aws

開發人員指南

Amazon DocumentDB



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

Amazon DocumentDB: 開發人員指南

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

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

Table of Contents

什麼是 Amazon DocumentDB 1
概觀 1
叢集 3
執行個體 3
區域和 AZ 6
大區 (Regions) 6
可用區域 7
定價 9
免費試用 10
監控 10
介面 10
AWS 管理主控台 10
AWS CLI 10
MongoDB 驅動程式 11
後續步驟？ 11
運作方式 11
Amazon DocumentDB 版本 12
Amazon DocumentDB 端點 13
TLS 支援 16
Amazon DocumentDB 儲存體 16
Amazon DocumentDB 複寫 17
Amazon DocumentDB 可靠性 17
讀取偏好設定選項 18
TTL 刪除 22
計費資源 22
什麼是文件資料庫？ 25
使用案例 26
了解文件 26
使用文件 32
入門指南 43
先決條件 43
步驟 1：建立叢集 45
步驟 2：連線至您的叢集 47
步驟 3：插入和查詢資料 48

步驟 4：探索 50
快速開始使用 CloudFormation 51
先決條件 51
啟動 Amazon DocumentDB CloudFormation 堆疊 52
存取 Amazon DocumentDB 叢集 56
終止保護和刪除保護 57
遷移和升級 58
快速入門指南 58
準備 DMS 來源 58
設定 DMS 59
啟用 DocumentDB 壓縮 59
建立複寫任務 59
監控進度 60
其他資訊 60
遷移 Runbook 60
相容性 61
工作負載探索 63
建立新的目標叢集 66
索引遷移 66
使用者遷移 67
資料遷移 69
監控 78
驗證 80
從 Couchbase Server 遷移 86
簡介 86
與 Amazon DocumentDB 的比較 86
探索 88
規劃 93
移轉 96
驗證 104
引擎版本支援日期 108
Amazon DocumentDB 主要版本的發行行事曆 108
Amazon DocumentDB 次要版本的發行行事曆 108
升級主要引擎版本 109
支援的升級路徑 109
先決條件 111

最佳實務 112
升級到 Amazon DocumentDB 8.0 後有何變更 115
從 3.6 或 4.0 升級之叢集的升級後考量事項 116
執行升級 116
疑難排解 119
升級次要引擎版本 119
執行次要版本升級之前 119
執行次要版本升級 120
驗證次要版本升級 121
使用 升級 AWS DMS 121
步驟 1：啟用變更串流 122
步驟 2：修改變更串流保留期間 122
步驟 3：遷移您的索引 122
步驟 4：建立 AWS DMS 複寫執行個體 123
步驟 5：建立 AWS DMS 來源端點 125
步驟 6：建立 AWS DMS 目標端點 128
步驟 7：建立和執行遷移任務 130
步驟 8：將應用程式端點變更為目標 Amazon DocumentDB 叢集 131
使用長期支援 (LTS) 版本 131
目前的 LTS 版本 131
誰應該使用 LTS 132
如何建立新的 LTS 叢集 132
如何從引擎版本 3.6.0 或 4.0.0 升級到 5.0.0 LTS 叢集 133
如何檢查我現有的引擎版本 5.0.0 叢集是否在 LTS 上 133
延伸支援 134
延伸支援概觀 135
延長支援費用 135
責任 135
功能和組態 137
引擎狀態 137
叢集類型 137
功能 138
執行個體類型 139
AWS 整合 140
選擇區域與可用區域 140
區域可用性 141

OpenSearch 整合 143
Amazon OpenSearch Service 做為目的地 144
限制 147
開發中 148
連線至 DocumentDB 148
以程式設計方式連線 149
從 Amazon VPC 外部連線 177
連接為複本集 178
使用 Studio 3T 連線 182
使用 DataGrip 連線 192
使用 Amazon EC2 連線 197
使用 JDBC 驅動程式連線 228
使用 ODBC 驅動程式連線 244
使用 Amazon DocumentDB 進行程式設計 257
DocumentDB Java 程式設計指南 257
使用 JSON 結構描述驗證 285
查詢 295
查詢文件 295
查詢計劃 297
說明結果 299
查詢規劃器 v2 304
查詢規劃器 v3 320
地理空間資料 324
部分索引 329
文字搜尋 336
索引 344
索引管理 344
索引類型 353
索引屬性 361
使用變更串流 379
受支援的操作 380
帳單 380
限制 380
啟用變更串流 381
搭配 Python 使用變更串流 382
完整文件查詢 385

繼續變更串流 386
使用 繼續 startAtOperationTime 387
使用 繼續 postBatchResumeToken 389
交易 390
修改日誌保留期間 391
變更次要執行個體上的串流 394
交易 395
要求 395
最佳實務 396
限制 396
監控和診斷 397
交易隔離層級 397
使用案例 398
支援的命令 435
不支援的功能 435
工作階段 436
交易錯誤 437
使用定序 438
限制 438
使用 檢視 439
最佳實務 439
彙總器運算子相容性 440
利用索引和檢視 440
生成式 AI 441
SageMaker Canvas 441
向量搜尋 443
使用 AWS Lambda 搭配變更串流 454
限制 454
MongoDB 相容性 455
MongoDB 8.0 相容性 455
Amazon DocumentDB 8.0 中的新功能 455
開始使用 Amazon DocumentDB 456
升級或遷移至 Amazon DocumentDB 5.0 或 8.0 456
功能差異 457
MongoDB 5.0 相容性 457
Amazon DocumentDB 5.0 中的新功能 457

Amazon DocumentDB 5.0.1 中的新功能 458
開始使用 Amazon DocumentDB 5.0 456
升級或遷移至 Amazon DocumentDB 5.0 456
功能差異 457
MongoDB 4.0 相容性 461
Amazon DocumentDB 4.0 功能 461
開始使用 Amazon DocumentDB 4.0 462
升級或遷移至 Amazon DocumentDB 4.0 462
功能差異 463
MongoDB 的功能差異 463
Amazon DocumentDB 的功能優勢 464
已更新功能差異 465
MongoDB 的功能差異 468
支援的 MongoDB APIs、操作和資料類型 476
資料庫命令 477
查詢和投影運算子 486
更新運算子 489
Geospatial (地理空間) 491
游標方法 492
彙總管道運算子 494
資料類型 510
索引和索引屬性 511
彙總管道運算子 513
Geospatial (地理空間) 1005
查詢和投影運算子 1029
更新運算子 1094
管理叢集和執行個體 1147
操作任務概觀 1147
將複本新增至 Amazon DocumentDB 叢集 1148
描述叢集和執行個體 1148
建立叢集快照 1150
從快照還原 1151
從叢集移除執行個體 1153
刪除叢集 1153
全域叢集 1154
什麼是全域叢集？ 1154

全域叢集有何用處？ 1154
全球叢集目前的限制為何？ 1154
快速入門指南 1155
管理全域叢集 1166
連接全域叢集 1172
監控全域叢集 1173
災難復原 1174
管理叢集 1195
了解叢集 1195
叢集設定 1197
叢集儲存組態 1200
判斷叢集的狀態 1202
叢集生命週期 1203
擴展叢集 1241
複製叢集的磁碟區 1244
了解叢集容錯能力 1256
管理執行個體 1257
判斷執行個體的狀態 1257
執行個體生命週期 1257
管理執行個體類別 1279
NVMe 支援的執行個體 1291
管理子網路群組 1293
建立子網路群組 1294
描述子網路群組 1299
修改子網路群組 1301
刪除子網路群組 1305
高可用性和複寫 1306
讀取擴展 1306
高可用性 1306
新增 複本 1308
容錯移轉 1308
複寫延遲 1312
管理事件 1313
檢視事件類別 1314
檢視 Amazon DocumentDB 事件 1316
管理叢集參數群組 1319

描述叢集參數群組 1319
建立叢集參數群組 1326
修改叢集參數群組 1328
修改叢集以使用自訂叢集參數群組 1333
複製叢集參數群組 1334
重設叢集參數群組 1336
刪除叢集參數群組 1339
叢集參數參考 1341
了解端點 1358
尋找叢集的端點 1359
尋找執行個體的端點 1361
連線至端點 1364
了解 Amazon DocumentDB ARNs 1366
建構 ARN 1366
尋找 ARN 1370
標記資源 1372
資源標籤概觀 1372
標籤限制條件 1373
新增或更新標籤 1373
列出標籤 1375
移除標籤 1376
維護 Amazon DocumentDB 1377
Amazon DocumentDB 的維護動作 1379
版本編號 1379
管理維護時段 1380
引擎修補程式通知 1382
檢視待定維護 1383
引擎更新 1386
次要版本 1389
作業系統更新 1390
使用者啟動的更新 1394
全域叢集 1395
了解服務連結角色 1395
服務連結角色許可 1396
建立服務連結角色 1398
修改服務連結角色 1398

刪除服務連結角色 1398
Amazon DocumentDB 服務連結角色支援的區域 1399
效能最佳化 1400
常見的效能問題 1400
高 CPU 1400
連線問題 1402
查詢執行緩慢 1408
反面模式 1412
複合索引反模式超過 3 個屬性 1412
長時間執行的查詢 1414
低使用率和備援索引 1416
具有大型陣列的多金鑰索引 1419
效能改善秘訣 1419
1. 使用 $match 作為彙總管道中的第一階段 1419
2. 在彙總管道中使用 $project 將管道資料大小降至最低 1420
3. 為更低的儲存成本、I/O 成本和改善的查詢效能啟用文件壓縮 1420
4. 利用索引獲得最佳查詢效能 1421
5. 根據查詢模式最佳化資料模型 1421
查詢計畫分析 1423
整體計時 1426
執行階段 1426
了解說明計畫執行統計資料中的 docsExamined 1427
最佳實務 1435
基本操作準則 1435
執行個體大小 1436
使用索引 1437
安全最佳實務 1438
成本最佳化 1439
使用指標來識別效能問題 1439
TTL 和時間序列工作負載 1443
遷移 1443
使用叢集參數群組 1443
彙總管道查詢 1443
`batchInsert` 和 `batchUpdate` 1444
管理索引 1444
建立 Amazon DocumentDB 索引 1444

維護 Amazon DocumentDB 索引 1449
管理文件壓縮 1451
管理文件壓縮 1452
監控文件壓縮 1453
管理字典型壓縮 1454
效能考量 1455
啟用字典型壓縮 1455
開始使用 1456
監控 1456
安全 1458
使用 進行密碼管理 AWS Secrets Manager 1459
Secrets Manager 與 Amazon DocumentDB 整合的限制 1459
使用 管理主要使用者密碼的概觀 AWS Secrets Manager 1459
在 中強制執行主要使用者密碼的 Amazon DocumentDB 管理 AWS Secrets Manager 1460
使用 Secrets Manager 管理叢集的主要使用者密碼 1461
資料保護 1461
用戶端欄位層級加密 1462
加密靜態資料 1468
加密傳輸中的資料 1475
金鑰管理 1485
身分和存取權管理 1486
目標對象 1486
使用身分驗證 1486
使用政策管理存取權 1488
Amazon DocumentDB 如何與 IAM 搭配使用 1489
身分型政策範例 1494
疑難排解 1497
管理 Amazon DocumentDB 資源的存取許可 1499
使用以身分為基礎的政策 (IAM 政策) 1503
AWS Amazon DocumentDB 的 受管政策 1507
Amazon DocumentDB API 許可參考 1525
預防跨服務混淆代理人 1533
使用 IAM 身分進行身分驗證 1534
IAM 使用者和角色入門 1535
設定運算類型 1538
監控身分驗證請求 1538

使用 IAM 身分驗證 1539
支援 IAM 的驅動程式 1539
IAM 身分驗證常見問答集 1539
使用 SCRAM-SHA-256 進行身分驗證 1540
建立 使用者 1541
更新使用者 1542
驗證 1542
管理 Amazon DocumentDB 使用者 1542
主要和serviceadmin使用者 1543
建立其他使用者 1543
自動輪換密碼 1545
以角色為基礎的存取控制 1546
RBAC 概念 1547
RBAC 內建角色入門 1548
RBAC 使用者定義角色入門 1552
以使用者身分連線至 Amazon DocumentDB 1556
常見命令 1558
功能差異 1562
限制 1563
使用角色型存取控制的資料庫存取 1563
更新憑證 1572
更新您的應用程式和 Amazon DocumentDB 叢集 1572
自動伺服器憑證輪換 1576
疑難排解 1577
常見問答集 1577
法規遵循驗證 1583
恢復能力 1583
基礎設施安全性 1584
VPC 端點 (AWS PrivateLink) 1585
VPC 端點的考量事項 1586
區域可用性 1586
為 Amazon DocumentDB API 建立介面 VPC 端點 1587
為 Amazon DocumentDB API 建立 VPC 端點政策 1587
安全最佳實務 1589
稽核事件 1590
支援的事件 1590

啟用稽核 1594
停用稽核 1600
存取您的事件 1602
篩選 DML 事件 1603
Amazon VPC 和 Amazon DocumentDB 1608
VPC 中的 DocumentDB 叢集 1609
存取 VPC 中的叢集 1619
為叢集建立IPv4-only 的 VPC 1621
為叢集建立雙堆疊 VPC 1627
備份與還原 1636
備份和還原：概念 1637
了解 備份儲存體用量 1638
傾印、還原、匯入和匯出資料 1639
mongodump 1640
mongorestore 1640
mongoexport 1641
mongoimport 1641
教學課程 1642
叢集快照考量事項 1644
備份儲存體 1645
備份時段 1645
Backup retention period (備份保留期間) 1647
複製叢集快照加密 1647
比較自動和手動快照 1648
建立手動叢集快照 1649
複製叢集快照 1652
複製共用快照 1653
跨 複製快照 AWS 區域 1653
限制 1653
處理加密 1653
參數群組考量 1654
複製叢集快照 1654
共用叢集快照 1663
共用加密快照 1663
共用快照 1666
從叢集快照還原 1670

還原至某個時間點 1676
刪除叢集快照 1682
監控和記錄 1684
監控叢集的狀態 1685
叢集狀態值 1685
監控叢集的狀態 1686
監控執行個體的狀態 1688
執行個體狀態值 1689
使用 AWS 管理主控台 或 監控執行個體狀態 AWS CLI 1690
執行個體運作狀態值 1692
使用 監控執行個體運作狀態 AWS 管理主控台 1692
檢視 Amazon DocumentDB 建議 1694
事件訂閱 1696
訂閱事件 1697
管理 訂閱 1699
類別和訊息 1702
使用 CloudWatch 監控 Amazon DocumentDB 1705
Amazon DocumentDB 指標 1705
檢視 CloudWatch 資料 1718
Amazon DocumentDB 維度 1724
監控交易指標 1724
監控資料庫連線 1725
使用 CloudTrail 記錄 Amazon DocumentDB API 呼叫 1725
CloudTrail 中的 Amazon DocumentDB 資訊 1725
分析操作 1726
受支援的操作 1727
限制 1727
啟用分析器 1728
停用分析器 1732
停用分析器日誌匯出 1732
存取您的分析器日誌 1735
常見查詢 1735
使用績效詳情進行監控 1736
績效詳情概念 1737
啟用和停用績效詳情 1740
設定績效詳情的存取政策 1743

使用績效詳情儀表板來分析指標 1747
使用績效詳情 API 來擷取指標 1764
Performance Insights 的 Amazon CloudWatch 指標 1778
計數器指標的績效詳情 1780
無伺服器 1782
無伺服器使用案例 1782
針對現有的佈建工作負載使用 Amazon DocumentDB 無伺服器 1783
Amazon DocumentDB 無伺服器的優點 1784
無伺服器的運作方式 1785
概觀 1785
Amazon DocumentDB 叢集的組態 1786
Amazon DocumentDB 無伺服器擴展容量 1786
Amazon DocumentDB 無伺服器擴展 1787
閒置狀態 (0.5 DCUs) 1789
無伺服器需求和限制 1789
功能需求 1789
建立使用無伺服器 的叢集 1791
建立 Amazon DocumentDB 無伺服器叢集 1791
新增 Amazon DocumentDB 無伺服器執行個體 1794
遷移至無伺服器 1794
將現有的 DocumentDB 叢集遷移至無伺服器 1795
從 MongoDB 遷移至 DocumentDB 無伺服器 1799
管理無伺服器 1799
檢視和修改叢集的擴展容量範圍組態 1799
檢視和修改無伺服器讀取器的提升層 1802
無伺服器執行個體限制 1804
無伺服器擴展組態 1806
選擇 DocumentDB 無伺服器叢集的擴展容量範圍 1807
選擇 DocumentDB 無伺服器叢集MinCapacity的設定 1807
選擇 DocumentDB 無伺服器叢集MaxCapacity的設定 1809
避免記憶體不足錯誤 1810
為什麼我的無伺服器執行個體沒有縮減規模？ 1811
監控無伺服器 1811
記憶體不足：不相容參數狀態 1812
DocumentDB Serverless 的 Amazon CloudWatch 指標 1812
使用績效詳情監控 DocumentDB 無伺服器效能 1816

彈性叢集 1817
彈性叢集使用案例 1818
使用者設定檔 1818
內容管理和歷史記錄 1818
彈性叢集的優點 1818
AWS 服務整合 1818
區域和版本可用性 1819
彈性叢集的區域可用性 1819
版本可用性 1820
限制 1820
彈性叢集管理 1820
查詢和寫入操作 1820
集合和索引管理 1821
管理和診斷 1821
選擇加入功能 1821
運作方式 1822
Amazon DocumentDB 彈性叢集碎片 1822
彈性叢集遷移 1826
彈性叢集擴展 1826
彈性叢集可靠性 1826
彈性叢集儲存和可用性 1826
Amazon DocumentDB 4.0 和彈性叢集之間的功能差異 1826
開始使用 1828
先決條件 1828
步驟 1：建立彈性叢集 1830
步驟 2：連線至您的彈性叢集 1835
步驟 3：碎片化您的集合、插入和查詢資料 1836
步驟 4：探索 1839
最佳實務 1840
選擇碎片索引鍵 1840
連線管理 1840
未碎片的集合 1840
擴展彈性叢集 1841
監控彈性叢集 1841
管理彈性叢集 1842
修改彈性叢集組態 1842

監控彈性叢集 1845
刪除彈性叢集 1848
管理彈性叢集快照 1850
停止和啟動彈性叢集 1863
維護彈性叢集 1867
靜態資料加密 1874
Amazon DocumentDB 彈性叢集如何在 中使用授予 AWS KMS 1876
建立客戶自管金鑰 1876
監控 Amazon DocumentDB 彈性叢集的加密金鑰 1877
進一步了解 1883
服務連結角色 1883
彈性叢集的服務連結角色許可 1883
配額 1887
支援的執行個體類型 1887
支援的區域 1889
區域配額 1891
彙總配額 1893
叢集配額 1893
執行個體配額 1895
命名限制條件 1902
TTL 限制條件 1904
彈性叢集配額 1904
彈性叢集碎片配額 1905
每個碎片的彈性叢集 CPU、記憶體、連線和游標配額 1905
疑難排解 1907
連線問題 1907
無法連線至 Amazon DocumentDB 端點 1907
測試與 Amazon DocumentDB 執行個體的連線 1912
連線至無效的端點 1913
影響連線數量的驅動程式組態 1913
索引 1914
索引建置失敗 1914
背景索引建置延遲問題和失敗 1914
索引膨脹 1915
效能和資源使用率 1916
檢視插入、更新和刪除統計資料 1916

分析快取效能 1918
尋找並終止長時間執行或封鎖的查詢 1919
查看查詢計畫並最佳化查詢 1920
如何查看彈性叢集中的查詢計劃？ 1922
列出執行個體上所有執行中的操作 1924
了解查詢何時進行 1927
判斷系統突然緩慢執行的原因 1929
判斷造成高 CPU 使用率的原因 1931
尋找執行個體上的開啟游標 1931
檢視目前的 Amazon DocumentDB 引擎版本 1932
分析索引用量並識別未使用的索引 1932
識別遺漏的索引 1934
如何判斷資料庫集合膨脹？ 1936
有用的查詢摘要 1937
垃圾回收 1938
了解 Amazon DocumentDB 中的垃圾回收 1938
垃圾回收程序 1939
儲存架構和擴充儲存 1939
監控垃圾回收 1940
collStats 輸出範例 1942
常見問答集 1944
錯誤代碼 1947
版本備註 1976
Amazon DocumentDB 引擎版本備註 1976
一般 Amazon DocumentDB 版本備註 2026
Amazon DocumentDB 彈性叢集版本備註 2031
文件歷史紀錄 2035
..... mmxliv

什麼是 Amazon DocumentDB （與 MongoDB 相容）

Amazon DocumentDB （與 MongoDB 相容） 是一種快速、可靠且全受管的資料庫服務。Amazon DocumentDB 可讓您在雲端輕鬆設定、操作和擴展與 MongoDB 相容的資料庫。透過 Amazon DocumentDB，您可以執行相同的應用程式程式碼，並使用與 MongoDB 搭配使用的相同驅動程式和工具。

使用 Amazon DocumentDB 之前，您應該檢閱中所述的概念和功能運作方式。接著再完成 入門指南 中的步驟。

主題

- Amazon DocumentDB 概觀
- 叢集
- 執行個體
- 區域與可用區域
- Amazon DocumentDB 定價
- 監控
- 介面
- 後續步驟？
- Amazon DocumentDB：運作方式
- 什麼是文件資料庫？

Amazon DocumentDB 概觀

以下是 Amazon DocumentDB 的一些高階功能：

- Amazon DocumentDB 支援兩種類型的叢集：執行個體型叢集和彈性叢集。彈性叢集支援每秒數百萬個讀取/寫入和 PB 儲存容量的工作負載。如需彈性叢集的詳細資訊，請參閱 使用 Amazon DocumentDB 彈性叢集。以下內容是指 Amazon DocumentDB 執行個體型叢集 。
- 隨著資料庫儲存需求的增加，Amazon DocumentDB 會自動增加儲存磁碟區的大小。您的儲存磁碟區會以 10 GB 的增量增加，引擎版本 8.0 及更高版本的最大容量為 256 TiB （舊引擎版本為 128 TiB)。您不需要為叢集佈建任何過多的儲存空間來處理未來的增長。
- 使用 Amazon DocumentDB，您可以建立最多 15 個複本執行個體，以增加讀取輸送量以支援大量應用程式請求。Amazon DocumentDB 複本共用相同的基礎儲存體，可降低成本並避免需要在複本節

點執行寫入。此功能可釋放更多處理能力來提供讀取請求，並將複本延遲時間縮短為單一位數毫秒。無論儲存磁碟區大小為何，您都可以在幾分鐘內新增複本。Amazon DocumentDB 也提供讀取器端點，因此應用程式可以連線，而無需在新增和移除複本時追蹤複本。

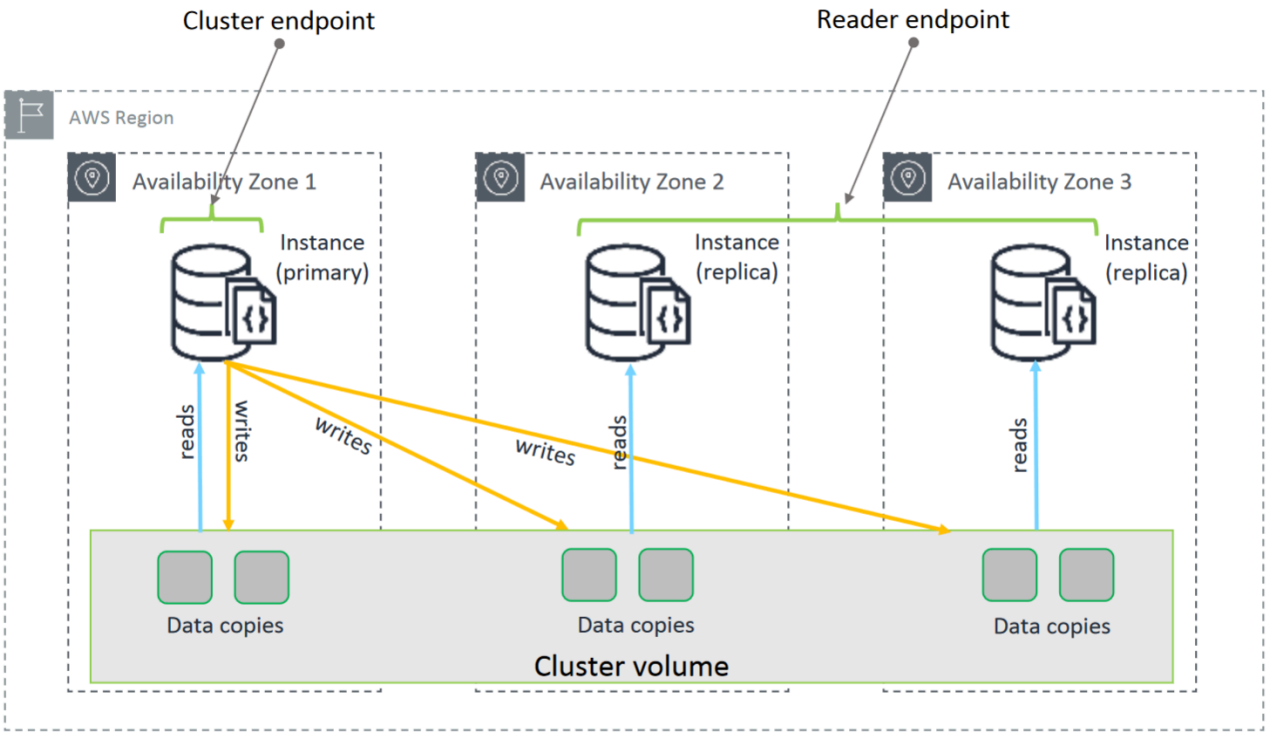
- Amazon DocumentDB 可讓您擴展或縮減每個執行個體的運算和記憶體資源。運算擴展操作通常可在幾分鐘內完成。
- Amazon DocumentDB 在 Amazon Virtual Private Cloud (Amazon VPC) 中執行，因此您可以在自己的虛擬網路中隔離資料庫。您也可以設定防火牆設定，以控制您叢集的網路存取。
- Amazon DocumentDB 會持續監控叢集的運作狀態。在執行個體故障時，Amazon DocumentDB 會自動重新啟動執行個體和相關聯的程序。Amazon DocumentDB 不需要資料庫重做日誌的損毀復原重播，可大幅縮短重新啟動時間。Amazon DocumentDB 也會隔離資料庫快取與資料庫程序，讓快取在執行個體重新啟動後仍然存在。
- 執行個體故障時，Amazon DocumentDB 會自動容錯移轉至您在其他可用區域中建立的最多 15 個 Amazon DocumentDB 複本之一。如果未佈建複本且發生故障，Amazon DocumentDB 會嘗試自動建立新的 Amazon DocumentDB 執行個體。
- Amazon DocumentDB 中的備份功能可為您的叢集啟用point-in-time復原。此功能可讓您將叢集還原至保留期間內任何一秒鐘的狀態，最多可恢復到前五分鐘的狀態。您可以設定最長 35 天的自動備份保留期。自動化備份存放在 Amazon Simple Storage Service (Amazon S3) 中，專為 99.999999999% 耐用性而設計。Amazon DocumentDB 備份是自動、增量和連續的，不會影響您的叢集效能。
- 使用 Amazon DocumentDB，您可以使用您透過 AWS Key Management Service () 建立和控制的金鑰來加密資料庫AWS KMS。在執行 Amazon DocumentDB 加密的資料庫叢集上，基礎儲存體中靜態儲存的資料會加密。在相同叢集中的自動備份、快照及複本也會經過加密。
- Amazon DocumentDB 已根據聯邦風險與授權管理計劃 (FedRAMP) 進行授權。它具有 FedRAMP 高授權 for AWS GovCloud (US) 區域和 FedRAMP AWS 美國東部/西部區域的中等授權。如需 AWS 和合規工作的詳細資訊，請參閱AWS 合規計劃範圍內的服務。

如果您是初次使用 AWS 服務，請使用下列資源進一步了解：

- AWS 提供運算、資料庫、儲存、分析和其他功能的 服務。如需所有 AWS 服務的概觀，請參閱使用 Amazon Web Services 進行雲端運算。
- AWS 提供多種資料庫服務。如需最適合您環境的服務指引，請參閱 上的資料庫 AWS。

叢集

叢集包含 0 到 16 個執行個體，以及管理這些執行個體資料的叢集儲存磁碟區。所有寫入都是透過主要執行個體完成。所有執行個體 (主要和複本) 都支援讀取。叢集的資料儲存在叢集磁碟區中，其複本會存放在三個不同的可用區域。



Amazon DocumentDB 5.0 執行個體型叢集支援資料庫叢集的兩種儲存組態：Amazon DocumentDB 標準和 Amazon DocumentDB I/O 最佳化。如需更多資訊，請參閱Amazon DocumentDB 叢集儲存組態。

執行個體

Amazon DocumentDB 執行個體是雲端中的隔離資料庫環境。執行個體可以包含多個使用者建立的資料庫。您可以使用 AWS 管理主控台 或 建立和修改執行個體 AWS CLI。

執行個體的運算和記憶體容量取決於其執行個體類別。您可以選擇最符合您需求的執行個體。如果您的需求在一段時間後改變，您可以選擇其他執行個體類別。如需執行個體類別規格，請參閱執行個體類別規格。

Amazon DocumentDB 執行個體僅在 Amazon VPC 環境中執行。Amazon VPC 可讓您控制虛擬聯網環境：您可以選擇自己的 IP 地址範圍、建立子網路，以及設定路由和存取控制清單 ACLs)。

您必須先建立叢集以包含執行個體，才能建立 Amazon DocumentDB 執行個體。

並非所有執行個體類別都支援 AWS 區域。下表顯示每個區域支援哪些執行個體類別。

> **Note**
>
> 如需每個執行個體類別中 Amazon DocumentDB 支援的執行個體類型完整清單，請參閱 執行個體類別規格。

各區域的支援執行個體類別

區域	R8G	R6GD	R6G	R5	R4	T4G	T3	無伺服器
美國東部 (俄亥俄)	支援	支援	支援	支援	支援	支援	支援	支援
美國東部 (維吉尼亞北部)	支援	支援	支援	支援	支援	支援	支援	支援
美國西部 (奧勒岡)	支援	支援	支援	支援	支援	支援	支援	支援
非洲 (開普敦)			支援	支援		支援	支援	支援
南美洲 (聖保羅)	支援	支援	支援	支援		支援	支援	支援
亞太地區 (香港)	支援		支援	支援		支援	支援	支援
亞太地區 (海德拉巴)			支援	支援		支援	支援	支援
亞太地區 (馬來西亞)			支援			支援	支援	支援
亞太地區 (孟買)	支援	支援	支援	支援		支援	支援	支援
亞太地區 (大阪)	支援	支援	支援	支援		支援	支援	支援
亞太地區 (首爾)	支援	支援	支援	支援		支援	支援	支援

區域	R8G	R6GD	R6G	R5	R4	T4G	T3	無伺服器
亞太地區 (悉尼)	支援	支援	支援	支援		支援	支援	支援
亞太地區 (雅加達)	支援	支援	支援	支援		支援	支援	支援
亞太地區 (墨爾本)			支援	支援		支援	支援	支援
亞太地區 (新加坡)	支援	支援	支援	支援		支援	支援	支援
亞太區域 (泰國)			支援			支援	支援	支援
亞太地區 (東京)	支援	支援	支援	支援		支援	支援	支援
加拿大 (中部)	支援	支援	支援	支援		支援	支援	支援
加拿大西部 (卡加利)			支援	支援		支援	支援	支援
歐洲 (法蘭克福)	支援	支援	支援	支援		支援	支援	支援
歐洲 (蘇黎世)		支援	支援	支援		支援	支援	支援
歐洲 (愛爾蘭)	支援	支援	支援	支援	支援	支援	支援	支援
歐洲 (倫敦)	支援	支援	支援	支援		支援	支援	支援
歐洲 (米蘭)			支援	支援		支援	支援	支援
Europe (Paris)	支援	支援	支援	支援		支援	支援	支援
歐洲 (西班牙)	支援	支援	支援	支援		支援	支援	支援
歐洲 (斯德哥爾摩)	支援	支援	支援	支援		支援	支援	支援
墨西哥 (中部)			支援			支援	支援	支援

區域	R8G	R6GD	R6G	R5	R4	T4G	T3	無伺服器
中東 (阿拉伯聯合大公國)			支援	支援		支援	支援	支援
中國 (北京)	支援	支援	支援	支援		支援	支援	支援
中國 (寧夏)	支援		支援	支援		支援	支援	支援
以色列 (特拉維夫)			支援	支援		支援	支援	支援
AWS GovCloud （美國西部）	支援	支援	支援	支援			支援	支援
AWS GovCloud （美國東部）		支援	支援	支援		支援	支援	支援

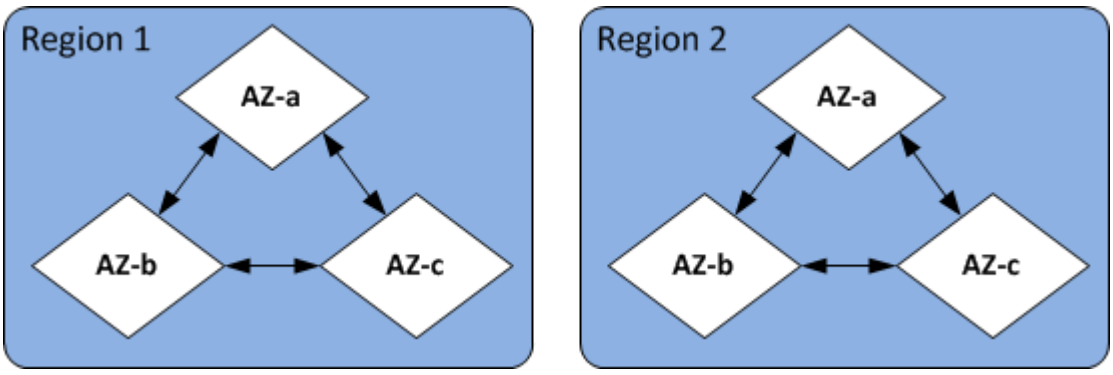
區域與可用區域

AWS 區域 和 可用區域定義叢集和執行個體的實體位置。

大區 (Regions)

AWS 雲端運算資源位於世界不同區域 （例如北美洲、歐洲或亞洲） 高度可用的資料中心設施。各資料中心位置皆稱為 AWS 區域。

每個 AWS 區域 都設計成與另一個完全隔離 AWS 區域。每個區域中皆有多個可用區域。藉由在不同的可用區域中啟動您的節點，您可以實現最大的容錯能力。下圖顯示 AWS 區域 和可用區域運作方式的高階檢視。



可用區域

每個 AWS 區域 都包含多個不同的位置，稱為可用區域。各個可用區域的設計，主要是為了與其他可用區域的故障事件隔離，並以低成本、低延遲的方式，透過網路連線至相同區域中的其他可用區域。藉由在多個可用區域中針對指定叢集啟動執行個體，您就可以保護應用程式免於發生可用區域故障的事件。

Amazon DocumentDB 架構會區隔儲存和運算。對於儲存層，Amazon DocumentDB 會跨三個 AWS 可用區域複寫資料的六個副本。例如，如果您在僅支援兩個可用區域的區域中啟動 Amazon DocumentDB 叢集，您的資料儲存體將跨三個可用區域以六種方式複寫，但您的運算執行個體只能在兩個可用區域中使用。

下表列出您可以在指定 中使用的可用區域數量 AWS 區域 ，以為您的叢集佈建運算執行個體。

區域名稱	區域	可用區域 （運算）
美國東部 (俄亥俄)	us-east-2	3
美國東部 (維吉尼亞北部)	us-east-1	6
美國西部 (奧勒岡)	us-west-2	4
非洲 (開普敦)	af-south-1	3
南美洲 (聖保羅)	sa-east-1	3
亞太地區 (香港)	ap-east-1	3
亞太地區 (海德拉巴)	ap-south-2	3

區域名稱	區域	可用區域（運算）
亞太地區 (馬來西亞)	ap-southeast-5	3
亞太地區 (孟買)	ap-south-1	3
亞太地區 (大阪)	ap-northeast-3	3
亞太地區 (首爾)	ap-northeast-2	4
亞太地區 (新加坡)	ap-southeast-1	3
亞太地區 (悉尼)	ap-southeast-2	3
亞太地區 (雅加達)	ap-southeast-3	3
亞太地區 (墨爾本)	ap-southeast-4	3
亞太區域 (泰國)	ap-southeast-7	3
亞太地區 (東京)	ap-northeast-1	3
加拿大 (中部)	ca-central-1	3
加拿大西部 (卡加利)	ca-west-1	3
中國 (北京) 區域	cn-north-1	3
中國 (寧夏)	cn-northwest-1	3
歐洲 (法蘭克福)	eu-central-1	3
歐洲 (蘇黎世)	eu-central-2	3
歐洲 (愛爾蘭)	eu-west-1	3
歐洲 (倫敦)	eu-west-2	3
歐洲 (米蘭)	eu-south-1	3
Europe (Paris)	eu-west-3	3

區域名稱	區域	可用區域（運算）
歐洲 (西班牙)	eu-south-2	3
歐洲 (斯德哥爾摩)	eu-north-1	3
墨西哥 (中部)	mx-central-1	3
中東 (阿拉伯聯合大公國)	me-central-1	3
以色列 (特拉維夫)	il-central-1	3
AWS GovCloud（美國西部）	us-gov-west-1	3
AWS GovCloud（美國東部）	us-gov-east-1	3

Amazon DocumentDB 定價

Amazon DocumentDB 叢集會根據下列元件計費：

- 執行個體時數（每小時）—根據執行個體的執行個體類別（例如 db.r5.xlarge)。定價以每小時為單位列出，但帳單已採用秒數為計算單位，並以十進位制顯示時間。Amazon DocumentDB 用量以一秒遞增計費，最少 10 分鐘。如需詳細資訊，請參閱管理執行個體類別。
- I/O 請求（每月每 100 萬個請求）— 您在計費週期中提出的儲存 I/O 請求總數。
- 備份儲存體（每月每 GiB) — 備份儲存體是與自動資料庫備份以及您拍攝的任何作用中資料庫快照相關聯的儲存體。延長您的備份保留期或拍攝額外的資料庫快照，會增加資料庫所消耗的備份儲存。備份儲存是以 GB 月計算，因此每秒不適用。如需詳細資訊，請參閱在 Amazon DocumentDB 中備份和還原。
- 資料傳輸（每 GB) — 從執行個體傳入或傳出至網際網路或其他 的資料傳輸 AWS 區域。

如需詳細資訊，請參閱 Amazon DocumentDB 定價。

免費試用

您可以使用 1 個月的免費試用，免費試用 Amazon DocumentDB。如需詳細資訊，請參閱 Amazon DocumentDB 定價網站上的免費試用，或參閱 Amazon DocumentDB 免費試用常見問答集。

監控

追蹤執行個體的效能與運作狀態有多種方式。您可以使用免費的 Amazon CloudWatch 服務來監控執行個體的效能和運作狀態。您可以在 Amazon DocumentDB 主控台上找到效能圖表。您可以訂閱 Amazon DocumentDB 事件，以便在執行個體、快照、參數群組或安全群組發生變更時收到通知。

如需詳細資訊，請參閱下列內容：

- 使用 CloudWatch 監控 Amazon DocumentDB
- 使用 記錄 Amazon DocumentDB API 呼叫 AWS CloudTrail

介面

您可以透過多種方式與 Amazon DocumentDB 互動，包括 AWS 管理主控台 和 AWS CLI。

AWS 管理主控台

AWS 管理主控台 是簡單的 Web 型使用者介面。不需編寫程式，即可從主控台管理您的叢集和執行個體。若要存取 Amazon DocumentDB 主控台，請登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/docdb 的 Amazon DocumentDB 主控台。

AWS CLI

您可以使用 AWS Command Line Interface (AWS CLI) 來管理您的 Amazon DocumentDB 叢集和執行個體。透過最少的組態，您可以從您最愛的終端機程式開始使用 Amazon DocumentDB 主控台提供的所有功能。

- 若要安裝 AWS CLI，請參閱安裝 AWS 命令列界面。
- 若要開始使用 AWS CLI 適用於 Amazon DocumentDB 的 ，請參閱適用於 AWS Amazon DocumentDB 的命令列界面參考。

MongoDB 驅動程式

若要針對 Amazon DocumentDB 叢集開發和撰寫應用程式，您也可以搭配 Amazon DocumentDB 使用 MongoDB 驅動程式。如需詳細資訊，請參閱 使用已啟用的 TLS 連線或 中的 MongoDB shell 標籤使用已停用的 TLS 連線。

後續步驟？

上述各節向您介紹 Amazon DocumentDB 提供的基本基礎設施元件。接下來該怎麼做？ 根據您的情況，請參閱下列其中一個主題以開始使用：

- 使用 建立叢集和執行個體，以開始使用 Amazon DocumentDB CloudFormation 使用 的 Amazon DocumentDB 快速入門 CloudFormation。
- 使用 中的指示建立叢集和執行個體，以開始使用 Amazon DocumentDB入門指南。
- 使用 中的指示建立彈性叢集，以開始使用 Amazon DocumentDBAmazon DocumentDB 彈性叢集入門。
- 使用 中的指引將您的 MongoDB 實作遷移至 Amazon DocumentDB 遷移和升級 Amazon DocumentDB

Amazon DocumentDB：運作方式

Amazon DocumentDB （與 MongoDB 相容） 是全受管、與 MongoDB 相容的資料庫服務。透過 Amazon DocumentDB，您可以執行相同的應用程式程式碼，並使用與 MongoDB 搭配使用的相同驅動程式和工具。Amazon DocumentDB 與 MongoDB 3.6、4.0、5.0 和 8.0 相容。

主題

- Amazon DocumentDB 版本
- Amazon DocumentDB 端點
- TLS 支援
- Amazon DocumentDB 儲存體
- Amazon DocumentDB 複寫
- Amazon DocumentDB 可靠性
- 讀取偏好設定選項
- TTL 刪除
- 計費資源

Amazon DocumentDB 版本

> **Note**
>
> 次要版本編號適用於 Amazon DocumentDB 5.0 和更新版本。Amazon DocumentDB 3.6 和 4.0 僅使用具有引擎修補程式的主要版本。

從 Amazon DocumentDB 5.0 開始，引擎版本使用四層編號機制： major.major.minor.patch。

- 主要版本 - 由版本的前兩個號碼識別（例如 5.0)。主要版本代表 MongoDB 相容性層級。主要版本推出新功能、效能改善和 MongoDB 相容性更新。在主要版本（例如，從 4.0 升級至 5.0) 之間升級需要主要版本升級 (MVU)。如需詳細資訊，請參閱Amazon DocumentDB 就地主要版本升級。
- 次要版本 - 以版本的第三個號碼識別（例如 5.0.1 1中的)。次要版本包括改善，例如相同主要版本中的新功能、安全性修正和錯誤修正。您可以手動升級至新的次要版本。如需詳細資訊，請參閱Amazon DocumentDB 次要版本升級。
- 修補程式版本 - 引擎修補程式會在次要版本中套用，並包含重要的安全性修正和錯誤修正。修補程式版本由引擎修補程式版本（例如 `3.0.17983`) 追蹤，您可以透過執行 來檢查`db.runCommand({getEngineVersion: 1})`。修補程式會在叢集的維護時段期間套用。如需詳細資訊，請參閱維護 Amazon DocumentDB。

建立新叢集時，您可以指定任何可用的引擎版本。如需可用版本和支援日期的相關資訊，請參閱 Amazon DocumentDB 引擎版本支援日期。

當您使用 Amazon DocumentDB 時，請先建立叢集。資料庫叢集包含零或多個資料庫執行個體，和一個管理這些執行個體資料的叢集磁碟區。Amazon DocumentDB 叢集磁碟 區是跨越多個可用區域的虛擬資料庫儲存磁碟區。每個可用區域具有叢集資料的複本。

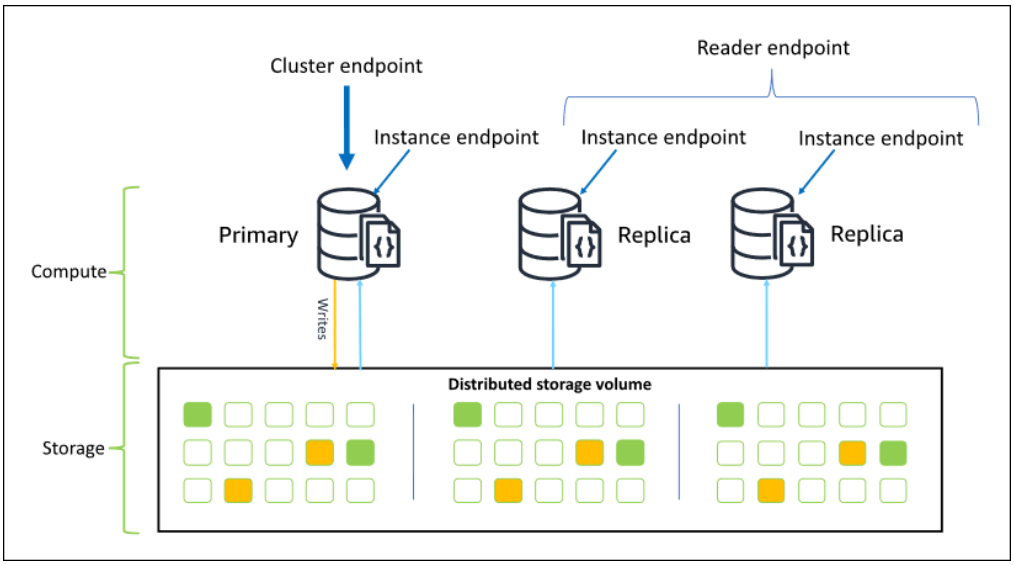
Amazon DocumentDB 叢集包含兩個元件：

- 叢集磁碟區 - 使用雲端原生儲存服務跨三個可用區域以六種方式複寫資料，提供高耐用性和可用的儲存。Amazon DocumentDB 叢集只有一個叢集磁碟區，可儲存引擎版本 8.0 及更高版本最多 256 TiB 的資料（舊版引擎為 128 TiB)。
- 執行個體 - 提供資料庫的處理能力、將資料寫入叢集儲存磁碟區，以及從中讀取資料。Amazon DocumentDB 叢集可以有 0–16 個執行個體。

執行個體提供以下兩種角色：

- 主要執行個體 - 支援讀取和寫入操作，並對叢集磁碟區執行所有資料修改。每個 Amazon DocumentDB 叢集都有一個主要執行個體。
- 複本執行個體 - 僅支援讀取操作。除了主要執行個體之外，Amazon DocumentDB 叢集最多可以有 15 個複本。擁有多個複本可讓您分配讀取工作負載。此外，透過將複本置於不同可用區域，您也可提高叢集可用性。

下圖說明 Amazon DocumentDB 叢集中的叢集磁碟區、主要執行個體和複本之間的關係：



叢集執行個體不需為相同執行個體類別，而您可以隨需要佈建和終止它們。這個架構可讓您擴展叢集的運算容量，而不影響儲存。

當您的應用程式會將資料寫入主執行個體，主執行個體會將耐用資料寫入叢集磁碟區。然後，它會將該寫入的狀態（而非資料）複寫到每個作用中的複本。Amazon DocumentDB 複本不參與處理寫入，因此 Amazon DocumentDB 複本有利於讀取擴展。Amazon DocumentDB 複本的讀取最終會與最小複本延遲一致，通常在主要執行個體寫入資料後不到 100 毫秒。從複本的讀取保證會以複本寫入主要執行個體的順序來進行讀取。複本延遲取決於資料變更的速率，以及高寫入活動的期間可能會增加複本延遲。如需詳細資訊，請參閱 Amazon DocumentDB 指標 的 `ReplicationLag` 指標。

Amazon DocumentDB 端點

Amazon DocumentDB 提供多種連線選項，適用於各種使用案例。若要連線至 Amazon DocumentDB 叢集中的執行個體，請指定執行個體的端點。端點是主機地址和連接埠號碼 (以冒號分隔)。

在複本集模式下使用叢集端點連線至叢集（請參閱 以複本集的形式連線至 Amazon DocumentDB)。除非您必須連線至讀取器端點或執行個體端點，否則請使用此方法。若要將請求路由到您的複

本，請選擇驅動程式讀取偏好設定，以發揮最大讀取擴展，同時符合您應用程式的讀取一致性要求。secondaryPreferred 讀取偏好設定會啟用複本讀取，並釋出主要執行個體以進行更多工作。

下列端點可從 Amazon DocumentDB 叢集取得。

叢集端點

叢集端點會連接至叢集目前的主要執行個體。您可以使用叢集端點來執行讀取和寫入操作。Amazon DocumentDB 叢集只有一個叢集端點。

叢集端點可為叢集的讀寫連接提供容錯移轉支援。如果叢集目前的主要執行個體發生故障，您的叢集至少有一個作用中的僅供讀取複本，叢集端點會自動重新導向到新主要執行個體的連線請求。連線至 Amazon DocumentDB 叢集時，請在複本集模式下使用叢集端點 （ 請參閱 以複本集的形式連線至 Amazon DocumentDB)。

以下是 Amazon DocumentDB 叢集端點的範例：

```
sample-cluster.cluster-123456789012.us-east-1.docdb.amazonaws.com:27017
```

以下是使用此叢集端點的範例連線字串：

```
mongodb://username:password@sample-cluster.cluster-123456789012.us-
east-1.docdb.amazonaws.com:27017
```

如需尋找叢集端點的詳細資訊，請參閱尋找叢集的端點。

讀取器端點

讀取器端點負載會平衡在叢集中所有可用複本的僅供讀取連線。如果您透過 replicaSet 模式連線，叢集讀取器端點將做為叢集端點執行，這表示在連線字串中，複本集參數為 &replicaSet=rs0。在這種情況下，您將能夠在主要 上執行寫入操作。不過，如果您連線到指定 的叢集directConnection=true，則嘗試透過與讀取器端點的連線執行寫入操作會導致錯誤。Amazon DocumentDB 叢集只有一個讀取器端點。

如果叢集僅包含一個 (主要) 執行個體，讀取器端點會連接至主要執行個體。當您將複本執行個體新增至 Amazon DocumentDB 叢集時，讀取器端點會在新複本處於作用中狀態後開啟其唯讀連線。

以下是 Amazon DocumentDB 叢集的範例讀取器端點：

```
sample-cluster.cluster-ro-123456789012.us-east-1.docdb.amazonaws.com:27017
```

以下是使用此讀取器端點的範例連線字串：

```
mongodb://username:password@sample-cluster.cluster-ro-123456789012.us-
east-1.docdb.amazonaws.com:27017
```

讀取器端點負載會平衡唯讀連線，而不是讀取請求。如果一些讀取器端點連線的使用量較其他連線頻繁，則系統可能無法在叢集中的執行個體間平均地平衡讀取請求。建議連線至做為複本集的叢集端點，並利用 secondaryPreferred 讀取偏好設定選項，來發佈請求。

如需尋找叢集端點的詳細資訊，請參閱尋找叢集的端點。

執行個體端點

執行個體端點會連線到叢集中的特定執行個體。您可以將目前主要執行個體的執行個體端點用於讀取和寫入操作。不過，嘗試對執行個體端點執行讀取複本的寫入操作可能會造成錯誤。Amazon DocumentDB 叢集每個作用中執行個體都有一個執行個體端點。

執行個體端點對特定執行個體的連接提供直接控制，使用叢集端點或讀取器端點的案例可能不適用。範例使用案例佈建的是適用於週期性唯讀分析工作負載。您可以佈建大於一般的複本執行個體、使用其執行個體端點直接連接到新的大型執行個體、執行分析查詢，接著終止執行個體。使用執行個體端點可讓分析流量不會影響到其他叢集執行個體。

以下是 Amazon DocumentDB 叢集中單一執行個體的範例執行個體端點：

```
sample-instance.123456789012.us-east-1.docdb.amazonaws.com:27017
```

以下是使用此執行個體端點的範例連線字串：

```
mongodb://username:password@sample-instance.123456789012.us-
east-1.docdb.amazonaws.com:27017
```

> **Note**
>
> 做為主要或複本的執行個體角色可以因為容錯移轉事件而變更。應用程式絕不應該假設特定執行個體端點是主執行個體。請勿連線至生產應用程式的執行個體端點。反之，請在複本集模式下使用叢集端點（請參閱 以複本集的形式連線至 Amazon DocumentDB）。如需執行個體容錯移轉優先順序更進階的控制，請參閱了解 Amazon DocumentDB 叢集容錯能力。

如需尋找叢集端點的詳細資訊，請參閱尋找執行個體的端點。

複本集模式

您可以指定複本集名稱 ，以複本集模式連線至 Amazon DocumentDB 叢集端點`rs0`。在複本集模式中連接可讓您指定 Read Concern、Write Concern 和 Read Preference 選項。如需詳細資訊，請參閱讀取一致性。

以下是在複本集模式中連接的範例連線字串：

```
mongodb://username:password@sample-cluster.cluster-123456789012.us-
east-1.docdb.amazonaws.com:27017/?replicaSet=rs0
```

當您以複本集模式連線時，Amazon DocumentDB 叢集會以複本集的形式顯示到您的驅動程式和用戶端。從 Amazon DocumentDB 叢集新增和移除的執行個體會自動反映在複本集組態中。

每個 Amazon DocumentDB 叢集由具有預設名稱 的單一複本集組成`rs0`。複本集名稱無法修改。

若是一般用途，建議連接到在複本集模式中的叢集端點。

> **Note**
>
> Amazon DocumentDB 叢集中的所有執行個體都會接聽相同的 TCP 連接埠以進行連線。

TLS 支援

如需使用 Transport Layer Security (TLS) 連線至 Amazon DocumentDB 的詳細資訊，請參閱 加密傳輸中的資料。

Amazon DocumentDB 儲存體

Amazon DocumentDB 資料存放在叢集磁碟區中，這是使用固態硬碟 (SSDs的單一虛擬磁碟區。叢集磁碟區包含資料的六個副本，會在單一可用區域中自動複寫 AWS 區域。此複寫有助於確保您的資料具有高耐用性，同時降低資料遺失的機率。它也有助於確保您的叢集在容錯移轉時提供更高的可用性，因為資料副本已存在於其他可用區域。這些副本可以繼續將資料請求提供給 Amazon DocumentDB 叢集中的執行個體。

資料儲存體的計費方式

隨著資料量增加，Amazon DocumentDB 會自動增加叢集磁碟區的大小。8.0 版及更高版本的引擎，Amazon DocumentDB 叢集磁碟區的大小上限為 256 TiB （舊版引擎為 128 TiB)。不過，您只需

支付您在 Amazon DocumentDB 叢集磁碟區中使用的空間費用。從 Amazon DocumentDB 4.0 開始，移除資料時，例如捨棄集合或索引，整體配置的空間會減少相當的數量。因此，您可以透過刪除不再需要的集合、索引和資料庫來降低儲存費用。在 Amazon DocumentDB 3.6 版中，叢集磁碟區可以重複使用移除資料時釋放的空間，但磁碟區本身的大小永遠不會減少。在 3.6 版中，即使您捨棄集合或索引，即使已釋放的空間已重複使用，您也可能不會見證儲存體中的任何變更。

> **Note**
>
> 使用 Amazon DocumentDB 3.6 時，儲存成本是根據儲存體「高浮水印」（任何時間點為 Amazon DocumentDB 叢集配置的最大數量）。您可以透過避免建立大量臨時資訊的 ETL 實務，或在移除不需要的舊資料之前載入大量新資料來管理成本。如果從 Amazon DocumentDB 叢集移除資料會產生大量已配置但未使用的空間，則重設高浮水印需要執行邏輯資料傾印，並使用 mongodump或 等工具還原至新叢集mongorestore。建立和還原快照不會減少配置的儲存空間，因為基礎儲存的實體配置在還原的快照中保持不變。

> **Note**
>
> 使用 mongodump 和 mongorestore 之類的公用程式會根據所讀取和寫入儲存磁碟區的資料大小產生 I/O 費用。

如需 Amazon DocumentDB 資料儲存和 I/O 定價的相關資訊，請參閱 Amazon DocumentDB（與 MongoDB 相容）定價和定價FAQs。

Amazon DocumentDB 複寫

在 Amazon DocumentDB 叢集中，每個複本執行個體都會公開獨立端點。這些複本端點對叢集磁碟區的資料提供唯讀存取權。它們可讓您隨著多個複寫執行個體擴展資料的讀取工作負載。它們也有助於改善資料讀取的效能，並提高 Amazon DocumentDB 叢集中資料的可用性。Amazon DocumentDB 複本也是容錯移轉目標，如果 Amazon DocumentDB 叢集的主要執行個體失敗，則會快速提升。

Amazon DocumentDB 可靠性

Amazon DocumentDB 的設計可靠、耐用且容錯。（為了改善可用性，您應該設定 Amazon DocumentDB 叢集，以便在不同的可用區域中擁有多個複本執行個體。) Amazon DocumentDB 包含數個自動功能，使其成為可靠的資料庫解決方案。

儲存體自動修復

Amazon DocumentDB 會在三個可用區域中維護資料的多個副本，大幅降低因儲存故障而遺失資料的機會。Amazon DocumentDB 會自動偵測叢集磁碟區中的失敗。當叢集磁碟區的區段失敗時，Amazon DocumentDB 會立即修復該區段。它會使用叢集磁碟區組成之其他磁碟區中的資料，以協助確保中修復區段的資料是最新的。因此，Amazon DocumentDB 可避免資料遺失，並減少執行point-in-time還原以從執行個體故障中復原的需求。

可存活的快取預備

Amazon DocumentDB 會在與資料庫不同的程序中管理其頁面快取，讓頁面快取可以獨立於資料庫。在極少的情況下，若資料庫發生故障，頁面快取仍會留在記憶體中。這可確保在資料庫重新啟動時緩衝集區仍是最新狀態。

損毀復原

Amazon DocumentDB 旨在從損毀中幾乎立即復原，並繼續提供應用程式資料。Amazon DocumentDB 會在平行執行緒上以非同步方式執行當機復原，讓您的資料庫在當機後幾乎可立即開啟和使用。

資源控管

Amazon DocumentDB 會保護關鍵程序的資源，例如運作狀態檢查。當執行個體遇到高記憶體壓力時，Amazon DocumentDB 會調節請求。有些操作可能會排入佇列，直到記憶體壓力下降為止。如果記憶體壓力持續，排入佇列的操作可能會逾時。

使用下列 CloudWatch 指標監控低記憶體限流：`LowMemThrottleQueueDepth`、`LowMemThrottleMaxQueueDepth`、`LowMemNumOperationsThro`
如需詳細資訊，請參閱使用 CloudWatch 監控 Amazon DocumentDB。

如果您看到持續的記憶體壓力，請擴展執行個體，為您的工作負載提供額外的記憶體。

讀取偏好設定選項

Amazon DocumentDB 使用雲端原生共用儲存服務，跨三個可用區域複寫六次資料，以提供高等級的耐用性。Amazon DocumentDB 不依賴將資料複寫至多個執行個體以達到耐久性。無論您的叢集資料包含的是單一執行個體或 15 個執行個體，它都具有耐用性。

主題

- 寫入耐久性
- 讀取隔離

- 讀取一致性
- 高可用性
- 擴展讀取

寫入耐久性

Amazon DocumentDB 使用唯一、分散式、容錯、自我修復的儲存系統。此系統跨三個 AWS 可用區域複寫六份資料 (V=6)，以提供高可用性和耐用性。寫入資料時，Amazon DocumentDB 會確保在大多數節點上持續記錄所有寫入，然後再確認寫入用戶端。如果您執行的是三節點 MongoDB 複本集，使用的寫入考量`{w:3, j:true}`會在與 Amazon DocumentDB 比較時產生最佳的可能組態。

Amazon DocumentDB 叢集的寫入必須由叢集的寫入器執行個體處理。嘗試寫入讀取器會導致錯誤。來自 Amazon DocumentDB 主要執行個體的已確認寫入是耐用的，無法復原。Amazon DocumentDB 預設具有高耐用性，不支援非耐久寫入選項。您無法修改耐用性層級 (即寫入考量)。Amazon DocumentDB 會忽略 w=任何內容，且實際上是 w：3 和 j： true。您無法減少它。

由於儲存和運算在 Amazon DocumentDB 架構中是分開的，因此具有單一執行個體的叢集非常耐用。耐用性會在儲存層處理。因此，具有單一執行個體和具有三個執行個體的 Amazon DocumentDB 叢集可達到相同等級的耐用性。您可以設定您的叢集，以適用於特定使用案例，同時又可為您的資料提供高耐用性。

在單一文件中寫入 Amazon DocumentDB 叢集是原子的。

Amazon DocumentDB 不支援 `wtimeout`選項，如果指定了值，則不會傳回錯誤。對主要 Amazon DocumentDB 執行個體的寫入保證不會無限期封鎖。

讀取隔離

從 Amazon DocumentDB 執行個體讀取的資料只會在查詢開始之前傳回耐久的資料。讀取永遠不會傳回在查詢開始執行後修改的資料，在任何情況下。也不可能會發生已變更讀取。

讀取一致性

從 Amazon DocumentDB 叢集讀取的資料很耐用，不會復原。您可以指定請求或連線的讀取偏好設定，以修改 Amazon DocumentDB 讀取的讀取一致性。Amazon DocumentDB 不支援非耐久讀取選項。

Amazon DocumentDB 叢集主要執行個體的讀取在正常操作條件下非常一致，且具有read-after-write一致性。如果在寫入和後續讀取之間發生容錯移轉事件，系統可以短暫傳回非極其一致的讀取。所有來自僅供讀取複本的讀取最終皆會一致，並以相同順序傳回資料，複本延遲通常小於 100 毫秒。

Amazon DocumentDB 讀取偏好設定

只有在複本集模式下從叢集端點讀取資料時，Amazon DocumentDB 才支援設定讀取偏好設定選項。設定讀取偏好設定選項會影響 MongoDB 用戶端或驅動程式將讀取請求路由到 Amazon DocumentDB 叢集中執行個體的方式。您可以為特定查詢設定讀取偏好設定選項，或將其設為 MongoDB 驅動程式的一般選項。(請參閱用戶端或驅動程式的文件，以取得如何設定讀取偏好設定選項。)

如果您的用戶端或驅動程式未在複本集模式下連線至 Amazon DocumentDB 叢集端點，則不會定義指定讀取偏好設定的結果。

Amazon DocumentDB 不支援將標籤集設定為讀取偏好設定。

支援的讀取偏好設定選項

- **primary**- 指定primary讀取偏好設定有助於確保所有讀取都路由到叢集的主要執行個體。如果主要執行個體不可用時，讀取操作會失敗。primary 讀取偏好設定會產生先寫後讀一致性，且適用於以下使用案例：將先寫後讀一致性的優先順序排在高可用性和讀取擴展之前。

  以下範例指定 primary 讀取偏好設定：

  ```
  db.example.find().readPref('primary')
  ```

- **primaryPreferred**- 在正常操作下指定primaryPreferred讀取偏好設定路由讀取至主要執行個體。如果是主要容錯移轉，用戶端會將請求路由到複本。primaryPreferred 讀取偏好會在一般操作期間提供先寫後讀一致性，並在容錯移轉事件期間提供最終一致讀取。primaryPreferred 讀取偏好設定適用於以下使用案例：將先寫後讀一致性的優先順序排在讀取擴展之前 (但仍需要高可用性)。

  以下範例指定 primaryPreferred 讀取偏好設定：

  ```
  db.example.find().readPref('primaryPreferred')
  ```

- **secondary**- 指定secondary讀取偏好設定可確保讀取只會路由至複本，而不會路由至主要執行個體。如果叢集中沒有複本執行個體，讀取請求會失敗。secondary 讀取偏好設定會產生最終一致讀取，且適用於以下使用案例：將主要執行個體寫入傳輸量的優先順序排在高可用性和先寫後讀一致性之前。

以下範例指定 secondary 讀取偏好設定：

```
db.example.find().readPref('secondary')
```

- **secondaryPreferred**- 指定secondaryPreferred讀取偏好設定可確保在一或多個複本處於作用中狀態時，讀取會路由至僅供讀取複本。如果沒有叢集中沒有作用中的複本執行個體，系統會將讀取請求路由到主要執行個體。secondaryPreferred 讀取偏好會在僅供讀取複本提供讀取服務時，提供最終一致讀取。它會在主要執行個體提供讀取服務時，提供先寫後讀一致性 (除了容錯移轉事件以外)。secondaryPreferred 讀取偏好設定適用於以下使用案例：將先寫後讀一致性的優先順序排在高可用性和讀取擴展之前。

  以下範例指定 secondaryPreferred 讀取偏好設定：

  ```
  db.example.find().readPref('secondaryPreferred')
  ```

- **nearest**- 指定nearest讀取偏好設定路由只會根據用戶端與 Amazon DocumentDB 叢集中所有執行個體之間的測量延遲進行讀取。nearest 讀取偏好會在僅供讀取複本提供讀取服務時，提供最終一致讀取。它會在主要執行個體提供讀取服務時，提供先寫後讀一致性 (除了容錯移轉事件以外)。nearest 讀取偏好設定適用於以下使用案例：將達到最低可能讀取延遲和高可用性的優先順序排在先寫後讀一致性和讀取擴展之前。

  以下範例指定 nearest 讀取偏好設定：

  ```
  db.example.find().readPref('nearest')
  ```

高可用性

Amazon DocumentDB 使用複本做為主要執行個體的容錯移轉目標，以支援高可用性叢集組態。如果主要執行個體失敗，Amazon DocumentDB 複本會提升為新的主要執行個體，並短暫中斷，在此期間對主要執行個體發出的讀取和寫入請求會失敗，但有例外。

如果您的 Amazon DocumentDB 叢集不包含任何複本，則會在失敗期間重新建立主要執行個體。由於提升 Amazon DocumentDB 複本比重新建立主要執行個體快得多，因此請建立一或多個 Amazon DocumentDB 複本做為容錯移轉目標。

做為容錯移轉目標的複本執行個體類別應與主要執行個體的執行個體類別相同。它們應該佈建在與主要執行個體不同的可用區域。您可以控制哪些複本是偏好做為容錯移轉目標。如需設定 Amazon DocumentDB 以獲得高可用性的最佳實務，請參閱 了解 Amazon DocumentDB 叢集容錯能力。

擴展讀取

Amazon DocumentDB 複本非常適合用於讀取擴展。他們完全專用於叢集磁碟區上的讀取操作，也就是說，複本並不會處理寫入。資料複寫發生於叢集磁碟區內，而非執行個體之間。因此，每個複本的資源都專用於處理您的查詢，而非複寫和寫入資料。

如果您的應用程式需要讀取容量，您可以快速將複本新增到您的叢集 (通常少於十分鐘)。如果您的讀取容量需求減少，您可以移除不需要的複本。使用 Amazon DocumentDB 複本時，您只需為所需的讀取容量付費。

Amazon DocumentDB 透過使用讀取偏好設定選項支援用戶端讀取擴展。如需詳細資訊，請參閱Amazon DocumentDB 讀取偏好設定。

TTL 刪除

透過背景程序從 TTL 索引刪除是最佳作法，但無法保證可在特定時間範圍內完成。像是執行個體大小、執行個體資源使用率、文件大小和整體輸送量等因素，都會影響 TTL 刪除的時間。

當 TTL 監控器刪除文件時，每個刪除都會產生 IO 成本，使帳單金額增加。如果輸送量和 TTL 刪除率增加，由於 IO 用量增加，您應該預期帳單會增加。

當您在現有集合上建立 TTL 索引時，您必須先刪除所有過期的文件，再建立索引。目前的 TTL 實作已針對刪除集合中的一小部分文件進行最佳化，這通常是從一開始在集合上啟用 TTL 的情況，而且如果需要一次刪除大量文件，可能會導致高於必要的 IOPS。

如果您不想建立 TTL 索引來刪除文件，您可以改為根據時間將文件分割為集合，並在不再需要文件時直接捨棄這些集合。例如：您可以每週建立一個集合並捨棄它，而不會產生 IO 成本。這比使用 TTL 索引更具成本效益。

計費資源

識別計費的 Amazon DocumentDB 資源

作為全受管資料庫服務，Amazon DocumentDB 會針對執行個體、儲存體、I/O、備份和資料傳輸收取費用。如需詳細資訊，請參閱 Amazon DocumentDB （與 MongoDB 相容） 定價。

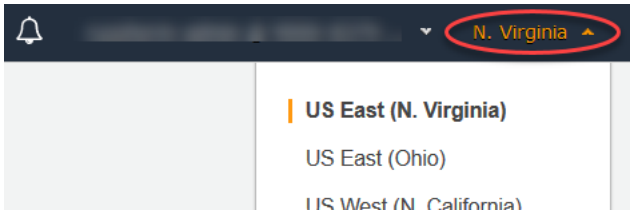
若要探索帳戶中的計費資源，並可能刪除資源，您可以使用 AWS 管理主控台 或 AWS CLI。

使用 AWS 管理主控台

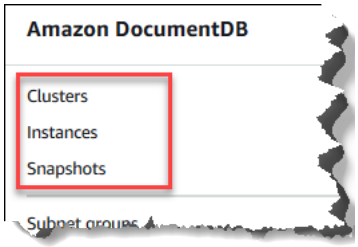
使用 AWS 管理主控台，您可以探索已為指定 佈建的 Amazon DocumentDB 叢集、執行個體和快照 AWS 區域。

找出叢集、執行個體和快照

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 若要探索預設區域以外的區域中的計費資源，請在畫面右上角選擇要 AWS 區域 搜尋的 。



3. 在導覽窗格中，選擇您感興趣的應計費資源類型：Clusters (叢集)、Instances (執行個體) 或 Snapshots (快照)。



4. 右窗格會列出您針對該區域佈建的所有叢集、執行個體或快照。您需要支付叢集、執行個體和快照的費用。

使用 AWS CLI

使用 AWS CLI，您可以探索已為指定 佈建的 Amazon DocumentDB 叢集、執行個體和快照 AWS 區域。

找出叢集和執行個體

以下程式碼會列出指定區域的所有叢集和執行個體。如果想要搜尋預設區域中的叢集和執行個體，您可以省略 `--region` 參數。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters \
    --region us-east-1 \
    --query 'DBClusters[?Engine==`docdb`]' | \
       grep -e "DBClusterIdentifier" -e "DBInstanceIdentifier"
```

針對 Windows：

```
aws docdb describe-db-clusters ^
    --region us-east-1 ^
    --query 'DBClusters[?Engine==`docdb`]' | ^
       grep -e "DBClusterIdentifier" -e "DBInstanceIdentifier"
```

此操作的輸出將會如下所示。

```
"DBClusterIdentifier": "docdb-2019-01-09-23-55-38",
        "DBInstanceIdentifier": "docdb-2019-01-09-23-55-38",
        "DBInstanceIdentifier": "docdb-2019-01-09-23-55-382",
"DBClusterIdentifier": "sample-cluster",
"DBClusterIdentifier": "sample-cluster2",
```

找出快照

以下程式碼會列出指定區域的所有快照。如果想要搜尋預設區域中的快照，您可以省略 `--region` 參數。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-snapshots \
  --region us-east-1 \
  --query 'DBClusterSnapshots[?Engine==`docdb`].
[DBClusterSnapshotIdentifier,SnapshotType]'
```

針對 Windows：

```
aws docdb describe-db-cluster-snapshots ^
  --region us-east-1 ^
  --query 'DBClusterSnapshots[?Engine==`docdb`].
[DBClusterSnapshotIdentifier,SnapshotType]'
```

此操作的輸出將會如下所示。

```
[
    [
        "rds:docdb-2019-01-09-23-55-38-2019-02-13-00-06",
        "automated"
    ],
    [
        "test-snap",
        "manual"
    ]
]
```

您只需要刪除 manual 快照。當您刪除叢集時，也會刪除 Automated 快照。

刪除不需要的計費資源

若要刪除某個叢集，您必須先刪除該叢集中的所有執行個體。

- 若要刪除執行個體，請參閱刪除 Amazon DocumentDB 執行個體。

> **Important**
>
> 即使您刪除叢集中的執行個體，您仍需要支付使用該叢集相關聯儲存和備份的費用。若要停止所有費用，您必須也要刪除您的叢集和手動快照。

- 若要刪除叢集，請參閱刪除 Amazon DocumentDB 叢集。
- 若要刪除手動快照，請參閱刪除叢集快照。

什麼是文件資料庫？

有些開發人員不是以正規化的列和欄來思考其資料模型。通常在應用程式層級，資料是以 JSON 文件來表示，因為開發人員以文件形式來思考資料模式更為直覺。

文件資料庫的普遍性一直在成長，因為它們可讓您使用和您的應用程式碼相同的文件模型格式，將資料存留於資料庫中。文件資料庫提供功能強大且直覺式的 API 進行靈活彈性開發。

主題

- 文件資料庫使用案例
- 了解文件
- 使用文件

文件資料庫使用案例

您的使用案例會影響您需要的是文件資料庫，或是其他類型的資料庫來管理您的資料。文件資料庫對於需要靈活結構描述以便快速進行重複性開發的工作負載來說非常適合。以下是一些使用案例的範例，文件資料庫可為這些使用案例提供相當明顯的優勢：

主題

- 使用者設定檔
- 即時大數據
- 內容管理

使用者設定檔

由於文件資料庫擁有靈活的結構描述，因此能夠存放擁有不同屬性和資料值的文件。文件資料庫對於不同使用者提供不同類型資訊的線上設定檔來說，是相當實用的解決方案。您可以使用文件資料庫單獨存放每個使用者特有的屬性，藉此有效率地存放每個使用者的設定檔。

假設使用者選擇新增或移除其設定檔中的資訊。在這種情況下，就可輕鬆將其文件取代為更新版本，該版本當中會包含任何最近新增的屬性和資料，或省略任何近期省略的屬性和資料。文件資料庫可輕鬆管理此層級的個別特性和流暢度。

即時大數據

過去，從操作資料擷取資訊的能力受到操作資料庫和分析資料庫在不同環境中維護的阻礙，分別是操作和商業/報告。在競爭激烈的商業環境中，即時擷取營運資訊的能力至關重要。透過使用文件資料庫，企業就能存放和管理任何來源的營運資料，並同時將資料饋送至選擇的 BI 引擎進行分析。但並不需要有兩個環境。

內容管理

為了有效管理內容，您必須能夠收集和彙總各種不同來源的內容，然後將內容傳遞給客戶。文件資料庫採用靈活的結構描述，是收集和存放任何類型資料最理想的選擇。您可以使用文件資料庫來建立和納入新的內容類型，包括使用者產生的內容，例如映像、評論和影片。

了解文件

文件資料庫用於將半結構化資料儲存為文件，而不是將多個資料表的資料標準化，每個資料表都有唯一且固定的結構，就像關聯式資料庫中一樣。存放在文件資料庫中的文件會使用巢狀索引鍵值組提供文件

的結構或結構描述。不過，不同類型的文件可以存放在相同的文件資料庫中，因此符合處理不同格式的類似資料時的需求。例如，由於每個文件是自我描述的，因此文件資料庫中的範例文件主題所述線上商店的 JSON 編碼文件可以存放在相同的文件資料庫中。

主題

- SQL 與非關聯式術語
- 簡易文件
- 內嵌文件
- 文件資料庫中的範例文件
- 了解文件資料庫中的標準化

SQL 與非關聯式術語

下表比較文件資料庫 (MongoDB) 所用的術語與 SQL 資料庫所用的術語。

SQL	MongoDB
資料表	收集
Row	文件
資料行	欄位
主索引鍵	ObjectId
索引	索引
檢視	檢視
巢狀表格或物件	內嵌文件
陣列	陣列

簡易文件

文件資料庫中的所有文件都是自我描述。本文採用類似 JSON 格式的文件，但您可以使用其他編碼方式。

簡易文件有一或多個欄位，這些欄位在文件中全都位於同一層級。在下列範例中，欄位 SSN、LName、FName、DOB、Street、City、State-Province、PostalCode 和 Country 在文件中全都是同級。

```
{
   "SSN": "123-45-6789",
   "LName": "Rivera",
   "FName": "Martha",
   "DOB": "1992-11-16",
   "Street": "125 Main St.",
   "City": "Anytown",
   "State-Province": "WA",
   "PostalCode": "98117",
   "Country": "USA"
}
```

將資訊整理到簡易文件中時，每個欄位都會分別管理。若要擷取人員的地址，您必須擷取 Street、City、State-Province、PostalCode 和 Country 做為個別資料項目。

內嵌文件

複雜文件會透過在文件中建立內嵌文件的方式組織其資料。內嵌文件有助於管理分組資料和個別資料項目，端看在所指案例中哪一種方式較有效率。使用上述範例可在主文件中內嵌 Address 文件。這樣做會產生以下文件結構：

```
{
   "SSN": "123-45-6789",
   "LName": "Rivera",
   "FName": "Martha",
   "DOB": "1992-11-16",
   "Address":
   {
       "Street": "125 Main St.",
       "City": "Anytown",
       "State-Province": "WA",
       "PostalCode": "98117",
       "Country": "USA"
   }
}
```

您現在可以以個別欄位 "SSN": ()、內嵌文件 "Address": () 或內嵌文件 () "Address": {"Street":} 的成員身分存取文件中的資料。

文件資料庫中的範例文件

如前面所述，由於文件資料庫中每個文件都是自我描述，因此文件資料庫內文件的結構可能各有不同。以下兩個文件，一個用於書籍，另一個用於期刊，兩者結構有所不同。然而兩個文件可以存放在相同文件資料庫中。

以下是範例書籍文件：

```
{
    "_id" : "9876543210123",
    "Type": "book",
    "ISBN": "987-6-543-21012-3",
    "Author":
    {
        "LName":"Roe",
        "MI": "T",
        "FName": "Richard"
    },
    "Title": "Understanding Document Databases"
}
```

以下是有兩篇文章的範例期刊文件：

```
{
    "_id" : "0123456789012",
    "Publication": "Programming Today",
    "Issue":
    {
        "Volume": "14",
        "Number": "09"
    },
    "Articles" : [
        {
            "Title": "Is a Document Database Your Best Solution?",
            "Author":
            {
                "LName": "Major",
                "FName": "Mary"
            }
        },
        {
            "Title": "Databases for Online Solutions",
            "Author":
```

```
            {
                "LName": "Stiles",
                "FName": "John"
            }
        }
    ],
    "Type": "periodical"
}
```

比較這兩個文件的結構。使用關聯式資料庫時，您需要使用個別的「期刊」和「書籍」表格，或是單一表格，其中未使用的欄位如「出版品」、「議題」、「文章」和「MI」為 null 值。由於文件資料庫為半結構化，每個文件都會定義自己的結構，因此這兩個文件在沒有 null 欄位的情況下，可並存於相同文件資料庫中。文件資料庫適合用於處理稀疏資料。

依照文件資料庫進行開發，即可進行快速、重複性的開發。這是因為您可以動態變更文件的資料結構，而不必變更整個集合的結構描述。文件資料庫非常適合靈活的開發工作和動態變化的環境。

了解文件資料庫中的標準化

文件資料庫不會正規化；在某一個文件中找到的資料可在另一個文件中重複找到。此外，文件之間可能存在一些資料差異。例如，假設您在線上商店購買的情境，而您購買的所有詳細資訊都儲存在單一文件中。此文件可能看起來會像下面的 JSON 文件：

```
{
    "DateTime": "2018-08-15T12:13:10Z",
    "LName" : "Santos",
    "FName" : "Paul",
    "Cart" : [
        {
            "ItemId" : "9876543210123",
            "Description" : "Understanding Document Databases",
            "Price" : "29.95"
        },
        {
            "ItemId" : "0123456789012",
            "Description" : "Programming Today",
            "Issue": {
                "Volume": "14",
                "Number": "09"
            },
            "Price" : "8.95"
        },
```

```
        {
            "ItemId": "234567890-K",
            "Description": "Gel Pen (black)",
            "Price": "2.49"
        }
    ],
    "PaymentMethod" :
    {
        "Issuer" : "MasterCard",
        "Number" : "1234-5678-9012-3456"
    },
    "ShopperId" : "1234567890"
}
```

這些資訊全都會做為文件儲存到交易集合中。之後您發現自己忘記購買某一個項目。因此，您再次登入同一家商店並再次購買，這次購買也會存放為交易集合中的另一個文件。

```
{
    "DateTime": "2018-08-15T14:49:00Z",
    "LName" : "Santos",
    "FName" : "Paul",
    "Cart" : [
        {
            "ItemId" : "2109876543210",
            "Description" : "Document Databases for Fun and Profit",
            "Price" : "45.95"
        }
    ],
    "PaymentMethod" :
    {
        "Issuer" : "Visa",
        "Number" : "0987-6543-2109-8765"
    },
    "ShopperId" : "1234567890"
}
```

請注意這兩個文件之間的備援：您的姓名和購物者 ID （如果您使用相同的信用卡，則為您的信用卡資訊）。但是，這種情況並無不妥，因為儲存空間並不昂貴，而每個文件會完整記錄單一交易，可利用簡單的索引鍵值查詢快速擷取。不需聯結。

這兩個文件之間也有明顯的差異：您的信用卡資訊。這只是一項明顯的差異，因為您可能每次購買都使用不同的信用卡。每個文件對於它所記載的交易而言都是正確無誤的。

使用文件

作為文件資料庫，Amazon DocumentDB 可讓您輕鬆儲存、查詢和索引 JSON 資料。在 Amazon DocumentDB 中，集合類似於關聯式資料庫中的資料表，但不會在所有文件上強制執行單一結構描述。集合可讓您將類似的文件分組，同時將它們全都保存在同一個資料庫中，而不要求文件的結構一致。

若使用前面各節的範例文件，您可能會有 `reading_material` 和 `office_supplies` 的集合。您的軟體需負責強制執行文件屬於哪一個集合。

以下範例使用 MongoDB API 顯示如何新增、查詢、更新和刪除文件。

主題

- 新增文件
- 查詢文件
- 更新文件
- 刪除文件

新增文件

在 Amazon DocumentDB 中，資料庫會在您第一次將文件新增至集合時建立。在此範例中，您在 `test` 資料庫中建立名為 `example` 的集合，這是當您連接到叢集時的預設資料庫。由於集合是在插入第一個文件時隱含建立的，因此集合名稱沒有錯誤檢查。因此，系列名稱中的錯字 (例如，`eexample` 而不是 `example`) 會建立文件，並將文件新增至 `eexample` 集合，而不是預期的集合。錯誤檢查必須由您的應用程式處理。

以下範例使用 MongoDB API 來新增文件。

主題

- 新增單一文件
- 新增多個文件

新增單一文件

若要新增單一文件至集合，請對您要新增至集合中的文件使用 `insertOne( {} )` 操作。

```
db.example.insertOne(
    {
```

```
        "Item": "Ruler",
        "Colors": ["Red","Green","Blue","Clear","Yellow"],
        "Inventory": {
            "OnHand": 47,
            "MinOnHand": 40
        },
        "UnitPrice": 0.89
    }
)
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "acknowledged" : true,
    "insertedId" : ObjectId("5bedafbcf65ff161707de24f")
}
```

新增多個文件

若要新增多個文件至集合，請對您要新增至集合中的文件清單使用 insertMany([{},...,{}]) 操作。雖然這份特定清單中的文件有不同的結構描述，但是全都可以新增至相同的集合。

```
db.example.insertMany(
    [
        {
            "Item": "Pen",
            "Colors": ["Red","Green","Blue","Black"],
            "Inventory": {
                "OnHand": 244,
                "MinOnHand": 72
            }
        },
        {
            "Item": "Poster Paint",
            "Colors": ["Red","Green","Blue","Black","White"],
            "Inventory": {
                "OnHand": 47,
                "MinOnHand": 50
            }
        },
        {
            "Item": "Spray Paint",
            "Colors": ["Black","Red","Green","Blue"],
```

```
            "Inventory": {
                "OnHand": 47,
                "MinOnHand": 50,
                "OrderQnty": 36
            }
        }
    ]
)
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "acknowledged" : true,
    "insertedIds" : [
            ObjectId("5bedb07941ca8d9198f5934c"),
            ObjectId("5bedb07941ca8d9198f5934d"),
            ObjectId("5bedb07941ca8d9198f5934e")
    ]
}
```

查詢文件

有時候，您可能需要查詢線上商店的庫存，讓客戶可以查看和購買您銷售的商品。查詢集合相當容易，無論您需要的是集合中的所有文件，或僅需要滿足特定條件的文件。

若要查詢文件，請使用 `find()` 操作。`find()` 命令擁有單一文件參數，負責定義選擇要傳回的文件時所使用的條件。`find()` 輸出的文件格式會是單一行文字且沒有換行。若要將輸出文件格式化，以更方便閱讀，請使用 `find().pretty()`。這個主題的所有範例都使用 `.pretty()` 設定輸出格式。

使用您在上述兩個練習中插入`example`集合的四個文件： `insertOne()`和 `insertMany()`。

主題

- 擷取集合中的所有文件
- 擷取符合欄位值的文件
- 擷取符合內嵌文件的文件
- 擷取符合內嵌文件中欄位值的文件
- 擷取符合陣列的文件
- 擷取符合陣列中值的文件
- 使用運算子擷取文件

擷取集合中的所有文件

若要擷取集合中的所有文件，請使用空的查詢文件使用 `find()` 操作。

以下查詢會傳回 `example` 集合中的所有文件。

```
db.example.find( {} ).pretty()
```

擷取符合欄位值的文件

若要擷取符合某個欄位和值的所有文件，請使用識別要符合的欄位和值的查詢文件使用 `find()` 操作。

使用先前文件，此查詢會傳回所有文件，其中「項目」欄位等於「筆」。

```
db.example.find( { "Item": "Pen" } ).pretty()
```

擷取符合內嵌文件的文件

若要尋找符合嵌入文件的所有文件，請使用 `find()` 操作，其附帶的查詢文件會指定內嵌文件名稱和該內嵌文件的所有欄位和值。

當符合嵌入文件，文件的內嵌文件必須擁有與查詢相同的名稱。此外，內嵌文件的欄位和值必須符合查詢。

以下查詢只會傳回「Poster Paint」(廣告顏料) 文件。這是因為「Pen」(筆) 的 `OnHand` 和 `MinOnHand` 有不同值，而「Spray Paint」(噴漆) 比查詢文件多一個欄位 (`OrderQnty`)。

```
db.example.find({"Inventory": {
    "OnHand": 47,
    "MinOnHand": 50 } } ).pretty()
```

擷取符合內嵌文件中欄位值的文件

若要尋找符合嵌入文件的所有文件，請使用 `find()` 操作，其附帶的查詢文件會指定內嵌文件名稱和該內嵌文件的所有欄位和值。

鑒於前述文件，以下查詢使用「點表示法」指定內嵌文件和關注欄位。符合這些的任何文件將會傳回，無論內嵌文件中可能有哪些其他欄位。查詢將傳回「廣告顏料」和「噴漆」，因為它們都符合指定的欄位和值。

```
db.example.find({"Inventory.OnHand": 47, "Inventory.MinOnHand": 50 }).pretty()
```

擷取符合陣列的文件

若要尋找符合陣列的所有文件，請使用 find() 操作與您感興趣的陣列名稱，以及在該陣列中的所有值。查詢將傳回所有具有該名稱的文件，其中陣列值與查詢中值及順序相同。

以下查詢只會傳回「筆」，因為「廣告顏料」有額外的顏色 (白色)，而「噴漆」有不同順序的顏色。

```
db.example.find( { "Colors": ["Red","Green","Blue","Black"] } ).pretty()
```

擷取符合陣列中值的文件

若要尋找具有特殊陣列值的所有文件，請使用 find() 操作與您感興趣的陣列名稱和值。

```
db.example.find( { "Colors": "Red" } ).pretty()
```

前述操作會傳回全部三個文件，因為每個文件都有一個名為 Colors 的陣列，且陣列中某處具有「Red」值。如果您指定數值「White」，查詢只會傳回「廣告顏料」。

使用運算子擷取文件

以下查詢會傳回其中「Inventory.OnHand」值小於 50 的所有文件。

```
db.example.find(
        { "Inventory.OnHand": { $lt: 50 } } )
```

如需支援的查詢運算子清單的詳細資訊，請參閱查詢和投影運算子。

更新文件

您的文件通常不是靜態的，而且會做為應用程式工作流程的一部分進行更新。以下範例說明一些可更新文件的方式。

若要更新現有文件，請使用 update() 操作。update() 操作有兩個文件參數。第一份文件識別要進行更新的文件。第二份文件指定要進行的更新。

當您更新現有欄位時，無論該欄位是簡單欄位、陣列或內嵌文件，您都可以指定欄位名稱及其值。操作到最後，舊文件的欄位已經由新欄位和值取代。

主題

- 更新現有欄位的值
- 新增欄位
- 取代內嵌文件
- 將新欄位插入內嵌文件
- 從文件移除欄位
- 從多個文件移除欄位

更新現有欄位的值

使用以下您稍早為下列更新操作新增的四個文件。

```
{
    "Item": "Ruler",
    "Colors": ["Red","Green","Blue","Clear","Yellow"],
    "Inventory": {
        "OnHand": 47,
        "MinOnHand": 40
    },
    "UnitPrice": 0.89
},
{
    "Item": "Pen",
    "Colors": ["Red","Green","Blue","Black"],
    "Inventory": {
        "OnHand": 244,
        "MinOnHand": 72
    }
},
{
    "Item": "Poster Paint",
    "Colors": ["Red","Green","Blue","Black","White"],
    "Inventory": {
        "OnHand": 47,
        "MinOnHand": 50
    }
},
{
    "Item": "Spray Paint",
    "Colors": ["Black","Red","Green","Blue"],
```

```
    "Inventory": {
        "OnHand": 47,
        "MinOnHand": 50,
        "OrderQnty": 36
    }
}
```

更新簡單欄位

若要更新簡單欄位，請搭配使用 update() 和 $set 以指定欄位名稱和新值。以下範例會將 Item 從「Pen」(筆) 變更為「Gel Pen」(簽字筆)。

```
db.example.update(
    { "Item" : "Pen" },
    { $set: { "Item": "Gel Pen" } }
)
```

此操作的結果將會如下所示。

```
{
    "Item": "Gel Pen",
    "Colors": ["Red","Green","Blue","Black"],
    "Inventory": {
        "OnHand": 244,
        "MinOnHand": 72
    }
}
```

更新陣列

以下範例會取代現有顏色陣列和新陣列，其包含 Orange ，並從顏色清單放棄 White。新顏色清單的順序為 update() 操作中所指定的順序。

```
db.example.update(
    { "Item" : "Poster Paint" },
    { $set: { "Colors": ["Red","Green","Blue","Orange","Black"] } }
)
```

此操作的結果將會如下所示。

```
{
```

```
    "Item": "Poster Paint",
    "Colors": ["Red","Green","Blue","Orange","Black"],
    "Inventory": {
        "OnHand": 47,
        "MinOnHand": 50
    }
}
```

新增欄位

若要藉由新增一或多個新的欄位來修改文件，請使用 update() 操作搭配查詢文件，以使用 $set 運算子識別要插入其中的文件及要插入的新欄位和值。

以下範例會將欄位 UnitPrice 的值 3.99 加入到噴漆文件。請注意，值 3.99 是數字，不是字串。

```
db.example.update(
    { "Item": "Spray Paint" },
    { $set: { "UnitPrice": 3.99 } }
)
```

此操作的結果如下所示 (JSON 格式) 。

```
{
    "Item": "Spray Paint",
    "Colors": ["Black","Red","Green","Blue"],
    "Inventory": {
        "OnHand": 47,
        "MinOnHand": 50,
        "OrderQnty": 36
    },
    "UnitPrice": 3.99
}
```

取代內嵌文件

若要藉由取代內嵌文件來修改文件，請使用 update() 操作搭配文件，以使用 $set 運算子識別內嵌文件及要插入的新欄位和值。

鑒於以下文件。

```
db.example.insert({
    "DocName": "Document 1",
```

```
    "Date": {
        "Year": 1987,
        "Month": 4,
        "Day": 18
    }
})
```

取代內嵌文件

以下範例將目前日期文件取代為新的欄位，其中只包含 Month 和 Day，Year 已刪除。

```
db.example.update(
    { "DocName" : "Document 1" },
    { $set: { "Date": { "Month": 4, "Day": 18 } } }
)
```

此操作的結果將會如下所示。

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18
    }
}
```

將新欄位插入內嵌文件

將新欄位加入至內嵌文件

若要藉由新增一或多個新欄位至內嵌文件來修改文件，請搭配文件使用 update() 操作，其識別內嵌文件及「點表示法」，以使用 $set 運算子指定內嵌文件和要插入的新欄位和值。

鑒於以下文件，以下程式碼使用「點表示法」將 Year 和 DoW 欄位插入到內嵌的 Date 文件，並將 Words 插入到父文件。

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18
    }
```

```
}
```

```
db.example.update(
    { "DocName" : "Document 1" },
    { $set: { "Date.Year": 1987,
              "Date.DoW": "Saturday",
              "Words": 2482 } }
)
```

此操作的結果將會如下所示。

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DoW": "Saturday"
    },
    "Words": 2482
}
```

從文件移除欄位

若要藉由移除文件中的欄位來修改文件，請使用 update() 操作搭配查詢文件，以使用 $unset 運算子識別要從中移除欄位的文件，以指定要移除的欄位。

下列範例會從前述文件中移除 Words 欄位。

```
db.example.update(
    { "DocName" : "Document 1" },
    { $unset: { Words:1 } }
)
```

此操作的結果將會如下所示。

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
```

```
        "Year": 1987,
        "DoW": "Saturday"
    }
}
```

從多個文件移除欄位

若要從多個文件中移除欄位來修改文件，請使用 $unset 操作搭配 multi 運算子，並將 update() 選項設定為 true。

以下範例會從範例集合的所有文件中移除 Inventory 欄位。如果文件沒有 Inventory 欄位，則不會對該文件執行任何動作。如果省略 multi: true，則只會對符合條件的第一個文件執行動作。

```
db.example.update(
    {},
    { $unset: { Inventory:1 } },
    { multi: true }
)
```

刪除文件

若要從資料庫中移除文件，請使用 remove() 操作，指定要移除的文件。以下程式碼會從您的 example 集合移除「簽字筆」。

```
db.example.remove( { "Item": "Gel Pen" } )
```

若要從資料庫中移除所有文件，請使用 remove()操作搭配空白查詢。

```
db.example.remove( { } )
```

開始使用 Amazon DocumentDB

連線和開始使用 Amazon DocumentDB 的方法有很多。本指南是使用者開始使用我們強大文件資料庫最快速、最簡單且最簡單的方法。本指南使用 直接從 AWS CloudShell連接和查詢 Amazon DocumentDB 叢集 AWS 管理主控台。符合 AWS 免費方案資格的新客戶可免費使用 Amazon DocumentDB 和 CloudShell。如果您的 AWS CloudShell 環境或 Amazon DocumentDB 叢集使用免費方案以外的資源，則會向您收取這些資源的正常 AWS 費率。本指南將協助您在五分鐘內開始使用 Amazon DocumentDB。

> **Note**
>
> 本指南中的指示專門用於建立和連線至 AWS CloudShell 可使用 Amazon DocumentDB 和 的 Amazon DocumentDB 執行個體型叢集。
>
> - 如果您想要建立並連線至 Amazon DocumentDB 彈性叢集，請參閱 Amazon DocumentDB 彈性叢集入門。
> - 如果您位於 AWS 中國區域，請參閱 自動連接 Amazon EC2。

主題

- 先決條件
- 步驟 1：建立叢集
- 步驟 2：連線至您的叢集
- 步驟 3：插入和查詢資料
- 步驟 4：探索
- 使用 的 Amazon DocumentDB 快速入門 CloudFormation

先決條件

建立第一個 Amazon DocumentDB 叢集之前，您必須執行下列動作：

建立 Amazon Web Services (AWS) 帳戶

您必須先擁有 Amazon Web Services () 帳戶，才能開始使用 Amazon DocumentDB。AWS AWS 該帳戶是免費的。您僅需按使用的服務和資源量付費。

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

   當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

設定 needed AWS Identity and Access Management (IAM) 許可。

管理 Amazon DocumentDB 資源的存取權，例如叢集、執行個體和叢集參數群組，需要 AWS 可用於驗證請求的登入資料。如需詳細資訊，請參閱Amazon DocumentDB 的 Identity and Access Management。

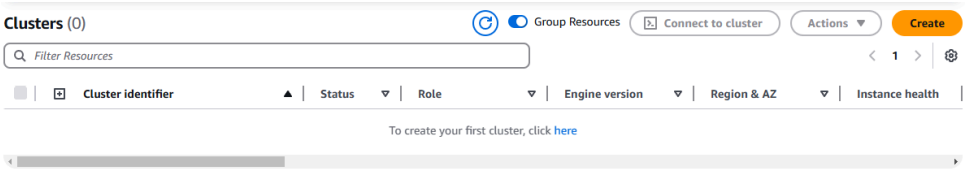
1. 在 的搜尋列中 AWS 管理主控台，在出現的下拉式選單中輸入**IAM**並選取 IAM。
2. 進入 IAM 主控台後，請從導覽窗格中選取使用者。
3. 選取您的使用者名稱。
4. 選擇新增許可。
5. 選取直接連接政策。
6. 在搜尋列`AmazonDocDBFullAccess`中輸入 ，並在搜尋結果中出現時選取它。
7. 選擇下一步。
8. 選擇新增許可。

> **Note**
>
> AWS 您的帳戶在每個區域中包含預設 VPC。如果您選擇使用 Amazon VPC，請完成《Amazon VPC 使用者指南》中建立 Amazon VPC 主題的步驟。

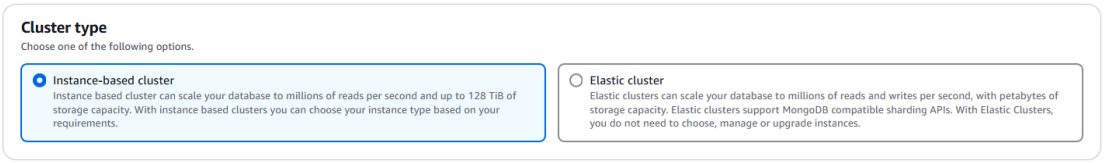
步驟 1：建立叢集

在此步驟中，您將建立 Amazon DocumentDB 叢集。

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在 Amazon DocumentDB 管理主控台的叢集下，選擇建立。



3. 在建立 Amazon DocumentDB 叢集頁面上的叢集類型區段中，選擇執行個體型叢集（這是預設選項）。

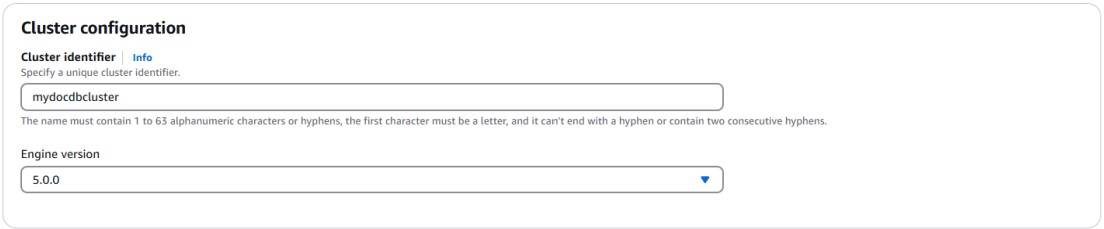


> **Note**
>
> 此類別中的另一個選項是彈性叢集。若要進一步了解 Amazon DocumentDB 彈性叢集，請參閱 使用 Amazon DocumentDB 彈性叢集

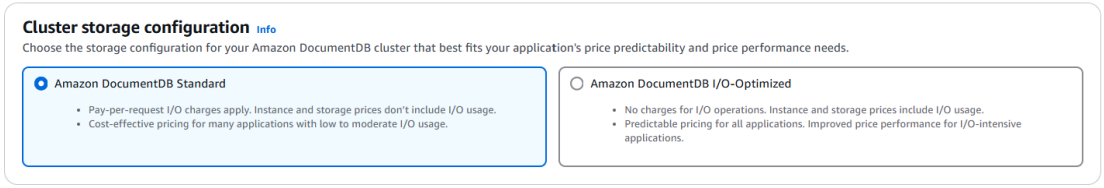
4. 在叢集組態區段中：

   a. 針對叢集識別符，輸入唯一名稱，例如 **`mydocdbcluster`**。請注意，無論如何輸入叢集名稱，主控台都會將所有叢集名稱變更為小寫。

   b. 針對引擎版本，選擇 5.0.0。



5. 在叢集儲存組態區段中，選擇 Amazon DocumentDB Standard（這是預設選項）。



> **Note**
>
> 此類別中的另一個選項是 Amazon DocumentDB I/O 最佳化。若要進一步了解任一選項，請參閱 Amazon DocumentDB 叢集儲存組態

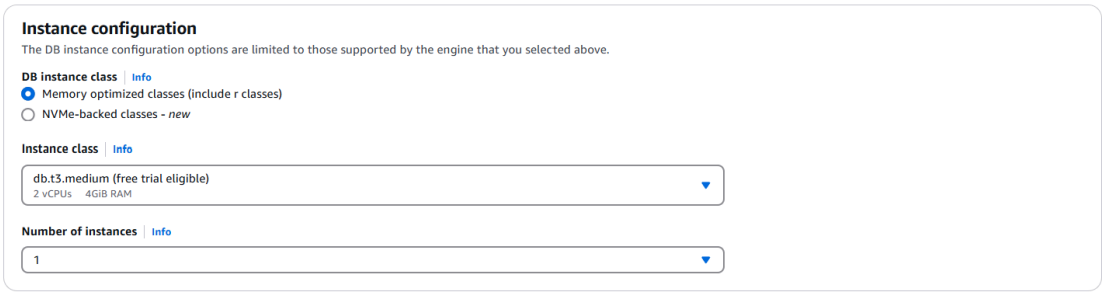
6. 在執行個體組態區段中：

   a. 針對資料庫執行個體類別，選擇記憶體最佳化類別（包括 r 類別）（這是預設值）。

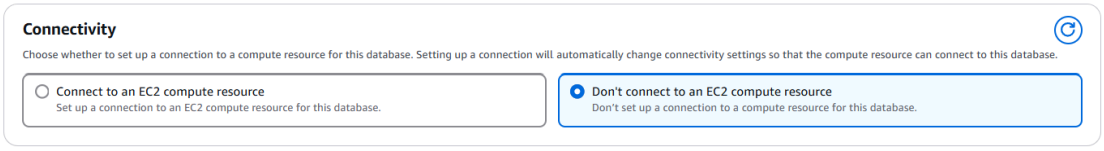
      另一個執行個體選項是 NVMe 支援的類別。如需詳細資訊，請參閱 NVMe 支援的執行個體。

   b. 針對執行個體類別，選擇 db.t3.medium。這符合 AWS 免費試用的資格。

   c. 針對執行個體數量，選擇 1 個執行個體。選擇一個執行個體有助於將成本降至最低。如果這是生產系統，我們建議您為高可用性佈建三個執行個體。

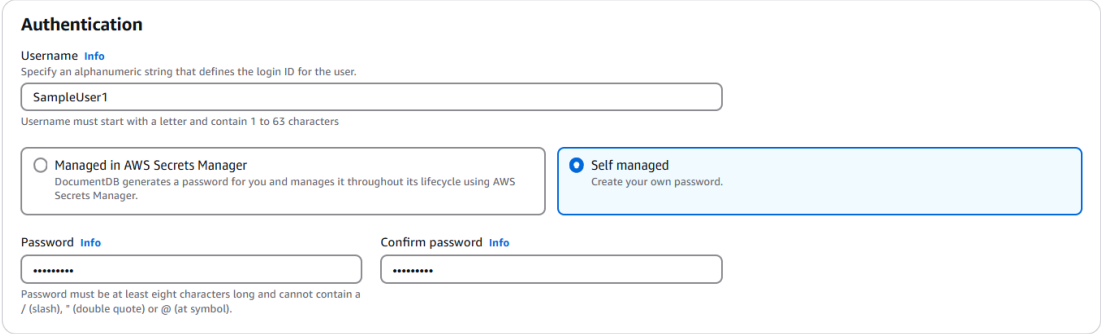


7. 在連線區段中，保留不連線至 EC2 運算資源的預設設定。



8. 在身分驗證區段中，輸入主要使用者的使用者名稱，然後選擇自我管理。輸入密碼，然後進行確認。

   如果您改為選擇 中的受管 AWS Secrets Manager，請參閱 使用 Amazon DocumentDB 和 進行密碼管理 AWS Secrets Manager 以取得詳細資訊。



9. 將所有其他選項保留為預設值，然後選擇建立叢集。

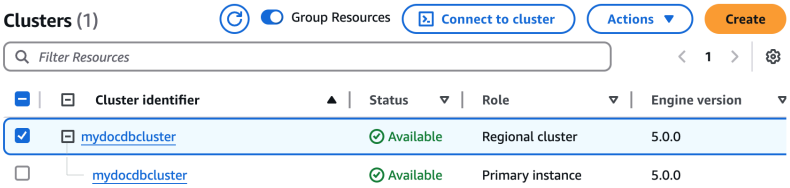
Amazon DocumentDB 現在正在佈建叢集，最多可能需要幾分鐘的時間才能完成。

> **Note**
>
> 如需叢集狀態值的資訊，請參閱監控 Amazon DocumentDB 章節叢集狀態值中的 。

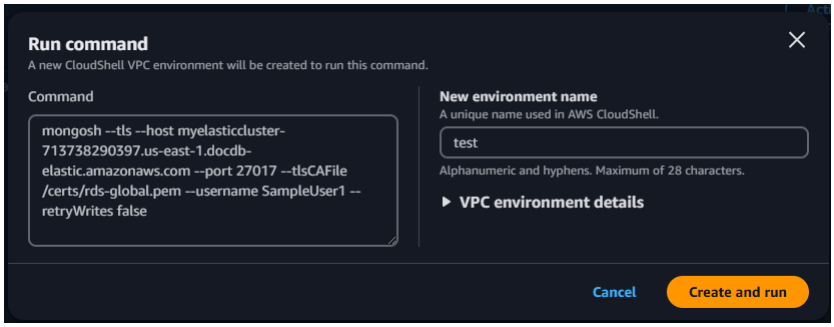
步驟 2：連線至您的叢集

使用 連線至 Amazon DocumentDB 叢集 AWS CloudShell。

1. 在 Amazon DocumentDB 管理主控台的叢集下，找到已建立的叢集。按一下叢集旁的核取方塊，選擇對應叢集。



2. 選擇連線至叢集（位於動作下拉式功能表旁）。只有在您按一下叢集旁的核取方塊，且區域叢集和主要執行個體（兩者）的狀態都顯示為可用之後，才會啟用此按鈕。CloudShell 執行命令畫面隨即顯示。
3. 在新增環境名稱欄位中，輸入唯一名稱 (例如 "test")，然後選擇建立並執行。VPC 環境詳細資訊會自動為您的 Amazon DocumentDB 資料庫設定。



4. 出現提示時，輸入您在步驟 1：建立 Amazon DocumentDB 叢集中建立的密碼（子步驟 7)。



輸入密碼且提示變成 後`rs0 [direct: primary] <env-name>>`，您已成功連線到 Amazon DocumentDB 叢集。

> **Note**
>
> 如需故障診斷的資訊，請參閱故障診斷 Amazon DocumentDB。

步驟 3：插入和查詢資料

現在您已連線至叢集，您可以執行一些查詢，以熟悉使用文件資料庫。

1. 若要插入單一文件，請輸入下列項目：

```
db.collection.insertOne({"hello":"DocumentDB"})
```

您會收到下列輸出：

```
{
  acknowledged: true,
  insertedId: ObjectId('673657216bdf6258466b128c')
```

```
}
```

2. 您可以使用 findOne()命令讀取您撰寫的文件 （因為它只會傳回單一文件）。輸入下列項目：

```
db.collection.findOne()
```

您會收到下列輸出：

```
{ "_id" : ObjectId("5e401fe56056fda7321fbd67"), "hello" : "DocumentDB" }
```

3. 若要執行更多查詢，請考慮遊戲設定檔使用案例。首先，將幾個項目插入標題為 的集合profiles。輸入下列項目：

```
db.profiles.insertMany([{ _id: 1, name: 'Matt', status: 'active', level: 12, score:
 202 },
      { _id: 2, name: 'Frank', status: 'inactive', level: 2, score: 9 },
      { _id: 3, name: 'Karen', status: 'active', level: 7, score: 87 },
      { _id: 4, name: 'Katie', status: 'active', level: 3, score: 27 }
])
```

您會收到下列輸出：

```
{ acknowledged: true, insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4 } }
```

4. 使用 find()命令來傳回設定檔集合中的所有文件。輸入下列項目：

```
db.profiles.find()
```

您會收到與您在步驟 3 中輸入的資料相符的輸出。

5. 使用篩選條件對單一文件使用查詢。輸入下列項目：

```
db.profiles.find({name: "Katie"})
```

您會收到下列輸出：

```
{ "_id" : 4, "name" : "Katie", "status": "active", "level": 3, "score":27}
```

6. 現在，讓我們嘗試尋找設定檔，並使用 findAndModify命令修改它。我們會為使用者 Matt 提供額外 10 點的下列程式碼：

```
db.profiles.findAndModify({
        query: { name: "Matt", status: "active"},
        update: { $inc: { score: 10 } }
    })
```

您會收到下列輸出（請注意，他的分數尚未增加）：

```
{
    [{_id : 1, name : 'Matt', status: 'active', level: 12, score: 202}]
```

7. 您可以使用下列查詢，確認他的分數已變更：

   ```
   db.profiles.find({name: "Matt"})
   ```

   您會收到下列輸出：

   ```
   { "_id" : 1, "name" : "Matt", "status" : "active", "level" : 12, "score" : 212 }
   ```

步驟 4：探索

恭喜您！您已成功完成 Amazon DocumentDB 執行個體型叢集的入門指南。

下一步是什麼？ 了解如何充分利用此資料庫及其一些熱門功能：

- 管理 Amazon DocumentDB
- 擴展
- 備份和還原

> **Note**
>
> 除非您刪除叢集，否則您從此入門練習建立的叢集將繼續產生成本。如需指示，請參閱刪除 Amazon DocumentDB 叢集。

使用 的 Amazon DocumentDB 快速入門 CloudFormation

本節包含步驟和其他資訊，可協助您使用 快速開始使用 Amazon DocumentDB （與 MongoDB 相容）CloudFormation。如需 Amazon DocumentDB 的一般資訊，請參閱 什麼是 Amazon DocumentDB （與 MongoDB 相容）。

這些指示使用 AWS CloudFormation 範本，在您的預設 Amazon VPC 中建立叢集和執行個體。如需自行建立這些資源的說明，請參閱開始使用 Amazon DocumentDB。

> **Important**
>
> 此範本建立的 CloudFormation 堆疊會建立多個資源，包括 Amazon DocumentDB （例如叢集和執行個體） 和 Amazon Elastic Compute Cloud （例如子網路群組） 中的資源。
> 其中部分資源不屬於免費方案資源。如需定價資訊，請參閱 Amazon DocumentDB 定價和 Amazon EC2 定價。當您完成後可以刪除堆疊，以停止計費。

此 CloudFormation 堆疊僅供教學使用。如果您將此範本用於生產環境，請使用更嚴格的 IAM 政策和安全。如需保護資源的詳細資訊，請參閱 Amazon VPC Security 和 Amazon EC2 Network and Security。

主題

- 先決條件
- 啟動 Amazon DocumentDB CloudFormation 堆疊
- 存取 Amazon DocumentDB 叢集
- 終止保護和刪除保護

先決條件

建立 Amazon DocumentDB 叢集之前，您必須具有下列項目：

- 預設 Amazon VPC
- 必要的 IAM 許可

所需的 IAM 許可

以下權限可讓您建立 CloudFormation 堆疊的資源：

AWS 受管政策

- `AWSCloudFormationReadOnlyAccess`
- `AmazonDocDBFullAccess`

其他 IAM 許可

下列政策概述建立和刪除此 CloudFormation 堆疊所需的其他許可。

在下列範例中，將每個`#########`取代為您資源的資訊。

Amazon EC2 金鑰對

您必須在要建立 CloudFormation 堆疊的區域中擁有可用的金鑰對（和 PEM 檔案）。如果您需要建立金鑰對，請參閱《Amazon EC2 使用者指南》中的使用 Amazon EC2 建立金鑰對。 Amazon EC2

啟動 Amazon DocumentDB CloudFormation 堆疊

本節說明如何啟動和設定 Amazon DocumentDB CloudFormation 堆疊。

1. 在 登入 AWS 管理主控台 https://console.aws.amazon.com/。
2. 下表列出每個 的 Amazon DocumentDB 堆疊範本 AWS 區域。針對 AWS 區域 您要啟動堆疊的 ，選擇啟動堆疊。

區域	檢視範本	在設計工具中檢視	啟動
美國東部 (俄亥俄)	檢視範本	在設計工具中檢視	Launch Stack
美國東部 (維吉尼亞北部)	檢視範本	在設計工具中檢視	Launch Stack
美國西部 (奧勒岡)	檢視範本	在設計工具中檢視	Launch Stack
亞太地區 (孟買)	檢視範本	在設計工具中檢視	Launch Stack
亞太地區 (首爾)	檢視範本	在設計工具中檢視	Launch Stack

區域	檢視範本	在設計工具中檢視	啟動
亞太地區 (新加坡)	檢視範本	在設計工具中檢視	Launch Stack
亞太地區 (悉尼)	檢視範本	在設計工具中檢視	Launch Stack
亞太地區 (東京)	檢視範本	在設計工具中檢視	Launch Stack
加拿大 (中部)	檢視範本	在設計工具中檢視	Launch Stack
歐洲 (法蘭克福)	檢視範本	在設計工具中檢視	Launch Stack
歐洲 (愛爾蘭)	檢視範本	在設計工具中檢視	Launch Stack
歐洲 (倫敦)	檢視範本	在設計工具中檢視	Launch Stack
Europe (Paris)	檢視範本	在設計工具中檢視	Launch Stack

3. 建立堆疊 — 描述您選取的 Amazon DocumentDB 範本。每個堆疊都是以範本為基礎：JSON 或 YAML 檔案，其中包含您要包含在堆疊中的 AWS 資源組態。由於您選擇從上述提供的範本啟動堆疊，因此您的範本已設定為為 AWS 區域 您選擇的 建立 Amazon DocumentDB 堆疊。

   當您啟動 CloudFormation 堆疊時，Amazon DocumentDB 叢集的刪除保護預設為停用。如果您要啟用叢集的刪除保護，請完成以下步驟。否則，請選擇 Next (下一步) 以繼續下一個步驟。

   若要啟用 Amazon DocumentDB 叢集的刪除保護：

   1. 從建立堆疊頁面的右下角選擇在設計工具中檢視。
   2. 在主控台產生的 CloudFormation 設計工具頁面中使用整合式 JSON 和 YAML 編輯器修改範本。捲動至 `Resources` 區段並修改它以包含 `DeletionProtection`，如下所示。如需使用 CloudFormation 設計工具的詳細資訊，請參閱什麼是 CloudFormation 設計工具？。

      JSON：

      ```
      "Resources": {
      ```

```
    "DBCluster": {
        "Type": "AWS::DocDB::DBCluster",
        "DeletionPolicy": "Delete",
        "Properties": {
            "DBClusterIdentifier": {
                "Ref": "DBClusterName"
            },
            "MasterUsername": {
                "Ref": "MasterUser"
            },
            "MasterUserPassword": {
                "Ref": "MasterPassword"
            },
            "DeletionProtection": "true"
        }
    },
```

YAML：

```
Resources:
  DBCluster:
    Type: 'AWS::DocDB::DBCluster'
    DeletionPolicy: Delete
    Properties:
      DBClusterIdentifier: !Ref DBClusterName
      MasterUsername: !Ref MasterUser
      MasterUserPassword: !Ref MasterPassword
      DeletionProtection: 'true'
```

3. 選擇頁面左上角的 Create Stack (建立堆疊)

   ( )

   以儲存變更，並在啟用這些變更的情況下建立堆疊。

4. 儲存變更之後，系統會將您重新導向至 Create stack (建立堆疊) 頁面。

5. 選擇 Next (下一步) 繼續。

4. 指定堆疊詳細資訊 — 輸入範本的堆疊名稱和參數。參數是在您的範本中定義，而且可讓您在建立或更新堆疊時輸入自訂值。

   - 在 Stack name (堆疊名稱) 下，輸入堆疊的名稱或接受提供的名稱。堆疊名稱可包含字母 (A—Z 和 a—z)、數字 (0—9) 和破折號 (—)。
   - 在 Parameters (參數) 下，輸入下列詳細資訊：

- DBClusterName — 輸入 Amazon DocumentDB 叢集的名稱，或接受提供的名稱。

  叢集命名限制條件：

  - 長度為 【1-63】 個字母、數字或連字號。
  - 第一個字元必須是字母。
  - 不能以連字號結尾，或包含兩個連續連字號。
  - 對於每個區域每個 Amazon RDS、Neptune 和 Amazon DocumentDB 的所有叢集 AWS 帳戶，必須是唯一的。
- DBInstanceClass — 從下拉式清單中，選取 Amazon DocumentDB 叢集的執行個體類別。
- DBInstanceName — 輸入 Amazon DocumentDB 執行個體的名稱或接受提供的名稱。

  執行個體命名限制：

  - 長度為 【1-63】 個字母、數字或連字號。
  - 第一個字元必須是字母。
  - 不能以連字號結尾，或包含兩個連續連字號。
  - 對於每個區域每個 Amazon RDS、Neptune 和 Amazon DocumentDB 的所有執行個體 AWS 帳戶，必須是唯一的。
- MasterPassword — 資料庫管理員帳戶密碼。
- MasterUser — 資料庫管理員帳戶使用者名稱。MasterUser 必須以字母開頭，且只能包含英數字元。

選擇 Next (下一步) 儲存變更並繼續。

5. 設定堆疊選項 — 設定堆疊的標籤、許可和其他選項。

   - 標籤 — 指定要套用至堆疊中資源的標籤 （ 索引鍵/值 ） 對。您最多可以為每個堆疊新增 50 個唯一的標籤。
   - 許可 — 選用。選擇 IAM 角色以明確定義 CloudFormation 如何在堆疊中建立、修改或刪除資源。如果您未選擇角色， 會根據您的使用者登入資料 CloudFormation 使用許可。在您指定服務角色之前，請確保您有傳遞許可 (`iam:PassRole`)。`iam:PassRole` 許可會指定您可使用哪些角色。

     > **Note**
     >
     > 當您指定服務角色時， CloudFormation 一律會將該角色用於在該堆疊上執行的所有操作。其他具有在此堆疊上執行這些操作之許可的使用者，都能夠使用此角色，即使

他們不具傳遞它的許可。如果該角色包含使用者不該有的許可，您可能在無意中提升使用者的許可。確保角色授予最低權限。

- 進階選項 — 您可以設定下列進階選項：
  - 堆疊政策 — 選用。可藉由此選項來定義堆疊更新期間，您欲避免意外更新的資源。在預設情況下，系統可以在堆疊更新期間將所有資源進行更新。

    您可以直接輸入堆疊政策作為 JSON，或上傳包含堆疊政策的 JSON 檔案。如需詳細資訊，請參閱避免更新堆疊資源。
  - 回復組態 — 選用。指定 CloudWatch Logs 警示 CloudFormation ，以在建立和更新堆疊時監控 。如果操作違反警示閾值， 會 CloudFormation 將其轉返。
  - 通知選項 — 選用。指定簡易通知系統 (SNS) 的主題。
  - 堆疊建立選項 — 選用。您可以指定下列選項：
    - 失敗時轉返 — 如果堆疊建立失敗，是否應該轉返堆疊。
    - 逾時 — 堆疊建立逾時之前的分鐘數。
    - 終止保護 — 防止堆疊意外刪除。

> **Note**
>
> CloudFormation 終止保護與刪除保護的 Amazon DocumentDB 概念不同。如需詳細資訊，請參閱終止保護和刪除保護。

選擇 Next (下一步) 繼續。

6. 檢閱 <stack-name> — 檢閱您的堆疊範本、詳細資訊和組態選項。您也可以在頁面底部開啟快速建立連結，以建立與此相同基本組態的堆疊。
   - 選擇 Create (建立) 以建立堆疊。
   - 或者，您可以選擇 Create change set (建立變更集)。變更集是建立堆疊之前如何配置此堆疊的預覽。這可讓您在執行變更集之前檢查各種模型組態。

存取 Amazon DocumentDB 叢集

CloudFormation 堆疊完成後，您可以使用 Amazon EC2 執行個體連線至 Amazon DocumentDB 叢集。如需有關使用 SSH 連線至 Amazon EC2 執行個體的資訊，請參閱《Amazon EC2 使用者指南》中的連線至您的 Linux 執行個體。

連線後，請參閱下列各節，其中包含使用 Amazon DocumentDB 的相關資訊。

- 步驟 3：插入和查詢資料
- 管理 Amazon DocumentDB 叢集和執行個體
- 在 Amazon DocumentDB 中監控和記錄

終止保護和刪除保護

啟用刪除保護和終止保護是 Amazon DocumentDB 最佳實務。CloudFormation 終止保護是與 Amazon DocumentDB 刪除保護功能截然不同的功能。

- 終止保護 — 您可以啟用 CloudFormation 堆疊的終止保護，以防止堆疊意外刪除。如果使用者嘗試刪除啟用終止保護的堆疊，則刪除會失敗，且堆疊保持不變。當您使用 CloudFormation 建立堆疊時，預設會停用終止保護。您可以在建立新的堆疊時啟用終止保護。如需詳細資訊，請參閱設定 CloudFormation 堆疊選項。
- 刪除保護 — Amazon DocumentDB 也提供為叢集啟用刪除保護的功能。如果使用者嘗試刪除已啟用刪除保護的 Amazon DocumentDB 叢集，刪除會失敗，且叢集保持不變。啟用時，刪除保護可防止意外從 Amazon DocumentDB AWS 管理主控台、 AWS CLI和 CloudFormation 刪除。如需啟用和停用 Amazon DocumentDB 叢集刪除保護的詳細資訊，請參閱 刪除保護。

遷移和升級 Amazon DocumentDB

Amazon DocumentDB 是與 MongoDB API 相容的全受管資料庫服務。您可以使用本節詳述的程序，將資料從內部部署或 Amazon Elastic Compute Cloud (Amazon EC2) 上執行的 MongoDB 資料庫遷移至 Amazon DocumentDB。 MongoDB Amazon EC2

主題

- 使用 AWS Database Migration Service (DMS) 遷移至 Amazon DocumentDB：快速入門指南
- Amazon DocumentDB 遷移 Runbook
- 從 Couchbase Server 遷移
- Amazon DocumentDB 引擎版本支援日期
- Amazon DocumentDB 就地主要版本升級
- Amazon DocumentDB 次要版本升級
- 使用 升級 Amazon DocumentDB 叢集 AWS Database Migration Service
- 使用長期支援 (LTS) 版本
- Amazon DocumentDB 延伸支援

使用 AWS Database Migration Service (DMS) 遷移至 Amazon DocumentDB：快速入門指南

主題

- 準備 DMS 來源
- 設定 DMS
- 啟用 DocumentDB 壓縮
- 建立複寫任務
- 監控進度
- 其他資訊

準備 DMS 來源

請參閱 啟用變更串流 以啟用 DocumentDB 變更串流，或啟用 MongoDB Oplog 以支援 DMS 變更資料擷取 (CDC)。

- DMS 來源必須保留所有持續變更，直到所有包含集合的 DMS 完全載入完成為止。
- DocumentDB 變更串流以時間為基礎。請確定您的`change_stream_log_retention_duration`設定夠大，足以涵蓋完成完整載入的時間。
- MongoDB Oplog 是固定大小。確定其大小足以在完全載入期間保留所有操作。

設定 DMS

建立 DMS 執行個體、來源和目標端點，並測試每個端點。

啟用 DocumentDB 壓縮

Amazon DocumentDB 5.0：將自訂參數群組連接至 Amazon DocumentDB 叢集並將 參數設定為 來啟用壓縮。 Amazon DocumentDB `default_collection_compression` `enabled` 如需詳細資訊，請參閱管理集合層級文件壓縮。

Amazon DocumentDB 8.0：預設會啟用 Zstd 壓縮。如需詳細資訊，請參閱在 Amazon DocumentDB 8.0 中管理字典型壓縮。

建立複寫任務

1. 在 DMS 主控台的導覽窗格中，選擇遷移或複寫，然後選擇任務。
2. 選擇 Create task (建立任務)。
3. 在建立任務頁面上的任務組態區段中：
   - 輸入唯一且有意義的任務識別符（例如，「mongodb-docdb-replication」）。
   - 選擇您先前在來源資料庫端點下拉式功能表中建立的來源端點。
   - 選擇您先前在目標資料庫端點下拉式功能表中建立的目標端點。
   - 針對任務類型，選擇遷移和複寫。
4. 在設定區段中：
   - 對於任務日誌，勾選開啟 CloudWatch 日誌方塊。
   - 對於編輯模式（在區段頂端），選擇 JSON 編輯器並設定下列屬性：
     - `ParallelApplyThreads` 設定為 5（在 下`TargetMetadata`)。這會在 CDC 中啟用每秒約 1000 個 insert/update/delete ops。
     - `MaxFullLoadSubTasks` 設定為 16（在 下`FullLoadSettings`)。考慮根據您的執行個體大小增加此值。

- 對於大型集合（超過 100 GB)，啟用自動分割（在資料表映射下和 `parallel-load` 屬性下）：
  - "type"： "partitions-auto"
  - 「number-of-partitions」：16

監控進度

使用 AWS DMS 主控台或建立自訂儀表板 (儀表板工具) 來追蹤遷移。專注於下列指標：

- FullLoadThroughputBandwidthTarget — 測量 DMS 在遷移完整載入階段將資料傳輸至目標資料庫時所使用的網路頻寬（以 KB/秒為單位）。
- CDCLatencyTarget — 測量當來源資料庫中發生變更，以及將該變更套用至目標資料庫時，所發生的時間延遲（以秒為單位）。
- CDCThroughputRowsTarget — 測量 DMS 在遷移持續複寫階段期間套用至目標資料庫的每秒資料列數。

其他資訊

如需 Amazon DocumentDB 和 的詳細資訊 AWS DMS，請參閱：

- Amazon DocumentDB 遷移 Runbook
- 從 MongoDB 遷移至 Amazon DocumentDB

Amazon DocumentDB 遷移 Runbook

此 Runbook 提供使用 AWS Database Migration Service (DMS) 將 MongoDB 資料庫遷移至 Amazon DocumentDB 的完整指南。它旨在在整個end-to-end遷移過程中支援資料庫管理員、雲端工程師和開發人員，從初始探索到遷移後驗證。

考慮到 MongoDB 和 Amazon DocumentDB 之間的實作和支援功能差異，此 Runbook 強調結構化和系統化的方法。它概述了必要的遷移前評估、強調相容性考量，並詳細說明確保成功遷移所需的關鍵任務，並將中斷降至最低。

Runbook 會組織成下列主題：

- 相容性 — 了解 Amazon DocumentDB 中支援的 MongoDB 功能和資料類型，並識別潛在的不相容。

- 工作負載探索 — 分析現有的 MongoDB 工作負載，包括讀取/寫入模式、資料磁碟區和效能基準。
- 索引遷移 — 分析擷取和轉換 MongoDB 索引的策略，以在 Amazon DocumentDB 中獲得最佳效能。
- 使用者遷移 — 詳細說明將資料庫使用者、角色和存取控制遷移至 Amazon DocumentDB 的方法。
- 資料遷移 — 涵蓋各種使用 進行資料遷移的方法 AWS DMS，包括完全載入和變更資料擷取 (CDC)。
- 監控 — 詳細說明使用 DMS 或原生工具進行遷移時的各種監控方法。
- 驗證 — 提供遷移後資料完整性檢查、功能驗證和效能比較的程序。

透過遵循此 Runbook 中的指引，團隊可以確保順利、安全且有效率地轉換至 Amazon DocumentDB，同時保留應用程式功能並將風險降至最低。

相容性

主題

- 核心功能相容性
- Amazon DocumentDB 相容性評估工具

從 MongoDB 遷移到 Amazon DocumentDB 時，完整的初始評估和功能相容性檢查對於成功遷移至關重要。此程序從 MongoDB 功能的全面清查開始，包括彙總管道運算子、查詢模式、索引和資料模型。

由於 Amazon DocumentDB 與 MongoDB 3.6、4.0、5.0 和 8.0 API 相容，因此使用較新 MongoDB 特定功能的應用程式可能需要重構。要評估的關鍵領域包括碎片機制 (Amazon DocumentDB 使用不同的方法）、交易實作、變更串流功能和索引類型 （特別是稀疏和部分索引）。

效能特性也有所不同，Amazon DocumentDB 已針對具有可預測效能的企業工作負載進行最佳化。測試應涉及對兩個系統執行代表性工作負載，以識別可能需要最佳化的查詢模式。

在評估階段期間，監控執行計畫以偵測潛在的效能差距非常重要。這有助於建立明確的遷移路線圖、識別必要的應用程式變更，並建立逼真的時間表，以實現順暢的轉換。

核心功能相容性

全方位功能支援

- CRUD 操作 — 享受對所有基本建立、讀取、更新和刪除操作的完整支援，包括大量和查詢運算子 - 提供無縫的應用程式相容性。

- 豐富的索引功能 — 利用對單一欄位、複合、TTL、部分、稀疏和 2dsphere 索引的全面支援，以最佳化文字型查詢的查詢效能和文字索引（第 5 版）。
- 企業級複寫 — 受益於強大的自動容錯移轉機制，其具有僅供讀取複本，可提供卓越的高可用性，無需營運開銷。
- 進階備份解決方案 — 使用具有Point-in-Time復原 (PITR) 的自動備份系統以及用於資料保護的隨需手動快照，輕鬆休息。

增強 AWS整合功能

- 簡化彙總 — 利用最常用的彙總階段 (`$match`、`$sort`、、 `$project`等）`$group`，為企業工作負載提供最佳化效能。
- 交易支援 — 實作多文件和多集合交易，非常適合大多數商業應用程式需求。
- 即時資料追蹤 — 透過簡單的命令啟用變更串流，並透過簡單的參數群組設定來增加變更串流保留期間，以進行即時資料變更監控。
- 位置型服務 — 實作地理空間應用程式，並支援`$geoNear`運算子和 2dsphere 索引。
- 文字搜尋功能 — 利用內建的文字搜尋功能來滿足內容探索需求。

現代架構優勢

- 雲端原生設計 — AWS享受最佳化的架構，可將 MapReduce 等傳統功能取代為更有效率的彙總管道操作。
- 增強安全性 — 受益於 AWS Identity and Access Management (IAM)、SCRAM-SHA-1、SCRAM-SHA-256、X.509 憑證身分驗證和密碼型身分驗證。
- 可預測的效能 — 體驗專為企業工作負載最佳化的一致效能。

如需 Amazon DocumentDB 功能的完整概觀，請參閱 Amazon DocumentDB 中支援的 MongoDB APIs、操作和資料類型和 功能差異：Amazon DocumentDB 和 MongoDB以最大化資料庫的潛力。

Amazon DocumentDB 不支援 MongoDB 提供的所有索引。我們在 GitHub 上提供免費索引工具，以檢查相容性。執行索引工具來評估不相容並相應地規劃解決方法。

Amazon DocumentDB 相容性評估工具

GitHub 上的 Amazon DocumentDB 相容性工具會分析您的MongoDB工作負載，並報告不支援或需要在 Amazon DocumentDB 中變更的運算子、命令和功能。它支援從 3.6 到 8.0.1 的目標版本（預設值：8.0.1)。

建議：URI 模式 (MongoDB 5.0+)

URI 模式會直接連線至 MongoDB 執行個體、即時取樣操作，而且不需要分析或日誌存取。

替代方法

- 日誌型分析 — 剖析MongoDB分析器日誌。擷取實際執行時間行為，但需要啟用分析，且僅涵蓋記錄期間。
- 原始程式碼分析 — 掃描應用程式原始程式碼以取得 MongoDB API 用量。提供全面的涵蓋範圍，但可能會標記未使用的程式碼路徑，並且無法偵測動態建構的查詢。

先決條件

- Python 3.7+
- 複製工具： `git clone https://github.com/awslabs/amazon-documentdb-tools.git`
- 安裝相依性： `pip install -r compat-tool/requirements.txt`

如需完整用量詳細資訊和所有可用選項，請參閱 GitHub 上的 README。

工作負載探索

從 MongoDB 遷移至 Amazon DocumentDB 需要徹底了解現有的資料庫工作負載。工作負載探索是分析資料庫使用模式、資料結構、查詢效能和操作相依性的程序，以確保無縫轉換，並將中斷降至最低。本節概述工作負載探索所涉及的關鍵步驟，以促進從 MongoDB 到 Amazon DocumentDB 的有效遷移。

主題

- 評估現有的 MongoDB 部署
- 識別資料模型差異
- 查詢和效能分析
- 安全性和存取控制檢閱
- 操作和監控考量

評估現有的 MongoDB 部署

在遷移之前，評估目前的 MongoDB 環境至關重要，包括：

- 叢集架構 — 識別節點、複本集和碎片組態的數量。從 MongoDB 遷移至 Amazon DocumentDB 時，了解您的 MongoDB 碎片組態很重要，因為 Amazon DocumentDB 不支援使用者控制的碎片。專為碎片 MongoDB 環境設計的應用程式將需要架構變更，因為 Amazon DocumentDB 在其儲存型架構中使用不同的擴展方法。移至 Amazon DocumentDB 時，您將需要調整資料分佈策略，並可能合併碎片集合。
- 儲存和資料磁碟區 — 測量叢集的總資料大小和索引大小。搭配 Oplog 檢閱工具來補充這一點，以了解寫入模式和資料成長速度。如需調整叢集大小的詳細資訊，請參閱執行個體大小。
- 工作負載模式 — 分析讀取和寫入輸送量、查詢執行頻率和索引效率。
- 操作相依性 — 記錄依賴 MongoDB 的所有應用程式、服務和整合。

識別資料模型差異

雖然 Amazon DocumentDB 與 MongoDB 相容，但支援的功能存在差異，例如：

- 交易 — Amazon DocumentDB 支援 ACID 交易，但有一些 限制。
- 結構描述設計 — 確保文件結構、內嵌文件和參考符合 Amazon DocumentDB 的最佳實務。

查詢和效能分析

了解查詢行為有助於最佳化遷移和遷移後效能。要分析的關鍵領域包括：

- 緩慢查詢 — 使用 MongoDB 的分析工具識別具有高執行時間的查詢。
- 查詢模式 — 分類常見的查詢類型，包括 CRUD 操作和彙總。
- 索引用量 — 評估索引是否有效使用，或是否需要在 Amazon DocumentDB 中最佳化。若要評估索引用量並最佳化 Amazon DocumentDB 中的效能，請在關鍵查詢上使用`$indexStats`彙總管道階段結合 `explain()`方法。首先執行 `db.collection.aggregate([{$indexStats{}}])`以識別正在使用的索引。您可以使用 執行最常用的查詢，以執行更詳細的分析`explainPlan`。
- 並行和工作負載分佈 — 評估讀取和寫入比率、連線集區和效能瓶頸。

安全性和存取控制檢閱

身分驗證和授權

- MongoDB RBAC 到 Amazon DocumentDB IAM 和 RBAC — 將 MongoDB 的角色型存取控制使用者和角色映射到 AWS Identity and Access Management (IAM) 政策和 Amazon DocumentDB SCRAM 身分驗證使用者。

- 使用者遷移策略 — 規劃將資料庫使用者、自訂角色和權限遷移至 Amazon DocumentDB 支援的身分驗證機制。
- 權限差異 — 識別沒有直接 Amazon DocumentDB 對等項目的 MongoDB 權限 （ 例如叢集管理角色 ）。 Amazon DocumentDB
- 應用程式身分驗證 — 更新 Amazon DocumentDB 密碼政策的連線字串和憑證管理。您可以使用秘密管理員來存放您的登入資料並輪換密碼。
- 服務帳戶管理 — 建立管理服務帳戶登入資料的程序 AWS Secrets Manager。
- 最低權限實作 — 檢閱和精簡存取控制，以在新環境中實作最低權限原則。

加密

確保靜態加密和傳輸中加密符合合規要求。

網路組態

規劃 Virtual Private Cloud (VPC) 設定和安全群組規則。

操作和監控考量

為了維護系統可靠性，工作負載探索也應該包含：

- 備份和還原策略 — 評估現有的備份方法和 Amazon DocumentDB 的備份功能。
- AWS Backup 整合 — 利用 AWS Backup 跨 AWS 服務集中備份管理，包括 Amazon DocumentDB。
- CloudWatch 指標 — 將 MongoDB 監控指標映射至 CPU、記憶體、連線和儲存體的 Amazon DocumentDB CloudWatch 指標。
- 績效詳情 — 實作 Amazon DocumentDB 績效詳情，透過詳細的查詢分析視覺化資料庫負載和分析效能問題。
- Profiler — 設定 Amazon DocumentDB Profiler 以擷取慢速執行的操作 （ 類似於 MongoDB 的 Profiler，但使用 Amazon DocumentDB 特定設定 ）。
  - 透過具有適當閾值的參數群組啟用 。
  - 分析分析分析器資料以識別最佳化機會
- CloudWatch Events — 設定 Amazon DocumentDB 叢集事件的事件驅動型監控。
  - 設定備份事件、維護時段和容錯移轉的通知。
  - 與 Amazon SNS 整合，用於提醒和 AWS Lambda 自動回應。

- 稽核記錄 — 規劃稽核記錄組態，以追蹤使用者活動和安全相關事件。
- 增強型監控 — 以 1 秒的間隔對精細作業系統層級指標啟用增強型監控。

建立新的目標叢集

依照中的步驟建立 Amazon DocumentDB 叢集建立 Amazon DocumentDB 叢集。設定叢集進行遷移時，請套用下列建議：

- 執行個體類別：選擇 R8G 以獲得最佳效能。如果您的目標區域無法使用 R8G，請使用 R6G。盡可能選擇大型執行個體，以獲得最佳完全載入輸送量。遷移完成後縮減規模。
- 引擎版本：最新 (8.0.1)
- 執行個體數量：選擇 1 個執行個體，將遷移期間的成本降至最低。完整載入遷移完成後，將 擴展至 3 個執行個體以獲得高可用性。
- 網路設定：確保您的 Amazon DocumentDB 安全群組允許來自 DMS 複寫執行個體安全群組 （用於線上遷移） 或從 EC2 遷移主機 （用於使用 mongorestore 的離線遷移） 的傳入連線。

索引遷移

從 MongoDB 遷移到 Amazon DocumentDB 不僅需要傳輸資料，還包含索引，以維持查詢效能並最佳化資料庫操作。本節概述將索引從 MongoDB 遷移至 Amazon DocumentDB 的詳細step-by-step程序，同時確保相容性和效率。

使用 Amazon DocumentDB 索引工具

在 GitHub 上複製索引工具

```
git clone https://github.com/awslabs/amazon-documentdb-tools.git
cd amazon-documentdb-tools/index-tool
```

```
pip install -r requirements.txt
```

從 MongoDB 匯出索引 （如果從 MongoDB 遷移）

```
python3 migrationtools/documentdb_index_tool.py \
  --dump-indexes \
  --dir index_export \
```

```
  --uri 'mongodb://<username>:<password>@<source_endpoint>:27017'
```

驗證索引

```
python3 migrationtools/documentdb_index_tool.py \
  --show-issues \
  --dir index_export
```

匯入索引

```
python3 migrationtools/documentdb_index_tool.py \
  --restore-indexes \
  --skip-incompatible \
  --dir index_export \
  --uri 'mongodb://<username>:<password>@<target_endpoint>:27017/?
tls=true&tlsCAFile=global-bundle.pem&replicaSet=rs0&retryWrites=false'
```

使用者遷移

將使用者從 MongoDB 遷移至 Amazon DocumentDB 是維護存取控制、身分驗證和資料庫安全性的必要條件。本節概述了使用 Amazon DocumentDB 匯出使用者工具成功遷移 MongoDB 使用者，同時保留其角色和許可的詳細步驟。

使用 Amazon DocumentDB 匯出使用者工具

從 MongoDB 或 Amazon DocumentDB `Export Users tool` 匯出使用者和角色到 JavaScript 檔案，然後可用於在另一個叢集中重新建立它們。

先決條件

```
# Clone the repository
git clone https://github.com/awslabs/amazon-documentdb-tools.git
cd amazon-documentdb-tools/migration/export-users
```

```
# Install required dependencies
pip install pymongo
```

步驟 1：匯出使用者和角色

```
# Export users and roles to JavaScript files
```

```
python3 docdbExportUsers.py \
  --users-file mongodb-users.js \
  --roles-file mongodb-roles.js \
  --uri "mongodb://<username>:<password>@<source_endpoint>:27017"
```

步驟 2：編輯使用者檔案

基於安全考量，密碼不會匯出。開啟 mongodb-users.js 檔案，並取代每個createUser陳述式中的pwd值，為每位使用者新增密碼：

```
db.getSiblingDB("admin").createUser({
  user: "appuser",
  pwd: "REPLACE_THIS_PASS",
  roles: [
    { role: "readWrite", db: "mydb" }
  ]
})
```

步驟 3：將自訂角色還原至 Amazon DocumentDB

```
# Import roles first
mongosh \
  --tls \
  --host <target_endpoint>:27017 \
  --tlsCAFile global-bundle.pem \
  --username <username> \
  --password <password> \
  mongodb-roles.js
```

步驟 4：將使用者還原至 Amazon DocumentDB

```
# Import users after roles are created
mongosh \
  --tls \
  --host <target_endpoint>:27017 \
  --tlsCAFile global-bundle.pem \
  --username <username> \
  --password <password> \
  mongodb-users.js
```

重要說明

- 基於安全考量，密碼不會匯出，且必須手動新增至 user.js 檔案。
- 必須在使用者之前匯入角色，以確保適當的角色指派。
- 此工具會產生可直接使用 mongosh shell 執行的 JavaScript 檔案。
- 自訂角色及其權限會在遷移期間保留。
- 此方法允許在匯入之前檢閱和修改使用者許可。

此方法提供安全且靈活的方法，可將使用者和角色從 MongoDB 遷移至 Amazon DocumentDB，同時允許在遷移程序期間重設密碼。

資料遷移

主題

- 線上遷移
- 離線移轉

線上遷移

本節提供使用 執行從 MongoDB 到 Amazon DocumentDB 線上遷移的詳細步驟 AWS DMS ，以盡可能減少停機時間和持續複寫。首先，您將 Amazon DocumentDB 叢集設定為目標，並確保 MongoDB 執行個體已正確設定為來源，通常需要複本集模式才能擷取變更資料。接著，您可以建立 DMS 複寫執行個體，並使用必要的連線詳細資訊定義來源和目標端點。驗證端點之後，您可以設定並啟動遷移任務，其中包含完整資料載入、持續複寫或兩者。

建立新的目標叢集

請參閱 建立新的目標叢集。

設定來源

MongoDB 和 Amazon DocumentDB 都可以做為遷移來源，視您的案例而定：

- MongoDB 作為來源 — 從內部部署或自我管理的 MongoDB 遷移到 Amazon DocumentDB 或其他 AWS 資料庫服務時很常見。需要在具有適當大小的 oplog 的複本集模式下執行 （確保其大小在完全載入期間保留所有操作），以支援遷移期間的變更資料擷取。
- Amazon DocumentDB 作為來源 — 通常用於跨區域複寫、版本升級或遷移至其他資料庫服務，例如 MongoDB Atlas。需要啟用變更串流透過在叢集`change_stream_log_retention_duration`參數群組中設定 參數來擷取遷移期間持續的變

更。請確定您的change_stream_log_retention_duration設定夠大，足以涵蓋完成完整載入所需的時間。

開始遷移之前，請將您的來源設定為允許 AWS DMS 存取。

建立具有適當許可的 MongoDB 使用者：

```
db.createUser({
  user: "dmsUser",
  pwd: "<password>",
  roles: [{ role: "readAnyDatabase", db: "admin" }]
})
```

設定網路和身分驗證。

設定 MongoDB 到 DMS 遷移的網路連線時：

EC2-hosted MongoDB 來源

- 修改 EC2 安全群組，以允許來自 DMS 複寫執行個體安全群組的傳入流量。
- 新增 TCP 連接埠 27017（或您的自訂 MongoDB 連接埠）的規則。
- 使用 DMS 複寫執行個體的安全群組 ID 做為精確存取控制的來源。
- 確保 EC2 執行個體的子網路具有通往 DMS 複寫執行個體子網路的路由。

內部部署 MongoDB 來源

- 設定防火牆以允許來自 DMS 複寫執行個體公有 IP 地址的傳入連線。
- 如果使用 Direct Connect 或 VPN，請確保網路與包含 DMS 執行個體的 VPC 之間的適當路由。
- 使用 telnet 或 nc 命令測試從 DMS 子網路到 MongoDB 伺服器的連線。

MongoDB Atlas 來源

- 將 DMS 複寫執行個體 IP 地址新增至 MongoDB Atlas IP 允許清單。
- 如果 Atlas 正在執行，請在 AWS VPC 和 MongoDB Atlas VPC 之間設定 VPC 對等互連 AWS。
- 如果在其他雲端提供者上執行，請設定 AWS PrivateLink 私有連線 (Enterprise 層）。
- 建立具有適當讀取/寫入許可的專用使用者。
- 使用將 SSL 模式設定為 "verify-full" 的 MongoDB Atlas 連線字串。

- 確保遷移持續時間有足夠的 oplog 大小。

Amazon DocumentDB 來源

設定來源 Amazon DocumentDB 安全群組，以允許來自 DMS 複寫執行個體安全群組的傳入流量。

建立 DMS 複寫執行個體

在 GitHub 上使用 DMS Buddy 來建立具有最佳 DMS 設定和執行個體大小的最佳遷移基礎設施。如果您想要手動設定 ，請依照下列步驟進行：

1. 在 AWS DMS 主控台的導覽窗格中，選擇遷移或複寫，然後選擇佈建執行個體。
2. 輸入複寫執行個體詳細資訊：
   - 執行個體名稱：選擇唯一的名稱。
   - 執行個體類別：根據工作負載選取 。範例：dms.r7i.large （小型工作負載）、dms.r7i.4xlarge （大型工作負載）。
   - 引擎版本：3.5.4
   - 配置的儲存空間：預設值為 50 GB （視需要增加）。這取決於文件大小、更新/秒和完全載入持續時間。
   - 異地同步備份部署：視需要啟用 以獲得高可用性。
   - 選擇與 Amazon DocumentDB 相同的 VPC。
   - 確保安全群組允許來自來源和 Amazon DocumentDB 的傳入流量。
3. 選擇建立複寫執行個體，並等待狀態可用。

建立 DMS 端點

建立來源端點

對於 MongoDB 來源

1. 在 DMS 主控台的導覽窗格中，選擇遷移或複寫，然後選擇端點。
2. 選擇建立端點。
3. 在建立端點頁面上，選擇來源端點。
4. 在端點組態區段中：
   - 輸入唯一且有意義的端點識別符 （例如，「mongodb-source」）。

- 選擇 MongoDB 做為來源引擎。
- 對於端點資料庫的存取，選擇手動提供存取資訊。
- 在伺服器名稱中，輸入您的 *MongoDB ### DNS ##/IP ##*。
- 針對連接埠，輸入 27017（預設 MongoDB 連接埠）。
- 對於身分驗證模式，為您的應用程式選擇適當的模式（密碼/SSL)（預設為 Secrets Manager)。
- 如果身分驗證模式是密碼，請提供：
  - 使用者名稱和密碼：輸入 MongoDB 登入資料。
  - 資料庫名稱：您的來源資料庫名稱。
  - 身分驗證機制：SCRAM-SHA-1（預設）或適當的機制

5. 對於中繼資料模式，請保留 文件的預設設定。
6. 其他連線屬性：

- authSource=admin（如果身分驗證資料庫不同）
- replicaSet=<your-replica-set-name> (CDC 需要）

對於 Amazon DocumentDB 來源

1. 在 DMS 主控台的導覽窗格中，選擇遷移或複寫，然後選擇端點。
2. 選擇建立端點。
3. 在建立端點頁面上，選擇來源端點。
4. 在端點組態區段中：

- 輸入唯一且有意義的端點識別符（例如 "docdb-source")。
- 選擇 Amazon DocumentDB 做為來源引擎。
- 對於端點資料庫的存取，選擇手動提供存取資訊。
- 在伺服器名稱中，輸入您的*## Amazon DocumentDB ####*。
- 針對連接埠，輸入 27017（預設 Amazon DocumentDB 連接埠）。
- 對於 SSL 模式，選擇 verify-full（建議用於 Amazon DocumentDB)。
- 針對 CA 憑證，選擇 Amazon RDS 根 CA 憑證。
- 對於身分驗證模式，為您的應用程式選擇適當的模式（密碼/SSL)（預設為 Secrets Manager)。
- 如果身分驗證模式是密碼，請提供：

- 使用者名稱和密碼：輸入 Amazon DocumentDB 登入資料。
- 資料庫名稱：您的來源資料庫名稱。
- 身分驗證機制：SCRAM-SHA-1 （預設） 或適當的機制

5. 對於中繼資料模式，請保留 文件的預設設定。

建立目標端點 (Amazon DocumentDB)

1. 在 DMS 主控台的導覽窗格中，選擇遷移或複寫，然後選擇端點。
2. 選擇建立端點。
3. 在建立端點頁面上，選擇目標端點。
4. 在端點組態區段中：

- 輸入唯一且有意義的端點識別符 （例如 "docdb-target")。
- 選擇 Amazon DocumentDB 作為目標引擎。
- 針對端點資料庫的存取，選擇您要用來驗證資料庫存取的方法：
  - 如果您選擇 AWS Secrets Manager，請在 Secret 欄位中選擇存放 Amazon DocumentDB 登入資料的秘密。
  - 如果您選擇手動提供存取資訊：
    - 在伺服器名稱中，輸入您的*## Amazon DocumentDB ####*。
    - 針對連接埠，輸入 27017 （預設 Amazon DocumentDB 連接埠）。
    - 對於 SSL 模式，選擇 verify-full （建議用於 Amazon DocumentDB)。
    - 對於 CA 憑證，請下載並指定用於 SSL 驗證的 CA 憑證套件。
    - 對於身分驗證模式，為您的應用程式選擇適當的模式 （密碼/SSL) （預設為 Secrets Manager)。
    - 如果身分驗證模式是密碼，請提供：
      - 使用者名稱和密碼：輸入 Amazon DocumentDB 登入資料。
      - 資料庫名稱：您的來源資料庫名稱。
      - 身分驗證機制：SCRAM-SHA-1 （預設） 或適當的機制

5. 對於中繼資料模式，請保留 文件的預設設定。

建立複寫任務

1. 在 DMS 主控台的導覽窗格中，選擇遷移或複寫，然後選擇任務。

2. 選擇 Create task (建立任務)。
3. 在建立任務頁面上的任務組態區段中：
   - 輸入唯一且有意義的任務識別符 （ 例如「mongodb-docdb-replication」)。
   - 在來源資料庫端點下拉式功能表中選擇您先前建立的來源端點。
   - 選擇您先前在目標資料庫端點下拉式功能表中建立的目標端點。
   - 選擇您先前在佈建執行個體下拉式功能表中建立的複寫執行個體。
   - 針對任務類型，選擇遷移和複寫。
   - 對於您希望繼續從來源複寫到目標的時間長度？，請選擇在有限時間內。
4. 在設定區段中：
   - 針對目標資料表準備模式，選擇不執行任何動作。否則，將會刪除已建立的集合和索引。
   - 對於任務日誌，勾選開啟 CloudWatch 日誌方塊。
   - 保留所有其他設定的預設值。
5. 回到設定區段頂端，針對編輯模式，選擇 JSON 編輯器並設定下列屬性：

```
{
  "TargetMetadata": {
    "ParallelApplyThreads": 5
  },
  "FullLoadSettings": {
    "MaxFullLoadSubTasks": 16
  }
}
```

   `MaxFullLoadSubTasks` 根據您的集合計數和執行個體容量進行設定。對於較少或較小的集合，請使用較低的值 (2–4)，對於許多大型集合，請使用較高的值 (8–16)。如需自動調整大小，請參閱 GitHub 上的 DMS Buddy。
6. 在資料表映射區段中，新增新的選取規則：
   - 針對結構描述名稱，新增要遷移的來源資料庫。使用 % 指定多個資料庫。
   - 針對結構描述資料表名稱，新增要遷移的來源集合。使用 % 指定多個集合。
   - 對於動作，保留 Include 的預設設定
7. 對於大型集合 （ 超過 100GB)，新增資料表設定規則：
   - 針對結構描述名稱，新增要遷移的來源資料庫。使用 % 指定多個資料庫。

- 針對結構描述資料表名稱，新增要遷移的來源集合。使用 % 指定多個集合。
- 對於分割區數量，請根據您的工作負載設定 。對於少量大型集合，請將分割區設為 ≥MaxFullLoadSubTasks，讓所有工作者保持忙碌。對於許多平行載入的集合，請設定 ≤ 的分割區，MaxFullLoadSubTasks因為集合計數本身提供足夠的平行處理。

8. 在預遷移評估區段中，確定已關閉。Amazon DocumentDB 不支援預遷移評估做為目標，且必須停用。

離線移轉

本節概述使用原生 MongoDB 工具從自我管理的 MongoDB 執行個體執行離線遷移至 Amazon DocumentDB MongoDB 的程序： mongodump和 mongorestore。

先決條件

來源 MongoDB 需求

- 使用適當的許可存取來源 MongoDB 執行個體。
- 視需要安裝 mongodump。（在 MongoDB 安裝期間安裝）。
- 請確定傾印檔案有足夠的磁碟空間。

目標 Amazon DocumentDB 需求

- 請確定您已佈建 Amazon DocumentDB 叢集。
- 確保與 Amazon DocumentDB 相同的 VPC 中有 EC2 執行個體，以利遷移。
- 您的來源環境和 Amazon DocumentDB 之間必須具備網路連線能力。
- mongorestore 必須安裝在遷移 EC2 執行個體上。
- 必須設定適當的 IAM 許可來存取 Amazon DocumentDB、

一般要求

- AWS CLI 必須設定 （如果使用 AWS 服務進行中繼儲存）
- 必須有足夠的頻寬才能進行資料傳輸。
- 應核准停機時間時段 （如果進行即時遷移，請考慮其他方法）

準備 Amazon DocumentDB 叢集

請參閱 建立新的目標叢集。

執行資料傾印 (mongodump)

選擇下列其中一個選項來建立傾印檔案：

- 選項 1：基本

```
mongodump \
  --uri="mongodb://<user>:<password>@<source_endpoint>:27017/<database>" \
  --out=/path/to/dump
```

- 選項 2：更好的控制和效能

  使用 gzip 壓縮輸出、平行傾印集合、啟用 TLS，並在可用時從次要複本讀取：

```
mongodump \
  --uri="mongodb://<user>:<password>@<source_endpoint>:27017" \
  --out=/path/to/dump \
  --gzip \
  --numParallelCollections=4 \
  --tls \
  --authenticationDatabase=admin \
  --readPreference=secondaryPreferred
```

- 選項 3：大型資料庫

  傾印特定資料庫和集合、依查詢篩選文件，以及輸出至單一壓縮封存：

```
mongodump \
  --host=<source_endpoint> \
  --port=27017 \
  --username=<user> \
  --password=<password> \
  --db=<specific_db> \
  --collection=<specific_collection> \
  --query='{ "date": { "$gt": "2020-01-01" } }' \
  --archive=/path/to/archive.gz \
  --gzip \
  --tls
```

將傾印檔案傳輸到還原環境

根據您的傾印大小選擇適當的方法：

- 小型：直接複製到遷移機器（您先前建立的 EC2 執行個體）：

```
scp -r /path/to/dump user@migration-machine:/path/to/restore
```

- 中 — 使用 Amazon S3 作為中繼儲存：

```
aws s3 cp --recursive /path/to/dump s3://your-bucket/mongodb-dump/
```

- 大型 — 對於非常大型的資料庫，請考慮 AWS DataSync 或實體傳輸。

將資料還原至 Amazon DocumentDB (mongorestore)

開始還原程序之前，請在 Amazon DocumentDB 中建立索引。您可以使用 Amazon DocumentDB 索引工具來匯出和匯入索引。

選擇下列其中一個選項來還原資料：

- 選項 1：基本還原

```
mongorestore \
  --uri="mongodb://<user>:<password>@<target_endpoint>:27017" \
  --tls \
  --tlsCAFile=global-bundle.pem \
  /path/to/dump
```

- 選項 2：更好的控制和效能

  使用平行收集和文件插入還原 gzip 壓縮的傾印。因為索引已預先建立，所以略過索引還原：

```
mongorestore \
  --uri="mongodb://<user>:<password>@<target_endpoint>:27017" \
  --tls \
  --tlsCAFile=global-bundle.pem \
  --gzip \
  --numParallelCollections=4 \
  --numInsertionWorkersPerCollection=4 \
  --noIndexRestore \
  /path/to/dump
```

- 選項 3：大型資料庫或特定控制項

  從具有命名空間篩選的壓縮封存還原，以包含或排除特定資料庫和集合。因為索引已預先建立，所以略過索引還原：

```
mongorestore \
  --host=<target_endpoint> \
  --port=27017 \
  --username=<user> \
  --password=<password> \
  --tls \
  --tlsCAFile=global-bundle.pem \
  --archive=/path/to/archive.gz \
  --gzip \
  --nsInclude="db1.*" \
  --nsExclude="db1.sensitive_data" \
  --noIndexRestore
```

監控

本節提供詳細監控程序，以追蹤持續遷移的進度、效能和運作狀態。

無論遷移方法為何 (AWS DMS、mongodump/mongorestore 或其他工具），都會套用監控步驟。

AWS DMS 遷移監控 （ 如適用 ）

監控下列主要 CloudWatch 指標：

完全載入階段指標

- FullLoadThroughputBandwidthTarget — 完全載入期間的網路頻寬 (KB/秒)
- FullLoadThroughputRowsTarget — 每秒載入的資料列/文件數
- FullLoadThroughputTablesTarget — 每分鐘完成的資料表/集合數量
- FullLoadProgressPercent — 已完成完全載入的百分比
- TablesLoaded — 成功載入的資料表/集合數目
- TablesLoading — 目前正在載入的資料表/集合數量
- TablesQueued — 等待載入的資料表/集合數目
- TablesErrored — 無法載入的資料表/集合數量

CDC 階段指標

- CDCLatencyTarget — 來源變更與目標應用程式之間的時間延遲（秒）
- CDCLatencySource — 來源變更和 DMS 讀取之間的時間延遲（秒）
- CDCThroughputRowsTarget — 持續複寫期間每秒套用的資料列數
- CDCThroughputBandwidthTarget — CDC 期間的網路頻寬 (KB/秒）
- CDCIncomingChanges — 從來源接收的變更事件數目
- CDCChangesMemoryTarget — 用於儲存目標端變更的記憶體 (MB)

資源指標

- CPUUtilization — 複寫執行個體的 CPU 用量
- FreeableMemory — 複寫執行個體上的可用記憶體
- FreeStorageSpace — 複寫執行個體上的可用儲存體
- NetworkTransmitThroughput — 複寫執行個體的網路輸送量
- NetworkReceiveThroughput — 複寫執行個體的網路輸送量

錯誤指標

- ErrorsCount — 遷移期間的錯誤總數
- TableErrorsCount — 資料表特定錯誤的數量
- RecordsErrorsCount — 記錄特定錯誤的數量

為 `CDCLatencyTarget`和 等關鍵指標建立 CloudWatch 警示`CPUUtilization`，以便在遷移效能降低時接收通知。

DMS 日誌 (CloudWatch 日誌）

1. 前往 Amazon CloudWatch Logs 主控台。
2. 在您的日誌群組上尋找並選擇 。它看起來類似於「dms-tasks –」。
3. 尋找可能包含錯誤資訊的日誌串流：
    - 名稱中有「錯誤」的串流
    - 具有任務 IDs串流

- 遷移期間的最新日誌串流

4. 在這些串流中，搜尋關鍵字，例如：

- 「錯誤」
- 「例外狀況」
- 「失敗」
- 「警告」

DMS 任務狀態（使用 AWS CLI)

```
aws dms describe-replication-tasks \
  --filters Name=replication-task-id,Values=<task_id> \
  --query "ReplicationTasks[0].Status"
```

預期的狀態流程：

建立 → ready → 執行中 → 停止 → 已停止（或失敗）

使用 監控 **docdb-dashboarder**

該docdb-dashboarder工具透過自動產生具有基本效能指標的 CloudWatch 儀表板，為 Amazon DocumentDB 叢集提供全面的監控。這些儀表板會顯示關鍵叢集層級指標（複本延遲、操作計數器）、執行個體層級指標 (CPU、記憶體、連線）和儲存指標（磁碟區使用量、備份儲存）。對於遷移案例，該工具提供專門的儀表板，使用 CDC 複寫延遲和操作速率等指標來追蹤遷移進度。儀表板可以同時監控多個叢集，並包含對 NVMe 後端執行個體的支援。透過視覺化這些指標，團隊可以主動識別效能瓶頸、最佳化資源配置，並確保其 Amazon DocumentDB 部署的順暢操作。該工具無需手動建立儀表板，同時在所有環境中提供一致的監控。如需設定說明和進階組態選項，請參閱 Amazon DocumentDB Dashboarder Tool GitHub 儲存庫。

驗證

主題

- 驗證檢查清單
- 結構描述和索引驗證
- 資料取樣和欄位層級驗證
- 使用 DataDiffer 工具進行驗證

本節提供詳細的驗證程序，以確保遷移後的資料一致性、完整性和應用程式相容性。

無論遷移方法為何 (AWS DMS、mongodump/mongorestore 或其他工具），驗證步驟都適用。

驗證檢查清單

確認每個集合中的文件數量符合來源和目標：

MongoDB 來源

```
mongosh \
  --host <source_endpoint> \
  --port 27017 \
  --username <user> \
  --password <password> \
  --eval "db.<collection>.countDocuments()"
```

Amazon DocumentDB 目標

```
mongosh \
  --host <target_endpoint> \
  --port 27017 \
  --username <user> \
  --password <password> \
  --tls \
  --tlsCAFile global-bundle.pem \
  --eval "db.<collection>.countDocuments()"
```

結構描述和索引驗證

請確認以下事項：

- 目標中存在所有集合。
- 索引已正確複寫。
- 結構描述定義（如果強制執行）完全相同。

檢查集合（來源與目標）

```
mongosh --host <source_endpoint> \
  --port 27017 \
  --username <username> \
```

```
  --password <password> \
  --eval "db = db.getSiblingDB('<database>'); db.getCollectionNames();"

mongosh --host <target_endpoint> \
  --port 27017 \
  --username <username> \
  --password <password> \
  --tls --tlsCAFile global-bundle.pem \
  --eval "db = db.getSiblingDB('<database>'); db.getCollectionNames();"
```

檢查索引（來源與目標）

```
mongosh --host <source_endpoint> \
  --port 27017 \
  --username <username> \
  --password <password> \
  --eval "db.<collection>.getIndexes()"

mongosh --host <target_endpoint> \
  --port 27017 \
  --username <username> \
  --password <password> \
  --tls --tlsCAFile global-bundle.pem \
  --eval "db.<collection>.getIndexes()"
```

比較集合清單，以確保沒有遺漏或額外的集合。

透過檢查索引名稱、索引鍵定義、唯一限制條件和 TTL 索引（如果有的話）來驗證索引。

檢查結構描述驗證規則（如果在 MongoDB 中使用結構描述驗證）

```
mongosh --host <source_endpoint> \
  --port 27017 \
  --username <username> \
  --password <password> \
  --eval "db.getCollectionInfos({name: '<collection>'})[0].options.validator"

mongosh --host <target_endpoint> \
  --port 27017 \
  --username <username> \
  --password <password> \
  --tls --tlsCAFile global-bundle.pem \
  --eval "db.getCollectionInfos({name: '<collection>'})[0].options.validator"
```

資料取樣和欄位層級驗證

您可以隨機取樣文件，並比較來源和目標之間的欄位。

手動取樣

擷取五個隨機文件（來源）：

```
mongosh --host <source_endpoint> \
  --port 27017 \
  --username <username> \
  --password <password> \
  --eval "db.<collection>.aggregate([{ \$sample: { size: 5 } }])"
```

擷取相同的文件 IDs（目標）：

```
mongosh --host <target_endpoint> \
  --port 27017 \
  --username <username> \
  --password <password> \
  --tls --tlsCAFile global-bundle.pem \
  --eval "db.<collection>.find({ _id: { \$in: [<list_of_ids>] } })"
```

自動取樣

```
import pymongo

# Connect to source and target
source_client = pymongo.MongoClient("<source_uri>")
target_client = pymongo.MongoClient("<target_uri>",
                                    tls=True,
                                    tlsCAFile='global-bundle.pem')
source_db = source_client["<db_name>"]
target_db = target_client["<db_name>"]

# Compare 100 random documents
for doc in source_db["<collection>"].aggregate([{"$sample": {"size": 100}}]):
    target_doc = target_db["<collection>"].find_one({"_id": doc["_id"]})
    if target_doc != doc:
        print(f"Mismatch in _id: {doc['_id']}")
    else:
        print(f"Match: {doc['_id']}")
```

使用 DataDiffer 工具進行驗證

GitHub 上的 DataDiffer 工具提供可靠的方法來比較來源和目標資料庫之間的資料。

先決條件

安裝 DataDiffer 工具之前，必須符合下列先決條件：

- Python 3.7+
- PyMongo 程式庫
- 與來源 MongoDB 和目標 Amazon DocumentDB 叢集的網路連線

設定和安裝

複製儲存庫並導覽至 DataDiffer 目錄

```
git clone https://github.com/awslabs/amazon-documentdb-tools.git
cd amazon-documentdb-tools/migration/data-differ
```

安裝必要的相依性

```
pip install -r requirements.txt
```

執行資料驗證

使用連線詳細資訊建立組態檔案 （ 例如 config.json)

```
{
  "source": {
    "uri": "mongodb://<username>:<password>@<source_endpoint>:27017/?replicaSet=rs0",
    "db": "your_database",
    "collection": "your_collection"
  },
  "target": {
    "uri": "mongodb://<username>:<password>@<target_endpoint>:27017/?
tls=true&tlsCAFile=global-bundle.pem&replicaSet=rs0&retryWrites=false",
    "db": "your_database",
    "collection": "your_collection"
  },
  "options": {
    "batch_size": 1000,
```

```
    "threads": 4,
    "sample_size": 0,
    "verbose": true
  }
}
```

執行 DataDiffer 工具

```
python differ.py --config config.json
```

對於大型集合，請使用抽樣來驗證資料子集

```
python differ.py --config config.json --sample-size 10000
```

若要驗證多個集合，請建立個別的組態檔案或使用批次模式

```
python differ.py --batch-config batch_config.json
```

解譯結果

工具將輸出：

- 來源和目標中的文件總數
- 相符的文件數量
- 遺失的文件數量
- 有差異的文件數量
- 差異的詳細報告 （如果有的話）

最佳實務

以下是使用 DataDiffer 工具時的最佳實務：

- 分階段執行 — 首先驗證文件計數，然後範例金鑰文件，並視需要執行完整比較。
- 檢查結構描述差異 — Amazon DocumentDB 與 MongoDB 相比有一些限制。工具會反白顯示不相容的資料類型或結構。
- 在靜止期間驗證 — 在寫入操作最少時執行驗證，以確保一致性。
- 監控資源用量 — 比較程序可能會耗費大量資源。相應地調整批次大小和執行緒計數。
- 驗證索引 — 驗證資料之後，請確定已在目標 Amazon DocumentDB 叢集上建立所有必要的索引。

- 文件驗證結果 — 將每個集合的驗證結果記錄為遷移文件的一部分。

從 Couchbase Server 遷移

主題

- 簡介
- 與 Amazon DocumentDB 的比較
- 探索
- 規劃
- 移轉
- 驗證

簡介

本指南介紹從 Couchbase Server 遷移到 Amazon DocumentDB 時要考慮的要點。它說明遷移探索、規劃、執行和驗證階段的考量。它還說明如何執行離線和線上遷移。

與 Amazon DocumentDB 的比較

	Couchbase 伺服器	Amazon DocumentDB
資料組織	在 7.0 版和更新版本中，資料會組織成儲存貯體、範圍和集合。在舊版中，資料會組織成儲存貯體。	資料會組織成資料庫和集合。
相容性	每個服務都有不同的 APIs（例如資料、索引、搜尋等）。次要查詢使用 SQL++（先前稱為 N1QL），這是一種以 ANSI 標準 SQL 為基礎的查詢語言，因此許多開發人員都很熟悉。	Amazon DocumentDB 與 MongoDB API 相容。
架構	儲存體會連接至每個叢集執行個體。您無法獨立於儲存體擴展運算。	Amazon DocumentDB 專為雲端而設計，可避免傳統資料庫架構的限制。運算和儲存層在 Amazon

	Couchbase 伺服器	Amazon DocumentDB
		DocumentDB 中分開，且運算層可以獨立於儲存體進行擴展。Amazon DocumentDB
隨需新增讀取容量	新增執行個體即可橫向擴展叢集。由於儲存體會連接至服務執行所在的執行個體，因此擴展所需的時間取決於需要移至新執行個體或重新平衡的資料量。	您可以在叢集中建立最多 15 個 Amazon DocumentDB 複本，以達成 Amazon DocumentDB 叢集的讀取擴展。 Amazon DocumentDB 不會影響儲存層。
從節點故障快速復原	叢集具有自動容錯移轉功能，但讓叢集恢復完整強度的時間取決於需要移至新執行個體的資料量。	Amazon DocumentDB 通常可以在 30 秒內容錯移轉主要伺服器，並在 8-10 分鐘內將叢集還原至完整強度，無論叢集中的資料量為何。
隨著資料成長擴展儲存體	對於自我管理叢集，儲存體和 IOs 不會自動擴展。	Amazon DocumentDB 儲存體和 IOs會自動擴展。
在不影響效能的情況下備份資料	備份是由備份服務執行，預設不會啟用。由於儲存和運算不會分開，因此可能會影響效能。	Amazon DocumentDB 備份預設為啟用，無法關閉。備份由儲存層處理，因此對運算層沒有影響。Amazon DocumentDB 支援從叢集快照還原，並還原至某個時間點。
資料耐久性	叢集中最多可以有 3 個資料複本，總共 4 個複本。執行資料服務的每個執行個體都會有作用中的資料複本和 1、2 或 3 個複本。	Amazon DocumentDB 會維護 6 個資料複本，無論有多少個運算執行個體的寫入規定人數為 4 且持續為 true。儲存層保留 4 個資料副本後，用戶端會收到確認。

	Couchbase 伺服器	Amazon DocumentDB
一致性	支援 K/V 操作的立即一致性。Couchbase SDK 會將 K/V 請求路由到包含資料作用中複本的特定執行個體，因此一旦確認更新，用戶端保證會讀取該更新。複寫其他 服務的更新（索引、搜尋、分析、事件）最終一致。	Amazon DocumentDB 複本最終一致。如果需要立即一致性讀取，用戶端可以從主要執行個體讀取。
複寫	跨資料中心複寫 (XDCR) 提供篩選、主動-被動/主動-主動複寫許多拓撲中的資料。	Amazon DocumentDB 全域叢集在 1：many（最多 10 個）拓撲中提供主動-被動複寫。

探索

遷移至 Amazon DocumentDB 需要徹底了解現有的資料庫工作負載。工作負載探索是分析 Couchbase 叢集組態和操作特性的程序，包括資料集、索引和工作負載，以協助確保無縫轉換，並將中斷降至最低。

叢集組態

Couchbase 使用以服務為中心的架構，其中每個功能都對應至服務。針對您的 Couchbase 叢集執行下列命令，以判斷正在使用的服務（請參閱取得節點資訊）：

```
curl -v -u <administrator>:<password> \
  http://<ip-address-or-hostname>:<port>/pools/nodes | \
  jq '[.nodes[].services[]] | unique'
```

輸出範例：

```
[
  "backup",
  "cbas",
  "eventing",
  "fts",
  "index",
  "kv",
```

```
  "n1ql"
]
```

Couchbase 服務包括下列項目：

資料服務 (kv)

資料服務提供記憶體和磁碟上資料的讀取/寫入存取權。

Amazon DocumentDB 透過 MongoDB API 支援 JSON 資料的 K/V 操作。

查詢服務 (n1ql)

查詢服務支援透過 SQL++ 查詢 JSON 資料。

Amazon DocumentDB 支援透過 MongoDB API 查詢 JSON 資料。

索引服務 （索引）

索引服務會建立和維護資料索引，從而加快查詢速度。

Amazon DocumentDB 支援預設主要索引，並透過 MongoDB API 在 JSON 資料上建立次要索引。

搜尋服務 （英尺）

搜尋服務支援建立全文搜尋的索引。

Amazon DocumentDB 的原生全文搜尋功能可讓您使用 MongoDB API，使用特殊用途文字索引對大型文字資料集執行基本文字查詢。對於模糊搜尋、多語言搜尋和跨集合搜尋等進階功能，請使用 Amazon DocumentDB 與 Amazon OpenSearch Service 的零 ETL 整合。

Analytics 服務 (cbas)

分析服務支援近乎即時地分析 JSON 資料。

Amazon DocumentDB 透過 MongoDB API 支援對 JSON 資料進行臨機操作查詢。您也可以使用在 Amazon EMR 上執行的 Apache Spark，在 Amazon DocumentDB 中對 JSON 資料執行複雜的查詢。

事件服務 （事件）

事件服務會執行使用者定義的商業邏輯，以回應資料變更。

每次資料隨 Amazon DocumentDB 叢集變更時，Amazon DocumentDB 都會叫用 函數，以自動化事件驅動的工作負載。 AWS Lambda Amazon DocumentDB

備份服務 （備份）

備份服務會排程完整和增量資料備份，以及先前資料備份的合併。

Amazon DocumentDB 會以 1–35 天的保留期間持續將您的資料備份至 Amazon S3，讓您可以快速還原至備份保留期間內的任何時間點。Amazon DocumentDB 也會擷取資料的自動快照，做為此持續備份程序的一部分。您也可以使用 管理 Amazon DocumentDB 的備份和還原 AWS Backup。

操作特性

使用適用於 Couchbase 的 Discovery Tool 取得有關資料集、索引和工作負載的下列資訊。此資訊可協助您調整 Amazon DocumentDB 叢集的大小。

資料集

此工具會擷取下列儲存貯體、範圍和集合資訊：

1. 儲存貯體名稱
2. 儲存貯體類型
3. 範圍名稱
4. 集合名稱
5. 總大小 （位元組）
6. 項目總數
7. 項目大小 （位元組）

索引

此工具會擷取所有儲存貯體的下列索引統計資料和所有索引定義。請注意，主要索引會被排除，因為 Amazon DocumentDB 會自動為每個集合建立主要索引。

1. 儲存貯體名稱
2. 範圍名稱
3. 集合名稱
4. 索引名稱

5. 索引大小（位元組）

工作負載

此工具會擷取 K/V 和 N1QL 查詢指標。K/V 指標值會在儲存貯體層級收集，SQL++ 指標則會在叢集層級收集。

工具命令列選項如下所示：

```
python3 discovery.py \
  --username <source cluster username> \
  --password <source cluster password> \
  --data_node <data node IP address or DNS name> \
  --admin_port <administration http REST port> \
  --kv_zoom <get bucket statistics for specified interval> \
  --tools_path <full path to Couchbase tools> \
  --index_metrics <gather index definitions and SQL++ metrics> \
  --indexer_port <indexer service http REST port> \
  --n1ql_start <start time for sampling> \
  --n1ql_step <sample interval over the sample period>
```

以下為範例命令：

```
python3 discovery.py \
  --username username \
  --password ******** \
  --data_node "http://10.0.0.1" \
  --admin_port 8091 \
  --kv_zoom week \
  --tools_path "/opt/couchbase/bin" \
  --index_metrics true \
  --indexer_port 9102 \
  --n1ql_start -60000 \
  --n1ql_step 1000
```

K/V 指標值將以過去一週每 10 分鐘的範例為基礎（請參閱 HTTP 方法和 URI)。SQL++ 指標值將以過去 60 秒內每 1 秒的範例為基礎（請參閱一般標籤)。命令的輸出將位於下列檔案中：

collection-stats.csv – 儲存貯體、範圍和集合資訊

```
bucket,bucket_type,scope_name,collection_name,total_size,total_items,document_size
```

```
beer-sample,membase,_default,_default,2796956,7303,383
gamesim-sample,membase,_default,_default,114275,586,196
pillowfight,membase,_default,_default,1901907769,1000006,1902
travel-sample,membase,inventory,airport,547914,1968,279
travel-sample,membase,inventory,airline,117261,187,628
travel-sample,membase,inventory,route,13402503,24024,558
travel-sample,membase,inventory,landmark,3072746,4495,684
travel-sample,membase,inventory,hotel,4086989,917,4457
...
```

index-stats.csv – 索引名稱和大小

```
bucket,scope,collection,index-name,index-size
beer-sample,_default,_default,beer_primary,468144
gamesim-sample,_default,_default,gamesim_primary,87081
travel-sample,inventory,airline,def_inventory_airline_primary,198290
travel-sample,inventory,airport,def_inventory_airport_airportname,513805
travel-sample,inventory,airport,def_inventory_airport_city,487289
travel-sample,inventory,airport,def_inventory_airport_faa,526343
travel-sample,inventory,airport,def_inventory_airport_primary,287475
travel-sample,inventory,hotel,def_inventory_hotel_city,497125
...
```

kv-stats.csv – 取得、設定和刪除所有儲存貯體的指標

```
bucket,gets,sets,deletes
beer-sample,0,0,0
gamesim-sample,0,0,0
pillowfight,369,521,194
travel-sample,0,0,0
```

n1ql-stats.csv – SQL++ 選取、刪除和插入叢集的指標

```
selects,deletes,inserts
0,132,87
```

indexes-<bucket-name>.txt – 儲存貯體中所有索引的索引定義。請注意，主要索引會被排除，因為 Amazon DocumentDB 會自動為每個集合建立主要索引。

```
CREATE INDEX `def_airportname` ON `travel-sample`(`airportname`)
```

```
CREATE INDEX `def_city` ON `travel-sample`(`city`)
CREATE INDEX `def_faa` ON `travel-sample`(`faa`)
CREATE INDEX `def_icao` ON `travel-sample`(`icao`)
CREATE INDEX `def_inventory_airport_city` ON `travel-
sample`.`inventory`.`airport`(`city`)
CREATE INDEX `def_inventory_airport_faa` ON `travel-
sample`.`inventory`.`airport`(`faa`)
CREATE INDEX `def_inventory_hotel_city` ON `travel-sample`.`inventory`.`hotel`(`city`)
CREATE INDEX `def_inventory_landmark_city` ON `travel-
sample`.`inventory`.`landmark`(`city`)
CREATE INDEX `def_sourceairport` ON `travel-sample`(`sourceairport`)
...
```

規劃

在規劃階段，您將判斷 Amazon DocumentDB 叢集需求，並將 Couchbase 儲存貯體、範圍和集合映射至 Amazon DocumentDB 資料庫和集合。

Amazon DocumentDB 叢集需求

使用探索階段中收集的資料來調整 Amazon DocumentDB 叢集的大小。如需調整 Amazon DocumentDB 叢集大小的詳細資訊，請參閱執行個體大小。 Amazon DocumentDB

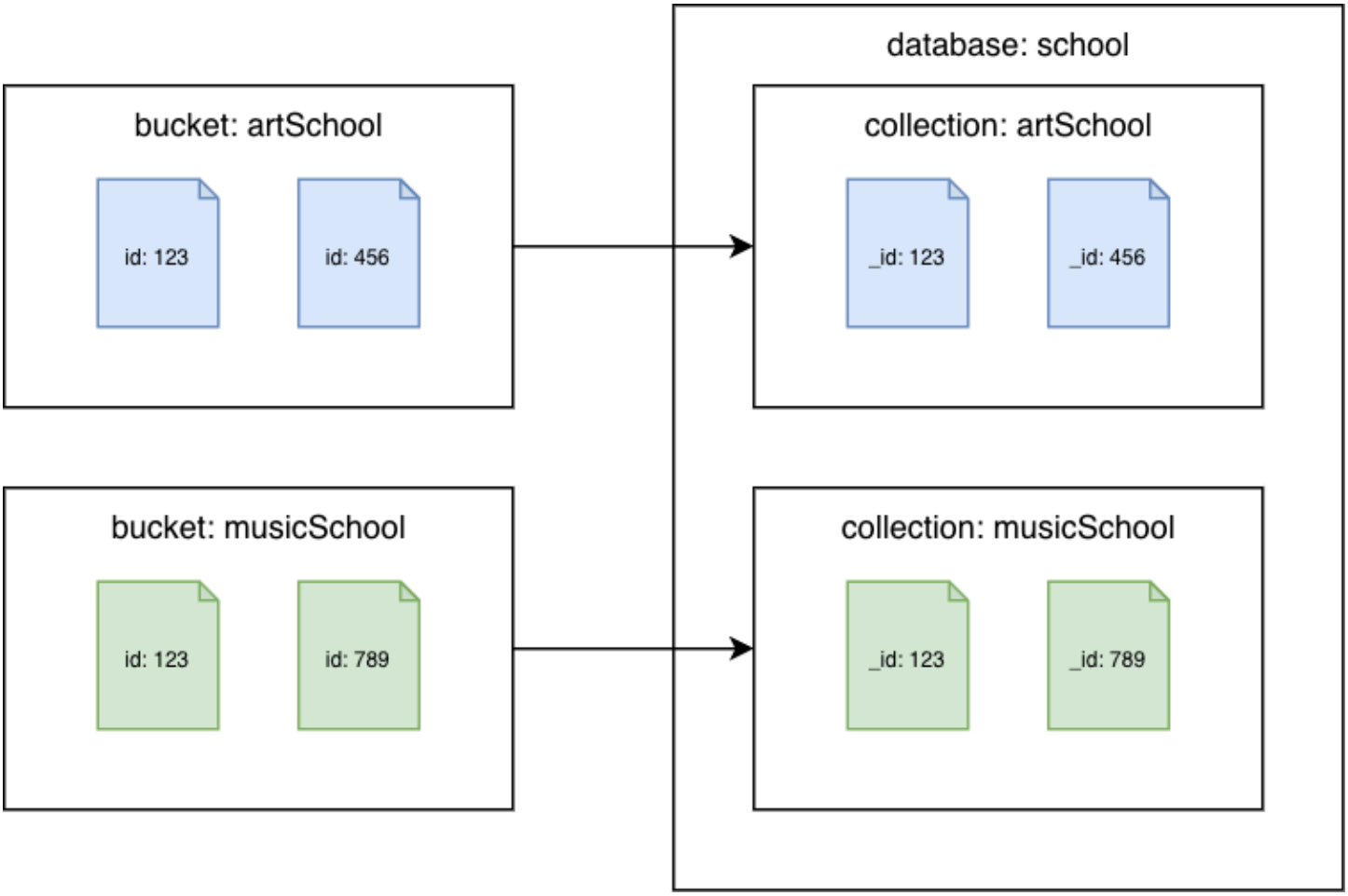
將儲存貯體、範圍和集合映射至資料庫和集合

確定將存在於 Amazon DocumentDB 叢集 (Amazon DocumentDB) 中的資料庫和集合。根據資料在 Couchbase 叢集中的組織方式，考慮下列選項。這些並非唯一的選項，而是提供起點供您考慮。

Couchbase Server 6.x 或更早版本

Amazon DocumentDB 集合的 Couchbase 儲存貯體

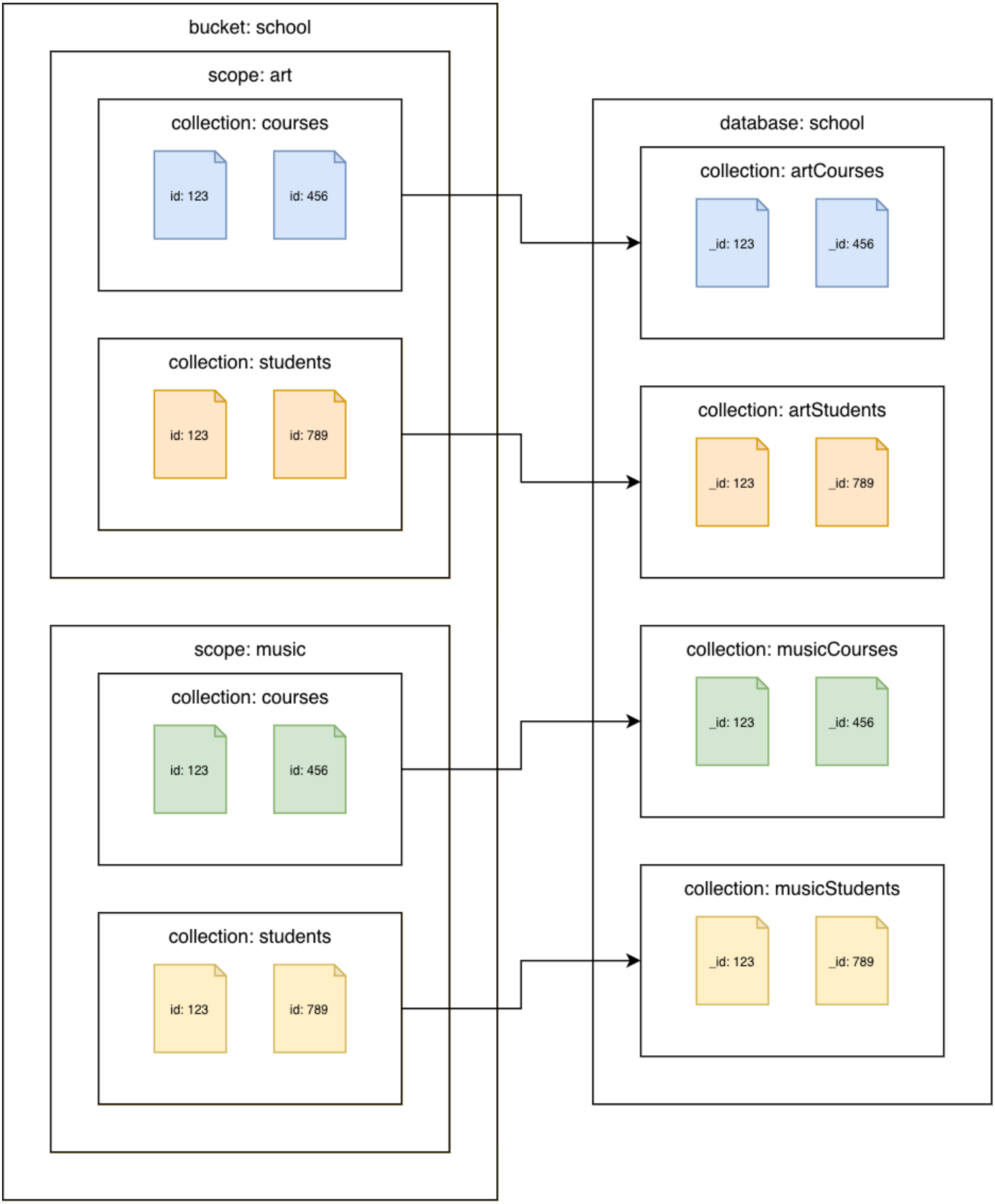
將每個儲存貯體遷移至不同的 Amazon DocumentDB 集合。在此案例中，Couchbase 文件`id`值將用作 Amazon DocumentDB `_id`值。



Couchbase Server 7.0 或更新版本

Amazon DocumentDB 集合的 Couchbase 集合

將每個集合遷移至不同的 Amazon DocumentDB 集合。在此案例中，Couchbase 文件`id`值將用作 Amazon DocumentDB `_id`值。


bucket: school
scope: art
collection: courses
id: 123
id: 456
collection: students
id: 123
id: 789
scope: music
collection: courses
id: 123
id: 456
collection: students
id: 123
id: 789
database: school
collection: artCourses
_id: 123
_id: 456
collection: artStudents
_id: 123
_id: 789
collection: musicCourses
_id: 123
_id: 456
collection: musicStudents
_id: 123
_id: 789

移轉

索引遷移

遷移至 Amazon DocumentDB 不僅需要傳輸資料，還需要建立索引，以維持查詢效能並最佳化資料庫操作。本節概述將索引遷移至 Amazon DocumentDB 時的詳細step-by-step程序，同時確保相容性和效率。

使用 Amazon Q （可在 AWS 管理主控台或 IDE 中使用）將 SQL++ `CREATE INDEX`陳述式轉換為 Amazon DocumentDB `createIndex()`命令。

1. 上傳 Discovery Tool for Couchbase 建立的 indexes-<bucket name>.txt 檔案 (s)。
2. 輸入下列提示：

```
Convert the Couchbase CREATE INDEX statements to Amazon DocumentDB
createIndex commands
```

Amazon Q 將產生同等的 Amazon DocumentDB `createIndex()`命令。請注意，您可能需要根據您將 Couchbase 儲存貯體、範圍和集合映射至 Amazon DocumentDB 集合的方式更新集合名稱。

例如：

indexes-beer-sample.txt

```
CREATE INDEX `beerType` ON `beer-sample`(`type`)
CREATE INDEX `code` ON `beer-sample`(`code`) WHERE (`type` = "brewery")
```

範例 Amazon Q 輸出（摘錄）：

```
db.beerSample.createIndex(
  { "type": 1 },
  {
    "name": "beerType",
    "background": true
  }
)

db.beerSample.createIndex(
  { "code": 1 },
  {
    "name": "code",
```

```
    "background": true,
    "partialFilterExpression": { "type": "brewery" }
  }
)
```

如需 Amazon Q 無法轉換的任何索引，請參閱管理 Amazon DocumentDB 索引和索引和索引屬性以取得詳細資訊。

重構程式碼以使用 MongoDB APIs

用戶端使用 Couchbase SDKs連線到 Couchbase Server。Amazon DocumentDB 用戶端使用 MongoDB 驅動程式連線至 Amazon DocumentDB。MongoDB 驅動程式也支援 Couchbase SDKs支援的所有語言。如需您語言驅動程式的詳細資訊，請參閱 MongoDB 驅動程式。

由於 Couchbase Server 和 Amazon DocumentDB 之間的 APIs 不同，因此您需要重構程式碼，才能使用適當的 MongoDB APIs。您可以使用 Amazon Q 將 K/V API 呼叫和 SQL++ 查詢轉換為同等的 MongoDB APIs：

1. 上傳原始程式碼檔案 (s)。
2. 輸入下列提示：

```
Convert the Couchbase API code to Amazon DocumentDB API code
```

使用 Hello Couchbase Python 程式碼範例，Amazon Q 會產生下列項目 （摘錄）：

```
from pymongo import MongoClient

# Connection parameters
database_name = "travel-sample"

# Connect to Amazon DocumentDB cluster
client = MongoClient('<Amazon DocumentDB connection string>',
                     tls=True,
                     tlsCAFile='global-bundle.pem')

# Get reference to database and collection
db = client['travel-sample']
airline_collection = db['airline']

# upsert document function
def upsert_document(doc):
```

```
    print("\nUpsert Result: ")
    try:
        # key will equal: "airline_8091"
        key = doc["type"] + "_" + str(doc["id"])
        doc['_id'] = key # Amazon DocumentDB uses _id as primary key
        result = airline_collection.update_one(
            {'_id': key},
            {'$set': doc},
            upsert=True
        )
        print(f"Modified count: {result.modified_count}")
    except Exception as e:
        print(e)

# get document function
def get_airline_by_key(key):
    print("\nGet Result: ")
    try:
        result = airline_collection.find_one({'_id': key})
        print(result)
    except Exception as e:
        print(e)

# query for document by callsign
def lookup_by_callsign(cs):
    print("\nLookup Result: ")
    try:
        result = airline_collection.find(
            {'callsign': cs},
            {'name': 1, '_id': 0}
        )
        for doc in result:
            print(doc['name'])
    except Exception as e:
        print(e)

# Test document
airline = {
    "type": "airline",
    "id": 8091,
    "callsign": "CBS",
    "iata": None,
    "icao": None,
    "name": "Couchbase Airways",
```

```
}

upsert_document(airline)
get_airline_by_key("airline_8091")
lookup_by_callsign("CBS")
```

如需在 Python、Node.js、PHP、Go、Java、C#/ 中連線至 Amazon DocumentDB 的範例，請參閱以程式設計方式連線至 Amazon DocumentDB。NET、R 和 Ruby。

選取遷移方法

將資料遷移至 Amazon DocumentDB 時，有兩個選項：

1. 離線遷移
2. 線上遷移

> **Note**
>
> Amazon DocumentDB 使用 _id 欄位做為每個文件的主索引鍵。本指南中的遷移方法會將 Couchbase 文件金鑰存放在 _id 欄位中。對於離線遷移，這會透過 中的 `--include-key _id`選項來完成`cbexport`。對於線上遷移，這會透過 Kafka 連接器的 完成`ProvidedInKeyStrategy`。

離線移轉

在下列情況下，請考慮離線遷移：

- 可接受停機時間：離線遷移涉及停止對來源資料庫的寫入操作、匯出資料，然後將其匯入 Amazon DocumentDB。此程序會為您的應用程式造成停機時間。如果您的應用程式或工作負載可以容忍這段時間無法使用，離線遷移是可行的選項。
- 遷移較小的資料集或執行概念驗證：對於較小的資料集，匯出和匯入程序所需的時間相對較短，使離線遷移成為快速簡單的方法。它也非常適合停機時間較不重要的開發、測試和proof-of-concept環境。
- 簡易性是優先順序：離線方法使用 cbexport 和 mongoimport，通常是遷移資料最直接的方法。它可避免線上遷移方法中涉及的變更資料擷取 (CDC) 的複雜性。
- 不需要複寫正在進行的變更：如果來源資料庫在遷移期間未主動接收變更，或在遷移過程中擷取和套用這些變更到目標並不重要，則離線方法是適當的。

主題

- Couchbase Server 6.x 或更早版本
- Couchbase Server 7.0 或更新版本

Couchbase Server 6.x 或更早版本

Couchbase 儲存貯體到 Amazon DocumentDB 集合

使用 cbexport json 匯出資料，以建立儲存貯體中所有資料的 JSON 傾印。對於 `--format`選項，您可以使用 `lines`或 `list`。

```
cbexport json \
  --cluster <source cluster endpoint> \
  --bucket <bucket name> \
  --format <lines | list> \
  --username <username> \
  --password <password> \
  --output export.json \
  --include-key _id
```

使用 mongoimport 搭配匯入行或清單的適當選項，將資料匯入 Amazon DocumentDB 集合：

行：

```
mongoimport \
  --db <database> \
  --collection <collection> \
  --uri "<Amazon DocumentDB cluster connection string>" \
  --ssl \
  --sslCAFile global-bundle.pem \
  --file export.json
```

清單：

```
mongoimport \
  --db <database> \
  --collection <collection> \
  --uri "<Amazon DocumentDB cluster connection string>" \
  --ssl \
  --sslCAFile global-bundle.pem \
  --jsonArray \
```

```
  --file export.json
```

Couchbase Server 7.0 或更新版本

若要執行離線遷移，請使用 cbexport 和 mongoimport 工具：

具有預設範圍和預設集合的 Couchbase 儲存貯體

使用 cbexport json 匯出資料，以建立儲存貯體中所有集合的 JSON 傾印。對於 `--format`選項，您可以使用 `lines`或 `list`。

```
cbexport json \
  --cluster <source cluster endpoint> \
  --bucket <bucket name> \
  --format <lines | list> \
  --username <username> \
  --password <password> \
  --output export.json \
  --include-key _id
```

使用 mongoimport 搭配匯入行或清單的適當選項，將資料匯入 Amazon DocumentDB 集合：

行：

```
mongoimport \
  --db <database> \
  --collection <collection> \
  --uri "<Amazon DocumentDB cluster connection string>" \
  --ssl \
  --sslCAFile global-bundle.pem \
  --file export.json
```

清單：

```
mongoimport \
  --db <database> \
  --collection <collection> \
  --uri "<Amazon DocumentDB cluster connection string>" \
  --ssl \
  --sslCAFile global-bundle.pem \
  --jsonArray \
  --file export.json
```

Amazon DocumentDB 集合的 Couchbase 集合

使用 cbexport json 匯出資料，以建立每個集合的 JSON 傾印。使用 `--include-data`選項匯出每個集合。對於 `--format`選項，您可以使用 `lines`或 `list`。使用 `--scope-field`和 `--collection-field`選項，將範圍和集合的名稱存放在每個 JSON 文件的指定欄位中。

```
cbexport json \
  --cluster <source cluster endpoint> \
  --bucket <bucket name> \
  --include-data <scope name>.<collection name> \
  --format <lines | list> \
  --username <username> \
  --password <password> \
  --output export.json \
  --include-key _id \
  --scope-field "_scope" \
  --collection-field "_collection"
```

由於 cbexport 將 `_scope`和 `_collection` 欄位新增至每個匯出的文件，因此您可以透過搜尋和取代`sed`、 或任何您偏好的方法，從匯出檔案中的每個文件中移除它們。

使用 mongoimport 搭配匯入行或清單的適當選項，將每個集合的資料匯入 Amazon DocumentDB 集合：

行：

```
mongoimport \
  --db <database> \
  --collection <collection> \
  --uri "<Amazon DocumentDB cluster connection string>" \
  --ssl \
  --sslCAFile global-bundle.pem \
  --file export.json
```

清單：

```
mongoimport \
  --db <database> \
  --collection <collection> \
  --uri "<Amazon DocumentDB cluster connection string>" \
  --ssl \
  --sslCAFile global-bundle.pem \
```

```
  --jsonArray \
  --file export.json
```

線上遷移

當您需要將停機時間降至最低，且持續變更需要近乎即時地複寫至 Amazon DocumentDB 時，請考慮線上遷移。

請參閱如何從 Couchbase 執行即時遷移至 Amazon DocumentDB，以了解如何執行即時遷移至 Amazon DocumentDB。文件會逐步引導您部署解決方案，並將儲存貯體即時遷移至 Amazon DocumentDB 叢集。

主題

- Couchbase Server 6.x 或更早版本
- Couchbase Server 7.0 或更新版本

Couchbase Server 6.x 或更早版本

Couchbase 儲存貯體到 Amazon DocumentDB 集合

Couchbase 的遷移公用程式已預先設定為執行 Couchbase 儲存貯體的線上遷移至 Amazon DocumentDB 集合。查看接收器連接器組態， `document.id.strategy` 參數設定為使用訊息金鑰值作為`_id`欄位值（請參閱接收器連接器 ID 策略屬性）：

```
ConnectorConfiguration:
  document.id.strategy:
 'com.mongodb.kafka.connect.sink.processor.id.strategy.ProvidedInKeyStrategy'
```

Couchbase Server 7.0 或更新版本

具有預設範圍和預設集合的 Couchbase 儲存貯體

Couchbase 的遷移公用程式已預先設定為執行將 Couchbase 儲存貯體線上遷移至 Amazon DocumentDB 集合。查看接收器連接器組態， `document.id.strategy` 參數設定為使用訊息金鑰值作為`_id`欄位值（請參閱接收器連接器 ID 策略屬性）：

```
ConnectorConfiguration:
```

```
  document.id.strategy:
 'com.mongodb.kafka.connect.sink.processor.id.strategy.ProvidedInKeyStrategy'
```

Amazon DocumentDB 集合的 Couchbase 集合

設定來源連接器，將每個範圍內的每個 Couchbase 集合串流至個別主題（請參閱來源組態選項)。例如：

```
ConnectorConfiguration:
  # add couchbase.collections configuration
  couchbase.collections: '<scope 1>.<collection 1>, <scope 1>.<collection 2>, ...'
```

設定接收器連接器以從每個主題串流到個別的 Amazon DocumentDB 集合（請參閱接收器連接器組態屬性)。例如：

```
ConnectorConfiguration:
  # remove collection configuration
  #collection: 'test'

  # modify topics configuration
  topics: '<bucket>.<scope 1>.<collection 1>, <bucket>.<scope 1>.<collection 2>, ...'

  # add topic.override.%s.%s configurations for each topic
  topic.override.<bucket>.<scope 1>.<collection 1>.collection: '<collection>'
  topic.override.<bucket>.<scope 1>.<collection 2>.collection: '<collection>'
```

驗證

本節提供詳細的驗證程序，以在遷移至 Amazon DocumentDB 後驗證資料一致性和完整性。無論遷移方法為何，驗證步驟都適用。

主題

- 確認目標中存在所有集合
- 驗證來源和目標叢集之間的文件計數
- 比較來源和目標叢集之間的文件

確認目標中存在所有集合

Couchbase 來源

選項 1：查詢工作台

```
SELECT RAW `path`
  FROM system:keyspaces
  WHERE `bucket` = '<bucket>'
```

選項 2：cbq 工具

```
cbq \
  -e <source cluster endpoint> \
  -u <username> \
  -p <password> \
  -q "SELECT RAW `path`
       FROM system:keyspaces
       WHERE `bucket` = '<bucket>'"
```

Amazon DocumentDB 目標

mongosh （請參閱連線至 Amazon DocumentDB 叢集）：

```
db.getSiblingDB('<database>')
db.getCollectionNames()
```

驗證來源和目標叢集之間的文件計數

Couchbase 來源

Couchbase Server 6.x 或更早版本

選項 1：查詢工作台

```
SELECT COUNT(*)
FROM `<bucket>`
```

選項 2：cbq

```
cbq \
```

```
  -e <source cluster endpoint> \
  -u <username> \
  -p <password> \
  -q "SELECT COUNT(*)
       FROM `<bucket>`"
```

Couchbase Server 7.0 或更新版本

選項 1：查詢工作台

```
SELECT COUNT(*)
FROM `<bucket>`.`<scope>`.`<collection>`
```

選項 2：cbq

```
cbq \
  -e <source cluster endpoint> \
  -u <username> \
  -p <password> \
  -q "SELECT COUNT(*)
       FROM `<bucket>`.`<scope>`.`<collection>`"
```

Amazon DocumentDB 目標

mongosh （請參閱連線至 Amazon DocumentDB 叢集）：

```
db = db.getSiblingDB('<database>')
db.getCollection('<collection>').countDocuments()
```

比較來源和目標叢集之間的文件

Couchbase 來源

Couchbase Server 6.x 或更早版本

選項 1：查詢工作台

```
SELECT META().id as _id, *
FROM `<bucket>`
LIMIT 5
```

選項 2：cbq

```
cbq \
  -e <source cluster endpoint> \
  -u <username> \
  -p <password> \
  -q "SELECT META().id as _id, *
       FROM `<bucket>`
       LIMIT 5"
```

Couchbase Server 7.0 或更新版本

選項 1：查詢工作台

```
SELECT META().id as _id, *
FROM `<bucket>`.`<scope>`.`<collection>`
LIMIT 5
```

選項 2：cbq

```
cbq \
  -e <source cluster endpoint> \
  -u <username> \
  -p <password> \
  -q "SELECT META().id as _id, *
       FROM `<bucket>`.`<scope>`.`<collection>`
       LIMIT 5"
```

Amazon DocumentDB 目標

mongosh（請參閱連線至 Amazon DocumentDB 叢集）：

```
db = db.getSiblingDB('<database>')
db.getCollection('<collection>').find({
  _id: {
    $in: [
      <_id 1>, <_id 2>, <_id 3>, <_id 4>, <_id 5>
    ]
  }
})
```

Amazon DocumentDB 引擎版本支援日期

您可以使用下列日期來規劃測試和升級週期。

Amazon DocumentDB 主要版本的發行行事曆

Amazon DocumentDB 目前支援下列主要版本。若要升級您的引擎版本，請參閱 升級主要引擎版本或使用 升級 AWS DMS。

引擎版本	版本日期	標準支援結束	延長支援開始（第 1 年定價）	延長支援開始（第 3 年定價）	延長支援結束
3.6 版	2019 年 1 月 9 日	2026 年 3 月 30 日	2026 年 3 月 31 日	2028 年 3 月 31 日	2029 年 3 月 30 日
4.0 版	2020 年 11 月 9 日	N/A	N/A	N/A	N/A
5.0 版	2023 年 3 月 1 日	N/A	N/A	N/A	N/A
8.0 版	2025 年 11 月 14 日	N/A	N/A	N/A	N/A

Amazon DocumentDB 在標準支援下支援多個引擎版本。您可以在標準支援結束日期之後繼續執行版本，需支付延長支援費用。如需詳細資訊，請參閱 Amazon DocumentDB 延伸支援和 Amazon DocumentDB 定價。

Amazon DocumentDB 次要版本的發行行事曆

Amazon DocumentDB 目前支援下列次要版本。發行排程會根據其他功能或修正而有所不同。次要版本可以在對應的主要版本之前終止標準支援。若要升級您的引擎版本，請參閱 升級次要引擎版本。

> **Note**
>
> 從主要 5.0 版開始，可使用次要版本。

引擎版本	版本日期
5.0.0 版 (LTS)	2026 年 2 月 16 日
5.0.1 版本	2026 年 6 月 8 日

Amazon DocumentDB 會將特定版本指定為長期支援 (LTS) 版本。如需詳細資訊，請參閱使用長期支援 (LTS) 版本。

Amazon DocumentDB 就地主要版本升級

您可以執行 Amazon DocumentDB 叢集的就地主要版本升級 (MVU)，同時保留相同的端點、儲存體和標籤。您的應用程式會繼續運作，無需修改。在提供 Amazon DocumentDB 的所有 AWS 區域 中，此功能皆可免費使用。

> **Important**
>
> 您的叢集在升級期間無法使用，並多次重新啟動。開始升級後，請勿連接、讀取或寫入叢集。停機時間取決於集合、索引、資料庫和執行個體的數量。在維護時段或低使用時間執行升級。升級後，您就無法降級至先前的版本。如有需要，您可以將預先升級快照還原至新的叢集。

主題

- 支援的升級路徑
- 先決條件
- 最佳實務
- 升級到 Amazon DocumentDB 8.0 後有何變更
- 從 3.6 或 4.0 升級之叢集的升級後考量事項
- 執行升級
- 疑難排解

支援的升級路徑

下表列出每個支援的就地主要版本升級路徑。您可以選取目標主要 上任何已發佈的次要版本做為目標引擎版本。

來源主要版本	目標主要版本	備註
Amazon DocumentDB 3.6	Amazon DocumentDB 5.0（任何已發佈的次要版本）	如需升級後考量the section called “從 3.6 或 4.0 升級之叢集的升級後考量事項”事項，請參閱 。
Amazon DocumentDB 3.6	Amazon DocumentDB 8.0（任何已發佈的次要版本）	如需升級後考量the section called “升級到 Amazon DocumentDB 8.0 後有何變更”事項，請參閱 the section called “從 3.6 或 4.0 升級之叢集的升級後考量事項”和 。
Amazon DocumentDB 4.0	Amazon DocumentDB 5.0（任何已發佈的次要版本）	如需升級後考量the section called “從 3.6 或 4.0 升級之叢集的升級後考量事項”事項，請參閱 。
Amazon DocumentDB 4.0	Amazon DocumentDB 8.0（任何已發佈的次要版本）	如需升級後考量the section called “升級到 Amazon DocumentDB 8.0 後有何變更”事項，請參閱 the section called “從 3.6 或 4.0 升級之叢集的升級後考量事項”和 。
Amazon DocumentDB 5.0（任何已發佈的次要版本）	Amazon DocumentDB 8.0（任何已發佈的次要版本）	如需功能變更the section called “升級到 Amazon DocumentDB 8.0 後有何變更”，請參閱 。

若要檢視您 AWS 區域中可用的次要版本，請使用 AWS CLI 命令 `aws docdb describe-db-engine-versions`。如需已發行次要版本的清單，請參閱 版本備註。

> **Note**
>
> 每個 MVU 都可以以目的地主要 上發佈的任何次要版本為目標。例如，從 Amazon DocumentDB 3.6 或 4.0 升級可以直接升級至最新發佈的 5.0 或 8.0 次要版本；不需要先升級至 `.0`次要版本，然後套用次要版本升級。
>
> 您可以在單一 MVU 中從 Amazon DocumentDB 3.6 或 4.0 直接升級至 8.0。也支援分階段升級到 5.0（先升級到 5.0 次要版本，然後再升級到 8.0 次要版本）。

> **Note**
>
> 全域叢集或彈性叢集不支援就地 MVU。若要升級全域叢集，請移除次要叢集、將主要叢集轉換為區域叢集、執行 MVU，然後使用相同名稱新增次要叢集來保留端點，以重新建立全域叢集。升級的主要 將資料複寫到新的次要檔案時，會產生 I/O 費用。如需詳細步驟，請參閱從 Amazon DocumentDB 全域叢集移除叢集。

先決條件

> **Important**
>
> 升級之前擴展爆量執行個體。如果您的叢集使用爆量執行個體類型（例如 db.t3.medium或 db.t4g.medium)，請在啟動升級db.r6g.large之前將主要執行個體擴展到至少 db.r5.large或 。爆量執行個體可能沒有足夠的 CPU 和記憶體來完成升級程序，這可能會導致升級失敗和擴充叢集無法使用。您可以在升級完成後縮減規模。
> 升級之前擴展無伺服器寫入器執行個體。如果您的叢集具有無伺服器寫入器執行個體，則在啟動升級之前，ServerlessV2ScalingConfiguration必須將 中的 DCU 上限設為 3 或更高。最大 DCU 低於 3 的叢集無法升級，因為它們沒有足夠的 CPU 或記憶體來完成升級。
> 升級之前檢查您的部分索引（從 Amazon DocumentDB 5.0 升級至 8.0)。開始升級之前，請檢閱partialFilterExpression您使用 $type 運算子來比對特定 BSON 類型的 的部分索引：特別是 binData、bool、decimal、 date double和 array類型別名，或數值類型代碼，例如 { "$type" : 1 }。這類部分索引會導致升級失敗，即使該索引適用於讀取和寫入也一樣正常。使用 db.collection.getIndexes()來檢閱部分索引partialFilterExpression的 。在開始升級之前捨棄受影響的部分索引，並在升級完成後重新建立它。如果您不確定索引是否受到影響，請在升級之前聯絡 AWS 支援。

- 執行個體類型 — Amazon DocumentDB 4.0+ 不支援 db.r4 執行個體。升級之前，請將任何db.r4.*執行個體修改為db.r5.*執行個體或更新版本。請參閱 修改 Amazon DocumentDB 執行個體 和 各區域的支援執行個體類別。
- 作業系統修補程式 — 升級之前，在所有執行個體上套用任何待定的作業系統維護動作。請參閱 Amazon DocumentDB 作業系統更新。

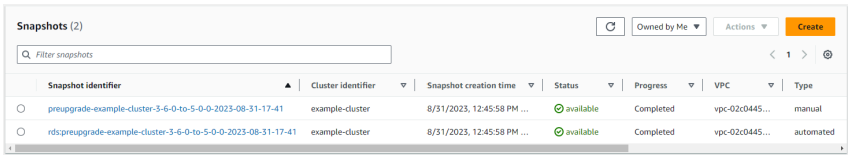
> **Note**
>
> 待定的叢集層級引擎修補程式可能會隱藏執行個體作業系統修補程式。如有需要，請先套用引擎修補程式。請參閱 對叢集的引擎版本執行修補程式更新。

- 高載執行個體 (t-family 執行個體） 的索引限制 — 如果您在高載執行個體上有超過 3，000 個索引，請在升級之前將主要執行個體擴展到至少 db.r5.large。您可以在升級完成後縮減規模。

執行個體	MVU 的索引上限
db.t4g.medium	3K
db.t3.medium	10K

- 參數群組 — 升級之前，請先準備好目標版本的自訂叢集參數群組。如果未指定參數群組，則會使用目標版本的預設參數群組 （ 例如 `default.docdb5.0`或 `default.docdb8.0`)。
- 手動快照 — 升級之前建立手動快照。升級程序會建立名為 的自動快照`preupgrade-<name>-<version>-<timestamp>`，但一律會建立您自己的備份。請參閱 建立手動叢集快照。

> **Note**
>
> 升級程序建立的自動快照不會在就地主要版本升級完成後自動刪除。只要此快照在保留期間內，就不會產生任何費用。驗證叢集成功升級後，您可以選擇刪除此快照。



最佳實務

升級前 - 使用複製進行測試

1. 使用 複製 Amazon DocumentDB 叢集的磁碟區建立叢集的複製。除非您修改複製上的資料，否則不會產生儲存成本。
2. 將複製的執行個體計數與目標叢集相符，以取得逼真的時間預估。

3. 在複製上執行 MVU，並完整測試功能差異。
4. 執行 describe-db-clusters並尋找 ，以檢查是否已排程升級PendingModifiedValues.EngineVersion。如果您已修改叢集並選取在下一個維護時段套用叢集，則無法在主控台中看到排程，但您可以在 中檢視排程 AWS CLI：

```
aws docdb describe-db-clusters \
  --region us-east-1 \
  --db-cluster-identifier mydocdbcluster
```

```
"PendingModifiedValues": {
    "EngineVersion": "5.0.0"
},
```

5. 如果測試成功，請在生產叢集上繼續升級。

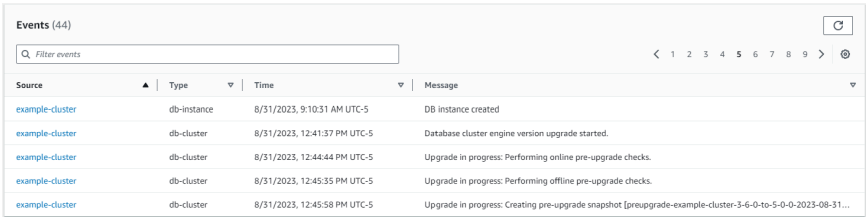
升級期間

您可以訂閱叢集維護事件，監控就地主要版本升級的進度。升級完成時，您將會收到「資料庫叢集主要版本已升級」事件。升級期間發生的此事件和其他事件會顯示在 Amazon DocumentDB 主控台叢集詳細資訊頁面的事件和標籤區段中。接著叢集狀態會從 變更為 upgrading available。

升級期間會產生下列事件：

1. 資料庫叢集引擎主要版本升級已開始。叢集保持線上狀態。
2. 升級準備進行中：開始線上升級預先檢查。
3. 升級準備進行中：已完成線上升級預先檢查。
4. 在主要執行個體完成修補/升級程序時，使資料庫叢集離線。
5. 升級準備進行中：開始離線升級預先檢查。
6. 升級準備進行中：已完成離線升級預先檢查。
7. 升級進行中：建立升級前快照 【preupgrade-<cluster-name>-<version-from>-to-<version-to>-<timestamp>】。
8. 升級進行中：複製磁碟區。
9. 升級進行中：升級寫入器。
10. 升級進行中：升級讀取器。
11. 資料庫叢集引擎主要版本已升級。

事件會顯示在主控台的事件頁面下方：



從 AWS CLI，您可以執行 `aws docdb describe-events`來監控升級進度。若要自動接收通知，請使用 `aws docdb create-event-subscription` 訂閱事件，並將其路由到 Amazon SNS 以進行電子郵件、推送訊息或其他傳遞方法。如需詳細資訊，請參閱訂閱 Amazon DocumentDB 事件。

```
aws docdb describe-events
  --source-identifier mydocdbcluster
  --source-type db-cluster
```

此命令會傳回類似以下的輸出：

```
{
    "Events": [
        {
            "SourceIdentifier": "mydocdbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster engine version upgrade started.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2023-07-11T23:20:32.444000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:xxxx:cluster:mycluster"
        }
    ]
}
```

升級之後

> **Warning**
>
> 在就地主要版本升級之後，Amazon DocumentDB 叢集會立即重新填入索引中繼資料，供資料庫引擎用來最佳化查詢執行計畫。一旦此程序完成，查詢效能就會返回預期的層級。它通常在幾分鐘內完成，但最多可能需要兩個小時，具體取決於叢集上的索引數量。

在此期間，請勿重新啟動、容錯移轉或擴展/縮減您的寫入器執行個體，因為這可能會中斷索引中繼資料的重新計算。請等到您觀察到預期的查詢效能，再進行此類變更。
透過下列叢集事件追蹤進度：

- 升級後叢集狀態：索引中繼資料重新整理程序已啟動
- 升級後叢集狀態：索引中繼資料重新整理程序在 X 秒內完成

如果索引中繼資料重新整理程序未在三小時內完成，或在程序完成後持續遇到效能問題，請聯絡 AWS 支援。

1. 如果您需要還原至升級後狀態，請手動拍攝升級叢集的快照。一旦就地主要版本升級完成，自動快照程序就會繼續。
2. 從 3.6 升級的標籤叢集。新增標籤以區分從 3.6 升級的叢集（請參閱 the section called “從 3.6 或 4.0 升級之叢集的升級後考量事項”)。
3. 更新您的驅動程式。若要使用新功能（例如 8.0 中的定序、檢視或 Zstd 壓縮），請升級至對應的 MongoDB API 版本。如需詳細資訊，請參閱Amazon DocumentDB 8.0 中的新功能。
4. 徹底測試。針對升級的叢集驗證您的應用程式。

升級到 Amazon DocumentDB 8.0 後有何變更

執行主要版本升級至 Amazon DocumentDB 8.0（從 Amazon DocumentDB 5.0 或直接從 3.6 或 4.0) 之後，會啟用或停用下列功能：

- 定序。Amazon DocumentDB 8.0 支援定序。升級之後，新的集合及其索引，以及現有集合上的新索引，預設會啟用定序。
- 文字索引。使用文字索引 V2 建立新的文字索引，該索引使用更新的文字搜尋剖析器來改善 MongoDB 相容性。現有的文字索引不受影響。
- 查詢規劃器版本。如果您沒有自訂參數群組，則會為 Amazon DocumentDB 8.0 建立新的預設參數群組，並自動選取規劃器第 3 版。透過查詢規劃器第 3 版，也可以使用檢視。
- 壓縮。Amazon DocumentDB 8.0 支援使用 Zstd 演算法的字典型文件壓縮。升級之後，系統會使用預設啟用的 Zstd 壓縮建立新的集合。來自 5.0 的現有集合會保留其壓縮設定。若要利用現有集合上的 Zstd 壓縮，您可以修改其壓縮設定。如需詳細資訊，請參閱在 Amazon DocumentDB 8.0 中管理字典型壓縮。

- 索引重建。如果您要從 Amazon DocumentDB 5.0 升級到 Amazon DocumentDB 8.0，則不需要重建索引。如果您要直接從 Amazon DocumentDB 3.6 或 4.0 升級至 8the section called “從 3.6 或 4.0 升級之叢集的升級後考量事項”.0，請如中所述重建索引。

> **Important**
>
> Amazon DocumentDB 8.0 需要 TLS 1.2 或更高版本。不再支援 TLS 1.0 和 TLS 1.1。

> **Note**
>
> 如需功能差異的完整清單，請參閱 Amazon DocumentDB 與 MongoDB 的相容性。

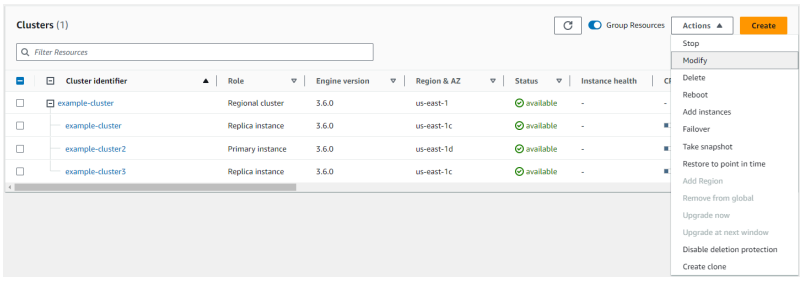
從 3.6 或 4.0 升級之叢集的升級後考量事項

- 索引重建。MVU 會保留原始索引。Amazon DocumentDB 5.0 和更新版本改善了索引維護和垃圾回收，尤其是低基數索引。從 3.6 或 4.0 升級 （ 升級至 5.0 或 8.0) 之後，請重建您的索引以確保最佳的查詢效能 （ 選用，涉及額外的 I/O)。請參閱 使用 的索引維護 `reIndex`。
- 子文件數值比較 （ 僅限 3.6)。從 3.6 升級的叢集會繼承 3.6 行為，其中子文件中的數值類型不會跨類型進行比較。例如， 在 3.6 `{a: {b: 1}}`中`{a: {b: NumberLong(1)}}`不相等，但在 4.0 和更新版本中相等。此行為會影響從 3.6 升級的任何叢集。

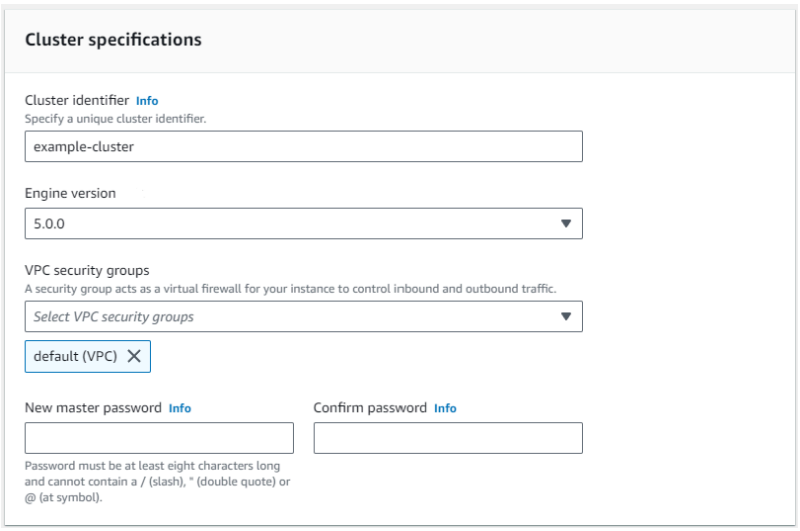
執行升級

Using the AWS 管理主控台

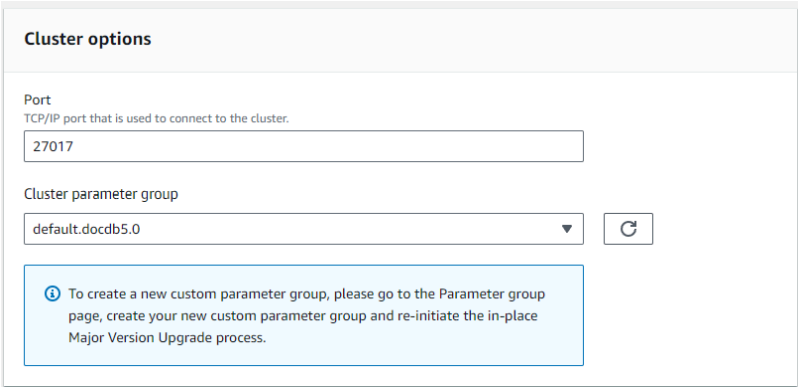
1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。
2. 在叢集表格中，選取來源叢集，選擇動作，然後選擇修改。



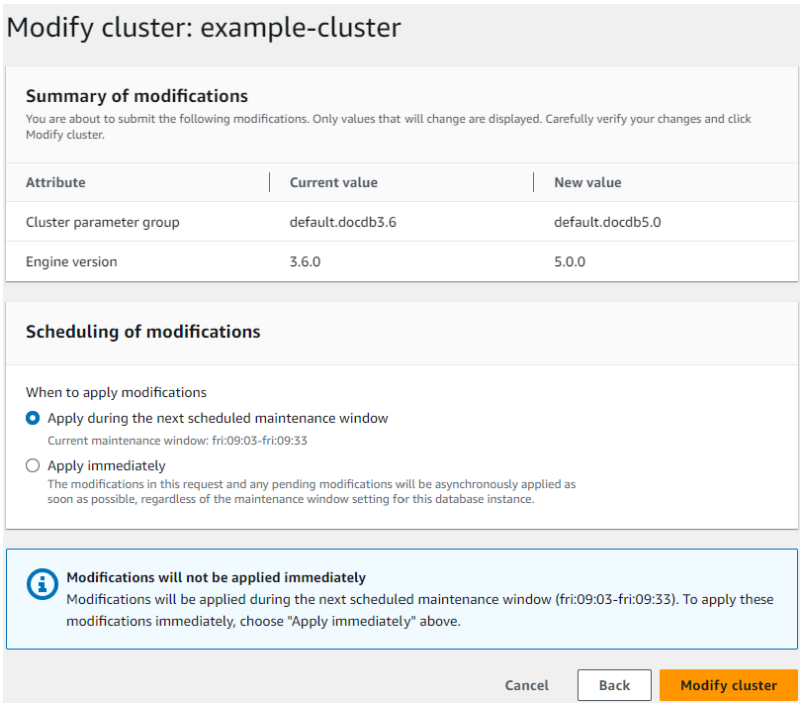
3. 在叢集規格中，從引擎版本下拉式清單中選擇目標版本 （ 例如 5.0.0 或 8.0.0)。



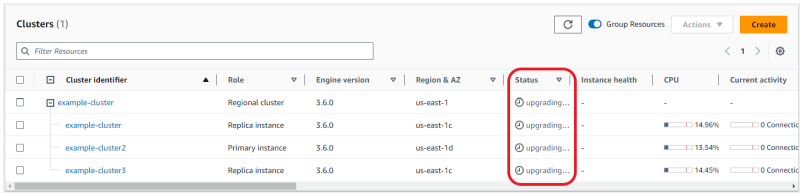
4. 在叢集選項中，選取目標引擎版本的叢集參數群組。您可以使用預設（例如 default.docdb5.0 或 default.docdb8.0) 或您建立的自訂參數群組。



5. 選擇繼續，選取排程偏好設定（立即套用或下一個維護時段），然後選擇修改叢集。



6. 在叢集資料表變更為升級時監控叢集狀態：



Using the AWS CLI

[modify-db-cluster](#) 搭配 `--allow-major-version-upgrade`旗標使用：

```
aws docdb modify-db-cluster \
  --db-cluster-identifier mydocdbcluster \
  --allow-major-version-upgrade \
  --engine-version 8.0.0 \
  --apply-immediately \
  --cluster-parameter-group mydocdbparametergroup \
  --region us-east-1
```

將每個####取代為您叢集的資訊。

疑難排解

- 升級前檢查失敗。升級開始之前，Amazon DocumentDB 會執行升級前驗證檢查。以下是預先檢查失敗的常見原因：
  - 升級無法繼續，因為集合的名稱有 58 個或更多字元 — 在重試升級之前，將受影響的集合重新命名為較短的名稱。
  - 升級無法繼續，因為索引計數超過執行個體類型的限制 — 在重試升級之前升級至較大的執行個體類型。如需依執行個體類型的索引限制，請參閱 先決條件。
- 升級失敗和復原。如果升級失敗，會自動嘗試回復。成功轉返會產生事件：「資料庫叢集處於無法升級的狀態」。您的叢集會返回其升級前狀態，您可以繼續使用它。在重新嘗試之前，請聯絡 AWS 支援部門進行故障診斷。
- 升級後效能。當索引中繼資料重新整理執行時，可能會發生暫時性效能降低和高 CPU 使用率。如果降級持續超過 3 小時，請聯絡 AWS 支援。

如需其他協助，請聯絡 AWS 支援。

Amazon DocumentDB 次要版本升級

> **Note**
>
> 本主題涵蓋相同主要版本中的次要版本升級（例如，從 5.0.0 升級到 5.0.1)。如需有關升級至不同主要版本的資訊，請參閱 Amazon DocumentDB 就地主要版本升級。

您可以透過修改叢集來升級 Amazon DocumentDB 叢集的次要引擎版本（例如，從 5.0.0 升級至 5.0.1)。您可以立即或在下一個維護時段套用次要版本升級。

執行次要版本升級之前

在執行次要版本升級之前，下列動作可能很有幫助：

- 在低流量期間執行升級，以減少任何短暫停機時間的影響。
- 使用 `describe-db-engine-versions`命令，確認目標次要版本是目前版本的有效升級目標。
- 對於全域叢集，次要叢集必須在主要叢集之前升級。如需詳細資訊，請參閱全域叢集修補。

Important

- 升級是單向操作 — 升級後無法降級。
- 在升級過程中，您的叢集將經歷短暫的停機時間。

執行次要版本升級

Using the AWS 管理主控台

若要使用 執行次要版本升級 AWS 管理主控台：

1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。
2. 在叢集表格中，選取您要升級的叢集，選擇動作，然後選擇修改。
3. 在修改叢集對話方塊的叢集規格區段中，從引擎版本下拉式功能表中選擇目標次要版本。
4. 向下捲動並選擇繼續。
5. 在修改排程區段中，選擇立即或在下一個維護時段套用變更。

   然後選擇 Modify cluster (修改叢集)。

Using the AWS CLI

使用 `modify-db-cluster`命令搭配 `--engine-version` 參數：

```
aws docdb modify-db-cluster \
    --db-cluster-identifier mydocdbcluster \
    --engine-version 5.0.1 \
    --apply-immediately \
    --region us-east-1
```

在上述範例中，將每個`#########`取代為您叢集的資訊。

Note

若要在下一個維護時段套用升級，而不是立即套用，請使用 `--no-apply-immediately`取代 `--apply-immediately`。

驗證次要版本升級

升級完成後，請使用 驗證引擎版本 AWS CLI：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier mydocdbcluster \
    --query 'DBClusters[0].EngineVersion'
```

此操作的輸出如下所示：

```
"5.0.1"
```

使用 升級 Amazon DocumentDB 叢集 AWS Database Migration Service

> **Important**
>
> Amazon DocumentDB 不會遵循與 MongoDB 相同的支援生命週期，而 MongoDB end-of-life 排程不適用於 Amazon DocumentDB。

您可以使用 將 Amazon DocumentDB 叢集升級至更高版本，並將停機時間降至最低 AWS DMS。AWS DMS 是一項全受管服務，可讓您輕鬆地從較舊的 Amazon DocumentDB 版本、關聯式資料庫和非關聯式資料庫遷移至目標 Amazon DocumentDB 叢集。

主題

- 步驟 1：啟用變更串流
- 步驟 2：修改變更串流保留期間
- 步驟 3：遷移您的索引
- 步驟 4：建立 AWS DMS 複寫執行個體
- 步驟 5：建立 AWS DMS 來源端點
- 步驟 6：建立 AWS DMS 目標端點
- 步驟 7：建立和執行遷移任務
- 步驟 8：將應用程式端點變更為目標 Amazon DocumentDB 叢集

步驟 1：啟用變更串流

若要執行最短的停機時間遷移， AWS DMS 需要存取叢集的變更串流。Amazon DocumentDB 變更串流提供在叢集集合和資料庫中發生的按時間順序排列的更新事件序列。從變更串流讀取 AWS DMS 可讓 執行變更資料擷取 (CDC)，並將增量更新套用至目標 Amazon DocumentDB 叢集。

若要針對特定資料庫上的所有集合啟用變更串流，請使用 mongo shell 驗證您的 Amazon DocumentDB 叢集，並執行下列命令：

```
db.adminCommand({modifyChangeStreams: 1,
    database: "db_name",
    collection: "",
    enable: true});
```

步驟 2：修改變更串流保留期間

接著，根據您想要在變更串流中保留變更事件的時間長度，修改變更串流保留期間。例如，如果您預期使用 AWS DMS 的 Amazon DocumentDB 叢集遷移需要 12 小時，您應該將變更串流保留設定為大於 12 小時的值。Amazon DocumentDB 叢集的預設保留期間為三小時。您可以使用 AWS 管理主控台或 ，將 Amazon DocumentDB 叢集的變更串流日誌保留期間修改為一小時到七天 AWS CLI。如需詳細資訊，請參閱修改變更串流日誌保留期間。

步驟 3：遷移您的索引

在目標 Amazon DocumentDB 叢集上建立與來源 Amazon DocumentDB 叢集上相同的索引。雖然 AWS DMS 會處理資料遷移，但不會遷移索引。若要遷移索引，請使用 Amazon DocumentDB Index Tool 從來源 Amazon DocumentDB 叢集匯出索引。您可以建立 Amazon DocumentDB 工具 GitHub 儲存庫的複製，並遵循 中的指示來取得工具README.md。您可以從 Amazon EC2 執行個體或在與 Amazon DocumentDB 叢集相同的 Amazon VPC 中執行 AWS Cloud9 的環境執行工具。

在下列範例中，將每個#########取代為您自己的資訊。

下列程式碼會從來源 Amazon DocumentDB 叢集傾印索引：

```
python migrationtools/documentdb_index_tool.py --dump-indexes
--uri mongodb://sample-user:user-password@sample-source-cluster.node.us-
east 1.docdb.amazonaws.com:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false'
--dir ~/index.js/
```

```
2020-02-11 21:51:23,245: Successfully authenticated to database: admin2020-02-11
 21:46:50,432: Successfully connected to instance docdb-40-xx.cluster-xxxxxxxx.us-
east-1.docdb.amazonaws.com:27017
2020-02-11 21:46:50,432: Retrieving indexes from server...2020-02-11 21:46:50,440:
 Completed writing index metadata to local folder: /home/ec2-user/index.js/
```

索引成功匯出後，請在目標 Amazon DocumentDB 叢集中還原這些索引。若要還原您在上一個步驟中匯出的索引，請使用 Amazon DocumentDB Index Tool。下列命令會從指定的目錄還原目標 Amazon DocumentDB 叢集中的索引。

```
python migrationtools/documentdb_index_tool.py --restore-indexes
--uri mongodb://sample-user:user-password@sample-destination-
cluster.node.us-east 1.docdb.amazonaws.com:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false'
--dir ~/index.js/

2020-02-11 21:51:23,245: Successfully authenticated to database: admin2020-02-11
 21:51:23,245: Successfully connected to instance docdb-50-xx.cluster-xxxxxxxx.us-
east-1.docdb.amazonaws.com:27017
2020-02-11 21:51:23,264: testdb.coll: added index: _id
```

若要確認您已正確還原索引，請使用 mongo shell 連線至目標 Amazon DocumentDB 叢集，並列出指定集合的索引。請參閱下列程式碼：

```
mongo --ssl
--host docdb-xx-xx.cluster-xxxxxxxx.us-east-1.docdb.amazonaws.com:27017
--sslCAFile rds-ca-2019-root.pem --username documentdb --password documentdb

db.coll.getIndexes()
```

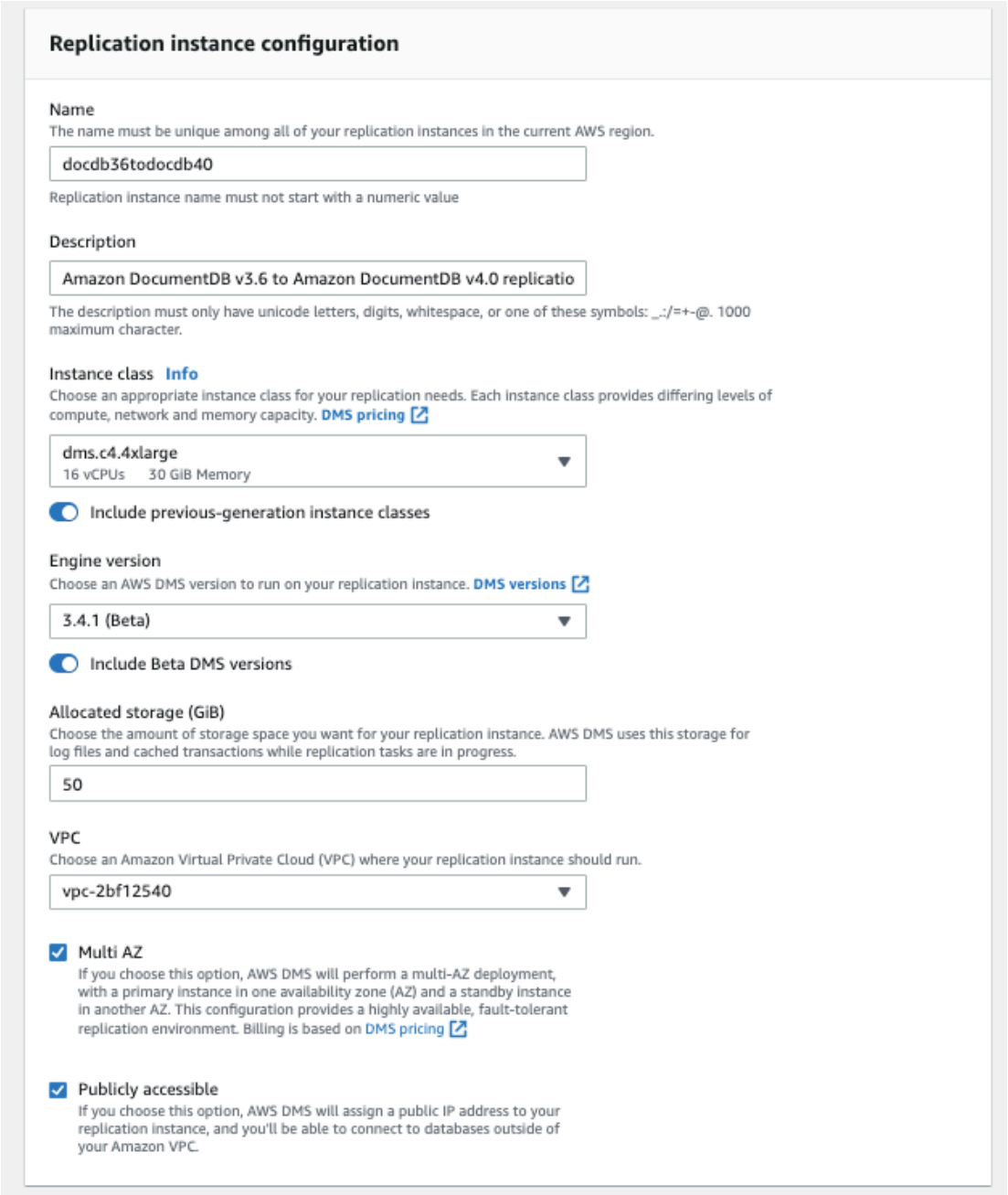
步驟 4：建立 AWS DMS 複寫執行個體

AWS DMS 複寫執行個體會從來源 Amazon DocumentDB 叢集連接和讀取資料，並將其寫入您的目標 Amazon DocumentDB 叢集。 AWS DMS 複寫執行個體可以同時執行大量載入和 CDC 操作。此處理大部分發生在記憶體中。不過，大型操作可能需要在磁碟上進行一些緩衝。快取交易和日誌檔案也會寫入磁碟。一旦遷移資料，複寫執行個體也會串流任何變更事件，以確保來源和目標處於同步狀態。

若要建立 AWS DMS 複寫執行個體：

1. 開啟 AWS DMS 主控台。

2. 在導覽窗格中，選擇 Replication instances (複寫執行個體)。
3. 選擇 Create replication instance (建立複寫執行個體) 並輸入下列資訊：
   - 在名稱中，輸入您選擇的名稱。例如 `docdb36todocdb40`。
   - 在描述中，輸入您選擇的描述。對於 listitem，Amazon DocumentDB 3.6 到 Amazon DocumentDB 4.0 複寫執行個體。
   - 針對執行個體類別，根據您的需求選擇大小。
   - 針對引擎版本，選擇 `3.4.1.`
   - 針對 Amazon VPC，選擇存放來源和目標 Amazon DocumentDB 叢集的 Amazon VPC。
   - 對於配置的儲存體 (GiB)，請使用預設值 50 GiB。如果您有高寫入輸送量工作負載，請增加此值以符合您的工作負載。
   - 對於異地同步備份，如果您需要高可用性和容錯移轉支援，請選擇是。
   - 對於 Publicly accessible (可公開存取)，啟用此選項。



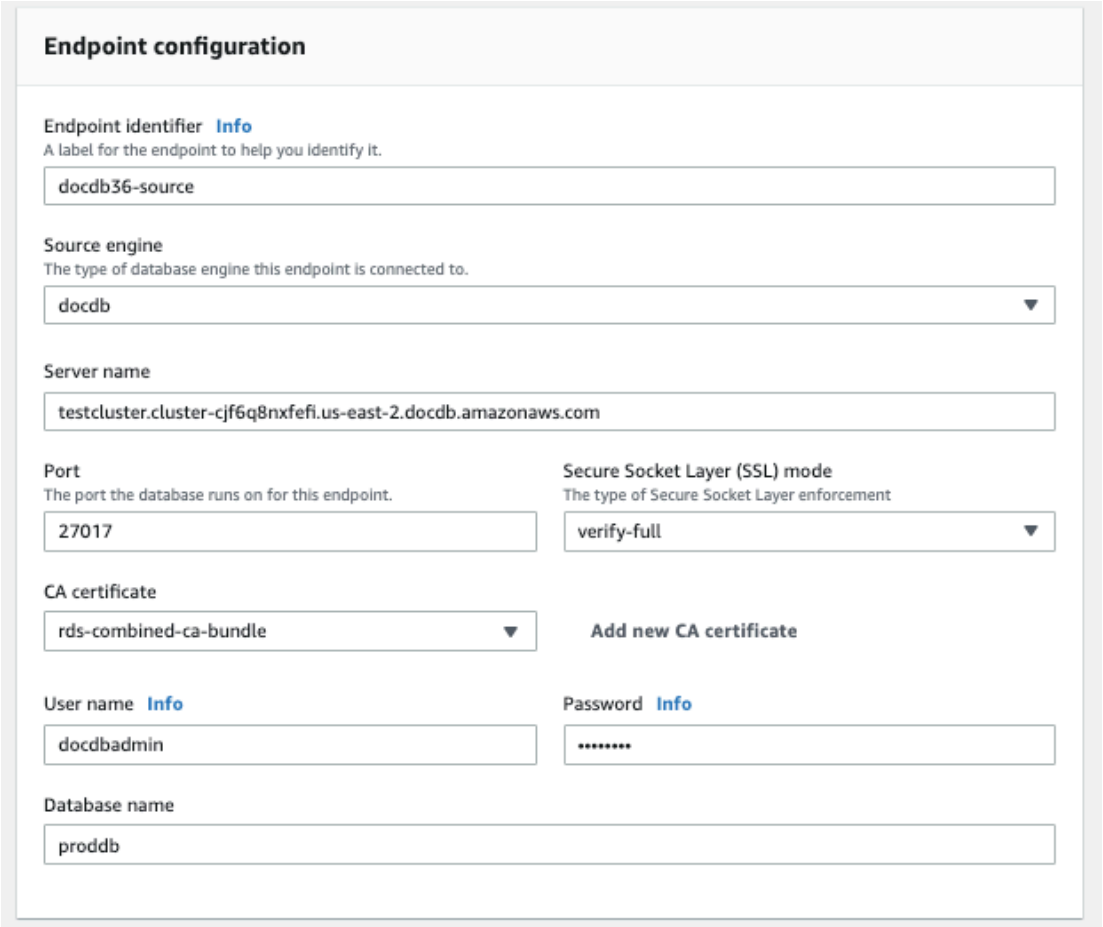
4. 選擇 Create replication instance (建立複寫執行個體)。

步驟 5：建立 AWS DMS 來源端點

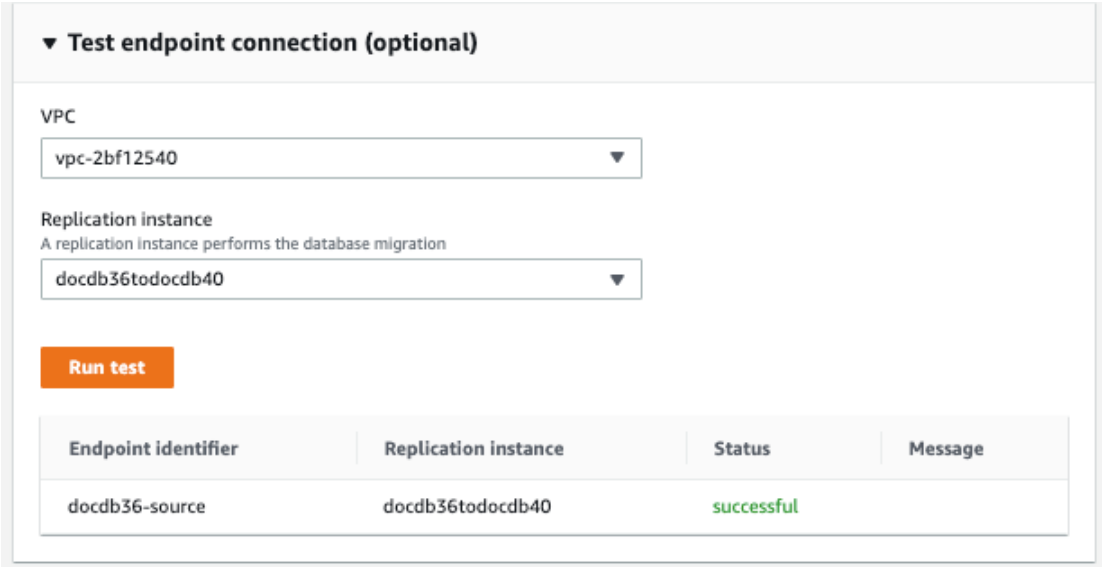
來源端點用於來源 Amazon DocumentDB 叢集。

建立來源端點

1. 開啟 AWS DMS 主控台。
2. 在導覽窗格中選擇端點。
3. 選擇 `Create endpoint`並輸入下列資訊：
   - 對於 Endpoint type (端點類型)，選擇 Source (來源)。
   - >對於端點識別符，輸入易於記住的名稱，例如 `docdb-source`。
   - 針對來源引擎，選擇 `docdb`。
   - 針對伺服器名稱，輸入來源 Amazon DocumentDB 叢集的 DNS 名稱。
   - 在連接埠中，輸入來源 Amazon DocumentDB 叢集的連接埠號碼。
   - 針對 SSL 模式，選擇 `verify-full`。
   - 針對 CA 憑證，選擇新增 CA 憑證。下載新的 CA 憑證 以建立 TLS 連線套件。針對憑證識別符，輸入 `rds-combined-ca-bundle` 。對於 Import certificate file (匯入憑證檔案)，請選擇 Choose file (選擇檔案)，然後瀏覽至您先前下載的 `.pem` 檔案。選取並開啟檔案。選擇匯入憑證，然後從選擇憑證下拉式清單`rds-combined-ca-bundle`中選擇
   - 針對使用者名稱，輸入來源 Amazon DocumentDB 叢集的主要使用者名稱。
   - 針對密碼，輸入來源 Amazon DocumentDB 叢集的主要密碼。
   - 針對資料庫名稱，輸入您要升級的資料庫名稱。



4. 測試您的連線以確認其已成功設定。



5. 選擇建立端點。

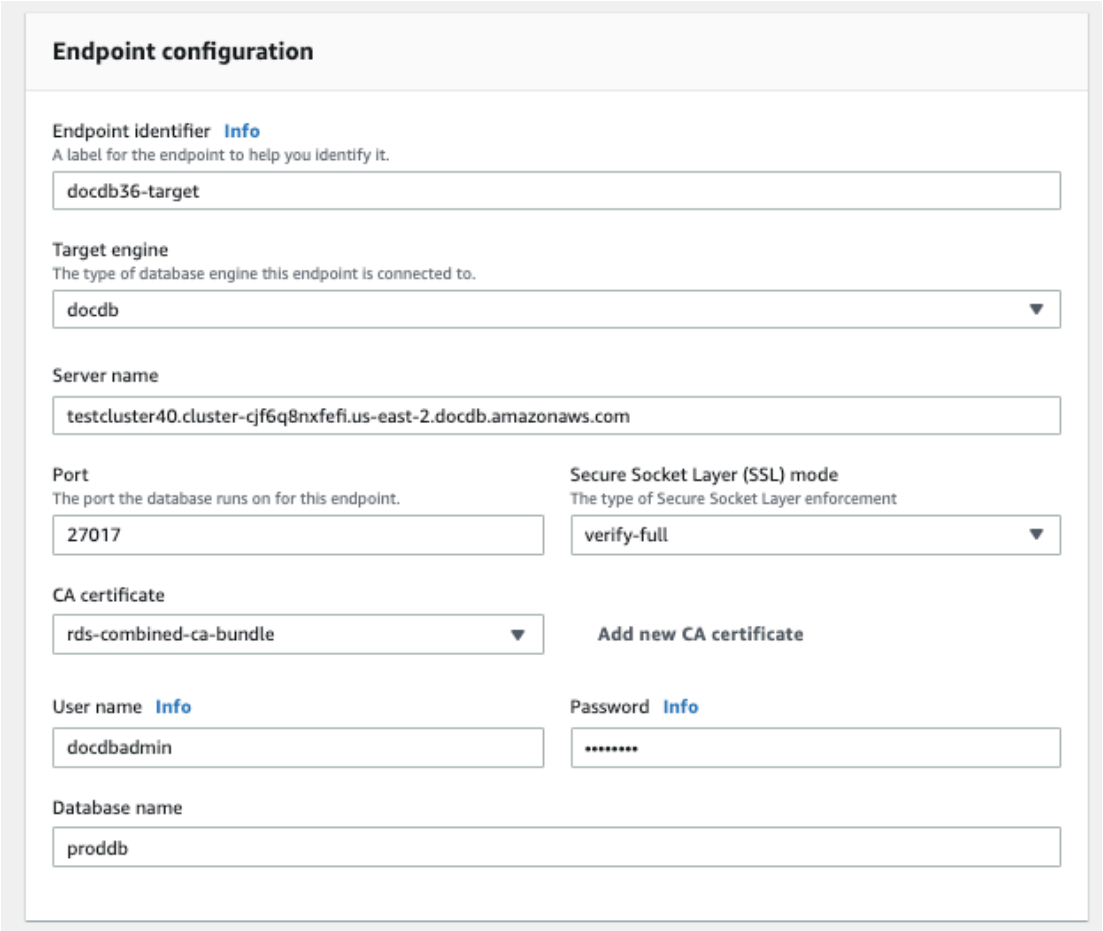
> **Note**
>
> AWS DMS 一次只能遷移一個資料庫。

步驟 6：建立 AWS DMS 目標端點

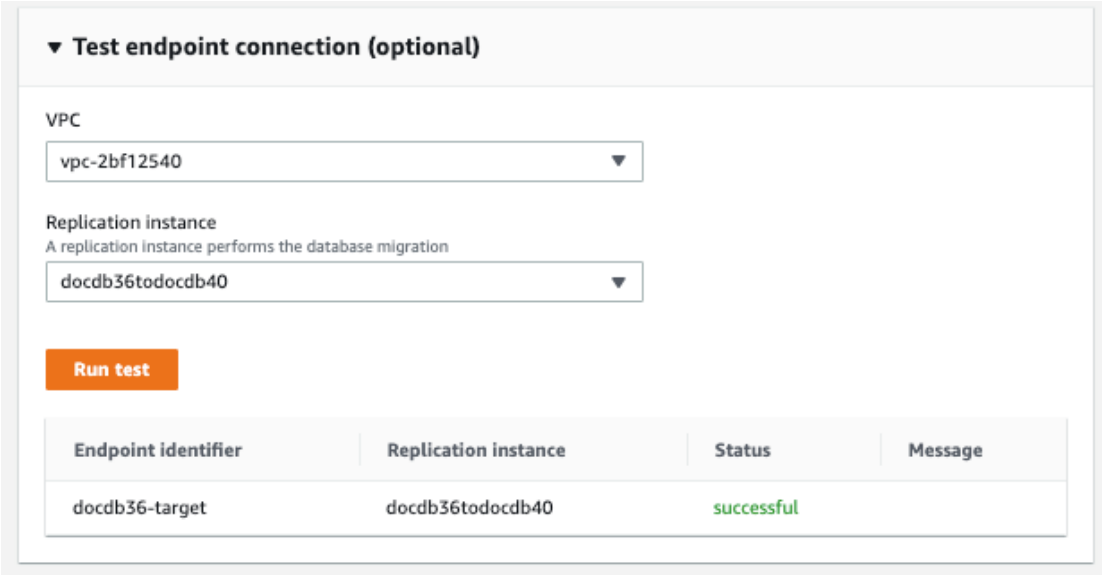
目標端點適用於您的目標 Amazon DocumentDB 叢集。

若要建立目標端點：

1. 開啟 AWS DMS 主控台。
2. 在導覽窗格中選擇端點。
3. 選擇 Create endpoint (建立端點) 並輸入下列資訊：
   - 對於 Endpoint type (端點類型)，選擇 Target (目標)。
   - 對於 Endpoint identifier (端點識別符)，輸入易記的名稱，例如 `docdb-target`。
   - 針對來源引擎，選擇 `docdb`。
   - 針對伺服器名稱，輸入目標 Amazon DocumentDB 叢集的 DNS 名稱。
   - 針對連接埠，輸入目標 Amazon DocumentDB 叢集的連接埠號碼。
   - 針對 SSL 模式，選擇 `verify-full`。
   - 對於 CA 憑證，請從選擇`rds-combined-ca-bundle`憑證下拉式清單中選擇現有憑證。
   - 針對使用者名稱，輸入目標 Amazon DocumentDB 叢集的主要使用者名稱。
   - 針對密碼，輸入目標 Amazon DocumentDB 叢集的主要密碼。
   - 針對資料庫名稱，輸入用於設定來源端點的相同資料庫名稱。



4. 測試您的連線以確認連線已成功設定。



5. 選擇建立端點。

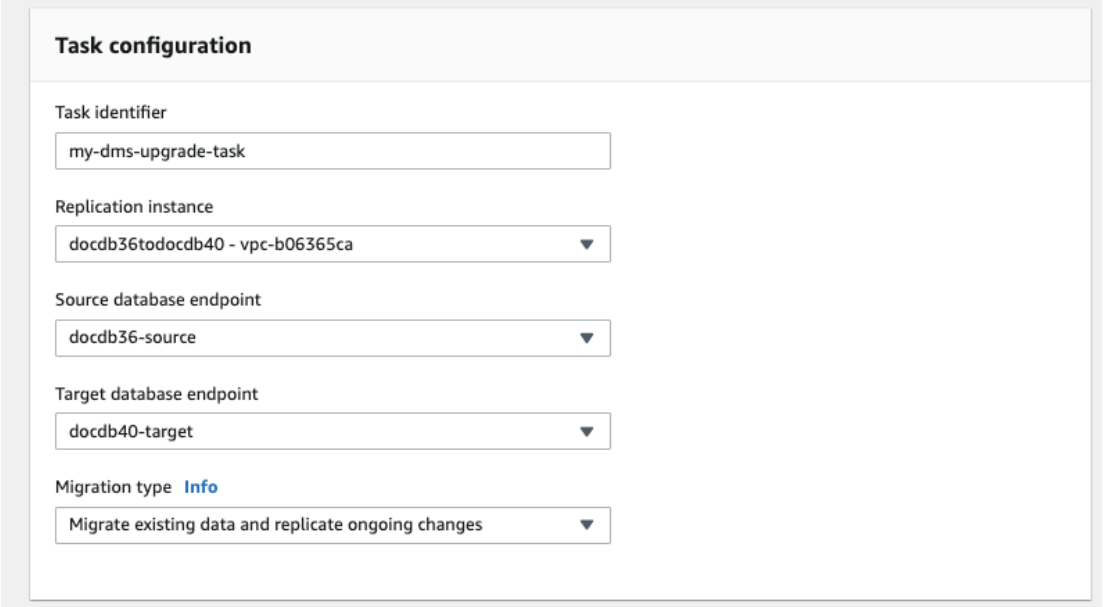
步驟 7：建立和執行遷移任務

AWS DMS 任務會將複寫執行個體與您的來源和目標執行個體繫結。當您建立遷移任務時，您可以指定來源端點、目標端點、複寫執行個體和任何所需的遷移設定。您可以使用三種不同的遷移類型建立 AWS DMS 任務 - 遷移現有資料、遷移現有資料，以及複寫持續變更或僅複寫資料變更。由於本演練的目的是以最短的停機時間升級 Amazon DocumentDB 叢集，因此步驟會利用 選項來遷移現有資料並複寫持續變更。使用此選項時， 會在遷移現有資料時 AWS DMS 擷取變更。即使已載入大量資料， 仍會 AWS DMS 繼續擷取和套用變更。來源和目標資料庫最終將保持同步，最大限度地減少停機時間。

以下是為最小停機時間遷移建立遷移任務的步驟：

1. 開啟 AWS DMS 主控台。
2. 在導覽窗格中，選擇資料庫遷移任務。
3. 選擇建立資料庫遷移任務，然後在任務組態區段中輸入下列資訊：

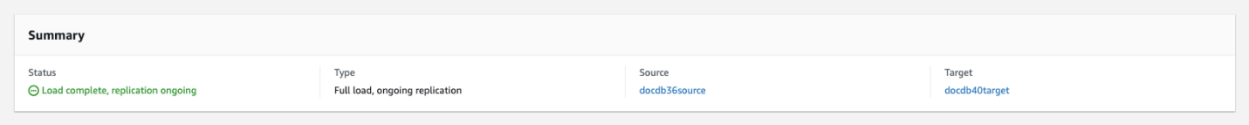
   - 針對任務識別符，輸入易於記住的名稱，例如 `my-dms-upgrade-task`。
   - 針對描述性 Amazon Resource Name (ARN)，輸入易於使用的名稱以覆寫預設 DMS ARN。
   - 對於 Replication instance (複寫執行個體)，選擇您在 步驟 4：建立 AWS DMS 複寫執行個體 中建立的複寫執行個體。
   - 針對來源資料庫端點，選擇您在 中建立的來源端點步驟 5：建立 AWS DMS 來源端點。
   - 針對目標資料庫端點，選擇您在 中建立的目標端點步驟 6：建立 AWS DMS 目標端點。
   - 針對遷移類型，選擇遷移和複寫。



4. 在任務設定區段中輸入下列資訊：

   - 針對目標資料表準備模式區段，選擇不執行任何動作。這將確保不會捨棄步驟 3 中建立的索引。
   - 針對任務日誌子區段，選取開啟 CloudWatch 日誌。
   - 針對遷移任務啟動組態，選擇建立時自動。這會在您建立遷移任務後自動啟動遷移任務。
   - 選擇建立資料庫遷移任務。

AWS DMS 現在開始將資料從來源 Amazon DocumentDB 叢集遷移至目標 Amazon DocumentDB 叢集。任務狀態應該會從開始變更為執行中。您可以在 AWS DMS 主控台中選擇任務來監控進度。幾分鐘/小時後 （取決於遷移的大小），狀態應該會從 變更為載入完成，並持續複寫。這表示 AWS DMS 已經完成來源 Amazon DocumentDB 叢集到目標 Amazon DocumentDB 叢集的完整載入遷移，現在正在複寫變更事件。



最後，您的來源和目標將同步。您可以在集合上執行 `count()`操作，以驗證所有變更事件是否已遷移，藉此驗證它們是否同步。

步驟 8：將應用程式端點變更為目標 Amazon DocumentDB 叢集

完全載入完成且 CDC 程序持續複寫後，您就可以將應用程式的資料庫連線端點從來源 Amazon DocumentDB 叢集變更為目標 Amazon DocumentDB 叢集。

使用長期支援 (LTS) 版本

Amazon DocumentDB 會將特定版本指定為長期支援 (LTS) 版本。與使用非 LTS 版本的叢集相比，使用 LTS 的資料庫叢集在相同版本上可以保留更長的時間，並且進行較少的升級週期。只有在出現關鍵穩定性和安全性修補程式時，才會修補 LTS 版本，而不會引入新功能。可能會釋出其他選用修補程式，例如用於偵錯嚴重失敗的指標。這些會累積套用到所需的修補程式。

目前的 LTS 版本

Amazon DocumentDB 目前的 LTS 版本為：

- 次要版本 5.0.0

若要判斷叢集所在的版本，請參閱 the section called “如何檢查我現有的引擎版本 5.0.0 叢集是否在 LTS 上”。

如需 LTS 版本支援時間表和發行週期的詳細資訊，請參閱 Amazon DocumentDB 引擎版本支援日期。Amazon DocumentDB 將透過標準通訊管道宣布下一個 LTS 版本。

> **Note**
>
> 尚未指定 Amazon DocumentDB 8.0 的 LTS 版本。如果您需要 LTS，請使用引擎版本 5.0.0 搭配修補程式版本 3.0.17983 或更新版本。

誰應該使用 LTS

Amazon DocumentDB LTS 版本專為具有有限升級時段的叢集而設計。LTS 非常適合資料庫可用性直接影響業務營運的生產環境。

考慮 LTS 版本，如果：

- 您的應用程式有嚴格的執行時間需求，限制升級機會
- 您擁有應用程式所需的所有必要資料庫引擎功能和增強功能

如果符合下列條件，請改為考慮標準版本：

- 您的應用程式可以處理最少的中斷
- 您想要在最新功能和增強功能發佈後立即進行

> **Important**
>
> - 升級是單向操作 - 升級後無法降級
> - 在升級過程中，您的叢集將經歷短暫的停機時間

如何建立新的 LTS 叢集

若要建立 LTS 叢集，請遵循下列步驟：

1. 在 管理主控台中，建立新的引擎版本 5.0.0 叢集
2. 使用 建立叢集後，請確認您在引擎版本 5.0.0 上 AWS CLI：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier mydocdbcluster \
    --query 'DBClusters[0].EngineVersion'
```

此操作的輸出如下所示：

```
"5.0.0"
```

如何從引擎版本 3.6.0 或 4.0.0 升級到 5.0.0 LTS 叢集

若要升級至 LTS 叢集，請遵循下列步驟：

1. 在 管理主控台中，將您的引擎版本 3.6.0 或 4.0.0 叢集的主要版本升級 (MVU) 為 5.0.0。如需執行 MVU 的說明，請參閱 Amazon DocumentDB 就地主要版本升級。
2. 使用 升級後，請確認您在引擎版本 5.0.0 上 AWS CLI：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier mydocdbcluster \
    --query 'DBClusters[0].EngineVersion'
```

如何檢查我現有的引擎版本 5.0.0 叢集是否在 LTS 上

若要檢查您的引擎版本 5.0.0 叢集是否正在執行 LTS 版本：

1. 使用 驗證您在引擎版本 5.0.0 上 AWS CLI：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier mydocdbcluster \
    --query 'DBClusters[0].EngineVersion'
```

2. 如有需要，請升級現有的引擎版本 5.0.0 叢集。如需執行修補程式更新的指示，請參閱 對叢集的引擎版本執行修補程式更新。

Amazon DocumentDB 延伸支援

Amazon DocumentDB 延伸支援可讓您在標準支援日期結束後繼續執行引擎版本，但需額外付費。一旦超過標準支援日期，您的叢集就會註冊到延伸支援。如果您在引擎版本標準支援日期結束前未完成升級，您的叢集會以延伸支援費率計費。

主題

- 延伸支援概觀
- 延長支援費用
- 責任

Amazon DocumentDB 延伸支援提供下列更新和技術支援：

- 執行個體或叢集的關鍵和高 CVEs 的安全更新
- 關鍵問題的錯誤修正和修補程式
- 在標準 Amazon DocumentDB 服務水準協議中開啟支援案例並接收疑難排解協助的能力

此付費方案可讓您有更多時間升級至支援的引擎版本。例如，Amazon DocumentDB 3.6 版的標準支援結束日期為 2026 年 3 月 30 日，但您尚未準備好在該日期之前升級至 5.0 或 8.0 版。在此情況下，Amazon DocumentDB 會在 2026 年 3 月 31 日在延伸支援中註冊您的 3.6 版叢集，讓您可以繼續使用 Amazon DocumentDB 3.6 版。從 2026 年 7 月 1 日開始， AWS 會向您收取 Amazon DocumentDB 3.6 版叢集的延長支援費用。

Amazon DocumentDB 延伸支援最長可在引擎版本標準支援日期後 3 年使用。在此期間之後，如果您尚未將引擎版本升級至支援的版本，Amazon DocumentDB 會升級您的引擎版本。盡快升級至支援的引擎版本。

Amazon DocumentDB 3.6 版標準支援結束日期和延長支援日期：

- 3.6 版 — 2019 年 1 月 9 日
- 3.6 版標準支援結束 — 2026 年 3 月 30 日
- 3.6 版延長支援第 1 年定價 — 2026 年 7 月 1 日
- 3.6 版延長支援第 3 年定價 — 2028 年 3 月 31 日
- 3.6 版終止延長支援 — 2029 年 3 月 30 日

延伸支援概觀

如果您的叢集尚未在該版本上，Amazon DocumentDB 會將您的叢集升級至上次發行的引擎版本 （延長支援日期結束之前）。在引擎版本的標準支援日期之後，升級期間才會發生。

如果您已經在該引擎版本上有叢集，則可以使用已達標準支援結束日期的引擎版本建立新的叢集。Amazon DocumentDB 會在 Amazon DocumentDB 延伸支援中註冊這些新叢集，並向您收取此方案的費用。如果您在 3.6 版的標準支援結束日期之前升級到仍在 Amazon DocumentDB 標準支援的引擎版本，則不會向您收取延長支援費用。

您可以隨時在 Amazon DocumentDB 延長支援中結束註冊。若要結束註冊，請將每個註冊的叢集升級至仍在 Amazon DocumentDB 標準支援的較新引擎版本。Amazon DocumentDB 延長支援註冊的結束將在您完成升級至仍在 Amazon DocumentDB 標準支援的較新引擎版本的那一天生效。

如需 Amazon DocumentDB 標準支援結束日期和 Amazon DocumentDB 延長支援結束日期的詳細資訊，請參閱 Amazon DocumentDB 引擎版本支援日期。

延長支援費用

自 2026 年 7 月 1 日起，在延伸支援上執行的所有叢集都會產生費用。如需標準支援結束日期的 Amazon DocumentDB 3.6 版，請參閱 Amazon DocumentDB 引擎版本支援日期。

當您採取下列其中一個動作時，Amazon DocumentDB 延伸支援 的額外費用會停止：

- 升級至標準支援涵蓋的引擎版本
- 刪除標準支援日期結束後執行 3.6 版的叢集

例如，Amazon DocumentDB 3.6 版於 2026 年 3 月 31 日進入延長支援，但在 2026 年 7 月 1 日之前不會開始收費。如果您在 2026 年 7 月 31 日將 Amazon DocumentDB 3.6 版叢集升級至 Amazon DocumentDB 5.0 版，您只需支付 Amazon DocumentDB 3.6 版的 30 天延長支援費用。如需詳細資訊，請參閱 Amazon DocumentDB 定價。

責任

以下是 Amazon DocumentDB 的責任，以及您對 Amazon DocumentDB 延伸支援的責任。

Amazon DocumentDB 責任

在 Amazon DocumentDB 3.6 版的標準支援日期結束後，Amazon DocumentDB 將針對在延伸支援中註冊的 3.6 版叢集套用修補程式、錯誤修正和升級。

您的責任

您有責任在 Amazon DocumentDB 3.6 版延長支援結束日期之前，將引擎升級至較新的引擎版本。如果您未升級叢集，在 Amazon DocumentDB 3.6 版的延長支援日期結束後，Amazon DocumentDB 會嘗試將叢集升級至 Amazon DocumentDB 標準支援支援支援的最新引擎版本。

Amazon DocumentDB 功能和組態

下表比較跨 Amazon DocumentDB 引擎版本和彈性叢集的功能、功能和支援的執行個體類型。使用這些資料表來判斷哪個引擎版本最適合您的工作負載需求。

引擎狀態

各版本的引擎狀態

特徵/功能	v3.6	4.0 版	5.0 版	8.0 版	
支援狀態	EOL：2026 年 3 月	作用中	作用中 (LTS 可用）	作用中 (2025 年 11 月 GA)	
提供延伸支援	是（已支付，直到 2029 年 3 月）	N/A	N/A	N/A	
可用的 LTS 版本	否	否	是 (v5.0 LTS)	否	
彈性叢集	否	否	是	否	

叢集類型

依版本的叢集類型

Cluster Type (叢集類型)	v3.6	4.0 版	5.0 版	8.0 版	彈性叢集
執行個體型叢集	是	是	是	是	是
無伺服器執行個體	否	否	是	是	否

功能

依引擎版本列出的功能和組態

功能	v3.6	4.0 版	5.0 版	8.0 版	彈性叢集
全域叢集	否	是	是	是	否
異地同步備份 (3 AZs同步備份）	是	是	是	是	是（每個碎片）
支援僅供讀取複本	15	15	15	15	每個碎片 15 個
ACID 交易	否	是	是	是	否
Change streams	是	是	是	是	否
向量搜尋	否	否	是	是	否
Performance Insights	是	是	是	是	否
就地主要版本升級 (MVU)	是（至 5.0 或 8.0)	是（至 5.0 或 8.0)	是（至 8.0)	否（僅限目標）	否
用戶端欄位層級加密 (FLE)	否	否	是	是	否
I/O 最佳化儲存	否	否	是	是	否
TLS 憑證輪換（不重新啟動）	是（修補程式 1.0.208662+)	是（修補程式 2.0.10179+)	是（修補程式 3.0.4780+)	是	是
跨區域快照複製	是	是	是	是	是
跨帳戶快照共用	是	是	是	是	否
稽核和慢查詢日誌	是	是	是	是	是

功能	v3.6	4.0 版	5.0 版	8.0 版	彈性叢集
雙堆疊 IPv4/IPv6	否	是（修補程式 2.0.11747 +）	是（修補程式 3.0.15902 +）	是	N/A
定序	否	否	否	是	否
檢視	否	否	否	是	否
壓縮支援	否	否	LZ4	LZ4/Ztsd	否
文字索引	否	否	V1	V2	否
FedRAMP (GovCloud)	否	否	是	是	否

執行個體類型

依版本提供的執行個體類型支援

執行個體類型	v3.6	4.0 版	5.0 版	8.0 版
db.r4（上一代）	是	否	否	否
db.r5	是	是	是	是
db.r6g (Graviton2)	否	是	是	是
db.r6gd (NVMe 後端 Graviton2)	否	否	是	是
db.r8g (Graviton4)	否	否	是	是
db.t3.medium	是	是	是	是
db.t4g.medium (Graviton 2)	否	是	是	是
serverless	否	否	是	是

AWS 整合

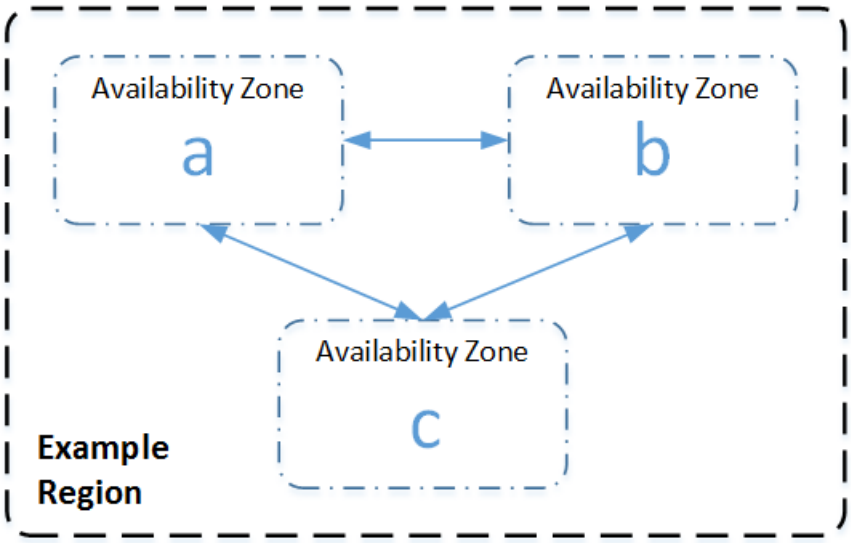
AWS 版本整合支援

AWS 整合/功能	v3.6	4.0 版	5.0 版	8.0 版	彈性叢集
AWS Backup	是	是	是	是	否
AWS Secrets Manager	是	是	是	是	否
AWS Lambda （原生 ESM)	否	是	是	否	否
Amazon OpenSearch Service 擷取 （零 ETL)	否	是	是	是	否
AWS DMS 遷移支援	是	是	是	是 （從 5.0 開始 AWS DMS)	是
AWS Identity and Access Management authentication	否	否	是 (5.0 執行個體型）	是	否
AWS CloudShell	否	否	是	是	否
AWS Key Management Service	是	是	是	是	是
AWS CloudTrail	是	是	是	是	是

選擇區域與可用區域

Amazon 雲端運算資源託管於全球的多個地點。這些位置由 AWS 區域 和 可用區域組成。每個 AWS 區域 都是不同的地理區域。每個區域擁有多個隔離位置，稱為可用區域。Amazon DocumentDB 可讓您在多個位置放置資源，例如執行個體和資料。除非您特別這麼做， AWS 區域 否則不會跨 資源複寫。

Amazon 營運的進階、高度可用的資料中心。儘管故障極為少見，但仍可能影響相同位置內執行個體的可用性。若您將所有執行個體都託管於單一位置，一旦該位置受故障影響，所有執行個體都將無法使用。下圖顯示 AWS 區域 具有三個可用區域的 。



請務必記住，每個區域都是獨立的。您啟動的任何 Amazon DocumentDB 活動（例如，建立執行個體或列出可用的執行個體）只會在您目前的預設值中執行 AWS 區域。您可以在主控台上設定 `EC2_REGION` 環境變數來變更預設區域。或者，您可以使用 AWS CLI中的 `--region` 參數來覆寫它。如需詳細資訊，請參閱設定 AWS Command Line Interface ，特別是環境變數和命令列選項的區段。

當您使用 Amazon DocumentDB 主控台建立叢集，並選擇在不同可用區域中建立複本時，Amazon DocumentDB 會建立兩個執行個體。它會在某一個可用區域中建立主要執行個體，並在另一個可用區域中建立複本執行個體。叢集磁碟區一律複寫到三個可用區域。

若要在特定 中建立或使用 Amazon DocumentDB 執行個體 AWS 區域，請使用對應的區域服務端點。

區域可用性

Amazon DocumentDB 可在下列 AWS 區域使用。

Amazon DocumentDB 支援的區域

區域名稱	區域	可用區域（運算）
美國東部 (俄亥俄)	`us-east-2`	3

區域名稱	區域	可用區域（運算）
美國東部 (維吉尼亞北部)	us-east-1	6
美國西部 (奧勒岡)	us-west-2	4
非洲 (開普敦)	af-south-1	3
南美洲 (聖保羅)	sa-east-1	3
亞太地區 (香港)	ap-east-1	3
亞太地區 (海德拉巴)	ap-south-2	3
亞太地區 (馬來西亞)	ap-southeast-5	3
亞太地區 (孟買)	ap-south-1	3
亞太地區 (大阪)	ap-northeast-3	3
亞太地區 (首爾)	ap-northeast-2	4
亞太地區 (新加坡)	ap-southeast-1	3
亞太地區 (悉尼)	ap-southeast-2	3
亞太地區 (雅加達)	ap-southeast-3	3
亞太地區 (墨爾本)	ap-southeast-4	3
亞太區域 (泰國)	ap-southeast-7	3
亞太地區 (東京)	ap-northeast-1	3
加拿大 (中部)	ca-central-1	3
加拿大西部 (卡加利)	ca-west-1	3
中國 (北京) 區域	cn-north-1	3
中國 (寧夏)	cn-northwest-1	3

區域名稱	區域	可用區域（運算）
歐洲 (法蘭克福)	eu-central-1	3
歐洲 (蘇黎世)	eu-central-2	3
歐洲 (愛爾蘭)	eu-west-1	3
歐洲 (倫敦)	eu-west-2	3
歐洲 (米蘭)	eu-south-1	3
Europe (Paris)	eu-west-3	3
歐洲 (西班牙)	eu-south-2	3
歐洲 (斯德哥爾摩)	eu-north-1	3
墨西哥 (中部)	mx-central-1	3
中東 (阿拉伯聯合大公國)	me-central-1	3
以色列 (特拉維夫)	il-central-1	3
AWS GovCloud（美國西部）	us-gov-west-1	3
AWS GovCloud（美國東部）	us-gov-east-1	3

根據預設，Amazon DocumentDB 叢集的時區為國際標準時間 (UTC)。

如需尋找特定區域中叢集和執行個體連線端點的資訊，請參閱 了解 Amazon DocumentDB 端點。

與 Amazon OpenSearch Service 的零 ETL 整合

主題

- Amazon OpenSearch Service 做為目的地

- 限制

Amazon OpenSearch Service 做為目的地

OpenSearch Service 與 Amazon DocumentDB 整合可讓您將完全載入和變更資料事件串流至 OpenSearch 網域。擷取基礎設施託管為 OpenSearch 擷取管道，並提供大規模、低延遲的機制，以持續從 Amazon DocumentDB 集合串流資料。

在完全載入期間，零 ETL 整合會先使用擷取管道，將歷史完全載入資料擷取至 OpenSearch。擷取完整載入資料後，OpenSearch 擷取管道會開始從 Amazon DocumentDB 變更串流讀取資料，最後追上進度，以在 Amazon DocumentDB 和 OpenSearch 之間維持近乎即時的資料一致性。OpenSearch 會將文件存放在索引中。從 Amazon DocumentDB 集合傳入的資料可以傳送到一個索引，也可以分割成不同的索引。擷取管道將同步 Amazon DocumentDB 集合中的所有建立、更新和刪除事件，作為 OpenSearch 文件的對應建立、更新和刪除，以保持兩個資料系統同步。擷取管道可設定為從一個集合讀取資料，並寫入至一個索引，或從一個集合讀取資料，並依條件路由至多個索引。

可以使用下列方式設定擷取管道，將資料從 Amazon DocumentDB 串流至 Amazon OpenSearch Service：

- 僅限完全載入
- 從 Amazon DocumentDB 串流變更串流事件，無需完全載入
- 完全載入，然後從 Amazon DocumentDB 變更串流

若要設定您的擷取管道，請執行下列步驟：

步驟 1：建立 Amazon OpenSearch Service 網域或 OpenSearch 無伺服器集合

需要具有適當讀取資料許可的 Amazon OpenSearch Service 集合。請參閱《Amazon OpenSearch Service 開發人員指南》中的 Amazon OpenSearch Service 入門OpenSearch或 Amazon OpenSearch Serverless 入門，以建立集合。請參閱《Amazon OpenSearch Service 開發人員指南》中的 Amazon OpenSearch Ingestion，以建立具有正確許可的 AIM 角色，將寫入資料存取至集合或網域。OpenSearch

步驟 2：在 Amazon DocumentDB 叢集上啟用變更串流

確定已在 Amazon DocumentDB 叢集中的必要集合上啟用變更串流。請參閱 搭配 Amazon DocumentDB 使用變更串流，了解詳細資訊。

步驟 3：設定具有寫入 Amazon S3 儲存貯體和目的地網域或集合許可的管道角色

建立 Amazon DocumentDB 集合並啟用變更串流後，請設定您要在管道組態中使用的管道角色，並在角色中新增下列許可：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "allowReadAndWriteToS3ForExport",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket/export/*"
            ]
        }
    ]
}
```

為了讓 OpenSearch 管道將資料寫入 OpenSearch 網域，網域必須具有網域層級存取政策，允許 sts_role_arn 管道角色存取它。下列範例網域存取政策允許您在上一個步驟中建立名為 pipeline-role的管道角色，將資料寫入名為 的網域ingestion-domain：

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{your-account-id}:role/{pipeline-role}"
      },
      "Action": ["es:DescribeDomain", "es:ESHttp*"],
      "Resource": "arn:aws:es:{region}:{your-account-id}:domain/{domain-name}/*"
    }
  ]
```

```
}
```

步驟 4：在管道角色上新增建立 X-ENI 所需的許可

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AttachNetworkInterface",
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DetachNetworkInterface",
                "ec2:DescribeNetworkInterfaces"
            ],
            "Resource": [
                "arn:aws:ec2:*:420497401461:network-interface/*",
                "arn:aws:ec2:*:420497401461:subnet/*",
                "arn:aws:ec2:*:420497401461:security-group/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:Describe*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [ "ec2:CreateTags" ],
            "Resource": "arn:aws:ec2:*:*:network-interface/*",
```

```
            "Condition": {
               "StringEquals": { "aws:RequestTag/OSISManaged": "true" }
            }
         }
      ]
}
```

步驟 5：建立管道

設定指定 Amazon DocumentDB 做為來源的 OpenSearch 擷取管道。此範例管道組態假設使用變更串流擷取機制。如需詳細資訊，請參閱《Amazon OpenSearch Service 開發人員指南》中的搭配使用 OpenSearch Ingestion 管道與 Amazon DocumentDB。 OpenSearch

限制

下列限制適用於 Amazon DocumentDB OpenSearch 整合：

- 僅支援一個 Amazon DocumentDB 集合做為每個管道的來源。
- 不支援跨區域資料擷取。您的 Amazon DocumentDB 叢集和 OpenSearch 網域必須位於相同的 中 AWS 區域。
- 不支援跨帳戶資料擷取。您的 Amazon DocumentDB 叢集和 OpenSearch 擷取管道必須位於相同的 AWS 帳戶中。
- 不支援 Amazon DocumentDB 彈性叢集。僅支援 Amazon DocumentDB 執行個體型叢集。
- 確保 Amazon DocumentDB 叢集已使用 AWS secrets. AWS secrets 啟用身分驗證，這是唯一支援的身分驗證機制。
- 現有的管道組態無法更新，以從不同的資料庫和/或不同的集合擷取資料。若要更新管道的資料庫和/或集合名稱，您必須建立新的管道。

使用 Amazon DocumentDB 進行開發

本節涵蓋連線至 Amazon DocumentDB 叢集、查詢和管理資料、使用索引，以及使用變更串流、交易和向量搜尋等功能來尋找生成式 AI 應用程式。

主題

- 連線至 Amazon DocumentDB
- 使用 Amazon DocumentDB 進行程式設計
- 在 Amazon DocumentDB 中查詢
- 索引
- 搭配 Amazon DocumentDB 使用變更串流
- Amazon DocumentDB 中的交易
- 在 Amazon DocumentDB 8.0 中使用定序
- 在 Amazon DocumentDB 8.0 中使用檢視
- Amazon DocumentDB 生成式人工智慧
- 使用 AWS Lambda 搭配變更串流

連線至 Amazon DocumentDB

Amazon DocumentDB 提供多種連線選項，包括使用 MongoDB 驅動程式的程式設計連線、透過 SSH 通道或 AWS VPN 從 VPC 外部連線，以及 Studio 3T 和 DataGrip 等熱門工具。服務支援複本集連線，以實現高可用性。對於分析和報告，您可以使用 JDBC 或 ODBC 驅動程式進行連線。相同 VPC 中的 EC2 執行個體可以直接連線至 DocumentDB 叢集。所有連線都需要 TLS 和適當的 AWS 登入資料/身分驗證。

主題

- 以程式設計方式連線至 Amazon DocumentDB
- 從 Amazon VPC 外部連線至 Amazon DocumentDB 叢集
- 以複本集的形式連線至 Amazon DocumentDB
- 從 Studio 3T 連線至 Amazon DocumentDB 叢集
- 使用 DataGrip 連線至 Amazon DocumentDB
- 使用 Amazon EC2 連線

- 使用 Amazon DocumentDB JDBC 驅動程式進行連線
- 使用 Amazon DocumentDB ODBC 驅動程式進行連線

以程式設計方式連線至 Amazon DocumentDB

本節包含程式碼範例，示範如何使用數種不同的語言連線至 Amazon DocumentDB （具有 MongoDB 相容性）。範例會根據您連線的叢集已啟用或停用 Transport Layer Security (TLS)，分成兩個部分。根據預設，在 Amazon DocumentDB 叢集上啟用 TLS。但如果您希望，您可以關閉 TLS。如需詳細資訊，請參閱加密傳輸中的資料。

如果您嘗試從叢集所在的 VPC 外部連線至 Amazon DocumentDB，請參閱 從 Amazon VPC 外部連線至 Amazon DocumentDB 叢集。

連線到您的叢集之前，您必須知道叢集是否已啟用 TLS。下節示範如何使用 AWS 管理主控台 或 AWS CLI判斷叢集的 `tls` 參數值。然後，您就可以尋找和套用適當的程式碼範例，繼續作業。

主題

- 判斷tls參數的值
- 使用已啟用的 TLS 連線
- 使用已停用的 TLS 連線
- Amazon DocumentDB 中的網路壓縮

判斷`tls`參數的值

判斷叢集是否已啟用 TLS，是您可以使用 AWS 管理主控台 或 執行的兩個步驟程序 AWS CLI。

1. 判斷管理您叢集的參數群組。

   Using the AWS 管理主控台

   1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
   2. 在左側導覽窗格中選擇叢集。
   3. 在叢集清單中，選取您叢集的名稱。
   4. 產生的頁面會顯示您所選取叢集的詳細資訊。選取 Configuration (組態) 索引標籤。在組態和狀態區段中，找到叢集參數群組下方的參數群組名稱。

Using the AWS CLI

下列 AWS CLI 程式碼會決定管理叢集的參數。請確定以您叢集的名稱取代 *sample-cluster*。

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterIdentifier,DBClusterParameterGroup]'
```

此操作的輸出如下所示：

```
[
       [
           "sample-cluster",
           "sample-parameter-group"
       ]
]
```

2. 判斷您叢集參數群組的 **tls** 參數值。

Using the AWS 管理主控台

1. 在導覽窗格中，選擇 Parameter groups (參數群組)。
2. 在叢集參數群組視窗中，從步驟 1d 選取叢集參數群組名稱。
3. 產生的頁面會顯示您叢集參數群組的參數。您可以在這裡看到 `tls` 的參數值。如需修改此參數的資訊，請參閱修改 Amazon DocumentDB 叢集參數群組。

Using the AWS CLI

您可以使用 `describe-db-cluster-parameters` AWS CLI 命令來檢視叢集參數群組中參數的詳細資訊。

- **`--describe-db-cluster-parameters`** — 列出參數群組內的所有參數及其值。
  - **`--db-cluster-parameter-group name`** – 必要。您叢集參數群組的名稱。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

此操作的輸出如下所示：

```
{
        "Parameters": [
            {
                "ParameterName": "profiler_threshold_ms",
                "ParameterValue": "100",
                "Description": "Operations longer than profiler_threshold_ms
 will be logged",
                "Source": "system",
                "ApplyType": "dynamic",
                "DataType": "integer",
                "AllowedValues": "50-2147483646",
                "IsModifiable": true,
                "ApplyMethod": "pending-reboot"
            },
            {
                "ParameterName": "tls",
                "ParameterValue": "disabled",
                "Description": "Config to enable/disable TLS",
                "Source": "user",
                "ApplyType": "static",
                "DataType": "string",
                "AllowedValues": "disabled,enabled,fips-140-3",
                "IsModifiable": true,
                "ApplyMethod": "pending-reboot"
            }
        ]
}
```

> **Note**
>
> Amazon DocumentDB 支援從這些區域中 Amazon DocumentDB 5.0（引擎版本 3.0.3727) 叢集開始的 FIPS 140-3 端點：ca-central-1、us-west-2、us-east-1、us-east-2、us-gov-east-1、us-gov-west-1。

判斷 tls 參數值之後，請使用後續章節的其中一個程式碼範例來繼續連線您的叢集。

- 使用已啟用的 TLS 連線
- 使用已停用的 TLS 連線

使用已啟用的 TLS 連線

若要以程式設計方式檢視連線至已啟用 TLS 的 Amazon DocumentDB 叢集的程式碼範例，請為您要使用的語言選擇適當的索引標籤。

若要加密傳輸中的資料，global-bundle.pem請使用下列操作下載名為 的 Amazon DocumentDB 公有金鑰。

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

若您的應用程式是位在 Microsoft Windows 上並且需要 PKCS7 檔案，您可以下載 PKCS7 憑證套件。此套件同時包含中繼憑證和根憑證 (下載連結：https://truststore.pki.rds.amazonaws.com/global/global-bundle.p7b)。

Python

下列程式碼示範如何在啟用 TLS 時使用 Python 連線至 Amazon DocumentDB。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
import pymongo
import sys

##Create a MongoDB client, open a connection to Amazon DocumentDB as a replica set
 and specify the read preference as secondary preferred
client = pymongo.MongoClient('mongodb://sample-user:password@sample-
cluster.node.us-east-1.docdb.amazonaws.com:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

##Specify the database to be used
db = client.sample_database

##Specify the collection to be used
col = db.sample_collection

##Insert a single document
```

```
col.insert_one({'hello':'Amazon DocumentDB'})

##Find the document that was previously written
x = col.find_one({'hello':'Amazon DocumentDB'})

##Print the result to the screen
print(x)

##Close the connection
client.close()
```

Node.js

下列程式碼示範如何在啟用 TLS 時使用 Node.js 連線至 Amazon DocumentDB。

> **Important**
>
> 6.13.1 版之前的 Node.js 驅動程式有已知的限制，Amazon DocumentDB 的 IAM 身分驗證目前不支援此限制。使用 Node.js 驅動程式的 Node.js 驅動程式和工具（例如 mongosh）必須升級為使用 Node.js 驅動程式 6.13.1 版或更新版本。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
var MongoClient = require('mongodb').MongoClient

//Create a MongoDB client, open a connection to DocDB; as a replica set,
//  and specify the read preference as secondary preferred

var client = MongoClient.connect(
'mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/sample-database?
tls=true&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false',
{
  tlsCAFile: `global-bundle.pem` //Specify the DocDB; cert
},
function(err, client) {
    if(err)
        throw err;

    //Specify the database to be used
    db = client.db('sample-database');
```

```
    //Specify the collection to be used
    col = db.collection('sample-collection');

    //Insert a single document
    col.insertOne({'hello':'Amazon DocumentDB'}, function(err, result){
      //Find the document that was previously written
      col.findOne({'hello':'Amazon DocumentDB'}, function(err, result){
        //Print the result to the screen
        console.log(result);

        //Close the connection
        client.close()
      });
   });
});
```

PHP

下列程式碼示範如何在啟用 TLS 時使用 PHP 連線至 Amazon DocumentDB。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
<?php
//Include Composer's autoloader
require 'vendor/autoload.php';

$TLS_DIR = "/home/ubuntu/global-bundle.pem";

//Create a MongoDB client and open connection to Amazon DocumentDB
$client = new MongoDB\Client("mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/?retryWrites=false", ["tls" => "true", "tlsCAFile"
 => $TLS_DIR ]);

//Specify the database and collection to be used
$col = $client->sampledatabase->samplecollection;

//Insert a single document
$result = $col->insertOne( [ 'hello' => 'Amazon DocumentDB'] );

//Find the document that was previously written
$result = $col->findOne(array('hello' => 'Amazon DocumentDB'));

//Print the result to the screen
```

```
print_r($result);
?>
```

Go

下列程式碼示範如何在啟用 TLS 時使用 Go 連線至 Amazon DocumentDB。

> **Note**
>
> 從 1.2.1 版開始，MongoDB Go 驅動程式將只使用 `sslcertificateauthorityfile` 中找到的第一個 CA 伺服器憑證。下列範例程式碼透過手動將 `sslcertificateauthorityfile` 中找到的所有伺服器憑證附加到用戶端建立期間使用的自訂 TLS 組態中，來因應此限制。

在下列範例中，將每個`#########`取代為您叢集的資訊。

```
package main

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

 "go.mongodb.org/mongo-driver/bson"
 "go.mongodb.org/mongo-driver/mongo"
 "go.mongodb.org/mongo-driver/mongo/options"

 "io/ioutil"
 "crypto/tls"
 "crypto/x509"
 "errors"
)

const (
 // Path to the AWS CA file
 caFilePath = "global-bundle.pem"

 // Timeout operations after N seconds
 connectTimeout  = 5
 queryTimeout    = 30
```

```
 username        = "sample-user"
 password        = "password"
 clusterEndpoint = "sample-cluster.node.us-east-1.docdb.amazonaws.com:27017"

 // Which instances to read from
 readPreference = "secondaryPreferred"

 connectionStringTemplate = "mongodb://%s:%s@%s/sample-database?
tls=true&replicaSet=rs0&readpreference=%s"
)

func main() {

 connectionURI := fmt.Sprintf(connectionStringTemplate, username, password,
 clusterEndpoint, readPreference)

 tlsConfig, err := getCustomTLSConfig(caFilePath)
 if err != nil {
  log.Fatalf("Failed getting TLS configuration: %v", err)
 }

 client, err :=
 mongo.NewClient(options.Client().ApplyURI(connectionURI).SetTLSConfig(tlsConfig))
 if err != nil {
  log.Fatalf("Failed to create client: %v", err)
 }

 ctx, cancel := context.WithTimeout(context.Background(),
 connectTimeout*time.Second)
 defer cancel()

 err = client.Connect(ctx)
 if err != nil {
  log.Fatalf("Failed to connect to cluster: %v", err)
 }

 // Force a connection to verify our connection string
 err = client.Ping(ctx, nil)
 if err != nil {
  log.Fatalf("Failed to ping cluster: %v", err)
 }

 fmt.Println("Connected to DocumentDB!")
```

```
 collection := client.Database("sample-database").Collection("sample-collection")

 ctx, cancel = context.WithTimeout(context.Background(), queryTimeout*time.Second)
 defer cancel()

 res, err := collection.InsertOne(ctx, bson.M{"name": "pi", "value": 3.14159})
 if err != nil {
  log.Fatalf("Failed to insert document: %v", err)
 }

 id := res.InsertedID
 log.Printf("Inserted document ID: %s", id)

 ctx, cancel = context.WithTimeout(context.Background(), queryTimeout*time.Second)
 defer cancel()

 cur, err := collection.Find(ctx, bson.D{})

 if err != nil {
  log.Fatalf("Failed to run find query: %v", err)
 }
 defer cur.Close(ctx)

 for cur.Next(ctx) {
  var result bson.M
  err := cur.Decode(&result)
  log.Printf("Returned: %v", result)

  if err != nil {
   log.Fatal(err)
  }
 }

 if err := cur.Err(); err != nil {
  log.Fatal(err)
 }

}

func getCustomTLSConfig(caFile string) (*tls.Config, error) {
 tlsConfig := new(tls.Config)
 certs, err := ioutil.ReadFile(caFile)

 if err != nil {
```

```
 return tlsConfig, err
}

tlsConfig.RootCAs = x509.NewCertPool()
ok := tlsConfig.RootCAs.AppendCertsFromPEM(certs)

if !ok {
 return tlsConfig, errors.New("Failed parsing pem file")
}

return tlsConfig, nil
```

Java

從 Java 應用程式連線至已啟用 TLS 的 Amazon DocumentDB 叢集時，您的程式必須使用 AWS提供的憑證授權機構 (CA) 檔案來驗證連線。若要使用 Amazon RDS CA 憑證，請執行下列動作：

1. 從 下載 https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem Amazon RDS CA 檔案。
2. 執行以下命令，建立將憑證授權機構憑證包含在檔案的信任存放區。請務必將 *truststore-password* 變更為其他項目。如果您正在存取同時包含舊憑證授權機構憑證 (rds-ca-2015-root.pem) 和新憑證授權機構憑證 (rds-ca-2019-root.pem) 的信任存放區，您可以將憑證套件匯入信任存放區。

   以下範例 Shell 指令碼將憑證套件匯入 Linux 作業系統上的信任存放區。在下列範例中，將每個*#########*取代為您自己的資訊。最值得注意的是，只要範例目錄「*mydir*」位於指令碼中，將其取代為您為此任務建立的目錄。

   ```
   mydir=/tmp/certs
   truststore=${mydir}/rds-truststore.jks
   storepassword=truststore-password

   curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem" >
    ${mydir}/global-bundle.pem
   awk 'split_after == 1 {n++;split_after=0} /-----END CERTIFICATE-----/
    {split_after=1}{print > "rds-ca-" n ".pem"}' < ${mydir}/global-bundle.pem

   for CERT in rds-ca-*; do
     alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /
   Subject:/; s/.*(CN=|CN = )//; print')
     echo "Importing $alias"
   ```

```
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep
 Alias | cut -d " " -f3- | while read alias
do
   expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword}
 -alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print
 "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

以下是範例 Shell 指令碼，會將憑證套件匯入 macOS 上的信任存放區。

```
mydir=/tmp/certs
truststore=${mydir}/rds-truststore.jks
storepassword=truststore-password

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem" >
 ${mydir}/global-bundle.pem
split -p "-----BEGIN CERTIFICATE-----" ${mydir}/global-bundle.pem rds-ca-

for CERT in rds-ca-*; do
  alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /
Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "
```

```
keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep
 Alias | cut -d " " -f3- | while read alias
do
   expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword}
 -alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print
 "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

3. 在連線到 Amazon DocumentDB 叢集之前，請在應用程式中設定下列系統屬性，以在keystore程式中使用 。

```
javax.net.ssl.trustStore: truststore
javax.net.ssl.trustStorePassword: truststore-password;
```

4. 下列程式碼示範如何在啟用 TLS 時使用 Java 連線至 Amazon DocumentDB。

   在下列範例中，將每個#########取代為您叢集的資訊。

```
package com.example.documentdb;

import com.mongodb.client.*;
import org.bson.Document;

public final class Test {
    private Test() {
    }
    public static void main(String[] args) {

        String template = "mongodb://%s:%s@%s/sample-database?
ssl=true&replicaSet=rs0&readpreference=%s";
        String username = "sample-user";
        String password = "password";
        String clusterEndpoint = "sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017";
        String readPreference = "secondaryPreferred";
        String connectionString = String.format(template, username, password,
 clusterEndpoint, readPreference);

        String truststore = "truststore";
        String truststorePassword = "truststore-password";

        System.setProperty("javax.net.ssl.trustStore", truststore);
```

```
        System.setProperty("javax.net.ssl.trustStorePassword",
 truststorePassword);

        MongoClient mongoClient = MongoClients.create(connectionString);

        MongoDatabase testDB = mongoClient.getDatabase("sample-database");
        MongoCollection<Document> numbersCollection =
 testDB.getCollection("sample-collection");

        Document doc = new Document("name", "pi").append("value", 3.14159);
        numbersCollection.insertOne(doc);

        MongoCursor<Document> cursor = numbersCollection.find().iterator();
        try {
            while (cursor.hasNext()) {
                System.out.println(cursor.next().toJson());
            }
        } finally {
            cursor.close();
        }

    }
}
```

C# / .NET

下列程式碼示範如何在啟用 TLS 時使用 C# / .NET 連線至 Amazon DocumentDB。

在下列範例中，將每個*########*取代為您叢集的資訊。

```
using System;
using System.Text;
using System.Linq;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;
using System.Net.Security;
using MongoDB.Driver;
using MongoDB.Bson;

namespace DocDB
{
    class Program
```

```
    {
        static void Main(string[] args)
        {
            string template = "mongodb://{0}:{1}@{2}/sampledatabase?
tls=true&replicaSet=rs0&readpreference={3}";
            string username = "sample-user";
            string password = "password";
            string readPreference = "secondaryPreferred";
            string clusterEndpoint="sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017";
            string connectionString = String.Format(template, username, password,
 clusterEndpoint, readPreference);

            string pathToCAFile = "<PATH/global-bundle.p7b_file>";

            // ADD CA certificate to local trust store
            // DO this once - Maybe when your service starts
            X509Store localTrustStore = new X509Store(StoreName.Root);
            X509Certificate2Collection certificateCollection = new
 X509Certificate2Collection();
            certificateCollection.Import(pathToCAFile);
            try
            {
                localTrustStore.Open(OpenFlags.ReadWrite);
                localTrustStore.AddRange(certificateCollection);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Root certificate import failed: " + ex.Message);
                throw;
            }
            finally
            {
                localTrustStore.Close();
            }

            var settings = MongoClientSettings.FromUrl(new
 MongoUrl(connectionString));
            var client = new MongoClient(settings);

            var database = client.GetDatabase("sampledatabase");
            var collection =
 database.GetCollection<BsonDocument>("samplecollection");
            var docToInsert = new BsonDocument { { "pi", 3.14159 } };
```

```
            collection.InsertOne(docToInsert);
        }
    }
}
```

MongoDB Shell

下列程式碼示範如何在啟用 TLS 時，使用最新版本、mongosh 或先前的 mongo shell 版本來連線至和查詢 Amazon DocumentDB。

使用 mongosh 連線至 Amazon DocumentDB

> **Important**
>
> 6.13.1 版之前的 Node.js 驅動程式有已知的限制，Amazon DocumentDB 的 IAM 身分驗證目前不支援此限制。使用 Node.js 驅動程式的 Node.js 驅動程式和工具（例如 mongosh) 必須升級為使用 Node.js 驅動程式 6.13.1 版或更新版本。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
mongosh --tls --host cluster-end-point:27017 --tlsCAFile  global-bundle.pem --
username sample-user --password password --retryWrites false
```

使用先前的 mongo shell 版本連線至 Amazon DocumentDB

如果您使用 IAM，則必須使用舊版的 mongo shell。輸入下列其中一個命令選項：

```
mongo --ssl --host cluster-end-point:27017 --sslCAFile global-bundle.pem --
username sample-user --password password
```

如果您使用的版本等於或大於 4.2，請使用下列程式碼進行連線。Amazon DocumentDB 不支援可重試的寫入。如果您使用的是舊版 mongo shell（非 mongosh)，請勿在任何程式碼字串中包含 `retryWrites=false`命令。預設會停用可重試寫入。包含 `retryWrites=false` 可能會導致正常讀取命令失敗。

```
mongo --tls --host cluster-end-point:27017 --tlsCAFile global-bundle.pem --
username sample-user --password password
```

測試連線

1. 插入單一文件。

```
db.myTestCollection.insertOne({'hello':'Amazon DocumentDB'})
```

2. 尋找之前插入的文件。

```
db.myTestCollection.find({'hello':'Amazon DocumentDB'})
```

R

下列程式碼示範如何在啟用 TLS 時使用 mongolite (https://jeroen.github.io/mongolite/) 連線至具有 R 的 Amazon DocumentDB。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
#Include the mongolite library.
library(mongolite)

mongourl <- paste("mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/test2?ssl=true&",
          "readPreference=secondaryPreferred&replicaSet=rs0", sep="")

#Create a MongoDB client, open a connection to Amazon DocumentDB as a replica
#   set and specify the read preference as secondary preferred
client <-  mongo(url = mongourl, options = ssl_options(weak_cert_validation = F, ca
 ="<PATH/global-bundle.pem>"))

#Insert a single document
str <- c('{"hello" : "Amazon DocumentDB"}')
client$insert(str)

#Find the document that was previously written
client$find()
```

Ruby

下列程式碼示範如何在啟用 TLS 時，使用 Ruby 連線至 Amazon DocumentDB。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
require 'mongo'
require 'neatjson'
require 'json'
client_host = 'mongodb://sample-cluster.node.us-east-1.docdb.amazonaws.com:27017'
client_options = {
   database: 'test',
   replica_set: 'rs0',
   read: {:secondary_preferred => 1},
   user: 'sample-user',
   password: 'password',
   ssl: true,
   ssl_verify: true,
   ssl_ca_cert: 'PATH/global-bundle.pem',
   retry_writes: false
}


begin
   ##Create a MongoDB client, open a connection to Amazon DocumentDB as a
   ##   replica set and specify the read preference as secondary preferred
   client = Mongo::Client.new(client_host, client_options)

   ##Insert a single document
   x = client[:test].insert_one({"hello":"Amazon DocumentDB"})

   ##Find the document that was previously written
   result = client[:test].find()

   #Print the document
   result.each do |document|
      puts JSON.neat_generate(document)
   end
end

#Close the connection
client.close
```

使用已停用的 TLS 連線

若要以程式設計方式檢視連線至停用 TLS 的 Amazon DocumentDB 叢集的程式碼範例，請選擇您要使用之語言的標籤。

Python

下列程式碼示範如何在停用 TLS 時使用 Python 連線至 Amazon DocumentDB。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
## Create a MongoDB client, open a connection to Amazon DocumentDB as a replica set
 and specify the read preference as secondary preferred

import pymongo
import sys

client = pymongo.MongoClient('mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/?
replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

##Specify the database to be used
db = client.sample_database

##Specify the collection to be used
col = db.sample_collection

##Insert a single document
col.insert_one({'hello':'Amazon DocumentDB'})

##Find the document that was previously written
x = col.find_one({'hello':'Amazon DocumentDB'})

##Print the result to the screen
print(x)

##Close the connection
client.close()
```

Node.js

下列程式碼示範如何在停用 TLS 時使用 Node.js 連線至 Amazon DocumentDB。

> **Important**
>
> 6.13.1 版之前的 Node.js 驅動程式有已知的限制，Amazon DocumentDB 的 IAM 身分驗證目前不支援此限制。使用 Node.js 驅動程式的 Node.js 驅動程式和工具（例如 mongosh）必須升級為使用 Node.js 驅動程式 6.13.1 版或更新版本。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
var MongoClient = require('mongodb').MongoClient;

//Create a MongoDB client, open a connection to Amazon DocumentDB as a replica set,
//  and specify the read preference as secondary preferred
var client = MongoClient.connect(
'mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/sample-database?
replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false',
{
  useNewUrlParser: true
},

function(err, client) {
    if(err)
        throw err;
    //Specify the database to be used
    db = client.db('sample-database');

    //Specify the collection to be used
    col = db.collection('sample-collection');

    //Insert a single document
    col.insertOne({'hello':'Amazon DocumentDB'}, function(err, result){
      //Find the document that was previously written
      col.findOne({'hello':'Amazon DocumentDB'}, function(err, result){
        //Print the result to the screen
        console.log(result);

        //Close the connection
        client.close()
      });
   });
});
```

PHP

下列程式碼示範如何在停用 TLS 時使用 PHP 連線至 Amazon DocumentDB。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
<?php
//Include Composer's autoloader
require 'vendor/autoload.php';

//Create a MongoDB client and open connection to Amazon DocumentDB
$client = new MongoDB\Client("mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/?retryWrites=false");

//Specify the database and collection to be used
$col = $client->sampledatabase->samplecollection;

//Insert a single document
$result = $col->insertOne( [ 'hello' => 'Amazon DocumentDB'] );

//Find the document that was previously written
$result = $col->findOne(array('hello' => 'Amazon DocumentDB'));

//Print the result to the screen
print_r($result);
?>
```

Go

下列程式碼示範如何在停用 TLS 時使用 Go 連線至 Amazon DocumentDB。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
package main

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

 "go.mongodb.org/mongo-driver/bson"
 "go.mongodb.org/mongo-driver/mongo"
 "go.mongodb.org/mongo-driver/mongo/options"
```

```
)

const (
 // Timeout operations after N seconds
 connectTimeout  = 5
 queryTimeout    = 30
 username        = "sample-user"
 password        = "password"
 clusterEndpoint = "sample-cluster.node.us-east-1.docdb.amazonaws.com:27017"

 // Which instances to read from
 readPreference           = "secondaryPreferred"
 connectionStringTemplate = "mongodb://%s:%s@%s/sample-database?
replicaSet=rs0&readpreference=%s"
)

func main() {

 connectionURI := fmt.Sprintf(connectionStringTemplate, username, password,
 clusterEndpoint, readPreference)

 client, err := mongo.NewClient(options.Client().ApplyURI(connectionURI))
 if err != nil {
  log.Fatalf("Failed to create client: %v", err)
 }

 ctx, cancel := context.WithTimeout(context.Background(),
 connectTimeout*time.Second)
 defer cancel()

 err = client.Connect(ctx)
 if err != nil {
  log.Fatalf("Failed to connect to cluster: %v", err)
 }

 // Force a connection to verify our connection string
 err = client.Ping(ctx, nil)
 if err != nil {
  log.Fatalf("Failed to ping cluster: %v", err)
 }

 fmt.Println("Connected to DocumentDB!")

 collection := client.Database("sample-database").Collection("sample-collection")
```

```
 ctx, cancel = context.WithTimeout(context.Background(), queryTimeout*time.Second)
 defer cancel()

 res, err := collection.InsertOne(ctx, bson.M{"name": "pi", "value": 3.14159})
 if err != nil {
  log.Fatalf("Failed to insert document: %v", err)
 }

 id := res.InsertedID
 log.Printf("Inserted document ID: %s", id)

 ctx, cancel = context.WithTimeout(context.Background(), queryTimeout*time.Second)
 defer cancel()

 cur, err := collection.Find(ctx, bson.D{})

 if err != nil {
  log.Fatalf("Failed to run find query: %v", err)
 }
 defer cur.Close(ctx)

 for cur.Next(ctx) {
  var result bson.M
  err := cur.Decode(&result)
  log.Printf("Returned: %v", result)

  if err != nil {
   log.Fatal(err)
  }
 }

 if err := cur.Err(); err != nil {
  log.Fatal(err)
 }

}
```

Java

下列程式碼示範如何在停用 TLS 時使用 Java 連線至 Amazon DocumentDB。

在下列範例中，將每個#########取代為您叢集的資訊。

```
package com.example.documentdb;

import com.mongodb.MongoClient;
import com.mongodb.MongoClientURI;
import com.mongodb.ServerAddress;
import com.mongodb.MongoException;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.MongoCollection;
import org.bson.Document;


public final class Main {
    private Main() {
    }
    public static void main(String[] args) {

        String template = "mongodb://%s:%s@%s/sample-database?
replicaSet=rs0&readpreference=%s";
        String username = "sample-user";
        String password = "password";
        String clusterEndpoint = "sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017";
        String readPreference = "secondaryPreferred";
        String connectionString = String.format(template, username, password,
 clusterEndpoint, readPreference);

        MongoClientURI clientURI = new MongoClientURI(connectionString);
        MongoClient mongoClient = new MongoClient(clientURI);

        MongoDatabase testDB = mongoClient.getDatabase("sample-database");
        MongoCollection<Document> numbersCollection = testDB.getCollection("sample-
collection");

        Document doc = new Document("name", "pi").append("value", 3.14159);
        numbersCollection.insertOne(doc);

        MongoCursor<Document> cursor = numbersCollection.find().iterator();
        try {
            while (cursor.hasNext()) {
                System.out.println(cursor.next().toJson());
            }
        } finally {
```

```
            cursor.close();
        }

    }
}
```

C# / .NET

下列程式碼示範如何在停用 TLS 時使用 C# / .NET 連線至 Amazon DocumentDB。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
using System;
using System.Text;
using System.Linq;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;
using System.Net.Security;
using MongoDB.Driver;
using MongoDB.Bson;

namespace CSharpSample
{
   class Program
    {
       static void Main(string[] args)
        {
           string template = "mongodb://{0}:{1}@{2}/sampledatabase?
replicaSet=rs0&readpreference={3}";
           string username = "sample-user";
           string password = "password";
           string clusterEndpoint = "sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017";
           string readPreference = "secondaryPreferred";
           string connectionString = String.Format(template, username, password,
 clusterEndpoint, readPreference);

           var settings = MongoClientSettings.FromUrl(new
 MongoUrl(connectionString));
           var client = new MongoClient(settings);

           var database = client.GetDatabase("sampledatabase");
```

```
            var collection =
 database.GetCollection<BsonDocument>("samplecollection");
            var docToInsert = new BsonDocument { { "pi", 3.14159 } };
             collection.InsertOne(docToInsert);
        }
    }
}
```

MongoDB Shell

下列程式碼示範如何在停用 TLS 時，使用最新版本、mongosh 或先前的 mongo shell 版本來連線至和查詢 Amazon DocumentDB。

使用 mongosh 連線至 Amazon DocumentDB

> **Important**
>
> 6.13.1 版之前的 Node.js 驅動程式有已知的限制，Amazon DocumentDB 的 IAM 身分驗證目前不支援此限制。使用 Node.js 驅動程式的 Node.js 驅動程式和工具（例如 mongosh）必須升級為使用 Node.js 驅動程式 6.13.1 版或更新版本。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
mongosh --host cluster-end-point:27017 --username sample-user --password password --
retryWrites false
```

使用先前的 mongo shell 版本連線至 Amazon DocumentDB

如果您使用 IAM，則必須使用舊版的 mongo shell。輸入下列其中一個命令選項：

```
mongo --host cluster-end-point:27017 --username sample-user --password password
```

如果您使用的版本等於或大於 4.2，請使用下列程式碼進行連線。Amazon DocumentDB 不支援可重試的寫入。如果您使用的是舊版 mongo shell（非 mongosh），請勿在任何程式碼字串中包含 `retryWrites=false`命令。預設會停用可重試寫入。包含 `retryWrites=false` 可能會導致正常讀取命令失敗。

```
mongo --host cluster-end-point:27017 --username sample-user --password password
```

測試連線

1. 插入單一文件。

```
db.myTestCollection.insertOne({'hello':'Amazon DocumentDB'})
```

2. 尋找之前插入的文件。

```
db.myTestCollection.find({'hello':'Amazon DocumentDB'})
```

R

下列程式碼示範如何在停用 TLS 時使用 mongolite (https://jeroen.github.io/mongolite/) 以 R 連線至 Amazon DocumentDB。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
#Include the mongolite library.
library(mongolite)

#Create a MongoDB client, open a connection to Amazon DocumentDB as a replica
#   set and specify the read preference as secondary preferred
client <- mongo(url = "mongodb://sample-user:password@sample-
cluster.node.us-east-1.docdb.amazonaws.com:27017/sample-database?
readPreference=secondaryPreferred&replicaSet=rs0")

##Insert a single document
str <- c('{"hello" : "Amazon DocumentDB"}')
client$insert(str)

##Find the document that was previously written
client$find()
```

Ruby

下列程式碼示範如何在停用 TLS 時，使用 Ruby 連線至 Amazon DocumentDB。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
require 'mongo'
require 'neatjson'
require 'json'
client_host = 'mongodb://sample-cluster.node.us-east-1.docdb.amazonaws.com:27017'
```

```
client_options = {
   database: 'test',
   replica_set: 'rs0',
   read: {:secondary_preferred => 1},
   user: 'sample-user',
   password: 'password',
   retry_writes: false
}

begin
   ##Create a MongoDB client, open a connection to Amazon DocumentDB as a
   ##   replica set and specify the read preference as secondary preferred
   client = Mongo::Client.new(client_host, client_options)

   ##Insert a single document
   x = client[:test].insert_one({"hello":"Amazon DocumentDB"})

   ##Find the document that was previously written
   result = client[:test].find()

   #Print the document
   result.each do |document|
      puts JSON.neat_generate(document)
   end
end

#Close the connection
client.close
```

Amazon DocumentDB 中的網路壓縮

透過線路通訊協定壓縮，您可以減少應用程式透過網路傳送的資料。Amazon DocumentDB 8.0.1 及更新版本支援壓縮。您可以改善傳輸大型文件或大型結果集的工作負載輸送量。啟用壓縮後，驅動程式和伺服器會交換壓縮的訊息。

啟用網路壓縮時，您會交易 CPU 資源以降低網路頻寬。在工作負載與網路繫結時啟用壓縮，並傳輸大型承載。對於小型請求和回應，您可能不會看到淨利益，因為壓縮開銷可能會超過節省成本。

支援的壓縮器

Amazon DocumentDB 支援`snappy`壓縮器。Amazon DocumentDB 不支援 MongoDB 驅動程式提供的其他壓縮器，例如 `zlib`和 `zstd`。

如果驅動程式請求壓縮器清單，Amazon DocumentDB 只會交涉支援的壓縮器清單。例如，如果驅動程式提供 snappy和 zstd，Amazon DocumentDB 會選取snappy並忽略 zstd。如果驅動程式僅提供不支援的壓縮器，Amazon DocumentDB 不會壓縮連線。在此情況下，驅動程式和伺服器在沒有壓縮的情況下交換訊息。

啟用壓縮

若要啟用壓縮，請將 compressors選項新增至您的連線字串，並將其設定為 snappy。在下列連線字串中，將 *username*、 *password*和 取代*docdb-cluster-endpoint*為您自己的值：

```
mongodb://username:password@docdb-cluster-endpoint:27017/?
tls=true&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false&compressors=snappy
```

當您使用 MongoDB 驅動程式時，請透過驅動程式的連線選項設定相同的選項。大多數驅動程式需要額外的用戶端相依性，才能執行快照壓縮：

- Node.js 驅動程式需要 snappy套件。
- Python 驅動程式 (PyMongo) 需要 python-snappy套件。在 Linux 上， python-snappy也需要系統快照程式庫 （ 例如，snappy-devel透過 dnf或 yum)。

如需確切的選項名稱和任何必要的套件，請參閱您的驅動程式文件。

> **Note**
>
> 如果缺少用戶端壓縮相依性，驅動程式會成功連線。不過，它會在沒有壓縮的情況下交換訊息，而不會傳回錯誤。若要確認壓縮作用中，請驗證交涉的壓縮器，如中所述驗證壓縮。

下列 Python 程式碼範例會連線至已啟用快照壓縮的叢集，並使用 try和 except區塊來處理連線錯誤：

```
import pymongo

try:
    client = pymongo.MongoClient(
        'mongodb://username:password@docdb-cluster-endpoint:27017/'
        '?tls=true&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false'
        '&compressors=snappy',
        tlsCAFile='global-bundle.pem')
```

```
    db = client['sample-database']
    col = db['sample-collection']
    col.insert_one({'item': 'DocumentDB'})
    print(col.find_one())
except Exception as error:
    print(f'Connection failed: {error}')
```

驗證壓縮

若要確認驅動程式和 Amazon DocumentDB 交涉壓縮，請檢查 `hello` MongoDB 線路通訊協定命令傳回`compression`的欄位。當驅動程式和 Amazon DocumentDB 交涉壓縮時，此欄位會列出商定的壓縮器。

從 Amazon VPC 外部連線至 Amazon DocumentDB 叢集

Amazon DocumentDB （具有 MongoDB 相容性） 叢集會部署在 Amazon Virtual Private Cloud (Amazon VPC) 中。Amazon EC2 執行個體或其他部署在相同 Amazon VPC 中的 AWS 服務可以直接存取它們。此外，Amazon DocumentDB 可以透過 VPCs 對等互連，由相同或其他區域中不同 VPC 中的 EC2 執行個體 AWS 區域 或其他 AWS 服務存取。

不過，假設您的使用案例要求您 （或您的應用程式） 從叢集的 VPC 外部存取 Amazon DocumentDB 資源。在這種情況下，您可以使用 SSH 通道 （也稱為連接埠轉送) 來存取 Amazon DocumentDB 資源。

不過，深入討論 SSH 通道超出本主題的範圍。如需 SSH 通道的詳細資訊，請參閱以下內容：

- SSH 通道
- SSH 連接埠轉送範例，具體來說是本機轉送一節

若要建立 SSH 通道，您需要在與 Amazon DocumentDB 叢集相同的 Amazon VPC 中執行的 Amazon EC2 執行個體。 Amazon DocumentDB 您可以使用相同 VPC 中現有的 EC2 執行個體做為您的叢集，或建立一個叢集。如需詳細資訊，請參閱適用您作業系統的主題：

- Amazon EC2 Linux 執行個體入門
- Amazon EC2 Windows 執行個體入門

通常您可能會使用下列命令連接到 EC2 執行個體。

```
ssh -i "ec2Access.pem" ubuntu@ec2-34-229-221-164.compute-1.amazonaws.com
```

若是如此，您可以在本機電腦上執行下列命令`sample-cluster.node.us-east-1.docdb.amazonaws.com`，以設定 Amazon DocumentDB 叢集的 SSH 通道。`-L` 旗標用於轉送本機連接埠。使用 SSH 通道時，請使用叢集端點連線至您的叢集，並且不要嘗試在複本集模式下連線（即在您的連線字串`replicaSet=rs0`中指定），因為這會導致錯誤。

```
ssh -i "ec2Access.pem" -L 27017:sample-cluster.node.us-east-1.docdb.amazonaws.com:27017
 ubuntu@ec2-34-229-221-164.compute-1.amazonaws.com -N
```

建立 SSH 通道之後，您向 發出的任何命令`localhost:27017`都會轉送至在 Amazon VPC 中執行的 Amazon DocumentDB 叢集。 `sample-cluster`如果您的 Amazon DocumentDB 叢集上已啟用 Transport Layer Security (TLS)，您需要從 下載 Amazon DocumentDB https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem 的公有金鑰。下列操作會下載此檔案：

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

> **Note**
>
> 新的 Amazon DocumentDB 叢集預設會啟用 TLS。不過，您可以停用它。如需詳細資訊，請參閱管理 Amazon DocumentDB 叢集 TLS 設定。

若要從 Amazon VPC 外部連線至 Amazon DocumentDB 叢集，請使用下列命令。

```
mongo --sslAllowInvalidHostnames --ssl --sslCAFile global-bundle.pem --username
 <yourUsername> --password <yourPassword>
```

以複本集的形式連線至 Amazon DocumentDB

當您針對 Amazon DocumentDB 進行開發時，請以複本集的形式連線至叢集，並使用驅動程式的內建讀取偏好設定功能，將讀取分散至複本執行個體。本節會更深入探討這意味著什麼，並說明如何使用適用於 Python 的 SDK 做為複本集來連線至 Amazon DocumentDB 叢集。

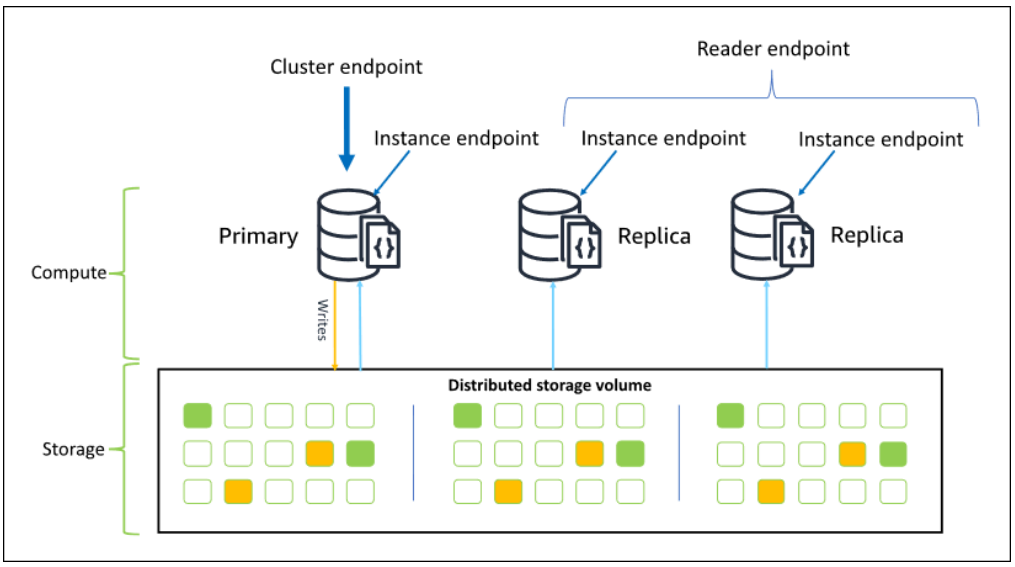
Amazon DocumentDB 有三個端點，可用來連線至叢集：

- 叢集端點
- 讀取器端點
- 執行個體端點

連線至 Amazon DocumentDB 時，請使用叢集端點。這是將指向您叢集中主要執行個體的 CNAME，如下圖所示。

使用 SSH 通道時，請使用叢集端點連線至您的叢集。請勿在複本集模式下連線（在連線字串`replicaSet=rs0`中指定），因為這會導致錯誤。

> **Note**
>
> 如需 Amazon DocumentDB 端點的詳細資訊，請參閱 Amazon DocumentDB 端點。



使用叢集端點時，您可以連接至處在複本集模式下的叢集。您就可以使用內建的讀取偏好設定驅動程式功能。在下列範例中，指定 `/?replicaSet=rs0` 時，系統會通知軟體開發套件您想要以複本集形式進行連接。如果您省略 `/?replicaSet=rs0'`，用戶端會將所有請求路由到叢集端點，也就是您的主要執行個體。

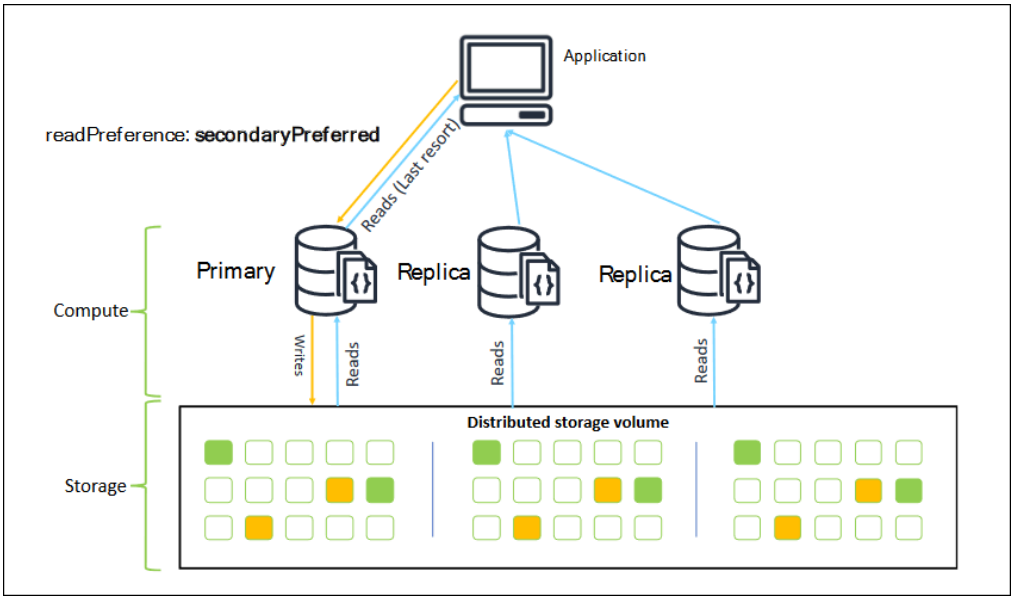
```
## Create a MongoDB client, open a connection to Amazon DocumentDB as a
   ## replica set and specify the read preference as secondary preferred
   client = pymongo.MongoClient('mongodb://<user-name>:<password>@mycluster.node.us-
east-1.docdb.amazonaws.com:27017/?replicaSet=rs0')
```

以複本集形式連接的優點，就是其可讓軟體開發套件自動探索叢集拓撲圖，包括當叢集新增或移除執行個體時。然後，您就可以將讀取請求路由至複本執行個體，提高叢集的使用效率。

當您以複本集形式連接時，您可以指定 `readPreference` 進行連線。如果您指定的是 `secondaryPreferred` 的讀取偏好設定，則用戶端會將讀取查詢路由到您的複本，並將查詢寫入到

您的主要執行個體 (如下圖所示)。這是較佳的叢集資源使用方式。如需詳細資訊，請參閱讀取偏好設定選項。

```
## Create a MongoDB client, open a connection to Amazon DocumentDB as a
   ##   replica set and specify the read preference as secondary preferred
client = pymongo.MongoClient('mongodb://<user-name>:<password>@mycluster.node.us-
east-1.docdb.amazonaws.com:27017/?replicaSet=rs0&readPreference=secondaryPreferred')
```



Amazon DocumentDB 複本的讀取最終會一致。它們會依據主要執行個體上的相同寫入順序來傳回資料，而且複寫延遲通常少於 50 毫秒。您可以使用 Amazon CloudWatch 指標DBInstanceReplicaLag和 監控叢集的複本延遲DBClusterReplicaLagMaximum。如需詳細資訊，請參閱使用 CloudWatch 監控 Amazon DocumentDB。

與傳統的單體資料庫架構不同，Amazon DocumentDB 會區隔儲存和運算。假設您使用這種現代架構，那麼我們鼓勵您讀取複本執行個體上的擴展。複本執行個體上的讀取不會封鎖從主要執行個體複寫的寫入。您在叢集中最多可以新增 15 個讀取複本執行個體，並將讀取量擴展到每秒數百萬次讀取。

以複本集形式連接，並將讀取分配至複本的主要優點，在於其會在叢集中增加可供您應用程式使用的整體資源。最佳實務是將 連線為複本集。這在下列案例中最有用：

- 您正使用主要執行個體上將近 100% 的 CPU。
- 緩衝區快取命中率接近零。
- 您的個別執行個體的連線或游標已達上限。

您可以選擇向上擴展叢集執行個體大小，而且在某些情況下，這可能是擴展叢集的最佳方式。但是，您也應該考慮如何提高叢集中既有複本的使用量。這可讓您直接增加擴展，而不用增加使用更大執行個體類型的成本。我們也建議您使用 CloudWatch 警示來監控和提醒這些限制 （即 `DatabaseConnections`、 `CPUUtilization`和 `BufferCacheHitRatio`)，讓您知道資源何時大量使用。

如需詳細資訊，請參閱下列主題：

- Amazon DocumentDB 的最佳實務
- Amazon DocumentDB 配額

使用叢集連線

請考慮使用叢集中所有連線的案例。例如，`r5.2xlarge` 執行個體的連線數目限為 4,500 個 (而開放游標限制為 450 個)。如果您建立三執行個體 Amazon DocumentDB 叢集，並僅使用叢集端點連線至主要執行個體，則開放連線和游標的叢集限制分別為 4，500 和 450。如果您建立的應用程式會使用多個將從容器啟動的 Worker，那麼您可能會達到這些限制。這些容器會一次開啟多個連線，並使叢集達到飽和。

反之，您可以連接 Amazon DocumentDB 叢集做為複本集，並將您的讀取分佈到複本執行個體。接著，您可以將叢集中的可用連線和游標數量成功增為三倍，分別為 13,500 和 1,350。將更多執行個體新增至叢集時，只會增加讀取工作負載的連線數和游標數。如果您需要在叢集中增加寫入連線數目，建議您增加執行個體大小。

> **Note**
>
> `large`、`xlarge` 和 `2xlarge` 執行個體的連線數目會隨著執行個體大小增加，最多達 4,500 個。`4xlarge` 個或超過 4,500 個執行個體時，每個執行個體的連線數目上限。如需依執行個體類型限制的詳細資訊，請參閱 執行個體配額。

一般而言，我們不建議您使用 `secondary` 的讀取偏好設定來連接至叢集 。原因在於如果您的叢集中沒有複本執行個體，該讀取將會失敗。例如，假設您有一個具有一個主要複本和一個複本的兩個執行個體 Amazon DocumentDB 叢集。如果複本發生問題，則設為 `secondary` 之連線集區的讀取請求將會失敗。使用 `secondaryPreferred` 的優點，在於如果用戶端找不到適合的複本執行個體來進行連接，則其會回復到主要執行個體來進行讀取。

多個連線集區

在某些情況下，應用程式中的讀取需要具有read-after-write一致性，只能從 Amazon DocumentDB 中的主要執行個體提供。在這些情況下，您可以建立兩個用戶端連線集區：一個用於寫入，另一個用於需要先寫後讀一致性的讀取。若要這樣做，您的程式碼應該與下列類似。

```
## Create a MongoDB client,
##    open a connection to Amazon DocumentDB as a replica set and specify the
 readPreference as primary
clientPrimary = pymongo.MongoClient('mongodb://<user-
name>:<password>@mycluster.node.us-east-1.docdb.amazonaws.com:27017/?
replicaSet=rs0&readPreference=primary')

## Create a MongoDB client,
##    open a connection to Amazon DocumentDB as a replica set and specify the
 readPreference as secondaryPreferred
secondaryPreferred = pymongo.MongoClient('mongodb://<user-
name>:<password>@mycluster.node.us-east-1.docdb.amazonaws.com:27017/?
replicaSet=rs0&readPreference=secondaryPreferred')
```

另一種選擇是建立單一連線集區，並覆寫指定集合的讀取偏好設定。

```
##Specify the collection and set the read preference level for that collection
col = db.review.with_options(read_preference=ReadPreference.SECONDARY_PREFERRED)
```

摘要

若要更好地使用叢集中的資源，請使用複本集模式進行連線。如果您的應用程式適合這種做法，您可以將讀取分配到該複本執行個體，讀取擴展您的應用程式。

從 Studio 3T 連線至 Amazon DocumentDB 叢集

Studio 3T 是使用 MongoDB 的開發人員和資料工程師的熱門 GUI 和 IDE。它提供資料的多種強大功能樹狀結構、資料表和 JSON 檢視、CSV、JSON、SQL 和 BSON/mongodump 輕鬆匯入/匯出、靈活的查詢選項、視覺化drag-and-drop UI、具有自動完成功能的內建 mongo shell、彙總管道編輯器，以及 SQL 查詢支援。

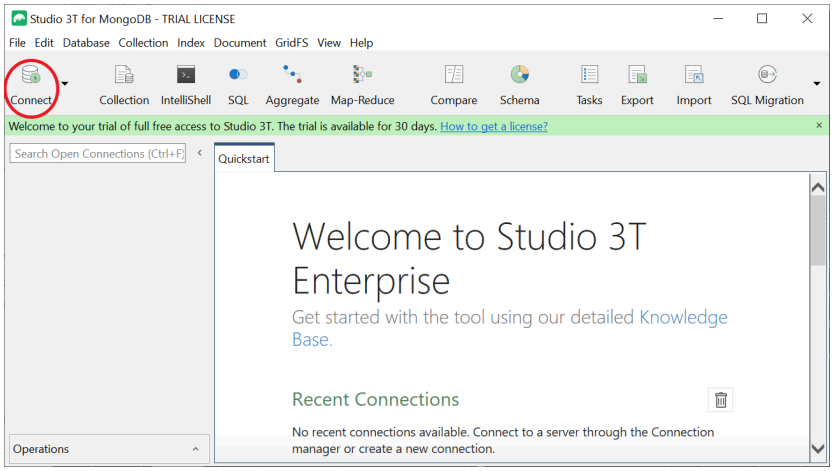
先決條件

- 如果您還沒有使用 Amazon EC2 做為堡壘/跳躍主機的 Amazon DocumentDB 叢集，請參閱 使用 Amazon EC2 連線。 Amazon EC2

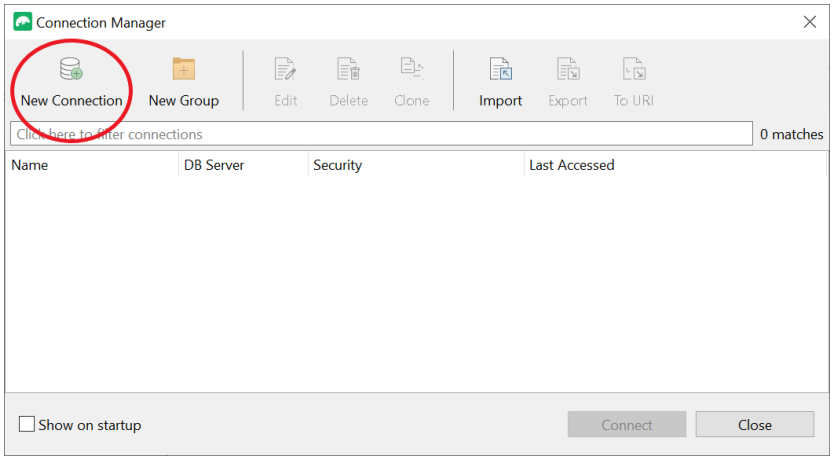
- 如果您沒有 Studio 3T，請下載並安裝它。

使用 Studio 3T 連線

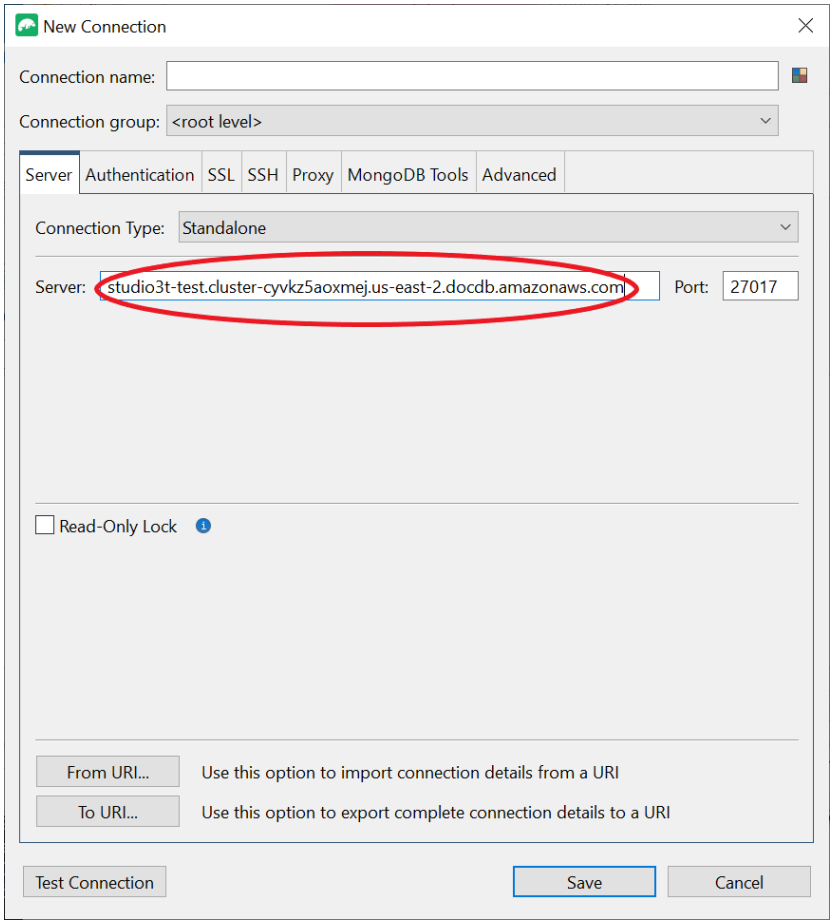
1. 選擇工具列左上角的連線。



2. 選擇工具列左上角的新連線。

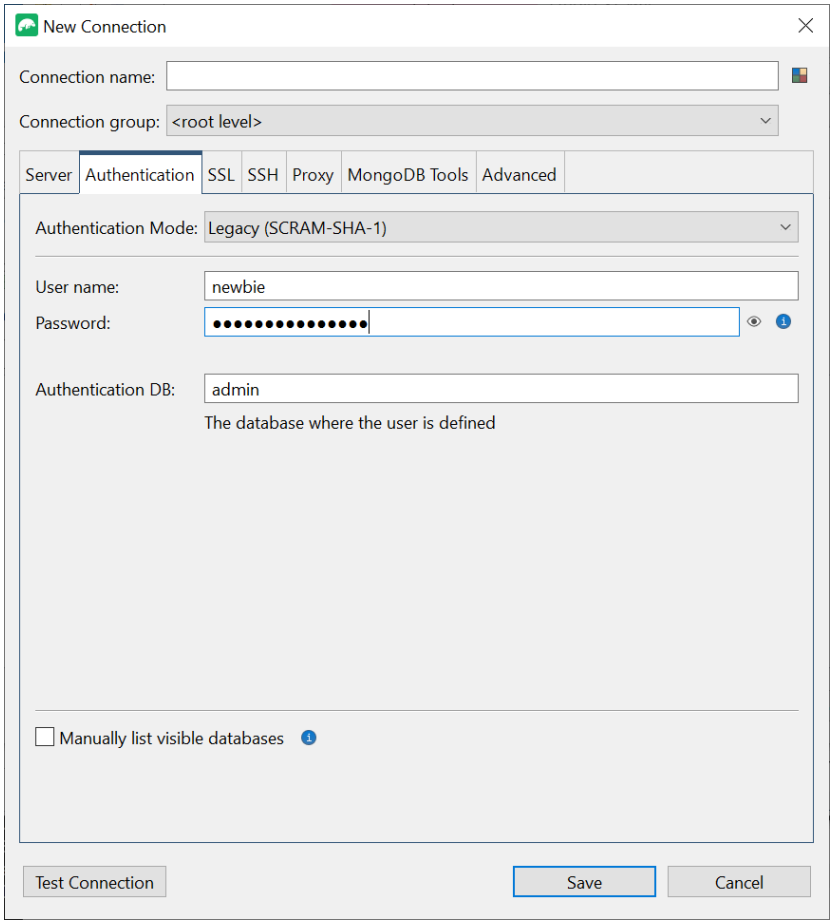


3. 在伺服器索引標籤的伺服器欄位中，輸入叢集端點資訊。

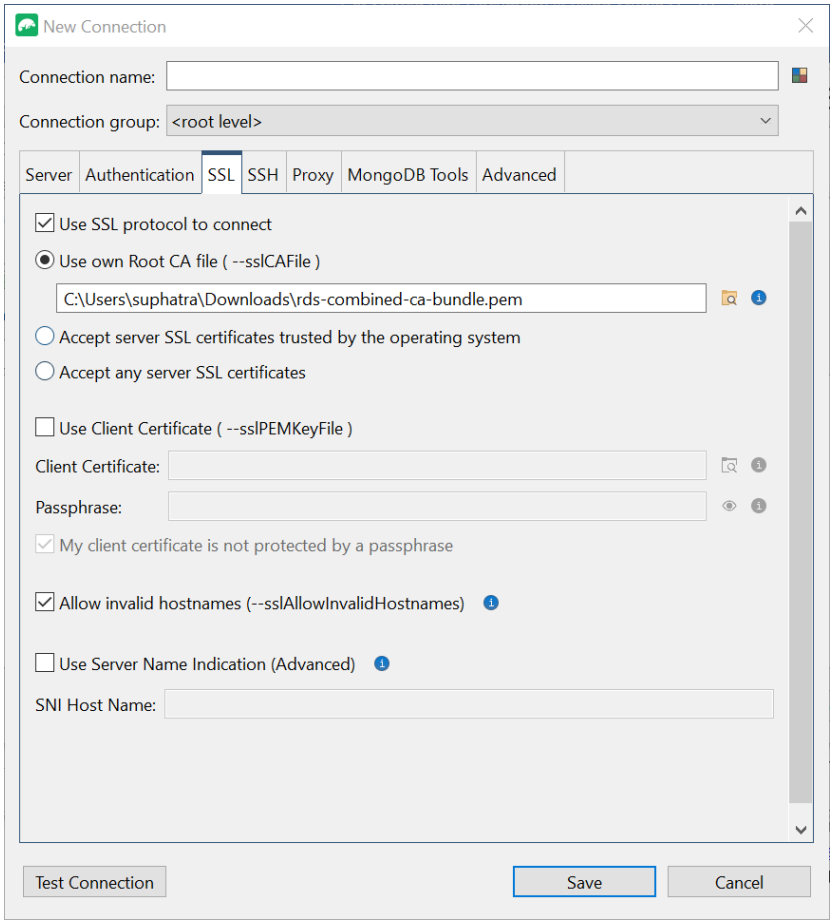


> **Note**
>
> 若要尋找叢集端點，請參閱 [尋找執行個體的端點](#)。

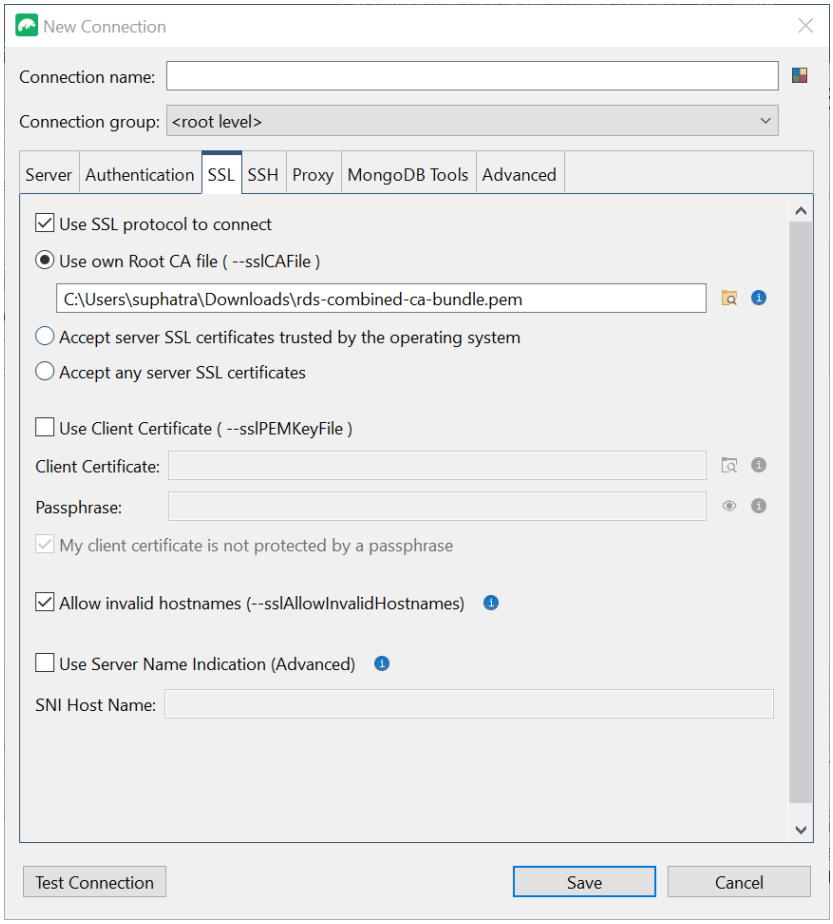
4. 選擇身分驗證索引標籤，然後在身分驗證模式的下拉式選單中選取舊版。



5. 在使用者名稱和密碼欄位中輸入您的使用者名稱和密碼。
6. 選擇 SSL 索引標籤，然後勾選使用 SSL 通訊協定進行連線。



7. 選擇使用自己的根 CA 檔案。然後新增 Amazon DocumentDB 憑證 （如果 DocumentDB 叢集上已停用 SSL，您可以略過此步驟）。勾選方塊以允許無效的主機名稱。



> **Note**
>
> 沒有憑證？ 您可以使用下列命令下載：
>
> ```
> wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
> ```

8. 如果您是從 Amazon VPC 外部的用戶端機器連線，則需要建立 SSH 通道。您將在 SSH 索引標籤中執行此操作。

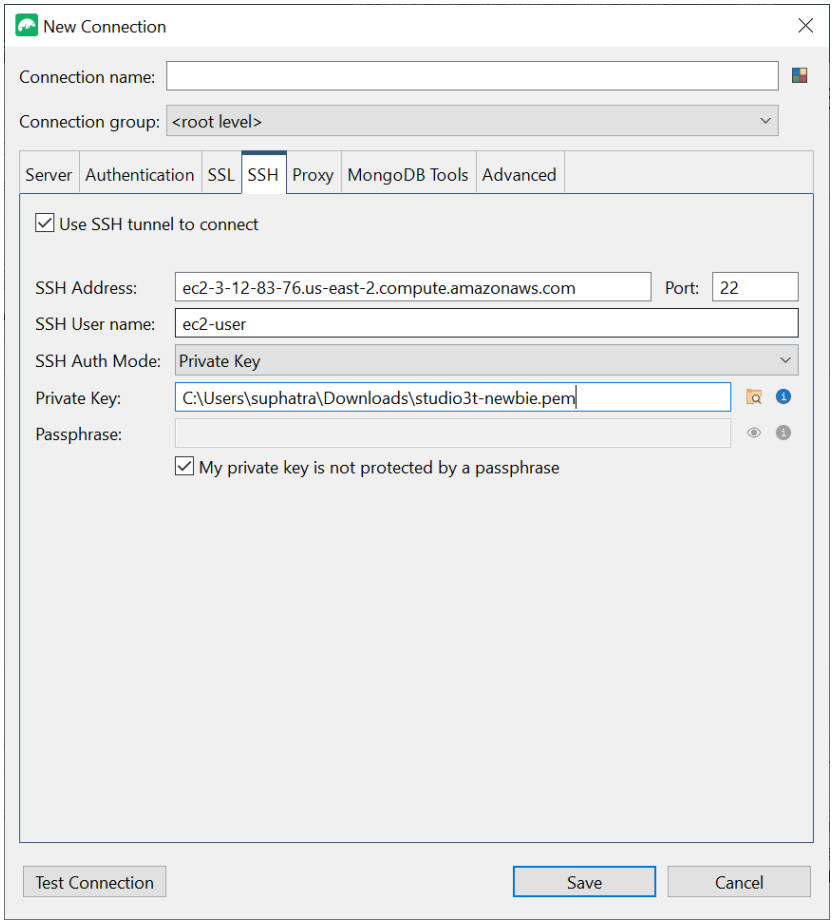
   a. 勾選使用 SSH 通道的方塊，然後在 SSH 地址欄位中輸入 SSH 地址。這是您的執行個體公有 DNS (IPV4)。您可以從 Amazon EC2 管理主控台取得此 URL。

   b. 輸入您的使用者名稱。這是 Amazon EC2 執行個體的使用者名稱

   c. 針對 SSH 驗證模式，選取私有金鑰。在私密金鑰欄位中，選擇檔案搜尋工具圖示以尋找並選擇 Amazon EC2 執行個體的私密金鑰。這是您在 Amazon EC2 主控台中建立執行個體時儲存的 .pem 檔案 （ 金鑰對 ）。

