他已授予的許可。這確保身分不會意外取得您從未想要授予的許可。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "DenyLargeCreate",
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseClass": [
                  "db.r3.8xlarge",
                  "db.m4.10xlarge"
               ]
            }
         }
      },
      {
         "Sid": "DenyPIOPSCreate",
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "NumericNotEquals": {
               "rds:Piops": "0"
            }
         }
      }
   ]
}
```

範例 3：限制可用來標記資源的一組標籤金鑰和值

下列政策使用 RDS 條件索引鍵，並允許將索引鍵為 `stage` 且值為 `test`、`qa` 和 `production` 的標籤新增給資源。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowTagEdits",
      "Effect": "Allow",
      "Action": [
        "rds:AddTagsToResource",
        "rds:RemoveTagsFromResource"
      ],
      "Resource": "arn:aws:rds:us-east-1:123456789012:db:db-123456",
      "Condition": {
        "StringEquals": {
          "rds:req-tag/stage": [
            "test",
            "qa",
            "production"
          ]
        }
      }
    }
  ]
}
```

指定條件：使用自訂標籤

Amazon Aurora 支援在 IAM 政策中使用自訂標籤來指定條件。

例如，假設您將值為 `environment`、`beta`、`staging` 等等的 `production` 標籤新增至資料庫執行個體。如果這樣做，您就可以建立政策，根據 `environment` 標籤值限制某些資料庫執行個體的使用者。

> **Note**
>
> 自訂標籤識別符會區分大小寫。

下表列出您可以在 Condition 元素中使用的 RDS 標籤識別符。

RDS 標籤識別符	適用對象
db-tag	資料庫執行個體，包括讀取複本
snapshot-tag	資料庫快照
ri-tag	預留資料庫執行個體
og-tag	資料庫選項群組
pg-tag	資料庫參數群組
subgrp-tag	資料庫子網路群組
es-tag	事件訂閱
cluster-tag	資料庫叢集
cluster-pg-tag	資料庫叢集參數群組
cluster-snapshot-tag	資料庫叢集快照

自訂標籤條件的語法如下：

```
"Condition":{"StringEquals":{"rds:rds-tag-identifier/tag-name":
["value"]} }
```

例如，下列 Condition 元素會套用至標籤名稱為 environment 且標籤值為 production 的資料庫執行個體。

```
"Condition":{"StringEquals":{"rds:db-tag/environment": ["production"]} }
```

如需建立標籤的相關資訊，請參閱標記 Amazon Aurora 和 Amazon RDS 資源。

> **Important**
>
> 如果您使用標記功能來管理 RDS 資源的存取，則我們建議您安全存取 RDS 資源的標籤。您可以建立 AddTagsToResource 和 RemoveTagsFromResource 動作的政策，來管理標籤的

存取。例如，下列政策會拒絕使用者可對所有資源新增或移除標籤的能力。然後，您可以建立政策來允許特定使用者新增或刪除標籤。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyTagUpdates",
         "Effect":"Deny",
         "Action":[
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource":"*"
      }
   ]
}
```

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

範例政策：使用自訂標籤

以下是如何在 Amazon Aurora IAM 許可政策中使用自訂標籤的範例。如需將標籤新增至 Amazon Aurora 資源的詳細資訊，請參閱Amazon RDS 中的 Amazon Resource Name (ARN)。

> **Note**
>
> 所有範例都使用 us-west-2 區域，且其中的帳戶 ID 皆為虛構。

範例 1：利用具有兩個不同值的特定標籤，對資源上的動作授予許可

下列政策允許在 `stage` 標籤設為 `development` 或 `test` 的資料庫執行個體上執行 `CreateDBSnapshot` API 操作的許可。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAnySnapshotName",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
      },
      {
         "Sid":"AllowDevTestToCreateSnapshot",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

下列政策允許在 `stage` 標籤設為 `development` 或 `test` 的資料庫執行個體上執行 `ModifyDBInstance` API 操作的許可。

JSON

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

```
      {
         "Sid":"AllowChangingParameterOptionSecurityGroups",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
            ],
          "Resource": [
            "arn:aws:rds:*:123456789012:pg:*",
            "arn:aws:rds:*:123456789012:secgrp:*",
            "arn:aws:rds:*:123456789012:og:*"
            ]
       },
       {
         "Sid":"AllowDevTestToModifyInstance",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
            ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
               "rds:db-tag/stage":[
                  "development",
                  "test"
                  ]
               }
            }
       }
    ]
}
```

範例 2：明確拒絕許可，不得建立一個使用所指定之資料庫參數群組的資料庫執行個體

下列政策明確拒絕許可，不得建立一個使用資料庫參數群組與特定標籤值搭配的資料庫執行個體。如果您需要在建立資料庫執行個體時，一律使用特定客戶建立的資料庫參數群組，則可能會套用此政策。使用 Deny 的政策最常用來限制更廣泛政策所授予的存取。

明確拒絕許可會取代任何其他已授予的許可。這確保身分不會意外取得您從未想要授予的許可。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyProductionCreate",
         "Effect":"Deny",
         "Action":"rds:CreateDBInstance",
         "Resource":"arn:aws:rds:*:123456789012:pg:*",
         "Condition":{
            "StringEquals":{
               "rds:pg-tag/usage":"prod"
            }
         }
      }
   ]
}
```

範例 3：對資料庫執行個體上的動作授予許可，而此資料庫執行個體的執行個體名稱字首會加上使用者名稱

下列政策允許在資料庫執行個體上呼叫任何 API (但 AddTagsToResource 或 RemoveTagsFromResource 除外) 的許可，而此資料庫執行個體具有字首會加上使用者名稱的資料庫執行個體名稱，以及具有稱為 stage 等於 devo 的標籤，或沒有稱為 stage 的標籤。

政策中的 Resource 一行會依其 Amazon Resource Name (ARN) 識別資源。如需使用 ARN 與 Amazon Aurora 資源的詳細資訊，請參閱Amazon RDS 中的 Amazon Resource Name (ARN)。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowFullDevAccessNoTags",
         "Effect":"Allow",
         "NotAction":[
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
```

```
            ],
            "Resource":"arn:aws:rds:*:123456789012:db:${aws:username}*",
            "Condition":{
               "StringEqualsIfExists":{
                  "rds:db-tag/stage":"devo"
               }
            }
         }
      ]
}
```

在建立期間授予標籤 Aurora 資源的許可

有些 RDS API 操作可讓您在建立資源時指定標籤。您可以使用資源標籤來實作以屬性為基礎的控制 (ABAC)。如需詳細資訊，請參閱什麼是 ABAC AWS？和使用標籤控制對 AWS 資源的存取。

使用者若要在建立時標記資源，他們必須具備建立資源動作 (例如 `rds:CreateDBCluster`) 的使用許可。若標籤於建立動作中指定，RDS 會針對 `rds:AddTagsToResource` 動作執行其他授權，以確認使用者具備建立標籤的許可。因此，使用者必須同時具備使用 `rds:AddTagsToResource` 動作的明確許可。

在 `rds:AddTagsToResource` 動作的 IAM 政策定義中，您可以使用 `aws:RequestTag` 條件索引鍵要求請求中的標籤來標記資源。

例如，下列政策允許使用者在建立資料庫執行個體期間建立資料庫執行個體並套用標籤，但只能使用特定標籤索引鍵 (`environment` 或 `project`)：

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Action": [
               "rds:CreateDBInstance"
           ],
           "Resource": "*"
       },
       {
           "Effect": "Allow",
```

```
            "Action": [
                "rds:AddTagsToResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/environment": ["production", "development"],
                    "aws:RequestTag/project": ["dataanalytics", "webapp"]
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": ["environment", "project"]
                }
            }
        }
    ]
}
```

此政策拒絕任何包含 environment 或 project 標籤以外之標籤的建立資料庫執行個體請求，或未指定這些標籤的建立資料庫執行個體請求。此外，使用者必須為符合政策中允許值的標籤指定值。

下列政策可讓使用者建立資料庫叢集，並在建立期間套用 environment=prod 標籤以外的任何標籤：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds:CreateDBCluster"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "rds:AddTagsToResource"
            ],
            "Resource": "*",
```

```
            "Condition": {
                "StringNotEquals": {
                    "aws:RequestTag/environment": "prod"
                }
            }
        }
    ]
}
```

建立時用於標記的支援 RDS API 動作

下列 RDS API 動作支援在建立資源時進行標記。對於這些動作，您可以在建立資源時指定標籤：

- CreateBlueGreenDeployment
- CreateCustomDBEngineVersion
- CreateDBCluster
- CreateDBClusterEndpoint
- CreateDBClusterParameterGroup
- CreateDBClusterSnapshot
- CreateDBInstance
- CreateDBInstanceReadReplica
- CreateDBParameterGroup
- CreateDBProxy
- CreateDBProxyEndpoint
- CreateDBSecurityGroup
- CreateDBShardGroup
- CreateDBSnapshot
- CreateDBSubnetGroup
- CreateEventSubscription
- CreateGlobalCluster
- CreateIntegration
- CreateOptionGroup
- CreateTenantDatabase
- CopyDBClusterParameterGroup

- CopyDBClusterSnapshot
- CopyDBParameterGroup
- CopyDBSnapshot
- CopyOptionGroup
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime
- RestoreDBInstanceFromDBSnapshot
- RestoreDBInstanceFromS3
- RestoreDBInstanceToPointInTime
- PurchaseReservedDBInstancesOffering

如果您使用 AWS CLI 或 API 建立具有標籤的資源， Tags 參數會用來在建立期間將標籤套用至資源。

對於這些 API 動作，如果標記失敗，則不會建立資源，且請求會失敗並顯示錯誤。這可確保使用標籤建立資源，或完全不建立資源，以防止在沒有預期標籤的情況下建立資源。

AWS Amazon RDS 的 受管政策

若要將許可新增至許可集和角色，使用 AWS 受管政策比自行撰寫政策更容易。建立 IAM 客戶受管政策需要時間和專業知識，而受管政策可為您的團隊提供其所需的許可。若要快速開始使用，您可以使用我們的 AWS 受管政策。這些政策涵蓋常見的使用案例，並可在您的 AWS 帳戶中使用。如需 AWS 受管政策的詳細資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策。

AWS 服務 維護和更新 AWS 受管政策。您無法變更 AWS 受管政策中的許可。服務偶爾會將其他許可新增至 AWS 受管政策，以支援新功能。此類型的更新會影響已連接政策的所有身分識別 (許可集和角色)。當新功能啟動或新操作可用時，服務最有可能更新 AWS 受管政策。服務不會從 AWS 受管政策中移除許可，因此政策更新不會破壞您現有的許可。

此外， AWS 支援跨多個 服務之任務函數的受管政策。例如， `ReadOnlyAccess` AWS 受管政策提供所有 AWS 服務 和 資源的唯讀存取權。當服務啟動新功能時， 會為新操作和資源 AWS 新增唯讀許可。如需任務職能政策的清單和說明，請參閱 IAM 使用者指南中有關任務職能的AWS 受管政策。

主題

- AWS 受管政策：AmazonRDSReadOnlyAccess
- AWS 受管政策：AmazonRDSFullAccess
- AWS 受管政策：AmazonRDSDataFullAccess
- AWS 受管政策：AmazonRDSEnhancedMonitoringRole
- AWS 受管政策：AmazonRDSPerformanceInsightsReadOnly
- AWS 受管政策：AmazonRDSPerformanceInsightsFullAccess
- AWS 受管政策：AmazonRDSDirectoryServiceAccess
- AWS 受管政策：AmazonRDSServiceRolePolicy
- AWS 受管政策：AmazonRDSPreviewServiceRolePolicy
- AWS 受管政策：AmazonRDSBetaServiceRolePolicy

AWS 受管政策：AmazonRDSReadOnlyAccess

此政策允許透過 對 Amazon RDS 進行唯讀存取 AWS 管理主控台。

許可詳細資訊

此政策包含以下許可：

- `rds` – 允許主體說明 Amazon RDS 資源並列出 Amazon RDS 資源的標籤。

- `cloudwatch` – 允許主體獲取 Amazon CloudWatch 指標統計資料。
- `ec2` – 允許主體說明可用區域和聯網資源。
- `logs` – 允許主體說明日誌群組的 CloudWatch Logs 日誌串流，並取得 CloudWatch Logs 事件。
- `devops-guru` - 允許主體描述具有 Amazon DevOps Guru 涵蓋範圍的資源，這是由 CloudFormation 堆疊名稱或資源標籤所指定的。

如需有關此政策的詳細資訊，包括 JSON 政策文件，請參閱《AWS 受管政策參考指南》中的 AmazonRDSReadOnlyAccess。

AWS 受管政策：AmazonRDSFullAccess

此政策透過 提供 Amazon RDS 的完整存取權 AWS 管理主控台。

許可詳細資訊

此政策包含以下許可：

- `rds` – 允許主體完整存取 Amazon RDS。
- `application-autoscaling` – 允許主體說明和管理 應用程式自動擴展擴展目標和政策。
- `cloudwatch` – 允許主體取得 CloudWatch 指標靜態並管理 CloudWatch 警示。
- `ec2` – 允許主體說明可用區域和聯網資源。
- `logs` – 允許主體說明日誌群組的 CloudWatch Logs 日誌串流，並取得 CloudWatch Logs 事件。
- `outposts` – 允許主體取得 AWS Outposts 執行個體類型。
- `pi` – 允許主體取得績效詳情指標。
- `sns` – 允許主體訂閱 Amazon Simple Notification Service (Amazon SNS) 和主題，並發佈 Amazon SNS 訊息。
- `devops-guru` - 允許主體描述具有 Amazon DevOps Guru 涵蓋範圍的資源，這是由 CloudFormation 堆疊名稱或資源標籤所指定的。

如需有關此政策的詳細資訊，包括 JSON 政策文件，請參閱《AWS 受管政策參考指南》中的 AmazonRDSFullAccess。

AWS 受管政策：AmazonRDSDataFullAccess

此政策允許完整存取在特定 Aurora Serverless叢集上使用資料 API 和查詢編輯器 AWS 帳戶。此政策允許 從 AWS 帳戶 取得秘密的值 AWS Secrets Manager。

您可將 AmazonRDSDataFullAccess 政策連接到 IAM 身分。

許可詳細資訊

此政策包含以下許可：

- dbqms – 允許主體存取、建立、刪除、說明和更新查詢。Database Query Metadata Service (dbqms) 是僅內部服務。它為多個 上的查詢編輯器提供最近和儲存 AWS 管理主控台 的查詢 AWS 服務，包括 Amazon RDS。
- rds-data – 允許主體在 Aurora Serverless 資料庫執行 SQL 陳述式。
- secretsmanager – 允許主體從中取得秘密的值 AWS Secrets Manager。

如需有關此政策的詳細資訊，包括 JSON 政策文件，請參閱《AWS 受管政策參考指南》中的 AmazonRDSDataFullAccess。

AWS 受管政策：AmazonRDSEnhancedMonitoringRole

此政策提供對 Amazon CloudWatch Logs 的存取權限，以進行 Amazon RDS 增強型監控。

許可詳細資訊

此政策包含以下許可：

- logs – 允許主體建立 CloudWatch Logs 日誌群組和保留政策，並建立和說明日誌群組的 CloudWatch Logs 日誌串流。其還允許主體放置並取得 CloudWatch Logs 日誌事件。

如需有關此政策的詳細資訊，包括 JSON 政策文件，請參閱《AWS 受管政策參考指南》中的 AmazonRDSEnhancedMonitoringRole。

AWS 受管政策：AmazonRDSPerformanceInsightsReadOnly

此政策提供對 Amazon RDS 資料庫執行個體和 Amazon Aurora 資料庫叢集的 Amazon RDS 績效詳情唯讀存取權限。

此政策現在包含 Sid (陳述式 ID) 作為政策陳述式的識別碼。

許可詳細資訊

此政策包含以下許可：

- `rds` – 允許主體說明 Amazon RDS 資料庫執行個體和 Amazon Aurora 資料庫叢集。
- `pi` – 允許主體呼叫 Amazon RDS 績效詳情 API 並存取績效詳情指標。

如需有關此政策的詳細資訊，包括 JSON 政策文件，請參閱《AWS 受管政策參考指南》中的 AmazonRDSPerformanceInsightsReadOnly。

AWS 受管政策：AmazonRDSPerformanceInsightsFullAccess

此政策提供對 Amazon RDS 資料庫執行個體和 Amazon Aurora 資料庫叢集的 Amazon RDS Performance Insights 完整存取權限。

此政策現在包含 `Sid` (陳述式 ID) 作為政策陳述式的識別符。

許可詳細資訊

此政策包含以下許可：

- `rds` – 允許主體說明 Amazon RDS 資料庫執行個體和 Amazon Aurora 資料庫叢集。
- `pi` – 允許主體呼叫 Amazon RDS Performance Insights API，以及建立、檢視和刪除績效分析報告。
- `cloudwatch`：允許主體列出 Amazon CloudWatch 指標並取得指標資料和統計資料。

如需有關此政策的詳細資訊，包括 JSON 政策文件，請參閱《AWS 受管政策參考指南》中的 AmazonRDSPerformanceInsightsFullAccess。

AWS 受管政策：AmazonRDSDirectoryServiceAccess

此政策允許 Amazon RDS 呼叫 Directory Service。

許可詳細資訊

此政策包含以下許可：

- `ds` – 允許主體描述 Directory Service 目錄並控制 Directory Service 目錄的授權。

如需有關此政策的詳細資訊，包括 JSON 政策文件，請參閱《AWS 受管政策參考指南》中的 AmazonRDSDirectoryServiceAccess。

AWS 受管政策：AmazonRDSServiceRolePolicy

您無法將 AmazonRDSServiceRolePolicy 政策附加至 IAM 實體。此政策會附加至服務連結角色，而此角色可讓 Amazon RDS 代表您執行動作。如需詳細資訊，請參閱Amazon Aurora 的服務連結角色許可。

AWS 受管政策：AmazonRDSPreviewServiceRolePolicy

您不應將 AmazonRDSPreviewServiceRolePolicy 連接到 IAM 實體。此政策會連接到服務連結角色，允許 Amazon RDS 代表 RDS 資料庫資源呼叫 AWS 服務。如需詳細資訊，請參閱Amazon RDS 預覽的服務連結角色。

許可詳細資訊

此政策包含以下許可：

- ec2 – 允許主體說明可用區域和聯網資源。
- secretsmanager – 允許主體從中取得秘密的值 AWS Secrets Manager。
- cloudwatch、logs - 允許 Amazon RDS 透過 CloudWatch 代理程式將資料庫執行個體指標和日誌上傳至 CloudWatch。

如需有關此政策的詳細資訊，包括 JSON 政策文件，請參閱《AWS 受管政策參考指南》中的 AmazonRDSPreviewServiceRolePolicy。

AWS 受管政策：AmazonRDSBetaServiceRolePolicy

您不應將 AmazonRDSBetaServiceRolePolicy 連接到 IAM 實體。此政策會連接到服務連結角色，允許 Amazon RDS 代表 RDS 資料庫資源呼叫 AWS 服務。如需詳細資訊，請參閱Amazon RDS Beta 的服務連結角色許可。

許可詳細資訊

此政策包含以下許可：

- ec2 - 允許 Amazon RDS 在提供時間點還原功能的資料庫執行個體上執行備份操作。
- secretsmanager - 允許 Amazon RDS 管理 Amazon RDS 建立的資料庫執行個體特定秘密。
- cloudwatch、logs - 允許 Amazon RDS 透過 CloudWatch 代理程式將資料庫執行個體指標和日誌上傳至 CloudWatch。

如需有關此政策的詳細資訊，包括 JSON 政策文件，請參閱《AWS 受管政策參考指南》中的 AmazonRDSBetaServiceRolePolicy。

Amazon RDS 的 AWS 受管政策更新

檢視自此服務開始追蹤 Amazon RDS AWS 受管政策更新以來的變更詳細資訊。如需有關此頁面變更的自動提醒，請訂閱 Amazon RDS Document history (文件歷程記錄) 頁面上的 RSS 摘要。

變更	描述	日期
AWS 受管政策：AmazonRDSPreviewServiceRolePolicy – 更新現有政策	Amazon RDS 已從 `AWSServiceRoleForRDSPreview` 服務連結角色的 `AmazonRDSPreviewServiceRolePolicy` 移除 `sns:Publish` 許可。如需更多詳細資訊，請參閱 AWS 受管政策：AmazonRDSPreviewServiceRolePolicy。	2024 年 8 月 7 日
AWS 受管政策：AmazonRDSBetaServiceRolePolicy – 更新現有政策	Amazon RDS 已從 `AWSServiceRoleForRDSBeta` 服務連結角色的 `AmazonRDSBetaServiceRolePolicy` 移除 `sns:Publish` 許可。如需更多詳細資訊，請參閱 AWS 受管政策：AmazonRDSBetaServiceRolePolicy。	2024 年 8 月 7 日
AWS 受管政策：AmazonRDSServiceRolePolicy – 更新現有政策	Amazon RDS 已從 `AWSServiceRoleForRDS` 服務連結角色的 `AmazonRDSServiceRolePolicy` 移除 `sns:Publish` 許可。如需更多詳細資訊，請參閱 AWS 受管政策：AmazonRDSServiceRolePolicy。	2024 年 7 月 2 日

變更	描述	日期
AWS Amazon RDS 的 受管政策 – 更新現有政策	Amazon RDS 將新的許可新增至 AWSServiceRoleForRDSCustom 服務連結角色的 AmazonRDSCustomServiceRolePolicy ，以允許 RDS Custom for SQL Server 修改基礎資料庫主機執行個體類型。RDS 也新增 ec2:DescribeInstanceTypes 許可，以取得資料庫主機的執行個體類型資訊。如需更多詳細資訊，請參閱 AWS Amazon RDS 的 受管政策。	2024 年 4 月 8 日
AWS Amazon RDS 的 受管政策 – 新政策	Amazon RDS 新增了名為 AmazonRDSCustomInstanceProfileRolePolicy 的新受管政策，以允許 RDS Custom 透過 EC2 執行個體設定檔執行自動化動作和資料庫管理任務。如需更多詳細資訊，請參閱 AWS Amazon RDS 的 受管政策。	2024 年 2 月 27 日
Amazon Aurora 的服務連結角色許可 – 更新現有政策	Amazon RDS 已將新陳述式 ID 新增至 AWSServiceRoleForRDS 服務連結角色的 AmazonRDSServiceRolePolicy 。 如需更多詳細資訊，請參閱 Amazon Aurora 的服務連結角色許可。	2024 年 1 月 19 日

變更	描述	日期
AWS Amazon RDS 的 受管政策 – 更新現有政策	AmazonRDSPerformanceInsightsReadOnly 和 AmazonRDSPerformanceInsightsFullAccess 受管政策現在包含 Sid (陳述式 ID) 作為政策陳述式中的識別符。 如需詳細資訊，請參閱 AWS 受管政策：AmazonRDSPerformanceInsightsReadOnly 和 AWS 受管政策：AmazonRDSPerformanceInsightsFullAccess	2023 年 10 月 23 日
AWS Amazon RDS 的 受管政策 – 更新現有政策	Amazon RDS 將新的許可新增到 AmazonRDSFullAccess 受管政策。這些許可允許您產生、檢視及刪除一段時間區間內的績效分析報告。 如需有關設定 Performance Insights 存取政策的詳細資訊，請參閱 設定績效詳情的存取政策	2023 年 8 月 17 日

變更	描述	日期
AWS Amazon RDS 的 受管政策 – 新政策和更新現有政策	Amazon RDS 新增了新的許可至 `AmazonRDSPerformanceInsightsReadOnly` 受管策略和名為 `AmazonRDSPerformanceInsightsFullAccess` 的新受管政策。這些許可允許您分析一段時間區間內的 Performance Insights、檢視分析結果及建議，以及刪除報告。 如需有關設定 Performance Insights 存取政策的詳細資訊，請參閱 設定績效詳情的存取政策	2023 年 8 月 16 日
AWS Amazon RDS 的 受管政策 – 更新現有政策	Amazon RDS 將新的 Amazon CloudWatch 命名空間 `ListMetrics` 新增至 `AmazonRDSFullAccess` 和 `AmazonRDSReadOnlyAccess` 。 Amazon RDS 需要此命名空間才能列出特定資源用量指標。 如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的 CloudWatch 資源存取許可管理概觀。	2023 年 4 月 4 日

變更	描述	日期
Amazon Aurora 的服務連結角色許可 – 更新現有政策	Amazon RDS 已將新許可新增至 `AWSServiceRoleForRDS` 服務連結角色的 `AmazonRDSServiceRolePolicy` ，以與 AWS Secrets Manager 整合。RDS 需要與 Secrets Manager 整合，才能在 Secrets Manager 中管理主要使用者密碼。密碼使用舊有的命名慣例並限制客戶更新。 如需更多詳細資訊，請參閱使用 Amazon Aurora 和 進行密碼管理 AWS Secrets Manager。	2022 年 12 月 22 日
AWS Amazon RDS 的 受管政策 – 更新現有政策	Amazon RDS 已將新許可新增至 `AmazonRDSFullAccess` 和 `AmazonRDSReadOnlyAccess` 受管政策，以允許您在 RDS 主控台中開啟 Amazon DevOps Guru。需要此許可才能檢查 DevOps Guru 是否已開啟。 如需更多詳細資訊，請參閱 設定 DevOps Guru for RDS 的 IAM 存取政策。	2022 年 12 月 19 日

變更	描述	日期
Amazon Aurora 的服務連結角色許可 – 更新現有政策	Amazon RDS 為 `PutMetricData` 將新的 Amazon CloudWatch 命名空間新增至 `AmazonRDSPreviewServiceRolePolicy` 。 Amazon RDS 需要此命名空間才能發佈資源用量指標。 如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的使用條件金鑰來限制對 CloudWatch 命名空間的存取。	2022 年 6 月 7 日
Amazon Aurora 的服務連結角色許可 – 更新現有政策	Amazon RDS 為 `PutMetricData` 將新的 Amazon CloudWatch 命名空間新增至 `AmazonRDSBetaServiceRolePolicy` 。 Amazon RDS 需要此命名空間才能發佈資源用量指標。 如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的使用條件金鑰來限制對 CloudWatch 命名空間的存取。	2022 年 6 月 7 日

變更	描述	日期
Amazon Aurora 的服務連結角色許可 – 更新現有政策	Amazon RDS 為 `PutMetricData` 將新的 Amazon CloudWatch 命名空間新增至 `AWSServiceRoleForRDS` 。 Amazon RDS 需要此命名空間才能發佈資源用量指標。 如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的使用條件金鑰來限制對 CloudWatch 命名空間的存取。	2022 年 4 月 22 日
AWS Amazon RDS 的 受管政策 – 新政策	Amazon RDS 新增名為 `AmazonRDSPerformanceInsightsReadOnly` 的新受管政策，以允許 Amazon RDS 代表您的資料庫執行個體呼叫 AWS 服務。 如需有關設定 Performance Insights 存取政策的詳細資訊，請參閱 設定績效詳情的存取政策	2022 年 3 月 10 日

變更	描述	日期
Amazon Aurora 的服務連結角色許可 – 更新現有政策	Amazon RDS 為 `PutMetricData` 將新的 Amazon CloudWatch 命名空間新增至 `AWSServiceRoleForRDS` 。 這些命名空間是 Amazon DocumentDB (具備 MongoDB 相容性) 和 Amazon Neptune 發佈 CloudWatch 指標所必需的。 如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的使用條件金鑰來限制對 CloudWatch 命名空間的存取。	2022 年 3 月 4 日
Amazon RDS 開始追蹤變更	Amazon RDS 開始追蹤其 AWS 受管政策的變更。	2021 年 10 月 26 日

防止跨服務混淆代理人問題

混淆代理人問題屬於安全性議題，其中沒有執行動作許可的實體可以強制具有更多許可的實體執行該動作。在 中AWS，跨服務模擬可能會導致混淆代理人問題。

在某個服務 (呼叫服務) 呼叫另一個服務 (被呼叫服務) 時，可能會發生跨服務模擬。可以操縱呼叫服務來使用其許可，以其不應有存取許可的方式對其他客戶的資源採取動作。為了避免這種情況， AWS提供的工具可協助您保護所有 服務的資料，其服務主體已獲得您帳戶中資源的存取權。如需詳細資訊，請參閱《IAM 使用者指南》中的混淆代理問題。

若要限制 Amazon RDS 為特定資源提供另一項服務的許可，建議在資源政策中使用 aws:SourceArn 和 aws:SourceAccount 全域條件內容索引鍵。

某些情況下，aws:SourceArn 值不包含帳戶 ID，例如使用 Amazon S3 儲存貯體的 Amazon 資源名稱 (ARN) 時。在這些情況下，請務必同時使用兩個全域條件內容索引鍵來限制許可。在某些情況下，可以同時使用全域條件內容索引鍵和包含帳號 ID 的 aws:SourceArn 值。在這些情況下，當在相同政策陳述式中使用 aws:SourceAccount 值和 aws:SourceArn 裡的帳戶時，請確認兩者使用同樣的帳戶 ID。如果您想要僅允許一個資源與跨服務存取相關聯，請使用 aws:SourceArn。如果您想要允許指定AWS帳戶中的任何資源與跨服務使用相關聯，請使用 aws:SourceAccount。

請確定 aws:SourceArn 的值是 Amazon RDS 資源類型的 ARN。如需詳細資訊，請參閱Amazon RDS 中的 Amazon Resource Name (ARN)。

防範混淆代理人問題的最有效方法是使用 aws:SourceArn 全域條件內容索引鍵，以及資源的完整 ARN。在某些情況下，您可能不知道資源的完整 ARN，或者指定了多個資源。在這些情況下，請針對 ARN 的未知部分，使用含有萬用字元 (*) 的全域條件內容索引鍵 (aws:SourceArn)。例如，arn:aws:rds:*:*123456789012*:*。

下列範例示範如何使用 Amazon RDS 中的 aws:SourceArn 和 aws:SourceAccount 全域條件內容索引鍵，來預防混淆代理人問題。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "rds.amazonaws.com"
```

```
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mydbinstance"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

如需更多使用 aws:SourceArn 和 aws:SourceAccount 全域條件內容索引鍵的政策範例，請參閱以下區段：

- 授予將通知發佈至 Amazon SNS 主題的許可
- 設定對 Amazon S3 儲存貯體的存取權 (PostgreSQL 匯入)
- 設定對 Amazon S3 儲存貯體的存取權 (PostgreSQL 匯出)

IAM 資料庫身分驗證

您可以使用 AWS Identity and Access Management (IAM) 資料庫身分驗證來驗證資料庫叢集。IAM 資料庫身分驗證可搭配 Aurora MySQL 和 Aurora PostgreSQL 運作。透過此身分驗證方法，您連線至資料庫叢集時不需要使用密碼。而是改用身分驗證字符。

身分驗證字符是 Amazon Aurora 依請求而產生的唯一字元字串。身分驗證字符是使用 AWS Signature 第 4 版產生。每個字符的存留期為 15 分鐘。您不需要將使用者登入資料存放在資料庫，因為身分驗證是利用 IAM 在外部管理。您仍可使用標準資料庫身分驗證。字符僅用於身分驗證，並且在建立後不會影響工作階段。

IAM 資料庫身分驗證提供下列優點：

- 使用 Secure Socket Layer (SSL) 或 Transport Layer Security (TLS) 來加密往返資料庫的網路流量。如需使用 SSL/TLS 搭配 Amazon Aurora 的詳細資訊，請參閱 使用 SSL/TLS 加密與資料庫叢集的連線。
- 您可以使用 IAM 來集中管理資料庫資源的存取，而不需要在每個資料庫叢集上個別地管理存取。
- 對於 Amazon EC2 上執行的應用程式，您可以使用 EC2 執行個體專用的設定檔登入資料 (而非密碼) 來存取資料庫，如此安全性更高。

通常，當應用程式每秒建立的連線少於 200 個，且您不希望直接以應用程式程式碼管理使用者名稱和密碼時，請考慮使用 IAM 資料庫身分驗證。

Amazon Web Services (AWS) JDBC 驅動程式支援 IAM 資料庫身分驗證。如需詳細資訊，請參閱 Amazon Web Services (AWS) JDBC 驅動程式 GitHub 儲存庫中的 AWS IAM 身分驗證外掛程式。

Amazon Web Services (AWS) Python 驅動程式支援 IAM 資料庫身分驗證。如需詳細資訊，請參閱 Amazon Web Services (AWS) Python 驅動程式 GitHub 儲存庫中的 AWS IAM 身分驗證外掛程式。

瀏覽下列主題，了解設定 IAM 進行資料庫身分驗證的程序：

- 啟用和停用 IAM 資料庫身分驗證
- 建立並使用 IAM 政策進行 IAM 資料庫存取
- 使用 IAM 身分驗證建立資料庫帳戶
- 使用 IAM 身分驗證連接至資料庫叢集

區域和版本可用性

功能可用性和支援會因每個 Aurora 資料庫引擎的特定版本以及 AWS 區域而有所不同。如需 Aurora 和 IAM 資料庫身分驗證的版本和區域可用性的詳細資訊，請參閱 IAM 資料庫身分驗證的支援區域和 Aurora 資料庫引擎。

對於 Aurora MySQL，所有支援的資料庫執行個體類別都支援 IAM 資料庫身分驗證，但 db.t2.small 和 db.t3.small 除外。如需有關支援的資料庫執行個體類別的資訊，請參閱 資料庫執行個體類別的支援資料庫引擎。

CLI 和開發套件支援

IAM 資料庫身分驗證適用於 AWS CLI和下列語言特定 AWS SDKs：

- 適用於 .NET 的 AWS SDK
- 適用於 C++ 的 AWS SDK
- 適用於 Go 的 AWS SDK
- 適用於 Java 的 AWS SDK
- 適用於 JavaScript 的 AWS SDK
- 適用於 PHP 的 AWS SDK
- 適用於 Python (Boto3) 的 AWS SDK
- 適用於 Ruby 的 AWS SDK

IAM 資料庫身分驗證的限制

使用 IAM 資料庫身分驗證，會套用以下限制：

- 目前，IAM 資料庫驗證不支援所有全域條件內容金鑰。

  如需有關全域條件內容索引鍵的詳細資訊，請參閱《IAM 使用者指南》中的 AWS 全域條件內容索引鍵。
- 對於 PostgreSQL，如果 IAM 角色 (`rds_iam`) 新增至使用者 (包括 RDS 主要使用者)，則 IAM 身分驗證優先於密碼身分驗證，因此使用者必須以 IAM 使用者身分登入。
- 您無法使用自訂 Route 53 DNS 記錄，替代資料庫叢集端點來產生身分驗證字符。
- CloudWatch 和 CloudTrail 不會記錄 IAM 身分驗證。這些服務不會追蹤授權 IAM 角色啟用資料庫連線的 `generate-db-auth-token` API 呼叫。

- IAM 資料庫身分驗證需要資料庫叢集上的運算資源。您的資料庫必須擁有 300 到 1000 MiB 的額外記憶體，才能提供可靠的連線能力。若要查看工作負載所需的記憶體，請在啟用 IAM 資料庫身分驗證前後，比較增強型監控處理清單中 RDS 程序的 RES 欄。請參閱 在 RDS 主控台中檢視作業系統指標。

  如果您使用的是爆量類別執行個體，請減少緩衝區和快取等其他參數所使用的記憶體數量，以避免記憶體不足。
- 對於 Aurora MySQL，您無法針對使用 IAM 身分驗證設定的資料庫使用者使用密碼型身分驗證。
- RDS on Outposts 不支援任何引擎的 IAM 資料庫身分驗證。

適用於 IAM 資料庫身分驗證的建議

使用 IAM 資料庫身分驗證時，建議您採取以下做法：

- 當您的應用程式每秒需要少於 200 個新的 IAM 資料庫驗證連線時，請使用 IAM 資料庫驗證。

  搭配 Amazon Aurora 使用的資料庫引擎完全不限制每秒的身分驗證嘗試次數。不過，當您使用 IAM 資料庫身分驗證時，應用程式必須產生身分驗證字符。然後，應用程式會使用該字符來連接至資料庫叢集。如果您超過每秒新連線數上限，IAM 資料庫身分驗證的額外負荷會導致連線調節。

  請考慮在應用程式中使用連線集區，以緩解持續建立連線的問題。如此可以減少 IAM 資料庫身分驗證的額外負荷，並允許您的應用程式重複使用現有連線。或者，請考慮針對這些使用案例採用 RDS Proxy。RDS Proxy需額外收費。請參閱 RDS Proxy 定價。
- IAM 資料庫身分驗證字符的大小取決於許多因素，包括 IAM 標籤的數量、IAM 服務政策、ARN 長度，以及其他 IAM 和資料庫屬性。此字符的最小大小一般約為 1 KB，但可以更大。因為此字符用作連線字串中的密碼，以使用 IAM 身分驗證連線至資料庫，所以您應該確保資料庫驅動程式 (例如 ODBC) 和/或任何工具不會限制或由於其大小而以其他方式截斷此字符。截斷的字符將導致資料庫和 IAM 執行的身分驗證失敗。
- 如果您是在建立 IAM 資料庫身分驗證字符時使用臨時憑證，則在使用 IAM 資料庫身分驗證字符發出連線請求時，臨時憑證必須仍然有效。

不支援的 AWS 全域條件內容索引鍵

IAM 資料庫身分驗證不支援下列 AWS 全域條件內容金鑰子集。

- `aws:Referer`
- `aws:SourceIp`

- aws:SourceVpc
- aws:SourceVpce
- aws:UserAgent
- aws:VpcSourceIp

如需詳細資訊，請參閱《IAM 使用者指南》中的AWS 全域條件內容金鑰。

啟用和停用 IAM 資料庫身分驗證

依預設，資料庫叢集上會停用 IAM 資料庫身分驗證。您可以使用 AWS 管理主控台 AWS CLI或 API 啟用或停用 IAM 資料庫身分驗證。

您可以在執行下列其中一個動作時啟用 IAM 資料庫身分驗證：

- 若要在啟用 IAM 資料庫身分驗證的情況下建立新資料庫叢集，請參閱建立 Amazon Aurora 資料庫叢集。
- 若要修改資料庫叢集以啟用 IAM 資料庫身分驗證，請參閱修改 Amazon Aurora 資料庫叢集。
- 如要從啟用 IAM 資料庫身分驗證的快照還原資料庫叢集，請參閱 從資料庫叢集快照還原。
- 若要在啟用 IAM 資料庫身分驗證的情況下，將資料庫執行個體叢集還原至某個時間點，請參閱 將資料庫叢集還原至指定時間。

主控台

每個建立或修改工作流程都有一個 Database authentication (資料庫身分驗證) 區段，您可以在其中啟用或停用 IAM 資料庫身分驗證。在該區段中，選擇 Password and IAM database authentication (密碼和 IAM 資料庫身分驗證) 以啟用 IAM 資料庫身分驗證。

為現有資料庫叢集啟用或停用 IAM 資料庫身分驗證

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中，選擇 Databases (資料庫)。
3. 選擇想要修改的資料庫叢集。

> **Note**
>
> 唯有資料庫叢集中的所有資料庫執行個體皆相容於 IAM，您才能啟用 IAM 身分驗證。查看區域和版本可用性 中的相容性要求。

4. 選擇 Modify (修改)。
5. 在資料庫身分驗證區段中，選擇 IAM 資料庫身分驗證以啟用 IAM 資料庫身分驗證。選擇密碼身分驗證或密碼和 Kerberos 身分驗證以停用 IAM 身分驗證。
6. 您也可以選擇啟用將 IAM 資料庫身分驗證日誌發佈至 CloudWatch Logs。在日誌匯出下，選擇 iam-db-auth-error 日誌選項。將日誌發佈到 CloudWatch Logs 會消耗儲存空間，而且您需要支付該儲存空間的費用。請務必刪除您不再需要的任何 CloudWatch Logs。
7. 選擇 Continue (繼續)。
8. 若要立即套用變更，請在 Scheduling of modifications (修改排程) 區段中選擇 Immediately (立即)。
9. 選擇 Modify cluster (修改叢集)。

AWS CLI

若要使用 建立具有 IAM 身分驗證的新資料庫叢集 AWS CLI，請使用 `create-db-cluster`命令。指定 `--enable-iam-database-authentication` 選項。

若要將現有資料庫叢集更新為進行或不進行 IAM 身分驗證，請使用 AWS CLI 命令 `modify-db-cluster`。視需要指定 `--enable-iam-database-authentication` 或 `--no-enable-iam-database-authentication` 選項。

> **Note**
>
> 唯有資料庫叢集中的所有資料庫執行個體皆相容於 IAM，您才能啟用 IAM 身分驗證。查看 區域和版本可用性 中的相容性要求。

依預設，Aurora 會在下一次維護時段執行修改。如果您不想等待，而希望儘快啟用 IAM 資料庫身分驗證，請使用 `--apply-immediately` 參數。

如果您要還原資料庫叢集，請使用下列其中一個 AWS CLI 命令：

- `restore-db-cluster-to-point-in-time`
- `restore-db-cluster-from-db-snapshot`

IAM 資料庫身分驗證設定預設為來源快照的設定。若要變更此設定，請視需要設定 `--enable-iam-database-authentication` 或 `--no-enable-iam-database-authentication` 選項。

RDS API

若要使用 API 建立支援 IAM 身分驗證的新資料庫執行個體，請使用 API 操作 `CreateDBCluster`。將 `EnableIAMDatabaseAuthentication` 參數設為 `true`。

若要更新現有資料庫叢集進行或不進行 IAM 身分驗證，請使用 API 操作 `ModifyDBCluster`。將 `EnableIAMDatabaseAuthentication` 參數設為 `true` 以啟用 IAM 身分驗證，或設為 `false` 表示停用。

> **Note**
>
> 唯有資料庫叢集中的所有資料庫執行個體皆相容於 IAM，您才能啟用 IAM 身分驗證。查看 區域和版本可用性 中的相容性要求。

如果您要還原資料庫叢集，請使用下列其中一個 API 操作：

- `RestoreDBClusterFromSnapshot`
- `RestoreDBClusterToPointInTime`

IAM 資料庫身分驗證設定預設為來源快照的設定。若要變更此設定，請將 `EnableIAMDatabaseAuthentication` 參數設為 `true` 以啟用 IAM 身分驗證，或設為 `false` 表示停用。

建立並使用 IAM 政策進行 IAM 資料庫存取

若要允許使用者或角色連接資料庫叢集，您必須建立 IAM 政策。然後，您可以將政策連接到許可集或角色。

> **Note**
>
> 若要進一步了解 IAM 政策，請參閱Amazon Aurora 的 Identity and access management。

以下範例政策可讓使用者利用 IAM 資料庫身分驗證來連接資料庫叢集。

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-2:111122223333:dbuser:cluster-
ABCDEFGHIJKL01234/db_user"
            ]
        }
    ]
}
```

Important

具有管理員許可的使用者可以存取資料庫叢集，而不需 IAM 政策中的明確許可。如果您想要將管理員的存取權限制為資料庫叢集，您可以建立具有適當、特殊權限較低的許可的 IAM 角色，並將其指派給管理員。

Note

請勿混淆 `rds-db:` 前綴和其他以 `rds:` 開頭的 RDS API 操作前綴。您只能對 IAM 資料庫身分驗證使用 `rds-db:` 字首和 `rds-db:connect` 動作。它們不適用於任何其他內容。

範例政策包含單一陳述式與下列元素：

- `Effect` – 指定 `Allow` 以授權存取資料庫叢集。如果您未明確允許存取，預設將會拒絕存取。
- `Action` – 指定 `rds-db:connect` 以允許連線至資料庫叢集。
- `Resource` – 指定 Amazon Resource Name (ARN) 以描述一個資料庫叢集中的一個資料庫帳戶。ARN 格式如下。

```
arn:aws:rds-db:region:account-id:dbuser:DbClusterResourceId/db-user-name
```

請在此格式中更換下列項目：

- *region* 是資料庫叢集 AWS 的區域。在範例政策中， AWS 區域為 us-east-2。
- *account-id* 是資料庫叢集的 AWS 帳號。在此範例政策中，帳戶號碼是 1234567890。使用者的帳戶必須與資料庫叢集的帳戶相同。

  若要執行跨帳户存取權，請使用資料庫叢集帳戶中，上述的政策來建立 IAM 角色，並允許您的其他帳戶擔任該角色。
- *DbClusterResourceId* 是資料庫叢集的識別符。此識別符對 AWS 區域是唯一的，絕不會變更。在此範例政策中，識別符是 cluster-ABCDEFGHIJKL01234。

  若要在適用於 Amazon Aurora AWS 管理主控台 的 中尋找資料庫叢集資源 ID，請選擇資料庫叢集以查看其詳細資訊。然後選擇 Configuration (組態) 標籤。Resource ID (資源 ID) 顯示在 Configuration (組態) 區段中。

  或者，您可以使用 AWS CLI 命令列出目前 AWS 區域中所有資料庫叢集的識別符和資源 IDs，如下所示。

  ```
  aws rds describe-db-clusters --query "DBClusters[*].
  [DBClusterIdentifier,DbClusterResourceId]"
  ```

  > **Note**
  >
  > 如果您是透過 RDS Proxy 連線至資料庫，請指定代理資源 ID，例如 prx-ABCDEFGHIJKL01234。如需搭配 RDS Proxy 使用 IAM 資料庫驗證的相關資訊，請參閱使用 IAM 身分驗證連線至資料庫。
- *db-user-name* 是要與 IAM 身分驗證產生關聯的資料庫帳戶的名稱。在範例政策中，資料庫帳戶是 db_user。

您可以建構其他 ARN 來支援各種存取模式。下列政策允許存取資料庫叢集中的兩個不同資料庫帳戶：

JSON

```
{
   "Version":"2012-10-17",
```

```
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-
ABCDEFGHIJKL01234/jane_doe",
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-
ABCDEFGHIJKL01234/mary_roe"
         ]
      }
   ]
}
```

下列政策使用「*」字元來比對特定 AWS 帳戶和 AWS 區域的所有資料庫叢集和資料庫帳戶。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-2:111122223333:dbuser:*/*"
            ]
        }
    ]
}
```

下列政策符合特定 AWS 帳戶和 AWS 區域的所有資料庫叢集。不過，此政策僅授權存取具有 `jane_doe` 資料庫帳戶的資料庫叢集。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:*/jane_doe"
         ]
      }
   ]
}
```

