aws

デベロッパーガイド

Amazon DocumentDB



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

Amazon DocumentDB: デベロッパーガイド

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

Amazon の商標およびトレードドレスは Amazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

Amazon DocumentDB とは ... 1
概要 ... 1
クラスター ... 3
インスタンス ... 4
リージョンと AZ ... 9
リージョン ... 9
アベイラビリティーゾーン ... 10
料金 ... 12
無料トライアル ... 13
モニタリング ... 13
インターフェイス ... 14
AWS マネジメントコンソール ... 14
AWS CLI ... 14
MongoDB ドライバー ... 14
次のステップ ... 14
仕組み ... 15
Amazon DocumentDB バージョン ... 15
Amazon DocumentDB エンドポイント ... 18
TLS サポート ... 21
Amazon DocumentDB のストレージ ... 22
Amazon DocumentDB レプリケーション ... 23
Amazon DocumentDB の信頼性 ... 23
読み込み設定のオプション ... 24
TTL 削除 ... 29
有料リソース ... 30
ドキュメントデータベースとは ... 33
ユースケース ... 33
ドキュメントについて理解する ... 34
ドキュメントでの作業 ... 40
入門ガイド ... 53
前提条件 ... 53
ステップ 1: クラスターを作成する ... 55
ステップ 2: クラスターに接続する ... 58
ステップ 3: データを挿入してクエリする ... 59

ステップ 4: 探索する 61
を使用したクイックスタート CloudFormation 61
前提条件 62
Amazon DocumentDB CloudFormation スタックの起動 63
Amazon DocumentDB クラスターにアクセスする 68
終了保護と削除保護 68
移行とアップグレード 70
「クイックスタートガイド」 70
DMC ソースの準備 71
DMS のセットアップ 71
DocumentDB 圧縮を有効にする 71
レプリケーションタスクを作成する 71
進行状況をモニタリングする 72
追加情報 72
移行ランブック 73
互換性 73
ワークロードの検出 76
新しいターゲットクラスターを作成する 79
インデックスの移行 80
ユーザー移行 81
データ移行 83
モニタリング 93
検証 96
Couchbase サーバーからの移行 101
序章 102
Amazon DocumentDB との比較 102
発見 104
計画 110
移行 113
検証 122
エンジンバージョンのサポート日 125
Amazon DocumentDB メジャーバージョンのリリースカレンダー 125
Amazon DocumentDB マイナーバージョンのリリースカレンダー 126
メジャーエンジンバージョンのアップグレード 127
サポートされているアップグレードパス 127
前提条件 128

ベストプラクティス 130
Amazon DocumentDB 8.0 にアップグレードした後の変更点 134
3.6 または 4.0 からアップグレードされたクラスターのアップグレード後の考慮事項 135
アップグレードの実行 135
トラブルシューティング 138
マイナーエンジンバージョンのアップグレード 138
マイナーバージョンアップグレードを実行する前に 139
マイナーバージョンアップグレードの実行 139
マイナーバージョンアップグレードの検証 140
を使用したアップグレード AWS DMS 140
ステップ 1: 変更ストリームを有効にする 141
ステップ 2: 変更ストリームログの保持期間を変更する 141
ステップ 3: インデックスを移行する 142
ステップ 4: AWS DMS レプリケーションインスタンスを作成する 143
ステップ 5: AWS DMS ソースエンドポイントを作成する 145
ステップ 6: AWS DMS ターゲットエンドポイントを作成する 148
ステップ 7: 移行タスクの作成と実行 150
ステップ 8: アプリケーションエンドポイントから Amazon DocumentDB クラスター への変更 152
長期サポート (LTS) リリースの使用 152
現在の LTS リリース 152
LTS を使用するユーザー 153
新しい LTS クラスターを作成する方法 153
エンジンバージョン 3.6.0 または 4.0.0 から 5.0.0 LTS クラスターにアップグレードする方法 154
既存のエンジンバージョン 5.0.0 クラスターが LTS にあるかどうかを確認する方法 154
延長サポート 154
延長サポートの概要 156
延長サポート料金 156
責任 157
機能と設定 158
エンジンのステータス 158
クラスタータイプ 158
機能 159
インスタンスのタイプ 160
AWS 統合 161

リージョンとアベイラビリティーゾーンの選択 162
リージョンの可用性 163
OpenSearch 統合 166
送信先としての Amazon OpenSearch Service。 166
制限事項 170
開発 171
DocumentDB に接続する 171
プログラムによる接続 172
Amazon VPC の外部からの接続 201
レプリカセットとして接続する 203
Studio 3T を使用して接続 208
DataGrip を使用して接続する 217
Amazon EC2 を使用した接続 222
JDBC ドライバーによる接続 259
ODBC ドライバーを使用して接続します 276
Amazon DocumentDB を使用したプログラム 290
DocumentDB Java プログラミングガイド 290
JSON スキーマ検証の使用 322
クエリ 333
ドキュメントのクエリ 333
クエリプラン 336
結果の説明 338
クエリプランナー v2 343
クエリプランナー v3 358
地理空間データ 363
部分インデックス 369
テキスト検索 377
インデックス 385
インデックス管理 385
インデックスタイプ 396
インデックスプロパティ 405
変更ストリームの使用 424
サポートされているオペレーション 425
拡張イベント 426
料金 427
制限事項 427

変更ストリームの有効化 ... 428
Python での変更ストリームの使用 ... 430
ドキュメント全体の参照 ... 432
$changeStreamSplitLargeEvent を使用して大きなイベントをフラグメントに分割する ... 433
変更ストリームの再開 ... 434
startAtOperationTime を用いて再開する ... 435
postBatchResumeToken を用いて再開する ... 437
取引 ... 439
ログ保持期間の変更 ... 439
セカンダリインスタンスでの変更ストリーム ... 442
取引 ... 444
要件 ... 444
ベストプラクティス ... 444
制限事項 ... 445
モニタリングと診断 ... 446
トランザクション分離レベル ... 446
ユースケース ... 447
サポートされているコマンド ... 485
サポート対象外の機能 ... 485
セッション ... 486
トランザクションエラー ... 488
再試行可能な書き込み ... 488
要件 ... 489
サポートされているオペレーション ... 489
制限事項 ... 490
有効化 ... 490
仕組み ... 491
エラー処理 ... 491
移行 ... 491
照合の使用 ... 492
制限事項 ... 492
ビューの使用 ... 493
ベストプラクティス ... 493
アグリゲータ演算子の互換性 ... 494
インデックスとビューの活用 ... 494
生成 AI ... 495

SageMaker Canvas ... 495
ベクトル検索 ... 498
変更ストリーム AWS Lambda での の使用 ... 511
制限事項 ... 512
MongoDB との互換性 ... 513
MongoDB 8.0 の互換性 ... 513
Amazon DocumentDB 8.0 の新機能 ... 513
Amazon DocumentDB の使用開始 ... 514
Amazon DocumentDB 5.0 または 8.0 へのアップグレードまたは移行 ... 515
機能的な違い ... 515
MongoDB 5.0 との互換性 ... 515
Amazon DocumentDB 5.0 の新機能 ... 516
Amazon DocumentDB 5.0.1 の新機能 ... 516
Amazon DocumentDB 5.0 の使用開始 ... 514
Amazon DocumentDB 5.0 へのアップグレードまたは移行 ... 515
機能的な違い ... 515
MongoDB 4.0 の互換 ... 520
Amazon DocumentDB 4.0 機能 ... 520
Amazon DocumentDB 4.0 の使用開始 ... 521
Amazon DocumentDB 4.0 へのアップグレードまたは移行 ... 522
機能的な違い ... 522
MongoDB との機能的な違い ... 523
Amazon DocumentDB の機能上の利点 ... 523
機能的な違いの更新 ... 525
MongoDB との機能的な違い ... 528
サポートされている MongoDB API、オペレーション、およびデータ型 ... 537
データベースコマンド ... 538
クエリおよびプロジェクション演算子 ... 548
更新演算子 ... 551
地理空間 ... 554
カーソルメソッド ... 555
集約パイプライン演算子 ... 557
データ型 ... 576
インデックスとインデックスプロパティ ... 577
集約パイプライン演算子 ... 579
地理空間 ... 1203

クエリおよびプロジェクション演算子 1228
更新演算子 1295
クラスターとインスタンスの管理 1349
運用タスクの概要 1349
Amazon DocumentDB クラスターにレプリカを追加する 1350
クラスターとインスタンスの説明 1350
クラスタースナップショットの作成 1353
スナップショットからの復元 1354
クラスターからインスタンスを削除する 1355
クラスターの削除 1355
グローバルクラスター 1356
グローバルクラスターとは 1356
グローバルクラスターはどのように役立つのか 1356
グローバルクラスターの現在の制限は何ですか。 1357
「クイックスタートガイド」 1358
グローバルクラスターの管理 1370
グローバルクラスターの接続 1378
グローバルクラスターのモニタリング 1379
ディザスタリカバリ 1379
クラスターの管理 1404
クラスターについて 1405
[クラスター設定] 1407
クラスターのストレージ設定 1410
クラスターのステータスの評価 1413
クラスターのライフサイクル 1415
クラスターのスケーリング 1458
クラスターボリュームのクローン作成 1462
クラスターの耐障害性について 1474
インスタンスの管理 1476
インスタンスのステータスの判断 1476
インスタンスのライフサイクル 1476
インスタンスクラスの管理 1500
NVMe-backed インスタンス 1514
サブネットグループの管理 1517
サブネットグループの作成 1518
サブネットグループの説明 1523

サブネットグループの変更 1526
サブネットグループの削除 1529
高可用性とレプリケーション 1531
読み取りのスケーリング 1531
高可用性 1532
レプリカの追加 1533
フェイルオーバー 1534
レプリケーションの遅延 1539
イベントの管理 1540
イベントカテゴリの表示 1540
Amazon DocumentDB イベントの表示 1543
クラスターパラメータグループの管理 1546
クラスターパラメータグループの説明 1546
クラスターパラメータグループの作成 1554
クラスターパラメータグループの変更 1557
カスタマイズしたクラスターパラメータグループを使用するためのクラスターの変更 1562
クラスターパラメータグループのコピー 1563
クラスターパラメータグループのリセット 1565
クラスターパラメータグループの削除 1568
クラスターパラメータリファレンス 1571
エンドポイントについて 1592
クラスターのエンドポイントの検索 1593
インスタンスのエンドポイントの検索 1595
エンドポイントへの接続 1599
Amazon DocumentDB ARN の理解 1600
ARN の構築 1600
ARN の検索 1605
リソースのタグ付け 1607
リソースタグの概要 1607
タグの制約 1608
タグの追加または更新するには 1609
タグを一覧表にする 1610
タグの削除 1612
Amazon DocumentDB のメンテナンス 1613
Amazon DocumentDB のメンテナンスアクション 1615
バージョン番号付け 1615

メンテナンスウィンドウの管理 ... 1616
エンジンパッチ通知 ... 1619
実施待ちのメンテナンスの確認 ... 1621
エンジンの更新 ... 1624
マイナーバージョン ... 1627
オペレーティングシステムの更新 ... 1628
ユーザーが起動する更新 ... 1632
グローバルクラスター ... 1633
サービスにリンクされたロールを理解する ... 1634
サービスにリンクされたロールのアクセス許可 ... 1634
サービスにリンクされたロールの作成 ... 1636
サービスにリンクされたロールの変更 ... 1636
サービスにリンクされたロールの削除 ... 1637
Amazon DocumentDB サービスにリンクされたロールでサポートされているリージョン . 1638
パフォーマンスの最適化 ... 1639
一般的なパフォーマンスの問題 ... 1639
ハイ CPU ... 1639
接続の問題 ... 1641
クエリの実行が遅い ... 1648
アンチパターン ... 1653
複合インデックスのアンチパターンが 3 つ以上の属性 ... 1653
長時間実行されるクエリ ... 1655
低使用インデックスと冗長インデックス ... 1658
大きな配列を持つマルチキーインデックス ... 1661
パフォーマンス向上のヒント ... 1662
1. 集約パイプラインの最初のステージとして $match を使用する ... 1662
2. 集約パイプラインで $project を使用してパイプラインデータサイズを最小化する ... 1662
3. ドキュメント圧縮を有効にしてストレージコスト、I/O コストを削減し、クエリパフォーマンスを向上させる ... 1663
4. インデックスを活用してクエリパフォーマンスを最適化する ... 1663
5. クエリパターンに基づいてデータモデルを最適化する ... 1664
クエリプラン分析 ... 1666
全体的なタイミング ... 1668
実行ステージ ... 1669
Explain Plan Execution docsExamined について ... 1670
インデックスのみのスキャン ... 1678

Amazon DocumentDB がインデックスのみのスキャンを使用する場合 1679
対象クエリの要件 1681
制限事項 1682
インデックスのみのスキャンの確認 1682
ベストプラクティス 1684
キャッシュフィルターの計画 1684
サポートされているコマンド 1685
サポートされているクエリシェイプ 1686
フィルターの設定、一覧表示、クリア 1687
フィルターが適用されていることを確認する 1689
例 1690
ヒントとインデックスの変更を含む動作 1693
ベストプラクティス 1693
基本運用ガイドライン 1694
インスタンスのサイズ指定 1695
インデックスの使用 1697
セキュリティのベストプラクティス 1698
コスト最適化 1698
メトリクスを使用したパフォーマンスの問題の特定 1699
TTL および時系列ワークロード 1704
移行 1704
クラスターパラメータグループの使用 1705
集約パイプラインクエリ 1705
`batchInsert` および `batchUpdate` 1705
インデックスの管理 1705
Amazon DocumentDB インデックスの作成 1705
Amazon DocumentDB インデックスのメンテナンス 1712
ドキュメント圧縮の管理 1714
ドキュメント圧縮の管理 1715
ドキュメント圧縮のモニタリング 1717
ディクショナリベースの圧縮の管理 1717
パフォーマンスに関する考慮事項 1718
ディクショナリベースの圧縮の有効化 1719
開始方法 1720
モニタリング 1720
セキュリティ 1722

を使用したパスワード管理 AWS Secrets Manager 1723
Secrets Manager と Amazon DocumentDB の統合に関する制限事項 1723
を使用したプライマリユーザーパスワードの管理の概要 AWS Secrets Manager 1724
でのプライマリユーザーパスワードの Amazon DocumentDB 管理の強制 AWS Secrets Manager 1725
Secrets Manager によるクラスターのプライマリユーザーパスワードの管理 1725
データ保護 1726
クライアント側のフィールドレベルの暗号化 1727
保管中のデータの暗号化 1734
Encrypting data in transit 1742
キー管理 1753
Identity and Access Management 1753
オーディエンス 1754
アイデンティティを使用した認証 1754
ポリシーを使用したアクセスの管理 1755
Amazon DocumentDB と IAM との連携方法 1757
アイデンティティベースのポリシーの例 1764
トラブルシューティング 1767
Amazon DocumentDB リソースへのアクセス許可の管理 1769
アイデンティティベースのポリシー (IAM ポリシー) の使用 1774
AWS Amazon DocumentDB の マネージドポリシー 1779
Amazon DocumentDB API 許可リファレンス 1798
サービス間での不分別な代理処理の防止 1807
IAM ID を使用した認証 1808
IAM ユーザーとロールの開始方法 1809
コンピューティングの設定 1812
認証リクエストのモニタリング 1813
IAM 認証の使用 1813
IAM をサポートするドライバー 1814
IAM ID 認証に関するよくある質問 1814
SCRAM-SHA-256 を使用した認証 1815
ユーザーの作成 1815
ユーザーの更新 1816
認証 1817
Amazon DocumentDB ユーザーの管理 1817
プライマリおよび `serviceadmin` ユーザ 1818

追加ユーザーの作成 1818
パスワードの自動ローテーション 1820
ロールベースのアクセスコントロール 1821
RBAC の概念 1822
RBAC 組み込みロールの開始方法 1824
RBAC ユーザー定義ロールの開始方法 1827
ユーザーとして Amazon DocumentDB に接続する 1832
一般的なコマンド 1833
機能的な違い 1838
制限 1839
ロールベースのアクセス制御 (RBAC) を使用したデータベースへのアクセス 1839
証明書の更新 1848
アプリケーションと Amazon DocumentDB クラスターの更新 1849
サーバー証明書の自動ローテーション 1853
トラブルシューティング 1854
よくある質問 1854
コンプライアンス検証 1861
耐障害性 1862
インフラストラクチャセキュリティ 1863
VPC エンドポイント (AWS PrivateLink) 1864
VPC エンドポイントに関する考慮事項 1865
リージョンの可用性 1865
Amazon DocumentDB API 用のインターフェイス VPC エンドポイントの作成 1866
Amazon DocumentDB API の VPC エンドポイントポリシーの作成 1867
セキュリティのベストプラクティス 1868
監査イベント 1869
サポートされるイベント 1870
監査を有効にする 1875
監査を無効にする 1882
イベントへのアクセス 1885
DML イベントをフィルタリングする 1885
Amazon VPC と Amazon DocumentDB 1891
VPC 内の DocumentDB クラスター 1891
VPC でのクラスターへのアクセス。 1904
クラスターの IPv4-only VPC を作成する 1907
クラスターのデュアルスタック VPC を作成する 1914

バックアップと復元 1925
バックアップと復元: 概念 1926
バックアップストレージの使用状況を確認する 1928
データのダンプ、復元、インポート、およびエクスポート 1929
mongodump 1930
mongorestore 1930
mongoexport 1931
mongoimport 1931
チュートリアル 1932
クラスタースナップショットに関する考慮事項 1935
バックアップストレージ 1935
バックアップウィンドウ 1936
バックアップの保存期間 1938
クラスタースナップショットの暗号化をコピーする 1938
自動スナップショットと手動スナップショットの比較 1939
手動クラスタースナップショットの作成 1941
クラスタースナップショットのコピー 1944
共有スナップショットのコピー 1945
間でのスナップショットのコピー AWS リージョン 1945
制限事項 1945
暗号化の処理 1946
パラメータグループに関する考慮事項 1946
クラスタースナップショットのコピー 1947
クラスターのスナップショットの共有 1956
暗号化されたスナップショットの共有 1957
スナップショットの共有 1960
クラスタースナップショットからの復元 1964
特定時点への復元 1971
クラスタースナップショットの削除 1978
モニタリングと口グ記録 1980
クラスターのステータスのモニタリング 1981
クラスターステータスの値 1982
クラスターのステータスのモニタリング 1983
インスタンスのステータスのモニタリング 1985
インスタンスのステータス値 1986

AWS マネジメントコンソール または を使用したインスタンスのステータスのモニタリング AWS CLI ... 1988
インスタンスのヘルスステータス値 ... 1990
を使用したインスタンスのヘルスステータスのモニタリング AWS マネジメントコンソール ... 1990
Amazon DocumentDB に関する推奨事項の表示 ... 1992
イベントサブスクリプション ... 1995
イベントのサブスクライブ ... 1996
サブスクリプションを管理する ... 1998
カテゴリとメッセージ ... 2002
Amazon DocumentDB と CloudWatch のモニタリング ... 2005
Amazon DocumentDB のメトリクス ... 2005
CloudWatch データの表示 ... 2024
Amazon DocumentDB ディメンション ... 2030
opcounter メトリクスのモニタリング ... 2031
データベース接続のモニタリング ... 2031
CloudTrail を使用した Amazon DocumentDB API コールのログ記録 ... 2031
CloudTrail 内の Amazon DocumentDB 情報 ... 2032
オペレーションのプロファイリング ... 2032
サポートされているオペレーション ... 2033
制限事項 ... 2034
プロファイラーの有効化 ... 2034
プロファイラーの無効化 ... 2039
プロファイラーログのエクスポートの無効化 ... 2040
プロファイラーのログへのアクセス ... 2042
一般的なクエリ ... 2043
Performance Insights を使用したモニタリング ... 2043
Performance Insights の概要 ... 2045
Performance Insights の有効化と無効化 ... 2049
Performance Insights 用のアクセスポリシーの設定 ... 2052
Performance Insights ダッシュボードを使用してメトリクスを分析する ... 2056
Performance Insights API によるメトリクスの取得 ... 2078
Performance Insights の Amazon CloudWatch メトリクス ... 2093
カウンターメトリクス用の Performance Insights ... 2095
サーバーレス ... 2097
サーバーレスユースケース ... 2097

既存のプロビジョニングされたワークロードに Amazon DocumentDB Serverless を使用する ... 2099
Amazon DocumentDB Serverless の利点 ... 2099
サーバーレスの仕組み ... 2101
概要 ... 2101
Amazon DocumentDB クラスターの設定 ... 2102
Amazon DocumentDB Serverless スケーリング容量 ... 2103
Amazon DocumentDB Serverless スケーリング ... 2104
アイドル状態 (0.5 DCU) ... 2107
サーバーレスの要件と制限 ... 2107
機能の要件 ... 2107
サーバーレスを使用するクラスターの作成 ... 2109
Amazon DocumentDB Serverless クラスターの作成 ... 2109
Amazon DocumentDB Serverless インスタンスの追加 ... 2112
サーバーレスへの移行 ... 2113
既存の DocumentDB クラスターをサーバーレスに移行する ... 2113
MongoDB から DocumentDB サーバーレスへの移行 ... 2118
サーバーレスの管理 ... 2118
クラスターのスケーリング容量範囲設定の表示と変更 ... 2118
サーバーレスリーダーの昇格階層の表示と変更 ... 2122
サーバーレスインスタンスの制限 ... 2124
サーバーレススケーリング設定 ... 2126
DocumentDB サーバーレスクラスターのスケーリング容量範囲の選択 ... 2127
DocumentDB サーバーレスクラスターの MinCapacity 設定の選択 ... 2127
DocumentDB サーバーレスクラスターの MaxCapacity 設定の選択 ... 2129
メモリ不足エラーを回避する ... 2131
サーバーレスインスタンスがスケールダウンしないのはなぜですか? ... 2132
サーバーレスのモニタリング ... 2133
メモリ不足: 互換性のないパラメータステータス ... 2134
DocumentDB サーバーレスの Amazon CloudWatch メトリクス ... 2134
パフォーマンスインサイトによる DocumentDB サーバーレスパフォーマンスのモニタリング ... 2138
エラスティッククラスター ... 2140
エラスティッククラスターのユースケース ... 2141
ユーザープロファイル ... 2141
コンテンツの管理と履歴の記録 ... 2141

エラスティッククラスターの利点 2141
AWS サービス統合 2141
リージョンとバージョンの可用性 2142
Elastic クラスターのリージョン可用性 2142
バージョンの可用性 2143
制限事項 2143
エラスティッククラスターの管理 2143
クエリ操作と書き込み操作 2144
コレクションとインデックスの管理 2144
管理と診断 2144
オプトイン機能 2145
仕組み 2145
Amazon DocumentDB エラスティッククラスターのシャーディング 2145
エラスティッククラスターの移行 2149
エラスティッククラスターのスケーリング 2149
エラスティッククラスターの信頼性 2149
エラスティッククラスターのストレージと可用性 2149
Amazon DocumentDB 4.0 と Elastic クラスター の機能上の違い 2150
はじめに 2151
前提条件 2152
ステップ 1: エラスティッククラスターを作成する 2153
ステップ 2: エラスティッククラスターに接続する 2159
ステップ 3: コレクションをシャーディングし、データを挿入してクエリする 2161
ステップ 4: 探索する 2164
ベストプラクティス 2164
シャードキーの選択 2165
接続管理 2165
非シャード型コレクション 2165
エラスティッククラスターのスケーリング 2166
エラスティッククラスターのモニタリング 2166
Elastic クラスターの管理 2167
Elastic クラスター設定の変更 2167
Elastic クラスターのモニタリング 2171
Elastic クラスターの削除 2175
Elastic クラススナップショットの管理 2177
Elastic クラスターの停止と開始 2192

エラスティッククラスターのメンテナンス ... 2196
保管時のデータ暗号化 ... 2204
Amazon DocumentDB エラスティッククラスターが で許可を使用する方法 AWS KMS ... 2206
カスタマーマネージドキーを作成する ... 2207
Amazon DocumentDB エラスティック クラスター の暗号化キーのモニタリング ... 2208
詳細情報 ... 2214
サービスリンクロール ... 2214
エラスティッククラスターに対するサービスにリンクされたロールのアクセス許可 ... 2215
クォータ ... 2219
サポートされるインスタンスタイプ ... 2219
サポート対象のリージョン ... 2221
リージョン別クォータ ... 2223
集計クォータ ... 2226
クラスタークォータ ... 2226
インスタンスクォータ ... 2228
命名に関する制約 ... 2235
TTL の制約 ... 2237
Elastic クラスターのクォータ ... 2237
Elastic クラスターシャードクォータ ... 2239
シャードあたりのエラスティッククラスター CPU、メモリ、接続、カーソルのクォータ ... 2239
トラブルシューティング ... 2241
接続の問題 ... 2241
Amazon DocumentDB エンドポイントに接続できない ... 2241
Amazon DocumentDB インスタンスへの接続のテスト ... 2247
無効なエンドポイントへの接続 ... 2247
接続数に影響するドライバー設定 ... 2248
インデックス ... 2248
インデックスの構築が失敗する ... 2248
バックグラウンドインデックスビルドのレイテンシーの問題と失敗 ... 2249
インデックスの肥大化 ... 2249
パフォーマンスとリソース使用率 ... 2251
挿入、更新、削除に関する統計の表示 ... 2251
キャッシュのパフォーマンスの分析 ... 2253
長時間実行またはブロックされているクエリを検索して終了する ... 2254
クエリプランを表示してクエリを最適化する ... 2256
Elastic クラスターのクエリプランを確認する方法 ... 2258

インスタンスで実行中のすべてのオペレーションを一覧表示する 2260
クエリの進行状況を把握する 2263
システムが突然遅く実行される理由を特定する 2266
CPU 使用率が高くなる原因を特定する 2267
インスタンス上で開いているカーソルを検索します。 2268
Amazon DocumentDB エンジンの現在のバージョンを表示します 2268
インデックスの使用状況を分析し、未使用のインデックスを特定します。 2269
見逃したインデックスの特定 2271
データベースコレクションの肥大化を確認するにはどうすればよいですか? 2273
便利なクエリの概要 2274
ガベージコレクション 2275
Amazon DocumentDB のガベージコレクションについて 2276
ガベージコレクションプロセス 2276
ストレージアーキテクチャと拡張ストレージ 2277
ガベージコレクションのモニタリング 2278
collStats 出力の例 2281
よくある質問 2283
エラーコード 2286
リリースノート 2318
Amazon DocumentDB エンジンバージョンのリリースノート 2319
Amazon DocumentDB の一般的なリリースノート 2381
Amazon DocumentDB エラスティッククラスターのリリースノート 2388
ドキュメント履歴 2392
..... mmcdv

Amazon DocumentDB (MongoDB 互換) とは

Amazon DocumentDB (MongoDB 互換) は高速で信頼性が高く、完全マネージド型のデータベースサービスです。Amazon DocumentDB は、クラウド内で MongoDB と互換性のあるデータベースを簡単にセットアップ、運用、スケールすることができます。Amazon DocumentDB では、同じアプリケーションコードを実行し、MongoDB で使用するのと同じドライバーとツールを使用できます。

Amazon DocumentDB を使用する前に、「仕組み」で説明されている概念と機能を確認しておく必要があります。その後で、「入門ガイド」の各ステップを実行します。

トピック

- Amazon DocumentDB の概要
- クラスター
- インスタンス
- リージョンとアベイラビリティーゾーン
- Amazon DocumentDB の料金
- モニタリング
- インターフェイス
- 次のステップ
- Amazon DocumentDB の 仕組み
- ドキュメントデータベースとは

Amazon DocumentDB の概要

次に示すのは Amazon DocumentDB の主な機能の一部です。

- Amazon DocumentDB は、インスタンスベースのクラスターとエラスティッククラスターの 2 種類のクラスターをサポートしています。エラスティッククラスターは、1 秒あたり数百万回の読み取り/書き込みとペタバイトのストレージ容量を持つワークロードをサポートします。エラスティッククラスターの詳細については、「Amazon DocumentDB Elastic クラスターの使用」を参照してください。以下の内容は Amazon DocumentDB インスタンスベースのクラスターに関するものです。
- Amazon DocumentDB は、データベースのストレージニーズが増大すると自動的にストレージボリュームのサイズを拡張します。ストレージボリュームは 10 GB 単位で増加し、エンジンバー

ジョン 8.0 以降では最大 256 TiB (以前のエンジンバージョンでは 128 TiB) まで増加します。将来の拡張に備えてクラスターに余分なストレージをプロビジョニングする必要はありません。

- Amazon DocumentDB では、最大 15 個のレプリカインスタンスを作成して、大量のアプリケーションリクエストをサポートように読み取りスループットを増やすことができます。Amazon DocumentDB レプリカは同じ基盤となるストレージを共有しているため、コストを削減でき、レプリカノードで書き込みを実行する必要はありません。この機能により、読み取り要求を処理するための処理能力が解放され、レプリカのラグタイム (多くの場合、一桁ミリ秒に短縮されます) が短縮されます。ストレージボリュームのサイズに関係なく、レプリカを数分で追加できます。Amazon DocumentDB にはリーダーエンドポイントも提供するため、アプリケーションは追加および削除されるときにレプリカを追跡することなく接続できます。
- Amazon DocumentDB では、各インスタンスを拡大または縮小するため、コンピューティングおよびメモリリソースをスケールすることができます。通常、コンピューティングのスケーリングは数分以内に完了します。
- Amazon DocumentDB は Amazon VPC (Amazon Virtual Private Cloud) で実行されるため、独自の仮想ネットワークでデータベースを分離することができます。また、ファイアウォール設定を指定して、クラスターへのネットワークアクセスを制御することができます。
- Amazon DocumentDB はクラスターの正常性を継続的にモニタリングします。インスタンスに障害が発生すると、Amazon DocumentDB はインスタンスと関連するプロセスを自動的に再起動します。Amazon DocumentDB では、データベース REDO ログのクラッシュリカバリのリプレイが必要ないため、再起動時間が大幅に短縮されます。Amazon DocumentDB は、データベースキャッシュをデータベースプロセスから分離し、インスタンスの再起動後もキャッシュを有効にします。
- インスタンスに障害が発生した場合、Amazon DocumentDB は、他のアベイラビリティーゾーンで作成する最大 15 個の Amazon DocumentDB レプリカの 1 つに自動的にフェイルオーバーします。レプリカがプロビジョニングされていない場合に障害が発生すると、Amazon DocumentDB は自動的に新しい Amazon DocumentDB インスタンスの作成を試みます。
- Amazon DocumentDB のバックアップ機能により、クラスターのポイントインタイムリカバリが可能です。この機能によって、最大 5 分前まで、保持期間内の任意の時点にクラスターを復元させることができます。自動バックアップ保持期間は、最大 35 日間まで設定できます。自動化されたバックアップは、99.999999999% の耐久性を持つように設計された Amazon S3 (Amazon Simple Storage Service) に保存されます。Amazon DocumentDB バックアップは自動、増分、継続的であり、クラスターのパフォーマンスへの影響はありません。
- Amazon DocumentDB では、 AWS Key Management Service () を使用して作成および制御するキーを使用してデータベースを暗号化できますAWS KMS。Amazon DocumentDB 暗号化を使用して実行されているデータベースクラスターでは、基盤となるストレージに保存されている保管中の

データは暗号化されます。同じクラスター内の自動バックアップ、スナップショット、およびレプリカも暗号化されます。

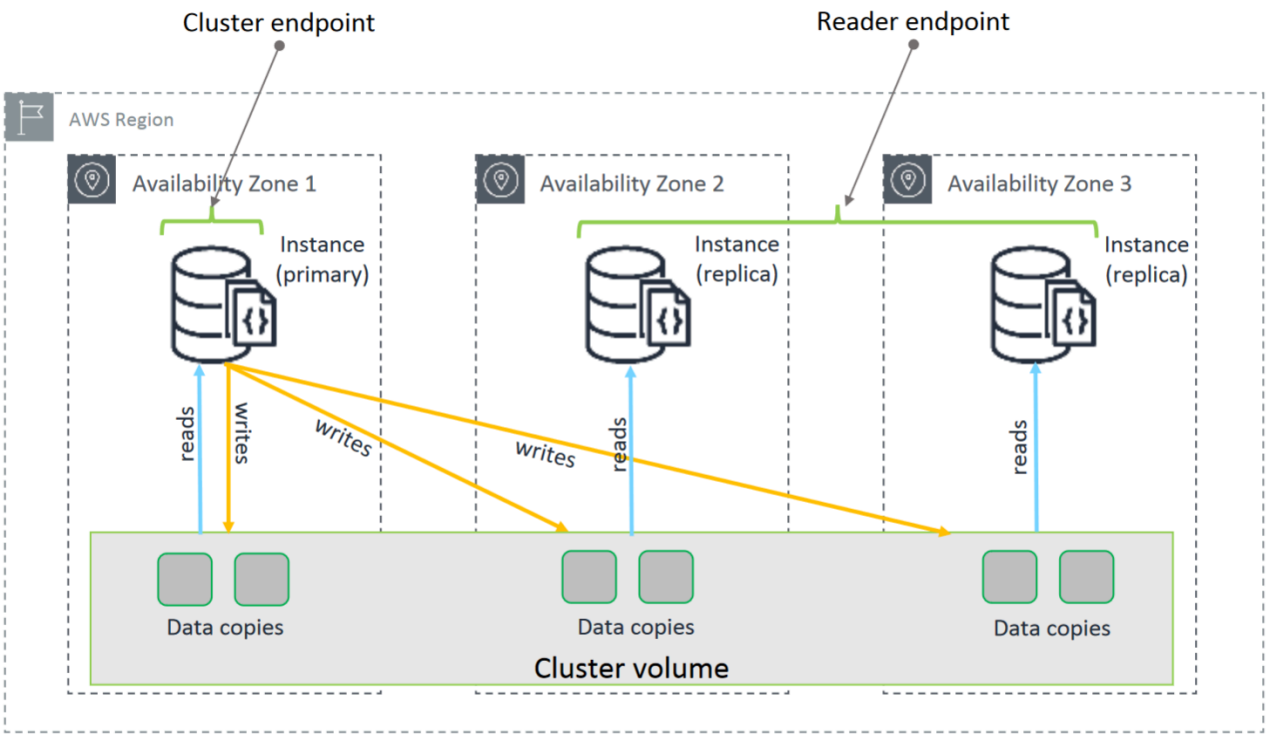
- Amazon DocumentDB は、「連邦リスク承認管理プログラム (FedRAMP)」の認可事業者です。これには、 AWS GovCloud (米国) リージョンの FedRAMP High 認可と AWS 、米国東部/西部リージョンの FedRAMP Moderate 認可があります。 AWS とコンプライアンスの取り組みの詳細については、AWS 「コンプライアンスプログラムによる対象範囲内のサービス」を参照してください。

AWS サービスを初めて使用する場合は、次のリソースを使用して詳細を確認してください。

- AWS は、コンピューティング、データベース、ストレージ、分析、その他の機能のためのサービスを提供します。すべての AWS サービスの概要については、「Amazon Web Services によるクラウドコンピューティング」を参照してください。
- AWS は、多数のデータベースサービスを提供します。使用している環境に最適なサービスに関するガイダンスについては、「AWSでのデータベース」を参照してください。

クラスター

クラスター は、0 ~ 16 のインスタンスと、これらのインスタンスのデータを管理する 1 つのクラスターストレージボリュームで構成されます。すべての書き込みはプライマリインスタンスを介して行われます。すべてのインスタンス (プライマリとレプリカ) は読み込みをサポートします。クラスターのデータはクラスターボリュームに保存され、3 つの異なるアベイラビリティーゾーンにコピーが保存されます。



Amazon DocumentDB 5.0 インスタンスベースのクラスターは、Amazon DocumentDB 標準と Amazon DocumentDB I/O 最適化の 2 つの Amazon DocumentDB ストレージ設定をサポートしています。詳細については、「Amazon DocumentDB クラスターストレージ設定」を参照してください。

インスタンス

Amazon DocumentDB インスタンスは、クラウド内の独立したデータベース環境です。インスタンスには、複数のユーザーが作成したデータベースを含めることができます。または を使用してインスタンスを作成および変更できます AWS マネジメントコンソール AWS CLI。

インスタンスの計算とメモリの容量は、インスタンスクラス によって決まります。お客様のニーズに最も合うインスタンスを選択できます。時間の経過とともにニーズが変化した場合は、別のインスタンスクラスを選択できます。インスタンスクラスの仕様については、「インスタンスクラスの仕様」を参照してください。

Amazon DocumentDB インスタンスは、Amazon VPC 環境でのみ実行されます。Amazon VPC により仮想ネットワーク環境を管理できます。独自の IP アドレスの範囲を選択し、サブネットを作成して、ルーティングおよびアクセスコントロールリスト (ACL) を設定できます。

Amazon DocumentDB インスタンスを作成する前に、クラスターを作成してインスタンスを含める必要があります。

すべてのインスタンスクラスがすべての でサポートされているわけではありません AWS リージョン。次の表は、各リージョンでサポートされているインスタンスクラスを示しています。

> **Note**
>
> 各インスタンスクラスの Amazon DocumentDB でサポートされているインスタンスタイプの詳細なリストについては、「インスタンスクラスの仕様」を参照してください。

リージョン別のサポートされるインスタンスクラス

リージョン	R8G	R6GD	R6G	R5	R4	T4G	T3	サーバーレス
米国東部 (オハイオ)	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート
米国東部 (バージニア北部)	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート
米国西部 (オレゴン)	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート
アフリカ (ケープタウン)			サポート対象	サポート対象		サポート対象	サポート対象	サポート
南米 (サンパウロ)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート

リージョン	R8G	R6GD	R6G	R5	R4	T4G	T3	サーバーレス
アジアパシフィック (香港)	サポート対象		サポート対象	サポート対象		サポート対象	サポート対象	サポート
アジアパシフィック (ハイデラバード)			サポート対象	サポート対象		サポート対象	サポート対象	サポート
アジアパシフィック (マレーシア)			サポート対象			サポート対象	サポート対象	サポート
アジアパシフィック (ムンバイ)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート
アジアパシフィック (大阪)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート
アジアパシフィック (ソウル)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート
アジアパシフィック (シドニー)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート
アジアパシフィック (ジャカルタ)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート
アジアパシフィック (メルボルン)			サポート対象	サポート対象		サポート対象	サポート対象	サポート

リージョン	R8G	R6GD	R6G	R5	R4	T4G	T3	サーバーレス
アジアパシフィック (シンガポール)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
アジアパシフィック (タイ)			サポート対象			サポート対象	サポート対象	サポート対象
アジアパシフィック (東京)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
カナダ (中部)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート
カナダ西部 (カルガリー)			サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
欧州 (フランクフルト)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート
欧州 (チューリッヒ)		サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
欧州 (アイルランド)	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象
欧州 (ロンドン)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象

リージョン	R8G	R6GD	R6G	R5	R4	T4G	T3	サーバーレス
欧州 (ミラノ)			サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
欧州 (パリ)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
欧州 (スペイン)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
欧州 (ストックホルム)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
メキシコ (中部)			サポート対象			サポート対象	サポート対象	サポート対象
中東 (アラブ首長国連邦)			サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
中国 (北京)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
中国 (寧夏)	サポート対象		サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
イスラエル (テルアビブ)			サポート対象	サポート対象		サポート対象	サポート対象	サポート

リージョン	R8G	R6GD	R6G	R5	R4	T4G	T3	サーバーレス
AWS GovCloud (米国西部)	サポート対象	サポート対象	サポート対象	サポート対象			サポート対象	サポート
AWS GovCloud (米国東部)		サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート

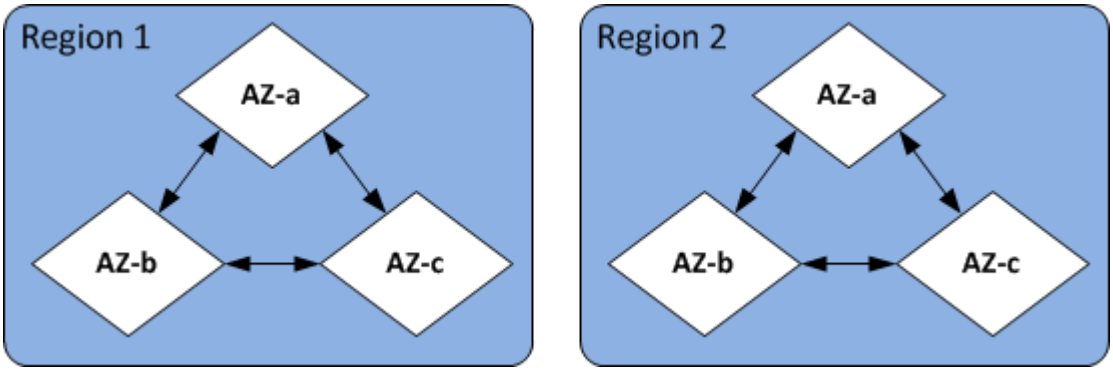
リージョンとアベイラビリティーゾーン

AWS リージョン および アベイラビリティーゾーンは、クラスターとインスタンスの物理的な場所を定義します。

リージョン

AWS クラウドコンピューティングリソースは、世界中のさまざまな地域 (北米、欧州、アジアなど) にある高可用性データセンター施設に収容されています。各データセンターの場所は、AWS リージョンと呼ばれます。

各 AWS リージョン は、他の から完全に分離されるように設計されています AWS リージョン。各リージョン内には複数のアベイラビリティーゾーンがあります。別のアベイラビリティーゾーンでノードを起動して、最大限の耐障害性を実現できます。次の図は、 AWS リージョン とアベイラビリティーゾーンの仕組みの概要を示しています。



アベイラビリティーゾーン

各 AWS リージョン には、アベイラビリティーゾーンと呼ばれる複数の異なる場所が含まれています。各アベイラビリティーゾーンは、他のアベイラビリティーゾーンにおける障害の影響は受けず、同じリージョン内の他のアベイラビリティーゾーンには、低コスト、低レイテンシーでネットワーク接続できるように設計されています。複数のアベイラビリティーゾーンで特定のクラスターのインスタンスを起動することにより、アベイラビリティーゾーンが失敗するというまれなイベントからアプリケーションを保護できます。

Amazon DocumentDB アーキテクチャは、ストレージとコンピューティングを分離します。ストレージレイヤーの場合、Amazon DocumentDB は 3 つの AWS アベイラビリティーゾーンにデータの 6 つのコピーをレプリケートします。例えば、2 つのアベイラビリティーゾーンのみをサポートするリージョンで Amazon DocumentDB クラスターを起動している場合、データストレージは 3 つのアベイラビリティーゾーンにわたって 6 つの方法でレプリケートされますが、コンピューティングインスタンスは 2 つのアベイラビリティーゾーンでしか使用できません。

次の表に、特定の でクラスターのコンピューティングインスタンスをプロビジョニング AWS リージョン するために使用できるアベイラビリティーゾーンの数を示します。

リージョン名	リージョン	アベイラビリティーゾーン (コンピューティング)
米国東部 (オハイオ)	us-east-2	3
米国東部 (バージニア北部)	us-east-1	6
米国西部 (オレゴン)	us-west-2	4
アフリカ (ケープタウン)	af-south-1	3
南米 (サンパウロ)	sa-east-1	3
アジアパシフィック (香港)	ap-east-1	3
アジアパシフィック (ハイデラバード)	ap-south-2	3

リージョン名	リージョン	アベイラビリティーゾーン (コンピューティング)
アジアパシフィック (マレーシア)	ap-southeast-5	3
アジアパシフィック (ムンバイ)	ap-south-1	3
アジアパシフィック (大阪)	ap-northeast-3	3
アジアパシフィック (ソウル)	ap-northeast-2	4
アジアパシフィック (シンガポール)	ap-southeast-1	3
アジアパシフィック (シドニー)	ap-southeast-2	3
アジアパシフィック (ジャカルタ)	ap-southeast-3	3
アジアパシフィック (メルボルン)	ap-southeast-4	3
アジアパシフィック (タイ)	ap-southeast-7	3
アジアパシフィック (東京)	ap-northeast-1	3
カナダ (中部)	ca-central-1	3
カナダ西部 (カルガリー)	ca-west-1	3
中国 (北京) リージョン	cn-north-1	3

リージョン名	リージョン	アベイラビリティーゾーン (コンピューティング)
中国 (寧夏)	cn-northwest-1	3
欧州 (フランクフルト)	eu-central-1	3
欧州 (チューリッヒ)	eu-central-2	3
欧州 (アイルランド)	eu-west-1	3
欧州 (ロンドン)	eu-west-2	3
欧州 (ミラノ)	eu-south-1	3
欧州 (パリ)	eu-west-3	3
欧州 (スペイン)	eu-south-2	3
欧州 (ストックホルム)	eu-north-1	3
メキシコ (中部)	mx-central-1	3
中東 (アラブ首長国連邦)	me-central-1	3
イスラエル (テルアビブ)	il-central-1	3
AWS GovCloud (米国西部)	us-gov-west-1	3
AWS GovCloud (米国東部)	us-gov-east-1	3

Amazon DocumentDB の料金

Amazon DocumentDB クラスターは、以下のコンポーネントに基づいて請求されます。

- インスタンス時間 (1 時間あたり) - インスタンスのインスタンスクラスに基づきます (例: `db.r5.xlarge`)。料金は 1 時間単位で表示されますが、請求の計算方法には秒単位が適用され、時間は 10 進形式で表示されます。Amazon DocumentDB の使用料は 1 秒ごとに課金され、10 分未満の場合は 10 分の料金が発生します。詳細については、「インスタンスクラスの管理」を参照してください。
- I/O リクエスト (1 か月あたりの 100 万リクエストごと) - 請求サイクル内で行ったストレージ I/O リクエストの合計数。
- バックアップストレージ (1 か月当たりの GiB) - バックアップストレージは、自動データベースバックアップおよび作成したアクティブなデータベースのスナップショットに関連付けられているストレージです。バックアップ保持期間を延長するか、追加のデータベーススナップショットを撮ると、データベースが消費するバックアップストレージが増加します。バックアップストレージは GB 月単位で計算され、1 秒単位は適用されません。詳細については、「Amazon DocumentDB でのバックアップと復元」を参照してください。
- データ転送 (GB あたり) — インスタンスとの間で送受信されるデータ転送 AWS リージョン。

詳細については、「Amazon DocumentDB の料金」を参照してください。

無料トライアル

Amazon DocumentDB は、1 か月間の無料トライアルを使用して無料でお試しいただけます。詳細については、Amazon DocumentDB 料金表サイトの無料トライアル」またはAmazon DocumentDB 無料トライアルに関するよくある質問」を参照してください。

モニタリング

インスタンスのパフォーマンスと正常性を追跡する方法は複数あります。無料の Amazon CloudWatch サービスを使用して、インスタンスのパフォーマンスと正常性をモニタリングできます。Amazon DocumentDB コンソールでパフォーマンスチャートを見つけることができます。インスタンス、スナップショット、パラメータグループ、セキュリティグループで変更が発生したときに通知される Amazon DocumentDB のイベントにサブスクライブできます。

詳細については次を参照してください:

- Amazon DocumentDB と CloudWatch のモニタリング
- を使用した Amazon DocumentDB API コールのログ記録 AWS CloudTrail

インターフェイス

Amazon DocumentDB を操作するには、 AWS マネジメントコンソール や など、複数の方法があります AWS CLI。

AWS マネジメントコンソール

AWS マネジメントコンソール はシンプルなウェブベースのユーザーインターフェイスです。プログラミングなしでコンソールからクラスターとインスタンスを管理できます。Amazon DocumentDB コンソールにアクセスするには、 にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。

AWS CLI

AWS Command Line Interface (AWS CLI) を使用して、Amazon DocumentDB クラスターとインスタンスを管理できます。最小限の設定で、使い慣れたターミナルプログラムから、Amazon DocumentDB コンソールで提供されるすべての機能の使用を開始できます。

- をインストールするには AWS CLI、AWS 「 コマンドラインインターフェイスのインストール」を参照してください。
- Amazon DocumentDB AWS CLI の の使用を開始するには、AWS Amazon DocumentDB のコマンドラインインターフェイスリファレンス」を参照してください。

MongoDB ドライバー

Amazon DocumentDB クラスターに対するアプリケーションの開発と作成に、MongoDB ドライバーを Amazon DocumentDB で使用することもできます。詳細については、 TLS が有効な場合の接続 または TLS が無効な場合の接続 の MongoDB シェルタブを参照してください。

次のステップ

前のセクションでは、Amazon DocumentDB が提供する基本的なインフラストラクチャのコンポーネントを紹介しました。次に実行すべきこと 状況に応じて、以下のトピックの 1 つを参照し、使用を開始してください。

- を使用してクラスターとインスタンスを作成して、Amazon DocumentDB の使用を開始します CloudFormation を使用した Amazon DocumentDB クイックスタート CloudFormation。

- 「入門ガイド」の手順を使用してクラスターとインスタンスを作成することにより、Amazon DocumentDB の使用を開始します。
- Amazon DocumentDB エラスティッククラスターの開始方法 の手順を使用してエラスティッククラスターを作成することにより、Amazon DocumentDB を開始します。
- Amazon DocumentDB の移行とアップグレード のガイダンスを使用して、MongoDB 実装を Amazon DocumentDB に移行します。

Amazon DocumentDB の 仕組み

Amazon DocumentDB (MongoDB 互換) は、フルマネージドの MongoDB と互換性のあるデータベースサービスです。Amazon DocumentDB では、MongoDB と同じアプリケーションコードを実行し、同じドライバーとツールを使用できます。Amazon DocumentDB は、MongoDB 3.6、4.0、5.0、および 8.0 と互換性があります。

トピック

- Amazon DocumentDB バージョン
- Amazon DocumentDB エンドポイント
- TLS サポート
- Amazon DocumentDB のストレージ
- Amazon DocumentDB レプリケーション
- Amazon DocumentDB の信頼性
- 読み込み設定のオプション
- TTL 削除
- 有料リソース

Amazon DocumentDB バージョン

> **Note**
>
> Amazon DocumentDB 5.0 以降にはマイナーバージョン番号が適用されます。Amazon DocumentDB 3.6 および 4.0 は、エンジンパッチを持つメジャーバージョンのみを使用します。

Amazon DocumentDB 5.0 以降、エンジンバージョンは major.major.minor.patch の 4 レベルの番号付けスキームを使用します。

- メジャーバージョン — バージョンの最初の 2 つの数字 (5.0 など) で識別されます。メジャーバージョンは、MongoDB の互換性レベルを表します。メジャーバージョンでは、新機能、パフォーマンスの向上、MongoDB 互換性の更新が導入されています。メジャーバージョン (4.0 から 5.0 など) 間でアップグレードするには、メジャーバージョンアップグレード (MVU) が必要です。詳細については、「Amazon DocumentDB のインプレースメジャーバージョンアップグレード」を参照してください。
- マイナーバージョン - バージョン 3 番目の番号で識別されます (例: 5.0.1 1 の)。マイナーバージョンには、同じメジャーバージョン内の新機能、セキュリティ修正、バグ修正などの改善が含まれています。新しいマイナーバージョンに手動でアップグレードできます。詳細については、「Amazon DocumentDB マイナーバージョンアップグレード」を参照してください。
- パッチバージョン — エンジンパッチはマイナーバージョン内に適用され、重要なセキュリティ修正とバグ修正が含まれています。パッチバージョンはエンジンパッチバージョン (など3.0.17983) によって追跡され、 を実行して確認できますdb.runCommand({getEngineVersion: 1})。パッチは、クラスターのメンテナンスウィンドウ中に適用されます。詳細については、「Amazon DocumentDB のメンテナンス」を参照してください。

新しいクラスターを作成するときに、使用可能なエンジンバージョンを指定できます。利用可能なバージョンとサポート日については、「」を参照してくださいAmazon DocumentDB エンジンバージョンのサポート日。

Amazon DocumentDB を使用するときは、最初に クラスター を作成します。クラスターは、ゼロ以上のデータベースインスタンスと、これらのインスタンスのデータを管理する 1 つのクラスターボリュームで構成されます。Amazon DocumentDB の クラスターボリューム は、複数のアベイラビリティーゾーンにまたがる仮想データベースストレージボリュームです。アベイラビリティーゾーンごとに、クラスターデータのコピーがあります。

Amazon DocumentDB クラスターは、主要な 2 つのコンポーネントで構成されています。

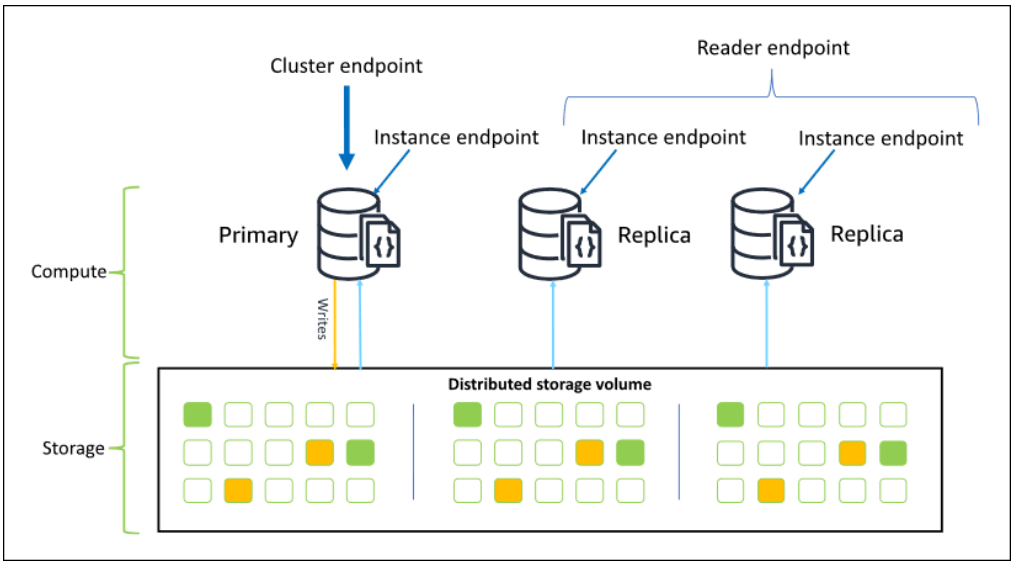
- クラスターボリューム - クラウドネイティブなストレージサービスを使用して、3 つのアベイラビリティーゾーンにわたって 6 つの方法でデータをレプリケートし、耐久性と可用性に優れたストレージを提供します。Amazon DocumentDB クラスターにはクラスターボリュームが 1 つだけあり、エンジンバージョン 8.0 以降では最大 256 TiB のデータを保存できます (以前のエンジンバージョンでは 128 TiB)。

- インスタンス - データベースの処理能力を提供し、クラスターストレージボリュームとの間でデータの書き込みと読み取りを行います。Amazon DocumentDB クラスターには 0 ~ 16 のインスタンスを持つことができます。