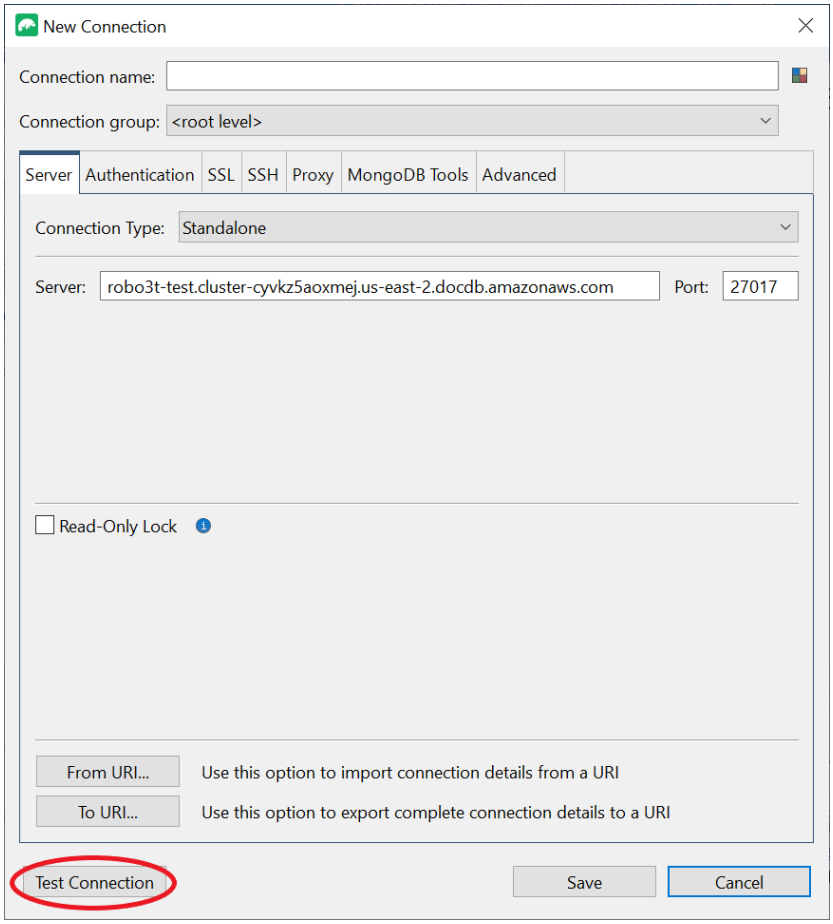
   d. 如果您在 Linux/macOS 用戶端機器上，您可能需要使用下列命令變更私有金鑰的許可：

```
chmod 400 /fullPathToYourPemFile/<yourKey>.pem
```

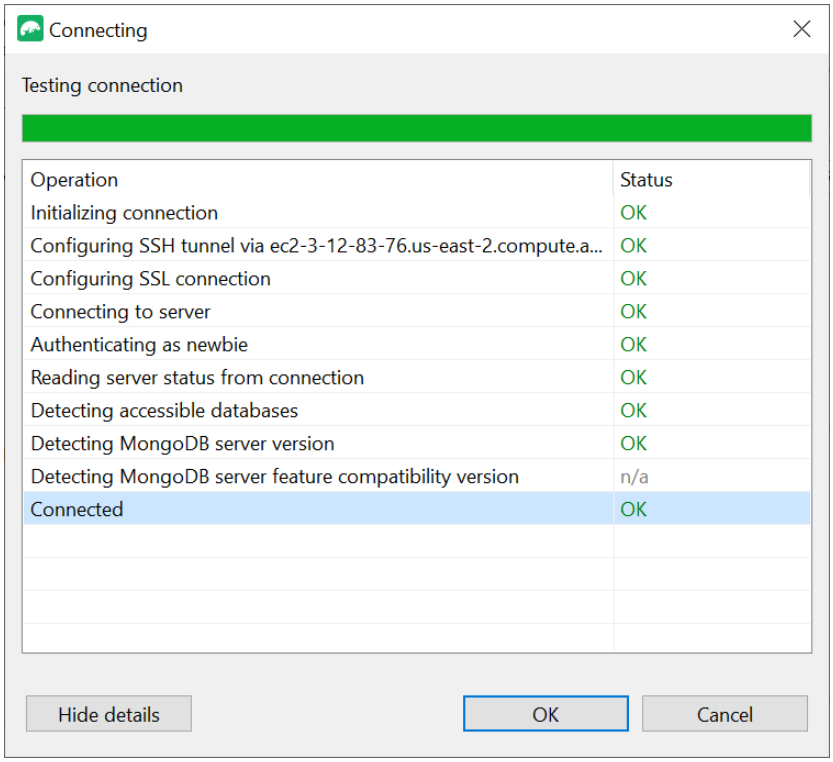


> **Note**
>
> 此 Amazon EC2 執行個體應與 DocumentDB 叢集位於相同的 Amazon VPC 和安全群組中。您可以從 Amazon EC2 管理主控台取得 SSH 地址、使用者名稱和私有金鑰。

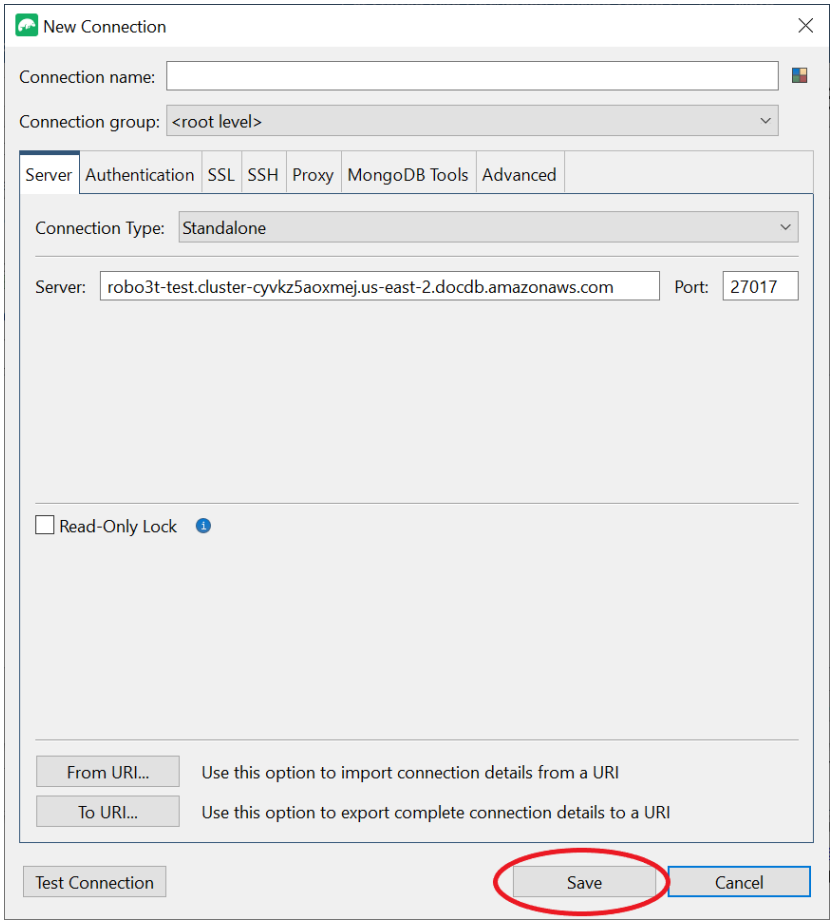
9. 現在，選擇測試連線按鈕來測試您的組態。



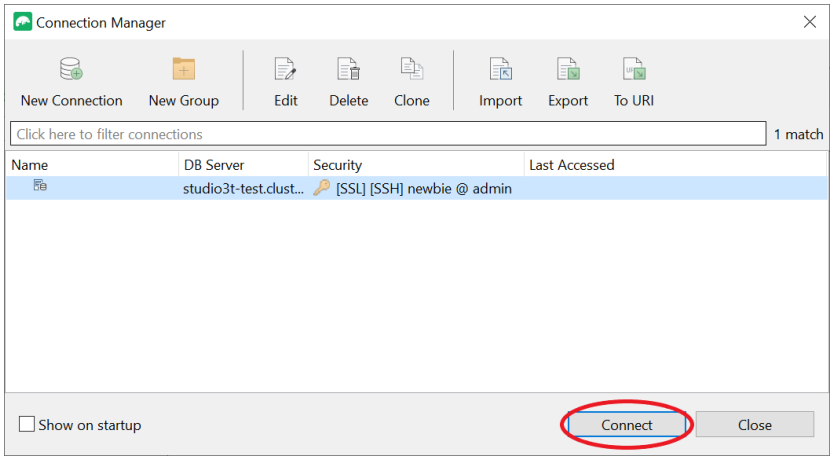
10. 診斷視窗應載入綠色長條，表示測試成功。現在，選擇確定以關閉診斷視窗。



11. 選擇儲存以儲存您的連線以供日後使用。



12. 現在選取您的叢集，然後選擇連線。



恭喜您！您現在已成功透過 Studio 3T 連線至 Amazon DocumentDB 叢集。

使用 DataGrip 連線至 Amazon DocumentDB

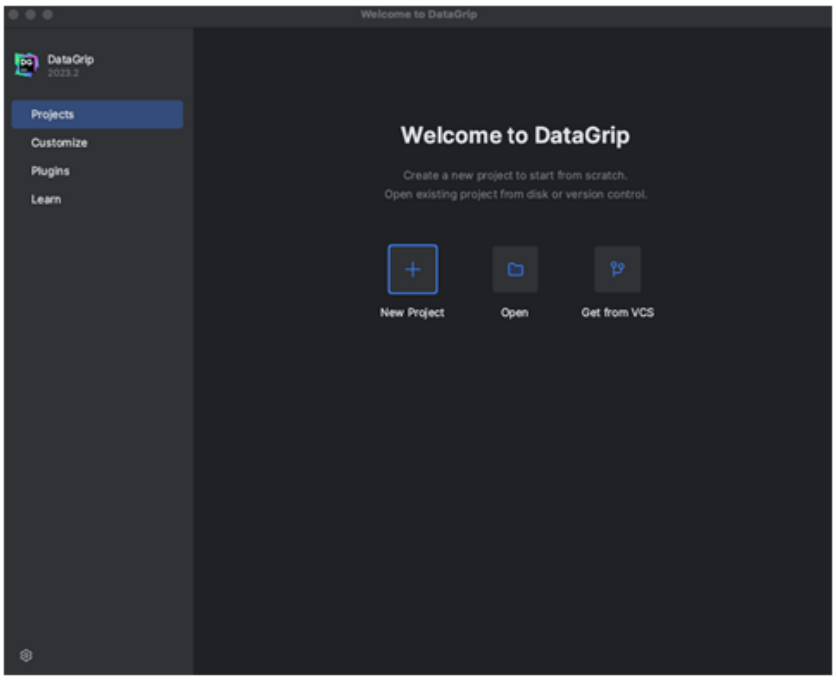
DataGrip 是強大的整合式開發環境 (IDE)，支援各種資料庫系統，包括 Amazon DocumentDB。本節會逐步解說使用 DataGrip 連線至 Amazon DocumentDB 叢集的步驟，讓您使用圖形界面輕鬆管理和查詢資料。

先決條件

- 安裝在機器上的 DataGrip IDE。您可以從 JetBrains 下載。
- 在與 Amazon DocumentDB 叢集相同的 VPC 中執行的 Amazon EC2 執行個體。 Amazon DocumentDB 您將使用此執行個體來建立從本機電腦到 Amazon DocumentDBcluster 的安全通道。遵循如何 的指示使用 Amazon EC2 連線。
- Amazon EC2 執行個體、VPN 連線或您已使用安全 VPN 存取 AWS 基礎設施的替代方案。如果您偏好此選項，請依照指示使用 安全地存取 Amazon DocumentDB AWS Client VPN。

使用 DataGrip 連線

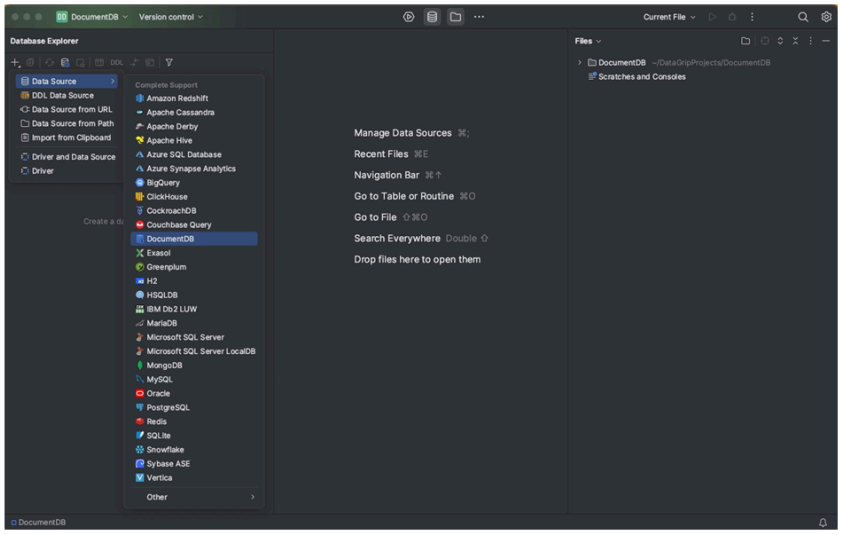
1. 在電腦上啟動 DataGrip 並建立新的專案。



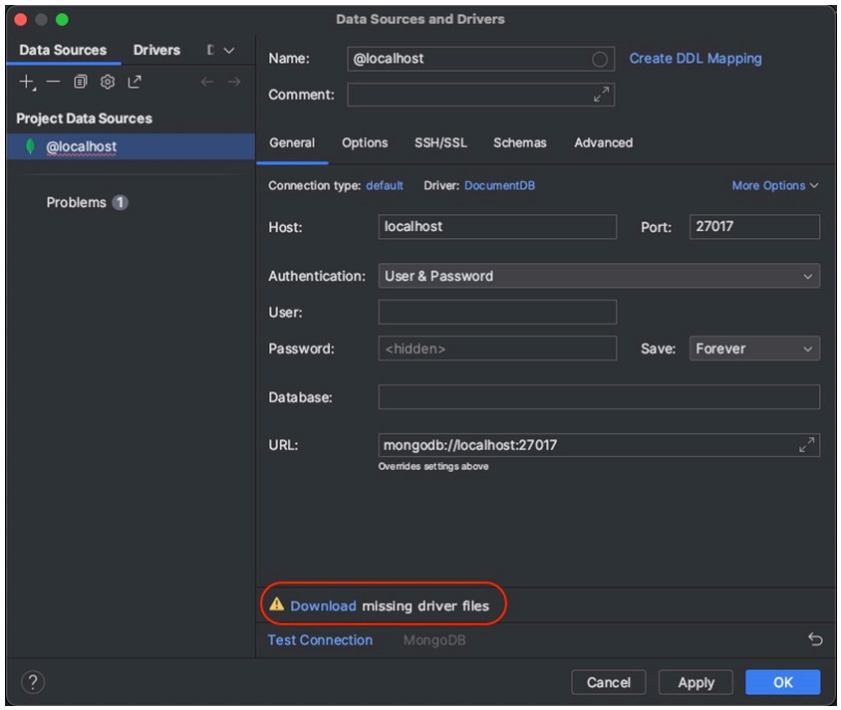
2. 使用下列其中一種方式新增資料來源：

   a. 從主功能表中，導覽至檔案 – 新增 – 資料來源，然後選取 DocumentDB

b. 在 Database Explorer 中，選擇工具列中的新圖示 (+)。導覽至資料來源，然後選取 DocumentDB。



3. 在「一般」索引標籤的資料來源頁面上，檢查連線設定區域底部是否有下載缺少驅動程式檔案連結。 選擇此連結可下載與資料庫互動所需的驅動程式。如需直接下載連結，請參閱 JetBrains JDBC 驅動程式。



4. 在一般索引標籤中，指定連線詳細資訊：

a. 在主機欄位中，指定 Amazon DocumentDB 叢集端點。

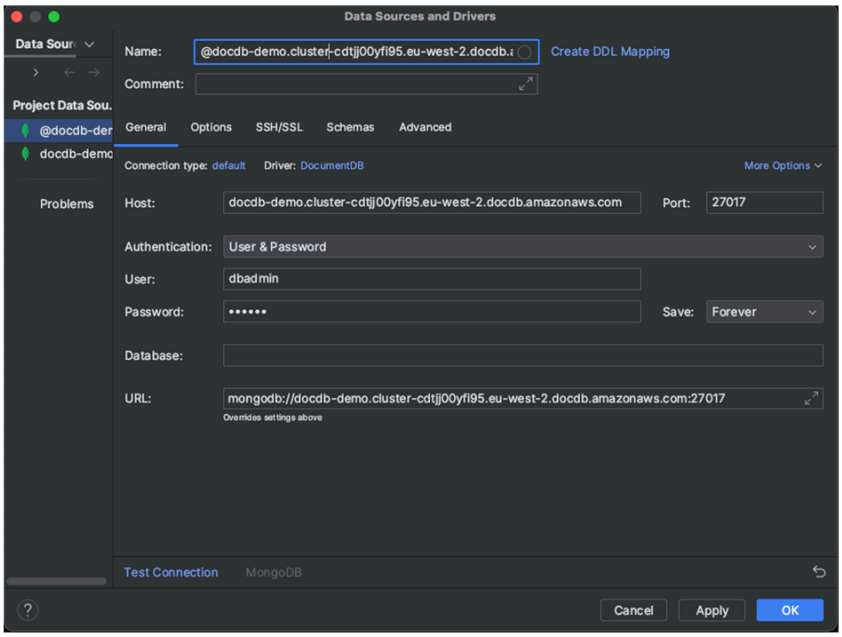
b. 連接埠已設定為 27017。如果您的叢集部署在不同連接埠上，請進行變更。

c. 針對身分驗證，選擇使用者和密碼。

d. 輸入您的使用者名稱和密碼資訊。

e. 資料庫欄位為選用。您可以指定要連線的資料庫。

f. 當您新增上述詳細資訊時，URL 欄位會自動完成。



5. 在 SSH/SSL 索引標籤中，啟用使用 SSH 通道，然後選擇 圖示以開啟 SSH 組態對話方塊。輸入下列資訊：

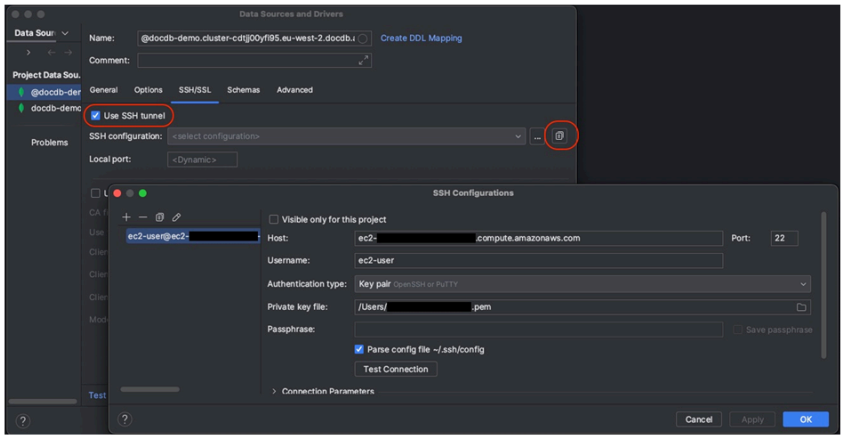
a. 在主機欄位中，輸入 Amazon EC2 執行個體的主機名稱。

b. 輸入 Amazon EC2 執行個體的使用者名稱和密碼。

c. 針對身分驗證類型，選擇金鑰對。

d. 輸入您的私有金鑰檔案。

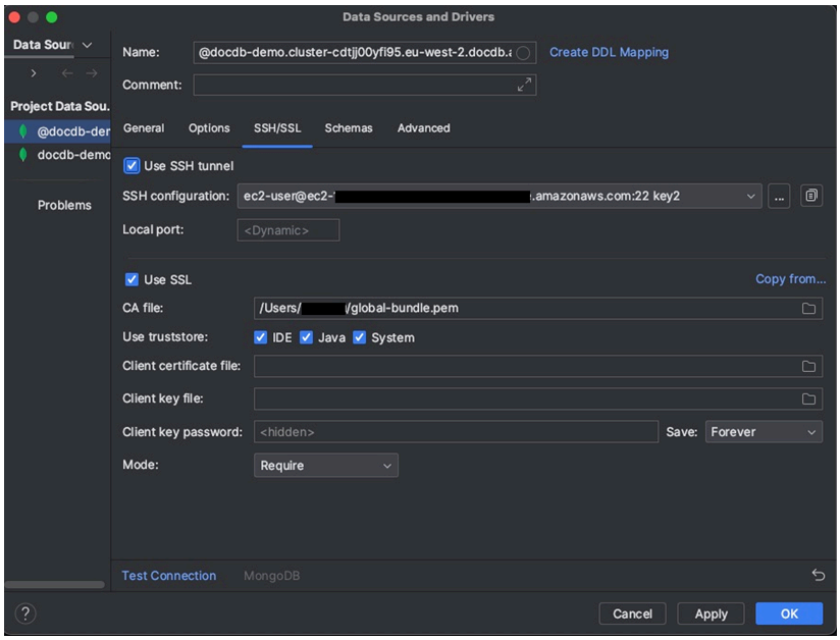
> **Note**
>
> 如果您使用 VPN 選項，則不需要設定 SSH 通道。



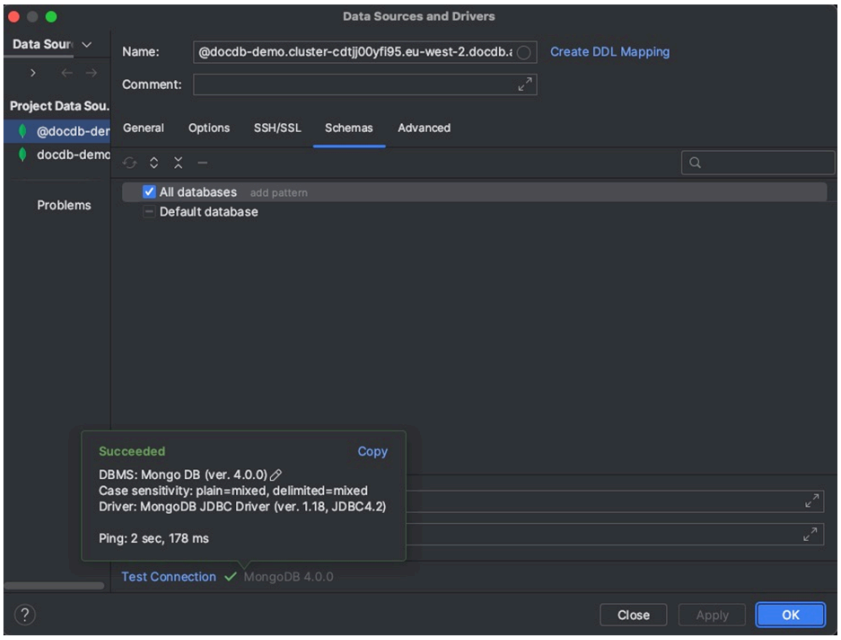
6. 在 SSH/SSL 索引標籤中，啟用使用 SSL。在 CA 檔案欄位中，輸入電腦上`global-bundle.pem`檔案的位置。若為 模式，請保留 需要的選項。

> **Note**
>
> 您可以從此位置或使用此命令下載憑證：wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem

> **Note**
>
> 如果您要連線至 Amazon DocumentDB 彈性叢集，則不需要指定 CA 檔案。保持勾選使用 SSL 選項，並將所有其他選項保留為預設值。



7. 在結構描述索引標籤中，選擇所有資料庫，或在結構描述模式欄位中輸入篩選條件「*：*」。選擇測試連線以測試連線。



8. 成功測試連線後，請選擇確定以儲存資料來源組態。

DataGrip 功能

DataGrip 提供各種功能，協助您有效率地使用 Amazon DocumentDB：

- SQL 編輯器 — 使用 DataGrip 中的 SQL 編輯器，在 DocumentDB 集合上寫入和執行類似 SQL 的查詢。
- Visual Query Builder — 使用視覺化查詢建置器以圖形方式建立查詢，而無需撰寫 SQL 程式碼。
- 結構描述管理 — 輕鬆管理您的資料庫結構描述，包括建立、更改和捨棄集合。
- 資料視覺化：使用 DataGrip 中提供的各種視覺化工具來檢視和分析您的資料。
- 匯出和匯入資料 — 使用 DataGrip 的匯出和匯入功能，在 Amazon DocumentDB 和其他資料庫之間傳輸資料。

如需使用 Amazon DocumentDB 和其他資料庫系統的更多進階功能和秘訣，請參閱官方 DataGrip 文件。

使用 Amazon EC2 連線

本節說明如何設定 Amazon DocumentDB 叢集與 Amazon EC2 之間的連線，並從 Amazon EC2 執行個體存取 Amazon DocumentDB 叢集。 Amazon EC2

有兩種設定 EC2 連線的選項：

- 自動將 EC2 執行個體連線至 Amazon DocumentDB 資料庫 — 使用 EC2 主控台中的自動連線功能，自動設定 EC2 執行個體與新的或現有的 Amazon DocumentDB 資料庫之間的連線。此連線可讓流量在 EC2 執行個體和 Amazon DocumentDB 資料庫之間傳輸。此選項通常用於測試和建立新的安全群組。
- 手動將 EC2 執行個體連線至 Amazon DocumentDB 資料庫 — 透過手動設定和指派安全群組來重現自動連線功能建立的組態，以設定 EC2 執行個體與 Amazon DocumentDB 資料庫之間的連線。此選項通常用於變更更進階的設定，以及使用現有的安全群組。

先決條件

無論 選項為何，在建立第一個 Amazon DocumentDB 叢集之前，您必須執行下列動作：

建立 Amazon Web Services (AWS) 帳戶

您必須先擁有 Amazon Web Services () 帳戶，才能開始使用 Amazon DocumentDB。AWS AWS 該帳戶是免費的。您僅需按使用的服務和資源量付費。

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

   當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

或者，設定 needed AWS Identity and Access Management (IAM) 許可。

管理 Amazon DocumentDB 資源的存取權，例如叢集、執行個體和叢集參數群組，需要 AWS 可用於驗證請求的登入資料。如需詳細資訊，請參閱Amazon DocumentDB 的 Identity and Access Management。

1. 在 的搜尋列中 AWS 管理主控台，在出現的下拉式選單中輸入**IAM**並選取 IAM。
2. 進入 IAM 主控台後，請從導覽窗格中選取使用者。
3. 選取您的使用者名稱。
4. 選擇新增許可。
5. 選取直接連接現有政策。
6. 在搜尋列`AmazonDocDBFullAccess`中輸入 ，並在搜尋結果中出現時選取它。
7. 選擇下一步：檢閱。
8. 選擇新增許可。

建立 Amazon Virtual Private Cloud (Amazon VPC)

視 AWS 區域 您所在的位置而定，您可能已建立或可能尚未建立預設 VPC。如果您沒有預設 VPC，請完成 Amazon VPC 使用者指南中 Amazon VPC 入門的步驟 1。 這需要不到五分鐘的時間。

自動連接 Amazon EC2

主題

- 自動將 EC2 執行個體連線至新的 Amazon DocumentDB 資料庫

- 自動將 EC2 執行個體連線至現有的 Amazon DocumentDB 資料庫
- 與 EC2 執行個體自動連線的概要
- 檢視已連線的運算資源

在 EC2 執行個體與新的 Amazon DocumentDB 資料庫之間設定連線之前，請確定您符合中所述的要求與 EC2 執行個體自動連線的概要。如果您在設定連線之後變更安全群組，這些變更可能會影響 EC2 執行個體和 Amazon DocumentDB 資料庫之間的連線。

> **Note**
>
> 您只能使用 自動設定 EC2 執行個體與 Amazon DocumentDB 資料庫之間的連線 AWS 管理主控台。您無法自動設定與 AWS CLI 或 Amazon DocumentDB API 的連線。

自動將 EC2 執行個體連線至新的 Amazon DocumentDB 資料庫

下列程序假設您已完成 先決條件主題中的步驟。

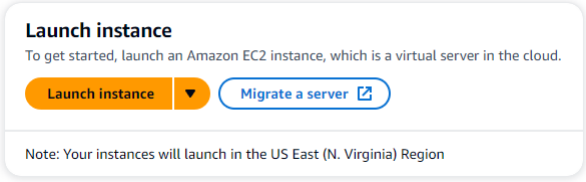
步驟

- 步驟 1：建立 Amazon EC2 執行個體
- 步驟 2：建立 Amazon DocumentDB 叢集
- 步驟 3：連線至您的 Amazon EC2 執行個體
- 步驟 4：安裝 MongoDB Shell
- 步驟 5：管理 Amazon DocumentDB TLS
- 步驟 6：連線至您的 Amazon DocumentDB 叢集
- 步驟 7：插入和查詢資料
- 步驟 8：探索

步驟 1：建立 Amazon EC2 執行個體

在此步驟中，您將在稍後用來佈建 Amazon DocumentDB 叢集的相同區域和 Amazon VPC 中建立 Amazon EC2 執行個體。 Amazon DocumentDB

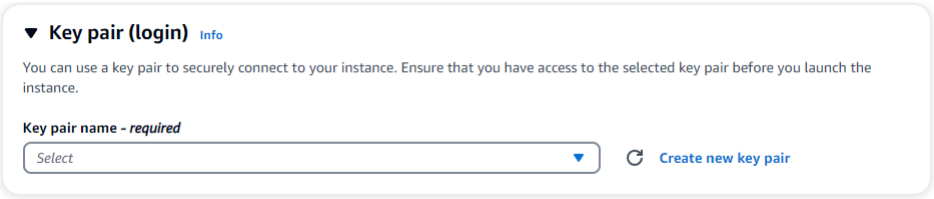
1. 在 Amazon EC2 主控台上，選擇啟動執行個體。



2. 在名稱和標籤區段的名稱欄位中輸入名稱或識別符。
3. 在 Amazon Machine Image (AMI) 下拉式清單中，找到 Amazon Linux 2 AMI 並選擇它。

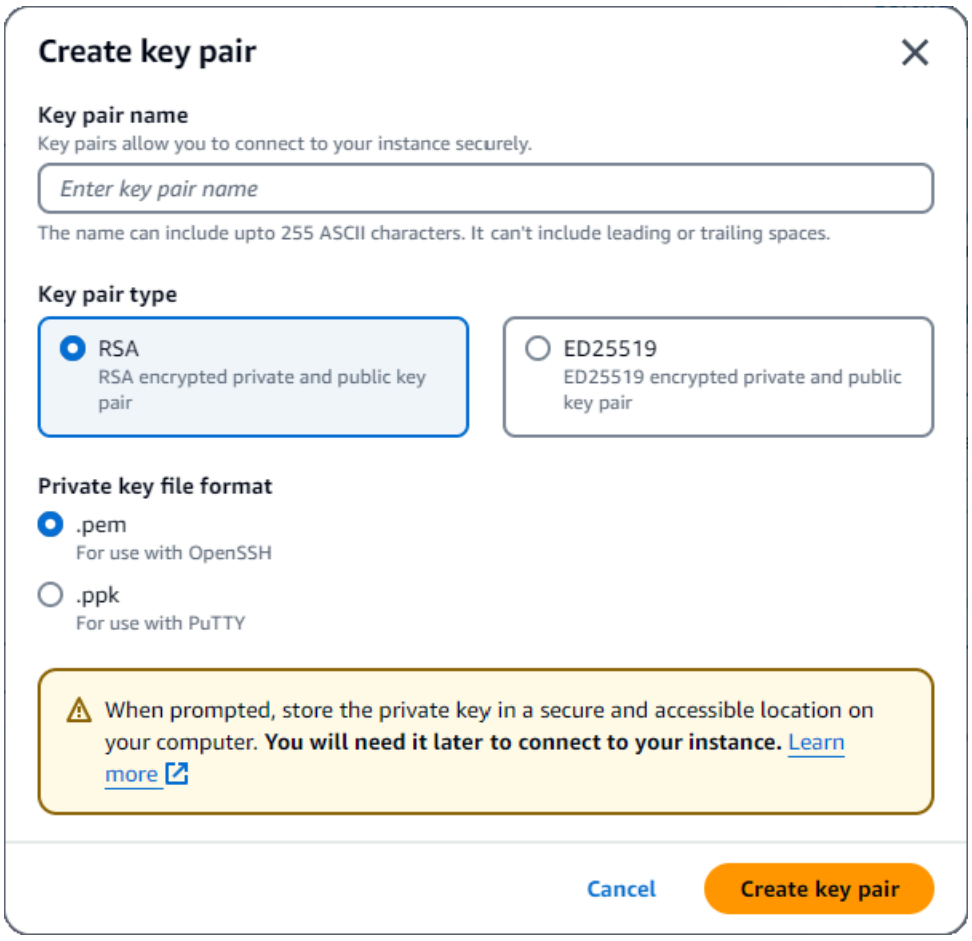


4. 在執行個體類型下拉式清單中尋找並選擇 t3.micro。
5. 在金鑰對 （登入） 區段中，輸入現有金鑰對的識別符，或選擇建立新金鑰對。



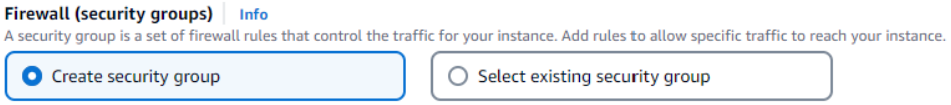
您必須提供 Amazon EC2 金鑰對。

- 如果您有 Amazon EC2 金鑰對：
  1. 選取金鑰對，從清單中選擇您的金鑰對。
  2. 您必須已有私有金鑰檔案 (.pem 或 .ppk 檔案） 可供登入 Amazon EC2 執行個體。
- 如果您沒有 Amazon EC2 金鑰對：
  1. 選擇建立新金鑰對，即會顯示建立金鑰對對話方塊。
  2. 在金鑰對名稱欄位中輸入名稱。
  3. 選擇金鑰對類型和私有金鑰檔案格式。
  4. 選擇 Create key pair (建立金鑰對)。



> **Note**
>
> 基於安全考量，我們強烈建議針對 SSH 和網際網路連線至 EC2 執行個體，使用金鑰對。

6. 選用：在網路設定區段的防火牆（安全群組）下，選擇建立安全群組。



選擇建立安全群組（檢查所有流量允許適用於 EC2 連線的規則）。

> **Note**
>
> 如果您想要使用現有的安全群組，請遵循 中的指示手動連接 Amazon EC2。

7. 在摘要區段中，檢閱您的 EC2 組態，如果正確，請選擇啟動執行個體。

步驟 2：建立 Amazon DocumentDB 叢集

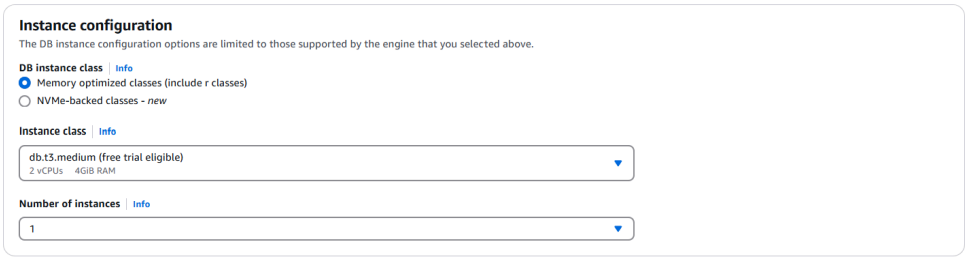
佈建 Amazon EC2 執行個體時，請建立 Amazon DocumentDB 叢集。

1. 導覽至 Amazon DocumentDB 主控台，然後從導覽窗格中選擇叢集。
2. 選擇建立。
3. 將叢集類型設定保留為執行個體型叢集的預設值。
4. 在叢集組態中，針對叢集識別符輸入唯一名稱。請注意，無論如何輸入叢集名稱，主控台都會將所有叢集名稱變更為小寫。

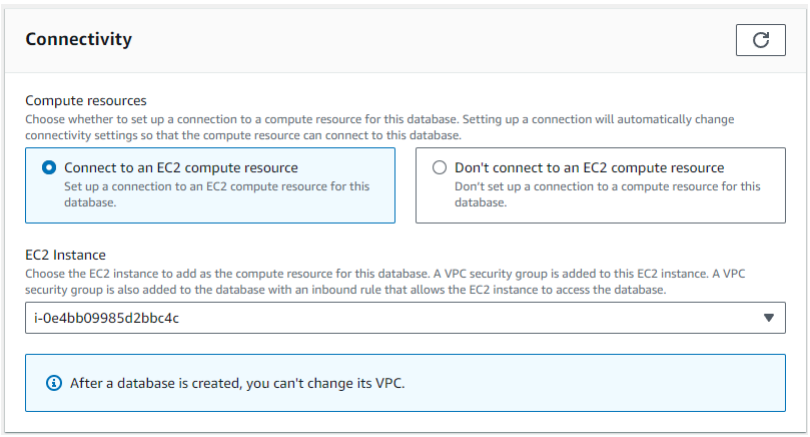
   將引擎版本保留為預設值 5.0.0。
5. 對於叢集儲存組態，請保留 Amazon DocumentDB Standard 的預設設定。
6. 在執行個體組態中：

   - 針對資料庫執行個體類別，選擇記憶體最佳化類別（包括 r 類別）（這是預設值）。

     另一個執行個體選項是 NVMe 支援的類別。如需詳細資訊，請參閱 NVMe 支援的執行個體。
   - 針對執行個體類別，選擇符合您需求的執行個體類型。如需執行個體類別的詳細說明，請參閱執行個體類別規格。
   - 針對執行個體數量，請選擇最能反映您需求的數字。請記住，數字越低，成本越低，叢集可管理的讀取/寫入磁碟區就越低。



7. 針對連線，選擇連線至 EC2 運算資源。這是您在步驟 1 中建立的 EC2 執行個體。



> **Note**
>
> 連線至 EC2 運算資源會自動為與 Amazon DocumentDB 叢集的 EC2 運算資源連線建立安全群組。當您完成建立叢集，並想要查看新建立的安全群組時，請導覽至叢集清單，然後選擇叢集的識別符。在連線與安全索引標籤中，前往安全群組，並在安全群組名稱 (ID) 下尋找您的群組。它將包含叢集的名稱，看起來像這樣：`docdb-ec2-docdb-2023-12-11-21-33-41:i-0e4bb09985d2bbc4c (sg-0238e0b0bf0f73877)`。

8. 在身分驗證區段中，輸入主要使用者的使用者名稱，然後選擇自我管理。輸入密碼，然後進行確認。

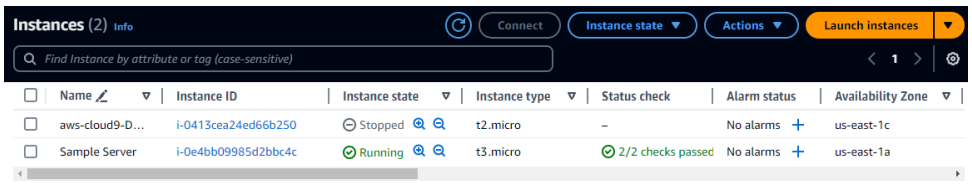
   如果您改為選擇 中的受管 AWS Secrets Manager，請參閱 使用 Amazon DocumentDB 和 進行密碼管理 AWS Secrets Manager 以取得詳細資訊。

9. 選擇 Create Cluster (建立叢集)。

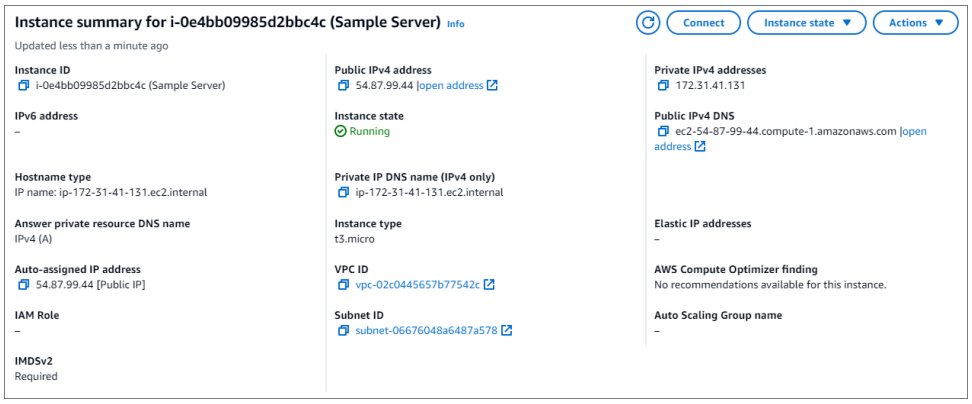
步驟 3：連線至您的 Amazon EC2 執行個體

若要安裝 mongo shell，您必須先連線到 Amazon EC2 執行個體。安裝 mongo shell 可讓您連線至 並查詢 Amazon DocumentDB 叢集。請完成下列步驟：

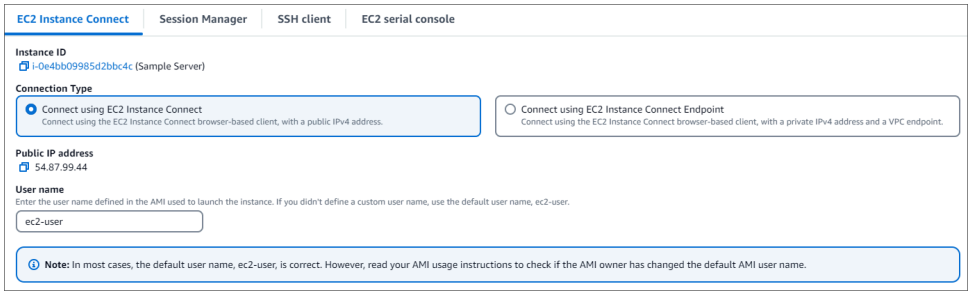
1. 在 Amazon EC2 主控台上，導覽至您的執行個體，並查看您剛建立的執行個體是否正在執行。如果是，請按一下執行個體 ID 來選取執行個體。



2. 選擇連線。



3. 您的連線方法有四個標籤選項：Amazon EC2 Instance Connect、Session Manager、SSH 用戶端或 EC2 序列主控台。您必須選擇一個 ，並遵循其指示。完成後，請選擇連線。



> **Note**
>
> 如果您的 IP 地址在您開始此演練後變更，或者您稍後會返回您的環境，您必須更新demoEC2安全群組傳入規則，以啟用來自新 API 地址的傳入流量。

步驟 4：安裝 MongoDB Shell

您現在可以安裝 MongoDB shell，這是用來連接和查詢 Amazon DocumentDB 叢集的命令列公用程式。MongoDB shell 目前有兩種版本：最新版本 mongosh 和舊版 mongo shell。

> **Important**
>
> 6.13.1 版之前的 Node.js 驅動程式有已知的限制，Amazon DocumentDB 的 IAM 身分驗證目前不支援這些限制。使用 Node.js 驅動程式的 Node.js 驅動程式和工具 （ 例如 mongosh) 必須升級為使用 Node.js 驅動程式 6.13.1 版或更新版本。

請依照下列指示，為您的作業系統安裝 MongoDB shell。

On Amazon Linux

在 Amazon Linux 上安裝 MongoDB shell

如果您不是使用 IAM 身分驗證，而且想要使用最新的 MongoDB shell (mongosh) 連線到您的 Amazon DocumentDB 叢集，請依照下列步驟進行：

1. 建立儲存庫檔案。在您建立的 EC2 執行個體命令列中，執行下列命令：

```
echo -e "[mongodb-org-5.0] \nname=MongoDB Repository\nbaseurl=https://
repo.mongodb.org/yum/amazon/2023/mongodb-org/5.0/x86_64/\ngpgcheck=1 \nenabled=1
 \ngpgkey=https://pgp.mongodb.com/server-5.0.asc" | sudo tee /etc/yum.repos.d/
mongodb-org-5.0.repo
```

2. 完成後，請在命令提示中使用以下兩個命令選項之一安裝 mongosh：

   選項 1 — 如果您在 Amazon EC2 組態期間選擇預設的 Amazon Linux 2023，請輸入此命令：

```
sudo yum install -y mongodb-mongosh-shared-openssl3
```

   選項 2 — 如果您在 Amazon EC2 組態期間選擇 Amazon Linux 2，請輸入此命令：

```
sudo yum install -y mongodb-mongosh
```

如果您使用 IAM 身分驗證，則必須使用舊版的 MongoDB shell (5.0) 來連線至 Amazon DocumentDB 叢集，請依照下列步驟執行：

1. 建立儲存庫檔案。在您建立的 EC2 執行個體命令列中，執行下列命令：

```
echo -e "[mongodb-org-5.0] \nname=MongoDB Repository\nbaseurl=https://
repo.mongodb.org/yum/amazon/2023/mongodb-org/5.0/x86_64/\ngpgcheck=1 \nenabled=1
 \ngpgkey=https://pgp.mongodb.com/server-5.0.asc" | sudo tee /etc/yum.repos.d/
mongodb-org-5.0.repo
```

2. 完成後，請在命令提示中使用以下命令選項安裝 mongodb 5.0 shell：

```
sudo yum install -y mongodb-org-shell
```

On Ubuntu

在 Ubuntu 上安裝 mongosh

1. 匯入公開金鑰供套件管理系統使用。

```
curl -fsSL https://pgp.mongodb.com/server-5.0.asc | sudo gpg --dearmor -o /usr/
share/keyrings/mongodb-server-5.0.gpg
```

2. 以適用於您 Ubuntu 版本的命令來為 MongoDB 建立清單檔案 `mongodb-org-5.0.list`。

```
echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-
server-5.0.gpg ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/5.0
 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-5.0.list
```

3. 使用下列命令匯入和更新本機套件資料庫：

```
sudo apt-get update
```

4. 安裝 mongosh。

```
sudo apt-get install -y mongodb-mongosh
```

如需在您的 Ubuntu 系統上安裝舊版 MongoDB 的詳細資訊，請參閱在 Ubuntu 上安裝 MongoDB Community 版本。

On other operating systems

若要在其他作業系統安裝 mongo 殼層，請參閱 MongoDB 文件中的安裝 MongoDB Community 版本。

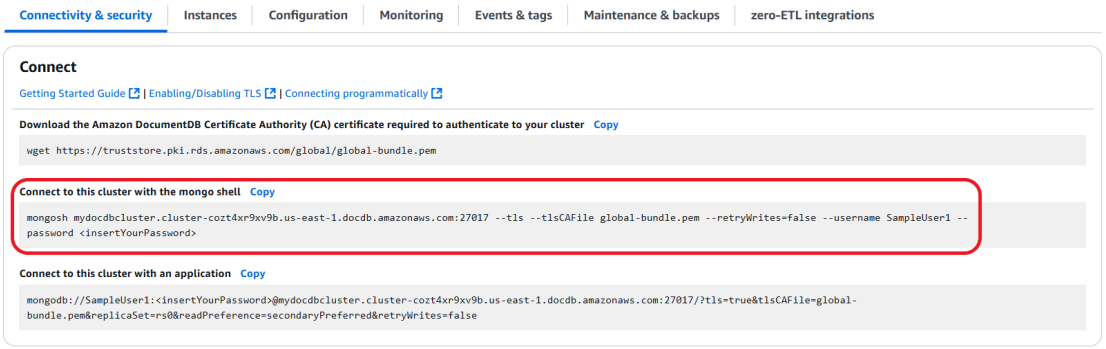
步驟 5：管理 Amazon DocumentDB TLS

使用下列程式碼下載 Amazon DocumentDB 的 CA 憑證： `wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem`

> **Note**
>
> 任何新的 Amazon DocumentDB 叢集預設都會啟用 Transport Layer Security (TLS)。如需詳細資訊，請參閱加密傳輸中的資料。

步驟 6：連線至您的 Amazon DocumentDB 叢集

1. 在 Amazon DocumentDB 主控台的叢集下，找到您的叢集。按一下該叢集的叢集識別符，選擇您建立的叢集。
2. 在連線和安全性索引標籤中，找到連線方塊中具有 mongo shell 的連線至此叢集：



   複製提供的連線字串並將其貼到您的終端機。

   對其進行下列變更：

   a. 請確定您在字串中具有正確的使用者名稱。
   b. 省略 ，<insertYourPassword>以便在連線時 mongo shell 提示您輸入密碼。
   c. 選用：如果您使用 IAM 身分驗證，或使用舊版 MongoDB shell，請修改您的連線字串，如下所示：

      ```
      mongo --ssl --host mydocdbcluster.cluster-cozt4xr9xv9b.us-east-1.docdb.amazonaws.com:27017 --sslCAFile global-bundle.pem --username SampleUser1 --password
      ```

      mydocdbcluster.cluster-cozt4xr9xv9b.us-east-1 將 取代為來自叢集的相同資訊。
3. 在終端機中按 Enter。現在系統會提示您輸入密碼。輸入您的密碼。
4. 當您輸入密碼並看到rs0 [direct: primary] <env-name>>提示時，您已成功連線到 Amazon DocumentDB 叢集。

若要解決連線問題，請參閱 Amazon DocumentDB 故障診斷。

步驟 7：插入和查詢資料

現在您已連線至叢集，您可以執行一些查詢，以熟悉使用文件資料庫。

1. 若要插入單一文件，請輸入下列項目：

```
db.collection.insertOne({"hello":"DocumentDB"})
```

您會收到下列輸出：

```
{
  acknowledged: true,
  insertedId: ObjectId('673657216bdf6258466b128c')
}
```

2. 您可以使用 findOne()命令讀取您撰寫的文件（因為它只會傳回單一文件）。輸入下列項目：

```
db.collection.findOne()
```

您會收到下列輸出：

```
{ "_id" : ObjectId("5e401fe56056fda7321fbd67"), "hello" : "DocumentDB" }
```

3. 若要執行更多查詢，請考慮遊戲設定檔使用案例。首先，將幾個項目插入標題為 的集合profiles。輸入下列項目：

```
db.profiles.insertMany([{ _id: 1, name: 'Matt', status: 'active', level: 12, score:
 202 },
      { _id: 2, name: 'Frank', status: 'inactive', level: 2, score: 9 },
      { _id: 3, name: 'Karen', status: 'active', level: 7, score: 87 },
      { _id: 4, name: 'Katie', status: 'active', level: 3, score: 27 }
])
```

您會收到下列輸出：

```
{ acknowledged: true, insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4 } }
```

4. 使用 find()命令來傳回設定檔集合中的所有文件。輸入下列項目：

```
db.profiles.find()
```

您會收到與您在步驟 3 中輸入的資料相符的輸出。

5. 使用篩選條件對單一文件使用查詢。輸入下列項目：

```
db.profiles.find({name: "Katie"})
```

您會收到下列輸出：

```
{ "_id" : 4, "name" : "Katie", "status": "active", "level": 3, "score":27}
```

6. 現在，讓我們嘗試尋找設定檔，並使用 findAndModify命令修改它。我們會為使用者 Matt 提供額外 10 點的下列程式碼：

```
db.profiles.findAndModify({
        query: { name: "Matt", status: "active"},
        update: { $inc: { score: 10 } }
    })
```

您會收到下列輸出（請注意，他的分數尚未增加）：

```
{
    [{_id : 1, name : 'Matt', status: 'active', level: 12, score: 202}]
```

7. 您可以使用下列查詢，確認他的分數已變更：

```
db.profiles.find({name: "Matt"})
```

您會收到下列輸出：

```
{ "_id" : 1, "name" : "Matt", "status" : "active", "level" : 12, "score" : 212 }
```

步驟 8：探索

恭喜您！您已成功完成 Amazon DocumentDB 的 Quick Start 指南。

下一步是什麼？ 了解如何充分利用這個功能強大的資料庫及其一些熱門功能：

- 管理 Amazon DocumentDB 叢集和執行個體
- 擴展
- 備份和還原

> **Note**
>
> 若要節省成本，您可以停止 Amazon DocumentDB 叢集以降低成本或刪除叢集。根據預設，在閒置 30 分鐘後，您的 AWS Cloud9 環境將停止基礎 Amazon EC2 執行個體。

自動將 EC2 執行個體連線至現有的 Amazon DocumentDB 資料庫

下列程序假設您擁有現有的 Amazon DocumentDB 叢集和現有的 Amazon EC2 執行個體。

存取 Amazon DocumentDB 叢集並設定 Amazon EC2 連線

1. 存取您的 Amazon DocumentDB 叢集。

   a. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。

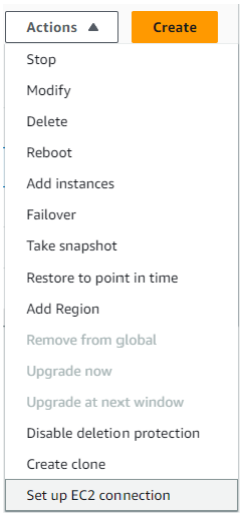
   b. 在導覽窗格中，選擇叢集。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

   c. 選擇叢集名稱左側的按鈕，指定您想要的叢集。

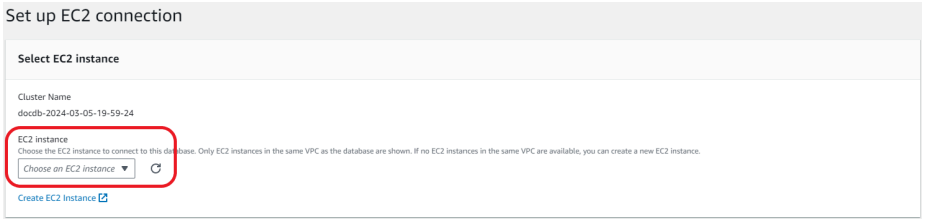
2. 設定 Amazon EC2 連線。

   a. 選擇動作，然後選擇設定 EC2 連線。



設定 EC2 連線對話方塊隨即出現。

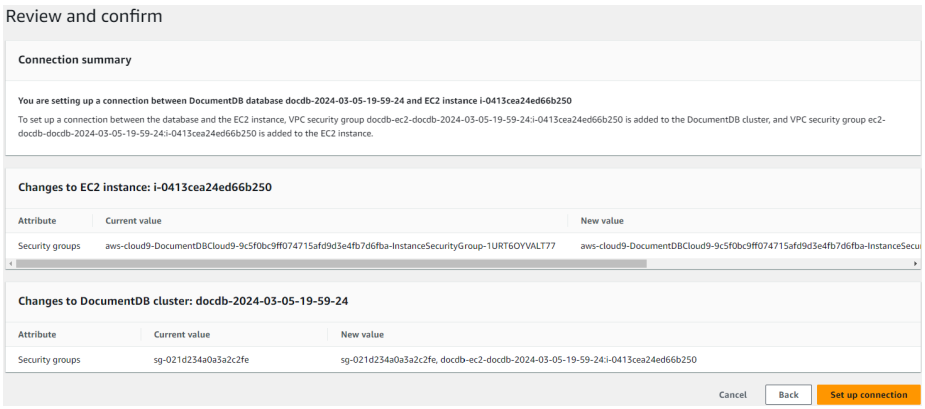
b. 在 EC2 執行個體欄位中，選擇您要連線至叢集的 EC2 執行個體。



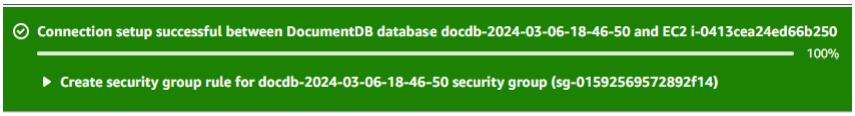
c. 選擇繼續。

檢閱和確認對話方塊隨即出現。

d. 請確定變更正確無誤。然後選擇設定連線。



如果成功，則會顯示下列驗證：



與 EC2 執行個體自動連線的概要

當您設定 EC2 執行個體與 Amazon DocumentDB 資料庫之間的連線時，Amazon DocumentDB 會自動為您的 EC2 執行個體和 Amazon DocumentDB 資料庫設定 VPC 安全群組。

以下是將 EC2 執行個體與 Amazon DocumentDB 資料庫連線的要求：

- EC2 執行個體必須與 Amazon DocumentDB 資料庫存在於相同的 VPC 中。

  如果沒有 EC2 執行個體存在於相同的 VPC 中，則主控台會提供一個連結來建立該執行個體。
- 設定連線的使用者必須擁有執行下列 Amazon EC2 操作的許可：

- ec2:AuthorizeSecurityGroupEgress
- ec2:AuthorizeSecurityGroupIngress
- ec2:CreateSecurityGroup
- ec2:DescribeInstances
- ec2:DescribeNetworkInterfaces
- ec2:DescribeSecurityGroups
- ec2:ModifyNetworkInterfaceAttribute
- ec2:RevokeSecurityGroupEgress

如果資料庫執行個體和 EC2 執行個體位於不同的可用區域，您的帳戶可能會產生跨可用區域成本。

當您設定與 EC2 執行個體的連線時，Amazon DocumentDB 會根據與 Amazon DocumentDB 資料庫和 EC2 執行個體相關聯的安全群組的目前組態運作，如下表所述：

目前的 Amazon DocumentDB 安全群組組態	目前 EC2 安全群組組態	Amazon DocumentDB 動作
有一或多個與 Amazon DocumentDB 資料庫相關聯的安全群組，其名稱符合模式 DocumentDB-ec2-n 。符合模式的安全群組尚未修改。此安全群組只包含一個傳入規則，其具有 EC2 執行個體的 VPC 安全群組做為來源。	有一或多個與 EC2 執行個體相關聯的安全群組，其名稱符合模式 DocumentDB-ec2-n（其中 n 是數字）。符合模式的安全群組尚未修改。此安全群組只有一個傳出規則，其中 Amazon DocumentDB 資料庫的 VPC 安全群組為來源。	Amazon DocumentDB 不採取任何動作。EC2 執行個體和 Amazon DocumentDB 資料庫之間已自動設定連線。由於 EC2 執行個體和 Amazon DocumentDB 資料庫之間已存在連線，因此不會修改安全群組。
以下任一種條件均適用： • 沒有與 Amazon DocumentDB 資料庫相關聯的安全群組，其名稱符合模式 DocumentDB-ec2-n 。 • 有一或多個與 Amazon DocumentDB 相關聯的安全群組，其名稱符合模式 DocumentDB-ec2-n 。不	以下任一種條件均適用： • 沒有與 EC2 執行個體相關聯的安全群組，其名稱符合模式 ec2-DocumentDB-n 。 • 有一或多個與 EC2 執行個體相關聯的安全群組，其名稱符合模式 ec2-DocumentDB-n 。不過，Amazo	Amazon DocumentDB 動作：建立新的安全群組

目前的 Amazon DocumentDB 安全群組組態	目前 EC2 安全群組組態	Amazon DocumentDB 動作
過，Amazon DocumentDB 無法將這些安全群組用於與 EC2 執行個體的連線。Amazon DocumentDB 無法使用沒有傳入規則的安全群組，並將 EC2 執行個體的 VPC 安全群組做為來源。Amazon DocumentDB 也無法使用已修改的安全群組。修改範例包括新增規則或變更現有規則的連線埠。	n DocumentDB 無法將這些安全群組用於與 Amazon DocumentDB 資料庫的連線。Amazon DocumentDB 無法使用沒有一個傳出規則的安全群組，並將 Amazon DocumentDB 資料庫的 VPC 安全群組做為來源。Amazon DocumentDB 也無法使用已修改的安全群組。	
有一或多個與 Amazon DocumentDB 資料庫相關聯的安全群組，其名稱符合模式 `DocumentDB-ec2-n` 。符合模式的安全群組尚未修改。此安全群組只包含一個傳入規則，其具有 EC2 執行個體的 VPC 安全群組做為來源。	有一或多個與 EC2 執行個體相關聯的安全群組，其名稱符合模式 `ec2-DocumentDB-n` 。不過，Amazon DocumentDB 無法將這些安全群組用於與 Amazon DocumentDB 資料庫的連線。Amazon DocumentDB 無法使用沒有一個傳出規則的安全群組，並將 Amazon DocumentDB 資料庫的 VPC 安全群組做為來源。Amazon DocumentDB 也無法使用已修改的安全群組。	Amazon DocumentDB 動作：建立新的安全群組
有一或多個與 Amazon DocumentDB 資料庫相關聯的安全群組，其名稱符合模式 `DocumentDB-ec2-n` 。符合模式的安全群組尚未修改。此安全群組只包含一個傳入規則，其具有 EC2 執行個體的 VPC 安全群組做為來源。	存在用於連線的有效 EC2 安全群組，但與 EC2 執行個體沒有相關聯。此安全群組具有符合模式 `DocumentDB-ec2-n` 的名稱。尚未將其修改。它只有一個傳出規則，並將 Amazon DocumentDB 資料庫的 VPC 安全群組做為來源。	Amazon DocumentDB 動作：關聯 EC2 安全群組

目前的 Amazon DocumentDB 安全群組組態	目前 EC2 安全群組組態	Amazon DocumentDB 動作
以下任一種條件均適用： • 沒有與 Amazon DocumentDB 資料庫相關聯的安全群組，其名稱符合模式 `DocumentDB-ec2-n` 。 • 有一或多個與 Amazon DocumentDB 資料庫相關聯的安全群組，其名稱符合模式 `DocumentDB-ec2-n` 。不過，Amazon DocumentDB 無法將這些安全群組用於與 EC2 執行個體的連線。Amazon DocumentDB 無法使用沒有傳入規則的安全群組，並將 EC2 執行個體的 VPC 安全群組做為來源。Amazon DocumentDB 也無法使用已修改的安全群組。	有一或多個與 EC2 執行個體相關聯的安全群組，其名稱符合模式 `DocumentDB-ec2-n`。符合模式的安全群組尚未修改。此安全群組只有一個傳出規則，其中 Amazon DocumentDB 資料庫的 VPC 安全群組為來源。	Amazon DocumentDB 動作：建立新的安全群組

Amazon DocumentDB 動作：建立新的安全群組

Amazon DocumentDB 會採取下列動作：

- 建立符合模式 `DocumentDB-ec2-n` 的新安全群組。此安全群組具有一個傳入規則，其具有 EC2 執行個體的 VPC 安全群組做為來源。此安全群組與 Amazon DocumentDB 資料庫相關聯，並允許 EC2 執行個體存取 Amazon DocumentDB 資料庫。
- 建立符合模式 `ec2-DocumentDB-n` 的新安全群組。此安全群組具有傳出規則，並將 Amazon DocumentDB 資料庫的 VPC 安全群組做為來源。此安全群組與 EC2 執行個體相關聯，並允許 EC2 執行個體將流量傳送至 Amazon DocumentDB 資料庫。

Amazon DocumentDB 動作：關聯 EC2 安全群組

Amazon DocumentDB 會將有效的現有 EC2 安全群組與 EC2 執行個體建立關聯。此安全群組允許 EC2 執行個體將流量傳送至 Amazon DocumentDB 資料庫。

檢視已連線的運算資源

您可以使用 AWS 管理主控台 檢視連線至 Amazon DocumentDB 資料庫的運算資源。顯示的資源包括已自動設定的運算資源連線。您可以使用下列方式自動設定與運算資源的連線：

- 您可以在建立資料庫時選取運算資源。如需詳細資訊，請參閱建立 Amazon DocumentDB 叢集和建立多可用區域資料庫叢集。
- 您可以設定現有資料庫與運算資源之間的連線。如需詳細資訊，請參閱自動連接 Amazon EC2。

列出的運算資源不包括已手動連線至資料庫的運算資源。例如，您可以透過將規則新增至與資料庫關聯的 VPC 安全群組，來允許運算資源手動存取資料庫。

針對要列出的運算資源，必須滿足下列條件：

- 與運算資源相關聯的安全群組名稱符合模式 `ec2-DocumentDB-n`（其中 n 是數字）。
- 與運算資源相關聯的安全群組具有傳出規則，其連接埠範圍設定為 Amazon DocumentDB 資料庫使用的連接埠。
- 與運算資源相關聯的安全群組具有傳出規則，其來源設定為與 Amazon DocumentDB 資料庫相關聯的安全群組。
- 與 Amazon DocumentDB 資料庫相關聯的安全群組名稱符合模式 `DocumentDB-ec2-n`（其中 n 是數字）。
- 與 Amazon DocumentDB 資料庫相關聯的安全群組具有傳入規則，其連接埠範圍設定為 Amazon DocumentDB 資料庫使用的連接埠。
- 與 Amazon DocumentDB 資料庫相關聯的安全群組具有傳入規則，來源設定為與運算資源相關聯的安全群組。

檢視連線至 Amazon DocumentDB 資料庫的運算資源

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇資料庫，然後選擇 Amazon DocumentDB 資料庫的名稱。
3. 在連線與安全索引標籤上，檢視連線運算資源區段中的運算資源。

手動連接 Amazon EC2

主題

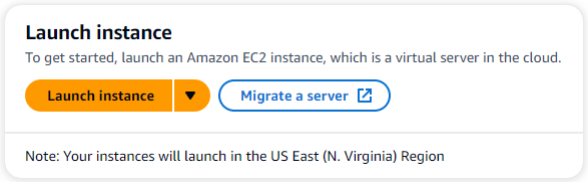
- 步驟 1：建立 Amazon EC2 執行個體
- 步驟 2：建立安全群組
- 步驟 3：建立 Amazon DocumentDB 叢集
- 步驟 4：連線至您的 Amazon EC2 執行個體
- 步驟 5：安裝 MongoDB Shell
- 步驟 6：管理 Amazon DocumentDB TLS
- 步驟 7：連線至您的 Amazon DocumentDB 叢集
- 步驟 8：插入和查詢資料
- 步驟 9：探索

下列步驟假設您已完成 先決條件主題中的步驟。

步驟 1：建立 Amazon EC2 執行個體

在此步驟中，您將在稍後用來佈建 Amazon DocumentDB 叢集的相同區域和 Amazon VPC 中建立 Amazon EC2 執行個體。 Amazon DocumentDB

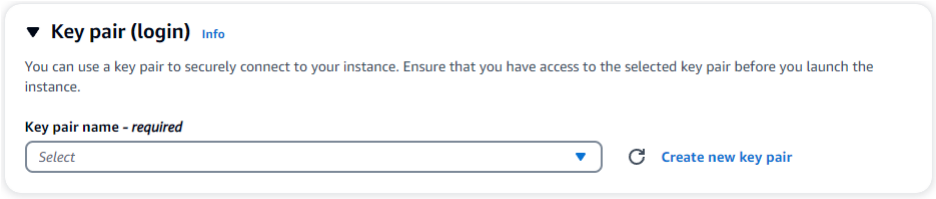
1. 在 Amazon EC2 主控台上，選擇啟動執行個體。



2. 在名稱和標籤區段的名稱欄位中輸入名稱或識別符。
3. 在 Amazon Machine Image (AMI) 下拉式清單中，找到 Amazon Linux 2 AMI 並選擇它。

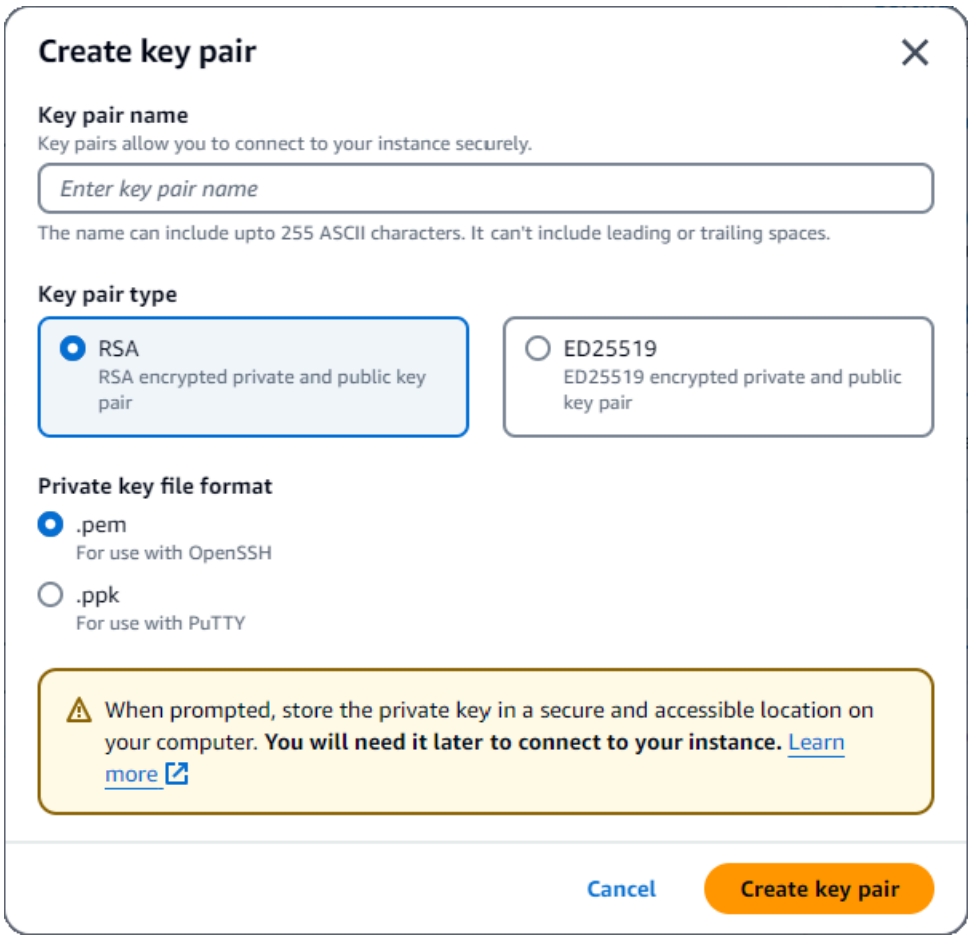


4. 在執行個體類型下拉式清單中尋找並選擇 t3.micro。
5. 在金鑰對 （登入） 區段中，輸入現有金鑰對的識別符，或選擇建立新的金鑰對。



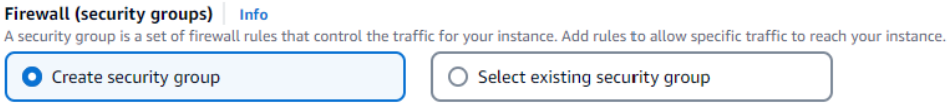
您必須提供 Amazon EC2 金鑰對。

- 如果您有 Amazon EC2 金鑰對：
  1. 選取金鑰對，從清單中選擇您的金鑰對。
  2. 您必須已有私有金鑰檔案 (.pem 或 .ppk 檔案） 可供登入 Amazon EC2 執行個體。
- 如果您沒有 Amazon EC2 金鑰對：
  1. 選擇建立新金鑰對，即會顯示建立金鑰對對話方塊。
  2. 在金鑰對名稱欄位中輸入名稱。
  3. 選擇金鑰對類型和私有金鑰檔案格式。
  4. 選擇 Create key pair (建立金鑰對)。



> **Note**
>
> 基於安全考量，我們強烈建議針對 SSH 和網際網路連線至 EC2 執行個體，使用金鑰對。

6. 在網路設定區段的防火牆 （安全群組） 下，選擇建立安全群組或選取現有的安全群組。



如果您選擇選取現有的安全群組，請從常見安全群組下拉式清單中選取一個。

如果您選擇建立新的安全群組，請執行下列動作：

a. 檢查所有流量允許適用於 EC2 連線的規則。

b. 在 IP 欄位中，選擇我的 IP 或選取自訂，從 CIDR 區塊、字首清單或安全群組清單中選擇。我們不建議選擇 Anywhere，除非您的 EC2 執行個體位於隔離的網路上，因為它允許存取 EC2 執行個體的任何 IP 地址。

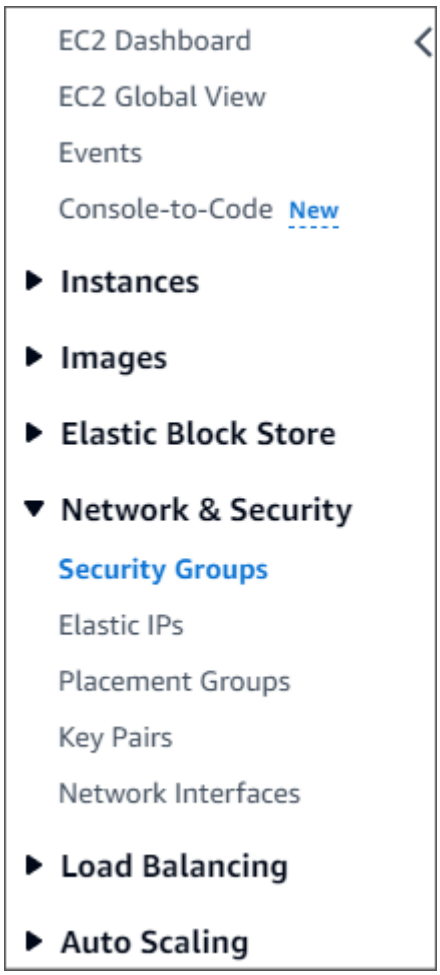


7. 在摘要區段中，檢閱您的 EC2 組態，如果正確，請選擇啟動執行個體。

步驟 2：建立安全群組

您現在將在預設 Amazon VPC 中建立新的安全群組。安全群組demoDocDB可讓您從 Amazon DocumentDB EC2 執行個體連線至連接埠 27017 (Amazon DocumentDB 的預設連接埠） 上的 Amazon DocumentDB 叢集。 Amazon EC2

1. 在 Amazon EC2 管理主控台的網路和安全性下，選擇安全群組。



2. 選擇建立安全群組。



3. 在基本詳細資訊區段中：

   a. 針對 Security group name (安全群組名稱)，輸入 demoDocDB。

   b. 在描述中，輸入描述。

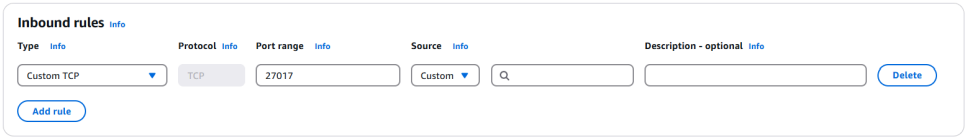
   c. 對於 VPC，接受預設 VPC 的使用。

4. 在 Inbound rules (傳入規則) 區段中，選擇 Add rule (新增規則)。

   a. 針對類型，選擇自訂 TCP 規則 （ 預設 ）。

   b. 針對連接埠範圍，輸入 27017。

   c. 對於 Source (資源)，選擇 Custom (自訂)。在旁邊的欄位中，搜尋您剛在步驟 1 中建立的安全群組。您可能需要重新整理瀏覽器，Amazon EC2 主控台才能自動填入來源名稱。



5. 接受所有其他預設值，然後選擇建立安全群組。



步驟 3：建立 Amazon DocumentDB 叢集

佈建 Amazon EC2 執行個體時，您將建立 Amazon DocumentDB 叢集。

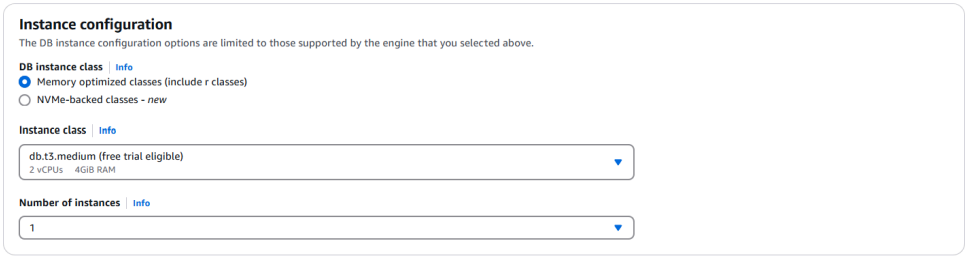
1. 導覽至 Amazon DocumentDB 主控台，然後從導覽窗格中選擇叢集。
2. 選擇建立。
3. 將叢集類型設定保留為執行個體型叢集的預設值。
4. 在叢集組態中，針對叢集識別符輸入唯一名稱。請注意，無論如何輸入叢集名稱，主控台都會將所有叢集名稱變更為小寫。

   將引擎版本保留為預設值 5.0.0。
5. 對於叢集儲存組態，請保留 Amazon DocumentDB Standard 的預設設定。
6. 在執行個體組態中：

   - 針對資料庫執行個體類別，選擇記憶體最佳化類別 （ 包括 r 類別 ） （ 這是預設值 ）。

另一個執行個體選項是 NVMe 支援的類別。如需詳細資訊，請參閱 NVMe 支援的執行個體。

- 針對執行個體類別，選擇符合您需求的執行個體類型。如需執行個體類別的詳細說明，請參閱執行個體類別規格。
- 針對執行個體數量，請選擇最能反映您需求的數字。請記住，數字越低，成本越低，叢集可管理的讀取/寫入磁碟區就越低。



7. 對於連線，請保留預設設定 不連線至 EC2 運算資源。

> **Note**
>
> 連線至 EC2 運算資源會自動為叢集的連線建立安全群組。由於您在上一個步驟中手動建立這些安全群組，因此您應該選取不要連線到 EC2 運算資源，以免建立第二組安全群組。

8. 在身分驗證區段中，輸入主要使用者的使用者名稱，然後選擇自我管理。輸入密碼，然後進行確認。

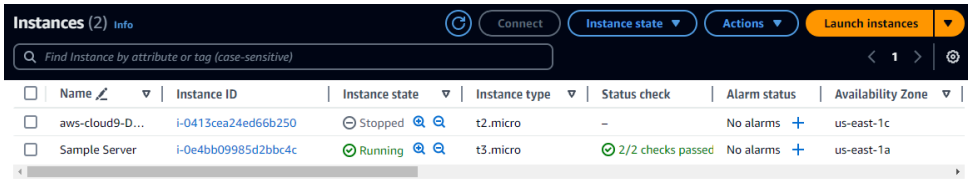
   如果您改為選擇 中的受管 AWS Secrets Manager，請參閱 使用 Amazon DocumentDB 和 進行密碼管理 AWS Secrets Manager 以取得詳細資訊。

9. 選擇 Create Cluster (建立叢集)。

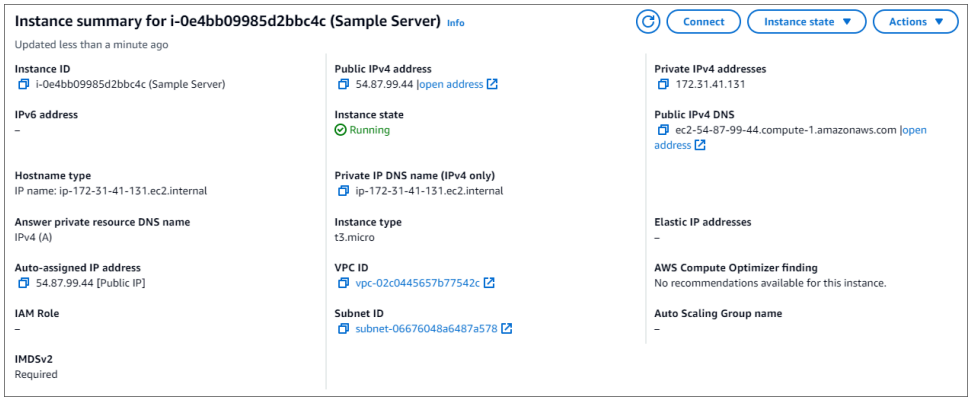
步驟 4：連線至您的 Amazon EC2 執行個體

連線至 Amazon EC2 執行個體可讓您安裝 MongoDB shell。安裝 mongo shell 可讓您連線至 並查詢 Amazon DocumentDB 叢集。請完成下列步驟：

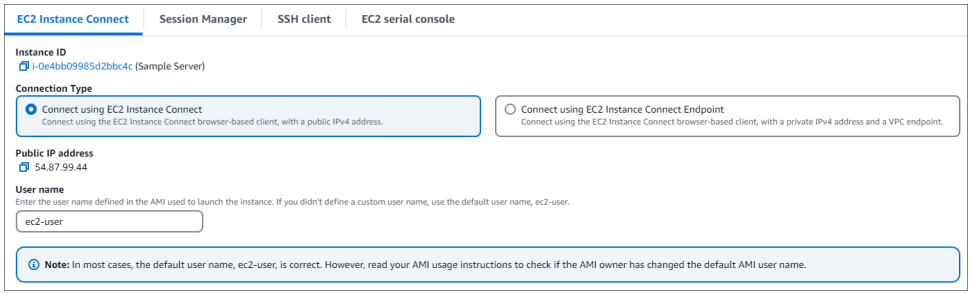
1. 在 Amazon EC2 主控台上，導覽至您的執行個體，並查看您剛建立的執行個體是否正在執行。如果是，請按一下執行個體 ID 來選取執行個體。



2. 選擇連線。



3. 您的連線方法有四個標籤選項：Amazon EC2 Instance Connect、Session Manager、SSH 用戶端或 EC2 序列主控台。您必須選擇一個 ，並遵循其指示。完成後，請選擇連線。



> **Note**
>
> 如果您的 IP 地址在您開始此演練後變更，或者您稍後會返回您的環境，您必須更新demoEC2安全群組傳入規則，以啟用來自新 API 地址的傳入流量。

步驟 5：安裝 MongoDB Shell

您現在可以安裝 MongoDB shell，這是用來連接和查詢 Amazon DocumentDB 叢集的命令列公用程式。MongoDB shell 目前有兩種版本：最新版本 mongosh 和舊版 mongo shell。

> **Important**
>
> 6.13.1 版之前的 Node.js 驅動程式有已知的限制，Amazon DocumentDB 的 IAM 身分驗證目前不支援這些限制。使用 Node.js 驅動程式的 Node.js 驅動程式和工具 （ 例如 mongosh) 必須升級為使用 Node.js 驅動程式 6.13.1 版或更新版本。

請依照下列指示，為您的作業系統安裝 MongoDB shell。

On Amazon Linux

在 Amazon Linux 上安裝 MongoDB shell

如果您不是使用 IAM，而且想要使用最新的 MongoDB shell (mongosh) 連線到您的 Amazon DocumentDB 叢集，請遵循下列步驟：

1. 建立儲存庫檔案。在您建立的 EC2 執行個體命令列中，執行下列命令：

```
echo -e "[mongodb-org-5.0] \nname=MongoDB Repository\nbaseurl=https://
repo.mongodb.org/yum/amazon/2023/mongodb-org/5.0/x86_64/\ngpgcheck=1 \nenabled=1
 \ngpgkey=https://pgp.mongodb.com/server-5.0.asc" | sudo tee /etc/yum.repos.d/
mongodb-org-5.0.repo
```

2. 完成後，請在命令提示中使用以下兩個命令選項之一安裝 mongosh：

   選項 1 — 如果您在 Amazon EC2 組態期間選擇預設的 Amazon Linux 2023，請輸入此命令：

```
sudo yum install -y mongodb-mongosh-shared-openssl3
```

   選項 2 — 如果您在 Amazon EC2 組態期間選擇 Amazon Linux 2，請輸入此命令：

```
sudo yum install -y mongodb-mongosh
```

如果您使用的是 IAM，則必須使用舊版 MongoDB shell (5.0) 來連線至 Amazon DocumentDB 叢集，請依照下列步驟執行：

1. 建立儲存庫檔案。在您建立的 EC2 執行個體命令列中，執行下列命令：

```
echo -e "[mongodb-org-5.0] \nname=MongoDB Repository\nbaseurl=https://
repo.mongodb.org/yum/amazon/2023/mongodb-org/5.0/x86_64/\ngpgcheck=1 \nenabled=1
```

```
 \ngpgkey=https://pgp.mongodb.com/server-5.0.asc" | sudo tee /etc/yum.repos.d/
mongodb-org-5.0.repo
```

2. 完成後，請在命令提示中使用以下命令選項安裝 mongodb 5.0 shell：

```
sudo yum install -y mongodb-org-shell
```

On Ubuntu

在 Ubuntu 上安裝 mongosh

1. 匯入公開金鑰供套件管理系統使用。

```
curl -fsSL https://pgp.mongodb.com/server-5.0.asc | sudo gpg --dearmor -o /usr/
share/keyrings/mongodb-server-5.0.gpg
```

2. 以適用於您 Ubuntu 版本的命令來為 MongoDB 建立清單檔案 `mongodb-org-5.0.list`。

```
echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-
server-5.0.gpg ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/5.0
 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-5.0.list
```

3. 使用下列命令匯入和更新本機套件資料庫：

```
sudo apt-get update
```

4. 安裝 mongosh。

```
sudo apt-get install -y mongodb-mongosh
```

如需在您的 Ubuntu 系統上安裝舊版 MongoDB 的詳細資訊，請參閱在 Ubuntu 上安裝 MongoDB Community 版本。

On other operating systems

若要在其他作業系統安裝 mongo 殼層，請參閱 MongoDB 文件中的安裝 MongoDB Community 版本。

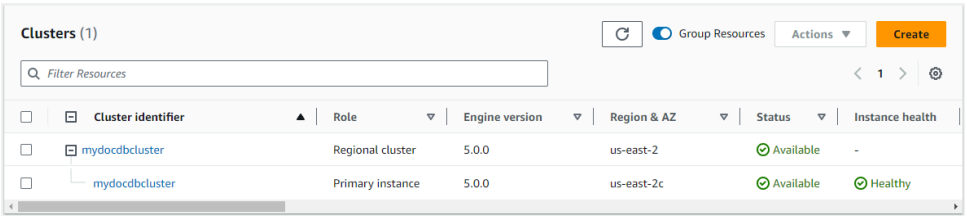
步驟 6：管理 Amazon DocumentDB TLS

使用下列程式碼下載 Amazon DocumentDB 的 CA 憑證： `wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem`

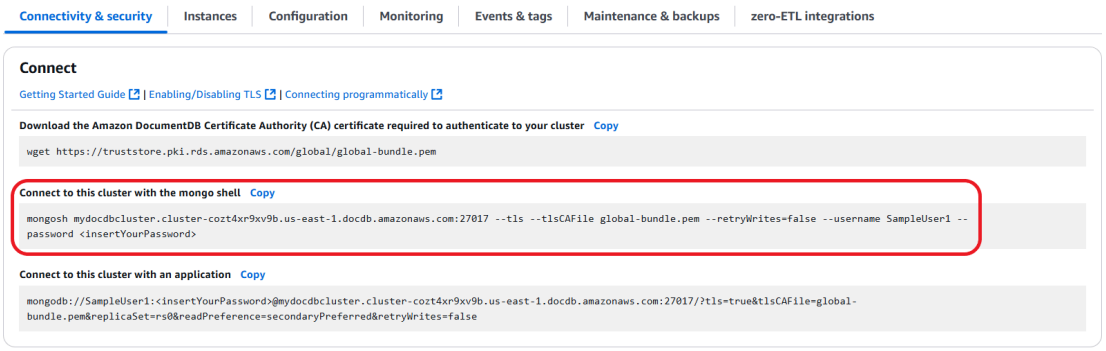
> **Note**
>
> 任何新的 Amazon DocumentDB 叢集預設都會啟用 Transport Layer Security (TLS)。如需詳細資訊，請參閱加密傳輸中的資料。

步驟 7：連線至您的 Amazon DocumentDB 叢集

1. 在 Amazon DocumentDB 主控台的叢集下，找到您的叢集。按一下該叢集的叢集識別符，以選擇您建立的叢集。



2. 在連線和安全性索引標籤中，找到連線方塊中具有 mongo shell 的連線至此叢集：



   複製提供的連線字串並將其貼到您的終端機。

   對其進行下列變更：

   a. 請確定您在字串中具有正確的使用者名稱。

   b. 省略 ，`<insertYourPassword>`以便在連線時 mongo shell 提示您輸入密碼。

   c. 選用：如果您使用 IAM 身分驗證，或使用舊版 MongoDB shell，請修改您的連線字串，如下所示：

```
mongo --ssl --host mydocdbcluster.cluster-cozt4xr9xv9b.us-
east-1.docdb.amazonaws.com:27017 --sslCAFile global-bundle.pem --
username SampleUser1 --password
```

`mydocdbcluster.cluster-cozt4xr9xv9b.us-east-1` 將 取代為來自叢集的相同資訊。

3. 在終端機中按 Enter。現在系統會提示您輸入密碼。輸入您的密碼。
4. 當您輸入密碼並看到`rs0 [direct: primary] <env-name>>`提示時，您已成功連線到 Amazon DocumentDB 叢集。

若要解決連線問題，請參閱 Amazon DocumentDB 故障診斷。

步驟 8：插入和查詢資料

現在您已連線至叢集，您可以執行一些查詢，以熟悉使用文件資料庫。

1. 若要插入單一文件，請輸入下列項目：

```
db.collection.insertOne({"hello":"DocumentDB"})
```

您會收到下列輸出：

```
{
  acknowledged: true,
  insertedId: ObjectId('673657216bdf6258466b128c')
}
```

2. 您可以使用 `findOne()`命令讀取您撰寫的文件（因為它只會傳回單一文件）。輸入下列項目：

```
db.collection.findOne()
```

您會收到下列輸出：

```
{ "_id" : ObjectId("5e401fe56056fda7321fbd67"), "hello" : "DocumentDB" }
```

3. 若要執行更多查詢，請考慮遊戲設定檔使用案例。首先，將幾個項目插入標題為 的集合`profiles`。輸入下列項目：

```
db.profiles.insertMany([{ _id: 1, name: 'Matt', status: 'active', level: 12, score:
 202 },
      { _id: 2, name: 'Frank', status: 'inactive', level: 2, score: 9 },
      { _id: 3, name: 'Karen', status: 'active', level: 7, score: 87 },
      { _id: 4, name: 'Katie', status: 'active', level: 3, score: 27 }
])
```

您會收到下列輸出：

```
{ acknowledged: true, insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4 } }
```

4. 使用 `find()`命令來傳回設定檔集合中的所有文件。輸入下列項目：

```
db.profiles.find()
```

您會收到與您在步驟 3 中輸入的資料相符的輸出。

5. 使用篩選條件對單一文件使用查詢。輸入下列項目：

```
db.profiles.find({name: "Katie"})
```

您會收到下列輸出：

```
{ "_id" : 4, "name" : "Katie", "status": "active", "level": 3, "score":27}
```

6. 現在，讓我們嘗試尋找設定檔，並使用 `findAndModify`命令修改它。我們會為使用者 Matt 提供額外 10 點的下列程式碼：

```
db.profiles.findAndModify({
        query: { name: "Matt", status: "active"},
        update: { $inc: { score: 10 } }
    })
```

您會收到下列輸出（請注意，他的分數尚未增加）：

```
{
    [{_id : 1, name : 'Matt', status: 'active', level: 12, score: 202}]
```

7. 您可以使用下列查詢，確認他的分數已變更：

```
db.profiles.find({name: "Matt"})
```

您會收到下列輸出：

```
{ "_id" : 1, "name" : "Matt", "status" : "active", "level" : 12, "score" : 212 }
```

步驟 9：探索

恭喜您！您已成功完成 Amazon DocumentDB 的 Quick Start 指南。

下一步是什麼？ 了解如何充分利用這個功能強大的資料庫及其一些熱門功能：

- 管理 Amazon DocumentDB 叢集和執行個體
- 擴展
- 備份和還原

> **Note**
>
> 若要節省成本，您可以停止 Amazon DocumentDB 叢集以降低成本或刪除叢集。根據預設，在閒置 30 分鐘後，您的 AWS Cloud9 環境將停止基礎 Amazon EC2 執行個體。

使用 Amazon DocumentDB JDBC 驅動程式進行連線

Amazon DocumentDB 的 JDBC 驅動程式為開發人員提供 SQL 相關界面，並啟用來自 Tableau 和 DbVisualizer 等 BI 工具的連線。

如需更多詳細資訊，請參閱 GitHub 上的 Amazon DocumentDB JDBC 驅動程式文件。

主題

- 開始使用
- 從 Tableau 桌面連線至 Amazon DocumentDB
- 從 DbVisualizer 連線至 Amazon DocumentDB
- JDBC 自動結構描述產生
- SQL 支援和限制
- 疑難排解

開始使用

步驟 1. 建立 Amazon DocumentDB 叢集

如果您沒有建立 Amazon DocumentDB 叢集，請使用 中的指示建立一個叢集開始使用 Amazon DocumentDB。

> **Note**
>
> Amazon DocumentDB 是僅限 Virtual Private Cloud (VPC) 的服務。如果您是從本機電腦連線至叢集的 VPC 外部，則需要建立與 Amazon EC2 執行個體的 SSH 連線。在此情況下，請使用 中的指示啟動叢集使用 Amazon EC2 連線。如需 SSH 通道的詳細資訊，以及何時需要，請參閱使用 SSH 通道連線至 Amazon DocumentDB。

步驟 2. JRE 或 JDK 安裝

根據您的 BI 應用程式，您可能需要確保電腦上已安裝 64 位元 JRE 或 JDK 安裝版本 8 或更新版本。您可以在這裡下載 Java SE 執行期環境 8。

步驟 3。下載 DocumentDB JDBC 驅動程式

從這裡下載 DocumentDB JDBC 驅動程式。驅動程式封裝為單一 JAR 檔案 （ 例如 documentdb-jdbc-1.0.0-all.jar)。

步驟 4. 使用 SSH 通道連線至 Amazon DocumentDB

Amazon DocumentDB （ 具有 MongoDB 相容性 ） 叢集會部署在 Amazon Virtual Private Cloud (Amazon VPC) 中。Amazon EC2 執行個體或其他部署在相同 Amazon VPC 中的 AWS 服務可以直接存取它們。此外，相同 AWS 區域或其他區域中不同 VPCs 中的 EC2a 執行個體或其他 AWS 服務可以透過 VPC 對等互連存取 Amazon DocumentDB。

您可以使用 SSH 通道 （ 也稱為連接埠轉送 ） 從叢集的 VPC 外部存取 Amazon DocumentDB 資源。對於大多數使用者在與 DocumentDB 叢集位於相同 VPC 的 VM 上執行其應用程式的情況。

若要建立 SSH 通道，您需要在與 Amazon DocumentDB 叢集相同的 Amazon VPC 中執行的 Amazon EC2 執行個體。 Amazon DocumentDB 您可以使用相同 VPC 中現有的 EC2 執行個體做為您的叢集，或建立一個叢集。您可以在本機電腦上`sample-cluster.node.us-east-1.docdb.amazonaws.com`執行下列命令，以設定 Amazon DocumentDB 叢集的 SSH 通道。

```
ssh -i "ec2Access.pem" -L 27017:sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017 ubuntu@ec2-34-229-221-164.compute-1.amazonaws.com -
N
```

-L 旗標用於轉送本機連接埠。這是連線到 VPC 外部用戶端上執行的任何 BI 工具的先決條件。執行上述步驟後，您就可以繼續進行所選 BI 工具的後續步驟。

如需 SSH 通道 的詳細資訊，請參閱使用 SSH 通道連線至 Amazon DocumentDB 的文件。

從 Tableau 桌面連線至 Amazon DocumentDB

主題

- 新增 Amazon DocumentDB JDBC 驅動程式
- 使用 Tableau - SSH 通道連線至 Amazon DocumentDB

新增 Amazon DocumentDB JDBC 驅動程式

若要從 Tableau 桌面連線至 Amazon DocumentDB，您必須下載並安裝 Amazon DocumentDB JDBC 驅動程式和 DocumentDB Tableau 連接器。

1. 從 Amazon DocumentDB JDBC 驅動程式儲存庫下載 Amazon DocumentDB JDBC 驅動程式 JAR 檔案，並根據作業系統將其複製到下列其中一個目錄：
   - Windows - `C:\Program Files\Tableau\Drivers`
   - MacOS - `~/Library/Tableau/Drivers`
2. 從 Tableau Exchange 網站下載 DocumentDB Tableau 連接器 (TACO 檔案），並將其複製到 My Tableau 儲存庫/Connectors 目錄。
   - Windows - `C:\Users\[user]\Documents\My Tableau Repository\Connectors`
   - MacOS - `/Users/[user]/Documents/My Tableau Repository/Connectors`

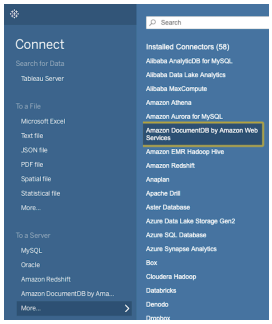
如需詳細資訊，請參閱 Tableau 文件。

> **Note**
>
> 如果您使用的是較新的 CA 憑證，請務必將 JDBC 驅動程式升級至 v1.4.5 （可從此 AWS GitHub 儲存庫取得）。

使用 Tableau - SSH 通道連線至 Amazon DocumentDB

若要從 DocumentDB 叢集 VPC 外部的用戶端機器連線至 Tableau，您必須先設定 SSH 通道，才能執行下列步驟：

1. 啟動 Tableau 桌面應用程式。
2. 導覽至連線 > 至伺服器 > 更多。
3. 在已安裝的連接器下，選擇 Amazon Web Services 的 Amazon DocumentDB。

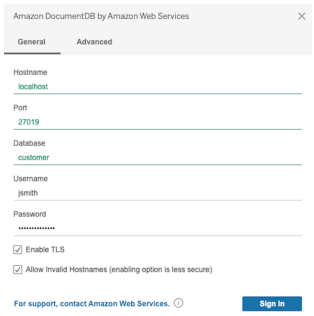


使用 Tableau 連線至 Amazon DocumentDB - 外部 SSH 通道

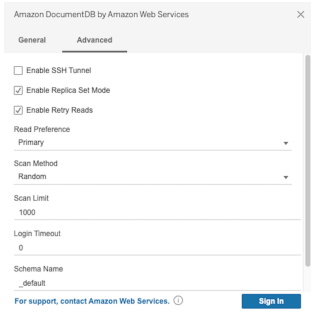
1. 輸入必要的連線參數 主機名稱、連接埠、資料庫、使用者名稱和密碼。下列範例中的連線參數等同於 JDBC 連線字串 ：

   ```
   jdbc:documentdb://localhost:27019/test?
   tls=true&tlsAllowInvalidHostnames=true&scanMethod=random&scanLimit=1000&logir
   ```

   在屬性集合中分別傳遞使用者名稱和密碼參數。如需連線字串參數的詳細資訊，請參閱 Amazon DocumentDB JDBC 驅動程式 github 文件。



2. （選用）更多進階選項可在進階索引標籤中找到。

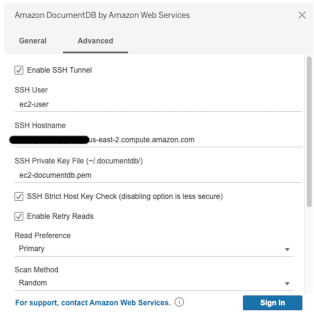


3. 選擇登入。

使用 Tableau 連線至 Amazon DocumentDB - 內部 SSH 通道

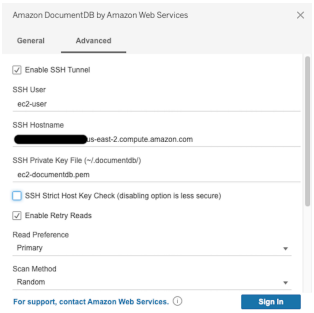
> **Note**
>
> 如果您不想使用終端機設定 SSH 通道，您可以使用 Tableau GUI 來指定 EC2 執行個體詳細資訊，JDBC 驅動程式本質上會用它來建立 SSH 通道。

1. 在進階索引標籤上，選擇啟用 SSH 通道選項以檢閱其他屬性。



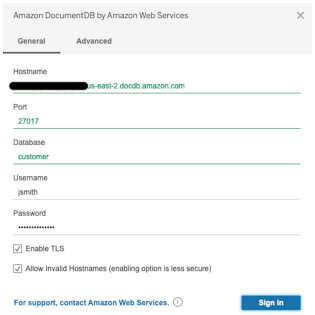
2. 輸入 SSH 使用者、SSH 主機名稱和 SSH 私密金鑰檔案。
3. （選用）您可以停用 SSH 嚴格主機金鑰檢查選項，該選項會略過已知主機檔案的主機金鑰檢查。

> **Note**
>
> 停用此選項較不安全，因為它可能會導致man-in-the-middle攻擊。



4. 輸入必要的參數：主機名稱、連接埠、資料庫、使用者名稱和密碼。

> **Note**
>
> 使用內部 SSH 通道選項時，請務必使用 DocumentDB 叢集端點，而非 localhost。



5. 選擇 Sign In (登入)。

從 DbVisualizer 連線至 Amazon DocumentDB

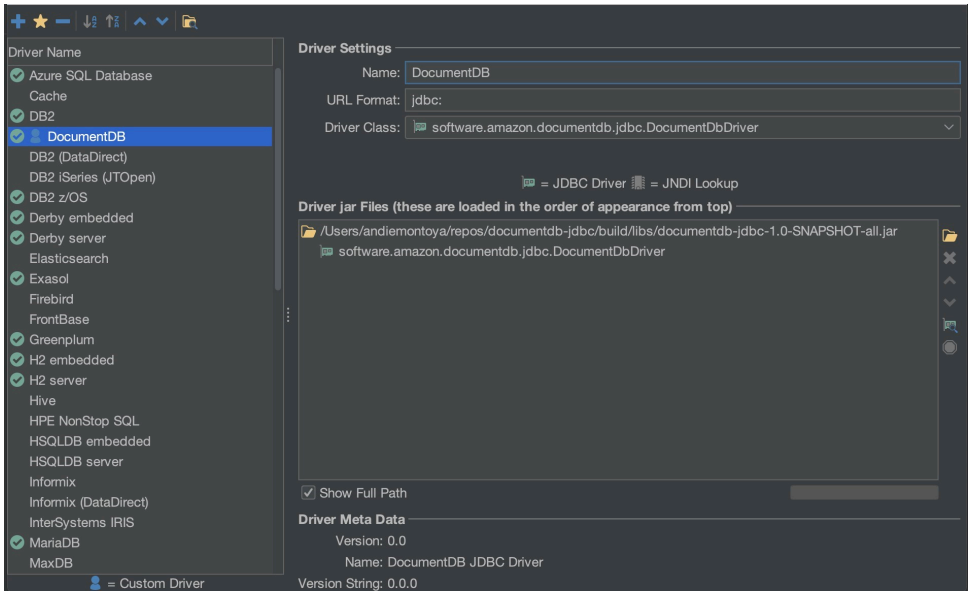
主題

- 新增 Amazon DocumentDB JDBC 驅動程式
- 使用 DbVisualizer 連線至 Amazon DocumentDB

新增 Amazon DocumentDB JDBC 驅動程式

若要從 DbVisualizer 連線至 Amazon DocumentDB，您必須先匯入 Amazon DocumentDB JDBC 驅動程式

1. 啟動 DbVisualizer 應用程式並導覽至選單路徑：工具 > 驅動程式管理員...
2. 選擇 + （或在功能表中，選取驅動程式 > 建立驅動程式)。
3. 將 Name (名稱) 設定為 DocumentDB。
4. 將 URL 格式設定為 `jdbc:documentdb://<host>[:port]/<database>[?option=value[&option=value[...]]]`
5. 選擇資料夾按鈕，然後選擇 Amazon DocumentDB JDBC 驅動程式 JAR 檔案，然後選擇開啟按鈕。
6. 確認驅動程式類別欄位設定為 `software.amazon.documentdb.jdbc.DocumentDbDriver`。DocumentDB 的 Driver Manager 設定應如下所示。



7. 關閉對話。Amazon DocumentDB JDBC 驅動程式將設定並可供使用。

使用 DbVisualizer 連線至 Amazon DocumentDB

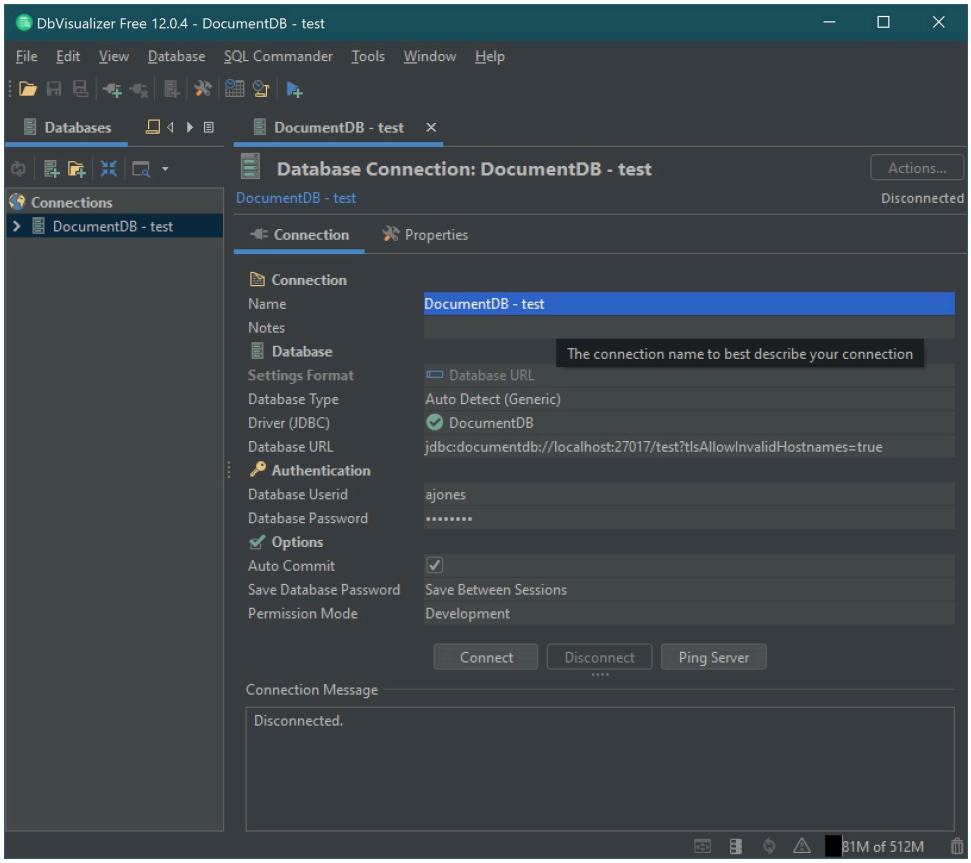
使用 DbVisualizer 連線至 Amazon DocumentDB

1. 如果您是從 Amazon DocumentDB 叢集的 VPC 外部連線，請確定您已設定 SSH 通道。
2. 從最上層功能表中選擇資料庫 > 建立資料庫連線。

3. 輸入名稱欄位的描述性名稱。
4. 將驅動程式 (JDBC) 設定為您在上一節中建立的 DocumentDB 驅動程式。
5. 將資料庫 URL 設定為 JDBC 連線字串。

   例如：`jdbc:documentdb://localhost:27017/database?tlsAllowInvalidHostnames=true`
6. 將 Database Userid 設定為 Amazon DocumentDB 使用者 ID。
7. 將資料庫密碼設定為使用者 ID 的對應密碼。

   您的資料庫連線對話方塊看起來應該如下所示：



8. 選擇連線。

JDBC 自動結構描述產生

Amazon DocumentDB 是文件資料庫，因此沒有資料表和結構描述的概念。不過，Tableau 等 BI 工具預期其連線的資料庫會顯示結構描述。具體而言，當 JDBC 驅動程式連線需要取得資料庫中集合的結構描述時，它會輪詢資料庫中的所有集合。驅動程式會判斷該集合的結構描述快取版本是否已存在。如果快取版本不存在，它會取樣文件的集合，並根據下列行為建立結構描述。

主題

- 結構描述產生限制
- 掃描方法選項
- Amazon DocumentDB 資料類型
- 映射純量文件欄位
- 物件和陣列資料類型處理

結構描述產生限制

DocumentDB JDBC 驅動程式對識別符的長度施加 128 個字元的限制。結構描述產生器可能會截斷產生的識別符（資料表名稱和資料欄名稱）的長度，以確保它們符合該限制。

掃描方法選項

您可以使用連線字串或資料來源選項來修改抽樣行為。

- scanMethod=<option>
  - random -（預設）- 以隨機順序傳回範例文件。
  - idForward - 依 ID 順序傳回範例文件。
  - idReverse - 以 ID 的相反順序傳回範例文件。
  - all - 取樣集合中的所有文件。
- scanLimit=<n> - 要取樣的文件數量。其值必須為正整數。預設值為 1000。如果 scanMethod 設為全部，則會忽略此選項。

Amazon DocumentDB 資料類型

Amazon DocumentDB 伺服器支援多種 MongoDB 資料類型。以下列出支援的資料類型及其相關聯的 JDBC 資料類型。

MongoDB 資料類型	DocumentDB 支援	JDBC 資料類型
二進位資料	是	VARBINARY
Boolean	是	BOOLEAN
Double	是	DOUBLE

MongoDB 資料類型	DocumentDB 支援	JDBC 資料類型
32 位元整數	是	INTEGER
64 位元整數	是	BIGINT
String	是	VARCHAR
ObjectId	是	VARCHAR
日期	是	TIMESTAMP
Null	是	VARCHAR
常規表達式	是	VARCHAR
時間戳記	是	VARCHAR
MinKey	是	VARCHAR
MaxKey	是	VARCHAR
物件	是	虛擬資料表
陣列	是	虛擬資料表
Decimal128	否	DECIMAL
JavaScript	否	VARCHAR
JavaScript（含範圍）	否	VARCHAR
未定義	否	VARCHAR
Symbol	否	VARCHAR
DBPointer (4.0+)	否	VARCHAR

映射純量文件欄位

從集合掃描文件範例時，JDBC 驅動程式會建立一或多個結構描述，以代表集合中的範例。一般而言，文件中的純量欄位會映射至資料表結構描述中的資料欄。例如，在名為 團隊的集合和單一文件中`{ "_id" : "112233", "name" : "Alastair", "age": 25 }`，這會映射到結構描述：

資料表名稱	資料行名稱	資料類型	金鑰
團隊	團隊 ID	VARCHAR	PK
團隊	name	VARCHAR	
團隊	age	INTEGER	

資料類型衝突提升

掃描抽樣文件時，欄位的資料類型可能因文件而不一致。在此情況下，JDBC 驅動程式會將 JDBC 資料類型提升為通用資料類型，以符合抽樣文件中的所有資料類型。

例如：

```
{
"_id" : "112233",
"name" : "Alastair", "age" : 25
}

{
"_id" : "112244",
"name" : "Benjamin",
"age" : "32"
}
```

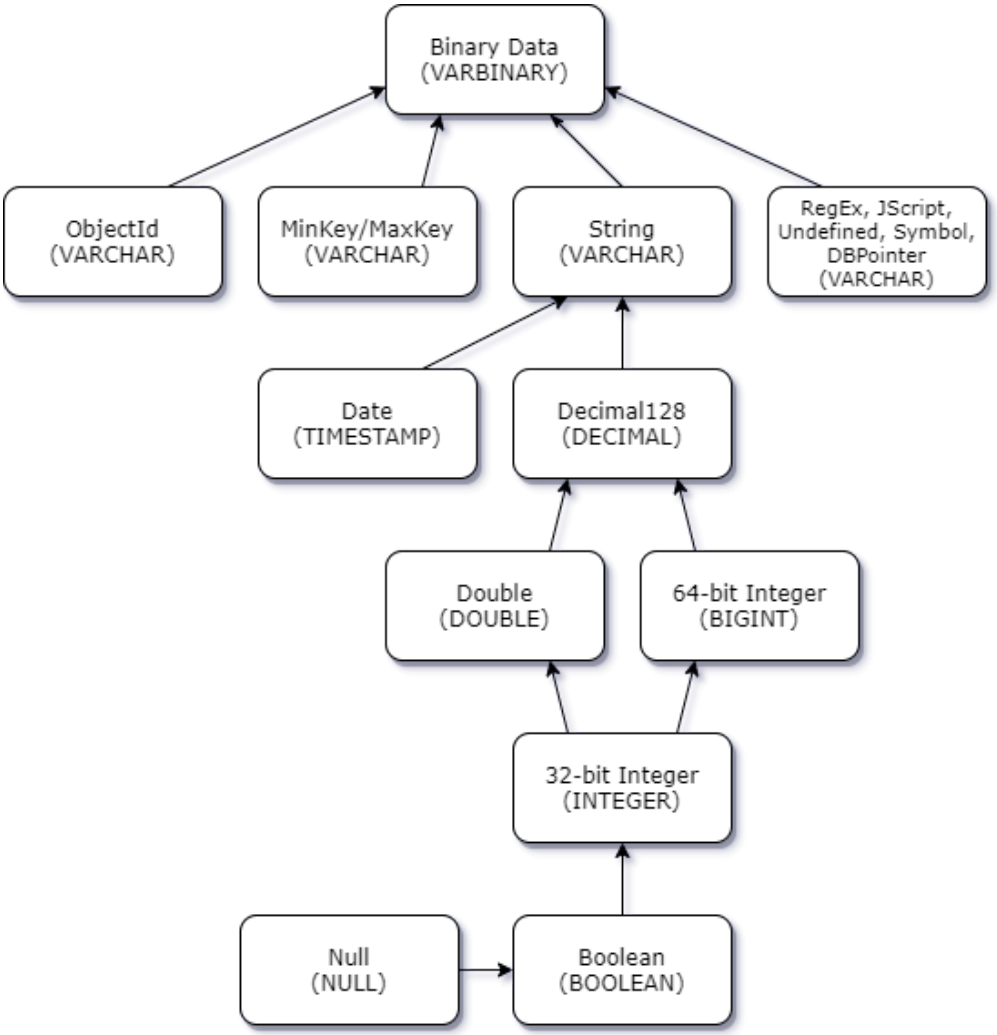
存留期欄位在第一個文件中為 32 位元整數類型，但在第二個文件中為字串。在這裡，JDBC 驅動程式會將 JDBC 資料類型提升為 VARCHAR，以便在遇到時處理任一資料類型。

資料表名稱	資料行名稱	資料類型	金鑰
團隊	團隊 ID	VARCHAR	PK

資料表名稱	資料行名稱	資料類型	金鑰
團隊	name	VARCHAR	
團隊	age	VARCHAR	

純量衝突提升

下圖顯示純量-純量資料類型衝突的解決方式。



Scalar-complex 類型衝突提升

與純量-純量類型衝突一樣，不同文件中的相同欄位在複雜（陣列和物件）和純量（整數、布林值等）之間可能會有衝突的資料類型。這些欄位的所有衝突都會解析（提升）為 VARCHAR。在此情況下，陣列和物件資料會以 JSON 表示形式傳回。

內嵌陣列 - 字串欄位衝突範例：

```
{
   "_id":"112233",
   "name":"George Jackson",
   "subscriptions":[
      "Vogue",
      "People",
      "USA Today"
   ]
}
{
   "_id":"112244",
   "name":"Joan Starr",
   "subscriptions":1
}
```

上述範例映射至 customer2 資料表的結構描述：

資料表名稱	資料行名稱	資料類型	金鑰
customer2	customer2 ID	VARCHAR	PK
customer2	name	VARCHAR	
customer2	訂閱	VARCHAR	

和 customer1_subscriptions 虛擬資料表：

資料表名稱	資料行名稱	資料類型	金鑰
customer1_subscriptions	customer1 ID	VARCHAR	PK/FK
customer1_subscriptions	subscriptions_index_lvl0	BIGINT	PK
customer1_subscriptions	value	VARCHAR	

資料表名稱	資料行名稱	資料類型	金鑰
customer_address	城市	VARCHAR	
customer_address	region	VARCHAR	
customer_address	國家/地區	VARCHAR	
customer_address	code	VARCHAR	

物件和陣列資料類型處理

到目前為止，我們只描述了純量資料類型的映射方式。物件和陣列資料類型會（目前）映射至虛擬資料表。JDBC 驅動程式會建立虛擬資料表，以代表文件中的物件或陣列欄位。映射虛擬資料表的名稱會串連原始集合的名稱，後面接著以底線字元 ("_") 分隔的欄位名稱。

基礎資料表的主索引鍵 ("_id") 會採用新虛擬資料表中的新名稱，並以外部索引鍵的形式提供給相關聯的基礎資料表。

對於內嵌陣列類型欄位，會產生索引欄，以在陣列的每個層級將索引表示為陣列。

內嵌物件欄位範例

對於文件中的物件欄位，JDBC 驅動程式會建立虛擬資料表的映射。

```
{
   "Collection: customer",
   "_id":"112233",
   "name":"George Jackson",
   "address":{
      "address1":"123 Avenue Way",
      "address2":"Apt. 5",
      "city":"Hollywood",
      "region":"California",
      "country":"USA",
      "code":"90210"
   }
}
```

上述範例映射到客戶資料表的結構描述：

資料表名稱	資料行名稱	資料類型	金鑰
customer	客戶 ID	VARCHAR	PK
customer	name	VARCHAR	

和 customer_address 虛擬資料表：

資料表名稱	資料行名稱	資料類型	金鑰
customer_address	客戶 ID	VARCHAR	PK/FK
customer_address	地址 1	VARCHAR	
customer_address	地址 2	VARCHAR	
customer_address	城市	VARCHAR	
customer_address	region	VARCHAR	
customer_address	國家/地區	VARCHAR	
customer_address	code	VARCHAR	

內嵌陣列欄位範例

對於文件中的陣列欄位，JDBC 驅動程式也會建立虛擬資料表的映射。

```
{
   "Collection: customer1",
   "_id":"112233",
   "name":"George Jackson",
   "subscriptions":[
      "Vogue",
      "People",
      "USA Today"
   ]
}
```

上述範例映射至 customer1 資料表的結構描述：

資料表名稱	資料行名稱	資料類型	金鑰
customer1	customer1 ID	VARCHAR	PK
customer1	name	VARCHAR	

和 customer1_subscriptions 虛擬資料表：

資料表名稱	資料行名稱	資料類型	金鑰
customer1_subscriptions	customer1 ID	VARCHAR	PK/FK
customer1_subscriptions	subscriptions_index_lvl0	BIGINT	PK
customer1_subscriptions	value	VARCHAR	
customer_address	城市	VARCHAR	
customer_address	region	VARCHAR	
customer_address	國家/地區	VARCHAR	
customer_address	code	VARCHAR	

SQL 支援和限制

Amazon DocumentDB JDBC 驅動程式是唯讀驅動程式，支援 SQL-92 子集和一些常見的擴充功能。如需詳細資訊，請參閱 SQL 限制文件和 JDBC 限制文件。

疑難排解

如果您在使用 Amazon DocumentDB JDBC 驅動程式時遇到問題，請參閱故障診斷指南。

使用 Amazon DocumentDB ODBC 驅動程式進行連線

Amazon DocumentDB 的 ODBC 驅動程式為開發人員提供 SQL 相關界面，並啟用 Power BI Desktop 和 Microsoft Excel 等 BI 工具的連線。

如需更多詳細資訊，請參閱 GitHub 上的 Amazon DocumentDB ODBC 驅動程式文件。

主題

- 開始使用
- 在 Windows 中設定 Amazon DocumentDB ODBC 驅動程式
- 從 Microsoft Excel 連線至 Amazon DocumentDB
- 從 Microsoft Power BI 桌面連線至 Amazon DocumentDB
- 自動產生結構描述
- SQL 支援和限制
- 疑難排解

開始使用

步驟 1. 建立 Amazon DocumentDB 叢集

如果您還沒有 Amazon DocumentDB 叢集，有多種方法可以開始使用。

> **Note**
>
> Amazon DocumentDB 是僅限 Virtual Private Cloud (VPC) 的服務。如果您是從叢集 VPC 外部的本機機器連線，則需要建立與 Amazon EC2 執行個體的 SSH 連線。如需詳細資訊，請參閱使用 Amazon EC2 連線。如需 SSH 通道的詳細資訊，請參閱使用 SSH 通道連線至 Amazon DocumentDB。

步驟 2. JRE 或 JDK 安裝

根據您的 BI 應用程式，您可能需要確保電腦上已安裝 64 位元 JRE 或 JDK 安裝版本 8 或更新版本。您可以在這裡下載 Java SE 執行期環境 8。

步驟 3. 下載 Amazon DocumentDB ODBC 驅動程式

在此處下載 Amazon DocumentDB ODBC 驅動程式。選擇適當的安裝程式 （ 例如 documentdb-odbc-1.0.0.msi)。請遵循 安裝指南。

步驟 4. 使用 SSH 通道連線至 Amazon DocumentDB

Amazon DocumentDB 叢集部署在 Amazon Virtual Private Cloud (Amazon VPC) 中。Amazon EC2 執行個體或其他部署在相同 Amazon VPC 中的 AWS 服務可以直接存取它們。此外，Amazon DocumentDB 可以透過 VPCs 中的 Amazon EC2 執行個體 AWS 區域 或其他 AWS 服務存取。

不過，假設您的使用案例要求您（或您的應用程式）從叢集的 VPC 外部存取 Amazon DocumentDB 資源。大多數使用者不會在與 Amazon DocumentDB 叢集位於相同 VPC 的 VM 上執行應用程式，就會發生這種情況。從 VPC 外部連線時，您可以使用 SSH 通道（也稱為連接埠轉送）來存取 Amazon DocumentDB 資源。

若要建立 SSH 通道，您需要在與 Amazon DocumentDB 叢集相同的 Amazon VPC 中執行的 Amazon EC2 執行個體。 Amazon DocumentDB 您可以使用相同 VPC 中現有的 EC2 執行個體做為您的叢集，或建立一個叢集。您可以在本機電腦上`sample-cluster.node.us-east-1.docdb.amazonaws.com`執行下列命令，以設定 Amazon DocumentDB 叢集的 SSH 通道：

```
ssh -i "ec2Access.pem" -L 27017:sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017 ubuntu@ec2-34-229-221-164.compute-1.amazonaws.com -
N
```

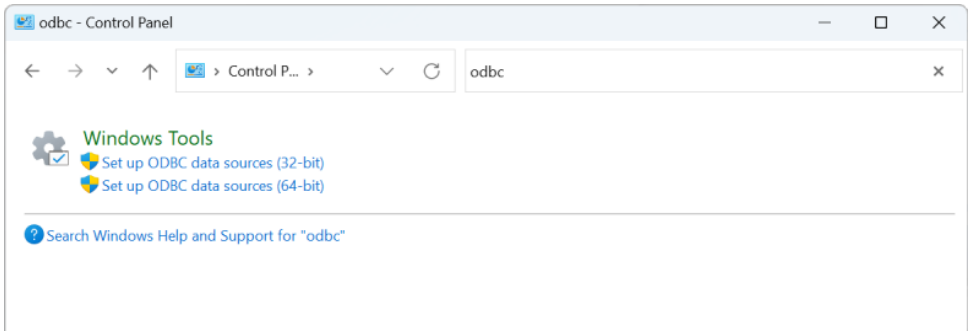
`-L` 旗標用於轉送本機連接埠。這是連線到 VPC 外部用戶端上執行的任何 BI 工具的先決條件。執行上述步驟後，您就可以繼續進行所選 BI 工具的後續步驟。

如需 SSH 通道的詳細資訊，請參閱使用 SSH 通道連線至 Amazon DocumentDB 的文件。

在 Windows 中設定 Amazon DocumentDB ODBC 驅動程式

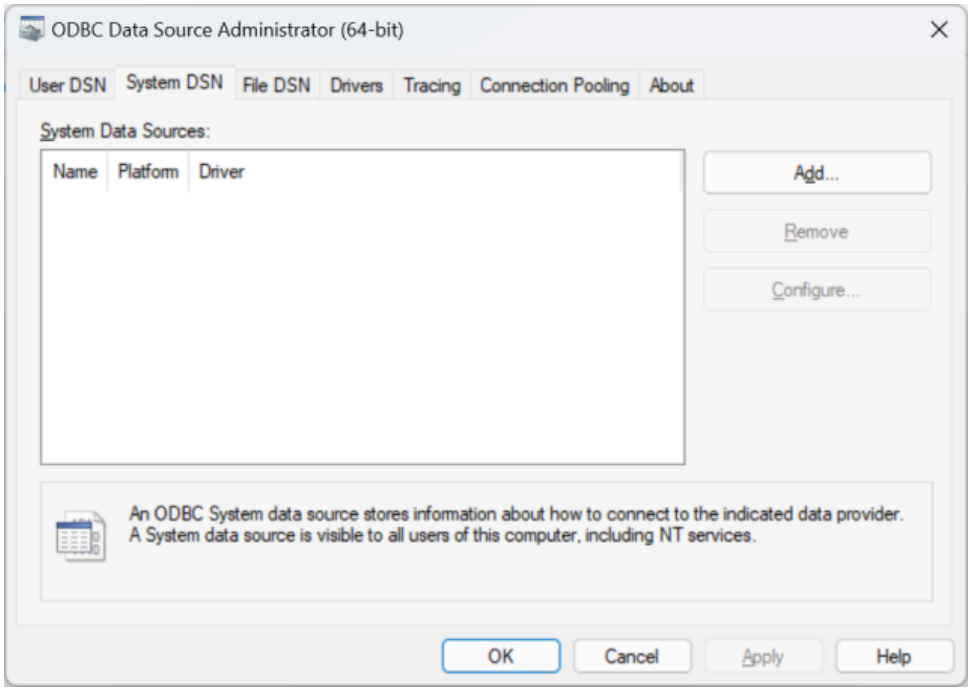
使用下列程序在 Windows 中設定 Amazon DocumentDB ODBC 驅動程式：

1. 在 Windows 中開啟控制面板並搜尋 ODBC（或在功能表中選取 Windows 工具 > ODBC 資料來源 (32 位元） 或 ODBC 資料來源 (64 位元））：

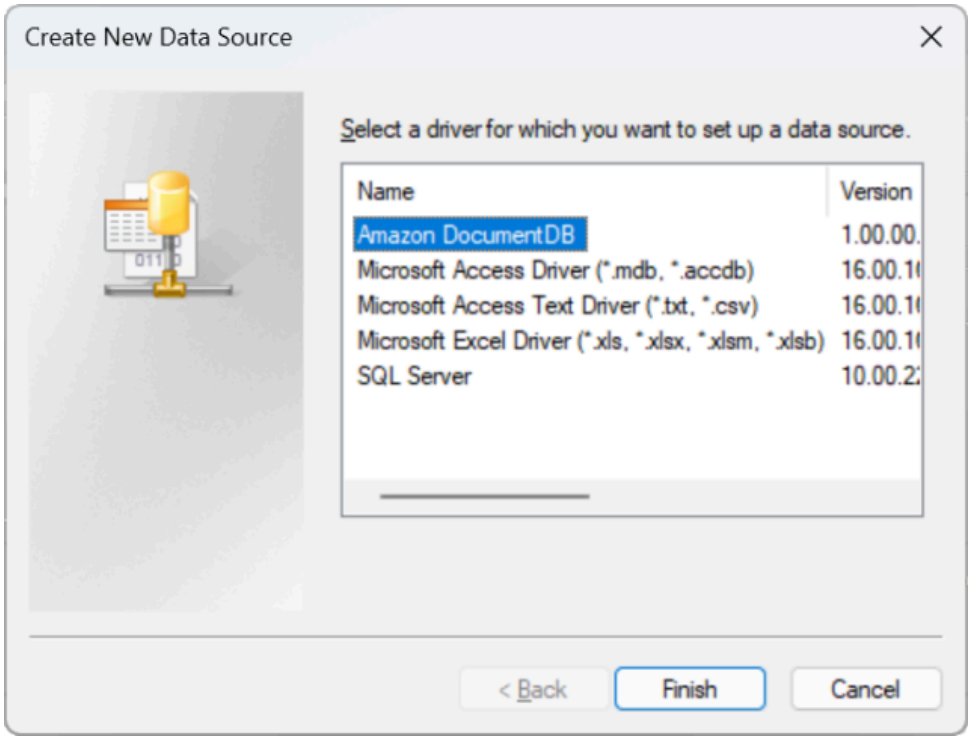


2. 選取適當的 ODBC 驅動程式資料來源管理員：如果已安裝 32 位元版本，請選擇該版本，否則請選擇 64 位元版本。

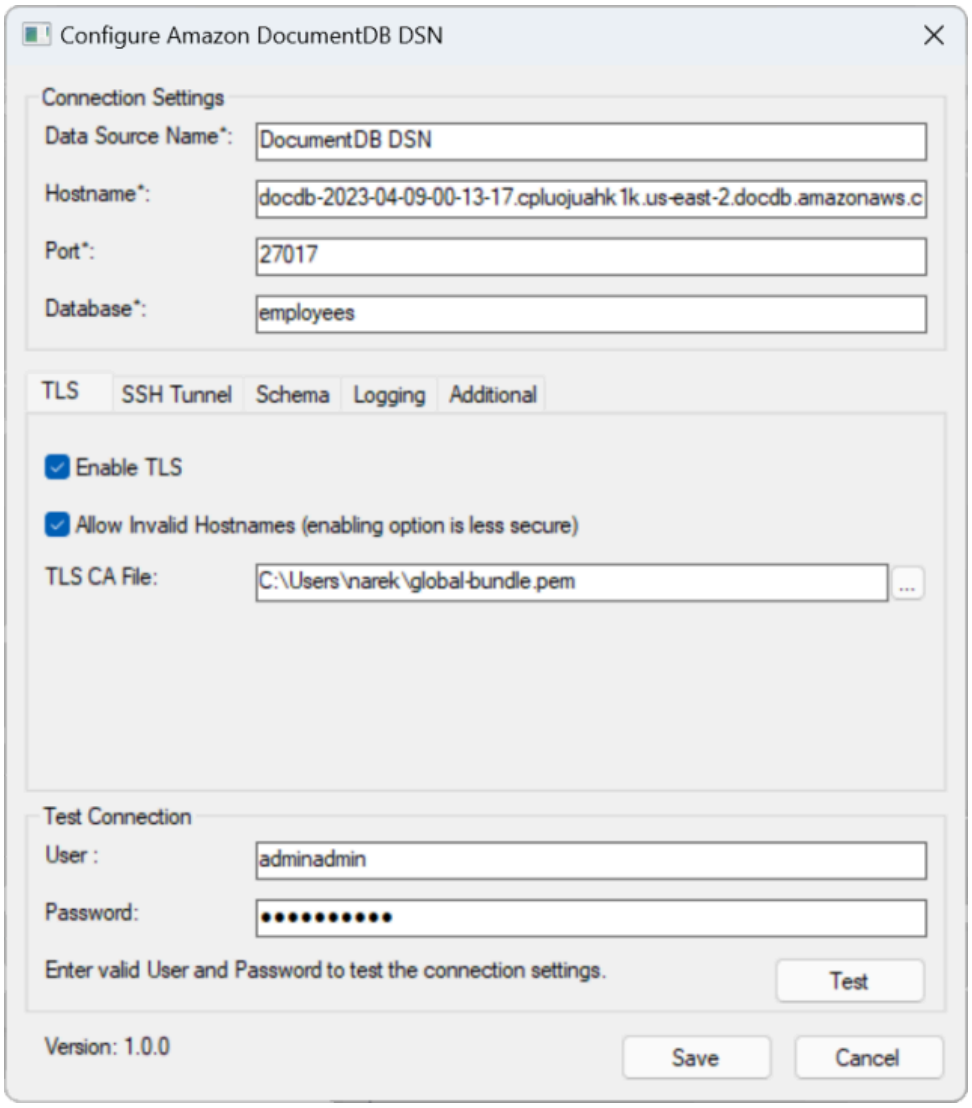
3. 選取 Sytem DSN 索引標籤，然後選擇新增...以新增新的 DSN：



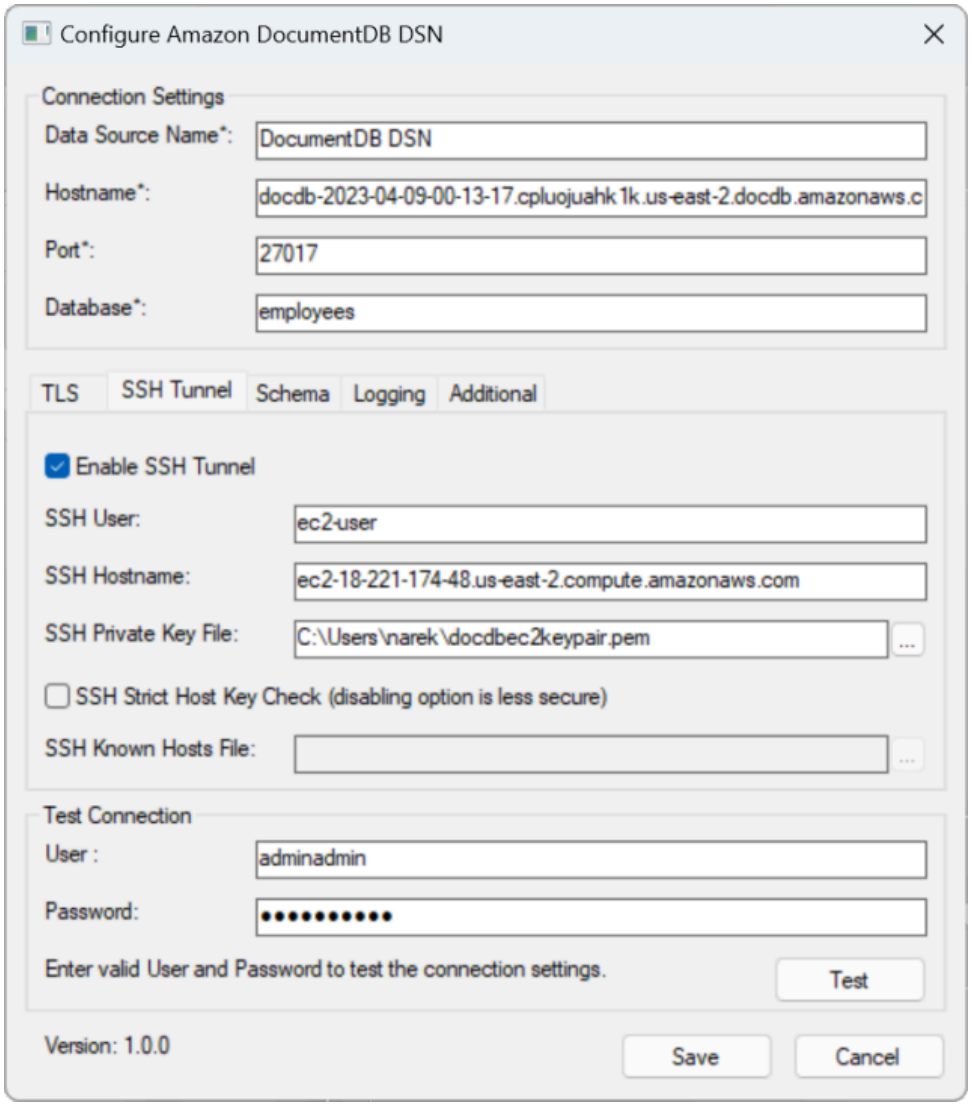
4. 從資料來源驅動程式清單中選擇 Amazon DocumentDB：



5. 在設定 Amazon DocumentDB DSN 對話方塊中，完成連線設定、TLS 索引標籤和測試連線欄位，然後選擇儲存：



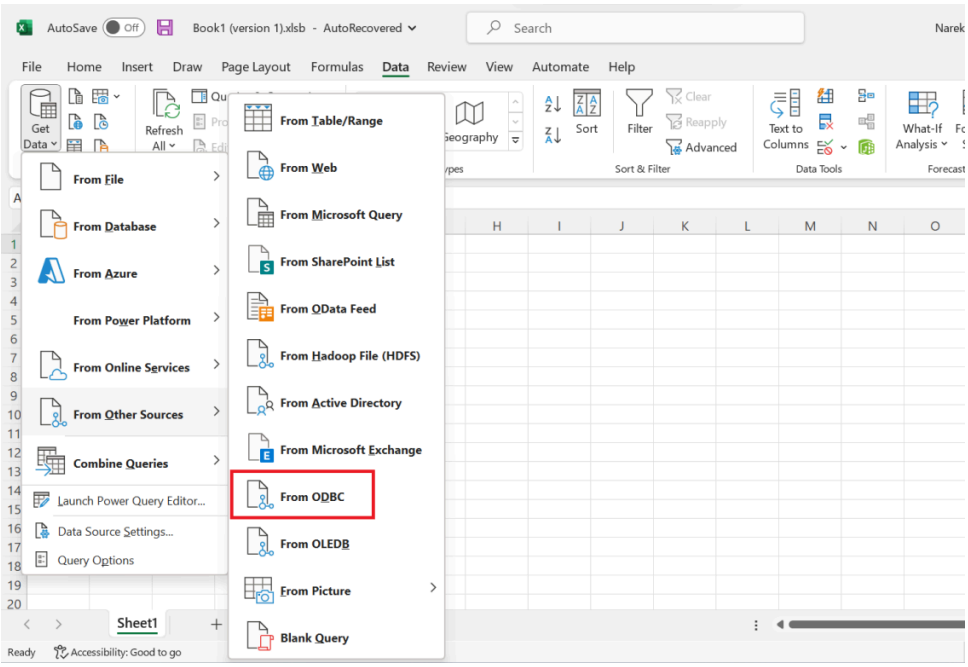
6. 請確定您準確填寫 Windows 表單，因為連線詳細資訊會因您選擇的 SSH 通道方法與 EC2 執行個體而有所不同。請參閱此處的 SSH 通道方法。如需每個屬性的詳細資訊，請參閱連線字串語法和選項。



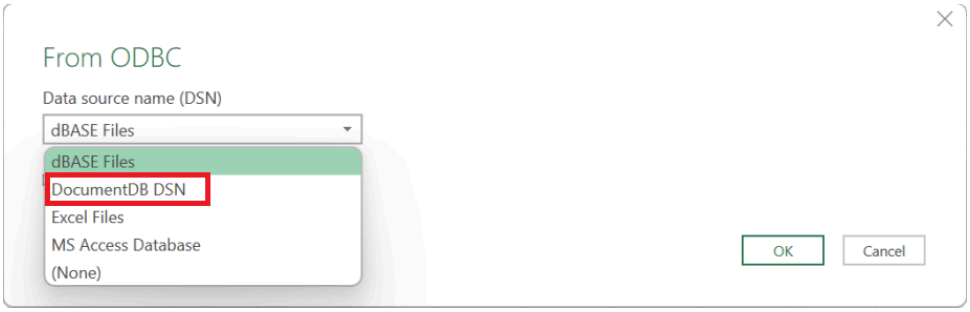
如需在 Windows 上設定 Amazon DocumentDB ODBC 驅動程式的詳細資訊，請參閱此處。

從 Microsoft Excel 連線至 Amazon DocumentDB

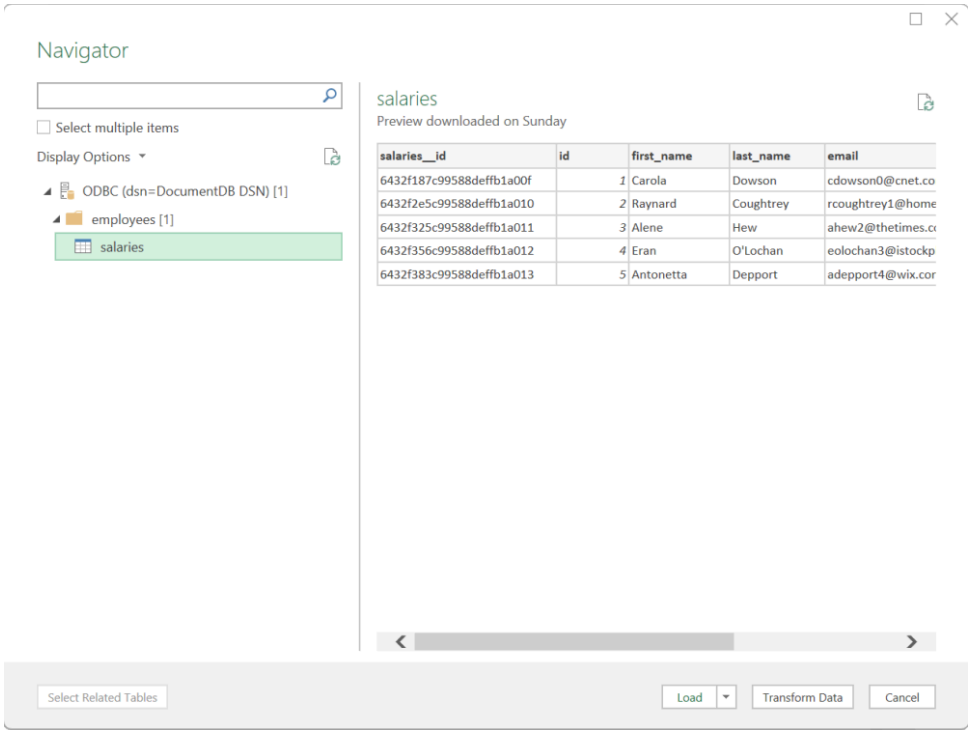
1. 確定已正確安裝和設定 Amazon DocumentDB 驅動程式。如需詳細資訊，請參閱在 Windows 中設定 Amazon DocumentDB ODBC 驅動程式。
2. 啟動 Microsoft Excel。
3. 導覽至資料 > 從其他來源取得資料 >。
4. 從 ODBC 選擇：



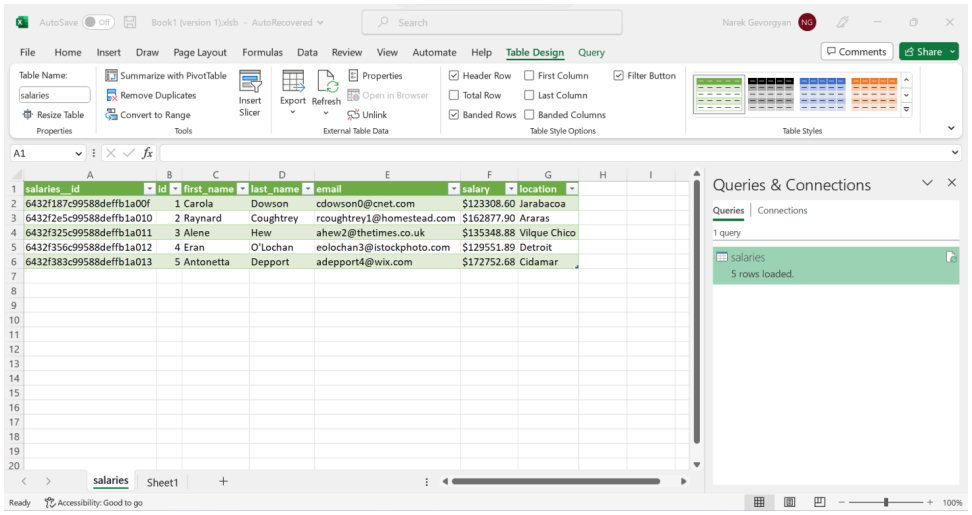
5. 從與 Amazon DocumentDB 相關聯的資料來源名稱 (DSN) 下拉式選單中選取資料來源：



6. 選擇您要從中將資料載入 Excel 的集合：



7. 將資料載入 Excel：



從 Microsoft Power BI 桌面連線至 Amazon DocumentDB

主題

- 先決條件
- 新增 Microsoft Power BI 桌面自訂連接器
- 使用 Amazon DocumentDB 自訂連接器進行連線
- 設定 Microsoft Power BI Gateway

先決條件

開始之前，請確定已正確安裝 Amazon DocumentDB ODBC 驅動程式。

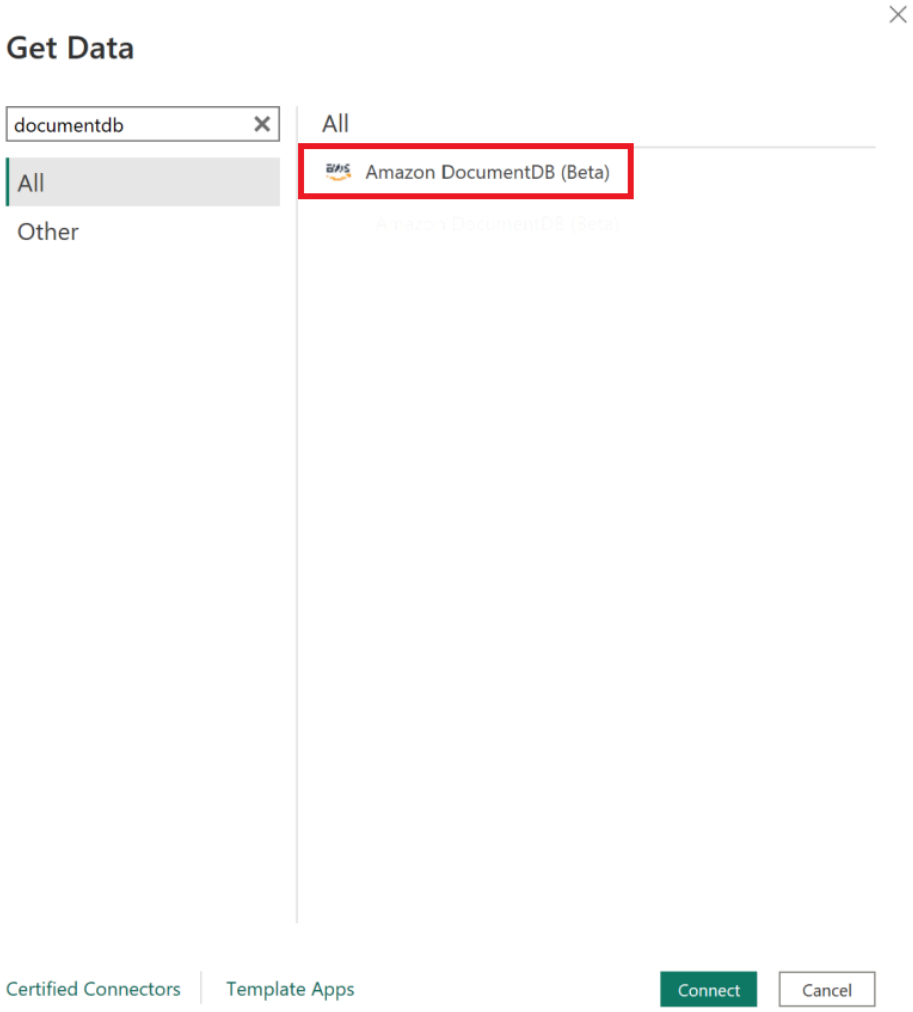
新增 Microsoft Power BI 桌面自訂連接器

將AmazonDocumentDBConnector.mez檔案複製到 <User>\Documents\Power BI Desktop \Custom Connectors\ 資料夾（如果使用 OneDrive<User>\OneDrive\Documents\Power BI Desktop\Custom Connectors，則複製到)。這將允許 Power BI 存取自訂連接器。您可以在此處取得 Power BI Desktop 的連接器。重新啟動 Power BI 桌面，以確保已載入連接器。

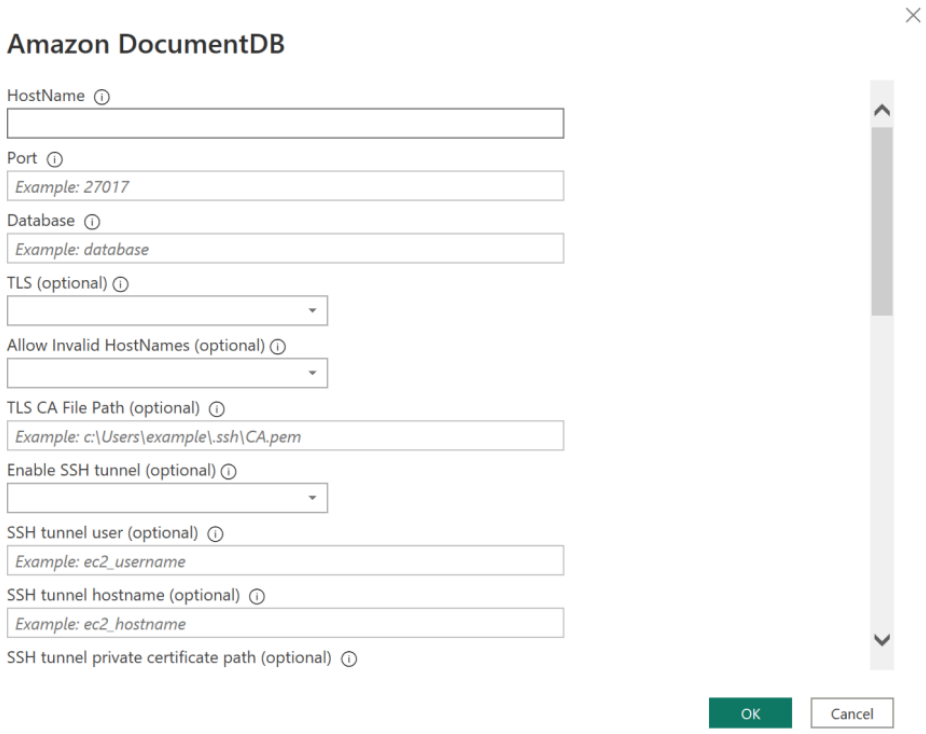
> **Note**
>
> 自訂連接器僅支援 Amazon DocumentDB 使用者名稱和密碼進行身分驗證。

使用 Amazon DocumentDB 自訂連接器進行連線

1. 從取得資料中選取 Amazon DocumentDB (Beta)，然後選擇連線。如果您收到使用第三方服務的警告，請選擇繼續。



2. 輸入所有必要資訊以連線至 Amazon DocumentDB 叢集，然後選擇確定：



> **Note**
>
> 視 ODBC 驅動程式的資料來源名稱 (DSN) 組態而定，如果您已在 DSN 設定中提供必要資訊，則可能不會顯示 SSH 連線詳細資訊畫面。

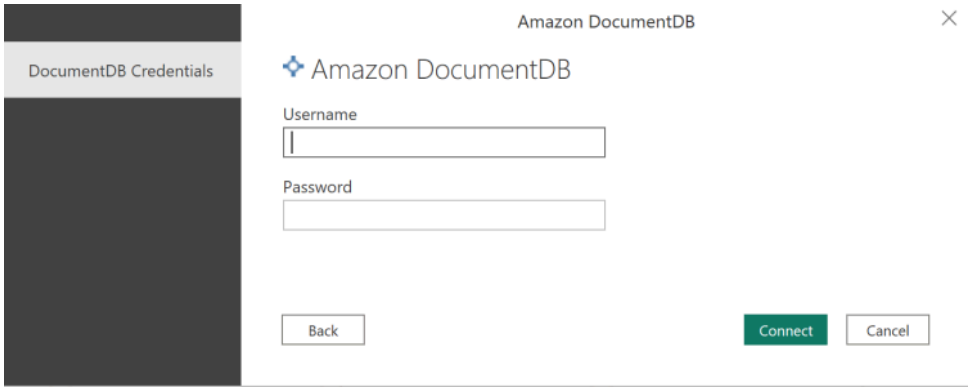
3. 選擇資料連線模式：

   - 匯入 - 載入所有資料並將資訊存放在磁碟上。必須重新整理並重新載入資料，以顯示資料更新。
   - 直接查詢 - 不會載入資料，但會對資料進行即時查詢。這表示不需要重新整理和重新載入資料，即可顯示資料更新。

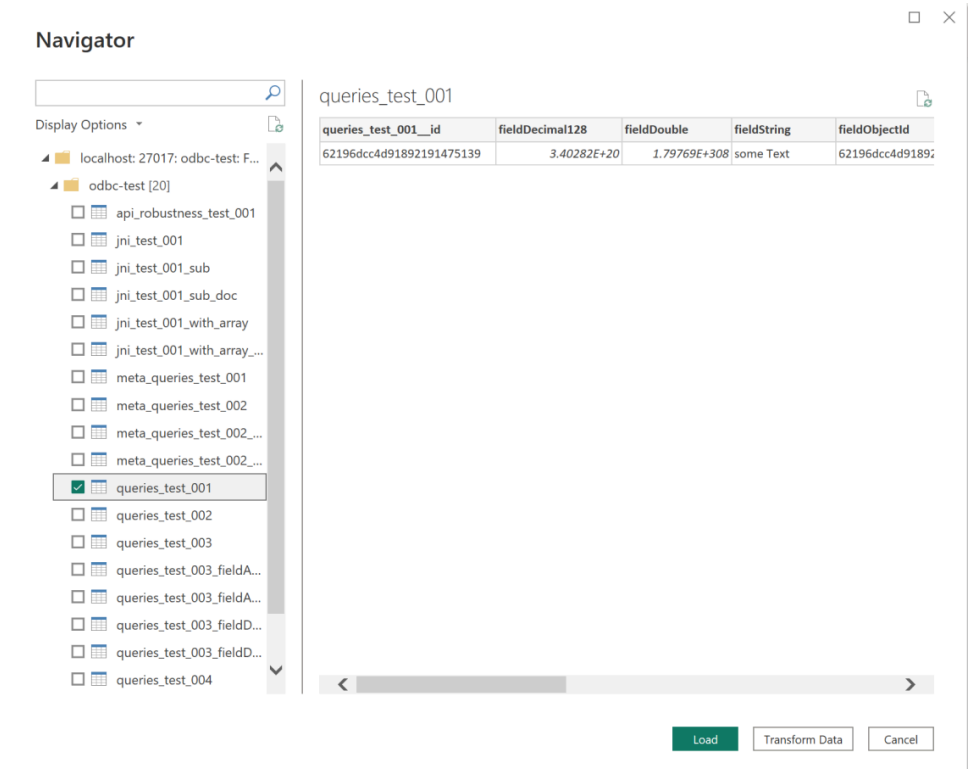


> **Note**
>
> 如果您使用的是非常大型的資料集，匯入所有資料可能需要更長的時間。

4. 如果這是第一次連接到此資料來源，請選取身分驗證類型，並在出現提示時輸入您的登入資料。然後選擇連線：



5. 在導覽器對話方塊中，選取您想要的資料庫資料表，然後選擇載入以載入資料，或選擇轉換資料以繼續轉換資料。



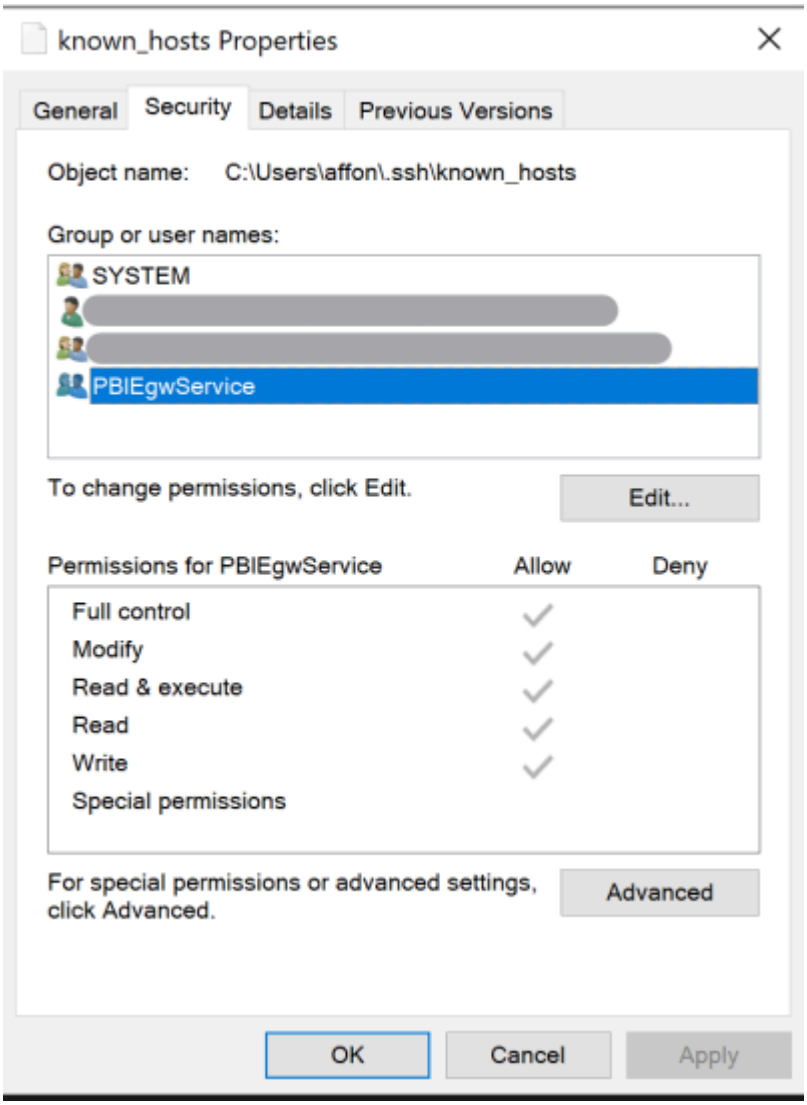
> **Note**
>
> 您的資料來源設定會在您連線後儲存。若要修改它們，請選取轉換資料 > 資料來源設定。

設定 Microsoft Power BI Gateway

先決條件：

- 確定自訂連接器可與 Power BI Gateway 搭配使用。
- 確定 ODBC DSN 是在安裝 Power BI Gateway 之機器的系統索引標籤中的 ODBC 資料來源中建立。

如果您使用的是內部 SSH 通道功能，檔案`known_hosts`必須位於 Power BI 服務帳戶可存取它的位置。



> **Note**
>
> 這也適用於您可能需要建立 Amazon DocumentDB 叢集連線的任何檔案 （例如憑證授權機構 (CA) 憑證檔案 (pem 檔案）。

自動產生結構描述

ODBC 驅動程式透過 JNI (Java 原生界面） 使用 Amazon DocumentDB JDBC 驅動程式 - 讓自動結構描述產生功能在 JDBC 驅動程式中類似運作。如需自動產生結構描述的詳細資訊，請參閱 JDBC 自動結構描述產生。如需 ODBC 驅動程式架構的詳細資訊，請參閱 GitHub 上的 Amazon DocumentDB ODBC 驅動程式架構。

SQL 支援和限制

Amazon DocumentDB ODBC 驅動程式是唯讀驅動程式，支援 SQL-92 的子集和一些常見的擴充功能。如需詳細資訊，請參閱 ODBC 支援和限制文件。

疑難排解

如果您在使用 Amazon DocumentDB ODBC 驅動程式時遇到問題，請參閱故障診斷指南。

使用 Amazon DocumentDB 進行程式設計

服務支援 JSON 結構描述驗證。

主題

- Amazon DocumentDB Java 程式設計指南
- 使用 JSON 結構描述驗證

Amazon DocumentDB Java 程式設計指南

此完整指南提供使用 MongoDB Java 驅動程式使用 Amazon DocumentDB 的詳細演練，涵蓋資料庫操作和管理的基本層面。

主題

- 簡介
- 先決條件
- 本指南的資料模型
- 使用 MongoDB Java 驅動程式連線至 Amazon DocumentDB MongoDB
- 使用 Java 在 Amazon DocumentDB 中執行 CRUD 操作
- 使用 Java 的 Amazon DocumentDB 中的索引管理
- 使用 Amazon DocumentDB 和 Java 進行事件驅動程式設計

簡介

本指南從連線開始，說明如何使用 MongoDB Java 驅動程式建立與 DocumentDB 叢集的安全連線。它詳細說明了連線字串元件、SSL/TLS 實作和各種連線選項，包括 IAM 身分驗證和連線集區，以及強大的錯誤處理策略。

在 CRUD（建立、讀取、更新、刪除）操作區段中，本指南徹底涵蓋文件操作，示範如何使用單一和大量操作建立、讀取、更新和刪除文件。它說明了篩選條件、查詢和各種操作選項的使用方式，同時強調錯誤處理和實作重試邏輯的最佳實務，以提高可靠性。本指南也廣泛涵蓋索引管理，詳細說明不同索引類型的建立和維護，包括單一欄位、複合、稀疏和文字索引。它說明如何透過適當的索引選擇和`explain()`函數的使用來最佳化查詢效能，以分析查詢執行計畫。

最後一節著重於使用 Amazon DocumentDB 變更串流的事件驅動程式設計，示範如何在 Java 應用程式中實作即時資料變更監控。它涵蓋變更串流游標的實作、持續操作的恢復權杖處理，以及歷史資料處理的時間型操作。本指南提供實用的程式碼範例和最佳實務，讓您在使用 Amazon DocumentDB 建置強大的 Java 應用程式時，成為寶貴的資源。

先決條件

開始前，請確定您具有下列項目：

- AWS 帳戶 具有已設定 Amazon DocumentDB 叢集的 。如需叢集設定，請參閱 AWS 資料庫部落格上的使用 Amazon EC2 的 Amazon DocumentDB 入門Amazon EC2。
- 已安裝 Java 開發套件 (JDK)（本指南將使用 Amazon Corretto 21)。
- Maven 用於相依性管理。

本指南的資料模型

本指南中的所有範例程式碼都假設連線到具有「Restaurants」集合的「ProgGuideData」測試資料庫。本指南中的所有範例程式碼都適用於餐廳清單系統，以下是此系統中文件的外觀範例：

```
{
    "_id": "ab6ad8f119b5bca3efa2c7ae",
    "restaurantId": "REST-CRT9BL",
    "name": "Thai Curry Palace",
    "description": "Amazing Restaurant, must visit",
    "cuisine": "Thai",
    "address": {
        "street": "914 Park Street",
        "city": "Bryan",
        "state": "AL",
        "zipCode": "96865",
        "location": {
            "type": "Point",
            "coordinates": [-25.4619, 8.389]
        }
```

```
    },
    "contact": {
        "phone": "(669) 915-9056 x6657"
    },
    "rating": {
        "average": 3.4,
        "totalReviews": 275
    },
    "priceRange": "$",
    "menu": [{
        "category": "Appetizers",
        "items": [{
            "name": "Buffalo Chicken Wings",
            "price": 13.42
        }]
    }],
    "features": [
        "Private Dining"
    ],
    "isActive": false“ michelin”: {“
        star”: 3,
        “ranking_years”: 4
    }
}
```

所有顯示 CRUD、索引管理和事件驅動程式設計的程式碼範例都會假設您有一個MongoClient物件 dbClient、一個MongoDatabase物件 connectionDB和一個MongoCollection物件 collection。

> **Note**
>
> 本指南中的所有程式碼範例都已使用 MongoDB Java 驅動程式 5.3.0 版進行測試。

使用 MongoDB Java 驅動程式連線至 Amazon DocumentDB MongoDB

本節提供使用 Java 驅動程式連線至 Amazon DocumentDB step-by-step指南。這將協助您開始將 DocumentDB 整合到您的 Java 應用程式。

主題

- 步驟 1：設定您的 專案

- 步驟 2：建立連線字串
- 步驟 3：撰寫連線程式碼
- 步驟 4：處理連線例外狀況
- 步驟 5：執行程式碼
- 連線最佳實務

步驟 1：設定您的 專案

1. 使用 Maven 建立 Java 專案：

```
mvn archetype:generate -DgroupId=com.docdb.guide -DartifactId=my-docdb-project -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

2. 新增 MongoDB Java 驅動程式做為「pom.xml」檔案中專案的相依性：

```
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-sync</artifactId>
    <version>5.3.0</version>
</dependency>
```

步驟 2：建立連線字串

Amazon DocumentDB 連線字串對於在您的應用程式與 DocumentDB 叢集之間建立連線至關重要。此字串會封裝重要資訊，例如叢集端點、連接埠、身分驗證詳細資訊和各種連線選項。若要建置 DocumentDB 連線字串，您通常會以基本格式開頭：

```
"mongodb://username:password@cluster-endpoint:port/?[connection options]"
```

您需要將 "username" 和 "password" 取代為實際的登入資料。您可以在 以及透過 找到叢集的端點 AWS 管理主控台 和連接埠號碼 AWS CLI。請參閱 尋找叢集的端點 尋找叢集的叢集端點。DocumentDB 的預設連接埠為 27017。

連線字串範例

- 使用傳輸中加密與 DocumentDB 建立連線，並確保讀取請求移至僅供讀取複本並寫入主要複本：

```
"mongodb://username:password@cluster-endpoint:27017/?tls=true&
```

```
   tlsCAFile=global-bundle.pem&
   readPreference=secondaryPreferred&
   retryWrites=false"
```

- 使用 IAM 身分驗證與 DocumentDB 建立連線：

```
"mongodb://cluster-endpoint:27017/?tls=true&
   tlsCAFile=global-bundle.pem&
   readPreference=secondaryPreferred&
   retryWrites=false&
   authSource=%24external&
   authMechanism=MONGODB-AWS"
```

連線字串可用的不同選項如下：

- TLS 憑證
- 從僅供讀取複本讀取
- 撰寫疑慮和日誌
- RetryWrites
- IAM 身分驗證
- 連線集區
- 連線逾時參數

TLS 憑證

`tls=true|false` — 此選項啟用或停用 Transport Layer Security (TLS)。根據預設，Amazon DocumentDB 叢集上會啟用傳輸中加密，因此，除非在叢集層級停用 TLS，否則此選項的值應為 `true`。

使用 TLS 時，程式碼需要在建立 DocumentDB 叢集的連線時提供 SSL 憑證。下載與叢集進行安全連線所需的憑證：`global-bundle.pem`。有兩種方式可以使用 `global-bundle.pem` 檔案。

- 選項 1 — 從 `global-bundle.pem` 檔案擷取所有憑證，並使用 Java 的 keytool 將憑證存放在`.jks`檔案，之後可在程式碼中使用。請參閱 中的 Java 索引標籤，使用已啟用的 TLS 連線以取得顯示如何執行此操作的指令碼。
- 選項 2 — 在程式碼中動態新增 `global-bundle.pem` 檔案、建置記憶體內金鑰存放區，並使用 `SSLContext`提供憑證做為建立連線的一部分。

從僅供讀取複本讀取

replicaSet=rs0&readPreference=secondaryPreferred — 指定這兩個選項會將所有讀取請求路由至僅供讀取複本，並將請求寫入主要執行個體。在連線字串`replicaSet=rs0`中使用 可讓 MongoDB 驅動程式維護叢集拓撲的自動更新檢視，允許應用程式在新增或移除執行個體時維持目前節點組態的可見性。未提供這些選項或指定 會將`readPreference=primary`所有讀取和寫入傳送至主要執行個體。如需 的更多選項`readPreference`，請參閱 讀取偏好設定選項。

撰寫疑慮和日誌

寫入考量會決定從資料庫請求寫入操作的確認層級。MongoDB 驅動程式提供調整寫入考量和日誌檔案的選項。Amazon DocumentDB 不預期您設定寫入考量和日誌，並忽略針對 `w`和 `j`(`writeConcern` 和) 傳送的值`journal`。DocumentDB 一律使用 `writeConcern`： `majority`和 `journal`： 寫入資料，`true`因此在將確認傳送給用戶端之前，會在大多數節點上持久記錄寫入。

RetryWrites

retryWrites=false — DocumentDB 不支援可重試的寫入，因此此屬性應一律設定為 `false`。

IAM 身分驗證

authSource=%24external 和 **authMechanism=MONGODB-AWS** — 這兩個參數用於使用 進行身分驗證 AWS Identity and Access Management。IAM 身分驗證目前僅適用於執行個體型叢集 5.0 版。如需詳細資訊，請參閱使用 IAM 身分進行身分驗證。

連線集區

這些選項可用於連線集區：

- **maxPoolSize** — 設定可在集區中建立的連線數目上限。當所有連線都在使用中且有新的請求進來時，它會等待連線變成可用。MongoDB Java 驅動程式的預設值為 100。
- **minPoolSize** — 表示應隨時在集區中維護的連線數下限。MongoDB Java 驅動程式的預設值為 0。
- **maxIdleTimeMS** — 決定在關閉和移除連線之前，連線可以在集區中保持閒置的時間長度。MongoDB Java 驅動程式的預設值為 100 毫秒。
- **waitQueueTimeoutMS** — 設定當集區達到其大小上限時，執行緒應等待連線可用的時間長度。如果連線在此期間內無法使用，則會擲回例外狀況。MongoDB Java 驅動程式的預設值為 120，000 毫秒 (2 分鐘) 。

連線逾時參數

逾時是一種機制，可限制操作或連線嘗試在被視為失敗之前可能需要的時間。下列逾時參數可用於防止無限期等待和管理資源配置：

- **connectTimeoutMS** — 設定驅動程式等待建立叢集連線的時間長度。預設值為 10，000 毫秒 (10 秒）。
- **socketTimeoutMS** — 指定驅動程式等待伺服器回應進行非寫入操作的時間長度。預設值為 0，（無逾時或無限）。
- **serverSelectionTimeoutMS** — 指定驅動程式在叢集中尋找可用伺服器的等待時間。此設定的預設值為 30 秒，足以在容錯移轉期間選擇新的主要執行個體。

步驟 3：撰寫連線程式碼

下列程式碼範例示範如何與 Amazon DocumentDB 建立 TLS 連線：

- 它會建立 Java 和 KeyStore SSLContext> 物件。
- 它也會透過將MongoClientSettings物件傳遞至物件來建立ConnectionString物件。若要進行 TLS 連線，您必須使用 MongoClientSettings 物件來繫結 connectionstring和 sslcontext。
- 使用 MongoClients取得MongoClient物件。

```
public static MongoClient makeDbConnection(String dbName, String DbUserName, String
 DbPassword,
    String DbClusterEndPoint, String keyStorePass) throws Exception {
    MongoClient connectedClient;
    String connectionOptions = "?
replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false";
    String connectionUrl = "mongodb://" + DbUserName + ":" + DbPassword + "@" +
 DbClusterEndPoint + ":27017/" +
        dbName + connectionOptions;

    try {
        KeyStore trustStore = KeyStore.getInstance(KeyStore.getDefaultType());
        try (FileInputStream fis = new FileInputStream("src/main/resources/certs/
truststore.jks")) {
            trustStore.load(fis, keyStorePass.toCharArray());
            TrustManagerFactory tmf =
 TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
```

```
            tmf.init(trustStore);

            SSLContext sslContext = SSLContext.getInstance("TLSv1.2");
            sslContext.init(null, tmf.getTrustManagers(), new SecureRandom());
            ConnectionString connectionString = new ConnectionString(connectionUrl);
            MongoClientSettings settings = MongoClientSettings.builder()
                .applyConnectionString(connectionString)
                .applyToSslSettings(builder - > {
                    builder.enabled(true);
                    builder.context(sslContext);
                })
                .build();
            connectedClient = MongoClients.create(settings);
        }
        return connectedClient;
    } catch (MongoException e5) {
        throw new RuntimeException(e5);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

步驟 4：處理連線例外狀況

在 Java 應用程式中使用 DocumentDB 時，處理連線例外狀況對於維護強大且可靠的資料庫操作至關重要。適當地管理這些例外狀況不僅有助於快速診斷問題，還可確保應用程式可以正常地處理暫時網路中斷或伺服器無法使用，進而改善穩定性和使用者體驗。與建立連線相關的一些重要例外狀況包括：

- **MongoException** — 一般例外狀況，可以在更具體例外狀況未涵蓋的各種情況下發出。確定在所有其他特定例外狀況之後處理此例外狀況，因為這是一般擷取所有 MongoDB 例外狀況。
- **MongoTimeoutException** — 操作逾時時發出。例如，查詢不存在的叢集端點。
- **MongoSocketException** — 針對網路相關問題發行。例如，在 操作期間突然中斷網路連線。
- **MongoSecurityException** — 身分驗證失敗時發出。例如，使用不正確的登入資料進行連線。
- **MongoConfigurationException** — 用戶端組態發生錯誤時發出。例如，使用無效的連線字串。

步驟 5：執行程式碼

下列程式碼範例會建立 Amazon DocumentDB 連線並列印所有資料庫：

```
public static void TestConnection() {
```

```
    try (MongoClient mongoClient = makeDbConnection(DATABASE_NAME, DB_USER_NAME,
 DB_PASSWORD, DB_CLUSTER_ENDPOINT, KEYSTORE_PASSWORD)) {
        List < String > databases = mongoClient.listDatabaseNames().into(new ArrayList
 < > ());
        System.out.println("Databases: " + databases);
    } catch (MongoException e) {
        System.err.println("MongoDB error: " + e.getMessage());
        throw new RuntimeException(e);
    }
}
```

連線最佳實務

以下是使用 MongoDB Java 驅動程式連線至 Amazon DocumentDB 時應考慮的最佳實務：

- 當您不再需要用戶端釋出資源`MongoClient`時，請務必關閉您的 。
- 適當地處理例外狀況，並實作適當的錯誤記錄。
- 使用環境變數或 AWS Secrets Manager 來存放敏感資訊，例如使用者名稱和密碼。

使用 Java 在 Amazon DocumentDB 中執行 CRUD 操作

本節討論如何使用 MongoDB Java 驅動程式在 Amazon DocumentDB 中執行 CRUD （建立、讀取、更新、刪除）操作。

主題

- 在 DocumentDB 集合中建立和插入文件
- 從 DocumentDB 集合讀取和擷取資料
- 更新 DocumentDB 集合中的現有文件
- 從 DocumentDB 集合移除文件
- 使用重試邏輯處理錯誤

在 DocumentDB 集合中建立和插入文件

將文件插入 Amazon DocumentDB 可讓您將新資料新增至集合。執行插入有幾種方式，取決於您的需求和正在使用的資料量。將個別文件插入集合的最基本方法是 `insertOne()`。若要一次插入多個文件，您可以使用 `insertMany()`方法，這可讓您在單一操作中新增文件陣列。在 DocumentDB 集合中插入許多文件的另一種方法是 `bulkWrite()`。在本指南中，我們會討論在 DocumentDB 集合中建立文件的所有這些方法。

insertOne()

讓我們從檢查如何將個別文件插入 Amazon DocumentDBB 集合開始。使用 insertOne()方法完成插入單一文件。此方法採用 BsonDocument 進行插入，並傳回可用來取得新插入文件物件 ID 的InsertOneResult物件。以下範例程式碼顯示將一個餐廳文件插入集合：

```
Document article = new Document()
    .append("restaurantId", "REST-21G145")
    .append("name", "Future-proofed Intelligent Bronze Hat")
    .append("cuisine", "International")
    .append("rating", new Document()
        .append("average", 1.8)
        .append("totalReviews", 267))
    .append("features", Arrays.asList("Outdoor Seating", "Live Music"));

try {
    InsertOneResult result = collection.insertOne(article);
    System.out.println("Inserted document with the following id: " +
 result.getInsertedId());
} catch (MongoWriteException e) {
    // Handle duplicate key or other write errors
    System.err.println("Failed to insert document: " + e.getMessage());
    throw e;
} catch (MongoException e) {
    // Handle other MongoDB errors
    System.err.println("MongoDB error: " + e.getMessage());
    throw e;
}
```

使用 時insertOne()，請務必包含適當的錯誤處理。例如，在上述程式碼中，「restaurantId」具有唯一的索引，因此再次執行此程式碼將引發下列 MongoWriteException：

```
Failed to insert document: Write operation error on server
 docdbCluster.docdb.amazonaws.com:27017.
Write error: WriteError{code=11000, message='E11000 duplicate key error collection:
 Restaurants index: restaurantId_1', details={}}.
```

insertMany()

用於將許多文件插入集合的主要方法是 insertMany() 和 bulkWrite()。

insertMany() 方法是在單一操作中插入多個文件的最簡單方法。它接受文件清單並將其插入集合。當您插入一批彼此獨立且不需要任何特殊處理或混合操作的新文件時，此方法很理想。下列程式碼顯示從檔案讀取 JSON 文件並將其插入集合。insertMany() 函數會傳回 InsertManyResultInsertManyResult 物件，可用來取得所有插入文件IDs。

```
// Read JSON file content
String content = new String(Files.readAllBytes(Paths.get(jsonFileName)));
JSONArray jsonArray = new JSONArray(content);

// Convert JSON articles to Documents
List < Document > restaurants = new ArrayList < > ();
for (int i = 0; i < jsonArray.length(); i++) {
    JSONObject jsonObject = jsonArray.getJSONObject(i);
    Document doc = Document.parse(jsonObject.toString());
    restaurants.add(doc);
}
//insert documents in collection
InsertManyResult result = collection.insertMany(restaurants);

System.out.println("Count of inserted documents: " + result.getInsertedIds().size());
```

bulkWrite()

bulkWrite() 方法允許 在單一批次中執行多個寫入操作 （插入、更新、刪除）。當您需要在單一批次中執行不同類型的操作bulkWrite()時，可以使用 ，例如插入一些文件，同時更新其他文件。bulkWrite()支援兩種類型的批次寫入，有序和無序：

- 排序操作 — （預設） Amazon DocumentDB 會依序處理寫入操作，並在遇到的第一個錯誤時停止。這在操作順序很重要時很有用，例如稍後的操作取決於較早的操作。不過，排序操作通常比未排序的操作慢。使用有序操作時，您必須解決批次在第一次錯誤時停止的情況，這可能會導致某些操作未處理。
- 未排序的操作 — 允許 Amazon DocumentDB 將插入處理為資料庫中的單一執行。如果一個文件發生錯誤，操作會繼續其餘文件。當您插入大量資料時，這特別有用，可以容忍一些失敗，例如在資料遷移或大量匯入期間，某些文件可能因重複的金鑰而失敗。使用未排序的操作時，您必須解決部分成功案例，其中某些操作成功，而其他操作則失敗。

使用 bulkWrite()方法時，需要一些基本類別。首先， WriteModel類別會做為所有寫入操作的基本類別，並搭配 InsertOneModel、DeleteOneModel、、 UpdateOneModel UpdateManyModel和 等特定實作DeleteManyModel來處理不同類型的操作。

BulkWriteOptions 類別是設定大量操作行為的必要項目，例如設定排序/未排序的執行或略過文件驗證。BulkWriteResult 類別提供執行結果的詳細資訊，包括插入、更新和刪除文件的計數。

對於錯誤處理， MongoBulkWriteException類別至關重要，因為它包含大量操作期間任何失敗的資訊，而 BulkWriteError類別提供個別操作失敗的特定詳細資訊。下列程式碼顯示插入文件清單，以及更新和刪除單一文件的範例，全部都在單一bulkWrite()方法呼叫的執行中。此程式碼也會示範如何使用 BulkWriteOptions和 BulkWriteResult，以及操作的適當錯誤處理bulkWrite()。

```
List < WriteModel < Document >> bulkOperations = new ArrayList < > ();
// get list of 10 documents representing 10 restaurants
List < Document > restaurantsToInsert = getSampleData();

for (Document doc: restaurantsToInsert) {
    bulkOperations.add(new InsertOneModel < > (doc));
}
// Update operation
bulkOperations.add(new UpdateOneModel < > (
    new Document("restaurantId", "REST-Y2E9H5"),
    new Document("", new Document("stats.likes", 20))
    .append("", new Document("rating.average", 4.5))));
// Delete operation
bulkOperations.add(new DeleteOneModel < > (new Document("restaurantId", "REST-
D2L431")));

// Perform bulkWrite operation
try {
    BulkWriteOptions options = new BulkWriteOptions()
        .ordered(false); // Allow unordered inserts

    BulkWriteResult result = collection.bulkWrite(bulkOperations, options);

    System.out.println("Inserted: " + result.getInsertedCount());
    System.out.println("Updated: " + result.getModifiedCount());
    System.out.println("Deleted: " + result.getDeletedCount());
} catch (MongoBulkWriteException e) {
    System.err.println("Bulk write error occurred: " + e.getMessage());
    // Log individual write errors
    for (BulkWriteError error: e.getWriteErrors()) {
        System.err.printf("Error at index %d: %s (Code: %d)%n", error.getIndex(),
 error.getMessage(),
            error.getCode());

        // Log the problematic document
```

```
        Document errorDoc = new Document(error.getDetails());
        if (errorDoc != null) {
            System.err.println("Problematic document: " + errorDoc);
        }
    }
} catch (Exception e) {
    System.err.println("Error during bulkWrite: " + e.getMessage());
}
```

可重試的寫入

與 MongoDB 不同，Amazon DocumentDB 不支援可重試的寫入。因此，您必須在其應用程式中實作自訂重試邏輯，特別是在處理網路問題或暫時服務無法使用時。實作良好的重試策略通常涉及增加重試嘗試之間的延遲，以及限制重試總數。如需使用錯誤處理建置重試邏輯的程式碼範例，請參閱使用重試邏輯處理錯誤下文。

從 DocumentDB 集合讀取和擷取資料

在 Amazon DocumentDB 中查詢文件圍繞著數個關鍵元件，可讓您精確擷取和操作資料。`find()` 方法是 MongoDB Java 驅動程式中的基本查詢 APIs。它允許複雜的資料擷取，具有許多篩選、排序和投影結果的選項。除了 `find()`方法之外， `FiltersFindIterable`也是提供 MongoDB Java 驅動程式中查詢操作之建置區塊的兩個其他基本元件。

`Filters` 類別是 MongoDB Java 驅動程式中的公用程式類別，可提供建構查詢篩選條件的流暢 API。此類別提供靜態原廠方法，可建立代表各種查詢條件的`Bson`物件執行個體。最常使用的方法包括`eq()`用於等式比較、`gt()`、`lt()`、`gte()`、 和 `lte()` 用於數值比較，`and()`以及`or()`用於合併多個條件，`in()`以及`nin()`用於陣列成員資格測試，以及`regex()`用於模式比對。類別設計為類型安全，並提供比原始文件型查詢更好的編譯時間檢查，使其成為在 Java 應用程式中建構 DocumentDB 查詢的首選方法。錯誤處理功能強大，因篩選條件建構無效而擲回明顯例外狀況。

`FindIterable` 是一種專門的界面，旨在處理 `find()`方法的結果。它提供一組豐富的方法來精簡和控制查詢執行，為方法鏈結提供流暢的 API。界面包含必要的查詢修改方法，例如`limit()`用於限制傳回的文件數量、`skip()`用於分頁、`sort()`用於排序結果、`projection()`用於選取特定欄位，以及`hint()`用於索引選取。中的批次、略過和限制操作`FindIterable`是必要的分頁和資料管理工具，可協助控制如何從資料庫擷取和處理文件。

批次處理 (`batchSize`) 控制 DocumentDB 在單一網路往返中傳回給用戶端的文件數量。當您設定批次大小時，DocumentDB 不會一次傳回所有相符的文件，而是在指定批次大小的群組中傳回它們。

略過可讓您偏移結果的起點，基本上是在開始傳回相符項目之前，告知 DocumentDB 略過指定數量的文件。例如， skip(20)會略過前 20 個相符的文件。這通常用於您想要擷取後續結果頁面的分頁案例。

限制會限制可從查詢傳回的文件總數。當您指定 時limit(n)，DocumentDB 會在傳回「n」文件後停止傳回文件，即使資料庫中有更多相符項目。

FindIterable 從 Amazon DocumentDB 擷取文件時， 同時支援迭代器和游標模式。使用FindIterable作為迭代器的好處是它允許延遲載入文件，並且僅在應用程式請求時擷取文件。使用迭代器的另一個好處是，您不需要負責維護與叢集的連線，因此不需要明確關閉連線。

FindIterable 也提供支援MongoCursor，允許在使用 Amazon DocumentDB 查詢時使用游標模式。 MongoCursor 是一種 MongoDB Java 驅動程式特定的實作，可讓您控制資料庫操作和資源管理。它實作 AutoCloseable界面，允許透過 try-with-resources 區塊進行明確的資源管理，這對於正確關閉資料庫連線和釋放伺服器資源至關重要。根據預設，游標會在 10 分鐘內逾時，而 DocumentDB 不會讓您選擇變更此逾時行為。使用批次資料時，請務必在游標逾時之前擷取下一批次的資料。使用時的一個關鍵考量MongoCursor是，它需要明確關閉以防止資源洩漏。

在本節中，會針對 find()、 Filters和 顯示數個範例FindIterable。

下列程式碼範例示範如何使用其「restaurantId」欄位find()來擷取單一文件：

```
Document filter = new Document("restaurantId", "REST-21G145");
Document result = collection.find(filter).first();
```

即使使用 Filters 可提供更好的編譯時間錯誤檢查，java 驅動程式也可讓您直接在 find()方法中指定Bson篩選條件。下列範例程式碼會將Bson文件傳遞給 find()：

```
result = collection.find(new Document("$and", Arrays.asList(
    new Document("rating.totalReviews", new Document("$gt", 1000)),
    new Document("priceRange", "$$"))))
```

下一個範例程式碼顯示將 Filters類別與 搭配使用的幾個範例find()：

```
FindIterable < Document > results;

// Exact match
results = collection.find(Filters.eq("name", "Thai Curry Palace"));

// Not equal
results = collection.find(Filters.ne("cuisine", "Thai"));
```

```
// find an element in an array
results = collection.find(Filters.in("features", Arrays.asList("Private Dining")));

// Greater than
results = collection.find(Filters.gt("rating.average", 3.5));

// Between (inclusive)
results = collection.find(Filters.and(
    Filters.gte("rating.totalReviews", 100),
    Filters.lte("rating.totalReviews", 200)));
// AND
results = collection.find(Filters.and(
    Filters.eq("cuisine", "Thai"),
    Filters.gt("rating.average", 4.5)));

// OR
results = collection.find(Filters.or(
    Filters.eq("cuisine", "Thai"),
    Filters.eq("cuisine", "American")));


// All document where the Field exists
results = collection.find(Filters.exists("michelin"));

// Regex
results = collection.find(Filters.regex("name", Pattern.compile("Curry",
 Pattern.CASE_INSENSITIVE)));

// Find all document where the array contain the list of value regardless of its order
results = collection.find(Filters.all("features", Arrays.asList("Private Dining",
 "Parking")));

// Array size
results = collection.find(Filters.size("features", 4));
```

下列範例示範如何在 FindIterable 物件batchSize()上鏈結 sort()、limit()、skip()和 的操作。提供這些操作的順序將影響查詢的效能。最佳實務是，這些操作的順序應為 sort()、skip()、 projection()limit()和 batchSize()。

```
FindIterable < Document > results = collection.find(Filters.gt("rating.totalReviews",
 1000))
    // Sorting
```

```
    .sort(Sorts.orderBy(
        Sorts.descending("address.city"),
        Sorts.ascending("cuisine")))
    // Field selection
    .projection(Projections.fields(
        Projections.include("name", "cuisine", "priceRange"),
        Projections.excludeId()))

    // Pagination
    .skip(20)
    .limit(10)
    .batchSize(2);
```

下列範例程式碼顯示在 上建立迭代器FindIterable。它使用 Java forEach 的建構來周遊結果集。

```
collection.find(Filters.eq("cuisine", "American")).forEach(doc ->
 System.out.println(doc.toJson()));
```

在最後一個find()程式碼範例中，它示範如何使用 cursor() 進行文件擷取。它會在嘗試區塊中建立游標，以確保在程式碼結束嘗試區塊時游標會關閉。

```
try (MongoCursor < Document > cursor = collection.find(Filters.eq("cuisine",
 "American"))
    .batchSize(25)
    .cursor()) {
    while (cursor.hasNext()) {
        Document doc = cursor.next();
        System.out.println(doc.toJson());
    }
} // Cursor automatically closed
```

更新 DocumentDB 集合中的現有文件

Amazon DocumentDB 提供靈活且強大的機制，用於修改現有文件，並在文件不存在時插入新文件。MongoDB Java 驅動程式提供數種更新方法：updateOne()適用於單一文件更新、updateMany()適用於多個文件更新，以及replaceOne()適用於完整文件取代。除了這三種方法之外，Updates、 UpdateOptions和 UpdateResult是為 MongoDB Java 驅動程式中的更新操作提供建置區塊的其他基本元件。

MongoDB Java 驅動程式中的 Updates類別是公用程式類別，可提供建立更新運算子的靜態原廠方法。它可做為主要建置器，以類型安全且可讀取的方式建構更新操作。set()、 unset()和 等基本方

法inc()允許直接修改文件。使用 Updates.combine()方法結合多個操作時，此類別的強大功能會變得明顯，這允許以原子方式執行多個更新操作，以確保資料一致性。

UpdateOptions 是 MongoDB Java 驅動程式中功能強大的組態類別，可為文件更新操作提供基本的自訂功能。此類別的兩個重要層面是為更新操作提供 upsert 和陣列篩選條件支援。透過 啟用的 upsert 功能upsert(true)允許在更新操作期間找不到相符的文件時建立新的文件。透過 arrayFilters()，更新操作可以精確更新符合特定條件的陣列元素。

UpdateResult MongoDB Java 驅動程式中的 提供回饋機制，詳細說明更新操作的結果。此類別會封裝三個關鍵指標：符合更新條件 () 的文件數量matchedCount、實際修改的文件數量 (modifiedCount)，以及任何備份文件的相關資訊 ()upsertedId。了解這些指標對於正確處理錯誤、驗證更新操作，以及維護應用程式中的資料一致性至關重要。

更新和取代單一文件

在 DocumentDB 中，可以使用 updateOne() 方法完成更新單一文件。此方法採用篩選參數 （通常由 Filters類別提供） 來識別要更新的文件、決定要更新哪些欄位的 Update 參數，以及用於設定更新不同選項的選用UpdateOptions參數。使用 updateOne()方法只會更新第一個符合選取條件的文件。下列範例程式碼會更新一個文件的單一欄位：

```
collection.updateOne(Filters.eq("restaurantId", "REST-Y2E9H5"),
    Updates.set("name", "Amazing Japanese sushi"));
```

若要更新一個文件中的多個欄位，請使用 updateOne()搭配 Update.combine()，如下列範例所示。此範例也會示範如何將項目新增至文件中的 陣列。

```
List<Bson> updates = new ArrayList<>();
// Basic field updates
updates.add(Updates.set("name", "Shanghai Best"));
// Array operations
updates.add(Updates.addEachToSet("features", Arrays.asList("Live Music")));
// Counter updates
updates.add(Updates.inc("rating.totalReviews", 10));
// Combine all updates
Bson combinedUpdates = Updates.combine(updates);
// Execute automic update with one call
collection.updateOne(Filters.eq("restaurantId","REST-1J83NH"), combinedUpdates);
```

下列程式碼範例示範如何更新資料庫中的文件。如果指定的文件不存在，操作會自動將其插入為新文件。此程式碼也會示範如何使用透過 UpdateResult 物件提供的指標。

```
Bson filter = Filters.eq("restaurantId", "REST-0Y9GL0");
Bson update = Updates.set("cuisine", "Indian");
// Upsert operation
UpdateOptions options = new UpdateOptions().upsert(true);
UpdateResult result = collection.updateOne(filter, update, options);

if (result.getUpsertedId() != null) {
    System.out.println("Inserted document with _id: " + result.getUpsertedId());
} else {
     System.out.println("Updated " + result.getModifiedCount() + " document(s)");
}
```

下列程式碼範例示範如何使用 replaceOne()方法將現有文件完全取代為新文件，而不是更新個別欄位。replaceOne() 方法會覆寫整個文件，只保留原始_id文件的欄位。如果多個文件符合篩選條件，則只會取代第一個遇到的文件。

```
Document newDocument = new Document()
                .append("restaurantId", "REST-0Y9GL0")
                .append("name", "Bhiryani Adda")
                .append("cuisine", "Indian")
                .append("rating", new Document()
                        .append("average", 4.8)
                        .append("totalReviews", 267))
                .append("features", Arrays.asList("Outdoor Seating", "Live Music"));

UpdateResult result = collection.replaceOne(
                    Filters.eq("restaurantId", "REST-0Y9GL0"),
                    newDocument);
System.out.printf("Modified %d document%n", result.getModifiedCount());
```

更新多個文件

有兩種方式可以同時更新集合中的多個文件。您可以使用 updateMany()方法，或使用 UpdateManyModel搭配 bulkWrite()方法。updateMany() 方法使用篩選參數來選取要更新的文件、用來識別要更新之欄位的Update參數，以及用來指定更新選項的選用UpdateOptions參數。

下列範例程式碼示範 updateMany()方法的使用方式：

```
Bson filter = Filters.and(
    Filters.in("features", Arrays.asList("Private Dining")),
    Filters.eq("cuisine", "Thai"));
```

```
UpdateResult result1 = collection.updateMany(filter, Updates.set("priceRange", "$$$"));
```

下列範例程式碼示範使用相同更新bulkWrite()的方法：

```
BulkWriteOptions options = new BulkWriteOptions().ordered(false);
List < WriteModel < Document >> updates = new ArrayList < > ();
Bson filter = Filters.and(
    Filters.in("features", Arrays.asList("Private Dining")),
    Filters.eq("cuisine", "Indian"));
Bson updateField = Updates.set("priceRange", "$$$");
updates.add(new UpdateManyModel < > (filter, updateField));
BulkWriteResult result = collection.bulkWrite(updates, options);
System.out.printf("Modified %d document%n", result.getModifiedCount());
```

從 DocumentDB 集合移除文件

MongoDB Java 驅動程式提供deleteOne()移除單一文件和移除符合特定條件的多個文件deleteMany()的功能。如同更新，刪除操作也可以與 bulkWrite()方法搭配使用。deleteOne() 和 都會deleteMany()傳回DeleteResult物件，提供操作結果的相關資訊，包括已刪除的文件計數。以下是使用 deleteMany()移除多個文件的範例：

```
Bson filter = Filters.and(
    Filters.eq("cuisine", "Thai"),
    Filters.lt("rating.totalReviews", 50));
DeleteResult result = collection.deleteMany(filter);
System.out.printf("Deleted %d document%n", result.getDeletedCount());
```

使用重試邏輯處理錯誤

Amazon DocumentDB 的強大錯誤處理策略應將錯誤分類實作為可重試（例如網路逾時、連線問題）和不可重試（例如身分驗證失敗、無效的查詢）。對於因應重試的錯誤而導致的操作失敗，它應該在每次重試以及最大重試嘗試之間實作時間延遲。CRUD 操作應該位於擷取 MongoException及其子類別的 try-catch 區塊中。此外，還應包括監控和記錄錯誤，以實現操作可見性。以下是示範如何實作重試錯誤處理的範例程式碼：

```
int MAX_RETRIES = 3;
int INITIAL_DELAY_MS = 1000;
int retryCount = 0;

while (true) {
```

```
    try {
        crud_operation(); //perform crud that will throw MongoException or one of its
 subclass
        break;
    } catch (MongoException e) {
        if (retryCount < MAX_RETRIES) {
            retryCount++;
            long delayMs = INITIAL_DELAY_MS * (long) Math.pow(2, retryCount - 1);
            try {
                TimeUnit.MILLISECONDS.sleep(delayMs);
            } catch (InterruptedException t) {
                Thread.currentThread().interrupt();
                throw new RuntimeException("Retry interrupted", t);
            }
            continue;
        } else
            throw new RuntimeException("Crud operation failed", e);
    }
}
```

使用 Java 的 Amazon DocumentDB 中的索引管理

索引可有效從 Amazon DocumentDB 集合擷取資料。如果沒有索引，DocumentDB 必須掃描集合中的每個文件，以傳回滿足指定查詢的結果。本主題提供如何使用 MongoDB Java 驅動程式建立、捨棄和列出索引的相關資訊。它也會討論如何判斷查詢中是否使用特定索引，以及如何向 Amazon DocumentDB 提供使用特定索引的提示。

主題

- 使用 Java 建立索引
- 捨棄索引
- 決定索引選擇並提供索引提示

Amazon DocumentDB 支援許多類型的索引。如需所有支援索引的完整概觀，請參閱 AWS 資料庫部落格上的如何在 Amazon DocumentDB 上編製索引。

使用 Java 建立索引

使用 MongoDB Java 驅動程式在 Amazon DocumentDB 中建立索引有兩種機制：透過 `runCommand()`，以及透過單一索引`createIndex()`的方法或多個索引`createIndexes()`的方法。使用 `createIndex()`和 `createIndexes()`方法的一個原因是，您可以透過擷取與索引建立相

關的特定錯誤來建立更好的錯誤處理。使用這些方法的另一個原因是 MongDB Java runCommand() 驅動程式提供一組豐富的支援類別，用於建立和操作索引。請注意，這些支援類別只能在您使用 createIndex()或 createIndexes()方法時使用。有三種支援類別：

- **Indexes** — 此類別可做為公用程式類別，提供靜態原廠方法來建立各種類型的索引。它簡化了建立複雜索引定義的程序，通常與其他索引相關類別搭配使用。
- **IndexModel** — 這是封裝索引鍵定義及其選項的基本類別。它代表完整的索引規格，結合要編製索引的內容（索引鍵）與如何編製索引（選項）。此類別在同時建立多個索引時特別有用，因為它可讓您定義可傳遞給 createIndexes()方法的索引規格集合。
- **IndexOptions** — 這是全方位的組態類別，提供一組豐富的方法來自訂索引行為。它包含唯一索引、稀疏索引、過期時間 (TTL) 和部分篩選條件表達式的設定。透過方法鏈結，您可以設定多個選項，例如背景索引建置和唯一限制條件。

建立單一索引

此範例示範如何在背景中使用 createIndex() 方法建立單一索引。若要了解背景和前景索引的建立，請參閱 索引建置類型。下列程式碼範例使用 在背景IndexOptions中建立名稱為「unique_restaurantId_idx」的唯一索引。此IndexOptions物件接著會傳遞至 createIndex()方法。

```
collection.createIndex(
    Indexes.ascending("restaurantId"),
    new IndexOptions()
        .unique(true)
        .name("unique_restaurantId_idx")
        .background(true));
```

建立多個索引

此範例使用 createIndexes()方法建立多個索引。它會先使用 IndexModel 物件為每個索引建置選項，然後將IndexModel物件清單傳遞至 createIndexes()方法。下列程式碼範例示範如何使用 Indexes公用程式類別建立複合索引。此類別也用於指定您是否要使用遞增或遞減排序順序建立索引。建立多個索引後，它會呼叫 listIndexes()方法來驗證索引建立。

```
// Single Field Index on cuisine
IndexModel singleIndex = new IndexModel(
    Indexes.ascending("cuisine"),
    new IndexOptions().name("cuisine_idx"));
```

```
// Compound Index
IndexModel compoundIndex = new IndexModel(
    Indexes.compoundIndex(
        Indexes.ascending("address.state"),
        Indexes.ascending("priceRange")),
    new IndexOptions().name("location_price_idx"));

// Build a list of IndexModel for the indexes
List < IndexModel > indexes = Arrays.asList(
    singleIndex,
    compoundIndex
);

collection.createIndexes(indexes);

// Verify created indexes
collection.listIndexes().forEach(index - > System.out.println("Created index: " +
 index.toJson()));
```

建立稀疏和部分索引

此範例顯示 IndexModel 會針對每種索引類型建立 ，以建立稀疏和部分索引。

```
// Sparse Index Model, this will identify only those documents that have a
// michelin star rating
IndexModel sparseIndex = new IndexModel(
    Indexes.ascending("michelin.star"),
    new IndexOptions()
    .name("michelin_sparse_idx")
    .sparse(true));

// Partial Index Model where the restaurant is active and has a rating of 4 and above
IndexModel partialIndex = new IndexModel(
    Indexes.ascending("rating.average"),
    new IndexOptions()
    .name("high_rated_active_idx")
    .partialFilterExpression(
        Filters.and(
            Filters.eq("isActive", true),
            Filters.gte("rating.average", 4.0))));
```

建立文字索引

此範例示範如何建立文字索引。集合上只允許一個文字索引，但一個文字索引可以是涵蓋多個欄位的複合索引。在文字索引中使用多個欄位時，您也可以將權重指派給索引中的每個欄位。Amazon DocumentDB 不支援陣列欄位上的文字索引，即使您可以在複合文字索引中使用最多 30 個欄位，但只能為三個欄位指派權重。

```
IndexModel textIndex = new IndexModel(
    new Document()
    .append("name", "text")
    .append("description", "text")
    .append("cuisine", "text"),
    new IndexOptions()
    .name("restaurant_text_idx")
    .weights(new Document()
        .append("name", 10) // Restaurant name gets highest weight
        .append("description", 5) // Description get medium weight
        .append("cuisine", 2) // Cuisine type gets low weight
    ));

collection.createIndex(textIndex.getKeys(), textIndex.getOptions());
```

使用 建立索引 **runCommand()**

Amazon DocumentDB 支援平行索引建立，以減少建立索引所需的時間。平行索引會使用多個並行工作者。用於建立索引的預設工作者為兩個。如需詳細資訊，請參閱 AWS 資料庫部落格上的解鎖 Amazon DocumentDB 中平行索引的力量。目前，當您使用 `createIndex()`或 時，MongDB Java 驅動程式不支援指定工作者選項，`createIndexes()`因此指定工作者的唯一方法是透過 `runCommand`。下列程式碼範例示範如何使用 `runCommand`建立索引，將工作者增加到四個：

```
Document command = new Document("createIndexes", "Restaurants")
    .append("indexes", Arrays.asList(
        new Document("key", new Document("name", 1))
        .append("name", "restaurant_name_idx")
        .append("workers", 4) // Specify number of workers
    ));

Document commendResult = connectedDB.runCommand(command);
```

捨棄索引

MongoDB Java 驅動程式提供多種方法，可讓您捨棄索引、處理不同的案例和您的偏好設定。您可以依名稱、金鑰規格捨棄索引，或一次捨棄所有索引。方法 `dropIndex()`和 `dropIndexes()`可

以在集合物件上調用，以捨棄索引。依名稱捨棄索引時，您應該確保索引使用正確的索引名稱，這不一定是直覺式的，尤其是複合或自動產生的索引。嘗試捨棄不存在的索引將導致 MongoCommandException。default _id 索引無法捨棄，因為它可確保集合中的文件唯一性。

下列程式碼範例示範如何透過提供建立索引的欄位名稱或刪除所有索引來捨棄索引：

```
String indexName = "unique_restaurantId_idx";
Document keys = new Document("cuisine", 1);
// Drop index by name
collection.dropIndex(indexName);

// Drop index by keys
collection.dropIndex(keys);

// Drop all indexes
collection.dropIndexes();
```

使用多個索引鍵捨棄索引時，請確定有一個複合索引包含所有指定的索引鍵，且索引鍵的順序正確。上述索引建立範例程式碼顯示 "cuisine" 和 功能的複合索引鍵。如果您嘗試捨棄該複合索引鍵，但順序不是用於建立的，則 MongoCommnadException 錯誤發生如下：

```
Document keys = new Document("features", 1)
    .append("cuisine", 1);
try {
    // Drop index by keys
    collection.dropIndex(keys);
    System.out.println("Successfully dropped index with keys: " + keys.toJson());

} catch (MongoCommandException commErr) {
    System.out.println("Error dropping index: " + commErr.getErrorMessage());
    throw new RuntimeException("MongoCommandException was thrown while dropping index",
 commErr);
}
```

隨即會顯示下列錯誤：

```
Error dropping index: Cannot drop index: index not found.
Tests run: 3, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.819 sec <<< FAILURE!
com.amazon.docdb.guide.DocDBGuideTest.testindexGuide()  Time elapsed: 0.817 sec  <<<
 FAILURE!
```

```
org.opentest4j.AssertionFailedError: Unexpected exception thrown:
 java.lang.RuntimeException: MongoCommandException was thrown while dropping index
```

決定索引選擇並提供索引提示

在 Amazon DocumentDB 中使用解釋功能對您了解查詢效能和索引用量至關重要。執行查詢時，您可以附加 explain()方法，以取得查詢計劃的詳細資訊，包括正在使用哪些索引。explain() 輸出提供查詢執行階段的洞見、檢查的文件數量，以及每個階段所花費的時間。此資訊對於識別特定索引是否有效使用，或查詢是否可能受益於不同的索引結構來說非常寶貴。

explain() 方法可以使用 find()方法鏈結。explain() 方法可以採用選用ExplainVerbosity列舉，以決定 傳回的詳細程度explain()。目前，DocumentDB 僅支援 QUERY_PLANNER EXECUTION_STATS和 列舉器。下列程式碼範例示範如何取得特定查詢的查詢規劃器：

```
// Query we want to analyze
Document query = new Document()
    .append("cuisine", "Thai")
    .append("rating.average", new Document("$gte", 4.0));


Document allPlansExplain =
 collection.find(query).explain(ExplainVerbosity.QUERY_PLANNER);
System.out.println("All Plans Explain:\n" + allPlansExplain.toJson());
```

查詢規劃器詳細資訊層級會傳回下列 JSON 文件：

```
{
  "queryPlanner": {
    "plannerVersion": 1,
    "namespace": "ProgGuideData.Restaurants",
    "winningPlan": {
      "stage": "IXSCAN",
      "indexName": "cuisine_idx",
      "direction": "forward"
    }
  },
  "serverInfo": {
    "host": "guidecluster3",
    "port": 27017,
    "version": "5.0.0"
  },
```

```
  "ok": 1,
  "operationTime": {
    "$timestamp": {
      "t": 1739221668,
      "i": 1
    }
  }
}
```

您有多種選項可影響或強制 Amazon DocumentDB 使用特定索引。hint() 和 hintString()方法可讓您透過明確指定要用於查詢的索引，覆寫查詢最佳化工具的預設索引選擇行為。雖然 DocumentDB 的查詢最佳化工具通常對索引選擇做出良好的選擇，但在某些情況下，強制特定索引通過hint()或hintString()可能有益，例如在處理偏斜資料或測試索引效能時。

下列程式碼範例強制對上述程式碼中執行的相同查詢使用複合索引 "cuisine_features_idx"：

```
// Query we want to analyze
Document query = new Document()
    .append("cuisine", "Thai")
    .append("rating.average", new Document("$gte", 4.0));

List < Document > queryDocs = new ArrayList < > ();
collection.find(query).hintString("cuisine_features_idx").forEach(doc - >
 queryDocs.add(doc));
```

使用 Amazon DocumentDB 和 Java 進行事件驅動程式設計

Amazon DocumentDB 內容中的事件驅動型程式設計代表強大的架構模式，其中資料庫變更可做為觸發後續商業邏輯和程序的主要事件產生器。在 DocumentDB 集合中插入、更新或刪除記錄時，這些變更會做為事件，自動啟動各種下游程序、通知或資料同步任務。此模式在現代分散式系統中特別重要，其中多個應用程式或服務需要即時回應資料變更。在 DocumentDB 中實作事件驅動型程式設計的主要機制是變更串流。

> **Note**
>
> 本指南假設您已在正在使用的集合上啟用變更串流。請參閱 以搭配 Amazon DocumentDB 使用變更串流了解如何在集合上啟用變更串流。

從 Java 應用程式使用變更串流

MongoDB Java 驅動程式中的 watch()方法是監控 Amazon DocumentDB 中即時資料變更的主要機制。MongoClient、 MongoDatabase和 MongoCollection 物件可以在 上呼叫 watch()方法。

watch() 方法會傳回支援各種組態選項ChangeStreamIterable的 執行個體，包括更新的完整文件查詢、提供可靠性的恢復權杖和時間戳記，以及篩選變更的管道彙總階段。

ChangeStreamIterable 實作核心 Java 界面Iterable，可與 搭配使用forEach()。若要使用 擷取事件forEach()，請將回呼函數傳遞至 forEach()以處理變更的事件。下列程式碼片段示範如何在集合上開啟變更串流，以開始變更事件監控：

```
ChangeStreamIterable < Document > iterator = collection.watch();
iterator.forEach(event - > {
    System.out.println("Received a change: " + event);
});
```

另一個周遊所有變更事件的方法是開啟游標，以維持與叢集的連線，並在發生變更事件時持續接收新的變更事件。若要取得變更串流游標，請使用 ChangeStreamIterable 物件的 cursor()方法。下列程式碼範例示範如何使用游標監控變更事件：

```
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch().cursor()) {
    System.out.println(cursor.tryNext());
}
```

最佳實務是在 try-with-resource 陳述式MongoChangeStreamCursor中建立 ，或手動關閉游標。在 上呼叫 cursor()方法會ChangeStreamIterable傳回透過ChangeStreamDocument物件建立MongoChangeStreamCursor的 。

ChangeStreamDocument 類別是代表串流中個別變更事件的重要元件。它包含每個修改的詳細資訊，包括操作類型（插入、更新、刪除、取代）、文件金鑰、命名空間資訊，以及可用時的完整文件內容。類別提供存取變更事件各種層面的方法，例如getOperationType()判斷變更類型、getFullDocument()存取完整文件狀態，以及getDocumentKey()識別修改後的文件。

ChangeStreamDocument 物件提供兩個重要的資訊，即變更事件的恢復權杖和時間。

在 DocumentDB 變更串流中恢復字符和以時間為基礎的操作，為維護持續性和管理歷史資料存取提供了關鍵機制。恢復權杖是為每個變更事件產生的唯一識別符，做為書籤，允許應用程式在中斷連線或失敗後從特定點重新啟動變更串流處理。建立變更串流游標時，可以透過 resumeAfter()選項使用先前存放的恢復權杖，讓串流從停止的位置繼續，而不是從開始或遺失事件開始。

變更串流中以時間為基礎的操作提供不同的方法來管理變更事件監控的起點。`startAtOperationTime()` 選項可讓您開始監看在特定時間戳記或之後發生的變更。這些以時間為基礎的功能在需要歷史資料處理、point-in-time復原或系統間同步的情況中特別重要。

下列程式碼範例會擷取與插入文件相關聯的事件、擷取其恢復權杖，然後提供該權杖，以在插入事件之後開始監控事件。事件與更新事件相關聯，然後在發生更新時取得叢集時間，並使用該時間戳記做為進一步處理的起點。

```
BsonDocument resumeToken;
BsonTimestamp resumeTime;

try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch().cursor()) {
    System.out.println("****************** Insert Document ******************");
    ChangeStreamDocument < Document > insertChange = cursor.tryNext();
    resumeToken = insertChange.getResumeToken();
    printJson(cursor.tryNext());
}
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch()
    .resumeAfter(resumeToken)
    .cursor()) {
    System.out.println("****************** Update Document ******************");
    ChangeStreamDocument < Document > insertChange = cursor.tryNext();
    resumeTime = insertChange.getClusterTime();
    printJson(cursor.tryNext());
}
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch()
    .startAtOperationTime(resumeTime)
    .cursor()) {
    System.out.println("****************** Delete Document ******************");
    printJson(cursor.tryNext());
  }
```

根據預設，更新變更事件不包含完整文件，而且只包含所做的變更。如果您需要存取已更新的完整文件，您可以呼叫 ChangeStreamIterable 物件上的 `fullDocument()`方法。請記住，當您要求針對更新事件傳回完整文件時，它會傳回進行變更串流呼叫時存在的文件。

此方法使用FullDocument列舉做為參數。目前，Amazon DocumentDB 僅支援 DEFAULT 和 UPDATE_LOOKUP 值。下列程式碼片段顯示如何在開始監看變更時，要求更新事件的完整文件：

```
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch().fullDocument(FullDocument.UPDATE_LOOKUP).cursor())
```

使用 JSON 結構描述驗證

使用$jsonSchema評估查詢運算子，您可以驗證要插入集合的文件。

主題

- 建立和使用 JSON 結構描述驗證
- 支援的關鍵字
- bypassDocumentValidation
- 限制

建立和使用 JSON 結構描述驗證

使用結構描述驗證建立集合

您可以使用 createCollection 操作和驗證規則建立集合。這些驗證規則會在 Amazon DocumentDB 文件的插入或更新期間套用。下列程式碼範例顯示一組員工的驗證規則：

```
db.createCollection("employees", {
   "validator": {
      "$jsonSchema": {
         "bsonType": "object",
         "title": "employee validation",
         "required": [ "name", "employeeId"],
         "properties": {
            "name": {
                  "bsonType": "object",
                  "properties": {
                     "firstName": {
                        "bsonType": ["string"]
                     },
                     "lastName": {
                        "bsonType": ["string"]
                     }
                  },
                  "additionalProperties" : false
            },
            "employeeId": {
```

```
               "bsonType": "string",
               "description": "Unique Identifier for employee"
            },
             "salary": {
               "bsonType": "double"
            },
            "age": {
               "bsonType": "number"
            }
         },
         "additionalProperties" : true
      }
   },
   "validationLevel": "strict", "validationAction": "error"
} )
```

插入有效的文件

下列範例會插入符合上述結構描述驗證規則的文件：

```
db.employees.insert({"name" : { "firstName" : "Carol" , "lastName" : "Smith"},
 "employeeId": "c720a" , "salary": 1000.0 })
db.employees.insert({ "name" : { "firstName" : "William", "lastName" : "Taylor" },
 "employeeId" : "c721a", "age" : 24})
```

插入無效的文件

下列範例會插入不符合上述結構描述驗證規則的文件。在此範例中，employeeId 值不是字串：

```
db.employees.insert({
    "name" : { "firstName" : "Carol" , "lastName" : "Smith"},
    "employeeId": 720 ,
    "salary": 1000.0
})
```

此範例顯示文件中的不正確語法。

修改集合

collMod 命令用於新增或修改現有集合的驗證規則。下列範例會將薪資欄位新增至必要欄位清單：

```
db.runCommand({"collMod" : "employees",
   "validator": {
```

```
      "$jsonSchema": {
         "bsonType": "object",
         "title": "employee validation",
         "required": [ "name", "employeeId", "salary"],
         "properties": {
            "name": {
                  "bsonType": "object",
                  "properties": {
                     "firstName": {
                        "bsonType": ["string"]
                     },
                     "lastName": {
                        "bsonType": ["string"]
                     }
                  },
                  "additionalProperties" : false
            },
            "employeeId": {
               "bsonType": "string",
               "description": "Unique Identifier for employee"
            },
             "salary": {
               "bsonType": "double"
            },
            "age": {
               "bsonType": "number"
            }
         },
         "additionalProperties" : true
      }
   }
} )
```

在驗證規則變更之前新增的定址文件

若要處理在驗證規則變更之前新增至集合的文件，請使用下列validationLevel修飾詞：

- strict：在所有插入和更新上套用驗證規則。
- 中等：將驗證規則套用至現有的有效文件。在更新期間，不會檢查現有的無效文件。

在下列範例中，更新名為 "employees" 集合的驗證規則後，薪資欄位為必要欄位。更新下列文件將會失敗：

```
db.runCommand({
    update: "employees",
    updates: [{
        q: { "employeeId": "c721a" },
        u: { age: 25 , salary : 1000},
        upsert: true }]
})
```

Amazon DocumentDB 傳回下列輸出：

```
{
"n" : 0,
    "nModified" : 0,
    "writeErrors" : [
        {
"index" : 0,
            "code" : 121,
            "errmsg" : "Document failed validation"
        }
    ],
    "ok" : 1,
    "operationTime" : Timestamp(1234567890, 1)
}
```

將驗證層級更新為 moderate將允許成功更新上述文件：

```
db.runCommand({
    "collMod" : "employees",
    validationLevel : "moderate"
})

db.runCommand({
    update: "employees",
    updates: [{
        q: { "employeeId": "c721a" },
        u: { age: 25 , salary : 1000},
        upsert: true }]
})
```

Amazon DocumentDB 傳回下列輸出：

```
{
```

```
"n" : 1,
    "nModified" : 1,
    "ok" : 1,
    "operationTime" : Timestamp(1234567890, 1)
}
```

使用 $jsonSchema 擷取文件

運算$jsonSchema子可以做為篩選條件來查詢符合 JSON 結構描述的文件。這是頂層運算子，可以做為頂層欄位出現在篩選文件中，或與查詢運算子搭配使用$or，例如 $and、 和 $nor。下列範例顯示使用 $jsonSchema 做為個別篩選條件，並與其他篩選條件運算子搭配使用：

插入「員工」集合的文件：

```
{ "name" : { "firstName" : "Carol", "lastName" : "Smith" }, "employeeId" : "c720a",
 "salary" : 1000 }
{ "name" : { "firstName" : "Emily", "lastName" : "Brown" }, "employeeId" : "c720b",
 "age" : 25, "salary" : 1050.2 }
{ "name" : { "firstName" : "William", "lastName" : "Taylor" }, "employeeId" : "c721a",
 "age" : 24, "salary" : 1400.5 }
{ "name" : { "firstName" : "Jane", "lastName" : "Doe" }, "employeeId" : "c721a",
 "salary" : 1300 }
```

僅以$jsonSchema運算子篩選的集合：

```
db.employees.find({
       $jsonSchema: { required: ["age"] } })
```

Amazon DocumentDB 傳回下列輸出：

```
{ "_id" : ObjectId("64e5f91c6218c620cf0e8f8b"), "name" : { "firstName" : "Emily",
 "lastName" : "Brown" }, "employeeId" : "c720b", "age" : 25, "salary" : 1050.2 }
{ "_id" : ObjectId("64e5f94e6218c620cf0e8f8c"), "name" : { "firstName" : "William",
 "lastName" : "Taylor" }, "employeeId" : "c721a", "age" : 24, "salary" : 1400.5 }
```

使用 $jsonSchema 運算子和另一個運算子篩選的集合：

```
db.employees.find({
       $or: [{ $jsonSchema: { required: ["age", "name"]}},
            { salary: { $lte:1000}}]});
```

Amazon DocumentDB 傳回下列輸出：

```
{ "_id" : ObjectId("64e5f8886218c620cf0e8f8a"), "name" : { "firstName" : "Carol",
 "lastName" : "Smith" }, "employeeId" : "c720a", "salary" : 1000 }
{ "_id" : ObjectId("64e5f91c6218c620cf0e8f8b"), "name" : { "firstName" : "Emily",
 "lastName" : "Brown" }, "employeeId" : "c720b", "age" : 25, "salary" : 1050.2 }
{ "_id" : ObjectId("64e5f94e6218c620cf0e8f8c"), "name" : { "firstName" : "William",
 "lastName" : "Taylor" }, "employeeId" : "c721a", "age" : 24, "salary" : 1400.5 }
```

集合在彙總篩選條件$match中以$jsonSchema運算子和 篩選：

```
db.employees.aggregate(
    [{ $match: {
        $jsonSchema: {
            required: ["name", "employeeId"],
            properties: {"salary" :{"bsonType": "double"}}
        }
       }
    }]
)
```

Amazon DocumentDB 傳回下列輸出：

```
{
"_id" : ObjectId("64e5f8886218c620cf0e8f8a"),
 "name" : { "firstName" : "Carol", "lastName" : "Smith" },
"employeeId" : "c720a",
"salary" : 1000
}
{
"_id" : ObjectId("64e5f91c6218c620cf0e8f8b"),
"name" : { "firstName" : "Emily", "lastName" : "Brown" },
"employeeId" : "c720b",
"age" : 25,
"salary" : 1050.2
}
{
"_id" : ObjectId("64e5f94e6218c620cf0e8f8c"),
"name" : { "firstName" : "William", "lastName" : "Taylor" },
"employeeId" : "c721a",
"age" : 24,
"salary" : 1400.5
}
```

```
{
"_id" : ObjectId("64e5f9786218c620cf0e8f8d"),
"name" : { "firstName" : "Jane", "lastName" : "Doe" },
"employeeId" : "c721a",
"salary" : 1300
}
```

檢視現有的驗證規則

若要查看集合上的現有驗證規則，請使用：

```
db.runCommand({
    listCollections: 1,
    filter: { name: 'employees' }
})
```

Amazon DocumentDB 傳回下列輸出：

```
{
    "waitedMS" : NumberLong(0),
    "cursor" : {
        "firstBatch" : [
            {
                "name" : "employees",
                "type" : "collection",
                "options" : {
                    "autoIndexId" : true,
                    "capped" : false,
                    "validator" : {
                        "$jsonSchema" : {
                            "bsonType" : "object",
                            "title" : "employee validation",
                            "required" : [
                                "name",
                                "employeeId",
                                "salary"
                            ],
                            "properties" : {
                                "name" : {
                                    "bsonType" : "object",
                                    "properties" : {
                                        "firstName" : {
                                            "bsonType" : [
```

```
                                        "string"
                                    ]
                                },
                                "lastName" : {
                                    "bsonType" : [
                                        "string"
                                    ]
                                }
                            },
                            "additionalProperties" : false
                        },
                        "employeeId" : {
                            "bsonType" : "string",
                            "description" : "Unique Identifier for employee"
                        },
                        "salary" : {
                            "bsonType" : "double"
                        },
                        "age" : {
                            "bsonType" : "number"
                        }
                    },
                    "additionalProperties" : true
                }
            },
            "validationLevel" : "moderate",
            "validationAction" : "error"
        },
        "info" : {
            "readOnly" : false
        },
        "idIndex" : {
            "v" : 2,
            "key" : {
                "_id" : 1
            },
            "name" : "_id_",
            "ns" : "test.employees"
        }
    }
],
"id" : NumberLong(0),
"ns" : "test.$cmd.listCollections"
},
```

```
    "ok" : 1,
    "operationTime" : Timestamp(1692788937, 1)
}
```

Amazon DocumentDB $out 也會在彙總階段中保留驗證規則。

支援的關鍵字

create 和 collMod命令支援下列欄位：

- **Validator** — 支援 $jsonSchem運算子。
- **ValidationLevel** — 支援 off、 strict和 moderate值。
- **ValidationAction** — 支援 error值。

$jsonSchema 運算子支援下列關鍵字：

- additionalItems
- additionalProperties
- allOf
- anyOf
- bsonType
- dependencies
- description
- enum
- exclusiveMaximum
- exclusiveMinimum
- items
- maximum
- minimum
- maxItems
- minItems
- maxLength
- minLength
- maxProperties

- minProperties
- multipleOf
- not
- oneOf
- pattern
- patternProperties
- properties
- required
- title
- type
- uniqueItems

bypassDocumentValidation

Amazon DocumentDB bypassDocumentValidation 支援下列命令和方法：

- insert
- update
- findAndModify
- $out aggregate命令中的 階段和db.collection.aggregate()方法中的 階段

Amazon DocumentDB 不支援 的下列命令bypassDocumentValidation：

- $merge 在 aggregate 命令中和在 db.collection.aggregate()方法中
- mapReduce 命令和db.collection.mapReduce()方法
- applyOps 命令

限制

下列限制適用於$jsonSchema驗證：

- 當 操作失敗驗證規則時，Amazon DocumentDB 會傳回錯誤「文件驗證失敗」。
- Amazon DocumentDB 彈性叢集不支援 $jsonSchema。

在 Amazon DocumentDB 中查詢

本節說明使用 Amazon DocumentDB 查詢的所有層面。

主題

- 查詢文件
- 查詢計劃
- 說明結果
- 查詢規劃器 v2
- 查詢規劃器 v3
- 使用 Amazon DocumentDB 查詢地理空間資料
- 部分索引
- 使用 Amazon DocumentDB 執行文字搜尋

查詢文件

有時候，您可能需要查詢線上商店的庫存，讓客戶可以查看和購買您銷售的商品。查詢集合相當容易，無論您需要的是集合中的所有文件，或僅需要滿足特定條件的文件。

若要查詢文件，請使用 `find()` 操作。`find()` 命令擁有單一文件參數，負責定義選擇要傳回的文件時所使用的條件。`find()` 輸出的文件格式會是單一行文字且沒有換行。若要將輸出文件格式化，以更方便閱讀，請使用 `find().pretty()`。這個主題的所有範例都使用 `.pretty()` 設定輸出格式。

下列程式碼範例使用您在上述兩個練習中插入`example`集合的四個文件 - `insertMany()` `insertOne()`位於 。 新增文件

主題

- 擷取集合中的所有文件
- 擷取符合欄位值的文件
- 擷取符合內嵌文件的文件
- 擷取符合內嵌文件中欄位值的文件
- 擷取符合陣列的文件
- 擷取符合陣列中值的文件
- 使用運算子擷取文件

擷取集合中的所有文件

若要擷取集合中的所有文件，請使用空的查詢文件使用 `find()` 操作。

以下查詢會傳回 `example` 集合中的所有文件。

```
db.example.find( {} ).pretty()
```

擷取符合欄位值的文件

若要擷取符合某個欄位和值的所有文件，請使用識別要符合的欄位和值的查詢文件使用 `find()` 操作。

使用先前文件，此查詢會傳回所有文件，其中「項目」欄位等於「筆」。

```
db.example.find( { "Item": "Pen" } ).pretty()
```

擷取符合內嵌文件的文件

若要尋找符合嵌入文件的所有文件，請使用 `find()` 操作，其附帶的查詢文件會指定內嵌文件名稱和該內嵌文件的所有欄位和值。

當符合嵌入文件，文件的內嵌文件必須擁有與查詢相同的名稱。此外，內嵌文件的欄位和值必須符合查詢。

以下查詢只會傳回「Poster Paint」(廣告顏料) 文件。這是因為「Pen」(筆) 的 `OnHand` 和 `MinOnHand` 有不同值，而「Spray Paint」(噴漆) 比查詢文件多一個欄位 (`OrderQnty`)。

```
db.example.find({"Inventory": {
    "OnHand": 47,
    "MinOnHand": 50 } } ).pretty()
```

擷取符合內嵌文件中欄位值的文件

若要尋找符合嵌入文件的所有文件，請使用 `find()` 操作，其附帶的查詢文件會指定內嵌文件名稱和該內嵌文件的所有欄位和值。

鑒於前述文件，以下查詢使用「點表示法」指定內嵌文件和關注欄位。符合這些的任何文件將會傳回，無論內嵌文件中可能有哪些其他欄位。查詢將傳回「廣告顏料」和「噴漆」，因為它們都符合指定的欄位和值。

```
db.example.find({"Inventory.OnHand": 47, "Inventory.MinOnHand": 50 }).pretty()
```

擷取符合陣列的文件

若要尋找符合陣列的所有文件，請使用 find() 操作與您感興趣的陣列名稱，以及在該陣列中的所有值。查詢將傳回所有具有該名稱的文件，其中陣列值與查詢中值及順序相同。

以下查詢只會傳回「筆」，因為「廣告顏料」有額外的顏色 (白色)，而「噴漆」有不同順序的顏色。

```
db.example.find( { "Colors": ["Red","Green","Blue","Black"] } ).pretty()
```

擷取符合陣列中值的文件

若要尋找具有特殊陣列值的所有文件，請使用 find() 操作與您感興趣的陣列名稱和值。

```
db.example.find( { "Colors": "Red" } ).pretty()
```

前述操作會傳回全部三個文件，因為每個文件都有一個名為 Colors 的陣列，且陣列中某處具有「Red」值。如果您指定數值「White」，查詢只會傳回「廣告顏料」。

使用運算子擷取文件

以下查詢會傳回其中「Inventory.OnHand」值小於 50 的所有文件。

```
db.example.find(
        { "Inventory.OnHand": { $lt: 50 } } )
```

如需支援的查詢運算子清單的詳細資訊，請參閱查詢和投影運算子。

查詢計劃

如何查看查詢計畫的 **executionStats** ？

判斷查詢執行速度比預期慢的原因時，了解查詢計劃的 executionStats 可能很有幫助。executionStats 提供從特定階段傳回的文件數量 (nReturned)、在每個階段花費的執行時間量 (executionTimeMillisEstimate)，以及產生查詢計劃(planningTimeMillis) 所需的時間量。您可以判斷查詢中最耗時的階段，以協助從 輸出集中最佳化工作executionStats，如下列查詢範例所示。executionStats 參數目前不支援 update 和 delete 命令。

> **Note**
>
> Amazon DocumentDB 在利用分散式、容錯、自我修復儲存系統的專用資料庫引擎上模擬 MongoDB 3.6 API。因此，Amazon DocumentDB 和 MongoDB 之間的查詢計劃和 的輸出`explain()`可能不同。想要控制其查詢計劃的客戶可以使用 `$hint` 運算子強制選取偏好的索引。

請在 `explain()` 命令下執行您要改善的查詢，如下所示。

```
db.runCommand({explain: {query document}}).
explain("executionStats").executionStats;
```

下面是範例操作。

```
db.fish.find({}).limit(2).explain("executionStats");
```

此操作的輸出將會如下所示。

```
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "test.fish",
        "winningPlan" : {
            "stage" : "SUBSCAN",
            "inputStage" : {
                "stage" : "LIMIT_SKIP",
                "inputStage" : {
                    "stage" : "COLLSCAN"
                }
            }
        }
    },
    "executionStats" : {
        "executionSuccess" : true,
        "executionTimeMillis" : "0.063",
        "planningTimeMillis" : "0.040",
        "executionStages" : {
            "stage" : "SUBSCAN",
            "nReturned" : "2",
```

```
            "executionTimeMillisEstimate" : "0.012",
            "inputStage" : {
                "stage" : "LIMIT_SKIP",
                "nReturned" : "2",
                "executionTimeMillisEstimate" : "0.005",
                "inputStage" : {
                    "stage" : "COLLSCAN",
                    "nReturned" : "2",
                    "executionTimeMillisEstimate" : "0.005"
                }
            }
        }
    },
    "serverInfo" : {
        "host" : "enginedemo",
        "port" : 27017,
        "version" : "3.6.0"
    },
    "ok" : 1
}
```

如果您只有興趣從上面的查詢查看 `executionStats`，可以使用下面的命令。對於小型集合，如果效能增益微乎其微，Amazon DocumentDB 查詢處理器可以選擇不使用索引。

```
db.fish.find({}).limit(2).explain("executionStats").executionStats;
```

查詢計劃快取

為了最佳化效能並縮短規劃持續時間，Amazon DocumentDB 會在內部快取查詢計劃。這可讓具有相同形狀的查詢直接使用快取計劃執行。

不過，此快取有時可能會導致相同查詢的隨機延遲；例如，通常需要一秒才能執行的查詢有時可能需要十秒的時間。這是因為隨著時間的推移，讀取器執行個體快取了查詢的各種形狀，因此會耗用記憶體。如果您遇到這種隨機慢速，則不需要採取任何動作來釋放記憶體，系統會為您管理記憶體用量，一旦記憶體達到特定閾值，就會自動釋放記憶體。

說明結果

如果您想要傳回查詢計劃的資訊，Amazon DocumentDB 支援詳細資訊模式 `queryPlanner`。`explain` 結果會以類似下列的格式傳回最佳化工具選擇的所選查詢計畫：

```
{
   "queryPlanner" : {
      "plannerVersion" : <int>,
      "namespace" : <string>,
      "winningPlan" : {
         "stage" : <STAGE1>,
         ...
         "inputStage" : {
            "stage" : <STAGE2>,
            ...
            "inputStage" : {
               ...
            }
         }
      }
   }
}
```

下列各節將定義常見explain結果。

主題

- 掃描和篩選階段
- 索引交集
- 索引聯集
- 多個索引交集/聯合
- 複合索引
- 排序階段
- 群組階段

掃描和篩選階段

最佳化工具可以選擇下列其中一個掃描：

COLLSCAN

此階段是循序收集掃描。

```
{
```

```
    "stage" : "COLLSCAN"
}
```

IXSCAN

此階段會掃描索引鍵。最佳化工具可能會在此階段擷取文件，這可能會導致稍後附加 FETCH 階段。

```
db.foo.find({"a": 1})
{
    "stage" : "IXSCAN",
    "direction" : "forward",
    "indexName" : <idx_name>
}
```

FETCH

如果最佳化工具在 IXSCAN 以外的階段擷取文件，結果將包含 FETCH 階段。例如，上述 IXSCAN 查詢可能會導致 FETCH 和 IXSCAN 階段的組合：

```
db.foo.find({"a": 1})
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXSCAN",
        "indexName" : <idx_name>
    }
}
```

IXONLYSCAN 只會掃描索引鍵。建立複合索引不會避免 FETCH。

索引交集

IXAND

如果 Amazon DocumentDB 可以使用索引交集，則可以包含具有 inputStages IXSCAN 陣列的 IXAND 階段。例如，我們可能會看到如下的輸出：

```
{
```

```
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXAND",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                "indexName" : "a_1"
            },
            {
                "stage" : "IXSCAN",
                "indexName" : "b_1"
            }
        ]
    }
}
```

索引聯集

IXOR

與索引交集類似，Amazon DocumentDB 可能包含具有運算`$or`子`inputStages`陣列的`IXOR`階段。

```
db.foo.find({"$or": [{"a": {"$gt": 2}}, {"b": {"$lt": 2}}]})
```

對於上述查詢，解釋輸出可能如下所示：

```
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXOR",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                "indexName" : "a_1"
            },
            {
                "stage" : "IXSCAN",
                "indexName" : "b_1"
            }
        ]
    }
```

```
}
```

多個索引交集/聯合

Amazon DocumentDB 可以將多個索引交集或聯集階段結合在一起，然後擷取結果。例如：

```
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXOR",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                ...
            },
            {
                "stage" : "IXAND",
                "inputStages" : [
                    {
                        "stage" : "IXSCAN",
                        ...
                    },
                    {
                        "stage" : "IXSCAN",
                        ...
                    }
                ]
            }
        ]
    }
}
```

索引交集或聯集階段的使用不受索引類型（稀疏、複合等）影響。

複合索引

Amazon DocumentDB 複合索引用量在索引欄位的開始子集中不受限制；它可以使用索引搭配尾碼部分，但可能不是非常有效率。

例如，的複合索引`{ a: 1, b: -1 }`可以支援以下這三個查詢：

```
db.orders.find( { a: 1 } )

db.orders.find( { b: 1 } )

db.orders.find( { a: 1, b: 1 } )
```

排序階段

如果請求的排序索引鍵上有索引 (Amazon DocumentDB)，Amazon DocumentDB 可以使用索引來取得順序。在這種情況下，結果不會包含SORT階段，而是IXSCAN階段。如果最佳化工具偏好純排序，它將包含如下所示的階段：

```
{
    "stage" : "SORT",
    "sortPattern" : {
        "a" : 1,
        "b" : -1
    }
}
```

群組階段

Amazon DocumentDB 支援兩種不同的群組策略：

- SORT_AGGREGATE：磁碟排序彙總。
- HASH_AGGREGATE：在記憶體雜湊彙總中。

查詢規劃器 v2

Amazon DocumentDB （規劃器 2.0 版） 的新查詢規劃器具有進階查詢最佳化功能並改善效能。使用 find和 update運算子搭配索引時，適用於 Amazon DocumentDB 5.0 的規劃器 2.0 版可提供比先前版本高達 10 倍的效能改善。效能改善主要來自使用更理想的索引計畫，以及啟用對否定運算子 ($neq、$nin) 和巢狀 等運算子的索引掃描支援$elementMatch。規劃器 2.0 版查詢透過更好的成本估算技術、最佳化演算法和增強的穩定性，更快速地執行。規劃器 2.0 版也支援計畫快取篩選條件 APIs，可增強規劃器穩定性。使用此功能時，Amazon DocumentDB 5.0 現在提供從不同版本的查詢規劃器中選取 的功能。

主題

- 先決條件
- 選取規劃器 2.0 版做為預設查詢規劃器
- 最佳實務
- 限制
- Find 和 Update運算子的改進
- 計劃快取篩選條件 API
- 規劃器版本 1.0、2.0 和 MongoDB 之間的潛在行為差異
- 規劃器 2.0 版使用 MongoDB 彌補行為差距

先決條件

下列先決條件適用於規劃器 2.0 版：

- 引擎 5.0 版的所有 AWS 區域 都提供規劃器 2.0 版。
- 若要選擇使用 2.0 版做為預設查詢規劃器，您的叢集必須位於 Amazon DocumentDB 5.0 版的引擎修補程式 3.0.15902 版或更新版本。如需更新至最新引擎版本修補程式的步驟，請參閱 對叢集的引擎版本執行修補程式更新。
- 若要將規劃器 2.0 版設定為預設查詢規劃器，您需要 IAM 許可才能更新叢集參數群組。
- 對於 Amazon DocumentDB 8.0，規劃器 3.0 版是預設查詢規劃器。如需詳細資訊，請參閱 查詢規劃器 v3。

選取規劃器 2.0 版做為預設查詢規劃器

使用下列步驟，從主控台或 CLI 選取 2.0 做為預設查詢規劃器：

- 請依照中的步驟修改 Amazon DocumentDB 叢集參數修改叢集的參數群組。
- 對於名為「plannerVersion」的參數，將值變更為 2.0，表示規劃器版本 2.0。
- 選取立即套用 （選取在重新開機時套用會使選取無效，直到叢集下次重新開機為止）。

最佳實務

如需預期結果，請在套用規劃器 2.0 版時使用下列最佳實務：

- 在全域叢集中，為兩個區域選取叢集參數群組中的相同`plannerVersion`值 (1.0 或 2.0)。請注意，在主要和次要區域中選取不同的規劃器版本可能會導致查詢行為和效能不一致。

- 在排定的維護時段或減少的流量期間更新至規劃器 2.0 版將最具破壞性，因為如果在工作負載主動執行時變更規劃器版本，可能會提高錯誤率。
- 規劃器 2.0 版最適用於 MongoDB shell 5.0 版。

限制

下列限制適用於規劃器 2.0 版：

- 彈性叢集不支援規劃器 2.0 版，這會回到規劃器 1.0 版。
- 彙總和不同的命令不支援規劃器 2.0 版，這會回到規劃器 1.0 版。
- 規劃器 2.0 版中的計劃快取篩選條件不支援包含 regex、文字搜尋、地理空間、jsonschema 或篩選條件`$expr`的查詢。

Find 和 **Update**運算子的改進

規劃器 2.0 版最佳化基本操作`update`，包括 `find`、`delete`、 和 `find-and-modify`命令。下列標籤區段顯示索引的增強功能，以及規劃器 2.0 版的查詢效能改善：

Enhanced index support

- 規劃器 2.0 版新增了對否定運算子的索引支援`$ne`，包括 `$nin`、`$not {eq}`、 和 `$not {in}`，以及 `$type`和 `$elemMatch`。

```
Sample Document: { "x": 10, "y": [1, 2, 3] }

db.foo.createIndex({ "x": 1, "y": 1 })
db.foo.find({ "x": {$nin: [20, 30] }})
db.foo.find({"x":{ $type: "string" }})

db.foo.createIndex({"x.y": 1})
db.foo.find({"x":{$elemMatch:{"y":{$elemMatch:{"$gt": 3 }}}}})
```

- 即使查詢表達式中`$exists`不存在 ，規劃器 2.0 版仍會利用稀疏或部分索引。

```
Sample Document: {"name": "Bob", "email": "example@fake.com" }

Using Planner Version 1.0, you can specify the command as shown below:
db.foo.find({email: "example@fake.com", email: {$exists: true}})
```

```
Using Planner Version 2.0, you can specify command without $exists:
db.foo.find({ email: "example@fake.com" })
```

- 即使查詢條件未完全符合部分索引篩選條件表達式，規劃器 2.0 版仍會利用部分索引。

```
Sample Document: {"name": "Bob", "age": 34}
db.foo.createIndex({"age":1},{partialFilterExpression:{"age":{$lt:50}}})

With Planner Version 1.0, index is used only when the query condition meets the
 partial
 index filter criterion:
     db.foo.find({"age":{$lt:50}})

With Planner Version 2.0, index is used even when the query condition doesn't meet
 the index
criterion:
db.foo.find({"age":{$lt:30}})
```

- 規劃器 2.0 版使用部分索引掃描搭配 $elemMatch 查詢。

```
Sample Document: {"name": "Bob", "age": [34,35,36]}
db.foo.createIndex({"age":1},{partialFilterExpression:{"age":{$lt:50,$gt:20}}})
db.foo.find({age:{$elemMatch:{$lt:50,$gt:20}}})
```

- 規劃器 2.0 版包含 的索引掃描支援$regex，而不需要在應用程式程式碼$hint中提供 。僅$regex支援字首搜尋的索引。

```
Sample Document: { "x": [1, 2, 3], "y": "apple" }
db.foo.createIndex({ "x": 1, "y": 1 })
db.foo.find({"y":{ $regex: "^a" }})
```

- 規劃器 2.0 版可改善涉及多金鑰索引的查詢效能，以及多金鑰欄位的相等條件。

```
Sample Document: {"x": [1, 2, 3],
"y": 5}
db.foo.createIndex({"x": 1, "y":1})
db.foo.find({"x": 2,
"y": {$gt: 1}}).limit(1)
```

- 規劃器 2.0 版可改善涉及多個篩選條件的查詢效能，尤其是文件大於 8 KB 的集合。

```
Sample Document: {"x": 2,
"y": 4,
```

```
"z": 9,
"t": 99}
db.foo.find({$and: [{"x": {$gt : 1}, "y": {$gt : 3}, "z": {$lt : 10},
"t":{$lt : 100}}]})
```

- 規劃器 2.0 版透過消除排序階段，在將$in運算子與複合索引搭配使用時改善查詢的效能。

```
Sample Document: {"x": 2,
"y": 4,
"z": 9,
"t": 99}
db.foo.createIndex({"x":1, "y":1})
db.foo.find({"x":2,
"y":$in:[1,2,3,4]}).sort({x:1,y:1})
```

它也可以改善使用多金鑰索引搭配 $in元素的查詢效能。

```
Sample Document: {"x": [1, 2, 3]}
db.foo.createIndex({"x": 1})
db.foo.find("x":{$in:[>100 elements]})
```

Query performance improvements

- 規劃器 2.0 版可改善涉及多個篩選條件的查詢效能，尤其是文件大於 8 KB 的集合。

```
Sample Document: {"x": 2,
"y": 4,
"z": 9,
"t": 99}
db.foo.find({$and: [{"x": {$gt : 1}, "y": {$gt : 3}, "z": {$lt : 10},
"t":{$lt : 100}}]})
```

- 規劃器 2.0 版透過消除排序階段，在將$in運算子與複合索引搭配使用時改善查詢的效能。

```
Sample Document: {"x": 2,
"y": 4,
"z": 9,
"t": 99}
db.foo.createIndex({"x":1, "y":1})
db.foo.find({"x":2,
"y":$in:[1,2,3,4]}).sort({x:1,y:1})
```

它也可以改善使用多索引鍵索引搭配 $in元素的查詢效能。

```
Sample Document: {"x": [1, 2, 3]}
db.foo.createIndex({"x": 1})
db.foo.find("x":{$in:[>100 elements]})
```

計劃快取篩選條件 API

> **Note**
>
> 計劃快取篩選條件不支援文字索引。

- 規劃器 2.0 版新增索引篩選功能的支援，可讓您指定特定查詢形狀可以使用的索引清單。此功能可透過 API 存取，並且可以從伺服器端控制。如果您遇到查詢迴歸，此功能可讓您更快速、更靈活地解決問題，而無需修改應用程式程式碼。

  ```
  db.runCommand({ planCacheSetFilter: <collection>, query: <query>,
  sort: <sort>, // optional,
  indexes: [ <index1>, <index2>, ...],
  comment: <any> // optional})
  ```

  若要列出集合上的所有篩選條件，請使用下列命令：

  ```
  db.runCommand(
  {
  planCacheListFilters: <collection>
  }
  )
  ```

  此命令會顯示集合上的所有索引篩選條件。輸出範例：

  ```
  {
  "filters" : [
  {
  "query" : {a: "@", b: "@"},
  "sort" : {a: 1},
  "indexes" : [
  ```

```
<index1>,
...
]
},
...
],
"ok": 1
}
```

- 您可以從explain命令輸出使用兩個新欄位來分析規劃器 2.0 版的索引篩選： indexFilterSet和indexFilterApplied。如果集合上有符合查詢形狀的索引篩選條件，indexFilterSet則 設定為「true」。如果 ，則 indexFilterApplied設定為「true」，並且只有在查詢套用索引篩選條件並使用篩選條件清單中的索引選擇計劃時。

  您可以使用下列命令清除索引篩選條件：

```
db.runCommand(
{
planCacheClearFilters: <collection>>
query: <query pattern>, // optional
sort: <sort specification>, // optional
comment: <any>. //optional
}
)
```

  若要清除集合 "foo" 上的所有篩選條件，請使用下列命令：

```
db.runCommand({planCacheClearFilters: "foo"})
```

  若要清除具有任何排序的特定查詢形狀，您可以從 的輸出複製並貼上查詢形狀planCacheListFilters：

```
db.runCommand({planCacheClearFilters: "foo", query: {a: @}})
```

  若要清除具有要排序之指定欄位的特定查詢形狀，您可以從 的輸出複製並貼上查詢形狀planCacheListFilters：

```
db.runCommand({planCacheClearFilters: "foo", query: {a: @},sort: {a: 1}})
```

規劃器版本 1.0、2.0 和 MongoDB 之間的潛在行為差異

在某些情況下，規劃器 2.0 版可能會產生與 MongoDB 中結果稍有不同的結果。本節會逐步解說這些可能性的一些範例。

$(update) and $(projection)

- 在某些情況下，MongoDB 中的 `$(update)`和 `$(projection)`運算子的行為可能與 Amazon DocumentDB 的規劃器 1.0 版不同。以下是一些範例：

```
db.students_list.insertMany( [ { _id: 5, student_ids: [ 100, 200 ], grades: [ 95,
 100 ], grad_year: [ 2024, 2023 ] } ] )
```

```
db.students_list.updateOne({ student_ids: 100, grades: 100, grad_year: 2024 },
{ $set: { “grad_year.$”: 2025 } }
```

  - 規劃器 1.0 版 — 更新欄位 2022
  - MongoDB — 更新欄位 2022
  - 規劃器 2.0 版 — 更新欄位 2021

-
```
db.col.insert({x:[1,2,3]})
db.col.update({$and:[{x:1},{x:3}]},{$set:{"x.$":500}})
```

  - 規劃器 1.0 版 — 隨機更新第一個相符的元素
  - MongoDB — 隨機更新第一個相符的元素
  - 規劃器 2.0 版 — 不進行更新

-
```
db.col.insert({x:[1,2,3]})
db.col.find()
```

  - 規劃器 1.0 版 — 隨機選取相符的元素
  - MongoDB — 隨機選取相符的元素
  - 規劃器 2.0 版 — 不進行選擇

-
```
db.col.insert({x:100})
db.col.update({x:100},{x:100})
```

  - 規劃器 1.0 版 — nModified變更

  - MongoDB — nModified count 變更
  - 規劃器 2.0 版 — 使用相同值更新時，nModified 計數不會變更。
- 當$(update)運算子與 搭配使用時$setOnInsert，規劃器 1.0 版和 MongoDB 擲回錯誤，但規劃器 2.0 版不會。
- 重新命名不存在的欄位以在規劃器 2.0 版中$field擲回錯誤，而 不會在規劃器 1.0 版和 MongoDB 中產生更新。

Index behavior

- 當 套用不適合的索引$hint時，規劃器 2.0 版會擲回錯誤，而規劃器 1.0 版和 MongoDB 則不會。

```
// Insert
db.col.insert({x:1})
db.col.insert({x:2})
db.col.insert({x:3})

// Create index on x with partialFilter Expression {x:{$gt:2}}
db.col.createIndex({x:1},{partialFilterExpression:{x:{$gt:2}}})

// Mongodb allows hint on the following queries
db.col.find({x:1}).hint("x_1")
// result is no documents returned because {x:1} is not indexed by the partial
 index
// Without $hint mongo should return {x:1}, thus the difference in result between
 COLSCAN and IXSCAN

DocumentDB will error out when $hint is applied on such cases.
db.col.find({x:1}).hint("x_1")
Error: error: {
    "ok" : 0,
    "operationTime" : Timestamp(1746473021, 1),
    "code" : 2,
    "errmsg" : "Cannot use Hint for this Query. Index is multi key index , partial
 index or sparse index and query is not optimized to use this index."
}

rs0:PRIMARY> db.runCommand({"planCacheSetFilter": "col", "query": { location:
 {$nearSphere: {$geometry: {type: "Point", coordinates: [1, 1]}}}}, "indexes":
 ["name_1"]})
```

```
{
    "ok" : 0,
    "operationTime" : Timestamp(1750815778, 1),
    "code" : 303,
    "errmsg" : "Unsupported query shape for index filter $nearSphere"
}
```

- $near 無法在規劃器 2.0 版$hint({“$natural”:1})中使用 。

```
// indexes present are index on x and geo index

rs0:PRIMARY> db.usarestaurants.getIndexes()
[
    {
        "v" : 4,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.usarestaurants"
    },
    {
        "v" : 4,
        "key" : {
            "location" : "2dsphere"
        },
        "name" : "location_2dsphere",
        "ns" : "test.usarestaurants",
        "2dsphereIndexVersion" : 1
    }
]

// Planner Version 2.0 will throw an error when $hint is applied with index "x_1"
rs0:PRIMARY> db.usarestaurants.find({    "location":{       "$nearSphere":{
         "$geometry":{             "type":"Point",             "coordinates":
[                -122.3516,                47.6156             ]          },
          "$minDistance":1,          "$maxDistance":2000       }    } }, {
  "name":1 }).hint({"$natural": 1})
Error: error: {
    "ok" : 0,
    "operationTime" : Timestamp(1746475524, 1),
    "code" : 291,
    "errmsg" : "unable to find index for $geoNear query"
}
```

```
// Planner Version 1.0 and MongoDB will not throw an error
 db.usarestaurants.find({    "location":{       "$nearSphere":{
  "$geometry":{             "type":"Point",             "coordinates":[
            -122.3516,               47.6156             ]          },
        "$minDistance":1,          "$maxDistance":2000       }    } }, {
  "name":1 }).hint({"$natural": 1})
{ "_id" : ObjectId("681918e087dadfd99b7f0172"), "name" : "Noodle House" }
```

- 雖然 MongoDB 支援完整的 regex 索引掃描，但規劃器 2.0 版僅支援字首欄位上的 regex 索引掃描。

```
// index on x
db.col.createIndex({x:1})

// index scan is used only for prefix regexes
rs0:PRIMARY> db.col.find({x: /^x/}).explain()
{
    "queryPlanner" : {
        "plannerVersion" : 2,
        "namespace" : "test.col",
        "winningPlan" : {
            "stage" : "IXSCAN",
            "indexName" : "x_1",
            "direction" : "forward",
            "indexCond" : {
                "$and" : [
                    {
                        "x" : {
                            "$regex" : /^x/
                        }
                    }
                ]
            },
            "filter" : {
                "x" : {
                    "$regex" : /^x/
                }
            }
        }
    },
    "indexFilterSet" : false,
    "indexFilterApplied" : false,
```

```
    "ok" : 1,
    "operationTime" : Timestamp(1746474527, 1)
}

// COLSCAN is used for non-prefix regexes
rs0:PRIMARY> db.col.find({x: /x$/}).explain()
{
    "queryPlanner" : {
        "plannerVersion" : 2,
        "namespace" : "test.col",
        "winningPlan" : {
            "stage" : "COLLSCAN",
            "filter" : {
                "x" : {
                    "$regex" : /x$/
                }
            }
        }
    },
    "indexFilterSet" : false,
    "indexFilterApplied" : false,
    "ok" : 1,
    "operationTime" : Timestamp(1746474575, 1)
```

- 相較於 MongoDB，搭配規劃器 2.0 版使用計劃快取篩選條件有一些固有差異。雖然規劃器 2.0 版不支援使用計劃快取篩選條件指定「投影」和「定序」，但 MongoDB 會這麼做。不過，MongoDB 索引篩選條件僅限記憶體內，重新啟動後會遺失。規劃器 2.0 版透過重新啟動和修補程式保留索引篩選條件。

Others

- 使用規劃器 2.0 版時的 DML 稽核日誌格式與規劃器 1.0 版略有不同。

```
command -  db.col.find({x:1})

************** Audit logs generated ******************

// v1 format for dml audit logs
{"atype":"authCheck","ts":1746473479983,"timestamp_utc":"2025-05-05
 19:31:19.983","remote_ip":"127.0.0.1:47022","users":
[{"user":"serviceadmin","db":"test"}],"param":
{"command":"find","ns":"test.col","args":{"batchSize":101,"filter":
```

```
{"x":1},"find":"col","limit":18446744073709551615,"lsid":
{"id":{"$binary":"P6RCGz9ZS4iWBSSHWXW15A==","$type":"4"},"uid":
{"$binary":"6Jo8PisnEi3dte03+pJFjdCyn/5cGQL8V2KqaoWsnk8=","$type":"0"}},"maxScan":18446744

// v2 formal for dml audit logs
{"atype":"authCheck","ts":1746473583711,"timestamp_utc":"2025-05-05
 19:33:03.711","remote_ip":"127.0.0.1:37754","users":
[{"user":"serviceadmin","db":"test"}],"param":
{"command":"find","ns":"test.col","args":{"find":"col","filter":{"x":1},"lsid":
{"id":{"$binary":"nJ88TGCSSd
+BeD2+ZtrhQg==","$type":"4"}},"$db":"test"},"result":0}}
```

- 做為解釋計畫一部分的索引條件：

```
rs0:PRIMARY> db.col.createIndex({index1:1})
{
 "createdCollectionAutomatically" : false,
 "numIndexesBefore" : 1,
 "numIndexesAfter" : 2,
 "ok" : 1,
 "operationTime" : Timestamp(1761149251, 1)
}
```

規劃器 2.0 版說明顯示索引條件和篩選條件的計畫輸出：

```
rs0:PRIMARY> db.col.find({$and:[{price:{$eq:300}},{item:
{$eq:"apples"}}]}).explain()
{
 "queryPlanner" : {
  "plannerVersion" : 2,
  "namespace" : "test.col",
  "winningPlan" : {
   "stage" : "IXSCAN",
   "indexName" : "price_1",
   "direction" : "forward",
   "indexCond" : {
    "$and" : [
     {
      "price" : {
       "$eq" : 300
      }
     }
    ]
```

```
   },
   "filter" : {
    "$and" : [
     {
      "item" : {
       "$eq" : "apples"
      }
     }
    ]
   }
  }
 },
 "indexFilterSet" : false,
 "indexFilterApplied" : false,
 "ok" : 1,
 "operationTime" : Timestamp(1761149497, 1)
}
```

規劃器 1.0 版說明計畫輸出：

```
rs0:PRIMARY> db.col.find({$and:[{price:{$eq:300}},{item:
{$eq:"apples"}}]}).explain()
{
 "queryPlanner" : {
  "plannerVersion" : 1,
  "namespace" : "test.col",
  "winningPlan" : {
   "stage" : "IXSCAN",
   "indexName" : "price_1",
   "direction" : "forward"
  }
 },
 "ok" : 1,
 "operationTime" : Timestamp(1761149533, 1)
}
```

規劃器 2.0 版使用 MongoDB 彌補行為差距

有些方面，規劃器 2.0 版會縮小 MongoDB 的行為差距：

- 規劃器 2.0 版允許對 的平面陣列進行數值索引查詢`$elemMatch`：

```
doc: {"x" : [ [ { "y" : 1 } ] ] }

// Planner Version 2 and mongo
> db.bar.find({"x.0": {$elemMatch: {y: 1}}})
{ "_id" : ObjectId("68192947945e5846634c455a"), "x" : [ [ { "y" : 1 } ] ] }
> db.bar.find({"x": {$elemMatch: {"0.y": 1}}})
{ "_id" : ObjectId("68192947945e5846634c455a"), "x" : [ [ { "y" : 1 } ] ] }

//Whereas Planner Version 1 wouldn't return any results.
> db.bar.find({"x.0": {$elemMatch: {y: 1}}})
> db.bar.find({"x": {$elemMatch: {"0.y": 1}}})
```

- 當規劃器 1.0 版在投影中排除字串時，規劃器 2.0 版的行為會與 MongoDB 保持一致，並將其視為常值」

```
// Planner V2/ MongoDB
> db.col.find()
{ "_id" : ObjectId("681537738aa101903ed2fe05"), "x" : 1, "y" : 1 }
> db.col.find({},{x:"string"})
{ "_id" : ObjectId("681537738aa101903ed2fe05"), "x" : "string" }

// Planner V1 treats strings as exclude in projection
rs0:PRIMARY> db.col.find()
{ "_id" : ObjectId("68153744d42969f11d5cca72"), "x" : 1, "y" : 1 }
rs0:PRIMARY> db.col.find({},{x:"string"})
{ "_id" : ObjectId("68153744d42969f11d5cca72"), "y" : 1 }
```

- 規劃器 2.0 版與 MongoDB 一樣，不允許對相同欄位「x」和「x.a」進行投影：

```
// Planner version 2/MongoDB will error out
> db.col.find()
{ "_id" : ObjectId("68153da2012265816bc9ba23"), "x" : [ { "a" : 1 }, 3 ] }
 db.col.find({},{"x.a":1,"x":1}) // error

// Planner Version 1 does not error out
db.col.find()
{ "_id" : ObjectId("68153da2012265816bc9ba23"), "x" : [ { "a" : 1 }, 3 ] }

db.col.find({},{"x.a":1,"x":1})
{ "_id" : ObjectId("68153d60143af947c720d099"), "x" : [ { "a" : 1 }, 3 ] }
```

- 規劃器 2.0 版，例如 MongoDB，允許子文件上的投影：

```
// Planner Version2/MongoDB supports projections on subdocuments
 db.col.find()
{ "_id" : ObjectId("681542d8f35ace71f0a50004"), "x" : [ { "y" : 100 } ] }
> db.col.find({},{"x":{"y":1}})
{ "_id" : ObjectId("681542b7a22d548e4ac9ddea"), "x" : [ { "y" : 100 } ] }

// Planner V1 throws error if projection is subdocument
 db.col.find()
{ "_id" : ObjectId("681542d8f35ace71f0a50004"), "x" : [ { "y" : 100 } ] }
rs0:PRIMARY> db.col.find({},{"x":{"y":1}})
Error: error: {
    "ok" : 0,
    "operationTime" : Timestamp(1746223914, 1),
    "code" : 2,
    "errmsg" : "Unknown projection operator y"
}
```

- 使用規劃器 2.0 版，例如 MongoDB，投影不支援運算$子後面的欄位：

```
// Mongo and Planner Version 2 will error out
  db.col.find()
{ "_id" : ObjectId("68155fa812f843439b593f3f"), "x" : [ { "a" : 100 } ] }
 db.col.find({"x.a":100},{"x.$.a":1}) - // error

// v1 will not error out
 db.col.find()
{ "_id" : ObjectId("68155fa812f843439b593f3f"), "x" : [ { "a" : 100 } ] }
db.col.find({"x.a":100},{"x.$.a":1})
{ "_id" : ObjectId("68155dee13b051d58239cd0a"), "x" : [ { "a" : 100 } ] }
```

- 規劃器 2.0 版，例如 MongoDB，允許$hint使用：

```
// v1 will error out on $hint if there are no filters
db.col.find({}).hint("x_1")
Error: error: {
    "ok" : 0,
    "operationTime" : Timestamp(1746466616, 1),
    "code" : 2,
    "errmsg" : "Cannot use Hint for this Query. Index is multi key index , partial
 index or sparse index and query is not optimized to use this index."
}
```

```
// Mongo and Planner Version 2 will allow $hint usage
 db.col.find({}).hint("x_1")
{ "_id" : ObjectId("6818f790d5ba9359d68169cf"), "x" : 1 }
```

查詢規劃器 v3

Amazon DocumentDB 8.0 中的規劃器第 3 版支援 21 個彙總階段，包括 6 個新階段。規劃器 V3 隨附對不同命令的內建支援。與 Amazon DocumentDB 5.0 中的規劃器 v2 相比，它可提供高達 2 倍的整體效能改善。Amazon DocumentDB 8.0 中的所有新功能和運算子都與規劃器 v3 相容。Amazon DocumentDB 8.0 中由規劃器 v3 支援的新彙總階段包括 $replaceWith、$vectorSearch、$merge、$set、$unset、$bucket。規劃器 v3 也支援 Amazon DocumentDB 8.0 的新功能和運算子，包括定序、檢視、$merge、$pow、$rand、$dateTrunc、$dateToParts、$dateFromParts。

主題

- 先決條件
- 選取規劃器 3.0 版做為預設查詢規劃器
- 最佳實務
- 限制
- aggregate 和 distinct運算子的改進
- 規劃器版本 1.0、3.0 和 MongoDB 之間的潛在行為差異

先決條件

下列先決條件適用於規劃器 3.0 版：

- 引擎 8.0 版的所有 AWS 區域 都提供規劃器 3.0 版。
- 選取引擎 8.0 版時，規劃器 3.0 版是預設查詢規劃器。

選取規劃器 3.0 版做為預設查詢規劃器

如果您在 Amazon DocumentDB 8.0 中變更預設查詢規劃器，且需要還原至規劃器 v3，您可以從主控台或 CLI 執行此操作：

- 請依照中的步驟修改 Amazon DocumentDB 叢集參數修改叢集的參數群組。

- 對於名為「plannerVersion」的參數，將值變更為 3.0，表示規劃器版本 3.0。
- 選取立即套用 （選取在重新開機時套用會使選取無效，直到叢集下次重新開機為止）。

最佳實務

如需預期結果，請在套用規劃器 3.0 版時使用下列最佳實務：

- 在全域叢集中，在兩個區域的叢集參數群組中選取相同的plannerVersion值 (1.0 或 2.0 或 3.0)。請注意，在主要和次要區域中選取不同的規劃器版本可能會導致查詢行為和效能不一致。
- 在排定的維護時段或減少的流量期間更新到規劃器 3.0 版將最不會造成干擾，因為如果在工作負載主動執行時變更規劃器版本，可能會提高錯誤率。

限制

下列限制適用於規劃器 3.0 版：

- 彈性叢集不支援規劃器 3.0 版，這會回到規劃器 1.0 版。

- 雖然規劃器 v1 允許使用 'planHint' 來確保查詢最佳化工具已選取特定查詢計畫，但規劃器 v3 不允許使用 'planHint'，並依賴內部最佳化來為指定的查詢選擇最佳計畫。

aggregate 和 **distinct**運算子的改進

規劃器 3.0 版推出跨 $aggregate 階段和 $distinct 命令的改進。以下是一些最值得注意的改進。

- 規劃器會盡可能提早在管道中移動 $match 階段，減少後續階段處理的文件數量。

```
//Planner v1
db.orders.aggregate([
  { $project: { customerId: 1, orderDate: 1, totalAmount: 1 } },
  { $match: { customerId: { $gt: 1000 } } }
])

// Planner v3 pulls up the match since customerId exists in original documents
// Optimized internally as:
db.orders.aggregate([
  { $match: { customerId: { $gt: 1000 } } },  // Pulled up before project
  { $project: { customerId: 1, orderDate: 1, totalAmount: 1 } }
```

```
])
```

- 規劃器在相同欄位操作時，會自動結合 $lookup 和 $unwind 階段，減少中繼資料處理並改善效能。

```
                Example query:
//Planner v1
db.orders.aggregate([
  { $lookup: { from: "products", localField: "productId", foreignField: "_id", as:
 "productInfo" } },
  { $unwind: "$productInfo" },
  { $project: { orderDate: 1, "productInfo.name": 1, "productInfo.price": 1 } }
])

// Planner version 3.0 optimizes this internally by coalescing the $lookup and
 $unwind stages
```

- 規劃器 3.0 版推出新的 Distinct Scan 執行策略，可大幅改善低基數索引上不同操作的效能。

```
                Example query:
//// If there is a low cardinality index on "category", you may see a query plan like
 below
db.explain().products.distinct("category")

"queryPlanner" : {
        "plannerVersion" : 3,
        "namespace" : "db.products",
        "winningPlan" : {
                "stage" : "AGGREGATE",
                "inputStage" : {
                        "stage" : "DISTINCT_SCAN",
                        "inputStage" : {
                                "stage" : "IXONLYSCAN",
                                "indexName" : "category_1",
                                "direction" : "forward"
                        }
                }
        }
}
```

規劃器版本 1.0、3.0 和 MongoDB 之間的潛在行為差異

在某些情況下，規劃器 3.0 版可能會產生與規劃器 1.0 版略有不同的結果。本節會逐步解說這些可能性的一些範例。

Feature Differences

- 在空集合上或配對等先前階段篩選所有文件時，規劃器 v1 會提供輸出「欄位」：0。規劃器 v3 和 MongoDB 不會提供任何輸出。
- 使用 Plannerv1，如果上一個階段傳回的文件少於 n 個，則 {"$skip"：n} 不會略過。無論傳回的文件數量為何，Plannerv3 和 MongoDB 都會正確略過。
- 當 $lookup 中參考的外部集合不存在時，Plannerv1 會擲回錯誤。規劃器 v3 和 MongoDB 會將外部集合視為空集合，並執行 $lookup。

```
db.coll.aggregate([
    {$lookup: {from: "does_not_exist", localField: "a", foreignField: "a", as:
 "c"}}
])
```

- 只有規劃器 v1 將允許在管道規劃器 v2/v3 中使用多個 $search 會擲回錯誤，且 MongoDB 不支援該錯誤。

```
VectorSearch = { "$search": { "vectorSearch": {
   "vector": [0.2, 0.5, 0.8],
   "path": "vectorEmbedding",
   "similarity": "cosine",
   "k": 2,
   "efSearch": 1
   }}}
db.coll.aggregate([VectorSearch, VectorSearch])
```

- 只有規劃器 v3 可在 vectorSearch 階段不是第一個階段時運作。在此情況下，MongoDB 會擲回錯誤，且規劃器 v1 不支援 $vectorSearch 階段。

```
VectorSearch = { {"$vectorSearch": {
    "queryVector": [0.2, 0.5, 0.8],
    "path": "vectorEmbedding",
    "similarity": "euclidean",
    "limit": 4,
    "numCandidates": 100} }
```

```
db.coll.aggregate([$match:{}, VectorSearch])
```

使用 Amazon DocumentDB 查詢地理空間資料

本節說明如何使用 Amazon DocumentDB 查詢地理空間資料。閱讀本節後，您將能夠回答如何在 Amazon DocumentDB 中儲存、查詢和索引地理空間資料。

主題

- 概觀
- 索引和儲存地理空間資料
- 查詢地理空間資料
- 限制

概觀

地理空間的常見使用案例涉及來自資料的鄰近性分析。例如，「尋找距離西雅圖 50 英里內的所有機場」或「從指定位置尋找最近的餐廳」。Amazon DocumentDB 使用 GeoJSON 規格來表示地理空間資料。GeoJSON 是座標空間中形狀 JSON 格式的開放原始碼規格。GeoJSON 座標會同時擷取經度和緯度，代表地球狀球體上的位置。

索引和儲存地理空間資料

Amazon DocumentDB 使用「Point」GeoJSON 類型來存放地理空間資料。每個 GeoJSON 文件 （或子文件） 通常由兩個欄位組成：

- type - 正在表示的形狀，這會通知 Amazon DocumentDB 如何解譯「座標」欄位。目前，Amazon DocumentDB 僅支援點
- coordinates – 表示為陣列中物件的緯度和經度對 – 【經度、緯度】

Amazon DocumentDB 也會使用 2dsphere 索引來為地理空間資料編製索引。Amazon DocumentDB 支援索引點。Amazon DocumentDB 支援使用 2dsphere 索引進行鄰近查詢。

假設您為食品交付服務建置應用程式的情況。您想要將餐廳緯度和經度配對存放在 Amazon DocumentDB 中。若要這樣做，請先在保留經緯度對的地理空間欄位上建立索引。

```
use restaurantsdb
```

```
db.usarestaurants.createIndex({location:"2dsphere"})
```

此命令的輸出看起來會像這樣：

```
{
 "createdCollectionAutomatically" : true,
 "numIndexesBefore" : 1,
 "numIndexesAfter" : 2,
 "ok" : 1
}
```

建立索引之後，您就可以開始將資料插入 Amazon DocumentDB 集合。

```
db.usarestaurants.insert({
   "state":"Washington",
   "city":"Seattle",
   "name":"Thai Palace",
   "rating": 4.8,
   "location":{
      "type":"Point",
      "coordinates":[
         -122.3264,
         47.6009
      ]
   }
});

db.usarestaurants.insert({
   "state":"Washington",
   "city":"Seattle",
   "name":"Noodle House",
   "rating": 4.8,
   "location":{
      "type":"Point",
      "coordinates":[
        -122.3517,
         47.6159
      ]
   }
});

db.usarestaurants.insert({
   "state":"Washington",
```

```
   "city":"Seattle",
   "name":"Curry House",
   "rating": 4.8,
   "location":{
      "type":"Point",
      "coordinates":[
         -121.4517,
         47.6229
      ]
   }
});
```

查詢地理空間資料

Amazon DocumentDB 支援地理空間資料的鄰近、包含和交集查詢。鄰近查詢的良好範例是尋找低於特定距離且距另一個點（城市）超過距離的所有點（所有機場）。包含查詢的良好範例是尋找位於指定區域/多邊形（紐約）的所有點（所有機場）。交集查詢的良好範例是尋找與點（城市）相交的多邊形（狀態）。您可以使用下列地理空間運算子，從資料中取得洞見。

- **$nearSphere** - $nearSphere 是一個尋找運算子，支援從最接近到最遠的 GeoJSON 點尋找點。
- **$geoNear** - $geoNear 是一種彙總運算子，支援計算與 GeoJSON 點之間的距離，以公尺為單位。
- **$minDistance** - $minDistance 是與 $nearSphere或 搭配使用的尋找運算子$geoNear，用於篩選至少與中心點指定最小距離的文件。
- **$maxDistance** - $maxDistance 是一種尋找運算子，可與 $nearSphere或 搭配使用，$geoNear以篩選與中心點最多達指定最大距離的文件。
- **$geoWithin** - $geoWithin 是一種尋找運算子，可支援尋找完全存在於多邊形等指定形狀內的地理空間資料文件。
- **$geoIntersects** - $geoIntersects 是一種尋找運算子，支援尋找地理空間資料與指定 GeoJSON 物件相交的文件。

> **Note**
>
> $geoNear 和 $nearSphere需要您在鄰近查詢中使用的 GeoJSON 欄位上的 2dsphere 索引。

範例 1

在此範例中，您將了解如何尋找與地址（點）最接近排序的所有餐廳（點）。

若要執行這類查詢，您可以使用 $geoNear 來計算一組點與另一個點的距離。您也可以新增 distanceMultiplier以測量以公里為單位的距離。

```
db.usarestaurants.aggregate([
   {
      "$geoNear":{
         "near":{
            "type":"Point",
            "coordinates":[
               -122.3516,
               47.6156
            ]
         },
         "spherical":true,
         "distanceField":"DistanceKilometers",
         "distanceMultiplier":0.001
      }
   }
])
```

上述命令會傳回與指定點相隔的距離（最接近最遠）排序餐廳。此命令的輸出看起來會像這樣

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "state" : "Washington", "city" :
 "Seattle", "name" : "Noodle House", "rating" : 4.8, "location" : { "type" : "Point",
 "coordinates" : [ -122.3517, 47.6159 ] }, "DistanceKilometers" : 0.03422834547294996 }
{ "_id" : ObjectId("611f3da185009a81ad38e74a"), "state" : "Washington", "city" :
 "Seattle", "name" : "Thai Palace", "rating" : 4.8, "location" : { "type" : "Point",
 "coordinates" : [ -122.3264, 47.6009 ] }, "DistanceKilometers" : 2.5009390081704277 }
{ "_id" : ObjectId("611f3dae85009a81ad38e74c"), "state" : "Washington", "city" :
 "Seattle", "name" : "Curry House", "rating" : 4.8, "location" : { "type" : "Point",
 "coordinates" : [ -121.4517, 47.6229 ] }, "DistanceKilometers" : 67.52845344856914 }
```

> **Note**
>
> 在 Amazon DocumentDB 8.0（引擎修補程式 4.0.5748 版）和更新版本中取代$geoNear階段的 limit和 num選項，以及預設限制為 100 個文件。下列使用 limit和 num 的範例僅適用於 Amazon DocumentDB 8.0（引擎修補程式 4.0.4448 版）和舊版。

若要限制查詢中的結果數量，請使用 `limit`或 num選項。

`limit`:

```
db.usarestaurants.aggregate([
   {
      "$geoNear":{
         "near":{
            "type":"Point",
            "coordinates":[
               -122.3516,
               47.6156
            ]
         },
         "spherical":true,
         "distanceField":"DistanceKilometers",
         "distanceMultiplier":0.001,
         "limit": 10
      }
   }
])
```

num:

```
db.usarestaurants.aggregate([
   {
      "$geoNear":{
         "near":{
            "type":"Point",
            "coordinates":[
               -122.3516,
               47.6156
            ]
         },
         "spherical":true,
         "distanceField":"DistanceKilometers",
         "distanceMultiplier":0.001,
         "num": 10
      }
   }
])
```

範例 2

在此範例中，您將了解如何尋找特定地址 （點） 2 公里內的所有餐廳 （點）。若要執行此類查詢，您可以在 $maxDistance GeoJSON 點的最小值$minDistance和最大值$nearSphere內使用

```
db.usarestaurants.find({
   "location":{
      "$nearSphere":{
         "$geometry":{
            "type":"Point",
            "coordinates":[
               -122.3516,
               47.6156
            ]
         },
         "$minDistance":1,
         "$maxDistance":2000
      }
   }
},
{
   "name":1
})
```

上述命令會傳回距離指定點 2 公里以內的餐廳。此命令的輸出看起來會像這樣

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "name" : "Noodle House" }
```

限制

Amazon DocumentDB 不支援查詢或索引多邊
形、LineString、MultiPoint、MultiPolygon、MultiLineString 和 GeometryCollection。

部分索引

部分索引會為集合中符合指定篩選條件的文件編製索引。Amazon DocumentDB 5.0 執行個體型叢集支援部分索引功能。

主題

- 建立部分索引

- 支援的運算子
- 使用部分索引查詢
- 部分索引功能
- 部分索引限制

建立部分索引

若要建立部分索引，請使用 `createIndex()`方法搭配 `partialFilterExpression`選項。例如，下列操作會在訂單集合中建立唯一的複合索引，為具有 `OrderID`且 `isDelivered` 欄位為 true 的文件編製索引：

```
db.orders.createIndex(
  {"category": 1, "CustomerId": 1, "OrderId": 1},
  {"unique": true, "partialFilterExpression":
    {"$and": [
      {"OrderId": {"$exists": true}},
      {"isDelivered": {"$eq": false}}
    ]}
  }
)
```

支援的運算子

- $eq
- $exists
- $and（僅限最上層）
- $gt/$gte/$lt/$lte（只有當查詢中述詞的篩選條件完全符合部分篩選條件表達式時，才會使用索引掃描）（請參閱限制）

使用部分索引查詢

下列查詢模式可以使用部分索引：

- 查詢述詞完全符合部分索引篩選條件表達式：

```
db.orders.find({"$and": [
```

```
    {"OrderId": {"$exists": true}},
    {"isDelivered": {"$eq": false}}
  ]}).explain()
```

- 查詢篩選條件的預期結果是部分篩選條件的邏輯子集：

```
db.orders.find({"$and": [
    {"OrderId": {"$exists": true}},
    {"isDelivered": {"$eq": false}},
    {"OrderAmount": {"$eq": "5"}}
  ]}).explain()
```

- 查詢的子述詞可以與其他索引搭配使用：

```
db.orders.createIndex({"anotherIndex":1})
db.orders.find({ "$or": [
      {"$and": [
        {"OrderId": {"$exists": true}},
        {"isDelivered": {"$eq": false}}
      ]},
      {"anotherIndex": {"$eq": 5}}
    ]
  }).explain()
```

> **Note**
>
> 如果這麼做有效率，查詢規劃器可能會選擇使用集合掃描，而不是索引掃描。對於會傳回大部分集合的非常小型集合或查詢，通常會看到這種情況。

部分索引功能

列出部分索引

使用 `getIndex`操作列出具有 partialFilterExpression 的部分索引。例如， 中發出的`getIndex`操作會列出具有索引鍵、名稱和 partialfilterExpressions 欄位的部分索引：

```
db.orders.getIndexes()
```

此範例會傳回下列輸出：

```
[
  {
    "v" : 4,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "ecommerceApp.orders"
  },
  {
    "v" : 4,
    "unique" : true,
    "key" : {
      "category" : 1,
      "" : 1,
      "CustomerId" : 1,
      "OrderId" : 1
    },
    "name" : "category_1_CustID_1_OrderId_1",
    "ns" : "ecommerceApp.orders",
    "partialFilterExpression" : {
      "$and" : [
        {"OrderId": {"$exists": true}},
        {"isDelivered": {"$eq": false}}
      ]
    }
  }
]
```

相同 key：order 上的多個部分篩選條件表達式

您可以為相同的欄位組合 (key：order) 建立不同的部分索引。這些索引必須具有不同的名稱。

```
db.orders.createIndex(
  {"OrderId":1},
  {
    name:"firstPartialIndex",
    partialFilterExpression:{"OrderId":{"$exists": true}}
  }
)
```

```
db.orders.createIndex(
```

```
  {"OrderId":1},
  {
    name:"secondPartialIndex",
    partialFilterExpression:{"OrderId":{"$gt": 1000}}
  }
)
```

執行 getIndexes操作以列出集合中的所有索引：

```
db.orders.getIndexes()
```

這些範例會傳回下列輸出：

```
[
  {
    "v" : 4,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "ecommerceApp.orders"
  },
  {
    "v" : 4,
    "key" : {
      "OrderId" : 1
    },
    "name" : "firstPartialIndex",
    "ns" : "ecommerceApp.orders",
    "partialFilterExpression" : {"OrderId":{"$exists": true}}
  },
  {
    "v" : 4,
    "key" : {
      "OrderId" : 1
    },
    "name" : "secondPartialIndex",
    "ns" : "ecommerceApp.orders",
    "partialFilterExpression" : {"OrderId":{"$gt": 1000}}
  }
]
```

> **Important**
>
> 索引名稱必須不同，且只能依名稱刪除。

具有部分和 TTL 屬性的索引

您也可以在建立索引期間同時指定 partialFilterExpression 和 expireAfterSeconds選項，以建立具有部分和 TTL 屬性的索引。這可讓您進一步控制哪些文件現在已從集合中移除。

例如，您可能有一個 TTL 索引，可識別在特定時段後要刪除的文件。您現在可以提供使用部分索引選項刪除文件時的額外條件：

```
db.orders.createIndex(
    { "OrderTimestamp": 1 },
    {
        expireAfterSeconds: 3600 ,
        partialFilterExpression: { "isDelivered": { $eq: true } }
    }
)
```

此範例會傳回下列輸出：

```
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1,
        "operationTime" : Timestamp(1234567890, 1)
}
```

執行 getIndexes操作以列出集合中存在的索引：

```
db.orders.getIndexes()
[
    {
        "v" : 4,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.orders"
```

```
    }
```

此範例會傳回下列輸出：

```
[
    {
        "v": 4,
        "key": {
            "_id": 1
        },
        "name": "_id_",
        "ns": "ecommerceApp.orders"
    },
    {
        "v": 4,
        "key": {
            "OrderTimestamp": 1
        },
        "name": "OrderTimestamp_1",
        "ns": "ecommerceApp.orders",
        "partialFilterExpression": {
            "isDelivered": {
                "$eq": true
            }
        },
        "expireAfterSeconds": 3600
    }
]
```

部分索引限制

下列限制適用於部分索引功能：

- 只有在查詢篩選條件述詞完全符合 `partialFilterExpression`且具有相同資料類型時，Amazon DocumentDB 中的不等式查詢才會使用部分索引。

  > **Note**
  >
  > 甚至`$hint`無法用於強制上述案例使用 IXSCAN。

  在下列範例中， `partialFilterExpression` 只會套用至 ，`field1`但不會套用至 `field2`：

```
db.orders.createIndex(
  {"OrderAmount": 1},
  {"partialFilterExpression": { OrderAmount : {"$gt" : 5}}}
)

db.orders.find({OrderAmount : {"$gt" : 5}}) // Will use partial index
db.orders.find({OrderAmount : {"$gt" : 6}}) // Will not use partial index
db.orders.find({OrderAmount : {"$gt" : Decimal128(5.00)}}) // Will not use partial
 index
```

- 不支援`partialFilterExpression`具有陣列運算子的 。下列操作將產生錯誤：

```
db.orders.createIndex(
  {"CustomerId":1},
  {'partialFilterExpression': {'OrderId': {'$eq': [1000, 1001, 1002]}}}
)
```

- partialFilterExpression 欄位不支援下列運算子：
  - `$all` （陣列運算子）
  - `$mod` （陣列運算子）
  - `$or`
  - `$xor`
  - `$not`
  - `$nor`
- 篩選條件表達式和篩選條件的資料類型應相同。

使用 Amazon DocumentDB 執行文字搜尋

Amazon DocumentDB 的原生全文搜尋功能（文字索引 v1) 可讓您使用特殊用途文字索引，對大型文字資料集執行文字搜尋。本節說明文字索引功能，並提供如何在 Amazon DocumentDB 中建立和使用文字索引的步驟。也會列出文字搜尋限制。

主題

- 支援的功能
- 使用 Amazon DocumentDB 文字索引
- MongoDB 的差異

- 最佳實務和指導方針
- 文字索引 V2
- 限制

支援的功能

Amazon DocumentDB 文字搜尋支援下列 MongoDB API 相容功能：

- 在單一欄位上建立文字索引。
- 建立包含多個文字欄位的複合文字索引。
- 執行單字或多字搜尋。
- 使用權重控制搜尋結果。
- 依分數排序搜尋結果。
- 在彙總管道中使用文字索引。
- 搜尋確切片語。

使用 Amazon DocumentDB 文字索引

若要在包含字串資料的欄位上建立文字索引，請指定字串「文字」，如下所示：

單一欄位索引：

```
db.test.createIndex({"comments": "text"})
```

此索引支援指定集合中「註解」字串欄位中的文字搜尋查詢。

在多個字串欄位上建立複合文字索引：

```
db.test.createIndex({"comments": "text", "title":"text"})
```

此索引支援指定集合中「註解」和「標題」字串欄位中的文字搜尋查詢。建立複合文字索引時，您最多可以指定 30 個欄位。建立之後，您的文字搜尋查詢將查詢所有索引欄位。

> **Note**
>
> 每個集合只允許一個文字索引。

在 Amazon DocumentDB 集合上列出文字索引

您可以在集合`getIndexes()`上使用 來識別和描述索引，包括文字索引，如下列範例所示：

```
rs0:PRIMARY> db.test.getIndexes()
[
   {
      "v" : 4,
      "key" : {
         "_id" : 1
      },
      "name" : "_id_",
      "ns" : "test.test"
   },
   {
      "v" : 1,
      "key" : {
         "_fts" : "text",
         "_ftsx" : 1
      },
      "name" : "contents_text",
      "ns" : "test.test",
      "default_language" : "english",
      "weights" : {
         "comments" : 1
      },
      "textIndexVersion" : 1
   }
]
```

建立索引後，開始將資料插入 Amazon DocumentDB 集合。

```
db.test.insertMany([{"_id": 1, "star_rating": 4, "comments": "apple is red"},
                    {"_id": 2, "star_rating": 5, "comments": "pie is delicious"},
                    {"_id": 3, "star_rating": 3, "comments": "apples, oranges - healthy
 fruit"},
                    {"_id": 4, "star_rating": 2, "comments": "bake the apple pie in the
 oven"},
                    {"_id": 5, "star_rating": 5, "comments": "interesting couch"},
                    {"_id": 6, "star_rating": 5, "comments": "interested in couch for
 sale, year 2022"}])
```

執行文字搜尋查詢

執行單字文字搜尋查詢

您需要使用 $text和 $search運算子來執行文字搜尋。下列範例會傳回文字索引欄位包含字串「apple」或「apple」的其他格式的所有文件，例如「apples」：

```
db.test.find({$text: {$search: "apple"}})
```

輸出：

此命令的輸出看起來像這樣：

```
{ "_id" : 1, "star_rating" : 4, "comments" : "apple is red" }
{ "_id" : 3, "star_rating" : 3, "comments" : "apples, oranges - healthy fruit" }
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven" }
```

執行多字文字搜尋

您也可以對 Amazon DocumentDB 資料執行多字文字搜尋。以下命令會傳回具有文字索引欄位的文件，其中包含「apple」或「pie」：

```
db.test.find({$text: {$search: "apple pie"}})
```

輸出：

此命令的輸出看起來像這樣：

```
{ "_id" : 1, "star_rating" : 4, "comments" : "apple is red" }
{ "_id" : 2, "star_rating" : 5, "comments" : "pie is delicious" }
{ "_id" : 3, "star_rating" : 3, "comments" : "apples, oranges - healthy fruit" }
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven" }
```

執行多字片語文字搜尋

對於多字片語搜尋，請使用此範例：

```
db.test.find({$text: {$search: "\"apple pie\""}})
```

輸出：

上述命令會傳回具有文字索引欄位的文件，其中包含確切的片語「apple pie」。此命令的輸出看起來像這樣：

```
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven" }
```

使用篩選條件執行文字搜尋

您也可以將文字搜尋與其他查詢運算子合併，以根據其他條件篩選結果：

```
db.test.find({$and: [{star_rating: 5}, {$text: {$search: "interest"}}]})
```

輸出：

上述命令會傳回具有文字索引欄位的文件，其中包含任何形式的「興趣」和等於 5 的「star_rating」。此命令的輸出看起來像這樣：

```
{ "_id" : 5, "star_rating" : 5, "comments" : "interesting couch" }
{ "_id" : 6, "star_rating" : 5, "comments" : "interested in couch for sale, year
 2022" }
```

限制文字搜尋中傳回的文件數量

您可以選擇使用 來限制傳回的文件數量`limit`：

```
db.test.find({$and: [{star_rating: 5}, {$text: {$search: "couch"}}]}).limit(1)
```

輸出：

上述命令會傳回一個滿足篩選條件的結果：

```
{ "_id" : 5, "star_rating" : 5, "comments" : "interesting couch" }
```

依文字分數排序結果

下列範例會依文字分數排序文字搜尋結果：

```
db.test.find({$text: {$search: "apple"}}, {score: {$meta: "textScore"}}).sort({score:
 {$meta: "textScore"}})
```

輸出：

上述命令會傳回具有文字索引欄位的文件，其中包含「apple」或「apple」的其他格式，例如「apples」，並根據文件與搜尋詞彙的相關性來排序結果。此命令的輸出看起來像這樣：

```
{ "_id" : 1, "star_rating" : 4, "comments" : "apple is red", "score" :
 0.6079270860936958 }
{ "_id" : 3, "star_rating" : 3, "comments" : "apples, oranges - healthy fruit",
 "score" : 0.6079270860936958 }
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven",
 "score" : 0.6079270860936958 }
```

$text 、aggregate、count、 和 delete命令$search也支援 findAndModifyupdate和 。

彙總運算子

使用 彙總管道 **$match**

```
db.test.aggregate(
   [{ $match: { $text: { $search: "apple pie" } } }]
)
```

輸出：

上述命令會傳回下列結果：

```
{ "_id" : 1, "star_rating" : 4, "comments" : "apple is red" }
{ "_id" : 3, "star_rating" : 3, "comments" : "apples, oranges - healthy fruit" }
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven" }
{ "_id" : 2, "star_rating" : 5, "comments" : "pie is delicious" }
```

其他彙總運算子的組合

```
db.test.aggregate(
   [
      { $match: { $text: { $search: "apple pie" } } },
      { $sort: { score: { $meta: "textScore" } } },
      { $project: { score: { $meta: "textScore" } } }
   ]
)
```

輸出：

上述命令會傳回下列結果：

```
{ "_id" : 4, "score" : 0.6079270860936958 }
{ "_id" : 1, "score" : 0.3039635430468479 }
{ "_id" : 2, "score" : 0.3039635430468479 }
{ "_id" : 3, "score" : 0.3039635430468479 }
```

建立文字索引時指定多個欄位

您可以在複合文字索引中將權重指派給最多三個欄位。指派給文字索引中欄位的預設權重為一 (1)。權重是選用參數，範圍必須介於 1 到 100000 之間。

```
db.test.createIndex(
   {
     "firstname": "text",
     "lastname": "text",
     ...
   },
   {
     weights: {
       "firstname": 5,
       "lastname":10,
       ...
     },
     name: "name_text_index"
   }
 )
```

MongoDB 的差異

Amazon DocumentDB 的文字索引功能使用反向索引搭配術語頻率演算法。根據預設，文字索引是稀疏的。由於剖析邏輯、字符化分隔符號和其他項目的差異，可能無法針對相同的資料集或查詢形狀傳回與 MongoDB 相同的結果集。

Amazon DocumentDB 文字索引與 MongoDB 之間存在下列其他差異：

- 不支援使用非文字索引的複合索引。
- Amazon DocumentDB 文字索引不區分大小寫。

- 文字索引僅支援英文。
- 不支援陣列（或多重索引鍵）欄位的文字索引。例如，使用文件 {「a」：【「apple」、「pie」】} 在「a」上建立文字索引將會失敗。
- 不支援萬用字元文字索引。
- 不支援唯一文字索引。
- 不支援排除字詞。

最佳實務和指導方針

- 若要在涉及依文字分數排序的文字搜尋查詢上獲得最佳效能，請在載入資料之前建立文字索引。
- 文字索引需要額外的儲存體，才能最佳化索引資料的內部複本。這會產生額外的成本影響。

文字索引 V2

Amazon DocumentDB 8.0 推出新版本的文字索引 (V2)，可變更基礎文字搜尋剖析器，以提供更多與 MongoDB 的相容性。

除了 V1 文字索引提供的功能之外，V2 文字索引還提供下列支援：

- 規劃器會盡可能提早在管道中移動 $match 階段，減少後續階段處理的文件數量。
- 改善文字欄位中特殊字元的字符化，例如電子郵件、URLs和檔案路徑。V2 可以剖析和比對這些字串中的個別字符，而 V1 無法。

```
rs0:PRIMARY> db.coll.createIndex({ "a": "text" });
rs0:PRIMARY> db.coll.find()
{ "_id" : 1, "a" : "jane.doe_1234@company.com" }
{ "_id" : 2, "a" : "janedoe@company.com" }
{ "_id" : 3, "a" : "/home/user/company/thesis.pdf" }
{ "_id" : 4, "a" : "/home/user/path/jane.pdf" }
{ "_id" : 5, "a" : "http://www.company.com/path" }
{ "_id" : 6, "a" : "https://company.com/path/../home" }

// Sample query
rs0:PRIMARY> db.coll.find({ $text: { $search: "jane" } });

// V1 text index – no results
None
```

```
// V2 text index – matches tokens within emails and file paths
{ "_id" : 1, "a" : "jane.doe_1234@company.com" }
{ "_id" : 4, "a" : "/home/user/path/jane.pdf" }
```

限制

文字搜尋在 Amazon DocumentDB 中具有下列限制：

- 文字索引存放 lexemes 及其位置資訊。在單一文件中，所有 lexemes 及其位置資訊的合併大小限制為 1MB。

索引

索引可讓 Amazon DocumentDB 快速尋找文件，而無需掃描集合中的每個文件，從而改善查詢效能。您可以在應用程式經常查詢的欄位上建立索引，以減少延遲和 I/O 成本。

如需其他索引相關主題，請參閱：

- 管理 Amazon DocumentDB 索引 — 建立和建置索引
- 使用索引 — 為最佳實務編製索引
- 索引故障診斷 — 解決索引建置失敗和膨脹
- 如何分析索引用量並識別未使用的索引？ — 尋找未使用和遺失的索引

主題

- 索引管理
- 索引類型
- 索引屬性

索引管理

本節涵蓋 Amazon DocumentDB 的索引管理主題。

主題

- 建立索引

- 維護索引
- 使用索引的最佳實務

建立索引

在 Amazon DocumentDB 中建置索引需要做出許多決策：

- 需要多快完成？
- 當組建發生時，集合是否無法存取？
- 可以配置多少執行個體的運算能力給組建？
- 應該建立哪種類型的索引？

本節可協助您回答這些問題，並提供命令和監控範例，以在執行個體型叢集集合上建立 Amazon DocumentDB 索引。

指導方針

下列準則包括在建立新索引時的基本限制和組態權衡：

- Amazon DocumentDB 版本支援 - 雖然所有 Amazon DocumentDB 版本都支援單一工作者索引，但僅在 Amazon DocumentDB 4.0 和 5.0 版上支援多個工作者索引。
- 效能權衡 - 在索引建立程序中增加工作者數量會增加 CPU 使用率，並在 Amazon DocumentDB 資料庫的主要執行個體上讀取 IO。建立新索引所需的資源將不適用於您執行中的工作負載。
- 彈性叢集 - Amazon DocumentDB 彈性叢集不支援平行索引。
- 無伺服器叢集 - 無伺服器叢集上的平行索引需要至少 32 個 DCUs的最小容量。低於此閾值時，系統會接受 `workers` 參數，但索引建置會使用單一工作者執行。平行工作者的最大數量會隨著您設定的最小容量而擴展：大約`min_DCU / 8`有工作者可用。例如，在 32 DCUs 中，您最多可以使用 4 個工作者，而在 64 DCUs 中，您最多可以使用 8 個工作者。

  > **Note**
  >
  > 變更新的平行工作者限制以生效後，需要重新啟動叢集。

- 工作者上限（以執行個體為基礎） - 您可以設定的工作者數目上限取決於資料庫叢集中主要執行個體的大小。這是資料庫叢集主要執行個體上 vCPUs總數的一半。例如，您可以在具有 64 個 vCPUs 的 db.r6g.16xlarge 執行個體上執行最多 32 個工作者。

> **Note**
>
> 2xlarge 以上的執行個體支援平行索引建置。

- 最低工作者 - 您可以設定的工作者數量下限為 1。在執行個體型叢集上建立索引的預設設定為兩個工作者。不過，您可以使用「工作者執行緒」選項，將工作者數量減少為一個。這將使用單一工作者執行程序。
- 索引壓縮 - Amazon DocumentDB 不支援索引壓縮。索引的資料大小可能大於您使用其他選項時的大小。
- 索引多個集合 - 資料庫叢集主要執行個體上的一半 vCPUs 可用於在多個集合上執行索引建立的設定工作者。
- 索引類型 - 索引類型 如需 Amazon DocumentDB 上支援的索引類型清單，請參閱 。

索引建置類型

索引建置的四種類型為：

- 前景：前景索引建置會封鎖建立索引之集合上的所有其他資料庫操作，直到索引建立完成為止。Amazon DocumentDB 前景建置由五個階段組成。
- 前景（唯一）：單一文件（唯一）前景索引建置會封鎖其他資料庫操作，例如一般前景建置。與基本前景建置不同，唯一建置會使用額外的階段（排序金鑰 2) 來尋找重複的金鑰。前景（唯一）建置由六個階段組成。
- 背景：背景索引建置允許在建立索引時，在前景中執行其他資料庫操作。Amazon DocumentDB 背景建置由八個階段組成。
- 背景（唯一）：單一文件（唯一）背景索引建置可讓其他資料庫操作在建立索引時於前景執行。與基本背景建置不同，唯一建置會使用額外的階段（排序金鑰 2) 來尋找重複的金鑰。背景（唯一）建置由九個階段組成。

預設索引建置類型為前景。

若要在背景中建置索引，請在建立索引時新增「背景」： true 參數。

開始使用

若要開始在集合上建立索引，請使用 createIndexes 命令。根據預設，命令會執行兩個平行工作者，將索引建立程序的速度增加兩次。

例如，下列命令程序示範如何在文件中為 "user_name" 欄位建立索引，並將索引程序速度提高到四名工作者：

1. 在叢集上使用兩個平行工作者在背景建立索引：

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx", "background":true }]})
```

2. 若要最佳化索引建立程序的速度，您可以使用 db.runCommand createIndexes 命令中的「工作者執行緒」選項 (「工作者」：<number>) 來指定工作者數量。

   將程序的速度提高到四個平行工作者：

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx", "workers":4}]})
```

> **Note**
>
> 工作者數量越高，建立索引的速度就越快。不過，工作者數量越高，主要執行個體vCPUs 和讀取 IO 的負載越高。確保您的叢集已充分佈建，以處理增加的負擔，而不會降低其他工作負載。

索引進度狀態

索引建立程序的運作方式是初始化、掃描集合、排序索引鍵，最後則是透過索引建置器插入索引鍵。當您在前景執行時，此程序最多有六個階段，而在背景執行時最多有九個階段。您可以依階段檢視狀態指標，例如完成百分比、掃描的儲存區塊總數、已排序的金鑰和插入的金鑰。

使用 mongo shell 中的 db.currentOp() 命令來監控索引程序的進度。完成最後一個階段的 100% 顯示已成功建立所有索引：

```
db.currentOp({"command.createIndexes": { $exists : true } })
```

> **Note**
>
> 只有 Amazon DocumentDB 5.0 才支援檢視索引進度狀態。

索引建置階段

階段	前景	前景（唯一）	背景介紹	背景（唯一）
初始化	1	1	1	1
建置索引：初始化	2	2	2	2
建置索引：掃描集合	3	3	3	3
建置索引：排序索引鍵 1	4	4	4	4
建置索引：排序索引鍵 2		5		5
建置索引：插入索引鍵	5	6	5	6
驗證：掃描索引			6	7
驗證：排序元組			7	8
驗證：掃描集合			8	9

- 初始化 - createIndex 正在準備索引建置器。此階段應該非常簡短。
- 建置索引：初始化 - 索引建置器正在準備建立索引。此階段應該非常簡短。
- 建置索引：掃描集合 - 索引建置器正在執行集合掃描以收集索引鍵。度量單位為「區塊」。

> **Note**
>
> 如果針對索引建置設定了多個工作者，則會顯示在此階段。「掃描集合」階段是唯一在索引建置過程中使用多個工作者的階段。所有其他階段都會顯示單一工作者。

- 建置索引：排序索引鍵 1 - 索引建置器正在排序收集的索引鍵。度量單位為「金鑰」。
- 建置索引：排序索引鍵 2 - 索引建置器正在排序與無效元組對應的收集索引鍵。此階段僅適用於唯一的索引建置。度量單位為「金鑰」。
- 建置索引：插入索引鍵 - 索引建置器正在將索引鍵插入新索引。度量單位為「金鑰」。
- 驗證：掃描索引 - createIndex 正在掃描索引，以尋找需要驗證的金鑰。度量單位為「區塊」。
- 驗證：排序元組 - createIndex 正在排序索引掃描階段的輸出。
- 驗證：掃描集合 - createIndex 正在掃描集合，以驗證前兩個階段中找到的索引鍵。度量單位為「區塊」。

索引建置輸出範例

在下列輸出範例中（前景索引建置），會顯示索引建立的狀態。"msg" 欄位會指出建置的階段和完成百分比，以摘要說明建置進度。「工作者」欄位指出索引建置的該階段期間所使用的工作者數量。「進度」欄位會顯示用於計算完成百分比的實際數字。

> **Note**
>
> Amazon DocumentDB 4.0 版不支援 "currentIndexBuildName"、"msg" 和 "progress" 欄位。

```
{
    "inprog" : [{
    …
        "command": {
            "createIndexes": "test",
            "indexes": [{
                "v": 2,
                "key": {
                    "user_name": 1
                },
                "name": "user_name_1"
```

```
            }],
            "lsid": {
                "id": UUID("094d0fba-8f41-4373-82c3-7c4c7b5ff13b")
            },
            "$db": "test"
        },
        "currentIndexBuildName": user_name_1,
        "msg": "Index Build: building index number_1, stage 6/6 building index:
 656860/1003520 (keys) 65%",
        "workers": 1,
        "progress": {
            "done": 656861,
            "total": 1003520
        },
    …
    ],

    "ok" : 1
}
```

疑難排解資源

請參閱下列有關對索引建立問題進行疑難排解的資源：

- 索引建置失敗
- 背景索引建置延遲問題和失敗

維護索引

索引膨脹

Amazon DocumentDB 使用多版本並行控制 (MVCC) 來管理並行交易。刪除或更新文件時，其先前的版本會保留在集合和索引中為「無效」版本。垃圾收集程序會自動從這些無效版本回收空間，以供未來操作使用。

當集合的索引因為頁面中累積無效或過時的索引項目或分段而變大時，就會發生索引膨脹。報告的百分比代表未來索引項目可以使用的索引空間量。此膨脹會耗用緩衝區快取和儲存體中的空間。如果您想要移除膨脹，則需要重建索引。

範例：執行下列命令來判斷索引的未使用儲存：

```
db.coll.aggregate({$indexStats:{}});
```

結果如下所示：

```
{
    "name" : "_id_",
    "key" : {
        "_id" : 1
    },
    "host" : "devbox-test.localhost.a2z.com:27317",
    "size" : NumberLong(827392),
    "accesses" : {
        "ops" : NumberLong(40000),
        "docsRead" : NumberLong(46049),
        "since" : ISODate("2025-04-03T21:44:51.251Z")
    },
    "cacheStats" : {
        "blksRead" : NumberLong(264),
        "blksHit" : NumberLong(140190),
        "hitRatio" : 99.8121
    },
    "unusedStorageSize" : {
        "unusedSizeBytes" : 409600,
        "unusedSizePercent" : 49.51
    }
}
```

您可以使用 reIndex 命令重建索引，無需停機，這需要掃描整個集合。請參閱使用 reIndex 進行索引維護。

使用 reIndex 進行索引維護

reIndex 是用來重建索引的命令。它通常在索引已損毀或效率不佳時使用。隨著時間的推移，索引可能會因為許多更新、插入或刪除而累積未使用的空間，導致效能降低。重新索引有助於移除這類未使用的空間，並還原索引的效率。

reIndex 準則

- 僅 Amazon DocumentDB 5.0 及更高版本支援 reIndex
- reIndex 一律會在背景執行。
- Amazon DocumentDB 支援在背景中重新索引單一索引，允許多個工作者。執行 reIndex 程序時，查詢可以使用舊索引。
- Amazon DocumentDB 支援透過 currentOp 編製進度報告的索引。您可以看到索引建置階段，類似於在索引建立期間檢視的索引建置階段。唯一的差別是 reIndex 一律有八個階段，無論其是否為唯一。沒有「建置索引：排序索引鍵 2」階段。
- reIndex 可以與相同集合上的索引相關命令以外的任何命令同時執行：createIndexes、dropIndexes、collMod 和 renameCollection。
- 文字、地理空間、向量和部分索引目前不支援 reIndex。

reIndex 組建

使用下列命令來重建索引：

```
db.runCommand({ reIndex: "collection-name", index: "index-name"})
```

您也可以選擇性地控制指派給重建程序的工作者數量：

```
db.runCommand({ reIndex: "collection-name", index: "index-name", workers: number })
```

如需使用 Java 管理索引的特定資訊，請參閱 使用 Java 的 Amazon DocumentDB 中的索引管理。

使用索引的最佳實務

建置索引

將資料匯入 Amazon DocumentDB 時，您應該先建立索引，再匯入大型資料集。您可以使用 Amazon DocumentDB Index Tool 從執行中的 MongoDB 執行個體或 mongodump 目錄擷取索引，並在 Amazon DocumentDB 叢集中建立這些索引。如需遷移的詳細指導方針，請參閱遷移和升級。

索引選擇性

將索引建立限制為欄位，其中重複值的數量小於集合中文件總數的 1%。例如，如果您的集合包含 100，000 個文件，則只會在相同值發生 1，000 次或更少的欄位上建立索引。

選擇具有大量唯一值（即高基數）的索引可確保篩選操作傳回少量文件，從而在索引掃描期間產生良好的效能。高基數索引範例像是唯一索引，它可以保證相等述詞最多只能傳回單一文件。低基數索引的範例像是布林欄位，以及對一週中某天的索引。由於效能不佳，資料庫的查詢最佳化工具不太可能選擇低基數索引。同時，低基數索引會繼續消耗磁碟空間和 I/O 等資源。根據經驗法則，您應該在典型值頻率為總集合大小的 1% 或更小的字段上鎖定索引。

此外，建議只在通常用作篩選器的欄位上建立索引，並定期尋找未使用的索引。如需詳細資訊，請參閱the section called “分析索引用量並識別未使用的索引”。

索引對寫入資料的影響

雖然索引可以透過避免掃描集合中的每個文件來改善查詢性能，但此改進方式有所權衡。對於集合上的每個索引，每次插入、更新或刪除文件時，資料庫必須更新集合並將欄位寫入集合的每個索引。例如，如果集合有九個索引，資料庫必須先執行十次寫入，才能確認用戶端的作業。因此，每個額外的索引都會產生額外的寫入延遲、I/O，並增加整體使用的儲存。

叢集執行個體的大小必須適當，才能保留所有工作集記憶體。如此可避免從儲存磁碟區持續讀取索引頁面的需求，進而對效能造成負面影響，並產生較高的 I/O 成本。如需詳細資訊，請參閱執行個體大小。

為了獲得最佳效能，請盡量減少集合中的索引數目，只新增必要的索引，以改善常見查詢的效能。雖然工作負載有所不同，但好的指導方針是將每個集合的索引數目保持在五個或更少。

識別缺少的索引

最佳實務是定期識別缺少的索引。如需詳細資訊，請參閱the section called “識別遺漏的索引”。

識別未使用的索引

最佳實務是定期識別和移除未使用的索引。如需詳細資訊，請參閱the section called “分析索引用量並識別未使用的索引”。

索引類型

主題

- 單一欄位索引
- 複合索引
- 多金鑰索引
- 文字索引
- 地理空間索引
- 向量索引

單一欄位索引

單一欄位索引會儲存文件集合中任何單一欄位的資訊。根據預設，所有集合在 _id 欄位上都有索引。您可以新增其他索引，以加速重要的查詢和操作。

下列情況下，單一欄位索引很有幫助：

- 您的應用程式經常根據特定欄位的值查詢或篩選文件。
- 您需要依特定欄位有效率地排序文件。
- 您想要透過建立唯一索引，確保集合中文件間特定欄位的唯一性。

支援的索引屬性

選項	3.6	4.0	5.0	8.0	彈性叢集
name	是	是	是	是	是
唯一	是	是	是	是	是
稀疏 *	是	是	是	是	是
partialFilterExpression *	否	否	是	是	否
expireAfterSeconds	是	是	是	是	是

* `sparse`和 `partialFilterExpression`選項無法在相同的索引定義中一起使用。如果您嘗試使用這些選項建立索引，則會失敗，並顯示下列錯誤：

```
Error in specification: cannot mix partialFilterExpression and sparse options
```

建立單一欄位索引

使用 `createIndex()`方法來建立單一欄位索引。語法為：`db.collection.createIndex(<key>, <options>)`。

key 參數是 JSON 文件，指定 欄位和索引排序順序：

```
{
  "<field>": <1 (ascending)|-1 (descending)>
}
```

options 參數是 JSON 文件，可指定索引的選項：

```
{
  "name": "<name>",
  "unique": <true | false>,
  "sparse": <true | false>,
  "partialFilterExpression": <filter expression>,
  "expireAfterSeconds": <seconds before expiry>
}
```

如需建立單一欄位索引的範例，請參閱索引屬性。

複合索引

複合索引會存放文件集合中兩個或多個欄位的資訊，允許查詢第一個欄位或任何字首欄位。這些索引會針對同時篩選多個欄位或結合篩選與排序操作的查詢最佳化效能。它們也適用於最左邊索引欄位上的單一條件查詢。資料庫會利用這些索引項目，有效率地尋找相符的文件，而無需執行完整的集合掃描。

下列情況下，複合欄位索引很有幫助：

- 您需要同時篩選多個欄位。
- 您需要結合篩選與排序操作。

支援的索引屬性

選項	3.6	4.0	5.0	8.0	彈性叢集
name	是	是	是	是	是
唯一	是	是	是	是	是
稀疏 *	是	是	是	是	是
partialFilterExpression *	否	否	是	是	否

* sparse和 partialFilterExpression選項無法在相同的索引定義中一起使用。如果您嘗試使用這些選項建立索引，則會失敗，並顯示下列錯誤：

```
Error in specification: cannot mix partialFilterExpression and sparse options
```

建立複合索引

使用 createIndex()方法來建立複合索引。方法語法為：
db.collection.createIndex(<keys>, <options>)

keys 參數是指定欄位和索引排序順序的 JSON 文件：

```
{
  "<field 1>": <1 (ascending)|-1 (descending)>,
  "<field 2>": <1 (ascending)|-1 (descending)>,
  ...
}
```

請注意，複合索引中只有一個欄位可以是陣列。如果您嘗試在兩個或多個陣列欄位上建立複合索引，則會失敗，並顯示下列錯誤：

```
multiple fields of compound index cannot be arrays
```

options 參數是 JSON 文件，可指定索引的選項：

```
{
  "name": "<name>",
  "unique": <true | false>,
  "sparse": <true | false>,
  "partialFilterExpression": <filter expression>
}
```

如需建立複合索引的範例，請參閱索引屬性。

多金鑰索引

對於具有陣列值的欄位，多索引鍵索引可讓您為陣列中的每個元素建立索引鍵。索引陣列會為陣列的每個元素建立索引項目。

當您的應用程式經常根據陣列中的值查詢或篩選文件時，多金鑰索引很有幫助。

支援的索引屬性

選項	3.6	4.0	5.0	8.0	彈性叢集
name	是	是	是	是	是
唯一	是	是	是	是	是
稀疏 *	是	是	是	是	是
partialFilterExpression *	否	否	是	是	否
expireAfterSeconds	是	是	是	是	是

* `sparse`和 `partialFilterExpression`選項無法在相同的索引定義中一起使用。如果您嘗試使用這些選項建立索引，則會失敗，並顯示下列錯誤：

```
Error in specification: cannot mix partialFilterExpression and sparse options
```

建立多金鑰索引

使用 `createIndex()`方法來建立多金鑰索引。方法語法為：
`db.collection.createIndex(<key>, <options>)`

key 參數是 JSON 文件，指定 欄位和索引排序順序：

```
{
  "<field>": <1 (ascending)|-1 (descending)>
}
```

options 參數是 JSON 文件，可指定索引的選項：

```
{
  "name": "<name>",
  "unique": <true | false>,
  "sparse": <true | false>,
```

```
  "partialFilterExpression": <filter expression>,
  "expireAfterSeconds": <seconds before expiry>
}
```

下列範例會在categories欄位上建立多重索引鍵索引，並以名稱 遞增排序book_categories：

```
db.collection.createIndex(
  {
    "categories": 1
  },
  {
    "name": "book_categories"
  }
)
```

如需建立多金鑰索引的範例，請參閱索引屬性。

文字索引

在文件中的文字欄位中搜尋單字或片語時，文字索引非常有用，可讓您執行全文搜尋功能。您可以使用多個文字欄位建立單一欄位文字索引或複合文字索引。不過，每個集合只能有一個文字索引。如需詳細資訊，請參閱使用 Amazon DocumentDB 執行文字搜尋。

支援的索引屬性

選項	3.6	4.0	5.0	8.0	彈性叢集
name	否	否	是	是	否

建立文字索引

使用 createIndex()方法來建立文字索引。方法語法為：
db.collection.createIndex(<keys>, <options>)

keys 參數是 JSON 文件，指定 欄位 (s) 和文字索引類型：

```
{
  "<field 1>": "text",
  "<field 2>": "text"
```

```
}
```

options 參數是指定索引名稱的 JSON 文件：

```
{
  "name": "<name>"
}
```

如需建立文字索引的範例，請參閱索引屬性。

地理空間索引

地理空間索引是一種特殊類型的索引，旨在有效率地查詢和管理存放在文件集合中的地理空間資料。Amazon DocumentDB 支援 2dsphere 索引，專門設計用於處理球體（例如地球）上的地理空間資料。這允許根據球形幾何進行準確的計算和查詢。

當您的應用程式需要執行以位置為基礎的查詢時，地理空間索引會很有幫助，例如：

- 尋找附近的興趣點，
- 判斷位置是否位於特定區域內
- 計算位置之間的距離

支援的索引屬性

選項	3.6	4.0	5.0	8.0	彈性叢集
name	是	是	是	是	是

建立地理空間索引

使用 createIndex()方法來建立地理空間索引。方法語法為：
db.collection.createIndex(<key>, <options>)

key 參數是 JSON 文件，可指定 欄位和 2dsphere 索引類型：

```
{
  "<field>": "2dsphere"
```

```
}
```

options 參數是 JSON 文件，可指定索引的選項：

```
{
  "name": "<name>"
}
```

如需建立地理空間索引的範例，請參閱索引屬性。

向量索引

向量索引是一種特殊類型的索引，旨在有效率地查詢和管理存放在文件集合中的向量資料。Amazon DocumentDB 支援階層式可導覽小型世界 (HNSW) 和具有平面壓縮 (IVFFlat) 索引的反轉檔案。

如需詳細資訊，請參閱Amazon DocumentDB 的向量搜尋。

向量索引適用於機器學習和生成式 AI 使用案例，例如：

- 語義搜尋
- 產品建議
- 個人化
- 聊天機器人
- 詐騙偵測
- 異常偵測

支援的索引屬性

選項	3.6	4.0	5.0	8.0	彈性叢集
name	否	否	是	是	否

建立向量索引

使用 createIndex 命令搭配 runCommand()方法來建立向量索引。語法是：

```
db.runCommand({
```

```
  "createIndexes": "<collection>",
  "indexes": [{
    "key": {
      "<field>": "vector"
    },
    "name": "<name>",
    "vectorOptions": {
      "type": "<hnsw> | <ivfflat>",
      "dimensions": <number of dimensions>,
      "similarity": "<euclidean>|<cosine>|<dotProduct>",
      "lists": <number_of_lists> [applicable for IVFFlat],
      "m": <max number of connections> [applicable for HNSW],
      "efConstruction": <size of the dynamic list for index build> [applicable for
 HNSW]
    }
  }]
})
```

金鑰參數是指定欄位和向量索引類型的 JSON 文件：

```
{
  "<field>": "vector"
}
```

如需建立向量索引的範例，請參閱索引屬性。

索引屬性

索引屬性定義 Amazon DocumentDB 中索引的行為和特性。了解這些屬性可協助您設計符合應用程式特定需求的高效索引。

主題

- 索引屬性：名稱
- 索引屬性：唯一
- 索引屬性：sparse
- 索引屬性：partialFilterExpression
- 索引屬性： expireAfterSeconds
- 索引屬性：隱藏

索引屬性：名稱

支援的索引類型

選項	3.6	4.0	5.0	8.0	彈性叢集
單一欄位	是	是	是	是	是
compound	是	是	是	是	是
多金鑰	是	是	是	是	是
text	否	否	是	是	否
地理空間	是	是	是	是	是
向量	否	否	是	是	否

使用名稱選項來提供索引的選用名稱。

所有範例都使用下列範例文件：

```
{
  "productId": "PROD133726",
  "sku": "SKU24224",
  "name": "Basic Printer",
  "manufacturer": "The Manufacturer",
  "tags": [ "printer", "basic", "electronics", "business" ],
  "barcodes": [ "542364671", "886330670", "437445606" ],
  "reviews": [
    {
      "review_date": ISODate('2024-01-19T21:37:10.585Z'),
      ...
    }
  ],
  "material": "Polycarbonate",
  "color": "Space Gray",
  "supplier": {
    "supplierId": "SUP4",
    "location": {
      "type": "Point",
```

```
      "coordinates": [ -71.0589, 42.3601 ]
    }
  },
  "productEmbedding": [
    -0.019320633663838058,
    0.019672111388113596
  ],
  "lastUpdated": ISODate('2025-10-20T21:37:10.585Z')
}
```

單一欄位

```
db.collection.createIndex(
  {
    "productId": 1
  },
  {
    "name": "single_field_index"
  }
)
```

複合

```
db.collection.createIndex(
  {
    "productId": 1,
    "manufacturer": 1
  },
  {
    "name": "compound_index"
  }
)
```

多金鑰

```
db.collection.createIndex(
  {
    "tags": 1
  },
  {
    "name": "multikey_index"
  }
```

```
)
```

文字

```
db.collection.createIndex(
  {
    "name": "text"
  },
  {
    "name": "text_index"
  }
)
```

Geospatial (地理空間)

```
db.collection.createIndex(
  {
    "supplier.location": "2dsphere"
  },
  {
    "name": "geospatial_index"
  }
)
```

向量

```
db.runCommand({
  "createIndexes": "collection",
  "indexes": [{
    "key": {
      "productEmbedding": "vector"
    },
    "name": "hnsw_index",
    "vectorOptions": {
      "type": "hnsw",
      "dimensions": 2,
      "similarity": "euclidean",
      "m": 16,
      "efConstruction": 64
    }
  }]
})
```

索引屬性：唯一

支援的索引類型

選項	3.6	4.0	5.0	8.0	彈性叢集
單一欄位	是	是	是	是	是
compound	是	是	是	是	是
多金鑰	是	是	是	是	是

使用唯一選項來確保集合中文件間（欄位）的唯一性。

範例

下列範例示範如何在下列範例文件上建立唯一索引：

```
{
  "productId": "PROD133726",
  "sku": "SKU24224",
  "name": "Basic Printer",
  "manufacturer": "The Manufacturer",
  "tags": [ "printer", "basic", "electronics", "business" ],
  "barcodes": [ "542364671", "886330670", "437445606" ],
  "reviews": [
    {
      "review_date": ISODate('2024-01-19T21:37:10.585Z'),
      ...
    }
  ],
  "material": "Polycarbonate",
  "color": "Space Gray",
  "supplier": {
    "supplierId": "SUP4",
    "location": {
      "type": "Point",
      "coordinates": [ -71.0589, 42.3601 ]
    }
  },
  "productEmbedding": [
    -0.019320633663838058,
```

```
    0.019672111388113596
  ],
  "lastUpdated": ISODate('2025-10-20T21:37:10.585Z')
}
```

單一欄位

在 productId 上建立唯一的單一欄位索引，以確保相同的 productId 不存在於超過 1 個文件中：

```
db.collection.createIndex(
  {
    "productId": 1
  },
  {
    name: "productId_unique",
    unique: true
  }
)
```

複合

在 sku 和製造商上建立唯一的複合索引，以確保在超過 1 個文件中不存在相同的 sku 和製造商組合：

```
db.collection.createIndex(
  {
    "sku": 1,
    "manufacturer": 1
  },
  {
    name: "sku_and_manufacturer_unique",
    unique: true
  }
)
```

多金鑰

在條碼上建立唯一的多金鑰索引，以確保條碼陣列中的任何值不存在於超過 1 個文件中：

```
db.collection.createIndex(
  {
    "barcodes": 1
  },
  {
```

```
    name: "barcodes_unique",
    unique: true
  }
)
```

索引陣列會為陣列的每個元素建立索引項目。例如，如果陣列有 50 個項目，則有 50 個索引項目。因此，唯一的多重索引鍵索引會強制執行所有個別項目的唯一性。例如，下列文件會違反值陣列欄位索引上的唯一限制：

```
{ "values": [ 1, 2, 3] }
{ "values": [ 3, 2 ] }   --> 3 and 2 already exist
{ "values": [ 1 ] }      --> 1 already exists
```

請注意具有唯一索引的下列行為：

1. 如果您在現有資料上建立唯一的索引，其中兩個（或更多）文件的索引欄位值相同，則索引建置將會失敗，並出現下列錯誤：`could not create unique index: <collection> index: <index name>`
2. 如果您插入的文件的索引欄位值與另一個文件中該欄位的值相符，則插入會失敗，並顯示下列錯誤：`E11000 duplicate key error collection: <collection> index: <index name>`
3. 如果您更新現有的文件，使索引欄位的新值與另一個文件中該欄位的值相符，則更新會失敗，並顯示下列錯誤：`E11000 duplicate key error collection: <collection> index: <index name>`
4. 如果文件中缺少索引欄位，則值會視為 null。如果兩個（或更多）文件缺少索引欄位，索引建置、插入和更新將會失敗，如上所述。

索引屬性：sparse

支援的索引類型

選項	3.6	4.0	5.0	8.0	彈性叢集
單一欄位	是	是	是	是	是
compound	是	是	是	是	是
多金鑰	是	是	是	是	是

使用稀疏選項略過缺少索引欄位的索引文件，減少索引大小並節省記憶體空間。由於索引大小較小，因此使用它的查詢會更有效率。若要讓查詢使用稀疏索引，您必須在索引欄位上使用 $exists 子句。如果您省略 $exists 子句，Amazon DocumentDB 將不會使用稀疏索引。

範例

下列範例示範如何在下列範例文件上建立稀疏索引：

```
{
  "productId": "PROD133726",
  "sku": "SKU24224",
  "name": "Basic Printer",
  "manufacturer": "The Manufacturer",
  "tags": [ "printer", "basic", "electronics", "business" ],
  "barcodes": [ "542364671", "886330670", "437445606" ],
  "reviews": [
    {
      "review_date": ISODate('2024-01-19T21:37:10.585Z'),
      ...
    }
  ],
  "material": "Polycarbonate",
  "color": "Space Gray",
  "supplier": {
    "supplierId": "SUP4",
    "location": {
      "type": "Point",
      "coordinates": [ -71.0589, 42.3601 ]
    }
  },
  "productEmbedding": [
    -0.019320633663838058,
    0.019672111388113596
  ],
  "lastUpdated": ISODate('2025-10-20T21:37:10.585Z')
}
```

請注意，審核、材料和顏色欄位並不存在於所有文件中。

單一欄位

在物料欄位上建立稀疏的單一欄位索引：

```
db.collection.createIndex(
  {
    "material": 1
  },
  {
    "name": "material_sparse",
    "sparse": true
  }
)
```

在尋找具有物料值的所有產品時，將會使用此索引：

```
db.collection.find({
  "material": {
    $exists: true
  }
})
```

複合

在材質和顏色欄位上建立稀疏複合索引：

```
db.collection.createIndex(
  {
    "material": 1,
    "color": 1
  },
  {
    "name": "material_and_color_sparse",
    "sparse": true
  }
)
```

在尋找具有任何材料和顏色值組合的所有產品時，將會使用此索引：

```
db.collection.find({
  "material": {
    $exists: true
  }
})
```

```
db.collection.find({
  "color": {
    $exists: true
  }
})

db.collection.find({
  $and: [
    {
      "material": {
        $exists: true
      }
    },
    {
      "color": {
        $exists:true
      }
    }
  ]
})

db.collection.find({
  $and: [
    {
      "color": {
        $exists: true
      }
    },
    {
      "material": {
        $exists:true
      }
    }
  ]
})
```

多金鑰

在檢閱陣列上建立稀疏的多金鑰索引：

```
db.collection.createIndex(
  {
    "reviews": 1
  },
```

```
  {
    "name": "reviews_sparse",
    "sparse": true
  }
)
```

尋找具有檢閱陣列的所有產品時，將會使用此索引：

```
db.collection.find({
  "reviews": {
    $exists: true
  }
})
```

索引屬性：partialFilterExpression

支援的索引類型

選項	3.6	4.0	5.0	8.0	彈性叢集
單一欄位	否	否	是	是	否
compound	否	否	是	是	否
多金鑰	否	否	是	是	否
地理空間	否	否	否	否	否

使用 partialFilterExpression 選項建立部分索引，只包含符合指定篩選條件的文件。這可讓您建立更有效率的索引，方法是僅編製集合中文件子集的索引，而不是編製所有文件的索引，減少索引大小並節省記憶體空間。由於索引大小較小，因此使用它的查詢會更有效率。在下列案例中，Amazon DocumentDB 將使用部分索引：

- 查詢述詞完全符合部分索引篩選條件表達式。
- 查詢篩選條件的預期結果是部分篩選條件的邏輯子集。
- 查詢的子述詞可以與其他索引搭配使用。

如需詳細資訊，請參閱部分索引。

範例

下列範例示範如何在下列範例文件上建立部分索引：

```
{
  "productId": "PROD133726",
  "sku": "SKU24224",
  "name": "Basic Printer",
  "manufacturer": "The Manufacturer",
  "tags": [ "printer", "basic", "electronics", "business" ],
  "barcodes": [ "542364671", "886330670", "437445606" ],
  "reviews": [
    {
      "review_date": ISODate('2024-01-19T21:37:10.585Z'),
      ...
    }
  ],
  "material": "Polycarbonate",
  "color": "Space Gray",
  "supplier": {
    "supplierId": "SUP4",
    "location": {
      "type": "Point",
      "coordinates": [ -71.0589, 42.3601 ]
    }
  },
  "productEmbedding": [
    -0.019320633663838058,
    0.019672111388113596
  ],
  "lastUpdated": ISODate('2025-10-20T21:37:10.585Z')
}
```

單一欄位

在製造商上為具有 Space Gray 顏色的產品建立部分單一欄位索引：

```
db.collection.createIndex(
  {
    "manufacturer": 1
  },
  {
    "name": "manufacturer_space_gray",
```

```
    "partialFilterExpression": {
      "color": {
        $eq: "Space Gray"
      }
    }
  }
)
```

依製造商尋找 Space Gray 顏色產品時，將會使用此索引：

```
db.collection.find({
  "manufacturer": "The Manufacturer",
  "color": {
    $eq: "Space Gray"
  }
})
```

複合

為具有 Poly carbonate 物料的產品，在製造商和顏色上建立部分複合索引：

```
db.collection.createIndex(
  {
    "manufacturer": 1,
    "color": 1
  },
  {
    "name": "manufacturer_and_color_polycarbonate",
    "partialFilterExpression": {
      "material": {
        $eq: "Polycarbonate"
      }
    }
  }
)
```

依製造商和顏色尋找聚碳化材料產品時，將會使用此索引：

```
db.collection.find({
  "manufacturer": "The Manufacturer",
  "color": "Space Gray",
  "material": {
```

```
    $eq: "Polycarbonate"
  }
})
```

多金鑰

在具有製造商之產品的標籤上建立部分多金鑰索引：

```
db.collection.createIndex(
  {
    "tags": 1
  },
  {
    "name": "tags_the_manufacturer",
    "partialFilterExpression": {
      "manufacturer": {
        $eq: "The Manufacturer"
      }
    }
  }
)
```

尋找具有製造商之印表機標籤的產品時，將會使用此索引：

```
db.collection.find({
  "tags": "printer",
  "manufacturer": {
    $eq: "The Manufacturer"
  }
})
```

索引屬性： expireAfterSeconds

支援的索引類型

索引類型	3.6	4.0	5.0	8.0	彈性叢集
單一欄位	是	是	是	是	是
多金鑰	是	是	是	是	是

使用 expireAfterSeconds 選項建立存留時間 (TTL) 索引。TTL 索引可讓您根據每個文件的逾時條件，根據文件的存留期刪除文件。當文件達到指定的 TTL 存留期時，會從集合中刪除。

如需 TTL 刪除的最佳實務，請參閱 TTL 和時間序列工作負載。

範例

下列範例示範如何在下列範例文件上建立 TTL 索引：

```
{
  "productId": "PROD133726",
  "sku": "SKU24224",
  "name": "Basic Printer",
  "manufacturer": "The Manufacturer",
  "tags": [ "printer", "basic", "electronics", "business" ],
  "barcodes": [ "542364671", "886330670", "437445606" ],
  "reviews": [
    {
      "review_date": ISODate('2024-01-19T21:37:10.585Z'),
      "rating": 4,
      "comment": "Good product"
    },
    {
      "review_date": ISODate('2024-03-15T14:22:33.120Z'),
      "rating": 5,
      "comment": "Excellent printer"
    },
    {
      "review_date": ISODate('2024-06-08T09:45:18.890Z'),
      "rating": 3,
      "comment": "Average quality"
    }
  ],
  "material": "Polycarbonate",
  "color": "Space Gray",
  "supplier": {
    "supplierId": "SUP4",
    "location": {
      "type": "Point",
      "coordinates": [ -71.0589, 42.3601 ]
    }
  },
  "productEmbedding": [
    -0.019320633663838058,
```

```
    0.019672111388113596
  ],
  "lastUpdated": ISODate('2025-10-20T21:37:10.585Z')
}
```

單一欄位

在 lastUpdated 上建立 TTL 索引 更新以刪除 90 天內未更新的文件：

```
db.collection.createIndex(
  {
    "lastUpdated": 1
  },
  {
    "name": "lastUpdated_ttl",
    "expireAfterSeconds": 7776000
  }
)
```

多金鑰

在 review.review_date 上建立 TTL 索引，以刪除過去一年沒有檢閱的文件：

```
db.collection.createIndex(
  {
    "reviews.review_date": 1
  },
  {
    "name": "reviews_review_date_ttl",
    "expireAfterSeconds": 31536000
  }
)
```

請注意，如果 TTL 索引位於陣列欄位上，則會檢查陣列中的所有項目。如果陣列中的任何時間戳記符合逾時條件，則會刪除文件。

索引屬性：隱藏

Amazon DocumentDB 支援從 Amazon DocumentDB 8.0.1 開始的hidden索引屬性。

查詢規劃器看不到隱藏的索引，也無法用於支援查詢。透過從規劃器隱藏索引，您可以評估捨棄索引的潛在影響，而不會實際捨棄索引。如果影響是負面的，您可以取消隱藏索引，而不必重新建立捨棄的索引。

所有索引類型都支援 hidden 屬性，包括單一欄位、複合、多索引鍵、文字、地理空間和向量索引。

Behavior (行為)

除了從規劃器中隱藏之外，隱藏的索引也會完全維護和強制執行。例如：

- 如果隱藏的索引是唯一的索引，則索引仍會將其唯一的限制套用至文件。
- 如果隱藏的索引是 TTL 索引，索引仍會過期文件。
- 隱藏的索引會包含在 `listIndexes`和 `db.collection.getIndexes()`結果中。
- 隱藏的索引會在集合的寫入操作上更新，並繼續耗用磁碟空間和記憶體。因此，它們會包含在統計資料操作中，例如 `db.collection.stats()`和 `$indexStats`。
- 隱藏未隱藏的索引，或取消隱藏的索引， 會重設其 `$indexStats`。隱藏已隱藏的索引，或取消隱藏已隱藏的索引， 不會重設 `$indexStats`。

限制

- 您無法隱藏`_id`索引。
- 您無法使用 `cursor.hint()`強制使用隱藏的索引。
- `$merge` 如果on欄位上所需的唯一索引隱藏，則 會失敗。在執行 之前，讓索引可見`$merge`。

範例

建立隱藏的索引

若要在隱藏狀態下建立索引，請使用 `db.collection.createIndex()`方法，並在索引定義`true`中將 `hidden`選項設定為 。

例如，下列操作會在 `productId` 欄位上建立隱藏的遞增索引：

```
db.collection.createIndex(
  { "productId": 1 },
  { "name": "productId_1", "hidden": true }
)
```

若要驗證，請在集合`listIndexes`上執行 `db.collection.getIndexes()`或 。隱藏索引的定義`"hidden": true`中包含 。

```
db.collection.getIndexes()
[
```

```
  {
    "v": 4,
    "key": { "_id": 1 },
    "name": "_id_"
  },
  {
    "v": 4,
    "key": { "productId": 1 },
    "name": "productId_1",
    "hidden": true
  }
]
```

只有當值為 時，hidden才會傳回索引選項true。

隱藏現有的索引

使用 collMod命令來隱藏現有的索引。依其 name或依其 識別索引keyPattern，並hidden設為 true。

下列範例會依名稱隱藏索引。

```
db.runCommand({
  collMod: "collection",
  index: {
    name: "productId_1", // Specify the index name
    hidden: true
  }
})
```

您也可以依索引的金鑰模式來識別索引。

```
db.runCommand({
  collMod: "collection",
  index: {
    keyPattern: { productId: 1 }, // Specify the index key specification document
    hidden: true
  }
})
```

取消隱藏現有索引

若要再次顯示隱藏的索引，請使用 collMod命令，並將 hidden設定為 false。

```
db.runCommand({
  collMod: "collection",
  index: { name: "productId_1", hidden: false }
})
```

由於索引會在隱藏時完全維護，因此索引會在取消隱藏時立即可供使用。

搭配 Amazon DocumentDB 使用變更串流

Amazon DocumentDB 中的變更串流功能（具有 MongoDB 相容性）提供在叢集集合中發生的按時間順序排列的變更事件序列。您可以從變更串流讀取事件，以實作各種不同的使用案例，包括下列各項：

- 變更通知
- 使用 Amazon OpenSearch Service (OpenSearch Service) 進行全文搜尋
- 使用 Amazon Redshift 進行分析

應用程式可以使用變更串流來訂閱個別集合上的資料變更。變更串流事件發生時便會排序這些事件，並在記錄事件後儲存這些事件 3 小時 (預設)。使用 `change_stream_log_retention_duration` 參數，保留期最多可延長 7 天。若要修改變更串流保留期間，請參閱 修改變更串流日誌保留期間。

主題

- 受支援的操作
- 帳單
- 限制
- 啟用變更串流
- 範例：搭配 Python 使用變更串流
- 完整文件查詢
- 繼續變更串流
- 使用 繼續變更串流 startAtOperationTime
- 使用 繼續變更串流 postBatchResumeToken
- 變更串流中的交易
- 修改變更串流日誌保留期間
- 在次要執行個體上使用變更串流

受支援的操作

Amazon DocumentDB 支援變更串流的下列操作：

- MongoDB `db.collection.watch()`、 `db.watch()`和 `client.watch()` API 支援的所有變更事件。
- 更新的完整文件查閱。
- 彙總階段：`$match`、`$redact`、 `$project`和 `$addFields`$replaceRoot。
- 從繼續字符繼續變更串流
- 使用 從時間戳記繼續變更串流 `startAtOperation` (適用於 Amazon DocumentDB 4.0+)

帳單

Amazon DocumentDB 變更串流功能預設為停用，在啟用此功能之前不會產生任何額外費用。在叢集中使用變更串流會產生額外的讀取和寫入 IOs和儲存成本。您可以使用 `modifyChangeStreams` API 操作為您的叢集啟用此功能。如需定價的詳細資訊，請參閱 Amazon DocumentDB 定價。

限制

變更串流在 Amazon DocumentDB 中具有下列限制：

- 在 Amazon DocumentDB 3.6. 和 Amazon DocumentDB 4.0 上，變更串流只能從 Amazon DocumentDB 叢集的主要執行個體連線開啟。Amazon DocumentDB 3.6. 和 Amazon DocumentDB 4.0 不支援從複本執行個體上的變更串流讀取。呼叫 `watch()` API 操作時，您必須指定 `primary` 讀取偏好設定，以確保所有讀取全都導向至主要執行個體 (請參閱範例一節)。
- 在 Amazon DocumentDB 5.0 上，可以從主要執行個體和次要執行個體開啟變更串流，包括全域叢集。您可以指定次要讀取偏好設定，將變更串流重新導向至次要執行個體。如需其他最佳實務和限制在次要執行個體上使用變更串流，請參閱 。
- 寫入集合變更串流的事件最多可使用 7 天 （ 預設值為 3 小時 ）。在日誌保留期間時段後將會刪除變更串流資料，即便未發生任何新變更。
- 在如 `updateMany` 或 `deleteMany` 等集合上長時間執行寫入操作時，會暫停寫入變更串流事件，直到長時間執行的寫入操作完成為止。
- Amazon DocumentDB 不支援 MongoDB 操作日誌 (`oplog`)。
- 使用 Amazon DocumentDB，您必須在指定的集合上明確啟用變更串流。
- 如果變更串流事件的總大小 (如有要求，包括變更資料與完整文件) 大於 16 MB，用戶端將會遇到變更串流讀取失敗的情況。

- `client.watch()` 搭配 Amazon DocumentDB 3.6 使用 `db.watch()`和 時，目前不支援 Ruby 驅動程式。
- 當欄位的更新值與上一個值相同時，Amazon DocumentDB 中的 `updateDescription`命令輸出與 MongoDB 中的輸出不同：
  - 如果在`$set`命令中指定提供的欄位，且其目標值已等於來源值，Amazon DocumentDB 不會傳回`updateDescription`輸出中的欄位。
  - MongoDB 會傳回輸出中的 欄位，即使指定的值等於目前的值。

啟用變更串流

您可以為指定資料庫中的所有集合啟用 Amazon DocumentDB 變更串流，或僅為選取的集合啟用 Amazon DocumentDB 變更串流。以下是針對使用 mongo 殼層的不同使用案例，如何啟用變更串流的範例。指定資料庫和集合名稱時，空的字串會被視為萬用字元。

```
//Enable change streams for the collection "foo" in database "bar"
db.adminCommand({modifyChangeStreams: 1,
    database: "bar",
    collection: "foo",
    enable: true});
```

```
//Disable change streams on collection "foo" in database "bar"
db.adminCommand({modifyChangeStreams: 1,
    database: "bar",
    collection: "foo",
    enable: false});
```

```
//Enable change streams for all collections in database "bar"
db.adminCommand({modifyChangeStreams: 1,
    database: "bar",
    collection: "",
    enable: true});
```

```
//Enable change streams for all collections in all databases in a cluster
db.adminCommand({modifyChangeStreams: 1,
    database: "",
    collection: "",
    enable: true});
```

如果下列任一項成立，集合將會啟用變更串流：

- 明確地啟用資料庫和集合。
- 已啟用包含集合的資料庫。
- 已啟用所有資料庫。

如果父資料庫也啟用變更串流或叢集內的所有資料庫都已啟用，捨棄資料庫的集合並不會停用該集合的變更串流。如果新的集合以遭刪除集合的相同名稱建立，該集合將會啟用變更串流。

您可以使用 $listChangeStreams 彙總管線階段列出叢集所有已啟用的變更串流。Amazon DocumentDB 支援的所有彙總階段都可以在管道中使用，以進行其他處理。如果已停用之前啟用的集合，該集合將不會在 $listChangeStreams 輸出中出現。

```
//List all databases and collections with change streams enabled
cursor = new DBCommandCursor(db,
    db.runCommand(
        {aggregate: 1,
        pipeline: [{$listChangeStreams: 1}],
        cursor:{}}));
```

```
//List of all databases and collections with change streams enabled
{ "database" : "test", "collection" : "foo" }
{ "database" : "bar", "collection" : "" }
{ "database" : "", "collection" : "" }
```

```
//Determine if the database “bar” or collection “bar.foo” have change streams enabled
cursor = new DBCommandCursor(db,
  db.runCommand(
      {aggregate: 1,
       pipeline: [{$listChangeStreams: 1},
                  {$match: {$or: [{database: "bar", collection: "foo"},
                                  {database: "bar", collection: ""},
                                  {database: "", collection: ""}]}}
                ],
      cursor:{}}));
```

範例：搭配 Python 使用變更串流

以下是在集合層級搭配 Python 使用 Amazon DocumentDB 變更串流的範例。

```
import os
import sys
from pymongo import MongoClient, ReadPreference

username = "DocumentDBusername"
password = <Insert your password>

clusterendpoint = "DocumentDBClusterEndpoint”
client = MongoClient(clusterendpoint, username=username, password=password, tls='true',
 tlsCAFile='global-bundle.pem')

db = client['bar']

#While ‘Primary’ is the default read preference, here we give an example of
#how to specify the required read preference when reading the change streams
coll = db.get_collection('foo', read_preference=ReadPreference.PRIMARY)
#Create a stream object
stream = coll.watch()
#Write a new document to the collection to generate a change event
coll.insert_one({'x': 1})
#Read the next change event from the stream (if any)
print(stream.try_next())

"""
Expected Output:
{'_id': {'_data': '015daf94f600000002010000000200009025'},
'clusterTime': Timestamp(1571788022, 2),
'documentKey': {'_id': ObjectId('5daf94f6ea258751778163d6')},
'fullDocument': {'_id': ObjectId('5daf94f6ea258751778163d6'), 'x': 1},
'ns': {'coll': 'foo', 'db': 'bar'},
'operationType': 'insert'}
"""

#A subsequent attempt to read the next change event returns nothing, as there are no
 new changes
print(stream.try_next())

"""
Expected Output:
None
"""

#Generate a new change event by updating a document
```

```
result = coll.update_one({'x': 1}, {'$set': {'x': 2}})
print(stream.try_next())

"""
Expected Output:
{'_id': {'_data': '015daf99d40000000101000000100009025'},
'clusterTime': Timestamp(1571789268, 1),
'documentKey': {'_id': ObjectId('5daf9502ea258751778163d7')},
'ns': {'coll': 'foo', 'db': 'bar'},
'operationType': 'update',
'updateDescription': {'removedFields': [], 'updatedFields': {'x': 2}}}
"""
```

以下是在資料庫層級搭配 Python 使用 Amazon DocumentDB 變更串流的範例。

```
import os
import sys
from pymongo import MongoClient

username = "DocumentDBusername"
password = <Insert your password>
clusterendpoint = "DocumentDBClusterEndpoint”
client = MongoClient(clusterendpoint, username=username, password=password, tls='true',
 tlsCAFile='global-bundle.pem')

db = client['bar']
#Create a stream object
stream = db.watch()
coll = db.get_collection('foo')
#Write a new document to the collection foo to generate a change event
coll.insert_one({'x': 1})

#Read the next change event from the stream (if any)
print(stream.try_next())

"""
Expected Output:
{'_id': {'_data': '015daf94f60000000201000000200009025'},
'clusterTime': Timestamp(1571788022, 2),
'documentKey': {'_id': ObjectId('5daf94f6ea258751778163d6')},
'fullDocument': {'_id': ObjectId('5daf94f6ea258751778163d6'), 'x': 1},
'ns': {'coll': 'foo', 'db': 'bar'},
'operationType': 'insert'}
```

```
"""
#A subsequent attempt to read the next change event returns nothing, as there are no
 new changes
print(stream.try_next())

"""
Expected Output:
None
"""

coll = db.get_collection('foo1')

#Write a new document to another collection to generate a change event
coll.insert_one({'x': 1})
print(stream.try_next())

"""
Expected Output: Since the change stream cursor was the database level you can see
 change events from different collections in the same database
{'_id': {'_data': '015daf94f600000002010000000200009025'},
'clusterTime': Timestamp(1571788022, 2),
'documentKey': {'_id': ObjectId('5daf94f6ea258751778163d6')},
'fullDocument': {'_id': ObjectId('5daf94f6ea258751778163d6'), 'x': 1},
'ns': {'coll': 'foo1', 'db': 'bar'},
'operationType': 'insert'}
"""
```

完整文件查詢

更新變更事件不包括完整文件；只包含所做的變更。如果您的使用案例需要受更新影響的完整文件，您可以在開啟串流時啟用完整文件查閱。

更新變更串流事件的 `fullDocument` 文件代表在文件查閱時已更新文件最新的版本。如果更新操作與 `fullDocument` 查閱之間發生變更，`fullDocument` 文件可能不會代表更新時的文件狀態。

若要建立啟用更新查詢的串流物件，請使用此範例：

```
stream = coll.watch(full_document='updateLookup')

#Generate a new change event by updating a document
result = coll.update_one({'x': 2}, {'$set': {'x': 3}})
```

```
stream.try_next()
```

串流物件的輸出看起來會像這樣：

```
{'_id': {'_data': '015daf9b7c000000010100000100009025'},
'clusterTime': Timestamp(1571789692, 1),
'documentKey': {'_id': ObjectId('5daf9502ea258751778163d7')},
'fullDocument': {'_id': ObjectId('5daf9502ea258751778163d7'), 'x': 3},
'ns': {'coll': 'foo', 'db': 'bar'},
'operationType': 'update',
'updateDescription': {'removedFields': [], 'updatedFields': {'x': 3}}}
```

繼續變更串流

您稍後可以使用繼續字符 (等於上次擷取之變更事件文件的 _id 欄位) 繼續變更串流。

```
import os
import sys
from pymongo import MongoClient

username = "DocumentDBusername"
password = <Insert your password>
clusterendpoint = "DocumentDBClusterEndpoint”
client = MongoClient(clusterendpoint, username=username, password=password, tls='true',
 tlsCAFile='global-bundle.pem', retryWrites='false')

db = client['bar']
coll = db.get_collection('foo')
#Create a stream object
stream = db.watch()
coll.update_one({'x': 1}, {'$set': {'x': 4}})
event = stream.try_next()
token = event['_id']
print(token)

"""
Output: This is the resume token that we will later us to resume the change stream
{'_data': '015daf9c5b000000010100000100009025'}
"""
#Python provides a nice shortcut for getting a stream’s resume token
print(stream.resume_token)
```

```
"""
Output
{'_data': '015daf9c5b0000000101000000100009025'}
"""
#Generate a new change event by updating a document
result = coll.update_one({'x': 4}, {'$set': {'x': 5}})
#Generate another change event by inserting a document
result = coll.insert_one({'y': 5})
#Open a stream starting after the selected resume token
stream = db.watch(full_document='updateLookup', resume_after=token)
#Our first change event is the update with the specified _id
print(stream.try_next())

"""
#Output: Since we are resuming the change stream from the resume token, we will see all
 events after the first update operation. In our case, the change stream will resume
 from the update operation {x:5}

{'_id': {'_data': '015f7e8f0c0000000601000000006000fe038'},
'operationType': 'update',
'clusterTime': Timestamp(1602129676, 6),
'ns': {'db': 'bar', 'coll': 'foo'},
'documentKey': {'_id': ObjectId('5f7e8f0ac423bafbfd9adba2')},
'fullDocument': {'_id': ObjectId('5f7e8f0ac423bafbfd9adba2'), 'x': 5},
'updateDescription': {'updatedFields': {'x': 5}, 'removedFields': []}}
"""
#Followed by the insert
print(stream.try_next())

"""
#Output:
{'_id': {'_data': '015f7e8f0c0000000701000000007000fe038'},
'operationType': 'insert',
'clusterTime': Timestamp(1602129676, 7),
'ns': {'db': 'bar', 'coll': 'foo'},
'documentKey': {'_id': ObjectId('5f7e8f0cbf8c233ed577eb94')},
'fullDocument': {'_id': ObjectId('5f7e8f0cbf8c233ed577eb94'), 'y': 5}}
"""
```

使用 繼續變更串流 **startAtOperationTime**

您可以使用 ，稍後從特定時間戳記繼續變更串流startAtOperationTime。

> **Note**
>
> 您可以在 Amazon DocumentDB 4.0+ `startAtOperationTime`中使用 功能。使用時`startAtOperationTime`，變更串流游標只會傳回在指定時間戳記或之後發生的變更。`startAtOperationTime` 和 `resumeAfter`命令是互斥的，因此無法一起使用。

```
import os
import sys
from pymongo import MongoClient

username = "DocumentDBusername"
password = <Insert your password>
clusterendpoint = "DocumentDBClusterEndpoint”
client = MongoClient(clusterendpoint, username=username, password=password, tls='true',
 tlsCAFile='rds-root-ca-2020.pem',retryWrites='false')
db = client['bar']
coll = db.get_collection('foo')
#Create a stream object
stream = db.watch()
coll.update_one({'x': 1}, {'$set': {'x': 4}})
event = stream.try_next()
timestamp = event['clusterTime']
print(timestamp)
"""
Output
Timestamp(1602129114, 4)
"""
#Generate a new change event by updating a document
result = coll.update_one({'x': 4}, {'$set': {'x': 5}})
result = coll.insert_one({'y': 5})
#Generate another change event by inserting a document
#Open a stream starting after specified time stamp

stream = db.watch(start_at_operation_time=timestamp)
print(stream.try_next())

"""
#Output: Since we are resuming the change stream at the time stamp of our first update
 operation (x:4), the change stream cursor will point to that event
{'_id': {'_data': '015f7e941a0000000301000000003000fe038'},
'operationType': 'update',
```

```
'clusterTime': Timestamp(1602130970, 3),
'ns': {'db': 'bar', 'coll': 'foo'},
'documentKey': {'_id': ObjectId('5f7e9417c423bafbfd9adbb1')},
'updateDescription': {'updatedFields': {'x': 4}, 'removedFields': []}}
"""

print(stream.try_next())
"""
#Output: The second event will be the subsequent update operation (x:5)
{'_id': {'_data': '015f7e9502000000050100000005000fe038'},
'operationType': 'update',
'clusterTime': Timestamp(1602131202, 5),
'ns': {'db': 'bar', 'coll': 'foo'},
'documentKey': {'_id': ObjectId('5f7e94ffc423bafbfd9adbb2')},
'updateDescription': {'updatedFields': {'x': 5}, 'removedFields': []}}
"""

print(stream.try_next())

"""
#Output: And finally the last event will be the insert operation (y:5)
{'_id': {'_data': '015f7e9502000000060100000006000fe038'},
'operationType': 'insert',
'clusterTime': Timestamp(1602131202, 6),
'ns': {'db': 'bar', 'coll': 'foo'},
'documentKey': {'_id': ObjectId('5f7e95025c4a569e0f6dde92')},
'fullDocument': {'_id': ObjectId('5f7e95025c4a569e0f6dde92'), 'y': 5}}
"""
```

使用 繼續變更串流 **postBatchResumeToken**

Amazon DocumentDB 變更串流現在會傳回名為 的其他欄位postBatchResumeToken。此欄位會從 $changestream命令和 getMore命令傳回。

Python 中的 $changestream命令範例：

```
db.command({"aggregate": "sales", "pipeline": [{ "$changeStream": {}}], "cursor":
 {"batchSize": 1}})
```

預期的輸出結果：

```
cursor" : {
```

```
   "firstBatch" : [ ],
   "postBatchResumeToken" : {"_data" : "0167c8cbe60000000004"},
   "id" : NumberLong("9660788144470"),
   "ns" : "test.sales"
}
```

Python 中的 getMore命令範例：

```
db.command({"getMore": NumberLong(<cursor id>), "collection": "sales", "batchSize":
 1 })
```

預期的輸出結果

```
cursor" : {
   "nextBatch" : [ ],
   "postBatchResumeToken" : {"_data" : "0167c8cbe60000000004"},
   "id" : NumberLong("9660788144470"),
   "ns" : "test.sales"
}
```

postBatchResumeToken 欄位可用來在 resumeAfter 欄位中開啟新的變更串流游標，類似於使用繼續字符的方式。

從選取的 開始開啟串流postBatchResumeToken：

```
post_batch_resume_token = output['cursor']['postBatchResumeToken']
stream = db.watch(full_document='updateLookup', resume_after=post_batch_resume_token)
```

與一律對應至反映實際事件的操作日誌 (oplog) 項目的一般繼續字符不同，postBatchResumeToken對應至變更串流在伺服器上掃描到的 oplog 項目，這不一定是相符的變更。

嘗試使用舊的規則繼續字符繼續，將強制資料庫掃描指定時間戳記與目前時間之間的所有 oplog 項目。這可能會在內部產生許多查詢，每次子查詢掃描都會產生一小段時間。這會導致 CPU 使用量激增，並降低資料庫效能。使用最後一個項目繼續會postBatchResumeToken略過掃描不相符的 oplog 項目。

變更串流中的交易

變更串流事件不會包含來自未遞交和/或中止交易的事件。例如，如果您使用一個INSERT操作和一個UPDATE操作啟動交易，而且如果INSERT操作成功，但UPDATE操作失敗，則會復原交易。由於此交易已復原，您的變更串流將不會包含此交易的任何事件。

修改變更串流日誌保留期間

您可以使用 AWS 管理主控台 或 ，將變更串流日誌保留期間修改為 1 小時至 7 天 AWS CLI。

Using the AWS 管理主控台

修改變更串流日誌保留期限

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在 Parameter groups (參數群組) 窗格中，選擇與叢集相關聯的叢集參數群組。若要識別與叢集相關聯的叢集參數群組，請參閱判斷 Amazon DocumentDB 叢集的參數群組。
4. 結果頁面會顯示這個叢集參數群組的參數，及其對應詳細資訊。選取 `change_stream_log_retention_duration` 參數。
5. 在頁面右上角，選擇 Edit (編輯) 以變更此參數值。`change_stream_log_retention_duration` 參數可以修改為 1 小時到 7 天。
6. 進行變更，然後選擇 Modify cluster parameter (修改叢集參數) 以儲存變更。若要捨棄變更，請選擇 Cancel (取消)。

Using the AWS CLI

若要修改叢集參數群組的 `change_stream_log_retention_duration` 參數，請使用 `modify-db-cluster-parameter-group` 操作搭配下列參數：

- **`--db-cluster-parameter-group-name`** – 必要。您想要修改的叢集參數群組名稱。若要識別與叢集相關聯的叢集參數群組，請參閱判斷 Amazon DocumentDB 叢集的參數群組。
- **`--parameters`** – 必要。您要修改的參數。每個參數項目必須包括下列：
  - **`ParameterName`** — 您要修改的參數名稱。在這種情況下，它是 `change_stream_log_retention_duration`
  - **`ParameterValue`** — 此參數的新值。

- **ApplyMethod** — 您希望如何套用此參數的變更。允許值為 `immediate` 和 `pending-reboot`。

  > **Note**
  >
  > `static` 必須擁有使用 ApplyType 的參數 ApplyMethod `pending-reboot`。

1. 若要變更參數 `change_stream_log_retention_duration` 的值，請執行下列命令，並以您想要修改參數的值取代 `parameter-value`。

   針對 Linux、macOS 或 Unix：

   ```
   aws docdb modify-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group \
       --parameters
    "ParameterName=change_stream_log_retention_duration,ParameterValue=<parameter-
   value>,ApplyMethod=immediate"
   ```

   針對 Windows：

   ```
   aws docdb modify-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group ^
       --parameters
    "ParameterName=change_stream_log_retention_duration,ParameterValue=<parameter-
   value>,ApplyMethod=immediate"
   ```

   此操作的輸出將會如下所示 (JSON 格式)。

   ```
   {
       "DBClusterParameterGroupName": "sample-parameter-group"
   }
   ```

2. 等待至少 5 分鐘。
3. 列出 `sample-parameter-group` 的參數值確保已進行變更。

   針對 Linux、macOS 或 Unix：

   ```
   aws docdb describe-db-cluster-parameters \
   ```

```
    --db-cluster-parameter-group-name sample-parameter-group
```

針對 Windows：

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name sample-parameter-group
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "Parameters": [
        {
            "ParameterName": "audit_logs",
            "ParameterValue": "disabled",
            "Description": "Enables auditing on cluster.",
            "Source": "system",
            "ApplyType": "dynamic",
            "DataType": "string",
            "AllowedValues": "enabled,disabled",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot"
        },
        {
            "ParameterName": "change_stream_log_retention_duration",
            "ParameterValue": "12345",
            "Description": "Duration of time in seconds that the change stream
 log is retained and can be consumed.",
            "Source": "user",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "3600-86400",
            "IsModifiable": true,
            "ApplyMethod": "immediate"
        }
    ]
}
```

> **Note**
>
> 變更串流日誌保留不會刪除早於設定change_stream_log_retention_duration值的日誌，直到日誌大小大於 (>) 51，200MB。

在次要執行個體上使用變更串流

若要開始在次要執行個體上使用變更串流，請使用 readPreference作為次要執行個體開啟變更串流游標。

您可以開啟變更串流游標，以監控特定集合或叢集或資料庫中所有集合的變更事件。您可以在任何 Amazon DocumentDB 執行個體上開啟變更串流游標，並從寫入器和讀取器執行個體擷取變更串流文件。您可以在寫入器和讀取器執行個體上開啟的不同變更串流游標之間共用變更串流字符 （ 例如 resumeToken或 startOperationTime)。

範例

```
import os
import sys
from pymongo import MongoClient, ReadPreference

username = "DocumentDBusername"
password = <Your password>

clusterendpoint = "DocumentDBClusterEndpoint"

client = MongoClient(clusterendpoint, username=username, password=password, tls='true',
 tlsCAFile='global-bundle.pem')

db = client['bar']

# Make sure to use SECONDARY to redirect cursor reads from secondary instances
coll = db.get_collection('foo', read_preference=ReadPreference.SECONDARY)

# Create a stream object on RO. The token needs to generated from PRIMARY.
stream = coll.watch(resumeAfter=token)

for event in stream:
   print(event)
```

次要執行個體上變更串流的準則和限制

- 變更串流事件需要從主要執行個體複寫到次要執行個體。您可以從 Amazon CloudWatch 中的 `DBInstanceReplicaLag` 指標監控延遲。
- 次要執行個體上的時間戳記不一定會與主要執行個體同步。在此情況下， 預期次要執行個體時間戳記會延遲，以便趕上進度。最佳實務是使用 `startAtOperationTime`或 `resumeToken` 在次要執行個體上啟動監看。
- 如果文件大小較大、您正在執行 `fullDocumentLookup`，且主要執行個體上的並行寫入工作負載很高，則次要執行個體的輸送量可能會比主要執行個體低。最佳實務是監控次要 上的緩衝區快取命中率，並確保緩衝區快取命中率很高。

Amazon DocumentDB 中的交易

Amazon DocumentDB （與 MongoDB 相容） 支援 Amazon DocumentDB 4.0 和更新版本中的交易。您可以跨多個文件、陳述式、集合和資料庫執行交易。交易可讓您跨 Amazon DocumentDB 叢集中的一或多個文件執行原子、一致、隔離和耐用 (ACID) 操作，以簡化應用程式開發。交易的常見使用案例包括財務處理、履行和管理訂單，以及建置多玩家遊戲。

交易無需額外費用。您只需為在交易中所使用的讀取和寫入 IOs 付費。

主題

- 要求
- 最佳實務
- 限制
- 監控和診斷
- 交易隔離層級
- 使用案例
- 支援的命令
- 不支援的功能
- 工作階段
- 交易錯誤

要求

若要使用交易功能，您需要符合下列要求：

- 您必須使用 Amazon DocumentDB 4.0 或更新版本。
- 您必須使用與 MongoDB 4.0 或更新版本相容的驅動程式。

最佳實務

以下是一些最佳實務，讓您可以充分利用與 Amazon DocumentDB 的交易。

- 交易完成後，一律遞交或中止交易。讓交易處於不完整狀態，會使資料庫資源產生關聯，並可能導致寫入衝突。
- 建議將交易保持在所需的最小命令數量。如果您有多個陳述式的交易可以分為多個較小的交易，建議您這樣做，以減少逾時的可能性。一律以建立短期交易為目標，而非長時間執行的讀取。

限制

- Amazon DocumentDB 不支援交易中的游標。
- Amazon DocumentDB 無法在交易中建立新的集合，也無法針對不存在的集合進行查詢/更新。
- 文件層級寫入鎖定有 1 分鐘的逾時，使用者無法設定。
- Amazon DocumentDB 不支援可重試寫入、可重試遞交和可重試中止命令。如果您使用的是舊版 mongo shell （非 mongosh)，請勿在任何程式碼字串中包含 `retryWrites=false`命令。預設會停用可重試寫入。包含 `retryWrites=false` 可能會導致正常讀取命令失敗。
- 每個 Amazon DocumentDB 執行個體對於可在執行個體上開啟的並行交易數量都有上限。如需詳細資訊，請參閱執行個體配額。
- 對於指定的交易，交易日誌大小必須小於 32MB。
- Amazon DocumentDB 在交易`count()`中支援 ，但並非所有驅動程式都支援此功能。另一種方法是使用 `countDocuments()` API，將計數查詢轉譯為用戶端的彙總查詢。
- 交易有一分鐘的執行限制，工作階段有 30 分鐘的逾時。如果交易逾時，將會中止，而且在現有交易的工作階段中發出的任何後續命令都會產生下列錯誤：

```
WriteCommandError({
"ok" : 0,
"operationTime" : Timestamp(1603491424, 627726),
"code" : 251,
"errmsg" : "Given transaction number 0 does not match any in-progress transactions."
}
```

監控和診斷

透過支援 Amazon DocumentDB 4.0 中的交易，新增了額外的 CloudWatch 指標，以協助您監控交易。

新的 CloudWatch 指標

- `DatabaseTransactions`：在一分鐘期間內進行的開啟交易數量。
- `DatabaseTransactionsAborted`：在一分鐘期間內進行的中止交易數量。
- `DatabaseTransactionsMax`：一分鐘期間內開啟的交易數量上限。
- `TransactionsAborted`：在一分鐘期間內在執行個體上中止的交易數量。
- `TransactionsCommitted`：在一分鐘期間內對執行個體遞交的交易數量。
- `TransactionsOpen`：在一分鐘期間內執行個體上開啟的交易數量。
- `TransactionsOpenMax`：在一分鐘期間內在執行個體上開啟的交易數量上限。
- `TransactionsStarted`：在一分鐘期間內在執行個體上啟動的交易數量。

> **Note**
>
> 如需 Amazon DocumentDB 的更多 CloudWatch 指標，請前往 使用 CloudWatch 監控 Amazon DocumentDB。

此外，新欄位已新增至`currentOp`「`idle transaction`」`lsid``transactionThreadId`和`serverStatus`交易的新狀態：`currentActive`、`currentInactive`、`currentOpen`、`totalCommitted`、`totalAborted`和 `totalStarted`。

交易隔離層級

開始交易時，您可以同時指定 `readConcern`和 `writeConcern`，如下列範例所示：

```
mySession.startTransaction({readConcern: {level: 'snapshot'}, writeConcern: {w: 'majority'}});
```

對於 `readConcern`，Amazon DocumentDB 預設支援快照隔離。如果指定本機、可用或大多數`readConcern`的 ，Amazon DocumentDB 會將`readConcern`關卡升級至快照。Amazon DocumentDB 不支援線性化`readConcern`，且指定此類讀取考量將導致錯誤。

對於 writeConcern，Amazon DocumentDB 預設支援大多數，並在三個 AZs 之間保留四個資料複本時達到寫入仲裁。如果writeConcern指定較低的值，Amazon DocumentDB 會將 writeConcern 升級至大多數。此外，所有 Amazon DocumentDB 寫入都會進行日誌記錄，且無法停用日誌記錄。

使用案例

在本節中，我們將逐步介紹交易的兩個使用案例：多陳述式和多集合。

多陳述式交易

Amazon DocumentDB 交易是多陳述式，這表示您可以撰寫跨越多個陳述式且具有明確遞交或轉返的交易。您可以將 insert、delete、 update和 findAndModify動作分組為單一原子操作。

多陳述式交易的常見使用案例是扣款額度交易。例如：您欠朋友衣服的錢。因此，您需要從您的 帳戶扣款（提取）500 美元，並將 500 美元（存款）的點數計入朋友的帳戶。若要執行該操作，您可以在單一交易中同時執行扣款和點數操作，以確保原子性。這樣做可以防止從您的帳戶中扣除 500 USD，但不會記入您朋友帳戶的情況。以下是此使用案例的外觀：

```
// *** Transfer $500 from Alice to Bob inside a transaction: Success Scenario***
// Setup bank account for Alice and Bob. Each have $1000 in their account

var databaseName = "bank";
var collectionName = "account";
var amountToTransfer = 500;

var session = db.getMongo().startSession({causalConsistency: false});
var bankDB = session.getDatabase(databaseName);
var accountColl = bankDB[collectionName];
accountColl.drop();

accountColl.insert({name: "Alice", balance: 1000});
accountColl.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountColl.find({"name": "Alice"}).next().balance;
var newAliceBalance = aliceBalance - amountToTransfer;
accountColl.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountColl.find({"name": "Alice"}).next().balance;
```

```
// add $500 to Bob's account
var bobBalance = accountColl.find({"name": "Bob"}).next().balance;
var newBobBalance = bobBalance + amountToTransfer;
accountColl.update({"name": "Bob"},{"$set": {"balance": newBobBalance}});
var findBobBalance = accountColl.find({"name": "Bob"}).next().balance;

session.commitTransaction();

accountColl.find();

// *** Transfer $500 from Alice to Bob inside a transaction: Failure Scenario***

// Setup bank account for Alice and Bob. Each have $1000 in their account
var databaseName = "bank";
var collectionName = "account";
var amountToTransfer = 500;

var session = db.getMongo().startSession({causalConsistency: false});
var bankDB = session.getDatabase(databaseName);
var accountColl = bankDB[collectionName];
accountColl.drop();

accountColl.insert({name: "Alice", balance: 1000});
accountColl.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountColl.find({"name": "Alice"}).next().balance;
var newAliceBalance = aliceBalance - amountToTransfer;
accountColl.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountColl.find({"name": "Alice"}).next().balance;

session.abortTransaction();
```

多集合交易

我們的交易也是多集合，這表示它們可以在單一交易和多個集合中執行多個操作。這可提供一致的資料檢視，並維護資料的完整性。當您將命令遞交為單一 時<>，交易是all-or-nothing執行，其中，它們全部成功或全部失敗。

以下是多集合交易的範例，使用多陳述式交易範例中的相同案例和資料。

```
// *** Transfer $500 from Alice to Bob inside a transaction: Success Scenario***

// Setup bank account for Alice and Bob. Each have $1000 in their account
var amountToTransfer = 500;
var collectionName = "account";

var session = db.getMongo().startSession({causalConsistency: false});
var accountCollInBankA = session.getDatabase("bankA")[collectionName];
var accountCollInBankB = session.getDatabase("bankB")[collectionName];

accountCollInBankA.drop();
accountCollInBankB.drop();

accountCollInBankA.insert({name: "Alice", balance: 1000});
accountCollInBankB.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountCollInBankA.find({"name": "Alice"}).next().balance;
var newAliceBalance = aliceBalance - amountToTransfer;
accountCollInBankA.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountCollInBankA.find({"name": "Alice"}).next().balance;

// add $500 to Bob's account
var bobBalance = accountCollInBankB.find({"name": "Bob"}).next().balance;
var newBobBalance = bobBalance + amountToTransfer;
accountCollInBankB.update({"name": "Bob"},{"$set": {"balance": newBobBalance}});
var findBobBalance = accountCollInBankB.find({"name": "Bob"}).next().balance;

session.commitTransaction();

accountCollInBankA.find(); // Alice holds $500 in bankA
accountCollInBankB.find(); // Bob holds $1500 in bankB

// *** Transfer $500 from Alice to Bob inside a transaction: Failure Scenario***

// Setup bank account for Alice and Bob. Each have $1000 in their account
var collectionName = "account";
var amountToTransfer = 500;

var session = db.getMongo().startSession({causalConsistency: false});
```

```
var accountCollInBankA = session.getDatabase("bankA")[collectionName];
var accountCollInBankB = session.getDatabase("bankB")[collectionName];

accountCollInBankA.drop();
accountCollInBankB.drop();

accountCollInBankA.insert({name: "Alice", balance: 1000});
accountCollInBankB.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountCollInBankA.find({"name": "Alice"}).next().balance;
var newAliceBalance = aliceBalance - amountToTransfer;
accountCollInBankA.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountCollInBankA.find({"name": "Alice"}).next().balance;

// add $500 to Bob's account
var bobBalance = accountCollInBankB.find({"name": "Bob"}).next().balance;
var newBobBalance = bobBalance + amountToTransfer;
accountCollInBankB.update({"name": "Bob"},{"$set": {"balance": newBobBalance}});
var findBobBalance = accountCollInBankB.find({"name": "Bob"}).next().balance;

session.abortTransaction();

accountCollInBankA.find(); // Alice holds $1000 in bankA
accountCollInBankB.find(); // Bob holds $1000 in bankB
```

回呼 API 的交易 API 範例

回呼 API 僅適用於 4.2+ 驅動程式。

Javascript

下列程式碼示範如何搭配 Javascript 使用 Amazon DocumentDB 交易 API。

```
// *** Transfer $500 from Alice to Bob inside a transaction: Success ***
// Setup bank account for Alice and Bob. Each have $1000 in their account
var databaseName = "bank";
var collectionName = "account";
var amountToTransfer = 500;

var session = db.getMongo().startSession({causalConsistency: false});
```

```
var bankDB = session.getDatabase(databaseName);
var accountColl = bankDB[collectionName];
accountColl.drop();

accountColl.insert({name: "Alice", balance: 1000});
accountColl.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountColl.find({"name": "Alice"}).next().balance;
assert(aliceBalance >= amountToTransfer);
var newAliceBalance = aliceBalance - amountToTransfer;
accountColl.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountColl.find({"name": "Alice"}).next().balance;
assert.eq(newAliceBalance, findAliceBalance);

// add $500 to Bob's account
var bobBalance = accountColl.find({"name": "Bob"}).next().balance;
var newBobBalance = bobBalance + amountToTransfer;
accountColl.update({"name": "Bob"},{"$set": {"balance": newBobBalance}});
var findBobBalance = accountColl.find({"name": "Bob"}).next().balance;
assert.eq(newBobBalance, findBobBalance);

session.commitTransaction();

accountColl.find();
```

Node.js

下列程式碼示範如何使用 Amazon DocumentDB 交易 API 搭配 Node.js。

```
// Node.js callback API:

const bankDB = await mongoclient.db("bank");
var accountColl = await bankDB.createCollection("account");
var amountToTransfer = 500;

const session = mongoclient.startSession({causalConsistency: false});
await accountColl.drop();

await accountColl.insertOne({name: "Alice", balance: 1000}, { session });
await accountColl.insertOne({name: "Bob", balance: 1000}, { session });
```

```
const transactionOptions = {
    readConcern: { level: 'snapshot' },
    writeConcern: { w: 'majority' }
    };

// deduct $500 from Alice's account
var aliceBalance = await accountColl.findOne({name: "Alice"}, {session});
assert(aliceBalance.balance >= amountToTransfer);
var newAliceBalance = aliceBalance - amountToTransfer;
session.startTransaction(transactionOptions);
await accountColl.updateOne({name: "Alice"}, {$set: {balance: newAliceBalance}},
 {session });
await session.commitTransaction();
aliceBalance = await accountColl.findOne({name: "Alice"}, {session});
assert(newAliceBalance == aliceBalance.balance);

// add $500 to Bob's account
var bobBalance = await accountColl.findOne({name: "Bob"}, {session});
var newBobBalance = bobBalance.balance + amountToTransfer;
session.startTransaction(transactionOptions);
await accountColl.updateOne({name: "Bob"}, {$set: {balance: newBobBalance}},
 {session });
await session.commitTransaction();
bobBalance = await accountColl.findOne({name: "Bob"}, {session});
assert(newBobBalance == bobBalance.balance);
```

C#

下列程式碼示範如何使用 Amazon DocumentDB 交易 API 搭配 C#。

```
// C# Callback API

var dbName = "bank";
var collName = "account";
var amountToTransfer = 500;

using (var session = client.StartSession(new ClientSessionOptions{CausalConsistency
 = false}))
{
    var bankDB = client.GetDatabase(dbName);
    var accountColl = bankDB.GetCollection<BsonDocument>(collName);
    bankDB.DropCollection(collName);
    accountColl.InsertOne(session, new BsonDocument { {"name", "Alice"}, {"balance",
 1000 } });
```

```
   accountColl.InsertOne(session, new BsonDocument { {"name", "Bob"}, {"balance",
1000 } });

   // start transaction
   var transactionOptions = new TransactionOptions(
           readConcern: ReadConcern.Snapshot,
           writeConcern: WriteConcern.WMajority);
   var result = session.WithTransaction(
       (sess, cancellationtoken) =>
       {
           // deduct $500 from Alice's account
           var aliceBalance = accountColl.Find(sess,
Builders<BsonDocument>.Filter.Eq("name",
"Alice")).FirstOrDefault().GetValue("balance");
           Debug.Assert(aliceBalance >= amountToTransfer);
           var newAliceBalance = aliceBalance.AsInt32 - amountToTransfer;
           accountColl.UpdateOne(sess, Builders<BsonDocument>.Filter.Eq("name",
"Alice"),
                                 Builders<BsonDocument>.Update.Set("balance",
newAliceBalance));
           aliceBalance = accountColl.Find(sess,
Builders<BsonDocument>.Filter.Eq("name",
"Alice")).FirstOrDefault().GetValue("balance");
           Debug.Assert(aliceBalance == newAliceBalance);

           // add $500 from Bob's account
           var bobBalance = accountColl.Find(sess,
Builders<BsonDocument>.Filter.Eq("name",
"Bob")).FirstOrDefault().GetValue("balance");
           var newBobBalance = bobBalance.AsInt32 + amountToTransfer;
           accountColl.UpdateOne(sess, Builders<BsonDocument>.Filter.Eq("name",
"Bob"),
                                 Builders<BsonDocument>.Update.Set("balance",
newBobBalance));
           bobBalance = accountColl.Find(sess,
Builders<BsonDocument>.Filter.Eq("name",
"Bob")).FirstOrDefault().GetValue("balance");
           Debug.Assert(bobBalance == newBobBalance);

           return "Transaction committed";
       }, transactionOptions);
  // check values outside of transaction
   var aliceNewBalance = accountColl.Find(Builders<BsonDocument>.Filter.Eq("name",
"Alice")).FirstOrDefault().GetValue("balance");
```

```
    var bobNewBalance = accountColl.Find(Builders<BsonDocument>.Filter.Eq("name",
 "Bob")).FirstOrDefault().GetValue("balance");
    Debug.Assert(aliceNewBalance ==  500);
    Debug.Assert(bobNewBalance ==  1500);
}
```

Ruby

下列程式碼示範如何使用 Amazon DocumentDB 交易 API 搭配 Ruby。

```
// Ruby Callback API

dbName = "bank"
collName = "account"
amountToTransfer = 500

session = client.start_session(:causal_consistency=> false)
bankDB = Mongo::Database.new(client, dbName)
accountColl = bankDB[collName]
accountColl.drop()

accountColl.insert_one({"name"=>"Alice", "balance"=>1000})
accountColl.insert_one({"name"=>"Bob", "balance"=>1000})

    # start transaction
    session.with_transaction(read_concern: {level: :snapshot}, write_concern:
 {w: :majority}) do
        # deduct $500 from Alice's account
        aliceBalance = accountColl.find({"name"=>"Alice"}, :session=>
 session).first['balance']
        assert aliceBalance >= amountToTransfer
        newAliceBalance = aliceBalance - amountToTransfer
        accountColl.update_one({"name"=>"Alice"}, { "$set" =>
 {"balance"=>newAliceBalance} }, :session=> session)
        aliceBalance = accountColl.find({"name"=>>"Alice"}, :session=>
 session).first['balance']
        assert_equal(newAliceBalance, aliceBalance)

        # add $500 from Bob's account
        bobBalance = accountColl.find({"name"=>"Bob"}, :session=>
 session).first['balance']
        newBobBalance = bobBalance + amountToTransfer
        accountColl.update_one({"name"=>"Bob"}, { "$set" =>
 {"balance"=>newBobBalance} }, :session=> session)
```

```
        bobBalance = accountColl.find({"name"=>"Bob"}, :session=>
 session).first['balance']
        assert_equal(newBobBalance, bobBalance)
    end

   # check results outside of transaction
    aliceBalance = accountColl.find({"name"=>"Alice"}).first['balance']
    bobBalance = accountColl.find({"name"=>"Bob"}).first['balance']
    assert_equal(aliceBalance, 500)
    assert_equal(bobBalance, 1500)

session.end_session
```

Go

下列程式碼示範如何使用 Amazon DocumentDB 交易 API 搭配 Go。

```
// Go - Callback API
type Account struct {
    Name string
    Balance  int
}

ctx := context.TODO()

dbName := "bank"
collName := "account"
amountToTransfer := 500

session, err := client.StartSession(options.Session().SetCausalConsistency(false))
assert.NilError(t, err)
defer session.EndSession(ctx)

bankDB := client.Database(dbName)
accountColl := bankDB.Collection(collName)
accountColl.Drop(ctx)

_, err = accountColl.InsertOne(ctx, bson.M{"name" : "Alice", "balance":1000})
_, err = accountColl.InsertOne(ctx, bson.M{"name" : "Bob", "balance":1000})

transactionOptions := options.Transaction().SetReadConcern(readconcern.Snapshot()).