使用者或角色只能存取資料庫使用者可存取的資料庫。例如，假設資料庫叢集有一個名為 dev 的資料庫，還有另一個名為 test 的資料庫。如果資料庫使用者 `jane_doe` 只能存取 dev，則任何以 `jane_doe` 使用者存取該資料庫叢集的使用者或角色，也只能存取 dev。其他資料庫物件同樣受此存取限制，例如資料表和檢視等。

管理員必須建立 IAM 政策，授予實體在其所需的指定資源上執行特定 API 操作的許可。管理員接著必須將這些政策連接至需要這些許可的許可集或角色。如需政策範例，請參閱 Amazon Aurora 以身分為基礎的政策範例。

將政 IAM 政策連接到許可集或角色

在您建立 IAM 政策以允許資料庫身分驗證之後，您必須將該政策連接至許可集或角色。如需本主題的教學課程，請參閱《IAM 使用者指南》中的建立並連接您的第一個客戶受管原則。

在您進行教學課程時，可使用本節所示的其中一個政策範例做為起點，並依您的需求進行自訂。在教學課程結束時，您會有一個具有連接政策且可使用 `rds-db:connect` 動作的許可集。

> **Note**
>
> 您可以將多個許可集或角色映射至相同的資料庫使用者帳戶。例如，假設 IAM 政策指定下列資源 ARN。

```
arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-12ABC34DEFG5HIJ6KLMNOP78QR/
jane_doe
```

如果您將此政策連接至 Jane、Bob 和 Diego，則每個使用者都能使用 jane_doe 資料庫帳戶連線至指定的資料庫叢集。

使用 IAM 身分驗證建立資料庫帳戶

採用 IAM 資料庫身分驗證時，不需要指派資料庫密碼給您所建立的使用者帳戶。如果您移除已映射至資料庫帳戶的使用者，則應該使用 DROP USER 陳述式來一併移除該資料庫帳戶。

> **Note**
>
> 用於 IAM 身分驗證的使用者名稱必須與資料庫中的使用者名稱大小寫相符。

主題

- 搭配 IAM 身分驗證使用 Aurora MySQL
- 搭配 AuroraPostgreSQL 使用 IAM 身分驗證

搭配 IAM 身分驗證使用 Aurora MySQL

使用 Aurora MySQL 時，身分驗證是由 AWS處理AWSAuthenticationPlugin，這是與 IAM 無縫搭配運作的外掛程式，可驗證您的使用者。以主要使用者身分或可以建立使用者並授予權限的不同使用者身分連線至資料庫叢集。連線後，發出 CREATE USER 陳述式，如下列範例所示。

```
CREATE USER 'jane_doe' IDENTIFIED WITH AWSAuthenticationPlugin AS 'RDS';
```

IDENTIFIED WITH 子句可讓 Aurora MySQL 使用 AWSAuthenticationPlugin 來驗證資料庫帳戶 (jane_doe)。AS 'RDS' 子句指的是身分驗證方法。確定指定的資料庫使用者名稱與 IAM 資料庫存取的 IAM 政策中的資源相同。如需詳細資訊，請參閱建立並使用 IAM 政策進行 IAM 資料庫存取。

> **Note**
>
> 如果您看到以下訊息，表示 AWS提供的外掛程式不適用於目前的資料庫叢集。
> `ERROR 1524 (HY000): Plugin 'AWSAuthenticationPlugin' is not loaded`
> 若要對此錯誤進行故障診斷，請確認您使用支援的組態，且已在資料庫叢集上啟用 IAM 資料庫身分驗證。如需更多詳細資訊，請參閱 區域和版本可用性 及 啟用和停用 IAM 資料庫身分驗證。

使用 `AWSAuthenticationPlugin` 建立帳戶之後，管理此帳戶的方式就像管理其他資料庫帳戶一樣。例如，您可以使用 `GRANT` 和 `REVOKE` 陳述式來修改帳戶權限，或使用 `ALTER USER` 陳述式來修改各種帳戶屬性。

使用 IAM 時，資料庫網路流量會使用 SSL/TLS 加密。若要允許 SSL 連線，請透過下列命令修改使用者帳戶。

```
ALTER USER 'jane_doe'@'%' REQUIRE SSL;
```

搭配 AuroraPostgreSQL 使用 IAM 身分驗證

若要搭配 Aurora PostgreSQL 使用 IAM 身分驗證，請以主要使用者身分或可以建立使用者並授予權限的不同使用者身分連線至資料庫叢集。連線後，請建立資料庫使用者，然後將 `rds_iam` 角色授予他們，如下列範例所示。

```
CREATE USER db_userx;
GRANT rds_iam TO db_userx;
```

確定指定的資料庫使用者名稱與 IAM 資料庫存取的 IAM 政策中的資源相同。如需詳細資訊，請參閱建立並使用 IAM 政策進行 IAM 資料庫存取。您必須授予 `rds_iam` 角色才能使用 IAM 身分驗證。您也可以使用巢狀成員資格或角色的間接授予。

請注意，PostgreSQL 資料庫使用者可以使用 IAM 或 Kerberos 身分驗證，但不能同時使用兩者，所以該使用者也不能使用 `rds_ad` 角色。這也適用於巢狀成員資格。如需詳細資訊，請參閱步驟 7：針對您的 Kerberos 主體建立 PostgreSQL 使用者。

使用 IAM 身分驗證連接至資料庫叢集

採用 IAM 資料庫身分驗證時，您會使用身分驗證字符來連接資料庫叢集。身分驗證字符是可用來代替密碼的字元字串。產生身分驗證字符之後，過期之前的有效期限為 15 分鐘。如果您嘗試使用過期的字符來連接，則會拒絕連接請求。

每個身分驗證字符必須附帶有效的簽章，並使用 AWS 簽章第 4 版。(如需詳細資訊，請參閱 AWS 一般參考 中的 Signature 第 4 版簽署程序。) AWS CLI 及 AWS SDK (例如 適用於 Java 的 AWS SDK 或 適用於 Python (Boto3) 的 AWS SDK) 可以自動簽署您建立的每個字符。

當您從另一項 AWS 服務 (例如 AWS Lambda) 連線至 Amazon Aurora 時，您可以使用身分驗證字符。使用字符就可以避免將密碼寫在程式碼中。或是您可使用 AWS 開發套件，以程式設計方式建立及簽署身分驗證字符。

取得已簽署的 IAM 身分驗證字符之後，您就可以連接至 Aurora 資料庫叢集。接下來，您可以了解如何使用命令列工具或 AWS 開發套件來這樣做，例如 適用於 Java 的 AWS SDK 或 適用於 Python (Boto3) 的 AWS SDK。

如需詳細資訊，請參閱下列部落格文章：

- 使用 IAM 身分驗證將 SQL Workbench/J 連線至 Aurora MySQL 或 Amazon RDS for MySQL
- Using IAM authentication to connect with pgAdmin Amazon Aurora PostgreSQL or Amazon RDS for PostgreSQL

先決條件

以下是使用 IAM 身分驗證連線至資料庫叢集的先決條件：

- 啟用和停用 IAM 資料庫身分驗證
- 建立並使用 IAM 政策進行 IAM 資料庫存取
- 使用 IAM 身分驗證建立資料庫帳戶

主題

- 使用 IAM 身分驗證和 AWS 驅動程式連線至資料庫叢集
- 從命令列 AWS CLI 和 mysql 用戶端使用 IAM 身分驗證連線至資料庫叢集
- 從命令列： AWS CLI 和 psql 用戶端使用 IAM 身分驗證連線至資料庫叢集
- 使用 IAM 身分驗證和 連線至資料庫叢集 適用於 .NET 的 AWS SDK
- 使用 IAM 身分驗證和 連線至資料庫叢集 適用於 Go 的 AWS SDK

- 使用 IAM 身分驗證和 連線至資料庫叢集 適用於 Java 的 AWS SDK
- 使用 IAM 身分驗證和 連線至資料庫叢集 適用於 Python (Boto3) 的 AWS SDK

使用 IAM 身分驗證和 AWS 驅動程式連線至資料庫叢集

驅動程式 AWS 套件旨在支援更快的轉換和容錯移轉時間，以及使用 AWS Secrets Manager、AWS Identity and Access Management (IAM) 和聯合身分進行身分驗證。AWS 驅動程式依賴監控資料庫叢集狀態，並注意叢集拓撲以判斷新的寫入器。相較於開放原始碼驅動程式的數十秒，此方法可將轉換和容錯移轉時間縮短為短短幾秒。

如需 AWS 驅動程式的詳細資訊，請參閱 Aurora MySQL 或 Aurora PostgreSQL 資料庫叢集的對應語言驅動程式。

從命令列 AWS CLI 和 mysql 用戶端使用 IAM 身分驗證連線至資料庫叢集

您可以使用 AWS CLI 和 `mysql`命令列工具，從命令列連線至 Aurora 資料庫叢集，如下所述。

先決條件

以下是使用 IAM 身分驗證連線至資料庫叢集的先決條件：

- 啟用和停用 IAM 資料庫身分驗證
- 建立並使用 IAM 政策進行 IAM 資料庫存取
- 使用 IAM 身分驗證建立資料庫帳戶

> **Note**
>
> 有關使用 SQL Workbench/J 與 IAM 身分驗證連接到您的資料庫的資訊，請參閱部落格文章 使用 IAM 身分驗證將 SQL Workbench/J 連線至 Aurora MySQL 或 Amazon RDS for MySQL。

主題

- 產生 IAM 身分驗證字符
- 連接至資料庫叢集

產生 IAM 身分驗證字符

以下範例顯示如何使用 AWS CLI來取得已簽署的身分驗證字符。

```
aws rds generate-db-auth-token \
   --hostname rdsmysql.123456789012.us-west-2.rds.amazonaws.com \
   --port 3306 \
   --region us-west-2 \
   --username jane_doe
```

範例中的參數如下：

- `--hostname` – 您想要存取之資料庫叢集的主機名稱
- `--port` – 用於連線資料庫叢集的連接埠號碼
- `--region` – 資料庫叢集執行所在的 AWS 區域
- `--username` – 您想要存取的資料庫帳戶

字符開頭的前幾個字元看起來如下所示。

```
rdsmysql.123456789012.us-west-2.rds.amazonaws.com:3306/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
>
> 您無法使用自訂 Route 53 DNS 記錄，替代資料庫叢集端點來產生身分驗證字符。

連接至資料庫叢集

連線的一般格式如下所示。

```
mysql --host=hostName --port=portNumber --ssl-ca=full_path_to_ssl_certificate --enable-
cleartext-plugin --user=userName --password=authToken
```

參數如下：

- `--host` – 您想要存取之資料庫叢集的主機名稱
- `--port` – 用於連線資料庫叢集的連接埠號碼
- `--ssl-ca` – 包含公有金鑰之 SSL 憑證檔案的完整路徑

  如需詳細資訊，請參閱Aurora MySQL 資料庫叢集的 TLS 連線。

若要下載 SSL 憑證，請參閱使用 SSL/TLS 加密與資料庫叢集的連線。

- `--enable-cleartext-plugin` – 指定 `AWSAuthenticationPlugin` 必須用於此連線的值

  如果您使用的是 MariaDB 用戶端，該 `--enable-cleartext-plugin` 選項不是必需的。
- `--user` – 您想要存取的資料庫帳戶
- `--password` – 已簽署的 IAM 身分驗證字符

身分驗證字符由數百個字元組成。在命令列上可能會顯得雜亂。此問題的一個解決方法是將字符儲存到環境變數，然後在連接時使用該變數。以下範例顯示此解決方法的一種型態。在範例中，*/sample_dir/* 是包含公有金鑰之 SSL 憑證檔案的完整路徑。

```
RDSHOST="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
TOKEN="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 3306 --region us-
west-2 --username jane_doe )"

mysql --host=$RDSHOST --port=3306 --ssl-ca=/sample_dir/global-bundle.pem --enable-
cleartext-plugin --user=jane_doe --password=$TOKEN
```

使用 `AWSAuthenticationPlugin` 來連接時會以 SSL 保護連線。若要確認是否如此，請在 `mysql>` 命令提示中輸入下列命令。

```
show status like 'Ssl%';
```

輸出中的以下幾行顯示更多詳細資訊。

```
+---------------+-------------+
| Variable_name | Value

                                                                    |

+---------------+-------------+
| ...           | ...
| Ssl_cipher    | AES256-SHA

                                                                    |

| ...           | ...
| Ssl_version   | TLSv1.1

                                                                    |
```

```
| ...           | ...
+-----------------------------+
```

如果要透過 Proxy 連線到資料庫叢集，請參閱 使用 IAM 身分驗證連線至資料庫。

從命令列： AWS CLI 和 psql 用戶端使用 IAM 身分驗證連線至資料庫叢集

您可以使用 AWS CLI 和 psql 命令列工具，從命令列連線至 Aurora PostgreSQL 資料庫叢集，如下所述。

先決條件

以下是使用 IAM 身分驗證連線至資料庫叢集的先決條件：

- 啟用和停用 IAM 資料庫身分驗證
- 建立並使用 IAM 政策進行 IAM 資料庫存取
- 使用 IAM 身分驗證建立資料庫帳戶

> **Note**
>
> 如需有關使用 pgAdmin 搭配 IAM 身分驗證連線到資料庫的資訊，請參閱部落格文章 使用 IAM 身分驗證與 pgAdmin Amazon Aurora PostgreSQL 或 Amazon RDS for PostgreSQL 連線。

主題

- 產生 IAM 身分驗證字符
- 連接至 Aurora PostgreSQL 叢集

產生 IAM 身分驗證字符

身分驗證字符由數百個字元組成，因此在命令列上可能會顯得雜亂。此問題的一個解決方法是將字符儲存到環境變數，然後在連接時使用該變數。下列範例示範如何使用 AWS CLI 來取得使用 `generate-db-auth-token`命令簽署的身分驗證字符，並將其存放在`PGPASSWORD`環境變數中。

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"
```

在範例中，`generate-db-auth-token` 命令的參數如下所示：

- `--hostname` – 您想要存取之資料庫叢集 (叢集端點) 的主機名稱
- `--port` – 用於連線資料庫叢集的連接埠號碼
- `--region` – 資料庫叢集執行所在的 AWS 區域
- `--username` – 您想要存取的資料庫帳戶

所產生字符開頭的前幾個字元看起來如下所示。

```
mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com:5432/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
>
> 您無法使用自訂 Route 53 DNS 記錄，替代資料庫叢集端點來產生身分驗證字符。

連接至 Aurora PostgreSQL 叢集

使用 psql 連線的一般格式如下所示。

```
psql "host=hostName port=portNumber sslmode=verify-full
 sslrootcert=full_path_to_ssl_certificate dbname=DBName user=userName
 password=authToken"
```

參數如下：

- `host` – 您想要存取之資料庫叢集 (叢集端點) 的主機名稱
- `port` – 用於連線資料庫叢集的連接埠號碼
- `sslmode` – 要使用的 SSL 模式

  當您使用 `sslmode=verify-full` 時，SSL 連線會根據 SSL 憑證中的端點來驗證資料庫叢集端點。
- `sslrootcert` – 包含公有金鑰之 SSL 憑證檔案的完整路徑

  如需詳細資訊，請參閱使用 SSL/TLS 保護 Aurora PostgreSQL 資料的安全。

  若要下載 SSL 憑證，請參閱使用 SSL/TLS 加密與資料庫叢集的連線。
- `dbname` – 您想要存取的資料庫

- `user` – 您想要存取的資料庫帳戶
- `password` – 已簽署的 IAM 身分驗證字符

> **Note**
>
> 您無法使用自訂 Route 53 DNS 記錄，替代資料庫叢集端點來產生身分驗證字符。

以下範例示範使用 psql 進行連線。在範例中，psql 針對主機使用環境變數 RDSHOST，產生的字符則使用環境變數 PGPASSWORD。此外，*/sample_dir/* 是包含公有金鑰之 SSL 憑證檔案的完整路徑。

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"

psql "host=$RDSHOST port=5432 sslmode=verify-full sslrootcert=/sample_dir/global-
bundle.pem dbname=DBName user=jane_doe password=$PGPASSWORD"
```

如果要透過 Proxy 連線到資料庫叢集，請參閱 使用 IAM 身分驗證連線至資料庫。

使用 IAM 身分驗證和 連線至資料庫叢集 適用於 .NET 的 AWS SDK

您可以使用 連線到 Aurora MySQL 或 Aurora PostgreSQL 資料庫叢集 適用於 .NET 的 AWS SDK ，如下所述。

先決條件

以下是使用 IAM 身分驗證連線至資料庫叢集的先決條件：

- 啟用和停用 IAM 資料庫身分驗證
- 建立並使用 IAM 政策進行 IAM 資料庫存取
- 使用 IAM 身分驗證建立資料庫帳戶

範例

以下程式碼範例顯示如何產生身分驗證字符，然後用來連線至資料庫叢集。

若要執行此程式碼範例，您需要在 AWS 網站上適用於 .NET 的 AWS SDK找到的 。`AWSSDK.CORE` 和 `AWSSDK.RDS` 套件是必需的。若要連接到資料庫叢集，請使用資料庫引擎的 .NET 資料庫連接器，例如 MariaDB 或 MySQL 的 MySqlConnector 或 PostgreSQL 的 Npgsql。

此程式碼連接到一個 Aurora MySQL 資料庫叢集。視需要修改下列變數的值：

- `server` – 您想要存取之資料庫叢集的端點
- `user` – 您想要存取的資料庫帳戶
- `database` – 您想要存取的資料庫
- `port` – 用於連線資料庫叢集的連接埠號碼
- `SslMode` – 要使用的 SSL 模式

  當您使用 `SslMode=Required` 時，SSL 連線會根據 SSL 憑證中的端點來驗證資料庫叢集端點。
- `SslCa` – Amazon Aurora 之 SSL 憑證的完整路徑

  若要下載憑證，請參閱使用 SSL/TLS 加密與資料庫叢集的連線。

> **Note**
>
> 您無法使用自訂 Route 53 DNS 記錄，替代資料庫叢集端點來產生身分驗證字符。

```
using System;
using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;
using Amazon;

namespace ubuntu
{
  class Program
  {
    static void Main(string[] args)
    {
      var pwd =
 Amazon.RDS.Util.RDSAuthTokenGenerator.GenerateAuthToken(RegionEndpoint.USEast1,
 "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com", 3306, "jane_doe");
      // for debug only Console.Write("{0}\n", pwd);  //this verifies the token is
 generated

      MySqlConnection conn = new
 MySqlConnection($"server=mysqlcluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;user=jane_doe;database=mydB;port=3306;password={pwd};SslMode=Required;
```

```
      conn.Open();

      // Define a query
      MySqlCommand sampleCommand = new MySqlCommand("SHOW DATABASES;", conn);

      // Execute a query
      MySqlDataReader mysqlDataRdr = sampleCommand.ExecuteReader();

      // Read all rows and output the first column in each row
      while (mysqlDataRdr.Read())
        Console.WriteLine(mysqlDataRdr[0]);

      mysqlDataRdr.Close();
      // Close connection
      conn.Close();
    }
  }
}
```

此程式碼連接到一個 Aurora PostgreSQL 資料庫叢集。

視需要修改下列變數的值：

- `Server` – 您想要存取之資料庫叢集的端點
- `User ID` – 您想要存取的資料庫帳戶
- `Database` – 您想要存取的資料庫
- `Port` – 用於連線資料庫叢集的連接埠號碼
- `SSL Mode` – 要使用的 SSL 模式

  當您使用 `SSL Mode=Required` 時，SSL 連線會根據 SSL 憑證中的端點來驗證資料庫叢集端點。
- `Root Certificate` – Amazon Aurora 之 SSL 憑證的完整路徑

  若要下載憑證，請參閱使用 SSL/TLS 加密與資料庫叢集的連線。

> **Note**
>
> 您無法使用自訂 Route 53 DNS 記錄，替代資料庫叢集端點來產生身分驗證字符。

```
using System;
using Npgsql;
using Amazon.RDS.Util;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            var pwd =
 RDSAuthTokenGenerator.GenerateAuthToken("postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com", 5432, "jane_doe");
// for debug only Console.Write("{0}\n", pwd);  //this verifies the token is generated

            NpgsqlConnection conn = new
 NpgsqlConnection($"Server=postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;User Id=jane_doe;Password={pwd};Database=mydb;SSL
 Mode=Require;Root Certificate=full_path_to_ssl_certificate");
            conn.Open();

            // Define a query
                   NpgsqlCommand cmd = new NpgsqlCommand("select count(*) FROM
 pg_user", conn);

            // Execute a query
            NpgsqlDataReader dr = cmd.ExecuteReader();

            // Read all rows and output the first column in each row
            while (dr.Read())
                Console.Write("{0}\n", dr[0]);

            // Close connection
            conn.Close();
        }
    }
}
```

如果要透過 Proxy 連線到資料庫叢集，請參閱 使用 IAM 身分驗證連線至資料庫。

使用 IAM 身分驗證和 連線至資料庫叢集 適用於 Go 的 AWS SDK

您可以使用 連線到 Aurora MySQL 或 Aurora PostgreSQL 資料庫叢集 適用於 Go 的 AWS SDK ，如下所述。

先決條件

以下是使用 IAM 身分驗證連線至資料庫叢集的先決條件：

- 啟用和停用 IAM 資料庫身分驗證
- 建立並使用 IAM 政策進行 IAM 資料庫存取
- 使用 IAM 身分驗證建立資料庫帳戶

範例

若要執行這些程式碼範例，您需要在 AWS 網站上適用於 Go 的 AWS SDK找到的 。

視需要修改下列變數的值：

- `dbName` – 您想要存取的資料庫
- `dbUser` – 您想要存取的資料庫帳戶
- `dbHost` – 您想要存取之資料庫叢集的端點

> **Note**
>
> 您無法使用自訂 Route 53 DNS 記錄，替代資料庫叢集端點來產生身分驗證字符。

- `dbPort` – 用於連線資料庫叢集的連接埠號碼
- `region` – 資料庫叢集執行所在的 AWS 區域

此外，請確定範例程式碼中匯入的程式庫存在於您的系統上。

> **Important**
>
> 本節中的範例使用下列程式碼來提供從本機環境存取資料庫的登入資料：
> `creds := credentials.NewEnvCredentials()`
> 如果您從 AWS 服務存取資料庫，例如 Amazon EC2 或 Amazon ECS，您可以將程式碼取代為下列程式碼：
> `sess := session.Must(session.NewSession())`

```
    creds := sess.Config.Credentials
    如果您進行此變更，請確定您新增了下列匯入：
    "github.com/aws/aws-sdk-go/aws/session"
```

主題

- 使用 IAM 身分驗證和 適用於 Go 的 AWS SDK V2 進行連線
- 使用 IAM 身分驗證和 適用於 Go 的 AWS SDK V1 進行連線。

使用 IAM 身分驗證和 適用於 Go 的 AWS SDK V2 進行連線

您可以使用 IAM 身分驗證和 適用於 Go 的 AWS SDK V2 連線到資料庫叢集。

以下程式碼範例顯示如何產生身分驗證字符，然後用來連線至資料庫叢集。

此程式碼連接到一個 Aurora MySQL 資料庫叢集。

```
package main

import (
     "context"
     "database/sql"
     "fmt"

     "github.com/aws/aws-sdk-go-v2/config"
     "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
     _ "github.com/go-sql-driver/mysql"
)

func main() {

     var dbName string = "DatabaseName"
     var dbUser string = "DatabaseUser"
     var dbHost string = "mysqlcluster.cluster-123456789012.us-
east-1.rds.amazonaws.com"
     var dbPort int = 3306
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
```

```
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authenticationToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

此程式碼連接到一個 Aurora PostgreSQL 資料庫叢集。

```
package main

import (
     "context"
     "database/sql"
     "fmt"

     "github.com/aws/aws-sdk-go-v2/config"
     "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
     _ "github.com/lib/pq"
)

func main() {

     var dbName string = "DatabaseName"
     var dbUser string = "DatabaseUser"
```

```
     var dbHost string = "postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com"
     var dbPort int = 5432
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authenticationToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

如果要透過 Proxy 連線到資料庫叢集，請參閱 使用 IAM 身分驗證連線至資料庫。

使用 IAM 身分驗證和 適用於 Go 的 AWS SDK V1 進行連線。

您可以使用 IAM 身分驗證和 適用於 Go 的 AWS SDK V1 連線到資料庫叢集

以下程式碼範例顯示如何產生身分驗證字符，然後用來連線至資料庫叢集。

此程式碼連接到一個 Aurora MySQL 資料庫叢集。

```
package main
```

```
import (
    "database/sql"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go/aws/credentials"
    "github.com/aws/aws-sdk-go/service/rds/rdsutils"
    _ "github.com/go-sql-driver/mysql"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 3306
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

此程式碼連接到一個 Aurora PostgreSQL 資料庫叢集。

```
package main
```

```
import (
 "database/sql"
 "fmt"

 "github.com/aws/aws-sdk-go/aws/credentials"
 "github.com/aws/aws-sdk-go/service/rds/rdsutils"
 _ "github.com/lib/pq"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 5432
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

如果要透過 Proxy 連線到資料庫叢集，請參閱 使用 IAM 身分驗證連線至資料庫。

使用 IAM 身分驗證和 連線至資料庫叢集 適用於 Java 的 AWS SDK

您可以使用 連線到 Aurora MySQL 或 Aurora PostgreSQL 資料庫叢集 適用於 Java 的 AWS SDK ，如下所述。

先決條件

以下是使用 IAM 身分驗證連線至資料庫叢集的先決條件：

- 啟用和停用 IAM 資料庫身分驗證
- 建立並使用 IAM 政策進行 IAM 資料庫存取
- 使用 IAM 身分驗證建立資料庫帳戶
- 設定適用於 Java 的 AWS SDK

如需如何使用 SDK for Java 2.x 的範例，請參閱使用 SDK for Java 2.x 的 Amazon RDS 範例。您也可以使用 AWS 進階 JDBC 包裝函式，請參閱AWS 進階 JDBC 包裝函式文件。

主題

- 產生 IAM 身分驗證字符
- 手動建構 IAM 身分驗證字符
- 連接至資料庫叢集

產生 IAM 身分驗證字符

如果您使用 撰寫程式 適用於 Java 的 AWS SDK，您可以使用 `RdsIamAuthTokenGenerator`類別取得已簽署的身分驗證字符。使用此類別需要您提供 AWS 登入資料。若要這樣做，您可以建立 `DefaultAWSCredentialsProviderChain`類別的執行個體。`DefaultAWSCredentialsProviderChain`會使用其在預設登入資料提供者鏈中找到的第一個 AWS 存取金鑰和私密金鑰。如需有關 AWS 存取金鑰的詳細資訊，請參閱管理使用者的存取金鑰。

> **Note**
>
> 您無法使用自訂 Route 53 DNS 記錄，替代資料庫叢集端點來產生身分驗證字符。

建立 `RdsIamAuthTokenGenerator` 的執行個體之後，您可以呼叫 `getAuthToken` 方法來取得已簽署的字符。提供 AWS 區域、主機名稱、連接埠名稱和使用者名稱。以下程式碼範例示範如何執行此作業。

```
package com.amazonaws.codesamples;

import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
```

```
import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;

public class GenerateRDSAuthToken {

    public static void main(String[] args) {

     String region = "us-west-2";
     String hostname = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
     String port = "3306";
     String username = "jane_doe";

     System.out.println(generateAuthToken(region, hostname, port, username));
    }

    static String generateAuthToken(String region, String hostName, String port, String
 username) {

     RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
      .credentials(new DefaultAWSCredentialsProviderChain())
      .region(region)
      .build();

     String authToken = generator.getAuthToken(
      GetIamAuthTokenRequest.builder()
      .hostname(hostName)
      .port(Integer.parseInt(port))
      .userName(username)
      .build());

     return authToken;
    }

}
```

手動建構 IAM 身分驗證字符

在 Java 中，產生身分驗證字符最簡單的方式就是使用 RdsIamAuthTokenGenerator。此類別會為您建立身分驗證字符，然後使用 AWS 簽章第 4 版進行簽署。如需詳細資訊，請參閱 AWS 一般參考 中的 Signature 第 4 版簽署程序。

不過，您也可以手動建構和簽署身分驗證字符，如下列程式碼範例所示。

```
package com.amazonaws.codesamples;
```

```
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.SigningAlgorithm;
import com.amazonaws.util.BinaryUtils;
import org.apache.commons.lang3.StringUtils;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.Charset;
import java.security.MessageDigest;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.SortedMap;
import java.util.TreeMap;

import static com.amazonaws.auth.internal.SignerConstants.AWS4_TERMINATOR;
import static com.amazonaws.util.StringUtils.UTF8;

public class CreateRDSAuthTokenManually {
    public static String httpMethod = "GET";
    public static String action = "connect";
    public static String canonicalURIParameter = "/";
    public static SortedMap<String, String> canonicalQueryParameters = new TreeMap();
    public static String payload = StringUtils.EMPTY;
    public static String signedHeader = "host";
    public static String algorithm = "AWS4-HMAC-SHA256";
    public static String serviceName = "rds-db";
    public static String requestWithoutSignature;

    public static void main(String[] args) throws Exception {

        String region = "us-west-2";
        String instanceName = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
        String port = "3306";
        String username = "jane_doe";

        Date now = new Date();
        String date = new SimpleDateFormat("yyyyMMdd").format(now);
        String dateTimeStamp = new
 SimpleDateFormat("yyyyMMdd'T'HHmmss'Z'").format(now);
        DefaultAWSCredentialsProviderChain creds = new
 DefaultAWSCredentialsProviderChain();
     String awsAccessKey = creds.getCredentials().getAWSAccessKeyId();
```

```
    String awsSecretKey = creds.getCredentials().getAWSSecretKey();
       String expiryMinutes = "900";

       System.out.println("Step 1:  Create a canonical request:");
       String canonicalString = createCanonicalString(username, awsAccessKey, date,
dateTimeStamp, region, expiryMinutes, instanceName, port);
       System.out.println(canonicalString);
       System.out.println();

       System.out.println("Step 2:  Create a string to sign:");
       String stringToSign = createStringToSign(dateTimeStamp, canonicalString,
awsAccessKey, date, region);
       System.out.println(stringToSign);
       System.out.println();

       System.out.println("Step 3:  Calculate the signature:");
       String signature = BinaryUtils.toHex(calculateSignature(stringToSign,
newSigningKey(awsSecretKey, date, region, serviceName)));
       System.out.println(signature);
       System.out.println();

       System.out.println("Step 4:  Add the signing info to the request");

       System.out.println(appendSignature(signature));
       System.out.println();

   }

   //Step 1: Create a canonical request date should be in format YYYYMMDD and dateTime
should be in format YYYYMMDDTHHMMSSZ
   public static String createCanonicalString(String user, String accessKey, String
date, String dateTime, String region, String expiryPeriod, String hostName, String
port) throws Exception {
       canonicalQueryParameters.put("Action", action);
       canonicalQueryParameters.put("DBUser", user);
       canonicalQueryParameters.put("X-Amz-Algorithm", "AWS4-HMAC-SHA256");
       canonicalQueryParameters.put("X-Amz-Credential", accessKey + "%2F" + date +
"%2F" + region + "%2F" + serviceName + "%2Faws4_request");
       canonicalQueryParameters.put("X-Amz-Date", dateTime);
       canonicalQueryParameters.put("X-Amz-Expires", expiryPeriod);
       canonicalQueryParameters.put("X-Amz-SignedHeaders", signedHeader);
       String canonicalQueryString = "";
       while(!canonicalQueryParameters.isEmpty()) {
           String currentQueryParameter = canonicalQueryParameters.firstKey();
```

```
            String currentQueryParameterValue =
 canonicalQueryParameters.remove(currentQueryParameter);
            canonicalQueryString = canonicalQueryString + currentQueryParameter + "=" +
 currentQueryParameterValue;
            if (!currentQueryParameter.equals("X-Amz-SignedHeaders")) {
                canonicalQueryString += "&";
            }
        }
        String canonicalHeaders = "host:" + hostName + ":" + port + '\n';
        requestWithoutSignature = hostName + ":" + port + "/?" + canonicalQueryString;

        String hashedPayload = BinaryUtils.toHex(hash(payload));
        return httpMethod + '\n' + canonicalURIParameter + '\n' + canonicalQueryString
 + '\n' + canonicalHeaders + '\n' + signedHeader + '\n' + hashedPayload;

    }

    //Step 2: Create a string to sign using sig v4
    public static String createStringToSign(String dateTime, String canonicalRequest,
 String accessKey, String date, String region) throws Exception {
        String credentialScope = date + "/" + region + "/" + serviceName + "/
aws4_request";
        return algorithm + '\n' + dateTime + '\n' + credentialScope + '\n' +
 BinaryUtils.toHex(hash(canonicalRequest));

    }

    //Step 3: Calculate signature
    /**
     * Step 3 of the &AWS; Signature version 4 calculation. It involves deriving
     * the signing key and computing the signature. Refer to
     * http://docs.aws.amazon
     * .com/general/latest/gr/sigv4-calculate-signature.html
     */
    public static byte[] calculateSignature(String stringToSign,
                                            byte[] signingKey) {
        return sign(stringToSign.getBytes(Charset.forName("UTF-8")), signingKey,
                SigningAlgorithm.HmacSHA256);
    }

    public static byte[] sign(byte[] data, byte[] key,
                              SigningAlgorithm algorithm) throws SdkClientException {
        try {
            Mac mac = algorithm.getMac();
```

```
            mac.init(new SecretKeySpec(key, algorithm.toString()));
            return mac.doFinal(data);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    public static byte[] newSigningKey(String secretKey,
                                   String dateStamp, String regionName, String
serviceName) {
        byte[] kSecret = ("AWS4" + secretKey).getBytes(Charset.forName("UTF-8"));
        byte[] kDate = sign(dateStamp, kSecret, SigningAlgorithm.HmacSHA256);
        byte[] kRegion = sign(regionName, kDate, SigningAlgorithm.HmacSHA256);
        byte[] kService = sign(serviceName, kRegion,
                SigningAlgorithm.HmacSHA256);
        return sign(AWS4_TERMINATOR, kService, SigningAlgorithm.HmacSHA256);
    }

    public static byte[] sign(String stringData, byte[] key,
                      SigningAlgorithm algorithm) throws SdkClientException {
        try {
            byte[] data = stringData.getBytes(UTF8);
            return sign(data, key, algorithm);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    //Step 4: append the signature
    public static String appendSignature(String signature) {
        return requestWithoutSignature + "&X-Amz-Signature=" + signature;
    }

    public static byte[] hash(String s) throws Exception {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update(s.getBytes(UTF8));
            return md.digest();
        } catch (Exception e) {
            throw new SdkClientException(
```

```
                    "Unable to compute hash while signing request: "
                            + e.getMessage(), e);
        }
    }
}
```

連接至資料庫叢集

以下程式碼範例顯示如何產生身分驗證權杖，然後用來連接至執行 Aurora MySQL 的叢集。

若要執行此程式碼範例，您需要在 AWS 網站上適用於 Java 的 AWS SDK找到的 。此外，您還需要下列項目：

- MySQL Connector/J。本程式碼範本經過 `mysql-connector-java-5.1.33-bin.jar` 的測試。
- 區域特有的 Amazon Aurora 中繼憑證 AWS 。(如需詳細資訊，請參閱「使用 SSL/TLS 加密與資料庫叢集的連線」。) 在執行時間，類別載入器會在此 Java 程式碼範例所在的同一個目錄中尋找憑證，因此類別載入器可以找到憑證。
- 視需要修改下列變數的值：
  - `RDS_INSTANCE_HOSTNAME` – 您想要存取之資料庫叢集的主機名稱。
  - `RDS_INSTANCE_PORT` – 用於連線至 PostgreSQL 資料庫叢集的連接埠號碼。
  - `REGION_NAME` – 資料庫叢集執行所在的 AWS 區域。
  - `DB_USER` – 您想要存取的資料庫帳戶。
  - `SSL_CERTIFICATE` – 特定於 AWS 區域的 Amazon Aurora SSL 憑證。

    若要下載適用於 AWS 區域的憑證，請參閱使用 SSL/TLS 加密與資料庫叢集的連線。將 SSL 憑證放入此 Java 程式檔案所在的同一個目錄中，讓類別載入器在執行時間可以找到憑證。

此程式碼範例會從預設 AWS 登入資料提供者鏈結取得登入資料。 https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/credentials.html#credentials-default

> **Note**
>
> 為 `DEFAULT_KEY_STORE_PASSWORD` 指定此處所顯示提示以外的密碼，作為安全最佳實務。

```
package com.amazonaws.samples;

import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;
```

```
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;

import java.net.URL;

public class IAMDatabaseAuthenticationTester {
    //&AWS; Credentials of the IAM user with policy enabling IAM Database Authenticated
 access to the db by the db user.
    private static final DefaultAWSCredentialsProviderChain creds = new
 DefaultAWSCredentialsProviderChain();
    private static final String AWS_ACCESS_KEY =
 creds.getCredentials().getAWSAccessKeyId();
    private static final String AWS_SECRET_KEY =
 creds.getCredentials().getAWSSecretKey();

    //Configuration parameters for the generation of the IAM Database Authentication
 token
    private static final String RDS_INSTANCE_HOSTNAME = "rdsmysql.123456789012.us-
west-2.rds.amazonaws.com";
    private static final int RDS_INSTANCE_PORT = 3306;
    private static final String REGION_NAME = "us-west-2";
    private static final String DB_USER = "jane_doe";
    private static final String JDBC_URL = "jdbc:mysql://" + RDS_INSTANCE_HOSTNAME +
 ":" + RDS_INSTANCE_PORT;

    private static final String SSL_CERTIFICATE = "rds-ca-2019-us-west-2.pem";

    private static final String KEY_STORE_TYPE = "JKS";
    private static final String KEY_STORE_PROVIDER = "SUN";
```

```
    private static final String KEY_STORE_FILE_PREFIX = "sys-connect-via-ssl-test-
cacerts";
    private static final String KEY_STORE_FILE_SUFFIX = ".jks";
    private static final String DEFAULT_KEY_STORE_PASSWORD = "changeit";

    public static void main(String[] args) throws Exception {
        //get the connection
        Connection connection = getDBConnectionUsingIam();

        //verify the connection is successful
        Statement stmt= connection.createStatement();
        ResultSet rs=stmt.executeQuery("SELECT 'Success!' FROM DUAL;");
        while (rs.next()) {
             String id = rs.getString(1);
            System.out.println(id); //Should print "Success!"
        }

        //close the connection
        stmt.close();
        connection.close();

        clearSslProperties();

    }

    /**
     * This method returns a connection to the db instance authenticated using IAM
 Database Authentication
     * @return
     * @throws Exception
     */
    private static Connection getDBConnectionUsingIam() throws Exception {
        setSslProperties();
        return DriverManager.getConnection(JDBC_URL, setMySqlConnectionProperties());
    }

    /**
     * This method sets the mysql connection properties which includes the IAM Database
 Authentication token
     * as the password. It also specifies that SSL verification is required.
     * @return
     */
    private static Properties setMySqlConnectionProperties() {
        Properties mysqlConnectionProperties = new Properties();
```

```
        mysqlConnectionProperties.setProperty("verifyServerCertificate","true");
        mysqlConnectionProperties.setProperty("useSSL", "true");
        mysqlConnectionProperties.setProperty("user",DB_USER);
        mysqlConnectionProperties.setProperty("password",generateAuthToken());
        return mysqlConnectionProperties;
    }

    /**
     * This method generates the IAM Auth Token.
     * An example IAM Auth Token would look like follows:
     * btusi123.cmz7kenwo2ye.rds.cn-north-1.amazonaws.com.cn:3306/?
Action=connect&DBUser=iamtestuser&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Date=20171003T010726Z&X-Amz-SignedHeaders=host&X-Amz-Expires=899&X-Amz-
Credential=AKIAPFXHGVDI5RNFO4AQ%2F20171003%2Fcn-north-1%2Frds-db%2Faws4_request&X-Amz-
Signature=f9f45ef96c1f770cdad11a53e33ffa4c3730bc03fdee820cfdf1322eed15483b
     * @return
     */
    private static String generateAuthToken() {
        BasicAWSCredentials awsCredentials = new BasicAWSCredentials(AWS_ACCESS_KEY,
 AWS_SECRET_KEY);

        RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
                .credentials(new
 AWSStaticCredentialsProvider(awsCredentials)).region(REGION_NAME).build();
        return generator.getAuthToken(GetIamAuthTokenRequest.builder()

 .hostname(RDS_INSTANCE_HOSTNAME).port(RDS_INSTANCE_PORT).userName(DB_USER).build());
    }

    /**
     * This method sets the SSL properties which specify the key store file, its type
 and password:
     * @throws Exception
     */
    private static void setSslProperties() throws Exception {
        System.setProperty("javax.net.ssl.trustStore", createKeyStoreFile());
        System.setProperty("javax.net.ssl.trustStoreType", KEY_STORE_TYPE);
        System.setProperty("javax.net.ssl.trustStorePassword",
 DEFAULT_KEY_STORE_PASSWORD);
    }

    /**
     * This method returns the path of the Key Store File needed for the SSL
 verification during the IAM Database Authentication to
```

```
     * the db instance.
     * @return
     * @throws Exception
     */
    private static String createKeyStoreFile() throws Exception {
        return createKeyStoreFile(createCertificate()).getPath();
    }

    /**
     *  This method generates the SSL certificate
     * @return
     * @throws Exception
     */
    private static X509Certificate createCertificate() throws Exception {
        CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
        URL url = new File(SSL_CERTIFICATE).toURI().toURL();
        if (url == null) {
            throw new Exception();
        }
        try (InputStream certInputStream = url.openStream()) {
            return (X509Certificate) certFactory.generateCertificate(certInputStream);
        }
    }

    /**
     * This method creates the Key Store File
     * @param rootX509Certificate - the SSL certificate to be stored in the KeyStore
     * @return
     * @throws Exception
     */
    private static File createKeyStoreFile(X509Certificate rootX509Certificate) throws
Exception {
        File keyStoreFile = File.createTempFile(KEY_STORE_FILE_PREFIX,
KEY_STORE_FILE_SUFFIX);
        try (FileOutputStream fos = new FileOutputStream(keyStoreFile.getPath())) {
            KeyStore ks = KeyStore.getInstance(KEY_STORE_TYPE, KEY_STORE_PROVIDER);
            ks.load(null);
            ks.setCertificateEntry("rootCaCertificate", rootX509Certificate);
            ks.store(fos, DEFAULT_KEY_STORE_PASSWORD.toCharArray());
        }
        return keyStoreFile;
    }

    /**
```

```
     * This method clears the SSL properties.
     * @throws Exception
     */
    private static void clearSslProperties() throws Exception {
           System.clearProperty("javax.net.ssl.trustStore");
           System.clearProperty("javax.net.ssl.trustStoreType");
           System.clearProperty("javax.net.ssl.trustStorePassword");
    }

}
```

如果要透過 Proxy 連線到資料庫叢集，請參閱 使用 IAM 身分驗證連線至資料庫。

使用 IAM 身分驗證和 連線至資料庫叢集 適用於 Python (Boto3) 的 AWS SDK

您可以使用 連線到 Aurora MySQL 或 Aurora PostgreSQL 資料庫叢集 適用於 Python (Boto3) 的 AWS SDK ，如下所述。

先決條件

以下是使用 IAM 身分驗證連線至資料庫叢集的先決條件：

- 啟用和停用 IAM 資料庫身分驗證
- 建立並使用 IAM 政策進行 IAM 資料庫存取
- 使用 IAM 身分驗證建立資料庫帳戶

此外，請確定範例程式碼中匯入的程式庫存在於您的系統上。

範例

程式碼範例使用設定檔進行共用登入資料。如需指定登入資料的資訊，請參閱 適用於 Python (Boto3) 的 AWS SDK 文件中的登入資料。

以下程式碼範例顯示如何產生身分驗證字符，然後用來連線至資料庫叢集。

若要執行此程式碼範例，您需要在 AWS 網站上適用於 Python (Boto3) 的 AWS SDK找到的 。

視需要修改下列變數的值：

- `ENDPOINT` – 您想要存取之資料庫叢集的端點
- `PORT` – 用於連線資料庫叢集的連接埠號碼

- USER – 您想要存取的資料庫帳戶
- REGION – 資料庫叢集執行所在的 AWS 區域
- DBNAME – 您想要存取的資料庫
- SSLCERTIFICATE – Amazon Aurora 之 SSL 憑證的完整路徑

  對於 ssl_ca，指定 SSL 憑證。若要下載 SSL 憑證，請參閱使用 SSL/TLS 加密與資料庫叢集的連線。

> **Note**
>
> 您無法使用自訂 Route 53 DNS 記錄，替代資料庫叢集端點來產生身分驗證字符。

此程式碼連接到一個 Aurora MySQL 資料庫叢集。

在執行此程式碼之前，請遵循 Python 套件索引中的指示，來安裝 PyMySQL 驅動程式。

```
import pymysql
import sys
import boto3
import os

ENDPOINT="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="3306"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"
os.environ['LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN'] = '1'

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='default')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn =
 pymysql.connect(auth_plugin_map={'mysql_clear_password':None},host=ENDPOINT,
 user=USER, password=token, port=PORT, database=DBNAME, ssl_ca='SSLCERTIFICATE',
 ssl_verify_identity=True, ssl_verify_cert=True)
```

```
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

此程式碼連接到一個 Aurora PostgreSQL 資料庫叢集。

在執行此程式碼之前，請按照 Psycopg 文件中的指示來安裝 psycopg2。

```
import psycopg2
import sys
import boto3
import os

ENDPOINT="postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="5432"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='RDSCreds')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn = psycopg2.connect(host=ENDPOINT, port=PORT, database=DBNAME, user=USER,
 password=token, sslrootcert="SSLCERTIFICATE")
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

如果要透過 Proxy 連線到資料庫叢集，請參閱 使用 IAM 身分驗證連線至資料庫。

IAM 資料庫身分驗證的疑難排解

您可在下列內容中找到某些常見 IAM 資料庫身分驗證問題的疑難排解想法，以及 IAM 資料庫身分驗證的 CloudWatch 日誌和指標的相關資訊。

將 IAM 資料庫身分驗證錯誤日誌匯出至 CloudWatch Logs

IAM 資料庫身分驗證錯誤日誌會存放在資料庫主機上，您可以將這些日誌匯出至您的 CloudWatch Logs 帳戶。使用此頁面中的日誌和修補方法對 IAM 資料庫身分驗證問題進行疑難排解。