インスタンスは以下の 2 つのロールのいずれかを提供します。

- プライマリインスタンス - 読み書きオペレーションをサポートし、クラスターボリュームに対するすべてのデータ変更を実行します。各 Amazon DocumentDB クラスターには 1 つのプライマリインスタンスがあります。
- レプリカインスタンス - 読み取りオペレーションのみをサポートします。Amazon DocumentDB クラスターには、プライマリインスタンスに加えて 15 個までのレプリカを含めることができます。複数のレプリカがあると、読み取りワークロードを分散させることができます。さらに、別のアベイラビリティーゾーンにレプリカを配置することによって、クラスターの可用性を高めることもできます。

次の図は、Amazon DocumentDB クラスター内のクラスターボリューム、プライマリインスタンス、およびレプリカの関係を示しています。



クラスターインスタンスは、同じインスタンスクラスである必要はなく、必要に応じてプロビジョニングおよび終了することができます。このアーキテクチャでは、ストレージとは無関係にクラスターのコンピューティング性能をスケールできます。

アプリケーションがプライマリインスタンスにデータを書き込むときに、プライマリはクラスターボリュームに堅牢な書き込みを実行します。次に、その書き込み (データではありません) の状態がアクティブな各レプリカにレプリケートされます。Amazon DocumentDB レプリカは書き込み処

理に関与しないため、Amazon DocumentDB レプリカは読み取りスケーリングにとって有利となります。Amazon DocumentDB レプリカからの読み取りは、最短のレプリカラグで結果整合性があります。通常、プライマリインスタンスがデータを書き込んだ後、100 ミリ秒未満になります。レプリカからの読み込みは、プライマリに書き込まれた順序で読み込まれることが保証されます。レプリカラグは、データ変更のレートによって異なり、書き込みアクティビティが多い期間はレプリカラグが大きくなる可能性があります。詳細については、「Amazon DocumentDB のメトリクス」で `ReplicationLag` のメトリクスを参照してください。

Amazon DocumentDB エンドポイント

Amazon DocumentDB は複数の接続オプションを提供し、幅広いユースケースに対応します。Amazon DocumentDB クラスターのインスタンスに接続するには、インスタンスのエンドポイントを指定します。エンドポイントでは、ホストアドレスとポート番号がコロンで区切られています。

レプリカセットモードでクラスターエンドポイントを使用してクラスターに接続します (「」を参照レプリカセットとしての Amazon DocumentDB に接続する)。リーダーエンドポイントまたはインスタンスエンドポイントに接続する必要がある場合を除き、このアプローチを使用します。レプリカにリクエストをルーティングするには、アプリケーションの読み込み整合性要件に合わせて読み取りスケーリングを最大化する、ドライバー読み込み設定を選択します。`secondaryPreferred` 読み込み設定を使用すると、レプリカの読み取りを有効にし、プライマリインスタンスを解放してより多くの作業を行うことができます。

次のエンドポイントは、Amazon DocumentDB クラスターから取得できます。

クラスターエンドポイント

クラスターエンドポイントはクラスターの現在のプライマリインスタンスに接続します。このクラスターエンドポイントは、読み取りと書き込みの両方のオペレーションに使用できます。Amazon DocumentDB クラスターには、厳密には 1 つのクラスターエンドポイントがあります。

クラスターエンドポイントは、クラスターへの読み取り/書き込み接続のフェイルオーバーサポートを提供します。クラスターの現在のプライマリインスタンスが失敗し、クラスターに少なくとも 1 つのアクティブなリードレプリカがある場合、クラスターエンドポイントは自動的に接続リクエストを新しいプライマリインスタンスにリダイレクトします。Amazon DocumentDB クラスターに接続するときは、クラスターエンドポイントをレプリカセットモードで使用します (「」を参照レプリカセットとしての Amazon DocumentDB に接続する)。

Amazon DocumentDB クラスターエンドポイントの例を以下に示します。

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

次の接続文字列は、このクラスターエンドポイントを使用した例です。

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

クラスターのエンドポイントを見つける方法については、「クラスターのエンドポイントの検索」を参照してください。

リーダーエンドポイント

リーダーエンドポイントは、クラスター内の使用可能なすべてのレプリカ間で、読み取り専用接続の負荷分散を行います。クラスターリーダーエンドポイントは、replicaSet モードで接続する場合、クラスターエンドポイントとして機能します。つまり、接続文字列では、レプリカセットパラメータは &replicaSet=rs0 となります。この場合、プライマリに対して書き込みオペレーションを実行できます。ただし、directConnection=true を指定してクラスターに接続した後、リーダーエンドポイントへの接続を経由して書き込みオペレーションを実行しようとすると、エラーが発生します。Amazon DocumentDB クラスターには、厳密には 1 つのリーダーエンドポイントがあります。

クラスターに 1 つの (プライマリ) インスタンスのみが含まれている場合、リーダーみエンドポイントはプライマリインスタンスのみに接続します。Amazon DocumentDB クラスターにレプリカインスタンスを追加した場合、リーダーエンドポイントは、アクティブになった時点で新しいレプリカへの読み取り専用接続を開きます。

Amazon DocumentDB クラスターのリーダーエンドポイントの例を以下に示します。

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

次の接続文字列は、リーダーエンドポイントを使用した例です。

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

リーダーエンドポイントは、読み込みリクエストではなく、読み取り専用接続の負荷分散を行います。一部のリーダーエンドポイント接続が他のリーダーエンドポイント接続よりも多く使用されている場合、読み込みリクエストはクラスター内のインスタンス間で均等に分散されない可能性がありま

す。クラスターエンドポイントにレプリカセットとして接続し、secondaryPreferred 読み込み設定オプションを使用して、リクエストを分散することをお勧めします。

クラスターのエンドポイントを見つける方法については、「クラスターのエンドポイントの検索」を参照してください。

インスタンスエンドポイント

クラスター内で特定のインスタンスに接続するインスタンスエンドポイントです。現在のプライマリインスタンスのインスタンスエンドポイントは、読み込み/書き込みオペレーションに使用できます。ただし、リードレプリカのインスタンスエンドポイントに書き込みオペレーションを実行しようとすると、エラーが発生します。Amazon DocumentDB クラスターは、アクティブなインスタンスごとに 1 つのインスタンスのエンドポイントを持ちます。

インスタンスエンドポイントは、クラスターエンドポイントやリーダーエンドポイントが適切でないシナリオ向けに、特定のインスタンスへの接続の直接制御を提供します。ユースケースの例として、定期的な読み取り専用分析ワークロード用のプロビジョニングがあります。通常よりも大きいレプリカインスタンスのプロビジョニング、インスタンスエンドポイントを使用した新しいより大きなインスタンスへの直接接続、分析クエリの実行を行った後で、インスタンスを終了できます。インスタンスエンドポイントを使用すると、分析トラフィックが他のクラスターインスタンスに影響を与えるのを防ぐことできます。

次の例では、Amazon DocumentDB クラスターの 1 つのインスタンスのインスタンスエンドポイントを示します。

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

次の接続文字列は、このインスタンスエンドポイントを使用した例です。

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

> **Note**
>
> プライマリまたはレプリカとしてのインスタンスのロールは、フェイルオーバーイベントが原因で変更される可能性があります。アプリケーションで、特定のインスタンスエンドポイントがプライマリインスタンスであることを前提にしないでください。本番アプリケーションのインスタンスエンドポイントに接続しないでください。代わりに、クラスターエンドポイントをレプリカセットモードで使用します (「」を参照レプリカセットとしての Amazon

DocumentDB に接続する)。インスタンスのフェイルオーバー優先度をより高度に制御する方法については、「Amazon DocumentDB クラスターの耐障害性について理解する」を参照してください。

クラスターのエンドポイントを見つける方法については、「インスタンスのエンドポイントの検索」を参照してください。

レプリカセットモード

レプリカセット名 `rs0` を指定して、レプリカセットモードで Amazon DocumentDB クラスターエンドポイントに接続できます。レプリカセットモードで接続すると、[読み取り保証]、[書き込み保証]、および [読み取り設定] のオプションを指定できます。詳細については、「読み込み整合性」を参照してください。

レプリカセットモードで接続する接続文字列の例を次に示します。

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

レプリカセットモードで接続するときは、Amazon DocumentDB クラスターはドライバーとクライアントに対してレプリカセットとして表示されます。Amazon DocumentDB クラスターに対して追加、削除したインスタンスは、レプリカセット設定に自動的に反映されます。

各 Amazon DocumentDB クラスターは、デフォルト名が `rs0` の 1 つのレプリカセットで構成されます。レプリカセットの名前を変更することはできません。

レプリカセットモードでのクラスターエンドポイントへの接続は、一般的な使用に推奨される方法です。

> **Note**
>
> Amazon DocumentDB クラスター内のすべてのインスタンスは、同じ TCP ポートで接続をリッスンします。

TLS サポート

TLS (Transport Layer Security) を使用した Amazon DocumentDB への接続方法については、「Encrypting data in transit」を参照してください。

Amazon DocumentDB のストレージ

Amazon DocumentDB のデータは クラスターボリューム に保存されます。これは、SSD (Solid State Drive) を使用する単一の仮想ボリュームです。クラスターボリュームはお客様のデータの 6 つのコピーで構成されます。これらのデータは、単一の AWS リージョンの複数のアベイラビリティーゾーン間で自動的にレプリケートされます。このレプリケーションにより、データの高い耐久性が保証され、データ損失の可能性が低くなります。また、データのコピーが他のアベイラビリティーゾーンにすでに存在するため、フェイルオーバー中のクラスターの可用性が高まります。これらのコピーは、継続して Amazon DocumentDB クラスター内のインスタンスに対するデータリクエストを処理できます。

データストレージに対する請求方法

Amazon DocumentDB は、データ量が増えるにつれて、クラスターボリュームのサイズを自動的に増やします。Amazon DocumentDB クラスターボリュームは、エンジンバージョン 8.0 以降では最大サイズ 256 TiB (以前のエンジンバージョンでは 128 TiB) まで拡張できます。ただし、Amazon DocumentDB クラスターボリュームで使用するスペースに対してのみ課金されます。Amazon DocumentDB 4.0 以降では、コレクションやインデックスの削除などでのデータを削除すると、割り当てられた領域全体が相応に減少します。したがって、不要になったコレクション、インデックス、データベースを削除することで、ストレージ料金を削減できます。Amazon DocumentDB バージョン 3.6 では、データを削除したときに解放されたスペースをクラスターボリュームで再利用できますが、ボリューム自体のサイズが小さくなることはありません。バージョン 3.6 での結果として、解放された領域が再利用されていても、コレクションやインデックスを削除したときにストレージの変更が見られない場合があります。

> **Note**
>
> Amazon DocumentDB 3.6 では、ストレージコストは「高水準点 (high water mark)」 (任意の時点で Amazon DocumentDB クラスター用に割り当てられた最大量) に基づきます。大量の一時情報を作成したり、古い不要データの削除前に新規データを大量にロードする ETL プラクティスを避けることによってコストを管理できます。Amazon DocumentDB クラスターからデータを削除したことによって大量の割り当て済み未使用領域が発生した場合に、高水準点をリセットするには、`mongodump` や `mongorestore` などのツールを使用して論理データダンプを作成し、新しいクラスターに復元する必要があります。スナップショットを作成および復元しても、割り当てられたストレージは削減されません。これは、基になるストレージの物理的なレイアウトが、復元されたスナップショットでも変更されないためです。

> **Note**
>
> `mongodump` や `mongorestore` などのユーティリティを使用すると、ストレージボリュームに対して読み書きするデータのサイズに応じた I/O 料金が発生します。

Amazon DocumentDB データストレージおよび I/O の料金に関する情報は、「Amazon DocumentDB (MongoDB 互換性) の料金] と [料金のよくある質問」を参照してください。

Amazon DocumentDB レプリケーション

Amazon DocumentDB クラスターでは、各レプリカインスタンスが独立したエンドポイントを公開します。これらのレプリカエンドポイントは、クラスターボリュームのデータへの読み取り専用アクセスを提供します。また、複数のレプリケートされたインスタンス経由で、データの読み取りワークロードをスケールできます。さらに、データ読み取りのパフォーマンスを向上させ、Amazon DocumentDB クラスター内のデータの可用性を高めるためにも有効です。Amazon DocumentDB レプリカはフェイルオーバーターゲットでもあり、Amazon DocumentDB クラスターのプライマリインスタンスに障害が発生した場合は迅速に昇格されます。

Amazon DocumentDB の信頼性

Amazon DocumentDB は、信頼性、耐久性、および耐障害性を持つように設計されています。(可用性を高めるため、Amazon DocumentDB クラスターを設定し、異なるアベイラビリティーゾーンに複数のレプリカインスタンスを存在させる必要があります。) Amazon DocumentDB には、信頼性の高いデータベースソリューションとなるいくつかの自動機能も含まれています。

ストレージの自動修復

Amazon DocumentDB では、データの複数のコピーを 3 つのアベイラビリティーゾーンに保持し、ストレージの障害によってデータが失われる可能性を最小限に抑えます。Amazon DocumentDB は、クラスターボリュームの障害を自動的に検出します。クラスターボリュームのセグメントで障害が発生すると、Amazon DocumentDB はすぐにそのセグメントを修復します。クラスターボリュームを構成する他のボリュームからのデータを使用して、修復されたセグメントのデータが最新であるようにします。その結果、Amazon DocumentDB ではデータ損失が回避され、インスタンス障害から回復するためのポイントインタイム復元を実行する必要性が低減します。

存続できるキャッシュのウォームアップ

Amazon DocumentDB はそのページキャッシュをデータベースとは別のプロセスで管理するため、ページキャッシュはデータベースとは無関係に存続できます。予期できないデータベース障害が発生した場合でも、ページキャッシュはメモリに保持されます。これにより、バッファプールはデータベースの再起動時に最新の状態にウォームアップされます。

クラッシュ回復

Amazon DocumentDB は、クラッシュからほぼ瞬時に回復し、アプリケーションデータを提供し続けるように設計されています。Amazon DocumentDB は、クラッシュ回復を並列スレッドで非同期に実行します。これにより、クラッシュのほぼ直後にデータベースを開き、使用できるようにします。

リソースガバナンス

Amazon DocumentDB は、ヘルスチェックなどの重要なプロセスのリソースを保護します。インスタンスに高いメモリ負荷が発生すると、Amazon DocumentDB はリクエストを調整します。一部のオペレーションは、メモリ負荷が下がるまでキューに入れられる場合があります。メモリ負荷が続くと、キューに入れられたオペレーションがタイムアウトすることがあります。

、LowMemThrottleQueueDepth、LowMemThrottleMaxQueueDepth、 の CloudWatch メトリクスを使用してLowMemNumOperationsThrottled、低メモリスロットリングをモニタリングしますLowMemNumOperationsTimedOut。詳細については、「Amazon DocumentDB と CloudWatch のモニタリング」を参照してください。

持続的なメモリ負荷が発生した場合は、インスタンスをスケールアップしてワークロードに追加のメモリを提供します。

読み込み設定のオプション

Amazon DocumentDB は、3 つのアベイラビリティーゾーンにわたってデータを 6 回レプリケートするクラウドネイティブな共有ストレージサービスを使用して、高レベルの耐久性を提供します。Amazon DocumentDB は、耐久性を実現するためにデータを複数のインスタンスにレプリケートすることに依存しません。クラスターのデータは、1 つのインスタンスが含まれているか、15 個のインスタンスが含まれているかにかかわらず、耐久性に優れています。

トピック

- 書き込みの耐久性
- 読み込みの分離

- 読み込み整合性
- 高可用性
- 読み取りスケーリング

書き込みの耐久性

Amazon DocumentDB は、独自の分散型の耐障害性がある自己修復機能を備えたストレージシステムを使用します。このシステムは、データの 6 つのコピー (V=6) を 3 つの AWS アベイラビリティーゾーンにレプリケートして、高可用性と耐久性を実現します。Amazon DocumentDB はデータを書き込む際、クライアントへの書き込みを確認する前に、すべての書き込みが大半のノードに永続的にレコードされることを確認します。3 ノードの MongoDB レプリカセットを実行している場合は、書き込み確認として `{w:3, j:true}` を使用すると、Amazon DocumentDB と比較したときに可能な限り最高の設定を得られます。

Amazon DocumentDB クラスターへの書き込みは、クラスターのライターインスタンスによって処理される必要があります。リーダーに書き込もうとすると、エラーが発生します。Amazon DocumentDB プライマリインスタンスからの確認済みの書き込みは耐久性が高く、ロールバックすることはできません。Amazon DocumentDB はデフォルトで高い耐久性に優れており、耐久性に優れない書き込みオプションはサポートされていません。耐久性レベルを変更することはできません (これは、書き込みについてです)。Amazon DocumentDB は w=anything を無視し、事実上 w: 3 と j: true となります。この設定を減らすことはできません。

Amazon DocumentDB アーキテクチャではストレージとコンピューティングが分離されているため、単一インスタンスのクラスターは高い耐久性を持ちます。耐久性はストレージレイヤーで処理されます。その結果、1 つのインスタンスと 3 つのインスタンスを持つ Amazon DocumentDB クラスターは、同じレベルの耐久性を実現します。データの高い耐久性を保持しながら、具体的なユースケースに対してクラスターを設定できます。

Amazon DocumentDB クラスターへの書き込みは、1 つのドキュメント内でアトミックです。

Amazon DocumentDB は `wtimeout` オプションをサポートしておらず、値が指定された場合はエラーを返しません。プライマリ Amazon DocumentDB インスタンスへの書き込みは、無期限にブロックしないことが保証されます。

読み込みの分離

Amazon DocumentDB インスタンスからの読み取りでは、クエリが開始される前に、耐久性があるデータのみが返されます。読み込みで、クエリの実行が開始された後で変更されたデータが返されることはなく、どのような状況でもダーティーリードが可能になることもありません。

読み込み整合性

Amazon DocumentDB クラスターから読み取られたデータは耐久性があり、ロールバックされません。Amazon DocumentDB 読み取りの読み取り整合性は、リクエストまたは接続の読み取り設定を指定して変更できます。Amazon DocumentDB は、耐久性に優れない読み取りオプションをサポートしていません。

Amazon DocumentDB クラスターのプライマリインスタンスからの読み取りは、通常のオペレーション条件下では整合性が高く、書き込み後読み取りの整合性を実現します。書き込みとそれ以降の読み取りの間にフェイルオーバーイベントが発生した場合、システムは高い整合性を持たない読み取りを一時的に返すことがあります。リードレプリカからのすべての読み取りは、結果的に整合性を持ち、同じ順序でデータを返します。レプリカラグは多くの場合、100 ミリ秒未満です。

Amazon DocumentDB 読み取り設定

Amazon DocumentDB は、レプリカセットモードのクラスターエンドポイントからデータを読み取るときのみ、読み取り設定オプションをサポートします。読み取り設定オプションの設定により、MongoDB クライアントまたはドライバーが Amazon DocumentDB クラスターでインスタンスに読み取りリクエストをルーティングする方法に影響が生じます。特定のクエリ用に読み込み設定オプションを指定するか、MongoDB ドライバーの全般オプションとして指定できます。(読み込み設定オプションの設定方法については、クライアントまたはドライバーのドキュメントを参照してください。)

クライアントまたはドライバーがレプリカセットモードの Amazon DocumentDB クラスターエンドポイントに接続していない場合、読み取り設定の指定結果は定義されていません。

Amazon DocumentDB は、読み取り設定としての タグセット の設定はサポートしていません。

サポートされている読み取り設定オプション

- **`primary`** - `primary` 読み取り設定を指定すると、すべての読み取りがクラスターのプライマリインスタンスにルーティングされます。プライマリインスタンスが使用できない場合、読み込みオペレーションは失敗します。`primary` 読み込み設定により、書き込み後読み取りの整合性が得られ、高可用性および読み込みスケーリングよりも書き込み後読み取りの整合性を優先するユースケースに適しています。

  次の例では、`primary` 読み取り設定を指定します。

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

- **primaryPreferred** - primaryPreferred 読み取り設定を指定すると、通常のオペレーションでは読み取りがプライマリインスタンスにルーティングされます。プライマリフェイルオーバーが発生した場合、クライアントはレプリカにリクエストをルーティングします。primaryPreferred 読み取り設定により、通常のオペレーションで書き込み後読み取り整合性と、フェイルオーバーイベント中の結果整合性のある読み込みが得られます。primaryPreferred 読み取り設定は、読み込みスケーリングよりも書き込み後読み取り整合性を優先するが、それでも高可用性を必要とするユースケースに適しています。

  次の例では、primaryPreferred 読み取り設定を指定します。

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

- **secondary** - secondary 読み取り設定を指定すると、読み取りはレプリカのみにルーティングされ、プライマリインスタンスにルーティングされることはありません。クラスター内のレプリカインスタンスがない場合、読み込みリクエストは失敗します。secondary 読み取り設定では、結果整合性のある読み込みが得られ、高可用性および書き込み後の読み取り整合性よりもプライマリインスタンスの書き込みスループットを優先するユースケースに適しています。

  次の例では、secondary 読み取り設定を指定します。

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

- **secondaryPreferred** - secondaryPreferred 読み取り設定を指定すると、1 つ以上のレプリカがアクティブになったときに、読み取りはリードレプリカにルーティングされます。クラスター内にアクティブなレプリカインスタンスがない場合、読み取りリクエストはプライマリインスタンスにルーティングされます。secondaryPreferred 読み取り設定では、読み取りがリードレプリカによって対応されたときに、結果整合性のある読み込みが得られます。これにより、読み取りがプライマリインスタンスによって処理されたときに (フェイルオーバーイベントがなければ)、書き込み後読み取り整合性が得られます。secondaryPreferred 読み取り設定は、書き込み後読み取り整合性よりも読み取りスケーリングや高可用性を優先するユースケースに適しています。

  次の例では、secondaryPreferred 読み取り設定を指定します。

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

- **`nearest`** - `nearest` 読み取り設定を指定すると、クライアントと Amazon DocumentDB クラスター内のすべてのインスタンス間で測定されたレイテンシーのみに基づいて、読み取りがルーティングされます。`nearest` 読み取り設定では、読み取りがリードレプリカによって対応されたときに、結果整合性のある読み込みが得られます。これにより、読み取りがプライマリインスタンスによって処理されたときに (フェイルオーバーイベントがなければ)、書き込み後読み取り整合性が得られます。`nearest` 読み取り設定は、書き込み後読み取り整合性と読み込みスケーリングよりも、可能な限り低い読み取りレイテンシーと高可用性の達成を優先するユースケースに適しています。

  次の例では、`nearest` 読み取り設定を指定します。

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

高可用性

Amazon DocumentDB は、レプリカをプライマリインスタンスのフェイルオーバーターゲットとして使用して、可用性の高いクラスター設定をサポートします。プライマリインスタンスが失敗すると、Amazon DocumentDB レプリカが新しいプライマリとして昇格します。ここでは、プライマリインスタンスに対して行われた読み取り要求と書き込み要求が失敗して例外が発生するときに、短時間の中断が発生します。

Amazon DocumentDB クラスターにレプリカが含まれていない場合は、障害の発生時にプライマリインスタンスが再作成されます。Amazon DocumentDB レプリカの昇格はプライマリインスタンスの再作成よりもはるかに高速であるため、フェイルオーバーターゲットとして 1 つ以上の Amazon DocumentDB レプリカを作成します。

フェイルオーバーターゲットとして使用することを目的としたレプリカは、プライマリインスタンスと同じインスタンスのものである必要があります。これらは、プライマリとは別のアベイラビリティーゾーンでプロビジョニングする必要があります。フェイルオーバーターゲットとして設定されるレプリカを制御できます。高可用性のための Amazon DocumentDB の設定に関するベストプラクティスについては、「Amazon DocumentDB クラスターの耐障害性について理解する」を参照してください。

読み取りスケーリング

Amazon DocumentDB レプリカは、読み取りスケーリングに最適です。これらは、クラスターボリュームでの読み取りオペレーション専用です。つまり、レプリカは書き込みを処理しません。データのレプリケーションは、インスタンス間ではなくクラスターボリューム内で行われます。したがって、各レプリカのリソースはクエリの処理専用となり、データのレプリケートおよび書き込み用とはなりません。

アプリケーションがより多くの読み取り容量を必要とする場合、すぐにクラスターにレプリカを追加できます (通常は 10 分未満)。読み取り容量要件が減少した場合、不要なレプリカを削除できます。Amazon DocumentDB レプリカを使用すると、必要な読み取り容量のみのお支払いで済みます。

Amazon DocumentDB は、読み取り設定オプションの使用を通じて、クライアント側の読み取りスケーリングをサポートします。詳細については、「Amazon DocumentDB 読み取り設定」を参照してください。

TTL 削除

バックグラウンドプロセスによって達成された TTL インデックスエリアからの削除は、ベストエフォートであり、特定の期間内に保証されるものではありません。インスタンスサイズ、インスタンスリソースの使用率、ドキュメントサイズ、全体的なスループットなどの要因は、TTL 削除のタイミングに影響を与える可能性があります。

TTL モニタがドキュメントを削除すると、削除ごとに IO コストが発生するため、請求額が増加します。スループットおよび TTL 削除のレートが増加した場合、IO 使用量が増加するため、請求の増加が予想されます。

既存のコレクションに TTL インデックスを作成する場合、インデックスを作成する前に、期限切れのドキュメントをすべて削除する必要があります。現在の TTL 実装は、コレクション内のドキュメントのごく一部を削除するために最適化されています。これは、最初からコレクションでTTLが有効化されていた場合であり、一度に多数のドキュメントを削除する必要がある場合、必要以上に高い IOPS が発生する可能性があるからです。

TTL インデックスを作成してドキュメントを削除する代わりに、時間に基づいてドキュメントをコレクションにセグメント化し、ドキュメントが不要になったときにそれらのコレクションをドロップすることができます。例えば、週に 1 つのコレクションを作成し、IO コストをかけずにドロップできます。これは、TTL インデックスを使用するよりもコスト効率が大幅に向上します。

有料リソース

有料 Amazon DocumentDB リソースの特定

Amazon DocumentDB はフルマネージドデータベースサービスであるため、インスタンス、ストレージ、I/O、バックアップ、およびデータ転送が有料です。詳細については、「Amazon DocumentDB (MongoDB 互換) の料金」を参照してください。

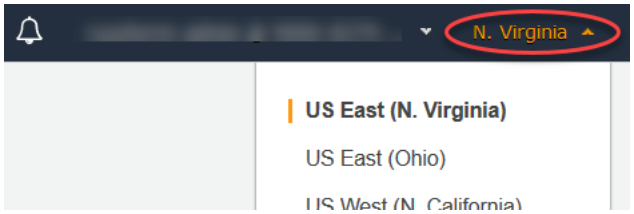
アカウント内の請求対象リソースを検出し、そのリソースを削除する場合は、 AWS マネジメントコンソール または を使用できます AWS CLI。

の使用 AWS マネジメントコンソール

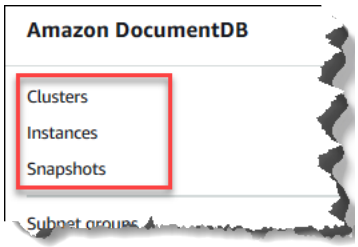
を使用すると AWS マネジメントコンソール、特定の に対してプロビジョニングした Amazon DocumentDB クラスター、インスタンス、スナップショットを検出できます AWS リージョン。

クラスター、インスタンス、およびスナップショットを検出するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. デフォルトリージョン以外のリージョンで請求対象リソースを検出するには、画面の右上隅で、検索 AWS リージョン する を選択します。



3. ナビゲーションペインで、該当する有料リソースのタイプ ([クラスター]、[インスタンス]、または [スナップショット]) を選択します。



4. そのリージョンについてプロビジョニングされたすべてのクラスター、インスタンス、またはスナップショットが右側のペインに表示されます。クラスター、インスタンス、およびスナップショットにたいして料金が発生します。

の使用 AWS CLI

を使用すると AWS CLI、特定の に対してプロビジョニングした Amazon DocumentDB クラスター、インスタンス、スナップショットを検出できます AWS リージョン。

クラスターおよびインスタンスを見つけるには

次のコードは、指定されたリージョンのすべてのクラスターとインスタンスをリスト表示しています。デフォルトのリージョンでクラスターとインスタンスを検索する場合は、`--region` パラメータを省略できます。

Example

Linux、macOS、Unix の場合:

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

Windows の場合:

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

このオペレーションによる出力は、次のようになります。

```
"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",
```

スナップショットを見つけるには

次のコードは、指定されたリージョンのすべてのスナップショットをリスト表示しています。デフォルトのリージョンでスナップショットを検索する場合は、`--region` パラメータを省略できます。

Linux、macOS、Unix の場合:

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

Windows の場合:

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

このオペレーションによる出力は、次のようになります。

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

削除する必要があるのは、manual スナップショットのみです。Automated スナップショットは、クラスターを削除すると削除されます。

不要な有料リソースの削除

クラスターを削除するには、まずクラスター内のインスタンスをすべて削除する必要があります。

- インスタンスを削除するには、「Amazon DocumentDB インスタンスの削除」を参照してください。

> **Important**
>
> クラスター内のインスタンスを削除しても、そのクラスターに関連付けられているストレージとバックアップの使用量に対して料金が発生します。すべての課金を停止するには、クラスターと手動スナップショットも削除する必要があります。

- クラスターを削除するには、「Amazon DocumentDB クラスターの削除」を参照してください。
- 手動スナップショットを削除するには、「クラススナップショットの削除」を参照してください。

ドキュメントデータベースとは

データモデルを正規化された行や列の観点から考えない開発者もいます。通常、アプリケーション層では、データは JSON ドキュメントとして表されます。これは、開発者にとって、データモデルをドキュメントとして考える方が直感的であるためです。

アプリケーションコードで使用するのと同じドキュメントモデル形式を使用してデータをデータベースに保持できるため、ドキュメントデータベースの人気が高まっています。ドキュメントデータベースは、柔軟でアジャイルな開発のための強力で直感的な API を提供します。

トピック

- ドキュメントデータベースのユースケース
- ドキュメントについて理解する
- ドキュメントでの作業

ドキュメントデータベースのユースケース

ユースケースにより、ドキュメントデータベースが必要か、またはデータを管理するためのその他の種類のデータベースが必要かが決まります。ドキュメントデータベースは、迅速で反復性のある開発のための柔軟なスキーマを必要とするワークロードに適しています。以下では、ドキュメントデータベースに大きな利点があるユースケースの例をいくつか示します。

トピック

- ユーザープロファイル
- リアルタイムのビッグデータ
- コンテンツ管理

ユーザープロファイル

ドキュメントデータベースには柔軟性が高いスキーマがあるため、属性とデータ値が異なるドキュメントを保存できます。ドキュメントデータベースは、さまざまなユーザーがさまざまな種類の情報を提供するオンラインプロファイルに対する実用的なソリューションです。ドキュメントデータベース

を使用して、各ユーザーに固有の属性のみを保存し、各ユーザーのプロファイルを効率的に保存することができます。

ユーザーが自分のプロファイル情報を追加または削除したとします。この場合、ドキュメントは、最近追加された属性とデータを含むか、新しく省略された属性とデータが省略された、更新されたバージョンに簡単に置き換えることができます。ドキュメントデータベースは、このようなレベルの個性や流動性を簡単に管理することができます。

リアルタイムのビッグデータ

従来、運用データから情報を抽出する能力は、運用データベースと分析データベースがそれぞれ異なる環境 (運用環境およびビジネス/レポート作成) で維持されているという事実によって妨げられていました。リアルタイムで動作情報を抽出できることは、競争の激しいビジネス環境において非常に重要です。ドキュメントデータベースを使用することで、企業は任意の送信元からの運用データを保存して管理し、同時に分析のため任意の BI エンジンにデータをフィードできます。2 つの環境を持つ必要はありません。

コンテンツ管理

コンテンツを効果的に管理するために、さまざまなソースからコンテンツを収集して集計し、お客様に提供する必要があります。ドキュメントデータベースは、柔軟性の高いスキーマがあるため、あらゆる種類のデータの収集および保存に最適です。それらを使用して、イメージ、コメント、ビデオなど、ユーザーが生成したコンテンツを含めて、新しい種類のコンテンツを作成し、組み込むことができます。

ドキュメントについて理解する

ドキュメントデータベースは、セミ構造化データをドキュメントとして格納するために使用されます。リレーショナルデータベースのように、それぞれが固有の固定構造を持つ複数のテーブルにわたってデータを正規化するのではなく、ドキュメントとして格納します。ドキュメントデータベースに保存されているドキュメントは、ネストされたキーと値のペアを使用して、ドキュメントの構造またはスキーマを提供します。ただし、さまざまな種類のドキュメントを同じドキュメントデータベースに保存できるため、形式が異なる類似したデータの処理要件を満たします。たとえば、各ドキュメントは自己記述型であるため、トピック「ドキュメントデータベースのドキュメントの例」で説明しているオンラインストアの JSON でエンコードされたドキュメントは、同じドキュメントデータベースに保存することができます。

トピック

- SQL 用語と非リレーショナル用語の比較

- シンプルなドキュメント
- 埋め込まれたドキュメント
- ドキュメントデータベースのドキュメントの例
- ドキュメントデータベースの正規化について

SQL 用語と非リレーショナル用語の比較

次の表は、SQL データベースによって使用される用語とドキュメントデータベース (MongoDB) で使用される用語を比較したものです。

SQL	MongoDB
[テーブル]	収集
Row	ドキュメント
列	フィールド
プライマリキー	ObjectId
[Index] (インデックス)	[Index] (インデックス)
表示	表示
ネストされたテーブルまたはオブジェクト	埋め込まれたドキュメント
配列	配列

シンプルなドキュメント

ドキュメントデータベースのすべてのドキュメントは自己記述型です。このドキュメントでは、JSON に似た形式のドキュメントを使用しますが、エンコードの他の方法も使用できます。

単純ドキュメントには、ドキュメント内ですべてが同じレベルの 1 つ以上のフィールドがあります。次の例では、フィールド `SSN`、`LName`、`FName`、`DOB`、`Street`、`City`、`State-Province`、`PostalCode`、および `Country` はすべてドキュメント内の兄弟です。

```
{
```

```
   "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"
}
```

情報を単純ドキュメントに整理すると、各フィールドは個別に管理されます。ユーザーのアドレスを取得するには、Street、City、State-Province、PostalCode、および Country を個別のデータ項目として取得する必要があります。

埋め込まれたドキュメント

複合ドキュメントは、ドキュメント内に埋め込まれたドキュメントを作成してデータを整理します。埋め込まれたドキュメントでは、データをグループまたは個別のデータ項目として管理するために役立ちます。ケース別に、どちらか効率的なほうを選択できます。前の例を使用すると、メインドキュメントに Address ドキュメントを埋め込むことができます。この操作を行うと、次のドキュメント構造になります。

```
{
   "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"
   }
}
```

ドキュメント内のデータは、個別のフィールド ("SSN":)、埋め込まれたドキュメント ("Address":)、または埋め込まれたドキュメントのメンバー ("Address":{"Street":}) としてアクセスできるようになりました。

ドキュメントデータベースのドキュメントの例

前に説明したように、ドキュメントデータベース内の各ドキュメントは自己記述型であるため、ドキュメントデータベース内のドキュメントの構造は、相互に異なる可能性があります。次の 2 つのドキュメント (1 つは書籍用、もう 1 つは定期刊行物用) は、構造的に異なります。ただし、両方とも同じドキュメントデータベースに存在できます。

書籍ドキュメントの例を次に示します。

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

2 つの記事を含む定期刊行物の例を以下に示します。

```
{
    "_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"
}
```

これら 2 つのドキュメントの構造を比較します。リレーショナルデータベースでは、別々の「定期刊行物」と「書籍」テーブル、または null の値として「刊行物」、「問題」、「記事」、「MI」など未使用のフィールドを持つ 1 つのテーブルが必要です。ドキュメントデータベースが半構造化され、各ドキュメントが独自の構造を定義しているため、それらの 2 つのドキュメントは null フィールドなしで同じドキュメントデータベースに共存できます。ドキュメントデータベースは、スパースデータの処理に適しています。

ドキュメントデータベースに対する開発により、迅速で反復的な開発が可能になります。これは、動的にドキュメントのデータ構造を変更でき、コレクション全体のスキーマを変更する必要がないためです。ドキュメントデータベースが適しているのは、アジャイル開発と動的に変化する環境です。

ドキュメントデータベースの正規化について

ドキュメントデータベースは正規化されていません。そのため、1 つのドキュメントで見つかったデータを別のドキュメントで繰り返すことができます。さらに、ドキュメント間で一部のデータに相違が発生する場合があります。たとえば、オンラインストアで購入を行い、購入のすべての詳細が 1 つのドキュメントに保存されるシナリオを考えてみます。このドキュメントは、たとえば次の JSON ドキュメントのようになります。

```
{
    "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"
}
```

この情報は、トランザクションコレクションでドキュメントとして保存されます。後で、1 つのアイテムの購入を忘れていることに気付きます。そのため、再度同じストアにログオンし、別の購入を行います。これもトランザクションコレクションの別のドキュメントとして保存されます。

```
{
    "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"
```

```
}
```

この 2 つのドキュメント、つまり名前と買い物客 ID (同じクレジットカードを使用した場合はクレジットカード情報) の冗長性に注目してください。しかし、ストレージは安価であり、各ドキュメントは結合が必要ない単純なキーと値のクエリで迅速に取得できる単一のトランザクションを完全に記録するため、問題はありません。

また、2 つの文書の間には、クレジットカード情報という明らかな相違があります。これは明らかな相違であるのは、各購入で別のクレジットカードを使用した可能性が高いためです。各ドキュメントは、記録されるトランザクションについては正確です。

ドキュメントでの作業

ドキュメントデータベースとしての Amazon DocumentDB は、JSON データの保存、クエリ実行、インデックス作成を容易にします。Amazon DocumentDB において、コレクションはリレーショナルデータベースのテーブルに似ています。ただし、すべてのドキュメントに適用される単一のスキーマが存在しない点が異なります。コレクションでは、すべてのドキュメントを同じデータベースに保持しながら類似したドキュメントを一緒にグループ化でき、構造が類似している必要はありません。

前のセクションのドキュメント例を使用すると、`reading_material` および `office_supplies` のコレクションがあります。ソフトウェアは、ドキュメントが属するコレクションを適用する必要があります。

以下の例では、MongoDB API を使用して、ドキュメントを追加、クエリ、更新、および削除する方法を示します。

トピック

- ドキュメントの追加
- ドキュメントのクエリ
- ドキュメントの更新
- ドキュメントの削除

ドキュメントの追加

Amazon DocumentDB では、コレクションにドキュメントを最初に追加したときにデータベースが作成されます。次の例では、`example` という名前のコレクションを `test` データベースに作成しています。これは、クラスターに接続するときのデフォルトのデータベースです。最初のドキュメント

の挿入時に接続が暗黙で作成されるため、コレクション名のエラーチェックはありません。したがって、コレクション名にタイプミス (example の代わりに eexample など) があると、正しいコレクションの代わりに eexample コレクションが作成され、このコレクションにドキュメントが追加されます。エラーチェックはアプリケーションで処理する必要があります。

次の例では、MongoDB API を使用してドキュメントを追加します。

トピック

- 単一のドキュメントの追加
- 複数のドキュメントの追加

単一のドキュメントの追加

単一のドキュメントをコレクションに追加するには、コレクションに追加したドキュメントで insertOne({}) オペレーションを使用します。

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

このオペレーションによる出力は、次のようになります (JSON 形式)。

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

複数のドキュメントの追加

複数のドキュメントをコレクションに追加するには、コレクションに追加したドキュメントのリストで insertMany([{},...,{}]) オペレーションを使用します。この特定のリストのドキュメントのスキーマは異なりますが、すべてのドキュメントを同じコレクションに追加できます。

```
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
            }
        }
    ]
)
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

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

ドキュメントのクエリ

ときどきオンラインストアのインベントリを検索し、販売商品を顧客が表示して購入できることを確認する必要があります。コレクションのクエリは、コレクションのすべてのドキュメントを対象とするか、特定の条件を満たすドキュメントのみを対象とするかにかかわらず、比較的簡単です。

ドキュメントのクエリを実行するには、find() オペレーションを使用します。この find() コマンドには、返すドキュメントを選択するために使用する条件を定義する単一のドキュメントパラメータがあります。find() からの出力は、改行のない 1 行のテキストとしてフォーマットされたドキュメントです。読みやすくするために出力ドキュメントをフォーマットするには、find().pretty() を使用します。このトピックのすべての例では、.pretty() を使用して出力を書式設定しています。

example と insertOne() の 2 つの演習で insertMany() コレクションに挿入した 4 つのドキュメントを使用してください。

トピック

- コレクション内のすべてのドキュメントの取得
- フィールドの値に一致するドキュメントの取得
- 埋め込みドキュメントに一致するドキュメントの取得
- 埋め込まれたドキュメントのフィールド値に一致するドキュメントの取得
- 配列に一致するドキュメントの取得
- 配列値に一致するドキュメントの取得
- 演算子を使用したドキュメントの取得

コレクション内のすべてのドキュメントの取得

コレクション内のすべてのドキュメントを取得するには、find() オペレーションで、空のクエリドキュメントを指定します。

以下のクエリは、example コレクション内のすべてのドキュメントを返します。

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

フィールドの値に一致するドキュメントの取得

フィールドおよび値に一致するすべてのドキュメントを取得するには、find() オペレーションで、一致するフィールドおよび値を識別するクエリドキュメントを指定します。

上記のドキュメントを使用して、このクエリは、「Item」フィールドが「Pen」に等しいすべてのドキュメントを返します。

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

埋め込みドキュメントに一致するドキュメントの取得

埋め込みドキュメントに一致するすべてのドキュメントを検索するには、find() オペレーションで、埋め込みドキュメントの名前と、その埋め込みドキュメントのすべてのフィールドおよび値を識別するクエリドキュメントを指定します。

埋め込みドキュメントに一致するには、そのドキュメントの埋め込みドキュメントの名前が、クエリで指定した名前と同じであることが必要です。さらに、埋め込みドキュメントのフィールドおよび値も、クエリで指定したものと同じであることが必要です。

次のクエリでは、「Poster Paint」ドキュメントのみ返ります。これは、「Pen」の値は「OnHand」や「MinOnHand」とは異なり、「Spray Paint」には、クエリドキュメントよりもフィールドが 1 つ (OrderQnty) 多いためです。

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

埋め込まれたドキュメントのフィールド値に一致するドキュメントの取得

埋め込みドキュメントに一致するすべてのドキュメントを検索するには、find() オペレーションで、埋め込みドキュメントの名前と、その埋め込みドキュメントのすべてのフィールドおよび値を識別するクエリドキュメントを指定します。

上記のドキュメントでは、以下のようにクエリで「ドット表記」を使用して埋め込みドキュメントと目的のフィールドを指定しています。埋め込みドキュメント内に他のフィールドがある場合でも、これらに一致するすべてのドキュメントが返されます。クエリは「Poster Paint」と「Spray Paint」を返します。それらはいずれも、指定したフィールドおよび値と一致するためです。

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

配列に一致するドキュメントの取得

配列に一致するすべてのドキュメントを検索するには、find() オペレーションで、目的の配列名とその配列内のすべての値を指定します。クエリは、配列値の名前および順序がクエリで指定したものと一致するすべてのドキュメントを返します。

以下のクエリは「Pen」のみを返します。「Poster Paint」では追加の色 (White) があり、「Spray Paint」では色の順序が異なるためです。

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

配列値に一致するドキュメントの取得

特定の属性配列を含むすべてのドキュメントを見つけるには、find() オペレーションで、目的の配列名と値を指定します。

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

上記のオペレーションは、すべてのドキュメントを返します。3 つのドキュメントのそれぞれで、Colors という配列があり、その配列のどこかに「Red」という値があるためです。値として「White」を指定すると、クエリからは「Poster Paint」のみが返されます。

演算子を使用したドキュメントの取得

以下のクエリは、「Inventory.OnHand」値が 50 未満のすべてのドキュメントを返します。

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

サポートされているクエリ演算子のリストについては、「クエリおよびプロジェクション演算子」を参照してください。

ドキュメントの更新

通常、ドキュメントは静的ではなく、アプリケーションワークフローの一部として更新されます。以下の例では、ドキュメントを更新する方法のいくつかを示します。

既存のドキュメントを更新するには、update() オペレーションを使用します。update() オペレーションには 2 つのドキュメントパラメータがあります。最初のドキュメントでは、更新するドキュメントを識別しています。2 番目のドキュメントでは、行う更新を指定しています。

既存のフィールド (シンプルなフィールド、配列、または埋め込みドキュメントは関係ありません) を更新する場合 フィールド名とその値を指定します。オペレーションの最後では、古いドキュメントのフィールドが新しいフィールドおよび値に置き換えられたようになります。

トピック

- 既存のフィールドの値の更新
- 新しいフィールドの追加
- 埋め込みドキュメントの置き換え
- 埋め込みドキュメントへの新しいフィールドの挿入
- ドキュメントからのフィールドの削除
- 複数のドキュメントからのフィールドの削除

既存のフィールドの値の更新

先ほど追加した次の 4 つのドキュメントを以下の更新オペレーションに使用します。

```
{
    "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
    }
}
```

シンプルなフィールドを更新するには

シンプルなフィールドを更新するには、update() で $set を使用してフィールド名と新しい値を指定します。以下の例では、Item を「Pen」から「Gel Pen」に変更します。

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

このオペレーションの結果は、次のようになります。

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

配列を更新するには

以下の例では、既存の色の配列を、色のリストに Orange を含み White を含まない新しい配列に置き換えます。新しい色のリストは、update() オペレーションで指定された順序になります。

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

このオペレーションの結果は、次のようになります。

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

新しいフィールドの追加

1 つ以上の新しいフィールドを追加してドキュメントを変更するには、挿入先のドキュメントと、$set 演算子を使用して挿入する新しいフィールドと値を識別するクエリドキュメントで、update() オペレーションを使用します。

以下の例では、値 3.99 を含むフィールド UnitPrice を Spray Paints ドキュメントに追加します。値 3.99 は数字であり、文字列ではありません。

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

このオペレーションの結果は、次の (JSON フォーマット) のようになります。

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

埋め込みドキュメントの置き換え

埋め込みドキュメントを置き換えてドキュメントを変更するには、update() オペレーションで、埋め込みドキュメントと、$set 演算子を使用して挿入する新しいフィールドおよび値を識別するドキュメントを指定します。

以下のドキュメントがあるとします。