 SetWriteConcern(writeconcern.New(writeconcern.WMajority()))
```

```
_, err = session.WithTransaction(ctx, func(sessionCtx mongo.SessionContext)
 (interface{}, error) {
    var result Account
    // deduct $500 from Alice's account
    err = accountColl.FindOne(sessionCtx, bson.M{"name": "Alice"}).Decode(&result)
    aliceBalance := result.Balance
    newAliceBalance := aliceBalance - amountToTransfer
    _, err = accountColl.UpdateOne(sessionCtx, bson.M{"name": "Alice"},
 bson.M{"$set": bson.M{"balance": newAliceBalance}})
    err = accountColl.FindOne(sessionCtx, bson.M{"name": "Alice"}).Decode(&result)
    aliceBalance = result.Balance
    assert.Equal(t, aliceBalance, newAliceBalance)

    // add $500 to Bob's account
    err = accountColl.FindOne(sessionCtx, bson.M{"name": "Bob"}).Decode(&result)
    bobBalance := result.Balance
    newBobBalance := bobBalance + amountToTransfer
    _, err = accountColl.UpdateOne(sessionCtx, bson.M{"name": "Bob"}, bson.M{"$set":
 bson.M{"balance": newBobBalance}})
    err = accountColl.FindOne(sessionCtx, bson.M{"name": "Bob"}).Decode(&result)
    bobBalance = result.Balance
    assert.Equal(t, bobBalance, newBobBalance)

    if err != nil {
        return nil, err
    }
    return "transaction committed", err
}, transactionOptions)

// check results outside of transaction
var result Account
err = accountColl.FindOne(ctx, bson.M{"name": "Alice"}).Decode(&result)
aliceNewBalance := result.Balance
err = accountColl.FindOne(ctx, bson.M{"name": "Bob"}).Decode(&result)
bobNewBalance := result.Balance
assert.Equal(t, aliceNewBalance, 500)
assert.Equal(t, bobNewBalance, 1500)
```

Java

下列程式碼示範如何搭配 Java 使用 Amazon DocumentDB 交易 API。

```
// Java (sync) - Callback API
MongoDatabase bankDB = mongoClient.getDatabase("bank");
```

```
MongoCollection accountColl = bankDB.getCollection("account");
accountColl.drop();
int amountToTransfer = 500;

// add sample data
accountColl.insertOne(new Document("name", "Alice").append("balance", 1000));
accountColl.insertOne(new Document("name", "Bob").append("balance", 1000));

TransactionOptions txnOptions = TransactionOptions.builder()
        .readConcern(ReadConcern.SNAPSHOT)
        .writeConcern(WriteConcern.MAJORITY)
        .build();
ClientSessionOptions sessionOptions =
 ClientSessionOptions.builder().causallyConsistent(false).build();
try ( ClientSession clientSession = mongoClient.startSession(sessionOptions) ) {
    clientSession.withTransaction(new TransactionBody<Void>() {
        @Override
        public Void execute() {
            // deduct $500 from Alice's account
            List<Document> documentList = new ArrayList<>();
            accountColl.find(clientSession, new Document("name",
 "Alice")).into(documentList);
            int aliceBalance = (int) documentList.get(0).get("balance");
            int newAliceBalance = aliceBalance - amountToTransfer;

            accountColl.updateOne(clientSession, new Document("name", "Alice"), new
 Document("$set", new Document("balance", newAliceBalance)));

            // check Alice's new balance
            documentList = new ArrayList<>();
            accountColl.find(clientSession, new Document("name",
 "Alice")).into(documentList);
            int updatedBalance = (int) documentList.get(0).get("balance");
            Assert.assertEquals(updatedBalance, newAliceBalance);

            // add $500 to Bob's account
            documentList = new ArrayList<>();
            accountColl.find(clientSession, new Document("name",
 "Bob")).into(documentList);
            int bobBalance = (int) documentList.get(0).get("balance");
            int newBobBalance = bobBalance + amountToTransfer;

            accountColl.updateOne(clientSession, new Document("name", "Bob"), new
 Document("$set", new Document("balance", newBobBalance)));
```

```
            // check Bob's new balance
            documentList = new ArrayList<>();
            accountColl.find(clientSession, new Document("name",
 "Bob")).into(documentList);
            updatedBalance = (int) documentList.get(0).get("balance");
            Assert.assertEquals(updatedBalance, newBobBalance);

            return null;
        }
    }, txnOptions);
}
```

C

下列程式碼示範如何使用 Amazon DocumentDB 交易 API 搭配 C。

```
// Sample Code for C with Callback

#include <bson.h>
#include <mongoc.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>

typedef struct {
    int64_t balance;
    bson_t *account;
    bson_t *opts;
    mongoc_collection_t *collection;
} ctx_t;

bool callback_session (mongoc_client_session_t *session, void *ctx, bson_t **reply,
 bson_error_t *error)
{
    bool r = true;
    ctx_t *data = (ctx_t *) ctx;
    bson_t local_reply;
    bson_t *selector = data->account;
    bson_t *update = BCON_NEW ("$set", "{", "balance", BCON_INT64 (data->balance),
 "}");

    mongoc_collection_update_one (data->collection, selector, update, data->opts,
 &local_reply, error);
```

```
    *reply = bson_copy (&local_reply);
    bson_destroy (&local_reply);
    bson_destroy (update);
    return r;
}

void test_callback_money_transfer(mongoc_client_t* client, mongoc_collection_t*
 collection, int amount_to_transfer){

    bson_t reply;
    bool r = true;
    const bson_t *doc;
    bson_iter_t iter;
    ctx_t alice_ctx;
    ctx_t bob_ctx;
    bson_error_t error;

    // find query
    bson_t *alice_query = bson_new ();
    BSON_APPEND_UTF8(alice_query, "name", "Alice");

    bson_t *bob_query = bson_new ();
    BSON_APPEND_UTF8(bob_query, "name", "Bob");

    // create session
    // set causal consistency to false
    mongoc_session_opt_t *session_opts = mongoc_session_opts_new ();
    mongoc_session_opts_set_causal_consistency (session_opts, false);
    // start the session
    mongoc_client_session_t *client_session = mongoc_client_start_session (client,
 session_opts, &error);

    // add session to options
    bson_t *opts = bson_new();
    mongoc_client_session_append (client_session, opts, &error);

    // deduct 500 from Alice
    // find account balance of Alice
    mongoc_cursor_t *cursor = mongoc_collection_find_with_opts (collection,
 alice_query, NULL, NULL);
    mongoc_cursor_next (cursor, &doc);
    bson_iter_init (&iter, doc);
    bson_iter_find (&iter, "balance");
```

```
    int64_t alice_balance = (bson_iter_value (&iter))->value.v_int64;
    assert(alice_balance >= amount_to_transfer);
    int64_t new_alice_balance = alice_balance - amount_to_transfer;

    // set variables which will be used by callback function
    alice_ctx.collection = collection;
    alice_ctx.opts = opts;
    alice_ctx.balance = new_alice_balance;
    alice_ctx.account = alice_query;

    // callback
    r = mongoc_client_session_with_transaction (client_session, &callback_session,
NULL, &alice_ctx, &reply, &error);
    assert(r);

    // find account balance of Alice after transaction
    cursor = mongoc_collection_find_with_opts (collection, alice_query, NULL, NULL);
    mongoc_cursor_next (cursor, &doc);
    bson_iter_init (&iter, doc);
    bson_iter_find (&iter, "balance");
    alice_balance = (bson_iter_value (&iter))->value.v_int64;
    assert(alice_balance == new_alice_balance);
    assert(alice_balance == 500);

        // add 500 to bob's balance
    // find account balance of Bob
    cursor = mongoc_collection_find_with_opts (collection, bob_query, NULL, NULL);
    mongoc_cursor_next (cursor, &doc);
    bson_iter_init (&iter, doc);
    bson_iter_find (&iter, "balance");
    int64_t bob_balance = (bson_iter_value (&iter))->value.v_int64;
    int64_t new_bob_balance = bob_balance + amount_to_transfer;

    bob_ctx.collection = collection;
    bob_ctx.opts = opts;
    bob_ctx.balance = new_bob_balance;
    bob_ctx.account = bob_query;

    // set read & write concern
    mongoc_read_concern_t *read_concern = mongoc_read_concern_new ();
    mongoc_write_concern_t *write_concern = mongoc_write_concern_new ();
    mongoc_transaction_opt_t *txn_opts = mongoc_transaction_opts_new ();

    mongoc_write_concern_set_w(write_concern, MONGOC_WRITE_CONCERN_W_MAJORITY);
```

```
    mongoc_read_concern_set_level(read_concern, MONGOC_READ_CONCERN_LEVEL_SNAPSHOT);
    mongoc_transaction_opts_set_write_concern (txn_opts, write_concern);
    mongoc_transaction_opts_set_read_concern (txn_opts, read_concern);

    // callback
    r = mongoc_client_session_with_transaction (client_session, &callback_session,
 txn_opts, &bob_ctx, &reply, &error);
    assert(r);

 // find account balance of Bob after transaction
    cursor = mongoc_collection_find_with_opts (collection, bob_query, NULL, NULL);
    mongoc_cursor_next (cursor, &doc);
    bson_iter_init (&iter, doc);
    bson_iter_find (&iter, "balance");
    bob_balance = (bson_iter_value (&iter))->value.v_int64;
    assert(bob_balance == new_bob_balance);
    assert(bob_balance == 1500);

    // cleanup
    bson_destroy(alice_query);
    bson_destroy(bob_query);
    mongoc_client_session_destroy(client_session);
    bson_destroy(opts);
    mongoc_transaction_opts_destroy(txn_opts);
    mongoc_read_concern_destroy(read_concern);
    mongoc_write_concern_destroy(write_concern);
    mongoc_cursor_destroy(cursor);
    bson_destroy(doc);
}
int main(int argc, char* argv[]) {
    mongoc_init ();
    mongoc_client_t* client = mongoc_client_new (<connection uri>);
    bson_error_t error;

    // connect to bank db
    mongoc_database_t *database = mongoc_client_get_database (client, "bank");
    // access account collection
    mongoc_collection_t* collection = mongoc_client_get_collection(client, "bank",
 "account");
    // set amount to transfer
    int64_t amount_to_transfer = 500;
    // delete the collection if already existing
    mongoc_collection_drop(collection, &error);
```

```
    // open Alice account
    bson_t *alice_account = bson_new ();
    BSON_APPEND_UTF8(alice_account, "name", "Alice");
    BSON_APPEND_INT64(alice_account, "balance", 1000);

    // open Bob account
    bson_t *bob_account = bson_new ();
    BSON_APPEND_UTF8(bob_account, "name", "Bob");
    BSON_APPEND_INT64(bob_account, "balance", 1000);

    bool r = true;

    r = mongoc_collection_insert_one(collection, alice_account, NULL, NULL, &error);
    if (!r) {printf("Error encountered:%s", error.message);}
    r = mongoc_collection_insert_one(collection, bob_account, NULL, NULL, &error);
    if (!r) {printf("Error encountered:%s", error.message);}

    test_callback_money_transfer(client, collection, amount_to_transfer);

}
```

Python

下列程式碼示範如何使用 Amazon DocumentDB 交易 API 搭配 Python。

```
// Sample Python code with callback api

import pymongo

def callback(session, balance, query):
    collection.update_one(query, {'$set': {"balance": balance}}, session=session)

client = pymongo.MongoClient(<connection uri>)
rc_snapshot = pymongo.read_concern.ReadConcern('snapshot')
wc_majority = pymongo.write_concern.WriteConcern('majority')

# To start, drop and create an account collection and insert balances for both Alice
 and Bob
collection = client.get_database("bank").get_collection("account")
collection.drop()
collection.insert_one({"_id": 1, "name": "Alice", "balance": 1000})
collection.insert_one({"_id": 2, "name": "Bob", "balance": 1000})

amount_to_transfer = 500
```

```
# deduct 500 from Alice's account
alice_balance = collection.find_one({"name": "Alice"}).get("balance")
assert alice_balance >= amount_to_transfer
new_alice_balance = alice_balance - amount_to_transfer

with client.start_session({'causalConsistency':False}) as session:
    session.with_transaction(lambda s: callback(s, new_alice_balance, {"name":
 "Alice"}), read_concern=rc_snapshot, write_concern=wc_majority)

updated_alice_balance = collection.find_one({"name": "Alice"}).get("balance")
assert updated_alice_balance == new_alice_balance

# add 500 to Bob's account
bob_balance = collection.find_one({"name": "Bob"}).get("balance")
assert bob_balance >= amount_to_transfer
new_bob_balance = bob_balance + amount_to_transfer

with client.start_session({'causalConsistency':False}) as session:
    session.with_transaction(lambda s: callback(s, new_bob_balance, {"name":
 "Bob"}), read_concern=rc_snapshot, write_concern=wc_majority)

updated_bob_balance = collection.find_one({"name": "Bob"}).get("balance")
assert updated_bob_balance == new_bob_balance
```

核心 API 的交易 API 範例

Javascript

下列程式碼示範如何搭配 Javascript 使用 Amazon DocumentDB 交易 API。

```
// *** Transfer $500 from Alice to Bob inside a transaction: Success ***
// Setup bank account for Alice and Bob. Each have $1000 in their account
var databaseName = "bank";
var collectionName = "account";
var amountToTransfer = 500;

var session = db.getMongo().startSession({causalConsistency: false});
var bankDB = session.getDatabase(databaseName);
var accountColl = bankDB[collectionName];
accountColl.drop();
```

```
accountColl.insert({name: "Alice", balance: 1000});
accountColl.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountColl.find({"name": "Alice"}).next().balance;
assert(aliceBalance >= amountToTransfer);
var newAliceBalance = aliceBalance - amountToTransfer;
accountColl.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountColl.find({"name": "Alice"}).next().balance;
assert.eq(newAliceBalance, findAliceBalance);

// add $500 to Bob's account
var bobBalance = accountColl.find({"name": "Bob"}).next().balance;
var newBobBalance = bobBalance + amountToTransfer;
accountColl.update({"name": "Bob"},{"$set": {"balance": newBobBalance}});
var findBobBalance = accountColl.find({"name": "Bob"}).next().balance;
assert.eq(newBobBalance, findBobBalance);

session.commitTransaction();

accountColl.find();
```

C#

下列程式碼示範如何使用 Amazon DocumentDB 交易 API 搭配 C#。

```
// C# Core API

public void TransferMoneyWithRetry(IMongoCollection<bSondocument> accountColl,
 IClientSessionHandle session)
{
    var amountToTransfer = 500;

    // start transaction
   var transactionOptions = new TransactionOptions(
                readConcern: ReadConcern.Snapshot,
                writeConcern: WriteConcern.WMajority);
    session.StartTransaction(transactionOptions);
   try
    {
        // deduct $500 from Alice's account
```

```
        var aliceBalance = accountColl.Find(session,
 Builders<bSondocument>.Filter.Eq("name",
 "Alice")).FirstOrDefault().GetValue("balance");
        Debug.Assert(aliceBalance >= amountToTransfer);
        var newAliceBalance = aliceBalance.AsInt32 - amountToTransfer;
        accountColl.UpdateOne(session, Builders<bSondocument>.Filter.Eq("name",
 "Alice"),
                                Builders<bSondocument>.Update.Set("balance",
 newAliceBalance));
        aliceBalance = accountColl.Find(session,
 Builders<bSondocument>.Filter.Eq("name",
 "Alice")).FirstOrDefault().GetValue("balance");
        Debug.Assert(aliceBalance == newAliceBalance);

        // add $500 from Bob's account
        var bobBalance = accountColl.Find(session,
 Builders<bSondocument>.Filter.Eq("name",
 "Bob")).FirstOrDefault().GetValue("balance");
        var newBobBalance = bobBalance.AsInt32 + amountToTransfer;
        accountColl.UpdateOne(session, Builders<bSondocument>.Filter.Eq("name",
 "Bob"),
                                Builders<bSondocument>.Update.Set("balance",
 newBobBalance));
        bobBalance = accountColl.Find(session,
 Builders<bSondocument>.Filter.Eq("name",
 "Bob")).FirstOrDefault().GetValue("balance");
        Debug.Assert(bobBalance == newBobBalance);

    }
    catch (Exception e)
    {
        session.AbortTransaction();
        throw;
    }

    session.CommitTransaction();
 }

}
public void DoTransactionWithRetry(MongoClient client)
{
    var dbName = "bank";
    var collName = "account";
```

```
   using (var session = client.StartSession(new
ClientSessionOptions{CausalConsistency = false}))
   {
       try
       {
           var bankDB = client.GetDatabase(dbName);
           var accountColl = bankDB.GetCollection<bSondocument>(collName);
           bankDB.DropCollection(collName);
           accountColl.InsertOne(session, new BsonDocument { {"name", "Alice"},
{"balance", 1000 } });
           accountColl.InsertOne(session, new BsonDocument { {"name", "Bob"},
{"balance", 1000 } });

           while(true) {
               try
               {
                       TransferMoneyWithRetry(accountColl, session);
                       break;
               }
               catch (MongoException e)
               {
                   if(e.HasErrorLabel("TransientTransactionError"))
                   {
                       continue;
                   }
                   else
                   {
                       throw;
                   }
               }
           }

           // check values outside of transaction
           var aliceNewBalance =
accountColl.Find(Builders<bSondocument>.Filter.Eq("name",
"Alice")).FirstOrDefault().GetValue("balance");
           var bobNewBalance =
accountColl.Find(Builders<bSondocument>.Filter.Eq("name",
"Bob")).FirstOrDefault().GetValue("balance");
           Debug.Assert(aliceNewBalance ==  500);
           Debug.Assert(bobNewBalance ==  1500);
       }
       catch (Exception e)
       {
```

```
            Console.WriteLine("Error running transaction: " + e.Message);
        }
    }
}
```

Ruby

下列程式碼示範如何使用 Amazon DocumentDB 交易 API 搭配 Ruby。

```
# Ruby Core API

def transfer_money_w_retry(session, accountColl)
    amountToTransfer = 500

    session.start_transaction(read_concern: {level: :snapshot}, write_concern:
 {w: :majority})
    # deduct $500 from Alice's account
    aliceBalance = accountColl.find({"name"=>"Alice"}, :session=>
 session).first['balance']
    assert aliceBalance >= amountToTransfer
    newAliceBalance = aliceBalance - amountToTransfer
    accountColl.update_one({"name"=>"Alice"}, { "$set" =>
 {"balance"=>newAliceBalance} }, :session=> session)
    aliceBalance = accountColl.find({"name"=>"Alice"}, :session=>
 session).first['balance']
    assert_equal(newAliceBalance, aliceBalance)

    # add $500 to Bob's account
    bobBalance = accountColl.find({"name"=>"Bob"}, :session=>
 session).first['balance']
    newBobBalance = bobBalance + amountToTransfer
    accountColl.update_one({"name"=>"Bob"}, { "$set" =>
 {"balance"=>newBobBalance} }, :session=> session)
    bobBalance = accountColl.find({"name"=>"Bob"}, :session=>
 session).first['balance']
    assert_equal(newBobBalance, bobBalance)

    session.commit_transaction

end

def do_txn_w_retry(client)
     dbName = "bank"
    collName = "account"
```

```
    session = client.start_session(:causal_consistency=> false)
    bankDB = Mongo::Database.new(client, dbName)
    accountColl = bankDB[collName]
    accountColl.drop()

    accountColl.insert_one({"name"=>"Alice", "balance"=>1000})
    accountColl.insert_one({"name"=>"Bob", "balance"=>1000})

    begin
        transferMoneyWithRetry(session, accountColl)
        puts "transaction committed"
    rescue Mongo::Error => e
        if e.label?('TransientTransactionError')
            retry
        else
            puts "transaction failed"
            raise
        end
    end

    # check results outside of transaction
    aliceBalance = accountColl.find({"name"=>"Alice"}).first['balance']
    bobBalance = accountColl.find({"name"=>"Bob"}).first['balance']
    assert_equal(aliceBalance, 500)
    assert_equal(bobBalance, 1500)

end
```

Go

下列程式碼示範如何使用 Amazon DocumentDB 交易 API 搭配 Go。

```
// Go - Core API
type Account struct {
    Name string
    Balance  int
}

func transferMoneyWithRetry(sessionContext mongo.SessionContext, accountColl
 *mongo.Collection, t *testing.T) error {
    amountToTransfer := 500
```

```
    transactionOptions :=
 options.Transaction().SetReadConcern(readconcern.Snapshot()).

 SetWriteConcern(writeconcern.New(writeconcern.WMajority()))
    if err := sessionContext.StartTransaction(transactionOptions); err != nil {
        panic(err)
    }

    var result Account
    // deduct $500 from Alice's account
    err := accountColl.FindOne(sessionContext, bson.M{"name":
 "Alice"}).Decode(&result)
    aliceBalance := result.Balance
    newAliceBalance := aliceBalance - amountToTransfer
    _, err = accountColl.UpdateOne(sessionContext, bson.M{"name": "Alice"},
 bson.M{"$set": bson.M{"balance": newAliceBalance}})
    if err != nil {
        sessionContext.AbortTransaction(sessionContext)
    }
    err = accountColl.FindOne(sessionContext, bson.M{"name":
 "Alice"}).Decode(&result)
    aliceBalance = result.Balance
    assert.Equal(t, aliceBalance, newAliceBalance)

    // add $500 to Bob's account
    err = accountColl.FindOne(sessionContext, bson.M{"name": "Bob"}).Decode(&result)
    bobBalance := result.Balance
    newBobBalance := bobBalance + amountToTransfer
    _, err = accountColl.UpdateOne(sessionContext, bson.M{"name": "Bob"},
 bson.M{"$set": bson.M{"balance": newBobBalance}})
    if err != nil {
        sessionContext.AbortTransaction(sessionContext)
    }
    err = accountColl.FindOne(sessionContext, bson.M{"name": "Bob"}).Decode(&result)
    bobBalance = result.Balance
    assert.Equal(t, bobBalance, newBobBalance)

    err = sessionContext.CommitTransaction(sessionContext)
    return err
}

func doTransactionWithRetry(t *testing.T) {
    ctx := context.TODO()
```

```
    dbName := "bank"
    collName := "account"
    bankDB := client.Database(dbName)
    accountColl := bankDB.Collection(collName)

    client.UseSessionWithOptions(ctx, options.Session().SetCausalConsistency(false),
 func(sessionContext mongo.SessionContext) error {
        accountColl.Drop(ctx)
        accountColl.InsertOne(sessionContext, bson.M{"name" : "Alice",
 "balance":1000})
        accountColl.InsertOne(sessionContext, bson.M{"name" : "Bob",
 "balance":1000})
        for {
            err := transferMoneyWithRetry(sessionContext, accountColl, t)
            if err == nil {
                println("transaction committed")
                return nil
            }
            if mongoErr := err.(mongo.CommandError);
 mongoErr.HasErrorLabel("TransientTransactionError") {
                continue
            }
            println("transaction failed")
            return err
        }
    })

    // check results outside of transaction
    var result Account
    accountColl.FindOne(ctx, bson.M{"name": "Alice"}).Decode(&result)
    aliceBalance := result.Balance
    assert.Equal(t, aliceBalance, 500)
    accountColl.FindOne(ctx, bson.M{"name": "Bob"}).Decode(&result)
    bobBalance := result.Balance
    assert.Equal(t, bobBalance, 1500)
}
```

Java

下列程式碼示範如何搭配 Java 使用 Amazon DocumentDB 交易 API。

```
// Java (sync) - Core API

public void transferMoneyWithRetry() {
```

```
   // connect to server
    MongoClientURI mongoURI = new MongoClientURI(uri);
    MongoClient mongoClient = new MongoClient(mongoURI);

    MongoDatabase bankDB = mongoClient.getDatabase("bank");
    MongoCollection accountColl = bankDB.getCollection("account");
    accountColl.drop();

   // insert some sample data
    accountColl.insertOne(new Document("name", "Alice").append("balance", 1000));
    accountColl.insertOne(new Document("name", "Bob").append("balance", 1000));

    while (true) {
        try {
            doTransferMoneyWithRetry(accountColl, mongoClient);
            break;
        } catch (MongoException e) {
            if (e.hasErrorLabel(MongoException.TRANSIENT_TRANSACTION_ERROR_LABEL)) {
                continue;
            } else {
                throw e;
            }
        }
    }
}

public void doTransferMoneyWithRetry(MongoCollection accountColl, MongoClient
 mongoClient) {
    int amountToTransfer = 500;

   TransactionOptions txnOptions = TransactionOptions.builder()
      .readConcern(ReadConcern.SNAPSHOT)
      .writeConcern(WriteConcern.MAJORITY)
      .build();
    ClientSessionOptions sessionOptions =
 ClientSessionOptions.builder().causallyConsistent(false).build();
    try ( ClientSession clientSession = mongoClient.startSession(sessionOptions) ) {
        clientSession.startTransaction(txnOptions);

        // deduct $500 from Alice's account
        List<Document> documentList = new ArrayList<>();
        accountColl.find(clientSession, new Document("name",
 "Alice")).into(documentList);
        int aliceBalance = (int) documentList.get(0).get("balance");
```

```
        Assert.assertTrue(aliceBalance >= amountToTransfer);
        int newAliceBalance = aliceBalance - amountToTransfer;
        accountColl.updateOne(clientSession, new Document("name", "Alice"), new
 Document("$set", new Document("balance", newAliceBalance)));

        // check Alice's new balance
        documentList = new ArrayList<>();
        accountColl.find(clientSession, new Document("name",
 "Alice")).into(documentList);
        int updatedBalance = (int) documentList.get(0).get("balance");
        Assert.assertEquals(updatedBalance, newAliceBalance);

        // add $500 to Bob's account
        documentList = new ArrayList<>();
        accountColl.find(clientSession, new Document("name",
 "Bob")).into(documentList);
        int bobBalance = (int) documentList.get(0).get("balance");
        int newBobBalance = bobBalance + amountToTransfer;
        accountColl.updateOne(clientSession, new Document("name", "Bob"), new
 Document("$set", new Document("balance", newBobBalance)));

        // check Bob's new balance
        documentList = new ArrayList<>();
        accountColl.find(clientSession, new Document("name",
 "Bob")).into(documentList);
        updatedBalance = (int) documentList.get(0).get("balance");
        Assert.assertEquals(updatedBalance, newBobBalance);

        // commit transaction
        clientSession.commitTransaction();
    }
}
// Java (async) -- Core API
public void transferMoneyWithRetry() {
    // connect to the server
    MongoClient mongoClient = MongoClients.create(uri);

    MongoDatabase bankDB = mongoClient.getDatabase("bank");
    MongoCollection accountColl = bankDB.getCollection("account");
    SubscriberLatchWrapper<Void> dropCallback = new SubscriberLatchWrapper<>();
    mongoClient.getDatabase("bank").drop().subscribe(dropCallback);
    dropCallback.await();

    // insert some sample data
```

```
    SubscriberLatchWrapper<InsertOneResult> insertionCallback = new
 SubscriberLatchWrapper<>();
    accountColl.insertOne(new Document("name", "Alice").append("balance",
 1000)).subscribe(insertionCallback);
    insertionCallback.await();

    insertionCallback = new SubscriberLatchWrapper<>();
    accountColl.insertOne(new Document("name", "Bob").append("balance",
 1000)).subscribe(insertionCallback);;
    insertionCallback.await();

    while (true) {
        try {
            doTransferMoneyWithRetry(accountColl, mongoClient);
            break;
        } catch (MongoException e) {
            if (e.hasErrorLabel(MongoException.TRANSIENT_TRANSACTION_ERROR_LABEL)) {
                continue;
            } else {
                throw e;
            }
        }
    }
}

public void doTransferMoneyWithRetry(MongoCollection accountColl, MongoClient
 mongoClient) {
    int amountToTransfer = 500;

    // start the transaction
    TransactionOptions txnOptions = TransactionOptions.builder()
            .readConcern(ReadConcern.SNAPSHOT)
            .writeConcern(WriteConcern.MAJORITY)
            .build();
    ClientSessionOptions sessionOptions =
 ClientSessionOptions.builder().causallyConsistent(false).build();

    SubscriberLatchWrapper<ClientSession> sessionCallback = new
 SubscriberLatchWrapper<>();
    mongoClient.startSession(sessionOptions).subscribe(sessionCallback);
    ClientSession session = sessionCallback.get().get(0);
    session.startTransaction(txnOptions);

    // deduct $500 from Alice's account
```

```
    SubscriberLatchWrapper<Document> findCallback = new SubscriberLatchWrapper<>();
    accountColl.find(session, new Document("name",
"Alice")).first().subscribe(findCallback);
    Document documentFound = findCallback.get().get(0);
    int aliceBalance = (int) documentFound.get("balance");
    int newAliceBalance = aliceBalance - amountToTransfer;

    SubscriberLatchWrapper<UpdateResult> updateCallback = new
SubscriberLatchWrapper<>();
    accountColl.updateOne(session, new Document("name",
"Alice"), new Document("$set", new Document("balance",
newAliceBalance))).subscribe(updateCallback);
    updateCallback.await();

    // check Alice's new balance
    findCallback = new SubscriberLatchWrapper<>();
    accountColl.find(session, new Document("name",
"Alice")).first().subscribe(findCallback);
    documentFound = findCallback.get().get(0);
    int updatedBalance = (int) documentFound.get("balance");
    Assert.assertEquals(updatedBalance, newAliceBalance);

    // add $500 to Bob's account
    findCallback = new SubscriberLatchWrapper<>();
    accountColl.find(session, new Document("name",
"Bob")).first().subscribe(findCallback);
    documentFound = findCallback.get().get(0);
    int bobBalance = (int) documentFound.get("balance");
    int newBobBalance = bobBalance + amountToTransfer;

    updateCallback = new SubscriberLatchWrapper<>();
    accountColl.updateOne(session, new Document("name", "Bob"), new Document("$set",
new Document("balance", newBobBalance))).subscribe(updateCallback);
    updateCallback.await();

    // check Bob's new balance
    findCallback = new SubscriberLatchWrapper<>();
    accountColl.find(session, new Document("name",
"Bob")).first().subscribe(findCallback);
    documentFound = findCallback.get().get(0);
    updatedBalance = (int) documentFound.get("balance");
    Assert.assertEquals(updatedBalance, newBobBalance);

    // commit the transaction
```

```
    SubscriberLatchWrapper<Void> transactionCallback = new
 SubscriberLatchWrapper<>();
    session.commitTransaction().subscribe(transactionCallback);
    transactionCallback.await();
}

public class SubscriberLatchWrapper<T> implements Subscriber<T> {

    /**
     * A Subscriber that stores the publishers results and provides a latch so can
 block on completion.
     *
     * @param <T> The publishers result type
     */
    private final List<T> received;
    private final List<RuntimeException> errors;
    private final CountDownLatch latch;
    private volatile Subscription subscription;
    private volatile boolean completed;

    /**
     * Construct an instance
     */
    public SubscriberLatchWrapper() {
        this.received = new ArrayList<>();
        this.errors = new ArrayList<>();
        this.latch = new CountDownLatch(1);
    }

    @Override
    public void onSubscribe(final Subscription s) {
        subscription = s;
        subscription.request(Integer.MAX_VALUE);
    }

    @Override
    public void onNext(final T t) {
        received.add(t);
    }

    @Override
    public void onError(final Throwable t) {
        if (t instanceof RuntimeException) {
            errors.add((RuntimeException) t);
```

```
        } else {
            errors.add(new RuntimeException("Unexpected exception", t));
        }
        onComplete();
    }

    @Override
    public void onComplete() {
        completed = true;
        subscription.cancel();
        latch.countDown();
    }

    /**
     * Get received elements
     *
     * @return the list of received elements
     */
    public List<T> getReceived() {
        return received;
    }

    /**
     * Get received elements.
     *
     * @return the list of receive elements
     */
    public List<T> get() {
        return await().getReceived();
    }

    /**
     * Await completion or error
     *
     * @return this
     */
    public SubscriberLatchWrapper<T> await() {
        subscription.request(Integer.MAX_VALUE);
        try {
            if (!latch.await(300, TimeUnit.SECONDS)) {
                throw new MongoTimeoutException("Publisher onComplete timed out for
300 seconds");
            }
        } catch (InterruptedException e) {
```

```
            throw new MongoInterruptedException("Interrupted waiting for
 observeration", e);
        }
        if (!errors.isEmpty()) {
            throw errors.get(0);
        }
        return this;
    }

    public boolean getCompleted() {
        return this.completed;
    }

    public void close() {
        subscription.cancel();
        received.clear();
    }
}
```

C

下列程式碼示範如何使用 Amazon DocumentDB 交易 API 搭配 C。

```
// Sample C code with core session

bool core_session(mongoc_client_session_t *client_session, mongoc_collection_t*
 collection, bson_t *selector, int64_t balance){
    bool r = true;
    bson_error_t error;
    bson_t *opts = bson_new();
    bson_t *update = BCON_NEW ("$set", "{", "balance", BCON_INT64 (balance), "}");

    // set read & write concern
    mongoc_read_concern_t *read_concern = mongoc_read_concern_new ();
    mongoc_write_concern_t *write_concern = mongoc_write_concern_new ();
    mongoc_transaction_opt_t *txn_opts = mongoc_transaction_opts_new ();

    mongoc_write_concern_set_w(write_concern, MONGOC_WRITE_CONCERN_W_MAJORITY);
    mongoc_read_concern_set_level(read_concern, MONGOC_READ_CONCERN_LEVEL_SNAPSHOT);
    mongoc_transaction_opts_set_write_concern (txn_opts, write_concern);
    mongoc_transaction_opts_set_read_concern (txn_opts, read_concern);

    mongoc_client_session_start_transaction (client_session, txn_opts, &error);
```

```
    mongoc_client_session_append (client_session, opts, &error);

    r = mongoc_collection_update_one (collection, selector, update, opts, NULL,
 &error);

    mongoc_client_session_commit_transaction (client_session, NULL, &error);
    bson_destroy (opts);
    mongoc_transaction_opts_destroy(txn_opts);
    mongoc_read_concern_destroy(read_concern);
    mongoc_write_concern_destroy(write_concern);
    bson_destroy (update);
    return r;
}

void test_core_money_transfer(mongoc_client_t* client, mongoc_collection_t*
 collection, int amount_to_transfer){

    bson_t reply;
    bool r = true;
    const bson_t *doc;
    bson_iter_t iter;
    bson_error_t error;

    // find query
    bson_t *alice_query = bson_new ();
    BSON_APPEND_UTF8(alice_query, "name", "Alice");

    bson_t *bob_query = bson_new ();
    BSON_APPEND_UTF8(bob_query, "name", "Bob");

    // create session
    // set causal consistency to false
    mongoc_session_opt_t *session_opts = mongoc_session_opts_new ();
    mongoc_session_opts_set_causal_consistency (session_opts, false);
    // start the session
    mongoc_client_session_t *client_session = mongoc_client_start_session (client,
 session_opts, &error);

    // add session to options
    bson_t *opts = bson_new();
    mongoc_client_session_append (client_session, opts, &error);

    // deduct 500 from Alice
    // find account balance of Alice
```

```
   mongoc_cursor_t *cursor = mongoc_collection_find_with_opts (collection,
alice_query, NULL, NULL);
   mongoc_cursor_next (cursor, &doc);
   bson_iter_init (&iter, doc);
   bson_iter_find (&iter, "balance");
   int64_t alice_balance = (bson_iter_value (&iter))->value.v_int64;
   assert(alice_balance >= amount_to_transfer);
   int64_t new_alice_balance = alice_balance - amount_to_transfer;

   // core
   r = core_session (client_session, collection, alice_query, new_alice_balance);
   assert(r);

   // find account balance of Alice after transaction
   cursor = mongoc_collection_find_with_opts (collection, alice_query, NULL, NULL);
   mongoc_cursor_next (cursor, &doc);
   bson_iter_init (&iter, doc);
   bson_iter_find (&iter, "balance");
   alice_balance = (bson_iter_value (&iter))->value.v_int64;
   assert(alice_balance == new_alice_balance);
   assert(alice_balance == 500);

   // add 500 to Bob's balance
   // find account balance of Bob
   cursor = mongoc_collection_find_with_opts (collection, bob_query, NULL, NULL);
   mongoc_cursor_next (cursor, &doc);
   bson_iter_init (&iter, doc);
   bson_iter_find (&iter, "balance");
   int64_t bob_balance = (bson_iter_value (&iter))->value.v_int64;
   int64_t new_bob_balance = bob_balance + amount_to_transfer;

   //core
   r = core_session (client_session, collection, bob_query, new_bob_balance);
   assert(r);

   // find account balance of Bob after transaction
   cursor = mongoc_collection_find_with_opts (collection, bob_query, NULL, NULL);
   mongoc_cursor_next (cursor, &doc);
   bson_iter_init (&iter, doc);
   bson_iter_find (&iter, "balance");
   bob_balance = (bson_iter_value (&iter))->value.v_int64;
   assert(bob_balance == new_bob_balance);
   assert(bob_balance == 1500);
```

```
    // cleanup
    bson_destroy(alice_query);
    bson_destroy(bob_query);
    mongoc_client_session_destroy(client_session);
    bson_destroy(opts);
    mongoc_cursor_destroy(cursor);
    bson_destroy(doc);
}

int main(int argc, char* argv[]) {
    mongoc_init ();
    mongoc_client_t* client = mongoc_client_new (<connection uri>);
    bson_error_t error;

    // connect to bank db
    mongoc_database_t *database = mongoc_client_get_database (client, "bank");
    // access account collection
    mongoc_collection_t* collection = mongoc_client_get_collection(client, "bank",
 "account");
    // set amount to transfer
    int64_t amount_to_transfer = 500;
    // delete the collection if already existing
    mongoc_collection_drop(collection, &error);

    // open Alice account
    bson_t *alice_account = bson_new ();
    BSON_APPEND_UTF8(alice_account, "name", "Alice");
    BSON_APPEND_INT64(alice_account, "balance", 1000);

    // open Bob account
    bson_t *bob_account = bson_new ();
    BSON_APPEND_UTF8(bob_account, "name", "Bob");
    BSON_APPEND_INT64(bob_account, "balance", 1000);

    bool r = true;

    r = mongoc_collection_insert_one(collection, alice_account, NULL, NULL, &error);
    if (!r) {printf("Error encountered:%s", error.message);}
    r = mongoc_collection_insert_one(collection, bob_account, NULL, NULL, &error);
    if (!r) {printf("Error encountered:%s", error.message);}

    test_core_money_transfer(client, collection, amount_to_transfer);
```

```
}
```

Scala

下列程式碼示範如何搭配 Scala 使用 Amazon DocumentDB 交易 API。

```
// Scala Core API
def transferMoneyWithRetry(sessionObservable: SingleObservable[ClientSession] ,
 database: MongoDatabase ): Unit = {
    val accountColl = database.getCollection("account")
    var amountToTransfer = 500

    var transactionObservable: Observable[ClientSession] =
 sessionObservable.map(clientSession => {
    clientSession.startTransaction()

    // deduct $500 from Alice's account
    var aliceBalance = accountColl.find(clientSession, Document("name" ->
 "Alice")).await().head.getInteger("balance")
    assert(aliceBalance >= amountToTransfer)
    var newAliceBalance = aliceBalance - amountToTransfer
    accountColl.updateOne(clientSession, Document("name" -> "Alice"),
 Document("$set" -> Document("balance" -> newAliceBalance))).await()
    aliceBalance = accountColl.find(clientSession, Document("name" ->
 "Alice")).await().head.getInteger("balance")
    assert(aliceBalance == newAliceBalance)

    // add $500 to Bob's account
    var bobBalance = accountColl.find(clientSession, Document("name" ->
 "Bob")).await().head.getInteger("balance")
    var newBobBalance = bobBalance + amountToTransfer
    accountColl.updateOne(clientSession, Document("name" -> "Bob"), Document("$set"
 -> Document("balance" -> newBobBalance))).await()
    bobBalance = accountColl.find(clientSession, Document("name" ->
 "Bob")).await().head.getInteger("balance")
    assert(bobBalance == newBobBalance)

    clientSession
    })

    transactionObservable.flatMap(clientSession =>
 clientSession.commitTransaction()).await()
}
```

```
def doTransactionWithRetry(): Unit = {
    val client: MongoClient = MongoClientWrapper.getMongoClient()
    val database: MongoDatabase = client.getDatabase("bank")
    val accountColl = database.getCollection("account")
    accountColl.drop().await()

    val sessionOptions =
 ClientSessionOptions.builder().causallyConsistent(false).build()
    var  sessionObservable: SingleObservable[ClientSession] =
 client.startSession(sessionOptions)
    accountColl.insertOne(Document("name" -> "Alice", "balance" -> 1000)).await()
    accountColl.insertOne(Document("name" -> "Bob", "balance" -> 1000)).await()

    var retry = true
    while (retry) {
        try {
        transferMoneyWithRetry(sessionObservable, database)
        println("transaction committed")
        retry = false
        }
        catch {
        case e: MongoException if
 e.hasErrorLabel(MongoException.TRANSIENT_TRANSACTION_ERROR_LABEL) => {
            println("retrying transaction")
        }
        case other: Throwable => {
            println("transaction failed")
            retry = false
            throw other

        }
        }
    }

    // check results outside of transaction
    assert(accountColl.find(Document("name" ->
 "Alice")).results().head.getInteger("balance") == 500)
    assert(accountColl.find(Document("name" ->
 "Bob")).results().head.getInteger("balance") == 1500)

    accountColl.drop().await()

}
```

Python

下列程式碼示範如何使用 Amazon DocumentDB 交易 API 搭配 Python。

```
// Sample Python code with Core api

import pymongo

client = pymongo.MongoClient(<connection_string>)
rc_snapshot = pymongo.read_concern.ReadConcern('snapshot')
wc_majority = pymongo.write_concern.WriteConcern('majority')

# To start, drop and create an account collection and insert balances for both Alice
 and Bob
collection = client.get_database("bank").get_collection("account")
collection.drop()
collection.insert_one({"_id": 1, "name": "Alice", "balance": 1000})
collection.insert_one({"_id": 2, "name": "Bob", "balance": 1000})

amount_to_transfer = 500

# deduct 500 from Alice's account
alice_balance = collection.find_one({"name": "Alice"}).get("balance")
assert alice_balance >= amount_to_transfer
new_alice_balance = alice_balance - amount_to_transfer

with client.start_session({'causalConsistency':False}) as session:
    session.start_transaction(read_concern=rc_snapshot, write_concern=wc_majority)
    collection.update_one({"name": "Alice"}, {'$set': {"balance":
 new_alice_balance}}, session=session)
    session.commit_transaction()

updated_alice_balance = collection.find_one({"name": "Alice"}).get("balance")
assert updated_alice_balance == new_alice_balance

# add 500 to Bob's account
bob_balance = collection.find_one({"name": "Bob"}).get("balance")
assert bob_balance >= amount_to_transfer
new_bob_balance = bob_balance + amount_to_transfer

with client.start_session({'causalConsistency':False}) as session:
    session.start_transaction(read_concern=rc_snapshot, write_concern=wc_majority)
    collection.update_one({"name": "Bob"}, {'$set': {"balance": new_bob_balance}},
 session=session)
```

```
    session.commit_transaction()

updated_bob_balance = collection.find_one({"name": "Bob"}).get("balance")
assert updated_bob_balance == new_bob_balance
```

支援的命令

命令	支援
abortTransaction	是
commitTransaction	是
endSessions	是
killSession	是
killAllSession	是
killAllSessionsByPattern	否
refreshSessions	否
startSession	是

不支援的功能

方法	階段或命令
db.collection.aggregate()	$collStats $currentOp $indexStats $listSessions $out

方法	階段或命令
db.collection.count() db.collection.countDocuments()	$where $near $nearSphere
db.collection.insert()	insert 如果未針對現有集合執行，則不支援。如果此方法以預先存在的集合為目標，則支援此方法。

工作階段

MongoDB 工作階段是一種架構，用於支援可重試寫入、因果一致性、交易和管理跨資料庫的操作。建立工作階段時，用戶端會產生邏輯工作階段識別符 (lsid)，並在將命令傳送至伺服器時用於標記該工作階段中的所有操作。

Amazon DocumentDB 支援使用工作階段來啟用交易，但不支援因果一致性或可重試的寫入。

在 Amazon DocumentDB 內使用交易時，將使用 session.startTransaction() API 從工作階段內啟動交易，並且工作階段一次支援單一交易。同樣地，交易會使用遞交 (session.commitTransaction()) 或中止 (session.abortTransaction()) APIs 來完成。

因果一致性

因果一致性保證在單一用戶端工作階段中，用戶端會遵守read-after-write一致性、單原子讀取/寫入和寫入會遵循讀取，而且這些保證適用於叢集中的所有執行個體，而不只是主要執行個體。Amazon DocumentDB 不支援因果一致性，下列陳述式將導致錯誤。

```
var mySession = db.getMongo().startSession();
var mySessionObject = mySession.getDatabase('test').getCollection('account');

mySessionObject.updateOne({"_id": 2}, {"$inc": {"balance": 400}});
//Result:{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

mySessionObject.find()
//Error: error: {
//         "ok" : 0,
```

```
//         "code" : 303,
//         "errmsg" : "Feature not supported: 'causal consistency'",
//         "operationTime" : Timestamp(1603461817, 493214)
//}

mySession.endSession()
```

您可以在工作階段中停用因果一致性。請注意，這樣做可讓您利用工作階段架構，但不會對讀取提供因果一致性保證。使用 Amazon DocumentDB 時，來自主要 的讀取將是read-after-write一致，而來自複本執行個體的讀取將最終一致。交易是使用工作階段的主要使用案例。

```
var mySession = db.getMongo().startSession({causalConsistency: false});
var mySessionObject = mySession.getDatabase('test').getCollection('account');

mySessionObject.updateOne({"_id": 2}, {"$inc": {"balance": 400}});
//Result:{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

mySessionObject.find()
//{ "_id" : 1, "name" : "Bob", "balance" : 100 }
//{ "_id" : 2, "name" : "Alice", "balance" : 1700 }
```

可重試的寫入

可重試寫入是一種功能，其中用戶端會嘗試重試寫入操作一次、發生網路錯誤或用戶端無法找到主要寫入操作。在 Amazon DocumentDB 中，不支援可重試的寫入，且必須停用。您可以在連線字串中使用命令 (`retryWrites=false`) 將其停用。

> **Note**
>
> 如果您使用的是舊版 mongo shell （非 mongosh)，請勿在任何程式碼字串中包含 `retryWrites=false`命令。預設會停用可重試寫入。包含 `retryWrites=false` 可能會導致正常讀取命令失敗。

交易錯誤

使用交易時，可能會發生錯誤，指出交易號碼不符合任何進行中的交易。

錯誤可以在至少兩個不同的案例中產生：

- 一分鐘交易逾時之後。
- 執行個體重新啟動後（由於修補、當機復原等），即使交易已成功遞交，也可能收到此錯誤。在執行個體重新啟動期間，資料庫無法分辨成功完成的交易與中止的交易之間的差異。換句話說，交易完成狀態不明確。

處理此錯誤的最佳方式是使交易更新等冪 -- 例如，使用 mutator `$set` 而非遞增/遞減操作。請參閱以下內容：

```
{ "ok" : 0,
"operationTime" : Timestamp(1603938167, 1),
"code" : 251,
"errmsg" : "Given transaction number 1 does not match any in-progress transactions."
}
```

在 Amazon DocumentDB 8.0 中使用定序

Amazon DocumentDB 8.0 現在支援定序。定序可讓您設定特定語言的字串比較規則。使用定序，您可以指定區分大小寫比較的規則，或指定語言地區設定。您可以在 DocumentDB 8.0 的集合層級或索引層級設定定序。當您在 DocumentDB 中使用定序時，內部會使用下列參數建立定序文件。

```
{
locale: string,
caseLevel: boolean,
caseFirst: string,
strength: int,
numericOrdering: boolean,
alternate: string,
maxVariable: string,
backwards: boolean,
normalization: boolean
}
```

限制

定序在 Amazon DocumentDB 中具有下列限制：

- 定序與 Amazon DocumentDB 8.0 中可用的規劃器 v3 相容。切換到 plannerv2 或 plannerv1 可能會導致不一致的行為，包括「找不到索引」錯誤。
- 由於固有的程式庫差異，如果匯出具有定序的 mongodb 集合，您將需要更新 metadata.bson 檔案，並在遷移之前將其版本從 57.1 變更為 60.2。
- 在極少數情況下，您的定序設定可能會超出字元計數的內部限制，導致下列錯誤。「錯誤：定序文件具有超過支援的非預設屬性。請減少選項的數量。」 在此情況下，請嘗試減少定序文件中提供的選項，或者您也可以盡可能嘗試使用預設值。

在 Amazon DocumentDB 8.0 中使用檢視

- Amazon DocumentDB 8.0 現在支援檢視。檢視 函數做為虛擬集合，這些集合會根據指定的彙總操作呈現資料。當您建立檢視時，您可以定義查詢，從一或多個來源集合轉換資料。Amazon DocumentDB 8.0 會在每次存取檢視時執行此查詢，而不會消耗額外的儲存資源。與標準集合不同，Amazon DocumentDB 8.0 中的檢視不會將文件存放在磁碟上，因此是將轉換或篩選資料呈現給應用程式的有效解決方案。若要在 Amazon DocumentDB 中建立檢視，您可以使用 createView 命令或 db.createView() 協助程式方法：

```
db.createView("viewName","sourceCollection",
[
    { $match: { status: "active" } },
    { $project: { _id: 1, name: 1, email: 1 } }
]
)
```

  這會根據「sourceCollection」建立名為「viewName」的檢視，其中僅包含作用中的文件和專案 _id、名稱和電子郵件欄位。Amazon DocumentDB 中的檢視為唯讀。檢視上的寫入操作將傳回錯誤。若要使用大型資料集獲得最佳效能，您可以建構檢視管道以最大化效率。對於複雜的彙總管道，建議使用 $match 階段作為管道的第一個階段或早期階段，以減少後續階段需要處理的文件數量，從而提高查詢效能。

最佳實務

以下列出一些要遵循檢視的最佳實務。

- 早期篩選：在檢視管道的早期使用 $match 階段，以減少處理的資料量。

- 避免複雜彙總：對於具有複雜彙總的經常存取檢視，請考慮建立具有預先計算結果的個別集合，這些結果會定期更新。
- 索引規劃：確保在檢視管道中使用的欄位，特別是在 $match 和 $sort 操作中，在來源集合上正確編製索引。
- 查詢最佳化：使用 explain 命令來了解檢視查詢的執行方式，並據此進行最佳化。
- 檢視的替代方案：考慮到 Amazon DocumentDB 和 MongoDB 檢視之間的功能差異，在遇到限制時，請考慮使用定期集合搭配排程更新，作為檢視的替代方案。

彙總器運算子相容性

Amazon DocumentDB 在檢視定義中支援許多彙總運算子，同時繼續擴展相容性。使用檢視時，請專注於這些支援的運算子：

- $match 用於篩選文件
- 用於欄位選擇和轉換的 $project
- $addFields 用於新增運算欄位
- 用於排序結果的 $sort
- $limit 和 $skip for pagination

有些專業運算子，例如 $currentOp、$replaceRoot 和 $geoNear，目前適用於直接彙總查詢，而不是檢視定義。

利用索引和檢視

Amazon DocumentDB 8.0 中的檢視會使用基礎集合的索引。因此，您無法直接在檢視上建立、捨棄或重建索引。不過，在來源集合上設計良好的索引可以大幅改善檢視查詢效能 以下是最佳化檢視查詢效能的一些步驟：

- 確保檢視管道中使用的來源集合欄位上存在適當的索引，特別是在 $match 和 $sort 操作中
- 使用 explain() 方法來分析查詢執行計劃並驗證索引用量。例如，`db.viewName.find({...}).explain()`

Amazon DocumentDB 生成式人工智慧

Amazon DocumentDB 提供功能，讓機器學習 (ML) 和生成式人工智慧 (AI) 模型能夠即時處理存放在 Amazon DocumentDB 中的資料。客戶不再需要花時間管理個別基礎設施、編寫程式碼以與其他 服務連線，以及從其主要資料庫複製資料。

如需人工智慧以及如何 AWS 支援 AI 需求的詳細資訊，請參閱這篇「什麼」文章。

主題

- 使用 Amazon SageMaker AI Canvas 進行無程式碼機器學習
- Amazon DocumentDB 的向量搜尋

使用 Amazon SageMaker AI Canvas 進行無程式碼機器學習

Amazon SageMaker AI Canvas 可讓您建置自己的 AI/ML 模型，而無需撰寫單行程式碼。您可以針對迴歸和預測等常見使用案例建置 ML 模型，並可從 Amazon Bedrock 存取和評估基礎模型 (FMs)。您也可以從 Amazon SageMaker AI JumpStart 存取公有 FMs，以進行內容產生、文字擷取和文字摘要，以支援生成式 AI 解決方案。

如何使用 SageMaker AI Canvas 建置無程式碼 ML 模型

Amazon DocumentDB 現在與 Amazon SageMaker AI Canvas 整合，以使用存放在 Amazon DocumentDB 中的資料啟用無程式碼機器學習 (ML)。您現在可以建置 ML 模型以因應迴歸和預測需求，並使用基礎模型進行內容摘要，並使用存放在 Amazon DocumentDB 中的資料產生，而無需撰寫單行程式碼。

SageMaker AI Canvas 提供視覺化界面，可讓 Amazon DocumentDB 客戶產生預測，而不需要任何 AI/ML 專業知識或撰寫單行程式碼。客戶現在可以從 啟動 SageMaker AI Canvas 工作區 AWS 管理主控台，匯入和聯結 Amazon DocumentDB 資料以進行資料準備和模型訓練。Amazon DocumentDB 中的資料現在可用於 SageMaker AI Canvas，以建置和增強模型來預測客戶流失、偵測詐騙、預測維護失敗、預測業務指標和產生內容。客戶現在可以使用 SageMaker AI Canvas 與 Quick 的原生整合，在團隊之間發佈和分享 ML 驅動的洞見。SageMaker AI Canvas 中的資料擷取管道預設會在 Amazon DocumentDB 次要執行個體上執行，以確保應用程式和 SageMaker AI Canvas 擷取工作負載的效能不會受到阻礙。

Amazon DocumentDB 客戶可以導覽至新的 SageMakerAmazon DocumentDB 無代碼 ML 主控台頁面，並連線至新的或可用的 SageMaker AI Canvas 工作區，以開始使用 SageMaker AI Canvas。

設定 SageMaker AI 網域和使用者設定檔

您可以從以僅限 VPC 模式執行的 SageMaker AI 網域連線至 Amazon DocumentDB 叢集。透過在 VPC 中啟動 SageMaker AI 網域，您可以控制來自 SageMaker AI Studio 和 Canvas 環境的資料流程。這可讓您限制網際網路存取、使用標準 AWS 聯網和安全功能監控和檢查流量，並透過 VPC 端點連接到其他 AWS 資源。請參閱《Amazon SageMaker AI 開發人員指南》中的 Amazon SageMaker AI Canvas 入門Amazon SageMaker和在沒有網際網路存取的情況下在 VPC 中設定 Amazon SageMaker AI Canvas Amazon SageMaker，以建立 SageMaker AI 網域以連線至您的 Amazon DocumentDB 叢集。

設定 Amazon DocumentDB 和 SageMaker AI Canvas 的 IAM 存取許可

已`AmazonDocDBConsoleFullAccess`連接至其相關聯角色和身分的 Amazon DocumentDB 使用者可以存取 AWS 管理主控台。將下列動作新增至上述角色或身分，以提供使用 Amazon SageMaker AI Canvas 進行無程式碼機器學習的存取權。

```
"sagemaker:CreatePresignedDomainUrl",
"sagemaker:DescribeDomain",
"sagemaker:ListDomains",
"sagemaker:ListUserProfiles"
```

為 SageMaker AI Canvas 建立資料庫使用者和角色

您可以在 Amazon DocumentDB 中使用角色型存取控制 (RBAC)，限制對使用者可以在資料庫上執行之動作的存取。RBAC 的運作方式是將一或多個角色授予使用者。這些角色決定使用者可以對資料庫資源執行的操作。

身為 Canvas 使用者，您可以使用使用者名稱和密碼登入資料連線到 Amazon DocumentDB 資料庫。您可以使用 Amazon DocumentDB RBAC 功能，為具有特定資料庫讀取存取權的 Canvas 使用者建立資料庫使用者/角色。

例如，使用 `createUser`操作：

```
db.createUser({
user: "canvas_user",
pwd: "<insert-password>",
roles: [{role: "read", db: "sample-database-1"}]
})
```

這會建立canvas_user具有sample-database-1資料庫讀取許可的 。Canvas 分析師可以使用此登入資料來存取 Amazon DocumentDB 叢集中的資料。如需進一步了解使用角色型存取控制的資料庫存取，請參閱 。

可用的區域

無程式碼整合適用於同時支援 Amazon DocumentDB 和 Amazon SageMaker AI Canvas 的區域。區域包括：

- us-east-1(維吉尼亞北部)
- us-east-2(俄亥俄)
- us-west-2(奧勒岡)
- ap-northeast-1(東京)
- ap-northeast-2(首爾)
- ap-south-1(孟買)
- ap-southeast-1(新加坡)
- ap-southeast-2(雪梨)
- eu-central-1(法蘭克福)
- eu-west-1(愛爾蘭)

如需最新的區域可用性，請參閱《Amazon SageMaker AI 開發人員指南》中的 Amazon SageMaker AI Canvas。 Amazon SageMaker

Amazon DocumentDB 的向量搜尋

向量搜尋是機器學習中使用的一種方法，透過使用距離或相似性指標來比較向量表示法，以尋找與特定資料點類似的資料點。兩個向量越接近向量空間，則視為基礎項目越相似。此技術有助於擷取資料的語意意義。此方法適用於各種應用程式，例如建議系統、自然語言處理和影像辨識。

Amazon DocumentDB 的向量搜尋結合了 JSON 型文件資料庫的彈性和豐富的查詢功能，以及向量搜尋的強大功能。如果您想要使用現有的 Amazon DocumentDB 資料或靈活的文件資料結構來建置機器學習和生成式 AI 使用案例，例如語意搜尋體驗、產品建議、個人化、聊天機器人、詐騙偵測和異常偵測，則 Amazon DocumentDB 的向量搜尋是您的理想選擇。向量搜尋可在 Amazon DocumentDB 5.0 執行個體型叢集上使用。

主題

- 插入向量
- 建立向量索引
- 取得索引定義
- 查詢向量
- 功能和限制
- 最佳實務

插入向量

若要將向量插入 Amazon DocumentDB 資料庫，您可以使用現有的插入方法：

範例

在下列範例中，測試資料庫中會建立五個文件的集合。每份文件都包含兩個欄位：產品名稱及其對應的向量內嵌。

```
db.collection.insertMany([
  {"product_name": "Product A", "vectorEmbedding": [0.2, 0.5, 0.8]},
  {"product_name": "Product B", "vectorEmbedding": [0.7, 0.3, 0.9]},
  {"product_name": "Product C", "vectorEmbedding": [0.1, 0.2, 0.5]},
  {"product_name": "Product D", "vectorEmbedding": [0.9, 0.6, 0.4]},
  {"product_name": "Product E", "vectorEmbedding": [0.4, 0.7, 0.2]}
]);
```

建立向量索引

Amazon DocumentDB 支援階層式可導覽小型世界 (HNSW) 索引和使用平面壓縮 (IVFFlat) 索引方法的反轉檔案。IVFFlat 索引會將向量分隔為清單，然後搜尋最接近查詢向量的所選清單子集。另一方面，HNSW 索引會將向量資料整理成多層圖形。雖然 HNSW 的建置時間比 IVFFlat 慢，但可提供更好的查詢效能和召回。與 IVFFlat 不同，HNSW 不涉及任何訓練步驟，因此無需載入任何初始資料即可產生索引。對於大多數使用案例，請使用 HNSW 索引類型進行向量搜尋。

如果您未建立向量索引，Amazon DocumentDB 會執行最接近的鄰近搜尋，以確保完美召回。不過，在生產案例中，速度至關重要。使用向量索引，這可能會交易一些召回以提高速度。請務必注意，新增向量索引可能會導致不同的查詢結果。

範本

您可以使用下列 `createIndex`或 `runCommand` 範本，在向量欄位上建置向量索引：

Using createIndex

在某些驅動程式中，例如 mongosh 和 Java，使用 中的vectorOptions參數createIndex可能會導致錯誤。在這種情況下，請使用 runCommand：

```
db.collection.createIndex(
  { "<vectorField>": "vector" },
  { "name": "<indexName>",
    "vectorOptions": {
      "type": " <hnsw> | <ivfflat> ",
      "dimensions": <number_of_dimensions>,
      "similarity": " <euclidean> | <cosine> | <dotProduct> ",
      "lists": <number_of_lists> [applicable for IVFFlat],
      "m": <max number of connections> [applicable for HNSW],
      "efConstruction": <size of the dynamic list for index build> [applicable for
 HNSW]
    }
  }
);
```

Using runCommand

在某些驅動程式中，例如 mongosh 和 Java，使用 中的vectorOptions參數createIndex可能會導致錯誤。在這種情況下，請使用 runCommand：

```
db.runCommand(
  { "createIndexes": "<collection>",
  "indexes": [{
      key: { "<vectorField>": "vector" },
      vectorOptions: {
          type: " <hnsw> | <ivfflat> ",
          dimensions: <number of dimensions>,
          similarity: " <euclidean> | <cosine> | <dotProduct> ",
          lists: <number_of_lists> [applicable for IVFFlat],
          m: <max number of connections> [applicable for HNSW],
          efConstruction: <size of the dynamic list for index build> [applicable for
 HNSW]
          },
      name: "myIndex"
      }]
  }
);
```

參數	需求	資料類型	說明	值 (s)
name	選擇性	string	指定索引的名稱。	英數字元
type	選擇性		指定索引的類型。	支援：hnsw 或 ivfflat 預設：HNSW（引擎修補程式 3.0.4574 以上版本）
dimensions	必要	integer	指定向量資料中的維度數目。	最多 2，000 個維度。
similarity	必要	string	指定用於相似度計算的距離指標。	• **euclidean** • **cosine** • **dotProduct**
lists	IVFFlat 需要	integer	指定 IVFFlat 索引用來將向量資料分組的叢集數量。建議設定是最多 1M 文件和`sqrt(# of documents)` 超過 1M 文件的文件/1M000 數量。	下限：1 上限：請參閱功能和限制下列各執行個體類型資料表的清單。
m	選擇性	integer	指定 HNSW 索引的最大連線數	預設：16 範圍 【2，100】
efConstruction	選擇性	integer	指定用於建構 HNSW 索引圖形	預設：64 範圍 【4，1000】

參數	需求	資料類型	說明	值 (s)
			的動態候選清單大小。 efConstruction 必須大於或等於 (2 * m)	

請務必適當地設定子參數的值，例如 lists for IVFFlat m和 efConstruction for HNSW，因為它會影響搜尋的準確性/召回、建置時間和效能。較高的清單值會增加查詢的速度，因為它會減少每個清單中的向量數量，導致較小的區域。不過，較小的區域大小可能會導致更多召回錯誤，進而降低準確度。對於 HNSW，增加 的值m並efConstruction提高準確性，但也會增加索引建置時間和大小。請參閱以下範例：

範例

HNSW

```
db.collection.createIndex(
  { "vectorEmbedding": "vector" },
  { "name": "myIndex",
    "vectorOptions": {
      "type": "hnsw",
      "dimensions": 3,
      "similarity": "euclidean",
      "m": 16,
      "efConstruction": 64
    }
  }
);
```

IVFFlat

```
db.collection.createIndex(
  { "vectorEmbedding": "vector" },
  { "name": "myIndex",
    "vectorOptions": {
      "type": "ivfflat",
      "dimensions": 3,
      "similarity": "euclidean",
```

```
      "lists":1
    }
  }
)
```

取得索引定義

您可以使用 getIndexes命令來檢視索引的詳細資訊，包括向量索引：

範例

```
db.collection.getIndexes()
```

範例輸出

```
[
 {
  "v" : 4,
  "key" : {
   "_id" : 1
  },
  "name" : "_id_",
  "ns" : "test.collection"
 },
 {
  "v" : 4,
  "key" : {
   "vectorEmbedding" : "vector"
  },
  "name" : "myIndex",
  "vectorOptions" : {
   "type" : "ivfflat",
   "dimensions" : 3,
   "similarity" : "euclidean",
   "lists" : 1
  },
  "ns" : "test.collection"
 }
]
```

查詢向量

Amazon DocumentDB 支援兩個向量搜尋運算子來查詢向量：

傳統向量搜尋運算子

使用下列範本來查詢向量：

```
db.collection.aggregate([
  {
    $search: {
      "vectorSearch": {
        "vector": <query vector>,
        "path": "<vectorField>",
        "similarity": "<distance metric>",
        "k": <number of results>,
        "probes":<number of probes> [applicable for IVFFlat],
        "efSearch":<size of the dynamic list during search> [applicable for HNSW]
      }
    }
  }
]);
```

參數	需求	Type	說明	值 (s)
vectorSearch	必要	operator	在 $search 命令內用於查詢向量。	
vector	必要	陣列	指出將用於尋找類似向量的查詢向量。	
path	必要	string	定義向量欄位的名稱。	
k	必要	integer	指定搜尋傳回的結果數目。	

參數	需求	Type	說明	值 (s)
similarity	必要	string	指定用於相似性計算的距離指標。	• **euclidean** • **cosine** • **dotProduct**
probes	選擇性	integer	您希望向量搜尋檢查的叢集數量。較高的值以速度的成本提供更好的召回。它可以設定為最接近鄰搜尋的清單數量（此時規劃器不會使用索引）。開始微調的建議設定為 `sqrt(# of lists)`。	預設：1
efSearch	選擇性	integer	指定 HNSW 索引在搜尋期間使用的動態候選清單大小。較高的值`efSearch`可提高召回速度的成本。	預設：40 範圍 【1，1000】

請務必微調 `efSearch`(HNSW) 或 `probes`(IVFFlat) 的值，以達到所需的效能和準確性。請參閱下列範例操作：

HNSW

```
db.collection.aggregate([
  {
    $search: {
      "vectorSearch": {
```

```
        "vector": [0.2, 0.5, 0.8],
        "path": "vectorEmbedding",
        "similarity": "euclidean",
        "k": 2,
        "efSearch": 40
      }
    }
  }
]);
```

IVFFlat

```
db.collection.aggregate([
  {
    $search: {
      "vectorSearch": {
        "vector": [0.2, 0.5, 0.8],
        "path": "vectorEmbedding",
        "similarity": "euclidean",
        "k": 2,
        "probes": 1
      }
    }
  }
]);
```

範例輸出

此操作的輸出如下所示：

```
{ "_id" : ObjectId("653d835ff96bee02cad7323c"), "product_name" : "Product A",
 "vectorEmbedding" : [ 0.2, 0.5, 0.8 ] }
{ "_id" : ObjectId("653d835ff96bee02cad7323e"), "product_name" : "Product C",
 "vectorEmbedding" : [ 0.1, 0.2, 0.5 ] }
```

`$vectorSearch` 運算子（可在 Amazon DocumentDB 8.0 或更新版本中使用）

使用下列範本來查詢向量：

```
db.collection.aggregate([
{
  "$vectorSearch": {
```

```
    "exact": true | false,
    "index": "<index-name>" [supports only HNSW index],
    "limit": <number-of-results> [same as k],
    "path": "<vector field-to-search>",
    "queryVector": <array-of-numbers>,
    "numCandidates": <number-of-candidates> [same as efSearch],
  }
}])
```

功能和限制

版本相容性

- Amazon DocumentDB 的向量搜尋僅適用於 Amazon DocumentDB 5.0+ 執行個體型叢集。

向量

- Amazon DocumentDB 最多可為 2，000 個維度的向量編製索引。不過，如果沒有索引，最多可儲存 16，000 個維度。

索引

- 對於 IVFFlat 索引建立，清單參數的建議設定是最多 100 1M文件和`sqrt(# of documents)`超過 100 萬份文件的文件/1M000。由於運作中的記憶體限制，Amazon DocumentDB 支援清單參數的特定最大值，取決於維度數目。下表提供向量 500、1000 和 2，000 維度的清單參數最大值供您參考：

執行個體類型	具有 500 個維度的清單	具有 1000 個維度的清單	具有 2000 個維度的清單
t3.med	372	257	150
r5.l	915	741	511
r5.xl	1，393	1，196	901
r5.2xl	5，460	5，230	4，788
r5.4xl	7，842	7，599	7，138

執行個體類型	具有 500 個維度的清單	具有 1000 個維度的清單	具有 2000 個維度的清單
r5.8xl	11，220	10，974	10，498
r5.12xl	13，774	13，526	13，044
r5.16xl	15，943	15，694	15，208
r5.24xl	19，585	19，335	18，845

- 向量索引`partial`不支援其他索引選項，例如 `compound`或 `sparse`。
- Amazon DocumentDB 5.0 中的 HNSW 索引不支援平行索引建置。

向量查詢

- 對於向量搜尋查詢，請務必微調參數，例如 `probes`或 `efSearch` 以獲得最佳結果。`probes` 或 `efSearch` 參數的值越高，召回程度越高，速度也越低。開始微調探查參數的建議設定是 `sqrt(# of lists)`。

最佳實務

了解在 Amazon DocumentDB 中使用向量搜尋的最佳實務。在新最佳實務確定時，會不斷更新此小節。

- 使用平面壓縮 (IVFFlat) 索引建立的反轉檔案涉及根據相似性叢集和組織資料點。若要讓索引更有效率，請在建立索引之前載入一些資料。
- 對於向量搜尋查詢，請務必微調參數，例如 `probes`或 `efSearch` 以獲得最佳結果。`probes` 或 `efSearch` 參數的值越高，召回率越高，速度越低。開始微調`probes`參數的建議設定為 `sqrt(lists)`。

資源

- Amazon [Amazon DocumentDB 的向量搜尋現在已全面推出](#) AWS 部落格
- [語意搜尋程式碼範例](#)
- [Amazon DocumentDB 向量搜尋程式碼範例](#)

使用 AWS Lambda 搭配變更串流

Amazon DocumentDB 已與 整合 AWS Lambda ，因此您可以使用 Lambda 函數來處理變更串流中的記錄。Lambda 事件來源映射是一種資源，可用來叫用 Lambda 函數來處理未直接叫用 Lambda 的 Amazon DocumentDB 事件。使用 Amazon DocumentDB 變更串流做為事件來源，您可以建置事件驅動型應用程式來回應資料中的變更。例如，您可以使用 Lambda 函數來處理新文件、追蹤現有文件的更新，或記錄已刪除的文件。

您可以設定事件來源映射，將記錄從 Amazon DocumentDB 變更串流傳送至 Lambda 函數。事件可以一次傳送一個事件，或批次處理以提高效率，並按順序處理。您可以根據特定時段持續時間 (0 - 300 秒）或批次記錄計數 （最多 10，000 筆記錄） 來設定事件來源映射的批次處理行為。您可以建立多個事件來源映射，以使用多個 Lambda 函數處理相同的資料，或使用單一函數處理來自多個串流的不同項目。

如果您的函數傳回錯誤，Lambda 會重試批次，直到成功處理為止。如果變更串流中的事件已過期，Lambda 會停用事件來源映射。在這種情況下，您可以建立新的事件來源映射，並使用您選擇的開始位置進行設定。由於輪詢器的分散式特性，Lambda 事件來源映射會至少處理一次事件。因此在極少數情況下，您的 Lambda 函數可能會收到重複的事件。遵循使用 AWS Lambda 函數和建置等冪函數的最佳實務，以避免與重複事件相關的問題。如需詳細資訊，請參閱《 AWS Lambda 開發人員指南》中的搭配使用 AWS Lambda console 與 Amazon DocumentDB。

作為效能最佳實務，Lambda 函式必須為短期函數。為了避免產生不必要的處理延遲，此函數也不應執行複雜的邏輯。尤其對於高速串流而言，相較於同步長時間執行的 Lambda，觸發非同步後續處理 Step Function 工作流程是較佳的做法。如需 的詳細資訊 AWS Lambda，請參閱 AWS Lambda 開發人員指南。

限制

以下是使用 Amazon DocumentDB 和 時應考慮的限制 AWS Lambda：

- AWS Lambda 目前僅支援 Amazon DocumentDB 4.0 和 5.0。
- AWS Lambda 目前不支援彈性叢集或全域叢集。
- AWS Lambda 承載大小不能超過 6MB。如需 Lambda 批次大小的詳細資訊，請參閱《 AWS Lambda 開發人員指南》中的 Lambda 事件來源映射一節中的「批次行為」。

Amazon DocumentDB 與 MongoDB 的相容性

Amazon DocumentDB 支援 MongoDB 相容性，包括 MongoDB 4.0、MongoDB 5.0 和 MongoDB 8.0。MongoDB 相容性表示您目前已與 MongoDB 資料庫搭配使用的絕大多數應用程式、驅動程式和工具，都可以與 Amazon DocumentDB 搭配使用，且幾乎沒有變更。本節說明有關 Amazon DocumentDB 與 MongoDB 相容性的所有須知，包括新功能和功能、入門、遷移路徑和功能差異。

主題

- MongoDB 8.0 相容性
- MongoDB 5.0 相容性
- MongoDB 4.0 相容性
- 功能差異：Amazon DocumentDB 和 MongoDB
- Amazon DocumentDB 中支援的 MongoDB APIs、操作和資料類型

MongoDB 8.0 相容性

主題

- Amazon DocumentDB 8.0 中的新功能
- 開始使用 Amazon DocumentDB 8.0
- 升級或遷移至 Amazon DocumentDB 5.0 或 8.0
- 功能差異

Amazon DocumentDB 8.0 中的新功能

Amazon DocumentDB 8.0 可將查詢效能提升高達 7 倍，並將壓縮率提升高達 5 倍，讓您以較低的成本建置高效能應用程式。Amazon DocumentDB 8.0 具有與 MongoDB 8.0 的完整線路通訊協定相容性。以下摘要介紹了 Amazon DocumentDB 8.0 中介紹的一些主要功能。若要查看新功能的完整清單，請參閱 版本備註。

- 透過新增對 MongoDB 8.0 API 驅動程式的支援，提供與 MongoDB 8.0 的相容性。也支援使用 MongoDB API 6.0 和 7.0 版建置的應用程式。
- Amazon DocumentDB 8.0 中的新查詢規劃器將效能改進擴展到彙總階段運算子，以及支援彙總管道最佳化和不同的命令。

- Amazon DocumentDB 8.0 支援定序和檢視。
- 提供 6 個新的彙總階段：$replaceWith、$vectorSearch、$merge、$set、$unset、$bucket 和 3 個新的彙總運算子 $pow、$rand、$dateTrunc。
- 新版本的文字索引：Amazon DocumentDB 8.0 中的文字索引 v2 引入了其他字符，增強了文字搜尋功能。
- 透過平行向量索引建置，Amazon DocumentDB 8.0 可將索引建置時間縮短高達 30 倍。
- 將儲存限制提高到 256 TiB。

開始使用 Amazon DocumentDB 8.0

若要開始使用 Amazon DocumentDB 8.0，請參閱 開始使用 Amazon DocumentDB。您可以使用 AWS 管理主控台 或 SDK AWS CLI或 建立新的 Amazon DocumentDB 8 AWS .0 叢集 CloudFormation。連線至 Amazon DocumentDB 時，您必須使用與 MongoDB 5.0 或更高版本相容的 MongoDB 驅動程式或公用程式。

> **Note**
>
> 使用 AWS SDK AWS CLI或 時 CloudFormation，引擎版本預設為 5.0.0。您必須明確指定 參數，`engineVersion = 8.0.0`以建立新的 Amazon DocumentDB 8.0 叢集`engineVersion = 4.0.0`，或建立新的 Amazon DocumentDB 4.0 叢集`engineVersion = 3.6.0`，或建立新的 Amazon DocumentDB 3.6 叢集。對於指定的 Amazon DocumentDB 叢集，您可以使用 來判斷叢集版本 AWS CLI ，以呼叫`describe-db-clusters`或使用 Amazon DocumentDB 管理主控台來檢視特定叢集的引擎版本編號。

Amazon DocumentDB 5.0 和 8.0 支援叢集的 Amazon EC2 `r8g` Graviton4 處理器和 Graviton2 處理器，例如 `r6g`和 `t4.medium`執行個體類型，且在所有支援的 AWS 區域 （ 請參閱 執行個體) 中提供。如需定價的詳細資訊，請參閱 Amazon DocumentDB （ 與 MongoDB 相容 ） 定價。

升級或遷移至 Amazon DocumentDB 5.0 或 8.0

您可以使用 或 公用程式，例如 mongodump、mongorestore、mongoimport 和 mongoexport，從 MongoDB 3.6 或 MongoDB 4.0 遷移至 Amazon DocumentDB 5.0 AWS DMS或 Amazon DocumentDB 8.0。 ???如需遷移的指示，請參閱 使用 升級 Amazon DocumentDB 叢集 AWS Database Migration Service。

功能差異

Amazon DocumentDB 5.0 和 8.0 之間的功能差異

隨著 Amazon DocumentDB 8.0 的發行，Amazon DocumentDB 5.0 和 Amazon DocumentDB 8.0 之間存在功能差異：

- 規劃器 v1 是 Amazon DocumentDB 5.0 中的預設查詢規劃器，而效能更高的規劃器 v3 是 Amazon DocumentDB 8.0 中的預設值。
- Amazon DocumentDB 8.0 中的新功能，包括檢視、定序和類似 $merge 的運算子，僅與規劃器 v3 相容。
- 在 Amazon DocumentDB 8.0 中，壓縮預設為開啟，並設定為使用 Zstandard 演算法。此外，在 Amazon DocumentDB 8.0 中，「啟用」不再是有效的選擇；您可以從 Zstd、LZ4 和無中選擇。
- Amazon DocumentDB 8.0 僅支援資料庫連線的 TLS 1.2 和更新版本。

> **Important**
>
> Amazon DocumentDB 8.0 需要 TLS 1.2 或更高版本。不再支援 TLS 1.0 和 TLS 1.1。

MongoDB 5.0 相容性

主題

- Amazon DocumentDB 5.0 中的新功能
- Amazon DocumentDB 5.0.1 中的新功能
- 開始使用 Amazon DocumentDB 5.0
- 升級或遷移至 Amazon DocumentDB 5.0
- 功能差異

Amazon DocumentDB 5.0 中的新功能

Amazon DocumentDB 5.0 推出新功能，包括儲存限制和用戶端欄位層級加密。以下摘要介紹了 Amazon DocumentDB 5.0 中介紹的一些主要功能。若要查看新功能的完整清單，請參閱 版本備註。

- 針對所有執行個體型 Amazon DocumentDB 叢集和碎片型彈性叢集，將儲存限制提高至 128 TiB。

- 推出 Amazon DocumentDB 5.0 引擎版本 3.0.775)
  - 支援 MongoDB 5.0 API 驅動程式
  - 支援用戶端欄位層級加密 (FLE)。您現在可以先加密用戶端的欄位，再將資料寫入 Amazon DocumentDB 叢集。如需詳細資訊，請參閱用戶端欄位層級加密。
  - 新的彙總運算子：`$dateAdd`、 `$dateSubtract`
  - 支援 運算`$elemMatch`子的索引。因此，具有 的查詢`$elemMatch`將導致索引掃描。

Amazon DocumentDB 不支援每個 MongoDB 5.0 功能。當我們建置 Amazon DocumentDB 5.0 時，我們會從客戶要求我們建置最多的特徵和功能向後工作。我們會根據客戶要求我們建置的內容，繼續新增額外的 MongoDB 5.0 功能。如需支援 APIs 的最新清單，請參閱 Amazon DocumentDB 中支援的 MongoDB APIs、操作和資料類型。

Amazon DocumentDB 5.0.1 中的新功能

Amazon DocumentDB 5.0.1 是 Amazon DocumentDB 5.0 主要版本的第一個次要版本版本。次要版本包括改善功能，例如新功能、安全性修正和錯誤修正。您可以使用次要版本升級，從 5.0.0 升級到 5.0.1。如需詳細資訊，請參閱Amazon DocumentDB 次要版本升級。

- 使用新運算子增強彙總功能 (`$rand`、`$pow`、`$dateToParts`、)`$dateFromParts`。
- 用於監控執行個體的作用中連線指標。如需詳細資訊，請參閱執行個體配額。
- CloudWatch 中用於 CRUD 操作 (`find`、`insert`、`findAndModify`、、`update`、`deleteaggregatecount`、`distinct`、`getMore`、a 公開 `AvgDuration`、 `P100Duration`和 `MaxConcurrent`維度的精細命令層級效能指標，以改善操作監控。
- Amazon DocumentDB 5.0.1 僅支援資料庫連線的 TLS 1.2 和更新版本。
- 在 `storageSegmentBase`和 `storageSegmentExtended` 中新增 `gcRuntimeStats`、`documentFragmentStats`和 集合層級指標的分解`collstats`，以將更多資訊呈現到垃圾收集程序的整體運作狀態。

開始使用 Amazon DocumentDB 5.0

若要開始使用 Amazon DocumentDB 5.0，請參閱 開始使用 Amazon DocumentDB。您可以使用 AWS 管理主控台 或 SDK AWS CLI或 建立新的 Amazon DocumentDB 5 AWS .0 叢集 CloudFormation。連線至 Amazon DocumentDB 時，您必須使用與 MongoDB 5.0 或更高版本相容的 MongoDB 驅動程式或公用程式。

> **Note**
>
> 使用 AWS SDK AWS CLI或 時 CloudFormation，引擎版本預設為 5.0.0。您必須明確指定 參數，`engineVersion = 4.0.0`才能建立新的 Amazon DocumentDB 4.0 叢集`engineVersion = 3.6.0`或建立新的 Amazon DocumentDB 3.6 叢集。對於指定的 Amazon DocumentDB 叢集，您可以使用 來決定叢集版本 AWS CLI ，以呼叫`describe-db-clusters`或使用 Amazon DocumentDB 管理主控台來檢視特定叢集的引擎版本編號。

Amazon DocumentDB 5.0 和 8.0 支援叢集的 Amazon EC2 `r8g` Graviton4 處理器和 Graviton2 處理器，例如 `r6g`和 `t4.medium`執行個體類型，且在所有支援的 AWS 區域 （ 請參閱 執行個體) 中提供。如需定價的詳細資訊，請參閱 Amazon DocumentDB （ 與 MongoDB 相容 ） 定價。

升級或遷移至 Amazon DocumentDB 5.0

您可以使用 或 公用程式，例如 mongodump、mongorestore、mongoimport 和 mongoexport，從 MongoDB 3.6 AWS DMS或 MongoDB 4.0 遷移至 Amazon DocumentDB 5.0。 ???如需遷移的指示，請參閱 使用 升級 Amazon DocumentDB 叢集 AWS Database Migration Service。

功能差異

Amazon DocumentDB 4.0 和 5.0 之間的功能差異

隨著 Amazon DocumentDB 5.0 的發行，Amazon DocumentDB 4.0 和 Amazon DocumentDB 5.0 之間存在功能差異：

- 備份內建角色現在支援 `serverStatus`。動作 - 具有備份角色的開發人員和應用程式可以收集有關 Amazon DocumentDB 叢集狀態的統計資料。
- `SecondaryDelaySecs` 欄位會在`replSetGetConfig`輸出`slaveDelay`中取代 。
- hello 命令會取代 `isMaster` - hello傳回描述 Amazon DocumentDB 叢集角色的文件。
- Amazon DocumentDB 5.0 現在支援在第一個巢狀層級使用`$elemMatch`運算子進行索引掃描。當僅查詢篩選條件具有一個層級的`$elemMatch`篩選條件時，支援索引掃描，但如果包含巢狀`$elemMatch`查詢，則不支援。

  例如，在 Amazon DocumentDB 5.0 中，如果您在巢狀層級中包含`$elemMatch`運算子，則不會像在 Amazon DocumentDB 4.0 中一樣傳回值：

```
db.foo.insert(
```

```
[
    {a: {b: 5}},
    {a: {b: [5]}},
    {a: {b: [3, 7]}},
    {a: [{b: 5}]},
    {a: [{b: 3}, {b: 7}]},
    {a: [{b: [5]}]},
    {a: [{b: [3, 7]}]},
    {a: [[{b: 5}]]},
    {a: [[{b: 3}, {b: 7}]]},
    {a: [[{b: [5]}]]},
    {a: [[{b: [3, 7]}]]}
]);

// DocumentDB 5.0
> db.foo.find({a: {$elemMatch: {b: {$elemMatch: {$lt: 6, $gt: 4}}}}}, {_id: 0})
{ "a" : [ { "b" : [ 5 ] } ] }

// DocumentDB 4.0
> db.foo.find({a: {$elemMatch: {b: {$elemMatch: {$lt: 6, $gt: 4}}}}}, {_id: 0})
{ "a" : [ { "b" : [ 5 ] } ] }
{ "a" : [ [ { "b" : [ 5 ] } ] ] }
```

- Amazon DocumentDB 4.0 中的「$」投影會傳回所有欄位的所有文件。使用 Amazon DocumentDB 5.0，具有「$」投影的find命令會傳回符合查詢參數的文件，僅包含符合「$」投影的欄位。
- 在 Amazon DocumentDB 5.0 中，具有 $regex和 $options查詢參數的find命令會傳回錯誤：「無法同時在 $regex和 中設定選項$options。
- 使用 Amazon DocumentDB 5.0， $indexOfCP 現在會在下列情況下傳回 "-1"：
  - 在字串表達式中找不到子字串，或
  - start 是大於 end 的數字，或
  - start 是大於字串位元組長度的數字。
- 在 Amazon DocumentDB 4.0 中，當起始位置的數字大於字串的結束或位元組長度時，會$indexOfCP傳回 "0"。
- 使用 Amazon DocumentDB 5.0，例如 中的投影操作_id fields{"_id.nestedField" : 1}，傳回僅包含投影欄位的文件。而在 Amazon DocumentDB 4.0 中，巢狀欄位投影命令不會篩選掉任何文件。

MongoDB 4.0 相容性

主題

- Amazon DocumentDB 4.0 功能
- 開始使用 Amazon DocumentDB 4.0
- 升級或遷移至 Amazon DocumentDB 4.0
- 功能差異

Amazon DocumentDB 4.0 功能

Amazon DocumentDB 4.0 推出許多新功能，包括 ACID 交易和變更串流的改進。以下摘要顯示 Amazon DocumentDB 4.0 中引進的一些主要功能。若要查看功能的完整清單，請參閱 版本備註。

- ACID 交易：Amazon DocumentDB 現在支援跨多個文件、陳述式、集合和資料庫執行交易的能力。交易可讓您跨 Amazon DocumentDB 叢集中的一或多個文件執行原子、一致、隔離和耐用 (ACID) 操作，以簡化應用程式開發。如需詳細資訊，請參閱Amazon DocumentDB 中的交易。
- 變更串流：您現在可以在叢集層級 (`client.watch()` 或 `mongo.watch()`) 和資料庫 (`db.watch()`) 開啟變更串流，您可以指定 `startAtOperationTime`來開啟變更串流游標，最後，您現在可以將變更串流保留期間延長為 7 天 （先前為 24 小時）。如需詳細資訊，請參閱搭配 Amazon DocumentDB 使用變更串流。
- AWS Database Migration Service (AWS DMS)：您現在可以使用 AWS DMS 將 MongoDB 4.0 工作負載遷移至 Amazon DocumentDB。 AWS DMS 現在支援 MongoDB 4.0 來源、Amazon DocumentDB 4.0 目標和 Amazon DocumentDB 3.6 來源，以在 Amazon DocumentDB 3.6 和 4.0 之間執行升級。如需詳細資訊，請參閱 AWS DMS 文件。
- 效能和索引：您現在可以使用具有 的索引`$lookup`、尋找具有包含一個欄位或一個欄位的投影的查詢，並且可以直接從索引提供 `_id` 欄位，而無需從集合讀取 （涵蓋的查詢）、`hint()`具有 的功能`findAndModify`、 的效能最佳化`$addToSet`，以及減少整體索引大小的改進。如需詳細資訊，請參閱版本備註。
- 運算子：Amazon DocumentDB 4.0 現在支援許多新的彙總運算子：`$ifNull`、`$replaceRoot`、`$setIsSubset`、`$setIntersection``$setUnion`、、`$setEquals`。您可以在 中查看我們支援的所有 MongoDB APIs、操作和資料類型Amazon DocumentDB 中支援的 MongoDB APIs、操作和資料類型。
- 角色型存取控制 (RBAC)：透過 `ListCollection`和 `ListDatabase`命令，您現在可以選擇性地使用 `authorizedCollections`和 `authorizedDatabases` 參數，允許使用者列出他們有權存取的

集合和資料庫，而不需要 `listCollections`和 `listDatabase`角色。您也可以刪除自己的游標，而不需要該`KillCursor`角色。

Amazon DocumentDB 不支援每個 MongoDB 4.0 功能。當我們建置 Amazon DocumentDB 4.0 時，我們從客戶要求我們建置最多的特徵和功能向後工作。我們會根據客戶要求我們建置的內容，繼續新增額外的 MongoDB 4.0 功能。例如，Amazon DocumentDB 4.0 目前不支援 MongoDB 4.0 中引入的類型轉換運算子或字串運算子。如需支援 APIs 的最新清單，請參閱 Amazon DocumentDB 中支援的 MongoDB APIs、操作和資料類型。

開始使用 Amazon DocumentDB 4.0

若要開始使用 Amazon DocumentDB 4.0，請參閱 開始使用 Amazon DocumentDB。您可以使用 AWS 管理主控台 或 SDK AWS CLI或 建立新的 Amazon DocumentDB 4 AWS .0 叢集 CloudFormation。連線至 Amazon DocumentDB 時，您必須使用與 MongoDB 4.0 或更高版本相容的 MongoDB 驅動程式或公用程式。

> **Note**
>
> 使用 AWS SDK AWS CLI或 時 CloudFormation，引擎版本預設為 5.0.0。您必須明確指定 參數，`engineVersion = 4.0.0`才能建立新的 Amazon DocumentDB 4.0 叢集`engineVersion = 3.6.0`或建立新的 Amazon DocumentDB 3.6 叢集。對於指定的 Amazon DocumentDB 叢集，您可以使用 來判斷叢集版本 AWS CLI ，以呼叫`describe-db-clusters`或使用 Amazon DocumentDB 管理主控台來檢視特定叢集的引擎版本編號。

Amazon DocumentDB 4.0 支援叢集的 `r5`、`t3.medium`、 `r6g`和 `t4g.medium`執行個體類型，並可在所有支援的 中使用。 AWS 區域使用 Amazon DocumentDB 4.0 無需額外費用。如需定價的詳細資訊，請參閱 Amazon DocumentDB （與 MongoDB 相容） 定價。

升級或遷移至 Amazon DocumentDB 4.0

您可以使用 或 公用程式，例如 mongodump、mongorestore、mongoimport 和 mongoexport，從 MongoDB 3.6 AWS DMS或 MongoDB 4.0 遷移至 Amazon DocumentDB 4.0。 ???同樣地，您可以使用相同的工具從 Amazon DocumentDB 3.6 升級到 Amazon DocumentDB 4.0。如需遷移的指示，請參閱 使用 升級 Amazon DocumentDB 叢集 AWS Database Migration Service。

功能差異

Amazon DocumentDB 3.6 和 4.0 之間的功能差異

隨著 Amazon DocumentDB 4.0 的發行，Amazon DocumentDB 3.6 和 Amazon DocumentDB 4.0 之間存在功能差異：

- 巢狀文件的投影：套用投影時，Amazon DocumentDB 3.6 會考慮巢狀文件中的第一個欄位。不過，Amazon DocumentDB 4.0 會剖析子文件，並將投影套用至每個子文件。例如：如果投影是 `"a.b.c": 1`，則兩個版本的行為都相同。不過，如果投影是 Amazon DocumentDB 3.6`{a:{b:{c:1}}}`，則只會將投影套用至 'a'，而不是 'b' 或 'c'。
- 、 **`minKey`**的行為**`maxKey`**：在 Amazon DocumentDB 4.0 中， 的行為不會`{x:{$gt:MaxKey}}`傳回任何內容，而 則不會`{x:{$lt:MaxKey}}`傳回所有內容。
- 文件比較差異：比較子文件中不同類型 （雙、整數、長） 的數值 （例如，在 b中`{"_id" :1, "a" :{"b":1}}`) 現在提供跨數值資料類型和文件每個層級的一致輸出。

Amazon DocumentDB 4.0 和 MongoDB 4.0 之間的功能差異

以下是 Amazon DocumentDB 4.0 和 MongoDB 4.0 之間的功能差異。

- 路徑中具有空索引鍵的查詢：當集合包含陣列中具有空索引鍵的文件 （例如 `{"x" : [ { "" : 10 }, { "b" : 20 } ]}`)，以及當查詢中使用的索引鍵結尾為空字串 （例如 `x.`) 時，Amazon DocumentDB 會傳回該文件，因為它會周遊陣列中的所有文件，而 MongoDB 不會傳回該文件。
- **`$setOnInsert`** 路徑**`$`**中的 與 ： 欄位運算子`$setOnInsert`不會在 Amazon DocumentDB 中的 路徑`$`中與 搭配使用，這也與 MongoDB 4.0 一致。

功能差異：Amazon DocumentDB 和 MongoDB

以下是 Amazon DocumentDB （與 MongoDB 相容） 和 MongoDB 之間的功能差異。

主題

- Amazon DocumentDB 的功能優勢
- 已更新功能差異
- MongoDB 的功能差異

Amazon DocumentDB 的功能優勢

隱含交易

在 Amazon DocumentDB 中，所有 CRUD update陳述式 (findAndModify、insert、、delete) 保證原子性和一致性，即使是修改多個文件的操作也一樣。隨著 Amazon DocumentDB 4.0 的推出，現在支援為多陳述式和多集合操作提供 ACID 屬性的明確交易。如需在 Amazon DocumentDB 中使用交易的詳細資訊，請參閱 Amazon DocumentDB 中的交易。

以下是 Amazon DocumentDB 中的操作範例，可修改滿足原子和一致行為的多個文件。

```
db.miles.update(
    { "credit_card": { $eq: true } },
    { $mul: { "flight_miles.$[]": NumberInt(2) } },
    { multi: true }
)
```

```
db.miles.updateMany(
    { "credit_card": { $eq: true } },
    { $mul: { "flight_miles.$[]": NumberInt(2) } }
)
```

```
db.runCommand({
  update: "miles",
  updates: [
    {
      q: { "credit_card": { $eq: true } },
      u: { $mul: { "flight_miles.$[]": NumberInt(2) } },
      multi: true
    }
  ]
})
```

```
db.products.deleteMany({
  "cost": { $gt: 30.00 }
})
```

```
db.runCommand({
  delete: "products",
  deletes: [{ q: { "cost": { $gt: 30.00 } }, limit: 0 }]
})
```

構成 updateMany 和 deleteMany 之類批量操作的個別操作是不可部分完成，但整個批量操作是可部分完成。例如，如果個別插入操作成功執行而沒有錯誤，則整個 insertMany 操作都是不可部分完成。如果 insertMany 操作遇到錯誤，則 insertMany 操作內的每個單獨插入陳述式都將當作不可部分完成操作來執行。如果您需要 insertMany、 updateMany和 deleteMany操作的 ACID 屬性，建議您使用交易。

已更新功能差異

Amazon DocumentDB 透過從客戶要求我們建置的功能向後工作，持續改善與 MongoDB 的相容性。本節包含我們在 Amazon DocumentDB 中移除的功能差異，讓客戶更輕鬆地遷移和建置應用程式。

主題

- 陣列索引
- 多金鑰索引
- 字串中的 Null 字元
- 角色類型存取控制
- $regex 索引
- 巢狀文件的投影

陣列索引

截至 2020 年 4 月 23 日，Amazon DocumentDB 現在支援為大於 2，048 個位元組的陣列編製索引。陣列中個別項目的限制仍為 2，048 個位元組，與 MongoDB 一致。

如果您要建立新索引，則不需要採取任何動作來利用改進的功能。如果您有現有的索引，您可以刪除索引再重新建立，以利用改進的功能。具有改進功能的最新索引版本為 "v" ： 3。

> **Note**
>
> 對於生產叢集，捨棄索引可能會影響您的應用程式效能。對生產系統進行變更時，請先測試並謹慎進行。此外，重新建立索引所需的時間取決於集合的整體資料大小。

您可以使用以下命令查詢索引的版本。

```
db.collection.getIndexes()
```

此操作的輸出將會如下所示。在此輸出中，索引的版本是 "v" ： 3，這是最新的索引版本。

```
[
    {
        "v" : 3,
        "key" : {
        "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.test"
    }
]
```

多金鑰索引

自 2020 年 4 月 23 日起，Amazon DocumentDB 現在支援在相同陣列中建立具有多個索引鍵的複合索引。

如果您要建立新索引，則不需要採取任何動作來利用改進的功能。如果您有現有的索引，您可以刪除索引再重新建立，以利用改進的功能。具有改進功能的最新索引版本為 "v" ： 3。

> **Note**
>
> 對於生產叢集，捨棄索引可能會影響您的應用程式效能。對生產系統進行變更時，請先測試並謹慎進行。此外，重新建立索引所需的時間取決於集合的整體資料大小。

您可以使用以下命令查詢索引的版本。

```
db.collection.getIndexes()
```

此操作的輸出將會如下所示。在此輸出中，索引的版本是 "v" ： 3，這是最新的索引版本。

```
[
    {
```

```
        "v" : 3,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.test"
    }
]
```

字串中的 Null 字元

自 2020 年 6 月 22 日起，Amazon DocumentDB 現在支援字串中的 null 字元 '\0' ()。

角色類型存取控制

截至 2020 年 3 月 26 日，Amazon DocumentDB 支援內建角色的角色型存取控制 (RBAC)。如需詳細資訊，請參閱 以角色為基礎的存取控制。

$regex 索引

自 2020 年 6 月 22 日起，Amazon DocumentDB 現在支援$regex運算子利用索引的能力。

若要使用與 $regex 運算子的索引，您必須使用 hint() 命令。使用 hint() 時，您必須指定要套用 $regex 之欄位的名稱。例如，如果您在欄位 product 有名為 p_1 的索引，db.foo.find({product: /^x.*/}).hint({product:1}) 將使用 p_1 索引，但 db.foo.find({product: /^x.*/}).hint("p_1") 不會使用該索引。您可以使用 explain() 命令或使用分析工具的記錄慢速查詢功能來驗證是否已選擇索引。例如 db.foo.find({product: /^x.*/}).hint("p_1").explain()。

> **Note**
>
> hint() 方法一次只能與一個索引一起使用。

$regex 查詢的索引使用已針對使用字首且未指定 i, 、m 或 o 規則運算式選項的查詢進行最佳化。

將 $regex 搭配索引使用時，建議您在有高度選擇性的欄位上建立索引，其中重複值的數目會小於集合中文件總數的 1%。例如，如果您的集合包含 100,000 份文件，則只在相同值出現 1000 次或更少的欄位上建立索引。

巢狀文件的投影

Amazon DocumentDB `$project` 和 MongoDB 在 3.6 版中有功能差異，已在 Amazon DocumentDB 4.0 中解決，但在 Amazon DocumentDB 3.6 中仍不受支援。

Amazon DocumentDB 3.6 只會在套用投影時考慮巢狀文件中的第一個欄位，而 MongoDB 3.6 也會剖析子文件，並將投影套用至每個子文件。

例如：如果投影為 `“a.b.c”: 1`，則行為在 Amazon DocumentDB 和 MongoDB 中都如預期般運作。不過，如果投影為 `{a:{b:{c:1}}}`，則 Amazon DocumentDB 3.6 只會將投影套用至 `a` ，而不是 `b`或 `c`。在 Amazon DocumentDB 4.0 中，投影`{a:{b:{c:1}}}`將套用至 `a`、 `b`和 `c`。

MongoDB 的功能差異

主題

- $vectorSearch運算子
- OpCountersCommand
- 管理員資料庫和集合
- cursormaxTimeMS
- explain()
- 索引組建
- 路徑中具有空索引鍵的查詢
- MongoDB APIs、操作和資料類型
- mongodump 和 mongorestore 公用程式
- 結果排序
- 可重試的寫入
- 稀鬆索引
- 在$all表達式$elemMatch內使用
- 欄位名稱中的 Dollar($) 和 dot(.)
- $lookup
- $natural 和反向排序

$vectorSearch運算子

Amazon DocumentDB 不支援 $vectorSearch 做為獨立運算子。相反地，我們在$search運算子vectorSearch內部支援 。如需詳細資訊，請參閱Amazon DocumentDB 的向量搜尋。

OpCountersCommand

Amazon DocumentDB OpCountersCommand的行為偏離 MongoDB 的行為opcounters.command ，如下所示：

- MongoDB opcounters.command計算所有命令，但插入、更新和刪除除外，而 Amazon DocumentDB 的 OpCountersCommand 也排除命令find。
- Amazon DocumentDB 會將一些內部命令計入 OpCountersCommand。

管理員資料庫和集合

Amazon DocumentDB 不支援管理員或本機資料庫，也不支援 MongoDB system.*或startup_log集合。

cursormaxTimeMS

在 Amazon DocumentDB 中， 會cursor.maxTimeMS重設每個getMore請求的計數器。因此，如果maxTimeMS指定 3000MS，則查詢需要 2800MS，而每個後續getMore請求需要 300MS，則游標不會逾時。游標只會在查詢或個別getMore請求的單一操作耗用超過指定的 時逾時maxTimeMS。此外，檢查游標執行時間的掃描器會以五 (5) 分鐘精細程度執行。

explain()

Amazon DocumentDB 在利用分散式、容錯、自我修復儲存系統的專用資料庫引擎上模擬 MongoDB 3.6、4.0、5.0 和 8.0 APIs。因此，Amazon DocumentDB 和 MongoDB 之間的查詢計劃和 的輸出explain()可能不同。想要控制其查詢計劃的客戶可以使用 $hint 運算子強制選取偏好的索引。

索引組建

Amazon DocumentDB 僅允許在任何指定時間在集合上發生一個索引組建。在前景或背景中。當索引建置目前正在進行時，如果在相同的集合上進行如 createIndex() 或 dropIndex() 等操作，最新嘗試進行的操作將會失敗。

根據預設，Amazon DocumentDB 和 MongoDB 4.0 版中的索引建置會在前景中發生。MongoDB 4.2 版和更新版本指定為 createIndexes 或其 Shell 協助程式`createIndex()`和 時，會忽略背景索引建置選項。 `createIndexes()`

存留時間 (TTL) 索引會在索引建置完成後開始過期文件。

路徑中具有空索引鍵的查詢

當您使用包含空字串做為路徑的一部分的索引鍵 （例如 `x.`、`x..b`) 進行查詢，且物件在陣列中具有空字串索引鍵路徑 （例如 `{"x" : [ { "" : 10 }, { "b" : 20 } ]}`) 時，Amazon DocumentDB 會傳回與在 MongoDB 中執行相同查詢不同的結果。

在 MongoDB 中，當空字串索引鍵不在路徑結尾時，陣列中的空索引鍵路徑查詢會如預期般運作。不過，當空字串索引鍵在路徑查詢結尾時，它不會查看陣列。

不過在 Amazon DocumentDB 中，只會讀取陣列中的第一個元素，因為 會將空字串`getArrayIndexFromKeyString`轉換為 `0`，因此字串索引鍵查詢會視為陣列索引查詢。

MongoDB APIs、操作和資料類型

Amazon DocumentDB 與 MongoDB 3.6、4.0、5.0 和 8.0 APIs。若要查看最新的支援功能清單，請參閱 Amazon DocumentDB 中支援的 MongoDB APIs、操作和資料類型。

mongodump 和 **mongorestore** 公用程式

Amazon DocumentDB 不支援管理員資料庫，因此在使用 或 `mongorestore`公用程式時不會傾印`mongodump`或還原管理員資料庫。當您使用 在 Amazon DocumentDB 中建立新的資料庫時`mongorestore`，除了還原操作之外，還需要重新建立使用者角色。

> MongoDB 資料庫工具版本需求
>
> 使用 MongoDB 資料庫工具，包括適用於 Amazon DocumentDB 的 100.11.0 版。若要下載，請參閱 MongoDB 網站上的 MongoDB 資料庫工具版本。 MongoDB

結果排序

Amazon DocumentDB 不保證結果集的隱含結果排序順序。為確保結果集的排序，請使用 `sort()` 明確指定排序順序。

以下範例會根據庫存欄位以遞減順序排序庫存集合中的項目。

```
db.inventory.find().sort({ stock: -1 })
```

使用$sort彙總階段時，除非$sort階段是彙總管道中的最後一個階段，否則不會保留排序順序。使用$sort彙總階段搭配$group彙總階段時，$sort彙總階段只會套用至 $last $first和 累積器。在 Amazon DocumentDB 4.0 中，新增了 的支援$push，以遵守上一個$sort階段的排序順序。

可重試的寫入

從 MongoDB 4.2 相容驅動程式開始，預設會啟用可重試寫入。不過，Amazon DocumentDB 目前不支援可重試的寫入。功能差異會在類似下列錯誤訊息中以資訊清單列出。

```
{"ok":0,"errmsg":"Unrecognized field: 'txnNumber'","code":9,"name":"MongoError"}
```

可透過連線字串（例如 MongoClient("mongodb://my.mongodb.cluster/db?retryWrites=false")) 或 MongoClient 建構函數的關鍵字引數（例如）停用可重試寫入MongoClient("mongodb://my.mongodb.cluster/db", retryWrites=False)。

以下是透過連線字串停用可重試寫入的 Python 範例。

```
client =
 pymongo.MongoClient('mongodb://
<username>:<password>@docdb-2019-03-17-16-49-12.cluster-ccuszbx3pn5e.us-
east-1.docdb.amazonaws.com:27017/?
replicaSet=rs0',w='majority',j=True,retryWrites=False)
```

稀鬆索引

若要使用您在查詢中建立的稀疏索引，您必須在涵蓋索引的欄位上使用 $exists 子句。如果您省略 $exists，Amazon DocumentDB 將不會使用稀疏索引。

下列是 範例。

```
db.inventory.count({ "stock": { $exists: true }})
```

對於稀疏的多索引鍵索引，如果查詢文件會導致一組值，且僅缺少一部分索引欄位，Amazon DocumentDB 不支援唯一的索引鍵限制條件。例如，若輸入為 "a" : [{ "b" : 2 }, { "c" :

1 }] 則不支援 createIndex({"a.b" : 1 }, { unique : true, sparse :true }) ，因為 "a.c" 會儲存在索引中。

在$all表達式$elemMatch內使用

Amazon DocumentDB 目前不支援在$all表達式中使用$elemMatch運算子。您可以如下所示搭配 $elemMatch 使用 $and ，解決問題。

原始操作：

```
db.col.find({
  qty: {
    $all: [
      { "$elemMatch": { part: "xyz", qty: { $lt: 11 } } },
      { "$elemMatch": { num: 40, size: "XL" } }
    ]
  }
})
```

更新的操作：

```
db.col.find({
  $and: [
    { qty: { "$elemMatch": { part: "xyz", qty: { $lt: 11 } } } },
    { qty: { "$elemMatch": { qty: 40, size: "XL" } } }
  ]
})
```

欄位名稱中的 Dollar($) 和 dot(.)

Amazon DocumentDB 不支援在巢狀物件中查詢 $in、$nin 和 $all 中的 Dollar($) 字首欄位。例如，下列查詢在 Amazon DocumentDB 中無效：

```
coll.find({"field": {"$all": [{ "$a": 1 }]}})
```

$lookup

Amazon DocumentDB 支援執行相等比對的能力 （ 例如，左外部聯結 ），也支援不相關的子查詢，但不支援關聯的子查詢。

使用索引搭配 $lookup

您現在可以將索引與$lookup階段運算子搭配使用。根據您的使用案例，您可以使用多種索引演算法來最佳化效能。本節將說明 的不同索引演算法，$lookup並協助您選擇最適合工作負載的索引演算法。

根據預設，Amazon DocumentDB 會在使用 allowDiskUse:false 時使用雜湊演算法，並在allowDiskUse:true使用 時排序合併。

> **Note**
>
> find 命令目前不支援 allowDiskUse選項。僅支援 選項做為彙總的一部分。使用彙總架構搭配 allowDiskUse:true來處理可能超過記憶體限制的大型查詢。

對於某些使用案例，可能需要強制查詢最佳化工具使用不同的演算法。以下是$lookup彙總運算子可以使用的不同索引演算法：

- 巢狀迴圈：如果外部集合 <1 GB，且外部集合中的 欄位具有索引，則巢狀迴圈計劃通常有利於工作負載。如果使用巢狀迴圈演算法，解釋計畫將顯示階段為 NESTED_LOOP_LOOKUP。
- 排序合併：如果外部集合在查詢中使用的欄位上沒有索引，且工作資料集不適合記憶體，則排序合併計畫通常有利於工作負載。如果正在使用排序合併演算法，解釋計畫會將階段顯示為 SORT_LOOKUP。
- 雜湊：如果外部集合 < 1 GB，且工作資料集適合記憶體，雜湊計劃通常有利於工作負載。如果使用雜湊演算法，解釋計畫會將階段顯示為 HASH_LOOKUP。

您可以在explain查詢上使用 來識別用於$lookup運算子的索引演算法。以下是範例：

```
db.localCollection.explain().aggregate(
   [
      {
         $lookup:
            {
               from: "foreignCollection",
               localField: "a",
               foreignField: "b",
               as: "joined"
            }
      }
```

```
   ]
)

output
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "test.localCollection",
        "winningPlan" : {
            "stage" : "SUBSCAN",
            "inputStage" : {
                "stage" : "SORT_AGGREGATE",
                "inputStage" : {
                    "stage" : "SORT",
                    "inputStage" : {
                        "stage" : "NESTED_LOOP_LOOKUP",
                        "inputStages" : [
                            {
                                "stage" : "COLLSCAN"
                            },
                            {
                                "stage" : "FETCH",
                                "inputStage" : {
                                    "stage" : "COLLSCAN"
                                }
                            }
                        ]
                    }
                }
            }
        }
    },
    "serverInfo" : {
        "host" : "devbox-test",
        "port" : 27317,
        "version" : "3.6.0"
    },
    "ok" : 1
}
```

除了使用 explain()方法之外，您也可以使用分析器來檢閱與您使用 $lookup運算子搭配使用的演算法。如需分析器的詳細資訊，請參閱 分析 Amazon DocumentDB 操作。

使用 **planHint**

如果您想要強制查詢最佳化工具搭配 使用不同的索引演算法$lookup，您可以使用 planHint。若要這樣做，請使用彙總階段選項中的註解來強制不同的計劃。以下是註解的語法範例：

```
comment : {
    comment :  "<string>",
    lookupStage : { planHint : "SORT" | "HASH" | "NESTED_LOOP" }
}
```

以下是使用 planHint強制查詢最佳化工具使用HASH索引演算法的範例：

```
db.foo.aggregate(
   [
      {
         $lookup:
            {
               from: "foo",
               localField: "_id",
               foreignField: "_id",
               as: "joined"
            },
      }
   ]
),
{
   comment : "{ \"lookupStage\" : { \"planHint\": \"HASH\" }}"
```

若要測試哪種演算法最適合您的工作負載，您可以使用 explain方法的 executionStats 參數來測量$lookup階段的執行時間，同時修改索引演算法 （即 HASH/SORT/NESTED_LOOP)。

下列範例示範如何使用 executionStats 來測量使用SORT演算法的$lookup階段執行時間。

```
db.foo.explain("executionStats").aggregate(
   [
      {
         $lookup:
            {
               from: "foo",
               localField: "_id",
```

```
                foreignField: "_id",
                as: "joined"
            },
        }
    ]
),
{
    comment : "{ \"lookupStage\" : { \"planHint\": \"SORT\" }}"
```

`$natural` 和反向排序

Amazon DocumentDB 僅支援`$natural`轉送集合掃描。反向收集掃描 (`{$natural: -1}`) 將導致 `MongoServerError`。

Amazon DocumentDB 中支援的 MongoDB APIs、操作和資料類型

Amazon DocumentDB （與 MongoDB 相容） 是一種快速、可擴展、高可用性且全受管的文件資料庫服務，可支援 MongoDB 工作負載。Amazon DocumentDB 與 MongoDB 3.6、4.0、5.0 和 8.0 APIs。本區段列出支援的功能。如需使用 MongoDB APIs和驅動程式的支援，請參閱 MongoDB 社群論壇。如需使用 Amazon DocumentDB 服務的支援，請聯絡適當的 AWS 支援團隊。如需 Amazon DocumentDB 和 MongoDB 之間的功能差異，請參閱 功能差異：Amazon DocumentDB 和 MongoDB。

不支援 MongoDB 命令和運算子，這些命令僅供內部使用或不適用於全受管服務，且未包含在支援的功能清單中。

自推出以來，我們已新增超過 50 項額外功能，並會繼續向客戶提供他們所需的功能。如需最新啟動的資訊，請參閱 Amazon DocumentDB 公告。

如果您有不支援我們建置的功能，請使用您的 accountID、請求的功能和使用案例傳送給 Amazon DocumentDB 服務團隊，讓我們知道。

主題

- 資料庫命令
- 查詢和投影運算子
- 更新運算子
- Geospatial (地理空間)

- 游標方法
- 彙總管道運算子
- 資料類型
- 索引

資料庫命令

主題

- 管理命令
- 聚合
- 身分驗證
- 診斷命令
- 查詢和寫入操作
- 角色管理命令
- 工作階段命令
- 使用者管理
- 碎片命令

管理命令

命令	3.6	4.0	5.0	8.0	彈性叢集
Capped Collections	否	否	否	否	否
cloneCollectionAsCapped	否	否	否	否	否
collMod	部分	部分	部分	部分	部分
collMod: expireAfterSeconds	是	是	是	是	是

命令	3.6	4.0	5.0	8.0	彈性叢集
collMod：隱藏	否	否	否	是 (8.0.1+)	否
convertToCapped	否	否	否	否	否
copydb	否	否	否	否	否
建立	是	是	是	是	是
createView	否	否	否	是	否
createIndexes	是	是	是	是	是
currentOp	是	是	是	是	是
drop	是	是	是	是	是
dropDatabase	是	是	是	是	是
dropIndexes	是	是	是	是	是
filemd5	否	否	否	否	否
getAuditConfig	否	是	是	是	否
killCursors	是	是	是	是	是
killOp	是	是	是	是	是
listCollections*	是	是	是	是	是
listDatabases	是	是	是	是	是
listIndexes	是	是	是	是	是

命令	3.6	4.0	5.0	8.0	彈性叢集
reIndex	否	否	是	是	否
renameCollection	是	是	是	是	否
setAuditConfig	否	是	是	是	否

* 不支援篩選選項中的type金鑰。

聚合

命令	3.6	4.0	5.0	8.0	彈性叢集
aggregate	是	是	是	是	是
count	是	是	是	是	是
distinct	是	是	是	是	是
mapReduce	否	否	否	是	否

身分驗證

命令	3.6	4.0	5.0	8.0	彈性叢集
authenticate	是	是	是	是	是
登出	是	是	是	是	是

診斷命令

命令	3.6	4.0	5.0	8.0	彈性叢集
buildInfo	是	是	是	是	是
collStats	是	是	是	是	是
connPoolStats	否	否	否	否	否
connectionStatus	是	是	是	是	是
dataSize	是	是	是	是	是
dbHash	否	否	否	否	否
dbStats	是	是	是	是	是
explain	是	是	是	是	是
explain: executionStats	是	是	是	是	是
功能	否	否	否	否	否
hostInfo	是	是	是	是	是
listCommands	是	是	是	是	是
profiler	是	是	是	是	否
serverStatus	是	是	是	是	是
top	是	是	是	是	是

查詢和寫入操作

命令	3.6	4.0	5.0	8.0	彈性叢集
Change streams	是	是	是	是	否
delete	是	是	是	是	是
find	是	是	是	是	是
findAndModify	是	是	是	是	是
getLastError	否	否	否	否	否
getMore	是	是	是	是	是
getPrevError	否	否	否	否	否
GridFS	是	是	是	是	否
insert	是	是	是	是	是
parallelCollectionScan	否	否	否	否	否
resetError	否	否	否	否	否
update	是	是	是	是	是
ReplaceOne	是	是	是	是	是

角色管理命令

命令	3.6	4.0	5.0	8.0	彈性叢集
c	是	是	是	是	否
e					
c	是	是	是	是	否
le					
ta					
d	是	是	是	是	否
g	是	是	是	是	否
s					
r	是	是	是	是	否
e					
e					
r	是	是	是	是	否
v					
c					
r	是	是	是	是	否
u	是	是	是	是	否
e					

工作階段命令

命令	3.6	4.0	5.0	8.0	彈性叢集
abortTransaction	否	是	是	是	否

命令	3.6	4.0	5.0	8.0	彈性叢集
commitTransaction	否	是	是	是	否
endSessions	否	否	否	否	否
killAllSessions	否	是	是	是	否
killAllSessionsByPattern	否	否	否	否	否
killSessions	否	是	是	是	否
refreshSessions	否	否	否	否	否
startSession	否	是	是	是	否

使用者管理

命令	3.6	4.0	5.0	8.0	彈性叢集
createUser	是	是	是	是	是
dropAllUsersFromDatabase	是	是	是	是	是
dropUser	是	是	是	是	是
grantRolesToUser	是	是	是	是	是
revokeRolesFromUser	是	是	是	是	是

命令	3.6	4.0	5.0	8.0	彈性叢集
updateUser	是	是	是	是	是
usersInfo	是	是	是	是	是

碎片命令

命令	彈性叢集
abortReshardCollection	否
addShard	否
addShardToZone	否
balancerCollectionStatus	否
balancerStart	否
balancerStatus	否
balancerStop	否
checkShardingIndex	否
clearJumboFlag	否
cleanupOrphaned	否
cleanupReshardCollection	否
commitReshardCollection	否
enableSharding	是
flushRouterConfig	否
getShardMap	否

命令	彈性叢集
getShardVersion	否
isdbgrid	否
listShards	否
medianKey	否
moveChunk	否
movePrimary	否
mergeChunks	否
refineCollectionShardKey	否
removeShard	否
removeShardFromZone	否
reshardCollection	否
setAllowMigrations	否
setShardVersion	否
shardCollection	是
shardingState	否
分割	否
splitVector	否
unsetSharding	否
updateZoneKeyRange	否

查詢和投影運算子

主題

- 陣列運算子
- 位元運算子
- 註解運算子
- 比較運算子
- 元素運算子
- 評估查詢運算子
- 邏輯運算子
- 投影運算子

陣列運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$all	是	是	是	是	是
$elemMatch	是	是	是	是	是
$size	是	是	是	是	是

位元運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$bitsAllSet	是	是	是	是	是
$bitsAnySet	是	是	是	是	是
$bitsAllClear	是	是	是	是	是
$bitsAnyClear	是	是	是	是	是

註解運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$comment	是	是	是	是	是

比較運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$eq	是	是	是	是	是
$gt	是	是	是	是	是
$gte	是	是	是	是	是
$in	是	是	是	是	是
$lt	是	是	是	是	是
$lte	是	是	是	是	是
$ne	是	是	是	是	是
$nin	是	是	是	是	是

元素運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$exists	是	是	是	是	是
$type	是	是	是	是	是

評估查詢運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$expr	否	是	是	是	否
$jsonSchema	否	是	是	是	否
$mod	是	是	是	是	是
$regex	是	是	是	是	是
$text	否	否	是	是	否
$where	否	否	否	否	否

邏輯運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$and	是	是	是	是	是
$nor	是	是	是	是	是
$not	是	是	是	是	是
$or	是	是	是	是	是

投影運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$	是	是	是	是	是
$elemMatch	是	是	是	是	是
$meta	否	否	是	是	否

命令	3.6	4.0	5.0	8.0	彈性叢集
$slice	是	是	是	是	是

更新運算子

主題

- 陣列運算子
- 位元運算子
- 欄位運算子
- 更新修飾詞

陣列運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$	是	是	是	是	是
$[]	是	是	是	是	是
$[<identifier>]	是	是	是	是	是
$addToSet	是	是	是	是	是
$pop	是	是	是	是	是
$pullAll	是	是	是	是	是
$pull	是	是	是	是	是
$push	是	是	是	是	是

位元運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$bit	是	是	是	是	是

欄位運算子

運算子	3.6	4.0	5.0	8.0	彈性叢集
$currentDate	是	是	是	是	是
$inc	是	是	是	是	是
$max	是	是	是	是	是
$min	是	是	是	是	是
$mul	是	是	是	是	是
$rename	是	是	是	是	是
$set	是	是	是	是	是
$setOnInsert	是	是	是	是	是
$unset	是	是	是	是	是

更新修飾詞

運算子	3.6	4.0	5.0	8.0	彈性叢集
$each	是	是	是	是	是
$position	是	是	是	是	是
$slice	是	是	是	是	是

運算子	3.6	4.0	5.0	8.0	彈性叢集
$sort	是	是	是	是	是

Geospatial (地理空間)

幾何指標

查詢選取器	3.6	4.0	5.0	8.0	彈性叢集
$box	否	否	否	否	否
$center	否	否	否	否	否
$centerSphere	否	否	否	否	否
$geometry	是	是	是	是	是
$maxDistance	是	是	是	是	是
$minDistance	是	是	是	是	是
$nearSphere	是	是	是	是	是
$polygon	否	否	否	否	否
$uniqueDocs	否	否	否	否	否

查詢選取器

命令	3.6	4.0	5.0	8.0	彈性叢集
$geoIntersects	是	是	是	是	是
$geoWithin	是	是	是	是	是

命令	3.6	4.0	5.0	8.0	彈性叢集
$near	是	是	是	是	是
$nearSphere	是	是	是	是	是
$polygon	否	否	否	否	否
$uniqueDocs	否	否	否	否	否

游標方法

命令	3.6	4.0	5.0	8.0	彈性叢集
cursor.batchSize()	是	是	是	是	是
cursor.close()	是	是	是	是	是
cursor.collation()	否	否	否	是	否
cursor.comment()	是	是	是	是	是
cursor.count()	是	是	是	是	是
cursor.explain()	是	是	是	是	否
cursor.forEach()	是	是	是	是	是
cursor.hasNext()	是	是	是	是	是
cursor.hint()	是	是	是	是	是*

命令	3.6	4.0	5.0	8.0	彈性叢集
cursor.isClosed()	是	是	是	是	是
cursor.isExhausted()	是	是	是	是	否
cursor.itcount()	是	是	是	是	否
cursor.limit()	是	是	是	是	否
cursor.map()	是	是	是	是	否
cursor.max()	否	否	否	是 (8.0.1+)	否
cursor.maxScan()	是	是	是	是	否
cursor.maxTimeMS()	是	是	是	是	否
cursor.min()	否	否	否	是 (8.0.1+)	否
cursor.next()	是	是	是	是	是
cursor.noCursorTimeout()	否	否	否	否	否
cursor.objsLeftInBatch()	是	是	是	是	否
cursor.pretty()	是	是	是	是	否
cursor.readConcern()	是	是	是	是	否

命令	3.6	4.0	5.0	8.0	彈性叢集
cursor.readPref()	是	是	是	是	否
cursor.returnKey()	否	否	否	否	否
cursor.showRecordId()	否	否	否	否	否
cursor.size()	是	是	是	是	否
cursor.skip()	是	是	是	是	否
cursor.sort()	是	是	是	是	否
cursor.tailable()	否	否	否	否	否
cursor.toArray()	是	是	是	是	否

* 索引表達式hint支援索引。例如 `db.foo.find().hint({x:1})`。

彙總管道運算子

主題

- 累積器表達式
- 算術運算子
- 陣列運算子
- 布林值運算子
- 比較運算子
- 條件式表達式運算子
- 資料類型運算子
- 資料大小運算子

- 日期運算子
- 文字運算子
- 合併運算子
- 自然運算子
- 設定運算子
- 階段運算子
- 字串運算子
- 系統變數
- 文字搜尋運算子
- 類型轉換運算子
- 變數運算子
- 三角運算子
- 位元運算子（彙總）
- 時間戳記運算子
- 其他運算子

累積器表達式

表達式	3.6	4.0	5.0	8.0	彈性叢集
$累積器	-	-	否	否	否
$addToSet	是	是	是	是	是
$avg	是	是	是	是	是
$底部	-	-	-	是 (8.0.1+)	否
$bottomN	-	-	-	是 (8.0.1+)	否
$count	-	-	否	是 (8.0.1+)	否
$covarianCePop	否	否	否	否	否

表達式	3.6	4.0	5.0	8.0	彈性叢集
$covarianceSamp	否	否	否	否	否
$denseRank	否	否	否	否	否
$衍生性	否	否	否	否	否
$documentNumber	否	否	否	否	否
$expMovingAvg	否	否	否	否	否
$first	是	是	是	是	是
$firstN	-	-	-	是 (8.0.1+)	否
$整合	否	否	否	否	否
$last	是	是	是	是	是
$lastN	-	-	-	是 (8.0.1+)	否
$max	是	是	是	是	是
$maxN	-	-	-	是 (8.0.1+)	否
$median	-	-	-	是 (8.0.1+)	否
$min	是	是	是	是	是
$minN	-	-	-	是 (8.0.1+)	否
$percentile	-	-	-	是 (8.0.1+)	否
$push	是	是	是	是	是
$rank	否	否	否	否	否
$shift	否	否	否	否	否

表達式	3.6	4.0	5.0	8.0	彈性叢集
$stdDevPop	否	否	否	是 (8.0.1+)	否
$stdDevSamp	否	否	否	是 (8.0.1+)	否
$sum	是	是	是	是	是
$top	-	-	-	是 (8.0.1+)	否
$topN	-	-	-	是 (8.0.1+)	否

算術運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$abs	是	是	是	是	是
$add	是	是	是	是	是
$ceil	否	是	是	是	是
$divide	是	是	是	是	是
$exp	否	是	是	是	是
$floor	否	是	是	是	是
$ln	否	是	是	是	是
$log	否	是	是	是	是
$log10	否	是	是	是	是
$mod	是	是	是	是	是
$multiply	是	是	是	是	是
$pow	否	否	是 (5.0.1+)	是	否

命令	3.6	4.0	5.0	8.0	彈性叢集
$round	-	-	否	是 (8.0.1+)	否
$sqrt	否	是	是	是	是
$subtract	是	是	是	是	是
$trunc	否	否	否	是 (8.0.1+)	否

陣列運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$arrayElemAt	是	是	是	是	是
$arrayToObject	是	是	是	是	是
$concatArrays	是	是	是	是	是
$filter	是	是	是	是	是
$first	-	-	是	是	否
$in	是	是	是	是	是
$indexOfArray	是	是	是	是	是
$isArray	是	是	是	是	是
$last	-	-	是	是	否
$objectToArray	是	是	是	是	是
$range	是	是	是	是	是

命令	3.6	4.0	5.0	8.0	彈性叢集
$reverseArray	是	是	是	是	是
$sortArray	否	否	否	是	否
$reduce	是	是	是	是	是
$size	是	是	是	是	是
$slice	是	是	是	是	是
$zip	是	是	是	是	是

布林值運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$and	是	是	是	是	是
$not	是	是	是	是	是
$or	是	是	是	是	是

比較運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$cmp	是	是	是	是	是
$eq	是	是	是	是	是
$gt	是	是	是	是	是
$gte	是	是	是	是	是
$lt	是	是	是	是	是

命令	3.6	4.0	5.0	8.0	彈性叢集
$lte	是	是	是	是	是
$ne	是	是	是	是	是

條件式表達式運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$cond	是	是	是	是	是
$ifNull	是	是	是	是	是
$switch	否	是	是	是	否

資料類型運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$type	是	是	是	是	是

資料大小運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$binarySize	-	-	否	是 (8.0.1+)	否
$bsonSize	-	-	否	是 (8.0.1+)	否

日期運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$dateAdd	否	否	是	是	是

命令	3.6	4.0	5.0	8.0	彈性叢集
$dateDiff	-	-	是	是	否
$dateFromParts	否	否	是 (5.0.1+)	否	否
$dateFromString	是	是	是	是	是
$dateSubtract	否	否	是	是	是
$dateToParts	否	否	是 (5.0.1+)	否	否
$dateToString	是	是	是	是	是
$dateTrunc	-	-	否	是	否
$dayOfMonth	是	是	是	是	是
$dayOfWeek	是	是	是	是	是
$dayOfYear	是	是	是	是	是
$hour	是	是	是	是	是
$isoDayOfWeek	是	是	是	是	是
$isoWeek	是	是	是	是	是
$isoWeekYear	是	是	是	是	是
$millisecond	是	是	是	是	是
$minute	是	是	是	是	是
$month	是	是	是	是	是

命令	3.6	4.0	5.0	8.0	彈性叢集
$second	是	是	是	是	是
$week	是	是	是	是	是
$year	是	是	是	是	是

文字運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$literal	是	是	是	是	是

合併運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$mergeObjects	是	是	是	是	是

自然運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$自然	是	是	是	是	是

設定運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$allElementsTrue	否	是	是	是	是

命令	3.6	4.0	5.0	8.0	彈性叢集
$anyElementTrue	否	是	是	是	是
$setDifference	否	是	是	是	是
$setEquals	是	是	是	是	是
$setIntersection	是	是	是	是	是
$setIsSubset	是	是	是	是	是
$setUnion	是	是	是	是	是
$setWindowFields	否	否	否	否	否

階段運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$addFields	是	是	是	是	是
$bucket	否	否	否	是	否
$bucketAuto	否	否	否	否	
$changeStream	是	是	是	是	否
$collStats	否	是	是	是	否
$count	是	是	是	是	是
$currentOp	是	是	是	是	是

命令	3.6	4.0	5.0	8.0	彈性叢集
$facet	否	否	否	否	否
$geoNear	是	是	是	是	是
$graphLookup	否	否	否	否	否
$group	是	是	是	是	是
$indexStats	是	是	是	是	否
$limit	是	是	是	是	是
$listLocalSessions	否	否	否	否	否
$listSessions	否	否	否	否	否
$lookup	是	是	是	是	是
$match	是	是	是	是	是
$merge	-	-	否	是	否
$out	是	是	是	是	否
$planCacheStats	-	-	否	否	否
$project	是	是	是	是	是
$redact	是	是	是	是	是
$replaceRoot	是	是	是	是	是
$sample	是	是	是	是	是
$set	-	-	否	是	否

命令	3.6	4.0	5.0	8.0	彈性叢集
$setWindowFields	-	-	否	否	否
$skip	是	是	是	是	是
$sort	是	是	是	是	是
$sortByCount	否	否	否	是 (8.0.1+)	否
$unionWith	-	-	否	否	否
$unset	-	-	否	是	否
$unwind	是	是	是	是	是
$replaceWith	否	否	否	是	否
$vectorSearch	否	否	否	是	否
$listSearchIndexes	-	-	-	是 (8.0.1+)	否

字串運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$concat	是	是	是	是	是
$indexOfBytes	是	是	是	是	是
$indexOfCP	是	是	是	是	是
$ltrim	否	是	是	是	否
$regexFind	-	-	是	是	否

命令	3.6	4.0	5.0	8.0	彈性叢集
$regexFindAll	-	-	是	是	否
$regexMatch	-	-	是	是	否
$replaceAll	-	-	是	是	否
$replaceOne	-	-	是	是	否
$rtrim	否	是	是	是	否
$split	是	是	是	是	是
$strcasecmp	是	是	是	是	是
$strLenBytes	是	是	是	是	是
$strLenCP	是	是	是	是	是
$substr	是	是	是	是	是
$substrBytes	是	是	是	是	是
$substrCP	是	是	是	是	是
$toLower	是	是	是	是	是
$toUpper	是	是	是	是	是
$trim	否	是	是	是	否

系統變數

命令	3.6	4.0	5.0	8.0	彈性叢集
$$CURRENT	否	否	否	否	否
$$DESCEND	是	是	是	是	是

命令	3.6	4.0	5.0	8.0	彈性叢集
$$KEEP	是	是	是	是	是
$$PRUNE	是	是	是	是	是
$$REMOVE	否	否	否	否	否
$ROOT	是	是	是	是	是

文字搜尋運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$meta	否	否	是	是	否
$search	否	否	是	是	否

類型轉換運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$轉換	否	是	是	是	是
$isNumber	-	-	否	是 (8.0.1+)	否
$toBool	否	是	是	是	是
$toDate	否	是	是	是	是
$toDecimal	否	是	是	是	是
$toDouble	否	是	是	是	是
$toInt	否	是	是	是	是
$toLong	否	是	是	是	是

命令	3.6	4.0	5.0	8.0	彈性叢集
$toObjectId	否	是	是	是	是
$toString	否	是	是	是	是
$toUUID	-	-	-	是 (8.0.1+)	否

變數運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$let	是	是	是	是	是
$map	是	是	是	是	是

三角運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$acos	-	-	-	是 (8.0.1+)	否
$acosh	-	-	-	是 (8.0.1+)	否
$asin	-	-	-	是 (8.0.1+)	否
$asinh	-	-	-	是 (8.0.1+)	否
$atan	-	-	-	是 (8.0.1+)	否
$atan2	-	-	-	是 (8.0.1+)	否
$atanh	-	-	-	是 (8.0.1+)	否
$cos	-	-	-	是 (8.0.1+)	否
$cosh	-	-	-	是 (8.0.1+)	否

命令	3.6	4.0	5.0	8.0	彈性叢集
$degreesToRadians	-	-	-	是 (8.0.1+)	否
$radiansToDegrees	-	-	-	是 (8.0.1+)	否
$sin	-	-	-	是 (8.0.1+)	否
$sinh	-	-	-	是 (8.0.1+)	否
$tan	-	-	-	是 (8.0.1+)	否
$tanh	-	-	-	是 (8.0.1+)	否

位元運算子（彙總）

命令	3.6	4.0	5.0	8.0	彈性叢集
$bitAnd	-	-	-	是 (8.0.1+)	否
$bitNot	-	-	-	是 (8.0.1+)	否
$bitOr	-	-	-	是 (8.0.1+)	否
$bitXor	-	-	-	是 (8.0.1+)	否

時間戳記運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$tsIncrement	-	-	-	是 (8.0.1+)	否
$tsSecond	-	-	-	是 (8.0.1+)	否

其他運算子

命令	3.6	4.0	5.0	8.0	彈性叢集
$getField	-	-	否	否	否
$rand	-	-	是 (5.0.1+)	是	否
$sampleRate	-	-	否	是 (8.0.1+)	否
$sigmoid	-	-	-	是 (8.0.1+)	否

資料類型

命令	3.6	4.0	5.0	8.0	彈性叢集
32 位元整數 (int)	是	是	是	是	是
64 位元整數（長）	是	是	是	是	是
陣列	是	是	是	是	是
二進位資料	是	是	是	是	是
Boolean	是	是	是	是	是
日期	是	是	是	是	是
DBPointer	否	否	否	否	否
DBRefs	否	否	否	否	否
Decimal128	是	是	是	是	是
Double	是	是	是	是	是
JavaScript	否	否	否	否	否

命令	3.6	4.0	5.0	8.0	彈性叢集
JavaScript（含範圍）	否	否	否	否	否
MaxKey	是	是	是	是	是
MinKey	是	是	是	是	是
Null	是	是	是	是	是
物件	是	是	是	是	是
ObjectId	是	是	是	是	是
Regular expression (常規表達式)	是	是	是	是	是
String	是	是	是	是	是
Symbol	否	否	否	否	否
時間戳記	是	是	是	是	是
未定義	否	否	否	否	否

索引和索引屬性

主題

- 索引
- 索引屬性

索引

命令	3.6	4.0	5.0	8.0	彈性叢集
2dsphere	是	是	是	是	是

命令	3.6	4.0	5.0	8.0	彈性叢集
2d 索引	否	否	否	否	否
複合索引	是	是	是	是	是
雜湊索引	否	否	否	否	否
多索引鍵索引	是	是	是	是	是
單一欄位索引	是	是	是	是	是
文字索引	否	否	是	是	否
萬用字元	否	否	否	否	否

索引屬性

命令	3.6	4.0	5.0	8.0	彈性叢集
背景介紹	是	是	是	是	是
不區分大小寫	否	否	否	是	否
Hidden	否	否	否	是 (8.0.1+)	否
部分	否	否	是	是	否
稀疏	是	是	是	是	是
文字	否	否	是	是	否
TTL	是	是	是	是	是
唯一	是	是	是	是	是
向量	否	否	是	是	否

如需特定 MongoDB 運算子的詳細資訊，請參閱下列主題：

- 彙總管道運算子
- Geospatial (地理空間)
- 投影運算子
- 更新運算子

彙總管道運算子

本節提供有關 Amazon DocumentDB 支援的彙總管道運算子的詳細資訊。

主題

- $$DESCEND
- $$KEEP
- $$PRUNE
- $ROOT
- $abs
- $acos
- $acosh
- $add
- $addToSet
- $addFields
- $allElementsTrue
- $and
- $anyElementTrue
- $arrayElemAt
- $arrayToObject
- $asin
- $asinh
- $atan
- $atan2
- $atanh

- $avg
- $binarySize
- $bitAnd
- $bitNot
- $bitOr
- $bitXor
- $底部
- $bottomN
- $bsonSize
- $bucket
- $ceil
- $changeStream
- $cmp
- $collStats
- $concat
- $concatArrays
- $cond
- $轉換
- $cos
- $cosh
- $count （階段）
- $count （累積器）
- $currentOp
- $dateAdd
- $dateDiff
- $dateFromParts
- $dateFromString
- $dateSubtract
- $dateToParts

- $dateToString
- $dateTrunc
- $dayOfMonth
- $dayOfWeek
- $dayOfYear
- $degreesToRadians
- $divide
- $eq
- $exp
- $filter
- $first
- $firstN
- $floor
- $geoNear
- $group
- $gt
- $gte
- $hour
- $ifNull
- $in
- $indexOfArray
- $indexOfBytes
- $indexOfCP
- $indexStats
- $isArray
- $isNumber
- $isoDayOfWeek
- $isoWeek
- $isoWeekYear

- $last
- $lastN
- $let
- $limit
- $listSearchIndexes
- $literal
- $ln
- $log
- $log10
- $lt
- $lte
- $lookup
- $ltrim
- $map
- $match
- $max
- $maxN
- $median
- $meta
- $merge
- $mergeObjects
- $millisecond
- $min
- $minN
- $minute
- $mod
- $month
- $multiply
- $自然

- $ne
- $not
- $objectToArray
- $or
- $out
- $percentile
- $pow
- $push
- $project
- $radiansToDegrees
- $rand
- $range
- $redact
- $reduce
- $regexFind
- $regexFindAll
- $regexMatch
- $replaceAll
- $replaceOne
- $replaceRoot
- $replaceWith
- $reverseArray
- $round
- $rtrim
- $sample
- $search
- $second
- $set
- $setDifference

- $setEquals
- $setIntersection
- $setIsSubset
- $setUnion
- $sigmoid
- $sin
- $sinh
- $skip
- $slice
- $size
- $sort
- $sortArray
- $sortByCount
- $split
- $sqrt
- $stdDevPop
- $stdDevSamp
- $strLenBytes
- $strLenCP
- $strcasecmp
- $substr
- $substrBytes
- $substrCP
- $subtract
- $sum
- $switch
- $tan
- $tanh
- $top

- $topN
- $trunc
- $tsIncrement
- $tsSecond
- $toBool
- $toDate
- $toDecimal
- $toDouble
- $toInt
- $toLong
- $toLower
- $toObjectId
- $toString
- $toUpper
- $toUUID
- $trim
- $type
- $unset
- $unwind
- $vectorSearch
- $week
- $year
- $zip

$$DESCEND

Amazon DocumentDB 中的$$DESCEND運算子是在$redact管道階段中使用的特殊位置陣列運算子。它會指示彙總管道遞減至目前的文件並處理所有欄位，無論其巢狀層級為何。

當$redact階段遇到$$DESCEND運算子時，它會讓目前文件中的所有欄位保持可見，並進一步處理它們。當您想要根據條件選擇性地修訂或剪除特定欄位，同時保留文件的結構時，這非常有用。

參數

無.

範例 (MongoDB Shell)

在此範例中，我們將使用 $redact階段搭配 $$DESCEND 運算子，選擇性地顯示 code 欄位等於 "Reg" 的文件。

建立範例文件

```
db.patient.insertMany([
  { "_id": 1, "code": "Emp", "patient": "John Doe", "DOB": "1/1/1980", "Hospital":
 "Main" },
  { "_id": 2, "code": "Reg", "patient": "Jane Doe", "DOB": "3/27/1989", "Hospital":
 "Cherry Hill" },
  { "_id": 3, "code": "Emp", "patient": "Bob Smith", "DOB": "6/15/1975", "Hospital":
 "Downtown" }
]);
```

查詢範例

```
db.patient.aggregate([
  { $redact: {
    $cond: {
      if: { $eq: ["Reg", "$code"] },
      then: "$$DESCEND",
      else: "$$PRUNE"
    }
  }}
]);
```

輸出

```
{
  "_id": 2,
  "code": "Reg",
  "patient": "Jane Doe",
  "DOB": "3/27/1989",
  "Hospital": "Cherry Hill"
}
```

程式碼範例

若要檢視使用 $$DESCEND命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function redactPatients() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('patient');

  const result = await collection.aggregate([
    { $redact: {
      $cond: {
        if: { $eq: ["Reg", "$code"] },
        then: "$$DESCEND",
        else: "$$PRUNE"
      }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

redactPatients();
```

Python

```
from pymongo import MongoClient

def redact_patients():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.patient

    result = list(collection.aggregate([
```

```
        { "$redact": {
            "$cond": {
                "if": { "$eq": ["Reg", "$code"] },
                "then": "$$DESCEND",
                "else": "$$PRUNE"
            }
        }}
    ]))

    print(result)
    client.close()

redact_patients()
```

$$KEEP

$$KEEP 系統變數會與彙總管道中的$redact階段搭配使用，以保持目前文件或欄位不變，並將其包含在輸出中。

參數

無

範例 (MongoDB Shell)

下列範例示範在 Amazon DocumentDB 彙總管道$$KEEP中使用 。只有在存取等於「公開」時才會保留文件，否則會將其移除。

建立範例文件

```
db.articles.insertMany([
  { title: "Article A", access: "public", content: "Visible content" },
  { title: "Article B", access: "private", content: "Hidden content" }
]);
```

查詢範例

```
db.articles.aggregate([
  {
    $redact: {
      $cond: [
        { $eq: ["$access", "public"] },
```

```
        "$$KEEP",
        "$$PRUNE"
      ]
    }
  }
]);
```

輸出

```
[
  {
    "_id" : ObjectId("..."),
    "title" : "Article A",
    "access" : "public",
    "content" : "Visible content"
  }
]
```

程式碼範例

若要檢視使用 $$KEEP命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function run() {
  const client = new MongoClient(
    'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-bundle.pem&replicaSet=rs0'
  );

  try {
    await client.connect();
    const db = client.db('test');
    const articles = db.collection('articles');

    const pipeline = [
      {
        $redact: {
          $cond: [
            { $eq: ["$access", "public"] },
            "$$KEEP",
```

```
            "$$PRUNE"
          ]
        }
      }
    ];

    const results = await articles.aggregate(pipeline).toArray();
    console.log(results);
  } finally {
    await client.close();
  }
}

run().catch(console.error);
```

Python

```
from pymongo import MongoClient

client = MongoClient(
    "mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-bundle.pem&replicaSet=rs0"
)

db = client.test
articles = db.articles

pipeline = [
    {
        "$redact": {
            "$cond": [
                {"$eq": ["$access", "public"]},
                "$$KEEP",
                "$$PRUNE"
            ]
        }
    }
]

results = list(articles.aggregate(pipeline))
print(results)

client.close()
```

$$PRUNE

$$PRUNE 系統變數會與彙總管道中的 $redact階段搭配使用，以從結果中排除文件或內嵌文件層級。當條件評估為 時$$PRUNE，目前的文件或子文件會從輸出中移除。它通常與 $$DESCEND（保留和周遊文件） 或 $$KEEP（保留文件的所有層級） 搭配使用。

參數

無. $$PRUNE 系統變數在沒有參數的情況下使用，且必須與 搭配使用$redact。

範例 (MongoDB Shell)

下列範例示範如何使用 $$PRUNE搭配 $redact，從結果中排除超過 30 年的使用者。

建立範例文件

```
db.users.insert([
  { _id:1, name: "Carlos Salazar", age: 35, address: { street: "123 Main St", city:
 "Anytown", state: "CA" } },
  { _id:2, name: "Saanvi Sarkar", age: 28, address: { street: "456 Oak Rd", city:
 "Someplace", state: "NY" } },
  { _id:3, name: "Li Juan", age: 42, address: { street: "789 Pine Ave", city:
 "Springfield", state: "TX" } }
])
```

查詢範例

```
db.users.aggregate([
  {
    $redact: {
      $cond: {
        if: { $gt: ["$age", 30] },
        then: "$$PRUNE",
        else: "$$DESCEND"
      }
    }
  }
])
```

輸出

```
[
  {
```

```
    "_id": 2,
    "name": "Saanvi Sarkar",
    "age": 28,
    "address": {
      "street": "456 Oak Rd",
      "city": "Someplace",
      "state": "NY"
    }
  }
]
```

程式碼範例

若要檢視使用 $$PRUNE命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

async function main() {
  await client.connect();
  const db = client.db('test');
  const users = db.collection('users');

  const result = await users.aggregate([
    {
      $redact: {
        $cond: {
          if: { $gt: ["$age", 30] },
          then: "$$PRUNE",
          else: "$$DESCEND"
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}
```

```
main();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
users = db['users']

result = list(users.aggregate([
    {
        '$redact': {
            '$cond': {
                'if': { '$gt': ['$age', 30] },
                'then': '$$PRUNE',
                'else': '$$DESCEND'
            }
        }
    }
]))

print(result)
client.close()
```

$ROOT

Amazon DocumentDB 中的$ROOT運算子用於參考彙總管道中的整個輸入文件。它可讓您存取和操作完整的文件，包括其所有巢狀欄位和結構。

參數

無

範例 (MongoDB Shell)

此範例示範使用 $ROOT來建立稽核日誌，該日誌會擷取完整的原始文件以及處理時的中繼資料。

建立範例文件

```
db.orders.insertMany([
```

```
  {
    _id: 1,
    orderId: "ORD-2024-001",
    customer: "María García",
    email: "maría@example.com",
    items: [
      { product: "Laptop", quantity: 1, price: 1299.99 }
    ],
    totalAmount: 1299.99
  },
  {
    _id: 2,
    orderId: "ORD-2024-002",
    customer: "Arnav Desai",
    email: "arnav@example.com",
    items: [
      { product: "Mouse", quantity: 2, price: 29.99 },
      { product: "Keyboard", quantity: 1, price: 89.99 }
    ],
    totalAmount: 149.97
  }
]);
```

查詢範例

```
db.orders.aggregate([
  {
    $project: {
      processedAt: new Date(),
      originalDocument: "$$ROOT",
      summary: {
        $concat: [
          "Order ",
          "$orderId",
          " for ",
          "$customer",
          " - Total: $",
          { $toString: "$totalAmount" }
        ]
      }
    }
  }
]);
```

輸出

```
[
  {
    _id: 1,
    processedAt: ISODate('2025-11-24T20:43:51.492Z'),
    originalDocument: {
      _id: 1,
      orderId: 'ORD-2024-001',
      customer: 'María García',
      email: 'maría@example.com',
      items: [ { product: 'Laptop', quantity: 1, price: 1299.99 } ],
      totalAmount: 1299.99
    },
    summary: 'Order ORD-2024-001 for María García - Total: $1299.99'
  },
  {
    _id: 2,
    processedAt: ISODate('2025-11-24T20:43:51.492Z'),
    originalDocument: {
      _id: 2,
      orderId: 'ORD-2024-002',
      customer: 'Arnav Desai',
      email: 'arnav@example.com',
      items: [
        { product: 'Mouse', quantity: 2, price: 29.99 },
        { product: 'Keyboard', quantity: 1, price: 89.99 }
      ],
      totalAmount: 149.97
    },
    summary: 'Order ORD-2024-002 for Arnav Desai - Total: $149.97'
  }
]
```

程式碼範例

若要檢視使用 $ROOT命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function createAuditLog() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const orders = db.collection('orders');

  const result = await orders.aggregate([
    {
      $project: {
        processedAt: new Date(),
        originalDocument: "$$ROOT",
        summary: {
          $concat: [
            "Order ",
            "$orderId",
            " for ",
            "$customer",
            " - Total: $",
            { $toString: "$totalAmount" }
          ]
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

createAuditLog();
```

Python

```
from pymongo import MongoClient
from datetime import datetime

def create_audit_log():
    client = MongoClient('mongodb://username:password@docdb-
cluster.cluster-123456789.us-east-1.docdb.amazonaws.com:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['orders']
```

```
    result = list(collection.aggregate([
        {
            '$project': {
                'processedAt': datetime.now(),
                'originalDocument': '$$ROOT',
                'summary': {
                    '$concat': [
                        "Order ",
                        "$orderId",
                        " for ",
                        "$customer",
                        " - Total: $",
                        { '$toString': "$totalAmount" }
                    ]
                }
            }
        }
    ]))

    print(result)
    client.close()

create_audit_log()
```

$abs

Amazon DocumentDB 中的$abs運算子會傳回數字的絕對值。它可用於彙總管道，對數值欄位執行數學操作。

參數

- number：將傳回絕對值的數值表達式。

範例 (MongoDB Shell)

此範例示範 $abs運算子的使用情況，以尋找數值欄位的絕對值。

建立範例文件

```
db.numbers.insertMany([
  { "_id": 1, "value": -5 },
```

```
  { "_id": 2, "value": 10 },
  { "_id": 3, "value": -3.14 },
  { "_id": 4, "value": 0 }
]);
```

查詢範例

```
db.numbers.aggregate([
  { $project: {
    "_id": 1,
    "absolute_value": { $abs: "$value" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "absolute_value": 5 },
  { "_id": 2, "absolute_value": 10 },
  { "_id": 3, "absolute_value": 3.14 },
  { "_id": 4, "absolute_value": 0 }
]
```

程式碼範例

若要檢視使用 $abs命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $project: {
      "_id": 1,
      "absolute_value": { $abs: "$value" }
    }}
```

```
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['numbers']

    result = list(collection.aggregate([
        { '$project': {
            "_id": 1,
            "absolute_value": { "$abs": "$value" }
        }}
    ]))

    print(result)
    client.close()

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

$acos

8.0.1 版的新功能。

Amazon DocumentDB 中的$acos運算子會傳回值的 arccosine （反餘弦）。輸入值必須介於 -1 和 1 之間。

參數

- expression：解析為介於 -1 和 1 之間的數字的表達式。

結果以弧度表示。若要取得度數，請$radiansToDegrees套用至輸出。

傳回類型預設為 double 。如果輸入是 128 位元的小數位數，則輸出也是 128 位元的小數位數。

範例 (MongoDB Shell)

下列範例示範如何使用 $acos運算子來計算數值的餘弦。

建立範例文件

```
db.values.insertMany([
  { "_id": 1, "value": 1 },
  { "_id": 2, "value": 0 },
  { "_id": 3, "value": -1 }
]);
```

查詢範例

```
db.values.aggregate([
  { $project: {
    "angle": { $acos: "$value" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 1.5707963267948966 },
  { "_id": 3, "angle": 3.141592653589793 }
]
```

程式碼範例

若要檢視使用 $acos 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "angle": { $acos: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'angle': { '$acos': '$value' }
        }}
    ]))

    print(result)
    client.close()

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

$acosh

8.0.1 版的新功能。

Amazon DocumentDB 中的$acosh運算子會傳回值的反雙曲餘弦 （雙曲餘弦）。輸入值必須大於或等於 1。

參數

- expression：解析為大於或等於 1 的數字的表達式。

傳回類型預設為 double 。如果輸入是 128 位元的小數位數，則輸出也是 128 位元的小數位數。

範例 (MongoDB Shell)

下列範例示範如何使用 $acosh運算子來計算數值的反雙曲餘弦。

建立範例文件

```
db.values.insertMany([
  { "_id": 1, "value": 1 },
  { "_id": 2, "value": 2 },
  { "_id": 3, "value": 10 }
]);
```

查詢範例

```
db.values.aggregate([
  { $project: {
    "result": { $acosh: "$value" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "result": 0 },
  { "_id": 2, "result": 1.3169578969248166 },
  { "_id": 3, "result": 2.993222846126381 }
]
```

程式碼範例

若要檢視使用 $acosh 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "result": { $acosh: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'result': { '$acosh': '$value' }
        }}
    ]))
```

```
    print(result)
    client.close()

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

$add

Amazon DocumentDB 中的$add運算子用來同時新增數字或日期。它可用於對數值欄位執行算術運算，或將時間單位數新增至日期欄位，以執行日期算術運算。

參數

- `expression1`：要新增的第一個數字或日期。
- `expression2`：要新增的第二個數字或日期。
- `expression3`：（選用）要新增的其他號碼或日期。

範例 (MongoDB Shell)

下列範例示範如何使用 $add運算子將兩個數字加在一起。

建立範例文件

```
db.numbers.insertMany([
  { _id: 1, a: 5, b: 3 },
  { _id: 2, a: 10, b: 7 },
  { _id: 3, a: 2, b: 8 }
]);
```

查詢範例

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    sum: { $add: ["$a", "$b"] }
  }}
])
```

輸出

```
[
  { "_id" : 1, "sum" : 8 },
  { "_id" : 2, "sum" : 17 },
  { "_id" : 3, "sum" : 10 }
]
```

在此範例中，$add運算子用於為每個文件新增 a和 b 欄位的值，並將結果存放在 sum 欄位中。

程式碼範例

若要檢視使用 $add命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $project: {
      _id: 1,
      sum: { $add: ['$a', '$b'] }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
db = client['test']
collection = db['numbers']

result = list(collection.aggregate([
    { '$project': {
        '_id': 1,
        'sum': { '$add': ['$a', '$b'] }
    }}
]))

print(result)
client.close()
```

$addToSet

$addToSet 彙總運算子會從每個群組的指定表達式傳回唯一值的陣列。它在$group階段內用來累積不同的值，自動消除重複項目。

參數

- expression：要評估群組中每個文件的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $addToSet 運算子來收集為每位客戶下訂單的唯一城市。

建立範例文件

```
db.orders.insertMany([
  { _id: 1, customer: "Alice", city: "Seattle", amount: 100 },
  { _id: 2, customer: "Alice", city: "Portland", amount: 150 },
  { _id: 3, customer: "Bob", city: "Seattle", amount: 200 },
  { _id: 4, customer: "Alice", city: "Seattle", amount: 75 },
  { _id: 5, customer: "Bob", city: "Boston", amount: 300 }
]);
```

查詢範例

```
db.orders.aggregate([
  {
    $group: {
```

```
      _id: "$customer",
      cities: { $addToSet: "$city" }
    }
  }
]);
```

輸出

```
[
  { _id: 'Bob', cities: [ 'Seattle', 'Boston' ] },
  { _id: 'Alice', cities: [ 'Seattle', 'Portland' ] }
]
```

程式碼範例

若要檢視使用$addToSet彙總運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('orders');

  const result = await collection.aggregate([
    {
      $group: {
        _id: "$customer",
        cities: { $addToSet: "$city" }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['orders']

    result = list(collection.aggregate([
        {
            '$group': {
                '_id': '$customer',
                'cities': { '$addToSet': '$city' }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$addFields

Amazon DocumentDB 彙總管道中的$addFields階段可讓您將新的運算欄位新增至文件。這對於將衍生或轉換的資料新增至文件非常有用。

參數

- newField：要新增的新欄位名稱。
- expression：解析為新欄位值的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $addFields 新增 欄位TotalInventory，以根據 Inventory.OnHand和 Inventory.OrderQnty 欄位計算總庫存。

建立範例文件

```
db.example.insertMany([
  {
    "Item": "Spray Paint",
    "Colors": ["Black", "Red", "Green", "Blue"],
    "Inventory": {
      "OnHand": 47,
      "MinOnHand": 50,
      "OrderQnty": 36
    },
    "UnitPrice": 3.99
  },
  {
    "Item": "Ruler",
    "Colors": ["Red", "Green", "Blue", "Clear", "Yellow"],
    "Inventory": {
      "OnHand": 47,
      "MinOnHand": 40
    },
    "UnitPrice": 0.89
  }
]);
```

查詢範例

```
db.example.aggregate([
  {
    $addFields: {
      TotalInventory: { $add: ["$Inventory.OnHand", "$Inventory.OrderQnty"] }
    }
  }
])
```

輸出

```
[
  {
    "_id" : ObjectId("5bedafbcf65ff161707de24f"),
    "Item" : "Ruler",
    "Colors" : [ "Red", "Green", "Blue", "Clear", "Yellow" ],
    "Inventory" : {
      "OnHand" : 47,
      "MinOnHand" : 40
```

```
    },
    "UnitPrice" : 0.89,
    "TotalInventory" : 47
  },
  {
    "_id" : ObjectId("5bedafbcf65ff161707de250"),
    "Item" : "Spray Paint",
    "Colors" : [ "Black", "Red", "Green", "Blue" ],
    "Inventory" : {
      "OnHand" : 47,
      "MinOnHand" : 50,
      "OrderQnty" : 36
    },
    "UnitPrice" : 3.99,
    "TotalInventory" : 83
  }
]
```

程式碼範例

若要檢視使用 $addFields命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
const db = client.db('test');
const collection = db.collection('example');

const result = await collection.aggregate([
  {
    $addFields: {
      TotalInventory: { $add: ['$Inventory.OnHand', '$Inventory.OrderQnty'] }
    }
  }
]).toArray();

console.log(result);

await client.close();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['example']

result = list(collection.aggregate([
    {
        '$addFields': {
            'TotalInventory': { '$add': ['$Inventory.OnHand',
 '$Inventory.OrderQnty'] }
        }
    }
]))

print(result)
client.close()
```

$allElementsTrue

4.0 版的新功能

`$allElementsTrue` 運算子用於檢查陣列中的所有元素是否評估為 true 值。

參數

- `expression`：評估為陣列的表達式。

範例 (MongoDB Shell)

下列範例示範使用 `$allElementsTrue`來檢查陣列中的所有元素是否為 true。

建立範例文件

```
db.collection.insert([
  { "name": "John", "scores": [100, 90, 80] },
  { "name": "Jane", "scores": [80, 85, 0] },
```

```
  { "name": "Bob", "scores": [90, 95, null] }
])
```

查詢範例

```
db.collection.find({
  "scores": { "$allElementsTrue": [{ "$gt": 0 }] }
})
```

輸出

```
[
  { "_id" : ObjectId("..."), "name" : "John", "scores" : [ 100, 90, 80 ] },
  { "_id" : ObjectId("..."), "name" : "Bob", "scores" : [ 90, 95, null ] }
]
```

在此範例中，查詢會檢查scores陣列中的所有元素是否大於 0。"name": "Jane" 排除具有 的文件，因為scores陣列包含 0，這是偽值。

程式碼範例

若要檢視使用 $allElementsTrue命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.find({
    "scores": { "$allElementsTrue": [{ "$gt": 0 }] }
  }).toArray();

  console.log(result);

  await client.close();
}
```

```
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['collection']

    result = list(collection.find({
        "scores": {"$allElementsTrue": [{"$gt": 0}]}
    }))

    print(result)

    client.close()

example()
```

$and

$and 彙總運算子會評估多個表達式，並true僅在所有表達式評估為 時傳回 true。如果任何表達式為 false，則會傳回 false。

參數

- expressions：要評估的表達式陣列。

範例 (MongoDB Shell)

下列範例示範如何使用 $and 運算子來檢查產品是否符合多個條件。

建立範例文件

```
db.products.insertMany([
  { _id: 1, name: "Laptop", price: 1200, inStock: true },
```

```
  { _id: 2, name: "Mouse", price: 25, inStock: false },
  { _id: 3, name: "Keyboard", price: 75, inStock: true }
]);
```

查詢範例

```
db.products.aggregate([
  {
    $project: {
      name: 1,
      price: 1,
      inStock: 1,
      affordable: {
        $and: [
          { $lt: ["$price", 100] },
          { $eq: ["$inStock", true] }
        ]
      }
    }
  }
]);
```

輸出

```
[
  { _id: 1, name: 'Laptop', price: 1200, inStock: true, affordable: false },
  { _id: 2, name: 'Mouse', price: 25, inStock: false, affordable: false },
  { _id: 3, name: 'Keyboard', price: 75, inStock: true, affordable: true }
]
```

程式碼範例

若要檢視使用$and彙總運算子的程式碼範例，請選擇您要使用之語言的標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
```

```
  const db = client.db('test');
  const collection = db.collection('products');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
        price: 1,
        inStock: 1,
        affordable: {
          $and: [
            { $lt: ["$price", 100] },
            { $eq: ["$inStock", true] }
          ]
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['products']

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'price': 1,
                'inStock': 1,
                'affordable': {
```

```
                    '$and': [
                        { '$lt': ['$price', 100] },
                        { '$eq': ['$inStock', True] }
                    ]
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$anyElementTrue

4.0 版的新功能

`$anyElementTrue` 運算子用於判斷陣列中的任何元素是否為 true。

參數

- `field`：要評估的陣列欄位。

範例 (MongoDB Shell)

下列範例示範 的使用方式，`$anyElementTrue`以檢查陣列中的任何元素是否為 true。

建立範例文件

```
db.grades.insertMany([
  { _id: 1, student: 'Tim', scores: [false, false, null] },
  { _id: 2, student: 'Bob', scores: [false, 0, false] },
  { _id: 3, student: 'Ivy', scores: [false, true, 0] }
])
```

查詢範例

```
db.grades.aggregate([
  {
```

```
    $project: {
      student: 1,
      isAnyTrue: { $anyElementTrue: ["$scores"] },
      _id: 0
    }
  }
])
```

輸出

```
[
  { student: 'Tim', isAnyTrue: false },
  { student: 'Bob', isAnyTrue: false },
  { student: 'Ivy', isAnyTrue: true }
]
```

程式碼範例

若要檢視使用 $anyElementTrue命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('grades');

  const result = await collection.aggregate([
    {
      $project: {
        student: 1,
        isAnyTrue: { $anyElementTrue: ["$scores"] },
        _id: 0
      }
    }
  ]).toArray();

  console.log(result);
```

```
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['grades']

    result = list(collection.aggregate([
      {
        "$project": {
          "student": 1,
          "isAnyTrue": { "$anyElementTrue": ["$scores"] },
          "_id": 0
        }
      }
    ]))

    print(result)

    client.close()

example()
```

$arrayElemAt

Amazon DocumentDB 中的$arrayElemAt運算子可讓您依索引位置從陣列擷取 元素。當您需要存取文件中陣列欄位中的特定元素時，此功能特別有用。

參數

- array：要從中擷取 元素的輸入陣列。
- index：要擷取之 元素的以零為基礎的索引位置。此值必須是非負整數。

範例 (MongoDB Shell)

在此範例中，我們將示範如何使用 $arrayElemAt運算子，從miles集合中的flight_miles陣列擷取特定元素。

建立範例文件

```
db.miles.insertMany([
  { "_id" : 1, "member_since" : ISODate("1987-01-01T00:00:00Z"), "credit_card" : false,
 "flight_miles" : [ 1205, 2560, 880 ]},
  { "_id" : 2, "member_since" : ISODate("1982-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 1205, 2560, 890, 2780 ]},
  { "_id" : 3, "member_since" : ISODate("1999-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 1205, 880 ]}
]);
```

查詢範例

```
db.miles.aggregate([
  { $project: {
    "_id": 1,
    "first_mile": { $arrayElemAt: [ "$flight_miles", 0 ] },
    "last_mile": { $arrayElemAt: [ "$flight_miles", -1 ] }
  }}
]);
```

輸出

```
{ "_id" : 1, "first_mile" : 1205, "last_mile" : 880 }
{ "_id" : 2, "first_mile" : 1205, "last_mile" : 2780 }
{ "_id" : 3, "first_mile" : 1205, "last_mile" : 880 }
```

在此範例中，我們使用 $arrayElemAt運算子來擷取每個文件flight_miles陣列的第一個和最後一個元素。

程式碼範例

若要檢視使用 $arrayElemAt命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('miles');

  const result = await collection.aggregate([
    { $project: {
      "_id": 1,
      "first_mile": { $arrayElemAt: [ "$flight_miles", 0 ] },
      "last_mile": { $arrayElemAt: [ "$flight_miles", -1 ] }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.mydatabase
    collection = db.miles

    result = list(collection.aggregate([
        { "$project": {
            "_id": 1,
            "first_mile": { "$arrayElemAt": [ "$flight_miles", 0 ] },
            "last_mile": { "$arrayElemAt": [ "$flight_miles", -1 ] }
        }}
    ]))

    print(result)
    client.close()
```

```
example()
```

$arrayToObject

Amazon DocumentDB 中的$arrayToObject運算子是$objectToArray運算子的反向。它需要一系列鍵值對文件，並將其轉換為單一文件。當您需要將一組鍵/值對轉換回物件或文件結構時，這特別有用。

參數

- `array expression`：解析為陣列的表達式。陣列元素必須是具有兩個欄位的文件：k（索引鍵）和 v（值）。

範例 (MongoDB Shell)

下列範例示範如何使用 $arrayToObject將鍵值對陣列轉換回文件。

建立範例文件

```
db.videos.insertMany([
  { "_id": 1, "name": "Live Soft", "inventory": { "Des Moines": 1000, "Ames": 500 } },
  { "_id": 2, "name": "Top Pilot", "inventory": { "Mason City": 250, "Des Moines":
 1000 } },
  { "_id": 3, "name": "Romancing the Rock", "inventory": { "Mason City": 250, "Ames":
 500 } },
  { "_id": 4, "name": "Bravemind", "inventory": { "Mason City": 250, "Des Moines":
 1000, "Ames": 500 } }
]);
```

查詢範例

```
db.videos.aggregate([
  { $project: { name: 1, videos: { $objectToArray: "$inventory" } } },
  { $project: { name: 1, inventory: { $arrayToObject: "$videos" } } }
]);
```

輸出

```
{ "_id" : 1, "name" : "Live Soft", "inventory" : { "Des Moines" : 1000, "Ames" :
 500 } }
{ "_id" : 2, "name" : "Top Pilot", "inventory" : { "Mason City" : 250, "Des Moines" :
 1000 } }
{ "_id" : 3, "name" : "Romancing the Rock", "inventory" : { "Mason City" : 250,
 "Ames" : 500 } }
{ "_id" : 4, "name" : "Bravemind", "inventory" : { "Mason City" : 250, "Des Moines" :
 1000, "Ames" : 500 } }
```

在此範例中，運算$objectToArray子用於將inventory物件轉換為金鑰/值對的陣列。然後，運算$arrayToObject子會用來將陣列轉換回文件，還原原始物件結構。

程式碼範例

若要檢視使用 $arrayToObject命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('videos');

  const result = await collection.aggregate([
    { $project: { name: 1, videos: { $objectToArray: '$inventory' } } },
    { $project: { name: 1, inventory: { $arrayToObject: '$videos' } } }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient
```

```
def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['videos']

    result = list(collection.aggregate([
        { '$project': { 'name': 1, 'videos': { '$objectToArray': '$inventory' } } },
        { '$project': { 'name': 1, 'inventory': { '$arrayToObject': '$videos' } } }
    ]))

    print(result)
    client.close()

example()
```

$asin

8.0.1 版的新功能。

Amazon DocumentDB 中的$asin運算子會傳回值的餘弦 （反正弦）。輸入值必須介於 -1 和 1 之間。

參數

- expression：解析為介於 -1 和 1 之間的數字的表達式。

結果以弧度表示。若要取得度數，請$radiansToDegrees套用至輸出。

傳回類型預設為 double 。如果輸入是 128 位元的小數位數，則輸出也是 128 位元的小數位數。

範例 (MongoDB Shell)

下列範例示範如何使用 $asin運算子來計算數值的餘弦。

建立範例文件

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 0.5 },
  { "_id": 3, "value": 1 }
]);
```

查詢範例

```
db.values.aggregate([
  { $project: {
    "angle": { $asin: "$value" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 0.5235987755982989 },
  { "_id": 3, "angle": 1.5707963267948966 }
]
```

程式碼範例

若要檢視使用 $asin 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "angle": { $asin: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'angle': { '$asin': '$value' }
        }}
    ]))

    print(result)
    client.close()

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

$asinh

8.0.1 版的新功能。

Amazon DocumentDB 中的$asinh運算子會傳回值的反雙曲正弦。在彙總管道中使用它，對數值欄位執行雙曲三角計算。

參數

- expression：解析為數字的表達式。

結果以弧度表示。若要取得度數，請$radiansToDegrees套用至輸出。

傳回類型預設為 double 。如果輸入是 128 位元的小數位數，則輸出也是 128 位元的小數位數。

範例 (MongoDB Shell)

下列範例示範如何使用 $asinh運算子來計算數值的反雙曲正弦值。

建立範例文件

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 1 },
  { "_id": 3, "value": -1 }
]);
```

查詢範例

```
db.values.aggregate([
  { $project: {
    "result": { $asinh: "$value" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "result": 0 },
  { "_id": 2, "result": 0.881373587019543 },
  { "_id": 3, "result": -0.881373587019543 }
]
```

程式碼範例

若要檢視使用 $asinh 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "result": { $asinh: "$value" }
```

```
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'result': { '$asinh': '$value' }
        }}
    ]))

    print(result)
    client.close()

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

$atan

8.0.1 版的新功能。

Amazon DocumentDB 中的$atan運算子會傳回值的反正切 (arc 正切）。

輸入表達式必須解析為數值。

結果以弧度表示。若要取得度數，請$radiansToDegrees套用至輸出。

傳回類型預設為 double 。如果輸入是 128 位元的小數位數，則輸出也是 128 位元的小數位數。

參數

- `expression`：解析為數字的表達式。

Behavior (行為)

Null 和 NaN

範例	結果
`{ $atan: NaN }`	`NaN`
`{ $atan: null }`	`null`

當輸入為 `null`或參考欄位遺失時，結果為 `null`。的輸入`NaN`會產生 `NaN`。

範例 (MongoDB Shell)

下列範例示範如何使用 `$atan`運算子來計算值的正切。

建立範例文件

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 1 },
  { "_id": 3, "value": -1 }
]);
```

查詢範例

```
db.values.aggregate([
  { $project: {
    "angle": { $atan: "$value" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "angle": 0 },
```

```
  { "_id": 2, "angle": 0.7853981633974483 },
  { "_id": 3, "angle": -0.7853981633974483 }
]
```

程式碼範例

若要檢視使用 $atan 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "angle": { $atan: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
```

```
        { '$project': {
            'angle': { '$atan': '$value' }
        }}
    ]))

    print(result)
    client.close()

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

$atan2

8.0.1 版的新功能。

Amazon DocumentDB 中的$atan2運算子會傳回 的反正切 (arc 正切）y / x，其中 y和 分別x是傳遞給表達式的第一和第二個值。

兩個表達式都必須解析為數值。運算子使用兩個引數的符號來判斷正確的 1/4。

結果以弧度表示。若要取得度數，請$radiansToDegrees套用至輸出。

傳回類型預設為 double 。如果輸入是 128 位元的小數位數，則輸出也是 128 位元的小數位數。

參數

- expression 1 (y)：解析為代表 y 座標之數字的表達式。
- expression 2 (x)：解析為代表 x 座標之數字的表達式。

Behavior (行為)

Null 和 NaN

如果其中一個引數是 null（或參考缺少的欄位），則結果為 null。如果任一引數為 NaN，則結果為 NaN。當一個引數為 null，另一個引數為 時NaN， null 優先。

範例	結果
{ $atan2: [NaN, <value>] }	NaN
{ $atan2: [<value>, NaN] }	NaN

範例	結果
`{ $atan2: [ null, <value> ] }`	null
`{ $atan2: [ <value>, null ] }`	null
`{ $atan2: [ NaN, null ] }`	null
`{ $atan2: [ null, NaN ] }`	null

範例 (MongoDB Shell)

下列範例顯示如何使用 $atan2 運算子來計算一組點的 y/x 正切。

建立範例文件

```
db.points.insertMany([
  { "_id": 1, "y": 1, "x": 1 },
  { "_id": 2, "y": 1, "x": 0 },
  { "_id": 3, "y": -1, "x": -1 }
]);
```

查詢範例

```
db.points.aggregate([
  { $project: {
    "angle": { $atan2: ["$y", "$x"] }
  }}
]);
```

輸出

```
[
  { "_id": 1, "angle": 0.7853981633974483 },
  { "_id": 2, "angle": 1.5707963267948966 },
  { "_id": 3, "angle": -2.356194490192345 }
]
```

程式碼範例

若要檢視使用 $atan2 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('points');

  const result = await collection.aggregate([
    { $project: {
      "angle": { $atan2: ["$y", "$x"] }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['points']

    result = list(collection.aggregate([
        { '$project': {
            'angle': { '$atan2': ['$y', '$x'] }
        }}
    ]))

    print(result)
    client.close()
```

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

$atanh

8.0.1 版的新功能。

Amazon DocumentDB 中的$atanh運算子會傳回值的反雙曲正切（雙曲電弧正切）。

輸入表達式必須解析為 -11（包含）範圍內的數字。超出此範圍的值會導致錯誤。

傳回類型預設為 double 。如果輸入是 128 位元的小數位數，則輸出也是 128 位元的小數位數。

參數

- expression：解析為介於 -1和 之間的數字的表達式1，包含 。

Behavior (行為)

null、NaN 和 +/- Infinity

範例	結果
{ $atanh: NaN }	NaN
{ $atanh: null }	null
{ $atanh: 1 }	Infinity
{ $atanh: -1 }	-Infinity
{ $atanh: Infinity } 或 { $atanh: -Infinity }	擲回錯誤。

當輸入為 null或參考欄位遺失時，結果為 null。的輸入NaN會產生 NaN。邊界值 1和 -Infinity分別-1產生 Infinity和 。超出[-1, 1]範圍（包括 ±Infinity) 的任何值都會導致錯誤。

範例 (MongoDB Shell)

下列範例顯示如何使用 $atanh運算子來計算值的反雙曲切線。

建立範例文件

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 0.5 },
  { "_id": 3, "value": -0.5 }
]);
```

查詢範例

```
db.values.aggregate([
  { $project: {
    "result": { $atanh: "$value" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "result": 0 },
  { "_id": 2, "result": 0.5493061443340548 },
  { "_id": 3, "result": -0.5493061443340548 }
]
```

程式碼範例

若要檢視使用 $atanh 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
```

```
      "result": { $atanh: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'result': { '$atanh': '$value' }
        }}
    ]))

    print(result)
    client.close()

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

$avg

Amazon DocumentDB $avg 中的彙總運算子會計算輸入階段的文件中指定表達式的平均值。此運算子有助於計算一組文件間數值欄位或表達式的平均值。

參數

- expression：用來計算平均值的表達式。這可以是欄位路徑（例如 "$field"）或表達式（例如 { $multiply: ["$field1", "$field2"] }）。

範例 (MongoDB Shell)

下列範例示範如何使用 $avg 運算子來計算一組學生文件的平均分數。

建立範例文件

```
db.students.insertMany([
  { name: "John", score: 85 },
  { name: "Jane", score: 92 },
  { name: "Bob", score: 78 },
  { name: "Alice", score: 90 }
]);
```

查詢範例

```
db.students.aggregate([
  { $group: {
    _id: null,
    avgScore: { $avg: "$score" }
  }}
]);
```

輸出

```
[
  {
    "_id": null,
    "avgScore": 86.25
  }
]
```

程式碼範例

若要檢視使用 $avg命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function calculateAvgScore() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
```

```
  const db = client.db('test');
  const result = await db.collection('students').aggregate([
    { $group: {
      _id: null,
      avgScore: { $avg: '$score' }
    }}
  ]).toArray();
  console.log(result);
  await client.close();
}

calculateAvgScore();
```

Python

```
from pymongo import MongoClient

def calculate_avg_score():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    result = list(db.students.aggregate([
        { '$group': {
            '_id': None,
            'avgScore': { '$avg': '$score' }
        }}
    ]))
    print(result)
    client.close()

calculate_avg_score()
```

$binarySize

8.0.1 版的新功能。

Amazon DocumentDB 中的$binarySize運算子會傳回指定字串或二進位資料值的位元組大小。對於字串值，這是 UTF-8 編碼的位元組計數，而不是字元計數。多位元組 UTF-8 字元（例如重音字元或 CJK 字元）會產生大於字串中字元數的位元組計數。

參數

- expression：解析為字串或二進位資料值的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $binarySize運算子傳回字串欄位的位元組大小。

建立範例文件

```
db.docs.insertMany([
  {_id: 1, content: "hello"},
  {_id: 2, content: "Amazon DocumentDB"},
  {_id: 3, content: ""}
]);
```

查詢範例

```
db.docs.aggregate([
  { $project: { size: { $binarySize: "$content" } } }
]);
```

輸出

```
[
  {_id: 1, size: 5},
  {_id: 2, size: 17},
  {_id: 3, size: 0}
]
```

程式碼範例

若要檢視使用 $binarySize 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
```

```
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('docs');
    const result = await collection.aggregate([
      { $project: { size: { $binarySize: "$content" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['docs']
        result = list(collection.aggregate([
            {'$project': {'size': {'$binarySize': '$content'}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$bitAnd

8.0.1 版的新功能。

Amazon DocumentDB 中的$bitAnd運算子會對整數或長值執行位元 AND 操作。

參數

- expressions：兩個或多個表達式的陣列，可解析為整數或長。

範例 (MongoDB Shell)

下列範例顯示如何使用 $bitAnd 運算子對兩個欄位執行位元 AND。

建立範例文件

```
db.flags.insertMany([
  {_id: 1, a: 13, b: 10},
  {_id: 2, a: 7, b: 5},
  {_id: 3, a: 15, b: 9}
]);
```

查詢範例

```
db.flags.aggregate([
  { $project: { result: { $bitAnd: ["$a", "$b"] } } }
]);
```

輸出

```
[
  {_id: 1, result: 8},
  {_id: 2, result: 5},
  {_id: 3, result: 9}
]
```

在二進位中：13 (1101) 和 10 (1010) = 8 (1000)；7 (0111) 和 5 (0101) = 5 (0101)；15 (1111) 和 9 (1001) = 9 (1001)。

程式碼範例

若要檢視使用 $bitAnd 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
```

```
    const db = client.db('test');
    const collection = db.collection('flags');
    const result = await collection.aggregate([
      { $project: { result: { $bitAnd: ["$a", "$b"] } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['flags']
        result = list(collection.aggregate([
            {'$project': {'result': {'$bitAnd': ['$a', '$b']}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$bitNot

8.0.1 版的新功能。

Amazon DocumentDB 中的$bitNot運算子會對整數或長值執行位元 NOT 操作，並傳回位元補數。對於指定的整數或長 n，結果為 -(n+1)。

參數

- `expression`：解析為整數或長的表達式。傳回指定整數或長 n 的位元補數，即 -(n+1)。

範例 (MongoDB Shell)

下列範例顯示如何使用 $bitNot 運算子運算整數值的位元補數。

建立範例文件

```
db.numbers.insertMany([
  {_id: 1, value: 0},
  {_id: 2, value: 5},
  {_id: 3, value: -3}
]);
```

查詢範例

```
db.numbers.aggregate([
  { $project: { result: { $bitNot: "$value" } } }
]);
```

輸出

```
[
  {_id: 1, result: -1},
  {_id: 2, result: -6},
  {_id: 3, result: 2}
]
```

在二進位：NOT 0 = -1；NOT 5 = -6；NOT -3 = 2。

程式碼範例

若要檢視使用 $bitNot 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
```

```
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('numbers');
    const result = await collection.aggregate([
      { $project: { result: { $bitNot: "$value" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['numbers']
        result = list(collection.aggregate([
            {'$project': {'result': {'$bitNot': '$value'}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$bitOr

8.0.1 版的新功能。

Amazon DocumentDB 中的$bitOr運算子會對整數或長值執行位元 OR 操作。

參數

- expressions：兩個或多個表達式的陣列，可解析為整數或長。

範例 (MongoDB Shell)

下列範例示範如何使用 $bitOr 運算子在兩個欄位上執行位元 OR。

建立範例文件

```
db.flags.insertMany([
  {_id: 1, a: 13, b: 10},
  {_id: 2, a: 7, b: 5},
  {_id: 3, a: 15, b: 9}
]);
```

查詢範例

```
db.flags.aggregate([
  { $project: { result: { $bitOr: ["$a", "$b"] } } }
]);
```

輸出

```
[
  {_id: 1, result: 15},
  {_id: 2, result: 7},
  {_id: 3, result: 15}
]
```

在二進位中：13 (1101) 或 10 (1010) = 15 (1111)；7 (0111) 或 5 (0101) = 7 (0111)；15 (1111) 或 9 (1001) = 15 (1111)。

程式碼範例

若要檢視使用 $bitOr 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
```

```
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('flags');
    const result = await collection.aggregate([
      { $project: { result: { $bitOr: ["$a", "$b"] } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['flags']
        result = list(collection.aggregate([
            {'$project': {'result': {'$bitOr': ['$a', '$b']}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$bitXor

8.0.1 版的新功能。

Amazon DocumentDB 中的`$bitXor`運算子會對整數或長值執行位元 XOR 操作。

參數

- `expressions`：兩個或多個表達式的陣列，可解析為整數或長。

範例 (MongoDB Shell)

下列範例顯示如何使用 $bitXor 運算子對兩個欄位執行位元 XOR。

建立範例文件

```
db.flags.insertMany([
  {_id: 1, a: 13, b: 10},
  {_id: 2, a: 7, b: 5},
  {_id: 3, a: 15, b: 9}
]);
```

查詢範例

```
db.flags.aggregate([
  { $project: { result: { $bitXor: ["$a", "$b"] } } }
]);
```

輸出

```
[
  {_id: 1, result: 7},
  {_id: 2, result: 2},
  {_id: 3, result: 6}
]
```

在二進位中：13 (1101) XOR 10 (1010) = 7 (0111)；7 (0111) XOR 5 (0101) = 2 (0010)；15 (1111) XOR 9 (1001) = 6 (0110)。

程式碼範例

若要檢視使用 $bitXor 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
```

```
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('flags');
    const result = await collection.aggregate([
      { $project: { result: { $bitXor: ["$a", "$b"] } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['flags']
        result = list(collection.aggregate([
            {'$project': {'result': {'$bitXor': ['$a', '$b']}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$底部

8.0.1 版的新功能。

使用$group階段中的$bottom累積器，根據指定的排序順序傳回每個群組的最低排名文件。

參數

- sortBy：指定排序順序的文件。1 使用 遞增或 -1 遞減。

- output：指定要從底部文件傳回之欄位的表達式。

範例 (MongoDB Shell)

下列範例顯示如何使用$bottom累積器來尋找銷售集合中每個項目的最低銷售 （最低數量）。

建立範例文件

```
db.sales.insertMany([
  { item: "abc", quantity: 10, price: 5 },
  { item: "abc", quantity: 5, price: 8 },
  { item: "xyz", quantity: 15, price: 3 },
  { item: "xyz", quantity: 7, price: 6 }
])
```

查詢範例

```
db.sales.aggregate([
  { $group: { _id: "$item", bottomSale: { $bottom: { sortBy: { quantity: -1 }, output:
 { quantity: "$quantity", price: "$price" } } } } }
])
```

輸出

```
[
  { "_id": "xyz", "bottomSale": { "quantity": 7, "price": 6 } },
  { "_id": "abc", "bottomSale": { "quantity": 5, "price": 8 } }
]
```

程式碼範例

若要檢視使用 $bottom 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
```

```
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const db = client.db('test');
    const collection = db.collection('sales');

    const result = await collection.aggregate([
      { $group: { _id: "$item", bottomSale: { $bottom: { sortBy: { quantity: -1 },
 output: { quantity: "$quantity", price: "$price" } } } } }
    ]).toArray();

    console.log(result);

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client['test']
        collection = db['sales']

        result = list(collection.aggregate([
            { '$group': { '_id': '$item', 'bottomSale': { '$bottom':
 { 'sortBy': { 'quantity': -1 }, 'output': { 'quantity': '$quantity', 'price':
 '$price' } } } } }
```

```
        ]))

        pprint(result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$bottomN

8.0.1 版的新功能。

使用$group階段中的$bottomN累積器，根據指定的排序順序傳回群組中的底部 N 元素。如果群組包含少於 N 個元素，則 會$bottomN傳回群組中的所有元素。

參數

- n：正整數或解析為一個的表達式，指定每個群組要傳回的底部結果數量。
- sortBy：指定排序順序的文件。1 使用 遞增或 -1 遞減。
- output：指定要從每個底部 N 文件傳回的欄位的表達式。

範例 (MongoDB Shell)

下列範例顯示如何使用$bottomN累積器來尋找銷售集合中每個項目的最低 2 個銷售額 （最低數量）。

建立範例文件

```
db.sales.insertMany([
  { item: "abc", quantity: 10, price: 5 },
  { item: "abc", quantity: 7, price: 8 },
  { item: "abc", quantity: 5, price: 10 },
  { item: "xyz", quantity: 15, price: 3 },
  { item: "xyz", quantity: 9, price: 6 },
  { item: "xyz", quantity: 3, price: 12 }
```

```
])
```

查詢範例

```
db.sales.aggregate([
  { $group: { _id: "$item", bottomTwoSales: { $bottomN: { n: 2, sortBy: { quantity:
 -1 }, output: { quantity: "$quantity", price: "$price" } } } } }
])
```

輸出

```
[
  { "_id": "xyz", "bottomTwoSales": [{ "quantity": 9, "price": 6 }, { "quantity": 3,
 "price": 12 }] },
  { "_id": "abc", "bottomTwoSales": [{ "quantity": 7, "price": 8 }, { "quantity": 5,
 "price": 10 }] }
]
```

程式碼範例

若要檢視使用 $bottomN 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const db = client.db('test');
    const collection = db.collection('sales');

    const result = await collection.aggregate([
      { $group: { _id: "$item", bottomTwoSales: { $bottomN: { n: 2, sortBy:
 { quantity: -1 }, output: { quantity: "$quantity", price: "$price" } } } } }
    ]).toArray();
```

```
    console.log(result);

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client['test']
        collection = db['sales']

        result = list(collection.aggregate([
            { '$group': { '_id': '$item', 'bottomTwoSales': { '$bottomN': { 'n':
 2, 'sortBy': { 'quantity': -1 }, 'output': { 'quantity': '$quantity', 'price':
 '$price' } } } } }
        ]))

        pprint(result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$bsonSize

8.0.1 版的新功能。

Amazon DocumentDB 中的$bsonSize運算子會在編碼為 BSON 時傳回文件的位元組大小。

參數

- `expression`：解析為文件或物件的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $bsonSize運算子傳回每個文件的 BSON 大小。

建立範例文件

```
db.items.insertMany([
  {_id: 1, name: "a", tags: ["x"]},
  {_id: 2, name: "ab", tags: ["x", "y", "z"]},
  {_id: 3, name: "abc", tags: []}
]);
```

查詢範例

```
db.items.aggregate([
  { $project: { docSize: { $bsonSize: "$$ROOT" } } }
]);
```

輸出

```
[
  {_id: 1, docSize: 46},
  {_id: 2, docSize: 65},
  {_id: 3, docSize: 39}
]
```

> **Note**
>
> 確切大小取決於文件欄位的 BSON 編碼。

程式碼範例

若要檢視使用 $bsonSize 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('items');
    const result = await collection.aggregate([
      { $project: { docSize: { $bsonSize: "$$ROOT" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['items']
        result = list(collection.aggregate([
            {'$project': {'docSize': {'$bsonSize': '$$ROOT'}}}
        ]))
        print(result)
    finally:
        client.close()
```

```
example()
```

$bucket

8.0 版的新功能

Elastic 叢集不支援。

Amazon DocumentDB $bucket 中的彙總階段可讓您根據指定的表達式和儲存貯體界限，將輸入文件分組至儲存貯體。這對於分析屬於特定值範圍或類別的資料非常有用。

參數

- groupBy （必要）：指定要分組之值的表達式。
- boundaries （必要）：定義儲存貯體邊界的雙值陣列。文件會根據落在指定邊界內的groupBy表達式值指派給儲存貯體。
- default （選用）：為groupBy表達式值未落在任何指定邊界的文件輸出的常值。
- output （選用）：指定要為每個儲存貯體輸出資訊的物件。您可以使用累積器運算子，例如 $sum、$min、 $avg和 $max來計算每個儲存貯體的彙總。

範例 (MongoDB Shell)

下列範例示範如何使用 $bucket階段，依價格範圍將銷售資料分組。

建立範例文件

```
db.sales.insertMany([
  { item: "abc", price: 10, quantity: 2, date: new Date("2020-09-01") },
  { item: "def", price: 20, quantity: 1, date: new Date("2020-10-01") },
  { item: "ghi", price: 5, quantity: 3, date: new Date("2020-11-01") },
  { item: "jkl", price: 15, quantity: 2, date: new Date("2020-12-01") },
  { item: "mno", price: 25, quantity: 1, date: new Date("2021-01-01") }
]);
```

查詢範例

```
db.sales.aggregate([
  {
```

```
    $bucket: {
      groupBy: "$price",
      boundaries: [0, 10, 20, 30],
      default: "Other",
      output: {
        "count": { $sum: 1 },
        "totalQuantity": { $sum: "$quantity" }
      }
    }
  },
  {
    $sort: { _id: 1 }
  }
])
```

輸出

```
[
  { _id: 0, count: 1, totalQuantity: 3 },
  { _id: 10, count: 2, totalQuantity: 4 },
  { _id: 20, count: 2, totalQuantity: 2 }
]
```

程式碼範例

若要檢視使用 $bucket命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const sales = db.collection('sales');

  const result = await sales.aggregate([
    {
      $bucket: {
        groupBy: "$price",
        boundaries: [0, 10, 20, 30],
```

```
        default: "Other",
        output: {
          "count": { $sum: 1 },
          "totalQuantity": { $sum: "$quantity" }
        }
      }
    },
    {
      $sort: { _id: 1 }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&lsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    sales = db['sales']

    result = list(sales.aggregate([
        {
            '$bucket': {
                'groupBy': '$price',
                'boundaries': [0, 10, 20, 30],
                'default': 'Other',
                'output': {
                    'count': {'$sum': 1},
                    'totalQuantity': {'$sum': '$quantity'}
                }
            }
        },
        {
            "$sort": { "_id": 1 }
```

```
        }
    ]))

    print(result)
    client.close()

example()
```

$ceil

4.0 版的新功能

Amazon DocumentDB 中的$ceil運算子與 MongoDB 一樣， 會將數字四捨五入到最接近的整數。當您需要對數值欄位執行數學操作，並確保結果是整數時，這會很有用。

參數

- expression：要四捨五入的數值表達式。

範例 (MongoDB Shell)

此範例示範如何使用 $ceil運算子來四捨五入數值欄位。

建立範例文件

```
db.numbers.insertMany([
  { "_id": 1, "value": 3.14 },
  { "_id": 2, "value": -2.7 },
  { "_id": 3, "value": 0 }
])
```

查詢範例

```
db.numbers.aggregate([
  { $project: {
    "roundedUp": { $ceil: "$value" }
  }}
])
```

輸出

```
{ "_id": 1, "roundedUp": 4 }
{ "_id": 2, "roundedUp": -2 }
{ "_id": 3, "roundedUp": 0 }
```

程式碼範例

若要檢視使用 $ceil命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $project: {
      "roundedUp": { $ceil: "$value" }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.numbers

    result = list(collection.aggregate([
```

```
        { '$project': {
            "roundedUp": { '$ceil': "$value" }
        }}
    ]))

    print(result)
    client.close()

example()
```

$changeStream

Elastic 叢集不支援。

$changeStream 彙總階段會開啟變更串流游標，以監控集合的即時變更。當發生插入、更新、取代或刪除操作時，它會傳回變更事件文件。

參數

- fullDocument：指定是否要傳回更新操作的完整文件。選項為 default 或 updateLookup。
- resumeAfter：選用。繼續字符，從變更串流中的特定點繼續。
- startAtOperationTime：選用。開始變更串流的時間戳記。
- allChangesForCluster：選用。布林值。當 時true， 會監控叢集的所有變更 （適用於管理員資料庫）。當 false（預設） 時， 只會監看指定的集合。

範例 (MongoDB Shell)

下列範例示範如何使用 $changeStream階段來監控集合的變更。

查詢範例

```
// Open change stream first
const changeStream = db.inventory.aggregate([
  { $changeStream: { fullDocument: "updateLookup" } }
]);

// In another session, insert a document
db.inventory.insertOne({ _id: 1, item: "Widget", qty: 10 });

// Back in the first session, read the change event
```

```
if (changeStream.hasNext()) {
  print(tojson(changeStream.next()));
}
```

輸出

```
{
  _id: { _data: '...' },
  operationType: 'insert',
  clusterTime: Timestamp(1, 1234567890),
  fullDocument: { _id: 1, item: 'Widget', qty: 10 },
  ns: { db: 'test', coll: 'inventory' },
  documentKey: { _id: 1 }
}
```

程式碼範例

若要檢視使用$changeStream彙總階段的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('inventory');

  // Open change stream
  const changeStream = collection.watch([]);

  changeStream.on('change', (change) => {
    console.log('Change detected:', change);
  });

  // Simulate insert in another operation
  setTimeout(async () => {
    await collection.insertOne({ _id: 1, item: 'Widget', qty: 10 });
  }, 1000);

  // Keep connection open to receive changes
```

```
  // In production, handle cleanup appropriately
}

example();
```

Python

```
from pymongo import MongoClient
import threading
import time

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['inventory']

    # Open change stream
    change_stream = collection.watch([])

    # Insert document in separate thread after delay
    def insert_doc():
        time.sleep(1)
        collection.insert_one({'_id': 1, 'item': 'Widget', 'qty': 10})

    threading.Thread(target=insert_doc).start()

    # Watch for changes
    for change in change_stream:
        print('Change detected:', change)
        break  # Exit after first change

    client.close()

example()
```

$cmp

Amazon DocumentDB 中的$cmp運算子用於比較兩個值，並傳回整數值，指出其相對順序。這是一個比較運算子，可比較兩個表達式，並傳回 -1、0 或 1 的整數值，取決於第一個值是否分別小於、等於或大於第二個值。

參數

- expression1：要比較的第一個表達式。
- expression2：要比較的第二個表達式。

範例 (MongoDB Shell)

下列範例示範運算$cmp子用來比較兩個數值的用量。

建立範例文件

```
db.collection.insertMany([
  { _id: 1, value1: 10, value2: 20 },
  { _id: 2, value1: 15, value2: 15 },
  { _id: 3, value1: 20, value2: 10 }
]);
```

查詢範例

```
db.collection.find({
  $expr: {
    $cmp: ["$value1", "$value2"]
  }
})
```

輸出

```
[
  { "_id" : 1, "value1" : 10, "value2" : 20 },
  { "_id" : 3, "value1" : 20, "value2" : 10 }
]
```

在此範例中，$cmp運算子會比較每個文件的 value1和 value2 欄位。結果為：

```
- `$cmp: ["$value1", "$value2"]` returns -1 for the first document (10 < 20), 0 for the
 second document (15 = 15), and 1 for the third document (20 > 10).
```

程式碼範例

若要檢視使用 $cmp命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

以下是在 Node.js 應用程式中搭配mongodb驅動程式使用 $cmp 運算子的範例：

```
const { MongoClient } = require('mongodb');

const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

async function main() {
  await client.connect();
  const db = client.db('test');
  const collection = db.collection('mycollection');

  // Insert sample documents
  await collection.insertMany([
    { _id: 1, value1: 10, value2: 20 },
    { _id: 2, value1: 15, value2: 15 },
    { _id: 3, value1: 20, value2: 10 }
  ]);

  // Query using $cmp operator
  const result = await collection.find({
    $expr: {
      $cmp: ['$value1', '$value2']
    }
  }).toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

以下是在 Python 應用程式中搭配pymongo驅動程式使用 $cmp 運算子的範例：

```
from pymongo import MongoClient
```

```
client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['mycollection']

# Insert sample documents
collection.insert_many([
    {'_id': 1, 'value1': 10, 'value2': 20},
    {'_id': 2, 'value1': 15, 'value2': 15},
    {'_id': 3, 'value1': 20, 'value2': 10}
])

# Query using $cmp operator
result = list(collection.find({
    '$expr': {
        '$cmp': ['$value1', '$value2']
    }
}))

print(result)

client.close()
```

Node.js 和 Python 範例的輸出將與 MongoDB Shell 範例相同：

```
[
  { "_id" : 1, "value1" : 10, "value2" : 20 },
  { "_id" : 2, "value1" : 15, "value2" : 15 },
  { "_id" : 3, "value1" : 20, "value2" : 10 }
]
```

$collStats

4.0 版的新功能

Amazon DocumentDB `$collStats` 中的彙總階段提供指定集合的統計資料，類似於 MongoDB shell 中的 `db.collection.stats()`命令。此階段可用來擷取集合的相關資訊，例如文件數量、集合的總大小，以及各種效能指標。

參數

- `latencyStats`：（選用）指定收集延遲統計資料選項的文件。Amazon DocumentDB 不支援此參數。
- `recordStats`：（選用）指定收集記錄統計資料選項的文件。Amazon DocumentDB 不支援此參數。
- `queryExecStats`：（選用）指定收集查詢執行統計資料選項的文件。Amazon DocumentDB 不支援此參數。

範例 (MongoDB Shell)

下列範例示範如何使用彙總階段來擷取db資料庫中名為 `$collStats` 之集合`test`的統計資料。

建立範例文件

```
db.test.insertMany([
  { "name": "John", "age": 30 },
  { "name": "Jane", "age": 25 },
  { "name": "Bob", "age": 35 }
]);
```

查詢範例

```
db.test.aggregate([
  { $collStats: {} }
]);
```

輸出

```
{
  "ns" : "db.test",
  "count" : 3,
  "size" : 87,
  "avgObjSize" : 29,
  "storageSize" : 40960,
  "capped" : false,
  "nindexes" : 1,
  "totalIndexSize" : 8192,
  "indexSizes" : {
    "_id_" : 8192
  },
  "collScans" : 0,
  "idxScans" : 0,
```

```
  "opCounter" : {
    "numDocsIns" : 3,
    "numDocsUpd" : 0,
    "numDocsDel" : 0
  },
  "cacheStats" : {
    "collBlksHit" : 0,
    "collBlksRead" : 0,
    "collHitRatio" : 0,
    "idxBlksHit" : 0,
    "idxBlksRead" : 0,
    "idxHitRatio" : 0
  },
  "lastReset" : "2023-04-11T12:00:00Z",
  "ok" : 1,
  "operationTime" : Timestamp(1681206000, 1)
}
```

程式碼範例

若要檢視使用 $collStats命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

以下是如何使用官方 MongoDB Node.js $collStats 驅動程式在 Node.js 應用程式中使用彙總階段的範例：

```
const { MongoClient } = require('mongodb');

async function runCollStatsExample() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('db');
  const collection = db.collection('test');

  const result = await collection.aggregate([
    { $collStats: {} }
  ]).toArray();

  console.log(result);

  await client.close();
```

```
}

runCollStatsExample();
```

Python

以下是如何使用 PyMongo `$collStats` 驅動程式在 Python 應用程式中使用彙總階段的範例：

```
from pymongo import MongoClient

def run_coll_stats_example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['db']
    collection = db['test']

    result = list(collection.aggregate([
        { '$collStats': {} }
    ]))

    print(result)

    client.close()

run_coll_stats_example()
```

$concat

Amazon DocumentDB `$concat` 中的彙總運算子串連（或合併）文件中的多個字串，以產生可傳回給應用程式的單一字串。這可減少在應用程式中完成的工作，因為字串操作是在資料庫層級執行。

參數

- `expression1`：要串連的第一個字串。
- `expression2`：要串連的第二個字串。
- `...`：要串連的其他字串（選用）。

範例 (MongoDB Shell)

在此範例中，我們會串連使用者的名字和姓氏，以產生每個人的全名。

建立範例文件

```
db.people.insertMany([
  { "_id":1, "first_name":"Jane", "last_name":"Doe", "DOB":"2/1/1999", "Desk": "MSP102-
MN"},
  { "_id":2, "first_name":"John", "last_name":"Doe", "DOB":"12/21/1992", "Desk":
 "DSM301-IA"},
  { "_id":3, "first_name":"Steve", "last_name":"Smith", "DOB":"3/21/1981",
 "Desk":"MKE233-WI"}
])
```

查詢範例

```
db.people.aggregate([
  { $project: { full_name: { $concat: [ "$first_name", " ", "$last_name"] } } }
])
```

輸出

```
{ "_id" : 1, "full_name" : "Jane Doe" }
{ "_id" : 2, "full_name" : "John Doe" }
{ "_id" : 3, "full_name" : "Steve Smith" }
```

程式碼範例

若要檢視使用 $concat命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function concatenateNames() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');

  const result = await db.collection('people').aggregate([
    { $project: { full_name: { $concat: [ "$first_name", " ", "$last_name"] } } }
  ]).toArray();
```

```
  console.log(result);

  await client.close();
}

concatenateNames();
```

Python

```
from pymongo import MongoClient

def concatenate_names():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']

    result = list(db.people.aggregate([
        { '$project': { 'full_name': { '$concat': [ '$first_name', ' ',
 '$last_name' ] } } }
    ]))

    print(result)

    client.close()

concatenate_names()
```

$concatArrays

Amazon DocumentDB $concatArrays 中的彙總運算子用於將兩個或多個陣列串連至單一陣列。當您需要將多個資料陣列合併為單一陣列以進行進一步處理或分析時，這非常有用。

參數

- array1：要串連的第一個陣列。
- array2：要串連的第二個陣列。
- [array3, ...]：（選用）要串連的其他陣列。

範例 (MongoDB Shell)

下列範例示範如何使用 $concatArrays運算子將兩個陣列合併為單一陣列。

建立範例文件

```
db.collection.insertMany([
  {
    "_id": 1,
    "name": "John Doe",
    "hobbies": ["reading", "swimming"],
    "skills": ["programming", "design"]
  },
  {
    "_id": 2,
    "name": "Jane Smith",
    "hobbies": ["hiking", "cooking"],
    "skills": ["marketing", "analysis"]
  }
]);
```

查詢範例

```
db.collection.aggregate([
  {
    $project: {
      _id: 0,
      name: 1,
      all_activities: { $concatArrays: ["$hobbies", "$skills"] }
    }
  }
]);
```

輸出

```
[
  {
    "name": "John Doe",
    "all_activities": [
      "reading",
      "swimming",
      "programming",
      "design"
```

```
    ]
  },
  {
    "name": "Jane Smith",
    "all_activities": [
      "hiking",
      "cooking",
      "marketing",
      "analysis"
    ]
  }
]
```

程式碼範例

若要檢視使用 $concatArrays命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('mydb');
  const collection = db.collection('mycollection');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 0,
        name: 1,
        all_activities: { $concatArrays: ['$hobbies', '$skills'] }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['mydb']
    collection = db['mycollection']

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 0,
                'name': 1,
                'all_activities': { '$concatArrays': ['$hobbies', '$skills'] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$cond

Amazon DocumentDB 中的$cond運算子用於評估條件式表達式，並傳回兩個可能的結果表達式之一。

參數

- if：要評估的布林表達式。
- then：如果表達式為 true，則傳回的if表達式。
- else：如果表達式為 false，則傳回的if表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $cond運算子，根據一個人的年齡傳回值。

建立範例文件

```
db.people.insertMany([
  { _id: 1, name: "John Doe", age: 35 },
  { _id: 2, name: "Jane Doe", age: 25 },
  { _id: 3, name: "Bob Smith", age: 65 }
]);
```

查詢範例

```
db.people.aggregate([
  {
    $project: {
      name: 1,
      ageGroup: {
        $cond: {
          if: { $lt: ["$age", 30] },
          then: "young",
          else: {
            $cond: {
              if: { $lt: ["$age", 65] },
              then: "middle-aged",
              else: "elderly"
            }
          }
        }
      }
    }
  }
])
```

輸出

```
[
  { "_id" : 1, "name" : "John Doe", "ageGroup" : "middle-aged" },
  { "_id" : 2, "name" : "Jane Doe", "ageGroup" : "young" },
  { "_id" : 3, "name" : "Bob Smith", "ageGroup" : "elderly" }
]
```

程式碼範例

若要檢視使用 $cond命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const result = await collection.aggregate([
    { $project: {
        name: 1,
        ageGroup: {
            $cond: {
                if: { $lt: ["$age", 30] },
                then: "young",
                else: {
                    $cond: {
                        if: { $lt: ["$age", 65] },
                        then: "middle-aged",
                        else: "elderly"
                    }
                }
            }
        }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient


def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.people

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'ageGroup': {
                    '$cond': {
                        'if': { '$lt': ["$age", 30]},
                        'then': "young",
                        'else': {
                            '$cond': {
                                'if': { '$lt': ["$age", 65]},
                                'then': "middle-aged",
                                'else': "elderly"
                            }
                        }
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$轉換

4.0 版的新功能

Amazon DocumentDB 中的$convert運算子用於將值從一種資料類型轉換為另一種資料類型。當您需要對不同類型的資料執行操作時，例如將字串轉換為數字或將日期轉換為時間戳記，此運算子非常有用。

參數

- to：要轉換值的目標資料類型。支援的值為 "string"、"double"、"long"、"int"、"date" 和 "boolean"。
- from：值的目前資料類型。如果未指定，Amazon DocumentDB 將嘗試自動偵測資料類型。
- onError：（選用）轉換失敗時要傳回的值。可以是特定值或下列其中一個特殊值："null"、"zerofill"或 "error"。
- onNull：（選用）如果輸入值為 時要傳回的值null。可以是特定值或下列其中一個特殊值："null"、 "zerofill"或 "error"。

範例 (MongoDB Shell)

下列範例示範使用 $convert運算子將字串值轉換為日期。

建立範例文件

```
db.users.insertMany([
  { _id: 1, name: "John Doe", joinedOn: "2022-01-01" },
  { _id: 2, name: "Jane Smith", joinedOn: "2023-02-15" },
  { _id: 3, name: "Bob Johnson", joinedOn: "invalid date" }
]);
```

查詢範例

```
db.users.aggregate([
  {
    $project: {
      _id: 1,
      name: 1,
      joinedOn: {
        $convert: {
          input: "$joinedOn",
          to: "date",
          onError: "null",
          onNull: "null"
        }
      }
    }
  }
])
```

輸出

```
[
  { "_id" : 1, "name" : "John Doe", "joinedOn" : ISODate("2022-01-01T00:00:00Z") },
  { "_id" : 2, "name" : "Jane Smith", "joinedOn" : ISODate("2023-02-15T00:00:00Z") },
  { "_id" : 3, "name" : "Bob Johnson", "joinedOn" : null }
]
```

程式碼範例

若要檢視使用 $convert命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require("mongodb");

async function example() {
  const client = await MongoClient.connect("mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false");
  const db = client.db("test");
  const users = db.collection("users");

  const results = await users.aggregate([
    {
      $project: {
        _id: 1,
        name: 1,
        joinedOn: {
          $convert: {
            input: "$joinedOn",
            to: "date",
            onError: "null",
            onNull: "null"
          }
        }
      }
    }
  ]).toArray();

  console.log(results);
  client.close();
}
```

```
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient("mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false")
    db = client["test"]
    users = db["users"]

    results = list(users.aggregate([
        {
            "$project": {
                "_id": 1,
                "name": 1,
                "joinedOn": {
                    "$convert": {
                        "input": "$joinedOn",
                        "to": "date",
                        "onError": "null",
                        "onNull": "null"
                    }
                }
            }
        }
    ]))

    print(results)
    client.close()

example()
```

$cos

8.0.1 版的新功能。

Amazon DocumentDB 中的$cos運算子會傳回以弧度測量的值餘弦。在彙總管道中使用它，對數值欄位執行三角計算。

參數

- `expression`：解析為弧度數字的表達式。

範例 (MongoDB Shell)

下列範例顯示如何使用 $cos運算子計算弧度角值的餘弦。

建立範例文件

```
db.angles.insertMany([
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 1.0471975511965976 },
  { "_id": 3, "angle": 3.141592653589793 }
]);
```

查詢範例

```
db.angles.aggregate([
  { $project: {
    "cosine": { $cos: "$angle" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "cosine": 1 },
  { "_id": 2, "cosine": 0.5000000000000001 },
  { "_id": 3, "cosine": -1 }
]
```

程式碼範例

若要檢視使用 $cos 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('angles');

  const result = await collection.aggregate([
    { $project: {
      "cosine": { $cos: "$angle" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['angles']

    result = list(collection.aggregate([
        { '$project': {
            'cosine': { '$cos': '$angle' }
        }}
    ]))

    print(result)
    client.close()

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

$cosh

8.0.1 版的新功能。

Amazon DocumentDB 中的$cosh運算子會傳回值的雙曲餘弦。在彙總管道中使用它，對數值欄位執行雙曲三角計算。

參數

- expression：解析為數字的表達式。

傳回類型預設為 double 。如果輸入是 128 位元的小數位數，則輸出也是 128 位元的小數位數。

範例 (MongoDB Shell)

下列範例顯示如何使用 $cosh運算子來計算數值的雙曲餘弦。

建立範例文件

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 1 },
  { "_id": 3, "value": -1 }
]);
```

查詢範例

```
db.values.aggregate([
  { $project: {
    "result": { $cosh: "$value" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "result": 1 },
  { "_id": 2, "result": 1.5430806348152437 },
  { "_id": 3, "result": 1.5430806348152437 }
]
```

程式碼範例

若要檢視使用 $cosh 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "result": { $cosh: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'result': { '$cosh': '$value' }
        }}
    ]))
```

```
    print(result)
    client.close()

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

$count （階段）

Amazon DocumentDB $count 中的彙總階段用於計算傳入階段的文件數量。它通常用作彙總管道中的最後一個階段，以傳回符合先前階段的文件總數。

參數

- `field`：要計數的欄位。此參數是選用的，如果未提供，則階段會計算輸入文件的總數。

範例 (MongoDB Shell)

下列範例示範如何使用 $count階段來取得集合中的文件總數。

建立範例文件

```
db.users.insertMany([
  { name: "John", age: 30 },
  { name: "Jane", age: 25 },
  { name: "Bob", age: 35 },
  { name: "Alice", age: 28 }
]);
```

查詢範例

```
db.users.aggregate([
  { $count: "total" }
]);
```

輸出

```
{ "total" : 4 }
```

此範例會彙總users集合，並使用 $count階段來計算文件的總數。

程式碼範例

若要檢視使用 $count命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function countDocuments() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('mydb');
  const collection = db.collection('users');

  const result = await collection.aggregate([
    { $count: "total" }
  ]).toArray();

  console.log(result[0].total);

  await client.close();
}

countDocuments();
```

Python

```
from pymongo import MongoClient

def count_documents():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['mydb']
    collection = db['users']

    result = list(collection.aggregate([
        { '$count': "total" }
    ]))

    print(result[0]['total'])
```

```
    client.close()

count_documents()
```

$count （累積器）

8.0.1 版的新功能。

使用$group階段內的$count累積器來傳回每個群組中的文件數量。它接受空物件{}做為其引數。

這與$count管道階段不同，這是一個獨立的階段，可計算通過管道的所有文件。$count 累積器會改為計算 產生的每個群組中的文件$group。

語法

```
{ $count: {} }
```

參數

- $count 累積器不採用任何引數。它接受空物件，{}並傳回每個群組中的文件計數。

範例 (MongoDB Shell)

下列範例顯示如何使用$count累積器計算每個類別中的產品數量。

建立範例文件

```
db.products.insertMany([
  { name: "Widget", category: "A" },
  { name: "Gadget", category: "A" },
  { name: "Doohickey", category: "B" },
  { name: "Thingamajig", category: "B" },
  { name: "Whatsit", category: "B" }
])
```

查詢範例

```
db.products.aggregate([
  { $group: { _id: "$category", count: { $count: {} } } }
```

```
])
```

輸出

```
[
  { "_id": "A", "count": 2 },
  { "_id": "B", "count": 3 }
]
```

程式碼範例

若要檢視使用$count累積器的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const db = client.db('test');
    const collection = db.collection('products');

    const result = await collection.aggregate([
      { $group: { _id: "$category", count: { $count: {} } } }
    ]).toArray();

    console.log(result);

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client['test']
        collection = db['products']

        result = list(collection.aggregate([
            { '$group': { '_id': '$category', 'count': { '$count': {} } } }
        ]))

        pprint(result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$currentOp

$currentOp 彙總階段會傳回資料庫中目前執行中操作的相關資訊。此階段適用於監控彙總管道中的作用中查詢和操作。

參數

- allUsers （選用）：設為 時true， 會傳回所有使用者的操作。預設值為 false。
- idleConnections （選用）：設為 時true， 包含閒置連線。預設值為 false。
- idleCursors （選用）：設為 時true，包含閒置游標的相關資訊。預設值為 false。
- idleSessions （選用）：設為 時true， 包含閒置工作階段的相關資訊。預設值為 true。

- localOps（選用）：設為 時true， 包含本機操作。預設值為 false。

範例 (MongoDB Shell)

下列範例示範如何使用$currentOp彙總階段來擷取作用中讀取操作的相關資訊。

查詢範例

```
db.aggregate([
  { $currentOp: { allUsers: true, idleConnections: false } },
  { $match: { op: "query" } }
])
```

輸出

```
[
  {
    "opid": "12345",
    "active": true,
    "op": "query",
    "ns": "test.users",
    "secs_running": 2
  }
]
```

程式碼範例

若要檢視使用 $currentOp命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('admin');

  const result = await db.aggregate([
    { $currentOp: { allUsers: true, idleConnections: false } },
    { $match: { op: "query" } }
```

```
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['admin']

    result = list(db.aggregate([
        { '$currentOp': { 'allUsers': True, 'idleConnections': False } },
        { '$match': { 'op': 'query' } }
    ]))

    print(result)
    client.close()

example()
```

$dateAdd

5.0 版的新功能

Amazon DocumentDB $dateAdd 中的彙總運算子可讓您將持續時間新增至日期和時間值。

參數

- date：要新增持續時間的日期和時間值。
- duration：要新增至date值的持續時間。這可以指定為具有 years、months、、weeks、dayshours、 minutes和 金鑰的物件seconds。
- timezone：（選用）執行日期新增時要使用的時區。如果未指定，則會使用 Amazon DocumentDB 叢集的預設時區。

範例 (MongoDB Shell)

下列範例示範如何使用 $dateAdd運算子將 2 天又 12 小時新增至日期。

建立範例文件

```
db.events.insertMany([
  { _id: 1, eventDate: ISODate("2023-04-01T10:00:00Z") },
  { _id: 2, eventDate: ISODate("2023-04-02T12:00:00Z") },
  { _id: 3, eventDate: ISODate("2023-04-03T14:00:00Z") }
]);
```

查詢範例

```
db.events.aggregate([
  {
    $project: {
      _id: 1,
      eventDate: 1,
      eventDatePlustwodaysandtwelvehours: {
        $dateAdd: {
          startDate: {
            $dateAdd: {
              startDate: "$eventDate",
              unit: "day",
              amount: 2
            }
          },
          unit: "hour",
          amount: 12
        }
      }
    }
  }
])
```

輸出

```
[
  {
    "_id": 1,
    "eventDate": "2023-04-01T10:00:00Z",
```

```
    "eventDatePlustwodaysandtwelvehours": ISODate("2023-04-03T22:00:00Z)"
  },
  {
    "_id": 2,
    "eventDate": "2023-04-02T12:00:00Z",
    "eventDatePlustwodaysandtwelvehours": ISODate("2023-04-05T00:00:00Z)"
  },
  {
    "_id": 3,
    "eventDate": "2023-04-03T14:00:00Z",
    "eventDatePlustwodaysandtwelvehours": ISODate("2023-04-06T02:00:00Z)"
  }
]
```

程式碼範例

若要檢視使用 $dateAdd命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('events');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        eventDate: 1,
        eventDatePlustwodaysandtwelvehours: {
          $dateAdd: {
            startDate: {
              $dateAdd: {
                startDate: "$eventDate",
                unit: "day",
                amount: 2
              }
            },
            unit: "hour",
```

```
            amount: 12
          }
        }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient
from bson.date_time import datetime

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['events']

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'eventDate': 1,
                'eventDatePlustwodaysandtwelvehours': {
                    '$dateAdd': {
                        'startDate': {
                            '$dateAdd' : {
                                'startDate': '$eventDate',
                                'unit': 'day',
                                'amount': 2,
                            }
                        },
                        'unit': 'hour',
                        'amount': 12,
                    }
```

```
                }
            }
        }
    ]))

    print(result)

    client.close()

example()
```

$dateDiff

5.0 版的新功能

Elastic 叢集不支援。

$dateDiff 彙總運算子會以指定的單位計算兩個日期之間的差異。它會傳回開始和結束日期之間跨越的單位邊界數目。

參數

- startDate：開始日期表達式。
- endDate：結束日期表達式。
- unit：差異的時間單位。支援的單位為 year、quarter、month、week、day、hourminute、、 second和 millisecond。

範例 (MongoDB Shell)

下列範例示範如何使用 $dateDiff運算子來計算訂單置放和交付之間的天數。

建立範例文件

```
db.shipments.insertMany([
  {
    orderId: 1001,
    orderDate: ISODate("2025-01-10T08:00:00Z"),
    deliveryDate: ISODate("2025-01-15T14:30:00Z")
  },
  {
    orderId: 1002,
```

```
    orderDate: ISODate("2025-02-05T10:00:00Z"),
    deliveryDate: ISODate("2025-02-12T16:45:00Z")
  }
]);
```

查詢範例

```
db.shipments.aggregate([
  {
    $project: {
      orderId: 1,
      orderDate: 1,
      deliveryDate: 1,
      daysToDeliver: {
        $dateDiff: {
          startDate: "$orderDate",
          endDate: "$deliveryDate",
          unit: "day"
        }
      }
    }
  }
]);
```

輸出

```
[
  {
    _id: ObjectId('6924a5f2d66dcae121d29517'),
    orderId: 1001,
    orderDate: ISODate('2025-01-10T08:00:00.000Z'),
    deliveryDate: ISODate('2025-01-15T14:30:00.000Z'),
    daysToDeliver: 5
  },
  {
    _id: ObjectId('6924a5f2d66dcae121d29518'),
    orderId: 1002,
    orderDate: ISODate('2025-02-05T10:00:00.000Z'),
    deliveryDate: ISODate('2025-02-12T16:45:00.000Z'),
    daysToDeliver: 7
  }
]
```

程式碼範例

若要檢視使用 $dateDiff命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const shipments = db.collection('shipments');

  const result = await shipments.aggregate([
    {
      $project: {
        orderId: 1,
        orderDate: 1,
        deliveryDate: 1,
        daysToDeliver: {
          $dateDiff: {
            startDate: "$orderDate",
            endDate: "$deliveryDate",
            unit: "day"
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    shipments = db['shipments']

    result = list(shipments.aggregate([
        {
            "$project": {
                "orderId": 1,
                "orderDate": 1,
                "deliveryDate": 1,
                "daysToDeliver": {
                    "$dateDiff": {
                        "startDate": "$orderDate",
                        "endDate": "$deliveryDate",
                        "unit": "day"
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$dateFromParts

5.0.1 和 8.0 版的新功能

Amazon DocumentDB $dateFromParts 中的彙總運算子會建構並傳回其組成部分的日期和時間值，例如年、月、日、小時、分鐘、秒和毫秒。

參數

- year：日曆年。不使用 ISO 8601 組件時為必要。
- month、day、hour、minute、second、millisecond：（選用）剩餘的日期和時間元件。省略的元件預設為最低的有效值。

- isoWeekYear、isoWeek、isoDayOfWeek：（選用）ISO 8601 日期元件。使用這些 而非 year、 month和 day來建構 ISO 8601 週日期的日期。
- timezone：（選用）建構日期時要使用的時區。如果未指定，則會使用 UTC。

範例 (MongoDB Shell)

下列範例示範如何使用 $dateFromParts運算子從個別元件建構日期。

建立範例文件

```
db.parts.insertMany([
  { _id: 1, y: 2023, m: 4, d: 1 },
  { _id: 2, y: 2023, m: 12, d: 25 }
]);
```

查詢範例

```
db.parts.aggregate([
  {
    $project: {
      _id: 1,
      constructedDate: {
        $dateFromParts: { year: "$y", month: "$m", day: "$d" }
      }
    }
  }
])
```

輸出

```
[
  { "_id": 1, "constructedDate": ISODate("2023-04-01T00:00:00Z") },
  { "_id": 2, "constructedDate": ISODate("2023-12-25T00:00:00Z") }
]
```

程式碼範例

若要檢視使用 $dateFromParts命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('parts');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        constructedDate: {
          $dateFromParts: { year: "$y", month: "$m", day: "$d" }
        }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['parts']

    result = list(collection.aggregate([
        {
            '$project': {
```

```
                '_id': 1,
                'constructedDate': {
                    '$dateFromParts': { 'year': '$y', 'month': '$m', 'day': '$d' }
                }
            }
        }
    ]))

    print(result)

    client.close()

example()
```

$dateFromString

Amazon DocumentDB $dateFromString 中的彙總運算子可讓您將日期時間字串轉換為日期物件。當您的資料儲存為字串，但您需要對資料執行以日期為基礎的操作時，這會很有用。

參數

- dateString：代表日期和時間的字串。
- format：（選用） 指定 格式的字串dateString。如果未提供，Amazon DocumentDB 將嘗試剖析 ISO-8601 格式的字串。
- timezone：（選用） 指定時區的字串。如果未提供，Amazon DocumentDB 將使用伺服器的時區。
- onError：（選用） 指定轉換失敗時要採取的動作。可能的值為 'error'（預設值，這會擲回錯誤）、 'null'（傳回 null) 或 'replace'（將值取代為 onErrorMessage選項中指定的取代字串）。
- onErrorMessage：（選用） 如果 onError 設定為 'replace'，此選項會指定取代字串。

範例 (MongoDB Shell)

下列範例示範如何使用 $dateFromString將日期字串轉換為 Amazon DocumentDB 中的日期物件。

建立範例文件

```
db.missionLog.insertMany([
```

```
{ _id: 1, event: "missionStart", logDate: "2020-03-15T13:41:33"},
{ _id: 2, event: "jumpPoint1", logDate: "2020-03-15T13:45:34"},
{ _id: 3, event: "jumpPoint2", logDate: "2020-03-15T13:48:21"},
{ _id: 4, event: "jumpPoint3", logDate: "2020-03-15T13:52:09"},
{ _id: 5, event: "missionEnd", logDate: "2020-03-15T13:58:44"}
]);
```

查詢範例

```
db.missionLog.aggregate([
  {
    $project: {
      event: '$event',
      logDate: {
        $dateFromString: {
          dateString: '$logDate'
        }
      }
    }
  }
]);
```

輸出

```
[
  {
    "_id": 1,
    "event": "missionStart",
    "logDate": ISODate("2020-03-15T13:41:33Z")
  },
  {
    "_id": 2,
    "event": "jumpPoint1",
    "logDate": ISODate("2020-03-15T13:45:34Z")
  },
  {
    "_id": 3,
    "event": "jumpPoint2",
    "logDate": ISODate("2020-03-15T13:48:21Z")
  },
  {
    "_id": 4,
    "event": "jumpPoint3",
```

```
    "logDate": ISODate("2020-03-15T13:52:09Z")
  },
  {
    "_id": 5,
    "event": "missionEnd",
    "logDate": ISODate("2020-03-15T13:58:44Z")
  }
]
```

程式碼範例

若要檢視使用 $dateFromString命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('missionLog');

  const result = await collection.aggregate([
    {
      $project: {
        event: '$event',
        logDate: {
          $dateFromString: {
            dateString: '$logDate'
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['missionLog']

    result = list(collection.aggregate([
        {
            '$project': {
                'event': '$event',
                'logDate': {
                    '$dateFromString': {
                        'dateString': '$logDate'
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$dateSubtract

5.0 版的新功能

Amazon DocumentDB $dateSubtract 中的彙總運算子可讓您從日期值中減去指定的持續時間。

參數

- date：解析為日期或時間戳記的日期表達式。
- subtrahend：持續時間表達式，指定要從date表達式中減去的時間量。
- unit：指定subtrahend表達式時間單位的字串。支援的單位為「年」、「季度」、「月」、「週」、「日」、「小時」、「分鐘」、「秒」和「毫秒」。

範例 (MongoDB Shell)

下列範例示範如何使用 $dateSubtract運算子來計算一年前從目前日期起算的日期。

建立範例文件

```
db.events.insertOne({
  eventName: "Player joined",
  eventTime: ISODate("2023-04-01T12:00:00Z")
});
```

查詢範例

```
db.events.aggregate([
  {
    $project: {
      eventName: 1,
      oneYearAgo: {
        $dateSubtract: {
          startDate: "$eventTime",
          amount: 1,
          unit: "year"
        }
      }
    }
  }
])
```

輸出

```
{
  "_id" : ObjectId("64567890abcdef012345678"),
  "eventName" : "Player joined",
  "oneYearAgo" : ISODate("2022-04-01T12:00:00Z")
}
```

程式碼範例

若要檢視使用 $dateSubtract命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const events = db.collection('events');

  const result = await events.aggregate([
    {
      $project: {
        eventName: 1,
        oneYearAgo: {
          $dateSubtract: {
            startDate: "$eventTime",
            amount: 1,
            unit: "year"
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient
from bson.date_time import datetime

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    events = db['events']

    result = list(events.aggregate([
        {
```

```
            '$project': {
                'eventName': 1,
                'oneYearAgo': {
                    '$dateSubtract': {
                        'startDate': '$eventTime',
                        'amount': 1,
                        'unit': 'year'
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$dateToParts

5.0.1 和 8.0 版的新功能

Amazon DocumentDB $dateToParts 中的彙總運算子會傳回文件，其中包含日期和時間值的組成部分，例如年、月、日、小時、分鐘、秒和毫秒。

參數

- date：要分段的日期和時間值。
- timezone：（選用）擷取日期部分時要使用的時區。如果未指定，則會以 UTC 傳回組件。
- iso8601：（選用）如果設定為 true，則傳回的文件會使用 ISO 8601 日期部分 (isoWeekYear、isoWeek、isoDayOfWeek)。預設值為 false。

範例 (MongoDB Shell)

下列範例示範如何使用 $dateToParts運算子來傳回日期的個別部分。

建立範例文件

```
db.events.insertMany([
  { _id: 1, eventDate: ISODate("2023-04-01T10:30:15Z") },
```

```
  { _id: 2, eventDate: ISODate("2023-12-25T18:45:00Z") }
]);
```

查詢範例

```
db.events.aggregate([
  {
    $project: {
      _id: 1,
      parts: { $dateToParts: { date: "$eventDate" } }
    }
  }
])
```

輸出

```
[
  {
    "_id": 1,
    "parts": { "year": 2023, "month": 4, "day": 1, "hour": 10, "minute": 30, "second":
 15, "millisecond": 0 }
  },
  {
    "_id": 2,
    "parts": { "year": 2023, "month": 12, "day": 25, "hour": 18, "minute": 45,
 "second": 0, "millisecond": 0 }
  }
]
```

程式碼範例

若要檢視使用 $dateToParts命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
```

```
  const collection = db.collection('events');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        parts: { $dateToParts: { date: "$eventDate" } }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['events']

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'parts': { '$dateToParts': { 'date': '$eventDate' } }
            }
        }
    ]))

    print(result)

    client.close()

example()
```

$dateToString

Amazon DocumentDB $dateToString 中的彙總運算子用於將日期或時間戳記值轉換為字串表示法。當您需要以特定方式格式化日期和時間來顯示或進一步處理時，這會很有用。

參數

- date：要轉換為字串的日期或時間戳記值。
- format：指定應代表日期的格式的字串。格式字串可以包含各種格式指標，例如%Y四位數年份、%m兩位數月份、該月%d兩位數日期等。
- timezone：（選用）用於轉換的時區。如果未指定，則會使用託管 Amazon DocumentDB 叢集之伺服器的時區。
- onNull：（選用）date 參數為 時要傳回的值null。

範例 (MongoDB Shell)

下列範例示範 $dateToString運算子的用途，以格式化missionLog集合logDate的欄位。

建立範例文件

```
db.missionLog.insertMany([
  { _id: 1, "event":"missionStart", logDate: new Date("2020-03-15T13:41:33Z") },
  { _id: 2, "event":"jumpPoint1", logDate: new Date("2020-03-15T13:45:34Z") },
  { _id: 3, "event":"jumpPoint2", logDate: new Date("2020-03-15T13:48:21Z") },
  { _id: 4, "event":"jumpPoint3", logDate: new Date("2020-03-15T13:52:09Z") },
  { _id: 5, "event":"missionEnd", logDate: new Date("2020-03-15T13:58:44Z") }
]);
```

查詢範例

```
db.missionLog.aggregate([
  {
    $project: {
      event: "$event",
      logDateFormatted: {
        $dateToString: {
          format: "%Y-%m-%d %H:%M:%S",
          date: "$logDate"
        }
      }
    }
```

```
  }
])
```

輸出

```
[
  {
    "_id": 1,
    "event": "missionStart",
    "logDateFormatted": "2020-03-15 13:41:33"
  },
  {
    "_id": 2,
    "event": "jumpPoint1",
    "logDateFormatted": "2020-03-15 13:45:34"
  },
  {
    "_id": 3,
    "event": "jumpPoint2",
    "logDateFormatted": "2020-03-15 13:48:21"
  },
  {
    "_id": 4,
    "event": "jumpPoint3",
    "logDateFormatted": "2020-03-15 13:52:09"
  },
  {
    "_id": 5,
    "event": "missionEnd",
    "logDateFormatted": "2020-03-15 13:58:44"
  }
]
```

程式碼範例

若要檢視使用 $dateToString命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

以下是在 Node.js 應用程式中使用 $dateToString 運算子的範例：

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('missionLog');

  const result = await collection.aggregate([
    {
      $project: {
        event: "$event",
        logDateFormatted: {
          $dateToString: {
            format: "%Y-%m-%d %H:%M:%S",
            date: "$logDate"
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

以下是在 Python 應用程式中使用 $dateToString 運算子的範例：

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['missionLog']

    pipeline = [
        {
            '$project': {
                'event': '$event',
```

```
                'logDateFormatted': {
                    '$dateToString': {
                        'format': '%Y-%m-%d %H:%M:%S',
                        'date': '$logDate'
                    }
                }
            }
        }
    ]

    result = list(collection.aggregate(pipeline))
    print(result)
    client.close()

example()
```

$dateTrunc

8.0 版的新功能

Elastic 叢集不支援。

Amazon DocumentDB $dateTrunc 中的彙總運算子會將日期截斷為指定的單位。

參數

- date：解析為日期或時間戳記的日期表達式。
- unit：指定子trahend 表達式時間單位的字串。支援的單位為 year、quarter、month、week、day、hourminute、、 second和 millisecond。

範例 (MongoDB Shell)

下列範例示範如何使用 $dateTrunc運算子將日期截斷為小時。

建立範例文件

```
db.events.insertMany([
  {
    eventName: "Event 1",
    eventTime: ISODate("2025-04-01T12:15:00Z")
  },
```

```
  {
    eventName: "Event 2",
    eventTime: ISODate("2025-08-15T14:33:22Z")
  },
]);
```

查詢範例

```
db.events.aggregate([
  {
    $project: {
      eventName: 1,
      eventTime: 1,
      truncatedToHour: {
        $dateTrunc: {
          date: "$eventTime",
          unit: "hour"
        }
      }
    }
  }
]);
```

輸出

```
[
  {
    _id: ObjectId('6924a258d66dcae121d29515'),
    eventName: 'Event 1',
    eventTime: ISODate('2025-04-01T12:15:00.000Z'),
    truncatedToHour: ISODate('2025-04-01T12:00:00.000Z')
  },
  {
    _id: ObjectId('6924a258d66dcae121d29516'),
    eventName: 'Event 2',
    eventTime: ISODate('2025-08-15T14:33:22.000Z'),
    truncatedToHour: ISODate('2025-08-15T14:00:00.000Z')
  }
]
```

程式碼範例

若要檢視使用 $dateTrunc命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const events = db.collection('events');

  const result = await events.aggregate([
    {
      $project: {
        eventName: 1,
        eventTime: 1,
        truncatedToHour: {
          $dateTrunc: {
            date: "$eventTime",
            unit: "hour"
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    events = db['events']
```

```
    result = list(events.aggregate([
        {
            "$project": {
                "eventName": 1,
                "eventTime": 1,
                "truncatedToHour": {
                    "$dateTrunc": {
                        "date": "$eventTime",
                        "unit": "hour"
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$dayOfMonth

Amazon DocumentDB $dayOfMonth 中的彙總運算子會擷取指定日期的當月日期 （ 從 1 到 31)。此運算子適用於分組、篩選或擷取文件中日期欄位的月份日期。

參數

- `date expression`：日期表達式可以是文件的日期欄位、日期物件或日期字串。

範例 (MongoDB Shell)

此範例示範如何使用 $dayOfMonth 運算子，從文件中的日期欄位中擷取當月的日期。

建立範例文件

```
db.events.insertMany([
  { _id: 1, eventDate: new Date("2022-01-15T12:00:00Z") },
  { _id: 2, eventDate: new Date("2022-02-28T15:30:00Z") },
  { _id: 3, eventDate: new Date("2022-03-01T09:45:00Z") },
  { _id: 4, eventDate: new Date("2022-04-30T23:59:59Z") }
]);
```

查詢範例

```
db.events.aggregate([
  { $project: {
    eventDay: { $dayOfMonth: "$eventDate" }
  }}
])
```

輸出

```
[
  { "_id" : 1, "eventDay" : 15 },
  { "_id" : 2, "eventDay" : 28 },
  { "_id" : 3, "eventDay" : 1 },
  { "_id" : 4, "eventDay" : 30 }
]
```

程式碼範例

若要檢視使用 $dayOfMonth命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const events = db.collection('events');

  const result = await events.aggregate([
    { $project: {
      eventDay: { $dayOfMonth: "$eventDate" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}
```

```
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.mydatabase
    events = db.events

    result = list(events.aggregate([
        { "$project": {
            "eventDay": { "$dayOfMonth": "$eventDate" }
        }}
    ]))

    print(result)
    client.close()

example()
```

$dayOfWeek

Amazon DocumentDB 中的$dayOfWeek運算子會從指定的日期欄位中擷取一週中的某一天。它以介於 1（週日）和 7（週六）之間的數字傳回星期幾，這是與 MongoDB 中相同的行為。

參數

- `date field`：要從中擷取星期幾的日期欄位。

範例 (MongoDB Shell)

此範例示範如何使用 $dayOfWeek 運算子，從weather集合中的 date 欄位擷取星期幾。

建立範例文件

```
db.weather.insertMany([
```

```
  {
    "temperature": 97.5,
    "humidity": 0.60,
    "date": new Date("2023-04-01")
  },
  {
    "temperature": 95.2,
    "humidity": 0.55,
    "date": new Date("2023-04-02")
  },
  {
    "temperature": 92.8,
    "humidity": 0.65,
    "date": new Date("2023-04-03")
  }
]);
```

查詢範例

```
db.weather.aggregate([
  {
    $project: {
      dayOfWeek: { $dayOfWeek: "$date" }
    }
  }
]).pretty();
```

輸出

```
{ "_id" : ObjectId("64272c6663f4f8ce422c2d91"), "dayOfWeek" : 7 }
{ "_id" : ObjectId("64272c6663f4f8ce422c2d92"), "dayOfWeek" : 1 }
{ "_id" : ObjectId("64272c6663f4f8ce422c2d93"), "dayOfWeek" : 2 }
```

程式碼範例

若要檢視使用 $dayOfWeek命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('weather');

  const result = await collection.aggregate([
    {
      $project: {
        dayOfWeek: { $dayOfWeek: '$date' }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.weather

    result = list(collection.aggregate([
        {
            '$project': {
                'dayOfWeek': { '$dayOfWeek': '$date' }
            }
        }
    ]))

    print(result)

    client.close()
```

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

$dayOfYear

Amazon DocumentDB 中的$dayOfYear運算子會以介於 1 到 366 之間的數字 （非閏年為 365) 傳回日期的年份日期。

參數

- expression：日期欄位或表達式，從中擷取一年中的某一天。

範例 (MongoDB Shell)

此範例示範如何使用 $dayOfYear 運算子，從 Amazon DocumentDB 集合中的日期欄位中擷取一年中的某一天。

建立範例文件

```
db.weather.insert([
  {
    "temperature" : 97.5,
    "humidity": 0.60,
    "date" : new Date("2023-03-15")
  },
  {
    "temperature" : 82.3,
    "humidity": 0.75,
    "date" : new Date("2023-12-31")
  }
])
```

查詢範例

```
db.weather.aggregate([
  {
    $project: {
      dayOfYear: { $dayOfYear: "$date" }
    }
  }
```

```
]).pretty()
```

輸出

```
{ "_id" : ObjectId("642b86fc7d8e07af279bbe63"), "dayOfYear" : 74 }
{ "_id" : ObjectId("642b86fc7d8e07af279bbe64"), "dayOfYear" : 365 }
```

程式碼範例

若要檢視使用 $dayOfYear命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('weather');

  const result = await collection.aggregate([
    {
      $project: {
        dayOfYear: { $dayOfYear: "$date" }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['weather']

    result = list(collection.aggregate([
        {
            '$project': {
                'dayOfYear': { '$dayOfYear': '$date' }
            }
        }
    ]))

    print(result)

    client.close()

example()
```

$degreesToRadians

8.0.1 版的新功能。

Amazon DocumentDB 中的$degreesToRadians運算子會將值從度數轉換為弧度。

參數

- `expression`：以度為單位解析為數值的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $degreesToRadians運算子將度數值轉換為弧度。

建立範例文件

```
db.angles.insertMany([
  { "_id": 1, "degrees": 0 },
  { "_id": 2, "degrees": 90 },
  { "_id": 3, "degrees": 180 },
  { "_id": 4, "degrees": 360 }
]);
```

查詢範例

```
db.angles.aggregate([
  { $project: {
    "radians": { $degreesToRadians: "$degrees" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "radians": 0 },
  { "_id": 2, "radians": 1.5707963267948966 },
  { "_id": 3, "radians": 3.141592653589793 },
  { "_id": 4, "radians": 6.283185307179586 }
]
```

程式碼範例

若要檢視使用 $degreesToRadians 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('angles');

  const result = await collection.aggregate([
    { $project: {
      "radians": { $degreesToRadians: "$degrees" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}
```

```
main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['angles']

    result = list(collection.aggregate([
        { '$project': {
            'radians': { '$degreesToRadians': '$degrees' }
        }}
    ]))

    print(result)
    client.close()

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

$divide

Amazon DocumentDB 彙總管道中的$divide運算子用於將一個數字除以另一個數字。這是一個實用的運算子，可對文件中的數值欄位執行數學操作。

參數

- numerator：要分割的股利或數字。
- denominator：要除以的除數或數字。

範例 (MongoDB Shell)

此範例示範如何使用 $divide 運算子，根據員工的年薪和每年工作時數來計算員工的每小時費率。

建立範例文件

```
db.employees.insertMany([
  { _id: 1, name: "John Doe", salary: 80000, hoursPerYear: 2080 },
  { _id: 2, name: "Jane Smith", salary: 90000, hoursPerYear: 2080 },
  { _id: 3, name: "Bob Johnson", salary: 75000, hoursPerYear: 2080 }
]);
```

查詢範例

```
db.employees.aggregate([
  {
    $project: {
      name: 1,
      hourlyRate: { $divide: ["$salary", "$hoursPerYear"] }
    }
  }
])
```

輸出

```
[
  { "_id" : 1, "name" : "John Doe", "hourlyRate" : 38.46153846153846 },
  { "_id" : 2, "name" : "Jane Smith", "hourlyRate" : 43.26923076923077 },
  { "_id" : 3, "name" : "Bob Johnson", "hourlyRate" : 36.05769230769231 }
]
```

程式碼範例

若要檢視使用 $divide命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function calculateHourlyRate() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const employees = db.collection('employees');

  const result = await employees.aggregate([
```

```
    {
      $project: {
        name: 1,
        hourlyRate: { $divide: ["$salary", "$hoursPerYear"] }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

calculateHourlyRate();
```

Python

```
from pymongo import MongoClient

def calculate_hourly_rate():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.mydatabase
    employees = db.employees

    result = list(employees.aggregate([
        {
            '$project': {
                'name': 1,
                'hourlyRate': { '$divide': ['$salary', '$hoursPerYear'] }
            }
        }
    ]))

    print(result)
    client.close()

calculate_hourly_rate()
```

$eq

$eq 彙總運算子會比較兩個值，並在它們相等true時傳回，否則傳回 false。

參數

- expression1：要比較的第一個值。
- expression2：要比較的第二個值。

範例 (MongoDB Shell)

下列範例示範如何使用 $eq 運算子來檢查產品數量是否符合目標值。

建立範例文件

```
db.inventory.insertMany([
  { _id: 1, item: "Widget", qty: 50, target: 50 },
  { _id: 2, item: "Gadget", qty: 30, target: 50 },
  { _id: 3, item: "Tool", qty: 50, target: 40 }
]);
```

查詢範例

```
db.inventory.aggregate([
  {
    $project: {
      item: 1,
      qty: 1,
      target: 1,
      meetsTarget: { $eq: ["$qty", "$target"] }
    }
  }
]);
```

輸出

```
[
  { _id: 1, item: 'Widget', qty: 50, target: 50, meetsTarget: true },
  { _id: 2, item: 'Gadget', qty: 30, target: 50, meetsTarget: false },
  { _id: 3, item: 'Tool', qty: 50, target: 40, meetsTarget: false }
]
```

程式碼範例

若要檢視使用$eq彙總運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('inventory');

  const result = await collection.aggregate([
    {
      $project: {
        item: 1,
        qty: 1,
        target: 1,
        meetsTarget: { $eq: ["$qty", "$target"] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['inventory']

    result = list(collection.aggregate([
        {
            '$project': {
                'item': 1,
```

```
                'qty': 1,
                'target': 1,
                'meetsTarget': { '$eq': ['$qty', '$target'] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$exp

4.0 版的新功能

Amazon DocumentDB 中的$exp運算子可讓您將常數 e 提高到指定的數字。

參數

- expression：要評估的表達式。這可以是任何有效的彙總表達式，包括欄位參考、算術運算和其他彙總階段。

範例 (MongoDB Shell)

下列範例示範如何使用 $exp運算子來尋找 quantity 欄位大於 price 欄位的所有文件。

建立範例文件

```
db.items.insertMany([
  { item: "canvas", quantity: 4 },
  { item: "journal", quantity: 2 }
]);
```

查詢範例

```
db.items.aggregate([
  { $project: {
    "quantityRaised": {$exp: "$quantity"}}
  }
]);
```

輸出

```
[
  {
    _id: ObjectId('6920b785311cf98b79d2950d'),
    quantityRaised: 54.598150033144236
  },
  {
    _id: ObjectId('6920b785311cf98b79d2950e'),
    quantityRaised: 7.38905609893065
  }
]
```

程式碼範例

若要檢視使用 $exp命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function aggregateExp() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const items = db.collection('items');

  const result = await items.aggregate([
    { $project: {
      "quantityRaised": {$exp: "$quantity"}}
    }
    ]).toArray();

  console.log(result);
  client.close();
}

aggregateExp();
```

Python

```
from pymongo import MongoClient
```

```
def aggregate_exp():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    items = db.items

    result = list(items.aggregate([
      { "$project": {
        "quantityRaised": {"$exp": "$quantity"}}
      }
    ]))

    print(result)
    client.close()

aggregate_exp()
```

$filter

Amazon DocumentDB 中的$filter運算子用於將篩選條件表達式套用至陣列的每個元素，並傳回僅包含符合指定條件之元素的陣列。當您需要對文件中的陣列欄位執行複雜的篩選操作時，此運算子非常有用。

參數

- input：要篩選的陣列欄位。
- as：用於cond表達式內input陣列每個元素的變數名稱。
- cond：布林值表達式，決定指定的元素是否應包含在輸出陣列中。

範例 (MongoDB Shell)

下列範例示範如何使用 $filter運算子來投影每個訂單的客戶，並建立新的陣列欄位 paidItems，只包含價格大於 15 的項目陣列中的項目。基本上，它會篩選每個訂單的項目，只包含成本超過 15 的產品。

建立範例文件

```
db.orders.insertMany([
```

```
  { _id: 1, customer: "abc123", items: [
    { name: "Product A", price: 10, qty: 2 },
    { name: "Product B", price: 20, qty: 1 }
  ]},
  { _id: 2, customer: "def456", items: [
    { name: "Product C", price: 5, qty: 3 },
    { name: "Product D", price: 15, qty: 4 }
  ]},
  { _id: 3, customer: "ghi789", items: [
    { name: "Product E", price: 8, qty: 3 },
    { name: "Product F", price: 12, qty: 1 }
  ]}
]);
```

查詢範例

```
db.orders.aggregate([
  {
    $project: {
      customer: 1,
      paidItems: {
        $filter: {
          input: "$items",
          as: "item",
          cond: { $gt: ["$$item.price", 15] }
        }
      }
    }
  }
]).pretty();
```

輸出

```
[
  {
    _id: 1,
    customer: 'abc123',
    paidItems: [ { name: 'Product B', price: 20, qty: 1 } ]
  },
  { _id: 2, customer: 'def456', paidItems: [] },
  { _id: 3, customer: 'ghi789', paidItems: [] }
]
```

程式碼範例

若要檢視使用 $filter命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const db = client.db('test');
    const collection = db.collection('orders');

    const result = await collection.aggregate([
      {
        $project: {
          customer: 1,
          paidItems: {
            $filter: {
              input: "$items",
              as: "item",
              cond: { $gt: ["$$item.price", 15] }
            }
          }
        }
      }
    ]).toArray();

    console.log(JSON.stringify(result, null, 2));

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
}
```

```
example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false'

    with MongoClient(uri) as client:
        db = client.test
        collection = db.orders

        result = list(collection.aggregate([
            {
                '$project': {
                    'customer': 1,
                    'paidItems': {
                        '$filter': {
                            'input': '$items',
                            'as': 'item',
                            'cond': { '$gt': ['$$item.price', 15] }
                        }
                    }
                }
            }
        ]))

        for doc in result:
            pprint(doc)

example()
```

$first

5.0 版的新功能。

Elastic 叢集不支援。

Amazon DocumentDB 中的$first運算子會從一組分組的文件中傳回第一個文件。它通常用於彙總管道，以擷取符合特定條件的第一個文件。

參數

- expression：傳回為每個群組中第一個值的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $first運算子來擷取彙總期間針對每個類別遇到的第一個項目值。

注意： 會根據管道中文件的目前順序$first傳回第一個文件。為了確保特定訂單（例如，依日期、價格等），階段$sort應該在$group階段之前使用。

建立範例文件

```
db.products.insertMany([
  { _id: 1, item: "abc", price: 10, category: "food" },
  { _id: 2, item: "jkl", price: 20, category: "food" },
  { _id: 3, item: "xyz", price: 5, category: "toy" },
  { _id: 4, item: "abc", price: 5, category: "toy" }
]);
```

查詢範例

```
db.products.aggregate([
  { $group: { _id: "$category", firstItem: { $first: "$item" } } }
]);
```

輸出

```
[
  { "_id" : "food", "firstItem" : "abc" },
  { "_id" : "toy", "firstItem" : "xyz" }
]
```

程式碼範例

若要檢視使用 $first命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const db = client.db('test');
    const collection = db.collection('products');

    const result = await collection.aggregate([
      { $group: { _id: "$category", firstItem: { $first: "$item" } } }
    ]).toArray();

    console.log(result);

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
        db = client['test']
        collection = db['products']

        result = list(collection.aggregate([
            { '$group': { '_id': '$category', 'firstItem': { '$first': '$item' } } }
        ]))

        pprint(result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$firstN

8.0.1 版的新功能。

Amazon DocumentDB 中的$firstN運算子會傳回前 N 個元素。在$group階段中用作累積器時，它會傳回每個群組中第一個 N 值的陣列。用作陣列表達式運算子時，它會傳回陣列的第一個 N 元素。

參數

- input：解析為要傳回值之欄位或陣列的表達式。
- n：正整數，指定要傳回多少值。在$group階段中用作累積器時，n也可以是表達式，只要它根據群組_id欄位解析為正整數即可。

範例 (MongoDB Shell)

下列範例顯示如何在彙總期間使用$firstN累積器擷取每個項目的前兩個數量。

> **Note**
>
> $firstN 會依文件到達$group階段的順序選取值。若要傳回特定排序的前 N 個值 （ 例如，依日期或分數 ），請在 之前新增$sort階段$group。

建立範例文件

```
db.sales.insertMany([
  { item: "abc", quantity: 10, date: ISODate("2023-01-01") },
  { item: "abc", quantity: 5, date: ISODate("2023-01-02") },
  { item: "abc", quantity: 8, date: ISODate("2023-01-03") },
  { item: "xyz", quantity: 15, date: ISODate("2023-01-01") },
  { item: "xyz", quantity: 7, date: ISODate("2023-01-02") },
  { item: "xyz", quantity: 3, date: ISODate("2023-01-03") }
]);
```

查詢範例

```
db.sales.aggregate([
  { $group: { _id: "$item", firstTwoQuantities: { $firstN: { input: "$quantity", n:
 2 } } } }
]);
```

輸出

```
[
  { "_id": "abc", "firstTwoQuantities": [10, 5] },
  { "_id": "xyz", "firstTwoQuantities": [15, 7] }
]
```

表達式用量範例 (MongoDB Shell)

運算$firstN子也可以用作$project階段內的表達式，以傳回陣列欄位的第一個 N 元素。

建立範例文件

```
db.inventory.insertMany([
  { _id: 1, item: "abc", tags: ["red", "green", "blue", "yellow", "purple"] },
  { _id: 2, item: "xyz", tags: ["alpha", "beta", "gamma"] }
]);
```

查詢範例

```
db.inventory.aggregate([
  { $project: {
      firstThreeTags: { $firstN: { input: "$tags", n: 3 } }
    }}
```

```
]);
```

輸出

```
[
  { "_id": 1, "firstThreeTags": ["red", "green", "blue"] },
  { "_id": 2, "firstThreeTags": ["alpha", "beta", "gamma"] }
]
```

程式碼範例

若要檢視使用$firstN累積器的程式碼範例，請選擇您要使用的語言標籤。下列範例顯示累積器用量（在 中$group) 和表達式用量（在 中$project)：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');

    // Accumulator usage: first N values per group
    const sales = db.collection('sales');
    const accumulatorResult = await sales.aggregate([
      { $group: { _id: "$item", firstTwoQuantities: { $firstN: { input: "$quantity",
 n: 2 } } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Expression usage: first N elements of an array field
    const inventory = db.collection('inventory');
    const expressionResult = await inventory.aggregate([
      { $project: { firstThreeTags: { $firstN: { input: "$tags", n: 3 } } } }
    ]).toArray();
    console.log('Expression result:', expressionResult);

  } finally {
```

```
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']

        # Accumulator usage: first N values per group
        sales = db['sales']
        accumulator_result = list(sales.aggregate([
            { '$group': { '_id': '$item', 'firstTwoQuantities': { '$firstN':
 { 'input': '$quantity', 'n': 2 } } } }
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: first N elements of an array field
        inventory = db['inventory']
        expression_result = list(inventory.aggregate([
            { '$project': { 'firstThreeTags': { '$firstN': { 'input': '$tags', 'n':
 3 } } } }
        ]))
        print('Expression result:', expression_result)

    finally:
        client.close()

example()
```

$floor

4.0 版的新功能。

Amazon DocumentDB 中的$floor運算子會傳回小於或等於指定數字的最大整數。此運算子適用於將數值四捨五入。

參數

- expression：要四捨五入的數值表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $floor運算子將小數值四捨五入到最接近的整數。

建立範例文件

```
db.numbers.insertOne({ value: 3.14 });
```

查詢範例

```
db.numbers.aggregate([
  { $project: { _id: 0, floored: { $floor: "$value" } } }
]);
```

輸出

```
{ "floored" : 3 }
```

程式碼範例

若要檢視使用 $floor命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();
```

```
    const db = client.db('test');
    const collection = db.collection('numbers');

    const result = await collection.aggregate([
      { $project: { _id: 0, floored: { $floor: "$value" } } }
    ]).toArray();

    console.log(result);

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client.test
        collection = db.numbers

        result = list(collection.aggregate([
            { '$project': { '_id': 0, 'floored': { '$floor': '$value' }}}
        ]))

        pprint(result)

    except Exception as e:
        print(f"An error occurred: {e}")
```

```
    finally:
        if client:
            client.close()

example()
```

$geoNear

$geoNear 彙總階段會依接近指定點的順序傳回文件。它會計算與點的距離，並在輸出文件中包含距離。

參數

- near：計算距離的點，指定為 GeoJSON 或舊版座標。
- distanceField：存放計算距離的欄位名稱。
- spherical：布林值，指出是否使用球形幾何 (GeoJSON 點需要) 。
- maxDistance：選用。與中心點的距離上限。
- minDistance：選用。與中心點的最小距離。
- query：選用。要套用的其他篩選條件。
- limit：選用。要傳回的文件數量上限。
- key：選用。當存在多個地理空間索引時，用於地理空間查詢的欄位。

範例 (MongoDB Shell)

下列範例示範如何使用 $geoNear階段尋找最接近指定位置的商店。

建立範例文件

```
db.stores.createIndex({ location: "2dsphere" });

db.stores.insertMany([
  { _id: 1, name: "Store A", location: { type: "Point", coordinates: [-122.4,
 37.8] } },
  { _id: 2, name: "Store B", location: { type: "Point", coordinates: [-122.5,
 37.7] } },
  { _id: 3, name: "Store C", location: { type: "Point", coordinates: [-122.3, 37.9] } }
]);
```

查詢範例

```
db.stores.aggregate([
  {
    $geoNear: {
      near: { type: "Point", coordinates: [-122.4, 37.8] },
      distanceField: "distance",
      spherical: true
    }
  }
]);
```

輸出

```
[
  { _id: 1, name: 'Store A', location: { type: 'Point', coordinates: [ -122.4,
 37.8 ] }, distance: 0 },
  { _id: 3, name: 'Store C', location: { type: 'Point', coordinates: [ -122.3,
 37.9 ] }, distance: 13877.82 },
  { _id: 2, name: 'Store B', location: { type: 'Point', coordinates: [ -122.5,
 37.7 ] }, distance: 15557.89 }
]
```

程式碼範例

若要檢視使用$geoNear彙總階段的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('stores');

  const result = await collection.aggregate([
    {
      $geoNear: {
        near: { type: "Point", coordinates: [-122.4, 37.8] },
```

```
        distanceField: "distance",
        spherical: true
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['stores']

    result = list(collection.aggregate([
        {
            '$geoNear': {
                'near': { 'type': 'Point', 'coordinates': [-122.4, 37.8] },
                'distanceField': 'distance',
                'spherical': True
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$group

Amazon DocumentDB $group 中的彙總階段可讓您依指定的表達式將文件分組，並對分組的資料執行各種累積操作。這對於根據分組資料計算總計、平均值或其他統計資料等任務很有用。

參數

- _id：指定輸入文件應分組的表達式。這可以是欄位名稱、運算表達式或兩者的組合。
- accumulator expressions：（選用）一或多個應套用至分組資料的累積器表達式。這些表達式使用上述的累積器運算子。

範例 (MongoDB Shell)

下列範例依客戶的城市分組，並計算每個城市的總訂單金額。

建立範例文件

```
db.customers.insertMany([
  { name: "John Doe", city: "New York", orders: [{ amount: 100 }, { amount: 200 }] },
  { name: "Jane Smith", city: "Los Angeles", orders: [{ amount: 150 }, { amount:
 300 }] },
  { name: "Bob Johnson", city: "New York", orders: [{ amount: 75 }, { amount: 125 }] },
  { name: "Samantha Lee", city: "Chicago", orders: [{ amount: 50 }, { amount: 100 }] }
]);
```

查詢範例

```
db.customers.aggregate([
  {
    $group: {
      _id: "$city",
      totalOrders: { $sum: { $sum: "$orders.amount" } }
    }
  }
]);
```

輸出

```
[
  { _id: 'Chicago', totalOrders: 150 },
  { _id: 'Los Angeles', totalOrders: 450 },
  { _id: 'New York', totalOrders: 500 }
]
```

程式碼範例

若要檢視使用 $group命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function groupByCity() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();
    const db = client.db('test');
    const result = await db.collection('customers').aggregate([
      { $unwind: '$orders' },
      {
        $group: {
          _id: '$city',
          totalOrders: { $sum: '$orders.amount' }
        }
      }
    ]).toArray();

    console.log(result);
  } catch (err) {
    console.error('Error during aggregation:', err);
  } finally {
    await client.close();
  }
}

groupByCity();
```

Python

```
from pymongo import MongoClient

def group_by_city():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client.test
        result = list(db.customers.aggregate([
            {'$unwind': '$orders'},
            {
                '$group': {
                    '_id': '$city',
                    'totalOrders': {'$sum': '$orders.amount'}
                }
            }
        ]))
        print(result)
    except Exception as e:
        print(f"Error during aggregation: {e}")
    finally:
        client.close()

group_by_city()
```

$gt

$gt 彙總運算子會比較兩個值，如果第一個值大於第二個值true，則傳回 ，否則傳回 false。

參數

- expression1：要比較的第一個值。
- expression2：要比較的第二個值。

範例 (MongoDB Shell)

下列範例示範如何使用 $gt 運算子來識別超過價格閾值的產品。

建立範例文件

```
db.products.insertMany([
  { _id: 1, name: "Laptop", price: 1200 },
  { _id: 2, name: "Mouse", price: 25 },
```

```
  { _id: 3, name: "Monitor", price: 400 }
]);
```

查詢範例

```
db.products.aggregate([
  {
    $project: {
      name: 1,
      price: 1,
      expensive: { $gt: ["$price", 100] }
    }
  }
]);
```

輸出

```
[
  { _id: 1, name: 'Laptop', price: 1200, expensive: true },
  { _id: 2, name: 'Mouse', price: 25, expensive: false },
  { _id: 3, name: 'Monitor', price: 400, expensive: true }
]
```

程式碼範例

若要檢視使用$gt彙總運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
```

```
        price: 1,
        expensive: { $gt: ["$price", 100] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['products']

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'price': 1,
                'expensive': { '$gt': ['$price', 100] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$gte

$gte 彙總運算子會比較兩個值，如果第一個值大於或等於第二個值，true則傳回 ，否則傳回 false。

參數

- expression1：要比較的第一個值。
- expression2：要比較的第二個值。

範例 (MongoDB Shell)

下列範例示範如何使用 $gte 運算子來檢查學生是否已通過檢查。

建立範例文件

```
db.students.insertMany([
  { _id: 1, name: "Alice", score: 85 },
  { _id: 2, name: "Bob", score: 60 },
  { _id: 3, name: "Charlie", score: 72 }
]);
```

查詢範例

```
db.students.aggregate([
  {
    $project: {
      name: 1,
      score: 1,
      passed: { $gte: ["$score", 70] }
    }
  }
]);
```

輸出

```
[
  { _id: 1, name: 'Alice', score: 85, passed: true },
  { _id: 2, name: 'Bob', score: 60, passed: false },
  { _id: 3, name: 'Charlie', score: 72, passed: true }
]
```

程式碼範例

若要檢視使用$gte彙總運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('students');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
        score: 1,
        passed: { $gte: ["$score", 70] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['students']

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'score': 1,
```

```
                'passed': { '$gte': ['$score', 70] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$hour

$hour 運算子會從日期或時間戳記欄位中擷取小時元件。

參數

- dateExpression：套用運算子的日期。這必須解析為有效的 BSON 日期 （例如，$createdAt 之類的欄位或日期常值）。

參數也可以指定為下列格式的文件：

{ 日期：<dateExpression>，時區：<timezoneExpression>}

這可讓 套用時區感知日期操作。

```
- `<tzExpression>`: (optional) The timezone of the operation result. It must be a valid
 expression that resolves to a string formatted as either an Olson Timezone Identifier
 or a UTC Offset. If no timezone is provided, the result is in UTC.
```

範例 (MongoDB Shell)

下列範例示範如何使用 $hour 運算子從日期欄位中擷取小時元件，並相應地將資料分組。

建立範例文件

```
db.events.insertMany([
  { timestamp: new Date("2023-04-01T10:30:00Z") },
  { timestamp: new Date("2023-04-01T12:45:00Z") },
  { timestamp: new Date("2023-04-02T08:15:00Z") },
  { timestamp: new Date("2023-04-02T16:20:00Z") },
  { timestamp: new Date("2023-04-03T23:59:00Z") }
```

```
]);
```

查詢範例

```
db.events.aggregate([
  {
    $project: {
      hour: { $hour: "$timestamp" }
    }
  },
  {
    $group: {
      _id: "$hour",
      count: { $sum: 1 }
    }
  },
  {
    $sort: { _id: 1 }
  }
]);
```

輸出

```
[
  { "_id": 8, "count": 1 },
  { "_id": 10, "count": 1 },
  { "_id": 12, "count": 1 },
  { "_id": 16, "count": 1 },
  { "_id": 23, "count": 1 }
]
```

此查詢會依timestamp欄位的小時元件將事件分組，並計算每小時的事件數。

程式碼範例

若要檢視使用 $hour命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('events');

    const result = await collection.aggregate([
      {
        $project: {
          hour: { $hour: "$timestamp" }
        }
      },
      {
        $group: {
          _id: "$hour",
          count: { $sum: 1 }
        }
      },
      {
        $sort: { _id: 1 }
      }
    ]).toArray();

    console.log(result);
  } catch (error) {
    console.error('Error occurred:', error);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from datetime import datetime

def example():
    try:
```

```
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client.test
        collection = db.events

        result = list(collection.aggregate([
            {
                "$project": {
                    "hour": {"$hour": "$timestamp"}
                }
            },
            {
                "$group": {
                    "_id": "$hour",
                    "count": {"$sum": 1}
                }
            },
            {
                "$sort": {"_id": 1}
            }
        ]))

        print(result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        client.close()

example()
```

$ifNull

如果輸入表達式評估為 null 或未定義，則$ifNull運算子會用來傳回指定的值。此運算子在您想要提供預設值或處理 null/未定義案例的情況下非常有用。

參數

- expression：要評估的表達式。

- replacement：如果 <expression>評估為 null 或未定義，要傳回的值。

範例 (MongoDB Shell)

下列範例示範當 name 欄位為 null 或未定義時， $ifNull運算子的使用情況可提供預設值。

建立範例文件

```
db.users.insertMany([
  { _id: 1, name: "John" },
  { _id: 2, name: null },
  { _id: 3 }
]);
```

查詢範例

```
db.users.aggregate([
  {
    $project: {
      _id: 1,
      name: { $ifNull: ["$name", "No Name"] }
    }
  }
]);
```

輸出

```
[
  { "_id": 1, "name": "John" },
  { "_id": 2, "name": "No Name" },
  { "_id": 3, "name": "No Name" }
]
```

程式碼範例

若要檢視使用 $ifNull命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  const db = client.db('test');

  const collection = db.collection('users');

  const pipeline = [
    {
      $project: {
        _id: 1,
        name: { $ifNull: ["$name", "No Name"] }
      }
    }
  ];

  const cursor = await collection.aggregate(pipeline);

  await cursor.forEach(doc => {
    console.log(doc);
  });

  await client.close();

}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    db = client.test
    collection = db.users

    pipeline = [
```

```
        {
            "$project": {
                "_id": 1,
                "name": { "$ifNull": ["$name", "No Name"] }
            }
        }
    ]

    result = collection.aggregate(pipeline)

    for doc in result:
        print(doc)

    client.close()

example()
```

$in

$in 彙總運算子會檢查陣列中是否存在指定的值。true 如果在陣列中找到值，則傳回 ，false否則傳回 。

參數

- value：要搜尋的值。
- array：要搜尋的陣列。

範例 (MongoDB Shell)

下列範例示範如何使用 $in 運算子來檢查每個員工技能集中是否存在特定技能。

建立範例文件

```
db.employees.insertMany([
  { _id: 1, name: "Sarah", skills: ["Python", "JavaScript", "SQL"] },
  { _id: 2, name: "Mike", skills: ["Java", "C++", "Go"] },
  { _id: 3, name: "Emma", skills: ["Python", "Ruby", "Rust"] }
]);
```

查詢範例

```
db.employees.aggregate([
  {
    $project: {
      name: 1,
      hasPython: { $in: ["Python", "$skills"] }
    }
  }
]);
```

輸出

```
[
  { _id: 1, name: 'Sarah', hasPython: true },
  { _id: 2, name: 'Mike', hasPython: false },
  { _id: 3, name: 'Emma', hasPython: true }
]
```

程式碼範例

若要檢視使用$in彙總運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('employees');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
        hasPython: { $in: ["Python", "$skills"] }
      }
    }
  ]).toArray();

  console.log(result);
```

```
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['employees']

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'hasPython': { '$in': ['Python', '$skills'] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$indexOfArray

Amazon DocumentDB 中的$indexOfArray運算子用於尋找陣列中第一次出現指定元素的索引。此運算子會傳回符合指定值之陣列中第一個元素的以零為基礎的索引位置。如果找不到值，則會傳回 -1。

參數

- array：要搜尋的陣列。
- value：要在陣列中搜尋的值。
- start：（選用）陣列中開始搜尋的位置。預設值為 0。

範例 (MongoDB Shell)

下列範例示範如何使用 $indexOfArray 運算子，在每個文件的「果」陣列中尋找元素「mango」第一次出現的索引。

建立範例文件

```
db.collection.insertMany([
  { _id: 1, fruits: ["apple", "banana", "cherry", "durian"] },
  { _id: 2, fruits: ["mango", "orange", "pineapple"] },
  { _id: 3, fruits: ["kiwi", "lemon", "mango"] }
]);
```

查詢範例

```
db.collection.aggregate([
  {
    $project: {
      _id: 1,
      fruitIndex: { $indexOfArray: ["$fruits", "mango"] }
    }
  }
]);
```

輸出

```
{ "_id" : 1, "fruitIndex" : 1 }
{ "_id" : 2, "fruitIndex" : 0 }
{ "_id" : 3, "fruitIndex" : 2 }
```

程式碼範例

若要檢視使用 $indexOfArray命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
```

```
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        fruitIndex: { $indexOfArray: ["$fruits", "mango"] }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['collection']

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'fruitIndex': { '$indexOfArray': ["$fruits", "mango"] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$indexOfBytes

Amazon DocumentDB 中的 $indexOfBytes 運算子用於根據字元的位元組位置，在字串中尋找子字串的起始索引。這在處理可能包含多位元組字元的文字資料時非常有用，例如在非 Latin 指令碼中找到的文字資料。

參數

- `string`：要搜尋的輸入字串。
- `substring`：要在輸入字串中搜尋的子字串。
- `[<start>]`：（選用）搜尋的開始位置（以零為基礎）。如果未指定，搜尋會從字串開頭開始。

範例 (MongoDB Shell)

下列範例示範如何使用 在代表桌面位置的一組字串中`$indexOfBytes`尋找第一個連字號字元的索引。

建立範例文件

```
db.people.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-BVV-021" },
  { "_id": 2, "Desk": "Munich-HGG-32a" },
  { "_id": 3, "Desk": "Cologne-ayu-892.50" },
  { "_id": 4, "Desk": "Dortmund-Hop-78" }
]);
```

查詢範例

```
db.people.aggregate([
  { $project: { stateLocation: { $indexOfBytes: ["$Desk", "-"] } } }
]);
```

輸出

```
{ "_id" : 1, "stateLocation" : 11 }
{ "_id" : 2, "stateLocation" : 6 }
{ "_id" : 3, "stateLocation" : 7 }
{ "_id" : 4, "stateLocation" : 8 }
```

程式碼範例

若要檢視使用 $indexOfBytes命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const events = db.collection('people');

  const result = await db.collection('people').aggregate([
    { $project: { stateLocation: { $indexOfBytes: ["$Desk", "-"] } } }
  ]).toArray();
  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['people']

    result = list(db.people.aggregate([
        { '$project': { 'stateLocation': { '$indexOfBytes': ['$Desk', '-'] } } }
    ]))
    print(result)
    client.close()

example()
```

$indexOfCP

Amazon DocumentDB 中的$indexOfCP運算子用於尋找字串表達式中第一次出現指定子字串的索引，以程式碼點 (CP) 為單位。這在剖析和擷取字串欄位的內容時非常有用。

參數

- `string expression`：要搜尋的字串。
- `substring`：要搜尋的子字串。
- `[<start>]`：（選用）開始搜尋的位置（以零為基礎的索引）。預設值為 0。

範例 (MongoDB Shell)

在此範例中，我們使用 $indexOfCP運算子在每個文件的桌面欄位中尋找第一個連字號字元的索引。

建立範例文件

```
db.people.insertMany([
  { "_id":1, "name":"John Doe", "Manager":"Jane Doe", "Role":"Developer", "Desk":
 "Düsseldorf-BVV-021"},
  { "_id":2, "name":"John Stiles", "Manager":"Jane Doe", "Role":"Manager", "Desk":
 "Munich-HGG-32a"},
  { "_id":3, "name":"Richard Roe", "Manager":"Jorge Souza", "Role":"Product", "Desk":
 "Cologne-ayu-892.50"},
  { "_id":4, "name":"Mary Major", "Manager":"Jane Doe", "Role":"Solution Architect",
 "Desk": "Dortmund-Hop-78"}
])
```

查詢範例

```
db.people.aggregate([
  { $project: { stateLocation: { $indexOfCP: [ "$Desk", "-"] } } }
])
```

輸出

```
{ "_id" : 1, "stateLocation" : 10 }
{ "_id" : 2, "stateLocation" : 6 }
{ "_id" : 3, "stateLocation" : 7 }
```

```
{ "_id" : 4, "stateLocation" : 8 }
```

程式碼範例

若要檢視使用 $indexOfCP命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const result = await collection.aggregate([
    { $project: { stateLocation: { $indexOfCP: [ "$Desk", "-"] } } }
  ]).toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['people']

result = list(collection.aggregate([
    { '$project': { 'stateLocation': { '$indexOfCP': [ '$Desk', '-' ] } } }
]))

print(result)
```

```
client.close()
```

$indexStats

Amazon DocumentDB $indexStats 中的彙總階段可讓您深入了解集合中索引的使用情況。此運算子可讓您監控索引的存取模式，這可協助您在索引管理和最佳化方面做出明智的決策。

參數

無

範例 (MongoDB Shell)

下列範例示範如何使用 $indexStats運算子來分析 Amazon DocumentDB 集合中的索引用量。

建立範例文件

```
db.grocery.insertMany([
  { _id: 1, product: "milk", quantity: 10 },
  { _id: 2, product: "eggs", quantity: 20 },
  { _id: 3, product: "bread", quantity: 5 },
  { _id: 4, product: "cheese", quantity: 15 },
  { _id: 5, product: "apple", quantity: 8 }
]);
```

查詢範例

```
db.grocery.aggregate([
  { $indexStats: {} }
]);
```

輸出

```
[
  {
    "name": "_id_",
    "key": {
      "_id": 1
    },
```

```
    "host": "docdb-cluster-1.cluster-123456789.us-west-2.docdb.amazonaws.com",
    "accesses": {
      "ops": NumberLong(5),
      "since": ISODate("2023-04-06T12:34:56.789Z")
    }
  },
  {
    "name": "product_1",
    "key": {
      "product": 1
    },
    "host": "docdb-cluster-1.cluster-123456789.us-west-2.docdb.amazonaws.com",
    "accesses": {
      "ops": NumberLong(10),
      "since": ISODate("2023-04-06T12:34:56.789Z")
    }
  }
]
```

在此範例中，$indexStats運算子顯示_id_自上次重設或伺服器重新啟動後，索引已存取 5 次，且product_1索引已存取 10 次。

程式碼範例

若要檢視使用 $indexStats命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function indexStats() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const result = await db.collection('grocery').aggregate([
    { $indexStats: {} }
  ]).toArray();
  console.log(result);
  await client.close();
}

indexStats();
```

Python

```
from pymongo import MongoClient

def index_stats():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    result = list(db.grocery.aggregate([
        { '$indexStats': {} }
    ]))
    print(result)
    client.close()

index_stats()
```

$isArray

Amazon DocumentDB 中的$isArray運算子用於檢查文件中的欄位是否為陣列。此運算子在彙總管道和條件表達式中非常有用，可處理陣列類型欄位。

參數

- field：要檢查是否為陣列的欄位路徑。

範例 (MongoDB Shell)

此範例示範如何使用 $isArray運算子來識別「庫存」欄位為陣列的文件。

建立範例文件

```
db.videos.insertMany([
  { "_id":1, "name":"Live Soft", "inventory": {"Des Moines": 1000, "Ames" : 500}},
  { "_id":2, "name":"Top Pilot", "inventory": {"Mason City": 250, "Des Moines": 1000}},
  { "_id":3, "name":"Romancing the Rock", "inventory": {"Mason City": 250, "Ames" :
 500}},
  { "_id":4, "name":"Bravemind", "inventory": [{"location": "Mason City", "count":
 250}, {"location": "Des Moines", "count": 1000}, {"location": "Ames", "count": 500}]}
]);
```

查詢範例

```
db.videos.aggregate([
  {
    $match: {
      $isArray: "$inventory"
    }
  },
  {
    $project: {
      _id: 1,
      name: 1,
      "inventory.location": 1,
      "inventory.count": 1
    }
  }
]).pretty();
```

輸出

```
{
  "_id": 4,
  "name": "Bravemind",
  "inventory": [
    {
      "location": "Mason City",
      "count": 250
    },
    {
      "location": "Des Moines",
      "count": 1000
    },
    {
      "location": "Ames",
      "count": 500
    }
  ]
}
```

程式碼範例

若要檢視使用 $isArray命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function run() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('videos');

  const result = await collection.aggregate([
    {
      $match: {
        $isArray: '$inventory'
      }
    },
    {
      $project: {
        _id: 1,
        name: 1,
        "inventory.location": 1,
        "inventory.count": 1
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

run();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['videos']
```

```
result = list(collection.aggregate([
    {
        '$match': {
            '$isArray': '$inventory'
        }
    },
    {
        '$project': {
            '_id': 1,
            'name': 1,
            'inventory.location': 1,
            'inventory.count': 1
        }
    }
]))

print(result)

client.close()
```

$isNumber

8.0.1 版的新功能。

Amazon DocumentDB 中的$isNumber運算子會傳回布林值，指出指定的表達式是否解析為數字類型（整數、小數、雙數、長）。請注意，即使陣列中的所有元素都是數字，陣列也不會被視為數字。例如，$isNumber套用至 值會[1, 2, 3]傳回 false。

參數

- expression：檢查其是否解析為數值類型的任何表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $isNumber 運算子來檢查欄位值是否為數值。

建立範例文件

```
db.data.insertMany([
  {_id: 1, value: 42},
  {_id: 2, value: "hello"},
```

```
  {_id: 3, value: 3.14},
  {_id: 4, value: null}
]);
```

查詢範例

```
db.data.aggregate([
  { $project: { isNum: { $isNumber: "$value" } } }
]);
```

輸出

```
[
  {_id: 1, isNum: true},
  {_id: 2, isNum: false},
  {_id: 3, isNum: true},
  {_id: 4, isNum: false}
]
```

程式碼範例

若要檢視使用 $isNumber 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('data');
    const result = await collection.aggregate([
      { $project: { isNum: { $isNumber: "$value" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
```

```
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['data']
        result = list(collection.aggregate([
            {'$project': {'isNum': {'$isNumber': '$value'}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$isoDayOfWeek

Amazon DocumentDB 中的$isoDayOfWeek運算子會將日期的一週中的 ISO 日期傳回為整數值。ISO 週日期系統定義每週從星期一開始到星期日結束，第 1 週是包含當年第一個星期四的一週。

參數

- expression：傳回一週中 ISO 日期的日期表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $isoDayOfWeek運算子來擷取一組事件文件的 ISO 星期幾。

建立範例文件

```
db.events.insertMany([
  { _id: 1, eventDate: ISODate("2023-04-01T12:00:00Z") },
  { _id: 2, eventDate: ISODate("2023-04-02T12:00:00Z") },
```

```
  { _id: 3, eventDate: ISODate("2023-04-03T12:00:00Z") },
  { _id: 4, eventDate: ISODate("2023-04-04T12:00:00Z") },
  { _id: 5, eventDate: ISODate("2023-04-05T12:00:00Z") },
  { _id: 6, eventDate: ISODate("2023-04-06T12:00:00Z") },
  { _id: 7, eventDate: ISODate("2023-04-07T12:00:00Z") }
]);
```

查詢範例

```
db.events.aggregate([
  { $project: {
    _id: 1,
    eventDate: 1,
    isoDayOfWeek: { $isoDayOfWeek: "$eventDate" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "eventDate": ISODate("2023-04-01T12:00:00Z"), "isoDayOfWeek": 6 },
  { "_id": 2, "eventDate": ISODate("2023-04-02T12:00:00Z"), "isoDayOfWeek": 7 },
  { "_id": 3, "eventDate": ISODate("2023-04-03T12:00:00Z"), "isoDayOfWeek": 1 },
  { "_id": 4, "eventDate": ISODate("2023-04-04T12:00:00Z"), "isoDayOfWeek": 2 },
  { "_id": 5, "eventDate": ISODate("2023-04-05T12:00:00Z"), "isoDayOfWeek": 3 },
  { "_id": 6, "eventDate": ISODate("2023-04-06T12:00:00Z"), "isoDayOfWeek": 4 },
  { "_id": 7, "eventDate": ISODate("2023-04-07T12:00:00Z"), "isoDayOfWeek": 5 }
]
```

程式碼範例

若要檢視使用 `$isoDayOfWeek`命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
```

```
  const events = db.collection('events');

  const result = await events.aggregate([
    {
      $project: {
        _id: 1,
        eventDate: 1,
        isoDayOfWeek: { $isoDayOfWeek: '$eventDate' }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    events = db.events

    result = list(events.aggregate([
        {
            '$project': {
                '_id': 1,
                'eventDate': 1,
                'isoDayOfWeek': { '$isoDayOfWeek': '$eventDate' }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$isoWeek

Amazon DocumentDB 中的$isoWeek運算子會傳回日期的 ISO 週數。ISO 週日期系統是一年中週數的一種方式，其中新一年的第一週是包含該年第一個星期四的一週。這與 Gregorian 行事曆不同，新年的第一週是包含 1 月 1 日的一週。

參數

無

範例 (MongoDB Shell)

下列範例示範如何使用 $isoWeek運算子來擷取指定日期的 ISO 週數。

建立範例文件

```
db.dates.insertMany([
  { _id: 1, date: new ISODate("2022-01-01") },
  { _id: 2, date: new ISODate("2022-12-31") },
  { _id: 3, date: new ISODate("2023-01-01") }
])
```

查詢範例

```
db.dates.aggregate([
  {
    $project: {
      _id: 1,
      isoWeek: { $isoWeek: "$date" }
    }
  }
])
```

輸出

```
[
  { "_id": 1, "isoWeek": 52 },
  { "_id": 2, "isoWeek": 52 },
  { "_id": 3, "isoWeek": 1 }
]
```

程式碼範例

若要檢視使用 $isoWeek命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('dates');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        isoWeek: { $isoWeek: "$date" }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['dates']

    result = list(collection.aggregate([
        {
```

```
            '$project': {
                '_id': 1,
                'isoWeek': { '$isoWeek': '$date' }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$isoWeekYear

Amazon DocumentDB 中的$isoWeekYear運算子會傳回指定日期的 ISO 8601 週年編號。ISO 週年編號與 Gregorian 曆年不同，因為該週年可能與曆年不同，尤其是在年初和年底。

參數

- expression：傳回 ISO 8601 週年編號的日期表達式。

範例 (MongoDB Shell)

此範例示範如何使用 $isoWeekYear 運算子來擷取事件集合中每個文件的日期欄位的 ISO 8601 週年。

建立範例文件

```
db.events.insertMany([
  { _id: 1, name: "Event 1", date: ISODate("2022-12-31T00:00:00Z") },
  { _id: 2, name: "Event 2", date: ISODate("2023-01-01T00:00:00Z") },
  { _id: 3, name: "Event 3", date: ISODate("2023-01-02T00:00:00Z") }
]);
```

查詢範例

```
db.events.aggregate([
  { $project: {
    name: 1,
    isoWeekYear: { $isoWeekYear: "$date" }
  }}
```

```
]);
```

輸出

```
[
  { "_id" : 1, "name" : "Event 1", "isoWeekYear" : 2023 },
  { "_id" : 2, "name" : "Event 2", "isoWeekYear" : 2023 },
  { "_id" : 3, "name" : "Event 3", "isoWeekYear" : 2023 }
]
```

程式碼範例

若要檢視使用 $isoWeekYear命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const events = db.collection('events');

  const result = await events.aggregate([
    { $project: {
      name: 1,
      isoWeekYear: { $isoWeekYear: '$date' }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    events = db.events

    result = list(events.aggregate([
        { '$project': {
            'name': 1,
            'isoWeekYear': { '$isoWeekYear': '$date' }
        }}
    ]))

    print(result)
    client.close()

example()
```

$last

Amazon DocumentDB 中的$last運算子用於傳回符合查詢條件之陣列中的最後一個元素。它特別適用於擷取陣列中符合特定條件的最新或最後一個元素。

參數

- expression：符合陣列元素的表達式。

範例 (MongoDB Shell)

下列範例示範使用 $last運算子搭配 $filter，從符合特定條件的陣列擷取最後一個元素 （例如，主體是 'science')。

建立範例文件

```
db.collection.insertMany([
  {
    "_id": 1,
    "name": "John",
    "scores": [
      { "subject": "math", "score": 82 },
      { "subject": "english", "score": 85 },
      { "subject": "science", "score": 90 }
```

```
    ]
  },
  {
    "_id": 2,
    "name": "Jane",
    "scores": [
      { "subject": "math", "score": 92 },
      { "subject": "english", "score": 88 },
      { "subject": "science", "score": 87 }
    ]
  },
  {
    "_id": 3,
    "name": "Bob",
    "scores": [
      { "subject": "math", "score": 75 },
      { "subject": "english", "score": 80 },
      { "subject": "science", "score": 85 }
    ]
  }
]);
```

查詢範例

```
db.collection.aggregate([
    { $match: { name: "John" } },
    {
      $project: {
        name: 1,
        lastScienceScore: {
          $last: {
            $filter: {
              input: "$scores",
              as: "score",
              cond: { $eq: ["$$score.subject", "science"] }
            }
          }
        }
      }
    }
  ]);
```

輸出

```
[
  {
    _id: 1,
    name: 'John',
    lastScienceScore: { subject: 'science', score: 90 }
  }
]
```

程式碼範例

若要檢視使用 $last命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.aggregate([
    { $match: { name: "John" } },
    {
      $project: {
        name: 1,
        lastScienceScore: {
          $last: {
            $filter: {
              input: "$scores",
              as: "score",
              cond: { $eq: ["$$score.subject", "science"] }
            }
          }
        }
      }
    }
  ]).toArray();

  console.log(JSON.stringify(result, null, 2));
```

```
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():

    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    db = client.test
    collection = db.collection

    pipeline = [
        { "$match": { "name": "John" } },
        {
            "$project": {
                "name": 1,
                "lastScienceScore": {
                    "$last": {
                        "$filter": {
                            "input": "$scores",
                            "as": "score",
                            "cond": { "$eq": ["$$score.subject", "science"] }
                        }
                    }
                }
            }
        }
    ]

    result = list(collection.aggregate(pipeline))

    print(result)

    client.close()

example()
```

$lastN

8.0.1 版的新功能。

Amazon DocumentDB 中的$lastN運算子會傳回最後一個 N 元素。在$group階段中用作累積器時，它會傳回每個群組中最後一個 N 值的陣列。用作陣列表達式運算子時，它會傳回陣列的最後 N 個元素。

參數

- input：解析為要傳回值之欄位或陣列的表達式。
- n：正整數，指定要傳回多少值。在$group階段中用作累積器時，n也可以是表達式，只要它根據群組_id欄位解析為正整數即可。

範例 (MongoDB Shell)

下列範例示範如何在彙總期間使用$lastN累積器擷取每個項目的最後兩個數量。

> **Note**
>
> $lastN 會依文件到達$group階段的順序選取值。若要傳回特定排序的最後一個 N 值 （ 例如，依日期或分數），請在 之前新增$sort階段$group。

建立範例文件

```
db.sales.insertMany([
  { item: "abc", quantity: 10, date: ISODate("2023-01-01") },
  { item: "abc", quantity: 5, date: ISODate("2023-01-02") },
  { item: "abc", quantity: 8, date: ISODate("2023-01-03") },
  { item: "xyz", quantity: 15, date: ISODate("2023-01-01") },
  { item: "xyz", quantity: 7, date: ISODate("2023-01-02") },
  { item: "xyz", quantity: 3, date: ISODate("2023-01-03") }
]);
```

查詢範例

```
db.sales.aggregate([
  { $group: { _id: "$item", lastTwoQuantities: { $lastN: { input: "$quantity", n:
 2 } } } }
```

```
]);
```

輸出

```
[
  { "_id": "abc", "lastTwoQuantities": [5, 8] },
  { "_id": "xyz", "lastTwoQuantities": [7, 3] }
]
```

表達式用量範例 (MongoDB Shell)

運算$lastN子也可以用作$project階段內的表達式，以傳回陣列欄位的最後 N 個元素。

建立範例文件

```
db.inventory.insertMany([
  { _id: 1, item: "abc", tags: ["red", "green", "blue", "yellow", "purple"] },
  { _id: 2, item: "xyz", tags: ["alpha", "beta", "gamma"] }
]);
```

查詢範例

```
db.inventory.aggregate([
  { $project: {
      lastThreeTags: { $lastN: { input: "$tags", n: 3 } }
    }}
]);
```

輸出

```
[
  { "_id": 1, "lastThreeTags": ["blue", "yellow", "purple"] },
  { "_id": 2, "lastThreeTags": ["alpha", "beta", "gamma"] }
]
```

程式碼範例

若要檢視使用$lastN累積器的程式碼範例，請選擇您要使用的語言標籤。下列範例顯示累積器用量（在 中$group) 和表達式用量（在 中$project)：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');

    // Accumulator usage: last N values per group
    const sales = db.collection('sales');
    const accumulatorResult = await sales.aggregate([
      { $group: { _id: "$item", lastTwoQuantities: { $lastN: { input: "$quantity",
 n: 2 } } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Expression usage: last N elements of an array field
    const inventory = db.collection('inventory');
    const expressionResult = await inventory.aggregate([
      { $project: { lastThreeTags: { $lastN: { input: "$tags", n: 3 } } } }
    ]).toArray();
    console.log('Expression result:', expressionResult);

  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
    try:
        db = client['test']

        # Accumulator usage: last N values per group
        sales = db['sales']
        accumulator_result = list(sales.aggregate([
            { '$group': { '_id': '$item', 'lastTwoQuantities': { '$lastN':
 { 'input': '$quantity', 'n': 2 } } } }
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: last N elements of an array field
        inventory = db['inventory']
        expression_result = list(inventory.aggregate([
            { '$project': { 'lastThreeTags': { '$lastN': { 'input': '$tags', 'n':
 3 } } } }
        ]))
        print('Expression result:', expression_result)

    finally:
        client.close()

example()
```

$let

Amazon DocumentDB 中的$let運算子用於將變數繫結至值，並在運算式中使用這些變數。它可讓您定義可在彙總管道相同階段內後續表達式中使用的本機變數。

參數

- vars：定義要在表達式中使用的變數的物件。
- in：使用 vars 參數中定義的變數的表達式。

範例 (MongoDB Shell)

此範例示範運算$let子用來計算矩形的區域。

建立範例文件

```
db.shapes.insertMany([
  { name: "Rectangle 1", length: 5, width: 3 },
  { name: "Rectangle 2", length: 7, width: 4 },
  { name: "Rectangle 3", length: 6, width: 2 }
]);
```

查詢範例

```
db.shapes.aggregate([
  {
    $project: {
      name: 1,
      area: {
        $let: {
          vars: {
            length: "$length",
            width: "$width"
          },
          in: {
            $multiply: ["$$length", "$$width"]
          }
        }
      }
    }
  }
])
```

輸出

```
[
  {
    "_id": ObjectId("6161e5b1a3eba3c7f2960d03"),
    "name": "Rectangle 1",
    "area": 15
  },
  {
    "_id": ObjectId("6161e5b1a3eba3c7f2960d04"),
    "name": "Rectangle 2",
    "area": 28
  },
  {
    "_id": ObjectId("6161e5b1a3eba3c7f2960d05"),
```

```
    "name": "Rectangle 3",
    "area": 12
  }
]
```

程式碼範例

若要檢視使用 $let命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function calculateRectangleAreas() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('shapes');

  const result = await shapes.aggregate([
    {
      $project: {
        name: 1,
        area: {
          $let: {
            vars: {
              length: '$length',
              width: '$width'
            },
            in: {
              $multiply: ['$$length', '$$width']
            }
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}
```

```
calculateRectangleAreas();
```

Python

```
from pymongo import MongoClient

def calculate_rectangle_areas():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.shapes

    result = list(shapes.aggregate([
        {
            '$project': {
                'name': 1,
                'area': {
                    '$let': {
                        'vars': {
                            'length': '$length',
                            'width': '$width'
                        },
                        'in': {
                            '$multiply': ['$$length', '$$width']
                        }
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

calculate_rectangle_areas()
```

$limit

Amazon DocumentDB 中的$limit運算子用於限制查詢傳回的文件數量。它類似於 MongoDB $limit運算子，但在搭配 Amazon DocumentDB 使用它時有一些特定的考量。

在 Amazon DocumentDB 中，$limit運算子對於分頁很有用，您想要擷取相符文件總數的子集。它可讓您控制每個回應中傳回的文件數量，改善效能並減少透過網路傳輸的資料量。

參數

- limit：要傳回的文件數量上限。這必須是非負整數值。

範例 (MongoDB Shell)

下列範例示範如何使用 $limit運算子傳回最多一份符合指定篩選條件的文件。

建立範例文件

```
db.test.insertMany([
  { "_id": 1, "star_rating": 4, "comments": "apple is red" },
  { "_id": 2, "star_rating": 5, "comments": "comfortable couch" },
  { "_id": 3, "star_rating": 3, "comments": "apples, oranges - healthy fruit" },
  { "_id": 4, "star_rating": 5, "comments": "this is a great couch" },
  { "_id": 5, "star_rating": 5, "comments": "interesting couch" }
]);
```

查詢範例

```
db.test.createIndex({ comments: "text" });

db.test.find({
  $and: [
    { star_rating: 5 },
    { $text: { $search: "couch" } }
  ]
}).limit(1);
```

輸出

```
[ { _id: 2, star_rating: 5, comments: 'comfortable couch' } ]
```

程式碼範例

若要檢視使用 $limit命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function limitExample() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
      await client.connect();

      const db = client.db('test');
      const collection = db.collection('test');

      await collection.createIndex({ comments: 'text' });

      const query = {
        $and: [
          { star_rating: 5 },
          { $text: { $search: "couch" } }
        ]
      };

      const result = await collection.find(query).limit(1).toArray();

      console.log(result);

    } catch (err) {
        console.error("Error:", err);
    } finally {
        await client.close();
    }

}

limitExample();
```

Python

```
from pymongo import MongoClient

def limit_example():
```

```
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client['test']
        collection = db['test']

        collection.create_index([('comments', 'text')])

        query = {
            '$and': [
                {'star_rating': 5},
                {'$text': {'$search': 'couch'}}
            ]
        }

        result = collection.find(query).limit(1)

        for doc in result:
            print(doc)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        client.close()

limit_example()
```

$listSearchIndexes

8.0.1 版的新功能。

Amazon DocumentDB `$listSearchIndexes` 中的彙總階段會傳回集合上現有搜尋索引的相關資訊。它必須是彙總管道中的第一個階段。

參數

- `name`：（選用）要傳回資訊的搜尋索引名稱。如果省略，則會傳回集合上的所有搜尋索引。

輸出欄位

每個傳回的文件都包含下列欄位：

- name：搜尋索引的名稱。
- status：索引的建置狀態。其中一個 READY（建置且有效）、 BUILDING（並行建置正在進行中）或 FAILED（索引建置未成功完成）。
- queryable：布林值，指出索引目前是否可以用於提供查詢。這是false針對隱藏的索引 （保持 READY) 和針對FAILED索引。隱藏索引是 status為 READY 但 queryable為 的情況false。
- latestDefinitionVersion：包含 version（索引格式版本） 和 createdAt（建立索引的時間） 的文件。

語法

```
db.collection.aggregate([
  { $listSearchIndexes: {} }
])

// Or filter by name:
db.collection.aggregate([
  { $listSearchIndexes: { name: "mySearchIndex" } }
])
```

範例 (MongoDB Shell)

下列範例示範如何使用 $listSearchIndexes階段列出集合上的所有搜尋索引。

查詢範例

```
db.movies.aggregate([
  { $listSearchIndexes: {} }
]);
```

輸出

```
[
  {
    "name": "default",
    "status": "READY",
    "queryable": true,
    "latestDefinitionVersion": {
      "version": 2,
```

```
      "createdAt": ISODate("2026-05-11T21:12:57.974Z")
    }
  }
]
```

隱藏的索引會保留，READY但會報告 queryable: false，因為它無法在隱藏時用來提供查詢：

```
[
  {
    "name": "myHiddenIndex",
    "status": "READY",
    "queryable": false,
    "latestDefinitionVersion": {
      "version": 2,
      "createdAt": ISODate("2026-05-11T21:12:57.974Z")
    }
  }
]
```

若要依特定索引名稱篩選：

```
db.movies.aggregate([
  { $listSearchIndexes: { name: "default" } }
]);
```

程式碼範例

若要檢視使用$listSearchIndexes階段的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('movies');
```

```
    // List all search indexes
    const allIndexes = await collection.aggregate([
      { $listSearchIndexes: {} }
    ]).toArray();
    console.log('All search indexes:', allIndexes);

    // List a specific search index by name
    const namedIndex = await collection.aggregate([
      { $listSearchIndexes: { name: "default" } }
    ]).toArray();
    console.log('Named index:', namedIndex);

  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']
        collection = db['movies']

        # List all search indexes
        all_indexes = list(collection.aggregate([
            { '$listSearchIndexes': {} }
        ]))
        print('All search indexes:', all_indexes)

        # List a specific search index by name
        named_index = list(collection.aggregate([
            { '$listSearchIndexes': { 'name': 'default' } }
        ]))
```

```
        print('Named index:', named_index)

    finally:
        client.close()

example()
```

$literal

Amazon DocumentDB 中的$literal運算子用於表示彙總管道階段中的常值。它可讓您包含特定值，例如數字、字串或布林值，而無需將其解譯為欄位參考或表達式。

當您需要在更複雜的彙總管道中包含常值時，例如建置動態查詢篩選條件或執行計算時，此運算子特別有用。

參數

無

範例 (MongoDB Shell)

下列範例示範如何使用 $literal運算子在彙總管道中包含常值。$literal 運算子用於在 $gt 表達式中包含值 18 作為常值。這可讓彙總管道將年齡欄位與常值 18 進行比較，以判斷該人員是否為成人。

建立範例文件

```
db.collection.insertMany([
  { "name": "John Doe", "age": 30, "city": "New York" },
  { "name": "Jane Doe", "age": 25, "city": "Los Angeles" },
  { "name": "Bob Smith", "age": 35, "city": "Chicago" }
]);
```

查詢範例

```
db.collection.aggregate([
  {
    $project: {
      name: 1,
      age: 1,
      city: 1,
```

```
      isAdult: { $gt: ["$age", { $literal: 18 }] }
    }
  }
]);
```

輸出

```
[
  {
    "_id": ObjectId("601234567890abcdef012345"),
    "name": "John Doe",
    "age": 30,
    "city": "New York",
    "isAdult": true
  },
  {
    "_id": ObjectId("601234567890abcdef012346"),
    "name": "Jane Doe",
    "age": 25,
    "city": "Los Angeles",
    "isAdult": true
  },
  {
    "_id": ObjectId("601234567890abcdef012347"),
    "name": "Bob Smith",
    "age": 35,
    "city": "Chicago",
    "isAdult": true
  }
]
```

程式碼範例

若要檢視使用 $literal命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
```

```
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
        age: 1,
        city: 1,
        isAdult: { $gt: ["$age", { $literal: 18 }] }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.collection

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'age': 1,
                'city': 1,
                'isAdult': { '$gt': ["$age", { '$literal': 18 }] }
            }
        }
    ]))

    print(result)
```

```
    client.close()

example()
```

$ln

4.0 版的新功能。

Amazon DocumentDB 中的$ln運算子會計算指定數字的自然對數 （基礎 e)。它會將數字的對數傳回基礎 e。

參數

- `expression`：將計算自然對數的數字。

範例 (MongoDB Shell)

下列範例示範運算$log子用來計算數字的自然對數。

建立範例文件

```
db.numbers.insertMany([
  { _id: 1, value: 10 },
  { _id: 2, value: 100 },
  { _id: 3, value: 1000 }
]);
```

查詢範例

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    naturalLog: { $ln: "$value" }
  }}
]);
```

輸出

```
[
  { "_id" : 1, "naturalLog" : 2.302585092994046 },
```

```
  { "_id" : 2, "naturalLog" : 4.605170185988092 },
  { "_id" : 3, "naturalLog" : 6.907755278982137 }
]
```

程式碼範例

若要檢視使用 $ln命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $project: {
      _id: 1,
      naturalLog: { $ln: "$value" }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():

    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
    db = client.test
    collection = db.numbers

    result = list(collection.aggregate([
        { "$project": {
            "_id": 1,
            "naturalLog": { "$ln": "$value" }
        }}
    ]))

    print(result)
    client.close()

example()
```

$log

4.0 版的新功能。

Amazon DocumentDB 中的$log運算子會計算數字的自然對數。它會傳回指定數字的 base-e 對數。

參數

- `expression`：將計算自然對數的數字。
- `base`：用於計算日誌的基本值。

範例 (MongoDB Shell)

下列範例示範運算$log子用來計算數字的自然對數。

建立範例文件

```
db.numbers.insertMany([
  { _id: 1, value: 10 },
  { _id: 2, value: 100 },
  { _id: 3, value: 1000 }
]);
```

查詢範例

```
db.numbers.aggregate([
```

```
  { $project: {
    _id: 1,
    naturalLog: { $log: ["$value", 10] }
  }}
]);
```

輸出

```
[
  { "_id" : 1, "naturalLog" : 1 },
  { "_id" : 2, "naturalLog" : 2 },
  { "_id" : 3, "naturalLog" : 2.9999999999999996 }
]
```

程式碼範例

若要檢視使用 $log命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $project: {
      _id: 1,
      naturalLog: { $log: ["$value", 10] }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    db = client.test
    collection = db.numbers

    result = list(collection.aggregate([
        { "$project": {
            "_id": 1,
            "naturalLog": { "$log": ["$value", 10] }
        }}
    ]))

    print(result)
    client.close()

example()
```

$log10

4.0 版的新功能。

Amazon DocumentDB 中的$log10運算子用於計算數字的 base-10 對數。它適用於對彙總管道中的數值欄位執行對數計算。

參數

- expression：計算 base-10 對數的數值表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $log10運算子來計算數值欄位的 base-10 對數。

建立範例文件

```
db.numbers.insertMany([
```

```
  { _id: 1, value: 1 },
  { _id: 2, value: 10 },
  { _id: 3, value: 100 },
  { _id: 4, value: 1000 }
]);
```

查詢範例

```
db.numbers.aggregate([
  {
    $project: {
      _id: 1,
      log10Value: { $log10: "$value" }
    }
  }
]);
```

輸出

```
[
  { "_id": 1, "log10Value": 0 },
  { "_id": 2, "log10Value": 1 },
  { "_id": 3, "log10Value": 2 },
  { "_id": 4, "log10Value": 3 }
]
```

程式碼範例

若要檢視使用 $log10命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  let client;

  try {
    client = await MongoClient.connect('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
```

```
    const db = client.db('test');
    const collection = db.collection('numbers');

    const result = await collection.aggregate([
      {
        $project: {
          _id: 1,
          log10Value: { $log10: "$value" }
        }
      }
    ]).toArray();

    console.log(result);
  } catch (error) {
    console.error("An error occurred:", error);
  } finally {
    if (client) {
      await client.close();
    }
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client.test
        collection = db.numbers

        result = list(collection.aggregate([
            {
                '$project': {
                    '_id': 1,
                    'log10Value': { '$log10': '$value' }
```

```
                }
            }
        ]))

        print(result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$lt

$lt 彙總運算子會比較兩個值，如果第一個值小於第二個值，true則傳回 ，否則傳回 false。

參數

- expression1：要比較的第一個值。
- expression2：要比較的第二個值。

範例 (MongoDB Shell)

下列範例示範如何使用 $lt運算子來識別低庫存項目。

建立範例文件

```
db.warehouse.insertMany([
  { _id: 1, item: "Bolts", stock: 5 },
  { _id: 2, item: "Nuts", stock: 25 },
  { _id: 3, item: "Screws", stock: 8 }
]);
```

查詢範例

```
db.warehouse.aggregate([
  {
```

```
    $project: {
      item: 1,
      stock: 1,
      lowStock: { $lt: ["$stock", 10] }
    }
  }
]);
```

輸出

```
[
  { _id: 1, item: 'Bolts', stock: 5, lowStock: true },
  { _id: 2, item: 'Nuts', stock: 25, lowStock: false },
  { _id: 3, item: 'Screws', stock: 8, lowStock: true }
]
```

程式碼範例

若要檢視使用$lt彙總運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('warehouse');

  const result = await collection.aggregate([
    {
      $project: {
        item: 1,
        stock: 1,
        lowStock: { $lt: ["$stock", 10] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
```

```
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['warehouse']

    result = list(collection.aggregate([
        {
            '$project': {
                'item': 1,
                'stock': 1,
                'lowStock': { '$lt': ['$stock', 10] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$lte

`$lte` 彙總運算子會比較兩個值，如果第一個值小於或等於第二個值，`true`則傳回 ，否則傳回 `false`。

參數

- `expression1`：要比較的第一個值。
- `expression2`：要比較的第二個值。

範例 (MongoDB Shell)

以下範例示範如何使用 $lte 運算子來識別經濟實惠的項目。

建立範例文件

```
db.menu.insertMany([
  { _id: 1, dish: "Salad", price: 8 },
  { _id: 2, dish: "Pasta", price: 12 },
  { _id: 3, dish: "Soup", price: 6 }
]);
```

查詢範例

```
db.menu.aggregate([
  {
    $project: {
      dish: 1,
      price: 1,
      affordable: { $lte: ["$price", 10] }
    }
  }
]);
```

輸出

```
[
  { _id: 1, dish: 'Salad', price: 8, affordable: true },
  { _id: 2, dish: 'Pasta', price: 12, affordable: false },
  { _id: 3, dish: 'Soup', price: 6, affordable: true }
]
```

程式碼範例

若要檢視使用$lte彙總運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('menu');

  const result = await collection.aggregate([
    {
      $project: {
        dish: 1,
        price: 1,
        affordable: { $lte: ["$price", 10] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['menu']

    result = list(collection.aggregate([
        {
            '$project': {
                'dish': 1,
                'price': 1,
                'affordable': { '$lte': ['$price', 10] }
            }
        }
    ]))
```

```
    print(result)
    client.close()

example()
```

$lookup

Amazon DocumentDB $lookup 中的彙總階段可讓您在兩個集合之間執行左側外部聯結。此操作可讓您根據相符的欄位值，合併來自多個集合的資料。當您需要將相關集合的資料納入查詢結果時，此功能特別有用。

參數

- `from`：要執行聯結的集合名稱。
- `localField`：輸入文件中要與 相符的欄位`foreignField`。
- `foreignField`：集合中`from`要比對之文件的欄位`localField`。
- `as`：要新增至輸出文件的新欄位名稱，其中包含來自`from`集合的相符文件。

範例 (MongoDB Shell)

下列範例示範將資料從`orders`集合加入`customers`集合的簡單`$lookup`操作。

建立範例文件

```
db.customers.insertMany([
  { _id: 1, name: "Alice" },
  { _id: 2, name: "Bob" },
  { _id: 3, name: "Charlie" }
]);

db.orders.insertMany([
  { _id: 1, customer_id: 1, total: 50 },
  { _id: 2, customer_id: 1, total: 100 },
  { _id: 3, customer_id: 2, total: 75 }
]);
```

查詢範例

```
db.customers.aggregate([
```

```
  {
    $lookup: {
      from: "orders",
      localField: "_id",
      foreignField: "customer_id",
      as: "orders"
    }
  }
]);
```

輸出

```
[
  {
    _id: 1,
    name: 'Alice',
    orders: [
      { _id: 2, customer_id: 1, total: 100 },
      { _id: 1, customer_id: 1, total: 50 }
    ]
  },
  { _id: 3, name: 'Charlie', orders: [] },
  {
    _id: 2,
    name: 'Bob',
    orders: [ { _id: 3, customer_id: 2, total: 75 } ]
  }
]
```

程式碼範例

若要檢視使用 $lookup命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  await client.connect();
```

```
  const db = client.db('test');

  const result = await db.collection('customers').aggregate([
    {
      $lookup: {
        from: 'orders',
        localField: '_id',
        foreignField: 'customer_id',
        as: 'orders'
      }
    }
  ]).toArray();

  console.log(JSON.stringify(result, null, 2));
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    db = client.test

    collection = db.customers

    pipeline = [
        {
            "$lookup": {
                "from": "orders",
                "localField": "_id",
                "foreignField": "customer_id",
                "as": "orders"
            }
        }
    ]
```

```
    result = collection.aggregate(pipeline)

    for doc in result:
        print(doc)

    client.close()

example()
```

$ltrim

4.0 版的新功能。

Elastic 叢集不支援。

Amazon DocumentDB 中的$ltrim運算子用於從字串中移除前置字元。根據預設，它會移除前導空格字元，但您也可以透過傳遞字元引數來指定要移除的一組字元。

參數

- input：要從中移除前導空格字元的輸入字串。
- chars：（選用）移除特定字元。

範例 (MongoDB Shell)

下列範例示範 的使用方式$ltrim，以從字串開頭移除指定的字元 (" *")。

建立範例文件

```
db.collection.insertMany([
  { name: " *John Doe", age: 30 },
  { name: "Jane Doe*", age: 25 },
  { name: "  Bob Smith  ", age: 35 }
]);
```

查詢範例

```
db.collection.aggregate([
  {
```

```
    $project: {
      _id: 0,
      name: {
        $ltrim: { input: "$name", chars: " *" }
      },
      age: 1
    }
  }
]);
```

輸出

```
[
  { "name": "John Doe", "age": 30 },
  { "name": "Jane Doe ", "age": 25 },
  { "name": "Bob Smith  ", "age": 35 }
]
```

程式碼範例

若要檢視使用 $ltrim命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();

    const db = client.db('test');
    const collection = db.collection('collection');

    const pipeline = [
      {
        $project: {
          _id: 0,
          name: {
            $ltrim: {
```

```
              input: '$name',
              chars: ' *'
            }
          },
          age: 1
        }
      }
    ];

    const result = await collection.aggregate(pipeline).toArray();

    console.dir(result, { depth: null });

  } finally {
    await client.close();
  }
}

example().catch(console.error);
```

Python

```
from pymongo import MongoClient

def example():
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client.test
        collection = db.collection

        pipeline = [
            {
                "$project": {
                    "_id": 0,
                    "name": {
                        "$ltrim": {
                            "input": "$name",
                            "chars": " *"
                        }
                    },
```

```
                    "age": 1
                }
            }
        ]

        results = collection.aggregate(pipeline)

        for doc in results:
            print(doc)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        client.close()

example()
```

$map

Amazon DocumentDB 中的$map運算子可讓您將指定的表達式套用至陣列中的每個元素，並傳回具有轉換元素的新陣列。此運算子特別適用於處理和轉換陣列中的資料，這有助於簡化應用程式程式碼，並透過將陣列處理推送至資料庫層級來改善查詢效能。

參數

- input：要轉換的陣列。
- as：（選用） 要在表達式 內使用的變數名稱，代表目前正在處理的元素。
- in：要套用至輸入陣列中每個元素的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $map 運算子轉換數字陣列，將每個值加倍。

建立範例文件

```
db.collection.insertMany([
  { _id: 1, numbers: [1, 2, 3, 4, 5] },
  { _id: 2, numbers: [10, 20, 30, 40, 50] }
])
```

查詢範例

```
db.collection.aggregate([
  {
    $project: {
      doubledNumbers: { $map: { input: "$numbers", as: "num", in: { $multiply: ["$
$num", 2] } } }
    }
  }
])
```

輸出

```
[
  { _id: 1, doubledNumbers: [2, 4, 6, 8, 10] },
  { _id: 2, doubledNumbers: [20, 40, 60, 80, 100] }
]
```

程式碼範例

若要檢視使用 $map命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.aggregate([
    {
      $project: {
        doubledNumbers: { $map: { input: "$numbers", as: "num", in: { $multiply: ["$
$num", 2] } } }
      }
    }
  ]).toArray();
```

```
  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.collection

    result = list(collection.aggregate([
        {
            '$project': {
                'doubledNumbers': { '$map': { 'input': '$numbers', 'as': 'num',
 'in': { '$multiply': ['$$num', 2] } } }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$match

Amazon DocumentDB 中的$match管道階段用於將輸入文件篩選為僅符合指定查詢條件的文件。這是彙總操作中最常用的管道階段之一。$match 階段會在任何其他管道階段之前套用，讓您有效率地減少後續階段需要處理的文件數量。

參數

- query：表達操作選擇條件的文件。查詢文件使用與 find()方法相同的語法。

範例 (MongoDB Shell)

下列範例示範如何使用 $match階段，根據特定欄位值篩選文件。

建立範例文件

```
db.collection.insertMany([
  { _id: 1, name: "John", age: 25, city: "New York" },
  { _id: 2, name: "Jane", age: 30, city: "Los Angeles" },
  { _id: 3, name: "Bob", age: 35, city: "Chicago" },
  { _id: 4, name: "Alice", age: 40, city: "Miami" }
]);
```

查詢範例

```
db.collection.aggregate([
  { $match: { age: { $gt: 30 } } },
  { $project: { _id: 1, name: 1, city: 1 } }
]);
```

輸出

```
[
  { "_id": 3, "name": "Bob", "city": "Chicago" },
  { "_id": 4, "name": "Alice", "city": "Miami" }
]
```

$match 階段會篩選文件，只包含age欄位大於 30 的文件。

程式碼範例

若要檢視使用 $match命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
```

```
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.aggregate([
    { $match: { age: { $gt: 30 } } },
    { $project: { _id: 1, name: 1, city: 1 } }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    db = client['test']
    collection = db['collection']

    result = list(collection.aggregate([
        { '$match': { 'age': { '$gt': 30 } } },
        { '$project': { '_id': 1, 'name': 1, 'city': 1 } }
    ]))

    print(result)
    client.close()

example()
```

$max

$max 彙總階段用於傳回管道階段中所有文件指定欄位的最大值。此運算子適用於尋找一組文件中的最高值。

參數

- expression：用來計算最大值的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $max運算子在學生文件集合中尋找最高分數。$group 階段將所有文件分組在一起，而$max運算子用於計算所有文件中 score 欄位的最大值。

建立範例文件

```
db.students.insertMany([
  { name: "John", score: 85 },
  { name: "Jane", score: 92 },
  { name: "Bob", score: 78 },
  { name: "Alice", score: 90 }
])
```

查詢範例

```
db.students.aggregate([
  { $group: { _id: null, maxScore: { $max: "$score" } } },
  { $project: { _id: 0, maxScore: 1 } }
])
```

輸出

```
[ { maxScore: 92 } ]
```

程式碼範例

若要檢視使用 $max命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findMaxScore() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const students = db.collection('students');
```

```
  const result = await students.aggregate([
    { $group: { _id: null, maxScore: { $max: "$score" } } }
  ]).toArray();

  console.log(result);
  await client.close();
}

findMaxScore();
```

Python

```
from pymongo import MongoClient

def find_max_score():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    students = db.students

    result = list(students.aggregate([
        { "$group": { "_id": None, "maxScore": { "$max": "$score" } } }
    ]))

    print(result)
    client.close()

find_max_score()
```

$maxN

8.0.1 版的新功能。

Amazon DocumentDB 中的$maxN運算子會傳回 N 個最大值。在$group階段中用作累積器時，它會傳回每個群組中 N 個最大值的陣列。用作陣列表達式運算子時，它會傳回陣列的 N 個元素上限。

參數

- input：解析為要傳回n最大值的陣列或欄位的表達式。
- n：解析為正整數的表達式，指定要傳回多少最大值。

Behavior (行為)

陣列表達式運算子行為

- 您無法指定n小於 的值1。
- $maxN 會篩選掉input陣列中找到null的值。
- 如果指定的 n 大於或等於input陣列中的元素數目，則 會$maxN傳回input陣列中的所有元素。
- 如果 input解析為非陣列值，則彙總操作錯誤。
- 如果 同時input包含數值和字串元素，則字串元素會根據 BSON 比較順序在數值元素之前排序。

累積器行為

- 用作累積器時， n 必須是正整數表達式，其為常數或取決於 _id的值$group。
- $maxN 會篩選掉 null 值和遺失值。
- 如果群組包含的n元素少於 個，則 會$maxN傳回群組中的所有元素。
- $maxN 會依照 BSON 比較順序比較輸入資料，以判斷適當的輸出類型。當輸入資料包含多個資料類型時，$maxN輸出類型在比較順序中最高。

輸出排序

$maxN 不會以特定排序順序傳回值。如果需要保證特定排序順序，請$topN改用 ，或使用 包裝結果$sortArray。

範例 (MongoDB Shell)

下列範例顯示如何使用$maxN累積器擷取每個主題的兩個最高分數。

建立範例文件

```
db.scores.insertMany([
  { subject: "math", score: 85 },
  { subject: "math", score: 72 },
  { subject: "math", score: 93 },
  { subject: "math", score: 68 },
  { subject: "science", score: 90 },
  { subject: "science", score: 78 },
  { subject: "science", score: 65 },
  { subject: "science", score: 88 }
```

```
]);
```

查詢範例

```
db.scores.aggregate([
  { $group: { _id: "$subject", highestTwo: { $maxN: { input: "$score", n: 2 } } } }
]);
```

輸出

```
[
  { "_id": "math", "highestTwo": [85, 93] },
  { "_id": "science", "highestTwo": [88, 90] }
]
```

表達式用量範例 (MongoDB Shell)

運算$maxN子也可以用作$project階段內的表達式，從陣列欄位中傳回 N 最大元素。

建立範例文件

```
db.readings.insertMany([
  { _id: 1, sensor: "A", values: [45, 12, 78, 3, 56] },
  { _id: 2, sensor: "B", values: [90, 23, 67, 11, 44] }
]);
```

查詢範例

```
db.readings.aggregate([
  { $project: {
      highestThree: { $maxN: { input: "$values", n: 3 } }
    }}
]);
```

輸出

```
[
  { "_id": 1, "highestThree": [45, 56, 78] },
  { "_id": 2, "highestThree": [44, 90, 67] }
]
```

程式碼範例

若要檢視使用 $maxN 運算子的程式碼範例，請選擇您要使用的語言標籤。下列範例顯示累積器用量（在 中$group) 和表達式用量（在 中$project)：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');

    // Accumulator usage: N largest values per group
    const scores = db.collection('scores');
    const accumulatorResult = await scores.aggregate([
      { $group: { _id: "$subject", highestTwo: { $maxN: { input: "$score", n:
 2 } } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Expression usage: N largest elements from an array field
    const readings = db.collection('readings');
    const expressionResult = await readings.aggregate([
      { $project: { highestThree: { $maxN: { input: "$values", n: 3 } } } }
    ]).toArray();
    console.log('Expression result:', expressionResult);

  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
```

```
def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']

        # Accumulator usage: N largest values per group
        scores = db['scores']
        accumulator_result = list(scores.aggregate([
            { '$group': { '_id': '$subject', 'highestTwo': { '$maxN': { 'input':
 '$score', 'n': 2 } } } }
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: N largest elements from an array field
        readings = db['readings']
        expression_result = list(readings.aggregate([
            { '$project': { 'highestThree': { '$maxN': { 'input': '$values', 'n':
 3 } } } }
        ]))
        print('Expression result:', expression_result)

    finally:
        client.close()

example()
```

$median

8.0.1 版的新功能。

Amazon DocumentDB 中的$median運算子會計算數值資料的中位數值。作為累積器，它會計算彙總管道$group階段中群組內文件的數值中位數。做為表達式，它會計算數字陣列的中位數。

參數

- input：解析為數值或數值陣列的表達式。
- method：指定計算方法的字串。目前僅"approximate"支援使用 t-digest 演算法的 。

Behavior (行為)

"approximate" 方法使用 t-digest 演算法來計算近似中位數。結果是來自資料集的現有值，而不是值之間的插補。精確度會隨著資料點數量的增加而提高。

範例 (MongoDB Shell)

下列範例示範如何使用 $median 運算子來計算每個類別的中位數測試分數。

建立範例文件

```
db.students.insertMany([
  { class: "A", score: 72 },
  { class: "A", score: 85 },
  { class: "A", score: 90 },
  { class: "A", score: 68 },
  { class: "A", score: 95 },
  { class: "B", score: 80 },
  { class: "B", score: 75 },
  { class: "B", score: 92 },
  { class: "B", score: 88 },
  { class: "B", score: 70 }
]);
```

查詢範例

```
db.students.aggregate([
  { $group: {
      _id: "$class",
      medianScore: { $median: { input: "$score", method: "approximate" } }
    }}
]);
```

輸出

```
[
  { "_id": "A", "medianScore": 85 },
  { "_id": "B", "medianScore": 80 }
]
```

表達式用量範例 (MongoDB Shell)

$median 運算子也可以用作$project階段內的表達式，以計算陣列欄位的中位數。

建立範例文件

```
db.surveys.insertMany([
  { _id: 1, ratings: [3, 5, 7, 9, 2] },
  { _id: 2, ratings: [10, 20, 30, 40, 50] },
  { _id: 3, ratings: [1, 1, 2, 3, 5] }
]);
```

查詢範例

```
db.surveys.aggregate([
  { $project: {
      medianRating: { $median: { input: "$ratings", method: "approximate" } }
    }}
]);
```

輸出

```
[
  { "_id": 1, "medianRating": 5 },
  { "_id": 2, "medianRating": 30 },
  { "_id": 3, "medianRating": 2 }
]
```

程式碼範例

若要檢視使用 $median 運算子的程式碼範例，請選擇您要使用的語言標籤。下列範例顯示累積器用量（在 中$group) 和表達式用量（在 中$project)：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();
```

```
    const db = client.db('test');

    // Accumulator usage: median across grouped documents
    const students = db.collection('students');
    const accumulatorResult = await students.aggregate([
      { $group: {
          _id: "$class",
          medianScore: { $median: { input: "$score", method: "approximate" } }
        }}
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Expression usage: median of an array field
    const surveys = db.collection('surveys');
    const expressionResult = await surveys.aggregate([
      { $project: {
          medianRating: { $median: { input: "$ratings", method: "approximate" } }
        }}
    ]).toArray();
    console.log('Expression result:', expressionResult);

  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']

        # Accumulator usage: median across grouped documents
        students = db['students']
        accumulator_result = list(students.aggregate([
```

```
            { '$group': {
                '_id': '$class',
                'medianScore': { '$median': { 'input': '$score', 'method':
 'approximate' } }
            }}
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: median of an array field
        surveys = db['surveys']
        expression_result = list(surveys.aggregate([
            { '$project': {
                'medianRating': { '$median': { 'input': '$ratings', 'method':
 'approximate' } }
            }}
        ]))
        print('Expression result:', expression_result)

    finally:
        client.close()

example()
```

$meta

$meta 彙總運算子會存取與彙總管道中文件相關聯的中繼資料。它通常用於擷取文字搜尋分數，並依相關性排序結果。

參數

- textScore：擷取文字搜尋分數，指出與搜尋查詢的文件相關性。

範例 (MongoDB Shell)

下列範例示範在彙總管道中使用 $meta 運算子，依文字搜尋分數擷取和排序。

建立範例文件

```
db.articles.createIndex({ content: "text" });

db.articles.insertMany([
```

```
  { _id: 1, title: "Python Programming", content: "Python is a versatile programming
 language used for web development." },
  { _id: 2, title: "Python Guide", content: "Learn Python programming with Python
 tutorials and Python examples." },
  { _id: 3, title: "Java Basics", content: "Java is another popular programming
 language." }
]);
```

查詢範例

```
db.articles.aggregate([
  { $match: { $text: { $search: "Python" } } },
  { $addFields: { score: { $meta: "textScore" } } },
  { $sort: { score: -1 } }
]);
```

輸出

```
[
  {
    _id: 2,
    title: 'Python Guide',
    content: 'Learn Python programming with Python tutorials and Python examples.',
    score: 1.5
  },
  {
    _id: 1,
    title: 'Python Programming',
    content: 'Python is a versatile programming language used for web development.',
    score: 0.75
  }
]
```

程式碼範例

若要檢視使用$meta彙總運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('articles');

  const result = await collection.aggregate([
    { $match: { $text: { $search: "Python" } } },
    { $addFields: { score: { $meta: "textScore" } } },
    { $sort: { score: -1 } }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['articles']

    result = list(collection.aggregate([
        { '$match': { '$text': { '$search': 'Python' } } },
        { '$addFields': { 'score': { '$meta': 'textScore' } } },
        { '$sort': { 'score': -1 } }
    ]))

    print(result)
    client.close()

example()
```

$merge

在 8.0 中推出

Amazon DocumentDB $merge 中的彙總階段用於將上一個管道階段的結果合併到目標集合中。這有助於根據輸入文件的資料，在目標集合中更新或插入文件。

$merge 階段可讓您根據輸入文件與目標集合之間的相符條件執行各種動作，例如：

```
- Insert new documents
- Update existing documents
- Delete documents
- Fail the operation if there are any conflicts
```

參數

- into：（必要）要合併輸入文件的目標集合名稱。
- on：（必要）做為輸入文件與目標集合之間相符條件的欄位 (s)。
- whenMatched：（選用）當輸入文件符合目標集合中的現有文件時，要執行的動作。支援的值為："merge"、"replace"、"keepExisting" 和 "fail"。
- whenNotMatched：（選用）當輸入文件不符合目標集合中的任何文件時，要執行的動作。支援的值為： "insert"和 "fail"。
- bypassDocumentValidation：（選用）設為 時true，階段寫入目標集合的文件會$merge略過該集合上定義的任何結構描述驗證規則 ($jsonSchema)。預設值為 false，可強制執行目標集合的驗證規則。這在 Amazon DocumentDB 8.0 中受支援。

範例 (MongoDB Shell)

下列範例示範如何使用 $merge階段，使用來自輸入管道的新資料來更新users集合。

建立範例文件

```
db.users.insertMany([
  { _id: 1, name: "John Doe", email: "john@example.com" },
  { _id: 2, name: "Jane Smith", email: "jane@example.com" }
]);

db.inputData.insertMany([
```

```
  { _id: 1, name: "John Doe", email: "john@example.com", phone: "123-456-7890" },
  { _id: 3, name: "Bob Johnson", email: "bob@example.com", phone: "987-654-3210" }
]);
```

查詢範例

```
db.inputData.aggregate([
  {
    $merge: {
      into: "users",
      on: "_id",
      whenMatched: "merge",
      whenNotMatched: "insert"
    }
  }
])
```

輸出

執行$merge管道之後，users集合將包含下列文件：