您可以從主控台 AWS CLI、和 RDS API 啟用日誌匯出至 CloudWatch Logs 的功能。如需主控台指示，請參閱 將資料庫日誌發佈至 Amazon CloudWatch Logs。

若要在從 AWS CLI 建立資料庫叢集時，將您的 IAM 資料庫身分驗證錯誤日誌匯出至 CloudWatch Logs，請使用下列命令：

```
aws rds create-db-cluster --db-cluster-identifier mydbinstance \
--region us-east-1 \
--engine postgres \
--engine-version 16 \
--master-username master \
--master-user-password password \
--publicly-accessible \
--enable-iam-database-authentication \
--enable-cloudwatch-logs-exports=iam-db-auth-error
```

若要在從 AWS CLI 修改資料庫叢集時，將您的 IAM 資料庫身分驗證錯誤日誌匯出至 CloudWatch Logs，請使用下列命令：

```
aws rds modify-db-cluster --db-cluster-identifier mydbcluster \
--region us-east-1 \
--cloudwatch-logs-export-configuration '{"EnableLogTypes":["iam-db-auth-error"]}'
```

若要驗證資料庫叢集是否正在將 IAM 資料庫身分驗證日誌匯出至 CloudWatch Logs，請檢查 `EnabledCloudwatchLogsExports` 參數是否在 `describe-db-instances` 命令的輸出中設定為 `iam-db-auth-error`。

```
aws rds describe-db-cluster --region us-east-1 --db-cluster-identifier mydbcluster
            ...

             "EnabledCloudwatchLogsExports": [
                "iam-db-auth-error"
```

```
        ],
        ...
```

IAM 資料庫身分驗證 CloudWatch 指標

Amazon Aurora 為您的 Amazon CloudWatch 帳戶提供近乎即時的 IAM 資料庫身分驗證指標。下表列出可使用 CloudWatch 的 IAM 資料庫身分驗證指標：

指標	描述
IamDbAuthConnectionRequests	使用 IAM 資料庫身分驗證提出的連線請求總數。
IamDbAuthConnectionSuccess	成功 IAM 資料庫身分驗證請求總數。
IamDbAuthConnectionFailure	失敗的 IAM 資料庫身分驗證請求總數。
IamDbAuthConnectionFailureInvalidToken	由於字符無效而失敗的 IAM 資料庫身分驗證請求總數。
IamDbAuthConnectionFailureInsufficientPermissions	由於政策或許可不正確的失敗 IAM 資料庫身分驗證請求總數。
IamDbAuthConnectionFailureThrottling	由於 IAM 資料庫身分驗證限流的失敗 IAM 資料庫身分驗證請求總數。
IamDbAuthConnectionFailureServerError	由於 IAM 資料庫身分驗證功能中內部伺服器錯誤的失敗 IAM 資料庫身分驗證請求總數。

常見問題與解決方案

使用 IAM 資料庫身分驗證時可能會遇到下列問題。使用資料表中的修補步驟來解決問題：

錯誤	指標	原因	解決方案
[ERROR] Failed to authenticate the connectio	IamDbAuthConnectionFailure	連線請求中的 IAM 資料庫身分驗證字符不是有效的 SigV4a	檢查應用程式中的字符產生策略。在某些情況下，請確定您使

錯誤	指標	原因	解決方案
n request for user *db_user* because the provided token is malformed or otherwise invalid. (Status Code: 400, Error Code: InvalidToken)	IamDbAuthConnectionFailureInvalidToken	字符，或者格式不正確。	用有效的格式傳遞字符。截斷字符 (或不正確的字串格式) 會使字符無效。
[ERROR] Failed to authenticate the connection request for user *db_user* because the token age is longer than 15 minutes. (Status Code: 400, Error Code:ExpiredToken)	IamDbAuthConnectionFailure IamDbAuthConnectionFailureInvalidToken	IAM 資料庫身分驗證字符已過期。字符的有效時間為 15 分鐘。	檢查您應用程式中的字符快取和/或字符重複使用邏輯。您不應重複使用超過 15 分鐘的字符。

錯誤	指標	原因	解決方案
[ERROR] Failed to authorize the connection request for user *db_user* because the IAM policy assumed by the caller 'arn:aws:sts::123456789012:assumed-role/ <RoleName>/ <RoleSession>' is not authorized to perform `rds-db:connect` on the DB instance. (Status Code: 403, Error Code:NotAuthorized)	IamDbAuthConnectionFailure IamDbAuthConnectionFailureInsufficientPermissions	此錯誤可能由下列原因造成： • 應用程式擔任的 IAM 政策不會授權 rds-db:connect 動作。 • 您擔任以連線到資料庫的 *db_user* 角色/政策不正確。 • 您擔任的 *db_user* 政策正確，但您未連線到正確的資料庫。	驗證您在應用程式中擔任的 IAM 角色和/或政策。請確定您擔任以產生字符的政策，與連線至資料庫的政策相同。
[ERROR] Failed to authorize the connection request for user *db_user* due to IAM DB authentication throttling. (Status Code: 429, Error Code: ThrottlingException)	IamDbAuthConnectionFailure IamDbAuthConnectionFailureThrottling	您在短時間內對資料庫提出太多連線請求。IAM 資料庫身分驗證限流限制為每秒 200 個連線。	降低使用 IAM 身分驗證建立新連線的速率。請考慮使用 RDS Proxy 實作連線集區，以便在應用程式中重複使用已建立的連線。

錯誤	指標	原因	解決方案
[ERROR] Failed to authorize the connection request for user *db_user* due to an internal IAM DB authentication error. (Status Code: 500, Error Code: InternalError)	IamDbAuthConnectionFailure IamDbAuthConnectionFailureThrottling	使用 IAM 資料庫身分驗證授權資料庫連線時發生內部錯誤。	請聯絡 https://aws.amazon.com/premiumsupport/ 以調查問題。

對 Amazon Aurora 身分與存取進行故障診斷

請使用以下資訊來協助您診斷和修復使用 Aurora 和 IAM 時發生的常見問題。

主題

- 我未獲授權在 Aurora 中執行動作
- 我未獲授權執行 iam:PassRole
- 我想要允許 AWS 帳戶外的人員存取我的 Aurora資源

我未獲授權在 Aurora 中執行動作

如果 AWS 管理主控台 告知您無權執行 動作，則必須聯絡您的管理員尋求協助。您的管理員是為您提供簽署憑證的人員。

以下範例錯誤會在 mateojackson 使用者嘗試使用主控台檢視 *Widget* 的詳細資訊，但卻沒有 rds:*GetWidget* 許可時發生。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 rds:GetWidget on resource: my-example-widget
```

在此情況下，Mateo 會請求管理員更新他的政策，允許他使用 *my-example-widget* 動作存取 rds:*GetWidget* 資源。

我未獲授權執行 iam:PassRole

若您收到錯誤，告知您並未獲得執行 `iam:PassRole` 動作的授權，您必須聯絡您的管理員以取得協助。您的管理員是為您提供簽署憑證的人員。要求該人員更新您的政策，允許您將角色傳遞給 Aurora。

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

以下範例錯誤會在名為 `marymajor` 的使用者嘗試使用主控台在 Aurora 中執行動作時發生。不過， 動作需要服務具有服務角色授予的許可。Mary 沒有將角色傳遞至該服務的許可。

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

在這種情況下，Mary 會請求管理員更新她的政策，允許她執行 `iam:PassRole` 動作。

我想要允許 AWS 帳戶外的人員存取我的 Aurora資源

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

若要進一步了解，請參閱以下內容：

- 若要了解 Aurora 是否支援這些功能，請參閱Amazon Aurora 如何搭配 IAM 運作。
- 若要了解如何在您擁有的 AWS 帳戶中提供資源的存取權，請參閱《IAM 使用者指南》中的為您擁有的另一個 AWS 帳戶中的 IAM 使用者提供存取權。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的將存取權提供給第三方擁有 AWS 的帳戶。
- 如需了解如何透過聯合身分提供存取權，請參閱 IAM 使用者指南中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 若要了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 角色與資源型政策的差異。

在 Amazon Aurora 中記錄和監控

監控是維護 Amazon Aurora 和您的 AWS 解決方案可靠性、可用性和效能的重要部分。您應該從 AWS 解決方案的所有部分收集監控資料，以便在發生多點失敗時更輕鬆地偵錯。 AWS 提供數種工具來監控 Amazon Aurora 資源並回應潛在事件：

Amazon CloudWatch 警示

使用 Amazon CloudWatch 警示，您可在自己指定的一段時間內監看單一指標。如果指標超過指定的閾值，則會將通知傳送至 Amazon SNS 主題或 AWS Auto Scaling 政策。CloudWatch 警示不會因為處於特定狀態而調用動作。必須是狀態已變更並維持了所指定的時間長度，才會呼叫動作。

AWS CloudTrail 日誌

對於使用者、角色或 AWS 服務在 Amazon Aurora 中採取的動作，CloudTrail 會提供記錄。CloudTrail 將 Amazon Aurora 的所有 API 呼叫擷取為事件，包括來自主控台的呼叫以及來自對 Amazon RDS API 操作發出的程式碼呼叫。使用由 CloudTrail 收集的資訊，您就可以判斷傳送至 Amazon Aurora 的請求、提出請求的 IP 地址、提出請求的對象、提出請求的時間，以及其他詳細資訊。如需詳細資訊，請參閱 在 AWS CloudTrail 中監控 Amazon Aurora API 呼叫。

Enhanced Monitoring (增強型監控)

Amazon Aurora 可針對資料庫叢集執行所在的作業系統 (OS) 即時提供指標。您可以使用主控台來檢視資料庫叢集的指標，或在您所選的監控系統中利用來自 Amazon CloudWatch Logs 的增強型監控 JSON 輸出。如需詳細資訊，請參閱 使用增強型監控來監控作業系統指標。

Amazon RDS 績效詳情

Performance Insights 會延伸現有 Amazon Aurora 監控功能的基礎，藉此說明資料庫效能，並幫助您分析可能影響效能的任何問題。利用績效詳情儀表板，您可以將資料庫負載視覺化，並依等候、SQL 陳述式、主機或使用者篩選負載。如需詳細資訊，請參閱 在 Amazon Aurora 上使用績效詳情監控資料庫負載。

資料庫日誌

您可以使用 、 或 RDS API 檢視 AWS 管理主控台 AWS CLI、下載和監看資料庫日誌。如需詳細資訊，請參閱 監控 Amazon Aurora 日誌檔案。

Amazon Aurora 建議事項

Amazon Aurora 提供自動化的資料庫資源建議。這些建議事項會透過分析資料庫叢集、組態、用量及效能資料以提供最佳實務指南。如需詳細資訊，請參閱 Amazon Aurora 的建議。

Amazon Aurora 事件通知

Amazon Aurora 使用 Amazon Simple Notification Service (Amazon SNS) 在 Amazon Aurora 事件發生時提供通知。這些通知可以是 AWS 區域 Amazon SNS 支援的任何通知形式，例如電子郵件、文字訊息或對 HTTP 端點的呼叫。如需詳細資訊，請參閱 使用 Amazon RDS 事件通知。

AWS Trusted Advisor

Trusted Advisor 利用從為數十萬 AWS 客戶提供服務的最佳實務。 會 Trusted Advisor 檢查您的 AWS 環境，然後在有機會節省成本、改善系統可用性和效能，或協助填補安全漏洞時提出建議。所有 AWS 客戶都可以存取五個 Trusted Advisor 檢查。擁有商業或企業支援計劃的客戶可以檢視所有 Trusted Advisor 檢查。

Trusted Advisor 具有下列 Amazon Aurora 相關檢查：

- Amazon Aurora 閒置資料庫執行個體
- Amazon Aurora 安全群組存取風險
- Amazon Aurora 備份
- Amazon Aurora 多個可用區域
- Aurora 資料庫執行個體存取能力

如需這些檢查的詳細資訊，請參閱 Trusted Advisor 最佳實務 (檢查)。

資料庫活動串流

資料庫活動串流可透過控制 DBA 對資料庫活動串流的存取，來保護資料庫不受內部威脅的傷害。因此，資料庫活動串流的收集、傳輸、儲存和後續處理都會超出管理資料庫的 DBA 存取權限範圍。資料庫活動串流可為您的資料庫提供保護，並符合合規與法規要求。如需詳細資訊，請參閱 使用資料庫活動串流來監控 Amazon Aurora。

如需監控 Aurora 的詳細資訊，請參閱在 Amazon Aurora 叢集中監控指標。

Amazon Aurora 的合規驗證

在多個 AWS 合規計劃中，第三方稽核人員會評估 Amazon Aurora 的安全與合規。這些計劃包括 SOC、PCI、FedRAMP、HIPAA 等等。

如需特定合規計劃範圍內AWS的服務清單，請參閱AWS合規計劃範圍內的服務。如需一般資訊，請參閱 AWS 合規計劃。

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

您使用 Amazon Aurora 時的合規責任取決於資料的機密性、組織的合規目標，以及適用的法律和法規。 AWS提供下列資源以協助合規：

- 安全與合規快速入門指南 – 這些部署指南討論架構考量，並提供在其中部署以安全與合規為重心的基準環境的步驟AWS。
- Amazon Web Services 上的 HIPAA 安全與合規架構 – 此白皮書說明公司如何使用 AWS來建立符合 HIPAA 規範的應用程式。
- AWS合規資源 – 此工作手冊和指南集合可能適用於您的產業和位置。
- AWS Config – AWS此服務會評估您的資源組態是否符合內部實務、產業準則和法規。
- AWS Security Hub CSPM – 這AWS 服務可讓您全面檢視其中的安全狀態AWS。Security Hub CSPM 使用安全控制來評估您的AWS資源，並檢查是否符合安全產業標準和最佳實務。如需支援的服務和控制項清單，請參閱 Security Hub CSPM 控制項參考。

Amazon Aurora 的復原功能

AWS 全球基礎設施是以 AWS 區域與可用區域為中心建置的。AWS區域提供多個分開且隔離的實際可用區域，它們以低延遲、高輸送量和高度備援聯網功能相互連結。透過可用區域，您所設計與操作的應用程式和資料庫，就能夠在可用區域之間自動容錯移轉，而不會發生中斷。可用區域的可用性、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

如需 AWS 區域與可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

除了 AWS 全球基礎設施外，Aurora 還提供支援資料復原和備份需求的多項功能。

備份和還原

Aurora 會自動備份您的叢集磁碟區，並在備份保留期間保留還原資料。Aurora 備份具有連續性和增量性，因此，您可以快速還原到備份保留期內的任何時間點。寫入備份資料時不會影響資料庫服務的效能或中斷服務。當您建立或修改資料庫叢集時，您可以指定 1 到 35 天的備份保留期。

如果希望備份可以保留超過備份保留期，您也可以在叢集磁碟區中建立資料快照。Aurora 在整個備份保留期內保留遞增還原資料。因此，您只需要建立希望在備份保留期後仍能保留的資料快照。您可以從快照建立新的資料庫叢集。

您可以從 Aurora 保留的備份資料或您已儲存的資料庫叢集快照來建立新的 Aurora 資料庫叢集，以復原資料。您可以從備份資料快速建立備份保留期內任何時間點的資料庫叢集新副本。Aurora 在備份保留期之內的連續和增量性質，意味著您不需要經常建立資料的快照來改善還原時間。

如需更多詳細資訊，請參閱 備份與還原 Amazon Aurora 資料庫叢集。

複寫

Aurora 複本是 Aurora 資料庫叢集中的獨立端點，最適合用於擴展讀取操作和提高可用性。最多 15 個 Aurora 複本可以分佈在 AWS 區域內資料庫叢集跨越的可用區域。資料庫叢集磁碟區由資料庫叢集的多個資料副本組成。然而，叢集磁碟區中的資料是以單一邏輯磁碟區的形式，向主要資料庫執行個體和資料庫叢集中的 Aurora 複本呈現。如果主要資料庫執行個體失敗，則 Aurora 複本會提升為主要資料庫執行個體。

Aurora 也支援專屬於 Aurora MySQL 和 Aurora PostgreSQL 的複寫選項。

如需更多詳細資訊，請參閱 以 Amazon Aurora 進行複寫。

容錯移轉

Aurora 將資料副本存放在單一 AWS 區域跨多個可用區域的資料庫叢集中。無論資料庫執行個體是否位在跨多個可用區的資料庫叢集中，都會執行此儲存。當您跨可用區域建立 Aurora 複本時，Aurora 會自動同步佈建並維護複本。主要資料庫執行個體會跨可用區域，同步複寫到 Aurora 複本，提供資料備援、排除輸入/輸出凍結以及降低系統備份時的延遲遽增等功能。執行具有高可用性的資料庫叢集，可以在規劃好的系統維護期間增強可用性，並有助於在失敗和可用區域停機時保護資料庫。

如需更多詳細資訊，請參閱 Amazon Aurora 的高可用性。

Amazon Aurora 中的基礎設施安全

Amazon Relational Database Service 是受管服務，受到 AWS 全球網路安全的保護。如需 AWS 安全服務以及如何 AWS 保護基礎設施的資訊，請參閱AWS 雲端安全。若要使用基礎設施安全的最佳實務來設計您的 AWS 環境，請參閱安全支柱 AWS Well-Architected Framework 中的基礎設施保護。

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

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

此外，Aurora 提供有助於支援基礎設施安全的功能。

安全群組

安全群組會控制進出資料庫叢集流量的存取權限。資料庫叢集的網路存取預設是關閉的。您可以在允許從 IP 地址範圍、連接埠或安全群組存取的安全群組中指定規則。設定傳入規則後，相同規則就會套用到與該安全群組相關聯的所有資料庫叢集。

如需詳細資訊，請參閱使用安全群組控制存取。

Public accessibility (公開存取性)

當您根據 Amazon VPC 服務啟動虛擬私有雲端 (VPC) 內的資料庫執行個體時，您可以開啟或關閉該資料庫執行個體的公開存取性。請使用 Public accessibility (公開存取權限) 參數，來指定您建立的資料庫執行個體是否有解析為公有 IP 地址的 DNS。使用此參數，您就可以指定資料庫執行個體是否可供公開存取。您可以修改 Public accessibility (公開存取權限) 參數，藉此修改資料庫執行個體設定，以開啟或關閉公開存取權限。

如需詳細資訊，請參閱在 VPC 中的網際網路中隱藏資料庫叢集。

> **Note**
>
> 如果您的資料庫執行個體位於 VPC 中，但無法公開存取，您也可以使用 an AWS Site-to-Site VPN 連線或 Direct Connect 連線從私有網路存取。如需詳細資訊，請參閱網際網路流量隱私權。

Amazon RDS API 和界面 VPC 端點 (AWS PrivateLink)

您可以建立界面 VPC 端點，以在您的 VPC 與 Amazon RDS API 端點之間建立私有連線。界面端點是採用 AWS PrivateLink 技術。

AWS PrivateLink 可讓您在沒有網際網路閘道、NAT 裝置、VPN 連線或 Direct Connect 連線的情況下，私下存取 Amazon RDS API 操作。VPC 中的資料庫執行個體不需要公有 IP 地址，就能與 Amazon RDS API 端點進行通訊，以啟動、修改或終止資料庫執行個體和資料庫叢集。您的資料庫執行個體也不需要公有 IP 地址，就能使用任何可用的 RDS API 操作。您的 VPC 與 Amazon RDS 之間的流量，都會在 Amazon 網路的範圍內。

每個界面端點都是由您子網路中的一或多個彈性網路界面表示。如需彈性網路界面的詳細資訊，請參閱 Amazon EC2 使用者指南中的彈性網路界面。

如需 VPC 端點的詳細資訊，請參閱《Amazon VPC 使用者指南》中的界面 VPC 端點 (AWS PrivateLink)。如需 RDS API 操作的資訊，請參閱 Amazon RDS API 參考。

您連線至資料庫叢集時不需要界面 VPC 端點。如需詳細資訊，請參閱在 VPC 中存取資料庫叢集的案例。

VPC 端點的考量事項

在設定 Amazon RDS API 端點的界面 VPC 端點前，請務必檢閱《Amazon VPC 使用者指南》中的界面端點屬性和限制。

所有與管理 Amazon Aurora 資源相關的 RDS API 操作都從使用 AWS PrivateLink的 VPC 提供。

RDS API 端點支援 VPC 端點政策。根據預設，允許透過端點對 RDS API 操作進行完整存取。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的使用 VPC 端點控制對服務的存取。

可用性

Amazon RDS API 目前在下列 AWS 區域中支援 VPC 端點：

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

- 非洲 (開普敦)
- 亞太區域 (香港)
- Asia Pacific (Mumbai)
- 亞太區域 (紐西蘭)
- 亞太地區 (大阪)
- 亞太區域 (首爾)
- 亞太區域 (新加坡)
- 亞太地區 (雪梨)
- 亞太區域 (台北)
- 亞太區域 (泰國)
- 亞太地區 (東京)
- 加拿大 (中部)
- 加拿大西部 (卡加利)
- 中國 (北京)
- 中國 (寧夏)
- 歐洲 (法蘭克福)
- 歐洲 (蘇黎世)
- 歐洲 (愛爾蘭)
- 歐洲 (倫敦)
- 歐洲 (巴黎)
- 歐洲 (斯德哥爾摩)
- 歐洲 (米蘭)
- 以色列 (特拉維夫)
- 墨西哥 (中部)
- Middle East (Bahrain)
- 南美洲 (聖保羅)
- AWS GovCloud （美國東部）
- AWS GovCloud （美國西部）

為 Amazon RDS API 建立界面 VPC 端點

您可以使用 Amazon VPC 主控台或 AWS Command Line Interface () 為 Amazon RDS API 建立 VPC 端點AWS CLI。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的建立介面端點。

使用服務名稱 `com.amazonaws.region.rds`，為 Amazon RDS API 建立 VPC 端點。

排除中國 AWS 的區域，如果您為端點啟用私有 DNS，您可以使用 VPC 端點對 Amazon RDS 提出 API 請求，並使用其區域的預設 DNS 名稱 AWS ，例如 `rds.us-east-1.amazonaws.com`。對於中國（北京）和中國（寧夏）AWS 區域，您可以`rds-api.cn-northwest-1.amazonaws.com.cn`分別使用 `rds-api.cn-north-1.amazonaws.com.cn`和 使用 VPC 端點提出 API 請求。

如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的透過介面端點存取服務。

為 Amazon RDS API 建立 VPC 端點政策

您可以將端點政策連接至控制 Amazon RDS API 存取權限的 VPC 端點。此政策會指定下列資訊：

- 可執行動作的主體。
- 可執行的動作。
- 可供執行動作的資源。

如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的使用 VPC 端點控制對服務的存取。

範例：Amazon RDS API 動作的 VPC 端點政策

以下是 Amazon RDS API 端點政策的範例。連接至端點後，此政策會針對所有資源上的所有委託人，授予列出的 Amazon RDS API 動作的存取權限。

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBInstance",
            "rds:ModifyDBInstance",
            "rds:CreateDBSnapshot"
         ],
         "Resource":"*"
```

```
      }
   ]
}
```

範例：拒絕來自指定 AWS 帳戶的所有存取的 VPC 端點政策

下列 VPC 端點政策拒絕 AWS 使用端點帳戶對資源123456789012的所有存取。此政策允許來自其他帳戶的所有動作。

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
   ]
}
```

Amazon Aurora 的安全最佳實務

使用 AWS Identity and Access Management (IAM) 帳戶來控制對 Amazon RDS API 操作的存取，尤其是建立、修改或刪除 Amazon Aurora 資源的操作。這類資源包含資料庫叢集、安全群組和參數群組。同時也請使用 IAM 控制執行常見管理動作的動作，例如備份和還原資料庫叢集。

- 為管理 Amazon Aurora 資源的每個人 (包括您自己) 建立個別使用者。請勿使用 AWS 根登入資料來管理 Amazon Aurora 資源。
- 授予每個使用者執行其職責所需最低程度的許可。
- 使用 IAM 群組來有效管理多個使用者的許可。
- 定期輪替您的 IAM 登入資料。
- 設定 AWS Secrets Manager 自動輪換 Amazon Aurora 的秘密。如需更多詳細資訊，請參閱 AWS Secrets Manager 使用者指南中的輪換 AWS Secrets Manager 密碼。您也可以透過 AWS Secrets

Manager 程式設計方式從 擷取登入資料。如需更多詳細資訊，請參閱 AWS Secrets Manager 使用者指南中的擷取密碼值。

如需 Amazon Aurora 安全性的詳細資訊，請參閱 Amazon Aurora 的安全性。如需關於 IAM 的詳細資訊，請參閱 AWS Identity and Access Management。如需 IAM 最佳實務的資訊，請參閱 IAM 最佳實務。

AWS Security Hub CSPM 使用安全控制來評估資源組態和安全標準，以協助您遵守各種合規架構。如需使用 Security Hub CSPM 評估 RDS 資源的詳細資訊，請參閱 AWS Security Hub 《使用者指南》中的 Amazon Relational Database Service 控制項。

您可以使用 Security Hub CSPM 來監控 RDS 的使用，因為它與安全最佳實務相關。如需詳細資訊，請參閱什麼是 AWS Security Hub CSPM？。

使用 AWS CLI、 AWS 管理主控台或 RDS API 來變更主要使用者的密碼。如果您使用其他工具 (如 SQL 用戶端) 來變更主要使用者的密碼，可能會導致系統意外撤銷使用者權限。

Amazon GuardDuty 是持續性的安全監控服務，用來分析和處理各種資料來源，包括 Amazon RDS 登入活動。其使用威脅智慧饋送和機器學習以在 AWS 環境內識別意外、可能未經授權、可疑登入行為且惡意的活動。

當 Amazon GuardDuty RDS 保護偵測到潛在的可疑或異常登入嘗試 (這表明資料庫存在安全威脅) 時，GuardDuty 會產生新的調查結果，其中包含可能被盜用之資料庫的詳細資訊。如需詳細資訊，請參閱 使用 Amazon GuardDuty RDS Protection for Amazon Aurora監控威脅。

使用安全群組控制存取

VPC 安全群組會控制進出資料庫叢集流量的存取權限。根據預設，您資料庫叢集的網路存取是關閉的。您可以在允許從 IP 地址範圍、連接埠或安全群組存取的安全群組中指定規則。設定傳入規則後，相同規則就會套用到與該安全群組相關聯的所有資料庫叢集。您最多可在安全群組中指定 20 條規則。

VPC 安全群組概觀

每個 VPC 安全群組都可能可以讓特定來源存取 VPC 中與該 VPC 安全群組相關聯的資料庫叢集。來源可以是地址的來源 (例如，203.0.113.0/24) 或另一個 VPC 安全群組。藉由指定 VPC 安全群組做為來源，您允許從使用來源 VPC 安全群組的所有執行個體 (通常指的是應用程式伺服器) 傳入的流量。VPC 安全群組可以擁有同時掌管入站和出站流量的規則。但是，傳出流量規則通常不適用於資料庫叢集。僅在資料庫叢集充當用戶端時，傳出流量規則才適用。您必須使用 Amazon EC2 API 或 VPC 主控台上的 Security Group (安全群組) 選項，才能建立 VPC 安全群組。

為您的 VPC 安全群組建立允許存取 VPC 中叢集的規則時，您必須針對規則允許存取的每個地址範圍指定一個連接埠。例如，如果您想要對 VPC 中的執行個體開啟安全殼 (SSH) 存取，則可以建立一個規則，允許存取所指定之地址範圍的 TCP 連接埠 22。

您可以設定多個 VPC 安全群組，允許存取 VPC 中不同執行個體的不同連接埠。例如，您可以建立 VPC 安全群組，允許存取 VPC 中 Web 伺服器的 TCP 連接埠 80。然後，您可以建立另一個 VPC 安全群組，允許存取 VPC 中 Aurora MySQL 資料庫執行個體的 TCP 連接埠 3306。

> **Note**
>
> 在 Aurora 資料庫叢集中，與資料庫叢集關聯的 VPC 安全群組也會與資料庫叢集中所有資料庫執行個體相關聯。如果您變更資料庫叢集或資料庫執行個體的 VPC 安全群組，則系統會將此變更自動套用到資料庫叢集中的所有資料庫執行個體。

如需 VPC 安全群組的詳細資訊，請參閱《Amazon Virtual Private Cloud 使用者指南》中的安全群組。

> **Note**
>
> 如果您的資料庫叢集位於 VPC 中，但無法公開存取，您也可以使用 an AWS Site-to-Site VPN 連線或 Direct Connect 連線從私有網路存取。如需詳細資訊，請參閱 網際網路流量隱私權。

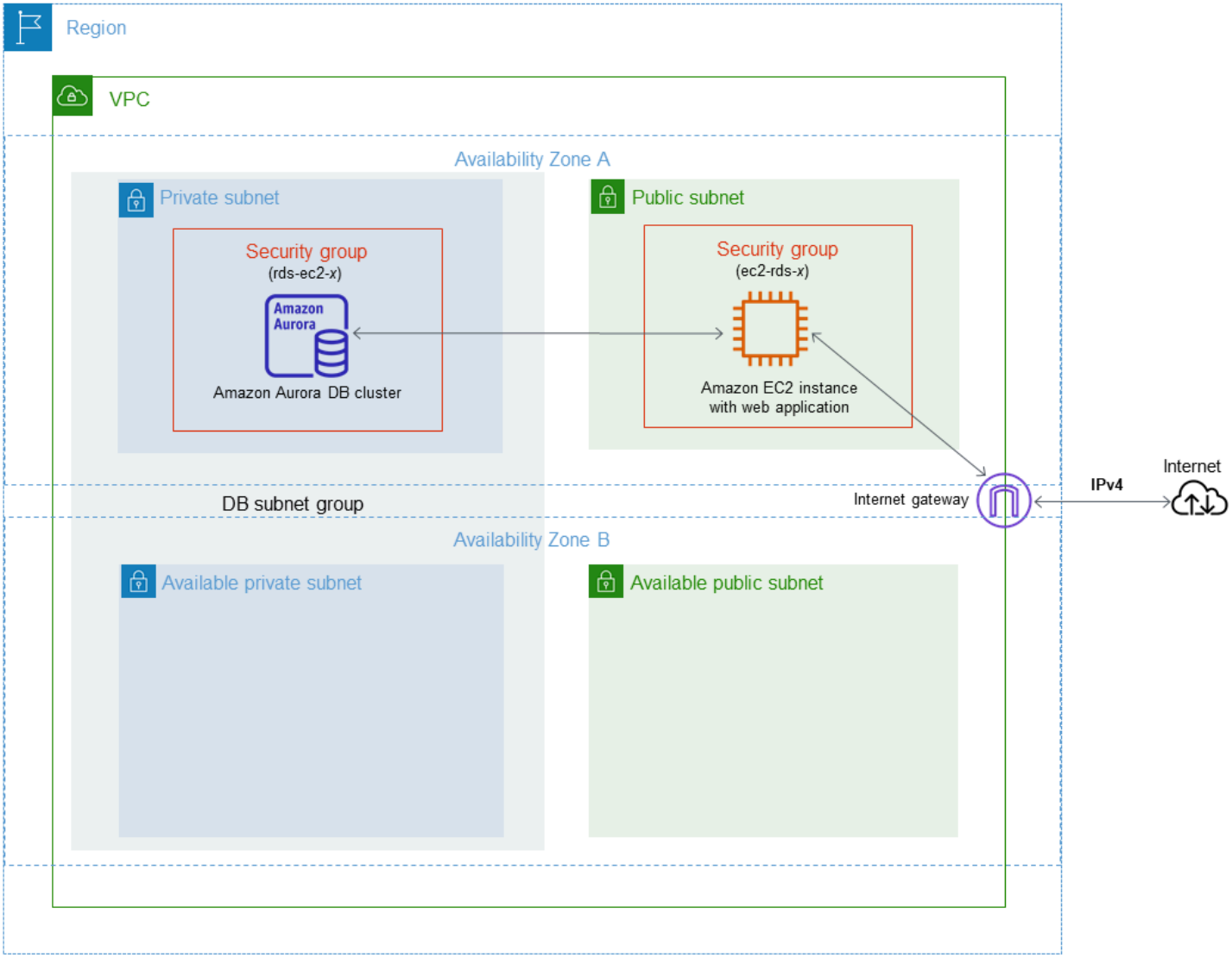
安全群組案例

VPC 中資料庫叢集的常見使用方式，是與在相同 VPC 的 Amazon EC2 執行個體中執行的應用程式伺服器共享資料，這是由 VPC 外的應用程式用戶端存取的。在此案例中，您可以使用 上的 RDS 和 VPC 頁面 AWS 管理主控台 或 RDS 和 EC2 API 操作來建立必要的執行個體和安全群組：

1. 建立 VPC 安全群組 (例如，`sg-0123ec2example`)，並定義使用用戶端應用程式之 IP 地址做為來源的傳入規則。此安全群組可讓您的用戶端應用程式連接至 VPC 中使用此安全群組的 EC2 執行個體。
2. 建立應用程式的 EC2 執行個體，並將 EC2 執行個體新增至您在前一個步驟中建立的 VPC 安全群組 (`sg-0123ec2example`)。
3. 建立第二個 VPC 安全群組 (例如，`sg-6789rdsexample`)，並建立新規則，方法為指定您在步驟 1 中建立的 VPC 安全群組 (`sg-0123ec2example`) 做為來源。

4. 建立新的資料庫叢集，並將資料庫叢集新增至您在前一個步驟中建立的 VPC 安全群組 (sg-6789rdsexample)。當建立資料庫叢集時，請使用您為步驟 3 中所建立 VPC 安全群組 (sg-6789rdsexample) 規則指定的相同連接埠號碼。

此案例可以下列圖表顯示。



如需此案例中設定 VPC 的詳細說明，請參閱 教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4)。如需使用 VPC 的詳細資訊，請參閱 Amazon VPC 和 Amazon Aurora。

建立 VPC 安全群組

您可以使用 VPC 主控台，建立資料庫執行個體的 VPC 安全群組。如需建立安全群組的相關資訊，請參閱《Amazon Virtual Private Cloud 使用者指南》中的建立安全群組以存取 VPC 中的資料庫叢集和安全群組。

將安全群組與資料庫叢集建立關聯

您可以在 RDS 主控台、`ModifyDBCluster`Amazon RDS API 或 `modify-db-cluster` AWS CLI 命令上使用修改叢集，將安全群組與資料庫叢集建立關聯。

下列 CLI 範例會建立特定 VPC 群組的關聯，並從資料庫叢集中移除資料庫安全群組

```
aws rds modify-db-cluster --db-cluster-identifier dbName --vpc-security-group-ids sg-ID
```

如需修改資料庫叢集的詳細資訊，請參閱 修改 Amazon Aurora 資料庫叢集。

主要使用者帳戶權限

在您建立新的資料庫叢集時，您使用的預設主要使用者會取得該資料庫叢集的特定權限。您無法在資料庫叢集建立之後變更主要使用者名稱。

> **Important**
>
> 我們強烈建議您不要直接在您的應用程式中使用主要使用者。而是遵循最佳實務，使用以應用程式所需的最低權限建立的資料庫使用者。

> **Note**
>
> 如果您不慎除主要使用者的許可，則可以透過修改資料庫叢集並設定新的主要使用者密碼來還原它們。如需修改資料庫叢集的詳細資訊，請參閱 修改 Amazon Aurora 資料庫叢集 。

下表顯示主要使用者針對每一個資料庫引擎取得的權限和資料庫角色。

資料庫引擎	系統權限	資料庫角色
Aurora MySQL	2 版： ALTER,ALTER ROUTINE, CREATE,CREATE ROUTINE,CREATE TEMPORARY TABLES,CREATE USER,CREATE VIEW,DELETE,DROP, EVENT,EXECUTE,GRANT OPTION,INDEX,INSERT, LOAD FROM S3,LOCK TABLES, PROCESS,REFERENCES , RELOAD,REPLICATI ON CLIENT , REPLICATION SLAVE ,SELECT, SELECT INTO S3,SHOW DATABASES , SHOW VIEW,TRIGGER, UPDATE	—
	第 3 版： ALTER,APPLICATION_PASSWORD_ADMIN , ALTER ROUTINE,CONNECTION_ADMIN , CREATE,CREATE ROLE,CREATE ROUTINE,CREATE TEMPORARY TABLES, CREATE USER,CREATE VIEW, DELETE,DROP,DROP ROLE, EVENT,EXECUTE,INDEX, INSERT,LOCK TABLES, PROCESS,REFERENCES , RELOAD,REPLICATION CLIENT , REPLICATI ON SLAVE ,ROLE_ADMIN , SET_USER_ ID ,SELECT,SHOW DATABASES ,SHOW VIEW,TRIGGER, UPDATE,XA_RECOVER_ADMIN 從 Aurora MySQL 3.04.0 版開始，主要使用者也會取得 SHOW_ROUTINE 權限。 從 Aurora MySQL 3.09.0 版開始，主要使用者也會取得 FLUSH_OPTIMIZER_COSTS 、FLUSH_STA TUS 、FLUSH_TABLES 和 FLUSH_USE R_RESOURCES 權限。	rds_superuser_role 如需 rds_superuser_role 的詳細資訊，請參閱 角色型權限模型。

資料庫引擎	系統權限	資料庫角色
Aurora PostgreSQL	`LOGIN,NOSUPERUSER ,INHERIT,CREATEDB, CREATEROLE ,NOREPLICATION ,VALID UNTIL 'infinity'`	`RDS_SUPERUSER` 如需 RDS_SUPERUSER 的詳細資訊，請參閱 了解 PostgreSQL 角色和許可。

使用 Amazon Aurora 的服務連結角色

Amazon Aurora 使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是直接連結至 Amazon Aurora 的一種特殊 IAM 角色類型。服務連結角色是由 Amazon Aurora 預先定義，並包含該服務代表您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓您更輕鬆使用 Amazon Aurora，因為您不需要手動新增必要許可。Amazon Aurora 定義其服務連結角色的許可，除非另有定義，否則僅有 Amazon Aurora 可以擔任其角色。定義的許可包括信任政策和許可政策，並且該許可政策不能附加到任何其他 IAM 實體。

您必須先刪除角色的相關資源，才能刪除角色。如此可保護您的 Amazon Aurora 資源，避免您不小心移除資源的存取許可。

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

Amazon Aurora 的服務連結角色許可

Amazon Aurora 使用名為 AWSServiceRoleForRDS 的服務連結角色，以允許 Amazon RDS 代表資料庫叢集呼叫 AWS 服務。

AWSServiceRoleForRDS 服務連結角色信任下列服務可擔任該角色：

- `rds.amazonaws.com`

此服務連結角色具有名為 `AmazonRDSServiceRolePolicy` 的許可政策，該政策會授予此角色在帳戶中操作的許可。

如需有關此政策的詳細資訊，包括 JSON 政策文件，請參閱《AWS 受管政策參考指南》中的 AmazonRDSServiceRolePolicy。

> **Note**
>
> 您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如果您遇到下列錯誤訊息：
> 無法建立資源。請確認您擁有建立服務連結角色的許可。否則請等待，然後再試一次。
> 請確定您已啟用下列許可：
>
> ```
> {
> ```