```
db.example.insert({
    "DocName": "Document 1",
    "Date": {
        "Year": 1987,
        "Month": 4,
        "Day": 18
    }
})
```

埋め込みドキュメントを置き換えるには

以下の例では、現在の Date ドキュメントを、フィールド Month と Day のみを含み Year を含まない新しいドキュメントに置き換えます。

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

このオペレーションの結果は、次のようになります。

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

埋め込みドキュメントへの新しいフィールドの挿入

埋め込みドキュメントにフィールドを追加するには

埋め込みドキュメントに 1 つ以上の新しいフィールドを追加してドキュメントを変更するには、update() オペレーションで、「ドット表記」を使用して、埋め込みドキュメントと、$set 演算子を使用して挿入する新しいフィールドおよび値を識別するドキュメントを指定します。

以下のドキュメントでは、コードに「ドット表記」を使用して Year フィールドと DoW フィールドを埋め込み Date ドキュメントに挿入し、Words フィールドを親ドキュメントに挿入しています。

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

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

このオペレーションの結果は、次のようになります。

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

ドキュメントからのフィールドの削除

ドキュメントからフィールドを削除することでドキュメントを変更するには、update() オペレーションで、フィールドを削除するドキュメントを識別するクエリドキュメントを指定し、$unset 演算子で、削除するフィールドを指定します。

以下の例では、上記のドキュメントから Words フィールドを削除します。

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

このオペレーションの結果は、次のようになります。

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

複数のドキュメントからのフィールドの削除

複数のドキュメントからフィールドを削除してドキュメントを変更するには、update( ) オペレーションを使用します ($unset 演算子を指定し、multi オプションを true に設定します)。

次の例では、コレクション例のすべてのドキュメントから Inventory フィールドを削除します。ドキュメントに Inventory フィールドがない場合、そのドキュメントに対して実行されるアクションはありません。multi: true が省略された場合、アクションは条件を満たす最初のドキュメントでのみ実行されます。

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

ドキュメントの削除

データベースからドキュメントを削除するには、remove( ) オペレーションを使用します。これにより、削除するドキュメントを指定します。以下のコードは example コレクションから「Gel Pen」を削除します。

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

データベースからすべてのドキュメントを削除するには、remove() オペレーションで空のクエリを使用します。

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

Amazon DocumentDB の使用開始

Amazon DocumentDB に接続して開始する方法は複数あります。このガイドは、ユーザーにとって最も速く、最もシンプルで、最も簡単にドキュメントデータベースを使い始めることができる方法を提供します。このガイドでは、AWS CloudShell を使用して、 AWS マネジメントコンソールから直接 Amazon DocumentDB クラスターに接続してクエリを実行します。 AWS 無料利用枠の対象となる新規のお客様は、Amazon DocumentDB と CloudShell を無料で使用できます。 AWS CloudShell 環境または Amazon DocumentDB クラスターが無料利用枠を超えるリソースを使用する場合、それらのリソースの通常の AWS 料金が課金されます。このガイドを読めば、Amazon DocumentDB を 5 分もかからずに使い始めることができます。

> **Note**
>
> このガイドの手順は、Amazon DocumentDB と が AWS CloudShell 利用可能な Amazon DocumentDB インスタンスベースのクラスターを作成して接続するためのものです。
>
> - Amazon DocumentDB エラスティッククラスターを作成して接続する場合は、「Amazon DocumentDB エラスティッククラスターの開始方法」を参照してください。
> - AWS 中国リージョンにお住まいの場合は、「」を参照してくださいAmazon EC2 への自動接続。

トピック

- 前提条件
- ステップ 1: クラスターを作成する
- ステップ 2: クラスターに接続する
- ステップ 3: データを挿入してクエリする
- ステップ 4: 探索する
- を使用した Amazon DocumentDB クイックスタート CloudFormation

前提条件

最初の Amazon DocumentDB クラスターを作成する前に、以下の操作を行う必要があります。

Amazon Web Services (AWS) アカウントを作成する

Amazon DocumentDB を使用する前に、Amazon Web Services (AWS) アカウントを持っている必要があります。 AWS アカウントは無料です。使用しているサービスとリソースに対してのみ料金をお支払いいただきます。

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://signin.aws.amazon.com/signup を開きますか?request_type=register。
2. オンラインの手順に従います。

   サインアップ手順の一環として、電話またはテキストメッセージを受け取り、電話キーパッドで検証コードを入力します。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザー が作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティベストプラクティスとして、ユーザーに管理アクセス権を割り当て、ルートユーザーアクセスが必要なタスクの実行にはルートユーザーのみを使用するようにしてください。

必要な AWS Identity and Access Management (IAM) アクセス許可を設定します。

クラスター、インスタンス、クラスターパラメータグループなどの Amazon DocumentDB リソースを管理するためのアクセスには、 がリクエストの認証 AWS に使用できる認証情報が必要です。詳細については、「Amazon DocumentDB の Identity and Access Management」を参照してください。

1. の検索バーで AWS マネジメントコンソール、表示されるドロップダウンメニューで IAM **IAM**を入力して選択します。
2. IAM コンソールにアクセスしたら、ナビゲーションペインから [ユーザー] を選択します。
3. ユーザーネームを選択します。
4. [Add permissions (許可の追加)] を選択します。
5. [ポリシーを直接アタッチする] を選択します。
6. 検索バーに `AmazonDocDBFullAccess` と入力し、検索結果に表示されたら、それを選択します。

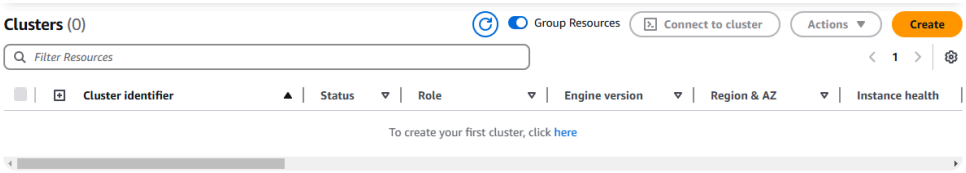
7. [次へ] を選択します。
8. [Add permissions (許可の追加)] を選択します。

> **Note**
>
> AWS アカウントには、各リージョンにデフォルトの VPC が含まれています。Amazon VPC を使用することを選択した場合は、「Amazon VPC ユーザーガイド」の「Amazon VPC を作成する」トピックの手順を完了します。

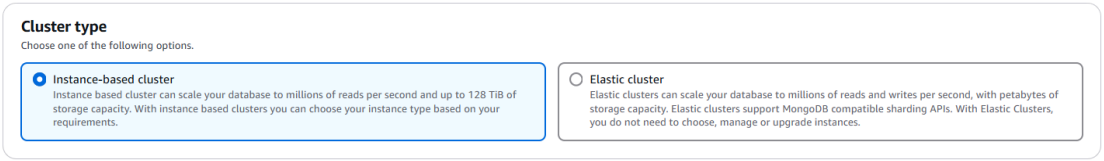
ステップ 1: クラスターを作成する

このステップでは、Amazon DocumentDB クラスターを作成します。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. Amazon DocumentDB マネジメントコンソールで、[クラスター] の下にある [作成] を選択します。



3. 「Amazon DocumentDB クラスターの作成」ページの [クラスタータイプ] セクションで、[インスタンスベースのクラスター] を選択します (これはデフォルトのオプションです)。

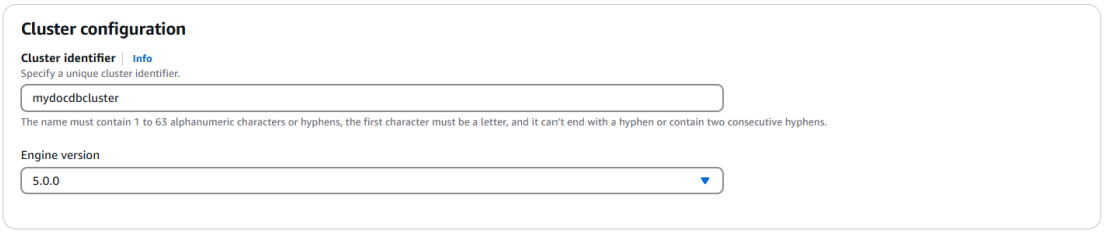


> **Note**
>
> このカテゴリのもう 1 つのオプションは Elastic クラスター です。「Amazon DocumentDB Elastic クラスター」の詳細については、「Amazon DocumentDB Elastic クラスターの使用」を参照してください。

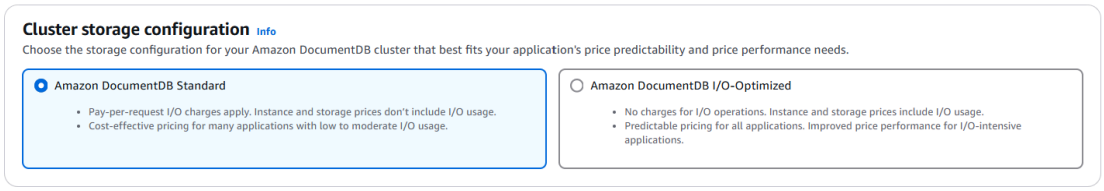
4. [クラスター設定] セクションで次を行います。

a. [クラスター識別子] には、**mydocdbcluster** などの一意の名称を入力します。コンソール操作では、入力方法に関係なくクラスター名のすべての文字が小文字に変換されることにご留意ください。

b. [エンジンバージョン] で、[5.0.0] を選択します。



5. [クラスターストレージ設定] セクションでは、[Amazon DocumentDB 標準] (デフォルトのオプションです) を選択します。



> **Note**
>
> このカテゴリのもう 1 つのオプションは、[Amazon DocumentDB I/O 最適化] です。いずれかのオプションの詳細については、「Amazon DocumentDB クラスターストレージ設定」を参照してください。

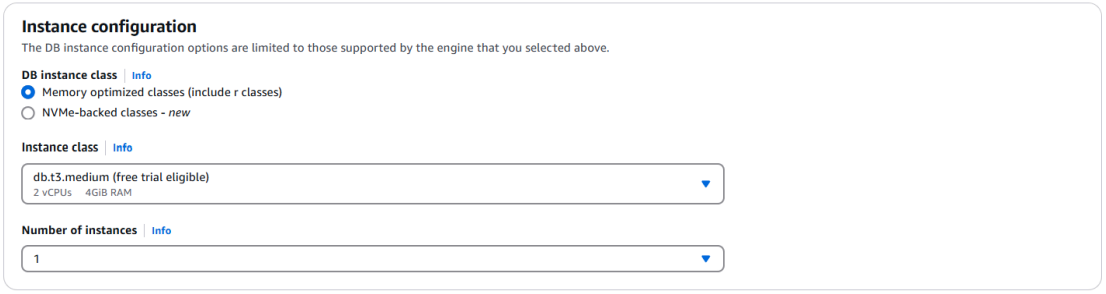
6. インスタンス設定 セクション:

a. [DB インスタンスクラス] では、メモリ最適化クラス (r クラスを含む) を選択します (デフォルト)。

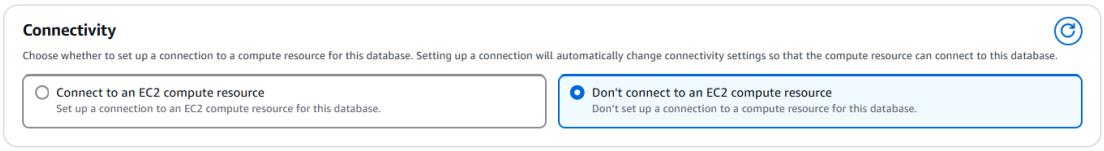
もう 1 つのインスタンスオプションは NVMe-backed クラス です。詳細については NVMe-backed インスタンスを参照してください。

b. [インスタンスクラス] で [t3.medium] を選択します。これは AWS 無料トライアルの対象となります。

c. [インスタンスの数] は、[1] を選択します。1 つのインスタンスを選択すると、コストを最小限に抑えることができます。これが本稼働システムの場合は、高可用性のために 3 つのインスタンスをプロビジョニングすることをお勧めします。

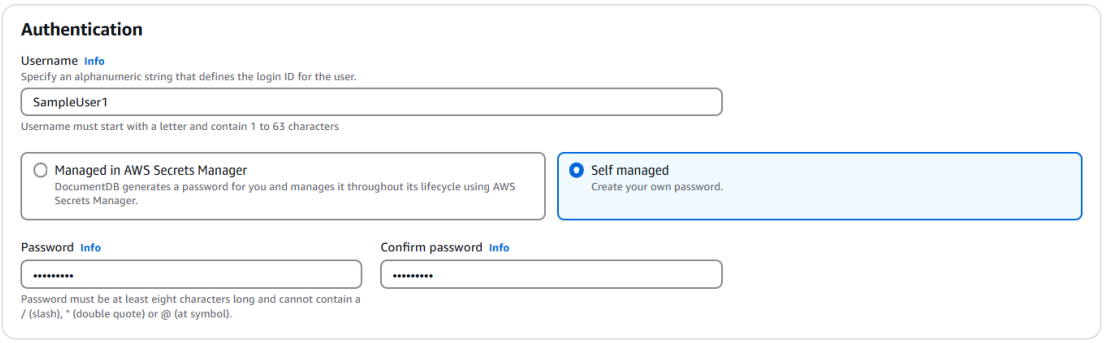


7. [接続] セクションで、デフォルト設定の [EC2 コンピューティングリソースに接続しない] のままにしておきます。



8. [認証] セクションで、プライマリユーザーのユーザー名を入力し、[セルフマネージド] を選択します。パスワードを入力して確認します。

   代わりに Managed in AWS Secrets Manager を選択した場合は、詳細については[Amazon DocumentDB と によるパスワード管理 AWS Secrets Manager]「」を参照してください。



9. 他のすべてのオプションはデフォルトのままにして、[クラスターの作成] を選択します。

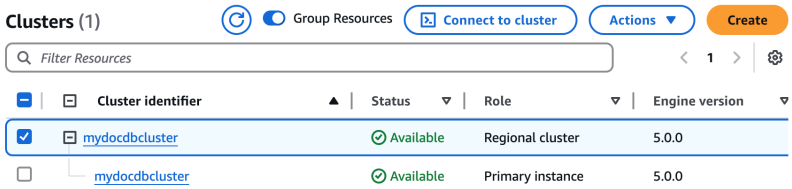
Amazon DocumentDB がクラスターをプロビジョニングしています。完了までに数分かかる場合もあります。

> **Note**
>
> クラスターのステータス値については、「Amazon DocumentDB のモニタリング」の章の「クラスターステータスの値」を参照してください。

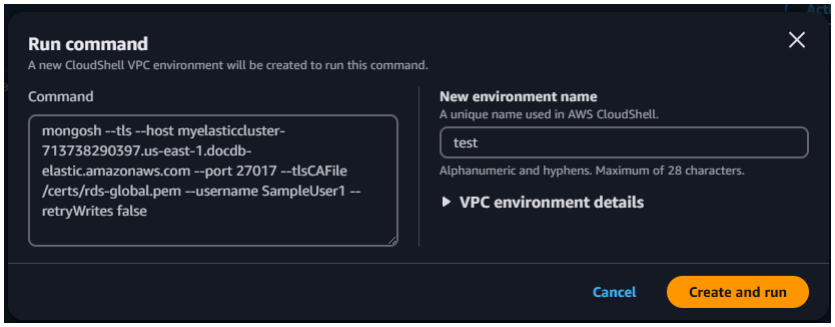
ステップ 2: クラスターに接続する

を使用して Amazon DocumentDB クラスターに接続します AWS CloudShell。

1. Amazon DocumentDB マネジメントコンソールの [クラスター] で、作成したクラスターを探します。クラスターの横にあるチェックボックスをクリックして選択します。



2. クラスターに接続する (アクションドロップダウンメニューの横にある) を選択します。このボタンは、クラスターの横にあるチェックボックスをクリックした後にのみ有効になり、リージョンクラスターとプライマリインスタンス (複数可) の両方のステータスが 使用可能 と表示されます。CloudShell の [Run コマンド] 画面が表示されます。
3. [新しい環境の名前] フィールドに、「test」などの一意の名前を入力し、[作成して実行する] を選択します。VPC 環境の詳細は、Amazon DocumentDB データベースに自動的に設定されます。



4. プロンプトが表示されたら、ステップ 1: Amazon DocumentDB クラスターを作成する (サブステップ 7) で作成したパスワードを入力します。



パスワードを入力して、プロンプトが `rs0 [direct: primary] <env-name>>` に変わったら、Amazon DocumentDB クラスターに正常に接続できています。

> **Note**
>
> トラブルシューティングについては、「Amazon DocumentDB のトラブルシューティング」を参照してください。

ステップ 3: データを挿入してクエリする

クラスターに接続できたので、いくつかのクエリを実行して、ドキュメントデータベースの使用に慣れることができます。

1. 1 つのドキュメントを挿入するには、次のように入力します。

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

   次の出力が返ります。

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

2. findOne() コマンドで書き込んだドキュメントを読み取ることができます (一つのドキュメントしか返さないため)。以下を入力します。

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

   次の出力が返ります。

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

3. さらにいくつかのクエリを実行するには、ゲームプロファイルのユースケースを検討してください。まず、「profiles」というタイトルのコレクションにエントリをいくつか挿入します。以下を入力します。

   ```
   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 }
])
```

次の出力が返ります。

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

4. find() コマンドを使用して、プロファイルコレクション内のすべてのドキュメントを返します。以下を入力します。

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

ステップ 3 で入力したデータと一致する出力を取得します。

5. フィルターを使用して 1 つのドキュメントに対して1つのクエリを使用します。以下を入力します。

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

次の出力が返ります。

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

6. それでは、プロファイルを探して、それを findAndModify コマンドで変更してみましょう。次のコードを使用して、ユーザー Matt に 10 ポイント追加します。

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

次のような出力が得られます (Mattのスコアはまだ上がっていないことに注意してください) 。

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

7. 次のクエリを使用して、Matt のスコアが変更されたことを確認できます。

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

次の出力が返ります。

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

ステップ 4: 探索する

お疲れ様でした。Amazon DocumentDB インスタンスベースのクラスター入門ガイドを正常に完了しました。

次のステップ このデータベースを一般的な機能の一部を使用して、完全に活用する方法を学びましょう。

- Amazon DocumentDB イベントの管理
- スケーリング
- バックアップと復元

> **Note**
>
> この入門演習で作成したクラスターは、削除しない限り、引き続きコストを計上します。手順については、以下を参照してください。Amazon DocumentDB クラスターを削除する。

を使用した Amazon DocumentDB クイックスタート CloudFormation

このセクションには、 CloudFormation を使用して Amazon DocumentDB (MongoDB 互換) をすぐに使い始めるための手順やその他の情報が含まれています。Amazon DocumentDB について一般的な情報は、「Amazon DocumentDB (MongoDB 互換) とは」を参照してください。

これらの手順では、 AWS CloudFormation テンプレートを使用して、デフォルトの Amazon VPC にクラスターとインスタンスを作成します。これらのリソースを自分で作成する方法については、「Amazon DocumentDB の使用開始」を参照してください。

> **Important**
>
> このテンプレートによって作成される CloudFormation スタックは、Amazon DocumentDB のリソース (クラスターやインスタンスなど) や Amazon Elastic Compute Cloud (サブネットグループなど) など、複数のリソースを作成します。
> これらのリソースの一部は無料利用枠のリソースではありません。料金情報については、「Amazon DocumentDB の料金」と「Amazon EC2 の料金」を参照してください。スタックを終了したら削除して、以降そのスタックの料金が発生しないようにできます。

この CloudFormation スタックはチュートリアルのみを目的としています。このテンプレートを本番環境で使用する場合は、より厳格な IAM ポリシーとセキュリティを使用してください。リソースのセキュリティについては「Amazon VPC セキュリティ」および「Amazon EC2 ネットワークとセキュリティ」を参照してください。

トピック

- 前提条件
- Amazon DocumentDB CloudFormation スタックの起動
- Amazon DocumentDB クラスターにアクセスする
- 終了保護と削除保護

前提条件

Amazon DocumentDB クラスターを作成する前に、以下の操作を行う必要があります。

- デフォルトのAmazon VPC
- 必須の IAM アクセス許可

必須の IAM アクセス許可

以下のアクセス許可を使用すると、 CloudFormation スタック用のリソースを作成できます。

AWS 管理ポリシー

- `AWSCloudFormationReadOnlyAccess`
- `AmazonDocDBFullAccess`

追加の IAM アクセス許可

次のポリシーは、この CloudFormation スタックの作成と削除に必要な追加のアクセス許可の概要を示しています。

次の例では ############## をリソースの情報で置き換えます。

Amazon EC2 キーペア

CloudFormation スタックを作成するリージョンで使用可能なキーペア (および PEM ファイル) が必要です。キーペアを作成する必要がある場合は、「Amazon EC2 ユーザーガイド」の「Amazon EC2 を使用したキーペアの作成」を参照してください。

Amazon DocumentDB CloudFormation スタックの起動

このセクションでは、Amazon DocumentDB CloudFormation スタックを起動して設定する方法について説明します。

1. AWS マネジメントコンソール で にサインインしますhttps://console.aws.amazon.com/。
2. 次の表に、各 AWS リージョン Amazon DocumentDB スタックテンプレートを示します。スタックを起動する の起動スタックを選択します。 AWS リージョン

リージョン	テンプレートを表示	デザイナーで表示	起動する
米国東部 (オハイオ)	テンプレートを表示	デザイナーで表示	Launch Stack
米国東部 (バージニア北部)	テンプレートを表示	デザイナーで表示	Launch Stack
米国西部 (オレゴン)	テンプレートを表示	デザイナーで表示	Launch Stack
アジアパシフィック (ムンバイ)	テンプレートを表示	デザイナーで表示	Launch Stack
アジアパシフィック (ソウル)	テンプレートを表示	デザイナーで表示	Launch Stack
アジアパシフィック (シンガポール)	テンプレートを表示	デザイナーで表示	Launch Stack

リージョン	テンプレートを表示	デザイナーで表示	起動する
アジアパシフィック (シドニー)	テンプレートを表示	デザイナーで表示	Launch Stack
アジアパシフィック (東京)	テンプレートを表示	デザイナーで表示	Launch Stack
カナダ (中部)	テンプレートを表示	デザイナーで表示	Launch Stack
欧州 (フランクフルト)	テンプレートを表示	デザイナーで表示	Launch Stack
欧州 (アイルランド)	テンプレートを表示	デザイナーで表示	Launch Stack
欧州 (ロンドン)	テンプレートを表示	デザイナーで表示	Launch Stack
欧州 (パリ)	テンプレートを表示	デザイナーで表示	Launch Stack

3. [スタックの作成] - 選択した Amazon DocumentDB テンプレートについて説明します。すべてのスタックは、スタックに含める AWS リソースに関する設定を含む JSON ファイルまたは YAML ファイルであるテンプレートに基づいています。上記のテンプレートからスタックを起動することを選択した場合、 AWS リージョン 選択した の Amazon DocumentDB スタックを作成するようにテンプレートが既に設定されています。

   CloudFormation スタックを起動すると、Amazon DocumentDB クラスターの削除保護はデフォルトで無効になります。クラスターの削除保護を有効にする場合は、次のステップを実行します。それ以外の場合は、[次へ] を選択して次のステップに進みます。

   Amazon DocumentDB クラスターの削除保護を有効にするには

   1. スタック作成 ページの右下隅から、デザイナーで表示 を選択します。
   2. コンソールの結果の CloudFormation デザイナーページで、統合された JSON および YAML エディタを使用してテンプレートを変更します。`Resources` セクションまでスクロールし、次のように変更して `DeletionProtection` を含めます。 CloudFormation デザイナーの使用の詳細については、CloudFormation 「デザイナーとは」を参照してください。

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. ページの左上隅にある [スタックの作成]
   ()
   を選択して変更を保存し、これらの変更を有効にしてスタックを作成します。
4. 変更を保存すると、[スタックの作成] ページにリダイレクトされます。
5. [次へ] を選択して続行します。

4. [スタックの詳細をの指定] - テンプレートのスタック名とパラメータを入力します。パラメータはテンプレートで定義されます。パラメータを使用すると、スタックの作成時または更新時にカスタム値を入力できます。

- [スタック名] に、スタックの名前を入力するか、指定された名前を受け入れます。スタック名には、文字 (A ~ Z および a ~ z)、数字 (0 ~ 9)、およびダッシュ (-) を使用できます。
- [パラメータ] に、次の詳細を入力します。
  - DBClusterName - Amazon DocumentDB クラスターの名前を入力するか、表示された名前を受け入れます。

    クラスターの名前付けの制約:
    - 長さは [1 ～ 63] 個の英字、数字、またはハイフンです。
    - 1 字目は英字である必要があります。
    - ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。
    - Amazon RDS、Neptune、Amazon DocumentDB のすべてのクラスターで AWS アカウント、リージョンごとに一意である必要があります。
  - DBInstanceClass - ドロップダウンリストから、Amazon DocumentDB クラスターのインスタンスクラスを選択します。
  - DBInstanceName - Amazon DocumentDB インスタンスの名前を入力するか、提供された名前を受け入れます。

    インスタンスの名前付けの制約:
    - 長さは [1 ～ 63] 個の英字、数字、またはハイフンです。
    - 1 字目は英字である必要があります。
    - ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。
    - AWS アカウントリージョンごとに、Amazon RDS、Neptune、Amazon DocumentDB のすべてのインスタンスで一意である必要があります。
  - マスターパスワード - データベース管理者アカウントのパスワード。
  - マスターユーザー - データベース管理者アカウントのユーザー名。MasterUser は英数字で始まる必要があり、英数字のみ使用できます。

  [次へ] を選択して変更を保存し、続行します。

5. [スタックオプションの設定] - スタックのタグ、アクセス権限、および追加オプションを設定します。
   - [タグ] - スタック内のリソースに適用するタグ (キーと値のペア) を指定します。スタックごとに最大 50 個の一意のタグを追加できます。

- [アクセス権限] - オプションです。IAM ロールを選択して、 CloudFormation がスタック内のリソースを作成、変更、または削除する方法を明示的に定義します。ロールを選択しない場合、 はユーザーの認証情報に基づいてアクセス許可 CloudFormation を使用します。サービスロールを指定する前に、サービスロールを割り当てるアクセス許可 (`iam:PassRole`) があるかどうかを確認します `iam:PassRole` アクセス許可は、どのロールを使用できるかを指定します。

  > **Note**
  >
  > サービスロールを指定すると、 は常にそのロールをそのスタックで実行されるすべてのオペレーション CloudFormation に使用します。このスタックでオペレーションを実行するアクセス許可を持つ他のユーザーは、割り当てるアクセス許可がない場合でも、このロールを使用できます。ユーザーが持つべきではないアクセス許可がロールに含まれる場合、ユーザーのアクセス許可を非意図的にエスカレーションできてしまいます。ロールが 最小特権 を付与することを確認します。

- [高度なオプション] - 以下の高度なオプションを設定することができます。
  - [スタックポリシー] - オプションです。スタックの更新中、意図しない更新から保護するリソースを定義します。デフォルトでは、スタックの更新時にすべてのリソースを更新できます。

    スタックポリシーを JSON として直接入力するか、スタックポリシーを含む JSON ファイルをアップロードすることができます。詳細については、「スタックのリソースが更新されないようにする」を参照してください。
  - [ロールバック構成] - オプションです。スタックの作成および更新時に がモニタリング CloudFormation する CloudWatch Logs アラームを指定します。オペレーションがアラームしきい値を超えた場合、 はアラームしきい値を CloudFormation ロールバックします。
  - [通知オプション] - オプションです。Simple Notification System (SNS) のトピックを指定します。
  - [スタック作成オプション] - オプションです。以下のオプションを指定できます。
    - [失敗時のロールバック] - スタックの作成に失敗した場合、スタックをロールバックするかどうかを指定します。
    - [タイムアウト] - スタック作成がタイムアウトするまでの時間 (分)です。
    - [終了保護] - スタックが誤って削除されるのを防ぎます。

> **Note**
>
> CloudFormation 終了保護はAmazon DocumentDB の削除保護の概念とは異なります。詳細については、「終了保護と削除保護」を参照してください。

[次へ] を選択して続行します。

6. [<stack-name> の確認] - スタックテンプレート、詳細、設定オプションを確認します。ページの下部にある quick-create link を開いて、これと同じ基本設定のスタックを作成することもできます。

   - [作成] を選択してスタックを作成します。
   - または、[変更セットの作成] を選択することもできます。変更セットは、スタックを作成する前に、このスタックがどのように設定されるかをプレビューするものです。これにより、変更セットを実行する前にさまざまな設定を調べることができます。

Amazon DocumentDB クラスターにアクセスする

CloudFormation スタックが完了したら、Amazon EC2 インスタンスを使用して Amazon DocumentDB クラスターに接続できます。SSH を使用して Amazon EC2 インスタンスに接続する方法については、「Amazon EC2 ユーザーガイド」の「Linux インスタンスへの接続」を参照してください。

接続後、Amazon DocumentDB の使用に関する情報を含む、次のセクションを参照してください。

- ステップ 3: データを挿入してクエリする
- Amazon DocumentDB クラスターとインスタンスの管理
- Amazon DocumentDB のモニタリングと口グ記録

終了保護と削除保護

Amazon DocumentDB のベストプラクティスとして、削除保護と終了保護を有効にすることをお勧めします。CloudFormation の終了保護は、Amazon DocumentDB の削除保護機能とは明確に異なる機能です。

- 終了保護 - CloudFormation スタックの終了保護を有効にすることによって、スタックが誤って削除されるのを防ぐことができます。終了保護を有効にした状態でスタックを削除しようとすると、

削除は失敗し、スタックは変更されません。CloudFormation を使用してスタックを作成すると、終了保護はデフォルトで無効になります。終了保護は、スタック作成時に有効にすることができます。詳細については、CloudFormation 「スタックオプションの設定」を参照してください。

- 削除保護 - Amazon DocumentDB は、クラスターに対して削除保護を有効にする機能も提供します。削除保護が有効になっている Amazon DocumentDB クラスターを削除しようとすると、削除は失敗し、クラスターは変更されません。削除保護を有効にする AWS CLIと、Amazon DocumentDB AWS マネジメントコンソール、、および CloudFormation からの偶発的な削除から保護されます。Amazon DocumentDB クラスターの削除保護の有効化、および無効化の詳細については 削除保護 を参照してください。

Amazon DocumentDB の移行とアップグレード

Amazon DocumentDB は、MongoDB API と互換性のあるフルマネージドデータベースサービスです。このセクションで詳しく説明するプロセスを使用して、オンプレミスまたは Amazon Elastic Compute Cloud (Amazon EC2)上で実行されている MongoDB データベースから Amazon DocumentDB にデータを移行できます。

トピック

- AWS Database Migration Service (DMS): クイックスタートガイドを使用して Amazon DocumentDB に移行する
- Amazon DocumentDB 移行ランブック
- Couchbase サーバーからの移行
- Amazon DocumentDB エンジンバージョンのサポート日
- Amazon DocumentDB のインプレースメジャーバージョンアップグレード
- Amazon DocumentDB マイナーバージョンアップグレード
- を使用した Amazon DocumentDB クラスターのアップグレード AWS Database Migration Service
- 長期サポート (LTS) リリースの使用
- Amazon DocumentDB 延長サポート

AWS Database Migration Service (DMS): クイックスタートガイドを使用して Amazon DocumentDB に移行する

トピック

- DMC ソースの準備
- DMS のセットアップ
- DocumentDB 圧縮を有効にする
- レプリケーションタスクを作成する
- 進行状況をモニタリングする
- 追加情報

DMC ソースの準備

DocumentDB 変更ストリームを有効にするか、MongoDB Oplog が DMS 変更データキャプチャ (CDC) をサポートできるようにするには、「変更ストリームの有効化」を参照してください。

- DMS ソースは、含まれているすべてのコレクションの DMS フルロードが完了するまで、進行中のすべての変更を保持する必要があります。
- DocumentDB 変更ストリームは時間ベースです。`change_stream_log_retention_duration` 設定がフルロードを完了する時間をカバーするのに十分な大きさであることを確認します。
- MongoDB Oplog は固定サイズです。フルロード中にすべてのオペレーションを保持するサイズであることを確認します。

DMS のセットアップ

DMS インスタンス、ソース、ターゲットエンドポイントを作成し、各エンドポイントをテストします。

DocumentDB 圧縮を有効にする

Amazon DocumentDB 5.0: Amazon DocumentDB クラスターにカスタムパラメータグループをアタッチし、 `default_collection_compression` パラメータを に設定することで、圧縮を有効にしますenabled。詳細については「コレクションレベルのドキュメント圧縮の管理」を参照してください。

Amazon DocumentDB 8.0: Zstd 圧縮はデフォルトで有効になっています。詳細については「Amazon DocumentDB 8.0 でのディクショナリベースの圧縮の管理」を参照してください。

レプリケーションタスクを作成する

1. DMS コンソールのナビゲーションペインで、[移行またはレプリケート] を選択し、[タスク] を選択します。
2. [Create task] (タスクの作成) を選択します。
3. [タスクの作成] ページの [タスク設定] セクションで、次の操作を行います。
    - 一意で意味のある [タスク識別子] (「mongodb-docdb-replication」など) を入力します。
    - 前に作成したソースエンドポイントを [ソースデータベースエンドポイント] ドロップダウンメニューで選択します。

- 前に作成したターゲットエンドポイントを [ターゲットデータベースエンドポイント] ドロップダウンメニューで選択します。
- [タスクタイプ] では、[移行とレプリケート] を選択します。

4. [設定] セクションで、次の操作を行います。

- [タスクログ] の場合は、[CloudWatch ログをオンにする] チェックボックスをオンにします。
- [編集モード] (セクションの上部) で、[JSON エディタ] を選択し、次の属性を設定します。
  - (`TargetMetadata` で) [`ParallelApplyThreads`] を「5」に設定します。これにより、CDC で 1 秒あたり約 1000 回の挿入/更新/削除オペレーションが可能になります。
  - (`FullLoadSettings` で) [`MaxFullLoadSubTasks`] を「16」に設定します。インスタンスのサイズに応じてこれを増やすことを検討してください。
  - 大規模なコレクション (100 GB 以上) では、(テーブルマッピングと `parallel-load` 属性で) 自動パーティションを有効にします。
    - "type": "partitions-auto"
    - "number-of-partitions": 16

進行状況をモニタリングする

AWS DMS コンソールを使用するか、カスタムダッシュボード (ダッシュボードツール) を作成して移行を追跡します。次のメトリクスに焦点を当てます。

- FullLoadThroughputBandwidthTarget — 移行のフルロードフェーズ中にターゲットデータベースにデータを転送するときに DMS が使用するネットワーク帯域幅 (KB/秒) を測定します。
- CDCLatencyTarget — ソースデータベースで変更が発生したときと、その変更がターゲットデータベースに適用されたときの間の遅延 (秒単位) を測定します。
- CDCThroughputRowsTarget — 移行の継続的なレプリケーションフェーズ中に DMS がターゲットデータベースに適用している 1 秒あたりの行数を測定します。

追加情報

Amazon DocumentDB と の詳細については AWS DMS、以下を参照してください。

- Amazon DocumentDB 移行ランブック
- MongoDB から Amazon DocumentDB への移行

Amazon DocumentDB 移行ランブック

このランブックは、 AWS Database Migration Service (DMS) を使用して MongoDB データベースを Amazon DocumentDB に移行するための包括的なガイドを提供します。これは、最初の検出から移行後の検証まで、エンドツーエンドの移行ジャーニーを通じてデータベース管理者、クラウドエンジニア、開発者をサポートするように設計されています。

MongoDB と Amazon DocumentDB の間の実装とサポートされている機能の違いを考慮すると、このランブックは構造化された体系的なアプローチを強調しています。重要な移行前評価を概説し、互換性に関する考慮事項を強調し、最小限の中断で移行を成功させるために必要な主要なタスクについて詳しく説明します。

ランブックは、以下のトピックで構成されています。

- 互換性 — Amazon DocumentDB でサポートされている MongoDB 機能とデータ型を理解し、潜在的な非互換性を特定します。
- ワークロードの検出 — 読み取り/書き込みパターン、データボリューム、パフォーマンスベースラインなど、既存の MongoDB ワークロードを分析します。
- インデックスの移行 — Amazon DocumentDB で最適なパフォーマンスを得るために MongoDB インデックスを抽出および変換するための戦略を分析します。
- ユーザー移行 — データベースユーザー、ロール、アクセスコントロールを Amazon DocumentDB に移行するためのアプローチについて詳しく説明します。
- データ移行 — 全ロードおよび変更データキャプチャ (CDC) など AWS DMS、 を使用したデータ移行のさまざまな方法について説明します。
- モニタリング — DMS またはネイティブツールを使用して移行する際のさまざまなモニタリングアプローチについて詳しく説明します。
- 検証 — 移行後のデータ整合性チェック、機能検証、パフォーマンス比較の手順を提供します。

このランブックのガイダンスに従うことで、チームはアプリケーションの機能を維持し、リスクを最小限に抑えながら、Amazon DocumentDB へのスムーズで安全かつ効率的な移行を確保できます。

互換性

トピック

- コア機能の互換性
- Amazon DocumentDB 互換性評価ツール

MongoDB から Amazon DocumentDB に移行する場合、移行を成功させるには、徹底的な初期評価と機能の互換性チェックが不可欠です。このプロセスは、集約パイプライン演算子、クエリパターン、インデックス、データモデルなど、MongoDB 機能の包括的なインベントリから始まります。

Amazon DocumentDB は MongoDB 3.6、4.0、5.0、8.0 API と互換性があるため、新しい MongoDB 固有の機能を使用するアプリケーションではリファクタリングが必要になる場合があります。評価する重要な領域には、シャーディングメカニズム (Amazon DocumentDB は異なるアプローチを使用) 、トランザクション実装、変更ストリームの機能、インデックスタイプ (特にスパースインデックスと部分インデックス) などがあります。

パフォーマンス特性も異なり、Amazon DocumentDB は予測可能なパフォーマンスを持つエンタープライズワークロード向けに最適化されています。テストでは、両方のシステムに対して代表的なワークロードを実行して、最適化が必要なクエリパターンを特定する必要があります。

評価フェーズでは、潜在的なパフォーマンスギャップを検出するための実行計画のモニタリングが重要です。これにより、明確な移行ロードマップを作成し、必要なアプリケーションの変更を特定し、スムーズな移行のための現実的なタイムラインを確立できます。

コア機能の互換性

包括的な機能サポート

- CRUD オペレーション — 一括演算子やクエリ演算子など、すべての基本的な作成、読み取り、更新、および削除オペレーションを完全にサポートし、シームレスなアプリケーションの互換性を実現します。
- 豊富なインデックス作成機能 — 単一フィールド、複合、TTL、部分、スパース、および 2dsphere インデックスの包括的なサポートを活用して、テキストベースのルックアップのクエリパフォーマンスとテキストインデックス (バージョン 5) を最適化します。
- エンタープライズグレードのレプリケーション — リードレプリカを備えた堅牢な自動フェイルオーバーメカニズムにより、運用上のオーバーヘッドなしで優れた高可用性を実現できます。
- 高度なバックアップソリューション — データ保護のためのポイントインタイムリカバリ (PITR) とオンデマンドの手動スナップショットを備えた自動バックアップシステムにより、安心して使用できます。

拡張 AWS統合機能

- 効率的な集約 — エンタープライズワークロードのパフォーマンスが最適化された、最も一般的に使用される集約ステージ (`$match`、`$group`、`$sort`、`$project` など) を活用します。

- トランザクションのサポート — マルチドキュメントトランザクションとマルチコレクショントランザクションを実装し、ほとんどのビジネスアプリケーションのニーズに最適です。
- リアルタイムデータ追跡 — シンプルなコマンドで変更ストリームを有効にし、リアルタイムのデータ変更モニタリングのためのシンプルなパラメータグループ設定で変更ストリームの保持期間を延長します。
- 位置情報ベースのサービス — `$geoNear` 演算子および 2dsphere インデックスをサポートする地理空間アプリケーションを実装します。
- テキスト検索機能 — コンテンツ検出のニーズに応じて、組み込みのテキスト検索機能を利用します。

最新のアーキテクチャの利点

- クラウドネイティブ設計 — MapReduce AWSなどのレガシー機能を、より効率的な集約パイプラインオペレーションに置き換える、エンゲージ最適化アーキテクチャ。
- セキュリティの強化 — AWS Identity and Access Management (IAM)、SCRAM-SHA-1、SCRAM-SHA-256、X.509 証明書認証、パスワードベースの認証のメリットがあります。
- 予測可能なパフォーマンス — 特にエンタープライズワークロード向けに最適化された一貫したパフォーマンスを体験できます。

Amazon DocumentDB の機能の包括的な概要については、「Amazon DocumentDB でサポートされている MongoDB API、オペレーション、およびデータ型」および「機能の違い: Amazon DocumentDB と MongoDB」を参照して、データベースの可能性を最大化してください。

Amazon DocumentDB は、MongoDB が提供するすべてのインデックスをサポートしているわけではありません。GitHub には、互換性を確認するための無料のインデックスツールが用意されています。インデックスツールを実行して非互換性を評価し、それに応じて回避策を計画します。

Amazon DocumentDB 互換性評価ツール

GitHub の Amazon DocumentDB Compatibility Tool はMongoDBワークロードを分析し、サポートされていない、または Amazon DocumentDB で変更が必要な演算子、コマンド、および機能をレポートします。3.6 から 8.0.1 (デフォルト: 8.0.1) までのターゲットバージョンをサポートしています。

推奨: URI モード (MongoDB 5.0 以降)

URI モードは MongoDB インスタンスに直接接続し、オペレーションをリアルタイムでサンプリングし、プロファイリングやログアクセスを必要としません。

代替方法

- ログベースの分析 — MongoDBプロファイラーログを解析します。実際のランタイム動作をキャプチャしますが、プロファイリングを有効にする必要があり、ログ記録期間のみをカバーします。
- ソースコード分析 — MongoDB API の使用についてアプリケーションのソースコードをスキャンします。包括的なカバレッジを提供しますが、未使用のコードパスにフラグを付け、動的に構築されたクエリを検出できない場合があります。

前提条件

- Python 3.7 以降
- ツールのクローンを作成します。 `git clone https://github.com/awslabs/amazon-documentdb-tools.git`
- 依存関係をインストールします。 `pip install -r compat-tool/requirements.txt`

使用状況の詳細と使用可能なすべてのオプションについては、GitHub の README を参照してください。

ワークロードの検出

MongoDB から Amazon DocumentDB に移行するには、既存のデータベースワークロードを完全に理解する必要があります。ワークロード検出は、データベースの使用パターン、データ構造、クエリパフォーマンス、運用上の依存関係を分析し、中断を最小限に抑えながらシームレスな移行を確保するプロセスです。このセクションでは、MongoDB から Amazon DocumentDB への効果的な移行を容易にするためのワークロード検出に関連する主要なステップを概説します。

トピック

- 既存の MongoDB デプロイの評価
- データモデルの違いを特定する
- クエリとパフォーマンスの分析
- セキュリティとアクセスコントロールの確認
- 運用とモニタリングに関する考慮事項

既存の MongoDB デプロイの評価

移行前に、以下を含む現在の MongoDB 環境を評価することが重要です。

- クラスターアーキテクチャ — ノード、レプリカセット、およびシャーディング設定の数を特定します。MongoDB から Amazon DocumentDB に移行する場合、Amazon DocumentDB はユーザー制御のシャーディングをサポートしていないため、MongoDB シャーディング設定を理解することが重要です。シャーディングされた MongoDB 環境用に設計されたアプリケーションでは、Amazon DocumentDB がストレージベースのアーキテクチャで異なるスケーリングアプローチを使用するため、アーキテクチャの変更が必要になります。Amazon DocumentDB に移行するときは、データ分散戦略を適応させ、場合によってはシャードコレクションを統合する必要があります。
- ストレージとデータボリューム — クラスターの合計データサイズとインデックスサイズを測定します。これを Oplog レビューツール で補完し、書き込みパターンとデータの増加速度を理解します。クラスターのサイジングの詳細については、「インスタンスのサイズ指定」を参照してください。
- ワークロードパターン — 読み取りおよび書き込みスループット、クエリ実行頻度、インデックス作成効率を分析します。
- 運用上の依存関係 — MongoDB に依存するすべてのアプリケーション、サービス、統合を文書化します。

データモデルの違いを特定する

Amazon DocumentDB は MongoDB と互換性がありますが、サポートされている機能には次のような違いがあります。

- トランザクション — Amazon DocumentDB は ACID トランザクションをサポートしますが、一部の 制限事項 があります。
- スキーマ設計 — ドキュメント構造、埋め込みドキュメント、およびリファレンスが Amazon DocumentDB のベストプラクティス と一致していることを確認します。

クエリとパフォーマンスの分析

クエリの動作を理解することで、移行と移行後のパフォーマンスを最適化できます。分析すべき主な領域は次のとおりです。

- スロークエリ — MongoDB のプロファイリングツールを使用して、実行時間が長いクエリを特定します。
- クエリパターン — CRUD オペレーションや集計など、一般的なクエリタイプを分類します。

- インデックスの使用 — インデックスが Amazon DocumentDB で効果的に使用されているか、最適化が必要かを評価します。Amazon DocumentDB でインデックスの使用を評価し、パフォーマンスを最適化するには、`$indexStats` 集約パイプラインステージを重要なクエリの `explain()` メソッドと組み合わせて使用します。まず、`db.collection.aggregate([{$indexStats{}}])` を実行して、使用されているインデックスを特定します。`explainPlan` を使用して最も頻繁にクエリを実行することで、より詳細な分析を行うことができます。
- 同時実行とワークロードの分散 — 読み取りと書き込みの比率、接続プーリング、パフォーマンスのボトルネックを評価します。

セキュリティとアクセスコントロールの確認

認証と認可

- MongoDB RBAC を Amazon DocumentDB IAM および RBAC にマッピング — MongoDB のロールベースのアクセスコントロールユーザーとロールを AWS Identity and Access Management (IAM) ポリシーと Amazon DocumentDB SCRAM 認証ユーザーにマッピングします。
- ユーザー移行戦略 — データベースユーザー、カスタムロール、および権限を Amazon DocumentDB でサポートされている認証メカニズムに移行する計画を立てます。
- 権限の違い — Amazon DocumentDB と同等のもの (クラスター管理ロールなど) を直接使用せずに MongoDB 権限を特定します。
- アプリケーション認証 —Amazon DocumentDB のパスワードポリシーに対応するように接続文字列と認証情報管理を更新します。Secrets Manager を使用して、認証情報を保存し、パスワードを更新できます。
- サービスアカウント管理 — でサービスアカウントの認証情報を管理するプロセスを確立します AWS Secrets Manager。
- 最小特権の実装 — アクセスコントロールを確認して絞り込み、新しい環境に最小特権の原則を実装します。

暗号化

保管中および転送中の暗号化がコンプライアンス要件と一致していることを確認します。

ネットワーク設定

仮想プライベートクラウド (VPC) のセットアップとセキュリティグループルールを計画します。

運用とモニタリングに関する考慮事項

システムの信頼性を維持するために、ワークロード検出には以下も含める必要があります。

- バックアップと復元戦略 — 既存のバックアップ方法と Amazon DocumentDB のバックアップ機能を評価します。
- AWS Backup 統合 — Amazon DocumentDB を含む AWS サービス全体で一元化されたバックアップ管理 AWS Backup に を活用します。
- CloudWatch メトリクス — MongoDB モニタリングメトリクスを、CPU、メモリ、接続、ストレージの Amazon DocumentDB CloudWatch メトリクスにマッピングします。
- Performance Insights — Amazon DocumentDB Performance Insights を実装して、データベースの負荷を視覚化し、詳細なクエリ分析でパフォーマンスの問題を分析します。
- プロファイラー — 実行速度の遅いオペレーションをキャプチャするように Amazon DocumentDB プロファイラーを設定します (MongoDB のプロファイラーに似ていますが、Amazon DocumentDB 固有の設定があります)。
  - 適切なしきい値を持つパラメータグループを使用して有効にします。
  - プロファイラーデータを分析して最適化の機会を特定する
- CloudWatch Events — Amazon DocumentDB クラスターイベントのイベント駆動型モニタリングをセットアップします。
  - バックアップイベント、メンテナンスウィンドウ、およびフェイルオーバーの通知を設定します。
  - アラートおよび自動応答のために Amazon SNS と統合 AWS Lambda します。
- 監査ログ記録 — ユーザーアクティビティとセキュリティ関連のイベントを追跡するための監査ログ記録設定を計画します。
- 拡張モニタリング — 1 秒間隔できめ細かな OS レベルのメトリクスの拡張モニタリングを有効にします。

新しいターゲットクラスターを作成する

の手順に従って Amazon DocumentDB クラスターを作成しますAmazon DocumentDB クラスターの作成。クラスターを移行用に設定するときは、次の推奨事項を適用します。

- インスタンスクラス: 最高のパフォーマンスを得るには R8G を選択します。ターゲットリージョンで R8G が使用できない場合は、R6G を使用します。最適なフルロードスループットを得るには、インスタンスをできるだけ大きく選択します。移行が完了したらスケールダウンします。

- エンジンバージョン: 最新 (8.0.1)
- インスタンスの数: インスタンスを 1 つ選択して、移行中のコストを最小限に抑えます。フルロード移行が完了したら、高可用性のために 3 インスタンスにスケールします。
- ネットワーク設定: Amazon DocumentDB セキュリティグループで、DMS レプリケーションインスタンスのセキュリティグループ (オンライン移行の場合) または EC2 移行ホスト (mongorestore を使用したオフライン移行の場合) からのインバウンド接続が許可されていることを確認します。

インデックスの移行

MongoDB から Amazon DocumentDB に移行するには、データだけでなくインデックスも転送して、クエリパフォーマンスを維持し、データベースオペレーションを最適化する必要があります。このセクションでは、互換性と効率を確保しながら、MongoDB から Amazon DocumentDB にインデックスを移行するための詳細なステップ倍ステップのプロセスを概説します。

Amazon DocumentDB インデックスツールの使用

GitHub でインデックスツールのクローンを作成する

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

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

MongoDB からインデックスをエクスポートする (MongoDB から移行する場合)

```
python3 migrationtools/documentdb_index_tool.py \
  --dump-indexes \
  --dir index_export \
  --uri 'mongodb://<username>:<password>@<source_endpoint>:27017'
```

インデックスを検証する

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

インデックスをインポートする

```
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'
```

ユーザー移行

MongoDB から Amazon DocumentDB にユーザーを移行することは、アクセスコントロール、認証、データベースセキュリティを維持する上で不可欠です。このセクションでは、Amazon MongoDB ユーザーを、 Amazon DocumentDB エクスポートユーザーツールを使用して、彼らのロールとアクセス許可を保持しながら正常に移行する詳細な手順を概説します。

Amazon DocumentDB ユーザーエクスポートツールの使用

Export Users tool は、ユーザーとロールを MongoDB または Amazon DocumentDB から JavaScript ファイルにエクスポートし、それを別のクラスターで再作成するために使用できます。

前提条件

```
# 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
```

ステップ 1: ユーザーとロールをエクスポートする

```
# 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"
```

ステップ 2: ユーザーファイルを編集する

セキュリティ上の理由から、パスワードはエクスポートされません。mongodb-users.js ファイルを開き、各createUserステートメントのpwd値を置き換えて、各ユーザーのパスワードを追加します。

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

ステップ 3: Amazon DocumentDB にカスタムロールを復元する

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

ステップ 4: Amazon DocumentDB にユーザーを復元する

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

重要な注意事項

- セキュリティ上の理由からパスワードはエクスポートされないため、users.js ファイルに手動で追加する必要があります。
- 適切なロールの割り当てを保証するには、ユーザーより前にロールをインポートする必要があります。
- このツールは、mongosh シェルで直接実行できる JavaScript ファイルを生成します。
- カスタムロールとその権限は移行中も保持されます。
- このアプローチにより、インポート前にユーザーアクセス許可を確認および変更できます。

この方法は、ユーザーとロールを MongoDB から Amazon DocumentDB に移行するための安全で柔軟なアプローチを提供し、移行プロセス中にパスワードをリセットできるようにします。

データ移行

トピック

- オンライン移行
- オフライン移行

オンライン移行

このセクションでは、最小限のダウンタイムと継続的なレプリケーションを可能にする AWS DMS ために、 を使用して MongoDB から Amazon DocumentDB へのオンライン移行を実行する詳細な手順について説明します。まず、Amazon DocumentDB クラスターをターゲットとしてセットアップし、MongoDB インスタンスがソースとして適切に設定されていることを確認します。通常、変更データキャプチャにはレプリカセットモードが必要です。次に、DMS レプリケーションインスタンスを作成し、必要な接続の詳細を使用してソースエンドポイントとターゲットエンドポイントを定義します。エンドポイントを検証したら、フルデータロード、継続的なレプリケーション、またはその両方を含む移行タスクを設定して開始します。

新しいターゲットクラスターを作成する

「新しいターゲットクラスターを作成する」を参照してください。

ソースを設定する

MongoDB と Amazon DocumentDB は、シナリオに応じて、どちらも移行ソースとして機能します。

- ソースとしての MongoDB — オンプレミスまたはセルフマネージド MongoDB から Amazon DocumentDB または他の AWS データベースサービスに移行する場合に一般的です。移行中の変更データキャプチャをサポートするために、適切なサイズの oplog を使用してレプリカセットモードで実行する必要があります (フルロード中にすべてのオペレーションを保持するサイズになっていることを確認してください)。
- ソースとしての Amazon DocumentDB — 通常、クロスリージョンレプリケーション、バージョンアップグレード、MongoDB Atlas などの他のデータベースサービスへの移行に使用されます。クラスターパラメータグループの `change_stream_log_retention_duration` パラメータを設定して、移行中に進行中の変更をキャプチャするために 変更ストリームの有効化 が必要で

す。change_stream_log_retention_duration 設定がフルロードの完了に必要な時間をカバーするのに十分な大きさであることを確認します。

移行を開始する前に、 AWS DMS アクセスを許可するようにソースを設定します。

適切なアクセス許可を持つ MongoDB ユーザーを作成します。

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

ネットワークと認証を設定します。

MongoDB から DMS への移行用のネットワーク接続を設定する場合:

EC2 がホストする MongoDB ソース

- EC2 セキュリティグループを変更して、DMS レプリケーションインスタンスのセキュリティグループからのインバウンドトラフィックを許可します。
- TCP ポート 27017 (またはカスタム MongoDB ポート) のルールを追加します。
- DMS レプリケーションインスタンスのセキュリティグループ ID を正確なアクセスコントロールのソースとして使用します。
- EC2 インスタンスのサブネットに DMS レプリケーションインスタンスのサブネットへのルートがあることを確認します。

オンプレミス MongoDB ソース

- DMS レプリケーションインスタンスのパブリック IP アドレスからのインバウンド接続を許可するようにファイアウォールを設定します。
- Direct Connect または VPN を使用している場合は、ネットワークと DMS インスタンスを含む VPC 間の適切なルーティングを確認してください。
- DMS サブネットから MongoDB サーバーへの telnet または nc コマンドを使用して接続をテストします。

MongoDB Atlas ソース

- DMS レプリケーションインスタンスの IP アドレスを MongoDB Atlas IP 許可リストに追加します。
- Atlas が実行されている場合は AWS 、VPC と MongoDB Atlas VPC 間の VPC ピアリングを設定します AWS。
- 別のクラウドプロバイダーで実行されている場合は、プライベート接続 (エンタープライズ階層) AWS PrivateLink を設定します。
- 適切な読み取り/書き込みアクセス許可を持つ専用ユーザーを作成します。
- SSL モードを「verify-full」に設定して MongoDB Atlas 接続文字列を使用します。
- 移行期間に十分な oplog サイズを確保します。

Amazon DocumentDB ソース

DMS レプリケーションインスタンスのセキュリティグループからのインバウンドトラフィックを許可するように、ソース Amazon DocumentDB セキュリティグループを設定します。

DMS レプリケーションインスタンスを作成する

GitHub の DMS Buddy を使用して、最適な DMS 設定とインスタンスサイズで最適な移行インフラストラクチャを作成します。手動で設定する場合、次の手順に従います。

1. AWS DMS コンソールのナビゲーションペインで、移行またはレプリケートを選択し、プロビジョニングされたインスタンスを選択します。
2. レプリケーションインスタンスの詳細を入力します。
   - [インスタンス名]: 一意の名前を選択します。
   - [インスタンスクラス]: ワークロードに基づいて選択します。例: dms.r7i.large (小規模なワークロード) 、dms.r7i.4xlarge (大規模なワークロード) 。
   - [エンジンバージョン]: 3.5.4
   - 割り当てられたストレージ: デフォルトは 50 GB (必要に応じて増加) です。これは、ドキュメントのサイズ、更新/秒、およびフルロード時間によって決まります。
   - マルチ AZ 配置: 必要に応じて高可用性を有効にします。
   - Amazon DocumentDB と同じ VPC を選択します。
   - セキュリティグループ がソースと Amazon DocumentDB からのインバウンドトラフィックを許可していることを確認します。
3. レプリケーションインスタンスの作成を選択し、ステータスが使用可能になるまで待ちます。

DMS エンドポイントを作成する

ソースエンドポイントを作成する

MongoDB ソースの場合

1. DMS コンソールのナビゲーションペインで、[移行またはレプリケート] を選択し、[エンドポイント]を選択します。
2. エンドポイントの作成 を選択します。
3. [エンドポイントの作成] ページで、[ソースエンドポイント] を選択します。
4. [エンドポイント設定] セクションで、次の操作を行います。

   - 一意で意味のある エンドポイント識別子 (「mongodb-source」など) を入力します。
   - [ソースエンジン] として MongoDB を選択します。
   - [Access to endpoint database] (エンドポイントデータベースへのアクセス) では、[Provide access information manually] (アクセス情報を手動で提供) を選択します。
   - [サーバー名] では、*[MongoDB ##### DNS #/IP ####]* を入力します。
   - [ポート] では、27017 (デフォルトの MongoDB ポート) と入力します。
   - [認証モード] では、アプリケーションに適したモード (パスワード/SSL) を選択します (デフォルトは Secrets Manager)。
   - [認証モード] が [パスワード] の場合は、以下を指定します。
     - [ユーザー名] と [パスワード]: MongoDB 認証情報を入力します。
     - [データベース名]: ソースデータベースの名前。
     - [認証メカニズム]: SCRAM-SHA-1 (デフォルト) または適切なメカニズム
5. [メタデータモード] では、[ドキュメント] のデフォルト設定のままにします。
6. 追加の接続属性:

   - authSource=admin (認証データベースが異なる場合)
   - replicaSet=<your-replica-set-name> (CDC に必須)

Amazon DocumentDB ソースの場合

1. DMS コンソールのナビゲーションペインで、[移行またはレプリケート] を選択し、[エンドポイント]を選択します。
2. エンドポイントの作成 を選択します。

3. [エンドポイントの作成] ページで、[ソースエンドポイント] を選択します。
4. [エンドポイント設定] セクションで、次の操作を行います。
   - 一意で意味のある エンドポイント識別子 (「docdb-sourc」など) を入力します。
   - [ソースエンジン] として Amazon DocumentDB を選択します。
   - [Access to endpoint database] (エンドポイントデータベースへのアクセス) では、[Provide access information manually] (アクセス情報を手動で提供) を選択します。
   - [サーバー名] では、*[### Amazon DocumentDB ############]* を入力します。
   - [ポート] では、27017 (Amazon DocumentDB のデフォルトのポート) を入力します。
   - [SSL モード] では、verify-full (Amazon DocumentDB に推奨) を選択します。
   - [CA 証明書] では、Amazon RDS ルート CA 証明書を選択します。
   - [認証モード] では、アプリケーションに適したモード (パスワード/SSL) を選択します (デフォルトは Secrets Manager)。
   - [認証モード] が [パスワード] の場合は、以下を指定します。
     - [ユーザー名] と [パスワード]: Amazon DocumentDB 認証情報を入力します。
     - [データベース名]: ソースデータベースの名前。
     - [認証メカニズム]: SCRAM-SHA-1 (デフォルト) または適切なメカニズム
5. [メタデータモード] では、[ドキュメント] のデフォルト設定のままにします。

ターゲットエンドポイント (Amazon DocumentDB) を作成する

1. DMS コンソールのナビゲーションペインで、[移行またはレプリケート] を選択し、[エンドポイント]を選択します。
2. エンドポイントの作成 を選択します。
3. [エンドポイントの作成] ページで、[ターゲットエンドポイント] を選択します。
4. [エンドポイント設定] セクションで、次の操作を行います。
   - 一意で意味のある エンドポイント識別子 (「docdb-target」など) を入力します。
   - [ターゲットエンジン] として [Amazon DocumentDB] を選択します。
   - [エンドポイントデータベースへのアクセス] で、データベースへのアクセスを認証するために使用する方法を選択します。
     - [AWS シークレットマネージャー] を選択した場合は、[シークレット] フィールドに Amazon DocumentDB 認証情報を保存するシークレットを選択します。

- [アクセス情報を手動で提供] を選択する場合:
  - [サーバー名] では、*[##### Amazon DocumentDB ############]* を入力します。
  - [ポート] では、27017 (Amazon DocumentDB のデフォルトのポート) を入力します。
  - [SSL モード] では、verify-full (Amazon DocumentDB に推奨) を選択します。
  - [CA 証明書] では、SSL 検証用の CA 証明書バンドルをダウンロードして指定します。
  - [認証モード] では、アプリケーションに適したモード (パスワード/SSL) を選択します (デフォルトは Secrets Manager)。
  - [認証モード] が [パスワード] の場合は、以下を指定します。
    - [ユーザー名] と [パスワード]: Amazon DocumentDB 認証情報を入力します。
    - [データベース名]: ソースデータベースの名前。
    - [認証メカニズム]: SCRAM-SHA-1 (デフォルト) または適切なメカニズム

5. [メタデータモード] では、[ドキュメント] のデフォルト設定のままにします。

レプリケーションタスクを作成する

1. DMS コンソールのナビゲーションペインで、[移行またはレプリケート] を選択し、[タスク] を選択します。
2. [Create task] (タスクの作成) を選択します。
3. [タスクの作成] ページの [タスク設定] セクションで、次の操作を行います。
   - 一意で意味のある [タスク識別子] (「mongodb-docdb-replication」など) を入力します。
   - 前に作成したソースエンドポイントを [ソースデータベースエンドポイント] ドロップダウンメニューで選択します。
   - 前に作成したターゲットエンドポイントを [ターゲットデータベースエンドポイント] ドロップダウンメニューで選択します。
   - プロビジョニングされたインスタンスのドロップダウンメニューで、前に作成したレプリケーションインスタンスを選択します。
   - [タスクタイプ] では、[移行とレプリケート] を選択します。
   - ソースからターゲットへのレプリケーションを継続する予定期間はどのくらいですか? で、期間限定で を選択します。
4. [設定] セクションで、次の操作を行います。

- ターゲットテーブルの準備モードでは、何もしないを選択します。それ以外の場合、既に作成されたコレクションとインデックスは削除されます。
- [タスクログ] の場合は、[CloudWatch ログをオンにする] チェックボックスをオンにします。
- 他のすべての設定にはデフォルト値のままにします。

5. 設定セクションの上部に戻り、編集モードで JSON エディタを選択し、次の属性を設定します。

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

コレクション数とインスタンス容量MaxFullLoadSubTasksに基づいて を設定します。少数または少数のコレクションには低い値 (2～4) を使用し、多くの大規模なコレクションには高い値 (8～16) を使用します。自動サイジングについては、GitHub の「DMS Buddy」を参照してください。

6. [テーブルマッピング] セクションで、新しい選択ルールを追加します。

- [スキーマ名] では、移行するソースデータベースを追加します。% を使用して複数のデータベースを指定します。
- [スキーマテーブル] 名では、移行するソースコレクションを追加します。% を使用して複数のコレクションを指定します。
- [アクション] では、デフォルト設定の [含める] のままにします。

7. 大規模なコレクション (100GB 以上) では、[テーブル設定ルール] を追加します。

- [スキーマ名] では、移行するソースデータベースを追加します。% を使用して複数のデータベースを指定します。
- [スキーマテーブル] 名では、移行するソースコレクションを追加します。% を使用して複数のコレクションを指定します。
- パーティションの数については、ワークロードに基づいて を設定します。少数の大規模なコレクションの場合、パーティション ≥ を設定MaxFullLoadSubTasksして、すべてのワーカーをビジー状態に保ちます。並行してロードする多くのコレクションでは、コレクション数

自体MaxFullLoadSubTasksが十分な並列処理を提供するため、パーティション ≤ を設定します。

8. [移行前評価] セクションでは、オフになっていることを確認します。移行前評価は、ターゲットとしての Amazon DocumentDB ではサポートされていないため、無効にする必要があります。

オフライン移行

このセクションでは、mongodump および mongorestore のネイティブ MongoDB ツールを使用して、セルフマネージド MongoDB インスタンスから Amazon DocumentDB へのオフライン移行を実行するプロセスを概説します。

前提条件

ソース MongoDB の要件

- 適切なアクセス許可を持つソース MongoDB インスタンスへのアクセス。
- 必要に応じて mongodump をインストールします (MongoDB のインストール中にインストールされます)。
- ダンプファイルに十分なディスク容量があることを確認します。

ターゲット Amazon DocumentDB の要件

- Amazon DocumentDB クラスターがプロビジョニングされていることを確認します。
- 移行を容易にするために、Amazon DocumentDB と同じ VPC に EC2 インスタンスがあることを確認します。
- ネットワーク接続は、ソース環境と Amazon DocumentDB の間で利用できる必要があります。
- mongorestore は、移行 EC2 インスタンスにインストールする必要があります。
- Amazon DocumentDB にアクセスするには、適切な IAM アクセス許可を設定する必要があります。

一般的な要件

- AWS CLI を設定する必要があります (中間ストレージに AWS サービスを使用している場合)
- データ転送には十分な帯域幅が必要です。
- ダウンタイムウィンドウを承認する必要があります (ライブ移行を行う場合は、他のアプローチを検討してください)

Amazon DocumentDB クラスターを準備する

「新しいターゲットクラスターを作成する」を参照してください。

データダンプを実行する (mongodump)

ダンプファイルを作成するために、次のいずれかのオプションを選択します。

- オプション 1: 基本

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

- オプション 2: より良い制御とパフォーマンス

  gzip で出力を圧縮し、コレクションを並行してダンプし、TLS を有効にして、使用可能な場合はセカンダリレプリカから読み取ります。

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

- オプション 3: 大規模なデータベース

  特定のデータベースとコレクションをダンプし、クエリでドキュメントをフィルタリングし、1 つの圧縮アーカイブに出力します。