```
[
  { _id: 1, name: "John Doe", email: "john@example.com", phone: "123-456-7890" },
  { _id: 2, name: "Jane Smith", email: "jane@example.com" },
  { _id: 3, name: "Bob Johnson", email: "bob@example.com", phone: "987-654-3210" }
]
```

程式碼範例

若要檢視使用 $merge命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

以下是在 Node.js 應用程式中使用 $merge Operator 的範例：

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
```

```
  const db = client.db('test');

  await db.collection('inputData').aggregate([
    {
      $merge: {
        into: 'users',
        on: '_id',
        whenMatched: 'merge',
        whenNotMatched: 'insert'
      }
    }
  ]).toArray();

  const users = await db.collection('users').find({}).toArray();
  console.log(users);

  await client.close();
}

example();
```

Python

以下是在 Python 應用程式中使用 $merge Operator 的範例：

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']

    # Assumes collections 'users' and 'inputData' already exist with sample data
    db.inputData.aggregate([
        {
            '$merge': {
                'into': 'users',
                'on': '_id',
                'whenMatched': 'merge',
                'whenNotMatched': 'insert'
            }
        }
    ])
```

```
    users = list(db.users.find({}))
    print(users)

    client.close()

example()
```

$mergeObjects

Amazon DocumentDB 中的$mergeObjects運算子用於將多個文件或物件合併為單一文件。當您需要合併兩個或多個文件或物件的內容時，此運算子特別有用，可能會覆寫一個物件的值與另一個物件的值。

參數

- expression1：要合併的第一個物件。
- expression2：（選用）要合併的第二個物件。
- expression3：（選用）要合併的其他物件。

範例 (MongoDB Shell)

下列範例示範如何使用 $mergeObjects運算子來結合兩個物件。

建立範例文件

```
db.collection.insertMany([
  { "_id": 1, "name": "John", "address": { "city": "New York", "state": "NY" } },
  { "_id": 2, "name": "Jane", "address": { "city": "Los Angeles", "state": "CA" } }
]);
```

查詢範例

```
db.collection.aggregate([
  {
    $project: {
      "combinedAddress": {
        $mergeObjects: ["$address", { "country": "USA" }]
      }
```

```
    }
  }
])
```

輸出

```
[
  {
    "_id": 1,
    "combinedAddress": {
      "city": "New York",
      "state": "NY",
      "country": "USA"
    }
  },
  {
    "_id": 2,
    "combinedAddress": {
      "city": "Los Angeles",
      "state": "CA",
      "country": "USA"
    }
  }
]
```

程式碼範例

若要檢視使用 $mergeObjects命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.aggregate([
    {
      $project: {
```

```
        "combinedAddress": {
          $mergeObjects: ["$address", { "country": "USA" }]
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['collection']

    result = list(collection.aggregate([
        {
            '$project': {
                "combinedAddress": {
                    "$mergeObjects": ["$address", { "country": "USA" }]
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$millisecond

Amazon DocumentDB 中的$millisecond運算子用於擷取日期值的毫秒部分。

參數

無

範例 (MongoDB Shell)

此範例示範如何使用 $millisecond運算子擷取日期值的毫秒部分。

建立範例文件

```
db.events.insert([
  {
    "name": "Event 1",
    "timestamp": ISODate("2023-04-21T10:30:15.123Z")
  },
  {
    "name": "Event 2",
    "timestamp": ISODate("2023-04-21T10:30:15.456Z")
  },
  {
    "name": "Event 3",
    "timestamp": ISODate("2023-04-21T10:30:15.789Z")
  }
])
```

查詢範例

```
db.events.aggregate([
  {
    $project: {
      name: 1,
      milliseconds: { $millisecond: "$timestamp" }
    }
  }
])
```

輸出

```
[
  {
    "_id": ObjectId("644332a42054ed1b0d15f0c1"),
    "name": "Event 1",
    "milliseconds": 123
```

```
  },
  {
    "_id": ObjectId("644332a42054ed1b0d15f0c2"),
    "name": "Event 2",
    "milliseconds": 456
  },
  {
    "_id": ObjectId("644332a42054ed1b0d15f0c3"),
    "name": "Event 3",
    "milliseconds": 789
  }
]
```

程式碼範例

若要檢視使用 $millisecond命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const events = db.collection('events');

  const result = await events.aggregate([
    {
      $project: {
        name: 1,
        milliseconds: { $millisecond: '$timestamp' }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
events = db['events']

result = list(events.aggregate([
    {
        '$project': {
            'name': 1,
            'milliseconds': { '$millisecond': '$timestamp' }
        }
    }
]))

print(result)

client.close()
```

$min

$min 運算子會從值陣列傳回最小值。它可用於彙總階段，以尋找跨多個文件指定欄位的最小值。

參數

- expression：要評估的表達式。這可以是欄位路徑、變數或任何解析為值的表達式。

範例 (MongoDB Shell)

下列範例示範 $min運算子的使用情況，以尋找跨多個文件之 age 欄位的最小值。

建立範例文件

```
db.users.insertMany([
  { name: "John", age: 35 },
  { name: "Jane", age: 28 },
  { name: "Bob", age: 42 },
```

```
  { name: "Alice", age: 31 }
]);
```

查詢範例

```
db.users.aggregate([
  { $group: { _id: null, minAge: { $min: "$age" } } },
  { $project: { _id: 0, minAge: 1 } }
])
```

輸出

```
[ { minAge: 28 } ]
```

程式碼範例

若要檢視使用 $min命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findMinAge() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const result = await users.aggregate([
    { $group: {
        _id: null,
        minAge: { $min: "$age" }
      }}
  ]).toArray();

  console.log(result);
  client.close();
}

findMinAge();
```

Python

```
from pymongo import MongoClient

def find_min_age():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    users = db.users

    result = list(users.aggregate([
        { "$group": {
            "_id": None,
            "minAge": { "$min": "$age" }
        }}
    ]))

    print(result)
    client.close()

find_min_age()
```

$minN

8.0.1 版的新功能。

Amazon DocumentDB 中的$minN運算子會傳回 N 個最小值。在$group階段中用作累積器時，它會傳回每個群組中 N 個最小值的陣列。用作陣列表達式運算子時，它會傳回陣列的 N 個最小元素。

參數

- input：解析為要傳回n最小值的陣列或欄位的表達式。
- n：解析為正整數的表達式，指定要傳回多少最小值。

Behavior (行為)

陣列表達式運算子行為

- 您無法指定n小於 的值1。
- $minN 會篩選掉input陣列中找到null的值。

- 如果指定的 n 大於或等於input陣列中的元素數目，則 會$minN傳回input陣列中的所有元素。
- 如果 input解析為非陣列值，則彙總操作錯誤。
- 如果 同時input包含數值和字串元素，則會根據 BSON 比較順序，在字串元素之前排序數值元素。

累積器行為

- 用作累積器時， n 必須是正整數表達式，其為常數或取決於 _id的值$group。
- $minN 會篩選掉 null 值和遺失值。
- 如果群組包含的n元素少於 個，則 會$minN傳回群組中的所有元素。
- $minN 會依照 BSON 比較順序比較輸入資料，以判斷適當的輸出類型。當輸入資料包含多個資料類型時，$minN輸出類型是比較順序中最低的。

輸出排序

$minN 不會以特定排序順序傳回值。如果需要保證特定排序順序，請$bottomN改用 ，或使用 包裝結果$sortArray。

範例 (MongoDB Shell)

下列範例顯示如何使用$minN累積器擷取每個主題的兩個最低分數。

建立範例文件

```
db.scores.insertMany([
  { subject: "math", score: 85 },
  { subject: "math", score: 72 },
  { subject: "math", score: 93 },
  { subject: "math", score: 68 },
  { subject: "science", score: 90 },
  { subject: "science", score: 78 },
  { subject: "science", score: 65 },
  { subject: "science", score: 88 }
]);
```

查詢範例

```
db.scores.aggregate([
  { $group: { _id: "$subject", lowestTwo: { $minN: { input: "$score", n: 2 } } } }
```

```
]);
```

輸出

```
[
  { "_id": "math", "lowestTwo": [72, 68] },
  { "_id": "science", "lowestTwo": [78, 65] }
]
```

表達式用量範例 (MongoDB Shell)

運算$minN子也可以用作$project階段內的表達式，從陣列欄位中傳回 N 個最小元素。

建立範例文件

```
db.readings.insertMany([
  { _id: 1, sensor: "A", values: [45, 12, 78, 3, 56] },
  { _id: 2, sensor: "B", values: [90, 23, 67, 11, 44] }
]);
```

查詢範例

```
db.readings.aggregate([
  { $project: {
      lowestThree: { $minN: { input: "$values", n: 3 } }
    }}
]);
```

輸出

```
[
  { "_id": 1, "lowestThree": [45, 12, 3] },
  { "_id": 2, "lowestThree": [44, 23, 11] }
]
```

程式碼範例

若要檢視使用 $minN 運算子的程式碼範例，請選擇您要使用的語言標籤。下列範例顯示累積器用量（在 中$group) 和表達式用量（在 中$project)：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');

    // Accumulator usage: N smallest values per group
    const scores = db.collection('scores');
    const accumulatorResult = await scores.aggregate([
      { $group: { _id: "$subject", lowestTwo: { $minN: { input: "$score", n:
 2 } } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Expression usage: N smallest elements from an array field
    const readings = db.collection('readings');
    const expressionResult = await readings.aggregate([
      { $project: { lowestThree: { $minN: { input: "$values", n: 3 } } } }
    ]).toArray();
    console.log('Expression result:', expressionResult);

  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
    try:
        db = client['test']

        # Accumulator usage: N smallest values per group
        scores = db['scores']
        accumulator_result = list(scores.aggregate([
            { '$group': { '_id': '$subject', 'lowestTwo': { '$minN': { 'input':
 '$score', 'n': 2 } } } }
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: N smallest elements from an array field
        readings = db['readings']
        expression_result = list(readings.aggregate([
            { '$project': { 'lowestThree': { '$minN': { 'input': '$values', 'n':
 3 } } } }
        ]))
        print('Expression result:', expression_result)

    finally:
        client.close()

example()
```

$minute

Amazon DocumentDB $minute 中的彙總管道階段會從日期或時間戳記欄位中擷取分鐘值。

當您需要在彙總管道中執行日期和時間型計算或分組時，此運算子非常有用。

參數

- expression：要從中擷取分鐘值的日期或時間戳記欄位。

範例 (MongoDB Shell)

下列範例示範如何使用 $minute運算子，依從時間戳記欄位擷取的分鐘值將文件分組，並計算每個群組中的文件數量。

建立範例文件

```
db.events.insertMany([
  { timestamp: new Date("2023-04-15T10:30:25.000Z") },
  { timestamp: new Date("2023-04-15T10:30:35.000Z") },
  { timestamp: new Date("2023-04-15T10:31:05.000Z") },
  { timestamp: new Date("2023-04-15T10:31:45.000Z") },
  { timestamp: new Date("2023-04-15T10:32:15.000Z") }
]);
```

查詢範例

```
db.events.aggregate([
  {
    $group: {
      _id: {
        minute: { $minute: "$timestamp" }
      },
      count: { $count: {} }
    }
  },
  { $sort: { "_id.minute": 1 } }
]);
```

輸出

```
[
  { "_id": { "minute": 30 }, "count": 2 },
  { "_id": { "minute": 31 }, "count": 2 },
  { "_id": { "minute": 32 }, "count": 1 }
]
```

程式碼範例

若要檢視使用 $minute命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
```

```
  const db = client.db('test');
  const collection = db.collection('events');

  await collection.insertMany([
    { timestamp: new Date("2023-04-15T10:30:25.000Z") },
    { timestamp: new Date("2023-04-15T10:30:35.000Z") },
    { timestamp: new Date("2023-04-15T10:31:05.000Z") },
    { timestamp: new Date("2023-04-15T10:31:45.000Z") },
    { timestamp: new Date("2023-04-15T10:32:15.000Z") }
  ]);

  const result = await collection.aggregate([
    {
      $group: {
        _id: {
          minute: { $minute: "$timestamp" }
        },
        count: { $count: {} }
      }
    },
    { $sort: { "_id.minute": 1 } }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient
from datetime import datetime

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['events']

    collection.insert_many([
        {'timestamp': datetime(2023, 4, 15, 10, 30, 25)},
```

```
        {'timestamp': datetime(2023, 4, 15, 10, 30, 35)},
        {'timestamp': datetime(2023, 4, 15, 10, 31, 5)},
        {'timestamp': datetime(2023, 4, 15, 10, 31, 45)},
        {'timestamp': datetime(2023, 4, 15, 10, 32, 15)}
    ])

    result = list(collection.aggregate([
        {
            '$group': {
                '_id': {
                    'minute': {'$minute': '$timestamp'}
                },
                'count': {'$count': {}}
            }
        },
        {'$sort': {'_id.minute': 1}}
    ]))

    print(result)
    client.close()

example()
```

$mod

$mod 運算子是對數字執行模數操作的算術運算子。它會傳回將一個數字除以另一個數字的其餘部分。此運算子通常用於判斷數字是奇數還是偶數，或將項目分配到有限數量的群組。

參數

- expression1：分配表達式。
- expression2：除數表達式。

範例 (MongoDB Shell)

此範例示範如何在以 100 個套件運送時，使用 $mod 運算子來判斷剩餘小工具的數量。

建立範例文件

```
db.widgets.insertMany([
  { "_id" : 1, "widget" : "A", "count" : 80372 },
```

```
  { "_id" : 2, "widget" : "B", "count" : 409282 },
  { "_id" : 3, "widget" : "C", "count" : 60739 }
])
```

查詢範例

```
db.widgets.aggregate([
  { $addFields: { leftOver: { $mod: [ "$count", 100 ] } } }
])
```

輸出

```
[
  { "_id" : 1, "widget" : "A", "count" : 80372, "leftOver" : 72 },
  { "_id" : 2, "widget" : "B", "count" : 409282, "leftOver" : 82 },
  { "_id" : 3, "widget" : "C", "count" : 60739, "leftOver" : 39 }
]
```

輸出顯示每個文件的其餘count部分除以 100，代表以 100 套件運送時剩餘的小工具數量。

程式碼範例

若要檢視使用 $mod命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const widgets = db.collection('widgets');

  await widgets.insertMany([
    { "_id" : 1, "widget" : "A", "count" : 80372 },
    { "_id" : 2, "widget" : "B", "count" : 409282 },
    { "_id" : 3, "widget" : "C", "count" : 60739 }
  ]);
```

```
  const result = await widgets.aggregate([
    { $addFields: { leftOver: { $mod: [ "$count", 100 ] } } }
  ]).toArray();

  console.log(result);
  client.close();
}

main();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
widgets = db['widgets']

widgets.insert_many([
    { "_id" : 1, "widget" : "A", "count" : 80372 },
    { "_id" : 2, "widget" : "B", "count" : 409282 },
    { "_id" : 3, "widget" : "C", "count" : 60739 }
])

result = list(widgets.aggregate([
    { "$addFields": { "leftOver": { "$mod": [ "$count", 100 ] } } }
]))

print(result)
client.close()
```

$month

Amazon DocumentDB 中的$month運算子會以 1 到 12 之間的數字傳回日期的月份。此運算子適用於從日期欄位擷取月份元件，以及執行以日期為基礎的彙總和分析。

參數

- `date_expression`：這是包含您要從中擷取月份之日期或時間戳記的表達式或欄位。

範例 (MongoDB Shell)

下列範例示範如何使用 $month運算子從日期欄位中擷取月份，並依月份分組資料。

建立範例文件

```
db.sales.insert([
  { product: "abc123", price: 10.99, date: new Date("2022-01-15") },
  { product: "def456", price: 15.50, date: new Date("2022-02-28") },
  { product: "ghi789", price: 8.25, date: new Date("2022-03-10") },
  { product: "jkl012", price: 12.75, date: new Date("2022-04-05") },
  { product: "mno345", price: 18.99, date: new Date("2022-05-20") }
]);
```

查詢範例

```
db.sales.aggregate([
  { $group: {
      _id: { month: { $month: "$date" } },
      totalSales: { $sum: "$price" }
    }},
  { $sort: { "_id.month": 1 } }
]);
```

輸出

```
[
  { _id: { month: 1 }, totalSales: 10.99 },
  { _id: { month: 2 }, totalSales: 15.5 },
  { _id: { month: 3 }, totalSales: 8.25 },
  { _id: { month: 4 }, totalSales: 12.75 },
  { _id: { month: 5 }, totalSales: 18.99 }
]
```

程式碼範例

若要檢視使用 $month命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function groupSalesByMonth() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('sales');

    const pipeline = [
      {
        $group: {
          _id: { month: { $month: "$date" } },
          totalSales: { $sum: "$price" }
        }
      },
      {
        $sort: { "_id.month": 1 }
      }
    ];

    const results = await collection.aggregate(pipeline).toArray();

    console.dir(results, { depth: null });

  } finally {
    await client.close();
  }
}

groupSalesByMonth().catch(console.error);
```

Python

```
from pymongo import MongoClient

def group_sales_by_month():

    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
    try:
        db = client.test
        collection = db.sales

        pipeline = [
            {
                "$group": {
                    "_id": { "$month": "$date" },
                    "totalSales": { "$sum": "$price" }
                }
            },
            {
                "$sort": { "_id": 1 }
            }
        ]

        results = collection.aggregate(pipeline)

        for doc in results:
            print(doc)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        client.close()

group_sales_by_month()
```

$multiply

Amazon DocumentDB 中的$multiply運算子用於將兩個或多個欄位或表達式的值相乘。此運算子對於對文件中的數值欄位執行算術操作特別有用。它可以用於彙總管道的各個階段，例如 $project和 $addFields，以建立新的欄位或修改現有的欄位。

參數

- expression1：要乘以的第一個數值表達式。
- expression2：要乘以的第二個數值表達式。
- [expression3, ...]：（選用）要乘以的其他數值表達式。

範例 (MongoDB Shell)

下列範例示範使用 $multiply來計算乘bonus_miles以 base_miles，以及bonus_rate針對使用信用卡進行旅程的客戶。

文件範例

```
db.miles.insertMany([
{ "_id": 1, "customer_name": "Arnav Desai", "member_since":
 ISODate("1997-03-01T00:00:00Z"), "base_miles": 2500, "bonus_rate": 1.8, "credit_card":
 true, "trip_cost": 250 },
{ "_id": 2, "customer_name": "Jorge Souza", "member_since":
 ISODate("2004-01-10T00:00:00Z"), "base_miles": 1890, "bonus_rate": 1.4, "credit_card":
 true, "trip_cost": 189 },
{ "_id": 3, "customer_name": "Saanvi Sarkar", "member_since":
 ISODate("1999-11-22T00:00:00Z"), "base_miles": 3250, "bonus_rate": 1.8, "credit_card":
 false, "trip_cost": 325 },
{ "_id": 4, "customer_name": "Paulo Santos", "member_since":
 ISODate("2021-06-19T00:00:00Z"), "base_miles": 2980, "bonus_rate": 1.2, "credit_card":
 true, "trip_cost": 298 },
{ "_id": 5, "customer_name": "Wang Xiulan", "member_since":
 ISODate("1995-12-04T00:00:00Z"), "base_miles": 1350, "bonus_rate": 1.9, "credit_card":
 false, "trip_cost": 135 }
]);
```

查詢範例

```
db.miles.aggregate([
  {
    $match: { credit_card: true }
  },
  {
    $project: {
      customer_name: 1,
      base_miles: 1,
      bonus_rate:1,
      credit_card: 1,
      total_miles: {
        $multiply: ["$base_miles", "$bonus_rate"]
      }
    }
  }
]);
```

輸出

```
[
  { _id: 1, customer_name: 'Arnav Desai', base_miles: 12500, bonus_rate: 1.8,
 credit_card: true, total_miles: 22500 },
  { _id: 3, customer_name: 'Saanvi Sarkar',base_miles: 15200, bonus_rate: 1.8,
 credit_card: true, total_miles: 27360 },
  { _id: 4, customer_name: 'Paulo Santos', base_miles: 3400, bonus_rate: 1.1,
 credit_card: true, total_miles: 3740 }
]
```

程式碼範例

若要檢視使用 $multiply命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function multiplyBonusMiles() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('miles');

  const result = await collection.aggregate([
    { $match: { credit_card: true } },
    {
      $project: {
        customer_name: 1,
        base_miles: 1,
        bonus_rate: 1,
        credit_card: 1,
        total_miles: {
          $multiply: ["$base_miles", "$bonus_rate"]
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
```

```
}

multiplyBonusMiles();
```

Python

```
from pymongo import MongoClient

def multiply_bonus_miles():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['miles']

    result = list(collection.aggregate([
        {'$match': {'credit_card': True}},
        {
            '$project': {
                'customer_name': 1,
                'base_miles': 1,
                'bonus_rate': 1,
                'credit_card': 1,
                'total_miles': {
                    '$multiply': ['$base_miles', '$bonus_rate']
                }
            }
        }
    ]))

    print(result)
    client.close()

multiply_bonus_miles()
```

$自然

Amazon DocumentDB 中的$natural運算子用於依其自然順序排序文件，這是它們插入集合的順序。這與預設排序行為相反，也就是根據指定欄位的值排序文件。

參數

無

範例 (MongoDB Shell)

下列範例示範如何使用 $natural運算子，以自然順序排序集合中的文件。

建立範例文件

```
db.people.insertMany([
  { "_id": 1, "name": "María García", "age": 28 },
  { "_id": 2, "name": "Arnav Desai", "age": 32 },
  { "_id": 3, "name": "Li Juan", "age": 25 },
  { "_id": 4, "name": "Carlos Salazar", "age": 41 },
  { "_id": 5, "name": "Sofia Martínez", "age": 35 }
]);
```

查詢範例

```
db.people.find({}, { "_id": 1, "name": 1 }).sort({ "$natural": 1 });
```

輸出

```
[
  { "_id": 1, "name": "María García" },
  { "_id": 2, "name": "Arnav Desai" },
  { "_id": 3, "name": "Li Juan" },
  { "_id": 4, "name": "Carlos Salazar" },
  { "_id": 5, "name": "Sofia Martínez" }
]
```

查詢會以其自然順序對集合中的文件進行排序，這是它們插入的順序。

程式碼範例

若要檢視使用 $natural命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const documents = await collection.find({}, { projection: { _id: 1, name: 1 } })
    .sort({ $natural: 1 })
    .toArray();

  console.log(documents);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['people']

    documents = list(collection.find({}, {'_id': 1, 'name': 1}).sort('$natural', 1))
    print(documents)

    client.close()

example()
```

$ne

$ne 彙總運算子會比較兩個值，並在它們不相等true時傳回 ，否則傳回 false。

參數

- expression1：要比較的第一個值。

- expression2：要比較的第二個值。

範例 (MongoDB Shell)

下列範例示範如何使用 $ne 運算子來識別具有狀態變更的訂單。

建立範例文件

```
db.orders.insertMany([
  { _id: 1, orderId: "A123", status: "shipped", expectedStatus: "shipped" },
  { _id: 2, orderId: "B456", status: "pending", expectedStatus: "shipped" },
  { _id: 3, orderId: "C789", status: "delivered", expectedStatus: "delivered" }
]);
```

查詢範例

```
db.orders.aggregate([
  {
    $project: {
      orderId: 1,
      status: 1,
      expectedStatus: 1,
      needsAttention: { $ne: ["$status", "$expectedStatus"] }
    }
  }
]);
```

輸出

```
[
  { _id: 1, orderId: 'A123', status: 'shipped', expectedStatus: 'shipped',
 needsAttention: false },
  { _id: 2, orderId: 'B456', status: 'pending', expectedStatus: 'shipped',
 needsAttention: true },
  { _id: 3, orderId: 'C789', status: 'delivered', expectedStatus: 'delivered',
 needsAttention: false }
]
```

程式碼範例

若要檢視使用$ne彙總運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('orders');

  const result = await collection.aggregate([
    {
      $project: {
        orderId: 1,
        status: 1,
        expectedStatus: 1,
        needsAttention: { $ne: ["$status", "$expectedStatus"] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['orders']

    result = list(collection.aggregate([
        {
            '$project': {
                'orderId': 1,
```

```
                'status': 1,
                'expectedStatus': 1,
                'needsAttention': { '$ne': ['$status', '$expectedStatus'] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$not

$not 彙總運算子會對表達式執行邏輯 NOT 操作。true 如果表達式評估為 false，且false如果表達式評估為 ，則傳回 true。

參數

- expression：要否定的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $not運算子反轉布林值。

建立範例文件

```
db.users.insertMany([
  { _id: 1, name: "Alice", active: true },
  { _id: 2, name: "Bob", active: false },
  { _id: 3, name: "Charlie", active: true }
]);
```

查詢範例

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      active: 1,
      inactive: { $not: ["$active"] }
```

```
    }
  }
]);
```

輸出

```
[
  { _id: 1, name: 'Alice', active: true, inactive: false },
  { _id: 2, name: 'Bob', active: false, inactive: true },
  { _id: 3, name: 'Charlie', active: true, inactive: false }
]
```

程式碼範例

若要檢視使用$not彙總運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('users');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
        active: 1,
        inactive: { $not: ["$active"] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['users']

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'active': 1,
                'inactive': { '$not': ['$active'] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$objectToArray

Amazon DocumentDB $objectToArray 中的彙總運算子會將物件 （或文件） 轉換為陣列。運算子的輸入是文件，而輸出包含輸入文件中每個欄位值對的陣列元素。當您需要使用文件的個別欄位做為陣列時，例如當您想要尋找具有特定欄位最大值或最小值的文件時，此運算子非常有用。

參數

- expression：要轉換為陣列的文件表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $objectToArray運算子來尋找具有影片租用存放區鏈結庫存上限的文件。

建立範例文件

```
db.videos.insertMany([
  {
    "_id": 1,
    "name": "Live Soft",
    "inventory": {
      "Des Moines": 1000,
      "Ames": 500
    }
  },
  {
    "_id": 2,
    "name": "Top Pilot",
    "inventory": {
      "Mason City": 250,
      "Des Moines": 1000
    }
  },
  {
    "_id": 3,
    "name": "Romancing the Rock",
    "inventory": {
      "Mason City": 250,
      "Ames": 500
    }
  },
  {
    "_id": 4,
    "name": "Bravemind",
    "inventory": {
      "Mason City": 250,
      "Des Moines": 1000,
      "Ames": 500
    }
  }
]);
```

查詢範例

```
db.videos.aggregate([
  {
    $project: {
      name: 1,
      videos: {
```

```
        $objectToArray: "$inventory"
      }
    }
  },
  {
    $unwind: "$videos"
  },
  {
    $group: {
      _id: "$name",
      maxInventory: {
        $max: "$videos.v"
      }
    }
  }
]);
```

輸出

```
[
  {
    "_id": "Bravemind",
    "maxInventory": 1000
  },
  {
    "_id": "Live Soft",
    "maxInventory": 1000
  },
  {
    "_id": "Romancing the Rock",
    "maxInventory": 500
  },
  {
    "_id": "Top Pilot",
    "maxInventory": 1000
  }
]
```

程式碼範例

若要檢視使用 $objectToArray命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findMaxInventory() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const videos = db.collection('videos');

  const result = await videos.aggregate([
    {
      $project: {
        name: 1,
        videos: {
          $objectToArray: "$inventory"
        }
      }
    },
    {
      $unwind: "$videos"
    },
    {
      $group: {
        _id: "$name",
        maxInventory: {
          $max: "$videos.v"
        }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

findMaxInventory();
```

Python

```
from pymongo import MongoClient
```

```
def find_max_inventory():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    videos = db['videos']

    result = list(videos.aggregate([
        {
            '$project': {
                'name': 1,
                'videos': {
                    '$objectToArray': '$inventory'
                }
            }
        },
        {
            '$unwind': '$videos'
        },
        {
            '$group': {
                '_id': '$name',
                'maxInventory': {
                    '$max': '$videos.v'
                }
            }
        }
    ]))

    print(result)
    client.close()

find_max_inventory()
```

$or

彙總運算子會評估多個表達式，如果至少一個表達式評估為 true，則傳回 $or true。false 只有在所有表達式都是 時，才會傳回 false。

參數

- expressions：要評估的表達式陣列。

範例 (MongoDB Shell)

下列範例示範如何使用 $or 運算子來檢查產品是否符合任何一個條件。

建立範例文件

```
db.items.insertMany([
  { _id: 1, name: "Widget", price: 150, onSale: false },
  { _id: 2, name: "Gadget", price: 45, onSale: false },
  { _id: 3, name: "Tool", price: 200, onSale: true }
]);
```

查詢範例

```
db.items.aggregate([
  {
    $project: {
      name: 1,
      price: 1,
      onSale: 1,
      goodDeal: {
        $or: [
          { $lt: ["$price", 50] },
          { $eq: ["$onSale", true] }
        ]
      }
    }
  }
]);
```

輸出

```
[
  { _id: 1, name: 'Widget', price: 150, onSale: false, goodDeal: false },
  { _id: 2, name: 'Gadget', price: 45, onSale: false, goodDeal: true },
  { _id: 3, name: 'Tool', price: 200, onSale: true, goodDeal: true }
]
```

程式碼範例

若要檢視使用$or彙總運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('items');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
        price: 1,
        onSale: 1,
        goodDeal: {
          $or: [
            { $lt: ["$price", 50] },
            { $eq: ["$onSale", true] }
          ]
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['items']
```

```
    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'price': 1,
                'onSale': 1,
                'goodDeal': {
                    '$or': [
                        { '$lt': ['$price', 50] },
                        { '$eq': ['$onSale', True] }
                    ]
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$out

Amazon DocumentDB 中的$out運算子用於將彙總管道的結果寫入指定的集合。

$out 應該是管道中的最後一個階段。

參數

- output_collection：要寫入彙總結果的輸出集合名稱。

注意：如果集合已存在，則會將其取代為彙總階段的結果。

範例 (MongoDB Shell)

下列範例示範如何在 Amazon DocumentDB 中使用 $out運算子，將彙總管道的結果寫入新集合。

建立範例文件

```
db.products.insertMany([
  { _id: 1, name: "Wireless Headphones", category: "Electronics", price: 100.0 },
```

```
  { _id: 2, name: "Smartphone", category: "Electronics", price: 200.0 },
  { _id: 3, name: "JavaScript Guide", category: "Books", price: 50.0 },
  { _id: 4, name: "Database Design Handbook", category: "Books", price: 75.0 }
]);
```

查詢範例

```
db.products.aggregate([
  { $group: { _id: "$category", totalPrice: { $sum: "$price" } } },
  { $out: "product_categories" }
])
```

輸出

無（結果會寫入輸出集合）。

彙總管道會依類別將產品分組，並計算每個類別的項目總價。$out 運算子會將結果寫入名為 "product_categories" 的新集合。

若要檢視輸出集合中的結果：

```
db.product_categories.find()
[
{ "_id" : "Books", "totalPrice" : 125 },
{ "_id" : "Electronics", "totalPrice" : 300 }
]
```

程式碼範例

若要檢視使用 $out命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function demo_out_operator() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const products = db.collection('products');
```

```
  // Execute aggregation with $out - results are stored in 'product_categories'
 collection
  await products.aggregate([
    { $group: { _id: "$category", totalPrice: { $sum: "$price" } } },
    { $out: "product_categories" }
  ]).toArray();

  // Retrieve the results from the output collection (limited to 20 records)
  const productCategories = db.collection('product_categories');
  const results = await productCategories.find({}).limit(20).toArray();

  console.log('Results stored in product_categories collection:', results);
  await client.close();
}

demo_out_operator();
```

Python

```
from pymongo import MongoClient

def demo_out_operator():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    products = db['products']

    # Execute aggregation with $out - results are stored in 'product_categories'
 collection
    list(products.aggregate([
        { '$group': { '_id': '$category', 'totalPrice': { '$sum': '$price' } } },
        { '$out': 'product_categories' }
    ]))

    # Retrieve the results from the output collection (limited to 20 records)
    product_categories = db['product_categories']
    results = list(product_categories.find({}).limit(20))

    print('Results stored in product_categories collection:', results)
    client.close()

demo_out_operator()
```

$percentile

8.0.1 版的新功能。

Amazon DocumentDB 中的$percentile運算子會計算數值資料的指定百分位數值。作為累積器，它會在彙總管道$group階段的 群組內跨文件計算百分位數值。做為表達式，它會計算數字陣列的百分位數。

參數

- input：解析為數值或數值陣列的表達式。
- p：介於 0 和 1 之間的百分位數值陣列，其中每個值代表要計算的百分位數。例如， 會[0.25, 0.5, 0.75]計算第 25、第 50 和第 75 個百分位數。
- method：指定計算方法的字串。目前僅支援 "approximate"。

Behavior (行為)

"approximate" 方法使用 t-digest 演算法來計算以百分位數為基礎的近似指標。使用小型資料集時，不同的百分位數值可能會解析為相同的結果。例如，每個群組只有 5 個值，p90 和 p99 可能會同時傳回群組中的最大值。精確度會隨著資料點數量的增加而提高。

範例 (MongoDB Shell)

下列範例顯示如何使用 $percentile 運算子計算每個類別分數的第 25 百分位數和第 75 百分位數。

建立範例文件

```
db.students.insertMany([
  { class: "A", score: 72 },
  { class: "A", score: 85 },
  { class: "A", score: 90 },
  { class: "A", score: 68 },
  { class: "A", score: 95 },
  { class: "B", score: 80 },
  { class: "B", score: 75 },
  { class: "B", score: 92 },
  { class: "B", score: 88 },
  { class: "B", score: 70 }
]);
```

查詢範例

```
db.students.aggregate([
  { $group: {
      _id: "$class",
      percentiles: { $percentile: { input: "$score", p: [0.25, 0.75], method:
 "approximate" } }
    }}
]);
```

輸出

```
[
  { "_id": "A", "percentiles": [72, 90] },
  { "_id": "B", "percentiles": [75, 88] }
]
```

表達式用量範例 (MongoDB Shell)

`$percentile` 運算子也可以用作`$project`階段內的表達式，以計算陣列欄位的百分位數。

建立範例文件

```
db.surveys.insertMany([
  { _id: 1, responses: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20] },
  { _id: 2, responses: [1, 3, 5, 7, 9, 11, 13, 15, 17, 19] }
]);
```

查詢範例

```
db.surveys.aggregate([
  { $project: {
      quartiles: { $percentile: { input: "$responses", p: [0.25, 0.5, 0.75], method:
 "approximate" } }
    }}
]);
```

輸出

```
[
  { "_id": 1, "quartiles": [6, 10, 16] },
  { "_id": 2, "quartiles": [5, 9, 15] }
```

```
]
```

程式碼範例

若要檢視使用 $percentile 運算子的程式碼範例，請選擇您要使用的語言標籤。下列範例顯示累積器用量 （在 中$group) 和表達式用量 （在 中$project)：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();
    const db = client.db('test');

    // Accumulator usage: percentiles across grouped documents
    const students = db.collection('students');
    const accumulatorResult = await students.aggregate([
      { $group: {
          _id: "$class",
          percentiles: { $percentile: { input: "$score", p: [0.25, 0.75], method:
 "approximate" } }
        }}
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Expression usage: percentiles of an array field
    const surveys = db.collection('surveys');
    const expressionResult = await surveys.aggregate([
      { $project: {
          quartiles: { $percentile: { input: "$responses", p: [0.25, 0.5, 0.75],
 method: "approximate" } }
        }}
    ]).toArray();
    console.log('Expression result:', expressionResult);

  } finally {
```

```
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']

        # Accumulator usage: percentiles across grouped documents
        students = db['students']
        accumulator_result = list(students.aggregate([
            { '$group': {
                '_id': '$class',
                'percentiles': { '$percentile': { 'input': '$score', 'p': [0.25,
 0.75], 'method': 'approximate' } }
            }}
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: percentiles of an array field
        surveys = db['surveys']
        expression_result = list(surveys.aggregate([
            { '$project': {
                'quartiles': { '$percentile': { 'input': '$responses', 'p': [0.25,
 0.5, 0.75], 'method': 'approximate' } }
            }}
        ]))
        print('Expression result:', expression_result)

    finally:
        client.close()

example()
```

$pow

在 8.0 中推出

Amazon DocumentDB 中的$pow運算子可讓您將數字提升為電源。這對於在彙總管道中執行指數計算很有用。

參數

- <number>（必要）：要提升為電源的數字。
- <exponent>（必要）：應提高數字的功率。

範例 (MongoDB Shell)

下列範例示範如何使用 $pow運算子來計算數字的平方。

建立範例文件

```
db.numbers.insertMany([
  { "_id": 1, "value": 2 },
  { "_id": 2, "value": 3 },
  { "_id": 3, "value": 4 }
]);
```

查詢範例

```
db.numbers.aggregate([
  { $addFields: { "square": { $pow: ["$value", 2] } } }
])
```

輸出

```
[
  { "_id": 1, "value": 2, "square": 4 },
  { "_id": 2, "value": 3, "square": 9 },
  { "_id": 3, "value": 4, "square": 16 }
]
```

程式碼範例

若要檢視使用 $pow命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

以下是在 Node.js 應用程式中使用 $pow 運算子的範例：

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $addFields: { "square": { $pow: ["$value", 2] } } }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

以下是在 Python 應用程式中使用 $pow 運算子的範例：

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['numbers']

    result = list(collection.aggregate([
        { "$addFields": { "square": { "$pow": ["$value", 2] } } }
    ]))

    print(result)
    client.close()
```

```
example()
```

$push

$push 彙總運算子會從每個群組的指定表達式傳回所有值的陣列。它通常在$group階段內用來將值累積到陣列中。

參數

- expression：要評估群組中每個文件的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $push運算子來收集每個類別的所有產品名稱。

建立範例文件

```
db.sales.insertMany([
  { _id: 1, category: "Electronics", product: "Laptop", amount: 1200 },
  { _id: 2, category: "Electronics", product: "Mouse", amount: 25 },
  { _id: 3, category: "Furniture", product: "Desk", amount: 350 },
  { _id: 4, category: "Furniture", product: "Chair", amount: 150 },
  { _id: 5, category: "Electronics", product: "Keyboard", amount: 75 }
]);
```

查詢範例

```
db.sales.aggregate([
  {
    $group: {
      _id: "$category",
      products: { $push: "$product" }
    }
  }
]);
```

輸出

```
[
  { _id: 'Furniture', products: [ 'Desk', 'Chair' ] },
```

```
  { _id: 'Electronics', products: [ 'Laptop', 'Mouse', 'Keyboard' ] }
]
```

程式碼範例

若要檢視使用$push彙總運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('sales');

  const result = await collection.aggregate([
    {
      $group: {
        _id: "$category",
        products: { $push: "$product" }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['sales']
```

```
    result = list(collection.aggregate([
        {
            '$group': {
                '_id': '$category',
                'products': { '$push': '$product' }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$project

Amazon DocumentDB 中的$project運算子可讓您選擇性地從輸出文件中包含或排除欄位、將值傳遞至下一個管道階段，以及從輸入文件值運算新欄位。

參數

- field：要從輸出文件中包含或排除的欄位，可以是欄位路徑 （ 例如 "a.b.c")。
- 1 或 true：在輸出中包含 欄位。
- 0 或 false：從輸出中排除 欄位。

範例 (MongoDB Shell)

下列範例示範在學生集合上使用 $project 運算子

建立範例文件

```
db.students.insertMany([
  { "_id": 1, "name": "Alejandro Rosalez", "math": 85, "science": 92, "grade": "A" },
  { "_id": 2, "name": "Carlos Salazar", "math": 78, "science": 84, "grade": "B" },
  { "_id": 3, "name": "Nikhil Jayashankar", "math": 95, "science": 89, "grade": "A" },
  { "_id": 4, "name": "Shirley Rodriguez", "math": 72, "science": 76, "grade": "B" }
  ]);
```

此查詢僅包含輸出中的 name和 math 欄位。除非明確排除，否則預設會包含 _id 欄位。

```
db.students.aggregate([
  { $project: { "name": 1, "math": 1 } }
])
```

輸出

```
{ _id: 1, name: "Alejandro Rosalez", math: 85 }
{ _id: 2, name: "Carlos Salazar", math: 78 }
{ _id: 3, name: "Nikhil Jayashankar", math: 95 }
{ _id: 4, name: "Shirley Rodriguez", math: 72 }
```

此查詢會從輸出中排除 grade和 _id 欄位，顯示所有其他欄位 (name、math、science)。

```
db.students.aggregate([
  { $project: { "grade": 0, "_id": 0 } }
])
```

輸出

```
{ name: "Alejandro Rosalez", math: 85, science: 92 }
{ name: "Carlos Salazar", math: 78, science: 84 }
{ name: "Nikhil Jayashankar", math: 95, science: 89 }
{ name: "Shirley Rodriguez", math: 72, science: 76 }
```

程式碼範例

若要檢視使用 $project命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('students');

  const result = await collection.aggregate([
    { $project: { "name": 1, "math": 1 } }
```

```
  ]).toArray();
  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['students']

    result = list(collection.aggregate([
        { '$project': { 'name': 1, 'math': 1 } }
    ]))
    print(result)

    client.close()

example()
```

$radiansToDegrees

8.0.1 版的新功能。

Amazon DocumentDB 中的$radiansToDegrees運算子會將值從弧度轉換為度數。

參數

- expression：解析為弧度數值的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $radiansToDegrees運算子將弧度值轉換為度數。

建立範例文件

```
db.angles.insertMany([
  { "_id": 1, "radians": 0 },
  { "_id": 2, "radians": 1.5707963267948966 },
  { "_id": 3, "radians": 3.141592653589793 },
  { "_id": 4, "radians": 6.283185307179586 }
]);
```

查詢範例

```
db.angles.aggregate([
  { $project: {
    "degrees": { $radiansToDegrees: "$radians" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "degrees": 0 },
  { "_id": 2, "degrees": 90 },
  { "_id": 3, "degrees": 180 },
  { "_id": 4, "degrees": 360 }
]
```

程式碼範例

若要檢視使用 $radiansToDegrees 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('angles');
```

```
  const result = await collection.aggregate([
    { $project: {
      "degrees": { $radiansToDegrees: "$radians" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['angles']

    result = list(collection.aggregate([
        { '$project': {
            'degrees': { '$radiansToDegrees': '$radians' }
        }}
    ]))

    print(result)
    client.close()

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

$rand

8.0 版的新功能

Amazon DocumentDB 中的$rand運算子用於產生介於 0 和 1 之間的隨機數字。

參數

無

範例 (MongoDB Shell)

下列範例示範如何使用 $rand運算子從temp集合中隨機選取兩個文件。

建立範例文件

```
db.items.insertMany([
  { "name": "pencil", "quantity": 110 },
  { "name": "pen", "quantity": 159 }
])
```

查詢範例

```
db.items.aggregate([
  {
    $project: {
      randomValue: { $rand: {} }
    }
  }
])
```

輸出

```
[
  {
    _id: ObjectId('6924a5edd66dcae121d29517'),
    randomValue: 0.8615243955294392
  },
  {
    _id: ObjectId('6924a5edd66dcae121d29518'),
    randomValue: 0.22815483022099903
  }
]
```

程式碼範例

若要檢視使用 $rand命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('items');

  const result = await collection.aggregate([
    {
      $project: {
        randomValue: { $rand: {} }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
  client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
  db = client['test']
  collection = db['items']

  result = list(collection.aggregate([
      {
        "$project": {
          "randomValue": { "$rand": {} }
        }
      }
  ]))

  print(result)
```

```
  client.close()

example()
```

$range

Amazon DocumentDB $range 中的彙總運算子用於建立指定範圍內的連續數字陣列。此運算子特別適用於產生數字序列，例如賽車中輔助站的英里標記，如下列範例所示。

參數

- start：範圍的起始值。
- end：範圍的結束值。
- step：（選用）產生範圍時要使用的步驟值。如果未提供，則預設步驟值為 1。

範例 (MongoDB Shell)

在此範例中，我們將使用 $range運算子為自行車競賽中的水站產生英里標記。

建立範例文件

```
db.races.insertMany([
  { _id: 0, race: "STP", distance: 206 },
  { _id: 1, race: "RSVP", distance: 160 },
  { _id: 2, race: "Chilly Hilly", distance: 33 },
  { _id: 3, race: "Flying Wheels", distance: 100 }
]);
```

查詢範例

```
db.races.aggregate([
  {
    $project: {
      race: 1,
      "waterStations": { $range: [20, "$distance", 20] }
    }
  }
]);
```

輸出

```
[
  {
    _id: 0,
    race: 'STP',
    waterStations: [
       20,  40,  60,  80,
      100, 120, 140, 160,
      180, 200
    ]
  },
  {
    _id: 1,
    race: 'RSVP',
    waterStations: [
       20,  40,  60, 80,
      100, 120, 140
    ]
  },
  { _id: 2, race: 'Chilly Hilly', waterStations: [ 20 ] },
  { _id: 3, race: 'Flying Wheels', waterStations: [ 20, 40, 60, 80 ] }
]
```

程式碼範例

若要檢視使用 $range命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('races');

    const pipeline = [
```

```
      {
        $project: {
          race: 1,
          waterStations: { $range: [20, "$distance", 20] }
        }
      }
    ];

    const results = await collection.aggregate(pipeline).toArray();

    console.dir(results, { depth: null });

  } finally {
    await client.close();
  }
}

example().catch(console.error);
```

Python

```
from pymongo import MongoClient

def example():

  client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

  try:
      db = client.test
      collection = db.races

      pipeline = [
          {
              "$project": {
                  "race": 1,
                  "waterStations": { "$range": [20, "$distance", 20] }
              }
          }
      ]

      results = collection.aggregate(pipeline)
```

```
        for doc in results:
            print(doc)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        client.close()

example()
```

$redact

Amazon DocumentDB $redact 中的彙總階段用於根據指定欄位的值，選擇性地包含或排除輸出文件中的內容。這在您需要根據存取層級或使用者許可控制敏感資料的可見性的情況下特別有用。

參數

- $cond：評估文件中$$DESCEND每個欄位 $$KEEP、 $$PRUNE或 的表達式。
- $$KEEP：保留輸出文件中的目前欄位。
- $$PRUNE：從輸出文件移除目前欄位。
- $$DESCEND：遞迴地將$redact階段套用至目前欄位，也就是物件或陣列。

範例 (MongoDB Shell)

在此範例中，我們將使用 $redact階段根據訂單狀態篩選訂單，只顯示具有特定狀態值的訂單。

建立範例文件

```
db.orders.insert([
  { "_id": 1, "status": "shipped", "customer": "Carlos Salazar", "total": 150.00,
 "date": "2025-01-15" },
  { "_id": 2, "status": "processing", "customer": "Saanvi Sarkar", "total": 89.99,
 "date": "2025-01-20" },
  { "_id": 3, "status": "cancelled", "customer": "Zhang Wei", "total": 220.50, "date":
 "2025-01-18" }
])
```

查詢範例

```
db.orders.aggregate([
  {
    $redact: {
      $cond: {
        if: { $in: ["$status", ["shipped", "processing"]] },
        then: "$$KEEP",
        else: "$$PRUNE"
      }
    }
  }
])
```

輸出

```
[
  { _id: 1, status: 'shipped', customer: 'Carlos Salazar', total: 150, date:
 '2025-01-15' },
  { _id: 2, status: 'processing', customer: 'Saanvi Sarkar', total: 89.99, date:
 '2025-01-20' }
]
```

在此範例中，$redact階段會檢查每個文件中 status 欄位的值。如果status「已運送」或「正在處理」，文件會保留 ($$KEEP)。否則，文件會遭到刪除 ($$PRUNE)。

程式碼範例

若要檢視使用 $redact命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('orders');

  const result = await collection.aggregate([
    {
      $redact: {
```

```
        $cond: {
          if: { $in: ['$status', ['shipped', 'processing']] },
          then: '$$KEEP',
          else: '$$PRUNE'
        }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['orders']

    result = list(collection.aggregate([
        {
            '$redact': {
                '$cond': {
                    'if': { '$in': ['$status', ['shipped', 'processing']] },
                    'then': '$$KEEP',
                    'else': '$$PRUNE'
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$reduce

Amazon DocumentDB $reduce 中的彙總運算子用於將兩個引數的函數累加到陣列的元素，以將陣列減少為單一值。此運算子特別適用於對彙總管道中的陣列資料執行複雜的計算或轉換。

參數

- input：要減少的陣列。
- initialValue：要在縮減操作中使用的初始值。
- in：要在input陣列的每個元素上評估的表達式。此表達式應該會傳回值，用於下一次的縮減反覆運算。

範例 (MongoDB Shell)

下列範例示範如何使用 $reduce運算子來計算陣列中所有元素的總和。

建立範例文件

```
db.orders.insertMany([
  { "_id": 1, "items": [1, 2, 3, 4, 5] },
  { "_id": 2, "items": [10, 20, 30] },
  { "_id": 3, "items": [5, 15, 25, 35] },
  { "_id": 4, "items": [100, 200] }
])
```

查詢範例

```
db.orders.aggregate([
  {
    $project: {
      total: {
        $reduce: {
          input: "$items",
          initialValue: 0,
          in: { $add: ["$$value", "$$this"] }
        }
      }
    }
  }
])
```

輸出

```
[
  { "_id": 1, "total": 15 },
  { "_id": 2, "total": 60 },
  { "_id": 3, "total": 80 },
  { "_id": 4, "total": 300 }
]
```

$reduce 運算子會反覆運算items陣列，將每個元素新增至 0 initialValue的 。結果是陣列中所有元素的總和。

程式碼範例

若要檢視使用 $reduce命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

以下是在 Node.js 應用程式中使用 $reduce 運算子的範例：

```
const { MongoClient } = require("mongodb");

async function main() {
  const client = await MongoClient.connect("mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false");
  const db = client.db("test");
  const orders = db.collection("orders");

  const result = await orders.aggregate([
    {
      $project: {
        total: {
          $reduce: {
            input: "$items",
            initialValue: 0,
            in: { $add: ["$$value", "$$this"] }
          }
        }
      }
    }
  ]).toArray();
```

```
  console.log(result);
  await client.close();
}

main();
```

Python

以下是在 python 應用程式中使用 $reduce運算子的範例：

```
from pymongo import MongoClient

def main():
    client = MongoClient("mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false")
    db = client["test"]
    orders = db["orders"]

    result = list(orders.aggregate([
        {
            "$project": {
                "total": {
                    "$reduce": {
                        "input": "$items",
                        "initialValue": 0,
                        "in": { "$add": ["$$value", "$$this"] }
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

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

$regexFind

5.0 版的新功能。

Elastic 叢集不支援。

Amazon DocumentDB 中的$regexFind運算子用於對文件中的字串欄位執行規則運算式比對。它可讓您搜尋和擷取符合指定規則表達式模式的特定子字串。

參數

- input：要搜尋的字串欄位或表達式。
- regex：要比對的規則表達式模式。
- options：（選用）指定規則表達式選用參數的物件，例如區分大小寫和多行比對。支援的選項為 i（不區分大小寫）和 m（多行）。

範例 (MongoDB Shell)

下列範例示範如何使用 $regexFind運算子來搜尋 name 欄位符合特定規則表達式模式的文件。

建立範例文件

```
db.users.insertMany([
  { "_id": 1, name: "John Doe", email: "john@example.com" },
  { "_id": 2, name: "Diego Ramirez", email: "diego@example.com" },
  { "_id": 3, name: "Alejandro Rosalez", email: "alejandro@example.com" },
  { "_id": 4, name: "Shirley Rodriguez", email: "shirley@example.com" }
]);
```

查詢範例

```
db.users.aggregate([
  {
    $project: {
      names: {
        $regexFind: { input: '$name', regex: 'j', options: 'i' }
      }
    }
  },
  { $match: {names: {$ne: null}}}
])
```

此查詢將傳回 name 欄位包含字母 "j"（不區分大小寫）的所有文件。

輸出

```
[
  { _id: 1, names: { match: 'J', idx: 0, captures: [] } }
]
```

注意：如果您的查詢使用 Amazon DocumentDB 規劃器第 1 版，您必須使用提示來使用索引。如果沒有提示，查詢可能會執行集合掃描。若要檢查您的規劃器版本並進一步了解如何使用提示，請參閱 查詢規劃器 v2。

程式碼範例

若要檢視使用 $regexFind命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const results = await users.aggregate([
    { $project: { names: { $regexFind: { input: "$name", regex: "john", options:
 "i" }}}},
    { $match: {names: {$ne: null}}}
  ]).toArray();


  console.log(results);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient
```

```
client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
users = db['users']

results = list(users.aggregate([
    { "$project": { "names": { "$regexFind": { "input": "$name", "regex": "john",
 "options": "i" }}}},
    { "$match": {"names": {"$ne": None}}}
]))

print(results)

client.close()
```

$regexFindAll

在 5.0 中推出

Amazon DocumentDB 中的$regexFindAll運算子用於對文件中的字串欄位執行規則運算式比對。它可讓您搜尋和擷取符合指定規則表達式模式的特定子字串，並傳回規則表達式的所有相符項目。

參數

- input：要搜尋的字串欄位或表達式。
- regex：要比對的規則表達式模式。
- options：（選用）指定規則表達式選用參數的物件，例如區分大小寫和多行比對。支援的選項為 i（不區分大小寫）和 m（多行）。

範例 (MongoDB Shell)

下列範例示範如何使用 $regexFindAll運算子從 email 欄位擷取所有字母序列。

建立範例文件

```
db.users.insertMany([
  { _id: 1, name: "John Doe", email: "john@example.com", phone: "555-1234" },
  { _id: 2, name: "Jane Roe", email: "jane@example.com", phone: "555-5678" },
  { _id: 3, name: "Carlos Salazar", email: "carlos@example.com", phone: "555-3456" },
  { _id: 4, name: "Saanvi Sarkar", email: "saanvi@example.com", phone: "555-7890" }
```

```
]);
```

查詢範例

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      emailMatches: {
        $regexFindAll: { input: '$email', regex: '[a-z]+', options: 'i' }
      }
    }
  }
])
```

輸出

```
[
  {
    _id: 1,
    name: 'John Doe',
    emailMatches: [
      { match: 'john', idx: 0, captures: [] },
      { match: 'example', idx: 5, captures: [] },
      { match: 'com', idx: 13, captures: [] }
    ]
  },
  {
    _id: 2,
    name: 'Jane Roe',
    emailMatches: [
      { match: 'jane', idx: 0, captures: [] },
      { match: 'example', idx: 5, captures: [] },
      { match: 'com', idx: 13, captures: [] }
    ]
  },
  {
    _id: 3,
    name: 'Carlos Salazar',
    emailMatches: [
      { match: 'carlos', idx: 0, captures: [] },
      { match: 'example', idx: 7, captures: [] },
```

```
      { match: 'com', idx: 15, captures: [] }
    ]
  },
  {
    _id: 4,
    name: 'Saanvi Sarkar',
    emailMatches: [
      { match: 'saanvi', idx: 0, captures: [] },
      { match: 'example', idx: 7, captures: [] },
      { match: 'com', idx: 15, captures: [] }
    ]
  }
]
```

注意：如果您的查詢使用 Amazon DocumentDB 規劃器第 1 版，您必須使用提示來使用索引。如果沒有提示，查詢可能會執行集合掃描。若要檢查您的規劃器版本並進一步了解如何使用提示，請參閱 查詢規劃器 v2。

程式碼範例

若要檢視使用 `$regexFindAll`命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

以下是在 Node.js 應用程式中使用 `$regexFind` 運算子的範例：

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const results = await users.aggregate([
    {
      $project: {
        name: 1,
        emailMatches: {
          $regexFindAll: { input: "$email", regex: "[a-z]+", options: "i" }
        }
      }
```

```
    }
  ]).toArray();

  console.log(JSON.stringify(results, null, 2));

  await client.close();
}

main();
```

Python

以下是在 Python 應用程式中使用 $regexFind 運算子的範例：

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
users = db['users']

results = list(users.aggregate([
    {
        "$project": {
            "name": 1,
            "emailMatches": {
                "$regexFindAll": {
                    "input": "$email",
                    "regex": "[a-z]+",
                    "options": "i"
                }
            }
        }
    }
]))

print(results)

client.close()
```

$regexMatch

5.0 版的新功能。Elastic 叢集不支援。

Amazon DocumentDB 中的$regexMatch運算子用於對字串欄位執行規則運算式比對。它會傳回布林值 (true 或 false)，指出輸入字串是否符合指定的模式。

參數

- input：針對規則表達式進行測試的字串。
- regex：要比對的規則表達式模式。
- options：（選用）修改規則表達式行為的旗標，例如不區分大小寫的比對 (i) 或多行比對 ()m。

範例 (MongoDB Shell)

下列範例示範如何使用 $regexMatch 運算子來檢查名稱開頭是否為字母 'M'。運算子false會為每個文件傳回 true或 。

建立範例文件

```
db.users.insertMany([
  { "_id":1, name: "María García", email: "maría@example.com" },
  { "_id":2, name: "Arnav Desai", email: "arnav@example.com" },
  { "_id":3, name: "Martha Rivera", email: "martha@example.com" },
  { "_id":4, name: "Richard Roe", email: "richard@example.com" },

]);
```

查詢範例

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      startsWithM: {
        $regexMatch: {
          input: "$name",
          regex: "^M",
          options: "i"
        }
      }
    }
```

```
  }
]);
```

輸出

```
{ _id: 1, name: 'María García', startsWithM: true },
{ _id: 2, name: 'Arnav Desai', startsWithM: false },
{ _id: 3, name: 'Martha Rivera', startsWithM: true },
{ _id: 4, name: 'Richard Roe', startsWithM: false }
```

程式碼範例

若要檢視使用 $regexMatch命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function checkNamePattern() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('users');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
        startsWithM: {
          $regexMatch: {
            input: "$name",
            regex: "^M",
            options: "i"
          }
        }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
```

```
}

checkNamePattern();
```

Python

```
from pymongo import MongoClient

def check_name_pattern():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.users

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'startsWithM': {
                    '$regexMatch': {
                        'input': '$name',
                        'regex': '^M',
                        'options': 'i'
                    }
                }
            }
        }
    ]))

    print(result)

    client.close()

check_name_pattern()
```

$replaceAll

在 5.0 中推出

Amazon DocumentDB 中的`$replaceAll`運算子用於以新字串取代欄位內所有出現的指定字串模式。此運算子對於資料正規化、文字清理和字串操作等任務非常有用。

參數

- `input`：包含要取代之字串的欄位或表達式。
- `find`：要搜尋和取代的字串模式。
- `replacement`：用來取代相符出現次數的字串。

範例 (MongoDB Shell)

下列範例示範如何在彙總管道中使用 `$replaceAll` 運算子，將字串 "Chocolatier" 的所有出現取代為 "Chocolate Co."。在「產品」集合的「brandName」欄位中。

建立範例文件

```
db.products.insertMany([
  {
    "_id": 1,
    "productId": "PROD-0Y9GL0",
    "brandName": "Gordon's Chocolatier",
    "category": "CPG",
    "rating": {
      "average": 4.8
    }
  },
  {
    "_id": 2,
    "productId": "PROD-1X2YZ3",
    "brandName": "Premium Chocolatier",
    "category": "CPG",
    "rating": {
      "average": 4.5
    }
  },
  {
    "_id": 3,
    "productId": "PROD-Y2E9H5",
    "name": "Nutrition Co. - Original Corn Flakes Cereal",
    "category": "Breakfast Cereals",
    "price": 8.5
  }
]);
```

查詢範例

```
db.products.aggregate([
  {
    $addFields: {
      "brandName": {
        $replaceAll: {
          input: "$brandName",
          find: "Chocolatier",
          replacement: "Chocolate Co."
        }
      }
    }
  }
])
```

輸出

```
[
  {
    _id: 1,
    productId: 'PROD-0Y9GL0',
    brandName: "Gordon's Chocolate Co.",
    category: 'CPG',
    rating: { average: 4.8 }
  },
  {
    _id: 2,
    productId: 'PROD-1X2YZ3',
    brandName: 'Premium Chocolate Co.',
    category: 'CPG',
    rating: { average: 4.5 }
  },
  {
    _id: 3,
    productId: 'PROD-Y2E9H5',
    name: 'Nutrition Co. - Original Corn Flakes Cereal',
    category: 'Breakfast Cereals',
    price: 8.5,
    brandName: null
  }
]
```

程式碼範例

若要檢視使用 $replaceAll命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function replaceAll() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  const results = await collection.aggregate([
    {
      $addFields: {
        "brandName": {
          $replaceAll: {
            input: "$brandName",
            find: "Chocolatier",
            replacement: "Chocolate Co."
          }
        }
      }
    }
  ]).toArray();

  console.log(results);

  await client.close();
}

replaceAll();
```

Python

```
from pymongo import MongoClient

def replace_all():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.products

    results = list(collection.aggregate([
        {
            "$addFields": {
                "brandName": {
                    "$replaceAll": {
                        "input": "$brandName",
                        "find": "Chocolatier",
                        "replacement": "Chocolate Co."
                    }
                }
            }
        }
    ]))

    print(results)

    client.close()

replace_all()
```

$replaceOne

在 5.0 中推出

Amazon DocumentDB 中的$replaceOne運算子是字串表達式運算子，用於彙總管道中，以替換字串取代字串中第一次出現的指定子字串。此運算子區分大小寫，只會取代找到的第一個相符項目。

參數

- input：要在其中執行尋找的字串 （欄位）。
- find：要在輸入中搜尋的字串。
- replacement：用來取代 input(field) 中第一次出現的 find 的字串。

範例 (MongoDB Shell)

下列範例示範如何在彙總管道中使用 $replaceOne運算子來取代產品名稱中的子字串。

建立範例文件

```
db.products.insertMany([
  { "_id":1, "productId": "PROD-0Y9GL0", "name": "Gordon's Extra Creamy Milk Chocolate
 - Pack of 4", "category": "Confectionery", "price": 24.99 },
  { "_id":2, "productId": "PROD-Y2E9H5", "name": "Nutrition Co. - Original Corn Flakes
 Cereal", "category": "Breakfast Cereals", "price": 8.50 },
  { "_id":3, "productId": "PROD-Z3F8K2", "name": "Gordon's Dark Chocolate (90% Cocoa)
 Pack - Pack of 4", "category": "Confectionery", "price": 28.99 }
]);
```

彙總範例

```
db.products.aggregate([
  {
    $addFields: {
      standardizedName: {
        $replaceOne: {
          input: "$name",
          find: "Pack",
          replacement: "Package"
        }
      }
    }
  }
]);
```

輸出

輸出顯示，每個產品名稱中只有第一次出現的「Pack」被取代為「Package」。

```
[
  {
    _id: 1,
    productId: 'PROD-0Y9GL0',
    name: "Gordon's Extra Creamy Milk Chocolate - Pack of 4",
    category: 'Confectionery',
    price: 24.99,
```

```
    standardizedName: "Gordon's Extra Creamy Milk Chocolate - Package of 4"
  },
  {
    _id: 2,
    productId: 'PROD-Y2E9H5',
    name: 'Nutrition Co. - Original Corn Flakes Cereal',
    category: 'Breakfast Cereals',
    price: 8.5,
    standardizedName: 'Nutrition Co. - Original Corn Flakes Cereal'
  },
  {
    _id: 3,
    productId: 'PROD-Z3F8K2',
    name: "Gordon's Dark Chocolate (90% Cocoa) Pack - Pack of 4",
    category: 'Confectionery',
    price: 28.99,
    standardizedName: "Gordon's Dark Chocolate (90% Cocoa) Package - Pack of 4"
  }
```

程式碼範例

若要檢視使用 $replaceOne命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function replaceOne() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  const pipeline = [
    {
      $addFields: {
        standardizedName: {
          $replaceOne: {
            input: '$name',
            find: 'Pack',
            replacement: 'Package'
          }
        }
```

```
      }
    }
  ];

  const result = await collection.aggregate(pipeline).toArray();
  console.log(result);

  await client.close();
}

replaceOne();
```

Python

```
from pymongo import MongoClient

def replaceOne():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['products']

    pipeline = [
        {
            '$addFields': {
                'standardizedName': {
                    '$replaceOne': {
                        'input': '$name',
                        'find': 'Pack',
                        'replacement': 'Package'
                    }
                }
            }
        }
    ]

    result = list(collection.aggregate(pipeline))
    print(result)

    client.close()

replaceOne()
```

$replaceRoot

$replaceRoot 運算子用來將根文件取代為指定的內嵌文件。當您想要將巢狀文件提升至最上層或重組資料輸出時，此功能非常有用。

參數

- newRoot：將取代現有根文件的新根文件。

範例 (MongoDB Shell)

此範例說明如何從客戶訂單擷取運送地址資訊，這對於產生運送標籤或地址清單非常有用。

建立範例文件

```
db.orders.insertMany([
  {
    "_id":1, "orderId": "ORD-2024-001", "customerId": "CUST-12345", "orderDate":
 "2024-01-15", "shippingAddress": { "name": "María García", "street": "123
 Main St", "city": "Seattle", "state": "WA", "zipCode": "98101", "country":
 "USA" },"totalAmount": 149.99 },
  { "_id":2, "orderId": "ORD-2024-002", "customerId": "CUST-67890", "orderDate":
 "2024-01-16", "shippingAddress": { "name": "Arnav Desai", "street": "456 Oak
 Ave", "city": "Portland", "state": "OR", "zipCode": "97201", "country": "USA" },
 "totalAmount": 89.50 } ])
```

查詢範例

```
db.orders.aggregate([
  {
    $replaceRoot: {
      newRoot: "$shippingAddress"
    }
  }
])
```

輸出

```
{
    name: 'María García',
    street: '123 Main St',
```

```
    city: 'Seattle',
    state: 'WA',
    zipCode: '98101',
    country: 'USA'
  },
  {
    name: 'Arnav Desai',
    street: '456 Oak Ave',
    city: 'Portland',
    state: 'OR',
    zipCode: '97201',
    country: 'USA'
  }
```

程式碼範例

若要檢視使用 $replaceRoot命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function extractShippingAddresses() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('orders');

  const result = await collection.aggregate([
    {
      $replaceRoot: {
        newRoot: "$shippingAddress"
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

extractShippingAddresses();
```

Python

```
from pymongo import MongoClient

def extract_shipping_addresses():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['orders']

    result = list(collection.aggregate([
        {
            "$replaceRoot": {
                "newRoot": "$shippingAddress"
            }
        }
    ]))

    print(result)

    client.close()

extract_shipping_addresses()
```

$replaceWith

8.0 版的新功能

Elastic 叢集不支援。

Amazon DocumentDB `$replaceWith` 中的彙總階段用於將輸入文件取代為新文件。輸入文件上的所有現有欄位，包括 _id 欄位，都會被新文件取代。 `$replaceWith` 通常用來扁平化文件，或將內嵌文件提升至頂層。

參數

- `<replacement>`（必要）：將取代現有文件的新文件。

範例 (MongoDB Shell)

下列範例示範如何使用 `$replaceWith`運算子取代 Amazon DocumentDB 集合中的現有文件。

建立範例文件

```
db.restaurants.insertMany([
  {
    "restaurantId": "REST-0Y9GL0",
    "name": "Biryani Adda",
    "cuisine": "Indian",
    "ratings": [ 3, 4, 3, 2, 2, 4, 1, 5, 5, 5 ]
  },
  {
    "restaurantId": "REST-8L2PX9",
    "name": "The Burger Spot",
    "cuisine": "American",
    "ratings": [ 2, 3, 4, 5, 3, 1, 1, 2, 4 ]
  }
]);
```

查詢範例

```
db.restaurants.aggregate([
  { $replaceWith: {
      name: "$name",
      cuisine: "$cuisine",
      rating: { $avg: "$ratings" }
    }
  }
]);
```

輸出

```
[
  {
    name: 'Biryani Adda',
    cuisine: 'Indian',
    rating: 3.4
  },
  {
    name: 'The Burger Spot',
    cuisine: 'American',
    rating: 2.7777777777777777
  }
]
```

程式碼範例

若要檢視使用 $replaceWith命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function replaceDoc() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('restaurants');

  const result = await collection.aggregate([
    { $replaceWith: {
        name: "description_index",
        cuisine: 2,
        rating: { $avg: "$ratings" }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

replaceDoc();
```

Python

```
from pymongo import MongoClient


def replace_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.restaurants

    result = list(collection.aggregate([
```

```
        {
            '$replaceWith': {
                'name': "$name",
                'cuisine': "$cuisine",
                'rating': { '$avg': "$ratings"}
            }
        }
    ]))

    print(result)
    client.close()

replace_document()
```

$reverseArray

Amazon DocumentDB 中的$reverseArray運算子用於依指定順序反轉陣列的元素。當您需要以反向重新排序陣列的元素時，此運算子非常有用。

參數

- expression：要反轉的陣列表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $reverseArray運算子來反轉陣列中的元素順序。

建立範例文件

```
db.miles.insertMany([
  { "_id" : 1, "member_since" : ISODate("1987-01-01T00:00:00Z"), "credit_card" : false,
 "flight_miles" : [ 1205, 2560, 880 ]},
  { "_id" : 2, "member_since" : ISODate("1982-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 1205, 2560, 890, 2780]},
  { "_id" : 3, "member_since" : ISODate("1999-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 1205, 880]}
]);
```

查詢範例

```
db.miles.aggregate([
```

```
  {
    $project: {
      _id: 1,
      member_since: 1,
      credit_card: 1,
      reversed_flight_miles: { $reverseArray: "$flight_miles" }
    }
  }
]);
```

輸出

```
{ "_id" : 1, "member_since" : ISODate("1987-01-01T00:00:00Z"), "credit_card" : false,
 "reversed_flight_miles" : [ 880, 2560, 1205 ] }
{ "_id" : 2, "member_since" : ISODate("1982-01-01T00:00:00Z"), "credit_card" : true,
 "reversed_flight_miles" : [ 2780, 890, 2560, 1205 ] }
{ "_id" : 3, "member_since" : ISODate("1999-01-01T00:00:00Z"), "credit_card" : true,
 "reversed_flight_miles" : [ 880, 1205 ] }
```

在此範例中，運算$reverseArray子用於反轉flight_miles陣列的順序。輸出中產生的reversed_flight_miles欄位會以反向順序顯示陣列的元素。

程式碼範例

若要檢視使用 $reverseArray命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

以下是在 Node.js 應用程式中使用 $reverseArray 運算子的範例：

```
const { MongoClient } = require('mongodb');

async function reverseArray() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('miles');

  const result = await collection.aggregate([
    {
      $project: {
```

```
        _id: 1,
        member_since: 1,
        credit_card: 1,
        reversed_flight_miles: { $reverseArray: '$flight_miles' }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

reverseArray();
```

Python

以下是在 Python 應用程式中使用 $reverseArray 運算子的範例：

```
from pymongo import MongoClient

def reverse_array():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.miles

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'member_since': 1,
                'credit_card': 1,
                'reversed_flight_miles': { '$reverseArray': '$flight_miles' }
            }
        }
    ]))

    print(result)
    client.close()

reverse_array()
```

$round

8.0.1 版的新功能。

Amazon DocumentDB 中的$round運算子會將數字四捨五入到指定的小數位數。當一個值剛好落在該位置兩個最接近的值之間時， 會$round四捨五入到最接近的偶數。例如，2.5 四捨五入為 2，3.5 四捨五入為 4。

參數

- number：解析為數字的表達式。
- place：選用。介於 -20 到 100 之間的整數表達式，指定要四捨五入的小數位數。負值會四捨五入到小數的左側。預設為 0。

範例 (MongoDB Shell)

下列範例示範如何使用 $round 運算子將數值四捨五入到小數點後一位。

使用正位值四捨五入

```
db.measurements.insertMany([
  {_id: 1, value: 3.456},
  {_id: 2, value: 7.891},
  {_id: 3, value: 12.345}
]);
```

查詢範例

```
db.measurements.aggregate([
  { $project: { rounded: { $round: ["$value", 1] } } }
]);
```

輸出

```
[
  {_id: 1, rounded: 3.5},
  {_id: 2, rounded: 7.9},
  {_id: 3, rounded: 12.3}
]
```

將中途值四捨五入到最接近的偶數

當一個值恰好落在目標位置兩個最接近的值之間時， 會$round四捨五入到最接近的偶數，而不是一律四捨五入。下列範例會將中途值四捨五入為整數。

```
db.halfway.insertMany([
  {_id: 1, value: 10.5},
  {_id: 2, value: 11.5},
  {_id: 3, value: 12.5},
  {_id: 4, value: 13.5}
]);
```

```
db.halfway.aggregate([
  { $project: { rounded: { $round: ["$value", 0] } } }
]);
```

輸出

```
[
  {_id: 1, rounded: 10},
  {_id: 2, rounded: 12},
  {_id: 3, rounded: 12},
  {_id: 4, rounded: 14}
]
```

由於 10.5 正好介於 10和 中間11，因此會四捨五入至最接近的偶數值 10。同樣地， 11.5 和 12.5都會四捨五入到最接近的偶數值 12，而 會13.5四捨五入到 14。

使用負位置值四捨五入

當 place 為負值時， 會$round四捨五入至小數點的左側。下列範例使用小數點左側的第一個數字將值四捨五入。

```
db.samples.insertMany([
  {_id: 1, value: 19.25},
  {_id: 2, value: 28.73},
  {_id: 3, value: 34.32}
]);
```

```
db.samples.aggregate([
```

```
  { $project: { rounded: { $round: ["$value", -1] } } }
]);
```

輸出

```
[
  {_id: 1, rounded: 20},
  {_id: 2, rounded: 30},
  {_id: 3, rounded: 30}
]
```

程式碼範例

若要檢視使用 $round 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('measurements');
    const result = await collection.aggregate([
      { $project: { rounded: { $round: ["$value", 1] } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['measurements']
        result = list(collection.aggregate([
            {'$project': {'rounded': {'$round': ['$value', 1]}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$rtrim

4.0 版的新功能。

Elastic 叢集不支援。

Amazon DocumentDB 中的$rtrim運算子用於從字串中移除結尾字元。根據預設，它會移除結尾空白字元，但您也可以透過傳遞字元引數來指定要移除的一組字元。

參數

- input：要從中移除結尾空格字元的輸入字串。
- chars：（選用）移除特定字元。

範例 (MongoDB Shell)

下列範例示範如何使用 $rtrim 運算子從字串中移除結尾指定的字元 (" *")。

建立範例文件

```
db.collection.insert([
  { "name": "John Doe*  ", "age": 30 },
  { "name": "Jane Smith  ", "age": 25 },
  { "name": "Bob Johnson", "age": 35 }
]);
```

查詢範例

```
db.collection.aggregate([
  {
    $project: {
      _id: 0,
      name: { $rtrim: { input: "$name", chars: " *" } },
      age: 1
    }
  }
]);
```

輸出

```
[
  { age: 30, name: 'John Doe' },
  { age: 25, name: 'Jane Smith' },
  { age: 35, name: 'Bob Johnson' }
]
```

程式碼範例

若要檢視使用 $rtrim命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient, Bson } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {

    const db = client.db('test');
    const collection = db.collection('collection');

    const pipeline = [
      {
        $project: {
          _id: 0,
```

```
          name: { $rtrim: { input: "$name", chars: " *" } },
          age: 1
        }
      }
    ];

    const results = await collection.aggregate(pipeline).toArray();
    console.dir(results, { depth: null });

  } catch (err) {
    console.error('Error occurred:', err);
  } finally {
    await client.close();
  }
}

example().catch(console.error);
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:

        db = client.test
        collection = db.collection

        pipeline = [
            {
                "$project": {
                    "_id": 0,
                    "name": { "$rtrim": { "input": "$name", "chars": " *" } },
                    "age": 1
                }
            }
        ]

        results = collection.aggregate(pipeline)
```

```
        for doc in results:
            print(doc)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        client.close()

example()
```

$sample

Amazon DocumentDB $sample 中的彙總階段用於從集合中隨機選取指定數量的文件。這適用於資料分析、測試和產生範例等任務，以供進一步處理。

參數

- size：要隨機選取的文件數量。

範例 (MongoDB Shell)

下列範例示範如何使用 $sample階段從temp集合中隨機選取兩個文件。

建立範例文件

```
db.temp.insertMany([
  { "_id": 1, "temperature": 97.1, "humidity": 0.60, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 2, "temperature": 98.2, "humidity": 0.59, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 3, "temperature": 96.8, "humidity": 0.61, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 4, "temperature": 97.9, "humidity": 0.61, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 5, "temperature": 97.5, "humidity": 0.60, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 6, "temperature": 98.0, "humidity": 0.59, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 7, "temperature": 97.2, "humidity": 0.60, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
```

```
  { "_id": 8, "temperature": 98.1, "humidity": 0.59, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 9, "temperature": 96.9, "humidity": 0.62, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 10, "temperature": 97.7, "humidity": 0.60, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") }
]);
```

查詢範例

```
db.temp.aggregate([
   { $sample: { size: 2 } }
])
```

輸出

```
{ "_id" : 4, "temperature" : 97.9, "humidity" : 0.61, "timestamp" :
 ISODate("2019-03-21T21:17:22.425Z") }
{ "_id" : 9, "temperature" : 96.9, "humidity" : 0.62, "timestamp" :
 ISODate("2019-03-21T21:17:22.425Z") }
```

結果顯示，10 份文件中有 2 份已隨機取樣。您現在可以使用這些文件來判斷平均值或執行最小/最大計算。

程式碼範例

若要檢視使用 $sample命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function sampleDocuments() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('temp');

  const result = await collection.aggregate([
    { $sample: { size: 2 } }
  ]).toArray();
```

```
  console.log(result);
  await client.close();
}

sampleDocuments();
```

Python

```
from pymongo import MongoClient

def sample_documents():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['temp']

    result = list(collection.aggregate([
        { '$sample': { 'size': 2 } }
    ]))

    print(result)
    client.close()

sample_documents()
```

$search

5.0 版的新功能。

Amazon DocumentDB 中的$search運算子用於提供文字搜尋功能。

參數

無

範例 (MongoDB Shell)

下列範例示範如何使用 $search運算子來執行文字搜尋查詢。

建立範例文件

```
db.textcollection.createIndex({"description": "text"});

db.textcollection.insertMany([
  { _id: 1, name: "John Doe", description: "This is a sample document about John
 Doe." },
  { _id: 2, name: "Jane Smith", description: "This is a sample document about Jane
 Smith." },
  { _id: 3, name: "Bob Johnson", description: "This is a sample document about Bob
 Johnson." },
  { _id: 4, name: "Jon Jeffries", description: "This is a sample document about Jon
 Jeffries." }
]);
```

查詢範例

```
db.textcollection.find(
  { $text: { $search: "John" } }
);
```

輸出

```
[
  {
    _id: 1,
    name: 'John Doe',
    description: 'This is a sample document about John Doe.'
  }
]
```

程式碼範例

若要檢視使用 $search命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findWithText() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
```

```
  const db = client.db('test');
  const collection = db.collection('textcollection');

  const result = await collection.find(
    { $text: { $search: "John" } }
  ).sort({ score: { $meta: "textScore" } }).toArray();

  console.log(result);
  client.close();
}

findWithText();
```

Python

```
from pymongo import MongoClient

def find_with_text():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['textcollection']

    result = list(collection.find(
        { '$text': { '$search': 'John' } }
    ))

    print(result)
    client.close()

find_with_text()
```

$second

Amazon DocumentDB 中的$second運算子會從日期或時間戳記擷取秒元件。它用於從日期或時間戳記欄位擷取秒值。

參數

- expression：要從中擷取秒值的日期或時間戳記欄位。此表達式可以是欄位路徑或任何解析為日期或時間戳記的有效表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $second運算子從日期欄位中擷取秒元件。

建立範例文件

```
db.users.insertMany([
  { name: "John", dob: new Date("1990-05-15T12:30:45Z") },
  { name: "Jane", dob: new Date("1985-09-20T23:59:59Z") },
  { name: "Bob", dob: new Date("2000-01-01T00:00:00Z") }
]);
```

查詢範例

```
db.users.aggregate([{ $project: { name: 1, dobSeconds: { $second: "$dob" } } }])
```

輸出

```
[
  { "_id" : ObjectId("6089a9c306a829d1f8b456a1"), "name" : "John", "dobSeconds" : 45 },
  { "_id" : ObjectId("6089a9c306a829d1f8b456a2"), "name" : "Jane", "dobSeconds" : 59 },
  { "_id" : ObjectId("6089a9c306a829d1f8b456a3"), "name" : "Bob", "dobSeconds" : 0 }
]
```

程式碼範例

若要檢視使用 $second命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const result = await users.aggregate([{ $project: { name: 1, dobSeconds:
 { $second: '$dob' } } }]).toArray();
  console.log(result);
```

```
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    users = db['users']

    result = list(users.aggregate([{'$project': {'name': 1, 'dobSeconds':
 {'$second': '$dob'}}}]))
    print(result)

    client.close()

example ()
```

$set

8.0 版的新功能

Elastic 叢集不支援。

Amazon DocumentDB $set 中的彙總階段可讓您在彙總管道期間新增欄位或更新文件中現有的欄位值。

參數

- expression：要評估的表達式。這可以是任何有效的彙總表達式，包括欄位參考和算術操作。

範例 (MongoDB Shell)

下列範例示範使用$set彙總階段，透過將 quantity 欄位乘以 price 欄位來計算總計。

建立範例文件

```
db.inventory.insertMany([
  { item: "pencil", quantity: 100, price: 0.24},
  { item: "pen", quantity: 204, price: 1.78 }
]);
```

彙總範例

```
db.inventory.aggregate([
  {
    $set: {
      total: { $multiply: ["$quantity", "$price"] }
    }
  }
])
```

輸出

```
[
  {
    _id: ObjectId('69248951d66dcae121d2950d'),
    item: 'pencil',
    quantity: 100,
    price: 0.24,
    total: 24
  },
  {
    _id: ObjectId('69248951d66dcae121d2950e'),
    item: 'pen',
    quantity: 204,
    price: 1.78,
    total: 363.12
  }
]
```

程式碼範例

若要檢視使用 $set命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const inventory = db.collection('inventory');

  const result = await inventory.aggregate([
      {
        $set: {
          total: { $multiply: ["$quantity", "$price"] }
        }
      }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    inventory = db['inventory']

    result = list(inventory.aggregate([
      {
        "$set": {
          "total": { "$multiply": ["$quantity", "$price"] }
        }
      }
    ]))

    print(result)
    client.close()
```

```
example()
```

$setDifference

4.0 版的新功能。

Amazon DocumentDB 中的$setDifference運算子用於比較兩組，並傳回第一組而非第二組中的元素。此運算子適用於尋找兩組之間的唯一元素。

參數

- firstSet ：要比較的第一個集合。
- secondSet ：要比較的第二組。

範例 (MongoDB Shell)

下列範例示範如何使用 $setDifference運算子來尋找兩組之間的唯一元素。

建立範例文件

```
db.collection.insertMany([
  { _id: 1, fruits: ["apple", "banana", "cherry", "date"] },
  { _id: 2, fruits: ["banana", "cherry", "date", "elderberry"] }
]);
```

查詢範例

```
db.collection.aggregate([
  {
    $project: {
      uniqueFruits: { $setDifference: ["$fruits", ["banana", "cherry", "date"]] }
    }
  }
]);
```

輸出

```
[
  { "_id": 1, "uniqueFruits": ["apple"] },
  { "_id": 2, "uniqueFruits": ["elderberry"] }
```

```
]
```

查詢會執行下列步驟：

1. 它使用 $project階段為每個uniqueFruits文件建立新欄位。

2. $setDifference 運算子會將fruits陣列與陣列進行比較，["banana", "cherry", "date"]並傳回fruits陣列中的唯一元素。

程式碼範例

若要檢視使用 $setDifference命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

以下是如何在 Node.js 應用程式中使用 $setDifference 運算子的範例：

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('mycollection');

  // Insert sample documents
  await collection.insertMany([
    { _id: 1, fruits: ["apple", "banana", "cherry", "date"] },
    { _id: 2, fruits: ["banana", "cherry", "date", "elderberry"] }
  ]);

 // Query using $setDifference
  const result = await collection.aggregate([
    {
      $project: {
        uniqueFruits: { $setDifference: ["$fruits", ["banana", "cherry", "date"]] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}
```

```
main();
```

Python

以下是如何在 Python 應用程式中使用 $setDifference 運算子的範例：

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['mycollection']

    # Insert sample documents
    collection.insert_many([
        {'_id': 1, 'fruits': ["apple", "banana", "cherry", "date"]},
        {'_id': 2, 'fruits': ["banana", "cherry", "date", "elderberry"]}
    ])

    # Query using $setDifference
    result = list(collection.aggregate([
        {
            '$project': {
                'uniqueFruits': {'$setDifference': ['$fruits', ["banana", "cherry",
 "date"]]}
            }
        }
    ]))

    print(result)
    client.close()

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

$setEquals

Amazon DocumentDB 中的$setEquals運算子用於判斷兩組是否相等。它會比較兩個陣列，如果true它們包含相同的不同元素，則無論它們的順序為何，都會傳回 。

參數

- expression1：要比較的第一個陣列。
- expression2：要比較的第二個陣列。

範例 (MongoDB Shell)

下列範例示範運算$setEquals子用來比較兩組值的使用情況。

建立範例文件

```
db.collection.insertMany([
  { _id: 1, fruits: ["apple", "banana", "cherry"] },
  { _id: 2, fruits: ["banana", "apple", "cherry"] },
  { _id: 3, fruits: ["apple", "banana", "orange"] }
])
```

查詢範例

```
db.collection.find({
  $expr: {
    $setEquals: ["$fruits", ["apple", "banana", "cherry"]]
  }
})
```

輸出

```
{ "_id" : 1, "fruits" : [ "apple", "banana", "cherry" ] }
{ "_id" : 2, "fruits" : [ "banana", "apple", "cherry" ] }
```

查詢使用 $setEquals運算子來比較每個文件fruits的欄位與陣列 ["apple", "banana", "cherry"]。傳回 fruits 欄位等於比較陣列的文件。

程式碼範例

若要檢視使用 $setEquals命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('mycollection');

  // Insert sample documents
  await collection.insertMany([
    { _id: 1, fruits: ["apple", "banana", "cherry"] },
    { _id: 2, fruits: ["banana", "apple", "cherry"] },
    { _id: 3, fruits: ["apple", "banana", "orange"] }
  ]);

  // Query using $setEquals
  const result = await collection.find({
    $expr: {
      $setEquals: ["$fruits", ["apple", "banana", "cherry"]]
    }
  }).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['mycollection']

    # Insert sample documents
    collection.insert_many([
        {"_id": 1, "fruits": ["apple", "banana", "cherry"]},
        {"_id": 2, "fruits": ["banana", "apple", "cherry"]},
```

```
        {"_id": 3, "fruits": ["apple", "banana", "orange"]}
    ])

    # Query using $setEquals
    result = list(collection.find({
        "$expr": {
            "$setEquals": ["$fruits", ["apple", "banana", "cherry"]]
        }
    }))

    print(result)
    client.close()

example()
```

$setIntersection

Amazon DocumentDB 中的$setIntersection運算子用於傳回兩個或多個陣列之間的常見元素。此運算子在使用資料集時特別有用，可讓您尋找多個集合的交集。

參數

- array1：要交集的第一個陣列。
- array2：要交集的第二個陣列。
- arrayN：（選用）要交集的其他陣列。

範例 (MongoDB Shell)

下列範例示範如何使用 $setIntersection運算子來尋找兩個陣列之間的常見元素。

建立範例文件

```
db.collection.insertMany([
  { _id: 1, colors: ["red", "blue", "green"] },
  { _id: 2, colors: ["blue", "yellow", "orange"] },
  { _id: 3, colors: ["red", "green", "purple"] }
])
```

查詢範例

```
db.collection.aggregate([
  { $project: {
      _id: 1,
      commonColors: { $setIntersection: ["$colors", ["red", "blue", "green"]] }
    }
  }
])
```

輸出

```
[
  { "_id": 1, "commonColors": ["red", "blue", "green"] },
  { "_id": 2, "commonColors": ["blue"] },
  { "_id": 3, "commonColors": ["red", "green"] }
]
```

程式碼範例

若要檢視使用 $setIntersection命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('mycollection');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        commonColors: { $setIntersection: ["$colors", ["red", "blue", "green"]] }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
```

```
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['mycollection']

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'commonColors': { '$setIntersection': ["$colors", ["red", "blue",
 "green"]] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$setIsSubset

Amazon DocumentDB 中的$setIsSubset運算子用於判斷一組值是否為另一個集合的子集。它適用於在陣列欄位上執行以集合為基礎的比較和操作。

參數

- field：要套用$setIsSubset運算子的欄位。
- set：要比較欄位的集合。

範例 (MongoDB Shell)

下列範例示範 $setIsSubset運算子的使用情況，以檢查 tags 欄位是否為指定集的子集。

建立範例文件

```
db.products.insertMany([
  { _id: 1, name: "Product A", tags: ["tag1", "tag2", "tag3"] },
  { _id: 2, name: "Product B", tags: ["tag1", "tag2"] },
  { _id: 3, name: "Product C", tags: ["tag2", "tag3"] }
]);
```

查詢範例

```
db.products.find({
  $expr: { $setIsSubset: [["tag1", "tag2"], "$tags"] }
})
```

注意： $setIsSubset是彙總運算子，無法直接用於 find() 查詢。在此範例中， $expr 與 搭配使用，find()以橋接查詢運算子和彙總表達式之間的間隙。

輸出

```
[
  { "_id" : 1, "name" : "Product A", "tags" : [ "tag1", "tag2", "tag3" ] },
  { "_id" : 2, "name" : "Product B", "tags" : [ "tag1", "tag2" ] }
]
```

查詢會傳回 tags 欄位是集合 子集的文件["tag1", "tag2"]。

程式碼範例

若要檢視使用 $setIsSubset命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
```

```
  const collection = db.collection('products');

  const result = await collection.find({
    $expr: { $setIsSubset: [["tag1", "tag2"], "$tags"] }
  }).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['products']

    result = list(collection.find({
        '$expr': {'$setIsSubset': [['tag1', 'tag2'], '$tags']}
    }))

    print(result)

    client.close()

example()
```

$setUnion

Amazon DocumentDB `$setUnion` 中的彙總運算子用於結合兩組或多組值，並傳回包含輸入集中所有唯一元素的集合。當您需要對文件中的陣列欄位執行設定型操作時，此運算子非常有用。

參數

- `expression1`：解析為陣列的表達式。

- expression2：解析為陣列的表達式。
- expressionN：解析為陣列的其他表達式（選用）。

範例 (MongoDB Shell)

下列範例示範如何使用 $setUnion運算子來結合集合中兩個陣列欄位的唯一元素。

建立範例文件

```
db.users.insertMany([
  { _id: 1, name: "Alice", hobbies: ["reading", "swimming"], skills: ["coding",
 "writing"] },
  { _id: 2, name: "Bob", hobbies: ["cooking", "gardening"], skills: ["coding",
 "photography"] },
  { _id: 3, name: "Charlie", hobbies: ["reading", "painting"], skills: ["gardening",
 "music"] }
]);
```

查詢範例

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      allInterests: { $setUnion: ["$hobbies", "$skills"] }
    }
  }
]);
```

輸出

```
[
  { "_id" : 1, "name" : "Alice", "allInterests" : [ "coding", "reading", "swimming",
 "writing" ] },
  { "_id" : 2, "name" : "Bob", "allInterests" : [ "coding", "cooking", "gardening",
 "photography" ] },
  { "_id" : 3, "name" : "Charlie", "allInterests" : [ "gardening", "music", "painting",
 "reading" ] }
]
```

在此範例中，運算$setUnion子用於結合每個使用者文件的 hobbies和 skills陣列欄位中的唯一元素。產生的allInterests欄位包含每個使用者所有唯一興趣和技能的聯集。

程式碼範例

若要檢視使用 $setUnion命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const result = await users.aggregate([
    {
      $project: {
        _id: 1,
        name: 1,
        allInterests: { $setUnion: ["$hobbies", "$skills"] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
```

```
    users = db['users']

    result = list(users.aggregate([
        {
            '$project': {
                '_id': 1,
                'name': 1,
                'allInterests': { '$setUnion': ["$hobbies", "$skills"] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$sigmoid

8.0.1 版的新功能。

Amazon DocumentDB 中的$sigmoid運算子會將邏輯 Sigmoid 函數 1 / (1 + e^(-x))套用至數值輸入。函數會將任何實數映射到介於 0 和 1 之間的值，產生 S 形曲線。當您需要將值標準化為 0–1 範圍時，例如將原始分數轉換為機率或相關性分數時，這會很有用。

參數

- expression：解析為數值的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $sigmoid運算子將 logit 值轉換為機率。

建立範例文件

```
db.predictions.insertMany([
  {_id: 1, logit: -2},
  {_id: 2, logit: 0},
  {_id: 3, logit: 2}
]);
```

查詢範例

```
db.predictions.aggregate([
  { $project: { probability: { $sigmoid: "$logit" } } }
]);
```

輸出

```
[
  {_id: 1, probability: 0.11920292202211755},
  {_id: 2, probability: 0.5},
  {_id: 3, probability: 0.8807970779778823}
]
```

程式碼範例

若要檢視使用 $sigmoid 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('predictions');
    const result = await collection.aggregate([
      { $project: { probability: { $sigmoid: "$logit" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['predictions']
        result = list(collection.aggregate([
            {'$project': {'probability': {'$sigmoid': '$logit'}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$sin

8.0.1 版的新功能。

Amazon DocumentDB 中的$sin運算子會傳回以弧度測量的值正弦值。在彙總管道中使用它，對數值欄位執行三角計算。

參數

- expression：解析為弧度數字的表達式。

範例 (MongoDB Shell)

下列範例顯示如何使用 $sin 運算子計算弧度角度值的正弦值。

建立範例文件

```
db.angles.insertMany([
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 0.5235987755982988 },
  { "_id": 3, "angle": 1.5707963267948966 }
```

```
]);
```

查詢範例

```
db.angles.aggregate([
  { $project: {
    "sine": { $sin: "$angle" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "sine": 0 },
  { "_id": 2, "sine": 0.49999999999999994 },
  { "_id": 3, "sine": 1 }
]
```

程式碼範例

若要檢視使用 $sin 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('angles');

  const result = await collection.aggregate([
    { $project: {
      "sine": { $sin: "$angle" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}
```

```
main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['angles']

    result = list(collection.aggregate([
        { '$project': {
            'sine': { '$sin': '$angle' }
        }}
    ]))

    print(result)
    client.close()

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

$sinh

8.0.1 版的新功能。

Amazon DocumentDB 中的$sinh運算子會傳回值的雙曲正弦值。在彙總管道中使用它，對數值欄位執行雙曲三角計算。

參數

- expression：解析為數字的表達式。

傳回類型預設為 double 。如果輸入是 128 位元的小數位數，則輸出也是 128 位元的小數位數。

範例 (MongoDB Shell)

下列範例顯示如何使用 $sinh運算子來計算數值的雙曲正弦值。

建立範例文件

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 1 },
  { "_id": 3, "value": -1 }
]);
```

查詢範例

```
db.values.aggregate([
  { $project: {
    "result": { $sinh: "$value" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "result": 0 },
  { "_id": 2, "result": 1.1752011936438014 },
  { "_id": 3, "result": -1.1752011936438014 }
]
```

程式碼範例

若要檢視使用 $sinh 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
```

```
      "result": { $sinh: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'result': { '$sinh': '$value' }
        }}
    ]))

    print(result)
    client.close()

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

$skip

在 Amazon DocumentDB 中，運算$skip子可用來偏移查詢結果的起點，讓您擷取相符文件的特定子集。這在您要擷取後續結果頁面的分頁案例中特別有用。

參數

- skip：傳回其餘文件之前要略過的文件數量。

範例 (MongoDB Shell)

下列範例示範如何使用 $skip運算子從集合擷取第二個結果頁面 （ 文件 11-20)。

建立範例文件

```
db.collection.insert([
  { "name": "Document 1" },
  { "name": "Document 2" },
  { "name": "Document 3" },
  { "name": "Document 4" },
  { "name": "Document 5" },
  { "name": "Document 6" },
  { "name": "Document 7" },
  { "name": "Document 8" },
  { "name": "Document 9" },
  { "name": "Document 10" },
  { "name": "Document 11" },
  { "name": "Document 12" },
  { "name": "Document 13" },
  { "name": "Document 14" },
  { "name": "Document 15" },
  { "name": "Document 16" },
  { "name": "Document 17" },
  { "name": "Document 18" },
  { "name": "Document 19" },
  { "name": "Document 20" }
]);
```

查詢範例

```
db.collection.find({}, { "name": 1 })
             .skip(10)
             .limit(10);
```

輸出

```
[
  { "_id" : ObjectId("..."), "name" : "Document 11" },
  { "_id" : ObjectId("..."), "name" : "Document 12" },
  { "_id" : ObjectId("..."), "name" : "Document 13" },
  { "_id" : ObjectId("..."), "name" : "Document 14" },
  { "_id" : ObjectId("..."), "name" : "Document 15" },
```

```
  { "_id" : ObjectId("..."), "name" : "Document 16" },
  { "_id" : ObjectId("..."), "name" : "Document 17" },
  { "_id" : ObjectId("..."), "name" : "Document 18" },
  { "_id" : ObjectId("..."), "name" : "Document 19" },
  { "_id" : ObjectId("..."), "name" : "Document 20" }
]
```

程式碼範例

若要檢視使用 $skip命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('collection');

  const results = await collection.find({}, { projection: { name: 1 } })
                                  .skip(10)
                                  .limit(10)
                                  .toArray();

  console.log(results);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.mydatabase
```

```
    collection = db.collection

    results = list(collection.find({}, {'name': 1})
                   .skip(10)
                   .limit(10))

    print(results)

    client.close()

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

$slice

$slice 彙總運算子可讓您從陣列的開頭或結尾周遊陣列，以傳回陣列的子集。這用於顯示陣列欄位中有限數量的項目，例如頂端或底部 N 個項目。

參數

- array：要分割的陣列欄位。
- n：整數，指定要傳回的元素數目。正值從陣列的開頭開始，負值則從陣列的結尾開始。

範例 (MongoDB Shell)

下列範例示範如何使用 $slice傳回每個 Chef 的前兩個最愛甜食。

建立範例文件

```
db.sweets.insertMany([
  { "_id" : 1, "name" : "Alvin", "favorites": [ "chocolate", "cake", "toffee",
 "beignets" ] },
  { "_id" : 2, "name" : "Tom", "favorites": [ "donuts", "pudding", "pie" ] },
  { "_id" : 3, "name" : "Jessica", "favorites": [ "fudge", "smores", "pudding",
 "cupcakes" ] },
  { "_id" : 4, "name" : "Rachel", "favorites": [ "ice cream" ] }
]);
```

查詢範例

```
db.sweets.aggregate([
```

```
  { $project: { _id: 0, name: 1, topTwoFavorites: { $slice: [ "$favorites", 2 ] } } }
]);
```

輸出

```
[
  { name: 'Alvin', topTwoFavorites: [ 'chocolate', 'cake' ] },
  { name: 'Tom', topTwoFavorites: [ 'donuts', 'pudding' ] },
  { name: 'Jessica', topTwoFavorites: [ 'fudge', 'smores' ] },
  { name: 'Rachel', topTwoFavorites: [ 'ice cream' ] }
]
```

在此範例中，運算$slice子用於從每個文件的favorites陣列中擷取前兩個元素。

程式碼範例

若要檢視使用 $slice命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('sweets');

  const result = await collection.aggregate([
    { $project: { name: 1, topTwoFavorites: { $slice: ['$favorites', 2] } } }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient
```

```
def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['sweets']

    result = list(collection.aggregate([
        { '$project': { 'name': 1, 'topTwoFavorites': { '$slice': ['$favorites',
 2] } } }
    ]))

    print(result)
    client.close()

example()
```

$size

$size 運算子用於傳回陣列欄位中的項目計數。這可用來判斷存放在文件中的陣列中的元素數目。

參數

- field：您要傳回其陣列大小的欄位路徑。

範例 (MongoDB Shell)

此範例說明如何使用 $size 運算子傳回每個使用者遵循的團隊計數。

建立範例文件

```
db.profiles.insertMany([
  { _id: 1, name: "John Doe", teams: ["Acme", "Widgets", "Gadgets"] },
  { _id: 2, name: "Jane Smith", teams: ["Acme", "Gadgets"] },
  { _id: 3, name: "Bob Johnson", teams: ["Acme", "Widgets", "Gadgets"] }
]);
```

查詢範例

```
db.profiles.aggregate([
  {
    $project: {
```

```
      _id: 0,
      name: 1,
      "numberOfTeams": { $size: "$teams" }
    }
  }
])
```

輸出

```
[
  { name: 'John Doe', numberOfTeams: 3 },
  { name: 'Jane Smith', numberOfTeams: 2 },
  { name: 'Bob Johnson', numberOfTeams: 3 }
]
```

程式碼範例

若要檢視使用 $size命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const profiles = db.collection('profiles');

  const result = await profiles.aggregate([
    {
      $project: {
        item: 1,
        "numberOfTeams": { $size: "$teams" }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}
```

```
main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    profiles = db.profiles

    result = list(profiles.aggregate([
        {
            '$project': {
                'item': 1,
                'numberOfTeams': { '$size': '$teams' }
            }
        }
    ]))

    print(result)
    client.close()

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

$sort

$sort 彙總階段會根據指定的欄位值來排序管道中的文件。文件會根據提供的排序條件，以遞增或遞減順序排列。

參數

- field：要排序的欄位名稱。
- order：1用於遞增順序或用於-1遞減順序。

範例 (MongoDB Shell)

下列範例示範如何使用 $sort階段，依價格以遞減順序訂購產品。

建立範例文件

```
db.products.insertMany([
  { _id: 1, name: "Laptop", category: "Electronics", price: 1200 },
  { _id: 2, name: "Mouse", category: "Electronics", price: 25 },
  { _id: 3, name: "Desk", category: "Furniture", price: 350 },
  { _id: 4, name: "Chair", category: "Furniture", price: 150 },
  { _id: 5, name: "Monitor", category: "Electronics", price: 400 }
]);
```

查詢範例

```
db.products.aggregate([
  { $sort: { price: -1 } }
]);
```

輸出

```
[
  { _id: 1, name: 'Laptop', category: 'Electronics', price: 1200 },
  { _id: 5, name: 'Monitor', category: 'Electronics', price: 400 },
  { _id: 3, name: 'Desk', category: 'Furniture', price: 350 },
  { _id: 4, name: 'Chair', category: 'Furniture', price: 150 },
  { _id: 2, name: 'Mouse', category: 'Electronics', price: 25 }
]
```

程式碼範例

若要檢視使用$sort彙總階段的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');
```

```
  const result = await collection.aggregate([
    { $sort: { price: -1 } }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['products']

    result = list(collection.aggregate([
        { '$sort': { 'price': -1 } }
    ]))

    print(result)
    client.close()

example()
```

$sortArray

8.0 版的新功能

Elastic 叢集不支援。

Amazon DocumentDB 中的$sortArray運算子會排序陣列的元素。對於純量值的陣列，陣列會依元素值排序。對於文件陣列，陣列會依指定的欄位排序。運算子會以遞增 (1) 或遞減 (-1) 順序傳回排序的陣列。

> **Note**
>
> $sortArray 在具有查詢規劃器第 3 版的彙總管道$project階段中支援 。

語法

```
{ $sortArray: { input: <array>, sortBy: <sort specification> } }
```

參數

- input：要排序的陣列。
- sortBy：排序順序。對於純量值陣列，1請為遞增或遞-1減指定 。對於文件陣列，{ field: -1 }請指定格式的文件{ field: 1 }或要排序的每個欄位。

範例 (MongoDB Shell)

下列範例示範如何使用 $sortArray運算子，以遞增順序排序陣列的元素。

建立範例文件

```
db.miles.insertMany([
  { "_id" : 1, "member_since" : ISODate("1987-01-01T00:00:00Z"), "credit_card" : false,
 "flight_miles" : [ 1205, 2560, 880 ]},
  { "_id" : 2, "member_since" : ISODate("1982-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 1205, 2560, 890, 2780]},
  { "_id" : 3, "member_since" : ISODate("1999-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 1205, 880]}
]);
```

查詢範例

```
db.miles.aggregate([
  {
    $project: {
      _id: 1,
      member_since: 1,
      credit_card: 1,
      sorted_flight_miles: { $sortArray: { input: "$flight_miles", sortBy: 1 } }
    }
  }
```

```
]);
```

輸出

```
{ "_id" : 1, "member_since" : ISODate("1987-01-01T00:00:00Z"), "credit_card" : false,
 "sorted_flight_miles" : [ 880, 1205, 2560 ] }
{ "_id" : 2, "member_since" : ISODate("1982-01-01T00:00:00Z"), "credit_card" : true,
 "sorted_flight_miles" : [ 890, 1205, 2560, 2780 ] }
{ "_id" : 3, "member_since" : ISODate("1999-01-01T00:00:00Z"), "credit_card" : true,
 "sorted_flight_miles" : [ 880, 1205 ] }
```

在此範例中，$sortArray運算子會以遞增順序排序flight_miles陣列。輸出中產生的sorted_flight_miles欄位會以排序順序顯示陣列的元素。

程式碼範例

若要檢視使用 $sortArray命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

以下是在 Node.js 應用程式中使用 $sortArray 運算子的範例：

```
const { MongoClient } = require('mongodb');

async function sortArray() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('miles');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        member_since: 1,
        credit_card: 1,
        sorted_flight_miles: { $sortArray: { input: '$flight_miles', sortBy: 1 } }
      }
    }
  ]).toArray();

  console.log(result);
```

```
  client.close();
}

sortArray();
```

Python

以下是在 Python 應用程式中使用 $sortArray 運算子的範例：

```
from pymongo import MongoClient

def sort_array():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.miles

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'member_since': 1,
                'credit_card': 1,
                'sorted_flight_miles': { '$sortArray': { 'input': '$flight_miles',
 'sortBy': 1 } }
            }
        }
    ]))

    print(result)
    client.close()

sort_array()
```

$sortByCount

8.0.1 版的新功能。

Amazon DocumentDB $sortByCount 中的彙總階段會根據指定表達式的值來分組傳入的文件，然後計算每個不同群組中的文件計數，並依計數以遞減順序排序結果。它相當於$group計數欄位上具有$sum累積器後接階段的$sort階段。

參數

- expression：分組依據的表達式。這可以是欄位路徑 （ 字首為 $) 或任何有效的彙總表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $sortByCount階段，依類別欄位計數和排序文件。

建立範例文件

```
db.products.insertMany([
  { name: "Widget", category: "electronics" },
  { name: "Gadget", category: "electronics" },
  { name: "Doohickey", category: "electronics" },
  { name: "Apple", category: "food" },
  { name: "Banana", category: "food" },
  { name: "Hammer", category: "tools" }
]);
```

查詢範例

```
db.products.aggregate([
  { $sortByCount: "$category" }
]);
```

輸出

```
[
  { "_id": "electronics", "count": 3 },
  { "_id": "food", "count": 2 },
  { "_id": "tools", "count": 1 }
]
```

結果會count依遞減順序自動排序。

程式碼範例

若要檢視使用$sortByCount階段的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('products');

    const result = await collection.aggregate([
      { $sortByCount: "$category" }
    ]).toArray();

    console.log(result);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']
        collection = db['products']

        result = list(collection.aggregate([
            { '$sortByCount': '$category' }
        ]))

        print(result)
    finally:
        client.close()
```

```
example()
```

$split

Amazon DocumentDB $split 中的彙總運算子用於根據指定的分隔符號，將字串分割為子字串陣列。這對於剖析複雜的字串欄位和擷取個別元件以進行進一步處理非常有用。

參數

- string：要分割的字串。
- delimiter：用來分割輸入字串的字元或字串。

範例 (MongoDB Shell)

在此範例中，我們使用 $split將 "Desk" 欄位的元件分隔為 陣列，讓您更輕鬆地處理資料。

建立範例文件

```
db.people.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-BVV-021" },
  { "_id": 2, "Desk": "Munich-HGG-32a" },
  { "_id": 3, "Desk": "Cologne-ayu-892.50" },
  { "_id": 4, "Desk": "Dortmund-Hop-78" }
]);
```

查詢範例

```
db.people.aggregate([
  { $project: { parts: { $split: ["$Desk", "-"] } } }
]);
```

輸出

```
{ "_id" : 1, "parts" : [ "Düsseldorf", "BVV", "021" ] }
{ "_id" : 2, "parts" : [ "Munich", "HGG", "32a" ] }
{ "_id" : 3, "parts" : [ "Cologne", "ayu", "892.50" ] }
{ "_id" : 4, "parts" : [ "Dortmund", "Hop", "78" ] }
```

的輸出會$split建立陣列，可用於應用程式中以顯示員工的資訊。

程式碼範例

若要檢視使用 $split命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const result = await db.collection('people').aggregate([
    { $project: { parts: { $split: ['$Desk', '-'] } } }
  ]).toArray();
  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    result = list(db.people.aggregate([
        { '$project': { 'parts': { '$split': ['$Desk', '-'] } } }
    ]))
    print(result)
    client.close()

example()
```

$sqrt

4.0 版的新功能。

Amazon DocumentDB 中的$sqrt運算子用於計算數字的平方根。

參數

- expression：引數可以是任何有效的表達式，只要解析為非負數即可。

範例 (MongoDB Shell)

下列範例示範運算$sqrt子用來計算數字平方根的用量。

建立範例文件

```
db.numbers.insertMany([
  { "_id": 1, "number": 16 },
  { "_id": 2, "number": 36 },
  { "_id": 3, "number": 64 }
]);
```

查詢範例

```
db.numbers.aggregate([
  { $project: {
    "_id": 1,
    "square_root": { $sqrt: "$number" }
  }}
]);
```

輸出

```
[
  { _id: 1, square_root: 4 },
  { _id: 2, square_root: 6 },
  { _id: 3, square_root: 8 }
]
```

程式碼範例

若要檢視使用 $sqrt命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
      await client.connect();

      const db = client.db('test');
      const collection = db.collection('numbers');

      const pipeline = [
        {
          $project: {
            _id: 1,
            square_root: { $sqrt: '$number' }
          }
        }
      ];

      const results = await collection.aggregate(pipeline).toArray();

      console.dir(results, { depth: null });

    } finally {
      await client.close();
    }
}

example().catch(console.error);
```

Python

```
from pymongo import MongoClient

def example():

  client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

  try:
```

```
        db = client.test
        collection = db.numbers

        pipeline = [
            {
                "$project": {
                    "_id": 1,
                    "square_root": {
                      "$sqrt": "$number"
                    }
                }
            }
        ]

        results = collection.aggregate(pipeline)

        for doc in results:
            print(doc)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        client.close()

example()
```

$stdDevPop

8.0.1 版的新功能。

Amazon DocumentDB 中的$stdDevPop運算子會計算數值的母群體標準差。作為累積器，它會計算彙總管道$group階段中群組內文件的母體標準差。做為表達式，它會計算數字陣列的母體標準差。母群體標準差使用 N 作為除數 （ 而非 N-1)。會忽略非數值。如果沒有數值，則會傳回 null。如果只有一個數值，則會傳回 0。

參數

- expression：解析為數值或數值陣列的表達式。

範例 (MongoDB Shell)

下列範例顯示如何使用 $stdDevPop 運算子計算每個主體的母群體分數標準差。

建立範例文件

```
db.scores.insertMany([
  { subject: "math", score: 60 },
  { subject: "math", score: 75 },
  { subject: "math", score: 85 },
  { subject: "math", score: 92 },
  { subject: "math", score: 78 },
  { subject: "science", score: 55 },
  { subject: "science", score: 70 },
  { subject: "science", score: 82 },
  { subject: "science", score: 91 },
  { subject: "science", score: 67 }
]);
```

查詢範例

```
db.scores.aggregate([
  { $group: {
      _id: "$subject",
      stdDev: { $stdDevPop: "$score" }
    }}
]);
```

輸出

```
[
  { "_id": "math", "stdDev": 10.75174404457249 },
  { "_id": "science", "stdDev": 12.441864811996632 }
]
```

表達式用量範例 (MongoDB Shell)

運算$stdDevPop子也可以用作$project階段內的表達式，以計算陣列欄位的母群體標準差。

建立範例文件

```
db.experiments.insertMany([
```

```
  { _id: 1, measurements: [10, 12, 14, 16, 18] },
  { _id: 2, measurements: [5, 5, 5, 5, 5] },
  { _id: 3, measurements: [2, 4, 6, 8, 10] }
]);
```

查詢範例

```
db.experiments.aggregate([
  { $project: {
      stdDev: { $stdDevPop: "$measurements" }
    }}
]);
```

輸出

```
[
  { "_id": 1, "stdDev": 2.8284271247461903 },
  { "_id": 2, "stdDev": 0 },
  { "_id": 3, "stdDev": 2.8284271247461903 }
]
```

程式碼範例

若要檢視使用 $stdDevPop 運算子的程式碼範例，請選擇您要使用的語言標籤。下列範例顯示累積器用量（在 中$group) 和表達式用量（在 中$project)：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();
    const db = client.db('test');

    // Accumulator usage: stdDevPop across grouped documents
    const scores = db.collection('scores');
```

```
    const accumulatorResult = await scores.aggregate([
      { $group: {
          _id: "$subject",
          stdDev: { $stdDevPop: "$score" }
        }}
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Expression usage: stdDevPop of an array field
    const experiments = db.collection('experiments');
    const expressionResult = await experiments.aggregate([
      { $project: {
          stdDev: { $stdDevPop: "$measurements" }
        }}
    ]).toArray();
    console.log('Expression result:', expressionResult);

  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']

        # Accumulator usage: stdDevPop across grouped documents
        scores = db['scores']
        accumulator_result = list(scores.aggregate([
            { '$group': {
                '_id': '$subject',
                'stdDev': { '$stdDevPop': '$score' }
            }}
```

```
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: stdDevPop of an array field
        experiments = db['experiments']
        expression_result = list(experiments.aggregate([
            { '$project': {
                'stdDev': { '$stdDevPop': '$measurements' }
            }}
        ]))
        print('Expression result:', expression_result)

    finally:
        client.close()

example()
```

$stdDevSamp

8.0.1 版的新功能。

Amazon DocumentDB 中的$stdDevSamp運算子會計算數值的範例標準差。作為累積器，它會計算彙總管道$group階段中群組內文件之間的範例標準差。做為表達式，它會計算數字陣列的樣本標準差。範例標準差使用 N-1 做為除數 (Bessel 的校正）。會忽略非數值。如果少於兩個數值，則會傳回null。

參數

- expression：解析為數值或數值陣列的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $stdDevSamp 運算子來計算每個主題的分數範例標準差。

建立範例文件

```
db.scores.insertMany([
  { subject: "math", score: 80 },
  { subject: "math", score: 90 },
  { subject: "math", score: 85 },
  { subject: "math", score: 95 },
  { subject: "science", score: 70 },
```

```
  { subject: "science", score: 75 },
  { subject: "science", score: 80 },
  { subject: "science", score: 85 }
]);
```

查詢範例

```
db.scores.aggregate([
  { $group: {
      _id: "$subject",
      stdDev: { $stdDevSamp: "$score" }
    }}
]);
```

輸出

```
[
  { "_id": "math", "stdDev": 6.454972243679028 },
  { "_id": "science", "stdDev": 6.454972243679028 }
]
```

表達式用量範例 (MongoDB Shell)

運算$stdDevSamp子也可以用作$project階段內的表達式，以計算陣列欄位的範例標準差。

建立範例文件

```
db.experiments.insertMany([
  { _id: 1, measurements: [10, 12, 14, 16, 18] },
  { _id: 2, measurements: [5, 5, 5, 5, 5] },
  { _id: 3, measurements: [2, 4, 6, 8, 10] }
]);
```

查詢範例

```
db.experiments.aggregate([
  { $project: {
      stdDev: { $stdDevSamp: "$measurements" }
    }}
]);
```

輸出

```
[
  { "_id": 1, "stdDev": 3.1622776601683795 },
  { "_id": 2, "stdDev": 0 },
  { "_id": 3, "stdDev": 3.1622776601683795 }
]
```

程式碼範例

若要檢視使用 $stdDevSamp 運算子的程式碼範例，請選擇您要使用的語言標籤。下列範例顯示累積器用量 （ 在 中$group) 和表達式用量 （ 在 中$project) ：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();
    const db = client.db('test');

    // Accumulator usage: stdDevSamp across grouped documents
    const scores = db.collection('scores');
    const accumulatorResult = await scores.aggregate([
      { $group: {
          _id: "$subject",
          stdDev: { $stdDevSamp: "$score" }
        }}
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Expression usage: stdDevSamp of an array field
    const experiments = db.collection('experiments');
    const expressionResult = await experiments.aggregate([
      { $project: {
          stdDev: { $stdDevSamp: "$measurements" }
        }}
    ]).toArray();
    console.log('Expression result:', expressionResult);
```

```
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']

        # Accumulator usage: stdDevSamp across grouped documents
        scores = db['scores']
        accumulator_result = list(scores.aggregate([
            { '$group': {
                '_id': '$subject',
                'stdDev': { '$stdDevSamp': '$score' }
            }}
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: stdDevSamp of an array field
        experiments = db['experiments']
        expression_result = list(experiments.aggregate([
            { '$project': {
                'stdDev': { '$stdDevSamp': '$measurements' }
            }}
        ]))
        print('Expression result:', expression_result)

    finally:
        client.close()

example()
```

$strLenBytes

Amazon DocumentDB 中的$strLenBytes運算子用於判斷字串的長度，以位元組為單位。當您需要了解字串欄位的儲存大小時，特別是在處理每個字元可能使用超過一個位元組的 Unicode 字元時，此功能非常有用。

參數

- `expression`：要計算 長度的字串表達式。

範例 (MongoDB Shell)

此範例示範如何使用 `$strLenBytes` 運算子來計算以位元組為單位的字串欄位長度。

建立範例文件

```
db.people.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-BVV-021" },
  { "_id": 2, "Desk": "Munich-HGG-32a" },
  { "_id": 3, "Desk": "Cologne-ayu-892.50" },
  { "_id": 4, "Desk": "Dortmund-Hop-78" }
]);
```

查詢範例

```
db.people.aggregate([
  {
    $project: {
      "Desk": 1,
      "length": { $strLenBytes: "$Desk" }
    }
  }
])
```

輸出

```
{ "_id" : 1, "Desk" : "Düsseldorf-BVV-021", "length" : 19 }
{ "_id" : 2, "Desk" : "Munich-HGG-32a", "length" : 14 }
{ "_id" : 3, "Desk" : "Cologne-ayu-892.50", "length" : 18 }
{ "_id" : 4, "Desk" : "Dortmund-Hop-78", "length" : 15 }
```

請注意，「Düsseldorf-BVV-021」字串的長度為 19 個位元組，不同於程式碼點 (18) 的數量，因為 Unicode 字元「Ü」佔用 2 個位元組。

程式碼範例

若要檢視使用 $strLenBytes命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const result = await collection.aggregate([
    {
      $project: {
        "Desk": 1,
        "length": { $strLenBytes: "$Desk" }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.people
```

```
    result = list(collection.aggregate([
        {
            '$project': {
                "Desk": 1,
                "length": { "$strLenBytes": "$Desk" }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$strLenCP

Amazon DocumentDB 中的$strLenCP運算子用於判斷程式碼點 (Unicode 字元） 中字串表達式的長度。當您需要知道字串中的字元數，而不是位元組數時，這會很有用。

參數

- expression：傳回程式碼點長度的字串表達式。

範例 (MongoDB Shell)

下列範例示範 $strLenCP運算子的使用情況，以判斷具有 Unicode 字元的字串長度。

建立範例文件

```
db.people.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-BVV-021" },
  { "_id": 2, "Desk": "Munich-HGG-32a" },
  { "_id": 3, "Desk": "Cologne-ayu-892.50" },
  { "_id": 4, "Desk": "Dortmund-Hop-78" }
])
```

查詢範例

```
db.people.aggregate([
  {
```

```
    $project: {
      "Desk": 1,
      "length": { $strLenCP: "$Desk" }
    }
  }
])
```

輸出

```
{ "_id" : 1, "Desk" : "Düsseldorf-BVV-021", "length" : 18 }
{ "_id" : 2, "Desk" : "Munich-HGG-32a", "length" : 14 }
{ "_id" : 3, "Desk" : "Cologne-ayu-892.50", "length" : 18 }
{ "_id" : 4, "Desk" : "Dortmund-Hop-78", "length" : 15 }
```

請注意 "Düsseldorf-BVV-021" 字串的長度測量差異，其中包含 Unicode 字元 (Ü)。$strLenCP 運算子會正確計算 Unicode 字元的數量，而$strLenBytes運算子則會計算位元組的數量。

程式碼範例

若要檢視使用 $strLenCP命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

以下是在 Node.js 應用程式中搭配 MongoDB 驅動程式使用 $strLenCP 運算子的範例：

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const result = await collection.aggregate([
    {
      $project: {
        "Desk": 1,
        "length": { $strLenCP: "$Desk" }
      }
    }
  ]).toArray();
```

```
  console.log(result);
  await client.close();
}

example();
```

Python

以下是在 Python 應用程式中搭配 PyMongo 驅動程式使用 $strLenCP 運算子的範例：

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.people

    result = list(collection.aggregate([
        {
            '$project': {
                "Desk": 1,
                "length": { "$strLenCP": "$Desk" }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$strcasecmp

Amazon DocumentDB 中的$strcasecmp運算子會在兩個字串之間執行不區分大小寫的比較。它會傳回整數值，指出兩個輸入字串的語彙比較，忽略大小寫差異。

參數

- string1：要比較的第一個字串。
- string2：要比較的第二個字串。

範例 (MongoDB Shell)

此範例示範如何使用 $strcasecmp 運算子來比較people集合中的桌面位置字串，忽略大小寫差異。

建立範例文件

```
db.people.insertMany([
  { "_id": 1, "Desk": "mke233-wi" },
  { "_id": 2, "Desk": "MKE233-WI" },
  { "_id": 3, "Desk": "mke233-wi" }
]);
```

查詢範例

```
db.people.aggregate([
  {
    $project: {
      item: 1,
      compare: { $strcasecmp: ["$Desk", "mke233-wi"] }
    }
  }
]);
```

輸出

```
{ "_id" : 1, "compare" : 0 }
{ "_id" : 2, "compare" : 0 }
{ "_id" : 3, "compare" : 0 }
```

輸出顯示 "Desk" 欄位與字串之間的比較"mke233-wi"會0傳回所有三個文件，表示忽略大小寫時字串相等。

程式碼範例

若要檢視使用 $strcasecmp命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const result = await collection.aggregate([
    {
      $project: {
        item: 1,
        compare: { $strcasecmp: ["$Desk", "mke233-wi"] }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.people

    result = list(collection.aggregate([
        {
            '$project': {
                'item': 1,
                'compare': { '$strcasecmp': ['$Desk', 'mke233-wi'] }
            }
        }
    ]))

    print(result)
```

```
    client.close()

example()
```

$substr

Amazon DocumentDB 中的$substr運算子用於從指定的字串擷取子字串。當您需要根據字元範圍而非位元組範圍來定義子字串時，此功能特別有用。這在處理 Unicode 字串時特別重要，其中用來代表字元的位元組數可能有所不同。

參數

- `string`：要從中擷取子字串的輸入字串。
- `start`：要擷取之子字串的起始位置（以零為基礎）。可以是非負整數表達式。
- `length`：擷取子字串中的字元數。可以是非負整數表達式。

範例 (MongoDB Shell)

在此範例中，我們將示範如何使用 從員工的桌面位置$substr擷取狀態縮寫。

建立範例文件

```
db.people.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-NRW-021" },
  { "_id": 2, "Desk": "Bremerhaven-HBB-32a" },
  { "_id": 3, "Desk": "Norderstedt-SHH-892.50" },
  { "_id": 4, "Desk": "Brandenburg-BBB-78" }
]);
```

查詢範例

```
db.people.aggregate([
  {
    $project: {
      "state": { $substr: ["$Desk", 12, 3] }
    }
  }
])
```

輸出

```
{ "_id": 1, "state": "NRW" },
{ "_id": 2, "state": "HBB" },
{ "_id": 3, "state": "SHH" },
{ "_id": 4, "state": "BBB" }
```

程式碼範例

若要檢視使用 $substr命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require("mongodb");

async function example() {
  const client = await MongoClient.connect("mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false");
  const db = client.db("test");
  const collection = db.collection("people");

  const result = await collection.aggregate([
    {
      $project: {
        "state": { $substrCP: ["$Desk", 12, 3] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
```

```
    client = MongoClient("mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false")
    db = client["test"]
    collection = db["people"]

    result = list(collection.aggregate([
        {
            "$project": {
                "state": { "$substrCP": ["$Desk", 12, 3] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$substrBytes

Amazon DocumentDB 中的$substrBytes運算子用於根據指定的位元組範圍從字串中擷取子字串。當您需要從字串中擷取子字串，且代表字串中每個字元所需的位元組數很重要時，此運算子非常有用。

與在 Unicode 程式碼點數量上運作$substrCP的 不同， 以代表字串中字元所需的位元組數量$substrBytes運作。這在使用包含非 ASCII 字元的字串時特別有用，因為這些字元可能需要多個位元組才能表示。

注意： 自 3.4 版以來$substr已棄用。 現在$substr是 的別名。 $substrBytes

參數

- string：將從中擷取子字串的輸入字串。
- startByte：要擷取之子字串的以零為基礎的起始位元組位置。負值可用來從字串結尾指定位置。
- length：子字串中要擷取的位元組數。

範例 (MongoDB Shell)

在此範例中，我們將使用 從包含非 ASCII 字元的字串中$substrBytes擷取子字串。

建立範例文件

```
db.people.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-NRW-021" },
  { "_id": 2, "Desk": "Bremerhaven-HBB-32a" },
  { "_id": 3, "Desk": "Norderstedt-SHH-892.50" },
  { "_id": 4, "Desk": "Brandenburg-BBB-78" }
]);
```

查詢範例

```
db.people.aggregate([
  {
    $project: {
      "state": { $substrBytes: [ "$Desk", 12, 3] }
    }
  }
])
```

輸出

```
{ "_id": 1, "state": "NRW" },
{ "_id": 2, "state": "HBB" },
{ "_id": 3, "state": "SHH" },
{ "_id": 4, "state": "BBB" }
```

在此範例中，我們使用 從 Desk 欄位的第 12 個位元組開始$substrBytes擷取 3 個位元組的子字串。這可讓我們擷取 2 個字元的狀態縮寫，即使字串可能包含非 ASCII 字元。

程式碼範例

若要檢視使用 $substrBytes命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const people = db.collection('people');
```

```
  const result = await people.aggregate([
    {
      $project: {
        "state": { $substrBytes: ["$Desk", 12, 3] }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    people = db.people

    result = list(people.aggregate([
        {
            '$project': {
                "state": { '$substrBytes': ["$Desk", 12, 3] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$substrCP

Amazon DocumentDB 中的$substrCP運算子用於從字串擷取子字串，其中子字串指定為 UTF-8 程式碼點 (CP) 的範圍。此運算子在使用 Unicode 字串時特別有用，因為它可讓您擷取子字串，而不必擔心字元的基礎位元組表示法。

與在位元組位置上操作的$substrBytes運算子不同，運算$substrCP子使用程式碼點位置。這可讓您更輕鬆地使用包含非 ASCII 字元的字串，因為程式碼點的數量可能不符合位元組或字元的數量。

參數

- `string`：要從中擷取子字串的輸入字串。
- `start`：要從中擷取子字串的起始碼點位置 （以零為基礎）。
- `length`：要擷取的程式碼點數目。

範例 (MongoDB Shell)

在此範例中，我們將使用 $substrCP運算子，從包含員工桌面位置的字串中擷取狀態縮寫。

建立範例文件

```
db.people.insert([
  { "_id": 1, "first_name": "Jane", "last_name": "Doe", "Desk": "12 Main St,
 Minneapolis, MN 55401" },
  { "_id": 2, "first_name": "John", "last_name": "Doe", "Desk": "456 Oak Rd, New
 Orleans, LA 70032" },
  { "_id": 3, "first_name": "Steve", "last_name": "Smith", "Desk": "789 Elm Ln,
 Bakersfield, CA 93263" }
]);
```

查詢範例

```
db.people.aggregate([
  {
    $project: {
      "state": { $substrCP: ["$Desk", 25, 2] }
    }
  }
]);
```

輸出

```
{ "_id" : 1, "state" : "MN" }
{ "_id" : 2, "state" : "LA" }
{ "_id" : 3, "state" : "CA" }
```

在此範例中，我們知道狀態縮寫從 Desk 欄位的第 25 個程式碼點開始，長度為 2 個程式碼點。透過使用 $substrCP 運算子，我們可以擷取狀態縮寫，而不必擔心字串的基礎位元組表示法。

程式碼範例

若要檢視使用 $substrCP命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findStates() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const result = await db.collection('people').aggregate([
    {
      $project: {
        "state": { $substrCP: ["$Desk", 25, 2] }
      }
    }
  ]).toArray();
  console.log(result);
  client.close();
}

findStates();
```

Python

```
from pymongo import MongoClient

def find_states():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
```

```
    result = list(db.people.aggregate([
        {
            '$project': {
                'state': { '$substrCP': ['$Desk', 25, 2] }
            }
        }
    ]))
    print(result)
    client.close()

find_states()
```

在 Node.js 和 Python 範例中，我們使用 $substrCP 運算子從 Desk 欄位擷取狀態縮寫，類似於 MongoDB Shell 範例。

$subtract

Amazon DocumentDB 中的$subtract運算子用於減去值。它可以用來減去日期、數字或兩者的組合。此運算子有助於計算兩個日期之間的差異，或從數字中減去值。

參數

- expression1：要減去的第一個值。
- expression2：要從 中減去的第二個值<expression1>。

範例 (MongoDB Shell)

下列範例示範如何使用 $subtract運算子來計算兩個日期之間的差異。

建立範例文件

```
db.dates.insert([
  {
  "_id": 1,
  "startDate": ISODate("2023-01-01T00:00:00Z"),
  "endDate": ISODate("2023-01-05T12:00:00Z")
  }
]);
```

查詢範例

```
db.dates.aggregate([
  {
    $project: {
      _id: 1,
      durationDays: {
        $divide: [
          { $subtract: ["$endDate", "$startDate"] },
          1000 * 60 * 60 * 24  // milliseconds in a day
        ]
      }
    }
  }
]);
```

輸出

```
[ { _id: 1, durationDays: 4.5 } ]
```

在此範例中，運算$subtract子用於計算 $endDate和 之間的差異，以天$startDate為單位。

程式碼範例

若要檢視使用 $subtract命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('dates');

    const pipeline = [
      {
        $project: {
          _id: 1,
```

```
          durationDays: {
            $divide: [
              { $subtract: ["$endDate", "$startDate"] },
              1000 * 60 * 60 * 24  // Convert milliseconds to days
            ]
          }
        }
      }
    ];

    const results = await collection.aggregate(pipeline).toArray();

    console.dir(results, { depth: null });

  } finally {
    await client.close();
  }
}

example().catch(console.error);
```

Python

```
from datetime import datetime, timedelta
from pymongo import MongoClient

def example():

    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client.test
        collection = db.dates

        pipeline = [
            {
                "$project": {
                    "_id": 1,
                    "durationDays": {
                        "$divide": [
                            { "$subtract": ["$endDate", "$startDate"] },
```

```
                            1000 * 60 * 60 * 24  # Convert milliseconds to days
                        ]
                    }
                }
            }
        ]

        results = collection.aggregate(pipeline)

        for doc in results:
            print(doc)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        client.close()

example()
```

$sum

Amazon DocumentDB 中的$sum運算子會傳回群組中每個文件的指定表達式總和。這是一個群組累積器運算子，通常用於彙總管道的 $group 階段，以執行總和計算。

參數

- `expression`：要加總的數值表達式。這可以是欄位路徑、表達式或常數。

範例 (MongoDB Shell)

下列範例示範如何使用 $sum運算子來計算每個產品的總銷售額。

建立範例文件

```
db.sales.insertMany([
  { product: "abc", price: 10, quantity: 2 },
  { product: "abc", price: 10, quantity: 3 },
  { product: "xyz", price: 20, quantity: 1 },
  { product: "xyz", price: 20, quantity: 5 }
]);
```

查詢範例

```
db.sales.aggregate([
  { $group: {
      _id: "$product",
      totalSales: { $sum: { $multiply: [ "$price", "$quantity" ] } }
    }}
]);
```

輸出

```
[
  { "_id": "abc", "totalSales": 50 },
  { "_id": "xyz", "totalSales": 120 }
]
```

程式碼範例

若要檢視使用 $sum命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const db = client.db('test');
    const collection = db.collection('sales');

    const result = await collection.aggregate([
      { $group: {
          _id: "$product",
          totalSales: { $sum: { $multiply: [ "$price", "$quantity" ] } }
        }}
    ]).toArray();
```

```
    console.log(result);

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client.test
        collection = db.sales

        result = list(collection.aggregate([
            { '$group': {
                '_id': '$product',
                'totalSales': { '$sum': { '$multiply': [ '$price', '$quantity' ] } }
            }}
        ]))

        pprint(result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$switch

4.0 版的新功能。

Elastic 叢集不支援。

$switch 運算子是 Amazon DocumentDB 中的條件式表達式運算子，可讓您評估案例表達式清單，並傳回評估為 true 的第一個案例的值，如果沒有案例表達式為 true，則傳回預設值。

參數

- branches：文件陣列，每個文件都有一個案例欄位，其中包含要評估的布林表達式，以及一個接著欄位，其中包含案例表達式為 true 時要傳回的值。
- default：（選用） 如果沒有任何案例表達式為 true 時要傳回的值。

範例 (MongoDB Shell)

下列範例示範如何使用 $switch運算子，根據訂單總計來判斷訂單的運送成本。

建立範例文件

```
db.orders.insertMany([
  { _id: 1, total: 50 },
  { _id: 2, total: 150 },
  { _id: 3, total: 250 }
]);
```

查詢範例

```
db.orders.aggregate([
  {
    $project: {
      _id: 1,
      total: 1,
      shippingCost: {
        $switch: {
          branches: [
            { case: { $lte: ["$total", 100] }, then: 5 },
            { case: { $lte: ["$total", 200] }, then: 10 },
            { case: { $gt: ["$total", 200] }, then: 15 }
          ],
```

```
          default: 0
        }
      }
    }
  }
])
```

輸出

```
[
  {
    "_id": 1,
    "total": 50,
    "shippingCost": 5
  },
  {
    "_id": 2,
    "total": 150,
    "shippingCost": 10
  },
  {
    "_id": 3,
    "total": 250,
    "shippingCost": 15
  }
]
```

程式碼範例

若要檢視使用 $switch命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('orders');

  const result = await collection.aggregate([
```

```
    {
      $project: {
        _id: 1,
        total: 1,
        shippingCost: {
          $switch: {
            branches: [
              { case: { $lte: ['$total', 100] }, then: 5 },
              { case: { $lte: ['$total', 200] }, then: 10 },
              { case: { $gt: ['$total', 200] }, then: 15 }
            ],
            default: 0
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.orders

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'total': 1,
                'shippingCost': {
                    '$switch': {
                        'branches': [
```

```
                            { 'case': { '$lte': ['$total', 100] }, 'then': 5 },
                            { 'case': { '$lte': ['$total', 200] }, 'then': 10 },
                            { 'case': { '$gt': ['$total', 200] }, 'then': 15 }
                        ],
                        'default': 0
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

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

$tan

8.0.1 版的新功能。

Amazon DocumentDB 中的$tan運算子會傳回以弧度測量的值切線。

輸入表達式必須解析為數值。如果您的值為度數，請在將其傳遞至 $degreesToRadians之前套用$tan。

傳回類型預設為 double 。如果輸入是 128 位元的小數位數，則輸出也是 128 位元的小數位數。

參數

- expression：解析為弧度數字的表達式。

Behavior (行為)

null、NaN 和 +/- Infinity

範例	結果
{ $tan: NaN }	NaN
{ $tan: null }	null

範例	結果
`{ $tan: Infinity }` 或 `{ $tan: -Infinity }`	擲回錯誤。

當輸入為 null或參考欄位遺失時，$tan會傳回 null。的輸入NaN會產生 NaN。正或負無限會導致錯誤，因為正切未定義在這些值。

範例 (MongoDB Shell)

下列範例示範如何使用 $tan運算子來計算弧度角的切線。

建立範例文件

```
db.angles.insertMany([
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 0.7853981633974483 },
  { "_id": 3, "angle": -0.7853981633974483 }
]);
```

查詢範例

```
db.angles.aggregate([
  { $project: {
    "tangent": { $tan: "$angle" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "tangent": 0 },
  { "_id": 2, "tangent": 0.9999999999999999 },
  { "_id": 3, "tangent": -0.9999999999999999 }
]
```

程式碼範例

若要檢視使用 $tan 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('angles');

  const result = await collection.aggregate([
    { $project: {
      "tangent": { $tan: "$angle" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['angles']

    result = list(collection.aggregate([
        { '$project': {
            'tangent': { '$tan': '$angle' }
        }}
    ]))

    print(result)
    client.close()
```

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

$tanh

8.0.1 版的新功能。

Amazon DocumentDB 中的$tanh運算子會傳回值的雙曲切線。

輸入表達式必須解析為數值。

傳回類型預設為 double 。如果輸入是 128 位元的小數位數，則輸出也是 128 位元的小數位數。

參數

- expression：解析為數字的表達式。

Behavior (行為)

null、NaN 和 +/- Infinity

範例	結果
{ $tanh: NaN }	NaN
{ $tanh: null }	null
{ $tanh: -Infinity }	-1
{ $tanh: Infinity }	1

當輸入為 null或參考欄位遺失時， $tanh會傳回 null。的輸入NaN會產生 NaN。對於正無限，結果為 1；對於負無限，結果為 -1。

範例 (MongoDB Shell)

下列範例示範如何使用 $tanh運算子來計算值的雙曲切線。

建立範例文件

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 1 },
  { "_id": 3, "value": -1 }
]);
```

查詢範例

```
db.values.aggregate([
  { $project: {
    "result": { $tanh: "$value" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "result": 0 },
  { "_id": 2, "result": 0.7615941559557649 },
  { "_id": 3, "result": -0.7615941559557649 }
]
```

程式碼範例

若要檢視使用 $tanh 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "result": { $tanh: "$value" }
    }}
```

```
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'result': { '$tanh': '$value' }
        }}
    ]))

    print(result)
    client.close()

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

$top

8.0.1 版的新功能。

使用$group階段中的$top累積器，根據指定的排序順序傳回每個群組的最高排名文件。

參數

- sortBy：指定排序順序的文件。1 使用 遞增或 -1 遞減。
- output：指定要從最上層文件傳回之欄位的表達式。

範例 (MongoDB Shell)

下列範例顯示如何使用$top累積器來尋找銷售集合中每個項目的最高銷售（最高數量）。

建立範例文件

```
db.sales.insertMany([
  { item: "abc", quantity: 10, price: 5 },
  { item: "abc", quantity: 5, price: 8 },
  { item: "xyz", quantity: 15, price: 3 },
  { item: "xyz", quantity: 7, price: 6 }
])
```

查詢範例

```
db.sales.aggregate([
  { $group: { _id: "$item", topSale: { $top: { sortBy: { quantity: -1 }, output:
 { quantity: "$quantity", price: "$price" } } } } }
])
```

輸出

```
[
  { "_id": "xyz", "topSale": { "quantity": 15, "price": 3 } },
  { "_id": "abc", "topSale": { "quantity": 10, "price": 5 } }
]
```

程式碼範例

若要檢視使用 $top 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
```

```
    await client.connect();

    const db = client.db('test');
    const collection = db.collection('sales');

    const result = await collection.aggregate([
      { $group: { _id: "$item", topSale: { $top: { sortBy: { quantity: -1 }, output:
 { quantity: "$quantity", price: "$price" } } } } }
    ]).toArray();

    console.log(result);

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client['test']
        collection = db['sales']

        result = list(collection.aggregate([
            { '$group': { '_id': '$item', 'topSale': { '$top': { 'sortBy':
 { 'quantity': -1 }, 'output': { 'quantity': '$quantity', 'price':
 '$price' } } } } }
        ]))

        pprint(result)
```

```
    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$topN

8.0.1 版的新功能。

使用$group階段中的$topN累積器，根據指定的排序順序傳回群組中的前 N 個元素。如果群組包含少於 N 個元素，則 會$topN傳回群組中的所有元素。

參數

- n：正整數或解析為一個的表達式，指定每個群組要傳回的最大結果數量。
- sortBy：指定排序順序的文件。1 使用 遞增或 -1 遞減。
- output：指定要從每個前 N 個文件傳回的欄位的表達式。

範例 (MongoDB Shell)

下列範例顯示如何使用$topN累積器來尋找銷售集合中每個項目的前 2 個銷售 （最高數量）。

建立範例文件

```
db.sales.insertMany([
  { item: "abc", quantity: 10, price: 5 },
  { item: "abc", quantity: 7, price: 8 },
  { item: "abc", quantity: 5, price: 10 },
  { item: "xyz", quantity: 15, price: 3 },
  { item: "xyz", quantity: 9, price: 6 },
  { item: "xyz", quantity: 3, price: 12 }
])
```

查詢範例

```
db.sales.aggregate([
  { $group: { _id: "$item", topTwoSales: { $topN: { n: 2, sortBy: { quantity: -1 },
 output: { quantity: "$quantity", price: "$price" } } } } }
])
```

輸出

```
[
  { "_id": "xyz", "topTwoSales": [{ "quantity": 15, "price": 3 }, { "quantity": 9,
 "price": 6 }] },
  { "_id": "abc", "topTwoSales": [{ "quantity": 10, "price": 5 }, { "quantity": 7,
 "price": 8 }] }
]
```

程式碼範例

若要檢視使用 $topN 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const db = client.db('test');
    const collection = db.collection('sales');

    const result = await collection.aggregate([
      { $group: { _id: "$item", topTwoSales: { $topN: { n: 2, sortBy: { quantity:
 -1 }, output: { quantity: "$quantity", price: "$price" } } } } }
    ]).toArray();

    console.log(result);

  } catch (error) {
    console.error('Error:', error);
```

```
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client['test']
        collection = db['sales']

        result = list(collection.aggregate([
            { '$group': { '_id': '$item', 'topTwoSales': { '$topN': { 'n': 2,
 'sortBy': { 'quantity': -1 }, 'output': { 'quantity': '$quantity', 'price':
 '$price' } } } } }
        ]))

        pprint(result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$trunc

8.0.1 版的新功能。

Amazon DocumentDB 中的$trunc運算子會將數字截斷為指定的小數位數，移除數字而不四捨五入。

參數

- number：解析為數字的表達式。
- place：選用。介於 -20 到 100 之間的整數表達式，指定要截斷的小數位數。負值會截斷到小數的左側。預設為 0。

範例 (MongoDB Shell)

下列範例顯示如何使用 $trunc運算子將數值截斷至小數點後一位。

使用正位置值截斷

```
db.measurements.insertMany([
  {_id: 1, value: 3.456},
  {_id: 2, value: 7.891},
  {_id: 3, value: 12.345}
]);
```

查詢範例

```
db.measurements.aggregate([
  { $project: { truncated: { $trunc: ["$value", 1] } } }
]);
```

輸出

```
[
  {_id: 1, truncated: 3.4},
  {_id: 2, truncated: 7.8},
  {_id: 3, truncated: 12.3}
]
```

使用負位置值截斷

當 place 為負時， 會將小數左側的數字$trunc取代為零。下列範例使用小數點左側的第一個數字截斷值。

```
db.samples.insertMany([
```

```
  {_id: 1, value: 19.25},
  {_id: 2, value: 28.73},
  {_id: 3, value: 34.32}
]);
```

```
db.samples.aggregate([
  { $project: { truncated: { $trunc: ["$value", -1] } } }
]);
```

輸出

```
[
  {_id: 1, truncated: 10},
  {_id: 2, truncated: 20},
  {_id: 3, truncated: 30}
]
```

程式碼範例

若要檢視使用 $trunc 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('measurements');
    const result = await collection.aggregate([
      { $project: { truncated: { $trunc: ["$value", 1] } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['measurements']
        result = list(collection.aggregate([
            {'$project': {'truncated': {'$trunc': ['$value', 1]}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$tsIncrement

8.0.1 版的新功能。

Amazon DocumentDB 中的$tsIncrement運算子會從時間戳記值傳回增量序數，做為長整數。

參數

- expression：解析為時間戳記的表達式。

範例 (MongoDB Shell)

下列範例顯示如何使用 $tsIncrement運算子從時間戳記值中擷取序數增量。

建立範例文件

```
db.events.insertMany([
  {_id: 1, ts: Timestamp(1678900000, 1)},
  {_id: 2, ts: Timestamp(1678900000, 2)},
  {_id: 3, ts: Timestamp(1678900001, 1)}
]);
```

查詢範例

```
db.events.aggregate([
  { $project: { increment: { $tsIncrement: "$ts" } } }
]);
```

輸出

```
[
  {_id: 1, increment: Long("1")},
  {_id: 2, increment: Long("2")},
  {_id: 3, increment: Long("1")}
]
```

程式碼範例

若要檢視使用 $tsIncrement 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient, Timestamp } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('events');
    const result = await collection.aggregate([
      { $project: { increment: { $tsIncrement: "$ts" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['events']
        result = list(collection.aggregate([
            {'$project': {'increment': {'$tsIncrement': '$ts'}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$tsSecond

8.0.1 版的新功能。

Amazon DocumentDB 中的$tsSecond運算子會以長整數 (Unix epoch 秒） 傳回時間戳記值的秒部分。

參數

- expression：解析為時間戳記的表達式。

範例 (MongoDB Shell)

下列範例顯示如何使用 $tsSecond 運算子從時間戳記值中擷取秒部分。

建立範例文件

```
db.events.insertMany([
  {_id: 1, ts: Timestamp(1678900000, 1)},
  {_id: 2, ts: Timestamp(1678900000, 2)},
  {_id: 3, ts: Timestamp(1678900001, 1)}
```

```
]);
```

查詢範例

```
db.events.aggregate([
  { $project: { seconds: { $tsSecond: "$ts" } } }
]);
```

輸出

```
[
  {_id: 1, seconds: Long("1678900000")},
  {_id: 2, seconds: Long("1678900000")},
  {_id: 3, seconds: Long("1678900001")}
]
```

程式碼範例

若要檢視使用 $tsSecond 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient, Timestamp } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('events');
    const result = await collection.aggregate([
      { $project: { seconds: { $tsSecond: "$ts" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['events']
        result = list(collection.aggregate([
            {'$project': {'seconds': {'$tsSecond': '$ts'}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$toBool

Amazon DocumentDB 中的$toBool運算子會將表達式轉換為布林值。

參數

- expression：要轉換為布林值的表達式。

注意：任何字串都會轉換為 true。

範例 (MongoDB Shell)

下列範例示範如何使用 $toBool運算子來標準化不同資料類型的裝置狀態值。

建立範例文件

```
db.deviceStates.insertMany([
  { _id: 1, deviceId: "sensor-001", status: true },
  { _id: 2, deviceId: "camera-002", status: 1 },
  { _id: 3, deviceId: "thermostat-003", status: "active" },
  { _id: 4, deviceId: "doorlock-004", status: 0 }
```

```
]);
```

查詢範例

```
db.deviceStates.aggregate([
  {
    $project: {
      _id: 1,
      deviceId: 1,
      isActive: { $toBool: "$status" }
    }
  }
]);
```

輸出

```
[
  { "_id": 1, "deviceId": "sensor-001", "isActive": true },
  { "_id": 2, "deviceId": "camera-002", "isActive": true },
  { "_id": 3, "deviceId": "thermostat-003", "isActive": true },
  { "_id": 4, "deviceId": "doorlock-004", "isActive": false }
]
```

程式碼範例

若要檢視使用 $toBool命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('deviceStates');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
```

```
        deviceId: 1,
        isActive: { $toBool: '$status' }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['deviceStates']

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'deviceId': 1,
                'isActive': { '$toBool': '$status' }
            }
        }
    ]))

    print(result)

    client.close()

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

$toDate

4.0 版的新功能

Amazon DocumentDB $toDate 中的彙總運算子用於將日期和時間字串轉換為 BSON 日期類型。這是$dateToString運算子的反向操作。

參數

- dateString：要轉換為 BSON 日期類型的日期和時間的字串表示。
- format：（選用） 指定 格式的字串dateString。如果未提供，運算子將嘗試以dateString各種標準日期和時間格式剖析 。
- timezone：（選用） 字串，代表用於轉換的時區。如果未提供，則會使用本機時區。

範例 (MongoDB Shell)

下列範例示範如何使用 $toDate運算子將日期字串轉換為 BSON 日期類型。

建立範例文件

```
db.events.insertMany([
  { _id: 1, eventName: "Mission Start", eventTime: "2023-04-15T10:30:00Z" },
  { _id: 2, eventName: "Checkpoint Reached", eventTime: "2023-04-15T11:15:00Z" },
  { _id: 3, eventName: "Mission End", eventTime: "2023-04-15T12:00:00Z" }
]);
```

查詢範例

```
db.events.aggregate([
  {
    $project: {
      eventName: 1,
      eventTimeDate: { $toDate: "$eventTime" }
    }
  }
]);
```

輸出

```
[
  {
```

```
    "_id": 1,
    "eventName": "Mission Start",
    "eventTimeDate": ISODate("2023-04-15T10:30:00Z")
  },
  {
    "_id": 2,
    "eventName": "Checkpoint Reached",
    "eventTimeDate": ISODate("2023-04-15T11:15:00Z")
  },
  {
    "_id": 3,
    "eventName": "Mission End",
    "eventTimeDate": ISODate("2023-04-15T12:00:00Z")
  }
]
```

程式碼範例

若要檢視使用 $toDate命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('events');

  const result = await collection.aggregate([
    {
      $project: {
        eventName: 1,
        eventTimeDate: { $toDate: '$eventTime' }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}
```

```
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['events']

    result = list(collection.aggregate([
        {
            '$project': {
                'eventName': 1,
                'eventTimeDate': { '$toDate': '$eventTime' }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$toDecimal

4.0 版的新功能

Amazon DocumentDB 中的$toDecimal運算子用於將值轉換為 Decimal128 資料類型。當您需要執行精確的小數算法或處理使用 Double 資料類型無法準確表示的大型小數值時，此功能非常有用。

參數

- expression：要轉換為 Decimal128 資料類型的表達式。

範例 (MongoDB Shell)

此範例示範如何使用 $toDecimal運算子將字串值轉換為 Decimal128 資料類型。

建立範例文件

```
db.numbers.insertOne({ _id: 1, value: "3.14" });
db.numbers.insertOne({ _id: 2, value: "2.71" });
```

查詢範例

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    decimalValue: { $toDecimal: "$value" }
  }}
])
```

輸出

```
[
  { "_id" : 1, "decimalValue" : Decimal128("3.14") },
  { "_id" : 2, "decimalValue" : Decimal128("2.71") }
]
```

程式碼範例

若要檢視使用 $toDecimal命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $project: {
      _id: 1,
      decimalValue: { $toDecimal: "$value" }
    }}
```

```
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.numbers

    result = list(collection.aggregate([
        {'$project': {
            '_id': 1,
            'decimalValue': {'$toDecimal': '$value'}
        }}
    ]))

    print(result)
    client.close()

example()
```

$toDouble

4.0 版的新功能

Amazon DocumentDB 中的$toDouble運算子用於將值轉換為雙精度 64 位元浮點數。當您需要對原本不是數字格式的值執行算術操作時，這很有用。

參數

<expression>：要轉換為雙值的表達式。這可以是解析為數值、字串或布林值的任何有效表達式。

範例 (MongoDB Shell)

此範例示範如何使用 $toDouble 運算子將字串值轉換為數值，以執行數學計算。

建立範例文件

```
db.numbers.insertMany([
  { _id: 1, value: "10.5" },
  { _id: 2, value: "20.25" },
  { _id: 3, value: "7" }
])
```

查詢範例

```
db.numbers.aggregate([
  {
    $project: {
      _id: 1,
      value: 1,
      double_value: { $toDouble: "$value" },
      double_plus_five: { $add: [{ $toDouble: "$value" }, 5] }
    }
  }
])
```

輸出

```
[
  { "_id" : 1, "value" : "10.5", "double_value" : 10.5, "double_plus_five" : 15.5 },
  { "_id" : 2, "value" : "20.25", "double_value" : 20.25, "double_plus_five" : 25.25 },
  { "_id" : 3, "value" : "7", "double_value" : 7.0, "double_plus_five" : 12.0 }
]
```

程式碼範例

若要檢視使用 $toDouble命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        value: 1,
        double_value: { $toDouble: "$value" },
        double_plus_five: { $add: [{ $toDouble: "$value" }, 5] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['numbers']

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'value': 1,
                'double_value': { '$toDouble': '$value' },
                'double_plus_five': { '$add': [{ '$toDouble': '$value' }, 5] }
            }
        }
```

```
    ]))

    print(result)
    client.close()

example()
```

$toInt

4.0 版的新功能

Amazon DocumentDB 中的$toInt運算子用於將輸入值轉換為整數資料類型。當您需要確保欄位或表達式以整數表示時，此運算子很有用，這對於某些操作或資料處理任務至關重要。

參數

- expression：要轉換為整數的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $toInt運算子將字串值轉換為整數。

建立範例文件

```
db.numbers.insertMany([
  { "name": "one", "value": "1" },
  { "name": "hundred", "value": "100" }
]);
```

查詢範例

```
db.numbers.aggregate([
  { $project: {
    "_id": 0,
    "name": 1,
    "intValue": { $toInt: "$value" }
  }}
]);
```

輸出

```
{ "name": "one", "intValue": 1 }
{ "name": "hundred", "intValue": 100 }
```

程式碼範例

若要檢視使用 $toInt命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $project: {
      "_id": 0,
      "name": 1,
      "intValue": { $toInt: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['numbers']
```

```
    result = list(collection.aggregate([
        { "$project": {
            "_id": 0,
            "name": 1,
            "intValue": { "$toInt": "$value" }
        }}
    ]))

    print(result)
    client.close()

example()
```

$toLong

4.0 版的新功能

Amazon DocumentDB 中的$toLong運算子用於將值轉換為 64 位元整數 （長） 資料類型。當您需要對可能儲存為字串或其他資料類型的數值執行算術操作或比較時，這會很有用。

參數

- `expression`：要轉換為 64 位元整數的表達式。

範例 (MongoDB Shell)

此範例示範如何使用 $toLong運算子將字串值轉換為 64 位元整數。

建立範例文件

```
db.numbers.insertMany([
  { _id: 1, value: "42" },
  { _id: 3, value: "9223372036854775807" }
]);
```

查詢範例

```
db.numbers.aggregate([
  {
    $project: {
```

```
      _id: 1,
      longValue: { $toLong: "$value" }
    }
  }
])
```

輸出

```
[
  { "_id" : 1, "longValue" : 42 },
  { "_id" : 3, "longValue" : 9223372036854775807 }
]
```

程式碼範例

若要檢視使用 $toLong命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const numbers = db.collection('numbers');

  const result = await numbers.aggregate([
    {
      $project: {
        _id: 1,
        longValue: { $toLong: "$value" }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    numbers = db.numbers

    result = list(numbers.aggregate([
        {
            '$project': {
                '_id': 1,
                'longValue': { '$toLong': '$value' }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$toLower

Amazon DocumentDB 中的$toLower運算子用於將字串轉換為小寫。

參數

- expression：要轉換為小寫的字串表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $toLower運算子將 Desk 欄位轉換為小寫。

建立範例文件

```
db.locations.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-BVV-021" },
  { "_id": 2, "Desk": "Munich-HGG-32a" }
```

```
]);
```

查詢範例

```
db.locations.aggregate([
  { $project: { item: { $toLower: "$Desk" } } }
]);
```

輸出

```
{ "_id" : 1, "item" : "düsseldorf-bvv-021" }
{ "_id" : 2, "item" : "munich-hgg-32a" }
```

程式碼範例

若要檢視使用 $toLower命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require("mongodb");

async function main() {
  const client = await MongoClient.connect("mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false");
  const db = client.db("test");
  const collection = db.collection("locations");

  const result = await collection.aggregate([
    { $project: { item: { $toLower: "$Desk" } } }
  ]).toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient
```

```
def main():
    client = MongoClient("mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false")
    db = client["test"]
    collection = db["locations"]

    result = list(collection.aggregate([
        { "$project": { "item": { "$toLower": "$Desk" } } }
    ]))

    print(result)

    client.close()

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

$toObjectId

4.0 版的新功能

Amazon DocumentDB 中的$toObjectId運算子用於將 ObjectId 的字串表示轉換為實際的 ObjectId 資料類型。這在處理以 ObjectIds 字串表示形式存放的資料時非常有用，因為它可讓您執行需要 ObjectId 資料類型的操作。

參數

- expression：代表有效 ObjectId 的字串表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $toObjectId運算子將 ObjectId 的字串表示轉換為 ObjectId 資料類型。

建立範例文件

```
db.employees.insertMany([
  { _id: 1, empId:"64e5f8886218c620cf0e8f8a", name: "Carol Smith", employeeId:
 "c720a" },
  { _id: 2, empId:"64e5f94e6218c620cf0e8f8c", name: "Bill Taylor", employeeId:
 "c721a" }
```

```
]);
```

查詢範例

```
db.employees.aggregate([
  { $project: {
    "empIdAsObjectId": {$toObjectId: "$empId"}}
  }
]);
```

輸出

```
[
  { _id: 1, empIdAsObjectId: ObjectId('64e5f8886218c620cf0e8f8a') },
  { _id: 2, empIdAsObjectId: ObjectId('64e5f94e6218c620cf0e8f8c') }
]
```

程式碼範例

若要檢視使用 $toObjectId命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('employees');

  const result = await collection.aggregate([
    { $project: {
      "empIdAsObjectId": {$toObjectId: "$empId"}}
    }
  ]).toArray();

  console.log(result);
  client.close();
}
```

```
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['employees']

    result = list(collection.aggregate([
      { "$project": {
        "empIdAsObjectId": {"$toObjectId": "$empId"}}
      }
    ]))

    print(result)
    client.close()

example()
```

$toString

4.0 版的新功能

Amazon DocumentDB 中的$toString運算子用於將任何類型的值（null 除外）轉換為字串表示法。當您需要對原本不是字串格式的值執行字串操作時，這會很有用。

參數

- expression：要轉換為字串的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $toString運算子將數值轉換為字串。

建立範例文件

```
db.numbers.insertMany([
```

```
  { "_id": 1, "value": 42 },
  { "_id": 2, "value": 3.14 }
]);
```

查詢範例

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    valueAsString: { $toString: "$value" }
  }}
]);
```

輸出

```
{ "_id": 1, "valueAsString": "42" }
{ "_id": 2, "valueAsString": "3.14" }
```

程式碼範例

若要檢視使用 $toString命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $project: {
      _id: 1,
      valueAsString: { $toString: '$value' }
    }}
  ]).toArray();

  console.log(result);
  client.close();
```

```
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['numbers']

    result = list(collection.aggregate([
        { '$project': {
            '_id': 1,
            'valueAsString': { '$toString': '$value' }
        }}
    ]))

    print(result)
    client.close()

example()
```

$toUpper

Amazon DocumentDB 中的$toUpper運算子用於將字串轉換為大寫。

參數

- expression：要轉換為大寫的字串表達式。

範例 (MongoDB Shell)

下列範例示範 $toUpper運算子將 Desk 欄位轉換為大寫的使用情況。

建立範例文件

```
db.locations.insertMany([
```

```
  { "_id": 1, "Desk": "düsseldorf-bvv-021" },
  { "_id": 2, "Desk": "munich-hgg-32a" }
]);
```

查詢範例

```
db.locations.aggregate([
  { $project: { item: { $toUpper: "$Desk" } } }
]);
```

輸出

```
{ "_id" : 1, "item" : "DüSSELDORF-BVV-021" }
{ "_id" : 2, "item" : "MUNICH-HGG-32A" }
```

程式碼範例

若要檢視使用 $toUpper命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('locations');

  const result = await collection.aggregate([
    { $project: { item: { $toUpper: '$Desk' } } }
  ]).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['locations']

    result = list(collection.aggregate([
        { '$project': { 'item': { '$toUpper': '$Desk' } } }
    ]))

    print(result)

    client.close()

example()
```

$toUUID

8.0.1 版的新功能。

Amazon DocumentDB 中的$toUUID運算子會將字串值轉換為 UUID （ 二進位子類型 4)。

參數

- expression：解析為 UUID 格式字串的表達式 （ 例如 "12345678-1234-1234-1234-1234-123456789abc")。

範例 (MongoDB Shell)

下列範例示範如何使用 $toUUID運算子將字串值轉換為 UUID 格式。

建立範例文件

```
db.records.insertMany([
  {_id: 1, uuidStr: "550e8400-e29b-41d4-a716-446655440000"},
  {_id: 2, uuidStr: "6ba7b810-9dad-11d1-80b4-00c04fd430c8"}
```

```
]);
```

查詢範例

```
db.records.aggregate([
  { $project: { uuid: { $toUUID: "$uuidStr" } } }
]);
```

輸出

```
[
  {_id: 1, uuid: UUID("550e8400-e29b-41d4-a716-446655440000")},
  {_id: 2, uuid: UUID("6ba7b810-9dad-11d1-80b4-00c04fd430c8")}
]
```

程式碼範例

若要檢視使用 $toUUID 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('records');
    const result = await collection.aggregate([
      { $project: { uuid: { $toUUID: "$uuidStr" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['records']
        result = list(collection.aggregate([
            {'$project': {'uuid': {'$toUUID': '$uuidStr'}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$trim

4.0 版的新功能

Amazon DocumentDB 中的$trim運算子用於從字串中移除開頭和/或結尾空格字元。

參數

- input：要修剪的字串表達式。
- chars：（選用）指定要從輸入開頭和結尾修剪的字元，預設值為空格。

範例 (MongoDB Shell)

下列範例示範如何使用 $trim運算子從字串中移除開頭和結尾空格。

建立範例文件

```
db.people.insertMany([
  { "name": "   John Doe   " },
  { "name": "   Bob Johnson   " }
])
```

查詢範例

```
db.people.aggregate([
  { $project: {
    "name": { $trim: {input: "$name"}}
  }}
])
```

輸出

```
[
  { "name": "John Doe" },
  { "name": "Bob Johnson" }
]
```

程式碼範例

若要檢視使用 $trim命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const result = await collection.aggregate([
    { $project: {
      "name": { $trim: {input: "$name" }}
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['people']

    result = list(collection.aggregate([
        {"$project": {
            "name": {"$trim": {"input": "$name"}}
        }}
    ]))

    print(result)
    client.close()

example()
```

$type

$type 彙總運算子會傳回指定欄位的 BSON 資料類型。這有助於識別彙總操作期間欄位值的資料類型。

參數

- expression：要傳回其類型的欄位或表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $type 運算子來識別每個產品價格欄位的資料類型。

建立範例文件

```
db.inventory.insertMany([
  { _id: 1, item: "Notebook", price: 15.99 },
  { _id: 2, item: "Pen", price: "2.50" },
  { _id: 3, item: "Eraser", price: 1 },
```

```
  { _id: 4, item: "Ruler", price: null }
]);
```

查詢範例

```
db.inventory.aggregate([
  {
    $project: {
      item: 1,
      price: 1,
      priceType: { $type: "$price" }
    }
  }
]);
```

輸出

```
[
  { _id: 1, item: 'Notebook', price: 15.99, priceType: 'double' },
  { _id: 2, item: 'Pen', price: '2.50', priceType: 'string' },
  { _id: 3, item: 'Eraser', price: 1, priceType: 'int' },
  { _id: 4, item: 'Ruler', price: null, priceType: 'null' }
]
```

程式碼範例

若要檢視使用$type彙總運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('inventory');

  const result = await collection.aggregate([
    {
      $project: {
```

```
        item: 1,
        price: 1,
        priceType: { $type: "$price" }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['inventory']

    result = list(collection.aggregate([
        {
            '$project': {
                'item': 1,
                'price': 1,
                'priceType': { '$type': '$price' }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$unset

8.0 版的新功能

Elastic 叢集不支援。

Amazon DocumentDB 中的$unset彙總階段可讓您從文件中移除欄位。

參數

- expression：欄位名稱或多個欄位名稱的清單。

範例 (MongoDB Shell)

下列範例示範如何使用$unset彙總階段來移除 price 欄位。

建立範例文件

```
db.inventory.insertMany([
  { item: "pencil", quantity: 100, price: 0.24},
  { item: "pen", quantity: 204, price: 1.78 }
]);
```

彙總範例

```
db.inventory.aggregate([
  {
    $unset: "price"
  }
])
```

輸出

```
[
  {
    _id: ObjectId('69248951d66dcae121d2950d'),
    item: 'pencil',
    quantity: 100
  },
  {
    _id: ObjectId('69248951d66dcae121d2950e'),
    item: 'pen',
    quantity: 204
  }
]
```

程式碼範例

若要檢視使用 $unset命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const inventory = db.collection('inventory');

  const result = await inventory.aggregate([
      {
        $unset: "price"
      }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    inventory = db['inventory']

    result = list(inventory.aggregate([
      {
        "$unset": "price"
      }
    ]))
```

```
    print(result)
    client.close()

example()
```

$unwind

$unwind 運算子用於從輸入文件解構陣列欄位，以輸出每個元素的文件。當您想要對陣列的個別元素執行操作，例如篩選、排序或轉換資料時，這會很有用。

參數

- path：要復原的陣列欄位路徑。
- includeArrayIndex：（選用）指定要保留陣列元素索引的新欄位名稱。
- preserveNullAndEmptyArrays：（選用）決定當陣列欄位為 null 或空白陣列時，操作會保留原始文件。

範例 (MongoDB Shell)

下列範例示範如何使用 $unwind運算子解構陣列欄位，並對個別元素執行進一步的操作。

建立範例文件

```
db.people.insertMany([
  { _id: 1, name: "jon", hobbies: ["painting", "dancing", "singing"] },
  { _id: 2, name: "jane", hobbies: ["reading", "swimming"] },
  { _id: 3, name: "jack", hobbies: [] }
])
```

查詢範例

```
db.people.aggregate([
  { $unwind: "$hobbies" }
])
```

輸出

```
[
  { _id: 1, name: 'jon', hobbies: 'painting' },
```

```
  { _id: 1, name: 'jon', hobbies: 'dancing' },
  { _id: 1, name: 'jon', hobbies: 'singing' },
  { _id: 2, name: 'jane', hobbies: 'reading' },
  { _id: 2, name: 'jane', hobbies: 'swimming' }
]
```

程式碼範例

若要檢視使用 $unwind命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const result = await collection.aggregate([
    { $unwind: '$hobbies' }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['people']

    result = list(collection.aggregate([
```

```
        { '$unwind': '$hobbies' }
    ]))

    print(result)
    client.close()

example()
```

$vectorSearch

8.0 版的新功能

Elastic 叢集不支援。

Amazon DocumentDB 中的$vectorSearch運算子可讓您執行向量搜尋，這是一種用於機器學習的方法，可透過使用距離或類似指標比較其向量表示法來尋找類似的資料點。此功能結合了 JSON 型文件資料庫的彈性和豐富的查詢，以及向量搜尋的強大功能，可讓您建置機器學習和生成式 AI 使用案例，例如語意搜尋、產品建議等。

參數

- <exact>（選用）：指定要執行確切最近鄰 (ENN) 還是最近鄰 (ANN) 搜尋的旗標。值可以是下列其中一項：
- false - 執行 ANN 搜尋
- true - 執行 ENN 搜尋

如果省略或設定為 false，numCandidates則為必要項目。

```
- `<index>` : Name of the Vector Search index to use.
- `<limit>` : Number of documents to return in the results.
- `<numCandidates>` (optional): This field is required if 'exact' is false or omitted.
 Number of nearest neighbors to use during the search. Value must be less than or equal
 to (<=) 1000. You can't specify a number less than the number of documents to return
 ('limit').
- `<path>` : Indexed vector type field to search.
- `<queryVector>` : Array of numbers that represent the query vector.
```

範例 (MongoDB Shell)

下列範例示範如何使用 $vectorSearch運算子，根據其向量表示法尋找類似的產品描述。

建立範例文件

```
db.products.insertMany([
  {
    _id: 1,
    name: "Product A",
    description: "A high-quality, eco-friendly product for your home.",
    description_vector: [ 0.2, 0.5, 0.8 ]
  },
  {
    _id: 2,
    name: "Product B",
    description: "An innovative and modern kitchen appliance.",
    description_vector: [0.7, 0.3, 0.9]
  },
  {
    _id: 3,
    name: "Product C",
    description: "A comfortable and stylish piece of furniture.",
    description_vector: [0.1, 0.2, 0.4]
  }
]);
```

建立向量搜尋索引

```
db.runCommand(
    {
        createIndexes: "products",
        indexes: [{
            key: {
                "description_vector": "vector"
            },
            vectorOptions: {
                type: "hnsw",
                dimensions: 3,
                similarity: "cosine",
                m: 16,
                efConstruction: 64
            },
            name: "description_index"
        }]
    }
);
```

查詢範例

```
db.products.aggregate([
  { $vectorSearch: {
      index: "description_index",
      limit: 2,
      numCandidates: 10,
      path: "description_vector",
      queryVector: [0.1, 0.2, 0.3]
    }
  }
]);
```

輸出

```
[
  {
    "_id": 1,
    "name": "Product A",
    "description": "A high-quality, eco-friendly product for your home.",
    "description_vector": [ 0.2, 0.5, 0.8 ]
  },
  {
    "_id": 3,
    "name": "Product C",
    "description": "A comfortable and stylish piece of furniture.",
    "description_vector": [ 0.1, 0.2, 0.4 ]
  }
]
```

程式碼範例

若要檢視使用 $vectorSearch命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findSimilarProducts(queryVector) {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
```

```
  const db = client.db('test');
  const collection = db.collection('products');

  const result = await collection.aggregate([
    { $vectorSearch: {
        index: "description_index",
        limit: 2,
        numCandidates: 10,
        path: "description_vector",
        queryVector: queryVector
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

findSimilarProducts([0.1, 0.2, 0.3]);
```

Python

```
from pymongo import MongoClient


def find_similar_products(query_vector):
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.products

    result = list(collection.aggregate([
        {
            '$vectorSearch': {
                'index': "description_index",
                'limit': 2,
                'numCandidates': 10,
                'path': "description_vector",
                'queryVector': query_vector
            }
        }
    ]))
```

```
    print(result)
    client.close()

find_similar_products([0.1, 0.2, 0.3])
```

$week

Amazon DocumentDB 中的$week運算子會根據 ISO 8601 標準傳回日期的週數 (0-53)。週數是根據年份和星期幾計算，而星期一是星期第一天。

參數

無

範例 (MongoDB Shell)

下列範例示範如何使用 $week運算子來擷取指定日期的週數。

建立範例文件

```
db.events.insertMany([
  { _id: 1, date: new Date("2023-01-01") },
  { _id: 2, date: new Date("2023-01-08") },
  { _id: 3, date: new Date("2023-12-31") }
]);
```

查詢範例

```
db.events.aggregate([
  { $project: {
    _id: 1,
    week: { $week: "$date" }
  }}
]);
```

輸出

```
[
  { "_id": 1, "week": 1 },
  { "_id": 2, "week": 2 },
```

```
  { "_id": 3, "week": 53 }
]
```

程式碼範例

若要檢視使用 $week命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const events = db.collection('events');

  const result = await events.aggregate([
    { $project: {
      _id: 1,
      week: { $week: "$date" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
  client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
  db = client['test']
  events = db['events']

  result = list(events.aggregate([
```

```
        {'$project': {
            '_id': 1,
            'week': {'$week': '$date'}
        }}
    ]))

    print(result)

    client.close()

example()
```

$year

Amazon DocumentDB 中的$year運算子會從日期或時間戳記擷取年份元件。

參數

- expression：要從中擷取年份元件的日期或時間戳記表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $year運算子從日期欄位中擷取年份元件。

建立範例文件

```
db.events.insertMany([
  { "_id": 1, "date": ISODate("2023-04-15T00:00:00Z") },
  { "_id": 3, "date": ISODate("2021-12-31T00:00:00Z") }
]);
```

查詢範例

```
db.events.aggregate([
  { $project: { year: { $year: "$date" } } }
]);
```

輸出

```
[
```

```
  { "_id": 1, "year": 2023 },
  { "_id": 3, "year": 2021 }
]
```

程式碼範例

若要檢視使用 $year命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('events');

  const result = await collection.aggregate([
    { $project: { year: { $year: "$date" } } }
  ]).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['events']

    result = list(collection.aggregate([
        {'$project': {'year': {'$year': '$date'}}}
```

```
    ]))

    print(result)

    client.close()

example()
```

$zip

Amazon DocumentDB 中的$zip運算子可讓您將多個陣列合併為單一元組陣列（索引鍵/值對）。當您需要透過合併文件中不同來源或陣列的資料來建立新的文件或物件時，此功能非常有用。

參數

- inputs：解析為陣列的表達式陣列。這些陣列將合併為單一元組陣列。
- useLongestLength：（選用）如果 true，輸出陣列的長度將是最長的輸入陣列，以null值填補較短的陣列。如果為 false，輸出陣列的長度將是最短的輸入陣列。
- defaults：（選用）如果對應的輸入陣列短於最長輸入陣列且 useLongestLength為 ，則用於元組的預設值陣列true。

範例 (MongoDB Shell)

下列範例示範如何使用 $zip運算子將兩個陣列合併為單一元組陣列。

建立範例文件

```
db.grades.insert([
  {
    "_id": 1,
    "name": "John",
    "scores": [90, 85, 92],
    "classes": ["Math", "English", "Science"]
  },
  {
    "_id": 2,
    "name": "Jane",
    "scores": [88, 91, 90, 85],
    "classes": ["Math", "English", "Science", "History"]
  }
```

```
])
```

查詢範例

```
db.grades.aggregate([
  {
    $project: {
      "name": 1,
      "scoredClasses": {
        $zip: {
          inputs: ["$scores", "$classes"],
          useLongestLength: true,
          defaults: [null, null]
        }
      }
    }
  }
])
```

輸出

```
[
  {
    "_id": 1,
    "name": "John",
    "scoredClasses": [
      [90, "Math"],
      [85, "English"],
      [92, "Science"],
      [null, null]
    ]
  },
  {
    "_id": 2,
    "name": "Jane",
    "scoredClasses": [
      [88, "Math"],
      [91, "English"],
      [90, "Science"],
      [85, "History"]
    ]
  }
```

```
]
```

程式碼範例

若要檢視使用 $zip命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('grades');

  const result = await collection.aggregate([
    {
      $project: {
        "name": 1,
        "scoredClasses": {
          $zip: {
            inputs: ["$scores", "$classes"],
            useLongestLength: true,
            defaults: [null, null]
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['grades']

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'scoredClasses': {
                    '$zip': {
                        'inputs': ['$scores', '$classes'],
                        'useLongestLength': True,
                        'defaults': [None, None]
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

Geospatial (地理空間)

本節提供有關 Amazon DocumentDB 支援的地理空間運算子的詳細資訊。

主題

- $geometry
- $geoIntersects
- $geoWithin
- $maxDistance
- $minDistance
- $near
- $nearSphere

$geometry

Amazon DocumentDB 中的$geometry運算子用於指定 GeoJSON 幾何物件做為地理空間查詢的一部分。此運算子會與其他地理空間查詢運算子搭配使用，例如 $geoWithin和 $geoIntersects，以對您的資料執行空間查詢。

在 Amazon DocumentDB 中，$geometry運算子支援下列 GeoJSON 幾何類型：

- 點 (Point)
- LineString
- 多邊形
- MultiPoint
- MultiLineString
- MultiPolygon
- GeometryCollection

參數

- type：GeoJSON 幾何物件的類型，例如 Point、 Polygon等。
- coordinates：代表幾何的座標陣列。座標陣列的結構取決於幾何類型。

範例 (MongoDB Shell)

下列範例示範如何使用 $geometry運算子在 Amazon DocumentDB 中執行$geoIntersects查詢。

建立範例文件

```
db.locations.insertMany([
  {
    "_id": 1,
    "name": "Location 1",
    "location": {
      "type": "Point",
      "coordinates": [-73.983253, 40.753941]
    }
  },
  {
```

```
    "_id": 2,
    "name": "Location 2",
    "location": {
      "type": "Polygon",
      "coordinates": [[
        [-73.998427, 40.730309],
        [-73.954348, 40.730309],
        [-73.954348, 40.780816],
        [-73.998427, 40.780816],
        [-73.998427, 40.730309]
      ]]
    }
  }
]);
```

查詢範例

```
db.locations.find({
  "location": {
    "$geoIntersects": {
      "$geometry": {
        "type": "Polygon",
        "coordinates": [[
          [-73.998, 40.730],
          [-73.954, 40.730],
          [-73.954, 40.781],
          [-73.998, 40.781],
          [-73.998, 40.730]
        ]]
      }
    }
  }
})
```

輸出

```
[
  {
    "_id": 2,
    "name": "Location 2",
    "location": {
      "type": "Polygon",
      "coordinates": [
```

```
        [
          [-73.998427, 40.730309],
          [-73.954348, 40.730309],
          [-73.954348, 40.780816],
          [-73.998427, 40.780816],
          [-73.998427, 40.730309]
        ]
      ]
    }
  }
]
```

程式碼範例

若要檢視使用 $geometry命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('locations');

  const query = {
    "location": {
      "$geoIntersects": {
        "$geometry": {
          "type": "Polygon",
          "coordinates": [[
            [-73.998, 40.730],
            [-73.954, 40.730],
            [-73.954, 40.781],
            [-73.998, 40.781],
            [-73.998, 40.730]
          ]]
        }
      }
    }
  };
```

```
  const result = await collection.find(query).toArray();
  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['locations']

    query = {
        "location": {
            "$geoIntersects": {
                "$geometry": {
                    "type": "Polygon",
                    "coordinates": [[
                        [-73.998, 40.730],
                        [-73.954, 40.730],
                        [-73.954, 40.781],
                        [-73.998, 40.781],
                        [-73.998, 40.730]
                    ]]
                }
            }
        }
    }

    result = list(collection.find(query))
    print(result)

    client.close()

example()
```

$geoIntersects

Amazon DocumentDB 中的$geoIntersects運算子用於尋找地理空間資料與指定 GeoJSON 物件相交的文件。此運算子適用於需要根據文件與指定地理形狀的空間關係來識別文件的應用程式，例如多邊形或多邊形。

參數

- $geometry：GeoJSON 物件，代表要檢查交集的形狀。支援的 GeoJSON 物件類型為 Point、Polygon、 LineString和 MultiPolygon。

範例 (MongoDB Shell)

下列範例示範如何使用 $geoIntersects運算子來尋找 Amazon DocumentDB 中指定座標集的狀態名稱。

建立範例文件

```
db.states.insertMany([
  {
    "name": "New York",
    "loc": {
      "type": "Polygon",
      "coordinates": [[
        [-74.25909423828125, 40.47556838210948],
        [-73.70819091796875, 40.47556838210948],
        [-73.70819091796875, 41.31342607582222],
        [-74.25909423828125, 41.31342607582222],
        [-74.25909423828125, 40.47556838210948]
      ]]
    }
  },
  {
    "name": "California",
    "loc": {
      "type": "Polygon",
      "coordinates": [[
        [-124.4091796875, 32.56456771381587],
        [-114.5458984375, 32.56456771381587],
        [-114.5458984375, 42.00964153424558],
        [-124.4091796875, 42.00964153424558],
        [-124.4091796875, 32.56456771381587]
```

```
      ]]
    }
  }
]);
```

查詢範例

```
var location = [-73.965355, 40.782865];

db.states.find({
  "loc": {
    "$geoIntersects": {
      "$geometry": {
        "type": "Point",
        "coordinates": location
      }
    }
  }
}, {
  "name": 1
});
```

輸出

```
{ "_id" : ObjectId("536b0a143004b15885c91a2c"), "name" : "New York" }
```

程式碼範例

若要檢視使用 $geoIntersects命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findStateByGeoIntersects(longitude, latitude) {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('states');

  const query = {
```

```
    loc: {
      $geoIntersects: {
        $geometry: {
          type: 'Point',
          coordinates: [longitude, latitude]
        }
      }
    }
  };

  const projection = {
    _id: 0,
    name: 1
  };

  const document = await collection.findOne(query, { projection });

  await client.close();

  if (document) {
    return document.name;
  } else {
    throw new Error('The geo location you entered was not found in the United
 States!');
  }
}
```

Python

```
from pymongo import MongoClient

def find_state_by_geointersects(longitude, latitude):
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
        db = client.test
        collection_states = db.states

        query_geointersects = {
            "loc": {
                "$geoIntersects": {
                    "$geometry": {
```

```
                        "type": "Point",
                        "coordinates": [longitude, latitude]
                    }
                }
            }
        }

        document = collection_states.find_one(query_geointersects,
                                              projection={
                                                  "_id": 0,
                                                  "name": 1
                                              })
        if document is not None:
            state_name = document['name']
            return state_name
        else:
            raise Exception("The geo location you entered was not found in the
United States!")
    except Exception as e:
        print('Exception in geoIntersects: {}'.format(e))
        raise
    finally:
        if client is not None:
            client.close()
```

$geoWithin

Amazon DocumentDB 中的$geoWithin運算子用於尋找位置資料（表示為 GeoJSON 物件）完全包含在指定形狀內的文件，例如多邊形或多邊形。這對於查詢位於特定地理區域內的物件非常有用。

參數

- $geometry：GeoJSON 物件，代表要查詢的形狀。

範例 (MongoDB Shell)

下列範例示範如何使用 $geoWithin運算子尋找位於紐約州的所有機場。

建立範例文件

```
// Insert state document
db.states.insert({
```

```
    "name": "New York",
    "loc": {
        "type": "Polygon",
        "coordinates": [[
            [-79.76278, 45.0],
            [-73.94, 45.0],
            [-73.94, 40.5],
            [-79.76278, 40.5],
            [-79.76278, 45.0]
        ]]
    }
});

// Insert airport documents
db.airports.insert([
    {
        "name": "John F. Kennedy International Airport",
        "type": "airport",
        "code": "JFK",
        "loc": {
            "type": "Point",
            "coordinates": [-73.7781, 40.6413]
        }
    },
    {
        "name": "LaGuardia Airport",
        "type": "airport",
        "code": "LGA",
        "loc": {
            "type": "Point",
            "coordinates": [-73.8772, 40.7769]
        }
    },
    {
        "name": "Buffalo Niagara International Airport",
        "type": "airport",
        "code": "BUF",
        "loc": {
            "type": "Point",
            "coordinates": [-78.7322, 42.9403]
        }
    }
]);
```

查詢範例

```
var state = db.states.findOne({"name": "New York"});

db.airports.find({
    "loc": {
        "$geoWithin": {
            "$geometry": state.loc
        }
    }
}, {
    "name": 1,
    "type": 1,
    "code": 1,
    "_id": 0
});
```

輸出

```
[
  {
    "name": "John F. Kennedy International Airport",
    "type": "airport",
    "code": "JFK"
  },
  {
    "name": "LaGuardia Airport",
    "type": "airport",
    "code": "LGA"
  }
]
```

程式碼範例

若要檢視使用 $geoWithin命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findAirportsWithinState(stateName) {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');

  const stateDoc = await db.collection('states').findOne({ name: stateName },
 { projection: { _id: 0, loc: 1 } });
  const airportDocs = await db.collection('airports').find({
    loc: {
      $geoWithin: {
        $geometry: stateDoc.loc
      }
    }
  }, { projection: { name: 1, type: 1, code: 1, _id: 0 } }).toArray();

  console.log(airportDocs);

  await client.close();
}

findAirportsWithinState('New York');
```

Python

```
from pymongo import MongoClient

def find_airports_within_state(state_name):
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
        db = client['test']
        state_doc = db.states.find_one({'name': state_name}, {'_id': 0, 'loc': 1})
        airport_docs = db.airports.find({
            'loc': {
                '$geoWithin': {
                    '$geometry': state_doc['loc']
                }
            }
        }, {'name': 1, 'type': 1, 'code': 1, '_id': 0})

        return list(airport_docs)
    except Exception as e:
```

```
        print(f'Error: {e}')
    finally:
        client.close()

airports = find_airports_within_state('New York')
print(airports)
```

$maxDistance

Amazon DocumentDB 中的$maxDistance運算子用於指定與 GeoJSON 點之間的最大距離（以公尺為單位），文件必須位於其中才能包含在查詢結果中。此運算子會與運算$nearSphere子搭配使用，以執行地理空間查詢。

參數

- $maxDistance：與文件必須位於 內的參考點之間的最大距離（公尺），以包含在查詢結果中。

範例 (MongoDB Shell)

下列範例示範如何在 Amazon DocumentDB 中使用 $maxDistance運算子來尋找距離波士頓 100 公里內的所有州首字母。

建立範例文件

```
db.capitals.insert([
  { state: "Massachusetts", city: "Boston", location: { type: "Point", coordinates:
 [-71.0589, 42.3601] } },
  { state: "Rhode Island", city: "Providence", location: { type: "Point", coordinates:
 [-71.4128, 41.8239] } },
  { state: "New Hampshire", city: "Concord", location: { type: "Point", coordinates:
 [-71.5383, 43.2067] } },
  { state: "Vermont", city: "Montpelier", location: { type: "Point", coordinates:
 [-72.5751, 44.2604] } }
]);
```

查詢範例

```
db.capitals.find(
  {
    location: {
```

```
      $nearSphere: {
        $geometry: { type: "Point", coordinates: [-71.0589, 42.3601] },
        $maxDistance: 100000
      }
    }
  },
  { state: 1, city: 1, _id: 0 }
);
```

輸出

```
[
  { "state": "Rhode Island", "city": "Providence" },
  { "state": "New Hampshire", "city": "Concord" }
]
```

程式碼範例

若要檢視使用 $maxDistance命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findCapitalsNearBoston() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const capitals = db.collection('capitals');

  const result = await capitals.find({
    location: {
      $nearSphere: {
        $geometry: { type: "Point", coordinates: [-71.0589, 42.3601] },
        $maxDistance: 100000
      }
    }
  }, {
    projection: { state: 1, city: 1, _id: 0 }
  }).toArray();

  console.log(result);
```

```
  await client.close();
}

findCapitalsNearBoston();
```

Python

```
from pymongo import MongoClient

def find_capitals_near_boston():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    capitals = db.capitals

    result = list(capitals.find(
        {
            "location": {
                "$nearSphere": {
                    "$geometry": { "type": "Point", "coordinates": [-71.0589,
 42.3601] },
                    "$maxDistance": 100000
                }
            }
        },
        {"state": 1, "city": 1, "_id": 0}
    ))

    print(result)
    client.close()

find_capitals_near_boston()
```

$minDistance

$minDistance 是與 $nearSphere或 搭配使用的尋找運算子$geoNear，用於篩選至少與中心點指定最小距離的文件。Amazon DocumentDB 支援此運算子，其函數類似於 MongoDB 中的對應函數。

參數

- $minDistance：在結果中包含文件的中心點最小距離（公尺）。

範例 (MongoDB Shell)

在此範例中，我們會在華盛頓州西雅圖特定位置的 2 公里半徑範圍內找到所有餐廳。

建立範例文件

```
db.usarestaurants.insertMany([
  {
    "state": "Washington",
    "city": "Seattle",
    "name": "Noodle House",
    "rating": 4.8,
    "location": {
      "type": "Point",
      "coordinates": [-122.3517, 47.6159]
    }
  },
  {
    "state": "Washington",
    "city": "Seattle",
    "name": "Pike Place Grill",
    "rating": 4.5,
    "location": {
      "type": "Point",
      "coordinates": [-122.3412, 47.6102]
    }
  },
  {
    "state": "Washington",
    "city": "Bellevue",
    "name": "The Burger Joint",
    "rating": 4.2,
    "location": {
      "type": "Point",
      "coordinates": [-122.2007, 47.6105]
    }
  }
]);
```

查詢範例

```
db.usarestaurants.find({
  "location": {
```

```
    "$nearSphere": {
      "$geometry": {
        "type": "Point",
        "coordinates": [-122.3516, 47.6156]
      },
      "$minDistance": 1,
      "$maxDistance": 2000
    }
  }
}, {
  "name": 1
});
```

輸出

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "name" : "Noodle House" }
{ "_id" : ObjectId("611f3da985009a81ad38e74c"), "name" : "Pike Place Grill" }
```

程式碼範例

若要檢視使用 $minDistance命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findRestaurantsNearby() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('usarestaurants');

  const result = await collection.find({
    "location": {
      "$nearSphere": {
        "$geometry": {
          "type": "Point",
          "coordinates": [-122.3516, 47.6156]
        },
        "$minDistance": 1,
        "$maxDistance": 2000
      }
```

```
    }
  }, {
    "projection": { "name": 1 }
  }).toArray();

  console.log(result);
  client.close();
}

findRestaurantsNearby();
```

Python

```
from pymongo import MongoClient

def find_restaurants_nearby():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.usarestaurants

    result = list(collection.find({
        "location": {
            "$nearSphere": {
                "$geometry": {
                    "type": "Point",
                    "coordinates": [-122.3516, 47.6156]
                },
                "$minDistance": 1,
                "$maxDistance": 2000
            }
        }
    }, {
        "projection": {"name": 1}
    }))

    print(result)
    client.close()

find_restaurants_nearby()
```

$near

Amazon DocumentDB 中的$near運算子用於尋找地理位置接近指定點的文件。它會傳回依距離排序的文件，以最接近的文件為先。此運算子需要 2dsphere 地理空間索引，對於位置資料的鄰近查詢非常有用。

參數

- $geometry：GeoJSON 點物件，定義近查詢的中心點。
- $maxDistance：（選用）從指定點開始，文件可以符合查詢的最大距離，以公尺為單位。
- $minDistance：（選用）文件可以符合查詢之指定點的最小距離，以公尺為單位。

索引需求

- 2dsphere index：GeoJSON Point 資料上的地理空間查詢需要。

範例 (MongoDB Shell)

下列範例示範如何使用 $near運算子尋找距離華盛頓州西雅圖特定位置最近的餐廳。

建立範例文件

```
db.usarestaurants.insert([
  {
    "name": "Noodle House",
    "city": "Seattle",
    "state": "Washington",
    "rating": 4.8,
    "location": { "type": "Point", "coordinates": [-122.3517, 47.6159] }
  },
  {
    "name": "Pike Place Grill",
    "city": "Seattle",
    "state": "Washington",
    "rating": 4.2,
    "location": { "type": "Point", "coordinates": [-122.3403, 47.6062] }
  },
  {
    "name": "Lola",
    "city": "Seattle",
    "state": "Washington",
```

```
    "rating": 4.5,
    "location": { "type": "Point", "coordinates": [-122.3407, 47.6107] }
  }
]);
```

建立 2dsphere 索引

```
db.usarestaurants.createIndex({ "location": "2dsphere" });
```

使用 GeoJSON Point 查詢範例

```
db.usarestaurants.find({
  location: {
    $near: {
      $geometry: {
        type: "Point",
        coordinates: [-122.3516, 47.6156]
      },
      $maxDistance: 100,
      $minDistance: 10
    }
  }
});
```

輸出

```
{
  "_id" : ObjectId("69031ec9ea1c2922a1ce5f4a"),
  "name" : "Noodle House",
  "city" : "Seattle",
  "state" : "Washington",
  "rating" : 4.8,
  "location" : {
    "type" : "Point",
    "coordinates" : [ -122.3517, 47.6159 ]
  }
}
```

程式碼範例

若要檢視使用 $near命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findNearbyRestaurants() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const restaurants = db.collection('usarestaurants');

  // Create 2dsphere index
  await restaurants.createIndex({ "location": "2dsphere" });

  const result = await restaurants.find({
    location: {
      $near: {
        $geometry: {
          type: "Point",
          coordinates: [-122.3516, 47.6156]
        },
        $maxDistance: 100,
        $minDistance: 10
      }
    }
  }).toArray();

  console.log(result);

  client.close();
}

findNearbyRestaurants();
```

Python

```
from pymongo import MongoClient

def find_nearby_restaurants():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
```

```
    restaurants = db['usarestaurants']

    # Create 2dsphere index
    restaurants.create_index([("location", "2dsphere")])

    result = list(restaurants.find({
        'location': {
            '$near': {
                '$geometry': {
                    'type': 'Point',
                    'coordinates': [-122.3516, 47.6156]
                },
                '$maxDistance': 100,
                '$minDistance': 10
            }
        }
    }))

    print(result)

    client.close()

find_nearby_restaurants()
```

$nearSphere

Amazon DocumentDB 中的$nearSphere運算子用於尋找地理空間點指定距離內的文件。此運算子對於地理空間查詢特別有用，例如尋找指定位置特定半徑內的所有餐廳。

參數

- $geometry：代表參考點的 GeoJSON 物件。必須是具有 type和 coordinates 欄位的Point物件。
- $minDistance：（選用） 文件必須距參考點的最小距離 （公尺）。
- $maxDistance：（選用） 文件必須距參考點的最大距離 （以公尺為單位）。

範例 (MongoDB Shell)

在此範例中，我們會在華盛頓州西雅圖特定位置 2 公里 (2000 公尺） 內找到所有餐廳。

建立範例文件

```
db.usarestaurants.insert([
  {
    name: "Noodle House",
    location: { type: "Point", coordinates: [-122.3516, 47.6156] }
  },
  {
    name: "Pike Place Grill",
    location: { type: "Point", coordinates: [-122.3403, 47.6101] }
  },
  {
    name: "Seattle Coffee Co.",
    location: { type: "Point", coordinates: [-122.3339, 47.6062] }
  }
]);
```

查詢範例

```
db.usarestaurants.find({
  location: {
    $nearSphere: {
      $geometry: {
        type: "Point",
        coordinates: [-122.3516, 47.6156]
      },
      $minDistance: 1,
      $maxDistance: 2000
    }
  }
}, {
  name: 1
});
```

輸出

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "name" : "Noodle House" }
{ "_id" : ObjectId("611f3da985009a81ad38e74c"), "name" : "Pike Place Grill" }
```

程式碼範例

若要檢視使用 $nearSphere命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findNearbyRestaurants() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const restaurants = db.collection('usarestaurants');

  const result = await restaurants.find({
    location: {
      $nearSphere: {
        $geometry: {
          type: "Point",
          coordinates: [-122.3516, 47.6156]
        },
        $minDistance: 1,
        $maxDistance: 2000
      }
    }
  }, {
    projection: { name: 1 }
  }).toArray();

  console.log(result);
  client.close();
}

findNearbyRestaurants();
```

Python

```
from pymongo import MongoClient

def find_nearby_restaurants():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    restaurants = db.usarestaurants
```

```
    result = list(restaurants.find({
        'location': {
            '$nearSphere': {
                '$geometry': {
                    'type': 'Point',
                    'coordinates': [-122.3516, 47.6156]
                },
                '$minDistance': 1,
                '$maxDistance': 2000
            }
        }
    }, {
        'name': 1
    }))

    print(result)
    client.close()

find_nearby_restaurants()
```

查詢和投影運算子

本節提供有關 Amazon DocumentDB 支援的查詢和投影運算子的詳細資訊。

主題

- $
- $all
- $and
- $bitsAllClear
- $bitsAllSet
- $bitsAnyClear
- $bitsAnySet
- $comment
- $elemMatch
- $eq
- $exists
- $expr

- $gt
- $gte
- $in
- $jsonSchema
- $lt
- $mod
- $lte
- $meta
- $ne
- $nin
- $nor
- $not
- $or
- $regex
- $sampleRate
- $slice
- $size
- $text
- $type

$

$ 投影運算子會限制陣列欄位的內容，只傳回符合查詢條件的第一個元素。它用於投影單一相符的陣列元素。

參數

- `field.$`：具有位置運算子的陣列欄位，用於投影第一個相符的元素。

範例 (MongoDB Shell)

下列範例示範使用$投影運算子僅傳回相符的陣列元素。

建立範例文件

```
db.students.insertMany([
  { _id: 1, name: "Alice", grades: [85, 92, 78, 95] },
  { _id: 2, name: "Bob", grades: [70, 88, 92, 65] },
  { _id: 3, name: "Charlie", grades: [95, 89, 91, 88] }
]);
```

查詢範例

```
db.students.find(
  { grades: { $gte: 90 } },
  { name: 1, "grades.$": 1 }
);
```

輸出

```
{ "_id" : 1, "name" : "Alice", "grades" : [ 92 ] }
{ "_id" : 2, "name" : "Bob", "grades" : [ 92 ] }
{ "_id" : 3, "name" : "Charlie", "grades" : [ 95 ] }
```

在此範例中，每個學生只會傳回大於或等於 90 的第一年級。

程式碼範例

若要檢視使用$投影運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('students');

  const result = await collection.find(
    { grades: { $gte: 90 } },
    { projection: { name: 1, "grades.$": 1 } }
  ).toArray();
```

```
  console.log(JSON.stringify(result, null, 2));
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['students']

    result = list(collection.find(
        {'grades': {'$gte': 90}},
        {'name': 1, 'grades.$': 1}
    ))

    print(result)
    client.close()

example()
```

$all

Amazon DocumentDB 中的$all運算子用於比對欄位值為陣列的文件，並包含所有指定的元素，無論陣列中的元素順序為何。

參數

- field：要檢查的欄位名稱。
- [value1, value2, ...]：陣列中要比對的值清單。

在**$all**表達式**$elemMatch**內使用

在$all表達式$elemMatch內使用 如需在$all表達式中使用$elemMatch運算子的限制，請參閱 。

欄位名稱中的美元 ($)

欄位名稱中的 Dollar($) 和 dot(.) 如需在巢狀物件$all中查詢字$首欄位的限制，請參閱 。

範例 (MongoDB Shell)

下列範例示範 $all運算子的使用情況，以擷取「顏色」欄位為包含「紅色」和「藍色」的陣列的文件。

建立範例文件

```
db.example.insertMany([
  { "Item": "Pen", "Colors": ["Red", "Blue", "Green"] },
  { "Item": "Notebook", "Colors": ["Blue", "White"] },
  { "Item": "Poster Paint", "Colors": ["Red", "Yellow", "White"] }
])
```

查詢範例

```
db.example.find({ "Colors": { $all: ["Red", "Blue"] } }).pretty()
```

輸出

```
{
  "_id" : ObjectId("6137d6c5b3a1d35e0b6ee6ad"),
  "Item" : "Pen",
  "Colors" : [
          "Red",
          "Blue",
          "Green"
  ]
}
```

程式碼範例

若要檢視使用 $all命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('example');

  const result = await collection.find({ "Colors": { $all: ["Red",
 "Blue"] } }).toArray();
  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['example']

    result = list(collection.find({ "Colors": { "$all": ["Red", "Blue"] } }))
    print(result)

    client.close()

example()
```

$and

Amazon DocumentDB 中的$and運算子用於合併多個表達式，並將其評估為單一條件。true 如果所有提供的表達式評估為 true，則傳回 ，false否則傳回 。此運算子適用於將多個條件套用至查詢。

參數

- expression1：評估為布林值的必要表達式。
- expression2：評估為布林值的必要表達式。
- ...：評估為布林值的其他必要表達式。

範例 (MongoDB Shell)

以下範例示範使用 $and 運算子來尋找「使用者」集合中「年齡」欄位大於 18 且「狀態」欄位為「作用中」的所有文件。

建立範例文件

```
db.users.insertMany([
  { name: "John", age: 25, status: "active" },
  { name: "Jane", age: 17, status: "active" },
  { name: "Bob", age: 30, status: "inactive" },
  { name: "Alice", age: 22, status: "active" }
]);
```

查詢範例

```
db.users.find({
  $and: [
    { age: { $gt: 18 } },
    { status: "active" }
  ]
});
```

輸出

```
[
  { "_id" : ObjectId("614e3c4b63f5892e7c4e2345"), "name" : "John", "age" : 25,
 "status" : "active" },
  { "_id" : ObjectId("614e3c4b63f5892e7c4e2347"), "name" : "Alice", "age" : 22,
 "status" : "active" }
]
```

程式碼範例

若要檢視使用 $and命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findActiveUsersOlderThan18() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const activeUsersOlderThan18 = await users.find({
    $and: [
      { age: { $gt: 18 } },
      { status: 'active' }
    ]
  }).toArray();

  console.log(activeUsersOlderThan18);
  await client.close();
}

findActiveUsersOlderThan18();
```

Python

```
from pymongo import MongoClient

def find_active_users_older_than_18():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    users = db['users']

    active_users_older_than_18 = list(users.find({
        '$and': [
            {'age': {'$gt': 18}},
            {'status': 'active'}
        ]
    }))

    print(active_users_older_than_18)
```

```
    client.close()

find_active_users_older_than_18()
```

$bitsAllClear

Amazon DocumentDB 中的$bitsAllClear運算子用於比對清除欄位中所有指定位元的文件 （設定為 0)。此運算子有助於對儲存的資料執行位元操作。

參數

- field：要檢查是否清除指定位元的欄位。
- value：指定應檢查哪些位元的數值位元遮罩，或要檢查的位元位置清單。數字位元遮罩可以是二進位 (0b...)、十進位、十六進位 (0x...)、八進位 (0o...) 或二進位 (BinData) 格式。在位元位置清單中，最低有效位元的位置為 0。

範例 (MongoDB Shell)

下列範例示範 Amazon DocumentDB 中$bitsAllClear運算子的使用情況。

建立範例文件

```
db.collection.insertMany([
  { _id: 1, bits: 0b1010 },
  { _id: 2, bits: 0b1100 },
  { _id: 3, bits: 0b0101 }
]);
```

查詢範例

```
db.collection.find({
  bits: { $bitsAllClear: 0b0011 }
})
```

輸出

```
{ "_id" : 2, "bits" : 12 }
```

查詢會檢查位元遮罩指定的所有位元 0b0011（兩個最不重要的位元）是否在 bits 欄位中清除。具有 _id 2 的文件符合此條件，因為其bits欄位已清除這些位元。

程式碼範例

若要檢視使用 $bitsAllClear命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.find({
    bits: { $bitsAllClear: 0b0011 }
  }).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['collection']

    result = list(collection.find({
        'bits': { '$bitsAllClear': 0b0011 }
    }))
```

```
    print(result)

    client.close()

example()
```

$bitsAllSet

Amazon DocumentDB 中的$bitsAllSet運算子用於查詢特定位元集在欄位中都設定為 1 的文件。此運算子可讓您對欄位值執行位元操作，並在您需要檢查數值內個別位元的狀態時很有用。

參數

- field：要在其中執行位元操作的數值欄位名稱。
- value：指定應檢查哪些位元的數值位元遮罩，或要檢查的位元位置清單。數字位元遮罩可以是二進位 (0b...)、十進位、十六進位 (0x...)、八進位 (0o...) 或二進位 (BinData) 格式。在位元位置清單中，最不重要的位元位置為 0。

範例 (MongoDB Shell)

下列範例示範如何使用 $bitsAllSet運算子來尋找flags欄位具有位元遮罩指定之所有位元集的文件。

建立範例文件

```
db.collection.insert([
  { _id: 1, flags: 0b1010 },
  { _id: 2, flags: 0b1100 },
  { _id: 3, flags: 0b1110 }
])
```

查詢範例

```
db.collection.find({ flags: { $bitsAllSet: 0b1100 } })
```

輸出

```
{ "_id": 2, "flags": 12 },
```

```
{ "_id": 3, "flags": 14 }
```

在此範例中，查詢會檢查flags欄位具有位元遮罩所指定之所有位元集的文件 0b1100（代表十進位值 12)。具有 _id 2 和 3 的文件符合此條件，因為其flags欄位值已設定所有必要的位元（第 3 個和第 4 個最不重要的位元）。

程式碼範例

若要檢視使用 $bitsAllSet命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findWithBitsAllSet() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.find({ flags: { $bitsAllSet:
 0b1100 } }).toArray();
  console.log(result);

  await client.close();
}

findWithBitsAllSet();
```

Python

```
from pymongo import MongoClient

def find_with_bits_all_set():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.collection

    result = list(collection.find({ 'flags': { '$bitsAllSet': 0b1100 } }))
```

```
    print(result)

    client.close()

find_with_bits_all_set()
```

$bitsAnyClear

Amazon DocumentDB 中的$bitsAnyClear運算子用於比對清除欄位中任何指定位元的文件 （設定為 0)。這對於對存放在文件中的欄位值執行位元操作很有用。

參數

- field：要檢查的欄位。
- value：指定應檢查哪些位元的數值位元遮罩，或要檢查的位元位置清單。數字位元遮罩可以是二進位 (0b...)、十進位、十六進位 (0x...)、八進位 (0o...) 或二進位 (BinData) 格式。在位元位置清單中，最低有效位元的位置為 0。

範例 (MongoDB Shell)

下列範例示範如何使用 $bitsAnyClear 運算子來檢查items集合的 status 欄位中是否有任何位元清晰。

建立範例文件

```
db.items.insertMany([
  { "_id": 1, "status": 7 },
  { "_id": 2, "status": 15 },
  { "_id": 3, "status": 31 }
]);
```

查詢範例

```
db.items.find({ "status": { $bitsAnyClear: 8 } })
```

輸出

```
{ "_id" : 1, "status" : 7 }
```

在此範例中，查詢會檢查status欄位在位元遮罩 （二進位）中是否有任何位元清除 8(0) 的文件1000。status 值為 7（二進位 111) 的文件符合查詢，因為它在提供的位元遮罩中至少有一個位元清除。相符的清除位元是第 4 個最不重要的位元。

程式碼範例

若要檢視使用 $bitsAnyClear命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('items');

  const result = await collection.find({ "status": { $bitsAnyClear:
 8 } }).toArray();
  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.items

    result = list(collection.find({ "status": { "$bitsAnyClear": 8 } }))
    print(result)

    client.close()
```

```
example()
```

$bitsAnySet

Amazon DocumentDB 中的$bitsAnySet運算子用於查詢至少有一個位元在欄位中指定位元中設定為 1 的文件。此運算子可讓您對存放在 欄位中的值執行位元操作，以位元特性有效率地查詢和分析資料。

參數

- field：要套用位元操作的欄位名稱。
- value：指定應檢查哪些位元的數值位元遮罩，或要檢查的位元位置清單。數字位元遮罩可以是二進位 (0b...)、十進位、十六進位 (0x...)、八進位 (0o...) 或二進位 (BinData) 格式。在位元位置清單中，最不重要的位元位置為 0。

範例 (MongoDB Shell)

下列範例示範如何使用 $bitsAnySet運算子來尋找在 flags 欄位中至少設定一個位元的文件。

建立範例文件

```
db.collection.insertMany([
  { _id: 1, flags: 0b1010 },
  { _id: 2, flags: 0b1100 },
  { _id: 3, flags: 0b0011 },
  { _id: 4, flags: 0b0100 }
]);
```

查詢範例

```
db.collection.find({
  flags: { $bitsAnySet: 0b1010 }
})
```

輸出

```
{ "_id" : 1, "flags" : 10 }
{ "_id" : 2, "flags" : 12 }
```

```
{ "_id" : 3, "flags" : 3 }
```

查詢會傳回至少0b1010在 flags 欄位中設定一個位元遮罩中指定的位元的文件。

程式碼範例

若要檢視使用 $bitsAnySet命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.find({
    flags: { $bitsAnySet: 0b1010 }
  }).toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['collection']

result = list(collection.find({
    'flags': { '$bitsAnySet': 0b1010 }
}))
```

```
print(result)

client.close()
```

$comment

Amazon DocumentDB 中的$comment運算子用於將註解附加至查詢。這有助於提供有關查詢的其他內容或資訊，這有助於偵錯或記錄目的。附加的註解將顯示為 db.currentOp() 等操作輸出的一部分。

參數

- `string`：附加至查詢的註解。

範例 (MongoDB Shell)

下列範例示範如何在 Amazon DocumentDB 中使用 $comment運算子。

建立範例文件

```
db.users.insertMany([
  { name: "John Doe", age: 30, email: "john.doe@example.com" },
  { name: "Jane Smith", age: 25, email: "jane.smith@example.com" },
  { name: "Bob Johnson", age: 35, email: "bob.johnson@example.com" }
]);
```

查詢範例

```
db.users.find({ age: { $gt: 25 } }, { _id: 0, name: 1, age: 1 }).comment("Retrieve
 users older than 25");
```

輸出

```
{ "name" : "John Doe", "age" : 30 }
{ "name" : "Bob Johnson", "age" : 35 }
```

程式碼範例

若要檢視使用 $comment命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const result = await users.find({ age: { $gt: 25 } }, { projection: { _id: 0,
 name: 1, age: 1 } })
                            .comment('Retrieve users older than 25')
                            .toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    users = db.users

    result = list(users.find({ 'age': { '$gt': 25 }}, { '_id': 0, 'name': 1, 'age':
 1 })
                .comment('Retrieve users older than 25'))

    print(result)

    client.close()

if __name__ == '__main__':
```

```
    main()
```

$elemMatch

Amazon DocumentDB 中的$elemMatch運算子用於查詢陣列欄位，並傳回陣列中至少有一個元素符合指定條件的文件。當您具有具有巢狀陣列或內嵌文件的複雜資料結構時，此運算子特別有用。

規劃器 2.0 版已新增 的索引支援。 $elemMatch

參數

- field：要查詢的陣列欄位。
- query：比對陣列元素的條件。

在**$all**表達式**$elemMatch**內使用

在$all表達式$elemMatch內使用 如需在$all表達式中使用$elemMatch運算子的限制，請參閱 。

範例 (MongoDB Shell)

下列範例示範如何使用 $elemMatch運算子來尋找parts陣列至少有一個符合指定條件的元素的文件。

建立範例文件

```
db.col.insertMany([
  { _id: 1, parts: [{ part: "xyz", qty: 10 }, { part: "abc", qty: 20 }] },
  { _id: 2, parts: [{ part: "xyz", qty: 5 }, { part: "abc", qty: 10 }] },
  { _id: 3, parts: [{ part: "xyz", qty: 15 }, { part: "abc", qty: 100 }] },
  { _id: 4, parts: [{ part: "abc", qty: 150 }] }
]);
```

查詢範例

```
db.col.find({
  parts: { "$elemMatch": { part: "xyz", qty: { $lt: 11 } } }
})
```

輸出

```
{ "_id" : 1, "parts" : [ { "part" : "xyz", "qty" : 10 }, { "part" : "abc", "qty" :
 20 } ] }
{ "_id" : 2, "parts" : [ { "part" : "xyz", "qty" : 5 }, { "part" : "abc", "qty" :
 10 } ] }
```

程式碼範例

若要檢視使用 $elemMatch命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const col = db.collection('col');

  const result = await col.find({
    parts: {
      "$elemMatch": { part: "xyz", qty: { $lt: 11 } }
    }
  }).toArray();

  console.log(JSON.stringify(result, null, 2));
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
```

```
    col = db['col']

    result = list(col.find({
      'parts': {
        '$elemMatch': {'part': 'xyz', 'qty': {'$lt': 11}}
      }
    }))

    print(result)
    client.close()

example()
```

$eq

Amazon DocumentDB 中的$eq運算子用於比對欄位值等於指定值的文件。此運算子常用於 find()方法，以擷取符合指定條件的文件。

參數

- field ：檢查等式條件的欄位。
- value ：要與 欄位比較的值。

範例 (MongoDB Shell)

下列範例示範 $eq運算子的使用情況，以尋找 name 欄位等於 的所有文件"Thai Curry Palace"。

建立範例文件

```
db.restaurants.insertMany([
  { name: "Thai Curry Palace", cuisine: "Thai", features: ["Private Dining"] },
  { name: "Italian Bistro", cuisine: "Italian", features: ["Outdoor Seating"] },
  { name: "Mexican Grill", cuisine: "Mexican", features: ["Takeout"] }
]);
```

查詢範例

```
db.restaurants.find({ name: { $eq: "Thai Curry Palace" } });
```

輸出

```
{ "_id" : ObjectId("68ee586f916df9d39f3d9414"), "name" : "Thai Curry Palace",
 "cuisine" : "Thai", "features" : [ "Private Dining" ] }
```

程式碼範例

若要檢視使用 $eq命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findByName(name) {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('restaurants');

  const results = await collection.find({ name: { $eq: name } }).toArray();
  console.log(results);

  await client.close();
}

findByName("Thai Curry Palace");
```

Python

```
from pymongo import MongoClient

def find_by_name(name):
    client = MongoClient("mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false")
    db = client["test"]
    collection = db["restaurants"]

    results = list(collection.find({ "name": { "$eq": name } }))
    print(results)

    client.close()
```

```
find_by_name("Thai Curry Palace")
```

$exists

$exists 運算子用於檢查某個欄位是否存在於文件中。在 Amazon DocumentDB 中使用稀疏索引時$exists特別有用，其中索引欄位可能不存在於所有文件中。

若要使用您在查詢中建立的稀疏索引，您必須在涵蓋索引的欄位上使用 $exists 子句。如果您省略 $exists，Amazon DocumentDB 將不會使用查詢的稀疏索引。

參數

- field：要檢查是否存在的欄位名稱。
- value：布林值 (true 或 false)，指定相符文件中的欄位是否應該存在 (true) 或不存在 (false)。

範例 (MongoDB Shell)

下列範例示範在food集合中的 special_diets 欄位上使用 $exists運算子搭配稀疏索引。

建立範例文件

```
db.food.insertMany([
  { _id: 1, name: "Apple", special_diets: ["vegetarian", "gluten-free"] },
  { _id: 2, name: "Broccoli" },
  { _id: 3, name: "Chicken", special_diets: ["dairy-free"] }
]);
```

在 `special_diets` 欄位上建立稀疏索引

```
db.food.createIndex({ "special_diets": 1 }, { sparse: true, name:
 "special_diets_sparse_asc" });
```

查詢範例

```
db.food.find({ "special_diets": { $exists: true } });
```

輸出

```
[
  { "_id" : 1, "name" : "Apple", "special_diets" : [ "vegetarian", "gluten-free" ] },
  { "_id" : 3, "name" : "Chicken", "special_diets" : [ "dairy-free" ] }
]
```

程式碼範例

若要檢視使用 $exists命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('food');

  const result = await collection.find({ "special_diets": { $exists:
 true } }).toArray();
  console.log(result);

  await client.close();
}

main();
```

Python

```
import pymongo

client = pymongo.MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['food']

result = list(collection.find({"special_diets": {"$exists": True}}))
print(result)
```

```
client.close()
```

$expr

4.0 版的新功能。

Elastic 叢集不支援。

Amazon DocumentDB 中的$expr運算子可讓您在查詢語言中使用彙總表達式。它可讓您對文件中的欄位執行複雜的比較和運算，類似於您使用彙總管道階段的方式。

參數

- expression：傳回布林值的表達式，可讓您對文件欄位執行比較和運算。

範例 (MongoDB Shell)

下列範例示範如何使用 $expr運算子來尋找 manufacturingCost 欄位大於 price 欄位的所有文件。

建立範例文件

```
db.inventory.insertMany([
  { item: "abc", manufacturingCost: 500, price: 100 },
  { item: "def", manufacturingCost: 300, price: 450 },
  { item: "ghi", manufacturingCost: 400, price: 120 }
]);
```

查詢範例

```
db.inventory.find({
  $expr: {
    $gt: ["$manufacturingCost", "$price"]
  }
})
```

輸出

```
{ "_id" : ObjectId("60b9d4d68d2cac581bc5a89a"), "item" : "abc", "manufacturingCost" :
 500, "price" : 100 },
```

```
{ "_id" : ObjectId("60b9d4d68d2cac581bc5a89c"), "item" : "ghi", "manufacturingCost" :
 400, "price" : 120 }
```

程式碼範例

若要檢視使用 $expr命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('inventory');

  const result = await collection.find({
    $expr: {
      $gt: ['$manufacturingCost', '$price']
    }
  }).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['inventory']

    result = list(collection.find({
        '$expr': {
```

```
            '$gt': ['$manufacturingCost', '$price']
        }
    }))

    print(result)
    client.close()

example()
```

$gt

Amazon DocumentDB 中的$gt運算子用於選取指定欄位值大於指定值的文件。此運算子適用於根據數值比較篩選和查詢資料。

參數

- `field`：要比較的欄位。
- `value`：要比較的值。

範例 (MongoDB Shell)

下列範例示範如何使用 $gt運算子來尋找 age 欄位大於 30 的所有文件。

建立範例文件

```
db.users.insertMany([
  { name: "John", age: 25 },
  { name: "Jane", age: 32 },
  { name: "Bob", age: 45 },
  { name: "Alice", age: 28 }
]);
```

查詢範例

```
db.users.find({ age: { $gt: 30 } });
```

輸出

```
{ "_id" : ObjectId("6249e5c22a5d39884a0a0001"), "name" : "Jane", "age" : 32 },
{ "_id" : ObjectId("6249e5c22a5d39884a0a0002"), "name" : "Bob", "age" : 45 }
```

程式碼範例

若要檢視使用 $gt命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findUsersOlderThan30() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = await db.collection('users').find({ age: { $gt: 30 } }).toArray();
  console.log(users);
  await client.close();
}

findUsersOlderThan30();
```

Python

```
from pymongo import MongoClient

def find_users_older_than_30():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    users = list(db.users.find({ 'age': { '$gt': 30 } }))
    print(users)
    client.close()

find_users_older_than_30()
```

$gte

Amazon DocumentDB 中的$gte運算子用於比對大於或等於指定值的值。此運算子適用於根據數值比較篩選和查詢資料。

參數

- `field`：要檢查所提供值的欄位。
- `value`：要與 欄位比較的值。

範例 (MongoDB Shell)

下列範例示範 Amazon DocumentDB 中`$gte`運算子的使用情況，以尋找「年齡」欄位大於或等於 25 的所有文件。

建立範例文件

```
db.users.insertMany([
  { _id: 1, name: "John", age: 20 },
  { _id: 2, name: "Jane", age: 25 },
  { _id: 3, name: "Bob", age: 30 },
  { _id: 4, name: "Alice", age: 35 }
]);
```

查詢範例

```
db.users.find({ age: { $gte: 25 } }, { _id: 0, name: 1, age: 1 });
```

輸出

```
{ "name" : "Jane", "age" : 25 }
{ "name" : "Bob", "age" : 30 }
{ "name" : "Alice", "age" : 35 }
```

程式碼範例

若要檢視使用 `$gte`命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findUsersAboveAge(age) {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
```

```
  const users = db.collection('users');

  const result = await users.find({ age: { $gte: age } }, { projection: { _id: 0,
 name: 1, age: 1 } }).toArray();
  console.log(result);

  await client.close();
}

findUsersAboveAge(25);
```

Python

```
from pymongo import MongoClient

def find_users_above_age(age):
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    users = db.users

    result = list(users.find({ 'age': { '$gte': age } }, { '_id': 0, 'name': 1,
 'age': 1 }))
    print(result)

    client.close()

find_users_above_age(25)
```

$in

Amazon DocumentDB 中的$in運算子是一種邏輯查詢運算子，可讓您尋找欄位值等於陣列中指定任何值的文件。

參數

- field：要檢查所提供陣列的欄位。
- [value1, value2, ...]：要比對指定欄位的值陣列。

欄位名稱中的美元 ($)

欄位名稱中的 Dollar($) 和 dot(.) 如需在巢狀物件$in中查詢字$首欄位的限制，請參閱 。

範例 (MongoDB Shell)

下列範例示範如何使用 $in運算子來尋找 color 欄位為所提供陣列中其中一個值的文件。

建立範例文件

```
db.colors.insertMany([
  { "_id": 1, "color": "red" },
  { "_id": 2, "color": "green" },
  { "_id": 3, "color": "blue" },
  { "_id": 4, "color": "yellow" },
  { "_id": 5, "color": "purple" }
])
```

查詢範例

```
db.colors.find({ "color": { "$in": ["red", "blue", "purple"] } })
```

輸出

```
{ "_id": 1, "color": "red" },
{ "_id": 3, "color": "blue" },
{ "_id": 5, "color": "purple" }
```

程式碼範例

若要檢視使用 $in命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findByIn() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
```

```
  const collection = db.collection('colors');

  const result = await collection.find({ "color": { "$in": ["red", "blue",
 "purple"] } }).toArray();
  console.log(result);

  await client.close();
}

findByIn();
```

Python

```
from pymongo import MongoClient

def find_by_in():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.colors

    result = list(collection.find({ "color": { "$in": ["red", "blue",
 "purple"] } }))
    print(result)

    client.close()

find_by_in()
```

$jsonSchema

4.0 版的新功能。

Elastic 叢集不支援。

Amazon DocumentDB 中的$jsonSchema運算子用於根據指定的 JSON 結構描述篩選文件。此運算子可讓您查詢符合特定 JSON 結構描述的文件，確保擷取的文件符合特定的結構和資料類型需求。

使用$jsonSchema評估查詢運算子作為集合建立的一部分，您可以驗證要插入集合的文件結構描述。如需其他資訊使用 JSON 結構描述驗證，請參閱 。

參數

- required （陣列）：指定文件中的必要欄位。
- properties （物件）：定義文件中每個欄位的資料類型和其他限制條件。

範例 (MongoDB Shell)

下列範例示範如何使用 $jsonSchema運算子篩選employees集合，以僅擷取具有 name、employeeId和 age 欄位的文件，且 employeeId 欄位類型為 string。

建立範例文件

```
db.employees.insertMany([
  { "name": { "firstName": "Carol", "lastName": "Smith" }, "employeeId": "1" },
  { "name": { "firstName": "Emily", "lastName": "Brown" }, "employeeId": "2", "age":
 25 },
  { "name": { "firstName": "William", "lastName": "Taylor" }, "employeeId": 3, "age":
 24 },
  { "name": { "firstName": "Jane", "lastName": "Doe" }, "employeeId": "4" }
]);
```

查詢範例

```
db.employees.aggregate([
  { $match: {
    $jsonSchema: {
      required: ["name", "employeeId", "age"],
      properties: { "employeeId": { "bsonType": "string" } }
    }
  }}
]);
```

輸出

```
{ "_id" : ObjectId("6908e8b61f77fc26b2ecd26f"), "name" : { "firstName" : "Emily",
 "lastName" : "Brown" }, "employeeId" : "2", "age" : 25 }
```

程式碼範例

若要檢視使用 $jsonSchema命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function filterByJsonSchema() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('employees');

  const result = await collection.aggregate([
    {
      $match: {
        $jsonSchema: {
          required: ['name', 'employeeId', 'age'],
          properties: { 'employeeId': { 'bsonType': 'string' } }
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

filterByJsonSchema();
```

Python

```
from pymongo import MongoClient

def filter_by_json_schema():
  client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
  db = client['test']
  collection = db['employees']

  result = list(collection.aggregate([
    {
      '$match': {
        '$jsonSchema': {
```

```
            'required': ['name', 'employeeId', 'age'],
            'properties': {'employeeId': {'bsonType': 'string'}}
          }
        }
      }
    ]))

    print(result)
    client.close()

filter_by_json_schema()
```

$lt

Amazon DocumentDB 中的$lt運算子用於比對小於指定值的值。此運算子適用於根據數值比較篩選和查詢資料。

參數

- field：要檢查的欄位名稱。
- value：要比較的值。

範例 (MongoDB Shell)

下列範例示範如何使用 $lt運算子擷取Inventory.OnHand值小於 50 的文件。

建立範例文件

```
db.example.insertMany([
  { "_id": 1, "Inventory": { "OnHand": 25, "Sold": 60 }, "Colors": ["Red", "Blue"] },
  { "_id": 2, "Inventory": { "OnHand": 75, "Sold": 40 }, "Colors": ["Red", "White"] },
  { "_id": 3, "Inventory": { "OnHand": 10, "Sold": 85 }, "Colors": ["Red", "Green"] }
]);
```

查詢範例

```
db.example.find({ "Inventory.OnHand": { $lt: 50 } })
```

輸出

```
{ "_id" : 1, "Inventory" : { "OnHand" : 25, "Sold" : 60 }, "Colors" : [ "Red",
 "Blue" ] }
{ "_id" : 3, "Inventory" : { "OnHand" : 10, "Sold" : 85 }, "Colors" : [ "Red",
 "Green" ] }
```

程式碼範例

若要檢視使用 $lt命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('example');

  const result = await collection.find({ "Inventory.OnHand": { $lt:
 50 } }).toArray();
  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['example']

    result = list(collection.find({ "Inventory.OnHand": { "$lt": 50 } }))
    print(result)
```

```
    client.close()

example()
```

$mod

$mod 查詢運算子會選取欄位值除以除數具有指定餘數的文件。這適用於根據模數算術條件篩選文件。

參數

- `divisor`：要除以的數字。
- `remainder`：預期的剩餘值。

範例 (MongoDB Shell)

下列範例示範如何使用 $mod 運算子來尋找數量為奇數的所有訂單。

建立範例文件

```
db.orders.insertMany([
  { _id: 1, item: "Widget", quantity: 15 },
  { _id: 2, item: "Gadget", quantity: 20 },
  { _id: 3, item: "Tool", quantity: 7 },
  { _id: 4, item: "Device", quantity: 12 },
  { _id: 5, item: "Part", quantity: 9 }
]);
```

查詢範例

```
db.orders.find({ quantity: { $mod: [2, 1] } });
```

輸出

```
{ "_id" : 1, "item" : "Widget", "quantity" : 15 }
{ "_id" : 3, "item" : "Tool", "quantity" : 7 }
{ "_id" : 5, "item" : "Part", "quantity" : 9 }
```

此查詢會傳回數量除以 2 的文件，其剩餘數量為 1，有效選取所有奇數數量。

程式碼範例

若要檢視使用$mod查詢運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('orders');

  const result = await collection.find({ quantity: { $mod: [2, 1] } }).toArray();

  console.log(JSON.stringify(result, null, 2));
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['orders']

    result = list(collection.find({'quantity': {'$mod': [2, 1]}}))

    print(result)
    client.close()

example()
```

$lte

Amazon DocumentDB 中的$lte運算子用於比對指定欄位的值小於或等於指定值的文件。此運算子適用於根據數值比較篩選和查詢資料。

參數

- field：要比較的欄位。
- value：要比較的值。

範例 (MongoDB Shell)

下列範例示範 $lte運算子的使用情況，以擷取quantity欄位小於或等於 10 的文件。

建立範例文件

```
db.inventory.insertMany([
  { item: "canvas", qty: 100 },
  { item: "paint", qty: 50 },
  { item: "brush", qty: 10 },
  { item: "paper", qty: 5 }
]);
```

查詢範例

```
db.inventory.find({ qty: { $lte: 10 } });
```

輸出

```
{ "_id" : ObjectId("..."), "item" : "brush", "qty" : 10 },
{ "_id" : ObjectId("..."), "item" : "paper", "qty" : 5 }
```

程式碼範例

若要檢視使用 $lte命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require("mongodb");
```

```
async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db("test");
  const collection = db.collection("inventory");

  const result = await collection.find({ qty: { $lte: 10 } }).toArray();
  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client["test"]
    collection = db["inventory"]

    result = list(collection.find({ "qty": { "$lte": 10 } }))
    print(result)

    client.close()

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

$meta

$meta 運算子用於存取與目前查詢執行相關聯的中繼資料。此運算子主要用於文字搜尋操作，其中中繼資料可以提供有關相符文件相關性的資訊。

參數

- `textScore`：擷取文件的文字搜尋分數。此分數表示文件與文字搜尋查詢的相關性。

範例 (MongoDB Shell)

下列範例示範如何使用 $meta運算子擷取符合文字搜尋查詢之文件的文字搜尋分數。

建立範例文件

```
db.documents.insertMany([
  { _id: 1, title: "Coffee Basics", content: "Coffee is a popular beverage made from
 roasted coffee beans." },
  { _id: 2, title: "Coffee Culture", content: "Coffee coffee coffee - the ultimate
 guide to coffee brewing and coffee preparation." },
  { _id: 3, title: "Tea vs Coffee", content: "Many people prefer tea over coffee for
 its health benefits." }
]);
```

建立文字索引

```
db.documents.createIndex({ content: "text" });
```

查詢範例

```
db.documents.find(
  { $text: { $search: "coffee" } },
  { _id: 0, title: 1, content: 1, score: { $meta: "textScore" } }
).sort({ score: { $meta: "textScore" } });
```

輸出

```
[
  {
    title: 'Coffee Culture',
    content: 'Coffee coffee coffee - the ultimate guide to coffee brewing and coffee
 preparation.',
    score: 0.8897688388824463
  },
  {
    title: 'Coffee Basics',
    content: 'Coffee is a popular beverage made from roasted coffee beans.',
    score: 0.75990891456604
  },
  {
    title: 'Tea vs Coffee',
```

```
    content: 'Many people prefer tea over coffee for its health benefits.',
    score: 0.6079270839691162
  }
]
```

程式碼範例

若要檢視使用 $meta命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findWithTextScore() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('documents');

  const result = await collection.find(
    { $text: { $search: "coffee" } },
    { projection: { _id: 0, title: 1, content: 1, score: { $meta: "textScore" } } }
  ).sort({ score: { $meta: "textScore" } }).toArray();

  console.log(result);
  client.close();
}

findWithTextScore();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['documents']

for doc in collection.find(
    {'$text': {'$search': 'coffee'}},
```

```
    {'_id': 0, 'title': 1, 'content': 1, 'score': {'$meta': 'textScore'}}
).sort([('score', {'$meta': 'textScore'})]):
    print(doc)

client.close()
```

$ne

$ne 運算子用於比對欄位值不等於指定值的文件。它是一種比較運算子，可用於查詢述詞來篩選文件。

規劃器 2.0 版已新增 的索引支援。 $ne

參數

- `field`：要檢查的欄位。
- `value`：要檢查的值。

範例 (MongoDB Shell)

在此範例中，我們會在 `users`集合中找到 `status` 欄位不等於 的所有文件`"active"`。

建立範例文件

```
db.users.insertMany([
  { name: "John", status: "active" },
  { name: "Jane", status: "inactive" },
  { name: "Bob", status: "suspended" },
  { name: "Alice", status: "active" }
]);
```

查詢範例

```
db.users.find({ status: { $ne: "active" } });
```

輸出

```
[
  {
    _id: ObjectId('...'),
```

```
    name: 'Jane',
    status: 'inactive'
  },
  {
    _id: ObjectId('...'),
    name: 'Bob',
    status: 'suspended'
  }
]
```

程式碼範例

若要檢視使用 $ne命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const result = await users.find({ status: { $ne: 'active' } }).toArray();
  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
users = db['users']
```

```
result = list(users.find({ 'status': { '$ne': 'active' } }))
print(result)

client.close()
```

$nin

$nin 運算子用於比對不在指定陣列中的值。這是$in運算子的反向，其符合指定陣列中的值。

規劃器 2.0 版已新增 的索引支援。 $nin

參數

- field：要檢查的欄位。
- array：要檢查的值陣列。

欄位名稱中的美元 (**$**)

欄位名稱中的 Dollar($) 和 dot(.) 如需在巢狀物件$nin中查詢字$首欄位的限制，請參閱 。

範例 (MongoDB Shell)

下列範例示範如何使用 $nin運算子來尋找category欄位不等於 "Fiction" 或 "Mystery" 的文件。

建立範例文件

```
db.books.insertMany([
  { title: "The Great Gatsby", author: "F. Scott Fitzgerald", category: "Fiction" },
  { title: "To Kill a Mockingbird", author: "Harper Lee", category: "Fiction" },
  { title: "The Girl on the Train", author: "Paula Hawkins", category: "Mystery" },
  { title: "The Martian", author: "Andy Weir", category: "Science Fiction" },
  { title: "The Alchemist", author: "Paulo Coelho", category: "Philosophy" }
])
```

查詢範例

```
db.books.find({
  category: {
    $nin: ["Fiction", "Mystery"]
  }
```

```
})
```

輸出

```
[
  {
    _id: ObjectId('...'),
    title: 'The Martian',
    author: 'Andy Weir',
    category: 'Science Fiction'
  },
  {
    _id: ObjectId('...'),
    title: 'The Alchemist',
    author: 'Paulo Coelho',
    category: 'Philosophy'
  }
]
```

程式碼範例

若要檢視使用 $nin命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findBooksNotInCategories(categories) {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const books = await db.collection('books').find({
    category: {
      $nin: categories
    }
  }).toArray();
  console.log(books);
  client.close();
}

findBooksNotInCategories(['Fiction', 'Mystery']);
```

Python

```
from pymongo import MongoClient

def find_books_not_in_categories(categories):
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    books = list(db.books.find({
        'category': {
            '$nin': categories
        }
    }))
    print(books)
    client.close()

find_books_not_in_categories(['Fiction', 'Mystery'])
```

$nor

$nor 運算子用於比對沒有任何指定查詢條件為 true 的文件。它類似於邏輯「NOR」操作，如果沒有任何運算元為 true，則結果為 true。

參數

- expression1：要評估的第一個表達式。
- expression2：要評估的第二個表達式。
- expressionN：要評估的其他表達式。

範例 (MongoDB Shell)

下列範例會擷取qty欄位不小於 20 且size欄位不等於「XL」的文件，以示範$nor運算子的使用情況。

建立範例文件

```
db.items.insertMany([
  { qty: 10, size: "M" },
  { qty: 15, size: "XL" },
```

```
  { qty: 25, size: "L" },
  { qty: 30, size: "XL" }
])
```

查詢範例

```
db.items.find({
  $nor: [
    { qty: { $lt: 20 } },
    { size: "XL" }
  ]
})
```

輸出

```
[
  { "_id" : ObjectId("..."), "qty" : 25, "size" : "L" }
]
```

程式碼範例

若要檢視使用 $nor命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('items');

  const result = await collection.find({
    $nor: [
      { qty: { $lt: 20 } },
      { size: "XL" }
    ]
  }).toArray();

  console.log(result);
```

```
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['items']

    result = list(collection.find({
        '$nor': [
            { 'qty': { '$lt': 20 } },
            { 'size': 'XL' }
        ]
    }))

    print(result)

    client.close()

example()
```

$not

$not 運算子用於否定指定表達式的結果。它可讓您選取指定條件不相符的文件。

規劃器 2.0 版新增了 $not {eq}和 的索引支援。 $not {in}

參數

- expression：要否定的表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $not運算子來尋找status欄位不等於「作用中」的文件。

建立範例文件

```
db.users.insertMany([
  { name: "John", status: "active" },
  { name: "Jane", status: "inactive" },
  { name: "Bob", status: "pending" },
  { name: "Alice", status: "active" }
]);
```

查詢範例

```
db.users.find({ status: { $not: { $eq: "active" } } });
```

輸出

```
[
  {
    _id: ObjectId('...'),
    name: 'Jane',
    status: 'inactive'
  },
  {
    _id: ObjectId('...'),
    name: 'Bob',
    status: 'pending'
  }
]
```

程式碼範例

若要檢視使用 $not命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient, Filters } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('users');
```

```
  const result = await collection.find({
    status: { $not: { $eq: "active" } }
  }).toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient
from bson.son import SON

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['users']

result = collection.find({
    "status": {"$not": {"$eq": "active"}}
})

for doc in result:
    print(doc)

client.close()
```

$or

$or 運算子用於在兩個或多個表達式的陣列上執行邏輯 OR 操作。它會傳回至少符合其中一個表達式的文件。當您需要查詢符合任何一種條件的文件時，此運算子非常有用。

參數

- expression1：要評估的第一個表達式。
- expression2：要評估的第二個表達式。

- `...`：要評估的其他表達式（選用）。

範例 (MongoDB Shell)

下列範例示範 $or運算子的用法，以尋找 make為 "TruckForYou" 且模型為 "Heavy H1" 或 "SportForYou" 且模型為 "Bolid 1" 的文件。

建立範例文件

```
db.cars.insertMany([
  { make: "TruckForYou", model: "Heavy H1", year: 2020 },
  { make: "SportForYou", model: "Bolid 1", year: 2021 },
  { make: "TruckForYou", model: "Cargo 5", year: 2019 },
  { make: "SportForYou", model: "Racer 2", year: 2022 }
]);
```

查詢範例

```
db.cars.find({
  $or: [
    { make: "TruckForYou", model: "Heavy H1" },
    { make: "SportForYou", model: "Bolid 1" }
  ]
});
```

輸出

```
[
  {
    _id: ObjectId('...'),
    make: 'TruckForYou',
    model: 'Heavy H1',
    year: 2020
  },
  {
    _id: ObjectId('...'),
    make: 'SportForYou',
    model: 'Bolid 1',
    year: 2021
  }
]
```

程式碼範例

若要檢視使用 $or命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findCarsByMakeModel() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const cars = db.collection('cars');

  const result = await cars.find({
    $or: [
      { make: "TruckForYou", model: "Heavy H1" },
      { make: "SportForYou", model: "Bolid 1" }
    ]
  }).toArray();

  console.log(result);
  client.close();
}

findCarsByMakeModel();
```

Python

```
from pymongo import MongoClient

def find_cars_by_make_model():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    cars = db.cars

    result = list(cars.find({
        '$or': [
            {'make': 'TruckForYou', 'model': 'Heavy H1'},
            {'make': 'SportForYou', 'model': 'Bolid 1'}
```

```
        ]
    }))

    print(result)
    client.close()

find_cars_by_make_model()
```

$regex

$regex 運算子可讓您對字串欄位執行規則運算式比對。它是根據複雜模式搜尋和篩選文件的強大工具。

參數

- regular expression：要比對 欄位的規則表達式模式。
- $options：（選用）提供修改搜尋行為的選項，例如區分大小寫、全域比對等。

範例 (MongoDB Shell)

以下範例示範 $regex運算子的使用情況，以搜尋 "name" 欄位符合特定模式的文件。

建立範例文件

```
db.users.insertMany([
  { name: "John Doe" },
  { name: "Jane Smith" },
  { name: "Alice Johnson" },
  { name: "Bob Williams" },
  { name: "Charlie Davis" }
]);
```

查詢範例

```
db.users.find({ name: { $regex: /^A/ } })
```

輸出

```
[
```

```
  { "_id" : ObjectId("..."), "name" : "Alice Johnson" }
]
```

此查詢將傳回 "name" 欄位以字母 "A" 開頭的所有文件。

程式碼範例

若要檢視使用 $regex命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('users');

  const results = await collection.find({ name: { $regex: /^A/ } }).toArray();
  console.log(results);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['users']

    results = list(collection.find({ 'name': { '$regex': '^A' } }))
    print(results)

    client.close()
```

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

$sampleRate

8.0.1 版的新功能。

Elastic 叢集不支援。

Amazon DocumentDB 中的$sampleRate運算子會根據指定的速率，比對輸入文件的隨機範例。您可以在find()查詢篩選條件或彙總$match階段中使用它。由於選擇是概率的，因此傳回的文件數量是近似值，並且可能因執行而異。

參數

- value：範圍 [0, 1]（包含）中的數字，直接傳遞給運算子（例如 { $sampleRate: 0.3 }），這會設定包含每個文件的機率。的值不會0傳回任何文件； 的值會1傳回所有文件。

範例 (MongoDB Shell)

下列範例使用 $sampleRate來傳回集合中大約 30% 的文件。

建立範例文件

```
db.events.insertMany([
  { _id: 1, type: "click" },
  { _id: 2, type: "view" },
  { _id: 3, type: "click" },
  { _id: 4, type: "view" },
  { _id: 5, type: "purchase" }
]);
```

查詢範例

```
db.events.aggregate([
  { $match: { $sampleRate: 0.3 } }
]);
```

輸出

操作會傳回文件的隨機子集。由於 $sampleRate 具有機率，因此每次查詢執行時，傳回的特定文件及其計數會有所不同。

程式碼範例

若要檢視使用 $sampleRate命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('events');

    // Return approximately 30% of the documents
    const result = await collection.aggregate([
      { $match: { $sampleRate: 0.3 } }
    ]).toArray();

    console.log(result);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
    try:
        db = client['test']
        collection = db['events']

        # Return approximately 30% of the documents
        result = list(collection.aggregate([
            { '$match': { '$sampleRate': 0.3 } }
        ]))

        print(result)
    finally:
        client.close()

example()
```

$slice

$slice 投影運算子會限制查詢結果中傳回的陣列元素數目。它可讓您從陣列欄位的開頭或結尾擷取特定數量的元素，而無需載入整個陣列。

參數

- field：要投影的陣列欄位。
- count：要傳回的元素數目。正值會從開頭傳回元素，從結尾傳回負值。

範例 (MongoDB Shell)

下列範例示範如何使用$slice投影運算子，只傳回陣列欄位中的前兩個項目。

建立範例文件

```
db.inventory.insertMany([
  { _id: 1, item: "notebook", tags: ["office", "school", "supplies", "writing"] },
  { _id: 2, item: "pen", tags: ["office", "writing"] },
  { _id: 3, item: "folder", tags: ["office", "supplies", "storage", "organization"] }
]);
```

查詢範例

```
db.inventory.find(
```

```
  {},
  { item: 1, tags: { $slice: 2 } }
)
```

輸出

```
{ "_id" : 1, "item" : "notebook", "tags" : [ "office", "school" ] }
{ "_id" : 2, "item" : "pen", "tags" : [ "office", "writing" ] }
{ "_id" : 3, "item" : "folder", "tags" : [ "office", "supplies" ] }
```

程式碼範例

若要檢視使用$slice投影運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('inventory');

  const result = await collection.find(
    {},
    { projection: { item: 1, tags: { $slice: 2 } } }
  ).toArray();

  console.log(JSON.stringify(result, null, 2));
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['inventory']

    result = list(collection.find(
        {},
        {'item': 1, 'tags': {'$slice': 2}}
    ))

    print(result)
    client.close()

example()
```

$size

$size 查詢運算子會比對陣列欄位具有指定數量元素的文件。這適用於根據陣列長度篩選文件。

參數

- field：要檢查的陣列欄位。
- count：陣列必須包含的確切元素數量。

範例 (MongoDB Shell)

以下範例示範如何使用 $size運算子來尋找具有恰好三個標籤的所有產品。

建立範例文件

```
db.products.insertMany([
  { _id: 1, name: "Laptop", tags: ["electronics", "computers", "portable"] },
  { _id: 2, name: "Mouse", tags: ["electronics", "accessories"] },
  { _id: 3, name: "Desk", tags: ["furniture", "office", "workspace"] },
  { _id: 4, name: "Monitor", tags: ["electronics"] }
]);
```

查詢範例

```
db.products.find({ tags: { $size: 3 } });
```

輸出

```
{ "_id" : 1, "name" : "Laptop", "tags" : [ "electronics", "computers", "portable" ] }
{ "_id" : 3, "name" : "Desk", "tags" : [ "furniture", "office", "workspace" ] }
```

程式碼範例

若要檢視使用$size查詢運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  const result = await collection.find({ tags: { $size: 3 } }).toArray();

  console.log(JSON.stringify(result, null, 2));
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['products']

    result = list(collection.find({'tags': {'$size': 3}}))

    print(result)
```

```
    client.close()

example()
```

$text

$text 運算子用於對文件集合中的文字索引欄位執行全文搜尋。此運算子可讓您搜尋包含特定單字或片語的文件，並可與其他查詢運算子結合，根據其他條件篩選結果。

參數

- $search：要搜尋的文字字串。

範例 (MongoDB Shell)

下列範例示範如何使用 $text運算子來搜尋包含「興趣」一詞的文件，並根據「star_rating」欄位篩選結果。

建立範例文件

```
db.test.insertMany([
  { "_id": 1, "star_rating": 4, "comments": "apple is red" },
  { "_id": 2, "star_rating": 5, "comments": "pie is delicious" },
  { "_id": 3, "star_rating": 3, "comments": "apples, oranges - healthy fruit" },
  { "_id": 4, "star_rating": 2, "comments": "bake the apple pie in the oven" },
  { "_id": 5, "star_rating": 5, "comments": "interesting couch" },
  { "_id": 6, "star_rating": 5, "comments": "interested in couch for sale, year 2022" }
]);
```

建立文字索引

```
db.test.createIndex({ comments: "text" });
```

查詢範例

```
db.test.find({$and: [{star_rating: 5}, {$text: {$search: "interest"}}]})
```

輸出

```
{ "_id" : 5, "star_rating" : 5, "comments" : "interesting couch" }
{ "_id" : 6, "star_rating" : 5, "comments" : "interested in couch for sale, year
 2022" }
```

上述命令會傳回具有文字索引欄位的文件，其中包含任何形式的「興趣」和等於 5 的「star_rating」。

程式碼範例

若要檢視使用 $text命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function searchDocuments() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('test');

  const result = await collection.find({
    $and: [
      { star_rating: 5 },
      { $text: { $search: 'interest' } }
    ]
  }).toArray();

  console.log(result);
  client.close();
}

searchDocuments();
```

Python

```
from pymongo import MongoClient

def search_documents():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
    db = client['test']
    collection = db.test

    result = list(collection.find({
        '$and': [
            {'star_rating': 5},
            {'$text': {'$search': 'interest'}}
        ]
    }))

    print(result)
    client.close()

search_documents()
```

$type

$type 運算子用於檢查文件中欄位的資料類型。它可以在需要特定類型操作或驗證時使用。$type 運算子會傳回評估表達式的 BSON 類型。傳回的類型是字串，對應至欄位或表達式的類型。

規劃器 2.0 版已新增 的索引支援。 $type

參數

- expression ：要評估的表達式。

範例 (MongoDB Shell)

建立範例文件

```
db.documents.insertMany([
  { _id: 1, name: "John", age: 30, email: "john@example.com" },
  { _id: 2, name: "Jane", age: "25", email: 123456 },
  { _id: 3, name: 123, age: true, email: null }
]);
```

查詢範例

```
db.documents.find({
  $or: [
    { age: { $type: "number" } },
```

```
    { email: { $type: "string" } },
    { name: { $type: "string" } }
  ]
})
```

輸出

```
[
  { "_id": 1, "name": "John", "age": 30, "email": "john@example.com" },
  { "_id": 2, "name": "Jane", "age": "25", "email": 123456 }
]
```

此查詢會傳回 age 欄位類型為 "number"、 email 欄位類型為 "string" 且 name 欄位類型為 "string" 的文件。

程式碼範例

若要檢視使用 $type命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function findByType() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('documents');

  const results = await collection.find({
    $or: [
      { age: { $type: 'number' } },
      { email: { $type: 'string' } },
      { name: { $type: 'string' } }
    ]
  }).toArray();

  console.log(results);
  client.close();
}
```

```
findByType();
```

Python

```
from pymongo import MongoClient

def find_by_type():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['documents']

    results = list(collection.find({
        '$or': [
            {'age': {'$type': 'number'}},
            {'email': {'$type': 'string'}},
            {'name': {'$type': 'string'}}
        ]
    }))

    print(results)
    client.close()

find_by_type()
```

更新運算子

本節提供有關 Amazon DocumentDB 支援的更新運算子的詳細資訊。

主題

- $
- $[]
- $[<identifier>]
- $addToSet
- $bit
- $currentDate
- $each

- $inc
- $max
- $min
- $mul
- $pop
- $position
- $pull
- $pullAll
- $push
- $rename
- $set
- $setOnInsert
- $slice
- $sort
- $unset

$

$ 位置運算子會更新符合查詢條件的第一個陣列元素。它可做為相符陣列元素位置的預留位置。

參數

- `field.$`：具有位置運算子的陣列欄位，用於更新第一個相符的元素。

範例 (MongoDB Shell)

下列範例示範如何使用$位置運算子更新特定陣列元素。

建立範例文件

```
db.inventory.insertMany([
  { _id: 1, item: "Widget", quantities: [10, 20, 30] },
  { _id: 2, item: "Gadget", quantities: [5, 15, 25] }
]);
```

查詢範例

```
db.inventory.updateOne(
  { _id: 1, quantities: 20 },
  { $set: { "quantities.$": 22 } }
);
```

輸出

```
{
  "_id" : 1,
  "item" : "Widget",
  "quantities" : [ 10, 22, 30 ]
}
```

程式碼範例

若要檢視使用$位置運算子的程式碼範例，請選擇您要使用之語言的標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function updateDocument() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('inventory');

  await collection.updateOne(
    { _id: 1, quantities: 20 },
    { $set: { "quantities.$": 22 } }
  );

  const updatedDocument = await collection.findOne({ _id: 1 });
  console.log(updatedDocument);

  await client.close();
}

updateDocument();
```

Python

```
from pymongo import MongoClient

def update_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.inventory

    collection.update_one(
        {'_id': 1, 'quantities': 20},
        {'$set': {'quantities.$': 22}}
    )

    updated_document = collection.find_one({'_id': 1})
    print(updated_document)

    client.close()

update_document()
```

$[]

$[] 所有位置運算子都會更新陣列中的所有元素。當您需要修改陣列欄位中的每個元素時，會使用它。

參數

- field.$[]：具有所有位置運算子的陣列欄位，用於更新所有元素。

範例 (MongoDB Shell)

下列範例示範如何使用 $[]運算子更新所有陣列元素。

建立範例文件

```
db.products.insertOne({
  _id: 1,
  name: "Laptop",
```

```
  prices: [1000, 1100, 1200]
});
```

查詢範例

```
db.products.updateOne(
  { _id: 1 },
  { $inc: { "prices.$[]": 50 } }
);
```

輸出

```
{
  "_id" : 1,
  "name" : "Laptop",
  "prices" : [ 1050, 1150, 1250 ]
}
```

程式碼範例

若要檢視使用 $[] 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function updateDocument() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  await collection.updateOne(
    { _id: 1 },
    { $inc: { "prices.$[]": 50 } }
  );

  const updatedDocument = await collection.findOne({ _id: 1 });
  console.log(updatedDocument);

  await client.close();
```

```
}

updateDocument();
```

Python

```
from pymongo import MongoClient

def update_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.products

    collection.update_one(
        {'_id': 1},
        {'$inc': {'prices.$[]': 50}}
    )

    updated_document = collection.find_one({'_id': 1})
    print(updated_document)

    client.close()

update_document()
```

$[<identifier>]

$[<identifier>] 篩選的位置運算子會更新符合指定篩選條件的所有陣列元素。它與 arrayFilters選項搭配使用，以選擇性地更新陣列元素。

參數

- field.$[identifier]：具有篩選位置運算子的陣列欄位。
- arrayFilters：篩選條件陣列，可決定要更新的元素。

範例 (MongoDB Shell)

下列範例示範如何使用 $[<identifier>]運算子根據條件更新特定陣列元素。

建立範例文件

```
db.students.insertOne({
  _id: 1,
  name: "Alice",
  grades: [
    { subject: "Math", score: 85 },
    { subject: "Science", score: 92 },
    { subject: "History", score: 78 }
  ]
});
```

查詢範例

```
db.students.updateOne(
  { _id: 1 },
  { $inc: { "grades.$[elem].score": 5 } },
  { arrayFilters: [{ "elem.score": { $gte: 80 } }] }
);
```

輸出

```
{
  "_id" : 1,
  "name" : "Alice",
  "grades" : [
    { "subject" : "Math", "score" : 90 },
    { "subject" : "Science", "score" : 97 },
    { "subject" : "History", "score" : 78 }
  ]
}
```

程式碼範例

若要檢視使用 $[<identifier>] 運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function updateDocument() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('students');

  await collection.updateOne(
    { _id: 1 },
    { $inc: { "grades.$[elem].score": 5 } },
    { arrayFilters: [{ "elem.score": { $gte: 80 } }] }
  );

  const updatedDocument = await collection.findOne({ _id: 1 });
  console.log(updatedDocument);

  await client.close();
}

updateDocument();
```

Python

```
from pymongo import MongoClient

def update_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.students

    collection.update_one(
        {'_id': 1},
        {'$inc': {'grades.$[elem].score': 5}},
        array_filters=[{'elem.score': {'$gte': 80}}]
    )

    updated_document = collection.find_one({'_id': 1})
    print(updated_document)

    client.close()

update_document()
```

$addToSet

只有在陣列中沒有值時，Amazon DocumentDB 中的$addToSet運算子才會用來將值新增至陣列。這有助於確保陣列包含唯一的元素。

參數

- field：要更新的欄位。
- value：要新增至陣列欄位的值。這可以是單一值或表達式。

範例 (MongoDB Shell)

下列範例示範如何使用 $addToSet運算子將唯一元素新增至陣列。

建立範例文件

```
db.products.insertMany([
  { "_id": 1, "item": "apple", "tags": ["fruit", "red", "round"] },
  { "_id": 2, "item": "banana", "tags": ["fruit", "yellow"] },
  { "_id": 3, "item": "cherry", "tags": ["fruit", "red"] }
])
```

查詢範例

```
db.products.update(
  { "item": "apple" },
  { $addToSet: { "tags": "green" } }
)
```

輸出

```
{ "_id": 1, "item": "apple", "tags": ["fruit", "red", "round", "green"] }
```

在此範例中，$addToSet運算子會將 "green" 標籤新增至文件的 "tags" 陣列，其中 "item" 欄位為 "apple"。由於 "green" 尚未在陣列中，因此已新增。

程式碼範例

若要檢視使用 $addToSet命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  await collection.updateOne(
    { "item": "apple" },
    { $addToSet: { "tags": "green" } }
  );

  const updatedDoc = await collection.findOne({ "item": "apple" });
  console.log(updatedDoc);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.products

    collection.update_one(
        {"item": "apple"},
        {"$addToSet": {"tags": "green"}}
    )

    updated_doc = collection.find_one({"item": "apple"})
    print(updated_doc)
```

```
    client.close()

example()
```

$bit

Amazon DocumentDB 中的$bit運算子可讓您對指定欄位的位元執行位元操作。這對於設定、清除或檢查數字中個別位元的狀態等任務很有用。

參數

- field：要在其中執行位元操作的欄位。
- and：整數值，用於對 欄位執行位元 AND 操作。
- or：用於對 欄位執行位元 OR 操作的整數值。
- xor：用於對 欄位執行位元 XOR 操作的整數值。

範例 (MongoDB Shell)

下列範例示範如何使用 $bit運算子對數值欄位執行位元操作。

建立範例文件

```
db.numbers.insert([
  { "_id": 1, "number": 5 },
  { "_id": 2, "number": 12 }
])
```

查詢範例

```
db.numbers.update(
  { "_id": 1 },
  { "$bit": { "number": { "and": 3 } } }
)
```

輸出

```
{
```

```
  "_id": 1,
  "number": 1
}
```

在此範例中，運算$bit子用於對 文件的「數字」欄位執行位元 AND 操作，其 _id為 1。結果是 "number" 欄位的值設定為 1，這是原始值 5 與值 3 之間的位元 AND 操作的結果。

程式碼範例

若要檢視使用 $bit命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  await collection.updateOne(
    { "_id": 1 },
    { "$bit": { "number": { "and": 3 } } }
  );

  const result = await collection.findOne({ "_id": 1 });
  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
db = client['test']
collection = db['numbers']

collection.update_one(
    {"_id": 1},
    {"$bit": {"number": {"and": 3}}}
)

result = collection.find_one({"_id": 1})
print(result)

client.close()
```

$currentDate

$currentDate 運算子用於將欄位的值設定為目前的日期和時間。此運算子適用於在插入或更新文件時，自動更新具有目前時間戳記的欄位。

參數

- field：要更新為目前日期和時間的欄位。
- type：（選用）指定要用於目前日期的 BSON 類型。可以是 date 或 timestamp。

範例 (MongoDB Shell)

下列範例示範如何在插入新文件時，使用 $currentDate 運算子將 lastModified 欄位設定為目前的日期和時間。

建立範例文件

```
db.users.insert({
  name: "John Doe",
  email: "john.doe@example.com"
})
```

查詢範例

```
db.users.updateOne(
  { name: "John Doe" },
```

```
  { $currentDate: { lastModified: true } }
)
```

檢視更新的文件

```
db.users.findOne({ name: "John Doe" })
```

輸出

```
{
  _id: ObjectId('...'),
  name: 'John Doe',
  email: 'john.doe@example.com',
  lastModified: ISODate('2025-10-25T22:50:29.963Z')
}
```

程式碼範例

若要檢視使用 $currentDate命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function updateUserWithCurrentDate() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  await users.updateOne(
    { name: 'John Doe' },
    { $currentDate: { lastModified: true } }
  );

  console.log('User updated with current date');
  client.close();
}

updateUserWithCurrentDate();
```

Python

```
from pymongo import MongoClient

def update_user_with_current_date():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    users = db.users

    result = users.update_one(
        {'name': 'John Doe'},
        {'$currentDate': {'lastModified': True}}
    )

    print('User updated with current date')
    client.close()

update_user_with_current_date()
```

$each

$each 運算子會與其他更新運算子搭配使用，例如 $push和 $addToSet，以將多個值新增至陣列欄位。它允許在單一操作中將多個元素新增至陣列，而不必執行多個更新操作。

參數

- value：要新增至陣列欄位的值陣列。

範例 (MongoDB Shell)

下列範例示範使用 $each運算子搭配 $push運算子，將多個元素新增至陣列欄位。

建立範例文件

```
db.fruits.insertOne({
  _id: 1,
  fruits: ["apple", "banana"]
})
```

查詢範例

```
db.fruits.updateOne(
  { _id: 1 },
  { $push: { fruits: { $each: ["cherry", "durian", "elderberry"] } } }
)
```

檢視更新的文件

```
db.fruits.findOne({ _id: 1 })
```

輸出

```
{
  _id: 1,
  fruits: [ 'apple', 'banana', 'cherry', 'durian', 'elderberry' ]
}
```

程式碼範例

若要檢視使用 $each命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('fruits');

  await collection.updateOne(
    { _id: 1 },
    { $push: { fruits: { $each: ["cherry", "durian", "elderberry"] } } }
  );

  await client.close();
}
```

```
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['fruits']

    collection.update_one(
        {'_id': 1},
        {'$push': {'fruits': {'$each': ['cherry', 'durian', 'elderberry']}}}
    )

    client.close()

example()
```

$inc

$inc 運算子用於將欄位的值增加指定的數量。它用於更新數值欄位，例如計數器或評分，而無需擷取目前值、計算新值，然後更新欄位。

參數

- field：要遞增的欄位名稱。
- amount：要增加欄位的數量。這可以是正值或負值。

範例 (MongoDB Shell)

下列範例示範如何使用 $inc運算子來增加文件age的欄位。

建立範例文件

```
db.users.insertOne({_id: 123, name: "John Doe", age: 30})
```

查詢範例

```
db.users.updateOne({_id: 123}, {$inc: {age: 1}})
```

檢視更新的文件

```
db.users.findOne({_id: 123})
```

輸出

```
{ "_id" : 123, "name" : "John Doe", "age" : 31 }
```

程式碼範例

若要檢視使用 $inc命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function updateWithInc() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('users');

  const result = await collection.updateOne(
    { _id: 123 },
    { $inc: { age: 1 } }
  );

  console.log(result);

  await client.close();
}

updateWithInc();
```

Python

```
from pymongo import MongoClient
```

```
def update_with_inc():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['users']

    result = collection.update_one(
        {'_id': 123},
        {'$inc': {'age': 1}}
    )

    print(result.modified_count)

    client.close()

update_with_inc()
```

$max

只有在指定的值大於目前的欄位值時，$max更新運算子才會更新欄位的值。此運算子有助於跨更新維持最大值。

參數

- `field`：要更新的欄位。
- `value`：要與目前欄位值比較的值。

範例 (MongoDB Shell)

下列範例示範如何使用 $max 運算子更新玩家記錄的最高分數。

建立範例文件

```
db.scores.insertMany([
  { _id: 1, player: "Alice", highScore: 85 },
  { _id: 2, player: "Bob", highScore: 92 },
  { _id: 3, player: "Charlie", highScore: 78 }
])
```

更新範例

```
db.scores.updateOne(
  { _id: 1 },
  { $max: { highScore: 95 } }
)
```

結果

Alice highScore的欄位會更新為 95，因為 95 大於目前值 85。

```
{ "_id": 1, "player": "Alice", "highScore": 95 }
```

程式碼範例

若要檢視使用 $max命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('scores');

  const result = await collection.updateOne(
    { _id: 1 },
    { $max: { highScore: 95 } }
  );

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient
```

```
def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['scores']

    result = collection.update_one(
        { '_id': 1 },
        { '$max': { 'highScore': 95 } }
    )

    print(result)
    client.close()

example()
```

$min

只有在指定的值小於目前的欄位值時，$min更新運算子才會更新欄位的值。此運算子有助於跨更新維持最小值。

參數

- field：要更新的欄位。
- value：要與目前欄位值比較的值。

範例 (MongoDB Shell)

下列範例示範如何使用 $min 運算子更新氣象站的最低記錄溫度。

建立範例文件

```
db.weather.insertMany([
  { _id: 1, station: "Station A", lowestTemp: 15 },
  { _id: 2, station: "Station B", lowestTemp: 20 },
  { _id: 3, station: "Station C", lowestTemp: 18 }
])
```

更新範例

```
db.weather.updateOne(
  { _id: 1 },
  { $min: { lowestTemp: 12 } }
)
```

結果

工作站 A lowestTemp的欄位會更新為 12，因為 12 小於目前值 15。

```
{ "_id": 1, "station": "Station A", "lowestTemp": 12 }
```

程式碼範例

若要檢視使用 $min命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('weather');

  const result = await collection.updateOne(
    { _id: 1 },
    { $min: { lowestTemp: 12 } }
  );

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient
```

```
def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['weather']

    result = collection.update_one(
        { '_id': 1 },
        { '$min': { 'lowestTemp': 12 } }
    )

    print(result)
    client.close()

example()
```

$mul

Amazon DocumentDB 中的$mul運算子用於將欄位的值乘以指定的數字。這對於以原子方式和一致的方式更新多個文件非常有用，例如根據信用卡狀態更新飛行里程。

參數

- field：要乘以的欄位。
- multiplier：要將欄位值乘以的數字。

範例 (MongoDB Shell)

此範例示範如何使用 $mul 運算子，將 credit_card 欄位為 之所有文件flight_miles的值加倍true。

建立範例文件

```
db.miles.insertMany([
  { "_id": 1, "member_since": new Date("1987-01-01"), "credit_card": false,
 "flight_miles": [1205, 2560, 880] },
  { "_id": 2, "member_since": new Date("1982-01-01"), "credit_card": true,
 "flight_miles": [2410, 5120, 1780, 5560] },
  { "_id": 3, "member_since": new Date("1999-01-01"), "credit_card": true,
 "flight_miles": [2410, 1760] }
```

```
]);
```

查詢範例

```
db.miles.update(
  { "credit_card": { "$eq": true } },
  { "$mul": { "flight_miles.$[]": NumberInt(2) } },
  { "multi": true }
);
```

輸出

```
{ "_id" : 1, "member_since" : ISODate("1987-01-01T00:00:00Z"), "credit_card" : false,
 "flight_miles" : [ 1205, 2560, 880 ] }
{ "_id" : 2, "member_since" : ISODate("1982-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 4820, 10240, 3560, 11120 ] }
{ "_id" : 3, "member_since" : ISODate("1999-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 4820, 3520 ] }
```

對於有信用卡的客戶，他們的飛行里程已加倍。

$[] 位置陣列運算子用於將 $mul操作套用至flight_miles陣列中的每個元素。

程式碼範例

若要檢視使用 $mul命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function updateFlightMiles() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('miles');

  await collection.updateMany(
    { credit_card: true },
    { $mul: { 'flight_miles.$[]': 2 } }
  );
```

```
  const documents = await collection.find().toArray();
  console.log(documents);

  await client.close();
}

updateFlightMiles();
```

Python

```
from pymongo import MongoClient

def update_flight_miles():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.miles

    collection.update_many(
        {'credit_card': True},
        {'$mul': {'flight_miles.$[]': 2}}
    )

    documents = list(collection.find())
    print(documents)

    client.close()

update_flight_miles()
```

$pop

Amazon DocumentDB 中的$pop運算子用於從陣列欄位移除第一個或最後一個元素。當您需要維護固定大小的陣列或在文件中實作類似佇列的資料結構時，此功能特別有用。

參數

- `field`：要從中移除元素的陣列欄位名稱。
- `value`：整數值，決定要移除之元素的位置。的值會1移除最後一個元素，而 的值則會-1移除第一個元素。

範例 (MongoDB Shell)

此範例示範如何使用 $pop 運算子從陣列欄位移除第一個和最後一個元素。

建立範例文件

```
db.users.insertMany([
  { "_id": 1, "name": "John Doe", "hobbies": ["reading", "swimming", "hiking"] },
  { "_id": 2, "name": "Jane Smith", "hobbies": ["cooking", "gardening", "painting"] }
])
```

查詢範例

```
// Remove the first element from the "hobbies" array
db.users.update({ "_id": 1 }, { $pop: { "hobbies": -1 } })

// Remove the last element from the "hobbies" array
db.users.update({ "_id": 2 }, { $pop: { "hobbies": 1 } })
```

輸出

```
{ "_id" : 1, "name" : "John Doe", "hobbies" : [ "swimming", "hiking" ] }
{ "_id" : 2, "name" : "Jane Smith", "hobbies" : [ "cooking", "gardening" ] }
```

程式碼範例

若要檢視使用 $pop命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('users');

  // Remove the first element from the "hobbies" array
  await collection.updateOne({ "_id": 1 }, { $pop: { "hobbies": -1 } });
```

```
  // Remove the last element from the "hobbies" array
  await collection.updateOne({ "_id": 2 }, { $pop: { "hobbies": 1 } });

  const users = await collection.find().toArray();
  console.log(users);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['users']

    # Remove the first element from the "hobbies" array
    collection.update_one({"_id": 1}, {"$pop": {"hobbies": -1}})

    # Remove the last element from the "hobbies" array
    collection.update_one({"_id": 2}, {"$pop": {"hobbies": 1}})

    users = list(collection.find())
    print(users)

    client.close()

example()
```

$position

Amazon DocumentDB `$position` 中的修飾詞會指定$push運算子插入元素的陣列中的位置。`$position` 如果沒有修改器，$push運算子會將元素插入陣列的結尾。

參數

- field：要更新的陣列欄位。
- num：陣列中應插入元素的位置，以零為基礎編製索引。

注意：若要使用$position修飾詞，它必須與$each修飾詞一起顯示。

範例 (MongoDB Shell)

下列範例示範如何使用 $position運算子在專案管理系統中的特定位置插入任務。

建立範例文件

```
db.projects.insertOne({ "_id": 1, "name": "Website Redesign", "tasks": ["Design
 mockups"] })
```

查詢範例 1 - 在開頭新增緊急任務

```
db.projects.updateOne(
   { _id: 1 },
   {
     $push: {
        tasks: {
           $each: ["Security audit", "Performance review"],
           $position: 0
        }
     }
   }
)
```

輸出 1

```
{ "_id": 1, "name": "Website Redesign", "tasks": ["Security audit", "Performance
 review", "Design mockups"] }
```

查詢範例 2 - 在特定位置新增任務

```
db.projects.insertOne({ "_id": 2, "name": "Mobile App", "tasks": ["Setup project",
 "Create wireframes", "Deploy to store"] })

db.projects.updateOne(
```

```
   { _id: 2 },
   {
     $push: {
        tasks: {
           $each: ["Code review", "Testing phase"],
           $position: 2
        }
     }
   }
)
```

輸出 2

```
{ "_id": 2, "name": "Mobile App", "tasks": ["Setup project", "Create wireframes", "Code
 review", "Testing phase", "Deploy to store"] }
```

程式碼範例

若要檢視使用 $position命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function insertTasksAtPosition() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('projects');

  await collection.updateOne(
    { _id: 1 },
    {
      $push: {
        tasks: {
          $each: ["Security audit", "Performance review"],
          $position: 0
        }
      }
    }
  );
```

```
  const updatedProject = await collection.findOne({ _id: 1 });
  console.log(updatedProject);

  await client.close();
}

insertTasksAtPosition();
```

Python

```
from pymongo import MongoClient

def insert_tasks_at_position():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['projects']

    result = collection.update_one(
        {'_id': 1},
        {
            '$push': {
                'tasks': {
                    '$each': ['Security audit', 'Performance review'],
                    '$position': 0
                }
            }
        }
    )

    updated_project = collection.find_one({'_id': 1})
    print(updated_project)

    client.close()

insert_tasks_at_position()
```

$pull

$pull 運算子用於從陣列中移除符合指定條件之值的所有執行個體。當您需要從文件中的陣列欄位中移除特定元素時，此運算子非常有用。

參數

- `field`：要從中移除值的陣列欄位名稱 (s)。
- `value`：決定要從陣列中移除哪個元素的值或條件 (s)。

範例 (MongoDB Shell)

下列範例示範如何使用 $pull運算子從陣列欄位移除元素。

建立範例文件

```
db.restaurants.insertMany([
  {
    name: "Pizza Hut",
    cuisine: "Italian",
    features: ["Delivery", "Takeout", "Dine-in"]
  },
  {
    name: "Sushi Saito",
    cuisine: "Japanese",
    features: ["Dine-in", "Private Dining"]
  },
  {
    name: "Taco Bell",
    cuisine: "Mexican",
    features: ["Delivery", "Takeout", "Drive-thru"]
  }
])
```

查詢範例

```
db.restaurants.updateMany(
  { cuisine: "Italian" },
  { $pull: { features: "Takeout" } }
)
```

輸出

```
{
  "acknowledged" : true,
```

```
  "matchedCount" : 1,
  "modifiedCount" : 1
}
```

上述查詢會從cuisine欄位為「義大利文」的所有文件中移除「擷取」功能。

程式碼範例

若要檢視使用 $pull命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function updateRestaurants() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const restaurants = db.collection('restaurants');

  await restaurants.updateMany(
    { cuisine: 'Italian' },
    { $pull: { features: 'Takeout' } }
  );

  const updatedRestaurants = await restaurants.find({}).toArray();
  console.log(updatedRestaurants);

  await client.close();
}

updateRestaurants();
```

Python

```
from pymongo import MongoClient

def update_restaurants():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
```

```
    restaurants = db.restaurants

    result = restaurants.update_many(
        { 'cuisine': 'Italian' },
        { '$pull': { 'features': 'Takeout' } }
    )

    updated_restaurants = list(restaurants.find({}))
    print(updated_restaurants)

    client.close()

update_restaurants()
```

$pullAll

Amazon DocumentDB 中的$pullAll運算子用於從陣列欄位移除指定值的所有執行個體。當您需要在單一操作中移除陣列中的多個元素時，此功能特別有用。

參數

- field：要從中移除元素的陣列欄位名稱。
- value：要從陣列欄位移除的值陣列。

範例 (MongoDB Shell)

下列範例示範如何使用 $pullAll運算子從陣列欄位移除多個元素。

建立範例文件

```
db.restaurants.insert([
  {
    "name": "Taj Mahal",
    "cuisine": "Indian",
    "features": ["Private Dining", "Live Music"]
  },
  {
    "name": "Golden Palace",
    "cuisine": "Chinese",
    "features": ["Private Dining", "Takeout"]
  },
```

```
  {
    "name": "Olive Garden",
    "cuisine": "Italian",
    "features": ["Private Dining", "Outdoor Seating"]
  }
])
```

查詢範例

```
db.restaurants.update(
  { "name": "Taj Mahal" },
  { $pullAll: { "features": ["Private Dining", "Live Music"] } }
)
```

輸出

```
{
  "name": "Taj Mahal",
  "cuisine": "Indian",
  "features": []
}
```

程式碼範例

若要檢視使用 $pullAll命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('restaurants');

  await collection.updateMany(
    { "name": "Taj Mahal" },
    { $pullAll: { "features": ["Private Dining", "Live Music"] } }
  );
```

```
  const updatedDocument = await collection.findOne({ "name": "Taj Mahal" });
  console.log(updatedDocument);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['restaurants']

    collection.update_many(
        {"name": "Taj Mahal"},
        {"$pullAll": {"features": ["Private Dining", "Live Music"]}}
    )

    updated_document = collection.find_one({"name": "Taj Mahal"})
    print(updated_document)

    client.close()

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

$push

Amazon DocumentDB 中的$push運算子用於將項目新增至文件中的陣列欄位。當您需要將新資料附加至現有陣列而不覆寫整個陣列時，此運算子特別有用。

參數

- `field`：應新增新元素的陣列欄位名稱。
- `value`：要新增至陣列的值。

- position：（選用）修飾詞，指定在陣列中應新增新元素的位置。支援的修飾詞包括 $（新增至陣列結尾）和 $[]（新增至陣列結尾，忽略任何陣列篩選條件）。

範例 (MongoDB Shell)

下列範例示範如何使用 $push運算子將新元素新增至文件中的陣列欄位。

建立範例文件

```
db.users.insert([
  { _id: 1, name: "John Doe", hobbies: ["reading", "swimming"] },
  { _id: 2, name: "Jane Smith", hobbies: ["gardening", "cooking"] }
])
```

查詢範例

```
db.users.updateOne(
  { _id: 1 },
  { $push: { hobbies: "hiking" } }
)
```

輸出

```
{
  "acknowledged" : true,
  "matchedCount" : 1,
  "modifiedCount" : 1
}
```

執行更新後，具有 的文件_id: 1會將hobbies陣列更新為 ["reading", "swimming", "hiking"]。

程式碼範例

若要檢視使用 $push命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function updateDocument() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('users');

  const result = await collection.updateOne(
    { _id: 1 },
    { $push: { hobbies: "hiking" } }
  );

  console.log(result);
  client.close();
}

updateDocument();
```

Python

```
from pymongo import MongoClient

def update_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['users']

    result = collection.update_one(
        {'_id': 1},
        {'$push': {'hobbies': 'hiking'}}
    )

    print(result.raw_result)
    client.close()

update_document()
```

$rename

Amazon DocumentDB 中的$rename運算子用於重新命名文件中的欄位。當您需要更新文件結構或使其與新的資料模型保持一致時，此運算子特別有用。

參數

- field：要重新命名的欄位。
- newName： 欄位的新名稱。

範例 (MongoDB Shell)

下列範例示範如何使用 $rename運算子，在將 "Date.DoW" 欄位設為的"Date.DayOfWeek"文件中將 "DocName" 欄位重新命名為 "Document 1"。

建立範例文件

```
db.example.insertOne({
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DoW": "Saturday"
    },
    "Words": 2482
})
```

查詢範例

```
db.example.update(
    { "DocName": "Document 1" },
    { $rename: { "Date.DoW": "Date.DayOfWeek" } }
)
```

輸出

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DayOfWeek": "Saturday"
```

```
    },
    "Words": 2482
}
```

程式碼範例

若要檢視使用 $rename命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
    const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
    const db = client.db('test');
    const collection = db.collection('example');

    await collection.updateOne(
        { "DocName": "Document 1" },
        { $rename: { "Date.DoW": "Date.DayOfWeek" } }
    );

    const updatedDoc = await collection.findOne({ "DocName": "Document 1" });
    console.log(updatedDoc);

    await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['example']
```

```
    collection.update_one(
        {"DocName": "Document 1"},
        {"$rename": {"Date.DoW": "Date.DayOfWeek"}}
    )

    updated_doc = collection.find_one({"DocName": "Document 1"})
    print(updated_doc)

    client.close()

example()
```

$set

Amazon DocumentDB 中的$set運算子用於更新文件中指定欄位的值。此運算子可讓您新增欄位或修改文件中現有的欄位。它是 MongoDB Java 驅動程式中的基本更新運算子，與 Amazon DocumentDB 相容。

參數

- field：要更新的欄位。
- value： 欄位的新值。

範例 (MongoDB Shell)

下列範例示範如何使用 $set運算子更新文件中Item的欄位。

建立範例文件

```
db.example.insert([
  {
    "Item": "Pen",
    "Colors": ["Red", "Green", "Blue", "Black"],
    "Inventory": {
      "OnHand": 244,
      "MinOnHand": 72
    }
  },
  {
    "Item": "Poster Paint",
    "Colors": ["Red", "Green", "Blue", "White"],
```

```
    "Inventory": {
      "OnHand": 120,
      "MinOnHand": 36
    }
  }
])
```

查詢範例

```
db.example.update(
  { "Item": "Pen" },
  { $set: { "Item": "Gel Pen" } }
)
```

輸出

```
{
  "Item": "Gel Pen",
  "Colors": ["Red", "Green", "Blue", "Black"],
  "Inventory": {
    "OnHand": 244,
    "MinOnHand": 72
  }
}
```

程式碼範例

若要檢視使用 $set命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function updateDocument() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('example');

  await collection.updateOne(
    { "Item": "Pen" },
```

```
    { $set: { "Item": "Gel Pen" } }
  );

  const updatedDocument = await collection.findOne({ "Item": "Gel Pen" });
  console.log(updatedDocument);

  await client.close();
}

updateDocument();
```

Python

```
from pymongo import MongoClient

def update_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.example

    collection.update_one(
        {"Item": "Pen"},
        {"$set": {"Item": "Gel Pen"}}
    )

    updated_document = collection.find_one({"Item": "Gel Pen"})
    print(updated_document)

    client.close()

update_document()
```

$setOnInsert

如果正在插入文件，Amazon DocumentDB 中的`$setOnInsert`運算子會用來設定欄位的值，但如果正在更新文件，則不會有任何影響。

參數

- `field`：要設定的欄位。

- value：要指派給欄位的值。

範例 (MongoDB Shell)

下列範例示範 Amazon DocumentDB 中$setOnInsert運算子的使用情況。如果文件尚未存在，它會建立新的文件，但如果文件正在更新，則不會有任何影響。

建立範例文件

```
db.users.insertOne({
  _id: 1,
  name: "John Doe",
  age: 30
})
```

查詢範例 1 - 更新現有文件

```
db.users.update(
  { _id: 1 },
  {
    $set: { age: 31 },
    $setOnInsert: { createdAt: new Date() }
  },
  { upsert: true }
)
```

輸出 1

```
{
  _id: 1,
  name: "John Doe",
  age: 31
}
```

輸出顯示文件已更新，但由於文件已存在，因此未新增 createdAt 欄位。運算$setOnInsert子僅適用於插入新文件時。

查詢範例 2 - 插入新文件 (upsert)

```
db.users.update(
```

```
  { _id: 2 },
  {
    $set: { name: "Jane Smith", age: 25 },
    $setOnInsert: { createdAt: new Date() }
  },
  { upsert: true }
)
```

輸出 2

```
{
  _id: 2,
  name: "Jane Smith",
  age: 25,
  createdAt: ISODate("2025-10-31T09:57:52.459Z")
}
}
```

程式碼範例

若要檢視使用 $setOnInsert命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function updateWithSetOnInsert() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  await users.updateOne(
    { _id: 1 },
    {
      $set: { age: 31 },
      $setOnInsert: { createdAt: new Date() }
    },
    { upsert: true }
  );

  const updatedUser = await users.findOne({ _id: 1 });
```

```
  console.log(updatedUser);

  await client.close();
}

updateWithSetOnInsert();
```

Python

```
from pymongo import MongoClient

def update_with_set_on_insert():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    users = db['users']

    result = users.update_one(
        {'_id': 1},
        {
            '$set': {'age': 31},
            '$setOnInsert': {'createdAt': datetime.datetime.now()}
        },
        upsert=True
    )

    updated_user = users.find_one({'_id': 1})
    print(updated_user)

    client.close()

update_with_set_on_insert()
```

$slice

$slice 更新運算子透過限制陣列的大小來修改陣列。與$push運算子搭配使用時，它會限制陣列中的元素數量，只保留指定數量的最新或最舊的元素。

參數

- field：要修改的陣列欄位。

- count：要保留的元素數目上限。正值保留前 N 個元素，負值保留最後 N 個元素。

範例 (MongoDB Shell)

下列範例示範如何使用$slice更新運算子搭配 $push來維護最近分數的固定大小陣列。

建立範例文件

```
db.students.insertOne({
  _id: 1,
  name: "Alice",
  scores: [85, 90, 78]
});
```

查詢範例

```
db.students.updateOne(
  { _id: 1 },
  {
    $push: {
      scores: {
        $each: [92, 88],
        $slice: -3
      }
    }
  }
)
```

輸出

```
{
  "_id" : 1,
  "name" : "Alice",
  "scores" : [ 78, 92, 88 ]
}
```

在此範例中，$slice: -3修改器只會在將新值推送至陣列後保留最後三個元素。

程式碼範例

若要檢視使用$slice更新運算子的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function updateDocument() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('students');

  await collection.updateOne(
    { _id: 1 },
    {
      $push: {
        scores: {
          $each: [92, 88],
          $slice: -3
        }
      }
    }
  );

  const updatedDocument = await collection.findOne({ _id: 1 });
  console.log(updatedDocument);

  await client.close();
}

updateDocument();
```

Python

```
from pymongo import MongoClient

def update_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.students

    collection.update_one(
```

```
        {'_id': 1},
        {
            '$push': {
                'scores': {
                    '$each': [92, 88],
                    '$slice': -3
                }
            }
        }
    )

    updated_document = collection.find_one({'_id': 1})
    print(updated_document)

    client.close()

update_document()
```

$sort

與$push運算子搭配使用時，$sort更新修飾詞會訂購陣列元素。它會根據指定的欄位值或元素本身，以遞增或遞減順序排列陣列元素。

參數

- field：要修改的陣列欄位。
- order：1用於遞增順序或用於-1遞減順序。

範例 (MongoDB Shell)

下列範例示範搭配 $sort 使用修飾詞$push來新增新的測驗分數，並依遞減順序進行排序。

建立範例文件

```
db.students.insertOne({
  _id: 1,
  name: "Bob",
  quizzes: [
    { score: 85, date: "2024-01-15" },
    { score: 92, date: "2024-02-10" }
  ]
```

```
});
```

查詢範例

```
db.students.updateOne(
  { _id: 1 },
  {
    $push: {
      quizzes: {
        $each: [{ score: 78, date: "2024-03-05" }],
        $sort: { score: -1 }
      }
    }
  }
)
```

輸出

```
{
  "_id" : 1,
  "name" : "Bob",
  "quizzes" : [
    { "score" : 92, "date" : "2024-02-10" },
    { "score" : 85, "date" : "2024-01-15" },
    { "score" : 78, "date" : "2024-03-05" }
  ]
}
```

程式碼範例

若要檢視使用$sort更新修飾詞的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function updateDocument() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('students');
```

```
  await collection.updateOne(
    { _id: 1 },
    {
      $push: {
        quizzes: {
          $each: [{ score: 78, date: "2024-03-05" }],
          $sort: { score: -1 }
        }
      }
    }
  );

  const updatedDocument = await collection.findOne({ _id: 1 });
  console.log(updatedDocument);

  await client.close();
}

updateDocument();
```

Python

```
from pymongo import MongoClient

def update_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.students

    collection.update_one(
        {'_id': 1},
        {
            '$push': {
                'quizzes': {
                    '$each': [{'score': 78, 'date': '2024-03-05'}],
                    '$sort': {'score': -1}
                }
            }
        }
    )
```

```
    updated_document = collection.find_one({'_id': 1})
    print(updated_document)

    client.close()

update_document()
```

$unset

Amazon DocumentDB 中的$unset運算子用於從文件中移除指定的欄位。使用 移除欄位時$unset，會從文件中刪除欄位，並相應地減少文件大小。當您想要從文件中移除不必要的資料時，這會很有用。

參數

- `field`：要從文件中移除的欄位。這可以是單一欄位或巢狀欄位的虛線路徑。

範例 (MongoDB Shell)

下列範例示範如何使用 $unset運算子，從example集合中的文件移除 Words 欄位。

建立範例文件

```
db.example.insert({
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DoW": "Saturday"
    },
    "Words": 2482
})
```

查詢範例

```
db.example.update(
    { "DocName" : "Document 1" },
    { $unset: { Words:1 } }
)
```

輸出

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DoW": "Saturday"
    }
}
```

在此範例中，運算$unset子用於從DocName等於「文件 1」的文件中移除 Words 欄位。產生的文件不再包含 Words 欄位。

程式碼範例

若要檢視使用 $unset命令的程式碼範例，請選擇您要使用的語言標籤：

Node.js

```
const { MongoClient } = require('mongodb');

async function removeField() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('example');

  const result = await collection.updateOne(
    { "DocName": "Document 1" },
    { $unset: { "Words": 1 } }
  );

  console.log(`Modified ${result.modifiedCount} document(s)`);
  client.close();
}

removeField();
```

Python

```
from pymongo import MongoClient

def remove_field():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['example']

    result = collection.update_one(
        {"DocName": "Document 1"},
        {"$unset": {"Words": 1}}
    )

    print(f"Modified {result.modified_count} document(s)")
    client.close()

remove_field()
```

管理 Amazon DocumentDB 叢集和執行個體

這些章節涵蓋各種元件及其管理 Amazon DocumentDB （與 MongoDB 相容） 實作的相關任務。

主題

- Amazon DocumentDB 操作任務概觀
- Amazon DocumentDB 全域叢集概觀
- 管理 Amazon DocumentDB 叢集
- 管理 Amazon DocumentDB 執行個體
- 管理 Amazon DocumentDB 子網路群組
- Amazon DocumentDB 高可用性和複寫
- 管理 Amazon DocumentDB 事件
- 管理 Amazon DocumentDB 叢集參數群組
- 了解 Amazon DocumentDB 端點
- 了解 Amazon DocumentDB Amazon Resource Name (ARNs)
- 標記 Amazon DocumentDB 資源
- 維護 Amazon DocumentDB
- 了解服務連結角色

Amazon DocumentDB 操作任務概觀

本節涵蓋 Amazon DocumentDB 叢集的操作任務，以及如何使用 完成這些任務 AWS CLI。

主題

- 將複本新增至 Amazon DocumentDB 叢集
- 描述叢集和執行個體
- 建立叢集快照
- 從快照還原
- 從叢集移除執行個體
- 刪除叢集

將複本新增至 Amazon DocumentDB 叢集

為 Amazon DocumentDB 叢集建立主要執行個體之後，您可以新增一或多個複本。複本是唯讀的執行個體，有兩種用途：

- 可擴展性 — 如果您有大量用戶端需要並行存取，您可以新增更多複本以進行讀取擴展。
- 高可用性 — 如果主要執行個體失敗，Amazon DocumentDB 會自動容錯移轉至複本執行個體，並將其指定為新的主要執行個體。若複本失敗，叢集中的其他執行個體仍然可以處理請求，直到可復原失敗的節點為止。

每個 Amazon DocumentDB 叢集最多可支援 15 個複本。

> **Note**
>
> 為取得最大的容錯能力，建議您將複本部署在分離的可用區域中。這有助於確保您的 Amazon DocumentDB 叢集可以繼續運作，即使整個可用區域無法使用。

下列 AWS CLI 範例示範如何新增複本。`--availability-zone` 參數會取代所指定可用區域中的複本。

```
aws docdb create-db-instance \
    --db-instance-identifier sample-instance \
    --db-cluster-identifier sample-cluster \
    --engine docdb \
    --db-instance-class db.r5.large \
    --availability-zone us-east-1a
```

描述叢集和執行個體

下列 AWS CLI 範例列出區域中的所有 Amazon DocumentDB 叢集。對於特定管理功能，例如叢集和執行個體生命週期管理，Amazon DocumentDB 會利用與 Amazon RDS 共用的操作技術。`filterName=engine,Values=docdb` 篩選條件參數只會傳回 Amazon DocumentDB 叢集。

如需描述和修改叢集的詳細資訊，請參閱Amazon DocumentDB 叢集生命週期。

```
aws docdb describe-db-clusters --filter Name=engine,Values=docdb
```

此操作的輸出將會如下所示。

```
{
    "DBClusters": [
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-1",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        },
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-2",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        },
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-3",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        }
    ]
```

```
}
```

下列 AWS CLI 範例列出 Amazon DocumentDB 叢集中的執行個體。如需描述和修改叢集的詳細資訊，請參閱Amazon DocumentDB 執行個體生命週期。

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterMembers]'
```

輸出類似如下所示。這個輸出中有兩個執行個體。主要執行個體為 sample-instance-1 ("IsClusterWriter": true)。另外還有複本執行個體 sample-instance2 ("IsClusterWriter: false")。

```
[
    [
        [
            {
                "DBInstanceIdentifier": "sample-instance-1",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            },
            {
                "DBInstanceIdentifier": "sample-cluster-2",
                "IsClusterWriter": false,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            }
        ]
    ]
]
```

建立叢集快照

叢集快照是 Amazon DocumentDB 叢集中資料的完整備份。建立快照時，Amazon DocumentDB 會直接從叢集磁碟區讀取您的資料。因此，即使叢集當時並未執行任何執行個體，您仍可以建立快照。建立快照所需的時間量取決於叢集磁碟區的大小。

Amazon DocumentDB 支援自動備份，每天在偏好的備份時段進行，一天中有 30 分鐘的時間。下列 AWS CLI 範例顯示如何檢視叢集的備份時段：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].PreferredBackupWindow'
```

輸出會顯示備份時段 (UTC)：

```
[
    "00:18-00:48"
]
```

您可以在建立 Amazon DocumentDB 叢集時定義備份時段。您也可以變更備份時段，如下列範例所示。如果您未定義備份時段，Amazon DocumentDB 會自動將備份時段指派給您的叢集。

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --preferred-backup-window "02:00-02:30"
```

除了自動備份之外，您可以隨時手動建立叢集快照。當您執行此操作時，您需要指定要備份的叢集，以及快照的唯一名稱，以便之後能夠還原。

下列 AWS CLI 範例示範如何建立資料的快照。

```
aws docdb create-db-cluster-snapshot \
    --db-cluster-identifier sample-cluster \
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

從快照還原

您可以將叢集快照還原至新的 Amazon DocumentDB 叢集。若要執行這項操作，請提供快照集的名稱和新叢集的名稱。您無法從快照還原至現有叢集；而是 Amazon DocumentDB 會在還原時建立新的叢集，然後填入快照資料。

以下範例顯示叢集 `sample-cluster` 的所有快照，

```
aws docdb describe-db-cluster-snapshots \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusterSnapshots[*].[DBClusterSnapshotIdentifier,SnapshotType,Status]'
```

輸出類似如下所示。手動快照是您手動建立的快照，而 Amazon DocumentDB 會在叢集備份時段內建立自動快照。

```
[
        "sample-cluster-snapshot",
        "manual",
        "available"
    ],
    [
        "rds:sample-cluster",
        "automated",
        "available"
    ]
]
```

下列範例示範如何從快照還原 Amazon DocumentDB 叢集。

```
aws docdb restore-db-cluster-from-snapshot \
    --engine docdb \
    --db-cluster-identifier new-sample-cluster \
    --snapshot-identifier sample-cluster-snapshot
```

新叢集沒有任何相關聯的執行個體；因此，如果您想要與叢集互動，必須在當中新增執行個體。

```
aws docdb create-db-instance \
    --db-instance-identifier new-sample-instance \
    --db-instance-class db.r5.large \
    --engine docdb \
    --db-cluster-identifier new-sample-cluster
```

您可以使用下列 AWS CLI 操作來監控叢集和執行個體建立的進度。當叢集和執行個體狀態都可取得時，您就可以連接至新叢集的端點並存取您的資料。

```
aws docdb describe-db-clusters \
    --db-cluster-identifier new-sample-cluster  \
    --query 'DBClusters[*].[Status,Endpoint]'
```

```
aws docdb describe-db-instances \
    --db-instance-identifier new-sample-instance \
    --query 'DBInstances[*].[DBInstanceStatus]'
```

從叢集移除執行個體

Amazon DocumentDB 會將所有資料存放在叢集磁碟區中。資料會保留在該叢集磁碟區中，即使您移除叢集中的所有執行個體也一樣。如果您需要再次存取資料，可以隨時新增執行個體至叢集，並從您上次離開的位置繼續操作。

下列範例示範如何從 Amazon DocumentDB 叢集中移除執行個體。

```
aws docdb delete-db-instance \
    --db-instance-identifier sample-instance
```

刪除叢集

您必須先移除其所有執行個體，才能刪除 Amazon DocumentDB 叢集。下列 AWS CLI 範例會傳回叢集中執行個體的相關資訊。如果此操作傳回任何執行個體識別碼，則您必須刪除每個執行個體。如需詳細資訊，請參閱從叢集移除執行個體。

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterMembers[*].DBInstanceIdentifier'
```

當沒有剩餘的執行個體時，您就可以刪除叢集。同時，您必須選擇以下其中一個選項：

- 建立最終快照 — 擷取快照中的所有叢集資料，以便稍後可以使用該資料重新建立新的執行個體。下列範例示範其做法：

  ```
  aws docdb delete-db-cluster \
      --db-cluster-identifier sample-cluster \
      --final-db-snapshot-identifier sample-cluster-snapshot
  ```

- 略過最終快照 — 永久捨棄所有叢集資料。這個動作無法還原。下列範例示範其做法：

  ```
  aws docdb delete-db-cluster \
      --db-cluster-identifier sample-cluster \
      --skip-final-snapshot
  ```

Amazon DocumentDB 全域叢集概觀

什麼是全域叢集？

全域叢集包含一個主要區域和最多 10 個唯讀次要區域。您直接對主要區域中的主要叢集發出寫入操作，Amazon DocumentDB 會使用專用基礎設施自動將資料複寫至次要區域。延遲通常不到一秒。

全域叢集有何用處？

- 從全區域中斷復原 – 在全區域中斷的情況下，您可以在幾分鐘內將其中一個次要叢集提升為主要叢集，典型的復原時間目標 (RTO) 不到一分鐘。復原點目標 (RPO) 通常以秒為單位測量，但這取決於失敗時整個網路的延遲。
- 具有本機延遲的全域讀取 - 如果您在世界各地設有辦公室，您可以使用全域叢集，讓主要區域的主要資訊來源保持最新狀態。其他區域的辦公室可以在本地延遲的情況下存取自己區域中的資訊。
- 可擴展的次要叢集 — 您可以將更多唯讀執行個體新增至次要區域，以擴展次要叢集。次要叢集是唯讀的，因此最多可以支援 16 個唯讀複本執行個體，而不是單一叢集的一般限制為 15 個。
- 從主要叢集快速複寫到次要叢集 — 全域叢集執行的複寫對主要資料庫叢集的效能影響不大。資料庫執行個體的資源完全投入處理應用程式讀取與寫入工作負載。
- 變更串流 — 您可以從全域叢集中的次要叢集讀取變更串流，減少主要叢集的負載，並允許您從次要區域擴展讀取工作負載。

全球叢集目前的限制為何？

- Amazon DocumentDB v3.6 不支援全域叢集。
- 除了 db.t3、db.t4g 和 db.r4 之外，所有執行個體類型都支援全域叢集。
- 全球叢集不適用於下列區域：南美洲（聖保羅）、歐洲（米蘭）、中國（北京）和中國（寧夏）。
- 當區域位於不同的引擎版本時，不支援切換和全域容錯移轉。當引擎版本不相符時，支援手動容錯移轉。
- 只有主要叢集會執行寫入操作。執行寫入操作的用戶端會連線至主要叢集的叢集端點。
- 叢集最多可以有 10 個次要區域和一個主要區域。
- 次要叢集無法停止。如果主要叢集具有與其相關聯的次要叢集，則無法停止主要叢集。只有沒有次要叢集的區域叢集才能停止。
- 連接到次要叢集的複本在某些情況下可以重新啟動。如果主要區域的執行個體重新啟動或容錯移轉，次要區域中的複本也會重新啟動。然後，叢集將無法使用，直到所有複本都與主要資料庫叢集的寫入

器執行個體恢復同步。此行為在預期當中。在對主要叢集進行變更之前，請務必了解對全域叢集的影響。

主題

- 快速入門指南：全域叢集
- 管理 Amazon DocumentDB 全域叢集
- 連線至 Amazon DocumentDB 全域叢集
- 監控 Amazon DocumentDB 全域叢集
- 災難復原和 Amazon DocumentDB 全域叢集

快速入門指南：全域叢集

主題

- 組態
- 建立 Amazon DocumentDB 全域叢集
- 將 AWS 區域 新增至 Amazon DocumentDB 全域叢集
- 為您的 Amazon DocumentDB 全域叢集使用快照

組態

Amazon DocumentDB 全域叢集至少跨越兩個 AWS 區域。主要區域支援具有一個主要（寫入器）執行個體和最多 15 個複本執行個體的叢集，而次要區域則執行完全由最多 16 個複本執行個體組成的唯讀叢集。全域叢集最多可以有 10 個次要區域。資料表列出全域叢集中允許的最大叢集、執行個體和複本。

說明	主要 AWS 區域	次要 AWS 區域
叢集	1	10 個 (上限)
寫入器執行個體	1	0
每個叢集的唯讀執行個體 (Amazon DocumentDB 複本)	15 (上限)	16 (總計)

說明	主要 AWS 區域	次要 AWS 區域
唯讀執行個體 (允許的上限，給定的次要區域實際數量)	15 - s	s = 次要的總數 AWS 區域

叢集有下列特定需求：

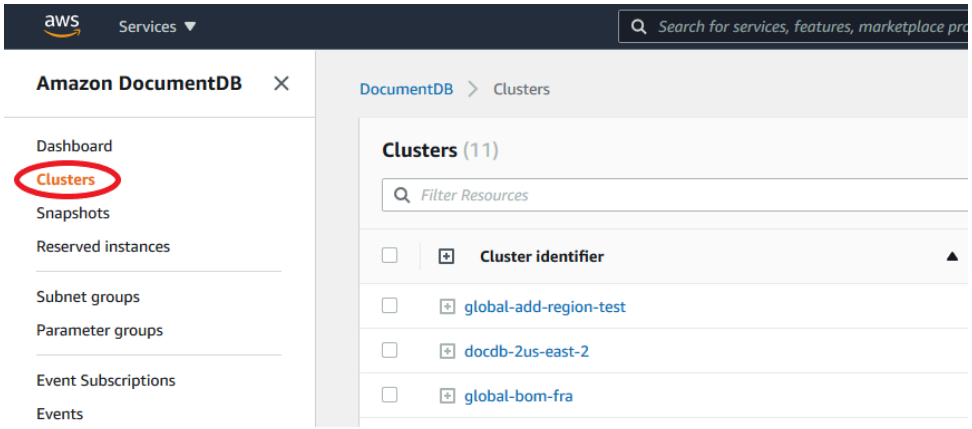
- 資料庫執行個體類別需求 — 您只能使用 `db.r5`和 `db.r6g` 執行個體類別。
- AWS 區域 需求 — 主要叢集必須位於一個區域，且至少有一個次要叢集必須位於相同帳戶的不同區域。您最多可以建立 10 個次要 （唯讀） 叢集，每個叢集必須位於不同的區域。換句話說，兩個叢集不能位於相同的區域。
- 命名要求 — 您為每個叢集選擇的名稱在所有區域中必須是唯一的。您無法對不同的叢集使用相同的名稱，即使它們位於不同的 區域。

建立 Amazon DocumentDB 全域叢集

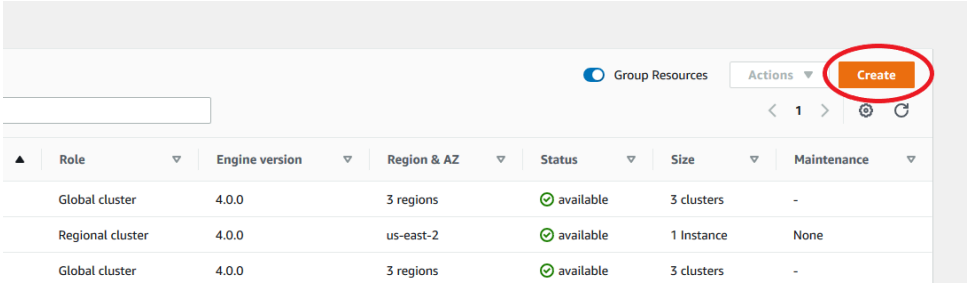
您準備好建置您的第一個全域叢集了嗎？ 在本節中，我們將說明如何使用 AWS 管理主控台 或 AWS CLI 搭配下列指示，建立具有新資料庫叢集和執行個體的全新全域叢集。

使用 AWS 管理主控台

1. 在 中 AWS 管理主控台，導覽至 Amazon DocumentDB。
2. 當您到達 Amazon DocumentDB 主控台時，請選擇叢集。

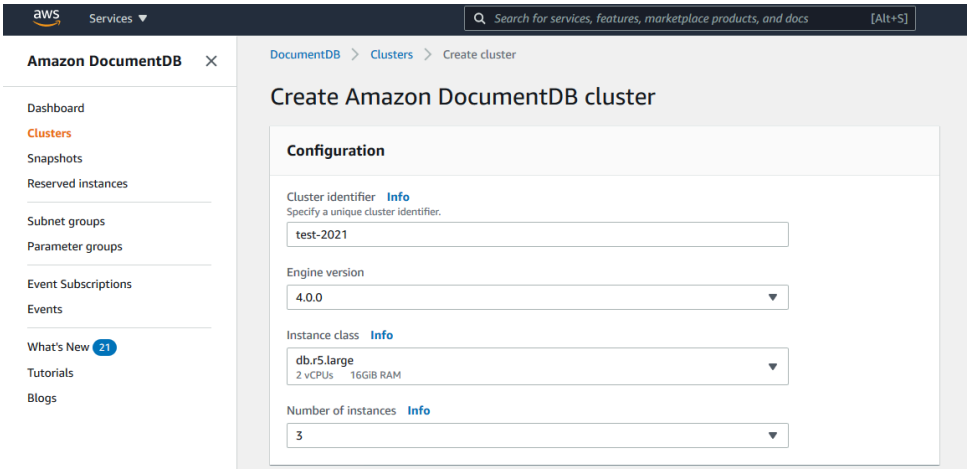


3. 選擇建立。

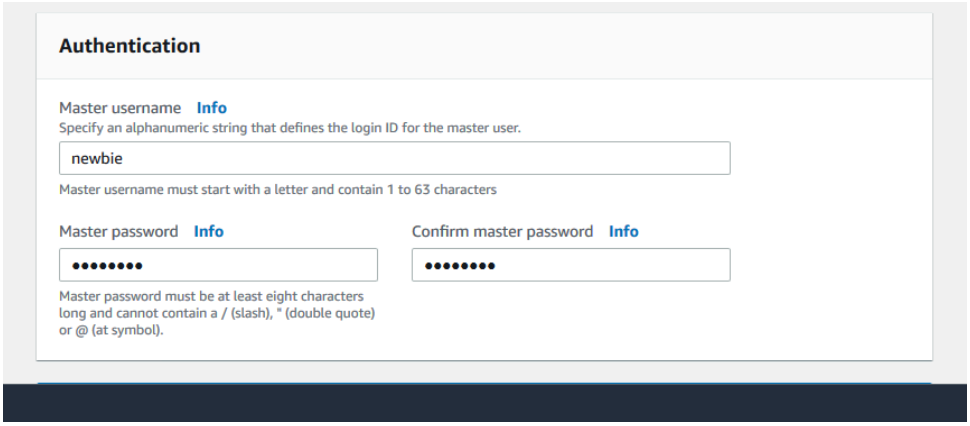


4. 相應地填寫建立 Amazon DocumentDB 叢集表單的組態區段：

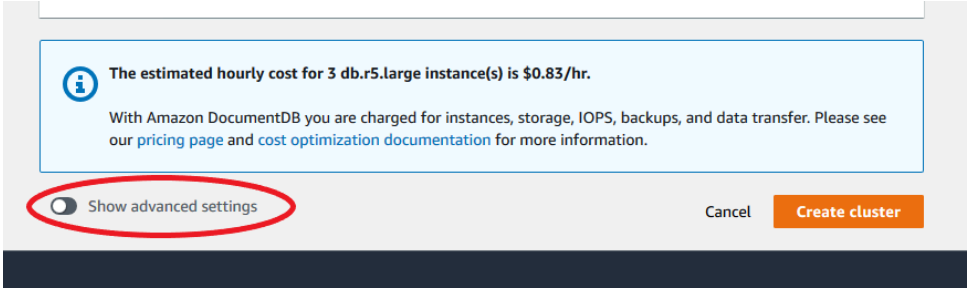
   - 叢集識別符：您可以為此執行個體輸入唯一識別符，或允許 Amazon DocumentDB 根據叢集識別符提供執行個體識別符。
   - 引擎版本：選擇 4.0.0
   - 執行個體類別：選擇 db.r5.large
   - 執行個體數量：選擇 3。



5. 在身分驗證區段中，填寫主要使用者名稱和主要密碼。

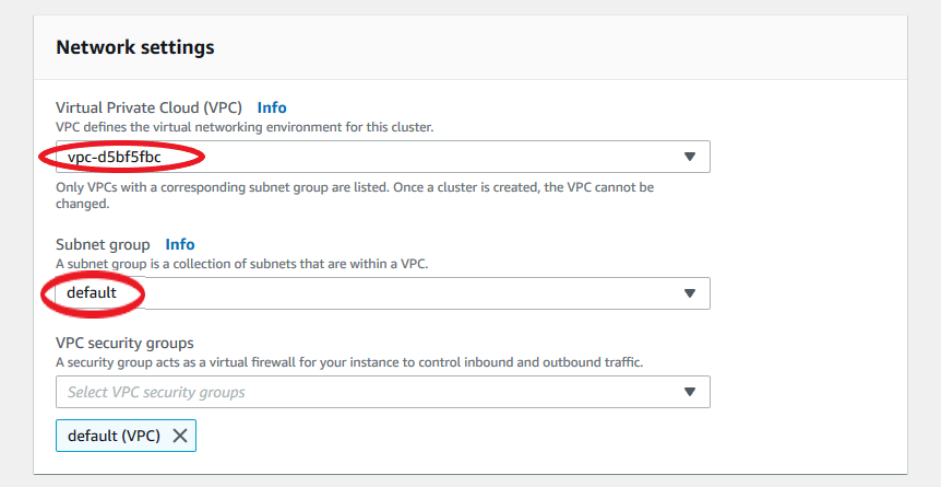


6. 選擇顯示進階設定。

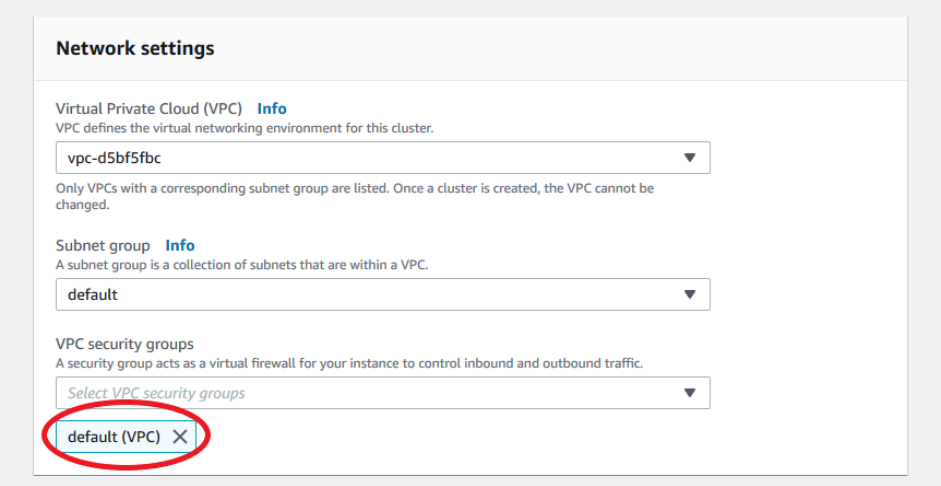


7. 在網路設定區段中：

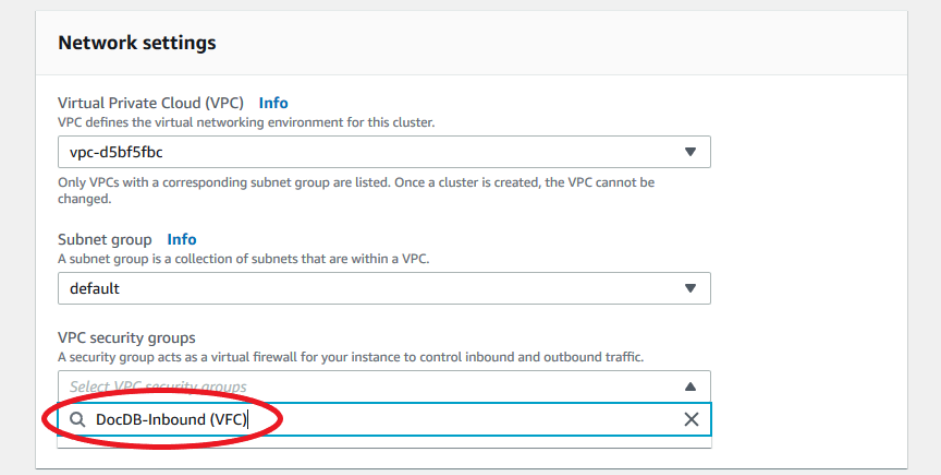
   - 保留虛擬私有雲端 (VPC) 和子網路群組的預設選項。



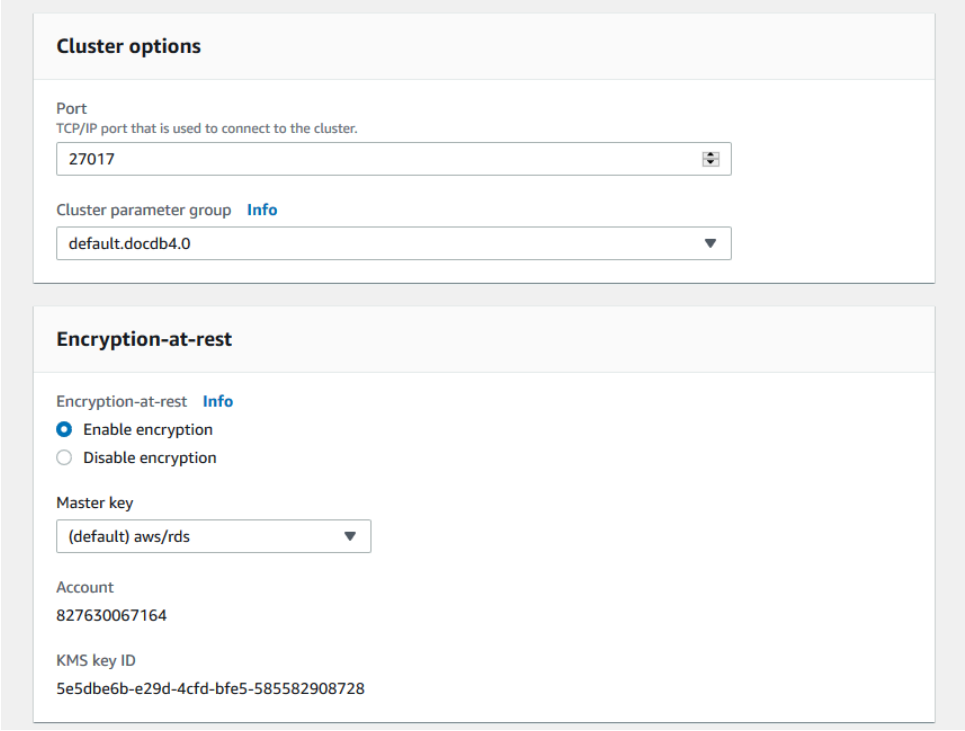
   - 對於 VPC 安全群組，應該已新增預設 (VPC)。



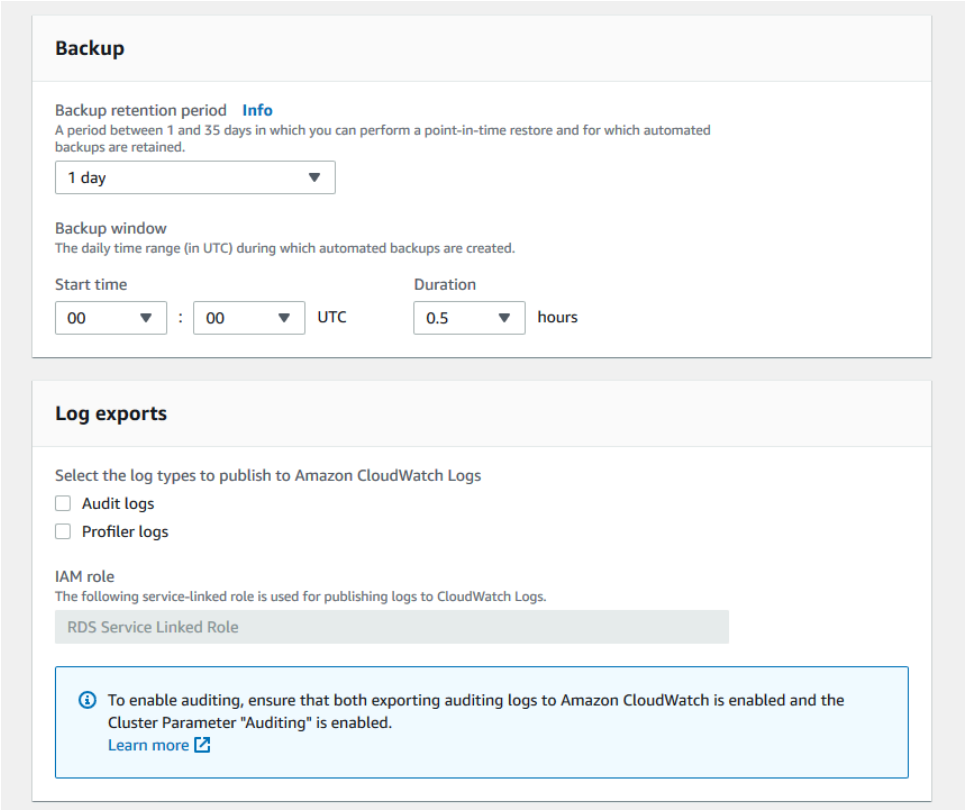
   - DocDB 在 VPC 安全群組欄位中輸入 ，然後選取 DocDB-Inbound (VFC)。



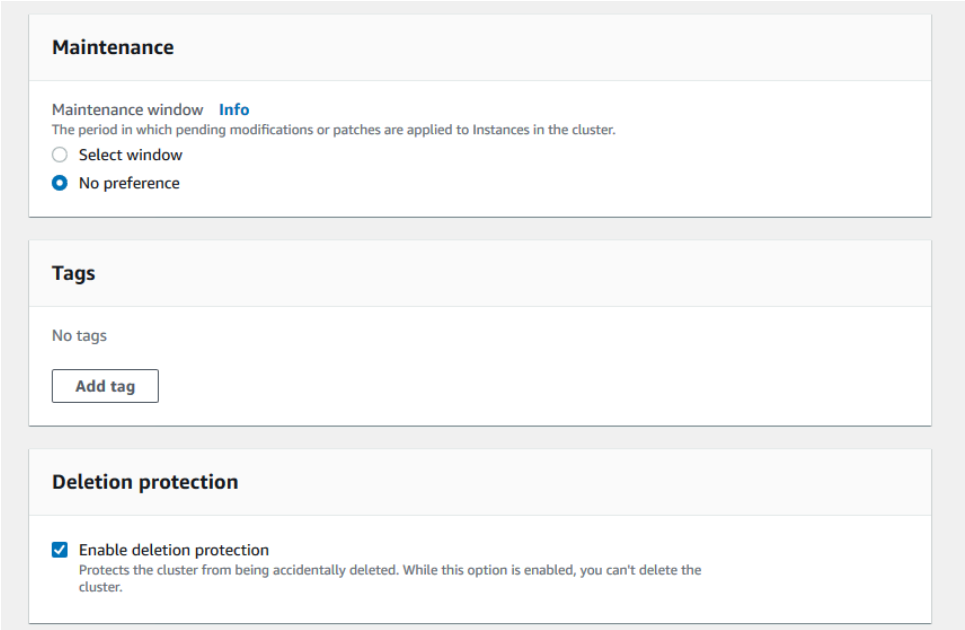
8. 對於叢集選項和Encryption-at-rest，請保留預設選項。



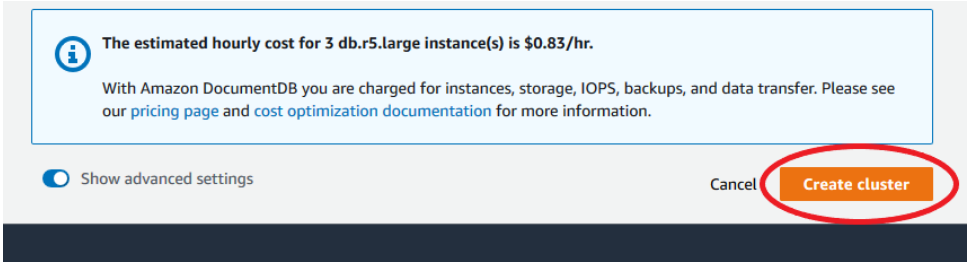
9. 對於備份和日誌匯出，保留預設選項。



10. 對於維護、標籤和刪除保護，保留預設選項。



11. 現在選擇建立叢集。



使用 AWS CLI

若要建立 Amazon DocumentDB 區域叢集，請呼叫 create-global-cluster AWS CLI。下列 AWS CLI 命令會建立名為 的 Amazon DocumentDB 叢集`global-cluster-id`。如需刪除保護的詳細資訊，請參閱 刪除 Amazon DocumentDB 叢集。

此外， `--engine-version` 是選用參數，預設為最新的主要引擎版本。目前的預設引擎版本為 `5.0.0` (注意：Amazon DocumentDB 8.0 可供使用，但必須明確指定為 `8.0.0`)。發行新的主要引擎版本時， 的預設引擎版本`--engine-version`會更新以反映最後一個主要引擎版本。因此，對於生產工作負載，特別是依賴指令碼、自動化或 CloudFormation 範本的工作負載，請將 明確指定`--engine-version`為預期的主要版本。

如果`vpc-security-group-id`未指定 `db-subnet-group-name`或 ，Amazon DocumentDB 將為指定區域使用預設子網路群組和 Amazon VPC 安全群組。

在下列範例中，將每個`#########`取代為您自己的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb create-db-cluster \
      --global-cluster-identifier global-cluster-id \
      --source-db-cluster-identifier arn:aws:rds:us-east-1:111122223333:cluster-id
```

針對 Windows：

```
aws docdb create-db-cluster ^
      --global-cluster-identifier global-cluster-id ^
      --source-db-cluster-identifier arn:aws:rds:us-east-1:111122223333:cluster-id
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "DBCluster": {
        "StorageEncrypted": false,
        "DBClusterMembers": [],
        "Engine": "docdb",
        "DeletionProtection" : "enabled",
        "ClusterCreateTime": "2018-11-26T17:15:19.885Z",
        "DBSubnetGroup": "default",
        "EngineVersion": "4.0.0",
        "MasterUsername": "masteruser",
        "BackupRetentionPeriod": 1,
        "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:cluster-id",
        "DBClusterIdentifier": "cluster-id",
        "MultiAZ": false,
        "DBClusterParameterGroup": "default.docdb4.0",
        "PreferredBackupWindow": "09:12-09:42",
        "DbClusterResourceId": "cluster-KQSGI4MHU4NTDDRVNLNTU7XVAY",
        "PreferredMaintenanceWindow": "tue:04:17-tue:04:47",
        "Port": 27017,
        "Status": "creating",
        "ReaderEndpoint": "cluster-id.cluster-ro-sfcrlcjcoroz.us-
east-1.docdb.amazonaws.com",
        "AssociatedRoles": [],
        "HostedZoneId": "ZNKXTT8WH85VW",
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-77186e0d",
                "Status": "active"
            }
        ],
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1c",
            "us-east-1e"
        ],
        "Endpoint": "cluster-id.cluster-sfcrlcjcoroz.us-east-1.docdb.amazonaws.com"
    }
}
```


建立叢集需要幾分鐘才能完成。您可以使用 AWS 管理主控台 或 AWS CLI 來監控叢集的狀態。如需詳細資訊，請參閱監控 Amazon DocumentDB 叢集的狀態。

> **Important**
>
> 當您使用 AWS CLI 建立 Amazon DocumentDB 區域叢集時，不會建立任何執行個體。因此，您必須明確建立主要執行個體，以及您需要的任何複本執行個體。您可以使用 主控台或 AWS CLI 來建立執行個體。如需詳細資訊，請參閱《Amazon DocumentDB API 參考》中的 將 Amazon DocumentDB 執行個體新增至叢集和 CreateDBCluster。

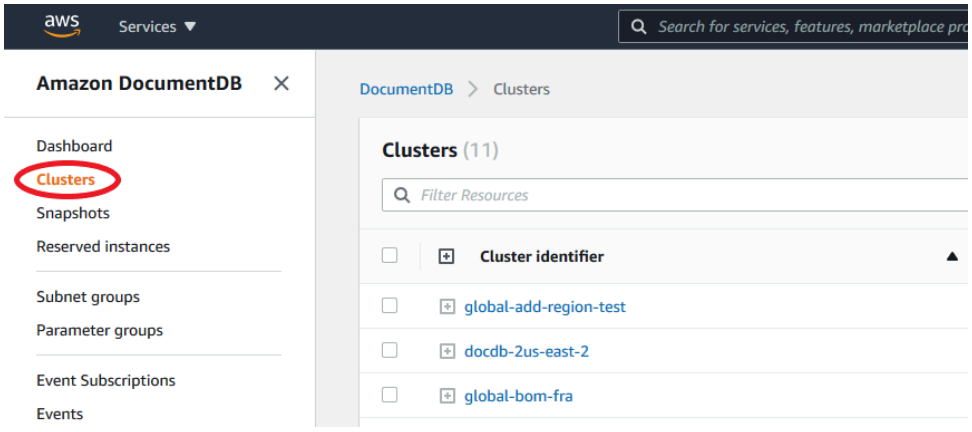
區域叢集可用後，您可以使用下列指示在另一個區域中新增次要叢集：將 AWS 區域 新增至 Amazon DocumentDB 全域叢集。當您新增區域時，您的區域叢集會成為您的主要叢集，並在您選擇的區域中擁有新的次要叢集。

將 AWS 區域 新增至 Amazon DocumentDB 全域叢集

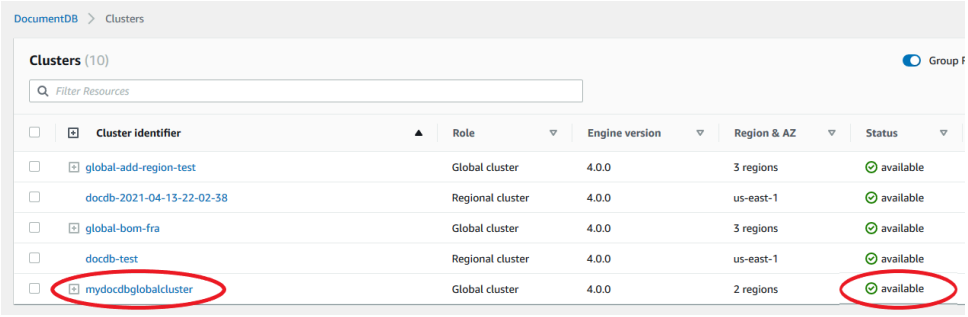
全域叢集需要在與主要叢集不同的區域中至少有一個次要叢集，而且您最多可以新增 10 個次要叢集。請注意，對於您新增的每個次要叢集，您必須將主要叢集中允許的複本數量減少一個。例如，如果您的全域叢集有 10 個次要區域，您的主要叢集只能有 5 個（而不是 15 個）複本。如需詳細資訊，請參閱組態。

使用 AWS 管理主控台

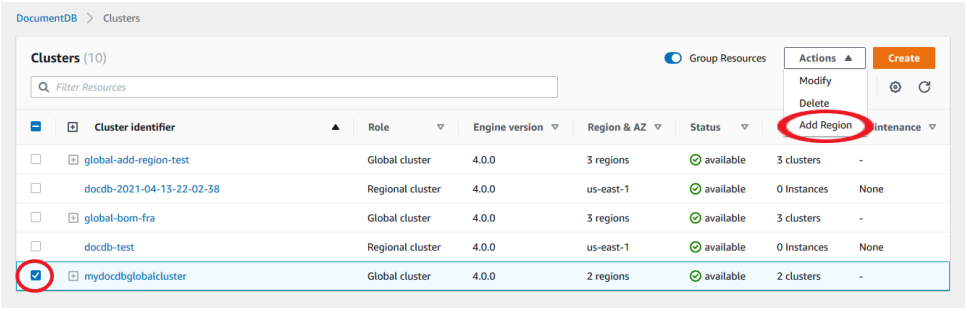
1. 登入 AWS 管理主控台 並開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。



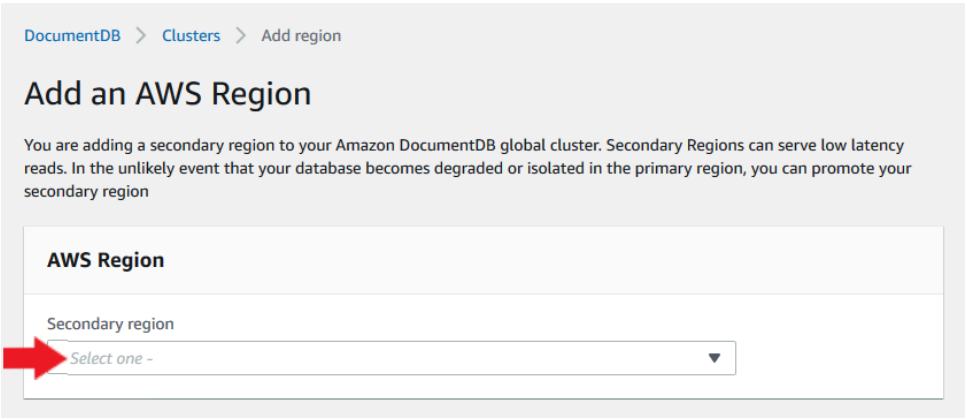
3. 選擇您要新增次要叢集的叢集。確定叢集是 `Available`。



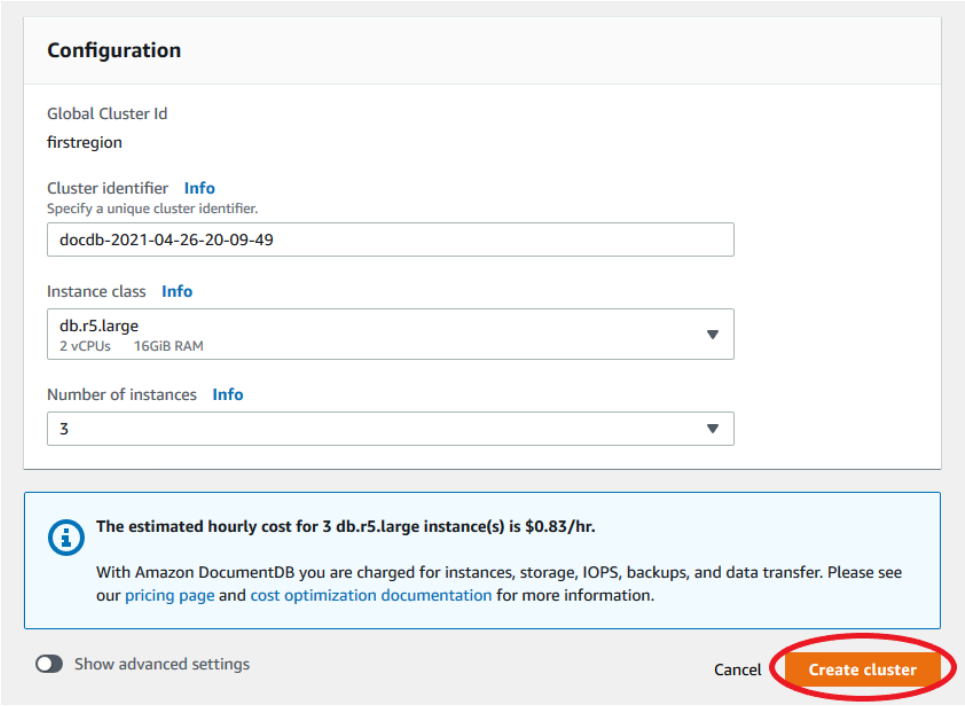
4. 選取動作的下拉式清單，然後選擇新增區域。



5. 在新增頁面上 AWS 區域，選擇次要區域。請注意，您無法為相同的全域叢集選擇已有次要叢集的區域。此外，它不能與主要叢集的區域相同。如果這是您要新增的第一個區域，您也必須指定您選擇的全域叢集識別符。



6. 完成新區域中次要叢集的其餘欄位，然後選取建立叢集。完成新增區域後，您可以在 中的叢集清單中看到它 AWS 管理主控台。



使用 AWS CLI

- 使用 `create-db-cluster` CLI (`--global-cluster-identifier`) 命令搭配全域叢集的名稱。對於其他參數，請執行下列動作：
  - 針對 `--region`，選擇 AWS 區域 與您主要區域不同的 。
  - 請選擇 `--engine` 和 `--engine-version` 參數的特定值。
  - 對於加密的叢集，請將您的主要叢集指定 AWS 區域 為用於加密`--source-region`的 。

下列範例會建立新的 Amazon DocumentDB 叢集，並將其連接至全域叢集做為唯讀次要叢集。在最後一個步驟中，執行個體會新增至新叢集。

在下列範例中，將每個*#########*取代為您自己的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb --region secondary-region-id \
  create-db-cluster \
    --db-cluster-identifier cluster-id \
    --global-cluster-identifier global-cluster-id \
    --engine-version version \
    --engine docdb
```

```
aws docdb --region secondary-region-id \
  create-db-instance \
    --db-cluster-identifier cluster-id \
    --global-cluster-identifier global-cluster-id \
    --engine-version version \
    --engine docdb
```

針對 Windows：

```
aws docdb --region secondary-region-id ^
  create-db-cluster ^
    --db-cluster-identifier cluster-id ^
    --global-cluster-identifier global-cluster-id ^
    --engine-version version ^
    --engine docdb

aws docdb --region secondary-region-id ^
  create-db-instance ^
    --db-cluster-identifier cluster-id ^
    --global-cluster-identifier global-cluster-id ^
    --engine-version version ^
    --engine docdb
```

為您的 Amazon DocumentDB 全域叢集使用快照

您可以還原 Amazon DocumentDB 叢集的快照，以用作全域叢集的起點。若要這樣做，您必須還原快照並建立新的叢集。這將做為您全域叢集的主要叢集。然後，您可以將另一個區域新增至還原的叢集，進而將其轉換為全域叢集。

管理 Amazon DocumentDB 全域叢集

您可以在組成全域叢集的個別叢集上執行大多數管理操作。當您在主控台的叢集頁面上選擇群組相關資源時，您會看到主要叢集和次要叢集在關聯的全域叢集下分組。

全域叢集的組態索引標籤會顯示叢集執行 AWS 區域 所在的 、版本和全域叢集識別符。

主題

- 修改 Amazon DocumentDB 全域叢集

- 修改 Amazon DocumentDB 全域叢集的參數
- 從 Amazon DocumentDB 全域叢集移除叢集
- 從 Amazon DocumentDB 全域叢集刪除叢集
- 在次要區域中建立無周邊 Amazon DocumentDB 叢集

修改 Amazon DocumentDB 全域叢集

中的叢集頁面會 AWS 管理主控台 列出所有全域叢集，顯示每個叢集的主要叢集和次要叢集。全域叢集有自己的組態設定。具體而言，它具有與其主要和次要叢集相關聯的區域。

當您變更全域叢集時，有機會取消變更。

當您選擇 Continue (繼續) 時，您可以確認變更。

修改 Amazon DocumentDB 全域叢集的參數

您可以針對全域叢集中的每個叢集獨立設定叢集參數群組。大多數參數的運作方式與其他類型的 Amazon DocumentDB 叢集相同。如果您將次要叢集提升為主要叢集，請讓全域資料庫中所有叢集的設定保持一致，以避免意外行為變更。

例如，讓時區和字元集使用相同設定，以避免不同叢集接管成為主要業集時發生不一致的行為。

從 Amazon DocumentDB 全域叢集移除叢集

在某些情況下，您可能想要從全域叢集中移除叢集。例如，如果主要叢集降級或隔離，您可能想要從全域叢集中移除叢集。然後，它會成為獨立的佈建叢集，可用於建立新的全域叢集。如需詳細資訊，請參閱 執行 Amazon DocumentDB 全域叢集的手動容錯移轉。

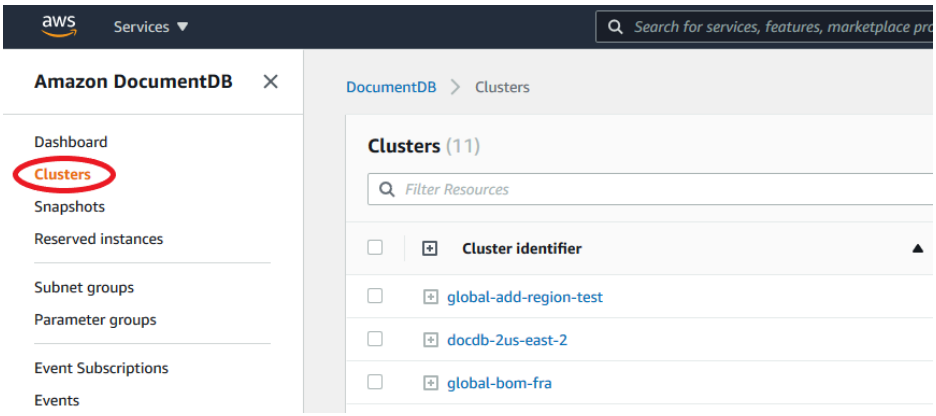
您也可以移除叢集，因為您想要刪除不再需要的全域叢集。在分離所有關聯的叢集之後，才能刪除全域叢集，最後保留主要叢集。如需詳細資訊，請參閱從 Amazon DocumentDB 全域叢集刪除叢集。

> **Note**
>
> 當叢集從全域叢集分離時，不會再與主要叢集同步。它成為具有完整讀取/寫入功能的獨立佈建叢集。此外，它不再顯示在 Amazon DocumentDB 主控台中。只有在您在叢集所在的主控台中選取區域時，才會顯示它。

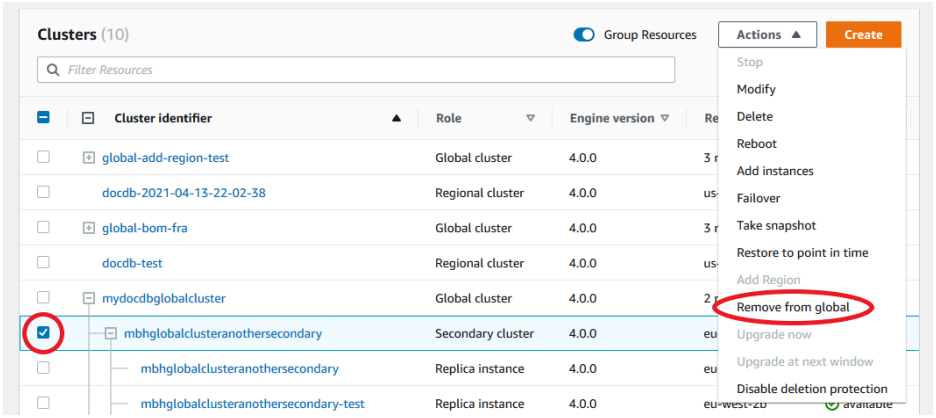
您可以使用 AWS CLI、 AWS 管理主控台或 RDS API 從全域叢集中移除叢集。

Using the AWS 管理主控台

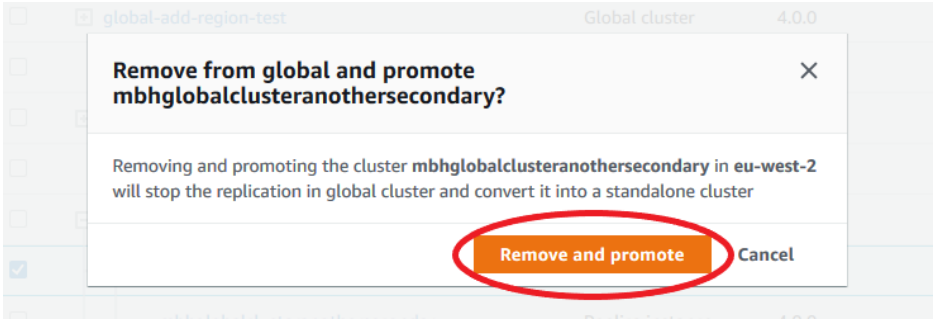
1. 登入 AWS 管理主控台 並導覽至 Amazon DocumentDB 主控台。
2. 選擇左側導覽上的叢集。



3. 展開全域叢集，以便您可以查看所有次要叢集。選取您要移除的次要叢集。選擇動作，然後在下拉式選單中，選擇從全域移除。



4. 隨即出現提示，要求您確認是否要從全域叢集分離次要叢集。選擇移除並提升，從全域叢集中移除叢集。



現在，叢集不再是次要叢集，也不再與主要叢集同步。它是具有完整讀取/寫入功能的獨立叢集。

移除或刪除所有次要叢集之後，您就可以用同樣的方式移除主要叢集。在移除所有次要叢集之前，您無法從全域叢集分離或移除主要叢集。全域叢集可能保留在叢集清單中，沒有區域和AZs。如果您不想再使用此全域叢集，可以刪除 。

Using the AWS CLI

若要從全域叢集中移除叢集，請使用下列參數執行 `remove-from-global-cluster` CLI 命令：

- `--global-cluster-identifier` — 全域叢集的名稱 （識別符）。
- `--db-cluster-identifier` — 從全域叢集中移除的每個叢集的名稱。

下列範例會先移除次要叢集，然後從全域叢集移除主要叢集。

針對 Linux、macOS 或 Unix：

```
aws docdb --region secondary_region \
  remove-from-global-cluster \
    --db-cluster-identifier secondary_cluster_ARN \
    --global-cluster-identifier global_cluster_id

aws docdb --region primary_region \
  remove-from-global-cluster \
    --db-cluster-identifier primary_cluster_ARN \
    --global-cluster-identifier global_cluster_id
```

對全域叢集中的每個次要區域重複 `remove-from-global-cluster--db-cluster-identifiersecondary_cluster_ARN`命令。

針對 Windows：

```
aws docdb --region secondary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier secondary_cluster_ARN ^
    --global-cluster-identifier global_cluster_id

aws docdb --region primary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier primary_cluster_ARN ^
    --global-cluster-identifier global_cluster_id
```

對全域叢集中的每個次要區域重複 `remove-from-global-cluster--db-cluster-identifiersecondary_cluster_ARN`命令。

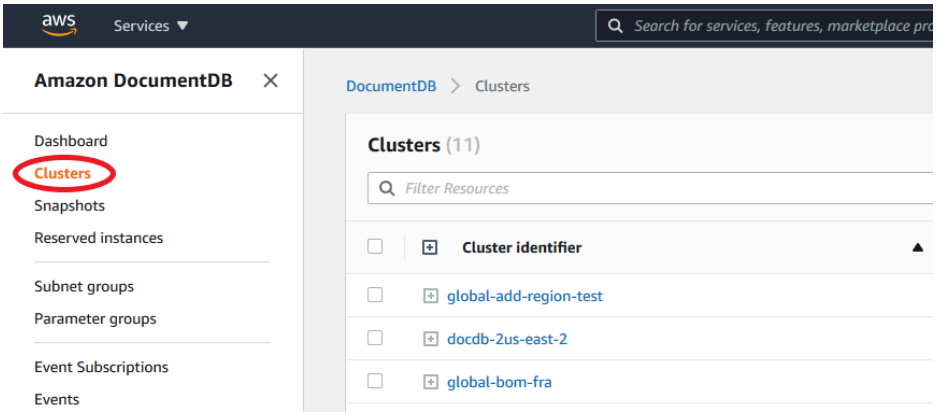
從 Amazon DocumentDB 全域叢集刪除叢集

若要刪除全域叢集，請執行下列動作：

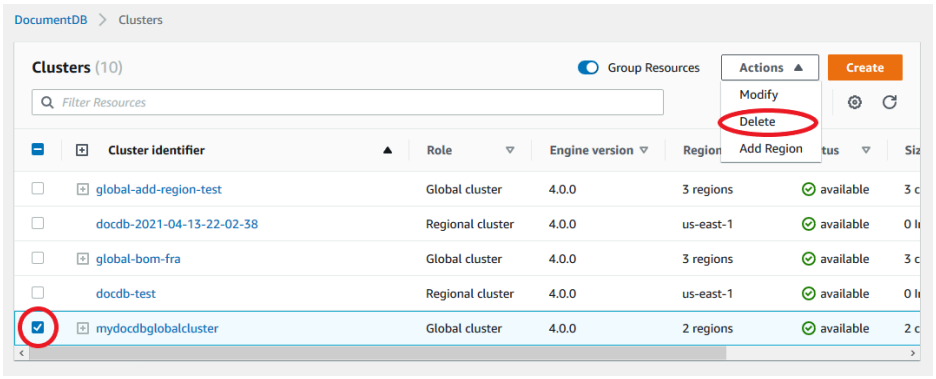
- 從全域叢集移除所有次要叢集。每個叢集都會成為獨立的叢集。請參閱上一節：從 Amazon DocumentDB 全域叢集移除叢集。
- 從每個獨立叢集刪除所有複本。
- 從全域叢集移除主要叢集。這將成為獨立的叢集。
- 從主要叢集，先刪除所有複本，然後刪除主要執行個體。從新獨立叢集刪除主要執行個體通常會同時移除叢集和全域叢集。

Using the AWS 管理主控台

1. 登入 AWS 管理主控台 並導覽至 Amazon DocumentDB 主控台。
2. 選擇叢集並尋找您要刪除的全域叢集。



3. 選取全域叢集後，從動作功能表中選擇刪除。



確認已從全域叢集中移除所有叢集。全域叢集應該會顯示零個區域和可用AZs，以及零個叢集的大小。如果全域叢集包含任何叢集，您便無法刪除它。您必須先遵循上一個步驟 中的指示從 Amazon DocumentDB 全域叢集移除叢集。

Using the AWS CLI

若要刪除全域叢集，請使用 的名稱 AWS 區域 和全域叢集識別符執行 `delete-global-cluster` CLI 命令，如下列範例所示。

針對 Linux、macOS 或 Unix：

```
aws docdb --region primary_region delete-global-cluster \
   --global-cluster-identifier global_cluster_id
```

針對 Windows：