```
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName":"rds.amazonaws.com"
        }
    }
}
```

如需詳細資訊，請參閱 IAM 使用者指南中的服務連結角色許可。

為 Amazon Aurora 建立服務連結角色

您不需要手動建立一個服務連結角色。當您建立資料庫叢集時，Amazon Aurora 會為您建立服務連結角色。

> **Important**
>
> Amazon Aurora 自 2017 年 12 月 1 日開始支援服務連結角色，若您在這之前就有使用此服務，則 Amazon Aurora 會在帳戶中建立 AWSServiceRoleForRDS 角色。若要進一步了解，請參閱我的 AWS 帳戶中出現的新角色。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。當您建立資料庫叢集時，Amazon Aurora 會再次為您建立服務連結角色。

為 Amazon Aurora 編輯服務連結角色

Amazon Aurora 不允許您編輯 AWSServiceRoleForRDS 服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的編輯服務連結角色。

為 Amazon Aurora 刪除服務連結角色

若您不再使用需要服務連結角色的功能或服務，我們建議您刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。但是，您必須先刪除所有資料庫叢集，才能刪除服務連結角色。

清除服務連結角色

您必須先確認服務連結角色沒有作用中的工作階段，並移除該角色使用的資源，之後才能使用 IAM 將其刪除。

檢查服務連結角色是否於 IAM 主控台有作用中的工作階段

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在 IAM 主控台的導覽窗格中，選擇角色。然後選擇 AWSServiceRoleForRDS 角色的名稱 (而非核取方塊)。
3. 在所選角色的摘要頁面中，選擇上次存取索引標籤。
4. 在上次存取索引標籤上，檢閱服務連結角色的近期活動。

> **Note**
>
> 如果您不確定 Amazon Aurora 是否正在使用 AWSServiceRoleForRDS 角色，可嘗試刪除該角色。如果服務正在使用該角色，則刪除會失敗，而您可以檢視正在使用該角色的 AWS 區域。如果服務正在使用該角色，您必須先等到工作階段結束，才能刪除該角色。您無法撤銷服務連結角色的工作階段。

如果您想要移除 AWSServiceRoleForRDS 角色，首先務必要刪除所有資料庫叢集。

刪除您的所有叢集

使用下列其中一個程序來刪除單一叢集。對您的每一個叢集重複此程序。

刪除叢集 (主控台)

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在 Databases (資料庫) 清單中，選擇您要刪除的叢集。
3. 對於 Cluster Actions (叢集動作)，請選擇 Delete (刪除)。
4. 選擇 刪除。

刪除叢集 (CLI)

請參閱 AWS CLI 命令參考中的 `delete-db-cluster`。

刪除叢集 (API)

請參閱 Amazon RDS API 參考中的 `DeleteDBCluster`。

使用 IAM 主控台、IAM CLI 或 IAM API 刪除 AWSServiceRoleForRDS 服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的刪除服務連結角色。

Amazon RDS Beta 的服務連結角色許可

Amazon Aurora 使用名為 的服務連結角色`AWSServiceRoleForRDSBeta`，以允許 Amazon Aurora 代表 RDS 資料庫資源呼叫 AWS 服務。

AWSServiceRoleForRDSBeta 服務連結角色信任下列服務可擔任該角色：

- `rds.amazonaws.com`

此服務連結角色具有名為 AmazonRDSBetaServiceRolePolicy 的許可政策，該政策會授予此角色在帳戶中操作的許可。如需詳細資訊，請參閱AWS 受管政策：AmazonRDSBetaServiceRolePolicy。

> **Note**
>
> 您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如果您遇到下列錯誤訊息：
> 無法建立資源。請確認您擁有建立服務連結角色的許可。否則請等待，然後再試一次。
> 請確定您已啟用下列許可：
>
> ```
> {
>     "Action": "iam:CreateServiceLinkedRole",
>     "Effect": "Allow",
>     "Resource": "arn:aws:iam::*:role/aws-service-role/custom.rds.amazonaws.com/
> AmazonRDSBetaServiceRolePolicy",
>     "Condition": {
>         "StringLike": {
>             "iam:AWSServiceName":"custom.rds.amazonaws.com"
>         }
>     }
> }
> ```
>
> 如需詳細資訊，請參閱 IAM 使用者指南中的服務連結角色許可。

Amazon RDS 預覽的服務連結角色

Amazon Aurora 使用名為 的服務連結角色AWSServiceRoleForRDSPreview，以允許 Amazon Aurora 代表 RDS 資料庫資源呼叫 AWS 服務。

AWSServiceRoleForRDSPreview 服務連結角色信任下列服務擔任該角色：

- rds.amazonaws.com

此服務連結角色具有名為 AmazonRDSPreviewServiceRolePolicy 的許可政策，該政策會授予此角色在帳戶中操作的許可。如需詳細資訊，請參閱AWS 受管政策：AmazonRDSPreviewServiceRolePolicy。

> **Note**
>
> 您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如果您遇到下列錯誤訊息：
> 無法建立資源。請確認您擁有建立服務連結角色的許可。否則請等待，然後再試一次。
> 請確定您已啟用下列許可：
>
> ```
> {
>     "Action": "iam:CreateServiceLinkedRole",
>     "Effect": "Allow",
>     "Resource": "arn:aws:iam::*:role/aws-service-role/custom.rds.amazonaws.com/
> AmazonRDSPreviewServiceRolePolicy",
>     "Condition": {
>         "StringLike": {
>             "iam:AWSServiceName":"custom.rds.amazonaws.com"
>         }
>     }
> }
> ```
>
> 如需詳細資訊，請參閱 IAM 使用者指南中的服務連結角色許可。

Amazon VPC 和 Amazon Aurora

Amazon Virtual Private Cloud (Amazon VPC) 可讓您在 Virtual Private Cloud (VPC) 中啟動 AWS 資源，例如 Aurora DB 叢集。

使用 VPC 時，您可以掌控您的虛擬聯網環境。您可以選擇自己的 IP 地址範圍、建立子網路，以及設定路由和存取控制清單。在 VPC 中執行資料庫叢集無需額外成本。

帳戶有預設 VPC。系統會在預設 VPC 中建立所有新的資料庫叢集，除非您另外指定。

主題

- 在 VPC 中使用資料庫叢集
- 在 VPC 中存取資料庫叢集的案例
- 教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4)
- 教學課程：建立要與資料庫叢集搭配使用的 (VPC)(雙堆疊模式)

在以下內容中，您可以找到 Amazon Aurora DB 叢集的 VPC 功能相關的討論。如需 Amazon VPC 的詳細資訊，請參閱 Amazon VPC 入門指南和 Amazon VPC 使用者指南。

在 VPC 中使用資料庫叢集

您的資料庫叢集位在虛擬私有雲端 (VPC) 內。VPC 是與 AWS 雲端中其他虛擬網路邏輯隔離的虛擬網路。Amazon VPC 可讓您在 VPC 中啟動 AWS 資源，例如 Amazon Aurora 資料庫叢集或 Amazon EC2 執行個體。VPC 可以是您帳戶隨附的預設 VPC，或是您自行建立的 VPC。所有 VPCs都與 AWS 您的帳戶相關聯。

您的預設 VPC 有三個子網路，可供您在 VPC 內隔離資源。預設的 VPC 也有網際網路閘道，只要利用網際網路閘道，即可允許從 VPC 以外的位置存取 VPC 內的資源。

如需涉及 VPC 內 Amazon Aurora 資料庫叢集的藍本清單，請參閱 在 VPC 中存取資料庫叢集的案例。

主題

- 在 VPC 中使用資料庫叢集
- VPC 加密控制
- 使用資料庫子網路群組
- 共用子網路
- Amazon Aurora IP 定址

- 在 VPC 中的網際網路中隱藏資料庫叢集
- 在 VPC 中建立資料庫叢集

於下列教學課程中，您可學習建立可用於常見 Amazon Aurora 案例的 VPC：

- 教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4)
- 教學課程：建立要與資料庫叢集搭配使用的 (VPC)(雙堆疊模式)

在 VPC 中使用資料庫叢集

在 VPC 中使用資料庫叢集的要訣如下：

- 您的 VPC 必須至少要有兩個子網路。這些子網路必須位於 AWS 區域 您要部署資料庫叢集的 中的兩個不同可用區域。子網路是指可供您指定之 VPC 的 IP 地址範圍區段，您可根據安全與運作需求將資料庫叢集分組。
- 如果您要開放資料庫叢集在 VPC 中供公開存取，須啟用 VPC 屬性 DNS hostnames (DNS 主機名稱) 和 DNS resolution (DNS 解析)。
- 您的 VPC 必須具備您所建立的資料庫子網路群組。您可以透過指定您所建立的子網路來建立資料庫子網路群組。從資料庫子網路群組中，Amazon Aurora 會選擇子網路和該子網路內的 IP 地址，以便與您的資料庫叢集中的主要資料庫執行個體建立關聯。主要資料庫執行個體使用包含子網路的可用區域。
- 您的 VPC 必須具備能允許存取資料庫叢集的 VPC 安全群組。

  如需詳細資訊，請參閱在 VPC 中存取資料庫叢集的案例。
- 您每個子網路中的 CIDR 區塊大小必須足以容納備用 IP 地址，以供 Amazon Aurora 在維護活動 (包括容錯移轉與運算擴展) 期間使用。例如，10.0.0.0/24 和 10.0.1.0/24 等範圍通常就夠大了。
- VPC 的執行個體租用屬性可以是 default (預設) 或 dedicated (專用)。所有預設 VPC 已將執行個體租用屬性設為預設，且預設的 VPC 可支援任何資料庫執行個體類別。

  若您選擇讓資料庫叢集位於執行個體租用屬性設為「專用」的專屬 VPC 中，資料庫叢集的資料庫執行個體類別必須選擇獲核准的 Amazon EC2 專用預留執行個體類型。舉例來說，r5.large EC2 專用預留執行個體對應到 db.r5.large 資料庫執行個體類別。如需 VPC 中執行個體租用的相關資訊，請參閱《Amazon Elastic Compute Cloud 使用者指南》中的專用執行個體。

  如需進一步了解專用執行個體中可能出現的執行個體類型，請參閱 Amazon EC2 定價頁面上的 Amazon EC2 專用執行個體。

> **Note**
>
> 將執行個體租用屬性設定為專用於資料庫叢集時，並不保證資料庫叢集會在專用執行個體上執行。

VPC 加密控制

VPC 加密控制可讓您針對 VPCs 內的所有網路流量強制執行encryption-in-transit。透過確保在指定的 VPCs 中只能佈建具備加密功能的 Nitro 型硬體，來使用加密控制來符合法規合規要求。加密控制也會在 API 請求時間攔截相容性問題，而不是在佈建期間。您現有的工作負載會繼續運作，而且只會封鎖新的不相容請求。

將 VPC 控制模式設定為 ，以設定您的 VPC 加密控制：

- 停用 （ 預設 ）
- 監控
- 強制執行

若要檢查 VPC 目前的控制模式，請使用 AWS 管理主控台 或 DescribeVpcs CLI 或 API 命令。

如果您的 VPC 強制執行加密，您只能佈建 Nitro 型資料庫叢集，在該 VPC 中支援傳輸中加密。如需詳細資訊，請參閱 資料庫執行個體類別的類型。如需 Nitro 執行個體的相關資訊，請參閱《Amazon EC2 使用者指南》中的在 AWS Nitro 系統上建置的執行個體。 Amazon EC2

> **Note**
>
> 如果您嘗試在加密強制執行的 VPC 中佈建不相容的資料庫叢集，Aurora 會傳回`VpcEncryptionControlViolationException`例外狀況。

對於 Aurora Serverless for MySQL 和 PostreSQL，加密控制需要平台版本 3 或更高版本。

使用資料庫子網路群組

Subnets (子網路) 是可供您指定之 VPC 的 IP 地址範圍區段，您可根據安全與運作需求將資源分組。資料庫子網路群組是一種子網路集合 (通常為私有)，您必須先在 VPC 中建立這些群組，然後指定群組

供您的資料庫叢集使用。透過使用資料庫子網路群組，您可以在使用 AWS CLI 或 RDS API 建立資料庫叢集時指定特定 VPC。如果使用主控台，您可以選擇要使用的 VPC 和子網路群組。

各個資料庫子網路群組在指定的 AWS 區域中，皆應在至少兩個可用區域中具有子網路。在 VPC 中建立資料庫叢集時，您必須為其選擇資料庫子網路群組。從資料庫子網路群組中，Amazon Aurora 會選擇子網路和該子網路內的 IP 地址，以便與您的資料庫叢集中的主要資料庫執行個體建立關聯。資料庫使用包含子網路的可用區域。Aurora 一律會從具有可用 IP 位址空間的子網路指派 IP 位址。

資料庫子網路群組中的子網路可設為公開或私有，視您針對子網路的網路存取控制清單 (網路 ACL) 和路由表所設定的組態而定，子網路為公開或私有。若要使資料庫叢集可公開存取，其資料庫子網路群組中的所有子網路皆必須為公有。若與可公開存取的資料庫叢集關聯的子網路從公有變更為私有，則可能會影響資料庫叢集可用性。

如要建立支援雙堆疊模式的資料庫子網路群組，請確保新增至資料庫子網路群組的每個子網路皆具有與其關聯的網際網路通訊協定第 6 版 (IPv6) CIDR 區塊。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 Amazon Aurora IP 定址 和遷移至 IPv6。

Amazon Aurora 在 VPC 中建立資料庫叢集時，會使用來自您資料庫子網路群組的 IP 地址，將網路介面指派給您的資料庫叢集。然而，我們強烈建議您使用網域名稱系統 (DNS) 名稱，與您的資料庫叢集連線。我們如此建議，是因為基本的 IP 地址在容錯移轉期間會有所變動。

> **Note**
>
> 在資料庫子網路群組的每個子網路中，請確保為 VPC 中執行的每個資料庫叢集保留至少一個地址，以供 Amazon Aurora 執行復原動作。

共用子網路

您可以在共用 VPC 中建立資料庫叢集。

使用共用 VPC 時要記住的一些考量：

- 您可以將資料庫叢集從共用 VPC 子網路移至非共用 VPC 子網路，反之亦然。
- 共用 VPC 的參與者必須在 VPC 中建立安全群組，才能允許他們建立資料庫叢集。
- 共用 VPC 中的擁有者和參與者可以使用 SQL 查詢存取資料庫。不過，只有資源的建立者才能對資源進行任何 API 呼叫。

Amazon Aurora IP 定址

IP 地址可讓您 VPC 中的資源彼此互相通訊，也能和網際網路上的資源通訊。Amazon Aurora 同時支援 IPv4 和 IPv6 定址通訊協定。依預設，Amazon Aurora 和 Amazon VPC 使用 IPv4 定址通訊協定。您無法關閉此行為。當您建立 VPC 時，請務必指定 IPv4 CIDR 區塊 (私有 IPv4 地址的範圍)。您可選擇將 IPv6 CIDR 區塊指派給 VPC 和子網路，並將 IPv6 地址從該區塊指派給子網路中的資料庫叢集。

對 IPv6 通訊協定的支援擴展受支援的 IP 地址的數量。使用 IPv6 通訊協定，您可以確保擁有足夠的可用地址，可應對網際網路的未來發展。新的和現有的 RDS 資源可於您的 VPC 中使用 IPv4 和 IPv6 地址。在應用程式的不同部分中使用的兩個通訊協定之間進行設定、保護和轉譯網路流量可能會造成操作額外負荷。您可在 IPv6 通訊協定上對 Amazon RDS 資源進行標準化，以簡化您的網路組態。

主題

- IPv4 地址
- IPv6 地址
- 雙堆疊模式

IPv4 地址

在您建立 VPC 時，必須以 CIDR 區塊的形式為 VPC 指定 IPv4 地址的範圍，例如 `10.0.0.0/16`。資料庫子網路群組定義資料庫叢集可以使用之此 CIDR 區塊中的 IP 地址範圍。這些 IP 地址可為私有或公有。

私有 IPv4 地址是無法在網際網路存取的 IP 地址。您可使用私有 IPv4 地址，在相同 VPC 中的資料庫叢集及其他資源 (例如 Amazon EC2 執行個體) 之間進行通訊。每個資料庫叢集具有用於在 VPC 中進行通訊的私有 IP 地址。

公有 IP 地址是可從網際網路存取的 IPv4 地址。您可使用公有地址，在資料庫叢集和網際網路上的資源 (例如 SQL 用戶端) 之間進行通訊。您控制您的資料庫叢集是否接收公有 IP 地址。

Amazon RDS 會將來自 EC2 公有 IPv4 地址集區的公有彈性 IPv4 地址用於可公開存取的資料庫執行個體。使用 CLI、API `describe-addresses` 或檢視 中的彈性 IPs(EIP) 區段 AWS 時，您的帳戶中會顯示這些 IP 地址 AWS 管理主控台。每個 RDS 受管 IP 地址都會以設定為 的`service_managed`屬性標示`"rds"`。

雖然您的帳戶中可以看到這些 IPs，但它們仍由 Amazon RDS 完全管理，且無法修改或釋出。當不再使用時，Amazon RDS 會將 IPs 發行回公有 IPv4 地址集區。

CloudTrail 會記錄與 RDS EIP 相關的 API 呼叫，例如 `AllocateAddress`。服務委託人 會叫用這些 API 呼叫`rds.amazonaws.com`。

> **Note**
>
> Amazon RDS 配置的 IPs 不會計入您帳戶的 EIP 限制。

如需示範如何建立僅包含私有 IPv4 地址以用於常見 Amazon Aurora 案例之 VPC 的教學課程，請參閱教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4)。

IPv6 地址

您可以選擇性的將 IPv6 CIDR 區塊與您的 VPC 和子網路建立關聯，並指派該區塊的 IPv6 地址給您 VPC 中的資源。每個 IPv6 地址都是全域唯一的。

您的 VPC 的 IPv6 CIDR 區塊會自動從 Amazon 的 IPv6 地址集區自動指派。您無法自行選擇範圍。

連線至 IPv6 地址時，請確保符合下列條件：

- 用戶端已設定為允許透過 IPv6 進行用戶端至資料庫的流量。
- 資料庫執行個體使用的 RDS 安全群組已正確設定，允許透過 IPv6 進行用戶端至資料庫的流量。
- 用戶端作業系統堆疊允許 IPv6 地址上的流量，且作業系統驅動程式和程式庫會進行設定以選擇正確的預設資料庫執行個體端點 (IPv4 或 IPv6)。

如需 IPv6 的詳細資訊，請參閱《Amazon VPC 使用者指南》中的 IP 定址。

雙堆疊模式

資料庫叢集可在透過 IPv4 和 IPv6 定址通訊協定進行通訊時，以雙堆疊模式執行。然後，資源可以使用 IPv4, IPv6 或兩種通訊協定與資料庫叢集通訊。私有雙堆疊模式資料庫執行個體的 IPv6 端點限制為僅限 VPC 存取，可確保您的 IPv6 端點保持私有。公有雙堆疊模式資料庫執行個體提供 IPv4 和 IPv6 端點，您可以從網際網路存取。

主題

- 雙堆疊模式和資料庫子網路群組
- 使用雙堆疊模式資料庫執行個體
- 將僅限 IPv4 的資料庫叢集修改為使用雙堆疊模式
- 雙堆疊網路資料庫叢集的可用性

- 雙堆疊網路資料庫叢集的限制

如需示範如何建立包含 IPv4 和 IPv6 兩個地址以用於常見 Amazon Aurora 案例之 VPC 的教學課程，請參閱 教學課程：建立要與資料庫叢集搭配使用的 (VPC)(雙堆疊模式)。

雙堆疊模式和資料庫子網路群組

如要使用雙堆疊式，請確保與資料庫叢集關聯之資料庫子網路群組中的每個子網路皆有一個與之關聯的 IPv6 CIDR 區塊。您可建立新的資料庫子網路群組或修改現有資料庫子網路群組，以滿足此要求。於資料庫 叢集處於雙堆疊模式後，用戶端可進行正常連線。確保客戶端安全防火牆和 RDS 資料庫執行個體安全群組已準確設定，允許透過 IPv6 的流量。如要進行連線，用戶端會使用資料庫叢集主執行個體的端點。用戶端應用程式可指定連線至資料庫時所偏好的通訊協定。於雙堆疊模式下，資料庫叢集會偵測用戶端的偏好網路通訊協定 (IPv4 或 IPv6)，並將該通訊協定用於連線。

若資料庫子網路群組因子網路刪除或 CIDR 斷開關聯而停止支援雙堆疊模式，則與資料庫子網路群組相關聯的資料庫執行個體存在網路狀態不相容的風險。此外，建立新的雙堆疊模式資料庫叢集時，無法使用資料庫子網路群組。

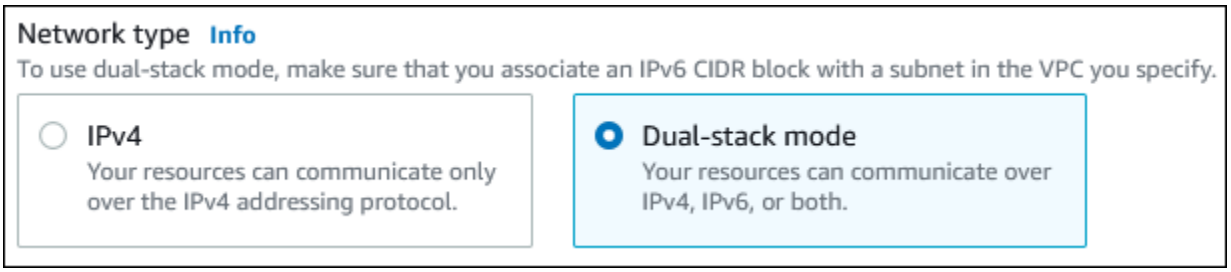
若要使用 判斷資料庫子網路群組是否支援雙堆疊模式 AWS 管理主控台，請在資料庫子網路群組的詳細資訊頁面上檢視網路類型。若要使用 判斷資料庫子網路群組是否支援雙堆疊模式 AWS CLI，請執行 describe-db-subnet-groups 命令並在輸出SupportedNetworkTypes中檢視 。

將僅供讀取複本視為獨立的資料庫執行個體，且可具有不同於主資料庫執行個體的網路類型。若您變更僅供讀取複本之主資料庫執行個體的網路類型，則僅供讀取複本不會受到影響。還原資料庫執行個體時，您可將其還原為支援的任何網路類型。

使用雙堆疊模式資料庫執行個體

當您建立或修改資料庫叢集時，您可指定雙堆疊模式，以允許您的資源透過 IPv4、IPv6 或兩者與資料庫叢集進行通訊。

當您使用 AWS 管理主控台 建立或修改資料庫執行個體時，您可以在網路類型區段中指定雙堆疊模式。下圖顯示主控台中的 Network type (網路類型) 區段。



當您使用 AWS CLI 建立或修改資料庫叢集時，請將 `--network-type`選項設定為DUAL使用雙堆疊模式。當您使用 RDS API 建立或修改資料庫叢集時，請將 `NetworkType` 參數設定為 `DUAL`，以使用雙堆疊模式。當您修改資料庫執行個體的網路類型時，可能會出現停機時間。若指定的資料庫引擎版本或資料庫子網路群組不支援雙堆疊模式，則會傳回 `NetworkTypeNotSupported` 錯誤。

如需建立資料庫叢集的詳細資訊，請參閱建立 Amazon Aurora 資料庫叢集。如需修改資料庫叢集的詳細資訊，請參閱修改 Amazon Aurora 資料庫叢集。

若要使用主控台來決定資料庫叢集是否處於雙堆疊模式，請檢視資料庫叢集 Connectivity & security (連線能力與安全性) 索引標籤上的 Network type (網路類型)。

將僅限 IPv4 的資料庫叢集修改為使用雙堆疊模式

您可將僅限 IPv4 的資料庫叢集修改為使用雙堆疊模式。如此，請變更資料庫叢集的網路類型。修改可能會造成停機。

建議您在維護時段變更 Amazon Aurora 資料庫叢集的網路類型。目前不支援將新執行個體的網路類型設定為雙堆疊模式。您可以使用 `modify-db-cluster` 命令手動設定網路類型。

將資料庫叢集修改為使用雙堆疊模式之前，請確保其資料庫子網路群組支援雙堆疊模式。若與資料庫叢集關聯的資料庫子網路群組不支援雙堆疊模式，請於修改資料庫叢集時指定支援該子網路群組的其他資料庫子網路群組。修改資料庫叢集的資料庫子網路群組可能會導致停機。

若在將資料庫叢集變更為使用雙堆疊模式之前修改資料庫叢集的資料庫子網路群組，請確保該資料庫子網路群組在變更前後對資料庫叢集有效。

我們建議您僅搭配值為 `DUAL` 的 `--network-type` 參數執行 modify-db-cluster API，以將 Amazon Aurora 叢集的網路變更為雙堆疊模式。在相同的 API 呼叫中新增其他參數和 `--network-type` 參數可能會導致停機時間。

若於變更後無法連線至資料庫叢集，請確認用戶端和資料庫安全防火牆和路由表已正確設定，以允許流量傳輸至選定網路上的資料庫 (IPv4 或 IPv6)。您可能還需要修改作業系統參數、程式庫或驅動程式才可使用 IPv6 地址進行連線。

如要將僅限 IPv4 的資料庫叢集修改為使用雙堆疊模式

1. 修改資料庫子網路群組以支援雙堆疊模式，或建立支援雙堆疊模式的資料庫子網路群組：

    a. 建立 IPv6 CIDR 區塊與 VPC 的關聯。

        如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的將 IPv6 CIDR 區塊新增至 VPC。

b. 將 IPv6 CIDR 區塊連接至資料庫子網路群組中的所有子網路。

如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的將 IPv6 CIDR 區塊新增至子網路。

c. 確認資料庫子網路群組支援雙堆疊模式。

如果您使用的是 AWS 管理主控台，請選取資料庫子網路群組，並確認支援的網路類型值為雙 IPv4。

如果您使用的是 AWS CLI，請執行 describe-db-subnet-groups 命令，並確認資料庫執行個體`SupportedNetworkType`的值為 `Dual, IPv4`。

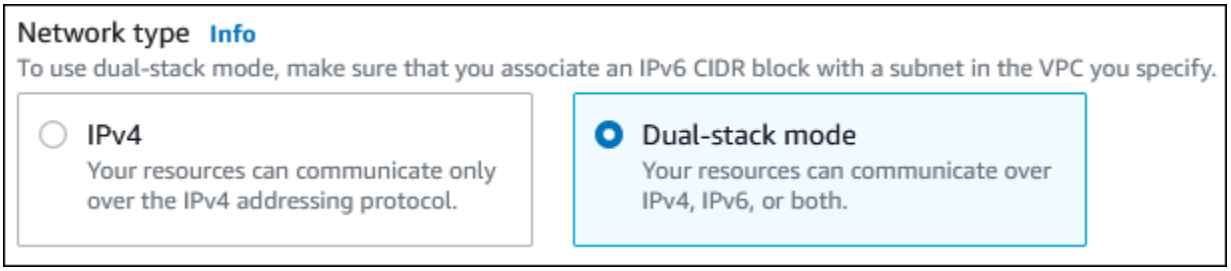
2. 修改與資料庫叢集關聯的安全群組，以允許 IPv6 連線至資料庫，或建立允許 IPv6 連線的新安全群組。

如需說明，請參閱《Amazon VPC 使用者指南》中的安全群組規則。

3. 修改資料庫叢集以支援雙堆疊模式。若要這麼做，請將 Network type (網路類型) 設為 Dual-stack mode (雙堆疊模式)。

若您使用主控台，請確保下列設定正確：

- Network type (網路類型) – Dual-stack mode (雙堆疊模式)



- DB subnet group (資料庫子網路群組) – 您在前一步驟中設定的資料庫子網路群組
- Security group (安全群組) – 您在上一個步驟中設定的安全性

如果您使用的是 AWS CLI，請確定下列設定正確：

- `--network-type` – `dual`
- `--db-subnet-group-name` – 您在前一步驟中設定的資料庫子網路群組
- `--vpc-security-group-ids` – 您在前一步驟中設定的 VPC 安全群組

例如：