```
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
```

ダンプファイルを復元環境に転送する

ダンプサイズに基づいて適切な方法を選択します。

- 小規模 — 移行マシン (前に作成した EC2 インスタンス) に直接コピーします。

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

- 中規模 — Amazon S3 を中間ストレージとして使用します。

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

- 大規模 — 非常に大規模なデータベースの場合は、 AWS DataSync または物理的な転送を検討してください。

Amazon DocumentDB (mongorestore) にデータを復元する

復元プロセスを開始する前に、Amazon DocumentDB でインデックスを作成します。Amazon DocumentDB インデックスツール を使用して、インデックスをエクスポートおよびインポートできます。

データを復元するには、以下のいずれかのオプションを選択します。

- オプション 1: 基本的な復元

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

- オプション 2: より良い制御とパフォーマンス

  並列コレクションとドキュメント挿入を使用して gzip 圧縮ダンプを復元します。インデックスが事前に作成されているため、インデックスの復元をスキップします。

```
mongorestore \
  --uri="mongodb://<user>:<password>@<target_endpoint>:27017" \
```

```
  --tls \
  --tlsCAFile=global-bundle.pem \
  --gzip \
  --numParallelCollections=4 \
  --numInsertionWorkersPerCollection=4 \
  --noIndexRestore \
  /path/to/dump
```

- オプション 3: 大規模なデータベースまたは特定のコントロール

  特定のデータベースとコレクションを含めるか除外するために、名前空間フィルタリングを使用して圧縮アーカイブから復元します。インデックスが事前に作成されているため、インデックスの復元をスキップします。

```
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
```

モニタリング

このセクションでは、進行中の移行の進行状況、パフォーマンス、状態を追跡するための詳細なモニタリングプロセスについて説明します。

モニタリング手順は、移行方法 (AWS DMS、mongodump/mongorestore、またはその他のツール) に関係なく適用されます。

AWS DMS 移行モニタリング (該当する場合)

次の主要な CloudWatch メトリクスをモニタリングします。

フルロードフェーズのメトリクス

- FullLoadThroughputBandwidthTarget — フルロード中のネットワーク帯域幅 (KB/秒)

- FullLoadThroughputRowsTarget — 1 秒あたりにロードされた行/ドキュメントの数
- FullLoadThroughputTablesTarget — 1 分あたりに完了したテーブル/コレクションの数
- FullLoadProgressPercent — フルロード完了の割合
- TablesLoaded — 正常にロードしたテーブル/コレクションの数
- TablesLoading — 現在ロード中のテーブル/コレクションの数
- TablesQueued — ロードを待機しているテーブル/コレクションの数
- TablesErrored — ロードに失敗したテーブル/コレクションの数

CDC フェーズメトリクス

- CDCLatencyTarget — ソース変更とターゲット適用間の遅延時間 (秒)
- CDCLatencySource — ソースの変更とそれを読み取る DMS の間の遅延時間 (秒)
- CDCThroughputRowsTarget — 継続的なレプリケーション中に適用される 1 秒あたりの行数
- CDCThroughputBandwidthTarget — CDC 中のネットワーク帯域幅 (KB/秒)
- CDCIncomingChanges — ソースから受信した変更イベントの数
- CDCChangesMemoryTarget — 変更をターゲット側に保存するために使用するメモリ (MB)

リソースメトリクス

- CPUUtilization — レプリケーションインスタンスの CPU 使用率
- FreeableMemory — レプリケーションインスタンスで使用可能なメモリ
- FreeStorageSpace — レプリケーションインスタンスで使用可能なストレージ
- NetworkTransmitThroughput — レプリケーションインスタンスのネットワークスループット
- NetworkReceiveThroughput — レプリケーションインスタンスのネットワークスループット

エラーメトリクス

- ErrorsCount — 移行中のエラーの合計数
- TableErrorsCount — テーブル固有のエラーの数
- RecordsErrorsCount — レコード固有のエラーの数

移行パフォーマンスが低下した場合に通知を受け取るには、 `CDCLatencyTarget` や `CPUUtilization` などの重要なメトリクスの CloudWatch アラームを作成します。

DMS ログ (CloudWatch ログ)

1. Amazon CloudWatch Logs コンソールに移動する。
2. ロググループで検索して選択します。「dms-tasks –」のようになります。
3. エラー情報を含む可能性のあるログストリームを探します。
   - 名前に「error」が含まれるストリーム
   - タスク ID またはエンドポイント名を含むストリーム
   - 移行中の最新のログストリーム
4. これらのストリーム内で、次のようなキーワードを検索します。
   - 「エラー」
   - 「例外」
   - 「失敗」
   - 「警告」

DMS タスクのステータス (使用 AWS CLI)

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

予想されるステータスフロー:

作成中 → 準備完了 → 実行中 → 停止中 → 停止 (または失敗)

docdb-dashboarder の使用によるモニタリング

この docdb-dashboarder ツールは、重要なパフォーマンスメトリクスを含む CloudWatch ダッシュボードを自動的に生成することで、Amazon DocumentDB クラスターの包括的なモニタリングを提供します。これらのダッシュボードには、重要なクラスターレベルのメトリクス (レプリカラグ、オペレーションカウンター)、インスタンスレベルのメトリクス (CPU、メモリ、接続)、ストレージメトリクス (ボリューム使用量、バックアップストレージ) が表示されます。移行シナリオの場合、このツールは、CDC レプリケーションラグや運用率などのメトリクスを使用して移行の進行状況を追跡する特殊なダッシュボードを提供します。ダッシュボードでは、複数のクラスターを同時にモニタリングし、NVMe-backed インスタンスのサポートを含めることができます。これらのメト

リクスを視覚化することで、チームはパフォーマンスのボトルネックをプロアクティブに特定し、リソース割り当てを最適化し、Amazon DocumentDB デプロイの円滑な運用を確保できます。このツールを使用すると、ダッシュボードを手動で作成する必要がなくなり、すべての環境で一貫したモニタリングが可能になります。セットアップ手順と高度な設定オプションについては、Amazon DocumentDB Dashboarder ツール GitHub リポジトリを参照してください。

検証

トピック

- 検証チェックリスト
- スキーマとインデックスの検証
- データサンプリングとフィールドレベルの検証
- DataDiffer ツールを使用した検証

このセクションでは、移行後のデータの整合性、整合性、アプリケーションの互換性を確保するための詳細な検証プロセスについて説明します。

検証手順は、移行方法 (AWS DMS、mongodump/mongorestore、またはその他のツール) に関係なく適用されます。

検証チェックリスト

各コレクションのドキュメント数がソースとターゲットの間で一致することを検証します。

MongoDB ソース

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

Amazon DocumentDB ターゲット

```
mongosh \
  --host <target_endpoint> \
  --port 27017 \
  --username <user> \
  --password <password> \
```

```
  --tls \
  --tlsCAFile global-bundle.pem \
  --eval "db.<collection>.countDocuments()"
```

スキーマとインデックスの検証

以下を確認してください。

- すべてのコレクションがターゲットに存在します。
- インデックスが正しくレプリケートされています。
- スキーマ定義 (強制されている場合) が同じです。

コレクションを確認する (ソースとターゲット)

```
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();"
```

インデックスを確認する (ソースとターゲット)

```
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()"
```

コレクションのリストを比較して、欠落しているコレクションや余分なコレクションがないことを確認します。

インデックス名、キー定義、一意の制約、TTL インデックス (存在する場合) を確認して、インデックスを検証します。

スキーマ検証ルールを確認する (MongoDB でスキーマ検証を使用する場合)

```
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"
```

データサンプリングとフィールドレベルの検証

ドキュメントをランダムにサンプリングし、ソースとターゲットのフィールドを比較できます。

手動サンプリング

5 つのランダムドキュメント (ソース) を取得します。

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

同じドキュメント ID (ターゲット) を取得します。

```
mongosh --host <target_endpoint> \
  --port 27017 \
  --username <username> \
  --password <password> \
  --tls --tlsCAFile global-bundle.pem \
```

```
  --eval "db.<collection>.find({ _id: { \$in: [<list_of_ids>] } })"
```

自動サンプリング

```
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']}")
```

DataDiffer ツールを使用した検証

GitHub の DataDiffer ツールは、ソースデータベースとターゲットデータベース間でデータを比較する信頼性の高い方法を提供します。

前提条件

DataDiffer ツールをインストールする前に、次の前提条件を満たす必要があります。

- Python 3.7 以降
- PyMongo ライブラリ
- ソース MongoDB クラスターとターゲット Amazon DocumentDB クラスターの両方へのネットワーク接続

セットアップとインストール

リポジトリのクローンを作成し、DataDiffer ディレクトリに移動する

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

```
cd amazon-documentdb-tools/migration/data-differ
```

必要な依存関係をインストールする

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

データ検証の実行

接続の詳細を含む設定ファイル (config.json など) を作成する

```
{
  "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
  }
}
```

DataDiffer ツールを実行する

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

大規模なコレクションの場合、サンプリングを使用してデータのサブセットを検証する

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

複数のコレクションを検証するには、個別の設定ファイルを作成するか、バッチモードを使用する

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

結果の解釈

ツールの出力は次のとおりです。

- ソースとターゲットの合計ドキュメント数
- 一致するドキュメントの数
- 欠落しているドキュメントの数
- 違いがあるドキュメントの数
- 違いの詳細レポート (存在する場合)

ベストプラクティス

DataDiffer ツールを使用する際のベストプラクティスは次のとおりです。

- 段階的に実行する — まずドキュメント数を検証し、次に主要なドキュメントのサンプルを作成し、最後に必要に応じて完全な比較を実行します。
- スキーマの違いを確認する — Amazon DocumentDB には、MongoDB と比較していくつかの制限があります。ツールでは、互換性のないデータ型または構造が強調表示されます。
- クワイエットピリオド中の検証 — 書き込みオペレーションが最小限であるときに検証を実行して整合性を確保します。
- リソース使用状況のモニタリング — 比較プロセスはリソースを大量に消費する可能性があります。それに応じてバッチサイズとスレッド数を調整します。
- インデックスの検証 — データの検証後、必要なインデックスがすべてターゲット Amazon DocumentDB クラスターに作成されていることを確認します。
- ドキュメントの検証結果 — 移行ドキュメントの一部として、各コレクションの検証結果を記録します。

Couchbase サーバーからの移行

トピック

- 序章
- Amazon DocumentDB との比較
- 発見
- 計画

- 移行
- 検証

序章

このガイドでは、Couchbase Server から Amazon DocumentDB への移行時に考慮すべき重要なポイントについて説明します。ここでは、移行の検出、計画、実行、検証の各フェーズに関する考慮事項について説明します。また、オフライン移行とオンライン移行を実行する方法についても説明します。

Amazon DocumentDB との比較

	Couchbase サーバー	Amazon DocumentDB
データ組織	バージョン 7.0 以降では、データはバケット、スコープ、コレクションに整理されます。以前のバージョンでは、データはバケットに整理されています。	データはデータベースとコレクションに整理されます。
互換性	サービスごとに個別の APIs (データ、インデックス、検索など) があります。セカンダリルックアップは、ANSI 標準 SQL に基づくクエリ言語である SQL++ (以前は N1QL と呼ばれていました) を使用するため、多くの開発者が使い慣れています。	Amazon DocumentDB は MongoDB API と互換性があります。
アーキテクチャ	ストレージは各クラスターインスタンスにアタッチされます。コンピューティングをストレージとは別にスケールすることはできません。	Amazon DocumentDB は、クラウド向けに設計されており、従来のデータベースアーキテクチャの制限を回避します。コンピューティングレイヤーとストレージレイヤーは Amazon DocumentDB で分離され、コンピューティングレイヤーはストレージとは別に

	Couchbase サーバー	Amazon DocumentDB
		スケーリングできます。 Amazon DocumentDB
オンデマンドで読み取り容量を追加する	クラスターは、インスタンスを追加することでスケールアウトできます。ストレージはサービスが実行されているインスタンスにアタッチされるため、スケールアウトにかかる時間は、新しいインスタンスに移動するか、再調整する必要があるデータの量によって異なります。	Amazon DocumentDB クラスターに最大 15 個の Amazon DocumentDB レプリカを作成することで、Amazon DocumentDB クラスターの読み取りスケーリングを実現できます。ストレージレイヤーには影響しません。
ノード障害からの迅速な復旧	クラスターには自動フェイルオーバー機能がありますが、クラスターを完全な強度に戻す時間は、新しいインスタンスに移動する必要があるデータの量によって異なります。	Amazon DocumentDB は、通常 30 秒以内にプライマリをフェイルオーバーし、クラスター内のデータ量に関係なく、8～10 分でクラスターを完全な強度に戻すことができます。
データの増加に応じてストレージをスケールする	セルフマネージド型クラスターの場合、ストレージと IOs は自動的にスケーリングされません。	Amazon DocumentDB ストレージと IOsスケーリングされます。
パフォーマンスに影響を与えずにデータをバックアップする	バックアップはバックアップサービスによって実行され、デフォルトでは有効になっていません。ストレージとコンピューティングは分離されていないため、パフォーマンスに影響を与える可能性があります。	Amazon DocumentDB バックアップはデフォルトで有効になっており、オフにすることはできません。バックアップはストレージレイヤーによって処理されるため、コンピューティングレイヤーには影響しません。Amazon DocumentDB は、クラスタースナップショットからの復元と特定の時点への復元をサポートしています。

	Couchbase サーバー	Amazon DocumentDB
データ耐久性	クラスター内のデータには最大 3 つのレプリカコピーがあり、合計 4 つのコピーがあります。データサービスが実行されている各インスタンスには、アクティブなデータのコピーが 1、2、または 3 つあります。	Amazon DocumentDB は、書き込みクォーラムが 4 のコンピューティングインスタンスの数に関係なく、データの 6 つのコピーを保持し、true のままにします。ストレージレイヤーがデータのコピーを 4 つ保持すると、クライアントは確認を受け取ります。
整合性	K/V オペレーションの即時整合性がサポートされています。Couchbase SDK は、K/V リクエストをデータのアクティブなコピーを含む特定のインスタンスにルーティングするため、更新が確認されると、クライアントはその更新を読み取ることが保証されます。他のサービス (インデックス、検索、分析、イベント) への更新のレプリケーションは、結果整合性があります。	Amazon DocumentDB レプリカは結果整合性があります。即時整合性読み取りが必要な場合は、クライアントはプライマリインスタンスから読み取ることができます。
レプリケーション	クロスデータセンターレプリケーション (XDCR) は、多対多トポロジでデータのフィルタリングされたアクティブ/パッシブ/アクティブ/アクティブレプリケーションを提供します。	Amazon DocumentDB グローバルクラスターは、1:many (最大 10) トポロジでアクティブ/パッシブレプリケーションを提供します。

発見

Amazon DocumentDB に移行するには、既存のデータベースワークロードを完全に理解する必要があります。ワークロード検出は、Couchbase クラスターの設定と、データセット、インデックス、

ワークロードなどの運用特性を分析し、中断を最小限に抑えながらシームレスな移行を確保するプロセスです。

クラスターの設定

Couchbase は、各機能がサービスに対応するサービス中心のアーキテクチャを使用します。Couchbase クラスターに対して次のコマンドを実行して、どのサービスが使用されているかを判断します (「Getting Information on Nodes」を参照)。

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

サンプル出力:

```
[
  "backup",
  "cbas",
  "eventing",
  "fts",
  "index",
  "kv",
  "n1ql"
]
```

Couchbase サービスには以下が含まれます。

データサービス (kv)

データサービスは、メモリとディスクのデータへの読み取り/書き込みアクセスを提供します。

Amazon DocumentDB は、MongoDB API を介した JSON データに対する K/V オペレーションをサポートしています。

クエリサービス (n1ql)

クエリサービスは、SQL++ を介した JSON データのクエリをサポートします。

Amazon DocumentDB は、MongoDB API を介した JSON データのクエリをサポートしています。

インデックスサービス (インデックス)

インデックスサービスは、データのインデックスを作成および維持し、クエリを高速化します。

Amazon DocumentDB は、デフォルトのプライマリインデックスと、MongoDB API を介した JSON データへのセカンダリインデックスの作成をサポートしています。

検索サービス (fts)

検索サービスは、全文検索のインデックスの作成をサポートします。

Amazon DocumentDB のネイティブフルテキスト検索機能を使用すると、 MongoDB API を使用して特殊目的のテキストインデックスを使用して、大きなテキストデータセットに対して基本的なテキストクエリを実行できます。あいまい検索、多言語検索、クロスコレクション検索などの高度な機能については、Amazon OpenSearch Service との Amazon DocumentDB ゼロ ETL 統合 OpenSearchを使用します。

分析サービス (cbas)

分析サービスは、JSON データをほぼリアルタイムで分析することをサポートしています。

Amazon DocumentDB は、MongoDB API を介した JSON データに対するアドホッククエリをサポートしています。Amazon EMR でAmazon DocumentDB の JSON データに対して複雑なクエリを実行することもできます。

イベントサービス (イベント)

イベントサービスは、データの変更に応じてユーザー定義のビジネスロジックを実行します。

Amazon DocumentDB は、Amazon DocumentDB クラスターでデータが変更されるたびに AWS Lambda 関数を呼び出すことで、イベント駆動型のワークロードを自動化します。

バックアップサービス (バックアップ)

バックアップサービスは、完全および増分データバックアップと以前のデータバックアップのマージをスケジュールします。

Amazon DocumentDB は、1～35 日間の保持期間でデータを Amazon S3 に継続的にバックアップするため、バックアップ保持期間内の任意の時点にすばやく復元できます。Amazon DocumentDB は、この継続的なバックアッププロセスの一環として、データの自動スナップショットも取得します。Amazon DocumentDB のバックアップと復元は、 を使用して管理することもできます AWS Backup。

運用特性

Discovery Tool for Couchbase を使用して、データセット、インデックス、ワークロードに関する次の情報を取得します。この情報は、Amazon DocumentDB クラスターのサイズ設定に役立ちます。

データセット

このツールは、次のバケット、スコープ、およびコレクション情報を取得します。

1. バケット名
2. バケットタイプ
3. スコープ名
4. コレクション名
5. 合計サイズ (バイト)
6. 合計項目
7. 項目サイズ (バイト)

インデックス

このツールは、すべてのバケットについて、次のインデックス統計とすべてのインデックス定義を取得します。Amazon DocumentDB はコレクションごとにプライマリインデックスを自動的に作成するため、プライマリインデックスは除外されることに注意してください。

1. バケット名
2. スコープ名
3. コレクション名
4. インデックス名
5. インデックスサイズ (バイト)

ワークロード

このツールは、K/V および N1QL クエリメトリクスを取得します。K/V メトリクス値はバケットレベルで収集され、SQL++ メトリクスはクラスターレベルで収集されます。

ツールのコマンドラインオプションは次のとおりです。

```
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>
```

コマンドの例を次に示します。

```
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 メトリクス値は、過去 1 週間の 10 分ごとのサンプルに基づきます (HTTP メソッドと URI を参照)。SQL++ メトリクス値は、過去 60 秒間の 1 秒ごとのサンプルに基づきます (「一般的なラベル」を参照)。コマンドの出力は次のファイルにあります。

collection-stats.csv – バケット、スコープ、コレクション情報

```
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 – インデックス名とサイズ

```
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 – すべてのバケットのメトリクスを取得、設定、削除する

```
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++ クラスターのメトリクスを選択、削除、挿入する

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

indexes-<bucket-name>.txt – バケット内のすべてのインデックスのインデックス定義。Amazon DocumentDB はコレクションごとにプライマリインデックスを自動的に作成するため、プライマリインデックスは除外されることに注意してください。

```
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`)
...
```

計画

計画フェーズでは、Amazon DocumentDB クラスターの要件を決定し、Couchbase バケット、スコープ、コレクションを Amazon DocumentDB データベースとコレクションにマッピングします。

Amazon DocumentDB クラスターの要件

検出フェーズで収集されたデータを使用して、Amazon DocumentDB クラスターのサイズを設定します。Amazon DocumentDB クラスターのサイズ設定の詳細については、「インスタンスのサイズ設定」を参照してください。

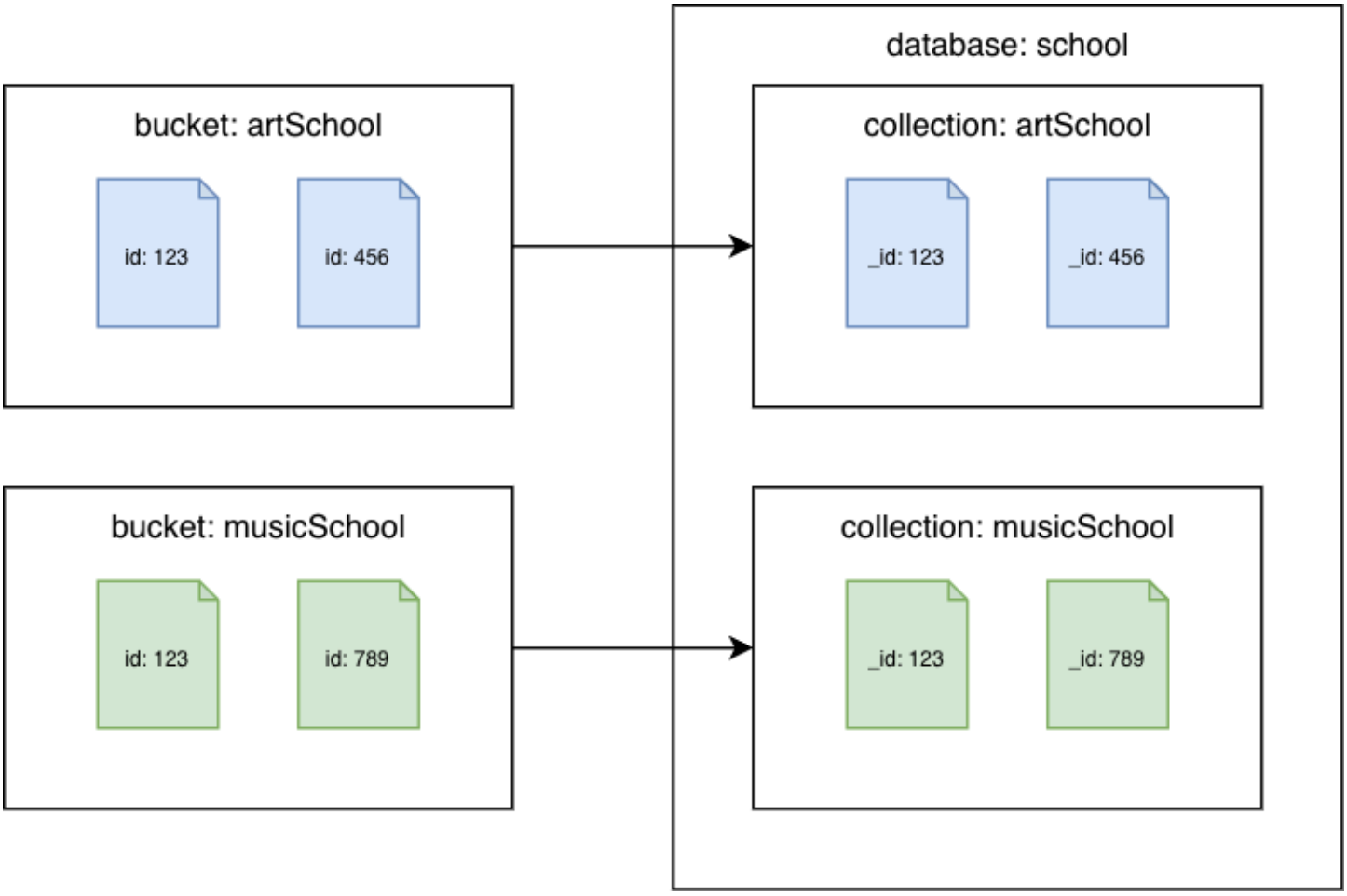
バケット、スコープ、コレクションをデータベースとコレクションにマッピングする

Amazon DocumentDB クラスター (複数可) に存在するデータベースとコレクションを決定します。Couchbase クラスター内のデータの整理方法に応じて、次のオプションを検討してください。これらは唯一のオプションではありませんが、考慮すべき出発点となります。

Couchbase Server 6.x 以前

Couchbase バケットから Amazon DocumentDB コレクションへ

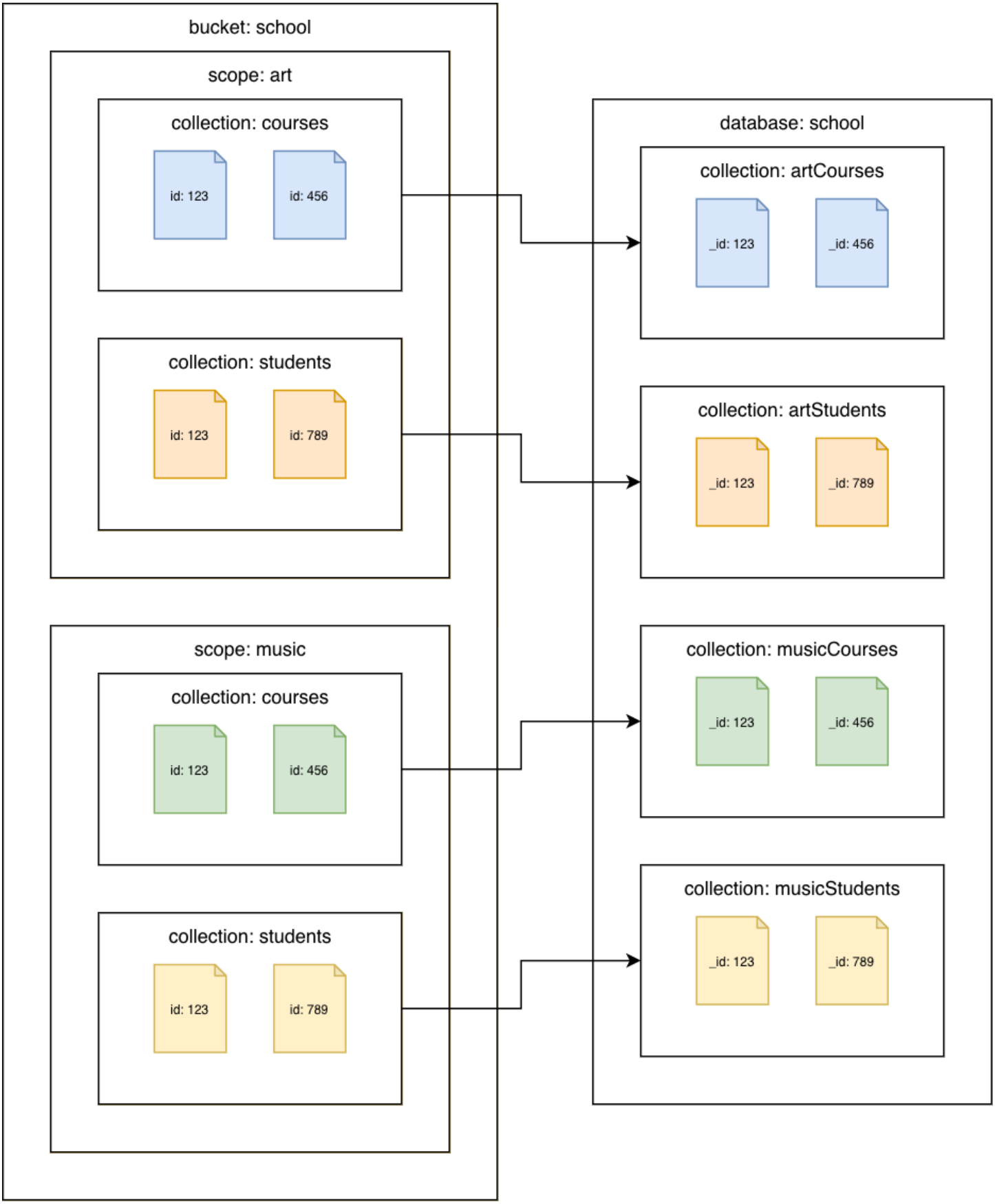
各バケットを別の Amazon DocumentDB コレクションに移行します。このシナリオでは、Couchbase ドキュメント`id`値が Amazon DocumentDB `_id`値として使用されます。



Couchbase Server 7.0 以降

Couchbase コレクションから Amazon DocumentDB コレクションへ

各コレクションを別の Amazon DocumentDB コレクションに移行します。このシナリオでは、Couchbase ドキュメント`id`値が Amazon DocumentDB `_id`値として使用されます。


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

移行

インデックスの移行

Amazon DocumentDB への移行には、データだけでなくインデックスも転送してクエリパフォーマンスを維持し、データベースオペレーションを最適化する必要があります。このセクションでは、互換性と効率を確保しながら Amazon DocumentDB にインデックスを移行するための詳細なstep-by-stepプロセスの概要を説明します。

Amazon Q (AWS マネジメントコンソールまたは IDE で使用可能) を使用して、SQL++ `CREATE INDEX`ステートメントを Amazon DocumentDB `createIndex()` コマンドに変換します。

1. Couchbase 用 Discovery Tool によって作成された indexes-<bucket name>.txt ファイル (複数可) をアップロードします。
2. 次のプロンプトを入力します。

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

Amazon Q は同等の Amazon DocumentDB `createIndex()` コマンドを生成します。Couchbase バケット、スコープ、コレクションを Amazon DocumentDB コレクションにマッピングする方法に基づいて、コレクション名を更新する必要がある場合があります。

例えば、次のようになります。

indexes-beer-sample.txt

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

Amazon Q 出力の例 (抜粋):

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

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

Amazon Q が変換できないインデックスの詳細については、Amazon DocumentDB インデックスとインデックスおよびインデックスプロパティの管理mongo-apis.html#mongo-apis-index」を参照してください。

MongoDB APIs を使用するためのリファクタリングコード

クライアントは Couchbase SDKs を使用して Couchbase Server に接続します。Amazon DocumentDB クライアントは、MongoDB ドライバーを使用して Amazon DocumentDB に接続します。Couchbase SDKs でサポートされているすべての言語は、MongoDB ドライバーでもサポートされています。言語のドライバーの詳細については、MongoDB ドライバー」を参照してください。

APIs は Couchbase Server と Amazon DocumentDB で異なるため、適切な MongoDB APIs を使用するようにコードをリファクタリングする必要があります。Amazon Q を使用して、K/V API コールと SQL++ クエリを同等の MongoDB APIsに変換できます。

1. ソースコードファイル (複数可) をアップロードします。
2. 次のプロンプトを入力します。

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

Hello Couchbase Python コードサンプルを使用して、Amazon Q は以下を生成します (抜粋) 。

```
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")
```

Python、Node.js、PHP、Go、Java、C#/ で Amazon DocumentDB に接続する例については、「プログラムによる Amazon DocumentDB への接続」を参照してください。NET、R、Ruby。

移行アプローチを選択する

Amazon DocumentDB にデータを移行する場合、次の 2 つのオプションがあります。

1. オフライン移行
2. オンライン移行

> **Note**
>
> Amazon DocumentDB は、各ドキュメントのプライマリキーとして _idフィールドを使用します。このガイドの移行アプローチでは、Couchbase ドキュメントキーを _idフィールドに保存します。オフライン移行の場合、これは の --include-key _idオプションを使用して行われますcbexport。オンライン移行の場合、これは Kafka コネクタの を介して行われますProvidedInKeyStrategy。

オフライン移行

次の場合は、オフライン移行を検討してください。

- ダウンタイムは許容されます。オフライン移行では、ソースデータベースへの書き込みオペレーションを停止し、データをエクスポートしてから Amazon DocumentDB にインポートします。このプロセスでは、アプリケーションのダウンタイムが発生します。アプリケーションまたはワークロードがこの使用不能期間を許容できる場合、オフライン移行は実行可能なオプションです。

- 小規模なデータセットの移行または概念実証の実施: 小規模なデータセットの場合、エクスポートとインポートのプロセスに必要な時間は比較的短く、オフライン移行は迅速かつ簡単な方法です。また、ダウンタイムの重要性が低い開発、テスト、proof-of-concept環境にも適しています。
- シンプルさが優先されます。cbexport と mongoimport を使用するオフラインメソッドは、一般的にデータを移行するための最も簡単なアプローチです。これにより、オンライン移行方法に伴う変更データキャプチャ (CDC) の複雑さを回避できます。
- 継続的な変更をレプリケートする必要はありません。移行中にソースデータベースが変更をアクティブに受信していない場合、または移行プロセス中にそれらの変更をキャプチャしてターゲットに適用することが重要でない場合は、オフラインアプローチが適切です。

トピック

- Couchbase Server 6.x 以前
- Couchbase Server 7.0 以降

Couchbase Server 6.x 以前

Couchbase バケットから Amazon DocumentDB コレクションへ

cbexport json を使用してデータをエクスポートし、バケット内のすべてのデータの JSON ダンプを作成します。`--format` オプションには、 `lines`または を使用できます`list`。

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

mongoimport と適切なオプションを使用して Amazon DocumentDB コレクションにデータをインポートし、行またはリストをインポートします。

行:

```
mongoimport \
  --db <database> \
  --collection <collection> \
  --uri "<Amazon DocumentDB cluster connection string>" \
```

```
  --ssl \
  --sslCAFile global-bundle.pem \
  --file export.json
```

リスト:

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

Couchbase Server 7.0 以降

オフライン移行を実行するには、cbexport ツールと mongoimport ツールを使用します。

デフォルトのスコープとデフォルトのコレクションを持つ Couchbase バケット

cbexport json を使用してデータをエクスポートし、バケット内のすべてのコレクションの JSON ダンプを作成します。`--format` オプションには、 `lines`または を使用できます`list`。

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

mongoimport と適切なオプションを使用して Amazon DocumentDB コレクションにデータをインポートし、行またはリストをインポートします。

行:

```
mongoimport \
  --db <database> \
  --collection <collection> \
  --uri "<Amazon DocumentDB cluster connection string>" \
```

```
  --ssl \
  --sslCAFile global-bundle.pem \
  --file export.json
```

リスト:

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

Couchbase コレクションから Amazon DocumentDB コレクションへ

cbexport json を使用してデータをエクスポートし、各コレクションの JSON ダンプを作成します。`--include-data` オプションを使用して、各コレクションをエクスポートします。`--format` オプションには、 `lines`または を使用できます`list`。`--scope-field` および `--collection-field`オプションを使用して、各 JSON ドキュメントの指定されたフィールドにスコープとコレクションの名前を保存します。

```
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"
```

cbexport はエクスポートされたすべてのドキュメントに `_scope`および `_collection`フィールドを追加したため、検索と置換、sed、または任意の方法でエクスポートファイル内のすべてのドキュメントから削除できます。

mongoimport と行またはリストをインポートする適切なオプションを使用して、各コレクションのデータを Amazon DocumentDB コレクションにインポートします。

行:

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

リスト:

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

オンライン移行

ダウンタイムを最小限に抑え、継続的な変更をほぼリアルタイムで Amazon DocumentDB にレプリケートする必要がある場合は、オンライン移行を検討してください。

Amazon Amazon DocumentDB へのライブ移行を実行する方法については、「Couchbase から Amazon Amazon DocumentDBへのライブ移行を実行する方法」を参照してください。このドキュメントでは、ソリューションをデプロイし、Amazon DocumentDB クラスターへのバケットのライブ移行を実行する方法について説明します。

トピック

- Couchbase Server 6.x 以前
- Couchbase Server 7.0 以降

Couchbase Server 6.x 以前

Couchbase バケットから Amazon DocumentDB コレクションへ

Couchbase の移行ユーティリティは、Couchbase バケットの Amazon DocumentDB コレクションへのオンライン移行を実行するように事前設定されています。シンクコネクタの設定を見ると、

document.id.strategyパラメータはメッセージキー値を_idフィールド値として使用するように設定されています (「シンクコネクタ ID Strategy Properties」を参照)。

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

Couchbase Server 7.0 以降

デフォルトのスコープとデフォルトのコレクションを持つ Couchbase バケット

Couchbase の移行ユーティリティは、Couchbase バケットの Amazon DocumentDB コレクションへのオンライン移行を実行するように事前設定されています。シンクコネクタの設定を見ると、document.id.strategyパラメータはメッセージキー値を_idフィールド値として使用するように設定されています (「シンクコネクタ ID Strategy Properties」を参照)。

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

Couchbase コレクションから Amazon DocumentDB コレクションへ

各スコープ内の各 Couchbase コレクションを個別のトピックにストリーミングするようにソースコネクタを設定します (「ソース設定オプション」を参照)。例えば、次のようになります。

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

各トピックから個別の Amazon DocumentDB コレクションにストリーミングするようにシンクコネクタを設定します (「シンクコネクタ設定プロパティ」を参照)。例えば、次のようになります。

```
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>'
```

検証

このセクションでは、Amazon DocumentDB に移行した後にデータの整合性と整合性を検証するための詳細な検証プロセスについて説明します。検証ステップは、移行方法に関係なく適用されます。

トピック

- すべてのコレクションがターゲットに存在することを確認する
- ソースクラスターとターゲットクラスター間のドキュメント数を検証する
- ソースクラスターとターゲットクラスター間でドキュメントを比較する

すべてのコレクションがターゲットに存在することを確認する

Couchbase ソース

オプション 1: クエリワークベンチ

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

オプション 2: cbq ツール

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

Amazon DocumentDB ターゲット

mongosh (Amazon DocumentDB クラスターに接続する」を参照):

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

ソースクラスターとターゲットクラスター間のドキュメント数を検証する

Couchbase ソース

Couchbase Server 6.x 以前

オプション 1: クエリワークベンチ

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

オプション 2: cbq

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

Couchbase Server 7.0 以降

オプション 1: クエリワークベンチ

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

オプション 2: cbq

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

```
        FROM `<bucket>`.`<scope>`.`<collection>`"
```

Amazon DocumentDB ターゲット

mongosh (Amazon DocumentDB クラスターに接続する」を参照):

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

ソースクラスターとターゲットクラスター間でドキュメントを比較する

Couchbase ソース

Couchbase Server 6.x 以前

オプション 1: クエリワークベンチ

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

オプション 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 以降

オプション 1: クエリワークベンチ

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

オプション 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 ターゲット

mongosh (Amazon DocumentDB クラスターに接続する」を参照):

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

Amazon DocumentDB エンジンバージョンのサポート日

次の日付を参考にすると、テストおよびアップグレードのサイクルを計画することができます。

Amazon DocumentDB メジャーバージョンのリリースカレンダー

Amazon DocumentDB は現在、次のメジャーバージョンをサポートしています。エンジンバージョンをアップグレードするには、メジャーエンジンバージョンのアップグレード「」または「」を参照してくださいを使用したアップグレード AWS DMS。

エンジンバージョン	リリース日	標準サポート終了日	延長サポート開始 (1 年目の価格設定)	延長サポート開始 (3 年目の価格設定)	延長サポートの終了日
バージョン 3.6	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
バージョン 4.0	2020 年 11 月 9 日	該当なし	該当なし	該当なし	該当なし

エンジンバージョン	リリース日	標準サポート終了日	延長サポート開始 (1 年目の価格設定)	延長サポート開始 (3 年目の価格設定)	延長サポートの終了日
バージョン 5.0	2023 年 3 月 1 日	該当なし	該当なし	該当なし	該当なし
バージョン 8.0	2025 年 11 月 14 日	該当なし	該当なし	該当なし	該当なし

Amazon DocumentDB は、標準サポートで複数のエンジンバージョンをサポートしています。標準サポート終了日を過ぎてもバージョンは有料 (延長サポートの手数料) で引き続き使用できます。詳細については、「Amazon DocumentDB 延長サポート」および「Amazon DocumentDB の料金」を参照してください。

Amazon DocumentDB マイナーバージョンのリリースカレンダー

Amazon DocumentDB は現在、次のマイナーバージョンをサポートしています。リリーススケジュールは、追加の機能や修正によって異なります。マイナーバージョンの標準サポートは、対応するメジャーバージョンの標準サポートより前に終了します。エンジンのバージョンをアップグレードするには、「」を参照してくださいマイナーエンジンバージョンのアップグレード。

> **Note**
>
> マイナーバージョンは、メジャーバージョン 5.0 以降で使用できます。

エンジンバージョン	リリース日
バージョン 5.0.0 (LTS)	2026 年 2 月 16 日
バージョン 5.0.1	2026 年 6 月 8 日

Amazon DocumentDB は、特定のバージョンを長期サポート (LTS) リリースとして指定します。詳細については、「長期サポート (LTS) リリースの使用」を参照してください。

Amazon DocumentDB のインプレースメジャーバージョンアップグレード

同じエンドポイント、ストレージ、タグを維持しながら、Amazon DocumentDB クラスターのインプレースメジャーバージョンアップグレード (MVU) を実行できます。アプリケーションは変更なしで動作し続けます。この機能は、Amazon DocumentDB が利用可能なすべての AWS リージョン で追加料金なしで使用できます。

> **Important**
>
> クラスターはアップグレード中に使用できず、複数回再起動します。アップグレードの開始後は、クラスターに接続したり、クラスターから読み取ったり、クラスターに書き込んだりしないでください。ダウンタイムは、コレクション、インデックス、データベース、インスタンスの数によって異なります。メンテナンスウィンドウまたは使用率の低い時間帯にアップグレードを実行します。
>
> アップグレード後、以前のバージョンにダウングレードすることはできません。必要に応じて、アップグレード前のスナップショットを新しいクラスターに復元できます。

トピック

- サポートされているアップグレードパス
- 前提条件
- ベストプラクティス
- Amazon DocumentDB 8.0 にアップグレードした後の変更点
- 3.6 または 4.0 からアップグレードされたクラスターのアップグレード後の考慮事項
- アップグレードの実行
- トラブルシューティング

サポートされているアップグレードパス

次の表に、ソースバージョンごとに選択できるターゲットエンジンバージョンを示します。最初に `.0` マイナーバージョンにアップグレードしてから、マイナーバージョンアップグレードを適用する必要はありません。

ソースバージョン	サポートされているターゲットバージョン
3.6	公開されている 5.0 または 8.0 マイナーバージョン
4.0	公開されている 5.0 または 8.0 マイナーバージョン
5.0.0	8.0.0、8.0.1、8.0.2
5.0.1	8.0.0、8.0.1、8.0.2
5.0.2	8.0.2

アップグレード後の考慮事項については、the section called “3.6 または 4.0 からアップグレードされたクラスターのアップグレード後の考慮事項”「」および「」を参照してくださいthe section called “Amazon DocumentDB 8.0 にアップグレードした後の変更点”。

AWS リージョンで使用可能なマイナーバージョンを表示するには、 AWS CLI コマンドを使用しますaws docdb describe-db-engine-versions。リリースされたマイナーバージョンのリストについては、「」を参照してくださいリリースノート。

> **Note**
>
> インプレース MVU は、グローバルクラスターまたは Elastic クラスターではサポートされていません。グローバルクラスターをアップグレードするには、セカンダリクラスターを削除し、プライマリクラスターをリージョンクラスターに変換して MVU を実行し、同じ名前を使用してセカンダリクラスターを追加してグローバルクラスターを再作成してエンドポイントを保持します。アップグレードされたプライマリが新しいセカンダリにデータをレプリケートしている間、I/O 料金が発生します。詳細なステップについては、「Amazon DocumentDB グローバルクラスターからのクラスターの削除」を参照してください。

前提条件

> **Important**
>
> アップグレードする前にバースト可能なインスタンスをスケールアップします。クラスターがバースト可能なインスタンスタイプ (db.t3.mediumや などdb.t4g.medium) を使用している場合は、アップグレードdb.r6g.largeを開始する前に、プライマリインスタンスを

少なくとも db.r5.large または にスケールアップします。バーストインスタンスには、アップグレードプロセスを完了するのに十分な CPU とメモリがない可能性があります。これにより、アップグレードが失敗し、クラスターが長時間使用できなくなる可能性があります。アップグレードが完了したら、スケールダウンできます。
アップグレードする前にサーバーレスライターインスタンスをスケールアップします。クラスターにサーバーレスライターインスタンスがある場合、アップグレードを開始する前に、の最大 DCU を 3 以上に設定ServerlessV2ScalingConfigurationする必要があります。最大 DCU が 3 未満のクラスターは、アップグレードを完了するのに十分な CPU またはメモリがないため、アップグレードできません。
アップグレードする前に部分インデックスを確認してください (Amazon DocumentDB 5.0 から 8.0 にアップグレード)。アップグレードを開始する前に、 $type演算子partialFilterExpressionを使用して特定の BSON タイプに一致する の部分インデックスを確認します。具体的には、binData、、booldatedouble、decimal、、および arrayタイプのエイリアス、または などの数値型コードです{ "$type" : 1 }。このような部分インデックスは、インデックスが読み取りと書き込みで正常に動作する場合でも、アップグレードが失敗します。db.collection.getIndexes() 部分インデックスpartialFilterExpressionの を確認するには、 を使用します。アップグレードを開始する前に影響を受ける部分インデックスを削除し、アップグレードの完了後に再作成します。インデックスが影響を受けるかどうか不明な場合は、アップグレードする前に AWS サポートにお問い合わせください。

- インスタンスタイプ — Amazon DocumentDB 4.0+ は db.r4 インスタンスをサポートしていません。アップグレードする前にdb.r4.*、インスタンスをdb.r5.*インスタンス以降に変更します。「Amazon DocumentDB インスタンスの変更」および「リージョン別のサポートされるインスタンスクラス」を参照してください。
- OS パッチ — アップグレードする前に、すべてのインスタンスに保留中の OS メンテナンスアクションを適用します。「Amazon DocumentDB オペレーティングシステムの更新」を参照してください。

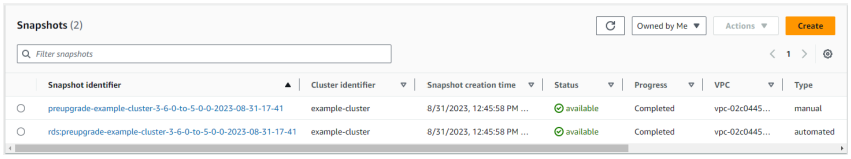
> **Note**
>
> クラスターレベルのエンジンパッチが保留中の場合、インスタンス OS パッチが非表示になることがあります。必要に応じて、最初にエンジンパッチを適用します。「クラスターのエンジンバージョンへのパッチ更新の実行」を参照してください。

- バーストインスタンス (T ファミリーインスタンス) のインデックス制限 — バーストインスタンスに 3,000 個を超えるインデックスがある場合は、アップグレードする前にプライマリを少なくとも db.r5.large にスケールアップします。アップグレードが完了したら、スケールダウンできます。

インスタンス	MVU の最大インデックス
db.t4g.medium	3K
db.t3.medium	10K

- パラメータグループ — アップグレードする前に、ターゲットバージョンのカスタムクラスターパラメータグループを用意します。指定しない場合、ターゲットバージョンのデフォルトのパラメータグループが使用されます (例: `default.docdb5.0`または `default.docdb8.0`)。
- 手動スナップショット — アップグレードする前に手動スナップショットを作成します。アップグレードプロセスでは、 という名前の自動スナップショットが作成されますが`preupgrade-<name>-<version>-<timestamp>`、常に独自のバックアップを作成します。「手動クラスタースナップショットの作成」を参照してください。

> **Note**
>
> アップグレードプロセスで自動作成されたスナップショットは、インプレースメジャーバージョンアップグレードが完了しても自動的には削除されません。このスナップショットは、保持期間内である限り料金は発生しません。クラスターのアップグレードが成功したことを確認したら、このスナップショットを削除できます。



ベストプラクティス

アップグレード前 — クローンを使用したテスト

1. Amazon DocumentDB クラスターのボリュームのクローン作成 を使用してクラスターのクローンを作成します。クローンのデータを変更しない限り、ストレージコストは発生しません。

2. クローンのインスタンス数をターゲットクラスターに一致させて、現実的な推定時間を確保します。
3. クローンで MVU を実行し、機能の違いを完全にテストします。
4. を実行して を探してdescribe-db-clusters、アップグレードが既にスケジュールされているかどうかを確認しますPendingModifiedValues.EngineVersion。クラスターを変更し、次のメンテナンスウィンドウで適用するように選択した場合、スケジュールはコンソールに表示されませんが、 で表示できます AWS CLI。

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

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

5. テストが成功したら、本番クラスターのアップグレードに進みます。

アップグレード中

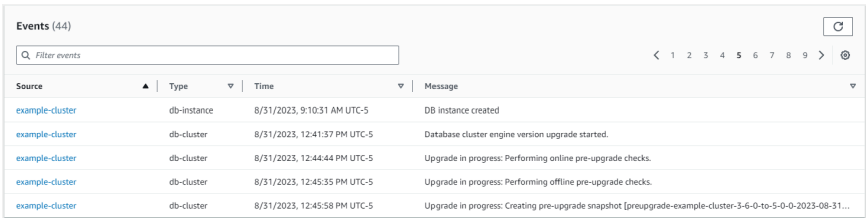
クラスターメンテナンスイベントにサブスクリプションすることで、インプレースメジャーバージョンアップグレードの進捗状況をモニタリングできます。アップグレードが完了すると、「データベースクラスターのメジャーバージョンがアップグレードされました」イベントが表示されます。アップグレード中に発生するこのイベントやその他のイベントは、Amazon DocumentDB コンソールのクラスター詳細ページのイベントとタグセクションに表示されます。その後、クラスターのステータスは から upgradingに変わりますavailable。

アップグレード中に次のイベントが生成されます。

1. データベースクラスターエンジンのメジャーバージョンアップグレードが開始されました。クラスターはオンラインのままです。
2. アップグレードの準備中: オンラインアップグレードの事前チェックを開始しました。
3. アップグレードの準備中: オンラインアップグレードの事前チェックを完了しました。
4. プライマリインスタンスがパッチ/アップグレードプロセスを完了する間、データベースクラスターをオフラインにします。
5. アップグレードの準備中: オフラインアップグレードの事前チェックを開始します。
6. アップグレードの準備中: オフラインアップグレードの事前チェックが完了しました。

7. アップグレード中: アップグレード前のスナップショットの作成 [preupgrade-<cluster-name>-<version-from>-to-<version-to>-<timestamp>]。
8. アップグレード中: ボリュームのクローンを作成しています。
9. アップグレード中: ライターをアップグレードしています。
10. アップグレード中: リーダーをアップグレードしています。
11. データベースクラスターエンジンのメジャーバージョンがアップグレードされました。

イベントは、イベントページのコンソールに表示されます。



から AWS CLI、 `aws docdb describe-events`を実行してアップグレードの進行状況をモニタリングできます。通知を自動的に受信するには、 `aws docdb create-event-subscription` を使用してイベントをサブスクライブし、E メール、プッシュメッセージ、またはその他の配信方法で Amazon SNS にルーティングします。詳細については、「Amazon DocumentDB イベントにサブスクライブする」を参照してください。

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

このコマンドにより、以下のような出力が返されます。