```
aws docdb --region primary_region delete-global-cluster ^
   --global-cluster-identifier global_cluster_id
```

在次要區域中建立無周邊 Amazon DocumentDB 叢集

雖然 Amazon DocumentDB 全域叢集需要在與主要叢集 AWS 區域 不同的 中至少有一個次要叢集，但您可以使用次要叢集的無周邊組態。無周邊次要 Amazon DocumentDB 叢集是沒有執行個體的叢集。這種類型的組態可以降低全域叢集的費用。在 Amazon DocumentDB 叢集中，運算和儲存會分離。如果沒有執行個體，您不需要支付運算費用，而只會支付儲存費用。如果設定正確，無周邊次要的儲存磁碟區會與主要叢集保持同步。

您可以像平常在建立 Amazon DocumentDB 全域叢集時一樣新增次要叢集。不過，在主要叢集開始複寫至次要叢集之後，您會從次要叢集刪除唯讀執行個體。此次要叢集現在被視為「無標題」，因為它不再具有執行個體。不過，儲存磁碟區會與主要 Amazon DocumentDB 叢集保持同步。

> **Important**
>
> 只有在您可以容忍整個區域的故障 15 分鐘或更長時間時，才能使用無周邊叢集。使用無周邊次要叢集從全區域故障復原時，您需要在容錯移轉後建立新的執行個體。新的執行個體可能需要 10-15 分鐘才能使用。

如何將無周邊次要叢集新增至全域叢集

1. 登入 AWS 管理主控台 並開啟 Amazon DocumentDB 主控台。
2. 選擇左側導覽上的叢集。
3. 選擇需要次要叢集的全域叢集。確定主要叢集是 Available。
4. 針對動作，選擇新增區域。
5. 在新增區域頁面上，選擇次要區域。

   > **Note**
   >
   > 您無法為相同的全域叢集選擇已有次要叢集的區域。此外，它不能與主要叢集的區域相同。

6. 對於新的 區域中的次要 叢集，填寫剩餘的欄位。這些是與任何叢集執行個體相同的組態選項。
7. 新增區域。將區域新增至全域叢集後，該區域會出現在 Clusters中的 清單中 AWS 管理主控台。
8. 使用 AWS 管理主控台 或 檢查次要叢集及其讀取器執行個體的狀態，然後再繼續 AWS CLI。如果您使用 ，以下是範例命令 AWS CLI：

   ```
   $ aws docdb describe-db-clusters --db-cluster-identifier secondary-cluster-id --
   query '*[].[Status]' --output text
   ```

   新增的次要叢集狀態可能需要幾分鐘的時間才能從建立變更為可用。當叢集可用時，您可以刪除讀取器執行個體。
9. 選取次要叢集中的讀取器執行個體，然後選擇刪除。
10. 刪除讀取器執行個體後，次要叢集仍是全域叢集的一部分。它應該沒有與其相關聯的執行個體。

> **Note**
>
> 如果發生此類中斷，您可以使用此無周邊次要 Amazon DocumentDB 叢集，從主要區域中的意外中斷手動復原 Amazon DocumentDB 全域叢集。

連線至 Amazon DocumentDB 全域叢集

連線至全域叢集的方式取決於您是否需要寫入叢集或從叢集讀取：

- 對於唯讀請求或查詢，您可以連線到 中叢集的讀取器端點 AWS 區域。
- 若要執行資料操作語言 (DML) 或資料定義語言 (DDL) 陳述式，您需要連接到主要叢集的叢集端點。此端點可能與您的應用程式 AWS 區域 位於不同的 中。

在主控台中檢視全域叢集時，您可以看到與其所有叢集相關聯的所有一般用途端點。

連線至全域叢集的方式取決於您是否需要寫入資料庫或從資料庫讀取。對於您想要從主要區域提供的 DDL、DML 和讀取操作，請使用複本集模式下的叢集端點連線至您的主要叢集，且讀取偏好設定為 `secondaryPreferred=true`。這會將寫入流量路由到主要叢集的寫入器執行個體，並將流量讀取到主要叢集的複本執行個體。

對於跨區域唯讀流量，請在複本集模式下使用叢集端點連線至其中一個次要叢集。由於所有執行個體都是唯讀複本執行個體，請指定 以外的讀取偏好設定 `primary`（例如 `secondary`、`secondaryPreferred`或 `nearest`)。若要將延遲降至最低，請選擇所在 區域或最接近之 區域中的讀取者端點。

監控 Amazon DocumentDB 全域叢集

Amazon DocumentDB （具有 MongoDB 相容性） 與 CloudWatch 整合，因此您可以收集和分析叢集的操作指標。您可以使用 CloudWatch 主控台、Amazon DocumentDB 主控台、 AWS Command Line Interface (AWS CLI) 或 CloudWatch API 來監控這些指標。

若要監控全域叢集，請使用下列 CloudWatch 指標。

指標	說明
`GlobalClusterReplicatedWriteIO`	從主要 中叢集磁碟區複寫 AWS 區域 到次要 中叢集磁碟區的平均計費寫入 I/O 操作數 AWS 區域，每隔 5 分鐘報告一次。複寫`ReplicatedWriteIOs` 到每個次要區域的數目與主要區域`VolumeWriteIOPs` 執行的區域內數目相同。
`GlobalClusterDataTransferBytes`	從主要叢集傳輸 AWS 區域 到次要叢集的資料量 AWS 區域，以位元組為單位。
`GlobalClusterReplicationLag`	將變更事件從主要叢集的 複寫 AWS 區域 到次要叢集的 時，延遲量，以毫秒為單位 AWS 區域

如需如何檢視這些指標的詳細資訊，請參閱檢視 CloudWatch 資料。

災難復原和 Amazon DocumentDB 全域叢集

主題

- 執行 Amazon DocumentDB 全域叢集的受管容錯移轉
- 執行 Amazon DocumentDB 全域叢集的手動容錯移轉
- 執行 Amazon DocumentDB 全域叢集的切換
- 解除封鎖全域叢集切換或容錯移轉
- 管理 Amazon DocumentDB 全域叢集RPOs

透過使用全域叢集，您可以快速從災難中復原，例如區域故障。從災難復原通常使用 RTO 和 RPO 的值來測量。

- 復原時間點目標 (RTO) — 系統在災難發生後恢復運作狀態所需的時間。換言之，RTO 會測量停機時間。對於全域叢集，RTO 會在幾分鐘內完成。
- 復原點目標 (RPO) — 可能遺失的資料量 (以時間為單位)。對於全域叢集，RDO 通常以秒為單位測量。
- 若要從意外中斷中復原，您可以執行跨區域容錯移轉到全域叢集中的其中一個次要節點。當您的全域叢集有多個次要區域時，請務必分離您想要提升為主要區域的所有次要區域。然後，您可以將其中一個次要區域提升為新的主要區域 AWS 區域。最後，您可以在每個其他次要區域中建立新的叢集，並將這些叢集連接到您的全域叢集。

執行 Amazon DocumentDB 全域叢集的受管容錯移轉

此方法的目的是在發生實際區域災難或服務完全中斷的情況下，提供業務持續性。

在受管容錯移轉期間，您的主要叢集會容錯移轉至您選擇的次要區域，同時維護 Amazon DocumentDB 全域叢集的現有複寫拓撲。選擇的次要叢集會將其中一個唯讀節點提升為完整寫入器狀態。此步驟可讓叢集擔任主要叢集的角色。當此叢集擔任這個新角色時，您的資料庫在短時間內無法使用。當此次要叢集成為新的主要叢集時，未從舊主要叢集複寫到所選次要叢集的資料可能會遺失。舊的主要磁碟區會盡最大努力嘗試擷取快照，然後再與新的主要磁碟區同步，以便在快照上保留未複寫的資料。

> **Note**
>
> 只有在主要叢集和所有次要叢集具有相同的引擎版本時，您才能在 Amazon DocumentDB 全域叢集上執行受管跨區域叢集容錯移轉。如果您的引擎版本不相容，您可以依照 執行 Amazon DocumentDB 全域叢集的手動容錯移轉 中的步驟手動執行容錯移轉。
> 如果區域的引擎版本不相符，則會封鎖容錯移轉。檢查是否有任何待定升級並套用，以確保所有區域的引擎版本相符，且全域叢集容錯移轉已解除封鎖。如需詳細資訊，請參閱解除封鎖全域叢集切換或容錯移轉。

若要將資料遺失降至最低，請在使用此功能之前執行下列動作：

- 讓應用程式離線，以防止寫入傳送至 Amazon DocumentDB 全域叢集的主要叢集。
- 檢查所有 Amazon DocumentDB 次要叢集的延遲時間。選擇複寫延遲最低的次要區域，即可將目前失敗的主要區域的資料遺失降到最低。檢視 Amazon CloudWatch 中的 指標，檢查全域叢集中所有 Amazon DocumentDB 次要叢集的延遲時間。 `GlobalClusterReplicationLag` Amazon CloudWatch 這些指標會顯示次要叢集複寫到主要叢集落後多遠 （以毫秒為單位）。

  如需 Amazon DocumentDB CloudWatch 指標的詳細資訊，請參閱 Amazon DocumentDB 指標。

在受管容錯移轉期間，所選次要叢集會提升為主要叢集的新角色。不過，它不會繼承主要叢集的各種組態選項。組態不相符可能會導致效能問題、工作負載不相容，以及其他異常行為。若要避免此類問題，請針對下列項目解決 Amazon DocumentDB 全域叢集之間的差異：

- 視需要為新的主要叢集設定 Amazon DocumentDB 叢集參數群組 — 您可以為 Amazon DocumentDB 全域叢集中的每個叢集獨立設定 Amazon DocumentDB 叢集參數群組。因此，當您提升次要叢集以接管主要角色時，次要叢集的參數群組的設定可能會與主要叢集不同。若是如此，請修改提升次要叢集的參數群組，以符合主要叢集的設定。若要了解作法，請參閱修改 Amazon DocumentDB 叢集參數群組。
- 設定監控工具和選項，例如 Amazon CloudWatch 事件和警示 — 視需要為提升的叢集設定相同的記錄功能、警示等。與參數群組一樣，在容錯移轉程序期間，這些功能的組態不會從主要叢集繼承。某些 CloudWatch 指標 (例如複寫延遲) 僅適用於次要區域。因此，容錯移轉會改變檢視這些指標並對其設定警示的方式，而且可能需要變更任何預先定義的儀表板。如需 Amazon DocumentDB 叢集和監控的詳細資訊，請參閱 在 Amazon DocumentDB 中監控和記錄。

一般而言，選擇的次要叢集會在一分鐘內擔任主要角色。一旦新的主要區域的寫入器節點可用，您就可以將應用程式連線到該節點並恢復工作負載。Amazon DocumentDB 提升新的主要叢集後，會自動重建所有其他次要區域叢集。

由於 Amazon DocumentDB 全域叢集使用非同步複寫，因此每個次要區域中的複寫延遲可能會有所不同。Amazon DocumentDB 會重建這些次要區域，使其具有與新主要區域叢集完全相同的point-in-time資料。整個重建工作的期間可能持續幾分鐘到數小時，取決於儲存磁碟區的大小和區域之間的距離。當次要區域叢集從新的主要區域完成重建時，便可供讀取存取使用。一旦新的主要寫入器提升並可用，新的主要區域的叢集就可以處理 Amazon DocumentDB 全域叢集的讀取和寫入操作。

若要還原全域叢集的原始拓撲，Amazon DocumentDB 會監控舊主要區域的可用性。一旦該區域正常運作且再次可用，Amazon DocumentDB 會自動將其新增回全域叢集做為次要區域。在舊主要區域中建立新的儲存磁碟區之前，Amazon DocumentDB 會嘗試在故障時擷取舊儲存磁碟區的快照。這樣做是為了讓您用它來復原任何遺失的資料。如果此操作成功，Amazon DocumentDB 會將名為 "rds：docdb-unplanned-global-failover-name-of-old-primary-DB-cluster-timestamp" 的快照置於 的快照區段中 AWS 管理主控台。您也可以在 `DescribeDBClusterSnapshots` API 操作傳回的資訊中看到此快照。

> **Note**
>
> 舊儲存磁碟區的快照是系統快照，會受到舊的主要叢集上設定的備份保留期限的限制。若要在保留期間之外保留此快照，您可以複製該快照並將它儲存為手動快照。若要進一步了解如何複製快照，包括定價在內，請參閱 複製叢集快照。

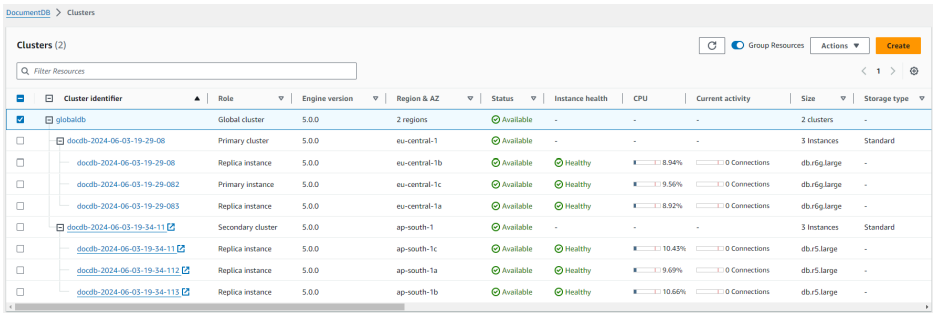
還原原始拓撲之後，您可以在對業務和工作負載最有意義時執行切換操作，將全域叢集容錯移轉回原始主要區域。若要啟用，請依照「執行 Amazon DocumentDB 全域叢集的切換」中的步驟進行。

您可以使用 AWS 管理主控台、 AWS CLI或 Amazon DocumentDB API 容錯移轉 Amazon DocumentDB 全域叢集。

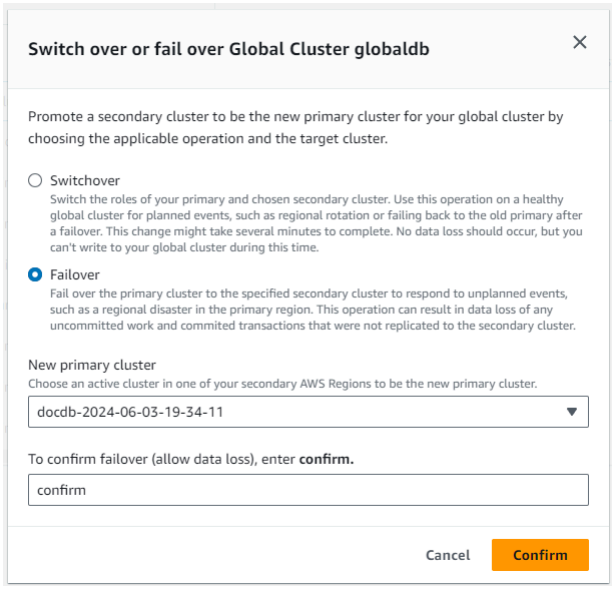
Using the AWS 管理主控台

在 Amazon DocumentDB 全域叢集上執行受管容錯移轉

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。
3. 尋找並選擇您要容錯移轉的 Amazon DocumentDB 全域叢集。

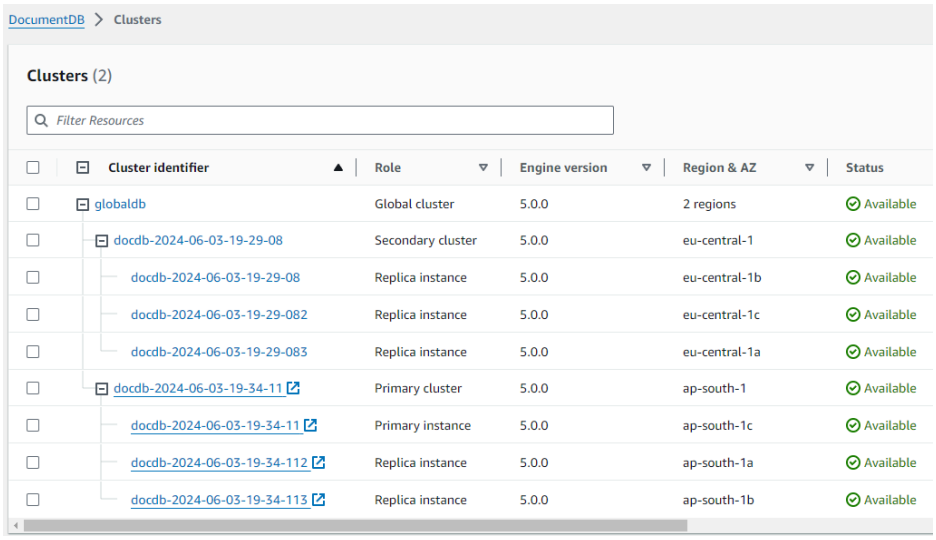


4. 從動作功能表中選擇切換或容錯移轉。
5. 在出現的對話方塊中，選擇容錯移轉，然後從新增主要叢集欄位下拉式清單中選擇次要叢集。



6. 在最後一個欄位中輸入「確認」。然後選擇 Confirm (確認)。

   主要叢集的狀態會變更為「容錯移轉」。此條件大約需要一分鐘。在此期間，新主要叢集的狀態會顯示「修改中...」。提升新的主要節點後，會顯示「可用」，並能夠提供讀取和寫入交易。包含舊主要節點的次要區域會在重新同步到新的主要節點時顯示「重新同步...」。與新的主要節點類似，只有在狀態變更為「可用」時，它才能提供交易。
7. 完成時，原始主要叢集會成為次要叢集。選取的次要叢集會成為主要叢集。



Using the AWS CLI

在 Amazon DocumentDB 全域叢集上執行受管容錯移轉

執行 failover-global-cluster CLI 命令以容錯移轉您的 Amazon DocumentDB 全域叢集。使用 命令，傳遞下列選項的值：

- `--region`
- `--global-cluster-identifier`
- `--target-db-cluster-identifier`
- `--allow-data-loss`

在下列範例中，將每個*########*取代為您叢集的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb failover-global-cluster \
   --region region_of_selected_secondary \
   --global-cluster-identifier global_cluster_id \
   --target-db-cluster-identifier arn_of_secondary_to_promote \
   --allow-data-loss
```

針對 Windows：

```
aws docdb failover-global-cluster ^
   --region region_of_selected_secondary ^
```

```
    --global-cluster-identifier global_cluster_id ^
    --target-db-cluster-identifier arn_of_secondary_to_promote ^
    --allow-data-loss
```

執行 Amazon DocumentDB 全域叢集的手動容錯移轉

如果一個叢集中的整個叢集 AWS 區域 無法使用，您可以將全域叢集中的另一個叢集提升為具有讀取/寫入功能。

如果不同 中的叢集 AWS 區域 是較佳的主要叢集選擇，您可以手動啟用全域叢集容錯移轉機制。例如，您可以增加其中一個次要叢集的容量，然後將它提升為主要叢集。或者， 之間的活動平衡 AWS 區域 可能會變更，因此將主要叢集切換到不同的 AWS 區域 叢集可能會降低寫入操作的延遲。

下列程序概述如何提升 Amazon DocumentDB 全域叢集中的其中一個次要叢集。

若要提升次要叢集：

1. 在中斷 AWS 區域 的情況下，停止對 中的主要叢集發出 DML 陳述式和其他寫入操作。
2. 從次要 識別叢集 AWS 區域 ，以用作新的主要叢集。如果您的全域叢集 AWS 區域 有兩個 （或更多）次要叢集，請選擇延遲時間最短的次要叢集。
3. 從全域叢集分離您選擇的次要叢集。

   從全域叢集移除次要叢集會立即停止從主要叢集到此次要叢集的複寫，並將其提升為具有完整讀取/寫入功能的獨立佈建叢集。與 區域中主要叢集相關聯的任何其他次要叢集與中斷仍然可用，並且可以接受來自應用程式的呼叫。它們也會取用資源。由於您要重新建立全域叢集，為了避免分裂大腦和其他問題，請在接下來的步驟中建立新的全域叢集之前移除其他次要叢集。

   如需分離的詳細步驟，請參閱從 Amazon DocumentDB 全域叢集移除叢集。
4. 當您開始在下一個步驟中新增區域時，此叢集會成為新全域叢集的主要叢集。
5. 將 AWS 區域 新增至叢集。當您執行這項操作時，從主要到次要的複寫程序即會開始。
6. AWS 區域 視需要新增更多 ，以重新建立支援應用程式所需的拓撲。請確定應用程式寫入會在進行這類變更之前、期間和之後傳送到正確的叢集，以避免全域叢集中叢集之間的資料不一致 （分裂大腦問題）。
7. 當中斷解決且您已準備好再次將原始 指派 AWS 區域 為主要叢集時，請反向執行相同的步驟。
8. 從全域叢集移除其中一個次要叢集。這將使其能夠提供讀取/寫入流量。
9. 將所有寫入流量重新導向至原始 中的主要叢集 AWS 區域。
10. 新增 AWS 區域 以在與之前 AWS 區域 相同的 中設定一或多個次要叢集。

您可以使用 AWS SDKs來管理 Amazon DocumentDB 全域叢集，讓您建立解決方案來自動化災難復原和業務持續性規劃使用案例的全域叢集容錯移轉程序。Apache 2.0 授權的客戶可以使用這類解決方案，也可以從此處的工具儲存庫存取。 https://github.com/awslabs/amazon-documentdb-tools/tree/master/global-clusters-automation此解決方案利用 Amazon Route 53 進行端點管理，並提供可以根據適當事件觸發的 AWS Lambda 函數。

執行 Amazon DocumentDB 全域叢集的切換

您可以使用切換來例行變更主要叢集的區域。此方法適用於受控情況，例如操作維護及其他計劃內操作程序。

使用切換有三種常見的使用案例：

- 對於特定行業強制實施的「區域輪換」需求。例如，金融服務法規可能希望第 0 層系統切換到不同的區域數個月時間，以確保定期演練災難復原程序。
- 對於多區域「全天候」應用程式。例如，某家公司可能希望根據不同時區的營業時間，在不同區域提供較低的延遲寫入。
- 可作為在容錯移轉後，容錯回復至原始主要區域的零資料遺失方法。

> **Note**
>
> 切換旨在用於運作狀態良好的 Amazon DocumentDB 全域叢集。若要從意外中斷復原，請依照 執行 Amazon DocumentDB 全域叢集的手動容錯移轉 中適當的程序進行。
> 若要執行切換，所有次要區域都必須執行與主要區域完全相同的引擎版本。如果區域的引擎版本不相符，則會封鎖切換。檢查是否有任何待定升級，並將其套用以確保所有區域的引擎版本相符，並且全域叢集切換已解除封鎖。如需詳細資訊，請參閱解除封鎖全域叢集切換或容錯移轉。

在切換期間，Amazon DocumentDB 會在維護您全域叢集的現有複寫拓撲時，將主要叢集切換到您選擇的次要區域。在開始切換程序之前，Amazon DocumentDB 會等待所有次要區域叢集與主要區域叢集完全同步。然後，主要區域中的資料庫叢集會變成唯讀，而選擇的次要叢集會將其中一個唯讀節點提升為完整寫入器狀態。將此節點提升為寫入器可讓該次要叢集擔任主要叢集的角色。由於所有次要叢集都在程序開始時與主要叢集同步，新的主要叢集會繼續 Amazon DocumentDB 全域叢集的操作，而不會遺失任何資料。您的資料庫在短時間內無法使用，因為同時間主要叢集和選取的次要叢集會承擔其新角色。

若要最佳化應用程式可用性，請在使用此功能之前執行下列動作：

- 在非尖峰時間或寫入主要叢集最少的另一個時間執行此操作。
- 讓應用程式離線，以防止寫入傳送至 Amazon DocumentDB 全域叢集的主要叢集。
- 檢視 Amazon CloudWatch 中的 指標，檢查全域叢集中所有 Amazon DocumentDB 次要叢集的延遲時間。 `GlobalClusterReplicationLag` Amazon CloudWatch 此指標顯示次要叢集複寫到主要叢集落後多遠 （以毫秒為單位）。此值與 Amazon DocumentDB 完成轉換所需的時間直接成正比。因此，延遲值越大，轉換所需的時間越長。

  如需 Amazon DocumentDB CloudWatch 指標的詳細資訊，請參閱 Amazon DocumentDB 指標。

在轉換期間，選擇的次要資料庫叢集會提升為作為主要資料庫叢集的新角色。但是，它不會繼承主要資料庫叢集的各種組態選項。組態不相符可能會導致效能問題、工作負載不相容，以及其他異常行為。若要避免此類問題，請針對下列項目解決 Amazon DocumentDB 全域叢集之間的差異：

- 視需要為新的主要叢集設定 Amazon DocumentDB 資料庫叢集參數群組 — 您可以為 Amazon DocumentDB 全域叢集中的每個叢集獨立設定 Amazon DocumentDB 叢集參數群組。這表示當您提升次要資料庫叢集以接管主要角色時，其參數群組可能會設定與主要資料庫叢集不同的參數群組。如果是這樣，請修改提升的次要資料庫叢集的參數群組，以符合主要叢集的設定。若要瞭解如何操作，請參閱管理 Amazon DocumentDB 叢集參數群組。
- 設定監控工具和選項，例如 Amazon CloudWatch Events 和警示 — 視需要為提升的叢集設定相同的記錄功能、警示等。與參數群組一樣，在轉換程序期間，這些功能的組態不會從主要叢集繼承。有些 CloudWatch 指標，例如複寫延遲，僅適用於主要區域。因此，轉換會改變檢視這些指標並對其設定警示的方式，而且可能需要變更任何預先定義的儀表板。如需詳細資訊，請參閱在 Amazon DocumentDB 中監控和記錄。

> **Note**
>
> 角色轉換通常需要數分鐘才能完成。

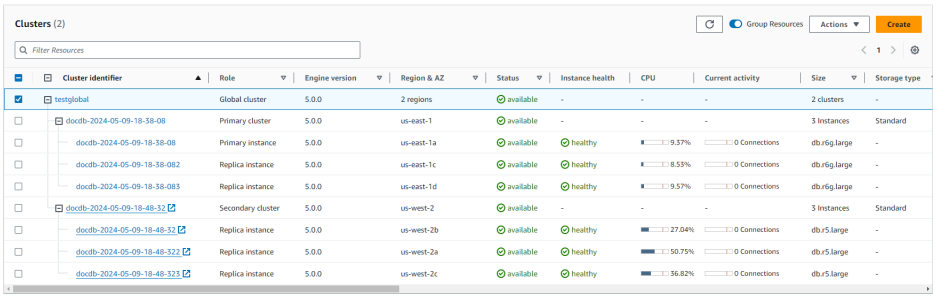
當切換程序完成時，提升的 Amazon DocumentDB 叢集可以處理全域叢集的寫入操作。

您可以使用 AWS 管理主控台 或 來切換 Amazon DocumentDB 全域叢集 AWS CLI：

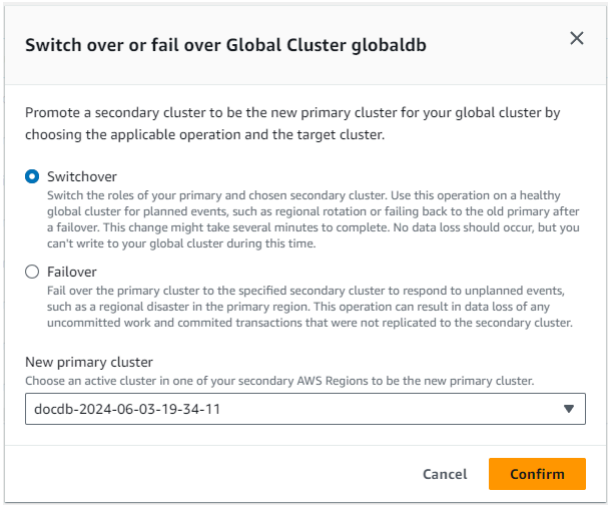
Using the AWS 管理主控台

在 Amazon DocumentDB 全域叢集上執行切換

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/docdb：// 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。
3. 尋找並選取您要切換的 Amazon DocumentDB 全域叢集。



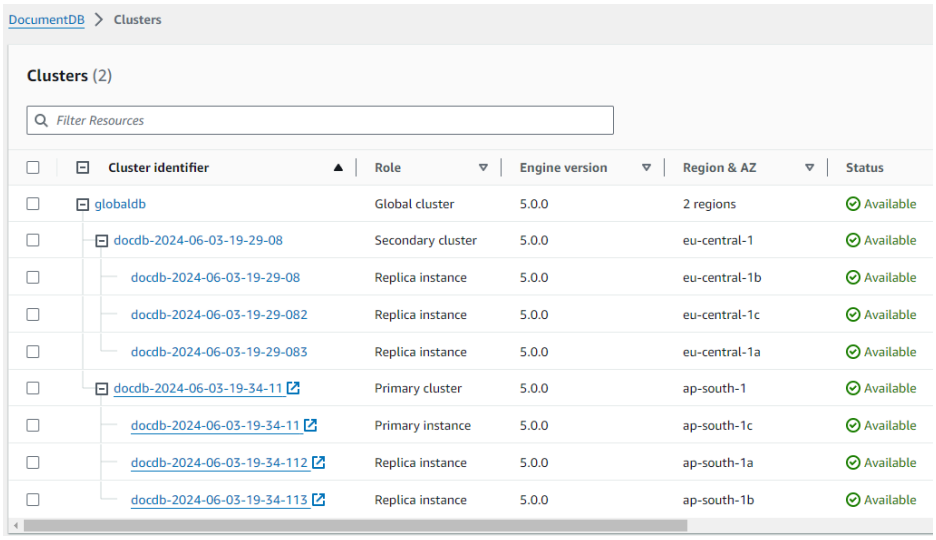
4. 從動作功能表中選擇切換或容錯移轉。
5. 在出現的對話方塊中，選擇切換，然後從新增主要叢集欄位下拉式清單中選擇次要叢集。



6. 選擇確認。

   主要叢集的狀態會變更為「切換」。此條件大約需要三分鐘。在此期間，所有區域叢集的狀態會顯示「修改中...」。同步區域並提升新的主要區域後，將顯示所有狀態欄位的「可用」，並能夠提供交易。

7. 完成時，原始主要叢集會成為次要叢集。選取的次要叢集會成為主要叢集。



Using the AWS CLI

在 Amazon DocumentDB 全域叢集上執行切換

執行 `switchover-global-cluster` CLI 命令來切換 Amazon DocumentDB 全域叢集。使用 命令，傳遞下列選項的值：

- `--region`
- `--global-cluster-identifier`
- `--target-db-cluster-identifier`

在下列範例中，將每個*#########*取代為您叢集的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb switchover-global-cluster \
   --region region_of_primary \
   --global-cluster-identifier global_cluster_id \
   --target-db-cluster-identifier arn_of_secondary_to_promote
```

針對 Windows：

```
aws docdb switchover-global-cluster ^
   --region region_of_primary ^
   --global-cluster-identifier global_cluster_id ^
```

```
    --target-db-cluster-identifier arn_of_secondary_to_promote
```

解除封鎖全域叢集切換或容錯移轉

不同引擎版本的區域

當全域叢集中的所有區域叢集都位於相同的引擎版本時，會封鎖全域叢集切換和容錯移轉。如果版本不相符，您可能會在呼叫切換或容錯移轉時看到此錯誤：指定的目標資料庫叢集正在執行與來源資料庫叢集具有不同修補程式層級的引擎版本。定期套用最新的引擎版本，讓您的全域叢集保持良好狀態。

若要解決此錯誤，請先套用任何待定的維護動作項目，將所有次要區域更新為相同的引擎版本。若要檢視待定的維護動作項目，並套用任何必要的變更以修正問題，請執行下列其中一個索引標籤中的指示：

Using the AWS 管理主控台

若要解除封鎖全域叢集切換或容錯移轉，您必須判斷叢集是否有任何待定的維護動作，並加以套用。請依照下列步驟檢視和套用維護動作：

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。
3. 在叢集表格中，在叢集識別符欄中尋找您的全域叢集。在全域叢集下，記下指定全域叢集的每個次要叢集和主要叢集，並為每個叢集執行以下步驟。
4. 對於每個次要叢集：
    a. 如果您的叢集有可用的更新，則會在維護欄中指出為可用、必要或下一個視窗。
    b. 若要採取動作，請選擇叢集以顯示其詳細資訊，然後選擇維護和備份。待定維護項目隨即出現。
    c. 在描述下，如果它指出「新的維護更新可用」，請選取它，然後選擇立即套用。
5. 對於您的主要叢集：
    a. 如果您的叢集有可用的更新，則會在維護欄中指出為可用、必要或下一個視窗。
    b. 若要採取動作，請選擇叢集以顯示其詳細資訊，然後選擇維護和備份。待定維護項目隨即出現。
    c. 在描述下，如果它指出「新的維護更新可用」，請選取它，然後選擇立即套用。

Using the AWS CLI

若要解除封鎖全域叢集切換或容錯移轉，您必須判斷叢集是否有任何待定的維護動作，並加以套用。請依照下列步驟，先在次要叢集上檢視和套用維護動作，然後在全域叢集的主要叢集上套用維護動作：

1. 先在每個次要區域的區域叢集上執行下列項目，然後針對主要區域區域叢集執行以下內容。
2. 使用 `--resource-identifier`選項執行 describe-pending-maintenance-actions CLI 命令，以判斷您的 Amazon DocumentDB 區域叢集是否有任何可用的維護動作。

   在下列範例中，將每個*#########*取代為您叢集的資訊。

   針對 Linux、macOS 或 Unix：

```
aws docdb describe-pending-maintenance-action \
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15
```

   針對 Windows：

```
aws docdb describe-pending-maintenance-action ^
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15
```

   結果如下所示：

```
{
    "PendingMaintenanceActions": [
        {
            "ResourceIdentifier": "arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "system-update",
                    "CurrentApplyDate": "2025-04-11T03:01:00Z",
                    "Description": "db-version-upgrade",
                    "ForcedApplyDate": "2025-06-18T03:01:00Z",
                    "AutoAppliedAfterDate": "2025-05-11T03:01:00Z"
                    "OptInStatus": "pending"
                }
            ]
```

```
        }
    ]
}
```

3. 如果需要維護動作，請使用下列選項執行 apply-pending-maintenance-action CLI 命令：

   - --resource-identifier
   - --apply-action
   - --opt-in-type
   - --region

   在下列範例中，將每個*#########*取代為您叢集的資訊。

   針對 Linux、macOS 或 Unix：

```
aws docdb apply-pending-maintenance-action \
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15 \
   --apply-action system-update \
   --opt-in-type immediate \
   --region us-east-1
```

   針對 Windows：

```
aws docdb apply-pending-maintenance-action ^
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15 ^
   --apply-action system-update ^
   --opt-in-type immediate ^
   --region us-east-1
```

4. 維護動作完成後，請再次執行 describe-pending-maintenance-actions命令，以確保叢集沒有其他待定動作。

   您想要的結果是：

```
{
    "PendingMaintenanceActions": []
}
```

Using the Amazon DocumentDB API

若要解除封鎖全域叢集切換或容錯移轉，您必須判斷叢集是否有任何待定的維護動作，並加以套用。使用下列 APIs來檢視和套用維護動作：

1. 先在每個次要區域的區域叢集上執行下列項目，然後針對主要區域區域叢集執行以下內容。
2. 呼叫 PendingMaintenanceAction API，以判斷您的 Amazon DocumentDB 全域叢集是否有任何可用的維護動作。
3. 呼叫 ApplyPendingMaintenanceAction API 來套用任何變更。

目標叢集上的待定變更

當目標次要叢集具有尚未套用的排程變更時，切換和容錯移轉也會遭到封鎖，例如您為下一個維護時段請求的修改或維護動作。在此情況下，您可能會在呼叫切換或容錯移轉時看到此錯誤：您無法容錯移轉至具有 ARN *arn#aws#rds#us-east-1#001234567890#cluster#docdb-2025-03-27-19-21-15* 的叢集，因為它正在修改或正在等待修改。當叢集可用時，請再試一次。若要解除封鎖操作，請移除目標叢集上的排程變更。步驟取決於變更的類型。移除排定的變更且叢集狀態傳回 後`available`，請重試切換或容錯移轉。

取消排定的維護動作

您無法從 取消排定的維護動作 （取消選擇加入） AWS 管理主控台，因此請使用 AWS CLI。

Using the AWS CLI

1. 使用 `--resource-identifier`選項執行 describe-pending-maintenance-actions CLI 命令，並尋找其 `OptInStatus`為 的維護動作`next-maintenance`。

   在下列範例中，將每個*#########*取代為您叢集的資訊。

   針對 Linux、macOS 或 Unix：

   ```
   aws docdb describe-pending-maintenance-actions \
      --resource-identifier arn:aws:rds:us-
   east-1:001234567890:cluster:docdb-2025-03-27-19-21-15 \
      --region us-east-1
   ```

   針對 Windows：

```
aws docdb describe-pending-maintenance-actions ^
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15 ^
   --region us-east-1
```

結果看起來與以下內容類似。請記下您在下一個步驟中使用的Action值 (os-upgrade在此範例中為)。

```
{
    "PendingMaintenanceActions": [
        {
            "ResourceIdentifier": "arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "os-upgrade",
                    "OptInStatus": "next-maintenance",
                    "CurrentApplyDate": "2026-09-02T03:02:00Z",
                    "Description": "New Operating System update is available"
                }
            ]
        }
    ]
}
```

2. 使用 執行 apply-pending-maintenance-action CLI 命令--opt-in-type undo-opt-in，將Action值從上一個步驟傳遞至 ，以取消排程動作--apply-action。

   針對 Linux、macOS 或 Unix：

```
aws docdb apply-pending-maintenance-action \
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15 \
   --apply-action os-upgrade \
   --opt-in-type undo-opt-in \
   --region us-east-1
```

   針對 Windows：

```
aws docdb apply-pending-maintenance-action ^
```

```
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15 ^
   --apply-action os-upgrade ^
   --opt-in-type undo-opt-in ^
   --region us-east-1
```

在回應中，維護動作不再有 OptInStatus，確認排定的選擇加入已取消。

```
{
    "ResourcePendingMaintenanceActions": {
        "ResourceIdentifier": "arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15",
        "PendingMaintenanceActionDetails": [
            {
                "Action": "os-upgrade",
                "Description": "New Operating System update is available"
            }
        ]
    }
}
```

還原排定的叢集修改

Using the AWS CLI

1. 執行 describe-db-clusters CLI 命令並檢查PendingModifiedValues輸出欄位，以檢查排定的修改。

> **Note**
>
> 使用此命令的 Amazon RDS CLI (aws rds)，而不是 Amazon DocumentDB CLI (aws docdb)，因為 Amazon DocumentDB describe-db-clusters API 不會傳回叢集層級PendingModifiedValues欄位。

在下列範例中，將每個*########*取代為您叢集的資訊。

針對 Linux、macOS 或 Unix：

```
aws rds describe-db-clusters \
```

```
    --db-cluster-identifier docdb-2025-03-27-19-21-15 \
    --query 'DBClusters[0].PendingModifiedValues' \
    --region us-east-1
```

針對 Windows：

```
aws rds describe-db-clusters ^
    --db-cluster-identifier docdb-2025-03-27-19-21-15 ^
    --query "DBClusters[0].PendingModifiedValues" ^
    --region us-east-1
```

結果會顯示排程的變更。在此範例中，叢集名稱 (DBClusterIdentifier) 的變更正在等待中，從 docdb-2025-03-27-19-21-15變更為 docdb-new-name。

```
{
    "DBClusterIdentifier": "docdb-new-name"
}
```

2. 使用 執行 modify-db-cluster CLI 命令來還原變更--apply-immediately，將修改後的值設回其原始值。此範例會將 設--new-db-cluster-identifier回叢集的目前名稱，以還原待定名稱變更。

   針對 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster \
    --db-cluster-identifier docdb-2025-03-27-19-21-15 \
    --new-db-cluster-identifier docdb-2025-03-27-19-21-15 \
    --apply-immediately \
    --region us-east-1
```

   針對 Windows：

```
aws docdb modify-db-cluster ^
    --db-cluster-identifier docdb-2025-03-27-19-21-15 ^
    --new-db-cluster-identifier docdb-2025-03-27-19-21-15 ^
    --apply-immediately ^
    --region us-east-1
```

管理 Amazon DocumentDB 全域叢集RPOs

透過 Amazon DocumentDB 全域叢集，您可以使用 `global_db_rpo` 參數來管理復原點目標 (RPO)。RPO 表示發生中斷時可能遺失的最大資料量。

當您為 Amazon DocumentDB 全域叢集設定 RPO 時，Amazon DocumentDB 會監控所有次要叢集的 RPO 延遲時間。此監控可確保至少一個次要叢集保持在目標 RPO 時段內。

RPO 設定控制 Amazon DocumentDB 如何在主要叢集上管理寫入交易，以在發生容錯移轉時限制潛在的資料遺失。Amazon DocumentDB 會評估 RPO 和 RPO 延遲時間，以在主要 上遞交（或封鎖）交易，如下所示：

- 如果至少有一個次要資料庫叢集的 RPO 延遲時間小於 RPO，則會遞交交易。
- 如果所有次要資料庫叢集的 RPO 延遲時間大於 RPO，則會封鎖交易。

換句話說，如果所有次要叢集都落後於目標 RPO，Amazon DocumentDB 會暫停主要叢集上的交易。一旦至少一個次要資料庫叢集的延遲時間低於 RPO，Amazon DocumentDB 會繼續並遞交暫停的交易。因此，在滿足 RPO 之前，沒有任何可以遞交的交易。

此 `global_db_rpo` 參數是動態的。如果您決定不要讓所有寫入交易都停止，直到延遲降低到足夠的程度，那麼您可以快速重設它。在此情況下，Amazon DocumentDB 會在短暫延遲後套用變更。

> **Important**
>
> 在只有兩個 AWS 區域的全域資料庫中，我們建議將`global_db_rpo`參數的預設值保留在次要區域的參數群組中。否則，由於主要 AWS 區域遺失而執行容錯移轉可能會導致 Amazon DocumentDB 暫停交易。相反地，請等到 Amazon DocumentDB 完成重建舊故障 AWS 區域中的叢集，再變更此參數以強制執行最大 RPO。

主題

- 設定復原點目標
- 檢視復原點目標
- 停用復原點目標

設定復原點目標

global_db_rpo 參數控制 Amazon DocumentDB 資料庫的 RPO 設定。有效值的範圍從 20 秒到 2，147，483，647 秒 (68 年）。選擇逼真的值來滿足您的業務需求。例如，您可能想要允許最多 10 分鐘的 RPO，在此情況下，您可以將值設定為 600。

您可以使用 AWS 管理主控台、 或 Amazon DocumentDB API AWS CLI，為您的 Amazon DocumentDB 全域叢集設定此值。

Using the AWS 管理主控台

設定 RPO

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 選擇 Amazon DocumentDB 全域叢集的主要叢集，然後開啟組態索引標籤以尋找其資料庫叢集參數群組。

   無法直接編輯參數群組。而是執行下列動作：

   - 使用適當的預設參數群組作為起點，建立自訂資料庫叢集參數群組。
   - 在自訂資料庫叢集參數群組上，設定 global_db_rpo 參數的值以符合您的使用案例。有效值範圍為 20 秒至最大整數值 2,147,483,647 (68 年)。
   - 將修改過的資料庫叢集參數群組套用至您的 Amazon DocumentDB 資料庫叢集。

如需修改資料庫叢集參數群組的詳細資訊，請參閱 修改 Amazon DocumentDB 叢集參數群組。

Using the AWS CLI

若要設定 global_db_rpo 參數，請使用 modify-db-cluster-parameter-group CLI 命令。在 命令中，指定主要叢集參數群組的名稱，以及 RPO 參數的值。

下列範例會針對名稱為 my_custom_global_parameter_group 的主要資料庫叢集參數群組，將 RPO 設定為 600 秒 (10 分鐘)。

針對 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my_custom_global_parameter_group \
    --parameters
 "ParameterName=global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

針對 Windows：

```
aws docdb modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my_custom_global_parameter_group ^
    --parameters
 "ParameterName=global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

Using the Amazon DocumentDB API

若要修改 global_db_rpo 參數，請使用 ModifyDBClusterParameterGroup API 操作。

檢視復原點目標

全域叢集的復原點目標 (RPO) 會存放在每個資料庫叢集的 global_db_rpo 參數中。

您可以使用 CLI 來檢視 Amazon DocumentDB 資料庫叢集的 global_db_rpo 參數。使用 --query選項僅傳回global_db_rpo參數群組中的 參數。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name my_custom_global_parameter_group \
    --query "Parameters[?ParameterName=='global_db_rpo']"
```

針對 Windows：

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name my_custom_global_parameter_group ^
    --query "Parameters[?ParameterName=='global_db_rpo']"
```

此命令會傳回如下輸出：

```
[
    {
        "ParameterName": "global_db_rpo",
        "Description": "(s) Recovery point objective threshold, in seconds, that blocks
 user commits when it is violated.",
        "Source": "engine-default",
        "ApplyType": "dynamic",
        "DataType": "integer",
```

```
        "AllowedValues": "20-2147483647",
        "IsModifiable": true,
        "ApplyMethod": "immediate"
    }
]
```

如需檢視叢集參數群組參數的詳細資訊，請參閱 管理 Amazon DocumentDB 叢集參數群組。

停用復原點目標

若要停用 RPO，請重設 global_db_rpo 參數。您可以使用 AWS 管理主控台、 AWS CLI或 Amazon DocumentDB API 重設參數。

Using the AWS 管理主控台

停用 RPO

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/docdb：// 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。
3. 在清單中，選擇主要資料庫叢集參數群組。
4. 選擇 global_db_rpo 參數旁的選項按鈕。
5. 選擇重設為預設值並確認。

如需如何使用 主控台重設參數的詳細資訊，請參閱 修改 Amazon DocumentDB 叢集參數群組。

Using the AWS CLI

若要重設 global_db_rpo 參數，請使用 reset-db-cluster-parameter-group 命令。

針對 Linux、macOS 或 Unix：

```
aws docdb reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my_custom_global_parameter_group \
    --parameters "ParameterName=global_db_rpo,ApplyMethod=immediate"
```

針對 Windows：

```
aws docdb reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my_custom_global_parameter_group ^
    --parameters "ParameterName=global_db_rpo,ApplyMethod=immediate"
```

若要重設 `global_db_rpo` 參數，請使用 ResetDBClusterParameterGroup API 操作。

管理 Amazon DocumentDB 叢集

若要管理 Amazon DocumentDB 叢集，您必須擁有具有適當 Amazon DocumentDB 控制平面許可的 IAM 政策。這些許可可讓您建立、修改和刪除叢集和執行個體。此`AmazonDocDBFullAccess`政策提供管理 Amazon DocumentDB 叢集所需的所有必要許可。

下列主題說明如何在使用 Amazon DocumentDB 叢集時執行各種任務，包括建立、刪除、修改、連線至和檢視叢集。

主題

- 了解叢集
- Amazon DocumentDB 叢集設定
- Amazon DocumentDB 叢集儲存組態
- 判斷叢集的狀態
- Amazon DocumentDB 叢集生命週期
- 擴展 Amazon DocumentDB 叢集
- 複製 Amazon DocumentDB 叢集的磁碟區
- 了解 Amazon DocumentDB 叢集容錯能力

了解叢集

Amazon DocumentDB 會分開運算和儲存，並將資料複寫和備份卸載至叢集磁碟區。叢集磁碟區提供耐用、可靠且高度可用的儲存層，在三個可用區域以六種方法複寫資料。複本提供了更高的資料可用性與讀取規模調整。每個叢集最多可擴展 15 個複本。

名詞	說明	API 操作 (動詞)

叢集	包含一或多個執行個體，以及一個管理這些執行個體資料的叢集儲存磁碟區。	`create-db-cluster`
		`delete-db-cluster`
		`describe-db-clusters`

名詞	說明	API 操作 (動詞)
		`modify-db-cluster`

執行個體	從叢集儲存磁碟區讀取資料或將資料寫入其中，都是透過執行個體完成。在指定的叢集，有兩種類型的執行個體：主要執行個體和複寫執行個體。叢集一律有一個主要執行個體，並且可以有 0-15 個複本。	`create-db-instance`
		`delete-db-instance`
		`describe-db-instances`
		`modify-db-instance`
		`describe-orderable-db-instance-options`
		`reboot-db-instance`

叢集磁碟區	虛擬資料庫儲存體磁碟區可以橫跨三個可用區域，且每個可用區域都有兩個叢集資料的複本。	N/A
主要執行個體	同時支援讀取和寫入操作，並執行叢集磁碟區所有資料修改。每個叢集具有一個主要執行個體。	N/A
複本執行個體	僅支援讀取操作。除了主要執行個體之外，每個 Amazon DocumentDB 叢集最多可以有 15 個複本執行個體。多個複本分配讀取工作負載。透過將複本放置在不同可用區域，您可提高資料庫可用性。	N/A

名詞	說明	API 操作 (動詞)
叢集端點	連接至叢集目前主要執行個體的 Amazon DocumentDB 叢集端點。每個 Amazon DocumentDB 叢集都有一個叢集端點和一個主要執行個體。	N/A
讀取器端點	連接至該叢集其中一個可用複本的 Amazon DocumentDB 叢集端點。每個 Amazon DocumentDB 叢集都有讀取器端點。如果有多個複本，讀取器端點會將每個連線請求導向其中一個 Amazon DocumentDB 複本。	N/A
執行個體端點	連接至特定執行個體之 Amazon DocumentDB 叢集中執行個體的端點。叢集中的每個執行個體，不論類型為何，都有自己的唯一執行個體端點。	N/A

Amazon DocumentDB 叢集設定

當您建立或修改叢集時，請務必了解哪些參數是不可變的以及哪些是在叢集建立後可修改的。下表列出叢集專屬的所有設定或參數。如資料表所指定的，有些可以修改，有些則不可以。

> **Note**
>
> 這些設定不應與 Amazon DocumentDB 叢集參數群組及其參數混淆。如需叢集參數群組的詳細資訊，請參閱 管理 Amazon DocumentDB 叢集參數群組。

參數	可修改	備註
DBClusterIdentifier	是	命名限制： • 長度為 【1-63】 個字母、數字或連字號。 • 第一個字元必須是字母。 • 不能以連字號結尾，或包含兩個連續連字號。 • 對於每個區域每個 Amazon RDS、Amazon Neptune 和 Amazon DocumentDB 的所有叢集 AWS 帳戶，必須是唯一的。
Engine	否	必須為 `docdb`。
BackupRetentionPeriod	是	必須介於 [1-35] 天。
DBClusterParameterGroupName	是	命名限制： • 長度為 【1—255】 個英數字元。 • 第一個字元必須是字母。 • 不能以連字號結尾，或包含兩個連續連字號。
DBSubnetGroupName	否	叢集建立之後，即無法修改叢集的子網路。
EngineVersion	否	值可以是 `5.0.0`（預設）`4.0.0`、 或 `3.6.0`。
KmsKeyId	否	如果您選擇加密叢集，則無法變更用來加密叢集的 AWS KMS 金鑰。
MasterUsername	否	叢集建立之後，即無法修改 `MasterUsername` 。 命名限制： • 長度為 【1-63】 個英數字元。 • 第一個字元必須是字母。 • 不能是資料庫引擎保留的字。

參數	可修改	備註
MasterUserPassword	是	限制條件： • 長度為 【8—100】 個可列印的 ASCII 字元。 • 可以使用下列項目以外的任何可列印 ASCII 字元： • **/** (正斜線) • **"** (雙引號) • **@** (@ 符號)
Port	是	連接埠號碼適用於在叢集中的所有執行個體。
PreferredBackupWindow	是	
PreferredMaintenanceWindow	是	
StorageEncrypted	否	如果您選擇加密叢集，就無法再解密。
StorageType	是	資料庫叢集的儲存類型：標準 (standard) 或 I/O 最佳化 (iopt1)。 預設：standard 您可以使用 CreateDBCluster 和 設定此參數ModifyDBCluster 。 如需詳細資訊，請參閱Amazon DocumentDB 叢集儲存組態。
Tags	是	
VpcSecurityGroupIds	否	建立叢集之後，您無法修改叢集所在的 VPC。

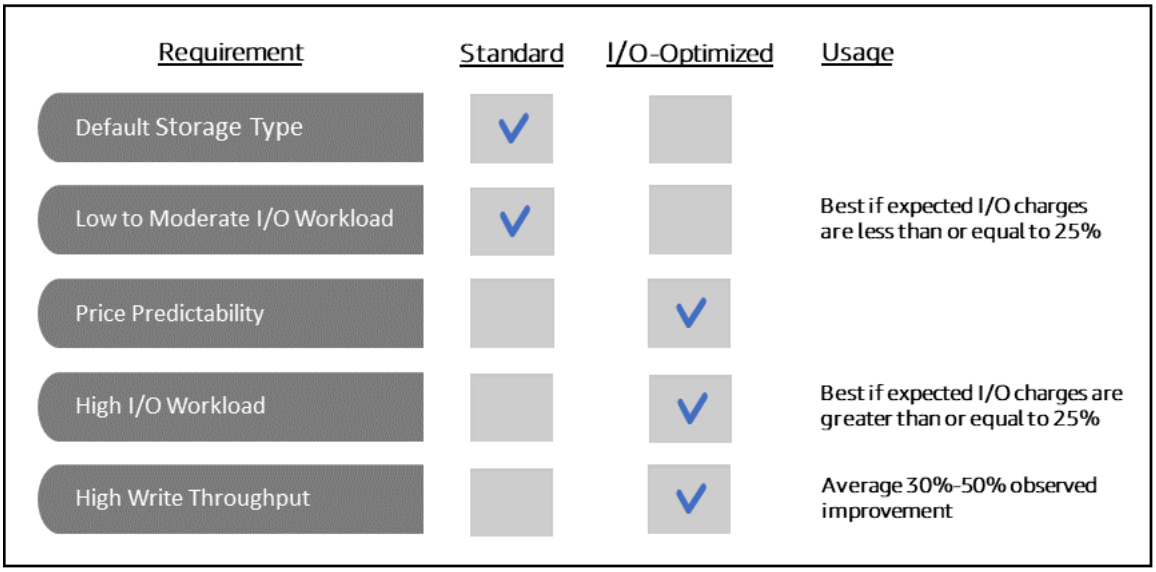
Amazon DocumentDB 叢集儲存組態

從 Amazon DocumentDB 5.0 開始，執行個體型叢集支援兩種儲存組態類型：

- Amazon DocumentDB 標準儲存：專為低到中等 I/O 耗用量的客戶而設計。如果您預期 I/O 成本低於 Amazon DocumentDB 叢集總數的 25%，則此選擇可能適合您。使用 Amazon DocumentDB 標準儲存體組態時，除了執行個體和儲存體費用之外，還會按pay-per-request I/O 收費。這表示根據用量，您的帳單可能因一個週期而異。此組態專為因應應用程式的 I/O 需求而量身打造。
- Amazon DocumentDB I/O 最佳化儲存：專為優先考慮價格可預測性或具有 I/O 密集型應用程式的客戶而設計。I/O 最佳化組態可為具有 I/O 密集型工作負載的客戶改善效能、提高輸送量並減少延遲。如果您預期 I/O 成本超過 Amazon DocumentDB 叢集總成本的 25%，此選項可提供增強的價格效能。透過 Amazon DocumentDB I/O 最佳化儲存組態，您不需要根據 I/O 操作付費，以確保每個計費週期的可預測成本。組態可穩定成本，同時改善效能。

您可以每 30 天將現有的資料庫叢集切換為 Amazon DocumentDB I/O 最佳化儲存。您可以隨時切換回 Amazon DocumentDB 標準儲存。修改儲存組態為 I/O 最佳化的下一個日期，可以使用 `describe-db-clusters`命令 AWS CLI 或透過叢集組態頁面 AWS 管理主控台 中的 來追蹤。

您可以建立新的資料庫叢集，包括 Amazon DocumentDB I/O 最佳化組態，或在 中按幾下滑鼠即可轉換現有的資料庫叢集AWS 管理主控台，在 AWS Command Line Interface (AWS CLI) 中或透過 AWS SDKs進行單一參數變更。在修改儲存體組態期間或之後，不需要停機或重新啟動執行個體。

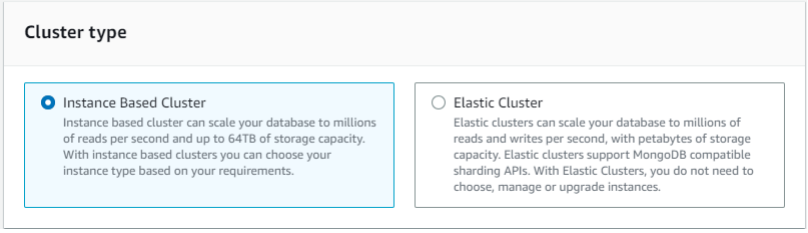


建立 I/O 最佳化叢集

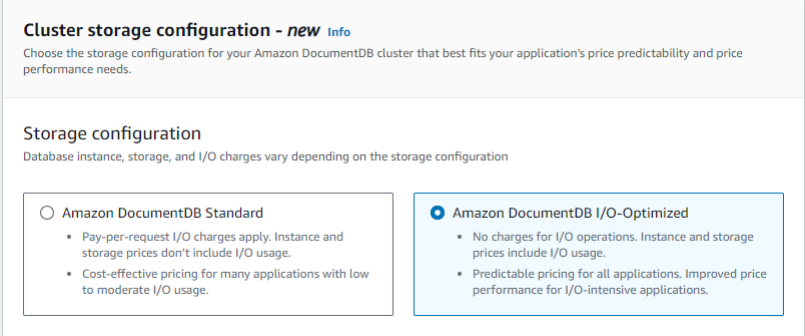
Using the AWS 管理主控台

若要使用 建立或修改 I/O 最佳化叢集 AWS 管理主控台：

1. 在 Amazon DocumentDB 管理主控台的叢集下，選擇建立或選取叢集，然後選擇動作，然後選擇修改。
2. 如果您要建立新的叢集，請務必在叢集類型區段中選擇執行個體型叢集 （這是預設選項）。



3. 在組態區段的叢集儲存組態下，選擇 Amazon DocumentDB I/O 最佳化。



4. 完成叢集建立或修改，然後選擇建立叢集或修改叢集。

   如需完整的建立叢集程序，請參閱 使用 建立叢集和主要執行個體 AWS 管理主控台。

   如需完整的修改叢集程序，請參閱 修改 Amazon DocumentDB 叢集。

Using the AWS CLI

若要使用 建立 I/O 最佳化叢集 AWS CLI：

在下列範例中，將每個*#########*取代為您自己的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb create-db-cluster \
      --db-cluster-identifier sample-cluster \
      --engine docdb \
```

```
    --engine-version 5.0.0 \
    --storage-type iopt1 \
    --deletion-protection \
    --master-username username \
    --master-user-password password
```

針對 Windows：

```
aws docdb create-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --engine docdb ^
    --engine-version 5.0.0 ^
    --storage-type iopt1 ^
    --deletion-protection ^
    --master-username username ^
    --master-user-password password
```

決定儲存組態的成本分析

使用 Amazon DocumentDB，您可以靈活地為每個資料庫叢集選擇儲存組態。為了在標準和 I/O 最佳化之間正確配置叢集，您可以逐個追蹤 Amazon DocumentDB 成本。若要這樣做，您可以將標籤新增至現有叢集、在AWS 帳單與成本管理 儀表板中啟用成本分配標記，以及分析 中指定叢集的成本AWS Cost Explorer Service。如需成本分析的資訊，請參閱我們的部落格使用成本分配標籤。

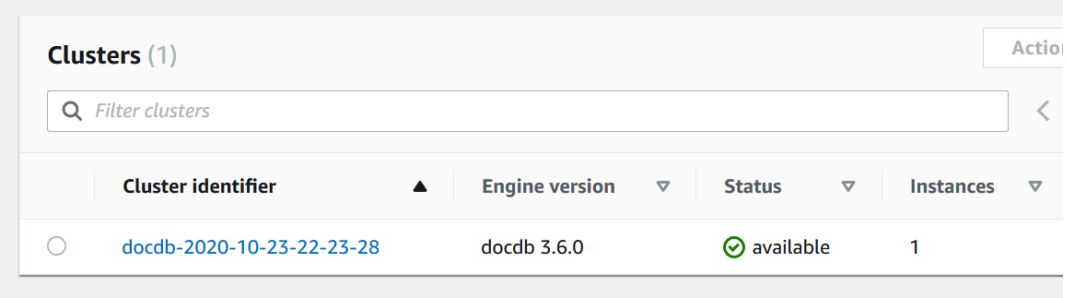
判斷叢集的狀態

您可以使用 AWS 管理主控台 或 來判斷叢集的狀態 AWS CLI。

Using the AWS 管理主控台

使用以下程序，使用 查看 Amazon DocumentDB 叢集的狀態 AWS 管理主控台

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。
3. 在 Cluster identifier (叢集識別碼) 欄中尋找您有興趣之叢集的名稱。然後，閱讀該列上的 Status (狀態) 欄找出叢集的狀態，如下所示。



Using the AWS CLI

使用 `describe-db-clusters`操作來查看使用 的 Amazon DocumentDB 叢集狀態 AWS CLI。

以下程式碼會尋找叢集 `sample-cluster` 的狀態。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster  \
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

針對 Windows：

```
aws docdb describe-db-clusters ^
    --db-cluster-identifier sample-cluster  ^
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

此操作的輸出將會如下所示 (JSON 格式)。

```
[
    [
        "sample-cluster",
        "available"
    ]
]
```

Amazon DocumentDB 叢集生命週期

Amazon DocumentDB 叢集的生命週期包括建立、描述、修改和刪除叢集。本節提供如何完成這些程序的相關資訊。

主題

- 建立 Amazon DocumentDB 叢集
- 描述 Amazon DocumentDB 叢集
- 修改 Amazon DocumentDB 叢集
- 判斷待定維護
- 對叢集的引擎版本執行修補程式更新
- 更新叢集的作業系統
- 停止和啟動 Amazon DocumentDB 叢集
- 刪除 Amazon DocumentDB 叢集

建立 Amazon DocumentDB 叢集

Amazon DocumentDB 叢集包含執行個體和代表叢集資料的叢集磁碟區。叢集磁碟區在三個可用區域間以六種方法複寫為單一虛擬磁碟區。叢集包含一個主要執行個體，而且最多可選擇性地包含 15 個複本執行個體。

下列各節說明如何使用 AWS 管理主控台 或 建立 Amazon DocumentDB 叢集 AWS CLI。您可以新增該叢集的其他複本執行個體。當您使用主控台建立 Amazon DocumentDB 叢集時，系統會自動為您同時建立主要執行個體。如果您使用 AWS CLI 建立 Amazon DocumentDB 叢集，則在叢集的狀態可用之後，您必須為該叢集建立主要執行個體。

先決條件

以下是建立 Amazon DocumentDB 叢集的先決條件。

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

   當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

VPC 先決條件

您只能在 Amazon Virtual Private Cloud (Amazon VPC) 中建立 Amazon DocumentDB 叢集。 Amazon Virtual Private Cloud 您的 Amazon VPC 必須在至少兩個可用區域中各有一個子網路，才能與 Amazon DocumentDB 叢集搭配使用。透過將叢集執行個體分散到可用區域，您可以確保在不太可能發生可用區域故障的情況下，叢集中有可用的執行個體。

子網路先決條件

建立 Amazon DocumentDB 叢集時，您必須選擇該 VPC 內的 VPC 和對應的子網路群組，才能啟動叢集。子網路決定可用區域，以及在該可用區域內您想要用來啟動執行個體的 IP 範圍。鑒於這項討論的目的，我們將交替使用「子網路」與「可用區域」這兩個名詞。子網路群組是一組具名的子網路 (或「可用區域」)。子網路群組允許您執行的操作是指定要用來啟動 Amazon DocumentDB 執行個體的可用區域。例如，在具有三個執行個體的叢集中，建議這些執行個體每一個在不同的可用區域佈建高可用性。因此，如果單一可用區域發生故障，它只會影響單一執行個體。

Amazon DocumentDB 執行個體目前最多可佈建在三個可用區域中。即使子網路群組有超過三個子網路，您仍只能使用其中三個子網路來建立 Amazon DocumentDB 叢集。因此，建議您在建立子網路群組時，只選擇您要部署執行個體的三個子網路。在美國東部（維吉尼亞北部），您的子網路群組可以有六個子網路（或可用區域）。不過，佈建 Amazon DocumentDB 叢集時，Amazon DocumentDB 會選擇其中三個可用區域來佈建執行個體。

例如，假設當您建立叢集時，Amazon DocumentDB 會選擇可用區域 {1A、1B 和 1C}。如果您嘗試在可用區域 {1D} 中建立執行個體，API 呼叫會失敗。不過，如果您選擇建立執行個體而不指定特定可用區域，則 Amazon DocumentDB 會代表您選擇可用區域。Amazon DocumentDB 使用演算法跨可用區域負載平衡執行個體，以協助您實現高可用性。例如，若佈建三個執行個體，預設會跨三個可用區域佈建執行個體，而不會全佈建在單一可用區域。

建議：

- 除非有特定的原因，否則一律使用三個子網路建立子網路群組。這樣做可協助確保具有三個或更多個執行個體的叢集能夠獲得更高的可用性，因為可以跨三個可用區域佈建執行個體。
- 一律將執行個體分散在多個可用區域以達到高可用性。請勿將叢集的所有執行個體放置在單一可用區域。
- 由於隨時都會發生容錯移轉事件，您不應假設主要執行個體或複本執行個體一律位在特定的可用區域。

其他先決條件

以下是建立 Amazon DocumentDB 叢集的一些額外先決條件：

- 如果您要 AWS 使用 AWS Identity and Access Management (IAM) 登入資料連線至 ，您的 IAM 帳戶必須具有授予執行 Amazon DocumentDB 操作所需許可的 IAM 政策。

  如果您使用 IAM 帳戶存取 Amazon DocumentDB 主控台，您必須先 AWS 管理主控台 使用 IAM 帳戶登入 。然後前往 Amazon DocumentDB 主控台，網址為 https：//https://console.aws.amazon.com/docdb。
- 如果您想要為叢集量身打造組態參數，則必須為叢集參數群組及參數群組指定必要的參數設定。如需建立或修改叢集參數群組或參數群組的相關資訊，請參閱管理 Amazon DocumentDB 叢集參數群組。
- 您必須決定要為叢集指定的 TCP/IP 連接埠號碼。某些公司的防火牆會封鎖與 Amazon DocumentDB 預設連接埠的連線。如果您公司的防火牆會封鎖預設連接埠，請為您的叢集選擇另一個連接埠。叢集中的所有執行個體都使用相同的連接埠。

使用 建立叢集和主要執行個體 AWS 管理主控台

下列程序說明如何使用 主控台啟動具有一或多個執行個體的 Amazon DocumentDB 叢集。

建立叢集：使用預設設定

使用 的預設設定建立具有執行個體的叢集 AWS 管理主控台

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/docdb：// 開啟 Amazon DocumentDB 主控台。
2. 如果您想要在美國東部 （維吉尼亞北部） 區域 AWS 區域 以外的 中建立叢集，請從主控台右上角的清單中選擇區域。
3. 在導覽窗格中，選擇 Clusters (叢集)，然後選擇 Create (建立)。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

4. 在建立 Amazon DocumentDB 叢集頁面上，完成組態窗格。

a. 叢集識別符 - 接受 Amazon DocumentDB 提供的名稱，或輸入叢集的名稱，例如 **`sample-cluster`**。

   叢集命名限制條件：

   - 長度為 【1-63】 個字母、數字或連字號。
   - 第一個字元必須是字母。
   - 不能以連字號結尾，或包含兩個連續連字號。
   - 對於每個區域每個 Amazon RDS、Neptune 和 Amazon DocumentDB 的所有叢集 AWS 帳戶，必須是唯一的。

b. 引擎版本 - 接受預設引擎版本 5.0.0，或選擇性地選擇 8.0.0、4.0.0 或 3.6.0。

c. 執行個體類別 - 接受預設的 `db.r5.large`，或從清單中選擇您想要的執行個體類別。

d. 執行個體數量 - 在清單中，選擇要使用此叢集建立的執行個體數量。第一個執行個體將是主要執行個體，而其他所有執行個體將是唯讀複本執行個體。如有必要，您可以稍後新增和刪除執行個體。根據預設，Amazon DocumentDB 叢集會啟動三個執行個體 （ 一個主要和兩個複本 ）。

5. 完成叢集儲存組態區段。

   選擇 Amazon DocumentDB Standard （ 預設 ） 或 Amazon DocumentDB I/O 最佳化。如需詳細資訊，請參閱Amazon DocumentDB 叢集儲存組態。

6. 完成 Authentication (認證) 窗格。

   a. 使用者名稱 - 輸入主要使用者的名稱。若要登入叢集，您必須使用主要使用者名稱。

      主要使用者命名限制：

      - 長度為 【1-63】 個英數字元。
      - 第一個字元必須是字母。
      - 不能是資料庫引擎保留的字。

   b. 選擇下列其中一個密碼選項：

      - Managed in AWS Secrets Manager - 如果您想要自動管理主要使用者密碼 AWS Secrets Manager ，請選擇此選項。

        如果您選擇此選項，請建立自己的金鑰或使用 Secrets Manager 建立的金鑰來設定 KMS 金鑰。

- 自我管理 - 如果您想要自我管理主要使用者密碼，請選擇此選項。如果您選擇此選項，請輸入主要使用者的密碼，然後確認。若要登入叢集，您必須使用主要使用者的密碼。

  密碼限制條件：

  - 長度為 [8-100] 個可列印的 ASCII 字元。
  - 可以使用下列項目以外的任何可列印 ASCII 字元：
    - **/** (正斜線)
    - **"** (雙引號)
    - **@** (@ 符號)

7. 在螢幕下方，選擇以下其中一項：

   - 若要立即建立叢集，請選擇 Create cluster (建立叢集)。
   - 若不建立叢集，請選擇 Cancel (取消)。
   - 若要在建立之前進一步設定叢集，請選擇 Show additional configurations (顯示其他組態)，然後繼續進行建立叢集：其他組態。

     Additional Configurations (其他組態) 部分涵蓋的組態如下所示：

     - 網路設定 - 預設為使用 `default` VPC 安全群組。
     - 叢集選項 - 預設為使用連接埠 27017 和預設參數群組。
     - Encryption - 預設為使用 `(default) aws/rds`金鑰啟用加密。

       > **Important**
       > 叢集加密後，就無法解密。

     - Backup - 預設為保留備份 1 天，並讓 Amazon DocumentDB 選擇備份時段。
     - 日誌匯出 - 預設為不將稽核日誌匯出至 CloudWatch Logs。
     - 維護 - 預設值是讓 Amazon DocumentDB 選擇維護時段。
     - 刪除保護 - 保護您的叢集免於意外刪除。使用主控台建立的叢集預設為已啟用。

     如果現在接受預設設定，您可以在稍後藉由修改叢集來將其大多數做變更。

8. 為叢集的安全群組啟用入站連線。

如果您未變更叢集的預設設定，請使用指定區域中預設 VPC 的預設安全群組來建立叢集。若要連線至 Amazon DocumentDB，您必須在叢集安全群組的連接埠 27017 （或您選擇的連接埠）上啟用傳入連線。

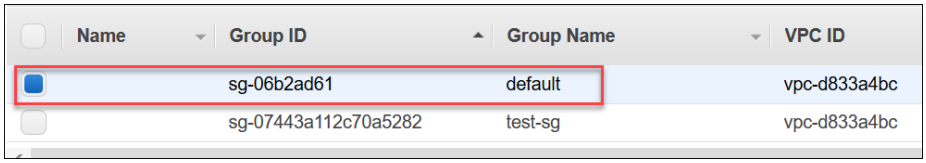
新增入站連線至叢集安全群組

a. 登入 AWS 管理主控台 ，並在 https://console.aws.amazon.com/ec2/ ：// 開啟 Amazon EC2 主控台。

b. 在主視窗的 Resources (資源) 區段中，選擇 Security groups (安全群組)。



c. 從安全群組清單中，找出您建立叢集時使用的安全群組 (最可能是預設安全群組)，並選擇安全群組名稱左側的方塊。



d. 在 Actions (動作) 功能表中，選擇 Edit inbound rules (編輯入站規則)，然後輸入規則限制。

   i. 類型 - 從清單中，選擇要向網路流量開啟的通訊協定。

   ii. Protocol - 從清單中，選擇通訊協定的類型。

   iii. 連接埠範圍 - 針對自訂規則，輸入連接埠號碼或連接埠範圍。請確定連接埠號碼或範圍包括您建立叢集時指定的連接埠 (預設：27107)。

   iv. 來源 - 指定可到達執行個體的流量。從清單中選擇流量來源。如果您選擇 Custom (自訂)，指定單一 IP 地址或是以 CIDR 表示法表示的 IP 地址範圍 (例如 203.0.113.5/32)。

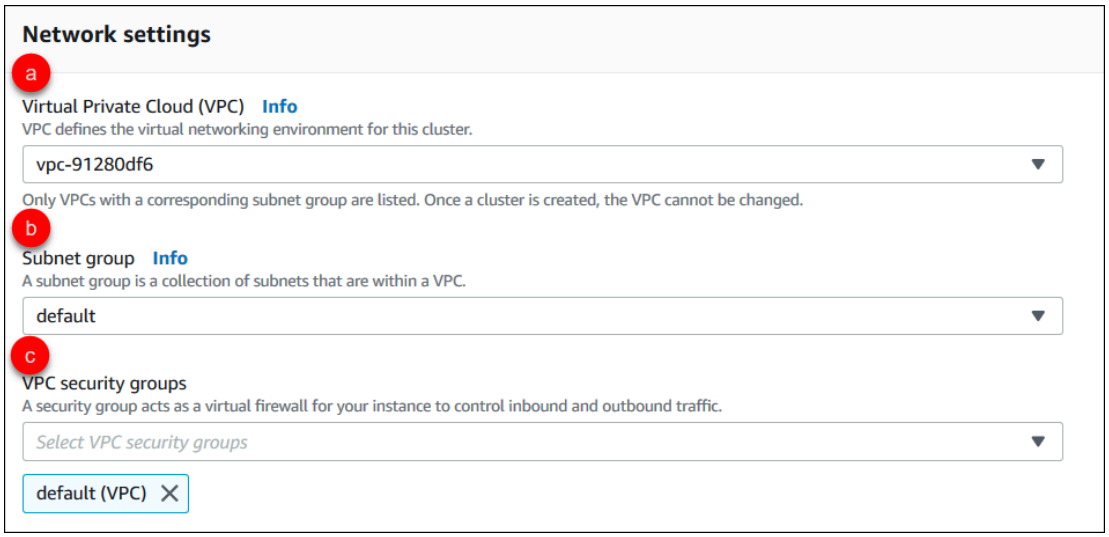
   v. 描述 - 輸入此規則的描述。

   vi. 完成建立規則後，選擇 Save (儲存)。

建立叢集：其他組態

如果您想要接受叢集的預設設定，您可以略過以下步驟，然後選擇 Create cluster (建立叢集)。

1. 完成 Network settings (網路設定) 窗格。



   a. Virtual Private Cloud (VPC)—在清單中，選擇要啟動此叢集的 Amazon VPC。

   b. 子網路群組 - 在清單中，選擇您要用於此叢集的子網路群組。

   c. VPC 安全群組 - 在清單中，選擇此叢集的 VPC 安全群組。

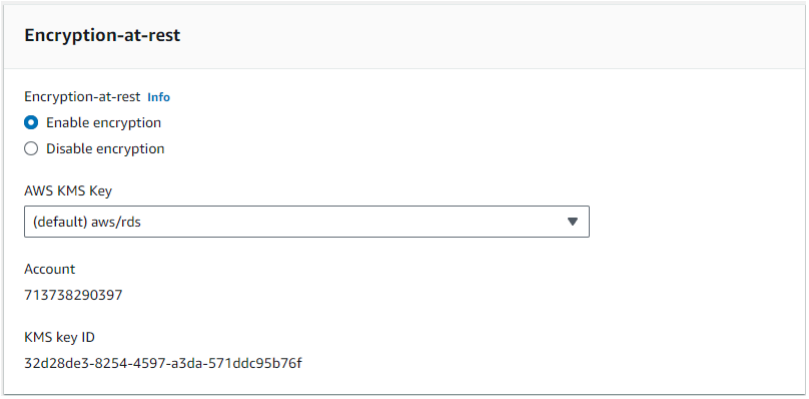
2. 完成 Cluster options (叢集選項) 窗格。



   a. 資料基礎連接埠 - 使用向上和向下箭頭來設定應用程式用來連線至執行個體的 TCP/IP 連接埠。

   b. 叢集參數群組 - 在參數群組清單中，選擇此叢集的叢集參數群組。

3. 完成 Encryption (加密) 窗格。



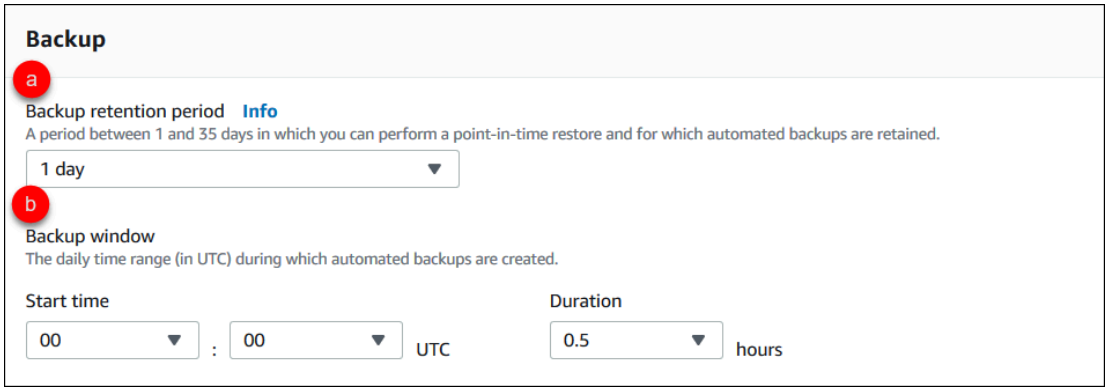
a. Encryption-at-rest - 選擇下列其中一項：

   - 啟用加密 - 預設。所有靜態資料已加密。如果您選擇加密資料，您無法復原此動作。
   - 停用加密 - 您的資料未加密。

b. AWS KMS 金鑰 - 只有在您加密資料時才能使用。在清單中，選擇要用於在此叢集中加密的金鑰。預設值為 `(default) aws/rds`。

   如果您選擇 Enter a key ARN (輸入金鑰 ARN)，您必須需要輸入金鑰的 Amazon Resource Name (ARN)。

4. 完成 Backup (加密) 窗格。



a. 備份保留期 - 在清單中，選擇在刪除叢集之前保留此叢集自動備份的天數。

b. 備份時段 - 設定 Amazon DocumentDB 要備份此叢集的每日時間和持續時間。

   i. 開始時間 - 在第一個清單中，選擇啟動自動備份的開始時間小時 (UTC)。在第二個清單中，選擇您要自動備份開始進行的時間 (分)。

   ii. 持續時間 - 在清單中，選擇要配置給建立自動備份的時數。

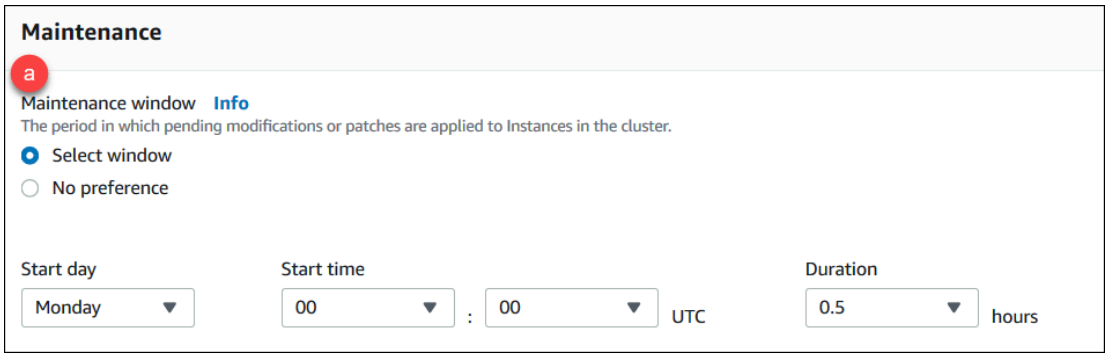
5. 選取您要匯出至 CloudWatch Logs 的日誌類型，以完成日誌匯出窗格。



- 稽核日誌 - 選取此選項可啟用將稽核日誌匯出至 Amazon CloudWatch Logs。如果您選取 Audit logs (稽核日誌)，則必須在叢集的自訂參數群組中啟用 `audit_logs`。如需詳細資訊，請參閱稽核 Amazon DocumentDB 事件。
- Profiler 日誌 - 選取此選項可啟用將操作 Profiler 日誌匯出至 Amazon CloudWatch Logs。如果您選取 Profiler logs (Profiler 日誌)，則還必須在叢集的自訂參數群組中修改下列參數：
  - `profiler`- 設定為 `enabled`。
  - `profiler_threshold_ms`- 將 設定為值`[0-INT_MAX]`，以設定分析操作的閾值。
  - `profiler_sampling_rate`- 將 設定為值`[0.0-1.0]`，將慢速操作的百分比設定為設定檔。

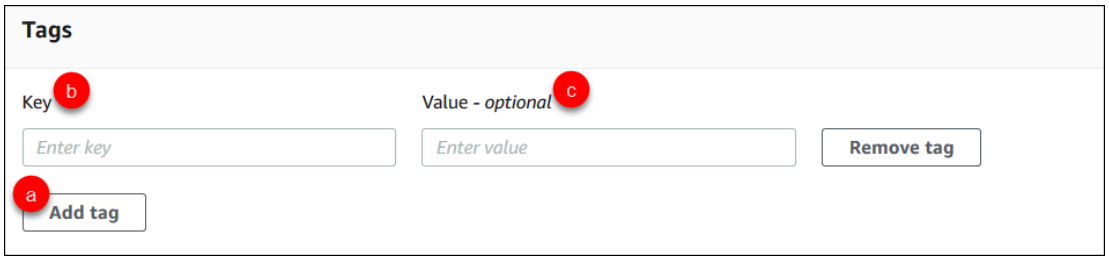
  如需詳細資訊，請參閱分析 Amazon DocumentDB 操作。

6. 完成 Maintenance (維護) 窗格。



- 選擇下列其中一項
  - 選取視窗 - 您可以指定星期幾、UTC 開始時間和持續時間，讓 Amazon DocumentDB 在您的叢集上執行維護。
    a. 開始日 - 在清單中，選擇星期幾以開始叢集維護。
    b. 開始時間 - 在清單中，選擇小時和分鐘 (UTC) 以開始維護。
    c. 持續時間 - 在清單中，選擇叢集維護的配置時間。如果無法在指定時間內完成維護作業，維護程序會持續超過指定的時間，直到完成為止。
  - 無偏好設定 - Amazon DocumentDB 會選擇星期幾、開始時間和持續時間來執行維護。

7. 如果您希望將一或多個標籤新增到此叢集，請完成 Tags (標籤) 窗格。



對於您要新增到叢集的每個標籤，請重複下列步驟。叢集上最多可以有 10 個。

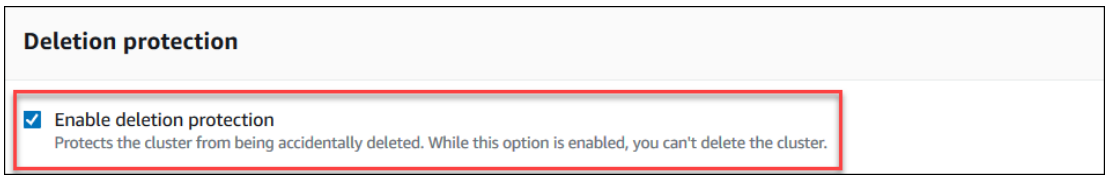
a. 選擇 Add tags (新增標籤)。

b. 輸入標籤的 Key (索引鍵)。

c. 選擇性輸入標籤的 Value (值)。

若要移除標籤，請選擇 Remove tag (移除標籤)。

8. 當您使用主控台建立叢集時，預設會啟用 Deletion Protection (刪除保護)。若要停用刪除保護，請清除 Enable deletion protection (啟用刪除保護)。若已啟用，刪除保護就會避免叢集遭到刪除。若要刪除已啟用刪除保護的叢集，您必須先修改叢集以停用刪除保護。



如需刪除保護的詳細資訊，請參閱 [刪除 Amazon DocumentDB 叢集]。

9. 若要建立叢集，請選擇 Create cluster (建立叢集)。否則，請選擇 Cancel (取消)。

使用 建立叢集 AWS CLI

下列程序說明如何使用 AWS CLI 啟動 Amazon DocumentDB 叢集並建立 Amazon DocumentDB 複本。

Parameters

- **`--db-cluster-identifier`** - 必要項目。識別此叢集的小寫字串。

叢集命名限制條件：

  - 長度為 【1–63】 個字母、數字或連字號。
  - 第一個字元必須是字母。
  - 不能以連字號結尾，或包含兩個連續連字號。
  - 對於每個區域每個 AWS 帳戶的所有叢集 （跨 Amazon RDS、Amazon Neptune 和 Amazon DocumentDB)，必須是唯一的。

- **`--engine`** - 必要項目。必須為 **`docdb`**。
- **`--deletion-protection | --no-deletion-protection`**- 選用。若已啟用刪除保護，它就會避免叢集遭到刪除。當您使用 時 AWS CLI，預設設定為停用刪除保護。

  如需刪除保護的詳細資訊，請參閱 刪除 Amazon DocumentDB 叢集。
- **`--storage-type standard | iopt1`**- 選用。預設：**`standard`**。叢集的儲存組態。有效值為 `standard`（標準） 或 `iopt1`(I/O 最佳化）。
- **`--master-username`** - 必要項目。用於驗證使用者的使用者名稱。

  主要使用者命名限制條件：

  - 長度為 [1-63] 英數字元。
  - 第一個字元必須是字母。
  - 不能是資料庫引擎保留的字。
- **`--master-user-password`**- 選用。用於驗證使用者的使用者密碼。

  主要密碼限制條件：

  - 長度為 [8-100] 個可列印的 ASCII 字元。
  - 可以使用下列項目以外的任何可列印 ASCII 字元：
    - **`/`** (正斜線)
    - **`"`** (雙引號)
    - **`@`** (@ 符號)
- **`--manage-master-user-password`**- 選用。Amazon DocumentDB 會產生主要使用者密碼，並在 Secrets Manager 的整個生命週期中進行管理。

如需其他參數，請參閱 CreateDBCluster。

使用 啟動 Amazon DocumentDB 叢集 AWS CLI

若要建立 Amazon DocumentDB 叢集，請呼叫 `create-db-cluster` AWS CLI。下列 AWS CLI 命令會建立名為 的 Amazon DocumentDB 叢集，`sample-cluster`並啟用刪除保護。如需刪除保護的詳細資訊，請參閱 刪除 Amazon DocumentDB 叢集。

此外， `--engine-version` 是選用參數，預設為最新的主要引擎版本。目前的預設引擎版本為 5.0.0（注意：Amazon DocumentDB 8.0 可供使用，但必須明確指定）。發行新的主要引擎版本時， 的預設引擎版本`--engine-version`將會更新，以反映最後一個主要引擎版本。因此，對於生產工作負載，特別是依賴指令碼、自動化或 CloudFormation 範本的工作負載，請將 明確指定`--engine-version`為預期的主要版本。

> **Note**
>
> 如果`vpc-security-group-id`未指定 `db-subnet-group-name`或 ，Amazon DocumentDB 將為指定區域使用預設子網路群組和 Amazon VPC 安全群組。

針對 Linux、macOS 或 Unix：

```
aws docdb create-db-cluster \
      --db-cluster-identifier sample-cluster \
      --engine docdb \
      --engine-version 5.0.0 \
      --deletion-protection \
      --master-username masteruser \
      --master-user-password password
```

針對 Windows：

```
aws docdb create-db-cluster ^
      --db-cluster-identifier sample-cluster ^
      --engine docdb ^
      --engine-version 5.0.0 ^
      --deletion-protection ^
      --master-username masteruser ^
      --master-user-password password
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "DBCluster": {
        "StorageEncrypted": false,
        "DBClusterMembers": [],
        "Engine": "docdb",
        "DeletionProtection" : "enabled",
        "ClusterCreateTime": "2018-11-26T17:15:19.885Z",
        "DBSubnetGroup": "default",
        "EngineVersion": "5.0.0",
        "MasterUsername": "masteruser",
        "BackupRetentionPeriod": 1,
        "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster",
        "DBClusterIdentifier": "sample-cluster",
        "MultiAZ": false,
        "DBClusterParameterGroup": "default.docdb5.0",
        "PreferredBackupWindow": "09:12-09:42",
        "DbClusterResourceId": "cluster-KQSGI4MHU4NTDDRVNLNTU7XVAY",
        "PreferredMaintenanceWindow": "tue:04:17-tue:04:47",
        "Port": 27017,
        "Status": "creating",
        "ReaderEndpoint": "sample-cluster.cluster-ro-sfcrlcjcoroz.us-
east-1.docdb.amazonaws.com",
        "AssociatedRoles": [],
        "HostedZoneId": "ZNKXTT8WH85VW",
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-77186e0d",
                "Status": "active"
            }
        ],
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1c",
            "us-east-1e"
        ],
        "Endpoint": "sample-cluster.cluster-sfcrlcjcoroz.us-east-1.docdb.amazonaws.com"
    }
}
```

建立叢集需要幾分鐘才能完成。您可以使用 AWS 管理主控台 或 AWS CLI 來監控叢集的狀態。如需詳細資訊，請參閱監控 Amazon DocumentDB 叢集的狀態。

> **Important**
>
> 當您使用 AWS CLI 建立 Amazon DocumentDB 叢集時，不會建立任何執行個體。因此，您必須明確建立主要執行個體，以及您需要的任何複本執行個體。您可以使用 主控台或 AWS CLI 來建立執行個體。如需詳細資訊，請參閱將 Amazon DocumentDB 執行個體新增至叢集。

如需詳細資訊，請參閱《Amazon DocumentDB API 參考》中的 CreateDBCluster。

描述 Amazon DocumentDB 叢集

您可以使用 Amazon DocumentDB 管理主控台或 AWS CLI 查看連線端點、安全群組、VPCs 和與 Amazon DocumentDB 叢集相關的參數群組等詳細資訊。

如需詳細資訊，請參閱下列內容：

- 監控 Amazon DocumentDB 叢集的狀態
- 尋找叢集的端點

Using the AWS 管理主控台

使用下列程序，使用主控台檢視指定 Amazon DocumentDB 叢集的詳細資訊。

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在叢集清單中，選擇想要檢視詳細資訊的叢集名稱。叢集的相關資訊分為下列群組：
   - 摘要 — 有關叢集的一般資訊，包括引擎版本、叢集狀態、待定維護及其參數群組的狀態。
   - 連線與安全性 — Connect 區段列出要使用 mongo shell 或 應用程式連線到此叢集的連線端點。Seucirty Groups (安全群組) 區段會列出與此叢集相關聯的安全群組，以及其 VPC ID 和描述。

- 組態 — 叢集詳細資訊區段列出叢集的詳細資訊，包括叢集的 Amazon Resource Name (ARN)、端點和參數群組。它也會列出叢集的備份資訊、維護詳細資訊，以及安全性和網路設定。Cluster instances (叢集執行個體) 區段會列出屬於您此叢集的執行個體，當中每個執行個體的角色和叢集參數群組狀態。
- 監控 — 此叢集的 Amazon CloudWatch Logs 指標。如需詳細資訊，請參閱使用 CloudWatch 監控 Amazon DocumentDB。
- 事件和標籤 — 最近事件區段列出此叢集的最近事件。Amazon DocumentDB 會保留與叢集、執行個體、快照、安全群組和叢集參數群組相關的事件記錄。此資訊包括與每個事件相關聯的日期、時間和訊息。Tags (標籤) 區段會列出連接至此叢集的標籤。

Using the AWS CLI

若要使用 檢視 Amazon DocumentDB 叢集的詳細資訊 AWS CLI，請使用 `describe-db-clusters`命令，如下列範例所示。如需詳細資訊，請參閱《Amazon DocumentDB 資源管理 API 參考》中的 DescribeDBClusters。

> **Note**
>
> 對於特定管理功能，例如叢集和執行個體生命週期管理，Amazon DocumentDB 會利用與 Amazon RDS 共用的操作技術。`filterName=engine,Values=docdb` 篩選條件參數只會傳回 Amazon DocumentDB 叢集。

Example

範例 1：列出所有 Amazon DocumentDB 叢集

下列 AWS CLI 程式碼列出區域中所有 Amazon DocumentDB 叢集的詳細資訊。

```
aws docdb describe-db-clusters --filter Name=engine,Values=docdb
```

此操作的輸出將會如下所示。

```
{
    "DBClusters": [
        {
            "AvailabilityZones": [
                "us-east-1c",
```

```
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-1",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        },
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-2",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        },
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-3",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        }
    ]
}
```

Example

範例 2：列出指定 Amazon DocumentDB 叢集的所有詳細資訊

下列 AWS CLI 程式碼列出叢集 的詳細資訊`sample-cluster`。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters \
   --filter Name=engine,Values=docdb \
   --db-cluster-identifier sample-cluster
```

針對 Windows：

```
aws docdb describe-db-clusters ^
   --filter Name=engine,Values=docdb ^
   --db-cluster-identifier sample-cluster
```

此操作的輸出將會如下所示。

```
{
    "DBClusters": [
        {
            "AllocatedStorage": 1,
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1a",
                "us-east-1d"
            ],
            "BackupRetentionPeriod": 2,
            "DBClusterIdentifier": "sample-cluster",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            "EarliestRestorableTime": "2023-11-07T22:34:08.148000+00:00",
            "Endpoint": "sample-cluster.node.us-east-1.amazon.com",
            "ReaderEndpoint": "sample-cluster.node.us-east-1.amazon.com",
            "MultiAZ": false,
            "Engine": "docdb",
            "EngineVersion": "5.0.0",
            "LatestRestorableTime": "2023-11-10T07:21:16.772000+00:00",
            "Port": 27017,
            "MasterUsername": "chimeraAdmin",
            "PreferredBackupWindow": "22:22-22:52",
            "PreferredMaintenanceWindow": "sun:03:01-sun:03:31",
            "ReadReplicaIdentifiers": [],
            "DBClusterMembers": [
                {
                    "DBInstanceIdentifier": "sample-instance-1",
```

```
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "PromotionTier": 1
                },
                {
                    "DBInstanceIdentifier": "sample-instance-2",
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "PromotionTier": 1
                },

            ],
            "VpcSecurityGroups": [
                {
                    "VpcSecurityGroupId": "sg-9084c2ec",
                    "Status": "active"
                }
            ],
            "HostedZoneId": "Z06853723JYKYBXTJ49RB",
            "StorageEncrypted": false,
            "DbClusterResourceId": "cluster-T4LGLANHVAPGQYYULWUDKLVQL4",
            "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:sample-
cluster",
            "AssociatedRoles": [],
            "IAMDatabaseAuthenticationEnabled": false,
            "ClusterCreateTime": "2023-11-06T18:05:41.568000+00:00",
            "EngineMode": "provisioned",
            "DeletionProtection": false,
            "HttpEndpointEnabled": false,
            "CopyTagsToSnapshot": false,
            "CrossAccountClone": false,
            "DomainMemberships": [],
            "TagList": [],
            "StorageType": "iopt1",
            "AutoMinorVersionUpgrade": false,
            "NetworkType": "IPV4",
            "IOOptimizedNextAllowedModificationTime":
 "2023-12-07T18:05:41.580000+00:00"
        }
    ]
}
```

Example

範例 3：列出 Amazon DocumentDB 叢集的特定詳細資訊

若要使用 列出叢集詳細資訊的子集 AWS CLI，請新增 `--query`，指定要列出`describe-db-clusters`操作的叢集成員。`--db-cluster-identifier` 參數是您想要顯示詳細資訊之特定叢集的識別碼。如需查詢的詳細資訊，請參閱AWS Command Line Interface 《 使用者指南》中的如何使用 `--query`選項篩選輸出。

下列範例列出 Amazon DocumentDB 叢集中的執行個體。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters \
    --filter Name=engine,Values=docdb \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterMembers]'
```

針對 Windows：

```
aws docdb describe-db-clusters ^
    --filter Name=engine,Values=docdb ^
    --db-cluster-identifier sample-cluster ^
    --query 'DBClusters[*].[DBClusterMembers]'
```

此操作的輸出將會如下所示。

```
[
    [
        [
            {
                "DBInstanceIdentifier": "sample-instance-1",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            },
            {
                "DBInstanceIdentifier": "sample-instance-2",
                "IsClusterWriter": false,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            }
```

```
        ]
    ]
]
```

修改 Amazon DocumentDB 叢集

若要修改叢集，叢集必須處於可用狀態。您無法修改已停止的叢集。如果叢集已停止，請先啟動叢集，等待叢集變成可用，然後進行所需的修改。如需詳細資訊，請參閱停止和啟動 Amazon DocumentDB 叢集。

Using the AWS 管理主控台

使用下列程序，使用主控台修改特定的 Amazon DocumentDB 叢集。

修改 Amazon DocumentDB 叢集

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

   > **Tip**
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 選擇叢集名稱左側的按鈕，以指定您要修改的叢集。
4. 選擇 Actions (動作)，然後選擇 Modify (修改)。
5. 在 Modify Cluster: <cluster-name> (修改叢集: <叢集-名稱>) 窗格中，進行您要的變更。您可以在以下領域進行變更：

   - 叢集規格 - 叢集的名稱、安全群組和登入資料管理。
   - 叢集儲存組態 - 叢集的資料儲存模式。選擇標準和 I/O 最佳化組態。
   - 叢集選項 - 叢集的連接埠和參數群組。
   - Backup - 叢集的備份保留期和備份時段。
   - 日誌匯出 - 啟用或停用匯出稽核或分析器日誌。
   - 維護 - 設定叢集的維護時段。

- 刪除保護 - 在叢集上啟用或停用刪除保護。預設會啟用刪除保護。

6. 完成後，請選擇 Continue (繼續) 以檢視變更的摘要。
7. 如果您滿意變更，可以選擇 Modify cluster (修改叢集) 來修改叢集。或者，您也可以選擇 Back (返回) 或 Cancel (取消)，分別編輯或取消變更。

套用您的變更需要幾分鐘的時間才會完成。您只能使用狀態為可用的叢集。您可以使用主控台或 AWS CLI監控叢集的狀態。如需詳細資訊，請參閱監控 Amazon DocumentDB 叢集的狀態。

Using the AWS CLI

使用 AWS CLI透過 `modify-db-cluster` 操作來修改指定的叢集。如需詳細資訊，請參閱《Amazon DocumentDB API 參考》中的 ModifyDBCluster。

Parameters

- **`--db-cluster-identifier`** - 必要項目。您要修改之 Amazon DocumentDB 叢集的識別符。
- **`--backup-retention-period`**- 選用。自動備份保留的天數。有效值為 1–35。
- **`--storage-type`**- 選用。叢集的儲存組態。有效值為 `standard`（標準） 或 `iopt1`(I/O 最佳化）。
- **`--db-cluster-parameter-group-name`**- 選用。要用於叢集的叢集參數群組名稱。
- **`--manage-master-user-password`**- 選用。Amazon DocumentDB 會產生主要使用者密碼，並在 Secrets Manager 的整個生命週期中進行管理。
- **`--rotate-master-user-password`**- 選用。Secrets Manager 會為現有秘密產生新的秘密版本。新版本的秘密包含新的主要使用者密碼。Amazon DocumentDB 會變更叢集的主要使用者密碼，以符合新秘密版本的密碼。

  輪換主要密碼時，您必須指定 **`--apply-immediately`**選項。
- **`--master-user-password`**- 選用。主要資料庫使用者的新密碼。

  密碼限制條件：
  - 長度為 【8—100】 個可列印的 ASCII 字元。
  - 可以使用下列項目以外的任何可列印 ASCII 字元：
    - **`/`** (正斜線)
    - **`"`** (雙引號)
    - **`@`** (@ 符號)

- **--new-db-cluster-identifier**- 選用。重新命名叢集時，叢集的新叢集識別碼。此值會以小寫字母字串的形式儲存。

  命名限制：

  - 長度為 【1-63】 個字母、數字或連字號。
  - 第一個字元必須是字母。
  - 不能以連字號結尾，或包含兩個連續連字號。
  - 對於每個區域每個 Amazon RDS、Amazon Neptune 和 Amazon DocumentDB 的所有叢集 AWS 帳戶，必須是唯一的。
- **--preferred-backup-window**- 選用。每日時間範圍，自動備份會在此期間建立，以國際標準時間 (UTC) 表示。
  - 格式：hh24:mm-hh24:mm
- **--preferred-maintenance-window**- 選用。系統維護可能發生的每週時間範圍，以 UTC 表示。
  - 格式：ddd:hh24:mm-ddd:hh24:mm
  - 有效天：Sun、Mon、Tue、Wed、Thu、Fri 和 Sat。
- **--deletion-protection** 或 **--no-deletion-protection**- 選用。是否應在此叢集上啟用刪除保護。在叢集修改成停用刪除保護前，刪除保護可避免意外刪除叢集。如需詳細資訊，請參閱刪除 Amazon DocumentDB 叢集。
- **--apply-immediately** 或 **--no-apply-immediately**— 用來立即--apply-immediately進行變更。使用 --no-apply-immediately 可在叢集下一次維護時段進行變更。

Example

以下程式碼會變更 sample-cluster 叢集的備份保留期。

針對 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster \
       --db-cluster-identifier sample-cluster \
       --apply-immediately \
       --backup-retention-period 7
```

針對 Windows：

```
aws docdb modify-db-cluster ^
       --db-cluster-identifier sample-cluster ^
       --apply-immediately ^
       --backup-retention-period 7
```

此操作的輸出將會如下所示。

```
{
    "DBCluster": {
        "BackupRetentionPeriod": 7,
        "DbClusterResourceId": "cluster-VDP53QEWST7YHM36TTXOPJT5YE",
        "Status": "available",
        "DBClusterMembers": [
            {
                "PromotionTier": 1,
                "DBClusterParameterGroupStatus": "in-sync",
                "DBInstanceIdentifier": "sample-cluster-instance",
                "IsClusterWriter": true
            }
        ],
        "ReadReplicaIdentifiers": [],
        "AvailabilityZones": [
            "us-east-1b",
            "us-east-1c",
            "us-east-1a"
        ],
        "ReaderEndpoint": "sample-cluster.cluster-ro-ctevjxdlur57.us-
east-1.rds.amazonaws.com",
        "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster",
        "PreferredMaintenanceWindow": "sat:09:51-sat:10:21",
        "EarliestRestorableTime": "2018-06-17T00:06:19.374Z",
        "StorageEncrypted": false,
        "MultiAZ": false,
        "AssociatedRoles": [],
        "MasterUsername": "<your-master-user-name>",
        "DBClusterIdentifier": "sample-cluster",
        "VpcSecurityGroups": [
            {
                "Status": "active",
                "VpcSecurityGroupId": "sg-77186e0d"
            }
        ],
        "HostedZoneId": "Z2SUY0A1719RZT",
```

```
        "LatestRestorableTime": "2018-06-18T21:17:05.737Z",
        "AllocatedStorage": 1,
        "Port": 27017,
        "Engine": "docdb",
        "DBClusterParameterGroup": "default.docdb3.4",
        "Endpoint": "sample-cluster.cluster-ctevjxdlur57.us-
east-1.rds.amazonaws.com",
        "DBSubnetGroup": "default",
        "PreferredBackupWindow": "00:00-00:30",
        "EngineVersion": "3.4",
        "ClusterCreateTime": "2018-06-06T19:25:47.991Z",
        "IAMDatabaseAuthenticationEnabled": false
    }
}
```

套用您的變更需要幾分鐘的時間才會完成。您只能使用狀態為可用的叢集。您可以使用主控台或 AWS CLI監控叢集的狀態。如需詳細資訊，請參閱監控 Amazon DocumentDB 叢集的狀態。

判斷待定維護

維護通常涉及對叢集中下列資源的更新：

- 叢集中執行個體的作業系統 (OS)
- 叢集的引擎版本

如需 Amazon DocumentDB 維護的詳細資訊，請參閱 維護 Amazon DocumentDB。

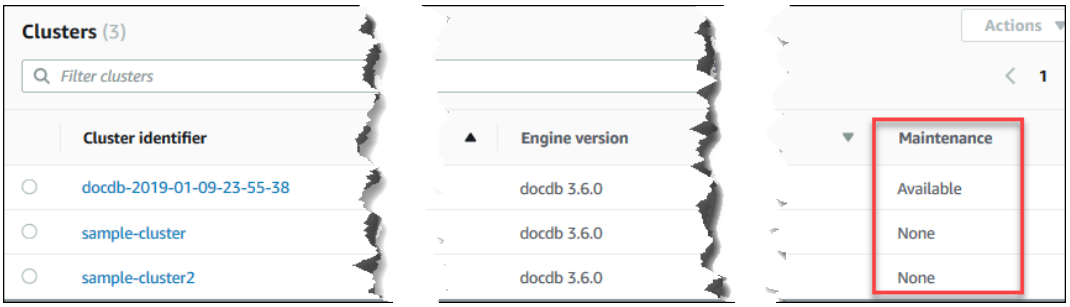
Using the AWS 管理主控台

您可以使用 AWS 管理主控台 來判斷叢集是否有待定的維護。

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

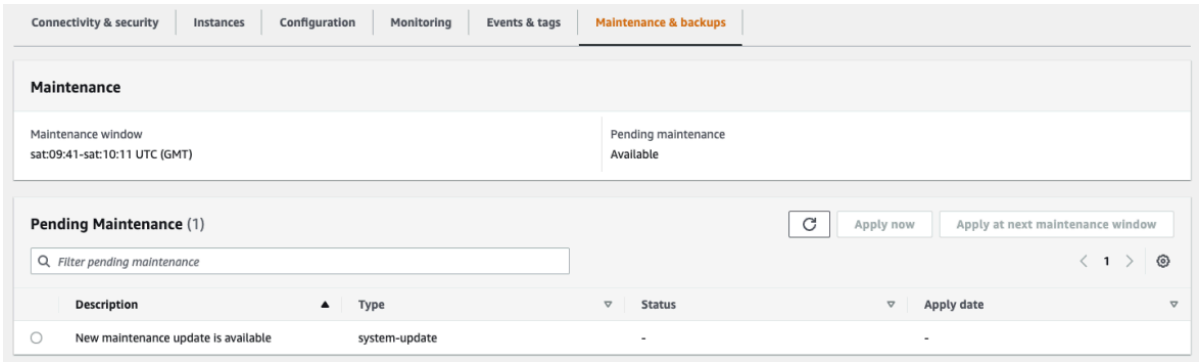
> **Tip**
>
> 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 找出 Maintenance (維護) 欄，判斷叢集是否有待定的維護。



無表示叢集正在執行最新的引擎修補程式版本，且其執行個體具有最新的作業系統更新。可用表示叢集有待定的維護，這可能表示需要引擎升級或作業系統更新。

4. 如果您的叢集有待定維護，您可以選擇叢集名稱並選取維護與備份索引標籤，以檢視待定的維護動作。待定維護區段會列出待定的維護動作。



Using the AWS CLI

您可以使用 AWS CLI 來判斷叢集是否具有最新的引擎版本和作業系統更新，方法是使用 `describe-pending-maintenance-actions`操作搭配下列參數。

Parameters

- **`--resource-identifier`**- 選用。資源的 ARN (叢集)。如果省略這個參數，則會列出所有叢集的待定維護動作。
- **`--region`**- 選用。您希望執行此操作的 AWS 區域，例如 `us-east-1`。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb describe-pending-maintenance-actions \
```

```
   --resource-identifier arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster \
   --region aa-example-1
```

針對 Windows：

```
aws docdb describe-pending-maintenance-actions ^
   --resource-identifier arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster ^
   --region aa-example-1
```

此操作的輸出將會如下所示。

```
{
    "PendingMaintenanceActions": [
        {
            "ResourceIdentifier": "arn:aws:rds:aa-
example-1:111122223333:cluster:sample-cluster",
            "PendingMaintenanceActionDetails": [
                {
                    "Description": "New feature",
                    "Action": "system-update",
                    "ForcedApplyDate": "2019-02-25T21:46:00Z",
                    "AutoAppliedAfterDate": "2019-02-25T07:41:00Z",
                    "CurrentApplyDate": "2019-02-25T07:41:00Z"
                }
            ]
        }
    ]
}
```

下列維護動作類型適用於 Amazon DocumentDB 叢集：

- `system-update` — 升級 Amazon DocumentDB 叢集的引擎修補程式。

  如果您的叢集有待定的引擎修補程式升級可用，請繼續執行 中的步驟對叢集的引擎版本執行修補程式更新。
- `os-upgrade` — 使用滾動升級更新 Amazon DocumentDB 叢集中所有資料庫執行個體的作業系統。

如需詳細資訊，請參閱Amazon DocumentDB 作業系統更新。

對叢集的引擎版本執行修補程式更新

在本節中，我們將說明如何使用 AWS 管理主控台 或 部署修補程式更新 AWS CLI。修補程式更新是相同引擎版本內的更新（例如，將 3.6 引擎版本更新為較新的 3.6 引擎版本）。您可以立即或在叢集的下一個維護時段進行更新。若要判斷您的引擎是否需要更新，請參閱 判斷待定維護。請注意，當您套用更新時，您的叢集將會遇到一些停機時間。

> **Note**
>
> 如果您嘗試從主要引擎版本升級到另一個版本，例如 3.6 到 5.0 或 5.0 到 8.0，請參閱 Amazon DocumentDB 就地主要版本升級或 使用 升級 Amazon DocumentDB 叢集 AWS Database Migration Service。

取得叢集引擎版本的最新修補程式更新有兩種組態需求：

- 叢集狀態必須為「可用」。
- 叢集必須執行舊版的引擎。

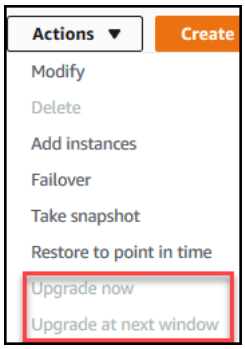
Using the AWS 管理主控台

下列程序會使用 主控台將修補程式更新套用至叢集的引擎版本。您可以選擇立即或在叢集的下一個維護時段進行更新。

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/docdb：// 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。在叢集清單中，選擇您想要升級之叢集的左側按鈕。叢集的狀態必須為「可用」。

> **Tip**
>
> 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 從 Actions (動作) 功能表選擇下列其中一個選項。您選擇的叢集必須不是執行最新的引擎版本，這些功能表選項才可供選取。



   - 立即升級 - 立即啟動升級程序。您的叢集會離線一段時間，讓叢集升級到最新的引擎版本。
   - 在下一個時段升級 - 在叢集的下一個維護時段啟動升級程序。您的叢集在升級到最新的引擎版本時，會離線一段時間。

4. 當確認視窗開啟時，請選擇下列其中一項：
   - 升級 - 根據上一個步驟中選擇的排程，將叢集升級至最新的引擎版本。
   - Cancel - 取消叢集的引擎升級，並繼續執行叢集目前的引擎版本。

Using the AWS CLI

您可以使用 AWS CLI 和 `apply-pending-maintenance-action`操作搭配下列參數，將修補程式更新套用至叢集。

Parameters

- **`--resource-identifier`** - 必要項目。您要升級之 Amazon DocumentDB 叢集的 ARN。
- **`--apply-action`** - 必要項目。使用 `system-update`將引擎修補程式更新套用至叢集。
- **`--opt-in-type`** - 必要項目。允許下列值。
  - `immediate`- 立即套用維護動作。
  - `next-maintenance`- 在下一個維護時段套用維護動作。
  - `undo-opt-in`- 取消任何現有的`next-maintenance`選擇加入請求。

Example

下列範例會將引擎修補程式更新套用至 `sample-cluster`。

針對 Linux、macOS 或 Unix：

```
aws docdb apply-pending-maintenance-action \
   --resource-identifier arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster \
   --apply-action system-update \
   --opt-in-type immediate
```

針對 Windows：

```
aws docdb apply-pending-maintenance-action ^
   --resource-identifier arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster ^
   --apply-action system-update ^
   --opt-in-type immediate
```

此操作的輸出如下所示：

```
{
    "ResourcePendingMaintenanceActions": {
        "ResourceIdentifier": "arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster",
        "PendingMaintenanceActionDetails": [
            {
                "CurrentApplyDate": "2019-02-20T20:57:06.904Z",
                "Description": "Bug fixes",
                "ForcedApplyDate": "2019-02-25T21:46:00Z",
                "OptInStatus": "immediate",
                "Action": "system-update",
                "AutoAppliedAfterDate": "2019-02-25T07:41:00Z"
            }
        ]
    }
}
```

更新叢集的作業系統

您可以使用叢集層級`os-upgrade`維護動作，將作業系統 (OS) 更新套用至叢集中的所有執行個體。Amazon DocumentDB 會以滾動方式更新執行個體，一次更新幾個執行個體，最後更新主要執行個體，將容錯移轉降至最低。更新會在叢集維護時段期間執行。執行個體收到作業系統更新後，其緩衝

區快取會開始空白，而該執行個體上的查詢可能會遇到更高的延遲，直到從儲存磁碟區重新填入工作集為止。

若要判斷叢集是否有待定的作業系統更新，請參閱 判斷待定維護。如需作業系統更新的完整詳細資訊，包括主控台和 CLI 程序，請參閱 Amazon DocumentDB 作業系統更新。

停止和啟動 Amazon DocumentDB 叢集

停止和啟動 Amazon DocumentDB 叢集可協助您管理開發和測試環境的成本。您不需要在每次使用 Amazon DocumentDB 時建立和刪除叢集和執行個體，您可以在不需要叢集中的所有執行個體時暫時停止。然後，您可以在恢復測試時再次啟動它們。

主題

- 停用和啟動叢集的概觀
- 您可以在已停止的叢集上執行的操作

停用和啟動叢集的概觀

在您不需要 Amazon DocumentDB 叢集的期間，您可以一次停止該叢集中的所有執行個體。一旦您需要叢集，即可隨時重新啟動它。啟動和停用可簡化用於下列操作之叢集的設定和卸除程序：開發、測試或不需要連續可用性的類似活動。您可以使用 AWS 管理主控台 或以單一動作 AWS CLI 來停止和啟動叢集，無論叢集中有多少執行個體。

雖然您的叢集已停止，但叢集儲存磁碟區保持不變。您只需支付指定的保留時段內儲存、手動快照和自動備份儲存的費用。您不需要支付任何執行個體時數的費用。Amazon DocumentDB 會在七天後自動啟動叢集，使其不會落後於任何必要的維護更新。當您的叢集在七天後啟動時，您將再度開始對叢集中的執行個體付費。當您的叢集停止時，您無法查詢儲存磁碟區，因為查詢時需要執行個體處於可用狀態。

當 Amazon DocumentDB 叢集停止時，叢集及其執行個體都無法以任何方式修改。這包括新增或移除執行個體、或刪除叢集。

Using the AWS 管理主控台

下列程序說明如何停止有一或多個執行個體處於可用狀態的叢集，或停止叢集。

停止或啟動 Amazon DocumentDB 叢集

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

2. 在導覽窗格中，選擇叢集。

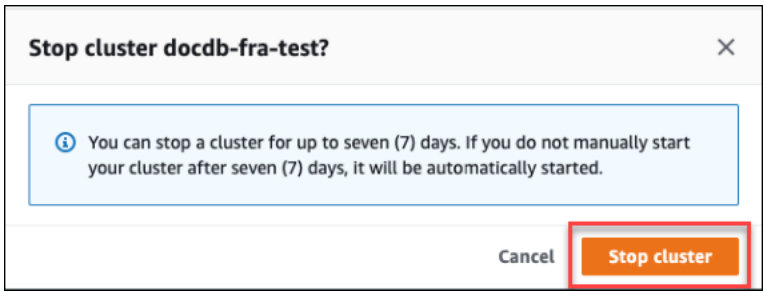
> **Tip**
>
> 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在叢集清單中，選擇您要停止或啟動之叢集名稱左側的按鈕。
4. 選擇 Actions (動作)，然後選擇您想要在叢集上執行的動作。
   - 如果您想要停止可用的叢集：
     a. 選擇停止。

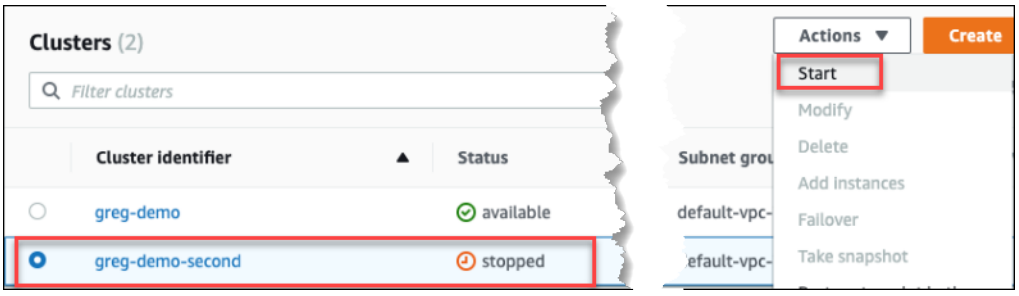


     為了避免啟動容錯移轉機制，停止操作會先停止複本執行個體，然後停止主要執行個體。

     b. 在確認對話方塊上，選擇 Stop cluster (停止叢集)，確認您要停止叢集，若要讓叢集持續執行，請選擇 Cancel (取消)。



   - 如果您想要啟動叢集，而叢集已停止，請選擇 Start (啟動)。



5. 監控叢集及其執行個體的狀態。如果您已啟動叢集，當叢集及其執行個體均可用時，您即可恢復使用叢集。如需詳細資訊，請參閱判斷叢集的狀態。



Using the AWS CLI

下列程式碼範例說明如何停止有一或多個執行個體處於可用狀態的叢集，或停止叢集。

若要使用 停止具有一或多個可用執行個體的叢集 AWS CLI，請使用 `stop-db-cluster`操作。若要啟動已停止的叢集，請使用 `start-db-cluster` 操作。這兩個操作均使用 `--db-cluster-identifier` 參數。

參數：

- **`--db-cluster-identifier`** - 必要項目。要停止或啟動的叢集名稱。

Example— 使用 停止叢集 AWS CLI

下列程式碼會停止叢集 `sample-cluster`。叢集必須有一或多個處於可用狀態的執行個體。

針對 Linux、macOS 或 Unix：

```
aws docdb stop-db-cluster \
   --db-cluster-identifier sample-cluster
```

針對 Windows：

```
aws docdb stop-db-cluster ^
   --db-cluster-identifier sample-cluster
```

Example— 使用 啟動叢集 AWS CLI

下列程式碼會啟動叢集 `sample-cluster`。叢集目前必須已停止。

針對 Linux、macOS 或 Unix：

```
aws docdb start-db-cluster \
   --db-cluster-identifier sample-cluster
```

針對 Windows：

```
aws docdb start-db-cluster ^
   --db-cluster-identifier sample-cluster
```

您可以在已停止的叢集上執行的操作

當 Amazon DocumentDB 叢集停止時，您可以在指定的自動備份保留時段內的任何時間點進行point-in-time還原。如需執行時間點還原的詳細資訊，請參閱還原至某個時間點。

您無法在叢集停止時修改 Amazon DocumentDB 叢集或其任何執行個體的組態。您也無法新增或移除叢集中的執行個體，或如果叢集仍有任何相關聯的執行個體，則無法刪除此叢集。您必須先啟動叢集，然後才能執行任何這類管理動作。

Amazon DocumentDB 只會在叢集再次啟動後，才會將任何排定的維護套用至已停止的叢集。七天後，Amazon DocumentDB 會自動啟動已停止的叢集，使其不會落後維護狀態太久。當叢集重新啟動時，您將再度開始對叢集中的執行個體付費。

當叢集停止時，Amazon DocumentDB 不會執行任何自動備份，也不會延長備份保留期。

刪除 Amazon DocumentDB 叢集

您可以使用 AWS 管理主控台 或 刪除 Amazon DocumentDB 叢集 AWS CLI。若要刪除叢集，叢集必須處於可用狀態，且不得有任何與叢集相關聯的執行個體。如果叢集已停止，請先啟動叢集，等待叢集變成可用，然後刪除叢集。如需詳細資訊，請參閱停止和啟動 Amazon DocumentDB 叢集。

刪除保護

為了避免您的叢集遭到意外刪除，您可以啟用刪除保護。當您使用主控台建立叢集時，預設會啟用刪除保護。不過，如果您使用 AWS CLI建立叢集，預設會停用刪除保護。

無論您使用主控台或 執行刪除操作，Amazon DocumentDB 都會對叢集強制執行刪除保護 AWS CLI。如果已啟用刪除保護，您就無法刪除叢集。若要刪除已啟用刪除保護的叢集，您必須先修改叢集並停用刪除保護。

使用叢集已啟用刪除保護的主控台時，您就無法刪除叢集的最後一個執行個體，因為這麼做也會刪除叢集。您可以使用 AWS CLI來刪除受刪除保護叢集的最後一個執行個體。不過，叢集本身仍會存在，而

您的資料會保留下來。您可為叢集建立新的執行個體，以便存取資料。如需啟用和停用刪除保護的詳細資訊，請參閱：

- 建立 Amazon DocumentDB 叢集
- 修改 Amazon DocumentDB 叢集

Using the AWS 管理主控台

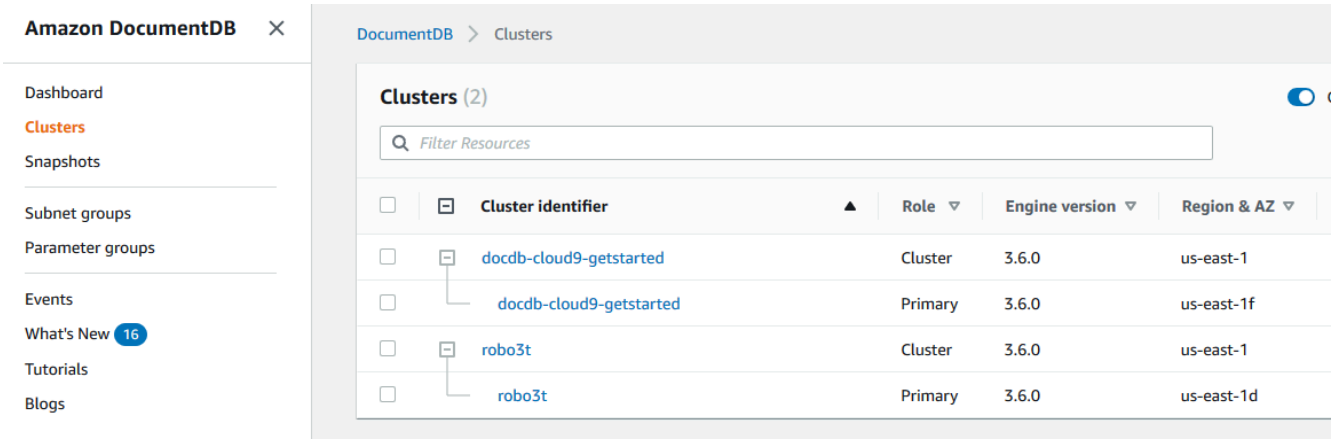
若要使用 刪除叢集 AWS 管理主控台，必須停用刪除保護。

判斷叢集是否已啟用刪除保護：

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

> **Tip**
>
> 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 請注意，在叢集導覽方塊中，叢集識別符欄會顯示叢集和執行個體。執行個體會列在叢集下方，類似於下列螢幕擷取畫面。



4. 選擇叢集的名稱，然後選取 Configuration (組態) 索引標籤。在 Cluster details (叢集詳細資訊) 區段中，找出 Deletion protection (刪除保護)。如果已啟用刪除保護，請修改叢集以停用刪除保護。如需修改叢集的詳細資訊，請參閱修改 Amazon DocumentDB 叢集。

   停用 Deletion protection (刪除保護) 之後，您就可以刪除叢集。

刪除叢集：

1. 在導覽窗格中，選擇叢集。
2. 透過檢查下列執行個體的叢集識別符欄，判斷叢集是否有任何執行個體。您必須先刪除叢集的所有執行個體，才能刪除叢集。如需詳細資訊，請參閱刪除 Amazon DocumentDB 執行個體。
3. 根據您的叢集是否有任何執行個體，執行下列其中一個步驟。
   - 如果叢集沒有執行個體，請選取叢集名稱左側的按鈕，然後選擇 Actions (動作)。從下拉式選單中選擇 Detete (刪除)。完成 Delete <cluster-name> (刪除 <cluster-name>) 對話方塊，然後選擇 Delete (刪除)。
   - 如果叢集有一或多個執行個體，請執行下列動作：
     a. 在導覽窗格中，選擇叢集。
     b. 選取叢集名稱左側的核取方塊，以刪除每個叢集的執行個體。選取 Actions (動作)，然後選擇 Delete (刪除)。完成 Delete <cluster-name> (刪除 <cluster-name>) 對話方塊，然後選擇 Delete (刪除)。

        當您刪除最後一個執行個體時，也會刪除叢集。如需刪除執行個體的詳細資訊，請參閱 刪除 Amazon DocumentDB 執行個體。

刪除叢集需要幾分鐘的時間。若要監控叢集的狀態，請參閱監控 Amazon DocumentDB 叢集的狀態。

Using the AWS CLI

您無法刪除與任何執行個體相關聯的叢集。若要判斷哪些執行個體與您的叢集相關聯，請執行 `describe-db-clusters` 命令並刪除叢集的所有執行個體。然後視需要停用叢集上的刪除保護，最後再刪除叢集。

1. 首先，刪除叢集的所有執行個體。

   若要判斷需要刪除哪些執行個體，請執行下列命令。

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].
[DBClusterIdentifier,DBClusterMembers[*].DBInstanceIdentifier]'
```

   此操作的輸出將會如下所示 (JSON 格式)。

```
[
    [
        "sample-cluster",
        [
            "sample-instance-1",
            "sample-instance-2"
        ]
    ]
]
```

如果您要刪除的叢集中有任何執行個體，請刪除它們，如下所示。

```
aws docdb delete-db-instance \
    --db-instance-identifier sample-instance
```

2. 第二，停用刪除保護。

使用 AWS CLI 刪除叢集的所有執行個體並不會刪除叢集。您也必須刪除叢集，但只有已停用刪除保護，您才能這麼做。

若要判斷叢集是否已啟用刪除保護，請執行下列命令。

> **Tip**
>
> 若要查看所有 Amazon DocumentDB 叢集的刪除保護狀態，請省略 `--db-cluster-identifier` 參數。

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterIdentifier,DeletionProtection]'
```

此操作的輸出將會如下所示。

```
[
    [
        "sample-cluster",
        "true"
```

```
    ]
]
```

如果已啟用刪除保護，請修改叢集並停用刪除保護。若要在叢集上停用刪除保護，請執行下列命令。

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --no-deletion-protection \
    --apply-immediately
```

3. 最後，刪除叢集。

停用刪除保護之後，您就可以刪除叢集。若要刪除叢集，請使用 delete-db-cluster 操作搭配下列參數。

- **--db-cluster-identifier** - 必要項目。您要刪除之叢集的識別碼。
- **--final-db-snapshot-identifier**- 選用。如果您要建立最終快照，則必須包含此參數及最終快照的名稱。您必須包含 --final-db-snapshot-identifier 或 --skip-final-snapshot。

  命名限制：

  - 長度為 【1-63】 個字母、數字或連字號。
  - 第一個字元必須是字母。
  - 不能以連字號結尾，或包含兩個連續連字號。
  - 對於每個區域每個 Amazon RDS、Amazon Neptune 和 Amazon DocumentDB 的所有叢集 AWS 帳戶，必須是唯一的。
- **--skip-final-snapshot**- 選用。若您刪除叢集前不要建立最後的快照，才使用此參數。預設設定是建立最後的快照。您必須包含 --final-db-snapshot-identifier 或 --skip-final-snapshot。

下列 AWS CLI 程式碼會刪除sample-cluster具有最終快照的叢集。如有任何與叢集相關聯的執行個體，或已啟用刪除保護，操作就會失敗。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --final-db-snapshot-identifier sample-cluster-final-snapshot
```

針對 Windows：

```
aws docdb delete-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --final-db-snapshot-identifier sample-cluster-final-snapshot
```

Example

下列 AWS CLI 程式碼會刪除叢集，`sample-cluster`而不拍攝最終快照。

針對 Linux、macOS 或 Unix：

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --skip-final-snapshot
```

針對 Windows：

```
aws docdb delete-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --skip-final-snapshot
```

`delete-db-cluster` 操作的輸出會是您要刪除的叢集。

刪除叢集需要幾分鐘的時間。若要監控叢集的狀態，請參閱監控叢集的狀態。

擴展 Amazon DocumentDB 叢集

Amazon DocumentDB 可讓您根據您的需求擴展叢集中的儲存和運算。本節說明如何使用儲存體擴展、執行個體擴展和讀取擴展來管理 Amazon DocumentDB 叢集和執行個體的效能和擴展。

主題

- 儲存體擴展
- 執行個體擴展
- 讀取擴展
- 寫入擴展

儲存體擴展

Amazon DocumentDB 儲存會自動隨叢集磁碟區中的資料擴展。隨著資料成長，您的叢集磁碟區儲存體會以 10 GiB 遞增，引擎版本 8.0 及更高版本最多可增加 256 TiB （舊版引擎為 128 TiB)。

執行個體擴展

您可以修改叢集中每個執行個體的執行個體類別，視需要擴展 Amazon DocumentDB 叢集。Amazon DocumentDB 支援針對 Amazon DocumentDB 最佳化的數個執行個體類別。

如需詳細資訊，請參閱修改 Amazon DocumentDB 執行個體。

讀取擴展

您可以在叢集中建立最多 15 個 Amazon DocumentDB 複本，以達成 Amazon DocumentDB 叢集的讀取擴展。每個 Amazon DocumentDB 複本會從叢集磁碟區傳回相同的資料，複本延遲最少，通常在主要執行個體寫入更新後不到 100 毫秒。隨著讀取流量的增加，您可以建立額外的 Amazon DocumentDB 複本，並直接與其連線，以分配叢集的讀取負載。Amazon DocumentDB 複本不必與主要執行個體具有相同的執行個體類別。

如需詳細資訊，請參閱將 Amazon DocumentDB 執行個體新增至叢集。

若要使用 Amazon DocumentDB 讀取擴展，請以複本集的形式連接至叢集，並使用驅動程式的內建讀取偏好設定功能將讀取分散至複本執行個體。如需詳細資訊，請參閱以複本集的形式連線至 Amazon DocumentDB。

寫入擴展

您可以增加叢集主要執行個體的大小，來擴展 Amazon DocumentDB 叢集的寫入容量。本節提供兩種方法，可根據您的需求調整叢集的主要執行個體。第一個選項旨在將應用程式影響降至最低，但需要更多步驟才能完成。第二個選項針對簡易性進行最佳化，因為它的步驟較少，但會對您的應用程式產生更多潛在的折衷影響。

視您的應用程式而定，您可以選擇最適合您的以下方法。如需可用執行個體大小和成本的詳細資訊，請參閱 Amazon DocumentDB 定價頁面。

1. 針對高可用性和效能進行最佳化：如果您以複本集模式（建議）連線到叢集，您可以使用下列程序，在擴展主要執行個體時將對應用程式的影響降至最低。此方法可將影響降至最低，因為它使叢集維持在高可用性或高於高可用性的水準，而讀取擴展目標會作為執行個體新增至叢集，而不是就地更新。

   a. 將一或多個較大執行個體類型的複本新增至叢集 (請參閱???)。確保所有複本都與主要複本具有相同或更大的執行個體類型。這樣可以避免不小心降低寫入效能，從容錯移轉到較小的執行個體類型。在大多數情況下，這表示暫時將叢集中的執行個體數量加倍，然後在擴展完成後移除較小的複本。

   b. 將所有新複本上的容錯移轉層設定為優先順序為零，確保較小執行個體類型的複本具有最高的容錯移轉優先順序。如需詳細資訊，請參閱???。

   c. 初始化手動容錯移轉，將其中一個新複本升級為主要執行個體。如需詳細資訊，請參閱???。

      > **Note**
      >
      > 這將導致您的叢集停機約 30 秒。請妥善規劃。

   d. 從叢集中移除小於新主要執行個體類型的所有複本。

   e. 將所有執行個體的容錯移轉層設定回相同的優先順序 (通常，這表示會將它們設定回 1)。

   舉例來說，假設您的叢集目前包含三個 `r5.large` 執行個體 (一個主要和兩個複本)，而您想要擴展到 `r5.xlarge` 執行個體類型。若要執行這項操作，您必須先將三個 `r5.xlarge` 複本執行個體新增至叢集，然後將新 `r5.xlarge` 複本的容錯移轉層設定為零。接下來，您將啟動手動容錯移轉 (您的應用程式將停機約 30 秒)。容錯移轉完成後，您會從叢集中移除所有三個 `r5.large` 執行個體，使叢集擴展至 `r5.xlarge` 執行個體。

   為了協助最佳化成本，Amazon DocumentDB 執行個體會以一秒遞增計費，在建立、修改或刪除執行個體等計費狀態變更之後，最低收費為 10 分鐘。如需詳細資訊，請參閱最佳實務文件中的成本最佳化。

2. 簡化最佳化 — 此方法可簡化最佳化。它不會擴展和收縮叢集，但可能會暫時降低您的讀取容量。

   變更複本的執行個體類別可能會導致該執行個體在幾秒鐘到不到 30 秒的短時間內沒有提供請求。如果您以複本集模式（建議）連接到叢集，則這會在擴展操作期間將讀取容量減少一個複本（例如，3 節點叢集的 66% 容量，或 4 節點叢集的 75% 容量等）。

a. 擴展叢集中的其中一個複本執行個體。如需詳細資訊，請參閱管理執行個體類別。

b. 等待執行個體可用（請參閱 監控 Amazon DocumentDB 執行個體的狀態)。

> **Note**
>
> 這將導致您的叢集停機約 30 秒。請妥善規劃。

c. 繼續執行步驟 1 和 2，直到已逐一擴展所有複本執行個體。

d. 啟動手動容錯移轉。這會將其中一個複本提升為主要執行個體。如需詳細資訊，請參閱Amazon DocumentDB 容錯移轉。

> **Note**
>
> 這將對您的叢集產生長達 30 秒的停機時間，但通常需要的時間比這更短。請妥善規劃。

e. 擴展先前的主要（現為複本）執行個體。

複製 Amazon DocumentDB 叢集的磁碟區

透過使用 Amazon DocumentDB 複製，您可以建立使用相同 Amazon DocumentDB 叢集磁碟區且具有與原始叢集相同資料的新叢集。該程序旨在快速且具有成本效益。與其相關聯的資料磁碟區新叢集稱為複製。與使用不同的技術 (如還原快照) 以物理方式複製資料相比，建立複製更快也更節省空間。

Amazon DocumentDB 支援從佈建的 Amazon DocumentDB 叢集建立 Amazon DocumentDB 佈建複製。當您使用與來源不同的部署組態建立複製時，會使用來源的最新版本 Amazon DocumentDB 引擎建立複製。

當您從 Amazon DocumentDB 叢集建立複製時，複製會在 AWS 您的帳戶中建立，也就是擁有來源 Amazon DocumentDB 叢集的相同帳戶。

主題

- Amazon DocumentDB 複製概觀
- Amazon DocumentDB 複製的使用注意事項
- Amazon DocumentDB 複製的運作方式
- 建立 Amazon DocumentDB 複製

Amazon DocumentDB 複製概觀

Amazon DocumentDB copy-on-write通訊協定來建立複製。此機制會使用最少的其他空間來建立初始複製。第一次建立複製時，Amazon DocumentDB 會保留來源資料庫叢集和新（複製）Amazon DocumentDB 叢集所使用的單一資料複本。只有當來源 Amazon DocumentDB 叢集或 Amazon DocumentDB 叢集複製對資料 (Amazon DocumentDB 儲存磁碟區）進行變更時，才會配置其他儲存體。若要進一步了解寫入時複製通訊協定，請參閱 Amazon DocumentDB 複製的運作方式。

Amazon DocumentDB 複製特別適用於使用生產資料快速設定測試環境，而不會造成資料損毀的風險。您可以為許多類型的應用程式使用複製，例如以下類型：

- 潛在變更實驗 (例如結構描述變更和參數群組變更)，以評估所有影響。
- 執行工作負載密集的操作，例如在複製上匯出資料或執行分析查詢。
- 為了開發、測試或其他用途建立生產資料庫叢集的副本。

您可以從相同的 Amazon DocumentDB 叢集建立多個複製。您也可以從另一個複製建立多個複製。

建立 Amazon DocumentDB 複製之後，您可以設定與來源 Amazon DocumentDB 叢集不同的 Amazon DocumentDB 執行個體。例如，您可能不需要為了開發目的而複製，才能符合與來源生產 Amazon DocumentDB 叢集相同的高可用性要求。在這種情況下，您可以使用單一 Amazon DocumentDB 執行個體設定複製，而不是 Amazon DocumentDB 叢集使用的多個資料庫執行個體。

在您完成使用複製來進行測試、開發或其他用途時，便可將其刪除。

Amazon DocumentDB 複製的使用注意事項

Amazon DocumentDB； 複製目前有下列限制：

- 您可以根據需要建立數量不拘的複製項，最多可達到 AWS 區域中允許的資料庫叢集數量上限。但在建立 15 個複製項後，下一個複製項是完整複本。複製操作的作用類似於時間點復原。
- 您無法在與來源 Amazon DocumentDB 叢集不同的 AWS 區域中建立複製。
- 您無法從沒有資料庫執行個體的 Amazon DocumentDB 叢集建立複製。您只能複製至少具有一個資料庫執行個體的 Amazon DocumentDB 叢集。
- 您可以在與 Amazon DocumentDB 叢集不同的虛擬私有雲端 (VPC) 中建立複製。不過，VPC 中的子網路必須映射至相同的可用區域。

Amazon DocumentDB 複製的運作方式

Amazon DocumentDB 複製可在 Amazon DocumentDB 叢集的儲存層運作。它使用copy-on-write通訊協定，在支援 Amazon DocumentDB 儲存磁碟區的基礎耐久媒體上既快速又節省空間。您可以在 中進一步了解 Amazon DocumentDB 叢集磁碟區管理 Amazon DocumentDB 叢集。

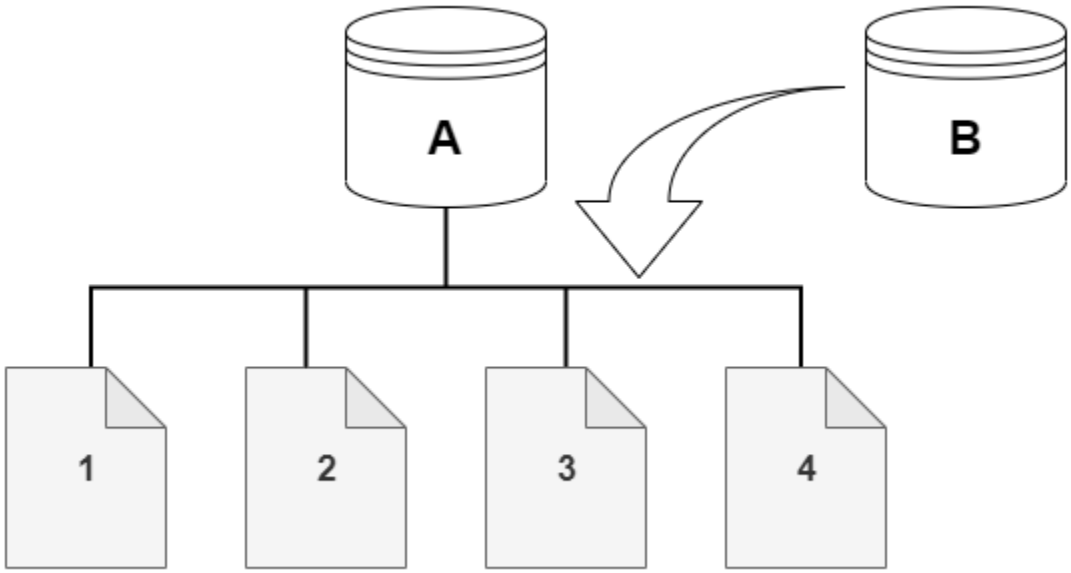
主題

- 了解寫入時複製通訊協定
- 刪除來源叢集磁碟區

了解寫入時複製通訊協定

Amazon DocumentDB 叢集會將資料存放在基礎 Amazon DocumentDB 儲存磁碟區中的頁面中。

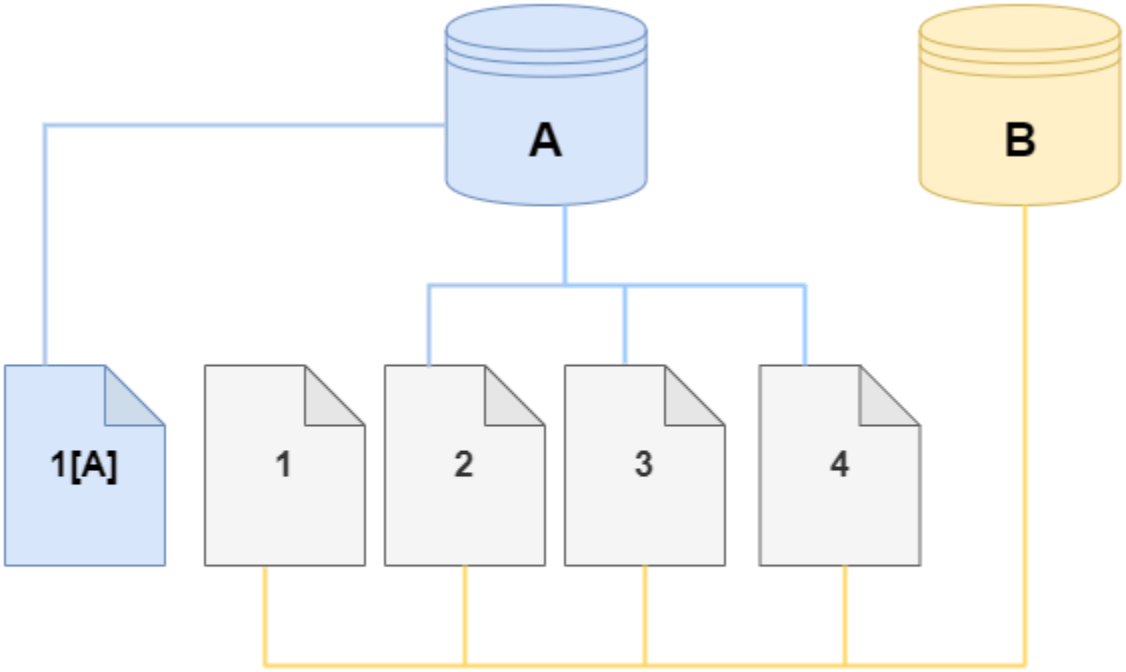
例如，在下圖中，您可以找到具有四個資料頁面 1、2、3 和 4 的 Amazon DocumentDB 叢集 (A)。假設複製 B 是從 Amazon DocumentDB 叢集建立的。建立複製時，不會複製任何資料。複製會指向與來源 Amazon DocumentDB 叢集相同的頁面集。



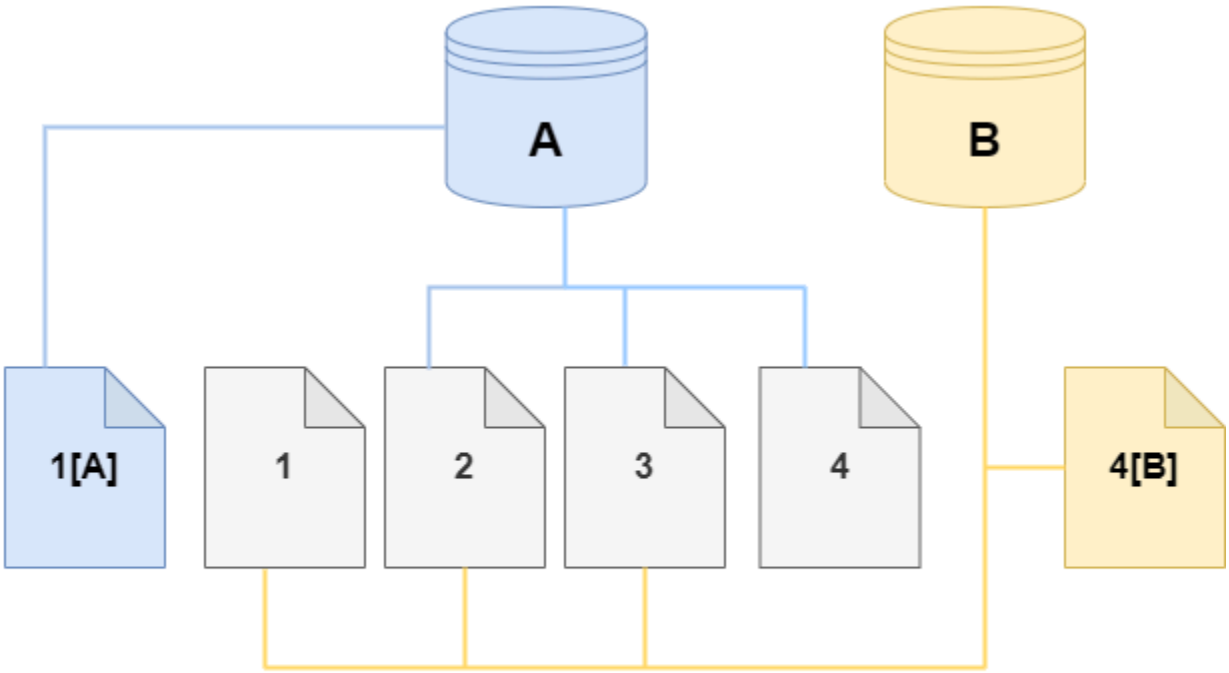
在建立複製時，通常不需要其他的儲存空間。寫入時複製通訊協定會使用與實體儲存媒體上相同的區段，作為來源區段。只有在來源區段的容量不足以容納整個複製區段時，才需要其他的儲存空間。如果是這種情況，來源區段會複製到另一個實體裝置上。

在下圖中，您可以找到使用相同叢集 A 及其複製 B 之動作的寫入時複製通訊協定範例，如上所示。假設您對 Amazon DocumentDB 叢集 (A) 進行變更，導致第 1 頁保留的資料發生變更。Amazon

DocumentDB 不會寫入原始頁面 1，而是建立新的頁面 1【A】。叢集 (A) 的 Amazon DocumentDB 叢集磁碟區現在指向第 1 頁【A】、第 2 頁、第 3 頁和第 4 頁，而複製 (B) 仍然參考原始頁面。



至於複製上，已對儲存磁碟區上的第 4 頁進行變更。Amazon DocumentDB 不會寫入原始頁面 4，而是建立新的頁面 4【B】。複製現在會指向第 1、2、3 和 4 頁 [B]，而叢集 (A) 會繼續指向 1 [A]、2、3 和 4。



隨著來源 Amazon DocumentDB 叢集磁碟區和複製中隨時間發生更多變更，需要更多儲存空間來擷取和儲存變更。

刪除來源叢集磁碟區

在您刪除具有一個或多個與其相關聯複製的來源叢集磁碟區時，複製不會受到影響。複製會繼續指向叢集磁碟區先前所擁有的頁面。

建立 Amazon DocumentDB 複製

您可以在與來源 Amazon DocumentDB 叢集相同的 AWS 帳戶中建立複製。若要這樣做，您可以使用 AWS 管理主控台 或 AWS CLI 和下列程序。

透過使用 Amazon DocumentDB 複製，您可以從佈建的 Amazon DocumentDB 叢集建立佈建的 Amazon DocumentDB 叢集複製。

Using the AWS 管理主控台

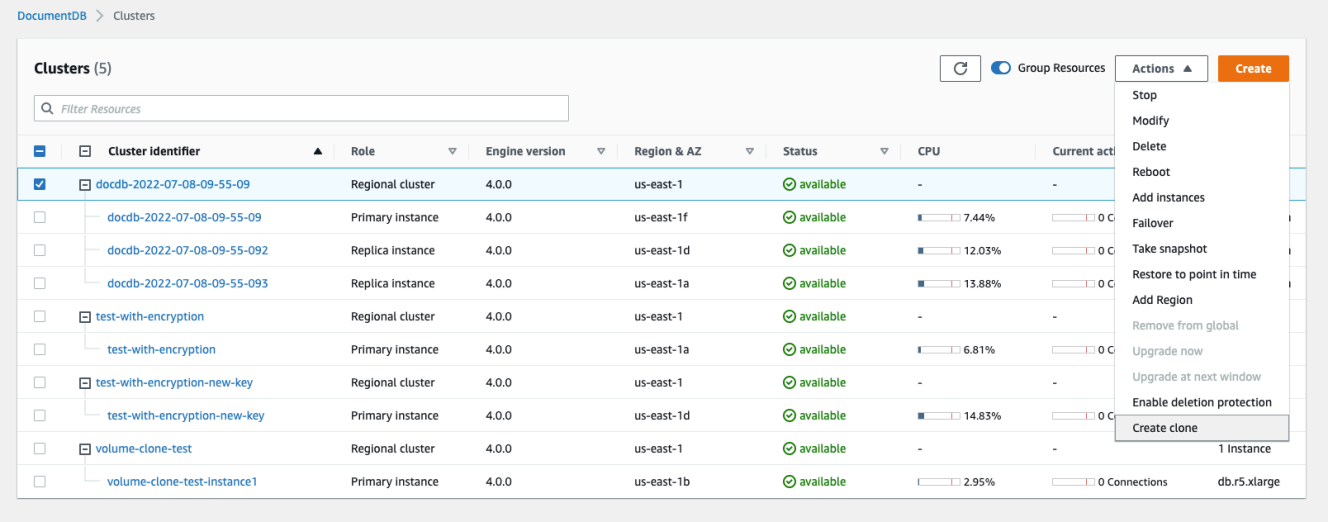
下列程序說明如何使用 複製 Amazon DocumentDB 叢集 AWS 管理主控台。

使用 在具有一個 Amazon DocumentDB 執行個體的 Amazon DocumentDB 叢集中 AWS 管理主控台 建立複製。

這些指示適用於建立複製之相同 AWS 帳戶所擁有的資料庫叢集。資料庫叢集必須擁有與 Amazon DocumentDB 不支援跨帳戶複製相同的 AWS 帳戶。

使用 建立您 AWS 帳戶所擁有資料庫叢集的複製 AWS 管理主控台

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/docdb：// 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。
3. 從清單中選擇您的 Amazon DocumentDB 叢集，然後在動作中，選擇建立複製。

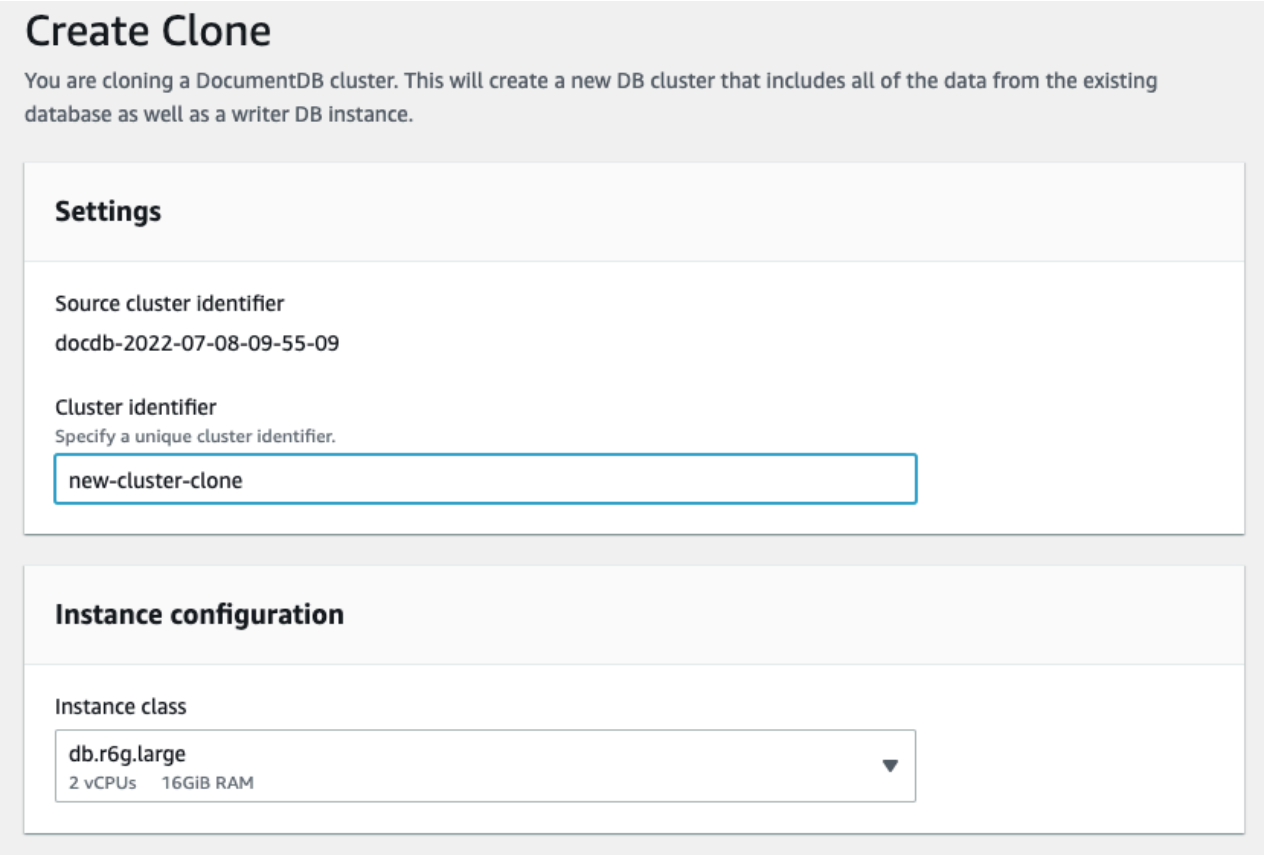


建立複製頁面隨即開啟，您可以在其中設定叢集識別符和執行個體類別，以及 Amazon DocumentDB 叢集複製的其他選項。

4. 在 Settings (設定) 區段中，執行下列動作：

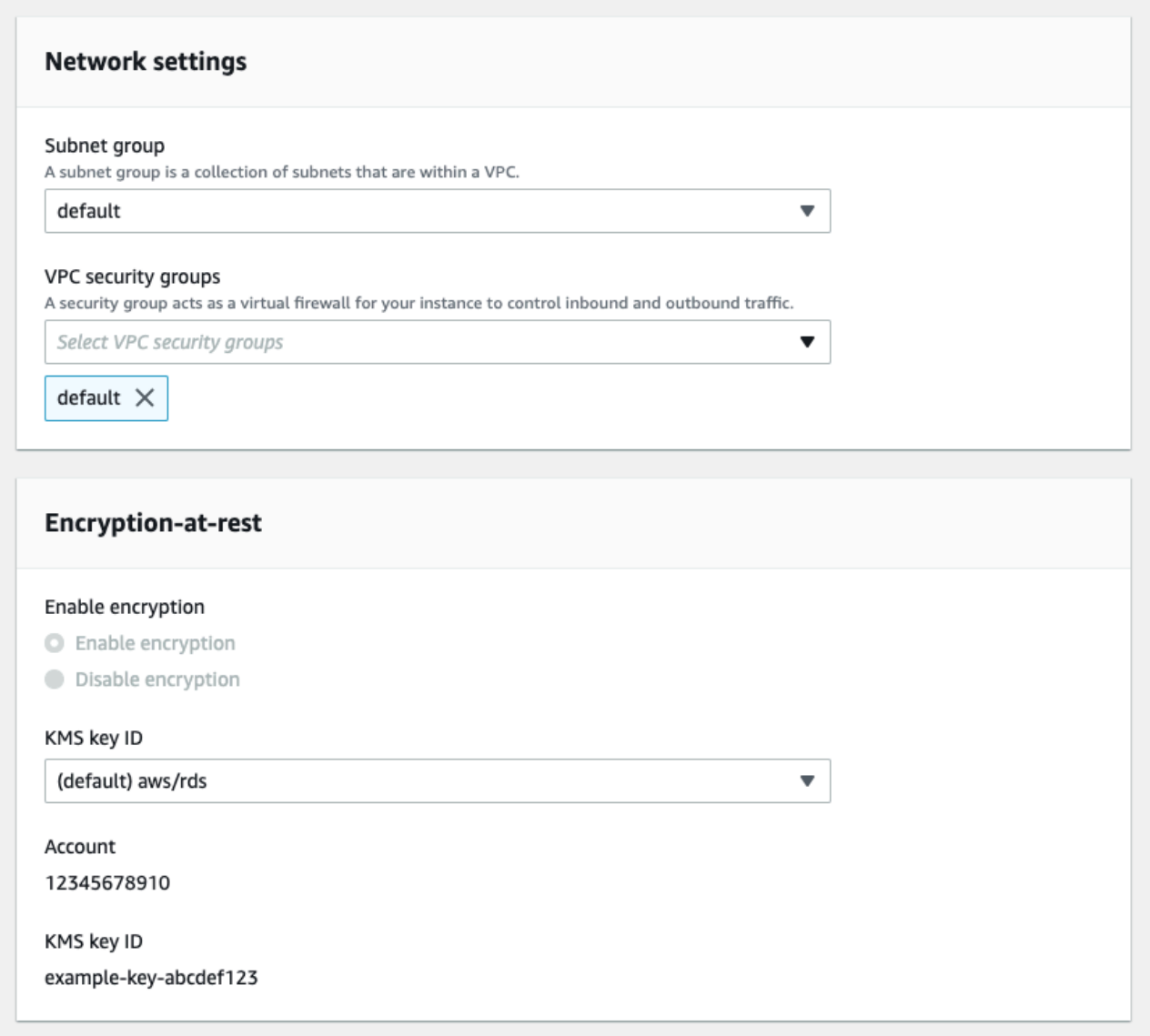
   a. 針對叢集識別符，輸入您要提供給複製 Amazon DocumentDB 叢集的名稱。

   b. 針對執行個體組態，為複製的 Amazon DocumentDB 叢集選取適當的執行個體類別。

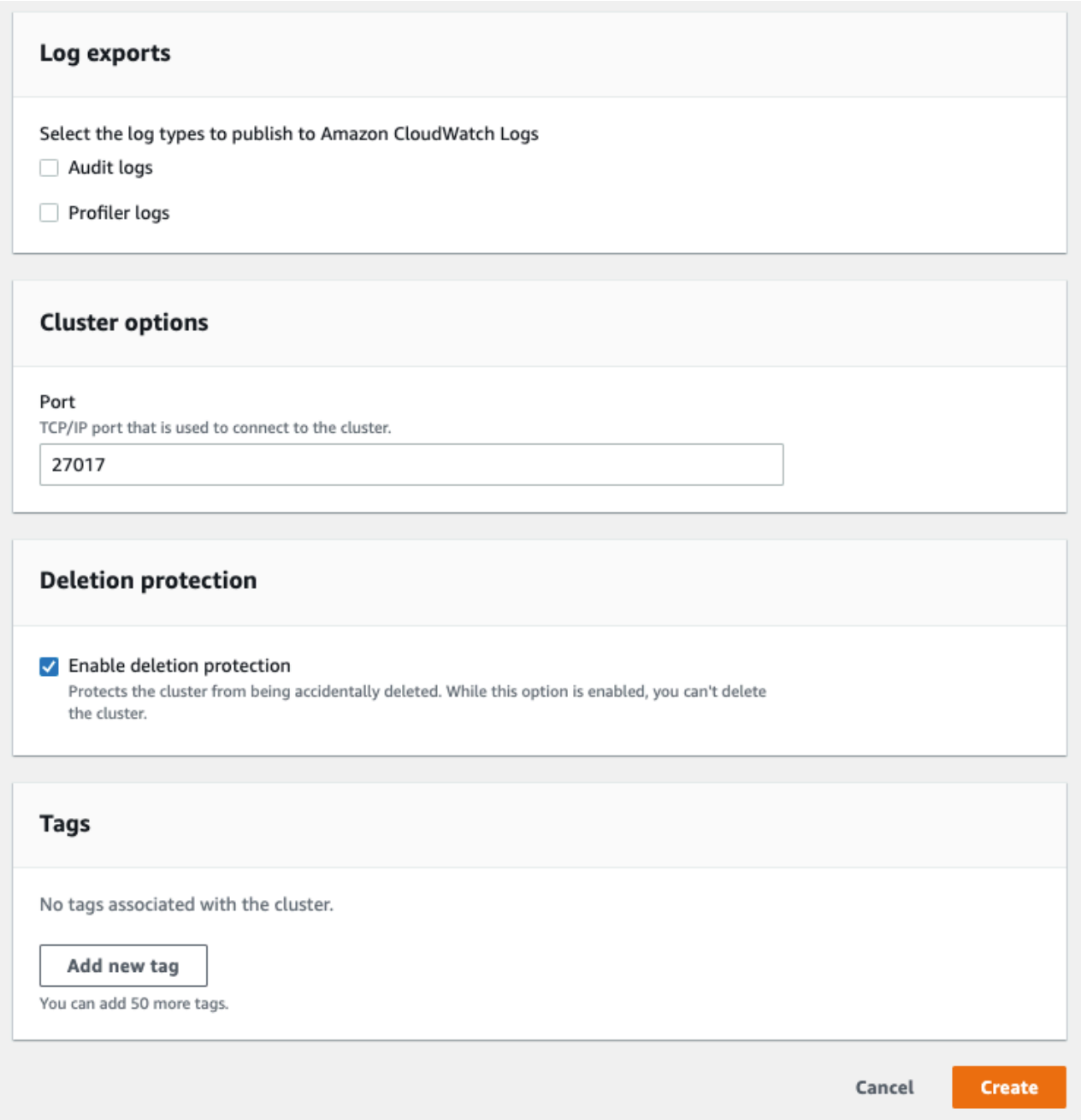


c. 針對網路設定，為您的使用案例和相關聯的 VPC 安全群組選擇子網路群組。

d. 對於Encryption-at-rest，如果來源叢集（正在複製的叢集）已啟用加密，則複製的叢集也必須啟用加密。如果此案例為 true，則啟用加密選項會顯示為灰色（停用），但已選取啟用加密選項。相反地，如果來源叢集未啟用加密，則可以使用啟用加密選項，您可以選擇啟用或停用加密。



e. 完成新的叢集複製組態，方法是選取要匯出的日誌類型（選用）、輸入用來連線至叢集的特定連接埠，以及啟用保護，避免意外刪除叢集（預設為啟用）。



f. 完成輸入 Amazon DocumentDB 叢集複製的所有設定。若要進一步了解 Amazon DocumentDB 叢集和執行個體設定，請參閱 管理 Amazon DocumentDB 叢集。

5. 選擇建立複製以啟動所選 Amazon DocumentDB 叢集的 Amazon DocumentDB 複製。

建立複製時，它會與主控台資料庫區段中的其他 Amazon DocumentDB 叢集一起列出，並顯示其目前狀態。在您複製的狀態為 Available (可用) 時，表示已可使用。

Using the AWS CLI

使用 AWS CLI 複製 Amazon DocumentDB 叢集需要幾個步驟。

您使用的`restore-db-cluster-to-point-in-time` AWS CLI 命令會產生具有 0 個 Amazon DocumentDB 執行個體的空 Amazon DocumentDB 叢集。也就是說，命令只會還原 Amazon DocumentDB 叢集，而不是該叢集的資料庫執行個體。您可以在複製可用之後分別執行該動作。程序珠的兩個步驟如下：

1. 透過使用 restore-db-cluster-to-point-in-time CLI 命令來建立複製。您與此命令搭配使用的參數會控制要建立之空 Amazon DocumentDB 叢集 （ 複製 ） 的容量類型和其他詳細資訊。
2. 使用 create-db-instance CLI 命令在還原的 Amazon DocumentDB 叢集中重新建立 Amazon DocumentDB 執行個體，為複製建立 Amazon DocumentDB 執行個體。

以下命令假設 AWS CLI 已使用您的 AWS 區域做為預設值進行設定。此方法可以避免在每個密令中傳遞 `--region` 名稱。如需詳細資訊，請參閱設定 AWS CLI。您還可以指定後續每個 CLI 命令中的 `--region`。

建立複製

您傳遞給 `restore-db-cluster-to-point-in-time` CLI 命令的特定參數皆有所不同。您傳遞的內容取決於您要建立的複製類型。

使用下列程序從佈建的 Amazon DocumentDB 叢集建立佈建的 Amazon DocumentDB 複製。

建立與來源 Amazon DocumentDB 叢集相同引擎模式的複製

- 使用 `restore-db-cluster-to-point-in-time` CLI 命令，並指定下列參數的值：
  - `--db-cluster-identifier` – 為您的複製選擇一個有意義的名稱。在您使用 restore-db-cluster-to-point-in-time CLI 命令時，為複製命名。
  - `--restore-type` – 使用 `copy-on-write` 來建立來源資料庫叢集的複製。如果沒有此參數， 會`restore-db-cluster-to-point-in-time`還原 Amazon DocumentDB 叢集，而不是建立複製。的預設值`restore-type`為 `full-copy`。
  - `--source-db-cluster-identifier` – 使用您要複製的來源 Amazon DocumentDB 叢集名稱。
  - `--use-latest-restorable-time` – 此值會指向複製的最新可還原磁碟區資料。此參數是 的必要參數，`restore-type copy-on-write`但您無法`restore-to-time parameter`搭配它使用 。

下列範例會從名為 my-source-cluster 的叢集建立名為 my-clone 的複製。

針對 Linux、macOS 或 Unix：

```
aws docdb restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

針對 Windows：

```
aws docdb restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

該命令會返回包含複製詳細資訊的 JSON 對象。在嘗試為複製建立資料庫執行個體之前，請檢查以確定您複製的資料庫叢集是否可用。如需詳細資訊，請參閱以下檢查狀態和取得複製詳細資訊：

檢查狀態並取得複製詳細資訊

您可以使用下列命令，來檢查您新建立之空白資料庫叢集的狀態。

```
$ aws docdb describe-db-clusters --db-cluster-identifier my-clone --query '*[].
[Status]' --output text
```

或者，您可以使用下列 AWS CLI 查詢，取得為複製建立資料庫執行個體所需的狀態和其他值：

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters --db-cluster-identifier my-clone \
  --query '*[].{Status:Status,Engine:Engine,EngineVersion:EngineVersion}'
```

針對 Windows：

```
aws docdb describe-db-clusters --db-cluster-identifier my-clone ^
  --query "*[].{Status:Status,Engine:Engine,EngineVersion:EngineVersion}"
```

此查詢會返回類似以下的內容：

```
[
  {
        "Status": "available",
        "Engine": "docdb",
        "EngineVersion": "4.0.0",
    }
]
```

為您的複製建立 Amazon DocumentDB 執行個體

使用 create-db-instance CLI 命令來建立您複製的資料庫執行個體。

`--db-instance-class` 參數僅用於佈建的 Amazon DocumentDB 叢集。

針對 Linux、macOS 或 Unix：

```
aws docdb create-db-instance \
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
    --db-instance-class  db.r5.4xlarge \
    --engine docdb
```

針對 Windows：

```
aws docdb create-db-instance ^
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class  db.r5.4xlarge ^
    --engine docdb
```

用於複製的參數

下表摘要說明與 搭配使用`restore-db-cluster-to-point-in-time`以複製 Amazon DocumentDB 叢集的各種參數。

參數	描述
--source-db-cluster-identifier	使用您要複製的來源 Amazon DocumentDB 叢集名稱。

參數	描述
--db-cluster-identifier	為您的複製選擇一個有意義的名稱。您使用 `restore-db-cluster-to-point-in-time` 命令為您的複製命名。然後，您將此名稱傳遞給 `create-db-instance` 命令。
--restore-type	指定 `copy-on-write` 做為來源資料庫叢集的複製`--restore-type`，而不是還原來源 Amazon DocumentDB 叢集。
--use-latest-restorable-time	此值會指向複製的最新可還原磁碟區資料。

了解 Amazon DocumentDB 叢集容錯能力

Amazon DocumentDB 叢集在設計上具有容錯能力。每個叢集的磁碟區在單一 中跨越多個可用區域 AWS 區域，而每個可用區域都包含叢集磁碟區資料的複本。此功能意味著叢集可承受可用區域故障，完全不會遺失資料，服務只會短暫中斷。

如果叢集中的主要執行個體失敗，Amazon DocumentDB 會以下列兩種方式之一自動容錯移轉至新的主要執行個體：

- 透過將現有的 Amazon DocumentDB 複本提升為根據每個複本的提升方案設定選擇的新主要執行個體，然後為先前的主要執行個體建立取代。複本執行個體的容錯移轉通常需要不到 30 秒的時間。在此期間，讀取和寫入操作可能會短暫中斷。若要提高叢集的可用性，請在兩個或多個不同的可用區域中建立至少一個或多個 Amazon DocumentDB 複本。
- 建立新的主要執行個體。只有在叢集中沒有複本執行個體且可能需要幾分鐘才能完成時，才會發生這種情況。

如果叢集有一或多個 Amazon DocumentDB 複本，Amazon DocumentDB 複本會在失敗事件期間提升為主要執行個體。失敗事件會導致短暫中斷，在此期間，讀取和寫入操作會失敗，並引發例外狀況。不過，服務通常會在 120 秒之內恢復，往往不超過 60 秒。若要提高叢集的可用性，請在兩個或多個不同的可用區域中建立至少一個或多個 Amazon DocumentDB 複本。

您可以透過為每個複本指派優先順序，自訂在失敗後將 Amazon DocumentDB 複本提升為主要執行個體的順序。優先順序從 0 (代表最高優先順序) 到 15 (代表最低優先順序)。如果主要執行個體失敗，具有最高優先順序的 Amazon DocumentDB 複本會提升為新的主要執行個體。您可以隨時修改 Amazon DocumentDB 複本的優先順序。修改優先順序不會觸發容錯移轉。您可以搭配 `--promotion-tier`

參數使用 `modify-db-instance` 操作。如需更多有關自訂執行個體容錯移轉優先順序的詳細資訊，請參閱Amazon DocumentDB 容錯移轉。

多個 Amazon DocumentDB 複本可以共用相同的優先順序，進而產生提升層。如果兩個或多個 Amazon DocumentDB 複本具有相同的優先順序，則大小最大的複本會提升為主要複本。如果兩個或多個 Amazon DocumentDB 複本具有相同的優先順序和大小，則會提升相同提升層中的任意複本。

如果叢集不包含任何 Amazon DocumentDB 複本，則會在失敗事件期間重新建立主要執行個體。失敗事件會導致中斷，在此期間，讀取和寫入操作會失敗，並引發例外狀況。建立新的主要執行個體後，服務就會恢復，通常不超過 10 分鐘。將 Amazon DocumentDB 複本提升為主要執行個體比建立新的主要執行個體快得多。

管理 Amazon DocumentDB 執行個體

下列主題提供的資訊可協助您管理 Amazon DocumentDB 執行個體。當中包括有關執行個體類別和狀態，以及如何建立、刪除和修改執行個體的詳細資訊。

主題

- 判斷執行個體的狀態
- Amazon DocumentDB 執行個體生命週期
- 管理執行個體類別
- NVMe 支援的執行個體

判斷執行個體的狀態

若要查看有效的執行個體狀態、它們的意義，以及如何判斷執行個體狀態的詳細資訊，請參閱監控 Amazon DocumentDB 執行個體的狀態。

Amazon DocumentDB 執行個體生命週期

Amazon DocumentDB 執行個體的生命週期包括建立、修改、維護和升級、執行備份和還原、重新啟動和刪除執行個體。本節提供如何完成這些程序的相關資訊。

主題

- 將 Amazon DocumentDB 執行個體新增至叢集
- 描述 Amazon DocumentDB 執行個體
- 修改 Amazon DocumentDB 執行個體

- 重新啟動 Amazon DocumentDB 執行個體
- 刪除 Amazon DocumentDB 執行個體

您可以使用 AWS 管理主控台 或 建立新的 Amazon DocumentDB 執行個體 AWS CLI。若要將執行個體新增至叢集，叢集必須處於可用狀態。您無法將執行個體新增至已停止的叢集。如果叢集已停止，請先啟動叢集，等待叢集變成可用，然後新增執行個體。如需詳細資訊，請參閱停止和啟動 Amazon DocumentDB 叢集。

> **Note**
>
> 如果您使用主控台建立 Amazon DocumentDB 叢集，系統會自動為您同時建立執行個體。如果您要建立其他執行個體，請使用下列程序之一。

將 Amazon DocumentDB 執行個體新增至叢集

Using the AWS 管理主控台

使用下列程序，使用 Amazon DocumentDB 主控台為您的叢集建立執行個體。

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 若要選擇要新增執行個體的叢集，請選取叢集名稱左側的按鈕。
4. 選擇 Actions (動作)，然後選擇 Add instances (新增執行個體)。
5. 在 Add instance to: (新增執行個體至)<cluster-name> 頁面，針對您想要加到叢集的每個執行個體重複以下步驟。您最多可以有 15 個。

   a. 執行個體識別符 - 您可以為此執行個體輸入唯一識別符，或允許 Amazon DocumentDB 根據叢集識別符提供執行個體識別符。

      執行個體命名限制：

- 長度為 【1-63】 個字母、數字或連字號。
- 第一個字元必須是字母。
- 不能以連字號結尾，或包含兩個連續連字號。
- 對於每個區域每個 Amazon RDS、Neptune 和 Amazon DocumentDB 的所有執行個體 AWS 帳戶，必須是唯一的。

b. 執行個體類別 — 從下拉式清單中，選擇您想要用於此執行個體的執行個體類型。

c. 提升方案：從下拉式清單中，選擇執行個體的提升方案，或選擇無偏好設定，以允許 Amazon DocumentDB 為您的執行個體設定提升方案。數字越小表示優先順序越高。如需詳細資訊，請參閱控制容錯移轉目標。

d. 若要新增更多執行個體，請選擇 Add additional instances (新增其他執行個體)，然後重複步驟 a、b 和 c。

6. 完成操作。

- 若要新增執行個體到您的叢集，請選擇 Create (建立)。
- 若要取消操作，請選擇 Cancel (取消)。

建立執行個體需要幾分鐘才能完成。您可以使用 主控台或 AWS CLI 來檢視執行個體的狀態。如需詳細資訊，請參閱監控執行個體的狀態。

Using the AWS CLI

使用 `create-db-instance` AWS CLI 操作搭配下列參數，為您的叢集建立主要執行個體。

- **`--db-instance-class`** – 必要。執行個體的運算與記憶體容量，例如 `db.m4.large`。並非所有執行個體類別都可在所有 中使用 AWS 區域。
- **`--db-instance-identifier`** – 必要。識別執行個體的字串。

  執行個體命名限制條件：

  - 長度為 【1-63】 個字母、數字或連字號。
  - 第一個字元必須是字母。
  - 不能以連字號結尾，或包含兩個連續連字號。
  - 對於每個區域每個 Amazon RDS、Neptune 和 Amazon DocumentDB 的所有執行個體 AWS 帳戶，必須是唯一的。
- **`--engine`** – 必要。必須為 `docdb`。

- **--availability-zone** — 選用。要在其中建立執行個體的可用區域。使用此參數尋找不同可用區域中的執行個體，以提高容錯能力。如需詳細資訊，請參閱Amazon DocumentDB 高可用性和複寫。
- **--promotion-tier** — 選用。此執行個體的容錯移轉優先順序方案。必須介於 0 和 15 之間，數字越小優先順序越高。如需詳細資訊，請參閱控制容錯移轉目標。

1. 首先，決定您可以在其中建立執行個體的可用區域。

   如果您想要在建立執行個體之前指定可用區域，請執行下列命令來判斷哪些可用區域可供您的 Amazon DocumentDB 叢集使用。

   針對 Linux、macOS 或 Unix：

   ```
   aws docdb describe-db-clusters \
          --query 'DBClusters[*].[DBClusterIdentifier,AvailabilityZones[*]]'
   ```

   針對 Windows：

   ```
   aws docdb describe-db-clusters ^
          --query 'DBClusters[*].[DBClusterIdentifier,AvailabilityZones[*]]'
   ```

   此操作的輸出將會如下所示。

   ```
   [
       [
           "sample-cluster",
           [
               "us-east-1c",
               "us-east-1b",
               "us-east-1a"
           ]
       ]
   ]
   ```

2. 其次，判定您可以在區域中建立哪些執行個體類別。

   若要判斷您區域中可用的執行個體類別，請執行以下命令。從輸出中，為您要新增至 Amazon DocumentDB 叢集的執行個體選擇執行個體類別。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-orderable-db-instance-options \
        --engine docdb \
        --query 'OrderableDBInstanceOptions[*].DBInstanceClass'
```

針對 Windows：

```
aws docdb describe-orderable-db-instance-options ^
        --engine docdb ^
        --query 'OrderableDBInstanceOptions[*].DBInstanceClass'
```

此操作的輸出將會如下所示。

```
[
    "db.r5.16xlarge",
    "db.r5.2xlarge",
    "db.r5.4xlarge",
    "db.r5.8xlarge",
    "db.r5.large",
    "db.r5.xlarge"
]
```

3. 最後，將執行個體新增至 Amazon DocumentDB 叢集。

若要將執行個體新增至 Amazon DocumentDB 叢集，請執行下列命令。

針對 Linux、macOS 或 Unix：

```
aws docdb create-db-instance \
       --db-cluster-identifier sample-cluster \
       --db-instance-identifier sample-instance-2 \
       --availability-zone us-east-1b \
       --promotion-tier 2 \
       --db-instance-class db.r5.xlarge \
       --engine docdb
```

針對 Windows：

```
aws docdb create-db-instance ^
```

```
    --db-cluster-identifier sample-cluster ^
    --db-instance-identifier sample-instance-2 ^
    --availability-zone us-east-1b ^
    --promotion-tier 2 ^
    --db-instance-class db.r5.xlarge ^
    --engine docdb
```

此操作的輸出將會如下所示。

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "sample-instance-2",
        "DBInstanceClass": "db.r5.xlarge",
        "Engine": "docdb",
        "DBInstanceStatus": "creating",
        "PreferredBackupWindow": "02:00-02:30",
        "BackupRetentionPeriod": 1,
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcd0123",
                "Status": "active"
            }
        ],
        "AvailabilityZone": "us-east-1b",
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-6242c31a",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-abcd0123",
                    "SubnetAvailabilityZone": {
                        "Name": "us-west-2a"
                    },
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-wxyz0123",
                    "SubnetAvailabilityZone": {
                        "Name": "us-west-2b"
                    },
                    "SubnetStatus": "Active"
```

```
                }
            ]
        },
        "PreferredMaintenanceWindow": "sun:11:35-sun:12:05",
        "PendingModifiedValues": {},
        "EngineVersion": "3.6.0",
        "AutoMinorVersionUpgrade": true,
        "PubliclyAccessible": false,
        "DBClusterIdentifier": "sample-cluster",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "CACertificateIdentifier": "rds-ca-2019",
        "PromotionTier": 2,
        "DBInstanceArn": "arn:aws:rds:us-east-1:<accountID>:db:sample-instance-2"
    }
}
```

建立執行個體需要幾分鐘才能完成。您可以使用 主控台或 AWS CLI 來檢視執行個體的狀態。如需詳細資訊，請參閱監控 Amazon DocumentDB 執行個體的狀態。

描述 Amazon DocumentDB 執行個體

您可以使用 Amazon DocumentDB 管理主控台或 查看 AWS CLI 連線端點、安全群組 VPCs、憑證授權單位和與 Amazon DocumentDB 執行個體相關的參數群組等詳細資訊。

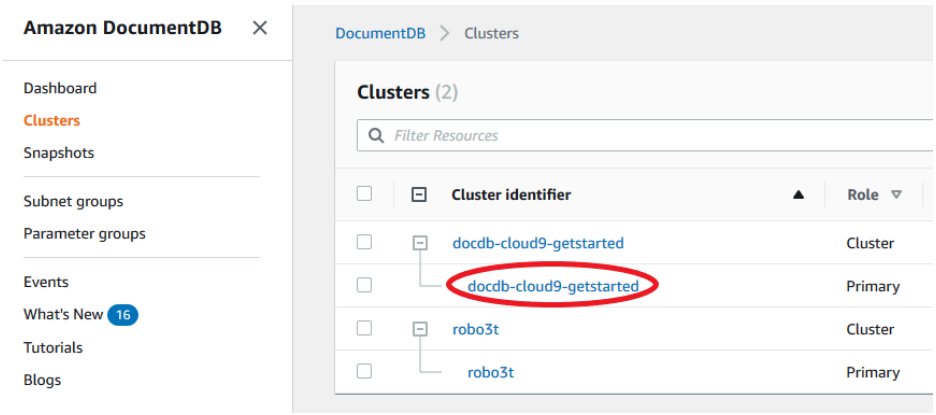
Using the AWS 管理主控台

若要使用 AWS 管理主控台檢視執行個體的詳細資訊，請依照下列步驟操作。

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

> **Tip**
>
> 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在叢集導覽方塊中，您會看到叢集識別符欄。您的執行個體會列在叢集下，類似於下列螢幕擷取畫面。



4. 在執行個體清單中，選擇您要查看其詳細資訊的執行個體名稱。執行個體的相關資訊分為下列群組：

   - 摘要 - 有關執行個體的一般資訊，包括引擎版本、類別、狀態和任何待定維護。
   - 連線與安全性 — Connect 區段列出要使用 mongo shell 或 應用程式連線到此執行個體的連線端點。Seucirty Groups (安全群組) 區段會列出與此執行個體相關聯的安全群組，以及其 VPC ID 和描述。
   - 組態 - 詳細資訊區段列出執行個體的組態和狀態，包括執行個體的 Amazon Resource Name (ARN)、端點、角色、類別和憑證授權單位。它也會列出執行個體的安全性和網路設定，以及備份資訊。Cluster details (叢集詳細資訊) 區段會列出此執行個體所屬叢集的詳細資訊。Cluster instances (叢集執行個體) 區段會列出屬於您叢集的所有執行個體，當中每個執行個體的角色和叢集參數群組狀態。

     > **Note**
     >
     > 您可以透過選取叢集詳細資訊標頭旁的修改，來修改與執行個體相關聯的叢集。 如需詳細資訊，請參閱修改 Amazon DocumentDB 叢集。

   - 監控 - 此執行個體的 CloudWatch Logs 指標。如需詳細資訊，請參閱使用 CloudWatch 監控 Amazon DocumentDB。
   - 事件和標籤 — 最近事件區段列出此執行個體的最近事件。Amazon DocumentDB 會保留與叢集、執行個體、快照、安全群組和叢集參數群組相關的事件記錄。此資訊包括與每個事件相關聯的日期、時間和訊息。Tags (標籤) 區段會列出連接至此叢集的標籤。如需詳細資訊，請參閱標記 Amazon DocumentDB 資源。

Using the AWS CLI

若要使用 檢視 Amazon DocumentDB 執行個體的詳細資訊 AWS CLI，請使用 `describe-db-clusters`命令，如下列範例所示。如需詳細資訊，請參閱《Amazon DocumentDB Resource Management API 參考[DescribeDBInstances](#)》中的 。

> **Note**
>
> 對於特定管理功能，例如叢集和執行個體生命週期管理，Amazon DocumentDB 會利用與 Amazon RDS 共用的操作技術。`filterName=engine,Values=docdb` 篩選條件參數只會傳回 Amazon DocumentDB 叢集。

1. 列出所有 Amazon DocumentDB 執行個體。

   下列 AWS CLI 程式碼列出區域中所有 Amazon DocumentDB 執行個體的詳細資訊。

   針對 Linux、macOS 或 Unix：

   ```
   aws docdb describe-db-instances \
       --filter Name=engine,Values=docdb
   ```

   針對 Windows：

   ```
   aws docdb describe-db-instances \
       --filter Name=engine,Values=docdb
   ```

2. 列出指定 Amazon DocumentDB 執行個體的所有詳細資訊

   以下程式碼會列出 `sample-cluster-instance` 的詳細資訊。將 `--db-instance-identifier` 參數加入執行個體名稱會將輸出限制為該特定執行個體的資訊。

   針對 Linux、macOS 或 Unix：

   ```
   aws docdb describe-db-instances \
       --db-instance-identifier sample-cluster-instance
   ```

   針對 Windows：

   ```
   aws docdb describe-db-instances \
   ```

```
    --db-instance-identifier sample-cluster-instance
```

此操作的輸出如下所示：

```
{
    "DBInstances": [
        {
            "DbiResourceId": "db-BJKKB54PIDV5QFKGVRX5T3S6GM",
            "DBInstanceArn": "arn:aws:rds:us-east-1:012345678901:db:sample-
cluster-instance-00",
            "VpcSecurityGroups": [
                {
                    "VpcSecurityGroupId": "sg-77186e0d",
                    "Status": "active"
                }
            ],
            "DBInstanceClass": "db.r5.large",
            "DBInstanceStatus": "creating",
            "AutoMinorVersionUpgrade": true,
            "PreferredMaintenanceWindow": "fri:09:32-fri:10:02",
            "BackupRetentionPeriod": 1,
            "StorageEncrypted": true,
            "DBClusterIdentifier": "sample-cluster",
            "EngineVersion": "3.6.0",
            "AvailabilityZone": "us-east-1a",
            "Engine": "docdb",
            "PromotionTier": 2,
            "DBInstanceIdentifier": "sample-cluster-instance",
            "PreferredBackupWindow": "00:00-00:30",
            "PubliclyAccessible": false,
            "DBSubnetGroup": {
                "DBSubnetGroupName": "default",
                "Subnets": [
                    {
                        "SubnetIdentifier": "subnet-4e26d263",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1a"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-afc329f4",
                        "SubnetAvailabilityZone": {
```

```
                            "Name": "us-east-1c"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-b3806e8f",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1e"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-53ab3636",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1d"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-991cb8d0",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1b"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-29ab1025",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1f"
                        },
                        "SubnetStatus": "Active"
                    }
                ],
                "VpcId": "vpc-91280df6",
                "DBSubnetGroupDescription": "default",
                "SubnetGroupStatus": "Complete"
            },
            "PendingModifiedValues": {},
            "KmsKeyId": "arn:aws:kms:us-east-1:012345678901:key/0961325d-
a50b-44d4-b6a0-a177d5ff730b"
        }
    ]
}
```

修改 Amazon DocumentDB 執行個體

您可以使用 AWS 管理主控台 或 來修改 Amazon DocumentDB 執行個體 AWS CLI。若要修改執行個體，執行個體必須處於可用狀態。您無法修改已停止的執行個體。如果叢集已停止，請先啟動叢集，等待執行個體變成可用，然後進行所需的修改。如需詳細資訊，請參閱停止和啟動 Amazon DocumentDB 叢集。

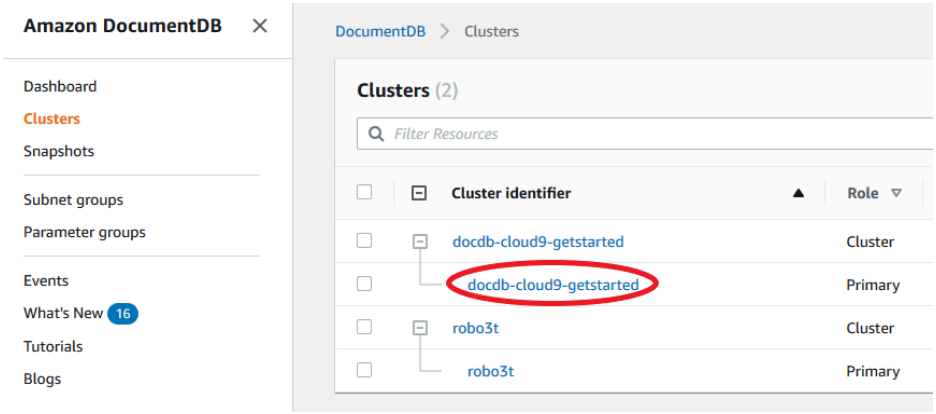
Using the AWS 管理主控台

若要使用主控台修改特定 Amazon DocumentDB 執行個體，請完成下列步驟。

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

> **Tip**
>
> 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在叢集導覽方塊中，您會看到叢集識別符欄。您的執行個體會列在叢集下，類似於下列螢幕擷取畫面。



4. 勾選您要修改之執行個體左側的方塊。
5. 選擇 Actions (動作)，然後選擇 Modify (修改)。
6. 在 Modify instance:<instance-name> (修改執行個體：<執行個體-名稱> 窗格中，進行您想要的變更。您可進行下列變更。

- 執行個體規格 — 執行個體識別符和類別。執行個體識別符命名限制：
  - 執行個體識別符 — 為 AWS 帳戶 目前區域中 擁有的所有執行個體輸入唯一的名稱。執行個體識別符必須包含 【1-63】 個英數字元或連字號、以字母做為第一個字元，且不能以連字號結尾或包含兩個連續連字號。
  - 執行個體類別 — 從下拉式功能表中，選取 Amazon DocumentDB 執行個體的執行個體類別。如需詳細資訊，請參閱管理執行個體類別。
- 憑證授權機構 — 此執行個體的伺服器憑證。如需詳細資訊，請參閱更新您的 Amazon DocumentDB TLS 憑證。
- 容錯移轉 — 在容錯移轉期間，具有最高提升層的執行個體將提升為主要執行個體。如需詳細資訊，請參閱Amazon DocumentDB 容錯移轉。
- 維護 — 將待定修改或修補程式套用至叢集中執行個體的維護時段。

7. 完成後，請選擇 Continue (繼續) 以查看變更的摘要。
8. 確認變更後，即可立即套用，或是在下一個維護時段在 Scheduling of modifications (修改的排程) 下套用。選擇 Modify Instance (修改執行個體)，以儲存變更。或者，您可以選擇 Cancel (取消) 以捨棄變更。

套用您的變更需要幾分鐘的時間才會完成。您只能使用狀態為可用的執行個體。您可以使用主控台或 AWS CLI監控執行個體的狀態。如需詳細資訊，請參閱監控 Amazon DocumentDB 執行個體的狀態。

Using the AWS CLI

若要使用 修改特定 Amazon DocumentDB 執行個體 AWS CLI，請使用 `modify-db-instance`搭配下列參數。如需詳細資訊，請參閱 ModifyDBInstance。以下程式碼將執行個體 `sample-instance` 的執行個體類別修改為 `db.r5.large`。

Parameters

- **`--db-instance-identifier`** – 必要。要修改的執行個體識別碼。
- **`--db-instance-class`** — 選用。執行個體的新運算和記憶體容量，例如 `db.r5.large`。並非所有執行個體類別都可在所有 中使用 AWS 區域。如果您修改執行個體類別，則會在變更期間發生中斷。除非此請求`ApplyImmediately`指定為 true，否則變更會在下一個維護時段套用。
- **`--apply-immediately`** 或 **`--no-apply-immediately`** — 選用。指定這項修改應立即套用，或等到下一次維護時段才套用。若省略此參數，則會在下一次維護時段進行修改。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb modify-db-instance \
       --db-instance-identifier sample-instance \
       --db-instance-class db.r5.large \
       --apply-immediately
```

針對 Windows：

```
aws docdb modify-db-instance ^
       --db-instance-identifier sample-instance ^
       --db-instance-class db.r5.large ^
       --apply-immediately
```

此操作的輸出將會如下所示。

```
{
    "DBInstances": [
        {
            "DBInstanceIdentifier": "sample-instance-1",
            "DBInstanceClass": "db.r5.large",
            "Engine": "docdb",
            "DBInstanceStatus": "modifying",
            "Endpoint": {
                "Address": "sample-instance-1.node.us-east-1.docdb.amazonaws.com",
                "Port": 27017,
                "HostedZoneId": "ABCDEFGHIJKLM"
            },
            "InstanceCreateTime": "2020-01-10T22:18:55.921Z",
            "PreferredBackupWindow": "02:00-02:30",
            "BackupRetentionPeriod": 1,
            "VpcSecurityGroups": [
                {
                    "VpcSecurityGroupId": "sg-abcd0123",
                    "Status": "active"
                }
            ],
            "AvailabilityZone": "us-east-1a",
            "DBSubnetGroup": {
                "DBSubnetGroupName": "default",
                "DBSubnetGroupDescription": "default",
```

```
                "VpcId": "vpc-abcd0123",
                "SubnetGroupStatus": "Complete",
                "Subnets": [
                    {
                        "SubnetIdentifier": "subnet-abcd0123",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1a"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-abcd0123",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1b"
                        },
                        "SubnetStatus": "Active"
                    }
                ]
            },
            "PreferredMaintenanceWindow": "sun:10:57-sun:11:27",
            "PendingModifiedValues": {
                "DBInstanceClass": "db.r5.large"
            },
            "EngineVersion": "3.6.0",
            "AutoMinorVersionUpgrade": true,
            "PubliclyAccessible": false,
            "DBClusterIdentifier": "sample-cluster",
            "StorageEncrypted": true,
            "KmsKeyId": "arn:aws:kms:us-east-1:123456789012:key/wJalrXUtnFEMI/
K7MDENG/bPxRfiCYEXAMPLEKEY",
            "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
            "CACertificateIdentifier": "rds-ca-2019",
            "PromotionTier": 1,
            "DBInstanceArn": "arn:aws:rds:us-east-1:123456789012:db:sample-
instance-1",
            "EnabledCloudwatchLogsExports": [
                "profiler"
            ]
        }
    ]
}
```

套用您的修改需要幾分鐘的時間才會完成。您只能使用狀態為可用的執行個體。您可以使用 AWS 管理主控台 或 監控執行個體的狀態 AWS CLI。如需詳細資訊，請參閱監控 Amazon DocumentDB 執行個體的狀態。

重新啟動 Amazon DocumentDB 執行個體

有時，您可能需要重新啟動 Amazon DocumentDB 執行個體，通常是基於維護原因。如果您進行某些變更，例如變更與叢集相關聯的叢集參數群組，則必須重新啟動叢集中的執行個體讓變更生效。您可以使用 AWS 管理主控台 或 重新啟動指定的執行個體 AWS CLI。

重新啟動執行個體，將重新啟動資料庫引擎服務。重新啟動會導致暫時中斷，在此期間，執行個體狀態會設定為 `rebooting`。重新啟動完成時，會建立 Amazon DocumentDB 事件。

重新啟動執行個體不會導致容錯移轉。若要容錯移轉 Amazon DocumentDB 叢集，請使用 AWS 管理主控台 或 AWS CLI 操作 `failover-db-cluster`。如需詳細資訊，請參閱Amazon DocumentDB 容錯移轉。

如果執行個體不是可用狀態，就無法重新啟動執行個體。資料庫無法使用可能由多種原因導致，例如，先前要求的修改，或維護時段的動作。如需執行個體狀態的詳細資訊，請參閱監控 Amazon DocumentDB 執行個體的狀態。

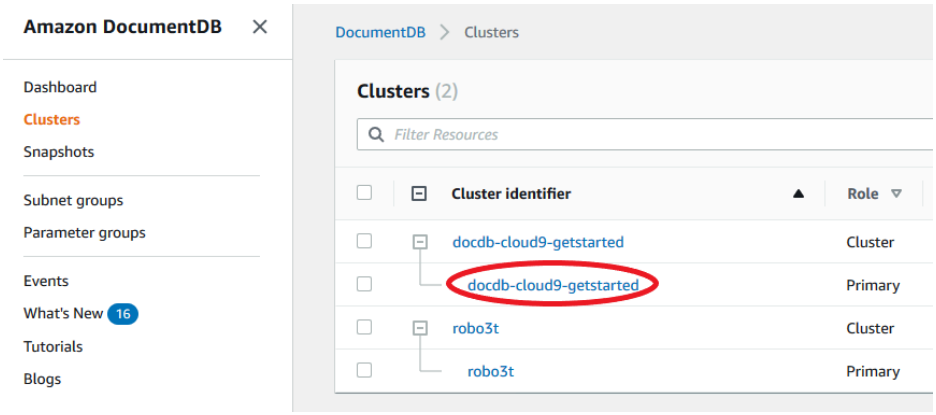
Using the AWS 管理主控台

以下程序會重新啟動您使用主控台指定的執行個體。

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/docdb：// 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在叢集導覽方塊中，您會看到叢集識別符欄。您的執行個體會列在叢集下，類似於下列螢幕擷取畫面。



4. 勾選您要重新啟動之執行個體左側的方塊。
5. 選擇Actions (動作)、選擇 Reboot (重新啟動)，然後選擇 Reboot (重新啟動) 以確認重新啟動。

執行個體重新啟動需要幾分鐘的時間來完成。您只能使用狀態為可用的執行個體。您可以使用主控台或 AWS CLI監控執行個體的狀態。如需詳細資訊，請參閱監控 Amazon DocumentDB 執行個體的狀態。

Using the AWS CLI

若要重新啟動 Amazon DocumentDB 執行個體，請使用 `reboot-db-instance`操作搭配 `--db-instance-identifier` 參數。此參數會指定要重新啟動之執行個體的識別符。

以下程式碼會重新啟動執行個體 `sample-instance`。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb reboot-db-instance \
       --db-instance-identifier sample-instance
```

針對 Windows：

```
aws docdb reboot-db-instance ^
       --db-instance-identifier sample-instance
```

此操作的輸出將會如下所示。

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "sample-instance",
```

```
        "DBInstanceClass": "db.r5.large",
        "Engine": "docdb",
        "DBInstanceStatus": "rebooting",
        "Endpoint": {
            "Address": "sample-instance.node.us-east-1.docdb.amazonaws.com",
            "Port": 27017,
            "HostedZoneId": "ABCDEFGHIJKLM"
        },
        "InstanceCreateTime": "2020-03-27T08:05:56.314Z",
        "PreferredBackupWindow": "02:00-02:30",
        "BackupRetentionPeriod": 1,
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcd0123",
                "Status": "active"
            }
        ],
        "AvailabilityZone": "us-east-1c",
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-abcd0123",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-abcd0123",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1a"
                    },
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-wxyz0123",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1b"
                    },
                    "SubnetStatus": "Active"
                }
            ]
        },
        "PreferredMaintenanceWindow": "sun:06:53-sun:07:23",
        "PendingModifiedValues": {},
        "EngineVersion": "3.6.0",
        "AutoMinorVersionUpgrade": true,
```

```
        "PubliclyAccessible": false,
        "DBClusterIdentifier": "sample-cluster",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "CACertificateIdentifier": "rds-ca-2019",
        "PromotionTier": 1,
        "DBInstanceArn": "arn:aws:rds:us-east-1:<accountID>:db:sample-instance",
        "EnabledCloudwatchLogsExports": [
            "profiler"
        ]
    }
}
```

執行個體重新啟動需要幾分鐘的時間來完成。您只能使用狀態為可用的執行個體。您可以使用主控台或 AWS CLI監控執行個體的狀態。如需詳細資訊，請參閱監控 Amazon DocumentDB 執行個體的狀態。

刪除 Amazon DocumentDB 執行個體

您可以使用 AWS 管理主控台 或 刪除 Amazon DocumentDB 執行個體 AWS CLI。若要刪除執行個體，執行個體必須處於可用狀態。您無法刪除已停止的執行個體。如果包含執行個體的 Amazon DocumentDB 叢集已停止，請先啟動叢集，等待執行個體變成可用，然後刪除執行個體。如需詳細資訊，請參閱停止和啟動 Amazon DocumentDB 叢集。

> **Note**
>
> Amazon DocumentDB 會將所有資料存放在叢集磁碟區中。資料會保留在該叢集磁碟區中，即使您移除叢集中的所有執行個體也一樣。如果您需要再次存取資料，可以隨時新增執行個體至叢集，並從您上次離開的位置繼續操作。

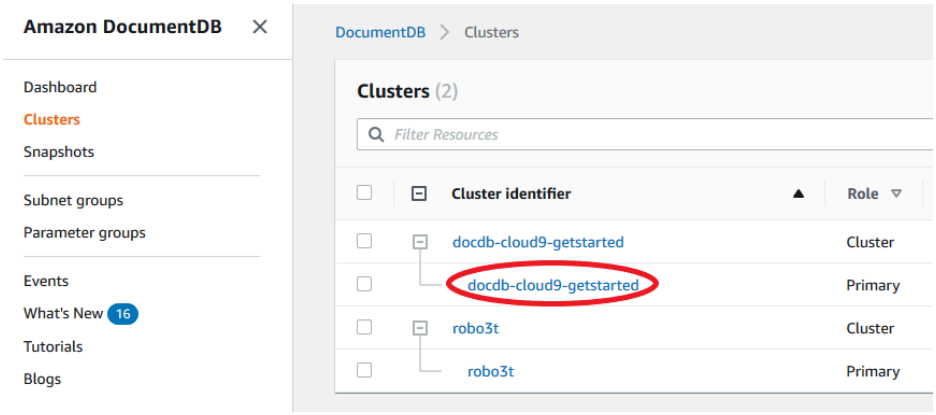
Using the AWS 管理主控台

下列程序會使用主控台刪除指定的 Amazon DocumentDB 執行個體。

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/docdb：// 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

> **Tip**
>
> 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在叢集導覽方塊中，您會看到叢集識別符欄。您的執行個體會列在叢集下，類似於下列螢幕擷取畫面。



4. 勾選您要刪除之執行個體左側的方塊。
5. 選取 Actions (動作)，然後選擇 Delete (刪除)。

   1. 如果您要刪除叢集中的最後一個執行個體：

      - Create final cluster snapshot? (建立最終叢集快照？) — 如果您想要在刪除叢集之前建立最終快照，請選擇是。否則請選擇否。
      - 最終快照名稱 — 如果您選擇建立最終快照，請輸入所建立新叢集快照的叢集快照識別符。
      - Delete <instance-name> instance? (刪除 <instance-name> 執行個體？) — 在欄位中輸入刪除整個叢集的片語，以確認刪除。

   2. 如果您不是刪除叢集中的最後一個執行個體：

      - Delete <instance-name> instance? (刪除 <instance-name> 執行個體？) — 在欄位中輸入刪除我以確認刪除的片語。

6. 選取 Delete (刪除) 以刪除執行個體。

刪除執行個體需要幾分鐘的時間。若要監控執行個體的狀態，請參閱監控 Amazon DocumentDB 執行個體的狀態。

Using the AWS CLI

下列程序會使用 刪除 Amazon DocumentDB 執行個體 AWS CLI。

1. 首先，判斷您的 Amazon DocumentDB 叢集中有多少執行個體：

   若要判斷叢集中有多少執行個體，請執行 `describe-db-clusters` 命令，如下所示。

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].
[DBClusterIdentifier,DBClusterMembers[*].DBInstanceIdentifier]'
```

   此操作的輸出將會如下所示。

```
[
    [
        "sample-cluster",
        [
            "sample-instance-1",
            "sample-instance-2"
        ]
    ]
]
```

2. 如果您的 Amazon DocumentDB 叢集中有多個執行個體：

   若要刪除指定的 Amazon DocumentDB 執行個體，請使用 `delete-db-instance`命令搭配 `--db-instance-identifier` 參數，如下所示。刪除執行個體需要幾分鐘的時間。若要監控執行個體的狀態，請參閱監控 Amazon DocumentDB 執行個體的狀態。

```
aws docdb delete-db-instance \
       --db-instance-identifier sample-instance-2
```

   此操作的輸出將會如下所示。

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "sample-instance-2",
        "DBInstanceClass": "db.r5.large",
        "Engine": "docdb",
```

```
        "DBInstanceStatus": "deleting",
        "Endpoint": {
            "Address": "sample-instance-2.node.us-east-1.docdb.amazonaws.com",
            "Port": 27017,
            "HostedZoneId": "ABCDEFGHIJKLM"
        },
        "InstanceCreateTime": "2020-03-27T08:05:56.314Z",
        "PreferredBackupWindow": "02:00-02:30",
        "BackupRetentionPeriod": 1,
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcd0123",
                "Status": "active"
            }
        ],
        "AvailabilityZone": "us-east-1c",
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-6242c31a",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-abcd0123",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1a"
                    },
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-wxyz0123",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1b"
                    },
                    "SubnetStatus": "Active"
                }
            ]
        },
        "PreferredMaintenanceWindow": "sun:06:53-sun:07:23",
        "PendingModifiedValues": {},
        "EngineVersion": "3.6.0",
        "AutoMinorVersionUpgrade": true,
        "PubliclyAccessible": false,
        "DBClusterIdentifier": "sample-cluster",
```

```
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "CACertificateIdentifier": "rds-ca-2019",
        "PromotionTier": 1,
        "DBInstanceArn": "arn:aws:rds:us-east-1:<accountID>:db:sample-instance-2",
        "EnabledCloudwatchLogsExports": [
            "profiler"
        ]
    }
}
```

3. 如果您要刪除的執行個體是 Amazon DocumentDB 叢集中的最後一個執行個體：

   如果您刪除 Amazon DocumentDB 叢集中的最後一個執行個體，您也會刪除該叢集，以及與該叢集相關聯的自動快照和持續備份。

   若要刪除叢集中的最後一個執行個體，您可以刪除該叢集並選擇性地建立最終快照。如需詳細資訊，請參閱刪除 Amazon DocumentDB 叢集。

刪除保護

刪除 Amazon DocumentDB 叢集的最後一個執行個體也會刪除叢集，以及與該叢集相關聯的自動快照和連續備份。無論您是使用 AWS 管理主控台 或 執行刪除操作，Amazon DocumentDB 都會對叢集強制執行刪除保護 AWS CLI。如果已啟用刪除保護，您就無法刪除叢集。

若要刪除已啟用刪除保護的叢集，您必須先修改叢集並停用刪除保護。如需詳細資訊，請參閱刪除 Amazon DocumentDB 叢集。

管理執行個體類別

執行個體類別會決定 Amazon DocumentDB （具有 MongoDB 相容性） 執行個體的運算和記憶體容量。您需要的執行個體類別取決於您的處理能力和記憶體需求。

Amazon DocumentDB 支援執行個體類別的 R4、R5、R6G、R8G、T3 和 T4G 系列。這些類別是新一代的執行個體類別，已針對記憶體密集型應用程式進行最佳化處理。如需這些類別的規格，請參閱執行個體類別規格。

主題

- 判斷執行個體類別

- 變更執行個體的類別
- 各區域的支援執行個體類別
- 執行個體類別規格

判斷執行個體類別

若要判斷執行個體的類別，您可以使用 AWS 管理主控台 或 `describe-db-instances` AWS CLI 操作。

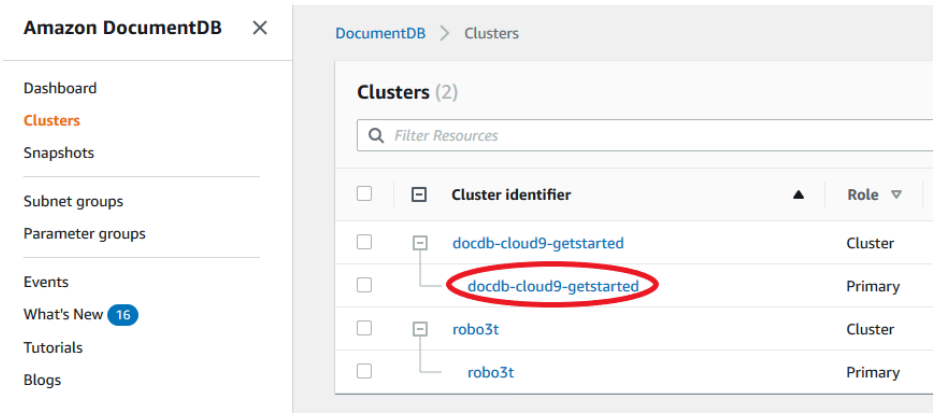
Using the AWS 管理主控台

若要判斷叢集執行個體的執行個體類別，請在 主控台中完成下列步驟。

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集以尋找您感興趣的執行個體。

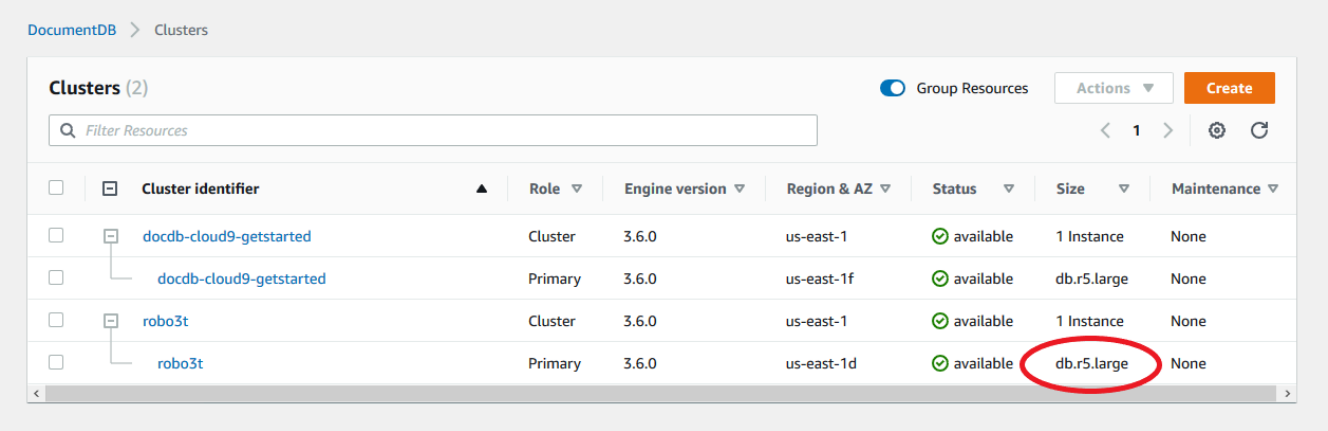
> **Tip**
>
> 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在叢集導覽方塊中，您會看到叢集識別符欄。您的執行個體會列在叢集下，類似於下列螢幕擷取畫面。



4. 在執行個體清單中，展開叢集以尋找您感興趣的執行個體。尋找您想要的執行個體。然後，查看執行個體資料列的大小欄，以查看其執行個體類別。

   在下圖中，執行個體 `robo3t` 的執行個體類別為 `db.r5.4xlarge`。



Using the AWS CLI

若要使用 判斷執行個體的類別 AWS CLI，請使用 `describe-db-instances`操作搭配下列參數。

- **`--db-instance-identifier`** — 選用。指定您要尋找其執行個體類別的執行個體。如果省略這個參數，`describe-db-instances` 會傳回最多 100 個執行個體的敘述。
- **`--query`** — 選用。指定要納入結果中的執行個體成員。如果省略此參數，則所有執行個體成員都會傳回。

Example

下列範例會尋找執行個體 的執行個體名稱和類別`sample-instance-1`。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-instances \
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' \
    --db-instance-identifier sample-instance-1
```

針對 Windows：

```
aws docdb describe-db-instances ^
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' ^
    --db-instance-identifier sample-instance-1
```

此操作的輸出將會如下所示。

```
[
    [
        "sample-instance-1",
        "db.r5.large"
    ]
```

Example

下列範例會尋找最多 100 個 Amazon DocumentDB 執行個體的執行個體名稱和類別。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-instances \
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' \
    --filter Name=engine,Values=docdb
```

針對 Windows：

```
aws docdb describe-db-instances ^
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' ^
    --filter Name=engine,Values=docdb
```

此操作的輸出將會如下所示。

```
[
    [
        "sample-instance-1",
        "db.r5.large"
    ],
    [
        "sample-instance-2",
        "db.r5.large"
    ],
    [
        "sample-instance-3",
        "db.r5.4xlarge"
    ],
    [
        "sample-instance-4",
        "db.r5.4xlarge"
    ]
```

```
]
```

如需詳細資訊，請參閱描述 Amazon DocumentDB 執行個體。

變更執行個體的類別

您可以使用 AWS 管理主控台 或 變更執行個體的執行個體類別 AWS CLI。如需詳細資訊，請參閱修改 Amazon DocumentDB 執行個體。

各區域的支援執行個體類別

Amazon DocumentDB 支援下列執行個體類別：

- R8G：最新一代採用 Arm 型 AWS Graviton4 處理器的記憶體最佳化執行個體，可提供比 R6G 執行個體高達 30% 的效能。
- R6G- 採用 Arm 型 AWS Graviton2 處理器的記憶體最佳化執行個體，相較於 R5 執行個體，效能提升高達 30%，且成本降低 5%。
- R6GD- 記憶體最佳化 R6G 執行個體，具有適用於暫時性資料的本機非揮發性記憶體快速 (NVMe) 型固態硬碟 (SSD) 儲存體。
- R5- 記憶體最佳化執行個體，以相同的執行個體成本提供比 R4 執行個體高達 100% 更好的效能。
- R4- 先前產生的記憶體最佳化執行個體。
- T4G- 最新一代的低成本爆量一般用途執行個體類型，採用 Arm 型 AWS Graviton2 處理器，可提供基準層級的 CPU 效能，比 T3 執行個體提供高達 35% 的更佳價格效能，非常適合使用中等 CPU 用量的應用程式，這些應用程式在使用時遇到暫時性尖峰。
- T3- 低成本爆量一般用途執行個體類型，可提供 CPU 效能的基準層級，並能夠視需要隨時爆量 CPU 用量。

如需這些執行個體類別的詳細規格，請參閱執行個體類別規格。

> **Note**
>
> I/O 最佳化儲存僅適用於引擎版本 5.0 和 8.0 （執行個體型叢集）。

特定 區域不一定支援特定執行個體類別。下表指定每個區域中 Amazon DocumentDB 支援的執行個體類別。

各區域的支援執行個體類別

區域	R8G	R6GD	R6G	R5	R4	T4G	T3	無伺服器
美國東部 (俄亥俄)	支援	支援	支援	支援	支援	支援	支援	支援
美國東部 (維吉尼亞北部)	支援	支援	支援	支援	支援	支援	支援	支援
美國西部 (奧勒岡)	支援	支援	支援	支援	支援	支援	支援	支援
非洲 (開普敦)			支援	支援		支援	支援	支援
南美洲 (聖保羅)	支援	支援	支援	支援		支援	支援	支援
亞太地區 (香港)	支援		支援	支援		支援	支援	支援
亞太地區 (海德拉巴)			支援	支援		支援	支援	支援
亞太地區 (馬來西亞)			支援			支援	支援	支援
亞太地區 (孟買)	支援	支援	支援	支援		支援	支援	支援
亞太地區 (大阪)	支援	支援	支援	支援		支援	支援	支援
亞太地區 (首爾)	支援	支援	支援	支援		支援	支援	支援
亞太地區 (悉尼)	支援	支援	支援	支援		支援	支援	支援
亞太地區 (雅加達)	支援	支援	支援	支援		支援	支援	支援
亞太地區 (墨爾本)			支援	支援		支援	支援	支援

區域	R8G	R6GD	R6G	R5	R4	T4G	T3	無伺服器
亞太地區 (新加坡)	支援	支援	支援	支援		支援	支援	支援
亞太區域 (泰國)			支援			支援	支援	支援
亞太地區 (東京)	支援	支援	支援	支援		支援	支援	支援
加拿大 (中部)	支援	支援	支援	支援		支援	支援	支援
加拿大西部 (卡加利)			支援	支援		支援	支援	支援
歐洲 (法蘭克福)	支援	支援	支援	支援		支援	支援	支援
歐洲 (蘇黎世)		支援	支援	支援		支援	支援	支援
歐洲 (愛爾蘭)	支援	支援	支援	支援	支援	支援	支援	支援
歐洲 (倫敦)	支援	支援	支援	支援		支援	支援	支援
歐洲 (米蘭)			支援	支援		支援	支援	支援
Europe (Paris)	支援	支援	支援	支援		支援	支援	支援
歐洲 (西班牙)	支援	支援	支援	支援		支援	支援	支援
歐洲 (斯德哥爾摩)	支援	支援	支援	支援		支援	支援	支援
墨西哥 (中部)			支援			支援	支援	支援
中東 (阿拉伯聯合大公國)			支援	支援		支援	支援	支援
中國 (北京)	支援	支援	支援	支援		支援	支援	支援
中國 (寧夏)	支援		支援	支援		支援	支援	支援

區域	R8G	R6GD	R6G	R5	R4	T4G	T3	無伺服器
以色列 (特拉維夫)			支援	支援		支援	支援	支援
AWS GovCloud（美國西部）	支援	支援	支援	支援			支援	支援
AWS GovCloud（美國東部）		支援	支援	支援		支援	支援	支援

執行個體類別規格

下表提供 Amazon DocumentDB 執行個體類別的詳細資訊，包括每個類別中支援的執行個體類型。您可以在表的下方找到每個表格欄的說明。

執行個體類別	vCPU[1]	記憶體 (GiB)[2]	NVMe SSD 分層快取 (GiB)[3]	最大溫度儲存 (GiB)[4]	基準/高載頻寬 (Gbps)[5]	支援引擎[6]
R8G – 以 Graviton4 為基礎的目前世代記憶體最佳化執行個體類別 R8G 是最新的執行個體系列 (Graviton4)，僅適用於引擎版本 5.0 和 8.0。						
db.r8g.large	2	16	-	30	0.937 / 12.5	5.0.0 和 8.0.0
db.r8g.xlarge	4	32	-	60	1.875 / 12.5	5.0.0 和 8.0.0
db.r8g.2xlarge	8	64	-	121	3.75 / 15.0	5.0.0 和 8.0.0
db.r8g.4xlarge	16	128	-	243	7.5 / 15.0	5.0.0 和 8.0.0

執行個體類別	vCPU[1]	記憶體 (GiB)[2]	NVMe SSD 分層快取 (GiB)[3]	最大溫度儲存 (GiB)[4]	基準/高載頻寬 (Gbps)[5]	支援引擎[6]
db.r8g.8xlarge	32	256	-	488	15	5.0.0 和 8.0.0
db.r8g.12xlarge	48	384	-	732	22	5.0.0 和 8.0.0
db.r8g.16xlarge	64	512	-	987	30	5.0.0 和 8.0.0
db.r8g.24xlarge	96	768	-	1484	40	5.0.0 和 8.0.0
db.r8g.48xlarge	192	1536	-	2967	50	5.0.0 和 8.0.0
R6G – 以 Graviton2 為基礎的目前世代記憶體最佳化執行個體類別						
db.r6g.large	2	16	-	32	0.75 / 10	4.0.0、5.0.0 和 8.0.0
db.r6g.xlarge	4	32	-	63	1.25 / 10	4.0.0、5.0.0 和 8.0.0
db.r6g.2xlarge	8	64	-	126	2.5 / 10	4.0.0、5.0.0 和 8.0.0
db.r6g.4xlarge	16	128	-	252	5.0 / 10	4.0.0、5.0.0 和 8.0.0
db.r6g.8xlarge	32	256	-	504	12	4.0.0、5.0.0 和 8.0.0
db.r6g.12xlarge	48	384	-	756	20	4.0.0、5.0.0 和 8.0.0

執行個體類別	vCPU[1]	記憶體 (GiB)[2]	NVMe SSD 分層快取 (GiB)[3]	最大溫度儲存 (GiB)[4]	基準/高載頻寬 (Gbps)[5]	支援引擎[6]
db.r6g.16xlarge	64	512	-	1008	25	4.0.0、5.0.0 和 8.0.0
R6GD – 以 Graviton2 為基礎的目前世代 NVMe 後端執行個體類別						
db.r6gd.xlarge	4	32	173	64	1.25 / 10	5.0.0 和 8.0.0
db.r6gd.2xlarge	8	64	346	128	2.5 / 10	5.0.0 和 8.0.0
db.r6gd.4xlarge	16	128	694	256	5.0 / 10	5.0.0 和 8.0.0
db.r6gd.8xlarge	32	256	1388	512	12	5.0.0 和 8.0.0
db.r6gd.12xlarge	48	384	2082	768	20	5.0.0 和 8.0.0
db.r6gd.16xlarge	64	512	2776	1024	25	5.0.0 和 8.0.0
R5 – 上一代記憶體最佳化執行個體類別						
db.r5.large	2	16	-	31	0.75 / 10	3.6.0、4.0.0、5.0.0 和 8.0.0
db.r5.xlarge	4	32	-	62	1.25 / 10	3.6.0、4.0.0、5.0.0 和 8.0.0

執行個體類別	vCPU[1]	記憶體 (GiB)[2]	NVMe SSD 分層快取 (GiB)[3]	最大溫度儲存 (GiB)[4]	基準/高載頻寬 (Gbps)[5]	支援引擎[6]
db.r5.2xlarge	8	64	-	124	2.5 / 10	3.6.0、4.0.0、5.0.0 和 8.0.0
db.r5.4xlarge	16	128	-	249	5.0 / 10	3.6.0、4.0.0、5.0.0 和 8.0.0
db.r5.8xlarge	32	256	-	504	10	3.6.0、4.0.0、5.0.0 和 8.0.0
db.r5.12xlarge	48	384	-	748	12	3.6.0、4.0.0、5.0.0 和 8.0.0
db.r5.16xlarge	64	512	-	1008	20	3.6.0、4.0.0、5.0.0 和 8.0.0
db.r5.24xlarge	96	768	-	1500	25	3.6.0、4.0.0、5.0.0 和 8.0.0
R4 – 上一代記憶體最佳化執行個體類別 R4 執行個體僅支援引擎 3.6 版。請注意，Amazon DocumentDB 3.6 已於 2026 年 3 月 30 日終止標準支援。延長支援（付費）開放至 2029 年 3 月。						
db.r4.large	2	15.25	-	30	0.75 / 10	僅限 3.6.0
db.r4.xlarge	4	30.5	-	60	1.25 / 10	僅限 3.6.0

執行個體類別	vCPU[1]	記憶體 (GiB)[2]	NVMe SSD 分層快取 (GiB)[3]	最大溫度儲存 (GiB)[4]	基準/高載頻寬 (Gbps)[5]	支援引擎[6]
db.r4.2xlarge	8	61	-	120	2.5 / 10	僅限 3.6.0
db.r4.4xlarge	16	122	-	240	5.0 /10	僅限 3.6.0
db.r4.8xlarge	32	244	-	480	10	僅限 3.6.0
db.r4.16xlarge	64	488	-	960	25	僅限 3.6.0
T4G – 以 Graviton2 為基礎的最新一代爆量效能執行個體類別 T 系列執行個體以無限制 CPU 高載模式執行。超出基準的爆量用量會額外計費。不支援全域叢集。						
db.t4g.medium	2	4	-	8.13	0.256 / 5	4.0.0、5.0.0 和 8.0.0
T3 – 上一代爆量效能執行個體類別						
db.t3.medium	2	4	-	7.5	0.256 / 5	3.6.0、4.0.0、5.0.0 和 8.0.0

執行個體類別	vCPU[1]	記憶體 (GiB)[2]	NVMe SSD 分層快取 (GiB)[3]	最大溫度儲存 (GiB)[4]	基準/高載頻寬 (Gbps)[5]	支援引擎[6]

1. vCPU — 虛擬中央處理單元 (CPUs的數量。虛擬 CPU 即為容量單位，可用來比較執行個體類別。您不再購買或租用特定的處理器並使用數月或數年，而是以小時為單位租用容量。我們的目標是無論實際的基礎硬體為何，均提供一致的 CPU 容量。
2. 記憶體 (GiB) — 配置給執行個體的 RAM，以 GB 為單位。記憶體和 vCPU 之間的比率通常是固定的。
3. NVMe SSD 分層快取 — SSD 磁碟區上的空間，以 GB 為單位，配置為擴充快取以存放暫時性資料。此快取僅適用於 NVMe 支援的執行個體。
4. 最大暫存儲存 (GiB) — 配置給執行個體用於非持久性暫存檔案儲存的空間，以 GB 為單位。對於 NVMe 後端執行個體，此儲存體託管在 NVMe 型 SSD 磁碟區上。在所有其他執行個體中，它會託管在 Amazon Elastic Block Store (EBS) 上。
5. 基準/高載頻寬 (Gbps) — 高載頻寬代表每秒 GB 的最大頻寬。將 8 除以 ，以取得每秒 GB 的預期輸送量。大小為 4xlarge 或更小的執行個體具有基準頻寬。若要滿足其他需求，他們可使用網路輸入/輸出額度機制，以實現超越基準頻寬的高載頻寬。執行個體可以在有限的時間內使用高載頻寬 (通常是 5 到 60 分鐘)，具體取決於執行個體大小。
6. 支援引擎 — 支援執行個體類別的 Amazon DocumentDB 引擎。

NVMe 支援的執行個體

NVMe 後端執行個體為大型資料集超過一般執行個體記憶體的應用程式提供高達 7 倍的查詢效能。這些執行個體利用 r6gd 執行個體上可用的本機非揮發性記憶體表達式 (NVMe) 型固態硬碟 (SSD) 儲存體來存放暫時性資料、減少網路型儲存體存取，以及改善讀取延遲和輸送量。

本機 SSD 空間分為兩個部分：

- 分層快取 — 大約 73% 的本機 SSD 配置為資料庫快取，可讓系統儲存比主要記憶體多最多五倍的資料庫頁面。本機 SSD 做為第二層快取，而現有的記憶體內緩衝區快取仍為第一層快取。只有在緩衝區快取和 SSD 快取都發生錯誤時，查詢才會存取外部儲存。
- 暫時儲存 — 剩餘的 27% 保留給非持久性暫存檔案儲存，用於涉及排序或索引組建等資源密集型操作的複雜查詢。在一般執行個體上，暫時空間位於 Amazon Elastic Block Store (EBS) 磁碟區上。SSD 上的本機託管臨時儲存可減少查詢延遲，涉及排序最多兩次，並加速索引組建等資源密集型操作。

有關支援的 NVMe 後端執行個體類型及其規格的詳細資訊，請參閱 執行個體類別規格。

主題

- NVMe 後端執行個體的建議使用案例
- 搭配 Amazon DocumentDB 使用 NVMe 支援的執行個體
- 監控 NVMe 支援的執行個體

NVMe 後端執行個體的建議使用案例

在下列案例中使用 NVMe 支援的執行個體：

- 高讀取量工作負載：如果您的工作負載需要大量讀取，且資料集大於緩衝區快取，則 NVMe `BufferCacheHitRatio` `ReadIOPS` NVMe 支援的執行個體可以提供效能優勢。
- 更新密集型工作負載：如果您的工作負載需要更新，且垃圾收集因為網路儲存上的讀取延遲而無法跟上進度，NVMe 支援的執行個體可協助減輕問題。

NVMe 後端執行個體可受益於各種使用案例，包括：

- 網路規模應用程式 — 付款處理、計費和電子商務等具有嚴格效能的服務水準協議 (SLAs) 的應用程式可以利用 NVMe 後端執行個體的效能優勢。
- 即時報告儀表板 — 為指標/資料收集執行數百個查詢的儀表板可以受益於 NVMe 後端執行個體的低延遲和高輸送量。
- 生成式 AI 應用程式 — 使用向量搜尋來尋找數百萬個向量內嵌之間確切或最接近的鄰體的應用程式，可以利用 NVMe 後端執行個體的高效能。

搭配 Amazon DocumentDB 使用 NVMe 支援的執行個體

若要使用 Amazon DocumentDB 的 NVMe 後端執行個體：

- 建立 Amazon DocumentDB 叢集，並新增其中一個 NVMe 支援的執行個體類別。如需詳細資訊，請參閱建立 Amazon DocumentDB 叢集。
- 或者，修改現有的 Amazon DocumentDB 叢集，以使用其中一個 NVMe 支援的執行個體類別。如需詳細資訊，請參閱修改 Amazon DocumentDB 叢集。

若要檢查不同 AWS 區域中 NVMe 後端執行個體的可用性，請參閱 各區域的支援執行個體類別。

如果您想要從 NVMe 後端執行個體切換回一般執行個體，請將 Amazon DocumentDB 執行個體的資料庫執行個體類別修改為沒有 NVMe 儲存體的類似執行個體類別。例如，如果您目前的執行個體類別是 'db.r6gd.4xlarge'，請選擇 'db.r6g.4xlarge' 以切換回。如需詳細資訊，請參閱修改 Amazon DocumentDB 叢集。

監控 NVMe 支援的執行個體

除了 Amazon CloudWatch 中可用的一般執行個體指標之外，NVMe 後端執行個體還會發出 NVMe型 SSD 儲存體、IOPS 和輸送量專用的其他指標。

```
NVMeStorageCacheHitRatio
FreeNVMeStorage
ReadIOPSNVMeStorage
ReadLatencyNVMeStorage
ReadThroughputNVMeStorage
WriteIOPSNVMeStorage
WriteLatencyNVMeStorage
WriteThroughputNVMeStorage
```

如需這些指標的詳細資訊，請參閱 NVMe 支援的執行個體指標

管理 Amazon DocumentDB 子網路群組

虛擬私有雲端 (VPC) 是您的專用虛擬網路 AWS 帳戶。它在邏輯上與 AWS 雲端中的其他虛擬網路隔離。您可以在 Amazon VPC 中啟動 AWS 資源，例如 Amazon DocumentDB 叢集。 Amazon DocumentDB 您可以為 VPC 指定 IP 地址範圍、新增子網路、與安全群組建立關聯，以及設定路由表。

子網路是 Amazon VPC 中的 IP 地址範圍。您可以在指定的子網路中啟動 AWS 資源。針對必須連接到網際網路的資源使用「公有」子網路。針對不會連線至網際網路的資源使用私有子網路。如需公有和私有子網路的詳細資訊，請參閱《Amazon Virtual Private Cloud 使用者指南》中的 VPC 和子網路基本概念。

資料庫子網路群組是您在 VPC 中建立，然後指定給叢集之子網路的集合。子網路群組可讓您在建立叢集時指定特定的 VPC。如果您使用 `default` 子網路群組，它將橫跨 VPC 中的所有子網路。

各個資料庫子網路群組在指定 區域中，都應在至少兩個可用區域中設有子網路。在 VPC 中建立資料庫叢集時，您必須選取資料庫子網路群組。Amazon DocumentDB 使用該資料庫子網路群組和您偏好的可用區域來選取子網路和該子網路內的 IP 地址，以與您的叢集建立關聯。如果主要執行個體失

敗，Amazon DocumentDB 可以將對應的複本執行個體提升為新的主要執行個體。然後，它可以使用前一個主要執行個體所在之子網路的 IP 地址，建立新的複本執行個體。

當 Amazon DocumentDB 在 VPC 中建立執行個體時，它會使用從資料庫子網路群組中選取的 IP 地址，將網路介面指派給您的叢集。一律使用 DNS 名稱，因為基礎 IP 地址可能會在容錯移轉期間變更。如需詳細資訊，請參閱Amazon DocumentDB 端點。

如需有關建立您自己的 VPC 和子網路的資訊，請參閱《Amazon Virtual Private Cloud 使用者指南》中的使用 VPCs 和子網路。

主題

- 建立 Amazon DocumentDB 子網路群組
- 描述 Amazon DocumentDB 子網路群組
- 修改 Amazon DocumentDB 子網路群組
- 刪除 Amazon DocumentDB 子網路群組

建立 Amazon DocumentDB 子網路群組

建立 Amazon DocumentDB 叢集時，您必須選擇該 Amazon VPC 內的 Amazon VPC 和對應的子網路群組，才能啟動叢集。子網路會決定您要用來啟動執行個體之可用區域內的可用區域和 IP 範圍。

子網路群組是一組具名的子網路 （或 AZs)，可讓您指定要用來啟動 Amazon DocumentDB 執行個體的可用區域。例如，在具有三個執行個體的叢集中，建議將每個執行個體佈建在個別AZs 中，因此 會針對高可用性進行最佳化。因此，如果單一 AZ 失敗，它只會影響單一執行個體。

Amazon DocumentDB 執行個體最多可佈建在三個可用區域中。即使子網路群組有超過三個子網路，您仍只能使用其中三個子網路來建立 Amazon DocumentDB 叢集。當您建立子網路群組時，請僅選擇您要部署執行個體的三個子網路。

例如：建立叢集，Amazon DocumentDB 選擇 AZs {1A、1B 和 1C}。如果您嘗試在 AZ 建立執行個體 {1D} API 呼叫將會失敗。不過，如果您選擇建立執行個體，而不指定特定 AZ，則 Amazon DocumentDB 會代表您選擇 AZ。Amazon DocumentDB 使用演算法在 AZs 之間負載平衡執行個體，以協助您實現高可用性。如果佈建三個執行個體，預設會跨三個 AZs 佈建，而且不會在單一 AZ 中佈建全部。

最佳實務

- 除非有特定的原因，否則一律使用三個子網路建立子網路群組。這可確保具有三個或更多執行個體的叢集將能夠實現更高的可用性，因為執行個體將佈建到三個AZs。

- 一律將執行個體分散在多個可用區域以達到高可用性。請勿將叢集的所有執行個體放置在單一可用區。
- 由於隨時都會發生容錯移轉事件，您不應假設主執行個體或複本執行個體會在特定的可用區域。

如何建立子網路群組

您可以使用 AWS 管理主控台 或 AWS CLI 來建立 Amazon DocumentDB 子網路群組：

Using the AWS 管理主控台

使用下列步驟來建立 Amazon DocumentDB 子網路群組。

建立 Amazon DocumentDB 子網路群組

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Subnet groups (子網路群組)，然後選擇 Create (建立)。

   > **Tip**
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在 Create subnet group (建立子網路群組) 頁面上：

   a. 在 Subnet group details (子網路群組詳細資訊) 區段中：

      i. 名稱 — 為子網路群組輸入有意義的名稱。

      ii. Description (描述)：輸入子網路群組的描述。

   b. 在 Add subnets (新增子網路) 區段中：

      i. VPC - 在清單中，為此子網路群組選擇 VPC。

      ii. 執行以下任意一項：

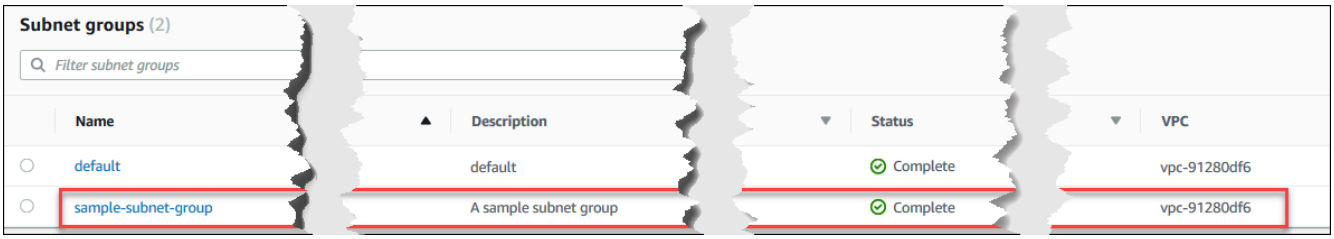
      - 若要將所有子網路包含在所選的 VPC 中，請選擇 Add all the subnets related to this VPC (新增與此 VPC 相關的所有子網路)。
      - 若要指定此子網路群組的子網路，請在想要包含子網路的每個可用區域中執行下列動作。您必須至少包含兩個可用區域。

A. 可用區域 - 在清單中，選擇可用區域。

B. 子網路 - 在清單中，從為此子網路群組選擇的可用區域選擇子網路。

C. 選擇 Add subnet (新增子網路)。

4. 選擇建立。建立子網路群組時，其會與您其他的子網路群組一起列出。



Using the AWS CLI

您必須先判斷可用的子網路 AWS CLI，才能使用 建立子網路群組。執行下列 AWS CLI 操作，列出可用區域及其子網路。

參數：

- **`--db-subnet-group`**- 選用。指定特定的子網路群組，會列出該群組的可用區域和子網路。略過此參數，會列出您所有子網路群組的可用區域和子網路。指定 `default` 子網路群組，會列出所有 VPC 的子網路。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-subnet-groups \
    --db-subnet-group-name default \
    --query 'DBSubnetGroups[*].[DBSubnetGroupName,Subnets[*].
[SubnetAvailabilityZone.Name,SubnetIdentifier]]'
```

針對 Windows：

```
aws docdb describe-db-subnet-groups ^
    --db-subnet-group-name default ^
    --query 'DBSubnetGroups[*].[DBSubnetGroupName,Subnets[*].
[SubnetAvailabilityZone.Name,SubnetIdentifier]]'
```

此操作的輸出將會如下所示 (JSON 格式)。

```
[
    [
        "default",
        [
            [
                "us-east-1a",
                "subnet-4e26d263"
            ],
            [
                "us-east-1c",
                "subnet-afc329f4"
            ],
            [
                "us-east-1e",
                "subnet-b3806e8f"
            ],
            [
                "us-east-1d",
                "subnet-53ab3636"
            ],
            [
                "us-east-1b",
                "subnet-991cb8d0"
            ],
            [
                "us-east-1f",
                "subnet-29ab1025"
            ]
        ]
    ]
]
```

使用先前操作的輸出可建立新的子網路群組。新的子網路群組必須包含至少兩個可用區域的子網路。

參數：

- **--db-subnet-group-name** - 必要項目。此子網路群組的名稱。
- **--db-subnet-group-description** - 必要項目。此子網路群組的描述。
- **--subnet-ids** - 必要項目。要包含在此子網路群組中之所有子網路的清單。範例：subnet-53ab3636。

- --Tags - 選用。要連接到此子網路群組的標籤 (金鑰值對) 清單。

以下程式碼會建立子網路群組 sample-subnet-group，其含有三個子網路 subnet-4e26d263、subnet-afc329f4 和 subnet-b3806e8f。

針對 Linux、macOS 或 Unix：

```
aws docdb create-db-subnet-group \
    --db-subnet-group-name sample-subnet-group \
    --db-subnet-group-description "A sample subnet group" \
    --subnet-ids subnet-4e26d263 subnet-afc329f4 subnet-b3806e8f \
    --tags Key=tag1,Value=One Key=tag2,Value=2
```

針對 Windows：

```
aws docdb create-db-subnet-group ^
    --db-subnet-group-name sample-subnet-group ^
    --db-subnet-group-description "A sample subnet group" ^
    --subnet-ids subnet-4e26d263 subnet-afc329f4 subnet-b3806e8f ^
    --tags Key=tag1,Value=One Key=tag2,Value=2
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "DBSubnetGroup": {
        "DBSubnetGroupDescription": "A sample subnet group",
        "DBSubnetGroupName": "sample-subnet-group",
        "Subnets": [
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1a"
                },
                "SubnetIdentifier": "subnet-4e26d263",
                "SubnetStatus": "Active"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1c"
                },
                "SubnetIdentifier": "subnet-afc329f4",
                "SubnetStatus": "Active"
```

```
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1e"
                },
                "SubnetIdentifier": "subnet-b3806e8f",
                "SubnetStatus": "Active"
            }
        ],
        "VpcId": "vpc-91280df6",
        "DBSubnetGroupArn": "arn:aws:rds:us-east-1:123SAMPLE012:subgrp:sample-
subnet-group",
        "SubnetGroupStatus": "Complete"
    }
}
```

描述 Amazon DocumentDB 子網路群組

您可以使用 AWS 管理主控台 或 AWS CLI 來取得 Amazon DocumentDB 子網路群組的詳細資訊。

Using the AWS 管理主控台

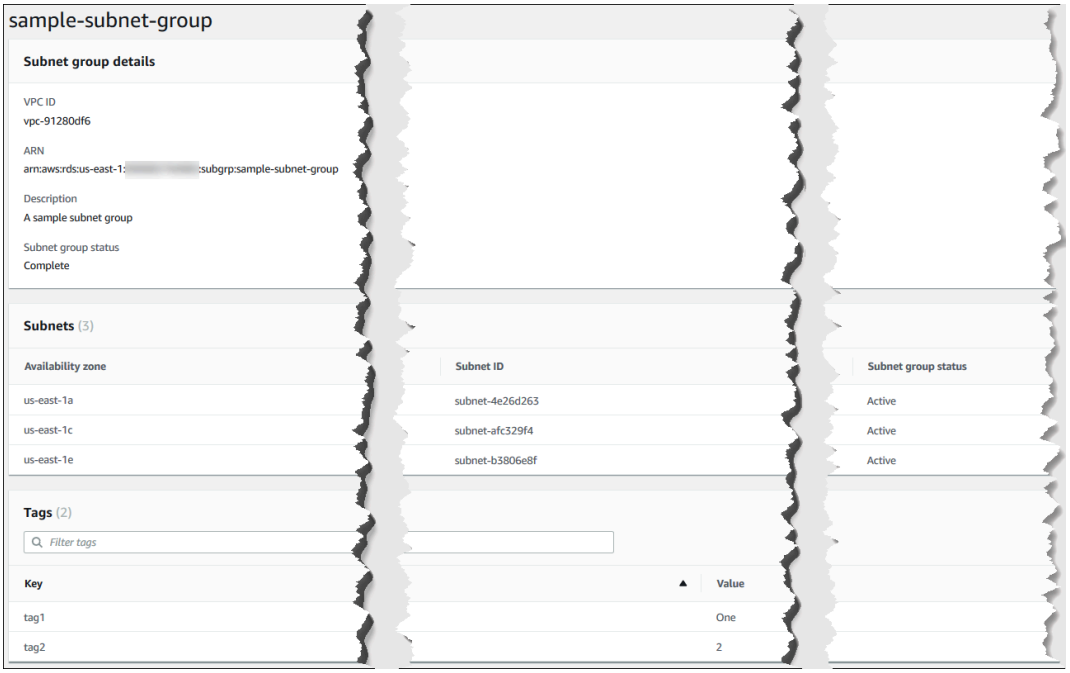
下列程序說明如何取得 Amazon DocumentDB 子網路群組的詳細資訊。

找出子網路群組的詳細資訊

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中選擇 Subnet groups (子網路群組)。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 若要查看子網路群組的詳細資訊，請選擇該子網路群組的名稱。



Using the AWS CLI

若要尋找 Amazon DocumentDB 子網路群組的詳細資訊，請使用 `describe-db-subnet-groups`操作搭配下列參數。

參數

- `--db-subnet=group-name`- 選用。如果包含，將列出指名子網路群組的詳細資訊。如果省略，將列出最多 100 個子網路群組的詳細資訊。

Example

以下程式碼將列出我們在第 建立 Amazon DocumentDB 子網路群組 節中建立的 `sample-subnet-group` 子網路群組的詳細資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-subnet-groups \
    --db-subnet-group-name sample-subnet-group
```

針對 Windows：

```
aws docdb describe-db-subnet-groups ^
```

```
    --db-subnet-group-name sample-subnet-group
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "DBSubnetGroup": {
        "DBSubnetGroupArn": "arn:aws:rds:us-east-1:123SAMPLE012:subgrp:sample-
subnet-group",
        "VpcId": "vpc-91280df6",
        "SubnetGroupStatus": "Complete",
        "DBSubnetGroupName": "sample-subnet-group",
        "Subnets": [
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1a"
                },
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-4e26d263"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1c"
                },
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-afc329f4"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1e"
                },
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-b3806e8f"
            }
        ],
        "DBSubnetGroupDescription": "A sample subnet group"
    }
}
```

修改 Amazon DocumentDB 子網路群組

您可以使用 AWS 管理主控台 或 AWS CLI 來修改子網路群組的描述，或從 Amazon DocumentDB 子網路群組新增或移除子網路。不過，您無法修改 `default` 子網路群組。

Using the AWS 管理主控台

您可以使用 AWS 管理主控台 來變更子網路群組的描述，或新增和移除子網路。請記住，完成後，您必須至少有兩個可用區域與子網路群組關聯。

修改子網路群組

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中選擇 Subnet groups (子網路群組)。然後，選擇子網路群組名稱左側的按鈕。請記住，您無法修改 `default` 子網路群組。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 選擇 Actions (動作)，然後選擇 Modify (修改)。
4. 描述 - 若要變更子網路群組的描述，請輸入新的描述。
5. 若要變更與子網路群組相關聯的子網路，請在 Add subnets (新增子網路) 區段中執行下列其中一個或多個動作：
   - 若要移除此子網路群組中的所有子網路，選擇 Remove all (全部移除)。
   - 若要從此子網路群組中移除特定的子網路，請為每個想要移除的子網路選擇 Remove (移除)。
   - 若要新增與此 VPC 相關的所有子網路，請選擇 Add all the subnets related to this VPC (新增與此 VPC 相關的所有子網路)。
   - 若要新增特定子網路至子網路群組，請為想要新增子網路的每個可用區域執行下列動作。
     a. 可用區域 - 在清單中，選擇新的可用區域。
     b. 子網路 - 在清單中，從為此子網路群組選擇的可用區域選擇子網路。
     c. 選擇 Add subnet (新增子網路)。
6. 在確認對話方塊中：
   - 若要對子網路群組進行這些變更，請選擇 Modify (修改)。
   - 若要保持子網路群組不變，請選擇 Cancel (取消)。

Using the AWS CLI

您可以使用 AWS CLI 來變更子網路群組的描述，或新增和移除子網路。請記住，完成後，您必須至少有兩個可用區域與子網路群組關聯。您無法修改 `default` 子網路群組。

參數：

- `--db-subnet-group-name` - 必要項目。您要修改的 Amazon DocumentDB 子網路群組名稱。
- `--subnet-ids` - 必要項目。完成此變更之後，您想要包含在子網路群組內的所有子網路的清單。

> **Important**
>
> 目前在子網路群組中但未包含在此清單內任何子網路將從子網路群組移除。如果您想要保留目前在子網路群組中的任何子網路，您必須將其加入到清單內。

- `--db-subnet-group-description`- 選用。子網路群組的描述。

Example

以下程式碼將修改描述並用子網路 `subnet-991cb8d0`、`subnet-53ab3636` 及 `subnet-29ab1025` 取代現有的子網路。

針對 Linux、macOS 或 Unix：

```
aws docdb modify-db-subnet-group \
    --db-subnet-group-name sample-subnet-group \
    --subnet-ids subnet-991cb8d0 subnet-53ab3636 subnet-29ab1025 \
    --db-subnet-group-description "Modified subnet group"
```

針對 Windows：

```
aws docdb modify-db-subnet-group ^
    --db-subnet-group-name sample-subnet-group ^
    --subnet-ids subnet-991cb8d0 subnet-53ab3636 subnet-29ab1025 ^
    --db-subnet-group-description "Modified subnet group"
```

此操作的輸出將會如下所示 (JSON 格式)。請注意，這是已在建立 Amazon DocumentDB 子網路群組一節中建立的同一個子網路群組。不過，子網路群組的子網路會取代為 modify-db-subnet-group 操作中列出的子網路。

```
{
    "DBSubnetGroup": {
        "DBSubnetGroupArn": "arn:aws:rds:us-east-1:123SAMPLE012:subgrp:sample-
subnet-group",
        "DBSubnetGroupDescription": "Modified subnet group",
        "SubnetGroupStatus": "Complete",
        "Subnets": [
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1d"
                },
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-53ab3636"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1b"
                },
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-991cb8d0"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1f"
                },
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-29ab1025"
            }
        ],
        "VpcId": "vpc-91280df6",
        "DBSubnetGroupName": "sample-subnet-group"
    }
}
```

刪除 Amazon DocumentDB 子網路群組

您可以使用 AWS 管理主控台 或 AWS CLI 來刪除 Amazon DocumentDB 子網路群組。不過，您無法刪除 `default` 子網路群組。

Using the AWS 管理主控台

您可以使用 AWS 管理主控台 來刪除子網路群組。但是，您無法刪除 `default` 子網路群組。

刪除子網路群組

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中選擇 Subnet groups (子網路群組)。然後，選擇子網路群組名稱左側的按鈕。請注意，您無法刪除 `default` 子網路群組。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 選擇動作，然後選擇刪除。
4. 在確認對話方塊中：
   - 若要刪除子網路群組，請選擇 Delete (刪除)。
   - 若要保留子網路群組，請選擇 Cancel (取消)。

Using the AWS CLI

若要使用 刪除 Amazon DocumentDB 子網路群組 AWS CLI，請使用 `delete-db-subnet-group`操作搭配下列參數。

參數

- `--db-subnet-group-name` - 必要項目。要刪除的 Amazon DocumentDB 子網路群組名稱。請注意，您無法刪除 `default` 子網路群組。

Example

以下程式碼會刪除 `sample-subnet-group`。

針對 Linux、macOS 或 Unix：

```
aws docdb delete-db-subnet-group \
    --db-subnet-group-name sample-subnet-group
```

針對 Windows：

```
aws docdb delete-db-subnet-group ^
    --db-subnet-group-name sample-subnet-group
```

此操作不會產生輸出。

Amazon DocumentDB 高可用性和複寫

您可以使用複本執行個體，在 Amazon DocumentDB 中實現高可用性和讀取擴展（與 MongoDB 相容）。單一 Amazon DocumentDB 叢集支援單一主要執行個體和最多 15 個複本執行個體。這些執行個體可以分佈在叢集區域內的可用區域之中。主要執行個體接受讀取和寫入流量，而複寫執行個體僅接受讀取請求。

叢集磁碟區是由叢集的多個資料複本組成。不過，叢集磁碟區中的資料會以單一邏輯磁碟區表示，傳送至主要執行個體和叢集中的 Amazon DocumentDB 複本。複本執行個體是最終一致性 它們會透過最低複寫延遲 (通常在主要執行個體寫入更新後短於 100 毫秒) 來傳回查詢結果。複本延遲會根據資料庫變更率而有所不同。也就是，在資料庫發生大量寫入操作期間，您可能會發現複本延遲增加。

讀取擴展

Amazon DocumentDB 複本非常適合讀取擴展，因為它們完全專用於叢集磁碟區上的讀取操作。寫入操作由主要執行個體管理。叢集中的所有執行個體將共用叢集磁碟區。因此，您不需要複寫和維護每個 Amazon DocumentDB 複本的資料副本。

高可用性

當您建立 Amazon DocumentDB 叢集時，取決於子網路群組中的可用區域數量（必須至少有兩個），Amazon DocumentDB 會跨可用區域佈建執行個體。當您在叢集中建立執行個體時，Amazon DocumentDB 會自動將執行個體分散到子網路群組中的可用區域，以平衡叢集。這個動作還可防止所有執行個體位於相同的可用區域。

範例

為說明要點，假設我們建立含有一個子網路群組，且該子網路群組有三個可用區域 (AZ1、AZ2 和 AZ3) 的叢集。

當叢集的第一個執行個體建立完成後，其為主執行個體，且位於其中一個可用區域。在此範例中，其位於 AZ1。建立的第二個執行個體為複本執行個體，其位於另外兩個可用區域其中之一，假設為 AZ2。建立的第三個執行個體為複本執行個體，其位於剩餘的可用區域，即 AZ3。如果您建立多個執行個體，它們會分佈在可用區域之中，讓您可在叢集中達到平衡。

如果故障發生在主執行個體 (AZ1)，將觸發容錯移轉，且其中一個現有的複本會提升至主要執行個體。當舊的主要執行個體恢復時，它會在佈建 (AZ1) 的相同可用區域中成為複本。當您佈建三執行個體叢集時，Amazon DocumentDB 會繼續保留該三執行個體叢集。Amazon DocumentDB 會自動處理執行個體故障的偵測、容錯移轉和復原，無需任何手動介入。

當 Amazon DocumentDB 執行容錯移轉並復原執行個體時，復原的執行個體會保留在原先佈建的可用區域中。不過，執行個體角色可能從主要變更為複本。這樣做可防止一連串的容錯移轉導致所有執行個體出現在同一個可用區域內。

您可以將 Amazon DocumentDB 複本指定為容錯移轉目標。也就是說，如果主要執行個體失敗，則從層指定的 Amazon DocumentDB 複本或複本會提升為主要執行個體。在對主要執行個體提出的讀取和寫入請求由於例外狀況而失敗的期間，會發生短暫的中斷。如果您的 Amazon DocumentDB 叢集不包含任何 Amazon DocumentDB 複本，當主要執行個體失敗時，會重新建立。提升 Amazon DocumentDB 複本比重新建立主要執行個體快得多。

為了獲得高可用性，請建立一或多個與不同可用區域中主要執行個體相同執行個體類別的 Amazon DocumentDB 複本。

如需詳細資訊，請參閱下列內容：

- 了解 Amazon DocumentDB 叢集容錯能力
- Amazon DocumentDB 容錯移轉
  - 控制容錯移轉目標

在 Amazon DocumentDB 5.0 和 8.0 版中，複本執行個體不會在主要執行個體重新啟動時重新啟動。它們會在主要執行個體重新啟動期間繼續提供讀取請求，以維持改善的可用性。

全球叢集的高可用性

如需跨多個叢集的高可用性 AWS 區域，您可以設定 Amazon DocumentDB 全域叢集。每個全域叢集橫跨多個區域，可實現低延遲的全域讀取，以及跨區域中斷的災難復原。Amazon DocumentDB 會自動將所有資料和更新從主要區域複寫到每個次要區域。

新增 複本

叢集中新增的第一個執行個體即為主要執行個體。在第一個執行個體後新增的每個執行個體即為複寫執行個體。除了主要叢集之外，叢集最多可以有 15 個複本執行個體。

當您使用 建立叢集時 AWS 管理主控台，會自動同時建立主要執行個體。若要在您建立叢集和主要執行個體的同時建立複本，請選擇 Create replica in different zone (在不同區域中建立複本)。如需詳細資訊，請參閱 建立 Amazon DocumentDB 叢集 中的步驟 4.d。若要將更多複本新增至 Amazon DocumentDB 叢集，請參閱 將 Amazon DocumentDB 執行個體新增至叢集。

使用 AWS CLI 建立叢集時，您必須明確建立主要執行個體和複本執行個體。如需詳細資訊，請參閱下列主題中的「使用 AWS CLI」一節：

- 建立 Amazon DocumentDB 叢集
- 將 Amazon DocumentDB 執行個體新增至叢集

Amazon DocumentDB 容錯移轉

在某些情況下，例如特定類型的計劃維護，或在不太可能發生主節點或可用區域故障的情況下，Amazon DocumentDB （具有 MongoDB 相容性） 會偵測故障並取代主節點。容錯移轉期間，寫入時間會降至最低。這是因為主要節點的角色會容錯移轉至其中一個僅供讀取複本，而非必須建立並佈建新的主要節點。此故障偵測及複本提升可確保您能在提升完成時立即繼續寫入新的主要節點。

若要讓容錯移轉運作，您的叢集必須至少有兩個執行個體：主要執行個體和至少一個複本執行個體。

> **Note**
>
> 本主題僅適用於原始 Amazon DocumentDB 執行個體型叢集。它不適用於彈性或全域叢集。

控制容錯移轉目標

Amazon DocumentDB 為您提供容錯移轉層，以控制發生容錯移轉時要將哪些複本執行個體提升為主要執行個體。

容錯移轉方案

每個複本執行個體都與容錯移轉層 (0–15) 相關聯。當由於維護或不太可能的硬體故障而發生容錯移轉時，主要執行個體會容錯移轉至具有最高優先順序 （編號最低的層） 的複本。如果多個複本具有相同的優先順序層，則主要 會容錯移轉到該層的複本，該複本的大小與上一個主要複本最接近。

選擇一組複本，並將其容錯移轉方案設定為 0 (最高優先順序)，就可以確保容錯移轉會提升到該群組的其中一個複本。為複本指派低優先順序方案 (數字較大)，就可以有效防止容錯移轉發生時，將特定複本提升成主要執行個體。這在特定複本接收來自應用程式的繁重使用情形時非常有用，容錯移轉至這些複本的其中一個將會對關鍵應用程式造成負面影響。

我們建議作為潛在容錯移轉目標的複本使用與主要執行個體相同的執行個體類別。如果容錯移轉在作業系統更新或其他維護期間提升複本，這可避免寫入效能降低。

您可以在建立執行個體時設定容錯移轉方案，或在稍後進行修改。藉由修改執行個體來設定執行個體容錯移轉方案，並不會觸發容錯移轉。如需詳細資訊，請參閱下列主題：

- 將 Amazon DocumentDB 執行個體新增至叢集
- 修改 Amazon DocumentDB 執行個體

手動啟動容錯移轉時，您有兩種方式來控制要將哪個執行個體複本提升為主要執行個體：透過前面所述的容錯移轉方案和 `--target-db-instance-identifier` 參數。

`--target-db-instance-identifier`

若要進行測試，您可以使用 `failover-db-cluster` 操作強制容錯移轉事件發生。您可以使用 `--target-db-instance-identifier` 參數來指定將哪些複本提升成主要執行個體。使用 `--target-db-instance-identifier` 參數會取代容錯移轉優先順序方案。如果您未指定 `--target-db-instance-identifier` 參數，主要容錯移轉會根據容錯移轉優先順序方案。

容錯移轉期間會發生什麼情況

Amazon DocumentDB 會自動處理容錯移轉，讓您的應用程式可以盡快恢復資料庫操作，而無需管理介入。

- 如果您在容錯移轉時在相同或不同的可用區域中有 Amazon DocumentDB 複本執行個體：Amazon DocumentDB 會翻轉執行個體的正式名稱記錄 (CNAME)，以指向運作狀態良好的複本，進而提升為新的主要複本。容錯移轉從開始到結束通常可在 30 秒內完成。
- 如果您沒有 Amazon DocumentDB 複本執行個體 （例如，單一執行個體叢集）：Amazon DocumentDB 會嘗試在與原始執行個體相同的可用區域中建立新的執行個體。已盡力進行這種原始執行個體的取代操作，但可能不成功，例如，在出現會廣泛影響可用區域的問題時。

您的應用程式應該會在發生連線中斷時重試資料庫連線。

測試容錯移轉

叢集的容錯移轉會將叢集中的其中一個 Amazon DocumentDB 複本 （唯讀執行個體） 提升為主要執行個體 （叢集寫入器）。

當主要執行個體失敗時，如果存在 Amazon DocumentDB 複本，Amazon DocumentDB 會自動容錯移轉至 Amazon DocumentDB 複本。當您想要模擬主要執行個體故障以進行測試時，可以強制容錯移轉。叢集中的每個執行個體都有自己的端點地址。因此，當容錯移轉完成時，您需要清除和重新建立任何使用這些端點地址的現有連線。

若要強制容錯移轉，請搭配這些參數使用 `failover-db-cluster` 操作。

- `--db-cluster-identifier` - 必要項目。要容錯移轉之叢集的名稱。
- `--target-db-instance-identifier`- 選用。要提升為主要執行個體之執行個體的名稱。

Example

以下操作會強制容錯移轉 `sample-cluster` 叢集。它不會指定要讓哪個執行個體成為新的主要執行個體，因此 Amazon DocumentDB 會根據容錯移轉層優先順序選擇執行個體。

針對 Linux、macOS 或 Unix：

```
aws docdb failover-db-cluster \
   --db-cluster-identifier sample-cluster
```

針對 Windows：

```
aws docdb failover-db-cluster ^
   --db-cluster-identifier sample-cluster
```

以下操作會強制容錯移轉 `sample-cluster` 叢集，指定 `sample-cluster-instance` 為要提升的主要角色。(請注意輸出中的 `"IsClusterWriter": true`。)

針對 Linux、macOS 或 Unix：

```
aws docdb failover-db-cluster \
   --db-cluster-identifier sample-cluster \
   --target-db-instance-identifier sample-cluster-instance
```

針對 Windows：

```
aws docdb failover-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --target-db-instance-identifier sample-cluster-instance
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "DBCluster": {
        "HostedZoneId": "Z2SUY0A1719RZT",
        "Port": 27017,
        "EngineVersion": "3.6.0",
        "PreferredMaintenanceWindow": "thu:04:05-thu:04:35",
        "BackupRetentionPeriod": 1,
        "ClusterCreateTime": "2018-06-28T18:53:29.455Z",
        "AssociatedRoles": [],
        "DBSubnetGroup": "default",
        "MasterUsername": "master-user",
        "Engine": "docdb",
        "ReadReplicaIdentifiers": [],
        "EarliestRestorableTime": "2018-08-21T00:04:10.546Z",
        "DBClusterIdentifier": "sample-cluster",
        "ReaderEndpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "DBClusterMembers": [
            {
                "DBInstanceIdentifier": "sample-cluster-instance",
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1,
                "IsClusterWriter": true
            },
            {
                "DBInstanceIdentifier": "sample-cluster-instance-00",
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1,
                "IsClusterWriter": false
            },
            {
                "DBInstanceIdentifier": "sample-cluster-instance-01",
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1,
                "IsClusterWriter": false
            }
        ],
        "AvailabilityZones": [
```

```
            "us-east-1b",
            "us-east-1c",
            "us-east-1a"
        ],
        "DBClusterParameterGroup": "default.docdb3.6",
        "Endpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "IAMDatabaseAuthenticationEnabled": false,
        "AllocatedStorage": 1,
        "LatestRestorableTime": "2018-08-22T21:57:33.904Z",
        "PreferredBackupWindow": "00:00-00:30",
        "StorageEncrypted": false,
        "MultiAZ": true,
        "Status": "available",
        "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster",
        "VpcSecurityGroups": [
            {
                "Status": "active",
                "VpcSecurityGroupId": "sg-12345678"
            }
        ],
        "DbClusterResourceId": "cluster-ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    }
}
```

複寫延遲

複寫延遲通常為 50 毫秒或更短。複本延遲增加的最常見原因是：

- 主要 上的高寫入速率，會導致僅供讀取複本落後主要複本。
- 長時間執行查詢 （例如，大型循序掃描、彙總查詢） 和傳入寫入複寫之間的僅供讀取複本爭用。
- 僅供讀取複本上非常大量的並行查詢。

若要將複寫延遲降至最低，請嘗試下列疑難排解技術：

- 如果您有高寫入速率或高 CPU 使用率，請擴展叢集中的執行個體。
- 如果您的僅供讀取複本上有長時間執行的查詢，並且非常頻繁地更新正在查詢的文件，請考慮更改長時間執行的查詢，或對主要/寫入複本執行查詢，以避免僅供讀取複本發生爭用。
- 如果僅供讀取複本上有非常大量的並行查詢或高 CPU 使用率，則另一個選項是擴展讀取複本數量以分散工作負載。

- 由於複寫延遲來自高寫入輸送量和長時間執行的查詢，因此請使用 `DBClusterReplicaLagMaximum` CloudWatch 指標搭配慢查詢記錄器和 `WriteThroughput`/`WriteIOPS` 指標來疑難排解複寫延遲。

確保您的所有複本都屬於相同的執行個體類型，以便叢集容錯移轉不會導致效能降低。

如果您選擇向上擴展和向外擴展（例如，6 個較小的執行個體與 3 個較大的執行個體），通常建議您在向外擴展之前先嘗試向上擴展（較大的執行個體），因為每個資料庫執行個體都會獲得較大的緩衝快取。

主動設定複寫延遲警示，並將其閾值設定為您認為在複本執行個體上的資料開始影響應用程式功能之前可落後（或過時）多遠的值上限。一般而言，由於暫時性工作負載，我們建議在發出警示之前超過數個資料點的複寫延遲閾值。

> **Note**
>
> 針對超過 10 秒的複寫延遲設定另一個警示。如果您針對多個資料點超過此閾值，請擴展執行個體或降低主要執行個體的寫入輸送量。

管理 Amazon DocumentDB 事件

Amazon DocumentDB（與 MongoDB 相容）會保留與叢集、執行個體、快照、安全群組和叢集參數群組相關的事件記錄。該資訊會包含事件的日期和時間、事件的來源名稱和來源類型，以及與事件相關的訊息。

> **Important**
>
> 對於某些管理功能，Amazon DocumentDB 使用與 Amazon RDS 和 Amazon Neptune 共用的操作技術。區域限制、區域層級管理的限制，會在 Amazon DocumentDB、Amazon RDS 和 Amazon Neptune 之間共用。如需詳細資訊，請參閱區域配額。

主題

- 檢視 Amazon DocumentDB 事件類別
- 檢視 Amazon DocumentDB 事件

檢視 Amazon DocumentDB 事件類別

每個 Amazon DocumentDB 資源類型都有與其相關聯的特定事件類型。您可以使用 AWS CLI `describe-event-categories`操作來檢視事件類型與 Amazon DocumentDB 資源類型之間的映射。

Parameters

- **`--source-type`**- 選用。使用 `--source-type` 參數查看特定來源類型的事件類別。以下是允許值：
  - `db-cluster`
  - `db-instance`
  - `db-parameter-group`
  - `db-security-group`
  - `db-cluster-snapshot`
- **`--filters`**- 選用。若要僅檢視 Amazon DocumentDB 的事件類別，請使用篩選條件 `--filter Name=engine,Values=docdb`。

Example

以下程式碼會列出與叢集關聯的事件類別。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-event-categories \
    --filter Name=engine,Values=docdb \
    --source-type db-cluster
```

針對 Windows：

```
aws docdb describe-event-categories ^
    --filter Name=engine,Values=docdb ^
    --source-type db-cluster
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
```

```
    "EventCategoriesMapList": [
        {
            "EventCategories": [
                "notification",
                "failure",
                "maintenance",
                "failover"
            ],
            "SourceType": "db-cluster"
        }
    ]
}
```

下列程式碼列出與每個 Amazon DocumentDB 來源類型相關聯的事件類別。

```
aws docdb describe-event-categories
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "EventCategoriesMapList": [
        {
            "SourceType": "db-instance",
            "EventCategories": [
                "notification",
                "failure",
                "creation",
                "maintenance",
                "deletion",
                "recovery",
                "restoration",
                "configuration change",
                "read replica",
                "backtrack",
                "low storage",
                "backup",
                "availability",
                "failover"
            ]
        },
        {
            "SourceType": "db-security-group",
            "EventCategories": [
```

```
                "configuration change",
                "failure"
            ]
        },
        {
            "SourceType": "db-parameter-group",
            "EventCategories": [
                "configuration change"
            ]
        },
        {
            "SourceType": "db-cluster",
            "EventCategories": [
                "notification",
                "failure",
                "maintenance",
                "failover"
            ]
        },
        {
            "SourceType": "db-cluster-snapshot",
            "EventCategories": [
                "backup"
            ]
        }
    ]
}
```

檢視 Amazon DocumentDB 事件

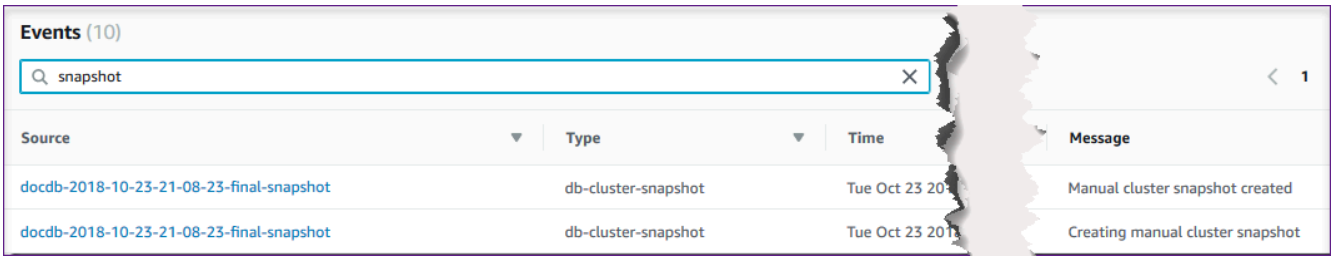
您可以透過 Amazon DocumentDB 主控台擷取 Amazon DocumentDB 資源的事件，此主控台會顯示過去 24 小時內的事件。您也可以使用 describe-events AWS CLI 命令或 DescribeEvents Amazon DocumentDB API 操作，擷取 Amazon DocumentDB 資源的事件。如果您使用 AWS CLI 或 Amazon DocumentDB API 來檢視事件，您最多可以擷取過去 14 天內的事件。

Using the AWS 管理主控台

檢視過去 24 小時的所有 Amazon DocumentDB 執行個體事件

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/docdb：// 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Events (事件)。可用的事件隨即會出現在清單中。

3. 使用 Filter (篩選條件) 清單可依類型篩選事件。在文字方塊中輸入期限，以進一步篩選結果。例如，下列螢幕擷取畫面顯示篩選快照事件的所有 Amazon DocumentDB 事件。



Using the AWS CLI

檢視過去 7 天的所有 Amazon DocumentDB 執行個體事件

您可以在 `--duration` 參數設定為 `10080`(10，080 分鐘） 的情況下執行 describe-events AWS CLI 操作，以檢視過去 7 天內的所有 Amazon DocumentDB 執行個體事件。

```
aws docdb describe-events --duration 10080
```

篩選 Amazon DocumentDB 事件

若要查看特定 Amazon DocumentDB 事件，請使用 `describe-events`操作搭配下列參數。

Parameters

- **`--filter`**- 需要將傳回的值限制為 Amazon DocumentDB 事件。使用 **`Name=engine,Values=docdb`** 僅篩選 Amazon DocumentDB 的所有事件。
- **`--source-identifier`**- 選用。要傳回其事件的事件來源識別碼。如果省略，則會傳回所有來源的事件。
- **`--source-type`**- 選用，除非`--source-identifier`提供 ，否則為必要。如果提供 `--source-identifier`，則 `--source-type` 必須和 `--source-identifier` 的類型一致。以下是允許值：
  - `db-cluster`
  - `db-instance`
  - `db-parameter-group`
  - `db-security-group`
  - `db-cluster-snapshot`

下列範例列出所有 Amazon DocumentDB 事件。

```
aws docdb describe-events --filters Name=engine,Values=docdb
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "Events": [
        {
            "SourceArn": "arn:aws:rds:us-east-1:123SAMPLE012:db:sample-cluster-
instance3",
            "Message": "instance created",
            "SourceType": "db-instance",
            "Date": "2018-12-11T21:17:40.023Z",
            "SourceIdentifier": "sample-cluster-instance3",
            "EventCategories": [
                "creation"
            ]
        },
        {
            "SourceArn": "arn:aws:rds:us-
east-1:123SAMPLE012:db:docdb-2018-12-11-21-08-23",
            "Message": "instance shutdown",
            "SourceType": "db-instance",
            "Date": "2018-12-11T21:25:01.245Z",
            "SourceIdentifier": "docdb-2018-12-11-21-08-23",
            "EventCategories": [
                "availability"
            ]
        },
        {
            "SourceArn": "arn:aws:rds:us-
east-1:123SAMPLE012:db:docdb-2018-12-11-21-08-23",
            "Message": "instance restarted",
            "SourceType": "db-instance",
            "Date": "2018-12-11T21:25:11.441Z",
            "SourceIdentifier": "docdb-2018-12-11-21-08-23",
            "EventCategories": [
                "availability"
            ]
        }
    ]
}
```

如需詳細資訊，請參閱稽核 Amazon DocumentDB 事件。

管理 Amazon DocumentDB 叢集參數群組

您可以在叢集參數群組中使用參數來管理 Amazon DocumentDB 引擎組態。叢集參數群組是 Amazon DocumentDB 組態值的集合，可讓您更輕鬆地管理 Amazon DocumentDB 叢集的參數。叢集參數群組扮演引擎組態值的容器，以套用至資料庫叢集中的所有執行個體。

本節說明如何建立、檢視和修改叢集參數群組。此外也會為您示範如何判斷有哪些叢集參數群組與指定的叢集相關。

主題

- 描述 Amazon DocumentDB 叢集參數群組
- 建立 Amazon DocumentDB 叢集參數群組
- 修改 Amazon DocumentDB 叢集參數群組
- 修改 Amazon DocumentDB 叢集以使用自訂叢集參數群組
- 複製 Amazon DocumentDB 叢集參數群組
- 重設 Amazon DocumentDB 叢集參數群組
- 刪除 Amazon DocumentDB 叢集參數群組
- Amazon DocumentDB 叢集參數參考

描述 Amazon DocumentDB 叢集參數群組

當您在新的 中建立第一個 Amazon DocumentDB 叢集 AWS 區域 或使用新的引擎時，系統會自動建立`default`叢集參數群組。後續叢集是在相同區域中建立且具有相同引擎版本的叢集會使用`default`叢集參數群組建立。

主題

- 描述 Amazon DocumentDB 叢集參數群組的詳細資訊
- 判斷 Amazon DocumentDB 叢集的參數群組
- 判斷與 Amazon DocumentDB 叢集參數群組相關聯的叢集和執行個體

描述 Amazon DocumentDB 叢集參數群組的詳細資訊

若要描述指定叢集參數群組的詳細資訊，請使用 AWS 管理主控台 或 AWS Command Line Interface () 完成下列步驟AWS CLI。

Using the AWS 管理主控台

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在 Cluster parameter groups (叢集參數群組) 窗格中，選取您想要查看詳細資訊的參數群組名稱。
4. 產生的頁面會顯示參數群組的參數、最近的活動和標籤。

   - 在 Cluster parameters (叢集參數) 下，您可以看到參數名稱、目前值、允許值、參數是否可修改、套用類型、資料類型和說明。您可以選取 參數，然後在叢集參數區段中選擇編輯，以修改個別參數。 如需詳細資訊，請參閱修改 Amazon DocumentDB 叢集參數。
   - 在 Recent events (最近的事件) 中，您可以查看此參數群組的最近事件。您可以使用本節中的搜尋列來篩選這些事件。如需詳細資訊，請參閱管理 Amazon DocumentDB 事件。
   - 在 Tags (標籤)，您可以查看在此叢集參數群組上的標籤。您可以在標籤區段中選擇編輯來新增或移除標籤。如需詳細資訊，請參閱標記 Amazon DocumentDB 資源。

Using the AWS CLI

您可以使用 `describe-db-cluster-parameter-groups` AWS CLI 命令來檢視 Amazon DocumentDB 的單一叢集參數群組或所有叢集參數群組的 Amazon Resource Name (ARN)、系列、描述和名稱。您也可以使用 `describe-db-cluster-parameters` AWS CLI 命令，在單一叢集參數群組中檢視參數及其詳細資訊。

- **`--describe-db-cluster-parameter-groups`** — 查看所有叢集參數群組及其詳細資訊的清單。

- **--db-cluster-parameter-group-name** — 選用。您想要說明的叢集參數群組名稱。如果省略這個參數，所有叢集參數群組會加上說明。

- **--describe-db-cluster-parameters** — 列出參數群組內的所有參數及其值。
  - **--db-cluster-parameter-group name** – 必要。您想要說明的叢集參數群組名稱。

Example

以下程式碼列出高達 100 個叢集參數群組及其 ARN、系列、說明和名稱。

```
aws docdb describe-db-cluster-parameter-groups
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
          "DBClusterParameterGroups": [
              {
                  "DBClusterParameterGroupArn": "arn:aws:rds:us-
east-1:012345678912:cluster-pg:default.docdb4.0",
                  "DBParameterGroupFamily": "docdb4.0",
                  "Description": "Default cluster parameter group for docdb4.0",
                  "DBClusterParameterGroupName": "default.docdb4.0"
              },
              {
                  "DBClusterParameterGroupArn": "arn:aws:rds:us-
east-1:012345678912:cluster-pg:sample-parameter-group",
                  "DBParameterGroupFamily": "docdb4.0",
                  "Description": "Custom docdb4.0 parameter group",
                  "DBClusterParameterGroupName": "sample-parameter-group"
              }
          ]
}
```

Example

以下程式碼列出 `sample-parameter-group` 的 ARN、系列、說明和名稱。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameter-groups \
```

```
          --db-cluster-parameter-group-name sample-parameter-group
```

針對 Windows：

```
aws docdb describe-db-cluster-parameter-groups ^
          --db-cluster-parameter-group-name sample-parameter-group
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
          "DBClusterParameterGroups": [
              {
                  "DBClusterParameterGroupArn": "arn:aws:rds:us-
east-1:123456789012:cluster-pg:sample-parameter-group",
                  "Description": "Custom docdb4.0 parameter group",
                  "DBParameterGroupFamily": "docdb4.0",
                  "DBClusterParameterGroupName": "sample-parameter-group"
              }
          ]
}
```

Example

以下程式碼列出 sample-parameter-group 中參數的值。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

針對 Windows：

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name sample-parameter-group
```

此操作的輸出將會如下所示 (JSON 格式)。

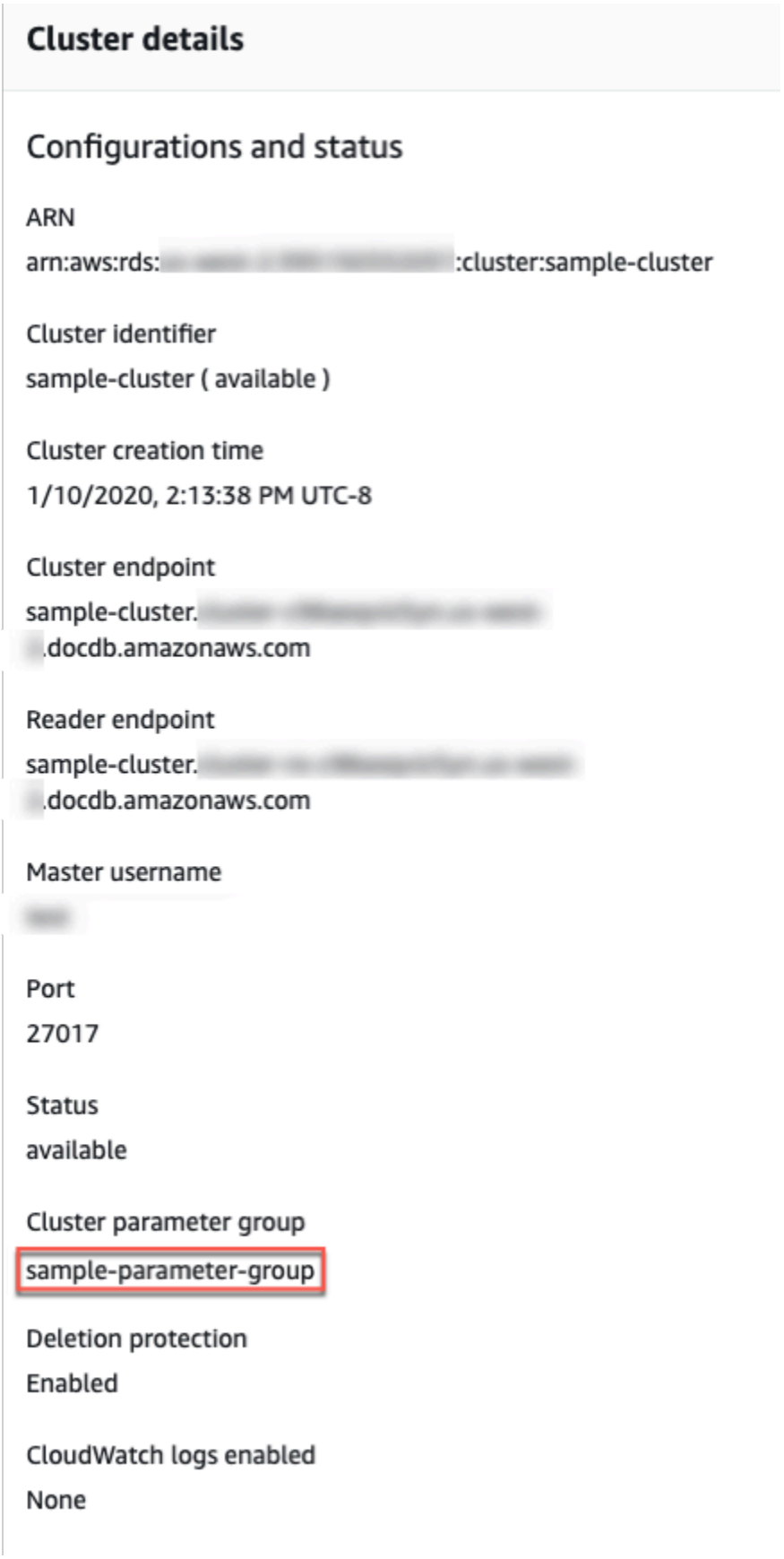
```
{
   "Parameters": [
         {
            "ParameterName": "audit_logs",
            "ParameterValue": "disabled",
            "Description": "Enables auditing on cluster.",
            "Source": "system",
            "ApplyType": "dynamic",
            "DataType": "string",
            "AllowedValues": "enabled,disabled",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot"
         },
         {
            "ParameterName": "change_stream_log_retention_duration",
            "ParameterValue": "17777",
            "Description": "Duration of time in seconds that the change stream log
 is retained and can be consumed.",
            "Source": "user",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "3600-86400",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot"
         }
   ]
}
```

判斷 Amazon DocumentDB 叢集的參數群組

若要判斷哪些參數群組與特定叢集相關聯，請使用 AWS 管理主控台 或 完成下列步驟 AWS CLI。

Using the AWS 管理主控台

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在左側導覽窗格中選擇叢集。
3. 在叢集的清單中，選取您有興趣之叢集的名稱。
4. 產生的頁面會顯示您所選取叢集的詳細資訊。向下捲動至 Cluster details (叢集詳細資訊)。在該區段底部的 Cluster parameter group (叢集參數群組) 下找到參數群組的名稱。


Cluster details
Configurations and status
ARN
arn:aws:rds: :cluster:sample-cluster
Cluster identifier
sample-cluster (available)
Cluster creation time
1/10/2020, 2:13:38 PM UTC-8
Cluster endpoint
sample-cluster.
.docdb.amazonaws.com
Reader endpoint
sample-cluster.
.docdb.amazonaws.com
Master username
Port
27017
Status
available
Cluster parameter group
sample-parameter-group
Deletion protection
Enabled
CloudWatch logs enabled
None

Using the AWS CLI

下列 AWS CLI 程式碼會決定管理叢集 的參數群組sample-cluster。

```
aws docdb describe-db-clusters \
      --db-cluster-identifier sample-cluster \
      --query 'DBClusters[*].[DBClusterIdentifier,DBClusterParameterGroup]'
```

此操作的輸出將會如下所示 (JSON 格式)。

```
[
       [
           "sample-cluster",
           "sample-parameter-group"
       ]
]
```

判斷與 Amazon DocumentDB 叢集參數群組相關聯的叢集和執行個體

若要判斷有多少叢集和執行個體與每個參數群組相關聯，請使用 完成以下步驟 AWS CLI。

Using the AWS CLI

下列 AWS CLI 程式碼會決定有多少叢集和執行個體與 相關聯sample-parameter-group。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters \
      --query 'DBClusters[*].
[DBClusterParameterGroup,DBClusterIdentifier,DBClusterMembers[*].DBInstanceIdentifier]'
 \
      --output json | \
      jq -r 'group_by(.[0]) | map({
         parameter_group_name: .[0][0],
         total_clusters: length,
         total_instances: map(.[2] | length) | add // 0
      }) | .[]'
```

針對 Windows：

```
aws docdb describe-db-clusters ^
```

```
      --query 'DBClusters[*].
[DBClusterParameterGroup,DBClusterIdentifier,DBClusterMembers[*].DBInstanceIdentifier]'
 ^
      --output json | ^
      jq -r 'group_by(.[0]) | map({
         parameter_group_name: .[0][0],
         total_clusters: length,
         total_instances: map(.[2] | length) | add // 0
      }) | .[]'
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
   "parameter_group_name": "sample-parameter-group",
   "total_clusters": 50,
   "total_instances": 150
}
```

建立 Amazon DocumentDB 叢集參數群組

當您使用新引擎版本和新引擎版本建立叢集時default.docdb3.6，會建立預設叢集參數群組，例如default.docdb5.0default.docdb4.0、 或 AWS 區域。在相同 區域中建立且具有相同引擎版本的後續叢集會繼承default叢集參數群組。建立後，即無法刪除或重新命名default參數群組。您可以建立具有偏好參數值的自訂參數群組，並將其連接至 Amazon DocumentDB 叢集，藉此修改叢集執行個體的引擎行為。

下列程序會引導您建立自訂叢集參數群組。您接著可在該參數群組中修改參數。

> **Note**
>
> 建立叢集參數群組後，您應等待至少 5 分鐘，然後再使用該特定參數群組。這可讓 Amazon DocumentDB 在叢集參數群組用於新叢集之前完全完成create動作。您可以使用 AWS 管理主控台 或 describe-db-cluster-parameter-groups AWS CLI 操作來驗證叢集參數群組是否已建立。如需詳細資訊，請參閱描述 Amazon DocumentDB 叢集參數群組。

Using the AWS 管理主控台

建立叢集參數群組

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在 Cluster parameter groups (叢集參數群組) 窗格中，選擇 Create (建立)。
4. 在 Create cluster parameter group (建立叢集參數群組) 窗格中，輸入以下項目：

   a. 新的叢集參數群組名稱 — 輸入叢集參數群組的名稱。例如 `sample-parameter-group`。叢集參數群組具有以下命名限制：

      - 長度為 [1-255] 個英數字元。
      - 第一個字元必須是字母。
      - 不能以連字號結尾，或包含兩個連續連字號。

   b. 系列 — 選擇您要用於叢集的 DocumentDB 版本。

   c. 描述 — 提供此叢集參數群組的描述。

5. 若要建立叢集參數群組，請選擇 Create (建立)。若要取消操作，請選擇 Cancel (取消)。
6. 選擇 Create (建立) 之後，頁面頂端會顯示下列文字，以確認您的叢集參數群組是否已成功建立：

```
Successfully created cluster parameter group 'sample-parameter-group'.
```

Using the AWS CLI

若要為 Amazon DocumentDB 4.0 叢集建立新的叢集參數群組，請使用 AWS CLI `create-db-cluster-parameter-group`操作搭配下列參數：

- **`--db-cluster-parameter-group-name`** — 自訂叢集參數群組的名稱。例如 `sample-parameter-group`。

- **--db-cluster-parameter-group-family** — 做為自訂叢集參數群組範本的叢集參數群組系列。
- **--description** — 此叢集參數群組的使用者提供描述。以下範例使用 "Custom docdb4.0 parameter group"。

針對 Linux、macOS 或 Unix：

Example

```
aws docdb create-db-cluster-parameter-group \
 --db-cluster-parameter-group-name sample-parameter-group \
 --db-parameter-group-family docdb5.0 \
 --description "Custom docdb5.0 parameter group"
```

針對 Windows：

```
aws docdb create-db-cluster-parameter-group ^
 --db-cluster-parameter-group-name sample-parameter-group ^
 --db-parameter-group-family docdb5.0 ^
 --description "Custom docdb5.0 parameter group"
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "sample-parameter-group",
        "DBParameterGroupFamily": "docdb5.0",
        "Description": "Custom docdb4.0 parameter group",
        "DBClusterParameterGroupArn": "sample-parameter-group-arn"
    }
}
```

修改 Amazon DocumentDB 叢集參數群組

本節說明如何修改自訂 Amazon DocumentDB 參數群組。在 Amazon DocumentDB 中，您無法修改在新區域中首次使用新引擎版本建立default叢集時建立的叢集參數群組。如果您的 Amazon DocumentDB 叢集使用預設叢集參數群組，而且您想要修改其中的值，您必須先建立新的參數群組或複製現有的參數群組、修改它，然後將修改後的參數群組套用至您的叢集。

完成下列步驟以修改自訂叢集參數群組。修改動作可能需要一段時間才能傳播。等待修改後的叢集參數群組可用，再將其連接至您的叢集。您可以使用 AWS 管理主控台 或 AWS CLI `describe-db-cluster-parameters`操作來驗證叢集參數群組是否已修改。如需詳細資訊，請參閱描述叢集參數群組。

Using the AWS 管理主控台

請依照下列步驟修改自訂 Amazon DocumentDB 參數群組。您無法修改 `default` 參數群組。如果您想要修改 `default` 參數群組中的值，可以複製預設叢集參數群組並加以修改，接著將修改的參數群組套用到叢集。如需將參數群組套用到叢集的詳細資訊，請參閱修改 Amazon DocumentDB 叢集。

修改自訂叢集參數群組

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Parameter groups (參數群組)。在參數群組清單中，選擇您要修改看的參數群組名稱。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 對於您要修改之參數群組中的每個參數，請執行下列動作：
   a. 找到您要修改的參數，並檢查其是否列為可修改資料欄`true`下的 ，以確認其可修改。
   b. 如果可以修改，請選取參數並從主控台頁面的右上角選擇 Edit (編輯)。
   c. 在修改 **`<parameter-name>`** 對話方塊中，進行所需的變更。然後，選擇 Modify cluster parameter (修改叢集參數)，或選擇 Cancel (取消) 以捨棄變更。

Using the AWS CLI

您可以使用 修改自訂 Amazon DocumentDB `ParameterValue`叢集參數群組中任何可修改參數`ApplyMethod`的 `Description`、 或 AWS CLI。您無法直接修改預設叢集參數群組。

若要修改自訂叢集參數群組的參數，請使用 `modify-db-cluster-parameter-group` 操作搭配下列參數。

- **--db-cluster-parameter-group-name** – 必要。您想要修改的叢集參數群組名稱。
- **--parameters** – 必要。您要修改的參數。如需套用至 Amazon DocumentDB 叢集中所有執行個體的參數清單，請參閱 Amazon DocumentDB 叢集參數參考。每個參數項目必須包括下列：
  - **ParameterName** — 您要修改的參數名稱。
  - **ParameterValue** — 此參數的新值。
  - **ApplyMethod** — 您希望如何套用此參數的變更。允許值為 `immediate` 和 `pending-reboot`。

> **Note**
>
> `static` 必須擁有使用 `ApplyType` 的參數 `ApplyMethod` `pending-reboot`。

Example- 修改參數值

在此範例中，您列出 `sample-parameter-group` 的參數值和修改 `tls` 參數。接著，在等待 5 分鐘後，您會再次列出 `sample-parameter-group` 的參數值，以查看變更的參數值。

1. 列出參數和其 `sample-parameter-group` 的值。

   針對 Linux、macOS 或 Unix：

   ```
   aws docdb describe-db-cluster-parameters \
         --db-cluster-parameter-group-name sample-parameter-group
   ```

   針對 Windows：

   ```
   aws docdb describe-db-cluster-parameters ^
         --db-cluster-parameter-group-name sample-parameter-group
   ```

   此操作的輸出將會如下所示 (JSON 格式)。

   ```
   {
         "Parameters": [
             {
                 "Source": "system",
                 "ApplyType": "static",
                 "AllowedValues": "disabled,enabled",
                 "ParameterValue": "enabled",
   ```

```
                "ApplyMethod": "pending-reboot",
                "DataType": "string",
                "ParameterName": "tls",
                "IsModifiable": true,
                "Description": "Config to enable/disable TLS"
            },
            {
                "Source": "user",
                "ApplyType": "dynamic",
                "AllowedValues": "disabled,enabled",
                "ParameterValue": "enabled",
                "ApplyMethod": "pending-reboot",
                "DataType": "string",
                "ParameterName": "ttl_monitor",
                "IsModifiable": true,
                "Description": "Enables TTL Monitoring"
            }
        ]
}
```

2. 修改 `tls` 參數，讓其值為 `disabled`。

   您無法修改 `ApplyMethod`，因為 `ApplyType` 是 `static`。

   針對 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group \
       --parameters
 "ParameterName"=tls,"ParameterValue"=disabled,"ApplyMethod"=pending-reboot
```

   針對 Windows：

```
aws docdb modify-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group ^
       --parameters
 "ParameterName"=tls,"ParameterValue"=disabled,"ApplyMethod"=pending-reboot
```

   此操作的輸出將會如下所示 (JSON 格式)。

```
{
       "DBClusterParameterGroupName": "sample-parameter-group"
```

```
    }
```

3. 等待至少 5 分鐘。
4. 列出 `sample-parameter-group` 的參數值，來確認 `tls` 參數已經過修改。

   針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameters \
       --db-cluster-parameter-group-name sample-parameter-group
```

   針對 Windows：

```
aws docdb describe-db-cluster-parameters ^
       --db-cluster-parameter-group-name sample-parameter-group
```

   此操作的輸出將會如下所示 (JSON 格式)。

```
{
       "Parameters": [
           {
               "ParameterValue": "false",
               "ParameterName": "enable_audit_logs",
               "ApplyType": "dynamic",
               "DataType": "string",
               "Description": "Enables auditing on cluster.",
               "AllowedValues": "true,false",
               "Source": "system",
               "IsModifiable": true,
               "ApplyMethod": "pending-reboot"
           },
           {
               "ParameterValue": "disabled",
               "ParameterName": "tls",
               "ApplyType": "static",
               "DataType": "string",
               "Description": "Config to enable/disable TLS",
               "AllowedValues": "disabled,enabled",
               "Source": "system",
               "IsModifiable": true,
               "ApplyMethod": "pending-reboot"
           }
       ]
```

```
}
```

修改 Amazon DocumentDB 叢集以使用自訂叢集參數群組

當您建立 Amazon DocumentDB 叢集時，會自動為該叢集建立`default.docdb4.0`參數群組。您無法修改 `default` 叢集參數群組。反之，您可以修改 Amazon DocumentDB 叢集，將新的自訂參數群組與其建立關聯。

本節說明如何使用 AWS 管理主控台 和 AWS Command Line Interface () 修改現有的 Amazon DocumentDB 叢集以使用自訂叢集參數群組AWS CLI。

Using the AWS 管理主控台

修改 Amazon DocumentDB 叢集以使用新的非預設叢集參數群組

1. 開始之前，請確定您已建立 Amazon DocumentDB 叢集和叢集參數群組。如需進一步說明，請參閱建立 Amazon DocumentDB 叢集與建立 Amazon DocumentDB 叢集參數群組。
2. 建立叢集參數群組之後，請在 開啟 Amazon DocumentDB 主控台https://console.aws.amazon.com/docdb。在導覽窗格中選擇 Clusters (叢集) 來將新的參數群組新增至叢集。
3. 選擇您要與參數群組建立關聯的叢集。選擇 Actions (動作)，然後選擇 Modify (修改) 來修改叢集。
4. 在 Cluster options (叢集選項) 下，選擇要與叢集建立關聯的參數群組。
5. 選擇 Continue (繼續) 以檢視修改摘要。
6. 確認變更後，即可立即套用，或是在下一個維護時段在 Scheduling of modifications (修改的排程) 下套用。
7. 選擇 Modify cluster (修改叢集) 來以新的參數群組更新叢集。

Using the AWS CLI

開始之前，請確定您已建立 Amazon DocumentDB 叢集和叢集參數群組。您可以使用 操作建立 Amazon DocumentDB 叢集。 AWS CLI `create-db-cluster`您可以使用 `create-db-cluster-parameter-group`操作建立叢集參數群組 AWS CLI 。

若要將新的叢集參數群組新增至叢集，請使用 AWS CLI `modify-db-cluster`操作搭配下列參數。

- --db-cluster-identifier — 叢集的名稱（例如 `sample-cluster`）。
- --db-cluster-parameter-group-name — 您要與叢集建立關聯的參數群組名稱（例如 `sample-parameter-group`）。

Example

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --db-cluster-parameter-group-name sample-parameter-group
```

此操作的輸出將會如下所示 (JSON 格式)。

```
"DBCluster": {
    "AvailabilityZones": [
            "us-west-2c",
            "us-west-2b",
            "us-west-2a"
    ],
    "BackupRetentionPeriod": 1,
    "DBClusterIdentifier": "sample-cluster",
    "DBClusterParameterGroup": "sample-parameter-group",
    "DBSubnetGroup": "default",
   ...
}
```

複製 Amazon DocumentDB 叢集參數群組

您可以使用 AWS 管理主控台 或 AWS Command Line Interface () 在 Amazon DocumentDB 中複製叢集參數群組AWS CLI。

Using the AWS 管理主控台

下列程序會藉由製作現有叢集參數群組的複本，引導您建立新的叢集參數群組。

複製叢集參數群組

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。

3. 在 Cluster parameter groups (叢集參數群組) 窗格中，選擇您想要複製的叢集參數群組名稱。
4. 依序選擇 Actions (動作) 和 Copy (複製) 來複製該參數群組。
5. 在 Copy options (複製選項) 下，輸入新叢集參數群組的名稱和說明。接著選擇 Copy (複製) 來儲存變更。

Using the AWS CLI

若要複製叢集參數群組，請使用 `copy-db-cluster-parameter-group` 操作搭配下列參數。

- **`--source-db-cluster-parameter-group-identifier`** – 必要。您要複製的叢集參數群組名稱或 Amazon Resource Name (ARN)。

  如果來源和目標叢集參數群組位於相同 中 AWS 區域，識別符可以是名稱或 ARN。

  如果來源和目標叢集參數群組位於不同的 中 AWS 區域，識別符必須是 ARN。
- **`--target-db-cluster-parameter-group-identifier`** – 必要。叢集參數群組複本的名稱或 ARN。

  限制條件：
  - 不得為無效、空白或留空。
  - 必須包含 1-255 個字母、數字或連字號。
  - 第一個字元必須是字母。
  - 不能以連字號結尾，或包含兩個連續連字號。
- **`--target-db-cluster-parameter-group-description`** – 必要。使用者提供叢集參數群組複本的說明。

Example

以下程式碼複製 `sample-parameter-group`，將複本命名為 `sample-parameter-group-copy`。

針對 Linux、macOS 或 Unix：

```
aws docdb copy-db-cluster-parameter-group \
    --source-db-cluster-parameter-group-identifier sample-parameter-group \
    --target-db-cluster-parameter-group-identifier sample-parameter-group-copy \
    --target-db-cluster-parameter-group-description "Copy of sample-parameter-group"
```

針對 Windows：

```
aws docdb copy-db-cluster-parameter-group ^
    --source-db-cluster-parameter-group-identifier sample-parameter-group ^
    --target-db-cluster-parameter-group-identifier sample-parameter-group-copy ^
    --target-db-cluster-parameter-group-description "Copy of sample-parameter-group"
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:sample-parameter-group-copy",
        "DBClusterParameterGroupName": "sample-parameter-group-copy",
        "DBParameterGroupFamily": "docdb4.0",
        "Description": "Copy of sample-parameter-group"
    }
}
```

重設 Amazon DocumentDB 叢集參數群組

您可以使用 或 AWS 管理主控台 AWS Command Line Interface (AWS CLI) 重設叢集參數群組，將部分或全部 Amazon DocumentDB 叢集參數群組的參數值重設為其預設值。

Using the AWS 管理主控台

請遵循這些步驟操作，將部分或全部叢集參數群組的參數值重設為預設值。

重設叢集參數群組的參數值

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Parameter groups (參數群組)。
3. 在 Cluster parameter groups (叢集參數群組) 窗格中，選擇您想要重設的叢集參數群組名稱。
4. 依序選擇 Actions (動作) 和 Reset (重設) 來重設該參數群組。
5. 在產生的 Cluster parameter group reset confirmation (叢集參數群組重設確認) 頁面上，確認您想要將該參數群組的所有叢集參數重設為預設值。然後選擇 Reset (重設) 以重設您的參數群組。您也可以選擇 Cancel (取消) 以捨棄變更。

Using the AWS CLI

若要將部分或全部叢集參數群組的參數值重設為預設值，使用 reset-db-cluster-parameter-group 操作搭配下列參數。

- **--db-cluster-parameter-group-name** – 必要。要重設之叢集參數群組的名稱。
- **--parameters** — 選用。叢集參數群組中要重設為預設值的 ApplyMethod 和 ParameterName 清單。靜態參數必須設定為 pending-reboot，才會在下一個執行個體重新啟動或 reboot-db-instance 請求時生效。您必須呼叫您希望將更新的靜態參數套用到其中的叢集中 reboot-db-instance 的每個執行個體。

  此參數和 --reset-all-parameters 互斥：您可以使用其中一個，但無法同時使用兩者。
- **--reset-all-parameters** 或 **--no-reset-all-parameters** — 選用。指定是否要將所有參數 (--reset-all-parameters) 或僅一些參數 (--no-reset-all-parameters) 重設為預設值。--reset-all-parameters 參數和 --parameters 互斥：您可以使用其中一個，但無法同時使用兩者。

  重設整個群組時，動態參數會立即更新。靜態參數必須設定為 pending-reboot，才會在下一個執行個體重新啟動或 reboot-db-instance 請求時生效。您必須呼叫您希望將更新的靜態參數套用到其中的叢集中 reboot-db-instance 的每個執行個體。

Example

範例 1：將所有參數重設為預設值

以下程式碼會將在叢集參數群組 sample-parameter-group 中的所有參數重設為預設值。

針對 Linux、macOS 或 Unix：

```
aws docdb reset-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group \
       --reset-all-parameters
```

針對 Windows：

```
aws docdb reset-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group ^
       --reset-all-parameters
```

範例 2：將指定參數重設為預設值

以下程式碼會將在叢集參數群組 `sample-parameter-group` 中的 `tls` 參數重設為預設值。

針對 Linux、macOS 或 Unix：

```
aws docdb reset-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group \
       --no-reset-all-parameters \
       --parameters ParameterName=tls,ApplyMethod=pending-reboot
```

針對 Windows：

```
aws docdb reset-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group ^
       --no-reset-all-parameters ^
       --parameters ParameterName=tls,ApplyMethod=pending-reboot
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
       "DBClusterParameterGroupName": "sample-parameter-group"
   }
```

重新啟動叢集執行個體

在靜態參數的值變更前，必須將叢集執行個體重新開機。在您希望將更新靜態參數套用到其中的叢集中重新啟動每個執行個體。

針對 Linux、macOS 或 Unix：

```
aws docdb reboot-db-instance \
       --db-instance-identifier sample-cluster-instance
```

針對 Windows：

```
aws docdb reboot-db-instance ^
       --db-instance-identifier sample-cluster-instance
```

刪除 Amazon DocumentDB 叢集參數群組

您可以使用 AWS 管理主控台 或 AWS Command Line Interface () 刪除自訂 Amazon DocumentDB 叢集參數群組AWS CLI。您無法刪除 `default.docdb4.0` 叢集參數群組。

Using the AWS 管理主控台

刪除叢集參數群組

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。

> **Tip**
> 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在 Parameter groups (參數群組) 窗格中，選擇您想要刪除的叢集參數群組左側旁的選項按鈕。
4. 選擇動作，然後選擇刪除。
5. 在 Delete (刪除) 確認窗格中，選擇 Delete (刪除) 來刪除叢集參數群組。如果要保留叢集參數群組，請選擇 Cancel (取消)。

Using the AWS CLI

若要刪除叢集參數群組，請使用 `delete-db-cluster-parameter-group` 操作搭配下列參數。

- **`--db-cluster-parameter-group-name`** – 必要。要刪除之叢集參數群組的名稱。這必須是現有叢集的參數群組。您無法刪除*`default.docdb4.0`*叢集參數群組。

Example- 刪除叢集參數群組

以下範例逐步引導您刪除叢集參數群組的三個步驟：

1. 尋找您想要刪除的叢集參數群組名稱。
2. 刪除指定的叢集參數群組。
3. 確認已刪除叢集的參數群組。

1. 尋找您想要刪除的叢集參數群組名稱。

以下程式碼列出所有叢集參數群組的名稱。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameter-groups \
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

針對 Windows：

```
aws docdb describe-db-cluster-parameter-groups ^
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

上述操作的輸出是與以下 (JSON 格式) 類似的叢集參數群組名稱清單。

```
[
       [
           "default.docdb4.0"
       ],
       [
           "sample-parameter-group"
       ],
       [
           "sample-parameter-group-copy"
       ]
   ]
```

2. 刪除特定叢集參數群組。

以下程式碼會刪除叢集參數群組 `sample-parameter-group-copy`。

針對 Linux、macOS 或 Unix：

```
aws docdb delete-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group-copy
```

針對 Windows：

```
aws docdb delete-db-cluster-parameter-group ^
```

```
        --db-cluster-parameter-group-name sample-parameter-group-copy
```

此操作將不會有輸出。

3. 確認已刪除指定的叢集參數群組。

以下程式碼列出所有剩餘叢集參數群組的名稱。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameter-groups \
        --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

針對 Windows：

```
aws docdb describe-db-cluster-parameter-groups ^
        --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

上述操作的輸出是與以下 (JSON 格式) 類似的叢集參數群組清單。您剛刪除的叢集參數群組不應在清單中。

此操作的輸出將會如下所示 (JSON 格式)。

```
[
        [
            "default.docdb4.0"
        ],
        [
            "sample-parameter-group"
        ]
   ]
```

Amazon DocumentDB 叢集參數參考

當您變更動態參數並儲存叢集參數群組時，無論 Apply Immediately (立即套用) 設定為何，變更都會立即套用。當您變更靜態參數並儲存叢集參數群組時，參數變更會在您手動重新啟動叢集執行個體之後生效。您可以使用 Amazon DocumentDB 主控台或明確呼叫 來重新啟動執行個體`reboot-db-instance`。

下表顯示套用至 Amazon DocumentDB 叢集中所有執行個體的參數。

Amazon DocumentDB 叢集層級參數

參數	預設值	有效值	可修改	套用類型	資料類型	說明
audit_logs	disabled	enabled、disable、ddl、dml_read、dml_write、全部、無	是	動態	String	定義是否啟用 Amazon CloudWatch 稽核日誌。 • **enabled** — 已啟用 CloudWatch 稽核日誌。 • **disabled** — CloudWatch 稽核日誌已停用。 • **ddl** — 啟用 DDL 事件的稽核。 • **dml_read** — 啟用 DML 讀取事件的稽核。 • **dml_write** — 啟用 DML 寫入事件的稽核。

參數	預設值	有效值	可修改	套用類型	資料類型	說明
						• **all** — 啟用所有資料庫事件的稽核。 • **none** — 稽核已停用。
change_stream_log_retention_duration	10800	3600-604800	是	動態	Integer	定義變更串流日誌保留並可使用的持續時間 (以秒為單位)。

參數	預設值	有效值	可修改	套用類型	資料類型	說明
default_collection_compression	disabled	啟用、停用 (Amazon DocumentDB 5.0)/zstd、lz4、無 (Amazon DocumentDB8.0)	是	動態	String	定義叢集中新集合的預設壓縮設定 • **enabled** — Lz4 壓縮預設為在 Amazon DocumentDB 5.0 中啟用。zstd 壓縮預設為在 Amazon DocumentDB 8.0 中啟用。 • **disabled, none** — 預設會停用壓縮（在 Amazon DocumentDB 5.0 中停用，在 Amazon DocumentDB 8.0 中不停用）。

參數	預設值	有效值	可修改	套用類型	資料類型	說明
profiler	停用	啟用，停用	是	動態	String	啟用緩慢操作的分析功能。 • **enabled** — 需要超過客戶定義閾值（例如 100 毫秒）的操作會記錄toAmazon CloudWatch Logs。 • **disabled** — 慢速操作不會記錄到 CloudWatch Logs。
profiler_sampling_rate	1.0	0.0-1.0	是	動態	Float	定義已記錄操作的取樣率。

參數	預設值	有效值	可修改	套用類型	資料類型	說明
profiler_threshold_ms	100	50-2147483646	是	動態	Integer	定義 profiler 的閾值。 • 所有大於 的操作profiler_threshold_ms 都會記錄到 CloudWatch Logs。

參數	預設值	有效值	可修改	套用類型	資料類型	說明
planner_version	3.0	1.0、2.0、3.0	是	動態	Float	定義要用於查詢的查詢規劃器版本。 • **1.0** — Amazon DocumentDB 5.0 中支援。 • **2.0** — 尋找和更新運算子的改進。Amazon DocumentDB 5.0 中支援。 • **3.0** — 尋找、更新和彙總運算子的改進。Amazon DocumentDB 8.0 中的預設查詢規劃器。

參數	預設值	有效值	可修改	套用類型	資料類型	說明
tls	已啟用	enabled、disable、fips-140-3、tls1.2+、tls1.3+	是	靜態	String	定義 Transport Layer Security (TLS) 連線是否必要。 • **enabled** — 需要 TLS 連線才能連線。對於 Amazon DocumentDB 5.0 和更早版本，這會啟用 TLS 1.0 到 1.3，但使用 TLS 1.2 或更高版本。從次要版本 5.0.1、Amazon DocumentDB 8.0 及更新版本開始，Amazon DocumentDB 5.0 僅支援 TLS

參數	預設值	有效值	可修改	套用類型	資料類型	說明
						1.2 及更新版本。 • **disabled** — TLS 連線無法用於連線。 • **fips-140-3** — 需要具有聯邦資訊處理標準 (FIPS) 屬性的 TLS 連線才能連線。叢集僅接受每個 FIPS 出版物 140-3 的安全連線。僅支援從這些區域中的 Amazon DocumentDB 5.0（引擎版本 3.0.3727) 叢集開始：ca-central-1、

參數	預設值	有效值	可修改	套用類型	資料類型	說明
						us-west-2、us-east-1、us-east-2、us-gov-east-1、us-gov-west-1。 • **tls1.2+** — 必須使用 TLS 1.2 版及更高版本進行 TLS 連線才能連線。僅支援從 Amazon DocumentDB 4.0（引擎版本 2.0.10980）和 Amazon DocumentDB 5.0（引擎版本 3.0.11051）開始。

參數	預設值	有效值	可修改	套用類型	資料類型	說明
						• **tls1.3+** — 必須使用 TLS 1.3 版及更高版本進行 TLS 連線才能連線。僅支援從 Amazon DocumentDB 4.0（引擎版本 2.0.10980）和 Amazon DocumentDB 5.0（引擎版本 3.0.11051）開始。

參數	預設值	有效值	可修改	套用類型	資料類型	說明
ttl_monitor	已啟用	啟用，停用	是	動態	String	定義是否為叢集啟用存活期 (TTL) 監控功能。 • **enabled** — 已啟用 TTL 監控。 • **disabled** — TTL 監控已停用。

修改 Amazon DocumentDB 叢集參數

在 Amazon DocumentDB 中，叢集參數群組由參數組成，這些參數適用於您在叢集中建立的所有執行個體。如為自訂叢集參數群組，您隨時可以為您建立的參數群組修改其參數值，或將所有參數值重設為預設值。本節說明如何檢視組成 Amazon DocumentDB 叢集參數群組的參數及其值，以及如何變更或更新這些值。

參數可以是 dynamic (動態) 或 static (靜態)。當您變更動態參數並儲存叢集參數群組時，無論 `Apply Immediately` 設定為何，變更都會立即套用。當您變更靜態參數並儲存叢集參數群組時，參數變更只會在您手動重新啟動叢集執行個體之後生效。

檢視 Amazon DocumentDB 叢集參數群組的參數

您可以使用 AWS 管理主控台 或 來查看 Amazon DocumentDB 叢集的參數及其值 AWS CLI。

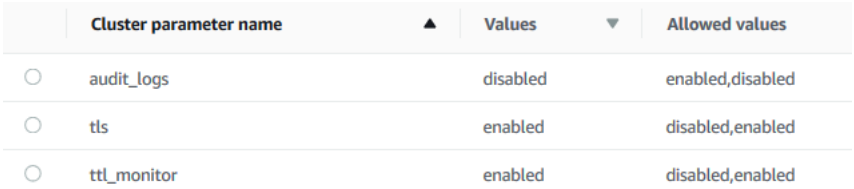
Using the AWS 管理主控台

若要檢視叢集參數群組的詳細資訊

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。

> **Tip**
>
> 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在 Parameter groups (參數群組) 窗格中，選擇您想要查看詳細資訊的叢集參數群組名稱。
4. 結果頁面會顯示各個參數的下列值：參數名稱、目前值、允許值、參數是否可修改、來源、資料類型和說明。



Cluster parameter name	Values	Allowed values
audit_logs	disabled	enabled,disabled
tls	enabled	disabled,enabled
ttl_monitor	enabled	disabled,enabled

Using the AWS CLI

若要查看叢集參數群組的參數和其值，請使用 `describe-db-cluster-parameters` 操作搭配下列參數。

- **`--db-cluster-parameter-group-name`** – 必要。您想要詳細參數清單的叢集參數群組名稱。
- **`--source`** — 選用。如果提供，只會傳回特定來源的參數。參數來源可以是 `engine-default`、`system` 或 `user`。

Example

以下程式碼列出 `custom3-6-param-grp` 參數群組的參數及其值。如需參數群組的詳細資訊，請省略 `--query` 行。如需所有參數群組的詳細資訊，請省略 `--db-cluster-parameter-group-name` 行。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameters \
   --db-cluster-parameter-group-name custom3-6-param-grp \
   --query 'Parameters[*].[ParameterName,ParameterValue]'
```

針對 Windows：

```
aws docdb describe-db-cluster-parameters ^
   --db-cluster-parameter-group-name custom3-6-param-grp ^
   --query 'Parameters[*].[ParameterName,ParameterValue]'
```

此操作的輸出將會如下所示 (JSON 格式)。

```
[
    [
        "audit_logs",
        "disabled"
    ],
    [
        "tls",
        "enabled"
    ],
    [
        "ttl_monitor",
        "enabled"
    ]
]
```

修改 Amazon DocumentDB 叢集參數群組的參數

您可以使用 AWS 管理主控台 或 修改參數群組的參數 AWS CLI。

Using the AWS 管理主控台

更新叢集參數群組的參數

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/docdb：// 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在 Parameter groups (參數群組) 窗格中，選擇您想要更新其中參數的叢集參數群組。
4. 結果頁面會顯示這個叢集參數群組的參數，及其對應詳細資訊。選取要更新的參數。

5. 在頁面右上角，選擇 Edit (編輯) 以變更此參數值。如需叢集參數類型的詳細資訊，請參閱Amazon DocumentDB 叢集參數參考。
6. 進行變更，然後選擇 Modify cluster parameter (修改叢集參數) 以儲存變更。若要捨棄變更，請選擇 Cancel (取消)。

Using the AWS CLI

若要修改叢集參數群組的參數，請使用 `modify-db-cluster-parameter-group` 操作搭配下列參數。

- **`--db-cluster-parameter-group-name`** – 必要。您想要修改的叢集參數群組名稱。
- **`--parameters`** – 必要。您正在修改的一或多個參數。每個參數項目必須包括下列：
  - **`ParameterName`** — 您要修改的參數名稱。
  - **`ParameterValue`** — 此參數的新值。
  - **`ApplyMethod`** — 您希望如何套用此參數的變更。允許值為 `immediate` 和 `pending-reboot`。

> **Note**
>
> `static` 必須擁有使用 `ApplyType` 的參數 `ApplyMethod` `pending-reboot`。

變更叢集參數群組的參數值 (AWS CLI)

下列範例會變更 `tls` 參數。

1. 列出參數和其 **`sample-parameter-group`** 的值

   針對 Linux、macOS 或 Unix：

   ```
   aws docdb describe-db-cluster-parameters \
       --db-cluster-parameter-group-name sample-parameter-group
   ```

   針對 Windows：

   ```
   aws docdb describe-db-cluster-parameters ^
       --db-cluster-parameter-group-name sample-parameter-group
   ```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "Parameters": [
        {
            "Source": "system",
            "ApplyType": "static",
            "AllowedValues": "disabled,enabled",
            "ParameterValue": "enabled",
            "ApplyMethod": "pending-reboot",
            "DataType": "string",
            "ParameterName": "tls",
            "IsModifiable": true,
            "Description": "Config to enable/disable TLS"
        },
        {
            "Source": "user",
            "ApplyType": "dynamic",
            "AllowedValues": "disabled,enabled",
            "ParameterValue": "enabled",
            "ApplyMethod": "pending-reboot",
            "DataType": "string",
            "ParameterName": "ttl_monitor",
            "IsModifiable": true,
            "Description": "Enables TTL Monitoring"
        }
    ]
}
```

2. 修改 **tls** 參數，讓其值為 **disabled**。您無法修改 ApplyMethod，因為 ApplyType 是 static。

   針對 Linux、macOS 或 Unix：

   ```
   aws docdb modify-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group \
       --parameters
    "ParameterName"=tls,ParameterValue=disabled,ApplyMethod=pending-reboot"
   ```

   針對 Windows：

```
aws docdb modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name sample-parameter-group ^
    --parameters "ParameterName=tls,ParameterValue=disabled,ApplyMethod=pending-
reboot"
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "DBClusterParameterGroupName": "sample-parameter-group"
}
```

3. 等待至少 5 分鐘。
4. 列出 **sample-parameter-group** 的參數值。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

針對 Windows：

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name sample-parameter-group
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "Parameters": [
        {
            "ParameterName": "audit_logs",
            "ParameterValue": "disabled",
            "Description": "Enables auditing on cluster.",
            "Source": "system",
            "ApplyType": "dynamic",
            "DataType": "string",
            "AllowedValues": "enabled,disabled",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot"
        },
        {
```

```
            "ParameterName": "tls",
            "ParameterValue": "disabled",
            "Description": "Config to enable/disable TLS",
            "Source": "user",
            "ApplyType": "static",
            "DataType": "string",
            "AllowedValues": "disabled,enabled",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot"
        }
    ]
}
```

了解 Amazon DocumentDB 端點

您可以使用 Amazon DocumentDB （與 MongoDB 相容） 端點來連線至叢集或執行個體。Amazon DocumentDB 有三種不同類型的端點，每個端點都有自己的用途。

主題

- 尋找叢集的端點
- 尋找執行個體的端點
- 連線至端點

叢集端點

叢集端點是 Amazon DocumentDB 叢集的端點，可連線至叢集目前的主要執行個體。每個 Amazon DocumentDB 叢集都有一個叢集端點和一個主要執行個體。如果發生容錯移轉，叢集端點會重新對應至新的主要執行個體。

讀取器端點

讀取器端點是 Amazon DocumentDB 叢集的端點，可連線至該叢集的其中一個可用複本。每個 Amazon DocumentDB 叢集都有一個讀取器端點。如果有多個複本，讀取器端點會將每個連線請求導向其中一個 Amazon DocumentDB 複本。

執行個體端點

執行個體端點會連接至特定的執行個體。叢集中的每個執行個體，不論是主要或複本執行個體，都有自己的唯一執行個體端點。最好不要在您的應用程式中使用執行個體端點。這是因為它們可在容錯移轉時變更角色，因此需要在您的應用程式中進行程式碼變更。

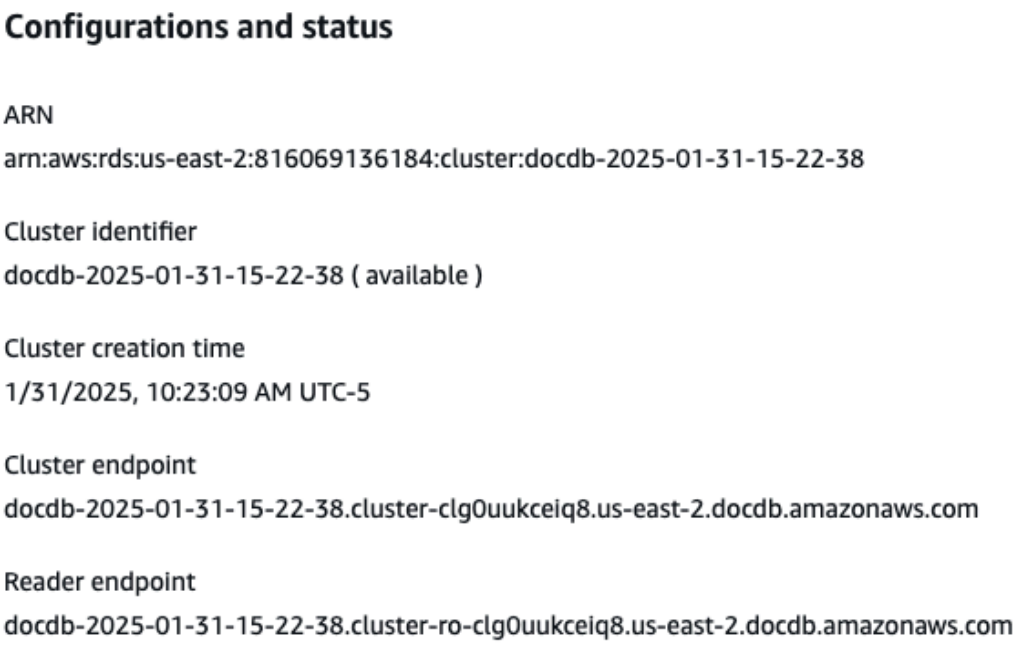
尋找叢集的端點

您可以使用 Amazon DocumentDB 主控台或 尋找叢集的叢集端點和讀取器端點 AWS CLI。

Using the AWS 管理主控台

若要使用主控台尋找叢集的端點：

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。
3. 從叢集清單中選擇您有興趣之叢集的名稱。
4. 在叢集詳細資訊頁面上，選取組態索引標籤。在組態和狀態區段中，您會找到叢集端點和讀取器端點。



5. 若要連線至此叢集，請選取連線與安全索引標籤。尋找 mongo Shell 的連線字串，以及可用於應用程式程式碼的連線字串，以連接至您的叢集。



Using the AWS CLI

若要使用 尋找叢集的叢集和讀取器端點 AWS CLI，請使用這些參數執行 `describe-db-clusters`命令。

Parameters

- **`--db-cluster-identifier`**- 選用。指定要傳回其端點的叢集。如果省略，則最多會傳回您的 100 個叢集的端點。
- **`--query`**- 選用。指定顯示的欄位。有助於減少您尋找端點時需要檢視的資料量。如果省略，將會傳回叢集的所有資訊。
- **`--region`**- 選用。使用 `--region` 參數來指定您要套用命令的區域。若省略，則使用您的預設區域。

Example

以下範例傳回 `sample-cluster` 的 `DBClusterIdentifier`、端點 (叢集端點)，以及 `ReaderEndpoint`。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters \
   --region us-east-1 \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[DBClusterIdentifier,Port,Endpoint,ReaderEndpoint]'
```

針對 Windows：

```
aws docdb describe-db-clusters ^
```

```
    --region us-east-1 ^
    --db-cluster-identifier sample-cluster ^
    --query 'DBClusters[*].[DBClusterIdentifier,Port,Endpoint,ReaderEndpoint]'
```

此操作的輸出將會如下所示 (JSON 格式)。

```
[
  [
     "sample-cluster",
     27017,
     "sample-cluster.cluster-corlsfccjozr.us-east-1.docdb.amazonaws.com",
     "sample-cluster.cluster-ro-corlsfccjozr.us-east-1.docdb.amazonaws.com"
  ]
]
```

現在，有了叢集端點，您可以使用 mongo 或 mongodb 連接至該叢集。如需詳細資訊，請參閱連線至端點。

尋找執行個體的端點

您可以使用 Amazon DocumentDB 主控台或 尋找執行個體的端點 AWS CLI。

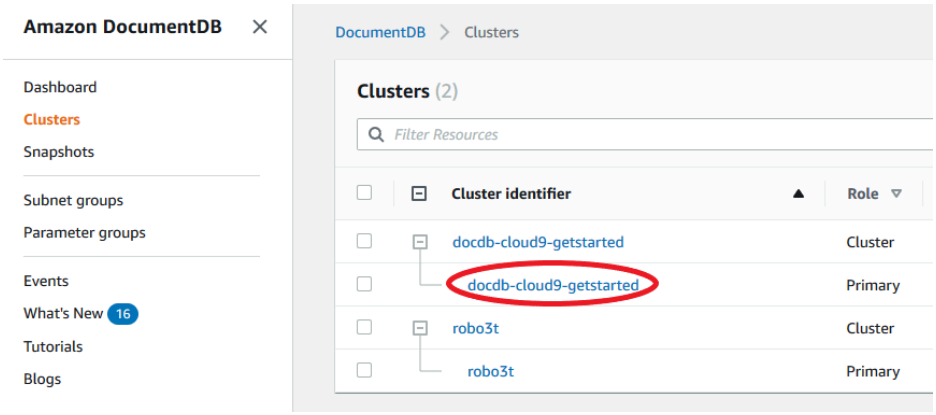
Using the AWS 管理主控台

使用主控台尋找執行個體的端點

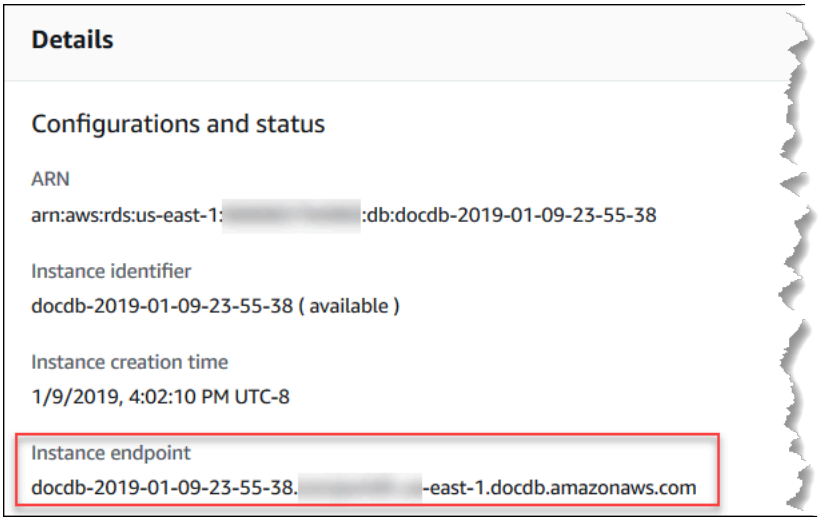
1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在叢集導覽方塊中，您會看到叢集識別符欄。您的執行個體會列在叢集下，類似於下列螢幕擷取畫面。



4. 勾選您感興趣的執行個體左側的方塊。
5. 向下捲動到 Details (詳細資訊) 區段，然後找到執行個體端點。



6. 若要連接至這個執行個體，捲動到 Connect (連接) 區段。找出 mongo 殼層的連線字串及可用在應用程式碼的連線字串，以連接至您的執行個體。



Using the AWS CLI

若要使用 尋找執行個體端點 AWS CLI，請使用這些引數執行下列命令。

引數

- **--db-instance-identifier**- 選用。指定要傳回其端點的執行個體。如果省略，則最多會傳回您的 100 個執行個體的端點。
- **--query**- 選用。指定顯示的欄位。有助於減少您尋找端點時需要檢視的資料量。如果省略，將會傳回執行個體的所有資訊。Endpoint 欄位有三個成員，若如以下範例將其列入查詢中，則會傳回全部三個成員。如果您只對某些 Endpoint 成員有興趣，請將查詢中的 Endpoint 取代為您有興趣的成員，如第二個範例所示。
- **--region**- 選用。使用 --region 參數來指定您要套用命令的區域。若省略，則使用您的預設區域。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-instances \
    --region us-east-1 \
    --db-instance-identifier sample-cluster-instance \
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint]'
```

針對 Windows：

```
aws docdb describe-db-instances ^
    --region us-east-1 ^
    --db-instance-identifier sample-cluster-instance ^
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint]'
```

此操作的輸出將會如下所示 (JSON 格式)。

```
[
    [
        "sample-cluster-instance",
        {
            "Port": 27017,
            "Address": "sample-cluster-instance.corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
            "HostedZoneId": "Z2R2ITUGPM61AM"
        }
    ]
```

```
]
```

您可以藉由指定 Endpoint.Port 和 Endpoint.Address 來修改您的查詢，降低輸出以消除端點的 HostedZoneId。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-instances \
    --region us-east-1 \
    --db-instance-identifier sample-cluster-instance \
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint.Port,Endpoint.Address]'
```

針對 Windows：

```
aws docdb describe-db-instances ^
    --region us-east-1 ^
    --db-instance-identifier sample-cluster-instance ^
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint.Port,Endpoint.Address]'
```

此操作的輸出將會如下所示 (JSON 格式)。

```
[
    [
        "sample-cluster-instance",
        27017,
        "sample-cluster-instance.corcjozrlsfc.us-east-1.docdb.amazonaws.com"
    ]
]
```

現在，有了執行個體端點，您可以使用 mongo 或 mongodb 連接至該執行個體。如需詳細資訊，請參閱連線至端點。

連線至端點

有了端點之後，您可以使用 mongo 殼層或連線字串連接至該叢集或執行個體。

使用 mongo shell 進行連線

使用以下結構來建構您需要的字串，以使用 mongo 殼層連接至叢集或執行個體：

```
mongo \
```

```
    --ssl \
    --host Endpoint:Port \
    --sslCAFile global-bundle.pem \
    --username UserName \
    --password Password
```

mongo 殼層範例

連接至叢集：

```
mongo \
    --ssl \
    --host sample-cluster.corcjozrlsfc.us-east-1.docdb.amazonaws.com:27017 \
    --sslCAFile global-bundle.pem \
    --username UserName \
    --password Password
```

連接至執行個體：

```
mongo \
    --ssl \
    --host sample-cluster-instance.corcjozrlsfc.us-east-1.docdb.amazonaws.com:27017 \
    --sslCAFile global-bundle.pem \
    --username UserName \
    --password Password
```

使用連線字串進行連線

使用以下結構來建構您需要的連線字串，以連接至叢集或執行個體。

```
mongodb://UserName:Password@endpoint:port?replicaSet=rs0&ssl_ca_certs=global-
bundle.pem
```

連線字串範例

連接至叢集：

```
mongodb://UserName:Password@sample-cluster.cluster-corlsfccjozr.us-
east-1.docdb.amazonaws.com:27017?replicaSet=rs0&ssl_ca_certs=global-bundle.pem
```

連接至執行個體：

```
mongodb://UserName:Password@sample-cluster-instance.cluster-corlsfccjozr.us-
east-1.docdb.amazonaws.com:27017?replicaSet=rs0&ssl_ca_certs=global-bundle.pem
```

了解 Amazon DocumentDB Amazon Resource Name (ARNs)

您在 中建立的資源 AWS 都會以 Amazon Resource Name (ARN) 唯一識別。對於某些 Amazon DocumentDB （具有 MongoDB 相容性） 操作，您必須指定其 ARN 以唯一識別 Amazon DocumentDB 資源。例如，當您新增標籤至資源時，您必須提供資源的 ARN。

主題

- 建構 Amazon DocumentDB 資源的 ARN
- 尋找 Amazon DocumentDB 資源 ARN

建構 Amazon DocumentDB 資源的 ARN

您可以使用下列語法建構 Amazon DocumentDB 資源的 ARN。Amazon DocumentDB 共用 Amazon Relational Database Service (Amazon RDS) ARNs。Amazon DocumentDB ARNs包含 `rds`，而非 `docdb`。

```
arn:aws:rds:region:account_number:resource_type:resource_id
```

區域名稱	區域	可用區域 （運算）
美國東部 (俄亥俄)	us-east-2	3
美國東部 (維吉尼亞北部)	us-east-1	6
美國西部 (奧勒岡)	us-west-2	4
非洲 (開普敦)	af-south-1	3
南美洲 (聖保羅)	sa-east-1	3
亞太地區 (香港)	ap-east-1	3

區域名稱	區域	可用區域（運算）
亞太地區 (海德拉巴)	ap-south-2	3
亞太地區 (馬來西亞)	ap-southeast-5	3
亞太地區 (孟買)	ap-south-1	3
亞太地區 (大阪)	ap-northeast-3	3
亞太地區 (首爾)	ap-northeast-2	4
亞太地區 (新加坡)	ap-southeast-1	3
亞太地區 (悉尼)	ap-southeast-2	3
亞太地區 (雅加達)	ap-southeast-3	3
亞太地區 (墨爾本)	ap-southeast-4	3
亞太區域 (泰國)	ap-southeast-7	3
亞太地區 (東京)	ap-northeast-1	3
加拿大 (中部)	ca-central-1	3
加拿大西部 (卡加利)	ca-west-1	3
中國 (北京) 區域	cn-north-1	3
中國 (寧夏)	cn-northwest-1	3
歐洲 (法蘭克福)	eu-central-1	3
歐洲 (蘇黎世)	eu-central-2	3
歐洲 (愛爾蘭)	eu-west-1	3
歐洲 (倫敦)	eu-west-2	3
歐洲 (米蘭)	eu-south-1	3

區域名稱	區域	可用區域（運算）
Europe (Paris)	eu-west-3	3
歐洲 (西班牙)	eu-south-2	3
歐洲 (斯德哥爾摩)	eu-north-1	3
墨西哥 (中部)	mx-central-1	3
中東 (阿拉伯聯合大公國)	me-central-1	3
以色列 (特拉維夫)	il-central-1	3
AWS GovCloud（美國西部）	us-gov-west-1	3
AWS GovCloud（美國東部）	us-gov-east-1	3

> **Note**
>
> Amazon DocumentDB 架構會區隔儲存和運算。對於儲存層，Amazon DocumentDB 會跨三個 AWS 可用區域 (AZs) 複寫資料的六個副本。上表列出的 AZs 是您可以在指定區域中用來佈建運算執行個體AZs 數量。例如，如果您在 ap-northeast-1 中啟動 Amazon DocumentDB 叢集，您的儲存體將跨三個可用AZs複寫六種方式，但您的運算執行個體只能在兩個可用AZs使用。

下表顯示您在建構特定 Amazon DocumentDB 資源的 ARN 時應使用的格式。Amazon DocumentDB 共用 Amazon RDS ARNs的格式。Amazon DocumentDB ARNs包含 `rds`，而非 `docdb`。

資源類型	ARN 格式/範例
執行個體 (db)	`arn:aws:rds:` *`region`*`:`*`account_number`*`:db:`*`resource_id`*

資源類型	ARN 格式/範例
	arn:aws:rds:us-east-1: *1234567890* :db:*sample-db-instance*
叢集 (cluster)	arn:aws:rds: *region*:*account_number* :cluster: *resource_id* arn:aws:rds:us-east-1: *1234567890* :cluster: *sample-db-cluster*
叢集參數群組 (cluster-pg)	arn:aws:rds: *region*:*account_number* :cluster-pg: *resource_id* arn:aws:rds:us-east-1: *1234567890* :cluster-pg: *sample-db-cluster-parameter-group*
安全群組 (secgrp)	arn:aws:rds: *region*:*account_number* :secgrp:*resource_id* arn:aws:rds:us-east-1: *1234567890* :secgrp:*sample-public-secgrp*
叢集快照 (cluster-snapshot)	arn:aws:rds: *region*:*account_number* :cluster-snapshot: *resource_id* arn:aws:rds:us-east-1: *1234567890* :cluster-snapshot: *sample-db-cluster-snapshot*
子網路群組 (subgrp)	arn:aws:rds: *region*:*account_number* :subgrp:*resource_id* arn:aws:rds:us-east-1: *1234567890* :subgrp:*sample-subnet-10*

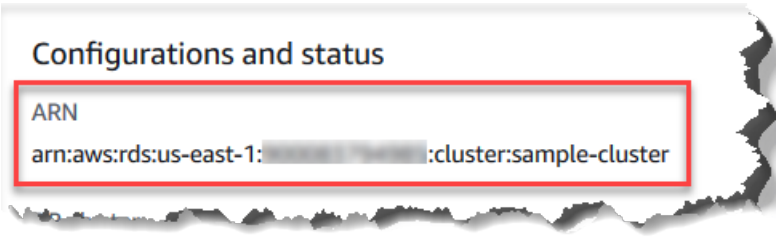
尋找 Amazon DocumentDB 資源 ARN

您可以使用 AWS 管理主控台 或 尋找 Amazon DocumentDB 資源的 ARN AWS CLI。

Using the AWS 管理主控台

若要使用主控台尋找 ARN，請瀏覽至您需要 ARN 的資源，然後檢視該資源的詳細資訊。

例如，您可以選取叢集詳細資訊頁面上的組態索引標籤，以取得叢集的 ARN。您可以在組態和狀態區段中找到 ARN，如下列螢幕擷取畫面所示。



Using the AWS CLI

若要使用特定 Amazon DocumentDB 資源 AWS CLI 的 取得 ARN，請使用該資源的 `describe`操作。下表顯示每個 AWS CLI 操作以及與 操作搭配使用以取得 ARN 的 ARN 屬性。

AWS CLI 命令	ARN 屬性
`describe-db-instances`	`DBInstanceArn`
`describe-db-clusters`	`DBClusterArn`
`describe-db-parameter-groups`	`DBParameterGroupArn`
`describe-db-cluster-parameter-groups`	`DBClusterParameterGroupArn`
`describe-db-security-groups`	`DBSecurityGroupArn`
`describe-db-snapshots`	`DBSnapshotArn`
`describe-db-cluster-snapshots`	`DBClusterSnapshotArn`
`describe-db-subnet-groups`	`DBSubnetGroupArn`

Example- 尋找叢集的 ARN

下列 AWS CLI 操作會尋找叢集 的 ARN`sample-cluster`。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterArn'
```

針對 Windows：

```
aws docdb describe-db-clusters ^
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterArn'
```

此操作的輸出將會如下所示 (JSON 格式)。

```
[
    "arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster"
]
```

Example- 尋找多個參數群組的 ARN

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameter-groups \
   --query 'DBClusterParameterGroups[*].DBClusterParameterGroupArn'
```

針對 Windows：

```
aws docdb describe-db-cluster-parameter-groups ^
   --query 'DBClusterParameterGroups[*].DBClusterParameterGroupArn'
```

此操作的輸出將會如下所示 (JSON 格式)。

```
[
    "arn:aws:rds:us-east-1:123456789012:cluster-pg:custom3-6-param-grp",
    "arn:aws:rds:us-east-1:123456789012:cluster-pg:default.aurora5.6",
    "arn:aws:rds:us-east-1:123456789012:cluster-pg:default.docdb3.6"
]
```

標記 Amazon DocumentDB 資源

您可以使用 Amazon DocumentDB （與 MongoDB 相容） 標籤，將中繼資料新增至 Amazon DocumentDB 資源。這些標籤可與 AWS Identity and Access Management (IAM) 政策搭配使用，以管理對 Amazon DocumentDB 資源的存取，並控制可套用至資源的動作。您也可以使用標籤將類似的標記資源分類為群組以追蹤成本。

您可以標記下列 Amazon DocumentDB 資源：

- 叢集
- 執行個體
- 快照
- 叢集快照
- 參數群組
- 叢集參數群組
- Security groups (安全群組)
- 子網路群組

Amazon DocumentDB 資源標籤概觀

Amazon DocumentDB 標籤是您定義並與 Amazon DocumentDB 資源建立關聯的名稱值對。此名稱叫做金鑰。為金鑰提供值是選用的。您可以使用標籤將任意資訊指派給 Amazon DocumentDB 資源。例如，您可以使用標籤金鑰來定義類別，此標籤值可以是該類別中的項目。例如，您可以定義 的標籤索引鍵`project`和 的標籤值`Salix`，表示 Amazon DocumentDB 資源已指派給 Salix 專案。您也可以使用 或 `environment=test` 等金鑰，使用標籤將 Amazon DocumentDB 資源指定為用於測試或生產`environment=production`。我們建議您使用一組一致的標籤金鑰，以更輕鬆地追蹤與 Amazon DocumentDB 資源相關聯的中繼資料。

您可以使用標籤來整理 AWS 帳單，以反映您自己的成本結構。若要這樣做，請註冊 以取得包含標籤索引鍵值的 AWS 帳戶 帳單。接著，若要查看合併資源的成本，請根據具有相同標籤鍵值的資源來整理您的帳單資訊。例如，您可以使用特定應用程式名稱來標記數個資源，然後整理帳單資訊以查看該應用程式跨數項服務的總成本。如需詳細資訊，請參閱《帳單和成本管理使用者指南》中的使用成本分配標籤。

每個 Amazon DocumentDB 資源都有一個標籤集，其中包含指派給該資源的所有標籤。標籤集最多可以包含 10 個標籤，也可以是空的。如果您將標籤新增至與資源上現有標籤具有相同金鑰的 Amazon DocumentDB 資源，則新值會覆寫舊值。

AWS 不會將任何語意意義套用至您的標籤；標籤會嚴格解譯為字元字串。Amazon DocumentDB 可以在執行個體或其他 Amazon DocumentDB 資源上設定標籤，取決於您在建立資源時使用的設定。例如，Amazon DocumentDB 可能會新增標籤，指出執行個體用於生產或測試。

您可以將標籤新增至快照，但是您的帳單將不會反映這個分組。

您可以使用 AWS 管理主控台 或 AWS CLI 來新增、列出和刪除 Amazon DocumentDB 資源上的標籤。使用 時 AWS CLI，您必須為要使用的資源提供 Amazon Resource Name (ARN)。如需 Amazon DocumentDB ARNs 的詳細資訊，請參閱 [了解 Amazon DocumentDB Amazon Resource Name (ARNs)]。

標籤限制條件

下列限制條件適用於 Amazon DocumentDB 標籤：

- 每個資源的標籤數目上限 - 10
- Key (索引鍵) 長度上限 - 128 個 Unicode 字元
- Value (值) 長度上限 - 256 個 Unicode 字元
- Key (索引鍵) 和 Value (值) 的有效字元 - UTF-8 字元集中的大小寫字母、數字、空格和以下字元：`_ . : / = + -` 和 @ (Java regex：`"^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$"`)
- 標籤鍵與值皆區分大小寫。
- 不要將 `aws:` 字首用於標籤索引鍵或值；它保留供 AWS使用。

在 Amazon DocumentDB 資源上新增和更新標籤

您可以使用 AWS 管理主控台 或 將最多 10 個標籤新增至資源 AWS CLI。

Using the AWS 管理主控台

無論您要將標籤新增至何種資源，將標籤新增至資源的程序都很相似。在此範例中，將標籤新增至叢集。

若要使用主控台將標籤新增至叢集或更新標籤

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 從導覽窗格選擇 clusters (叢集)。

3. 選擇您要新增標籤至其中的叢集名稱。
4. 向下捲動到 Tags (標籤) 區段，然後選擇 Edit (編輯)。
5. 對於您要新增至此資源的每個標籤，執行以下操作：

    a. 若要新增新標籤，請在 Key (索引鍵) 方塊中輸入標籤的名稱。若要變更標籤的值，請在 Key (索引鍵) 欄中尋找標籤的名稱。
    b. 若要給予標籤新值或更新其值，請在 Value (值) 方塊中輸入標籤的值。
    c. 如果您想新增更多標籤，請選擇 Add (新增)。否則，於完成時選擇 Save (儲存)。

Using the AWS CLI

無論您要將標籤新增至何種資源，將標籤新增至資源的程序都很相似。在此範例中，將三個標籤新增至叢集。第二個標籤 key2 沒有值。

將 AWS CLI 操作`add-tags-to-resource`與這些參數搭配使用。

Parameters

- **`--resource-name`**- 您要新增標籤之 Amazon DocumentDB 資源的 ARN。
- **`--tags`**- 列出您要以 格式新增至此資源的標籤 （ 鍵/值對 ）`Key=`*`key-name`*`,Value=`*`tag-value`*。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb add-tags-to-resource \
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster \
    --tags Key=key1,Value=value1 Key=key2 Key=key3,Value=value3
```

針對 Windows：

```
aws docdb add-tags-to-resource ^
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster \
    --tags Key=key1,Value=value1 Key=key2 Key=key3,Value=value3
```

`add-tags-to-resource` 操作不會產生輸出。若要查看操作的結果，請使用 `list-tags-for-resource` 操作。

列出 Amazon DocumentDB 資源上的標籤

您可以使用 AWS 管理主控台 或 AWS CLI 來取得 Amazon DocumentDB 資源的標籤清單。

Using the AWS 管理主控台

無論您要將標籤新增至何種資源，在資源上列出標籤的程序都很相似。在此範例中，列出叢集的標籤。

使用主控台列出叢集標籤

1. 在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 從導覽窗格選擇 clusters (叢集)。
3. 選擇您要列出其標籤的叢集名稱。
4. 若要查看此資源上的標籤清單，請向下捲動至 Tags (標籤) 區段。

Using the AWS CLI

無論您要列出何種資源的標籤，在資源上列出標籤的程序都很相似。在此範例中，列出叢集上的標籤。

將 AWS CLI 操作`list-tags-for-resource`與這些參數搭配使用。

Parameters

- **`--resource-name`** - 必要項目。您要列出標籤之 Amazon DocumentDB 資源的 ARN。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb list-tags-for-resource \
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster
```

針對 Windows：

```
aws docdb list-tags-for-resource ^
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "TagList": [
        {
            "Key": "key1",
            "Value": "value1"
        },
        {
            "Key": "key2",
            "Value": ""
        },
        {
            "Key": "key3",
            "Value": "value3"
        }
    ]
}
```

從 Amazon DocumentDB 資源移除標籤

您可以使用 AWS 管理主控台 或 從 Amazon DocumentDB 資源 AWS CLI 移除標籤。

Using the AWS 管理主控台

無論您要將標籤新增至何種資源，從資源中移除標籤的程序都很相似。在此範例中，從叢集中移除標籤。

使用主控台將標籤從叢集中移除

1. 在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 從導覽窗格選擇 clusters (叢集)。
3. 選擇您要從中移除標籤的叢集名稱。
4. 向下捲動到 Tags (標籤) 區段，然後選擇 Edit (編輯)。
5. 如果您要移除此資源的所有標籤，請選擇 Remove all (全部移除)。否則，針對您要從此資源中移除的每個標籤，執行以下操作：

    a. 在 Key (索引鍵) 欄中尋找標籤的名稱。

    b. 在標籤索引鍵的同一列上選擇 Remove (移除)。

c. 完成時，選擇儲存。

Using the AWS CLI

無論您要從何種資源中移除標籤，從資源中移除標籤的程序都很相似。在此範例中，從叢集中移除標籤。

將 AWS CLI 操作`remove-tags-from-resource`與這些參數搭配使用。

- **`--resource-name`** - 必要項目。您要從中移除標籤的 Amazon DocumentDB 資源 ARN。
- **`--tag-keys`** - 必要項目。您要從資源中移除的標籤索引鍵清單。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb remove-tags-from-resource \
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster \
    --tag-keys key1 key3
```

針對 Windows：

```
aws docdb remove-tags-from-resource ^
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster \
    --tag-keys key1 key3
```

`removed-tags-from-resource` 操作不會產生輸出。若要查看操作的結果，請使用 `list-tags-for-resource` 操作。

維護 Amazon DocumentDB

Amazon DocumentDB 會定期執行兩種維護：

- 叢集維護會更新資料庫引擎。引擎更新包含安全性修正、錯誤修正、新功能和其他引擎增強功能。
- 執行個體維護會更新執行個體上的作業系統 (OS)。

引擎修補程式和作業系統更新使用相同的三個生命週期類別 - 選用、必要和強制 - 具有相同的通知，並為每個類別套用行為。引擎版本也有第四個類別：次要版本，您可以手動升級至 。類別包括：

- 選用 - 包含非關鍵改進。沒有自動套用日期，也沒有 AHD 通知；在適合您時套用。（對於作業系統更新，您可以訂閱 RDS-EVENT-0230，以便在有可用時收到通知。）
- 必要 - 包含安全性和其他重要修正。您透過 Health 儀板表 (AHD) 和電子郵件收到通知。必要的動作會在叢集或執行個體維護時段後自動套用AutoAppliedAfterDate。您可以在該日期之前變更維護時段來延遲。
- 強制 - 罕見、高度關鍵的修正。在維護時段之後自動套用到維護時段之外ForcedApplyDate。Amazon DocumentDB 只會指定在沒有其他選項可用時強制執行的動作。
- 次要版本（僅限引擎版本）— 主要版本上的編號引擎版本（例如 5.0.1)。使用者驅動：您可以透過修改叢集的引擎版本進行升級。永不自動套用；無 AHD 通知。次要版本不會針對 5.0 之前的主要版本發佈。

引擎修補程式會發佈至單一類別（選用、必要或強制），並保留在其中。作業系統更新進度：大多數會以選用的開始，如果未套用，則轉換為必要和最終強制。確切的計時取決於修補程式，並發佈在 AHD 通知中，以及 傳回的日期欄位 describe-pending-maintenance-actions（請參閱 套用日期)。Amazon DocumentDB 版本備註會在宣布引擎變更時使用這些類別名稱。

套用任何引擎修補程式會使叢集短暫離線。本主題的其餘部分會逐步解說維護時段的運作方式、如何尋找待定工作、如何套用引擎修補程式和次要版本、作業系統更新的運作方式，以及全域叢集的特殊處理方式。

主題

- Amazon DocumentDB 的維護動作
- 引擎版本編號
- 管理您的 Amazon DocumentDB 維護時段
- Amazon DocumentDB 引擎修補程式的通知
- 檢視待定的 Amazon DocumentDB 維護動作
- Amazon DocumentDB 引擎更新
- 次要版本升級
- Amazon DocumentDB 作業系統更新
- 使用者啟動的更新
- 全域叢集修補

Amazon DocumentDB 的維護動作

下列維護動作適用於 Amazon DocumentDB 叢集：

- `system-update` — 升級 Amazon DocumentDB 叢集的引擎修補程式。如需詳細資訊，請參閱Amazon DocumentDB 引擎更新。
- `os-upgrade` — 使用滾動升級更新 Amazon DocumentDB 叢集中所有資料庫執行個體的作業系統。如需詳細資訊，請參閱Amazon DocumentDB 作業系統更新。

下列維護動作適用於 Amazon DocumentDB 執行個體：

- `system-update` — 升級 Amazon DocumentDB 執行個體的作業系統。我們建議您改用叢集層級`os-upgrade`維護動作。如需詳細資訊，請參閱Amazon DocumentDB 作業系統更新。

引擎版本編號

Amazon DocumentDB 使用兩個不同的版本識別符：

- 引擎版本 - 格式的三部分編號 *`major`*`.`*`major`*`.`*`minor`*（例如，`5.0.0`或 `5.0.1`)。前兩個部分 (`5.0`) 是 MongoDB 相容性版本；第三個部分是次要版本，當 Amazon DocumentDB 發佈包含錯誤修正和非中斷改進的次要版本時遞增。這是您在建立或升級叢集時指定的版本。
- 引擎修補程式版本 - 格式的個別三部分編號 *`major`*`.0.`*`patch`*（例如 `3.0.17983`)，用於識別套用至叢集的修補程式層級。中間位數一律為 `0`。修補程式版本包含重要的安全性和穩定性修正。

您可以從引擎修補程式版本的字首判斷引擎版本，如下表所示。

引擎修補程式版本字首	Amazon DocumentDB 引擎版本
`1.0.`*`x`*	3.6
`2.0.`*`x`*	4.0
`3.0.`*`x`*	5.0
`4.0.`*`x`*	8.0

若要檢查叢集正在執行的修補程式版本，請連接並執行 `db.runCommand({getEngineVersion: 1})`。

如需發行的引擎修補程式版本清單以及每個版本包含的內容，請參閱 版本備註。

管理您的 Amazon DocumentDB 維護時段

每個叢集和每個執行個體都有自己的每週 30 分鐘維護時段，也就是排程修改和軟體修補執行的期間。大多數事件會在 30 分鐘內完成；較大的事件可能會執行更長的時間。

如果您在建立資源時未選擇時段，Amazon DocumentDB 會在隨機選取日期的 8 小時每日區塊內隨機指派一個時段。選擇將對應用程式的影響降至最低的時段，例如晚上或週末。

對於資料庫引擎升級，Amazon DocumentDB 會使用叢集的視窗，而不是個別執行個體的視窗。

下表顯示每個區域的預設時段。

區域名稱	區域	UTC 時間區塊
美國東部 (俄亥俄)	us-east-2	03:00-11:00
美國東部 (維吉尼亞北部)	us-east-1	03:00-11:00
美國西部 (奧勒岡)	us-west-2	06:00-14:00
Africa (Cape Town)	af-south-1	03：00–11：00
亞太地區 (香港)	ap-east-1	06:00-14:00
亞太地區 (海德拉巴)	ap-south-2	06：30–14：30
亞太地區 (馬來西亞)	ap-southeast-5	13:00-21:00
亞太區域 (孟買)	ap-south-1	06:00-14:00
亞太地區 (大阪)	ap-northeast-3	12：00-20：00
亞太區域 (首爾)	ap-northeast-2	13:00-21:00
亞太區域 (新加坡)	ap-southeast-1	14：00-22：00
亞太地區 (悉尼)	ap-southeast-2	12：00-20：00

區域名稱	區域	UTC 時間區塊
亞太地區 (雅加達)	ap-southeast-3	08：00-16：00
亞太地區 (墨爾本)	ap-southeast-4	11：00-19：00
亞太區域 (泰國)	ap-southeast-7	15：00-23：00
亞太區域 (東京)	ap-northeast-1	13:00-21:00
加拿大 (中部)	ca-central-1	03:00-11:00
加拿大西部 (卡加利)	ca-west-1	18：00-02：00
中國 (北京)	cn-north-1	06:00-14:00
中國 (寧夏)	cn-northwest-1	06:00-14:00
歐洲 (法蘭克福)	eu-central-1	21：00-05：00
歐洲 (蘇黎世)	eu-central-2	02：00-10：00
歐洲 (愛爾蘭)	eu-west-1	22:00-06:00
歐洲 (倫敦)	eu-west-2	22:00-06:00
歐洲 (米蘭)	eu-south-1	02：00-10：00
Europe (Paris)	eu-west-3	23：59-07：29
歐洲 (西班牙)	eu-south-2	02：00–10：00
Europe (Stockholm)	eu-north-1	04：00–12：00
墨西哥 (中部)	mx-central-1	03:00-11:00
中東 (阿拉伯聯合大公國)	me-central-1	05：00–13：00
南美洲 (聖保羅)	sa-east-1	00：00-08：00
以色列 (特拉維夫)	il-central-1	04：00-12：00

區域名稱	區域	UTC 時間區塊
AWS GovCloud （美國東部）	us-gov-east-1	17：00-01：00
AWS GovCloud （美國西部）	us-gov-west-1	06:00-14:00

變更 Amazon DocumentDB 維護時段

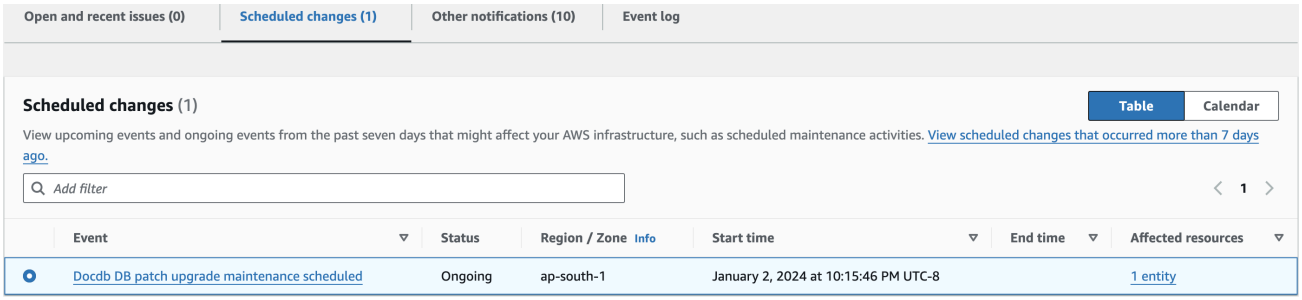
選擇您可以的最低流量時段，並在流量模式轉移時隨著時間進行調整。只有在系統變更 - 例如擴展儲存操作或執行個體類別變更 - 需要中斷，且只要該變更實際需要，叢集或執行個體才會在時段內無法使用。

變更維護時段

- 針對叢集：請參閱修改 Amazon DocumentDB 叢集。
- 針對執行個體：請參閱修改 Amazon DocumentDB 執行個體。

Amazon DocumentDB 引擎修補程式的通知

當 AWS 區域中可用的必要引擎修補程式時，在該區域中具有受影響 Amazon DocumentDB 叢集的每個 AWS 帳戶都會透過電子郵件 Health 儀板表 (AHD) 和 （傳送至 AWS 帳戶的根使用者地址） 收到通知。每個受影響的 Amazon DocumentDB 引擎版本都會傳送一個通知。您可以在 AHD 中的排程變更下找到它們。每個通知都會列出修補程式可用性時間、自動套用排程、受影響的叢集，以及版本備註。



所需的引擎修補程式會遵循大約 30 天的單一前置時間。當您的 區域有可用的修補程式時，Amazon DocumentDB 會傳送上述通知。此時，修補程式的 AutoAppliedAfterDate 會在大約 30 天後設定為 。在該日期之前，修補程式會保持待定狀態：您可以隨時套用，或將叢集的維護時段移至稍後日期來延遲。在 或之後AutoAppliedAfterDate，修補程式會在叢集的下一個維護時段自動套用。

例如，在 2026 年 6 月 1 日可用的必要修補程式，其 AutoAppliedAfterDate大約是 2026 年 7 月 1 日。您會在 2026 年 6 月 1 日收到通知，而且如果您不採取任何動作，修補程式會在 2026 年 7 月 1 日當天或之後叢集的第一個維護時段內自動套用。