```
aws rds modify-db-cluster --db-cluster-identifier my-cluster --network-type "DUAL"
```

4. 確認資料庫叢集是否支援雙堆疊模式。

   若您使用主控台，請選擇資料庫叢集的 Configuration (連線能力與安全性) (組態) 索引標籤。在該索引標籤，確保 Network type (網路類型) 值為 Dual-stack mode (雙堆疊模式)。

   如果您使用的是 AWS CLI，請執行 describe-db-clusters 命令，並確認資料庫叢集`NetworkType`的值為 `dual`。

   執行寫入器資料庫執行個體端點上的 `dig` 命令來辨識與其關聯的 IPv6 地址。

   ```
   dig db-instance-endpoint AAAA
   ```

   使用寫入器資料庫執行個體端點 (非 IPv6 地址)，連線至資料庫叢集。

雙堆疊網路資料庫叢集的可用性

除了以下項目 AWS 區域 之外，雙堆疊網路資料庫叢集可在所有 中使用：

- 亞太地區 (海德拉巴)
- 亞太地區 (馬來西亞)
- 亞太地區 (墨爾本)
- 亞太區域 (泰國)
- 加拿大西部 (卡加利)
- 歐洲 (西班牙)
- 歐洲 (蘇黎世)
- 以色列 (特拉維夫)
- 墨西哥 (中部)
- 中東 (阿拉伯聯合大公國)

下列資料庫引擎版本支援雙堆疊網路資料庫叢集：

- Aurora MySQL 版本：
  - 3.02 和更新的 3 版本

  - 2.09.1 和更新的 2 版本

  如需 Aurora MySQL 版本的更多資訊，請參閱 Aurora MySQL 版本備註。
- Aurora PostgreSQL 版本：
  - 15.2 版和所有更新版本
  - 14.3 版和更新的 14 版
  - 13.7 和更新的第 13 版

  如需 Aurora PostgreSQL 版本的更多資訊，請參閱 Aurora PostgreSQL 版本備註。

雙堆疊網路資料庫叢集的限制

下列限制適用於雙堆疊網路資料庫叢集：

- 資料庫叢集無法僅使用 IPv6 通訊協定。其可專門使用 IPv4，也可使用 IPv4 和 IPv6 通訊協定 (雙堆疊模式)。
- Amazon RDS 不支援原生 IPv6 子網路。
- 您無法將 RDS Proxy 和雙堆疊模式資料庫叢集搭配使用。

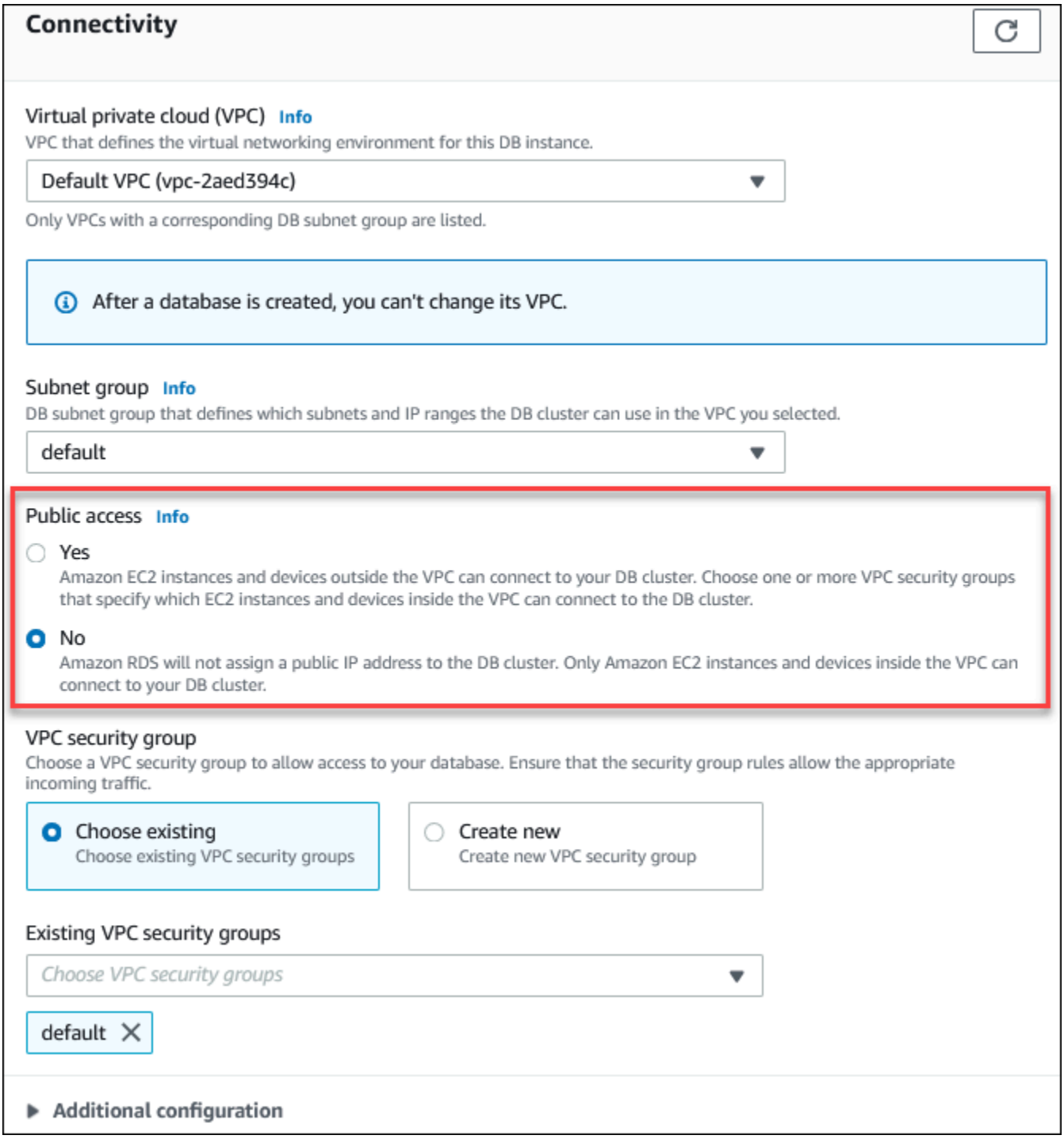
在 VPC 中的網際網路中隱藏資料庫叢集

VPC 中，有個 Amazon EC2 執行個體上有開放存取的 Web 應用程式，同時資料庫叢集上有不可公開存取的資料庫，這樣的 Amazon Aurora 藍本並不罕見。例如，您可以建立具有公有子網路和私有子網路的 VPC。作為 Web 伺服器的 EC2 執行個體可部署於公有子網路中，而資料庫叢集則可部署於私有子網路中。在這樣的部署下，只有 Web 伺服器可以存取資料庫叢集。如需這種案例的圖示，請參閱 由相同 VPC 中的 Amazon EC2 執行個體在 VPC 中存取的資料庫叢集。

當您在 VPC 內啟動資料庫叢集時，資料庫叢集在 VPC 內具有流量的私人的 IP 地址。此私人 IP 地址無法公開存取。您可以使用 Public access (公用存取) 選項來指定除了私有 IP 地址之外，資料庫叢集是否還具有公有 IP 地址。如果指定資料庫叢集為可公開存取，則其 DNS 端點會從 VPC 內解析為私人 IP 地址。它會從 VPC 外部解析為公用 IP 地址。資料庫叢集的存取權限最終是由其使用的安全群組所控制。若指派給該資料庫叢集的安全群組沒有包含允許的傳入規則，則該資料庫執行個體就無法開放供公開存取。若要開放公開存取資料庫叢集，其資料庫子網路群組中的子網路必須具備網際網路閘道。如需詳細資訊，請參閱 無法連線至 Amazon RDS 資料庫執行個體

您可以修改 Public access (公用存取) 選項，藉此修改資料庫叢集以開啟或關閉公開存取性。下圖顯示 Additional connectivity configuration (其他連線能力組態) 區段中的 Public access (公用存取) 選項。若

要設定此選項，請開啟 Connectivity (連線能力) 區段中的 Additional connectivity configuration (其他連線能力組態) 區段。



如需有關修改資料庫執行個體以設定 Public access (公用存取) 選項的資訊，請參閱修改資料庫叢集中的資料庫執行個體。

在 VPC 中建立資料庫叢集

下列步驟可協助您在 VPC 中建立資料庫叢集。若要使用預設 VPC，您可以從步驟 2 開始，使用系統已為您建立的 VPC 和資料庫子網路群組。您想要建立額外的 VPC 的話，可以建立新的 VPC。

> **Note**
>
> 如果您要開放資料庫叢集在 VPC 中供公開存取，須啟用 VPC 屬性 DNS hostnames (DNS 主機名稱) 和 DNS resolution (DNS 解析)，更新 VPC 的 DNS 資訊。若要進一步了解如何更新 VPC 執行個體的 DNS 資訊，請參閱更新 VPC 的 DNS 支援。

請依照以下步驟在 VPC 中建立資料庫執行個體：

- 步驟 1：建立 VPC
- 步驟 2：建立資料庫子網路群組
- 步驟 3：建立 VPC 安全群組
- 步驟 4：在 VPC 中建立資料庫執行個體

步驟 1：建立 VPC

建立在至少兩個可用區域內有子網路的 VPC。建立資料庫子網路群組時，您會需要使用這些子網路。如有預設的 VPC，則系統會自動為您於所在 AWS 區域的每個可用區域中建立子網路。

如需詳細資訊，請參閱 建立含私有和公有子網路的 VPC 或 Amazon VPC 使用者指南中的建立 VPC。

步驟 2：建立資料庫子網路群組

資料庫子網路群組是一種子網路集合 (通常為私有)，您必須先為 VPC 建立這些群組，然後指定給資料庫叢集使用。資料庫子網路群組可讓您在使用 AWS CLI 或 RDS API 建立資料庫叢集時指定特定 VPC。若使用主控台，只需選擇要使用的 VPC 和子網路即可。各個資料庫子網路群組在 AWS 區域中，須至少於兩個可用區域中具有至少一個子網路。作為最佳實務，每個資料庫子網路群組應於 AWS 區域中，為每個可用區域擁有至少有一個子網路。

若要開放資料庫叢集供公開存取，資料庫子網路群組中的子網路必須具備網際網路閘道。如需子網路網際網路閘道的詳細資訊，請參閱 Amazon VPC 使用者指南中的使用網際網路閘道連線至網際網路。

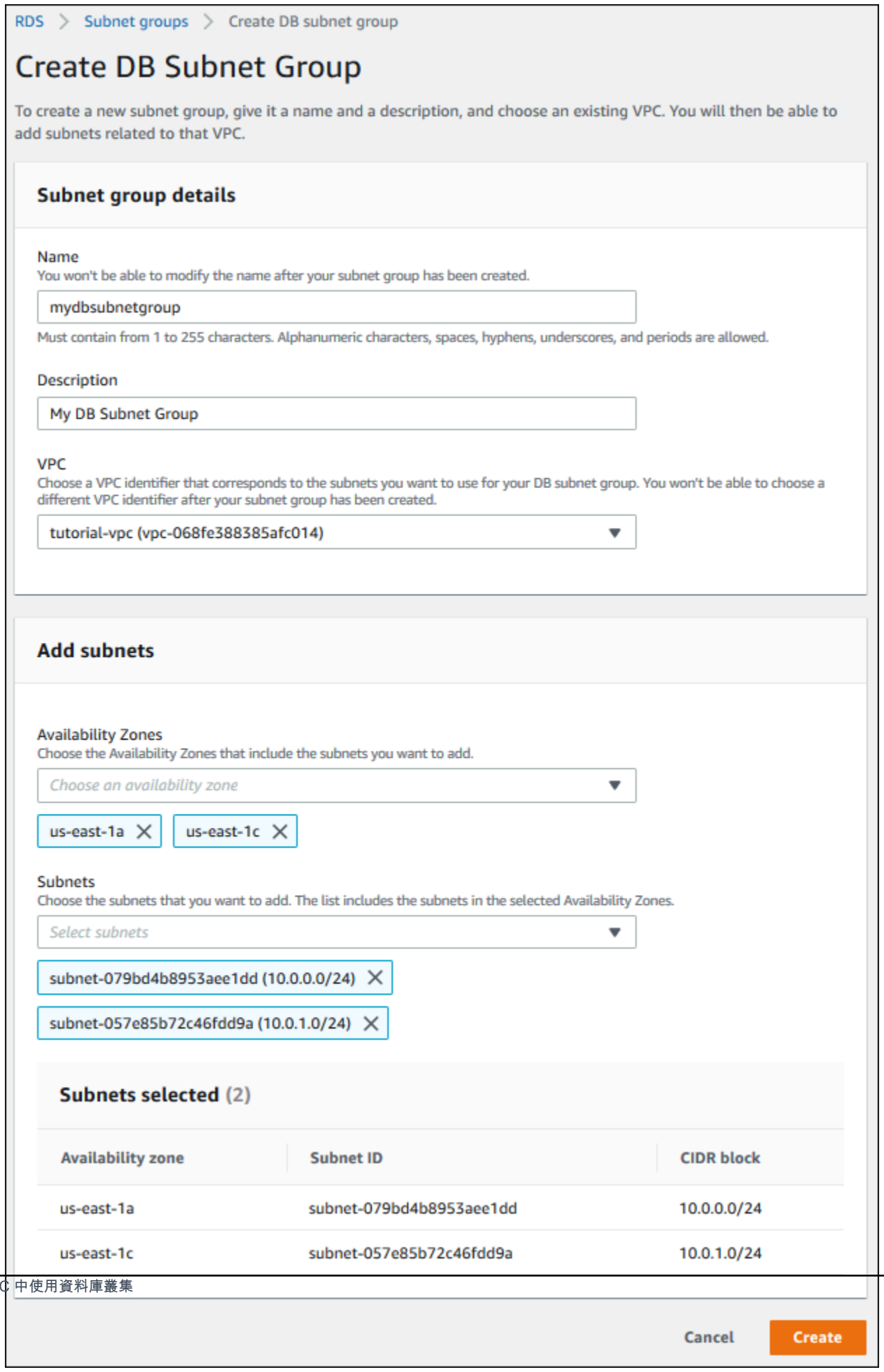
在 VPC 中建立資料庫叢集時，您可以選擇資料庫子網路群組。Amazon Aurora 會選擇子網路和該子網路內的 IP 地址，以便與您的資料庫叢集建立關聯。如果資料庫子網路群組不存在，建立預設資料庫叢

集時，Amazon Aurora 會建立預設子網路群組。Amazon Aurora 會建立彈性網路界面，並透過該 IP 地址將該界面與資料庫叢集相關聯。資料庫叢集使用包含子網路的可用區域。

在此步驟中，您會建立資料庫子網路群組，並新增您為 VPC 建立的子網路。

建立資料庫子網路群組

1. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
2. 在導覽窗格中選擇 Subnet groups (子網路群組)。
3. 選擇 Create DB Subnet Group (建立資料庫子網路群組)。
4. 在 Name (名稱) 欄位輸入您資料庫子網路群組的名稱。
5. 在 Description (描述) 欄位輸入您資料庫子網路群組的描述。
6. 對於 VPC，選擇預設 VPC 或先前建立的 VPC。
7. 在 Add subnets (新增子網路) 區段中，選擇包含 Availability Zones (可用區域) 中子網路的可用區域，然後從 Subnets (子網路) 選擇子網路。


RDS > Subnet groups > Create DB subnet group
Create DB Subnet Group
To create a new subnet group, give it a name and a description, and choose an existing VPC. You will then be able to add subnets related to that VPC.
Subnet group details
Name
You won't be able to modify the name after your subnet group has been created.
mydbsubnetgroup
Must contain from 1 to 255 characters. Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.
Description
My DB Subnet Group
VPC
Choose a VPC identifier that corresponds to the subnets you want to use for your DB subnet group. You won't be able to choose a different VPC identifier after your subnet group has been created.
tutorial-vpc (vpc-068fe388385afc014)
Add subnets
Availability Zones
Choose the Availability Zones that include the subnets you want to add.
Choose an availability zone
us-east-1a
us-east-1c
Subnets
Choose the subnets that you want to add. The list includes the subnets in the selected Availability Zones.
Select subnets
subnet-079bd4b8953aee1dd (10.0.0.0/24)
subnet-057e85b72c46fdd9a (10.0.1.0/24)
Subnets selected (2)
Availability zone
Subnet ID
CIDR block
us-east-1a
subnet-079bd4b8953aee1dd
10.0.0.0/24
us-east-1c
subnet-057e85b72c46fdd9a
10.0.1.0/24
Cancel
Create

8. 選擇建立。

   您的新資料庫子網路群組會顯示在 RDS 主控台的資料庫子網路群組清單中。您可以選擇資料庫子網路群組，在視窗底部的詳細資訊窗格查看詳細資訊，包括與該群組相關聯的所有子網路。

步驟 3：建立 VPC 安全群組

建立資料庫叢集之前，您必須先建立 VPC 安全群組，才能與您的資料庫叢集建立關聯。如果您沒有建立 VPC 安全群組，您可以在建立資料庫叢集時使用預設安全群組。如需如何建立資料庫叢集安全群組的說明，請參閱 建立私有資料庫叢集的 VPC 安全群組，或參閱 Amazon VPC 使用者指南中的使用安全群組控制到資源的流量。

步驟 4：在 VPC 中建立資料庫執行個體

在此步驟中，您將建立資料庫叢集，並使用您在先前步驟中建立的 VPC 名稱、資料庫子網路群組以及 VPC 安全群組。

> **Note**
>
> 如果您要開放資料庫叢集在 VPC 中供公開存取，須啟用 VPC 屬性 DNS hostnames (DNS 主機名稱) 和 DNS resolution (DNS 解析)。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的 VPC 的 DNS 屬性。

如需如何建立資料庫叢集的詳細資訊，請參閱建立 Amazon Aurora 資料庫叢集。

Connectivity (連線) 區段中顯示提示訊息時，請輸入 VPC 名稱、資料庫子網路群組以及 VPC 安全群組。

> **Note**
>
> Aurora DB 叢集目前不支援更新 VPC。

在 VPC 中存取資料庫叢集的案例

Amazon Aurora 支援下列在 VPC 中存取資料庫叢集的使用案例：

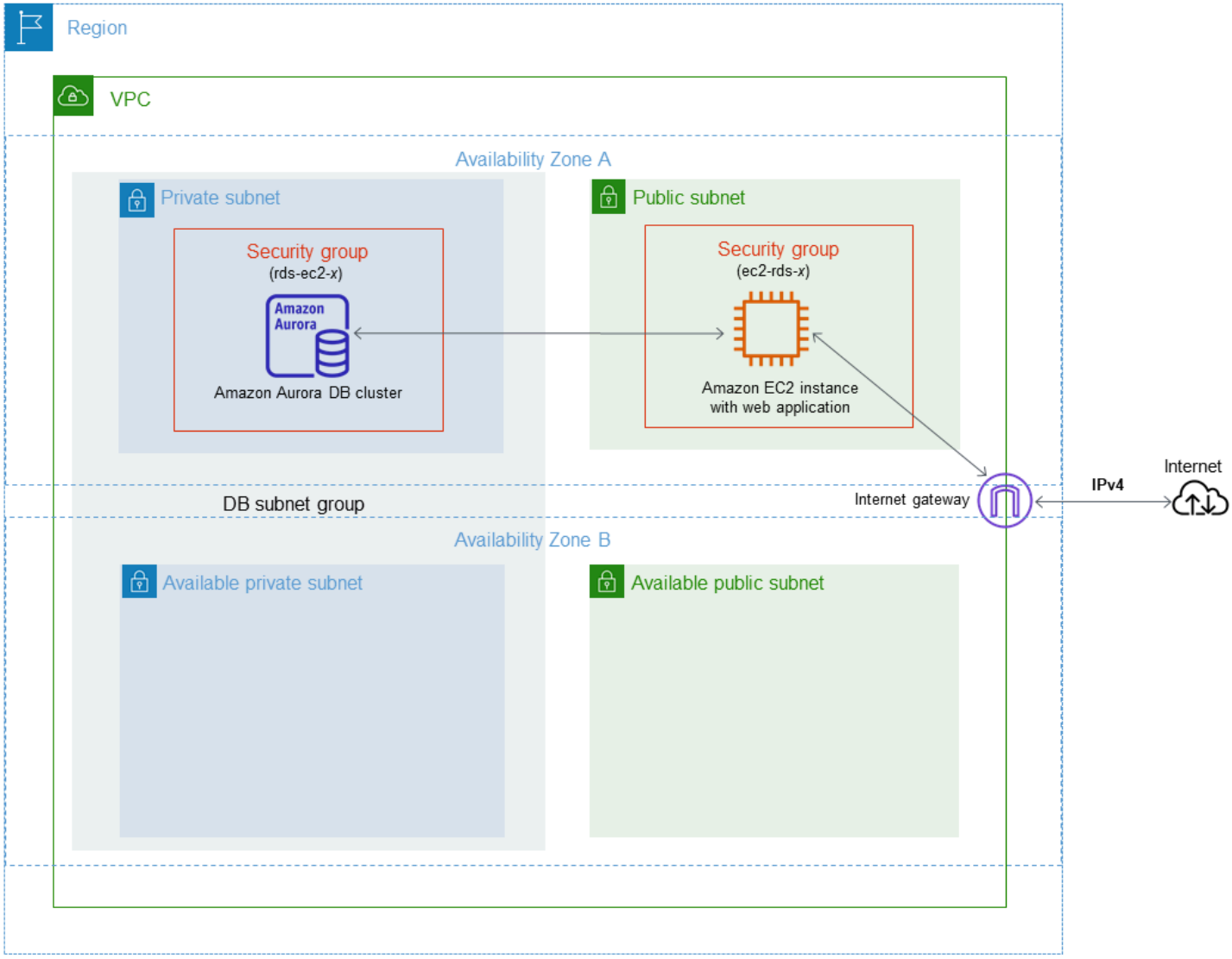
- 相同 VPC 中的 Amazon EC2 執行個體

- EC2 執行個體位於不同 VPC
- 連上網際網路的用戶端應用程式
- 私有網路

由相同 VPC 中的 Amazon EC2 執行個體在 VPC 中存取的資料庫叢集

資料庫叢集在 VPC 中常見的使用方式，是與在同一 VPC 之 Amazon EC2 執行個體中執行的應用程式伺服器共用資料。

此案例可以下列圖表顯示。



若要管理相同 VPC 中 EC2 執行個體與資料庫叢集之間的存取權限，最簡單的方式如下：

- 建立將包含資料庫叢集的 VPC 安全群組。此安全群組可用來限制資料庫叢集的存取權限。例如，您可以為此安全群組建立自訂規則，允許使用您在建立自訂規則時指派給資料庫叢集的連接埠存取 TCP，並可建立一組存取資料庫叢集的 IP 地址，做為開發或其他用途使用。
- 建立將包含 EC2 執行個體 (web 伺服器和用戶端) 的 VPC 安全群組。若有需要，此安全群組可允許藉由使用 VPC 路由表存取網際網路上的 EC2 執行個體。舉例來說，您可以在此安全群組上設定規則，允許 TCP 透過連接埠 22 存取 EC2 執行個體。
- 在您資料庫叢集的安全群組中建立自訂規則，允許來自您為 EC2 執行個體所建立之安全群組的連線要求。這些規則可能會允許安全群組的所有成員存取資料庫叢集。

在單獨的可用區域中，還有一個額外的公有和私有子網路。RDS 資料庫子網路群組需要至少兩個可用區域中的子網路。額外的子網路可讓您在未來輕鬆切換到多可用區域資料庫執行個體部署。

如需相關教學課程，了解如何為此案例建立包含公有和私有子網路的 VPC，請參閱教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4)。

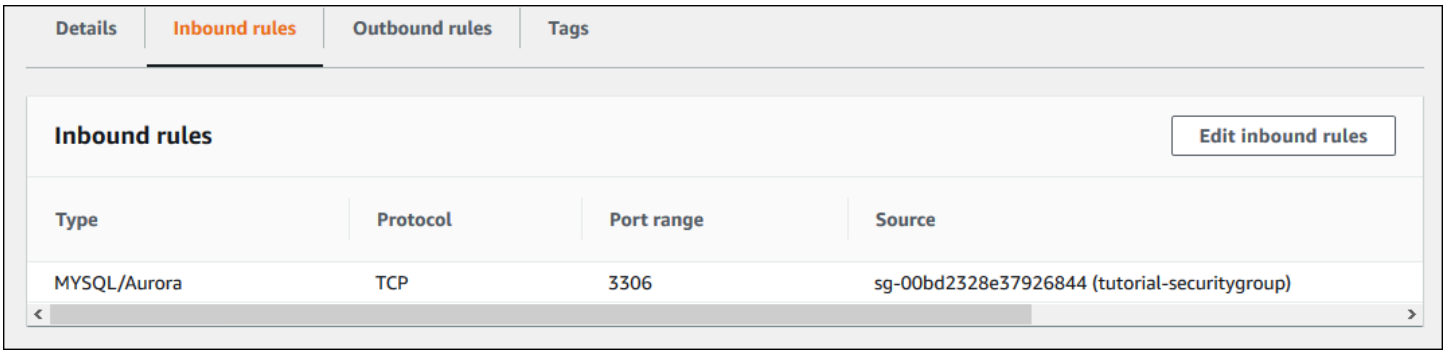
> **Tip**
>
> 您可以在建立資料庫叢集時，自動設定 Amazon EC2 執行個體與資料庫叢集之間的連線。如需更多詳細資訊，請參閱 設定與 EC2 執行個體的自動網路連線。

若要在允許其他安全群組連線要求的 VPC 安全群組中建立規則，請按照以下步驟操作：

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/vpc：// 開啟 Amazon VPC 主控台。
2. 在導覽窗格中，選擇安全群組。
3. 選擇或建立要允許其他安全群組成員存取的安全群組。在前一個藍本中，這是您用於資料庫叢集的安全群組。選擇 Inbound rules (傳入規則) 索引標籤，然後選擇 Edit inbound rules (編輯對內規則)。
4. 在 Edit inbound rules (編輯對內規則) 頁面上，選擇 Add rule (新增規則)。
5. 對於類型，選擇對應您在建立資料庫叢集時所使用之連接埠的項目，例如 MYSQL/Aurora。
6. 在來源方塊中，開始輸入安全群組 ID，這會列出相符的安全群組。選擇安全群組，允許其成員存取由此安全群組所保護的資源。在前一個藍本中，這是您用於 EC2 執行個體的安全群組。
7. 視需要使用所有 TCP 做為類型並在來源方塊中輸入安全群組來建立規則，以重複 TCP 通訊協定的步驟。若您打算使用 UDP 通訊協定，請使用 All UDP (所有 UDP) 作為 Type (類型)，並在 Source (來源) 中輸入安全群組，以建立規則。

8. 選擇儲存規則。

下列畫面顯示安全群組針對其來源的對內規則。

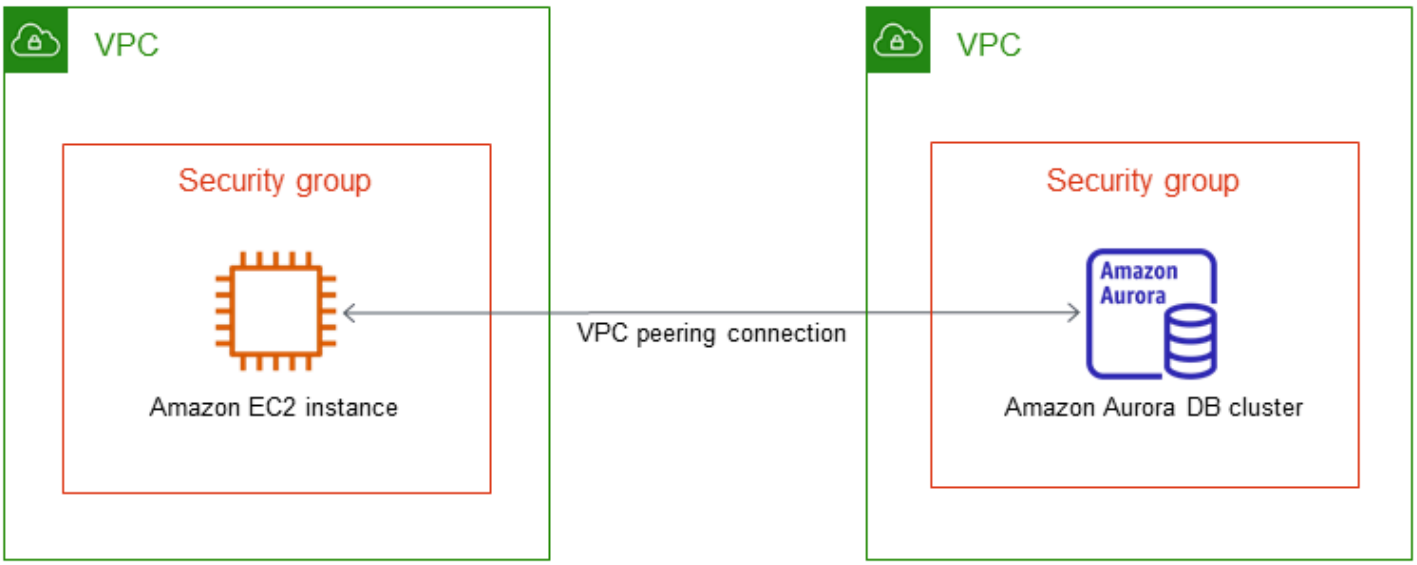


如需從 EC2 執行個體連線至資料庫叢集的詳細資訊，請參閱 連接至 Amazon Aurora 資料庫叢集。

由不同 VPC 中的 EC2 叢集存取 VPC 中的資料庫執行個體

當您的資料庫叢集與您用來存取資料庫執行個體叢集的 EC2 執行個體不在相同 VPC 中時，您就能使用 VPC 對等連線存取該資料庫叢集。

此案例可以下列圖表顯示。

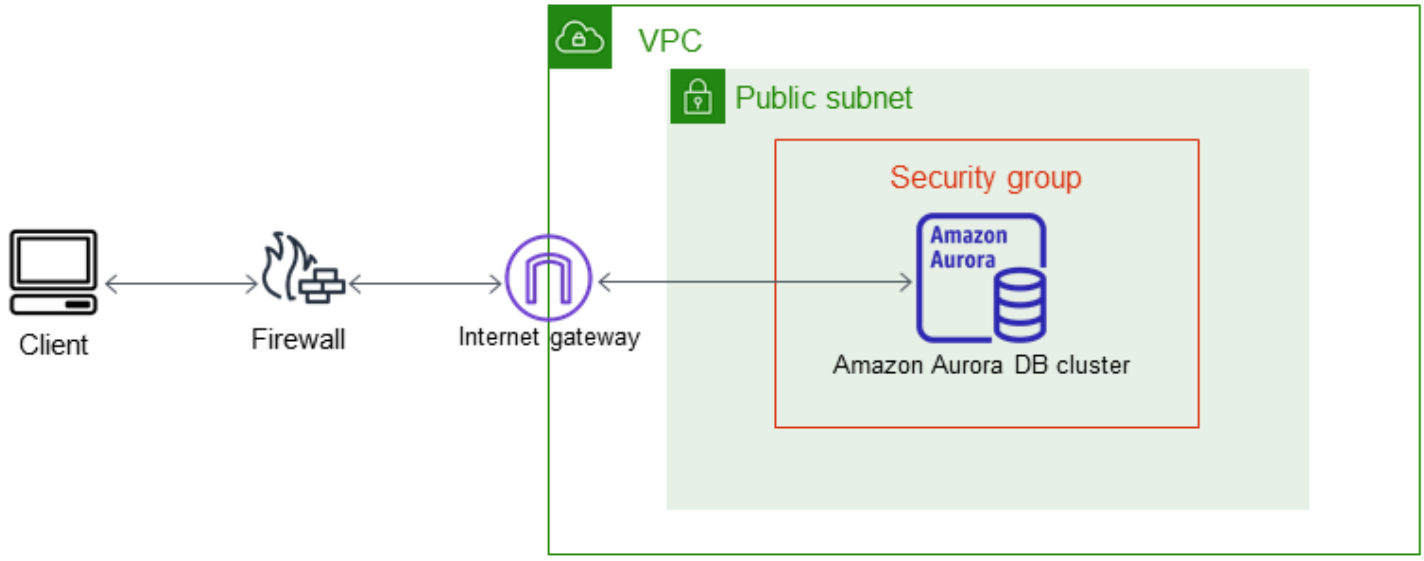


VPC 對等連線是指兩個 VPC 之間的網路連線，透過此機制，您就可以使用私有 IP 地址在兩者之間路由流量。這兩個 VPC 中的資源能彼此通訊，有如位於相同網路中一樣。您可以在自己的 VPCs 之間建立 VPC 對等互連、在另一個 AWS 帳戶中建立 VPC，或在不同的 VPC 之間建立 VPC 對等互連 AWS 區域。若要進一步了解 VPC 互連，請參閱《Amazon Virtual Private Cloud 使用者指南》中的 VPC 互連。

由用戶端應用程式透過網際網路存取 VPC 中的資料庫叢集

若要由用戶端應用程式透過網際網路存取 VPC 中的資料庫叢集，您必須設定單一公有子網路的 VPC 以及網際網路閘道，啟用網際網路通訊。

此案例可以下列圖表顯示。



我們建議您使用下列組態：

- 大小為 /16 的 VPC (例如，CIDR: 10.0.0.0/16)。此大小可提供 65,536 個私有 IP 地址。
- 大小為 /24 的子網路 (例如，CIDR: 10.0.0.0/24)。此大小可提供 256 個私有 IP 地址。
- 與 VPC 和子網路相關聯的 Amazon Aurora 資料庫叢集。Amazon RDS 會將子網路中的 IP 地址指派給資料庫叢集。
- 網際網路閘道會將 VPC 連線至網際網路和其他 AWS 產品。
- 與資料庫叢集相關聯的安全群組。安全群組的傳入規則允許您的用戶端應用程式存取您的資料庫叢集。

如需有關在 VPC 中建立資料庫叢集的資訊，請參閱 在 VPC 中建立資料庫叢集。

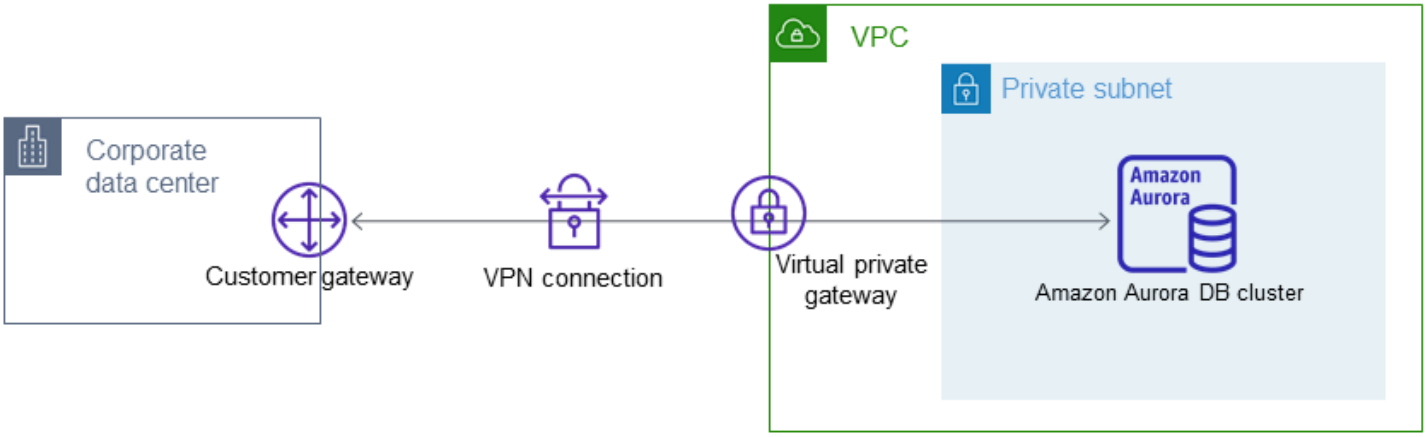
透過私有網路存取 VPC 中的資料庫叢集

如果您的資料庫叢集無法公開存取，可以使用下列選項從私有網路存取：

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

- AWS Client VPN 連線。如需詳細資訊，請參閱什麼是 AWS Client VPN？

下圖顯示具有 AWS Site-to-Site連線的案例。

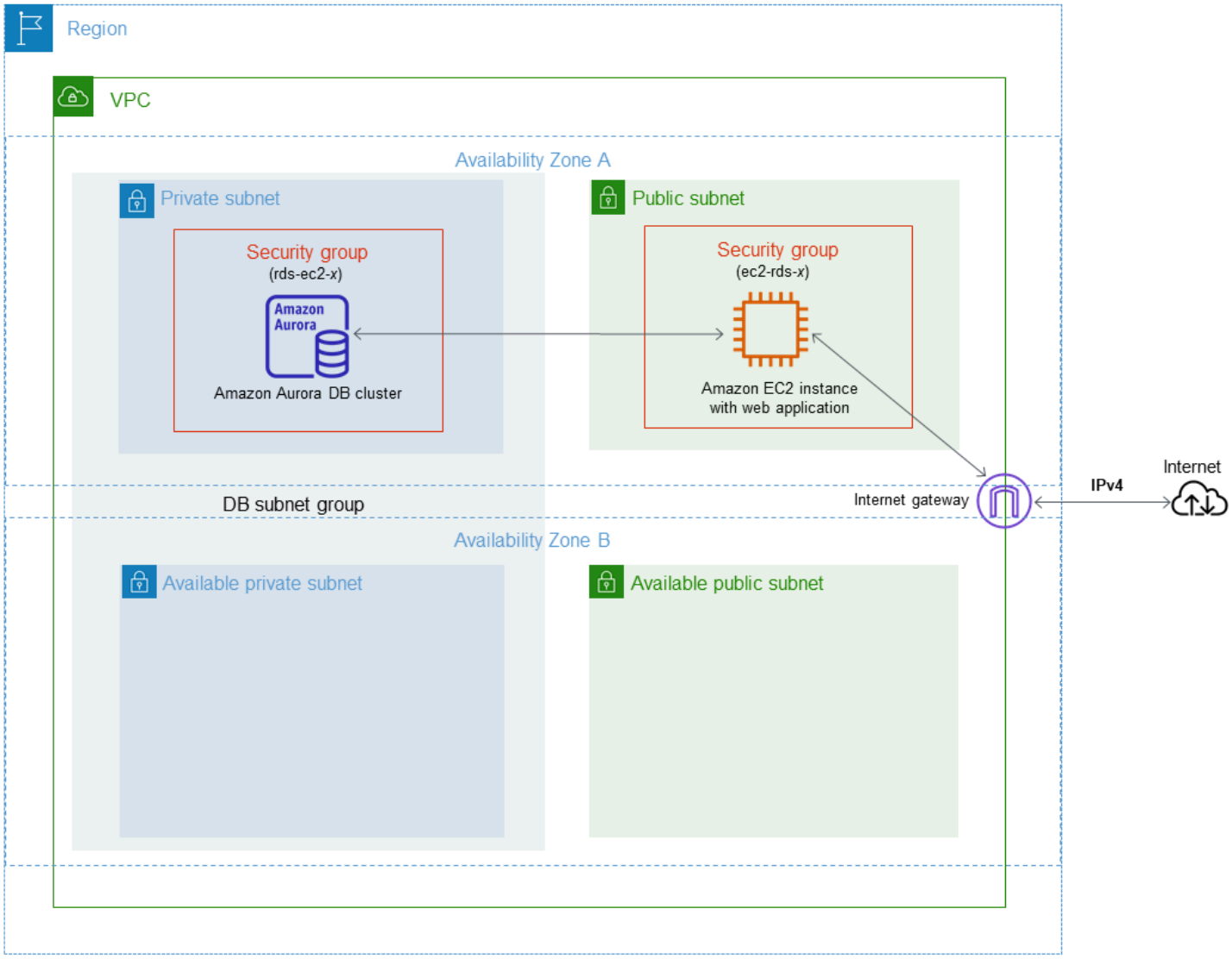


如需詳細資訊，請參閱網際網路流量隱私權。

教學課程：建立要與資料庫叢集搭配使用的 VPC (僅限 IPv4)

常用案例包括以 Amazon VPC 服務為基礎的虛擬私有雲端 (VPC) 中的資料庫叢集。此 VPC 與在相同 VPC 中執行的 Web 伺服器共用資料。在本教學課程中，您會針對此案例建立 VPC。

此案例可以下列圖表顯示。如需其他案例的相關資訊，請參閱在 VPC 中存取資料庫叢集的案例。



您的資料庫叢集僅需供您的 Web 伺服器使用，無需供公有網際網路使用。因此，您建立同時包含公有和私有子網路的 VPC。Web 伺服器是在公有子網路中託管，以便它可以到達公有網際網路。資料庫叢集被託管於私有子網路中。Web 伺服器可連線至資料庫叢集，因為其託管於相同 VPC 中。但是，公有網際網路無法使用資料庫叢集，提供更高的安全性。

本教學課程會在單獨的可用區域中設定額外的公有子網路和私有子網路。教學課程不會使用這些子網路。RDS 資料庫子網路群組需要至少兩個可用區域中的子網路。額外的子網路可讓您更輕鬆地設定多個 Aurora 資料庫執行個體。

本教學課程說明為 Amazon Aurora 資料庫執行個體配置 VPC。如需展示如何為此 VPC 案例建立 Web 伺服器的教學課程，請參閱教學：建立 Web 伺服器和 Amazon Aurora 資料庫叢集。如需 Amazon VPC 的詳細資訊，請參閱 Amazon VPC 入門指南和 Amazon VPC 使用者指南。

> **Tip**
>
> 當您建立資料庫叢集時，您可以在 Amazon EC2 執行個體和資料庫叢集之間自動設定網路連線。網路組態與本教學課程中描述的組態類似。如需詳細資訊，請參閱 設定與 EC2 執行個體的自動網路連線。

建立含私有和公有子網路的 VPC

使用下列程序來建立一個同時含公有和私有子網路的 VPC。

建立 VPC 和子網路

1. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
2. 在 的右上角 AWS 管理主控台，選擇要在其中建立 VPC 的區域。此範例使用 美國西部 (奧勒岡) 區域。
3. 在左上角，選擇 VPC Dashboard (VPC 儀表板)。若要開始建立 VPC，請選擇 Create VPC (建立 VPC)。
4. 在 VPC Settings (VPC 設定) 的 Resources to create (建立資源) 下，選擇 VPC and more (VPC 和更多)。
5. 對於 VPC settings (VPC 設定)，設定這些值：
   - Name tag auto-generation (自動產生名稱標籤) – **`tutorial`**
   - IPv4 CIDR block (IPv4 CIDR 區塊) – **`10.0.0.0/16`**
   - IPv6 CIDR block (IPv6 CIDR 區塊) – No IPv6 CIDR Block (無 IPv6 CIDR 區塊)
   - Tenancy (租用) – Default (預設)
   - Number of Availability Zones (AZs) (可用區域 (AZ) 的數量) – 2
   - Customize AZs (自訂可用區域) - 保留預設值。

- Number of public subnet (公有子網路數量) – 2
- Number of private subnets (私有子網路數量) – 2
- Customize subnets CIDR blocks (自訂子網路 CIDR 區塊) – 保留預設值。
- NAT gateways ($) (NAT 閘道 ($)) – None (無)
- VPC endpoints (VPC 端點) – None (無)
- DNS options (DNS 選項) – 保留預設值。

6. 選擇建立 VPC。

建立公有 Web 伺服器的 VPC 安全群組

接著建立公開存取的 VPC 安全群組。若要連線至 VPC 中的公有 EC2 執行個體，請將傳入規則新增至 VPC 安全群組。這些規則允許流量從網際網路連線。

建立 VPC 安全群組

1. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
2. 選擇 VPC Dashboard (VPC 儀表板)，再選擇 Security Groups (安全群組)，然後選擇 Create security group (建立安全群組)。
3. 在 Create security group (建立安全群組) 頁面上，設定下列值：
   - 安全群組名稱：**`tutorial-securitygroup`**
   - 描述: **`Tutorial Security Group`**
   - VPC: 選擇您先前建立的 VPC，例如：vpc-**`###`** (tutorial-vpc)
4. 將傳入規則新增至安全群組
   a. 決定用於使用安全殼層 (SSH) 連接至 VPC 中 EC2 執行個體的 IP 地址。若要判斷公有 IP 地址，您可以在不同的瀏覽器視窗或索引標籤中使用 https://checkip.amazonaws.com 中的服務。IP 地址的範例為 `203.0.113.25/32`。

      在許多情況下，您可能透過網際網路服務供應商 (ISP) 或是從沒有靜態 IP 地址的防火牆進行連線。若是如此，請找出用戶端電腦所使用的 IP 地址範圍。

      > ⚠ Warning
      >
      > 如果您使用 `0.0.0.0/0` 進行 SSH 存取，則可讓所有 IP 地址使用 SSH 存取您的公有執行個體。通常在測試環境中短暫使用此方法是沒有問題的，但在生產環境則不安

全。在生產環境中，您應只授權特定 IP 地址或特定範圍的地址可使用 SSH 存取您的執行個體。

b. 在 Inbound rules (傳入規則) 區段中，選擇 Add rule (新增規則)。

c. 針對您的新傳入規則設定下列值，以允許透過 SSH 存取您的 Amazon EC2 執行個體。若是這麼做，則您可以連線至 Amazon EC2 執行個體來安裝 Web 伺服器和其他公用程式。您也會連線至 EC2 執行個體來上傳 Web 伺服器的內容。

- 類型: **SSH**
- Source (來源): 來自步驟 a 的 IP 地址或範圍，例如：**203.0.113.25/32**。

d. 選擇新增規則。

e. 針對您的新傳入規則設定下列值，以允許透過 HTTP 存取您的 Web 伺服器。

- Type (類型)：**HTTP**
- 來源：**0.0.0.0/0**

5. 請選擇 Create security group (建立安全群組) 以建立安全群組。

請記下安全群組 ID，因為稍後在本教學中會需要它。

建立私有資料庫叢集的 VPC 安全群組

若要讓您的資料庫叢集保持私有，請建立第二個安全群組供私有存取。若要連接至 VPC 中的私有資料庫叢集，請將傳入規則新增至 VPC 安全群組，僅允許來自 Web 伺服器的流量。

建立 VPC 安全群組

1. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
2. 選擇 VPC Dashboard (VPC 儀表板)，再選擇 Security Groups (安全群組)，然後選擇 Create security group (建立安全群組)。
3. 在 Create security group (建立安全群組) 頁面上，設定下列值：

- 安全群組名稱：**tutorial-db-securitygroup**
- 描述: **Tutorial DB Instance Security Group**
- VPC: 選擇您先前建立的 VPC，例如：vpc-**###** (tutorial-vpc)

4. 將傳入規則新增至安全群組

a. 在 Inbound rules (傳入規則) 區段中，選擇 Add rule (新增規則)。

b. 針對您的新傳入規則設定下列值，來允許連接埠 3306 上來自 Amazon EC2 執行個體的 MySQL 流量。如果這樣做，您可以從 Web 伺服器連線至資料庫叢集。這樣做，您可以將 Web 應用程式中的資料存放並擷取至資料庫。

- Type (類型)：**MySQL/Aurora**
- Source (來源)： 您先前在本教學課程中建立的 tutorial-securitygroup 安全群組的識別符，例如：sg-9edd5cfb。

5. 請選擇 Create security group (建立安全群組) 以建立安全群組。

建立資料庫子網路群組

資料庫子網路群組是您在 VPC 中建立，然後指派給資料庫叢集的子網路的集合。資料庫子網路群組可讓您在建立資料庫叢集時指定特定的 VPC。

建立資料庫子網路群組

1. 識別 VPC 中資料庫的私有子網路。

   a. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。

   b. 選擇 VPC Dashboard (VPC 儀表板)，然後選擇 Subnets (子網路)。

   c. 記下名為 tutorial-subnet-private1-us-west-2a 和 tutorial-subnet-private2-us-west-2b 之子網路的子網路 ID。

      建立資料庫子網路群組時，您會需要這些子網路 ID。

2. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。

   請確定您連接到 Amazon RDS 主控台，而非 Amazon VPC 主控台。

3. 在導覽窗格中選擇 Subnet groups (子網路群組)。
4. 選擇 Create DB Subnet Group (建立資料庫子網路群組)。
5. 在 Create DB subnet group (建立資料庫子網路群組) 頁面上，於 Subnet group details (子網路群組詳細資訊) 中設定下列值：

   - 名稱: **tutorial-db-subnet-group**
   - 描述: **Tutorial DB Subnet Group**
   - VPC: tutorial-vpc (vpc-**###**)

6. 在 Add subnets (新增子網路) 區段中，選擇 Availability Zones (可用區域) 和 Subnets (子網路)。

對於此教學課程，為 Availability Zones (可用區域) 選擇 us-west-2a 和 us-west-2b。對於 Subnets (子網路)，選擇您在上一個步驟找到的私有子網路。

7. 選擇建立。

   您的新資料庫子網路群組會顯示在 RDS 主控台的資料庫子網路群組清單中。您可以選擇資料庫子網路群組，在視窗底部的詳細資訊窗格查看詳細資訊。這些詳細資料包括與群組關聯的所有子網路。

> **Note**
>
> 如果您已建立此 VPC 以完成 教學：建立 Web 伺服器和 Amazon Aurora 資料庫叢集，請依照 建立 Amazon Aurora 資料庫叢集 中的下列指示建立資料庫叢集。

刪除 VPC

為此教學課程建立 VPC 和其他資源後，如果不再需要這些資源，便可以將它們刪除。

> **Note**
>
> 若您在針對此教學課程建立的 VPC 中新增資源，則可能需要先刪除這些資源，才能刪除 VPC。例如，這些資源可能包括 Amazon EC2 執行個體或 Amazon RDS 資料庫叢集。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的刪除 VPC。

刪除 VPC 和相關資源

1. 刪除資料庫子網路群組。

   a. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
   b. 在導覽窗格中選擇 Subnet groups (子網路群組)。
   c. 選取您要刪除的資料庫子網路群組，例如 tutorial-db-subnet-group。
   d. 選擇 Delete (刪除)，然後在確認視窗中選擇 Delete (刪除)。

2. 請注意 VPC ID。

   a. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
   b. 選擇 VPC Dashboard (VPC 儀表板)，然後選擇 VPC。

c. 在清單中，找到您建立的 VPC，例如 tutorial-vpc。

d. 請記下您所建立的 VPC 的 VPC ID。在稍後的步驟中，您需要 VPC ID。

3. 刪除安全群組。

a. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。

b. 選擇 VPC Dashboard (VPC 儀表板)，然後選擇 Security Groups (安全群組)。

c. 選取 Amazon RDS 資料庫執行個體的安全群組，例如 tutorial-db-securitygroup。

d. 若為 Actions (動作)，選擇 Delete security groups (刪除安全群組)，然後在確認頁面上選擇 Delete (刪除)。

e. 在 Security Groups (安全群組) 頁面上，選取 Amazon EC2 執行個體的安全群組，例如 tutorial-securitygroup。

f. 若為 Actions (動作)，選擇 Delete security groups (刪除安全群組)，然後在確認頁面上選擇 Delete (刪除)。

4. 刪除 VPC。

a. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。

b. 選擇 VPC Dashboard (VPC 儀表板)，然後選擇 VPC。

c. 選取您要刪除的 VPC，例如 tutorial-vpc。

d. 對於 Actions (動作)，請選擇 Delete VPC (刪除 VPC)。

確認頁面會顯示與 VPC 相關聯的其他資源，這些資源也將遭到刪除，包括與其相關聯的子網路。

e. 在確認頁面上，輸入 **`delete`**，然後選擇 Delete (刪除)。

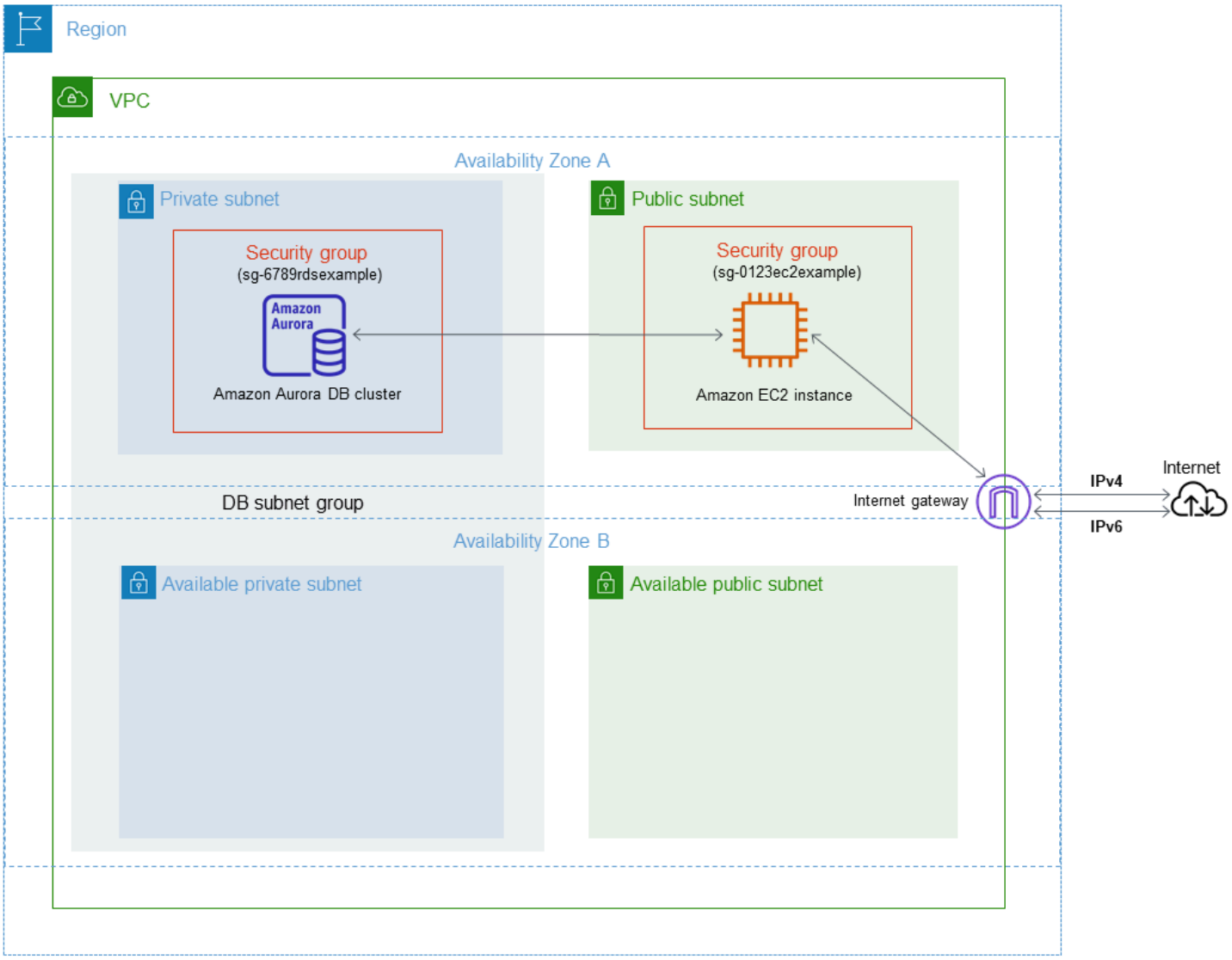
教學課程：建立要與資料庫叢集搭配使用的 (VPC)(雙堆疊模式)

常用案例包括以 Amazon VPC 服務為基礎的虛擬私有雲端 (VPC) 中的資料庫叢集。此 VPC 與在相同 VPC 中執行的公有 Amazon EC2 執行個體共用資料。

於本教學課程中，您將為此案例建立與以雙堆疊模式執行之資料庫搭配使用的 VPC。雙堆疊模式，可透過 IPv6 定址通訊協定啟用連線。若需 IP 定址的詳細資訊，請參閱 Amazon Aurora IP 定址。

大部分地區都支援雙堆疊網路叢集。如需更多資訊，請參閱雙堆疊網路資料庫叢集的可用性。若要查看雙堆疊模式的限制，請參閱 雙堆疊網路資料庫叢集的限制。

此案例可以下列圖表顯示。



如需其他案例的相關資訊，請參閱在 VPC 中存取資料庫叢集的案例。

您的資料庫叢集僅需供您的 Amazon EC2 執行個體使用，無需供公有網際網路使用。因此，您建立同時包含公有和私有子網路的 VPC。Amazon EC2 執行個體是在公有子網路中託管，如此其可連接至公有網際網路。資料庫叢集被託管於私有子網路中。Amazon EC2 執行個體可連線至資料庫叢集，因為其託管於相同 VPC 中。但是，資料庫叢集不可用於公有網際網路，以提供更高的安全性。

本教學課程會在單獨的可用區域中設定額外的公有子網路和私有子網路。教學課程不會使用這些子網路。RDS 資料庫子網路群組需要至少兩個可用區域中的子網路。額外的子網路可讓您輕鬆設定多個 Aurora 資料庫執行個體。

若要建立使用雙堆疊模式的資料庫叢集，請針對 Network type (網路類型) 設定指定 Dual-stack mode (雙堆疊模式)。您也可以使用相同設定來修改資料庫叢集。如需建立資料庫叢集的詳細資訊，請參閱建立 Amazon Aurora 資料庫叢集。如需修改資料庫叢集的詳細資訊，請參閱修改 Amazon Aurora 資料庫叢集。

本教學課程說明為 Amazon Aurora 資料庫執行個體配置 VPC。如需 Amazon VPC 的詳細資訊，請參閱《Amazon VPC 使用者指南》https://docs.aws.amazon.com/vpc/latest/userguide/。

建立含私有和公有子網路的 VPC

使用下列程序來建立一個同時含公有和私有子網路的 VPC。

建立 VPC 和子網路

1. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
2. 在 的右上角 AWS 管理主控台，選擇要在其中建立 VPC 的區域。此範例使用 美國東部 (俄亥俄) 區域。
3. 在左上角，選擇 VPC Dashboard (VPC 儀表板)。若要開始建立 VPC，請選擇 Create VPC (建立 VPC)。
4. 在 VPC Settings (VPC 設定) 的 Resources to create (建立資源) 下，選擇 VPC and more (VPC 和更多)。
5. 對於剩下的 VPC 設定，設定這些值：

   - Name tag auto-generation (自動產生名稱標籤) – **`tutorial-dual-stack`**
   - IPv4 CIDR block (IPv4 CIDR 區塊) – **`10.0.0.0/16`**
   - IPv6 CIDR block (IPv6 CIDR 區塊) – Amazon-provided IPv6 CIDR block (Amazon 提供的 IPv6 CIDR 區塊)
   - Tenancy (租用) – Default (預設)
   - Number of Availability Zones (AZs) (可用區域 (AZ) 的數量) – 2

- Customize AZs (自訂可用區域) - 保留預設值。
- Number of public subnet (公有子網路數量) – 2
- Number of private subnets (私有子網路數量) – 2
- Customize subnets CIDR blocks (自訂子網路 CIDR 區塊) – 保留預設值。
- NAT gateways ($) (NAT 閘道 ($)) – None (無)
- Egress only internet gateway (僅輸出網際網路閘道) – No (否)
- VPC endpoints (VPC 端點) – None (無)
- DNS options (DNS 選項) – 保留預設值。

> **Note**
>
> Amazon RDS 需要至少兩個不同可用區域中的子網路，以支援多可用區域資料庫執行個體部署。本教學課程會建立單一可用區部署，但是，可讓您會來輕鬆轉換為多區域可用資料庫執行個體部署。

6. 選擇建立 VPC。

建立公有 Amazon EC2 執行個體的 VPC 安全群組

接著建立公開存取的 VPC 安全群組。若要連接至 VPC 中的公有 EC2 執行個體，請將傳入規則新增至 VPC 安全群組，以允許流量從網際網路連接。

建立 VPC 安全群組

1. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
2. 選擇 VPC Dashboard (VPC 儀表板)，再選擇 Security Groups (安全群組)，然後選擇 Create security group (建立安全群組)。
3. 在 Create security group (建立安全群組) 頁面上，設定下列值：
   - 安全群組名稱：**`tutorial-dual-stack-securitygroup`**
   - 描述: **`Tutorial Dual-Stack Security Group`**
   - VPC: 選擇您先前建立的 VPC，例如 vpc-**###** (tutorial-dual-stack-vpc)
4. 將傳入規則新增至安全群組
   a. 決定用於使用安全殼層 (SSH) 連接至 VPC 中 EC2 執行個體的 IP 地址。

網際網路通訊協定第 4 版 (IPv4) 位址的範例為 `203.0.113.25/32`。網際網路通訊協定第 6 版 (IPv6) 地址範圍的範例為 `2001:db8:1234:1a00::/64`。

在許多情況下，您可能透過網際網路服務供應商 (ISP) 或是從沒有靜態 IP 地址的防火牆進行連線。若是如此，請找出用戶端電腦所使用的 IP 地址範圍。

> **Warning**
>
> 若您將 `0.0.0.0/0` 用於 IPv4 或 `::0` 用於 IPv6，則可讓所有 IP 地址使用 SSH 存取您的公有執行個體。通常在測試環境中短暫使用此方法是沒有問題的，但在生產環境則不安全。在生產環境中，建議您只授權特定 IP 地址或特定範圍的地址存取您的執行個體。

b. 在 Inbound rules (傳入規則) 區段中，選擇 Add rule (新增規則)。

c. 針對您的新傳入規則設定下列值，來允許透過 Secure Shell (SSH) 存取您的 Amazon EC2 執行個體。若執行此動作，您可連線至 EC2 執行個體來安裝 SQL 用戶端和其他應用程式。指定 IP 地址，讓您可以存取 EC2 執行個體：

- Type (類型)：**`SSH`**
- Source (來源)：來自步驟 a 的 IP 地址或範圍。IPv4 IP 地址的範例為 **`203.0.113.25/32`**。IPv6 IP 地址的範例為 **`2001:DB8::/32`**。

5. 請選擇 Create security group (建立安全群組) 以建立安全群組。

   請記下安全群組 ID，因為稍後在本教學中會需要它。

建立私有資料庫叢集的 VPC 安全群組

若要讓您的資料庫叢集保持私有，請建立第二個安全群組供私有存取。若要連線至 VPC 中的私有資料庫叢集，請將傳入規則新增至 VPC 安全群組。這些規則僅允許來自 Amazon EC2 執行個體的流量。

建立 VPC 安全群組

1. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
2. 選擇 VPC Dashboard (VPC 儀表板)，再選擇 Security Groups (安全群組)，然後選擇 Create security group (建立安全群組)。
3. 在 Create security group (建立安全群組) 頁面上，設定下列值：

- 安全群組名稱：**tutorial-dual-stack-db-securitygroup**
- 描述: **Tutorial Dual-Stack DB Instance Security Group**
- VPC: 選擇您先前建立的 VPC，例如 vpc-**###** (tutorial-dual-stack-vpc)

4. 將傳入規則新增至安全群組：

   a. 在 Inbound rules (傳入規則) 區段中，選擇 Add rule (新增規則)。

   b. 針對您的新傳入規則設定下列值，來允許連接埠 3306 上來自 Amazon EC2 執行個體的 MySQL 流量。如果這樣做，您可以從 EC2 執行個體連線至資料庫叢集。這樣做表示您可以從 EC2 執行個體將資料傳送至您的資料庫。

      - Type (類型)：MySQL/Aurora
      - Source (來源)：您先前在本教學課程中建立的 tutorial-dual-stack-securitygroup 安全群組的識別符，例如 sg-9edd5cfb。

5. 若要建立安全群組，請選擇 Create security group (建立安全群組)。

建立資料庫子網路群組

資料庫子網路群組是您在 VPC 中建立，然後指派給資料庫叢集的子網路的集合。透過使用資料庫子網路群組，您可在建立資料庫叢集時指定特定的 VPC。如要建立與 DUAL 相容的資料庫子網路群組，則所有子網路必須與 DUAL 相容。如要成為 DUAL 相容，子網路必須具有與其關聯的 IPv6 CIDR。

建立資料庫子網路群組

1. 識別 VPC 中資料庫的私有子網路。

   a. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。

   b. 選擇 VPC Dashboard (VPC 儀表板)，然後選擇 Subnets (子網路)。

   c. 記下名為 tutorial-dual-stack-subnet-private1-us-west-2a 和 tutorial-dual-stack-subnet-private2-us-west-2b 之子網路的子網路 ID。

      當您建立資料庫子網路群組時，您將需要子網路 ID。

2. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。

   請確定您連接到 Amazon RDS 主控台，而非 Amazon VPC 主控台。

3. 在導覽窗格中選擇 Subnet groups (子網路群組)。

4. 選擇 Create DB Subnet Group (建立資料庫子網路群組)。

5. 在 Create DB subnet group (建立資料庫子網路群組) 頁面上，於 Subnet group details (子網路群組詳細資訊) 中設定下列值：

   - 名稱: **`tutorial-dual-stack-db-subnet-group`**
   - 描述: **`Tutorial Dual-Stack DB Subnet Group`**
   - VPC (VPC)： tutorial-dual-stack-vpc (vpc-***###***)

6. 在 Add subnets (新增子網路) 區段中，選擇 Availability Zones (可用區域) 和 Subnets (子網路) 選項的值。

   對於此教學課程，為 Availability Zones (可用區域) 選擇 us-east-2a 和 us-east-2b。對於 Subnets (子網路)，選擇您在上一個步驟找到的私有子網路。

7. 選擇建立。

您的新資料庫子網路群組會顯示在 RDS 主控台的資料庫子網路群組清單中。您可以選擇資料庫子網路群組來查看其詳細資訊。其中包括受支援的定址通訊協定、與該組相關聯的所有子網路，以及資料庫子網路群組支援的網路類型。

在雙堆疊模式下建立 Amazon EC2 執行個體

若要建立 Amazon EC2 執行個體，請遵循《Amazon EC2 使用者指南》中的使用新的啟動執行個體精靈啟動執行個體。

在 Configure Instance Details (設定執行個體詳細資訊) 頁面上設定這些值，而其他值都維持預設值：

- 網路：選擇具備公有和私有子網路的現有 VPC，例如 建立含私有和公有子網路的 VPC 中建立的 tutorial-dual-stack-vpc (vpc-*`identifier`*)。
- Subnet (子網路) – 選擇現有的公有子網路，例如 建立公有 Amazon EC2 執行個體的 VPC 安全群組 中建立的 subnet-***###*** | tutorial-dual-stack-subnet-public1-us-east-2a | us-east-2a。
- Auto-assign Public IP (自動指派公有 IP) – 選擇 Enable (啟用)。
- Auto-assign IPv6 IP (自動指派 IPv6 IP) – 選擇 Enable (啟用)。
- Firewall (security groups) 防火牆 (安全群組) – 選擇 Select an existing security group (選取現有的安全群組)。
- Common security groups (一般安全群組) - 選擇現有的安全群組，例如在 建立公有 Amazon EC2 執行個體的 VPC 安全群組 中建立的 `tutorial-securitygroup`。請確定您選擇的安全性群組包含安全殼層 (SSH) 和 HTTP 存取的輸入規則。

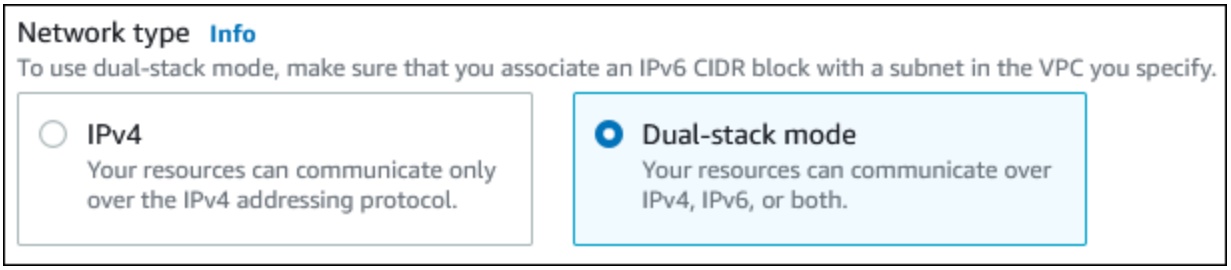
在雙堆疊模式下建立資料庫叢集

在此步驟中，您將建立在雙堆疊模式下執行的資料庫叢集。

建立資料庫執行個體

1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
2. 在主控台的右上角，選擇 AWS 區域 您要建立資料庫叢集的 。此範例使用 美國東部 (俄亥俄) 區域。
3. 在導覽窗格中，選擇 Databases (資料庫)。
4. 選擇 Create database (建立資料庫)。
5. 在 Create database (建立資料庫) 頁面上，確保已選擇 Standard create (標準建立) 選項，接著選擇 Aurora MySQL 資料庫引擎類型。
6. 在 Connectivity (連線) 區段中，設定下列值：

   - Network type (網路類型) – 選擇 Dual-stack mode (雙堆疊模式)。

   - Virtual private cloud (VPC) 虛擬私有雲端 (VPC) – 選擇具備公有和私有子網路的現有 VPC，例如 建立含私有和公有子網路的 VPC 中建立的 tutorial-dual-stack-vpc (vpc-###)。

     VPC 必須具有位於不同可用區域的子網路。

   - DB Subnet group (資料庫子網路群組) – 選擇 VPC 的資料庫子網路群組，例如 建立資料庫子網路群組 中建立的 tutorial-dual-stack-db-subnet-group。
   - Public access (公有存取) – 選擇 No (否)。
   - VPC security group (firewall) (VPC 安全群組 (防火牆)) – 選取 Choose existing (選擇現有)。
   - Existing VPC security groups (現有 VPC 安全群組) – 選擇設定為私有存取的現有 VPC 安全群組，例如在 建立私有資料庫叢集的 VPC 安全群組 中建立的 tutorial-dual-stack-db-securitygroup。

     選擇與各項相關聯的 X，以移除其他安全群組，例如預設安全群組。

- Availability Zone (可用區域) – 選擇 us-west-2a。

  若要避免跨可用區域流量，請確定資料庫執行個體和 EC2 執行個體位於相同的可用區域。

7. 在其餘區段，指定資料庫叢集設定。如需每項設定的相關資訊，請參閱 Aurora 資料庫叢集的設定。

連接至您的 Amazon EC2 執行個體和資料庫叢集

在雙堆疊模式下建立 Amazon EC2 執行個體和資料庫叢集後，您可以使用 IPv6 通訊協定連線到每個執行個體。如要使用 IPv6 通訊協定連線至 Amazon EC2 執行個體，請按照《Amazon EC2 使用者指南》中連接至您的 Linux 執行個體的指示進行。

若要從 Amazon EC2 執行個體連線至您的 Aurora MySQL 資料庫叢集，請依照連線至 Aurora MySQL 資料庫叢集中的指示進行。

刪除 VPC

為此教學課程建立 VPC 和其他資源後，如果不再需要這些資源，便可以將它們刪除。

若您在針對此教學課程建立的 VPC 中新增資源，則可能需要先刪除這些資源，才能刪除 VPC。資源範例為 Amazon EC2 執行個體或資料庫叢集。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的刪除 VPC。

刪除 VPC 和相關資源

1. 刪除資料庫子網路群組：

   a. 前往 https://console.aws.amazon.com/rds/，開啟 Amazon RDS 主控台。
   b. 在導覽窗格中選擇 Subnet groups (子網路群組)。
   c. 選取要刪除的資料庫子網路群組，例如 tutorial-db-subnet-group。
   d. 選擇 Delete (刪除)，然後在確認視窗中選擇 Delete (刪除)。

2. 請記下 VPC ID：

   a. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
   b. 選擇 VPC Dashboard (VPC 儀表板)，然後選擇 VPC。
   c. 在清單中，識別您建立的 VPC，例如 tutorial-dual-stack-vpc。
   d. 請記下您所建立 VPC 的 VPC ID。於後續步驟中，您會需要此 VPC ID。

3. 刪除安全群組：

a. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
b. 選擇 VPC Dashboard (VPC 儀表板)，然後選擇 Security Groups (安全群組)。
c. 選取 Amazon RDS 資料庫執行個體的安全群組，例如 tutorial-dual-stack-db-securitygroup。
d. 若為 Actions (動作)，選擇 Delete security groups (刪除安全群組)，然後在確認頁面上選擇 Delete (刪除)。
e. 在 Security Groups (安全群組) 頁面上，選取 Amazon EC2 執行個體的安全群組，例如 tutorial-dual-stack-securitygroup。
f. 若為 Actions (動作)，選擇 Delete security groups (刪除安全群組)，然後在確認頁面上選擇 Delete (刪除)。

4. 刪除 NAT 閘道：

a. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
b. 選擇 VPC Dashboard (VPC 儀表板)，然後選擇 NAT Gateways (NAT 閘道)。
c. 選取您建立之 VPC 的 NAT 閘道。使用 VPC ID 識別正確的 NAT 閘道。
d. 若為 Actions (動作)，選擇 Delete NAT gateway (刪除 NAT 閘道)。
e. 在確認頁面上，輸入 **`delete`**，然後選擇 Delete (刪除)。

5. 刪除 VPC

a. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
b. 選擇 VPC Dashboard (VPC 儀表板)，然後選擇 VPC。
c. 選取您要刪除的 VPC，例如 tutorial-dual-stack-vpc。
d. 對於 Actions (動作)，請選擇 Delete VPC (刪除 VPC)。

   確認頁面會顯示與 VPC 相關聯的其他資源，這些資源也將遭到刪除，包括與其相關聯的子網路。

e. 在確認頁面上，輸入 **`delete`**，然後選擇 Delete (刪除)。

6. 釋放彈性 IP 地址：

a. 前往 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
b. 選擇 EC2 Dashboard (EC2 儀表板)，然後選擇 Elastic IPs (彈性 IP)。
c. 選取您要釋放的彈性 IP 地址。
d. 若為 Actions (動作)，選擇 Release Elastic IP addresses (釋放彈性 IP 地址)。
e. 在確認頁面上，選擇 Release (釋放)。

Amazon Aurora 的配額和條件限制

您可以在下面找到 Amazon Aurora 的資源配額和命名限制的說明。

主題

- Amazon Aurora 中的配額
- Amazon Aurora 中的命名限制
- Amazon Aurora 大小限制

Amazon Aurora 中的配額

每個 AWS 帳戶對每個 AWS 區域都有可建立的 Amazon Aurora 資源數量配額。在達到資源的配額後，建立該資源的額外呼叫便會失敗並發生例外狀況。

下表列出每個 AWS 區域的資源及其配額。

名稱	預設	可調整	Description
每個資料庫安全群組的授權	每個受支援的區域：20	否	每個資料庫安全群組的安全群組授權數
每個資料庫叢集的自訂端點	每個受支援的區域：5	是	目前區域中，您可以在此帳戶中為每個 Aurora 資料庫叢集建立的自訂端點數量上限。此值反映帳戶中資料庫叢集中自訂端點的數量上限。帳戶中的其他資料庫叢集可能會有較少數量的自訂端點。
自訂引擎版本	每個受支援的區域：40	是	在目前區域中，此帳戶所允許的自訂引擎版本數量上限

名稱	預設	可調整	Description
資料庫叢集參數群組	每個受支援的區域：50	是	資料庫叢集參數群組的數量上限
資料庫叢集	每個受支援的區域：40	是	在目前區域中，此帳戶允許地的 Aurora 叢集數量上限
資料庫執行個體	ap-south-1：20 每個其他支援的區域：40	是	在目前區域中，此帳戶允許地的資料庫執行個體數量上限
資料庫碎片群組	每個受支援的區域：5	是	目前區域中此帳戶中 Aurora Limitless Database 的資料庫碎片群組數目上限
資料庫子網路群組	ap-south-1：20 每個其他支援的區域：50	是	資料庫子網路群組的數量上限
資料 API HTTP 要求主體大小	每個受支援的區域：4 MB	否	HTTP 請求內文允許的大小上限。
資料 API 並行叢集密碼配對上限	每個受支援的區域：30	否	目前 AWS 區域中此帳戶並行資料 API 請求中 Aurora Serverless v1 資料庫叢集和秘密的唯一對數上限。

名稱	預設	可調整	Description
資料 API 並行請求上限	每個受支援的區域：500	否	對 Aurora Serverless 第 1 版資料庫叢集提出的資料 API 請求數量上限，這些請求使用相同的秘密，且系統會同時處理這些請求。其他請求會排入佇列，並在同處理序請求完成時進行處理。
資料 API 結果集大小上限	每個受支援的區域：1 MB	否	資料 API 可傳回的資料庫結果集大小上限。
JSON 回應字串的資料 API 大小上限	每個受支援的區域：10 MB	否	RDS 資料 API 傳回的簡化 JSON 回應字串大小上限。
每秒的資料 API 請求數	每個受支援的區域：每秒 1,000	否	目前 AWS 區域中此帳戶每秒允許對資料 API 的請求數目上限。此配額僅適用於 Amazon Aurora Serverless v1 叢集。
事件訂閱	每個受支援的區域：20	是	事件訂閱的數量上限
每個資料庫叢集的 IAM 角色	每個受支援的區域：5	是	與資料庫叢集相關聯之 IAM 角色的數量上限
每個資料庫執行個體的 IAM 角色	每個受支援的區域：5	是	與資料庫執行個體相關聯的 IAM 角色數量上限
整合	每個受支援的區域：100	否	目前 AWS 區域中此帳戶中允許的整合數目上限

名稱	預設	可調整	Description
手動資料庫叢集快照	每個受支援的區域：100	是	手動資料庫叢集快照的數量上限
資料庫執行個體手動快照	每個受支援的區域：100	是	手動資料庫執行個體快照的數量上限
選項群組	每個受支援的區域：20	是	選項群組的數量上限
參數群組	ap-south-1：20 每個其他支援的區域：50	是	參數群組的數量上限
代理	每個受支援的區域：20	是	目前 AWS 區域中此帳戶中允許的代理數目上限
每個主要項目的僅供讀取複本	每個受支援的區域：15	是	每個主資料庫執行個體的僅供讀取複本數量上限 此配額無法針對 Amazon Aurora 調整。
預留資料庫執行個體	ap-south-1：20 每個其他支援的區域：40	是	目前 AWS 區域中此帳戶中允許的預留資料庫執行個體數目上限
Security groups (安全群組)	ap-south-1：20 每個其他支援的區域：25	是	資料庫安全群組數量上限
每個資料庫子網路群組的子網路	每個受支援的區域：20	否	每個資料庫子網路群組的子網路數目上限

名稱	預設	可調整	Description
所有資料庫執行個體的儲存總量	每個受支援的區域：100,000	是	EBS 磁碟區上所有 Amazon RDS 資料庫執行個體加在一起的儲存體總量上限 (以 GB 為單位)。此配額不適用於 Amazon Aurora，因為其中每個資料庫叢集的叢集磁碟區上限為 128 TiB。

Note

根據預設，您最多可以擁有 40 個資料庫執行個體。RDS 資料庫執行個體、Aurora 資料庫執行個體、Amazon Neptune 執行個體和 Amazon DocumentDB 執行個體皆適用此配額。
如果您的應用程式需要更多資料庫執行個體，您可以透過開啟 Service Quotas 主控台來請求額外的資料庫執行個體。在導覽窗格中，選擇 AWS services (AWS 服務)。選擇 Amazon Relational Database Service (Amazon RDS)，再選擇配額，然後按照指示請求增加配額。如需詳細資訊，請參閱《Service Quotas 使用者指南》中的請求提高配額。
對於 RDS for Oracle，您可以在每個區域中為每個來源資料庫執行個體建立最多 15 個僅供讀取複本，但我們建議將複本限制為 5 個，以盡可能縮短複寫延遲。
管理的備份 AWS Backup 會被視為手動資料庫叢集快照，但不計入手動叢集快照配額。如需的詳細資訊 AWS Backup，請參閱 AWS Backup 開發人員指南。

如果您使用任何 RDS API 操作，且超過每秒呼叫次數的預設配額，Amazon RDS API 會發出類似下面的錯誤。

ClientError：呼叫 *API_name* 操作時發生錯誤 (ThrottlingException)：超過速率。

在這裡，減少每秒呼叫次數。配額旨在涵蓋大多數使用案例。如果需要較高的配額，您可以使用下列其中一個選項來請求提高配額：

- 從主控台，開啟 Service Quotas 主控台。
- 從 中 AWS CLI，使用 request-service-quota-increase AWS CLI 命令。

如需詳細資訊，請參閱「Service Quotas 使用者指南」。

Amazon Aurora 中的命名限制

以下說明 Amazon Aurora 中的命名限制條件：

- 資料庫叢集識別符：
  - 必須包含 1–63 個英數字元或連字號。
  - 第一個字元必須是字母。
  - 不能以一個連字號結尾或是連續包含兩個連字號。
  - 每個區域每個 AWS 帳戶 AWS 的所有資料庫執行個體必須是唯一的。
- 初始資料庫名稱：資料庫名稱限制條件在 Aurora MySQL 和 Aurora PostgreSQL 間會有所不同。如需更多詳細資訊，請參閱建立每個資料庫叢集時的可用設定。
- 主要使用者名稱：每個資料庫引擎的主要使用者名稱限制條件都有所不同。如需更多詳細資訊，請參閱建立資料庫叢集時的可用設定。
- 主要密碼：
  - 資料庫主要使用者密碼可包含任何可印刷的 ASCII 字元，但 /、'、"、@ 或空格除外。
  - 取決於資料庫引擎，密碼可能包含下列數量的可列印 ASCII 字元：
    - Aurora MySQL：8–41
    - Aurora PostgreSQL：8–99
- 資料庫參數群組：
  - 必須包含 1–255 個英數字元。
  - 第一個字元必須是字母。
  - 允許連字號，但名稱不得以連字號為結尾或包含兩個連續的連字號。
- 資料庫子網路群組：
  - 必須包含 1–255 個字元。
  - 允許英數字母、空格、連字號、底線和句點。

Amazon Aurora 大小限制

儲存體大小限制

Aurora 叢集磁碟區大小上限因引擎版本而異：

最多 256 TiB：

- Aurora PostgreSQL 版本：
  - 17.5 版和所有更新版本
  - 16.9 及更新版本
  - 15.13 及更新版本
- Aurora MySQL 3.10 版 (與 MySQL 8.0.42 相容) 及更新版本

最多 128 TiB：

- 所有較舊的 Aurora PostgreSQL 版本
- 所有可用的 Aurora MySQL 第 3 版、Aurora MySQL 第 2 版、第 2.09 版及更新版本

如需自動儲存體調整規模的詳細資訊，請參閱Aurora 儲存體如何自動調整大小。

若要監視剩餘的儲存空間，您可以使用 `AuroraVolumeBytesLeftTotal` 指標。如需詳細資訊，請參閱Amazon Aurora 的叢集層級指標。

SQL 資料表大小限制

對於 Aurora MySQL 資料庫叢集，表格大小上限為 64 TiB。對於 Aurora PostgreSQL 資料庫叢集，表格大小上限為 32 TiB。在進行大型資料表分割等操作時，建議您遵循資料表設計最佳實務。

表格空間 ID 限制

Aurora 的表格空間 ID 上限為 2147483647。如果您經常建立和捨棄資料表，請務必注意您的表格空間 ID，並計劃使用邏輯傾出。如需詳細資訊，請參閱使用 mysqldump 從 MySQL 到 Amazon Aurora 的邏輯遷移。

Amazon Aurora 故障診斷

下列各節可幫助您對 Amazon RDS 和 Amazon Aurora 資料庫執行個體的問題進行故障診斷。

主題

- 無法連線至 Amazon RDS 資料庫執行個體
- Amazon RDS 安全問題
- 重新設定資料庫執行個體擁有者密碼
- Amazon RDS 資料庫執行個體停機或重新開機
- Amazon RDS 資料庫參數變更未生效
- Amazon Aurora 中的可用記憶體問題
- Amazon Aurora MySQL 複寫問題

如需使用 Amazon RDS API 偵錯問題的相關資訊，請參閱對 Aurora 上的應用程式進行故障診斷。

無法連線至 Amazon RDS 資料庫執行個體

無法連線至資料庫執行個體時，下列是常見的原因：

- 傳入規則 – 您的本機防火牆強制執行的存取規則，與您獲授權可存取資料庫執行個體的 IP 地址可能不符。問題很可能在於安全群組中的傳入規則。

  預設情況下，資料庫執行個體不允許存取。存取是透過與 VPC 相關聯的安全群組授予，該群組允許可傳入和傳出資料庫執行個體的流量。如有必要，請將特定情況的傳入和傳出規則新增至安全群組。您可以指定 IP 地址、IP 地址範圍，或其他 VPC 安全群組。

  > **Note**
  >
  > 新增傳入規則時，您可以針對 Source (來源) 選擇 My IP (我的 IP)，以允許從瀏覽器中偵測到的 IP 位址存取資料庫執行個體。

  如需設定安全群組的詳細資訊，請參閱建立安全群組以存取 VPC 中的資料庫叢集。

> **Note**
>
> 不允許從 169.254.0.0/16 範圍內的 IP 地址連線的用戶端。這是「自動私有 IP 定址範圍」(APIPA)，它用於本機連結定址。

- 公開存取性– 若要從 VPC 外部連線至資料庫執行個體 (例如，使用用戶端應用程式)，則執行個體必須具有指派給它的公用 IP 位址。

  若要讓執行個體可供公開存取，請修改並選擇 Public accessibility (公開存取性) 下方的 Yes (是)。如需詳細資訊，請參閱在 VPC 中的網際網路中隱藏資料庫叢集。

- 連接埠：由於本機防火牆限制，建立資料庫執行個體時指定的連接埠無法用來傳送或接收通訊。若要判斷您的網路是否允許將指定的連線埠用於傳入和傳出通訊，請洽詢網路管理員。
- Availability (可用性) – 對於新建立的資料庫執行個體，資料庫執行個體的狀態為 `creating`，直到資料庫執行個體可供使用為止。狀態變更為 `available` 時，您便能連線至資料庫執行個體。根據資料庫執行個體的大小，可能需要最多 20 分鐘的時間，執行個體才會可用。
- 網際網路閘道 – 若要開放公開存取資料庫執行個體，其資料庫子網路群組中的子網路必須具備網際網路閘道。

  設定子網路的網際網路閘道

  1. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/rds/ ：// 開啟 Amazon RDS 主控台。
  2. 在導覽窗格中，選擇 Databases (資料庫)，然後選擇資料庫執行個體的名稱。
  3. 在 Connectivity & security (連線能力和安全性) 標籤中，寫下 VPC 下的 VPC ID，以及 Subnets (子網路) 下的子網路 ID。
  4. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
  5. 在導覽窗格中，選擇 Internet Gateways (網際網路閘道)。請確認已經有網際網路閘道連結到 VPC。否則，請選擇 Create Internet Gateway (建立網際網路閘道)，來建立網際網路閘道。選取網際網路閘道，然後選取 Attach to VPC (連結至 VPC)，接著遵循指示，來將此閘道連結至 VPC。
  6. 在導覽窗格中，選取 Subnets (子網)，然後選取子網。
  7. 在 Route Table (路由表) 索引標籤中，確認有包含 `0.0.0.0/0` 的路由做為目的地，以及有 VPC 的網際網路閘道做為目標。

如果您是使用其 IPv6 地址來連線到執行個體，請確認有針對所有 IPv6 流量 (`::/0`) 的路由，來指向網際網路閘道。若否，請執行下列操作：

a. 選擇路由表 ID (rtb-xxxxxxxx) 以導覽至路由表。

b. 在 Routes (路由) 標籤中，選擇 Edit routes (編輯路由)。選擇 Add route (新增路由)，使用 `0.0.0.0/0` 做為目的地，並以網際網路閘道為目標。

   若是 IPv6，請選擇 Add route (新增路由)，使用 `::/0` 做為目的地，並以網際網路閘道為目標。

c. 選擇 Save routes (儲存路由)。

此外，若要嘗試連線至 IPv6 端點，請確保用戶端 IPv6 地址範圍有權連線至資料庫執行個體。

如需詳細資訊，請參閱在 VPC 中使用資料庫叢集。

測試資料庫執行個體的連線

您可以使用通用的 Linux 或 Microsoft Windows 工具來測試您對資料庫執行個體的連線。

您可以從 Linux 或 Unix 終端機測試連線，方法是輸入下列命令。將 *DB-instance-endpoint* 取代為端點並將 *port* 取代為資料庫執行個體的連線埠。