```
{
    "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"
        }
```

```
    ]
}
```

アップグレード後

> **Warning**
>
> インプレースメジャーバージョンアップグレードの直後に、Amazon DocumentDB クラスターは、データベースエンジンがクエリ実行プランを最適化するために使用するインデックスメタデータを再入力します。このプロセスが完了すると、クエリのパフォーマンスは期待されるレベルに戻ります。通常は数分で終了しますが、クラスターのインデックス数によっては最大 2 時間かかる場合があります。
> インデックスメタデータの再計算が中断される可能性があるため、この間はライターインスタンスを再起動、フェイルオーバー、またはスケールアップ/ダウンしないでください。このような変更を行う前に、予想されるクエリパフォーマンスを確認するまで待ちます。
> 次のクラスターイベントを使用して進行状況を追跡します。
>
> - アップグレード後のクラスターステータス: インデックスメタデータの更新プロセスが開始されました
> - アップグレード後のクラスターステータス: X 秒で完了したインデックスメタデータの更新プロセス
>
> インデックスメタデータの更新プロセスが 3 時間以内に完了していない場合、またはプロセスの完了後もパフォーマンスの問題が続く場合は、 AWS サポートにお問い合わせください。

1. アップグレード後の状態に復元する必要がある場合に備えて、アップグレードされたクラスターの手動スナップショットを作成します。自動スナップショットプロセスは、インプレースメジャーバージョンアップグレードが完了するとすぐに再開されます。
2. 3.6 からアップグレードされたクラスターにタグを付けます。タグを追加して、3.6 からアップグレードされたクラスターを区別します (「」を参照the section called “3.6 または 4.0 からアップグレードされたクラスターのアップグレード後の考慮事項”)。
3. ドライバーを更新します。新機能 (8.0 での照合、ビュー、Zstd 圧縮など) を使用するには、対応する MongoDB API バージョンにアップグレードします。詳細については、「Amazon DocumentDB 8.0 の新機能」を参照してください。

4. 徹底的にテストします。アップグレードされたクラスターに対してアプリケーションを検証します。

Amazon DocumentDB 8.0 にアップグレードした後の変更点

Amazon DocumentDB 8.0 へのメジャーバージョンアップグレード (Amazon DocumentDB 5.0 から、または 3.6 または 4.0 から直接) を実行すると、次の機能が有効化または変更されます。

- 照合。Amazon DocumentDB 8.0 は照合をサポートしています。アップグレード後、新しいコレクションとそのインデックス、および既存のコレクションの新しいインデックスは、デフォルトで照合が有効になります。
- テキストインデックス。新しいテキストインデックスは、更新されたテキスト検索パーサーを使用して MongoDB の互換性を向上させるテキストインデックス V2 を使用して作成されます。既存のテキストインデックスは影響を受けません。
- クエリプランナーのバージョン。カスタムパラメータグループがない場合、Amazon DocumentDB 8.0 の新しいデフォルトパラメータグループが作成され、プランナーバージョン 3 が自動的に選択されます。クエリプランナーバージョン 3 では、ビューも利用できます。
- 圧縮。Amazon DocumentDB 8.0 は、Zstd アルゴリズムを使用したディクショナリベースのドキュメント圧縮をサポートしています。アップグレード後、Zstd 圧縮がデフォルトで有効になっている新しいコレクションが作成されます。5.0 の既存のコレクションは、圧縮設定を保持します。既存のコレクションで Zstd 圧縮を利用するには、圧縮設定を変更できます。詳細については、「Amazon DocumentDB 8.0 でのディクショナリベースの圧縮の管理」を参照してください。
- インデックスの再構築。Amazon DocumentDB 5.0 から Amazon DocumentDB 8.0 にアップグレードする場合、インデックスの再構築は必要ありません。Amazon DocumentDB 3.6 または 4.0 から 8.0 に直接アップグレードする場合は、「」の説明に従ってインデックスを再構築しますthe section called “3.6 または 4.0 からアップグレードされたクラスターのアップグレード後の考慮事項”。

> **Important**
>
> Amazon DocumentDB 8.0 には TLS 1.2 以降が必要です。TLS 1.0 および TLS 1.1 はサポートされなくなりました。

> **Note**
>
> 機能の違いの完全なリストについては、「」を参照してくださいAmazon DocumentDB の MongoDB との互換性。

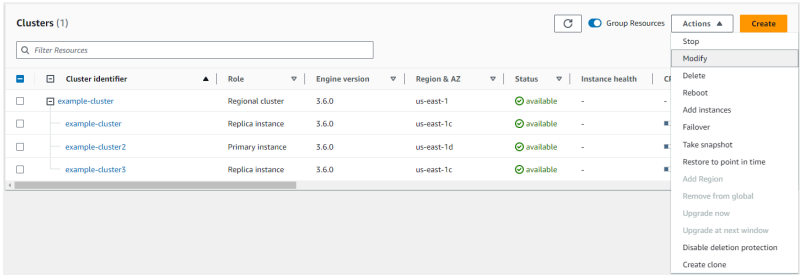
3.6 または 4.0 からアップグレードされたクラスターのアップグレード後の考慮事項

- インデックスの再構築。MVU は元のインデックスを保持します。Amazon DocumentDB 5.0 以降では、特に低カーディナリティインデックスのインデックスメンテナンスとガベージコレクションが改善されました。3.6 または 4.0 (5.0 または 8.0) からアップグレードした後、最適なクエリパフォーマンスを確保するためにインデックスを再構築します (オプション、追加の I/O を含む)。「reIndex を使用したインデックスのメンテナンス」を参照してください。
- サブドキュメントの数値比較 (3.6 のみ) 。3.6 からアップグレードされたクラスターは、サブドキュメントの数値型が型間で比較されない 3.6 の動作を継承します。たとえば、 `{a: {b: NumberLong(1)}}`は 3.6 `{a: {b: 1}}`では等しくありませんが、4.0 以降では等しくなります。この動作は、3.6 からアップグレードされたクラスターに影響します。

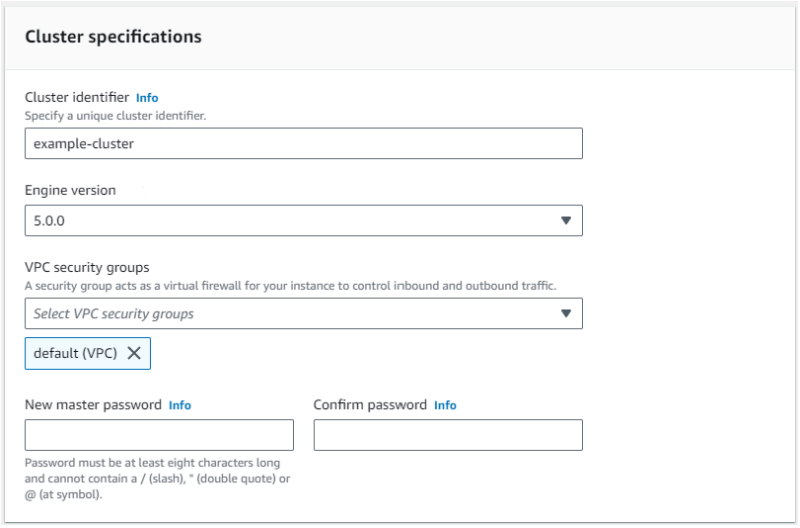
アップグレードの実行

Using the AWS マネジメントコンソール

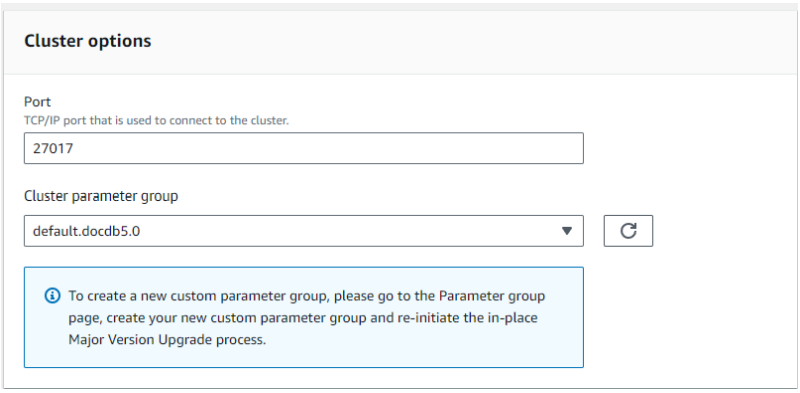
1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソールを開きます。
2. クラスターテーブルで、ソースクラスターを選択し、アクションを選択し、変更します。



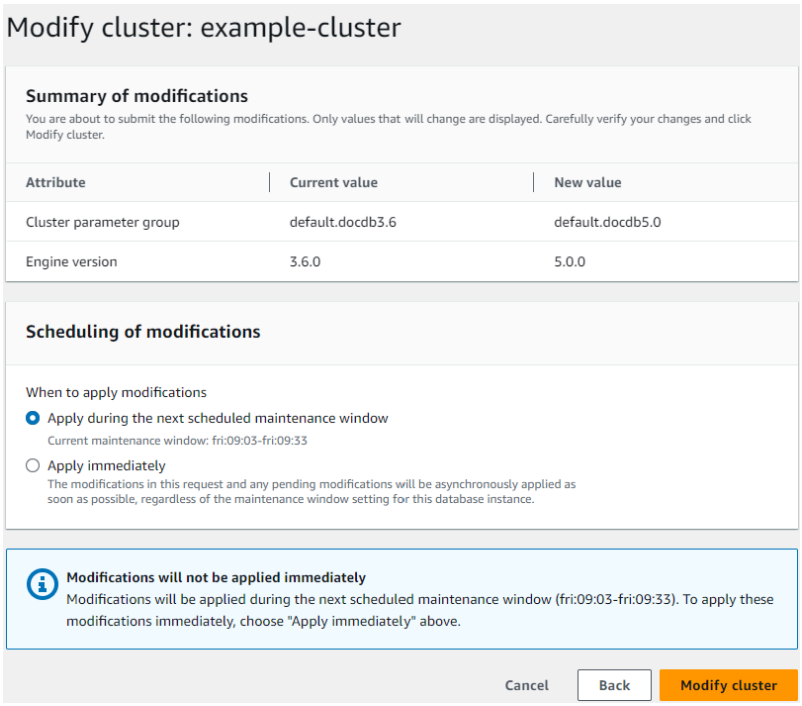
3. クラスターの仕様で、エンジンバージョンドロップダウンからターゲットバージョン (5.0.0 や 8.0.0 など) を選択します。



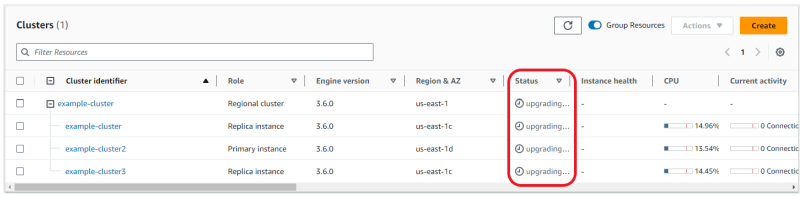
4. クラスターオプションで、ターゲットエンジンバージョンのクラスターパラメータグループを選択します。デフォルト (default.docdb5.0 や default.docdb8.0 など) または作成したカスタムパラメータグループを使用できます。



5. 続行を選択し、スケジュール設定 (即時または次のメンテナンスウィンドウを適用) を選択し、クラスターの変更を選択します。



6. アップグレードに変わると、クラスターテーブルのクラスターステータスをモニタリングします。



Using the AWS CLI

`--allow-major-version-upgrade` フラグ[modify-db-cluster](#)で を使用します。

```
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
```

各########をクラスターの情報に置き換えます。

トラブルシューティング

- アップグレード前のチェックに失敗しました。アップグレードを開始する前に、Amazon DocumentDB はアップグレード前の検証チェックを実行します。事前チェックの失敗の一般的な原因は次のとおりです。
  - collection (s) に 58 文字以上の名前があるため、アップグレードを続行できませんでした。アップグレードを再試行する前に、影響を受けるコレクションの名前を短い名前に変更します。
  - インデックス数がインスタンスタイプの制限を超えているため、アップグレードを続行できませんでした — アップグレードを再試行する前に、より大きなインスタンスタイプにアップグレードします。インスタンスタイプ別のインデックス制限については、「」を参照してください前提条件。
- アップグレードの失敗とロールバック。アップグレードが失敗すると、自動的にロールバックが試行されます。ロールバックが成功すると、「データベースクラスターはアップグレードできない状態です」というイベントが生成されます。クラスターはアップグレード前の状態に戻され、引き続き使用できます。再試行する前に AWS 、 サポートに連絡してトラブルシューティングを行ってください。
- アップグレード後のパフォーマンス。インデックスメタデータの更新の実行中に、パフォーマンスが一時的に低下し、CPU 使用率が高くなる可能性があります。劣化が 3 時間を超えて持続する場合は、 AWS サポートにお問い合わせください。

さらにサポートが必要な場合は、AWS サポート にお問い合わせください。

Amazon DocumentDB マイナーバージョンアップグレード

> **Note**
>
> このトピックでは、同じメジャーバージョン内のマイナーバージョンアップグレード (5.0.0 から 5.0.1 へのアップグレードなど) について説明します。別のメジャーバージョンへのアップグレードについては、「」を参照してくださいAmazon DocumentDB のインプレースメジャーバージョンアップグレード。

クラスターを変更することで、Amazon DocumentDB クラスターのマイナーエンジンバージョン (5.0.0 から 5.0.1 など) をアップグレードできます。マイナーバージョンアップグレードは、すぐに適用することも、次のメンテナンス期間中に適用することもできます。

マイナーバージョンアップグレードを実行する前に

マイナーバージョンアップグレードを実行する前に、次のアクションが役立つ場合があります。

- 短いダウンタイムの影響を軽減するために、トラフィックが少ない期間中にアップグレードを実行します。
- `describe-db-engine-versions` コマンドを使用して、ターゲットのマイナーバージョンが現在のバージョンに対して有効なアップグレードターゲットであることを確認します。
- グローバルクラスターの場合、セカンダリクラスターはプライマリクラスターの前にアップグレードする必要があります。詳細については、「グローバルクラスターのパッチ適用」を参照してください。

> **Important**
>
> - アップグレードは一方向オペレーションであり、アップグレード後にダウングレードすることはできません。
> - クラスターでは、アップグレードプロセス中に短いダウンタイムが発生します。

マイナーバージョンアップグレードの実行

Using the AWS マネジメントコンソール

を使用してマイナーバージョンアップグレードを実行するには AWS マネジメントコンソール:

1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソールを開きます。
2. クラスターテーブルで、アップグレードするクラスターを選択し、アクションを選択し、変更を選択します。
3. クラスターの変更ダイアログのクラスター仕様セクションで、エンジンバージョンドロップダウンメニューからターゲットのマイナーバージョンを選択します。
4. 下にスクロールして続行を選択します。
5. 変更のスケジュールセクションで、変更をすぐに適用するか、次のメンテナンスウィンドウ中に適用するかを選択します。

   その後、[Modify Cluster] を選択します。

Using the AWS CLI

`--engine-version` パラメータがある `modify-db-cluster` コマンドを使用します。

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

上記の例では、各 `##############` をクラスターの情報に置き換えます。

> **Note**
>
> アップグレードをすぐに適用するのではなく、次のメンテナンスウィンドウ中に適用するには、 `--no-apply-immediately` の代わりに を使用します`--apply-immediately`。

マイナーバージョンアップグレードの検証

アップグレードが完了したら、以下を使用してエンジンのバージョンを確認します AWS CLI。

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

このオペレーションによる出力は、次のようになります。

```
"5.0.1"
```

を使用した Amazon DocumentDB クラスターのアップグレード AWS Database Migration Service

> **Important**
>
> Amazon DocumentDB は MongoDB と同じサポートライフサイクルに従わず、MongoDB のライフサイクルの終了スケジュールは Amazon DocumentDB には適用されません。

Amazon DocumentDB クラスターを、最小限のダウンタイムで上位バージョンにアップグレードできます AWS DMS。 AWS DMS は、古い Amazon DocumentDB バージョン、リレーショナルデータベース、非リレーショナルデータベースからターゲット Amazon DocumentDB クラスターに簡単に移行できるフルマネージドサービスです。

トピック

- ステップ 1: 変更ストリームを有効にする
- ステップ 2: 変更ストリームログの保持期間を変更する
- ステップ 3: インデックスを移行する
- ステップ 4: AWS DMS レプリケーションインスタンスを作成する
- ステップ 5: AWS DMS ソースエンドポイントを作成する
- ステップ 6: AWS DMS ターゲットエンドポイントを作成する
- ステップ 7: 移行タスクの作成と実行
- ステップ 8: アプリケーションエンドポイントから Amazon DocumentDB クラスター への変更

ステップ 1: 変更ストリームを有効にする

最小限のダウンタイム移行を実行するには、クラスターの変更ストリームへのアクセス AWS DMS が必要です。Amazon DocumentDB 変更ストリーム は、クラスターのコレクションとデータベース内で発生する変更イベントの時系列シーケンスを提供します。変更ストリームから読み取ると AWS DMS 、 は変更データキャプチャ (CDC) を実行し、ターゲット Amazon DocumentDB クラスターに増分更新を適用できます。

特定のデータベースのすべてのコレクションの変更ストリームを有効にするには、[mongo] シェルを使用して Amazon DocumentDB クラスターを認証し、次のコマンドを実行します。

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

ステップ 2: 変更ストリームログの保持期間を変更する

次に、変更ストリームに変更イベントを保持したい期間に基づいて、変更ストリームの保持期間を変更します。例えば、 を使用した Amazon DocumentDB クラスターの移行 AWS DMS に 12 時間かかることが予想される場合は、変更ストリームの保持期間を 12 時間を超える値に設定する必要があり

ます。Amazon DocumentDB クラスターのデフォルトの保持期間は 3 時間です。 AWS マネジメントコンソール または を使用して、Amazon DocumentDB クラスターの変更ストリームログの保持期間を 1 時間から 7 日の間で変更できます AWS CLI。詳細については、「変更ストリームログの保持期間の変更」を参照してください。

ステップ 3: インデックスを移行する

Amazon DocumentDB クラスターと同じインデックスを、Amazon DocumentDB クラスター上に作成します。はデータの移行 AWS DMS を処理しますが、インデックスは移行されません。インデックスを移行するには、Amazon DocumentDB インデックスツールを使用して、Amazon DocumentDB クラスターからインデックスをエクスポートします。このツールを入手するには、Amazon DocumentDB ツール GitHub リポジトリのクローンを作成し、README.md 中の手順に従います。このツールは、Amazon DocumentDB クラスターと同じ Amazon VPC で実行されている Amazon EC2 インスタンスまたは AWS Cloud9 環境から実行できます。 Amazon DocumentDB

次の例では、各 ############## を独自の情報に置き換えます。

次のコードは、Amazon DocumentDB クラスターからインデックスをダンプします。

```
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/
```

インデックスが正常にエクスポートされたら、ターゲットの Amazon DocumentDB クラスター中にそれらのインデックスを復元します。前のステップでエクスポートしたインデックスを復元するには、Amazon DocumentDB インデックスツールを使用します。次のコマンドは、指定したディレクトリから Amazon DocumentDB クラスター中のインデックスを復元します。

```
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
```

インデックスを正しく復元したことを確認するには、Mongo シェルを使用して Amazon DocumentDB クラスターに接続し、特定のコレクションのインデックスをリストします。以下のコードを参照してください。

```
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()
```

ステップ 4: AWS DMS レプリケーションインスタンスを作成する

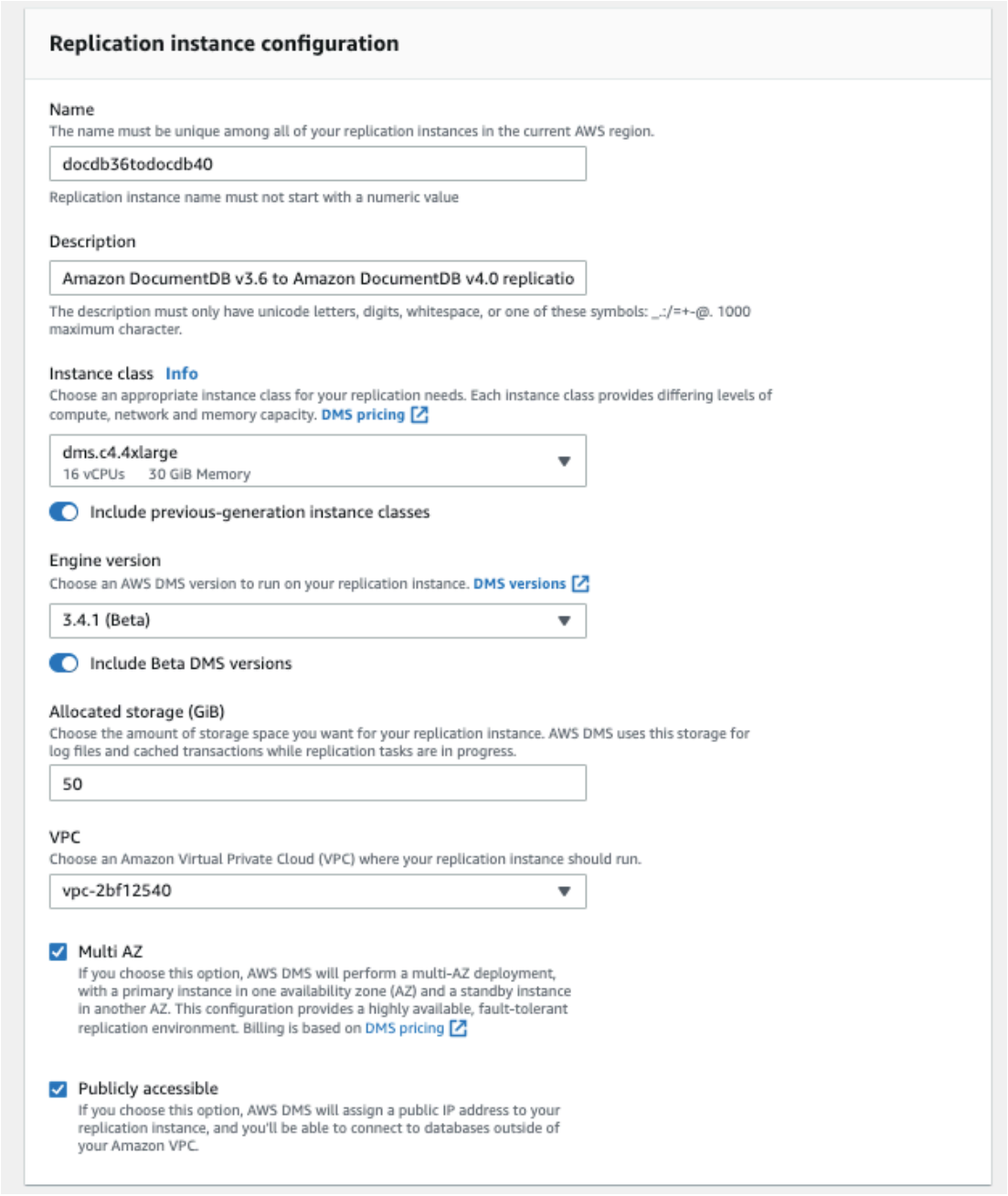
AWS DMS レプリケーションインスタンスは、ソース Amazon DocumentDB クラスターに接続してデータを読み取り、ターゲット Amazon DocumentDB クラスターに書き込みます。 AWS DMS レプリケーションインスタンスは、一括ロードと CDC オペレーションの両方を実行できます。この処理のほとんどはメモリ内で行われます。ただし、大きいトランザクションではディスク上でのバッファリングが必要になることがあります。キャッシュされたトランザクションとログファイルもディスクに書き込まれます。データが移行されると、レプリケーションインスタンスは変更イベントをストリーミングして、ソースとターゲットが同期していることを確認します。

AWS DMS レプリケーションインスタンスを作成するには:

1. AWS DMS コンソールを開きます。
2. ナビゲーションペインで [Replication instances] (レプリケーション インスタンス) を選択します。
3. [Create replication instance (レプリケーションインスタンスの作成)] を選択し、次の情報を入力します。

   - 名前については、希望する名前を入力します。例えば、`docdb36todocdb40`。
   - 説明 については、選んだ説明を入力します。リストアイテムについては、Amazon DocumentDB 3.6 を Amazon DocumentDB 4.0 レプリケーションインスタンスに。
   - インスタンスクラス については、必要に応じてサイズを選択します。

- エンジンバージョン については、`3.4.1.` を選択します。
- Amazon VPC については、ソースとターゲットの Amazon DocumentDB クラスターを収容する Amazon VPC を選択します。
- ストレージ割り当て (GiB) については、デフォルトの 50 GiB を使用します。書き込みスループットのワークロードが高い場合は、ワークロードに合わせてこの値を増やします。
- マルチ AZ については、高可用性とフェイルオーバーサポートが必要な場合は はい を選択します。
- [Publicly accessible (パブリックアクセス可能)] を有効にします。



4. [Create replication instance] (レプリケーションインスタンスの作成) を選択します。

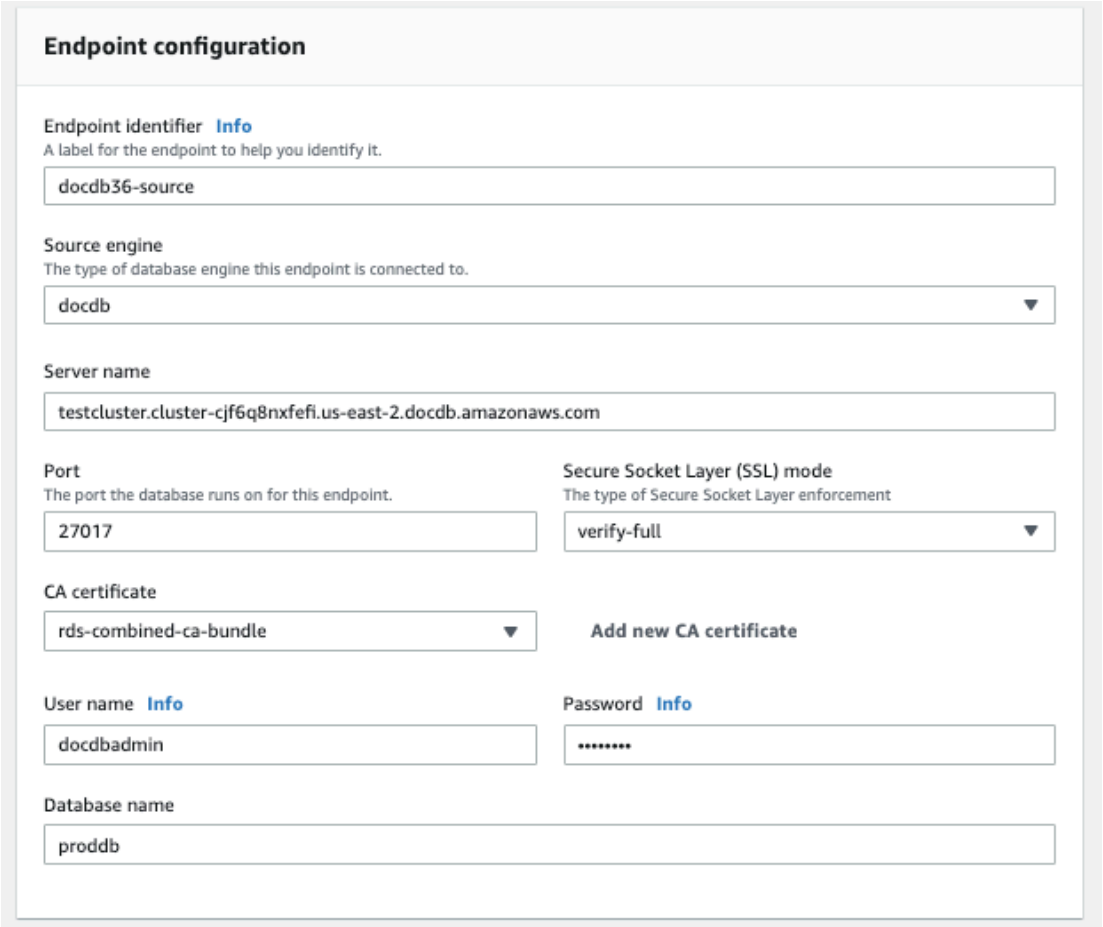
ステップ 5: AWS DMS ソースエンドポイントを作成する

ソースエンドポイントは、ソース Amazon DocumentDB クラスターに使用されます。

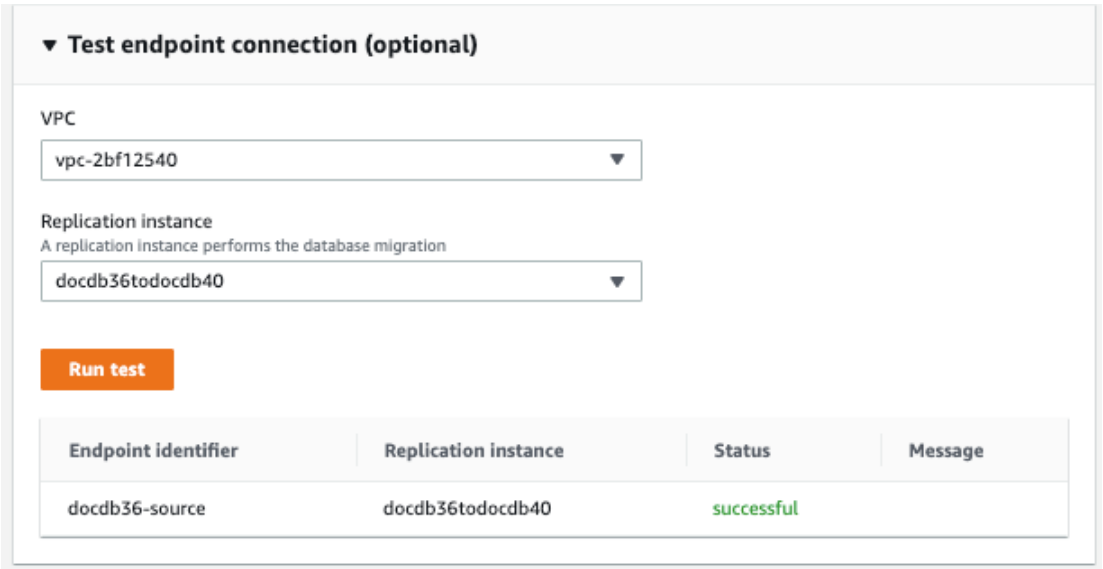
ソースエンドポイントを作成するには

1. AWS DMS コンソールを開きます。
2. ナビゲーションペインで、[エンドポイント] を選択します。
3. `Create endpoint` を選択し、以下の情報を入力します。

   - [Endpoint type (エンドポイントタイプ)] で [Source (ソース)] を選択します。
   - >エンドポイント識別子 については、覚えやすい名前を入力します (例: `docdb-source`)。
   - ソースエンジン については、`docdb` を選択します。
   - サーバー名 については、Amazon DocumentDB クラスターの DNS 名を入力します。
   - ポート については、ソース Amazon DocumentDB クラスターのポート番号を入力します。
   - SSL モード については、`verify-full` を選択します。
   - CA 証明書 については、新しい CA 証明書を追加する を選択します。TLS 接続バンドルを作成するために、新しい CA 証明書 をダウンロードします。証明書識別子 については、`rds-combined-ca-bundle` を入力します。[Import certificate file (証明書ファイルのインポート)] で、[Choose file (ファイルの選択)] を選択し、以前にダウンロードした `.pem` ファイルに移動します。ファイルを選択して開きます。証明書をインポートする を選択し、次に 証明書を選択する のドロップダウンから `rds-combined-ca-bundle` を選択します。
   - [ユーザーネーム] には、アップグレード元の Amazon DocumentDB クラスターのプライマリユーザー名を入力します。
   - [パスワード] には、アップグレード元の Amazon DocumentDB クラスターのプライマリパスワードを入力します。
   - データベース については、アップグレードするデータベース名を入力します。



4. 接続をテストして、正常にセットアップされたことを確認します。



5. エンドポイントの作成 を選択します。

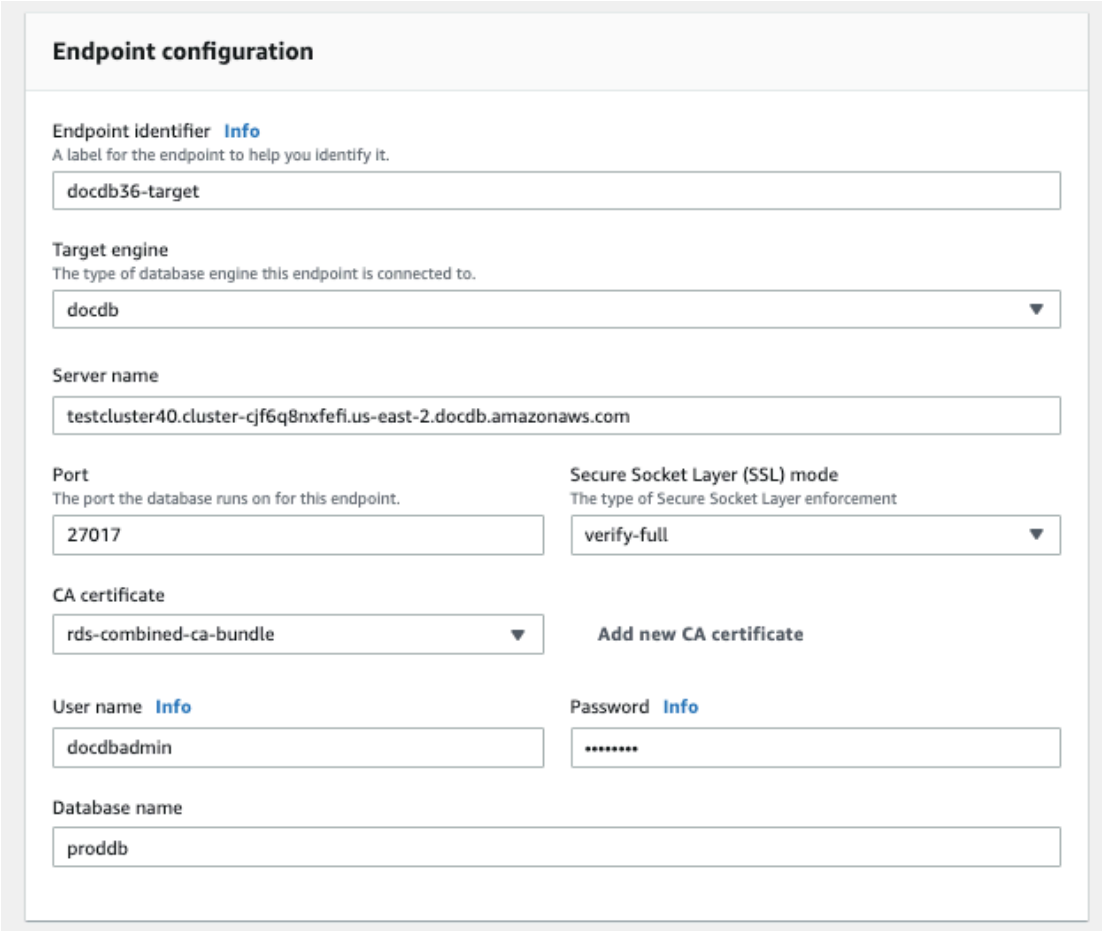
> **Note**
>
> AWS DMS は一度に 1 つのデータベースのみを移行できます。

ステップ 6: AWS DMS ターゲットエンドポイントを作成する

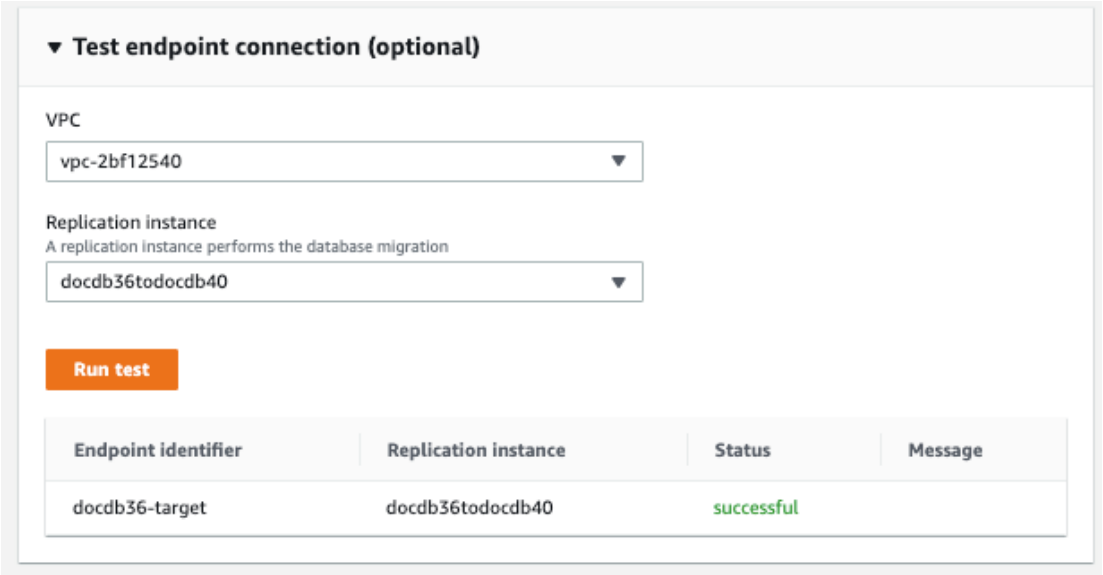
ターゲットエンドポイントは Amazon DocumentDB クラスター用です。

ターゲットエンドポイントを作成するには

1. AWS DMS コンソールを開きます。
2. ナビゲーションペインで、[エンドポイント] を選択します。
3. [Create endpoint (エンドポイントの作成)] を選択し、次の情報を入力します。
   - [Endpoint type (エンドポイントタイプ)] で [Target (ターゲット)] を選択します。
   - [Endpoint identifier (エンドポイント識別子)] に、覚えやすい名前を入力します (例: `docdb-target`)。
   - ソースエンジン については、`docdb` を選択します。
   - サーバー名 については、ターゲット Amazon DocumentDB クラスターの DNS 名を入力します。
   - ポート については、ターゲット Amazon DocumentDB クラスターのポート番号を入力します。
   - SSL モード については、`verify-full` を選択します。
   - CA 証明書については、証明書を選択する というドロップダウンから既存の `rds-combined-ca-bundle` 証明書を選択します。
   - [ユーザーネーム] には、アップグレード先の Amazon DocumentDB クラスターのプライマリユーザー名を入力します。
   - [パスワード] については、アップグレード先の Amazon DocumentDB クラスターのプライマリパスワードを入力します。
   - データベース名 については、ソースエンドポイントの設定に使用したのと同じデータベース名を入力します。



4. 接続をテストして、正常にセットアップされたことを確認します。



5. エンドポイントの作成 を選択します。

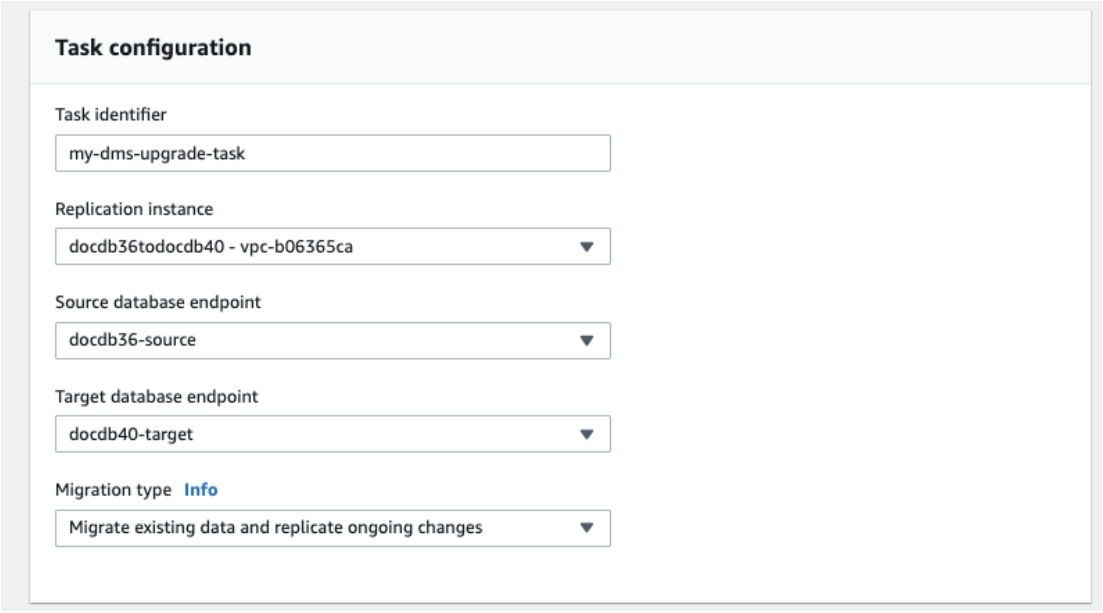
ステップ 7: 移行タスクの作成と実行

AWS DMS タスクは、レプリケーションインスタンスをソースインスタンスとターゲットインスタンスにバインドします。タスクを作成するときに、ソースエンドポイント、ターゲットエンドポイント、レプリケーションインスタンスおよび移行設定を特定します。 AWS DMS タスクは、既存のデータの移行、既存のデータの移行、進行中の変更のレプリケート、データ変更のレプリケートのみの 3 つの異なる移行タイプで作成できます。このウォークスルーの目的は、Amazon DocumentDB クラスターを最小限のダウンタイムでアップグレードすることなので、手順では既存のデータを移行し、進行中の変更をレプリケートするオプションを利用します。このオプションでは、 は既存のデータの移行中に変更を AWS DMS キャプチャします。 は、バルクデータがロードされた後も変更をキャプチャして適用 AWS DMS し続けます。最終的にソースデータベースとターゲットデータベースは同期され、移行に伴うダウンタイムは最小限に抑えられます。

以下は、ダウンタイムを最小限に抑えた移行のための移行タスクの作成手順です。

1. AWS DMS コンソールを開きます。
2. ナビゲーションペインで、[データベース移行タスク] を選択します。
3. [データベース移行タスクの作成] を選択し、[タスク設定] セクションに次の情報を入力します。

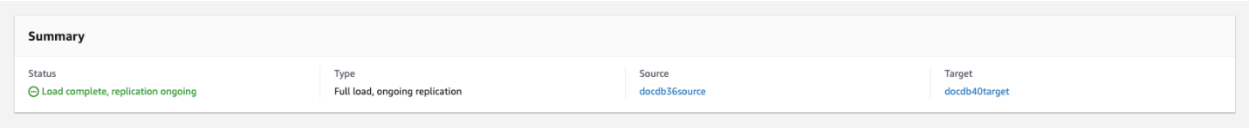
   - [タスク識別子] に、覚えやすい名前を入力します (例: `my-dms-upgrade-task`)。
   - [記述型 Amazon リソースネーム (ARN)] には、デフォルトの DMS ARN を上書きするわかりやすい名前を入力します。
   - [Replication instance (レプリケーションインスタンス)] で、「ステップ 4: AWS DMS レプリケーションインスタンスを作成する」で作成したレプリケーションインスタンスを選択します。
   - [ソースデータベースエンドポイント] で、「ステップ 5: AWS DMS ソースエンドポイントを作成する」で作成したソースエンドポイントを選択します。
   - [ターゲットデータベースエンドポイント] で、「ステップ 6: AWS DMS ターゲットエンドポイントを作成する」で作成したターゲットエンドポイントを選択します。
   - [移行タイプ] では、[移行とレプリケート] を選択します。



4. [タスク設定] セクションで、以下の情報を入力します。

   - [ターゲットテーブル準備モード] セクションで、[何もしない]を選択します。これにより、ステップ 3 で作成したインデックスが削除されないようになります。
   - [タスクログ] のサブセクションで、[CloudWatch ログをオンにする] を選択します。
   - 移行タスクのスタートアップ の設定については、[作成時に自動で設定する] を選択します。これにより、移行タスクを作成すると、自動的に移行タスクが開始されます。
   - [データベース移行タスクの作成] を選択します。

AWS DMS は、ソース Amazon DocumentDB クラスターからターゲット Amazon DocumentDB クラスターへのデータの移行を開始するようになりました。タスクのステータスが、[Starting (開始中)] から [Running (実行中)] に変わります。 AWS DMS コンソールでタスクを選択すると、進行状況をモニタリングできます。数分/時間後 (移行のサイズに応じて)、ステータスが [ロード完了、レプリケーション継続中] に変わります。つまり、 AWS DMS はソース Amazon DocumentDB クラスターからターゲット Amazon DocumentDB クラスターへのフルロード移行を完了し、変更イベントをレプリケートしています。



最終的には、ソースとターゲットが同期します。同期しているかどうかを確認するためには、コレクションの `count()` 操作で、すべての変更イベントが移行されたことを確認できます。

ステップ 8: アプリケーションエンドポイントから Amazon DocumentDB クラスター への変更

フルロードが完了し、CDC プロセスが継続的にレプリケートされると、アプリケーションのデータベース接続エンドポイントをソース Amazon DocumentDB クラスターからターゲット Amazon DocumentDB クラスター クラスターに変更できます。

長期サポート (LTS) リリースの使用

Amazon DocumentDB は、特定のバージョンを長期サポート (LTS) リリースとして指定します。LTS を使用するデータベースクラスターは、LTS 以外のリリースを使用するクラスターよりも長く同じバージョンにとどまり、アップグレードサイクルが少なくなります。LTS バージョンには、重要な安定性とセキュリティパッチの場合にのみパッチが適用され、新機能は導入されません。重大な障害をデバッグするためのメトリクスなど、追加のオプションのパッチがリリースされる場合があります。これらは、必要なパッチで累積的に適用されます。

現在の LTS リリース

Amazon DocumentDB の現在の LTS リリースは次のとおりです。

- マイナーバージョン 5.0.0

クラスターのバージョンを確認するには、「」を参照してくださいthe section called “既存のエンジンバージョン 5.0.0 クラスターが LTS にあるかどうかを確認する方法”。

LTS バージョンのサポートタイムラインとリリースサイクルの詳細については、「」を参照してくださいAmazon DocumentDB エンジンバージョンのサポート日。Amazon DocumentDB は、標準通信チャネルを通じて次の LTS バージョンを発表します。

> **Note**
>
> Amazon DocumentDB 8.0 の LTS リリースはまだ指定されていません。LTS が必要な場合は、エンジンバージョン 5.0.0 とパッチバージョン 3.0.17983 以降を使用します。

LTS を使用するユーザー

Amazon DocumentDB LTS リリースは、アップグレードウィンドウが制限されたクラスター向けに設計されています。LTS は、データベースの可用性が事業運営に直接影響する本番環境に最適です。

以下の場合は LTS リリースを検討してください。

- アプリケーションには、アップグレードの機会を制限する厳格な稼働時間要件があります。
- アプリケーションに必要なすべてのデータベースエンジン機能と機能強化がある

以下の場合は、代わりに標準リリースを検討してください。

- アプリケーションは最小限の中断を処理できます
- リリースされたらすぐに最新の機能と機能強化が必要

> **⚠ Important**
>
> - アップグレードは一方向オペレーションです。アップグレード後にダウングレードすることはできません
> - アップグレードプロセス中にクラスターに短いダウンタイムが発生する

新しい LTS クラスターを作成する方法

LTS クラスターを作成するには、次の手順に従います。

1. マネジメントコンソールで、新しいエンジンバージョン 5.0.0 クラスターを作成します。
2. 以下を使用して、クラスターの作成後にエンジンバージョン 5.0.0 を使用していることを確認します AWS CLI。

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

このオペレーションによる出力は、次のようになります。