收到通知後，您有兩個選項：在自動套用日期之前自行套用修補程式，或在即將到來的維護時段（預設值）等待它自動套用。若要自行套用，請開啟叢集的維護與備份索引標籤，並尋找類型 的項目system-update。

> **Note**
>
> AHD 中的通知狀態會持續進行，直到 Amazon DocumentDB 發行另一個具有新修補程式版本的引擎修補程式為止。
> 套用修補程式後，叢集的引擎修補程式版本會更新以符合通知中的版本。執行 驗證新版本db.runCommand({getEngineVersion: 1})。

選用修補程式和新的次要版本不會產生 AHD 或電子郵件通知。若要追蹤它們，請觀看 Amazon DocumentDB 版本備註。

強制修補程式（最罕見的類別，預留給最關鍵的安全性修正）也會透過 AHD 和電子郵件宣布。與必要的修補程式不同，它們會在維護時段之外套用，因此上述自動套用計時範例不適用。

以程式設計方式回應修補程式通知

AWS Health 與 Amazon EventBridge 整合，可讓您跨超過 20 個目標建置事件驅動型應用程式，包括 AWS Lambda 和 Amazon Simple Queue Service (SQS)。若要以程式設計方式對引擎修補程式可用性做出反應，請針對AWS_DOCDB_DB_PATCH_UPGRADE_MAINTENANCE_SCHEDULED事件設定 EventBridge。您可以從那裡擷取事件資料、引發其他事件、透過 傳送推播通知 AWS Console Mobile Application，或採取您所需的任何其他動作。

如果 Amazon DocumentDB 取消修補程式（罕見），您會收到 AHD 通知和有關取消的電子郵件。使用AWS_DOCDB_DB_PATCH_UPGRADE_MAINTENANCE_CANCELLED事件碼搭配 Amazon EventBridge 來處理此案例。如需撰寫規則的詳細資訊，請參閱《Amazon EventBridge 使用者指南》。

檢視待定的 Amazon DocumentDB 維護動作

使用 AWS 管理主控台 或 AWS CLI 來檢查叢集或執行個體的維護待定。

待定更新會以動作類型 顯示system-update，其中涵蓋引擎修補程式和作業系統更新。

當更新處於待定狀態時，您可以：

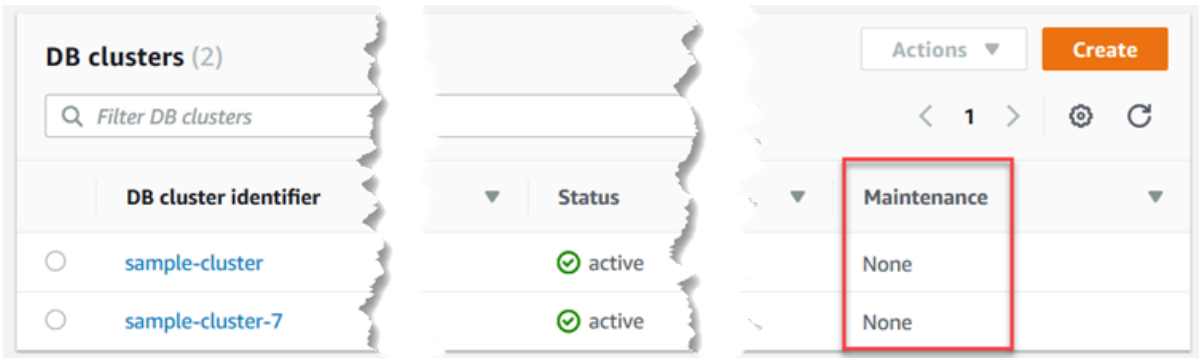
- 立即套用。
- 排程下一個維護時段。
- 透過在 之前變更維護時段來推遲它 （僅限引擎修補程式和作業系統更新）`AutoAppliedAfterDate`。在該日期過後，動作會在下一個維護時段自動套用。一旦`ForcedApplyDate`通過，就無法再延遲。

> **Note**
>
> 如果您不採取任何動作，則需要執行必要的維護動作，例如在即將到來的維護時段自動套用所需的引擎修補程式。選用的修補程式和次要版本永遠不會自動套用。

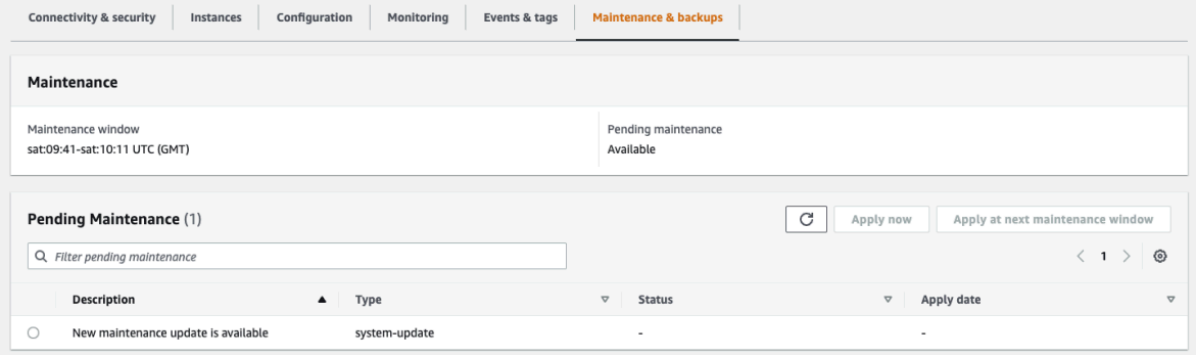
維護時段控制待定操作開始的時間，而不是完成所需的時間。

Using the AWS 管理主控台

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。
3. 當更新待定時，叢集的維護欄會顯示可用、必要或下一個時段。



4. 開啟叢集，然後選擇維護和備份，以查看待定的維護項目並對其採取行動。



Using the AWS CLI

執行 `describe-pending-maintenance-actions`以查看待定項目。下列範例顯示沒有待處理動作的帳戶。

```
aws docdb describe-pending-maintenance-actions
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "PendingMaintenanceActions": []
}
```

具有待定動作的帳戶會傳回如下所示的輸出：

```
{
    "PendingMaintenanceActions": [
        {
            "ResourceIdentifier": "arn:aws:rds:us-
east-1:123456789012:cluster:sample-cluster",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "system-update",
                    "Description": "db-version-upgrade",
                    "CurrentApplyDate": "2026-05-15T03:01:00Z",
                    "AutoAppliedAfterDate": "2026-05-15T03:01:00Z"
                }
            ]
        }
    ]
}
```

您可以將清單範圍限定為具有 的特定叢集`--filters`，格式為 `Name=filter-name,Values=resource-id,...`。接受的篩選條件`Name`是 `db-cluster-id`，會取得叢集識別符或 ARNs的清單。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb describe-pending-maintenance-actions \
   --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

針對 Windows：

```
aws docdb describe-pending-maintenance-actions ^
   --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

套用日期

每個待定維護動作最多會套用三個套用日期。它們會出現在 的 AWS CLI 輸出中`describe-pending-maintenance-actions`，並指出動作何時執行。欄位`null`用於選用維護。

- **`CurrentApplyDate`**- 動作排定執行時，現在或下一個維護時段。填入必要和強制動作。
- **`AutoAppliedAfterDate`**- 在叢集或執行個體維護時段期間自動套用開始的日期。填入必要動作。
- **`ForcedApplyDate`**- 硬性截止日期。在此日期之後，無論維護時段為何，動作都會自動執行。填入強制動作。

若要延遲待定動作，請將您的維護時段移至 之前的隔天`AutoAppliedAfterDate`。一旦`AutoAppliedAfterDate`通過，動作會在下一個維護時段自動套用。一旦`ForcedApplyDate`通過，就無法再延遲。每個修補程式的確切延遲時段各不相同；日期會發佈在 AHD 通知和 AWS CLI 輸出中。

Amazon DocumentDB 引擎更新

當您識別待定引擎修補程式時，請使用下列其中一個程序來套用或排程它。您可以從 AWS 管理主控台或 執行這些程序 AWS CLI。

Using the AWS 管理主控台

管理叢集的更新

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。
3. 選取您要更新的叢集。
4. 從動作功能表中，選擇：

   - 立即升級 - 立即執行待定維護。
   - 在下一個時段升級 - 在叢集的下一個維護時段執行。

   您也可以從叢集維護和備份索引標籤的待定維護區段，使用立即套用或在下一個維護時段套用（請參閱 檢視待定的 Amazon DocumentDB 維護動作)。

   > **Note**
   >
   > 如果沒有待處理項目，則所有這些選項都會處於非作用中狀態。

Using the AWS CLI

使用 套用待定更新`apply-pending-maintenance-action`。

Parameters

- **`--resource-identifier`**- 待定動作目標之資源的 Amazon DocumentDB Amazon Resource Name (ARN)。
- **`--apply-action`**- 要套用的待定維護動作。使用 `system-update` 套用引擎修補程式。
- **`--opt-in-type`**- 選擇加入請求的類型，或是否要復原。有效值：
  - `immediate`- 立即套用。提交後就無法復原。
  - `next-maintenance`- 在資源的下一個維護時段套用。
  - `undo-opt-in`- 取消現有的`next-maintenance`選擇加入。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb apply-pending-maintenance-action \
    --resource-identifier arn:aws:rds:us-east-1:123456789012:db:sample-cluster-instance-1 \
    --apply-action system-update \
    --opt-in-type immediate
```

針對 Windows：

```
aws docdb apply-pending-maintenance-action ^
    --resource-identifier arn:aws:rds:us-east-1:123456789012:db:sample-cluster-instance-1 ^
    --apply-action system-update ^
    --opt-in-type immediate
```

修補期間的讀取可用性

當叢集有多個執行個體時，Amazon DocumentDB 引擎 5.0 和 8.0 會在修補期間保留讀取可用性。Amazon DocumentDB 會以滾動方式在三個群組中修補讀取器執行個體，因此剩餘的讀取器會繼續為流量提供服務。寫入器在修補時短暫無法使用。若要實現零讀取停機時間，請設定您的讀取偏好設定，讓讀取可能會落回寫入器： `secondaryPreferred` 或 `primaryPreferred` 工作；`primary`或`secondary`單獨使用可能會導致讀取停機時間。

讀取偏好設定模式	在寫入器升級期間	在讀取器升級期間	零讀取停機時間所需的讀取器數目下限
`primary`	讀取/寫入停機時間	沒有影響	N/A
`primaryPreferred`	寫入停機時間	沒有影響	1
`secondary`	寫入停機時間	讀取停機時間（如果只有一個讀取器）	2
`secondaryPreferred`	寫入停機時間	沒有影響	1

讀取偏好設定模式	在寫入器升級期間	在讀取器升級期間	零讀取停機時間所需的讀取器數目下限
nearest	寫入停機時間	沒有影響	1

當讀取器進行修補時，整體叢集讀取輸送量會暫時下降。若要保持輸送量穩定，請在升級之前佈建其他讀取器，並在完成之後將其移除。

在引擎 3.6 和 4.0 上，這些讀取可用性功能不適用：引擎修補程式會導致更長的停機時間，這會影響讀取和寫入。若要升級至主要版本，請參閱 Amazon DocumentDB 就地主要版本升級。

修補程式停機時間長度

引擎修補停機時間會有所不同。最大的因素是修補時執行個體上的 CPU 使用率和記憶體壓力，因此正確調整執行個體的大小至關重要。若要將停機時間降至最低，請執行最新的 Amazon DocumentDB 主要引擎版本，並將執行個體分散到多個可用區域。

修補程式更新和替換

Amazon DocumentDB 會在發行後監控修補程式。在極少數情況下，Amazon DocumentDB 會在準備更新版本時暫停推展。發生這種情況時，尚未收到修補程式的叢集不會再將其視為可用的維護動作，並且 Health 儀板表 會撤銷 中對應的排程變更通知。已執行受影響版本的叢集會繼續正常運作，不需要您採取任何動作。

更新後的修補程式很快就會出現。當您的 區域可用時，您會透過 Health 儀板表 和電子郵件收到新的通知，如中所述Amazon DocumentDB 引擎修補程式的通知。

次要版本升級

Amazon DocumentDB 會在主要版本 5.0 和更新版本（例如 5.0.1) 上發佈次要版本。次要版本不會針對 5.0 之前的主要版本發佈。次要版本的行為與必要和選用的引擎修補程式不同：

- 它們不會顯示為待定的維護動作，也不會自動套用。
- 它們不會產生 AHD 或電子郵件通知。Amazon DocumentDB 版本備註中會宣布新的次要版本。
- 若要升級，請修改叢集的引擎版本（立即或在下一個維護時段期間）。次要版本升級需要短暫的停機時間，且為單向 - 您無法降級至較早的次要版本。對於全域叢集，請在主要叢集之前升級次要叢集。

閱讀更多：Amazon DocumentDB 次要版本升級。

Amazon DocumentDB 作業系統更新

執行個體偶爾需要作業系統更新。Amazon DocumentDB 會更新作業系統，以改善效能並加強安全性。作業系統更新會讓叢集引擎版本和執行個體類別保持不變。與引擎修補程式一樣，作業系統更新使用本主題頂端所述的選用 / 必要 / 強制生命週期；與引擎修補程式不同，如果您延遲更新，作業系統更新可以隨著時間轉換這些類別。在作業系統更新可用時立即套用，並將叢集和執行個體維護時段設定為符合您業務需求的時間。

使用叢集層級`os-upgrade`維護動作，將作業系統更新套用至叢集中的所有執行個體。Amazon DocumentDB 會以滾動方式更新執行個體，一次更新幾個執行個體，最後更新主要執行個體，將容錯移轉降至最低。更新會在您設定的叢集維護時段執行，而非個別執行個體維護時段。

執行個體收到作業系統更新後，其緩衝區快取會開始空白。在從儲存磁碟區重新填入工作集之前，該執行個體上的查詢可能會遇到較高的延遲和較低的 `BufferCacheHitRatio`。

當 Amazon DocumentDB 更新主要執行個體時，容錯移轉會將複本提升為新的主要執行個體。使用叢集端點，讓您的應用程式以透明的方式處理。若要在執行個體更新時維持讀取可用性，請將您的讀取偏好設定設為 `secondaryPreferred`或 ，`primaryPreferred`讓讀取可以回到可用的執行個體。將潛在的容錯移轉目標 （ 具有最高優先順序層的複本 ） 保留在與主要執行個體相同的執行個體類別。這可避免提升後的寫入效能降低。如需詳細資訊，請參閱Amazon DocumentDB 容錯移轉。

叢集層級`os-upgrade`和執行個體層級`system-update`動作可能會在 中同時顯示`describe-pending-maintenance-actions`為可用動作。不過，您無法同時排程兩者。如果在任何執行個體上主動排程執行個體層級`system-update`動作，您必須在排程叢集層級`os-upgrade`動作之前取消或完成這些動作，反之亦然。

> **Important**
>
> 您的 Amazon DocumentDB 執行個體會離線以進行作業系統更新。多執行個體叢集可將影響降至最低。如果您執行單一執行個體叢集，您可以暫時為更新新增次要叢集，並在之後將其移除。次要 會在存在時產生一般費用。

> **Note**
>
> 執行個體層級system-update動作仍可用於回溯相容性。如果您必須使用它，請先更新複本，最後更新主要複本，避免同時修補複本，因為在修補期間進行容錯移轉可能會延長停機時間。

若要在新的選用作業系統更新送達時取得事件，請在安全修補事件類別RDS-EVENT-0230中訂閱 。如需詳細資訊，請參閱訂閱 Amazon DocumentDB 事件。

> **Note**
>
> 可能需要保持最新選用和必要的更新，才能合規。在維護時段期間定期套用os-upgrade動作。

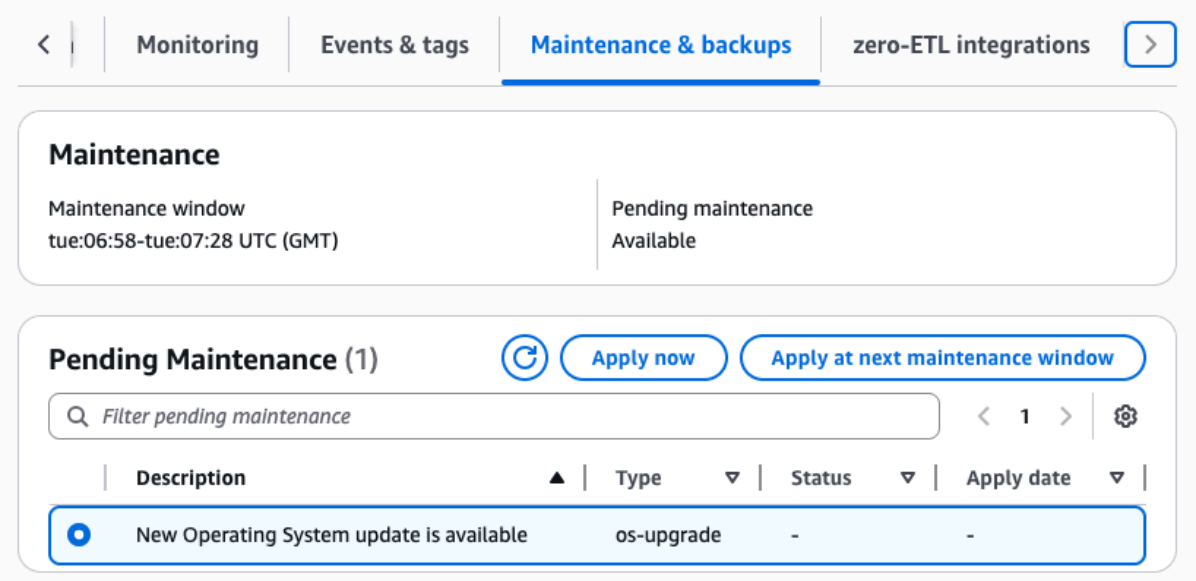
作業系統更新與特定執行個體類別相關聯，因此不同的執行個體在不同時間符合資格。如果您的叢集不在最新的引擎修補程式上，作業系統更新可能不會出現，請先套用最新的引擎修補程式 （ 請參閱 Amazon DocumentDB 引擎更新)。

使用 AWS 管理主控台 或 AWS CLI 來檢查是否有可用的更新。

Using the AWS 管理主控台

若要從主控台檢查作業系統更新：

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集，然後選取叢集名稱。
3. 選擇維護與備份索引標籤。
4. 在待維護下，如果作業系統更新可用，則會顯示 os-upgrade動作。



5. 選取os-upgrade動作，然後選擇立即套用或在下一個維護時段套用。如果值是下一個視窗，只要動作尚未啟動，您就可以延遲延遲升級。

Using the AWS CLI

檢查是否有待定的作業系統更新：

```
aws docdb describe-pending-maintenance-actions
```

```
{
    "PendingMaintenanceActions": [
        {
            "ResourceIdentifier": "arn:aws:rds:aa-
example-1:111122223333:cluster:sample-cluster",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "os-upgrade",
                    "Description": "New Operating System update is available"
                }
            ]
        },
        {
            "ResourceIdentifier": "arn:aws:rds:aa-example-1:111122223333:db:sample-
cluster-instance-1",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "system-update",
```

```
                    "Description": "New Operating System update is available"
                }
            ]
        },
        {
            "ResourceIdentifier": "arn:aws:rds:aa-example-1:111122223333:db:sample-
cluster-instance-2",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "system-update",
                    "Description": "New Operating System update is available"
                }
            ]
        }
    ]
}
```

作業系統更新在叢集層級顯示為 os-upgrade，在執行個體層級顯示為 system-update。使用叢集層級os-upgrade動作。

Example

下列範例會立即套用作業系統更新。

針對 Linux、macOS 或 Unix：

```
aws docdb apply-pending-maintenance-action \
    --resource-identifier arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster \
    --apply-action os-upgrade \
    --opt-in-type immediate
```

針對 Windows：

```
aws docdb apply-pending-maintenance-action ^
    --resource-identifier arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster ^
    --apply-action os-upgrade ^
    --opt-in-type immediate
```

使用者啟動的更新

您自行開始的一些變更，例如，將執行個體類別換成具有更多或更少記憶體的執行個體類別，或變更叢集的參數群組。Amazon DocumentDB 對待這些內容的方式與其啟動的更新不同。如需詳細資訊，請參閱：

- 修改 Amazon DocumentDB 叢集
- 修改 Amazon DocumentDB 執行個體

若要列出仍在擱置中的使用者起始變更：

Example

列出執行個體的待處理使用者起始變更

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-instances \
    --query 'DBInstances[*].
[DBClusterIdentifier,DBInstanceIdentifier,PendingModifiedValues]'
```

針對 Windows：

```
aws docdb describe-db-instances ^
    --query 'DBInstances[*].
[DBClusterIdentifier,DBInstanceIdentifier,PendingModifiedValues]'
```

此操作的輸出將會如下所示 (JSON 格式)。

在此範例中，`sample-cluster-instance`有 的待定變更`db.r5.xlarge`；`sample-cluster-instance-2`沒有。

```
[
    [
        "sample-cluster",
        "sample-cluster-instance",
        {
            "DBInstanceClass": "db.r5.xlarge"
        }
```

```
    ],
    [
        "sample-cluster",
        "sample-cluster-instance-2",
        {}
    ]
]
```

全域叢集修補

在全域叢集中，每個成員叢集的主要和次要叢集都會在自己的維護時段升級。當每個區域都提供必要的引擎修補程式時，您會收到 AHD 和電子郵件通知。選用修補程式和新的次要版本不會產生通知；請查看 Amazon DocumentDB 版本備註。

如果您自行套用，一律先修補次要項目，最後修補主要項目。此順序可讓容錯移轉和切換在整個推出期間保持可用。

> **Important**
>
> 如果您錯誤地先修補主要版本，請盡快將所有次要版本更新為相同的版本。容錯移轉和切換會保持停用狀態，直到每個叢集都位於相同版本為止。

如果您不採取任何動作，修補程式會在每個叢集的下一個維護時段自動套用：先進行次要作業，然後在次要作業完成後於其時段套用主要作業。

將主要和次要資料庫叢集保留在相同版本上。受管跨區域容錯移轉只有在每個叢集共用相同的引擎版本和修補程式層級時，才適用於全球資料庫。如果您新增使用較主要版本較新引擎版本的新次要 - 在主要版本上建立新的次要 ，再將其加入全域資料庫，則同樣適用。

修補程式通知之後，請盡早將主要和次要 升級至最新版本，以保持容錯移轉和切換運作。如果容錯移轉或切換請求遭拒，請比較跨叢集的引擎修補程式版本；如果不相符，請在延遲叢集上套用可用的修補程式。

了解服務連結角色

Amazon DocumentDB （與 MongoDB 相容） 使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是直接連結至 Amazon DocumentDB 的唯一 IAM 角色類型。服務連結角色是由 Amazon DocumentDB 預先定義，並包含該服務代表您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓您更輕鬆地使用 Amazon DocumentDB，因為您不必手動新增必要的許可。Amazon DocumentDB 會定義其服務連結角色的許可，除非另有定義，否則只有 Amazon DocumentDB 可以擔任其角色。定義的許可包括信任政策和許可政策，並且該許可政策不能附加到任何其他 IAM 實體。

您必須先刪除角色的相關資源，才能刪除角色。這可保護您的 Amazon DocumentDB 資源，因為您不會不小心移除存取資源的許可。

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

Amazon DocumentDB 服務連結角色許可

Amazon DocumentDB （具有 MongoDB 相容性） 使用名為 AWSServiceRoleForRDS 的服務連結角色，以允許 Amazon DocumentDB 代您的叢集呼叫 AWS 服務。

AWSServiceRoleForRDS 服務連結角色信任下列服務可擔任該角色：

- `docdb.amazonaws.com`

角色許可政策允許 Amazon DocumentDB 對指定的資源完成下列動作：

- 在 `ec2` 上的動作：
  - `AssignPrivateIpAddresses`
  - `AuthorizeSecurityGroupIngress`
  - `CreateNetworkInterface`
  - `CreateSecurityGroup`
  - `DeleteNetworkInterface`
  - `DeleteSecurityGroup`
  - `DescribeAvailabilityZones`
  - `DescribeInternetGateways`
  - `DescribeSecurityGroups`
  - `DescribeSubnets`
  - `DescribeVpcAttribute`
  - `DescribeVpcs`
  - `ModifyNetworkInterfaceAttribute`

  - RevokeSecurityGroupIngress
  - UnassignPrivateIpAddresses
- 在 sns 上的動作：
  - ListTopic
  - Publish
- 在 cloudwatch 上的動作：
  - PutMetricData
  - GetMetricData
  - CreateLogStream
  - PullLogEvents
  - DescribeLogStreams
  - CreateLogGroup

> **Note**
>
> 您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。您可能會遇到下列錯誤訊息：
> 無法建立資源。請確認您擁有建立服務連結角色的許可。否則請等待，然後再試一次。
> 如果您看到此錯誤，請確認您已啟用以下許可：
>
> ```
> {
>     "Action": "iam:CreateServiceLinkedRole",
>     "Effect": "Allow",
>     "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
> AWSServiceRoleForRDS",
>     "Condition": {
>         "StringLike": {
>             "iam:AWSServiceName":"rds.amazonaws.com"
>         }
>     }
> }
> ```

如需詳細資訊，請參閱《IAM 使用者指南》中的服務連結角色許可。

建立 Amazon DocumentDB 服務連結角色

您不需要手動建立服務連結角色，當您建立叢集時，Amazon DocumentDB 會為您建立服務連結角色。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。當您建立叢集時，Amazon DocumentDB 會再次為您建立服務連結角色。

修改 Amazon DocumentDB 服務連結角色

Amazon DocumentDB 不允許您修改 AWSServiceRoleForRDS 服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。不過，您可以使用 IAM 修改角色的描述。如需詳細資訊，請參閱《IAM 使用者指南》中的編輯服務連結角色。

刪除 Amazon DocumentDB 服務連結角色

如果您不再需要使用需要服務連結角色的功能或服務，請刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。不過您必須先刪除您的所有 叢集，才能刪除服務連結角色。

清除 Amazon DocumentDB 服務連結角色

您必須先確認服務連結角色沒有作用中的工作階段，並移除該角色使用的資源，之後才能使用 IAM 將其刪除。

檢查服務連結角色是否有使用主控台的作用中工作階段

1. 登入 AWS 管理主控台 ，並在 開啟 IAM 主控台https://console.aws.amazon.com/iam/。
2. 在 IAM 主控台的導覽窗格中，選擇角色，然後選擇 AWSServiceRoleForRDS 角色的名稱 （ 而非核取方塊 ）。
3. 在所選角色的 Summary (摘要) 頁面中，選擇 Access Advisor (存取 Advisor) 分頁。
4. 在 Access Advisor (存取 Advisor) 標籤中，檢閱服務連結角色的近期活動。

> **Note**
>
> 如果您不確定 Amazon DocumentDB 是否使用 AWSServiceRoleForRDS 角色，您可以嘗試刪除該角色。如果服務正在使用該角色，則刪除會失敗，而您可以檢視正在使用該角色的 區域。如果服務正在使用該角色，您必須先等到工作階段結束，才能刪除該角色。您無法撤銷服務連結角色的工作階段。

如果您想要移除 AWSServiceRoleForRDS 角色，您必須先刪除所有執行個體和叢集。如需有關刪除執行個體和叢集的詳細資訊，請參閱下列主題：

- 刪除 Amazon DocumentDB 執行個體
- 刪除 Amazon DocumentDB 叢集

Amazon DocumentDB 服務連結角色支援的區域

Amazon DocumentDB 支援在所有提供服務的區域中使用服務連結角色。如需詳細資訊，請參閱https://docs.aws.amazon.com/documentdb/latest/devguide/regions-and-azs.html#regions-and-azs-availability。

效能最佳化

本章提供最佳化 Amazon DocumentDB 效能和疑難排解常見效能問題的指引。

主題

- 常見的效能問題
- 反面模式
- 效能改善秘訣
- 查詢計畫分析
- Amazon DocumentDB 的最佳實務
- 管理 Amazon DocumentDB 索引
- 管理集合層級文件壓縮
- 在 Amazon DocumentDB 8.0 中管理字典型壓縮

常見的效能問題

本節涵蓋 Amazon DocumentDB 可能遇到的常見效能問題，以及如何解決這些問題。

主題

- 高 CPU 使用率
- Amazon DocumentDB 的連線問題
- 查詢執行緩慢

高 CPU 使用率

高 CPU 使用率是您在使用 Amazon DocumentDB 時可能遇到的問題之一。本節提供解決問題的洞見。

識別 - 找出問題

Amazon DocumentDB 執行個體上的高 CPU 使用率會提高所有資料庫操作的延遲，導致應用程式速度變慢。當 Amazon DocumentDB CPU 用量超過最佳層級時，查詢執行時間會增加、建立連線需要更長的時間，以及應用程式回應能力降低。主要執行個體上的 CPU 峰值會減慢寫入操作，而複本上的峰值會導致讀取操作的延遲較高。

Amazon DocumentDB 中高 CPU 使用率的一些常見原因可能是：

- 集合掃描：沒有適當索引的查詢會強制完整集合掃描，耗用過多 CPU 來檢查每個文件。
- 連線峰值：應用程式連線突然激增，導致身分驗證和工作階段管理所需的 CPU 資源龐大。
- 複雜彙總：多階段操作（排序、分組、聯結）會耗用大量 CPU，尤其是在沒有最佳化索引的大型資料集。
- 低效率查詢：掃描大型結果集和 regex 操作會產生高 CPU 使用率。
- 寫入密集型工作負載：頻繁更新和大量插入會對 CPU 造成壓力，包括文件處理、索引更新和交易記錄。

調查 - 收集指標

Amazon DocumentDB 透過 Amazon CloudWatch 提供監控。Amazon DocumentDB 指標可以廣泛分類為叢集層級和執行個體層級指標。對於與 CPU、記憶體和連線相關的資源，請查看執行個體層級指標，因為資源消耗是執行個體特有的。識別顯示 CPU 峰值的執行個體（主要或複本），並記下峰值的時間。

適用於 的 CPU 使用率的 CloudWatch 指標`CPUUtilization`。

診斷 - 尋找根本原因

識別 CPU 峰值的執行個體和時間軸之後，接下來是診斷對 CPU 用量造成很大影響的原因。由於有多種原因可能導致 CPU 峰值，因此 讓我們找出造成 CPU 峰值的原因：

連線：檢查 CloudWatch 指標，包括 `DatabaseConnections`和 `DatabaseConnectionsMax`，如果連線中有峰值，並且與 CPU 峰值時間一致，則短時間內連線激增可能是 CPU 峰值的原因。

查詢：如果不是連線，則查詢可能是原因。如果您已啟用效能洞察，請前往執行個體的績效洞察主控台，並查看執行個體上的查詢。另一個要查看查詢的事項是來自分析器。

解決 - 修正問題

連線暴增：如果連線暴增是 CPU 峰值的原因，請考慮在應用程式中重複使用連線或實作連線集區。如需最佳化高連線的詳細資訊，請參閱 the section called “連線問題”。

查詢分佈：如果 CPU 峰值是由主要執行個體上的讀取查詢造成，如果您的查詢結果最終可以一致，請考慮將讀取操作重新導向至複本執行個體。

查詢最佳化：如果查詢或一組查詢導致 CPU 峰值，請考慮最佳化查詢，例如，如果查詢未使用索引，則新增索引。如需最佳化查詢的詳細資訊，請參閱 the section called “查詢執行緩慢”。

執行個體擴展：如果在最佳化工作後 CPU 使用率持續很高，請升級至具有更多 CPU 容量的大型執行個體類別，以有效地處理您的工作負載需求。

無伺服器：對於造成短期 CPU 尖峰的不可預測工作負載，請考慮使用 使用 Amazon DocumentDB 無伺服器。無伺服器執行個體會根據需求自動擴展資源，無需為此類工作負載佈建固定執行個體大小。

Amazon DocumentDB 的連線問題

識別 - 找出問題

常見原因

連線問題通常來自三個主要區域：

當應用程式透過用戶端連線集區限制或伺服器端執行個體限制達到其對 Amazon DocumentDB 允許的連線上限時，就會發生連線集區耗盡。當新的連線請求排入佇列或拒絕時，此條件會導致應用程式效能降低、逾時和潛在的失敗。

當 Amazon DocumentDB 遇到過多並行身分驗證請求時，尤其是在短時間內處理超過 1，000 個新連線時，就會發生身分驗證過載。在身分驗證維護期間，Amazon DocumentDB 會在工作階段映射上保留專屬鎖定，導致後續身分驗證嘗試排入佇列，直到維護完成為止。

Amazon DocumentDB 中的組態問題通常來自聯網、安全和用戶端設定中的組態錯誤。這包括不正確的安全群組設定、不正確的 VPC 組態或 SSL/TLS 憑證問題等項目。了解適當的組態對於維護安全可靠的資料庫存取至關重要。

診斷 - 尋找根本原因

連線集區

連線集區會在建立 MongoClient 執行個體時初始化。每個集區會根據兩個關鍵參數維護連線：

minPoolSize - 維護的連線數下限

maxPoolSize - 允許的最大連線數

當請求需要連線時：

1. 集區會檢查可用的閒置連線
2. 如果不存在且集區大小 < maxPoolSize，則會建立新的連線
3. 如果位於 maxPoolSize，請求會進入等待佇列
4. 如果佇列已滿或已達到逾時，則會擲回 MongoWaitQueueFullException

等待佇列的行為會透過下列參數處理：

waitQueueTimeoutMS - 連線的等待時間上限

waitQueueSize - 佇列請求上限

以下是連線至每次建立新集區之 Amazon DocumentDB 的有問題方法範例：

```
for(Request request : requests) {
    MongoClient client = MongoClients.create(settings);
    // Process request
    client.close();
}
```

要監控的關鍵 CloudWatch 指標為：

- `DatabaseConnections` - 在執行個體上以 1 分鐘的頻率開啟的連線數（作用中和閒置）。
- `DatabaseConnectionsMax` - 執行個體在 1 分鐘內開啟的資料庫連線（作用中和閒置）數目上限。
- `DatabaseConnectionsLimit` - 執行個體在任何指定時間允許的並行資料庫連線數目上限（作用中和閒置）。
- `LowMemNumOperationsThrottled` - 由於 1 分鐘內可用記憶體不足而調節的請求數量。

如需每個執行個體類別的配額，請參閱 執行個體配額。

應用程式層級連線集區問題的常見警告訊號包括：

- 增加連線擷取時間
- 增加等待佇列大小
- 逾時例外狀況數量增加

身分驗證過載

連線至 Amazon DocumentDB 的流程如下所示：

連線請求 → SSL 交握 → 身分驗證 → 工作階段建立 → 連線就緒

處理 >1，000 個新連線時，在完成 SSL 交握後，其他連線請求將進入身分驗證佇列。在這些過載事件期間，您應用程式的平均連線時間將會增加。

要監控的關鍵 CloudWatch 指標為：

- `DatabaseConnections` - 在執行個體上以 1 分鐘的頻率開啟的連線數（作用中和閒置）。
- `DatabaseConnectionsMax` - 執行個體在 1 分鐘內開啟的資料庫連線（作用中和閒置）數目上限。
- `DatabaseConnectionsLimit` - 執行個體在任何指定時間允許的並行資料庫連線數目上限（作用中和閒置）。

組態問題

最常見的組態問題是嘗試從無法存取私有網路環境的環境連線至私有 Amazon DocumentDB 叢集端點時所造成。Amazon DocumentDB 是僅限虛擬私有雲端 (VPC)，目前不支援公有端點。您無法從筆記型電腦或 VPC 外部的本機開發環境直接連線至 Amazon DocumentDB 叢集。

這將出現如下錯誤：

```
Error: couldn't connect to server...
Failed to connect to...
exception: connect failed
connection attempt failed
```

不正確的安全群組組態也可能導致連線失敗。根據預設，Amazon DocumentDB 叢集會接聽 TCP 連接埠 27017 上的連線。如果嘗試連線到與部署叢集時不同的連接埠，或者如果叢集的傳入安全群組組態中未涵蓋應用程式，您的應用程式將會失敗。

不正確的憑證管理也可能導致連線問題。根據預設，系統會為新建立的 Amazon DocumentDB 叢集啟用傳輸中加密。啟用傳輸中加密時，必須使用 TLS 的安全連線，才能使用 global-bundle.pem 憑證連線至叢集。如果您嘗試使用不正確的憑證，您將會收到錯誤，例如：

```
unable to get local issuer certificate
```

如果嘗試連接到啟用 TLS 的叢集，但未指定 TLS 參數，您將會收到錯誤，例如：

```
Server selection timed out after 30000 ms
```

解決 - 修正問題

連線集區：實作或調整集區大小以符合工作負載需求，以檢閱連線集區。最佳集區組態取決於您的工作負載和需求。您應該保留 minPoolSize，讓核心連線準備就緒且可用，如果集區已用盡，則 maxWaitTime 短到可以快速失敗。

以下是如何重複使用單一集區而不每次建立新集區的範例：

```
MongoClient client = MongoClients.create(settings);
    for(Request request : requests) {
    // Process request
}
```

身分驗證過載：透過實作漸進式連線提升，並一次限制 1，000 個新連線來管理身分驗證。使用連線集區有效地重複使用已驗證的連線。若要避免 Amazon DocumentDB 叢集的連線超載，請實作連線漸進升級策略。

```
public class ConnectionManager {
    private static final int BATCH_SIZE = 100;
    private static final int DELAY_MS = 1000;

    public void establishConnections(int totalRequired) {
        int established = 0;
        while (established < totalRequired) {
            int batch = Math.min(BATCH_SIZE, totalRequired - established);
            createConnections(batch);
            Thread.sleep(DELAY_MS);
            established += batch;
        }
    }
}
```

您也可以設定連線集區設定，以限制允許的連線總數。

```
MongoClientSettings settings = MongoClientSettings.builder()
    .applyToConnectionPoolSettings(builder -> {
        builder.maxSize(500)                    // Limit total connections
```

```
                .minSize(10)                        // Maintain base connections
                .maxConnectionLifeTime(3600000)     // Rotate connections hourly
    })
    .applyToServerSettings(builder -> {
        builder.heartbeatFrequency(10000)       // Regular server checks
    })
    .build();
```

組態問題：確保您的應用程式可存取 Amazon DocumentDB 資源所在的私有 VPC 和子網路。如果使用 VPC 對等互連，請參閱開發人員指南對 VPC 對等互連進行故障診斷以取得更多資訊。您也可以檢閱 知識中心文章如何疑難排解從網際網路到 VPC 內 Amazon EC2 執行個體的連線問題？

對於安全群組組態，您必須在 Amazon DocumentDB 安全群組中包含傳入規則，以允許來自應用程式的連線。

```
{
  "SecurityGroupIngress": [
    {
      "IpProtocol": "tcp",
      "FromPort": 27017,
      "ToPort": 27017,
      "SourceSecurityGroupId": "<application-security-group>",
      "Description": "DocumentDB access from application tier"
    }
  ],
  "SecurityGroupEgress": [
    {
      "IpProtocol": "-1",
      "FromPort": -1,
      "ToPort": -1,
      "CidrIp": "0.0.0.0/0"
    }
  ]
}
```

如果叢集設定為 TLS 加密，請下載名為 global-bundle.pem 的 Amazon DocumentDB 的 TLS 憑證，並在連線至叢集時使用它。

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

長期解決方案

透過升級至較大的執行個體類別或新增僅供讀取複本來分配連線負載，可能需要執行個體擴展。適當的負載平衡實作可確保整個叢集的最佳資源使用率。

應用程式變更應著重於實作強大的連線處理、全方位監控，以及遵循連線集區最佳實務。這包括適當的錯誤處理和連線生命週期管理。

架構改善可能涉及針對可變工作負載採用 Amazon DocumentDB Serverless、實作複雜的重試邏輯，以及設計容錯能力。考慮重組應用程式架構，以更好地處理連線管理。

最佳實務

連線集區

透過適當的連線集區管理和監控，應用程式可以維持穩定的資料庫連線，同時防止可能影響系統可靠性和效能的耗盡情況。根據工作負載的特性，設定適當的逾時並調整集區大小。

連線集區設定範例

```
MongoClientSettings settings = MongoClientSettings.builder()
    .applyToConnectionPoolSettings(builder ->
        builder.maxSize(10))
    .applyToConnectionPoolSettings(builder ->
        builder.maxWaitQueueSize(2))
    .applyToConnectionPoolSettings(builder ->
        builder.maxConnectionIdleTime(10, TimeUnit.MINUTES))
    .build();
```

如需詳細資訊，請參閱使用 Amazon DocumentDB 建置彈性應用程式 – 第 1 部分：用戶端組態。

身分驗證過載

一律根據您的工作負載為參數實作具有適當值的連線集區。使用逐步建立連線技術，並盡可能維持持久性連線。實作適當的連線清除，以確保不會浪費閒置資源。

組態問題

確定您已設定從應用程式到 Amazon DocumentDB 資源的適當路由。使用 TLS 進行傳輸中的加密，並實作最低權限存取。驗證您的 Amazon DocumentDB 登入資料並驗證連線字串值。

查詢執行緩慢

識別 - 找出問題

慢速執行查詢是超過您業務需求預期執行時間的查詢。構成慢查詢的定義會因不同的工作負載而有所不同。您可以透過分析器日誌或績效詳情來識別慢查詢。

1. 如果尚未啟用，請啟用分析器。Profiler 日誌會寫入日誌群組下的 Amazon CloudWatch：`/aws/docdb/<cluster-identifier>/profiler`。使用 CloudWatch Logs Insights 來查詢它們。

   取得電子商務產品集合 10 個最慢查詢的 CloudWatch 日誌分析查詢範例：

   ```
   filter ns="ecommerce.products" | sort millis desc | limit 10
   ```

2. 使用績效詳情近乎即時地識別昂貴的查詢。如果尚未啟用績效詳情，請將其啟用。

   a. 開啟 AWS 主控台，導覽至 Amazon Amazon DocumentDB，然後 Performance Insights，然後選取您的叢集執行個體。

   b. 檢閱資料庫載入 (AAS) 時間軸和熱門查詢 （依資料庫載入）。展開查詢摘要以檢視文字子陳述式。

   c. 擷取需要分析的查詢。

   > **Note**
   >
   > 並非所有績效詳情中的查詢都無效或緩慢查詢。

調查 — 收集資訊

1. 分析工具提供與其相關聯的查詢執行計畫和關鍵指標，包括毫秒、Nreturned 和 planSummary （索引用量）：

   ```
   {
       "op": "query",
       "ts": 1721374275673,
       "ns": "test.perf",
       "command": {
   ```

```
        "find": "perf",
        "filter": {
            "threadRunCount": 0
        },
        "$db": "test",
        "lsid": {
            "id": {
                "$binary": "oO2wEtpgQIK+y9KGByYnsw==",
                "$type": "4"
            }
        },
        "$readPreference": {
            "mode": "secondaryPreferred"
        }
    },
    "cursorExhausted": true,
    "nreturned": 0,
    "responseLength": 0,
    "protocol": "op_query",
    "millis": 137,
    "planSummary": "IXSCAN",
    "execStats": {
        "stage": "FETCH",
        "nReturned": "0",
        "executionTimeMillisEstimate": "100.346",
        "inputStage": {
            "stage": "IXSCAN",
            "nReturned": "0",
            "executionTimeMillisEstimate": "100.342",
            "indexName": "threadRunCount_1"
        }
    },
    "client": "172.31.6.165:43154",
    "appName": "ProdAppTester14",
    "user": "adminuser"
}
```

若要尋找 COLLSCAN 查詢：

```
filter planSummary="COLLSCAN" | sort millis desc | limit 20
```

2. 使用績效詳情來即時分析查詢執行趨勢，例如等待、載入和資源影響 （例如每個查詢的 IOPS 或 CPU)。

   由於 Performance Insights 不提供查詢執行計畫，請擷取查詢，並在 Amazon DocumentDB 叢集上的查詢上執行 explain("executionStats")：

   ```
   db.ecommerce.products.find().explain("executionStats")
   ```

3. 或者，將分析器指標與績效詳情資料建立關聯 （例如，將分析器中的高毫秒查詢與績效詳情中的常用查詢進行比對）。

診斷 — 尋找根本原因

在此步驟中，診斷查詢計畫以識別潛在的最佳化。使用下列流程 - 症狀 → 可能原因：

- 徵狀： planSummary： "COLLSCAN"

  原因：索引遺失或不正確。

- 徵狀：使用 $group / $sort 緩慢彙總

  原因：彙總管道可能正在處理記憶體中的過多資料。

- 徵狀：高延遲，而績效詳情會顯示依 IO 分組的資料庫負載等待，但已使用索引 (planSummary： "IXSCAN")。

  原因：I/O 繫結查詢；索引或工作集超過執行個體上可用的緩衝區快取。

- 徵狀：PI 顯示 CPU 等待狀態，由於查詢很少，因此 AAS 很高

  原因：CPU 繫結查詢 （複雜彙總、$regex)。

- 徵狀：許多緩慢的查詢，但沒有顯示昂貴的 planSummary

  原因：外部瓶頸 （網路、限流、維護任務、查詢量）。

解決 — 修正問題

當您實作修正時，請務必使用 explain("executionStats") 驗證並監控 Performance Insights 資料庫負載。

1. planSummary：「COLLSCAN」
   - 建立目標索引。

     範例：針對依 { category， price } 篩選並依價格遞減排序的頻繁查詢：

     ```
     db.products.createIndex({ category: 1, price: -1 })
     ```

2. 使用 $group / $sort 緩慢彙總
   - 提早推送 $match 和 $project，以減少流入 $group / $sort 的文件。
   - 儘早限制管道中的欄位數量：

     ```
     db.products.explain("executionStats").aggregate([
       { $match: { category: "Electronics", price: { $gt: 0 } } },   // early filter
       { $project: { price: 1, category: 1 } },                     // reduce document
      size
       { $group: { _id: "$category", avgPrice: { $avg: "$price" } } }
     ])
     ```

3. 高延遲，而績效詳情顯示依 IO 分組的資料庫負載等待
   - 驗證 BufferCacheHitRatio 是否低，或執行個體上的 ReadIOPS 是否高。增加執行個體記憶體（升級執行個體類別，例如 r6g.large → r6g.xlarge) 或使用 NVMe 執行個體類別。
   - 減少索引足跡。
   - 新增僅供讀取複本以卸載讀取流量（如果查詢可容忍最終一致性，請使用 readPreference 設定來重新導向複本上的流量）。
4. PI 顯示 CPU 等待狀態，由於查詢很少，因此 AAS 很高
   - 以索引字首搜尋或 $text index 取代昂貴的 $regex。
   - 批次寫入（插入）以減少寫入放大。

> **Note**
>
> 在提升生產變更之前，請務必在較低的環境中測試您的變更（非生產）。

反面模式

本節涵蓋可能會對 Amazon DocumentDB 效能產生負面影響的常見反模式。

主題

- 具有 3 個以上屬性的複合索引
- 長時間執行的查詢
- 低使用率和備援索引
- 具有大型陣列的多重索引鍵

具有 3 個以上屬性的複合索引

概觀

複合或複合索引會維護單一索引結構中多個欄位的參考。這些索引會針對同時篩選多個欄位或結合篩選與排序操作的查詢最佳化效能。它們也適用於最左邊索引欄位上的單一條件查詢。資料庫會利用這些索引項目，有效率地尋找相符的文件，而無需執行完整的集合掃描。

Amazon DocumentDB 可以使用複合索引來支援查詢，其中包含索引欄位的任何主要子集，這個概念稱為索引字首。例如，如果您在 上有複合索引`{state: 1, city: 1, zipcode: 1}`，Amazon DocumentDB 可以有效率地處理使用欄位 'state' 單獨、欄位 'state' 和 'city' 一起，或所有三個欄位 'state'、'city' 和 'zipcode' 的查詢。不過，查詢必須使用從左到右的欄位，而不會略過其中的任何欄位。這表示僅使用欄位 'city' 或 'zipcode' 或組合的查詢，例如 'state' 和 'zipcode'（略過 'city'），無法完全利用索引。

在大多數真實世界案例中，具有三個或更少屬性的複合索引通常會達到最佳效能和資源效率。雖然複合索引中有超過 3 個屬性是可行的，但通常會導致資源消耗增加，超過優點。

對叢集的影響

雖然想要建立涵蓋查詢中所有條件的索引，以達到最大理論效能，但大多數資料篩選是透過複合索引中的前 1-3 個屬性進行。超過此閾值的其他欄位主要有助於索引大小，而不是有意義的查詢最佳化。

儲存體和 I/O 額外負荷：具有許多屬性的複合索引會耗用比較簡單替代方案更多的儲存空間。大小與索引的屬性數量以及索引值本身的大小成正比。

記憶體佔用空間：具有許多屬性的複合索引的大型儲存佔用空間會在記憶體中建立對應的佔用空間，讓您的工作集變大，並從緩衝集區取代其他經常存取的資料。

寫入操作：每個影響多個索引欄位的文件修改都需要更新整個複合索引項目，乘以完成寫入操作所需的工作。

如何識別

首先檢閱集合中的所有索引，以識別具有三個以上屬性的複合索引：

```
// List all indexes for the collection
db.collection.getIndexes()

// Look for indexes with 3+ fields like:

// { "userId": 1, "status": 1, "category": 1, "priority": 1, "region": 1 }
// { "orderId": 1, "customerId": 1, "productId": 1, "timestamp": 1, "warehouse": 1 }
```

修補

如果使用具有三個以上屬性的複合索引，請識別使用它的查詢，並尋找最佳化機會。請考慮以包含三個或更少屬性的更有效率索引取代這些索引。實作新索引後，移除包含三個以上屬性的舊索引。

> **Note**
>
> 在捨棄任何索引之前，請務必與利益相關者協調並驗證效能影響。

建立複合索引時，請遵循等式、排序、範圍 (ESR) 規則。

等式排序範圍 (ESR) 規則：此排序會先以等式條件篩選資料集，然後在減少的集合上套用排序操作，最後在最小的可能子集上執行範圍掃描，藉此最大化索引效率

- 將欄位排序為等式 （ 完全相符 ），
- 排序 （ 排序 ）、
- 範圍 (> 、 < 、$in)。

```
// Query pattern
db.orders.find({
```

```
  userId: "user123",                    // Equality
  price: { $gte: 50, $lte: 200 }      // Range
}).sort({ createdAt: -1 })             // Sort

// Optimal index following ESR rule
db.orders.createIndex({ userId: 1, createdAt: -1, price: 1 })
//                      Equality   Sort         Range
```

長時間執行的查詢

概觀

長時間執行的查詢可能會層疊到整個叢集的效能問題。這些查詢可能會干擾 Amazon DocumentDB 的多版本並行控制 (MVCC) 垃圾收集程序，導致舊版本的文件累積，並降低叢集的效能。Amazon DocumentDB 實作 2 小時伺服器端逾時做為安全機制，以限制失控查詢無限期地耗用資源。

什麼是長時間執行的查詢：

- 長時間執行的查詢 （通常 > 30 分鐘）
- 開啟處於作用中狀態數小時的游標 （如果游標處於作用中狀態，則 2 小時逾時不適用）

對叢集的影響

長時間執行的查詢可能會干擾垃圾回收程序

- 舊版本累積：垃圾收集器無法回收舊文件版本
- 集合和索引膨脹：集合和索引項目會隨著時間累積，膨脹增加，這可能會導致更多的儲存成本。
- CPU 和記憶體壓力：由於舊文件版本、索引項目和交易 IDs 數量增加的處理效率不佳，CPU 和記憶體壓力會增加。

長時間執行的查詢 → 封鎖 GC → 儲存成長 → CPU 和記憶體壓力 → 更長的查詢

監控和偵測

1。若要尋找長時間執行的查詢，請使用 currentOp 命令。

```
// To find a query running for more than 30 mins
db.adminCommand({
    aggregate: 1,
    pipeline: [
        {$currentOp: {}},
        {$match:
            {$or:
                [{secs_running: {$gt: 1800}},
                 {WaitState: {$exists: true}}]}}],
    cursor: {}
});
```

2. 尋找作用中超過 30 分鐘的游標

```
// To find cursor which is running more than 30 mins
db.adminCommand({
    "currentOp": true,
    "active": true,
    "$all": true
}).inprog.filter(function(op) {
    return op.desc == "Cursor" &&
           op.secs_running > 1800 &&
           op.active == true;
}).sort((a, b) => b.microsecs_running - a.microsecs_running)
```

3. 透過 CloudWatch 監控垃圾收集器進度

`LongestRunningGCProcess`— 最長作用中垃圾回收程序的持續時間，以秒為單位。每分鐘更新一次，並僅追蹤作用中的操作，不包括在一分鐘時段內完成的程序。

`AvailableMVCCIds` - 顯示達到零之前可用剩餘寫入操作數量的計數器。當此計數器達到零時，您的叢集會進入唯讀模式IDs直到 ID 回收和回收為止。計數器會隨著每次寫入操作而減少，並隨著垃圾回收回收舊的 MVCC IDs 而增加。

> **Note**
>
> 較低的 MVCC IDs 和延長的垃圾收集期間並非僅歸因於長時間執行的查詢。資源受限執行個體上的寫入密集型工作負載也可能導致 MVCC ID 可用性降低和垃圾回收週期延長。

修復策略

- 在應用程式中實作查詢逾時
- 請勿讓游標存活更長的時間
- 最佳化查詢以獲得更好的效能。
- 偏好批次寫入操作

低使用率和備援索引

概觀

索引可改善工作負載的整體查詢效能，但會帶來成本。每個索引都會耗用儲存體和 I/O 資源，而寫入操作需要更新集合上的所有索引。如果您的資料庫包含未使用、未充分利用或備援的索引，會導致叢集的額外負荷，進而降低效能。多個不必要的索引會建立反模式，造成效能問題、增加儲存成本和更高的操作開銷。

次佳索引案例

- 未使用的索引：為一次性查詢建立的索引，或在目前查詢模式不再存取的先前產品反覆運算中建立的索引。若要識別集合上未使用的索引，您可以使用 indexStats。如需詳細資訊，請參閱the section called “分析索引用量並識別未使用的索引”。
- 備援索引：涵蓋重疊金鑰模式或相同查詢模式的多個索引。範例：當複合索引同時涵蓋屬性 A 和 B 時，屬性 A 上的單一索引鍵會變成備援，因為複合索引可以單獨處理屬性 A 上的查詢。
- 過度索引：在不分析實際查詢模式或效能需求的情況下建立索引「以防萬一」。
- 低基數索引：在具有幾個不同值 （例如布林值欄位、狀態旗標） 的欄位上索引，可提供最小的查詢最佳化效益。

對叢集效能的影響

- 儲存和 IO：每個索引都會耗用儲存空間和 IO 資源，這有助於整體成本。如果索引未使用或備援，則表示叢集上不必要的成本。
- 降低的寫入效能：插入、更新和刪除操作必須維護索引，建立消耗資源的額外負荷。未使用的或備援索引會為這些操作增加不必要的額外負荷。
- 記憶體和 CPU 壓力：索引會競爭緩衝集區記憶體，如果工作集不適合執行個體記憶體，導致 BufferCacheHitRatio 欠佳，則可能會移出經常存取的資料。系統也會使用維護索引結構的 CPU 資源。

可用於識別索引最佳化機會的工具

- 索引檢閱工具：
- 索引基數偵測工具

收集工具組：

```
git clone https://github.com/awslabs/amazon-documentdb-tools.git
```

1。索引檢閱工具

索引檢閱工具會分析所有集合和索引，以提供索引清單及其使用模式，並識別 Amazon DocumentDB 叢集上的備援索引。在所有叢集執行個體上執行此工具，以進行全面分析。如需詳細資訊，請參閱索引檢閱工具。

用量：

```
cd performance/index-review/

python3 index-review.py --server-alias <server-alias> --uri <mongodb-uri>
```

輸出檔案

- {server-alias}-collections.csv：集合層級統計資料
- {server-alias}-indexes.csv：詳細的索引用量指標
- {server-alias}-{timestamp}-index-review.json：用於進一步分析的原始資料

> **Note**
>
> 使用獨立別名為叢集中的每個執行個體執行 ，以從每個執行個體產生統計資料，然後執行 工具以取得累積分析

例如：`python3 index-review.py --files "{server-alias}-{timestamp}-index-review.json, {server-alias}-{timestamp}-index-review.json" --server-alias full-review`

2. 索引基數偵測工具

此工具可識別基數低且查詢效能效率不佳的索引。

用途

```
cd performance/index-cardinality-detection/

python3 detect-cardinality.py --uri <mongodb-uri>
```

參數

- `--threshold`：基數閾值百分比（預設值：1%)
- `--sample-count`：要取樣的文件數量（預設值：100，000)
- `--max-collections`：每個資料庫要掃描的集合上限（預設值：100)

如需詳細資訊，請參閱 https：//https://aws.amazon.com/blogs/database/detect-and-fix-low-cardinality-indexes-in-amazon-documentdb/

修復策略

1。未使用和備援索引移除

```
// Example: Drop unused index
db.collection.dropIndex("unused_index_name")
```

> **Note**
>
> 在未與所有感興趣的各方討論並測試效能之前，絕不應捨棄索引。

2. 低基數索引最佳化

- 搭配篩選條件使用部分索引
- 將單一低基數索引轉換為複合索引

具有大型陣列的多重索引鍵

概觀

Amazon DocumentDB 中的多金鑰索引可讓您有效率地查詢陣列欄位。此索引策略可在搜尋陣列資料時大幅降低查詢延遲，使其成為具有複雜資料結構之應用程式具吸引力的選項。

不過，在集合中使用大型陣列或多個陣列欄位時，請務必了解資源影響和效能特性，以做出明智的架構決策。當您在陣列欄位上建立索引時，Amazon DocumentDB 會為陣列中的每個元素產生個別索引項目。

對叢集的影響

- 儲存和 IO 額外負荷：多金鑰索引有時可能會耗用最多多次的基礎資料表儲存空間，大小與具有索引陣列屬性的集合中的文件、陣列中的元素數量和元素的大小直接成正比。
- 記憶體用量：由於儲存體使用量相對較大，這些索引在記憶體中有對應的使用量，因此產生較大的工作集。
- 寫入操作：多金鑰索引會在寫入操作期間建立額外負荷。每個陣列元素都會產生個別的索引項目，將完成寫入操作所需的工作相乘。

修復策略

- 如果需要，請建立多金鑰索引，並限制陣列中的欄位數量。
- 限制集合上的多金鑰索引數量
- 考慮修改您的資料模型

效能改善秘訣

本節為 Amazon DocumentDB 提供五種效能最佳化策略，以改善應用程式效率和查詢執行。

1. 使用 $match 作為彙總管道中的第一階段

一律將 $match 作為在彙總管道中篩選的第一個階段，以最大化效能。當 $match 領導管道時，Amazon DocumentDB 將有效利用索引，讓資料庫能夠及早篩選資料並減少處理開銷。

```
// Optimized approach
```

```
db.orders.aggregate([
  { $match: { status: "active", category: "electronics" } }, // Index utilization
  { $group: { _id: "$category", total: { $sum: "$price" } } },
  { $sort: { total: -1 } }
])
```

影響：早期篩選可減少後續管道階段中處理的文件數量，進而加快查詢執行速度並降低資源消耗。

2. 在彙總管道中使用 $project 將管道資料大小降至最低

僅透過彙總管道階段攜帶必要欄位，以將資料大小降至最低並改善效能。以策略方式使用 $project，只包含您需要的資料。

```
// Efficient pipeline design
db.orders.aggregate([
  { $match: { orderDate: { $gte: new Date("2024-01-01") } } },
  { $project: { customerId: 1, totalAmount: 1, status: 1 } }, // Only needed fields
  { $group: { _id: "$customerId", totalSpent: { $sum: "$totalAmount" } } }
])
```

影響：較小的文件可減少記憶體用量並改善管道處理效率，進而提升整體查詢效能。

3. 為更低的儲存成本、I/O 成本和改善的查詢效能啟用文件壓縮

從叢集參數群組啟用文件壓縮，以降低儲存成本、I/O 成本並提升查詢效能。Amazon DocumentDB 會將壓縮文件存放在磁碟和 RAM 中，以減少記憶體佔用量和 I/O 成本。

Impact: (影響：)

- 更多文件適用於可用的記憶體
- 降低磁碟讀取速度，加快資料存取速度
- 降低儲存成本、I/O 成本並改善查詢效能

> **Note**
>
> Amazon DocumentDB 預設不會為 5.0 版啟用壓縮。您可以在 5.0 叢集的集合或叢集層級啟用壓縮。使用 Amazon DocumentDB 的壓縮檢閱公用程式來分析集合的壓縮比率。

對於 Amazon DocumentDB 8.0，預設會啟用壓縮。

4. 利用索引獲得最佳查詢效能

確保您的查詢一律使用索引來獲得最佳效能。Amazon DocumentDB 提供多種索引類型，以符合不同的使用案例。

索引原則：

- 每個查詢都應利用適當的索引
- Amazon DocumentDB 提供多種索引類型
- 複合索引支援具有單一索引的各種查詢形狀，可提供最大的彈性
- 設計索引以同時支援排序和篩選操作

了解索引字首：複合索引適用於索引字首 - Amazon DocumentDB 可以使用索引欄位的任何left-to-right子集。例如，索引會`{ category: 1, price: -1, inStock: 1 }`建立這些可用的字首：

- `{ category: 1 }` - 僅支援依類別篩選的查詢
- `{ category: 1, price: -1 }` - 支援依類別篩選查詢，以及依價格排序/篩選
- `{ category: 1, price: -1, inStock: 1 }` - 支援完整的複合查詢

僅查詢價格、inStock 或 inStock 將不會使用此索引，因為它們不是以第一個欄位（類別）開頭。

如何識別未使用索引的查詢：使用 explain() 方法來分析查詢執行，並識別執行集合掃描的查詢，而不是使用索引。

影響：沒有索引使用率的查詢會導致集合掃描，導致執行個體的記憶體和 CPU 壓力增加，以及查詢延遲增加。

5. 根據查詢模式最佳化資料模型

讓您的資料模型與您的應用程式查詢和更新資料的方式保持一致。資料建模是高效能 Amazon DocumentDB 應用程式的基礎。

最佳化策略：

效能內嵌

- 當相關資料經常以單位形式存取時，請將相關資料存放在一起
- 一律一起擷取的內嵌文件
- 適合one-to-few的關係

```
// Embedded approach for frequently accessed data
{
  _id: ObjectId("..."),
  customerName: "John Doe",
  address: {
    street: "123 Main St",
    city: "Seattle",
    zipCode: "98101"
  },
  recentOrders: [
    { orderId: "ORD001", amount: 99.99, date: "2024-01-15" }
  ]
}
```

彈性參考

- 使用大型或不常存取資料的參考
- 建議與大型資料集建立one-to-many關係
- 防止文件膨脹並改善更新效能

集合分割策略

當大型文件中只有幾個欄位經常更新時，或當大型不常存取的資料膨脹文件時，請考慮分割集合：

- 在個別、較小的集合中保留經常更新的欄位
- 在另一個集合中存放靜態或不常存取的資料
- 視需要將它們與參考連結

```
// Before: Large document with mixed access patterns
{
  _id: ObjectId("..."),
```

```
  productId: "PROD123",
  name: "Wireless Headphones",        // Frequently accessed
  price: 99.99,                       // Frequently accessed
  inventory: 45,                      // Updated frequently
  lastSold: "2024-01-15",           // Updated frequently
  detailedSpecs: { /* large object */ }, // Infrequently accessed
  manualPDF: "base64...",           // Large, rarely accessed
  reviewHistory: [/* large array */] // Infrequently accessed
}

// After: Split into collections based on access patterns
// products collection (frequently accessed data)
{
  _id: ObjectId("..."),
  productId: "PROD123",
  name: "Wireless Headphones",
  price: 99.99,
  inventory: 45,
  lastSold: "2024-01-15"
}

// product_details collection (infrequently accessed data)
{
  _id: ObjectId("..."),
  productId: "PROD123",           // Reference to products collection
  detailedSpecs: { /* large object */ },
  manualPDF: "base64...",
  reviewHistory: [/* large array */]
}
```

效能提升：文件越小表示更新速度越快、記憶體用量越低，快取效率也越好。

影響：低效率的資料建模會導致查詢不佳、文件大小增加和記憶體用量增加，進而降低應用程式效能並提高營運成本。

查詢計畫分析

透過解釋計畫進行查詢計畫分析，可提供 Amazon DocumentDB 查詢效能的基本洞見。具有 executionStats 的查詢計劃會顯示關鍵指標，包括：

- 每個階段傳回的文件 (nReturned)

- 階段特定的執行時間 (executionTimeMillisEstimate)
- 計畫產生持續時間 (planningTimeMillis)

透過檢查查詢計畫輸出，開發人員可以分析執行模式、評估索引使用率，並識別查詢管道階段的潛在最佳化機會。

若要分析查詢計劃，您可以使用下列格式的 explain() 命令。

```
db.runCommand({explain: {query document}, verbosity: "executionStats"})
db.collection.find().explain("executionStats");
```

以下是操作範例：

```
db.collection.find({ companyname: { '$eq': 'ANYCOMPANY' }, isDeleted: { '$eq':
 false } }).sort({"createdAt":1}).limit(2).explain("executionStats");
```

此操作的輸出如下所示：

```
{
  queryPlanner: {
    plannerVersion: 2,
    namespace: 'limit_test.test',
    winningPlan: {
      stage: 'LIMIT_SKIP',
      inputStage: {
        stage: 'SORT',
        sortPattern: { createdAt: 1 },
        inputStage: {
          stage: 'IXSCAN',
          indexName: 'companyname_1_createdAt_1_isDeleted_1',
          direction: 'forward',
          indexCond: {
            '$and': [
              { companyname: { '$eq': 'ANYCOMPANY' } },
              { isDeleted: { '$eq': false } }
            ]
```

```
          }
        }
      }
    }
  },
  indexFilterSet: false,
  indexFilterApplied: false,
  executionStats: {
    executionSuccess: true,
    executionTimeMillis: '4.186',
    planningTimeMillis: '3.909',
    executionStages: {
      stage: 'LIMIT_SKIP',
      nReturned: '2',
      executionTimeMillisEstimate: '0.199',
      inputStage: {
        stage: 'SORT',
        nReturned: '2',
        executionTimeMillisEstimate: '0.197',
        sortPattern: { createdAt: 1 },
        inputStage: {
          stage: 'IXSCAN',
          nReturned: '34',
          executionTimeMillisEstimate: '0.151',
          indexName: 'companyname_1_createdAt_1_isDeleted_1',
          direction: 'forward',
          indexCond: {
            '$and': [
              { companyname: { '$eq': 'ANYCOMPANY' } },
              { isDeleted: { '$eq': false } }
            ]
          }
        }
      }
    }
  },
  serverInfo: { host: 'demo-cluster', port: 27017, version: '5.0.0' },
  ok: 1,
  operationTime: Timestamp({ t: 1759915116, i: 1 })
}
```

以下是 Amazon DocumentDB 查詢執行計劃的詳細分析，細分每個元件及其效能特性。

整體計時

executionTimeMillis 代表查詢所花費的總時間，包括規劃時間。

planningTimeMillis 代表查詢所花費的總規劃時間。

執行階段

它描述了 Amazon DocumentDB 用來執行查詢step-by-step程序，顯示資料如何流經不同的操作。

```
"executionStages": {
    "stage": "[STAGE_NAME]",
    "nReturned": "[NUMBER_OF_DOCS]",
    "executionTimeMillisEstimate": "[TIME]",
    "inputStage": {
        // Nested stages
    }
}
```

查詢計畫中的常見階段

以下是查詢計劃中常見的執行階段。每個階段都會傳回 executionTimeMillisEstimate （執行時間） 和 nReturned （文件數量） 指標，以協助評估每個階段的查詢效能。

COLLSCAN （集合掃描）

- 依文件掃描整個集合文件
- nReturned：傳回集合中的所有相符文件
- 可以是昂貴的操作，當沒有索引可用時可見

IXSCAN （索引掃描）

- 使用索引尋找相符的文件
- nReturned：僅根據索引比對文件
- 高效率操作，優於 COLLSCAN

SORT

- 根據指定的欄位排序文件
- 如果查詢中的排序屬性不是索引的一部分，則此階段將明確顯示
- nReturned：與輸入文件相同的數字
- 大型結果集的記憶體密集型。若要最佳化，請將排序欄位名稱新增至索引

LIMIT_SKIP

- 控制傳回和略過的文件數量
- nReturned：受 LIMIT 值限制
- 用於分頁或 LIMIT 操作

子掃描

- 執行巢狀查詢操作
- nReturned：根據子查詢結果而有所不同
- 用於具有多個階段的複雜查詢

具有高executionTimeMillisEstimate 是最佳化的理想候選者。

> **Note**
>
> executionStats 參數目前不支援更新和刪除命令。

了解說明計畫執行統計資料中的 docsExamined

當您使用 執行查詢時`explain("executionStats")`，Amazon DocumentDB 會提供檢查指標，協助您了解已掃描多少文件來產生查詢結果。這些指標有助於識別效率不佳的查詢計劃和最佳化效能。

> **Note**
>
> 僅適用於 Amazon DocumentDB 5.0.1+ 和 8.0.0+。

欄位

欄位	說明	Level
totalDocsExamined	在所有執行階段中檢查的文件總數。	頂層 executionStats
docsExamined	特定執行階段檢查的文件數量。	階段層級

`docsExamined` 欄位會出現在下列階段：

階段	說明
COLLSCAN	集合掃描。會檢查集合中的所有文件。
IXSCAN	索引掃描。只會檢查符合索引條件的文件。
FETCH	當最佳化工具從 IXSCAN 擷取不同階段的文件時，FETCH 階段會報告 docsExamined。在索引掃描計劃中，子 IXSCAN 階段不會報告 docsExamined。在 $lookup 計劃中，FETCH 階段及其子階段都可以報告 docsExamined。

> **Note**
>
> `docsExamined` 不會出現在 IXONLYSCAN 階段上，因為查詢完全從索引滿足而不存取文件。

在 Explain Plan executionStats Output 中顯示 docsExamined

下列範例使用具有 500，000 個文件的集合，並示範在不同查詢執行階段之間如何`docsExamined`變更。

建立範例資料：

```
for (let i = 0; i < 500000; i++) {
    db.coll.insertOne({ number: i, arr: [i, [i+1]], value: "test", bool: i % 2 ===
 0 });
```

```
}
```

集合掃描 (COLLSCAN)

如果沒有索引，Amazon DocumentDB 會執行完整的集合掃描。

```
db.coll.find({ "number": { "$lt": 500 } }).explain("executionStats").executionStats
```

輸出：

```
{
    "executionSuccess" : true,
    "nReturned" : "500",
    "executionTimeMillis" : "282.055",
    "planningTimeMillis" : "0.085",
    "totalDocsExamined" : "500000",
    "executionStages" : {
        "stage" : "COLLSCAN",
        "nReturned" : "500",
        "executionTimeMillisEstimate" : "281.915",
        "docsExamined" : "500000",
        "filter" : {
            "number" : {
                "$lt" : 500
            }
        }
    }
}
```

已檢查所有 500，000 個文件，以傳回 500 個結果。在數字欄位上建立索引可改善此狀況。

索引掃描 (IXSCAN)

```
db.coll.createIndex({ number: 1 });
db.coll.find({ "number": { "$lt": 5000 } }).explain("executionStats").executionStats
```

輸出：

```
{
    "executionSuccess" : true,
    "nReturned" : "5000",
    "executionTimeMillis" : "3.047",
    "planningTimeMillis" : "0.296",
    "totalDocsExamined" : "5000",
    "executionStages" : {
        "stage" : "IXSCAN",
        "nReturned" : "5000",
        "executionTimeMillisEstimate" : "2.576",
        "indexName" : "number_1",
        "direction" : "forward",
        "docsExamined" : "5000",
        "indexCond" : {
            "$and" : [
                {
                    "number" : {
                        "$lt" : 5000
                    }
                }
            ]
        }
    }
}
```

使用 索引時，只會檢查和擷取 500，000 個文件中的 5，000 個，符合傳回的數目。索引條件篩選掉 495，000 個不符合查詢的文件。

含剩餘篩選條件的索引掃描

```
db.coll.find({ "number": { "$lt": 5000 }, "arr": { "$gt":
 4000 } }).explain("executionStats").executionStats
```

輸出：

```
{
```

```
    "executionSuccess" : true,
    "nReturned" : "999",
    "executionTimeMillis" : "15.367",
    "planningTimeMillis" : "0.115",
    "totalDocsExamined" : "5000",
    "executionStages" : {
        "stage" : "IXSCAN",
        "nReturned" : "999",
        "executionTimeMillisEstimate" : "15.170",
        "indexName" : "number_1",
        "direction" : "forward",
        "docsExamined" : "5000",
        "indexCond" : {
            "$and" : [
                {
                    "number" : {
                        "$lt" : 5000
                    }
                }
            ]
        },
        "filter" : {
            "arr" : {
                "$gt" : 4000
            }
        }
    }
}
```

索引條件符合 500，000 個文件中的 5，000 個文件，然後 上的剩餘篩選條件將結果arr減少為 999。docsExamined 值 5，000 反映在索引條件之後，但在套用剩餘篩選條件之前檢查的所有文件。

使用 IXSCAN 擷取

```
db.coll.find({ "$or": [{ "number": { "$lt": 100000 } }, { "number": { "$gt":
 400000 } }] }).explain("executionStats").executionStats
```

輸出：

```
{
    "executionSuccess" : true,
    "nReturned" : "199999",
    "executionTimeMillis" : "899.801",
    "planningTimeMillis" : "0.183",
    "totalDocsExamined" : "199999",
    "executionStages" : {
        "stage" : "FETCH",
        "nReturned" : "199999",
        "executionTimeMillisEstimate" : "894.141",
        "docsExamined" : "199999",
        "inputStage" : {
            "stage" : "IXOR",
            "nReturned" : "0",
            "executionTimeMillisEstimate" : "874.897",
            "inputStages" : [
                {
                    "stage" : "IXSCAN",
                    "nReturned" : "100000",
                    "executionTimeMillisEstimate" : "462.208",
                    "indexName" : "number_1",
                    "indexCond" : {
                        "$and" : [
                            {
                                "number" : {
                                    "$lt" : 100000
                                }
                            }
                        ]
                    }
                },
                {
                    "stage" : "IXSCAN",
                    "nReturned" : "99999",
                    "executionTimeMillisEstimate" : "412.684",
                    "indexName" : "number_1",
                    "indexCond" : {
                        "$and" : [
                            {
                                "number" : {
                                    "$gt" : 400000
                                }
                            }
                        ]
```

```
                    }
                }
            ]
        }
    }
}
```

當 Amazon DocumentDB 最佳化工具使用擷取階段擷取文件時，FETCH 階段會報告 docsExamined。子 IXSCAN 階段不會報告，docsExamined因為它們只會掃描索引鍵而不直接存取文件。

FETCH 搭配彙總查詢

```
db.coll.explain("executionStats").aggregate([
    { $match: { "number": { "$lt": 5 } } },
    { $lookup: { from: "coll", pipeline: [{ $match: { "number": { "$lt": 3 } } }], as:
 "sub" } }
]).executionStats
```

輸出：

```
{
    "executionSuccess" : true,
    "nReturned" : "5",
    "executionTimeMillis" : "0.525",
    "planningTimeMillis" : "0.327",
    "totalDocsExamined" : "9",
    "executionStages" : {
        "stage" : "NESTED_LOOP_LOOKUP",
        "nReturned" : "5",
        "executionTimeMillisEstimate" : "0.163",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                "nReturned" : "5",
                "executionTimeMillisEstimate" : "0.039",
                "indexName" : "number_1",
                "direction" : "forward",
                "docsExamined" : "5",
```

```
                "indexCond" : {
                    "$and" : [
                        {
                            "number" : {
                                "$lt" : 5
                            }
                        }
                    ]
                }
            },
            {
                "stage" : "FETCH",
                "nReturned" : "1",
                "executionTimeMillisEstimate" : "0.009",
                "docsExamined" : "1",
                "inputStage" : {
                    "stage" : "AGGREGATE",
                    "nReturned" : "1",
                    "executionTimeMillisEstimate" : "0.044",
                    "inputStage" : {
                        "stage" : "IXSCAN",
                        "nReturned" : "3",
                        "executionTimeMillisEstimate" : "0.013",
                        "indexName" : "number_1",
                        "direction" : "forward",
                        "docsExamined" : "3",
                        "indexCond" : {
                            "$and" : [
                                {
                                    "number" : {
                                        "$lt" : 3
                                    }
                                }
                            ]
                        }
                    }
                }
            }
        ]
    }
}
```

外部 IXSCAN 檢查了 5 個符合編號 < 5 的文件。內部 IXSCAN 檢查了 3 個符合編號 < 3 的文件，FETCH 階段檢查了 1 個彙總結果。9 `totalDocsExamined`的 是所有階段的總和 (5 + 3 + 1)。

Amazon DocumentDB 的最佳實務

了解使用 Amazon DocumentDB （與 MongoDB 相容） 的最佳實務。在新最佳實務確定時，會不斷更新此小節。

主題

- 基本操作準則
- 執行個體大小
- 使用索引
- 安全最佳實務
- 成本最佳化
- 使用指標來識別效能問題
- TTL 和時間序列工作負載
- 遷移
- 使用叢集參數群組
- 彙總管道查詢
- batchInsert 和 batchUpdate

基本操作準則

以下是每個人在使用 Amazon DocumentDB 時應遵循的基本操作準則。Amazon DocumentDB 服務水準協議要求您遵循這些準則。

- 在兩個可用區域中部署由兩個 AWS 或多個 Amazon DocumentDB 執行個體組成的叢集。對於生產工作負載，在三個可用區域中部署由三個或多個 Amazon DocumentDB 執行個體組成的叢集。
- 在指定的服務限制內使用服務。如需詳細資訊，請參閱Amazon DocumentDB 配額。
- 監控您的記憶體、CPU、連線和儲存體用量。為了協助您維持系統效能和可用性，請設定 Amazon CloudWatch，以便在使用模式變更或接近部署容量時通知您。
- 當您接近容量限制時擴展您的執行個體。您的執行個體應該佈建足夠的運算資源 (例如 RAM、CPU)，以因應應用程式不可預見的需求增加。

- 設定備份保留期以符合您的復原點目標。
- 測試叢集的容錯移轉，以了解您的使用案例執行此程序需時多長。如需詳細資訊，請參閱Amazon DocumentDB 容錯移轉。
- 使用叢集端點（請參閱 Amazon DocumentDB 端點) 和複本集模式（請參閱 以複本集的形式連線至 Amazon DocumentDB) 連線至 Amazon DocumentDB 叢集，將容錯移轉對應用程式的影響降至最低。
- 選擇驅動程式讀取偏好設定，以發揮最大讀取擴展，同時符合您應用程式的讀取一致性要求。`secondaryPreferred` 讀取偏好設定會啟用複本讀取，並釋出主要執行個體以進行更多工作。如需詳細資訊，請參閱讀取偏好設定選項。
- 設計您的應用程式，以便在網路和資料庫發生錯誤時，保有故障恢復的應變能力。使用您驅動程式的錯誤機制，區分暫時性錯誤或永久性錯誤。適當時，使用指數退避機制重新嘗試暫時性錯誤。確認您的應用程式在實作重試邏輯時會考慮資料一致性。
- 為所有生產叢集或任何具有重要資料的叢集啟用叢集刪除保護。刪除 Amazon DocumentDB 叢集之前，請拍攝最終快照。如果您要使用 部署資源 CloudFormation，請啟用終止保護。如需詳細資訊，請參閱終止保護和刪除保護。
- 建立 Amazon DocumentDB 叢集時， `--engine-version`是選用參數，預設為最新的主要引擎版本。目前的預設引擎版本為 5.0.0（注意：Amazon DocumentDB 8.0 可供使用，但必須明確指定）。發行新的主要引擎版本時， 的預設引擎版本`--engine-version`將會更新，以反映最後一個主要引擎版本。因此，對於生產工作負載，特別是依賴指令碼、自動化或 CloudFormation 範本的工作負載，請將 明確指定`--engine-version`為預期的主要版本。

執行個體大小

在 Amazon DocumentDB 中選擇執行個體大小的其中一個最重要方面是快取的 RAM 數量。Amazon DocumentDB 會為自己的服務保留三分之一的 RAM，這表示只有三分之二的執行個體 RAM 可供快取使用。因此，Amazon DocumentDB 最佳實務是選擇具有足夠 RAM 的執行個體類型，以符合記憶體中的工作集（即資料和索引）。具有適當大小的執行個體將有助於最佳化整體效能，並可能將 I/O 成本降至最低。您可以使用第三方 Amazon DocumentDB 大小計算器來估計特定工作負載的執行個體大小。

若要判斷應用程式的工作集是否適用記憶體，請使用 Amazon CloudWatch 針對叢集中負載下的每個執行個體監控 `BufferCacheHitRatio`。

`BufferCacheHitRatio` CloudWatch 指標會測量從執行個體的記憶體快取 (與儲存磁碟區) 提供的資料和索引百分比。一般來說，`BufferCacheHitRatio` 的數值應該盡可能高，因為從工作集記憶體讀

取資料比從儲存磁碟區讀取更快速且更具成本效益。儘管盡可能保持 BufferCacheHitRatio 接近 100% 是理想的做法，但是最可能達到的數值將取決於您的應用程式的存取模式和效能需求。為了盡可能保持最高的 BufferCacheHitRatio，建議您叢集中的執行個體佈建足夠的 RAM，以便在記憶體中容納索引和工作資料集。

如果您的索引不適用記憶體，您會看到一個較低的 BufferCacheHitRatio。從磁碟持續讀取會產生額外的 I/O 成本，而且不優於從記憶體讀取。如果您的 BufferCacheHitRatio 比例低於預期，請為您的叢集擴充執行個體大小，以提供更多 RAM 以符合記憶體中的工作集資料。如果擴大執行個體類別會導致 BufferCacheHitRatio 大幅增加，則應用程式的工作集不適用記憶體。持續擴展直到擴展操作後 BufferCacheHitRatio 不再大幅增加。如需監控執行個體指標的相關資訊，請參閱Amazon DocumentDB 指標。

根據您的工作負載和延遲需求，您的應用程式在穩定狀態使用期間具有較高的 BufferCacheHitRatio 值，而在執行個體上執行需要掃描整個集合的分析查詢時，BufferCacheHitRatio 時而下降，是可接受的情況。BufferCacheHitRatio 中這些定期的下降對後續的查詢可能會顯示為較高的延遲，因為需要將工作集資料從儲存磁碟區重新填入緩衝區快取。先使用代表性的生產工作負載在生產前環境中測試工作負載，以了解效能特性，**BufferCacheHitRatio**再將工作負載部署到生產環境。

BufferCacheHitRatio 是執行個體特定的指標，因此相同叢集中的不同執行個體可能會有不同的 BufferCacheHitRatio 值，視讀取作業在主要和複本執行個體之間的分配情況而定。如果您的操作工作負載無法處理在執行分析查詢後重新植入工作集快取時而增加的延遲，您應該嘗試將一般工作負載的緩衝區快取與分析查詢隔離。您可以將操作查詢導向至主要執行個體，而只將分析查詢導向至複本執行個體，以達到完全 BufferCacheHitRatio 隔離。您也可以透過將分析查詢導向至特定複本執行個體，在了解某些百分比的一般查詢也會在該複本上執行，且可能受到影響的情況下，達成部分隔離。

適當的 BufferCacheHitRatio 值取決於您的使用案例和應用程式需求。此指標沒有最佳或最小值；只有您自己可以決定從成本和效能的角度來看，是否可以接受暫時較低 BufferCacheHitRatio 的折衷。

使用索引

建置索引

將資料匯入 Amazon DocumentDB 時，您應該先建立索引，再匯入大型資料集。您可以使用 Amazon DocumentDB Index Tool 從執行中的 MongoDB 執行個體或mongodump目錄中擷取索引，並在 Amazon DocumentDB 叢集中建立這些索引。如需遷移的詳細指導方針，請參閱遷移和升級 Amazon DocumentDB。

索引選擇性

將索引建立限制為欄位，其中重複值的數量小於集合中文件總數的 1%。例如，如果您的集合包含 100,000 份文件，則只在相同值出現 1000 次或更少的欄位上建立索引。

選擇具有大量唯一值 (即高基數) 的索引可確保篩選器作業傳回少量文件，從而在索引掃描期間產生良好的效能。高基數索引範例像是唯一索引，它可以保證相等述詞最多只能傳回單一文件。低基數索引的範例像是布林欄位，以及對一週中某天的索引。由於效能不佳，資料庫的查詢最佳化工具不太可能選擇低基數索引。同時，低基數索引會繼續消耗磁碟空間和 I/O 等資源。根據經驗法則，您應該在典型值頻率為總集合大小的 1% 或更小的字段上鎖定索引。

此外，建議只在通常用作篩選器的欄位上建立索引，並定期尋找未使用的索引。如需詳細資訊，請參閱如何分析索引用量並識別未使用的索引？。

索引對寫入資料的影響

雖然索引可以透過避免掃描集合中的每個文件來改善查詢性能，但此改進方式有所權衡。對於集合上的每個索引，每次插入、更新或刪除文件時，資料庫必須更新集合並將欄位寫入集合的每個索引。例如，如果集合有九個索引，資料庫必須先執行十次寫入，才能確認用戶端的作業。因此，每個額外的索引都會造成額外的寫入延遲、I/O 以及整體使用的儲存量增加。

叢集執行個體的大小必須適當，才能保留所有工作集記憶體。如此可避免從儲存磁碟區持續讀取索引頁面的需求，進而對效能造成負面影響，並產生較高的 I/O 成本。如需詳細資訊，請參閱執行個體大小。

為了獲得最佳效能，請盡量減少集合中的索引數目，只新增必要的索引，以改善常見查詢的效能。雖然工作負載有所不同，但好的指導方針是將每個集合的索引數目保持在五個或更少。

識別缺少的索引

最佳實務是定期識別缺少的索引。如需詳細資訊，請參閱如何識別缺少的索引？。

識別未使用的索引

最佳實務是定期識別和移除未使用的索引。如需詳細資訊，請參閱如何分析索引用量並識別未使用的索引？。

安全最佳實務

如需 Amazon DocumentDB 的安全最佳實務，請參閱 Amazon DocumentDB 的安全最佳實務。

成本最佳化

下列最佳實務可協助您管理和最小化使用 Amazon DocumentDB 時的成本。如需定價資訊，請參閱 Amazon DocumentDB （與 MongoDB 相容） 定價和 Amazon DocumentDB （與 MongoDB 相容） FAQs。

- 建立帳單提醒，閾值為您預期的當月帳單的 50% 和 75%。如需有關建立帳單提醒的詳細資訊，請參閱建立帳單警示。
- Amazon DocumentDB 的架構會區隔儲存和運算，因此即使是單一執行個體叢集也非常耐用。叢集儲存磁碟區在三個可用區域之間以六個方向複寫資料，無論叢集的執行個體數量為何，都能提供極高的耐用性。典型的生產叢集可以包含三種或多種執行個體，以提供高可用性。不過，如果不需要高可用性，則您可以使用單一執行個體開發叢集來使成本達到最佳化。
- 對於開發和測試案例，請停止不再需要的叢集，並且在恢復開發時啟動叢集。如需詳細資訊，請參閱停止和啟動 Amazon DocumentDB 叢集。
- TTL 和變更串流在資料寫入、讀取和刪除時都會產生 I/O。如果您已啟用這些功能，但未在應用程式中使用這些功能，停用這些功能有助於降低成本。

使用指標來識別效能問題

主題

- 檢視效能指標
- 設定 CloudWatch 警示
- 評估效能指標
- 使用 CloudWatch 指標評估 Amazon DocumentDB 執行個體用量
- 調校查詢

若要識別資源不足和其他常見瓶頸所造成的效能問題，您可以監控 Amazon DocumentDB 叢集可用的指標。

檢視效能指標

定期監控效能指標以查看各種時間範圍的平均值、最大值和最小值。效能降級時您便可以得知。您也可以針對特定指標閾值設定 Amazon CloudWatch 警示，以便在達到它們時收到提醒。

為了對效能問題進行故障診斷，請務必了解系統的基礎效能。您設定新的叢集並執行一般工作負載後，請針對不同間隔 (例如 1 小時、24 小時、1 週、2 週) 的所有效能指標擷取平均、最大值和最低值。這

可讓您了解正常運作情況為何。這有助於比較尖峰與離峰時段的操作。之後您可以使用此資訊來得知效能是否下降至標準層級之下。

您可以使用 AWS 管理主控台 或 檢視效能指標 AWS CLI。如需詳細資訊，請參閱檢視 CloudWatch 資料。

設定 CloudWatch 警示

若要設定 CloudWatch 警示，請參閱《Amazon CloudWatch 使用者指南》中的使用 Amazon CloudWatch 警示。 Amazon CloudWatch

評估效能指標

執行個體含有幾種不同的指標類別。您判定值是否為可接受，取決於指標。

CPU

- CPU 使用率 — 使用的電腦處理容量百分比。

記憶體

- 可用記憶體 — 執行個體上有多少 RAM 可用。
- 交換用量 — 執行個體使用多少交換空間，以 MB 為單位。

輸入/輸出操作

- 讀取 IOPS、寫入 IOPS — 每秒磁碟讀取或寫入操作的平均數量。
- 讀取延遲、寫入延遲 — 讀取或寫入操作的平均時間，以毫秒為單位。
- 讀取輸送量、寫入輸送量 — 每秒讀取或寫入磁碟的平均 MB 數。
- 磁碟佇列深度 — 等待寫入磁碟或從磁碟讀取的 I/O 操作數目。

網路流量

- 網路接收輸送量、網路傳輸輸送量 — 往返執行個體的網路流量速率，以每秒 MB 為單位。

資料庫連線

- 資料庫連線 — 連線至執行個體的用戶端工作階段數目。

一般來說，效能指標的可接受值視您的基準為何，以及您的應用程式所做之事而定。調查距離基準的一致或趨勢變異。

以下是關於特定類型指標的建議：

- 高 CPU 耗用量 — 高 CPU 耗用量值可能是適當的，前提是這些值符合您應用程式的目標（例如輸送量或並行），而且是預期的。如果您的 CPU 消耗量持續超過 80%，請考慮擴展您的執行個體。
- 高 RAM 耗用 — 如果您的`FreeableMemory`指標經常低於執行個體記憶體總量的 10%，請考慮擴展執行個體。如需 DocumentDB 執行個體發生高記憶體壓力時所發生情況的詳細資訊，請參閱 Amazon DocumentDB 資源控管。
- 交換用量 — 此指標應維持在零或接近零。如果您的交換用量相當龐大， 請考慮擴展您的執行個體。
- 網路流量 — 對於網路流量，請與您的系統管理員討論，以了解網域網路和網際網路連線的預期輸送量。調查網路流量的傳輸量是否如預期一致地降低。
- 資料庫連線 — 如果您看到大量使用者連線，並降低執行個體效能和回應時間，請考慮限制資料庫連線。執行個體使用者連接的最佳數量，將因執行個體類別和要執行的操作複雜性而不同。如有任何校能指標的問題，您可以執行以改善效能的其中一個動作是調校最常使用與最昂貴的查詢，以了解該調整是否降低對系統資源的壓力。

如果您的查詢經過調校且問題仍然存在，請考慮將您的 Amazon DocumentDB 執行個體類別升級至具有更多資源 (CPU、RAM、磁碟空間、網路頻寬、I/O 容量） 且與您遇到的問題相關的類別。

使用 CloudWatch 指標評估 Amazon DocumentDB 執行個體用量

您可以使用 CloudWatch 指標來監看執行個體輸送量，並探索執行個體類別是否為您的應用程式提供足夠的資源。如需執行個體類別配額的相關資訊，請參閱執行個體配額並尋找執行個體類別的規格，以尋找您的網路效能。

如果您的執行個體用量接近執行個體類別限制，則效能可能會開始變慢。CloudWatch 指標可以確認這種情況，因此您可以規劃手動縱向擴展到更大的執行個體類別。

結合下列 CloudWatch 指標值，以了解您是否接近執行個體類別限制：

- NetworkThroughput - 用戶端為 Amazon DocumentDB 叢集中的每個執行個體接收和傳輸的網路輸送量。此輸送量值不包含叢集中執行個體與叢集儲存磁碟區之間的網路流量。
- StorageNetworkThroughput - Amazon DocumentDB 叢集中每個執行個體接收並傳送至 Amazon DocumentDB 叢集儲存磁碟區的網路輸送量。

將 NetworkThroughput 新增至 StorageNetworkThroughput，以尋找 Amazon DocumentDB 叢集中每個執行個體從 Amazon DocumentDB 叢集儲存磁碟區接收並傳送至 Amazon DocumentDB 叢集儲存磁碟區的網路輸送量。執行個體的執行個體類別限制應該大於這兩個合併指標之和。

在傳送和接收時，您可以使用下列指標，來檢閱來自用戶端應用程式之網路流量的其他詳細資訊：

- NetworkReceiveThroughput - Amazon DocumentDB 叢集中每個執行個體從用戶端接收的網路輸送量。此輸送量不包含叢集中執行個體與叢集儲存磁碟區之間的網路流量。
- NetworkTransmitThroughput - Amazon DocumentDB 叢集中每個執行個體傳送給用戶端的網路輸送量。此輸送量不包含叢集中執行個體與叢集儲存磁碟區之間的網路流量。
- StorageNetworkReceiveThroughput - 叢集中每個執行個體從 Amazon DocumentDB 叢集儲存磁碟區接收的網路輸送量。
- StorageNetworkTransmitThroughput - 叢集中每個執行個體傳送至 Amazon DocumentDB 叢集儲存磁碟區的網路輸送量。

將所有這些指標加在一起，以評估您的網路用量與執行個體類別限制的比較情形。執行個體類別限制應該大於這些個合併指標之和。

執行個體的網路限制和 CPU 使用率是相互的。當網路輸送量增加時，CPU 使用率也會增加。監控 CPU 和網路用量可提供資源如何和為何耗盡的相關資訊。

若要協助將網路用量降至最低，您可以考慮：

- 使用更大的執行個體類別。
- 批次分割寫入要求，以減少整體交易。
- 將唯讀工作負載導向至唯讀執行個體。
- 刪除任何未使用的索引。

調校查詢

改善叢集執行個體效能的一個最佳方式是調校最常使用與資源最密集的查詢，讓執行成本較不昂貴。

您可以使用 Profiler (請參閱 分析 Amazon DocumentDB 操作) 來記錄叢集上執行之操作的執行時間和詳細資訊。Profiler 適用於監控叢集上最慢的操作，以協助您改善個別查詢效能和整體叢集效能。

您也可以使用 `explain` 命令了解如何分析特定查詢的查詢計劃。使用此資訊可修改查詢或基礎集合，藉此改善查詢效能 (例如，新增索引)。

TTL 和時間序列工作負載

TTL 指數到期後刪除文件是最勞心費神的過程。本公司不保證在任何特定期間內刪除文件。執行個體大小、執行個體資源使用率、文件大小、整體輸送量、索引數目以及索引和工作集是否適用記憶體等因素，都會影響 TTL 處理程序刪除過期文件的時間。

當 TTL 監控器刪除文件時，每個刪除都會產生 IO 成本，使帳單金額增加。如果輸送量和 TTL 刪除率提高，您可預期會因為 I/O 使用量增加而產生較高的費用。不過，如果您不建立 TTL 索引來刪除文件，而是根據時間將文件分割為集合，並在不再需要這些集合時直接捨棄，則不會產生任何 IO 成本。這比使用 TTL 索引更具成本效益。

對於時間序列工作負載，您可以考慮建立滾動集合而非 TTL 索引，因為滾動集合可能是刪除資料和降低 I/O 密集度的更好方法。如果您有大型集合（特別是超過 1TB 的集合）或 TTL 刪除 I/O 成本的問題，請根據時間將文件分割為集合，並在不再需要文件時捨棄集合。您可以資料擷取率為根據，每天或每個禮拜建立一個集合。雖然需求將根據您的應用程式而有所不同，但好的經驗法是擁有更小的集合而不是一些大型集合。刪除這些集合不會產生 IO 成本，且成本效益明顯比使用 TTL 索引還高。

遷移

將資料遷移至 Amazon DocumentDB 的最佳實務是先在 Amazon DocumentDB 中建立索引，然後遷移資料。首先建立索引可以縮短整體時間並提高遷移速度。若要這樣做，您可以使用 Amazon DocumentDB Index Tool。如需遷移的詳細資訊，請參閱 遷移和升級 Amazon DocumentDB。

我們也建議您在遷移生產資料庫之前，最佳實務是在 Amazon DocumentDB 上完整測試您的應用程式，並考量功能、效能、操作和成本。

使用叢集參數群組

在將變更套用至生產叢集之前，請先嘗試測試叢集上的叢集參數群組變更。如需備份叢集的詳細資訊，請參閱在 Amazon DocumentDB 中備份和還原。

彙總管道查詢

建立具有多個階段的彙總管道查詢，並僅評估查詢中的一部分資料時，請使用 `$match` 階段做為第一個階段或管道的開頭。使用 `$match` 會先減少彙總管道查詢中需要處理的文件後續階段數量，從而提高查詢的效能。

batchInsert 和 batchUpdate

在您的主要執行個體上執行高速並行 batchInsert 和/或 batchUpdate 操作，並且 FreeableMemory (CloudWatch 指標) 的數量降至零時，您可以減少批次插入或更新工作負載的並行性，或者若是無法減少工作負載的並行性，請增加執行個體大小以增加 FreeableMemory 的數量。

管理 Amazon DocumentDB 索引

主題

- 建立 Amazon DocumentDB 索引
- 維護 Amazon DocumentDB 索引

建立 Amazon DocumentDB 索引

在 Amazon DocumentDB 中建置索引需要做出許多決策：

- 需要多快完成？
- 當組建發生時，集合是否無法存取？
- 可以配置多少執行個體的運算能力給組建？
- 應該建立哪種類型的索引？

本節可協助您回答這些問題，並提供命令和監控範例，以在執行個體型叢集集合上建立 Amazon DocumentDB 索引。

指導方針

下列準則包括在建立新索引時的基本限制和組態權衡：

- Amazon DocumentDB 版本支援 - 雖然所有 Amazon DocumentDB 版本都支援單一工作者索引，但僅在 Amazon DocumentDB 4.0、5.0 和 8.0 版上支援多個工作者索引。
- 效能權衡 - 在索引建立程序中增加工作者數量會增加 CPU 使用率，並在 Amazon DocumentDB 資料庫的主要執行個體上讀取 IO。建立新索引所需的資源將不適用於您執行中的工作負載。
- 彈性叢集 - Amazon DocumentDB 彈性叢集不支援平行索引。
- 無伺服器叢集 - 無伺服器叢集上的平行索引需要至少 32 個 DCUs的最小容量。低於此閾值時，系統會接受 workers 參數，但索引建置會使用單一工作者執行。平行工作者的最大數量會隨著您設定的

最小容量而擴展：大約`min_DCU / 8`有工作者可用。例如，在 32 DCUs 中，您最多可以使用 4 個工作者，而在 64 DCUs 中，您最多可以使用 8 個工作者。

> **Note**
>
> 變更新的平行工作者限制以生效後，需要重新啟動叢集。

- 工作者上限（以執行個體為基礎）- 您可以設定的工作者數目上限取決於資料庫叢集中主要執行個體的大小。這是資料庫叢集主要執行個體上 vCPUs總數的一半。例如，您可以在具有 64 個 vCPUs 的 db.r6g.16xlarge 執行個體上執行最多 32 個工作者。

> **Note**
>
> 2xlarge 和更低的執行個體類別不支援平行工作者。

- 最低工作者 - 您可以設定的工作者數量下限為 1。在執行個體型叢集上建立索引的預設設定為兩個工作者。不過，您可以使用「工作者執行緒」選項，將工作者數量減少為一個。這將使用單一工作者執行程序。
- 索引壓縮 - Amazon DocumentDB 不支援索引壓縮。索引的資料大小可能大於您使用其他選項時的大小。
- 索引多個集合 - 資料庫叢集主要執行個體上的一半 vCPUs 可用於在多個集合上執行索引建立的設定工作者。
- 索引類型 - 如需 Amazon DocumentDB 上支援索引類型的完整說明，請參閱 AWS 資料庫部落格上的如何在 Amazon DocumentDB 上編製索引。

開始使用

若要開始在集合上建立索引，請使用 `createIndexes`命令。根據預設，命令會執行兩個平行工作者，將索引建立程序的速度增加兩次。

例如，下列命令程序示範如何為文件中的「user_name」欄位建立索引，並將索引程序速度提高到四名工作者：

1. 在叢集上使用兩個平行工作者建立索引：

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx"}]})
```

2. 若要最佳化索引建立程序的速度，您可以使用 db.runCommand createIndexes命令中的「工作者執行緒」選項 ("workers":<number>) 來指定工作者數量。

   將程序的速度提高到四個平行工作者：

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx", "workers":4}]})
```

> **Note**
>
> 工作者數量越高，建立索引的速度就越快。不過，工作者數量越高，主要執行個體vCPUs和讀取 IO 負載越高。確保您的叢集已充分佈建，以處理增加的負擔，而不會降低其他工作負載。

索引進度狀態

索引建立程序的運作方式是初始化、掃描集合、排序索引鍵，最後則是透過索引建置器插入索引鍵。當您在前景執行時，此程序最多有六個階段，而在背景執行時最多有九個階段。您可以依階段檢視狀態指標，例如完成百分比、掃描的儲存區塊總數、已排序的金鑰和插入的金鑰。

使用 mongo shell 中的 db.currentOp()命令來監控索引程序的進度。完成最後一個階段的 100% 顯示已成功建立所有索引：

```
db.currentOp({"command.createIndexes": { $exists : true } })
```

> **Note**
>
> 只有 Amazon DocumentDB 5.0 才支援檢視索引進度狀態。

索引建置類型

索引建置的四種類型為：

- 前景 — 前景索引建置會封鎖所有其他資料庫操作，直到建立索引為止。Amazon DocumentDB 前景建置由五個階段組成。

- 前景（唯一）— 單一文件（唯一）前景索引建置會封鎖其他資料庫操作，例如一般前景建置。與基本前景建置不同，唯一建置會使用額外的階段（排序金鑰 2) 來尋找重複的金鑰。前景（唯一）建置由六個階段組成。
- 背景：背景索引建置允許在建立索引時，在前景中執行其他資料庫操作。Amazon DocumentDB 背景建置由八個階段組成。
- 背景（唯一）— 單一文件（唯一）背景索引建置可讓其他資料庫操作在建立索引時於前景執行。與基本背景建置不同，唯一建置會使用額外的階段（排序金鑰 2) 來尋找重複的金鑰。背景（唯一）建置由九個階段組成。

索引建置階段

階段	前景	前景（唯一）	背景介紹	背景（唯一）
初始化	1	1	1	1
建置索引：初始化	2	2	2	2
建置索引：掃描集合	3	3	3	3
建置索引：排序索引鍵 1	4	4	4	4
建置索引：排序索引鍵 2		5		5
建置索引：插入索引鍵	5	6	5	6
驗證：掃描索引			6	7
驗證：排序元組			7	8
驗證：掃描集合			8	9

- 初始化 - createIndex 正在準備索引建置器。此階段應該非常簡短。

- 建置索引：初始化 - 索引建置器正在準備建立索引。此階段應該非常簡短。
- 建置索引：掃描集合 - 索引建置器正在執行集合掃描以收集索引鍵。度量單位為「區塊」。

> **Note**
>
> 如果針對索引建置設定了多個工作者，則會顯示在此階段。「掃描集合」階段是唯一在索引建置過程中使用多個工作者的階段。所有其他階段都會顯示單一工作者。

- 建置索引：排序索引鍵 1 - 索引建置器正在排序收集的索引鍵。度量單位為「金鑰」。
- 建置索引：排序索引鍵 2 - 索引建置器正在排序與無效元組對應的收集索引鍵。此階段僅適用於唯一的索引建置。度量單位為「金鑰」。
- 建置索引：插入索引鍵 - 索引建置器正在將索引鍵插入新索引。度量單位為「金鑰」。
- 驗證：掃描索引 - createIndex 正在掃描索引，以尋找需要驗證的金鑰。度量單位為「區塊」。
- 驗證：排序元組 - createIndex 正在排序索引掃描階段的輸出。
- 驗證：掃描集合 - createIndex 正在掃描集合，以驗證前兩個階段中找到的索引鍵。度量單位為「區塊」。

索引建置輸出範例

在下列輸出範例中（前景索引建置），會顯示索引建立的狀態。「訊息」欄位會指出建置的階段和完成百分比，以摘要說明建置進度。「工作者」欄位表示索引建置的該階段期間所使用的工作者數量。「進度」欄位會顯示用於計算完成百分比的實際數字。

> **Note**
>
> Amazon DocumentDB 4.0 版不支援「currentIndexBuildName」、「msg」和「progress」欄位。

```
{
    "inprog" : [{
    …
        "command": {
            "createIndexes": "test",
            "indexes": [{
                "v": 2,
                "key": {
                    "user_name": 1
```

```
                },
                "name": "user_name_1"
            }],
            "lsid": {
                "id": UUID("094d0fba-8f41-4373-82c3-7c4c7b5ff13b")
            },
            "$db": "test"
        },
        "currentIndexBuildName": user_name_1,
        "msg": "Index Build: building index number_1, stage 6/6 building index:
 656860/1003520 (keys) 65%",
        "workers": 1,
        "progress": {
            "done": 656861,
            "total": 1003520
        },
    …
    ],

    "ok" : 1
}
```

維護 Amazon DocumentDB 索引

主題

- 索引膨脹
- 使用 的索引維護 reIndex

索引膨脹

Amazon DocumentDB 使用多版本並行控制 (MVCC) 來管理並行交易。刪除或更新文件時，其先前的版本會保留在集合和索引中為「無效」版本。垃圾收集程序會自動從這些無效版本回收空間，以供未來操作使用。

當集合的索引因為頁面中累積無效或過時的索引項目或分段而變大時，就會發生索引膨脹。報告的百分比代表未來索引項目可以使用的索引空間量。此膨脹會耗用緩衝區快取和儲存體中的空間。如果您想要移除膨脹，則需要重建索引。

Example範例

執行下列命令來判斷索引的未使用儲存：

```
db.coll.aggregate({$indexStats:{}});
```

結果如下所示：

```
{
    "name" : "_id_",
    "key" : {
        "_id" : 1
    },
    "host" : "devbox-test.localhost.a2z.com:27317",
    "size" : NumberLong(827392),
    "accesses" : {
        "ops" : NumberLong(40000),
        "docsRead" : NumberLong(46049),
        "since" : ISODate("2025-04-03T21:44:51.251Z")
    },
    "cacheStats" : {
        "blksRead" : NumberLong(264),
        "blksHit" : NumberLong(140190),
        "hitRatio" : 99.8121
    },
    "unusedStorageSize" : {
        "unusedSizeBytes" : 409600,
        "unusedSizePercent" : 49.51
    }
}
```

您可以使用 reIndex命令重建索引，而不需要停機，這需要掃描整個集合。請參閱 使用 的索引維護 reIndex。

使用 的索引維護 **reIndex**

reIndex 是用來重建索引的命令。它通常在索引已損毀或效率不佳時使用。隨著時間的推移，索引可能會因為許多更新、插入或刪除而累積未使用的空間，導致效能降低。重新索引有助於移除這類未使用的空間，並還原索引的效率。

reIndex 準則

- reIndex 僅支援 Amazon DocumentDB 5.0。
- Amazon DocumentDB 支援背景中的reindex單一索引，允許多個工作者。程序reIndex執行時，舊索引可供查詢使用。

- Amazon DocumentDB 支援透過 編製進度報告的索引`currentOp`。您可以看到索引建置階段，類似於在索引建立期間索引建置階段檢視的 。唯一的區別是， `reIndex`一律有八個階段，無論其是否為唯一。沒有「建置索引：排序索引鍵 2」階段。
- `reIndex` 可與相同集合上的索引相關命令以外的任何命令同時執行：`createIndexes`、`collMod`、 `dropIndexes`和 `renameCollection`。
- `reIndex` 目前不支援文字、地理空間、向量和部分索引。

`reIndex` 組建

使用下列命令來重建索引：

```
db.runCommand({ reIndex: "collection-name", index: "index-name"})
```

您也可以選擇性地控制指派給重建程序的工作者數量：

```
db.runCommand({ reIndex: "collection-name", index: "index-name", workers: number })
```

管理集合層級文件壓縮

Amazon DocumentDB 5.0 集合層級文件壓縮可讓您壓縮集合中的文件，以降低儲存和 IO 成本。您可以在集合層級啟用文件壓縮，並視需要檢視壓縮指標，方法是透過壓縮文件的儲存大小和壓縮狀態等壓縮指標來測量儲存增益。Amazon DocumentDB 5.0 使用 LZ4 壓縮演算法來壓縮文件。

Amazon DocumentDB 支援從 5.0 版開始的文件壓縮。以下是集合層級文件壓縮函數：

- 預設行為 — 5.0 叢集上新集合的預設壓縮設定是由叢集參數 決定。`default_collection_compression`此參數預設為「停用」。
- 壓縮現有集合 — 可使用 `collMod`命令變更現有集合的壓縮設定。
- 變更壓縮閾值 — 預設壓縮閾值為 2KB。您可以使用 `createCollection`命令為新集合指定此值，並使用 `collMod`命令為現有集合變更此值。

> **Note**
>
> Amazon DocumentDB 3.6 和 4.0 版不支援 Amazon DocumentDB 文件壓縮。

主題

- 管理文件壓縮
- 監控文件壓縮

管理文件壓縮

在集合中啟用文件壓縮

使用 db.createCollection()方法在 Amazon DocumentDB 5.0 上建立集合時啟用文件壓縮：

```
db.createCollection( sample_collection,{
    storageEngine : {
        documentDB: {
            compression:{enable: <true | false>}
        }
    }
})
```

在叢集中啟用文件壓縮

叢集層級上的所有新集合預設都可以啟用文件壓縮，方法是將叢集參數default_collection_compression設定為「啟用」。當此參數設為「啟用」時，叢集上新建立的集合預設會啟用壓縮，壓縮閾值為 2 KB。

壓縮現有的集合

您也可以使用 collMod操作並指定下列storageEngine組態，修改現有集合的壓縮設定。請注意，使用此命令所做的變更僅適用於新插入/更新的文件，且先前插入文件的壓縮不會變更。

```
db.runCommand({
    collMod: "orders",
    storageEngine: {
        documentDB: {compression: {enable: <true | false>} }
    }
})
```

設定壓縮閾值

根據預設，壓縮集合的壓縮閾值為 2032 個位元組。建立啟用壓縮的新集合時，可以在createCollection命令中設定此閾值：

```
db.createCollection( sample_collection, {
    storageEngine : {
        documentDB: {
            compression: {
                enable: true,
                threshold: <128 - 8000>
            }
        }
    }
})
```

您也可以使用 collMod操作並指定下列storageEngine組態，修改現有壓縮集合的壓縮閾值：

```
db.runCommand({
    collMod: "orders",
    storageEngine: {
        documentDB: {
            compression: {
                enable: true,
                threshold: <128 - 8000>
            }
        }
    }
})
```

請注意，壓縮閾值只能設定為介於 128 到 8000 個位元組之間的值。此外，指定壓縮閾值時，需要將enable選項設定為「true」。

監控文件壓縮

您可以檢查集合是否已壓縮，並計算其壓縮比率，如下所示。

從 mongo shell 執行 db.printCollectionStats()或 db.collection.stats()命令來檢視壓縮統計資料。輸出會顯示您可以比較的原始大小和壓縮大小，以分析文件壓縮的儲存增益。在此範例中，名為「sample_collection」的集合統計資料如下所示。擴展係數 1024*1024 用於輸出 size和 storageSize 值，以 MB 為單位。

```
db.sample_collection.stats(1024*1024)
```

以下是上述命令的輸出範例：

```
{
    "ns" : "test.sample_collection",
    "count" : 1000000,
    "size" : 3906.3,
    "avgObjSize" : 4096,
    "storageSize" : 1953.1,
    compression:{"enabled" : true,"threshold" : 2032},
    ...
}
```

- size - 文件集合的原始大小。
- avgObjSize - 壓縮前的平均文件大小，四捨五入為第一個小數。度量單位為位元組。
- storageSize - 壓縮後集合的儲存大小。度量單位為位元組。
- 已啟用 - 指出是否啟用或停用壓縮。

若要計算實際壓縮比率，請將集合大小除以儲存大小 (size/storageSize)。針對上述範例，計算為 3906.3/1953.1，其轉換為 2：1 壓縮率。

在 Amazon DocumentDB 8.0 中管理字典型壓縮

Amazon DocumentDB 8.0 推出新的文件壓縮演算法 zstd，作為 LZ4 的改善替代方案。當您選取 Zstd 做為壓縮演算法，在 Amazon DocumentDB 8.0 集合上啟用字典壓縮時，會取樣集合中的文件，以建立自訂壓縮字典。新的和更新的文件會使用此字典和 zstd 演算法壓縮。這種方法通常會達到比標準壓縮方法更高的壓縮比率，特別是對於具有一致文件結構描述或重複欄位名稱的集合。

Lz4 是一種專為快速壓縮和解壓縮而設計的演算法。已知在 CPU 上很亮，同時實現顯著的壓縮。Zstd 是一種一般用途演算法，具有預設設定，使用更多的 CPU，但壓縮率比 lz4 更好。字典的使用可進一步改善大多數 JSON 文件的壓縮。Zstd 演算法的一些優點如下：

- 降低儲存成本：較高的壓縮比率意味著較低的儲存用量和較低的成本。
- 較低的 I/O：壓縮的文件需要較少的 I/O，可能會改善效能。
- 針對您的集合進行最佳化：字典專門針對集合的資料模式進行訓練。

> **Note**
>
> Amazon DocumentDB 3.6、4.0 和 5.0 版不支援字典型壓縮。

效能考量

Zstd 壓縮涉及下列權衡：

- 儲存體與 CPU 的比較：Zstd 壓縮可實現更好的壓縮率，但相較於 LZ4 壓縮，可能會使用稍微多一點的 CPU 資源。
- 初始壓縮：在插入足夠的文件來訓練有效的字典之前，新集合可能無法達到最佳壓縮。目前，如果集合至少 100 個文件，則會訓練字典。
- 工作負載類型：讀取密集型工作負載，其中整個資料都符合緩衝區快取，可能會因為解壓縮負荷而增加延遲和 CPU 用量。

Zstd 壓縮對於具有小型文件、文件陣列和重複欄位名稱的集合特別有效。

啟用字典型壓縮

對於新集合，您可以使用下列命令來啟用 Zstd 壓縮：

```
db.createCollection("myCollection",
    {
        storageEngine: {
            documentDB: {
                compression: {
                    enable: true,
                    algorithm: "zstd"
                }
            }
        }
    }
 )
```

您也可以在現有集合上啟用或修改壓縮：

```
db.runCommand({
    collMod: "myCollection",
    storageEngine: {
        documentDB: {
            compression: {
```

```
                enable: true,
                algorithm: "zstd"
            }
        }
    }
})
```

若要啟用叢集上所有集合的 Zstd 演算法，您可以修改叢集參數群組，以選取「zstd」作為參數「default_collection_compression」的值。

開始使用

Amazon DocumentDB 8.0 預設會開啟 Zstd 壓縮。您一律可以在叢集參數群組中將 'default_compression' 的值設定為停用，以將其關閉。請注意，從 Amazon DocumentDB 8.0 開始，「啟用」不再是 default_compression 的有效選擇，您必須從 Zstd 和 LZ4 中選取。

監控

您可以使用下列其中一個命令來檢視集合的壓縮資訊：

- db.runCommand({ collStats： "myCollection" }) 或
- db.collection.stats()

這些命令會傳回可用於計算壓縮比率的金鑰統計資料：

- compression.algorithm：使用的演算法 ("lz4" 或 "zstd")
- storageSize：壓縮後集合使用的實際儲存體。請注意，此數字包含分段 （ 也就是資料庫頁面中未使用的空間 ）
- avgObjSize：解壓縮集合文件的平均邏輯大小。請注意，如果您的集合有超過 20，000 個文件，此數字將根據 20，000 個文件的範例進行估算。
- size：沒有壓縮的集合邏輯大小。此數字是透過將 avgObjSize 乘以集合中的文件總數來取得，因此如果 avgObjSize 是近似值，則此數字也會是近似值。
- 計數：集合中的文件數量

下列 CloudWatch 指標在評估字典型壓縮時很有幫助：

- CPUUtilization

- FreeableMemory
- VolumeBytesUsed
- VolumeReadIOPs
- VolumeWriteIOPs

collStats 指標：

- storageSize
- size

此外，追蹤應用程式特定的指標很有用，例如每個查詢類型或 API 的延遲和輸送量。

Amazon DocumentDB 的安全性

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

安全性是 AWS 與您之間的共同責任。本文件可協助您了解如何在使用 Amazon DocumentDB 時套用共同責任模型。共同責任模型 將此描述為雲端的安全和雲端內的安全：

- 雲端的安全性 — AWS 負責保護在 AWS 雲端中執行 AWS 服務的基礎設施。 AWS 也為您提供可安全使用的服務。在 AWS 合規計畫中，第三方稽核員會定期測試並驗證我們的安全功效。若要了解適用於 Amazon DocumentDB 的合規計劃（具有 MongoDB 相容性），請參閱AWS 合規計劃範圍內的服務。
- 雲端的安全性 — 您的責任取決於您使用 AWS 的服務。您也必須對資料敏感度、組織要求，以及適用法律和法規等其他因素負責。

Amazon DocumentDB 根據聯邦風險與授權管理計劃 (FedRAMP) 授權。它具有 FedRAMP 高授權 for AWS GovCloud (US) 區域和 FedRAMP AWS 美國東部/西部區域的中等授權。如需 AWS 和合規工作的詳細資訊，請參閱AWS 合規計劃範圍內的服務。

> **Note**
>
> 本章適用於執行個體型叢集和彈性叢集。如需詳細資訊，請參閱下列主題。

您也會了解如何使用其他 AWS 服務來協助您監控和保護 Amazon DocumentDB 資源。下列主題說明如何設定 Amazon DocumentDB 以符合您的安全與合規目標。

主題

- 使用 Amazon DocumentDB 和 進行密碼管理 AWS Secrets Manager
- Amazon DocumentDB 中的資料保護
- Amazon DocumentDB 的 Identity and Access Management
- 預防跨服務混淆代理人
- 使用 IAM 身分進行身分驗證
- 使用 SCRAM-SHA-256 進行身分驗證
- 管理 Amazon DocumentDB 使用者

- 使用角色型存取控制的資料庫存取
- 更新您的 Amazon DocumentDB TLS 憑證
- Amazon DocumentDB 中的合規驗證
- Amazon DocumentDB 中的彈性
- Amazon DocumentDB 中的基礎設施安全性
- Amazon DocumentDB API 和界面 VPC 端點 (AWS PrivateLink)
- Amazon DocumentDB 的安全最佳實務
- 稽核 Amazon DocumentDB 事件
- Amazon VPC 和 Amazon DocumentDB

使用 Amazon DocumentDB 和 進行密碼管理 AWS Secrets Manager

Amazon DocumentDB 與 Secrets Manager 整合，以管理叢集的主要使用者密碼。

主題

- Secrets Manager 與 Amazon DocumentDB 整合的限制
- 使用 管理主要使用者密碼的概觀 AWS Secrets Manager
- 在 中強制執行主要使用者密碼的 Amazon DocumentDB 管理 AWS Secrets Manager
- 使用 Secrets Manager 管理叢集的主要使用者密碼

Secrets Manager 與 Amazon DocumentDB 整合的限制

下列功能不支援使用 Secrets Manager 管理主要使用者密碼：

- 屬於 Amazon DocumentDB 全域資料庫的叢集
- Amazon DocumentDB 跨區域僅供讀取複本

使用 管理主要使用者密碼的概觀 AWS Secrets Manager

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

當您將資料庫秘密存放在 Secrets Manager 時， AWS 您的帳戶會產生費用。如需定價的詳細資訊，請參閱 AWS Secrets Manager 定價。

當您執行下列其中一個操作時，您可以指定 Amazon DocumentDB 在 Secrets Manager 中管理 Amazon DocumentDB 叢集的主要使用者密碼：

- 建立叢集
- 修改叢集

當您指定 Amazon DocumentDB 在 Secrets Manager 中管理主要使用者密碼時，Amazon DocumentDB 會產生密碼並將其存放在 Secrets Manager 中。您可以直接與秘密互動，以擷取主要使用者的登入資料。您也可以指定客戶受管金鑰來加密秘密，或使用 Secrets Manager 提供的 KMS 金鑰。

Amazon DocumentDB 會管理秘密的設定，並依預設每七天輪換一次秘密。您可以修改某些設定，例如輪換排程。如果您刪除在 Secrets Manager 中管理秘密的叢集，則也會刪除秘密及其相關聯的中繼資料。

若要使用秘密中的登入資料連線到叢集，您可以從 Secrets Manager 擷取秘密。如需詳細資訊，請參閱AWS Secrets Manager 《 使用者指南》中的從 取得秘密 AWS Secrets Manager，並使用 JDBC 搭配 AWS Secrets Manager 秘密中的登入資料連線至 SQL 資料庫。

在 中強制執行主要使用者密碼的 Amazon DocumentDB 管理 AWS Secrets Manager

您可以使用 IAM 條件金鑰來強制執行主要使用者密碼的 Amazon DocumentDB 管理 AWS Secrets Manager。下列政策不允許使用者建立或還原執行個體或叢集，除非主要使用者密碼是由 Secrets Manager 中的 Amazon DocumentDB 管理。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "rds:CreateDBCluster"
            ],
```

```
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "rds:ManageMasterUserPassword": false
                }
            }
        }
    ]
}
```

使用 Secrets Manager 管理叢集的主要使用者密碼

當您執行下列動作時，可以在 Secrets Manager 中設定主要使用者密碼的 Amazon DocumentDB 管理：

- 建立 Amazon DocumentDB 叢集
- 修改 Amazon DocumentDB 叢集

您可以使用 Amazon DocumentDB 主控台或 AWS CLI 來執行這些動作。

Amazon DocumentDB 中的資料保護

共同責任模型適用於 AWS Amazon DocumentDB 中的資料保護。如此模型所述， AWS 負責保護執行所有 的全域基礎設施 AWS 雲端。您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使用 AWS 服務 的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱資料隱私權常見問答集。如需歐洲資料保護的詳細資訊，請參閱一般資料保護規則 (GDPR) 中心。

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

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

- 如果您在 AWS 透過命令列界面或 API 存取 時需要 FIPS 140-3 驗證的密碼編譯模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-3。

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

主題

- 用戶端欄位層級加密
- 加密靜態 Amazon DocumentDB 資料
- 加密傳輸中的資料
- 金鑰管理

用戶端欄位層級加密

Amazon DocumentDB 用戶端欄位層級加密 (FLE) 可讓您先加密用戶端應用程式中的敏感資料，再傳輸至 Amazon DocumentDB 叢集。敏感資料在叢集中存放和處理時保持加密，並在擷取時在用戶端應用程式解密。

主題

- 開始使用
- 在用戶端 FLE 中查詢

開始使用

Amazon DocumentDB 中用戶端 FLE 的初始組態是一個四步驟程序，包括建立加密金鑰、將角色與應用程式建立關聯、設定應用程式，以及使用加密選項定義 CRUD 操作。

主題

- 步驟 1：建立加密金鑰
- 步驟 2：將角色與應用程式建立關聯
- 步驟 3：設定應用程式
- 步驟 4：定義 CRUD 操作
- 範例：用戶端欄位層級加密組態檔案

步驟 1：建立加密金鑰

使用 AWS Key Management Service建立對稱客戶受管金鑰，以加密和解密敏感資料欄位，並授予其必要的 IAM 使用許可。 會 AWS KMS 存放客戶受管金鑰，用於加密資料金鑰。在 中存放客戶受管金鑰可 AWS KMS 強化您的安全狀態。資料金鑰是存放在 Amazon DocumentDB 集合中的次要金鑰，在將文件存放在 Amazon DocumentDB 之前，需要先加密敏感欄位。客戶受管金鑰會加密資料金鑰，進而加密和解密您的資料。如果您使用全域叢集，您可以建立多區域金鑰，讓不同的服務角色可以在不同的區域中使用。

如需 的詳細資訊 AWS Key Management Service，包括如何建立金鑰，請參閱《 AWS Key Management Service 開發人員指南》中的AWS Key Management Service 概念。

步驟 2：將角色與應用程式建立關聯

建立具有適當 AWS KMS 許可的 IAM 政策。此政策允許其連接的 IAM 身分加密和解密資源欄位中指定的 KMS 金鑰。您的應用程式會擔任此 IAM 角色進行身分驗證 AWS KMS。

政策看起來應該類似：

```
{ "Effect": "Allow",
"Action": ["kms:Decrypt", "kms:Encrypt"],
"Resource": "Customer Key ARN"
}
```

步驟 3：設定應用程式

您已在 中定義客戶受管金鑰， AWS KMS 並建立具有必要存取許可的 IAM 角色。匯入下列必要的套件。

```
import boto3
import json
import base64
from pymongo import MongoClient
from pymongo.encryption import (Algorithm,
                                ClientEncryption)
```

```
# create a session object:
my_session = boto3.session.Session()

# get access_key and secret_key programmatically using get_frozen_credentials() method:
```

```
current_credentials = my_session.get_credentials().get_frozen_credentials()
```

1. 指定 aws做為 AWS KMS 提供者類型，並輸入您在上一個步驟中擷取的帳戶登入資料。

```
provider = "aws"
kms_providers = {
    provider: {
        "accessKeyId": current_credentials.access_key,
        "secretAccessKey": current_credentials.secret_key
    }
}
```

2. 指定客戶受管金鑰來加密資料金鑰：

```
customer_key = {
“region”: “AWS region of the customer_key”,
    “key”: “customer_key ARN”
}

key_vault_namespace = "encryption.dataKeys"

key_alt_name = 'TEST_DATA_KEY'
```

3. 設定 MongoClient 物件：

```
client = MongoClient(connection_string)

coll = client.test.coll
coll.drop()

client_encryption = ClientEncryption(
    kms_providers, # pass in the kms_providers variable from the previous step
    key_vault_namespace = key_vault_namespace,
    client,
    coll.codec_options
)
```

4. 產生資料金鑰：

```
data_key_id = client_encryption.create_data_key(provider,
    customer_key,
    key_alt_name = [key_alt_name])
```

5. 擷取現有的資料金鑰：

```
data_key = DataKey("aws",
    master_key = customer_key)
key_id = data_key["_id"]
data_key_id = client[key_vault_namespace].find_one({"_id": key_id})
```

步驟 4：定義 CRUD 操作

使用加密選項定義 CRUD 操作。

1. 定義要write/read/delete單一文件的集合：

```
coll = client.gameinfo.users
```

2. 明確加密 - 加密欄位並插入：

> **Note**
>
> 必須提供「key_id」或「key_alt_name」中的一個。

```
encrypted_first_name = client_encryption.encrypt(
    "Jane",
    Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Deterministic,
    key_id=data_key_id
)
encrypted_last_name = client_encryption.encrypt(
    "Doe",
    Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Deterministic,
    key_id=data_key_id
)
encrypted_dob = client_encryption.encrypt(
    "1990-01-01",
    Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Random,
    key_id=data_key_id
)

coll.insert_one(
    {"gamerTag": "jane_doe90",
    "firstName": encrypted_first_name,
```

```
    "lastName": encrypted_last_name,
    "dateOfBirth":encrypted_dob,
    "Favorite_games":["Halo","Age of Empires 2","Medal of Honor"]
})
```

範例：用戶端欄位層級加密組態檔案

在下列範例中，將每個####取代為您自己的資訊。

```
# import python packages:
import boto3
import json
import base64
from pymongo import MongoClient
from pymongo.encryption import (Algorithm,
                                ClientEncryption)

def main():

    # create a session object:
    my_session = boto3.session.Session()

    # get aws_region from session object:
    aws_region = my_session.region_name

    # get access_key and secret_key programmatically using get_frozen_credentials()
 method:
    current_credentials = my_session.get_credentials().get_frozen_credentials()
    provider = "aws"

    # define the kms_providers which is later used to create the Data Key:
    kms_providers = {
        provider: {
            "accessKeyId": current_credentials.access_key,
            "secretAccessKey": current_credentials.secret_key
        }
    }

    # enter the kms key ARN. Replace the example ARN value.
    kms_arn = "arn:aws:kms:us-east-1:123456789:key/abcd-efgh-ijkl-mnop"
    customer_key = {
        "region": aws_region,
        "key":kms_arn
```

```
    }

    # secrets manager is used to store and retrieve user credentials for connecting to
 an Amazon DocumentDB cluster.
    # retrieve the secret using the secret name. Replace the example secret key.
    secret_name = "/dev/secretKey"
    docdb_credentials = json.loads(my_session.client(service_name = 'secretsmanager',
 region_name = "us-east-1").get_secret_value(SecretId = secret_name)['SecretString'])

    connection_params = '/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false'
    conn_str = 'mongodb://' + docdb_credentials["username"] + ':' +
 docdb_credentials["password"] + '@' + docdb_credentials["host"] + ':' +
 str(docdb_credentials["port"]) + connection_params
    client = MongoClient(conn_str)

    coll = client.test.coll
    coll.drop()

    # store the encryption data keys in a key vault collection (having naming
 convention as db.collection):
    key_vault_namespace = "encryption.dataKeys"
    key_vault_db_name, key_vault_coll_name = key_vault_namespace.split(".", 1)

    # set up the key vault (key_vault_namespace) for this example:
    key_vault = client[key_vault_db_name][key_vault_coll_name]
    key_vault.drop()
    key_vault.create_index("keyAltNames", unique=True)

    client_encryption = ClientEncryption(
        kms_providers,
        key_vault_namespace,
        client,
        coll.codec_options)

    # create a new data key for the encrypted field:
    data_key_id = client_encryption.create_data_key(provider, master_key=customer_key,
 key_alt_names=["some_key_alt_name"], key_material = None)

    # explicitly encrypt a field:
    encrypted_first_name = client_encryption.encrypt(
    "Jane",
    Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Deterministic,
    key_id=data_key_id
```

```
    )
    coll.insert_one(
    {"gamerTag": "jane_doe90",
    "firstName": encrypted_first_name
    })
    doc = coll.find_one()
    print('Encrypted document: %s' % (doc,))

    # explicitly decrypt the field:
    doc["firstName"] = client_encryption.decrypt(doc["firstName"])
    print('Decrypted document: %s' % (doc,))

    # cleanup resources:
    client_encryption.close()
    client.close()

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

在用戶端 FLE 中查詢

Amazon DocumentDB 僅支援用戶端 FLE 加密值的等式查詢。

例如，若要查詢加密的玩家分數等於 500 的文件，用戶端會使用明確加密方法來加密查詢值：

```
encrypted_gamerscore_filter = client_encryption.encrypt(
    500,
    Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Deterministic,
    key_id=data_key_id
)

coll.find( {
    "gamerscore" : { "$eq" : encrypted_gamerscore_filter }
} )
```

加密靜態 Amazon DocumentDB 資料

> **Note**
>
> AWS KMS 正在將術語客戶主金鑰 (CMK) 取代為 AWS KMS key和 KMS 金鑰。概念並沒有變更。為了防止中斷變更， AWS KMS 正在保留此術語的一些變化。

您可以在建立叢集時指定儲存加密選項，以加密 Amazon DocumentDB 叢集中的靜態資料。儲存加密功能一經啟用，範圍擴及整個叢集，並套用到所有執行個體，包括主執行個體和任何複本。它也會套用至叢集的儲存磁碟區、資料、索引、日誌、自動備份和快照。

Amazon DocumentDB 使用 256 位元進階加密標準 (AES-256)，使用存放在 AWS Key Management Service () 中的加密金鑰來加密您的資料AWS KMS。使用啟用靜態加密的 Amazon DocumentDB 叢集時，您不需要修改應用程式邏輯或用戶端連線。Amazon DocumentDB 透明地處理資料的加密和解密，對效能的影響最小。

Amazon DocumentDB 與 整合， AWS KMS 並使用稱為信封加密的方法來保護您的資料。當 Amazon DocumentDB 叢集使用 加密時 AWS KMS，Amazon DocumentDB AWS KMS 會要求 使用您的 KMS 金鑰來產生加密文字資料金鑰來加密儲存磁碟區。加密文字資料金鑰會使用您定義的 KMS 金鑰進行加密，並與加密的資料和儲存中繼資料一起存放。當 Amazon DocumentDB 需要存取加密的資料時，它會請求使用 KMS 金鑰 AWS KMS 解密加密文字資料金鑰，並在記憶體中快取純文字資料金鑰，以有效率地加密和解密儲存磁碟區中的資料。

Amazon DocumentDB 中的儲存加密設施適用於所有支援的執行個體大小，以及可使用 Amazon DocumentDB 的所有 AWS 區域 執行個體。

啟用 Amazon DocumentDB 叢集的靜態加密

您可以使用 或 AWS 管理主控台 AWS Command Line Interface () 佈建叢集時，在 Amazon DocumentDB 叢集上啟用或停用靜態加密AWS CLI。您使用主控台所建立的叢集，依預設會啟用靜態加密。您使用 建立的叢集預設 AWS CLI 會停用靜態加密。因此，您必須使用 `--storage-encrypted` 參數明確啟用靜態加密。在任何一種情況下，當叢集建立之後，您都無法變更靜態加密選項。

Amazon DocumentDB 使用 AWS KMS 來擷取和管理加密金鑰，並定義控制如何使用這些金鑰的政策。如果您未指定 AWS KMS 金鑰識別符，Amazon DocumentDB 會使用預設的 AWS 受管服務 KMS 金鑰。Amazon DocumentDB 會為 AWS 區域 中的每個 建立單獨的 KMS 金鑰 AWS 帳戶。如需詳細資訊，請參閱 AWS Key Management Service 概念。

若要開始建立您自己的 KMS 金鑰，請參閱《 AWS Key Management Service 開發人員指南》中的入門。

> **Important**
>
> 您必須使用對稱加密 KMS 金鑰來加密叢集，因為 Amazon DocumentDB 僅支援對稱加密 KMS 金鑰。請勿使用非對稱 KMS 金鑰來嘗試加密 Amazon DocumentDB 叢集中的資料。如

需詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》中的 中的非對稱金鑰 AWS KMS。

如果 Amazon DocumentDB 無法再存取叢集的 KMS 金鑰 — 例如，當擁有金鑰 AWS 帳戶 的 暫停、停用金鑰、排程刪除金鑰，或移除 Amazon DocumentDB 依賴的金鑰政策或授權時 — 叢集會先轉換為 `inaccessible-encryption-credentials-recoverable` 狀態。當叢集處於此狀態時，Amazon DocumentDB 會停止叢集的執行個體，您無法從叢集讀取或寫入叢集，但如果在 7 天內還原對 KMS 金鑰的存取，仍可復原叢集。如果未在 7 天內還原存取權，叢集會轉換為終端機`inaccessible-encryption-credentials`狀態。從終端狀態，叢集不再可用，且資料庫的目前狀態無法復原 – 您只能從備份還原或使用原始 KMS 金鑰執行point-in-time還原。對於 Amazon DocumentDB，備份一律會啟用至少 1 天。

Note

屬於全域叢集的叢集行為不同。當 Amazon DocumentDB 偵測到無法再存取 KMS 金鑰時，全域叢集中的所有叢集會直接轉換為終端機`inaccessible-encryption-credentials`狀態，略過可復原狀態。這是因為屬於全域叢集的叢集只有在它是全域叢集中唯一的叢集時，才能停止和啟動。若要復原，您必須從快照還原或執行point-in-time還原。若要刪除原始叢集，您必須先從全域叢集中移除每個叢集，然後將其刪除。

Important

建立加密叢集之後，就無法變更該叢集的 KMS 金鑰。務必在加密叢集建立之前決定您的加密金鑰需求。

Using the AWS 管理主控台

您可以在建立叢集時指定靜態加密選項。當您使用 AWS 管理主控台建立叢集時，依預設會啟用靜態加密。叢集建立之後就無法變更此選項。

在建立叢集時指定靜態加密選項

1. 如入門一節所述建立 Amazon DocumentDB 叢集。不過，在步驟 6 中，請勿選擇 Create cluster (建立叢集)。
2. 在 Authentication (身分驗證) 區段下，選擇 Show advanced settings (顯示進階設定)。

3. 向下捲動到 Encryption-at-rest (靜態加密) 區段。
4. 選擇您要靜態加密採用的選項。無論選擇哪個選項，叢集建立之後都無法變更。
   - 若要對此叢集中的資料啟用靜態加密，請選擇 Enable encryption (啟用加密)。
   - 如果您不要對此叢集中的資料啟用靜態加密，請選擇 Disable encryption (停用加密)。
5. 選擇您想要的主索引鍵。Amazon DocumentDB 使用 AWS Key Management Service (AWS KMS) 來擷取和管理加密金鑰，並定義控制如何使用這些金鑰的政策。如果您未指定 AWS KMS 金鑰識別符，Amazon DocumentDB 會使用預設的 AWS 受管服務 KMS 金鑰。如需詳細資訊，請參閱 AWS Key Management Service 概念。

   > **Note**
   >
   > 建立加密叢集後，您無法變更該叢集的 KMS 金鑰。務必在加密叢集建立之前決定您的加密金鑰需求。

6. 請視需要完成其他區段，並建立您的叢集。

Using the AWS CLI

若要使用 加密 Amazon DocumentDB 叢集 AWS CLI，請執行 `create-db-cluster`命令並指定 `--storage-encrypted`選項。根據預設，使用 建立的 Amazon DocumentDB 叢集 AWS CLI 不會啟用儲存加密。

下列範例會建立已啟用儲存加密的 Amazon DocumentDB 叢集。

在下列範例中，將每個*#########*取代為您叢集的資訊。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb create-db-cluster \
  --db-cluster-identifier mydocdbcluster \
  --port 27017 \
  --engine docdb \
  --master-username SampleUser1 \
  --master-user-password primaryPassword \
  --storage-encrypted
```

針對 Windows：

```
aws docdb create-db-cluster ^
  --db-cluster-identifier SampleUser1 ^
  --port 27017 ^
  --engine docdb ^
  --master-username SampleUser1 ^
  --master-user-password primaryPassword ^
  --storage-encrypted
```

當您建立加密的 Amazon DocumentDB 叢集時，您可以指定 AWS KMS 金鑰識別符，如下列範例所示。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb create-db-cluster \
  --db-cluster-identifier SampleUser1 \
  --port 27017 \
  --engine docdb \
  --master-username primaryUsername \
  --master-user-password yourPrimaryPassword \
  --storage-encrypted \
  --kms-key-id key-arn-or-alias
```

針對 Windows：

```
aws docdb create-db-cluster ^
  --db-cluster-identifier SampleUser1 ^
  --port 27017 ^
  --engine docdb ^
  --master-username SampleUser1 ^
  --master-user-password primaryPassword ^
  --storage-encrypted ^
  --kms-key-id key-arn-or-alias
```

> **Note**
>
> 建立加密叢集後，您無法變更該叢集的 KMS 金鑰。務必在加密叢集建立之前決定您的加密金鑰需求。

解決處於無法存取加密狀態的 Amazon DocumentDB 叢集

當 Amazon DocumentDB 無法存取叢集加密所用的 KMS 金鑰時，Amazon DocumentDB 叢集會移至無法存取的加密狀態。有兩種這類狀態，而復原路徑取決於叢集所在的狀態。

`inaccessible-encryption-credentials-recoverable` 狀態

當叢集處於 `inaccessible-encryption-credentials-recoverable` 狀態時，您可以透過`available`還原 Amazon DocumentDB 對 KMS 金鑰的存取權，然後啟動叢集，將叢集傳回 。若要解決此狀態，請執行下列動作：

1. 確認 AWS 帳戶 擁有 KMS 金鑰的 處於作用中狀態。如果帳戶遭到暫停，請重新啟用。
2. 確認 KMS 金鑰已啟用。如需詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》中的啟用和停用金鑰。
3. 檢查 KMS 金鑰是否已排定刪除。如果是，請取消排程金鑰刪除。如需詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》中的排程和取消金鑰刪除。
4. 確認 Amazon DocumentDB 依賴的 KMS 金鑰政策和任何授予仍允許 Amazon DocumentDB 使用金鑰。
5. 還原 KMS 金鑰的存取權後，請使用 AWS 管理主控台 或執行 `start-db-cluster` AWS CLI 命令來啟動叢集。

   Example

   針對 Linux、macOS 或 Unix：

   ```
   aws docdb start-db-cluster \
     --db-cluster-identifier example-cluster
   ```

   針對 Windows：

   ```
   aws docdb start-db-cluster ^
     --db-cluster-identifier example-cluster
   ```

> **Important**
>
> 如果未在 7 天內還原對 KMS 金鑰的存取，叢集會轉換為無法從中啟動的終端機`inaccessible-encryption-credentials`狀態。

`inaccessible-encryption-credentials` 狀態

`inaccessible-encryption-credentials` 狀態為終端機。無法啟動叢集，也無法復原資料庫的執行中狀態。若要復原資料，請從快照還原或執行point-in-time還原至新叢集。您仍然必須能夠存取原始 KMS 金鑰才能執行還原。如果刪除 KMS 金鑰，則無法復原資料。

如需詳細資訊，請參閱從叢集快照還原及還原至某個時間點。

> **Note**
>
> 屬於全域叢集的叢集會在無法存取 KMS 金鑰時直接轉換為 `inaccessible-encryption-credentials` 狀態，因為屬於全域叢集的叢集只有在它是全域叢集中唯一的叢集時，才能停止和啟動。若要刪除處於此狀態的叢集，請先從全域叢集中移除每個叢集，然後個別刪除叢集。

如果您因為已啟用刪除保護而無法刪除處於 `inaccessible-encryption-credentials` 狀態的叢集，請在重試刪除 AWS CLI 之前使用 關閉刪除保護。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster \
  --db-cluster-identifier example-cluster \
  --no-deletion-protection
```

針對 Windows：

```
aws docdb modify-db-cluster ^
  --db-cluster-identifier example-cluster ^
  --no-deletion-protection
```

然後，您可以使用 `delete-db-cluster`命令刪除叢集。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb delete-db-cluster \
  --db-cluster-identifier example-cluster \
```

```
  --skip-final-snapshot
```

針對 Windows：

```
aws docdb delete-db-cluster ^
  --db-cluster-identifier example-cluster ^
  --skip-final-snapshot
```

如果叢集在您執行上述命令後未刪除，請聯絡 AWS 支援。

Amazon DocumentDB 加密叢集的限制

Amazon DocumentDB 加密叢集存在下列限制。

- 您只能在建立 Amazon DocumentDB 叢集時啟用或停用靜態加密，而不是在建立叢集之後。不過，您可以建立未加密叢集的快照，然後將未加密的快照還原為新叢集，同時指定靜態加密選項，以建立未加密叢集的加密複本。

  如需詳細資訊，請參閱下列主題：

  - 建立手動叢集快照
  - 從叢集快照還原
  - 複製 Amazon DocumentDB 叢集快照
- 啟用儲存加密的 Amazon DocumentDB 叢集無法修改以停用加密。
- Amazon DocumentDB 叢集中的所有執行個體、自動備份、快照和索引都會使用相同的 KMS 金鑰加密。

加密傳輸中的資料

您可以使用 Transport Layer Security (TLS) 來加密應用程式與 Amazon DocumentDB 叢集之間的連線。預設會為新建立的 Amazon DocumentDB 叢集啟用傳輸中加密。它可以選擇性地在建立叢集時或稍後停用。當啟用傳輸中的加密時，需要使用 TLS 的安全連線來連線到叢集。如需使用 TLS 連接到 Amazon DocumentDB 的詳細資訊，請參閱以程式設計方式連線至 Amazon DocumentDB。

管理 Amazon DocumentDB 叢集 TLS 設定

Amazon DocumentDB 叢集的傳輸中加密是透過叢集參數群組中的 TLS 參數進行管理。您可以使用 AWS 管理主控台 或 AWS Command Line Interface () 管理您的 Amazon DocumentDB 叢集 TLS 設定 AWS CLI。請參閱下列章節，了解如何驗證和修改您目前的 TLS 設定。

Using the AWS 管理主控台

請依照下列步驟，使用主控台執行 TLS 加密的管理任務，例如識別參數群組、驗證 TLS 值，以及進行必要的修改。

> **Note**
>
> 除非您在建立叢集時有不同指定，否則系統會使用預設的叢集參數群組來建立您的叢集。`default` 叢集參數群組中的參數無法修改 (例如，啟用/停用的 `tls`)。因此如果您的叢集使用 `default` 叢集參數群組，則您需要修改叢集，才能使用非預設叢集參數群組。首先，您可能需要建立自訂叢集參數群組。如需詳細資訊，請參閱建立 Amazon DocumentDB 叢集參數群組。

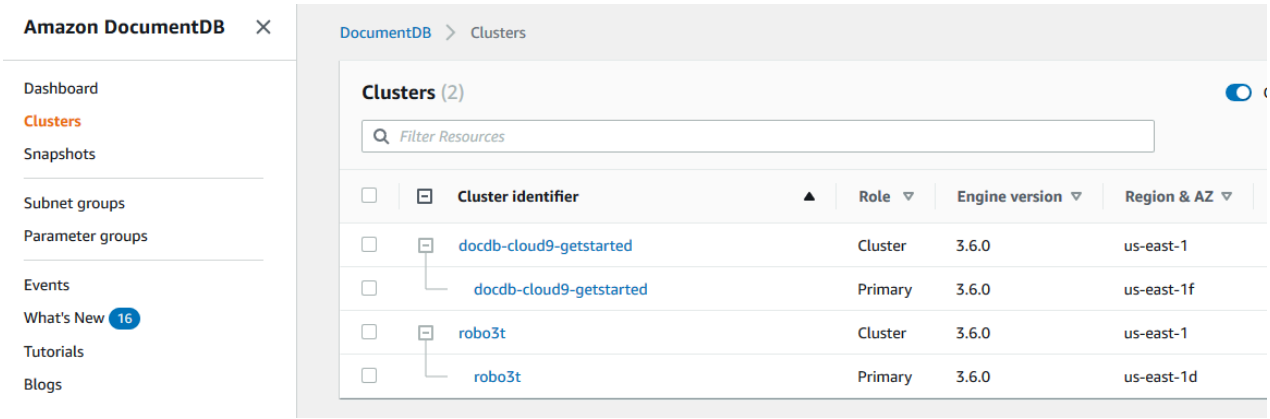
1. 判定您叢集正在使用的叢集參數群組。

   a. 開啟位於 https：//https://console.aws.amazon.com/docdb 的 Amazon DocumentDB 主控台。

   b. 在導覽窗格中，選擇叢集。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

   c. 請注意，在叢集導覽方塊中，資料欄叢集識別符會顯示叢集和執行個體。執行個體會列在叢集下方。如需參考，請參閱下列螢幕擷取畫面。



   d. 選擇您感興趣的叢集。

e. 選擇組態索引標籤並向下捲動至叢集詳細資訊底部，然後尋找叢集參數群組。請記下叢集參數群組的名稱。

如果叢集參數群組的名稱為 `default` (例如，`default.docdb3.6`)，則您必須具有一個自訂叢集參數群組，並使其成為叢集的參數群組，然後再繼續。如需詳細資訊，請參閱下列內容：

1. 建立 Amazon DocumentDB 叢集參數群組 — 如果您沒有可以使用的自訂叢集參數群組，請建立一個。
2. 修改 Amazon DocumentDB 叢集 — 修改您的叢集以使用自訂叢集參數群組。

2. 判定 **`tls`** 叢集參數的目前值。

a. 開啟位於 https：//https://console.aws.amazon.com/docdb 的 Amazon DocumentDB 主控台。

b. 在導覽窗格中，選擇 Parameter groups (參數群組)。

c. 從叢集參數群組的清單中，選擇您有興趣之叢集參數群組的名稱。

d. 找出 Cluster parameters (叢集參數) 區段。在叢集參數清單中，找出 `tls` 叢集參數列。此時，必須留意下列四欄：

- 叢集參數名稱 — 叢集參數的名稱。若為管理 TLS，您對 `tls` 叢集參數有興趣。
- 值 — 每個叢集參數的目前值。
- 允許的值 — 可套用至叢集參數的值清單。
- 套用類型 — 靜態或動態。靜態叢集參數的變更必須在執行個體重新啟動後才能套用。可以立即或在執行個體重新啟動後套用動態叢集參數的變更。

3. 修改 **`tls`** 叢集參數的值。

如果 `tls` 的值不是所需的值，請針對這個叢集參數群組修改其值。若要變更 `tls` 叢集參數的值，請按照以下步驟從上節繼續進行。

a. 選擇叢集參數名稱 (`tls`) 左側的按鈕。

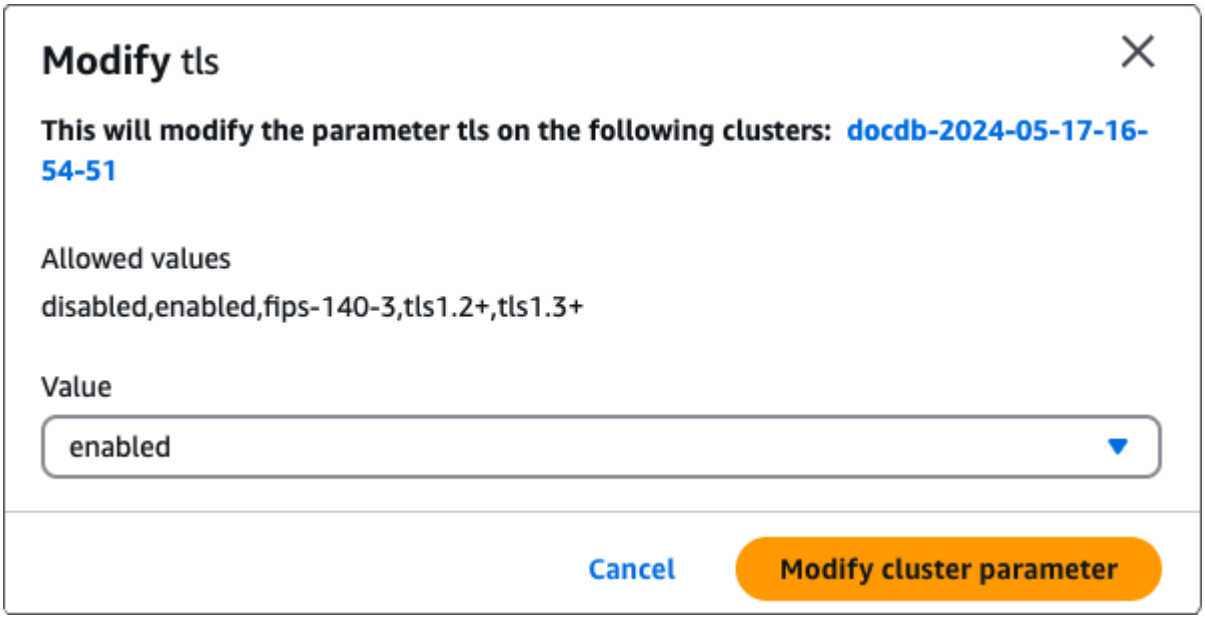
b. 選擇編輯。

c. 若要變更 `tls` 的值，請在 Modify **`tls`** (修改) 對話方塊中，從下拉式清單選擇您想要給與叢集參數的值。

有效的值如下：

- 停用 — 停用 TLS

- 啟用 — 啟用 TLS。對於 Amazon DocumentDB 5.0 和更早版本，這會啟用 TLS 1.0 到 1.3，但使用 TLS 1.2 或更高版本。從次要版本 5.0.1、Amazon DocumentDB 8.0 及更新版本開始，Amazon DocumentDB 5.0 僅支援 TLS 1.2 及更新版本。
- fips-140-3 — 使用 FIPS 啟用 TLS。叢集僅接受符合聯邦資訊處理標準 (FIPS) 版本 140-3 要求的安全連線。僅支援從這些區域中的 Amazon DocumentDB 5.0（引擎版本 3.0.3727) 叢集開始：ca-central-1、us-west-2、us-east-1、us-east-2、us-gov-east-1、us-gov-west-1。
- tls1.2+ — 啟用 TLS 1.2 版和更新版本。僅支援從 Amazon DocumentDB 4.0（引擎版本 2.0.10980) 和 Amazon DocumentDB（引擎版本 3.0.11051) 開始。
- tls1.3+ — 啟用 TLS 1.3 版和更新版本。僅支援從 Amazon DocumentDB 4.0（引擎版本 2.0.10980) 和 Amazon DocumentDB（引擎版本 3.0.11051) 開始。



d. 選擇 Modify cluster parameter (修改叢集參數)。此變更必須等到每個叢集執行個體重新啟動後才會套用。

4. 重新啟動 Amazon DocumentDB 執行個體。

   重新啟動叢集的每個執行個體，讓變更套用到叢集中的所有執行個體。

   a. 開啟位於 https：//https://console.aws.amazon.com/docdb 的 Amazon DocumentDB 主控台。

   b. 在導覽窗格中，選擇執行個體。

c. 若要指定要重新啟動的執行個體，請在執行個體清單中找到執行個體，並選擇其名稱左側的按鈕。

d. 選擇 Actions (動作)，然後 Reboot (重新啟動)。選擇 Reboot (重新啟動)，確認您要重新啟動。

Using the AWS CLI

請依照下列步驟，使用 執行 TLS 加密的管理任務 AWS CLI，例如識別參數群組、驗證 TLS 值，以及進行必要的修改。

> **Note**
>
> 除非您在建立叢集時有不同指定，否則系統會使用預設的叢集參數群組來建立該叢集。`default` 叢集參數群組中的參數無法修改 (例如，啟用/停用的 `tls`)。因此如果您的叢集使用 `default` 叢集參數群組，則您需要修改叢集，才能使用非預設叢集參數群組。您可能需要先建立自訂叢集參數群組。如需詳細資訊，請參閱建立 Amazon DocumentDB 叢集參數群組。

1. 判定您叢集正在使用的叢集參數群組。

   使用下列選項執行 `describe-db-clusters`命令：

   - `--db-cluster-identifier`
   - `--query`

   在下列範例中，將每個*#########*取代為您叢集的資訊。

   ```
   aws docdb describe-db-clusters \
     --db-cluster-identifier mydocdbcluster \
     --query 'DBClusters[*].[DBClusterIdentifier,DBClusterParameterGroup]'
   ```

   此操作的輸出如下所示 (JSON 格式) ：

   ```
   [
       [
           "mydocdbcluster",
           "myparametergroup"
   ```

```
    ]
]
```

如果叢集參數群組的名稱為 default (例如，default.docdb3.6)，則您必須具有一個自訂叢集參數群組，並使其成為叢集的參數群組，然後再繼續。如需詳細資訊，請參閱下列主題：

1. 建立 Amazon DocumentDB 叢集參數群組 — 如果您沒有可以使用的自訂叢集參數群組，請建立一個。
2. 修改 Amazon DocumentDB 叢集 — 修改叢集以使用自訂叢集參數群組。

2. 判定 **tls** 叢集參數的目前值。

若要取得此叢集參數群組的詳細資訊，請使用下列選項執行 describe-db-cluster-parameters命令：

- --db-cluster-parameter-group-name
- --query

  將輸出限制為僅感興趣的欄位：ParameterValueAllowedValues、 ParameterName和 ApplyType。

在下列範例中，將每個*#########*取代為您叢集的資訊。

```
aws docdb describe-db-cluster-parameters \
  --db-cluster-parameter-group-name myparametergroup \
  --query 'Parameters[*].[ParameterName,ParameterValue,AllowedValues,ApplyType]'
```

此操作的輸出如下所示 (JSON 格式) ：

```
[
    [
        "audit_logs",
        "disabled",
        "enabled,disabled",
        "dynamic"
    ],
    [
        "tls",
        "disabled",
        "disabled,enabled,fips-140-3,tls1.2+,tls1.3+",
```

```
        "static"
    ],
    [
        "ttl_monitor",
        "enabled",
        "disabled,enabled",
        "dynamic"
    ]
]
```

3. 修改 **tls** 叢集參數的值。

   如果 tls 的值不是所需的值，請針對這個叢集參數群組修改其值。若要變更tls叢集參數的值，請使用下列選項執行 modify-db-cluster-parameter-group命令：

   - --db-cluster-parameter-group-name – 必要。要修改之叢集參數群組的名稱。這不得為 default.* 叢集參數群組。
   - --parameters – 必要。叢集參數群組之要修改的參數清單。
     - ParameterName – 必要。要修改之叢集參數的名稱。
     - ParameterValue – 必要。此叢集參數的新值。必須是其中一個叢集參數的 AllowedValues。
       - enabled — 叢集接受使用 TLS 的安全連線。對於 Amazon DocumentDB 5.0 和更早版本，這會啟用 TLS 1.0 到 1.3，但使用 TLS 1.2 或更高版本。從次要版本 5.0.1、Amazon DocumentDB 8.0 及更新版本開始，Amazon DocumentDB 5.0 僅支援 TLS 1.2 及更新版本。
       - disabled — 叢集不接受使用 TLS 的安全連線。
       - fips-140-3 — 叢集僅接受符合聯邦資訊處理標準 (FIPS) 版本 140-3 要求的安全連線。僅支援從這些區域中的 Amazon DocumentDB 5.0（引擎版本 3.0.3727) 叢集開始：ca-central-1、us-west-2、us-east-1、us-east-2、us-gov-east-1、us-gov-west-1。
       - tls1.2+ — 叢集接受使用 TLS 1.2 版及更高版本的安全連線。僅支援從 Amazon DocumentDB 4.0（引擎版本 2.0.10980) 和 Amazon DocumentDB 5.0（引擎版本 3.0.11051) 開始。
       - tls1.3+ — 叢集接受使用 TLS 1.3 版及更高版本的安全連線。僅支援從 Amazon DocumentDB 4.0（引擎版本 2.0.10980) 和 Amazon DocumentDB 5.0（引擎版本 3.0.11051) 開始。
     - ApplyMethod — 套用此修改時。若為靜態叢集參數，例如 tle，此值必須是 pending-reboot。

- pending-reboot — 變更只有在重新啟動後才會套用至執行個體。您必須個別地重新啟動每個叢集執行個體，此變更才能在所有叢集執行個體中發生。

在下列範例中，將每個*#########*取代為您叢集的資訊。

下列程式碼會停用 tls，並在重新啟動時將變更套用至每個執行個體。

```
aws docdb modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=tls,ParameterValue=disabled,ApplyMethod=pending-
reboot"
```

下列程式碼會啟用 tls，並在重新啟動時將變更套用至每個執行個體。

```
aws docdb modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=tls,ParameterValue=enabled,ApplyMethod=pending-
reboot"
```

下列程式碼會使用 啟用 TLSfips-140-3，並在重新啟動時將變更套用至每個執行個體。

```
aws docdb modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup2 \
  --parameters "ParameterName=tls,ParameterValue=fips-140-3,ApplyMethod=pending-
reboot"
```

此操作的輸出如下所示 (JSON 格式）：

```
{
    "DBClusterParameterGroupName": "myparametergroup"
}
```

4. 重新啟動 Amazon DocumentDB 執行個體。

重新啟動叢集的每個執行個體，讓變更套用到叢集中的所有執行個體。若要重新啟動 Amazon DocumentDB 執行個體，請使用下列選項執行 reboot-db-instance命令：

- --db-instance-identifier

以下程式碼會重新啟動執行個體 mydocdbinstance。

在下列範例中，將每個*########*取代為您叢集的資訊。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb reboot-db-instance \
  --db-instance-identifier mydocdbinstance
```

針對 Windows：

```
aws docdb reboot-db-instance ^
  --db-instance-identifier mydocdbinstance
```

此操作的輸出如下所示 (JSON 格式) ：

```
{
    "DBInstance": {
        "AutoMinorVersionUpgrade": true,
        "PubliclyAccessible": false,
        "PreferredMaintenanceWindow": "fri:09:32-fri:10:02",
        "PendingModifiedValues": {},
        "DBInstanceStatus": "rebooting",
        "DBSubnetGroup": {
            "Subnets": [
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1a"
                    },
                    "SubnetIdentifier": "subnet-4e26d263"
                },
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1c"
                    },
                    "SubnetIdentifier": "subnet-afc329f4"
```

```
                },
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1e"
                    },
                    "SubnetIdentifier": "subnet-b3806e8f"
                },
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1d"
                    },
                    "SubnetIdentifier": "subnet-53ab3636"
                },
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1b"
                    },
                    "SubnetIdentifier": "subnet-991cb8d0"
                },
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1f"
                    },
                    "SubnetIdentifier": "subnet-29ab1025"
                }
            ],
            "SubnetGroupStatus": "Complete",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-91280df6",
            "DBSubnetGroupName": "default"
        },
        "PromotionTier": 2,
        "DBInstanceClass": "db.r5.4xlarge",
        "InstanceCreateTime": "2018-11-05T23:10:49.905Z",
        "PreferredBackupWindow": "00:00-00:30",
        "KmsKeyId": "arn:aws:kms:us-east-1:012345678901:key/0961325d-a50b-44d4-
b6a0-a177d5ff730b",
        "StorageEncrypted": true,
        "VpcSecurityGroups": [
            {
```

```
                "Status": "active",
                "VpcSecurityGroupId": "sg-77186e0d"
            }
        ],
        "EngineVersion": "3.6.0",
        "DbiResourceId": "db-SAMPLERESOURCEID",
        "DBInstanceIdentifier": "mydocdbinstance",
        "Engine": "docdb",
        "AvailabilityZone": "us-east-1a",
        "DBInstanceArn": "arn:aws:rds:us-east-1:012345678901:db:sample-cluster-
instance-00",
        "BackupRetentionPeriod": 1,
        "Endpoint": {
            "Address": "mydocdbinstance.corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
            "Port": 27017,
            "HostedZoneId": "Z2R2ITUGPM61AM"
        },
        "DBClusterIdentifier": "mydocdbcluster"
    }
}
```

執行個體重新啟動需要幾分鐘的時間來完成。您只能使用狀態為可用的執行個體。您可以使用主控台或 AWS CLI監控執行個體的狀態。如需詳細資訊，請參閱監控 Amazon DocumentDB 執行個體的狀態。

金鑰管理

Amazon DocumentDB 使用 AWS Key Management Service (AWS KMS) 來擷取和管理加密金鑰。AWS KMS 結合安全、高可用性的硬體和軟體，以提供針對雲端擴展的金鑰管理系統。使用 AWS KMS，您可以建立加密金鑰並定義控制如何使用這些金鑰的政策。 AWS KMS 支援 AWS CloudTrail，因此您可以稽核金鑰使用情況，以確認金鑰是否適當使用。

您可以搭配使用 KMS 金鑰與 Amazon DocumentDB 和支援的 AWS 服務，例如 Amazon Simple Storage Service (Amazon S3)、Amazon Relational Database Service (Amazon RDS)、Amazon Elastic Block Store (Amazon EBS) 和 Amazon Redshift。如需 支援的 服務清單 AWS KMS，請參閱《 AWS Key Management Service 開發人員指南》中的 AWS 服務使用方式 AWS KMS。如需 的資訊 AWS KMS，請參閱什麼是 AWS Key Management Service？

Amazon DocumentDB 的 Identity and Access Management

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

主題

- 目標對象
- 使用身分驗證
- 使用政策管理存取權
- Amazon DocumentDB 如何與 IAM 搭配使用
- Amazon DocumentDB 的身分型政策範例
- 對 Amazon DocumentDB 身分和存取進行故障診斷
- 管理 Amazon DocumentDB 資源的存取許可
- 針對 Amazon DocumentDB 使用身分型政策 (IAM 政策)
- AWS Amazon DocumentDB 的 受管政策
- Amazon DocumentDB API 許可：動作、資源和條件參考

目標對象

使用方式 AWS Identity and Access Management (IAM) 會根據您的角色而有所不同：

- 服務使用者 — 若無法存取某些功能，請向管理員申請所需許可 (請參閱 對 Amazon DocumentDB 身分和存取進行故障診斷)
- 服務管理員 — 負責設定使用者存取權並提交相關許可請求 (請參閱 Amazon DocumentDB 如何與 IAM 搭配使用)
- IAM 管理員 — 撰寫政策以管理存取控制 (請參閱 Amazon DocumentDB 的身分型政策範例)

使用身分驗證

身分驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者、IAM 使用者或擔任 IAM 角色身分進行身分驗證。

您可以使用身分來源的登入資料，例如 AWS IAM Identity Center (IAM Identity Center)、單一登入身分驗證或 Google/Facebook 登入資料，以聯合身分的形式登入。如需有關登入的詳細資訊，請參閱《AWS 登入 使用者指南》中的如何登入您的 AWS 帳戶。

對於程式設計存取， AWS 提供 SDK 和 CLI 以密碼編譯方式簽署請求。如需詳細資訊，請參閱《IAM 使用者指南》中的 API 請求的AWS 第 4 版簽署程序。

AWS 帳戶 根使用者

當您建立 時 AWS 帳戶，您會從一個名為 AWS 帳戶 theroot 使用者的登入身分開始，該身分具有對所有 AWS 服務 和 資源的完整存取權。強烈建議不要使用根使用者來執行日常任務。有關需要根使用者憑證的任務，請參閱《IAM 使用者指南》中的需要根使用者憑證的任務。

聯合身分

最佳實務是要求人類使用者使用聯合身分提供者，以 AWS 服務 使用臨時憑證存取 。

聯合身分是您企業目錄、Web 身分提供者的使用者，或是 AWS 服務 使用身分來源的憑證 Directory Service 存取的使用者。聯合身分會擔任角色，而該角色會提供臨時憑證。

若需集中化管理存取權限，建議使用 AWS IAM Identity Center。如需詳細資訊，請參閱 AWS IAM Identity Center 使用者指南中的什麼是 IAM Identity Center？。

IAM 使用者和群組

IAM 使用者https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html是一種身分具備單人或應用程式的特定許可權。建議以臨時憑證取代具備長期憑證的 IAM 使用者。如需詳細資訊，請參閱《IAM 使用者指南》中的要求人類使用者使用聯合身分提供者來 AWS 使用臨時憑證存取 。

IAM 群組會指定 IAM 使用者集合，使管理大量使用者的許可權更加輕鬆。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 使用者的使用案例。

IAM 角色

IAM 角色https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html的身分具有特定許可權，其可以提供臨時憑證。您可以透過從使用者切換到 IAM 角色 （主控台） 或呼叫 AWS CLI 或 AWS API 操作來擔任角色。如需詳細資訊，請參閱《IAM 使用者指南》中的擔任角色的方法。

IAM 角色適用於聯合身分使用者存取、臨時 IAM 使用者許可、跨帳戶存取權與跨服務存取，以及在 Amazon EC2 執行的應用程式。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的快帳戶資源存取。

使用政策管理存取權

您可以透過 AWS 建立政策並將其連接到身分或資源來控制 AWS 中的存取。政策定義與身分或資源相關聯的許可。當委託人提出請求時 AWS ， 會評估這些政策。大多數政策會以 JSON 文件 AWS 形式存放在 中。如需進一步了解 JSON 政策文件，請參閱《IAM 使用者指南》中的 JSON 政策概觀。

管理員會使用政策，透過定義哪些主體可在哪些條件下對哪些資源執行動作，以指定可存取的範圍。

預設情況下，使用者和角色沒有許可。IAM 管理員會建立 IAM 政策並將其新增至角色，供使用者後續擔任。IAM 政策定義動作的許可，無論採用何種方式執行。

身分型政策

身分型政策是附加至身分 (使用者、使用者群組或角色) 的 JSON 許可政策文件。這類政策控制身分可對哪些資源執行哪些動作，以及適用的條件。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

身分型政策可分為內嵌政策 (直接內嵌於單一身分) 與受管政策 (可附加至多個身分的獨立政策)。如需了解如何在受管政策及內嵌政策之間做選擇，請參閱《IAM 使用者指南》中的在受管政策與內嵌政策之間選擇。

資源型政策

資源型政策是附加到資源的 JSON 政策文件。範例包括 IAM 角色信任政策與 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。您必須在資源型政策中指定主體。

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

其他政策類型

AWS 支援其他政策類型，可設定更多常見政策類型授予的最大許可：

- 許可界限 — 設定身分型政策可授與 IAM 實體的最大許可。如需詳細資訊，請參閱《 IAM 使用者指南》中的 IAM 實體許可界限。
- 服務控制政策 (SCP) — 為 AWS Organizations中的組織或組織單位指定最大許可。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的服務控制政策。
- 資源控制政策 (RCP) — 設定您帳戶中資源可用許可的上限。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的資源控制政策 (RCP)。

- 工作階段政策 — 在以程式設計方式為角色或聯合身分使用者建立臨時工作階段時，以參數形式傳遞的進階政策。如需詳細資訊，請參《IAM 使用者指南》中的工作階段政策。

多種政策類型

當多種類型的政策適用於請求時，產生的許可會更複雜而無法理解。若要了解如何 AWS 在涉及多個政策類型時決定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

Amazon DocumentDB 如何與 IAM 搭配使用

在您使用 IAM 管理 Amazon DocumentDB 的存取權之前，請先了解哪些 IAM 功能可與 Amazon DocumentDB 搭配使用。

您可以搭配 Amazon DocumentDB 使用的 IAM 功能

IAM 功能	執行個體型叢集	彈性叢集
身分型政策	是	是
資源型政策	否	否
政策動作	是	是
政策資源	是	是
政策條件索引鍵 (服務特定)	是	是
ACL	否	否
ABAC(政策中的標籤)	部分	是
臨時憑證	是	是
主體許可	是	是
服務角色	是	是
服務連結角色	否	是

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

Amazon DocumentDB 的身分型政策

支援身分型政策：是

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

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。如要了解您在 JSON 政策中使用的所有元素，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素參考。

Amazon DocumentDB 的身分型政策範例

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

Amazon DocumentDB 中的資源型政策

支援資源型政策：否

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

如需啟用跨帳戶存取權，您可以在其他帳戶內指定所有帳戶或 IAM 實體作為資源型政策的主體。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的快帳戶資源存取。

Amazon DocumentDB 的政策動作

支援政策動作：是

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

JSON 政策的 `Action` 元素描述您可以用來允許或拒絕政策中存取的動作。政策會使用動作來授予執行相關聯動作的許可。

> **Note**
>
> 對於某些管理功能，Amazon DocumentDB 使用與 Amazon Relational Database Service (Amazon RDS) 共享的操作技術。
> 若要查看 RDS 動作的清單，請參閱《服務授權參考》中的 Amazon Relational Database Service 定義的動作。
> 若要檢視 Amazon DocumentDB 彈性叢集的政策動作，請參閱《服務授權參考》中的 Amazon DocumentDB 彈性叢集定義的動作。

Amazon DocumentDB 中的政策動作在動作之前使用以下字首：

```
aws
```

若要在單一陳述式中指定多個動作，請用逗號分隔。

```
"Action": [
      "aws:action1",
      "aws:action2"
         ]
```

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

Amazon DocumentDB 的政策資源

支援政策資源：是

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

`Resource` JSON 政策元素可指定要套用動作的物件。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。若動作不支援資源層級許可，使用萬用字元 (*) 表示該陳述式適用於所有資源。

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

> **Note**
>
> 對於某些管理功能，Amazon DocumentDB 使用與 Amazon Relational Database Service (Amazon RDS) 共享的操作技術。
> 若要檢視 RDS 資源類型及其 ARNs，請參閱《服務授權參考》中的 Amazon Relational Database Service 定義的資源。若要了解您可以使用哪些動作指定每個資源的 ARN，請參閱 Amazon Relational Database Service 定義的動作。
> 若要檢視 Amazon DocumentDB 彈性叢集的資源類型，請參閱服務授權參考中的 Amazon DocumentDB 彈性叢集定義的資源類型。

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

Amazon DocumentDB 的政策條件索引鍵

支援服務特定政策條件金鑰：是

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

`Condition` 元素會根據定義的條件，指定陳述式的執行時機。您可以建立使用條件運算子的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。若要查看所有 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的AWS 全域條件內容索引鍵。

> **Note**
>
> 對於某些管理功能，Amazon DocumentDB 使用與 Amazon Relational Database Service (Amazon RDS) 共享的操作技術。
> 若要查看 RDS 條件索引鍵的清單，請參閱《服務授權參考》中的 Amazon Relational Database Service 的條件索引鍵。若要了解您可以使用條件金鑰的動作和資源，請參閱 Amazon Relational Database Service 定義的動作。
> 若要檢視 Amazon DocumentDB 彈性叢集的條件金鑰，請參閱《服務授權參考》中的 Amazon DocumentDB 彈性叢集的條件金鑰。

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

Amazon DocumentDB 中的 ACLs

支援 ACL：否

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

ABAC 搭配 Amazon DocumentDB

> **Note**
>
> ABAC 僅部分支援以執行個體為基礎的叢集，但支援彈性叢集。

屬性型存取控制 (ABAC) 是一種授權策略，依據稱為標籤的屬性來定義許可。您可以將標籤連接至 IAM 實體 AWS 和資源，然後設計 ABAC 政策，以便在委託人的標籤符合資源上的標籤時允許操作。

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

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

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

搭配 Amazon DocumentDB 使用臨時憑證

支援臨時憑證：是

臨時登入資料提供 AWS 資源的短期存取權，當您使用聯合身分或切換角色時會自動建立。 AWS 建議您動態產生臨時登入資料，而不是使用長期存取金鑰。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的臨時安全憑證與可與 IAM 搭配運作的AWS 服務。

Amazon DocumentDB 的跨服務主體許可

支援轉寄存取工作階段 (FAS)：是

轉送存取工作階段 (FAS) 使用呼叫 的委託人許可 AWS 服務，並結合 AWS 服務 請求向下游服務提出請求。如需提出 FAS 請求時的政策詳細資訊，請參閱轉發存取工作階段。

Amazon DocumentDB 的服務角色

支援服務角色：是

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

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

Amazon DocumentDB 的服務連結角色

> **Note**
>
> 執行個體型叢集不支援服務連結角色，但彈性叢集則支援 。

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

如需建立或管理服務連結角色的詳細資訊，請參閱可搭配 IAM 運作的AWS 服務。在資料表中尋找服務，其中包含服務連結角色欄中的 Yes。選擇是連結，以檢視該服務的服務連結角色文件。

Amazon DocumentDB 的身分型政策範例

根據預設，使用者和角色沒有建立或修改 Amazon DocumentDB 資源的許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。

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

如需 Amazon DocumentDB 定義的動作和資源類型的詳細資訊，包括每種資源類型的 ARNs 格式，請參閱《服務授權參考》中的 Amazon Relational Database Service 的動作、資源和條件索引鍵。

> **Note**
>
> 對於某些管理功能，Amazon DocumentDB 使用與 Amazon Relational Database Service (Amazon RDS) 共享的操作技術。
> 如需 Amazon DocumentDB 彈性叢集的政策動作，請參閱《服務授權參考》中的 Amazon DocumentDB 彈性叢集的動作、資源和條件索引鍵。

主題

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

政策最佳實務

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

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予您的使用者和工作負載，請使用將許可授予許多常見使用案例的 AWS 受管政策。它們可在您的 中使用 AWS 帳戶。我們建議您定義特定於使用案例 AWS 的客戶受管政策，以進一步減少許可。如需更多資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策或任務職能的AWS 受管政策。
- 套用最低權限許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱《IAM 使用者指南》中的 IAM 中的政策和許可。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。如果透過特定 例如 使用服務動作 AWS 服務，您也可以使用條件來授予其存取權 CloudFormation。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您撰寫安全且實用的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM Access Analyzer 驗證政策。
- 需要多重要素驗證 (MFA) – 如果您的案例需要 IAM 使用者或 中的根使用者 AWS 帳戶，請開啟 MFA 以提高安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的透過 MFA 的安全 API 存取。

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

使用 Amazon DocumentDB 主控台

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

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

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

允許使用者檢視他們自己的許可

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

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
```

```
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

對 Amazon DocumentDB 身分和存取進行故障診斷

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

主題

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

我無權在 Amazon DocumentDB 中執行動作

如果您收到錯誤，告知您未獲授權執行動作，您的政策必須更新，允許您執行動作。

下列範例錯誤會在mateojackson IAM 使用者嘗試使用主控台檢視一個虛構 *my-example-widget* 資源的詳細資訊，但卻無虛構 aws:*GetWidget* 許可時發生。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 aws:GetWidget on resource: my-example-widget
```

在此情況下，必須更新 mateojackson 使用者的政策，允許使用 aws:*GetWidget* 動作存取 *my-example-widget* 資源。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我未獲得執行 iam:PassRole 的授權

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

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

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

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

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

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我想要允許 以外的人員 AWS 帳戶 存取我的 Amazon DocumentDB 資源

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

如需進一步了解，請參閱以下內容：

- 若要了解 Amazon DocumentDB 是否支援這些功能，請參閱 Amazon DocumentDB 如何與 IAM 搭配使用。
- 若要了解如何 AWS 帳戶 在您擁有的 資源之間提供存取權，請參閱《IAM 使用者指南》中的在您擁有 AWS 帳戶 的另一個 IAM 使用者中提供存取權。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的將存取權提供給第三方 AWS 帳戶 擁有。
- 如需了解如何透過聯合身分提供存取權，請參閱《IAM 使用者指南》中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。

管理 Amazon DocumentDB 資源的存取許可

每個 AWS 資源都由 擁有 AWS 帳戶，而建立或存取資源的許可是由許可政策管理。帳戶管理員可以將許可政策連接到 IAM 身分（即使用者、群組和角色），某些服務（例如 AWS Lambda) 也支援將許可政策連接到 資源。

> **Note**
>
> 帳戶管理員 (或管理員使用者) 是具有管理員許可的使用者。如需詳細資訊，請參《IAM 使用者指南》中的 IAM 最佳實務。

主題

- Amazon DocumentDB 資源和操作
- 了解資源所有權
- 管理 資源的存取
- 指定政策元素：動作、效果、資源和主體
- 在政策中指定條件

Amazon DocumentDB 資源和操作

在 Amazon DocumentDB 中，主要資源是叢集。Amazon DocumentDB 支援可與主要資源搭配使用的其他資源，例如執行個體、參數群組和事件訂閱。這些資源稱為「子資源」。

這些資源和子資源都有獨一無二的 Amazon Resource Name (ARN) 與其相關聯，如下表所示。

資源類型	ARN 格式
叢集	`arn:aws:rds:` *`region`*`:`*`account-id`* `:cluster:` *`db-cluster-name`*
叢集參數群組	`arn:aws:rds:` *`region`*`:`*`account-id`* `:cluster-pg:` *`cluster-parameter-group-name`*
叢集快照	`arn:aws:rds:` *`region`*`:`*`account-id`* `:cluster-snapshot:` *`cluster-snapshot-name`*

資源類型	ARN 格式
執行個體	arn:aws:rds: *region*:*account-id* :db:*db-instance-name*
安全群組	arn:aws:rds: *region*:*account-id* :secgrp:*security-group-name*
子網路群組	arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*

Amazon DocumentDB 提供一組操作來使用 Amazon DocumentDB 資源。如需可用操作的清單，請參閱動作。

了解資源所有權

資源擁有者是建立資源 AWS 帳戶 的 。也就是說，資源擁有者是驗證建立資源之請求 AWS 帳戶 的委託人實體 （ 根帳戶、IAM 使用者或 IAM 角色 ） 的 。下列範例說明其如何運作：

- 如果您使用 的根帳戶登入 AWS 帳戶 資料來建立 Amazon DocumentDB 資源，例如 執行個體，則您的 AWS 帳戶 是 Amazon DocumentDB 資源的擁有者。
- 如果您在 中建立 IAM 使用者， AWS 帳戶 並將建立 Amazon DocumentDB 資源的許可授予該使用者，則使用者可以建立 Amazon DocumentDB 資源。不過， AWS 帳戶使用者所屬的 擁有 Amazon DocumentDB 資源。
- 如果您在 中建立 AWS 帳戶 具有建立 Amazon DocumentDB 資源許可的 IAM 角色，則任何可以擔任該角色的人都可以建立 Amazon DocumentDB 資源。 AWS 帳戶角色所屬的 擁有 Amazon DocumentDB 資源。

管理 資源的存取

許可政策描述誰可以存取哪些資源。下一節說明可用來建立許可政策的選項。

> **Note**
>
> 本節討論在 Amazon DocumentDB 的內容中使用 IAM。它不提供 IAM 服務的詳細資訊。如需完整的 IAM 文件，請參閱IAM 使用者指南中的什麼是 IAM。如需有關 IAM 政策語法和說明的資訊，請參閱IAM 使用者指南中的 AWS IAM 政策參考。

連接至 IAM 身分的政策稱為身分識別型政策 (IAM 政策)。連接到資源的政策稱為「資源類型」政策。Amazon DocumentDB 僅支援以身分為基礎的政策 (IAM 政策) 。

主題

- 身分類型政策 (IAM 政策)
- 資源型政策

身分類型政策 (IAM 政策)

您可以將政策連接到 IAM 身分。例如，您可以執行下列動作：

- 將許可政策連接至您帳戶中的使用者或群組 – 帳戶管理員可以使用與特定使用者相關聯的許可政策，授予該使用者建立 Amazon DocumentDB 資源的許可，例如 執行個體。
- 將許可政策連接至角色 (授予跨帳戶許可)：您可以將身分識別型許可政策連接至 IAM 角色，藉此授予跨帳戶許可。例如，管理員可以建立角色，將跨帳戶許可授予另一個 AWS 帳戶 或 AWS 服務，如下所示：
  1. 帳戶 A 管理員建立 IAM 角色，並將許可政策連接到可授與帳戶 A 中資源許可的角色。
  2. 帳戶 A 管理員將信任政策連接至該角色，識別帳戶 B 做為可擔任該角的委託人。
  3. 帳戶 B 管理員接著可以將擔任該角色的許可委派給帳戶 B 中的任何使用者。這樣做可讓帳戶 B 中的使用者在帳戶 A 中建立或存取資源。如果您想要將擔任該角色的許可授予服務，則信任政策中的委託人也可以是 AWS 服務 AWS 委託人。

  如需使用 IAM 來委派許可的相關資訊，請參閱《IAM 使用者指南》中的存取管理。

以下是允許 ID 為 的使用者為您的 123456789012建立執行個體的範例政策 AWS 帳戶。新的執行個體還須使用選項群組，以及以 default 開頭的資料庫參數群組，而且它還須使用 default 子網路群組。

JSON

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

```
            "Action": [
                "rds:CreateDBInstance"
            ],
            "Resource": [
                "arn:aws:rds:*:123456789012:db:test*",
                "arn:aws:rds:*:123456789012:pg:cluster-pg:default*",
                "arn:aws:rds:*:123456789012:subgrp:default"
            ]
        }
    ]
}
```

如需搭配 Amazon DocumentDB 使用身分型政策的詳細資訊，請參閱 針對 Amazon DocumentDB 使用身分型政策 (IAM 政策) 。如需使用者、群組、角色和許可的詳細資訊，請參閱《IAM 使用者指南》中的身分 (使用者、群組和角色)。

資源型政策

其他 服務，例如 Amazon Simple Storage Service (Amazon S3)，也支援以資源為基礎的許可政策。例如，您可以將政策連接至 Simple Storage Service (Amazon S3) 儲存貯體，以管理該儲存貯體的存取許可。Amazon DocumentDB 不支援以資源為基礎的政策。

指定政策元素：動作、效果、資源和主體

對於每個 Amazon DocumentDB 資源 （請參閱Amazon DocumentDB 資源和操作)，服務會定義一組 API 操作。如需詳細資訊，請參閱動作。若要授予這些 API 操作的許可，Amazon DocumentDB 會定義一組您可以在政策中指定的動作。執行一項 API 操作可能需要多個動作的許可。

以下是基本的政策元素：

- 資源 – 在政策中，您可以使用 Amazon Resource Name (ARN) 來識別要套用政策的資源。
- 動作：使用動作關鍵字識別您要允許或拒絕的資源操作。例如，`rds:DescribeDBInstances` 許可允許使用者執行 `DescribeDBInstances` 操作。
- 效果 - 您可以指定使用者要求特定動作時會有什麼效果；可為允許或拒絕。如果您未明確授予存取 (允許) 資源，則隱含地拒絕存取。您也可以明確拒絕資源存取，這樣做可確保使用者無法存取資源，即使不同政策授予存取也是一樣。
- 委託人：在以身分為基礎的政策 (IAM 政策) 中，政策所連接的使用者就是隱含委託人。對於資源型政策，您可以指定想要收到許可的使用者、帳戶、服務或其他實體 (僅適用於資源型政策)。Amazon DocumentDB 不支援以資源為基礎的政策。

如需進一步了解有關 IAM 政策語法和說明的詳細資訊，請參閱《IAM 使用者指南》 中的 AWS IAM 政策參考。

如需顯示所有 Amazon DocumentDB API 動作及其適用的資源的資料表，請參閱 Amazon DocumentDB API 許可：動作、資源和條件參考。

在政策中指定條件

當您授與許可時，您可以使用 IAM 政策語言指定政策生效時間的條件。例如，建議只在特定日期之後套用政策。如需使用政策語言指定條件的詳細資訊，請參閱IAM 使用者指南中的條件。

欲表示條件，您可以使用預先定義的條件金鑰。Amazon DocumentDB 沒有可在 IAM 政策中使用的服務特定內容金鑰。如需所有 服務可用的全域條件內容金鑰清單，請參閱《IAM 使用者指南》中的可用全域條件金鑰。

針對 Amazon DocumentDB 使用身分型政策 (IAM 政策)

> **Important**
>
> 對於某些管理功能，Amazon DocumentDB 使用與 Amazon RDS 共用的操作技術。Amazon DocumentDB 主控台 AWS CLI和 API 呼叫會記錄為對 Amazon RDS API 發出的呼叫。
> 首先檢閱簡介主題，其中說明管理 Amazon DocumentDB 資源存取權的基本概念和選項。如需詳細資訊，請參閱管理 Amazon DocumentDB 資源的存取許可。

這個主題提供以身分為基礎的政策範例，在該政策中帳戶管理員可以將許可政策連接至 IAM 身分 (即使用者、群組和角色)。

以下是 IAM 政策的範例。

JSON

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

```
                "rds:CreateDBInstance"
            ],
            "Resource": [
                "arn:aws:rds:*:123456789012:db:test*",
                "arn:aws:rds:*:123456789012:pg:cluster-pg:default*",
                "arn:aws:rds:*:123456789012:subgrp:default"
            ]
        }
    ]
}
```

政策包括單一陳述式，指定 IAM 使用者的下列許可：

- 此政策允許 IAM 使用者使用 CreateDBInstance 動作建立執行個體 （這也適用於 create-db-instance AWS CLI 操作和 AWS 管理主控台)。
- `Resource` 元素指定使用者可對資源或搭配資源執行動作。您可以使用 Amazon Resource Name (ARN) 來指定資源。此 ARN 包含資源所屬的服務名稱 (`rds`)、 AWS 區域 (* 表示此範例中的任何區域）、使用者帳戶號碼 (123456789012 是此範例中的使用者 ID)，以及資源類型。

  範例中的 `Resource` 元素對使用者的資源指定下列政策限制：

  - 新執行個體的執行個體識別碼必須以 `test` 開頭 (例如，`testCustomerData1`、`test-region2-data`)。
  - 新執行個體的參數群組必須以 `default` 開頭。
  - 新執行個體的子網路群組必須是 `default` 子網路群組。

此政策不指定 `Principal` 元素，因為您不會在以身分為基礎的政策中，指定取得許可的主體。當您將政策連接至使用者時，這名使用者即為隱含主體。當您將許可政策連接至 IAM 角色，該角色的信任政策中所識別的委託人即取得許可。

如需顯示所有 Amazon DocumentDB API 操作及其適用的資源的資料表，請參閱 Amazon DocumentDB API 許可：動作、資源和條件參考。

使用 Amazon DocumentDB 主控台所需的許可

若要讓使用者使用 Amazon DocumentDB 主控台，該使用者必須擁有一組最低許可。這些許可允許使用者描述其 的 Amazon DocumentDB 資源， AWS 帳戶 並提供其他相關資訊，包括 Amazon EC2 安全性和網路資訊。

如果您建立比最基本必要許可更嚴格的 IAM 政策，則對於採取該 IAM 政策的使用者而言，主控台就無法如預期運作。為了確保這些使用者仍然可以使用 Amazon DocumentDB 主控台，也請將 `AmazonDocDBConsoleFullAccess`受管政策連接至使用者，如中所述AWS Amazon DocumentDB 的 受管政策。

對於僅呼叫 AWS CLI 或 Amazon DocumentDB API 的使用者，您不需要允許最低主控台許可。

客戶管理政策範例

在本節中，您可以找到授予各種 Amazon DocumentDB 動作許可的使用者政策範例。當您使用 Amazon DocumentDB API 動作、 AWS SDKs或 時，這些政策即可運作 AWS CLI。當您使用主控台時，需要授予主控台特定的額外許可，這會在「使用 Amazon DocumentDB 主控台所需的許可」中予以討論。

對於某些管理功能，Amazon DocumentDB 使用與 Amazon Relational Database Service (Amazon RDS) 和 Amazon Neptune 共用的操作技術。

> **Note**
>
> 所有範例都使用美國東部 （維吉尼亞北部） 區域 (`us-east-1`)，並包含虛構的帳戶 IDs。

範例

- 範例 1：允許使用者對任何 Amazon DocumentDB 資源執行任何描述動作
- 範例 2：防止使用者刪除執行個體
- 範例 3：除非啟用儲存加密，否則防止使用者建立叢集

範例 1：允許使用者對任何 Amazon DocumentDB 資源執行任何描述動作

下列許可政策會授予使用者執行開頭為 `Describe` 之所有動作的許可。這些動作會顯示 Amazon DocumentDB 資源的相關資訊，例如 執行個體。`Resource` 元素中的萬用字元 (*) 表示允許帳戶擁有的所有 Amazon DocumentDB 資源執行動作。

JSON

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

```
            "Sid":"AllowRDSDescribe",
            "Effect":"Allow",
            "Action":"rds:Describe*",
            "Resource":"*"
         }
      ]
}
```

範例 2：防止使用者刪除執行個體

以下許可政策授予許可，以防止使用者刪除特定的執行個體。例如，您可能想拒絕給予任何非管理員的使用者刪除生產執行個體的能力。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyDelete1",
         "Effect":"Deny",
         "Action":"rds:DeleteDBInstance",
         "Resource":"arn:aws:rds:us-east-1:123456789012:db:my-db-instance"
      }
   ]
}
```

範例 3：除非啟用儲存加密，否則防止使用者建立叢集

除非啟用儲存體加密，否則下列許可政策會拒絕使用者建立 Amazon DocumentDB 叢集的許可。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "PreventUnencryptedDocumentDB",
         "Effect": "Deny",
```

```
            "Action": "RDS:CreateDBCluster",
            "Condition": {
            "Bool": {
            "rds:StorageEncrypted": "false"
        },
            "StringEquals": {
            "rds:DatabaseEngine": "docdb"
            }
        },
        "Resource": "*"
        }
    ]
}
```

AWS Amazon DocumentDB 的 受管政策

若要新增許可給使用者、群組和角色，使用 AWS 受管政策比自行撰寫政策更容易。建立 IAM 客戶受管政策需要時間和專業知識，而受管政策可為您的團隊提供其所需的許可。若要快速開始使用，您可以使用我們的 AWS 受管政策。這些政策涵蓋常見的使用案例，並可在您的 AWS 帳戶中使用。如需 AWS 受管政策的詳細資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策。

AWS 服務會維護和更新 AWS 受管政策。您無法變更 AWS 受管政策中的許可。服務偶爾會將其他許可新增至 AWS 受管政策，以支援新功能。此類型的更新會影響已連接政策的所有身分識別 (使用者、群組和角色)。當新功能啟動或新操作可用時，服務最有可能更新 AWS 受管政策。服務不會從 AWS 受管政策中移除許可，因此政策更新不會破壞您現有的許可。

此外， AWS 支援跨多個 服務之任務函數的受管政策。例如， `ViewOnlyAccess` AWS 受管政策提供許多 AWS 服務和資源的唯讀存取權。當服務啟動新功能時， 會為新操作和資源 AWS 新增唯讀許可。如需任務職能政策的清單和說明，請參閱 IAM 使用者指南中有關任務職能的AWS 受管政策。

下列 AWS 受管政策是 Amazon DocumentDB 特有的，您可以連接到您帳戶中的使用者：

- AmazonDocDBFullAccess – 授予根目錄所有 Amazon DocumentDB 資源的完整存取權 AWS 帳戶。
- AmazonDocDBReadOnlyAccess – 授予對根目錄所有 Amazon DocumentDB 資源的唯讀存取權 AWS 帳戶。
- AmazonDocDBConsoleFullAccess – 授予使用 管理 Amazon DocumentDB 和 Amazon DocumentDB 彈性叢集資源的完整存取權 AWS 管理主控台。
- AmazonDocDBElasticReadOnlyAccess – 授予對根目錄所有 Amazon DocumentDB 彈性叢集資源的唯讀存取權 AWS 帳戶。

- AmazonDocDBElasticFullAccess – 授予根目錄所有 Amazon DocumentDB 彈性叢集資源的完整存取權 AWS 帳戶。

AmazonDocDBFullAccess

此政策會授予管理許可，允許委託人完整存取所有 Amazon DocumentDB 動作。此政策中的許可分組如下：

- Amazon DocumentDB 許可允許所有 Amazon DocumentDB 動作。
- 此政策中的某些 Amazon EC2 許可需要驗證 API 請求中傳遞的資源。這是為了確保 Amazon DocumentDB 能夠成功地將資源與叢集搭配使用。此政策中的其餘 Amazon EC2 許可允許 Amazon DocumentDB 建立所需的 AWS 資源，讓您可以連線到叢集。
- API 呼叫期間會使用 Amazon DocumentDB 許可來驗證請求中傳遞的資源。Amazon DocumentDB 需要它們才能搭配 Amazon DocumentDB 叢集使用傳遞的金鑰。
- Amazon DocumentDB 需要 CloudWatch Logs，才能確保日誌交付目的地可連線，且適用於代理程式日誌。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "rds:AddRoleToDBCluster",
                "rds:AddSourceIdentifierToSubscription",
                "rds:AddTagsToResource",
                "rds:ApplyPendingMaintenanceAction",
                "rds:CopyDBClusterParameterGroup",
                "rds:CopyDBClusterSnapshot",
                "rds:CopyDBParameterGroup",
                "rds:CreateDBCluster",
                "rds:CreateDBClusterParameterGroup",
                "rds:CreateDBClusterSnapshot",
                "rds:CreateDBInstance",
                "rds:CreateDBParameterGroup",
                "rds:CreateDBSubnetGroup",
                "rds:CreateEventSubscription",
                "rds:DeleteDBCluster",
```

```
                "rds:DeleteDBClusterParameterGroup",
                "rds:DeleteDBClusterSnapshot",
                "rds:DeleteDBInstance",
                "rds:DeleteDBParameterGroup",
                "rds:DeleteDBSubnetGroup",
                "rds:DeleteEventSubscription",
                "rds:DescribeAccountAttributes",
                "rds:DescribeCertificates",
                "rds:DescribeDBClusterParameterGroups",
                "rds:DescribeDBClusterParameters",
                "rds:DescribeDBClusterSnapshotAttributes",
                "rds:DescribeDBClusterSnapshots",
                "rds:DescribeDBClusters",
                "rds:DescribeDBEngineVersions",
                "rds:DescribeDBInstances",
                "rds:DescribeDBLogFiles",
                "rds:DescribeDBParameterGroups",
                "rds:DescribeDBParameters",
                "rds:DescribeDBSecurityGroups",
                "rds:DescribeDBSubnetGroups",
                "rds:DescribeEngineDefaultClusterParameters",
                "rds:DescribeEngineDefaultParameters",
                "rds:DescribeEventCategories",
                "rds:DescribeEventSubscriptions",
                "rds:DescribeEvents",
                "rds:DescribeOptionGroups",
                "rds:DescribeOrderableDBInstanceOptions",
                "rds:DescribePendingMaintenanceActions",
                "rds:DescribeValidDBInstanceModifications",
                "rds:DownloadDBLogFilePortion",
                "rds:FailoverDBCluster",
                "rds:ListTagsForResource",
                "rds:ModifyDBCluster",
                "rds:ModifyDBClusterParameterGroup",
                "rds:ModifyDBClusterSnapshotAttribute",
                "rds:ModifyDBInstance",
                "rds:ModifyDBParameterGroup",
                "rds:ModifyDBSubnetGroup",
                "rds:ModifyEventSubscription",
                "rds:PromoteReadReplicaDBCluster",
                "rds:RebootDBInstance",
                "rds:RemoveRoleFromDBCluster",
                "rds:RemoveSourceIdentifierFromSubscription",
                "rds:RemoveTagsFromResource",
```

```
                "rds:ResetDBClusterParameterGroup",
                "rds:ResetDBParameterGroup",
                "rds:RestoreDBClusterFromSnapshot",
                "rds:RestoreDBClusterToPointInTime"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
        {
            "Action": [
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcs",
                "kms:ListAliases",
                "kms:ListKeyPolicies",
                "kms:ListKeys",
                "kms:ListRetirableGrants",
                "logs:DescribeLogStreams",
                "logs:GetLogEvents",
                "sns:ListSubscriptions",
                "sns:ListTopics",
                "sns:Publish"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
        {
            "Action": "iam:CreateServiceLinkedRole",
            "Effect": "Allow",
            "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
            "Condition": {
                "StringLike": {
                    "iam:AWS ServiceName": "rds.amazonaws.com"
                }
```

```
            }
        }
    ]
}
```

AmazonDocDBReadOnlyAccess

此政策授予唯讀許可，允許使用者檢視 Amazon DocumentDB 中的資訊。附加此政策的主體無法進行任何更新或刪除結束的資源，也無法建立新的 Amazon DocumentDB 資源。例如，具有這些許可的主體可以檢視與其帳戶相關聯的叢集和組態清單，但無法變更任何叢集的組態或設定。此政策中的許可分組如下：

- Amazon DocumentDB 許可可讓您列出 Amazon DocumentDB 資源、描述這些資源，以及取得這些資源的相關資訊。
- Amazon EC2 許可用於描述與叢集相關聯的 Amazon VPC、子網路、安全群組和 ENIs。
- Amazon DocumentDB 許可用於描述與叢集相關聯的金鑰。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "rds:DescribeAccountAttributes",
                "rds:DescribeCertificates",
                "rds:DescribeDBClusterParameterGroups",
                "rds:DescribeDBClusterParameters",
                "rds:DescribeDBClusterSnapshotAttributes",
                "rds:DescribeDBClusterSnapshots",
                "rds:DescribeDBClusters",
                "rds:DescribeDBEngineVersions",
                "rds:DescribeDBInstances",
                "rds:DescribeDBLogFiles",
                "rds:DescribeDBParameterGroups",
                "rds:DescribeDBParameters",
                "rds:DescribeDBSubnetGroups",
                "rds:DescribeEventCategories",
                "rds:DescribeEventSubscriptions",
```

```
                "rds:DescribeEvents",
                "rds:DescribeOrderableDBInstanceOptions",
                "rds:DescribePendingMaintenanceActions",
                "rds:DownloadDBLogFilePortion",
                "rds:ListTagsForResource"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeInternetGateways",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcs"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "kms:ListKeys",
                "kms:ListRetirableGrants",
                "kms:ListAliases",
                "kms:ListKeyPolicies"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "logs:DescribeLogStreams",
                "logs:GetLogEvents"
```

```
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:logs:*:*:log-group:/aws/rds/*:log-stream:*",
                "arn:aws:logs:*:*:log-group:/aws/docdb/*:log-stream:*"
            ]
        }
    ]
}
```

AmazonDocDBConsoleFullAccess

授予完整存取權，以使用 管理 Amazon DocumentDB 資源 AWS 管理主控台 ，如下所示：

- 允許所有 Amazon DocumentDB和 Amazon DocumentDB 叢集動作的 Amazon DocumentDB 許可。
- 此政策中的某些 Amazon EC2 許可需要驗證 API 請求中傳遞的資源。這是為了確保 Amazon DocumentDB 能夠成功使用 資源來佈建和維護叢集。此政策中的其餘 Amazon EC2 許可允許 Amazon DocumentDB 建立所需的 AWS 資源，讓您可以連線到 VPCEndpoint 等叢集。
- AWS KMS 許可會在 API 呼叫 時使用， AWS KMS 以驗證請求中傳遞的資源。Amazon DocumentDB 需要使用它們，才能使用傳遞的金鑰，透過 Amazon DocumentDB 彈性叢集來加密和解密靜態資料。
- Amazon DocumentDB 需要 CloudWatch Logs，才能確保日誌交付目的地可連線，且適用於稽核和分析日誌。
- 需要 Secrets Manager 許可才能驗證指定的秘密，並使用它來設定 Amazon DocumentDB 彈性叢集的管理員使用者。
- Amazon DocumentDB 叢集管理動作需要 Amazon RDS 許可。對於某些管理功能，Amazon DocumentDB 使用與 Amazon RDS 共用的操作技術。
- SNS 許可允許主體使用 Amazon Simple Notification Service (Amazon SNS) 訂閱和主題，以及發佈 Amazon SNS 訊息。
- 建立指標和日誌發佈所需的服務連結角色時，需要 IAM 許可。

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "DocdbSids",
            "Effect": "Allow",
            "Action": [
                "docdb-elastic:CreateCluster",
                "docdb-elastic:UpdateCluster",
                "docdb-elastic:GetCluster",
                "docdb-elastic:DeleteCluster",
                "docdb-elastic:ListClusters",
                "docdb-elastic:CreateClusterSnapshot",
                "docdb-elastic:GetClusterSnapshot",
                "docdb-elastic:DeleteClusterSnapshot",
                "docdb-elastic:ListClusterSnapshots",
                "docdb-elastic:RestoreClusterFromSnapshot",
                "docdb-elastic:TagResource",
                "docdb-elastic:UntagResource",
                "docdb-elastic:ListTagsForResource",
                "docdb-elastic:CopyClusterSnapshot",
                "docdb-elastic:StartCluster",
                "docdb-elastic:StopCluster",
                "docdb-elastic:GetPendingMaintenanceAction",
                "docdb-elastic:ListPendingMaintenanceActions",
                "docdb-elastic:ApplyPendingMaintenanceAction",
                "rds:AddRoleToDBCluster",
                "rds:AddSourceIdentifierToSubscription",
                "rds:AddTagsToResource",
                "rds:ApplyPendingMaintenanceAction",
                "rds:CopyDBClusterParameterGroup",
                "rds:CopyDBClusterSnapshot",
                "rds:CopyDBParameterGroup",
                "rds:CreateDBCluster",
                "rds:CreateDBClusterParameterGroup",
                "rds:CreateDBClusterSnapshot",
                "rds:CreateDBInstance",
                "rds:CreateDBParameterGroup",
                "rds:CreateDBSubnetGroup",
                "rds:CreateEventSubscription",
                "rds:CreateGlobalCluster",
                "rds:DeleteDBCluster",
                "rds:DeleteDBClusterParameterGroup",
                "rds:DeleteDBClusterSnapshot",
                "rds:DeleteDBInstance",
                "rds:DeleteDBParameterGroup",
```

```
                "rds:DeleteDBSubnetGroup",
                "rds:DeleteEventSubscription",
                "rds:DeleteGlobalCluster",
                "rds:DescribeAccountAttributes",
                "rds:DescribeCertificates",
                "rds:DescribeDBClusterParameterGroups",
                "rds:DescribeDBClusterParameters",
                "rds:DescribeDBClusterSnapshotAttributes",
                "rds:DescribeDBClusterSnapshots",
                "rds:DescribeDBClusters",
                "rds:DescribeDBEngineVersions",
                "rds:DescribeDBInstances",
                "rds:DescribeDBLogFiles",
                "rds:DescribeDBParameterGroups",
                "rds:DescribeDBParameters",
                "rds:DescribeDBSecurityGroups",
                "rds:DescribeDBSubnetGroups",
                "rds:DescribeEngineDefaultClusterParameters",
                "rds:DescribeEngineDefaultParameters",
                "rds:DescribeEventCategories",
                "rds:DescribeEventSubscriptions",
                "rds:DescribeEvents",
                "rds:DescribeGlobalClusters",
                "rds:DescribeOptionGroups",
                "rds:DescribeOrderableDBInstanceOptions",
                "rds:DescribePendingMaintenanceActions",
                "rds:DescribeValidDBInstanceModifications",
                "rds:DownloadDBLogFilePortion",
                "rds:FailoverDBCluster",
                "rds:ListTagsForResource",
                "rds:ModifyDBCluster",
                "rds:ModifyDBClusterParameterGroup",
                "rds:ModifyDBClusterSnapshotAttribute",
                "rds:ModifyDBInstance",
                "rds:ModifyDBParameterGroup",
                "rds:ModifyDBSubnetGroup",
                "rds:ModifyEventSubscription",
                "rds:ModifyGlobalCluster",
                "rds:PromoteReadReplicaDBCluster",
                "rds:RebootDBInstance",
                "rds:RemoveFromGlobalCluster",
                "rds:RemoveRoleFromDBCluster",
                "rds:RemoveSourceIdentifierFromSubscription",
                "rds:RemoveTagsFromResource",
```

```
                "rds:ResetDBClusterParameterGroup",
                "rds:ResetDBParameterGroup",
                "rds:RestoreDBClusterFromSnapshot",
                "rds:RestoreDBClusterToPointInTime"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "DependencySids",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics",
                "ec2:AllocateAddress",
                "ec2:AssignIpv6Addresses",
                "ec2:AssignPrivateIpAddresses",
                "ec2:AssociateAddress",
                "ec2:AssociateRouteTable",
                "ec2:AssociateSubnetCidrBlock",
                "ec2:AssociateVpcCidrBlock",
                "ec2:AttachInternetGateway",
                "ec2:AttachNetworkInterface",
                "ec2:CreateCustomerGateway",
                "ec2:CreateDefaultSubnet",
                "ec2:CreateDefaultVpc",
                "ec2:CreateInternetGateway",
                "ec2:CreateNatGateway",
                "ec2:CreateNetworkInterface",
                "ec2:CreateRoute",
                "ec2:CreateRouteTable",
                "ec2:CreateSecurityGroup",
                "ec2:CreateSubnet",
                "ec2:CreateVpc",
                "ec2:CreateVpcEndpoint",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAddresses",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeCustomerGateways",
                "ec2:DescribeInstances",
                "ec2:DescribeNatGateways",
```

```
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribePrefixLists",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroupReferences",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeVpcs",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:ModifySubnetAttribute",
                "ec2:ModifyVpcAttribute",
                "ec2:ModifyVpcEndpoint",
                "kms:DescribeKey",
                "kms:ListAliases",
                "kms:ListKeyPolicies",
                "kms:ListKeys",
                "kms:ListRetirableGrants",
                "logs:DescribeLogStreams",
                "logs:GetLogEvents",
                "sns:ListSubscriptions",
                "sns:ListTopics",
                "sns:Publish"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "DocdbSLRSid",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "rds.amazonaws.com"
                }
            }
        },
        {
            "Sid": "DocdbElasticSLRSid",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
```

```
            "Resource": "arn:aws:iam::*:role/aws-service-role/docdb-
elastic.amazonaws.com/AWSServiceRoleForDocDB-Elastic",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "docdb-elastic.amazonaws.com"
                }
            }
        }
    ]
}
```

AmazonDocDBElasticReadOnlyAccess

此政策授予唯讀許可，允許使用者在 Amazon DocumentDB 中檢視彈性叢集資訊。附加此政策的主體無法進行任何更新或刪除結束的資源，也無法建立新的 Amazon DocumentDB 資源。例如，具有這些許可的主體可以檢視與其帳戶相關聯的叢集和組態清單，但無法變更任何叢集的組態或設定。此政策中的許可分組如下：

- Amazon DocumentDB 彈性叢集許可可讓您列出 Amazon DocumentDB 彈性叢集資源、加以描述，以及取得相關資訊。
- CloudWatch 許可用於驗證服務指標。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "docdb-elastic:ListClusters",
            "docdb-elastic:GetCluster",
            "docdb-elastic:ListClusterSnapshots",
            "docdb-elastic:GetClusterSnapshot",
            "docdb-elastic:ListTagsForResource"
         ],
         "Resource": "*"
      },
      {
         "Effect": "Allow",
```

```
            "Action": [
               "cloudwatch:GetMetricData",
               "cloudwatch:ListMetrics",
               "cloudwatch:GetMetricStatistics"
            ],
            "Resource": "*"
         }
      ]
}
```

AmazonDocDBElasticFullAccess

此政策授予管理許可，允許主體完整存取 Amazon DocumentDB 彈性叢集的所有 Amazon DocumentDB 動作。

此政策在條件中使用 AWS 標籤 (https://docs.aws.amazon.com/tag-editor/latest/userguide/tagging.html：//) 來限制對 資源的存取。如果您使用的是秘密，則必須使用標籤索引鍵`DocDBElasticFullAccess`和標籤值來標記秘密。如果您使用客戶受管金鑰，則必須使用標籤金鑰`DocDBElasticFullAccess`和標籤值來標記。

此政策中的許可分組如下：

- Amazon DocumentDB 彈性叢集許可允許所有 Amazon DocumentDB 動作。
- 此政策中的某些 Amazon EC2 許可需要驗證 API 請求中傳遞的資源。這是為了確保 Amazon DocumentDB 能夠成功使用 資源來佈建和維護叢集。此政策中的其餘 Amazon EC2 許可允許 Amazon DocumentDB 建立所需的 AWS 資源，讓您可以像 VPC 端點一樣連線到叢集。
- AWS KMS 需要 許可，Amazon DocumentDB 才能使用傳遞的金鑰來加密和解密 Amazon DocumentDB 彈性叢集內的靜態資料。

  > **Note**
  >
  > 客戶受管金鑰必須具有具有金鑰`DocDBElasticFullAccess`和標籤值的標籤。

- 需要 SecretsManager 許可才能驗證指定的秘密，並使用它來設定 Amazon DocumentDB 彈性叢集的管理員使用者。

> **Note**
>
> 使用的秘密必須具有具有金鑰DocDBElasticFullAccess和標籤值的標籤。

- 建立指標和日誌發佈所需的服務連結角色時，需要 IAM 許可。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DocdbElasticSid",
            "Effect": "Allow",
            "Action": [
                "docdb-elastic:CreateCluster",
                "docdb-elastic:UpdateCluster",
                "docdb-elastic:GetCluster",
                "docdb-elastic:DeleteCluster",
                "docdb-elastic:ListClusters",
                "docdb-elastic:CreateClusterSnapshot",
                "docdb-elastic:GetClusterSnapshot",
                "docdb-elastic:DeleteClusterSnapshot",
                "docdb-elastic:ListClusterSnapshots",
                "docdb-elastic:RestoreClusterFromSnapshot",
                "docdb-elastic:TagResource",
                "docdb-elastic:UntagResource",
                "docdb-elastic:ListTagsForResource",
                "docdb-elastic:CopyClusterSnapshot",
                "docdb-elastic:StartCluster",
                "docdb-elastic:StopCluster",
                "docdb-elastic:GetPendingMaintenanceAction",
                "docdb-elastic:ListPendingMaintenanceActions",
                "docdb-elastic:ApplyPendingMaintenanceAction"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "EC2Sid",
```

```
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVpcEndpoint",
                "ec2:DescribeVpcEndpoints",
                "ec2:DeleteVpcEndpoints",
                "ec2:ModifyVpcEndpoint",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:DescribeAvailabilityZones",
                "secretsmanager:ListSecrets"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:CalledViaFirst": "docdb-elastic.amazonaws.com"
                }
            }
        },
        {
            "Sid": "KMSSid",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": [
                        "docdb-elastic.*.amazonaws.com"
                    ],
                    "aws:ResourceTag/DocDBElasticFullAccess": "*"
                }
            }
        },
        {
            "Sid": "KMSGrantSid",
            "Effect": "Allow",
            "Action": [
```

```
                "kms:CreateGrant"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/DocDBElasticFullAccess": "*",
                    "kms:ViaService": [
                        "docdb-elastic.*.amazonaws.com"
                    ]
                },
                "Bool": {
                    "kms:GrantIsForAWSResource": true
                }
            }
        },
        {
            "Sid": "SecretManagerSid",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:ListSecretVersionIds",
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:GetResourcePolicy"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "secretsmanager:ResourceTag/DocDBElasticFullAccess": "*"
                },
                "StringEquals": {
                    "aws:CalledViaFirst": "docdb-elastic.amazonaws.com"
                }
            }
        },
        {
            "Sid": "CloudwatchSid",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:ListMetrics",
                "cloudwatch:GetMetricStatistics"
            ],
            "Resource": [
                "*"
```

```
            ]
        },
        {
            "Sid": "SLRSid",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/docdb-
elastic.amazonaws.com/AWSServiceRoleForDocDB-Elastic",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "docdb-elastic.amazonaws.com"
                }
            }
        }
    ]
}
```

AmazonDocDB-ElasticServiceRolePolicy

您無法AmazonDocDBElasticServiceRolePolicy連接至 AWS Identity and Access Management 實體。此政策會連接到服務連結角色，允許 Amazon DocumentDB 代表您執行動作。如需詳細資訊，請參閱彈性叢集中的服務連結角色。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": [
                        "AWS/DocDB-Elastic"
                    ]
                }
            }
```

```
        }
    ]
}
```

AWS 受管政策的 Amazon DocumentDB 更新

變更	描述	日期
AmazonDocDBElasticFullAccess、 AmazonDocDBConsoleFullAccess - 變更	已更新政策以新增待定的維護動作。	2/11/2025
AmazonDocDBElasticFullAccess、 AmazonDocDBConsoleFullAccess - 變更	已更新政策以新增開始/停止叢集和複製叢集快照動作。	2/21/2024
AmazonDocDBElasticReadOnlyAccess、 AmazonDocDBElasticFullAccess - 變更	已更新政策以新增`cloudwatch:GetMetricData` 動作。	6/21/2023
AmazonDocDBElasticReadOnlyAccess – 新政策	Amazon DocumentDB 彈性叢集的新受管政策。	6/8/2023
AmazonDocDBElasticFullAccess – 新政策	Amazon DocumentDB 彈性叢集的新受管政策。	6/5/2023
AmazonDocDB-ElasticServiceRolePolicy – 新政策	Amazon DocumentDB 會為 Amazon DocumentDB 彈性叢集建立新的 AWS ServiceRoleForDocDB-Elastic 服務連結角色。	11/30/2022
AmazonDocDBConsoleFullAccess - 變更	已更新政策以新增 Amazon DocumentDB 全域和彈性叢集許可。	11/30/2022

變更	描述	日期
AmazonDocDBConsoleFullAccess、AmazonDocDBFullAccess、 AmazonDocDBReadOnlyAccess - 新政策	服務啟動。	1/19/2017

Amazon DocumentDB API 許可：動作、資源和條件參考

當您設定針對 Amazon DocumentDB 使用身分型政策 (IAM 政策) 和寫入可連接到 IAM 身分 （身分型政策） 的許可政策時，請使用下列各節做為參考。

以下列出每個 Amazon DocumentDB API 操作。清單中包含的對應動作是您可以授予執行動作的許可、您可以授予許可 AWS 的資源，以及您可以包含用於精細存取控制的條件索引鍵。您可以在政策的 `Action` 欄位中指定動作、在政策的 `Resource` 欄位中指定資源值，以及在政策的 `Condition` 欄位中指定條件。如需條件的相關資訊，請參閱在政策中指定條件。

您可以在 Amazon DocumentDB 政策中使用 AWS整體條件金鑰來表達條件。如需 AWS全系列金鑰的完整清單，請參閱《IAM 使用者指南》中的可用金鑰。

您可以使用 IAM 政策模擬器來測試 IAM 政策。它會自動提供每個 AWS 動作所需的資源和參數清單，包括 Amazon DocumentDB 動作。IAM 政策模擬器會決定您指定的每個動作所需的許可。如需 IAM 政策模擬器的相關資訊，請參閱《IAM 使用者指南》中的使用 IAM 政策模擬器測試 IAM 政策。

> **Note**
>
> 若要指定動作，請使用後接 API 操作名稱的 `rds:` 字首 (例如，`rds:CreateDBInstance`)。

以下列出 Amazon RDS API 操作及其相關動作、資源和條件索引鍵。

主題

- 支援資源層級許可的 Amazon DocumentDB 動作
- 不支援資源層級許可的 Amazon DocumentDB 動作

支援資源層級許可的 Amazon DocumentDB 動作

資源層級許可可讓您指定允許使用者執行動作的資源。Amazon DocumentDB 部分支援資源層級許可。這表示對於某些 Amazon DocumentDB 動作，您可以根據必須滿足的條件，或允許使用者使用的特定資源，控制何時允許使用者使用這些動作。例如，您可以授予使用者只修改特定執行個體的許可。

以下列出 Amazon DocumentDB API 操作及其相關動作、資源和條件索引鍵。

> **Note**
>
> 對於某些管理功能，Amazon DocumentDB 使用與 Amazon RDS 共用的操作技術。如需更多 Amazon DocumentDB 動作和許可，請參閱《服務授權參考》中的 Amazon RDS 的動作、資源和條件索引鍵。

Amazon DocumentDB API 操作和動作	Resources	條件金鑰
AddTagsToResource `rds:AddTagsToResource`	執行個體 `arn:aws:rds:` *`region`*`:`*`account-id`*` :db:`*`db-instance-name`*	`rds:db-tag`
	子網路群組 `arn:aws:rds:` *`region`*`:`*`account-id`*` :subgrp:`*`subnet-group-name`*	`rds:subgrp-tag`
ApplyPendingMaintenanceAction `rds:ApplyPendingMaintenanceAction`	執行個體 `arn:aws:rds:` *`region`*`:`*`account-id`*` :db:`*`db-instance-name`*	`rds:db-tag`
CopyDBClusterSnapshot	叢集快照	`rds:cluster-snapshot-tag`

Amazon DocumentDB API 操作和動作	Resources	條件金鑰
rds:CopyDBClusterSnapshot	arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	
CreateDBCluster rds:CreateDBCluster	叢集 arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	rds:cluster-tag
	叢集參數群組 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
	子網路群組 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
CreateDBClusterParameterGroup rds:CreateDBClusterParameterGroup	叢集參數群組 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
CreateDBClusterSnapshot rds:CreateDBClusterSnapshot	叢集 arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	rds:cluster-tag

Amazon DocumentDB API 操作和動作	Resources	條件金鑰
	叢集快照 arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
CreateDBInstance rds:CreateDBInstance	執行個體 arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:DatabaseClass rds:db-tag
	叢集 arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	rds:cluster-tag
CreateDBSubnetGroup rds:CreateDBSubnetGroup	子網路群組 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
DeleteDBInstance rds:DeleteDBInstance	執行個體 arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
DeleteDBSubnetGroup rds:DeleteDBSubnetGroup	子網路群組 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag

Amazon DocumentDB API 操作和動作	Resources	條件金鑰
DescribeDBClusterParameterGroups rds:DescribeDBClusterParameterGroups	叢集參數群組 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
DescribeDBClusterParameters rds:DescribeDBClusterParameters	叢集參數群組 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
DescribeDBClusters rds:DescribeDBClusters	叢集 arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
DescribeDBClusterSnapshotAttributes rds:DescribeDBClusterSnapshotAttributes	叢集快照 arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
DescribeDBSubnetGroups rds:DescribeDBSubnetGroups	子網路群組 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag

Amazon DocumentDB API 操作和動作	Resources	條件金鑰
DescribePendingMaintenanceActions rds:DescribePendingMaintenanceActions	執行個體 arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:DatabaseClass rds:db-tag
FailoverDBCluster rds:FailoverDBCluster	叢集 arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
ListTagsForResource rds:ListTagsForResource	執行個體 arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
	子網路群組 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
ModifyDBCluster rds:ModifyDBCluster	叢集 arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	rds:cluster-tag
	叢集參數群組 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag

Amazon DocumentDB API 操作和動作	Resources	條件金鑰
ModifyDBClusterParameterGroup rds:ModifyDBClusterParameterGroup	叢集參數群組 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
ModifyDBClusterSnapshotAttribute rds:ModifyDBClusterSnapshotAttribute	叢集快照 arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
ModifyDBInstance rds:ModifyDBInstance	執行個體 arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:DatabaseClass rds:db-tag
RebootDBInstance rds:RebootDBInstance	執行個體 arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
RemoveTagsFromResource rds:RemoveTagsFromResource	執行個體 arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
	子網路群組 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag

Amazon DocumentDB API 操作和動作	Resources	條件金鑰
ResetDBClusterParameterGroup rds:ResetDBClusterParameterGroup	叢集參數群組 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
RestoreDBClusterFromSnapshot rds:RestoreDBClusterFromSnapshot	叢集 arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
	叢集快照 arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
RestoreDBClusterToPointInTime rds:RestoreDBClusterToPointInTime	叢集 arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
	子網路群組 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag

不支援資源層級許可的 Amazon DocumentDB 動作

您可以使用 IAM 政策中的所有 Amazon DocumentDB 動作來授予或拒絕使用者使用該動作的許可。不過，並非所有 Amazon DocumentDB 動作都支援資源層級許可，這可讓您指定可執行動作的資源。

下列 Amazon DocumentDB API 動作目前不支援資源層級許可。因此，若要在 IAM 政策中使用這些動作，您必須為陳述式中的 Resource 元素使用*萬用字元，授予使用者使用動作所有資源的許可。

- rds:DescribeDBClusterSnapshots
- rds:DescribeDBInstances

預防跨服務混淆代理人

混淆代理人問題屬於安全性問題，其中沒有執行動作許可的實體可以強制具有更多許可的實體執行該動作。在 中 AWS，跨服務模擬可能會導致混淆代理人問題。在某個服務 (呼叫服務) 呼叫另一個服務 (被呼叫服務) 時，可能會發生跨服務模擬。可以操縱呼叫服務來使用其許可，以其不應有存取許可的方式對其他客戶的資源採取動作。為了預防這種情況， AWS 提供的工具可協助您保護所有服務的資料，而這些服務主體已獲得您帳戶中資源的存取權。

在資源政策中使用 aws:SourceArn和 aws:SourceAccount全域條件內容索引鍵，以限制 Amazon DocumentDB 為資源提供其他服務的許可。如果您想要僅允許一個資源與跨服務存取相關聯，則請使用 aws:SourceArn。如果您想要允許該帳戶中的任何資源與跨服務使用相關聯，請使用 aws:SourceAccount。

防範混淆代理人問題的最有效方法是使用 aws:SourceArn 全域條件內容索引鍵，以及資源的完整 ARN。如果不知道資源的完整 ARN，或者如果您指定了多個資源，請使用 aws:SourceArn 全域內容條件索引鍵搭配萬用字元 (*) 來表示 ARN 的未知部分。例如 arn:aws:*servicename*:*:*123456789012*:*。

如果 aws:SourceArn 值不包含帳戶 ID (例如 Amazon S3 儲存貯體 ARN)，您必須使用這兩個全域條件內容索引鍵來限制許可。

的值aws:SourceArn必須是叢集。

下列範例示範如何在 Amazon DocumentDB 中使用 aws:SourceArn和 aws:SourceAccount全域條件內容金鑰，以防止混淆代理人問題。

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "servicename.amazonaws.com"
```

```
    },
    "Action": "servicename:ActionName",
    "Resource": [
      "arn:aws:servicename:::ResourceName/*"
    ],
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:servicename:*:123456789012:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

使用 IAM 身分進行身分驗證

Amazon DocumentDB 使用者和應用程式可以使用 IAM 使用者和角色來驗證 Amazon DocumentDB 叢集。Amazon DocumentDB IAM 身分驗證是一種無密碼身分驗證方法。此外，使用 IAM 角色/使用者時，用戶端應用程式不會將密碼秘密傳送至 Amazon DocumentDB 叢集。反之，用戶端連線會使用 AWS STS 臨時安全字符進行身分驗證。非管理使用者和應用程式現在可以在連線至不同的 Amazon DocumentDB 叢集和其他 AWS 服務時，使用相同的 IAM 身分 ARN。

您也可以選擇同時使用密碼型和 IAM 身分驗證，對 Amazon DocumentDB 叢集的使用者和應用程式進行身分驗證。IAM 身分驗證僅適用於 Amazon DocumentDB 執行個體型叢集 5.0 版。Amazon DocumentDB 主要使用者不支援使用 IAM 身分 ARNs IAM 身分驗證。

> **Note**
>
> 主要使用者只能使用現有的密碼型身分驗證進行身分驗證。

主題

- 開始使用 IAM 使用者和角色進行身分驗證
- 設定 AWS 運算類型以使用 IAM AWS 驗證 Amazon DocumentDB
- 監控 IAM 身分驗證請求
- 使用 IAM 身分驗證

- 支援 IAM 的驅動程式
- IAM 身分驗證常見問答集

開始使用 IAM 使用者和角色進行身分驗證

具有 IAM 身分的 Amazon DocumentDB 使用者和角色會在 `$external` 資料庫中建立和管理。

建立使用者

以主要使用者身分連線，然後建立 IAM 使用者和角色：

```
use $external;
db.createUser(
    {
        user: "arn:aws:iam::123456789123:user/iamuser",
        mechanisms: ["MONGODB-AWS"],
        roles: [ { role: "readWrite", db: "readWriteDB" } ]
    }
);
```

或者，使用 IAM 角色新增 Amazon DocumentDB 使用者：

```
use $external;
db.createUser(
    {
        user: "arn:aws:iam::123456789123:role/iamrole",
        mechanisms: ["MONGODB-AWS"],
        roles: [ { role: "readWrite", db: "readWriteDB" } ]
    }
);
```

修改 IAM 使用者或角色

修改現有的 IAM 使用者：

```
use $external;
db.updateUser(
  "arn:aws:iam::123456789123:user/iamuser",
  {
    roles: [ { role: "read", db: "readDB" } ]
```

```
  }
);
```

修改現有的 IAM 角色：

```
use $external;
db.updateUser(
  "arn:aws:iam::123456789123:role/iamrole",
  {
    roles: [ { role: "read", db: "readDB" } ]
  }
);
```

若要授予或撤銷 IAM 使用者的角色：

```
use $external;
db.grantRolesToUser(
  "arn:aws:iam::123456789123:user/iamuser",
  [ { db: "admin", role: "readWriteAnyDatabase" } ]
);
```

```
use $external;
db.revokeRolesFromUser(
  "arn:aws:iam::123456789123:user/iamuser",
  [ { db: "admin", role: "readWriteAnyDatabase" } ]
);
```

若要從 IAM 角色授予或撤銷角色：

```
use $external;
db.grantRolesToUser(
  "arn:aws:iam::123456789123:user/iamrole",
  [ { db: "admin", role: "readWriteAnyDatabase" } ]
);
```

```
use $external;
db.revokeRolesFromUser(
  "arn:aws:iam::123456789123:user/iamrole",
  [ { db: "admin", role: "readWriteAnyDatabase" } ]
);
```

捨棄 IAM 使用者或角色

若要捨棄現有的 IAM 使用者：

```
use $external;
db.dropUser("arn:aws:iam::123456789123:user/iamuser");
```

若要捨棄現有的 IAM 角色：

```
use $external;
db.dropUser("arn:aws:iam::123456789123:role/iamrole");
```

設定連線 URI 以使用 IAM AWS 進行驗證

若要使用 AWS IAM 進行驗證，請使用下列 URI 參數： `authSource` 和 `$externalauthMechanism`做為 `MONGODB-AWS`。如果您使用的是 IAM 使用者，使用者名稱和密碼欄位會分別由存取金鑰和私密金鑰取代。如果您擔任 IAM 角色， 會連接到您所在的環境（例如，AWS Lambda function、Amazon EC2 執行個體）。使用 `MONGODB-AWS`機制進行身分驗證時，您不需要特別傳遞任何登入資料。如果您使用的是支援`MONGODB-AWS`身分驗證機制的 MongoDB 驅動程式，則驅動程式也能夠從運算執行個體擷取 IAM 角色憑證（例如 Amazon EC2、Lambda 函數等）。下列範例使用 mongo shell，`MONGODB-AWS`透過手動傳遞存取金鑰和私密金鑰 (IAM 使用者的） 來使用 進行身分驗證，以示範對 Amazon DocumentDB 的身分驗證。

下列範例使用 Python 程式碼使用 進行身分驗證，`MONGODB-AWS`而不明確傳遞任何登入資料（使用連接到環境的 IAM 角色），以示範對 Amazon DocumentDB 的身分驗證。

```
##Create a MongoDB client, open a connection to Amazon DocumentDB using an IAM role
    client = pymongo.MongoClient('mongodb://<DocDBEndpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false&authSource=
%24external&authMechanism=MONGODB-AWS')
```

下列範例使用 mongo shell，透過手動傳遞存取金鑰和私密金鑰 (IAM 使用者的） 來使用`MONGODB-AWS`機制進行身分驗證，以示範對 Amazon DocumentDB 的身分驗證。

```
$ mongo 'mongodb://<access_key>:<secret_key>@<cluster_endpoint>:<db_port>/test?
authSource=%24external&authMechanism=MONGODB-AWS'
```

下列範例使用 mongo shell 來驗證 ，`MONGODB-AWS`而不明確傳遞任何登入資料（使用連接到環境的 IAM 角色），以示範對 Amazon DocumentDB 的身分驗證。

```
$ mongo 'mongodb://<cluster_endpoint>:<db_port>/test?authSource=
%24external&authMechanism=MONGODB-AWS'
```

設定 AWS 運算類型以使用 IAM AWS 驗證 Amazon DocumentDB

使用 Amazon EC2/AWS Lambda/AWS Fargate

Amazon EC2 使用以下環境變數。如果您有連接到 EC2 執行個體的 IAM 角色，或與 Lambda 函數或 Amazon ECS 任務相關聯的執行 IAM 角色，則會自動填入這些變數，而且驅動程式可以從環境擷取這些值：

```
AWS_ACCESS_KEY_ID
AWS_SECRET_ACCESS_KEY
AWS_SESSION_TOKEN
```

如需環境變數的詳細資訊，請參閱《 AWS Lambda 開發人員指南》中的使用 Lambda 環境變數。

使用 Amazon EKS

將角色指派給 Amazon Elastic Kubernetes Service (Amazon EKS) Pod 會自動設定下列兩個環境變數：

```
AWS_WEB_IDENTITY_TOKEN_FILE - path of web identity token file
AWS_ROLE_ARN - Name of IAM role to connect with
```

透過這些變數的協助，使用適用於 的 AWS SDK 呼叫，從程式碼手動擔任角色`AssumeRoleWithWebIdentity`：

- 省略 `ProviderID` 參數。
- 在`AWS_WEB_IDENTITY_TOKEN_FILE`環境變數中所述的檔案中尋找 `WebIdentityToken` 參數的值。

如需 Amazon EKS 的詳細資訊，請參閱《Amazon EKS 使用者指南》中的什麼是 Amazon EKS。

監控 IAM 身分驗證請求

使用 Amazon DocumentDB 稽核

前往 Amazon CloudWatch 中的稽核日誌資料夾，並使用不同的搜尋模式來取得 IAM 身分驗證的日誌。例如，使用 `{ $.param.mechanism = "MONGODB-AWS" }`做為「搜尋所有日誌串流」的搜尋模式。

如需稽核中支援事件的詳細資訊，請參閱 稽核 Amazon DocumentDB 事件。

使用 Amazon CloudWatch 指標

`StsGetCallerIdentityCalls`：此指標顯示 Amazon DocumentDB 執行個體對 regionized AWS Security Token Service (AWS STS) 端點進行的`GetCallerIdentity`呼叫數量。請參閱身分`MONGODB-AWS`驗證規格，了解資料庫執行個體為何需要進行 STS `GetCallerIdentity`呼叫。

使用 IAM 身分驗證

當您不想在自己的資料庫中管理使用者名稱和密碼時，您可以使用 IAM 身分驗證。IAM 身分驗證僅適用於 Amazon DocumentDB 執行個體型叢集 5.0 版。

IAM 身分驗證對 AWS Security Token Service () 具有相依性AWS STS。如果您在使用 AWS STS IAM 身分驗證時遇到調節例外狀況，請降低連線速率。

如需 IAM 配額，請參閱《IAM 使用者指南》中的 IAM 和 AWS STS 配額。

支援 IAM 的驅動程式

支援 Amazon DocumentDB 5.0 和`MONGODB-AWS`身分驗證機制的驅動程式應使用 Amazon DocumentDB 中的 IAM 身分驗證實作。

> **Important**
>
> 6.13.1 版之前的 Node.js 驅動程式有已知的限制，Amazon DocumentDB 的 IAM 身分驗證目前不支援此限制。使用 Node.js 驅動程式的 Node.js 驅動程式和工具（例如 mongosh) 必須升級為使用 Node.js 驅動程式 6.13.1 版或更新版本。

IAM 身分驗證常見問答集

我可以參考任何範例嗎？

如需範例使用案例和組態，請參閱這些頁面：

- 人類使用者如何使用 IAM 使用者和 IAM 角色向 Amazon DocumentDB 進行身分驗證
- 使用 IAM 角色對 Amazon DocumentDB 進行無密碼身分驗證

使用 Python 驅動程式時發生錯誤：「pymongo.errors.ConfigurationError： MONGODB-AWS authentication require pymongo-auth-aws」。如何解決此問題？

使用 IAM 身分驗證安裝 Python 驅動程式時，請務必使用以下陳述式：

```
pip install 'pymongo[aws]'
```

這將安裝 IAM 身分驗證運作所需的其他 AWS 相依性。

IAM 角色臨時登入資料過期時，我的連線是否會中斷？

否，暫時 IAM 登入資料僅用於建立連線和身分驗證。然後，所有進一步的授權都會發生在 Amazon DocumentDB 叢集中。即使 IAM 登入資料輪換/過期，連線也不會中斷或過時。

使用 SCRAM-SHA-256 進行身分驗證

Amazon DocumentDB （與 MongoDB 相容） 支援密碼型身分驗證的 Salted Challenge Response Authentication Mechanism (SCRAM)。透過 SCRAM，伺服器會驗證用戶端是否知道使用者的密碼，而不會將密碼傳輸至伺服器。Amazon DocumentDB 支援兩種 SCRAM 機制： SCRAM-SHA-1和 SCRAM-SHA-256。 SCRAM-SHA-256使用更強大的 SHA-256 雜湊函數，並使用 FIPS 模式。

當您建立或更新使用者時，您可以將一個或兩個 SCRAM 機制指派給使用者。使用 建立的使用者會SCRAM-SHA-1繼續運作，而不會進行任何變更。SCRAM-SHA-256 身分驗證適用於 Amazon DocumentDB 執行個體型叢集次要版本 5.0.1+ 和 8.0.1+。

> **SCRAM 和 IAM 身分驗證是互斥的**
>
> 使用者可以擁有 SCRAM 機制或MONGODB-AWS機制，但不能同時擁有兩者。如需 IAM 身分驗證，請參閱 使用 IAM 身分進行身分驗證。

主題

- 使用 SCRAM 機制建立使用者
- 更新現有使用者

- 使用 SCRAM-SHA-256 驗證

使用 SCRAM 機制建立使用者

建立使用者時，您可以使用 mechanisms 欄位指定要啟用的 SCRAM 機制。下列範例SCRAM-SHA-256僅建立SCRAM-SHA-1具有 和兩種機制的使用者：

```
db.runCommand({
    createUser: 'sha1user',
    pwd: passwordPrompt(),
    roles: ['readWrite'],
    mechanisms: ['SCRAM-SHA-1']
});
db.runCommand({
    createUser: 'sha256user',
    pwd: passwordPrompt(),
    roles: ['readWrite'],
    mechanisms: ['SCRAM-SHA-256']
});
db.runCommand({
    createUser: 'bothuser',
    pwd: passwordPrompt(),
    roles: ['readWrite'],
    mechanisms: ['SCRAM-SHA-1','SCRAM-SHA-256']
});
```

預設機制

如果您未指定 mechanisms 欄位，Amazon DocumentDB 會SCRAM-SHA-256同時為使用者啟用SCRAM-SHA-1和 。下列兩個命令相當：

```
db.runCommand({
    createUser: 'user1',
    pwd: passwordPrompt(),
    roles: ['readWrite']
});
// is equivalent to:
db.runCommand({
    createUser: 'user1',
    pwd: passwordPrompt(),
    roles: ['readWrite'],
```

```
    mechanisms: ['SCRAM-SHA-1','SCRAM-SHA-256']
});
```

根據預設，啟用這兩個機制可維持與僅支援 之較舊 MongoDB 驅動程式的相容性SCRAM-SHA-1。較新的驅動程式和 FIPS 模式偏好 SCRAM-SHA-256。如果您不希望任何使用者使用 進行身分驗證SCRAM-SHA-1，請只在 SCRAM-SHA-256 mechanisms 欄位中為所有使用者指定 。

更新現有使用者

使用 建立的使用者會SCRAM-SHA-1繼續受到支援，不會有任何變更。若要將現有使用者從 切換SCRAM-SHA-1到 SCRAM-SHA-256，請使用新密碼和 SCRAM-SHA-256機制更新使用者：

```
db.runCommand({
    updateUser: 'existing',
    pwd: passwordPrompt(),
    mechanisms: ['SCRAM-SHA-256']
});
```

使用 SCRAM-SHA-256 驗證

您的驅動程式會自動使用連線字串參數進行身分驗證，然後再在每個連線上傳送第一個命令。若要在連線 URI SCRAM-SHA-256中選取 ，請將 authMechanism 參數設定為 SCRAM-SHA-256：

```
mongodb://username:password@hostname:port/authenticationDatabase?authMechanism=SCRAM-
SHA-256
```

您也可以從 mongosh 傳送明確的身分驗證請求：

```
db.auth({user: 'sha256user', pwd: passwordPrompt(), mechanism: 'SCRAM-SHA-256'})
Enter password:
1
```

如果您未在連線選項中指定身分驗證機制，驅動程式會自動選取目標資料庫使用者支援的機制。

管理 Amazon DocumentDB 使用者

在 Amazon DocumentDB 中，使用者搭配密碼對叢集進行驗證。每個叢集都有在叢集建立期間建立的主要登入憑證。

> **Note**
>
> 所有在 2020 年 3 月 26 日之前建立的新使用者均獲授予 dbAdminAnyDatabase、readWriteAnyDatabase 和 clusterAdmin 角色。建議您重新評估所有使用者並視需要修改角色，以對叢集中所有使用者強制執行最低權限。
> 如需詳細資訊，請參閱使用角色型存取控制的資料庫存取。

主要和serviceadmin使用者

新建立的 Amazon DocumentDB 叢集有兩個使用者：主要使用者和serviceadmin使用者。

主要使用者是具有權限的單一使用者，可執行管理任務並建立具有 角色的其他使用者。當您第一次連線到 Amazon DocumentDB 叢集時，您必須使用主要登入憑證進行身分驗證。建立叢集時，主要使用者會收到 Amazon DocumentDB 叢集的這些管理許可，並獲授予 的角色root。

建立叢集時，會隱含地建立 serviceadmin 使用者。每個 Amazon DocumentDB 叢集都有一個serviceadmin使用者，可讓您 AWS 管理叢集。您無法以 serviceadmin 身分進行登入、捨棄、重新命名、變更密碼或變更許可。任何這麼做的嘗試都會造成錯誤。

> **Note**
>
> 無法刪除 Amazon DocumentDB 叢集的主要和serviceadmin使用者，也無法刪除root主要使用者 的角色。
> 如果您忘記主要使用者密碼，可以使用 AWS 管理主控台 或 重設密碼 AWS CLI。

建立其他使用者

以主要使用者 （或任何具有角色 的使用者createUser) 身分連線之後，您可以建立新的使用者，如下所示。

```
db.createUser(
    {
        user: "sample-user-1",
        pwd: "password123",
        roles:
            [{"db":"admin", "role":"dbAdminAnyDatabase" }]
    }
```

```
)
```

若要檢視使用者詳細資訊，您可以使用 show users 命令，如下所示。您另外可以使用 dropUser 命令移除使用者。如需詳細資訊，請參閱常見命令。

```
show users
{
    "_id" : "serviceadmin",
    "user" : "serviceadmin",
    "db" : "admin",
    "roles" : [
     {
            "role" : "root",
            "db" : "admin"
        }
    ]
},

{
    "_id" : "myPrimaryUser",
    "user" : "myPrimaryUser",
    "db" : "admin",
    "roles" : [
     {
            "role" : "root",
            "db" : "admin"
        }
    ]
},

{
    "_id" : "sample-user-1",
    "user" : "sample-user-1",
    "db" : "admin",
    "roles" : [
     {
            "role" : "dbAdminAnyDatabase",
            "db" : "admin"
     }
    ]
}
```

在此範例中，新使用者 sample-user-1 歸屬於 admin 資料庫。新使用者一律如此。Amazon DocumentDB 沒有 的概念，authenticationDatabase因此所有身分驗證都會在admin資料庫的內容中執行。

建立使用者時，如果您在指定角色時省略 db 欄位，Amazon DocumentDB 會將角色隱含地歸因於發出連線的資料庫。例如，如果您的連線是針對資料庫 sample-database 發出，並執行以下命令，則使用者 sample-user-2 將建立於 admin 資料庫中，並將具有資料庫 sample-database 的 readWrite 許可。

```
db.createUser(
    {
        user: "sample-user-2",
        pwd: "password123",
        roles:
            ["readWrite"]
    }
)
```

建立具有所有資料庫範圍的角色的使用者 (例如，readInAnyDatabase) 時，您必須在建立使用者時位於 admin 資料庫內容中，或是在建立使用者時明確地陳述角色的資料庫。

要切換資料庫的內容，您可以使用下面的命令。

```
use admin
```

若要進一步了解以角色為基礎的存取控制，以及在叢集中的使用者之間強制執行最低權限，請參閱使用角色型存取控制的資料庫存取。

自動輪換 Amazon DocumentDB 的密碼

使用 時 AWS Secrets Manager，您可以將程式碼中的硬式編碼登入資料 （包括密碼） 取代為對 Secrets Manager 的 API 呼叫，以程式設計方式擷取秘密。這有助於確保不讓某人研究您的程式碼而盜用秘密，因為秘密根本不在程式碼中。此外，您可以設定 Secrets Manager，根據您指定的排程自動輪換秘密。這可讓您以短期秘密取代長期秘密，有助於大幅降低洩漏風險。

使用 Secrets Manager，您可以使用 Secrets Manager 提供的 AWS Lambda 函數自動輪換 Amazon DocumentDB 密碼 （即秘密)。

如需 AWS Secrets Manager 和 Amazon DocumentDB 原生整合的詳細資訊，請參閱下列內容：

- 部落格：如何在 Secrets Manager 中 AWS 輪換 Amazon DocumentDB 和 Amazon Redshift 登入資料
- 什麼是 AWS Secrets Manager？
- 輪換 AWS Secrets Manager 機密
- Secrets Manager 中的 Amazon DocumentDB 登入資料

使用角色型存取控制的資料庫存取

您可以在 Amazon DocumentDB （具有 MongoDB 相容性） 中使用角色型存取控制 (RBAC)，限制對使用者可以在資料庫上執行之動作的存取。RBAC 的運作方式是將一或多個角色授予使用者。這些角色決定使用者可以對資料庫資源執行的操作。Amazon DocumentDB 目前支援範圍在資料庫層級的內建角色，例如 `read`、`clusterAdmin`、、 `readWrite` `readAnyDatabase`和使用者定義的角色，這些角色可以範圍限定在特定動作和精細的資源，例如根據您的需求收集。

RBAC 的常見使用案例包括透過建立對叢集中的資料庫或集合具有唯讀存取權的使用者來強制執行最低權限，以及多租用戶應用程式設計，可讓單一使用者存取叢集中的指定資料庫或集合。

> **Note**
>
> 所有在 2020 年 3 月 26 日之前建立的新使用者均獲授予 `dbAdminAnyDatabase`、`readWriteAnyDatabase` 和 `clusterAdmin` 角色。建議您重新評估所有現有使用者，並視需要修改角色，以強制執行叢集的最低權限。

主題

- RBAC 概念
- RBAC 內建角色入門
- RBAC 使用者定義角色入門
- 以使用者身分連線至 Amazon DocumentDB
- 常見命令
- 功能差異
- 限制
- 使用角色型存取控制的資料庫存取

RBAC 概念

以下是與以角色為基礎的存取控制相關的重要術語與概念。如需 Amazon DocumentDB 使用者的詳細資訊，請參閱 管理 Amazon DocumentDB 使用者。

- 使用者 — 可對資料庫進行身分驗證並執行操作的個別實體。
- 密碼 — 用來驗證使用者的秘密。
- 角色 — 授權使用者在一或多個資料庫上執行動作。
- Admin Database — 存放和授權使用者的資料庫。
- 資料庫 (**db**) — 叢集內的命名空間，其中包含用於存放文件的集合。

以下命令會建立一個名為 sample-user 的使用者。

```
db.createUser({user: "sample-user", pwd: "abc123", roles: [{role: "read", db: "sample-database"}]})
```

在此範例中：

- user: "sample-user" — 表示使用者名稱。
- pwd: "abc123" — 表示使用者密碼。
- role: "read", "db: "sample-database" — 表示使用者sample-user在 中具有讀取許可sample-database。



以下範例顯示在您透過 db.getUser(sample-user) 取得使用者 sample-user 後的輸出 。在此範例中，使用者 sample-user 位於 admin 資料庫中，但具有 sample-database 資料庫的讀取角色。



建立使用者時，如果您在指定角色時省略 db 欄位，Amazon DocumentDB 會將角色隱含地歸因於發出連線的資料庫。例如，如果您的連線是針對資料庫 sample-database 發出，並執行以下命令，則使用者 sample-user 將建立於 admin 資料庫中，並將具有資料庫 sample-database 的 readWrite 許可。

```
db.createUser({user: "sample-user", pwd: "abc123", roles: ["readWrite"]})
```

此操作的輸出將會如下所示。

```
{
   "user":"sample-user",
   "roles":[
      {
         "db":"sample-database",
         "role":"readWrite"
      }
   ]
}
```

建立具有所有資料庫範圍的角色的使用者 (例如，readAnyDatabase) 時，您必須在建立使用者時位於 admin 資料庫內容中，或是在建立使用者時明確地陳述角色的資料庫。要對 admin 資料庫發出命令，您可以使用命令 use admin。如需詳細資訊，請參閱常見命令。

RBAC 內建角色入門

為了協助您開始使用以角色為基礎的存取控制，本節會逐一解說透過為具有不同任務職能的三個使用者建立角色，強制最低權限的案例範例。

- user1 是一名新主管，需要能夠檢視和存取叢集中的所有資料庫。

- user2 是一名新員工，只需要存取同一個叢集中的一個資料庫 sample-database-1。
- user3 是現有員工，需要在同一個叢集中檢視和存取他們之前無法存取的不同資料庫 sample-database-2。

user1 和 user2 稍後離職，因此必須撤銷他們的存取權。

若要建立使用者並授予角色，您向叢集驗證的使用者必須具有可對 createUser 和 grantRole 執行動作的相關角色。例如，角色 admin 和 userAdminAnyDatabase 都可以授予這些能力。如需每個角色的動作，請參閱使用角色型存取控制的資料庫存取。

> **Note**
>
> 在 Amazon DocumentDB 中，無論您是否針對admin資料庫發出命令，所有使用者和角色操作 get（例如 create、drop、grant、revoke、 等） 都會隱含地在admin資料庫中執行。

首先，要了解叢集中目前有哪些使用者和角色，您可以執行 show users 命令，如下列範例所示。您將看到兩個使用者，serviceadmin以及叢集的主要使用者。這兩個使用者永遠存在且無法刪除。如需詳細資訊，請參閱管理 Amazon DocumentDB 使用者。

```
show users
```

對於 user1，使用下列命令建立對整個叢集中所有資料庫具有讀寫存取權的角色。

```
db.createUser({user: "user1", pwd: "abc123", roles: [{role: "readWriteAnyDatabase", db:
 "admin"}]})
```

此操作的輸出將會如下所示。

```
{
   "user":"user1",
   "roles":[
      {
         "role":"readWriteAnyDatabase",
         "db":"admin"
      }
   ]
}
```

對於 user2，使用下列命令建立對資料庫 sample-database-1 具有唯讀存取權的角色。

```
db.createUser({user: "user2", pwd: "abc123", roles: [{role: "read", db: "sample-
database-1"}]})
```

此操作的輸出將會如下所示。

```
{
   "user":"user2",
   "roles":[
      {
         "role":"read",
         "db":"sample-database-1"
      }
   ]
}
```

若要模擬 user3 為現有使用者的案例，請先建立使用者 user3，然後指派新角色給 user3。

```
db.createUser({user: "user3", pwd: "abc123", roles: [{role: "readWrite", db: "sample-
database-1"}]})
```

此操作的輸出將會如下所示。

```
{
   "user":"user3",
   "roles":[
      {
         "role":"readWrite",
         "db":"sample-database-1"
      }
   ]
}
```

建立使用者 user3 後，請將 user3 的角色 read 指派給 sample-database-2。

```
db.grantRolesToUser("user3", [{role: "read", db: "sample-database-2"}])
```

最後，user1 和 user2 都離職了，而需要撤銷他們對叢集的存取權。您可以透過捨棄使用者來做到這一點，如下所示。

```
db.dropUser("user1")
db.dropUser("user2")
```

若要確保所有使用者都具有適當的角色，您可以使用下列命令列出所有使用者。

```
show users
```

此操作的輸出將會如下所示。

```
{
   "_id":"serviceadmin",
   "user":"serviceadmin",
   "db":"admin",
   "roles":[
      {
         "db":"admin",
         "role":"root"
      }
   ]
}
{
   "_id":"master-user",
   "user":"master-user",
   "db":"admin",
   "roles":[
      {
         "db":"admin",
         "role":"root"
      }
   ]
}
{
   "_id":"user3",
   "user":"user3",
   "db":"admin",
   "roles":[
      {
         "db":"sample-database-2",
         "role":"read"
      },
      {
         "db":"sample-database-1",
```

```
         "role":"readWrite"
      }
   ]
}
```

RBAC 使用者定義角色入門

為了協助您開始使用使用者定義的角色，本節會逐步解說範例案例：透過為具有不同任務職能的三個使用者建立角色，強制執行最低權限。

在此範例中，適用下列內容：

- user1 是一名新主管，需要能夠檢視和存取叢集中的所有資料庫。
- user2 是新員工，只需要對該相同叢集sample-database-1中的一個資料庫 執行「尋找」動作。
- user3 是現有員工，需要檢視和存取不同資料庫中的特定集合 col2，sample-database-2他們之前在相同叢集中無法存取該集合。
- 對於 user1，使用下列命令建立對整個叢集中所有資料庫具有讀寫存取權的角色。

```
db.createUser(
 {
     user: "user1", pwd: "abc123",
     roles: [{role: "readWriteAnyDatabase", db: "admin"}]
 }
 )
```

此操作的輸出將會如下所示。

```
{
   "user":"user1",
   "roles":[
      {
         "role":"readWriteAnyDatabase",
         "db":"admin"
      }
   ]
}
```

對於 user2，sample-database-1使用下列命令，為資料庫中的所有集合建立具有「尋找」權限的角色。請注意，此角色會確保任何相關聯的使用者只能執行尋找查詢。

```
db.createRole(
{
    role: "findRole",
    privileges: [
    {
        resource: {db: "sample-database-1", collection: ""}, actions: ["find"]
    }],
    roles: []
}
)
```

此操作的輸出將會如下所示。

```
{
   "role":"findRole",
   "privileges":[
      {
         "resource":{
            "db":"sample-database-1",
            "collection":""
         },
         "actions":[
            "find"
         ]
      }
   ],
   "roles":[

   ]
}
```

接著，建立使用者 (user2) 並將最近建立的角色連接到findRole使用者。

```
db.createUser(
{
    user: "user2",
    pwd: "abc123",
    roles: []
})

 db.grantRolesToUser("user2",["findRole"])
```

若要模擬user3現有使用者的案例，請先建立使用者 user3，然後建立名為 collectionRole 的新角色，我們將在下一個步驟中說明該角色user3。

現在您可以將新角色指派給 user3。此新角色將允許 user3插入、更新、刪除和尋找 中某個特定集合 col2 的存取權sample-database-2。

```
db.createUser(
{
    user: "user3",
    pwd: "abc123",
    roles: []
})

db.createRole(
{
    role: "collectionRole",
    privileges: [
    {
        resource: {db: "sample-database-2", collection: "col2"}, actions: ["find",
 "update", "insert", "remove"]
    }],
    roles: []
}
)
```

此操作的輸出將會如下所示。

```
{
   "role":"collectionRole",
   "privileges":[
      {
         "resource":{
            "db":"sample-database-2",
            "collection":"col2"
         },
         "actions":[
            "find",
            "update",
            "insert",
            "remove"
         ]
      }
   ],
```

```
   "roles":[

   ]
}
```

現在user3已建立使用者，您可以授予user3角色 collectionFind。

```
db.grantRolesToUser("user3",["collectionRole"])
```

最後，user1 和 user2 都離職了，而需要撤銷他們對叢集的存取權。您可以透過捨棄使用者來做到這一點，如下所示。

```
db.dropUser("user1")
db.dropUser("user2")
```

若要確保所有使用者都具有適當的角色，您可以使用下列命令列出所有使用者。

```
show users
```

此操作的輸出將會如下所示。

```
{
   "_id":"serviceadmin",
   "user":"serviceadmin",
   "db":"admin",
   "roles":[
      {
         "db":"admin",
         "role":"root"
      }
   ]
}
{
   "_id":"master-user",
   "user":"master-user",
   "db":"admin",
   "roles":[
      {
         "db":"admin",
         "role":"root"
      }
```

```
   ]
}
{
   "_id":"user3",
   "user":"user3",
   "db":"admin",
   "roles":[
      {
         "db":"admin",
         "role":"collectionRole"
      }
   ]
}
```

以使用者身分連線至 Amazon DocumentDB

連線至 Amazon DocumentDB 叢集時，您會在特定資料庫的內容中連線。根據預設，如果您未在連線字串中指定資料庫，則會自動連接到 `test` 資料庫內容中的叢集。會對 `test` 資料庫中的集合發出所有像 `insert` 和 `find` 之類的集合層級命令。

若要查看您位於 或 的資料庫，換句話說，針對 發出命令，請在 mongo shell 中使用 db命令，如下所示。

查詢：

```
db
```

輸出：

```
test
```

雖然預設連線可能在 `test` 資料庫的內容中，但這並不一定表示與連線關聯的使用者有權對 `test` 資料庫執行動作。在上述範例案例中，如果您以具有 `sample-database-1` 資料庫 `readWrite` 角色的使用者 `user3` 身分驗證，則連線的預設內容是 `test` 資料庫。不過，如果您嘗試將文件插入 `test` 資料庫上的集合中，您會收到授權失敗的錯誤訊息。這是因為該使用者未獲權在該資料庫上執行該命令，如下圖所示。

查詢：

```
db
```

輸出：

```
test
```

查詢：

```
db.col.insert({x:1})
```

輸出：

```
WriteCommandError({ "ok" : 0, "code" : 13, "errmsg" : "Authorization failure" })
```

如果您變更連接到 sample-database-1 資料庫的內容，則可以寫入使用者有權執行此操作的集合。

查詢：

```
use sample-database-1
```

輸出：

```
switched to db sample-database-1
```

查詢：

```
db.col.insert({x:1})
```

輸出：

```
WriteResult({ "nInserted" : 1})
```

當您對具有特定使用者的叢集進行驗證時，也可以在連線字串中指定資料庫。如此在使用者通過 admin 資料庫驗證之後就不必執行 use 命令。

以下連線字串會對 admin 資料庫驗證使用者，但連線的內容將是針對 sample-database-1 資料庫。

```
mongo "mongodb://user3:abc123@sample-cluster.node.us-east-1.docdb.amazonaws.com:27017/
sample-database-2"
```

常見命令

本節提供在 Amazon DocumentDB 中使用角色型存取控制的常見命令範例。您必須位於 admin 資料庫的內容中，才能建立和修改使用者和角色。您可以使用 use admin 命令切換至 admin 資料庫。

> **Note**
>
> 對使用者和角色的修改將隱含地發生在 admin 資料庫中。建立具有所有資料庫範圍之角色的使用者 (例如，readAnyDatabase) 時，您必須在建立使用者時位於 admin 資料庫內容中 (亦即 use admin)，或是在建立使用者時明確地陳述角色的資料庫 (如本節中的範例 2 所示)。

範例 1：建立具有資料庫 read角色的使用者foo。

```
db.createUser({user: "readInFooBar", pwd: "abc123", roles: [{role: "read", db:
 "foo"}]})
```

此操作的輸出將會如下所示。

```
{
   "user":"readInFooBar",
   "roles":[
      {
         "role":"read",
         "db":"foo"
      }
   ]
}
```

範例 2：在所有資料庫上建立具有讀取存取權的使用者。

```
db.createUser({user: "readAllDBs", pwd: "abc123", roles: [{role: "readAnyDatabase", db:
 "admin"}]})
```

此操作的輸出將會如下所示。

```
{
   "user":"readAllDBs",
   "roles":[
      {
```

```
         "role":"readAnyDatabase",
         "db":"admin"
      }
   ]
}
```

範例 3：將read角色授予新資料庫上的現有使用者。

```
db.grantRolesToUser("readInFooBar", [{role: "read", db: "bar"}])
```

範例 4：更新使用者的角色。

```
db.updateUser("readInFooBar", {roles: [{role: "read", db: "foo"}, {role: "read", db:
 "baz"}]})
```

範例 5：撤銷使用者資料庫的存取權。

```
db.revokeRolesFromUser("readInFooBar", [{role: "read", db: "baz"}])
```

範例 6：描述內建角色。

```
db.getRole("read", {showPrivileges:true})
```

此操作的輸出將會如下所示。

```
{
   "role":"read",
   "db":"sample-database-1",
   "isBuiltin":true,
   "roles":[

   ],
   "inheritedRoles":[

   ],
   "privileges":[
      {
         "resource":{
            "db":"sample-database-1",
            "collection":""
         },
```

```
         "actions":[
            "changeStream",
            "collStats",
            "dbStats",
            "find",
            "killCursors",
            "listCollections",
            "listIndexes"
         ]
      }
   ],
   "inheritedPrivileges":[
      {
         "resource":{
            "db":"sample-database-1",
            "collection":""
         },
         "actions":[
            "changeStream",
            "collStats",
            "dbStats",
            "find",
            "killCursors",
            "listCollections",
            "listIndexes"
         ]
      }
   }
```

範例 7：從叢集捨棄使用者。

```
db.dropUser("readInFooBar")
```

此操作的輸出將會如下所示。

```
true
```

範例 8：建立具有特定集合讀取和寫入存取權的角色

```
db.createRole(
{
    role: "collectionRole",
```

```
    privileges: [
    {
        resource: {db: "sample-database-2", collection: "col2"}, actions: ["find",
 "update", "insert", "remove"]
    }],
    roles: []
}
)
```

此操作的輸出將會如下所示。

```
{
   "role":"collectionRole",
   "privileges":[
      {
         "resource":{
            "db":"sample-database-2",
            "collection":"col2"
         },
         "actions":[
            "find",
            "update",
            "insert",
            "remove"
         ]
      }
   ],
   "roles":[

   ]
}
```

範例 9：建立使用者並指派使用者定義的角色

```
db.createUser(
{
    user: "user3",
    pwd: "abc123",
    roles: []
})

db.grantRolesToUser("user3",["collectionRole"])
```

範例 10：將其他權限授予使用者定義的角色

```
db.grantPrivilegesToRole(
  "collectionRole",
  [
    {
      resource: { db: "sample-database-1", collection: "col1" },
      actions: ["find", "update", "insert", "remove"]
    }
  ]
)
```

範例 11：從使用者定義的角色移除權限

```
db.revokePrivilegesFromRole(
  "collectionRole",
  [
    {
      resource: { db: "sample-database-1", collection: "col2" },
      actions: ["find", "update", "insert", "remove"]
    }
  ]
)
```

範例 12：更新現有的使用者定義角色

```
db.updateRole(
  "collectionRole",
  {
    privileges: [
    {
        resource: {db: "sample-database-3", collection: "sample-collection-3"},
 actions: ["find", "update", "insert", "remove"]
    }],
    roles: []
  }
)
```

功能差異

在 Amazon DocumentDB 中，使用者和角色定義會存放在admin資料庫中，並對admin資料庫進行身分驗證。此功能不同於 MongoDB Community Edition，但與 MongoDB Atlas 一致。

Amazon DocumentDB 也支援變更串流，提供在叢集集合中發生的按時間順序排列的變更事件序列。listChangeStreams 動作會套用到叢集層級（也就是所有資料庫），而modifyChangeStreams動作則可套用到資料庫層級和叢集層級。

限制

下表包含 Amazon DocumentDB 中角色型存取控制的限制。

說明	限制
每個叢集的使用者數目	1000
與使用者相關聯的角色數目	1000
使用者定義角色的數量	100
與權限相關聯的資源數量	100

使用角色型存取控制的資料庫存取

您可以使用以角色為基礎的存取控制建立使用者並授予其一或多個角色，以決定使用者可以在資料庫或叢集中執行哪些操作。

以下是 Amazon DocumentDB 目前支援的內建角色清單。

> **Note**
>
> 在 Amazon DocumentDB 4.0 和 5.0 中， ListCollection和 ListDatabase命令可以選擇性地使用 authorizedCollections和 authorizedDatabases 參數，分別列出使用者有權使用 和 listDatabase角色存取的集合listCollections和資料庫。此外，使用者現在可以終止自己的游標，而不需要該KillCursor角色。

Database user

角色名稱	說明	動作
read	授予使用者對指定資料庫的讀取權限。	changeStreams

角色名稱	說明	動作
		collStats dbStats find killCursors listIndexes listCollections
readWrite	授予使用者對指定資料庫的讀取和寫入權限。	read 許可中的所有動作。 createCollection dropCollection createIndex dropIndex insert killCursors listIndexes listCollections remove update

Cluster user

角色名稱	說明	動作
readAnyDatabase	授予使用者對叢集中所有資料庫的讀取存取權。	read 許可中的所有動作。 listChangeStreams listDatabases
readWriteAnyDatabase	授予使用者對叢集中所有資料庫的讀寫存取權。	readWrite 許可中的所有動作。 listChangeStreams listDatabases
userAdminAnyDatabase	授予使用者指派和修改任何使用者對指定資料庫所具有的角色或許可的能力。	changeCustomData changePassword createUser dropRole dropUser grantRole listDatabases revokeRole viewRole viewUser

角色名稱	說明	動作
dbAdminAnyDatabase	授予使用者在任何指定的資料庫上執行資料庫管理角色的能力。	dbAdmin 許可中的所有動作。 dropCollection listDatabases listChangeStreams modifyChangeStreams

Superuser

角色名稱	說明	動作
root	授予使用者存取合併下列所有角色的資源與操作：readWriteAnyDatabase 、dbAdminAnyDatabase 、userAdminAnyDatabase 、clusterAdmin 、restore 和 backup。	來自 readWriteAnyDatabase 、dbAdminAnyDatabase 、userAdminAnyDatabase 、clusterAdmin 、restore 和 backup 的所有動作。

Database administrator

角色名稱	說明	動作
dbAdmin	授予使用者在指定的資料庫上執行管理任務的能力。	bypassDocumentValidation

角色名稱	說明	動作
		collMod collStats createCollection createIndex dropCollection dropDatabase dropIndex dbStats find killCursors listIndexes listCollections modifyChangeStreams
dbOwner	透過結合角色 dbAdmin 和 readWrite ，授予使用者在指定的資料庫上執行任何管理任務的能力。	來自 dbAdmin 和 readWrite 的所有動作。

Cluster administrator

角色名稱	說明	動作
clusterAdmin	透過結合 clusterManager 、clusterMonitor 和 hostManager 角色，授予使用者最大的叢集管理存取權。	來自 clusterManager 、clusterMonitor 和 hostManager 的所有動作。 listChangeStreams dropDatabase modifyChangeStreams
clusterManager	授予使用者對指定叢集執行管理和監控動作的能力。	listChangeStreams listSessions modifyChangeStreams replSetGetConfig
clusterMonitor	授予使用者對監視工具具有唯讀存取權的能力。	collStats dbStats find getParameter hostInfo indexStats

角色名稱	說明	動作
		killCursors listChangeStreams listCollections listDatabases listIndexes listSessions replSetGetConfig serverStatus top
hostManager	授予使用者監控和管理伺服器的能力。	auditConfigure killCursors killAnyCursor killAnySession killop

Backup administrator

角色名稱	說明	動作
backup	授予使用者備份資料所需的存取權。	getParameter insert

角色名稱	說明	動作
		find listChangeStreams listCollections listDatabases listIndexes update

角色名稱	說明	動作
restore	授予使用者還原資料所需的存取權。	bypassDocumentValidation changeCustomData changePassword collMod createCollection createIndex createUser dropCollection dropRole dropUser getParameter grantRole find insert listCollections modifyChangeStreams revokeRole

角色名稱	說明	動作

		`remove`
		`viewRole`
		`viewUser`
		`update`

更新您的 Amazon DocumentDB TLS 憑證

主題

- 更新您的應用程式和 Amazon DocumentDB 叢集
- 自動伺服器憑證輪換
- 疑難排解
- 常見問答集

Amazon DocumentDB 叢集的憑證授權機構 (CA) 憑證已於 2024 年 8 月更新。如果您使用已啟用 Transport Layer Security (TLS) 的 Amazon DocumentDB 叢集 （預設設定），而且尚未輪換用戶端應用程式和伺服器憑證，則需要下列步驟來緩解應用程式和 Amazon DocumentDB 叢集之間的連線問題。

- 步驟 1：下載新的 CA 憑證並更新您的應用程式
- 步驟 2：更新伺服器憑證

CA 和伺服器憑證已更新為 Amazon DocumentDB 標準維護和安全最佳實務的一部分。用戶端應用程式必須將新的 CA 憑證新增至其信任存放區，而且必須更新現有的 Amazon DocumentDB 執行個體，才能在此過期日期之前使用新的 CA 憑證。

更新您的應用程式和 Amazon DocumentDB 叢集

請遵循本節所述步驟更新您應用程式的憑證授權機構憑證套件 (步驟 1) 和您叢集的伺服器憑證 (步驟 2)。將變更套用至您的生產環境前，我們強烈在開發或預備環境中測試這些步驟。

> **Note**
>
> 您必須在您擁有 Amazon DocumentDB 叢集的每個 AWS 區域 中完成步驟 1 和 2。

步驟 1：下載新的 CA 憑證並更新您的應用程式

下載新的 CA 憑證並更新您的應用程式，以使用新的 CA 憑證建立與 Amazon DocumentDB 的 TLS 連線。請從下列位置下載新的 CA 憑證套件：https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem。此操作會下載名為 `global-bundle.pem` 的檔案。

> **Note**
>
> 如果您要存取同時包含舊 CA 憑證 (`rds-ca-2019-root.pem`) 和新 CA 憑證 (`rds-ca-rsa2048-g1`、`rds-ca-rsa4096-g1`、`rds-ca-ecc384-g1`) 的金鑰存放區，請確認金鑰存放區已選取 `global-bundle`。

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

接著更新您的應用程式，以使用新的憑證套件。新的 CA 套件包含舊 CA 憑證 (rds-ca-2019) 和新的 CA 憑證 (rds-ca-rsa2048-g1、rds-ca-rsa4096-g1、rds-ca-ecc384-g1)。只要在新的 CA 套件中有這兩個憑證，即可用兩個步驟更新您的應用程式和叢集。

對於 Java 應用程式，您必須使用新的 CA 憑證建立新的信任存放區。如需說明，請參閱 使用已啟用的 TLS 連線主題中的 Java 標籤。

如果要確認應用程式是否有在使用最新的憑證授權機構憑證套件，請參閱我如何確定自己使用的是最新的 CA 套件？。如果您已經在應用程式中使用最新的憑證授權機構憑證套件，可以跳到步驟 2。

如需搭配您的應用程式使用 CA 套件的範例，請參閱 加密傳輸中的資料 和 使用已啟用的 TLS 連線。

> **Note**
>
> MongoDB Go Driver 1.2.1 目前只接受 `sslcertificateauthorityfile` 中的一個 CA 伺服器憑證。如需在啟用 TLS 時使用 Go 連線至 Amazon DocumentDB 使用已啟用的 TLS 連線的詳細資訊，請參閱 。

步驟 2：更新伺服器憑證

應用程式更新為使用新的 CA 套件後，下一步是透過修改 Amazon DocumentDB 叢集中的每個執行個體來更新伺服器憑證。若要修改執行個體以使用新的伺服器憑證，請參閱下列指示。

Amazon DocumentDB 提供下列 CAs 來簽署資料庫執行個體的資料庫伺服器憑證：

- rds-ca-ecc384-g1 — 使用具有 ECC 384 私有金鑰演算法和 SHA384 簽署演算法的憑證授權機構。此 CA 支援自動伺服器憑證輪換。這僅在 Amazon DocumentDB 4.0 和 5.0 上受支援。
- rds-ca-rsa2048-g1 — 最多使用具有 RSA 2048 私有金鑰演算法和 SHA256 簽署演算法的憑證授權機構 AWS 區域。此 CA 支援自動伺服器憑證輪換。
- rds-ca-rsa4096-g1 — 搭配 RSA 4096 私有金鑰演算法和 SHA384 簽署演算法使用憑證授權機構。此 CA 支援自動伺服器憑證輪換。

> **Note**
>
> 如果您使用的是 AWS CLI，您可以使用 describe-certificates 來查看上一節中列出的憑證授權單位的有效性。

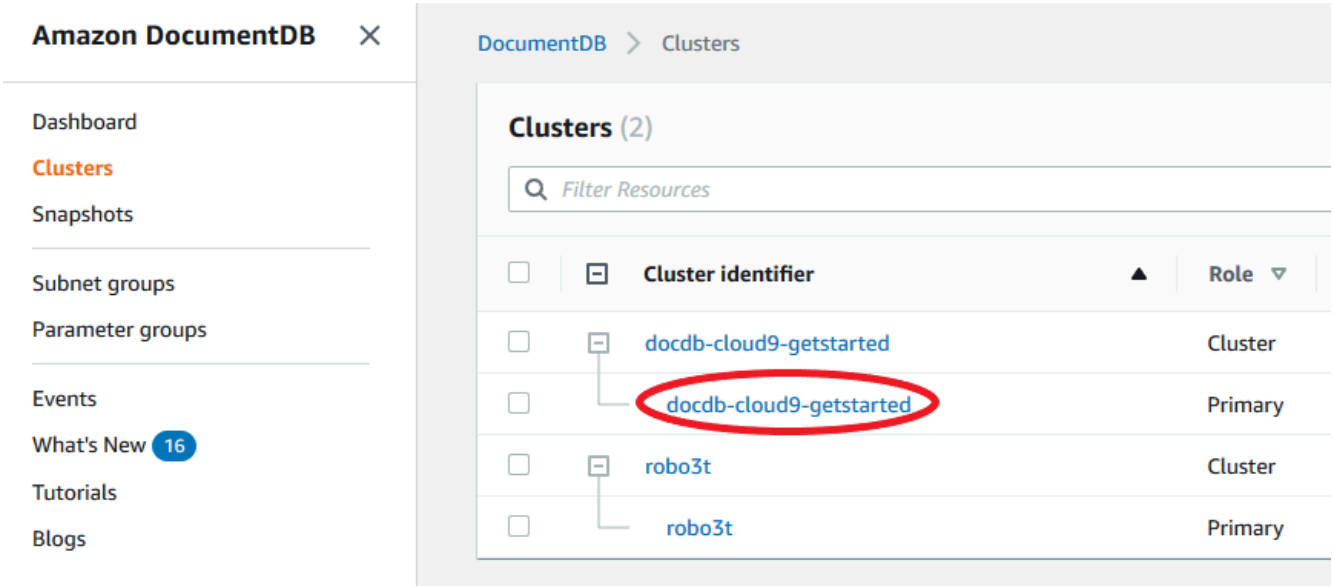
這些 CA 憑證包含在區域和全域憑證套件中。當您搭配資料庫使用 rds-ca-rsa2048-g1、rds-ca-rsa4096-g1 或 rds-ca-ecc384-g1 CA 時，Amazon DocumentDB 會管理資料庫上的資料庫伺服器憑證。Amazon DocumentDB 會在資料庫伺服器憑證過期之前自動輪換資料庫伺服器憑證。

> **Note**
>
> 如果您的叢集正在下列引擎修補程式版本上執行，Amazon DocumentDB 不需要重新啟動以進行憑證輪換：
>
> - Amazon DocumentDB 3.6：1.0.208662 或更新版本
> - Amazon DocumentDB 4.0：2.0.10179 或更新版本
> - Amazon DocumentDB 5.0：3.0.4780 或更高版本
>
> 您可以執行下列命令來判斷目前的 Amazon DocumentDB 引擎修補程式版本：`db.runCommand({getEngineVersion: 1})`。
> 更新伺服器憑證之前，請確定您已完成 步驟 1。

Using the AWS 管理主控台

完成下列步驟，使用 來識別和輪換現有 Amazon DocumentDB 執行個體的舊伺服器憑證 AWS 管理主控台。

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/docdb：// 開啟 Amazon DocumentDB 主控台。
2. 在畫面右上角的區域清單中，選擇 AWS 區域 叢集所在的 。
3. 在主控台左側的導覽窗格中選擇叢集。
4. 您可能需要識別哪些執行個體仍在舊的伺服器憑證上 (`rds-ca-2019`)。您可以在位於叢集資料表最右側的憑證授權單位欄中執行此操作。
5. 在叢集表格中，您會看到最左邊的欄位叢集識別符。您的執行個體會列在叢集下，類似於下列螢幕擷取畫面。



6. 勾選您感興趣的執行個體左側的方塊。
7. 選擇 Actions (動作)，然後選擇 Modify (修改)。
8. 在 Certificate authority (憑證授權單位) 下方，選取此執行個體新的伺服器憑證 (`rds-ca-rsa2048-g1`)。
9. 您可以在下一頁上查看變更摘要。請注意，修改執行個體之前，會出現其他提醒來提醒您確保應用程式正使用最新的憑證 CA 套件，以免導致連線中斷。
10. 您可以在下次維護時段中套用修改，或立即套用。如果您想要立即修改伺服器憑證，請使用 Apple Immediately (立即套用) 選項。
11. 選擇 Modify instance (修改執行個體) 以完成更新。

Using the AWS CLI

完成下列步驟，使用 識別和輪換現有 Amazon DocumentDB 執行個體的舊伺服器憑證 AWS CLI。

1. 若要立即修改執行個體，請對叢集中的每個執行個體執行下列命令。

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier>
 --ca-certificate-identifier rds-ca-rsa2048-g1 --apply-immediately
```

2. 若要修改叢集內的執行個體，以在叢集的下一個維護時段使用新的憑證授權機構憑證，請對叢集中的每個執行個體執行下列命令。

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier>
 --ca-certificate-identifier rds-ca-rsa2048-g1 --no-apply-immediately
```

自動伺服器憑證輪換

Amazon DocumentDB 支援自動伺服器憑證輪換。伺服器憑證是發給每個叢集執行個體的分葉憑證。與根 CA 憑證相反，伺服器憑證的有效性很短 (12 個月），Amazon DocumentDB 會自動處理輪換，而無需您採取任何動作。Amazon DocumentDB 對此自動輪換使用相同的根 CA，因此您不需要下載新的 CA 套件。

> **Important**
>
> 連線至 Amazon DocumentDB 叢集時，請信任根 CA 套件，而不是直接信任每個伺服器憑證。這可防止伺服器憑證輪換後的連線錯誤。請參閱 使用已啟用的 TLS 連線。

Amazon DocumentDB 會在您偏好的維護時段中，嘗試在伺服器憑證的一半生命週期輪換您的伺服器憑證。新的伺服器憑證有效期為 12 個月。

使用 describe-db-engine-versions命令並檢查 SupportsCertificateRotationWithoutRestart旗標，以識別引擎版本是否支援在不重新啟動的情況下輪換憑證。

> **Note**
>
> 如果您的叢集在下列引擎修補程式版本上執行，Amazon DocumentDB 支援伺服器憑證輪換，而不會重新啟動：

- Amazon DocumentDB 3.6：1.0.208662 或更新版本
- Amazon DocumentDB 4.0：2.0.10179 或更新版本
- Amazon DocumentDB 5.0：3.0.4780 或更高版本

您可以執行此命令來判斷目前的 Amazon DocumentDB 引擎修補程式版本：`db.runCommand({getEngineVersion: 1})`。
如果您使用的是較舊的引擎修補程式版本，Amazon DocumentDB 會輪換伺服器憑證，並在您偏好的維護時段排程資料庫重新啟動事件。

疑難排解

如果您在憑證輪換時遇到連線至叢集的問題，請嘗試下列動作：

- 確認您的用戶端使用最新的憑證服務套件。請參閱 我如何確定自己使用的是最新的 CA 套件？。
- 確認您的執行個體使用最新的憑證。請參閱 如何知道哪些 Amazon DocumentDB 執行個體正在使用舊/新伺服器憑證？。
- 確認您的應用程式正在使用最新的憑證 CA。某些驅動程式 (例如 Java 和 Go) 需要額外的程式碼，才能將憑證套件中的多個憑證匯入信任存放區。如需使用 TLS 連線至 Amazon DocumentDB 的詳細資訊，請參閱 以程式設計方式連線至 Amazon DocumentDB。
- 聯絡支援。如果您有疑問或問題，請聯絡 支援。

常見問答集

以下是與 TLS 憑證相關的一些常見問題。

要是我有問題，該怎麼辦？

如果您有疑問或問題，請聯絡 支援。

如何知道我是否使用 TLS 連線到我的 Amazon DocumentDB 叢集？

您可以檢查叢集參數群組的 `tls` 參數，判斷您的叢集是否正在使用 TLS。如果 `tls` 參數設為 `enabled`，即表示您正在使用 TLS 憑證連線到叢集。如需詳細資訊，請參閱管理 Amazon DocumentDB 叢集參數群組。

為何需要更新 CA 和伺服器憑證？

Amazon DocumentDB CA 和伺服器憑證正在更新，作為 Amazon DocumentDB 標準維護和安全性最佳實務的一部分。

如果在過期日期之前未採取任何動作，會發生什麼情況？

如果您使用 TLS 以過期的 CA 憑證連線至 Amazon DocumentDB 叢集，則透過 TLS 連線的應用程式將無法再與 Amazon DocumentDB 叢集通訊。

Amazon DocumentDB 不會在過期前自動輪換資料庫憑證。您必須在過期日期之前或之後更新您的應用程式和叢集，以使用新的 CA 憑證。

如何知道哪些 Amazon DocumentDB 執行個體正在使用舊/新伺服器憑證？

若要識別仍使用舊伺服器憑證的 Amazon DocumentDB 執行個體，您可以使用 Amazon DocumentDB AWS 管理主控台 或 AWS CLI。

使用 AWS 管理主控台

辨識在您的叢集中仍在使用舊憑證的執行個體

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在畫面右上角的區域清單中，選擇 AWS 區域 執行個體所在的 。
3. 在主控台左側的導覽窗格中選擇叢集。
4. 憑證授權單位欄 （ 靠近資料表的最右側） 會顯示哪些執行個體仍在舊的伺服器憑證 (`rds-ca-2019`) 和新的伺服器憑證 () 上`rds-ca-rsa2048-g1`。

使用 AWS CLI

若要辨識您的叢集中正使用舊伺服器憑證的執行個體，請搭配下列參數使用 `describe-db-clusters` 命令。

```
aws docdb describe-db-instances \
    --filters Name=engine,Values=docdb \
    --query 'DBInstances[*].
{CertificateVersion:CACertificateIdentifier,InstanceID:DBInstanceIdentifier}'
```

如何修改 Amazon DocumentDB 叢集中的個別執行個體以更新伺服器憑證？

同時更新指定叢集中所有執行個體的伺服器憑證。若要修改叢集中的執行個體，您可以使用主控台或 AWS CLI。

> **Note**
>
> 更新伺服器憑證前，請先確認您已完成步驟 1。

使用 AWS 管理主控台

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在畫面右上角的區域清單中，選擇 AWS 區域 叢集所在的 。
3. 在主控台左側的導覽窗格中選擇叢集。
4. 憑證授權單位欄 （ 靠近資料表的最右側 ） 會顯示哪些執行個體仍在舊的伺服器憑證 () 上`rds-ca-2019`。
5. 在叢集資料表的叢集識別符下，選取要修改的執行個體。
6. 選擇 Actions (動作)，然後選擇 Modify (修改)。
7. 在 Certificate authority (憑證授權單位) 下方，選取此執行個體新的伺服器憑證 (`rds-ca-rsa2048-g1`)。
8. 您可以在下一頁上查看變更摘要。請注意，修改執行個體之前，會出現其他提醒來提醒您確保應用程式正使用最新的憑證 CA 套件，以免導致連線中斷。
9. 您可以在下次維護時段中套用修改，或立即套用。
10. 選擇 Modify instance (修改執行個體) 以完成更新。

使用 AWS CLI

完成下列步驟，使用 來識別和輪換現有 Amazon DocumentDB 執行個體的舊伺服器憑證 AWS CLI。

1. 若要立即修改執行個體，請對叢集中的每個執行個體執行下列命令。

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier> --
ca-certificate-identifier rds-ca-rsa2048-g1 --apply-immediately
```

2. 若要修改叢集內的執行個體，以在叢集的下一個維護時段使用新的憑證授權機構憑證，請對叢集中的每個執行個體執行下列命令。

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier> --
ca-certificate-identifier rds-ca-rsa2048-g1 --no-apply-immediately
```

如果我將新的執行個體新增至現有叢集，會發生什麼情況？

所有建立的新執行個體將會使用舊的伺服器憑證，且需要使用舊 CA 憑證的 TLS 連線。2024 年 1 月 25 日之後建立的任何新 Amazon DocumentDB 執行個體都會預設為使用新的憑證 rds-ca-rsa2048-g1。

如果我的叢集發生執行個體替換或容錯移轉作業，會發生什麼情況？

如果您的叢集發生執行個體替換作業，建立的新執行個體會繼續使用執行個體之前使用的相同伺服器憑證。我們建議您同時為所有執行個體更新伺服器憑證。如果我的叢集發生容錯移轉作業，會在新的主要伺服器上會使用伺服器憑證。

如果我沒有使用 TLS 連線到我的叢集，我仍需更新各個執行個體嗎？

我們強烈建議您啟用 TLS。如果您未啟用 TLS，仍建議您在未來計劃使用 TLS 連線到叢集時輪換 Amazon DocumentDB 執行個體上的憑證。如果您從未計劃使用 TLS 連線到 Amazon DocumentDB 叢集，則不需要採取任何動作。

如果我沒有使用 TLS 連線到自己的叢集，但打算在未來這樣做，我該怎麼辦？

如果您在 2024 年 1 月之前建立叢集，請遵循上一節的步驟 1 和步驟 2，以確保您的應用程式使用更新的 CA 套件，且每個 Amazon DocumentDB 執行個體都使用最新的伺服器憑證。如果您在 2024 年 1 月 25 日之後建立叢集，您的叢集將已有最新的伺服器憑證 (rds-ca-rsa2048-g1)。若要驗證您的應用程式是否使用最新的 CA 套件，請參閱 如果我沒有使用 TLS 連線到我的叢集，我仍需更新各個執行個體嗎？

截止日期是否可以延長到 2024 年 8 月之後？

如果您的應用程式透過 TLS 連線，則無法延長截止日期。

我如何確定自己使用的是最新的 CA 套件？

若要驗證您是否擁有最新的套件，請使用下列命令。若要執行此命令，您必須安裝 java，而且 java 工具必須位於 shell 的 PATH 變數中。如需詳細資訊，請參閱使用 Java

macOS 和 Amazon Linux

```
keytool -printcert -v -file global-bundle.pem
```

Windows

```
keytool -printcert -v -file global-bundle.p7b
```

我為何在 CA 套件的名稱中看到「RDS」？

對於某些管理功能，例如憑證管理，Amazon DocumentDB 會使用與 Amazon Relational Database Service (Amazon RDS) 共用的操作技術。

新憑證何時過期？

新的伺服器憑證將過期（通常），如下所示：

- rds-ca-rsa2048-g1 — 2061 年過期
- rds-ca-rsa4096-g1 — 過期 2121
- rds-ca-ecc384-g1 — 過期 2121

如果在憑證過期之前未採取行動，我會看到哪些類型的錯誤？

錯誤訊息會根據您的驅動程式而有所不同。一般而言，您會看到包含「憑證已過期」字串的憑證驗證錯誤。

如果我套用新的伺服器憑證，我可以還原為舊的伺服器憑證嗎？

如果您需要將執行個體還原為舊的伺服器憑證，請對叢集中的所有執行個體執行此操作。您可以使用 AWS 管理主控台 或 來還原叢集中每個執行個體的伺服器憑證 AWS CLI。

使用 AWS 管理主控台

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在畫面右上角的區域清單中，選擇 AWS 區域 叢集所在的 。
3. 在主控台左側的導覽窗格中選擇叢集。
4. 在叢集資料表的叢集識別符下，選取要修改的執行個體。選擇 Actions (動作)，然後選擇 Modify (修改)。

5. 在 Certificate authority (憑證授權單位) 下方，您可以選取舊的伺服器憑證 (`rds-ca-2019`)。
6. 選擇 Continue (繼續) 以檢視修改摘要。
7. 在此產生的頁面中，您可以選擇排程要在下一次維護時段中套用的修改項目，或立即套用修改項目。進行選取，並選擇 Modify instance (修改執行個體)。

> **Note**
>
> 若您選擇立即套用您的修改項目，則系統會一併套用待定修改佇列內的任何變更。如果有任何待定修改需要停機才能執行，則選擇此選項可能會導致未預期的停機。

使用 AWS CLI

```
aws docdb modify-db-instance --db-instance-identifier <db_instance_name> ca-
certificate-identifier rds-ca-2019 <--apply-immediately | --no-apply-immediately>
```

如果選擇 `--no-apply-immediately`，將在叢集的下一個維護時段套用變更。

如果我們從快照或時間點還原回復，是否會有新的伺服器憑證？

如果您在 2024 年 8 月之後還原快照或執行point-in-time還原，則建立的新叢集將使用新的 CA 憑證。

如果我無法從任何 Mac OS 直接連線至 Amazon DocumentDB 叢集，該怎麼辦？

Mac OS 已更新信任憑證的要求。信任的憑證現在必須在 397 天內有效 （ 請參閱 https://support.apple.com/en-us/HT211025)。

> **Note**
>
> 在較新版本的 Mac OS 中觀察到此限制。

Amazon DocumentDB 執行個體憑證的有效期超過四年，超過 Mac OS 上限。若要從執行 Mac OS 的電腦直接連線至 Amazon DocumentDB 叢集，您必須在建立 TLS 連線時允許無效的憑證。在此情況下，無效的憑證表示有效期間超過 397 天。在連線至 Amazon DocumentDB 叢集時允許無效的憑證之前，您應該先了解風險。

若要使用 從 Mac OS 連線至 Amazon DocumentDB 叢集 AWS CLI，請使用 `tlsAllowInvalidCertificates` 參數。

```
mongo --tls --host <hostname> --username <username> --password <password> --port 27017
 --tlsAllowInvalidCertificates
```

Amazon DocumentDB 中的合規驗證

Amazon DocumentDB 的安全性和合規性由第三方稽核人員評估為多個 AWS 合規計劃的一部分，包括下列項目：

- 系統和組織控制 (SOC) 1、2、3。如需詳細資訊，請參閱 SOC。
- 聯邦風險與授權管理計劃 (FedRAMP)。如需詳細資訊，請參閱合規計畫的AWS 服務範圍。
- 支付卡產業資料安全標準 (PCI DSS)。如需詳細資訊，請參閱 PCI DSS 。
- ISO 9001、27001、27017 和 27018。如需詳細資訊，請參閱 ISO 認證。
- 健康保險流通與責任法案商業夥伴協議 (HIPAA BAA)。如需詳細資訊，請參閱 HIPAA 合規。

AWS 提供由合規計劃在 AWS 服務範圍內特定AWS 合規計劃的經常更新服務清單。

可使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱以 AWS 成品下載報告。

如需 AWS 合規計劃的詳細資訊，請參閱AWS 合規計劃。

您使用 Amazon DocumentDB 時的合規責任取決於資料的敏感度、組織的合規目標，以及適用的法律和法規。如果您使用 Amazon DocumentDB 符合 HIPAA 或 PCI 等標準， AWS 會提供資源來協助：

- AWS 合規資源 – 工作手冊和指南的集合，可能適用於您的產業和位置。
- 安全與合規快速入門指南 – 部署指南，討論架構考量，並提供在其中部署以安全與合規為重心的基準環境的步驟 AWS。
- AWS Config – 評估資源組態是否符合內部實務、產業準則和法規的服務。
- AWS Security Hub – 內安全狀態的全方位檢視 AWS ，可協助您檢查是否符合安全產業標準和最佳實務。
- HIPAA 安全與合規架構白皮書 – 說明公司如何 AWS 用來建立 HIPAA 合規應用程式的白皮書。

Amazon DocumentDB 中的彈性

AWS 全球基礎設施是以 AWS 區域 和 可用區域為基礎建置。 AWS 區域 提供多個實體分隔和隔離的可用區域，這些可用區域與低延遲、高輸送量和高備援聯網連接。透過可用區域，您所設計與操作的應

用程式和資料庫，就能夠在可用區域之間自動容錯移轉，而不會發生中斷。可用區域的可用性、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

Amazon DocumentDB 叢集只能在至少在兩個可用區域中具有至少兩個子網路的 Amazon VPC 中建立。透過將叢集執行個體分散到至少兩個可用區域，Amazon DocumentDB 有助於確保在不太可能發生可用區域故障時，叢集中有可用的執行個體。Amazon DocumentDB 叢集的叢集磁碟區一律跨越三個可用區域，以提供耐久的儲存體，減少資料遺失的可能性。

如需 AWS 區域 和可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

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

容錯能力和自我修復儲存

儲存磁碟區的每個 10 GB 部分會以六種方式跨三個可用區域複寫。Amazon DocumentDB 使用容錯儲存，可透明地處理最多兩個資料複本的遺失，而不會影響資料庫寫入可用性，以及最多三個複本，而不會影響讀取可用性。Amazon DocumentDB 儲存體也是自我修復；資料區塊和磁碟會持續掃描錯誤並自動取代。

手動備份和還原

Amazon DocumentDB 提供建立叢集完整備份的功能，以便長期保留和復原。如需詳細資訊，請參閱在 Amazon DocumentDB 中備份和還原。

時間點復原

Point-in-time復原有助於保護您的 Amazon DocumentDB 叢集免於意外寫入或刪除操作。有了時間點復原，就無需為建立、維護或排程隨需備份而煩惱。如需詳細資訊，請參閱還原至某個時間點。

Amazon DocumentDB 中的基礎設施安全性

Amazon DocumentDB 是受管服務，受到 AWS 全球網路安全的保護。如需 AWS 安全服務以及 如何 AWS 保護基礎設施的相關資訊，請參閱AWS 雲端安全。若要使用基礎設施安全的最佳實務設計您的 AWS 環境，請參閱安全支柱 AWS Well-Architected Framework 中的基礎設施保護。

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

- Transport Layer Security (TLS)。我們需要 TLS 1.2 並建議使用 TLS 1.3。

- 具備完美轉送私密(PFS)的密碼套件，例如 DHE (Ephemeral Diffie-Hellman)或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

您可以從任何網路位置呼叫這些 API 操作。您可以使用 Amazon DocumentDB 政策來控制來自特定 Amazon Virtual Private Cloud (Amazon VPC) 端點或特定 VPCs存取。實際上，這只會隔離網路中特定 VPC 對指定 Amazon DocumentDB 資源 AWS 的網路存取。

> **Note**
>
> Amazon DocumentDB 不支援以資源為基礎的存取政策。

Amazon DocumentDB API 和界面 VPC 端點 (AWS PrivateLink)

> **Note**
>
> Amazon DocumentDB 彈性叢集不支援 AWS PrivateLink VPC 端點。

您可以建立介面 VPC 端點，在 VPC 和 Amazon DocumentDB API 端點之間建立私有連線。介面端點採用 技術 AWS PrivateLink。

雖然 Amazon DocumentDB 執行個體型叢集不需要介面 VPC 端點連線，但 AWS PrivateLink 可讓您在沒有網際網路閘道、NAT 裝置、VPN 連線或 Direct Connect 連線的情況下私下存取 Amazon DocumentDB API 操作。VPC 中的 Amazon DocumentDB 執行個體不需要公有 IP 地址，即可與 Amazon DocumentDB API 端點通訊，以啟動、修改或終止資料庫執行個體和資料庫叢集。您的 Amazon DocumentDB 執行個體也不需要公有 IP 地址，即可使用任何可用的 Amazon DocumentDB API 操作。VPC 和 Amazon DocumentDB 之間的流量不會離開 Amazon 網路。

每個介面端點都由子網路中的一個或多個彈性網路介面表示。如需詳細，請參閱《Amazon EC2 使用者指南》中的彈性網絡介面。

如需 VPC 端點的詳細資訊，請參閱《Amazon Virtual Private Cloud (AWS PrivateLink) 使用者指南》中的AWS 服務 使用介面 VPC 端點存取 。如需 Amazon DocumentDB 操作的詳細資訊，請參閱 Amazon DocumentDB API 參考。

主題

- VPC 端點的考量事項
- 區域可用性
- 為 Amazon DocumentDB API 建立介面 VPC 端點
- 為 Amazon DocumentDB API 建立 VPC 端點政策

VPC 端點的考量事項

為 Amazon DocumentDB API 端點設定介面 VPC 端點之前，請務必檢閱《Amazon Virtual Private Cloud (AWS PrivateLink) 使用者指南》中的介面端點先決條件。

所有與管理 Amazon DocumentDB 資源相關的 Amazon DocumentDB API 操作都可從您的 VPC 使用取得 AWS PrivateLink。

Amazon DocumentDB API 端點支援 VPC 端點政策。根據預設，允許透過端點完整存取 Amazon DocumentDB API 操作。如需詳細資訊，請參閱《Amazon Virtual Private Cloud (AWS PrivateLink) 使用者指南》中的使用端點政策控制對 VPC 端點的存取。

區域可用性

Amazon DocumentDB API 目前支援下列 VPC 端點 AWS 區域：

- 美國東部 (俄亥俄)
- 美國東部 (維吉尼亞北部)
- 美國西部 (奧勒岡)
- 非洲 (開普敦)
- 亞太區域 (香港)
- 亞太地區 (孟買)
- 亞太地區 (海德拉巴)
- 亞太地區 (大阪)
- 亞太區域 (首爾)
- 亞太區域 (新加坡)
- 亞太地區 (雪梨)
- 亞太區域 (東京)
- 加拿大 (中部)

- 中國 (北京)
- 中國 (寧夏)
- 歐洲 (法蘭克福)
- 歐洲 (愛爾蘭)
- 歐洲 (倫敦)
- Europe (Paris)
- 歐洲 (西班牙)
- 歐洲 (米蘭)
- 中東 (阿拉伯聯合大公國)
- 南美洲 (聖保羅)
- AWS GovCloud （美國東部）
- AWS GovCloud （美國西部）

為 Amazon DocumentDB API 建立介面 VPC 端點

您可以使用 Amazon VPC 主控台或 () 為 Amazon DocumentDB API 建立 VPC 端點AWS CLI。 AWS Command Line Interface 如需詳細資訊，請參閱《Amazon Virtual Private Cloud (AWS PrivateLink) 使用者指南》中的AWS 服務 使用介面 VPC 端點存取 。

使用服務名稱 為 Amazon DocumentDB API 建立 VPC 端點`com.amazonaws.region.rds`。

AWS 區域 在中國除外，如果您為端點啟用私有 DNS，則可以使用 的預設 DNS 名稱向 Amazon DocumentDB 提出 API 請求，例如 AWS 區域 rds.us-east-1.amazonaws.com。對於中國 （北京） 和中國 （寧夏） AWS 區域，您可以分別使用 rds-api.cn-north-1.amazonaws.com.cn 和 rds-api.cn-northwest-1.amazonaws.com.cn 使用 VPC 端點提出 API 請求。

如需詳細資訊，請參閱《Amazon Virtual Private Cloud (AWS PrivateLink) 使用者指南》中的AWS 服務 使用介面 VPC 端點存取 。

為 Amazon DocumentDB API 建立 VPC 端點政策

您可以將端點政策連接至 VPC 端點，以控制對 Amazon DocumentDB API 的存取。此政策會指定下列資訊：

- 可執行動作的主體。

- 可執行的動作。
- 可供執行動作的資源。

如需詳細資訊，請參閱《Amazon Virtual Private Cloud (AWS PrivateLink) 使用者指南》中的使用端點政策控制對 VPC 端點的存取。

範例：Amazon DocumentDB API 動作的 VPC 端點政策

以下是 Amazon DocumentDB API 的端點政策範例。連接到端點時，此政策會授予所有資源上所有主體的所列 Amazon DocumentDB API 動作的存取權。

```
{
"Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "docdb:CreateDBInstance",
            "docdb:ModifyDBInstance",
            "docdb:CreateDBSnapshot"
         ],
         "Resource":"*"
      }
   ]
}
```

範例：拒絕來自指定 AWS 帳戶的所有存取的 VPC 端點政策

下列 VPC 端點政策拒絕 AWS 帳戶 123456789012 使用端點存取資源。此政策允許來自其他帳戶的所有動作。

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
```

```
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
   ]
}
```

Amazon DocumentDB 的安全最佳實務

Amazon DocumentDB 提供安全功能，包括 Amazon VPC 隔離、使用 進行靜態加密 AWS KMS、使用 TLS 傳輸中的加密，以及透過 IAM 進行精細存取控制。使用下列最佳實務來保護 Amazon DocumentDB 叢集、執行個體和資料。

使用 AWS Identity and Access Management (IAM) 角色來控制對 Amazon DocumentDB API 操作的存取。這包括建立、修改或刪除 Amazon DocumentDB 資源的操作，例如叢集、安全群組和參數群組。當應用程式或其他 AWS 服務需要存取 Amazon DocumentDB 資源時，請使用 IAM 角色提供暫時登入資料，而不是將長期登入資料存放在您的應用程式中。建立 IAM 角色時，請套用最低權限原則。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 角色和角色的常見案例。

- 使用角色類型存取控制強制執行最低權限。
- 請勿使用 AWS 帳戶 根使用者來管理 Amazon DocumentDB 資源。反之，請使用 AWS Identity and Access Management Identity Center 建立具有臨時登入資料的人工使用者。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM Identity Center 使用者。
- 授予每個身分執行其職責所需的最低許可集。
- 使用 IAM 政策來管理許可，並遵循最低權限原則。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的安全性最佳實務。
- 如果您使用具有長期登入資料的 IAM 使用者，請定期輪換這些登入資料。

> **Warning**
>
> 此種情況需要具備程式化存取權限且擁有長期憑證的 IAM 使用者，此舉將構成安全風險。為了協助降低此風險，建議您只為這些使用者提供執行任務所需的許可，並在不再需要這些使用者時將其移除。必要時可以更新存取金鑰。如需詳細資訊，請參閱《IAM 使用者指南》中的更新存取金鑰。

- 設定 Secrets Manager 以自動輪換 Amazon DocumentDB 的秘密。如需詳細資訊，請參閱AWS Secrets Manager 《 使用者指南》中的輪換您的秘密和輪換 Amazon DocumentDB 的秘密。
- 使用 Transport Layer Security (TLS) 和靜態加密來加密您的資料。

稽核 Amazon DocumentDB 事件

透過 Amazon DocumentDB （與 MongoDB 相容），您可以稽核在叢集中執行的事件。已記錄事件的範例包括成功和失敗的身分驗證嘗試、在資料庫中放入集合，或建立索引。根據預設，稽核會在 Amazon DocumentDB 上停用，並要求您選擇使用此功能。

啟用稽核時，Amazon DocumentDB 會將資料定義語言 (DDL)、資料處理語言 (DML)、身分驗證、授權和使用者管理事件記錄到 Amazon CloudWatch Logs。啟用稽核時，Amazon DocumentDB 會將叢集的稽核記錄 (JSON 文件） 匯出至 Amazon CloudWatch Logs。您可以使用 Amazon CloudWatch Logs 來分析、監控和封存 Amazon DocumentDB 稽核事件。

雖然 Amazon DocumentDB 不會收取啟用稽核的額外費用，但您需要支付使用 CloudWatch Logs 的標準費率。如需 CloudWatch Logs 定價的資訊，請參閱 Amazon CloudWatch 定價。

Amazon DocumentDB 稽核功能與 監控的服務資源用量明顯不同 AWS CloudTrail。CloudTrail 會記錄使用 AWS Command Line Interface (AWS CLI) 或在叢集、執行個體、參數群組和快照等資源 AWS 管理主控台 上執行的操作。使用 CloudTrail 稽核資源預設為開啟，且無法停用。Amazon DocumentDB 稽核功能是一種選擇加入功能。它記錄您叢集內發生於物件的操作，例如資料庫、集合、索引和使用者。

主題

- 支援的事件
- 啟用稽核
- 停用稽核
- 存取您的稽核事件
- 篩選 DML 稽核事件

支援的事件

Amazon DocumentDB 稽核支援下列事件類別：

- 資料定義語言 (DDL) - 包括資料庫管理操作、連線、使用者管理和授權。
- 資料處理語言讀取事件 (DML 讀取） - 包括 `find()`和各種彙總運算子、算術運算子、布林運算子和其他讀取查詢運算子。
- 資料處理語言寫入事件 (DML 寫入） - 包含 `insert()`, `update()`, `delete()`,和 `bulkWrite()`運算子

事件類型如下。

事件類型	類別	描述
authCheck	Authorization	結果代碼 0：成功
		結果碼 13：未經授權嘗試執行 操作。
authenticate	連線	成功或失敗的新連線身分驗證嘗試。
auditConfigure	DDL	稽核篩選條件組態。
createDatabase	DDL	建立新的資料庫。
createCollection	DDL	在資料庫內建立新的集合。
createIndex	DDL	在集合內建立新的索引。
dropCollection	DDL	在資料庫內放入集合。
dropDatabase	DDL	放入資料庫。
dropIndex	DDL	在集合內放入索引。
modifyChangeStreams	DDL	已建立變更串流。
renameCollection	DDL	重新命名資料庫中的集合。
createRole	角色管理	建立角色。
dropAllRolesFromDatabase	角色管理	捨棄資料庫中的所有角色。
dropRole	角色管理	捨棄角色。

事件類型	類別	描述
grantPrivilegesToRole	角色管理	將權限授予角色。
grantRolesToRole	角色管理	將角色授予使用者定義的角色。
revokePrivilegesFromRole	角色管理	從角色撤銷權限。
revokeRolesFromRole	角色管理	從使用者定義的角色撤銷角色。
updateRole	角色管理	更新角色。
createUser	使用者管理	建立新的使用者。
dropAllUsersFromDatabase	使用者管理	在資料庫內放入所有使用者。
dropUser	使用者管理	放入現有的使用者。
grantRolesToUser	使用者管理	將角色授予使用者。
revokeRolesFromUser	使用者管理	從使用者撤銷角色。
updateUser	使用者管理	更新現有的使用者。
insert	DML 寫入	將文件或文件插入集合。
delete	DML 寫入	從集合中刪除文件或文件。
update	DML 寫入	修改集合中的現有文件或文件。
bulkWrite	DML 寫入	使用 控制執行順序來執行多個寫入操作。
setAuditConfig	DML 寫入	設定 DML 稽核的新篩選條件。

事件類型	類別	描述
count	DML 讀取	傳回符合集合或檢視之 find() 查詢的文件計數。
countDocuments	DML 讀取	傳回符合集合或檢視查詢的文件計數。
find	DML 讀取	選取集合或檢視中的文件，並將游標傳回至選取的文件。
getAuditConfig	DML 讀取	擷取目前篩選條件以進行 DML 稽核。
findAndModify	DML 讀取和 DML 寫入	修改並傳回單一文件。
findOneAndDelete	DML 讀取和 DML 寫入	根據篩選條件和排序條件刪除單一文件，並傳回已刪除的文件。
findOneAndReplace	DML 讀取和 DML 寫入	根據指定的篩選條件取代單一文件。
findOneAndUpdate	DML 讀取和 DML 寫入	根據篩選條件和排序條件更新單一文件。
aggregate	DML 讀取和 DML 寫入	支援彙總管道中的 APIs。
distinct	DML 讀取	尋找單一集合或檢視中指定欄位的不同值，並在陣列中傳回結果。

> **Note**
>
> DML 事件文件參數欄位中的值有 1KB 的大小限制。如果值超過 1KB，Amazon DocumentDB 會截斷該值。

> **Note**
>
> 目前不會稽核 TTL 刪除事件。

啟用稽核

在叢集上啟用稽核的程序包含兩個步驟。確定兩個步驟都已完成，否則稽核日誌將不會傳送至 CloudWatch Logs。

步驟 1. 啟用 audit_logs 叢集參數

若要啟用稽核，您需要修改`audit_logs`參數群組中的 參數。 `audit_logs` 是以逗號分隔的事件清單。事件必須以小寫指定，且清單元素之間不應有空格。

您可以為參數群組設定下列值：

Value	說明	
`ddl`	設定此項目將啟用 DDL 事件的稽核，例如 createDatabase、dropDatabase、createCollection、dropCollection、createIndex、dropIndex、authCheck、 驗證、createUser、dropUser、grantRolesToUser、 revokeRolesFromUse	

Value	說明	
	r、updateUser 和 dropAllUsersFromDatabase	
dml_read	設定此項目將啟用 DML 讀取事件的稽核，例如 尋找、排序計數、相異、群組、專案a、展開、geoNear、geoIntersects、geoWithin 和其他 MongoDB 讀取查詢運算子。	
dml_write	設定此項目將啟用 DML 寫入事件的稽核，例如 insert()、update()、Delete() 和 bulkWrite()	
all	設定此項目將啟用資料庫事件的稽核，例如讀取查詢、寫入查詢、資料庫動作和管理員動作。	
none	設定此項目將停用稽核	

Value	說明	
`enabled` (舊版)	這是相當於 'ddl' 的舊版參數設定。設定此項目將啟用 DDL 事件的稽核，例如 createDatabase、dropDatabase、createCollection、dropCollection、createIndex、dropIndex、authCheck、 驗證、createUser、dropUser、grantRolesToUser、 revokeRolesFromUser、updateUser 和 dropAllUsersFromDatabase。我們不建議使用此設定，因為它是舊版設定。	
`disabled` （舊版）	這是相當於「無」的舊版參數設定。我們不建議使用此設定，因為它是舊版設定。	

> **Note**
>
> audit_logs 叢集參數的預設值為 none（舊版 "`disabled`")。

您也可以將上述值組合使用。

Value	說明	
`ddl, dml_read`	設定此項目將啟用 DDL 事件和 DML 讀取事件的稽核。	
`ddl, dml_write`	設定此項目將啟用 DDL 事件和 DML 寫入的稽核	
`dml_read, dml_write`	設定此項目將啟用所有 DML 事件的稽核	

> **Note**
>
> 您無法修改預設參數群組。

如需詳細資訊，請參閱下列內容：

- 建立 Amazon DocumentDB 叢集參數群組

  建立自訂參數群組之後，修改為將 `audit_logs` 參數值變更為 `all`。
- 修改 Amazon DocumentDB 叢集參數群組

步驟 2. 啟用 Amazon CloudWatch Logs 匯出

當`audit_logs`叢集參數的值為 `enabled`、`dml_read`、 `ddl`或 `dml_write`時，您還必須啟用 Amazon DocumentDB 將日誌匯出至 Amazon CloudWatch。如果您省略其中一個步驟，稽核日誌將不會傳送至 CloudWatch。

建立叢集、執行point-in-time-restore或還原快照時，您可以依照下列步驟啟用 CloudWatch Logs。

Using the AWS 管理主控台

若要使用主控台啟用 Amazon DocumentDB 匯出日誌至 CloudWatch，請參閱下列主題：

- 建立叢集時 — 在 中使用 建立叢集和主要執行個體 AWS 管理主控台，請參閱建立叢集：其他組態 （步驟 5，日誌匯出)
- 修改現有叢集時 — 修改 Amazon DocumentDB 叢集
- 執行叢集快照還原時 — 從叢集快照還原
- 執行point-in-time還原時 — 還原至某個時間點

Using the AWS CLI

建立新叢集時啟用稽核日誌

下列程式碼會建立叢集`sample-cluster`並啟用 CloudWatch 稽核日誌。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb create-db-cluster \
    --db-cluster-identifier sample-cluster \
    --port 27017 \
    --engine docdb \
    --master-username master-username \
    --master-user-password password \
    --db-subnet-group-name default \
    --enable-cloudwatch-logs-exports audit
```

針對 Windows：

```
aws docdb create-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --port 27017 ^
    --engine docdb ^
    --master-username master-username ^
    --master-user-password password ^
    --db-subnet-group-name default ^
    --enable-cloudwatch-logs-exports audit
```

修改現有叢集時啟用稽核日誌

下列程式碼會修改叢集`sample-cluster`並啟用 CloudWatch 稽核日誌。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster \
   --db-cluster-identifier sample-cluster \
   --cloudwatch-logs-export-configuration '{"EnableLogTypes":["audit"]}'
```

針對 Windows：

```
aws docdb modify-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --cloudwatch-logs-export-configuration '{"EnableLogTypes":["audit"]}'
```

這些操作的輸出類似以下內容 (JSON 格式)。

```
{
    "DBCluster": {
        "HostedZoneId": "ZNKXH85TT8WVW",
        "StorageEncrypted": false,
        "DBClusterParameterGroup": "default.docdb4.0",
        "MasterUsername": "<user-name>",
        "BackupRetentionPeriod": 1,
        "Port": 27017,
        "VpcSecurityGroups": [
            {
                "Status": "active",
                "VpcSecurityGroupId": "sg-77186e0d"
            }
        ],
        "DBClusterArn": "arn:aws:rds:us-east-1:900083794985:cluster:sample-cluster",
        "Status": "creating",
        "Engine": "docdb",
        "EngineVersion": "4.0.0",
        "MultiAZ": false,
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1c",
            "us-east-1f"
        ],
        "DBSubnetGroup": "default",
        "DBClusterMembers": [],
```

```
        "ReaderEndpoint": "sample-cluster.cluster-ro-corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
        "EnabledCloudwatchLogsExports": [
            "audit"
        ],
        "PreferredMaintenanceWindow": "wed:03:08-wed:03:38",
        "AssociatedRoles": [],
        "ClusterCreateTime": "2019-02-13T16:35:04.756Z",
        "DbClusterResourceId": "cluster-YOS52CUXGDTNKDQ7DH72I4LED4",
        "Endpoint": "sample-cluster.cluster-corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
        "PreferredBackupWindow": "07:16-07:46",
        "DBClusterIdentifier": "sample-cluster"
    }
}
```

停用稽核

您可以透過停用 CloudWatch Logs 匯出和停用 `audit_logs` 參數來停用稽核。

停用 CloudWatch Logs 匯出

您可以使用 AWS 管理主控台 或 來停用匯出稽核日誌 AWS CLI。

Using the AWS 管理主控台

下列程序使用 AWS 管理主控台 來停用 Amazon DocumentDB 匯出日誌至 CloudWatch。

停用稽核日誌

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。接著對於您要停止匯出日誌的叢集，選擇其名稱左側的按鈕。
3. 選擇 Actions (動作)，然後選擇 Modify (修改)。
4. 向下捲動到 Log exports (日誌匯出) 區段，然後選擇 Disabled (停用)。
5. 選擇繼續。
6. 檢閱您的變更，然後選擇希望叢集套用此變更的時機。
   - Apply during the next scheduled maintenance window (在下次排定的維護時段套用)
   - Apply immediately (立即套用)

7. 選擇修改叢集。

Using the AWS CLI

下列程式碼會修改叢集`sample-cluster`並停用 CloudWatch 稽核日誌。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster \
   --db-cluster-identifier sample-cluster \
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["audit"]}'
```

針對 Windows：

```
aws docdb modify-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["audit"]}'
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "DBCluster": {
        "DBClusterParameterGroup": "default.docdb4.0",
        "HostedZoneId": "ZNKXH85TT8WVW",
        "MasterUsername": "<user-name>",
        "Status": "available",
        "Engine": "docdb",
        "Port": 27017,
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1c",
            "us-east-1f"
        ],
        "EarliestRestorableTime": "2019-02-13T16:35:50.387Z",
        "DBSubnetGroup": "default",
        "LatestRestorableTime": "2019-02-13T16:35:50.387Z",
        "DBClusterArn": "arn:aws:rds:us-east-1:900083794985:cluster:sample-
cluster2",
        "Endpoint": "sample-cluster2.cluster-corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
```

```
        "ReaderEndpoint": "sample-cluster2.cluster-ro-corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
        "BackupRetentionPeriod": 1,
        "EngineVersion": "4.0.0",
        "MultiAZ": false,
        "ClusterCreateTime": "2019-02-13T16:35:04.756Z",
        "DBClusterIdentifier": "sample-cluster2",
        "AssociatedRoles": [],
        "PreferredBackupWindow": "07:16-07:46",
        "DbClusterResourceId": "cluster-YOS52CUXGDTNKDQ7DH72I4LED4",
        "StorageEncrypted": false,
        "PreferredMaintenanceWindow": "wed:03:08-wed:03:38",
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "Status": "active",
                "VpcSecurityGroupId": "sg-77186e0d"
            }
        ]
    }
}
```

停用 audit_logs 參數

若要停用叢集的 audit_logs 參數，您可以修改叢集，以便它使用 audit_logs 參數值為 disabled 的參數群組。或者，您可以修改叢集參數群組中的 audit_logs 參數值，使其成為 disabled。

如需詳細資訊，請參閱下列主題：

- 修改 Amazon DocumentDB 叢集
- 修改 Amazon DocumentDB 叢集參數群組

存取您的稽核事件

使用下列步驟在 Amazon CloudWatch 上存取您的稽核事件。

1. 透過 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 請確定您位於與 Amazon DocumentDB 叢集相同的區域。
3. 在導覽窗格中，選擇日誌。

4. 若要尋找叢集的稽核日誌，請從清單尋找並選擇 **/aws/docdb/*yourClusterName*/audit**。

   您可在每個執行個體名稱下分別取得每個執行個體的稽核事件。

篩選 DML 稽核事件

DML 稽核篩選入門

DML 稽核事件可在寫入 Amazon CloudWatch 之前進行篩選。若要使用此功能，必須啟用稽核日誌和 DML 記錄。Amazon DocumentDB 支援篩選 `atype`、`command`、`namespace`、 `user`和 `auditAuthorizationSuccess`。

> **Note**
>
> 不會篩選 DDL 事件。

您可以隨時使用 `db.adminCommand( { command } )`操作中的 `setAuditConfig`、 和 `auditAuthorizationSuccess` 參數來指定稽核篩選條件`filter`，以啟用稽核篩選：

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter:
         {
            //filter conditions
         },
      auditAuthorizationSuccess: true | false
   }
)
```

您也可以執行下列命令來擷取稽核篩選條件設定：

```
db.admin.runCommand( { getAuditConfig: 1})
```

安全需求

只有具有特殊權限動作的資料庫使用者/角色，`auditConfigure`才能在設定或列出 DML 稽核篩選條件`admindb`時對 執行上述命令。您可以使用 【`clusterAdmin`、、`root`】 的其中一

個內建角色hostManager，或建立具有 auditConfigure 權限的自訂角色。以下是使用具有 auditConfigure權限的現有角色和具有自訂角色的範例的範例。

具有內建角色的使用者：

```
use admin
db.createUser(
  {
    user: "myClusterAdmin",
    pwd: "password123",
    roles: [ { role: "clusterAdmin", db: "admin" } ]
  }
)
```

具有自訂角色的使用者：

```
use admin
db.createRole(
   {
     role: "myRole",
     privileges: [
       { resource: { cluster: true }, actions: [ "auditConfigure" ] }
     ],
     roles: []
   }
)
db.createUser(
  {
    user: "myUser",
    pwd: "myPassword",
    roles: [ { role: "myRole", db: "admin" } ]
  }
)
```

篩選使用案例

範例：依命令篩選事件

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
```

```
        "$and": [
         {
            "param.command":
               {
                  $in: [ "find","count", "insert", "delete", "update",
 "findandmodify" ]
               }
         }
         ]
      },
      auditAuthorizationSuccess: true
   }
)
```

範例：依使用者名稱篩選事件

在此範例中，只會記錄使用者「myUser」：

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
      "$and": [
         {
            "param.user":
               {
                  $in: [ "myUser" ]
               }
         }
         ]},
      auditAuthorizationSuccess: true})
```

範例：依 篩選 `atype`

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {atype: "authCheck"},
      auditAuthorizationSuccess: true
   })
```

> **Note**
>
> 所有 DML 日誌都有 authCheck做為 atype。只有 DDL 具有不同的 atype。如果您在 authCheck中放置 以外的值filter，則不會在 CloudWatch 中產生 DML 日誌。

範例：使用運算子加入的多個篩選條件進行篩選

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
      "$and": [
         {
            "param.command":
               {
                  $in: [ "find","count", "insert", "delete", "update",
 "findandmodify" ]
               }
         }
         ],
       "$nor": [
         {
            "param.command":
               {
                  $in: ["count", "insert", "delete", "update", "findandmodify" ]
               }
         }]
       },
      auditAuthorizationSuccess: true})
```

> **Note**
>
> 在頂層，僅$nor支援 $and、 $or和 。不支援任何其他運算子，並會導致錯誤。

範例：依事件篩選 **auditAuthorizationSuccess**

在此篩選條件中，不會記錄成功通過授權的所有命令：

```
db.admin.runCommand(
```

```
   {
      setAuditConfig: 1,
      filter: {},
      auditAuthorizationSuccess: false
   }
)
```

範例：使用 $in和 $nin條件進行篩選

在 $in和 中使用 時$nin，不會記錄命令，因為條件之間會有隱含的「和」。在此範例中，regex 會封鎖find命令，因此不會記錄任何內容：

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
      "$and": [
         {
            atype: "authCheck",
            "param.command":
               {
                  $in: [ "find", "insert", "delete", "update", "findandmodify" ],
                  $nin: ["count", "insert", "delete", "update", "findandmodify" ],
                  $not: /^find.*/
               }
         },
         ],
       "$or": [
         {
            "param.command":
               {
                  $nin: ["count", "insert", "delete", "update", "findandmodify" ]
               }
         }]
       },
      auditAuthorizationSuccess: true})
```

範例：依 篩選 namespace

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
```

```
      filter: {
      "$and": [
         {
            "param.ns":
               {
                  $in: [ "test.foo" ]
               }
         }
         ]},
      auditAuthorizationSuccess: true})
```

範例：重設為預設篩選條件

重設為預設值表示將記錄每個 DML 稽核事件。若要將篩選重設為預設值，請執行下列命令：

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {},
      auditAuthorizationSuccess: true
   }
)
```

Amazon VPC 和 Amazon DocumentDB

Amazon Virtual Private Cloud (Amazon VPC) 可讓您在虛擬私有雲端 (VPC) 中啟動 AWS 資源，例如 Amazon DocumentDB 執行個體。

使用 VPC 時，您可以掌控您的虛擬聯網環境。您可以選擇自己的 IP 地址範圍、建立子網路，以及設定路由和存取控制清單。在 VPC 中執行叢集無需額外費用。

帳戶有預設 VPC。除非您另有指定，否則所有新叢集都會在預設 VPC 中建立。

主題

- VPC 中的 DocumentDB 叢集
- 在 VPC 中存取 Amazon DocumentDB 叢集
- 建立IPv4-only的 VPC 以與 DocumentDB 叢集搭配使用
- 建立雙堆疊 VPC 以與 DocumentDB 叢集搭配使用

您可以在下面找到與 Amazon DocumentDB 叢集相關的 VPC 功能的討論。如需 Amazon VPC 的詳細資訊，請參閱《Amazon VPC 使用者指南》https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html。

VPC 中的 DocumentDB 叢集

您的 Amazon DocumentDB 叢集位於虛擬私有雲端 (VPC) 中。VPC 是一種虛擬網路，在邏輯上與中的其他虛擬網路隔離 AWS 雲端。Amazon VPC 可讓您在 VPC 中啟動 AWS 資源，例如 Amazon DocumentDB 叢集或 Amazon EC2 執行個體。VPC 可以是您帳戶隨附的預設 VPC，或是您自行建立的 VPC。所有 VPCs都與 AWS 您的帳戶相關聯。

您的預設 VPC 有三個子網路，可供您在 VPC 內隔離資源。預設的 VPC 也有網際網路閘道，只要利用網際網路閘道，即可允許從 VPC 以外的位置存取 VPC 內的資源。

如需在 VPC 中和 VPC 外部涉及 Amazon DocumentDB 叢集的案例，請參閱 在 VPC 中存取 Amazon DocumentDB 叢集。

主題

- 在 VPC 中使用叢集
- 使用子網路群組
- Amazon DocumentDB IP 定址
- 在 VPC 中建立叢集

在 VPC 中使用叢集

以下是在 VPC 中使用叢集的一些秘訣：

- 您的 VPC 必須至少要有兩個子網路。這些子網路必須位於 AWS 區域 您要部署叢集的 中的兩個不同可用區域。子網路是 VPC IP 地址範圍的區段，您可以指定該區段，而且您可以根據安全性和操作需求來分組叢集。
- 如果您希望 VPC 中的叢集可公開存取，請務必開啟 VPC 屬性 `DNS hostnames`和 `DNS resolution`。
- 您的 VPC 必須擁有您建立的子網路群組。您可以透過指定您建立的子網路來建立子網路群組。Amazon DocumentDB 會在該子網路群組內選擇要與叢集中主要執行個體建立關聯的子網路和 IP 地址。主要執行個體使用包含子網路的可用區域。
- 您的 VPC 必須具有允許存取叢集的 VPC 安全群組。

  如需詳細資訊，請參閱在 VPC 中存取 Amazon DocumentDB 叢集。

- 每個子網路中的 CIDR 區塊必須大到足以容納 Amazon DocumentDB 在維護活動期間使用的備用 IP 地址，包括容錯移轉和運算擴展。例如，10.0.0.0/24 和 10.0.1.0/24 等範圍通常就夠大了。
- VPC 可以具有 `instance tenancy` `default`或 的屬性`dedicated`。所有預設 VPCs都會將 `instance tenancy` 屬性設定為預設，且預設 VPC 可以支援任何執行個體類別。

  如果您選擇在`instance tenancy`屬性設定為 的專用 VPC 中擁有叢集`dedicated`，則叢集的執行個體類別必須是已核准的 Amazon EC2 專用執行個體類型之一。例如，`r5.large`EC2 專用執行個體對應至`db.r5.large`執行個體類別。如需 VPC 中執行個體租用的相關資訊，請參閱《Amazon Elastic Compute Cloud 使用者指南》中的 Amazon EC2 執行個體。

  如需進一步了解專用執行個體中可能出現的執行個體類型，請參閱 Amazon EC2 定價頁面上的 Amazon EC2 專用執行個體。

  > **Note**
  >
  > 當您將叢集`dedicated`的 屬性設定為 `instance tenancy` 時，不保證叢集會在專用主機上執行。

使用子網路群組

Subnets (子網路) 是可供您指定之 VPC 的 IP 地址範圍區段，您可根據安全與運作需求將資源分組。子網路群組是您在 VPC 中建立，然後為叢集指定的子網路集合（通常是私有）。透過使用子網路群組，您可以在使用 AWS CLI 或 Amazon DocumentDB API 建立叢集時指定特定 VPC。如果使用主控台，您可以選擇要使用的 VPC 和子網路群組。

每個子網路群組在指定的可用區域中應至少有兩個子網路 AWS 區域。在 VPC 中建立叢集時，您可以選擇其子網路群組。從子網路群組中，Amazon DocumentDB 會選擇子網路和該子網路內的 IP 地址，以與叢集中的主要執行個體建立關聯。資料庫使用包含子網路的可用區域。DocumentDB 一律會從具有可用 IP 地址空間的子網路指派 IP 地址。

子網路群組中的子網路為公有或私有，取決於您為其網路存取控制清單（網路 ACLs) 和路由表設定的組態。若要讓叢集可公開存取，其子網路群組中的所有子網路都必須是公開的。如果與可公開存取叢集相關聯的子網路從公有變更為私有，可能會影響叢集可用性。

若要建立支援雙堆疊模式的子網路群組，請確定您新增至子網路群組的每個子網路都有與其相關聯的網際網路通訊協定第 6 版 (IPv6) CIDR 區塊。如需詳細資訊，請參閱《Amazon Virtual Private Cloud 使用者指南》中的 Amazon DocumentDB IP 定址和 VPC IPv6 支援。 IPv6

當 Amazon DocumentDB 在 VPC 中建立叢集時，它會使用子網路群組的 IP 地址，將網路介面指派給叢集。一律使用網域名稱系統 (DNS) 名稱來連線至叢集，因為基礎 IP 地址會在容錯移轉期間變更。

> **Note**
>
> 對於您在 VPC 中執行的每個叢集，請務必在子網路群組中的每個子網路中至少保留一個地址，以供 Amazon DocumentDB 用於復原動作。

共用子網路

您可以在共用 VPC 中建立叢集。

使用共用 VPC 時要記住的一些考量：

- 您可以將叢集從共用 VPC 子網路移至非共用 VPC 子網路，反之亦然。
- 共用 VPC 中的參與者必須在 VPC 中建立安全群組，以允許他們建立叢集。
- 共用 VPC 中的擁有者和參與者可以使用 DocumentDB 查詢來存取資料庫。不過，只有資源的建立者才能對資源進行任何 API 呼叫。

Amazon DocumentDB IP 定址

IP 地址可讓您 VPC 中的資源彼此互相通訊，也能和網際網路上的資源通訊。Amazon DocumentDB 同時支援 IPv4 和 IPv6 定址通訊協定。根據預設，Amazon DocumentDB 和 Amazon VPC 會使用 IPv4 定址通訊協定。您無法關閉此行為。當您建立 VPC 時，請務必指定 IPv4 CIDR 區塊 (私有 IPv4 地址的範圍)。您可以選擇性地將 IPv6 CIDR 區塊指派給 VPC 和子網路，並將該區塊的 IPv6 地址指派給子網路中的叢集。

> **Note**
>
> 僅 Amazon DocumentDB 4.0 和 5.0 版支援雙堆疊模式 (IPv6 定址）。

對 IPv6 通訊協定的支援擴展受支援的 IP 地址的數量。使用 IPv6 通訊協定，您可以確保擁有足夠的可用地址，可應對網際網路的未來發展。新的和現有的 DocumentDB 資源可以在 VPC 中使用 IPv4 和 IPv6 地址。在應用程式的不同部分中使用的兩個通訊協定之間進行設定、保護和轉譯網路流量可能會造成操作額外負荷。您可以標準化 Amazon DocumentDB 資源的 IPv6 通訊協定，以簡化網路組態。

主題

- IPv4 地址
- IPv6 地址
- 雙堆疊模式

IPv4 地址

在您建立 VPC 時，必須以 CIDR 區塊的形式為 VPC 指定 IPv4 地址的範圍，例如 `10.0.0.0/16`。子網路群組定義此 CIDR 區塊中叢集可以使用的 IP 地址範圍。這些 IP 地址可為私有或公有。

私有 IPv4 地址是無法在網際網路存取的 IP 地址。您可以在相同的 VPC 中使用私有 IPv4 地址在叢集和其他資源之間進行通訊，例如 Amazon EC2 執行個體。每個叢集都有用於 VPC 中通訊的私有 IP 地址。

公有 IP 地址是可從網際網路存取的 IPv4 地址。DocumentDB 叢集不允許公有 IP 定址。任何公有 IP 地址都應由網際網路閘道和公有子網路中的 EC2 解析。

若要了解如何建立僅具有私有 IPv4 地址的 VPC，可用於常見的 Amazon DocumentDB 案例，請參閱建立IPv4-only的 VPC 以與 DocumentDB 叢集搭配使用。

IPv6 地址

您可以選擇性的將 IPv6 CIDR 區塊與您的 VPC 和子網路建立關聯，並指派該區塊的 IPv6 地址給您 VPC 中的資源。每個 IPv6 地址都是全域唯一的。

您的 VPC 的 IPv6 CIDR 區塊會自動從 Amazon 的 IPv6 地址集區自動指派。您無法自行選擇範圍。

連線至 IPv6 地址時，請確保符合下列條件：

- 用戶端已設定為允許透過 IPv6 進行用戶端至資料庫的流量。
- 叢集使用的 DocumentDB 安全群組已正確設定，以便允許用戶端透過 IPv6 進行資料庫流量。
- 用戶端作業系統堆疊允許 IPv6 地址上的流量，且作業系統驅動程式和程式庫已設定為選擇正確的預設叢集端點 (IPv4 或 IPv6)。

如需 IPv6 的詳細資訊，請參閱《Amazon Virtual Private Cloud 使用者指南》中的 IP 定址。

雙堆疊模式

當叢集可以透過 IPv4 和 IPv6 定址通訊協定進行通訊時，它會以雙堆疊模式執行。因此，資源可以透過 IPv4、IPv6 或兩者與叢集通訊。DocumentDB 會停用私有雙堆疊模式叢集 IPv6 端點的網際網路閘道存取。DocumentDB 這樣做是為了確保您的 IPv6 端點是私有的，並且只能在 VPC 內存取。

主題

- 雙堆疊模式和子網路群組
- 使用雙堆疊模式叢集
- 修改IPv4-only的叢集以使用雙堆疊模式
- 雙堆疊模式區域和版本可用性
- 雙堆疊網路叢集的限制

雙堆疊模式和子網路群組

若要使用雙堆疊模式，請確定與叢集建立關聯的子網路群組中的每個子網路都有與其建立關聯的 IPv6 CIDR 區塊。您可以建立新的子網路群組或修改現有的子網路群組以符合此需求。叢集處於雙堆疊模式後，用戶端可以正常連接到叢集。請確定已正確設定用戶端安全防火牆和 DocumentDB 叢集安全群組，以允許透過 IPv6 的流量。若要連線，用戶端會使用叢集的端點。用戶端應用程式可指定連線至資料庫時所偏好的通訊協定。在雙堆疊模式中，叢集會偵測用戶端偏好的網路通訊協定，即 IPv4 或 IPv6，並使用該通訊協定進行連線。

如果子網路群組因為子網路刪除或 CIDR 取消關聯而停止支援雙堆疊模式，則與子網路群組相關聯的叢集會有網路狀態不相容的風險。此外，您無法在建立新的雙堆疊模式叢集時使用子網路群組。

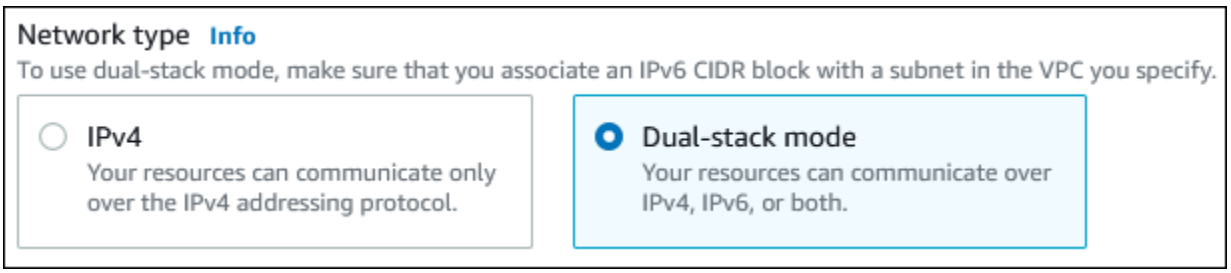
若要使用 判斷子網路群組是否支援雙堆疊模式 AWS 管理主控台，請在子網路群組的詳細資訊頁面上檢視網路類型。若要使用 判斷子網路群組是否支援雙堆疊模式 AWS CLI，請執行 `describe-db-subnet-groups`命令並在輸出`SupportedNetworkTypes`中檢視 。

僅供讀取複本視為獨立叢集，並且可以具有與主要叢集不同的網路類型。如果您變更僅供讀取複本主要叢集的網路類型，則僅供讀取複本不會受到影響。還原叢集時，您可以將叢集還原至支援的任何網路類型。

使用雙堆疊模式叢集

當您建立或修改叢集時，您可以指定雙堆疊模式，以允許資源透過 IPv4、IPv6 或兩者與叢集通訊。

當您使用 AWS 管理主控台 建立或修改叢集時，您可以在網路類型區段中指定雙堆疊模式。下圖顯示主控台中的網路類型區段：



當您使用 AWS CLI 建立或修改叢集時，請將 `--network-type`選項設定為DUAL使用雙堆疊模式。當您使用 DocumentDB API 建立或修改叢集時，請將 `NetworkType` 參數設定為 DUAL 以使用雙堆疊模式。如果指定的 DocumentDB 引擎版本或子網路群組不支援雙堆疊模式，則會傳回`NetworkTypeNotSupported`錯誤。

如需建立叢集的詳細資訊，請參閱 建立 Amazon DocumentDB 叢集。如需修改叢集的詳細資訊，請參閱 修改 Amazon DocumentDB 叢集。

若要使用主控台判斷叢集是否處於雙堆疊模式，請在叢集的連線與安全索引標籤上檢視網路類型。

修改IPv4-only的叢集以使用雙堆疊模式

您可以修改IPv4-only的叢集，以使用雙堆疊模式。若要這樣做，請變更叢集的網路類型。

建議您在維護時段期間變更 Amazon DocumentDB 叢集的網路類型。您可以使用 `modify-db-cluster`命令手動設定網路類型。

在修改叢集以使用雙堆疊模式之前，請確定其子網路群組支援雙堆疊模式。如果與叢集相關聯的子網路群組不支援雙堆疊模式，請在修改叢集時指定支援它的不同子網路群組。修改叢集的子網路群組可能會導致停機時間。

如果您在將叢集變更為使用雙堆疊模式之前修改叢集的子網路群組，請確定子網路群組在變更前後對叢集有效。

僅將 `NetworkType` 參數設定為 來執行`ModifyDBCluster`呼叫DUAL，以將網路變更為雙堆疊模式。在相同的 API 呼叫`NetworkType`中新增其他參數以及 可能會導致停機時間。若要修改多個參數，請確定已成功完成網路類型修改，然後再傳送另一個具有其他參數的 `ModifyDBCluster` 請求。

如果您在變更後無法連線至叢集，請確定已正確設定用戶端和資料庫安全防火牆和路由表，以允許流量流向所選網路上的資料庫 (IPv4 或 IPv6)。您可能還需要修改作業系統參數、程式庫或驅動程式才可使用 IPv6 地址進行連線。

修改IPv4-only的叢集以使用雙堆疊模式

1. 修改子網路群組以支援雙堆疊模式，或建立支援雙堆疊模式的子網路群組：

   a. 建立 IPv6 CIDR 區塊與 VPC 的關聯。

      如需說明，請參閱《Amazon VPC 使用者指南》中的新增或移除 CIDR 區塊至您的 VPC。

   b. 將 IPv6 CIDR 區塊連接至子網路群組中的所有子網路。

      如需說明，請參閱《Amazon Virtual Private Cloud 使用者指南》中的新增或移除 IPv6 CIDR 區塊至子網路。

   c. 確認子網路群組支援雙堆疊模式。

      如果您使用的是 AWS 管理主控台，請選取子網路群組，並確認支援的網路類型值為雙。

      如果您使用的是 AWS CLI，請執行 `describe-db-subnet-groups`命令，並確認叢集`SupportedNetworkType`的值為 `Dual`。

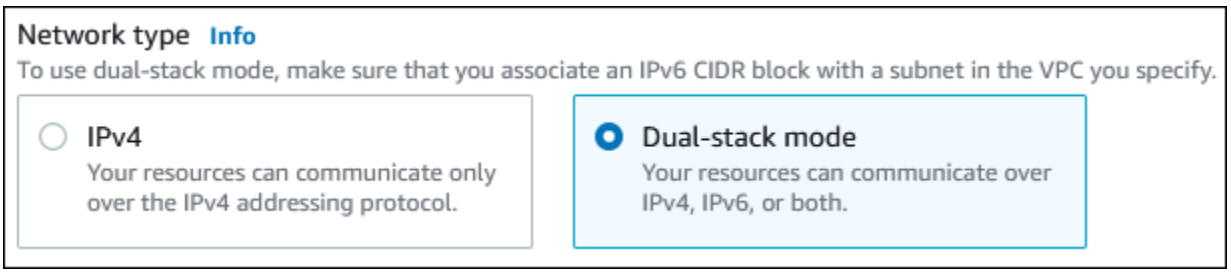
2. 修改與叢集相關聯的安全群組，以允許與資料庫的 IPv6 連線，或建立新的安全群組以允許 IPv6 連線。

   如需說明，請參閱《Amazon Virtual Private Cloud 使用者指南》中的安全群組規則。

3. 修改叢集以支援雙堆疊模式。若要這麼做，請將 Network type (網路類型) 設為 Dual-stack mode (雙堆疊模式)。

   若您使用主控台，請確保下列設定正確：

   - 網路類型 — 雙堆疊模式

   - 子網路群組 — 您在上一個步驟中設定的子網路群組
   - 安全群組 — 您在上一個步驟中設定的安全性

   如果您使用的是 AWS CLI，請確定下列設定正確：

   - `--network-type` — `dual`
   - `--db-subnet-group-name` — 您在上一個步驟中設定的子網路群組

- `--vpc-security-group-ids` — 您在上一個步驟中設定的 VPC 安全群組

例如：