```
nc -zv DB-instance-endpoint port
```

例如，下列示範範例命令和傳回值：

```
nc -zv postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299

  Connection to postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299 port [tcp/
vvr-data] succeeded!
```

Windows 使用者可以使用 Telnet 來測試對資料庫執行個體的連線。Telnet 動作僅支援用於測試連線。如果連線成功，動作不會傳回訊息。如果連線不成功，您會收到類似如下的錯誤訊息：

```
C:\>telnet sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com 819
```

```
 Connecting To sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com...Could not
open
 connection to the host, on port 819: Connect failed
```

如果 Telnet 動作成功傳回，您的安全群組即已正確設定。

> **Note**
>
> Amazon RDS 不接受網際網路控制訊息通訊協定 (ICMP) 流量，包括 ping。

對連線身分驗證進行故障診斷

在某些情況下，您可以連線至資料庫執行個體，但會收到身分驗證錯誤。在這些情況下，您可能會重設資料庫執行個體的主要使用者密碼。請修改 RDS 執行個體，以便執行此動作。

Amazon RDS 安全問題

為了避免安全問題，請勿將您的 AWS 帳戶 根使用者電子郵件地址和密碼用於使用者帳戶。最佳實務是使用您的根使用者來建立使用者，並將那些使用者指派給資料庫使用者帳戶。如果必要，也可以使用您的根使用者來建立其他使用者帳戶。

如需建立使用者的相關資訊，請參閱在您的 AWS 帳戶中建立 IAM 使用者。如需在 中建立使用者的資訊 AWS IAM Identity Center，請參閱在 IAM Identity Center 中管理身分。

錯誤訊息「無法擷取帳戶屬性，某些主控台功能可能受損。」

您可以基於數個原因而收到此錯誤。這可能是因為您的帳戶缺少權限，或是您的帳戶並未適當設定。如果您的帳戶是新帳戶，您可能就不需等待帳戶就緒。如果這是現有的帳戶，您的存取政策中可能缺乏許可，因此無法執行某些動作 (例如建立資料庫執行個體)。若要修正問題，您的管理員需要為您的帳戶提供必要角色。如需詳細資訊，請參閱 IAM 文件。

重新設定資料庫執行個體擁有者密碼

如果您遭到鎖定資料庫 叢集，則可以用主要使用者身分登入。然後，您可以重設其他系統管理使用者或角色的認證。如果您無法以主要使用者身分登入， AWS 帳戶擁有者可以重設主要使用者密碼。如需您可能需要重設哪些管理帳戶或角色的詳細資訊，請參閱主要使用者帳戶權限。

您可以使用 Amazon RDS 主控台、 AWS CLI 命令 modify-db-instance，或使用 ModifyDBInstance API 操作來變更資料庫執行個體密碼。如需在資料庫叢集中修改資料庫執行個體的詳細資訊，請參閱修改資料庫叢集中的資料庫執行個體。

Amazon RDS 資料庫執行個體停機或重新開機

重新啟動資料庫執行個體時，可能會發生資料庫執行個體當機。當資料庫執行個體進入防止受到存取的狀態，以及當資料庫重新啟動時，均可能發生當機。在您手動將資料庫執行個體重新開機時，就會重新開機。當您變更需要重新開機才會生效的資料庫執行個體設定時，也會發生重新開機。

當您變更需要重新開機才會生效的設定，或當您手動進行重新開機時，才會發生資料庫執行個體重新開機。如果您變更設定並要求該變更立即生效，即會立即發生重新開機。或者，它會在資料庫執行個體的維護時段期間發生。

發生下列其中一個動作時，資料庫執行個體會立即重新開機：

- 您將資料庫執行個體的備份保留期間從 0 變更為非零值，或從非零值變更為 0。然後將 Apply Immediately (立即套用) 設為 `true`。
- 您變更資料庫執行個體類別，並且將 Apply Immediately (立即套用) 設為 `true`。

發生下列其中一個動作時，維護時段期間會發生資料庫執行個體重新開機：

- 您將資料庫執行個體的備份保留期間從 0 變更為非零值，或從非零值變更為 0，並且將 Apply Immediately (立即套用) 設為 `false`。
- 您變更資料庫執行個體類別，並且將 Apply Immediately (立即套用) 設為 `false`。

變更資料庫參數群組中的靜態參數時，在與參數群組相關聯的資料庫執行個體重新開機之後，變更才會生效。變更需要手動重新開機。在維護時段期間，不會自動重新啟動資料庫執行個體。

Amazon RDS 資料庫參數變更未生效

在某些情況下，您可能會變更資料庫參數群組中的參數，但不會看到變更生效。若是如此，您可能需要重新啟動與 DB 參數群組關聯的資料庫執行個體。當您變更動態參數時，變更會立即生效。當您變更靜態參數時，在您重新啟動與參數群組關聯的資料庫實體之前，變更將不會生效。

您可以使用 RDS 主控台重新啟動資料庫執行個體。或者，您可以明確呼叫 `RebootDBInstance` API 操作。如果資料庫執行個體處於多可用區部署中，即可重新啟動，而不需容錯移轉。若在變更靜態參

數之後，要求重新啟動相關聯的資料庫執行個體，將有助於降低參數組態錯誤影響 API 呼叫的風險。其中一個例子是呼叫 `ModifyDBInstance` 變更資料庫執行個體類別。如需詳細資訊，請參閱修改 Amazon Aurora 中的資料庫參數群組中的參數。

Amazon Aurora 中的可用記憶體問題

可用記憶體是資料庫執行個體上可供資料庫引擎使用的隨機存取記憶體 (RAM) 總計。它是可用作業系統 (OS) 記憶體與可用緩衝區和頁面快取記憶體的總和。資料庫引擎使用主機上大多數的記憶體，但作業系統處理程序也使用一些 RAM。目前配置給資料庫引擎或作業系統處理程序目前所使用的記憶體不包括在可用記憶體中。當資料庫引擎記憶體不足時，資料庫執行個體會使用一般用於緩衝和快取的暫時空間。如前所述，此暫時空間包括在可用記憶體中。

您可以在 Amazon CloudWatch 中使用 `FreeableMemory` 指標以監控可用記憶體。如需詳細資訊，請參閱Amazon Aurora 的監控工具。

如果資料庫執行個體在可用記憶體方面持續不足或使用交換空間，請考慮縱向擴展為更大的資料庫執行個體類別。如需詳細資訊，請參閱Amazon Aurora 資料庫執行個體類別。

您也可以變更記憶體設定。例如，在 Aurora MySQL ，您可以調整 `innodb_buffer_pool_size` 參數的大小。依預設，此參數設定為實體記憶體的 75%。如需更多 MySQL 的疑難排解秘訣，請參閱如何疑難排解 Amazon RDS for MySQL 資料庫中可用記憶體不足的問題？

對於 Aurora Serverless v2，`FreeableMemory` 表示當 Aurora Serverless v2 資料庫執行個體擴展至其最大容量時，可用的未使用記憶體量。您可能已將執行個體縮小到相對較低的容量，但它仍然報告 `FreeableMemory` 的值很高，因為執行個體會擴展。這個記憶體目前不可用，但如果需要，您可以使其可供使用。

對於目前容量低於最大容量的每個 Aurora 容量單位 (ACU)，`FreeableMemory` 增加約 2 GiB。因此，在資料庫執行個體盡可能向上擴展之前，該指標不會接近零。

若此指標接近 `0` 值，則資料庫執行個體已盡可能地縱向擴展。它即將接近其可用記憶體的限制。考慮增加叢集的最大 ACU 設定。若此指標在讀取器資料庫執行個體上接近 `0` 的值，請考慮將其他讀取器資料庫執行個體新增至叢集。如此，工作負載的唯讀部分可分佈於更多資料庫執行個體上，進而減少每個讀取器資料庫執行個體上的記憶體用量。如需詳細資訊，請參閱Aurora Serverless v2 的重要 Amazon CloudWatch 指標。

針對 Aurora Serverless v1，您可以變更容量範圍以使用更多 ACU。如需詳細資訊，請參閱修改 Aurora Serverless v1 資料庫叢集。

Amazon Aurora MySQL 複寫問題

某些 MySQL 複寫問題也適用於 Aurora MySQL。您能夠診斷這些問題，並加以修正。

主題

- 診斷和解決僅供讀取複本之間的延遲
- 診斷和解決 MySQL 讀取複寫失敗
- 複寫已停止錯誤
- 僅供讀取複本複寫無法初始化中繼資料結構

診斷和解決僅供讀取複本之間的延遲

建立 MySQL 僅供讀取複本且僅供讀取複本可供使用之後，Amazon RDS 會先複寫從開始僅供讀取複本建立操作起始起，對來源資料庫執行個體進行的變更。在此階段期間，僅供讀取複本的複寫延遲時間將大於 0。您可以在 Amazon CloudWatch 中檢視 Amazon RDS `AuroraBinlogReplicaLag` 指標，進而監控此延遲時間。

該 `AuroraBinlogReplicaLag` 指標回報 MySQL `Seconds_Behind_Master` 命令的 `SHOW REPLICA STATUS` 欄位值。如需詳細資訊，請參閱 MySQL 文件中的 SHOW REPLICA STATUS 陳述式。

當 `AuroraBinlogReplicaLag` 指標到達 0，複本即已跟上來源資料庫執行個體。如果 `AuroraBinlogReplicaLag` 指標傳回 -1，複寫可能為非作用中。若要對複寫錯誤進行故障診斷，請參閱 診斷和解決 MySQL 讀取複寫失敗。如果 `AuroraBinlogReplicaLag` 的值為 -1，也可能表示系統無法判斷 `Seconds_Behind_Master` 值，或該值為 `NULL`。

> **Note**
>
> Aurora MySQL 以前的版本使用 `SHOW SLAVE STATUS` 而不是 `SHOW REPLICA STATUS`。如果您使用的是 Aurora MySQL 第 1 版或第 2 版，請使用 `SHOW SLAVE STATUS`。將 `SHOW REPLICA STATUS` 用於 Aurora MySQL 第 3 版及更新版本。

在網路中斷期間，或在維護時段套用修補程式時， `AuroraBinlogReplicaLag` 指標即會傳回 -1。在這種情況下，請等候系統還原網路連線，或是等待維護時段結束，隨後再重新檢查 `AuroraBinlogReplicaLag` 指標。

MySQL 讀取複製技術是非同步的。因此您可以預期來源資料庫執行個體的 BinLogDiskUsage 指標與僅供讀取複本上的 AuroraBinlogReplicaLag 指標偶爾會增加。例如，請考慮一個情況，其中來源資料庫執行個體可同時出現大量寫入操作。同時，會使用單一輸入/輸出執行緒，對僅供讀取複本的寫入操作進行序列化。這種情況可能會導致來源執行個體和僅供讀取複本之間發生延遲。

如需僅供讀取複本和 MySQL 的詳細資訊，請參閱 MySQL 文件中的複寫實作詳細資訊。

您可以利用執行下列動作，減少對來源資料庫執行個體的更新與對僅供讀取複本的後續更新之間的延遲：

- 將僅供讀取複本的資料庫執行個體類別設定為與來源資料庫執行個體具有相等的儲存體大小。
- 確保來源資料庫執行個體和僅供讀取複本所使用的資料庫參數群組中的參數設定相容。如需詳細資訊和範例，請參閱下一節中 max_allowed_packet 參數的討論。
- 停用查詢快取。針對經常修改的資料表，因為快取已遭鎖定並且經常重新整理，使用查詢快取可能會增加複本延遲。若是這種情況，如果停用查詢快取，您可能會發現複本延遲較少。您可以在資料庫執行個體的資料庫參數群組中將 query_cache_type parameter 設定為 0，以停用查詢快取。如需查詢快取的詳細資訊，請參閱查詢快取組態。
- 為 MySQL 的緩衝區集區。如果您有經常更新的一組小型資料表，並且使用 InnoDB 或 XtraDB 資料表結構描述。在這種情況下，傾印僅供讀取複本上的那些資料表。這麼做會造成資料庫引擎掃描來自磁碟之那些資料表的資料列，然後將它們快取在緩衝集區中。如此可減少複本延遲。下列顯示一個範例。

  對於 Linux、macOS 或 Unix：

```
PROMPT> mysqldump \
    -h <endpoint> \
    --port=<port> \
    -u=<username> \
    -p <password> \
    database_name table1 table2 > /dev/null
```

  在 Windows 中：

```
PROMPT> mysqldump ^
    -h <endpoint> ^
    --port=<port> ^
    -u=<username> ^
    -p <password> ^
    database_name table1 table2 > /dev/null
```

診斷和解決 MySQL 讀取複寫失敗

Amazon RDS 會監控僅供讀取複本的複寫狀態。如果複寫因任何原因停止，RDS 會將僅供讀取複本執行個體的 Replication State (複寫狀態) 欄位更新為 `Error`。您可以透過檢視 Replication Error (複寫錯誤) 欄位，檢閱 MySQL 引擎擲出之相關聯錯誤的詳細資訊。也會產生指出僅供讀取複本之狀態的事件，包括 RDS-EVENT-0045、RDS-EVENT-0046 和 RDS-EVENT-0057。如需事件和訂閱事件的詳細資訊，請參閱使用 Amazon RDS 事件通知。如果傳回 MySQL 錯誤訊息，請在 MySQL 錯誤訊息文件中檢閱錯誤。

可能造成複寫錯誤的常見情況包括下列：

- 僅供讀取複本之 `max_allowed_packet` 參數的值小於來源資料庫執行個體之 `max_allowed_packet` 參數的值。

  `max_allowed_packet` 參數是您可以在資料庫參數群組中設定的自訂參數。`max_allowed_packet` 參數可用來指定可在資料庫上執行的資料處理語言 (DML) 大小上限。在某些情況下，來源資料庫執行個體的 `max_allowed_packet` 值可能大於僅供讀取複本的 `max_allowed_packet` 值。若是如此，複寫程序會擲回錯誤並停止複寫。最常見的錯誤為 `packet bigger than 'max_allowed_packet' bytes`。您可以透過讓來源和僅供讀取複本使用具有相同 `max_allowed_packet` 參數值的資料庫參數群組，藉此修正錯誤。
- 寫入僅供讀取複本上的資料表。如果您在僅供讀取複本上建立索引，您需要將 `read_only` 參數設為 0 才能建立索引。如果您要寫入僅供讀取複本上的資料表，可能中斷複寫。
- 使用非交易儲存引擎 (例如 MyISAM)。僅供讀取複本需要交易儲存引擎。複寫只支持以下儲存引擎：MySQL 或 MariaDB 的 InnoDB。

  您可以使用下列命令，將 MyISAM 資料表轉換為 InnoDB：

  ```
  alter table <schema>.<table_name> engine=innodb;
  ```
- 使用不安全的非確定性查詢 (例如 `SYSDATE()`)。如需詳細資訊，請參閱 MySQL 文件中的二進位日誌記錄中安全和不安全陳述式的判定。

下列步驟有助於解決您的複寫錯誤：

- 如果發生邏輯錯誤，則可遵循略過目前複寫錯誤中所述的步驟，放心地略過此錯誤。您的 Aurora MySQL 資料庫執行個體必須執行包含 `mysql_rds_skip_repl_error` 程序的版本。如需詳細資訊，請參閱 mysql_rds_skip_repl_error。

- 如果遇到二進位日誌 (binlog) 位置問題，您可以變更複本重播位置。對於 Aurora MySQL 第 1 版和第 2 版，您可以使用 `mysql.rds_next_master_log` 命令執行此動作。對於 Aurora MySQL 第 3 版和更高版本，您可以使用 `mysql.rds_next_source_log` 命令執行此動作。您的 Aurora MySQL 資料庫執行個體必須執行支援此命令的版本，才能變更複本重播位置。如需版本資訊，請參閱 mysql_rds_next_master_log。
- 如果由於 DML 負載過高而遇到暫時的效能問題，您可以在僅供讀取複本上的資料庫參數群組中將 `innodb_flush_log_at_trx_commit` 參數設定為 2。這麼做有助於僅供讀取複本跟上，不過會暫時減少不可分割性、一致性、隔離和耐用性 (ACID)。
- 您可以刪除僅供讀取複本，並使用相同的資料庫執行個體識別符來建立執行個體。如此一來，端點仍會與舊的僅供讀取複本相同。

如果複寫錯誤已修復，Replication State (複寫狀態) 會變更為 replicating (複寫中)。如需詳細資訊，請參閱故障診斷 MySQL 僅供讀取複本的問題。

複寫已停止錯誤

當您呼叫 `mysql.rds_skip_repl_error` 命令時，可能會收到錯誤訊息，指出複寫已關閉或停用。

因為複寫已停止且無法重新啟動，因此出現此錯誤訊息。

如果您需要略過大量錯誤，複寫延遲可能增加至超出二進位記錄檔的預設保留期間。在此情況下，由於在清除二進位日誌檔之前已在複本上重播該檔案，您可能會遇到嚴重錯誤。此清除動作會導致複寫停止，而您將無法再呼叫 `mysql.rds_skip_repl_error` 命令來略過複寫錯誤。

透過增加二進位記錄檔在複寫主控端上保留的小時數，即可以減輕此問題。在延長二進位記錄檔保留時間之後，您可以重新啟動複寫，並視需要呼叫 `mysql.rds_skip_repl_error` 命令。

若要設定 binlog 保留時間，請使用 mysql_rds_set_configuration 程序。並指定 'binlog retention hours' 組態參數，以及資料庫叢集上保留二進位記錄檔的時數 (最多 2160 小時 (90 天))。Aurora MySQL 的預設值為 24 小時 (1 天)。下列範例會將 binlog 檔案的保留期間設定為 48 小時。

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

僅供讀取複本複寫無法初始化中繼資料結構

嘗試開始複寫時，您收到下列錯誤訊息：

```
Read Replica Replication Error - SQLError: 13124, reason: Replica failed to initialize
 applier metadata structure from the repository
```

當複本的中繼資料結構發生問題時，就會發生此錯誤。若要修正中繼資料結構，您必須建立新的複本。

若要防止未來發生這種情況，請執行下列其中一個動作：

- 如果可能，請在複本停用多執行緒。從 MySQL 8.0.27 開始，預設會啟用多執行緒。
- 如果您需要在複本使用多執行緒，我們建議您使用 GTID 式複寫。如需詳細資訊，請參閱使用 GTID 式複寫。

Amazon RDS API 參考

除了 AWS 管理主控台 和 AWS Command Line Interface (AWS CLI) 之外，Amazon RDS 還提供一個 API。您可以使用 API，將管理資料庫執行個體以及 Amazon RDS 中其他物件的任務自動化。

- 如需依字母排序的 API 操作清單，請參閱動作。
- 如需依字母排序的資料類型清單，請參閱資料類型。
- 如需常用查詢參數的清單，請參閱常用參數。
- 如需錯誤碼的說明，請參閱常見錯誤。

如需 AWS CLI 的詳細資訊，請參閱 Amazon RDS 的 AWS Command Line Interface 參考。

主題

- 使用查詢 API
- 對 Aurora 上的應用程式進行故障診斷

使用查詢 API

下節討論與查詢 API 搭配使用的參數和請求身分驗證。

如需查詢 API 運作方式的一般資訊，請參閱《Amazon EC2 API Reference》中的查詢請求。

查詢參數

HTTP 查詢型請求是使用 HTTP 動詞 GET 或 POST，以及名為 `Action` 之查詢參數的 HTTP 請求。

每一個查詢請求都須包括一些常用參數，來處理動作的身分驗證和選取。

部分操作有數個參數清單。這些清單是使用 `param.`*`n`* 表示法來指定的。*n* 的值是從 1 開始的整數。

如需 Amazon RDS 區域和端點的相關資訊，請參閱 Amazon Web Services 一般參考 中「區域與端點」一節的 Amazon Relational Database Service (RDS)。

查詢請求身分驗證

您只能透過 HTTPS 傳送查詢請求，而且必須在每一個查詢請求中包括一個簽章。您必須使用 AWS 簽章第 4 版或簽章第 2 版。如需詳細資訊，請參閱簽章第 4 版簽署程序和簽章第 2 版簽署程序。

對 Aurora 上的應用程式進行故障診斷

Amazon RDS 會提供特定和描述性錯誤，以協助您在與 Amazon RDS API 互動時進行故障診斷。

主題

- 擷取錯誤
- 對秘訣進行故障診斷

如需 Amazon RDS 資料庫執行個體疑難排解的資訊，請參閱 Amazon Aurora 故障診斷。

擷取錯誤

通常，您想要應用程式在您花費任何時間處理結果之前，先檢查請求是否已產生錯誤。若要了解系統是否發生錯誤，最簡單的方式即為在 Amazon RDS API 的回應中，尋找 `Error` 節點。

XPath 語法提供簡單的方式，來搜尋 `Error` 節點是否存在。它還提供了一種相對簡單的方式，來擷取錯誤碼和訊息。下列程式碼片段使用 Perl 和 XML::XPath 模組，來判斷請求期間是否發生錯誤。如果發生錯誤，程式碼會列印回應中的第一個錯誤碼和訊息。