```
"5.0.0"
```

エンジンバージョン 3.6.0 または 4.0.0 から 5.0.0 LTS クラスターにアップグレードする方法

LTS クラスターにアップグレードするには、次の手順に従います。

1. マネジメントコンソールで、エンジンバージョン 3.6.0 または 4.0.0 クラスターを 5.0.0 にメジャーバージョンアップグレード (MVU) します。MVU を実行する手順については、「」を参照してくださいAmazon DocumentDB のインプレースメジャーバージョンアップグレード。
2. 以下を使用して、アップグレード後にエンジンバージョン 5.0.0 を使用していることを確認しますAWS CLI。

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

既存のエンジンバージョン 5.0.0 クラスターが LTS にあるかどうかを確認する方法

エンジンバージョン 5.0.0 クラスターで LTS バージョンが実行されているかどうかを確認するには:

1. AWS CLI以下を使用して、エンジンバージョン 5.0.0 を使用していることを確認します。

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

2. 必要に応じて、既存のエンジンバージョン 5.0.0 クラスターをアップグレードします。パッチ更新を実行する手順については、「」を参照してくださいクラスターのエンジンバージョンへのパッチ更新の実行。

Amazon DocumentDB 延長サポート

Amazon DocumentDB 延長サポートを利用すると、標準サポート終了日以降も、エンジンバージョンを引き続き実行できます (追加料金がかかります)。クラスターは、標準サポート終了日を過ぎると

延長サポートに登録されます。エンジンバージョンの標準サポート終了日までにアップグレードを完了しない場合、クラスターには延長サポート料金が請求されます。

トピック

- 延長サポートの概要
- 延長サポート料金
- 責任

Amazon DocumentDB 延長サポートでは、以下の更新とテクニカルサポートが提供されます。

- インスタンスまたはクラスターの 重大および高 CVE に対するセキュリティ更新
- 重大な問題のバグ修正とパッチ
- 標準 Amazon DocumentDB サービスレベル契約の範囲内でサポートケースを開き、トラブルシューティングのヘルプを受け取る機能

この有料サービスを利用すると、サポートされているエンジンバージョンへのアップグレードにかかる時間が長くなります。例えば、Amazon DocumentDB のバージョン 3.6 の標準サポート終了日は 2026 年 3 月 30 日ですが、その日より前にバージョン 5.0 または 8.0 にアップグレードする準備ができていません。この場合、Amazon DocumentDB は 2026 年 3 月 31 日にバージョン 3.6 クラスターを延長サポートに登録するため、Amazon DocumentDB バージョン 3.6 を引き続き使用できます。2026 年 7 月 1 日以降、 は Amazon DocumentDB バージョン 3.6 クラスターの延長サポートプレミアムを AWS 請求します。

Amazon DocumentDB 延長サポートは、エンジンバージョンの標準サポート終了日から最大 3 年間ご利用いただけます。この期間が過ぎてもエンジンバージョンをサポート対象バージョンにアップグレードしていない場合、Amazon DocumentDB はエンジンバージョンをアップグレードします。サポートされているエンジンバージョンにできるだけ早くアップグレードします。

Amazon DocumentDB バージョン 3.6 の標準サポート終了日と延長サポート終了日:

- バージョン 3.6 のリリース — 2019 年 1 月 9 日
- バージョン 3.6 標準サポートの終了 — 2026 年 3 月 30 日
- バージョン 3.6 の開始延長サポート 1 年目の料金 — 2026 年 7 月 1 日
- バージョン 3.6 延長サポート 3 年目の料金の開始 — 2028 年 3 月 31 日
- バージョン 3.6 延長サポートの終了 — 2029 年 3 月 30 日

延長サポートの概要

Amazon DocumentDB は、クラスターがまだそのバージョンでない場合、クラスターをリリースされた最後のエンジンバージョンにアップグレードします (延長サポート終了日前)。アップグレード期間は、そのエンジンバージョンの標準サポート日の後まで発生しません。

そのエンジンバージョンに既にクラスターがある場合は、標準サポート終了日に達したエンジンバージョンで新しいクラスターを作成できます。Amazon DocumentDB は、これらの新しいクラスターを Amazon DocumentDB 延長サポートに登録し、このサービスに対して課金します。バージョン 3.6 の標準サポート終了日より前に Amazon DocumentDB 標準サポートがまだ適用されているエンジンバージョンにアップグレードした場合、延長サポートプレミアムは課金されません。

Amazon DocumentDB 延長サポートへの登録はいつでも終了できます。登録を終了するには、登録した各クラスターを、Amazon DocumentDB 標準サポートがまだ適用されている、より新しいエンジンバージョンにアップグレードします。Amazon DocumentDB 延長サポートへの登録の終了は、Amazon DocumentDB 標準サポートがまだ適用されている、より新しいエンジンバージョンへのアップグレードを完了した日から有効になります。

Amazon DocumentDB の標準サポート終了日と Amazon DocumentDB 延長サポート終了日の詳細については、「Amazon DocumentDB エンジンバージョンのサポート日」を参照してください。

延長サポート料金

2026 年 7 月 1 日以降、延長サポートで実行されているすべてのクラスターに対して料金が発生します。Amazon DocumentDB バージョン 3.6 の標準サポート終了日については、「Amazon DocumentDB エンジンバージョンのサポート日」を参照してください。

Amazon DocumentDB 延長サポートの追加料金は、以下のいずれかのアクションを実行すると、停止します。

- 標準サポートの対象となるエンジンバージョンにアップグレードする
- 標準サポート終了日を過ぎてバージョン 3.6 を実行しているクラスターを削除する

例えば、Amazon DocumentDB バージョン 3.6 は 2026 年 3 月 31 日に延長サポートに入りますが、料金は 2026 年 7 月 1 日まで開始されません。2026 年 7 月 31 日に Amazon DocumentDB バージョン 3.6 クラスターを Amazon DocumentDB バージョン 5.0 にアップグレードした場合、Amazon DocumentDB バージョン 3.6 での 30 日間の延長サポートに対してのみ課金されます。詳細については、Amazon DocumentDB の料金 を参照してください。

責任

Amazon DocumentDB 延長サポートに関する Amazon DocumentDB の責任とお客様の責任は次のとおりです。

Amazon DocumentDB の責任

Amazon DocumentDB バージョン 3.6 の標準サポート終了日を過ぎると、Amazon DocumentDB は延長サポートに登録されているバージョン 3.6 クラスターにパッチ、バグ修正、アップグレードを適用します。

お客様の責任

また、お客様は、Amazon DocumentDB バージョン 3.6 の延長サポート終了日より前にお客様のエンジンをより新しいエンジンバージョンにアップグレードする責任もあります。クラスターをアップグレードしない場合、Amazon DocumentDB バージョン 3.6 延長サポート終了日後、Amazon DocumentDB はクラスターを Amazon DocumentDB 標準サポートでサポートされている最新のエンジンバージョンにアップグレードしようとします。

Amazon DocumentDB の機能と設定

次の表は、Amazon DocumentDB エンジンバージョンと Elastic クラスターの機能、機能、サポートされているインスタンスタイプを比較したものです。これらのテーブルを使用して、ワークロード要件に最適なエンジンバージョンを決定します。

エンジンのステータス

バージョン別のエンジンステータス

機能	v3.6	v4.0	v5.0	v8.0	
サポートステータス	EOL: 2026 年 3 月	アクティブ	アクティブ (LTS が利用可能)	アクティブ (2025 年 11 月 GA)	
延長サポートが利用可能に	はい (2029 年 3 月 まで有料)	該当なし	該当なし	該当なし	
LTS リリースが利用可能に	いいえ	いいえ	はい (v5.0 LTS)	いいえ	
Elastic クラスター	いいえ	なし	はい	いいえ	

クラスタータイプ

バージョン別のクラスタータイプ

クラスタータイプ	v3.6	v4.0	v5.0	v8.0	Elastic クラスター
インスタンスベースのクラスター	はい	はい	はい	はい	はい
サーバーレスインスタンス	いいえ	なし	はい	はい	いいえ

機能

エンジンバージョン別の機能と設定

機能	v3.6	v4.0	v5.0	v8.0	Elastic クラスター
グローバルクラスター	いいえ	はい	はい	はい	いいえ
マルチ AZ (3 AZs)	はい	はい	はい	はい	はい (シャードあたり)
リードレプリカのサポート	15	15	15	15	シャードあたり 15
ACID トランザクション	いいえ	はい	はい	はい	いいえ
再試行可能な書き込み	いいえ	いいえ	いいえ	はい (8.0.2 以降)	いいえ
Change streams	はい	はい	はい	はい	いいえ
ベクトル検索	いいえ	なし	はい	はい	いいえ
Performance Insights	はい	はい	はい	はい	いいえ
インプレースメジャーバージョンアップグレード (MVU)	はい (5.0 または 8.0 まで)	はい (5.0 または 8.0 まで)	はい (8.0 まで)	いいえ (ターゲットのみ)	いいえ
クライアント側のフィールドレベルの暗号化 (FLE)	いいえ	なし	はい	はい	いいえ
I/O 最適化ストレージ	いいえ	なし	はい	はい	いいえ
TLS 証明書のローテーション (再起動なし)	はい (パッチ	はい (パッチ	はい (パッチ	はい	はい

機能	v3.6	v4.0	v5.0	v8.0	Elastic クラスター
	1.0.208662 以降)	2.0.10179 以降)	3.0.4780 以降)		
クロスリージョンスナップショットのコピー	はい	はい	はい	はい	はい
クロスアカウントスナップショット共有	はい	はい	はい	はい	いいえ
監査ログとスロークエリログ	はい	はい	はい	はい	はい
デュアルスタック IPv4/IPv6	いいえ	はい (パッチ 2.0.11747 以降)	はい (パッチ 3.0.15902 以降)	はい	該当なし
照合	いいえ	いいえ	なし	はい	いいえ
ビュー	いいえ	いいえ	なし	はい	いいえ
圧縮のサポート	いいえ	いいえ	LZ4	LZ4/Ztsd	いいえ
テキストインデックス	いいえ	いいえ	V1	V2	いいえ
FedRAMP (GovCloud)	いいえ	なし	はい	はい	いいえ

インスタンスのタイプ

バージョン別のインスタンスタイプのサポート

インスタンスタイプ	v3.6	v4.0	v5.0	v8.0	
db.r4 (旧世代)	はい	いいえ	いいえ	いいえ	
db.r5	はい	はい	はい	はい	

インスタンスタイプ	v3.6	v4.0	v5.0	v8.0	
db.r6g (Graviton2)	いいえ	はい	はい	はい	
db.r6gd (NVMe-backed Graviton2)	いいえ	なし	はい	はい	
db.r8g (Graviton4)	いいえ	なし	はい	はい	
db.t3.medium	はい	はい	はい	はい	
db.t4g.medium (Graviton 2)	いいえ	はい	はい	はい	
サーバーレス	いいえ	なし	はい	はい	

AWS 統合

AWS バージョン別の統合サポート

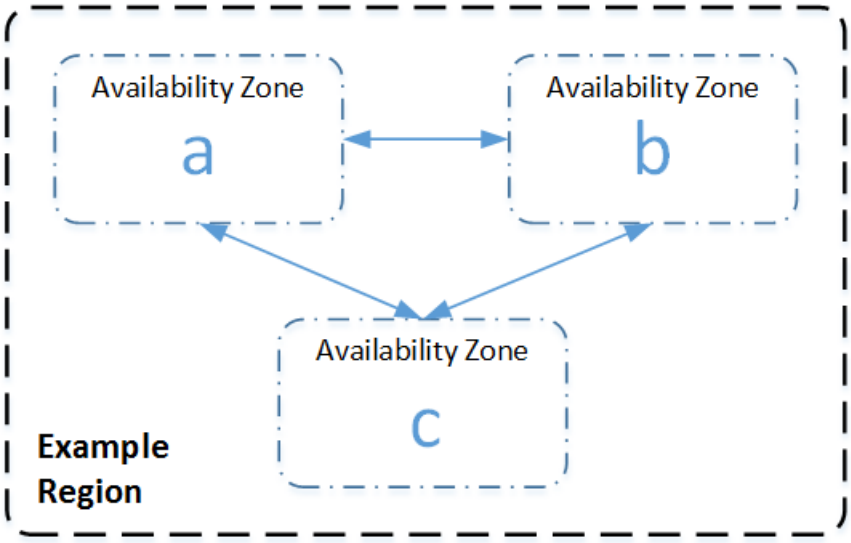
AWS 統合/機能	v3.6	v4.0	v5.0	v8.0	Elastic クラスター
AWS Backup	はい	はい	はい	はい	いいえ
AWS Secrets Manager	はい	はい	はい	はい	いいえ
AWS Lambda (ネイティブ ESM)	いいえ	はい	はい	いいえ	いいえ
Amazon OpenSearch Service Ingestion (ゼロ ETL)	いいえ	はい	はい	はい	いいえ
AWS DMS 移行のサポート	はい	はい	はい	はい (経由で 5.0 から AWS DMS)	はい

AWS 統合/機能	v3.6	v4.0	v5.0	v8.0	Elastic クラスター
AWS Identity and Access Management 認証	いいえ	いいえ	はい (5.0 インスタンスベース)	はい	いいえ
AWS CloudShell	いいえ	なし	はい	はい	いいえ
AWS Key Management Service	はい	はい	はい	はい	はい
AWS CloudTrail	はい	はい	はい	はい	はい

リージョンとアベイラビリティーゾーンの選択

Amazon クラウドコンピューティングリソースは、世界各地の多くの場所でホストされています。これらの口ケーションは、 AWS リージョン および アベイラビリティーゾーンで構成されます。各 AWS リージョン は地理的に離れた領域です。リージョンごとにアベイラビリティーゾーンと呼ばれる複数の独立した場所があります。Amazon DocumentDB では、インスタンスなどのリソースやデータを複数の口ケーションに配置することができます。特に AWS リージョン そうしない限り、リソースは にレプリケートされません。

Amazon は、可用性の高い高度なデータセンターを運用しています。しかし、非常にまれですが、同じ場所にあるインスタンスすべての可用性に影響する障害が発生することもあります。もし、すべてのインスタンスを 1 か所でホストしている場合、そのような障害が起きたとき、インスタンスがすべて利用できなくなります。次の図は、3 つのアベイラビリティーゾーン AWS リージョン を持つを示しています。



各リージョンは独立していることを覚えておくことが重要です。Amazon DocumentDB のアクティビティ (インスタンスの作成や利用可能なインスタンスのリストなど) は、現在のデフォルトの AWS リージョンでしか実行されません。EC2_REGION 環境変数を設定して、コンソールのデフォルトリージョンを変更できます。または、 AWS CLIで --region パラメータを使用して上書きすることもできます。詳細については、「 の設定 AWS Command Line Interface」、特に環境変数とコマンドラインオプションのセクションを参照してください。

Amazon DocumentDB コンソールを使用してクラスターを作成し、別のアベイラビリティーゾーンにレプリカを作成することを選択した場合、Amazon DocumentDB は 2 つのインスタンスを作成します。1 つのアベイラビリティーゾーンでプライマリインスタンスが作成され、別のアベイラビリティーゾーンでレプリカインスタンスが作成されます。クラスターボリュームは常に 3 つのアベイラビリティーゾーンにレプリケートされます。

特定の で Amazon DocumentDB インスタンスを作成または操作するには AWS リージョン、対応するリージョンサービスエンドポイントを使用します。

リージョンの可用性

Amazon DocumentDB は、次の AWS リージョンで使用できます。

Amazon DocumentDB でサポートされているリージョン

リージョン名	リージョン	アベイラビリティーゾーン (コンピューティング)
米国東部 (オハイオ)	us-east-2	3

リージョン名	リージョン	アベイラビリティーゾーン (コンピューティング)
米国東部 (バージニア北部)	us-east-1	6
米国西部 (オレゴン)	us-west-2	4
アフリカ (ケープタウン)	af-south-1	3
南米 (サンパウロ)	sa-east-1	3
アジアパシフィック (香港)	ap-east-1	3
アジアパシフィック (ハイデラバード)	ap-south-2	3
アジアパシフィック (マレーシア)	ap-southeast-5	3
アジアパシフィック (ムンバイ)	ap-south-1	3
アジアパシフィック (大阪)	ap-northeast-3	3
アジアパシフィック (ソウル)	ap-northeast-2	4
アジアパシフィック (シンガポール)	ap-southeast-1	3
アジアパシフィック (シドニー)	ap-southeast-2	3
アジアパシフィック (ジャカルタ)	ap-southeast-3	3

リージョン名	リージョン	アベイラビリティーゾーン (コンピューティング)
アジアパシフィック (メルボルン)	ap-southeast-4	3
アジアパシフィック (タイ)	ap-southeast-7	3
アジアパシフィック (東京)	ap-northeast-1	3
カナダ (中部)	ca-central-1	3
カナダ西部 (カルガリー)	ca-west-1	3
中国 (北京) リージョン	cn-north-1	3
中国 (寧夏)	cn-northwest-1	3
欧州 (フランクフルト)	eu-central-1	3
欧州 (チューリッヒ)	eu-central-2	3
欧州 (アイルランド)	eu-west-1	3
欧州 (ロンドン)	eu-west-2	3
欧州 (ミラノ)	eu-south-1	3
欧州 (パリ)	eu-west-3	3
欧州 (スペイン)	eu-south-2	3
欧州 (ストックホルム)	eu-north-1	3
メキシコ (中部)	mx-central-1	3

リージョン名	リージョン	アベイラビリティーゾーン (コンピューティング)
中東 (アラブ首長国連邦)	me-central-1	3
イスラエル (テルアビブ)	il-central-1	3
AWS GovCloud (米国西部)	us-gov-west-1	3
AWS GovCloud (米国東部)	us-gov-east-1	3

デフォルトでは、Amazon DocumentDB クラスターのタイムゾーンは協定世界時 (UTC) です。

特定のリージョンのクラスターとインスタンスの接続エンドポイントを検索する方法については、「」を参照してくださいAmazon DocumentDB エンドポイントについて。

Amazon OpenSearch Service とのゼロ ETL 統合

トピック

- 送信先としての Amazon OpenSearch Service。
- 制限事項

送信先としての Amazon OpenSearch Service。

OpenSearch Service と Amazon DocumentDB の統合により、フルロードやデータ変更イベントを OpenSearch ドメインにストリーミングすることができます。取り込みインフラストラクチャは OpenSearch 取り込みパイプラインとしてホストされ、Amazon DocumentDB コレクションから継続的にデータをストリーミングするための、高スケールで低レイテンシーなメカニズムを提供します。

フルロード中、ゼロ ETL 統合はまず、取り込みパイプラインを使用してフルロードデータ履歴を OpenSearch に抽出します。フルロードデータが取り込まれると、OpenSearch 取り込みパイプラインは Amazon DocumentDB 変更ストリームからのデータの読み取りを開始し、最終的にすべての変

更を反映させて、Amazon DocumentDB と OpenSearch 間のほぼリアルタイムのデータ整合性を維持します。OpenSearch は、ドキュメントをインデックスに保存します。Amazon DocumentDB コレクションからの受信データは、1 つのインデックスに送信することも、異なるインデックスに分割することもできます。取り込みパイプラインは、Amazon DocumentDB コレクション内のすべての作成、更新、削除イベントを OpenSearch ドキュメントの対応する作成、更新、削除イベントと同期させ、2 つのデータシステムの同期を維持します。取り込みパイプラインは、1 つのコレクションからデータを読み取って 1 つのインデックスに書き込むように設定することも、あるいは 1 つのコレクションからデータを読み取って条件に応じて複数のインデックス間でルーティングするように設定することもできます。

取り込みパイプラインは、以下のモードで Amazon DocumentDB から Amazon OpenSearch Service にデータをストリーミングするように設定できます。

- フルロードのみ
- フルロードなしで Amazon DocumentDB から変更ストリームイベントのみをストリーミングする
- Amazon DocumentDB からまずフルロードを実施し、次に変更ストリームをストリーミングする

取り込みパイプラインを設定するには、次の手順を実行します。

ステップ 1: Amazon OpenSearch Service ドメインまたは OpenSearch サーバーレスコレクションを作成する

データを読み取るための適切なアクセス許可を持つ Amazon OpenSearch Service コレクションが必要です。「Amazon OpenSearch Service 開発者ガイド」の「Amazon OpenSearch Service の使用開始」または「Amazon OpenSearch Serverless の使用開始」でコレクションの作成方法を確認してください。また、「Amazon OpenSearch Service 開発者ガイド」の「Amazon OpenSearch によるデータ取り込み」を参照して、コレクションまたはドメインへの正しい書き込みアクセス許可を取得するための AIM ロールの作成方法を確認してください。

ステップ 2: Amazon DocumentDB クラスターで変更ストリームを有効にする

Amazon DocumentDB クラスター内の必要なコレクションで変更ストリームが有効になっていることを確認します。詳細については、「Amazon DocumentDB を用いた変更ストリームの使用」を参照してください。

ステップ 3: Amazon S3 バケットと送信先のドメインまたはコレクションへの書き込みアクセス許可を持つパイプラインロールを設定する

Amazon DocumentDB コレクションを作成して変更ストリームを有効にしたら、パイプライン設定に使用するパイプラインロールを設定し、次のアクセス許可をロールに付与します。

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/*"
            ]
        }
    ]
}
```

OpenSearch パイプラインが OpenSearch ドメインにデータを書き込めるようにするためには、当該ドメインが、sts_role_arn パイプラインロールにドメインへのアクセスを許可するドメインレベルのアクセスポリシーを持っている必要があります。次のドメインアクセスポリシーの例では、前のステップで作成した `pipeline-role` と言う名前のパイプラインロールに、`ingestion-domain` と言う名前のドメインへの、データの書き込みが許可されています。

```
{
  "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}/*"
    }
  ]
}
```

ステップ 4: X-ENI を作成するために必要なアクセス許可をパイプラインロールに追加する

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" }
            }
        }
    ]
}
```

ステップ 5: パイプラインを作成する

Amazon DocumentDB をソースとして指定する OpenSearch 取り込みパイプラインを設定します。このサンプルパイプライン設定は、変更ストリームフェッチメカニズムの使用を前提としています。詳細については、「Amazon OpenSearch Service 開発者ガイド」の「Amazon DocumentDB での OpenSearch 取り込みパイプラインの使用」を参照してください。

制限事項

Amazon DocumentDB と OpenSearch の統合には、次の制限事項が適用されます。

- パイプラインごとにソースとしてサポートする Amazon DocumentDB コレクションは 1 つのみとなります。
- リージョン間のデータインジェストは、サポートされていません。Amazon DocumentDB クラスターと OpenSearch ドメインは同じ にある必要があります AWS リージョン。
- 複数アカウント間のデータインジェストはサポートされていません。Amazon DocumentDB クラスターと OpenSearch 取り込みパイプラインは、同じ AWS アカウントに存在する必要があります。
- Amazon DocumentDB Elastic クラスターはサポートされていません。Amazon DocumentDB インスタンスベースのクラスターのみがサポートされています。
- Amazon DocumentDB クラスターで AWS secrets. AWS secrets を使用した認証が有効になっていることが、サポートされている唯一の認証メカニズムであることを確認します。
- 既存のパイプライン設定を更新して、別のデータベースや別のコレクションからデータを取り込むことはできません。パイプラインのデータベース名やコレクション名を更新するには、新しいパイプラインを作成する必要があります。

Amazon DocumentDB の操作

このセクションでは、Amazon DocumentDB クラスターへの接続、データのクエリと管理、インデックスの操作、生成 AI アプリケーションの変更ストリーム、トランザクション、ベクトル検索などの機能の使用について説明します。

トピック

- Amazon DocumentDB に接続する
- Amazon DocumentDB を使用したプログラム
- Amazon DocumentDB でのクエリの概要
- インデックス
- Amazon DocumentDB を用いた変更ストリームの使用
- Amazon DocumentDB でのトランザクション
- Amazon DocumentDB での再試行可能な書き込み
- Amazon DocumentDB 8.0 での照合の使用
- Amazon DocumentDB 8.0 でのビューの使用
- Amazon DocumentDB 生成人工知能
- 変更ストリーム AWS Lambda での の使用

Amazon DocumentDB に接続する

Amazon DocumentDB には、MongoDB ドライバーを使用したプログラムによる接続、SSH トンネリングまたは AWS VPN を介した VPC 外部からの接続、Studio 3T や DataGrip などの一般的なツールなど、複数の接続オプションが用意されています。このサービスは、高可用性のためにレプリカセット接続をサポートしています。分析とレポートの場合は、JDBC または ODBC ドライバーを使用して接続できます。同じ VPC 内の EC2 インスタンスは、DocumentDB クラスターに直接接続できます。すべての接続には TLS と適切な AWS 認証情報/認証が必要です。

トピック

- プログラムによる Amazon DocumentDB への接続
- Amazon VPC の外部から Amazon DocumentDB クラスターへの接続
- レプリカセットとしての Amazon DocumentDB に接続する

- Studio 3T から Amazon DocumentDB クラスターに接続する
- DataGrip を使用して Amazon DocumentDB に接続する
- Amazon EC2 を使用した接続
- Amazon DocumentDB JDBC ドライバーを使用して接続する
- Amazon DocumentDB JDBC ドライバーを使用して接続する

プログラムによる Amazon DocumentDB への接続

このセクションでは、複数の言語を使用して Amazon DocumentDB (MongoDB 互換) に接続する方法を示すコード例を紹介します。コード例は、Transport Layer Security (TLS) が有効または無効になっているクラスターに接続しているかどうかに基づいて、2 つのセクションに分かれています。TLS は、新しい Amazon DocumentDB クラスターに対してデフォルトで有効になっています。ただし、必要に応じて TLS を無効にすることができます。詳細については、「Encrypting data in transit」を参照してください。

クラスターが存在する VPC の外部から Amazon DocumentDB に接続しようとする場合は、「」を参照してくださいAmazon VPC の外部から Amazon DocumentDB クラスターへの接続。

クラスターに接続する前に、TLS がクラスターで有効になっているかどうかを把握しておく必要があります。次のセクションでは、`tls` または AWS マネジメントコンソール を使用してクラスターの AWS CLIパラメータの値を確認する方法について説明します。続いて、適切なコード例を見つけて適用する操作について説明します。

トピック

- tls パラメータの値の確認
- TLS が有効な場合の接続
- TLS が無効な場合の接続
- Amazon DocumentDB でのネットワーク圧縮

`tls` パラメータの値の確認

クラスターで TLS が有効になっているかどうかの判断は、 AWS マネジメントコンソール または を使用して実行できる 2 ステップのプロセスです AWS CLI。

1. クラスターを管理しているパラメータグループを確認します。

Using the AWS マネジメントコンソール

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. 左のナビゲーションペインで [Clusters] (クラスター) を選択します。
3. クラスターのリストで、クラスターの名前を選択します。
4. 結果のページには、選択したクラスターの詳細が表示されます。[Configuration (設定)] タブを選択します。[設定とステータス] セクションで、[クラスターパラメータグループ] の下にあるパラメータグループの名前を見つけます。

Using the AWS CLI

次の AWS CLI コードは、クラスターを管理しているパラメータを決定します。*sample-cluster* は、必ずクラスターの名前に置き換えてください。

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

このオペレーションによる出力は、次のようになります。

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

2. クラスターの **tls** パラメータグループでパラメータの値を確認します。

Using the AWS マネジメントコンソール

1. ナビゲーションペインで、[パラメータグループ] を選択します。
2. [クラスターパラメータグループ] ウィンドウで、ステップ 1 からクラスターパラメータグループ名を選択します。

3. 結果ページに、クラスターパラメータグループのパラメータが表示されます。ここで tls パラメータの値を確認できます。このパラメータの変更については、「Amazon DocumentDB クラスターパラメータグループを変更する」を参照してください。

Using the AWS CLI

describe-db-cluster-parameters AWS CLI コマンドを使用して、クラスターパラメータグループのパラメータの詳細を表示できます。

- **--describe-db-cluster-parameters** — パラメータグループ内のすべてのパラメータとその詳細の一覧を表示するには。
  - **--db-cluster-parameter-group name** — 必須。クラスターパラメータグループの名前。

次の例では###############をユーザー自身の情報で置き換えます。

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

このオペレーションによる出力は、次のようになります。

```
{
        "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 は、ca-central-1、us-west-2、us-east-1、us-east-2、us-gov-east-1、us-gov-west-1 の各リージョンで Amazon DocumentDB 5.0 (エンジンバージョン 3.0.3727) クラスターから始まる FIPS 140-3 エンドポイントをサポートしています。

`tls` パラメータの値を確認したら、続いて以下のセクションのコード例のうち 1 つを使用してクラスターに接続します。

- TLS が有効な場合の接続
- TLS が無効な場合の接続

TLS が有効な場合の接続

TLS が有効化されている Amazon DocumentDB クラスターにプログラムで接続するためのコード例を表示するには、使用する言語に該当するタブを選択してください。

転送中のデータを暗号化するには、以下のオペレーションを使用して、 `global-bundle.pem` という Amazon DocumentDB のパブリックキーをダウンロードします。

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

アプリケーションが Microsoft Windows 上にあり、PKCS7 ファイルが必要な場合は、PKCS7 証明書バンドルをダウンロードできます。このバンドルには、https://truststore.pki.rds.amazonaws.com/global/global-bundle.p7b にある中間証明書とルート証明書の両方が含まれています。

Python

次のコードは、TLS が有効になっているときに Python を使用して Amazon DocumentDB に接続する方法を示しています。

次の例では、各 *############* をクラスターの情報に置き換えます。

```
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

次のコードは、TLS が有効になっているときに Python を使用して Amazon DocumentDB に接続する方法を示しています。

> **Important**
>
> なお、バージョン 6.13.1 より前の Node.js ドライバーには既知の制限事項があり、現在 Amazon DocumentDB の IAM ID 認証ではサポートされていません。Node.js ドライバー

を使用する Node.js ドライバーとツール (例えば、mongosh) は、Node.js ドライバーバージョン 6.13.1 以降を使用するようにアップグレードする必要があります。

次の例では、各 ############## をクラスターの情報に置き換えます。

```
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

次のコードは、TLS が有効になっているときに Python を使用して Amazon DocumentDB に接続する方法を示しています。

次の例では、各 *############* をクラスターの情報に置き換えます。

```
<?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

次のコードは、TLS が有効になっているときに Go を使用して Amazon DocumentDB に接続する方法を示しています。

> **Note**
>
> バージョン 1.2.1 現在、MongoDB Go ドライバーは `sslcertificateauthorityfile` にある最初の CA サーバー証明書のみを使用します。次のコード例では、この制限に対処するために、`sslcertificateauthorityfile` にあるすべてのサーバー証明書を、クライアントの作成時に使用されるカスタム TLS 設定に手動で追加しています。

次の例では##############をユーザー自身の情報で置き換えます。

```
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

Java アプリケーションから TLS 対応の Amazon DocumentDB クラスターに接続する場合、プログラムは AWSが提供する認証機関 (CA) ファイルを使用して接続を検証する必要があります。Amazon RDS CA 証明書を使用するには、以下の手順に従ってください。

1. Amazon RDS CA ファイルを https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem ダウンロード。
2. 次のコマンドを実行し、ファイルに含まれている CA 認定を使用して信頼ストアを作成します。*truststore-password* は必ず変更してください。古い CA 認定 (rds-ca-2015-root.pem) と新しい CA 認定 (rds-ca-2019-root.pem) の両方を含む信頼ストアにアクセスする場合は、認定バンドルを信頼ストアにインポートできます。

   Linux オペレーティングシステムで、証明書バンドルを信頼ストアにインポートするサンプルシェルスクリプトを次に示します。次の例では*###############*をユーザー自身の情報で置き換えます。特に、サンプルディレクトリ「*mydir*」がスクリプトに含まれている場合は、このタスク用に作成したディレクトリに置き換えます。

```
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
```

macOS で証明書バンドルを信頼ストアにインポートするサンプルシェルスクリプトを次に示します。

```
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. Amazon DocumentDB クラスターに接続する前に、プログラムで keystore を使用するために、アプリケーションで次のシステムプロパティを設定します。

```
javax.net.ssl.trustStore: truststore
javax.net.ssl.trustStorePassword: truststore-password;
```

4. 次のコードは、TLS が有効になっているときに Java を使用して Amazon DocumentDB に接続する方法を示しています。

   次の例では、各 ############## をクラスターの情報に置き換えます。

```
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

次のコードは、TLS が有効になっているときに C# / .NET を使用して Amazon DocumentDB に接続する方法を示しています。

次の例では、各 *############* をクラスターの情報に置き換えます。

```
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

次のコードは、TLS が有効化されている場合に、最新バージョンの mongosh、または以前の mongo シェルバージョンを使用して Amazon DocumentDB に接続してクエリする方法を示しています。

mongosh を使用して Amazon DocumentDB に接続

> **Important**
>
> なお、バージョン 6.13.1 より前の Node.js ドライバーには既知の制限事項があり、現在 Amazon DocumentDB の IAM ID 認証ではサポートされていません。Node.js ドライバーを使用する Node.js ドライバーとツール (例えば、mongosh) は、Node.js ドライバーバージョン 6.13.1 以降を使用するようにアップグレードする必要があります。

次の例では*##############*をユーザー自身の情報で置き換えます。

```
mongosh --tls --host cluster-end-point:27017 --tlsCAFile  global-bundle.pem --
username sample-user --password password --retryWrites false
```

以前の mongo シェルバージョンを使用して Amazon DocumentDB に接続

IAM を使用する場合は、mongo シェルの以前のバージョンを使用する必要があります。次のコマンドオプションのいずれかを入力します。

```
mongo --ssl --host cluster-end-point:27017 --sslCAFile global-bundle.pem --
username sample-user --password password
```

4.2 かそれ以降のバージョンの Mongo シェルを使用している場合は、次のコードを使用して接続します。エンジンバージョン 8.0.2 以降、Amazon DocumentDB は再試行可能な書き込みをサポートしています。以前のエンジンバージョンでは、再試行可能な書き込みはサポートされていません。例外: mongo シェル を使用している場合は、どのコード文字列にも `retryWrites=false` コマンドを含めないでください。詳細については、「Amazon DocumentDB での再試行可能な書き込み」を参照してください。

```
mongo --tls --host cluster-end-point:27017 --tlsCAFile global-bundle.pem --
username sample-user --password password
```

接続をテストする

1. 単一のドキュメントを挿入します。

```
db.myTestCollection.insertOne({'hello':'Amazon DocumentDB'})
```

2. 以前に挿入されたドキュメントを検索します。

```
db.myTestCollection.find({'hello':'Amazon DocumentDB'})
```

R

次のコードは、TLS が有効になっているときに mongolite (https://jeroen.github.io/mongolite/) を使用して Amazon DocumentDB に接続する方法を示しています。

次の例では、各 ############## をクラスターの情報に置き換えます。

```
#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

次のコードは、TLS が有効になっているときに Ruby を使用して Amazon DocumentDB に接続する方法を示しています。

次の例では、各 ############## をクラスターの情報に置き換えます。

```
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
```

TLS が無効な場合の接続

TLS が無効化されている Amazon DocumentDB クラスターにプログラムで接続するためのコード例を表示するには、使用する言語のタブを選択してください。

Python

次のコードは、TLS が無効になっているときに Python を使用して Amazon DocumentDB に接続する方法を示しています。

次の例では、各 *############* をクラスターの情報に置き換えます。

```
## 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

次のコードは、TLS が無効になっているときに Node.js を使用して Amazon DocumentDB に接続する方法を示しています。

> **Important**
>
> なお、バージョン 6.13.1 より前の Node.js ドライバーには既知の制限事項があり、現在 Amazon DocumentDB の IAM ID 認証ではサポートされていません。Node.js ドライバーを使用する Node.js ドライバーとツール (例えば、mongosh) は、Node.js ドライバーバージョン 6.13.1 以降を使用するようにアップグレードする必要があります。

次の例では、各 *############* をクラスターの情報に置き換えます。

```
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

次のコードは、TLS が無効になっているときに PHP を使用して Amazon DocumentDB に接続する方法を示しています。

次の例では、各 ############## をクラスターの情報に置き換えます。

```
<?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

次のコードは、TLS が無効になっているときに Go を使用して Amazon DocumentDB に接続する方法を示しています。

次の例では、各 ############## をクラスターの情報に置き換えます。

```
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

次のコードは、TLS が無効になっているときに Java を使用して Amazon DocumentDB に接続する方法を示しています。

次の例では、各 *############* をクラスターの情報に置き換えます。

```
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

次のコードは、TLS が無効になっているときに C# / .NET を使用して Amazon DocumentDB に接続する方法を示しています。

次の例では、各 *############* をクラスターの情報に置き換えます。

```
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

次のコードは、TLS が無効化されている場合に、最新バージョンの mongosh、または以前の mongo シェルバージョンを使用して Amazon DocumentDB に接続してクエリする方法を示しています。

mongosh を使用して Amazon DocumentDB に接続

> **Important**
>
> なお、バージョン 6.13.1 より前の Node.js ドライバーには既知の制限事項があり、現在 Amazon DocumentDB の IAM ID 認証ではサポートされていません。Node.js ドライバーを使用する Node.js ドライバーとツール (例えば、mongosh) は、Node.js ドライバーバージョン 6.13.1 以降を使用するようにアップグレードする必要があります。

次の例では*##############*をユーザー自身の情報で置き換えます。

```
mongosh --host cluster-end-point:27017 --username sample-user --password password --
retryWrites false
```

以前の mongo シェルバージョンを使用して Amazon DocumentDB に接続

IAM を使用する場合は、mongo シェルの以前のバージョンを使用する必要があります。次のコマンドオプションのいずれかを入力します。

```
mongo --host cluster-end-point:27017 --username sample-user --password password
```

4.2 かそれ以降のバージョンの Mongo シェルを使用している場合は、次のコードを使用して接続します。再試行可能な書き込みは Amazon DocumentDB ではサポートされていません。例外: mongo シェル を使用している場合は、どのコード文字列にも `retryWrites=false` コマンドを含めないでください。デフォルトでは、再試行可能な書き込みは無効になっています。`retryWrites=false` を含めると、通常の読み取りコマンドでは失敗する可能性があります。

```
mongo --host cluster-end-point:27017 --username sample-user --password password
```

接続をテストする

1. 単一のドキュメントを挿入します。

   ```
   db.myTestCollection.insertOne({'hello':'Amazon DocumentDB'})
   ```

2. 以前に挿入されたドキュメントを検索します。

   ```
   db.myTestCollection.find({'hello':'Amazon DocumentDB'})
   ```

R

次のコードは、TLS が無効になっているときに mongolite (https://jeroen.github.io/mongolite/) を使用して Amazon DocumentDB に接続する方法を示しています。

次の例では、各 `##############` をクラスターの情報に置き換えます。

```
#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

次のコードは、TLS が無効になっているときに Ruby を使用して Amazon DocumentDB に接続する方法を示しています。

次の例では、各 *############* をクラスターの情報に置き換えます。

```
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 でのネットワーク圧縮

ワイヤプロトコル圧縮を使用すると、アプリケーションがネットワーク経由で送信するデータを減らすことができます。Amazon DocumentDB 8.0.1 以降では、圧縮がサポートされています。大きなドキュメントや大きな結果セットを転送するワークロードのスループットが向上します。圧縮を有効にすると、ドライバーとサーバーは圧縮メッセージを交換します。

ネットワーク圧縮を有効にすると、CPU リソースをネットワーク帯域幅の削減と交換できます。ワークロードがネットワークバインドされていて、大きなペイロードを転送する場合は、圧縮を有効にします。小さなリクエストやレスポンスでは、圧縮オーバーヘッドが削減額を超える可能性があるため、純利益が得られない場合があります。

サポートされている圧縮機

Amazon DocumentDB は、 `snappy` 圧縮機をサポートしています。Amazon DocumentDB は、MongoDB ドライバーが提供する `zlib`や などの他のコンプレッサーをサポートしていません`zstd`。

ドライバーがコンプレッサーのリストをリクエストした場合、Amazon DocumentDB はサポートされているもののみをネゴシエートします。たとえば、ドライバーが `snappy`と を提供する場合`zstd`、Amazon DocumentDB は を選択`snappy`および無視します`zstd`。ドライバーがサポートされていない圧縮機のみを提供する場合、Amazon DocumentDB は接続を圧縮しません。この場合、ドライバーとサーバーは圧縮せずにメッセージを交換します。

圧縮を有効にする

圧縮を有効にするには、接続文字列に `compressors`オプションを追加し、 に設定します`snappy`。次の接続文字列*`docdb-cluster-endpoint`*で、*`username`*、*`password`*、 を独自の値に置き換えます。

```
mongodb://username:password@docdb-cluster-endpoint:27017/?
tls=true&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false&compressors=snappy
```

MongoDB ドライバーを使用する場合は、ドライバーの接続オプションを使用して同じオプションを設定します。ほとんどのドライバーでは、スナップ圧縮を実行するために追加のクライアント側の依存関係が必要です。

- Node.js ドライバーには `snappy`パッケージが必要です。
- Python ドライバー (PyMongo) には `python-snappy`パッケージが必要です。Linux では、 にはシステムスナップライブラリ (`snappy-devel`または など) `dnf` `python-snappy`も必要です`yum`。

正確なオプション名と必要なパッケージについては、ドライバーのドキュメントを参照してください。

> **Note**
>
> クライアント側の圧縮依存関係がない場合、ドライバーは正常に接続されます。ただし、圧縮せずにメッセージを交換し、エラーは返しません。圧縮がアクティブであることを確認するには、「」の説明に従ってネゴシエートされた圧縮機を確認します圧縮の検証。

次の Python コード例では、スナップ圧縮を有効にしてクラスターに接続し、 `try` および `except`ブロックを使用して接続エラーを処理します。

```
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}')
```

圧縮の検証

ドライバーと Amazon DocumentDB が圧縮をネゴシエートしたことを確認するには、helloMongoDB ワイヤプロトコルコマンドが返す `compression` フィールドを検査します。ドライバーと Amazon DocumentDB が圧縮をネゴシエートすると、このフィールドには合意された圧縮機が一覧表示されます。

Amazon VPC の外部から Amazon DocumentDB クラスターへの接続

Amazon DocumentDB (MongoDB 互換) クラスターは、Amazon VPC (Amazon Virtual Private Cloud) 内にデプロイされます。これらは、Amazon EC2 インスタンス、または同じ Amazon VPC にデプロイされている他の AWS サービスから直接アクセスできます。さらに、Amazon DocumentDB

は、VPC ピアリングを介して、同じ AWS リージョン リージョンまたは他のリージョンの異なる VPCs 内の EC2 インスタンスまたは他の AWS サービスからアクセスできます。

ただし、お客様のユースケースで、お客様 (またはお客様のアプリケーション) が、クラスターの VPC の外部から Amazon DocumentDB リソースにアクセスする必要があるとします。その場合は、SSH トンネリング (ポート転送 とも呼ばれる) を使用して、Amazon DocumentDB リソースにアクセスできます。

SSH トンネリングに関する詳細な説明は、このトピックでは扱いません。SSH トンネリングの詳細については、次のドキュメントを参照してください。

- SSH トンネル
- SSH ポート転送の例、特に「ローカル転送」セクション

SSH トンネルを作成するには、Amazon DocumentDB クラスターと同じ Amazon VPC で実行中の Amazon EC2 インスタンスが必要です。クラスターと同じ VPC で既存の EC2 インスタンスを使用するか、新しい EC2 インスタンスを作成できます。詳細については、以下でオペレーティングシステムに該当するトピックを参照してください。

- Amazon EC2 Linux インスタンスの使用開始
- Amazon EC2 Windows インスタンスの開始方法

通常、以下のコマンドを使用して EC2 インスタンスに接続します。

```
ssh -i "ec2Access.pem" ubuntu@ec2-34-229-221-164.compute-1.amazonaws.com
```

その場合、ローカルコンピュータで以下のコマンドを実行することにより Amazon DocumentDB クラスター`sample-cluster.node.us-east-1.docdb.amazonaws.com` に SSH トンネルを設定できます。ローカルポートの転送を行う場合は、`-L` フラグを使用します。SSH トンネルを使用する場合は、クラスターエンドポイントを使用してクラスターに接続し、エラーが発生するため、レプリカセットモード (接続文字列`replicaSet=rs0`で を指定) で接続しようとしないでください。

```
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
```

SSH トンネルが作成されると、`localhost:27017` に対して発行するすべてのコマンドは、Amazon VPC で実行中の Amazon DocumentDB クラスター `sample-cluster` に転送されます。Amazon DocumentDB クラスターで TLS (Transport Layer Security) が有効になっている場合

は、Amazon DocumentDB のhttps://truststore.pki.rds.amazonaws.com/global/global-bundle.pem パブリックキーをダウンロードする必要があります。次のオペレーションでは、このファイルをダウンロードします。

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

> **Note**
>
> TLS は、新しい Amazon DocumentDB クラスターに対してデフォルトで有効になっています。ただし、無効にすることは可能です。詳細については、「Amazon DocumentDB クラスター TLS 設定の管理」を参照してください。

Amazon VPC の外部から Amazon DocumentDB クラスターに接続するには、以下のコマンドを使用します。

```
mongo --sslAllowInvalidHostnames --ssl --sslCAFile global-bundle.pem --username
 <yourUsername> --password <yourPassword>
```

レプリカセットとしての Amazon DocumentDB に接続する

Amazon DocumentDB に対して開発する場合は、レプリカセットとしてクラスターに接続し、ドライバーの組み込み読み取り設定機能を使用してレプリカインスタンスに読み取りを分散します。このセクションでは、その意味について詳しく説明し、Python を例として使用してレプリカセットとして Amazon DocumentDB クラスターに接続する方法について説明します。

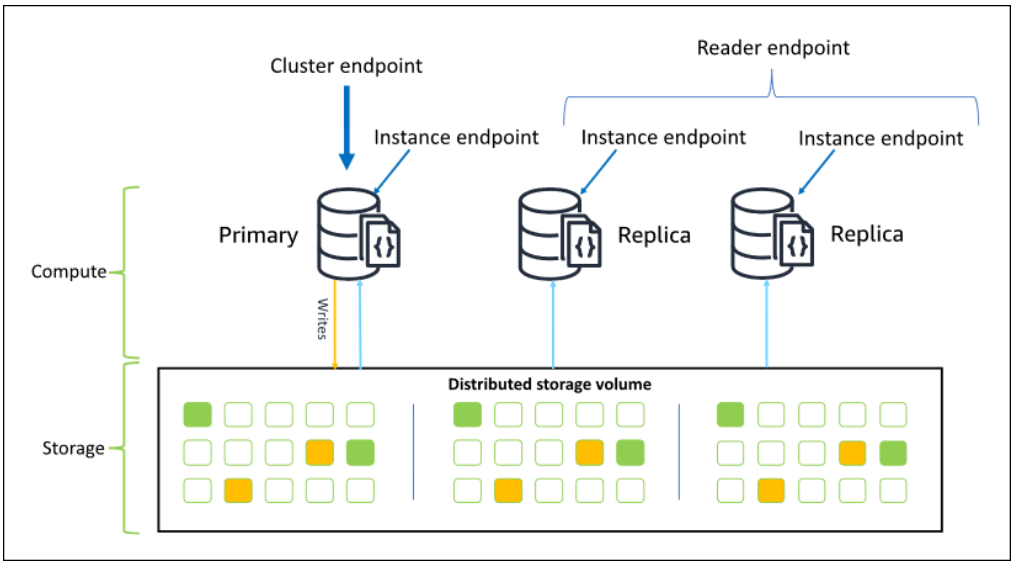
Amazon DocumentDB には、クラスターへの接続に使用できる 3 つのエンドポイントがあります。

- クラスターエンドポイント
- リーダーエンドポイント
- インスタンスエンドポイント

Amazon DocumentDB に接続するときは、クラスターエンドポイントを使用します。これは、次の図に示すように、クラスター内のプライマリインスタンスをポイントする CNAME です。

SSH トンネルを使用する場合は、クラスターエンドポイントを使用してクラスターに接続します。エラーが発生するため、レプリカセットモード (接続文字列`replicaSet=rs0`で を指定) で接続しないでください。

> **Note**
>
> Amazon DocumentDB ウェブサイトのエンドポイントの詳細については、「Amazon DocumentDB エンドポイント」を参照してください。



クラスターエンドポイントを使用して、レプリカセットモードでクラスターに接続できます。その後、組み込まれた読み込み設定ドライバー機能を使用できます。次の例では、/?replicaSet=rs0 を指定することで、レプリカセットとして接続することを SDK を示します。/?replicaSet=rs0' を省略すると、クライアントはすべてのリクエストをプライマリインスタンスであるクラスターエンドポイントにルーティングします。

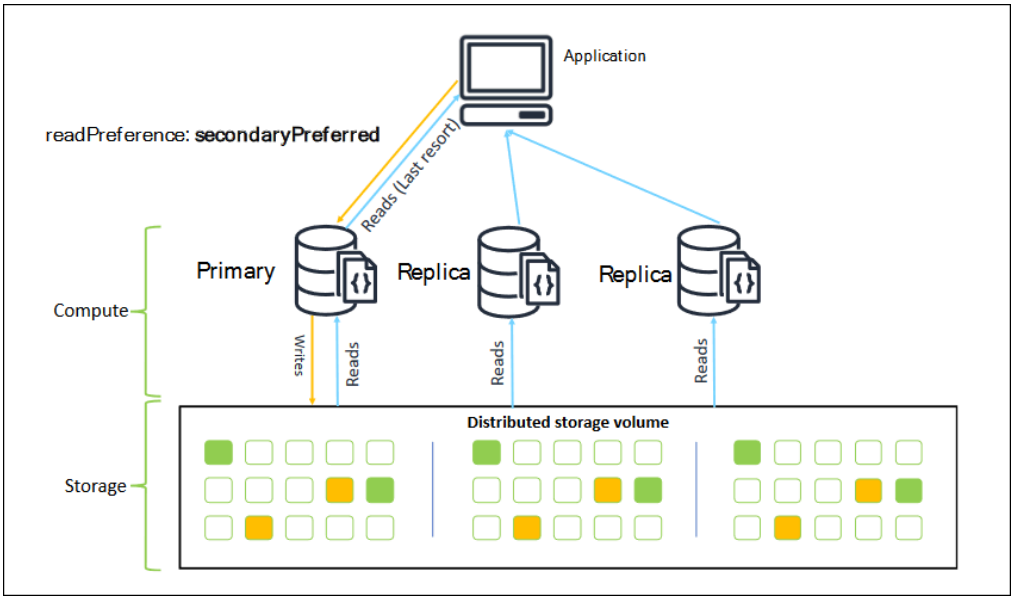
```
## Create a MongoDB client, open a connection to Amazon DocumentDB as a
   ## replica set and specify the read preference as secondary preferred
   client = pymongo.MongoClient('mongodb://<user-name>:<password>@mycluster.node.us-
east-1.docdb.amazonaws.com:27017/?replicaSet=rs0')
```

レプリカセットとして接続する利点は、インスタンスがクラスターに対して追加または削除されるタイミングを含め、SDK がクラスタートポグラフィを自動的に検出できることです。その後、レプリカインスタンスに読み込みリクエストをルーティングすることで、クラスターをより効率的に使用できます。

レプリカセットとして接続するときは、接続の readPreference を指定できます。secondaryPreferred の読み込み設定を指定した場合、クライアントは (以下の図のように) 読み取りクエリをレプリカにルーティングし、書き込みクエリをプライマリインスタンスにルー

ティングします。これにより、クラスターリソースをより適切に利用できます。詳細については、「読み込み設定のオプション」を参照してください。

```
## 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レプリカからの読み込みには結果整合性があります。プライマリに書き込まれたのと同じ順序でデータを返し、50 ミリ秒未満のレプリケーション遅延もよくあります。Amazon CloudWatch メトリクス DBInstanceReplicaLag と DBClusterReplicaLagMaximum を使用して、クラスターのレプリカラグをモニタリングできます。詳細については、「Amazon DocumentDB と CloudWatch のモニタリング」を参照してください。

従来のモノリシックデータベースアーキテクチャとは異なり、Amazon DocumentDB はストレージとコンピューティングを分離します。この最新のアーキテクチャでは、レプリカインスタンスの読み取りのスケールをお勧めします。レプリカインスタンスの読み込みは、プライマリインスタンスからレプリケートされる書き込みをブロックしません。クラスターに最大 15 個のリードレプリカインスタンスを追加し、1 秒あたり数百万回の読み込みにスケールアウトできます。

レプリカセットとして接続し、レプリカに読み取りを分散する主な利点は、アプリケーションの作業に利用できるクラスター内のリソース全体が増加することです。ベストプラクティスとして、レプリカセットとして接続します。これは、以下のシナリオで最も有益です。

- プライマリでほぼ 100% の CPU を使用している。

- バッファキャッシュヒット率がほぼゼロである。
- 個別のインスタンスで接続またはカーソルの制限に達する。

クラスターインスタンスサイズのスケールアップは 1 つの選択肢であり、場合によっては、これがクラスターをスケーリングするための最良の方法です。ただし、クラスターにすでに存在するレプリカをより適切に使用する方法も考慮する必要があります。これにより、より大きいインスタンスタイプを使用することでコストが増大することなく、スケールを増加できます。また、CloudWatch アラームを使用してこれらの制限 (`CPUUtilization`、`DatabaseConnections`、および `BufferCacheHitRatio`) をモニタリングおよびアラートし、リソースが頻繁に使用されているタイミングを把握することをお勧めします。

詳細については、以下の各トピックを参照してください。

- Amazon DocumentDB のベストプラクティス
- Amazon DocumentDB クォータ

クラスター接続の使用

クラスター内のすべての接続を使用するシナリオを検討します。たとえば、`r5.2xlarge` インスタンスの接続は 4,500 個 (および開いているカーソルは 450 個) に制限されています。3 つのインスタンスの Amazon DocumentDB クラスターを作成し、クラスターエンドポイントを使用してプライマリインスタンスにのみ接続する場合、開いている接続とカーソルのクラスター制限はそれぞれ 4,500 と 450 になります。コンテナでスピンアップする多くのワーカーを使用するアプリケーションを構築する場合、これらの制限に達する可能性があります。コンテナは、一度に多数の接続を開き、クラスターを飽和させます。

代わりに、レプリカセットとして Amazon DocumentDB クラスターに接続し、レプリカインスタンスに読み取りを分散できます。その後、クラスターで使用できる接続とカーソルの数をそれぞれ 13,500 と 1,350 に実質的に 3 倍にできます。クラスターにインスタンスを追加すると、読み取りワークロードの接続とカーソルの数が増えるだけです。クラスターへの書き込みの接続数を増やす必要がある場合は、インスタンスサイズを大きくすることをお勧めします。

> **Note**
>
> `large` インスタンス、`xlarge` インスタンス、および `2xlarge` インスタンスの接続数は、インスタンスサイズが最大 4,500 に達するまで増加します。`4xlarge` インスタンス以上の

場合、インスタンスあたりの最大接続数は 4,500 です。インスタンスタイプごとの制限の詳細については、「インスタンスクォータ」を参照してください。

通常、secondary の読み込み設定を使用してクラスターに接続することはお勧めしません。これは、クラスターにレプリカインスタンスがない場合、読み込みが失敗するためです。例えば、1 つのプライマリと 1 つのレプリカを持つ 2 つのインスタンスの Amazon DocumentDB クラスターがあるとします。レプリカに問題がある場合、secondary に設定されている接続プールからの読み取りリクエストは失敗します。secondaryPreferred の利点は、クライアントが接続する適切なレプリカインスタンスを見つけることができない場合、読み取りのためにプライマリにフォールバックすることです。

複数の接続プール

シナリオによっては、アプリケーションの読み取りに、書き込み後読み取り整合性が必要です。これは、Amazon DocumentDB のプライマリインスタンスからのみ提供できます。このようなシナリオでは、2 つのクライアント接続プールを作成できます。1 つは書き込み用、もう 1 つは書き込み後読み込み整合性を必要とする読み込み用です。そのためには、コードは次のようになります。

```
## 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')
```

もう 1 つのオプションは、1 つの接続プールを作成し、特定のコレクションの読み込み設定を上書きすることです。