```
aws docdb modify-db-cluster --db-cluster-identifier <cluster-name> --network-type
 "DUAL"
```

4. 確認叢集支援雙堆疊模式。

   如果您使用的是 主控台，請選擇叢集的連線與安全索引標籤，並確認網路類型值為雙堆疊模式。

   如果您使用的是 AWS CLI，請執行 `describe-db-cluster`命令，並確認叢集`NetworkType`的值為 `dual`。

   在叢集端點上執行 `dig`命令，以識別與其相關聯的 IPv6 地址：

   ```
   dig <db-cluster-endpoint> AAAA
   ```

   使用叢集端點，而不是 IPv6 地址來連線至叢集。

雙堆疊模式區域和版本可用性

功能的可用性和支援會有所不同 AWS 區域。

區域支援

Amazon DocumentDB 的雙堆疊模式目前適用於所有商業。 AWS 區域

版本支援

Amazon DocumentDB 4.0、5.0 和 8.0 版支援雙堆疊模式。如果您無法存取任何這些版本的雙堆疊模式，請確定您在叢集上執行的是最新的引擎修補程式版本。

雙堆疊網路叢集的限制

下列限制適用於雙堆疊網路叢集：

- 叢集無法只使用 IPv6 通訊協定。其可專門使用 IPv4，也可使用 IPv4 和 IPv6 通訊協定 (雙堆疊模式)。
- Amazon DocumentDB 不支援原生 IPv6 子網路。

- 使用雙堆疊模式的叢集必須是私有的。其無法公開存取。

在 VPC 中建立叢集

下列程序可協助您在 VPC 中建立叢集。若要使用預設 VPC，您可以從步驟 2 開始，並使用已為您建立的 VPC 和子網路群組。您也可以視需要建立其他 VPCs。

> **Note**
>
> 如果您希望 VPC 中的叢集可公開存取，則必須啟用 VPC 屬性 `DNS hostnames`和 來更新 VPC 的 DNS 資訊`DNS resolution`。如需有關更新 VPC 執行個體 DNS 的資訊，請參閱《Amazon Virtual Private Cloud 使用者指南》中的檢視和更新 VPC 的 DNS 屬性。

請依照下列步驟在 VPC 中建立叢集：

- 步驟 1：建立 VPC
- 步驟 2：建立子網路群組
- 步驟 3：建立 VPC 安全群組
- 步驟 4：在 VPC 中建立叢集

步驟 1：建立 VPC

建立在至少兩個可用區域內有子網路的 VPC。您在建立子網路群組時使用這些子網路。如有預設的 VPC，則系統會自動為您於所在 AWS 區域的每個可用區域中建立子網路。

如需詳細資訊，請參閱 建立IPv4-only的 VPC 以與 DocumentDB 叢集搭配使用，或參閱《Amazon Virtual Private Cloud 使用者指南》中的建立 VPC。

步驟 2：建立子網路群組

子網路群組是您為 VPC 建立，然後為叢集指定的子網路集合（通常為私有）。當您使用 AWS CLI 或 DocumentDB API 建立叢集時，子網路群組可讓您指定特定的 VPC。如果您使用 AWS 管理主控台，您只需選擇要使用的 VPC 和子網路。每個子網路群組在 中至少有兩個可用區域中必須至少有一個子網路 AWS 區域。最佳實務是，每個子網路群組在 中的每個可用區域應至少有一個子網路 AWS 區域。

若要公開存取叢集，子網路群組中的子網路必須具有網際網路閘道。如需子網路網際網路閘道的詳細資訊，請參閱《Amazon Virtual Private Cloud 使用者指南》中的使用網際網路閘道啟用 VPC 的網際網路存取。

> **Note**
>
> 本機區域的子網路群組只能有一個子網路。

在 VPC 中建立叢集時，您可以選擇子網路群組。Amazon DocumentDB 會選擇要與叢集建立關聯的子網路和 IP 地址（在該子網路內）。如果不存在子網路群組，Amazon DocumentDB 會在您建立叢集時建立預設子網路群組。DocumentDB 會建立彈性網路界面，並將其與叢集與該 IP 地址建立關聯。叢集使用包含子網路的可用區域。

在此步驟中，您會建立子網路群組，並新增您為 VPC 建立的子網路。

建立子網路群組

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中選擇 Subnet groups (子網路群組)。
3. 選擇建立。
4. 在名稱中，輸入子網路群組的名稱。
5. 針對描述，輸入子網路群組的描述。
6. 在新增子網路區段中，針對 VPC，選擇您建立的預設 VPC 或 VPC。然後從可用區域選擇包含子網路的可用區域，然後從子網路選擇子網路。
7. 選擇建立。

   您的新子網路群組會出現在 DocumentDB 主控台的子網路群組清單中。您可以在視窗底部的詳細資訊窗格中，選擇子網路群組以查看詳細資訊，包括與群組相關聯的所有子網路。

步驟 3：建立 VPC 安全群組

建立叢集之前，請先建立要與其建立關聯的 VPC 安全群組。如果您未建立 VPC 安全群組，您可以在建立叢集時使用預設安全群組。如需如何為叢集建立安全群組的指示，請參閱 建立IPv4-only的 VPC 以與 DocumentDB 叢集搭配使用，或參閱《Amazon Virtual Private Cloud 使用者指南》中的使用安全群組控制 AWS 資源的流量。

步驟 4：在 VPC 中建立叢集

在此步驟中，您會使用在先前步驟中建立的 VPC 名稱、子網路群組和 VPC 安全群組來建立叢集。

> **Note**
>
> 如果您希望 VPC 中的叢集可公開存取，則必須啟用 VPC 屬性 `DNS hostnames`和 `DNS resolution`。如需詳細資訊，請參閱《Amazon Virtual Private Cloud 使用者指南》中的檢視和更新 VPC 的 DNS 屬性。

如需如何建立叢集的詳細資訊，請參閱 建立 Amazon DocumentDB 叢集。

在連線區段中出現提示時，輸入 VPC 名稱、子網路群組和 VPC 安全群組。

> **Note**
>
> DocumentDBDocumentDB 叢集目前不支援更新 VPCs。

在 VPC 中存取 Amazon DocumentDB 叢集

Amazon DocumentDB 支援下列在 VPC 中存取叢集的案例：

主題

- 由相同 VPC 中的 Amazon EC2 執行個體存取之 VPC 中的叢集
- 由不同 VPC 中的 Amazon EC2 執行個體存取之 VPC 中的叢集

由相同 VPC 中的 Amazon EC2 執行個體存取之 VPC 中的叢集

VPC 中叢集的常見用途是與在相同 VPC 中 Amazon EC2 執行個體中執行的應用程式伺服器共用資料。

若要管理相同 VPC 中 EC2 執行個體與叢集之間的存取權限，最簡單的方式如下：

- 為您的叢集建立 VPC 安全群組。此安全群組可用來限制對叢集的存取。例如，您可以為此安全群組建立自訂規則，這可能會允許使用您在建立叢集時指派給叢集的連接埠，以及您用於存取叢集以進行開發或其他用途的 IP 地址進行 TCP 存取。
- 建立將包含 EC2 執行個體 (web 伺服器和用戶端) 的 VPC 安全群組。若有需要，此安全群組可允許藉由使用 VPC 路由表存取網際網路上的 EC2 執行個體。舉例來說，您可以在此安全群組上設定規則，允許 TCP 透過連接埠 22 存取 EC2 執行個體。

- 在叢集的安全群組中建立自訂規則，以允許來自您為 EC2 執行個體建立之安全群組的連線。這些規則可能會允許安全群組的任何成員存取叢集。

在單獨的可用區域中，還有一個額外的公有和私有子網路。DocumentDB 子網路群組至少需要兩個可用區域中的子網路。額外的子網路可讓您在未來輕鬆切換到多可用區域叢集部署。

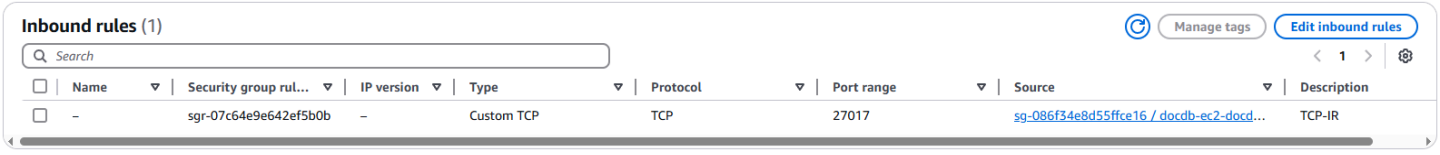
如需如何為此案例建立具有公有和私有子網路的 VPC 的說明，請參閱 建立IPv4-only的 VPC 以與 DocumentDB 叢集搭配使用。

> **Tip**
>
> 您可以在建立叢集時自動設定 Amazon EC2 執行個體與 DocumentDB 叢集之間的網路連線。如需詳細資訊，請參閱自動連接 Amazon EC2。

若要在允許來自另一個安全群組連線的 VPC 安全群組中建立規則，請執行下列動作：

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/vpc 的 Amazon VPC 主控台。
2. 在導覽窗格中，尋找並選擇安全群組。
3. 選擇或建立要允許其他安全群組成員存取的安全群組。這是您用於叢集的安全群組。選擇 Inbound rules (傳入規則) 索引標籤，然後選擇 Edit inbound rules (編輯對內規則)。
4. 在 Edit inbound rules (編輯對內規則) 頁面上，選擇 Add rule (新增規則)。
5. 針對類型，選擇與您建立叢集時所使用的連接埠對應的項目，例如自訂 TCP。
6. 在來源欄位中，開始輸入安全群組的 ID，其中列出相符的安全群組。選擇安全群組，允許其成員存取由此安全群組所保護的資源。在前一個藍本中，這是您用於 EC2 執行個體的安全群組。
7. 如有需要，請在來源欄位中建立所有 TCP 做為類型和安全群組的規則，以重複 TCP 通訊協定的步驟。若您打算使用 UDP 通訊協定，請使用 All UDP (所有 UDP) 作為 Type (類型)，並在 Source (來源) 中輸入安全群組，以建立規則。
8. 選擇儲存規則。

下列畫面顯示安全群組針對其來源的對內規則。



如需從 EC2 執行個體連線至叢集的詳細資訊，請參閱 自動連接 Amazon EC2。

由不同 VPC 中的 Amazon EC2 執行個體存取之 VPC 中的叢集

當您的叢集與您用來存取叢集的 EC2 執行個體位於不同的 VPC 中時，您可以使用 VPC 對等互連來存取叢集。

VPC 對等連線是指兩個 VPC 之間的網路連線，透過此機制，您就可以使用私有 IP 地址在兩者之間路由流量。這兩個 VPC 中的資源能彼此通訊，有如位於相同網路中一樣。您可以在自己的 VPCs 之間建立 VPC 對等互連、在另一個 AWS 帳戶中建立 VPC，或在不同的 VPC 之間建立 VPC 對等互連 AWS 區域。若要進一步了解 VPC 互連，請參閱《Amazon Virtual Private Cloud 使用者指南》中的 VPC 互連。

建立IPv4-only的 VPC 以與 DocumentDB 叢集搭配使用

常見案例包括以 Amazon VPC 服務為基礎的虛擬私有雲端 (VPC) 中的叢集。例如，此 VPC 可以與在相同 VPC 中執行的服務或應用程式共用資料。在本主題中，您會為此案例建立 VPC。

主題

- 步驟 1：建立具有私有和公有子網路的 VPC
- 步驟 2：為公有應用程式建立 VPC 安全群組
- 步驟 3：建立私有叢集的 VPC 安全群組
- 步驟 4：建立子網路群組
- 刪除 VPC

您的叢集必須僅適用於您的應用程式，而非公有網際網路。因此，您建立同時包含公有和私有子網路的 VPC。應用程式託管在公有子網路中，因此可以連線到公有網際網路。叢集託管在私有子網路中。應用程式可以連接到叢集，因為它託管在相同的 VPC 中。但是， 叢集不適用於公有網際網路，可提供更高的安全性。

本主題中的程序會在個別可用區域中設定額外的公有和私有子網路。程序不會使用這些子網路。DocumentDB 子網路群組至少需要兩個可用區域中的子網路。額外的子網路可讓您更輕鬆地設定多個 DocumentDB 執行個體。

本主題說明為 Amazon DocumentDB 叢集設定 VPC。如需 Amazon VPC 的詳細資訊，請參閱《Amazon VPC 使用者指南》。

> **Tip**
>
> 您可以在建立叢集時，自動設定 Amazon EC2 執行個體與 DocumentDB 叢集之間的網路連線。網路組態類似於本案例所述的組態。如需詳細資訊，請參閱自動連接 Amazon EC2。

步驟 1：建立具有私有和公有子網路的 VPC

使用下列程序來建立一個同時含公有和私有子網路的 VPC。

建立 VPC 和子網路

1. 開啟位於 https：//https://console.aws.amazon.com/vpc 的 Amazon VPC 主控台。
2. 在 的右上角 AWS 管理主控台，選擇要在其中建立 VPC 的區域。此範例使用 美國西部 (奧勒岡) 區域。
3. 在左上角，選擇 VPC Dashboard (VPC 儀表板)。若要開始建立 VPC，請選擇 Create VPC (建立 VPC)。
4. 在 VPC Settings (VPC 設定) 的 Resources to create (建立資源) 下，選擇 VPC and more (VPC 和更多)。
5. 對於 VPC settings (VPC 設定)，設定這些值：
   - 名稱標籤自動產生 — **`example`**
   - IPv4 CIDR 區塊 — **`10.0.0.0/16`**
   - IPv6 CIDR 區塊 — 無 IPv6 CIDR 區塊
   - 租用 — 預設
   - 可用區域 (AZs) 的數量 — 2
   - 自訂 AZs — 保留預設值
   - 公有子網路數量 — 2
   - 私有子網路數量 — 2
   - 自訂子網路 CIDR 區塊 — 保留預設值
   - NAT 閘道 ($) — 無
   - VPC 端點 — 無
   - DNS 選項 — 保留預設值
6. 選擇建立 VPC。

步驟 2：為公有應用程式建立 VPC 安全群組

接著，建立用於公開存取的安全群組。若要連線至 VPC 中的公有 EC2 執行個體，請將傳入規則新增至 VPC 安全群組。這些規則允許流量從網際網路連線。

建立 VPC 安全群組

1. 開啟位於 https：//https://console.aws.amazon.com/vpc 的 Amazon VPC 主控台。
2. 選擇 VPC Dashboard (VPC 儀表板)，再選擇 Security Groups (安全群組)，然後選擇 Create security group (建立安全群組)。
3. 在 Create security group (建立安全群組) 頁面上，設定下列值：
   - 安全群組名稱 — **example-securitygroup**
   - 描述 — **Application security group**
   - VPC — 選擇您先前建立的 VPC，例如：vpc-example。
4. 將傳入規則新增至安全群組
   a. 決定用於使用安全殼層 (SSH) 連接至 VPC 中 EC2 執行個體的 IP 地址。若要判斷公有 IP 地址，您可以在不同的瀏覽器視窗或索引標籤中使用 https://checkip.amazonaws.com 中的服務。IP 地址的範例為 `203.0.113.25/32`。

      在許多情況下，您可能透過網際網路服務供應商 (ISP) 或是從沒有靜態 IP 地址的防火牆進行連線。若是如此，請找出用戶端電腦所使用的 IP 地址範圍。

      > ⚠ Warning
      >
      > 如果您使用 `0.0.0.0/0` 進行 SSH 存取，則可讓所有 IP 地址使用 SSH 存取您的公有執行個體。通常在測試環境中短暫使用此方法是沒有問題的，但在生產環境則不安全。在生產環境中，您應只授權特定 IP 地址或特定範圍的地址可使用 SSH 存取您的執行個體。

   b. 在 Inbound rules (傳入規則) 區段中，選擇 Add rule (新增規則)。
   c. 針對您的新傳入規則設定下列值，以允許透過 SSH 存取您的 Amazon EC2 執行個體。執行此操作後，您可以連線至 EC2 執行個體來安裝應用程式和其他公用程式。您也可以連線到 EC2 執行個體，為您的應用程式上傳內容。
      - 類型 — **SSH**
      - 來源 — 您從步驟 a 建立的 IP 地址或範圍，例如： **203.0.113.25/32**

d. 選擇新增規則。

e. 為您的新傳入規則設定下列值，以允許 HTTP 存取您的應用程式：

- 類型 — **HTTP**
- 來源 — **0.0.0.0/0**

5. 請選擇 Create security group (建立安全群組) 以建立安全群組。

請注意安全群組 ID，因為您稍後在另一個程序中需要它。

步驟 3：建立私有叢集的 VPC 安全群組

若要保持叢集的私密性，請建立第二個安全群組以進行私有存取。若要連線到 VPC 中的私有叢集，請將傳入規則新增至 VPC 安全群組，只允許來自應用程式的流量。

建立 VPC 安全群組

1. 開啟位於 https：//https://console.aws.amazon.com/vpc 的 Amazon VPC 主控台。
2. 選擇 VPC Dashboard (VPC 儀表板)，再選擇 Security Groups (安全群組)，然後選擇 Create security group (建立安全群組)。
3. 在 Create security group (建立安全群組) 頁面上，設定下列值：

- 安全群組名稱 — **example-securitygroup**
- 描述 — **Instance security group**
- VPC — 選擇您先前建立的 VPC，例如：vpc-example

4. 將傳入規則新增至安全群組

a. 在 Inbound rules (傳入規則) 區段中，選擇 Add rule (新增規則)。

b. 為您的新傳入規則設定下列值，以允許來自 Amazon EC2 執行個體的連接埠 27017 上的 DocumentDB 流量。執行此操作後，您可以從應用程式連線到叢集。如此一來，您就可以將資料從應用程式存放和擷取到資料庫。

- 類型 — **Custom TCP**
- 來源 — 您先前在本主題中建立之應用程式安全群組的識別符，例如：sg-9edd5cfb。

c. 選擇新增規則。

d. 為您的新傳入規則設定下列值，以允許 HTTP 存取您的應用程式：

- 類型 — **HTTP**

- 來源 — **0.0.0.0/0**

5. 請選擇 Create security group (建立安全群組) 以建立安全群組。

步驟 4：建立子網路群組

子網路群組是您在 VPC 中建立，然後為叢集指定的子網路集合。子網路群組可讓您在建立叢集時指定特定的 VPC。

建立子網路群組

1. 識別 VPC 中資料庫的私有子網路。

   a. 在 https：//https://console.aws.amazon.com/vpc 開啟 Amazon VPC 主控台。

   b. 選擇 VPC Dashboard (VPC 儀表板)，然後選擇 Subnets (子網路)。

   c. 請注意您在名為 的步驟 1 中建立之子網路IDs，例如：example-subnet-private1-us-west-2a 和 example-subnet-private2-us-west-2b。建立子網路群組時，您需要子網路 IDs。

2. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。

   請確定您連線到 Amazon DocumentDB 主控台，而不是 Amazon VPC 主控台。

3. 在導覽窗格中選擇 Subnet groups (子網路群組)。
4. 選擇建立。
5. 在建立子網路群組頁面上，在子網路群組詳細資訊區段中設定這些值：

   - 名稱 — **example-db-subnet-group**
   - 描述 — **Instance security group**

6. 在新增子網路區段中，設定這些值：

   - VPC — 選擇您先前建立的 VPC，例如：vpc-example
   - 可用區域 — 選取在步驟 1 中建立的兩個可用區域。範例：us-west-2a 和 us-west-2b
   - 子網路 — 選擇您在步驟 1 中建立的私有子網路。

7. 選擇建立。

您的新子網路群組會出現在 DocumentDB 主控台的子網路群組清單中。您可以選擇子網路群組，以在詳細資訊窗格中查看詳細資訊。這些詳細資料包括與群組關聯的所有子網路。

> **Note**
>
> 如果您建立此 VPC 以與 DocumentDB 叢集建立關聯，請依照 中的指示建立叢集建立 Amazon DocumentDB 叢集。

刪除 VPC

您可以刪除不再需要的 VPC 和其中使用的其他資源。

> **Note**
>
> 如果您在本主題中建立的 VPC 中新增資源，您可能需要先刪除這些資源，才能刪除 VPC。例如，這些資源可能包含 Amazon EC2 執行個體或 DocumentDB 叢集。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的刪除 VPC。

刪除 VPC 和相關資源

1. 刪除子網路群組：
   a. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
   b. 在導覽窗格中選擇 Subnet groups (子網路群組)。
   c. 選取您要刪除的子網路群組，例如 example-db-subnet-group。
   d. 選擇 Delete (刪除)，然後在確認視窗中選擇 Delete (刪除)。
2. 請記下 VPC ID：
   a. 開啟位於 https：//https://console.aws.amazon.com/vpc 的 Amazon VPC 主控台。
   b. 選擇 VPC 儀表板，然後選擇您的 VPCs。
   c. 在清單中，識別您建立的 VPC，例如 vpc-example。
   d. 請記下您所建立的 VPC 的 VPC ID。在稍後的步驟中，您需要 VPC ID。
3. 刪除安全群組：
   a. 開啟位於 https：//https://console.aws.amazon.com/vpc 的 Amazon VPC 主控台。
   b. 選擇 VPC Dashboard (VPC 儀表板)，然後選擇 Security Groups (安全群組)。
   c. 選取 Amazon DocumentDB 叢集的安全群組，例如 example-securitygroup。

d. 針對動作，選擇刪除安全群組，然後在確認對話方塊中選擇刪除。

e. 返回安全群組頁面，選取 Amazon EC2 執行個體的安全群組，例如 example-securitygroup。

f. 針對動作，選擇刪除安全群組，然後在確認對話方塊中選擇刪除。

4. 刪除 VPC

a. 開啟位於 https：//https://console.aws.amazon.com/vpc 的 Amazon VPC 主控台。

b. 選擇 VPC 儀表板，然後選擇您的 VPCs。

c. 選取您要刪除的 VPC，例如 vpc-example。

d. 針對動作，選擇刪除 VPC。

確認頁面會顯示與 VPC 相關聯的其他資源，這些資源也將遭到刪除，包括與其相關聯的子網路。

e. 在確認對話方塊中，輸入 **delete**，然後選擇刪除。

建立雙堆疊 VPC 以與 DocumentDB 叢集搭配使用

常見案例包括以 Amazon VPC 服務為基礎的虛擬私有雲端 (VPC) 中的叢集。此 VPC 與在相同 VPC 中執行的公有 Amazon EC2 執行個體共用資料。在本主題中，您會為此案例建立 VPC。

主題

- 步驟 1：建立具有私有和公有子網路的 VPC
- 步驟 2：為公有 Amazon EC2 執行個體建立 VPC 安全群組
- 步驟 3：建立私有叢集的 VPC 安全群組
- 步驟 4：建立子網路群組
- 步驟 5：以雙堆疊模式建立 Amazon EC2 執行個體
- 步驟 6：以雙堆疊模式建立叢集
- 步驟 7：連線至您的 Amazon EC2 執行個體和資料庫叢集
- 刪除 VPC

在此程序中，您會為此案例建立 VPC，該 VPC 適用於在雙堆疊模式下執行的資料庫。雙堆疊模式可透過 IPv6 定址通訊協定啟用連線。若需 IP 定址的詳細資訊，請參閱 Amazon DocumentDB IP 定址。

大多數 區域都支援雙堆疊網路叢集。如需更多資訊，請參閱雙堆疊模式區域和版本可用性。若要查看雙堆疊模式的限制，請參閱 雙堆疊網路叢集的限制。

本主題和IPv4-only主題會在相同的 VPC 中建立公有和私有子網路。如需有關在一個 VPC 中建立 Amazon DocumentDB 叢集和在不同 VPC 中建立 Amazon EC2 執行個體的資訊，請參閱 在 VPC 中存取 Amazon DocumentDB 叢集。

您的 DocumentDB 叢集必須僅適用於您的 Amazon EC2 執行個體，而非公有網際網路。因此，您建立同時包含公有和私有子網路的 VPC。EC2 執行個體託管在公有子網路中，因此可以連線到公有網際網路。叢集託管在私有子網路中。EC2 執行個體可以連接到叢集，因為它託管在相同的 VPC 內。不過，叢集不適用於公有網際網路，可提供更高的安全性。

本主題中的程序會在個別可用區域中設定額外的公有和私有子網路。程序不會使用這些子網路。DocumentDB 子網路群組至少需要兩個可用區域中的子網路。額外的子網路可讓您輕鬆設定多個 DocumentDB 執行個體。

若要建立使用雙堆疊模式的叢集，請為網路類型設定指定雙堆疊模式 e。您也可以使用相同的設定來修改叢集。如需建立叢集的詳細資訊，請參閱 建立 Amazon DocumentDB 叢集。如需修改資料庫叢集的詳細資訊，請參閱修改 Amazon DocumentDB 叢集。

本主題說明為 Amazon DocumentDB 叢集設定 VPC。如需 Amazon VPC 的詳細資訊，請參閱《Amazon VPC 使用者指南》。

步驟 1：建立具有私有和公有子網路的 VPC

使用下列程序來建立一個同時含公有和私有子網路的 VPC。

建立 VPC 和子網路

1. 開啟位於 https：//https://console.aws.amazon.com/vpc 的 Amazon VPC 主控台。
2. 在 的右上角 AWS 管理主控台，選擇要在其中建立 VPC 的區域。此範例使用 美國西部 (奧勒岡) 區域。
3. 在左上角，選擇 VPC Dashboard (VPC 儀表板)。若要開始建立 VPC，請選擇 Create VPC (建立 VPC)。
4. 在 VPC Settings (VPC 設定) 的 Resources to create (建立資源) 下，選擇 VPC and more (VPC 和更多)。
5. 對於 VPC settings (VPC 設定)，設定這些值：
   - 名稱標籤自動產生 — **`example-dual-stack`**
   - IPv4 CIDR 區塊 — **`10.0.0.0/16`**
   - IPv6 CIDR 區塊 — Amazon 提供的 IPv6 CIDR 區塊

- 租用 — 預設
- 可用區域 (AZs) 的數量 — 2
- 自訂 AZs — 保留預設值
- 公有子網路數量 — 2
- 私有子網路數量 — 2
- 自訂子網路 CIDR 區塊 — 保留預設值
- NAT 閘道 ($) — 無
- 僅輸出網際網路閘道 — 否
- VPC 端點 — 無
- DNS 選項 — 保留預設值

6. 選擇建立 VPC。

步驟 2：為公有 Amazon EC2 執行個體建立 VPC 安全群組

接著建立公開存取的 VPC 安全群組。若要連接至 VPC 中的公有 EC2 執行個體，請將傳入規則新增至 VPC 安全群組，以允許流量從網際網路連接。

建立 VPC 安全群組

1. 開啟位於 https：//https://console.aws.amazon.com/vpc 的 Amazon VPC 主控台。
2. 選擇 VPC Dashboard (VPC 儀表板)，再選擇 Security Groups (安全群組)，然後選擇 Create security group (建立安全群組)。
3. 在 Create security group (建立安全群組) 頁面上，設定下列值：

   - 安全群組名稱 — **`example-dual-stack-securitygroup`**
   - 描述 — **`Dual-stack security group`**
   - VPC — 選擇您先前建立的 VPC，例如：vpc-example-dual-stack。

4. 將傳入規則新增至安全群組

   a. 決定用於使用安全殼層 (SSH) 連接至 VPC 中 EC2 執行個體的 IP 地址。若要判斷公有 IP 地址，您可以在不同的瀏覽器視窗或索引標籤中使用 https://checkip.amazonaws.com 中的服務。

      網路通訊協定第 4 版 (IPv4) 地址範圍的範例為 `203.0.113.25/32`。網際網路通訊協定第 6 版 (IPv6) 地址範圍的範例為 `2001:db8:1234:1a00::/64`。

在許多情況下，您可能透過網際網路服務供應商 (ISP) 或是從沒有靜態 IP 地址的防火牆進行連線。若是如此，請找出用戶端電腦所使用的 IP 地址範圍。

> **Warning**
>
> 若您將 `0.0.0.0/0` 用於 IPv4 或 `::0` 用於 IPv6，則可讓所有 IP 地址使用 SSH 存取您的公有執行個體。通常在測試環境中短暫使用此方法是沒有問題的，但在生產環境則不安全。在生產環境中，建議您只授權特定 IP 地址或特定範圍的地址存取您的執行個體。

b. 在 Inbound rules (傳入規則) 區段中，選擇 Add rule (新增規則)。

c. 針對您的新傳入規則設定下列值，以允許透過 SSH 存取您的 Amazon EC2 執行個體。執行此操作後，您可以連線至 EC2 執行個體來安裝應用程式或其他公用程式。指定 IP 地址，讓您可以存取 EC2 執行個體：

- 類型 — **SSH**
- 來源 — 您從步驟 a 建立的 IP 地址或範圍。IPv4 地址範圍的範例為 **203.0.113.25/3**2。IPv6 地址範圍的範例為 **2001:DB8::/32**。

d. 選擇新增規則。

5. 請選擇 Create security group (建立安全群組) 以建立安全群組。

   請注意安全群組 ID，因為您稍後在另一個程序中需要它。

步驟 3：建立私有叢集的 VPC 安全群組

若要保持叢集的私密性，請建立第二個安全群組以進行私有存取。若要連線到 VPC 中的私有叢集，請將傳入規則新增至 VPC 安全群組。這些規則僅允許來自 Amazon EC2 執行個體的流量。

建立 VPC 安全群組

1. 開啟位於 https：//https://console.aws.amazon.com/vpc 的 Amazon VPC 主控台。
2. 選擇 VPC Dashboard (VPC 儀表板)，再選擇 Security Groups (安全群組)，然後選擇 Create security group (建立安全群組)。
3. 在 Create security group (建立安全群組) 頁面上，設定下列值：

   - 安全群組名稱 — **example-dual-stack-cluster-securitygroup**
   - 描述 — **Dual-stack cluster security group**

- VPC — 選擇您先前建立的 VPC，例如：vpc-example-dual-stack

4. 將傳入規則新增至安全群組

   a. 在 Inbound rules (傳入規則) 區段中，選擇 Add rule (新增規則)。

   b. 為您的新傳入規則設定下列值，以允許來自 Amazon EC2 執行個體的連接埠 27017 上的 DocumentDB 流量。執行此操作後，您可以從 EC2 執行個體連線到叢集。如此一來，您就可以將資料從 EC2 執行個體傳送至資料庫。

      - 類型 — **`Custom TCP`**
      - 來源 — 您先前在本主題中建立的 EC2 安全群組識別符，例如：sg-9edd5cfb。

5. 請選擇 Create security group (建立安全群組) 以建立安全群組。

步驟 4：建立子網路群組

子網路群組是您在 VPC 中建立，然後為叢集指定的子網路集合。透過使用子網路群組，您可以在建立叢集時指定特定 VPC。若要建立DUAL相容的子網路群組，所有子網路都必須DUAL相容。如要成為 DUAL 相容，子網路必須具有與其關聯的 IPv6 CIDR。

建立子網路群組

1. 識別 VPC 中資料庫的私有子網路。

   a. 開啟位於 https：//https://console.aws.amazon.com/vpc 的 Amazon VPC 主控台。

   b. 選擇 VPC Dashboard (VPC 儀表板)，然後選擇 Subnets (子網路)。

   c. 請注意您在名為 的步驟 1 中建立之子網路IDs，例如：example-dual-stack-subnet-private1-us-west-2a 和 example-dual-stack-subnet-private2-us-west-2b。建立子網路群組時，您需要子網路 IDs。

2. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。

   請確定您連線到 Amazon DocumentDB 主控台，而不是 Amazon VPC 主控台。

3. 在導覽窗格中選擇 Subnet groups (子網路群組)。

4. 選擇建立。

5. 在建立子網路群組頁面上，在子網路群組詳細資訊區段中設定這些值：

   - 名稱 — **`example-dual-stack-cluster-subnet-group`**

- 描述 — **`Dual-stack cluster subnet group`**

6. 在新增子網路區段中，設定這些值：

   - VPC — 選擇您先前建立的 VPC，例如：vpc-example-dual-stack
   - 可用區域 — 選取在步驟 1 中建立的兩個可用區域。範例：us-west-2a 和 us-west-2b
   - 子網路 — 選擇您在步驟 1 中建立的私有子網路。

7. 選擇建立。

您的新子網路群組會出現在 DocumentDB 主控台的子網路群組清單中。您可以選擇子網路群組，以在詳細資訊窗格中查看詳細資訊。這些詳細資料包括與群組關聯的所有子網路。

步驟 5：以雙堆疊模式建立 Amazon EC2 執行個體

若要建立 Amazon EC2 執行個體，請遵循《Amazon Elastic Compute Cloud 使用者指南》中的使用主控台中的啟動執行個體精靈啟動 EC2 執行個體中的指示。

在 Configure Instance Details (設定執行個體詳細資訊) 頁面上設定這些值，而其他值都維持預設值：

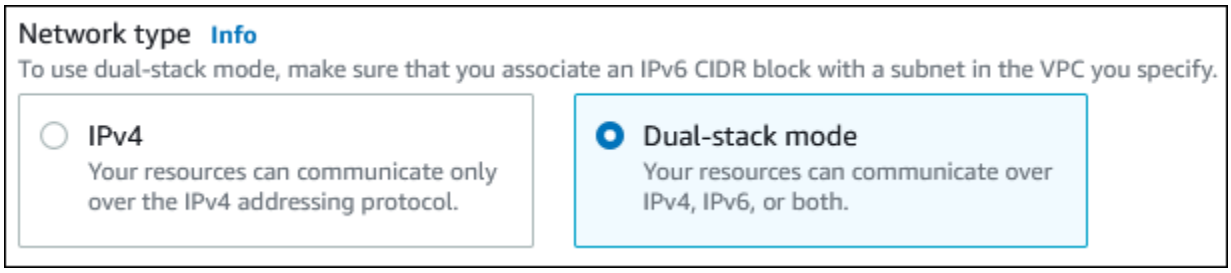
- 網路 — 選擇同時具有公有和私有子網路的現有 VPC，例如在 中建立的 vpc-example-dual-stack-vpc (vpc-identifier)步驟 1：建立具有私有和公有子網路的 VPC。
- 子網路 — 選擇現有的公有子網路，例如在 中建立的子網路識別符 | example-dual-stack-subnet-public1-us-east-2a | us-east-2a步驟 2：為公有 Amazon EC2 執行個體建立 VPC 安全群組。
- 自動指派公有 IP — 選擇啟用。
- 自動指派 IPv6 IP — 選擇啟用。
- 防火牆 （安全群組） 選擇選取現有的安全群組。—
- 常見安全群組 — 選擇現有的安全群組，例如在 中建立的 example-dual-stack-securitygroup步驟 2：為公有 Amazon EC2 執行個體建立 VPC 安全群組。請確定您選擇的安全性群組包含安全殼層 (SSH) 和 HTTP 存取的輸入規則。

步驟 6：以雙堆疊模式建立叢集

在此步驟中，您會建立以雙堆疊模式執行的資料庫叢集。！！！ 注意：在對主控台進行 IPv6 更新後，本節需要編輯 ！！！

在雙堆疊模式中建立叢集

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在主控台的右上角，選擇您要建立 DocumentDB 叢集的 AWS 區域 。此範例使用 美國東部 (俄亥俄) 區域。
3. 在導覽窗格中，選擇叢集。
4. 在叢集清單頁面上，選擇建立。
5. 在建立 Amazon DocumentDB 叢集頁面上，確定已選擇執行個體型叢集選項。
6. 在連線區段的網路類型下，選擇雙堆疊模式。



7. 在頁面底部，開啟顯示進階設定。
8. 在網路設定區段中，設定這些值：

   - 虛擬私有雲端 (VPC) — 選擇同時具有公有和私有子網路的現有 VPC，例如在 中建立的 vpc-example-dual-stack (vpc-identifier)步驟 1：建立具有私有和公有子網路的 VPC。

     VPC 必須具有位於不同可用區域的子網路。
   - 子網路群組 — 選擇 VPC 的子網路群組，例如在 中建立的 example-dual-stack-cluster-subnet-group步驟 4：建立子網路群組。
   - 公開存取 — 選擇否。
   - VPC 安全群組 （ 防火牆 ） — 選取選擇現有。
   - 現有 VPC 安全群組 — 選擇為私有存取設定的現有 VPC 安全群組，例如在 中建立的 example-dual-stack-cluster-securitygroup步驟 3：建立私有叢集的 VPC 安全群組。

     選擇與各項相關聯的 X，以移除其他安全群組，例如預設安全群組。
   - 可用區域 — 選擇您在步驟 1 中建立的可用區域。範例：us-west-2a。

     若要避免跨可用區域流量，請確定叢集和 EC2 執行個體位於相同的可用區域。
9. 對於其餘區段，指定您的叢集設定。如需每項設定的相關資訊，請參閱 建立 Amazon DocumentDB 叢集。

步驟 7：連線至您的 Amazon EC2 執行個體和資料庫叢集

在以雙堆疊模式建立 Amazon EC2 執行個體和 DocumentDB 叢集之後，您可以使用 IPv6 通訊協定連線到每個叢集。若要使用 IPv6 通訊協定連線至 EC2 執行個體，請遵循《Amazon Elastic Compute Cloud 使用者指南》中的連接 EC2 執行個體中的指示。

若要從 EC2 執行個體連線至 DocumentDB 叢集，請遵循 中的說明步驟 5：安裝 MongoDB Shell，在 Connect Amazon EC2 手動主題中 （並在相同的程序中繼續後續步驟 6 和步驟 7)。

刪除 VPC

您可以刪除不再需要的 VPC 和其中使用的其他資源。

> **Note**
>
> 如果您在本主題中建立的 VPC 中新增資源，您可能需要先刪除這些資源，才能刪除 VPC。例如，這些資源可能包含 Amazon EC2 執行個體或 DocumentDB 叢集。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的刪除 VPC。

刪除 VPC 和相關資源

1. 刪除子網路群組：
   a. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
   b. 在導覽窗格中選擇 Subnet groups (子網路群組)。
   c. 選取您要刪除的子網路群組，例如 example-dual-stack-cluster-subnet-group。
   d. 選擇 Delete (刪除)，然後在確認視窗中選擇 Delete (刪除)。
2. 請記下 VPC ID：
   a. 開啟位於 https：//https://console.aws.amazon.com/vpc 的 Amazon VPC 主控台。
   b. 選擇 VPC 儀表板，然後選擇您的 VPCs。
   c. 在清單中，識別您建立的 VPC，例如 vpc-example-dual-stack。
   d. 請記下您所建立的 VPC 的 VPC ID。在稍後的步驟中，您需要 VPC ID。
3. 刪除安全群組：
   a. 開啟位於 https：//https://console.aws.amazon.com/vpc 的 Amazon VPC 主控台。

b. 選擇 VPC Dashboard (VPC 儀表板)，然後選擇 Security Groups (安全群組)。
c. 選取 Amazon DocumentDB 叢集的安全群組，例如 example-dual-stack-securitygroup。
d. 針對動作，選擇刪除安全群組，然後在確認對話方塊中選擇刪除。
e. 返回安全群組頁面，選取 Amazon EC2 執行個體的安全群組，例如 example-securitygroup。
f. 針對動作，選擇刪除安全群組，然後在確認對話方塊中選擇刪除。

4. 刪除 NAT 閘道：

a. 開啟位於 https：//https://console.aws.amazon.com/vpc 的 Amazon VPC 主控台。
b. 選擇 VPC Dashboard (VPC 儀表板)，然後選擇 Security Groups (安全群組)。
c. 選取您建立之 VPC 的 NAT 閘道。使用 VPC ID 識別正確的 NAT 閘道。
d. 若為 Actions (動作)，選擇 Delete NAT gateway (刪除 NAT 閘道)。
e. 在確認對話方塊中，輸入 **`delete`**，然後選擇刪除。

5. 刪除 VPC

a. 開啟位於 https：//https://console.aws.amazon.com/vpc 的 Amazon VPC 主控台。
b. 選擇 VPC 儀表板，然後選擇您的 VPCs。
c. 選取您要刪除的 VPC，例如 vpc-example-dual-stack。
d. 針對動作，選擇刪除 VPC。

   確認頁面會顯示與 VPC 相關聯的其他資源，這些資源也將遭到刪除，包括與其相關聯的子網路。

e. 在確認對話方塊中，輸入 **`delete`**，然後選擇刪除。

6. 釋出彈性 IP 地址：

a. 在 https：//https://console.aws.amazon.com/ec2 開啟 EC2 主控台。
b. 選擇 EC2 Dashboard (EC2 儀表板)，然後選擇 Elastic IPs (彈性 IP)。
c. 選取您要釋放的彈性 IP 地址。
d. 若為 Actions (動作)，選擇 Release Elastic IP addresses (釋放彈性 IP 地址)。
e. 在確認對話方塊中，選擇釋出。

在 Amazon DocumentDB 中備份和還原

Amazon DocumentDB （與 MongoDB 相容） 會持續將您的資料備份至 Amazon Simple Storage Service (Amazon S3) 1-35 天，讓您可以快速還原至備份保留期間內的任何時間點。Amazon DocumentDB 也會自動擷取資料的快照，做為此持續備份程序的一部分。

> **Note**
>
> 這些是服務管理的 Amazon S3 儲存貯體，您將無法存取備份檔案。若要控制您自己的備份，請參閱 傾印、還原、匯入和匯出資料。

如果要將備份資料保留超過備份保留期，您也可以建立叢集資料的手動快照。備份過程不會影響您的叢集效能。

本節討論 Amazon DocumentDB 中備份功能的使用案例，並說明如何管理 Amazon DocumentDB 叢集的備份。

主題

- 備份和還原：概念
- 了解 備份儲存體用量
- 傾印、還原、匯入和匯出資料
- 叢集快照考量事項
- 比較自動和手動快照
- 建立手動叢集快照
- 複製 Amazon DocumentDB 叢集快照
- 共用 Amazon DocumentDB 叢集快照
- 從叢集快照還原
- 還原至某個時間點
- 刪除叢集快照

備份和還原：概念

名詞	說明	API (動詞)

Backup retention period (備份保留期間)	在 1 到 35 天的期限內，您可以執行時間點還原。	`create-db-cluster`
		`modify-db-cluster`
		`restore-db-cluster-to-point-in-time`

Amazon DocumentDB 儲存磁碟區	高可用性和高耐用性的儲存磁碟區，可透過三個可用區域以六種方式複寫資料。無論叢集中的執行個體數量為何，Amazon DocumentDB 叢集都非常耐用。	`create-db-cluster` `delete-db-cluster`

備份時段	1 天內建立自動快照的期間。	`create-db-cluster`
		`describe-db-cluster`
		`modify-db-cluster`

自動快照	屬於叢集完整備份的每日快照，並由 Amazon DocumentDB 中的連續	`restore-db-cluster-from-snapshot`
		`describe-db-cluster-snapshot-attributes`
		`describe-db-cluster-snapshots`

名詞	說明	API (動詞)
	備份程序自動建立。	

手動快照	您手動建立的快照，可在超過備份保留期之後用來保留叢集的完整備份。	`create-db-cluster-snapshot`
		`copy-db-cluster-snapshot`
		`delete-db-cluster-snapshot`
		`describe-db-cluster-snapshot-attributes`
		`describe-db-cluster-snapshots`
		`modify-db-cluster-snapshot-attribute`

了解 備份儲存體用量

Amazon DocumentDB 備份儲存包含備份保留期間內的連續備份，以及保留期間內以外的手動快照。若要控制您的備份儲存用量，您可以降低備份保留間隔、移除不再需要的舊手動快照，或同時進行兩者。如需 Amazon DocumentDB 備份的一般資訊，請參閱 在 Amazon DocumentDB 中備份和還原。如需 Amazon DocumentDB 備份儲存體的定價資訊，請參閱 Amazon DocumentDB 定價。

若要控制您的成本，您可以監控連續備份和超出保留期的手動快照所消耗的儲存量。然後，您可以降低備份保留間隔，並移除不再需要的手動快照。

您可以使用 Amazon CloudWatch 指標 `TotalBackupStorageBilled`、`SnapshotStorageUsed`和 `BackupRetentionPeriodStorageUsed`來檢閱和監控 Amazon DocumentDB 備份所使用的儲存量，如下所示：

- `BackupRetentionPeriodStorageUsed` 表示目前用於儲存連續備份的備份儲存量。此指標值取決於叢集磁碟區的大小，以及您在保留期中所做的變更數。不過，為了方便計費，指標不會超過保留期內累積的叢集磁碟區大小。例如，如果您的叢集大小為 100 GiB，而保留期為兩天，則 `BackRetentionPeriodStorageUsed` 的最大值為 200 GiB (100 GiB+100 GiB)。

- SnapshotStorageUsed 表示用於儲存在備份保留期外之手動快照的備份儲存量。在保留期內拍攝的手動快照不計入您的備份儲存。同樣地，自動快照不會計入您的備份儲存。每個快照的大小是您取得快照時的叢集磁碟區大小。SnapshotStorageUsed 值取決於您保留的快照數量和每個快照的大小。例如，假設您在保留期之外有一個快照，且叢集磁碟區在拍攝快照時的大小為 100 GiB。則 SnapshotStorageUsed 的數量為 100 GiB。

- TotalBackupStorageBilled 表示 BackupRetentionPeriodStorageUsed 和 SnapshotStorageUsed 的總和，減去相當於叢集磁碟區一天大小的免費備份儲存量。例如，如果您的叢集大小為 100 GiB，您有一天的保留日，而且您在保留期間之外有一個快照，則 TotalBackupStorageBilled為 100 GiB (100 GiB + 100 GiB - 100 GiB)。

- 這些指標會針對每個 Amazon DocumentDB 叢集獨立計算。

您可以透過 CloudWatch 主控台監控 Amazon DocumentDB 叢集，並使用 CloudWatch 指標建置報告。 CloudWatch 如需如何使用 CloudWatch 指標的詳細資訊，請參閱在 Amazon DocumentDB 中監控和記錄。

傾印、還原、匯入和匯出資料

您可以使用 mongodump、mongoexport、 mongorestore和 mongoimport公用程式，將資料移入和移出 Amazon DocumentDB 叢集。本節探討這些工具和組態的目的，以協助您達到更佳效能。

> **MongoDB 資料庫工具版本需求**
>
> 針對 Amazon DocumentDB 使用 MongoDB 資料庫工具，包括 100.11.0 版。 Amazon DocumentDB 若要下載，請參閱 MongoDB 網站上的 MongoDB 資料庫工具版本。 MongoDB

主題

- mongodump
- mongorestore
- mongoexport
- mongoimport

- 教學課程

mongodump

該 mongodump 公用程式建立 MongoDB 資料庫的二進位 (BSON) 備份。由於以二進位格式存放資料所達到的大小效率，因此mongodump工具是希望將其還原至 Amazon DocumentDB 叢集時，從來源 MongoDB 部署傾印資料的偏好方法。

根據您用來執行 命令的執行個體或機器上可用的資源，您可以使用 --numParallelCollections選項增加從預設 1 傾印的平行集合數目，mongodump以加速您的 。一個良好的經驗法則是從 Amazon DocumentDB 叢集主要執行個體上每個 vCPU 一個工作者開始。

範例使用方式

以下是 Amazon DocumentDB 叢集 中mongodump公用程式的範例用量sample-cluster。

```
mongodump --ssl \
    --host="sample-cluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --collection=sample-collection \
    --db=sample-database \
    --out=sample-output-file \
    --numParallelCollections 4  \
    --username=sample-user \
    --password=abc0123 \
    --sslCAFile global-bundle.pem
```

mongorestore

mongorestore 公用程式可讓您還原與 mongodump 公用程式一起建立的資料庫的二進位 (BSON) 備份。您可以在使用 --numInsertionWorkersPerCollection 選項還原期間，增加每個集合的工作者數目，提升還原效能 (預設為 1)。一個良好的經驗法則是從 Amazon DocumentDB 叢集主要執行個體上每個 vCPU 一個工作者開始。

範例使用方式

以下是 Amazon DocumentDB 叢集中 mongorestore 公用程式的範例用量sample-cluster。

```
mongorestore --ssl \
    --host="sample-cluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --username=sample-user \
```

```
    --password=abc0123 \
    --sslCAFile global-bundle.pem <fileToBeRestored>
```

mongoexport

此mongoexport工具會將 Amazon DocumentDB 中的資料匯出為 JSON、CSV 或 TSV 檔案格式。mongoexport 工具是匯出需可人工或機器讀取之資料的偏好方法。

> **Note**
>
> mongoexport 不直接支援平行匯出。不過，您可以藉由同時針對不同的集合執行多個 mongoexport 任務來提高效能。

範例使用方式

以下是 Amazon DocumentDB 叢集 中mongoexport工具的範例用量sample-cluster。

```
mongoexport --ssl \
    --host="sample-cluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --collection=sample-collection \
    --db=sample-database \
    --out=sample-output-file \
    --username=sample-user \
    --password=abc0123 \
    --sslCAFile global-bundle.pem
```

mongoimport

此mongoimport工具會將 JSON、CSV 或 TSV 檔案的內容匯入 Amazon DocumentDB 叢集。您可以使用 --numInsertionWorkers 參數來平行化和加速匯入 (預設值為 1)。

範例使用方式

以下是 Amazon DocumentDB 叢集 中mongoimport工具的範例用量sample-cluster。

```
mongoimport --ssl \
    --host="sample-cluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --collection=sample-collection \
    --db=sample-database \
    --file=<yourFile> \
```

```
    --numInsertionWorkers 4 \
    --username=sample-user \
    --password=abc0123 \
    --sslCAFile global-bundle.pem
```

教學課程

下列教學課程說明如何使用 mongodump、mongoexport、 mongorestore和 mongoimport公用程式將資料移入和移出 Amazon DocumentDB 叢集。

1. 先決條件 — 開始之前，請確定已佈建 Amazon DocumentDB 叢集，而且您可以存取與叢集相同 VPC 中的 Amazon EC2 執行個體。如需詳細資訊，請參閱使用 Amazon EC2 連線。

   為了能夠使用 mongo 公用程式工具，您必須在 EC2 執行個體中安裝 mongodb-org-tools 套件，如下所示。

   ```
   sudo yum install mongodb-org-tools-4.0.18
   ```

   由於 Amazon DocumentDB 預設使用 Transport Layer Security (TLS) 加密，因此您還必須下載 Amazon RDS 憑證授權機構 (CA) 檔案，以使用 mongo shell 進行連線，如下所示。

   ```
   wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
   ```

2. 下載範例資料：在本教學課程中，您將下載一些包含餐廳相關資訊的範例資料。

   ```
   wget https://raw.githubusercontent.com/ozlerhakan/mongodb-json-files/master/
   datasets/restaurant.json
   ```

3. 將範例資料匯入 Amazon DocumentDB — 由於資料採用邏輯 JSON 格式，您將使用 mongoimport公用程式將資料匯入 Amazon DocumentDB 叢集。

   ```
   mongoimport --ssl \
       --host="tutorialCluster.amazonaws.com:27017" \
       --collection=restaurants \
       --db=business \
       --file=restaurant.json \
       --numInsertionWorkers 4 \
       --username=<yourUsername> \
       --password=<yourPassword> \
       --sslCAFile global-bundle.pem
   ```

4. 使用 傾印資料 **mongodump** — 現在您已在 Amazon DocumentDB 叢集中擁有資料，您可以使用mongodump公用程式取得該資料的二進位傾印。

```
mongodump --ssl \
    --host="tutorialCluster.us-east-1.docdb.amazonaws.com:27017"\
    --collection=restaurants \
    --db=business \
    --out=restaurantDump.bson \
    --numParallelCollections 4 \
    --username=<yourUsername> \
    --password=<yourPassword> \
    --sslCAFile global-bundle.pem
```

5. 捨棄**restaurants**集合 - 在還原business資料庫中的restaurants集合之前，您必須先捨棄該資料庫中已存在的集合，如下所示。

```
use business
```

```
db.restaurants.drop()
```

6. 使用 還原資料 **mongorestore** — 使用步驟 3 資料的二進位傾印，您現在可以使用mongorestore公用程式將資料還原至 Amazon DocumentDB 叢集。

```
mongorestore --ssl \
    --host="tutorialCluster.us-east-1.docdb.amazonaws.com:27017" \
    --numParallelCollections 4 \
    --username=<yourUsername> \
    --password=<yourPassword> \
    --sslCAFile global-bundle.pem restaurantDump.bson
```

7. 使用 匯出資料 **mongoexport** — 若要完成教學課程，請以 JSON 檔案的格式從叢集匯出資料，與您在步驟 1 中匯入的檔案不同。

```
mongoexport --ssl \
    --host="tutorialCluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --collection=restaurants \
    --db=business \
    --out=restaurant2.json \
    --username=<yourUsername> \
    --password=<yourPassword> \
    --sslCAFile global-bundle.pem
```

8. 驗證 — 您可以使用下列命令，驗證步驟 5 的輸出是否產生與步驟 1 相同的結果。

```
wc -l restaurant.json
```

從這個命令的輸出：

```
2548 restaurant.json
```

```
wc -l restaurant2.json
```

從這個命令的輸出：

```
2548 restaurant2.json
```

叢集快照考量事項

Amazon DocumentDB 會在叢集的備份時段期間建立叢集的每日自動快照。Amazon DocumentDB 會根據您指定的備份保留期儲存叢集的自動快照。如有需要，您可將叢集復原至備份保留期間的任何時間點。在相同叢集的相同區域執行複製操作時，自動快照不會發生。

主題

- 備份儲存體
- 備份時段
- Backup retention period (備份保留期間)
- 複製叢集快照加密

除了自動叢集快照，您也可以手動建立叢集快照。您可以複製自動與手動快照。如需詳細資訊，請參閱建立手動叢集快照及複製 Amazon DocumentDB 叢集快照。

> **Note**
>
> 您的叢集必須為可用狀態，才能建立自動快照。
> 您無法共用 Amazon DocumentDB 自動化叢集快照。因應措施是，您可以藉由複製自動快照來建立手動快照，然後共用該複本。如需複製快照的詳細資訊，請參閱 複製 Amazon DocumentDB 叢集快照。如需從快照還原叢集的詳細資訊，請參閱 從叢集快照還原。

備份儲存體

每個 的 Amazon DocumentDB 備份儲存 AWS 區域 體是由備份保留期間所需的備份儲存體組成，其中包括該區域中的自動和手動叢集快照。預設備份保留期間為 1 天。如需備份儲存定價的詳細資訊，請參閱 Amazon DocumentDB 定價。

刪除叢集時，所有自動快照將一併刪除，且無法復原。不過，刪除叢集時不會刪除手動快照。如果您選擇讓 Amazon DocumentDB 在刪除叢集之前建立最終快照（手動快照），您可以使用最終快照來復原叢集。

如需快照與儲存的詳細資訊，請參閱 了解 備份儲存體用量。

備份時段

自動快照會每天在您偏好的備份時段內執行。如果快照需要的時間超過所分配的備份時段，備份會在時段結束後繼續執行直到完成。備份時段不可與每週叢集維護時段重疊。

如果您在建立叢集時未指定偏好的備份時段，Amazon DocumentDB 會指派預設的 30 分鐘備份時段。這個時段是根據與叢集區域相關聯的 8 小時時段內隨機選擇。您可以修改叢集來變更偏好的備份時段。如需詳細資訊，請參閱修改 Amazon DocumentDB 叢集。

區域名稱	區域	UTC 時間區塊
美國東部 (俄亥俄)	us-east-2	03:00-11:00
美國東部 (維吉尼亞北部)	us-east-1	03:00-11:00
美國西部 (奧勒岡)	us-west-2	06:00-14:00
Africa (Cape Town)	af-south-1	03：00–11：00
亞太地區 (香港)	ap-east-1	06:00-14:00
亞太地區 (海德拉巴)	ap-south-2	06：30–14：30
亞太地區 (馬來西亞)	ap-southeast-5	13:00-21:00
亞太區域 (孟買)	ap-south-1	06:00-14:00
亞太地區 (大阪)	ap-northeast-3	12：00-20：00

區域名稱	區域	UTC 時間區塊
亞太區域 (首爾)	ap-northeast-2	13:00-21:00
亞太區域 (新加坡)	ap-southeast-1	14：00-22：00
亞太地區 (悉尼)	ap-southeast-2	12：00-20：00
亞太地區 (雅加達)	ap-southeast-3	08：00-16：00
亞太地區 (墨爾本)	ap-southeast-4	11：00-19：00
亞太區域 (泰國)	ap-southeast-7	15：00-23：00
亞太區域 (東京)	ap-northeast-1	13:00-21:00
加拿大 (中部)	ca-central-1	03:00-11:00
加拿大西部 (卡加利)	ca-west-1	18：00-02：00
中國 (北京)	cn-north-1	06:00-14:00
中國 (寧夏)	cn-northwest-1	06:00-14:00
歐洲 (法蘭克福)	eu-central-1	21：00-05：00
歐洲 (蘇黎世)	eu-central-2	02：00-10：00
歐洲 (愛爾蘭)	eu-west-1	22:00-06:00
歐洲 (倫敦)	eu-west-2	22:00-06:00
歐洲 (米蘭)	eu-south-1	02：00-10：00
Europe (Paris)	eu-west-3	23：59-07：29
歐洲 (西班牙)	eu-south-2	02：00–10：00
Europe (Stockholm)	eu-north-1	04：00–12：00
墨西哥 (中部)	mx-central-1	03:00-11:00

區域名稱	區域	UTC 時間區塊
中東 (阿拉伯聯合大公國)	me-central-1	05：00–13：00
南美洲 (聖保羅)	sa-east-1	00：00-08：00
以色列 (特拉維夫)	il-central-1	04：00-12：00
AWS GovCloud （美國東部）	us-gov-east-1	17：00-01：00
AWS GovCloud （美國西部）	us-gov-west-1	06:00-14:00

Backup retention period (備份保留期間)

備份保留期間是在自動刪除之前保留自動備份的天數。Amazon DocumentDB 支援 1–35 天的備份保留期。

您可以在建立叢集時設定備份保留期。如果您沒有明確設定備份保留期，則會為叢集指派預設的備份保留期 1 天。建立叢集之後，您可以使用 AWS 管理主控台 或 修改叢集，以修改備份保留期間 AWS CLI。如需詳細資訊，請參閱修改 Amazon DocumentDB 叢集。

複製叢集快照加密

叢集和快照加密是以 KMS 加密金鑰為基礎。KMS 金鑰 ID 是 KMS 加密金鑰的 Amazon Resource Name (ARN)、KMS 金鑰識別符或 KMS 金鑰別名。

適用下列準則和限制：

- 建立快照時，會從叢集推斷加密。如果叢集已加密，則該叢集的快照會使用相同的 KMS 金鑰加密。如果叢集未加密，則快照不會加密。
- 如果您從 Amazon Web Services 帳戶複製加密的叢集快照，您可以指定 的值`KmsKeyId`，以使用新的 KMS 加密金鑰來加密複本。如果您未指定 `KmsKeyId` 的值，則會使用與來源叢集快照相同的 KMS 金鑰來加密叢集快照的複本。
- 如果您複製從另一個 Amazon Web Services 帳戶共用的加密叢集快照，則必須指定 的值`KmsKeyId`。
- 若要將加密的叢集快照複製到另一個 Amazon Web Services 區域，請將 `KmsKeyId` 設定為您要用來加密目的地區域中叢集快照複本的 KMS 金鑰 ID。KMS 加密金鑰專屬於在其中建立的 Amazon

Web Services 區域，您無法在另一個 Amazon Web Services 區域中使用來自某個 Amazon Web Services 區域的加密金鑰。

- 如果您複製未加密的叢集快照並指定 KmsKeyId 參數的值，則會傳回錯誤。

比較自動和手動快照

以下是 Amazon DocumentDB （與 MongoDB 相容） 自動和手動快照的主要功能。

Amazon DocumentDB 自動快照具有下列主要功能：

- 自動快照命名 — 自動快照名稱遵循模式 `rds:<cluster-name>-yyyy-mm-dd-hh-mm`，yyyy-mm-dd-hh-mm代表快照建立的日期和時間。
- 排程自動建立 — 當您建立或修改叢集時，您可以將備份保留期設定為 1 到 35 天的整數值。根據預設，新叢集的備份保留期為 1 天。備份保留期定義自動刪除自動快照之前的保留天數。您無法在 Amazon DocumentDB 叢集上停用自動備份。

  除了設定備份保留期之外，您也可以設定備份時段，也就是一天中建立自動快照的期間。
- 刪除自動快照 — 當您刪除自動快照的叢集時，會自動刪除快照。您無法手動刪除自動快照。
- 增量 — 在備份保留期間，會記錄資料庫更新，以便有變更的增量記錄。
- 從自動快照還原 — 您可以使用 AWS 管理主控台 或 從自動快照還原 AWS CLI。當您使用 從快照還原時 AWS CLI，您必須在叢集可用後分別新增執行個體。
- 共用 — 您無法共用 Amazon DocumentDB 自動化叢集快照。因應措施是，您可以藉由複製自動快照來建立手動快照，然後共用該複本。如需複製快照的詳細資訊，請參閱 複製 Amazon DocumentDB 叢集快照。如需從快照還原叢集的詳細資訊，請參閱 從叢集快照還原。
- 您可以從備份保留期間內的任何時間點還原 - 由於資料庫更新會遞增記錄，因此您可以將叢集還原至備份保留期間內的任何時間點。

  當您使用 從自動快照或從point-in-time還原還原時 AWS CLI，您必須在叢集可用後分別新增執行個體。

Amazon DocumentDB 手動快照具有下列主要功能：

- 隨需建立 — Amazon DocumentDB 手動快照是使用 Amazon DocumentDB 管理主控台或 隨需建立 AWS CLI。
- 刪除手動快照 — 只有當您使用 Amazon DocumentDB 主控台或 明確刪除手動快照時，才會刪除手動快照 AWS CLI。當您刪除手動快照的叢集時，並不會刪除手動快照。

- 完整備份 — 手動快照建立並儲存叢集資料的完整備份時。
- 手動快照命名 — 您可以指定手動快照名稱。Amazon DocumentDB `datetime` 不會將戳記新增至名稱，因此如果您想要將該資訊包含在名稱中，則必須新增該資訊。
- 從手動快照還原 — 您可以使用主控台或 從手動快照還原 AWS CLI。當您使用 從快照還原時 AWS CLI，您必須在叢集可用後分別新增執行個體。
- Service Quotas — 每個 最多只能有 100 個手動快照 AWS 區域。
- 共用 — 您可以共用手動叢集快照，這些快照可由授權複製 AWS 帳戶。您可以共用加密或未加密的手動快照。如需複製快照的詳細資訊，請參閱 複製 Amazon DocumentDB 叢集快照。
- 手動快照建立時，您會還原至 ：從手動快照還原時，您會在手動快照建立時還原至 。

  當您使用 從快照還原時 AWS CLI，您必須在叢集可用後分別新增執行個體。

建立手動叢集快照

您可以使用 AWS 管理主控台 或 建立手動快照 AWS CLI。建立快照所需的時間量因資料庫的大小而異。建立快照時，您必須執行下列作業：

1. 找出要備份的叢集。
2. 為您的快照命名。您稍後可用它來進行還原。

Using the AWS 管理主控台

若要使用 建立手動快照 AWS 管理主控台，您可以遵循下列任一方法。

1. 方法 1：

   1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
   2. 在導覽窗格中，選擇 Snapshots (快照)。

      > **Tip**
      > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

   3. 在 Snapshots (快照) 頁面選擇 Create (建立)。

4. 在 Create cluster snapshot (建立叢集快照) 頁面：

   a. 叢集識別符 — 從叢集的下拉式清單中，選擇要建立快照的叢集。

   b. 快照識別符 — 輸入快照的名稱。

      快照命名限制條件：

      - 長度為 【1–255】 個字母、數字或連字號。
      - 第一個字元必須是字母。
      - 不能以連字號結尾，或包含兩個連續連字號。
      - 對於每個 AWS 區域每個帳戶的所有叢集 （跨 Amazon RDS、Amazon Neptune 和 Amazon DocumentDB)，必須是唯一的。

   c. 選擇建立。

2. 方法 2：

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在 Clusters (叢集) 頁面，找到您要建立快照的叢集，並選擇叢集左側的按鈕。
4. 在 Actions (動作) 功能表上，選擇 Take snapshot (建立快照)。
5. 在 Create cluster snapshot (建立叢集快照) 頁面：

   a. 快照識別符 — 輸入快照的名稱。

      快照命名限制條件：

      - 長度為 【1–63】 個字母、數字或連字號。
      - 第一個字元必須是字母。
      - 不能以連字號結尾，或包含兩個連續連字號。
      - 對於每個 AWS 區域每個帳戶的所有叢集 （跨 Amazon RDS、Amazon Neptune 和 Amazon DocumentDB)，必須是唯一的。

   b. 選擇建立。

Using the AWS CLI

若要使用 建立叢集快照 AWS CLI，請使用 `create-db-cluster-snapshot`操作搭配下列參數。

Parameters

- **`--db-cluster-identifier`** – 必要。您正在建立快照的叢集名稱。此叢集必須存在並且為可用。
- **`--db-cluster-snapshot-identifier`** – 必要。您正在建立的手動快照名稱。

下列範例會為名為 `sample-cluster` 的叢集建立一個名為 `sample-cluster-snapshot` 的快照。

針對 Linux、macOS 或 Unix：

```
aws docdb create-db-cluster-snapshot \
    --db-cluster-identifier sample-cluster \
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

針對 Windows：

```
aws docdb create-db-cluster-snapshot ^
    --db-cluster-identifier sample-cluster ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

此操作的輸出將會如下所示。

```
{
    "DBClusterSnapshot": {
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1b",
            "us-east-1c"
        ],
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot",
        "DBClusterIdentifier": "sample-cluster",
        "SnapshotCreateTime": "2020-04-24T04:59:08.475Z",
        "Engine": "docdb",
        "Status": "creating",
```

```
        "Port": 0,
        "VpcId": "vpc-abc0123",
        "ClusterCreateTime": "2020-01-10T22:13:38.261Z",
        "MasterUsername": "master-user",
        "EngineVersion": "4.0.0",
        "SnapshotType": "manual",
        "PercentProgress": 0,
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DBClusterSnapshotArn": "arn:aws:rds:us-east-1:<accountID>:cluster-
snapshot:sample-cluster-snapshot"
    }
}
```

複製 Amazon DocumentDB 叢集快照

在 Amazon DocumentDB 中，您可以在相同 AWS 區域 或不同的 中複製快照 AWS 區域。您也可以將共用快照複製到相同 AWS 區域 或不同的 帳戶中 AWS 區域。如需有關共用快照的詳細資訊，請參閱 共用 Amazon DocumentDB 叢集快照。

> **Note**
>
> Amazon DocumentDB 會根據您保留的備份和快照資料量，以及您保留資料的期間向您收費。如需與 Amazon DocumentDB 備份和快照相關聯之儲存體的詳細資訊，請參閱 了解 備份儲存體用量。如需 Amazon DocumentDB 儲存體的定價資訊，請參閱 Amazon DocumentDB 定價。

主題

- 複製共用快照
- 跨 複製快照 AWS 區域
- 限制
- 處理加密
- 參數群組考量
- 複製叢集快照

複製共用快照

您可以複製其他 AWS 帳戶與您共用的快照。如果您要複製從另一個 AWS 帳戶共用的加密快照，您必須能夠存取用來加密快照的 AWS KMS 加密金鑰。如需詳細資訊，請參閱處理加密。

跨 複製快照 AWS 區域

當您將快照複製到 AWS 區域 與來源快照不同的 時 AWS 區域，每個複本都是完整快照。完整快照副本包含還原 Amazon DocumentDB 叢集所需的所有資料和中繼資料。

限制

以下是複製快照時的一些限制：

- 如果您在目標快照變成可用之前刪除來源快照，則快照複製可能會失敗。刪除來源快照之前，請確認目標快照的狀態為 AVAILABLE。
- 對於單一目的地區域，每一帳戶最多可有 5 個快照複製請求在進行中。
- 根據 AWS 區域 涉及的 和要複製的資料量，跨區域快照複本可能需要數小時才能完成。有時某個來源 AWS 區域會出現大量跨區域快照複製請求。在這些情況下，Amazon DocumentDB 可能會將來自該來源的新跨區域複製請求 AWS 區域 放入佇列，直到某些進行中的副本完成為止。位於佇列中的複製請求不會顯示進度資訊。複製開始時才會顯示進度資訊。

處理加密

您可以複製以 AWS KMS 加密金鑰所加密的快照。如果您複製加密快照，則快照的副本也必須加密。如果您在相同的 中複製加密快照 AWS 區域，您可以使用與原始快照相同的 AWS KMS 加密金鑰來加密複本，也可以指定不同的 AWS KMS 加密金鑰。如果您跨區域複製加密快照，則無法使用與來源快照相同的 AWS KMS 加密金鑰，因為 AWS KMS 金鑰是區域特定的。反之，您必須在目的地 n AWS 區域中指定有效的 AWS KMS 金鑰。

在整個複製過程中來源快照仍會保持加密狀態。如需詳細資訊，請參閱Amazon DocumentDB 中的資料保護。

> **Note**
>
> 對於 Amazon DocumentDB 叢集快照，您無法在複製快照時加密未加密的叢集快照。

參數群組考量

當您跨區域複製快照時，複本不包含原始 Amazon DocumentDB 叢集所使用的參數群組。當您還原快照以建立新的叢集時，該叢集會取得 AWS 區域 其建立所在 的預設參數群組。若要為新叢集提供與原始叢集相同的參數，您必須執行下列動作：

1. 在目的地中 AWS 區域，使用與原始叢集相同的設定建立 Amazon DocumentDB 叢集參數群組。如果新的 中已存在， AWS 區域您可以使用該項目。
2. 在目的地還原快照後 AWS 區域，請修改新的 Amazon DocumentDB 叢集，並從上一個步驟新增新的或現有的參數群組。如需詳細資訊，請參閱修改 Amazon DocumentDB 叢集。

複製叢集快照

您可以使用 AWS 管理主控台 或 複製 Amazon DocumentDB 叢集 AWS CLI，如下所示。

Using the AWS 管理主控台

若要使用 複製叢集快照 AWS 管理主控台，請完成下列步驟。此程序適用於在相同區域或 AWS 區域 跨區域中複製加密或未加密的叢集快照。

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇快照，然後選擇您要複製之快照左側的按鈕。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 從 Actions (動作) 功能表中，選擇 Copy (複製)。
4. 在產生的建立叢集快照複本頁面中，完成設定區段。

   a. 目的地區域 — 選用。若要將叢集快照複製到不同的 AWS 區域，請 AWS 區域 針對目的地區域選擇該快照。

   b. 新快照識別符 — 輸入新快照的名稱。

      目標快照命名限制條件：

      - 不能是現有快照的名稱。

- 長度為 【1-63】 個字母、數字或連字號。
- 第一個字元必須是字母。
- 不能以連字號結尾，或包含兩個連續連字號。
- 對於每個區域每個 Amazon RDS、Neptune 和 Amazon DocumentDB 的所有叢集 AWS 帳戶，必須是唯一的。

c. 複製標籤 — 若要將來源快照上擁有的任何標籤複製到快照複本，請選擇複製標籤。

5. 填妥 Encryption-at-rest (靜態加密) 部分。

a. 靜態加密 — 如果您的快照未加密，您將無法使用這些選項，因為您無法從未加密的快照建立加密的副本。如果您的快照已加密，您可以變更靜態加密期間 AWS KMS key 使用的 。

如需加密快照複本的詳細資訊，請參閱 複製叢集快照加密。

如需靜態加密的詳細資訊，請參閱 加密靜態 Amazon DocumentDB 資料。

b. AWS KMS 金鑰 — 從下拉式清單中，選擇下列其中一項：

- （預設） aws/rds — 帳戶號碼和 AWS KMS 金鑰 ID 會列在此選項後面。
- <some-key-name> — 如果您建立金鑰，則會列出它並供您選擇。
- 輸入金鑰 ARN — 在 ARN 方塊中，輸入 AWS KMS 金鑰的 Amazon Resource Name (ARN)。ARN 的格式是 `arn:aws:kms:<region>:<accountID>:key/<key-id>` 。

6. 若要製作所選快照的複本，請選擇 Copy snapshot (複製快照)。或者，您可以選擇取消，不複製快照。

Using the AWS CLI

若要使用 複製未加密的叢集快照 AWS CLI，請使用 `copy-db-cluster-snapshot`操作搭配下列參數。如果您要將快照複製到另一個快照 AWS 區域，請在要複製快照 AWS 區域 的 中執行 命令。

- **`--source-db-cluster-snapshot-identifier`** – 必要。要製作複本之叢集快照的識別碼。叢集快照必須存在且位於可用狀態。如果您要將快照複製到另一個快照， AWS 區域 或要複製共用叢集快照，則此識別符必須是來源叢集快照的 ARN 格式。此參數不區分大小寫。
- **`--target-db-cluster-snapshot-identifier`** – 必要。要從來源叢集快照建立之新叢集快照的識別碼。此參數不區分大小寫。

目標快照命名限制條件：

- 不能是現有快照的名稱。
- 長度為 【1-63】 個字母、數字或連字號。
- 第一個字元必須是字母。
- 不能以連字號結尾，或包含兩個連續連字號。
- 對於每個區域每個 Amazon RDS、Neptune 和 Amazon DocumentDB 的所有叢集 AWS 帳戶，必須是唯一的。

- **`--source-region`** — 如果您要將快照複製到另一個快照 AWS 區域，請指定要從中複製加密叢集快照 AWS 區域 的 。

  如果您要將快照複製到另一個 AWS 區域 快照，但未指定 `--source-region`，則必須改為指定 `pre-signed-url`選項。此`pre-signed-url`值必須是 URL，其中包含簽章第 4 版簽署的請求，以便在複製叢集快照 AWS 區域 的來源中呼叫`CopyDBClusterSnapshot`動作。若要進一步了解 `pre-signed-url`，請參閱 CopyDBClusterSnapshot。

- **`--kms-key-id`** — 金鑰的 KMS 金鑰識別符，用於加密叢集快照的副本。

  如果您要將加密的叢集快照複製到另一個快照 AWS 區域，則需要此參數。您必須指定目的地的 KMS 金鑰 AWS 區域。

  如果您要在相同 中複製加密叢集快照 AWS 區域，則 AWS KMS 金鑰參數為選用。叢集快照的副本會使用與來源叢集快照相同的 AWS KMS 金鑰加密。如果您想要指定用來加密複本的新 AWS KMS 加密金鑰，您可以使用此參數執行此操作。

- **`--copy-tags`** — 選用。要複製的標籤和值。

若要在複製操作進行時取消複製操作，您可以在叢集快照處於複製狀態`TargetDBClusterSnapshotIdentifier`時刪除由 `--target-db-cluster-snapshot-identifier`或 識別的目標叢集快照。

Example

範例 1：將未加密快照複製到相同的區域

下列 AWS CLI 範例會在 AWS 區域 與來源快照相同的 `sample-cluster-snapshot-copy`中建立`sample-cluster-snapshot`名為 的複本。建立副本時，原始快照的所有標籤會複製到快照副本。

針對 Linux、macOS 或 Unix：

```
aws docdb copy-db-cluster-snapshot \
    --source-db-cluster-snapshot-identifier sample-cluster-snapshot \
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy \
    --copy-tags
```

針對 Windows：

```
aws docdb copy-db-cluster-snapshot ^
    --source-db-cluster-snapshot-identifier sample-cluster-snapshot ^
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy ^
    --copy-tags
```

此操作的輸出將會如下所示。

```
{
    "DBClusterSnapshot": {
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1b",
            "us-east-1c"
        ],
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot-copy",
        "DBClusterIdentifier": "sample-cluster",
        "SnapshotCreateTime": "2020-03-27T08:40:24.805Z",
        "Engine": "docdb",
        "Status": "copying",
        "Port": 0,
        "VpcId": "vpc-abcd0123",
        "ClusterCreateTime": "2020-01-10T22:13:38.261Z",
        "MasterUsername": "master-user",
        "EngineVersion": "4.0.0",
        "SnapshotType": "manual",
        "PercentProgress": 0,
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:111122223333:key/sample-key-id",
        "DBClusterSnapshotArn": "arn:aws:rds:us-east-1:111122223333:cluster-
snapshot:sample-cluster-snapshot-copy",
        "SourceDBClusterSnapshotArn": "arn:aws:rds:us-east-1:111122223333:cluster-
snapshot:sample-cluster-snapshot"
    }
}
```

Example

範例 2：跨 複製未加密的快照 AWS 區域

下列 AWS CLI 範例會建立具有 sample-cluster-snapshotARN 的 複本arn:aws:rds:us-east-1:123456789012:cluster-snapshot:sample-cluster-snapshot。此副本名為 sample-cluster-snapshot-copy，位於執行命令 AWS 區域 的 中。

針對 Linux、macOS 或 Unix：

```
aws docdb copy-db-cluster-snapshot \
    --source-db-cluster-snapshot-identifier arn:aws:rds:us-
east-1:123456789012:cluster-snapshot:sample-cluster-snapshot \
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy
```

針對 Windows：

```
aws docdb copy-db-cluster-snapshot ^
    --source-db-cluster-snapshot-identifier arn:aws:rds:us-
east-1:123456789012:cluster-snapshot:sample-cluster-snapshot ^
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy
```

此操作的輸出將會如下所示。

```
{
    "DBClusterSnapshot": {
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1b",
            "us-east-1c"
        ],
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot-copy",
        "DBClusterIdentifier": "sample-cluster",
        "SnapshotCreateTime": "2020-04-29T16:45:51.239Z",
        "Engine": "docdb",
        "AllocatedStorage": 0,
        "Status": "copying",
        "Port": 0,
        "VpcId": "vpc-abc0123",
        "ClusterCreateTime": "2020-04-28T16:43:00.294Z",
        "MasterUsername": "master-user",
        "EngineVersion": "4.0.0",
```

```
        "LicenseModel": "docdb",
        "SnapshotType": "manual",
        "PercentProgress": 0,
        "StorageEncrypted": false,
        "DBClusterSnapshotArn": "arn:aws:rds:us-east-1:123456789012:cluster-
snapshot:sample-cluster-snapshot-copy",
        "SourceDBClusterSnapshotArn": "arn:aws:rds:us-east-1:123456789012:cluster-
snapshot:sample-cluster-snapshot",
    }
}
```

Example

範例 3：跨 複製加密快照 AWS 區域

下列 AWS CLI 範例會建立sample-cluster-snapshot從 us-west-2 區域到 us-east-1 區域的 複本。此命令會在 us-east-1 區域中呼叫。

針對 Linux、macOS 或 Unix：

```
aws docdb copy-db-cluster-snapshot \
    --source-db-cluster-snapshot-identifier arn:aws:rds:us-
west-2:123456789012:cluster-snapshot:sample-cluster-snapshot \
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy \
    --source-region us-west-2 \
    --kms-key-id sample-us-east-1-key
```

針對 Windows：

```
aws docdb copy-db-cluster-snapshot ^
    --source-db-cluster-snapshot-identifier arn:aws:rds:us-
west-2:123456789012:cluster-snapshot:sample-cluster-snapshot ^
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy ^
    --source-region us-west-2 ^
    --kms-key-id sample-us-east-1-key
```

此操作的輸出將會如下所示。

```
{
    "DBClusterSnapshot": {
        "AvailabilityZones": [],
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot-copy",
```

```
        "DBClusterIdentifier": "sample-cluster",
        "SnapshotCreateTime": "2020-04-29T16:45:53.159Z",
        "Engine": "docdb",
        "AllocatedStorage": 0,
        "Status": "copying",
        "Port": 0,
        "ClusterCreateTime": "2020-04-28T16:43:07.129Z",
        "MasterUsername": "chimera",
        "EngineVersion": "4.0.0",
        "LicenseModel": "docdb",
        "SnapshotType": "manual",
        "PercentProgress": 0,
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:123456789012:key/0846496a-
d48e-41c4-9353-86d8301d7e35",
        "DBClusterSnapshotArn": "arn:aws:rds:us-east-1:123456789012:cluster-
snapshot:sample-cluster-snapshot-copy",
        "SourceDBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:sample-cluster-snapshot",
    }
}
```

Example

範例 4：跨 複製未加密的共用快照 AWS 區域

下列 AWS CLI 範例 帳戶 - 會123456789012建立帳戶sample-cluster-snapshot共用的未加密叢集快照複本 - 999999999999 從 us-east-1 區域到 us-west-2 區域。此命令在 us-west-2 區域中呼叫。如需有關共用快照的詳細資訊，請參閱 共用快照。

針對 Linux、macOS 或 Unix：

```
aws docdb copy-db-cluster-snapshot \
--region us-west-2 \
--source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:999999999999:cluster-
snapshot:sample-cluster-snapshot \
--target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy
```

針對 Windows：

```
aws docdb copy-db-cluster-snapshot ^
--region us-west-2 ^
```

```
--source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:999999999999:cluster-
snapshot:sample-cluster-snapshot ^
--target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy
```

此操作的輸出將會如下所示。

```
{
 "DBClusterSnapshots": [
 {
 "AvailabilityZones": [],
 "DBClusterSnapshotIdentifier": "sample-cluster-snapshot-copy",
 "DBClusterIdentifier": "sample-cluster",
 "SnapshotCreateTime": "2025-08-22T11:27:00.497000+00:00",
 "Engine": "docdb",
 "Status": "copying",
 "Port": 0,
 "ClusterCreateTime": "2024-07-02T16:44:50.246000+00:00",
 "MasterUsername": "master-user",
 "EngineVersion": "5.0.0",
 "SnapshotType": "manual",
 "PercentProgress": 0,
 "StorageEncrypted": false,
 "DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:sample-cluster-snapshot-copy",
 "SourceDBClusterSnapshotArn": "arn:aws:rds:us-east-1:999999999999:cluster-
snapshot:sample-cluster-snapshot"
 }
 ]
}
```

Example

範例 5：將加密的共用快照複製到 AWS 區域

下列 AWS CLI 範例，帳戶 - 會123456789012建立帳戶sample-cluster-snapshot共用的加密叢集快照複本 - 999999999999 從 us-east-1 區域到 us-west-2 區域。目標快照會使用客戶受管 KMS 金鑰 - 加密arn:aws:kms:us-west-2:123456789012:key/6c1f3264-1797-472b-ba37-03011e682d28。此命令在 us-west-2 區域中呼叫。如需有關共用快照的詳細資訊，請參閱共用快照。

針對 Linux、macOS 或 Unix：

```
aws docdb copy-db-cluster-snapshot \
```

```
--region us-west-2 \
--source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:999999999999:cluster-
snapshot:sample-cluster-snapshot \
--target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy \
--kms-key-id arn:aws:kms:us-west-2:123456789012:key/6c1f3264-1797-472b-
ba37-03011e682d28
```

針對 Windows：

```
aws docdb copy-db-cluster-snapshot ^
--region us-west-2 ^
--source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:999999999999:cluster-
snapshot:sample-cluster-snapshot ^
--target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy ^
--kms-key-id arn:aws:kms:us-west-2:123456789012:key/6c1f3264-1797-472b-
ba37-03011e682d28
```

此操作的輸出將會如下所示。

```
{
 "DBClusterSnapshots": [
 {
 "AvailabilityZones": [],
 "DBClusterSnapshotIdentifier": "sample-cluster-snapshot-copy",
 "DBClusterIdentifier": "sample-cluster",
 "SnapshotCreateTime": "2025-08-22T11:27:00.497000+00:00",
 "Engine": "docdb",
 "Status": "copying",
 "Port": 0,
 "ClusterCreateTime": "2024-07-02T16:44:50.246000+00:00",
 "MasterUsername": "master-user",
 "EngineVersion": "5.0.0",
 "SnapshotType": "manual",
 "PercentProgress": 0,
 "StorageEncrypted": true,
 "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/6c1f3264-1797-472b-
ba37-03011e682d28",
 "DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:sample-cluster-snapshot-copy",
 "SourceDBClusterSnapshotArn": "arn:aws:rds:us-east-1:999999999999:cluster-
snapshot:sample-cluster-snapshot"
 }
 ]
```

```
}
```

> **Note**
>
> 如需加密快照複本的詳細資訊，請參閱 複製叢集快照加密。
> 如需靜態加密的詳細資訊，請參閱 加密靜態 Amazon DocumentDB 資料。

共用 Amazon DocumentDB 叢集快照

您可以使用 Amazon DocumentDB，以下列方式共用手動叢集快照：

- 共用手動叢集快照，無論是加密或未加密，都可讓授權 AWS 帳戶複製快照。
- 共用手動叢集快照，無論是加密或未加密，都可讓授權 AWS 帳戶直接從快照還原叢集，而不是複製叢集並從中還原叢集。

> **Note**
>
> 若要共用自動化叢集快照，請複製自動化快照來建立手動叢集快照，然後共用該複本。此程序也適用於 AWS Backup 產生的資源。

您可以與其他最多 20 個 共用手動快照 AWS 帳戶。您也可以將未加密的手動快照以公有形式共用，讓所有 帳戶都可使用此快照。將快照以公有形式共用時，請確保不要在任何公有快照中包含您的私人資訊。

與其他 共用手動快照， AWS 帳戶並使用 AWS CLI 或 Amazon DocumentDB API 從共用快照還原叢集時，您必須指定共用快照的 Amazon Resource Name (ARN) 做為快照識別符。

共用加密快照

共用加密快照有下列限制：

- 您無法將加密快照以公有形式共用。
- 您無法共用已使用共用快照之帳戶的預設 AWS KMS 加密金鑰加密的快照。

請按照以下步驟共用加密的快照。

1. 與您想要能夠存取快照的任何帳戶共用用來加密快照的 AWS Key Management Service (AWS KMS) 加密金鑰。

   您可以將其他 AWS 帳戶新增至金鑰政策，以與其他帳戶共用 AWS KMS 加密 AWS KMS 金鑰。如需更新金鑰政策的詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》中的在 AWS KMS 中使用金鑰政策。有關建立金鑰政策的範例，請參閱本主題稍後的建立 IAM 政策以允許複製加密快照。
2. 使用 與其他帳戶共用加密快照 AWS CLI，如下所示。

允許存取 AWS KMS 加密金鑰

若要 AWS 帳戶 讓另一個 複製從您的帳戶共用的加密快照，您與之共用快照的帳戶必須能夠存取加密快照的 AWS KMS 金鑰。若要允許另一個帳戶存取 AWS KMS 金鑰，請將 AWS KMS 金鑰的 AWS KMS 金鑰政策更新為您在金鑰政策中以委託人身分共用之帳戶的 ARN。然後允許 `kms:CreateGrant` 動作。

授予 帳戶存取 AWS KMS 加密金鑰的權限後，若要複製加密快照，如果該帳戶還沒有加密快照，則必須建立 AWS Identity and Access Management (IAM) 使用者。此外，該帳戶也必須將 IAM 政策連接至該 IAM 使用者，以允許使用者使用您的 AWS KMS 金鑰複製加密快照。帳戶必須是 IAM 使用者，且由於 AWS KMS 安全限制，不能是根 AWS 帳戶 身分。

在下列金鑰政策範例中，使用者 123451234512 是 AWS KMS 加密金鑰的擁有者。使用者 123456789012 是共用金鑰的帳戶。此更新的金鑰政策可讓帳戶存取 AWS KMS 金鑰。它透過將使用者 123456789012 根 AWS 帳戶 身分的 ARN 作為政策的主體，以及允許 `kms:CreateGrant`動作來執行此操作。

JSON

```
{
    "Id": "key-policy-1",
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {"AWS": [
                "arn:aws:iam::123451234512:user/KeyUser",
                "arn:aws:iam::123456789012:root"
            ]},
```

```
            "Action": [
                "kms:CreateGrant",
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
            "Resource": "*"},
            {
            "Sid": "Allow attachment of persistent resources",
            "Effect": "Allow",
            "Principal": {"AWS": [
                "arn:aws:iam::123451234512:user/KeyUser",
                "arn:aws:iam::123456789012:root"
            ]},
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
            "Resource": "*",
            "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
        }
    ]
}
```

建立 IAM 政策以允許複製加密快照

當外部 AWS 帳戶 可以存取您的 AWS KMS 金鑰時，該帳戶的擁有者可以建立政策，以允許為帳戶建立的 IAM 使用者複製使用該 AWS KMS 金鑰加密的加密快照。

下列範例顯示可連接至 IAM 使用者 AWS 帳戶 123456789012 的政策。此政策可讓 IAM 使用者從已在 c989c1dd-a3f2-4a5d-8d96-e793d082ab26 us-west-2 區域中使用 AWS KMS 金鑰加密的帳戶 123451234512 複製共用快照。

JSON

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

```
        {
            "Sid": "AllowUseOfTheKey",
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey",
                "kms:CreateGrant",
                "kms:RetireGrant"
            ],
            "Resource": ["arn:aws:kms:us-west-2:123451234512:key/c989c1dd-
a3f2-4a5d-8d96-e793d082ab26"]
        },
        {
            "Sid": "AllowAttachmentOfPersistentResources",
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
            "Resource": ["arn:aws:kms:us-west-2:123451234512:key/c989c1dd-
a3f2-4a5d-8d96-e793d082ab26"],
            "Condition": {
                "Bool": {
                    "kms:GrantIsForAWSResource": true
                }
            }
        }
    ]
}
```

如需更新金鑰政策的詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》中的在 中使用金鑰政策 AWS KMS。

共用快照

您可以使用 或 共用 Amazon DocumentDB 手動叢集快照 (AWS 管理主控台 或自動化快照的副本) AWS CLI：

Using the AWS 管理主控台

若要使用 共用快照 AWS 管理主控台，請完成下列步驟：

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/docdb：// 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Snapshots (快照)。
3. 選取您要共享的手動快照。
4. 在動作下拉式功能表中，選擇共用。
5. 為資料庫快照可見性選擇下列其中一個選項：

   - 如果來源未加密，請選擇公有，以允許所有 AWS 帳戶從手動快照還原叢集。或者，選擇私有以僅允許您指定的 AWS 帳戶從手動快照還原叢集。

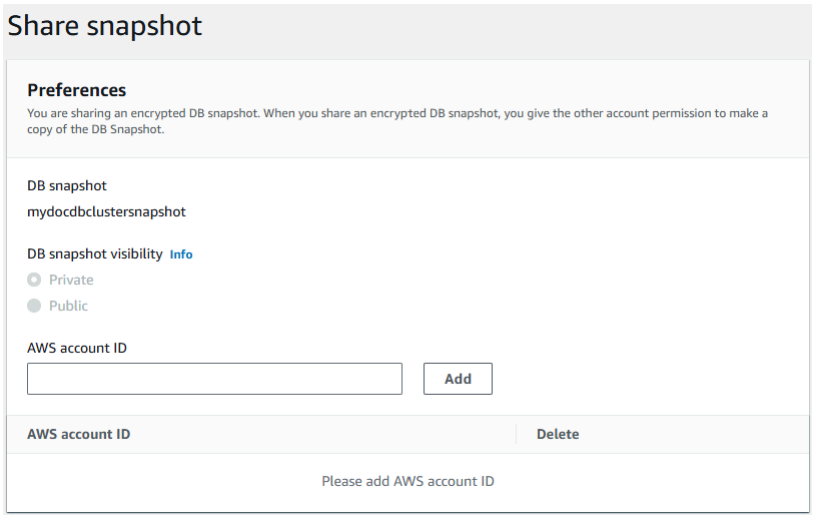
     > **Warning**
     >
     > 如果您將資料庫快照可見性設定為公有，則所有 AWS 帳戶都可以從手動快照還原叢集，並有權存取您的資料。請勿將任何包含私有資訊的手動叢集快照共享為公有。

   - 如果來源資料庫叢集已加密，DB snapshot visibility (資料庫快照可見度) 會設為 Private (私有)，因為加密快照無法以公有形式共用。

     > **Note**
     >
     > 使用預設加密的快照 AWS KMS key 無法共用。

6. 針對AWS 帳戶 ID，輸入您要允許 從手動快照還原叢集之帳戶 AWS 的帳戶識別符，然後選擇新增。重複 以包含額外的 AWS 帳戶識別符，最多 20 個 AWS 帳戶。

   如果您在將 AWS 帳戶識別符新增至允許的帳戶清單時發生錯誤，您可以選擇不正確 AWS 帳戶識別符右側的刪除，將其從清單中刪除。



7. 為要允許還原手動快照的所有 AWS 帳戶新增識別符後，請選擇儲存以儲存變更。

Using the AWS CLI

若要使用 共用快照 AWS CLI，請使用 Amazon DocumentDB `modify-db-snapshot-attribute`操作。使用 `--values-to-add` 參數來新增 AWS 帳戶 有權還原手動快照之 IDs 清單。

下列範例允許兩個 AWS 帳戶 識別符 123451234512 和 123456789012 還原名為 的快照`manual-snapshot1`。這還會移除 `all` 屬性值，以將快照標記為私有。

針對 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster-snapshot-attribute \
    --db-cluster-snapshot-identifier sample-cluster-snapshot \
    --attribute-name restore \
    --values-to-add '["123451234512","123456789012"]'
```

針對 Windows：

```
aws docdb modify-db-cluster-snapshot-attribute ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot ^
    --attribute-name restore ^
    --values-to-add '["123451234512","123456789012"]'
```

此操作的輸出將會如下所示。

```
{
```

```
    "DBClusterSnapshotAttributesResult": {
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot",
        "DBClusterSnapshotAttributes": [
            {
                "AttributeName": "restore",
                "AttributeValues": [
                    "123451234512",
                    "123456789012"
                ]
            }
        ]
    }
}
```

若要從清單中移除 AWS 帳戶 識別符，請使用 `--values-to-remove` 參數。下列範例可防止 AWS 帳戶 ID 123456789012 還原快照。

針對 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster-snapshot-attribute \
    --db-cluster-snapshot-identifier sample-cluster-snapshot \
    --attribute-name restore \
    --values-to-remove '["123456789012"]'
```

針對 Windows：

```
aws docdb modify-db-cluster-snapshot-attribute ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot ^
    --attribute-name restore ^
    --values-to-remove '["123456789012"]'
```

此操作的輸出將會如下所示。

```
{
    "DBClusterSnapshotAttributesResult": {
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot",
        "DBClusterSnapshotAttributes": [
            {
                "AttributeName": "restore",
                "AttributeValues": [
                    "123451234512"
```

```
                ]
            }
        ]
    }
}
```

從叢集快照還原

Amazon DocumentDB （與 MongoDB 相容） 會建立儲存磁碟區的叢集快照。您可以藉由從叢集快照進行還原來建立新叢集。還原叢集時，您需提供做為還原來源的叢集快照名稱，以及該還原所建立的新叢集名稱。您無法從快照還原到現有叢集，因為新叢集會在您還原時建立。

當您從叢集快照還原叢集時：

- 這個動作只會還原叢集，而不會還原該叢集的執行個體。您必須呼叫 `create-db-instance` 動作，為還原的叢集建立執行個體，並且在 `--db-cluster-identifier` 中指定所還原叢集的識別碼。只有當叢集可用時，才可以建立執行個體。
- 您無法將已加密的快照還原至未加密的叢集。不過，您可以透過指定 AWS KMS 金鑰，將未加密的快照還原至加密的叢集。
- 若要從加密快照還原叢集，您必須能夠存取 AWS KMS 金鑰。

> **Note**
>
> 您無法將 3.6 叢集還原至 4.0 叢集，但可以從一個叢集版本遷移到另一個叢集版本。如需詳細資訊，請前往 遷移和升級 Amazon DocumentDB。

Using the AWS 管理主控台

下列程序說明如何使用 Amazon DocumentDB 管理主控台從叢集快照還原 Amazon DocumentDB 叢集。

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Snapshots (快照)，找到您要用於還原至叢集的快照，並選擇其左側的按鈕。

> **Tip**
>
> 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在操作功能表上，選擇 Restore (還原)。
4. 在 Restore snapshot (還原快照) 頁面，完成 Configuration (組態設定) 區段。
   a. 叢集識別符 — 新叢集的名稱。您可以接受 Amazon DocumentDB 提供的名稱，或輸入您偏好的名稱。Amazon DocumentDBsupplied的名稱格式為 `docdb-`加上 UTC 時間戳記；例如 `docdb-`*`yyyy-mm-dd-hh-mm-ss*。
   b. 執行個體類別 — 新叢集的執行個體類別。您可以接受預設執行個體類別，或從下拉式清單中選擇執行個體類別。
   c. 執行個體數目 — 您希望使用此叢集建立的執行個體數目。您可以接受預設的 3 個執行個體 (1 個主要讀取/寫入和 2 個唯讀複本)，或從下拉式清單中選擇執行個體數目。
5. 針對叢集儲存組態，選擇儲存選項。

> **Note**
>
> Amazon DocumentDB I/O 最佳化儲存組態僅適用於 Amazon DocumentDB 5.0 和 8.0 引擎版本。

6. 如果您滿意此叢集組態，請選擇 Restore cluster (還原叢集)，等待叢集還原。
7. 如果您想要變更某些組態，例如指定非預設的 Amazon VPC 或安全群組，請選擇從頁面左下角顯示進階設定，然後繼續執行下列步驟。
   a. 完成 Network settings (網路設定) 區段。
      - Virtual Private Cloud (VPC) — 接受目前的 VPC，或從下拉式清單中選擇 VPC。
      - 子網路群組 — 接受`default`子網路群組，或從下拉式清單中選擇一個。
      - VPC 安全群組 — 接受`default (VPC)`安全群組，或從清單中選擇一個安全群組。
   b. 填寫 Cluster options (叢集選項) 區段。
      - 資料庫連接埠 — 接受預設連接埠 `27017`或使用向上或向下箭頭來設定您要用於應用程式連線的連接埠。
   c. 填寫 Encryption (加密) 區段。

- 靜態加密 — 如果您的快照已加密，則無法使用這些選項。如果未加密，您可以選擇以下其中一項：
  - 若要加密所有叢集的資料，請選擇 Enable encryption-at-rest (啟用靜態加密)。如果您選擇此選項，則必須指定 KMS 金鑰。
  - 若不加密叢集的資料，請選擇 Disable encryption-at-rest (停用靜態加密)。如果選擇此選項，您會在加密區段完成。
- AWS KMS 金鑰 — 從下拉式清單中選擇下列其中一項：
  - （預設） aws/rds — 帳戶號碼和 AWS KMS 金鑰 ID 會列在此選項後面。
  - 客戶受管金鑰 — 只有在您在 AWS Identity and Access Management (IAM) 主控台中建立 IAM 加密金鑰時，才能使用此選項。您可以選擇金鑰加密您的叢集。
  - 輸入金鑰 ARN — 在 ARN 方塊中，輸入 AWS KMS 金鑰的 Amazon Resource Name (ARN)。ARN 的格式是 `arn:aws:kms:<region>:<accountID>:key/<key-id>`。

d. 完成 Log exports (日誌匯出) 區段。

- 選取要發佈至 CloudWatch 的日誌類型 — 選擇下列其中一項：
  - 已啟用 — 讓您的叢集將 DDL 記錄匯出至 Amazon CloudWatch Logs。
  - 停用 — 防止您的叢集將 DDL 日誌匯出至 Amazon CloudWatch Logs。Disabled (已停用) 是預設值。
- IAM 角色 - 從清單中選擇 RDS 服務連結角色。

e. 完成 Tags (標籤) 區段。

- 新增標籤 — 在金鑰方塊中，輸入叢集的標籤名稱。在 Value (值) 方塊中，選擇性地輸入標籤值。標籤會與 AWS Identity and Access Management (IAM) 政策搭配使用，以管理對 Amazon DocumentDB 資源的存取，並控制可套用至資源的動作。

f. 完成 Deletion protection (刪除保護) 區段。

- 啟用刪除保護 — 保護叢集免於意外刪除。啟用此選項後，您無法刪除叢集。

8. 選擇 Restore cluster (還原叢集)。

Using the AWS CLI

若要使用 從快照還原叢集 AWS CLI，請使用 `restore-db-cluster-from-snapshot`操作搭配下列參數。如需詳細資訊，請參閱 RestoreDBClusterFromSnapshot。

- **--db-cluster-identifier** – 必要。這項操作會建立的叢集名稱。執行此操作之前，該叢集名稱不可以存在。

  叢集命名限制條件：

  - 長度為 【1-63】 個字母、數字或連字號。
  - 第一個字元必須是字母。
  - 不能以連字號結尾，或包含兩個連續連字號。
  - 對於每個區域每個 Amazon RDS、Neptune 和 Amazon DocumentDB 的所有叢集 AWS 帳戶，必須是唯一的。
- **--snapshot-identifier** – 必要。用來還原的快照名稱。此名稱的快照必須存在且位於可用狀態。
- **--engine** – 必要。必須為 docdb。
- **--storage-type standard | iopt1** — 選用。預設：standard。
- **--kms-key-id** — 選用。從未加密快照還原加密快照或加密叢集時要使用的 AWS KMS 金鑰識別符 ARN。提供 AWS KMS 金鑰 ID 會導致還原的叢集使用 AWS KMS 金鑰加密，無論快照是否已加密。

  --kms-key-id 的格式是 arn:aws:kms:<region>:<accountID>:key/<key-id>。如果您未指定 --kms-key-id 參數的值，會發生以下情況：

  - 如果 中的快照--snapshot-identifier已加密，則還原的叢集會使用用來加密快照的相同 AWS KMS 金鑰進行加密。
  - 如果 --snapshot-identifier 中的快照未加密，則不會加密已還原的叢集。

針對 Linux、macOS 或 Unix：

```
aws docdb restore-db-cluster-from-snapshot \
    --db-cluster-identifier sample-cluster-restore \
    --snapshot-identifier sample-cluster-snapshot \
    --engine docdb \
    --kms-key-id arn:aws:kms:us-east-1:123456789012:key/SAMPLE-KMS-KEY-ID
```

針對 Windows：

```
aws docdb restore-db-cluster-from-snapshot ^
    --db-cluster-identifier sample-cluster-restore ^
    --snapshot-identifier sample-cluster-snapshot ^
```

```
    --engine docdb ^
    --kms-key-id arn:aws:kms:us-east-1:123456789012:key/SAMPLE-KMS-KEY-ID
```

此操作的輸出將會如下所示。

```
{
    "DBCluster": {
        "AvailabilityZones": [
            "us-east-1c",
            "us-east-1b",
            "us-east-1a"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "sample-cluster-restore",
        "DBClusterParameterGroup": "default.docdb4.0",
        "DBSubnetGroup": "default",
        "Status": "creating",
        "Endpoint": "sample-cluster-restore.cluster-node.us-
east-1.docdb.amazonaws.com",
        "ReaderEndpoint": "sample-cluster-restore.cluster-node.us-
east-1.docdb.amazonaws.com",
        "MultiAZ": false,
        "Engine": "docdb",
        "EngineVersion": "4.0.0",
        "Port": 27017,
        "MasterUsername": "<master-user>",
        "PreferredBackupWindow": "02:00-02:30",
        "PreferredMaintenanceWindow": "tue:09:50-tue:10:20",
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcdefgh",
                "Status": "active"
            }
        ],
        "HostedZoneId": "ABCDEFGHIJKLM",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/<sample-key-id>",
        "DbClusterResourceId": "cluster-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "DBClusterArn": "arn:aws:rds:us-east-1:<accountID>:cluster:sample-cluster-
restore",
        "AssociatedRoles": [],
        "ClusterCreateTime": "2020-04-01T01:43:40.871Z",
```

```
        "DeletionProtection": true
    }
}
```

當叢集狀態可用時，為叢集至少建立一個執行個體。

針對 Linux、macOS 或 Unix：

```
aws docdb create-db-instance \
    --db-cluster-identifier sample-cluster-restore  \
    --db-instance-identifier sample-cluster-restore-instance \
    --availability-zone us-east-1b \
    --promotion-tier 2 \
    --db-instance-class db.r5.large \
    --engine docdb
```

針對 Windows：

```
aws docdb create-db-instance ^
    --db-cluster-identifier sample-cluster-restore  ^
    --db-instance-identifier sample-cluster-restore-instance ^
    --availability-zone us-east-1b ^
    --promotion-tier 2 ^
    --db-instance-class db.r5.large ^
    --engine docdb
```

此操作的輸出將會如下所示。

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "sample-cluster-restore-instance",
        "DBInstanceClass": "db.r5.large",
        "Engine": "docdb",
        "DBInstanceStatus": "creating",
        "PreferredBackupWindow": "02:00-02:30",
        "BackupRetentionPeriod": 1,
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcdefgh",
                "Status": "active"
            }
        ],
```

```
        "AvailabilityZone": "us-west-2b",
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-6242c31a",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-abcdefgh",
                    "SubnetAvailabilityZone": {
                        "Name": "us-west-2a"
                    },
                    "SubnetStatus": "Active"
                },
                {
                    ...
                }
            ]
        },
        "PreferredMaintenanceWindow": "fri:09:43-fri:10:13",
        "PendingModifiedValues": {},
        "EngineVersion": "4.0.0",
        "AutoMinorVersionUpgrade": true,
        "PubliclyAccessible": false,
        "DBClusterIdentifier": "sample-cluster-restore",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/<sample-key-id>",
        "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "CACertificateIdentifier": "rds-ca-2019",
        "PromotionTier": 2,
        "DBInstanceArn": "arn:aws:rds:us-east-1:<accountID>:db:sample-cluster-
restore-instance"
    }
}
```

還原至某個時間點

您可以使用 AWS 管理主控台 或 AWS Command Line Interface ()，將叢集還原至叢集備份保留期間內的任何時間點AWS CLI。

> **Note**
>
> 您無法將 point-in-time還原至 4.0 叢集，但可以從一個叢集版本遷移至另一個叢集版本。如需詳細資訊，請前往 遷移和升級 Amazon DocumentDB。

將叢集還原至某個時間點時，請謹記下列事項。

- 除非是以預設的參數群組建立新叢集，否則會以與來源叢集相同的組態建立新叢集。若要將新叢集的參數群組設為來源叢集的參數群組，請在叢集狀態變為「可用」後修改它。如需修改叢集的詳細資訊，請參閱 修改 Amazon DocumentDB 叢集。

Using the AWS 管理主控台

您可以藉由使用 AWS 管理主控台完成下列項目，以將叢集還原至其備份保留期間內的某個時間點。

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。在叢集清單中，選擇您要還原之叢集左側的按鈕。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在 Actions (動作) 功能表中，選擇 Restore to point in time (還原至時間點)。
4. 填妥 Restore time (還原時間) 部分，這裡指定還原的日期和時間。

   a. 還原日期 - 選擇或輸入最早還原時間和最晚還原時間之間的日期。

   b. 還原時間 - 選擇或輸入最早還原時間和最晚還原時間之間的小時、分鐘和秒數。
5. 填妥 Configuration (組態) 部分。

   a. 叢集識別符 — 接受預設識別符，或輸入您偏好的識別符。

   叢集命名限制條件：

   - 長度為 【1-63】 個字母、數字或連字號。

- 第一個字元必須是字母。
- 不能以連字號結尾，或包含兩個連續連字號。
- 對於每個區域每個 Amazon RDS、Neptune 和 Amazon DocumentDB 的所有叢集 AWS 帳戶，必須是唯一的。

b. 執行個體類別 — 從下拉式清單中，選擇您要用於叢集執行個體的執行個體類別。

c. 執行個體數量 — 從下拉式清單中，選擇您要在叢集還原時建立的執行個體數量。

6. 針對叢集儲存組態，選擇儲存選項。

> **Note**
>
> Amazon DocumentDB I/O 最佳化儲存組態僅適用於 Amazon DocumentDB 5.0 和 8.0 引擎版本。

7. 選用。若要設定網路設定、叢集選項並啟用日誌匯出，請選擇 Show advanced settings (顯示進階設定)，然後完成下列區段。否則，請繼續至下一個步驟。

- Network settings (網路設定)
  1. Virtual Private Cloud (VPC) — 從下拉式清單中，選擇您要用於此叢集的 VPC。
  2. 子網路群組 — 從下拉式清單中，選擇此叢集的子網路群組。
  3. VPC 安全群組 — 從下拉式清單中，選擇此叢集的 VPC 安全群組。

- Cluster options (叢集選項)
  1. 連接埠 — 接受預設連接埠 (27017)，或使用向上和向下箭頭設定連接埠以與此叢集通訊。

- Log exports (日誌匯出)
  1. 稽核日誌 — 選取此選項可啟用將稽核日誌匯出至 Amazon CloudWatch Logs。如果您選取此選項，則必須在叢集的自訂參數群組中啟用 `audit_logs`。如需詳細資訊，請參閱稽核 Amazon DocumentDB 事件。
  2. Profiler 日誌 — 選取此選項可啟用將操作 Profiler 日誌匯出至 Amazon CloudWatch Logs。如果您選取此選項，則還必須在叢集的自訂參數群組中修改下列參數：
     - `profiler` — 設定為 `enabled`。
     - `profiler_threshold_ms` — 設定為值`[0-INT_MAX]`以設定分析操作的閾值。

- `profiler_sampling_rate` — 設定為值`[0.0-1.0]`，將慢速操作的百分比設定為設定檔。

  如需詳細資訊，請參閱分析 Amazon DocumentDB 操作。

3. Profiler 日誌 — 將 Profiler 日誌匯出至 Amazon CloudWatch
4. IAM 角色 — 從下拉式清單中，選擇 RDS 服務連結角色。

- Tags (標籤)
  1. 新增標籤 — 在金鑰方塊中，輸入叢集的標籤名稱。在 Value (值) 方塊中，選擇性地輸入標籤值。標籤會與 AWS Identity and Access Management (IAM) 政策搭配使用，以管理對 Amazon DocumentDB 資源的存取，並控制可套用至資源的動作。
- 刪除保護
  1. 啟用刪除保護 — 保護叢集免於意外刪除。啟用此選項後，您無法刪除叢集。

8. 若要還原叢集，請選擇 Create cluster (建立叢集)。或者，您可以選擇 Cancel (取消) 以取消操作。

Using the AWS CLI

若要將叢集還原至快照備份保留期內的某個時間點，請搭配下列參數使用 `restore-db-cluster-to-point-in-time` 操作。

- **`--db-cluster-identifier`**— 必要。要建立之新叢集的名稱。操作前，此叢集不可以存在。參數值必須滿足下列限制條件。

  叢集命名限制條件：

  - 長度為 【1-63】 個字母、數字或連字號。
  - 第一個字元必須是字母。
  - 不能以連字號結尾，或包含兩個連續連字號。
  - 對於每個區域每個 Amazon RDS、Neptune 和 Amazon DocumentDB 的所有叢集 AWS 帳戶，必須是唯一的。
- **`--restore-to-time`** — 還原叢集的 UTC 日期和時間。例如 `2018-06-07T23:45:00Z`。

  時間限制：

  - 必須在叢集的最近一次可還原時間之前。
  - 如果未提供 `--use-latest-restorable-time` 參數，則必須指定。
  - 如果 `--use-latest-restorable-time` 參數為 `true`，則無法指定。

  - 如果 --restore-type 參數值為 copy-on-write，則無法指定。
- **--source-db-cluster-identifier** — 要從中還原的來源叢集名稱。此叢集必須存在並且為可用。
- **--use-latest-restorable-time** 或 **--no-use-latest-restorable-time** — 是否還原至最新的可還原備份時間。如果未提供 --restore-to-time 參數，則無法指定。
- **--storage-type standard | iopt1** — 選用。預設：standard。

AWS CLI 操作restore-db-cluster-to-point-in-time只會還原叢集，而不是該叢集的執行個體。您必須呼叫 create-db-instance 操作，為還原的叢集建立執行個體，並且在 --db-cluster-identifier 中指定所還原叢集的識別碼。只有在 restore-db-cluster-to-point-in-time 操作完成後，且還原的叢集為可用時，您才能建立執行個體。

Example

以下範例會從快照 sample-cluster-snapshot 的最近一次可還原時間建立 sample-cluster-restored。

針對 Linux、macOS 或 Unix：

```
aws docdb restore-db-cluster-to-point-in-time \
    --db-cluster-identifier sample-cluster-restored \
    --source-db-cluster-identifier sample-cluster-snapshot \
    --use-latest-restorable-time
```

針對 Windows：

```
aws docdb restore-db-cluster-to-point-in-time ^
    --db-cluster-identifier sample-cluster-restored ^
    --source-db-cluster-identifier sample-cluster-snapshot ^
    --use-latest-restorable-time
```

Example

以下範例會從快照 sample-cluster-snapshot 的 sample-cluster 備份保留期間中的 2018 年 12 月 11 日 03:15 (UTC) 建立 sample-cluster-restored。

針對 Linux、macOS 或 Unix：

```
aws docdb restore-db-cluster-to-point-in-time \
    --db-cluster-identifier sample-cluster-restore \
```

```
    --source-db-cluster-identifier sample-cluster \
    --restore-to-time 2020-05-12T03:15:00Z
```

針對 Windows：

```
aws docdb restore-db-cluster-to-point-in-time ^
    --db-cluster-identifier sample-cluster-restore ^
    --source-db-cluster-identifier sample-cluster ^
    --restore-to-time 2020-05-12T03:15:00Z
```

此操作的輸出將會如下所示。

```
{
    "DBCluster": {
        "AvailabilityZones": [
            "us-east-1c",
            "us-west-2b",
            "us-west-2a"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "sample-cluster-restored",
        "DBClusterParameterGroup": "sample-parameter-group",
        "DBSubnetGroup": "default",
        "Status": "creating",
        "Endpoint": "sample-cluster-restored.node.us-east-1.docdb.amazonaws.com",
        "ReaderEndpoint": "sample-cluster-restored.node.us-
east-1.docdb.amazonaws.com",
        "MultiAZ": false,
        "Engine": "docdb",
        "EngineVersion": "4.0.0",
        "Port": 27017,
        "MasterUsername": "master-user",
        "PreferredBackupWindow": "02:00-02:30",
        "PreferredMaintenanceWindow": "tue:09:50-tue:10:20",
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abc0123",
                "Status": "active"
            }
        ],
        "HostedZoneId": "ABCDEFGHIJKLM",
        "StorageEncrypted": true,
```

```
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID^>:key/sample-key",
        "DbClusterResourceId": "cluster-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "DBClusterArn": "arn:aws:rds:us-east-1:<accountID>:cluster:sample-cluster-
restored",
        "AssociatedRoles": [],
        "ClusterCreateTime": "2020-04-24T20:14:36.713Z",
        "DeletionProtection": false
    }
}
```

刪除叢集快照

手動快照是完整備份，只有在您使用 AWS 管理主控台 或 手動刪除它時才會刪除 AWS CLI。您無法手動刪除自動快照，只有在快照保留期間結束或您刪除快照叢集時，才會刪除自動快照。

Using the AWS 管理主控台

若要使用 刪除手動叢集快照 AWS 管理主控台，請完成下列步驟。

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/docdb：// 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Snapshots (快照)。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在快照清單中，找到您要刪除的快照，並選擇左側的按鈕。快照的類型必須為手動。

   1. 您可以檢查快照是否列為類型欄`automatic`下`manual`，以確認快照的類型為手動。
4. 在 Actions (動作) 選單中，選擇 Delete (刪除)。如果 Delete (刪除) 選項無法使用，表示您可能選擇了自動快照。
5. 在刪除確認畫面上，選擇 Delete (刪除) 來刪除快照。如要保留快照，請選擇 Cancel (取消)。

Using the AWS CLI

Amazon DocumentDB 手動叢集快照是您可以使用 手動刪除的完整備份 AWS CLI。您無法手動刪除自動快照。

若要使用 刪除手動叢集快照 AWS CLI，請使用 delete-db-cluster-snapshot操作搭配下列參數。

Parameters

- **--db-cluster-snapshot-identifier** – 必要。要刪除的手動快照名稱。

以下範例會刪除 sample-cluster-snapshot 叢集快照。

針對 Linux、macOS 或 Unix：

```
aws docdb delete-db-cluster-snapshot \
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

針對 Windows：

```
aws docdb delete-db-cluster-snapshot ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

此操作的輸出會列出您刪除之叢集快照的詳細資料。

在 Amazon DocumentDB 中監控和記錄

監控您的 AWS 服務是保持系統正常運作的重要部分。從 AWS 解決方案的所有部分收集監控資料，讓您可以更輕鬆地偵錯和修正故障或效能問題。開始監控解決方案 AWS 之前，請考慮下列問題：

- 監控目標是什麼？
- 您要監控哪些資源？
- 監控這些資源的頻率為何？
- 要使用哪些監控工具？
- 誰負責進行監控？
- 若發生錯誤，誰會收到通知以及透過何種方式？

若要了解目前的效能模式、識別效能異常，並制定方法來解決問題，您應該針對各種不同的時間、依不同的負載條件建立基準效能指標。當您監控解決方案 AWS 時，請存放歷史監控資料以供未來參考和建立基準。

一般來說，效能指標的可接受值視您的基準看起來像什麼，以及您的應用程式正在做什麼而定。調查距離基準的一致或趨勢變異。以下是關於特定類型指標的建議：

- 高 CPU 或 RAM 使用量 — 高 CPU 或 RAM 使用量值可能是適當的，前提是這些值符合您應用程式的目標（例如輸送量或並行），並且是預期的。
- 儲存磁碟區耗用 — 如果使用的空間持續等於或高於總儲存磁碟區空間的 85%，則調查儲存耗用 (`VolumeBytesUsed`)。判斷您是否能從儲存磁碟區刪除資料，或將資料存檔至不同的系統來釋出空間。如需詳細資訊，請參閱Amazon DocumentDB 儲存體及Amazon DocumentDB 配額。
- 網路流量 — 對於網路流量，請與您的系統管理員討論，以了解網域網路和網際網路連線的預期輸送量。調查網路流量的傳輸量是否如預期一致地降低。
- 資料庫連線 — 如果您看到大量使用者連線並降低執行個體效能和回應時間，請考慮限制資料庫連線。資料庫執行個體使用者連接的最佳數量，將因執行個體類別和要執行的操作複雜性而不同。
- IOPS 指標 — IOPS 指標的預期值取決於磁碟規格和伺服器組態，因此請使用基準來了解典型值。調查值是否與您的基準一致地不同。為獲得最佳 IOPS 效能，請確定您的一般工作集將放入記憶體中，以將讀取和寫入操作降到最低。

Amazon DocumentDB（與 MongoDB 相容）提供各種 Amazon CloudWatch 指標，您可以監控這些指標來判斷 Amazon DocumentDB 叢集和執行個體的運作狀態和效能。您可以使用各種工具檢視

Amazon DocumentDB 指標，包括 Amazon DocumentDB 主控台 AWS CLI、CloudWatch API 和績效詳情。

主題

- 監控 Amazon DocumentDB 叢集的狀態
- 監控 Amazon DocumentDB 執行個體的狀態
- 檢視 Amazon DocumentDB 建議
- 使用 Amazon DocumentDB 事件訂閱
- 使用 CloudWatch 監控 Amazon DocumentDB
- 使用 記錄 Amazon DocumentDB API 呼叫 AWS CloudTrail
- 分析 Amazon DocumentDB 操作
- 使用績效詳情進行監控

監控 Amazon DocumentDB 叢集的狀態

根據叢集運作狀態的叢集狀態。您可以使用 Amazon DocumentDB 主控台或 AWS CLI `describe-db-clusters` 命令來檢視叢集的狀態。

主題

- 叢集狀態值
- 監控叢集的狀態

叢集狀態值

下表列出叢集狀態的有效值。

叢集狀態	說明
`active`	叢集處於作用中狀態。此狀態僅適用於彈性叢集。
`available`	叢集運作正常且可供使用。此狀態僅適用於執行個體型叢集。

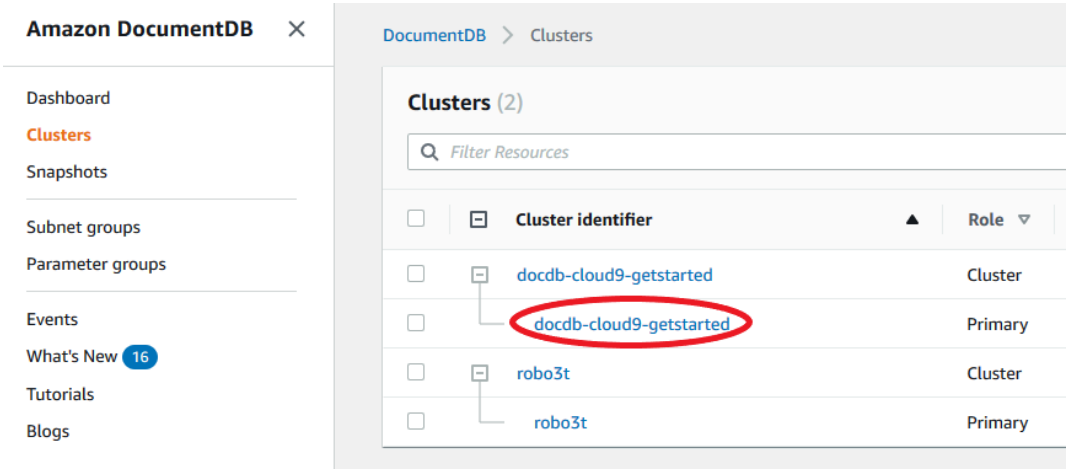
叢集狀態	說明
backing-up	目前正在備份資料庫叢集。
creating	正在建立叢集。正在建立時無法存取它。
deleting	正在刪除叢集。正在建立時無法將它刪除。
failing-over	正在從主要執行個體容錯移轉至 Amazon DocumentDB 複本。
inaccessible-encryption-credentials	用於加密或解密叢集的 AWS KMS 金鑰無法存取。
maintenance	正在將維護更新套用到叢集。此狀態用於 Amazon DocumentDB 事先排程的叢集層級維護。
migrating	正在將叢集快照還原至叢集。
migration-failed	無法進行遷移。
modifying	客戶請求修改叢集，因此正在進行修改。
renaming	客戶請求重新命名叢集，因此正在重新命名。
resetting-master-credentials	客戶請求重新設定叢集的主要登入資料，因此正在重新設定。
upgrading	正在升級叢集引擎版本。

監控叢集的狀態

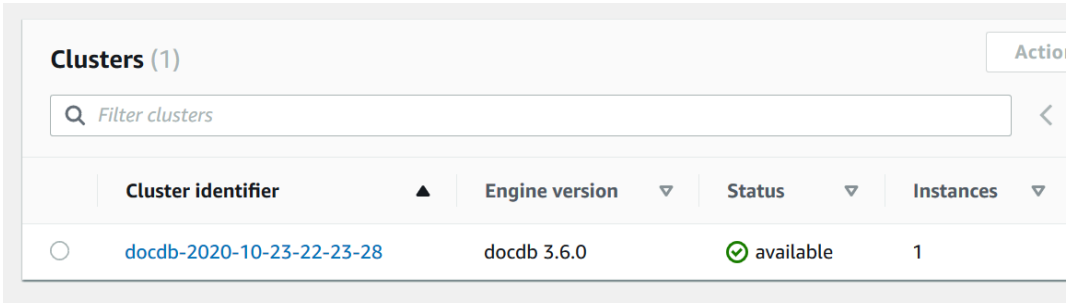
Using the AWS 管理主控台

使用 AWS 管理主控台 判斷叢集的狀態時，請使用下列程序。

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。
3. 在叢集導覽方塊中，您會看到叢集識別符欄。您的執行個體會列在叢集下，類似於下列螢幕擷取畫面。



4. 在叢集識別符欄中，尋找您感興趣的執行個體名稱。然後，若要尋找執行個體的狀態，請讀取該列中的狀態欄，如下所示。



Using the AWS CLI

使用 AWS CLI 判斷叢集的狀態時，請使用 `describe-db-clusters`操作。以下程式碼會尋找叢集 `sample-cluster` 的狀態。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster  \
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

針對 Windows：

```
aws docdb describe-db-clusters ^
    --db-cluster-identifier sample-cluster  ^
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

此操作的輸出將會如下所示。

```
[
    [
        "sample-cluster",
        "available"
    ]
]
```

監控 Amazon DocumentDB 執行個體的狀態

Amazon DocumentDB 提供資料庫中每個已設定執行個體目前條件的相關資訊。

您可以檢視 Amazon DocumentDB 執行個體的三種狀態類型：

- 執行個體狀態：此狀態會顯示在 中叢集資料表的狀態欄中， AWS 管理主控台 並顯示執行個體目前的生命週期條件。狀態欄中顯示的值衍生自 `DescribeDBCluster` API 回應的 `Status` 欄位。
- 執行個體運作狀態：此狀態會顯示在 中叢集資料表的執行個體運作狀態欄中， AWS 管理主控台 並顯示負責管理和擷取資料的資料庫引擎元件是否正在執行。執行個體運作狀態欄中顯示的值是以 Amazon CloudWatch `EngineUptime`系統指標為基礎。
- 維護狀態：此狀態會顯示在 中叢集資料表的維護欄中， AWS 管理主控台 並指出任何需要套用至執行個體的維護事件狀態。維護狀態獨立於其他執行個體狀態，且衍生自 `PendingMaintenanceAction` API。如需維護狀態的詳細資訊，請參閱維護 Amazon DocumentDB。

主題

- 執行個體狀態值
- 使用 AWS 管理主控台 或 監控執行個體狀態 AWS CLI
- 執行個體運作狀態值
- 使用 監控執行個體運作狀態 AWS 管理主控台

執行個體狀態值

下表列出執行個體的可能狀態值，以及每種狀態的計費方式。表中列出是否將依執行個體和儲存體計費、只針對儲存體計費、或不計費。在所有執行個體狀態下，一律要為備份用途付費。

執行個體狀態	計費	說明
available	計費	執行個體運作正常可供使用。
backing-up	計費	目前正在備份執行個體。
configuring-log-exports	計費	正在啟用或停用此執行個體的 Amazon CloudWatch Logs 發佈日誌檔案。
creating	不計費	正在建立執行個體。當執行個體正在建立時，無法存取執行個體。
deleting	不計費	正在刪除執行個體。
failed	不計費	執行個體失敗，Amazon DocumentDB 無法復原它。若要復原資料，請執行時間點還原，還原到執行個體最近一次可還原時間。
inaccessible-encryption-credentials	不計費	用於加密或解密執行個體的 AWS KMS 金鑰無法存取。
incompatible-network	不計費	Amazon DocumentDB 正在嘗試對執行個體執行復原動作，但無法這麼做，因為 VPC 處於阻止動作完成的狀態。例如，如果子網路中的所有可用 IP 地址都在使用中，且 Amazon DocumentDB 無法取得執行個體的 IP 地址，則可能會發生此狀態。
maintenance	計費	Amazon DocumentDB 正在將維護更新套用至執行個體。此狀態用於 Amazon DocumentDB 事先排程的執行個體層級維護。我們正在評估透過此狀態對客戶公開其他維護動作的方式。

執行個體狀態	計費	說明
modifying	計費	由於請求修改執行個體，因此正在修改執行個體。
rebooting	計費	由於請求或需要重新啟動執行個體的 Amazon DocumentDB 程序，執行個體正在重新啟動。
renaming	計費	由於請求重新命名執行個體，因此正在重新命名執行個體。
resetting-master-credentials	計費	由於請求重設執行個體的主要登入資料，因此正在重設主要登入資料。
restore-error	計費	執行個體在嘗試還原到某個時間點或從快照還原時發生錯誤。
starting	針對儲存計費	執行個體正在啟動。
stopped	針對儲存計費	執行個體已停止。
stopping	針對儲存計費	正在停止執行個體。
storage-full	計費	執行個體已達到其配置的儲存容量。這是一個重要的狀態，且應立即解決；請修改執行個體以擴展您的儲存。設定 Amazon CloudWatch 警示，在儲存空間不足時提醒您，以免遇到這種情況。

使用 AWS 管理主控台 或 監控執行個體狀態 AWS CLI

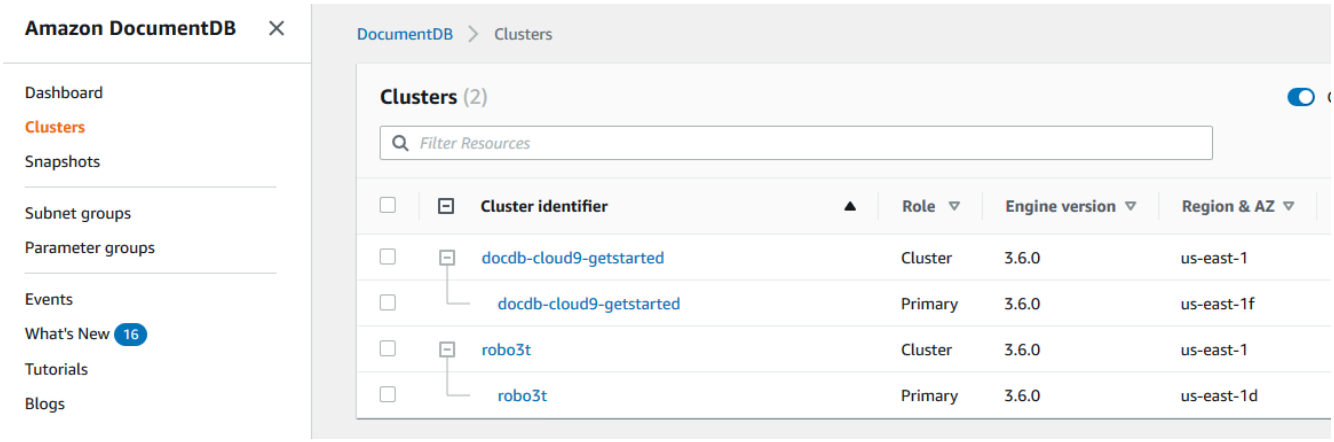
使用 AWS 管理主控台 或 AWS CLI 來監控執行個體的狀態。

Using the AWS 管理主控台

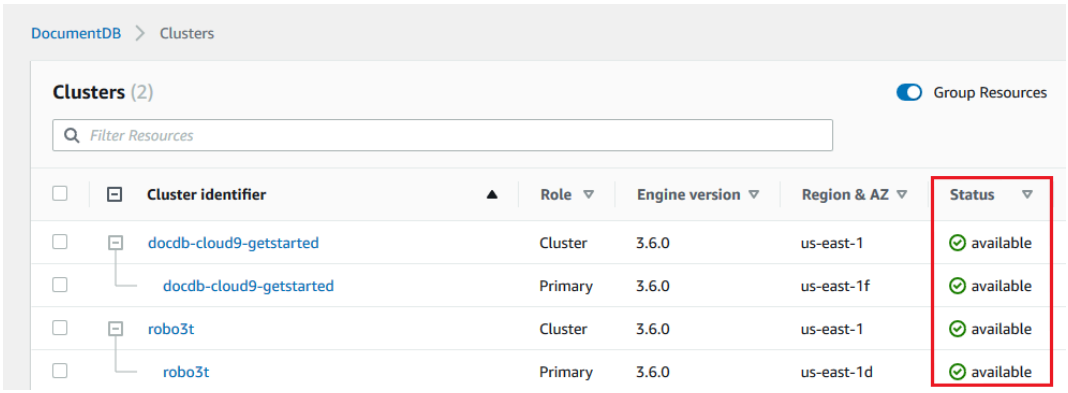
使用 AWS 管理主控台 判斷叢集的狀態時，請使用下列程序。

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

> **Note**
>
> 請注意，在叢集導覽方塊中，欄位叢集識別符會顯示叢集和執行個體。執行個體會列在叢集下方，如下圖所示。



3. 尋找您感興趣的執行個體名稱。然後，閱讀該列上的 Status (狀態) 欄找出執行個體的狀態，如下所示。



Using the AWS CLI

使用 AWS CLI 判斷叢集的狀態時，請使用 `describe-db-instances`操作。以下程式碼會尋找執行個體 `sample-cluster-instance-01` 的狀態。

針對 Linux、macOS 或 Unix：

```
aws docdb describe-db-instances \
          --db-instance-identifier sample-cluster-instance-01  \
          --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceStatus]'
```

針對 Windows：

```
aws docdb describe-db-instances ^
          --db-instance-identifier sample-cluster-instance-01  ^
          --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceStatus]'
```

此操作的輸出將會如下所示。

```
[
          [
              "sample-cluster-instance-01",
              "available"
          ]
      ]
```

執行個體運作狀態值

下表列出執行個體的可能運作狀態值。位於 叢集資料表中的執行個體運作狀態欄 AWS 管理主控台會顯示資料庫引擎、負責儲存、管理和擷取資料的元件是否正常運作。此欄也會指出 CloudWatch 中可用的`EngineUptime`系統指標是否顯示每個執行個體的運作狀態。

執行個體運作狀態	說明
狀態良好	資料庫引擎正在 Amazon DocumentDB 執行個體中執行。
運作狀態不佳	資料庫引擎未執行或在一分鐘內重新啟動。

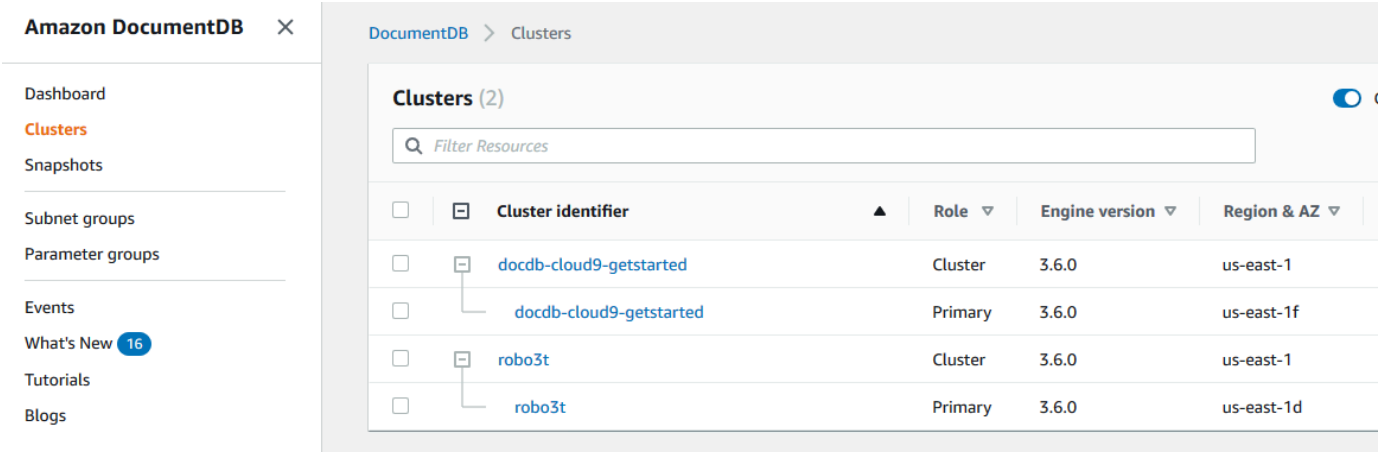
使用 監控執行個體運作狀態 AWS 管理主控台

使用 AWS 管理主控台 來監控執行個體的運作狀態。

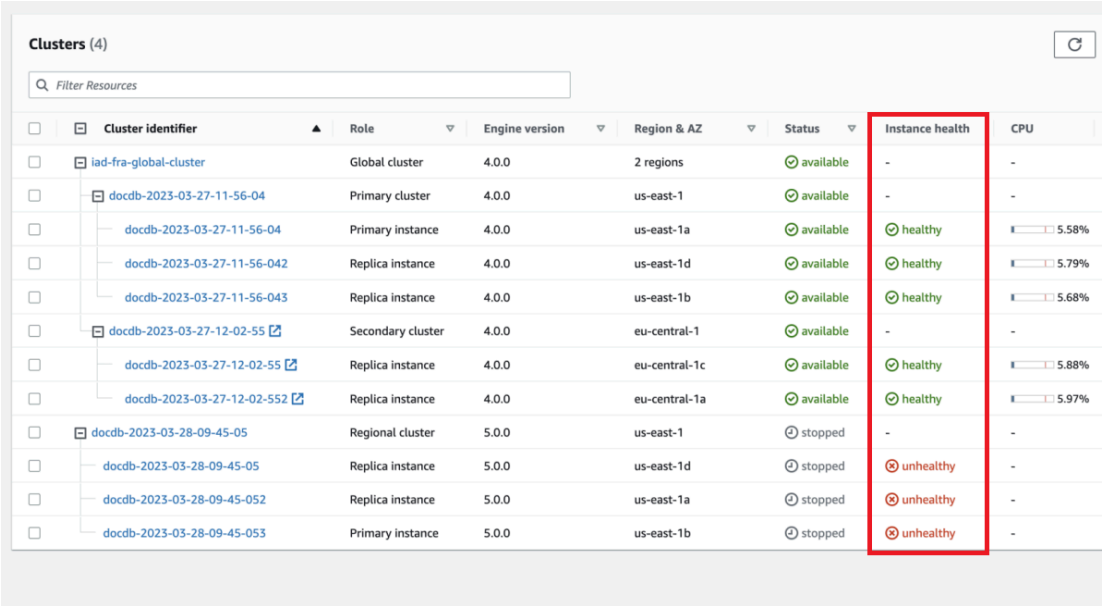
使用 時 AWS 管理主控台，請使用下列步驟來了解執行個體的運作狀態。

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

> **Note**
>
> 在叢集導覽方塊中，欄位叢集識別符會顯示叢集和執行個體。執行個體會列在叢集下方，如下圖所示。



3. 尋找您感興趣的執行個體名稱。然後，若要尋找執行個體的狀態，請讀取該列中的執行個體運作狀態欄，如下圖所示：



> **Note**
>
> 執行個體運作狀態輪詢每 60 秒進行一次，並以 CloudWatch `EngineUptime`系統指標為基礎。執行個體運作狀態欄中的值會自動更新。

檢視 Amazon DocumentDB 建議

Amazon DocumentDB 提供資料庫資源的自動化建議清單，例如執行個體和叢集。這些建議會分析您的叢集和執行個體組態，以提供最佳實務指引。

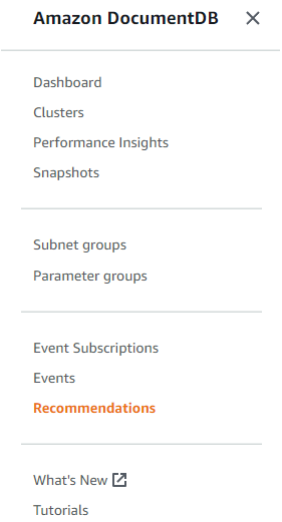
作為這些建議的範例，請參閱以下內容：

Type	描述	建議	其他資訊
一個 執行個體	叢集只包含一個執行個體	效能和可用性：在不同的可用區域中新增具有相同執行個體類別的另一個執行個體。	Amazon DocumentDB 高可用性和複寫

建立或修改資源時，Amazon DocumentDB 會產生資源的建議。Amazon DocumentDB 也會定期掃描您的資源並產生建議。

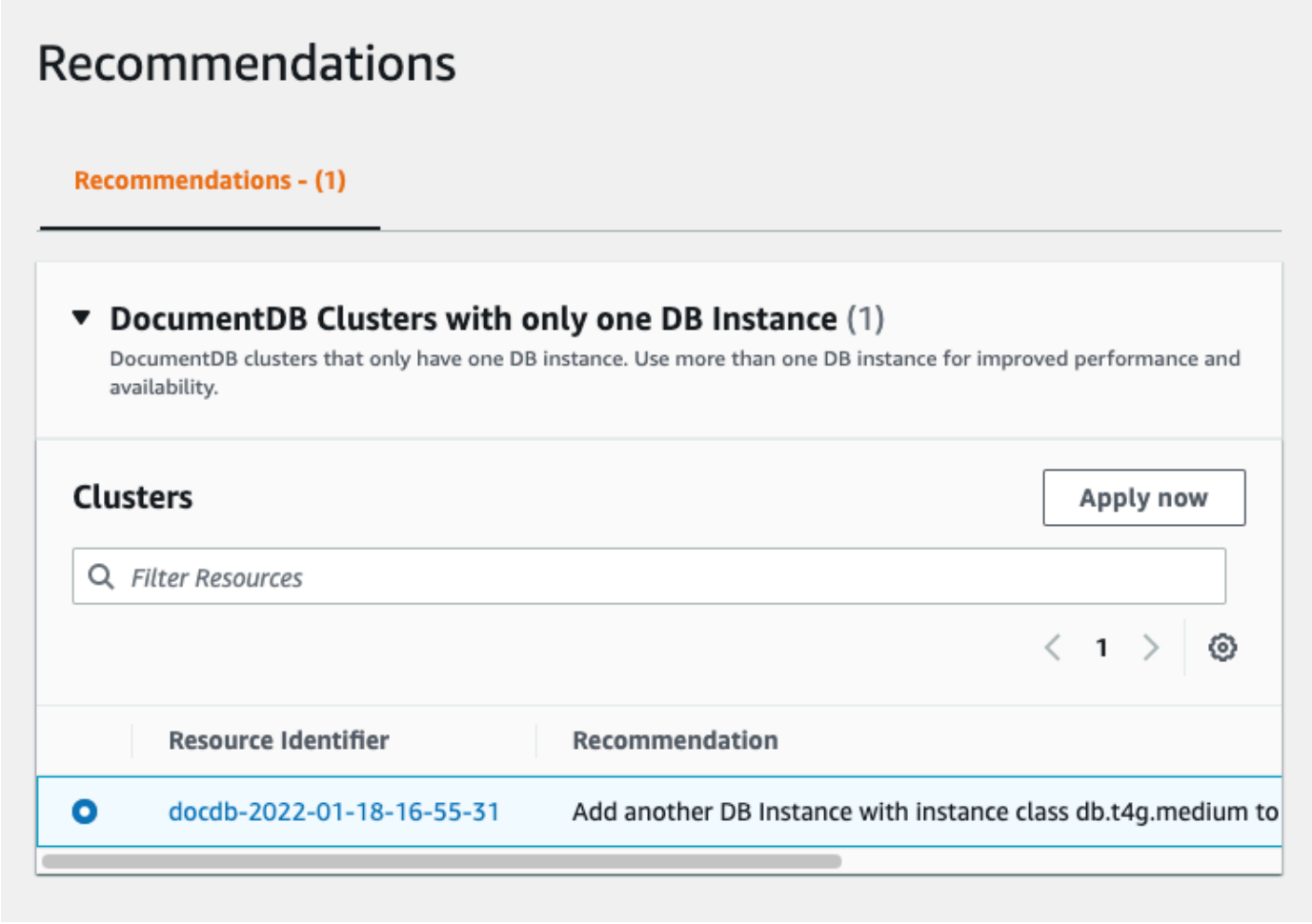
檢視並對 Amazon DocumentDB 建議採取動作

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇建議：



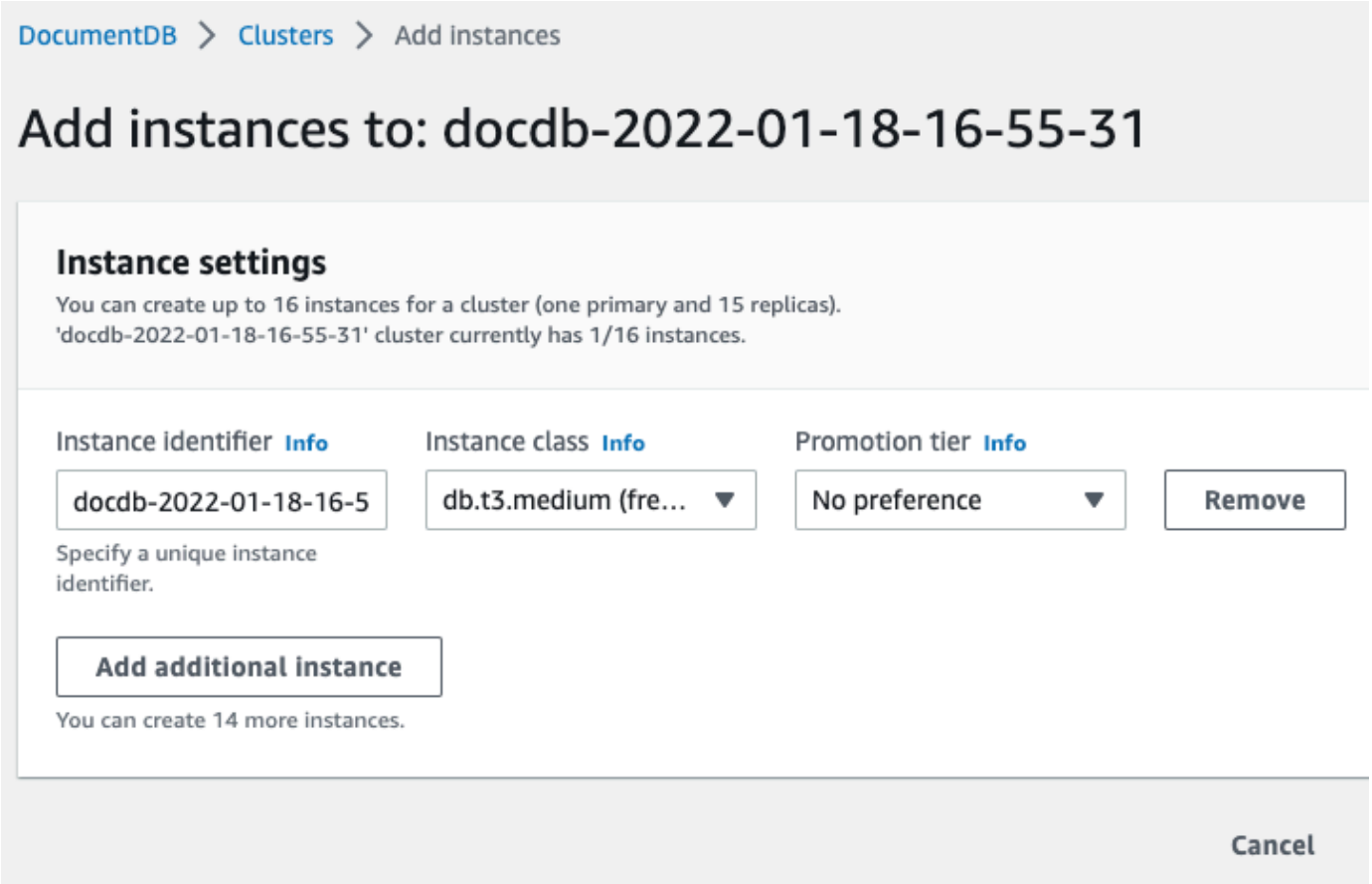
3. 在建議對話方塊中，展開感興趣的區段，然後選取建議的任務。

   在下列範例中，建議的任務會套用至只有一個執行個體的 Amazon DocumentDB 叢集。建議是新增另一個執行個體來改善效能和可用性。



4. 選擇 Apply now (立即套用)。

   在此範例中，會顯示新增執行個體對話方塊：



5. 修改新執行個體的設定，然後選擇建立。

使用 Amazon DocumentDB 事件訂閱

Amazon DocumentDB 使用 Amazon Simple Notification Service (Amazon SNS) 在 Amazon DocumentDB 事件發生時提供通知。這些通知可以是 Amazon SNS 支援的任何形式 AWS 區域，例如電子郵件、文字訊息或對 HTTP 端點的呼叫。

Amazon DocumentDB 將這些事件分組為您可以訂閱的類別，以便在發生該類別中的事件時收到通知。您可以訂閱執行個體、叢集、快照、叢集快照或參數群組的事件類別。例如，如果您訂閱指定執行個體的 Backup 類別，每當發生會影響執行個體的備份相關事件時，您都會收到通知。當事件訂閱變更時，您也會收到通知。

事件同時發生在叢集和執行個體層級，因此如果您訂閱叢集或執行個體，可以接收事件。

事件訂閱會傳送到您在建立訂閱時提供的地址。您可能想要建立數個不同的訂閱，例如接收所有事件通知的訂閱，以及僅包含生產執行個體關鍵事件的另一個訂閱。您可以輕鬆關閉通知，而無需刪除訂閱。若要這樣做，請在 Amazon DocumentDB 主控台中將已啟用選項按鈕設定為否。

> **Important**
>
> Amazon DocumentDB 不保證事件串流中傳送的事件順序。事件順序可能會改變。

Amazon DocumentDB 使用 Amazon SNS 主題的 Amazon Resource Name (ARN) 來識別每個訂閱。Amazon DocumentDB 主控台會在您建立訂閱時為您建立 ARN。

Amazon DocumentDB 事件訂閱的計費是透過 Amazon SNS。使用事件通知需要支付 Amazon SNS 費用。如需詳細資訊，請參閱 Amazon Simple Notification Service 定價。除了 Amazon SNS 費用之外，Amazon DocumentDB 不會收取事件訂閱的費用。

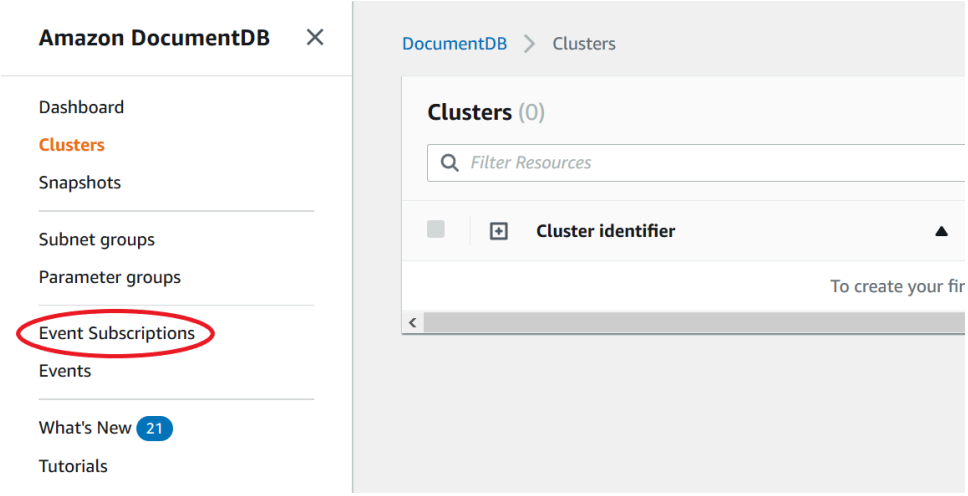
主題

- 訂閱 Amazon DocumentDB 事件
- 管理 Amazon DocumentDB 事件通知訂閱
- Amazon DocumentDB 事件類別和訊息

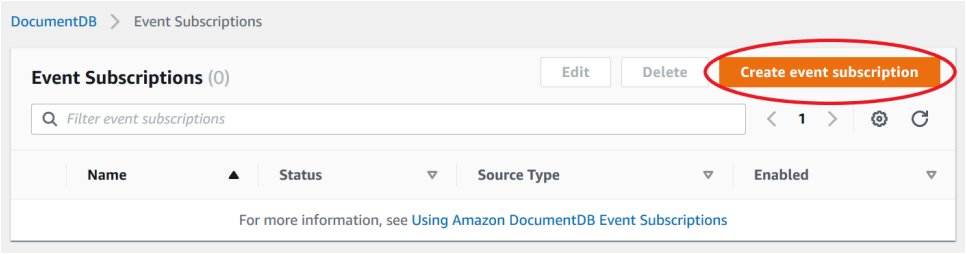
訂閱 Amazon DocumentDB 事件

您可以使用 Amazon DocumentDB 主控台來訂閱事件訂閱，如下所示：

1. 在 登入 AWS 管理主控台 https://console.aws.amazon.com/docdb。
2. 在導覽窗格中，選擇 Event subscriptions (事件訂閱)。

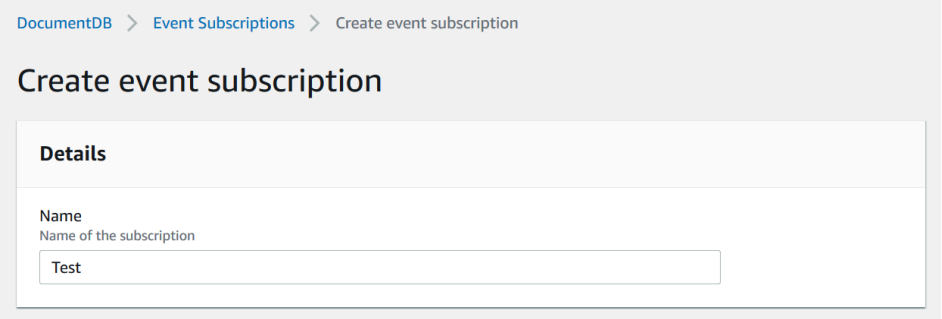


3. 在 Event subscriptions (事件訂閱) 窗格中，選擇 Create event subscription (建立事件訂閱)。

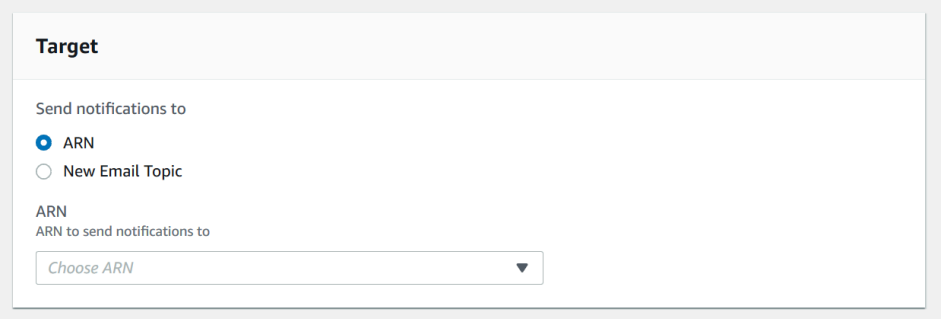


4. 在 Create event subscription (建立事件訂閱) 對話方塊中，執行下列動作：

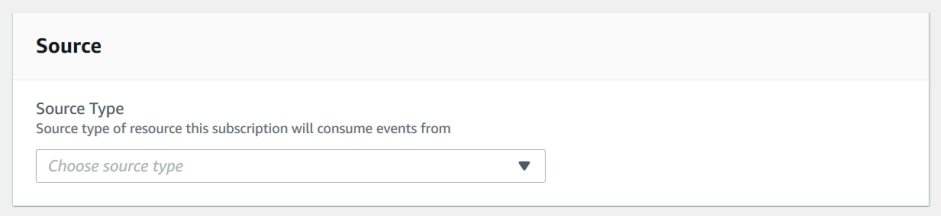
   - 在 Name (名稱) 中，輸入事件通知訂閱的名稱。



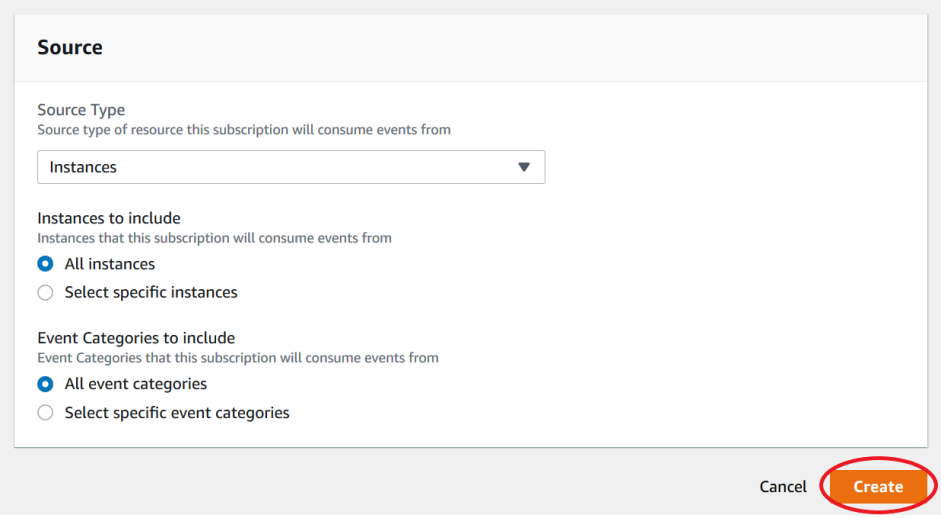
   - 針對目標，選擇您要傳送通知的目標。您可以選擇現有的 ARN 或選擇新增電子郵件主題，以輸入主題的名稱和收件人清單。



- 針對來源，選擇來源類型。根據您選擇的來源類型，選擇事件類別以及您要從中接收事件通知的來源。



- 選擇建立。



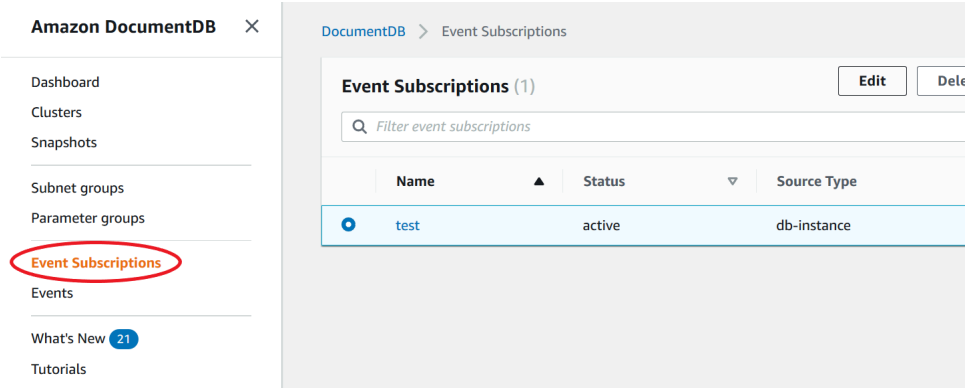
管理 Amazon DocumentDB 事件通知訂閱

如果您在 Amazon DocumentDB 主控台的導覽窗格中選擇事件訂閱，您可以檢視訂閱類別和目前訂閱的清單。您也可以修改或刪除特定訂閱。

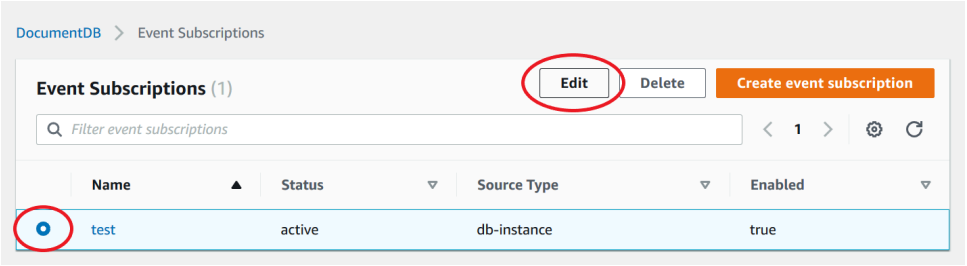
修改您目前的 Amazon DocumentDB 事件通知訂閱

1. 在 登入 AWS 管理主控台 https://console.aws.amazon.com/docdb。

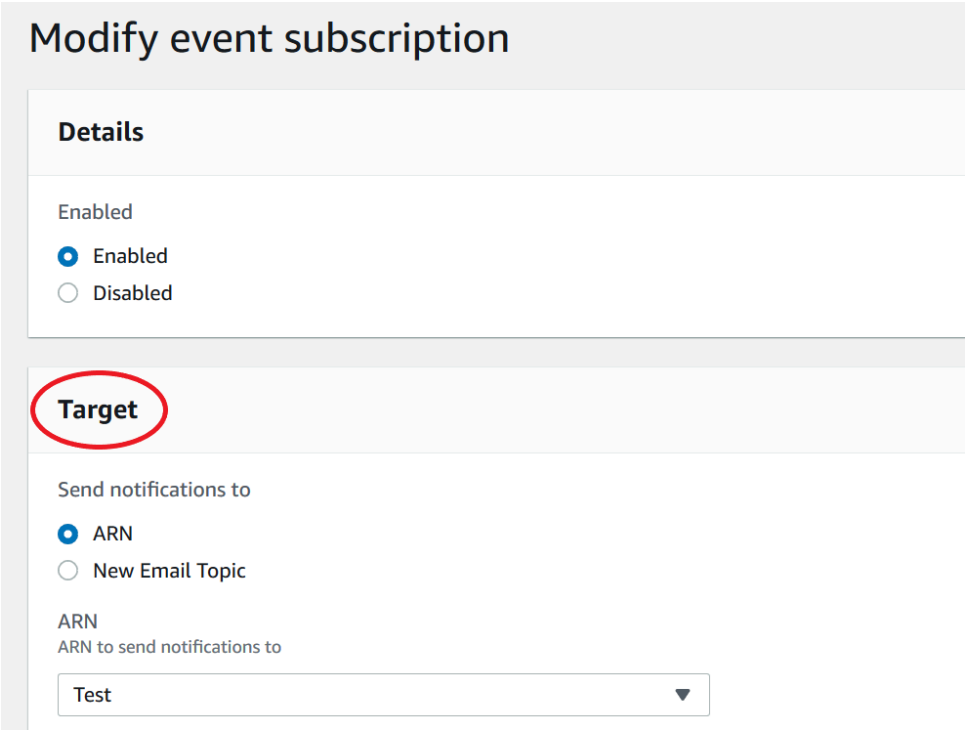
2. 在導覽窗格中，選擇 Event subscriptions (事件訂閱)。Event subscriptions (事件訂閱) 窗格顯示所有事件通知訂閱。



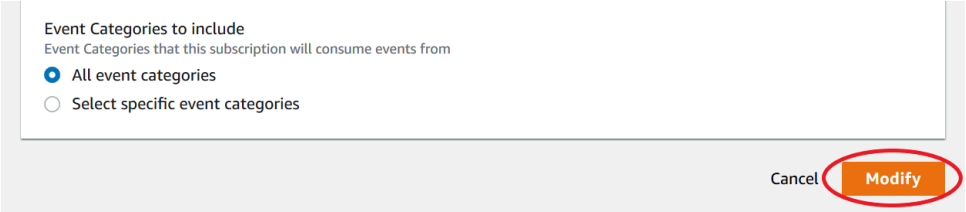
3. 在 Event subscriptions (事件訂閱) 窗格中，選擇您要修改的訂閱，然後選擇 Edit (編輯)。



4. 在 Target (目標) 或 Source (來源) 區段中，對訂閱進行變更。您可以新增或移除來源識別符，方法是在來源區段中選取或取消選取它們。



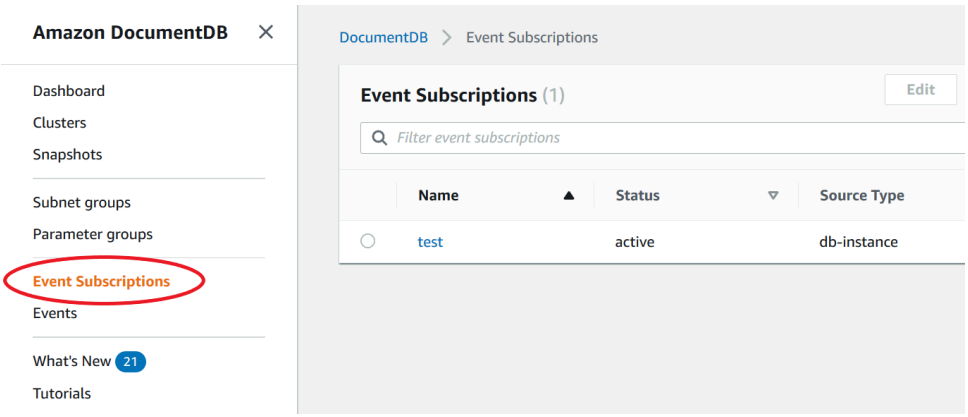
5. 選擇 Modify (修改)。Amazon DocumentDB 主控台指出正在修改訂閱。



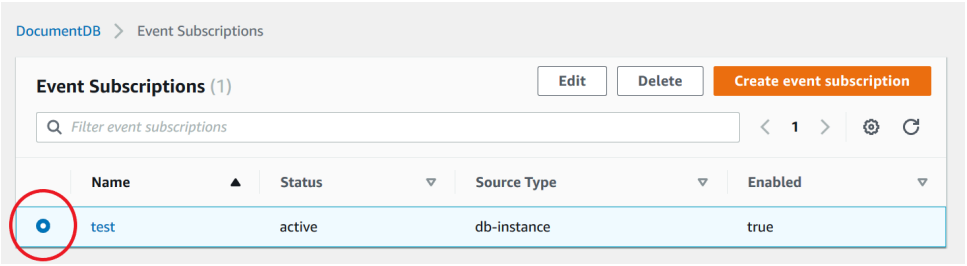
刪除 Amazon DocumentDB 事件通知訂閱

您可以刪除不再需要的訂閱。該主題的所有訂閱者將不會再收到該訂閱指定的事件通知。

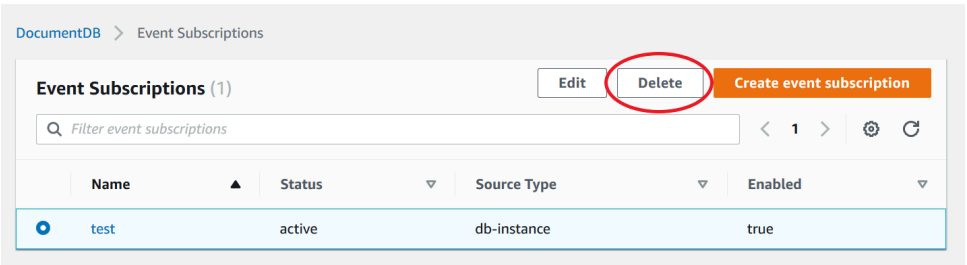
1. 在 登入 AWS 管理主控台 https://console.aws.amazon.com/docdb。
2. 在導覽窗格中，選擇 Event subscriptions (事件訂閱)。



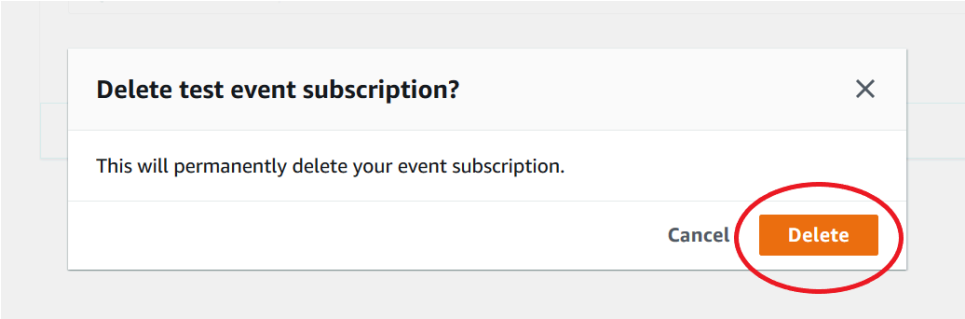
3. 在事件訂閱窗格中，選擇您要刪除的訂閱。



4. 選擇 刪除。



5. 隨即出現快顯視窗，詢問您是否要永久刪除此通知。選擇 刪除。



Amazon DocumentDB 事件類別和訊息

Amazon DocumentDB 在您可以使用主控台訂閱的類別中產生大量事件。每個類別都適用於來源類型，可以是執行個體、叢集、快照或參數群組。

> **Note**
>
> Amazon DocumentDB 使用現有的 Amazon RDS 事件定義和 IDs。

源自執行個體的 Amazon DocumentDB 事件

類別	描述
可用性	執行個體已重新啟動。
可用性	執行個體關閉。
組態變更	將修改套用至執行個體類別。
組態變更	已完成將修改套用至執行個體類別。
組態變更	重設主要登入資料。
建立	執行個體已建立。
刪除	執行個體已刪除
失敗	由於組態不相容或基礎儲存體問題，執行個體失敗。開始執行個體的point-in-time-restore。

類別	描述
notification	執行個體已停止。
notification	執行個體已啟動。
notification	執行個體正在啟動，因為它超過允許的停止時間上限。
復原	執行個體的復原已開始。復原時間依據恢復的資料量而有不同。
復原	執行個體的復原已完成。
安全性修補程式	作業系統更新可供您的執行個體使用。如需套用更新的相關資訊，請參閱 維護 Amazon DocumentDB。

源自叢集的 Amazon DocumentDB 事件

類別	描述
建立	叢集已建立
刪除	叢集已刪除。
容錯移轉	再次提升先前的主要節點。
容錯移轉	已完成容錯移轉至執行個體。
容錯移轉	已開始容錯移轉至資料庫執行個體：%s
容錯移轉	已啟動相同的 AZ 容錯移轉至資料庫執行個體：%s
容錯移轉	已開始跨可用區容錯移轉至資料庫執行個體：%s
維護	叢集已修補。

類別	描述
維護	資料庫叢集處於無法升級的狀態：%s
notification	叢集已停止。
notification	叢集已啟動。
notification	叢集停止失敗。
notification	叢集正在啟動，因為它超過允許的停止時間上限。
notification	已將叢集從 %s 重新命名為 %s。

源自叢集快照的 Amazon DocumentDB 事件

下表顯示當 Amazon DocumentDB 叢集快照是來源類型時的事件類別和事件清單。

類別	描述
備份	正在建立手動叢集快照。
備份	已建立手動叢集快照。
備份	建立自動叢集快照。
備份	已建立自動叢集快照。

源自參數群組的 Amazon DocumentDB 事件

下表顯示當參數群組為來源類型時的事件類別和事件清單。

類別	描述
組態變更	使用套用方法 %s 將參數 %s 更新至 %s

使用 CloudWatch 監控 Amazon DocumentDB

Amazon DocumentDB（具有 MongoDB 相容性）與 Amazon CloudWatch 整合，因此您可以收集和分析叢集的操作指標。您可以使用 CloudWatch 主控台、Amazon DocumentDB 主控台、 AWS Command Line Interface (AWS CLI) 或 CloudWatch API 來監控這些指標。

CloudWatch 也可讓您設定警示，以便在指標值超過指定閾值時收到通知。您甚至可以設定 Amazon CloudWatch Events 在發生違規時採取修正動作。如需使用 CloudWatch 和警示的詳細資訊，請參閱 Amazon CloudWatch 文件。

主題

- Amazon DocumentDB 指標
- 檢視 CloudWatch 資料
- Amazon DocumentDB 維度
- 監控交易指標
- 監控資料庫連線

Amazon DocumentDB 指標

若要監控 Amazon DocumentDB 叢集和執行個體的運作狀態和效能，您可以在 Amazon DocumentDB 主控台中檢視下列指標。

> **Note**
>
> 下表中的指標適用於執行個體型叢集和彈性叢集。

主題

- 資源使用率指標
- 延遲指標
- NVMe 支援的執行個體指標
- 操作指標
- 命令層級指標
- 輸送量指標
- 系統指標

- T3 執行個體指標

資源使用率指標

指標	說明
BackupRetentionPeriodStorageUsed	用於支援 point-in-time還原功能的備份儲存總量，以位元組為單位。 Amazon DocumentDB 包含在 TotalBackupStorageBilled 指標報告的總數中。針對每個 Amazon DocumentDB 叢集單獨計算。
ChangeStreamLogSize	您的叢集用於存放變更串流日誌的儲存容量 (MB)。此數值是叢集的儲存總量子集 (VolumeBytesUsed)，且會影響叢集費用。如需儲存定價資訊，請參閱 Amazon DocumentDB 產品頁面。變更串流日誌大小是您叢集所進行之變更程度與變更串流長期保留期間的函數。如需變更串流的詳細資訊，請參閱搭配 Amazon DocumentDB 使用變更串流。
CPUUtilization	執行個體所使用 CPU 的百分比。
DatabaseConnections	在執行個體上以 1 分鐘的頻率開啟的連線數（作用中和閒置）。

指標	說明
DatabaseConnections Max	執行個體在 1 分鐘內開啟的資料庫連線數目上限（作用中和閒置）。
DatabaseConnections Limit	執行個體在任何指定時間允許的並行資料庫連線數目上限（作用中和閒置）。
DatabaseConnections ActiveMax	執行個體在 1 分鐘內的作用中資料庫連線數目上限。作用中連線是目前執行 操作的連線，而不是閒置連線。如需每個執行個體類型的作用中連線配額，請參閱 執行個體配額。
DatabaseCursors	在執行個體上以 1 分鐘的頻率開啟的游標數目。
DatabaseCursorsMax	在 1 分鐘內執行個體上開啟的游標數目上限。
DatabaseCursorsLimit	執行個體在任何指定時間允許的游標數目上限。
DatabaseCursorsTim edOut	在 1 分鐘內逾時的游標數量。
FreeableMemory	可用的隨機存取記憶體的數量，以位元組為單位。

指標	說明
FreeLocalStorage	此指標回報可供各執行個體用於暫時資料表與記錄的儲存空間量。此值取決於執行個體類別。您可以為執行個體選擇較大的執行個體，以增加執行個體的可用儲存空間量。（這不適用於 DocumentDB Serverless。）
LowMemThrottleQueueDepth	由於以 1 分鐘頻率取得的可用記憶體不足而調節的請求佇列深度。
LowMemThrottleMaxQueueDepth	在 1 分鐘內因可用記憶體不足而調節的請求的最大佇列深度。
LowMemNumOperationsThrottled	由於 1 分鐘內可用記憶體不足而調節的請求數量。
SnapshotStorageUsed	在備份保留時段外，特定 Amazon DocumentDB 叢集的所有快照所耗用的備份儲存總量，以位元組為單位。包含在 TotalBackupStorageBilled 指標報告的總數中。針對每個 Amazon DocumentDB 叢集單獨計算。
SwapUsage	執行個體上使用的交換空間量。

指標	說明
TotalBackupStorageBilled	針對指定的 Amazon DocumentDB 叢集，以位元組為單位的備份儲存總量。包括以 BackupRetentionPeriodStorageUsed 和 SnapshotStorageUsed 指標衡量的備份儲存。針對每個 Amazon DocumentDB 叢集單獨計算。
TransactionsOpen	在執行個體上以 1 分鐘的頻率開啟的交易數量。
TransactionsOpenMax	執行個體在 1 分鐘內開啟的交易數量上限。
TransactionsOpenLimit	執行個體在任何指定時間允許的並行交易數目上限。
VolumeBytesUsed	您的叢集使用的儲存空間量 (以位元組為單位)。這個值會影響叢集的成本。如需定價資訊，請參閱 Amazon DocumentDB 產品頁面。

延遲指標

指標	說明
DBClusterReplicaLagMaximum	主要執行個體與叢集中每個 Amazon DocumentDB 執行個體之間的延遲量上限，以毫秒為單位。

指標	說明	
DBClusterReplicaLagMinimum	主要執行個體與叢集中每個複寫執行個體之間的最小延遲量 (單位毫秒)。	
DBInstanceReplicaLag	從主要執行個體複寫更新至複寫執行個體時的延遲量 (單位毫秒)。	
ReadLatency	平均每次磁碟輸入/輸出操作耗用的時間量。	
WriteLatency	平均每次磁碟 I/O 操作耗用的時間量 (單位毫秒)。	

NVMe 支援的執行個體指標

指標	說明	
NVMeStorageCacheHitRatio	分層快取提供的請求百分比。	
FreeNVMeStorage	可用的暫時性 NVMe 儲存量。	
ReadIOPSNVMeStorage	暫時性 NVMe 儲存的磁碟讀取 I/O 操作的平均數量。	
ReadLatencyNVMeStorage	暫時性 NVMe 儲存每次磁碟讀取 I/O 操作所花費的平均時間。	
ReadThroughputNVMeStorage	暫時性 NVMe 儲存每秒從磁碟讀取的平均位元組數。	
WriteIOPSNVMeStorage	暫時性 NVMe 儲存的磁碟寫入 I/O 操作的平均數量。	

指標	說明	
WriteLatencyNVMeStorage	暫時性 NVMe 儲存每次磁碟寫入 I/O 操作所花費的平均時間。	
WriteThroughputNVMeStorage	暫時性 NVMe 儲存每秒寫入磁碟的平均位元組數。	

操作指標

指標	說明	
DocumentsDeleted	1 分鐘內已刪除的文件數量。	
DocumentsInserted	1 分鐘內插入的文件數量。	
DocumentsReturned	1 分鐘內傳回的文件數量。	
DocumentsUpdated	1 分鐘內更新的文件數量。	
OpcountersCommand	在 1 分鐘內發出的命令數量。	
OpcountersDelete	在 1 分鐘內發出的刪除操作數量。	
OpcountersGetmore	在 1 分鐘內發行的 getmores 數量。	
OpcountersInsert	在 1 分鐘內發出的插入操作數量。	
OpcountersQuery	在 1 分鐘內發出的查詢數量。	
OpcountersUpdate	在 1 分鐘內發出的更新操作數量。	
TransactionsStarted	在 1 分鐘內在執行個體上啟動的交易數量。	

指標	說明
TransactionsCommitted	在 1 分鐘內對執行個體遞交的交易數量。
TransactionsAborted	執行個體在 1 分鐘內中止的交易數量。
TTLDeletedDocuments	TTLMonitor 在 1 分鐘內刪除的文件數量。

命令層級指標

Amazon DocumentDB 會將每個命令延遲和並行指標發佈至 CloudWatch。這些指標可協助您識別哪些操作對您的工作負載貢獻最大，並偵測尾端延遲問題或並行壓力。

下表中的每個命令會在 1 分鐘內發出三個指標：

- AvgDuration — 命令的平均執行時間，以微秒為單位。
- P100Duration — 命令的最大執行時間（第 100 個百分位數），以微秒為單位。
- MaxConcurrent — 命令的並行執行數目上限。

指標名稱遵循模式 Command.*commandName*.*metricName*。例如 Command.find.AvgDuration 或 Command.aggregate.MaxConcurrent。

指標	說明
Command.find	使用 尋找讀取查詢。
Command.insert	文件插入操作。
Command.findAndModify	原子find-and-modify操作。
Command.update	文件更新操作。
Command.delete	文件刪除操作。

指標	說明
Command.aggregate	彙總管道執行。
Command.count	計數操作。
Command.distinct	不同的值查詢。
Command.getMore	游標批次擷取操作。
Command.abortTransaction	交易中止操作。
Command.commitTransaction	交易遞交操作。

輸送量指標

指標	說明
NetworkReceiveThroughput	叢集中每個執行個體從用戶端接收到的網路傳輸量 (以位元組/秒為單位)。此傳輸量不包含叢集中的執行個體與叢集磁碟區之間的網路流量。
NetworkThroughput	Amazon DocumentDB 叢集中每個執行個體從用戶端接收和傳輸的網路輸送量，以每秒位元組為單位。此傳輸量不包含叢集中的執行個體與叢集磁碟區之間的網路流量。
NetworkTransmitThroughput	叢集中每個執行個體傳送至用戶端的網路傳輸量 (以位元組/秒為單位)。此傳輸量不包含叢

指標	說明
	集中的執行個體與叢集磁碟區之間的網路流量。
ReadIOPS	磁碟讀取輸入/輸出操作的每秒平均次數。Amazon DocumentDB 會以 1 分鐘的間隔分別報告讀取和寫入 IOPS。
ReadThroughput	平均每秒從磁碟讀取的位元組數目。
StorageNetworkReceiveThroughput	叢集中每個執行個體從 Amazon DocumentDB 叢集儲存磁碟區接收的網路輸送量，以每秒位元組為單位。
StorageNetworkTransmitThroughput	叢集中每個執行個體傳送至 Amazon DocumentDB 叢集儲存磁碟區的網路輸送量，以每秒位元組為單位。
StorageNetworkThroughput	Amazon DocumentDB 叢集中每個執行個體接收並傳送至 Amazon DocumentDB 叢集儲存磁碟區的網路輸送量，以每秒位元組為單位。

指標	說明	
VolumeReadIOPs	來自叢集磁碟區的計費讀取 I/O 操作平均數量，每隔 5 分鐘回報一次。計費的讀取操作以叢集磁碟區層級計算，彙整來自資料庫叢集中的所有執行個體，每隔 5 分鐘回報一次。此值以每隔 5 分鐘取得的讀取操作指標值進行計算。您可以藉由取得計費讀取操作指標並除以 300 秒，計算出每秒的計費讀取操作量。 例如，如果 VolumeReadIOPs 傳回 13，686，則每秒計費讀取操作為 45 (13，686 / 300 = 45.62)。 您的查詢提出資料庫頁面請求，該頁面不存在於緩衝區快取，因此必須從儲存空間載入，因而產生計費讀取操作。您可能會在計費讀取操作中發現峰值，那是因為從儲存空間讀取查詢結果，然後載入至緩衝區快取。	

指標	說明	
VolumeWriteIOPs	來自叢集磁碟區的計費寫入 I/O 操作平均數量，每隔 5 分鐘回報一次。計費的寫入操作以叢集磁碟區層級計算，彙整來自資料庫叢集中的所有執行個體，每隔 5 分鐘回報一次。此值以每隔 5 分鐘取得的寫入操作指標值進行計算。您可以藉由取得計費寫入操作指標值並除以 300 秒，計算出每秒的計費寫入操作量。 例如，如果 VolumeWriteIOPs 傳回 13，686，則每秒計費寫入操作為 45 (13，686 / 300 = 45.62)。 請注意， VolumeReadIOPs 和 VolumeWriteIOPs 指標由 DocumentDB 儲存層計算，並包含主要執行個體和複本執行個體執行的 IOs。資料每 20-30 分鐘彙總一次，然後每隔 5 分鐘報告一次，因此在期間內會發出指標的相同資料點。如果您要在 1 分鐘內尋找與插入操作相關聯的指標，您可以使用執行個體層級 WriteIOPS 指標。指標可在 Amazon DocumentDB 主要執行個體的監控索引標籤中使用。	

指標	說明	
WriteIOPS	磁碟寫入輸入/輸出操作的每秒平均次數。在叢集層級上使用時，WriteIOPs 會跨叢集中的所有執行個體進行評估。每隔 1 分鐘分別回報讀取與寫入 IOPS。	
WriteThroughput	平均每秒寫入磁碟的位元組數目。	

系統指標

指標	說明	
AvailableMVCCIds	計數器，顯示達到零之前可用的剩餘寫入操作數目。當此計數器達到零時，您的叢集會進入唯讀模式IDs直到 ID 回收和回收為止。計數器會隨著每次寫入操作而減少，並隨著垃圾回收回收舊的 MVCC IDs 而增加。	
BufferCacheHitRatio	由緩衝區快取提供服務的請求的百分比。	
DiskQueueDepth	等待寫入磁碟或從磁碟讀取的 I/O 操作數目。	
EngineUptime	執行個體已執行的時間量 (單位秒)。	
IndexBufferCacheHitRatio	緩衝區快取提供的索引請求百分比。在捨棄索引、集合或資料庫之後，您可能會看到指標	

指標	說明
	的峰值大於 100%。這將在 60 秒後自動更正。此限制將在未來的修補程式更新中修正。
LongestActiveGCRuntime	最長作用中垃圾回收程序的持續時間，以秒為單位。每分鐘更新一次，並僅追蹤作用中的操作，不包括在一分鐘時段內完成的程序。

T3 執行個體指標

指標	說明
CPUCreditUsage	在測量期間花費的 CPU 點數。
CPUCreditBalance	執行個體累積的 CPU 點數。當 CPU 效能瞬間突增，以及 CPU 額度用掉的速度比獲得的快時，此額度餘額會用完。
CPUSurplusCreditBalance	當 CPUCreditBalance 值為零時，用於維持 CPU 效能的剩餘 CPU 點數。
CPUSurplusCreditsCharged	超過 24 小時內可獲得的 CPU 點數上限的剩餘 CPU 點數，因此會產生額外費用。如需詳細資訊，請參閱監控您的 CPU 點數。

檢視 CloudWatch 資料

您可以使用 Amazon CloudWatch DocumentDB 主控台 AWS Command Line Interface (AWS CLI) 或 CloudWatch API 檢視 Amazon CloudWatch 資料。 Amazon DocumentDB CloudWatch

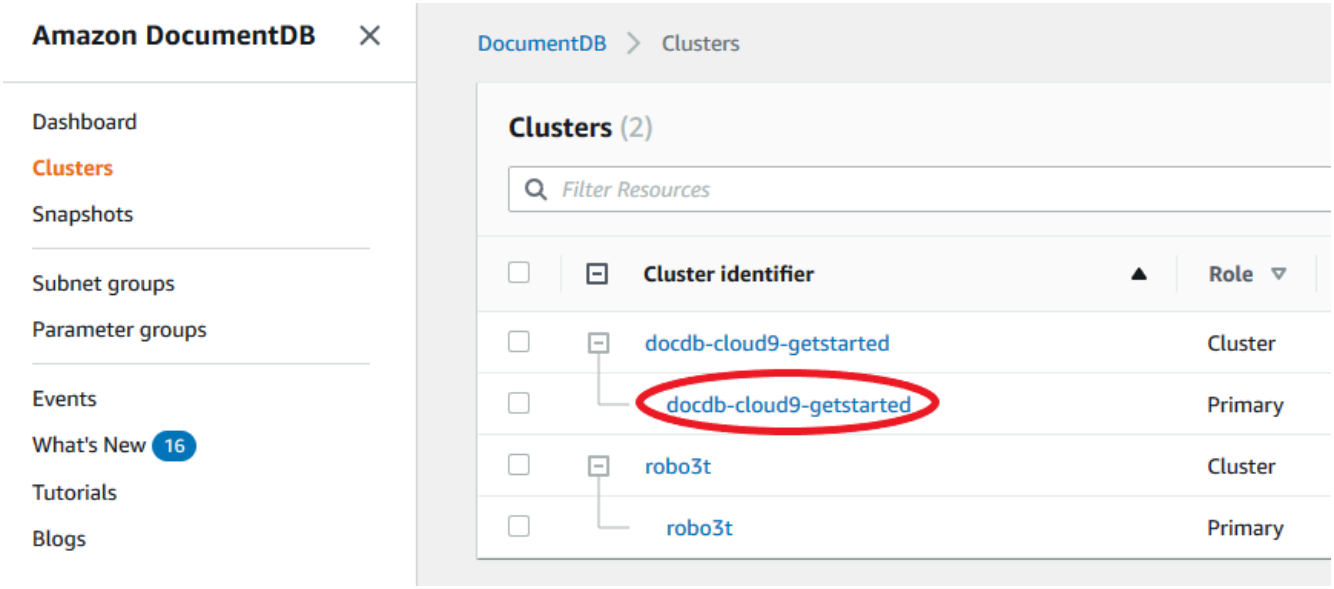
Using the AWS 管理主控台

若要使用 Amazon DocumentDB 管理主控台檢視 CloudWatch 指標，請完成下列步驟。

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

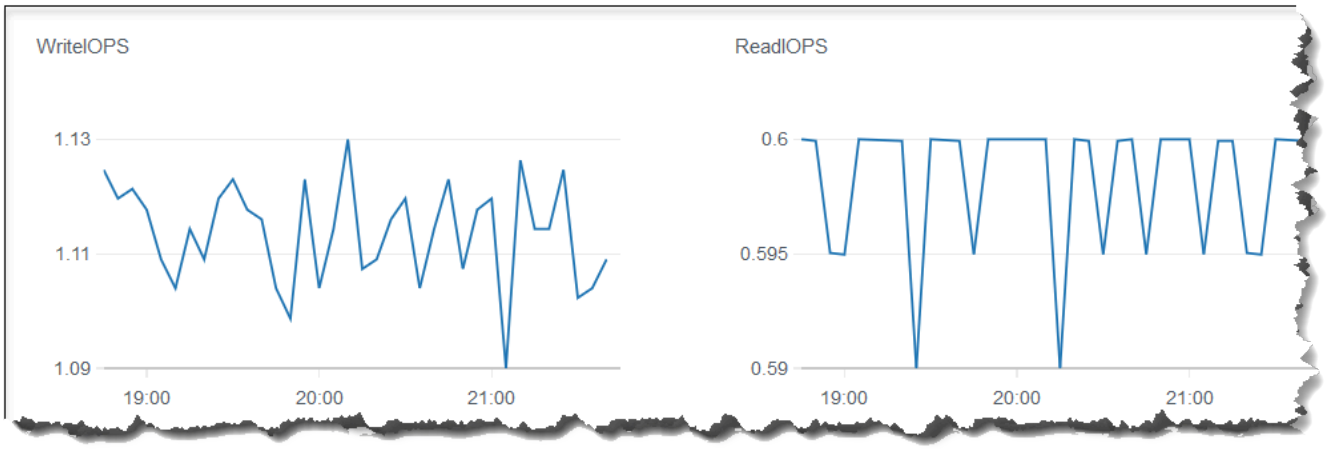
   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

3. 在叢集導覽方塊中，您會看到叢集識別符欄。您的執行個體會列在叢集下，類似於下列螢幕擷取畫面。



4. 從執行個體清單中，選擇您想要指標的執行個體名稱。
5. 在產生的執行個體摘要頁面中，選擇監控索引標籤以檢視 Amazon DocumentDB 執行個體指標的圖形表示。由於每個指標都必須產生圖形，因此可能需要幾分鐘的時間 CloudWatch (CloudWatch) 圖形才會填入。

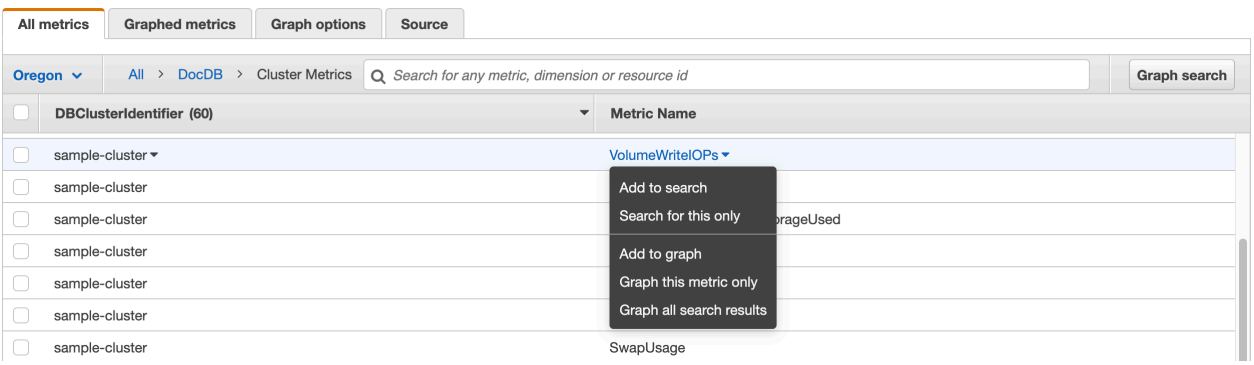
   下圖顯示 Amazon DocumentDB 主控台中兩個 CloudWatch 指標的圖形表示法，`WriteIOPS`以及 `ReadIOPS`。



Using the CloudWatch Management Console

若要使用 CloudWatch 管理主控台檢視 CloudWatch 指標，請完成下列步驟。

1. 登入 AWS 管理主控台，並在 開啟 Amazon DocumentDB 主控台https://console.aws.amazon.com/cloudwatch。
2. 在導覽窗格中，選擇 指標。然後，從服務名稱清單中，選擇 DocDB。
3. 選擇指標維度 （ 例如，叢集指標)。
4. 所有指標索引標籤會顯示 DocDB 中該維度的所有指標。
   a. 若要將資料表排序，請使用直欄標題。
   b. 若要將指標圖形化，請勾選指標旁的核取方塊。若要選擇所有指標，請勾選表格標題列中的核取方塊。
   c. 若要依指標篩選，請將滑鼠游標暫留在指標名稱上，然後選取指標名稱旁的下拉箭頭。然後，選擇新增至搜尋，如下圖所示。



Using the AWS CLI

若要檢視 Amazon DocumentDB 的 CloudWatch 資料，請使用 CloudWatch `get-metric-statistics`操作搭配下列參數。

Parameters

- **`--namespace`** – 必要。您想要 CloudWatch 指標的服務命名空間。對於 Amazon DocumentDB，這必須是 `AWS/DocDB`。
- **`--metric-name`** – 必要。您想要資料之指標的名稱。
- **`--start-time`** – 必要。決定要傳回的第一個資料點的時間戳記。

  包含指定的值；結果包括具有指定時間戳記的資料點。時間戳記必須採用 ISO 8601 UTC 格式 (例如 2016-10-03T23:00:00Z)。
- **`--end-time`** – 必要。決定要傳回的最後一個資料點的時間戳記。

  包含指定的值；結果包括具有指定時間戳記的資料點。時間戳記必須採用 ISO 8601 UTC 格式 (例如 2016-10-03T23:00:00Z)。
- **`--period`** – 必要。所傳回資料點的精細程度 (單位秒)。若是使用一般解析度的指標，期間最短可以是一分鐘 (60 秒) 且必須是 60 的倍數。對於依少於一分鐘的間隔收集的高解析度指標，期間可以是 1、5、10、30、60，或 60 的任何倍數。
- **`--dimensions`** — 選用。如果指標包含多個維度，您必須包含每個維度的值。CloudWatch 會將不同的維度組合視為不同指標。如果未發佈特定維度組合，您就無法擷取其統計資料。您必須指定建立指標時所使用的相同維度。
- **`--statistics`** — 選用。百分位數以外的指標統計資料。如需百分位數統計資料，請使用 `ExtendedStatistics`。呼叫 `GetMetricStatistics` 時，您必須指定 `Statistics` 或 `ExtendedStatistics`，但不能同時指定兩者。

  允許的值：

  - `SampleCount`
  - `Average`
  - `Sum`
  - `Minimum`
  - `Maximum`

- **--extended-statistics** — 選用。percentile 統計資料。指定介於 p0.0 至 p100 之間的值。呼叫 GetMetricStatistics 時，您必須指定 Statistics 或 ExtendedStatistics，但不能同時指定兩者。
- **--unit** — 選用。特定指標的單位。指標可能採用多種單位回報。若未提供單位，則會導致傳回所有單位的結果。如果您唯一指定的單位是指標不會報告的單位，則呼叫結果會是 null。

  可能的值如下：

  - Seconds
  - Microseconds
  - Milliseconds
  - Bytes
  - Kilobytes
  - Megabytes
  - Gigabytes
  - Terabytes
  - Bits
  - Kilobytes
  - Megabits
  - Gigabits
  - Terabits
  - Percent
  - Count
  - Bytes/Second
  - Kilobytes/Second
  - Megabytes/Second
  - Gigabytes/Second
  - Terabytes/Second
  - Bits/Second
  - Kilobits/Second
  - Megabits/Second
  - Gigabits/Second

- Terabits/Second
- Count/Second
- None

Example

以下範例會找出 2 小時期間的最大 CPUUtilization，每 60 秒取樣一次。

針對 Linux、macOS 或 Unix：

```
aws cloudwatch get-metric-statistics \
       --namespace AWS/DocDB \
       --dimensions \
           Name=DBInstanceIdentifier,Value=docdb-2019-01-09-23-55-38 \
       --metric-name CPUUtilization \
       --start-time 2019-02-11T05:00:00Z \
       --end-time 2019-02-11T07:00:00Z \
       --period 60 \
       --statistics Maximum
```

針對 Windows：

```
aws cloudwatch get-metric-statistics ^
       --namespace AWS/DocDB ^
       --dimensions ^
           Name=DBInstanceIdentifier,Value=docdb-2019-01-09-23-55-38 ^
       --metric-name CPUUtilization ^
       --start-time 2019-02-11T05:00:00Z ^
       --end-time 2019-02-11T07:00:00Z ^
       --period 60 ^
       --statistics Maximum
```

此操作的輸出如下所示。

```
{
       "Label": "CPUUtilization",
       "Datapoints": [
           {
               "Unit": "Percent",
               "Maximum": 4.49152542374361,
```

```
            "Timestamp": "2019-02-11T05:51:00Z"
        },
        {
            "Unit": "Percent",
            "Maximum": 4.25000000000485,
            "Timestamp": "2019-02-11T06:44:00Z"
        },

        ********* some output omitted for brevity *********

        {
            "Unit": "Percent",
            "Maximum": 4.33333333331878,
            "Timestamp": "2019-02-11T06:07:00Z"
        }
    ]
}
```

Amazon DocumentDB 維度

Amazon DocumentDB 的指標取決於帳戶或操作的值。您可以使用 CloudWatch 主控台來擷取由下表中任一維度篩選的 Amazon DocumentDB 資料。

維度	說明
DBClusterIdentifier	篩選您為特定 Amazon DocumentDB 叢集請求的資料。
DBClusterIdentifier, Role	篩選您為特定 Amazon DocumentDB 叢集請求的資料，依執行個體角色 (WRITER/READER) 彙總指標。例如，您可以為屬於某個叢集的所有讀取者執行個體彙總指標。
DBInstanceIdentifier	篩選您對特定資料庫執行個體請求的資料。

監控交易指標

對閒置叢集的反向指標具有非零值 （通常為 ~50)。這是因為 Amazon DocumentDB 會執行定期運作狀態檢查、內部操作和指標收集任務。

監控資料庫連線

當您使用 等資料庫引擎命令來檢視連線數時`db.runCommand( { serverStatus: 1 })`，您最多可能會看到比`DatabaseConnections`透過 CloudWatch 在 中看到的連線多 10 個。這是因為 Amazon DocumentDB 會執行定期運作狀態檢查，以及未在 中說明的指標收集任務`DatabaseConnections`。 僅`DatabaseConnections`代表客戶起始的連線。

使用 記錄 Amazon DocumentDB API 呼叫 AWS CloudTrail

Amazon DocumentDB （具有 MongoDB 相容性） 已與 服務整合 AWS CloudTrail，此服務可提供 Amazon DocumentDB （具有 MongoDB 相容性） 中使用者、角色或 AWS 服務所採取之動作的記錄。CloudTrail 會將 Amazon DocumentDB 的所有 AWS CLI API 呼叫擷取為事件，包括來自 Amazon DocumentDB 主控台的呼叫，以及來自對 Amazon DocumentDB SDK 的程式碼呼叫。如果您建立線索，則可以將 CloudTrail 事件持續交付至 Amazon S3 儲存貯體，包括 Amazon DocumentDB 的事件。即使您未設定追蹤，依然可以透過 CloudTrail 主控台中的 Event history (事件歷史記錄) 檢視最新事件。您可以使用 CloudTrail 所收集的資訊，判斷對 Amazon DocumentDB 提出的請求 （與 MongoDB 相容）、提出請求的 IP 地址、提出請求的人員、提出請求的時間，以及其他詳細資訊。

> **Important**
>
> 對於某些管理功能，Amazon DocumentDB 使用與 Amazon Relational Database Service (Amazon RDS) 共享的操作技術。Amazon DocumentDB 主控台 AWS CLI和 API 呼叫會記錄為對 Amazon RDS API 發出的呼叫。

若要進一步了解 AWS CloudTrail，請參閱 AWS CloudTrail 使用者指南。

CloudTrail 中的 Amazon DocumentDB 資訊

當您建立帳戶 AWS 帳戶 時，您的 上會啟用 CloudTrail。當活動在 Amazon DocumentDB （與 MongoDB 相容） 中發生時，該活動會記錄在 CloudTrail 事件中，以及事件歷史記錄中的其他 AWS 服務事件。您可以在 中檢視、搜尋和下載最近的事件 AWS 帳戶。如需詳細資訊，請參閱《使用 CloudTrail 事件歷史記錄檢視事件》https://docs.aws.amazon.com/awscloudtrail/latest/userguide/view-cloudtrail-events.html。

若要持續記錄 中的事件 AWS 帳戶，包括 Amazon DocumentDB 的事件 （具有 MongoDB 相容性），請建立追蹤。線索能讓 CloudTrail 將日誌檔案交付至 Amazon S3 儲存貯體。依預設，當您在主控台中建立追蹤時，該追蹤會套用至所有的 AWS 區域。該追蹤會記錄來自 AWS 分割區中所有區域的事件，

並將日誌檔案交付到您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中收集的事件資料。如需詳細資訊，請參閱 AWS CloudTrail 使用者指南中的以下主題：

- 建立追蹤的概觀
- CloudTrail 支援的服務和整合
- 設定 CloudTrail 的 Amazon SNS 通知
- 從多個區域接收 CloudTrail 日誌檔案
- 從多個帳戶接收 CloudTrail 日誌檔案

每一筆事件或記錄項目都會包括產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 該請求是否使用根或使用者憑證提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 請求是否由其他 AWS 服務提出。

如需詳細資訊，請參閱 CloudTrail userIdentity 元素。

分析 Amazon DocumentDB 操作

您可以使用 Amazon DocumentDB 中的分析器（與 MongoDB 相容）來記錄叢集上執行的操作的執行時間和詳細資訊。Profiler 適用於監控叢集上最慢的操作，以協助您改善個別查詢效能和整體叢集效能。

根據預設會停用 Profiler 功能。啟用時，分析器會將花費超過客戶定義閾值（例如 100 毫秒）的操作記錄到 Amazon CloudWatch Logs。記錄的詳細資訊包括已分析的命令、時間、計劃摘要和用戶端中繼資料。操作記錄到 CloudWatch Logs 之後，您可以使用 CloudWatch Logs Insights 來分析、監控和封存 Amazon DocumentDB 分析資料。常見查詢小節提供常用的查詢。

啟用時，Profiler 會使用叢集的額外資源。從高閾值（例如 500 毫秒）開始，並逐漸降低該值以識別慢速操作。對於高輸送量應用程式，從閾值 50 毫秒開始可能會在叢集上造成效能問題。分析器會在叢集層級啟用，並在叢集中的所有執行個體和資料庫上運作。Amazon DocumentDB 會盡力將操作記錄到 Amazon CloudWatch Logs。

雖然 Amazon DocumentDB 不收取額外費用來啟用分析器，但您需要支付使用 CloudWatch Logs 的標準費率。如需 CloudWatch Logs 定價的資訊，請參閱 Amazon CloudWatch 定價。

主題

- 受支援的操作
- 限制
- 啟用 Amazon DocumentDB 分析器
- 停用 Amazon DocumentDB 分析器
- 停用分析器日誌匯出
- 存取 Amazon DocumentDB 分析器日誌
- 常見查詢

受支援的操作

Amazon DocumentDB Profiler 支援下列操作：

- `aggregate`
- `count`
- `delete`
- `distinct`
- `find` (OP_QUERY 和命令)
- `findAndModify`
- `insert`
- `update`

限制

慢查詢分析器只有在查詢的整個結果集能夠容納在一個批次中，且結果集低於 16MB （最大 BSON 大小） 時，才能發出分析器日誌。大於 16MB 的結果集會自動分割成多個批次。

大多數驅動程式或 Shell 可能會設定較小的預設批次大小。您可以在查詢中指定批次大小。若要擷取慢查詢日誌，請使用超過預期結果集大小的批次大小。如果您不確定結果集大小，或是大小不同，請將批次大小設定為較大的數字 （例如 100k)。

不過，使用較大的批次大小表示在將回應傳送到用戶端之前，必須從資料庫擷取更多結果。對於某些查詢，這可能會在取得結果之前產生較長的延遲。如果您不打算使用整個結果集，則可能會花費更多 I/O 來處理查詢並捨棄結果。

啟用 Amazon DocumentDB 分析器

在叢集上啟用 Profiler 包含三個步驟。確保所有步驟都已完成，否則分析日誌將不會傳送至 CloudWatch Logs。Profiler 是在叢集層級設定，並在叢集的所有資料庫和執行個體上執行。

在叢集上啟用 Profiler

1. 由於您無法修改預設叢集參數群組，請確保您有可用的自訂叢集參數群組。如需詳細資訊，請參閱建立 Amazon DocumentDB 叢集參數群組。
2. 使用可用的自訂叢集參數群組，修改下列參數：`profiler`、`profiler_threshold_ms` 和 `profiler_sampling_rate`。如需詳細資訊，請參閱修改 Amazon DocumentDB 叢集參數群組。
3. 建立或修改叢集以使用自訂叢集參數群組，並啟用將`profiler`日誌匯出至 CloudWatch Logs。

下列各節說明如何使用 AWS 管理主控台 和 AWS Command Line Interface () 實作這些步驟AWS CLI。

Using the AWS 管理主控台

1. 開始之前，如果您還沒有 Amazon DocumentDB 叢集和自訂叢集參數群組，請建立叢集參數群組。如需詳細資訊，請參閱建立 Amazon DocumentDB 叢集參數群組及建立 Amazon DocumentDB 叢集。
2. 使用可用的自訂叢集參數群組，修改下列參數。如需詳細資訊，請參閱修改 Amazon DocumentDB 叢集參數群組。

   - `profiler` — 啟用或停用查詢分析。允許值為 `enabled` 和 `disabled`。預設值為 `disabled`。若要啟用分析，請將值設定為 `enabled`。
   - `profiler_threshold_ms` — 當 `profiler` 設定為 時`enabled`，所有花費超過 的命令 `profiler_threshold_ms`都會記錄到 CloudWatch。允許值為 `[50-INT_MAX]`。預設值為 `100`。
   - `profiler_sampling_rate` — 應分析或記錄的慢速操作部分。允許值為 `[0.0-1.0]`。預設值為 `1.0`。
3. 修改您的叢集以使用自訂叢集參數群組，並將分析器日誌匯出設定為發佈至 Amazon CloudWatch。

   a. 在導覽窗格中，選擇 Clusters (叢集) 來將自訂參數群組新增至叢集。

b. 選擇參數群組要與其建立關聯之叢集名稱左側的按鈕。選取 Actions (動作)，然後選取 Modify (修改) 以修改您的叢集。

c. 在 Cluster (叢集) 選項下，從上述步驟中選擇自訂參數群組，以將其新增至您的叢集。

d. 在日誌匯出下，選取要發佈至 Amazon CloudWatch 的分析器日誌。

e. 選擇 Continue (繼續) 以檢視修改摘要。

f. 確認變更後，即可立即套用，或是在下一個維護時段在 Scheduling of modifications (修改的排程) 下套用。

g. 選擇 Modify cluster (修改叢集) 來以新的參數群組更新叢集。

Using the AWS CLI

下列程序會在叢集 `sample-cluster` 的所有受支援操作上啟用 Profiler。

1. 在開始之前，請執行下列命令，並檢閱名稱中沒有 `default` 且具有 `docdb3.6` 作為參數群組系列之叢集參數群組的輸出，以確定您擁有可用的自訂叢集參數群組。如果您沒有非預設的叢集參數群組，請參閱建立 Amazon DocumentDB 叢集參數群組。

```
aws docdb describe-db-cluster-parameter-groups \
    --query 'DBClusterParameterGroups[*].
[DBClusterParameterGroupName,DBParameterGroupFamily]'
```

在下列輸出中，僅 `sample-parameter-group` 符合這兩個條件。

```
[
       [
           "default.docdb3.6",
           "docdb3.6"
       ],
       [
           "sample-parameter-group",
           "docdb3.6"
       ]
]
```

2. 使用您的自訂叢集參數群組，修改下列參數：

- `profiler` — 啟用或停用查詢分析。允許值為 `enabled` 和 `disabled`。預設值為 `disabled`。若要啟用分析，請將值設定為 `enabled`。

- profiler_threshold_ms — 當 profiler 設為 時enabled，所有花費超過 的命令profiler_threshold_ms都會記錄到 CloudWatch。允許值為 [50-INT_MAX]。預設值為 100。
- profiler_sampling_rate — 應分析或記錄的慢速操作部分。允許值為 [0.0-1.0]。預設值為 1.0。

```
aws docdb modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name sample-parameter-group \
    --parameters
 ParameterName=profiler,ParameterValue=enabled,ApplyMethod=immediate \

 ParameterName=profiler_threshold_ms,ParameterValue=100,ApplyMethod=immediate \

 ParameterName=profiler_sampling_rate,ParameterValue=0.5,ApplyMethod=immediate
```

3. 修改 Amazon DocumentDB 叢集，使其使用上一個步驟的sample-parameter-group自訂叢集參數群組，並將 參數--enable-cloudwatch-logs-exports設定為 profiler。

   下列程式碼會將叢集修改sample-cluster為使用上一個步驟sample-parameter-group的 ，並新增至已啟用profiler的 CloudWatch Logs 匯出。

```
aws docdb modify-db-cluster \
       --db-cluster-identifier sample-cluster \
       --db-cluster-parameter-group-name sample-parameter-group \
       --cloudwatch-logs-export-configuration '{"EnableLogTypes":["profiler"]}'
```

   此操作的輸出將會如下所示。

```
{
    "DBCluster": {
        "AvailabilityZones": [
            "us-east-1c",
            "us-east-1b",
            "us-east-1a"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "sample-cluster",
        "DBClusterParameterGroup": "sample-parameter-group",
        "DBSubnetGroup": "default",
```

```
        "Status": "available",
        "EarliestRestorableTime": "2020-04-07T02:05:12.479Z",
        "Endpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "ReaderEndpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "MultiAZ": false,
        "Engine": "docdb",
        "EngineVersion": "3.6.0",
        "LatestRestorableTime": "2020-04-08T22:08:59.317Z",
        "Port": 27017,
        "MasterUsername": "test",
        "PreferredBackupWindow": "02:00-02:30",
        "PreferredMaintenanceWindow": "tue:09:50-tue:10:20",
        "DBClusterMembers": [
            {
                "DBInstanceIdentifier": "sample-instance-1",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            },
            {
                "DBInstanceIdentifier": "sample-instance-2",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            }
        ],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcd0123",
                "Status": "active"
            }
        ],
        "HostedZoneId": "ABCDEFGHIJKLM",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DbClusterResourceId": "cluster-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "DBClusterArn": "arn:aws:rds:us-east-1:<accountID>:cluster:sample-
cluster",
        "AssociatedRoles": [],
        "ClusterCreateTime": "2020-01-10T22:13:38.261Z",
        "EnabledCloudwatchLogsExports": [
            "profiler"
        ],
        "DeletionProtection": true
```

```
    }
}
```

停用 Amazon DocumentDB 分析器

若要停用分析器，請同時停用 `profiler` 參數和將`profiler`日誌匯出至 CloudWatch Logs。

停用 Profiler

您可以使用 AWS 管理主控台 或 停用 `profiler` 參數 AWS CLI，如下所示。

Using the AWS 管理主控台

下列程序使用 AWS 管理主控台 來停用 Amazon DocumentDB `profiler`。

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。然後選擇您要停用 Profiler 的叢集參數群組的名稱。
3. 在產生的 Cluster parameters (叢集參數) 頁面中，選取 `profiler` 參數左邊的按鈕，然後選擇 Edit (編輯)。
4. 在 Modify profiler (修改 Profiler) 對話方塊中，選擇清單中的 `disabled`。
5. 選擇 Modify cluster parameter (修改叢集參數)。

Using the AWS CLI

若要使用 AWS CLI在叢集上停用 `profiler`，請修改叢集，如下所示。

```
aws docdb modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name sample-parameter-group \
    --parameters
 ParameterName=profiler,ParameterValue=disabled,ApplyMethod=immediate
```

停用分析器日誌匯出

您可以使用 AWS 管理主控台 或 來停用將`profiler`日誌匯出至 CloudWatch Logs AWS CLI，如下所示。

Using the AWS 管理主控台

下列程序使用 AWS 管理主控台 來停用 Amazon DocumentDB 匯出日誌至 CloudWatch。

1. 在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。對於您要停止匯出日誌的叢集，選擇其名稱左側的按鈕。
3. 在 Actions (動作) 功能表上，選擇 Modify (修改)。
4. 向下捲動至 Log exports (日誌匯出) 區段，並取消選取 Profiler logs (分析工具日記)。
5. 選擇繼續。
6. 檢閱您的變更，然後選擇希望叢集套用此變更的時機：
   - Apply during the next scheduled maintenance window (在下次排定的維護時段套用)
   - Apply immediately (立即套用)
7. 選擇修改叢集。

Using the AWS CLI

下列程式碼會修改叢集`sample-cluster` ，並停用 CloudWatch Profiler 日誌。

Example

針對 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster \
   --db-cluster-identifier sample-cluster \
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["profiler"]}'
```

針對 Windows：

```
aws docdb modify-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["profiler"]}'
```

此操作的輸出將會如下所示。

```
{
    "DBCluster": {
        "AvailabilityZones": [
            "us-east-1c",
            "us-east-1b",
```

```
            "us-east-1a"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "sample-cluster",
        "DBClusterParameterGroup": "sample-parameter-group",
        "DBSubnetGroup": "default",
        "Status": "available",
        "EarliestRestorableTime": "2020-04-08T02:05:17.266Z",
        "Endpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "ReaderEndpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "MultiAZ": false,
        "Engine": "docdb",
        "EngineVersion": "3.6.0",
        "LatestRestorableTime": "2020-04-09T05:14:44.356Z",
        "Port": 27017,
        "MasterUsername": "test",
        "PreferredBackupWindow": "02:00-02:30",
        "PreferredMaintenanceWindow": "tue:09:50-tue:10:20",
        "DBClusterMembers": [
            {
                "DBInstanceIdentifier": "sample-instance-1",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            },
            {
                "DBInstanceIdentifier": "sample-instance-2",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            }
        ],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcd0123",
                "Status": "active"
            }
        ],
        "HostedZoneId": "ABCDEFGHIJKLM",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DbClusterResourceId": "cluster-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "DBClusterArn": "arn:aws:rds:us-east-1:<accountID>:cluster:sample-cluster",
        "AssociatedRoles": [],
```

```
        "ClusterCreateTime": "2020-01-10T22:13:38.261Z",
        "DeletionProtection": true
    }
}
```

存取 Amazon DocumentDB 分析器日誌

請依照下列步驟，在 Amazon CloudWatch 上存取您的設定檔日誌。

1. 透過 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 請確定您位於與 Amazon DocumentDB 叢集相同的區域。
3. 在導覽窗格中，選擇日誌。
4. 若要尋找叢集的分析工具日誌，請在清單中選擇 `/aws/docdb/yourClusterName/profiler`。

   您可在每個執行個體名稱下分別取得每個執行個體的描述檔日誌。

常見查詢

以下是一些常用的查詢，可用來分析您已側錄的命令。如需 CloudWatch Logs Insights 的詳細資訊，請參閱使用 CloudWatch Logs Insights 分析日誌資料和範例查詢。

取得指定集合上 10 個最慢的操作

```
filter ns="test.foo" | sort millis desc | limit 10
```

取得花費超過 60 毫秒集合的所有更新操作

```
filter millis > 60 and op = "update"
```

取得上個月 10 個最慢的操作

```
sort millis desc | limit 10
```

使用 COLLSCAN 計劃摘要取得所有查詢

```
filter planSummary="COLLSCAN"
```

使用績效詳情進行監控

Performance Insights 會新增現有的 Amazon DocumentDB 監控功能，以說明您的叢集效能，並協助您分析影響叢集效能的任何問題。使用績效詳情儀表板，您可以視覺化資料庫負載，並依等待、查詢陳述式、主機或應用程式篩選負載。

> **Note**
>
> Performance Insights 僅適用於 Amazon DocumentDB 3.6、4.0、5.0 和 8.0 執行個體型叢集。

它如何有用？

- 視覺化資料庫效能 — 視覺化負載，以判斷負載在資料庫上的時間和位置
- 判斷造成資料庫負載的原因 — 判斷哪些查詢、主機和應用程式造成執行個體負載
- 判斷資料庫何時載入 - 放大績效詳情儀表板以專注於特定事件，或縮小以查看較大時間範圍的趨勢
- 資料庫負載提醒 — 從 CloudWatch 自動存取新的資料庫負載指標，您可以在其中監控資料庫負載指標和其他 Amazon DocumentDB 指標，並設定提醒

Amazon DocumentDB Performance Insights 有哪些限制？

- 無法使用 AWS GovCloud （美國東部） 和 AWS GovCloud （美國西部） 區域中的績效詳情
- Amazon DocumentDB 的績效詳情會保留最多 7 天的效能資料
- 績效詳情中不會彙總超過 1，024 個位元組的查詢

主題

- 績效詳情概念
- 啟用和停用績效詳情
- 設定績效詳情的存取政策

- 使用績效詳情儀表板來分析指標
- 使用績效詳情 API 來擷取指標
- Performance Insights 的 Amazon CloudWatch 指標
- 計數器指標的績效詳情

績效詳情概念

主題

- 平均作用中工作階段
- 維度
- 最大 vCPU

平均作用中工作階段

資料庫負載 (DB 負載) 測量資料庫中的活動程度。績效詳情中的關鍵指標為 DB Load，每秒收集一次。DBLoad 指標的單位是 Amazon DocumentDB 執行個體的平均作用中工作階段 (AAS)。

作用中工作階段是已將工作提交至 Amazon DocumentDB 執行個體並等待回應的連線。例如，如果您將查詢提交至 Amazon DocumentDB 執行個體，資料庫工作階段會在執行個體處理查詢時處於作用中狀態。

若要取得平均作用中工作階段 (AAS)，績效詳情會取樣同時執行查詢的工作階段數目。AAS 是工作階段總數除以樣本總數。下表顯示執行中查詢的五個連續範例。

樣本	執行查詢的工作階段數目	AAS	算式
1	2	2	2 個工作階段 / 1 個樣本
2	0	1	2 個工作階段 / 2 個樣本
3	4	2	6 個工作階段 / 3 個樣本

樣本	執行查詢的工作階段數目	AAS	算式
4	0	1.5	6 個工作階段 / 4 個樣本
5	4	2	10 個工作階段 / 5 個樣本

在上述範例中，1-5 時間間隔的資料庫負載為 2 個 AAS。資料庫負載的增加意味著，就平均而言，更多的工作階段正執行於資料庫上。

維度

DB Load 指標與其他時間序列指標不同，因為您可以將它分為名為維度的子元件。您可以將維度視為 DB Load 指標的不同特性類別。當您診斷效能問題時，最有用的維度是等待狀態和熱門查詢。

等待狀態

等待狀態會導致查詢陳述式等待特定事件發生，然後才能繼續執行。例如，查詢陳述式可能會等到鎖定的資源解除鎖定。透過DB Load結合等待狀態，您可以取得工作階段狀態的完整資訊。以下是各種 Amazon DocumentDB 等待狀態：

Amazon DocumentDB 等待狀態	等待狀態描述
鎖定	當工作階段正在等待分頁緩衝集區時，會發生鎖定等待狀態。當系統經常處理大型查詢、集合掃描，或緩衝集區太小而無法處理工作集時，經常分頁和分頁緩衝集區。
CPU	當工作階段在 CPU 上等待時，會發生 CPU 等待狀態。
CollectionLock	當工作階段正在等待取得集合的鎖定時，會發生 CollectionLock 等待狀態。當集合上有 DDL 操作時，就會發生這些事件。
DocumentLock	當工作階段正在等待取得文件鎖定時，會發生 DocumentLock 等待狀態。對相同文件進行大量

Amazon DocumentDB 等待狀態	等待狀態描述
	並行寫入會導致該文件的 DocumentLock 等待狀態變多。
SystemLock	當工作階段在系統上等待時，會發生 SystemLock 等待狀態。當系統經常出現長時間執行的查詢、長時間執行的交易或高並行時，就會發生這種情況。
IO	當工作階段等待 IO 完成時，會發生 IO 等待狀態。
BufferLock	當工作階段正在緩衝區中的共用頁面上等待取得鎖定時，BufferLock 等待狀態就會發生。如果其他程序在請求的頁面上保留開啟游標，BufferLock 等待狀態可能會延長。
LowMemThrottle	當工作階段因 Amazon DocumentDB 執行個體上的記憶體壓力過大而等待時，會發生 LowMemThrottle 等待狀態。如果此狀態持續很長時間，請考慮擴展執行個體以提供額外的記憶體。如需詳細資訊，請參閱 Resource Governor。
BackgroundActivity	當工作階段正在等待內部系統程序時，會發生 BackgroundActivity 等待狀態。
其他	其他等待狀態是內部等待狀態。如果此狀態持續很長時間，請考慮終止此查詢。如需詳細資訊，請參閱如何尋找和終止長時間執行或封鎖的查詢？

熱門查詢

等待狀態會顯示瓶頸，而熱門查詢會顯示哪些查詢對資料庫負載的影響最大。例如，許多查詢目前可能正在資料庫上執行，但單一查詢可能會耗用 99% 的資料庫負載。在此情況下，高負載可能表示查詢發生問題。

最大 vCPU

在儀表板中，資料庫負載圖表會收集、彙整並顯示工作階段資訊。若要查看作用中的工作階段是否超過最大 CPU，請查看它與最大 vCPU 數線的關係。最大 vCPU 值取決於 Amazon DocumentDB 執行個體的 vCPU （虛擬 CPU) 核心數量。

若資料庫負載通常高於最大 vCPU 數線，而主要等待狀態為 CPU，則 CPU 會超過負載。在這種情況下，您可能想要調節執行個體的連線、調整具有高 CPU 負載的任何查詢，或考慮較大的執行個體類別。處於任何等待狀態的密集且穩定的執行個體表示可能有您應解決的瓶頸或資源爭用問題。即使資料庫負載未超過最大 vCPU 數線，仍可能會有上述的問題。

啟用和停用績效詳情

若要使用績效詳情，請在資料庫執行個體上啟用。您可以視需要稍後停用它。啟用和停用績效詳情不會造成停機、重新開機或容錯移轉。

績效詳情代理程式會耗用資料庫主機上有限的 CPU 和記憶體。當資料庫負載過高時，代理程式會減少收集資料的頻率來限制效能影響。

建立叢集時啟用績效詳情

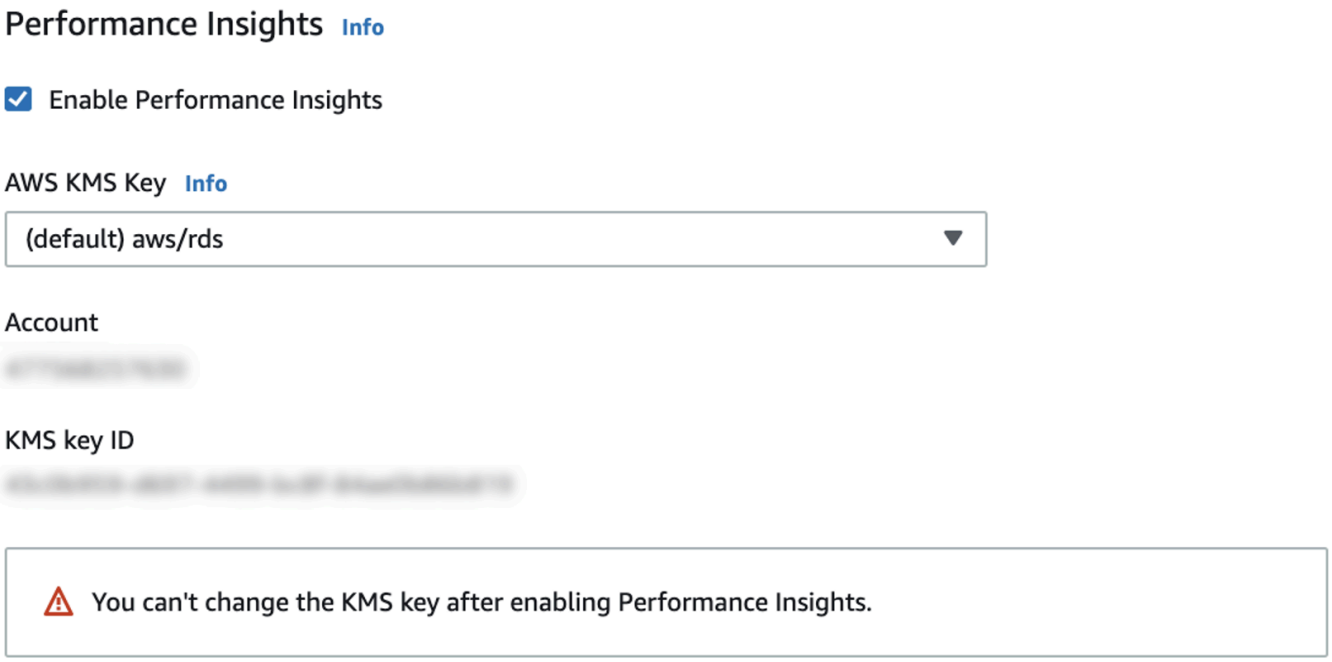
在主控台中，您可以在建立或修改新的資料庫執行個體時啟用或停用績效詳情。

使用 AWS 管理主控台

在 主控台中，您可以在建立 Amazon DocumentDB 叢集時啟用績效詳情。當您建立新的 Amazon DocumentDB 叢集時，請在績效詳情區段中選擇啟用績效詳情來啟用績效詳情。

主控台指示

1. 若要建立叢集，請遵循建立 Amazon DocumentDB 叢集的指示。
2. 在績效詳情區段中選取啟用績效詳情。



> **Note**
>
> Performance Insights 資料保留期間為七天。

AWS KMS key — 指定您的 AWS KMS 金鑰。Performance Insights 會使用 AWS KMS 金鑰加密所有潛在的敏感資料。將會對傳輸中與靜態資料進行加密。如需詳細資訊，請參閱設定績效詳情 AWS AWS KMS 的政策。

修改執行個體時啟用和停用

您可以使用 主控台或 修改資料庫執行個體來啟用或停用績效詳情 AWS CLI。

Using the AWS 管理主控台

主控台指示

1. 登入 AWS 管理主控台，並在 https://console.aws.amazon.com/docdb：// 開啟 Amazon DocumentDB 主控台。
2. 選擇 Clusters (叢集)。
3. 選擇資料庫執行個體，然後選擇 修改 (Modify)。

4. 在績效詳情區段中，選擇啟用績效詳情或停用績效詳情。

> **Note**
>
> 如果您選擇啟用績效詳情，您可以指定您的 AWS AWS KMS 金鑰。Performance Insights 會使用 AWS KMS 金鑰加密所有潛在的敏感資料。將會對傳輸中與靜態資料進行加密。如需詳細資訊，請參閱加密靜態 Amazon DocumentDB 資料。

5. 選擇 Continue (繼續)。
6. 在 Scheduling of Modifications (修改排程) 中，選擇 Apply immediately (立即套用)。如果您在下一個排定的維護時段選擇套用，您的執行個體會忽略此設定，並立即啟用績效詳情。
7. 選擇 Modify instance (修改執行個體)。

Using the AWS CLI

使用 `create-db-instance`或 `modify-db-instance` AWS AWS CLI 命令時，您可以透過指定 來啟用績效詳情`--enable-performance-insights`，或透過指定 來停用它`--no-enable-performance-insights`。

下列程序說明如何使用 AWS AWS CLI啟用或停用資料庫執行個體的績效詳情。

AWS AWS CLI 指示

呼叫 `modify-db-instance` AWS AWS CLI 命令並提供下列值：

- `--db-instance-identifer` — 資料庫執行個體的名稱
- `--enable-performance-insights` 以啟用或 `--no-enable-performance-insights` 以停用

Example

下列範例會啟用 的績效詳情`sample-db-instance`：

For Linux, macOS, or Unix:

```
aws docdb modify-db-instance \
    --db-instance-identifier sample-db-instance \
```

```
    --enable-performance-insights
```

For Windows:

```
aws docdb modify-db-instance ^
    --db-instance-identifier sample-db-instance ^
    --enable-performance-insights
```

設定績效詳情的存取政策

若要存取績效詳情，您必須具有來自 AWS Identity and Access Management (IAM) 的適用許可。您可以使用下列選項授予存取權：

- 將 `AmazonRDSPerformanceInsightsReadOnly` 受管政策連接到許可集或角色。
- 建立自訂 IAM 政策，並將其連接至許可集或角色。

此外，如果您在開啟績效詳情時指定客戶受管金鑰，請確保您帳戶中的使用者具備 KMS 金鑰的 `kms:Decrypt` 和 `kms:GenerateDataKey` 許可。

> **Note**
>
> 對於使用 AWS KMS 金鑰和安全群組管理encryption-at-rest，Amazon DocumentDB 會利用與 Amazon RDS 共用的操作技術。

將 AmazonRDSPerformanceInsightsReadOnly 政策連接至 IAM 主體

`AmazonRDSPerformanceInsightsReadOnly` 是一種 AWS受管政策，可授予 Amazon DocumentDB Performance Insights API 所有唯讀操作的存取權。目前，此 API 中的所有操作都是唯讀操作。若您將 `AmazonRDSPerformanceInsightsReadOnly` 連接至許可集或角色，收件人可使用績效詳情搭配其他主控台功能。

建立績效詳情的自訂 IAM 政策

對於沒有 `AmazonRDSPerformanceInsightsReadOnly` 政策提供的使用者，您可以建立或修改由使用者管理的 IAM 政策來授予績效詳情的存取權。當您將政策連接到許可集或角色時，收件人可以使用績效詳情。

建立自訂政策

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在導覽窗格中，選擇政策。
3. 選擇 Create policy (建立政策)。
4. 在 Create Policy (建立政策) 頁面上，選擇 JSON 標籤。
5. 複製並貼上下列文字，將 *us-east-1* 取代為您的 AWS 區域名稱，並將 *111122223333* 取代為您的客戶帳戶號碼。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rds:DescribeDBInstances",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "rds:DescribeDBClusters",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:DescribeDimensionKeys",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:GetDimensionKeyDetails",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:GetResourceMetadata",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": "pi:GetResourceMetrics",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:ListAvailableResourceDimensions",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:ListAvailableResourceMetrics",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        }
    ]
}
```

6. 選擇 Review policy (檢閱政策)。
7. 為政策提供名稱並選擇性輸入描述，然後選擇 Create policy (建立政策)。

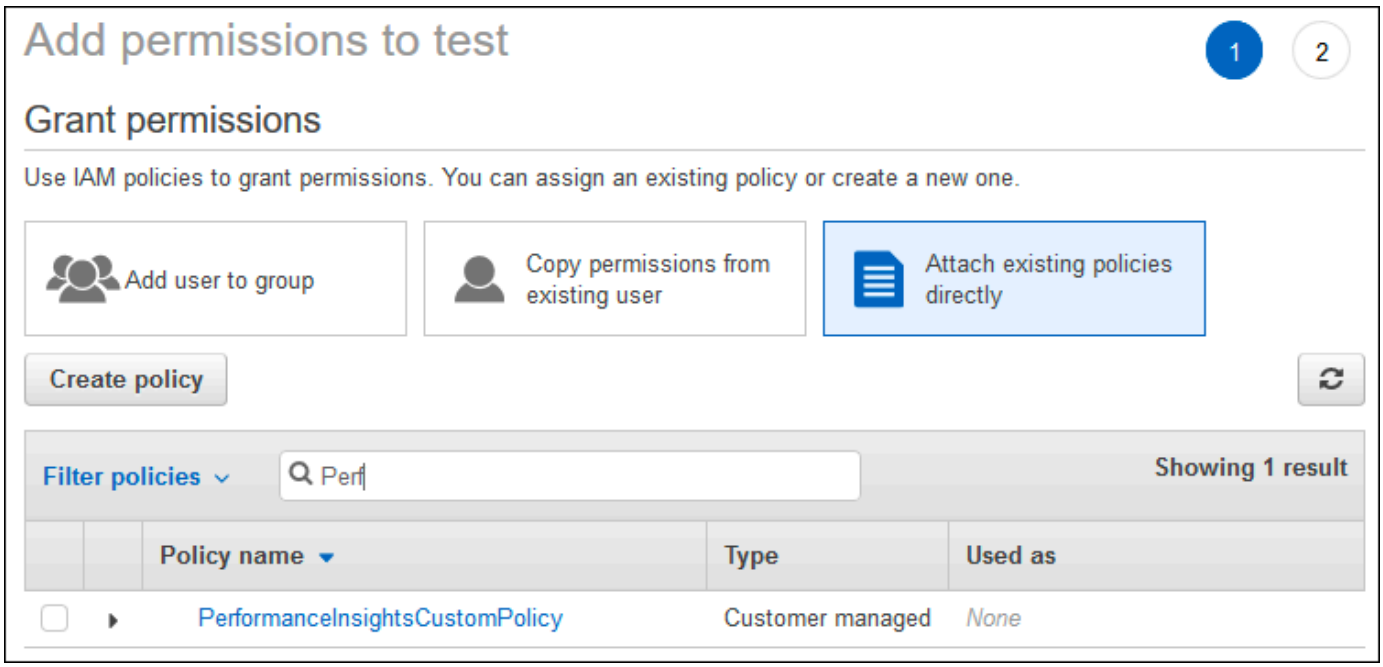
現在您可以將政策連接到許可集或角色。以下程序假設您已有基於此用途使用的使用者。

將政策連接至使用者

1. 在以下網址開啟 IAM 主控台：https://console.aws.amazon.com/iam/。
2. 在導覽窗格中，選擇 Users (使用者)。
3. 從清單中選擇現有的使用者。

> **Important**
>
> 若要使用績效詳情，請確定除了自訂政策之外，您還可以存取 Amazon DocumentDB。例如，AmazonDocDBReadOnlyAccess 預先定義的政策提供 Amazon DocDB.For詳細資訊，請參閱使用政策管理存取權。

4. 在 Summary (摘要) 頁面上，選擇 Add permissions (新增許可)。
5. 選擇 Attach existing policies directly (直接連接現有政策)。對於 Search (搜尋)，請輸入政策名稱的前幾個字母，如下所示。



6. 選擇您的政策，然後選擇 Next: Review (下一步：檢閱)。
7. 選擇 Add permissions (新增許可)。

設定 Performance Insights AWS KMS 的政策

Performance Insights 使用 AWS KMS key 來加密敏感資料。當您透過 API 或主控台啟用績效詳情時，可使用下列選項：

- 選擇預設值 AWS 受管金鑰。

  Amazon DocumentDB 會將 AWS 受管金鑰 用於新的資料庫執行個體。Amazon DocumentDB 會 AWS 受管金鑰 為 AWS 您的帳戶建立 。每個 AWS 區域的 Amazon DocumentDB AWS 受管金鑰 AWS 帳戶都有不同的 。

- 選擇客戶受管金鑰。

  如果您指定客戶受管金鑰，則您帳戶中呼叫績效詳情 API 的使用者需要 KMS 金鑰的 `kms:Decrypt` 和 `kms:GenerateDataKey` 許可。使用金鑰政策管理 KMS AWS KMS 金鑰的許可，而不是使用 IAM 政策。如需詳細資訊，請參閱在 中使用金鑰政策 AWS KMS。

Example

下列範例金鑰政策說明如何將陳述式新增至 KMS 金鑰政策。這些陳述式允許存取績效詳情。視您使用的方式而定 AWS KMS，您可能想要變更一些限制。在將陳述式新增至您的政策之前，請先移除所有註解。

使用績效詳情儀表板來分析指標

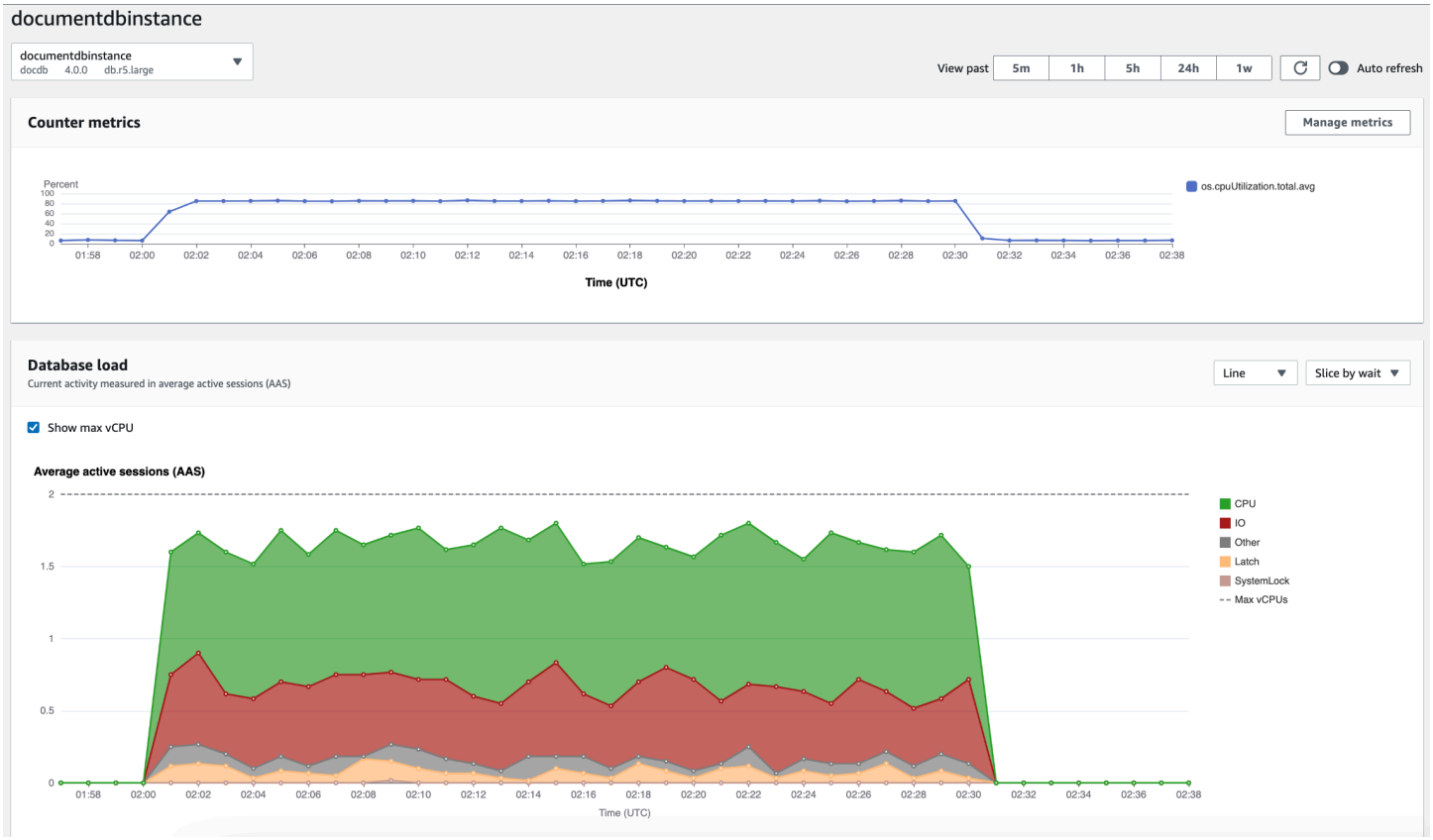
績效詳情儀表板包含資料庫效能資訊，可協助您分析效能問題並對其進行故障排除。在主儀表板頁面上，您可以檢視資料庫載入（資料庫載入）的相關資訊。您可以依等待狀態或查詢等維度來「配量」資料庫負載。

主題

- 績效詳情儀表板概觀
- 開啟績效詳情儀表板
- 依等待狀態分析資料庫負載
- 熱門查詢索引標籤概觀
- 在資料庫負載圖表上放大

績效詳情儀表板概觀

儀表板是與績效詳情進行互動的最簡單方式。下列範例顯示 Amazon DocumentDB 執行個體的儀表板。根據預設，績效詳情儀表板會顯示過去一小時內的資料。



儀表板分成下列部分：

1. 計數器指標 – 顯示特定效能計數器指標的資料。
2. 資料庫負載 – 顯示資料庫負載與資料庫執行個體容量的比較，以最大 vCPU 行表示。
3. 最高維度 - 顯示造成資料庫負載的最高維度。這些維度包括 `waits`、`queries`、`databases`、`hosts`和 `applications`。

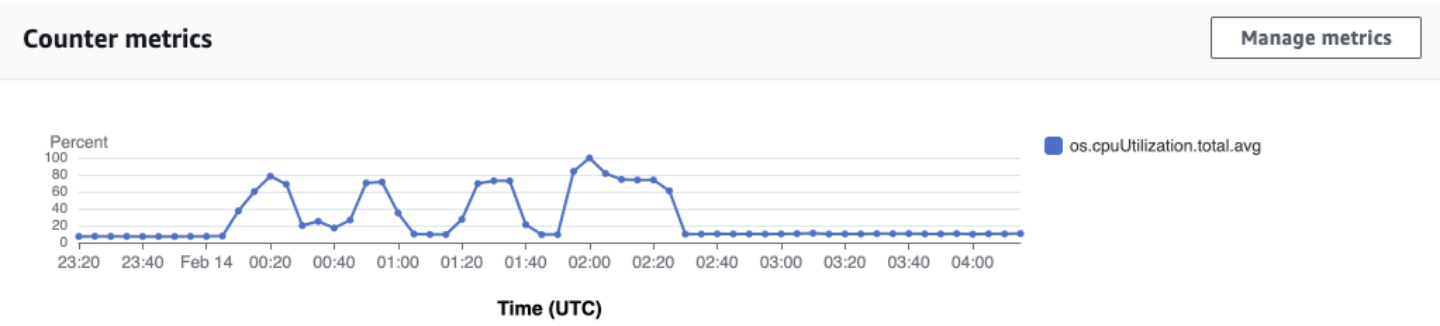
主題

- 計數器指標圖表
- 資料庫負載圖表
- 最高維度表格

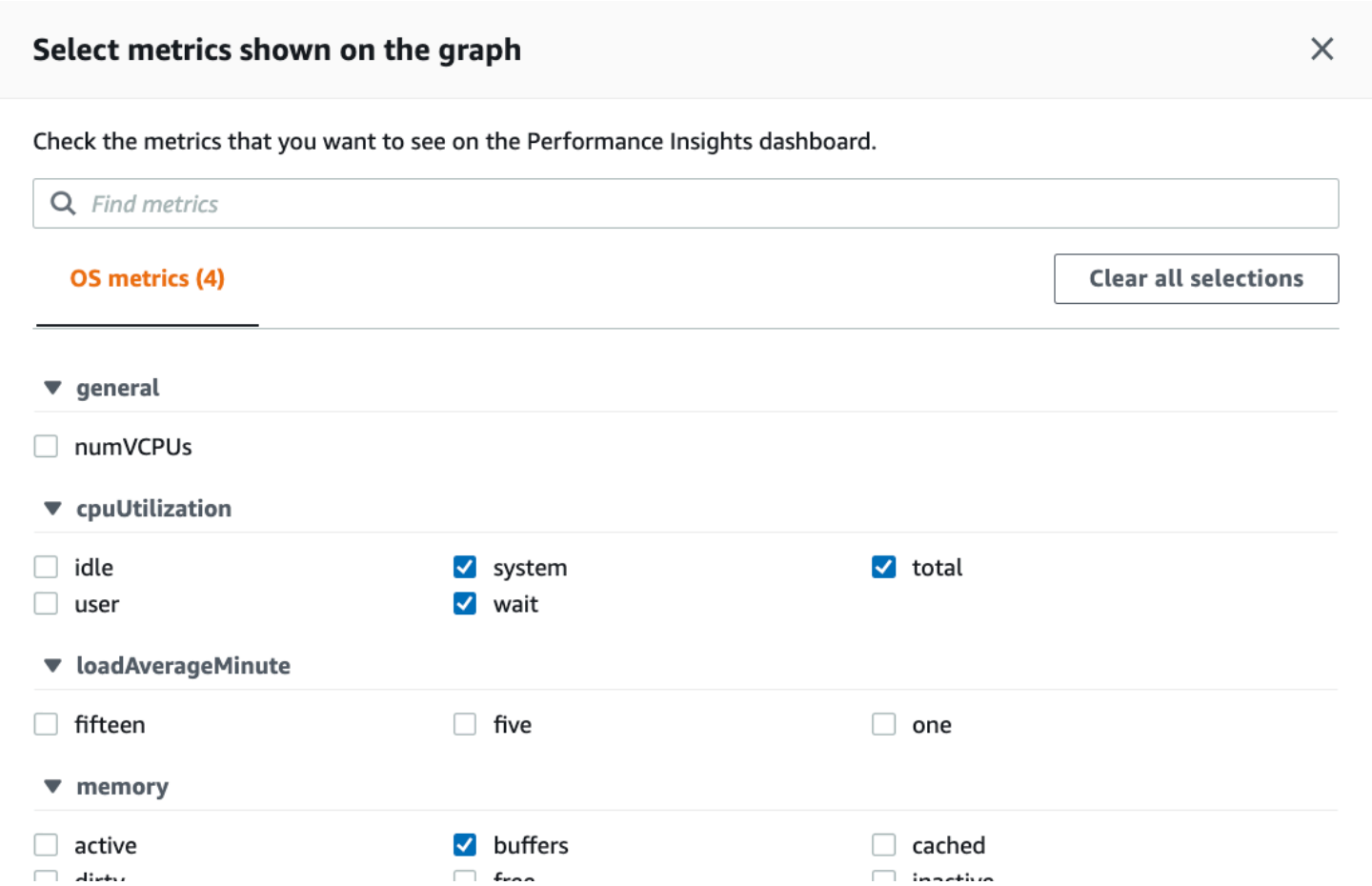
計數器指標圖表

您可以使用計數器指標來自訂績效詳情儀表板，以包含高達 10 個其它圖表。這些圖表顯示數十種作業系統指標的選擇。您可以將此資訊與資料庫負載相互關聯，以協助識別和分析效能問題。

Counter Metrics (計數器指標) 圖表顯示效能計數器的資料。



若要變更效能計數器，請選擇管理指標。您可以選取多個作業系統指標，如下列螢幕擷取畫面所示。若要查看任何指標的詳細資訊，請將游標移到指標名稱上。



資料庫負載圖表

資料庫負載圖表顯示資料庫活動與執行個體容量的比較，如最大 vCPU 行所表示。依預設，堆疊折線圖以每個時間單位的平均作用中工作階段數來表示資料庫負載。資料庫負載依等待狀態切割 (分組)。



資料庫負載依維度配量

您可以選擇將負載顯示為作用中工作階段 (依任何支援維度分組)。下圖顯示 Amazon DocumentDB 執行個體的維度。

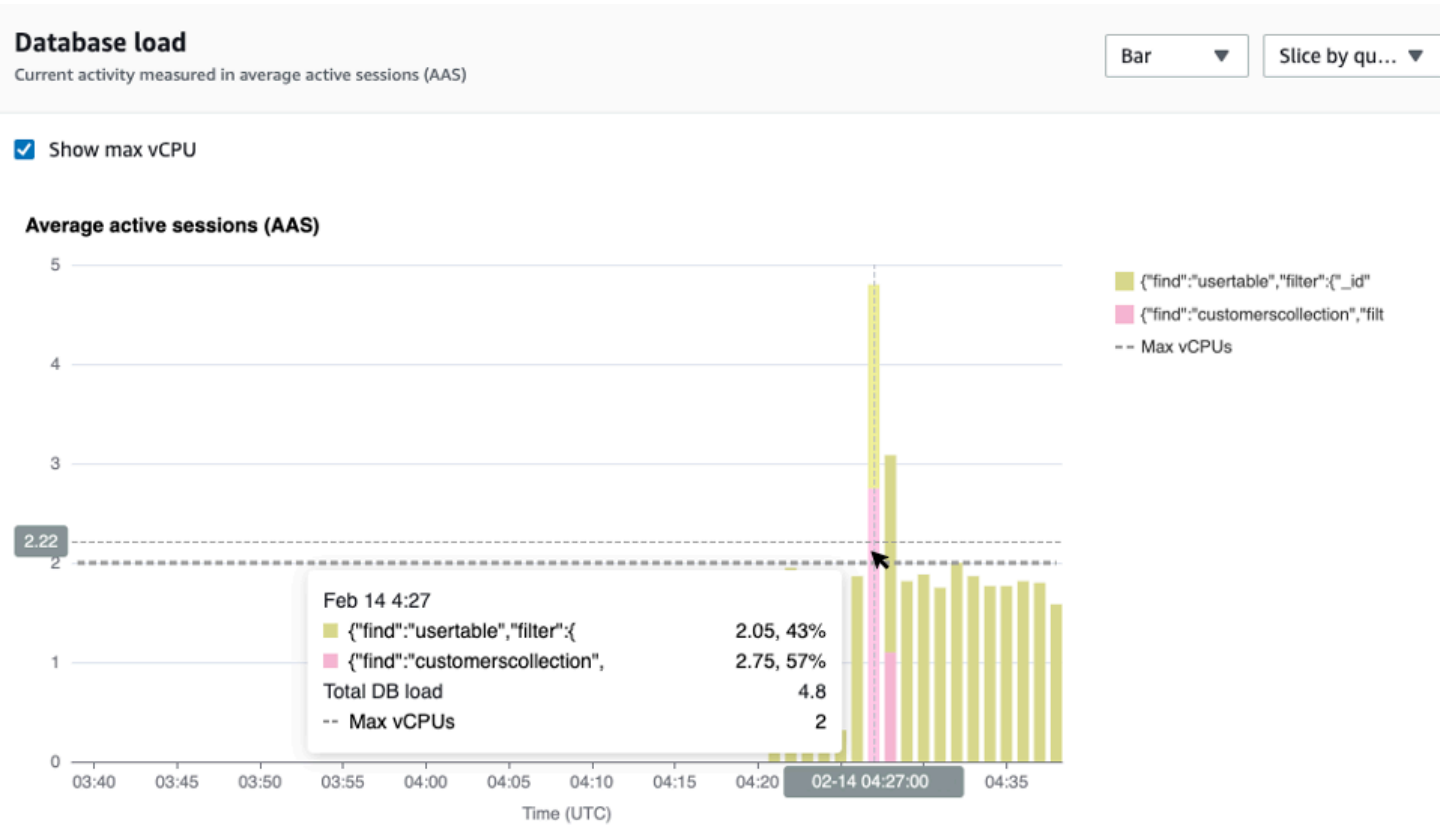


維度項目的資料庫負載詳細資訊

若要查看有關維度內資料庫負載項目的詳細資訊，請將游標移到項目名稱上。下圖顯示查詢陳述式的詳細資訊。



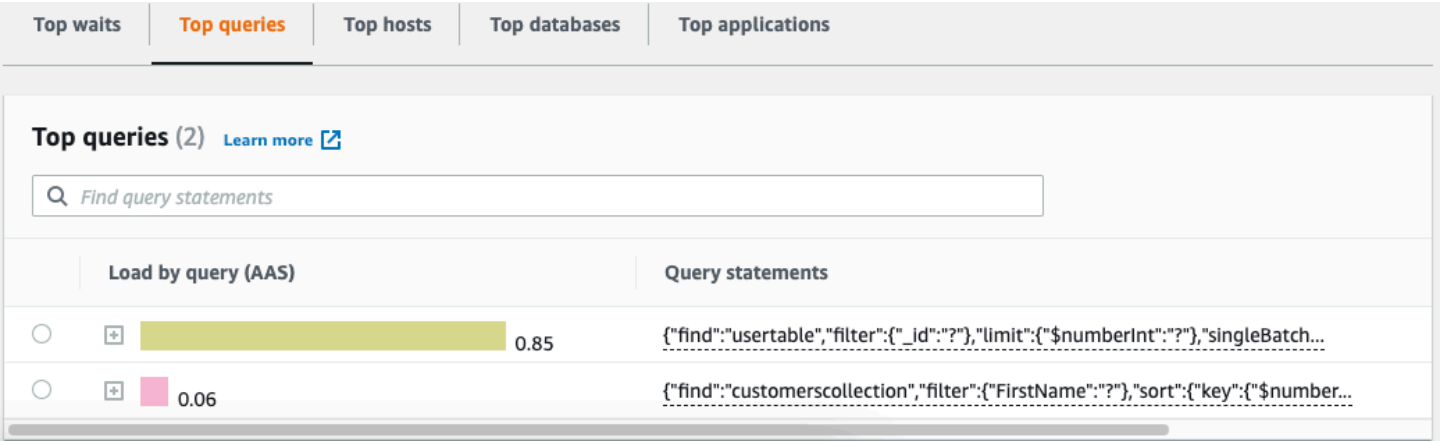
若要以圖例查看在所選時段內任何項目的詳細資訊，請將游標移到該項目上。



最高維度表格

最高維度資料表會依不同維度分割資料庫負載。維度是資料庫負載各種特性的類別或「配量依據」。如果維度是查詢，則熱門查詢會顯示對資料庫負載貢獻最大的查詢陳述式。

選擇下列任一維度索引標籤：



下表提供每個索引標籤的簡短描述。

標籤	說明
最高資料庫後端正在等待的事件	
熱門查詢	

說明

執行的查詢陳述式

用戶端應用程式機器的主機IP和連接埠

最高負載資料庫的資料庫名稱

說明
應用程式名稱

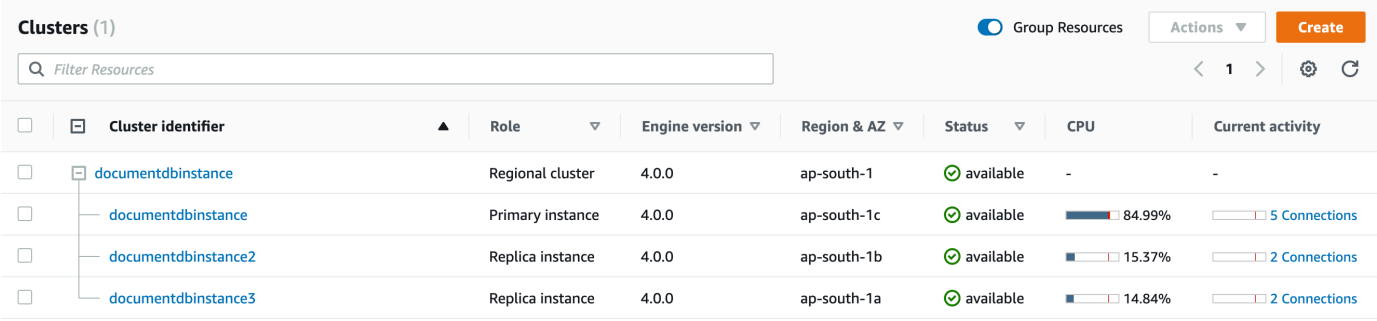
若要了解如何使用熱門查詢索引標籤來分析查詢，請參閱 熱門查詢索引標籤概觀。

開啟績效詳情儀表板

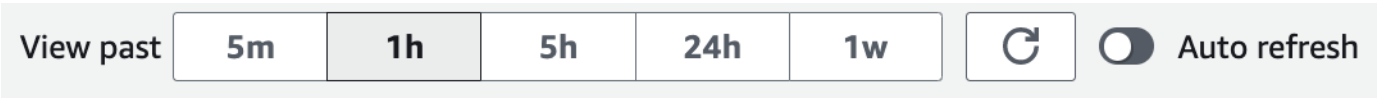
若要在 AWS 管理主控台中檢視績效詳情儀表板，請使用下列步驟：

1. 在 https：//https://console.aws.amazon.com/docdb/ 開啟績效詳情主控台。
2. 選擇資料庫執行個體。會顯示該 Amazon DocumentDB 執行個體的績效詳情儀表板。

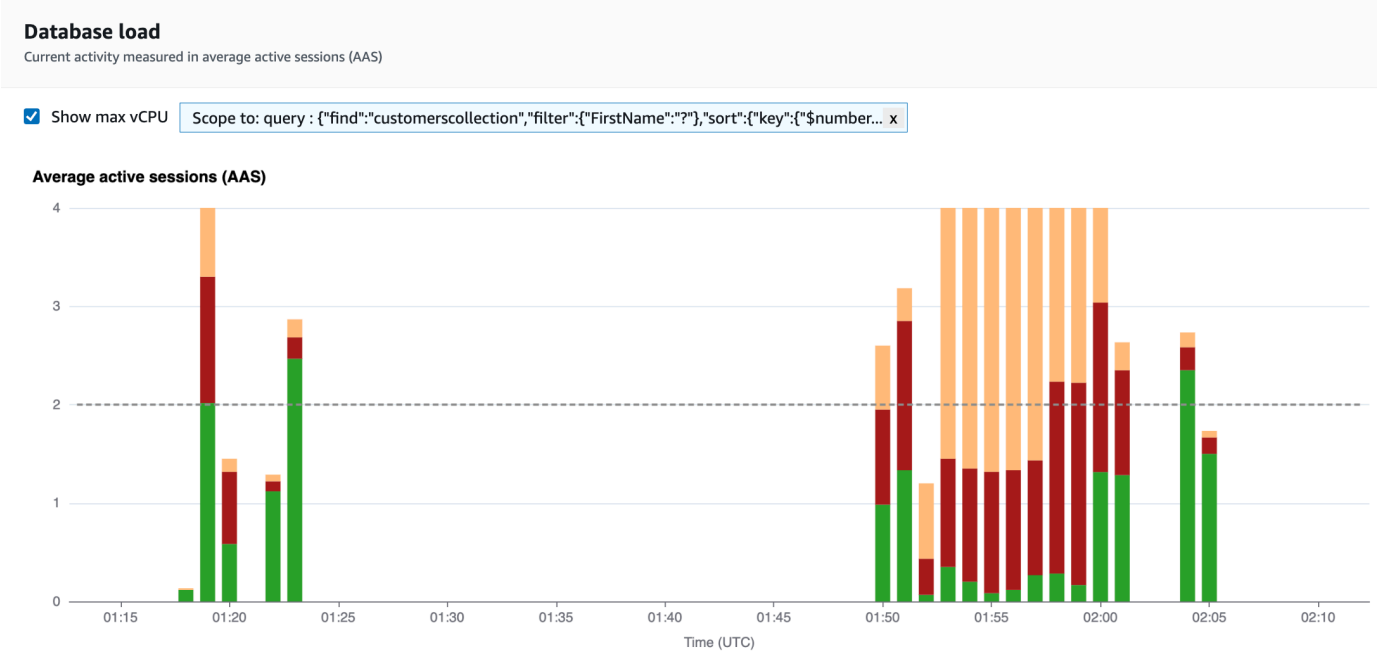
   對於已啟用績效詳情的 Amazon DocumentDB 執行個體，您也可以選擇執行個體清單中的工作階段項目來存取儀表板。在 Current activity (目前活動) 中，Sessions (工作階段) 項目顯示了過去五分鐘內平均作用中工作階段的資料庫負載。負載以進度條圖形的方式顯示。當長條為空時，執行個體會閒置。隨著負載增加，進度條會填入藍色。當負載通過執行個體類別上的虛擬 CPUs vCPUs) 數量時，長條會變成紅色，表示潛在的瓶頸。



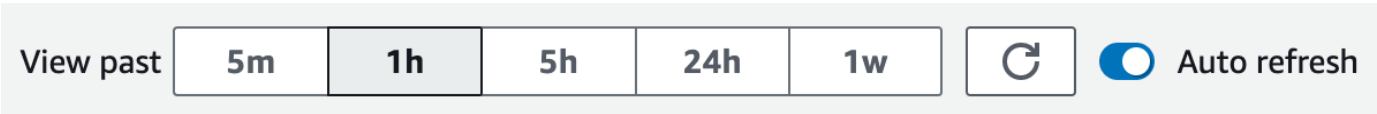
3. (選用) 選取右上角中的按鈕，以選擇不同的時間間隔。例如，若要將間隔變更為 1 小時，請選取 1 小時。



在下列螢幕擷取畫面中，資料庫載入間隔為 1 小時。



4. 若要自動重新整理資料，請啟用自動重新整理。



績效詳情儀表板會自動以新資料進行重新整理。重新整理速度取決於顯示的資料量：

- 5 分鐘的資料每 5 秒重新整理一次。
- 1 小時內的資料每分鐘重新整理一次。

- 5 小時內的資料每分鐘重新整理一次。
- 24 小時內的資料每 5 分鐘重新整理一次。
- 1 週內的資料每小時重新整理。

依等待狀態分析資料庫負載

如果資料庫負載（資料庫負載）圖表顯示瓶頸，您可以找出負載的來源。若要這麼做，請查看資料庫負載圖表下的最高負載項目表格。選擇特定項目，例如查詢或應用程式，以深入了解該項目並查看其詳細資訊。

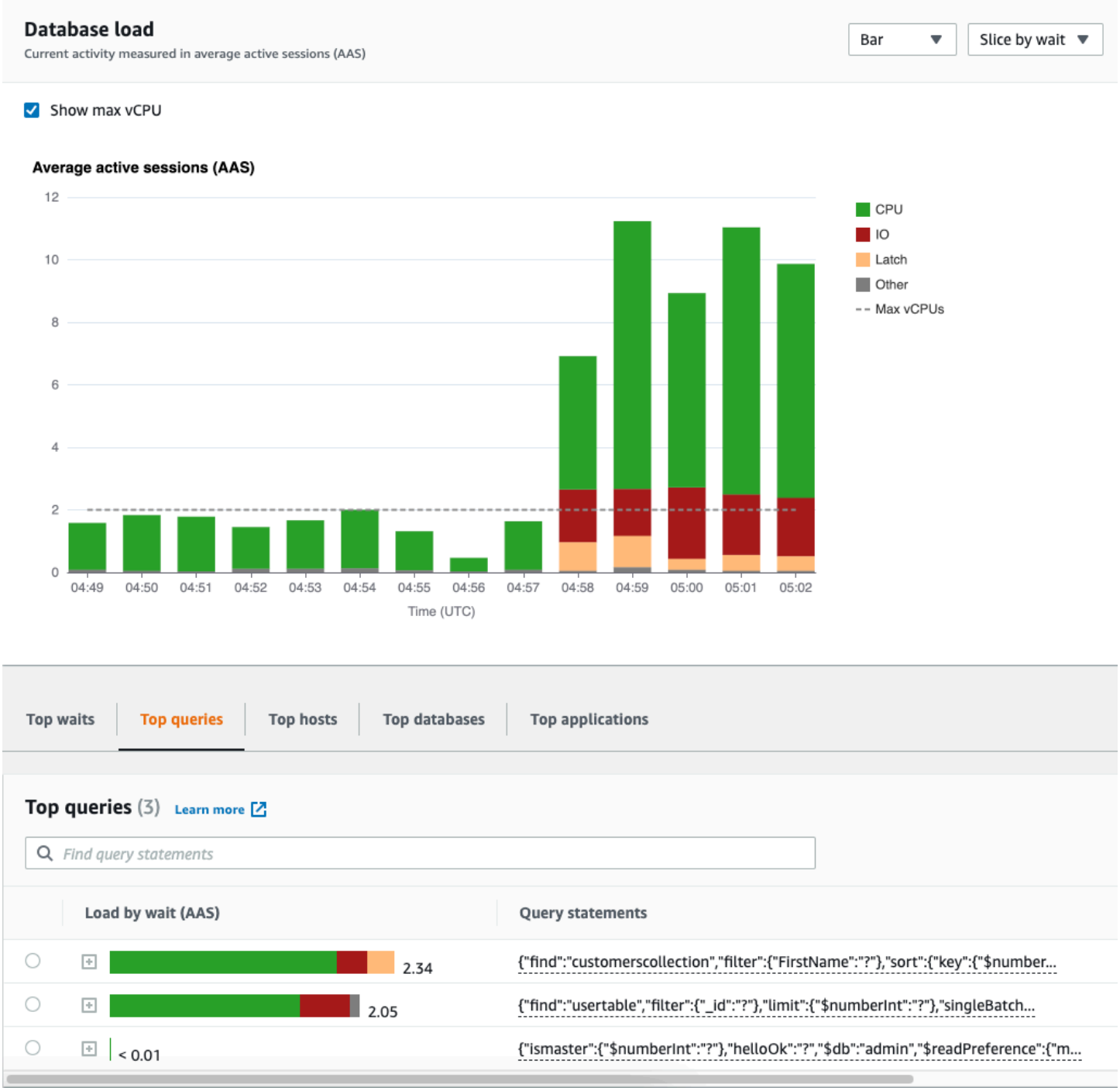
依等待和熱門查詢分組的資料庫負載通常會提供效能問題的最深入見解。根據等待分組的資料庫負載顯示該資料庫中是否有任何資源或正在發生的瓶頸。在此情況下，頂端載入項目資料表的頂端查詢索引標籤會顯示哪些查詢正在驅動該載入。

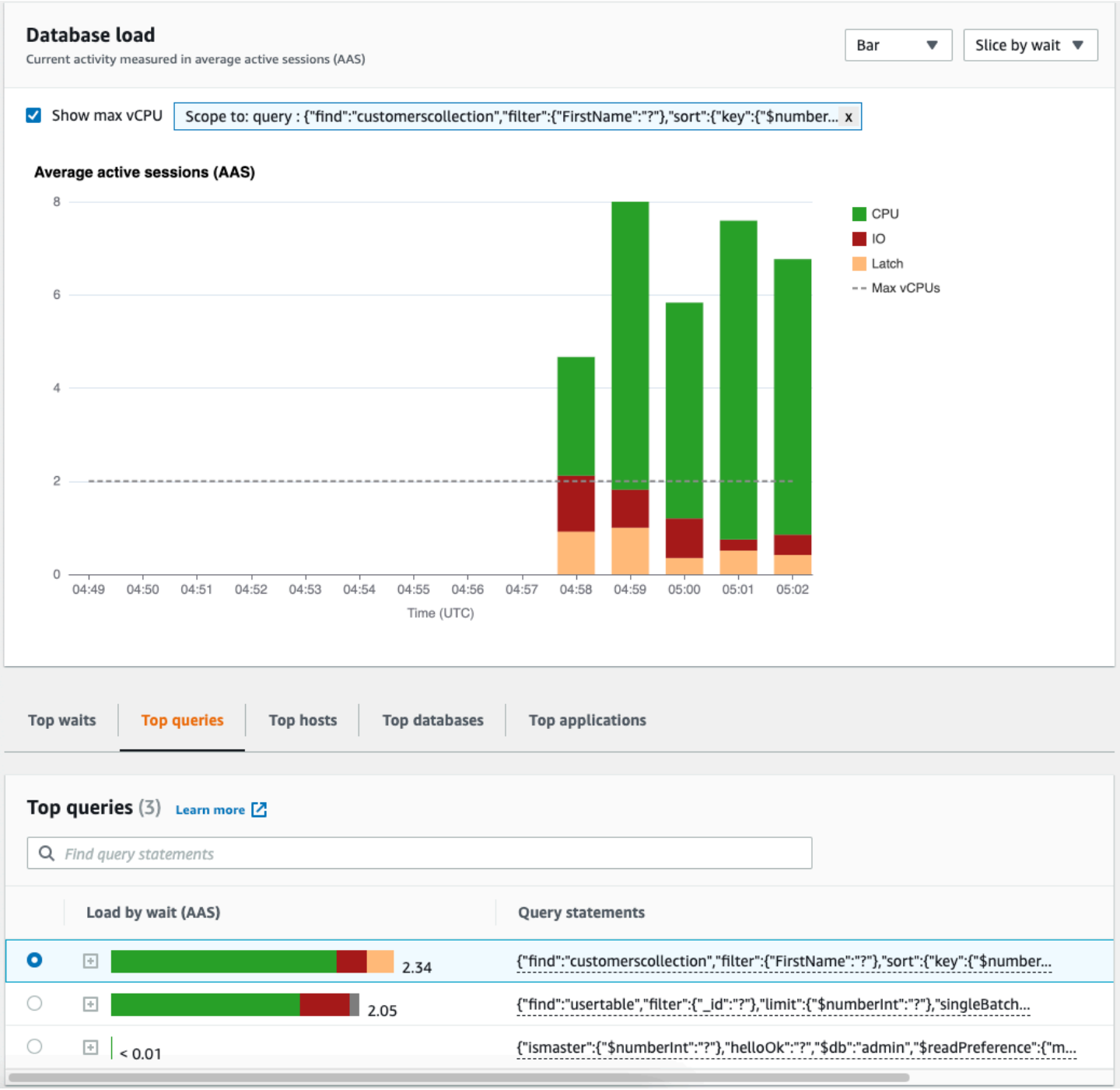
診斷效能問題的典型工作流程如下：

1. 檢閱資料庫負載圖表並查看是否有任何資料庫負載超越最高 CPU 線的情況。
2. 若有，請查看資料庫負載圖表，並找出哪一個或那幾個等待狀態是主因。
3. 透過查看頂端載入項目資料表上的熱門查詢索引標籤對這些等待狀態貢獻最大，來識別導致載入的摘要查詢。您可以依「等待載入 (AAS)」欄來識別這些項目。
4. 在熱門查詢索引標籤中選擇其中一個摘要查詢來展開查詢，並查看由其構成的子查詢。

您也可以分別選取最高主機或最高應用程式，查看哪些主機或應用程式造成最大負載。應用程式名稱是在 Amazon DocumentDB 執行個體的連線字串中指定。 Unknown表示未指定應用程式欄位。

例如，在下列儀表板中，CPU 會等待大部分資料庫負載的帳戶。在 最高查詢 下選取最高查詢 將限制資料庫負載圖表的範圍，以專注於選取查詢所貢獻的最大負載。


Database load
Current activity measured in average active sessions (AAS)
Bar
Slice by wait
Show max vCPU
Average active sessions (AAS)
12
10
8
6
4
2
0
04:49
04:50
04:51
04:52
04:53
04:54
04:55
04:56
04:57
04:58
04:59
05:00
05:01
05:02
Time (UTC)
CPU
IO
Latch
Other
Max vCPUs
Top waits
Top queries
Top hosts
Top databases
Top applications
Top queries (3)
Learn more
Find query statements
Load by wait (AAS)
Query statements
2.34
{"find":"customerscollection","filter":{"FirstName":"?"},"sort":{"key":{"$number...
2.05
{"find":"usertable","filter":{"_id":"?"},"limit":{"$numberInt":"?"},"singleBatch...
< 0.01
{"ismaster":{"$numberInt":"?"},"helloOk":"?","$db":"admin","$readPreference":{"m...



熱門查詢索引標籤概觀

根據預設，頂端查詢索引標籤會顯示對資料庫載入貢獻最大的查詢。您可以分析查詢文字，以協助調整查詢。

主題

- 查詢摘要

- 依等待分組的負載 (AAS)
- 檢視詳細查詢資訊
- 存取陳述式查詢文字
- 檢視和下載陳述式查詢文字

查詢摘要

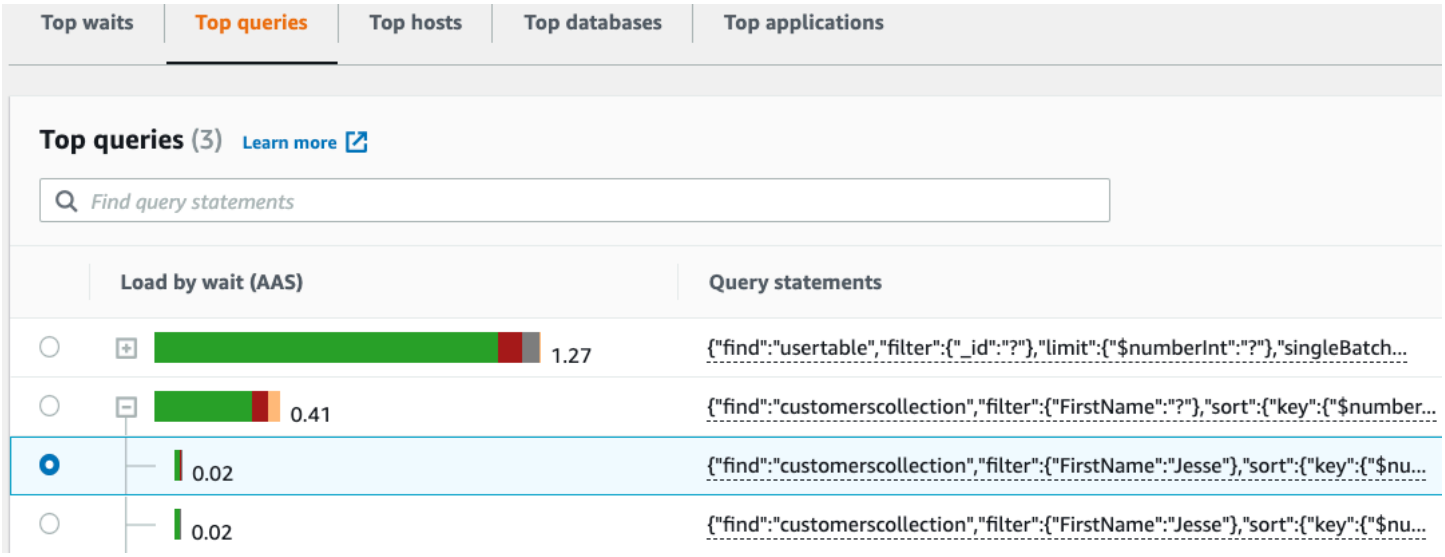
查詢摘要是多個實際查詢的複合，這些查詢在結構上類似，但可能有不同的常值。摘要中，問號會取代硬式編碼值。例如，查詢摘要可能如下所示：

```
{"find":"customerscollection","filter":{"FirstName":"?"},"sort":{"key":
{"$numberInt":"?"}},"limit":{"$numberInt":"?"}}
```

此摘要可能包含下列子查詢：

```
{"find":"customerscollection","filter":{"FirstName":"Karrie"},"sort":{"key":
{"$numberInt":"1"}},"limit":{"$numberInt":"3"}}
{"find":"customerscollection","filter":{"FirstName":"Met"},"sort":{"key":
{"$numberInt":"1"}},"limit":{"$numberInt":"3"}}
{"find":"customerscollection","filter":{"FirstName":"Rashin"},"sort":{"key":
{"$numberInt":"1"}},"limit":{"$numberInt":"3"}}
```

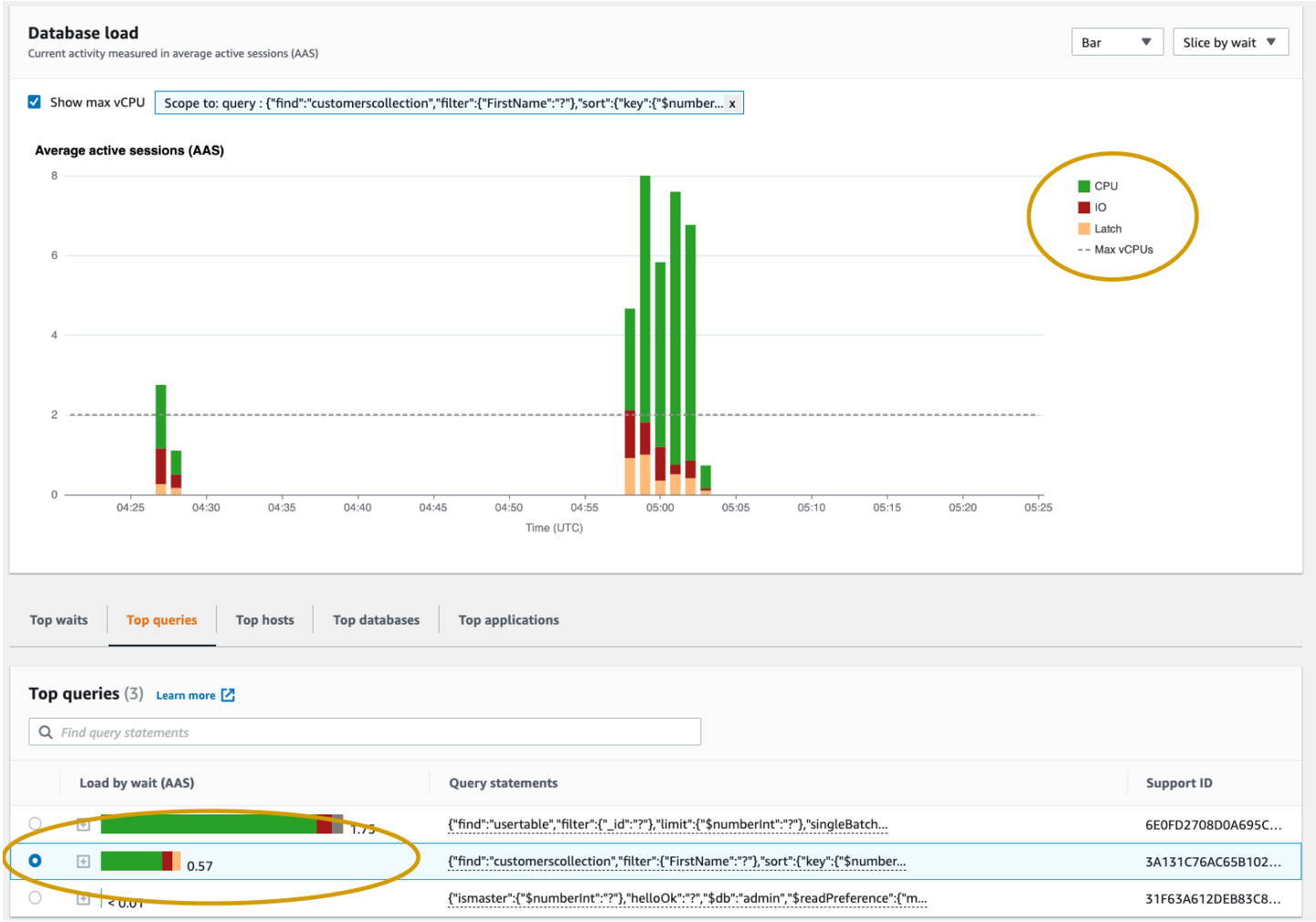
若要查看摘要中的常值查詢陳述式，請選取查詢，然後選擇加號 (+)。在下列螢幕擷取畫面中，所選查詢為摘要。



> **Note**
>
> 查詢摘要會將類似的查詢陳述式分組，但不會修訂敏感資訊。

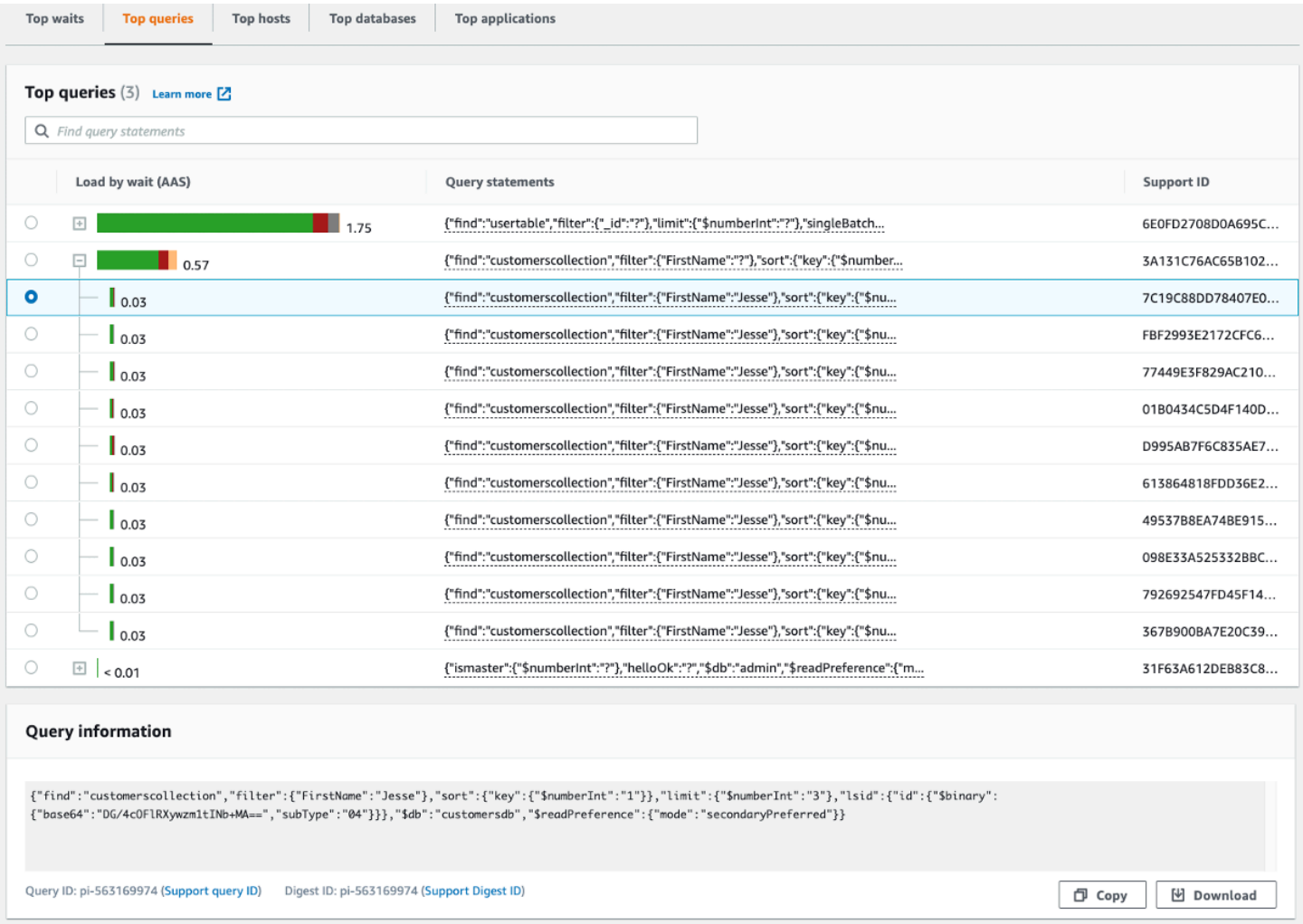
依等待分組的負載 (AAS)

在最高查詢中，依等待載入 (AAS) 欄說明與每個最高載入項目相關聯的資料庫載入百分比。此欄會依目前在資料庫負載圖表中選取的任何分組來反映該項目的負載。例如，您可以依等待狀態將 DB load (資料庫負載) 圖表分組。在此情況下，根據等待列出資料庫負載列較大且分段，並以顏色為代碼來顯示該查詢所帶來的指定等待狀態程度。這也會顯示哪些等待狀態會影響選取的查詢。



檢視詳細查詢資訊

在頂端查詢表格中，您可以開啟摘要陳述式來檢視其資訊。資訊會顯示在底部窗格中。



下列識別符 (IDs) 類型與查詢陳述式相關聯：

1. 支援查詢 ID – 查詢 ID 的雜湊值。此值僅適用於使用 AWS Support 時參考查詢 ID。 AWS Support 無法存取您的實際查詢 IDs和查詢文字。
2. 支援摘要 ID – 摘要 ID 的雜湊值。此值僅適用於使用 AWS Support 時參考摘要 ID。 AWS Support 無法存取您的實際摘要 IDs 和查詢文字。

存取陳述式查詢文字

根據預設，熱門查詢表格中的每一列都會為每個查詢陳述式顯示 500 個位元組的查詢文字。當摘要陳述式超過 500 個位元組時，您可以在績效詳情儀表板中開啟陳述式來檢視更多文字。在此情況下，所顯示查詢的長度上限為 1 KB。如果您檢視完整的查詢陳述式，您也可以選擇下載。

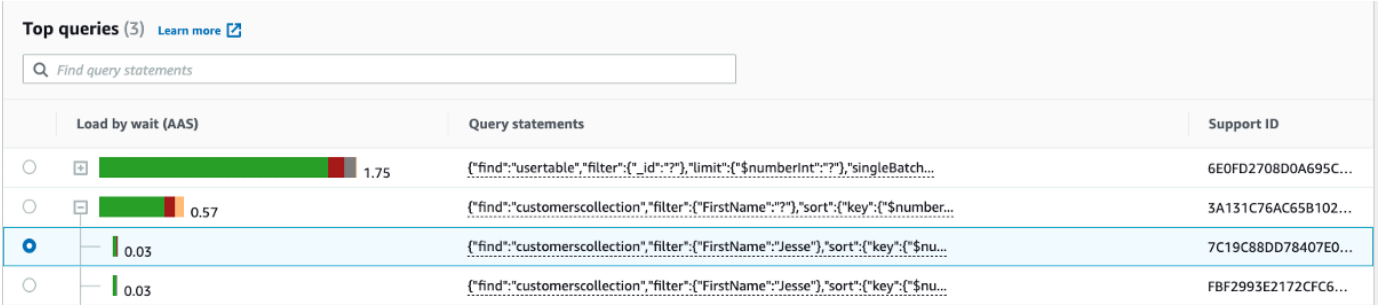
檢視和下載陳述式查詢文字

在績效詳情儀表板中，您可以檢視或下載查詢文字。

在績效詳情儀表板中檢視更多查詢文字

1. 開啟 Amazon DocumentDB 主控台，網址為：https：//https://console.aws.amazon.com/docdb/
2. 在導覽窗格中，選擇 Performance Insights (績效詳情)。
3. 選擇資料庫執行個體。顯示該資料庫執行個體的績效詳情儀表板。

   文字大於 500 個位元組的查詢陳述式看起來如下圖所示：



4. 檢查查詢資訊區段以檢視更多查詢文字。



績效詳情儀表板最多可顯示每個完整查詢陳述式的 1 KB。

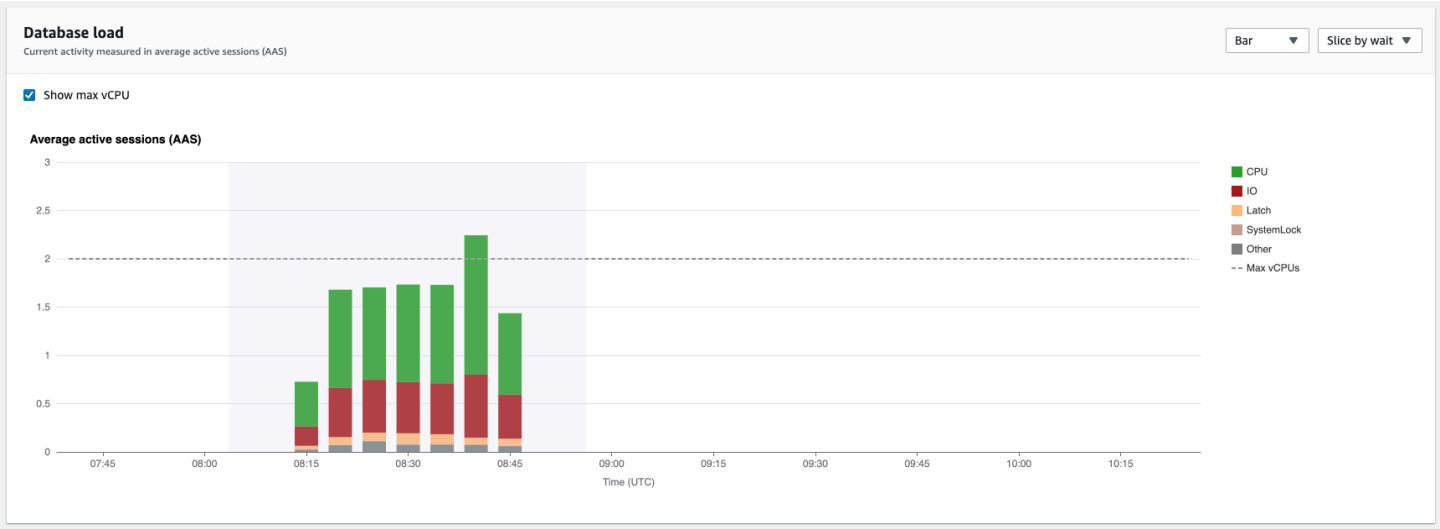
> **Note**
>
> 若要複製或下載查詢陳述式，請停用任何快顯封鎖程式。

在資料庫負載圖表上放大

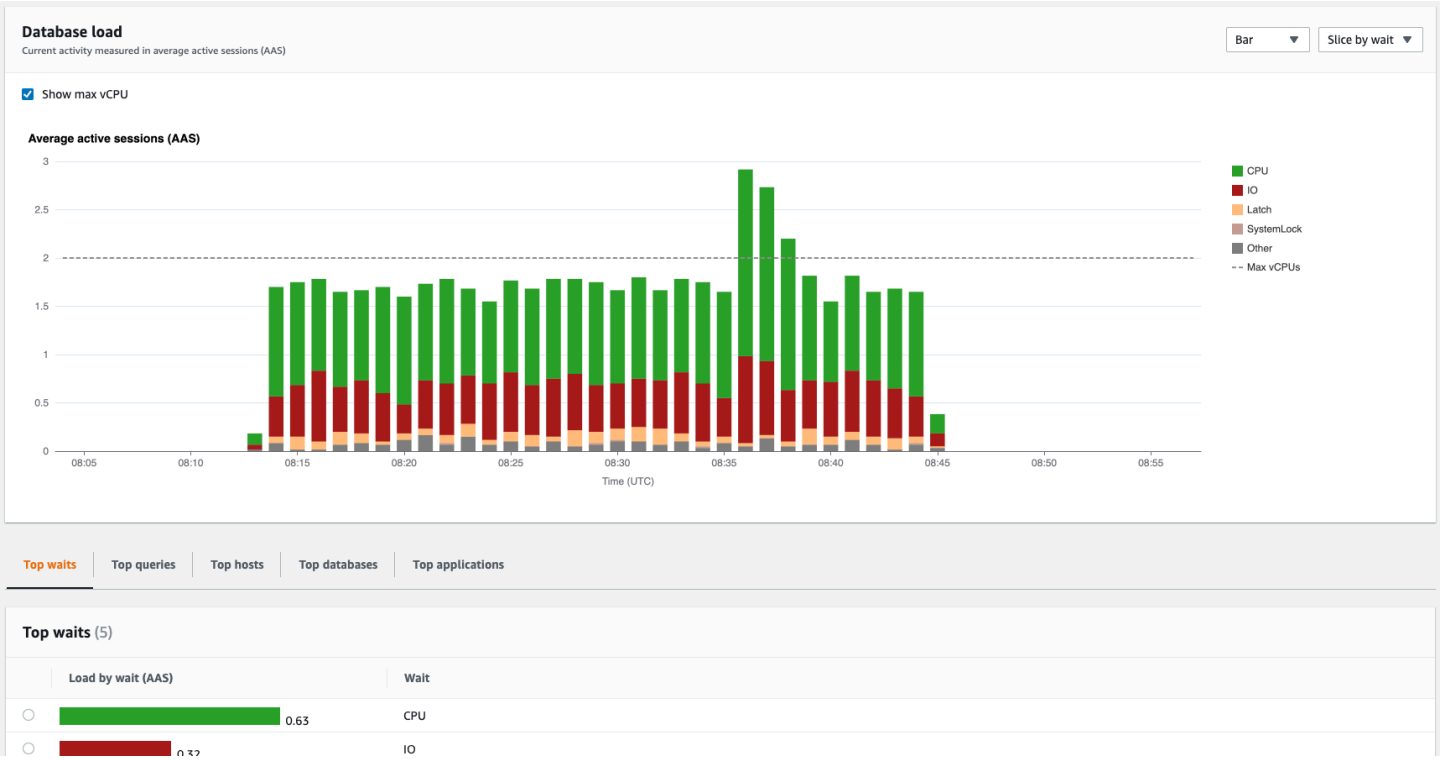
您可以使用績效詳情使用者界面的其他功能來協助並分析效能資料。

長按拖曳來放大

在績效詳情界面中，您可以選擇負載圖表的一小部分，並放大詳細資訊。



若要放大小部分的負載表格，選擇開始時間並拖曳到您想要的時間範圍結束時間。您這麼做的話，所選區域將有反白顯示。當您放開滑鼠時，負載圖表會放大所選區域，並重新計算最高項目表。



使用績效詳情 API 來擷取指標

在啟用績效詳情時，API 會提供對執行個體效能的可見性。Amazon CloudWatch Logs 提供授權來源，以用於 AWS 服務的付費監控指標。

績效詳情提供以平均作用中工作階段 (AAS) 評估的資料庫負載特定網域檢視。此指標在 API 消費者看來是二維時間序列資料集。資料的時間維度提供查詢的時間範圍內各時間點的資料庫負載資料。每個時間點會根據請求的維度來分解整體負載，例如 `Query`、`Wait-state`、`Application`、或者 `Host`，在該時間點所測得。

Amazon DocumentDB Performance Insights 會監控您的 Amazon DocumentDB 資料庫執行個體，讓您可以分析資料庫效能並進行疑難排解。檢視績效詳情資料的一個方法就是使用 AWS 管理主控台。績效詳情也提供公有 API，讓您可以查詢自己的資料。您可以使用 API 執行下列動作：

- 將資料卸載至資料庫
- 將績效詳情資料新增至現有監控儀表板
- 建置監控工具

若要使用績效詳情 API，請在其中一個 Amazon DocumentDB 執行個體上啟用績效詳情。如需啟用績效詳情的相關資訊，請參閱 啟用和停用績效詳情。如需績效詳情 API 的相關詳細資訊，請參閱 績效詳情 API 參考。

績效詳情 API 提供下列操作。

績效詳情動作	AWS CLI 命令	說明
DescribeDimensionKeys	aws pi describe-dimension-keys	針對特定時段，擷取其指標的前 N 個維度金鑰。
GetDimensionKeyDetails	aws pi get-dimension-key-details	擷取資料庫執行個體或資料來源之指定維度群組的屬性。例如，如果您指定查詢 ID，而且有可用的維度詳細資訊，會`GetDimensionKeyDetails` 擷取與此 ID `db.query.statement` 相關聯的維度全文。此操作非常有用，因為 `GetResourceMetrics` 和 `DescribeDimensionKeys` 不支援擷取大型查詢陳述式文字。

績效詳情動作	AWS CLI 命令	說明
GetResourceMetadata	aws pi get-resource-metadata	檢索不同功能的中繼資料。例如，中繼資料可能指出特定資料庫執行個體上某項功能已開啟或關閉。
GetResourceMetrics	aws pi get-resource-metrics	擷取一組資料來源某個時段的績效詳情指標。您可以提供特定維度群組和維度，以及為每個群組提供彙總和篩選條件。
ListAvailableResourceDimensions	aws pi list-available-resource-dimensions	檢索指定執行個體上每個指定指標類型可查詢的維度。
ListAvailableResourceMetrics	aws pi list-available-resource-metrics	檢索可為指定資料庫執行個體查詢的指定指標類型中所有可用的指標。

主題

- AWS CLI for Performance Insights
- 擷取時間序列指標
- AWS CLI 績效詳情的範例

AWS CLI for Performance Insights

您可以使用 AWS CLI檢視績效詳情資料。您可以透過在命令列輸入以下內容來檢視績效詳情 AWS CLI 命令的說明。

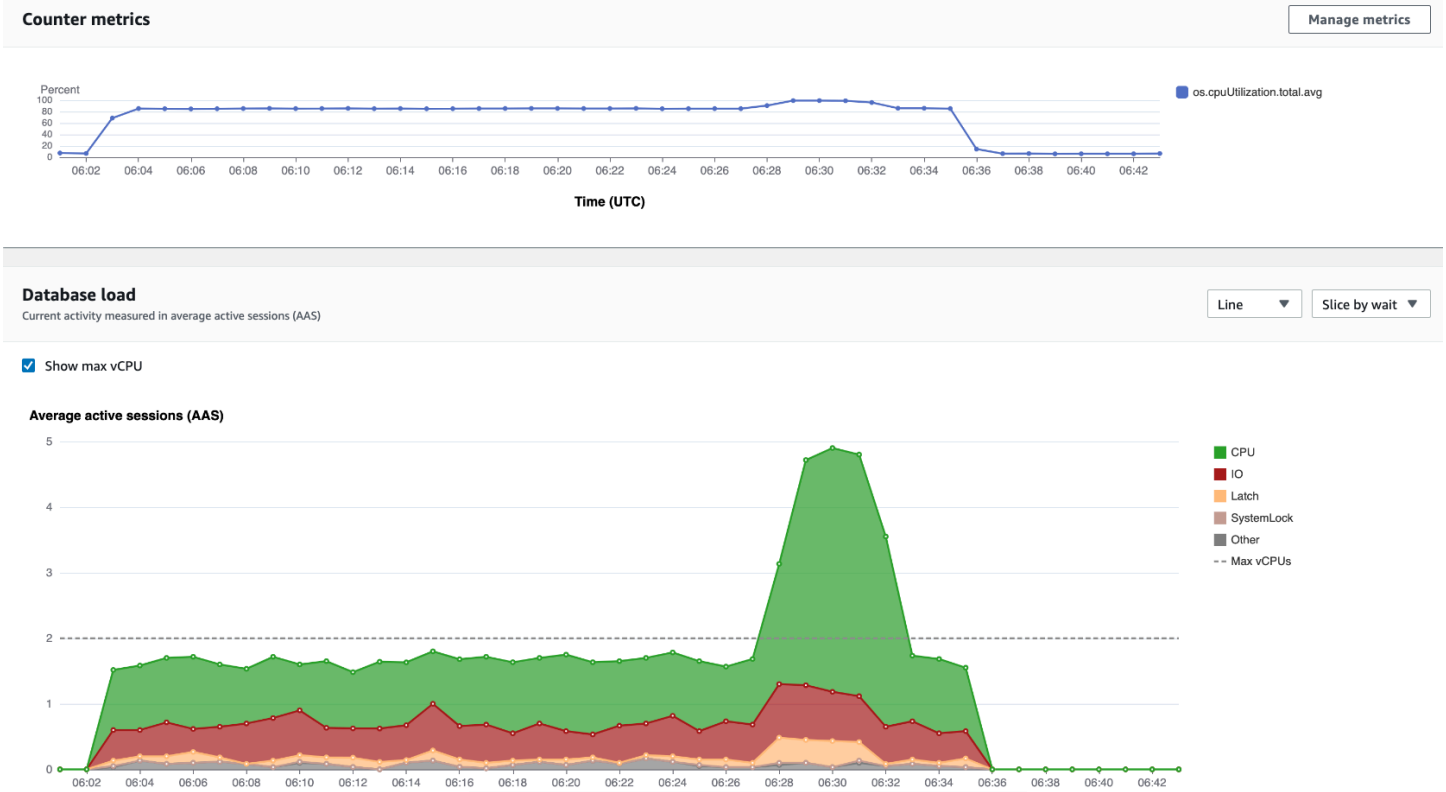
```
aws pi help
```

如果您尚未 AWS CLI 安裝 ，請參閱AWS CLI 《 使用者指南》中的安裝 AWS 命令列界面，以取得安裝它的相關資訊。

擷取時間序列指標

GetResourceMetrics 操作會從績效詳情資料中擷取一或多個時間時間序列指標。GetResourceMetrics 需要指標和時間間隔，並傳回含資料點清單的回應。

例如， AWS 管理主控台 使用 GetResourceMetrics 填入計數器指標圖表和資料庫負載圖表，如下圖所示。



GetResourceMetrics 傳回的所有指標，除 db.load 之外，皆為標準的時間序列指標。此指標會顯示在 Database Load (資料庫負載) 圖表中。db.load 指標與其他時間序列指標不同，因為您可以將它分為名為維度的子元件。在先前的影像中，db.load 已被細分，分組依據為組成 db.load 的等待狀態。

> **Note**
>
> GetResourceMetrics 也可以傳回 db.sampleload 指標，但 db.load 指標適用於大部分情況。

如需 GetResourceMetrics 所傳回指標的相關資訊，請參閱計數器指標的績效詳情。

這些指標支援下列計算：

- 平均值 – 指標在一段時間內的平均值。將 .avg 附加至指標名稱。
- 最小值 – 指標在一段時間內的最小值。將 .min 附加至指標名稱。
- 最大值 – 指標在一段時間內的最大值。將 .max 附加至指標名稱。
- 總和 – 指標值在一段時間內的總和。將 .sum 附加至指標名稱。
- 取樣計數 – 在一段時間內收集指標的次數。將 .sample_count 附加至指標名稱。

例如，假設收集指標的時間為 300 秒 (5 分鐘)，且每分鐘收集一次指標。每分鐘的值為 1、2、3、4 和 5。在此情況下，會傳回下列計算：

- 平均值 – 3
- 最小值 – 1
- 最大值 – 5
- 總和 – 15
- 取樣計數 – 5

如需使用 get-resource-metrics AWS CLI 命令的詳細資訊，請參閱 get-resource-metrics。

對於 --metric-queries 選項，請指定您要取得結果的一或多個查詢。每個查詢的組成為必要的 Metric 和選用的 GroupBy 及 Filter 參數。以下是 --metric-queries 選項規格的範例。

```
{
   "Metric": "string",
   "GroupBy": {
     "Group": "string",
     "Dimensions": ["string", ...],
     "Limit": integer
   },
   "Filter": {"string": "string"
     ...}
```

AWS CLI 績效詳情的範例

下列範例示範如何使用 AWS CLI for Performance Insights 的 。

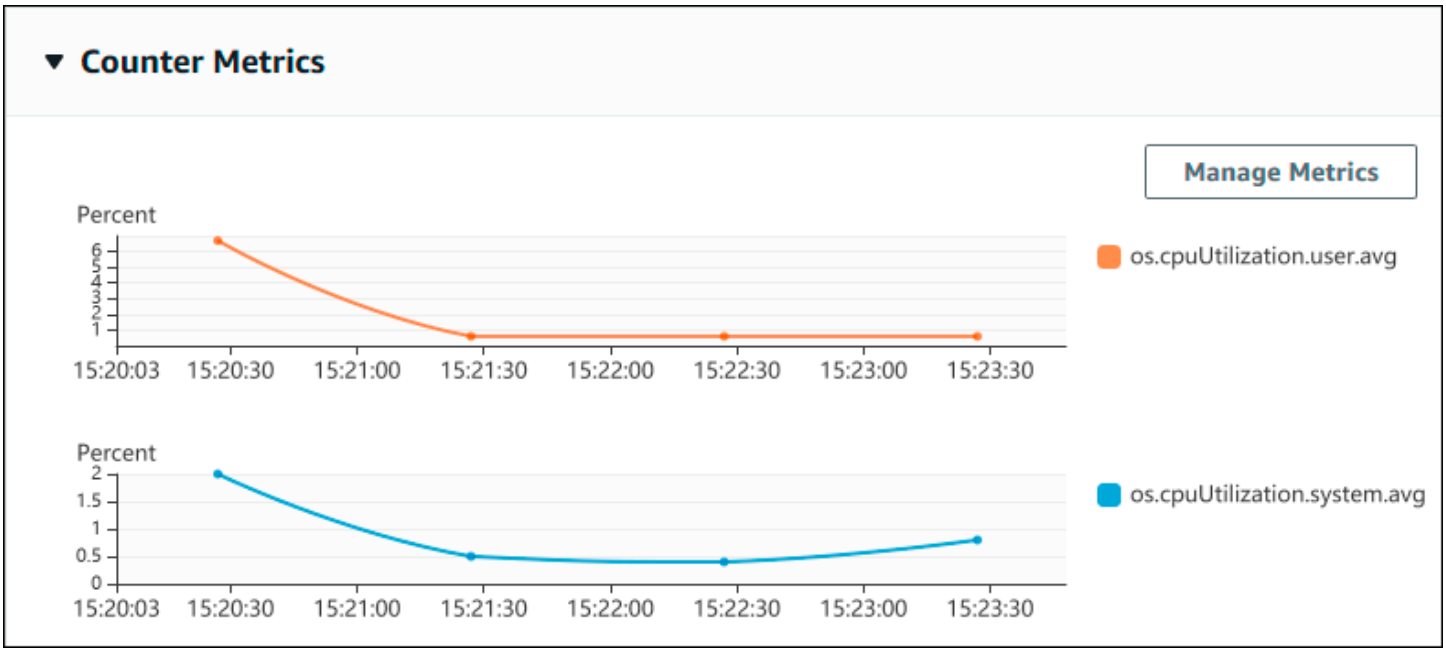
主題

- 擷取計數器指標

- 擷取最高等待狀態的資料庫負載平均值
- 擷取熱門查詢的資料庫負載平均值
- 擷取依查詢篩選的資料庫負載平均值

擷取計數器指標

下列螢幕擷取畫面顯示 AWS 管理主控台中的兩個計數器指標圖表。



下凸顯是如何蒐集 AWS 管理主控台 用來產生兩個計數器指標圖表的相同資料。

針對 Linux、macOS 或 Unix：

```
aws pi get-resource-metrics \
   --service-type DOCDB \
   --identifier db-ID \
   --start-time 2022-03-13T8:00:00Z \
   --end-time   2022-03-13T9:00:00Z \
   --period-in-seconds 60 \
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

針對 Windows：

```
aws pi get-resource-metrics ^
   --service-type DOCDB ^
```

```
   --identifier db-ID ^
   --start-time 2022-03-13T8:00:00Z ^
   --end-time   2022-03-13T9:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

您也可以透過指定 `--metrics-query` 選項的檔案來提高命令的可讀性。以下範例會將名為 query.json 的檔案用於此選項。此檔案的內容如下。

```
[
    {
        "Metric": "os.cpuUtilization.user.avg"
    },
    {
        "Metric": "os.cpuUtilization.idle.avg"
    }
]
```

執行下列命令來使用檔案。

針對 Linux、macOS 或 Unix：

```
aws pi get-resource-metrics \
   --service-type DOCDB \
   --identifier db-ID \
   --start-time 2022-03-13T8:00:00Z \
   --end-time   2022-03-13T9:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

針對 Windows：

```
aws pi get-resource-metrics ^
   --service-type DOCDB ^
   --identifier db-ID ^
   --start-time 2022-03-13T8:00:00Z ^
   --end-time   2022-03-13T9:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

先前的範例會為選項指定下列值：

- `--service-type` – DOCDB適用於 Amazon DocumentDB
- `--identifier`– 資料執行個體的資源 ID
- `--start-time` 和 `--end-time` – 要查詢期間的 ISO 8601 DateTime 值，支援多種格式

它會查詢一小時的時間範圍：

- `--period-in-seconds`–60 適用於每分鐘的查詢
- `--metric-queries`– 兩個查詢的陣列，一個指標剛好一個查詢。

  此指標名稱會使用點將指標分類在實用的類別，其中最後一個元素則做為函數。在此範例中，此函數是每個查詢的 `avg`。如同 Amazon CloudWatch，支援的函數是 `min`、`max`、`total` 和 `avg`。

回應看起來類似以下的內容。

```
{
    "AlignedStartTime": "2022-03-13T08:00:00+00:00",
    "AlignedEndTime": "2022-03-13T09:00:00+00:00",
    "Identifier": "db-NQF3TTMFQ3GTOKIMJODMC3KQQ4",
    "MetricList": [
        {
            "Key": {
                "Metric": "os.cpuUtilization.user.avg"
            },
            "DataPoints": [
                {
                    "Timestamp": "2022-03-13T08:01:00+00:00", //Minute1
                    "Value": 3.6
                },
                {
                    "Timestamp": "2022-03-13T08:02:00+00:00", //Minute2
                    "Value": 2.6
                },
                //.... 60 datapoints for the os.cpuUtilization.user.avg metric
        {
            "Key": {
                "Metric": "os.cpuUtilization.idle.avg"
            },
            "DataPoints": [
                {
                    "Timestamp": "2022-03-13T08:01:00+00:00",
```

```
                    "Value": 92.7
                },
                {
                    "Timestamp": "2022-03-13T08:02:00+00:00",
                    "Value": 93.7
                },
                //.... 60 datapoints for the os.cpuUtilization.user.avg metric
            ]
        }
    ] //end of MetricList
} //end of response
```

回應具有 Identifier、AlignedStartTime 和 AlignedEndTime。--period-in-seconds 值為 60，開始和結束時間皆一致使用分鐘。如果 --period-in-seconds 是 3600，開始和結束時間則會一致使用小時。

回應中的 MetricList 擁有許多項目，每個都包含 Key 和 DataPoints 項目。每個 DataPoint 都有 Timestamp 和 Value。每個 Datapoints 清單有 60 個資料點，因為查詢是適用於一小時中的每分鐘資料，內含 Timestamp1/Minute1、Timestamp2/Minute2 等，最多可達 Timestamp60/Minute60。

因為此查詢是適用於兩個不同的計數器指標，回應 MetricList 中會有兩個元素。

擷取最高等待狀態的資料庫負載平均值

下列範例與 AWS 管理主控台 用來產生堆疊區域折線圖的查詢相同。此範例會擷取最後一個小時db.load.avg的 根據前七個等待狀態分割負載的 。此命令與 擷取計數器指標 中的命令相同。然而，查詢 query.json 檔案有以下內容。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_state", "Limit": 7 }
    }
]
```

執行下列命令。

針對 Linux、macOS 或 Unix：

```
aws pi get-resource-metrics \
```

```
   --service-type DOCDB \
   --identifier db-ID \
   --start-time 2022-03-13T8:00:00Z \
   --end-time   2022-03-13T9:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

針對 Windows：

```
aws pi get-resource-metrics ^
   --service-type DOCDB ^
   --identifier db-ID ^
   --start-time 2022-03-13T8:00:00Z ^
   --end-time   2022-03-13T9:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

此範例會指定 的指標，db.load.avg以及前七個等待狀態GroupBy的 。如需此範例有效值的詳細資訊，請參閱績效詳情 API 參考中的DimensionGroup。

回應看起來類似以下的內容。

```
{
    "AlignedStartTime": "2022-04-04T06:00:00+00:00",
    "AlignedEndTime": "2022-04-04T06:15:00+00:00",
    "Identifier": "db-NQF3TTMFQ3GTOKIMJODMC3KQQ4",
    "MetricList": [
        {//A list of key/datapoints
            "Key": {
                //A Metric with no dimensions. This is the total db.load.avg
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": "2022-04-04T06:01:00+00:00",//Minute1
                    "Value": 0.0
                },
                {
                    "Timestamp": "2022-04-04T06:02:00+00:00",//Minute2
                    "Value": 0.0
```

```
                },
                //... 60 datapoints for the total db.load.avg key
                ]
        },
        {
            "Key": {
                //Another key. This is db.load.avg broken down by CPU
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_state.name": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": "2022-04-04T06:01:00+00:00",//Minute1
                    "Value": 0.0
                },
                {
                    "Timestamp": "2022-04-04T06:02:00+00:00",//Minute2
                    "Value": 0.0
                },
                //... 60 datapoints for the CPU key
            ]
        },//... In total we have 3 key/datapoints entries, 1) total, 2-3) Top Wait
 States
    ] //end of MetricList
} //end of response
```

在此回應中， 中有三個項目MetricList。總計 有一個項目db.load.avg，根據前三個等待狀態之一db.load.avg分割的 各有三個項目。由於有分組維度（與第一個範例不同），因此指標的每個分組都必須有一個索引鍵。每個指標不能只有一個索引鍵，如同基本計數器指標使用案例。

擷取熱門查詢的資料庫負載平均值

下列範例db.wait_state依前 10 個查詢陳述式分組。查詢陳述式有兩種不同的群組：

- db.query – 完整的查詢陳述式，例如 {"find":"customers","filter": {"FirstName":"Jesse"},"sort":{"key":{"$numberInt":"1"}}}
- db.query_tokenized – 字符化查詢陳述式，例如 {"find":"customers","filter": {"FirstName":"?"},"sort":{"key":{"$numberInt":"?"}},"limit": {"$numberInt":"?"}}

分析資料庫效能時，將僅因參數而不同的查詢陳述式視為一個邏輯項目非常有用。因此，您可以在查詢時使用 db.query_tokenized。不過，特別是當您對 感興趣時explain()，有時使用參數檢查完整的查詢陳述式會比較有用。字符化查詢與完整查詢之間存在父子關係，多個完整查詢 （子） 分組在相同的字符化查詢 （父） 下。

此範例中的命令與 擷取最高等待狀態的資料庫負載平均值 中的命令類似。然而，查詢 query.json 檔案有以下內容。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.query_tokenized", "Limit": 10 }
    }
]
```

以下範例使用 db.query_tokenized。

針對 Linux、macOS 或 Unix：

```
aws pi get-resource-metrics \
   --service-type DOCDB \
   --identifier db-ID \
   --start-time 2022-03-13T8:00:00Z \
   --end-time   2022-03-13T9:00:00Z \
   --period-in-seconds 3600 \
   --metric-queries file://query.json
```

針對 Windows：

```
aws pi get-resource-metrics ^
   --service-type DOCDB ^
   --identifier db-ID ^
   --start-time 2022-03-13T8:00:00Z ^
   --end-time   2022-03-13T9:00:00Z  ^
   --period-in-seconds 3600 ^
   --metric-queries file://query.json
```

此範例查詢時間超過 1 小時，間隔為一分鐘period-in-seconds

此範例會指定 的指標，db.load.avg以及前七個等待狀態GroupBy的 。如需此範例有效值的詳細資訊，請參閱績效詳情 API 參考中的DimensionGroup。

回應看起來類似以下的內容。

```
{
    "AlignedStartTime": "2022-04-04T06:00:00+00:00",
    "AlignedEndTime": "2022-04-04T06:15:00+00:00",
    "Identifier": "db-NQF3TTMFQ3GTOKIMJODMC3KQQ4",
    "MetricList": [
        {//A list of key/datapoints
            "Key": {
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                //... 60 datapoints for the total db.load.avg key
                ]
        },
               {
            "Key": {//Next key are the top tokenized queries
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.query_tokenized.db_id": "pi-1064184600",
                    "db.query_tokenized.id": "77DE8364594EXAMPLE",
                    "db.query_tokenized.statement": "{\"find\":\"customers\",\"filter
\":{\"FirstName\":\"?\"},\"sort\":{\"key\":{\"$numberInt\":\"?\"}},\"limit\"
:{\"$numberInt\":\"?\"},\"$db\":\"myDB\",\"$readPreference\":{\"mode\":\"primary\"}}"
                }
            },
            "DataPoints": [
            //... 60 datapoints
            ]
        },
        // In total 11 entries, 10 Keys of top tokenized queries, 1 total key
    ] //End of MetricList
} //End of response
```

此回應在 中有 11 個項目 MetricList（總計 1 個，最多 10 個字符化查詢），每個項目每小時 24 個DataPoints。

對於字符化查詢，每個維度清單中有三個項目：

- db.query_tokenized.statement – 字符化查詢陳述式。
- db.query_tokenized.db_id – Performance Insights 為您產生的合成 ID。此範例會傳回 pi-1064184600 合成 ID。

- db.query_tokenized.id– 績效詳情中查詢的 ID。

  在 中 AWS 管理主控台，此 ID 稱為支援 ID。其名稱如下，因為 ID 是 AWS Support 可以檢查的資料，可協助您對資料庫的問題進行故障診斷。 非常重視資料 AWS 的安全性和隱私權，而且幾乎所有資料都會以您的 加密儲存 AWS KMS key。因此，內部沒有人 AWS 可以查看此資料。在先前的範例中，tokenized.statement 和 tokenized.db_id 都同時會以加密的形式存放。如果您的資料庫發生問題， AWS Support 可以透過參考支援 ID 來協助您。

進行查詢時，在 Group 中指定 GroupBy 可能會讓您省下不少心力。然而，如需對已傳回的資料進行更精細的控制，請指定維度的清單。例如，如果所需的是 db.query_tokenized.statement，則可將 Dimensions 屬性新增至 query.json 檔案。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": {
            "Group": "db.query_tokenized",
            "Dimensions":["db.query_tokenized.statement"],
            "Limit": 10
        }
    }
]
```

擷取依查詢篩選的資料庫負載平均值

此範例中的對應 API 查詢與 擷取熱門查詢的資料庫負載平均值 中的命令類似。然而，查詢 query.json 檔案有以下內容。

```
[
 {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_state", "Limit": 5  },
        "Filter": { "db.query_tokenized.id": "AKIAIOSFODNN7EXAMPLE" }
    }
]
```

在此回應中，會根據 query.json 檔案中指定的字符化查詢 AKIAIOSFODNN7EXAMPLE 貢獻來篩選所有值。金鑰也可能遵循與沒有篩選條件的查詢不同的順序，因為這是影響篩選查詢的前五個等待狀態。

Performance Insights 的 Amazon CloudWatch 指標

績效詳情會自動將指標發佈至 Amazon CloudWatch。您可以透過績效詳情來查詢相同的資料，但在 CloudWatch 中具備這些指標可讓您輕鬆新增 CloudWatch 警示。您也可以輕鬆將指標新增至現有的 CloudWatch 儀表板。

指標	描述
DBLoad	Amazon DocumentDB 的作用中工作階段數目。您通常會需要平均作用中工作階段數量的資料。在績效詳情中，系統會以 `db.load.avg` 的形式來查詢此資料。
DBLoadCPU	等待狀態類型為 CPU 的作用中工作階段數量。在績效詳情中，此資料會查詢為 `db.load.avg` ，並依等待狀態類型 進行篩選CPU。
DBLoadNonCPU	等待狀態類型不是 CPU 的作用中工作階段數量。

> **Note**
>
> 只有當資料庫執行個體上有負載時，這些指標才會發佈至 CloudWatch。

您可以使用 CloudWatch 主控台、 AWS CLI或 CloudWatch API 來檢查這些指標。

例如，您可以執行 get-metric-statistics 命令來取得 `DBLoad` 指標的統計。

```
aws cloudwatch get-metric-statistics \
    --region ap-south-1 \
    --namespace AWS/DocDB \
    --metric-name DBLoad  \
    --period 360 \
    --statistics Average \
    --start-time 2022-03-14T8:00:00Z \
    --end-time 2022-03-14T9:00:00Z \
    --dimensions Name=DBInstanceIdentifier,Value=documentdbinstance
```

此範例會產生類似下列範例的輸出結果。

```
{
    "Datapoints": [
        {
            "Timestamp": "2022-03-14T08:42:00Z",
            "Average": 1.0,
            "Unit": "None"
        },
        {
            "Timestamp": "2022-03-14T08:24:00Z",
            "Average": 2.0,
            "Unit": "None"
        },
        {
            "Timestamp": "2022-03-14T08:54:00Z",
            "Average": 6.0,
            "Unit": "None"
        },
        {
            "Timestamp": "2022-03-14T08:36:00Z",
            "Average": 5.7,
            "Unit": "None"
        },
        {
            "Timestamp": "2022-03-14T08:06:00Z",
            "Average": 4.0,
            "Unit": "None"
        },
        {
            "Timestamp": "2022-03-14T08:00:00Z",
            "Average": 5.2,
            "Unit": "None"
        }
    ],
    "Label": "DBLoad"
}
```

您可以使用 CloudWatch 主控台中的DB_PERF_INSIGHTS指標數學函數來查詢 Amazon DocumentDB Performance Insights 計數器指標。DB_PERF_INSIGHTS 函數也包含以次分鐘為間隔的DBLoad指標。您可以根據這些指標設定 CloudWatch 警示。如需如何建立警示的詳細資訊，請參閱從 AWS 資料庫在績效詳情計數器指標上建立警示。

如需 CloudWatch 的詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的什麼是 Amazon CloudWatch？。

計數器指標的績效詳情

計數器指標是績效詳情儀表板中的作業系統指標。若要協助識別並分析效能問題，您可以將計數器指標與資料庫負載相互關聯。

績效詳情作業系統計數器

Amazon DocumentDB Performance Insights 提供下列作業系統計數器。

計數器	類型	指標
作用中	memory	os.memory.active
buffers	memory	os.memory.buffers
cached	memory	os.memory.cached
dirty	memory	os.memory.dirty
free	memory	os.memory.free
inactive	memory	os.memory.inactive
mapped	memory	os.memory.mapped
pageTables	memory	os.memory.pageTables
slab	memory	os.memory.slab
total	memory	os.memory.total
writeback	memory	os.memory.writeback
idle	cpuUtilization	os.cpuUtilization.idle
system	cpuUtilization	os.cpuUtilization.system
total	cpuUtilization	os.cpuUtilization.total

計數器	類型	指標
user	cpuUtilization	os.cpuUtilization.user
wait	cpuUtilization	os.cpuUtilization.wait
one	loadAverageMinute	os.loadAverageMinute.one
fifteen	loadAverageMinute	os.loadAverageMinute.fifteen
five	loadAverageMinute	os.loadAverageMinute.five
cached	交換	os.swap.cached
free	交換	os.swap.free
in	交換	os.swap.in
out	交換	os.swap.out
total	交換	os.swap.total
rx	網路	os.network.rx
tx	網路	os.network.tx
numVCPUs	一般	os.general.numVCPUs

使用 Amazon DocumentDB 無伺服器

Amazon DocumentDB Serverless 是一種隨需的自動擴展組態，可根據應用程式需求自動調整 Amazon DocumentDB 資料庫容量的程序。您只需為叢集使用的資源付費。因此，DocumentDB Serverless 可協助您保持在預算內，並避免支付未使用的電腦資源。

這種類型的自動化對於具有高度可變和不可預測工作負載的環境特別有用，例如多租戶資料庫、分散式資料庫，以及開發和測試系統。

主題

- DocumentDB 無伺服器使用案例
- Amazon DocumentDB 無伺服器的優點
- Amazon DocumentDB 無伺服器的運作方式
- DocumentDB 無伺服器的需求和限制
- 建立使用 Amazon DocumentDB 無伺服器的叢集
- 遷移至 Amazon DocumentDB 無伺服器
- 管理 Amazon DocumentDB 無伺服器
- Amazon DocumentDB 無伺服器執行個體限制
- Amazon DocumentDB 無伺服器擴展組態
- 監控 Amazon DocumentDB 無伺服器

DocumentDB 無伺服器使用案例

Amazon DocumentDB 佈建叢集和 DocumentDB Serverless 都支援許多類型的資料庫工作負載，從開發和測試環境，到需要大規模和可用性的最嚴苛、業務關鍵應用程式。但是 DocumentDB Serverless 為客戶工作負載新增了另一個維度，也就是支援工作負載無法預測的網站和應用程式的能力。

DocumentDB 無伺服器對於下列使用案例特別有用：

- 可變工作負載：您正在執行的工作負載活動突然且無法預測地增加。例如一個交通站點，在開始下雨時會看到活動突然增加。另一個是電子商務網站，當您提供銷售或特別促銷活動時，流量會增加。使用 DocumentDB 無伺服器，您的資料庫會自動擴展容量以滿足應用程式尖峰負載的需求，並在活動激增結束時縮減規模。使用 DocumentDB 無伺服器時，您不再需要佈建峰值或平均容量。您可以指定容量上限來處理最壞情況，除非需要，否則不會使用該容量。

  - DocumentDB Serverless 中的精細擴展可協助您將容量與資料庫的需求緊密匹配。對於佈建的叢集，向上擴展需要新增全新的執行個體。當只需要多一點容量時，DocumentDB 無伺服器可以新增一半的 DCU。它可以根據處理工作負載增加所需的額外容量，新增 0.5、1、1.5、2 或其他半 DCUs。當工作負載減少且不再需要該容量時，它可以移除 0.5、1、1.5、2 或其他半個 DCUs。
- 多租戶應用程式：使用 DocumentDB 無伺服器，您不需要個別管理機群中每個應用程式的資料庫容量。DocumentDB Serverless 會為您管理個別資料庫容量。
  - 您可以為每個租用戶建立叢集。如此一來，您就可以使用複製和快照還原等功能來增強每個租戶適用的高可用性和災難復原。
  - 根據一天中的時間、一年中的時間、促銷活動等，每個租用戶可能具有特定的繁忙和閒置時段。每個叢集都可能具有廣泛的容量範圍。如此一來，活動量低的叢集會產生最低的執行個體費用。任何叢集都可以快速擴充規模，以便處理高活動期間。
- 新應用程式：您正在部署新應用程式，且不確定所需的執行個體大小。透過使用 DocumentDB Serverless，您可以設定具有一或多個執行個體的叢集，並讓資料庫根據應用程式的容量需求自動擴展。
- 混合使用應用程式 — 假設您有線上交易處理 (OLTP) 應用程式，但您定期遇到查詢流量激增的情況。透過為叢集中的 DocumentDB 無伺服器執行個體指定提升層，您可以設定叢集，讓讀取器執行個體可以獨立於寫入器執行個體進行擴展，以處理額外的負載。當用量尖峰下降時，讀取器執行個體會縮減規模以符合寫入器執行個體的容量。
- 容量規劃 — 假設您通常透過修改叢集中所有執行個體的執行個體類別，來調整資料庫容量，或驗證工作負載的最佳資料庫容量。使用 DocumentDB 無伺服器，您可以避免此管理開銷。您可以執行工作負載並檢查執行個體實際擴展的程度，以判斷適當的最小和最大容量。
  - 您可以將現有的執行個體從佈建修改為 DocumentDB 無伺服器，或從 DocumentDB 無伺服器修改為佈建。在這種情況下，您不需要建立新的叢集或新的執行個體。
- 開發和測試 — 除了執行最嚴苛的應用程式之外，您還可以使用 DocumentDB Serverless 進行開發和測試環境。使用 DocumentDB 無伺服器，您可以建立具有低最小容量的執行個體，而不是使用爆量 db.t* 執行個體類別。您可以設定足夠高的最大容量，讓這些執行個體仍可執行大量工作負載，而不會在記憶體上執行不足。當資料庫未使用時，所有執行個體都會縮減規模，以避免不必要的費用。

針對現有的佈建工作負載使用 Amazon DocumentDB 無伺服器

假設您已在佈建叢集上執行 DocumentDB 應用程式。您可以將一或多個 DocumentDB 無伺服器執行個體新增至現有叢集做為讀取器執行個體，以檢查應用程式如何使用 DocumentDB 無伺服器。您可以檢查讀取器執行個體擴展和縮減的頻率。您可以使用 DocumentDB 容錯移轉機制，將 DocumentDB 無伺服器執行個體提升為寫入器，並檢查其如何處理讀取/寫入工作負載。如此一來，您可以用最短的停機

時間進行切換，而無需變更用戶端應用程式使用的端點。如需將現有叢集轉換為 DocumentDB 無伺服器程序的詳細資訊，請參閱 遷移至 Amazon DocumentDB 無伺服器。

Amazon DocumentDB 無伺服器的優點

DocumentDB 無伺服器適用於變數或「尖峰」工作負載。由於這種不可預測的工作負載，您可能難以規劃何時應變更資料庫容量。您也可以使用熟悉的機制快速變更容量，例如新增執行個體或變更執行個體類別。DocumentDB Serverless 提供下列優點，以協助處理這類使用案例：

- 比佈建的容量管理更簡單 — DocumentDB 無伺服器可減少在工作負載變更時規劃執行個體大小和調整執行個體大小的負擔。它還減少了為叢集中的所有執行個體維持一致容量的工作量。

- 在高活動期間更快速且更輕鬆地擴展 - DocumentDB 無伺服器可視需要擴展運算和記憶體容量，而不會中斷用戶端交易或整體工作負載。除了垂直擴展之外，將讀取器執行個體與 DocumentDB 無伺服器搭配使用還可協助您利用水平擴展。使用 DocumentDB 全域叢集的功能表示您可以將 DocumentDB 無伺服器讀取工作負載分散到多個 AWS 區域。此功能比已佈建叢集的擴展機制更方便。

- 在活動量低期間符合成本效益 — DocumentDB 無伺服器可協助您避免過度佈建執行個體。當執行個體擴展時，DocumentDB 無伺服器會以精細遞增的方式新增資源。您只需為使用的資料庫資源付費。DocumentDB 無伺服器資源用量是以每秒為單位測量。如此一來，當執行個體縮減規模時，就會立即註冊減少的資源用量。

- 已佈建的功能同位 — 您可以將所有 DocumentDB 功能與 DocumentDB 無伺服器搭配使用。例如，透過 DocumentDB 無伺服器，您可以使用讀取器執行個體、 AWS Identity and Access Management (IAM) 資料庫身分驗證和績效詳情。

  特別是，使用 DocumentDB 無伺服器時，您可以從佈建的叢集中利用下列功能：

  - 讀取器執行個體 — DocumentDB 無伺服器可以利用讀取器執行個體水平擴展。當叢集包含一或多個讀取器執行個體時，如果寫入器執行個體發生問題，叢集可能會立即容錯移轉。
  - 多可用區域叢集 — 您可以將叢集的 DocumentDB 無伺服器執行個體分散到多個可用區域 (AZs)。設定多可用區域叢集有助於確保業務持續性，即使在遇到會影響整個可用區域的極少數情況下也是如此。
  - 全域叢集 — 您可以結合 DocumentDB 全域叢集使用 DocumentDB 無伺服器，在其他 中建立叢集的其他唯讀複本 AWS 區域 ，以用於災難復原目的。

Amazon DocumentDB 無伺服器的運作方式

主題

- 概觀
- Amazon DocumentDB 叢集的組態
- Amazon DocumentDB 無伺服器擴展容量
- Amazon DocumentDB 無伺服器擴展
- 閒置狀態 (0.5 DCUs)

概觀

Amazon DocumentDB Serverless 適用於要求最嚴苛、高度可變的工作負載。例如，您的資料庫使用量可能在短時間內很大，然後是長時間的輕度活動或完全沒有活動。某些範例是具有定期促銷活動的零售網站、遊戲或體育網站，以及在需要時產生報告的資料庫。其他則為開發和測試環境，及用量可能會迅速增加的新應用程式。對於這類類似案例和許多其他情況，並不一定可以使用佈建的模型事先正確設定容量。如果您過度佈建且具有未使用的容量，也可能會產生較高的成本。

相反地，DocumentDB 佈建叢集適用於穩定的工作負載。使用佈建叢集時，您可以選擇具有預先定義記憶體數量、CPU 電源、I/O 頻寬等的執行個體類別。若您的工作負載發生變化，您可以手動修改寫入器和讀取器的執行個體類別。當您可以在預期的耗用模式之前調整容量，佈建的模型就能順利運作，並且在您變更集中寫入器和讀取器的執行個體類別時，短暫中斷是可以接受的。

DocumentDB 無伺服器從頭開始架構，以支援可立即擴展的無伺服器叢集。DocumentDB 無伺服器旨在提供與佈建寫入器和讀取器相同的安全性和隔離程度。這些方面在多租戶無伺服器雲端環境中至關重要。動態擴展機制的額外負荷很小，因此可以快速回應資料庫工作負載的變化。其功能強大，足以滿足大幅增加的處理需求。

透過使用 DocumentDB 無伺服器，您可以建立 DocumentDB 叢集，而不會鎖定每個寫入器和讀取器的特定資料庫容量。您指定最小和最大的容量範圍。DocumentDB 會在該容量範圍內擴展叢集中的每個 DocumentDB 無伺服器寫入器或讀取器。透過使用每個寫入器或讀取器都可以動態擴展的多可用區域叢集，您可以利用動態擴展和高可用性。

DocumentDB 無伺服器會根據您的最小和最大容量規格自動擴展資料庫資源。擴展速度很快，因為大多數擴展事件作業皆將寫入器或讀取器保留在同一主機上。在極少數情況下，DocumentDB 無伺服器寫入器或讀取器會從一個主機移至另一個主機，DocumentDB 無伺服器會自動管理連線。您無需變更資料庫用戶端應用程式程式碼或資料庫連線字串。

與佈建叢集一樣，使用 DocumentDB 無伺服器時，儲存容量和運算容量是分開的。當我們參考 DocumentDB 無伺服器容量和擴展時，一律會運算增加或減少的容量。因此，即使 CPU 和記憶體容量縮減規模到低等級，您的叢集也可以包含許多 TB 的資料。

您無需佈建和管理資料庫伺服器，而是指定資料庫容量。根據您的工作負載，每個 DocumentDB 無伺服器寫入器或讀取器的實際容量會隨著時間而不同。如需有關該機制的詳細資料，請參閱 Amazon DocumentDB 無伺服器擴展。

Amazon DocumentDB 叢集的組態

對於每個 Amazon DocumentDB 叢集，您可以選擇 DocumentDB 無伺服器容量、佈建容量或兩者的任意組合。

您可以設定包含 DocumentDB 無伺服器和佈建容量的叢集，稱為混合組態叢集。例如，假設您需要比 DocumentDB 無伺服器寫入器更多的讀取/寫入容量。在這種情況下，您可以使用非常大的已佈建寫入器來設定叢集。然後，您仍然可以將 DocumentDB 無伺服器用於讀取器。或者假設叢集的寫入工作負載有變化，但讀取工作負載穩定。在此情況下，您可以使用 DocumentDB 無伺服器寫入器和一或多個佈建讀取器來設定叢集。

您也可以設定叢集，其中所有容量都由 DocumentDB Serverless 管理。若要這樣做，您可以建立新的叢集，並從頭開始使用 DocumentDB 無伺服器。或者，您可以將現有叢集中的所有佈建容量取代為 DocumentDB 無伺服器。如需使用 DocumentDB 無伺服器建立新叢集或將現有叢集切換至 DocumentDB 無伺服器的程序，請參閱 建立使用 Amazon DocumentDB 無伺服器的叢集和 遷移至 Amazon DocumentDB 無伺服器。

如果您在叢集中完全不使用 DocumentDB 無伺服器，則會佈建叢集中的所有寫入器和讀取器。這是大多數使用者熟悉的最常見叢集類型。已佈建容量是常數。費用相對容易預測。但是，您必須提前預測您需要多少容量。在某些情況下，您的預測可能不準確，或者您的容量需求可能會發生變化。在這些情況下，您的叢集可能會變得佈建不足（比您想要的還慢）或過度佈建（比您想要的更昂貴）。

Amazon DocumentDB 無伺服器擴展容量

Amazon DocumentDB Serverless 的度量單位是 DocumentDB 容量單位 (DCU)。DocumentDB 無伺服器擴展容量與您用於佈建叢集的執行個體類別無關。

每個 DCU 都是大約 2 GB (GiB) 記憶體、對應 CPU 和聯網的組合。您可以使用此度量單位指定資料庫容量範圍。`ServerlessDatabaseCapacity` 和 `DCUUtilization` CloudWatch 指標可協助您判斷資料庫實際使用的容量，以及該容量落在指定範圍內的位置。

在任何時間，每個 DocumentDB 無伺服器寫入器或讀取器都有容量。容量是代表 DCUs浮點數。每當寫入器或讀取器擴展時，容量都會增加或減少。此值每秒測量一次。對於您要使用 DocumentDB 無伺服器的每個叢集，您可以定義容量範圍：每個 DocumentDB 無伺服器寫入器或讀取器可以擴展的最小和最大容量值。叢集中每個 DocumentDB 無伺服器寫入器或讀取器的容量範圍都相同。每個 DocumentDB 無伺服器寫入器或讀取器都有自己的容量，落在該範圍內。

DocumentDB 5.0 和 8.0 支援 DocumentDB 無伺服器，容量範圍為 0.5 - 256 DCUs。

您可以定義的最小 DocumentDB 無伺服器容量為 0.5 DCUs。如果小於或等於支援的最大容量值，您可以指定較高的數字。將最小容量設定為少量，可讓輕度載入的叢集使用最少的運算資源。同時，它們隨時準備好立即接受連線，並在忙碌時擴充規模。

將最小值設定為允許每個寫入器或讀取器在緩衝集區中保留應用程式工作集的值。這樣，緩衝集區的內容在閒置期間不會被丟棄。如需選擇擴展容量範圍時的所有考量，請參閱 選擇 DocumentDB 無伺服器叢集的擴展容量範圍。

根據您在多可用區域部署中設定讀取器的方式，讀取器的容量可以繫結至寫入器的容量，也可以獨立地調整。如需有關如何執行此作業的詳細資訊，請參閱 檢視和修改無伺服器讀取器的提升層。

監控 DocumentDB 無伺服器涉及測量叢集中寫入器和讀取器隨時間的容量值。若資料庫未縮減規模至最小容量，則可採取諸如調整最小容量和最佳化資料庫應用程式等作業。如果資料庫持續達到其最大容量，則可採取諸如增加最大值之類的動作。您還可以最佳化資料庫應用程式，並將查詢負載分佈到更多讀取器中。

DocumentDB 無伺服器容量的費用是以 DCU 小時為單位來衡量。如需如何計算 DocumentDB 無伺服器費用的資訊，請參閱 Amazon DocumentDB 定價。假設叢集中的寫入器和讀取器總數為 n。在這種情況下，當您未執行任何資料庫操作時，叢集會耗用大約 n x 的DCUs。Amazon DocumentDB 本身可能會執行導致少量負載的監控或維護操作。當資料庫以完整容量執行時，該叢集會耗用不超過 n x 個最大 DCUs。

如需選擇適當最小和最大 DCU 值的詳細資訊，請參閱 選擇 DocumentDB 無伺服器叢集的擴展容量範圍。您指定的最小和最大 DCU 值也會影響某些 Amazon DocumentDB 執行個體限制。如需容量範圍與執行個體限制之間互動的詳細資訊，請參閱 Amazon DocumentDB 無伺服器執行個體限制。

Amazon DocumentDB 無伺服器擴展

對於每個 DocumentDB 無伺服器寫入器或讀取器，Amazon DocumentDB 會持續追蹤 CPU、記憶體和網路等資源的使用率。這些測量統稱為負載。負載包括應用程式執行的資料庫作業。它還包括資料庫伺服器和 Amazon DocumentDB 管理任務的背景處理。當容量受到這些限制時，DocumentDB 無伺服器會擴展。DocumentDB 無伺服器也會在偵測到效能問題時向上擴展，因為這樣可以解決。您可以

使用 中的程序，監控資源使用率及其如何影響 DocumentDB 無伺服器擴展監控 Amazon DocumentDB 無伺服器。

負載可能因叢集中的寫入器和讀取器而有所不同。寫入器會處理寫入操作，並對叢集磁碟區執行所有資料修改。讀者可以處理唯讀請求。

擴展是增加或減少資料庫 DocumentDB 無伺服器容量的操作。使用 DocumentDB 無伺服器時，每個寫入器和讀取器都有自己的目前容量值，以 DCUs測量。當寫入器或讀取器目前的容量太低而無法處理負載時，DocumentDB 無伺服器會將寫入器或讀取器擴展到更高的容量。當其目前容量高於需要時，它會將寫入器或讀取器縮減到更低的容量。

DocumentDB 無伺服器可以增量增加容量。當您的工作負載需求開始達到寫入器或讀取器的目前資料庫容量時，DocumentDB 無伺服器會增加該寫入器或讀取器DCUs 數量。DocumentDB 無伺服器會以所需的增量擴展容量，為消耗的資源提供最佳效能。擴展會以小至 0.5 DCUs的增量進行。目前容量越大，擴展增量就越大，因此可以發生更快的擴展。

由於 DocumentDB 無伺服器擴展非常頻繁、精細且不中斷，因此不會在 中造成離散事件 AWS 管理主控台。反之，您可以測量 Amazon CloudWatch 指標，例如 `serverlessDatabaseCapacity`和 `DCUUtilization`，並追蹤其隨時間的最小值、最大值和平均值。若要進一步了解如何監控 DocumentDB 無伺服器，請參閱 監控 Amazon DocumentDB 無伺服器。

向上或向下擴展可能由下列原因造成：

- 記憶體使用率
- CPU 使用率
- 網路使用率
- 儲存使用率

您可以監控這些在 DocumentDB 無伺服器執行個體上向上/向下擴展的原因。如需詳細資訊，請參閱監控 Amazon DocumentDB 無伺服器。

您可以選擇與關聯的寫入器同時擴展讀取器，也可以獨立於寫入器進行擴展。您可以透過為該讀取器指定提升層來執行此操作。

- DocumentDB 無伺服器讀取器在提升層 0 和 1 中，與寫入器同時擴展。這種擴展行為使優先順序層 0 和 1 中的讀取器成為可用性的理想選擇。這是因為它們總是調整到適當的容量，以便在發生容錯移轉時從寫入器處接管工作負載。
- 提升層 2 至 15級中的讀取器可獨立於寫入器擴展。每個讀取器都會保持在您為叢集指定的最小和最大 DCU 值內。當讀取器獨立於關聯的寫入器資料庫進行擴展時，它可能會處於閒置狀態並縮減規

模，而寫入器繼續處理大量交易。如果在較低的提升層中沒有其他讀取器可用，則它仍可用作容錯移轉目標。但是，如果將其提升為寫入器，則可能需要擴充規模以處理寫入器的全部工作負載。

如需檢視和變更無伺服器執行個體提升方案的詳細資訊，請參閱 檢視和修改無伺服器讀取器的提升層。

當資料庫連線開啟、交易正在進行等時，可能會發生 DocumentDB 無伺服器擴展。DocumentDB 無伺服器不會等待安靜點開始擴展。擴展不會中斷正在進行的任何資料庫作業。

如果您的工作負載需要的讀取容量超過單一寫入器和單一讀取器的可用容量，您可以將多個 DocumentDB 無伺服器讀取器新增至叢集。每個 DocumentDB 無伺服器讀取器可以在您為叢集指定的最小和最大容量值範圍內進行擴展。您可以使用叢集的讀取器端點，將唯讀工作階段導向讀取器，並減少寫入器的負載。

DocumentDB 無伺服器是否執行擴展，以及擴展開始後的速度有多快，也取決於叢集的最小和最大 DCU 設定。此外，這取決於讀取器是設定為與寫入器一起擴展還是獨立擴展。如需擴展組態的詳細資訊，請參閱 Amazon DocumentDB 無伺服器擴展組態。

閒置狀態 (0.5 DCUs)

當 Amazon DocumentDB 無伺服器寫入器或讀取器閒置時，如果叢集的 MinCapacity 設定為 0.5，則 DocumentDB 無伺服器執行個體支援縮減至閒置狀態 0.5 DCUs。

在閒置狀態下，DocumentDB 無伺服器執行個體沒有足夠的 CPU 運算容量來支援大多數生產工作負載，但已準備好快速擴展以支援新的工作負載。在非閒置狀態下，DocumentDB 無伺服器執行個體通常至少需要 1.0 - 2.5 DCUs。因此，當 DocumentDB 無伺服器執行個體從閒置狀態擴展到非閒置狀態時，它們會直接擴展到 1.0 - 2.5 DCUs（如果較低，則為 MaxCapacity 的值）。

為了在閒置時支援縮減至 0.5 DCUs，如果 MinCapacity 設定為小於或等於 1.0 DCUs執行個體限制會受到限制。如需 MinCapacity 組態如何影響限制的詳細資訊，請參閱 Amazon DocumentDB 無伺服器執行個體限制。

DocumentDB 無伺服器的需求和限制

功能需求

區域可用性

如需 Amazon DocumentDB 無伺服器執行個體類型可用 AWS 區域 位置的詳細資訊，請參閱 中的無伺服器欄各區域的支援執行個體類別。

下列 AWS CLI 命令可用來驗證特定區域中提供的確切 DocumentDB 無伺服器執行個體選項：

```
aws docdb describe-orderable-db-instance-options \
        --region my_region \
        --db-instance-class db.serverless \
        --engine docdb
```

引擎版本可用性

Amazon DocumentDB 5.0 和 8.0 引擎版本支援 DocumentDB 無伺服器。 Amazon DocumentDB 它不適用於引擎 3.6 或 4.0 版。

> **Note**
>
> 僅較新的 5.0.0 修補程式版本支援 DocumentDB 無伺服器。確保您的叢集已更新為最新的引擎修補程式版本。如需維護 Amazon DocumentDB 叢集的詳細資訊，請參閱 維護 Amazon DocumentDB

叢集組態

在將任何 Amazon DocumentDB 無伺服器執行個體新增至 Amazon DocumentDB 叢集之前，叢集也必須設定 `ServerlessV2ScalingConfiguration` 參數。這會定義叢集中 DocumentDB 無伺服器執行個體的容量範圍。如需擴展組態的詳細資訊，請參閱 Amazon DocumentDB 無伺服器擴展組態。

特定 Amazon DocumentDB 功能的最小擴展容量範圍設定

有些 Amazon DocumentDB 功能可與 DocumentDB 無伺服器搭配使用，但如果您的容量範圍低於特定工作負載之這些功能的記憶體需求，可能會導致問題。於該種情況下，您的資料庫可能無法像往常一樣執行，或者可能會遇到記憶體不足錯誤。

下列功能需要設定更高的 `MinCapacity`和/或 `MaxCapacity`值，才能獲得最佳操作：

- Performance Insights
- 全域叢集 （僅適用於主要叢集 AWS 區域)
- 在具有大型資料磁碟區的叢集上建立無伺服器執行個體

  這包括建立無伺服器執行個體做為叢集還原的一部分。

如需設定適當容量範圍的建議（如果您使用此功能），請參閱 選擇 DocumentDB 無伺服器叢集的擴展容量範圍。有關資料庫因容量範圍錯誤設定而造成記憶體不足錯誤時的疑難排解資訊，請參閱 避免記憶體不足錯誤。

建立使用 Amazon DocumentDB 無伺服器的叢集

建立 Amazon DocumentDB 無伺服器叢集

使用 Amazon DocumentDB 無伺服器時，您的叢集可與佈建的叢集互換。您可以擁有叢集，其中有些執行個體使用無伺服器，而有些執行個體已佈建。

確認所需的區域和引擎版本支援 DocumentDB 無伺服器。請參閱 DocumentDB 無伺服器的需求和限制。

若要在其中新增無伺服器執行個體的 Amazon DocumentDB 叢集，請遵循與 相同的程序建立 Amazon DocumentDB 叢集。唯一的差別是也必須提供 `ServerlessV2ScalingConfiguration`引數。

`ServerlessV2ScalingConfiguration` 引數指定 DocumentDB 無伺服器執行個體的擴展容量範圍。它包含套用至叢集中所有 DocumentDB 無伺服器執行個體的最小和最大 DocumentDB 容量單位 (DCU) 值：

- `MinCapacity` 值指定最小擴展容量。
- `MaxCapacity` 值指定最大擴展容量。

如需擴展的詳細資訊，請參閱Amazon DocumentDB 無伺服器擴展組態。

Using the AWS 管理主控台

下列 AWS 管理主控台 組態範例示範如何建立 DocumentDB 無伺服器叢集。

1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

> **Tip**
>
> 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

叢集資料表隨即出現。

3. 選擇建立。

   建立 Amazon DocumentDB 叢集頁面隨即出現。

4. 在建立 Amazon DocumentDB 叢集頁面上的叢集類型區段中，選擇執行個體型叢集（這是預設選項）。

5. 在叢集組態區段中：

   a. 針對叢集識別符，輸入唯一名稱，例如 **myserverlesscluster**。請注意，無論如何輸入叢集名稱，主控台都會將所有叢集名稱變更為小寫。

   b. 針對引擎版本，選擇 5.0.0（這是預設選項）。

6. 在叢集儲存組態區段中，選擇 Amazon DocumentDB Standard（這是預設選項）。

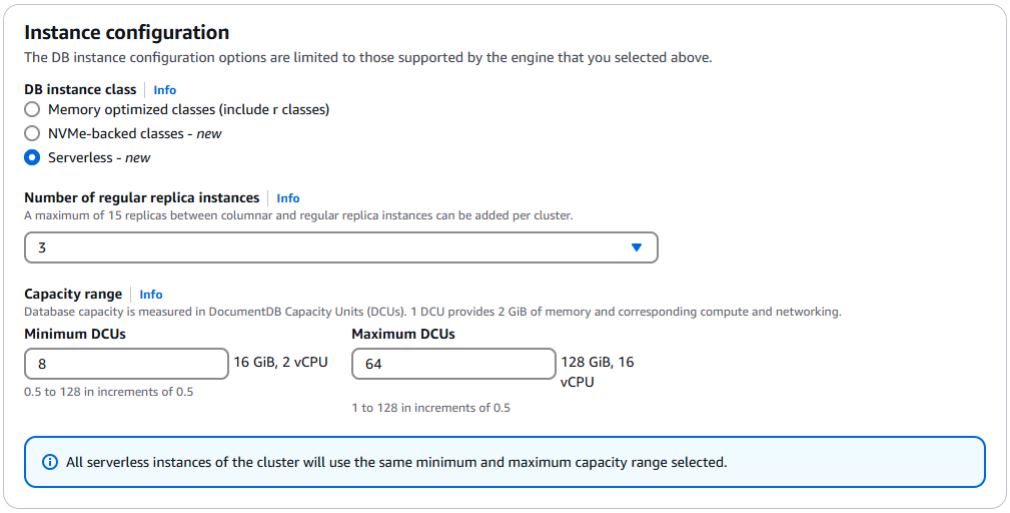
   > **Note**
   >
   > 此類別中的另一個選項是 Amazon DocumentDB I/O 最佳化。若要進一步了解任一選項，請參閱 Amazon DocumentDB 叢集儲存組態

7. 在執行個體組態區段中：

   a. 選擇無伺服器資料庫執行個體類別。

   b. 針對一般複本執行個體的數量，選擇 3（這是預設選項）。

   c. 在容量範圍區段中，保留最小值 DCUs和最大值 DCUs的預設值。如需設定這些參數的資訊，請參閱 Amazon DocumentDB 無伺服器執行個體限制。



8. 在連線區段中，保留不連線至 EC2 運算資源的預設設定。
9. 在身分驗證區段中，輸入主要使用者的使用者名稱，然後選擇自我管理。輸入密碼，然後進行確認。

   如果您改為選擇 中的受管 AWS Secrets Manager，請參閱 使用 Amazon DocumentDB 和 進行密碼管理 AWS Secrets Manager 以取得詳細資訊。
10. 將所有其他選項保留為預設值，然後選擇建立叢集。

Using the AWS CLI

在下列範例中，將每個*#########*取代為您自己的資訊或組態參數。

若要使用 建立與 DocumentDB 無伺服器執行個體相容的叢集 AWS CLI，請參閱 使用 建立叢集 AWS CLI。

在`create-db-cluster`命令中包含下列其他參數：

```
--serverless-v2-scaling-configuration
     MinCapacity=minimum_capacity,MaxCapacity=maximum_capacity
```

範例：

```
aws docdb create-db-cluster \
      --db-cluster-identifier sample-cluster \
      --engine docdb \
```

```
    --engine-version 5.0.0 \
    --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16 \
    --master-username user-name \
    --master-user-password password
```

新增 Amazon DocumentDB 無伺服器執行個體

若要新增 DocumentDB 無伺服器執行個體，請遵循 中的相同程序將 Amazon DocumentDB 執行個體新增至叢集，務必將 db.serverless 指定為執行個體類別。

使用 新增無伺服器執行個體 AWS 管理主控台。

若要使用主控台新增 DocumentDB 無伺服器執行個體，請參閱將 Amazon DocumentDB 執行個體新增至叢集並選擇使用 AWS 管理主控台標籤。

使用 新增無伺服器執行個體 AWS CLI

若要使用 新增 DocumentDB 無伺服器執行個體 AWS CLI，請參閱將 Amazon DocumentDB 執行個體新增至叢集並選擇使用 AWS CLI標籤。

使用下列執行個體類別 CLI 引數：

```
--db-instance-class db.serverless
```

範例：

```
aws docdb create-db-instance \
      --db-cluster-identifier sample-cluster \
      --db-instance-identifier sample-instance \
      --db-instance-class db.serverless \
      --engine docdb
```

遷移至 Amazon DocumentDB 無伺服器

主題

- 將現有的 DocumentDB 叢集遷移至無伺服器
- 從 MongoDB 遷移至 DocumentDB 無伺服器

將現有的 DocumentDB 叢集遷移至無伺服器

升級叢集的引擎版本

如果您的佈建叢集執行的是不支援 DocumentDB Serverless 的較低引擎版本，您必須先將叢集升級至支援的引擎版本。如需詳細資訊，請參閱Amazon DocumentDB 就地主要版本升級。

將佈建叢集遷移至 DocumentDB 無伺服器

若要切換佈建叢集以使用 DocumentDB 無伺服器，請遵循下列步驟：

1. 檢查佈建叢集的引擎版本是否需要升級，才能與 DocumentDB Serverless 搭配使用。請參閱 DocumentDB 無伺服器的需求和限制。

   > **Note**
   >
   > 如果佈建的叢集執行的引擎版本不適用於 DocumentDB Serverless，請升級叢集的引擎版本。請參閱 Amazon DocumentDB 就地主要版本升級。

2. 設定叢集的擴展組態。如需選擇擴展組態的詳細資訊，請參閱 選擇 DocumentDB 無伺服器叢集的擴展容量範圍。若要修改叢集的擴展組態，請參閱 檢視和修改叢集的擴展容量範圍組態。
3. 設定任何其他叢集屬性，以考量來自 的 DocumentDB 無伺服器需求和限制DocumentDB 無伺服器的需求和限制。
4. 將一或多個 DocumentDB 無伺服器執行個體新增至叢集。請遵循 新增 Amazon DocumentDB 無伺服器執行個體 中的程序。

   > **Note**
   >
   > 在某些情況下，您可能已在叢集中擁有一或多個佈建的讀取器執行個體。如果是這樣，您可以選擇將其中一個讀取器轉換為 DocumentDB 無伺服器執行個體，而不是建立新的執行個體。若要執行此作業，請依照 變更執行個體的類別 中的程序進行。

5. （選用）執行容錯移轉操作，讓 DocumentDB 無伺服器執行個體成為叢集寫入器。請參閱 Amazon DocumentDB 容錯移轉。
6. （選用）將任何剩餘的佈建 Amazon DocumentDB 執行個體轉換為 DocumentDB 無伺服器執行個體（請參閱 變更執行個體的類別），或從叢集中移除執行個體（請參閱 刪除 Amazon DocumentDB 執行個體）。

Using the AWS 管理主控台

下列 AWS 管理主控台 組態範例顯示使用執行 Amazon DocumentDB 5.0.0 的 Amazon DocumentDB 佈建叢集的遷移程序，這不需要引擎版本升級，即可開始使用 DocumentDB 無伺服器。叢集名為 `sample`，開頭為三個名為 `sample`、 `sample2`和 的佈建執行個體`sample3`。在此範例中，這三個執行個體將被三個無伺服器執行個體取代。

1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

   叢集資料表隨即出現。
3. 在叢集表格中，選取您要新增無伺服器執行個體之叢集的核取方塊。
4. 選擇 Actions (動作)，然後選擇 Add instances (新增執行個體)。
5. 在新增執行個體對話方塊中，針對您要建立的每個新無伺服器執行個體，在資料庫執行個體類別區段中選取無伺服器。
6. 對於無伺服器容量設定，請根據對話方塊中的容量描述來設定擴展組態。
7. （選用）若要新增另一個執行個體，請選擇新增執行個體。繼續新增執行個體，直到您達到所需的新執行個體數量為止。

   在此範例中，會建立三個新的無伺服器執行個體。
8. 選擇建立。

   建立執行個體需要幾分鐘才能完成。您可以使用 主控台或 AWS CLI 檢視執行個體的狀態。如需詳細資訊，請參閱監控 Amazon DocumentDB 叢集的狀態。
9. 返回叢集資料表，選取所有三個原始佈建執行個體的核取方塊。
10. 選擇動作，然後選擇刪除。

在刪除期間，會自動執行容錯移轉，將剩餘的其中一個執行個體提升為寫入器。幾分鐘後，刪除程序即完成。現有叢集現在包含三個 DocumentDB 無伺服器執行個體 （如大小欄所定義）。

Using the AWS CLI

下列 AWS CLI 組態範例顯示使用執行 Amazon DocumentDB 5.0.0 的 Amazon DocumentDB 佈建叢集的遷移程序，這不需要引擎版本升級，即可開始使用 DocumentDB 無伺服器。叢集名為 sample，開頭為三個名為 sample、 sample2和 的佈建執行個體sample3。在此範例中，這三個執行個體將被三個無伺服器執行個體取代。叢集名為 sample-cluster，開頭為兩個名為 sample-provisioned-instance-1和 的佈建執行個體sample-provisioned-instance-2，一個寫入器執行個體和一個讀取器執行個體。

在下列範例中，將每個*#########*取代為您自己的資訊或組態參數。

使用 aws docdb describe-db-clusters操作來判斷叢集的狀態。下列程式碼會尋找叢集的狀態，sample-cluster並將結果輸出至資料表：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterMembers' \
    --output table
```

```
----------------------------------------------------------------------------------------------
|                                          DescribeDBClusters
                  |
+---------------------------------+---------------------------------
+------------------+----------------+
|  DBClusterParameterGroupStatus |      DBInstanceIdentifier       | IsClusterWriter
  | PromotionTier  |
+---------------------------------+---------------------------------
+------------------+----------------+
|  in-sync                       |  sample-provisioned-instance-2  |  False
  |  1             |
|  in-sync                       |  sample-provisioned-instance-1  |  True
  |  1             |
+---------------------------------+---------------------------------
+------------------+----------------+
```

新增叢集的擴展組態：

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16
```

新增無伺服器執行個體。在此範例中，sample-serverless-instance-2會新增名為 sample-serverless-instance-1和 的新無伺服器執行個體：

```
aws docdb create-db-instance \
    --db-cluster-identifier sample-cluster \
    --db-instance-identifier sample-serverless-instance-1 \
    --db-instance-class db.serverless \
    --engine docdb

aws docdb create-db-instance \
    --db-cluster-identifier sample-cluster \
    --db-instance-identifier sample-serverless-instance-2 \
    --db-instance-class db.serverless \
    --engine docdb
```

輸入以下內容以等待無伺服器執行個體可用，然後再繼續：

```
aws docdb wait db-instance-available \
    --db-instance-identifier sample-serverless-instance-1

aws docdb wait db-instance-available \
    --db-instance-identifier sample-serverless-instance-2
```

執行容錯移轉，讓新sample-serverless-instance-1執行個體成為叢集寫入器：

```
aws docdb failover-db-cluster \
    --db-cluster-identifier sample-cluster \
    --target-db-instance-identifier sample-serverless-instance-1
```

容錯移轉需要幾秒鐘才能完成，之後 sample-serverless-instance-1 會成為叢集寫入器。使用以下輸入進行驗證：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterMembers' \
    --output table
```

```
-------------------------------------------------------------------------------------------
|                                        DescribeDBClusters
                  |
```

```
+---------------------------------+---------------------------------
+------------------+----------------+
|  DBClusterParameterGroupStatus |      DBInstanceIdentifier       | IsClusterWriter
  | PromotionTier  |
+---------------------------------+---------------------------------
+------------------+----------------+
|  in-sync                       |  sample-provisioned-instance-2  |  False
  |  1             |
|  in-sync                       |  sample-provisioned-instance-1  |  False
  |  1             |
|  in-sync                       |  sample-serverless-instance-2   |  False
  |  1             |
|  in-sync                       |  sample-serverless-instance-1   |  True
  |  1             |
+---------------------------------+---------------------------------
+------------------+----------------+
```

最後，刪除原始佈建的執行個體：

```
aws docdb delete-db-instance \
    --db-instance-identifier sample-provisioned-instance-1

aws docdb delete-db-instance \
    --db-instance-identifier sample-provisioned-instance-2
```

從 MongoDB 遷移至 DocumentDB 無伺服器

您可以將 MongoDB 資料庫遷移至 DocumentDB 無伺服器，就像佈建的 Amazon DocumentDB 一樣。如需詳細資訊，請參閱遷移和升級 Amazon DocumentDB。

管理 Amazon DocumentDB 無伺服器

檢視和修改叢集的擴展容量範圍組態

`ServerlessV2ScalingConfiguration` 引數指定 DocumentDB 無伺服器執行個體的擴展容量範圍。它由套用至叢集中所有 DocumentDB 無伺服器執行個體的最小和最大 DocumentDB 容量單位 (DCU) 值組成。

- **`MinCapacity`** — 叢集中任何 DocumentDB 無伺服器執行個體的最低擴展容量。
- **`MaxCapacity`** — 叢集中任何 DocumentDB 無伺服器執行個體的最大擴展容量。

> **Note**
>
> 下列擴展組態修改需要重新啟動執行個體，以反映新的 MinCapacity和 MaxCapacity值：
>
> - MaxCapacity 值的任何變更
> - 從較高的MinCapacity值將值變更為 1.0 或更低
> - 從較低的MinCapacity值將值變更為大於 1.0

如需擴展組態以及如何選取適當擴展容量限制的詳細資訊，請參閱Amazon DocumentDB 無伺服器擴展組態。

Using the AWS 管理主控台

下列 AWS 管理主控台 組態範例示範如何檢視和編輯 DocumentDB 無伺服器叢集的擴展組態設定。

1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

   叢集資料表隨即出現。
3. 在叢集表格中，選取您要修改擴展容量之叢集的核取方塊。
4. 選擇 Actions (動作)，然後選擇 Modify (修改)。

   修改叢集對話方塊隨即出現。
5. 找到無伺服器容量設定區段，並根據對話方塊中的容量描述設定擴展組態 (容量範圍)。

   如需擴展和容量範圍的詳細資訊，請參閱Amazon DocumentDB 無伺服器擴展組態。
6. 選擇 Continue (繼續)。
7. 在 Scheduling of modifications (修改排程) 中，選擇 Apply immediately (立即套用)。
8. 選擇修改叢集。

9. 修改完成後，每個無伺服器執行個體都應重新啟動。若要將寫入器無法使用降至最低，請執行下列操作序列：

   a. 重新啟動每個無伺服器讀取器執行個體。

      i. 選取讀取器執行個體，選擇動作，然後選擇重新啟動。

      ii. 等待執行個體狀態回到可用。

   b. 執行容錯移轉至重新啟動的無伺服器執行個體。

      i. 選取叢集，選擇動作，然後選擇容錯移轉。

      ii. 等待容錯移轉操作完成。

   c. 重新啟動剩餘的無伺服器執行個體。

      i. 選取剩餘的執行個體，選擇動作，然後選擇重新啟動。

      ii. 等待執行個體狀態回到可用。

Using the AWS CLI

下列 AWS CLI 組態範例顯示目前的擴展組態。

在下列範例中，將每個*#########*取代為您自己的資訊或組態參數。

您可以使用 `describe-db-clusters` AWS CLI 命令來檢視叢集目前的擴展組態：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[0].ServerlessV2ScalingConfiguration'
```