```
use XML::XPath;
    my $xp = XML::XPath->new(xml =>$response);
    if ( $xp->find("//Error") )
    {print "There was an error processing your request:\n", " Error code: ",
    $xp->findvalue("//Error[1]/Code"), "\n", " ",
    $xp->findvalue("//Error[1]/Message"), "\n\n"; }
```

對秘訣進行故障診斷

我們建議以下列程序，來診斷並解決 Amazon RDS API 發生的問題。

- 檢查 http://status.aws.amazon.com，驗證 Amazon RDS 在目標 AWS 區域中是否能正常運作。
- 檢查請求的結構。

  在 Amazon RDS API 參考中，每項 Amazon RDS 操作都會有一個參考頁面。再次檢查您是否正確使用參數。如需有關可能出錯的概念，請查看範例請求或使用者案例，來查看那些範例是否執行類似操作。
- 檢查 AWS re:Post。

Amazon RDS 具有開發社群，您可在其中搜尋其他人在過程中所遇到問題的解決方案。如要檢視主題，請移至 AWS re:Post。

文件歷史記錄

目前的 API 版本：2014-10-31

下表說明 Amazon Aurora 使用者指南的重要變更。如需有關此文件更新的通知，您可以訂閱 RSS 摘要。如需 Amazon Relational Database Service (Amazon RDS) 的詳細資訊，請參閱《Amazon Relational Database Service 使用者指南》。

> **Note**
>
> 在 2018 年 8 月 31 日之前，已於 Amazon Relational Database Service 使用者指南中記載 Amazon Aurora。如需先前的 Aurora 文件歷史記錄，請參閱《Amazon Relational Database Service 使用者指南》中的文件歷史記錄。

您可以在資料庫的最新資訊頁面上篩選新的 Amazon Aurora 功能。對於 Products (產品) 篩選條件，請選擇 Amazon Aurora。然後使用關鍵字搜尋，例如 **`global database`** 或 **`Serverless`**。

變更	描述	日期
與 的 Aurora PostgreSQL 零 ETL 整合Amazon SageMaker 支援其他區域	與 的 Aurora PostgreSQL 零 ETL 整合現在Amazon SageMaker可在其他 區域使用。如需詳細資訊，請參閱適用於零 ETL 整合的支援區域和 Aurora 資料庫引擎。	2025 年 10 月 15 日
Aurora PostgreSQL 無限資料庫可在其他 AWS 區域中使用	Aurora PostgreSQL 無限資料庫現可在 中使用 AWS GovCloud (US) Regions。如需詳細資訊，請參閱 Aurora PostgreSQL 無限資料庫需求和考量事項。	2025 年 9 月 16 日
Amazon RDS Data API 支援 IPv6	Amazon RDS Data API 現在支援 IPv6 連線，可讓您使用 IPv6 地址連線至 Aurora 資料	2025 年 8 月 30 日

	庫。如需詳細資訊，請參閱使用 Data API。	
Amazon Aurora 可在亞太區域（紐西蘭）區域使用	Amazon Aurora 現已在亞太區域（紐西蘭）區域提供。如需更多詳細資訊，請參閱區域和可用區域。	2025 年 8 月 28 日
Amazon RDS Data API 可在其他 AWS 區域中使用	Amazon RDS Data API 現可於歐洲 (西班牙) 區域使用。如需詳細資訊，請參閱支援的 RDS Data API 區域和 Aurora 資料庫引擎。	2025 年 8 月 1 日
與 Amazon SageMaker 資料湖倉的零 ETL 整合已全面推出	零 ETL 整合可讓交易資料在寫入至 Aurora MySQL 資料庫執行個體後的幾秒內，即可在 Amazon SageMaker 中使用。如需詳細資訊，請參閱使用 Aurora 零 ETL 整合。	2025 年 6 月 30 日
Amazon Aurora 可在亞太區域（台北）區域使用	Amazon Aurora 現已在亞太區域（台北）區域提供。如需更多詳細資訊，請參閱區域和可用區域。	2025 年 6 月 5 日
Performance Insights 的終止服務資訊	AWS 已宣布績效詳情的end-of-life：2025 年 11 月 30 日。在此日期之後，Amazon RDS 將不再支援 Performance Insights 主控台體驗、彈性保留期間及其相關定價。建議您在此日期之前將資料庫叢集升級至 Database Insights。如需詳細資訊，請參閱在 Amazon Aurora 上使用 Performance Insights 監控資料庫負載。	2025 年 5 月 30 日

Aurora PostgreSQL Limitless Database 支援 CloudWatch Database Insights	您現在可以使用 CloudWatch Database Insights 大規模監控 Amazon Aurora PostgreSQL Limitless Database。如需詳細資訊，請參閱使用 CloudWatch Database Insights 監控 Amazon Aurora PostgreSQL Limitless Database。	2025 年 5 月 22 日
檢視開放原始碼引擎主要版本的支援日期	您現在可以使用 和 RDS API，在 RDS 標準支援和 RDS 延伸支援中檢視開放原始碼引擎主要版本的開始 AWS CLI 和結束日期相關資訊。如需詳細資訊，請參閱在 Amazon RDS 延長支援中檢視引擎版本的支援日期。	2025 年 5 月 20 日
Aurora PostgreSQL 零 ETL 整合支援其他區域	Aurora PostgreSQL 零 ETL 整合現在可在其他區域使用。如需詳細資訊，請參閱支援的區域和 Aurora 資料庫引擎，以與 Amazon Redshift 進行零 ETL 整合。	2025 年 2 月 17 日
Amazon Aurora 可在墨西哥（中部）區域使用	Amazon Aurora 現已在墨西哥（中部）區域提供。如需更多詳細資訊，請參閱區域和可用區域。	2025 年 1 月 14 日
Amazon Aurora 可在亞太區域（泰國）區域使用	Amazon Aurora 現已在亞太區域（泰國）區域提供。如需更多詳細資訊，請參閱區域和可用區域。	2025 年 1 月 7 日

Aurora Serverless v1 的終止服務資訊	AWS 已宣布 的end-of-lifeAurora Serverless v1：2025 年 3 月 31 日。我們強烈建議您在此日期之前，將任何 Aurora Serverless v1 資料庫叢集升級至 Aurora Serverless v2。升級可能涉及資料庫引擎主要版本編號的變更。因此，請務必在終止服務日期前規劃、測試和實作此轉換。從 2025 年 1 月 8 日開始，客戶將無法再使用 或 CLI 建立新的Aurora Serverless v1叢集 AWS 管理主控台 或執行個體。如需從 Aurora Serverless v1 升級至 Aurora Serverless v2 之升級程序的詳細資訊，請參閱從 Aurora Serverless v1 叢集升級至 Aurora Serverless v2。	2024 年 12 月 6 日
Aurora 支援 CloudWatch Database Insights	您現在可以使用 CloudWatch Database Insights 大規模監控 Amazon Aurora 資料庫叢集。如需詳細資訊，請參閱使用 CloudWatch Database Insights 監控 Amazon Aurora 資料庫。	2024 年 12 月 1 日
Amazon Aurora 支援 db.r8g 執行個體類別	您現在可以使用 db.r8g 執行個體類別，建立 Aurora 資料庫叢集。如需詳細資訊，請參閱 Aurora 資料庫執行個體類別。	2024 年 11 月 21 日

Aurora Serverless v2 支援自動暫停和繼續	在閒置一段時間後，Aurora Serverless v2 資料庫執行個體現在可以完全縮減至零個 Aurora 容量單位 (ACU)。若要啟用此功能，請將最小容量值設定為 0 ACU。如需詳細資訊，請參閱 Aurora Serverless v2 的自動暫停和繼續。	2024 年 11 月 20 日
Amazon Aurora 支援自動遷移 EC2 資料庫	您可以使用 RDS 主控台將 EC2 資料庫遷移至 Aurora。Aurora 使用 AWS Database Migration Service (AWS DMS) 遷移來源 EC2 資料庫。 AWS DMS 可讓您將關聯式資料庫遷移至 AWS 雲端。如需詳細資訊，請參閱使用 AWS Database Migration Service 將 EC2 資料庫自動遷移到 Amazon Aurora。	2024 年 11 月 20 日
AWS 進階 NodeJS 包裝函式已全面推出	Amazon Web Services (AWS) 進階 NodeJS 包裝函式是現有 NodeJS 驅動程式的補充和延伸功能。如需詳細資訊，請參閱使用 AWS 驅動程式連線至 Aurora 資料庫叢集。	2024 年 11 月 19 日
Amazon Aurora 支援 db.r7i 執行個體類別	您現在可以使用 db.r7i 執行個體類別，建立 Aurora 資料庫叢集。如需詳細資訊，請參閱 Aurora 資料庫執行個體類別。	2024 年 11 月 18 日

適用於 Aurora MySQL 的 RDS Data API 支援所有 Performance Insights 檢視	您現在可以使用 Performance Insights 的最高主機或最高應用程式檢視，來監控使用適用於 Aurora MySQL 的 RDS Data API 執行的 SQL 操作。如需 Performance Insights 支援的詳細資訊，請參閱使用 Performance Insights 監控 RDS Data API 查詢。	2024 年 10 月 31 日
Aurora PostgreSQL Limitless Database	Amazon Aurora PostgreSQL Limitless Database 是 Amazon Aurora 的全新、自動化、水平擴展 (碎片化) 功能。Aurora PostgreSQL Limitless Database 可讓您透過將資料庫工作負載分散到多個 Aurora 寫入器執行個體，同時維持將其用作單一資料庫的能力，以擴展超過現有的 Aurora 寫入輸送量和儲存限制。如需詳細資訊，請參閱 Amazon Aurora PostgreSQL Limitless Database。	2024 年 10 月 31 日
Amazon Aurora MySQL 第 2 版標準支援結束	Aurora MySQL 第 2 版已在 2024 年 10 月 31 日達到標準支援結束。如需詳細資訊，請參閱為 Amazon Aurora MySQL 相容版本第 2 版標準支援結束做好準備。	2024 年 10 月 31 日

Amazon Aurora 支援叢集層級作業系統升級	您現在可以在資料庫叢集層級套用 Aurora 作業系統升級。滾動升級一次會自動將升級套用至幾個讀取器資料庫執行個體，因而保留讀取可用性。如需詳細資訊，請參閱 Aurora 資料庫叢集的作業系統更新。	2024 年 10 月 30 日
Aurora 全域叢集支援標記	您現在可以將標籤新增至 Aurora 全域叢集。Aurora 全域叢集是父資源，其中包含 Aurora 全球資料庫中的主要和次要資料庫叢集。如需標記的一般資訊，請參閱標記 Aurora 和 Amazon RDS 資源。如需使用 Aurora 全球資料庫進行標記的詳細資訊，請參閱標記 Aurora 全球資料庫資源。	2024 年 10 月 24 日
Aurora 全球資料庫寫入器端點	每個 Aurora 全球資料庫都有一個寫入器端點，Aurora 會自動更新該端點，在 Aurora 全球資料庫轉換和容錯移轉操作之後，將請求路由到主要資料庫叢集的目前寫入器執行個體。若要進一步了解如何使用寫入器端點以及 Aurora 全球資料庫轉換和容錯移轉，請參閱在 Amazon Aurora 全球資料庫中使用切換和容錯移轉。	2024 年 10 月 22 日

與 Amazon Redshift 的 Aurora PostgreSQL 零 ETL 整合已全面推出	零 ETL 整合可讓交易資料在寫入至 Aurora PostgreSQL 來源資料庫叢集後的幾秒內，即可在 Amazon Redshift 中使用。此功能現已全面推出。如需詳細資訊，請參閱使用與 Amazon Redshift 的 Aurora 零 ETL 整合。	2024 年 10 月 15 日
Aurora Serverless v2 支援 256 個 ACU	您現在可以建立最大容量為 256 個 Aurora 容量單位 (ACU) 的 Aurora Serverless v2 資料庫叢集。如需詳細資訊，請參閱 Aurora Serverless v2 容量。	2024 年 10 月 3 日
Amazon Aurora 支援 Console-to-Code	您現在可以使用 Console-to-Code，從您在 RDS 主控台執行的動作產生程式碼。產生的程式碼可協助您編寫程式碼，以自動使用其他 AWS 服務。如需詳細資訊，請參閱使用 Console-to-Code 為您的 Aurora 主控台動作產生程式碼。	2024 年 10 月 3 日
RDS Data API 支援 Performance Insights	您現在可以使用 Performance Insights 來監控使用 RDS Data API 執行的 SQL 操作。如需 Performance Insights 支援的詳細資訊，請參閱使用 Performance Insights 監控 RDS Data API 查詢。	2024 年 9 月 26 日

適用於 Aurora Serverless v2 和佈建的 RDS Data API 可供 Aurora MySQL 使用	RDS Data API 現在可供使用 Aurora Serverless v2 或佈建執行個體的 Aurora MySQL 叢集使用。如需 RDS Data API 的使用資訊，請參閱使用 RDS Data API。如需 RDS Data API 的 Aurora MySQL 版本和 AWS 區域支援的相關資訊，請參閱使用 Aurora MySQL Serverless v2 和 佈建的資料 API。	2024 年 9 月 26 日
Amazon Aurora 可於亞太區域 (馬來西亞) 使用	Amazon Aurora 現在可於亞太區域 (馬來西亞) 使用。如需更多詳細資訊，請參閱區域和可用區域。	2024 年 8 月 22 日
更新現有政策	Amazon RDS 已從 `AWSServiceRoleForRDSBeta` 服務連結角色的 `AmazonRDSBetaServiceRolePolicy` 移除 `sns:Publish` 許可。如需詳細資訊，請參閱 Amazon RDS AWS 受管政策的更新。	2024 年 8 月 7 日
更新現有政策	Amazon RDS 已從 `AWSServiceRoleForRDSPreview` 服務連結角色的 `AmazonRDSPreviewServiceRolePolicy` 移除 `sns:Publish` 許可。如需詳細資訊，請參閱 Amazon RDS AWS 受管政策的更新。	2024 年 8 月 7 日

Aurora Serverless v2 支援自動暫停 (縮減為零)	在閒置一段時間後，Aurora Serverless v2 資料庫執行個體現在可以完全縮減至零個 Aurora 容量單位 (ACU)。若要啟用此功能，請將最小容量值設定為 0。如需更多詳細資訊，請參閱使用 Aurora Serverless v2。	2024 年 7 月 25 日
AWS 適用於 MySQL 的 ODBC 驅動程式已全面推出	Amazon Web Services (AWS) ODBC Driver for MySQL 是專為 Aurora MySQL 的高可用性而設計的用戶端驅動程式。如需詳細資訊，請參閱使用 Amazon Web Services (AWS) ODBC Driver for MySQL 連線至 Aurora MySQL。	2024 年 7 月 18 日
適用於 Aurora Serverless v2 的 RDS Data API 可在更多區域使用	RDS Data API 現已在數個其他 AWS 區域中可供 Aurora PostgreSQL 使用。如需 RDS Data API 的區域支援詳細資訊，請參閱使用 Aurora PostgreSQL Serverless v2 和佈建的 Data API。	2024 年 7 月 9 日
AWS Python 驅動程式已全面推出	Amazon Web Services (AWS) Python 驅動程式設計為進階 Python 包裝函式。此包裝函式是開放原始碼 Psycopg 驅動程式的補充和延伸功能。如需詳細資訊，請參閱使用 AWS 驅動程式連線至 Aurora 資料庫叢集。	2024 年 5 月 23 日

中國區域提供零 ETL 整合	中國 AWS 區域（北京）和中國（寧夏）現在提供零 ETL 整合。如需詳細資訊，請參閱與 Amazon Redshift 的零 ETL 整合。	2024 年 5 月 21 日
RDS Proxy 可在更多區域使用	RDS Proxy 現已可在亞太區域 (海德拉巴)、亞太區域 (墨爾本)、中東 (阿拉伯聯合大公國)、以色列 (特拉維夫)、加拿大西部 (卡加利) 和歐洲 (蘇黎世) 區域使用。如需 RDS Proxy 的詳細資訊，請參閱使用 Amazon RDS Proxy。	2024 年 5 月 21 日
Amazon RDS 支援 Performance Insights 的精細存取	您現在可以允許或拒絕對 Performance Insights 中個別維度的存取。此精細存取可用於 `GetResourceMetrics` 、`DescribeDimensionKeys` 和 `GetDimensionKeyDetails` 動作。如需詳細資訊，請參閱https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_PerfInsights.access-control.dimensionAccess-policy.html。	2024 年 5 月 21 日

Amazon RDS 延長支援	現在，在建立或還原 Aurora MySQL 第 2 版或第 3 版或 Aurora PostgreSQL 11 版資料庫時，會自動將該資料庫註冊到 Amazon RDS 延長支援中，以便您現有的應用程式繼續正常運作。您可以選擇退出 RDS 延長支援，以避免在資料庫引擎的 Aurora 標準支援日期結束後產生費用。如需詳細資訊，請參閱使用 Amazon RDS 延長支援。	2024 年 3 月 21 日
零 ETL 整合的資料篩選	Amazon RDS 支援在資料庫和資料表層級進行資料篩選，以便與 Amazon Redshift 進行零 ETL 整合。如需詳細資訊，請參閱使用資料篩選以便與 Amazon Redshift 進行 Aurora 零 ETL 整合。	2024 年 3 月 20 日
Aurora MySQL 與 Amazon Bedrock 整合	您現在可以將 Amazon Aurora MySQL 資料庫與 Amazon Bedrock 整合，以支援生成式 AI 應用程式。如需詳細資訊，請參閱搭配使用 Amazon Aurora機器學習與 Aurora MySQL。	2024 年 3 月 8 日

新的 AWS 受管政策	Amazon RDS 新增了名為 AmazonRDSCustom InstanceProfileRolePolicy 的新受管政策，以允許 RDS Custom 透過 EC2 執行個體設定檔執行自動化動作和資料庫管理任務。如需詳細資訊，請參閱 AWS 受管政策的 Amazon RDS 更新。	2024 年 2 月 27 日
Amazon RDS AWS Secrets Manager 在以色列（特拉維夫）區域的 支援	Amazon RDS 支援在以色列 (特拉維夫) 區域使用 Secrets Manager。如需詳細資訊，請參閱使用 Amazon RDS 和 AWS Secrets Manager進行密碼管理。	2024 年 2 月 21 日
Amazon RDS 延長支援	現在，當資料庫執行個體和全域叢集中的 Aurora MySQL 和 Aurora PostgreSQL 主要引擎版本達到 Aurora 標準支援結束日期時，Amazon RDS 會自動啟用 Amazon RDS 延長支援。如需詳細資訊，請參閱使用 Amazon RDS 延長支援。	2024 年 2 月 15 日
Aurora PostgreSQL 16.1 支援 Babelfish for Aurora PostgreSQL 4.0.0	Aurora PostgreSQL 16.1 支援 Babelfish 4.0.0。如需新功能的清單，請參閱 16.1。如需每個 Babelfish 版本支援的功能清單，請參閱 Babelfish 各版本支援的功能。如需用量詳細資訊，請參閱使用 Babelfish for Aurora PostgreSQL。	2024 年 1 月 31 日

更新為預設 CA 憑證	預設 CA 憑證設定為 `rds-ca-rsa2048-g1` 。如需詳細資訊，請參閱使用 SSL/TLS 加密資料庫叢集的連線。	2024 年 1 月 26 日
RDS Proxy 可於歐洲 (西班牙) 區域使用	RDS Proxy 現已可在歐洲 (西班牙) 區域使用。如需 RDS Proxy 的詳細資訊，請參閱使用 Amazon RDS Proxy。	2024 年 1 月 8 日
RDS Data API 搭配 Aurora PostgreSQL Serverless v2 與佈建	您現在可以將 RDS Data API 與 Aurora PostgreSQL Serverless v2 和佈建資料庫叢集搭配使用。使用 RDS Data API，您可以透過安全的 HTTP 端點存取 Aurora 叢集，並執行 SQL 陳述式，而無需使用資料庫驅動程式或管理連線。如需詳細資訊，請參閱使用 RDS Data API。	2023 年 12 月 21 日
Aurora PostgreSQL 與 Amazon Bedrock 整合	您現在可以將 Amazon Aurora PostgreSQL 資料庫與 Amazon Bedrock 整合，以支援生成式 AI 應用程式。如需詳細資訊，請參閱搭配使用 Amazon Aurora機器學習與 Aurora PostgreSQL。	2023 年 12 月 21 日
Amazon Aurora 可於加拿大西部 (卡加利) 區域使用	Amazon Aurora 現在可於加拿大西部 (卡加利) 區域使用。如需更多詳細資訊，請參閱區域和可用區域。	2023 年 12 月 20 日

Amazon RDS 支援檢視和回應建議	Amazon Aurora 建議現已包含閾值型主動建議和機器學習型被動建議。如需詳細資訊，請參閱 Amazon Aurora 的建議。	2023 年 12 月 19 日
與 Amazon Redshift 的 Aurora PostgreSQL 零 ETL 整合 (預覽)	您可以使用 Aurora PostgreSQL 來源資料庫叢集，建立與 Amazon Redshift 的零 ETL 整合。對於預覽版本，您必須在美國東部（俄亥俄）(us-east-2) 的 Amazon RDS 資料庫預覽環境中建立所有整合 AWS 區域。如需詳細資訊，請參閱使用與 Amazon Redshift 的 Aurora 零 ETL 整合。	2023 年 11 月 28 日
Amazon Aurora PostgreSQL 支援全域資料庫寫入轉送	您現在可以在 Aurora PostgreSQL 型全域資料庫中啟用次要叢集上的寫入轉送。如需詳細資訊，請參閱在 Aurora PostgreSQL 全域資料庫中使用寫入轉送。	2023 年 11 月 9 日
Optimized Reads 支援 Aurora PostgreSQL	您可以使用 Aurora Optimized Reads，為 Aurora PostgreSQL 實現更快的查詢處理。如需詳細資訊，請參閱使用 Amazon RDS Optimized Reads 改善 Aurora PostgreSQL 的查詢效能。	2023 年 11 月 8 日

Amazon RDS 會將 Performance Insights 指標匯出至 Amazon CloudWatch	Performance Insights 可讓您將預先設定或自訂指標儀表板匯出至 Amazon CloudWatch。匯出的指標儀表板可在 CloudWatch 主控台中檢視。您也可以在 CloudWatch 主控台中匯出選取的 Performance Insights 指標小工具，並檢視指標資料。如需詳細資訊，請參閱將 Performance Insights 指標匯出至 CloudWatch。	2023 年 11 月 8 日
與 Amazon Redshift 一般可用性的 Aurora MySQL 零 ETL 整合	Amazon Redshift 的零 ETL 整合現在廣泛地適用於 Aurora MySQL。如需詳細資訊，請參閱使用與 Amazon Redshift 的 Aurora 零 ETL 整合。	2023 年 11 月 7 日
藍/綠部署支援 Aurora PostgreSQL	您現在可以從 Aurora PostgreSQL 資料庫叢集建立藍/綠部署。如需詳細資訊，請參閱使用 Amazon RDS 藍/綠部署進行資料庫更新。	2023 年 10 月 26 日

Aurora MySQL 支援使用 AWS KMS keys (SSE-KMS) 的伺服器端加密	在 Aurora MySQL 3.05 版及更高版本中，您可以使用 SSE-KMS，包括 AWS 受管金鑰 和 客戶受管金鑰，對您從 Amazon S3 載入或儲存至 Amazon S3 的資料進行伺服器端加密。如需詳細資訊，請參閱從 Amazon S3 儲存貯體中的文字檔案將資料載入 Amazon Aurora MySQL 資料庫叢集將來自 Amazon Aurora MySQL 資料庫叢集的資料儲存至 Amazon S3 儲存貯體中的文字檔案。	2023 年 10 月 25 日
Aurora MySQL 最佳化會減少資料庫重新啟動時間	在 Aurora MySQL 3.05 版及更新版本中，我們引入了可減少資料庫重新啟動時間的最佳化。這些最佳化比沒有最佳化時可減少高達 65% 的停機時間，並且在重新啟動之後減少資料庫工作負載的中斷次數。如需詳細資訊，請參閱最佳化以減少資料庫重新啟動時間。	2023 年 10 月 25 日
更新受 AWS 管政策	`AmazonRDSPerformanceInsightsReadOnly` 和 `AmazonRDSPerformanceInsightsFullAccess` 受管政策現在包含 `Sid` (陳述式 ID) 作為政策陳述式中的識別符。如需詳細資訊，請參閱 AWS 受管政策的 Amazon RDS 更新。	2023 年 10 月 23 日

Amazon RDS 對 Amazon CloudWatch 發布了 Performance Insights 計數器指標	CloudWatch 主控台中的 DB_PERF_INSIGHTS 指標數學函數可讓您查詢 Amazon RDS for Performance Insights 計數器指標。如需詳細資訊，請參閱建立 Amazon CloudWatch 警示來監控 Amazon Aurora。	2023 年 9 月 20 日
Amazon Aurora 支援point-in-time(PITR) AWS Backup	您現在可以在 中管理 Aurora 自動化（連續）備份 AWS Backup ，並從中還原至指定的時間。如需詳細資訊，請參閱使用 AWS Backup將資料庫叢集還原到指定的時間。	2023 年 9 月 7 日
Amazon RDS 延長支援	Amazon Aurora 宣布，將在 Aurora 標準支援結束日期過後，提供在您的資料庫執行個體中繼續執行 Aurora MySQL 和 Aurora PostgreSQL 主要引擎版本的功能。如需詳細資訊，請參閱使用 Amazon RDS 延長支援。	2023 年 9 月 1 日
Amazon Aurora MySQL 延長了對 Percona XtraBackup 的支援	您現在可以執行 MySQL 8.0 資料庫到 Aurora MySQL 第 3 版資料庫叢集的實體遷移。如需詳細資訊，請參閱使用 Percona XtraBackup 和 Amazon S3 從 MySQL 進行實體遷移。	2023 年 8 月 24 日

Aurora 全球資料庫支援全球資料庫容錯移轉	Aurora 全球資料庫現在支援受管全球容錯移轉，可讓您更輕鬆地從重大的區域災難或服務完全中斷的情況復原。若要深入了解此功能，請參閱對 Aurora 全球資料庫執行受管容錯移轉。舊稱為「受管計畫容錯移轉」的功能，現在改稱為「轉換」。如需有關轉換的資訊，請參閱對 Amazon Aurora 全球資料庫執行轉換。	2023 年 8 月 21 日
更新受 AWS 管政策許可	AmazonRDSFullAccess 受管政策具有新的權限，可讓您產生、檢視和刪除一段時間區間內的效能分析報告。如需詳細資訊，請參閱 Amazon RDS AWS 受管政策的更新。	2023 年 8 月 17 日
更新受 AWS 管政策許可	透過向受管政策 AmazonRDSPerformanceInsightsReadOnly 新增新許可權和新增新的受管政策 AmazonRDSPerformanceInsightsFullAccess ，您可以產生一段時間區間內的資料庫負載分析報告。如需詳細資訊，請參閱 Amazon RDS AWS 受管政策的更新。	2023 年 8 月 16 日
Amazon RDS 支援使用 Performance Insights 進行資料庫負載時間區間分析	Performance Insights 可讓您建立特定時間區間內的績效分析報告。該報告提供識別出的洞見和解決效能問題的建議。如需詳細資訊，請參閱分析一段時間區間內的資料庫負載。	2023 年 8 月 16 日

Amazon Aurora 支援保留資料庫叢集的自動備份	您現在可以保留已刪除 Aurora 叢集的自動備份，並將其還原到指定的時間點。如需詳細資訊，請參閱保留自動備份。	2023 年 8 月 4 日
Amazon Aurora 可在以色列 (特拉維夫) 區域使用	Amazon Aurora 現可於以色列 (特拉維夫) 區域使用。如需更多詳細資訊，請參閱區域和可用區域。	2023 年 8 月 1 日
Amazon Aurora MySQL 支援本機 (叢集內) 寫入轉送	您現在可以將寫入作業從讀取器資料庫執行個體轉送至 Aurora MySQL 資料庫叢集中的寫入器資料庫執行個體。如需詳細資訊，請參閱在 Amazon Aurora MySQL 資料庫叢集中使用寫入轉送。	2023 年 7 月 31 日
Amazon Aurora 支援Aurora Serverless v2額外的 AWS 區域	您現在可以在亞太區域 Aurora Serverless v2 （墨爾本） 建立資料庫叢集 AWS 區域。如需 Aurora Serverless v2 的相關資訊，請參閱使用 Aurora Serverless v2。	2023 年 6 月 28 日
Amazon Aurora 推出與 Amazon Redshift 的零 ETL 整合 (預覽)	Amazon Aurora 零 ETL 整合提供全受管解決方案，其可讓交易資料在寫入 Aurora MySQL 資料庫叢集之後，幾秒內可在 Amazon Redshift 中使用。如需詳細資訊，請參閱使用與 Amazon Redshift 的 Aurora 零 ETL 整合。	2023 年 6 月 28 日

Amazon RDS 會在績效詳情儀表板中提供組合的績效詳情和 CloudWatch 指標檢視	Amazon RDS 現在會在績效詳情儀表板中提供績效詳情和 CloudWatch 指標的合併檢視。如需詳細資訊，請參閱使用 Performance Insights 儀表板檢視組合指標。	2023 年 5 月 24 日
Amazon Aurora 支援 db.r7g 執行個體類別	您現在可以使用 db.r7g 執行個體類別，建立 Aurora 資料庫叢集。如需詳細資訊，請參閱 Aurora 資料庫執行個體類別。	2023 年 5 月 11 日
Amazon Aurora 支援新的資料庫叢集儲存組態	使用 Aurora I/O-Optimized，您只需為資料庫叢集的用量和儲存付費，而讀取和寫入 I/O 操作無須額外付費。如需詳細資訊，請參閱 Amazon Aurora 資料庫叢集的儲存組態。	2023 年 5 月 11 日
Amazon Aurora 在其他 Aurora Serverless v2中支援 AWS 區域	您現在可以在下列項目中建立資料庫叢集 AWS 區域：亞太區域 Aurora Serverless v2（海德拉巴）、歐洲（西班牙）、歐洲（蘇黎世）和中東（阿拉伯聯合大公國）。如需 Aurora Serverless v2 的相關資訊，請參閱使用 Aurora Serverless v2。	2023 年 5 月 4 日
Aurora Serverless v1 支援轉換為已佈建	您可將 Aurora Serverless v1 資料庫叢集直接轉換為已佈建資料庫叢集。如需詳細資訊，請參閱將 Aurora Serverless v1 資料庫叢集轉換為已佈建。	2023 年 4 月 27 日

Aurora Serverless v1 支援 Amazon Aurora PostgreSQL 第 13 版	您現在可以建立執行 Aurora PostgreSQL 第 13 版的 Aurora Serverless v1 資料庫叢集。如需詳細資訊，請參閱Aurora Serverless v1。	2023 年 4 月 27 日
中國區域的 Amazon Aurora AWS Secrets Manager 支援	Amazon Aurora 在中國 (北京) 和中國 (寧夏) 區域支援 Secrets Manager。如需詳細資訊，請參閱使用 Amazon Aurora 和 AWS Secrets Manager進行密碼管理。	2023 年 4 月 20 日
Amazon Aurora 支援將事件與標籤發佈至主題訂閱者	傳送至 Amazon Simple Notification Service (Amazon SNS) 或 Amazon EventBridge 的 Amazon Aurora 事件通知，現在會在訊息本文中包含事件標籤。這些標籤提供受服務事件影響的資源資料。如需詳細資訊，請參閱 Amazon RDS 事件通知標籤和屬性。	2023 年 4 月 17 日
更新至 IAM 服務連結角色許可	`AmazonRDSFullAccess` 和 `AmazonRDSReadOnlyAccess` 政策現在會授予額外許可，以允許在 RDS 主控台中顯示 Amazon DevOps Guru 調查結果。如需詳細資訊，請參閱 Amazon RDS AWS 受管政策的更新。	2023 年 3 月 30 日

Amazon Aurora 在亞太區域 (墨爾本) 區域支援全球資料庫	您現在可在亞太區域 (墨爾本) 區域建立 Aurora 全球資料庫。如需 Aurora Global Database 的相關資訊，請參閱使用 Amazon Aurora 全球資料庫。	2023 年 3 月 22 日
更新受 AWS 管政策許可	`AmazonRDSFullAccess` 和 `AmazonRDSReadOnlyAccess` 政策現在會將額外許可授予 Amazon CloudWatch。如需詳細資訊，請參閱 Amazon RDS AWS 受管政策的更新。	2023 年 3 月 16 日
RDS Proxy 已在中國區域提供	RDS Proxy 現已在中國 (北京) 和中國 (寧夏) 區域提供。如需 RDS Proxy 的詳細資訊，請參閱使用 Amazon RDS Proxy。	2023 年 3 月 15 日
Amazon Aurora 在中國區域支援 Aurora Serverless v2	Aurora Serverless v2 現已在中國 (北京) 和中國 (寧夏) 區域提供。如需詳細資訊，請參閱Aurora Serverless v2。	2023 年 3 月 15 日
RDS Proxy 已在亞太區域 (雅加達) 區域提供	RDS Proxy 現已在亞太區域 (雅加達) 區域提供。如需 RDS Proxy 的詳細資訊，請參閱使用 Amazon RDS Proxy。	2023 年 3 月 8 日
Amazon Aurora MySQL 支援 Kerberos 身分驗證	您現在可以使用 Kerberos 身分驗證來在使用者連線到您的 Aurora MySQL 資料庫叢集時對其進行身分驗證。如需詳細資訊，請參閱針對 Aurora MySQL 使用 Kerberos 身分驗證。	2023 年 3 月 8 日

Amazon Aurora 在其他 中支援全域資料庫 AWS 區域	您現在可在下列區域建立 Aurora 全球資料庫：非洲 (開普敦)、亞太區域 (香港)、亞太區域 (海德拉巴)、亞太區域 (雅加達)、歐洲 (米蘭)、歐洲 (西班牙)、歐洲 (蘇黎世)、中東 (巴林) 和中東 (阿拉伯聯合大公國)。如需 Aurora Global Database 的相關資訊，請參閱使用 Amazon Aurora 全球資料庫。	2023 年 3 月 6 日
Amazon Aurora 支援在其他 中複製資料庫叢集快照 AWS 區域	您現在可在下列區域複製資料庫叢集快照：非洲 (開普敦)、亞太區域 (香港)、亞太區域 (海德拉巴)、亞太區域 (雅加達)、亞太區域 (墨爾本)、歐洲 (米蘭)、歐洲 (西班牙)、歐洲 (蘇黎世)、中東 (巴林) 和中東 (阿拉伯聯合大公國)。如需複製資料庫叢集快照的相關資訊，請參閱複製資料庫叢集快照。	2023 年 3 月 6 日
Amazon DevOps Guru for RDS 支援主動洞見	Amazon DevOps Guru for RDS 會發佈主動洞見及建議，協助您在 Aurora 資料庫的問題發生之前就解決。如需詳細資訊，請參閱 DevOps Guru for RDS 的運作方式。	2023 年 2 月 28 日
Amazon Aurora MySQL 第 1 版已被取代	Aurora MySQL 第 1 版 (與 MySQL 5.6 相容) 已被取代。如需詳細資訊，請參閱 Amazon Aurora 主要版本可用的時間會維持多久。	2023 年 2 月 28 日

Aurora Serverless v1 支援設定資料庫叢集的維護時段	您現在可以設定 Aurora Serverless v1 資料庫叢集的維護時段。如需詳細資訊，請參閱調整偏好的資料庫叢集維護時段。	2023 年 2 月 27 日
Amazon Aurora 支援亞太區域 (海德拉巴)、歐洲 (西班牙) 和中東 (阿拉伯聯合大公國) 區域中的資料庫活動串流。	如需詳細資訊，請參閱資料庫活動串流。	2023 年 1 月 27 日
Amazon Aurora 可於亞太區域 (墨爾本) 區域使用	Amazon Aurora 現可於亞太區域 (墨爾本) 區域使用。如需更多詳細資訊，請參閱區域和可用區域。	2023 年 1 月 23 日
在建立資料庫叢集期間指定憑證授權單位 (CA)	您現在可以指定哪個 CA 要在建立資料庫叢集期間用於資料庫叢集的伺服器憑證。如需詳細資訊，請參閱憑證授權單位。	2023 年 1 月 5 日
Aurora MySQL 3.* 支援恢復	Aurora MySQL 3.* 現在可從使用者的錯誤操作中迅速復原，像是捨棄不該捨棄的資料表或刪除不該刪除的列。恢復可讓您將資料庫回溯至先前的時間點，無需從備份還原，在數秒內即可完成，即使是大型資料庫也一樣。如需詳細資訊，請參閱回溯 Aurora 資料庫叢集。	2023 年 1 月 4 日
使用其他 中可用的 Amazon RDS 藍/綠部署 AWS 區域	現可在中國 (北京) 和中國 (寧夏) 區域使用藍/綠部署功能。如需詳細資訊，請參閱使用 Amazon RDS 藍/綠部署進行資料庫更新。	2022 年 12 月 22 日

更新至 IAM 服務連結角色許可	AmazonRDSServiceRolePolicy 政策現在授予其他許可給 AWS Secrets Manager。如需詳細資訊，請參閱 Amazon RDS AWS 受管政策的更新。	2022 年 12 月 22 日
Amazon Aurora 與 整合 AWS Secrets Manager 以進行密碼管理	Aurora 可以在 Secrets Manager 中管理資料庫叢集的主要使用者密碼。如需詳細資訊，請參閱使用 Amazon Aurora 和 AWS Secrets Manager進行密碼管理。	2022 年 12 月 22 日
Amazon Aurora 在其他 Aurora Serverless v2中支援 AWS 區域	Aurora Serverless v2 現已在非洲 (開普敦) 和歐洲 (米蘭) 區域提供。如需詳細資訊，請參閱Aurora Serverless v2。	2022 年 12 月 21 日
Aurora PostgreSQL 支援 RDS Proxy 搭配 PostgreSQL 14	您現在可以使用 Aurora PostgreSQL 14 資料庫叢集建立 RDS Proxy。如需 RDS Proxy 的詳細資訊，請參閱使用 Amazon RDS Proxy。	2022 年 12 月 13 日
Amazon Aurora 提醒您 Amazon DevOps Guru 最近偵測到的異常	主控台的資料庫詳細資訊頁面會提醒您目前狀態，以及過去 24 小時內發生的異常。如需詳細資訊，請參閱 DevOps Guru for RDS 的運作方式。	2022 年 12 月 13 日
Amazon RDS Proxy 支援全域資料庫	您現在可以搭配 Aurora 全域資料庫使用 RDS Proxy。如需詳細資訊，請參閱搭配 Aurora 全域資料庫使用 RDS Proxy。	2022 年 12 月 7 日

Aurora PostgreSQL 資料庫叢集支援適用於 PostgreSQL 的受信任語言延伸模組	適用於 PostgreSQL 的受信任語言延伸模組是開放原始碼開發套件，可讓您建置高效能 PostgreSQL 延伸模組，並在 Aurora PostgreSQL 資料庫叢集上安全地執行這些延伸模組。如需詳細資訊，使用適用於 PostgreSQL 的受信任語言延伸模組。	2022 年 11 月 30 日
Amazon GuardDuty RDS Protection 會監控存取威脅	當您開啟 GuardDuty RDS Protection 時，GuardDuty 會從您的 Aurora 資料庫取用 RDS 登入事件、監控這些事件，並對其進行描述，以找出潛在的內部威脅或外部演員。當 GuardDuty RDS Protection 偵測到潛在威脅時，GuardDuty 會產生新的問題清單，其中包含可能洩露之資料庫的詳細資訊。如需詳細資訊，請參閱使用 GuardDuty RDS Protection 監控威脅。	2022 年 11 月 30 日
使用 Amazon RDS 藍/綠部署進行資料庫更新	您可以在預備環境中對資料庫叢集進行變更，並在不影響生產資料庫叢集的情況下測試變更。備妥後，您可以將預備環境提升為新的生產環境，停機時間很短。如需詳細資訊，請參閱使用 Amazon RDS 藍/綠部署進行資料庫更新。	2022 年 11 月 27 日
Amazon Aurora 可於亞太區域 (海德拉巴) 區域使用	Amazon Aurora 現可於亞太區域 (海德拉巴) 區域使用。如需更多詳細資訊，請參閱區域和可用區域。	2022 年 11 月 22 日

Amazon Aurora 可於歐洲 (西班牙) 區域使用	Amazon Aurora 現可於歐洲 (西班牙) 區域使用。如需更多詳細資訊，請參閱區域和可用區域。	2022 年 11 月 16 日
Amazon Aurora 可於歐洲 (蘇黎世) 區域使用	Amazon Aurora 現可於歐洲 (蘇黎世) 區域使用。如需更多詳細資訊，請參閱區域和可用區域。	2022 年 11 月 9 日
Amazon Aurora Proxy 支援將資料從資料庫叢集匯出至 Amazon S3	您現在可以將 Aurora 叢集資料直接匯出到 S3，而不必先建立快照。如需詳細資訊，請參閱將資料庫叢集資料匯出到 Amazon S3。	2022 年 10 月 27 日
Amazon Aurora MySQL 支援更快匯出到 Amazon S3	對於 MySQL 5.7 和 8.0 相容的 Aurora MySQL 叢集，您現在可以看到將資料庫叢集快照資料匯出到 S3 時，速度效能提升高達 10 倍。如需詳細資訊，請參閱將資料庫叢集快照資料匯出至 Amazon S3。	2022 年 10 月 20 日
Amazon Aurora 支援在 Aurora 資料庫叢集與 EC2 執行個體之間自動設定連線	您可以使用 AWS 管理主控台來設定現有 Aurora 資料庫叢集與 EC2 執行個體之間的連線。如需詳細資訊，請參閱自動連線 EC2 執行個體和 Aurora 資料庫叢集。	2022 年 10 月 14 日

AWS JDBC Driver for PostgreSQL 已全面推出	AWS JDBC Driver for PostgreSQL 是專為 Aurora PostgreSQL 設計的用戶端驅動程式。 AWS JDBC Driver for PostgreSQL 現已正式推出。如需詳細資訊，請參閱與 AWS JDBC Driver for PostgreSQL 連線。	2022 年 10 月 6 日
Amazon Aurora 支援 MySQL 5.7 相容的 Aurora MySQL 進行就地升級	您可以執行就地升級，將現有 MySQL 5.7 相容的 Aurora MySQL 叢集變更為 MySQL 8.0 相容的 Aurora MySQL 叢集。如需詳細資訊，請參閱從 Aurora MySQL 2.x升級至 3.x。	2022 年 9 月 26 日
Performance Insights 顯示前 25 個 SQL 查詢	在 Performance Insights 儀表板中，Top SQL (最高 SQL) 索引標籤會顯示在資料庫負載中佔最大比例的 25 個 SQL 查詢。如需詳細資訊，請參閱最高 SQL 索引標籤概觀。	2022 年 9 月 13 日
Amazon Aurora 支援新的資料庫執行個體類別	您現在可以針對 Aurora MySQL 資料庫叢集使用 db.r6i 資料庫執行個體類別叢集。如需更多詳細資訊，請參閱資料庫執行個體類別。	2022 年 9 月 13 日
Amazon Aurora 可於中東 (阿拉伯聯合大公國) 區域使用	Amazon Aurora 現可於中東 (阿拉伯聯合大公國) 區域使用。如需更多詳細資訊，請參閱區域和可用區域。	2022 年 8 月 30 日

Amazon Aurora 支援自動設定 EC2 執行個體的連線	當您建立 Aurora 資料庫叢集時，您可以使用 AWS 管理主控台 來設定 Amazon Elastic Compute Cloud 執行個體與新資料庫叢集之間的連線。如需詳細資訊，請參閱《使用 EC2 執行個體設定自動網路連線》。	2022 年 8 月 22 日
Amazon Aurora 支援雙堆疊模式	資料庫叢集現在可以雙堆疊模式執行。於雙堆疊模式中，資源可透過 IPv4、IPv6 或兩者與資料庫叢集進行通訊。如需詳細資訊，請參閱 Amazon Aurora IP 地址。	2022 年 8 月 17 日
Amazon Aurora 支援 PostgreSQL 相容的就地升級 Aurora Serverless v1	您可以針對 PostgreSQL 10 相容 Aurora Serverless v1 叢集執行就地升級，將現有叢集變更為 PostgreSQL 11 相容 Aurora Serverless v1 叢集。有關就地升級程序，請參閱修改 Aurora Serverless v1 資料庫叢集。	2022 年 8 月 8 日
績效詳情支援亞太區域 (雅加達) 區域使用	之前您無法在亞太區域 (雅加達) 使用績效詳情。此限制已經移除。如需詳細資訊，請參閱績效詳情的AWS 區域 支援。	2022 年 7 月 21 日
Amazon Aurora 支援新的資料庫執行個體類別	您現在可以對 Aurora PostgreSQL 資料庫叢集使用 db.r6i 資料庫執行個體類別叢集。如需更多詳細資訊，請參閱資料庫執行個體類別。	2022 年 7 月 14 日

RDS 績效詳情支援額外的保留期間	之前，績效詳情僅提供兩個保留期間：7 天 (預設) 或 2 年 (731 天)。現在，如果您需要保留績效資料超過 7 天，可以指定 1-24 個月。如需詳細資訊，請參閱績效詳情的定價和資料保留。	2022 年 7 月 1 日
Amazon Aurora 支援就地升級與 MySQL 相容 Aurora Serverless v1	您可以針對 MySQL 5.6 相容 Aurora Serverless v1 叢集執行就地升級，將現有叢集變更為 MySQL 5.7 相容 Aurora Serverless v1 叢集。有關就地升級程序，請參閱修改 Aurora Serverless v1 資料庫叢集。	2022 年 6 月 16 日
Aurora 支援在 RDS 主控台中開啟 Amazon DevOps Guru	您可以從 RDS 主控台為 Aurora 資料庫開啟 DevOps Guru 涵蓋範圍。如需詳細資訊，請參閱設定 DevOps Guru for RDS。	2022 年 6 月 9 日
Amazon Aurora 支援將事件發佈到加密的 Amazon SNS 主題	Amazon Aurora 現在可以將事件發佈到已啟用伺服器端加密 (SSE) 的 Amazon Simple Notification Service (Amazon SNS) 主題，以便為帶有敏感資料的事件提供額外保護。如需詳細資訊，請參閱訂閱 Amazon RDS 事件通知。	2022 年 6 月 1 日

Amazon RDS 會將用量指標發佈至 Amazon CloudWatch	Amazon CloudWatch 中的 `AWS/Usage` 命名空間包括您 Amazon RDS Service Quotas 的帳戶層級的用量指標。如需詳細資訊，請參閱 Amazon Aurora 的 Amazon CloudWatch 用量指標。	2022 年 4 月 28 日
JSON 格式的資料 API 結果集	`ExecuteStatement` 函數的選用參數會造成查詢結果集以 JSON 格式的字串形式傳回。JSON 結果集可以簡單方便地轉換為應用程式語言的資料結構。如需詳細資訊，請參閱以 JSON 格式處理查詢結果。	2022 年 4 月 27 日
Amazon Aurora Serverless v2 現已全面推出	Amazon Aurora Serverless v2 通常適用於所有使用者。如需更多詳細資訊，請參閱使用 Aurora Serverless v2。	2022 年 4 月 21 日
Aurora MySQL 支援可設定的密碼套件	利用 Aurora MySQL，您現在可使用可設定的密碼套件來控制資料庫伺服器接受的連線加密。如需詳細資訊，請參閱 為 Aurora MySQL 資料庫叢集的連線設定密碼套件。	2022 年 4 月 15 日
Aurora PostgreSQL 支援 RDS Proxy 搭配 PostgreSQL 13	您現在可以使用 Aurora PostgreSQL 13 資料庫叢集建立 RDS Proxy。如需 RDS Proxy 的詳細資訊，請參閱使用 Amazon RDS Proxy。	2022 年 4 月 4 日

Aurora PostgreSQL 版本備註	現在有一個單獨的 Amazon Aurora PostgreSQL 版本備註的指南。如需詳細資訊，請參閱「Aurora PostgreSQL 版本備註」。	2022 年 3 月 22 日
Aurora MySQL 版本備註	現在有一個單獨的 Amazon Aurora MySQL 版本備註的指南。如需詳細資訊，請參閱《Aurora MySQL 版本備註》。	2022 年 3 月 22 日
Aurora PostgreSQL 支援多個主要版本升級	現在，您可以跨多個主要版本執行 Aurora PostgreSQL 資料庫叢集的版本升級。如需詳細資訊，請參閱如何執行主要版本升級。	2022 年 3 月 4 日
Aurora PostgreSQL 支援可設定的密碼套件	您現在可以透過 Aurora PostgreSQL 11.8 版及更高版本，使用可設定的密碼套件來控制資料庫伺服器接受的連線加密。如需使用 Aurora PostgreSQL 搭配可設定密碼套件的詳細資訊，請參閱為 Aurora PostgreSQL 資料庫叢集的連線設定密碼套件。	2022 年 3 月 4 日
AWS JDBC Driver for MySQL 已全面推出	AWS JDBC Driver for MySQL 是專為 Aurora MySQL 高可用性所設計的用戶端驅動程式。 AWS JDBC Driver for MySQL 現已正式推出。如需詳細資訊，請參閱與 Amazon Web Services JDBC Driver for MySQL 連線。	2022 年 3 月 2 日

Aurora PostgreSQL 13.5 支援 Babelfish for Aurora PostgreSQL 1.1.0	Aurora PostgreSQL 13.5 支援 Babelfish 1.1.0。如需新功能的清單，請參閱 13.5。如需每個 Babelfish 版本支援的功能清單，請參閱 Babelfish 各版本支援的功能。如需用量詳細資訊，請參閱使用 Babelfish for Aurora PostgreSQL。	2022 年 2 月 28 日