```
##Specify the collection and set the read preference level for that collection
col = db.review.with_options(read_preference=ReadPreference.SECONDARY_PREFERRED)
```

概要

クラスター内のリソースをより適切に使用するには、レプリカセットモードを使用して接続します。アプリケーションに適している場合は、レプリカインスタンスに読み取りを分散することで、アプリケーションの読み取りをスケーリングできます。

Studio 3T から Amazon DocumentDB クラスターに接続する

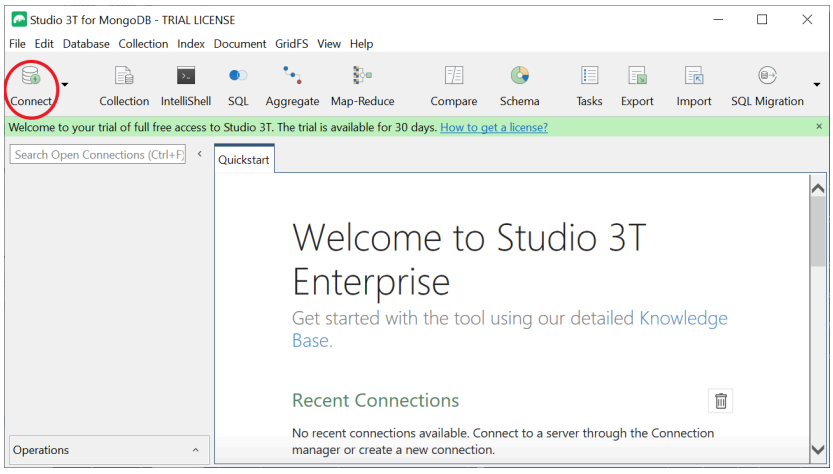
Studio 3T は、MongoDB で作業するデベロッパーやデータエンジニアにとって人気のある GUI と IDE です。データのツリー、表、および JSON ビュー、CSV、JSON、SQL および BSON/MongoDump での簡単なインポート/エクスポート、柔軟なクエリオプション、視覚的なドラッグアンドドロップ UI、自動補完機能を備えた組み込みの Mongo シェル、集計パイプラインエディタ、および SQL クエリのサポートを提供しています。

前提条件

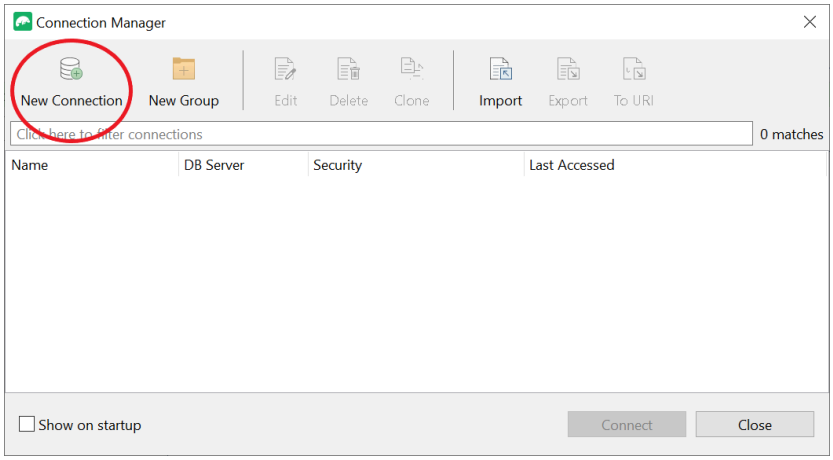
- Amazon EC2 を踏み台/ジャンプホストとして使用する Amazon DocumentDB クラスターをまだお持ちでない場合は、「」を参照してくださいAmazon EC2 を使用した接続。
- Studio 3T をお持ちでない場合は、ダウンロードしてインストールします。

Studio 3T に接続する

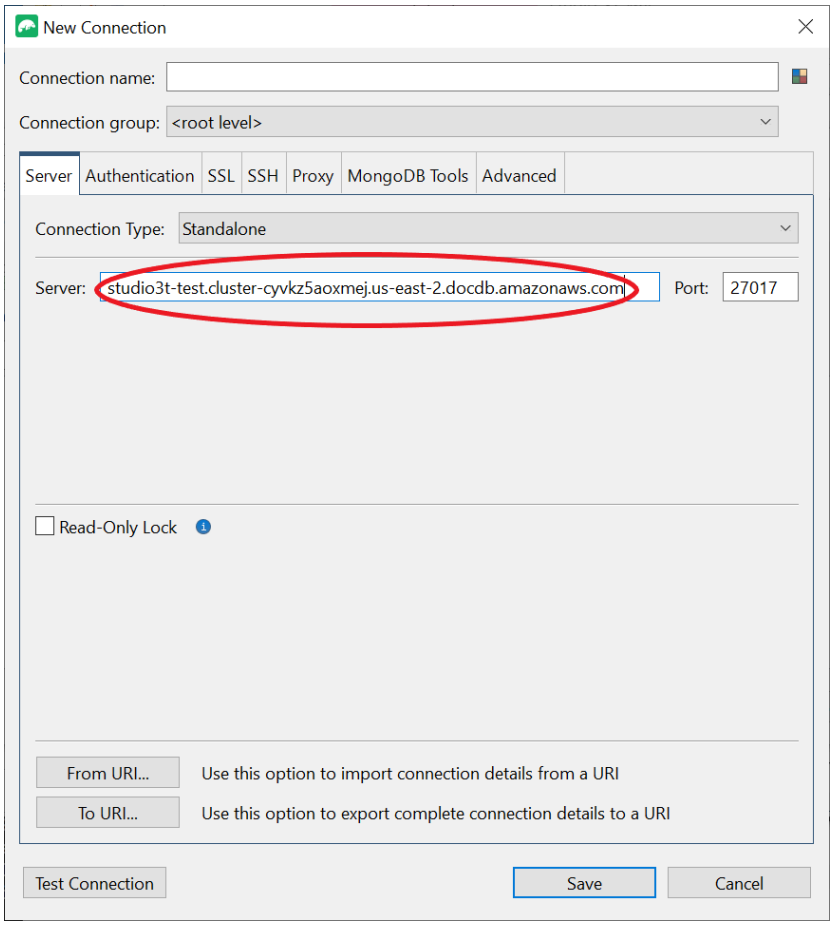
1. ツールバーの左上隅にある 接続 を選択します。



2. ツールバーの左上隅にある 新規接続 を選択します。

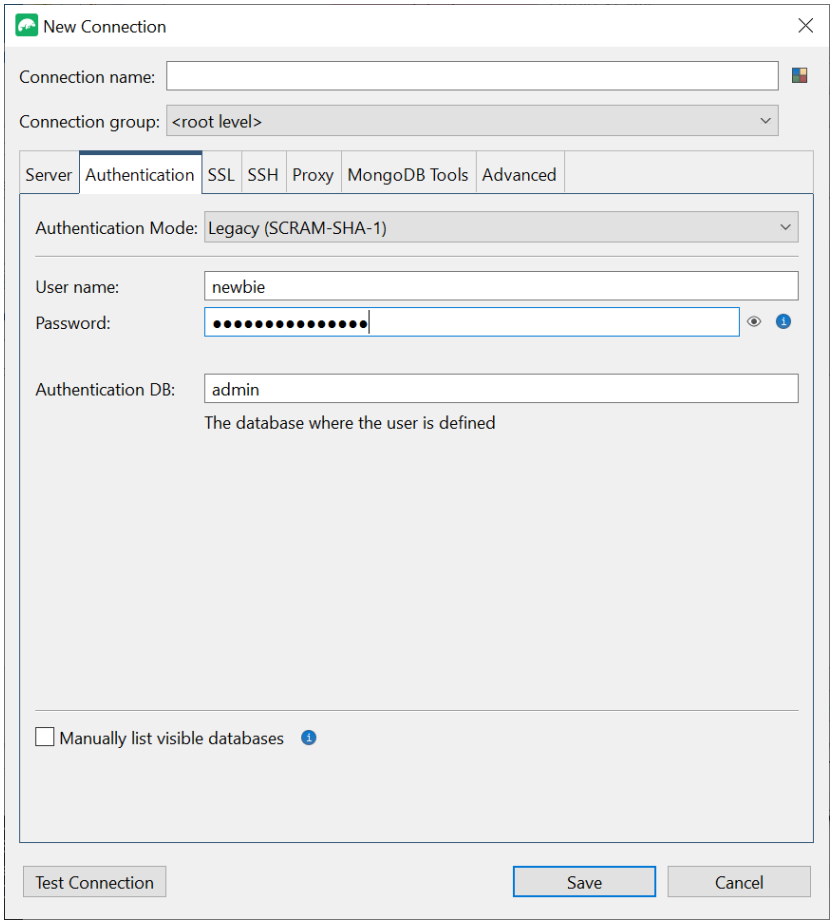


3. サーバー タブの サーバー フィールドに、クラスターエンドポイント情報を入力します。

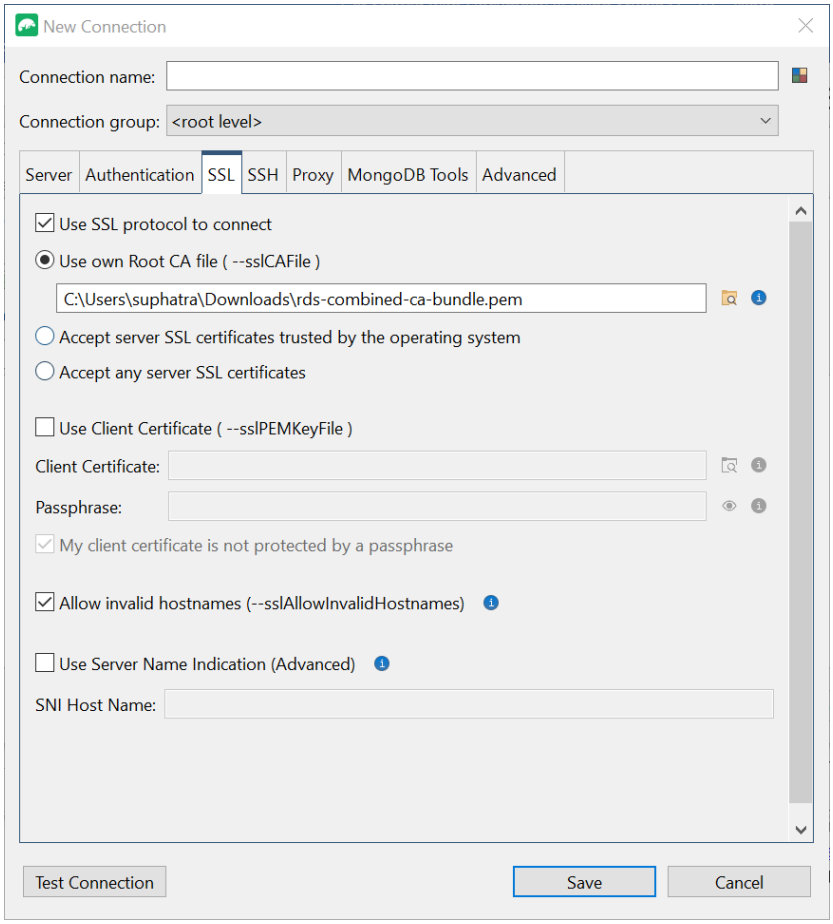


> **Note**
>
> クラスターエンドポイントを検索するには、「」を参照してくださいインスタンスのエンドポイントの検索。

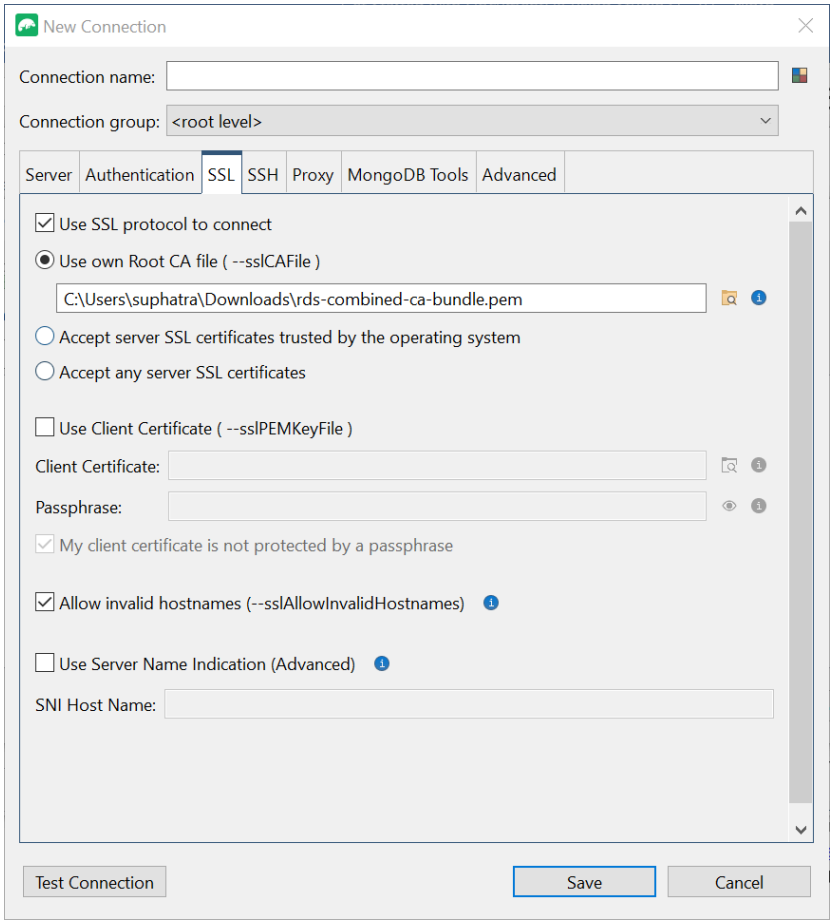
4. 認証 タブを選択し、認証モード のドロップダウンメニューで レガシー を選択します。



5. ユーザー名 フィールドと パスワード フィールドにユーザーネームと資格情報を入力します。
6. SSL タブを選択し、SSLプロトコルを使用して接続する チェックボックスをオンにします。



7. 独自のルート CA ファイルを使用する を選択します。次に、Amazon DocumentDB 証明書を追加します (DocumentDB クラスターで SSL が無効になっている場合は、このステップをスキップできます)。無効なホスト名 を許可するには、チェックボックスをオンにします。



> **Note**
>
> 証明書を持っていない場合。次のコマンドでダウンロードできます。
>
> ```
> wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
> ```

8. Amazon VPC 外のクライアントマシンから接続する場合は、SSH トンネルを作成する必要があります。SSH タブでこれを行います。

   a. SSHトンネルを使用する のチェックボックスをオンにして、SSHアドレス フィールドに SSH アドレスを入力します。これはインスタンスの公開 DNS (IPV4) です。この URL は Amazon EC2 マネジメントコンソール から取得できます。

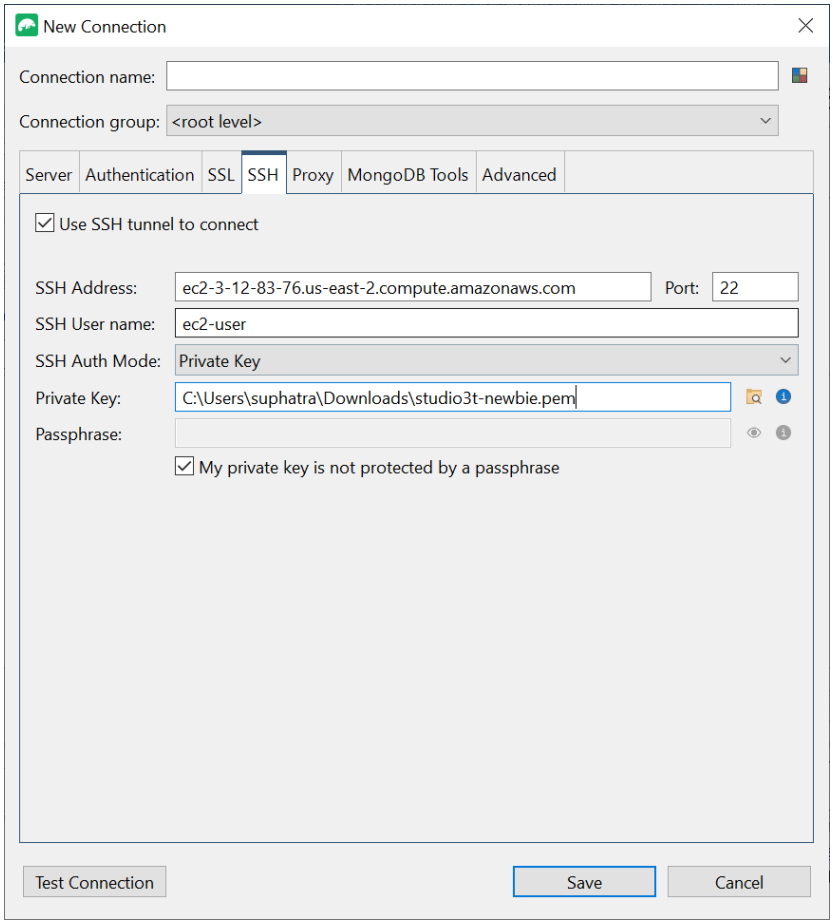
   b. ユーザーネームを入力します。これは Amazon EC2 インスタンスのユーザーネームです。

   c. SSH 認証モードで、プライベートキー を選択します。プライベートキー フィールドで、ファイルファインダーアイコンを選択し、Amazon EC2 インスタンスのプライベートキー

を選択します。これは、Amazon EC2 コンソールでインスタンスを作成するときに保存した.pem ファイル (キーペア) です。

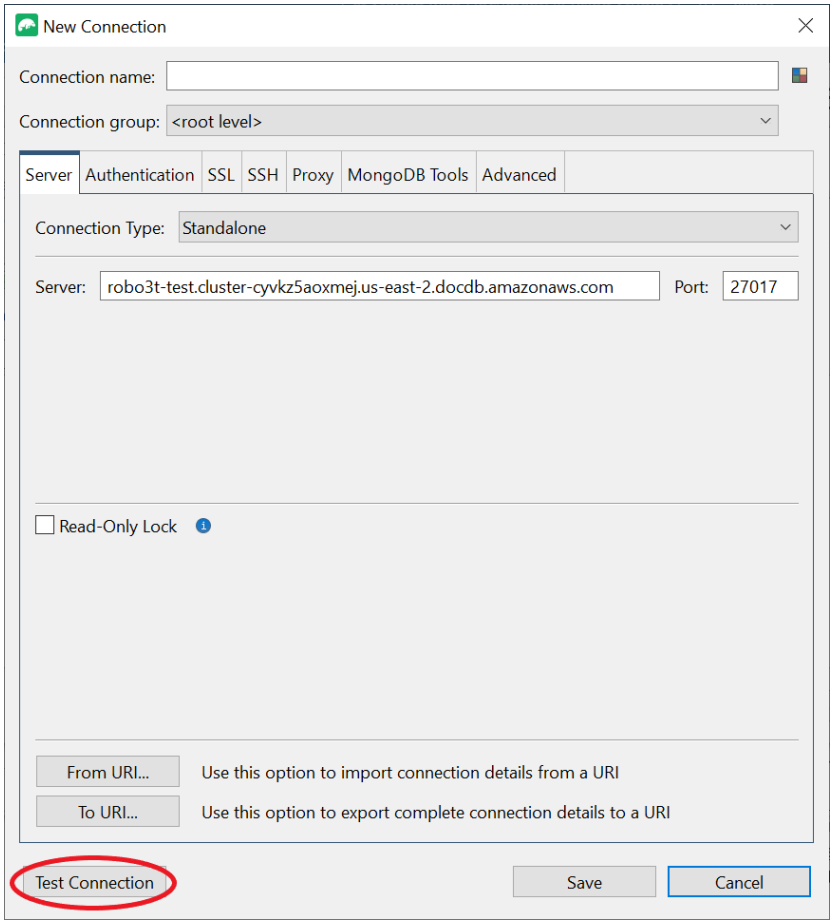
d. Linux/macOS クライアントマシンを使用している場合は、次のコマンドを使用してプライベートキーの許可を変更する必要があることがあります。

```
chmod 400 /fullPathToYourPemFile/<yourKey>.pem
```

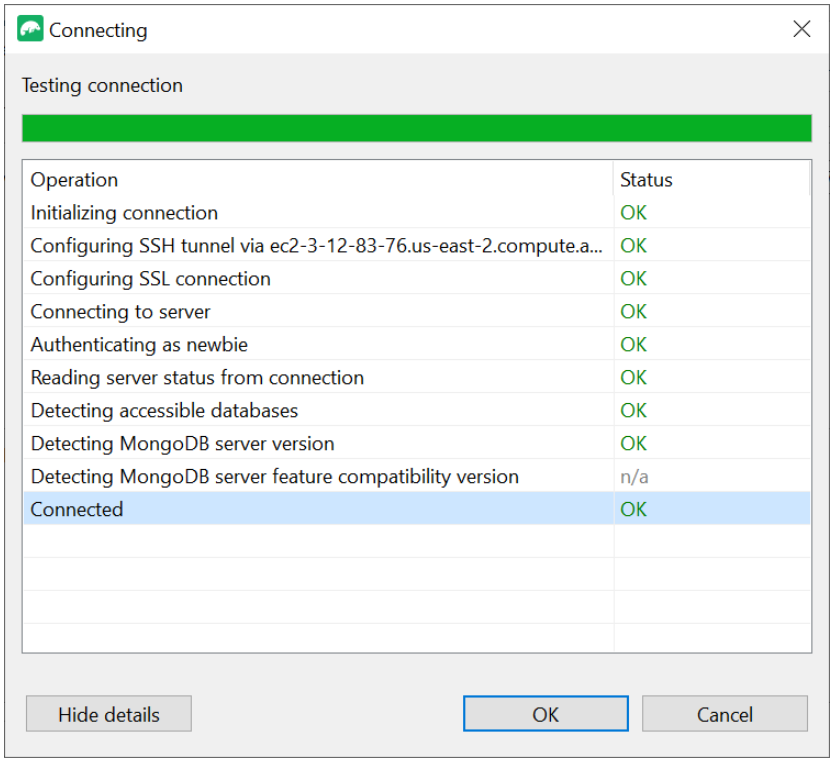


> **Note**
>
> この Amazon EC2 インスタンスは、DocumentDB クラスターと同じ Amazon VPC およびセキュリティグループ内に存在する必要があります。SSH アドレス、ユーザーネーム、プライベートキーは、Amazon EC2 マネジメントコンソール から取得できます。

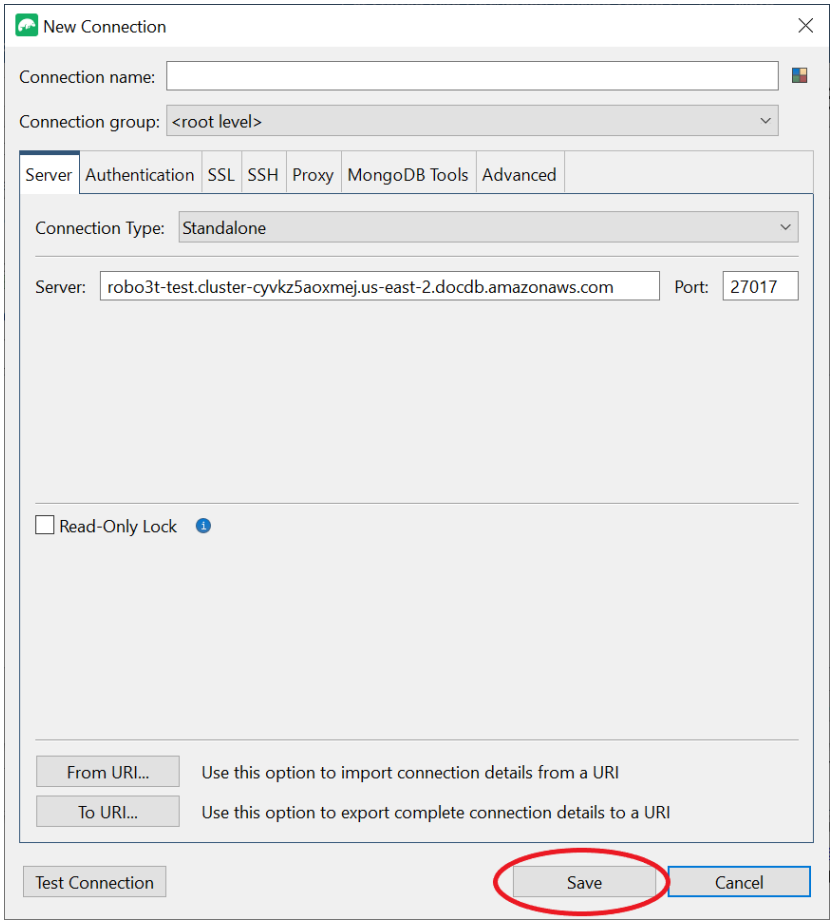
9. 次に 接続のテスト ボタンを選択して構成をテストします。



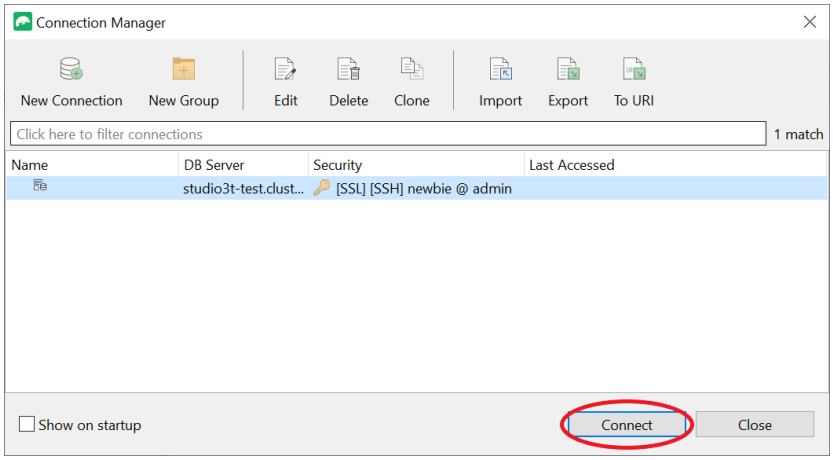
10. 診断ウィンドウには、テストが成功したことを示す緑色のバーがロードされます。次に OK を選択して、診断ウィンドウを閉じます。



11. 保存 を選択して、将来使用するために接続を保存します。



12. 次にクラスターを選択し、接続 を選択します。



お疲れ様でした。Studio 3T を介して Amazon DocumentDB クラスターに正常に接続されました。

DataGrip を使用して Amazon DocumentDB に接続する

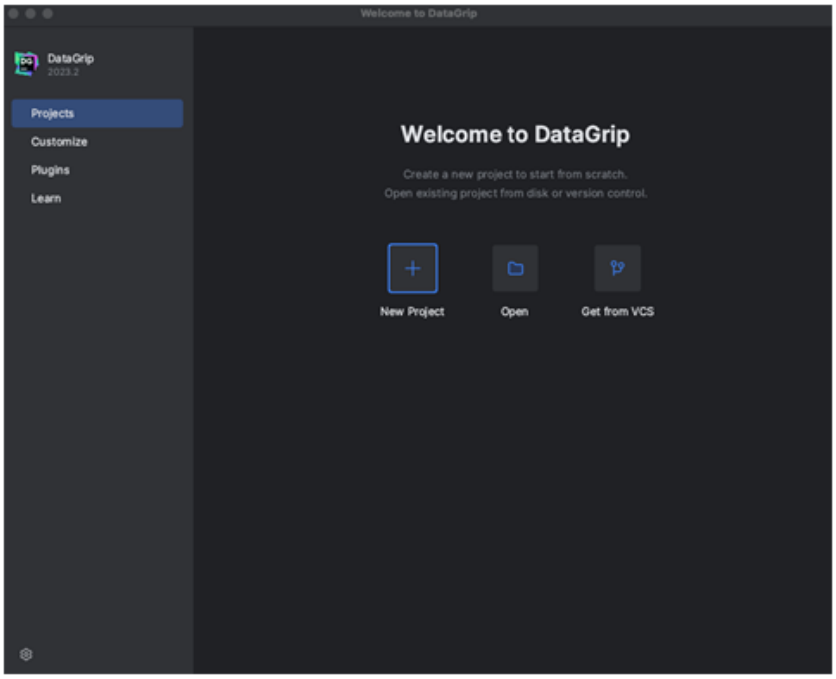
DataGrip は、Amazon DocumentDB を含むさまざまなデータベースシステムをサポートする強力な統合開発環境 (IDE) です。このセクションでは、DataGrip を使用して Amazon DocumentDB クラスターに接続する手順を説明します。これにより、グラフィカルインターフェイスを使用してデータを簡単に管理およびクエリできます。

前提条件

- DataGrip IDE がマシンにインストールされています。JetBrains からダウンロードできます。
- Amazon DocumentDB クラスターと同じ VPC で実行中の Amazon EC2 インスタンス。このインスタンスを使用して、ローカルマシンから Amazon DocumentDBCluster までの安全なトンネルを確立します。Amazon EC2 を使用した接続 方法に関する指示に従ってください。
- Amazon EC2 インスタンス、VPN 接続の代わりに、または安全な VPN を使用して AWS インフラストラクチャに既にアクセスしている場合。このオプションを使用する場合は、 を使用して Amazon DocumentDB に安全にアクセスする AWS Client VPN手順に従います。

DataGrip を使用して接続する

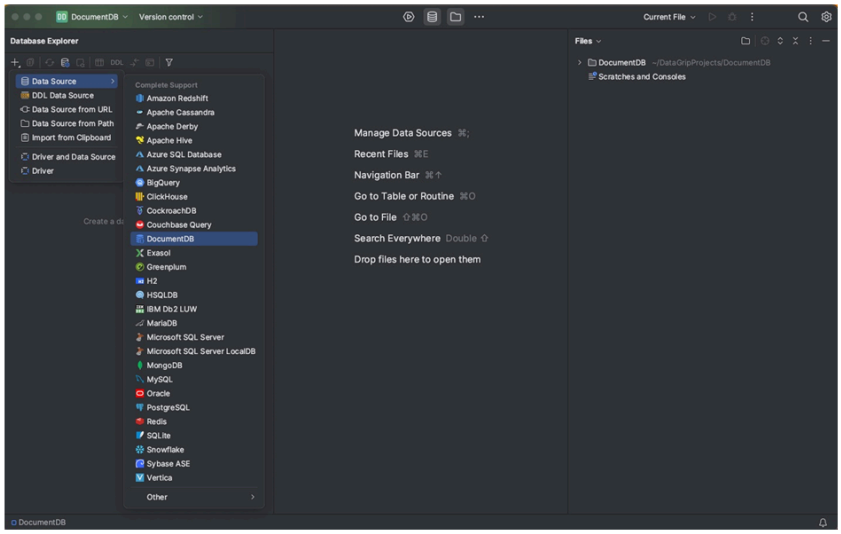
1. コンピューターで DataGrip を起動し、[新しいプロジェクト]を作成します。



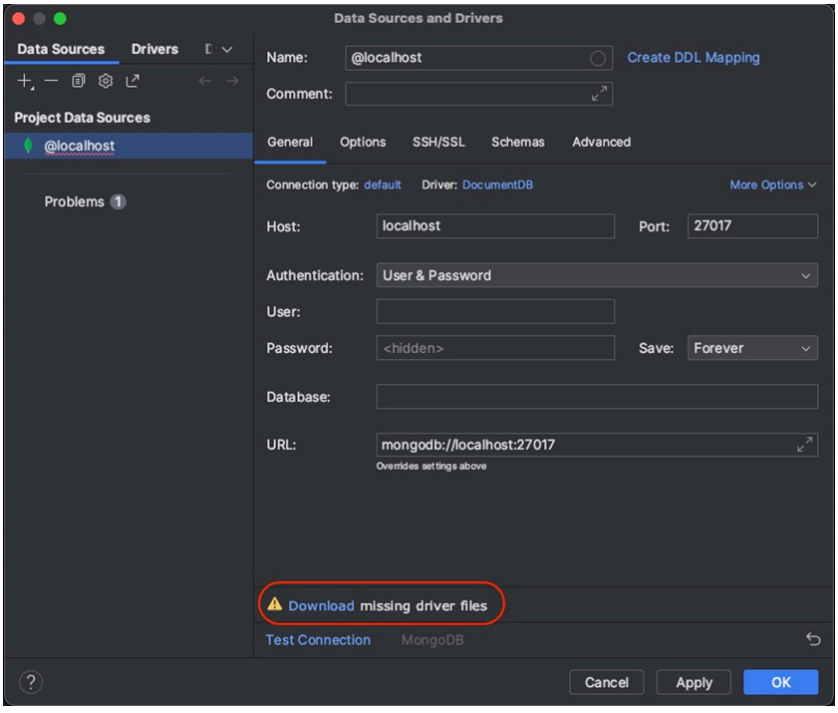
2. 以下のいずれかの方法を使用して、新しいデータソースを追加します。

a. メインメニューから [ファイル]-[新規]-[データソース] に移動し、[DocumentDB] を選択します。

b. Database Explorer で、ツールバーの新しいアイコン (+) を選択します。[データソース] に移動し、[DocumentDB] を選択します。



3. [全般] タブの [データソース] ページで、接続設定領域の下部に [見つからないドライバファイルのダウンロード] リンクがあるかどうかを確認します。データベースを操作するために必要なドライバーをダウンロードするには、このリンクを選択します。直接ダウンロードリンクについては、JetBrains JDBC ドライバーを参照してください。



4. [一般] タブで、接続の詳細を指定します。

a. [ホスト] フィールドで、Amazon DocumentDB クラスターエンドポイントを指定します。

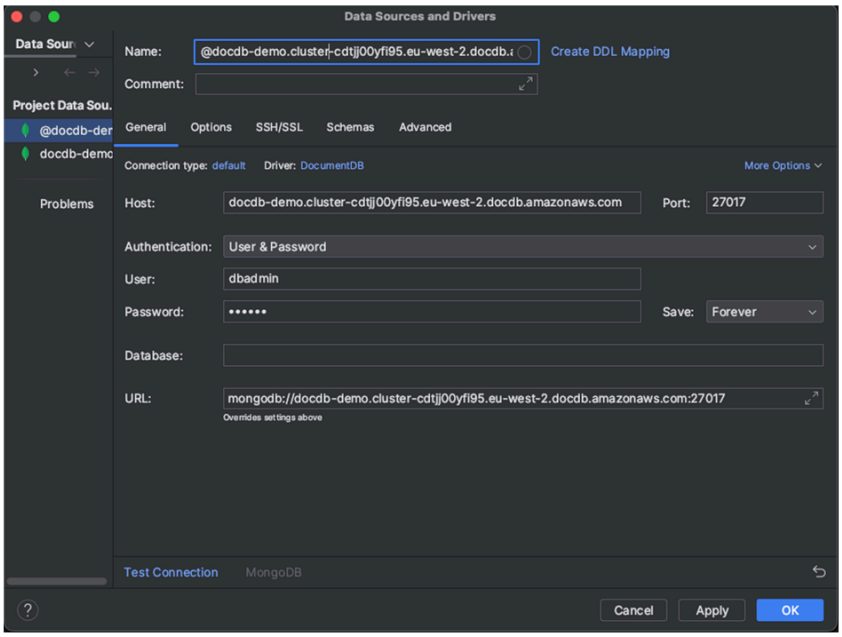
b. [ポート] は既に 27017 に設定されています。クラスターが別のポートにデプロイされている場合は変更してください。

c. [認証] で、[ユーザーとパスワード] を選択します。

d. 自分のユーザー名とパスワードを入力します。

e. [データベース] フィールドはオプションです。接続先のデータベースを指定できます。

f. URL フィールドは、前述の詳細を追加すると自動入力されます。



5. SSH/SSL タブで、SSH トンネルの使用 を有効にし、アイコンを選択して SSH 設定ダイアログを開きます。次の情報を入力します。

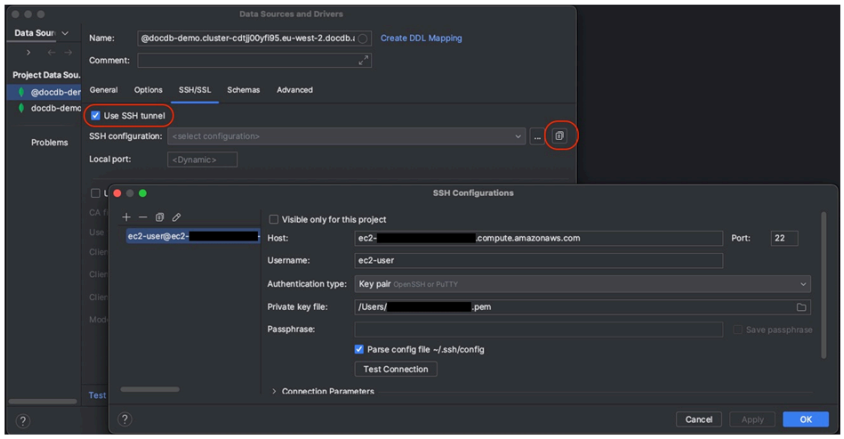
a. [ホスト] フィールドに、Amazon EC2 インスタンスのホスト名を入力します。

b. Amazon EC2 インスタンスのユーザーネームとパスワードを入力します。

c. [認証タイプ] で、[キーペア] を選択します。

d. [プライベートキーファイル] を入力します。

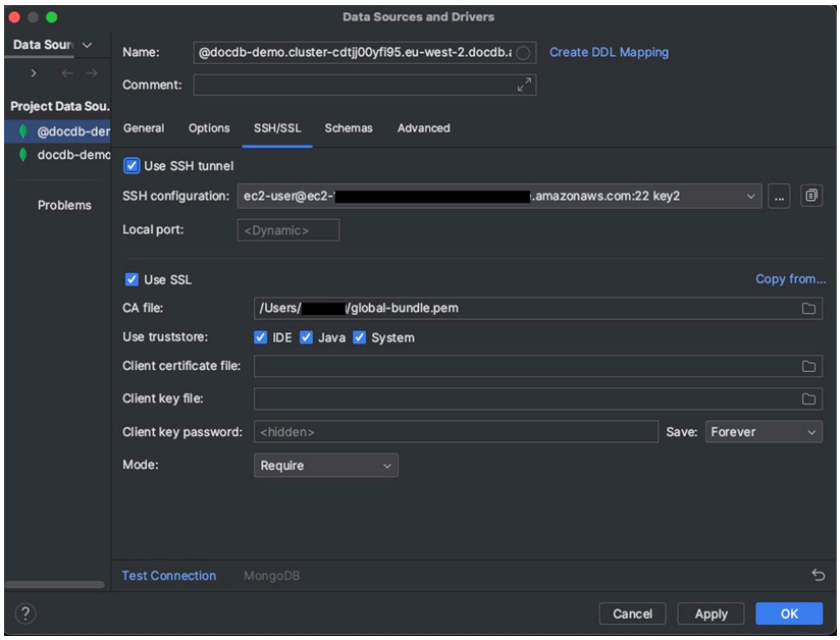
> **Note**
>
> VPN オプションを使用している場合は、SSH トンネルを設定する必要はありません。



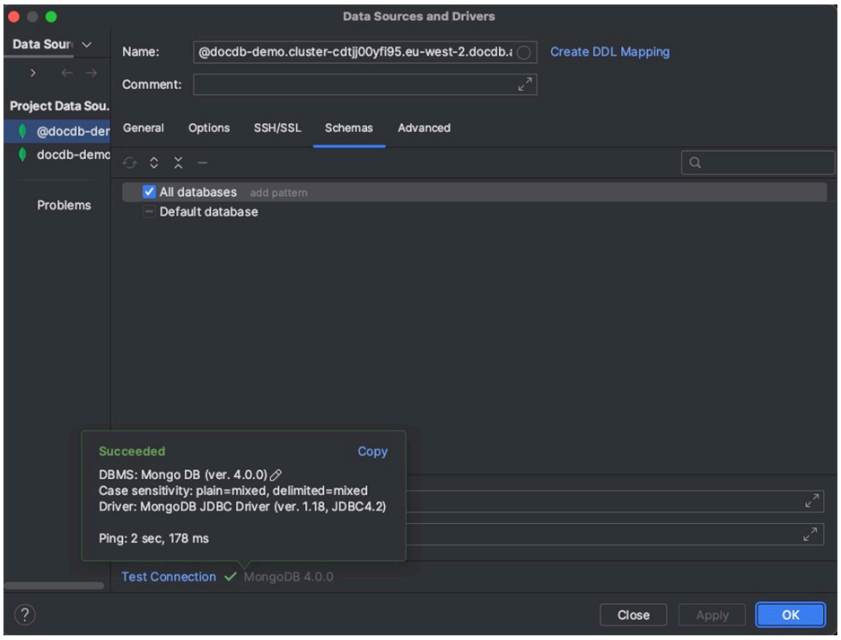
6. [SSH/SSL] タブで、[SSL を使用する] を有効にします。[CA ファイル] フィールドに、`global-bundle.pem` コンピューター上のファイルの場所を入力します。[モード] は [必須] オプションのままにしておきます。

> **Note**
>
> 証明書は次の場所からダウンロードするか、wgethttps://truststore.pki.rds.amazonaws.com/global/global-bundle.pem コマンドを使用してダウンロードできます。

> **Note**
>
> Amazon DocumentDB エラスティッククラスターに接続する場合、CA ファイルを指定する必要はありません。[SSLを使用する] オプションはオンのままにして、他のすべてのオプションはデフォルト値のままにします。



7. [スキーマ] タブの [すべてのデータベース] を選択するか、[スキーマパターン] フィールドに「*: *」というフィルターを入力します。接続をテストするには、接続のテストを選択します。



8. 接続が正常にテストされたら、OK を選択してデータソース設定を保存します。

DataGrip の機能

DataGrip には Amazon DocumentDB を使用して効率的に作業するためのさまざまな機能が用意されています。

- SQL エディター — DataGrip の SQL エディターを使用してDocumentDB コレクションに SQL のようなクエリを記述して実行します。
- ビジュアルクエリビルダー — ビジュアルクエリビルダーを使用すると、SQL コードを記述せずにグラフィカルにクエリを作成できます。
- スキーマ・マネジメント — コレクションの作成、変更、削除など、データベーススキーマを簡単に管理できます。
- データ・ビジュアライゼーション — DataGrip のさまざまな視覚化ツールを使用して、データを表示および分析します。
- データのエクスポートとインポート — DataGrip のエクスポートおよびインポート機能を使用して、Amazon DocumentDB と他のデータベース間でデータを転送します。

Amazon DocumentDB やその他のデータベースシステムの操作に関するより高度な機能やヒントについては、公式 DataGrip ドキュメントを参照してください。

Amazon EC2 を使用した接続

このセクションでは、Amazon DocumentDB クラスターと Amazon EC2 間の接続を設定し、Amazon EC2 インスタンスから Amazon DocumentDB クラスターにアクセスする方法について説明します。

EC2 接続を設定する方法は複数あります。

- EC2 インスタンスを Amazon DocumentDB データベースに自動的に接続する — EC2 コンソールの自動接続機能を使用して、EC2 インスタンスと新規または既存の Amazon DocumentDB データベース間の接続を自動で設定できます。この接続により、EC2 インスタンスと Amazon DocumentDB データベース間でやり取りされるトラフィックを確立できます。このオプションは通常、新しいセキュリティグループのテストと作成に使用されます。
- EC2 インスタンスを Amazon DocumentDB データベースに手動で接続する — 自動接続機能によって作成されるのと同じ設定を再現することにより、セキュリティグループを手動で設定して割り当て、EC2 インスタンスと Amazon DocumentDB データベース間の接続を設定できます。このオプションは、通常、高度な設定を変更したり、既存のセキュリティグループを使用するために使用されます。

前提条件

いずれのオプションを使用する場合でも、最初の Amazon DocumentDB クラスターを作成する前に、以下の操作を行う必要があります。

Amazon Web Services (AWS) アカウントを作成する

Amazon DocumentDB を使用する前に、Amazon Web Services (AWS) アカウントを持っている必要があります。 AWS アカウントは無料です。使用しているサービスとリソースに対してのみ料金をお支払いいただきます。

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://signin.aws.amazon.com/signup を開きますか?request_type=register。
2. オンラインの手順に従います。

   サインアップ手順の一環として、電話またはテキストメッセージを受け取り、電話キーパッドで検証コードを入力します。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザー が作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティベストプラクティスとして、ユーザーに管理アクセス権を割り当て、ルートユーザーアクセスが必要なタスクの実行にはルートユーザーのみを使用するようにしてください。

必要に応じて、必要な AWS Identity and Access Management (IAM) アクセス許可を設定します。

クラスター、インスタンス、クラスターパラメータグループなどの Amazon DocumentDB リソースを管理するためのアクセスには、 がリクエストの認証 AWS に使用できる認証情報が必要です。詳細については、「Amazon DocumentDB の Identity and Access Management」を参照してください。

1. の検索バーで AWS マネジメントコンソール、表示されるドロップダウンメニューで IAM **IAM**を入力して選択します。
2. IAM コンソールにアクセスしたら、ナビゲーションペインから [ユーザー] を選択します。
3. ユーザーネームを選択します。
4. [Add permissions (許可の追加)] を選択します。

5. [Attach existing policies directly (既存のポリシーを直接アタッチ)] を選択します。
6. 検索バーに `AmazonDocDBFullAccess` と入力し、検索結果に表示されたら、それを選択します。
7. [次へ: レビュー] を選択します。
8. [Add permissions] を選択します。

Amazon VPC (Amazon Virtual Private Cloud) を作成する

どの AWS リージョン にあるかによって、デフォルトの VPC が既に作成されているかどうかがわかります。デフォルトの VPC がない場合は、Amazon VPC のユーザーガイド の Amazon VPC の開始方法 のステップ 1 を完了させます。これには 5 分もかかりません。

Amazon EC2 への自動接続

トピック

- EC2 インスタンスを新しい Amazon DocumentDB データベースに自動接続する
- EC2 インスタンスを既存の Amazon DocumentDB データベースに自動的に接続する
- EC2 インスタンスとの自動接続の概要
- 接続中のコンピューティングリソースを表示する

EC2 インスタンスと新しい Amazon DocumentDB データベース間の接続を設定する前に、「EC2 インスタンスとの自動接続の概要」に記載される要件を満たしているか確認してください。接続の設定後にこれらのセキュリティグループに変更を加えると、EC2 インスタンスと Amazon DocumentDB データベース間の接続に影響を及ぼす可能性があります。

> **Note**
>
> AWS マネジメントコンソールを使用することでのみ、EC2 インスタンスと Amazon DocumentDB データベース間の接続を自動的に設定できます。 AWS CLI または Amazon DocumentDB API を使用して接続を自動的にセットアップすることはできません。

EC2 インスタンスを新しい Amazon DocumentDB データベースに自動接続する

以下のプロセスは、前提条件 トピック記載のステップが完了済みであることを前提としています。

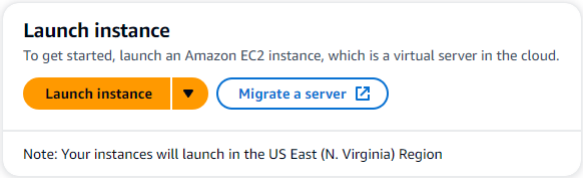
Steps

- ステップ 1: Amazon EC2 インスタンスを作成する
- ステップ 2: Amazon DocumentDB クラスターを作成する
- ステップ 3: Amazon EC2 インスタンスに接続する
- ステップ 4: MongoDB シェルをインストールする
- ステップ 5: Amazon DocumentDB TLS の管理
- ステップ 6: Amazon DocumentDB クラスターに接続する
- ステップ 7: データの挿入とクエリ
- ステップ 8: 使ってみる

ステップ 1: Amazon EC2 インスタンスを作成する

このステップでは、後で Amazon DocumentDB クラスターのプロビジョニングに使用するのと同じリージョンと Amazon VPC に Amazon EC2 インスタンスを作成します。

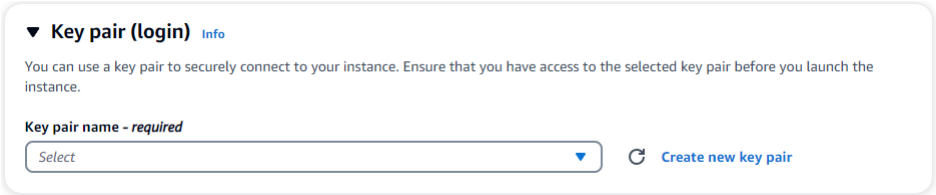
1. Amazon EC2 コンソールで、[インスタンスを起動] を選択します。



2. [名前とタグ] セクション内の [名前] フィールドに、名前または識別子を入力します。
3. [Amazon マシンイメージ (AMI)] ドロップダウンリストで、[Amazon Linux 2 AMI] を選択します。

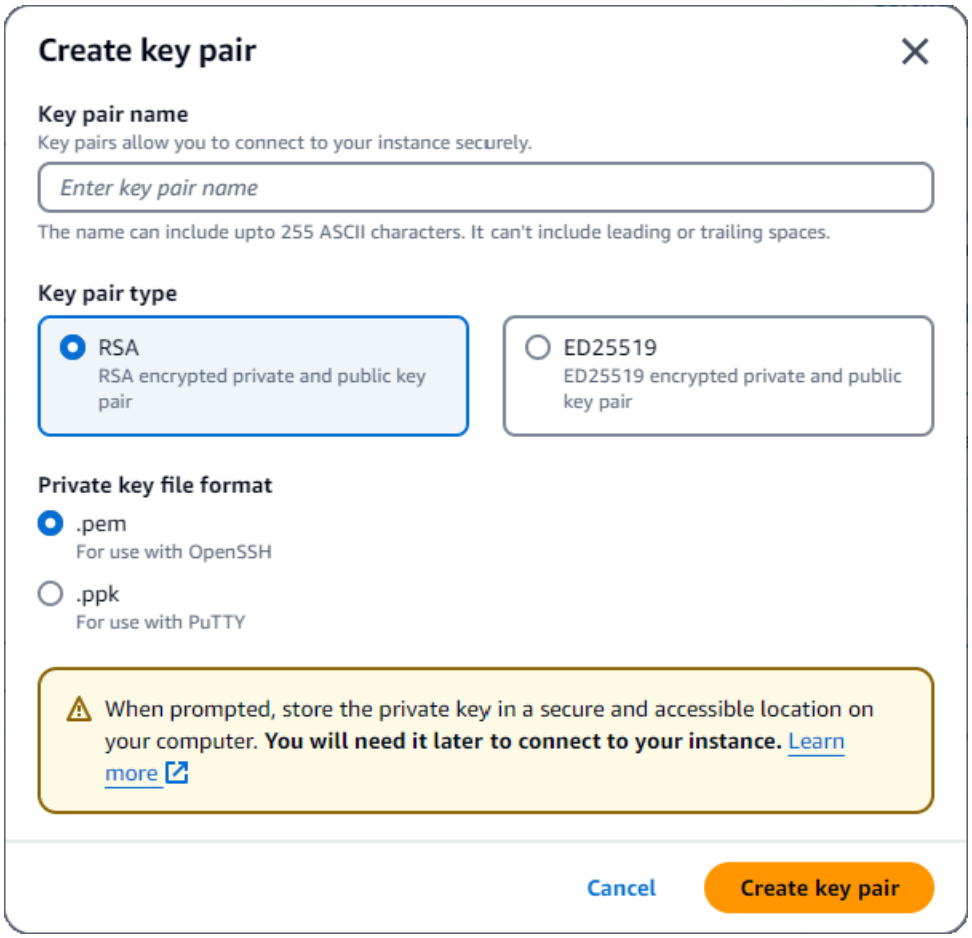


4. [インスタンスタイプ] のドロップダウンリストで、[t3.micro] を選択します。
5. [キーペア (ログイン)] セクションで、既存のキーペアの識別子を入力するか、もしくは [新しいキーペアの作成] を選択します。



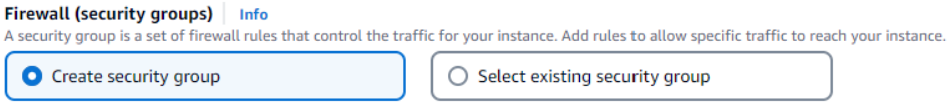
Amazon EC2 キーペアを指定する必要があります。

- Amazon EC2 のキーペアがある場合 :
  1. キーペアを選択し、リストからご自身のキーペアを選択します。
  2. Amazon EC2 インスタンスにログインするには、あからじめプライベートキーファイル (.pem もしくは.ppk ファイル) を利用可能にしておく必要があります。
- Amazon EC2 のキーペアがない場合:
  1. [新しいキーペアの作成] を選択すると、[キーペアの作成] ダイアログボックスが表示されます。
  2. [キーペア名] フィールドに名前を入力します。
  3. [キーペアタイプ] と [プライベートキーファイル形式] を選択します。
  4. [キーペアの作成] を選択してください。



> **Note**
>
> セキュリティ上の理由から、SSH とインターネット両方からの EC2 インスタンスへの接続にキーペアを使用することを強くお勧めします。

6. オプション: ネットワーク設定セクション の ファイアウォール (セキュリティグループ) で、[セキュリティグループの作成] を選択します。



[セキュリティグループの作成] を選択します (EC2 接続に適用されるすべてのトラフィック許可ルールを確認します)。

> **Note**
>
> 既存のセキュリティグループを使用する場合は、「Amazon EC2 を手動で接続する」の手順に従います。

7. [概要] セクションで、EC2 設定を確認し、問題がなければ [インスタンスを起動] を選択します。

ステップ 2: Amazon DocumentDB クラスターを作成する

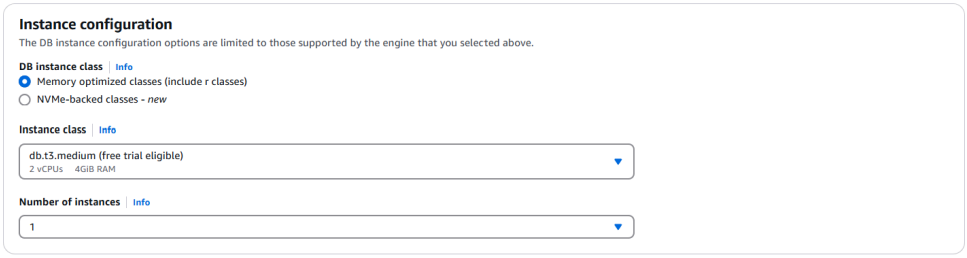
Amazon EC2 インスタンスのプロビジョニング中は、Amazon DocumentDB クラスターを作成します。

1. Amazon DocumentDB コンソールに移動し、ナビゲーションペインから クラスター を選択します。
2. [作成] を選択します。
3. [クラスタータイプ] 設定は、デフォルトの [インスタンスベースのクラスター] のままにしておきます。
4. [クラスター設定] では、一意の [クラスター名] を入力します。コンソール操作では、入力方法に関係なくクラスター名のすべての文字が小文字に変換されることにご留意ください。

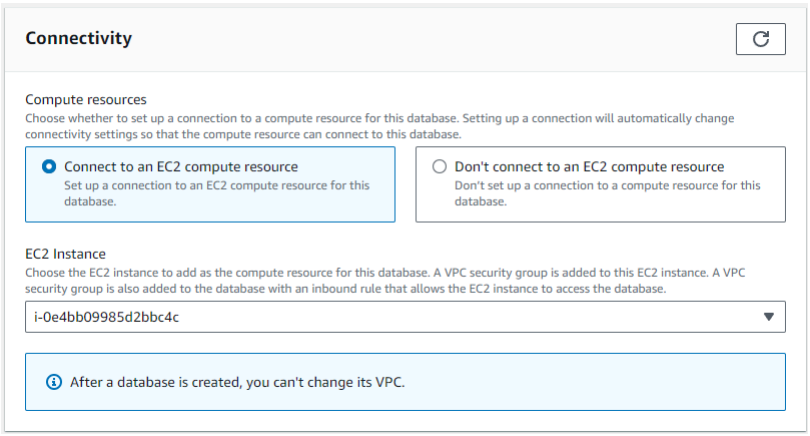
   [エンジンバージョン] はデフォルト値の 5.0.0 のままにします。

5. [クラスターストレージ設定] では、Amazon DocumentDB Standard のデフォルト設定のままにします。
6. [インスタンスの設定] で:
   - [DB インスタンスクラス] では、メモリ最適化クラス (r クラスを含む) を選択します (デフォルト)。

     もう 1 つのインスタンスオプションは NVMe-backed クラス です。詳細については NVMe-backed インスタンス を参照してください。
   - [インスタンスクラス] では、ニーズに合ったインスタンスタイプを選択します。インスタンスクラスの詳細については、「インスタンスクラスの仕様」を参照してください。
   - [インスタンスの数] では、ニーズを最も反映する数を選択します。数値が低いほど、コストが低くなり、クラスターで管理できる読み取り/書き込みボリュームが低くなることに注意してください。



7. [接続] で、[EC2 コンピューティングリソースに接続する] を選択します。これは、ステップ 1 で作成した EC2 インスタンスです。



> **Note**
>
> EC2 コンピューティングリソースに接続すると、Amazon DocumentDB クラスターへの EC2 コンピューティングリソース接続向けセキュリティグループが自動的に作成されます。クラスターの作成完了後、新しく作成したセキュリティグループを確認したい場合は、クラスターリストに移動して該当するクラスター識別子を選択します。[接続とセキュリティ] タブ内で [セキュリティグループ] に移動して、[セキュリティグループ名 (ID)] で見たいグループを見つけます。クラスターの名前と `docdb-ec2-docdb-2023-12-11-21-33-41:i-0e4bb09985d2bbc4c (sg-0238e0b0bf0f73877)` のような内容が表示されます。

8. [認証] セクションで、プライマリユーザーのユーザー名を入力し、[セルフマネージド] を選択します。パスワードを入力して確認します。

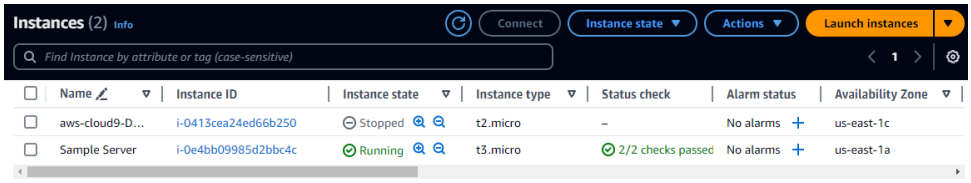
   代わりに Managed in AWS Secrets Manager を選択した場合は、詳細については[Amazon DocumentDB と によるパスワード管理 AWS Secrets Manager]「」を参照してください。

9. [クラスターを作成] を選択します。

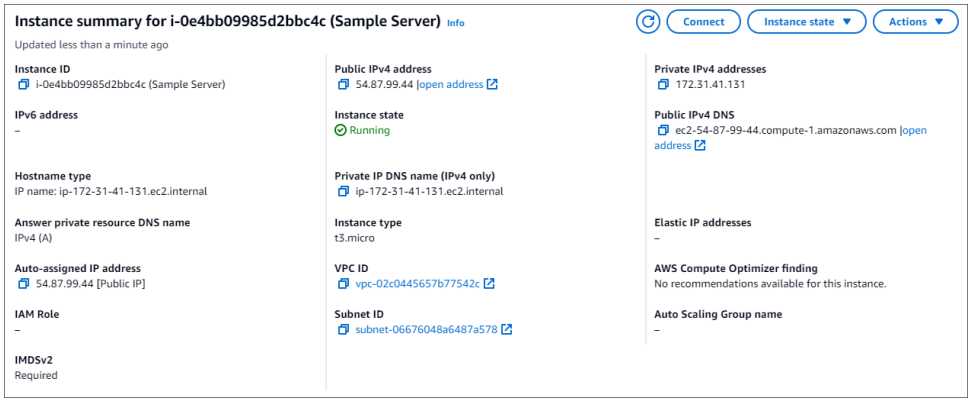
ステップ 3: Amazon EC2 インスタンスに接続する

Mongo Shell をインストールするには、まず Amazon EC2 インスタンスに接続する必要があります。Mongo シェルをインストールすると、Amazon DocumentDB クラスターに接続してクエリを実行できます。以下のステップを実行します。

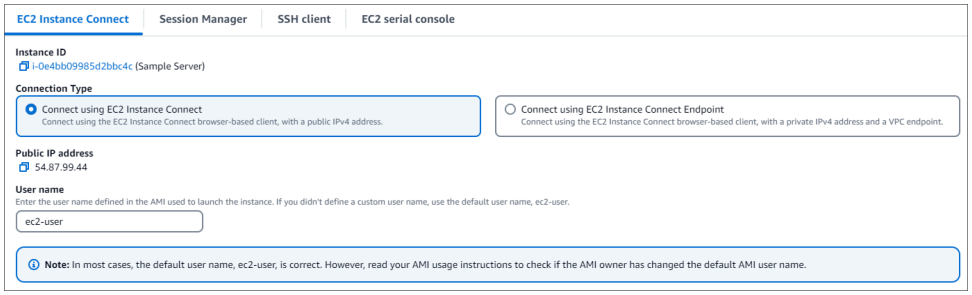
1. Amazon EC2 コンソールで、インスタンスに移動し、作成したインスタンスが実行中かどうかを確認します。実行されている場合は、インスタンス ID をクリックしてインスタンスを選択します。



2. 接続 を選択します。



3. 接続方法として、Amazon EC2 Instance Connect、セッションマネージャー、SSH クライアント、EC2 シリアルコンソールの 4 つのオプションがタブ形式で表示されます。いずれかを選択し、その指示に従わなければなりません。完了したら、[接続する] を選択します。



> **Note**
>
> このチュートリアルの開始後に IP アドレスが変更された場合、または後で環境に戻る場合は、demoEC2 セキュリティグループのインバウンドルールを更新して、新しい API アドレスからのインバウンドトラフィックを有効化します。

ステップ 4: MongoDB シェルをインストールする

これで、MongoDB シェルをインストールできます。これは、Amazon DocumentDB クラスターに接続してクエリするために使用するコマンドラインのユーティリティです。現在、MongoDB シェルには最新バージョンの mongosh と以前のバージョンの mongo シェルの 2 つのバージョンがあります。

> **Important**
>
> なお、バージョン 6.13.1 より前の Node.js ドライバーには既知の制限事項があり、現在 Amazon DocumentDB の IAM ID 認証ではサポートされていません。Node.js ドライバーを使

用する Node.js ドライバーとツール (例えば、mongosh) は、Node.js ドライバーバージョン 6.13.1 以降を使用するようにアップグレードする必要があります。

以下の手順に従って、オペレーションシステム用の MongoDB シェルをインストールします。

On Amazon Linux

Amazon Linux で MongoDB シェルをインストールするには

IAM 認証を使用しておらず、最新の MongoDB シェル (mongosh) を使用して Amazon DocumentDB クラスターに接続する場合は、次の手順に従います。

1. リポジトリファイルを作成します。作成した EC2 インスタンスのコマンドラインで、次のコマンドを入力します。