以下是此命令的輸出：

```
{
    "MinCapacity": 0.5,
    "MaxCapacity": 16.0
}
```

您可以使用 `modify-db-cluster`命令修改叢集的擴展組態：

```
aws docdb modify-db-cluster \
```

```
    --db-cluster-identifier sample-cluster \
    --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=32
```

完成後，每個無伺服器執行個體都應該重新啟動。為了盡量減少寫入器無法使用，我們可以執行下列操作序列：

```
aws docdb reboot-db-instance \
    --db-instance-identifier sample-serverless-instance-reader

aws docdb wait db-instance-available \
    --db-instance-identifier sample-serverless-instance-reader

aws docdb failover-db-cluster \
   --db-cluster-identifier sample-cluster \
   --target-db-instance-identifier sample-serverless-instance-reader

aws docdb reboot-db-instance \
    --db-instance-identifier sample-serverless-instance-writer

aws docdb wait db-instance-available \
    --db-instance-identifier sample-serverless-instance-writer

aws docdb failover-db-cluster \
   --db-cluster-identifier sample-cluster \
   --target-db-instance-identifier sample-serverless-instance-writer
```

檢視和修改無伺服器讀取器的提升層

對於包含多個 DocumentDB 無伺服器執行個體或佈建和無伺服器執行個體混合的叢集，請注意每個無伺服器執行個體的提升層設定。此設定控制無伺服器執行個體比佈建執行個體的行為更多。

對於佈建的執行個體，選擇層 0–15 只會決定 Amazon DocumentDB 在容錯移轉操作期間選擇要提升寫入器的讀取器執行個體順序。不過，對於無伺服器執行個體，層編號也會決定執行個體是否擴展以符合寫入器執行個體的容量，還是根據自己的工作負載獨立擴展。第 0 層或第 1 層中的無伺服器讀取器執行個體，其容量至少與寫入器執行個體一樣高。如此一來，它們就可以在容錯移轉時從寫入器執行個體接管。如果寫入器執行個體是佈建執行個體，Amazon DocumentDB 會估計同等的 DocumentDB 無伺服器容量。它使用該估計值作為無伺服器讀取器執行個體的最低容量。

第 2-15 層中的 DocumentDB 無伺服器讀取器執行個體在其容量下限上沒有相同的限制。當閒置時，他們可以縮減至叢集容量範圍中指定的 DocumentDB 容量單位 (DCU) 最小值。

Using the AWS 管理主控台

下列 AWS 管理主控台 組態範例示範如何檢視和修改 DocumentDB 無伺服器執行個體讀取器的促銷層設定。

1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。

   每個執行個體的提升方案會顯示在 的提升方案欄中 AWS 管理主控台。
2. 在導覽窗格中，選擇叢集。

   > **Tip**
   >
   > 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示
   > (≡)。

   叢集資料表隨即出現。
3. 在叢集表格中，選取您要修改提升層之執行個體的核取方塊。
4. 選擇 Actions (動作)，然後選擇 Modify (修改)。

   修改執行個體對話方塊隨即出現。
5. 找到容錯移轉區段，並將提升層設定為所需的層級。
6. 選擇 Continue (繼續)。
7. 在 Scheduling of modifications (修改排程) 中，選擇 Apply immediately (立即套用)。
8. 選擇 Modify instance (修改執行個體)。

Using the AWS CLI

下列 AWS CLI 組態範例顯示目前的擴展組態。

在下列範例中，將每個*#########*取代為您自己的資訊或組態參數。

您可以使用 `describe-db-clusters` AWS CLI 命令來檢視叢集中所有執行個體的提升方案：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[0].DBClusterMembers' \
    --output table
```

以下是此命令的輸出：

```
-----------------------------------------------------------------------------------------------------
|                                          DescribeDBClusters
                  |
+---------------------------------+---------------------------------
+------------------+----------------+
|  DBClusterParameterGroupStatus |      DBInstanceIdentifier       | IsClusterWriter
  | PromotionTier  |
+---------------------------------+---------------------------------
+------------------+----------------+
|  in-sync                        |  sample-serverless-instance-2   |  False
  |  1             |
|  in-sync                        |  sample-serverless-instance-1   |  True
  |  1             |
+---------------------------------+---------------------------------
+------------------+----------------+
```

您可以使用 modify-db-instance命令修改特定執行個體的提升層：

```
aws docdb modify-db-instance \
    --db-instance-identifier sample-serverless-instance-2 \
    --promotion-tier 3
```

Amazon DocumentDB 無伺服器執行個體限制

對於 DocumentDB 無伺服器執行個體，每個執行個體的下列限制取決於執行個體目前的擴展容量：

- 執行個體記憶體 (GiB)
- 連線 （ 全部 ）
- 游標限制
- 開啟交易
- 連線 （ 作用中 ）

下表說明無伺服器執行個體的每個執行個體限制如何相對於執行個體目前的擴展容量進行擴展。

請注意，限制值不同，取決於叢集的MinCapacity擴展組態是否已設定為大於兩 (2) 個。如果 MinCapacity 設定為小於或等於 2，則游標限制、開啟交易和連線 （ 作用中 ） 的限制值會限制為

較低的最大值。（上限的技術原因是支援縮減至「閒置」狀態。如需詳細資訊，請參閱閒置狀態 (0.5 DCUs)。

db.serverless 執行個體會在 MinCapacity ≤1 時限制

目前容量 (DCU)	執行個體記憶體 (GiB)	連線（全部）	游標限制	開啟交易	連線（作用中）
0.5	1	250	6	12	39
1	2	500	12	24	79
2	4	1000	24	48	173
4	8	2000	48	96	416
8	16	4000	96	192	1071
16	32	8000	132	264	1550
32	64	16000	132	264	1550
64	128	32000	132	264	1550
128	256	60000	132	264	1550
256	512	60000	132	264	1550

db.serverless 執行個體會在 MinCapacity >1 時限制

目前容量 (DCU)	執行個體記憶體 (GiB)	連線（全部）	游標限制	開啟交易	連線（作用中）
1.5	3	750	18	36	124
2	4	1000	24	48	173
4	8	2000	48	96	416
8	16	4000	96	192	1071

目前容量 (DCU)	執行個體記憶體 (GiB)	連線（全部）	游標限制	開啟交易	連線（作用中）
16	32	8000	192	384	2709
32	64	16000	384	768	4500
64	128	32000	768	1536	4500
128	256	60000	1536	3072	4500
256	512	60000	3072	6144	4500

您可以使用下列 CloudWatch 指標，監控和警示每個執行個體限制的 。如需 Amazon DocumentDB CloudWatch 指標的詳細資訊，請參閱使用 CloudWatch 監控 Amazon DocumentDB。

資源	CloudWatch 限制指標	CloudWatch 用量指標（最多 1 分鐘）	CloudWatch 用量指標
執行個體記憶體	-	-	FreeableMemory
連線（全部）	DatabaseConnectionsLimit	DatabaseConnectionsMax	DatabaseConnections
游標	DatabaseCursorsLimit	DatabaseCursorsMax	DatabaseCursors
交易	TransactionsOpenLimit	TransactionsOpenMax	TransactionsOpen

Amazon DocumentDB 無伺服器擴展組態

主題

- 選擇 DocumentDB 無伺服器叢集的擴展容量範圍
- 選擇 DocumentDB 無伺服器叢集MinCapacity的設定
- 選擇 DocumentDB 無伺服器叢集MaxCapacity的設定
- 避免記憶體不足錯誤

- 為什麼我的無伺服器執行個體沒有縮減規模？

選擇 DocumentDB 無伺服器叢集的擴展容量範圍

將任何 DocumentDB 無伺服器執行個體新增至 Amazon DocumentDB 叢集之前，叢集也必須設定 `ServerlessV2ScalingConfiguration` 參數。

`ServerlessV2ScalingConfiguration` 參數包含兩個值，定義叢集中任何無伺服器執行個體的無伺服器擴展容量範圍：

- **`MinCapacity`** — 叢集中任何 DocumentDB 無伺服器執行個體的最低擴展容量。
- **`MaxCapacity`** — 叢集中任何 DocumentDB 無伺服器執行個體的最大擴展容量。

選擇 DocumentDB 無伺服器叢集**`MinCapacity`**的設定

總是為 選擇 0.5 很吸引人`MinCapacity`。該值可讓執行個體在完全閒置時縮減至最小容量，同時保持作用中狀態。但是，依據您使用該叢集的方式及您設定的其他設定，不同的最小容量可能是最有效的。在選擇最小容量設定時，請考慮下列因素：

- DocumentDB 無伺服器執行個體的擴展率取決於其目前的容量。目前的容量越高，則擴充規模就越快。如果您需要執行個體快速擴展到非常高的容量，請考慮將最小容量設定為擴展速率符合您需求的值。
- 如果您通常會修改執行個體的執行個體類別，預期工作負載特別高或低，您可以使用該體驗來粗略估計同等的 DocumentDB 無伺服器容量範圍。若要判斷佈建 Amazon DocumentDB 執行個體類型的記憶體大小，請參閱 執行個體配額。

  例如，假設您在叢集的工作負載較低時，使用`db.r6g.xlarge`執行個體類別。該執行個體類別有 32 GiB 的記憶體。因此，您可以指定 16 `MinCapacity`的 來設定可縮減到大約相同容量的無伺服器執行個體。這是因為每個 DCU 對應大約 2 GiB 的記憶體。您可以指定較低的值，讓執行個體進一步縮減，以防您的`db.r6g.xlarge`執行個體有時未充分利用。
- 如果您的應用程式在緩衝快取中有特定數量的資料時最有效率地運作，請考慮指定記憶體大小足以容納經常存取資料的最小 DCU 設定。否則，當無伺服器執行個體縮減至較低的記憶體大小時，某些資料會從緩衝區快取中移出。然後，當執行個體向上擴展時，資訊會隨著時間重新讀回緩衝區快取。如果將資料帶回緩衝區快取的 I/O 量很大，選擇較高的最小 DCU 值可能會更有效。如需詳細資訊，請參閱執行個體大小。

- 如果您的 DocumentDB 無伺服器執行個體大部分時間以特定容量執行，請考慮指定低於該基準的最小容量設定，但不要太低。當目前的容量未大幅低於所需的容量時，無伺服器執行個體可以有效地估計擴展的程度和速度。
- 如果您的佈建工作負載對於 T3 或 T4g 等小型執行個體類別的記憶體需求太高，請選擇可提供與 R5 或 R6g 執行個體相當之記憶體的最低 DCU 設定。
- 針對指定的功能使用下列最小值 `MinCapacity`（可能會有所變更）：
  - 績效詳情 — 2 DCUs
  - 全域叢集 — 8 DCUs（僅適用於主要叢集 AWS 區域)
- 在 Amazon DocumentDB 中，複寫發生在儲存層，因此讀取器容量不會直接影響複寫。不過，對於獨立擴展的 DocumentDB 無伺服器讀取器執行個體，請確定最小容量足以在寫入密集期間處理工作負載，以避免查詢延遲。如果提升層 2–15 中的讀取器執行個體遇到效能問題，請考慮增加叢集的最小容量。如需變更讀取器執行個體是否與寫入器一起擴展或獨立擴展的詳細資訊，請參閱 檢視和修改無伺服器讀取器的提升層。

  如果您有具有 DocumentDB 無伺服器讀取器執行個體的叢集，當讀取器的提升層不是 0 或 1 時，讀取器不會與寫入器執行個體一起擴展。在這種情況下，設定較低的最小容量可能會導致過多的複寫延遲。這是因為讀取器可能沒有足夠的容量，無法在資料庫繁忙時套用寫入器的變更。將最小容量設定為代表與寫入器執行個體相當記憶體和 CPU 數量的值。
- DocumentDB 無伺服器執行個體從其最小容量擴展到其最大容量所需的時間，取決於其最小和最大 DCU 值之間的差異。當執行個體目前的容量很大時，DocumentDB 無伺服器會以較大的增量擴展，而不是從較小的容量啟動執行個體。因此，如果您指定相對較大的最大容量，且執行個體花費大部分時間接近該容量，請考慮增加最小 DCU 設定。如此一來，閒置的執行個體可以更快地擴展到最大容量。
- 某些執行個體限制取決於無伺服器執行個體的目前容量，例如連線限制、游標限制和開放交易限制。如果執行個體目前的容量很小，則限制也會相對較小。如果這些限制是將您的無伺服器執行個體縮減至其`MinCapacity`值時的問題，請考慮增加`MinCapacity`至較高的值。如需詳細資訊，請參閱Amazon DocumentDB 無伺服器執行個體限制。
- 此外，如果 `MinCapacity` 設定為小於或等於 1.0 DCUs例如作用中連線限制、游標限制和開放交易限制，則某些執行個體限制會限制為較低的最大值。如果這些上限限制不足以滿足您的工作負載，請使用至少 1.5 個 DCUs`MinCapacity`的值。如需詳細資訊，請參閱Amazon DocumentDB 無伺服器執行個體限制。

如需如何修改叢集擴展組態的說明，請參閱 管理 Amazon DocumentDB 無伺服器。

選擇 DocumentDB 無伺服器叢集MaxCapacity的設定

總是為 DocumentDB 無伺服器容量上限設定選擇一些高值很吸引人。較大的最大容量可讓執行個體在執行密集型工作負載時擴展最多。較低的值可避免未預期的收費可能性。依據您使用該叢集的方式及您設定的其他設定，最有效的值可能高於或低於您最初想象的值。在選擇最大容量設定時，請考慮下列因素：

- 最大容量必須至少與最小容量相同。您可將最小和最大容量設定為相同。但是，在這種情況下，容量永遠不會擴展或縮小。因此，除了在測試情況下，對最小和最大容量使用相同的值並不合適。
- 最大容量必須至少為 1.0 DCUs且最多必須為 256 DCUs。
- 監控無伺服器執行個體的擴展和資源用量。如果您的無伺服器執行個體經常擴展到最大容量並達到資源限制（例如，當DCUUtilization指標為 100.0 時），請選取較高的MaxCapacity值。如需詳細資訊，請參閱監控 Amazon DocumentDB 無伺服器。
- 如果您通常會修改佈建執行個體的執行個體類別，預期工作負載特別高或低，您可以使用該體驗來估計同等的 DocumentDB 無伺服器容量範圍。若要判斷佈建 Amazon DocumentDB 執行個體的記憶體大小，請參閱 執行個體配額。

  例如，假設您在叢集具有高工作負載時使用db.r6g.4xlarge執行個體類別。該執行個體類別有 128 GiB 的記憶體。因此，您可以指定最大 DCU 設定 64，以設定可擴展到大約相同容量的無伺服器執行個體。這是因為每個 DCU 對應大約 2 GiB 的記憶體。您可以指定更高的值，讓執行個體進一步擴展，以防您的db.r6g.4xlarge執行個體有時沒有足夠的容量來有效處理工作負載。
- 如果您的資料庫用量有預算上限，請選擇保持在該上限內的值，即使您的所有無伺服器執行個體始終以最大容量執行。請記住，當您的叢集中有 n 個無伺服器執行個體時，叢集隨時可以使用的理論無伺服器容量上限是叢集最大 DCU 設定的 n 倍。(實際消耗量可能會減少，例如，若某些讀取器獨立於寫入器擴展。)
- 如果您使用無伺服器讀取器執行個體從寫入器執行個體卸載一些唯讀工作負載，您可以選擇較低的最大容量設定。這樣做是為了反映每個讀取器執行個體不需要像叢集只包含單一執行個體一樣擴展。
- 假設您想要防止因資料庫參數設定錯誤或應用程式中的效率低查詢而造成的過度使用。於這種情況下，您可透過選擇低於您可設定的絕對最高容量設定來避免意外過度使用。
- 若因實際使用者活動所引起的峯值很少，但確實發生，您可在選擇最大容量設定時考慮這些情況。若優先順序是讓應用程式以完整的效能和可擴展性持續執行，您可指定一個高於正常用量下所觀察到的最大容量設定。若應用程式可在活動極端高峯期間以縮減的輸送量執行，則可選擇稍低的最大容量設定。確保您選擇的設定仍具有足夠的記憶體和 CPU 資源，以保持應用程式的執行。
- 如果您在叢集中開啟會增加每個執行個體記憶體用量的設定，請在決定最大 DCU 值時考慮該記憶體。這類設定包括績效詳情和全域叢集的設定。請確定最大 DCU 值允許無伺服器執行個體在使用

這些功能時擴展到足以處理工作負載的程度。如需結合低最大 DCU 設定和施加記憶體額外負荷的 Amazon DocumentDB 功能所造成的問題疑難排解資訊，請參閱 避免記憶體不足錯誤（下方）。

- 針對指定的功能使用下列最小值 `MaxCapacity`（可能會有所變更）：
  - 在具有大型資料磁碟區的叢集上建立無伺服器執行個體 – 2 個 DCUs（這包括建立無伺服器執行個體作為叢集還原的一部分）。
- 某些執行個體限制取決於執行個體目前的容量，例如連線限制、游標限制和開放交易限制。選擇工作負載`MaxCapacity`的值時，請務必謹記這些執行個體限制，以避免其中一個限制造成瓶頸。如需詳細資訊，請參閱Amazon DocumentDB 無伺服器執行個體限制。

如需如何修改叢集擴展組態的說明，請參閱 管理 Amazon DocumentDB 無伺服器。

避免記憶體不足錯誤

如果其中一個 DocumentDB 無伺服器執行個體持續達到容量上限，Amazon DocumentDB 會將執行個體設定為不相容參數的狀態，以指出此條件。雖然執行個體具有不相容參數狀態，但某些操作會遭到封鎖。例如，您無法升級引擎版本。如需 Amazon DocumentDB 執行個體狀態的詳細資訊，請參閱 監控 Amazon DocumentDB 執行個體的狀態。

一般而言，您的執行個體會在因為out-of-memory錯誤而頻繁重新啟動時進入此狀態。當這種類型的重新啟動發生時，Amazon DocumentDB 會記錄事件。若要檢視資源事件，請參閱 檢視 Amazon DocumentDB 事件。由於開啟績效詳情等設定而造成額外負荷，可能會發生異常高的記憶體用量。它也可以來自執行個體上的繁重工作負載，或來自管理與大量結構描述物件相關聯的中繼資料。

如果記憶體壓力變得較低，使得執行個體不常達到最大容量，Amazon DocumentDB 會自動將執行個體狀態變更為可用。

如要從此狀況復原，您可進行下列部分或全部動作：

- 透過變更叢集的最小 DocumentDB 容量單位 (DCU) 值，提高無伺服器執行個體的容量下限。如此可避免閒置資料庫縮減規模至記憶體容量少於叢集中啟用功能所需的容量問題。變更叢集的 DCU 設定後，重新啟動無伺服器執行個體。這樣做會評估 Amazon DocumentDB 是否可以將狀態重設回可用狀態。
- 透過變更叢集的最大 DCU 值，提高無伺服器執行個體的容量上限。如此可避免繁忙的資料庫無法擴充規模至具足夠記憶體的容量，以滿足叢集和資料庫工作負載中所開啟功能的問題。變更叢集的 DCU 設定後，重新啟動無伺服器執行個體。這樣做會評估 Amazon DocumentDB 是否可以將狀態重設回可用狀態。

- 請關閉需要記憶體額外負荷的組態設定。例如，假設您已開啟績效詳情等功能，但不使用它。如果是這樣，您可以將其關閉。或者，您可以提高叢集的容量下限和上限值，以考慮這些類型的功能所使用的記憶體。如需有關選擇最小和最大容量設定的指導方針，請參閱 選擇 DocumentDB 無伺服器叢集的擴展容量範圍。
- 減少執行個體上的工作負載。例如，您可以將讀取器執行個體新增至叢集，將唯讀查詢的負載分散到更多執行個體。

為什麼我的無伺服器執行個體沒有縮減規模？

在某些情況下，即使資料庫沒有負載，DocumentDB 無伺服器也不會縮減至最小容量。這種情況可能是由於下列原因而發生：

- Performance Insights 可以增加資源用量，並防止資料庫縮減到最小容量。重要功能如下所示：
- 如果讀取器執行個體未縮減至最低規模，且容量維持與寫入器執行個體相同或更高，請檢查讀取器執行個體的優先順序層。第 0 層或第 1 層中的 DocumentDB 無伺服器讀取器執行個體，其容量至少與寫入器執行個體一樣高。將讀取器的優先順序方案變更為 2 或更高，以便其可以獨立縱向擴展和縮減，與寫入器無關。如需詳細資訊，請參閱Amazon DocumentDB 無伺服器擴展。
- 繁重的資料庫工作負載會增加資源使用量。
- 大型資料庫磁碟區會增加資源使用量。Amazon DocumentDB 使用記憶體和 CPU 資源進行叢集管理。Amazon DocumentDB 需要更多 CPU 和記憶體來管理資料庫磁碟區的叢集。如果叢集的容量下限低於叢集管理所需的最小容量，則您的叢集將不會縮減至容量下限。
- 背景維護活動可以定期增加資源用量。

如果資料庫仍無法縮減至設定的最小容量，請停止並重新啟動資料庫，以回收可能隨著時間累積的任何記憶體片段。

> **Note**
>
> 停止和啟動資料庫會導致停機時間。僅在必要時執行此操作。

監控 Amazon DocumentDB 無伺服器

若要進一步了解 Amazon DocumentDB 中的監控，請參閱 在 Amazon DocumentDB 中監控和記錄。

主題

- 記憶體不足：不相容參數狀態
- DocumentDB Serverless 的 Amazon CloudWatch 指標
- 使用績效詳情監控 DocumentDB 無伺服器效能

記憶體不足：不相容參數狀態

如果其中一個無伺服器執行個體持續達到容量上限，Amazon DocumentDB 會將執行個體設定為不相容參數的狀態，以指出此條件。如需詳細資訊，請參閱避免記憶體不足錯誤。

DocumentDB Serverless 的 Amazon CloudWatch 指標

若要進一步了解如何搭配 Amazon DocumentDB 使用 CloudWatch，請參閱 使用 CloudWatch 監控 Amazon DocumentDB。

您可以在 CloudWatch 中檢視無伺服器執行個體，以使用 `ServerlessDatabaseCapacity` 指標監控每個執行個體耗用的容量。您也可以監控所有標準 DocumentDB CloudWatch 指標，例如 `DatabaseConnections`和 查詢。如需您可以為 Amazon DocumentDB 監控的 CloudWatch 指標完整清單，請參閱 使用 CloudWatch 監控 Amazon DocumentDB。下列 CloudWatch 執行個體層級指標對於監控您了解 DocumentDB 無伺服器執行個體如何向上和向下擴展非常重要。所有這些指標每秒計算一次。如此一來，您就可以監控無伺服器執行個體的目前狀態。您可以設定警示，以便在任何無伺服器執行個體接近容量相關指標的閾值時通知您。您可決定最小和最大容量設定是否合適，或者是否需要進行調整。您可決定將精力集中於最佳化您資料庫效率之處。

- **`ServerlessDatabaseCapacity`** — 做為執行個體層級指標，它會報告目前執行個體容量所代表的 DCUs 數量。做為叢集層級指標，代表叢集中所有 DocumentDB 無伺服器執行個體`ServerlessDatabaseCapacity`的值平均值。
- **`DCUUtilization.`** — 此指標是 DocumentDB Serverless 的新指標。此值以百分比表示。其計算方式為`ServerlessDatabaseCapacity`指標值除以叢集的最大 DCU 值。請考慮下列指導方針來解譯此指標並採取行動：
  - 如果此指標接近 的值`100.0`，則執行個體已盡可能向上擴展。考慮增加叢集的最大 DCU 設定。如此一來，寫入器和讀取器執行個體都可以擴展到更高的容量。
  - 假設唯讀工作負載會導致讀取器執行個體接近 `DCUUtilization`的 `100.0`，而寫入器執行個體並未接近其最大容量。在此情況下，請考慮將其他讀取器執行個體新增至叢集。如此一來，您可以將工作負載的唯讀部分分散到更多執行個體，以減少每個讀取器執行個體的負載。
  - 假設您正在執行生產應用程式，其中效能和可擴展性是主要考慮因素。在這種情況下，您可以將叢集的最大 DCU 值設定為高數字。您的目標是讓`DCUUtilization`指標一律低於 100.0。透過高最

大 DCU 值，您可以確信有足夠的空間，以防資料庫活動發生意外尖峰。您僅需為實際使用的資料庫容量付費。

- **CPUUtilization** — 此指標在 DocumentDB 無伺服器中與佈建執行個體中的解譯方式不同。對於 DocumentDB 無伺服器，此值是以目前使用的 CPU 數量計算的百分比，除以叢集最大 DCU 值下可用的 CPU 容量。Amazon DocumentDB 會自動監控此值，並在執行個體持續使用高比例的 CPU 容量時擴展無伺服器執行個體。

  如果此指標接近 的值`100.0`，則執行個體已達到其 CPU 容量上限。考慮增加叢集的最大 DCU 設定。如果此指標接近讀取器執行個體`100.0`上的 值，請考慮將其他讀取器執行個體新增至叢集。如此一來，您可以將工作負載的唯讀部分分散到更多執行個體，以減少每個讀取器執行個體的負載。

- **FreeableMemory** — 此值代表 DocumentDB 無伺服器執行個體擴展至其最大容量時可用的未使用記憶體數量。對於目前容量低於最大容量的每個 DCU，此值會增加約 2 GiB。因此，此指標在盡可能地縱向擴展執行個體之前不會接近零。

  如果此指標接近零值，則執行個體已盡可能向上擴展，並接近其可用記憶體的限制。考慮增加叢集的最大 DCU 設定。如果此指標在讀取器執行個體上接近零的值，請考慮將其他讀取器執行個體新增至叢集。如此一來，工作負載的唯讀部分就可以分散到更多執行個體，減少每個讀取器執行個體的記憶體用量。

- **TempStorageIops** — 在連接至執行個體的本機儲存上完成的 IOPS 數量。其包括讀取和寫入的 IOPS。此指標表示計數，且每秒測量一次。這是 DocumentDB Serverless 的新指標。如需詳細資訊，請參閱使用 CloudWatch 監控 Amazon DocumentDB。

- **TempStorageThroughput** — 與執行個體相關聯的本機儲存體往返傳輸的資料量。此指標表示位元組，且每秒測量一次。這是 DocumentDB Serverless 的新指標。如需詳細資訊，請參閱使用 CloudWatch 監控 Amazon DocumentDB。

一般而言，大多數的 DocumentDB 無伺服器執行個體擴展都是由記憶體用量和 CPU 活動造成。`TempStorageIops` 和 `TempStorageThroughput`指標可協助您診斷極少數情況下，執行個體與本機儲存裝置之間傳輸的網路活動會導致意外容量增加。如要監控其他網路活動，您可使用以下現有指標：

- `NetworkReceiveThroughput`
- `NetworkThroughput`
- `NetworkTransmitThroughput`
- `StorageNetworkReceiveThroughput`
- `StorageNetworkThroughput`

- StorageNetworkTransmitThroughput

DocumentDB 無伺服器 CloudWatch 指標如何套用至您的 AWS 帳單

AWS 帳單上的 DocumentDB 無伺服器費用是根據您可以監控的相同ServerlessDatabaseCapacity指標計算。如果您僅在一小時中使用 DocumentDB 無伺服器容量，則計費機制可能與此指標的計算 CloudWatch 平均值不同。若系統問題造成 CloudWatch 指標在短時間內無法使用，其亦可能會有所不同。因此，與從ServerlessDatabaseCapacity平均值自行計算數字時相比，您可能會在帳單上看到略有不同的 DCU 小時值。

DocumentDB 無伺服器指標的 Amazon CloudWatch CLI 命令範例

下列 AWS CLI 範例示範如何監控與 DocumentDB Serverless 相關的最重要 CloudWatch 指標。在每個案例中，將 --dimensions 參數的Value=字串取代為您自己的 DocumentDB 無伺服器執行個體的識別符。

下列 Linux 範例顯示執行個體的最小、最大和平均容量值，在一小時內每 10 分鐘測量一次。Linux 日期命令會指定相對於目前日期和時間的開始和結束時間。--query 參數中的 sort_by 函數依據 Timestamp 欄位依時間順序對結果進行排序。

```
aws cloudwatch get-metric-statistics \
    --metric-name "ServerlessDatabaseCapacity" \
    --start-time "$(date -d '1 hour ago')" \
    --end-time "$(date -d 'now')" \
    --period 600 \
    --namespace "AWS/DocDB" \
    --statistics Minimum Maximum Average \
    --dimensions Name=DBInstanceIdentifier,Value=my_instance \
    --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' \
    --output table
```

下列 Linux 範例示範監控叢集中執行個體的容量。它測量執行個體的最小、最大和平均容量使用率。在三個小時內每小時進行一次測量。這些範例使用代表 DCUs 上限百分比的DCUUtilization指標，而不是ServerlessDatabaseCapacity代表固定數量DCUs。如此一來，您就不需要知道容量範圍內最小和最大 DCU 值的實際數字。您可看到從 0 到 100 的百分比。

```
aws cloudwatch get-metric-statistics \
    --metric-name "DCUUtilization" \
```

```
    --start-time "$(date -d '3 hours ago')" \
    --end-time "$(date -d 'now')" \
    --period 3600 \
    --namespace "AWS/DocDB" \
    --statistics Minimum Maximum Average \
    --dimensions Name=DBInstanceIdentifier,Value=my_instance \
    --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' \
    --output table
```

下列 Linux 範例執行與先前測量類似的測量。於此狀況下，測量適用於 CPUUtilization 指標。在 1 小時內每 10 分鐘進行一次測量。數字代表使用之可用 CPU 的百分比，以執行個體最大容量設定的可用 CPU 資源為基礎。

```
aws cloudwatch get-metric-statistics \
    --metric-name "CPUUtilization" \
    --start-time "$(date -d '1 hour ago')" \
    --end-time "$(date -d 'now')" \
    --period 600 \
    --namespace "AWS/DocDB" \
    --statistics Minimum Maximum Average \
    --dimensions Name=DBInstanceIdentifier,Value=my_instance \
    --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' \
    --output table
```

下列 Linux 範例執行與先前測量類似的測量。於此狀況下，測量適用於 FreeableMemory 指標。在 1 小時內每 10 分鐘進行一次測量。

```
aws cloudwatch get-metric-statistics \
    --metric-name "FreeableMemory" \
    --start-time "$(date -d '1 hour ago')" \
    --end-time "$(date -d 'now')" \
    --period 600 \
    --namespace "AWS/DocDB" \
    --statistics Minimum Maximum Average \
    --dimensions Name=DBInstanceIdentifier,Value=my_instance \
    --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' \
    --output table
```

使用績效詳情監控 DocumentDB 無伺服器效能

您可以使用 Performance Insights 來監控 DocumentDB 無伺服器執行個體的效能。如需有關績效詳情的程序，請參閱 使用績效詳情進行監控。

下列新的績效詳情計數器適用於 DocumentDB 無伺服器執行個體：

- **os.general.serverlessDBCapacity** — DCUs 中執行個體的目前容量。此值對應至執行個體的 ServerlessDatabaseCapacity CloudWatch 指標。
- **os.general.dcuUtilization** — 目前容量超出最大設定容量的百分比。此值對應至執行個體的 DCUUtilization CloudWatch 指標。
- **os.general.maxConfiguredDcu** — 您為此 DocumentDB 無伺服器執行個體設定的容量上限。它以 DCUs測量。
- **os.general.minConfiguredDcu** — 您為此 DocumentDB 無伺服器執行個體設定的最小容量。它以 DCUs測量。

如需績效詳情計數器的完整清單，請參閱 計數器指標的績效詳情。

在績效詳情中顯示 DocumentDB 無伺服器執行個體的 vCPU 值時，這些值代表根據執行個體 DCU 值的預估值。在預設的一分鐘間隔中，任何小數 vCPU 值皆會無條件進位至最接近的非負整數。若為較長的時間間隔，顯示的 vCPU 值為每分鐘整數 vCPU 值的平均值。

使用 Amazon DocumentDB 彈性叢集

Amazon DocumentDB 彈性叢集支援每秒數百萬個讀取/寫入以及 PB 儲存容量的工作負載。彈性叢集也透過免除選擇、管理或升級執行個體的需求，簡化開發人員與 Amazon DocumentDB 的互動方式。

> **Note**
>
> 彈性叢集使用 MongoDB 5.0 相容線路通訊協定。Amazon DocumentDB 引擎 8.0 版不支援它們，並且不適用於中國區域或 AWS GovCloud。

Amazon DocumentDB 彈性叢集的建立目的是：

- 為尋找資料庫的客戶提供解決方案，該資料庫提供幾乎無限的擴展，具有豐富的查詢功能和 MongoDB API 相容性。
- 為客戶提供更高的連線限制，並減少修補的停機時間。
- 繼續投資雲端原生、彈性和類別領導的 JSON 工作負載架構。

主題

- 彈性叢集使用案例
- 彈性叢集的優點
- 彈性叢集區域和版本可用性
- 限制
- Amazon DocumentDB 彈性叢集：運作方式
- Amazon DocumentDB 彈性叢集入門
- Amazon DocumentDB 彈性叢集最佳實務
- 管理 Amazon DocumentDB 彈性叢集
- Amazon DocumentDB 彈性叢集的靜態資料加密
- 彈性叢集中的服務連結角色

彈性叢集使用案例

文件資料庫對於需要靈活結構描述以便快速進行重複性開發的工作負載來說非常適合。如需 Amazon DocumentDB 使用案例的範例，請參閱 文件資料庫使用案例。

以下是彈性叢集可提供顯著優勢的一些使用案例範例：

使用者設定檔

由於文件資料庫具有彈性的結構描述，因此它們可以大規模存放具有不同屬性和資料值的文件。彈性叢集是線上設定檔的實用解決方案，其中不同的使用者提供不同類型的資訊。假設您的應用程式支援數億使用者設定檔。您可以使用彈性叢集來支援此類應用程式，因為它們可以向上擴展和向外擴展，以支援數百萬筆寫入和讀取這些使用者設定檔。您也可以縮減離峰時數以降低成本。

內容管理和歷史記錄

為了有效管理內容，您必須能夠收集和彙總各種不同來源的內容，然後將內容傳遞給客戶。文件資料庫採用靈活的結構描述，是收集和存放任何類型資料最理想的選擇。您可以使用它們來建立和整合新的內容類型，包括使用者產生的內容，例如影像、評論和影片。隨著時間的推移，您的資料庫可能需要更多儲存空間。透過彈性叢集，您可以將資料分散到更多儲存磁碟區，讓您可以將 PB 的資料存放在單一叢集中。

彈性叢集的優點

AWS 服務整合

Amazon DocumentDB 彈性叢集與其他 AWS 服務整合的方式與 Amazon DocumentDB 相同：

- 遷移 — 您可以使用 AWS Database Migration Service (DMS) 從 MongoDB 和其他關聯式資料庫遷移至 Amazon DocumentDB 彈性叢集。
- 監控 — 您可以使用 Amazon CloudWatch 監控彈性叢集的運作狀態和效能。
- 安全：您可以透過 AWS Identity and Access Management (IAM) 設定身分驗證和授權，以管理彈性叢集資源，例如建立和更新彈性叢集，並使用 Amazon VPC 進行僅限 VPC 的安全連線。不支援彈性叢集的 IAM 身分驗證登入。
- 資料管理 — 您可以使用 從 Amazon S3、Amazon Redshift 和 Amazon OpenSearch Service 等 AWS 其他服務 AWS Glue 匯入和匯出資料。

彈性叢集區域和版本可用性

彈性叢集的區域可用性

下表顯示目前可使用 Amazon DocumentDB 彈性叢集 AWS 區域 的 ，以及每個區域的端點。

區域名稱	區域	可用區域
美國東部 (維吉尼亞北部)	us-east-1	5
美國東部 (俄亥俄)	us-east-2	3
美國西部 (奧勒岡)	us-west-2	3
亞太地區 (香港)	ap-east-1	3
亞太地區 (孟買)	ap-south-1	3
亞太地區 (首爾)	ap-northeast-2	3
亞太地區 (新加坡)	ap-southeast-1	3
亞太地區 (悉尼)	ap-southeast-2	3
亞太地區 (東京)	ap-northeast-1	3
加拿大 (中部)	ca-central-1	3
南美洲 (聖保羅)	sa-east-1	3
歐洲 (法蘭克福)	eu-central-1	3
歐洲 (愛爾蘭)	eu-west-1	3
歐洲 (倫敦)	eu-west-2	3
歐洲 (米蘭)	eu-south-1	3
Europe (Paris)	eu-west-3	3

版本可用性

彈性叢集支援 MongoDB 5.0 相容線路通訊協定。如需 Amazon DocumentDB 4.0 執行個體型叢集和彈性叢集之間的差異，請參閱 Amazon DocumentDB 4.0 和彈性叢集之間的功能差異。注意：Amazon DocumentDB 8.0 不支援彈性叢集

限制

彈性叢集管理

此版本不支援下列叢集管理功能：

- 自訂連接埠（僅支援 27017)
- 能夠使用 IAM 使用者和角色對資料庫進行身分驗證
- 能夠建立全域叢集
- 現有的 Amazon DocumentDB 事件和訂閱事件
- 範圍碎片
- 碎片現有集合
- 多欄位碎片索引鍵
- 變更碎片索引鍵
- Point-in-time還原
- 複製
- Performance Insights

> **Note**
>
> 如需彈性叢集配額的相關資訊，請參閱 Amazon DocumentDB 配額。

查詢和寫入操作

此版本不支援下列查詢和寫入操作命令和功能：

- 擴展操作期間的 DDL 命令
- Profiler
- 參數群組
- AWS Config
- AWS Backup

集合和索引管理

此版本不支援下列集合和索引管理功能：

- 唯一索引
- 部分索引
- 文字索引
- 向量索引
- 文件壓縮

管理和診斷

此版本不支援下列管理和診斷命令和功能：

- AWS Secrets Manager
- Role-based-access-control(RBAC) 自訂角色。
- 連線時，不支援寫入問題 0。
- 變更屬於目前未指派給現有彈性叢集之 VPC 的子網路。

選擇加入功能

此版本不支援下列 Amazon DocumentDB 選擇加入功能：

- ACID 交易
- DDL/DML 稽核
- Change streams
- 工作階段命令

Amazon DocumentDB 彈性叢集：運作方式

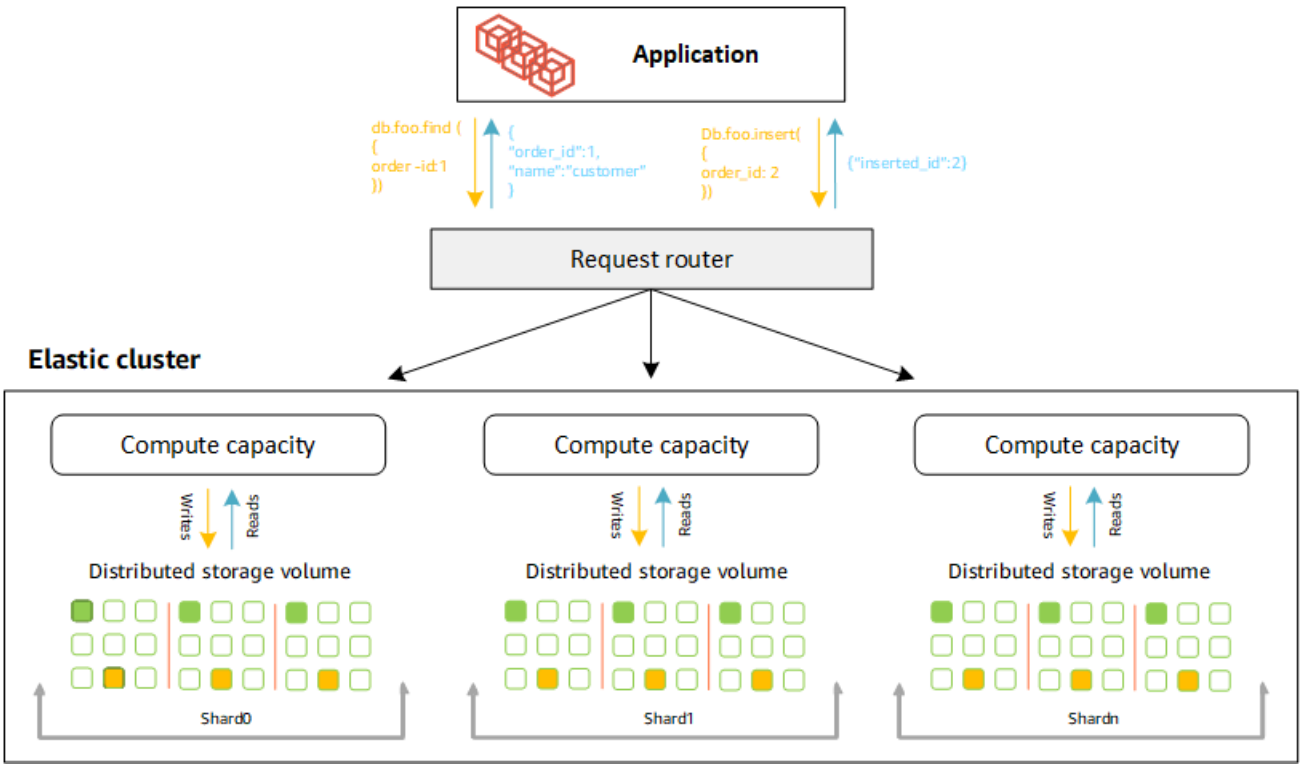
本節中的主題提供有關為 Amazon DocumentDB 彈性叢集提供支援的機制和功能的資訊。

主題

- Amazon DocumentDB 彈性叢集碎片
- 彈性叢集遷移
- 彈性叢集擴展
- 彈性叢集可靠性
- 彈性叢集儲存和可用性
- Amazon DocumentDB 4.0 和彈性叢集之間的功能差異

Amazon DocumentDB 彈性叢集碎片

Amazon DocumentDB 彈性叢集使用雜湊式碎片來分割分散式儲存系統中的資料。碎片也稱為分割，可將大型資料集分割為跨多個節點的小型資料集，讓您將資料庫擴展到超過垂直擴展限制的範圍。彈性叢集使用 Amazon DocumentDB 中運算和儲存的分離或解耦，可讓您獨立擴展。彈性叢集不會透過在運算節點之間移動少量資料來重新分割集合，而是在分散式儲存系統中有效率地複製資料。



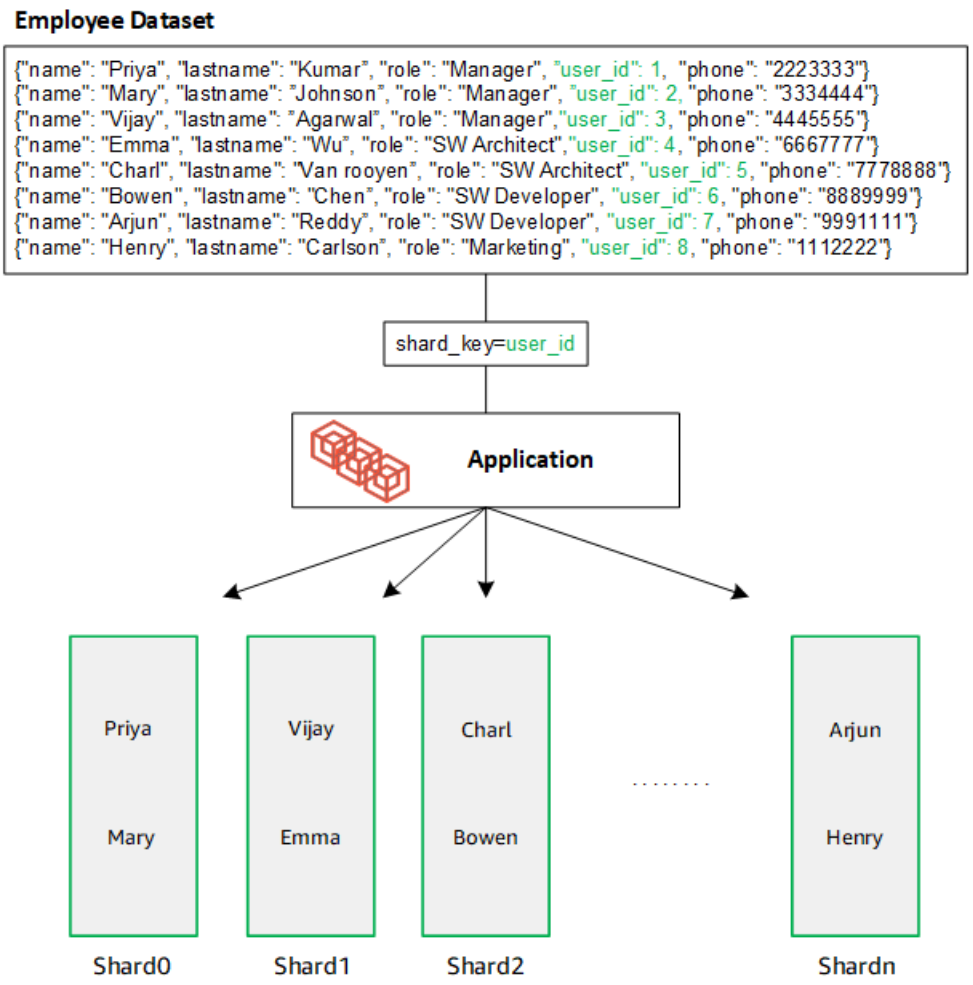
碎片定義

碎片命名法的定義：

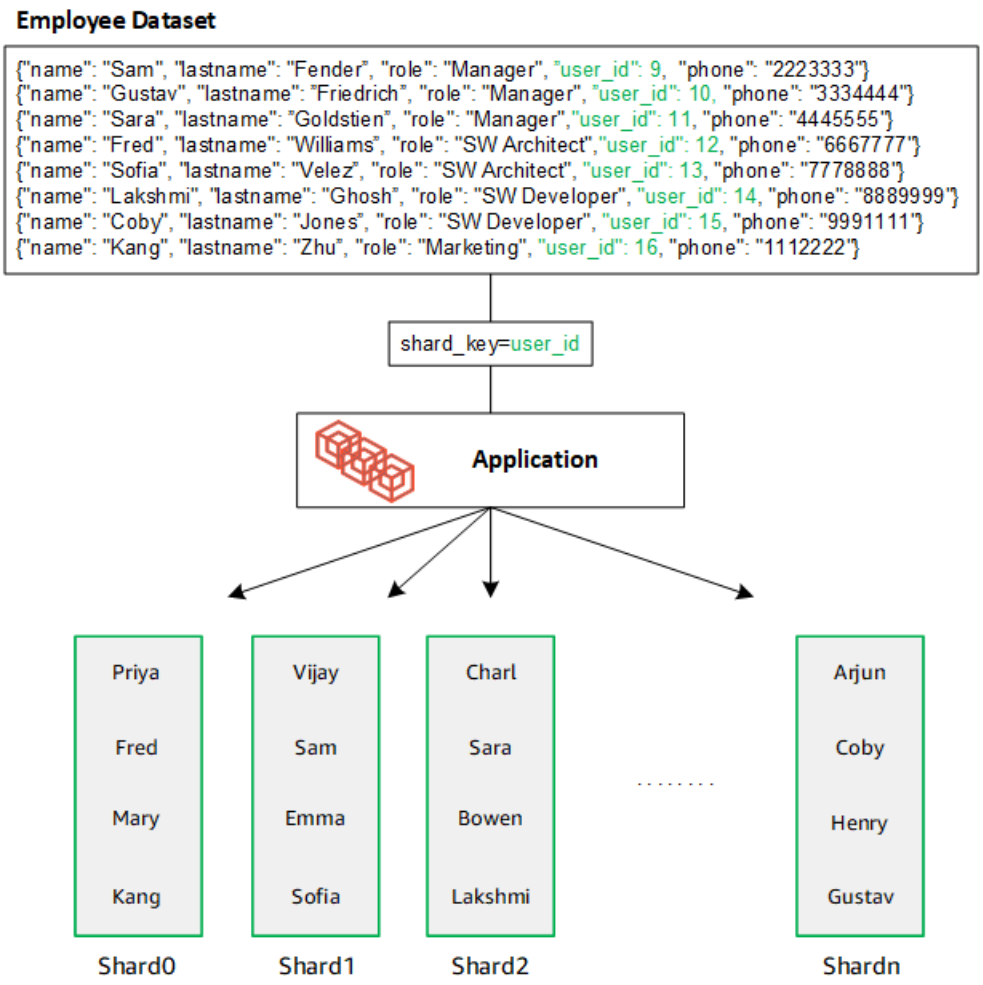
- 碎片 — 碎片提供彈性叢集的運算。它將有一個寫入器執行個體和 0–15 個僅供讀取複本。根據預設，碎片將有兩個執行個體：寫入器和單一僅供讀取複本。您最多可以設定 32 個碎片，每個碎片執行個體最多可以有 64 vCPUs。
- 碎片索引鍵 — 碎片索引鍵是 JSON 文件中碎片集合的必要欄位，彈性叢集會用來將讀取和寫入流量分散至相符的碎片。
- 碎片集合 — 碎片集合是一種集合，其資料分散在資料分割區中的彈性叢集。
- 分割區 — 分割區是碎片資料的邏輯部分。當您建立碎片集合時，資料會根據碎片金鑰自動組織成每個碎片內的分割區。每個碎片都有多個分割區。

在設定的碎片之間分配資料

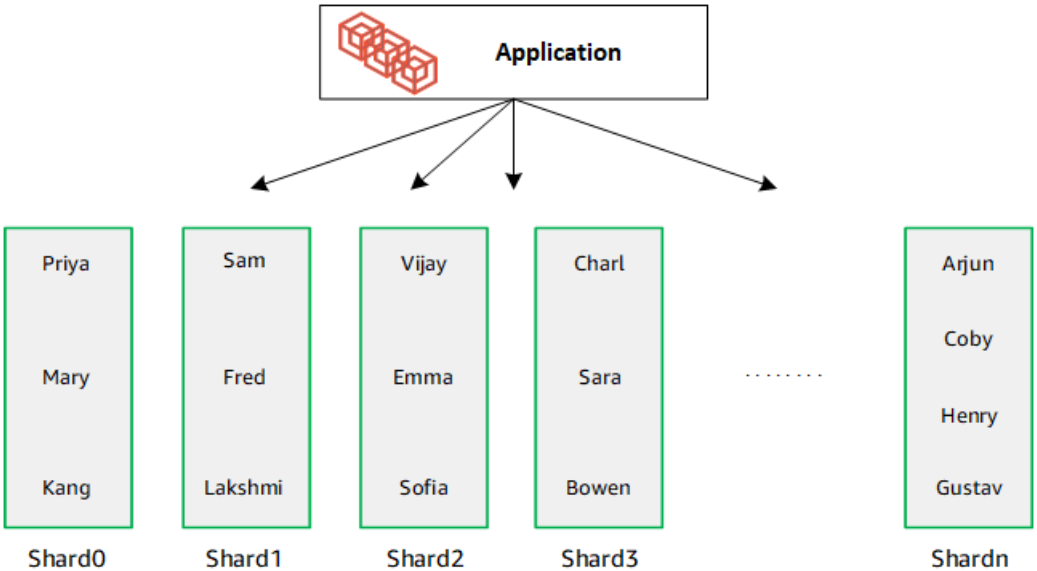
建立具有許多唯一值的碎片金鑰。良好的碎片金鑰會將您的資料平均分割到基礎碎片，為您的工作負載提供最佳輸送量和效能。下列範例是使用名為 "user_id" 碎片金鑰的員工名稱資料：



DocumentDB 使用雜湊碎片將資料分割至基礎碎片。插入和分發額外資料的方式相同：



當您透過新增其他碎片來擴展資料庫時，Amazon DocumentDB 會自動重新分配資料：



彈性叢集遷移

Amazon DocumentDB 支援將 MongoDB 碎片資料遷移至彈性叢集。支援離線、線上和混合遷移方法。如需詳細資訊，請參閱遷移和升級 Amazon DocumentDB。

彈性叢集擴展

Amazon DocumentDB 彈性叢集可讓您增加彈性叢集中的碎片數量（向外擴展），以及套用至每個碎片的 vCPUs 數量（向上擴展）。您也可以視需要減少碎片和運算容量 (vCPUs的數量。

如需擴展最佳實務，請參閱 擴展彈性叢集。

> **Note**
>
> 也可使用叢集層級擴展。如需詳細資訊，請參閱擴展 Amazon DocumentDB 叢集。

彈性叢集可靠性

Amazon DocumentDB 的設計可靠、耐用且容錯。為了改善可用性，彈性叢集會在不同可用區域的每個碎片部署兩個節點。Amazon DocumentDB 包含數個自動功能，使其成為可靠的資料庫解決方案。如需詳細資訊，請參閱Amazon DocumentDB 可靠性。

彈性叢集儲存和可用性

Amazon DocumentDB 資料存放在叢集磁碟區中，這是使用固態硬碟 (SSDs的單一虛擬磁碟區。叢集磁碟區包含六個資料複本，這些複本會自動複寫到單一 AWS 區域中的多個可用區域。此複寫有助於確保您的資料具有高耐用性，同時降低資料遺失的機率。它也有助於確保您的叢集在容錯移轉時提供更高的可用性，因為資料副本已存在於其他可用區域。如需儲存、高可用性和複寫的詳細資訊，請參閱 Amazon DocumentDB：運作方式。

Amazon DocumentDB 4.0 和彈性叢集之間的功能差異

Amazon DocumentDB 4.0 和彈性叢集之間存在下列功能差異。

- 來自 `top`和 的結果`collStats`會依碎片分割。對於碎片集合，資料會分散在多個分割區中`collScans`，並從分割區彙總`collStats`的報告。
- 叢集`collStats`碎片計數變更時，會重設碎片集合的 `top`和 集合統計資料。

- 備份內建角色現在支援 serverStatus。動作 - 具有備份角色的開發人員和應用程式可以收集有關 Amazon DocumentDB 叢集狀態的統計資料。
- SecondaryDelaySecs 欄位會在replSetGetConfig輸出slaveDelay中取代 。
- hello 命令會取代 isMaster - hello傳回描述彈性叢集角色的文件。
- 彈性叢集中的$elemMatch運算子僅符合陣列的第一個巢狀層級中的文件。在 Amazon DocumentDB 4.0 中，運算子會先周遊所有層級，再傳回相符的文件。例如：

```
db.foo.insert(
[
    {a: {b: 5}},
    {a: {b: [5]}},
    {a: {b: [3, 7]}},
    {a: [{b: 5}]},
    {a: [{b: 3}, {b: 7}]},
    {a: [{b: [5]}]},
    {a: [{b: [3, 7]}]},
    {a: [[{b: 5}]]},
    {a: [[{b: 3}, {b: 7}]]},
    {a: [[{b: [5]}]]},
    {a: [[{b: [3, 7]}]]}
]);
// Elastic clusters
> db.foo.find({a: {$elemMatch: {b: {$elemMatch: {$lt: 6, $gt: 4}}}}}, {_id: 0})
{ "a" : [ { "b" : [ 5 ] } ] }

// Docdb 4.0: traverse more than one level deep
> db.foo.find({a: {$elemMatch: {b: {$elemMatch: {$lt: 6, $gt: 4}}}}}, {_id: 0})
{ "a" : [ { "b" : [ 5 ] } ] }
{ "a" : [ [ { "b" : [ 5 ] } ] ] }
```

- Amazon DocumentDB 4.0 中的「$」投影會傳回所有欄位的所有文件。使用彈性叢集時，具有「$」投影的find命令會傳回符合查詢參數的文件，只包含符合「$」投影的欄位。
- 在彈性叢集中，具有 $regex和 $options查詢參數的find命令會傳回錯誤：「無法在 $regex 和 $options 中設定選項」。

- 使用彈性叢集時， $indexOfCP 現在會在下列情況下傳回 "-1"：

- 在 中找不到 子字串`string expression`，或
- `start` 是大於 的數字`end`，或
- `start` 是大於字串位元組長度的數字。

在 Amazon DocumentDB 4.0 中，當`start`位置的數字大於 `end`或字串的位元組長度時，會`$indexOfCP`傳回 "0"。

- 使用彈性叢集時， 中的投影操作，`_id fields`例如 `{"_id.nestedField" : 1}`，會傳回僅包含投影欄位的文件。同時，在 Amazon DocumentDB 4.0 中，巢狀欄位投影命令不會篩選掉任何文件。

Amazon DocumentDB 彈性叢集入門

此入門區段會逐步解說如何建立和查詢您的第一個彈性叢集。

連線和開始使用 Amazon DocumentDB 的方法有很多。下列程序是使用者開始使用我們強大文件資料庫最快速、最簡單且最簡單的方法。本指南使用 直接從 AWS CloudShell連接和查詢 Amazon DocumentDB 叢集 AWS 管理主控台。符合 AWS 免費方案資格的新客戶可免費使用 Amazon DocumentDB 和 CloudShell。如果您的 AWS CloudShell 環境或 Amazon DocumentDB 彈性叢集使用免費方案以外的資源，則會向您收取這些資源的正常 AWS 費率。本指南將協助您在不到 5 分鐘內開始使用 Amazon DocumentDB。

主題

- 先決條件
- 步驟 1：建立彈性叢集
- 步驟 2：連線至您的彈性叢集
- 步驟 3：碎片化您的集合、插入和查詢資料
- 步驟 4：探索

先決條件

建立第一個 Amazon DocumentDB 叢集之前，您必須執行下列動作：

建立 Amazon Web Services (AWS) 帳戶

您必須先擁有 Amazon Web Services () 帳戶，才能開始使用 Amazon DocumentDB。AWS AWS 帳戶是免費的。您僅需按使用的服務和資源量付費。

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

   當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

設定 needed AWS Identity and Access Management (IAM) 許可。

管理叢集、執行個體和叢集參數群組等 Amazon DocumentDB 資源的存取權，需要 AWS 可用於驗證請求的登入資料。如需詳細資訊，請參閱Amazon DocumentDB 的 Identity and Access Management。

1. 在 的搜尋列中 AWS 管理主控台，在下拉式選單中輸入**IAM**並選取 IAM。
2. 進入 IAM 主控台後，請從導覽窗格中選取使用者。
3. 選取您的使用者名稱。
4. 選擇新增許可。
5. 選取直接連接政策。
6. 在搜尋列`AmazonDocDBElasticFullAccess`中輸入 ，並在搜尋結果中出現時選取它。
7. 選擇下一步。
8. 選擇新增許可。

> **Note**
>
> AWS 您的帳戶在每個區域中包含預設 VPC。如果您選擇使用 Amazon VPC，請完成《Amazon VPC 使用者指南》中建立 Amazon VPC 主題的步驟。

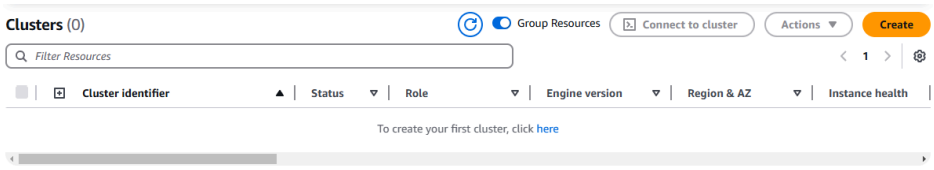
步驟 1：建立彈性叢集

在本節中，我們將說明如何使用 AWS 管理主控台 或 AWS CLI 搭配下列指示來建立新的彈性叢集。

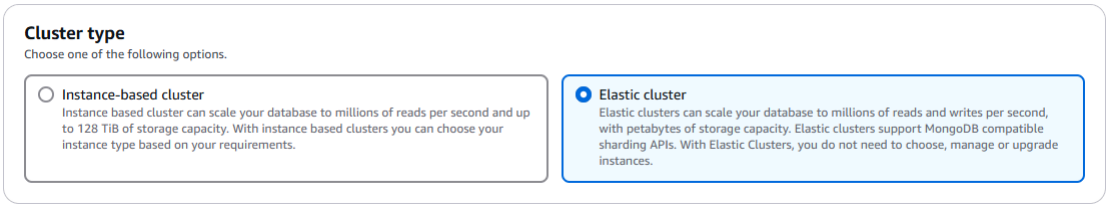
Using the AWS 管理主控台

若要使用 建立彈性叢集組態 AWS 管理主控台：

1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在 Amazon DocumentDB 管理主控台的叢集下，選擇建立。



3. 在建立 Amazon DocumentDB 叢集頁面上的叢集類型區段中，選擇彈性叢集。



4. 在組態區段中，設定下列項目：

   a. 在叢集名稱欄位中，輸入唯一的叢集識別符（遵循 欄位下方的命名要求）。

   b. 在碎片計數欄位中，輸入叢集中您想要的碎片數量。每個叢集的碎片數目上限為 32。

   > **Note**
   >
   > 每個碎片將部署兩個節點。兩個節點都有相同的碎片容量。

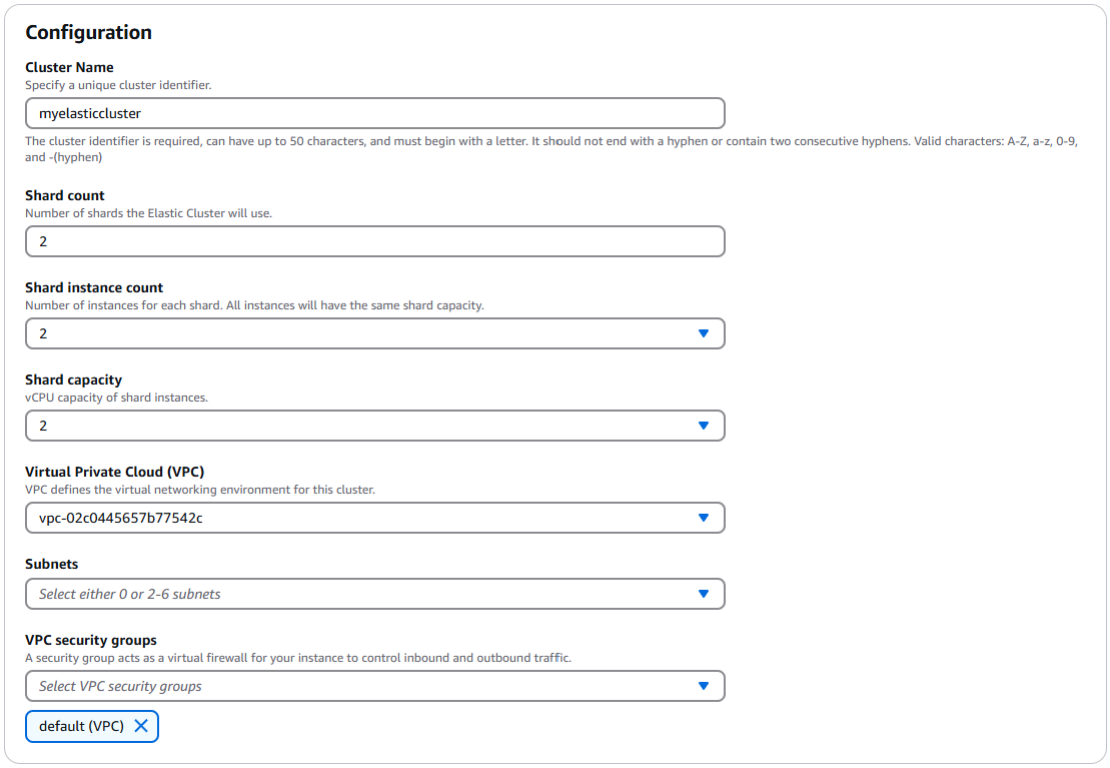
   c. 在碎片執行個體計數欄位中，選擇您希望與每個碎片相關聯的複本執行個體數量。碎片執行個體的數量上限為 16，增量為 1。所有複本執行個體都具有與下列欄位中定義的相同碎片容量。基於測試目的，預設值 2 應該已足夠。

> **Note**
>
> 複本執行個體的數量會套用至彈性叢集中的所有碎片。碎片執行個體計數值 1 表示有一個寫入器執行個體，而任何其他執行個體都是複本，可用於讀取並改善可用性。基於測試目的，預設值 2 應該已足夠。

d. 在碎片容量欄位中，選擇您希望與每個碎片執行個體相關聯的虛擬 CPUs (vCPUs) 數量。每個碎片執行個體vCPUs 數目上限為 64。允許的值為 2、4、8、16、32、64。基於測試目的，預設值 2 應該已足夠。

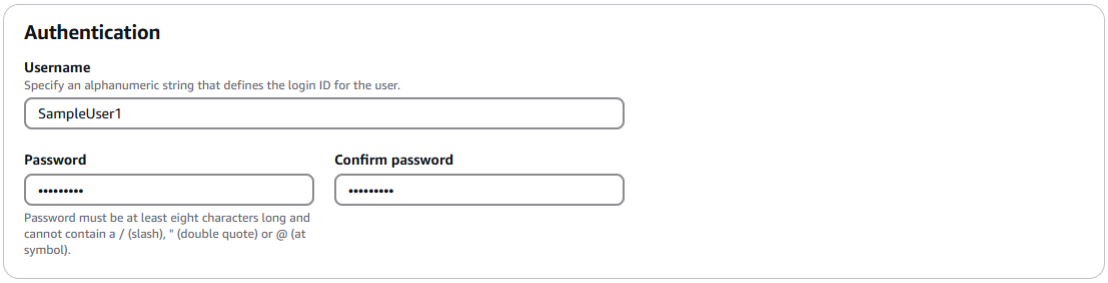
e. 在虛擬私有雲端 (VPC) 欄位中，從下拉式清單中選擇 VPC。

f. 對於子網路和 VPC 安全群組，您可以使用預設值或選取您選擇的三個子網路，以及最多三個 VPC 安全群組 （至少一個）。



5. 在身分驗證區段中，在使用者名稱欄位中輸入可識別主要使用者的登入名稱的字串。

   在密碼欄位中，輸入符合指示的唯一密碼，然後確認。



6. 在加密區段中，保留預設設定（預設金鑰)。

   或者，您可以輸入您建立的 AWS KMS key ARN。如需詳細資訊，請參閱Amazon DocumentDB 彈性叢集的靜態資料加密。

   > **Important**
   >
   > 必須為彈性叢集啟用加密。

7. 在備份區段中，根據您的備份需求編輯欄位。基於測試目的，您可以保留預設設定。



   a. 備份保留期 - 在清單中，選擇在刪除叢集之前保留此叢集自動備份的天數。

   b. 備份時段 - 設定 Amazon DocumentDB 要備份此叢集的每日時間和持續時間。

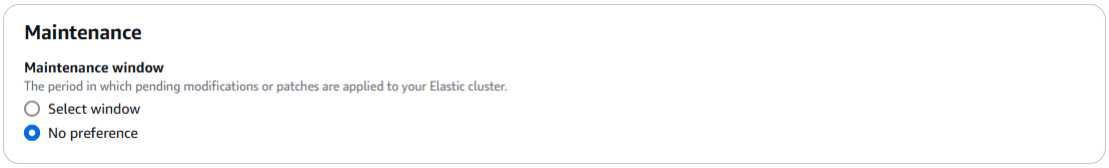
      i. 如果您想要設定建立備份的時間和持續時間，請選擇選取視窗。

         開始時間 - 在第一個清單中，選擇啟動自動備份的開始時間小時 (UTC)。在第二個清單中，選擇您要自動備份開始進行的時間 (分)。

         持續時間 - 在清單中，選擇要配置給建立自動備份的時數。

      ii. 如果您希望 Amazon DocumentDB 選擇建立備份的時間和持續時間，請選擇無偏好設定。

8. 在維護區段中，選擇修改或修補程式套用至叢集的日期、時間和持續時間。基於測試目的，您可以保留預設設定。



9. 選擇 Create Cluster (建立叢集)。

彈性叢集現在正在佈建。這可能需要幾分鐘的時間才能完成。當彈性叢集狀態在叢集清單中顯示為可用時，您可以連線至叢集。

Using the AWS CLI

若要使用 建立彈性叢集 AWS CLI，請使用 `create-cluster`操作搭配下列參數：

- --cluster-name - 必要項目。在建立或上次修改期間輸入的彈性擴展叢集的目前名稱。
- --shard-capacity - 必要項目。指派給每個碎片vCPUs 數量。上限為 64。允許的值為 2、4、8、16、32、64。
- --shard-count - 必要項目。指派給叢集的碎片數量。上限為 32。
- --shard-instance-count- 選用。套用至此叢集中所有碎片的複本執行個體數目。上限為 16。
- --admin-user-name - 必要項目。與管理員使用者相關聯的使用者名稱。
- --admin-user-password - 必要項目。與管理員使用者相關聯的密碼。
- --auth-type - 必要項目。用來決定在何處擷取用於存取彈性叢集之密碼的身分驗證類型。有效類型為 `PLAIN_TEXT`或 `SECRET_ARN`。
- --vpc-security-group-ids- 選用。設定要與此叢集建立關聯的 EC2 VPC 安全群組清單。
- --preferred-maintenance-window- 選用。設定每週可以進行系統維護的時間範圍，以國際標準時間 (UTC) 為單位。

  格式為：`ddd:hh24:mi-ddd:hh24:mi`。有效天數 (ddd)：週一、週二、週三、週四、週五、週六、週日

  預設值是從每個 Amazon Web Services 區域的 8 小時時段中隨機選取的 30 分鐘時段，該時段隨機發生在一週中的某一天。

  最短 30 分鐘時段。
- --kms-key-id- 選用。設定加密叢集的 KMS 金鑰識別符。

KMS 金鑰識別符是 AWS KMS 加密金鑰的 Amazon Resource Name (ARN)。如果您使用擁有用來加密新叢集之 KMS 加密金鑰的相同 Amazon Web Services 帳戶來建立叢集，您可以使用 KMS 金鑰別名，而不是 KMS 加密金鑰的 ARN。

如果未在 KmsKeyId 中指定加密金鑰，且 參數`StorageEncrypted`為 true，Amazon DocumentDB 會使用預設加密金鑰。

- --preferred-backup-window- 選用。建立自動備份的每日偏好時間範圍。預設值是從 8 小時時段中隨機選取的 30 分鐘時段 AWS 區域。
- --backup-retention-period- 選用。自動備份保留的天數。預設值為 1.
- --storage-encrypted- 選用。設定叢集是否已加密。

  `--no-storage-encrypted` 指定叢集未加密。
- --subnet-ids- 選用。設定網路子網路 ID。

在下列範例中，將每個*#########*取代為您自己的資訊。

> **Note**
>
> 下列範例包括建立特定的 KMS 金鑰。若要使用預設 KMS 金鑰，請勿包含 `--kms-key-id` 參數。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic create-cluster \
     --cluster-name sample-cluster-123 \
     --shard-capacity 8 \
     --shard-count 4 \
     --shard-instance-count 3 \
     --auth-type PLAIN_TEXT \
     --admin-user-name testadmin \
     --admin-user-password testPassword \
     --vpc-security-group-ids ec-65f40350 \
     --kms-key-id arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 \
     --subnet-ids subnet-9253c6a3, subnet-9f1b5af9 \
     --preferred-backup-window 18:00-18:30 \
     --backup-retention-period 7
```

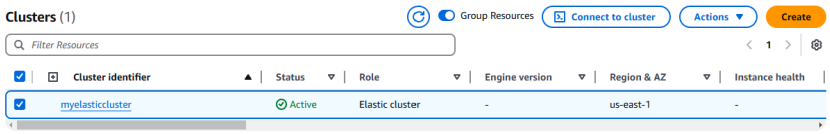
針對 Windows：

```
aws docdb-elastic create-cluster ^
     --cluster-name sample-cluster-123 ^
     --shard-capacity 8 ^
     --shard-count 4 ^
     --shard-instance-count 3 ^
     --auth-type PLAIN_TEXT ^
     --admin-user-name testadmin ^
     --admin-user-password testPassword ^
     --vpc-security-group-ids ec-65f40350 ^
     --kms-key-id arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 ^
     --subnet-ids subnet-9253c6a3, subnet-9f1b5af9 \
     --preferred-backup-window 18:00-18:30 \
     --backup-retention-period 7
```

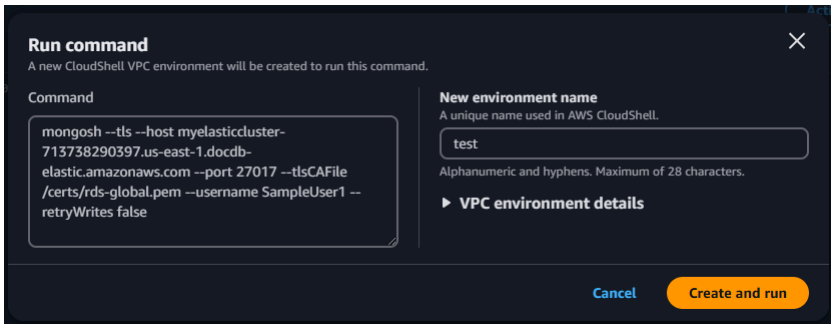
步驟 2：連線至您的彈性叢集

使用 連線至 Amazon DocumentDB 彈性叢集 AWS CloudShell。

1. 在 Amazon DocumentDB 管理主控台的叢集下，找到您建立的彈性叢集。按一下叢集旁的核取方塊，選擇對應叢集。



2. 選擇連線至叢集（位於動作下拉式功能表旁。只有在您按一下叢集旁的核取方塊，且叢集的狀態顯示為可用之後，才會啟用此按鈕。CloudShell 執行命令畫面隨即顯示。
3. 在新增環境名稱欄位中，輸入唯一名稱 (例如 "test")，然後選擇建立並執行。VPC 環境詳細資訊會自動為您的 Amazon DocumentDB 資料庫設定。



4. 出現提示時，輸入您在步驟 1：建立 Amazon DocumentDB 彈性叢集中建立的密碼（子步驟 5)。



輸入密碼且提示變成 後direct: mongos] <env-name>>，您已成功連線到 Amazon DocumentDB 叢集

> **Note**
>
> 如需故障診斷的資訊，請參閱故障診斷 Amazon DocumentDB。

步驟 3：碎片化您的集合、插入和查詢資料

彈性叢集新增對 Amazon DocumentDB 中碎片的支援。現在您已連線至叢集，您可以碎片化叢集、插入資料並執行一些查詢。

1. 若要分割集合，請輸入下列內容：

```
sh.shardCollection("db.Employee1" , { "Employeeid" : "hashed" })
```

2. 若要插入單一文件，請輸入下列項目：

```
db.Employee1.insertOne({"Employeeid":1, "Name":"Joe", "LastName": "Bruin", "level":
 1 })
```

隨即顯示下列輸出：

```
WriteResult({ "nInserted" : 1 })
```

3. 若要讀取您撰寫的文件，請輸入 findOne()命令（它傳回單一文件）：

```
db.Employee1.findOne()
```

隨即顯示下列輸出：

```
{
"_id" : ObjectId("61f344e0594fe1a1685a8151"),
"EmployeeID" : 1,
"Name" : "Joe",
"LastName" : "Bruin",
"level" : 1
}
```

4. 若要執行更多查詢，請考慮遊戲設定檔使用案例。首先，將幾個項目插入名為 "Employee" 的集合。輸入下列資料：

```
db.profiles.insertMany([ { "_id": 1, "name": "Matt", "status": "active", "level":
 12, "score": 202 },
     { "_id": 2, "name": "Frank", "status": "inactive", "level": 2, "score": 9 },
     { "_id": 3, "name": "Karen", "status": "active", "level": 7, "score": 87 },
     { "_id": 4, "name": "Katie", "status": "active", "level": 3, "score": 27 }
])
```

隨即顯示下列輸出：

```
{ acknowledged: true,
     insertedIds: {
        '0': ObjectId('679d02cd6b5a0581be78bcbd'),
        '1': ObjectId('679d02cd6b5a0581be78bcbe'),
        '2': ObjectId('679d02cd6b5a0581be78bcbf'),
        '3': ObjectId('679d02cd6b5a0581be78bcc0')
    }
}
```

5. 若要傳回設定檔集合中的所有文件，請輸入 `find()` 命令：

```
db.Employee.find()
```

您在步驟 4 中輸入的資料隨即顯示。

6. 若要查詢單一文件，請包含篩選條件 （ 例如："Katie")。輸入下列資料：

```
db.Employee.find({name: "Katie"})
```

隨即顯示下列輸出：

```
[

   {

     _id: ObjectId('679d02cd6b5a0581be78bcc0'),

     Employeeid: 4,

     name: 'Katie',

     lastname: 'Schaper',

     level: 3

   }

]
```

7. 若要尋找設定檔並進行修改，請輸入 findAndModify命令。在此範例中，員工 "Matt" 獲得更高層級的 "14"：

```
db.Employee.findAndModify({
    query: { "Employeeid" : 1, "name" : "Matt"},
    update: { "Employeeid" : 1, "name" : "Matt", "lastname" : "Winkle", "level" :
 14 }
})
```

隨即顯示下列輸出（請注意，關卡尚未變更）：

```
{

   _id: ObjectId('679d02cd6b5a0581be78bcbd'),

   Employeeid: 1,

   name: 'Matt',

   lastname: 'Winkle',
```

```
    level: 12

}
```

8. 若要驗證關卡增加，請輸入下列查詢：

```
db.Employee.find({name: "Matt"})
```

隨即顯示下列輸出：

```
[
   {

   _id: ObjectId('679d02cd6b5a0581be78bcbd'),

   Employeeid: 1,

   name: 'Matt',

   lastname: 'Winkle',

   level: 14

   }
]
```

步驟 4：探索

恭喜您！您已成功完成 Amazon DocumentDB 彈性叢集的入門程序。

下一步是什麼？ 了解如何充分利用此資料庫及其一些熱門功能：

- Amazon DocumentDB 彈性叢集最佳實務
- 管理 Amazon DocumentDB 彈性叢集

> **Note**
>
> 除非您刪除彈性叢集，否則您從此入門程序建立的彈性叢集將繼續產生成本。如需說明，請參閱 刪除彈性叢集。

Amazon DocumentDB 彈性叢集最佳實務

了解使用 Amazon DocumentDB 彈性叢集的最佳實務。執行個體型 Amazon DocumentDB 叢集的所有最佳實務也適用於彈性叢集。在新最佳實務確定時，會不斷更新此小節。

主題

- 選擇碎片索引鍵
- 連線管理
- 未碎片的集合
- 擴展彈性叢集
- 監控彈性叢集

選擇碎片索引鍵

下列清單說明建立碎片金鑰的指導方針。

- 使用平均分佈的雜湊金鑰，將資料分佈到叢集中的所有碎片（避免使用熱鍵）。
- 在所有read/update/delete請求中使用您的碎片金鑰，以避免散佈收集查詢。
- 執行read/update/delete操作時，請避免巢狀碎片金鑰。
- 進行批次操作時，請將 `ordered`設定為 false，以便所有碎片可以平行執行並改善延遲。

連線管理

下列清單說明管理資料庫連線的指導方針。

- 監控您的連線計數，以及新連線開啟的頻率。
- 將連線分散到應用程式組態中的所有子網路。如果您的叢集是在多個子網路中設定，但您只利用子網路的子集，您可能會遇到最大連線的瓶頸。

未碎片的集合

以下說明未碎片集合的指導方針。

- 使用未碎片集合時，若要分配負載，請嘗試在不同資料庫上保留高度使用的未碎片集合。Amazon DocumentDB 彈性叢集會將資料庫放在不同的碎片上，並將相同資料庫的未碎片集合共置在同一碎片上。

擴展彈性叢集

下列清單說明擴展彈性叢集的指導方針。

- 擴展操作可能會導致短暫的間歇性資料庫和網路錯誤。如果可能，請避免在尖峰時段進行擴展。嘗試在維護時段期間擴展。
- 擴展和縮減碎片容量（變更每個碎片的 vCPU 計數）以增加運算比增加或減少碎片計數更好，因為它速度更快，並且具有較短的間歇性資料庫和網路錯誤持續時間。
- 預測增長時，最好增加碎片計數，而不是擴展碎片容量。這可讓您透過為需要快速擴展的案例增加碎片容量來擴展叢集。
- 監控用戶端重試政策，並使用指數退避和抖動重試，以避免在擴展時發生錯誤時讓資料庫超載。
- 變更碎片數量可能需要一些時間，因為資料必須安全地從一個碎片移至另一個碎片。為了將錯誤風險降至最低，請勿在變更進行時停止或對叢集狀態進行任何其他變更。

監控彈性叢集

下列清單說明監控彈性叢集的準則。

- 追蹤每個碎片指標的peak-to-average比率，以判斷您是否正在驅動不均勻的流量（具有熱鍵/熱點）。追蹤peak-to-average比率的關鍵指標為：
  - `PrimaryInstanceCPUUtilization`
    - 這可以在每個碎片層級進行監控。
    - 在叢集層級，您可以監控 p99 偏斜的平均值。
  - `PrimaryInstanceFreeableMemory`
    - 這可以在每個碎片層級進行監控。
    - 在叢集層級，您可以監控 p99 偏斜的平均值。
  - `DatabaseCursorsMax`
    - 這應該在每個碎片層級進行監控，以判斷扭曲。
  - `Documents-Inserted/Updated/Returned/Deleted`
    - 這應該在每個碎片層級進行監控，以判斷扭曲。

管理 Amazon DocumentDB 彈性叢集

若要管理 Amazon DocumentDB 彈性叢集，您必須擁有具有適當 Amazon DocumentDB 控制平面許可的 IAM 政策。這些許可可讓您建立、修改和刪除叢集。Amazon DocumentDBFullAccess 政策提供管理 Amazon DocumentDB 彈性叢集的所有必要許可。

下列主題說明如何在使用 Amazon DocumentDB 彈性叢集時執行各種任務。

主題

- 修改彈性叢集組態
- 監控彈性叢集
- 刪除彈性叢集
- 管理彈性叢集快照
- 停止和啟動 Amazon DocumentDB 彈性叢集
- 維護 Amazon DocumentDB 彈性叢集

修改彈性叢集組態

在本節中，我們會說明如何使用 AWS 管理主控台 或 AWS CLI 搭配下列指示來修改彈性叢集。

修改叢集的主要用途是透過增加或減少碎片計數和/或碎片運算容量來擴展碎片。

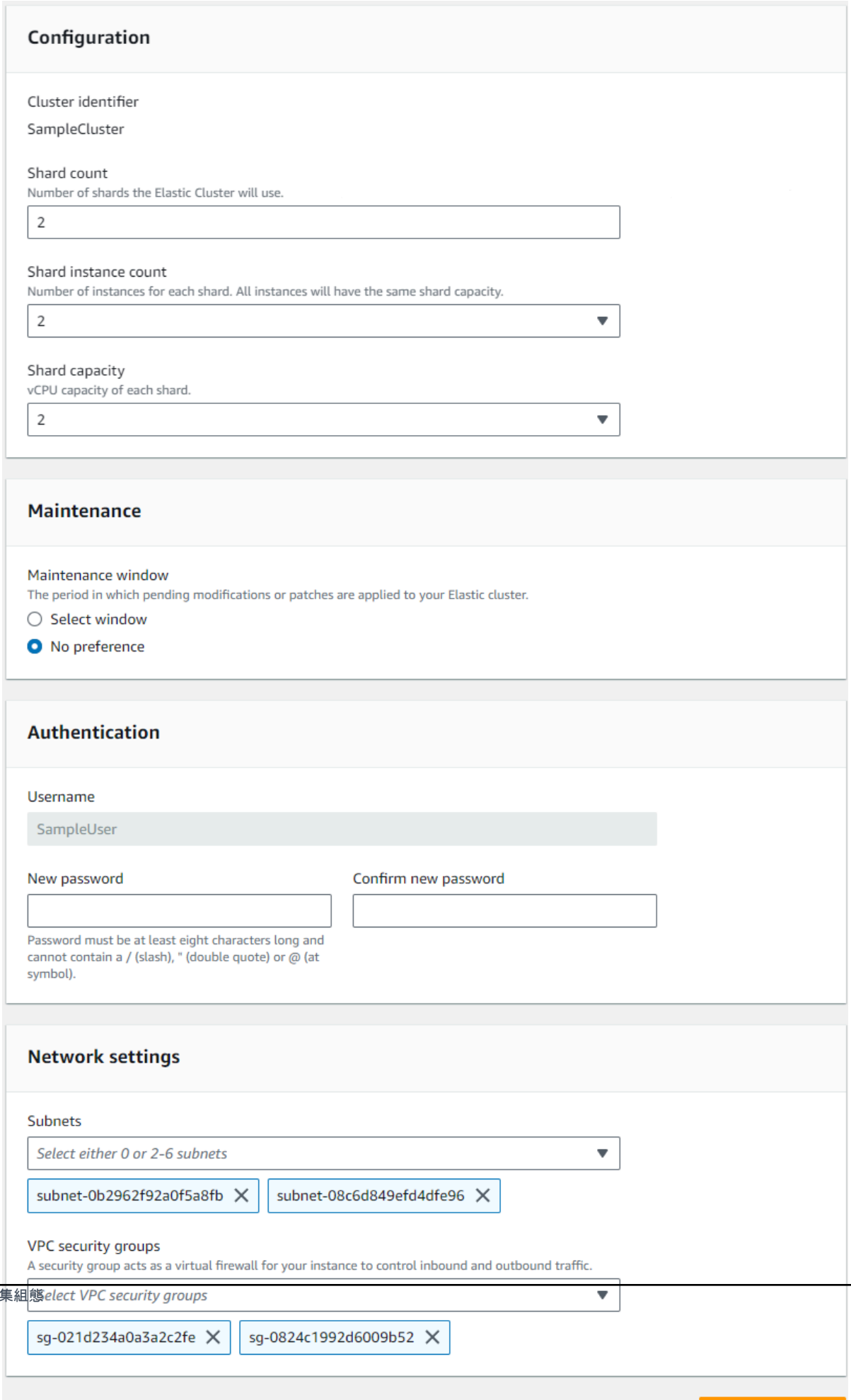
Using the AWS 管理主控台

若要使用 修改彈性叢集組態 AWS 管理主控台：

1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

> **Tip**
>
> 如果您在螢幕左側沒有看到導覽窗格，請選擇導覽窗格左上角的選單圖示。

3. 在叢集識別符欄中選擇您要修改的叢集名稱。
4. 選擇 Modify (修改)。
5. 編輯您要變更的欄位，然後選取修改叢集。


Configuration
Cluster identifier
SampleCluster
Shard count
Number of shards the Elastic Cluster will use.
2
Shard instance count
Number of instances for each shard. All instances will have the same shard capacity.
2
Shard capacity
vCPU capacity of each shard.
2
Maintenance
Maintenance window
The period in which pending modifications or patches are applied to your Elastic cluster.
Select window
No preference
Authentication
Username
SampleUser
New password
Confirm new password
Password must be at least eight characters long and cannot contain a / (slash), " (double quote) or @ (at symbol).
Network settings
Subnets
Select either 0 or 2-6 subnets
subnet-0b2962f92a0f5a8fb
subnet-08c6d849efd4dfe96
VPC security groups
A security group acts as a virtual firewall for your instance to control inbound and outbound traffic.
Select VPC security groups
sg-021d234a0a3a2c2fe
sg-0824c1992d6009b52

> **Note**
>
> 或者，您可以前往叢集頁面，勾選叢集旁的方塊，選擇動作，然後選擇修改，以存取修改叢集對話方塊。

Using the AWS CLI

若要使用 修改彈性叢集組態 AWS CLI，請使用 `update-cluster`操作搭配下列參數：

- **`--cluster-arn`** - 必要項目。您要修改之叢集的 ARN 識別符。
- **`--shard-capacity`**- 選用。指派給每個碎片vCPUs 數量。上限為 64。允許的值為 2、4、8、16、32、64。
- **`--shard-count`**- 選用。指派給叢集的碎片數量。上限為 32。
- **`--shard-instance`**-count - 選用。套用至此叢集中所有碎片的複本執行個體數目。上限為 16。
- **`--auth-type`**- 選用。用來決定在何處擷取用於存取彈性叢集之密碼的身分驗證類型。有效類型為 `PLAIN_TEXT`或 `SECRET_ARN`。
- **`--admin-user-password`**- 選用。與管理員使用者相關聯的密碼。
- **`--vpc-security-group-ids`**- 選用。設定要與此叢集建立關聯的 Amazon EC2 和 Amazon Virtual Private Cloud (VPC) 安全群組清單。
- **`--preferred-maintenance-window`**- 選用。設定每週執行系統維護的時間範圍，以國際標準時間 (UTC) 為單位

  格式為：`ddd:hh24:mi-ddd:hh24:mi`。有效天數 (ddd)：週一、週二、週三、週四、週五、週六、週日

  預設值是從每個 Amazon Web Services 區域的 8 小時時段中隨機選取的 30 分鐘時段，該時段隨機發生在一週中的某一天。

  最短 30 分鐘時段。
- **`--subnet-ids`**- 選用。設定網路子網路 ID。

在下列範例中，將每個`#########`取代為您自己的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic update-cluster \
    --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 \
    --shard-capacity 8 \
    --shard-count 4 \
    --shard-instance-count 3 \
    --admin-user-password testPassword \
    --vpc-security-group-ids ec-65f40350 \
    --subnet-ids subnet-9253c6a3, subnet-9f1b5af9
```

針對 Windows：

```
aws docdb-elastic update-cluster ^
    --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 ^
    --shard-capacity 8 ^
    --shard-count 4 ^
    --shard-instance-count 3 ^
    --admin-user-password testPassword ^
    --vpc-security-group-ids ec-65f40350 ^
    --subnet-ids subnet-9253c6a3, subnet-9f1b5af9
```

若要在修改後監控彈性叢集的狀態，請參閱監控彈性叢集。

監控彈性叢集

在本節中，我們會說明如何使用 AWS 管理主控台 或 AWS CLI 搭配下列指示來監控您的彈性叢集。

Using the AWS 管理主控台

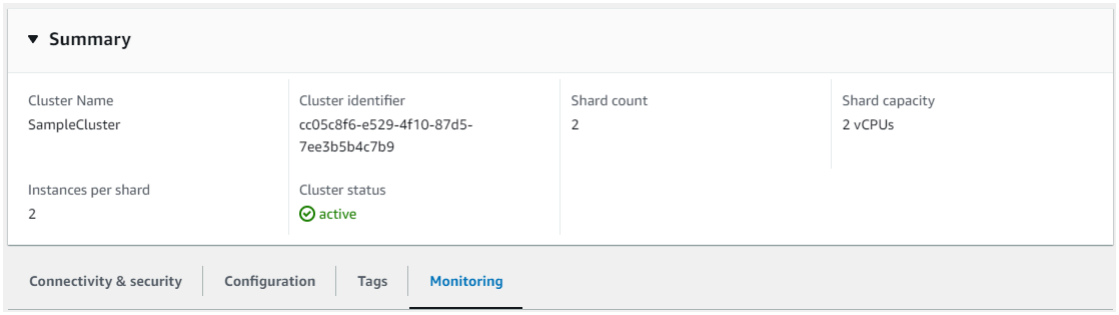
若要使用 監控彈性叢集組態 AWS 管理主控台：

1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

> **Tip**
>
> 如果您在螢幕左側沒有看到導覽窗格，請選擇導覽窗格左上角的選單圖示。

3. 在叢集識別符欄中選擇您要監控的叢集名稱。

4. 選擇 Monitoring (監控) 索引標籤。



下列監控類別會顯示來自 Amazon CloudWatch 的數個圖表：

- 資源使用率
- 輸送量
- 作業
- 系統

您也可以透過 存取 Amazon CloudWatch AWS 管理主控台 ，為您的彈性叢集設定自己的監控環境。

Using the AWS CLI

若要使用 監控特定彈性叢集組態 AWS CLI，請使用 `get-cluster`操作搭配下列參數：

- **`--cluster-arn`** - 必要項目。您想要其資訊的叢集 ARN 識別符。

在下列範例中，將每個*#########*取代為您自己的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic get-cluster \
    --cluster-arn arn:aws:docdb-elastic:us-west-2:123456789012:cluster:/68ffcdf8-e3af-40a3-91e4-24736f2dacc9
```

針對 Windows：

```
aws docdb-elastic get-cluster ^
```

```
    --cluster-arn arn:aws:docdb:-elastic:us-west-2:123456789012:cluster:/68ffcdf8-
e3af-40a3-91e4-24736f2dacc9
```

此操作的輸出如下所示：

```
"cluster": {
        ...
        "clusterArn": "arn:aws:docdb-elastic:us-
west-2:123456789012:cluster:/68ffcdf8-e3af-40a3-91e4-24736f2dacc9",
        "clusterEndpoint": "stretch-11-477568257630.us-east-1.docdb-
elastic.amazonaws.com",
        "readerEndpoint": "stretch-11-477568257630-ro.us-east-1.docdb-
elastic.amazonaws.com",
        "clusterName": "stretch-11",
        "shardCapacity": 2,
        "shardCount": 3,
        "shardInstanceCount: 5,
        "status": "ACTIVE",
        ...
 }
```

如需詳細資訊，請參閱《Amazon DocumentDB Resource Management API 參考DescribeClusterSnapshot》中的 。

若要使用 檢視所有彈性叢集的詳細資訊 AWS CLI，請使用 list-clusters操作搭配下列參數：

- **--next-token**- 選用。如果項目輸出數量 (--max-results) 少於基礎 API 呼叫傳回的項目總數，則輸出會包含一個 NextToken，提供您在後續命令中傳送以擷取下一組項目。
- **--max-results**- 選用。要在命令輸出中傳回的總項目數。如果存在的結果超過指定的max-results值，則回應中會包含分頁字符 (next-token)，以便擷取剩餘的結果。
  - 預設：100
  - 最小值為 20，最大值為 100

在下列範例中，將每個#########取代為您自己的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic list-clusters \
    --next-token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ== \
```

```
    --max-results 2
```

針對 Windows：

```
aws docdb-elastic list-clusters ^
    --next-token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ== ^
    --max-results 2
```

此操作的輸出如下所示：

```
{
   "Clusters": [
      {
         "ClusterIdentifier":"mycluster-1",
         "ClusterArn":"arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster"
         "Status":"available",
         "ClusterEndpoint":"sample-cluster.sharded-cluster-corcjozrlsfc.us-
west-2.docdb.amazonaws.com"
       }
       {
         "ClusterIdentifier":"mycluster-2",
         "ClusterArn":"arn:aws:docdb:us-west-2:987654321098:sharded-cluster:sample-
cluster"
         "Status":"available",
         "ClusterEndpoint":"sample-cluster2.sharded-cluster-corcjozrlsfc.us-
west-2.docdb.amazonaws.com"
       }
   ]
}
```

刪除彈性叢集

在本節中，我們會說明如何使用 AWS 管理主控台 或 AWS CLI 搭配下列指示來刪除彈性叢集。

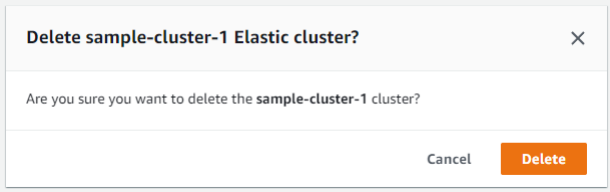
Using the AWS 管理主控台

若要使用 刪除彈性叢集組態 AWS 管理主控台：

1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

> **Tip**
>
> 如果您在螢幕左側沒有看到導覽窗格，請選擇導覽窗格左上角的選單圖示。

3. 在叢集清單表格中，選取您要刪除之叢集名稱左側的核取方塊，然後選擇動作。從下拉式選單中選擇 Detete (刪除)。
4. 在刪除「叢集名稱」彈性叢集中？ 對話方塊，選擇刪除。



刪除叢集需要幾分鐘的時間。若要監控叢集的狀態，請參閱監控 Amazon DocumentDB 叢集的狀態。

Using the AWS CLI

若要使用 刪除彈性叢集 AWS CLI，請使用 `delete-cluster`操作搭配下列參數：

- **`--cluster-arn`** - 必要項目。您要刪除之叢集的 ARN 識別符。
- **`--no-skip-final-backup`**- 選用。如果您想要最終備份，您必須將此參數包含最終備份的名稱。您必須包含 `--final-backup-identifier` 或 `--skip-final-backup`。
- **`--skip-final-backup`**- 選用。只有在您不想在刪除叢集之前進行最終備份時，才使用此參數。預設設定是建立最後的快照。

下列 AWS CLI 程式碼範例使用最終備份刪除 ARN 為 arn：aws：docdb：us-west-2：123456789012：sharded-cluster：sample-cluster 的叢集。

在下列範例中，將每個*#########*取代為您自己的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic delete-cluster \
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-cluster \
    --no-skip-final-backup \
```

```
    --final-backup-identifier finalArnBU-arn:aws:docdb:us-
west-2:123456789012:sharded-cluster:sample-cluster
```

針對 Windows：

```
aws docdb-elastic delete-cluster ^
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster ^
    --no-skip-final-backup ^
    --final-backup-identifier finalArnBU-arn:aws:docdb:us-
west-2:123456789012:sharded-cluster:sample-cluster
```

下列 AWS CLI 程式碼範例會刪除 ARN 為 arn：aws：docdb：us-west-2：123456789012：sharded-cluster：sample-cluster 的叢集，無需進行最終備份。

在下列範例中，將每個*########*取代為您自己的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic delete-cluster \
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster \
    --skip-final-backup \
```

針對 Windows：

```
aws docdb-elastic delete-cluster ^
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster ^
    --skip-final-backup ^
```

`delete-cluster` 操作的輸出是您要刪除之叢集的顯示。

刪除叢集需要幾分鐘的時間。若要監控叢集的狀態，請參閱監控 Amazon DocumentDB 叢集的狀態。

管理彈性叢集快照

建立彈性叢集後，即可取得手動快照。自動備份會在建立彈性叢集快照時建立。

> **Note**
>
> 您的彈性叢集必須處於 `Available` 狀態，才能擷取手動快照。

本節說明如何建立、檢視、還原和刪除彈性叢集快照。

下列主題說明如何在使用 Amazon DocumentDB 彈性叢集快照時執行各種任務。

主題

- 建立手動彈性叢集快照
- 檢視彈性叢集快照
- 從快照還原彈性叢集
- 複製彈性叢集快照
- 刪除彈性叢集快照
- 管理彈性叢集快照自動備份

建立手動彈性叢集快照

在本節中，我們會說明如何使用 AWS 管理主控台 或 AWS CLI 搭配下列指示來建立手動彈性叢集快照。

Using the AWS 管理主控台

若要使用 建立手動彈性叢集快照 AWS 管理主控台：

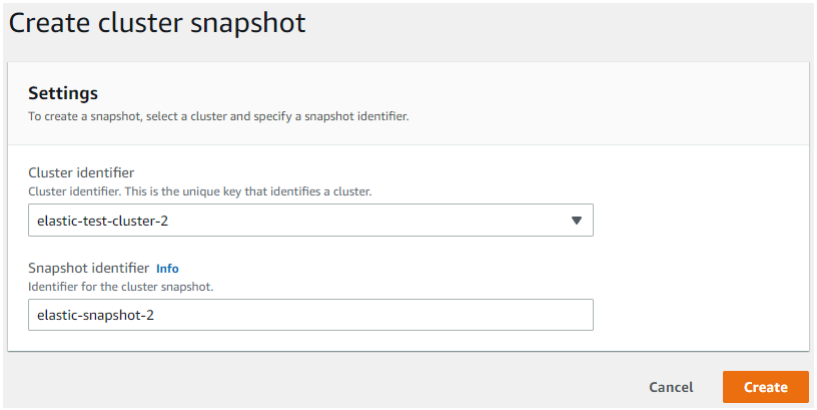
1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Snapshots (快照)。

   > **Tip**
   >
   > 如果您在螢幕左側沒有看到導覽窗格，請選擇導覽窗格左上角的選單圖示。

3. 在 Snapshots (快照) 頁面選擇 Create (建立)。
4. 在建立叢集快照頁面上的叢集識別符欄位中，從下拉式清單中選擇您的彈性叢集。

   在快照識別符欄位中，輸入彈性叢集的唯一識別符。

選擇建立。



> **Note**
>
> 或者，您可以前往叢集頁面，勾選叢集旁的方塊，然後選擇動作，然後拍攝快照，以存取建立叢集快照對話方塊。

您的彈性叢集快照現在正在佈建。這可能需要幾分鐘的時間才能完成。當狀態`Available`在快照清單中顯示為 時，您可以從快照檢視和還原。

Using the AWS CLI

若要使用 建立手動彈性叢集快照 AWS CLI，請使用 `create-cluster-snapshot`操作搭配下列參數：

- **`--snapshot-name`** - 必要項目。您要建立的叢集快照名稱。
- **`--cluster-arn`** - 必要項目。您要建立快照之叢集的 ARN 識別符。

在下列範例中，將每個`#########`取代為您自己的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic create-cluster-snapshot \
    --snapshot-name sample-snapshot-1 \
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster
```

針對 Windows：

```
aws docdb-elastic create-cluster-snapshot ^
    --snapshot-name sample-snapshot-1 ^
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster
```

檢視彈性叢集快照

在本節中，我們會說明如何使用 AWS 管理主控台 或 AWS CLI 搭配下列指示來檢視彈性叢集快照資訊。

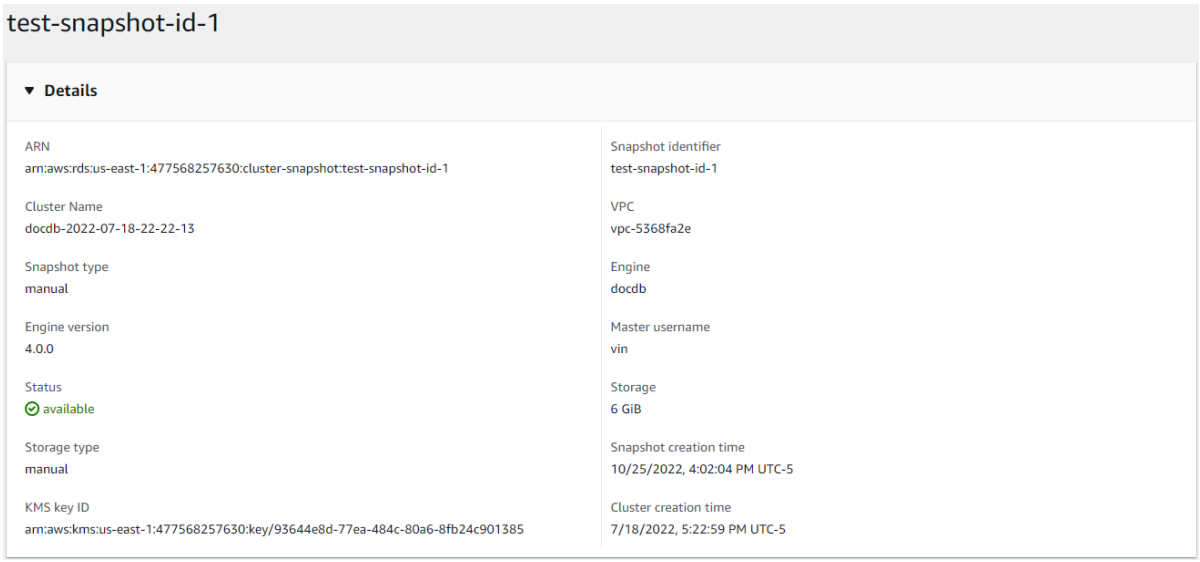
Using the AWS 管理主控台

若要使用 檢視特定彈性叢集快照的相關資訊 AWS 管理主控台：

1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Snapshots (快照)。

> **Tip**
>
> 如果您在螢幕左側沒有看到導覽窗格，請選擇導覽窗格左上角的選單圖示。

3. 在快照頁面上，按一下快照識別符欄中的名稱，從清單中選擇快照。
4. 在詳細資訊中檢視快照的資訊。



Using the AWS CLI

若要使用 檢視特定彈性叢集快照的相關資訊 AWS CLI，請使用 `get-cluster-snapshot`操作搭配下列參數：

- **`--snapshot-arn`** - 必要項目。您想要其資訊的快照 ARN 識別符。

在下列範例中，將每個*#########*取代為您自己的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic get-cluster-snapshot \
    --snapshot-arn sampleResourceName
```

針對 Windows：

```
aws docdb-elastic get-cluster-snapshot ^
    --snapshot-arn sampleResourceName
```

若要使用 檢視特定彈性叢集快照的相關資訊 AWS CLI，請使用 `get-cluster-snapshot`操作搭配下列參數：

- **`--snapshot-arn`** - 必要項目。您想要其資訊的快照 ARN 識別符。

在下列範例中，將每個*#########*取代為您自己的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic get-cluster-snapshot \
    --snapshot-arn sampleResourceName
```

針對 Windows：

```
aws docdb-elastic get-cluster-snapshot ^
    --snapshot-arn sampleResourceName
```

若要使用 檢視所有彈性叢集快照的相關資訊 AWS CLI，請使用 `list-cluster-snapshots`操作搭配下列參數：

- **--snapshot-type**- 選用。要傳回的叢集快照類型。您可以指定下列其中一個值：
  - automated - 傳回 Amazon DocumentDB 為 AWS 您的帳戶自動建立的所有叢集快照。
  - manual - 傳回您為 AWS 帳戶手動建立的所有叢集快照。
  - shared - 傳回已與 AWS 您的帳戶共用的所有手動叢集快照。
  - public - 傳回已標記為公有的所有叢集快照。
- **--next-token**- 選用。前一個請求提供的選用分頁字符。如果指定此參數，回應只會包含超過此字符的記錄，直到 指定的值為止max-results。
- **--max-results**- 選用。要在回應中包含的結果數目上限。如果存在的結果超過指定的max-results值，則回應中會包含分頁字符 (next-token)，以便擷取剩餘的結果。
  - 預設：100
  - 最小值為 20，最大值為 100

在下列範例中，將每個*#########*取代為您自己的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic list-cluster-snapshots \
    --snapshot-type  value \
    --next-token  value \
    --max-results 50
```

針對 Windows：

```
aws docdb-elastic list-cluster-snapshots ^
    --snapshot-type  value ^
    --next-token  value ^
    --max-results 50
```

從快照還原彈性叢集

在本節中，我們會說明如何使用 AWS 管理主控台 或 AWS CLI 搭配下列指示，從快照還原彈性叢集。

Using the AWS 管理主控台

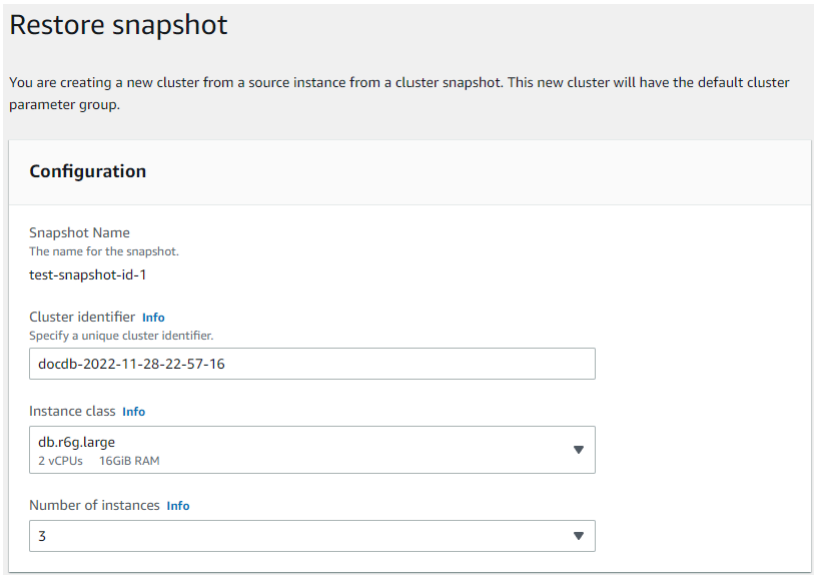
若要使用 從快照還原彈性叢集 AWS 管理主控台：

1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。

2. 在導覽窗格中，選擇 Snapshots (快照)。

> **Tip**
>
> 如果您在螢幕左側沒有看到導覽窗格，請選擇導覽窗格左上角的選單圖示。

3. 在快照識別符欄中，選擇您要用來還原叢集的快照左側按鈕。
4. 選擇動作，再選擇還原。



5. 在還原快照頁面上，在叢集識別符欄位中輸入新叢集的名稱。

> **Note**
>
> 對於任何手動快照還原，您必須建立新的叢集。

6. 在虛擬私有雲端 (VPC) 欄位中，從下拉式清單中選擇 VPC。
7. 對於子網路和 VPC 安全群組，您可以使用預設值或選擇您選擇的三個子網路，以及最多三個 VPC 安全群組 （至少一個）。
8. 如果您滿意此叢集組態，請選擇 Restore cluster (還原叢集)，等待叢集還原。

Using the AWS CLI

若要使用 從快照還原彈性叢集 AWS CLI，請使用 `restore-cluster-from-snapshot`操作搭配下列參數：

- **`--cluster-name`** - 必要項目。在建立或上次修改期間輸入的彈性叢集的目前名稱。

- **--snapshot-arn** - 必要項目。用於還原叢集之快照的 ARN 識別符。
- **--vpc-security-group-ids**- 選用。要與叢集建立關聯的一或多個 Amazon EC2 和 Amazon Virtual Private Cloud (VPC) 安全群組。
- **--kms-key-id**- 選用。設定加密叢集的 KMS 金鑰識別符。

  KMS 金鑰識別符是 AWS KMS 加密金鑰的 Amazon Resource Name (ARN)。如果您使用擁有用來加密新叢集之 KMS 加密金鑰的相同 Amazon Web Services 帳戶來建立叢集，您可以使用 KMS 金鑰別名，而不是 KMS 加密金鑰的 ARN。

  如果未在 KmsKeyId 中指定加密金鑰，且 參數`StorageEncrypted`為 true，Amazon DocumentDB 會使用預設加密金鑰。
- **--subnet-ids**- 選用。網路子網路 ID。

在下列範例中，將每個*#########*取代為您自己的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic restore-cluster-from-snapshot \
     --cluster-name elastic-sample-cluster \
     --snapshot-arn sampleResourceName \
     --vpc-security-group-ids value ec-65f40350 \
     --kms-key-id arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 \
     --subnet-ids subnet-9253c6a3, subnet-9f1b5af9
```

針對 Windows：

```
aws docdb-elastic restore-cluster-from-snapshot ^
     --cluster-name elastic-sample-cluster ^
     --snapshot-arn sampleResourceName ^
     --vpc-security-group-ids value ec-65f40350 ^
     --kms-key-id arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 ^
     --subnet-ids subnet-9253c6a3, subnet-9f1b5af9
```

複製彈性叢集快照

在 Amazon DocumentDB 中，您可以在相同區域和相同帳戶中複製手動和自動彈性叢集快照。在本節中，我們會說明如何使用 AWS 管理主控台 或 複製彈性叢集快照 AWS CLI。

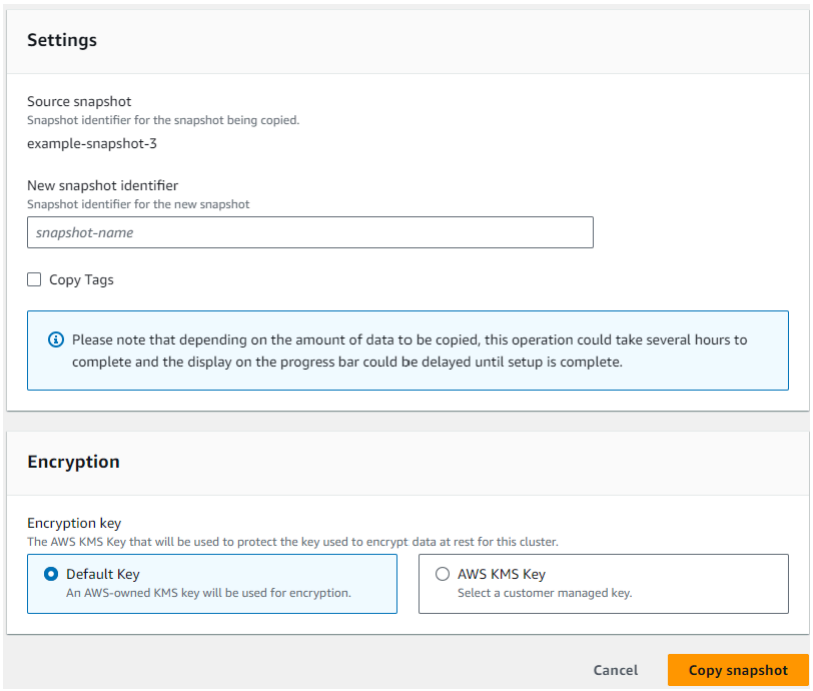
Using the AWS 管理主控台

若要使用 複製彈性叢集快照 AWS 管理主控台：

1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇 Snapshots (快照)。

> **Tip**
>
> 如果您在螢幕左側沒有看到導覽窗格，請選擇導覽窗格左上角的選單圖示。

3. 在快照識別符欄中，選擇您要複製的快照左側按鈕。
4. 選擇動作，然後選擇複製。



5. 針對新快照識別符，輸入新快照的名稱。
6. 對於複製標籤，如果您想要將所有標籤從來源彈性叢集快照複製到目標彈性叢集快照，請勾選此方塊。
7. 針對加密，選擇您選擇的預設 AWS KMS 金鑰或 KMS 金鑰。第二個選項可讓您選取已建立的現有 KMS 金鑰，或允許您建立新的 KMS 金鑰。
8. 完成後，請選擇複製快照。

Using the AWS CLI

若要使用 複製彈性叢集快照 AWS CLI，請使用 `copy-cluster-snapshot`操作搭配下列參數：

- **`--source-db-cluster-snapshot-identifier`** - 必要項目。要複製之現有彈性叢集快照的識別符。彈性叢集快照必須存在且處於可用狀態。此參數不區分大小寫。
- **`--target-db-cluster-snapshot-identifier`** - 必要項目。要從現有叢集快照建立之新彈性叢集快照的識別符。此參數不區分大小寫。

  目標快照名稱限制條件：
  - 不能是現有快照的名稱。
  - 長度為 【1-63】 個字母、數字或連字號。
  - 第一個字元必須是字母。
  - 不能以連字號結尾，或包含兩個連續連字號。

在下列範例中，將每個*#########*取代為您自己的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic copy-cluster-snapshot \
     --source-cluster-snapshot-arn <sample ARN> \
     --target-cluster-snapshot-name my-target-copied-snapshot
```

針對 Windows：

```
aws docdb-elastic copy-cluster-snapshot ^
     --source-cluster-snapshot-arn <sample ARN> ^
     --target-cluster-snapshot-name my-target-copied-snapshot
```

刪除彈性叢集快照

在本節中，我們會說明如何使用 AWS 管理主控台 或 刪除彈性叢集快照 AWS CLI。

Using the AWS 管理主控台

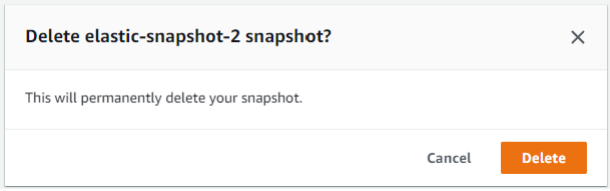
若要使用 從快照還原彈性叢集 AWS 管理主控台：

1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。

2. 在導覽窗格中，選擇 Snapshots (快照)。

> **Tip**
>
> 如果您在螢幕左側沒有看到導覽窗格，請選擇導覽窗格左上角的選單圖示。

3. 在快照識別符欄中，選擇您要用來還原叢集的快照左側按鈕。
4. 依序選擇動作和刪除。



5. 在刪除「snapshot-name」快照對話方塊中，選擇刪除。

Using the AWS CLI

若要使用 刪除彈性叢集快照 AWS CLI，請使用 `delete-cluster-snapshot`操作搭配下列參數：

- **`--snapshot-arn`** - 必要項目。用於還原叢集之快照的 ARN 識別符。

在下列範例中，將每個*#########*取代為您自己的資訊。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic delete-cluster-snapshot \
     --snapshot-arn sampleResourceName
```

針對 Windows：

```
aws docdb-elastic delete-cluster-snapshot ^
     --snapshot-arn sampleResourceName
```

管理彈性叢集快照自動備份

Amazon DocumentDB 會為您的彈性叢集拍攝每日快照。您可以在新的或現有的彈性叢集快照組態中指定偏好的備份時段和備份保留期間。在本節中，我們會說明如何使用 AWS 管理主控台 或 ，在彈性叢集快照中設定自動備份參數 AWS CLI。

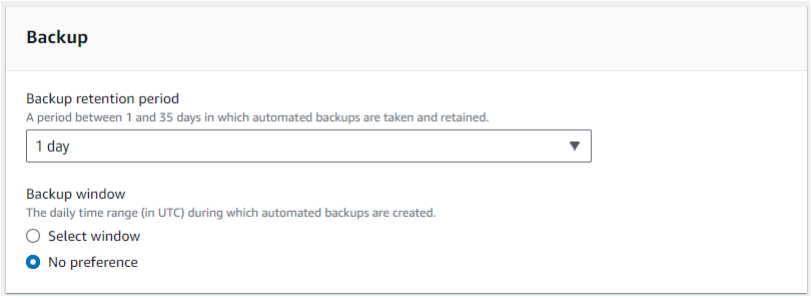
Using the AWS 管理主控台

若要使用 設定新彈性叢集快照的自動備份 AWS 管理主控台：

1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

> **Tip**
>
> 如果您在螢幕左側沒有看到導覽窗格，請選擇導覽窗格左上角的選單圖示。

3. 在叢集識別符欄中，選擇您要變更其備份設定的叢集左側按鈕。
4. 選擇動作，然後選擇修改。
5. 在備份區段中，根據您的備份需求編輯欄位。



   a. 備份保留期 - 在清單中，選擇在刪除叢集之前保留此叢集自動備份的天數。
   b. 備份時段 - 設定 Amazon DocumentDB 要備份此叢集的每日時間和持續時間。
      i. 如果您想要設定建立備份的時間和持續時間，請選擇選取視窗。

         開始時間 - 在第一個清單中，選擇啟動自動備份的開始時間小時 (UTC)。在第二個清單中，選擇您要自動備份開始進行的時間 (分)。

         持續時間 - 在清單中，選擇要配置給建立自動備份的時數。
      ii. 如果您希望 Amazon DocumentDB 選擇建立備份的時間和持續時間，請選擇無偏好設定。

6. 完成時選擇修改叢集。

Using the AWS CLI

若要使用 設定新彈性叢集快照的自動備份 AWS CLI，請使用 `create-cluster-snapshot`操作搭配下列參數：

- **`--preferred-backup-window`**- 選用。建立自動備份的每日偏好時間範圍。預設值是從 8 小時時段中隨機選取的 30 分鐘時段 AWS 區域。

  限制條件：

  - 格式必須為 `hh24:mi-hh24:mi`。
  - 必須以國際標準時間 (UTC) 表示。
  - 不得和慣用的維護時段衝突。
  - 必須至少 30 分鐘。
- **`--backup-retention-period`**- 選用。自動備份保留的天數。預設值為 1.

  限制條件：

  - 必須指定最小值 1。
  - 範圍從 1 到 35。

> **Note**
>
> 只有在叢集處於「作用中」狀態時，才會進行自動備份。

> **Note**
>
> 您也可以使用 `aws docdb-elastic update-cluster`命令修改現有彈性叢集的 `preferred-backup-window`和 `backup-retention-period` 參數。

在下列範例中，將每個*#########*取代為您自己的資訊。

下列`create-cluster`範例會建立 Amazon DocumentDB 彈性叢集 *`sample-cluster`*，其自動備份的保留期間為 *`7`* 天，而偏好的備份時段為 *`18#00-18#30 UTC`*。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic create-cluster \
     --cluster-name sample-cluster \
     --shard-capacity 2 \
     --shard-count 2 \
     --admin-user-name SampleAdmin \
     --auth-type PLAIN_TEXT \
     --admin-user-password SamplePass123! \
     --preferred-backup-window 18:00-18:30 \
     --backup-retention-period 7
```

針對 Windows：

```
aws docdb-elastic create-cluster ^
     --cluster-name sample-cluster ^
     --shard-capacity 2 ^
     --shard-count 2 ^
     --admin-user-name SampleAdmin ^
     --auth-type PLAIN_TEXT ^
     --admin-user-password SamplePass123! ^
     --preferred-backup-window 18:00-18:30 ^
     --backup-retention-period 7
```

停止和啟動 Amazon DocumentDB 彈性叢集

停止和啟動 Amazon DocumentDB 彈性叢集可協助您管理開發和測試環境的成本。您不需要在每次使用 Amazon DocumentDB 時建立和刪除彈性叢集，而是可以在不需要叢集時暫時停止叢集。然後，您可以在繼續測試時再次啟動它。

主題

- 停止和啟動彈性叢集的概觀
- 您可以在已停止的彈性叢集上執行的操作

停止和啟動彈性叢集的概觀

在您不需要 Amazon DocumentDB 彈性叢集的期間，您可以停止叢集。一旦您需要叢集，即可隨時重新啟動它。啟動和停止可簡化彈性叢集的設定和縮減程序，用於不需要持續可用性的開發、測試或類似活動。您可以使用 AWS 管理主控台 或 AWS CLI 單一動作來停止和啟動彈性叢集。

當您的彈性叢集停止時，叢集儲存磁碟區會保持不變。您只需支付指定的保留時段內儲存、手動快照和自動備份儲存的費用。Amazon DocumentDB 會在七天後自動啟動您的彈性叢集，使其不會落後於任何必要的維護更新。當您的叢集在七天後啟動時，將會再次開始向您收取使用彈性叢集的費用。當叢集停止時，您無法查詢儲存磁碟區，因為查詢需要叢集處於可用狀態。

當 Amazon DocumentDB 彈性叢集停止時，無法以任何方式修改叢集。這包括刪除叢集。

Using the AWS 管理主控台

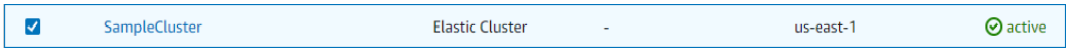
下列程序說明如何在可用狀態下停止彈性叢集，或啟動已停止的彈性叢集。

停止或啟動 Amazon DocumentDB 彈性叢集

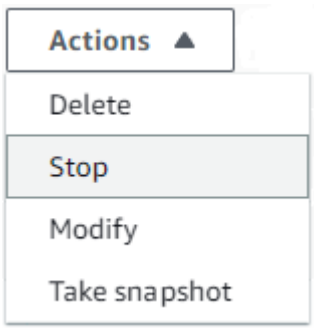
1. 登入 AWS 管理主控台，並在 https：//https://console.aws.amazon.com/docdb 開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。

> **Tip**
>
> 如果畫面左側沒有出現導覽窗格，請選擇頁面左上角的功能表圖示 (≡)。

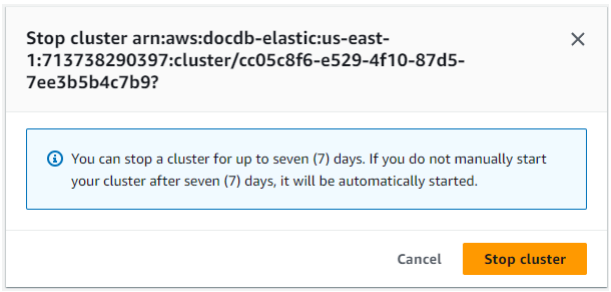
3. 在叢集清單中，選擇您要停止或啟動之叢集名稱左側的按鈕。



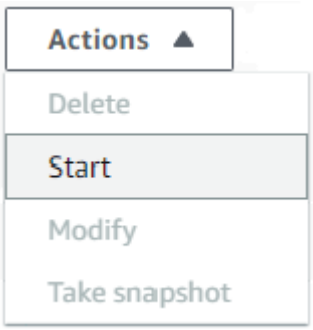
4. 選擇 Actions (動作)，然後選擇您想要在叢集上執行的動作。
   - 如果您想要停止可用的叢集：
     a. 選擇停止。



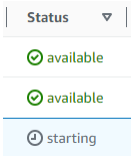
     b. 在確認對話方塊中，選擇停止叢集來確認您想要停止彈性叢集，或選擇取消以保持叢集執行。



- 如果您想要啟動叢集，而叢集已停止，請選擇 Start (啟動)。



5. 監控彈性叢集的狀態。如果您啟動叢集，您可以在叢集可用時繼續使用叢集。如需詳細資訊，請參閱判斷叢集的狀態。



Using the AWS CLI

下列程式碼範例示範如何停止處於作用中或可用狀態的彈性叢集，或啟動已停止的彈性叢集。

若要使用 停止彈性叢集 AWS CLI，請使用 `stop-cluster`操作。若要啟動已停止的叢集，請使用 `start-cluster` 操作。這兩個操作均使用 `--cluster-arn` 參數。

參數：

- **`--cluster-arn`** - 必要項目。您要停止或啟動之彈性叢集的 ARN 識別符。

Example— 使用 停止彈性叢集 AWS CLI

在下列範例中，將每個`#########`取代為您自己的資訊。

下列程式碼會停止 ARN 為 的彈性叢集arn:aws:docdb-elastic:us-east-1:477568257630:cluster/b9f1d489-6c3e-4764-bb42-da62ceb7bda2。

> **Note**
>
> 彈性叢集必須處於作用中或可用狀態。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic stop-cluster \
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

針對 Windows：

```
aws docdb-elastic stop-cluster ^
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

Example— 使用 啟動彈性叢集 AWS CLI

在下列範例中，將每個#########取代為您自己的資訊。

下列程式碼會以 的 ARN 啟動彈性叢集arn:aws:docdb-elastic:us-east-1:477568257630:cluster/b9f1d489-6c3e-4764-bb42-da62ceb7bda2。

> **Note**
>
> 彈性叢集目前必須停止。

針對 Linux、macOS 或 Unix：

```
aws docdb-elastic start-cluster \
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

針對 Windows：

```
aws docdb-elastic start-cluster ^
```

```
    --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

您可以在已停止的彈性叢集上執行的操作

當叢集停止時，您無法修改 Amazon DocumentDB 彈性叢集的組態。您必須先啟動叢集，然後才能執行任何這類管理動作。

Amazon DocumentDB 只有在重新啟動後，才會將任何排定的維護套用至已停止的彈性叢集。七天後，Amazon DocumentDB 會自動啟動已停止的彈性叢集，使其不會落後維護狀態太久。當彈性叢集重新啟動時，您將再次開始支付叢集中的碎片費用。

當彈性叢集停止時，Amazon DocumentDB 不會執行任何自動備份，也不會延長備份保留期。

維護 Amazon DocumentDB 彈性叢集

主題

- 檢視待定的彈性叢集維護動作
- 彈性叢集引擎更新
- 彈性叢集作業系統更新

Amazon DocumentDB 會定期對 Amazon DocumentDB 彈性叢集資源執行維護。維護通常涉及資料庫引擎的更新（彈性叢集維護）或彈性叢集的基礎作業系統 (OS 更新）。資料庫引擎更新是必要的修補程式，包括資料庫引擎的安全性修正、錯誤修正和增強功能。雖然大多數作業系統修補程式都是選用的，但如果您有一段時間未套用，則可能需要修補程式並自動套用，以維護您的安全狀態。在 Amazon DocumentDB 彈性叢集可用時，立即將作業系統更新套用至叢集。

資料庫引擎修補程式需要您讓 Amazon DocumentDB 彈性叢集短暫離線。一旦可用，這些修補程式會自動排程在 Amazon DocumentDB 彈性叢集的近期排程維護時段套用。

彈性叢集有自己的個別維護時段。您已選擇不立即套用的彈性叢集修改會在維護時段期間套用。根據預設，當您建立彈性叢集時，Amazon DocumentDB 會為您的彈性叢集指派維護時段。您可以在建立彈性叢集時選擇維護時段。您也可以隨時修改維護時段，使符合您的業務排程或實務。我們通常會建議您選擇對應用程式影響最低的維護時段 (例如，在晚上或週末)。

檢視待定的彈性叢集維護動作

您可以使用 檢視您的彈性叢集是否可使用維護更新 AWS CLI。

如有更新可用，您可執行下列操作之一：

- 延遲目前排程用於下一個維護時段的維護動作（僅適用於作業系統修補程式）。
- 立即套用維護動作。
- 排程在下次的維護時段啟動維護動作。
- 排程要在所選套用時段內啟動的維護動作。

維護時段會判斷待定的操作何時開始，但不限制這些操作的總執行時間。

使用下列 AWS CLI 操作來判斷待定的維護動作。列出所有待定的維護動作：

```
aws docdb-elastic list-pending-maintenance-actions
```

此操作的輸出如下所示 (JSON 格式）：

```
{
'ResourcePendingMaintenanceActions': [
    {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'ENGINE_UPDATE',
                'AutoAppliedAfterDate': 'string',
                'ForcedApplyDate': 'string',
                'OptInStatus': 'string',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            },
        ]
    },
],
'NextToken': 'string'
}
```

取得指定 上的待定維護動作（如果有的話）`resourceArn`：

```
aws docdb-elastic get-pending-maintenance-action --resource-arn string-arn
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    'ResourcePendingMaintenanceAction': {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'ENGINE_UPDATE',
                'AutoAppliedAfterDate': 'string',
                'ForcedApplyDate': 'string',
                'OptInStatus': 'string',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            }
        ]
    }
}
```

參數：

- ResourceArn- 套用待定維護動作之資源的 Amazon DocumentDB Amazon Resource Name (ARN)。
- Action- 套用至資源的待定維護動作。

  有效值：

  - ENGINE_UPDATE
  - ENGINE_UPGRADE
  - SECURITY_UPDATE
  - OS_UPDATE
  - MASTER_USER_PASSWORD_UPDATE
- AutoAppliedAfterDate- 此日期之後的第一個維護時段。在此情況下NEXT_MAINTENANCE OPT_IN會忽略 。
- ForcedApplyDate- 無論維護時段為何都會套用。在此情況下IMMEDIATE OPT_IN， 會被忽略。
- OptInStatus- 指定選擇加入請求類型的值，或復原選擇加入請求。無法復原 IMMEDIATE 類型的選擇使用請求。

  有效值：

  - IMMEDIATE- 立即套用維護動作。
  - NEXT_MAINTENANCE- 在資源的下一個維護時段套用維護動作。

  - APPLY_ON- 在指定的套用日期套用維護動作，無論資源的下一個維護時段為何。
  - UNDO_OPT_IN- 取消任何現有的請求NEXT_MAINTENANCE或APPLY_ON選擇加入請求。
- CurrentApplyDate- 如果opt-in-type為 則會顯示APPLY_ON。
- Description- 維護動作的選項描述。

彈性叢集引擎更新

使用 Amazon DocumentDB，您可以選擇何時套用維護操作。您可以使用 決定 Amazon DocumentDB 何時套用更新 AWS CLI。

套用待定的維護動作：

```
aws docdb-elastic apply-pending-maintenance-action
--resource-arn string-arn
--apply-action string-enum
--opt-in-type string-enum
[--apply-on string-date-range]
```

參數：

- **--resource-arn**- 套用待定維護動作之資源的 Amazon DocumentDB Amazon Resource Name (ARN)。
- **--apply-action**- 要套用至此資源的待定維護動作。

  有效值：

  - ENGINE_UPDATE
  - ENGINE_UPGRADE
  - SECURITY_UPDATE
  - OS_UPDATE
  - MASTER_USER_PASSWORD_UPDATE
- **--opt-in-type**- 指定選擇加入請求類型的值，或復原選擇加入請求。無法復原 IMMEDIATE 類型的選擇使用請求。

  有效值：

  - IMMEDIATE- 立即套用維護動作。
  - NEXT_MAINTENANCE- 在資源的下一個維護時段套用維護動作。

  - APPLY_ON- 在指定的套用日期套用維護動作，無論資源的下一個維護時段為何。
  - UNDO_OPT_IN- 取消任何現有的請求NEXT_MAINTENANCE或APPLY_ON選擇加入請求。
- **[--apply-on]**- 如果opt-in-type為 則為必要APPLY_ON。格式：yyyy/MM/dd HH:mm-yyyy/MM/dd HH:mm (此選項使用 UTC 時間。 開始時間可以是未來任何時間，從最少 30 分鐘到最多 14 天，或對待定動作強制/套用日期，以較早者為準。 從開始到結束時間的時段最短可達 30 分鐘，最長可達 8 小時。)

此操作的輸出如下所示 (JSON 格式) ：

```
{
 'ResourcePendingMaintenanceAction': {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'SECURITY_UPDATE',
                'AutoAppliedAfterDate': 'string',
                'ForcedApplyDate': 'string',
                'OptInStatus': 'IMMEDIATE',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            },
        ]
 }
}
```

參數：

- ResourceArn- 套用待定維護動作之資源的 Amazon DocumentDB Amazon Resource Name (ARN)。
- Action- 套用至資源的待定維護動作。

  有效值：

  - ENGINE_UPDATE
  - ENGINE_UPGRADE
  - SECURITY_UPDATE
  - OS_UPDATE
  - MASTER_USER_PASSWORD_UPDATE

- AutoAppliedAfterDate- 此日期之後的第一個維護時段。在此情況下NEXT_MAINTENANCE OPT_IN會忽略 。
- ForcedApplyDate- 無論維護時段為何都會套用。在此情況下IMMEDIATE OPT_IN， 會被忽略。
- OptInStatus- 指定選擇加入請求類型的值，或復原選擇加入請求。無法復原 IMMEDIATE 類型的選擇使用請求。

  有效值：

  - IMMEDIATE- 立即套用維護動作。
  - NEXT_MAINTENANCE- 在資源的下一個維護時段套用維護動作。
  - APPLY_ON- 在指定的套用日期套用維護動作，無論資源的下一個維護時段為何。
  - UNDO_OPT_IN- 取消任何現有的請求NEXT_MAINTENANCE或APPLY_ON選擇加入請求。
- CurrentApplyDate- 如果opt-in-type為 則會顯示APPLY_ON。
- Description- 維護動作的選項描述。

套用日期

每個維護動作都有各自的套用日期，您可以在描述待定維護動作時找到。當您從 讀取待定維護動作的輸出時 AWS CLI，會列出三個日期：

- CurrentApplyDate- 將立即套用維護動作的日期，或在下一個維護時段套用。如果維護是選用的，則此值可以是 null。
- ForcedApplyDate- 自動套用維護的日期，與您的維護時段無關。
- AutoAppliedAfterDate- 在叢集的維護時段期間套用維護的日期。

使用者建立的維護動作

身為 Amazon DocumentDBelastic 叢集使用者，您可以啟動叢集組態的更新。

更新叢集主要密碼

```
aws docdb-elastic update-cluster
--cluster-arn string-arn
[--admin-user-password string]
[--auth-type string-enum]
[--apply-method string-enum]
[--apply-on string-date-range]
```

```
#... other parameters of the API that follow here are not relevant for this
 configuration
```

參數：

- **--cluster-arn**- 要套用維護動作之資源的 Amazon DocumentDB Amazon Resource Name (ARN)。
- **[--admin-user-password]**- 與管理員使用者相關聯的密碼。
- **[--auth-type]**- 用來決定在何處擷取用於存取彈性叢集之密碼的身分驗證類型。有效類型為 PLAIN_TEXT或 SECRET_ARN。
- **[--apply-method]**- 指定所套用方法類型的值。允許的值為 IMMEDIATE 和 APPLY_ON。預設值為 IMMEDIATE。
- **[--apply-on]**- 如果 apply-method為 則為必要APPLY_ON。格式：yyyy/MM/dd HH:mm-yyyy/MM/dd HH:mm（此選項使用 UTC 時間。 開始時間可以是未來任何時間，從最少 30 分鐘到最多 14 天。 從開始到結束時間的時段最短可達 30 分鐘，最長可達 8 小時。）

此操作的輸出如下所示 (JSON 格式）：

```
{
 'ResourcePendingMaintenanceAction': {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'MASTER_USER_PASSWORD_UPDATE',
                'OptInStatus': 'APPLY_ON',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            },
        ]
 }
}
```

變更 Amazon DocumentDB 維護時段

維護時段應落在使用量最低的時段，因此可能需要不時進行變更。只有在套用系統變更（例如擴展儲存操作變更）且需要中斷時，您的彈性叢集才會在此期間無法使用。它只能在進行必要變更所需的最短時間內無法使用。

預設值是從每個 Amazon Web Services 區域的 8 小時時段中隨機選取的 30 分鐘時段，該時段隨機發生在一週中的某一天。

若要變更維護時段，請參閱 修改彈性叢集組態。

彈性叢集作業系統更新

Amazon DocumentDB 彈性叢集偶爾需要作業系統更新。Amazon DocumentDB 會將作業系統升級至較新的版本，以改善資料庫效能和客戶的整體安全狀態。作業系統更新不會變更 Amazon DocumentDB 彈性叢集的叢集引擎版本。

Amazon DocumentDB 彈性叢集的大多數作業系統更新都是選用的，而且沒有要套用它們的設定日期。不過，如果您有一段時間沒有套用這些更新，它們最終可能會變成必要，並在叢集維護時段期間自動套用。這是為了協助維護資料庫的安全狀態。為了避免意外停機，請在 Amazon DocumentDB 彈性叢集可用時立即套用作業系統更新，並在符合您業務需求的時間設定叢集維護時段。

Amazon DocumentDB 彈性叢集的靜態資料加密

下列主題可協助您了解、建立和監控 Amazon DocumentDB 彈性叢集的 AWS Key Management Service 加密金鑰：

主題

- Amazon DocumentDB 彈性叢集如何在 中使用授予 AWS KMS
- 建立客戶自管金鑰
- 監控 Amazon DocumentDB 彈性叢集的加密金鑰
- 進一步了解

Amazon DocumentDB 彈性叢集會自動與 AWS Key Management Service (AWS KMS) 整合以進行金鑰管理，並使用稱為信封加密的方法來保護您的資料。如需封套加密的詳細資訊，請參閱 AWS Key Management Service 開發人員指南中的封套加密。

AWS KMS key 是金鑰的邏輯表示法。KMS 金鑰包含金鑰 ID、建立日期、說明和金鑰狀態等中繼資料。KMS 金鑰也包含可用來加密和解密資料的金鑰材料。如需 KMS 金鑰的詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的 AWS KMS keys。

Amazon DocumentDB 彈性叢集支援使用兩種類型的金鑰加密：

- AWS 擁有的金鑰 — Amazon DocumentDB 彈性叢集預設使用這些金鑰自動加密個人身分識別資料。您無法檢視、管理或使用擁有 AWS的金鑰，或稽核其使用方式。不過，您不需要採取任何動

作或變更任何程式，即可保護加密您資料的金鑰。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的 AWS 擁有的金鑰。

- 客戶受管金鑰 — 您建立、擁有和管理 AWS KMS keys 的對稱。您可以完全控制此層加密，因此能執行以下任務：
  - 建立和維護金鑰政策
  - 建立和維護 IAM 政策和授予操作
  - 啟用和停用金鑰政策
  - 輪換金鑰密碼編譯資料
  - 新增 標籤
  - 建立金鑰別名
  - 安排金鑰供刪除

  如需更多資訊，請參閱 AWS Key Management Service 開發人員指南中的客戶受管金鑰。

Important

您必須使用對稱加密 KMS 金鑰來加密叢集，因為 Amazon DocumentDB 僅支援對稱加密 KMS 金鑰。請勿使用非對稱 KMS 金鑰來嘗試加密 Amazon DocumentDB 彈性叢集中的資料。如需詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》中的 中的非對稱金鑰 AWS KMS。

如果 Amazon DocumentDB 無法再存取叢集的加密金鑰，例如，當金鑰的存取遭到撤銷時，加密的叢集會進入結束狀態。在此情況下，您只能從備份中還原叢集。對於 Amazon DocumentDB，備份一律啟用 1 天。此外，如果您停用加密 Amazon DocumentDB 叢集的 金鑰，最終將會失去該叢集的讀取和寫入存取權。當 Amazon DocumentDB 遇到由無法存取的金鑰加密的叢集時，它會讓叢集進入終端狀態。在此情況下，該叢集再也無法使用，而且無法復原資料庫的目前狀態。若要還原叢集，您必須重新啟用對 Amazon DocumentDB 加密金鑰的存取，然後從備份還原叢集。

Important

建立加密叢集之後，就無法變更該叢集的 KMS 金鑰。建立加密的彈性叢集之前，請務必判斷您的加密金鑰需求。

Amazon DocumentDB 彈性叢集如何在 中使用授予 AWS KMS

Amazon DocumentDB 彈性叢集需要授予才能使用客戶受管金鑰。

當您建立使用客戶受管金鑰加密的叢集時，Amazon DocumentDB 彈性叢集會透過傳送`CreateGrant`請求至 來代表您建立授予 AWS KMS。中的授予 AWS KMS 用於授予 Amazon DocumentDB 彈性叢集存取客戶帳戶中 KMS 金鑰的權限。

Amazon DocumentDB 彈性叢集需要授予 ，才能將客戶受管金鑰用於下列內部操作：

- 傳送`DescribeKey`請求至 ， AWS KMS 以確認建立彈性叢集時輸入的對稱客戶受管金鑰 ID 是否有效。
- 將`GenerateDataKey`請求傳送至 AWS KMS ，以產生由客戶受管金鑰加密的資料金鑰。
- 將`Decrypt`請求傳送至 AWS KMS 以解密加密的資料金鑰，以便用來加密您的資料。
- 您可以隨時撤銷授予的存取權，或移除服務對客戶受管金鑰的存取權。如果您這麼做，Amazon DocumentDB 彈性叢集將無法存取客戶受管金鑰加密的任何資料，這會影響相依於該資料的操作。

建立客戶自管金鑰

您可以使用 AWS 管理主控台 或 AWS KMS API 來建立對稱客戶受管金鑰。

建立對稱客戶受管金鑰

遵循《 AWS Key Management Service 開發人員指南》中建立對稱客戶受管金鑰的步驟。

金鑰政策

金鑰政策會控制客戶受管金鑰的存取權限。每個客戶受管金鑰都必須只有一個金鑰政策，其中包含決定誰可以使用金鑰及其使用方式的陳述式。在建立客戶受管金鑰時，可以指定金鑰政策。如需詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》AWS Key Management Service 概觀中的 KMS 金鑰存取資訊。

若要將客戶受管金鑰與 Amazon DocumentDB 彈性叢集資源搭配使用，必須在金鑰政策中允許下列 API 操作：

- `kms:CreateGrant`：新增客戶受管金鑰的授權。授予控制對指定 KMS 金鑰的存取，允許存取 Amazon DocumentDB 彈性叢集所需的授予操作。如需使用授與的詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》中的 中的授與 AWS KMS。

- kms:DescribeKey – 提供客戶受管金鑰詳細資訊，以允許 Amazon DocumentDB 彈性叢集驗證金鑰。
- kms:Decrypt – 允許 Amazon DocumentDB 彈性叢集使用儲存的加密資料金鑰來存取加密的資料。
- kms:GenerateDataKey – 允許 Amazon DocumentDB 彈性叢集產生加密的資料金鑰並將其存放，因為資料金鑰不會立即用於加密。

如需詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》中的金鑰政策中的 AWS 服務許可和對金鑰存取進行故障診斷。

透過 IAM 政策限制客戶受管金鑰存取

除了 KMS 金鑰政策之外，您也可以在 IAM 政策中限制 KMS 金鑰許可。

您可以透過各種方式使 IAM 政策更加嚴格。例如，若要允許客戶受管金鑰僅用於源自 Amazon DocumentDB 彈性叢集的請求，您可以使用 kms:ViaService條件金鑰搭配 docdb-elastic.<region-name>.amazonaws.com值。

如需詳細資訊，請參閱 AWS Key Management Service 開發人員指南中的允許其他帳戶中的使用者使用 KMS 金鑰。

監控 Amazon DocumentDB 彈性叢集的加密金鑰

當您搭配 Amazon DocumentDB 彈性叢集資源使用客戶受管金鑰時，您可以使用 AWS CloudTrail 或 Amazon CloudWatch Logs 來追蹤 Amazon DocumentDB 彈性叢集傳送的請求 AWS KMS。

下列範例是 CreateGrant、Decrypt、 GenerateDataKeyWithoutPlainText和 AWS CloudTrail 的事件DescribeKey，用於監控 Amazon DocumentDB 彈性叢集呼叫 AWS KMS key 的操作，以存取客戶受管金鑰加密的資料：

CreateGrant

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
```

```
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-05-09T23:04:20Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "docdb-elastic.amazonaws.com"
    },
    "eventTime": "2023-05-09T23:55:48Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "docdb-elastic.amazonaws.com",
    "userAgent": "docdb-elastic.amazonaws.com",
    "requestParameters": {
        "retiringPrincipal": "docdb-elastic.us-east-1.amazonaws.com",
        "granteePrincipal": "docdb-elastic.us-east-1.amazonaws.com",
        "operations": [
            "Decrypt",
            "Encrypt",
            "GenerateDataKey",
            "GenerateDataKeyWithoutPlaintext",
            "ReEncryptFrom",
            "ReEncryptTo",
            "CreateGrant",
            "RetireGrant",
            "DescribeKey"
        ],
        "keyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
```

```
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

GenerateDataKey

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-05-10T18:02:59Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "docdb-elastic.amazonaws.com"
    },
```

```
    "eventTime": "2023-05-10T18:03:25Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "docdb-elastic.amazonaws.com",
    "userAgent": "docdb-elastic.amazonaws.com",
    "requestParameters": {
        "keySpec": "AES_256",
        "keyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Decrypt

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
```

```
                "arn": "arn:aws:iam::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-05-10T18:05:49Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "docdb-elastic.amazonaws.com"
    },
    "eventTime": "2023-05-10T18:06:19Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "docdb-elastic.amazonaws.com",
    "userAgent": "docdb-elastic.amazonaws.com",
    "requestParameters": {
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


DescribeKey


```
{
    "eventVersion": "1.08",
```

```
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-05-09T23:04:20Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "docdb-elastic.amazonaws.com"
    },
    "eventTime": "2023-05-09T23:55:48Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "docdb-elastic.amazonaws.com",
    "userAgent": "docdb-elastic.amazonaws.com",
    "requestParameters": {
        "keyId": "alias/SampleKmsKey"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
```

```
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

進一步了解

下列資源提供靜態資料加密的詳細資訊：

- 如需 AWS KMS 概念的詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》中的AWS Key Management Service 基本概念。
- 如需 AWS KMS 安全性的詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》中的的安全最佳實務 AWS Key Management Service。

彈性叢集中的服務連結角色

Amazon DocumentDB 彈性叢集使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是直接連結至 Amazon DocumentDB 彈性叢集的唯一 IAM 角色類型。服務連結角色是由 Amazon DocumentDB 彈性叢集預先定義，並包含該服務代表您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓您更輕鬆地使用 Amazon DocumentDB 彈性叢集，因為您不需要手動新增必要的許可。Amazon DocumentDB 彈性叢集會定義其服務連結角色的許可，除非另有定義，否則只有 Amazon DocumentDB 彈性叢集可以擔任其角色。定義的許可包括信任政策和許可政策，並且該許可政策不能附加到任何其他 IAM 實體。您必須先刪除角色的相關資源，才能刪除角色。這可保護您的 Amazon DocumentDB 彈性叢集資源，因為您不會不小心移除存取資源的許可。

如需有關支援服務連結角色的其他 服務的資訊，請參閱AWS 使用 IAM 的服務，並在服務連結角色欄中尋找標記為是的服務。選擇具有連結的是，以檢視該服務的服務連結角色文件。

彈性叢集的服務連結角色許可

Amazon DocumentDB 彈性叢集使用名為 的服務連結角色`AWS ServiceRoleForDocDB-Elastic`，以允許 Amazon DocumentDB 彈性叢集代表您的叢集呼叫 AWS 服務。

此服務連結角色具有名為 `AmazonDocDB-ElasticServiceRolePolicy` 的許可政策，該政策會授予此角色在帳戶中操作的許可。角色許可政策允許 Amazon DocumentDB 彈性叢集對指定的資源完成下列動作：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": [
                        "AWS/DocDB-Elastic"
                    ]
                }
            }
        }
    ]
}
```

> **Note**
>
> 您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如果您遇到下列錯誤訊息：「無法建立資源。請確認您擁有建立服務連結角色的許可。否則請等待稍後再試一次。」，請確定您已啟用下列許可：

```
{
"Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/docdb-elastic.amazonaws.com/
AWSServiceRoleForDocDB-Elastic",
    "Condition": {
"StringLike": {
"iam:AWSServiceName":"docdb-elastic.amazonaws.com"
```

```
        }
    }
}
```

如需詳細資訊，請參閱 IAM 使用者指南中的服務連結角色許可。

為 Amazon DocumentDB 彈性叢集建立服務連結角色

您不需要手動建立服務連結角色，當您建立資料庫執行個體時，Amazon DocumentDB 彈性叢集會為您建立服務連結角色。

編輯 Amazon DocumentDB 彈性叢集的服務連結角色

Amazon DocumentDB 彈性叢集不允許您編輯AWS ServiceRoleForDocDB-Elastic服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的編輯服務連結角色。

刪除 Amazon DocumentDB 彈性叢集的服務連結角色

如果您不再需要使用需要服務連結角色的功能或服務，請刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。不過您必須先刪除您的所有 叢集，才能刪除服務連結角色。

清除服務連結角色

您必須先確認服務連結角色沒有作用中的工作階段，並移除該角色使用的資源，之後才能使用 IAM 將其刪除。

若要在 IAM 主控台中檢查服務連結角色是否具有作用中工作階段：

1. 登入 AWS 管理主控台並開啟 IAM 主控台。
2. 在 IAM 主控台的導覽窗格中，選擇角色。然後選擇 AWS ServiceRoleForDocDB-Elastic 角色的名稱 (而非核取方塊)。
3. 在所選角色的 Summary (摘要) 頁面中，選擇 Access Advisor (存取 Advisor) 分頁。

> **Note**
>
> 如果您不確定 Amazon DocumentDB 彈性叢集是否正在使用該AWS ServiceRoleForDocDB-Elastic角色，您可以嘗試刪除該角色。如果服務使用 角色，則刪

除會失敗，而且您可以檢視使用該角色 AWS 區域 的 。如果服務正在使用該角色，您必須先等到工作階段結束，才能刪除該角色。您無法撤銷服務連結角色的工作階段。
如果您想要移除AWS ServiceRoleForDocDB-Elastic角色，您必須先刪除所有叢集。

刪除您的所有叢集

若要在 Amazon DocumentDB 主控台中刪除叢集：

1. 登入 AWS 管理主控台並開啟 Amazon DocumentDB 主控台。
2. 在導覽窗格中，選擇叢集。
3. 選擇您要刪除的叢集。
4. 對於 Actions (動作)，請選擇 Delete (刪除)。
5. 如果系統提示您建立最終快照？，請選擇是或否。
6. 如果您在前一個步驟中選擇 Yes (是)，則對於 Final snapshot name (最終快照名稱)，輸入您的最終快照名稱。
7. 選擇 刪除。

> **Note**
>
> 您可以使用 IAM 主控台、IAM CLI 或 IAM API 刪除 AWS ServiceRoleForDocDB-Elastic 服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的刪除服務連結角色。

Amazon DocumentDB 配額

本主題說明 Amazon DocumentDB （具有 MongoDB 相容性） 的資源配額和命名限制。

對於某些管理功能，Amazon DocumentDB 使用與 Amazon Relational Database Service (Amazon RDS) 和 Amazon Neptune 共用的操作技術。

主題

- 支援的執行個體類型
- 支援的區域
- 區域配額
- 彙總配額
- 叢集配額
- 執行個體配額
- 命名限制條件
- TTL 限制條件
- 彈性叢集配額
- 彈性叢集碎片配額
- 每個碎片的彈性叢集 CPU、記憶體、連線和游標配額

支援的執行個體類型

Amazon DocumentDB 支援隨需執行個體和下列執行個體類型：

- NVMe 後端：
  - R6GD 執行個體類型：db.r6gd.xlarge、db.r6gd.2xlarge、db.r6gd.4xlarge、db.r6gd.8xlarge、db.r6gd.
- 記憶體優化:
  - R8G 執行個體類型：db.r8g.large、db.r8g.xlarge、db.r8g.2xlarge、db.r8g.4xlarge、db.r8g.8xlarge
  - R6G 執行個體類型：db.r6g.large、db.r6g.xlarge、db.r6g.2xlarge、db.r6g.4xlarge、db.r6g.8xlarge

  - R5 執行個體類型：db.r5.large、db.r5.xlarge、db.r5.2xlarge、db.r5.4xlarge、db.r5.8xlargedb.r5
  - R4 執行個體類型：db.r4.large、db.r4.2xlarge、db.r4.4xlarge、db.r4.8xlarge、db.r4.16xlarge。
- 爆量效能：
  - T4G 執行個體類型：db.t4g.medium。
  - T3 執行個體類型：db.t3.medium。

如需有關支援的執行個體類型及其規格的詳細資訊，請參閱執行個體類別規格。

支援的區域

Amazon DocumentDB 可在下列內容中使用 AWS 區域：

區域名稱	區域	可用區域（運算）
美國東部 (俄亥俄)	us-east-2	3
美國東部 (維吉尼亞北部)	us-east-1	6
美國西部 (奧勒岡)	us-west-2	4
非洲 (開普敦)	af-south-1	3
南美洲 (聖保羅)	sa-east-1	3
亞太地區 (香港)	ap-east-1	3
亞太地區 (海德拉巴)	ap-south-2	3
亞太地區 (馬來西亞)	ap-southeast-5	3
亞太地區 (孟買)	ap-south-1	3
亞太地區 (大阪)	ap-northeast-3	3
亞太地區 (首爾)	ap-northeast-2	4
亞太地區 (新加坡)	ap-southeast-1	3
亞太地區 (悉尼)	ap-southeast-2	3
亞太地區 (雅加達)	ap-southeast-3	3
亞太地區 (墨爾本)	ap-southeast-4	3
亞太區域 (泰國)	ap-southeast-7	3
亞太地區 (東京)	ap-northeast-1	3

區域名稱	區域	可用區域（運算）
加拿大 (中部)	ca-central-1	3
加拿大西部 (卡加利)	ca-west-1	3
中國 (北京) 區域	cn-north-1	3
中國 (寧夏)	cn-northwest-1	3
歐洲 (法蘭克福)	eu-central-1	3
歐洲 (蘇黎世)	eu-central-2	3
歐洲 (愛爾蘭)	eu-west-1	3
歐洲 (倫敦)	eu-west-2	3
歐洲 (米蘭)	eu-south-1	3
Europe (Paris)	eu-west-3	3
歐洲 (西班牙)	eu-south-2	3
歐洲 (斯德哥爾摩)	eu-north-1	3
墨西哥 (中部)	mx-central-1	3
中東 (阿拉伯聯合大公國)	me-central-1	3
以色列 (特拉維夫)	il-central-1	3
AWS GovCloud（美國西部）	us-gov-west-1	3
AWS GovCloud（美國東部）	us-gov-east-1	3

區域配額

對於某些管理功能，Amazon DocumentDB 使用與 Amazon Relational Database Service (Amazon RDS) 共享的操作技術。下表包含 Amazon DocumentDB 和 Amazon RDS 之間共用的區域配額。

> **Note**
>
> 上述 Amazon RDS 共用技術僅適用於 Amazon DocumentDB 執行個體型叢集。Amazon DocumentDB 彈性叢集不會與 Amazon RDS 共用技術。

下列配額適用於 Amazon DocumentDB 執行個體型叢集，且 AWS 帳戶 依區域而定。

資源	AWS 預設配額
叢集	40
叢集參數群組	50
每個參數群組的叢集	100
每個參數群組的執行個體	300
事件訂閱	20
執行個體	40
手動叢集快照	100
僅供讀取複本數 (每個叢集)	15
子網路群組	50
每個子網路群組的子網路	20
每個資源的標籤	50
每個執行個體的 VPC 安全群組	5

下列配額適用於 Amazon DocumentDB 彈性叢集，每個區域各有 AWS 帳戶 。

資源	AWS 預設配額
彈性叢集	20
彈性叢集 vCPU	1024
手動彈性叢集快照	20

如果配額可調整，您可以使用 Service Quotas 要求增加配額。有些請求會自動解決，有些則提交至 支援。您可以追蹤提交到的配額增加請求的狀態 支援。增加服務配額的請求不會獲得優先支援。如果您有緊急請求，請聯絡 支援。如需服務配額的詳細資訊，請參閱什麼是服務配額？

若要請求提高 Amazon DocumentDB 的配額：

1. 在 https://console.aws.amazon.com/servicequotas 開啟 Service Quotas (服務配額) 主控台，如有需要，請登入。
2. 在導覽窗格中，選擇 AWS services (AWS 服務)。
3. 從清單中選擇 Amazon DocumentDB （與 MongoDB 相容性） 或 Amazon DocumentDB Elastic Cluster，或在搜尋欄位中輸入其中一項。
4. 如果配額可調整，您可以選取其選項按鈕或名稱，然後選擇頁面右上角的 Request quota increase (請求增加配額)。
5. 對於 Change quota value (變更配額值)，輸入新值。新值必須大於目前的值。
6. 選擇請求。解決請求之後，配額的 Applied quota value (套用的配額值) 會設定為新值。
7. 若要檢視任何擱置或最近解決的請求，請從瀏覽窗格中選擇 Dashboard (儀表板)。對於擱置的請求，請選擇請求狀態以開啟請求回條。請求的初始狀態為 `Pending`。狀態變更為 後`Quota requested`，您會看到 的案例編號 支援。選擇案例編號，為請求開啟票證。

彙總配額

下表說明 Amazon DocumentDB 中的彙總配額。

資源	配額
支援之階段數上限	500

叢集配額

下表說明 Amazon DocumentDB 執行個體型叢集配額。

資源	配額
叢集大小 (所有集合和索引的總和)	適用於引擎 8.0 版及更高版本的 256 TiB （適用於較早引擎版本的 128 TiB)
集合大小 （所有集合的總和不可超過叢集配額） – 不包含索引大小	32 TiB
每個叢集的集合	100,000
每個叢集的資料庫數量	100,000
資料庫大小 （所有資料庫的總和不得超過叢集配額）	適用於引擎 8.0 版及更高版本的 256 TiB （適用於較早引擎版本的 128 TiB)
文件巢狀深度	200 個層級
文件大小	16 MiB
索引鍵大小	2048 個位元組
每個集合的索引	64
在複合索引的索引鍵	32

資源	配額
單一批次命令中的寫入次數上限	100,000
每個叢集的使用者數目	1000

執行個體配額

下表說明每個執行個體的 Amazon DocumentDB 配額。

Amazon DocumentDB 8.0

執行個體類型	執行個體記憶體 (GiB)	連線（全部）	游標配額	開啟交易	連線（作用中）
T3.medium	4	1000	30	50	102
T4G.medium	4	1000	30	50	102
R5.large	16	3400	450	200	1100
R5.xlarge	32	7000	450	400	2700
R5.2xlarge	64	14200	450	800	4500
R5.4xlarge	128	28400	760	1600	4500
R5.8xlarge	256	60000	1520	3200	4500
R5.12xlarge	384	60000	2280	4800	4500
R5.16xlarge	512	60000	3040	6400	4500
R6G.large*	16	3400	450	200	1100
R6G.xlarge*	32	7000	450	400	2700
R6G.2xlarge*	64	14200	450	800	4500
R6G.4xlarge*	128	28400	760	1600	4500
R6G.8xlarge*	256	60000	1520	3200	4500

執行個體類型	執行個體記憶體 (GiB)	連線（全部）	游標配額	開啟交易	連線（作用中）
R6G.12xlarge*	384	60000	2280	4800	4500
R6G.16xlarge*	512	60000	3040	6400	4500
R8G.large	16	3400	450	200	1100
R8G.xlarge	32	7000	450	400	2700
R8G.2xlarge	64	14200	450	800	4500
R8G.4xlarge	128	28400	760	1600	4500
R8G.8xlarge	256	60000	1520	3200	4500
R8G.12xlarge	384	60000	2280	4800	4500
R8G.16xlarge	512	60000	3040	6400	4500
R8G.24xlarge	768	60000	4560	9600	4500
R8G.48xlarge	1536	60000	9120	9600	4500

* 包括 R6GD

Amazon DocumentDB 5.0

執行個體類型	執行個體記憶體 (GiB)	連線（全部）	游標配額	開啟交易	連線（作用中）
T3.medium	4	1000	30	50	102

執行個體類型	執行個體記憶體 (GiB)	連線（全部）	游標配額	開啟交易	連線（作用中）
T4G.medium	4	1000	30	50	102
R5.large	16	3400	450	200	1100
R5.xlarge	32	7000	450	400	2700
R5.2xlarge	64	14200	450	800	4500
R5.4xlarge	128	28400	760	1600	4500
R5.8xlarge	256	60000	1520	3200	4500
R5.12xlarge	384	60000	2280	4800	4500
R5.16xlarge	512	60000	3040	6400	4500
R6G.large*	16	3400	450	200	1100
R6G.xlarge*	32	7000	450	400	2700
R6G.2xlarge*	64	14200	450	800	4500
R6G.4xlarge*	128	28400	760	1600	4500
R6G.8xlarge*	256	60000	1520	3200	4500
R6G.12xlarge*	384	60000	2280	4800	4500
R6G.16xlarge*	512	60000	3040	6400	4500
R8G.large	16	3400	450	200	1100
R8G.xlarge	32	7000	450	400	2700

執行個體類型	執行個體記憶體 (GiB)	連線（全部）	游標配額	開啟交易	連線（作用中）
R8G.2xlarge	64	14200	450	800	4500
R8G.4xlarge	128	28400	760	1600	4500
R8G.8xlarge	256	60000	1520	3200	4500
R8G.12xlarge	384	60000	2280	4800	4500
R8G.16xlarge	512	60000	3040	6400	4500
R8G.24xlarge	768	60000	4560	9600	4500
R8G.48xlarge	1536	60000	9120	9600	4500

* 包括 R6GD

Amazon DocumentDB 4.0

執行個體類型	執行個體記憶體 (GiB)	連線（全部）	游標配額	開啟交易	連線（作用中）
T3.medium	4	1000	30	50	102
T4G.medium	4	1000	30	50	102
R5.large	16	3400	450	200	1100
R5.xlarge	32	7000	450	400	2700
R5.2xlarge	64	14200	450	800	4500
R5.4xlarge	128	28400	760	1600	4500

執行個體類型	執行個體記憶體 (GiB)	連線（全部）	游標配額	開啟交易	連線（作用中）
R5.8xlarge	256	60000	1520	3200	4500
R5.12xlarge	384	60000	2280	4800	4500
R5.16xlarge	512	60000	3040	6400	4500
R6G.large*	16	3400	450	200	1100
R6G.xlarge*	32	7000	450	400	2700
R6G.2xlarge*	64	14200	450	800	4500
R6G.4xlarge*	128	28400	760	1600	4500
R6G.8xlarge*	256	60000	1520	3200	4500
R6G.12xlarge*	384	60000	2280	4800	4500
R6G.16xlarge*	512	60000	3040	6400	4500

* 包括 R6GD

Amazon DocumentDB 3.6

執行個體類型	執行個體記憶體 (GiB)	連線（全部）	游標配額	開啟交易	連線（作用中）
T3.medium	4	500	30	N/A	102
R4.large	15.25	1700	450	N/A	1100
R4.xlarge	30.5	3400	450	N/A	2700

執行個體類型	執行個體記憶體 (GiB)	連線（全部）	游標配額	開啟交易	連線（作用中）
R4.2xlarge	61	6800	450	N/A	4500
R4.4xlarge	122	13600	725	N/A	4500
R4.8xlarge	288	27200	1450	N/A	4500
R4.16xlarge	488	30000	2900	N/A	4500
R5.large	16	1700	450	N/A	1100
R5.xlarge	32	3500	450	N/A	2700
R5.2xlarge	64	7100	450	N/A	4500
R5.4xlarge	128	14200	760	N/A	4500
R5.8xlarge	256	28400	1520	N/A	4500
R5.12xlarge	384	30000	2280	N/A	4500
R5.16xlarge	512	30000	3040	N/A	4500

您可以使用下列 CloudWatch 指標，監控和警示每個執行個體限制的 。如需 Amazon DocumentDB CloudWatch 指標的詳細資訊，請參閱使用 CloudWatch 監控 Amazon DocumentDB。

資源	CloudWatch 配額指標	CloudWatch 用量指標（最多 1 分鐘）	CloudWatch 用量指標
執行個體記憶體	-	-	FreeableMemory
連線（全部）	DatabaseConnectionsLimit	DatabaseConnectionsMax	DatabaseConnections
游標	DatabaseCursorsLimit	DatabaseCursorsMax	DatabaseCursors
交易	TransactionsOpenLimit	TransactionsOpenMax	TransactionsOpen

命名限制條件

下表說明 Amazon DocumentDB 中的命名限制。

資源	預設配額
叢集識別碼	• 長度為 【1–63】 個字母、數字或連字號。 • 第一個字元必須是字母。 • 不能以連字號結尾，或包含兩個連續連字號。 • 對於每個區域每個 AWS 帳戶的所有叢集 （跨 Amazon RDS、Amazon Neptune 和 Amazon DocumentDB)，必須是唯一的。
執行個體識別碼	• 長度為 【1–63】 個字母、數字或連字號 • 第一個字元必須是字母 • 不能以連字號結尾或連續包含兩個連字號 • 對於每個區域每個 AWS 帳戶的所有執行個體 （跨 Amazon RDS、Amazon Neptune 和 Amazon DocumentDB)，必須是唯一的。
集合名稱：<col>	長度為 【1–255】 個字元 (Amazon DocumentDB 5.0 及更新版本）。 長度為 【1–57】 個字元 (Amazon DocumentDB 4.0 及更早版本）。
資料庫名稱：<db>	長度為 【1–63】 個字元。

資源	預設配額
完整集合名稱： <db>.<col>	長度為 【3–255】 個字元 (Amazon DocumentDB 5.0 及更新版本）。 長度為 【3–120】 個字元 (Amazon DocumentDB 4.0 及更早版本）。
完整索引名稱： <db>.<col>$<index>	長度為 【6–511】 個字元 (Amazon DocumentDB 5.0 及更新版本）。 長度為 【6–127】 個字元 (Amazon DocumentDB 4.0 及更早版本）。
索引名稱	長度為 【1–255】 個字元 (Amazon DocumentDB 5.0 及更新版本）。 合併 <col>$<index> (Amazon DocumentDB 4.0 及更早版本） 的長度為 【3–63】 個字元。

主要密碼	• 長度為 [8-100] 個可列印的 ASCII 字元。
	• 可以使用下列項目以外的任何可列印 ASCII 字元：
	  • **/** (正斜線)
	  • **"** (雙引號)
	  • **@** (@ 符號)

主要使用者名稱	• 長度為 [1-63] 英數字元。 • 第一個字元必須是字母。 • 不能是資料庫引擎保留的字。
參數群組名稱	• 長度為 [1-255] 個英數字元。 • 第一個字元必須是字母。 • 不能以連字號結尾，或包含兩個連續連字號。

TTL 限制條件

在特定時間範圍內無法保證可從 TTL 索引中刪除，而且這是最佳作法。像是執行個體資源使用率、文件大小和整體輸送量等因素，都會影響 TTL 刪除的時間。

彈性叢集配額

下表說明 Amazon DocumentDB 彈性叢集的最大配額。

資源	配額
每個區域的彈性叢集	20
每個區域所有彈性叢集的 vCPU 加總	1024
每個區域的手動叢集快照	20
每個叢集的碎片	32
每個叢集的儲存體（當資料由碎片金鑰平均分佈時）	4 PiB
叢集的連線	每個碎片 300，000 個或碎片數量的較低值 x 與 vCPU 相關聯的連線配額
UnSharded集合大小	32 TiB
碎片收集大小（當資料由碎片索引鍵平均分佈時）	1PB
每個叢集的資料庫數量	10,000
每個叢集UnSharded集合	100,000
每個叢集的碎片集合	1000
每個叢集的使用者	100
在單一批次命令中寫入	100,000

資源	配額
每個集合的索引	64
文件巢狀深度	100 個層級
文件大小	16MB
索引鍵大小	2048 位元組
在複合索引的索引鍵	32

彈性叢集碎片配額

下表說明 Amazon DocumentDB 彈性叢集的最大碎片配額。

資源	配額
每個碎片執行個體的 vCPU	64
每個碎片的執行個體數	16
每個碎片的儲存體	128 TiB
每個碎片每個集合的儲存體	32 TiB

每個碎片的彈性叢集 CPU、記憶體、連線和游標配額

下表說明 Amazon DocumentDB 彈性叢集碎片中的最大 CPU、記憶體、連線和游標限制。

每個碎片vCPUs	執行個體記憶體 (GiB)	連線配額	游標配額
2	16	1700	450
4	32	3500	450
8	64	7100	450

每個碎片vCPUs	執行個體記憶體 (GiB)	連線配額	游標配額
16	128	14200	760
32	256	28400	1520
48	384	30000	2280
64	512	30000	3040

Amazon DocumentDB 故障診斷

下列各節提供有關如何對使用 Amazon DocumentDB 時可能遇到的問題進行故障診斷的資訊（具有 MongoDB 相容性）。

主題

- 故障診斷連線問題
- 索引故障診斷
- 對效能和資源使用率進行故障診斷
- Amazon DocumentDB 中的垃圾回收
- 錯誤代碼

故障診斷連線問題

連線時發生問題？ 以下是一些常見的案例，以及如何解決這些案例。

主題

- 無法連線至 Amazon DocumentDB 端點
- 測試與 Amazon DocumentDB 執行個體的連線
- 連線至無效的端點
- 影響連線數量的驅動程式組態

無法連線至 Amazon DocumentDB 端點

當您嘗試連線至 Amazon DocumentDB 時，下列是您可能會收到的最常見錯誤訊息之一。

```
connecting to: mongodb://docdb-2018-11-08-21-47-27.cluster-ccuszbx3pn5e.us-east-
1.docdb.amazonaws.com:27017/
2018-11-14T14:33:46.451-0800 W NETWORK [thread1] Failed to connect to
172.31.91.193:27017 after 5000ms milliseconds, giving up.
2018-11-14T14:33:46.452-0800 E QUERY [thread1] Error: couldn't connect to server
docdb-2018-11-08-21-47-27.cluster-ccuszbx3pn5e.us-east-1.docdb.amazonaws.com:27017,
connection attempt failed :
connect@src/mongo/shell/mongo.js:237:13
```

```
@(connect):1:6
exception: connect failed
```

此錯誤訊息通常表示您的用戶端 （此範例中的 mongo shell) 無法存取 Amazon DocumentDB 端點。這可能有幾個原因：

主題

- 從公有端點連線
- 跨區域連線
- 從不同的 Amazon VPCs 連線
- 安全群組封鎖傳入連線
- Java Mongo 驅動程式讀取偏好設定問題

從公有端點連線

您正在嘗試直接從筆記型電腦或本機開發機器連線至 Amazon DocumentDB 叢集。

嘗試直接從您的筆記型電腦或本機開發機器等公有端點連線至 Amazon DocumentDB 叢集將會失敗。Amazon DocumentDB 是僅限虛擬私有雲端 (VPC)，目前不支援公有端點。因此，您無法從筆記型電腦或 VPC 外部的本機開發環境直接連線至 Amazon DocumentDB 叢集。

若要從 Amazon VPC 外部連線至 Amazon DocumentDB 叢集，您可以使用 SSH 通道。如需詳細資訊，請參閱從 Amazon VPC 外部連線至 Amazon DocumentDB 叢集。此外，如果您的開發環境位於不同的 Amazon VPC 中，您也可以使用 VPC 對等互連，並從相同區域或不同區域中的其他 Amazon VPC 連線至您的 Amazon DocumentDB 叢集。

跨區域連線

您嘗試連線到另一個區域中的 Amazon DocumentDB 叢集。

如果您嘗試從叢集區域以外的 區域中的 Amazon EC2 執行個體連線至 Amazon DocumentDB 叢集，例如，嘗試從美國西部 （奧勒岡） 區域 (us-west-2) 連線至美國東部 （維吉尼亞北部） 區域 (us-east-1) 中的叢集，連線將會失敗。

若要驗證 Amazon DocumentDB 叢集的區域，請執行下列命令。區域在端點中。

```
aws docdb describe-db-clusters \
   --db-cluster-identifier sample-cluster \
```

```
   --query 'DBClusters[*].Endpoint'
```

此操作的輸出將會如下所示。

```
[
    "sample-cluster.node.us-east-1.docdb.amazonaws.com"
]
```

為了驗證您的 EC2 執行個體的區域，請執行以下命令。

```
 aws ec2 describe-instances \
     --query 'Reservations[*].Instances[*].Placement.AvailabilityZone'
```

此操作的輸出將會如下所示。

```
[
    [
        "us-east-1a"
    ]
]
```

從不同的 Amazon VPCs 連線

您嘗試從與您部署叢集的 Amazon VPC 不同的 VPC 連線至 Amazon DocumentDB 叢集。

如果您的 Amazon DocumentDB 叢集和 Amazon EC2 執行個體都位於相同的 Amazon VPC 中 AWS 區域，但不在相同的 Amazon VPC 中，除非在兩個 Amazon VPCs 之間啟用 VPC 對等互連，否則您無法直接連線至 Amazon DocumentDB 叢集。

若要驗證 Amazon DocumentDB 執行個體的 Amazon VPC，請執行下列命令。

```
aws docdb describe-db-instances \
   --db-instance-identifier sample-instance \
   --query 'DBInstances[*].DBSubnetGroup.VpcId'
```

若要驗證 Amazon EC2 執行個體的 Amazon VPC，請執行下列命令。

```
aws ec2 describe-instances \
   --query 'Reservations[*].Instances[*].VpcId'
```

安全群組封鎖傳入連線

您嘗試連線到 Amazon DocumentDB 叢集，而叢集的安全群組不允許叢集連接埠上的傳入連線 （ 預設連接埠：27017)。

假設您的 Amazon DocumentDB 叢集和 Amazon EC2 執行個體都位於相同的區域和 Amazon VPC，並使用相同的 Amazon VPC 安全群組。如果您無法連線至 Amazon DocumentDB 叢集，可能原因為叢集的安全群組 （ 也就是防火牆 ） 不允許您為 Amazon DocumentDB 叢集選擇的連接埠 （ 預設連接埠為 27017) 上的傳入連線。

若要驗證 Amazon DocumentDB 叢集的連接埠，請執行下列命令。

```
aws docdb describe-db-clusters \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[DBClusterIdentifier,Port]'
```

若要取得叢集的 Amazon DocumentDB 安全群組，請執行下列命令。

```
aws docdb describe-db-clusters \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[VpcSecurityGroups[*],VpcSecurityGroupId]'
```

若要檢查安全群組的傳入規則，請參閱 Amazon EC2 文件中的下列主題：

- 授權 Linux 執行個體的傳入流量
- 授權 Windows 執行個體的傳入流量

Java Mongo 驅動程式讀取偏好設定問題

除非重新啟動，否則不會遵守用戶端讀取偏好設定，而且某些用戶端無法在容錯移轉後寫入 Amazon DocumentDB。

此問題在 Java Mongo Driver 3.7.x 中首次發現，發生在用戶端使用 建立與 Amazon DocumentDB 的連線`MongoClientSettings`，特別是鏈結`applyToClusterSettings`方法時。MongoClient 叢集設定可以使用幾種不同的方法來定義，例如 `hosts()`、 `requiredReplicaSetName()`和 `mode()`。

當用戶端在 `hosts()`方法中只指定一個主機時，模式會設定為 `ClusterConnectionMode.SINGLE`而非 `ClusterConnectionMode.MULTIPLE` 這會讓用戶端忽

略讀取偏好設定，並只連線到 中設定的伺服器hosts()。因此，即使用戶端設定初始化如下，所有讀取仍會移至主要 而非次要 。

```
final ServerAddress serverAddress0 = new ServerAddress("cluster-endpoint", 27317));
    final MongoCredential credential = MongoCredential.createCredential("xxx",
            "admin", "xxxx".toCharArray());
    final MongoClientSettings settings = MongoClientSettings.builder()
            .credential(credential)
            .readPreference(ReadPreference.secondaryPreferred())
            .retryWrites(false)
            .applyToSslSettings(builder -> builder
                    .enabled(false))
            .applyToClusterSettings(builder -> builder.hosts(
                            Arrays.asList(serverAddress0
                            ))
                    .requiredReplicaSetName("rs0"))
            .build();
    MongoClient mongoClient = MongoClients.create(settings);
```

容錯移轉案例

使用上述用戶端連線設定，如果叢集寫入器端點有容錯移轉和延遲 DNS 記錄更新，用戶端仍會嘗試向舊寫入器發出寫入（現在容錯移轉後的讀取器）。這會導致伺服器端錯誤（非主要），Java 驅動程式未適當處理（這仍在調查中）。因此，用戶端可能會處於錯誤狀態，直到應用程式伺服器重新啟動為止。

有兩種解決方法：

- 透過連線字串連線至 Amazon DocumentDB 的用戶端不會發生此問題，因為ClusterConnectionMode會在設定讀取偏好設定MULTIPLE時設定為 。

  ```
  MongoClientURI mongoClientURI = new MongoClientURI("mongodb://usr:pass:cluster-
  endpoint:27317/test?ssl=false&replicaSet=rs0&readpreference=secondaryPreferred");
  MongoClient mongoClient = MongoClients.create(mongoClientURI.getURI());
  ```

  或搭配 applyConnectionString方法使用MongoClientSettings建置器。

  ```
  final MongoClientSettings settings = MongoClientSettings.builder()
          .credential(credential)
          .applyConnectionString(new ConnectionString("usr:pass:cluster-endpoint:27317/
  test?ssl=false&replicaSet=rs0&readpreference=secondaryPreferred"))
  ```

```
        .retryWrites(false)
        .applyToSslSettings(builder # builder
                .enabled(false))
        .build();
MongoClient mongoClient = MongoClients.create(settings);
```

- 明確ClusterConnectionMode設定為 MULTIPLE。這只有在使用 applyToClusterSettings和 時才需要hosts().size() == 1。

```
final ServerAddress serverAddress0 = new ServerAddress("cluster-endpoint", 27317));
final MongoCredential credential = MongoCredential.createCredential("xxx","admin",
 "xxxx".toCharArray());
final MongoClientSettings settings = MongoClientSettings.builder()
    .credential(credential)
    .readPreference(ReadPreference.secondaryPreferred())
    .retryWrites(false)
    .applyToSslSettings(builder # builder
    .enabled(false))
    .applyToClusterSettings(builder # builder
                .hosts(Arrays.asList(serverAddress0))
                .requiredReplicaSetName("rs0"))
                .mode(ClusterConnectionMode.MULTIPLE))
    .build();
MongoClient mongoClient = MongoClients.create(settings);
```

測試與 Amazon DocumentDB 執行個體的連線

您可以使用通用的 Linux 或 Windows 工具來測試您的叢集連線。

從 Linux 或 Unix 終端機測試連線，方法是輸入下列命令 (以端點取代 `cluster-endpoint`，以及以執行個體的連接埠取代 `port`)：

```
nc -zv cluster-endpoint port
```

以下是範例操作和傳回值的範例：

```
nc -zv docdbTest.d4c7nm7stsfc0.us-west-2.docdb.amazonaws.com 27017

Connection to docdbTest.d4c7nm7stsfc0.us-west-2.docdb.amazonaws.com 27017 port [tcp/*]
 succeeded!
```

連線至無效的端點

連線至 Amazon DocumentDB 叢集並使用無效的叢集端點時，會出現類似以下的錯誤。

```
mongo --ssl \
   --host sample-cluster.node.us-east-1.docdb.amazonaws.com:27017 \
   --sslCAFile global-bundle.pem \
   --username <user-name> \
   --password <password>
```

輸出如下：

```
MongoDB shell version v3.6
connecting to: mongodb://sample-cluster.node.us-east-1.docdb.amazonaws.com:27017/
2018-11-14T17:21:18.516-0800 I NETWORK [thread1] getaddrinfo("sample-cluster.node.us-
east-1.docdb.amazonaws.com") failed:
nodename nor servname provided, or not known 2018-11-14T17:21:18.537-0800 E QUERY
 [thread1] Error: couldn't initialize
connection to host sample-cluster.node.us-east-1.docdb.amazonaws.com, address is
 invalid :
connect@src/mongo/shell/mongo.js:237:13@(connect):1:6
exception: connect failed
```

若要取得有效的叢集端點，請執行下列命令：

```
aws docdb describe-db-clusters \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[Endpoint,Port]'
```

若要取得有效的執行個體端點，請執行下列命令：

```
aws docdb describe-db-instances \
   --db-instance-identifier sample-instance \
   --query 'DBInstances[*].[Endpoint.Address,Endpoint.Port]'
```

如需詳細資訊，請參閱了解 Amazon DocumentDB 端點。

影響連線數量的驅動程式組態

使用用戶端驅動程式連線至 Amazon DocumentDB 叢集時，請務必考慮`maxPoolSize`組態參數。此`maxPoolSize`設定決定用戶端驅動程式在其連線集區中將維持的最大連線數。

索引故障診斷

下列主題討論，如果您的索引或背景索引建置失敗，怎麼辦。

主題

- 索引建置失敗
- 背景索引建置延遲問題和失敗
- 資料庫索引膨脹

索引建置失敗

Amazon DocumentDB 會使用執行個體上的本機儲存體做為索引建立程序的一部分。您可以使用 FreeLocalStorage CloudWatch 指標 () 監控此磁碟用量`CloudWatch -> Metrics -> DocDB -> Instance Metrics`。當索引建置使用所有本機磁碟並失敗時，您將會收到錯誤。將資料遷移至 Amazon DocumentDB 時，建議您先建立索引，然後插入資料。如需遷移策略和建立索引的詳細資訊，請參閱遷移和升級 Amazon DocumentDB《Amazon DocumentDB 文件》中的 和部落格：使用離線方法從 MongoDB 遷移至 Amazon DocumentDB。

在現有叢集上建立索引時，如果索引建置所花費的時間超過預期或失敗，請擴展執行個體以建立索引，然後在建立索引後縮減。Amazon DocumentDB 可讓您使用 AWS 管理主控台 或 ，在幾分鐘內快速擴展執行個體大小 AWS CLI。如需詳細資訊，請參閱管理執行個體類別。透過每秒執行個體定價，您只需支付使用的資源，以秒計費。

背景索引建置延遲問題和失敗

Amazon DocumentDB 中的背景索引建置不會啟動，除非在索引建置啟動之前啟動的主要執行個體上的所有查詢都已完成執行。如果有長時間執行的查詢，背景索引建置會封鎖，直到查詢完成，因此可能需要比預期更長的時間才能完成。即使集合是空的，也是如此。

前景索引建置不會顯示相同的封鎖行為。相反地，前景索引組建會對集合進行獨佔鎖定，直到索引組建完成為止。若要在空集合上建立索引，並避免封鎖長時間執行的查詢，請使用前景索引組建。

> **Note**
>
> Amazon DocumentDB 僅允許在任何指定時間在集合上建立一個背景索引組建。如果在背景索引建置期間 DDL (資料定義語言) 操作 (如 `createIndex()` 或 `dropIndex()`) 發生於同一個集合，則背景索引建置會失敗。

資料庫索引膨脹

Amazon DocumentDB 使用多版本並行控制 (MVCC) 來管理並行交易。刪除或更新文件時，其先前的版本會保留在集合和索引中為「無效」版本。垃圾收集程序會自動從這些無效版本回收空間，以供未來操作使用。

當集合的索引因為頁面中累積無效或過時的索引項目或分段而變大時，就會發生索引膨脹。報告的百分比代表未來索引項目可以使用的索引空間量。此膨脹會耗用緩衝區快取和儲存體中的空間。如果您想要移除膨脹，則需要重建索引。

Example範例

執行下列命令來判斷索引的未使用儲存：

```
db.coll.aggregate({$indexStats:{}});
```

結果如下所示：

```
{
    "name" : "_id_",
    "key" : {
        "_id" : 1
    },
    "host" : "devbox-test.localhost.a2z.com:27317",
    "size" : NumberLong(827392),
    "accesses" : {
        "ops" : NumberLong(40000),
        "docsRead" : NumberLong(46049),
        "since" : ISODate("2025-04-03T21:44:51.251Z")
    },
    "cacheStats" : {
        "blksRead" : NumberLong(264),
        "blksHit" : NumberLong(140190),
        "hitRatio" : 99.8121
    },
    "unusedStorageSize" : {
        "unusedSizeBytes" : 409600,
        "unusedSizePercent" : 49.51
    }
}
```

您可以使用 reIndex命令重建索引，而不需要停機，這需要掃描整個集合。請參閱使用 的索引維護 reIndex。

對效能和資源使用率進行故障診斷

本節提供 Amazon DocumentDB 部署中常見診斷問題的問題和解決方案。這些範例會提供使用 mongo 殼層，而且範圍限定在個別執行個體的示範。若要尋找執行個體端點，請參閱了解 Amazon DocumentDB 端點。

主題

- 如何判斷透過 Mongo API 在集合上執行的插入、更新和刪除操作數目？
- 如何分析快取效能？
- 如何尋找並終止長時間執行或封鎖的查詢？
- 如何查看查詢計畫並最佳化查詢？
- 如何查看彈性叢集中的查詢計劃？
- 如何列出執行個體上所有正在執行的操作？
- 如何知道查詢何時進行？
- 如何判斷系統突然緩慢執行的原因？
- 如何判斷一或多個叢集執行個體上高 CPU 使用率的原因？
- 如何判斷執行個體上的開啟游標？
- 如何判斷目前的 Amazon DocumentDB 引擎版本？
- 如何分析索引用量並識別未使用的索引？
- 如何識別缺少的索引？
- 如何判斷資料庫集合膨脹？
- 有用的查詢摘要

如何判斷透過 Mongo API 在集合上執行的插入、更新和刪除操作數目？

若要檢視在特定集合上執行的插入、更新和刪除操作數目，請在該集合上執行下列命令：

```
db.collection.stats()
```

此命令的輸出在其 opCounters 欄位下描述下列項目：

- numDocsIns - 插入此集合的文件數量。這包括使用 insert和 insertMany命令插入的文件，以及 upsert 插入的文件。
- numDocsUpd - 此集合中的文件更新數量。這包括使用 update和 findAndModify命令更新的文件。
- numDocsDel - 從此集合中刪除的文件數量。這包括使用 deleteOne、remove、 deleteMany和 findAndModify命令刪除的文件。
- lastReset - 這些計數器上次重設的時間。此命令提供的統計資料會在啟動/停止叢集或向上/向下擴展執行個體時重設。

執行中 的範例輸出db.collection.stats()如下所示。

```
{
    "ns" : "db.test",
    "count" : ...,
    "size" : ...,
    "avgObjSize" : ...,
    "storageSize" : ...,
    "capped" : false,
    "nindexes" : ...,
    "totalIndexSize" : ...,
    "indexSizes" : {
        "_id_" : ...,
        "x_1" : ...
    },
    "collScans" : ...,
    "idxScans" : ...,
    "opCounter" : {
        "numDocsIns" : ...,
        "numDocsUpd" : ...,
        "numDocsDel" : ...
    },
    "cacheStats" : {
        "collBlksHit" : ...,
        "collBlksRead" : ..,
        "collHitRatio" : ...,
        "idxBlksHit" : ...,
        "idxBlksRead" : ...,
        "idxHitRatio" : ...
    },
    "lastReset" : "2022-09-02 19:41:40.471473+00",
    "ok" : 1,
```

```
    "operationTime" : Timestamp(1662159707, 1)
}
```

透過 Mongo API 檢視插入、更新和刪除操作的集合特定計數器時，應使用此 stats 命令。檢視集合特定操作計數器的另一種方法是啟用 DML 稽核。您可以在 中檢視一分鐘時間間隔內所有集合的插入、更新和刪除操作數目使用 CloudWatch 監控 Amazon DocumentDB。

如何分析快取效能？

分析快取效能可以深入了解資料擷取和系統效能的效率，並根據從磁碟讀取的資料量與快取的資料量而定。我們提供有關快取命中次數（從快取讀取的資料）和快取未命中次數（在快取中找不到並從磁碟讀取的資料）的快取統計資料，以便深入了解快取效能。透過在該集合上執行下列命令，即可找到特定集合的快取統計資料：

```
db.collection.stats()
```

此命令輸出中的 cacheStats 欄位值提供集合的快取統計資料，以及在集合上建立之索引的快取統計資料總計。以下列出這些統計資料：

- **collBlksHit** - 在此集合操作期間從快取讀取的區塊數量。
- **collBlksRead** - 在此集合操作期間從磁碟讀取的區塊數量（快取遺漏）。
- **collHitRatio** - 此集合的快取命中率 (100 * [collBlksHit / (collBlksHit + collBlksRead)])。
- **idxBlksHit** - 為此集合上建立的任何索引從快取讀取的區塊數量。
- **idxBlksRead** - 此集合上建立的任何索引從磁碟讀取的區塊數量（快取遺漏）。
- **idxHitRatio** - 在此集合上建立之索引的快取命中率 (100 * [idxBlksHit / (idxBlksHit + idxBlksRead)])。
- **lastReset** - 上次重設這些統計資料的時間。提供的統計資料db.collection.stats()會在啟動/停止叢集或向上/向下擴展執行個體時重設。

您也可以使用 indexStats命令找到每個索引的 idxBlksHit和 idxBlksRead 欄位明細。執行下列命令，即可找到索引特定的快取統計資料：

```
db.collection.aggregate([{$indexStats:{}}]).pretty()
```

對於每個索引，您可以在 cacheStats 欄位下找到下列快取統計資料：

- **blksHit** - 此索引從快取讀取的區塊數量。
- **blksRead** - 從此索引磁碟讀取的區塊數量。
- **blksHitRatio** - 快取命中率四捨五入到四個小數位數，由 計算`100 * [blksHit / (blksHit + blksRead)]`。

如何尋找並終止長時間執行或封鎖的查詢？

使用者查詢執行緩慢的原因，可能是查詢計畫不佳，或因為資源爭用而遭封鎖。

若要尋找因查詢計畫不佳而造成長時間執行查詢的效率變慢，或因為資源爭用而遭封鎖的查詢，請先使用 currentOp 命令。您可以篩選命令，以利減縮要終止的相關查詢清單。您必須具備與長時間執行查詢相關聯的 opid，才能夠終止查詢。

以下查詢會使用 currentOp 命令來列出所有遭到封鎖或執行超過 10 秒的查詢。

```
db.adminCommand({
    aggregate: 1,
    pipeline: [
        {$currentOp: {}},
        {$match:
            {$or: [
                {secs_running: {$gt: 10}},
                {WaitState: {$exists: true}}]}},
        {$project: {_id:0, opid: 1, secs_running: 1}}],
    cursor: {}
});
```

接著，您就可以縮減查詢，尋找並終止執行超過 10 秒的查詢 opid。

尋找和終止執行超過 10 秒的查詢

1. 尋找查詢的 opid。

   ```
   db.adminCommand({
       aggregate: 1,
       pipeline: [
           {$currentOp: {}},
           {$match:
               {$or:
                   [{secs_running: {$gt: 10}},
   ```

```
                {WaitState: {$exists: true}}]}}],
    cursor: {}
});
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "waitedMS" : NumberLong(0),
    "cursor" : {
        "firstBatch" : [
            {
                "opid" : 24646,
                "secs_running" : 12
            }
        ],
        "id" : NumberLong(0),
        "ns" : "admin.$cmd"
    },
    "ok" : 1
}
```

2. 使用 killOp 操作終止查詢。

```
db.adminCommand({killOp: 1, op: 24646});
```

如何查看查詢計畫並最佳化查詢？

查詢的執行速度如果緩慢，可能是由於查詢執行需要完整掃描集合來選擇相關的文件。有時，建立適當的索引能夠使查詢的執行速度更快。若要偵測這種情況並決定建立索引的欄位，請使用 explain 命令。

> **Note**
>
> Amazon DocumentDB 在利用分散式、容錯、自我修復儲存系統的專用資料庫引擎上模擬 MongoDB 3.6 API。因此，Amazon DocumentDB 和 MongoDB 之間的查詢計劃和 的輸出explain()可能不同。想要控制其查詢計劃的客戶可以使用 $hint 運算子強制選取偏好的索引。

請在 explain 命令下執行您要改善的查詢，如下所示。

```
db.runCommand({explain: {<query document>}})
```

下面是範例操作。

```
db.runCommand({explain:{
    aggregate: "sample-document",
    pipeline: [{$match: {x: {$eq: 1}}}],
    cursor: {batchSize: 1}}
});
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "db.test",
        "winningPlan" : {
            "stage" : "COLLSCAN"
        }
    },
    "serverInfo" : {
        "host" : "...",
        "port" : ...,
        "version" : "..."
    },
    "ok" : 1
}
```

上述輸出表示 $match 階段需要掃描完整的集合，並確認每個文件中的欄位 "x" 皆等於 1。如果集合中有多個文件，則集合掃描速度 (和必然的整體查詢效能) 將非常緩慢。因此，explain 命令輸出中的 "COLLSCAN" 表示可以透過建立適當的索引來提高查詢效能。

在此範例中，查詢會檢查所有文件中的欄位 "x" 是否等於 1。因此建立欄位 "x" 的索引，便可讓查詢避免完整集合掃描，並且使用索引加快相關文件傳回。

在欄位 "x" 上建立索引後，explain 輸出如下所示。

```
{
    "queryPlanner" : {
         "plannerVersion" : 1,
         "namespace" : "db.test",
         "winningPlan" : {
```

```
            "stage" : "IXSCAN",
            "indexName" : "x_1",
            "direction" : "forward"
         }
    },
    "serverInfo" : {
        "host" : "...",
        "port" : ...,
        "version" : "..."
    },
    "ok" : 1
}
```

在欄位 "x" 上建立索引將使 $match 階段能夠使用索引掃描，以減少必須進行評估之述詞 "x = 1" 的文件數量。

對於小型集合，如果效能增益微乎其微，Amazon DocumentDB 查詢處理器可以選擇不使用索引。

如何查看彈性叢集中的查詢計劃？

若要檢查彈性叢集中的查詢計劃，請使用 explain命令。以下是以碎片集合為目標之尋找查詢的範例explain操作：

```
db.runCommand(
   {
     explain: { find: "cities", filter: {"name": "Seoul"}}
   }
)
```

> **Note**
>
> Amazon DocumentDB 在專用資料庫引擎上模擬 MongoDB。因此，Amazon DocumentDB 和 MongoDB 之間的查詢計劃和 的輸出explain()可能不同。您可以使用 $hint運算子來控制查詢計劃，以強制選取偏好的索引。

此操作的輸出看起來可能會類似以下 (JSON 格式）：

```
{
  "queryPlanner" : {
    "elasticPlannerVersion" : 1,
```

```
    "winningPlan" : {
      "stage" : "SINGLE_SHARD",
      "shards" : [
        {
          "plannerVersion" : 1,
          "namespace" : "population.cities",
          "winningPlan" : {
            "stage" : "SHARD_MERGE",
            "shards" : [
              {
                "shardName" : "f2cf5cfd-fe9c-40ca-b4e5-298ca0d11111",
                "plannerVersion" : 1,
                "namespace" : "population.cities",
                "winningPlan" : {
                  "stage" : "PARTITION_MERGE",
                  "inputStages" : [
                    {
                      "stage" : "COLLSCAN",
                      "partitionCount" : 21
                    }
                  ]
                }
              },
              {
                "shardName" : "8f3f80e2-f96c-446e-8e9d-aab8c7f22222",
                "plannerVersion" : 1,
                "namespace" : "population.cities",
                "winningPlan" : {
                  "stage" : "PARTITION_MERGE",
                  "inputStages" : [
                    {
                      "stage" : "COLLSCAN",
                      "partitionCount" : 21
                    }
                  ]
                }
              },
              {
                "shardName" : "32c5a06f-1b2b-4af1-8849-d7c4a033333",
                "plannerVersion" : 1,
                "namespace" : "population.cities",
                "winningPlan" : {
                  "stage" : "PARTITION_MERGE",
                  "inputStages" : [
```

```
                    {
                      "stage" : "COLLSCAN",
                      "partitionCount" : 22
                    }
                  ]
                }
              }
            ]
          },
          "shardName" : "32c5a06f-1b2b-4af1-8849-d7c4a0f3fb58"
        }
      ]
    }
  },
  "serverInfo" : {
    "host" : "example-4788267630.us-east-1.docdb-elastic.amazonaws.com:27017",
    "version" : "5.0.0"
  },
  "ok" : 1,
  "operationTime" : Timestamp(1695097923, 1)
}
```

上述輸出顯示三個碎片叢集上查詢的find查詢計劃。每個碎片都有多個資料分割區，可以有不同的輸入階段。在此範例中，「COLLSCAN」（集合掃描）會先在所有分割區上執行，再在每個碎片中的「PARTITION_MERGE」階段合併結果。然後將碎片的結果合併到「SHARD_MERGE」階段，再傳回用戶端。

如何列出執行個體上所有正在執行的操作？

身為使用者或主要使用者，您通常想要列出執行個體上執行的所有目前操作，以進行診斷和故障診斷。(如需有關管理使用者的詳細資訊，請參閱 管理 Amazon DocumentDB 使用者。)

使用 mongo shell，您可以使用下列查詢列出 Amazon DocumentDB 執行個體上執行的所有操作。

```
db.adminCommand({currentOp: 1, $all: 1});
```

此查詢會傳回一份完整清單，內含所有使用者查詢和執行個體目前正在操作的內部系統任務。

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "inprog" : [
```

```
        {
            "desc" : "INTERNAL"
        },
        {
            "desc" : "TTLMonitor",
            "active" : false
        },
        {
            "client" : ...,
            "desc" : "Conn",
            "active" : true,
            "killPending" : false,
            "opid" : 195,
            "ns" : "admin.$cmd",
            "command" : {
                "currentOp" : 1,
                "$all" : 1
            },
            "op" : "command",
            "$db" : "admin",
            "secs_running" : 0,
            "microsecs_running" : NumberLong(68),
            "clientMetaData" : {
            "application" : {
                "name" : "MongoDB Shell"
            },
            "driver" : {
                ...
            },
            "os" : {
                ...
            }
          }
       },
       {
          "desc": "GARBAGE_COLLECTION",
          "garbageCollection": {
             "databaseName": "testdb",
             "collectionName": "testCollectionA"
          },
          "secs_running": 3,
          "microsecs_running": NumberLong(3123456)
       },
       {
```

```
            "desc": "GARBAGE_COLLECTION",
            "garbageCollection": {
               "databaseName": "testdb",
               "collectionName": "testCollectionB"
            },
            "secs_running": 4,
            "microsecs_running": NumberLong(4123456)
         }
    ],
    "ok" : 1
}
```

以下是 "desc" 欄位的有效值：

- **INTERNAL** — 內部系統任務，例如游標清除或過時的使用者清除任務。
- **TTLMonitor** — 存留時間 (TTL) 監控執行緒。其執行狀態會反映在 "active" 欄位。
- **GARBAGE_COLLECTION** — 內部垃圾收集器執行緒。
- **CONN** — 使用者查詢。
- **CURSOR** — 操作是等待使用者呼叫「getMore」命令以取得下一批結果的閒置游標。在此狀態下，游標會耗用記憶體，但不會耗用任何運算。

前述輸出也會列出系統中執行的所有使用者查詢。每個使用者查詢會在資料庫和集合的內容中執行，這兩者的聯集稱為命名空間。"ns" 欄位會提供每個使用者查詢的命名空間。

有時您需要列出在特定命名空間中執行的所有使用者查詢。因此，之前的輸出必須經過 "ns" 欄位篩選。下面是為達到篩選輸出的範例查詢。查詢會列出資料庫 "db" 和集合 "test" (即 "db.test" 命名空間) 中目前正在執行的所有使用者查詢。

```
db.adminCommand({aggregate: 1,
    pipeline: [{$currentOp: {allUsers: true, idleConnections: true}},
               {$match: {ns: {$eq: "db.test"}}}],
    cursor: {}
});
```

身為系統的主要使用者，您可以看到所有使用者和所有內部系統任務的查詢。所有其他使用者都只能查看其各自查詢。

如果查詢和內部系統任務的總數超過預設批次處理游標大小，則 mongo 殼層將自動產生迭代器物件 'it' 以查看其餘結果。請持續執行命令 'it'，直到所有結果都已用盡。

如何知道查詢何時進行？

使用者查詢執行緩慢的原因，可能是查詢計畫不佳，或因為資源爭用而遭封鎖。偵錯此類查詢需要進行多個步驟，而其有時需要多次執行相同的步驟。

偵錯的第一步是列出長時間執行或封鎖的所有查詢。下列查詢列出執行超過 10 秒或等待資源的所有使用者查詢。

```
db.adminCommand({aggregate: 1,
                 pipeline: [{$currentOp: {}},
                            {$match: {$or: [{secs_running: {$gt: 10}},
                                            {WaitState: {$exists: true}}]}},
                            {$project: {_id:0,
                                        opid: 1,
                                        secs_running: 1,
                                        WaitState: 1,
                                        blockedOn: 1,
                                        command: 1}}],
                 cursor: {}
                });
```

請定期重複上述查詢，判斷查詢清單是否變更，並找出長時間執行的查詢或遭封鎖的查詢。

如果您查詢的輸出文件中包含 WaitState 欄位，則表示資源爭用是造成查詢執行速度緩慢或遭封鎖的原因。資源爭用的可能成因是 IO、內部系統任務或其他使用者查詢。

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "waitedMS" : NumberLong(0),
    "cursor" : {
        "firstBatch" : [
            {
                "opid" : 201,
                "command" : {
                    "aggregate" : ...
                },
                "secs_running" : 208,
                "WaitState" : "IO"
            }
        ],
        "id" : NumberLong(0),
```

```
        "ns" : "admin.$cmd"
    },
    "ok" : 1
}
```

如果同時在相同執行個體上執行的不同集合中有多個查詢，或執行個體大小對執行查詢的資料集而言太小，則 I/O 可能成為瓶頸。如果查詢是唯讀查詢，則您可以在不同複本上分隔每個集合的查詢，有效緩解這個情況。若是跨不同集合的並行更新，或執行個體的大小對資料集而言太小，則您可以向上擴展執行個體來進行緩解。

如果資源爭用是由其他使用者查詢所引起，則輸出文件中的 "blockedOn" 欄位將包含影響此查詢之查詢的 "opid"。使用 "opid" 追蹤所有查詢的 "WaitState" 和 "blockedOn" 欄位鏈結，以在鏈結的前端尋找查詢。

如果鏈結前端的任務為內部任務，則唯一的緩解措施是終止查詢，並在一段時間後重新執行該查詢。

在以下的範例輸出中，其尋找查詢在另一個任務擁有的集合鎖上遭到封鎖。

```
{
    "inprog" : [
        {
            "client" : "...",
            "desc" : "Conn",
            "active" : true,
            "killPending" : false,
            "opid" : 75,
            "ns" : "...",
            "command" : {
                "find" : "...",
                "filter" : {

                }
            },
            "op" : "query",
            "$db" : "test",
            "secs_running" : 9,
            "microsecs_running" : NumberLong(9449440),
            "threadId" : 24773,
            "clientMetaData" : {
                "application" : {
                   "name" : "MongoDB Shell"
                },
```

```
                "driver" : {
                    ...
                },
                "os" : {
                    ...
                }
            },
            "WaitState" : "CollectionLock",
            "blockedOn" : "INTERNAL"
        },
        {
            "desc" : "INTERNAL"
        },
        {
            "client" : "...",
            ...
            "command" : {
                "currentOp" : 1
            },
            ...
        }
    ],
    "ok" : 1
}
```

如果 "WaitState" 具有值 "Latch"、"SystemLock"、"BufferLock"、"BackgroundActivity" 或 "Other"，則資源爭用來源為內部系統任務。如果情況持續很長一段時間，則唯一的緩解措施是終止該查詢，並在一段時間後重新執行該查詢。

如何判斷系統突然緩慢執行的原因？

下面列出系統變慢的一些常見原因：

- 並行查詢之間的資源爭用過多
- 作用中並行查詢的數量會隨著時間增加
- 內部系統任務，例如 "GARBAGE_COLLECTION"

若要隨著時間監控系統使用量，請定期執行下列 "currentOp" 查詢，並將結果輸出到外部存放區。該查詢會計算系統中每個命名空間的查詢和操作數量。接著，您就可以分析系統使用量結果來了解系統負載，並採取適當行動。

```
db.adminCommand({aggregate: 1,
                 pipeline: [{$currentOp: {allUsers: true, idleConnections: true}},
                            {$group: {_id: {desc: "$desc", ns: "$ns", WaitState:
 "$WaitState"}, count: {$sum: 1}}}],
                 cursor: {}
                });
```

此查詢會傳回每個命名空間中執行之所有查詢和所有內部系統任務的彙總，以及每個命名空間專屬的等待狀態數量。

此操作的輸出將會如下所示 (JSON 格式)。

```
{
    "waitedMS" : NumberLong(0),
    "cursor" : {
        "firstBatch" : [
            {
                "_id" : {
                    "desc" : "Conn",
                    "ns" : "db.test",
                    "WaitState" : "CollectionLock"
                },
               "count" : 2
            },
            {
                "_id" : {
                    "desc" : "Conn",
                    "ns" : "admin.$cmd"
                },
                "count" : 1
            },
            {
                "_id" : {
                    "desc" : "TTLMonitor"
                },
                "count" : 1
            }
        ],
        "id" : NumberLong(0),
        "ns" : "admin.$cmd"
    },
    "ok" : 1
```

```
}
```

在前述輸出中，命名空間 "db.test" 中有 2 個使用者查詢在集合鎖定上遭到封鎖：其中 1 個查詢位於命名空間 "admin.$cmd" 中，另一個位於內部 "TTLMonitor" 任務中。

如果輸出指出許多查詢處於封鎖等待狀態，請參閱如何尋找並終止長時間執行或封鎖的查詢？

如何判斷一或多個叢集執行個體上高 CPU 使用率的原因？

下列各節可協助您識別高執行個體 CPU 使用率的原因。結果可能因工作負載而有差異。

- 若要判斷執行個體突然執行緩慢的原因，請參閱如何判斷系統突然緩慢執行的原因？
- 若要識別和終止特定執行個體的長時間執行查詢，請參閱如何尋找並終止長時間執行或封鎖的查詢？
- 若要了解查詢是否正在進行，請參閱如何知道查詢何時進行？
- 若要判斷查詢需要長期執行時間的原因，請參閱如何查看查詢計畫並最佳化查詢？
- 若要追蹤隨時間演進的長時間執行查詢，請參閱分析 Amazon DocumentDB 操作。

根據造成高執行個體 CPU 使用率的原因，執行下列一項或多項動作可能有所幫助。

- 如果主要執行個體顯示高 CPU 使用率，但複本執行個體未顯示同樣情況，則請考慮透過用戶端讀取偏好設定 (例如 secondaryPreferred)，將讀取流量分配至所有複本。如需詳細資訊，請參閱以複本集的形式連線至 Amazon DocumentDB。

  使用複本進行讀取時，因為主要執行個體可以處理更多寫入流量，進而有效提升叢集資源的使用效能。來自複本的讀取最終會保持一致。
- 如果寫入工作負載的結果是造成高 CPU 使用率，則將叢集執行個體的大小變更為較大的執行個體類型，將會增加可用於服務工作負載的 CPU 核心數量。如需詳細資訊，請參閱管理 Amazon DocumentDB 執行個體及執行個體類別規格。
- 如果所有叢集執行個體都呈現高 CPU 使用率，且工作負載使用複本進行讀取，則新增更多複本到叢集時，可供讀取流量使用的資源將會增加。如需詳細資訊，請參閱將 Amazon DocumentDB 執行個體新增至叢集。

如何判斷執行個體上的開啟游標？

連線至 Amazon DocumentDB 執行個體時，您可以使用 命令db.runCommand("listCursors")列出該執行個體上的開啟游標。指定 Amazon DocumentDB 執行個體上隨時最多可開啟 4，560 個作用

中游標的配額，視執行個體類型而定。關閉不再使用的游標，因為游標會利用執行個體上的資源。如需詳細資訊，請參閱Amazon DocumentDB 配額。

```
db.runCommand("listCursors")
```

如何判斷目前的 Amazon DocumentDB 引擎版本？

若要判斷您目前的 Amazon DocumentDB 引擎版本，請執行下列命令。

```
db.runCommand({getEngineVersion: 1})
```

此操作的輸出將會如下所示 (JSON 格式)。

```
{ "engineVersion" : "2.x.x", "ok" : 1 }
```

> **Note**
>
> Amazon DocumentDB 3.6 的引擎版本為 1.x.x，Amazon DocumentDB 4.0 的引擎版本為 2.x.x，Amazon DocumentDB 5.0 的引擎版本為 3.x.x，Amazon DocumentDB 8.0 的引擎版本為 4.x.x。

如何分析索引用量並識別未使用的索引？

若要識別指定集合的索引，請執行下列命令：

```
db.collection.getIndexes()
```

若要分析在集合上執行的操作期間使用多少索引，可以使用 `collStats`和 `indexStats`命令。若要檢視使用索引執行的掃描總數（索引掃描）與不使用索引執行的掃描總數（集合掃描），請執行下列命令：

```
db.collection.stats()
```

此命令的輸出包含下列值：

- **`idxScans`** - 在此集合上使用索引執行的掃描次數。

- **collScans** - 在此集合上執行的掃描數目，而不使用索引。這些掃描會涉及逐一查看集合中的文件。
- **lastReset** - 這些計數器上次重設的時間。此命令提供的統計資料會在啟動/停止叢集或向上/向下擴展執行個體時重設。

您可以在下列命令的輸出中找到每個索引的使用量明細。最佳實務是定期識別和移除未使用的索引，以提高效能並降低成本，因為它消除了用於維護索引的不必要的運算、儲存和 I/O。

```
db.collection.aggregate([{$indexStats:{}}]).pretty()
```

此命令的輸出為集合上建立的每個索引提供下列值：

- **ops** - 使用索引的操作數目。如果您的工作負載已執行足夠長的時間，而且您確信您的工作負載處於穩定狀態，ops 的零值表示完全不使用索引。
- **numDocsRead** - 在此索引操作期間讀取的文件數量。
- **since** - Amazon DocumentDB 開始收集索引用量統計資料以來的時間，通常是自上次資料庫重新啟動或維護動作以來的值。
- **size** - 此索引的大小，以位元組為單位。

下列範例是執行上述命令的範例輸出：

```
{
    "name" : "_id_",
    "key" : {
        "_id" : 1
    },
    "host" : "example-host.com:12345",
    "size" : NumberLong(...),
    "accesses" : {
        "ops" : NumberLong(...),
        "docsRead" : NumberLong(...),
        "since" : ISODate("...")
    },
    "cacheStats" : {
        "blksRead" : NumberLong(...),
        "blksHit" : NumberLong(...),
        "hitRatio" : ...
    }
}
```

```
{
    "name" : "x_1",
    "key" : {
        "x" : 1
    },
    "host" : "example-host.com:12345",
    "size" : NumberLong(...),
    "accesses" : {
        "ops" : NumberLong(...),
        "docsRead" : NumberLong(...),
        "since" : ISODate("...")
    },
    "cacheStats" : {
        "blksRead" : NumberLong(...),
        "blksHit" : NumberLong(...),
        "hitRatio" : ...
    }
}
```

若要判斷集合的整體索引大小，請執行下列命令：

```
db.collection.stats()
```

若要刪除未使用的索引，請執行下列命令：

```
db.collection.dropIndex("indexName")
```

如何識別缺少的索引？

您可以使用 Amazon DocumentDB 分析工具來記錄慢查詢。在慢速查詢日誌中重複出現查詢，可能表示需要額外的索引來改善該查詢的效能。

您可以透過尋找具有至少執行一個COLLSCAN階段之一或多個階段的長時間執行查詢來識別有用索引的機會，這表示查詢階段必須讀取集合中的每個文件，以提供查詢的回應。

下列範例顯示在大型集合上執行的計程車乘車查詢。

```
db.rides.count({"fare.totalAmount":{$gt:10.0}}))
```

為了執行這個範例，查詢必須執行集合掃描 (即讀取集合中的每個文件)，因為該 `fare.totalAmount` 欄位上沒有索引。此查詢的 Amazon DocumentDB 分析工具輸出如下所示：

```
{
    ...
    "cursorExhausted": true,
    "nreturned": 0,
    "responseLength": 0,
    "protocol": "op_query",
    "millis": 300679,
    "planSummary": "COLLSCAN",
    "execStats": {
        "stage": "COLLSCAN",
        "nReturned": "0",
        "executionTimeMillisEstimate": "300678.042"
    },
    "client": "172.31.5.63:53878",
    "appName": "MongoDB Shell",
    "user": "example"
}
```

為了加快這個例子中的查詢，您可在 `fare.totalAmount` 建立一個索引，如下圖所示。

```
db.rides.createIndex( {"fare.totalAmount": 1}, {background: true} )
```

> **Note**
>
> 在前景中建立的索引 (也就是說，如果建立索引時未提供 {background:true} 選項) 會採用獨佔寫入鎖定，這樣可以防止應用程式在索引建立完成之前將資料寫入集合。在生產叢集上建立索引時，請注意這項潛在影響。建立索引時，請設定 {background:true}。

一般而言，您會希望在具有高基數 (例如，大量唯一值) 的欄位上建立索引。在低基數的欄位上建立索引可能會導致有大型索引未使用。建立查詢計劃時，Amazon DocumentDB 查詢最佳化工具會考慮集合的整體大小和索引的選擇性。有時候，您會看到查詢處理器選擇一個 COLLSCAN，即使索引已存在。當查詢處理器預估，相較於掃描整個集合，利用索引並不會產生效能優勢時，就會發生這種情況。如果您想強制查詢處理器利用一個特定索引，您可以使用 `hint()` 運算子，如下圖所示。

```
db.collection.find().hint("indexName")
```

如何判斷資料庫集合膨脹？

當集合的大小因資料庫頁面內累積過時或過時的文件或分段而變大時，就會發生集合膨脹。報告的百分比代表未來文件可以使用的文件空間量。此膨脹會耗用緩衝區快取和儲存體中的空間。若要移除膨脹，集合必須透過傾印/還原重新載入，或在維護時段期間使用遷移回送和切換。

Example範例

執行下列命令來判斷集合的未使用儲存：

```
db.runCommand({collStats:'coll'})
```

結果如下所示：

```
{
        "ns" : "test.coll",
        "count" : 7500,
        "size" : 23250,
        "avgObjSize" : 31,
        "storageSize" : 106496,
        "unusedStorageSize" : {
                "unusedBytes" : 16384,
                "unusedPercent" : 25.12
        },
        "compression" : {
                "enable" : false
        },
        "capped" : false,
        "nindexes" : 1,
        "totalIndexSize" : 57344,
        "indexSizes" : {
                "_id_" : 57344
        },
        "collScans" : 4,
        "idxScans" : 10000,
        "opCounter" : {
                "numDocsIns" : 1000,
                "numDocsUpd" : 0,
                "numDocsDel" : 250
        },
        "cacheStats" : {
                "collBlksHit" : 3570,
```

```
                "collBlksRead" : 8,
                "collHitRatio" : 99.7765,
                "idxBlksHit" : 12293,
                "idxBlksRead" : 6,
                "idxHitRatio" : 99.9513
        },
        "lastReset" : "2024-12-18 00:30:21.552019+00",
        "ok" : 1,
        "operationTime" : Timestamp(1734632375, 1)
}
```

有用的查詢摘要

下列查詢有助於監控 Amazon DocumentDB 中的效能和資源使用率。

- 使用下列命令來檢視特定集合的統計資料，包括操作計數器、快取統計資料、存取統計資料和大小統計資料：

```
db.collection.stats()
```

- 使用下列命令來檢視在集合上建立的每個索引的統計資料，包括索引的大小、索引特定的快取統計資料和索引用量統計資料：

```
db.collection.aggregate([{$indexStats:{}}]).pretty()
```

- 使用以下查詢來列出所有活動。

```
db.adminCommand({currentOp: 1, $all: 1});
```

- 下面程式碼會列出所有長時間執行或遭封鎖的查詢。

```
db.adminCommand({aggregate: 1,
                 pipeline: [{$currentOp: {}},
                            {$match: {$or: [{secs_running: {$gt: 10}},
                                            {WaitState: {$exists: true}}]}},
                            {$project: {_id:0,
                                        opid: 1,
                                        secs_running: 1,
                                        WaitState: 1,
                                        blockedOn: 1,
                                        command: 1}}],
                 cursor: {}
```

```
                });
```

- 下列程式碼會終止查詢。

```
db.adminCommand({killOp: 1, op: <opid of running or blocked query>});
```

- 使用下列程式碼來取得系統狀態的彙總檢視。

```
db.adminCommand({aggregate: 1,
                 pipeline: [{$currentOp: {allUsers: true, idleConnections: true}},
                            {$group: {_id: {desc: "$desc", ns: "$ns", WaitState:
 "$WaitState"}, count: {$sum: 1}}}],
                 cursor: {}
                });
```

Amazon DocumentDB 中的垃圾回收

Amazon DocumentDB 實作多版本並行控制 (MVCC) 資料庫架構，為每個更新操作建立新的文件和索引項目版本。此架構提供啟用交易隔離，防止一個交易的變更出現在另一個交易中。

主題

- 了解 Amazon DocumentDB 中的垃圾回收
- 垃圾回收程序
- 儲存架構和擴充儲存
- 監控垃圾回收
- collStats 輸出範例
- 常見問答集

了解 Amazon DocumentDB 中的垃圾回收

垃圾回收 (GC) 是一種自動化背景程序，可在 Amazon DocumentDB 中維持最佳的系統效能和可用性。如同許多現代資料庫，Amazon DocumentDB 的 MVCC 架構會在每次更新時建立新的文件和索引版本。每個寫入操作都會從有限計數器使用唯一的 MVCC ID。這些 IDs會識別文件版本所屬的交易，以及文件版本是否已遞交或復原。隨著時間的推移，這些舊版本及其 MVCC IDs會累積，需要清除以防止效能降低。

垃圾回收的函數

垃圾收集器提供三個基本函數：

- 回收儲存空間 — 它移除作用中查詢不再需要的過時文件和索引版本，為未來的寫入操作釋放空間。
- 防止 MVCC ID 溢位 — 透過管理 MVCC ID 的有限計數器來防止 MVCC IDs 溢位。如果沒有此管理，計數器最終會達到其限制，強制資料庫進入暫時唯讀模式，直到 IDs回收為止。
- 維持查詢效能 — 它透過消除會累積和減慢查詢處理速度的無效文件版本，來維持最佳的查詢效能。

垃圾回收程序

GC 程序會依集合操作，並可在不同的集合上同時執行多個程序。程序包含四個循序階段：

1. 識別 — 系統會識別作用中交易或查詢不再參考的文件和索引版本。
2. 記憶體載入 — 如果舊文件和索引項目尚未存在，則會載入記憶體。
3. 刪除 — 永久刪除過時的版本以回收儲存空間。
4. MVCC ID 回收 — 系統會從已刪除的版本回收 MVCC IDs 以進行新操作。

當垃圾收集完成舊文件版本的處理時，它會從系統中移除最舊的 MVCC IDs。此清除對於回收 MVCC ID 來防止 MVCC IDs 溢位至關重要，使其可用於整個叢集的新寫入操作。如果沒有此回收程序，系統最終會耗盡其有限的 MVCC ID 計數器，並進入唯讀狀態。

垃圾回收排程

垃圾回收會定期在背景中自動執行。時間和頻率會根據系統負載、可用資源、寫入磁碟區和 MVCC ID 耗用量層級動態調整。在高寫入活動期間，GC 程序會更頻繁地執行，以管理增加的文件版本數量。

儲存架構和擴充儲存

Amazon DocumentDB 使用複雜的儲存架構，將文件儲存區分成兩個不同的區段：

基本儲存區段

基本儲存區段包含主要文件資料和中繼資料。此區段存放：

- 符合標準頁面大小 (8 KB) 的文件內容。
- 文件中繼資料和結構資訊。
- 主要索引及其項目。

- 集合層級統計資料和組態。

擴充儲存區段

延伸的儲存區段使用專門的大型文件物件存放區，旨在處理超過標準儲存頁面大小的文件。此區段提供：

- 高效率大型文件處理 — 大於基本儲存閾值的文件會自動移至擴充儲存區段。
- 最佳化儲存配置 — 區段使用針對大型物件最佳化的不同儲存格式，減少分段並改善存取模式。
- 獨立垃圾回收 — 延伸的儲存區段具有自己的垃圾回收程序，可以獨立於基本儲存清理執行。
- 透明存取 — 應用程式無縫存取大型文件，而不需要知道哪些儲存區段包含資料。

延伸儲存區段特別有益於：

- 包含大型內嵌陣列之文件的集合。
- 具有廣泛巢狀結構的文件。
- 儲存二進位資料或大型文字欄位的集合。
- 混合文件大小的應用程式，其中某些文件明顯超過平均大小。

監控垃圾回收

叢集層級指標

`AvailableMVCCIds`

- 位置 — Amazon CloudWatch
- 描述 — 顯示最大限制為 18 億的剩餘寫入操作數量的計數器。當此計數器達到零時，您的叢集會進入唯讀模式IDs直到 ID 回收和回收為止。計數器會隨著每次寫入操作而減少，並隨著垃圾回收回收舊的 MVCC IDs 而增加。
- 建議 — 當 Amazon DocumentDB 5.0 和更新版本上的值低於 9 億，或 Amazon DocumentDB 3.6 和 4.0 上的值低於 13 億時，設定警示。此提早警告可讓您採取稍後討論的建議步驟。

`LongestActiveGCRuntime`

- 位置 — Amazon CloudWatch

- 描述 — 最長作用中垃圾回收程序的持續時間，以秒為單位。每分鐘更新一次，並僅追蹤作用中的操作，不包括在一分鐘時段內完成的程序。
- 建議 – 與gcRuntimeStats歷史資料進行比較，以識別異常的垃圾收集行為，例如在大量刪除期間延長執行時間。

集合層級指標

MVCCIDStats: MVCCIdScale

- 位置 — 資料庫 collStats 命令
- 描述 — 以 0 到 1 的規模測量 MVCC ID 存留期，其中 1 表示叢集進入唯讀狀態之前的最大存留期。將此指標與 一起AvailableMVCCIds用於識別包含正在使叢集老化之最舊 MVCC IDs集合。
- 建議 — 將每個集合的值保持在 0.3 以下。

gcRuntimeStats

- 位置 — 資料庫 collStats 命令
- 描述 — 提供垃圾收集指標的兩個月歷史記錄，包括總執行次數、平均持續時間和最長持續時間。僅包含持續超過五分鐘的垃圾回收操作，以確保有意義的統計資料。

storageSizeStats

- 位置 — 資料庫 collStats 命令
- 描述 — 提供不同儲存區段的儲存使用率詳細明細：
  - storageSegmentBase — 標準文件的基礎儲存區段所使用的儲存體
  - storageSegmentExtended — 擴充儲存區段用於大型文件的儲存體
- 用量 — 協助識別具有大量大型文件儲存的集合，並了解儲存分佈模式。

unusedStorageSize（集合層級）

- 位置 — 資料庫 collStats 命令
- 描述 — 根據取樣的統計資料，估計集合中未使用的儲存空間。它包含已刪除文件和空白區段的空間。指標同時提供合併總計和每個區段明細：
  - 在所有儲存區段unusedPercent中結合 unusedBytes和

- storageSegmentBase — 特別在基本儲存區段中未使用的空間
- storageSegmentExtended — 在擴充儲存區段中特別未使用的空間

documentFragmentStats

- 位置 — 資料庫 collStats 命令
- 描述 — 提供有關集合中文件片段和無效資料的詳細資訊。文件片段代表資料庫引擎使用的內部儲存單位，而無效片段則表示無法再存取但尚未回收的資料。此指標包括：
  - totalDocFragmentsCount — 集合中的文件片段總數
  - deadDocFragmentsCount — 包含無效（無法存取）資料的片段數量
  - deadDocFragmentsPercent — 包含無效資料的片段百分比
  - deadDocFragmentBytes — 無效文件片段耗用的預估位元組數
  - storageSegmentBase 和 的每個區段明細 storageSegmentExtended
- 用量 — 監控此指標以了解垃圾收集的有效性，並識別可能受益於維護操作的收集。高百分比的無效片段表示垃圾回收可能落後，或集合將受益於最佳化。

索引層級指標

unusedStorageSize（索引層級）

- Location — 資料庫 indexStats 命令
- 描述 — 根據取樣的統計資料，估計索引中未使用的儲存空間。它包含來自過時索引項目和空區段的空間。
- 建議 — 使用 reIndex命令來重建索引，無需停機時間並回收未使用的空間。如需詳細資訊，請參閱管理索引。

collStats 輸出範例

下列範例顯示具有垃圾收集和儲存指標的典型collStats輸出：

```
{
    "ns" : "Mvcc_consumption_test_db.mvcc_test_collection",
    "MVCCIdStats" : {
        "MVCCIdScale" : 0.03
    },
```

```
    "gcRuntimeStats" : {
        "numRuns" : 1,
        "historicalAvgRuntime" : 3295,
        "historicalMaxRuntime" : 3295,
        "lastRuntime" : 3295,
        "lastRuntimeStart" : ISODate("2025-06-24T08:47:14Z")
    },
    "documentFragmentStats" : {
        "totalDocFragmentsCount" : 45000000,
        "deadDocFragmentsCount" : 2250000,
        "deadDocFragmentsPercent" : 5.0,
        "deadDocFragmentBytes" : 98304000,
        "storageSegmentBase" : {
            "totalDocFragmentsCount" : 30000000,
            "deadDocFragmentsCount" : 1500000,
            "deadDocFragmentsPercent" : 5.0,
            "deadDocFragmentBytes" : 65536000
        },
        "storageSegmentExtended" : {
            "totalDocFragmentsCount" : 15000000,
            "deadDocFragmentsCount" : 750000,
            "deadDocFragmentsPercent" : 5.0,
            "deadDocFragmentBytes" : 32768000
        }
    },
    "collScans" : 14,
    "count" : 30000000,
    "size" : 1320000000,
    "avgObjSize" : 44,
    "storageSize" : 6461497344,
    "storageSizeStats" : {
        "storageSegmentBase" : 4307664896,
        "storageSegmentExtended" : 2153832448
    },
    "capped" : false,
    "nindexes" : 2,
    "totalIndexSize" : 9649553408,
    "indexSizes" : {
        "_id_" : 1910661120,
        "c_1" : 7738892288
    },
    "unusedStorageSize" : {
        "unusedBytes" : 4201881600,
        "unusedPercent" : 65.05,
```

```
        "storageSegmentBase" : {
            "unusedBytes" : 2801254400,
            "unusedPercent" : 65.05
        },
        "storageSegmentExtended" : {
            "unusedBytes" : 1400627200,
            "unusedPercent" : 65.05
        }
    },
    "cacheStats" : {
        "collBlksHit" : 171659016,
        "collBlksRead" : 754061,
        "collHitRatio" : 99.5627,
        "idxBlksHit" : 692563636,
        "idxBlksRead" : 1177921,
        "idxHitRatio" : 99.8303
    },
    "idxScans" : 41823984,
    "opCounter" : {
        "numDocsIns" : 0,
        "numDocsUpd" : 20911992,
        "numDocsDel" : 0
    },
    "lastReset" : "2025-06-24 05:57:08.219711+00",
    "ok" : 1,
    "operationTime" : Timestamp(1750968826, 1)
}
```

常見問答集

如何識別垃圾回收是否無法有效運作？

監控這些警告訊號，指出垃圾回收效率不佳：

- 過多的集合 Bloat — 在大量寫入或大量刪除期間穩定增加`unusedStorageSize`指標，尤其是大型索引。
- 高無效片段百分比 — `documentFragmentStats`持續顯示高`deadDocFragmentsPercent`值（超過 10-15%)。
- 查詢延遲降級 — 由於累積的無效文件而增加的查詢延遲。
- 延長 GC 持續時間 — 垃圾回收操作耗時超過 中的歷史平均值`gcRuntimeStats`。
- 提升 GC 處理 — 高`LongestActiveGCRuntime`表示垃圾收集器無法跟上系統需求。

垃圾回收是否會影響我的資料庫效能？

在正常情況下，垃圾回收對效能的影響最小。不過，當垃圾回收落後時，您可能會遇到：

- 從累積的無效文件增加儲存成本。
- 因索引項目過時而降低查詢效能。
- 如果 MVCC IDs，則為暫時唯讀模式。
- 密集收集執行期間較高的資源使用量，特別是在較小的執行個體上。
- 降低大型文件擴充儲存區段操作的效率。

我可以手動觸發垃圾回收嗎？

否，Amazon DocumentDB 中的垃圾回收無法手動觸發。系統會自動管理垃圾回收，做為其內部維護操作的一部分。

我應該將哪些警示設定為操作最佳實務？

在叢集和集合層級設定監控，以確保 Amazon DocumentDB 系統的最佳效能。

對於叢集層級監控，請先為 `AvailableMVCCIds` Amazon DocumentDB 5.0 和更新版本上的閾值為 9 億的指標建立 Amazon CloudWatch 警示，或在 Amazon DocumentDB 3.6 和 4.0 上建立閾值為 13 億的指標。 Amazon DocumentDB 這可讓您有足夠的時間在指標達到零之前採取動作，此時您的叢集將進入唯讀模式。請記住，此指標可能會根據您的特定使用模式而波動，有些客戶會看到它低於閾值，然後隨著垃圾回收完成其工作而復原超過 15 億。

透過 Amazon CloudWatch 監控`LongestActiveGCRuntime`指標也很重要。此指標與`gcRuntimeStats`可協助您了解廢棄項目收集在系統中的效率。

對於集合層級監控，請專注於這些關鍵指標：

- `MVCCIdScale` — 留意顯示 MVCC IDs 正在老化且可能需要注意的增加值。
- `gcRuntimeStats` — 識別需要異常長時間或持續數天的垃圾回收程序。
- `documentFragmentStats` — 監控`deadDocFragmentsPercent`值 - 持續高百分比 （超過 10-15%) 可能表示垃圾回收落後。
- `storageSizeStats` 和 `unusedStorageSize` — 追蹤儲存使用率模式，並識別任一儲存區段中具有顯著未使用空間的集合。

具有頻繁寫入操作的集合需要額外注意，因為它們會為垃圾收集器產生更多工作。更頻繁地檢查這些指標是否有具有繁重寫入活動的集合，以確保垃圾收集與您的工作負載保持同步。

請注意，這些監控建議做為起點。當您更熟悉系統的行為時，建議您調整這些閾值，以更符合您的特定使用模式和需求。

如果我的 **AvailableMVCCIds** 低於警示閾值，該怎麼辦？

如果您的AvailableMVCCIds指標低於警示閾值，請立即採取行動，以防止您的叢集進入唯讀模式。首先，擴展執行個體大小，為垃圾收集器提供更多運算資源。這可讓您的應用程式繼續正常操作，同時為垃圾收集器提供追上所需的額外能力。

如果單獨擴展無法改善情況，請考慮減少您的寫入操作。使用 MVCCIdScale 指標來識別哪些特定集合包含需要注意的較舊 MVCC IDs。此外，監控 documentFragmentStats 以識別具有高無效片段百分比的集合，這些百分比可能會導致垃圾回收效率低下。

識別這些集合之後，您可能需要暫時減少這些集合的寫入操作，以允許垃圾收集追上進度。在復原期間， 會密切監控 AvailableMVCCIds 指標，以確保您的動作具有所需的效果。一旦AvailableMVCCIds值回到 15 億或更高，您的叢集就會被視為正常運作。

請記住，這些步驟是預防措施，可協助您的系統在達到關鍵狀態之前復原。看到指標低於閾值後，您越早採取動作，就越有可能避免對寫入操作造成任何影響。

批次操作是否會影響 MVCC ID 耗用率？

是。由於單一原子交易中執行的批次只會使用一個 MVCC ID，因此將寫入分組成批次會降低使用 MVCC IDs的速率。最佳實務是，updateMany盡可能使用 insertMany和 將 MVCC ID 耗用量降至最低，並減少垃圾回收的負載。

為什麼在達到特定閾值之前都無法**AvailableMVCCIds**復原？

對於尚未使用足夠 MVCC IDs 來觸發垃圾回收週期的寫入主導型工作負載，預期 會暫時穩定下降AvailableMVCCIds而沒有復原。當符合下列任何條件時，垃圾回收會觸發：集合內文件的特定部分已淘汰、插入新文件的特定部分，或AvailableMVCCIds低於回收閾值。

在AvailableMVCCIds低於回收閾值之前，不會發生強制回收。完成後，回收會觸發並AvailableMVCCIds復原。不過，長時間執行的交易和開啟的游標可以鎖定回收期限，並防止在達到閾值之後回收。

錯誤代碼

當命令失敗時，您會收到來自 Amazon DocumentDB （具有 MongoDB 相容性） 的錯誤，其中包含數值錯誤code和描述性訊息 ()errmsg。使用下表來識別 Amazon DocumentDB 8.0 可能傳回用戶端應用程式的錯誤代碼。當錯誤碼對應至單一條件時，描述會符合伺服器傳回的訊息文字。涵蓋多個條件的程式碼具有一般描述。

> **Important**
>
> 請勿建置依賴特定數值錯誤代碼的應用程式邏輯，因為 Amazon DocumentDB 可能會在主要版本之間變更錯誤代碼。

錯誤碼	說明
1	找不到項目
2	命令引數、選項或欄位值無效或不支援。
4	缺少必要欄位或組態值格式不正確。
5	檢視定義會參考本身、建立週期或發生檔案系統錯誤。
9	命令、欄位、提示或繼續字符無法剖析或格式無效。
10	找不到目標集合，或未指定集合。
11	找不到參考的使用者，或目標集合已存在。
13	您無權執行此命令，或您的身分驗證無效。
14	欄位或值的資料類型錯誤。
15	值在評估期間溢位其允許的範圍。
16	寫入批次為空白、格式不正確或超過允許的操作計數。
17	並非所有物件都有相同的類型
18	驗證失敗，因為登入資料無效或使用者不存在。

錯誤碼	說明
19	找不到資料處理基本操作或基本類型
20	在此內容中不允許請求的操作。
26	目標命名空間或集合不存在。
27	請求的索引不存在或沒有適當的索引可用。
37	管道或彙總階段引數遺失、空白或無效。
38	找不到資料庫名稱
39	操作已取消，因為它超過 maxTimeMS、執行時間過長或超過可用的記憶體。
40	更新與另一個欄位衝突，或寫入會受到垃圾回收調節或停用。
4.1	JavaScript 錯誤
43	指定的游標不再存在或已過期。
44	陣列索引超出範圍
48	具有該名稱的集合、檢視或命名空間已存在。
51	非法更新運算子
52	欄位名稱使用不適用於儲存的預留 '$' 字首。
53	_id 欄位具有無效或不支援的類型，例如陣列或 regex。
54	無法推斷要設定的查詢欄位
55	交易已遞交
56	請求中的欄位名稱為空白，這是不允許的。
57	其他交易使交易失效
58	欄位值太長

錯誤碼	說明
59	請求的命令未知或不支援。
61	無效的集合
62	無法變更集合
65	發生批次操作錯誤
66	嘗試修改不可變 _id 欄位的更新。
67	由於無效或不支援的規格，無法建立索引。
68	已存在具有衝突屬性的索引。
70	與 $snippet 搭配使用的欄位未設定為 snippet 資料欄
71	索引已存在，具有不同的選項
72	為命令或索引指定的一或多個選項無效。
73	命名空間、資料庫或集合名稱無效或不允許用於此操作。
74	集合未定義文字索引。
75	此集合會定義文字索引，但不會為此欄位定義。
77	無法搭配排序使用快照
78	找不到文件
80	查詢超過允許的大小上限。
82	時間範圍的格式無效。有效格式：'1 m' 或 '3 y' '<number> h/d/w/s/m/y'
85	具有該名稱的索引已存在，但具有不同的選項。
86	請求的索引與具有相同名稱的現有索引衝突。
87	$sort 索引鍵排序必須是 1（用於遞增）或 -1（用於遞減）

錯誤碼	說明
92	沒有足夠的空間進行操作
93	使用者已存在
94	目標集合、叢集或資源權限無效或無法使用。
95	資料庫不存在
96	'writeConcern' 參數無效
97	密碼不能為空白
99	未定義的角色或權限
102	無法新增已內建的角色
103	無法從集合中擷取有效的角色。
105	使用者無效或授權程式庫無法初始化。
107	資料庫無法從其他資料庫授予角色
111	提供更新文件時，必須準確指定 _id
112	寫入衝突
113	指定的角色不存在。
114	角色已存在
115	指定時 anyResource 應為 true
116	資源應該有 db 和集合集、叢集集anyResource集
117	操作會與另一個進行中的操作衝突，例如作用中的交易。
120	無法指定 'showPrivileges'，因為我們顯示所有使用者
121	文件驗證失敗

錯誤碼	說明
122	更新錯誤
124	處理結構描述 ID 時發生錯誤
125	命令失敗，通常是因為參考不明確或缺少工作階段。
126	資料庫已存在
127	資料庫名稱已存在於不同的案例名稱中
128	太多欄位
129	等待捨棄資料庫
130	參數值錯誤
136	游標位置遺失，或未啟用必要的變更串流功能。
139	JS 解譯器失敗
145	未在查詢中定義別名
147	不是主控
148	資源不足，無法執行查詢
149	已在查詢中定義的別名
154	引數具有非法值或不正確（非數值）類型。
155	遇到 NULL 值
156	遇到 UNDEFINED 值
157	遇到 NAN 值
158	集合已用於其他用途
159	持續時間值溢出或網路操作失敗。

錯誤碼	說明
161	請求的定序版本無效或無法使用。
163	欄位或名稱對儲存無效。
164	查詢運算子以不允許的方式巢狀化。
165	檢視超過循環中 20 或參考本身的最大巢狀深度。
166	檢視不支援此操作，或使用太多位置元素。
167	檢視不支援請求的操作或階段。
168	表達式或管道運算子無法辨識或具有無效的引數。
169	BadValue $maxTimeMS 具有非整合值
170	BadValue $maxTimeMS 超出範圍
171	索引無法建置在多個陣列欄位上。
173	略過值必須為正值
174	預期的日誌詳細程度為數字
181	索引鍵模式符合多個索引；請改為依名稱指定索引。
197	索引規格包含無效或不支援的欄位。
207	uuid 必須是具有 UUID (4) 子類型的 16 位元組二進位欄位
224	目前內容中不允許此查詢功能。
225	工作階段上已啟動較新的交易，因此較舊的交易無法繼續。
241	值無法在 $convert 中轉換為請求的類型。
250	無效的檢視
251	指定的交易號碼不符合任何作用中的交易。

錯誤碼	說明
252	已達到遞迴限制
253	無法將項目插入檢視
254	無法從檢視刪除項目
256	依 $having 子句篩選的文件
257	交易中所有操作的總大小超過允許的限制。
263	此操作無法在多文件交易內執行。
291	找不到 $geoNear 查詢的索引
300	由於 SASL 交握或工作階段錯誤，身分驗證失敗。
301	不支援身分驗證機制或請求的操作。
302	身分驗證需要額外的資料才能完成 SASL 交換。
303	不支援請求的階段、選項或角色組態。
304	不支援請求的彙總階段。
305	不支援彙總投影運算子。
306	不支援指定的彙總群組累積器運算子。
502	資料欄已存在
505	此處不允許插入
601	座標超出範圍
602	索引錯誤
700	即時模式中不允許變更此參數
701	此參數的值無效

錯誤碼	說明
703	此參數的不合法狀態轉換
999	無效的交易 ID
4570	$lookup 參數預期字串值
4572	$lookup 缺少必要欄位
10065	必須為物件 （ 例如 vars 或 $rand) 的參數會獲得非物件值。
10107	不是主控
11000	重複的索引鍵違反唯一的索引限制。
11601	操作已中斷。
11603	用戶端請求損毀或非法語法
13103	複合索引鍵太多
13297	具有相同名稱的資料庫已存在，但只有字母大小寫才不同。
13388	無法辨識的管道階段名稱：$collStats
13435	不是主伺服器且 slaveOk=false
14031	磁碟空間不足
15888	必須傳遞集合名稱才能建立
15947	群組的欄位必須在 物件中指定
15950	群組彙總欄位名稱不能是運算子
15952	未知的 $group Operator
15955	群組彙總運算子中的語法錯誤：必須指定 _id 欄位
15956	$skip 引數為負數，必須是非負數。

錯誤碼	說明
15957	限制必須指定為數字
15958	$limit 值必須是正數。
15959	比對篩選條件必須是 物件中的表達式
15969	語法錯誤：$project Operator 的值必須是文件。
15972	$skip 的引數必須是數字
15973	$sort 金鑰規格必須是 物件
15974	$sort 索引鍵排序必須使用數字指定
15975	$sort 索引鍵排序必須是 1 （用於遞增） 或 -1 （用於遞減）
15976	$sort 階段必須至少有一個排序索引鍵
15981	查詢欄位不是字串
15983	代表表達式的物件必須只有一個欄位。
15992	物件表達式中不允許的欄位類型 NULL
15998	欄位路徑名稱不能是空字串。
16006	值無法轉換為日期，因為它不是有效的日期類型。
16007	值無法轉換為字串，因為預期的是字串。
16020	呼叫的表達式運算子引數數目錯誤。
16034	$substrBytes 起始索引必須是數值類型。
16035	$substrBytes 長度引數必須是數值類型。
16395	$match 彙總表達式內不允許 $where
16400	無法新增欄位的表達式，因為該欄位或其子欄位之一已有表達式。

錯誤碼	說明
16401	無法新增子欄位的表達式，因為已有套用至整個欄位的表達式
16404	在此內容中不允許 $ 表達式
16406	無法排除 _id 以外的欄位
16410	$project 欄位名稱或路徑非法開頭為 或包含 '$'。
16411	欄位名稱不能包含 null 字元 (s)
16419	欄位路徑不得包含內嵌 null 字元
16424	$match 彙總表達式內不允許 $near
16426	$nearSphere/$near 不允許在 $match 彙總表達式內
16435	管道元件必須包含一個元素
16493	嘗試建立長度超過 16MB 的字串
16554	$add 僅支援數值或日期類型。
16605	$geoNear 需要有效的「接近」欄位，並正確輸入 geoNear 選項。
16606	$geoNear 需要 'distanceField' 選項做為字串
16607	$geoNear 需要指定指定的選項做為字串。
16608	除以零
16610	$mod 獲得的除數為零。
16611	$mod 僅支援兩個運算元的數值類型。
16612	$add 表達式中只允許一個日期
16613	操作嘗試從日期減去非數字、非日期的值。
16702	$concat 僅支援字串引數。

錯誤碼	說明
16730	textIndexVersion 超出範圍
16755	無法以嚴格的繞組順序為幾何編製索引
16800	不能有 2 個地理欄位
16801	2d 必須是索引中的第一個
16804	預期位置物件，位置陣列格式不正確
16810	$mod 需要兩個值的陣列做為其參數。
16837	更新路徑無法多次修改相同的欄位。
16868	無效的變數名稱
16869	無效的變數名稱
16870	無效的變數名稱
16872	欄位路徑是空的，或單獨使用 '$' 做為 $project 中的欄位路徑。
16873	Fieldpath 的開頭不是 $
16874	$let 僅支援物件做為其引數
16875	$let 獲得無法辨識的參數。
16876	缺少 $let 的 'vars' 參數
16877	缺少 $let 的 'in' 參數
16878	$map 僅支援物件做為其引數
16879	$map 會提供無法辨識的參數。
16880	$map 缺少其所需的 'input' 參數。
16881	缺少必要欄位

錯誤碼	說明
16882	缺少 $map 的 'in' 參數
16883	$map 的輸入必須是陣列。
16898	$minDistance 必須為非負數
16899	$maxDistance 必須是數字
16900	$maxDistance 必須為非負數
16945	超過 $group 或 $lookup 的記憶體限制。傳遞 allowDiskUse：true 以選擇加入。
16949	無效的彙總選項；allowDiskUse 必須是布林值。
16990	$out 需要字串或物件引數來命名輸出目標。
16996	插入 $out 失敗
16997	$out 失敗，因為目標集合的索引在處理期間變更。
17040	$allElementsTrue 的引數必須是陣列。
17041	$anyElementTrue 的引數必須是陣列。
17042	$setIsSubset 的兩個運算元都必須是陣列。
17043	$setUnion 的所有運算元都必須是陣列。
17044	$setEquals 的運算元不是陣列。
17045	$setEquals 至少需要兩個引數。
17046	$setIsSubset 的兩個運算元都必須是陣列。
17047	$setIntersection 的所有運算元都必須是陣列。
17048	$setDifference 的兩個運算元都必須是陣列。
17049	$setDifference 的兩個運算元都必須是陣列。

錯誤碼	說明
17053	除了 $$KEEP、$DESCEND 和 $$PRUNE 變數之外，$redact 的表達式不應傳回任何內容
17080	缺少 'if' 參數至 cond
17081	缺少 'then' 參數至 cond
17082	缺少 'else' 參數至 cond
17083	$cond 的無法辨識參數
17124	$size 的引數必須是陣列。
17275	無法重新定義 ROOT
17276	找不到變數
17280	索引鍵值太大，無法編製索引。
17287	$merge 無法多次修改相同的目標資料列。
17293	文字索引選項 'textIndexVersion' 必須是數字
17302	此引數值必須大於或等於零。
17304	$geoNear 需要有效的「接近」點和地理查詢引數。
17308	$meta 獲得不支援的引數。
17310	$setIsSubset 的兩個運算元都必須是陣列。
17313	$match with $text 只允許做為第一個管道階段。
17320	createCollection 無法在包含預留名稱的命名空間上執行。
17385	無法 $out 至特殊集合
17390	$group 不支援包含樣式表達式
17419	更新後產生的文件超過允許的大小上限。

錯誤碼	說明
17420	到 upsert 的文件大於 16777216
18533	格式欄位必須是字串
18534	無法辨識的欄位至 $dateToString
18535	日期格式字串無效，因為結尾百分比符號。
18536	無效的 $dateToString 時間格式
18537	$dateToString 無法格式化 0-9999 範圍外的日期元件。
18627	缺少 $dateToString 的 'format' 參數
18628	缺少 $dateToString 的 'date' 參數
18629	不正確的引數類型
25278	$geoNear 需要 'distanceField' 引數
26823	操作 ID 無法在可用的位元中表示。
28539	不同命令的第一個引數必須是字串
28646	$Filter 預期物件做為參數
28647	$filter 的非預期欄位
28648	$Filter 需要 'input' 欄位
28649	$Filter 需要 'as' 欄位
28650	$Filter 需要 'cond' 欄位
28651	需要陣列輸入
28656	$substrBytes：無效的範圍，起始索引是 UTF-8 接續位元組。
28657	$substrBytes：範圍無效，結束索引位於 UTF-8 字元中間。

錯誤碼	說明
28664	$concatArrays 僅支援陣列輸入。
28667	$slice、$range 或 $indexOfArray 等表達式的引數數目不正確。
28680	$abs 會套用至無法否定的值，例如長度下限。
28689	運算子引數必須是陣列，但提供不同的類型。
28690	arrayElemAt 的第二個引數必須是數值
28691	arrayElemAt 的第二個引數必須以 32 位元整數表示
28714	$sqrt 的引數必須大於或等於 0
28724	$slice 的第一個引數必須是陣列。
28725	$slice 的第二個引數必須是數值。
28726	$slice 的第二個引數必須符合 32 位元整數。
28727	$slice 的第三個引數必須是數字。
28728	$slice 的第三個引數必須是 32 位元整數。
28729	$slice 的第三個引數必須是正 32 位元整數。
28745	$sample 階段規格必須是 物件
28746	$sample 的大小引數必須是數字
28747	$sample 的大小引數不得為負數
28748	$sample 收到無法辨識的選項。
28749	$sample 階段必須指定大小
28756	$log 的引數必須是數值。
28757	$log 要求其基本引數為數值。

錯誤碼	說明
28758	$log 的引數必須是正數。
28759	$log 的基礎必須是不等於 1 的正數。
28761	$log10 的引數必須是正數。
28762	$pow 的基底必須是數值。
28763	$pow 的指數必須為數字。
28764	$pow 不能取 0 的基數和負指數
28765	$ceil、$exp、$floor、$ln 或 $log10 會收到非數字類型。
28766	$ln 需要正數引數。
28812	Fieldpath 為空
28818	Fieldpath 的開頭不是 '$'
31022	此運算子需要指定 'input' 參數。
31023	此運算子需要 'regex' 參數。
31024	運算子會收到不明的引數。
31032	欄位名稱不能包含內嵌的 null 位元組。
31034	值超出 【-32768， 32767】 的允許範圍。
31095	'isoWeekYear' 必須評估為介於 1 到 9999 之間的整數。
31138	用於排序的 $meta 值不是有效的中繼資料關鍵字。
31170	預期為空物件，但提供了非空值。
31249	投影路徑與投影中的另一個欄位衝突。
31254	無法在排除投影中指定包含項目。

錯誤碼	說明
31393	$bsonSize 需要文件做為輸入。
31394	位置投影運算子不能出現在欄位路徑的中間。
34413	在查詢附近的地理中使用無效的運算子。
34435	$reverseArray 的引數必須是陣列。
34443	$range 需要數字起始值。
34444	$range 需要可表示為 32 位元整數的起始值。
34445	$range 要求其結束值為數值。
34446	$range 結束值必須符合 32 位元整數。
34447	$range 需要數值步驟值。
34448	$range 需要可表示為 32 位元整數的步驟值。
34449	$range 需要非零的步驟值
34450	$substrCP 的起始索引必須是數值類型。
34451	$substrCP 起始索引不能以 32 位元整數表示。
34452	$substrCP 的長度引數不是數字類型。
34453	$substrCP 長度不能以 32 位元整數表示。
34454	$substrCP：長度必須是非負整數。
34455	$substrCP：起始索引必須是非負整數。
34460	$zip 僅支援物件做為其引數。
34461	$zip 輸入必須是表達式陣列。
34462	'defaults' 選項必須是表達式陣列。

錯誤碼	說明
34463	$zip 的 useLongestLength 選項必須是布林值。
34464	$zip 獲得不明引數。
34465	$zip 至少需要一個輸入陣列
34466	除非 useLongestLength 為 true，否則無法指定預設值
34467	預設和輸入的長度必須相同
34468	$zip 在其輸入中收到非陣列表達式。
34471	$strLenCP 需要字串引數。
34473	$strLenBytes 需要字串引數。
40060	$switch 需要物件做為其引數。
40061	$switch 需要 'branches' 做為陣列。
40062	$switch 需要每個分支都是 物件。
40063	$switch 會在分支中指定未知的引數。
40064	$switch 需要每個分支都有 'case' 表達式。
40065	$switch 需要每個分支都有 'then' 表達式。
40066	$switch 找不到與輸入相符的分支，且未指定預設值。
40067	$switch 表達式包含未知的引數。
40068	$switch 需要至少一個分支
40075	$reduce 需要物件
40076	$reduce 的未知引數
40077	$reduce 需要 'input'

錯誤碼	說明
40078	$reduce 需要 'initialValue'
40079	$reduce 需要 'in'
40080	$reduce 的輸入必須是陣列。
40081	運算子需要 陣列做為其第二個引數。
40085	$split 需要其第一個引數才能評估為字串。
40086	$split 的第二個引數必須評估為字串。
40087	$split 需要非空白分隔符號
40090	$indexOfArray 需要陣列做為第一個引數
40091	$indexOfBytes 的第一個引數不是字串。
40092	$indexOfBytes 需要字串作為其第二個引數。
40093	$indexOfCP 需要字串做為其第一個引數。
40094	$indexOfCP 需要字串作為其第二個引數。
40096	$indexOfArray 需要整數的開始和結束索引值。
40097	$indexOfArray 需要非負數的開始和結束索引。
40148	sortByCount 欄位必須定義為物件內的 $ 字首路徑或表達式
40149	sortByCount 欄位必須指定為字串或物件
40156	計數欄位必須是非空白字串
40157	計數欄位必須是非空白字串
40158	計數欄位不能是 $ 字首路徑
40159	計數欄位不能包含 null 位元組

錯誤碼	說明
40160	計數欄位不能包含 '.'
40169	$facet 規格必須是非空白物件
40170	$facet 的引數必須是陣列
40171	$facet 規格中陣列的 元素必須是非空白物件
40172	$facet 階段中的子管道不能為空
40176	指定的衝突欄位名稱
40177	$project 和 $addFields 至少需要一個輸出欄位。
40178	無法在包含投影中排除 '_id' 以外的欄位
40179	不能在排除投影中包含欄位或重設欄位或新增運算欄位
40180	$project 不允許空白物件做為欄位值。
40181	無效的 $project ：：： 由 ：： 引起的表達式規格必須只包含一個欄位。
40183	無法在子物件內使用虛線欄位名稱。
40192	$bucket 'boundaries' 欄位必須至少有 2 個值
40193	$bucket 的「邊界」選項中的所有值都必須具有相同的類型
40194	必須排序 $bucket 的「邊界」選項，但元素不是遞增順序
40195	$bucket 'default' 欄位必須是常數表達式
40196	$bucket 'output' 欄位必須是 物件
40197	無法辨識的 $bucket 選項
40198	$bucket 需要指定 'groupBy' 和 'boundaries'
40199	$bucket 'default' 欄位必須小於最低界限，或大於或等於最高界限。

錯誤碼	說明
40200	$bucket 'boundaries' 欄位必須是陣列
40201	$bucket 階段的引數必須是 物件
40202	$bucket 'groupBy' 欄位必須定義為 $ 字首路徑或表達式
40203	欄位必須是累積器物件
40204	欄位名稱不能包含 '.'
40205	欄位名稱不能是運算子名稱
40206	累積器是統一運算子
40218	查詢需要文字分數中繼資料，但無法使用
40228	$replaceRoot 'newRoot' 表達式必須評估為 物件。
40229	$replaceRoot 需要物件做為其規格。
40230	$replaceRoot 只接受 'newRoot' 選項。
40234	$group 欄位必須是累積器物件。
40235	欄位名稱不能包含 '.' 字元。
40236	欄位名稱不能以 '$' 開頭
40237	此累積器是單元運算子，並採用單一引數。
40239	$bucketAuto 'groupBy' 欄位必須定義為 $ 字首路徑或表達式物件
40240	$bucketAuto 的引數必須是 物件
40241	$bucketAuto 'buckets' 欄位必須是數值
40242	$bucketAuto 'buckets' 欄位必須可表示為 32 位元整數
40243	$bucketAuto 'buckets' 欄位必須大於 0

錯誤碼	說明
40244	$bucketAuto 'output' 欄位必須是 物件
40245	$bucketAuto 的無法辨識選項
40246	$bucketAuto 需要指定 'groupBy' 和 'buckets'
40257	不明的四捨五入精細程度
40258	$bucketAuto 只能指定具有數值邊界的「精細度」
40260	$bucketAuto 只能指定具有非負數的「精細度」，但找到負數
40261	$bucketAuto 'granularity' 欄位必須是字串
40272	$addFields 階段規格必須是 物件。
40319	$lookup 階段規格必須是 物件
40320	缺少「從」選項至 $lookup 階段規格
40321	'from' 至 $lookup 的選項必須是字串
40322	無效的 $lookup 命名空間
40323	管道階段規格物件必須只包含一個欄位。
40324	管道包含無法辨識的階段名稱。
40331	$facet 管道內不允許此彙總階段。
40352	$project 欄位路徑不能是空字串。
40353	欄位路徑結尾不得為 '.'。
40386	$arrayToObject 需要陣列輸入。
40390	$objectToArray 需要文件做為輸入。
40391	$arrayToObject 需要一致的輸入格式。元素必須全部為陣列或全部為物件。偵測到物件，現在找到：陣列

錯誤碼	說明
40392	$arrayToObject 需要金鑰為「k」和「v」的物件。
40393	$arrayToObject 需要金鑰為 'k' 和 'v' 的物件。缺少其中一個或兩個索引鍵
40394	$arrayToObject 需要具有字串 'k' 索引鍵和 'v' 值的物件。
40395	$arrayToObject 需要金鑰值組，其金鑰為字串。
40396	$arrayToObject 需要一致的輸入格式。元素必須全部為陣列或全部為物件。偵測到陣列，現在找到： 物件
40397	$arrayToObject 需要兩個元素陣列的陣列。
40398	$arrayToObject 會獲得無法辨識類型或格式的輸入。
40400	$mergeObjects 要求所有輸入都是物件。
40414	必要的命令欄位遺失或驗證失敗。
40415	命令包含無法辨識的欄位。
40422	BSON 陣列欄位名稱的值無效。
40485	提供了無法辨識的時區識別符。
40489	$dateFromParts 不允許混合一般行事曆欄位與 ISO 週日期欄位
40515	$dateFromParts 引數不會評估為整數。
40516	$dateFromParts 需要 'year' 或 'isoWeekYear' 存在
40517	'timezone' 選項必須評估為日期運算子的字串。
40518	$dateFromParts 或 $dateToParts 有無法辨識或輸入錯誤的參數。
40519	$dateFromParts 僅支援物件做為其引數。
40520	$dateToParts 獲得無法辨識的參數。
40522	$dateToParts 需要具有 'date' 參數的物件引數。

錯誤碼	說明
40523	$dateFromParts 'year' 必須是 1 到 9999 範圍內的整數。
40525	$dateFromParts ISO 週/天值超出範圍或缺少必要欄位。
40533	時區引數必須是字串。
40535	提供了無法辨識的選項給階段或表達式。
40536	指定陣列時，表達式只接受一個引數。
40539	缺少日期運算子所需的「日期」引數。
40540	$dateFromString 僅支援物件做為引數
40541	$dateFromString 收到無法辨識的引數。
40542	缺少 $dateString的 'dateString' 參數 dateFromString
40543	$dateFromString 要求 'dateString' 為字串。
40545	日期/時間字串不完整，缺少必要元素。
40551	您無法將時區資訊與時區引數一起傳入日期/時間字串
40553	$dateFromString 無法剖析指定的日期字串。
40554	您無法將 GMT 位移的日期/時間字串與時區引數一起傳入
40575	$changeStream 'fullDocument' 必須是 "default" 或 "updateLookup"。
40600	$out 無法在 $facet 階段內使用。
40601	$out 和 $merge 只能是管道中的最後一個階段。
40602	$changeStream 或 $geoNear 等階段不是管道中的第一個階段。
40603	$geoNear 不是最佳化後管道中的第一個階段。是否已停用或禁止最佳化？
40647	變更串流繼續字符具有遺失或錯誤類型的 _data 欄位。

錯誤碼	說明
40674	僅允許一種類型的繼續選項，但找到多個。
50001	指定的集合不允許 autoIndexId：false。
50694	此運算子獲得不明的引數。
50695	此運算子需要 'input' 欄位。
50696	運算子僅支援 物件做為其引數。
50699	此表達式運算子要求其輸入為字串。
50700	'chars' 引數必須是字串。
50736	getMore 無法在工作階段中未建立的游標上於工作階段中執行。
50737	在工作階段建立的游標上執行 getMore 需要提供 lsid。
50752	$substrBytes 起始索引為負。
50808	$changeStream 階段預期文件做為其引數。
51024	定序「強度」值必須為 5 或更少。
51047	$lookup 子管道內不允許此運算子或階段。
51075	無法在 $regex 和 $options 中設定選項
51076	記憶體不足，無法為指定的清單數量建置向量索引。
51078	找不到索引名稱。
51079	不支援延遲統計資料
51091	規則表達式無效或格式錯誤。
51103	運算子預期具有具名引數的物件，但收到另一種類型。
51104	運算子要求其「輸入」引數為字串。

錯誤碼	說明
51105	運算子的 'regex' 引數不是字串或 regex 類型。
51106	運算子需要其「選項」參數為字串。
51107	$regex 選項無法在 'regex' 和 'options' 欄位中指定。
51108	規則表達式選項包含無效的旗標或內嵌的 null 位元組。
51109	規則表達式不能包含內嵌的 null 位元組。
51110	規則表達式選項不能包含內嵌的 null 位元組。
51111	提供給此運算子的規則表達式無效。
51151	儲存輸出的緩衝區大小超過資料類型限制
51156	規則表達式無法在評估期間於 $regex 中執行。
51270	空的子投影物件不是有效的投影值。
51272	語法錯誤：$projection 至少需要一個輸出欄位。
51744	運算子需要 'replacement' 做為字串。
51745	運算子要求其「尋找」引數為字串。
51746	運算子的「輸入」引數不是字串。
51747	運算子需要 'replacement' 參數。
51748	此運算子需要 'find' 參數。
51749	此運算子需要 'input' 參數。
51750	彙總運算子收到不明的引數。
100003	處理最佳化查詢路徑時發生內部系統錯誤
100005	已超過設定的限制，例如集合、自訂角色或權限。

錯誤碼	說明
100006	已超過允許的使用者數量上限。
3040501	$rand 目前不接受引數
3041701	$getField 收到 $project 中的未知引數。
3041702	無效的 $project ： ： ，原因為 ： ： $getField 需要指定 'field'
3041703	無效的 $project ： ： ，原因為 ： ： $getField 需要指定 'input'
3041704	$getField 需要 'field' 引數才能評估為字串。
4161100	$setField 和 $unsetField 僅支援物件做為其引數。
4161101	$setField 或 $unsetField 獲得不明引數。
4161102	$setField 和 $unsetField 需要指定 'field'。
4161103	無效的 $project ： ： ，原因為 ： ： $setField 需要指定 'value'
4161105	$setField 或 $unsetField 需要 'input' 才能評估物件。
4161106	$setField 和 $unsetField 需要 'field' 評估為常數。
4161107	$setField 或 $unsetField 需要 'field' 來評估字串。
4161108	$project 不允許此處的欄位路徑參考；請改用 {$literal： ...}。
4161109	$setField 和 $unsetField 需要指定 'input' 欄位。
4662500	命令包含無法辨識的欄位。
4862100	完整命名空間超過允許的最大長度。
5166301	$dateDiff 僅支援物件做為其引數
5166302	$dateDiff 獲得無法辨識的引數。
5166303	缺少 $dateDiff 的 'startDate' 參數

錯誤碼	說明
5166304	缺少 $dateDiff 的 'endDate' 參數
5166305	缺少 $dateDiff 的 'unit' 參數
5166307	$dateDiff 引數不是日期。
5166308	$dateDiff 在計算日期差異時溢出。
5166400	$dateAdd 和 $dateSubtract 要求其引數為 物件。
5166401	$dateAdd 或 $dateSubtract 取得無法辨識的引數。
5166402	$dateAdd 或 $dateSubtract 需要 startDate、單位和金額才能存在。
5166403	$dateAdd 需要 startDate 才能轉換為日期
5166404	$dateAdd 預期定義時間單位的字串
5166405	$dateAdd 預期整數時間單位數量
5166406	dateAdd 溢位
5166503	提供了未知的時間單位值。
5439002	$dateTrunc 在計算截斷日期時溢出。
5439006	dateTrunc 不支援的 binSize 值
5439007	$dateTrunc 僅支援物件做為其引數
5439008	無法辨識的 $dateTrunc 引數；預期日期、單位、binSize、時區、startOfWeek。
5439009	缺少 $dateTrunc 的 'date' 參數
5439010	缺少 $dateTrunc 的 'unit' 參數
5439012	$dateTrunc 要求 'date' 為日期。
5439013	$dateDiff 和 $dateTrunc 需要 'unit' 作為字串。

錯誤碼	說明
5439014	$dateDiff 和 $dateTrunc 'unit' 值不是可辨識的時間單位。
5439015	$dateDiff 或 $dateTrunc 需要 'startOfWeek' 做為字串。
5439016	$dateDiff 或 $dateTrunc 'startOfWeek' 值不是一週中的有效一天。
5439017	$dateTrunc 要求 'binSize' 為 64 位元整數。
5439018	$dateTrunc 需要 'binSize' 大於 0。
5447000	$collStats 階段必須是巢狀物件。
5626500	在這種情況下，不允許 $geoNear、$near 和 $nearSphere，因為這些運算子需要排序地理空間資料。如果您不需要排序，請考慮改用 $geoWithin。
5654600	在此內容中，$project 不允許欄位路徑參考。
5654601	無效的 $project ： ： 由 造成 ： ： $getField 需要 'field' 評估為常數，但取得非常數引數
5654602	$getField 需要 'field' 才能評估字串。
5733415	不能 $mod x 零
5860400	$geoNear 需要 'near' 引數
5887502	$setEquals 的所有運算元都必須是陣列。
5911200	彙總 ： ： 期間的 PlanExecutor 錯誤，原因為 ： ： $mergeObjects 僅支援物件
5976500	$dateAdd 獲得無效的 'amount' 參數值。
6045000	$dateSubtract 獲得無效的 'amount' 參數值。
7158303	$switch 找不到符合輸入的分支，且未指定預設值。

版本備註

這些版本備註說明由主要引擎版本整理的 Amazon DocumentDB 功能、改善和錯誤修正。在每個版本區段中，項目會以資料表格式依日期列出。

如需如何編號引擎版本和修補程式版本的詳細資訊（包括引擎修補程式版本字首和 Amazon DocumentDB 引擎版本之間的映射），請參閱 引擎版本編號。

您可以執行下列命令來判斷目前的 Amazon DocumentDB 引擎修補程式版本：

```
db.runCommand({getEngineVersion: 1})
```

如果您的叢集不在最新版本的引擎上，您可能有待定的維護，其將升級您的引擎。如需詳細資訊，請參閱《開發人員指南》維護 Amazon DocumentDB中的 。

主題

- Amazon DocumentDB 引擎版本備註
- 一般 Amazon DocumentDB 版本備註
- Amazon DocumentDB 彈性叢集版本備註

Amazon DocumentDB 引擎版本備註

依主要版本整理的 Amazon DocumentDB 引擎版本的版本備註。

Amazon DocumentDB 8.0

Amazon DocumentDB 引擎 8.0 版的版本備註 (MongoDB 8.0 相容性）。

Amazon DocumentDB 8.0.1

日期	版本備註
2026 年 8 月 31 日	新功能 Amazon DocumentDB 現在支援直接從引擎版本 3.6 和 4.0 就地主要版本升級 (MVU) 到 8.0 版。這消除了中繼版本升級的需求，並可讓您升級 3.6 或 4.0 叢集，同時保留現有的資料、組態和叢集設定。Amazon

日期	版本備註
	DocumentDB 3.6 和 4.0 到 8.0 的主要版本升級適用於目前支援這些版本的所有 AWS 區域。如需詳細資訊，請參閱Amazon DocumentDB 就地主要版本升級。
2026 年 8 月 18 日	安全性和穩定性改善 Amazon DocumentDB 8.0.1 （引擎版本：4.0.9938) 【進行中】 此引擎修補程式提供下列改善： • 改善的請求處理 Amazon DocumentDB 現在可更徹底地驗證連線和訊息剖析路徑的傳入用戶端請求。這可改善更廣泛輸入模式的彈性和穩定性。 • 身分驗證交握強化 Amazon DocumentDB 現在可更徹底地驗證身分驗證交換。這可改善處理非預期或格式不正確的輸入時的穩定性。 Amazon DocumentDB 會在您下一個維護時段自動套用此修補程式。
2026 年 8 月 11 日	新功能 Amazon DocumentDB 8.0.1 （引擎版本：4.0.8864) • 已啟用 SCRAM-SHA-256 身分驗證的支援。 • 為 Amazon DocumentDB 全域叢集啟用可調校的 RPO。 • 改善背景垃圾收集期間的儲存回收，以減少一段時間內未使用的儲存空間。
2026 年 7 月 22 日	新功能 執行引擎版本 8.0 和更新版本的 Amazon DocumentDB 叢集現在支援 256 TiB 的最大儲存限制，將先前限制的 128 TiB 加倍。如需詳細資訊，請參閱Amazon DocumentDB 配額。

日期	版本備註
2026 年 7 月 13 日	Amazon DocumentDB 8.0.1 （引擎版本：4.0.8864) 新功能和改進 1. 在預設規劃器 (v3) 中支援 13 個累積器運算子：$top、$bottom、$topN、$bottomN$count、$firstN、$lastN、le 、$stdDevPop 、、。 $stdDevSamp 2. 在預設規劃器 (v3) 中支援 36 個表達式運算子：$median、$percentile 、$stdDevPop $stdDevSamp $round、、$sigmoid$trunc$firstN、、$lastN$maxN、$minN$bze 、$bsonSize 、、$tsIncrement $tsSecond 、$cos、$acos、$cosh、、$acosh$sin、、$asin$sinh$asinh$taroRadians 、、、、、、、$radiansToDegrees 、、、、、、、$isNumber 、、、、、、、

Amazon DocumentDB 8.0.0

日期	版本備註
2026 年 8 月 18 日	安全性和穩定性改善 Amazon DocumentDB 8.0.0 （ 引擎版本：4.0.9912) 【進行中】 此引擎修補程式提供下列改善： • 改善的請求處理 Amazon DocumentDB 現在可更徹底地驗證連線和訊息剖析路徑的傳入用戶端請求。這可改善更廣泛輸入模式的彈性和穩定性。 • 身分驗證交握強化 Amazon DocumentDB 現在可更徹底地驗證身分驗證交換。這可改善處理非預期或格式不正確的輸入時的穩定性。 如果您未更早套用，Amazon DocumentDB 會在發行 30 天後的下一個維護時段自動套用此修補程式。
2026 年 7 月 22 日	新功能 執行引擎版本 8.0 和更新版本的 Amazon DocumentDB 叢集現在支援 256 TiB 的最大儲存限制，將先前限制的 128 TiB 加倍。如需詳細資訊，請參閱Amazon DocumentDB 配額。
2026 年 5 月 13 日	新功能 Amazon DocumentDB Serverless 現已可用於 Amazon DocumentDB 8.0。如需詳細資訊，請參閱使用 Amazon DocumentDB 無伺服器。
2026 年 5 月 6 日	錯誤修正和其他變更 Amazon DocumentDB 8.0 （ 引擎版本：4.0.6940) 此更新提供具有下列改進功能的選用引擎修補程式： 1. 修正垃圾回收中的錯誤，在邊緣案例中，可能會導致讀取器當機。
2026 年 4 月 20 日	新功能

日期	版本備註
	現已提供從 Amazon DocumentDB 5.0 到 Amazon DocumentDB 8.0 的就地主要版本升級 (MVU)。如需詳細資訊，請參閱Amazon DocumentDB 就地主要版本升級。

日期	版本備註
2026 年 4 月 20 日	新功能和改進 Amazon DocumentDB 8.0 （引擎版本：4.0.5995、4.0.6473) 這是必要的引擎修補程式，其中包含下列新功能、改善和修正： 1. $sortArray 中的運算子$project： • 支援規劃$sortArray 器第 3 版彙總管道中的$project階段運算子。$sortArray 運算子會根據陣列的元素（純量值）或指定欄位（文件陣列）來排序陣列，並以遞增 (1) 或遞減 (-1) 順序傳回排序的陣列。 2. 支援 $out、$collStats 、$indexStats 、$geoNear使用規劃器 v3。使用規劃器 V3，$out階段也支援寫入不同資料庫的集合。 3. 即使$unwind階段包含 includeArrayIndex 參數，彙總管道現在也會聯結 $lookup和 $unwind階段。先前，includeArrayIndex 當 存在時，會略過此最佳化。 4. 效能增強可改善在 I/O 最佳化組態上執行之 db.r8g.8xlarge 和較大執行個體的寫入輸送量。 5. 您現在可以監控執行個體的 Connections (active) 指標。如需詳細資訊，請參閱執行個體配額。 6. 支援 $dateFromParts $dateToParts 和 彙總運算子。 7. 命令層級效能指標現在可在 CloudWatch 中用於 CRUD 操作 (find、insert、findAndModify 、update、deleteaggregate count、distinct、getMore、abortTransaction 、commitTransaction 、)AvgDuration 、公開 P100Duration 、 和 MaxConcurrent 維度，以改善操作監控。 8. 已修正在輸出文件中缺少包含 true/false 布林值或 0/1 數值newRoot的欄位$replaceRoot 的行為錯誤。 9. 修正 中的錯誤$indexOfArray ，其中有效的查詢傳回錯誤，指出它們無效。

日期	版本備註
	10.解釋計畫現在在executionStats 輸出中包含docsExamined 指標。 11.改善 SASL 身分驗證效能。 12.最佳化文件更新，以減少索引值保持不變且文件符合現有頁面時的儲存開銷。 13.支援 bypassDocumentValidation 中的 $merge。 所有 Amazon DocumentDB 8.0 叢集都會在下一個維護時段自動升級至此修補程式版本。
2026 年 3 月 26 日	新功能 Amazon DocumentDB 現在支援最新一代的記憶體最佳化執行個體 db.r8g，採用 Arm 型 AWS Graviton4 處理器，比 R6G 執行個體提供高達 30% 的效能。如需詳細資訊，請參閱本指南管理執行個體類別中的

日期	版本備註
2025 年 11 月 11 日	新功能 Amazon DocumentDB 現在提供與 MongoDB 8.0 的完整線路通訊協定相容性。Amazon DocumentDB 8.0 改善了查詢效能，可讓您以較低的成本建置高效能應用程式。 Amazon DocumentDB 8.0（引擎修補程式 4.0.4448 版） • 新增對 MongoDB 8.0 API 驅動程式的支援。 • 新增對規劃器Version3 的支援，可將效能改進延伸至彙總階段運算子，以及支援彙總管道最佳化和不同的命令。 • 新增透過 Zstandard 壓縮演算法對字典型壓縮的支援，將壓縮率提高 5 倍。 • 新增新功能：Amazon DocumentDB 8.0 支援定序和檢視。 • 新增對新彙總階段和運算子的支援：$replaceWith、$vectorSearch、$merge、$set、$unset、$bucket 和 3 個新彙總運算子 $pow、$rand、$dateTrunc。 • 新增對新版本文字索引的支援：Amazon DocumentDB 8.0 中的文字索引 v2 引入了額外的字符，增強了文字搜尋功能。 • 包括透過平行向量索引建置的向量搜尋改進，減少高達 30 倍的索引建置時間。 • 已移除對 TLS 1.0 和 TLS 1.1 的支援。最小 TLS 版本現在為 TLS 1.2。

Amazon DocumentDB 5.0

Amazon DocumentDB 引擎 5.0 版的版本備註 (MongoDB 5.0 相容性）。

Amazon DocumentDB 5.0.1

日期	版本備註
2026 年 8 月 18 日	安全性和穩定性改善

日期	版本備註
	Amazon DocumentDB 5.0.1 （引擎版本：3.0.22319) 【進行中】 此引擎修補程式提供下列改善： • 改善的請求處理 Amazon DocumentDB 現在可更徹底地驗證連線和訊息剖析路徑的傳入用戶端請求。這可改善更廣泛輸入模式的彈性和穩定性。 • 身分驗證交握強化 Amazon DocumentDB 現在可更徹底地驗證身分驗證交換。這可改善處理非預期或格式不正確的輸入時的穩定性。 如果您未更早套用，Amazon DocumentDB 會在發行 30 天後的下一個維護時段自動套用此修補程式。
2026 年 8 月 11 日	新功能 Amazon DocumentDB 5.0.1 （引擎版本：3.0.21138) 已啟用 SCRAM-SHA-256 身分驗證的支援。

日期	版本備註
2026 年 6 月 8 日	新的次要版本 Amazon DocumentDB 5.0.1 （引擎版本 3.0.21138) Amazon DocumentDB 5.0.1 是 Amazon DocumentDB 5.0 主要版本的第一個次要版本版本。次要版本包括改善功能，例如新功能、安全性修正和錯誤修正。您可以使用次要版本升級，從 5.0.0 升級到 5.0.1。如需詳細資訊，請參閱Amazon DocumentDB 次要版本升級。 此次要版本包含下列新功能和改進： 1. 在 `storageSegmentBase` 和 `storageSegmentExtended` 中新增 `gcRuntimeStats` 、 `documentFragmentStats` 和 集合層級指標的分解`collstats` ，以將更多資訊呈現到垃圾收集程序的整體運作狀態。如需詳細資訊，請參閱 Amazon DocumentDB 中的垃圾回收。 2. 支援 `$pow` `$rand`和 彙總運算子。 3. 您現在可以監控執行個體的 `Connections (active)` 指標。如需詳細資訊，請參閱執行個體配額。 4. 透過消除使用全域可存活唯讀重新啟動的次要叢集，改善全域叢集跨區域彈性。 5. 支援 `$dateFromParts` `$dateToParts` 和 彙總運算子。 6. 命令層級效能指標現在可在 CloudWatch 中用於 CRUD 操作 (`find`、`insert`、`findAndModify` 、`update`、`deleteaggregate` `count`、`distinct`、`getMore`、`abortTransaction` 、`commitTransaction` 、)`AvgDuration` 、公開 `P100Duration` 、 和 `MaxConcurrent` 維度，以改善操作監控。 7. `$replaceRoot` 會輸出包含布林值 (`true`/`false`) 和數值 (`0`/) `newRoot`的欄位1。 8. `$indexOfArray` 支援更廣泛的有效查詢表達式範圍。 9. 解釋計畫在`executionStats` 輸出中包含`docsExamined` 指標。 10改善 SASL 身分驗證效能。

Amazon DocumentDB 5.0.0

日期	版本備註
2026 年 8 月 18 日	安全性和穩定性改善 Amazon DocumentDB 5.0.0 （ 引擎版本：3.0.22303) 【進行中】 此引擎修補程式提供下列改善： • 改善的請求處理 Amazon DocumentDB 現在可更徹底地驗證連線和訊息剖析路徑的傳入用戶端請求。這可改善更廣泛輸入模式的彈性和穩定性。 • 身分驗證交握強化 Amazon DocumentDB 現在可更徹底地驗證身分驗證交換。這可改善處理非預期或格式不正確的輸入時的穩定性。 • 存取控制強制執行 Amazon DocumentDB 現在會在巢狀查詢操作的所有巢狀層級一致地驗證必要的權限。 如果您未更早套用，Amazon DocumentDB 會在發行 30 天後的下一個維護時段自動套用此修補程式。
2026 年 5 月 6 日	錯誤修正和其他變更 Amazon DocumentDB 5.0 （ 引擎版本：3.0.20421) 此版本適用於必要的安全修補程式。
2026 年 4 月 20 日	新功能 現已提供從 Amazon DocumentDB 5.0 到 Amazon DocumentDB 8.0 的就地主要版本升級 (MVU)。如需詳細資訊，請參閱Amazon DocumentDB 就地主要版本升級。
2026 年 4 月 13 日	錯誤修正和其他變更 Amazon DocumentDB 5.0 （ 引擎版本：3.0.19917) 這是必要的引擎修補程式，其中包含下列新功能、改善和修正：

日期	版本備註
	1. `DatabaseConnectionsActiveMax` 指標現在`Connections (active)`會追蹤執行個體。如需詳細資訊，請參閱執行個體配額。 2. 改善 SASL 身分驗證效能。 所有 Amazon DocumentDB 5.0 叢集都會在下一個維護時段自動升級至此修補程式版本。
2025 年 10 月 22 日	新功能 Amazon DocumentDB 5.0 （引擎修補程式 3.0.17983 版） Amazon DocumentDB 現在支援最新一代的記憶體最佳化執行個體 db.r8g，採用 Arm 型 AWS Graviton4 處理器，比 R6G 執行個體提供高達 30% 的效能。如需詳細資訊，請參閱本指南管理執行個體類別中的和此部落格文章。 新功能 Amazon DocumentDB 5.0 （引擎修補程式 3.0.15902 版） • 新增對較長集合名稱的支援（命名空間長度上限為 255 個字元）。 • 引進新的查詢規劃器（規劃器 2.0 版）。如需詳細資訊，請參閱本指南中的 查詢規劃器 v2。 • 在 中`collstats` 新增了「MVCCIDStats」和「gcRuntimeStats」的新欄位，提供關於垃圾收集和帳齡收集的集合整體運作狀態資訊。
2025 年 10 月 16 日	新功能 Amazon DocumentDB 5.0 （引擎修補程式 3.0.15902 版） 當您的叢集位於虛擬私有雲端 (VPC) 時，Amazon DocumentDB 現在支援雙堆疊模式 (IPv4/IPv6)。 如需詳細資訊，請參閱此部落格文章和《Amazon DocumentDB 開發人員指南Amazon VPC 和 Amazon DocumentDB》中的 。

日期	版本備註
2025 年 9 月 15 日	錯誤修正和其他變更 Amazon DocumentDB 5.0 （引擎修補程式 3.0.17983 版） 修正查詢包含後續 `$replaceRoot` 和 `$lookup`階段時的引擎當機問題。
2025 年 7 月 29 日	新功能 Amazon DocumentDB 5.0 （引擎修補程式 3.0.15902 版） • 新增對`$regexFindAll` 彙總運算子的支援。 • 啟動索引膨脹和集合型膨脹指標 • 新增 CloudWatch 指標`AvailableMVCCIds` `LongestActiveGCRuntime` ，並將有關垃圾收集和帳齡收集的資訊提供給叢集的整體運作狀態。 • `serverStatus` 命令現在包含 "nvme_cache" 欄位，顯示 NVMe 後端執行個體上寫入和未寫入 NVMe NVMe 快取的頁面計數。 • 已移除 `$group`和 的欄位數量限制 `$project` • 支援下列字串運算子：`$trim`、`$rtrim`、`$ltrim`、`$replaceOne` 、 `$regexFindAll` 和 `$replaceAll` • 支援下列陣列運算子： `$first`和 `$last` • 支援下列階段運算子： `$collStats` 錯誤修正和其他變更 Amazon DocumentDB 5.0 （引擎修補程式 3.0.15902 版） 修正索引建立期間影響大型集合的問題。

日期	版本備註
2025 年 7 月 28 日	新功能 Amazon DocumentDB 5.0（引擎修補程式 3.0.14228 版） Amazon DocumentDB 執行個體型叢集現在支援 Amazon DocumentDB Serverless，這是一種隨需的自動擴展組態。 如需詳細資訊，請參閱 使用 Amazon DocumentDB 無伺服器和這是新的部落格文章。 https://aws.amazon.com/about-aws/whats-new/2025/07/amazon-documentdb-serverless/
2025 年 3 月 24 日	新功能 Amazon DocumentDB 5.0（引擎修補程式 3.0.11800 版） 在變更串流`postBatchResumeToken`　中新增對 的支援。 如需詳細資訊，請參閱使用 繼續變更串流 `postBatchResumeToken` 。
2025 年 1 月 15 日	新功能 Amazon DocumentDB 5.0（引擎修補程式 3.0.11800 版） Amazon DocumentDB 現在會透過寫入器執行個體重新啟動來維持讀取可用性。讀取器執行個體現在會在寫入器執行個體重新啟動期間繼續提供讀取請求。 錯誤修正和其他變更 Amazon DocumentDB 5.0（引擎修補程式 3.0.11800 版） • 修正處理大量插入特殊情況的`killOp`命令。 • 改善 Amazon DocumentDB 執行個體上的網路 I/O 相關記憶體使用量。 • 已修正`$text`篩選條件查詢的`count`命令。

日期	版本備註
2024 年 12 月 18 日	新功能 Amazon DocumentDB 5.0 （引擎修補程式 3.0.5721 版） Amazon DocumentDB 執行個體型叢集現在支援 NVMe 支援的 R6GD 執行個體。 如需詳細資訊，請參閱 NVMe 支援的執行個體和這是新的部落格文章。https://aws.amazon.com/about-aws/whats-new/2024/12/nvme-backed-r6gd-instances-amazon-documentdb-mongodb-compatibility/
2024 年 11 月 6 日	新功能 Amazon DocumentDB 5.0 （引擎修補程式 3.0.11051 版） • 新增對最低 TLS 版本`tls1.2+`和`tls1.3+``tls`叢集參數的支援。 • 已啟用欄位名稱中字元 USD($) 和 dot(.) 的支援。如需功能差異，請參閱 欄位名稱中的 Dollar($) 和 dot(.)。

日期	版本備註
2024 年 9 月 17 日	新功能 Amazon DocumentDB 5.0 （引擎修補程式 3.0.10696 版） Amazon DocumentDB 現在支援在 `reIndex`中使用 重建索引`runCommand` 。如需詳細資訊，請參閱使用 的索引維護 `reIndex`。 Note `reIndex` 僅支援 Amazon DocumentDB 5.0 （引擎修補程式 3.0.10696 版及更新版本）。 錯誤修正和其他變更 Amazon DocumentDB 5.0 （引擎修補程式 3.0.10696 版） • `$setOnInsert` 現在在因更新操作而產生的插入期間支援 `_id` 欄位。 • 已修正主要版本升級至 Amazon DocumentDB 5.0 後，無法回收儲存空間的問題。
2024 年 8 月 22 日	新功能 Amazon DocumentDB 5.0 （所有引擎修補程式版本） Amazon DocumentDB 全域叢集現在支援叢集切換和受管叢集容錯移轉。如需詳細資訊，請參閱執行 Amazon DocumentDB 全域叢集的切換及執行 Amazon DocumentDB 全域叢集的受管容錯移轉。 Note 僅 Amazon DocumentDB 4.0 和 5.0 支援全域叢集切換和容錯移轉。

<table>
<tr><th>日期</th><th>版本備註</th></tr>
<tr><td>2024 年 7 月 23 日</td><td>新功能

Amazon DocumentDB 5.0（引擎修補程式 3.0.8126 版）

• 新增對較長索引名稱的支援（最多 255 個字元）。如需詳細資訊，請參閱命名限制條件
• 將最大連線限制增加兩倍：

<table>
<tr><th>執行個體類型</th><th>原始限制</th><th>新的限制</th></tr>
<tr><td>t3.medium</td><td>500</td><td>1000</td></tr>
<tr><td>t4g.medium</td><td>500</td><td>1000</td></tr>
<tr><td>r5.large</td><td>1700</td><td>3400</td></tr>
<tr><td>r5.xlarge</td><td>3500</td><td>7000</td></tr>
<tr><td>r5.2xlarge</td><td>7100</td><td>14200</td></tr>
<tr><td>r5.4xlarge</td><td>14200</td><td>28400</td></tr>
<tr><td>r5.8xlarge</td><td>28400</td><td>60000</td></tr>
<tr><td>r5.12xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r5.16xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r5.24xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r6g.large</td><td>1700</td><td>3400</td></tr>
<tr><td>r6g.xlarge</td><td>3500</td><td>7000</td></tr>
<tr><td>r6g.2xlarge</td><td>7100</td><td>14200</td></tr>
<tr><td>r6g.4xlarge</td><td>14200</td><td>28400</td></tr>
<tr><td>r6g.8xlarge</td><td>28400</td><td>60000</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>日期</th><th>版本備註</th></tr>
<tr><td></td><td>
<table>
<tr><th>執行個體類型</th><th>原始限制</th><th>新的限制</th></tr>
<tr><td>r6g.12xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r6g.16xlarge</td><td>30000</td><td>60000</td></tr>
</table>
如需詳細資訊，請參閱執行個體配額。

錯誤修正和其他變更

Amazon DocumentDB 5.0（引擎修補程式 3.0.8126 版）

增強邏輯以同步和讀取器上的<code>CurrentTime ResumeToken</code> 變更串流。
</td></tr>
<tr><td>2024 年 7 月 22 日</td><td>
新功能

Amazon DocumentDB 5.0（引擎修補程式 3.0.6742 版）

• 新增支援 DML 稽核篩選。您現在可以根據 DML 稽核日誌的特定需求來設定篩選條件，而不是記錄每個 DML 查詢。如需詳細資訊，請參閱篩選 DML 稽核事件。
• 新增對下列文件壓縮的支援：
  • 設定最小壓縮閾值
  • 啟用現有集合的壓縮（適用於新文件）
  • 允許叢集層級的預設壓縮設定

  如需詳細資訊，請參閱管理集合層級文件壓縮。
• 新增在讀取器執行個體上使用變更串流的支援。如需詳細資訊，請參閱在次要執行個體上使用變更串流。
</td></tr>
</table>

日期	版本備註
2024 年 7 月 9 日	新功能 對於 Amazon DocumentDB 5.0（最低引擎修補程式版本 3.0.4780），TLS CA 憑證更新不再需要系統重新啟動。如需詳細資訊，請參閱更新您的 Amazon DocumentDB TLS 憑證。
2024 年 5 月 29 日	新功能 Amazon DocumentDB 5.0（引擎修補程式 3.0.6742 版） • 新增對 `$regexMatch` 和 `$regexFind` 運算子的支援。 • 新增了在解決大型整數時確保稽核日誌中完整精確度的支援。稽核日誌現在會維持所有數字的確切數值表示，防止任何精確度損失。
2024 年 4 月 3 日	新功能 Amazon DocumentDB 5.0（引擎修補程式 3.0.5721 版） • 新增對 `bypassDocumentValidation` 和 精細錯誤訊息的支援 `$jsonSchema` 。如需 `bypassDocumentValidation` 的相關資訊，請參閱 bypassDocumentValidation。 • 新增對 的支援`$expr`。 • 新增對 中不相關聯結的支援`$lookup`。 • 新增在`$out`彙總階段中保留驗證規則的支援。

日期	版本備註
2024 年 1 月 10 日	新功能 Amazon DocumentDB 5.0 （ 引擎修補程式版本 3.0.4574、3.0.4780、3.0.4960) • 新增對 HNSW 向量索引的支援。如需詳細資訊，請參閱Amazon DocumentDB 的向量搜尋。 • 新增對部分索引的支援。如需詳細資訊，請參閱部分索引。 • 在currentOp 命令中的集合上新增對 GC 執行時間的支援。 • 新增支援 Amazon DocumentDB 上原生文字搜尋的文字索引。如需詳細資訊，請參閱使用 Amazon DocumentDB 執行文字搜尋。 • 新增對$jsonSchema 結構描述關鍵字 type、allOf、、oneOfanyOfnot、、maxItems、minItems、maxProperties 、minProperties 、、patternpatternProperties 、、multipleOf 、 dependencies 和 的支援uniqueItems 。 如需更多資訊，請參閱使用 JSON 結構描述驗證。 • 新增對算術運算子 $ceil、$floor、$ln、$log、$log10、$sqrt和 的支援$exp。 如需更多資訊，請參閱算術運算子。 • 新增對條件式表達式運算子 的支援$switch。 • 新增對平行IVFFLAT向量索引建置的支援。文件已更新，方法是從開發人員指南中移除平行IVFFLAT向量索引建置限制。 錯誤修正和其他變更 • 新增用於叫用 的反區分大小寫功能db.runCommand("dbstats") 。3.0.4960 或 2.0.10221 之前引擎修補程式版本的 Amazon DocumentDB 5.0 客戶應套用這些最新的引擎修補程式。 • 修正在 mongo shell 1.7 版和更新版本db.coll.stats() 上調用時出現的錯誤。文件已更新，方法是從開發人員指南中移除 mongo shell db.coll.stats() 疑難排解提示。

日期	版本備註
2023 年 11 月 29 日	新功能 Amazon DocumentDB 5.0（引擎修補程式 3.0.3727 版） 新增向量搜尋的支援。如需詳細資訊，請參閱此部落格文章，並參閱《Amazon DocumentDB 開發人員指南Amazon DocumentDB 的向量搜尋》中的 。
2023 年 11 月 21 日	新功能 Amazon DocumentDB 5.0（引擎修補程式 3.0.3727 版） 新增對 I/O 最佳化儲存的支援。如需詳細資訊，請參閱《Amazon DocumentDB 開發人員指南Amazon DocumentDB 叢集儲存組態》中的 。 新增與 SageMaker Canvas 的無程式碼機器學習整合。如需詳細資訊，請參閱《Amazon DocumentDB 開發人員指南使用 Amazon SageMaker AI Canvas 進行無程式碼機器學習》中的 。

日期	版本備註
2023 年 11 月 6 日	Amazon DocumentDB 5.0 （引擎修補程式版本 3.0.3727) 新功能 • 新增對$jsonSchema 結構描述關鍵字 maxLength 、minLength 、、maximumminimum、exclusiveMaximum 、exclusive Minimum 、 items和 的支援additionalItems 。 請注意，僅執行個體型叢集支援 JSON 結構描述驗證。 • 新增對$convert彙總管道運算子及其短期衍生運算子 $toBool、$toInt、$toLong、$toDouble 、$toDecimal 、 $toObjectId 和 $toString 的支援$toDate。 • 新增對設定表達式運算子 $setDifference 、 $anyEleme ntTrue 和 的支援$allElementTrue 。 錯誤修正和其他變更 已修正NaN未顯示從 -NaN到 的變更串流更新的問題。
2023 年 9 月 15 日	新功能 Amazon DocumentDB 5.0 （引擎修補程式 3.0.3140 版） 僅在執行個體型叢集中新增對 $jsonSchema 結構描述驗證器的支援。如需更多資訊，請參閱使用 JSON 結構描述驗證。

日期	版本備註
2023 年 7 月 13 日	新功能 Amazon DocumentDB 5.0（引擎修補程式 3.0.1948 版） • 新增對文件壓縮的支援。 • 新增對平行索引建置的支援。 • 新增對索引建置狀態的支援。 錯誤修正和其他變更 Amazon DocumentDB 5.0（引擎修補程式 3.0.1948 版） • 已修正當使用者無法存取系統集合時，Amazon DocumentDB 彈性叢集`createCollection` 的 身分驗證問題。 • 已修正次要區域執行個體無法使用相同主要區域執行個體名稱的問題。
2023 年 6 月 7 日	錯誤修正和其他變更 Amazon DocumentDB 5.0 • Amazon DocumentDB 5.0 現在支援 r5 和 t3.medium 執行個體。 • `engineVersion` 選項預設為在 AWS SDK `5.0.0`中 AWS CLI，以及 CloudFormation。
2023 年 5 月 10 日	錯誤修正和其他變更 Amazon DocumentDB 5.0（引擎修補程式 3.0.1361 版） • 在 createIndex命令`ignoreunknownindexoptions` 中新增對 的支援。 • 已停止將內部監控查詢新增至稽核日誌。 • 使用者定義的本機變數名稱現在支援投影運算子的 “_”（底線），例如 `$let`和 `$filter`。

日期	版本備註

2023 年 3 月 1 日	新功能
	
	Amazon DocumentDB 5.0 （引擎修補程式 3.0.775 版）
	
	• 推出 Amazon DocumentDB 5.0
	  • MongoDB 5.0 相容性（支援 MongoDB 5.0 API 驅動程式）
	  • 支援用戶端欄位層級加密 (FLE)。您現在可以先加密用戶端的欄位，再將資料寫入 Amazon DocumentDB 叢集。如需詳細資訊，請參閱用戶端欄位層級加密
	  • 新的彙總運算子：`$dateAdd`、 `$dateSubtract`
	• 針對所有執行個體型 Amazon DocumentDB 叢集和碎片型彈性叢集，將儲存限制提高至 128 TiB。
	• Amazon DocumentDB 5.0 現在支援在第一個巢狀層級使用`$elemMatch` 運算子進行索引掃描。當查詢只有一個層級的`$elemMatch` 篩選條件，且巢狀`$elemMatch` 查詢不支援索引掃描時，支援索引掃描。
	
	支援索引掃描的查詢形狀：
	
	`db.foo.find( { "a": {$elemMatch: { "b": "xyz", "c": "abc"} } })`
	
	不支援索引掃描的查詢形狀：
	
	`db.foo.find( { "a": {$elemMatch: { "b": {$elemMatch: { "d": "xyz", "e": "abc"} }} } })`

Amazon DocumentDB 4.0

Amazon DocumentDB 引擎 4.0 版的版本備註 (MongoDB 4.0 相容性）。

日期	版本備註
2026 年 8 月 31 日	新功能

日期	版本備註
	Amazon DocumentDB 現在支援直接從引擎 4.0 版到 8.0 版的就地主要版本升級 (MVU)。這消除了中繼版本升級的需求，並可讓您升級 4.0 叢集，同時保留現有的資料、組態和叢集設定。此升級適用於目前支援這些版本的所有 AWS 區域。如需詳細資訊，請參閱Amazon DocumentDB 就地主要版本升級。
2026 年 8 月 18 日	安全性和穩定性改善 Amazon DocumentDB 4.0（引擎版本：2.0.14287）【進行中】 此引擎修補程式提供下列改善： • 改善的請求處理 Amazon DocumentDB 現在可更徹底地驗證連線和訊息剖析路徑的傳入用戶端請求。這可改善更廣泛輸入模式的彈性和穩定性。 • 身分驗證交握強化 Amazon DocumentDB 現在可更徹底地驗證身分驗證交換。這可改善處理非預期或格式不正確的輸入時的穩定性。 • 存取控制強制執行 Amazon DocumentDB 現在會在巢狀查詢操作的所有巢狀層級一致地驗證必要的權限。 如果您未更早套用，Amazon DocumentDB 會在發行 30 天後的下一個維護時段自動套用此修補程式。
2026 年 4 月 27 日	錯誤修正和其他變更 Amazon DocumentDB 4.0（引擎版本：2.0.14278） 這是必要的引擎修補程式，可解決下列問題： 1. 修正在極少數情況下，可能導致point-in-time(PITR) 和主要版本升級 (MVU) 變得沒有回應的問題。 所有 Amazon DocumentDB 4.0 叢集都會在下一個維護時段自動升級至此修補程式版本。

日期	版本備註
2026 年 3 月 9 日	錯誤修正和其他變更 Amazon DocumentDB 4.0 （引擎版本：2.0.14165) 此更新提供具有下列改進功能的選用引擎修補程式： 1. `DatabaseConnectionsActiveMax` 指標現在`Connections (active)`會追蹤執行個體。如需詳細資訊，請參閱執行個體配額。 2. 改善 SASL 身分驗證效能。
2025 年 10 月 22 日	新功能 Amazon DocumentDB 4.0 （引擎修補程式 2.0.11747 版） 在 中`collstats` 新增了「MVCCIDStats」和「gcRuntimeStats」的新欄位，提供關於垃圾收集和帳齡收集的集合整體運作狀態資訊。
2025 年 10 月 16 日	新功能 Amazon DocumentDB 4.0 （引擎修補程式 2.0.11747 版） 當您的叢集位於虛擬私有雲端 (VPC) 時，Amazon DocumentDB 現在支援雙堆疊模式 (IPv4/IPv6)。 如需詳細資訊，請參閱此部落格文章和《Amazon DocumentDB 開發人員指南Amazon VPC 和 Amazon DocumentDB》中的 。

日期	版本備註
2025 年 7 月 29 日	新功能 Amazon DocumentDB 4.0（引擎修補程式 2.0.11747 版） • 啟動索引膨脹和集合型膨脹指標 • 新增 CloudWatch 指標`LongestActiveGCRun time` ，`AvailableMVCCIds` 並將有關垃圾收集和帳齡收集的資訊提供給叢集的整體運作狀態。 • 支援下列字串運算子：`$rtrim`、 `$trim`和 `$ltrim` • 支援下列階段運算子： `$collStats` 錯誤修正和其他變更 Amazon DocumentDB 4.0（引擎修補程式 2.0.11747 版） 修正索引建立期間影響大型集合的問題。
2025 年 1 月 15 日	錯誤修正和其他變更 Amazon DocumentDB 4.0（引擎修補程式 2.0.11096 版） • 修正處理大量插入特殊情況的`killOp`命令。 • 改善 Amazon DocumentDB 執行個體上的網路 I/O 相關記憶體使用量。
2024 年 11 月 6 日	新功能 Amazon DocumentDB 4.0（引擎修補程式 2.0.10980 版） • 新增對最低 TLS 版本`tls1.2+`和`tls1.3+``tls`叢集參數的支援。

日期	版本備註
2024 年 9 月 17 日	錯誤修正和其他變更 Amazon DocumentDB 4.0（引擎修補程式 2.0.10898 版） • `$setOnInsert` 現在在因更新操作而產生的插入期間支援 `_id` 欄位。 • 已修正主要版本升級至 Amazon DocumentDB 5.0 後，無法回收儲存空間的問題。
2024 年 8 月 22 日	新功能 Amazon DocumentDB 4.0（引擎修補程式 2.0.5704 版） Amazon DocumentDB 全域叢集現在支援叢集切換和受管叢集容錯移轉。如需詳細資訊，請參閱執行 Amazon DocumentDB 全域叢集的切換及執行 Amazon DocumentDB 全域叢集的受管容錯移轉。 **Note** 僅 Amazon DocumentDB 4.0 和 5.0 支援全域叢集切換和容錯移轉。

<table>
<tr><th>日期</th><th>版本備註</th></tr>
<tr><td>2024 年 7 月 23 日</td><td>新功能

Amazon DocumentDB 4.0（引擎修補程式 2.0.10709 版）

• 新增對較長索引名稱（最多 255 個字元）的支援。如需詳細資訊，請參閱命名限制條件
• 將最大連線限制增加兩倍：

<table>
<tr><th>執行個體類型</th><th>原始限制</th><th>新的限制</th></tr>
<tr><td>t3.medium</td><td>500</td><td>1000</td></tr>
<tr><td>t4g.medium</td><td>500</td><td>1000</td></tr>
<tr><td>r5.large</td><td>1700</td><td>3400</td></tr>
<tr><td>r5.xlarge</td><td>3500</td><td>7000</td></tr>
<tr><td>r5.2xlarge</td><td>7100</td><td>14200</td></tr>
<tr><td>r5.4xlarge</td><td>14200</td><td>28400</td></tr>
<tr><td>r5.8xlarge</td><td>28400</td><td>60000</td></tr>
<tr><td>r5.12xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r5.16xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r5.24xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r6g.large</td><td>1700</td><td>3400</td></tr>
<tr><td>r6g.xlarge</td><td>3500</td><td>7000</td></tr>
<tr><td>r6g.2xlarge</td><td>7100</td><td>14200</td></tr>
<tr><td>r6g.4xlarge</td><td>14200</td><td>28400</td></tr>
<tr><td>r6g.8xlarge</td><td>28400</td><td>60000</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>日期</th><th>版本備註</th></tr>
<tr><td></td><td>
<table>
<tr><th>執行個體類型</th><th>原始限制</th><th>新的限制</th></tr>
<tr><td>r6g.12xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r6g.16xlarge</td><td>30000</td><td>60000</td></tr>
</table>
如需詳細資訊，請參閱執行個體配額。
</td></tr>
<tr><td>2024 年 7 月 22 日</td><td>新功能

Amazon DocumentDB 4.0（引擎修補程式 2.0.10593 版）

• 新增支援 DML 稽核篩選。您現在可以根據 DML 稽核日誌的特定需求來設定篩選條件，而不是記錄每個 DML 查詢。如需詳細資訊，請參閱篩選 DML 稽核事件。</td></tr>
<tr><td>2024 年 7 月 9 日</td><td>新功能

對於 Amazon DocumentDB 4.0（最低引擎修補程式版本 2.0.10179），TLS CA 憑證更新不再需要系統重新啟動。如需詳細資訊，請參閱更新您的 Amazon DocumentDB TLS 憑證。</td></tr>
<tr><td>2024 年 5 月 29 日</td><td>新功能

Amazon DocumentDB 4.0（引擎修補程式 2.0.10593 版）

• 新增了在解決大型整數時確保稽核日誌中完整精確度的支援。稽核日誌現在會維持所有數字的確切數值表示，防止任何精確度損失。</td></tr>
</table>

日期	版本備註
2024 年 4 月 3 日	新功能 Amazon DocumentDB 4.0（引擎修補程式 2.0.10392 版） • 新增對 bypassDocumentValidation 的支援$jsonSchema 。如需 bypassDocumentValidation 的相關資訊，請參閱 bypassDocumentValidation。 • 新增對 的支援$expr。 • 新增對 中不相關聯結的支援$lookup。 • 新增在$out彙總階段中保留驗證規則的支援。
2024 年 1 月 10 日	新功能 Amazon DocumentDB 4.0（引擎修補程式版本 2.0.10124、2.0.10179、2.0.10221) • 在currentOp 命令中的集合上新增對 GC 執行時間的支援。 • 新增對$jsonSchema 結構描述關鍵字 type、allOf、、oneOfanyOfnot、、maxItems、minItems、maxProperties 、minProperties 、、patternpatternProperties 、、multipleOf 、 dependencies 和 的支援uniqueItems 。 如需更多資訊，請參閱使用 JSON 結構描述驗證。 • 新增對算術運算子 $ceil、$floor、$ln、$log、$sqrt、$log10和 的支援$exp。 如需更多資訊，請參閱算術運算子。 • 新增對條件式表達式運算子 的支援$switch。
2023 年 12 月 20 日	其他變更 在 Amazon DocumentDB 4.0 中啟用就地主要版本升級的支援。如需詳細資訊，請參閱Amazon DocumentDB 就地主要版本升級。

日期	版本備註
2023 年 11 月 6 日	Amazon DocumentDB 4.0 （引擎修補程式 2.0.9876 版） 新功能 • 新增對$jsonSchema 結構描述關鍵字 maxLength 、minLength 、、maximumminimum、exclusiveMaximum 、exclusive Minimum 、 items和 的支援additionalItems 。 請注意，僅執行個體型叢集支援 JSON 結構描述驗證。 • 新增對$convert彙總管道運算子及其短期衍生運算子 $toBool、$toInt、$toLong、$toDouble 、$toDecimal 、 $toObjectId 和 $toString 的支援$toDate。 • 新增對設定表達式運算子 $setDifference 、 $anyEleme ntTrue 和 的支援$allElementTrue 。 錯誤修正和其他變更 已修正NaN未顯示從 -NaN到 的變更串流更新的問題。
2023 年 9 月 20 日	新功能 新增對 Amazon DocumentDB 4.0 中就地主要版本升級的支援。如需更多資訊，請參閱Amazon DocumentDB 就地主要版本升級。
2023 年 9 月 15 日	新功能 Amazon DocumentDB 4.0 （引擎修補程式 2.0.9686 版） 僅在執行個體型叢集中新增對 $jsonSchema 結構描述驗證器的支援。如需更多資訊，請參閱使用 JSON 結構描述驗證。

日期	版本備註
2023 年 7 月 13 日	新功能 Amazon DocumentDB 4.0（引擎修補程式 2.0.9259 版） • 新增對平行索引建置的支援。 錯誤修正和其他變更 Amazon DocumentDB 4.0（引擎修補程式 2.0.9259 版） • 已停止將內部監控查詢新增至稽核日誌。
2023 年 4 月 4 日	錯誤修正和其他變更 Amazon DocumentDB 4.0（引擎修補程式 2.0.8934 版） • 修正在持續工作負載期間啟用 DML 稽核的問題。 • 已修正使用提示彙總命令傳遞字串值時的 DML 稽核問題。 • 已修正當具有 readwriteanydatabase 角色的使用者同時將 authorizedCollections 和 nameOnly 選項設定為 true 時，`listCollections` 命令無法運作的問題。 • 已修正在欄位名稱中正確剖析數值字串的問題。 • 在影響垃圾回收時取消長時間執行的游標。 • 使用者定義的本機變數名稱現在支援投影運算子的 “_”（底線），例如 `$let`和 `$filter`。
2023 年 2 月 27 日	錯誤修正和其他變更 Amazon DocumentDB 4.0 新增 的支援 AWS Lambda。如需詳細資訊，請參閱搭配使用 AWS Lambda 與變更串流。

日期	版本備註
2022 年 8 月 9 日	新功能 4.0 • 新增對 Decimal128 資料類型的支援。Decimal128 是在提供 DocumentDB 的所有區域中支援的 BSON 資料類型。 如需詳細資訊，請參閱資料類型。 • 新增使用 Amazon CloudWatch Logs 進行 DML 查詢稽核的支援。現在，Amazon DocumentDB 可以將資料處理語言 (DML) 事件和資料定義語言 (DDL) 事件記錄到 Amazon CloudWatch Logs。 如需詳細資訊，請參閱這篇部落格文章。 錯誤修正和其他變更 4.0 • 您現在可以使用具有`changeOwnPassword` 權限的自有密碼來變更自己的 passwoprd。
2022 年 7 月 25 日	新功能 Amazon DocumentDB 4.0 您現在可以更快速地建立叢集，並能夠建立使用相同 DocumentDB 叢集磁碟區且具有與原始叢集相同資料的複製。如需詳細資訊，請參閱管理 Amazon DocumentDB 叢集。
2022 年 6 月 27 日	新功能 Amazon DocumentDB 4.0（引擎修補程式 2.0.7509 版） Amazon DocumentDB 會根據用量模式動態調整資料庫的大小。新增更多資料可將空間增加至 64 TB (TiB)，而刪除資料可減少配置的空間。

日期	版本備註
2022 年 4 月 7 日	新功能 4.0 （引擎修補程式版本 2.0.6142 和 2.0.6948) Amazon DocumentDB Performance Insights 現在處於預覽狀態。您現在可以在滾動時段中存放七天的效能歷史記錄，無需額外費用。如需詳細資訊，請參閱使用績效詳情監控。

2022 年 1 月 24 日	新功能
	
	4.0 （引擎修補程式 2.0.5170 版）
	
	• Amazon DocumentDB 現在提供免費試用。如需詳細資訊，請參閱 Amazon DocumentDB 免費試用頁面。
	• 您現在可以搭配地理空間查詢使用增強功能，包括下列 APIs：
	  • `$geoWithin`
	  • `$geoIntersects`
	• 新增對下列 MongoDB 運算子的支援：
	  • `$mergeObjects`
	  • `$reduce`
	
	如需詳細資訊，請參閱使用 Amazon DocumentDB 查詢地理空間資料。

日期	版本備註

2022 年 1 月 21 日	新功能
	
	Amazon DocumentDB 4.0 （引擎修補程式 2.0.5706 版）
	
	• 現在支援 Amazon DocumentDB Graviton2 (r6g.large、r6g.2xlarge、r6g.4xlarge、r6g.8xlarge、r6g.12xlarge、r6g.16xlarge 和 t4g.medium) 執行個體。
	
	新功能
	
	4.0 （引擎修補程式 2.0.5706 版）
	
	• 新增對下列 MongoDB APIs支援：
	  • `$reduce`
	  • `$mergeObjects`
	  • `$geoWithin`
	  • `$geoIntersects`

日期	版本備註

2021 年 10 月 25 日	新功能
	
	4.0 （引擎修補程式 2.0.5704 版）
	
	• 新增對下列 MongoDB APIs支援
	  • `$literal`
	  • `$map`
	  • `$$ROOT`
	• 支援 GeoSpatial Query 功能。如需詳細資訊，請參閱此部落格文章
	• 支援具有使用者定義角色的存取控制。如需詳細資訊，請參閱此部落格文章
	• Amazon DocumentDB JDBC 驅動程式可從 Tableau 和 SQL Workbench 等查詢工具等 BI 工具啟用連線
	
	錯誤修正和其他變更
	
	4.0 （引擎修補程式 2.0.5704 版）
	
	• 當明確 與 `.sort()` 同時存在時，錯誤修正`$natural`可讓 正確排序 `$natural`
	• 使用變更串流的錯誤修正 `$redact`
	• 使用空白陣列`$ifNull`的 錯誤修正
	• 當目前登入的使用者遭到刪除或該使用者對進行中活動的權限遭到撤銷時，資源消耗過多/伺服器當機的錯誤修正
	• 中的錯誤修正`listDatabase` 和`listCollection` 權限檢查
	• 多重金鑰元素的錯誤修正重複資料刪除邏輯

日期	版本備註
2021 年 6 月 24 日	新功能 4.0 （引擎修補程式 2.0.3371 版） • 現在支援 r5.8xlarge 和 r5.16xlarge 執行個體。如需進一步了解，請參閱部落格文章 Amazon DocumentDB Now Support r5.8xlarge 和 r5.16xlarge 執行個體。 • 全球叢集現在支援從全區域中斷提供災難復原，並允許從最近的 Amazon DocumentDB 叢集讀取，以啟用低延遲全域讀取。請注意，南美洲（聖保羅）、歐洲（米蘭）、中國（北京）和中國（寧夏）區域目前不支援全域叢集。
2021 年 5 月 4 日	新功能 4.0 （引擎修補程式 2.0.3371 版） • `renameCollection` • `$zip` • `$indexOfArray` • `$reverseArray` • `$natural` • `$hint` 支援更新 • 的索引掃描 `distinct` 錯誤修正和其他變更 4.0 （引擎修補程式 2.0.3371 版） • 減少`$in`查詢的記憶體用量 • 修正多金鑰索引中的記憶體流失 • 已修正 的 explain 計劃和分析器輸出 `$out` • 新增內部監控系統操作的逾時，以改善可靠性 • 已修正影響傳遞至多索引鍵索引之查詢述詞的瑕疵

日期	版本備註
2021 年 1 月 15 日	新功能 Amazon DocumentDB 4.0（引擎修補程式 2.0.722 版） • 無

日期	版本備註
2020 年 11 月 9 日	新功能 Amazon DocumentDB 4.0 （引擎修補程式 2.0.722 版） • MongoDB 4.0 相容性 • ACID 交易 • 支援 cluster(client.watch() 或 mongo.watch()) 和資料庫層級(db.watch()) 變更串流 • 能夠使用 啟動或繼續變更串流 startAtOperationTime • 將您的變更串流保留期間延長至 7 天（先前為 24 小時） • AWS DMS Amazon DocumentDB 4.0 的目標 • CloudWatch 指標：TransactionsOpen 、TransactionsOpenMax 、TransactionsStarted 、 TransactionsAborted 和 TransactionsCommitted • currentOp 、 ServerStatus 和 中交易的新欄位profiler。 • 能夠搭配$lookup彙總階段使用索引 • find() 具有投影的查詢可以從 索引提供方向（涵蓋的查詢） • 可與 hint()搭配使用 findAndModify • 運算$addToSet 子的效能最佳化 • 改善以減少整體索引大小。 • 新的彙總運算子：$ifNull、$replaceRoot 、$setIsSubset 、$setUnion 、 $setInstersection 和 $setEquals • 使用 ListCollection 和 ListDatabase 命令，您現在可以選擇性地使用 authorizedCollections 和 authorizedDatabases 參數，允許使用者列出他們有權存取的集合和資料庫，而分別不需要 listCollections 和 listDatabase 角色 • 使用者也可以結束自己的游標，而不需要 KillCursor 角色 • 比較子文件的數值類型現在與比較第一層文件的數值類型一致。Amazon DocumentDB 4.0 中的行為現在與 MongoDB 相容。 錯誤修正和其他變更

日期	版本備註
	Amazon DocumentDB 4.0 （引擎修補程式 2.0.722 版） • $setOnInsert 使用位置運算子 時， 不再允許更新$。Amazon DocumentDB 4.0 中的行為現在與 MongoDB 相容。 • 修正 $createCollection 和 設定的問題 autoIndexId • 巢狀文件的投影 • 變更工作記憶體的預設設定，以隨執行個體記憶體大小擴展 • 垃圾回收改善 • 路徑中具有空索引鍵的查詢，與 mongo 的行為差異 • 修正時區行為的dateToString 錯誤 • 修正 $push（彙總） 以遵守排序順序 • 已修正 中$currentOp 具有彙總的錯誤 • 修正次要 readPreference 上的 問題 • 修正驗證$createIndex 與發出命令的資料庫相同的問題 • 已修正 不一致的行為minKey，maxKey查詢失敗 • 修正運算$size子無法使用複合陣列的問題 • 已修正$in使用 regex 否定 的問題 • 修正針對檢視執行$distinct 命令的問題 • 修正彙總的問題，並尋找以不同方式排序遺漏欄位的命令 • 修正$eq為規則表達式而不檢查類型 • 修正時間戳記順序位置行為中的$currentDate 錯誤 • 的固定毫秒精細程度 $currentDate

Amazon DocumentDB 3.6

Amazon DocumentDB 引擎 3.6 版的版本備註 (MongoDB 3.6 相容性）。

日期	版本備註
2026 年 8 月 31 日	新功能

日期	版本備註
	Amazon DocumentDB 現在支援直接從引擎 3.6 版到 8.0 版的就地主要版本升級 (MVU)。這消除了中繼版本升級的需求，並可讓您升級 3.6 叢集，同時保留現有的資料、組態和叢集設定。此升級適用於目前支援這些版本的所有 AWS 區域。如需詳細資訊，請參閱Amazon DocumentDB 就地主要版本升級。
2026 年 8 月 18 日	安全性和穩定性改善 Amazon DocumentDB 3.6（引擎版本：1.0.208702) 【進行中】 此引擎修補程式提供下列改善： • 改善的請求處理 Amazon DocumentDB 現在可更徹底地驗證連線和訊息剖析路徑的傳入用戶端請求。這可改善更廣泛輸入模式的彈性和穩定性。 • 身分驗證交握強化 Amazon DocumentDB 現在可更徹底地驗證身分驗證交換。這可改善處理非預期或格式不正確的輸入時的穩定性。 如果您未更早套用，Amazon DocumentDB 會在發行 30 天後的下一個維護時段自動套用此修補程式。
2026 年 4 月 27 日	新功能 Amazon DocumentDB 3.6（引擎版本：1.0.208692) 此更新提供具有下列改進功能的選用引擎修補程式： 1. 在變更串流`startAtOperationTime` 中新增對 的支援。
2026 年 2 月 12 日	錯誤修正和其他變更 Amazon DocumentDB 3.6（引擎版本：1.0.208675) 1. `DatabaseConnectionsActiveMax` 指標現在`Connections (active)`會追蹤執行個體。如需詳細資訊，請參閱執行個體配額。 2. 改善 SASL 身分驗證效能。 3. $indexStats 命令的錯誤修正。

日期	版本備註
2024 年 8 月 20 日	新功能 對於 Amazon DocumentDB 3.6（最低引擎修補程式版本 1.0.208662），TLS CA 憑證更新不再需要系統重新啟動。如需詳細資訊，請參閱更新您的 Amazon DocumentDB TLS 憑證。
2024 年 7 月 22 日	新功能 Amazon DocumentDB 3.6（引擎修補程式版本 1.0.208662） 已移除 Amazon DocumentDB MVU 中 db.r5.* 和 db.r6.* 執行個體的索引限制。如需詳細資訊，請參閱先決條件。 錯誤修正和其他變更 Amazon DocumentDB 3.6（引擎修補程式版本 1.0.208662） Amazon DocumentDB 現在會將 -NaN 識別為有效的 JSON 字符。
2023 年 12 月 20 日	其他變更 在 Amazon DocumentDB 3.6 中啟用就地主要版本升級的支援。如需詳細資訊，請參閱Amazon DocumentDB 就地主要版本升級。
2023 年 11 月 17 日	錯誤修正和其他變更 Amazon DocumentDB 3.6（引擎修補程式版本 1.0.208570） 使用者定義的本機變數名稱現在支援投影運算子的 “_”（底線），例如 `$let`和 `$filter`。
2023 年 9 月 20 日	新功能 新增對 Amazon DocumentDB 3.6 中就地主要版本升級的支援。如需更多資訊，請參閱Amazon DocumentDB 就地主要版本升級。

日期	版本備註
2023 年 2 月 2 日	錯誤修正和其他變更 Amazon DocumentDB 3.6 （引擎修補程式版本 1.0.208432) • 修正在持續工作負載期間啟用 DML 稽核的問題。 • 已修正使用提示彙總命令傳遞字串值時的 DML 稽核問題。 • 已修正當具有 readwriteanydatabase 角色的使用者同時將 authorizedCollections 和 nameOnly 選項設定為 true 時，`listCollections` 命令無法運作的問題。 • 修正在欄位名稱中正確剖析數值字串的問題。 • 在影響垃圾回收時取消長時間執行的游標。
2022 年 8 月 9 日	新功能 Amazon DocumentDB 3.6 （引擎修補程式版本 1.0.208152) • 新增對 Decimal128 資料類型的支援。Decimal128 是在提供 DocumentDB 的所有區域中支援的 BSON 資料類型。 如需詳細資訊，請參閱資料類型。 • 新增使用 Amazon CloudWatch Logs 進行 DML 查詢稽核的支援。現在，Amazon DocumentDB 可以將資料處理語言 (DML) 事件和資料定義語言 (DDL) 事件記錄到 Amazon CloudWatch Logs。 如需詳細資訊，請參閱這篇部落格文章。 錯誤修正和其他變更 Amazon DocumentDB 3.6 （引擎修補程式版本 1.0.208152) • 您現在可以使用具有`changeOwnPassword` 權限的自有密碼來變更自己的 passwoprd。

日期	版本備註
2022 年 4 月 7 日	新功能 Amazon DocumentDB 3.6（引擎修補程式版本 1.0.207836 和 1.0.208015) Amazon DocumentDB Performance Insights 現在處於預覽狀態。您現在可以在滾動時段中存放七天的效能歷史記錄，無需額外費用。如需詳細資訊，請參閱使用績效詳情監控。

2022 年 1 月 24 日	新功能
	
	Amazon DocumentDB 3.6（引擎修補程式版本 1.0.207684)
	
	• Amazon DocumentDB 現在提供免費試用。如需詳細資訊，請參閱 Amazon DocumentDB 免費試用頁面。
	• 您現在可以搭配地理空間查詢使用增強功能，包括下列 APIs：
	  • `$geoWithin`
	  • `$geoIntersects`
	• 新增對下列 MongoDB 運算子的支援：
	  • `$mergeObjects`
	  • `$reduce`
	
	如需詳細資訊，請參閱使用 Amazon DocumentDB 查詢地理空間資料。

2022 年 1 月 21 日	新功能
	
	Amazon DocumentDB 3.6（引擎修補程式版本 1.0.207781)
	
	• 新增對下列 MongoDB APIs支援：
	  • `$reduce`
	  • `$mergeObjects`
	  • `$geoWithin`
	  • `$geoIntersects`

日期	版本備註

2021 年 10 月 25 日	新功能
	
	Amazon DocumentDB 3.6 （引擎修補程式版本 1.0.207780)
	
	• 新增對下列 MongoDB APIs支援
	  • `$literal`
	  • `$map`
	  • `$$ROOT`
	• 支援 GeoSpatial Query 功能。如需詳細資訊，請參閱此部落格文章
	• 支援具有使用者定義角色的存取控制。如需詳細資訊，請參閱此部落格文章
	• Amazon DocumentDB JDBC 驅動程式可從 Tableau 和 SQL Workbench 等查詢工具等 BI 工具啟用連線
	
	錯誤修正和其他變更
	
	Amazon DocumentDB 3.6 （引擎修補程式版本 1.0.207780)
	
	• 當明確 與 `.sort()` 同時存在時，錯誤修正`$natural`可讓 正確排序 `$natural`
	• 使用變更串流的錯誤修正 `$redact`
	• 修正錯誤`$ifNull`，讓 使用空白陣列
	• 當目前登入的使用者遭到刪除或該使用者對進行中活動的權限遭到撤銷時，資源消耗過多/伺服器當機的錯誤修正
	• 中的錯誤修正`listDatabase` 和`listCollection` 權限檢查
	• 多重金鑰元素的錯誤修正重複資料刪除邏輯

日期	版本備註
2021 年 6 月 24 日	新功能 Amazon DocumentDB 3.6（引擎修補程式版本 1.0.207117) • 現在支援 r5.8xlarge 和 r5.16xlarge 執行個體。如需進一步了解，請參閱部落格文章 Amazon DocumentDB Now Support r5.8xlarge 和 r5.16xlarge 執行個體。 • 全球叢集現在支援從全區域中斷提供災難復原，並允許從最近的 Amazon DocumentDB 叢集讀取，以啟用低延遲全域讀取。請注意，南美洲（聖保羅）、歐洲（米蘭）、中國（北京）和中國（寧夏）區域目前不支援全域叢集。
2021 年 5 月 4 日	新功能 Amazon DocumentDB 3.6（引擎修補程式版本 1.0.207117) • `renameCollection` • `$zip` • `$indexOfArray` • `$reverseArray` • `$natural` • `$hint` 支援更新 • 的索引掃描 `distinct` 錯誤修正和其他變更 Amazon DocumentDB 3.6（引擎修補程式版本 1.0.207117) • 減少`$in`查詢的記憶體用量 • 修正多金鑰索引中的記憶體流失 • 修正 的 explain 計劃和分析器輸出 `$out` • 新增來自內部監控系統之操作的逾時，以改善可靠性 • 已修正影響傳遞至多金鑰索引之查詢述詞的瑕疵

日期	版本備註
2021 年 1 月 15 日	新功能 Amazon DocumentDB 3.6 （引擎修補程式版本 1.0.206295) • 能夠搭配$lookup彙總階段使用索引 • find() 具有投影的查詢可以從 索引提供方向 （涵蓋的查詢） • 可與 hint() 搭配使用 findAndModify • 運算$addToSet 子的效能最佳化 • 改善以減少整體索引大小 • 新的彙總運算子：$ifNull、$replaceRoot 、$setIsSubset 、$setUnion 、 $setInstersection 和 $setEquals • 使用者也可以結束自己的游標，而不需要該KillCursor 角色
2020 年 11 月 9 日	新功能 Amazon DocumentDB 3.6 （引擎修補程式版本 1.0.206295) • 無 錯誤修正和其他變更 Amazon DocumentDB 3.6 （引擎修補程式版本 1.0.206295) • 無

日期	版本備註
2020 年 10 月 30 日	新功能 Amazon DocumentDB 3.6 （引擎修補程式版本 1.0.206295) • 新增在叢集層級(`client.watch()` 或 `mongo.watch()`) 和 資料庫開啟變更串流游標的功能 (`db.watch()`) • 能夠將變更串流保留期增加為 7 天 （先前為 24 小時） 錯誤修正和其他變更 Amazon DocumentDB 3.6 （引擎修補程式版本 1.0.206295) • 各種一般案例效能改善 • 目標安全性改善 • 修正複合索引的第二個欄位上略過排序的問題 • 在多索引鍵索引的單一欄位上啟用等式的規則索引 （非複合） • 已修正身分驗證競爭條件 • 已修正導致不常發生垃圾回收損毀的問題 • RBAC 安全性改善 • 新增`databaseConnectionsMax` 指標 • 改善`r5.24xlarge` 執行個體上特定工作負載的效能
2020 年 9 月 22 日	新功能 Amazon DocumentDB 3.6 （引擎修補程式版本 1.0.206295) • `$out` 彙總階段 • 每個執行個體的連線和游標數量上限增加高達 10 倍 錯誤修正和其他變更 Amazon DocumentDB 3.6 （引擎修補程式版本 1.0.206295) • 無

日期	版本備註
2020 年 7 月 10 日	新功能 Amazon DocumentDB 3.6（引擎修補程式版本 1.0.206295) • 跨區域快照複製 錯誤修正和其他變更 Amazon DocumentDB 3.6（引擎修補程式版本 1.0.206295) • 無

日期	版本備註
2020 年 6 月 30 日	新功能 Amazon DocumentDB 3.6 （ 引擎修補程式版本 1.0.206295) • T3 中型執行個體 錯誤修正和其他變更 Amazon DocumentDB 3.6 （ 引擎修補程式版本 1.0.206295) • t3 執行個體的閒置記憶體回收 • 身分驗證改善 • 改善 SASL 身分驗證效能 • 已修正超過可能操作上限時`currentOp` 的問題 • 修正大量更新和刪除`killOps`的問題 • 使用 改善`$sample`效能 `$match` • 已修正在修訂階段`$$`的 cond 案例中對 的支援 • 修正各種經常性當機根本原因 • 改善 TTL 掃描以減少 IOs和延遲 • 最佳化的記憶體使用率 `$unwind` • 已修正具有捨棄索引的集合統計資料競爭條件 • 已修正並行索引建置期間的競爭條件 • 修正索引`hash_search` 中不常當機的問題

一般 Amazon DocumentDB 版本備註

一般服務更新，包括區域可用性、工具整合和跨版本功能。

日期	版本備註
2026 年 4 月 29 日	新功能

日期	版本備註
	Amazon DocumentDB 現已在加拿大西部（卡加利）區域提供。如需詳細資訊，請參閱這篇部落格文章。
2026 年 4 月 17 日	新功能 DocumentDB Serverless 上的擴展速度更快，現已推出。
2026 年 2 月 11 日	新功能 Amazon DocumentDB 現已在亞太區域（墨爾本）區域提供。如需詳細資訊，請參閱這篇部落格文章。 Amazon DocumentDB 現已在歐洲（蘇黎世）區域提供。如需詳細資訊，請參閱這篇部落格文章。
2026 年 1 月 8 日	新功能 Amazon DocumentDB 現已在亞太區域（雅加達）區域提供。如需詳細資訊，請參閱這篇部落格文章。
2025 年 10 月 13 日	新功能 Amazon DocumentDB 現在支援亞太區域（海德拉巴）區域中的 R6G 和 T4G 執行個體。 如需詳細資訊，請參閱各區域的支援執行個體類別。
2025 年 10 月 7 日	新功能 Amazon DocumentDB 現在可在下列區域使用： • 亞太地區 (馬來西亞) • 亞太地區 (大阪) • 亞太區域 (泰國) • 墨西哥 (中部) 如需詳細資訊，請參閱這篇部落格文章。

日期	版本備註
2025 年 9 月 26 日	新功能 Amazon DocumentDB 現在支援跨區域和跨帳戶快照複製和快照共用。 如需詳細資訊，請參閱跨 複製快照 AWS 區域及共用 Amazon DocumentDB 叢集快照。
2025 年 6 月 18 日	新功能 Amazon DocumentDB 現已在以色列（特拉維夫）區域提供。
2025 年 5 月 8 日	新功能 Amazon DocumentDB 現已在歐洲（斯德哥爾摩）區域提供。如需詳細資訊，請參閱這篇部落格文章。
2025 年 4 月 2 日	錯誤修正和其他變更 已修正使用背景索引建立向量索引時的錯誤。
2025 年 2 月 6 日	新功能 Amazon DocumentDB 現在已與 整合 AWS Toolkit for Visual Studio Code。如需詳細資訊，請參閱此最新消息部落格文章，並參閱AWS Toolkit for Visual Studio Code 《使用者指南》中的 Toolkit 中的使用 Amazon DocumentDB。
2024 年 10 月 22 日	新功能 Amazon DocumentDB 現在支援儲存網路輸送量指標 `StorageNetworkReceiveThroughput` 、 `StorageNetworkTransmitThroughput` 和 `StorageNetworkThroughput` 。如需詳細資訊，請參閱使用 CloudWatch 指標評估 Amazon DocumentDB 執行個體用量。

日期	版本備註
2024 年 9 月 18 日	新功能 Amazon DocumentDB 現已在非洲（開普敦）區域提供。如需詳細資訊，請參閱這篇部落格文章。 Amazon DocumentDB 現已在歐洲（西班牙）區域提供。如需詳細資訊，請參閱這篇部落格文章。
2024 年 6 月 25 日	新功能 使用 AWS IAM ARNs身分驗證可在所有支援區域的 Amazon DocumentDB 執行個體型 5.0 叢集中使用。如需詳細資訊，請參閱使用 IAM 身分進行身分驗證。
2024 年 4 月 3 日	新功能 Amazon DocumentDB 現已在中東（阿拉伯聯合大公國）區域提供。如需詳細資訊，請參閱這篇部落格文章。 錯誤修正和其他變更 • 修正在 mongo shell 1.7 版和更新版本`db.coll.stats()` 上調用 時出現的錯誤。 • 修正在相同彙總管道`$regex`中包含 的變更串流查詢的記憶體流失問題。
2024 年 1 月 30 日	新功能 Amazon DocumentDB 全域叢集 全球叢集現在可在下列兩個 AWS GovCloud (US) 區域使用： AWS GovCloud（美國東部）和 AWS GovCloud（美國西部）。
2023 年 12 月 13 日	新功能 新增對一鍵式 EC2 連線的支援。如需詳細資訊，請參閱使用 Amazon EC2 連線。

日期	版本備註
2023 年 11 月 17 日	新功能 Amazon DocumentDB 現已在 AWS GovCloud （美國東部） 區域提供。如需詳細資訊，請參閱這篇部落格文章。
2023 年 9 月 25 日	新功能 Amazon DocumentDB 現已在亞太區域 （香港） 區域提供。如需詳細資訊，請參閱這篇部落格文章。
2023 年 9 月 11 日	新功能 Amazon DocumentDB 現已在亞太區域 （海德拉巴） 區域提供。如需詳細資訊，請參閱這篇部落格文章。
2022 年 4 月 29 日	新功能 Amazon DocumentDB 現已在中國 （北京） 區域提供。如需詳細資訊，請參閱這篇部落格文章。
2022 年 3 月 16 日	新功能 Amazon DocumentDB 現已在歐洲 （米蘭） 區域提供。如需詳細資訊，請參閱這篇部落格文章。
2022 年 2 月 8 日	新功能 Amazon DocumentDB R6G 和 T4G 執行個體現已在亞太區域、南美洲和歐洲提供。如需詳細資訊，請參閱這篇部落格文章。
2021 年 5 月 4 日	新功能 請參閱此部落格文章中的所有新功能。
2020 年 11 月 9 日	新功能 請參閱此部落格文章中的所有新功能。

日期	版本備註
2020 年 10 月 30 日	新功能 查看此部落格文章中的所有新功能。
2020 年 9 月 22 日	新功能 請參閱此部落格文章中的所有新功能。
2020 年 7 月 10 日	新功能 請參閱此部落格文章中的所有新功能。
2020 年 6 月 30 日	新功能 請參閱此部落格文章中的所有新功能。

Amazon DocumentDB 彈性叢集版本備註

Amazon DocumentDB 彈性叢集的版本備註。

日期	版本備註
2025 年 1 月 28 日	新功能 對於一鍵式連線，Amazon DocumentDB 現在已與 整合 AWS CloudShell，用於執行個體型和彈性叢集。 如需詳細資訊，請參閱下列任何或所有項目： • 開始使用 Amazon DocumentDB • Amazon DocumentDB 彈性叢集入門 • 什麼是新的部落格文章 • 技術操作指南部落格文章
2024 年 11 月 12 日	新功能 Amazon DocumentDB 彈性叢集現在支援背景索引。

日期	版本備註
2024 年 11 月 1 日	新功能 Amazon DocumentDB 現在支援彈性叢集維護動作。如需詳細資訊，請參閱維護 Amazon DocumentDB 彈性叢集。
2024 年 8 月 8 日	新功能 Amazon DocumentDB 彈性叢集現可於亞太區域（香港）、加拿大（中部）和歐洲（巴黎）區域使用。如需詳細資訊，請參閱彈性叢集的區域可用性。
2024 年 7 月 8 日	新功能 Amazon DocumentDB 彈性叢集現已在歐洲（米蘭）區域提供。如需詳細資訊，請參閱彈性叢集的區域可用性。
2024 年 2 月 22 日	新功能 Amazon DocumentDB 彈性叢集 Amazon DocumentDB 彈性叢集現在支援下列功能： • 可讀取的次要碎片執行個體複本 - 如需詳細資訊，請參閱 的步驟 5b步驟 1：建立彈性叢集。 • 開始/停止叢集 - 如需詳細資訊，請參閱 停止和啟動 Amazon DocumentDB 彈性叢集。 • 可設定的碎片執行個體 - 如需詳細資訊，請參閱 的步驟 5b步驟 1：建立彈性叢集。 • 快照的自動備份 - 如需詳細資訊，請參閱 管理彈性叢集快照自動備份。 • 複製快照 - 如需詳細資訊，請參閱 複製彈性叢集快照。

日期	版本備註
2024 年 1 月 30 日	新功能 Amazon DocumentDB 彈性叢集 Amazon DocumentDB 彈性叢集現在可在下列區域使用： • 亞太地區 (孟買) • 亞太地區 (首爾) • 南美洲 (聖保羅) • 歐洲 (倫敦) 如需詳細資訊，請參閱彈性叢集區域和版本可用性。

2023 年 8 月 3 日	新功能
	
	Amazon DocumentDB 彈性叢集
	
	• Amazon DocumentDB Elastic 叢集現在支援下列操作：
	  • `top`
	  • `collStats`
	  • `hint`
	  • `dataSize`
	
	  Amazon DocumentDB 中支援的 MongoDB APIs、操作和資料類型 如需支援命令和操作的完整清單，請參閱 。
	• 現在支援存留時間 (TTL) 索引。
	• 索引運算式`hints`現在支援索引。

2023 年 3 月 22 日	新功能 Amazon DocumentDB 彈性叢集現在可在亞太區域 （新加坡）、亞太區域 （雪梨） 和亞太區域 （東京） 區域使用。如需詳細資訊，請參閱彈性叢集區域和版本可用性。

日期	版本備註
2022 年 11 月 30 日	新功能 Amazon DocumentDB 彈性叢集 Amazon DocumentDB 彈性叢集是一種新型的 Amazon DocumentDB 叢集，可讓使用者利用 MongoDB 碎片 APIs 來擴展其叢集。彈性叢集透過將資料和運算分散到多個基礎運算執行個體和磁碟區，以 PB 儲存容量處理幾乎任意數量的讀取和寫入。若要進一步了解，請參閱使用 Amazon DocumentDB 彈性叢集。

Amazon DocumentDB 開發人員指南的文件歷史記錄

- API 版本：2014-10-31
- 文件最近更新時間：2026 年 5 月 27 日

下表說明此版本 Amazon DocumentDB 開發人員指南的文件。

變更	描述	日期
新的 Amazon DocumentDB 開發人員指南	Amazon DocumentDB 開發人員指南現在會在新的 URL 發佈。API 參考可在 Amazon DocumentDB API 參考中單獨取得。	2026 年 5 月 27 日
AWS 受管政策更新 - 政策變更	Amazon DocumentDB 會更新彈性叢集的完整存取政策。	2025 年 2 月 11 日
AWS 受管政策更新 - 政策變更	Amazon DocumentDB 會更新彈性叢集的完整存取政策。	2024 年 2 月 21 日
AWS 受管政策更新 - 政策變更	Amazon DocumentDB 會更新彈性叢集的唯讀和完整存取政策。	2023 年 6 月 21 日
AWS 受管政策更新 - 新政策	Amazon DocumentDB 為彈性叢集引進新的唯讀政策。	2023 年 6 月 8 日
AWS 受管政策更新 - 新政策	Amazon DocumentDB 為彈性叢集推出新的完整存取政策。	2023 年 6 月 5 日
MongoDB 5.0 相容性	Amazon DocumentDB 現在與 MongoDB 的 5.0 版相容。	2023 年 3 月 1 日
政策更新	為了支援 Amazon DocumentDB 彈性叢集功能，AmazonDocDBConsoleFullAccess 政策已更新，並	2022 年 11 月 30 日

	推出 AmazonDocDB-ElasticServiceRolePolicy。	
彈性叢集	新增 Elastic Cluster 功能，支援跨 Amazon DocumentDB 分散式儲存系統以雜湊為基礎的資料分割（分片）。	2022 年 11 月 30 日
全域叢集	新增有關如何使用 全域叢集的文件。	2021 年 6 月 2 日
事件訂閱	新增事件訂閱文件。	2021 年 3 月 26 日
3.6 版升級	已記錄 3.6 版在角色型存取控制、彙總運算子和效能方面的改進。	2021 年 1 月 15 日
MongoDB 4.0 相容性	Amazon DocumentDB 現在與 MongoDB 的 4.0 版相容。	2020 年 11 月 9 日
入門指南	使用、Amazon EC2、Robo3T 或 Studio3T 開始使用 Amazon DocumentDB 的新入門指南。AWS Cloud9 Amazon EC2 Robo3T Studio3T	2020 年 8 月 15 日
支援的其他可用區域	Amazon DocumentDB 新增對亞太區域（首爾）(ap-northeast-2) 中其他可用區域的支援。	2020 年 7 月 14 日
新增跨區域複製快照的支援。	Amazon DocumentDB 新增了跨 複製叢集快照的支援 AWS 區域。如需詳細資訊，請參閱跨區域複製快照。	2020 年 7 月 10 日

新增對 T3 執行個體類別的支援。	新增支援 Amazon DocumentDB 的所有區域中對 T3 執行個體類型的支援。如需詳細資訊，請參閱區域和執行個體類別規格支援的執行個體類別。	2020 年 6 月 30 日
新增 的支援 AWS GovCloud (US)。	Amazon DocumentDB 現已在 AWS GovCloud (US) 區域 (us-gov-west-1) 提供。	2020 年 6 月 29 日
新增了 16 個新的 CloudWatch 指標。	Amazon DocumentDB 新增對 16 個新 Amazon CloudWatch 指標的支援。如需詳細資訊，請參閱使用 CloudWatch 監控 Amazon DocumentDB。	2020 年 6 月 23 日
新增對 null 字元和 $regex 運算子的支援。	Amazon DocumentDB 新增對字串中 null 字元的支援，以及使用 $regex 索引的能力。若要檢視 Amazon DocumentDB 支援的 MongoDB APIs 和彙總管道功能，請參閱與 MongoDB 的功能差異。	2020 年 6 月 22 日
新增對改善多金鑰索引功能的支援。	Amazon DocumentDB 新增了對改進的多金鑰索引功能的支援，包括對大於 2，048 個位元組的陣列編製索引，以及能夠建立具有相同陣列中多個金鑰的複合多金鑰索引。如需詳細資訊，請參閱 MongoDB 的功能差異。	2020 年 4 月 23 日
新增對 Amazon DocumentDB CloudFormation 堆疊刪除保護的支援。	Amazon DocumentDB 新增了在建立 Amazon DocumentDB CloudFormation 堆疊時啟用刪除保護的支援。	2020 年 4 月 20 日

新增對角色型存取控制的支援。	Amazon DocumentDB 新增對使用內建角色的角色型存取控制的支援。	2020 年 3 月 26 日
新增對加拿大（中部）(ca-central-1) 中其他可用區域的支援。	Amazon DocumentDB 現在可在具有 R5 類別執行個體和 3 個可用區域的加拿大（中部）區域 (ca-central-1) 中使用。	2020 年 3 月 26 日
新增對兩個額外 MongoDB APIs支援。	Amazon DocumentDB 新增對 `$dateFromString` 和 `executionStats` MongoDB APIs支援。	2020 年 3 月 23 日
新增對五個額外 MongoDB APIs支援。	Amazon DocumentDB 新增對 `$objectToArray` 、`$arrayTo0bject` 、`$mod`、 `$slice`和 `$range` MongoDB APIs支援。	2020 年 2 月 6 日
新增對加拿大（中部）的支援。	Amazon DocumentDB 現在可在具有 R5 類別執行個體的加拿大（中部）區域 (ca-central-1) 中使用。	2019 年 12 月 11 日
新增對 ChangeStreamLogSize 的支援。	Amazon DocumentDB 新增對 Cloudwatch 指標`ChangeStreamLogSize` 的支援。	2019 年 11 月 22 日
新增對歐洲（巴黎）區域的支援	Amazon DocumentDB 現在可在具有 R5 類別執行個體的歐洲（巴黎）區域 (eu-west-3) 中使用。	2019 年 10 月 30 日
新增對亞太區域（孟買）區域的支援	Amazon DocumentDB 現在可在具有 R5 類別執行個體的亞太區域（孟買）區域 (ap-south-1) 中使用。	2019 年 10 月 17 日

新增對三個額外 MongoDB APIs支援	Amazon DocumentDB 新增對 `$addFields` 、 `$concatArrays` 和 `$lookup` MongoDB APIs支援。	2019 年 10 月 16 日
新增對亞太區域（新加坡）區域的支援	Amazon DocumentDB 現在可在具有 R5 類別執行個體的亞太區域（新加坡）區域 (ap-southeast-1) 中使用。	2019 年 10 月 14 日
新增更新 TLS 憑證的文件	新增更新 CA 憑證以使用新的 CA 憑證建立 TLS 連線的說明。	2019 年 10 月 2 日
新增憑證的 API 支援	Amazon DocumentDB 執行個體的新憑證資料類型。如需詳細資訊，請參閱 DBInstance。	2019 年 10 月 1 日
支援查詢分析	Amazon DocumentDB 新增了在叢集執行個體和資料庫上描述支援操作的功能。	2019 年 8 月 19 日
在亞太區域（東京）新增了第三個可用區域	Amazon DocumentDB 為亞太區域（東京）的運算執行個體新增了第三個可用區域 (AZ)。	2019 年 8 月 9 日
支援其他 Mongo APIs	新增對其他彙總管道功能的支援，包括 `$in`、`$isoWeek`、`$isoDayOfWeek` 、 `$isoWeekYear` 和 `$dateToString` 彙總運算子，以及`$addToSet` 彙總階段。Amazon DocumentDB 也新增了對集合層級診斷命令的支援，以及使用 `top()` 命令修改 TTL 索引`expireAfterSeconds` 參數的能力`collMod()` 。	2019 年 7 月 31 日

新增對歐洲（倫敦）的支援	Amazon DocumentDB 現已在歐洲（倫敦）(eu-west-2) 使用 R5 類別執行個體。	2019 年 7 月 18 日
新增程式碼範例	在 R 和 Ruby 中新增以程式設計方式連線至 Amazon DocumentDB 的程式碼範例。	2019 年 7 月 17 日
新增最佳實務	新增最佳實務，協助您管理 Amazon DocumentDB 成本。	2019 年 7 月 17 日
支援停止和啟動叢集	Amazon DocumentDB 新增了停止和啟動叢集的支援，以管理開發和測試環境的成本。	2019 年 7 月 1 日
支援叢集刪除保護	為了保護您的叢集免於意外刪除，Amazon DocumentDB 新增了刪除保護。如需詳細資訊，請參閱 API 主題 DBCluster `DeletionProtection` 中的下列主題：建立 Amazon DocumentDB 叢集、修改 Amazon DocumentDB 叢集、刪除 Amazon DocumentDB 叢集。	2019 年 7 月 1 日
功能差異更新	在功能差異中新增了隱含交易。	2019 年 6 月 26 日
新增功能差異	新增有關 Amazon DocumentDB 中儲存和索引壓縮的備註。	2019 年 6 月 13 日
支援的其他區域	Amazon DocumentDB 現在可在具有 R5 類別執行個體的亞太區域（雪梨）(ap-southeast-2) 中使用。	2019 年 6 月 5 日

其他 區域中支援的 R5 執行個體類別	新增 4 個其他區域的 R5 執行個體類別支援：美國東部（俄亥俄）、美國東部（維吉尼亞北部）、美國西部（奧勒岡）和歐洲（愛爾蘭）。透過此變更，支援 Amazon DocumentDB 的所有區域都支援 R5 執行個體。	2019 年 5 月 17 日
支援的其他區域	新增對具有 R5 執行個體類別之其他 2 個區域的支援：亞太區域（東京）(ap-northeast-1) 和亞太區域（首爾）(ap-northeast-2)。如需詳細資訊，請參閱各區域支援的執行個體類別和執行個體類別規格。	2019 年 5 月 8 日
新增更多連線程式碼範例	在 Java 和 C# 中新增連線至 Amazon DocumentDB 的程式碼範例。	2019 年 4 月 24 日
其他 Mongo API 支援	已新增下列支援：7 個彙總字串運算子 (`$indexOfBytes`、`$indexOfCP`、`$strLenBytes`、`$strLenCP`、`$toLower`、`$toUpper` 和 `$split`)、9 個日期時間運算子 (`$dayOfYear`、`$dayOfMonth`、`$dayOfWeek`、`$year`、`$month`、`$hour`、`$minute`、`$second` 和 `$millisecond`)，以及 `$sample` 彙總管道階段。	2019 年 4 月 4 日

新增連線程式碼範例	在 Python、Node.js、PHP 和 Go 中新增連線至 Amazon DocumentDB 的程式碼範例。	2019 年 3 月 21 日
支援法蘭克福區域和 R5 執行個體	新增對具有 R5 執行個體類別的歐洲（法蘭克福）區域 (eu-central-1) 的支援。如需詳細資訊，請參閱各區域支援的執行個體類別和執行個體類別規格。	2019 年 3 月 13 日
彙總管道運算子支援	已新增下列支援：新的彙總字串運算子 ($concat、$substr、$substrBytes 、$substrCP 、$strcasecmp)、陣列彙總運算子 ($size)、彙總群組累加運算子 ($push) 和彙總階段 ($redact 和 $indexStats)。我們也新增了位置陣列運算子 ($[] 和 $[<identifier>]) 及 hint() 的支援。	2019 年 2 月 28 日
引擎升級	已新增文件，以判斷待定叢集修改和升級您叢集的引擎版本。	2019 年 2 月 15 日
稽核事件	新增使用 Amazon CloudWatch Logs 稽核資料庫事件的支援。	2019 年 2 月 12 日
Quick Start	新增 Quick Start 主題，以協助您輕鬆地開始使用 Amazon DocumentDB CloudFormation。	2019 年 1 月 11 日

公有版本	這是 Amazon DocumentDB 的初始公有版本（與 MongoDB 相容）。此版本包括開發人員指南和整合的資源管理 API 參考。	2019 年 1 月 9 日

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。