Amazon Aurora 支援亞太區域 (雅加達) 區域的資料庫活動串流	如需詳細資訊，請參閱支援 AWS 區域 的資料庫活動串流。	2022 年 2 月 16 日
績效詳情支援新的 API 作業	績效詳情現在支援下列 API 作業：`GetResourceMetadata` 、`ListAvailableResourceDimensions` 及 `ListAvailableResourceMetrics` 。如需詳細資訊，請參閱本手冊中的使用績效詳情 API 來擷取指標，及 Amazon RDS 績效詳情 API 參考。	2022 年 1 月 12 日
Amazon RDS Proxy 支援事件	現在您可以在 CloudWatch Events 中訂閱和查看 RDS Proxy 產生的事件，或設定為傳送到 Amazon EventBridge。如需詳細資訊，請參閱使用 RDS Proxy 事件。	2022 年 1 月 11 日

RDS Proxy 可在其他 中使用 AWS 區域	RDS Proxy 現在已在下列區域開放使用：非洲 (開普敦)、亞太區域 (香港)、亞太區域 (大阪)、歐洲 (米蘭)、歐洲 (巴黎)、歐洲 (斯德哥爾摩)、中東 (巴林) 和南美洲 (聖保羅)。如需 RDS Proxy 的詳細資訊，請參閱使用 Amazon RDS Proxy。	2022 年 1 月 5 日
Amazon Aurora 可於亞太區域 (雅加達) 區域使用	Amazon Aurora 現可於亞太區域 (雅加達) 區域使用。如需更多詳細資訊，請參閱區域和可用區域。	2021 年 12 月 13 日
DevOps Guru for Amazon RDS 為 Amazon Aurora 提供詳細的洞察和建議	DevOps Guru for RDS 會探勘績效詳情，以取得效能相關資料。此服務會使用此資料分析 Amazon Aurora 資料庫執行個體的效能，並協助您解決效能問題。若要進一步了解，請參閱本指南中的使用 DevOps Guru for RDS 分析效能異常，以及參閱《Amazon DevOps Guru 使用者指南》中的 DevOps Guru for RDS 概觀。	2021 年 12 月 1 日
Aurora PostgreSQL 支援 RDS Proxy 搭配 PostgreSQL 12	您現在可以使用 Aurora PostgreSQL 12 資料庫叢集建立 RDS Proxy。如需 RDS Proxy 的詳細資訊，請參閱使用 Amazon RDS Proxy。	2021 年 11 月 22 日

Aurora 支援資料庫活動串流的 AWS Graviton2 執行個體類別	您可以針對 Aurora MySQL 和 Aurora PostgreSQL 搭配使用資料庫活動資料流與 db.r6g 執行個體類別。如需更多詳細資訊，請參閱支援的資料庫執行個體類別。	2021 年 11 月 3 日
Amazon Aurora 支援跨帳戶 AWS KMS keys	將資料庫快照匯出至 Amazon S3 時，您可以使用來自不同的 KMS 金鑰 AWS 帳戶 進行加密。如需更多詳細資訊，請參閱將資料庫快照資料匯出至 Amazon S3。	2021 年 11 月 3 日
Amazon Aurora 支援 Babelfish for Aurora PostgreSQL	Babelfish for Aurora PostgreSQL 會擴展您的 Amazon Aurora PostgreSQL 相容版本，使其能夠接受來自 Microsoft SQL Server 用戶端的資料庫連線。如需詳細資訊，請參閱使用 Babelfish for Aurora PostgreSQL。	2021 年 10 月 28 日
Aurora Serverless v1 可能需要 SSL 進行連線	Aurora Serverless v1 現支援 Aurora 叢集參數 `force_ssl` for PostgreSQL 和 `require_secure_transport` for MySQL。如需詳細資訊，請參閱使用 TLS/SSL 搭配 Aurora Serverless v1。	2021 年 10 月 26 日
Amazon Aurora 在其他 中支援績效詳情 AWS 區域	績效詳情已在中東 (巴林)、非洲 (開普敦)、歐洲 (米蘭) 和亞太區域 (大阪) 區域推出。如需詳細資訊，請參閱績效詳情的 AWS 區域 支援。	2021 年 10 月 5 日

Aurora Serverless v1 的可設定自動擴展逾時	您可以選擇 Aurora Serverless v1 等待找到自動擴展點的時間長度。如果在該期間內找不到任何自動擴展點，Aurora Serverless v1 會取消擴展事件或強制容量變更，端視您選取的逾時動作而定。如需詳細資訊，請參閱 Aurora Serverless v1 的自動擴展。	2021 年 9 月 10 日
Aurora 支援 X2g 和 T4g 執行個體類別	Aurora MySQL 和 Aurora PostgreSQL 現在都可以使用 X2g 和 T4g 執行個體類別。您可以使用的執行個體類別取決於 Aurora MySQL 或 Aurora PostgreSQL 的版本。如需受支援之執行個體類型的詳細資訊，請參閱資料庫執行個體類別。	2021 年 9 月 10 日
Amazon RDS 支援共用 VPC 中的 RDS Proxy	您現在可於共用 virtual private cloud (VPC) 中建立 RDS Proxy。如需 RDS Proxy 的詳細資訊，請參閱《Amazon RDS 使用者指南》或《Aurora 使用者指南》中的「使用 Amazon RDS Proxy 管理連線」。	2021 年 8 月 6 日
Aurora 版本政策頁面	此 Amazon Aurora 使用者指南現在包含一個章節，其中描述 Aurora 版本和相關政策的一般資訊。如需詳細資訊，請參閱 Amazon Aurora 版本。	2021 年 7 月 14 日

從 AWS CloudTrail 追蹤中排除資料 API 事件	您可以將資料 API 事件從 CloudTrail 權杖中排除。如需詳細資訊，請參閱從 AWS CloudTrail 追蹤排除資料 API 事件。	2021 年 7 月 2 日
Amazon Aurora PostgreSQL 相容版本支援額外的延伸	最新支援的延伸包括 pg_bigm、pg_cron、pg_partman 和 pg_proctab。如需更多詳細資訊，請參閱 Amazon Aurora PostgreSQL 相容版本的延伸版本。	2021 年 6 月 17 日
複製 Aurora Serverless 叢集	您現在可以建立 Aurora Serverless 的複製叢集。如需有關複製的詳細資訊，請參閱複製 Aurora DB 叢集叢集磁碟區。	2021 年 6 月 16 日
Aurora 全球資料庫可在中國 (北京) 和中國 (寧夏) 區域使用	您現在可以在中國 (北京) 和中國 (寧夏) 區域中建立 Aurora 全域資料庫。如需 Aurora Global Database 的相關資訊，請參閱使用 Amazon Aurora Global Database。	2021 年 5 月 19 日
適用於資料 API 的 FIPS 140-2 支援	資料 API 支援 SSL/TLS 連線的聯邦資訊處理標準發行版 140-2 (FIPS 140-2)。如需更多詳細資訊，請參閱資料 API 可用性。	2021 年 5 月 14 日

AWS 適用於 PostgreSQL 的 JDBC 驅動程式 （ 預覽 ）	AWS JDBC Driver for PostgreSQL 現已提供預覽版，是專為 Aurora PostgreSQL 高可用性而設計的用戶端驅動程式。如需更多詳細資訊，請參閱與 Amazon Web Services JDBC Driver for PostgreSQL 連線 (預覽版)。	2021 年 4 月 27 日
其他 中可用的資料 API AWS 區域	資料 API 現在可在亞太區域 (首爾) 和加拿大 (中部) 區域中使用。如需更多詳細資訊，請參閱資料 API 的可用性。	2021 年 4 月 9 日
Amazon Aurora 支援 Graviton2 資料庫執行個體類別	您現在可以使用 Graviton2 資料庫執行個體類別 db.r6g.x，來建立執行 MySQL 或 PostgreSQL 的資料庫叢集。如需更多詳細資訊，請參閱資料庫執行個體類型。	2021 年 3 月 12 日
RDS 代理端點增強功能	您可以建立與每個 RDS 代理相關聯的其他端點。在不同的 VPC 中建立端點，可啟用代理的跨 VPC 存取。Aurora MySQL 叢集代理也可以擁有唯讀端點。這些讀取器端點會連線到叢集中的讀取者資料庫執行個體，並可改善查詢密集型應用程式的讀取延展性和可用性。如需 RDS Proxy 的詳細資訊，請參閱 Amazon RDS 使用者指南或 Aurora 使用者指南中的「使用 Amazon RDS Proxy 管理連線」。	2021 年 3 月 8 日

Amazon Aurora 可於亞太區域 (大阪) 區域使用	Amazon Aurora 現可於亞太區域 (大阪) 區域使用。如需更多詳細資訊，請參閱區域和可用區域。	2021 年 3 月 1 日
Aurora PostgreSQL 支援在同一個資料庫叢集上同時啟用 IAM 和 Kerberos 身分驗證	Aurora PostgreSQL 現在支援在同一個資料庫叢集上同時啟用 IAM 身分驗證和 Kerberos 身分驗證。如需更多詳細資訊，請參閱 Amazon Aurora 的資料庫身分驗證。	2021 年 2 月 24 日
Aurora 全球資料庫現在支援受管計劃容錯移轉	Aurora 全域資料庫現在支援受管計劃容錯移轉，可讓您更輕鬆地變更 Aurora 全域資料庫的主要 AWS 區域。您只能將受管計劃容錯移轉與運作狀態良好的 Aurora 全域資料庫搭配使若要進一步了解，請參閱災難復原和 Amazon Aurora 全域資料庫。如需參考資訊，請參閱 Amazon RDS API Reference 中的 `FailoverGlobalCluster`。	2021 年 2 月 11 日
適用於 Aurora Serverless 的資料 API 現在支援更多資料類型	透過 Aurora Serverless 的資料 API，您現在可以使用 `UUID` 和 `JSON` 資料類型作為資料庫的輸入。此外，透過 Aurora Serverless 的資料 API，您現在可以從資料庫傳回一個 `LONG` 類型值作為 `STRING` 值。若要進一步了解，請參閱呼叫資料 API。如需有關支援資料類型的參考資訊，請參閱 Amazon RDS 資料服務 API 參考資料中的 `SqlParameter`。	2021 年 2 月 2 日

Aurora PostgreSQL 可支援將主要版本升級至 PostgreSQL 12	透過 Aurora PostgreSQL，您現在可以將資料庫引擎升級至主要版本 12。如需更多詳細資訊，請參閱 升級 Aurora PostgreSQL 的 PostgreSQL 資料庫引擎。	2021 年 1 月 28 日
Aurora MySQL 支援就地升級	您可以將 Aurora MySQL 1.x 叢集升級至 Aurora MySQL 2.x，以保留原始叢集的資料庫執行個體、端點等。憑藉此就地升級技術，可以避免藉由還原快照來設定全新叢集的不便。此外，它還避免了將所有資料表中的資料複製到新叢集的開銷。如需更多詳細資訊，請參閱將 Aurora MySQL 資料庫叢集的主要版本從 1.x 升級至 2.x。	2021 年 1 月 11 日
AWS JDBC Driver for MySQL（預覽版）	AWS JDBC Driver for MySQL 現已提供預覽版，是專為 Aurora MySQL 高可用性而設計的用戶端驅動程式。如需更多詳細資訊，請參閱連線使用 Amazon Web Services JDBC Driver for MySQL (預覽版)。	2021 年 1 月 7 日
Aurora 支援全域資料庫次要叢集上的資料庫活動串流	您可以在 Aurora PostgreSQL 或 Aurora MySQL 的主要或次要叢集上，啟動資料庫活動串流。如需支援的引擎版本，請參閱 Aurora 全域資料庫的限制。	2020 年 12 月 22 日

具有 4 個資料庫執行個體的多主機叢集	Aurora MySQL 多主機叢集中的資料庫執行個體數目上限現在是四個。之前，上限是兩個資料庫執行個體。如需更多詳細資訊，請參閱處理 Aurora 多主機叢集。	2020 年 12 月 17 日
Aurora PostgreSQL 支援 AWS Lambda 函數	您現在可以叫用 Aurora PostgreSQL 資料庫叢集的 AWS Lambda 函數。如需更多詳細資訊，請參閱從 Aurora PostgreSQL 資料庫叢集叫用 Lambda 函數。	2020 年 12 月 11 日
Amazon Aurora 支援預覽版 Graviton2 DB 資料庫執行個體類別	您現在可以在預覽版 Graviton2 資料庫執行個體類別 db.r6g.x，來建立執行 MySQL 或 PostgreSQL 的資料庫叢集。如需更多詳細資訊，請參閱資料庫執行個體類型。	2020 年 12 月 11 日
Amazon Aurora Serverless v2 現在提供預覽版。	Amazon Aurora Serverless v2 提供預覽版。若要使用 Amazon Aurora Serverless v2，必須申請存取權。如需詳細資訊，請參閱 Aurora Serverless v2 頁面。	2020 年 12 月 1 日

Aurora PostgreSQL 現可用於更多 AWS 區域中的 Aurora Serverless。	Aurora PostgreSQL 現可用於更多 AWS 區域中的 Aurora Serverless。您現在可以選擇在提供 AWS 區域 的相同 Aurora PostgreSQL Serverless v1中執行 Aurora MySQL Serverless v1。其他 AWS 區域 Aurora Serverless 支援包括美國西部（加利佛尼亞北部）、亞太區域（新加坡）、亞太區域（雪梨）、亞太區域（首爾）、亞太區域（孟買）加拿大（中部）歐洲（倫敦）和歐洲（巴黎）。如需所有區域清單和適用於 Aurora Serverless 的受支援 Aurora 資料庫引擎清單，請參閱 Aurora Serverless v1 的支援區域和 Aurora 資料庫引擎。適用於 Aurora Serverless 的 Amazon RDS 資料 API 目前於這些相同的 AWS 區域中開放使用。如需支援適用於 Aurora Serverless 的 Data API 的所有區域清單，請參閱具有 Aurora MySQL Serverless v1 的 Data API	2020 年 11 月 24 日

Amazon RDS 績效詳情引進新的維度	您可以根據資料庫、應用程式 (PostgreSQL) 和工作階段類型 (PostgreSQL) 的維度群組對資料庫負載分組。Amazon RDS 還支援維度 db.name、db.application.name (PostgreSQL) 和 db.session_type.name (PostgreSQL)。如需更多詳細資訊，請參閱最高負載資料表。	2020 年 11 月 24 日
Aurora Serverless 支援 Aurora PostgreSQL 10.12 版	在支援適用於 Aurora Serverless 的 Aurora PostgreSQL 的 AWS 區域中，適用於 Aurora Serverless 的 Aurora PostgreSQL 已升級至 Aurora PostgreSQL 10.12 版。如需詳細資訊，請參閱 Aurora Serverless v1 的支援區域和 Aurora 資料庫引擎。	2020 年 11 月 4 日
資料 API 現在支援標籤型授權	資料 API 支援標籤型授權。如果您已使用標籤來標示 RDS 叢集資源，則可以在政策陳述式中使用這些標籤來透過資料 API 控制存取。如需更多詳細資訊，請參閱授權資料 API 的存取。	2020 年 10 月 27 日
Amazon Aurora 將對匯出快照的支援延伸至 Amazon S3	您現在可將資料庫快照資料匯出到所有商業 AWS 區域中的 Amazon S3。如需更多詳細資訊，請參閱將資料庫快照資料匯出至 Amazon S3。	2020 年 10 月 22 日

Aurora 全球資料庫支援複製	您現在可以建立 Aurora 全域資料庫的主要和次要資料庫叢集的複製。您可以使用 AWS 管理主控台 並選擇建立複製功能表選項來執行此操作。您也可以使用 AWS CLI ，並使用 `--restore-type copy-on-write` 選項執行 `restore-db-cluster-to-point-in-time`命令。使用 AWS 管理主控台 或 AWS CLI，您也可以跨帳戶從 Aurora 全域資料庫複製資料庫叢集 AWS 。如需有關複製的詳細資訊，請參閱複製 Aurora 資料庫叢集磁碟區。	2020 年 10 月 19 日
Amazon Aurora 支援動態調整叢集磁碟區的大小	從 Aurora MySQL 1.23 和 2.09，以及 Aurora PostgreSQL 3.3.0 和 Aurora PostgreSQL 2.6.0 開始，Aurora 會在您透過操作 (例如 `DROP TABLE`) 移除資料之後減少叢集磁碟區的大小。若要利用此增強功能，請根據叢集所使用的資料庫引擎，升級至其中一個適當的版本。如需此功能以及如何檢查 Aurora 叢集的已使用和可用儲存空間的相關資訊，請參閱管理 Aurora 資料庫叢集的效能和擴展。	2020 年 10 月 13 日

Amazon Aurora 支援的磁碟區大小最多至 128 TiB	新的和現有的 Aurora 叢集磁碟區現在最大可成長到 128 tebibytes (TiB) 的大小。如需更多詳細資訊，請參閱 Aurora 儲存體的成長方式。	2020 年 9 月 22 日
在中國 (寧夏) 區域，Aurora PostgreSQL 支援 db.r5 和 db.t3 資料庫執行個體類別	您現在可以在中國 (寧夏) 區域建立使用 db.r5 和 db.t3 資料庫執行個體類別的 Aurora PostgreSQL 資料庫叢集。如需更多詳細資訊，請參閱資料庫執行個體類別。	2020 年 9 月 3 日

Aurora 平行查詢增強功能	從 Aurora MySQL 2.09 和 1.23 開始，您可以利用平行查詢功能的增強功能。建立平行查詢叢集不再需要特殊的引擎模式。您現在可以使用任何執行相容 Aurora MySQL 版本的佈建叢集的 `aurora_parallel_query` 組態選項來開啟和關閉平行查詢。您可以將現有叢集升級為相容 Aurora MySQL 版本並使用平行查詢，而不是建立新叢集並將資料匯入其中。您可以針對平行查詢叢集使用績效詳情。您可以停止並啟動平行查詢叢集。您可以建立與 MySQL 5.7 相容的 Aurora 平行查詢叢集。平行查詢適用於使用 `DYNAMIC` 資料列格式的資料表。平行查詢叢集可以使用 AWS Identity and Access Management (IAM) 身分驗證。平行查詢叢集中的讀取器資料庫執行個體可以利用 `READ COMMITTED` 隔離等級。您現在也可以在其他 AWS 區域中建立平行查詢叢集。如需平行查詢功能和這些增強功能的詳細資訊，請參閱 Aurora MySQL 的平行查詢。	2020 年 9 月 2 日
Aurora MySQL 參數 binlog_rows_query_log_events 的變更	您現在可以變更 Aurora MySQL 組態參數 `binlog_rows_query_log_events` 的值。以前，這個參數是不可修改的。	2020 年 8 月 26 日

支援 Aurora MySQL 的自動次要版本升級	使用 Aurora MySQL，現在當您為 Aurora MySQL 資料庫叢集指定 Enable auto minor version upgrade (啟用自動次要版本升級) 時，此設定將會生效。當您啟用自動次要版本升級時，Aurora 會在新次要版本發行時自動升級。自動升級會在資料庫的維護期間進行。對於 Aurora MySQL，此功能僅適用於與 MySQL 5.7 相容的 Aurora MySQL 版本 2。一開始，自動升級程序會使 Aurora MySQL 資料庫叢集成為 2.07.2 版。如需有關此功能如何在 Aurora MySQL 運作的詳細資訊，請參閱 Amazon Aurora MySQL 的資料庫升級和修補程式。	2020 年 8 月 3 日
Aurora PostgreSQL 可支援將主要版本升級至 PostgreSQL 第 11 版	透過 Aurora PostgreSQL，您現在可以將資料庫引擎升級至主要版本 11。如需更多詳細資訊，請參閱 升級 Aurora PostgreSQL 的 PostgreSQL 資料庫引擎。	2020 年 7 月 28 日
Amazon Aurora 支援 AWS PrivateLink	Amazon Aurora 現在支援為 Amazon RDS API 呼叫建立 Amazon VPC 端點，以將應用程式和 Aurora 之間的流量保留在 AWS 網路中。如需更多詳細資訊，請參閱 Amazon Aurora 和界面 VPC 端點 (AWS PrivateLink)。	2020 年 7 月 9 日

RDS Proxy 已正式上市	RDS Proxy 現已正式上市。您可以將 RDS Proxy 與用於 MySQL 的 RDS、Aurora MySQL、用於 PostgreSQL 的 RDS 和用於生產工作負載的 Aurora PostgreSQL 搭配使用。如需 RDS Proxy 的詳細資訊，請參閱 Amazon RDS 使用者指南或 Aurora 使用者指南中的「使用 Amazon RDS Proxy 管理連線」。	2020 年 6 月 30 日
Aurora 全球資料庫寫入轉送	您現在可以在全域資料庫中啟用次要叢集的寫入功能。使用寫入轉送時，您可以在次要叢集上發出 DML 陳述式，Aurora 會將寫入轉送到主要叢集，並將更新的資料複寫到所有次要叢集。如需詳細資訊，請參閱AWS 區域 使用 Aurora 全域資料庫寫入次要 的轉送。	2020 年 6 月 18 日
Aurora 支援與 整合 AWS Backup	您可以使用 AWS Backup 來管理 Aurora 資料庫叢集的備份。如需更多詳細資訊，請參閱備份與還原 Aurora 資料庫叢集概觀。	2020 年 6 月 10 日
Aurora PostgreSQL 支援 db.t3.large 資料庫執行個體類別	您現在可以建立使用 db.t3.large 資料庫執行個體類別的 Aurora PostgreSQL 資料庫叢集。如需更多詳細資訊，請參閱資料庫執行個體類別。	2020 年 6 月 5 日

Aurora Global Database 支援 PostgreSQL 11.7 版和受管的復原點目標 (RPO)	您現在可以為 PostgreSQL 資料庫引擎 11.7 版建立 Aurora 全域資料庫。您也可以使用復原點目標 (RPO) 來管理 PostgreSQL 全域資料庫如何從失敗中復原。如需更多詳細資訊，請參閱 Aurora Global Database 的跨區域災難復原。	2020 年 6 月 4 日
Aurora MySQL 支援使用資料庫活動串流監控資料庫	Aurora MySQL 現在包含資料庫活動串流，可提供關聯式資料庫中資料庫活動近乎即時的資料串流。如需更多詳細資訊，請參閱使用資料庫活動串流。	2020 年 6 月 2 日
其他 中可用的查詢編輯器 AWS 區域	Aurora Serverless 的查詢編輯器現在可在其他 中使用 AWS 區域。如需更多詳細資訊，請參閱查詢編輯器的可用性。	2020 年 5 月 28 日
其他 中可用的資料 API AWS 區域	資料 API 現在可在其他 中使用 AWS 區域。如需更多詳細資訊，請參閱資料 API 的可用性。	2020 年 5 月 28 日
RDS 代理伺服器可在加拿大 (中部) 區域使用	您現在可以在加拿大 (中部) 區域使用 RDS 代理伺服器預覽。如需 RDS Proxy 的詳細資訊，請參閱使用 Amazon RDS Proxy 管理連線 (預覽版)。	2020 年 5 月 28 日

Aurora 全球資料庫和跨區域僅供讀取複本	對於 Aurora Global Database，您無法在次要叢集所在的同一區域從主要叢集建立 Aurora MySQL 跨區域的讀取複本。如需 Aurora Global Database 和跨區域僅供讀取複本的詳細資訊，請參閱使用 Amazon Aurora Global Database和複寫 Amazon Aurora MySQL 資料庫。	2020 年 5 月 18 日
RDS Proxy 可在更多 中使用 AWS 區域	您現在可以在美國西部 (加利佛尼亞北部) 區域、歐洲 (倫敦) 區域、歐洲 (法蘭克福) 區域、亞太區域 (首爾)、亞太區域 (孟買) 區域、亞太區域 (新加坡) 區域和亞太區域 (雪梨)中使用 RDS Proxy 預覽版。如需 RDS Proxy 的詳細資訊，請參閱使用 Amazon RDS Proxy 管理連線 (預覽版)。	2020 年 5 月 13 日
Aurora PostgreSQL 相容版本支援內部部署或自我託管的 Microsoft Active Directory	您現在可以使用內部部署或自我託管的 Active Directory，在使用者連線至 Aurora PostgreSQL 資料庫叢集時進行 Kerberos 身分驗證。如需更多詳細資訊，請參閱搭配 Aurora PostgreSQL 使用 Kerberos 身分驗證。	2020 年 5 月 7 日

Aurora MySQL 多主機叢集提供更多 AWS 區域	您現在可以在亞太區域 (首爾)、亞太區域 (東京)、亞太區域 (孟買) 區域 和歐洲 (法蘭克福) 區域中建立 Aurora 多主機叢集。如需多主機叢集的詳細資訊，請參閱使用 Aurora 多主機叢集。	2020 年 5 月 7 日
績效詳情支援分析執行中 Aurora MySQL 查詢的統計資訊	您現在可以使用績效詳情，對 Aurora MySQL 資料庫執行個體分析執行中查詢的統計資訊。如需更多詳細資訊，請參閱分析執行中查詢的統計資料。	2020 年 5 月 5 日
適用於資料 API 的 Java 用戶端程式庫已全面推出	適用於資料 API 的 Java 用戶端程式庫現已全面推出。您可以下載並使用適用於資料 API 的 Java 用戶端程式庫。它可讓您將用戶端類別映射至資料 API 的要求和回應。如需更多詳細資訊，請參閱使用適用於資料 API 的 Java 用戶端程式庫。	2020 年 4 月 30 日
Amazon Aurora 可於歐洲 (米蘭) 區域使用	Amazon Aurora 現可於歐洲 (米蘭) 區域使用。如需更多詳細資訊，請參閱區域和可用區域。	2020 年 4 月 28 日
Amazon Aurora 可於歐洲 (米蘭) 區域使用	Amazon Aurora 現可於歐洲 (米蘭) 區域使用。如需更多詳細資訊，請參閱區域和可用區域。	2020 年 4 月 27 日

Amazon Aurora 可於非洲 (開普敦) 區域使用	Amazon Aurora 現可於非洲 (開普敦) 區域使用。如需更多詳細資訊，請參閱區域和可用區域。	2020 年 4 月 22 日
Aurora PostgreSQL 現在支援 db.r5.16xlarge 和 db.r5.8xlarge 資料庫執行個體類別	您現在可以建立執行 PostgreSQL 的 Aurora PostgreSQL 資料庫叢集，這些叢集使用 db.r5.16xlarge 和 db.r5.8xlarge 的資料庫執行個體類別。如需更多詳細資訊，請參閱Aurora 的資料庫執行個體類別的硬體規格。	2020 年 4 月 8 日
適用於 PostgreSQL 的 Amazon RDS Proxy	Amazon RDS Proxy 現在可用於 PostgreSQL。您可以使用 RDS Proxy 來減少叢集上的連線管理額外負荷，也可以減少發生「連線過多」錯誤的機會。RDS Proxy 目前位於 PostgreSQL 的公開預覽中。如需詳細資訊，請參閱使用 Amazon RDS Proxy 管理連線 (預覽版)。	2020 年 4 月 8 日
Aurora 全球資料庫現在支援 Aurora PostgreSQL	您現在可以為 PostgreSQL 資料庫引擎建立 Aurora 全球資料庫。Aurora 全域資料庫跨越多個 AWS 區域，可實現全區域中斷的低延遲全域讀取和災難復原。如需更多詳細資訊，請參閱使用 Amazon Aurora Global Database。	2020 年 3 月 10 日

支援 Aurora PostgreSQL 的主要版本升級	藉著 Aurora PostgreSQL，您可以將資料庫引擎升級為主要版本。如此一來，您就可以在升級選取的 PostgreSQL 引擎版本時，跳至較新的主要版本。如需更多詳細資訊，請參閱 升級 Aurora PostgreSQL 的 PostgreSQL 資料庫引擎。	2020 年 3 月 4 日
Aurora PostgreSQL 支援 Kerberos 身分驗證	您現在可以使用 Kerberos 身分驗證來在使用者連線到您的 Aurora PostgreSQL 資料庫叢集時對其進行身分驗證。如需更多詳細資訊，請參閱搭配 Aurora PostgreSQL 使用 Kerberos 身分驗證。	2020 年 2 月 28 日
資料 API 支援 AWS PrivateLink	資料 API 現在支援為資料 API 呼叫建立 Amazon VPC 端點，以保留 AWS 網路中應用程式和資料 API 之間的流量。如需更多詳細資訊，請參閱為資料 API 建立 Amazon VPC 端點 (AWS PrivateLink)。	2020 年 2 月 6 日
Aurora PostgreSQL 中的 Aurora 機器學習支援	此 `aws_ml` Aurora PostgreSQL 延伸模組提供您在資料庫查詢中使用的函數，以呼叫 Amazon Comprehend 進行情緒分析，以及呼叫 SageMaker AI 執行您自己的機器學習模型。如需更多詳細資訊，請參閱使用機器學習 (ML) 功能搭配 Aurora。	2020 年 2 月 5 日

Aurora PostgreSQL 支援將資料匯出至 Amazon S3	您可以從 Aurora PostgreSQL 資料庫叢集查詢資料，然後將資料直接匯出至 Amazon S3 儲存貯體中存放的檔案。如需更多詳細資訊，請參閱從 Aurora PostgreSQL 資料庫叢集將資料匯出至 Amazon S3。	2020 年 2 月 5 日
支援將資料庫快照資料匯出至 Amazon S3	Amazon Aurora 支援將資料庫快照資料匯出至適用於 MySQL 和 PostgreSQL 的 Amazon S3。如需更多詳細資訊，請參閱將資料庫快照資料匯出至 Amazon S3。	2020 年 1 月 9 日
文件歷程記錄中的 Aurora MySQL 版本備註	本節現在包含 2018 年 8 月 31 日後發佈版本的 Aurora MySQL 相容版本 版本備註歷程記錄。如需特定版本的完整版本備註，請選擇歷程記錄項目第一欄中的連結。	2019 年 12 月 13 日
Amazon RDS Proxy	您可以使用 Amazon RDS Proxy，減少叢集上連線管理的負擔，並減少發生「連線過多」錯誤的機會。您可以將每個代理與 RDS 資料庫執行個體或 Aurora 資料庫叢集建立關聯。然後，您可以在應用程式的連線字串中使用代理端點。Amazon RDS Proxy 目前處於公開預覽狀態。它支援 Aurora MySQL 資料庫引擎。如需更多詳細資訊，請參閱使用 Amazon RDS Proxy 管理連線 (預覽版)。	2019 年 12 月 3 日

適用於 Aurora Serverless v1 的資料 API 支援資料類型映射提示	您現在可以使用提示，指示適用於 Aurora Serverless v1 的資料 API 將 `String` 值以不同類型傳送至資料庫。如需更多詳細資訊，請參閱呼叫資料 API。	2019 年 11 月 26 日
適用於 Aurora Serverless v1 的資料 API 支援 Java 用戶端程式庫 (預覽版)	您可以下載並使用適用於資料 API 的 Java 用戶端程式庫。它可讓您將用戶端類別映射至資料 API 的要求和回應。如需更多詳細資訊，請參閱使用適用於資料 API 的 Java 用戶端程式庫。	2019 年 11 月 26 日
Aurora PostgreSQL 符合 FedRAMP HIGH	Aurora PostgreSQL 符合 FedRAMP HIGH。如需 AWS 和合規工作的詳細資訊，請參閱AWS 合規計劃範圍內的服務。	2019 年 11 月 26 日

為 Amazon Aurora MySQL 複本啟用 READ COMMITTED 隔離層級	您現在可以在 Aurora MySQL 複本上啟用 `READ COMMITTED` 隔離層級。若要這樣做，您必須在工作階段層級啟用 `aurora_read_replica_read_committed_isolation_enabled` 組態設定。對於 OLTP 叢集上長時間執行的查詢，使用 `READ COMMITTED` 隔離層級有助於解決歷史記錄清單過長的問題。啟用此設定之前，務必了解 Aurora 複本的隔離行為與 `READ COMMITTED` 的平常 MySQL 實作有何不同。如需更多詳細資訊，請參閱 Aurora MySQL 隔離層級。	2019 年 11 月 25 日
「績效詳情」支援分析執行 Aurora PostgreSQL 查詢的統計資訊	您現在可以使用績效詳情，分析對 Aurora PostgreSQL 資料庫執行個體執行查詢的統計資訊。如需更多詳細資訊，請參閱分析執行中查詢的統計資料。	2019 年 11 月 25 日
Aurora 全球資料庫中有更多叢集	您現在可以將多個次要區域新增至 Aurora 全球資料庫。您可以跨更廣的地理區域使用低延遲全球讀取和災難復原。如需 Aurora Global Database 的相關資訊，請參閱使用 Amazon Aurora Global Database。	2019 年 11 月 25 日

Aurora MySQL 中的 Aurora 機器學習支援	在 Aurora MySQL 2.07 和更高版本中，您可以呼叫 Amazon Comprehend 進行情緒分析，呼叫 SageMaker AI 進行多種機器學習演算法。您將預存函數的呼叫嵌入查詢中，以直接在資料庫應用程式中使用結果。如需更多詳細資訊，請參閱使用機器學習 (ML) 功能搭配 Aurora。	2019 年 11 月 25 日
Aurora 全球資料庫不再需要引擎模式設定	在建立預定加入 Aurora 全球資料庫中的叢集時，您不再需要指定 `--engine-mode=global` 。所有符合相容性需求的 Aurora 叢集都有資格成為全球資料庫的一部分。例如，叢集目前必須使用與 MySQL 5.6 相容的 Aurora MySQL 第 1 版。如需 Aurora Global Database 的相關資訊，請參閱使用 Amazon Aurora Global Database。	2019 年 11 月 25 日
Aurora 全球資料庫適用於 Aurora MySQL 第 2 版	從 Aurora MySQL 2.07 開始，您可以建立與 MySQL 5.7 相容的 Aurora 全球資料庫。您不需要為主要或次要叢集指定 `global` 引擎模式。您可以將任何新佈建的叢集，連同 Aurora MySQL 2.07 或更高版本，一起新增至 Aurora 全球資料庫。如需 Aurora Global Database 的相關資訊，請參閱使用 Amazon Aurora Global Database。	2019 年 11 月 25 日

支援 Aurora MySQL 熱門資料列爭用最佳化，不需要實驗室模式	熱門資料列爭用最佳化現在全面開放給 Aurora MySQL，不需要將 Aurora 實驗室模式設為 ON。此功能可以在相同頁面上有許多交易爭用資料列的情況下大幅改善工作負載的傳輸量。此改善涉及變更 Aurora MySQL 所使用的鎖定釋放演算法。	2019 年 11 月 19 日
支援 Aurora MySQL 雜湊聯結，不需要實驗室模式	雜湊聯結功能現在全面開放給 Aurora MySQL，不需要將 Aurora 實驗室模式設為 ON。當您需要透過使用對等聯結來聯結大量資料時，此功能可改善查詢效能。如需使用此功能的詳細資訊，請參閱在 Aurora MySQL 中使用雜湊聯結。	2019 年 11 月 19 日
Aurora MySQL 2.* 支援更多 db.r5 執行個體類別	Aurora MySQL 叢集現在支援執行個體類型 db.r5.8xlarge、db.r5.16xlarge 和 db.r5.24xlarge。如需 Aurora MySQL 叢集的執行個體類型的詳細資訊，請參閱選擇資料庫執行個體類別。	2019 年 11 月 19 日
Aurora MySQL 2.* 支援恢復	Aurora MySQL 2.* 現在可從使用者的錯誤操作中迅速復原，像是捨棄不該捨棄的資料表或刪除不該刪除的列。恢復可讓您將資料庫回溯至先前的時間點，無需從備份還原，在數秒內即可完成，即使是大型資料庫也一樣。如需詳細資訊，請參閱回溯 Aurora 資料庫叢集。	2019 年 11 月 19 日

帳單標籤支援 Aurora	現在您可以使用標籤追蹤各項資源的成本分配，例如 Aurora 叢集、Aurora 叢集中的資料庫執行個體、輸入/輸出、備份、快照等。您可以使用 AWS Cost Explorer 查看與每個標籤相關的成本。如需搭配 Aurora 使用標籤的詳細資訊，請參閱標記 Amazon RDS 資源。有關標籤的一般資訊以及使用標籤進行成本分析的方法，請參閱使用成本分配標籤和使用者定義成本分配標籤。	2019 年 10 月 23 日
適用於 Aurora PostgreSQL 的資料 API	Aurora PostgreSQL 現在支援對 Amazon Aurora Serverless v1 資料庫叢集使用資料 API。如需更多詳細資訊，請參閱使用適用於 Aurora Serverless v1 的資料 API。	2019 年 9 月 23 日
Aurora PostgreSQL 支援將資料庫日誌至上傳至 CloudWatch logs	您可以設定 Aurora PostgreSQL 資料庫叢集，以將日誌資料發佈至 Amazon CloudWatch Logs 中的日誌群組。使用 CloudWatch Logs，您可以執行日誌資料的即時分析，並使用 CloudWatch 來建立警示和檢視指標。您可以使用 CloudWatch Logs 將日誌記錄存放在高耐用性的儲存裝置中。如需更多詳細資訊，請參閱將 Aurora PostgreSQL 日誌發佈至 Amazon CloudWatch Logs。	2019 年 8 月 9 日

Aurora MySQL 多主機叢集	您可以設定 Aurora MySQL 多主機叢集。在這些叢集中，每個資料庫執行個體都有讀寫能力。如需更多詳細資訊，請參閱處理 Aurora 多主機叢集。	2019 年 8 月 8 日
Aurora PostgreSQL 支援 Aurora Serverless v1	您現在可以使用 Amazon Aurora Serverless v1 搭配 Aurora PostgreSQL。Aurora Serverless 資料庫叢集可根據應用程式的需要而自動啟動、關閉和擴展或縮減運算容量。如需更多詳細資訊，請參閱使用 Amazon Aurora Serverless v1。	2019 年 7 月 9 日
Aurora MySQL 的跨帳戶複製	您現在可以在 AWS 帳戶之間複製 Aurora MySQL 資料庫叢集的叢集磁碟區。您可以透過 AWS Resource Access Manager (AWS RAM) 授權共用。經過複製的叢集磁碟區會使用寫入時複製機制，只需要針對新的或變更的資料具備額外儲存體。如需 Aurora 複製的詳細資訊，請參閱複製 Aurora 資料庫叢集中的資料庫。	2019 年 7 月 2 日
Aurora PostgreSQL 支援 db.t3 資料庫執行個體類別	您現在可以建立使用 db.t3 資料庫執行個體類別的 Aurora PostgreSQL 資料庫叢集。如需更多詳細資訊，請參閱資料庫執行個體類別。	2019 年 6 月 20 日

支援針對 Aurora PostgreSQL 匯入 Amazon S3 中的資料	您現在可以將資料從 Amazon S3 檔案匯入 Aurora PostgreSQL 資料庫叢集中的資料表。如需更多詳細資訊，請參閱將 Amazon S3 資料匯入 Aurora PostgreSQL 資料庫叢集。	2019 年 6 月 19 日
Aurora PostgreSQL 現在可透過叢集快取管理來提供快速容錯移轉復原	Aurora PostgreSQL 現提供叢集快取管理，確保主要資料庫執行個體發生容錯移轉時快速恢復。如需更多詳細資訊，請參閱透過叢集快取管理在容錯移轉後快速復原。	2019 年 6 月 11 日
適用於 Aurora Serverless v1 的資料 API 已全面推出	您可以從以 Web 服務為基礎的應用程式使用資料 API 存取 Aurora Serverless v1 叢集。如需更多詳細資訊，請參閱使用適用於 Aurora Serverless v1 的資料 API。	2019 年 5 月 30 日
Aurora PostgreSQL 支援使用資料庫活動串流監控資料庫	Aurora PostgreSQL 現在包含資料庫活動串流，可提供關聯式資料庫中資料庫活動近乎即時的資料串流。如需更多詳細資訊，請參閱使用資料庫活動串流。	2019 年 5 月 30 日
Amazon Aurora 建議	Amazon Aurora 現提供自動化的 Aurora 資源建議。如需更多詳細資訊，請參閱使用 Amazon Aurora 建議。	2019 年 5 月 22 日

績效詳情支援 Aurora 全球資料庫	Aurora 全球資料庫現可使用績效詳情。如需 Aurora 績效詳情的相關資訊，請參閱使用 Amazon RDS 績效詳情。如需 Aurora Global Database 的相關資訊，請參閱使用 Aurora Global Database。	2019 年 5 月 13 日
Aurora MySQL 5.7 現可使用績效詳情功能	Amazon RDS 績效詳情現可供 Aurora MySQL 2.x 版使用，這些版本與 MySQL 5.7 相容。如需更多詳細資訊，請參閱使用 Amazon RDS 績效詳情。	2019 年 5 月 3 日
Aurora 全域資料庫可在更多 中使用 AWS 區域	您現在可以在大部分可使用 Aurora AWS 區域 的地方建立 Aurora 全域資料庫。如需 Aurora Global Database 的相關資訊，請參閱使用 Amazon Aurora Global Database。	2019 年 4 月 30 日
Aurora Serverless v1 的容量下限為 1	Aurora Serverless v1 叢集可用的容量下限設定為 1。之前，此下限為 2。如需指定 Aurora Serverless 容量值的相關資訊，請參閱設定 Aurora Serverless v1 資料庫叢集的容量。	2019 年 4 月 29 日
Aurora Serverless v1 逾時動作	您現在可以指定 Aurora Serverless v1 容量變更逾時情況下採取的動作。如需更多詳細資訊，請參閱容量變更的逾時動作。	2019 年 4 月 29 日

以秒計費	Amazon RDS 現在以 1 秒遞增計費，隨需執行個體的所有 Amazon RDS 除 AWS GovCloud (US) AWS 區域 除外。如需更多詳細資訊，請參閱 Aurora 資料庫執行個體計費。	2019 年 4 月 25 日
跨 共用Aurora Serverless v1快照 AWS 區域	對於 Aurora Serverless v1，快照一律加密。如果您使用自己的快照加密 AWS KMS key，您現在可以複製或共用快照 AWS 區域。如需 Aurora Serverless v1 資料庫叢集快照的相關資訊，請參閱 Aurora Serverless v1 及快照。	2019 年 4 月 17 日
從 Amazon S3 還原 MySQL 5.7 備份	現在，您能夠建立 MySQL 5.7 版資料庫的備份，並將其存放在 Amazon S3，接著將備份檔案還原至新的 Aurora MySQL 資料庫叢集。如需更多詳細資訊，請參閱從外部 MySQL 資料庫將資料遷移至 Aurora MySQL 資料庫叢集。	2019 年 4 月 17 日
跨區域共用 Aurora Serverless v1 快照	對於 Aurora Serverless v1，快照一律加密。如果您使用自己的快照加密 AWS KMS key，您現在可以跨區域複製或共用快照。如需 Aurora Serverless v1 資料庫叢集快照的相關資訊，請參閱 Aurora Serverless 和快照。	2019 年 4 月 16 日

Aurora 概念驗證教學課程	您可學習如何執行概念驗證，以搭配 Aurora 試用您的應用程式與工作負載。如需完整教學，請參閱執行 Aurora 概念驗證。	2019 年 4 月 16 日
Aurora Serverless v1 支援從 Amazon S3 備份進行還原	您現在可以從 Amazon S3 將備份匯入至 Aurora Serverless 叢集。如需程序的詳細資訊，請參閱使用 Amazon S3 儲存貯體從 MySQL 遷移資料。	2019 年 4 月 16 日

Aurora Serverless v1 有可修改的新參數	您現在可以對 Aurora Serverless v1 叢集修改下列資料庫參數：innodb_file_format 、innodb_file_per_table 、innodb_large_prefix 、innodb_lock_wait_timeout 、innodb_monitor_disable 、innodb_monitor_enable 、innodb_monitor_reset 、innodb_monitor_reset_all 、innodb_print_all_deadlocks 、log_warnings 、net_read_timeout 、net_retry_count 、net_write_timeout 、sql_mode 及 tx_isolation 。如需 Aurora Serverless v1 叢集組態參數的詳細資訊，請參閱 Aurora Serverless v1 及參數群組。	2019 年 4 月 4 日
Aurora PostgreSQL 支援 db.r5 資料庫執行個體類別	您建立的 Aurora PostgreSQL 資料庫叢集，現可使用 db.r5 資料庫執行個體類別。如需更多詳細資訊，請參閱資料庫執行個體類別。	2019 年 4 月 4 日

Aurora PostgreSQL 邏輯複寫	您現可使用 PostgreSQL 邏輯複寫，複寫 Aurora PostgreSQL 資料庫叢集資料庫的一部分。如需更多詳細資訊，請參閱使用 PostgreSQL 邏輯複寫。	2019 年 3 月 28 日
Aurora MySQL 2.04 的 GTID 支援	複寫現可搭配 MySQL 5.7 的全域交易 ID (GTID) 功能使用。此功能可簡化 Aurora MySQL 與外部相容於 MySQL 5.7 之資料庫之間的二進位日誌 (binlog) 複寫作業。此複寫作業可將 Aurora MySQL 叢集當做來源或目的地。此功能於 Aurora MySQL 2.04 或更新版本提供。如需 GTID 式複寫及 Aurora MySQL 的詳細資訊，請參閱對 Aurora MySQL 使用 GTID 式複寫。	2019 年 3 月 25 日
將 Aurora Serverless v1 日誌上傳到 Amazon CloudWatch	您現在可以讓 Aurora 將資料庫日誌上傳至 Aurora Serverless v1 叢集的 CloudWatch。如需更多詳細資訊，請參閱檢視 Aurora Serverless 資料庫叢集。在此次增強中，您現可在資料庫叢集參數群組內定義執行個體層級參數的值，而且這些值會套用至叢集內的所有資料庫執行個體，除非您在資料庫參數群組內將之覆寫。如需更多詳細資訊，請參閱使用資料庫參數群組和資料庫叢集參數群組。	2019 年 2 月 25 日

Aurora MySQL 支援 db.t3 資料庫執行個體類別	您現在可以建立使用 db.t3 資料庫執行個體類別的 Aurora MySQL 資料庫叢集。如需更多詳細資訊，請參閱資料庫執行個體類別。	2019 年 2 月 25 日
Aurora MySQL 支援 db.r5 資料庫執行個體類別	您建立的 Aurora MySQL 資料庫叢集，現可使用 db.r5 資料庫執行個體類別。如需更多詳細資訊，請參閱資料庫執行個體類別。	2019 年 2 月 25 日
Aurora MySQL 的績效詳情計數器	您現在可以將效能計數器新增至 Aurora MySQL 資料庫執行個體的績效詳情圖表。如需更多詳細資訊，請參閱績效詳情儀表板元件。	2019 年 2 月 19 日
Amazon RDS 績效詳情支援 Aurora MySQL 更多的 SQL 文字檢視	Amazon RDS 績效詳情現可在績效詳情儀表板中，檢視 Aurora MySQL 資料庫執行個體更多的 SQL 文字。如需更多詳細資訊，請參閱在績效詳情儀表板內檢視更多 SQL 文字。	2019 年 2 月 6 日
Amazon RDS 績效詳情支援 Aurora PostgreSQL 更多的 SQL 文字檢視	Amazon RDS 績效詳情現可在績效詳情儀表板中，檢視 Aurora PostgreSQL 資料庫執行個體更多的 SQL 文字。如需更多詳細資訊，請參閱在績效詳情儀表板內檢視更多 SQL 文字。	2019 年 1 月 24 日

AAurora 備份計費	您可以使用 Amazon CloudWatch 指標 `TotalBackupStorageBilled` 、`SnapshotStorageUsed` 和 `BackupRetentionPeriodStorageUsed` ，來監控 Aurora 備份的空間使用量。如需如何使用 CloudWatch 指標的詳細資訊，請參閱監控概觀。如需如何使用管理備份資料儲存體的詳細資訊，請參閱了解 Aurora 備份儲存用量。	2019 年 1 月 3 日
績效詳情計數器	您現在可以將效能計數器新增至您的績效詳情圖表。如需更多詳細資訊，請參閱績效詳情儀表板元件。	2018 年 12 月 6 日
Aurora 全球資料庫	您現在可以建立 Aurora 全球資料庫。Aurora 全域資料庫跨越多個 AWS 區域，可實現全區域中斷的低延遲全域讀取和災難復原。如需更多詳細資訊，請參閱使用 Amazon Aurora Global Database。	2018 年 11 月 28 日
Aurora PostgreSQL 中的查詢計畫管理	Aurora PostgreSQL 現在提供查詢計劃管理，供您用來管理 PostgreSQL 查詢執行計劃。如需更多詳細資訊，請參閱管理 Aurora PostgreSQL 的查詢執行計劃。	2018 年 11 月 20 日

Aurora Serverless v1 的查詢編輯器 (Beta 版)	您可以在 Amazon RDS 主控台對 Aurora Serverless v1 叢集執行 SQL 陳述式。如需更多詳細資訊，請參閱使用 Aurora Serverless v1 的查詢編輯器。	2018 年 11 月 20 日
適用於 Aurora Serverless v1 的資料 API (beta 版)	您可以從以 Web 服務為基礎的應用程式使用資料 API 存取 Aurora Serverless v1 叢集。如需更多詳細資訊，請參閱使用 Aurora Serverless 的資料 API。	2018 年 11 月 20 日
Aurora Serverless v1 的 TLS 支援	Aurora Serverless v1 叢集支援 TLS/SSL 加密。如需更多詳細資訊，請參閱 Aurora Serverless 的 TLS/SSL。	2018 年 11 月 19 日
自訂端點	您現在可以建立與任意資料庫執行個體集合相關聯的端點。此功能對 Aurora 叢集 (其中的部分資料庫執行個體與其他執行個體有不同的容量或組態) 的負載平衡和高可用性有幫助。您可以使用自訂端點，而不需透過其執行個體端點連線至特定資料庫執行個體。如需詳細資訊，請參閱 Amazon Aurora 端點。	2018 年 11 月 12 日
Aurora PostgreSQL 中的 IAM 身分驗證支援	Aurora PostgreSQL 現在支援 IAM 身分驗證。如需詳細資訊，請參閱 IAM 資料庫身分驗證。	2018 年 11 月 8 日

用於還原和時間點復原的自訂參數群組	當還原快照或執行時間點復原操作時，您現在可以指定自訂參數群組。如需更多詳細資訊，請參閱從資料庫叢集快照還原和將資料庫叢集還原至指定的時間。	2018 年 10 月 15 日
Aurora 資料庫叢集的刪除保護	當您針對資料庫叢集啟用刪除保護時，任何使用者皆無法刪除資料庫。如需更多詳細資訊，請參閱刪除資料庫叢集。	2018 年 9 月 26 日
停止/啟動 Aurora 功能	您現在可以採用單一操作來停用或啟動整個 Aurora 叢集。如需更多詳細資訊，請參閱停用和啟動 Aurora 叢集。	2018 年 9 月 24 日
Aurora MySQL 的平行查詢功能	Aurora MySQL 現在提供一個選項，可跨 Aurora 儲存基礎設施平行化查詢的輸入/輸出工作。此功能可加速資料密集分析查詢，這通常是工作負載中最耗費時間的操作。如需詳細資訊，請參閱 Aurora MySQL 的平行查詢。	2018 年 9 月 20 日
新的指南	此為 Amazon Aurora 使用者指南的第一版。	2018 年 8 月 31 日

AWS 詞彙表

如需最新的 AWS 術語，請參閱 AWS 詞彙表 參考中的AWS 詞彙表。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。