```
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. 完了したら、コマンドプロンプトで次の 2 つのコマンドオプションのいずれかを使用して mongosh をインストールします。

   オプション 1 — Amazon EC2 の設定中にデフォルトの Amazon Linux 2023 を選択した場合は、次のコマンドを入力します。

```
sudo yum install -y mongodb-mongosh-shared-openssl3
```

   オプション 2 — Amazon EC2 の設定中に Amazon Linux 2 を選択した場合は、次のコマンドを入力します。

```
sudo yum install -y mongodb-mongosh
```

IAM 認証を使用している場合は、以前のバージョンの MongoDB シェル (5.0) を使用して Amazon DocumentDB クラスターに接続する必要があり、次の手順を実行します。

1. リポジトリファイルを作成します。作成した EC2 インスタンスのコマンドラインで、次のコマンドを入力します。

```
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. 完了したら、コマンドプロンプトで次のコマンドオプションを使用して mongodb 5.0 シェルをインストールします。

```
sudo yum install -y mongodb-org-shell
```

On Ubuntu

Ubuntu に mongosh をインストールするには

1. パッケージ管理システムで使用されるパブリックキーをインポートします。

```
curl -fsSL https://pgp.mongodb.com/server-5.0.asc | sudo gpg --dearmor -o /usr/
share/keyrings/mongodb-server-5.0.gpg
```

2. Ubuntu のバージョンに適切なコマンドを使用して、MongoDB の リストファイル `mongodb-org-5.0.list` を作成します。

```
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. 次のコマンドを使用してローカルパッケージデータベースをインポートして更新します。

```
sudo apt-get update
```

4. mongosh をインストールします。

```
sudo apt-get install -y mongodb-mongosh
```

以前のバージョンの MongoDB の Ubuntu システムへのインストールについては、「Install MongoDB Community Edition on Ubuntu」を参照してください。

On other operating systems

mongo シェルを他のオペレーティングシステムにインストールするには、MongoDB ドキュメントの「 MongoDB Community Edition のインストール 」を参照してください。

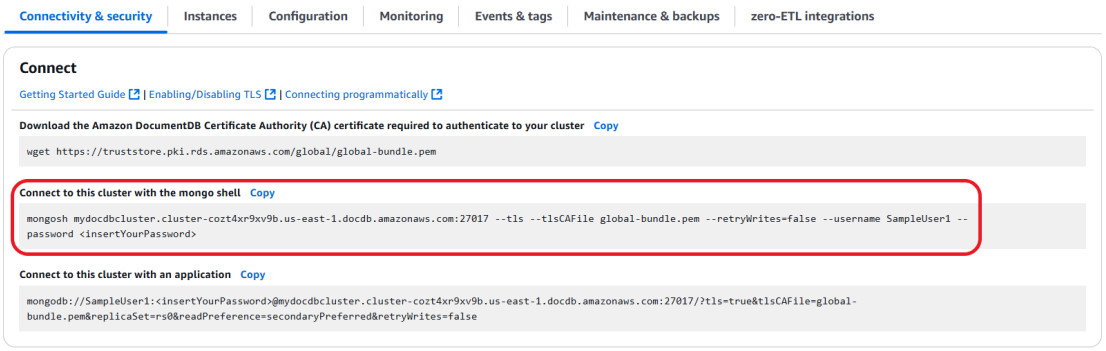
ステップ 5: Amazon DocumentDB TLS の管理

次のコード `wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem` を使用して Amazon DocumentDB の CA 証明書をダウンロードします。

> **Note**
>
> 全ての新しい Amazon DocumentDB クラスターに対して、デフォルトで Transport Layer Security (TLS) が有効になっています。詳細については、「Encrypting data in transit」を参照してください。

ステップ 6: Amazon DocumentDB クラスターに接続する

1. Amazon DocumentDB コンソールの [クラスター] で、接続したいクラスターを見つけます。そのクラスターの [クラスター識別子] をクリックして、作成済みのクラスターを選択します。
2. [接続とセキュリティ] タブの [接続する] ボックス内で、[mongo シェルを使用してこのクラスターに接続する] を見つけます。



表示された接続文字列をコピーし、ターミナルに貼り付けます。

これに以下の変更を加えます。

a. 文字列に正しいユーザー名が含まれているのを確認してください。
b. `<insertYourPassword>` は省略します。これにより、接続時に mongo シェルによりパスワードの入力を求められます。

c. オプション: IAM 認証を使用している場合、または以前のバージョンの MongoDB シェルを使用している場合は、次のように接続文字列を変更します。

```
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 をクラスターの同じ情報に置き換えます。

3. ターミナル上で Enter キーを押します。ここで、パスワードの入力を求められます。パスワードを入力します。
4. パスワードを入力し、rs0 [direct: primary] <env-name>> プロンプトが表示されたら、Amazon DocumentDB クラスターに正常に接続されています。

接続の問題を解決するには、「」を参照してくださいAmazon DocumentDB のトラブルシューティング。

ステップ 7: データの挿入とクエリ

クラスターに接続できたので、いくつかのクエリを実行して、ドキュメントデータベースの使用に慣れることができます。

1. 1 つのドキュメントを挿入するには、次のように入力します。

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

次の出力が返ります。

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

2. findOne() コマンドで書き込んだドキュメントを読み取ることができます (一つのドキュメントしか返さないため)。以下を入力します。

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

次の出力が返ります。

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

3. さらにいくつかのクエリを実行するには、ゲームプロファイルのユースケースを検討してください。まず、「profiles」というタイトルのコレクションにエントリをいくつか挿入します。以下を入力します。

```
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 }
])
```

次の出力が返ります。

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

4. find() コマンドを使用して、プロファイルコレクション内のすべてのドキュメントを返します。以下を入力します。

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

ステップ 3 で入力したデータと一致する出力を取得します。

5. フィルターを使用して 1 つのドキュメントに対して1つのクエリを使用します。以下を入力します。

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

次の出力が返ります。

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

6. それでは、プロファイルを探して、それを findAndModify コマンドで変更してみましょう。次のコードを使用して、ユーザー Matt に 10 ポイント追加します。

```
db.profiles.findAndModify({
```

```
        query: { name: "Matt", status: "active"},
        update: { $inc: { score: 10 } }
    })
```

次のような出力が得られます (Mattのスコアはまだ上がっていないことに注意してください) 。

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

7. 次のクエリを使用して、Matt のスコアが変更されたことを確認できます。

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

   次の出力が返ります。

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

ステップ 8: 使ってみる

お疲れ様でした。Amazon DocumentDB のクイックスタートガイドを正常に完了しました。

次のステップ この強力なデータベースを一般的な特徴をいくつか使って十分に活用する方法を学びましょう。

- Amazon DocumentDB クラスターとインスタンスの管理
- スケーリング
- バックアップと復元

> **Note**
>
> コストを節約するために、Amazon DocumentDB クラスターを停止してコストを削減するか、クラスターを削除することができます。デフォルトでは、30 分間非アクティブになると、 AWS Cloud9 環境は基盤となる Amazon EC2 インスタンスを停止します。

EC2 インスタンスを既存の Amazon DocumentDB データベースに自動的に接続する

次の手順では、既存の Amazon DocumentDB クラスターと既存の Amazon EC2 インスタンスを利用できることを前提としています。

Amazon DocumentDB クラスターにアクセスし、Amazon EC2 接続をセットアップする

1. Amazon DocumentDB クラスターにアクセスします。

   a. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。

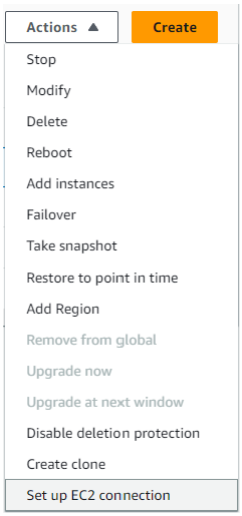
   b. ナビゲーションペインで クラスター を選択します。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

   c. クラスター名の左側にあるボタンを選択して、アクセスしたいクラスターを指定します。

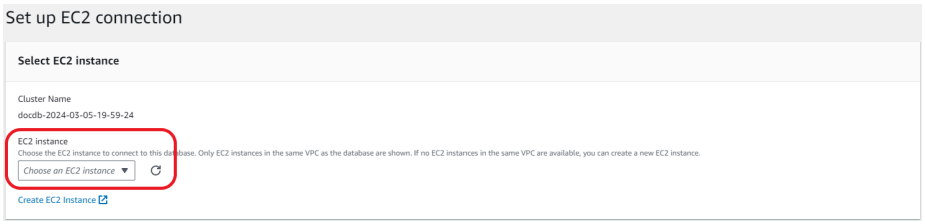
2. Amazon EC2 接続を設定します。

   a. [アクション] をクリックし、[EC2 接続の設定] をクリックします。

   [EC2 接続の設定] ダイアログが開きます。

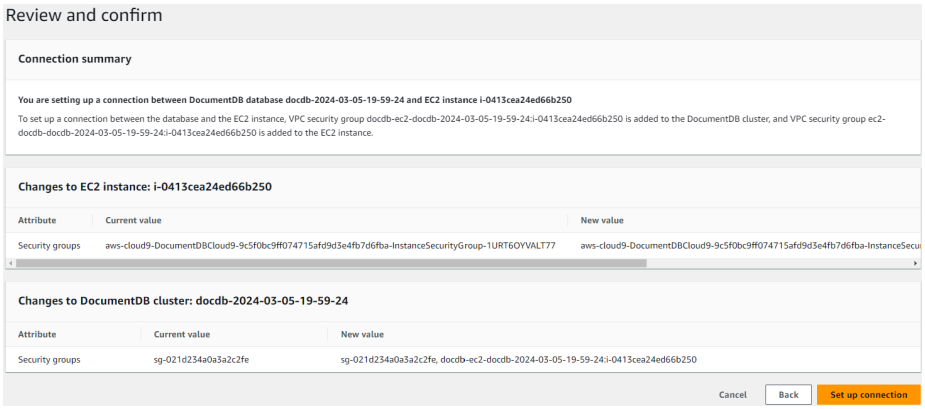
   b. [EC2 インスタンス] フィールドで、クラスターに接続する EC2 インスタンスを選択します。



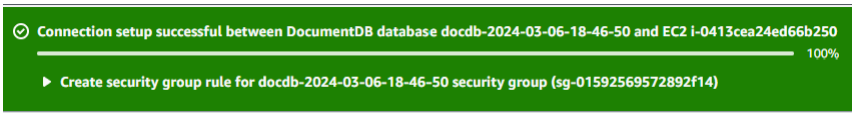
c. [続行] をクリックしてください。

[確認と確定] ダイアログページが表示されます。

d. 変更内容が正しいことを確認します。続いて、[接続の設定] を選択します。



正常に接続が設定されると、以下の検証結果が表示されます。



EC2 インスタンスとの自動接続の概要

EC2 インスタンスと Amazon DocumentDB データベース間の接続を設定すると、Amazon DocumentDB ではその EC2 インスタンスと Amazon DocumentDB データベース向けの VPC セキュリティグループを自動的に設定します。

EC2 インスタンスと Amazon DocumentDB データベースを接続するための要件は次のとおりです。

- EC2 インスタンスは、Amazon DocumentDB データベースと同じ VPC 内に存在する必要があります。

  同じ VPC に EC2 インスタンスが存在しない場合、コンソールには EC2 インスタンス作成用のリンクが表示されます。

- 接続を設定するユーザーには、以下の Amazon EC2 オペレーションを実行するアクセス許可が必要です。
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeSecurityGroups`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

DB インスタンスと EC2 インスタンスが異なるアベイラビリティーゾーンにある場合、アベイラビリティーゾーン間のコストが発生する可能性があります。

EC2 インスタンスへの接続を設定すると、Amazon DocumentDB は、次の表に示すように、その Amazon DocumentDB データベースと EC2 インスタンスに関連付けられているセキュリティグループの現在の設定に従って動作します。

現在の Amazon DocumentDB セキュリティグループ設定	現在の EC2 セキュリティグループの設定	Amazon DocumentDB の動作
パターン `DocumentDB-ec2-n` に一致する名前をもつ Amazon DocumentDB データベースに関連付けられたセキュリティグループが 1 つ以上存在します。パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、EC2 インスタンスの VPC セキュリティグループをソースとするインバウンドルールが 1 つのみ存在します。	パターン `DocumentDB-ec2-n` (n は数字) に一致する名前の EC2 インスタンスに関連付けられたセキュリティグループが 1 つ以上存在します。パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、Amazon DocumentDB データベースの VPC セキュリティグループをソースとするアウトバウンドルールが 1 つのみ存在します。	Amazon DocumentDB はアクションを実行しません。EC2 インスタンスと Amazon DocumentDB データベース間の接続は、既に自動で設定されています。EC2 インスタンスと Amazon DocumentDB データベース間には既に接続が存在するため、セキュリティグループの変更は行われません。

現在の Amazon DocumentDB セキュリティグループ設定	現在の EC2 セキュリティグループの設定	Amazon DocumentDB の動作
次の条件のいずれかが適用されます。 • パターン DocumentDB-ec2-n に一致する名前をもつ Amazon DocumentDB データベースに関連付けられたセキュリティグループは存在しません。 • パターン DocumentDB-ec2-n に一致する名前をもつ Amazon DocumentDB に関連付けられたセキュリティグループが 1 つ以上存在します。ただし、Amazon DocumentDB は、これらのセキュリティグループのいずれも、EC2 インスタンスとの接続には使用できません。Amazon DocumentDB は、EC2 インスタンスの VPC セキュリティグループをソースとするインバウンドルールが 1 つも存在しないセキュリティグループを使用できません。また、Amazon DocumentDB は、変更が加えられたセキュリティグループを使用できません。変更の例としては、ルールの追加や、既存ルールのポート変更などがあります。	次の条件のいずれかが適用されます。 • EC2 インスタンスに関連付けされた、パターン ec2-DocumentDB-n に一致する名前のセキュリティグループは存在しません。 • EC2 インスタンスに関連付けられた、パターン ec2-DocumentDB-n に一致する名前のセキュリティグループが 1 つまたは複数存在します。ただし、Amazon DocumentDB は、これらのセキュリティグループのいずれも、Amazon DocumentDB データベースとの接続には使用できません。Amazon DocumentDB は、Amazon DocumentDB データベースの VPC セキュリティグループをソースとするアウトバウンドルールが 1 つもないセキュリティグループを使用できません。また、Amazon DocumentDB は、変更が加えられたセキュリティグループを使用できません。	Amazon DocumentDB アクション: 新しいセキュリティグループを作成する

現在の Amazon DocumentDB セキュリティグループ設定	現在の EC2 セキュリティグループの設定	Amazon DocumentDB の動作
パターン `DocumentDB-ec2-n` に一致する名前をもつ Amazon DocumentDB データベースに関連付けられたセキュリティグループが 1 つ以上存在します。パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、EC2 インスタンスの VPC セキュリティグループをソースとするインバウンドルールが 1 つのみ存在します。	EC2 インスタンスに関連付けられた、パターン `ec2-DocumentDB-n` に一致する名前のセキュリティグループが 1 つまたは複数存在します。ただし、Amazon DocumentDB は、これらのセキュリティグループのいずれも、Amazon DocumentDB データベースとの接続には使用できません。Amazon DocumentDB は、Amazon DocumentDB データベースの VPC セキュリティグループをソースとするアウトバウンドルールが 1 つもないセキュリティグループを使用できません。また、Amazon DocumentDB は、変更が加えられたセキュリティグループを使用できません。	Amazon DocumentDB アクション: 新しいセキュリティグループを作成する

現在の Amazon DocumentDB セキュリティグループ設定	現在の EC2 セキュリティグループの設定	Amazon DocumentDB の動作
パターン `DocumentDB-ec2-n` に一致する名前をもつ Amazon DocumentDB データベースに関連付けられたセキュリティグループが 1 つ以上存在します。パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、EC2 インスタンスの VPC セキュリティグループをソースとするインバウンドルールが 1 つのみ存在します。	接続に有効な EC2 セキュリティグループは存在しますが、EC2 インスタンスに関連付けられていません。このセキュリティグループには、パターン `DocumentDB-ec2-n` に一致する名前が付いています。このセキュリティグループには変更が加えられていません。このセキュリティグループには、Amazon DocumentDB データベースの VPC セキュリティグループをソースとするアウトバウンドルールが 1 つのみ存在します。	Amazon DocumentDB アクション: EC2 セキュリティグループを関連付ける

現在の Amazon DocumentDB セキュリティグループ設定	現在の EC2 セキュリティグループの設定	Amazon DocumentDB の動作
次の条件のいずれかが適用されます。 • パターン DocumentDB-ec2-n に一致する名前をもつ Amazon DocumentDB データベースに関連付けられたセキュリティグループは存在しません。 • パターン DocumentDB-ec2-n に一致する名前をもつ Amazon DocumentDB データベースに関連付けられたセキュリティグループが 1 つ以上存在します。ただし、Amazon DocumentDB は、これらのセキュリティグループのいずれも、EC2 インスタンスとの接続には使用できません。Amazon DocumentDB は、EC2 インスタンスの VPC セキュリティグループをソースとするインバウンドルールが 1 つも存在しないセキュリティグループを使用できません。また、Amazon DocumentDB は、変更が加えられたセキュリティグループを使用できません。	EC2 インスタンスに関連付けられた、パターン DocumentDB-ec2-n に一致する名前のセキュリティグループが 1 つまたは複数存在します。パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、Amazon DocumentDB データベースの VPC セキュリティグループをソースとするアウトバウンドルールが 1 つのみ存在します。	Amazon DocumentDB アクション: 新しいセキュリティグループを作成する

Amazon DocumentDB アクション: 新しいセキュリティグループを作成する

Amazon DocumentDB は以下のアクションを実行します。

- パターン `DocumentDB-ec2-n` に一致する新しいセキュリティグループを作成します。このセキュリティグループには、EC2 インスタンスの VPC セキュリティグループをソースとするインバウンドルールが存在します。このセキュリティグループは Amazon DocumentDB データベースに関連付けられており、EC2 インスタンスが Amazon DocumentDB データベースにアクセスするのを許可します。
- パターン `ec2-DocumentDB-n` に一致する新しいセキュリティグループを作成します。このセキュリティグループには、Amazon DocumentDB データベースの VPC セキュリティグループをソースとするアウトバウンドルールが存在します。このセキュリティグループには EC2 インスタンスに関連付けられており、EC2 インスタンスが Amazon DocumentDB データベースにトラフィックを送信するのを許可します。

Amazon DocumentDB アクション: EC2 セキュリティグループを関連付ける

Amazon DocumentDB は、有効な既存の EC2 セキュリティグループを EC2 インスタンスに関連付けます。このセキュリティグループは、EC2 インスタンスが Amazon DocumentDB データベース にトラフィックを送信するのを許可します。

接続中のコンピューティングリソースを表示する

を使用して AWS マネジメントコンソール 、Amazon DocumentDB データベースに接続されているコンピューティングリソースを表示できます。表示されるリソースには、自動的に設定されたコンピューティングリソース接続が含まれます。コンピューティングリソースとの接続は、次の方法で自動的に設定できます。

- データベースを作成するときに、コンピューティングリソースを選択できます。詳細については、「Amazon DocumentDB クラスターの作成」および「マルチ AZ DB クラスターの作成」を参照してください。
- 既存のデータベースとコンピューティングリソース間の接続を設定できます。詳細については、「Amazon EC2 への自動接続」を参照してください。

コンピューティングリソースリストには、手動でデータベースに接続されたものは含まれていません。例えば、データベースに関連付けられた VPC セキュリティグループにルールを追加することで、コンピューティングリソースがデータベースに手動でアクセスできるようになります。

コンピューティングリソースをリスト化するには、次の条件を満たしている必要があります。

- コンピューティングリソースに関連付けられているセキュリティグループの名前がパターン `ec2-DocumentDB-n` (n は数字) と一致する。
- コンピューティングリソースに関連付けられたセキュリティグループには、Amazon DocumentDB データベースが使用するポート向けに設定されたポート範囲を持つアウトバウンドルールがある。
- コンピューティングリソースに関連付けられたセキュリティグループには、Amazon DocumentDB データベースに関連付けられたセキュリティグループをソースとするアウトバウンドルールがある。
- Amazon DocumentDB データベースに関連付けられたセキュリティグループの名前が、パターン `DocumentDB-ec2-n` (n は数字) に一致する。
- Amazon DocumentDB データベースに関連付けられたセキュリティグループには、Amazon DocumentDB データベースが使用するポート向けに設定されたポート範囲を持つインバウンドルールがある。
- Amazon DocumentDB データベースに関連付けられたセキュリティグループには、コンピューティングリソースに関連付けられたセキュリティグループをソースとするインバウンドルールがある。

Amazon DocumentDB データベースに接続されているコンピューティングリソースを確認するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[データベース] を選択し、確認したい Amazon DocumentDB データベースの名前を選択します。
3. [接続とセキュリティ] タブ内の [接続されたコンピューティングリソース] で、接続されているコンピューティングリソースを確認します。

Amazon EC2 を手動で接続する

トピック

- ステップ 1: Amazon EC2 インスタンスを作成する
- ステップ 2: セキュリティグループを作成する
- ステップ 3: Amazon DocumentDB クラスターを作成する
- ステップ 4: Amazon EC2 インスタンスに接続します
- ステップ 5: MongoDB シェルをインストールする
- ステップ 6: Amazon DocumentDB TLS の管理

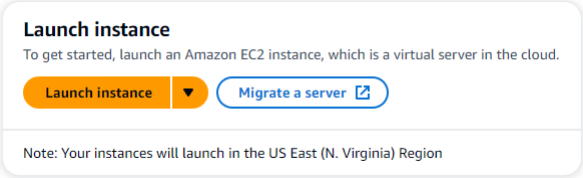
- ステップ 7: Amazon DocumentDB クラスターに接続する
- ステップ 8: データの挿入とクエリ
- ステップ 9: 環境情報の確認

以下の手順は、「前提条件」トピックに記載されるステップが完了済みであることを前提としています。

ステップ 1: Amazon EC2 インスタンスを作成する

このステップでは、後で Amazon DocumentDB クラスターのプロビジョニングに使用するのと同じリージョンと Amazon VPC に Amazon EC2 インスタンスを作成します。

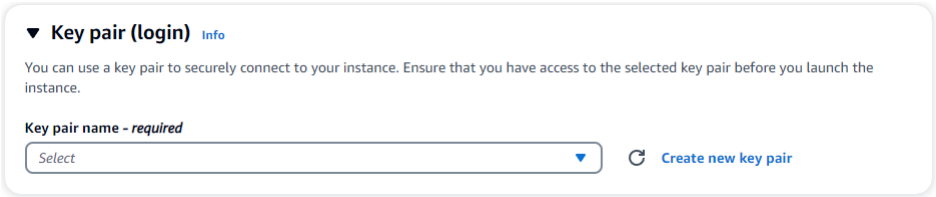
1. Amazon EC2 コンソールで、[インスタンスを起動] を選択します。



2. [名前とタグ] セクション内の [名前] フィールドに、名前または識別子を入力します。
3. [Amazon マシンイメージ (AMI)] ドロップダウンリストで、[Amazon Linux 2 AMI] を選択します。



4. [インスタンスタイプ] のドロップダウンリストで、[t3.micro] を選択します。
5. [キーペア (ログイン)] セクションで、既存のキーペアの識別子を入力するか、もしくは [新しいキーペアの作成] を選択します。



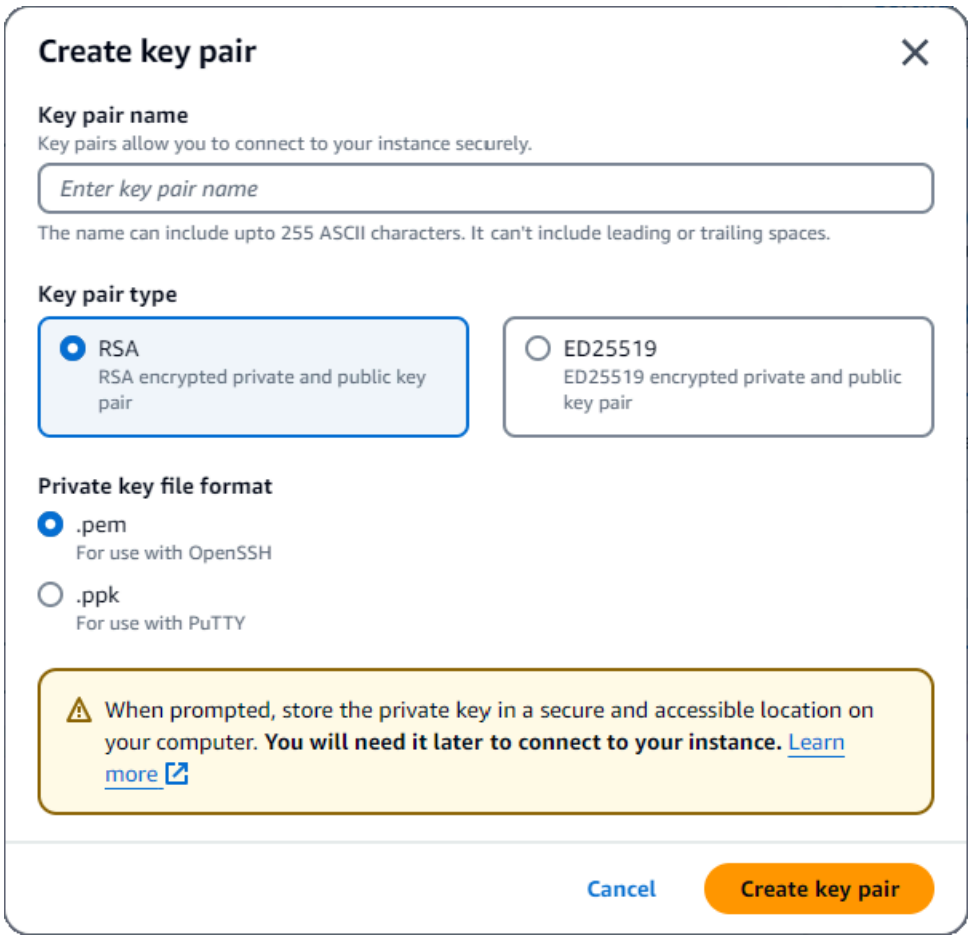
Amazon EC2 キーペアを指定する必要があります。

- Amazon EC2 のキーペアがある場合 :
  1. キーペアを選択し、リストからご自身のキーペアを選択します。

2. Amazon EC2 インスタンスにログインするには、あからじめプライベートキーファイル (.pem もしくは.ppk ファイル) を利用可能にしておく必要があります。

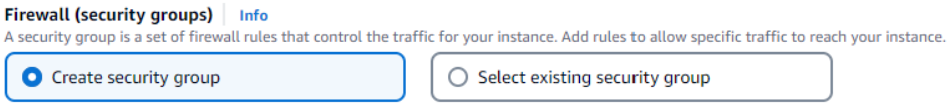
- Amazon EC2 のキーペアがない場合:

1. [新しいキーペアの作成] を選択すると、[キーペアの作成] ダイアログボックスが表示されます。
2. [キーペア名] フィールドに名前を入力します。
3. [キーペアタイプ] と [プライベートキーファイル形式] を選択します。
4. [キーペアの作成] を選択してください。



> **Note**
>
> セキュリティ上の理由から、SSH とインターネット両方からの EC2 インスタンスへの接続にキーペアを使用することを強くお勧めします。

6. [ネットワーク設定] セクションの [ファイアウォール (セキュリティグループ)] で、[セキュリティグループの作成] または [既存のセキュリティグループを選択する] を選択します。



既存のセキュリティグループを選択する場合は、[共通セキュリティグループ] ドロップダウンリストから選択します。

新しいセキュリティグループの作成を選択する場合は、以下を実行します。

a. EC2 への接続に適用されるすべてのトラフィック許可ルールを確認します。

b. [IP] フィールドで、[マイ IP] または [カスタム] を選択して、CIDR ブロックのリスト、プレフィックスリスト、もしくはセキュリティグループのいずれかを選択します。ここで [Anywhere] を選択するとどの IP アドレスからでも EC2 インスタンス へのアクセスが可能になるため、EC2 インスタンスが分離されたネットワーク上にある場合を除き、[Anywhere] を選択することはお勧めしません。

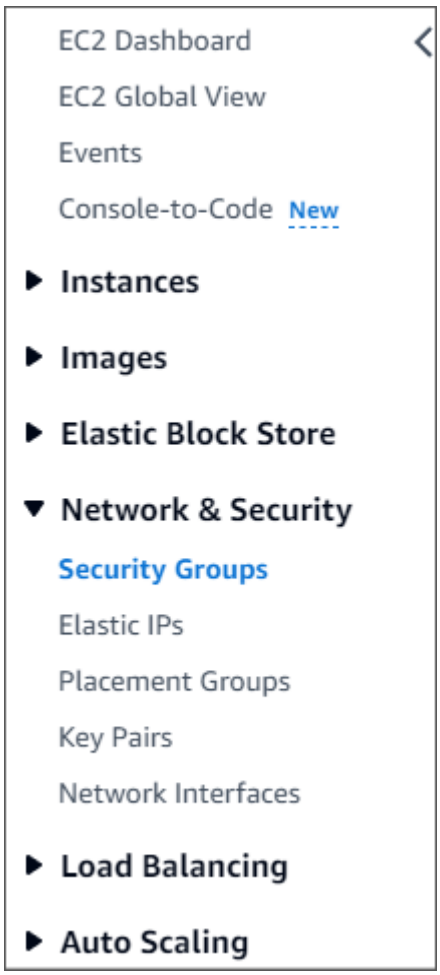


7. [概要] セクションで、EC2 設定を確認し、問題がなければ [インスタンスを起動] を選択します。

ステップ 2: セキュリティグループを作成する

次に、デフォルトの Amazon VPC に新しいセキュリティグループを作成します。セキュリティグループ demoDocDB では、Amazon EC2 インスタンスからポート 27017 (Amazon DocumentDB のデフォルトポート) で Amazon DocumentDB クラスターに接続できます。

1. Amazon EC2 マネジメントコンソール で、[ネットワークとセキュリティ] の下にある [セキュリティグループ] を選択します。



2. [セキュリティグループの作成] を選択してください。



3. [基本的な詳細] セクションで、次の操作を行います。

   a. [Security group name (セキュリティグループ名)] に「 demoDocDB 」と入力します。

   b. [説明] に説明を入力します。

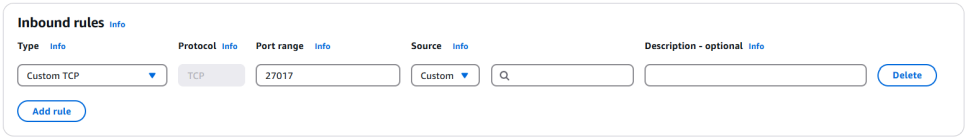
   c. VPC については、デフォルトの VPC の使用を受け入れます。

4. [インバウンドルール] セクションで、[ルールの追加] を選択します。

   a. [タイプ] で [カスタム TCP ルール] を選択します (デフォルト)。

   b. [ポート範囲] に 27017 と入力します。

   c. [ソース] で [カスタム] を選択します。その横にあるフィールドで、先ほどステップ 1 で作成したセキュリティグループを検索します。 のソース名を自動入力するには、Amazon EC2 コンソールでブラウザを更新することが必要な場合があります。



5. 他のすべてのデフォルトを受け入れ、[セキュリティグループの作成] を選択します。



ステップ 3: Amazon DocumentDB クラスターを作成する

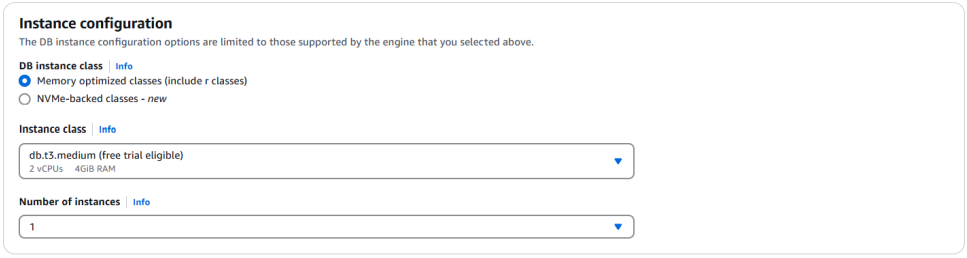
Amazon EC2 インスタンスのプロビジョニング中は、Amazon DocumentDB クラスターを作成します。

1. Amazon DocumentDB コンソールに移動し、ナビゲーションペインから クラスター を選択します。
2. [作成] を選択します。
3. [クラスタータイプ] 設定は、デフォルトの [インスタンスベースのクラスター] のままにしておきます。
4. [クラスター設定] では、一意の [クラスター名] を入力します。コンソール操作では、入力方法に関係なくクラスター名のすべての文字が小文字に変換されることにご留意ください。

   [エンジンバージョン] はデフォルト値の 5.0.0 のままにします。
5. [クラスターストレージ設定] では、Amazon DocumentDB Standard のデフォルト設定のままにします。
6. [インスタンスの設定] で:

   - [DB インスタンスクラス] では、メモリ最適化クラス (r クラスを含む) を選択します (デフォルト)。

     もう 1 つのインスタンスオプションは NVMe-backed クラス です。詳細については NVMe-backed インスタンス を参照してください。
   - [インスタンスクラス] では、ニーズに合ったインスタンスタイプを選択します。インスタンスクラスの詳細については、「インスタンスクラスの仕様」を参照してください。
   - [インスタンスの数] では、ニーズを最も反映する数を選択します。数値が低いほど、コストが低くなり、クラスターで管理できる読み取り/書き込みボリュームが低くなることに注意してください。



7. [接続] は、デフォルト設定の [EC2 コンピューティングリソースに接続しない] のままにしておきます。

> **Note**
>
> EC2 コンピューティングリソースに接続すると、お使いのクラスターへの接続向けセキュリティグループが自動的に作成されます。前のステップでセキュリティグループを既に手動で作成しているため、追加のセキュリティグループのセットが作成されないように、[EC2 コンピューティングリソースに接続しない] を選択しておく必要があります。

8. [認証] セクションで、プライマリユーザーのユーザー名を入力し、[セルフマネージド] を選択します。パスワードを入力して確認します。

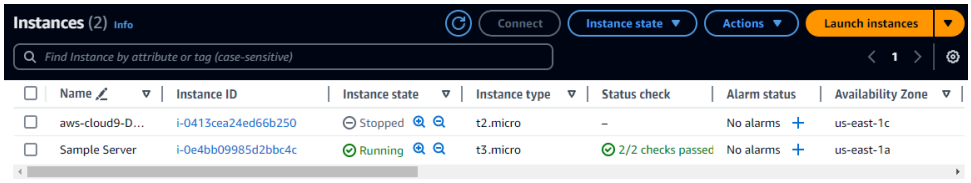
   代わりに Managed in AWS Secrets Manager を選択した場合は、詳細については[Amazon DocumentDB と によるパスワード管理 AWS Secrets Manager]「」を参照してください。

9. [クラスターを作成] を選択します。

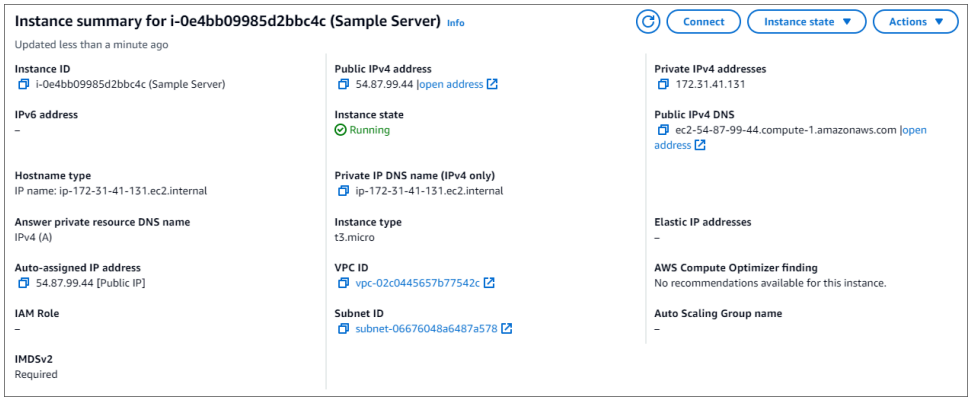
ステップ 4: Amazon EC2 インスタンスに接続します

Amazon EC2 インスタンスに接続すると、MongoDB シェルをインストールできるようになります。Mongo シェルをインストールすると、Amazon DocumentDB クラスターに接続してクエリを実行できます。以下のステップを実行します。

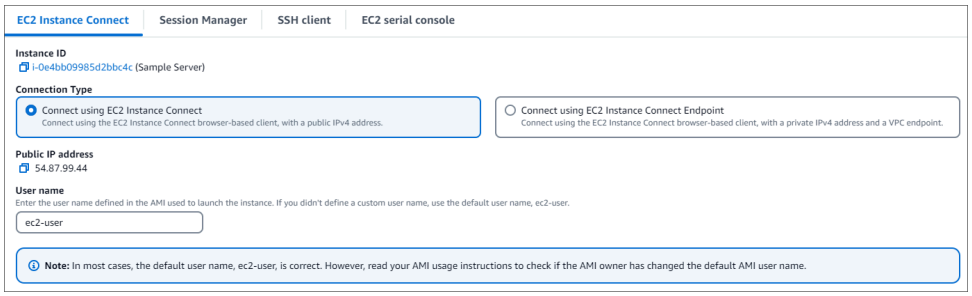
1. Amazon EC2 コンソールで、インスタンスに移動し、作成したインスタンスが実行中かどうかを確認します。実行されている場合は、インスタンス ID をクリックしてインスタンスを選択します。



2. 接続 を選択します。



3. 接続方法として、Amazon EC2 Instance Connect、セッションマネージャー、SSH クライアント、EC2 シリアルコンソールの 4 つのオプションがタブ形式で表示されます。いずれかを選択し、その指示に従わなければなりません。完了したら、[接続する] を選択します。



> **Note**
>
> このチュートリアルの開始後に IP アドレスが変更された場合、または後で環境に戻る場合は、demoEC2 セキュリティグループのインバウンドルールを更新して、新しい API アドレスからのインバウンドトラフィックを有効化します。

ステップ 5: MongoDB シェルをインストールする

これで、MongoDB シェルをインストールできます。これは、Amazon DocumentDB クラスターに接続してクエリするために使用するコマンドラインのユーティリティです。現在、MongoDB シェル

には最新バージョンの mongosh と以前のバージョンの mongo シェルの 2 つのバージョンがあります。

> **Important**
>
> なお、バージョン 6.13.1 より前の Node.js ドライバーには既知の制限事項があり、現在 Amazon DocumentDB の IAM ID 認証ではサポートされていません。Node.js ドライバーを使用する Node.js ドライバーとツール (例えば、mongosh) は、Node.js ドライバーバージョン 6.13.1 以降を使用するようにアップグレードする必要があります。

以下の手順に従って、オペレーションシステム用の MongoDB シェルをインストールします。

On Amazon Linux

Amazon Linux で MongoDB シェルをインストールするには

IAM を使用しておらず、最新の MongoDB シェル (mongosh) を使用して Amazon DocumentDB クラスターに接続する場合は、次の手順に従います。

1. リポジトリファイルを作成します。作成した EC2 インスタンスのコマンドラインで、次のコマンドを入力します。

```
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. 完了したら、コマンドプロンプトで次の 2 つのコマンドオプションのいずれかを使用して mongosh をインストールします。

   オプション 1 — Amazon EC2 の設定中にデフォルトの Amazon Linux 2023 を選択した場合は、次のコマンドを入力します。

```
sudo yum install -y mongodb-mongosh-shared-openssl3
```

   オプション 2 — Amazon EC2 の設定中に Amazon Linux 2 を選択した場合は、次のコマンドを入力します。

```
sudo yum install -y mongodb-mongosh
```

IAM を使用している場合は、以前のバージョンの MongoDB シェル (5.0) を使用して Amazon DocumentDB クラスターに接続する必要があり、次の手順を実行します。

1. リポジトリファイルを作成します。作成した EC2 インスタンスのコマンドラインで、次のコマンドを入力します。

```
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. 完了したら、コマンドプロンプトで次のコマンドオプションを使用して mongodb 5.0 シェルをインストールします。

```
sudo yum install -y mongodb-org-shell
```

On Ubuntu

Ubuntu に mongosh をインストールするには

1. パッケージ管理システムで使用されるパブリックキーをインポートします。

```
curl -fsSL https://pgp.mongodb.com/server-5.0.asc | sudo gpg --dearmor -o /usr/
share/keyrings/mongodb-server-5.0.gpg
```

2. Ubuntu のバージョンに適切なコマンドを使用して、MongoDB の リストファイル `mongodb-org-5.0.list` を作成します。

```
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. 次のコマンドを使用してローカルパッケージデータベースをインポートして更新します。

```
sudo apt-get update
```

4. mongosh をインストールします。

```
sudo apt-get install -y mongodb-mongosh
```

以前のバージョンの MongoDB の Ubuntu システムへのインストールについては、「Install MongoDB Community Edition on Ubuntu」を参照してください。

On other operating systems

mongo シェルを他のオペレーティングシステムにインストールするには、MongoDB ドキュメントの「MongoDB Community Edition のインストール」を参照してください。

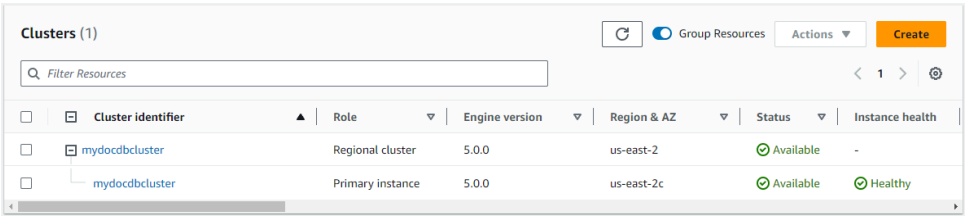
ステップ 6: Amazon DocumentDB TLS の管理

次のコード `wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem` を使用して Amazon DocumentDB の CA 証明書をダウンロードします。

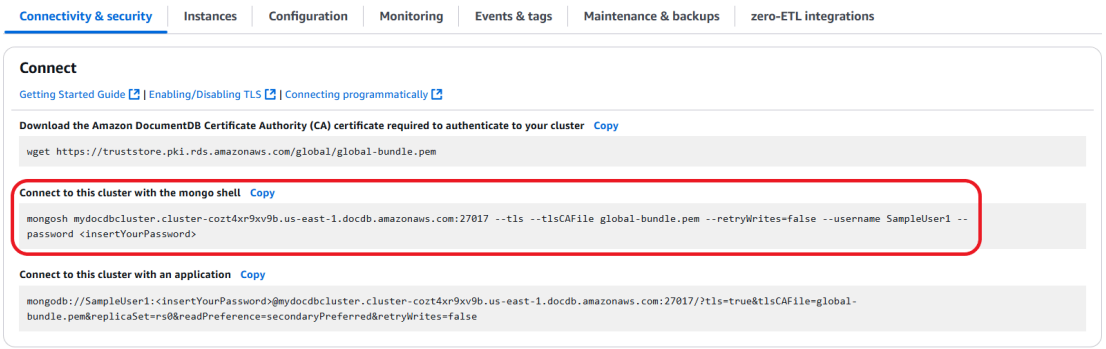
> **Note**
>
> 全ての新しい Amazon DocumentDB クラスターに対して、デフォルトで Transport Layer Security (TLS) が有効になっています。詳細については、「Encrypting data in transit」を参照してください。

ステップ 7: Amazon DocumentDB クラスターに接続する

1. Amazon DocumentDB コンソールの [クラスター] で、接続したいクラスターを見つけます。そのクラスターの [クラスター識別子] をクリックして、作成済みのクラスターを選択します。



2. [接続とセキュリティ] タブの [接続する] ボックス内で、[mongo シェルを使用してこのクラスターに接続する] を見つけます。



表示された接続文字列をコピーし、ターミナルに貼り付けます。

これに以下の変更を加えます。

a. 文字列に正しいユーザー名が含まれているのを確認してください。
b. <insertYourPassword> は省略します。これにより、接続時に mongo シェルによりパスワードの入力を求められます。
c. オプション: IAM 認証を使用している場合、または以前のバージョンの MongoDB シェルを使用している場合は、次のように接続文字列を変更します。

```
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 をクラスターの同じ情報に置き換えます。

3. ターミナル上で Enter キーを押します。ここで、パスワードの入力を求められます。パスワードを入力します。
4. パスワードを入力し、rs0 [direct: primary] <env-name>> プロンプトが表示されたら、Amazon DocumentDB クラスターに正常に接続されています。

接続の問題を解決するには、「」を参照してくださいAmazon DocumentDB のトラブルシューティング。

ステップ 8: データの挿入とクエリ

クラスターに接続できたので、いくつかのクエリを実行して、ドキュメントデータベースの使用に慣れることができます。

1. 1 つのドキュメントを挿入するには、次のように入力します。

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

次の出力が返ります。

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

```
}
```

2. findOne() コマンドで書き込んだドキュメントを読み取ることができます (一つのドキュメントしか返さないため)。以下を入力します。

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

次の出力が返ります。

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

3. さらにいくつかのクエリを実行するには、ゲームプロファイルのユースケースを検討してください。まず、「profiles」というタイトルのコレクションにエントリをいくつか挿入します。以下を入力します。

```
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 }
])
```

次の出力が返ります。

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

4. find() コマンドを使用して、プロファイルコレクション内のすべてのドキュメントを返します。以下を入力します。

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

ステップ 3 で入力したデータと一致する出力を取得します。

5. フィルターを使用して 1 つのドキュメントに対して1つのクエリを使用します。以下を入力します。

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

次の出力が返ります。

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

6. それでは、プロファイルを探して、それを findAndModify コマンドで変更してみましょう。次のコードを使用して、ユーザー Matt に 10 ポイント追加します。

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

次のような出力が得られます (Mattのスコアはまだ上がっていないことに注意してください) 。

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

7. 次のクエリを使用して、Matt のスコアが変更されたことを確認できます。

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

次の出力が返ります。

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

ステップ 9: 環境情報の確認

お疲れ様でした。Amazon DocumentDB のクイックスタートガイドを正常に完了しました。

次のステップ この強力なデータベースを一般的な特徴をいくつか使って十分に活用する方法を学びましょう。

- Amazon DocumentDB クラスターとインスタンスの管理
- スケーリング
- バックアップと復元

> **Note**
>
> コストを節約するために、Amazon DocumentDB クラスターを停止してコストを削減するか、クラスターを削除することができます。デフォルトでは、30 分間非アクティブになると、 AWS Cloud9 環境は基盤となる Amazon EC2 インスタンスを停止します。

Amazon DocumentDB JDBC ドライバーを使用して接続する

Amazon DocumentDB 用の JDBC ドライバーは、デベロッパーのための SQL リレーショナルインターフェイスを提供し、Tableau や DBVisualizer などの BI ツールからの接続を可能にします。

詳細については、GitHub 上の Amazon DocumentDB JDBC ドライバーのドキュメント を参照してください。

トピック

- 開始方法
- Tableau デスクトップから Amazon DocumentDB に接続する
- DbVisualizer から Amazon DocumentDB に接続する
- JDBC スキーマの自動生成
- SQL サポートと制限
- トラブルシューティング

開始方法

ステップ 1. Amazon DocumentDB クラスターの作成

Amazon DocumentDB クラスターが作成されていない場合は、「」の手順を使用してクラスターを作成しますAmazon DocumentDB の使用開始。

> **Note**
>
> Amazon DocumentDB は仮想プライベートクラウド (VPC) 専用サービスです。クラスターの VPC 外にあるローカルマシンから接続する場合は、Amazon EC2 インスタンスへの SSH 接続を作成する必要があります。この場合、「」の手順を使用してクラスターを起動しますAmazon EC2 を使用した接続。SSH トンネリングおよび必要な場合の詳細に

ついては、SSH トンネルを使って Amazon DocumentDB に接続する を参照してください。

ステップ 2. JRE または JDK インストール

BI アプリケーションによっては、64 ビット JRE または JDK インストールバージョン 8 以降がコンピュータにインストールされていることを確認する必要がある場合があります。Java SE ランタイム環境 8 をダウンロードは こちら。

ステップ 3. DocumentDB JDBC ドライバーのダウンロード

here(こちら) からのDocumentDB JDBC ドライバーのダウンロード ドライバーは単一の JAR ファイルとしてパッケージ化されています (例: documentdb-jdbc-1.0.0-all.jar)。

ステップ 4. SSH トンネルを使用して Amazon DocumentDB に接続する

Amazon DocumentDB (MongoDB との互換性) クラスターは、Amazon Virtual Private Cloud (Amazon VPC) 内にデプロイされます。これらは、Amazon EC2 インスタンス、または同じ Amazon VPC にデプロイされている他の AWS サービスから直接アクセスできます。さらに、Amazon DocumentDB は、VPC ピアリングを介して、同じ AWS リージョンまたは他のリージョンの異なる VPCs 内の EC2a インスタンスまたは他の AWS サービスからアクセスできます。

SSH トンネリング (ポート転送とも呼ばれます) を使用して、クラスターの VPC 外部から Amazon DocumentDB リソースにアクセスできます。これは、ほとんどのユーザーが DocumentDB クラスターと同じ VPC 内の仮想マシンでアプリケーションを実行していない場合です。

SSH トンネルを作成するには、Amazon DocumentDB クラスターと同じ Amazon VPC で実行中の Amazon EC2 インスタンスが必要です。クラスターと同じ VPC で既存の EC2 インスタンスを使用するか、新しい EC2 インスタンスを作成できます。その場合、ローカルコンピュータで以下のコマンドを実行することにより Amazon DocumentDB クラスター `sample-cluster.node.us-east-1.docdb.amazonaws.com` に SSH トンネルを設定できます。

```
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
```

ローカルポートの転送を行う場合は、-L フラグを使用します。これは、VPC 外のクライアントで実行されている BI ツールに接続するための前提条件です。上記のステップを実行したら、選択した BI ツールの次のステップに進むことができます。

SSH トンネリング の詳細については、「SSH トンネルを使用して Amazon DocumentDB に接続する」のドキュメントを参照してください。

Tableau デスクトップから Amazon DocumentDB に接続する

トピック

- Amazon DocumentDB JDBC ドライバーの追加
- Tableau を使用して Amazon DocumentDB に接続する - SSH トンネル

Amazon DocumentDB JDBC ドライバーの追加

Tableau デスクトップから Amazon DocumentDB に接続するには、DocumentDB JDBC ドライバーと DocumentDB Tableau コネクタをダウンロードしてインストールする必要があります。

1. Amazon DocumentDB JDBC ドライバー JAR ファイルを Amazon DocumentDB JDBC ドライバーレポジトリ からダウンロードし、ご使用のオペレーティングシステムに応じて次のいずれかのディレクトリにコピーします。
   - Windows – `C:\Program Files\Tableau\Drivers`
   - MacOS – `~/Library/Tableau/Drivers`
2. DocumentDB Tableau コネクタ (TACO ファイル) を Tableau Exchange ウェブサイト からダウンロードし、My Tableau リポジトリ/コネクタディレクトリ にコピーします。
   - Windows – `C:\Users\[user]\Documents\My Tableau Repository\Connectors`
   - MacOS – `/Users/[user]/Documents/My Tableau Repository/Connectors`

詳細 (追加情報) については、Tableau のドキュメント を参照してください。

> **Note**
>
> 新しい CA 証明書を使用している場合は、JDBC ドライバーを v1.4.5 (この AWS GitHub リポジトリで利用可能) にアップグレードしてください。

Tableau を使用して Amazon DocumentDB に接続する - SSH トンネル

DocumentDB クラスターの VPC 外のクライアントマシンから Tableau に接続するには、以下のステップを実行する前に SSH トンネルを設定する必要があります。

1. Tableau デスクトップアプリケーションを起動します。
2. Connect > To A Server > More に移動します。
3. [インストールされたコネクタ] で Amazon Web Services の Amazon DocumentDB を選択します。

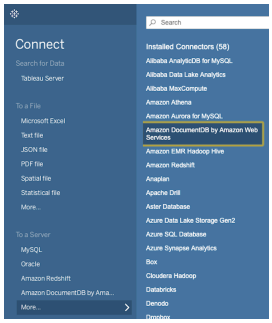
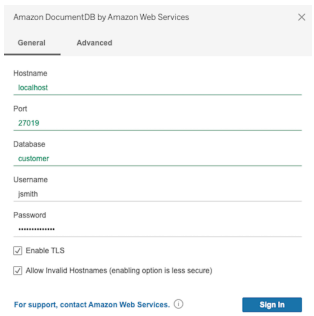


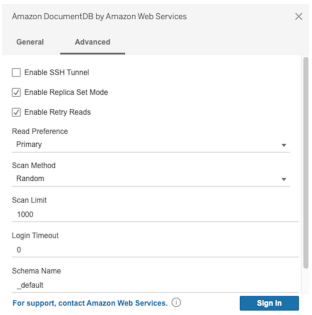
Tableau を使用した Amazon DocumentDB への接続 - 外部 SSH トンネル

1. 必要な接続パラメータを入力します。Hostname、ポート、データベース、ユーザーネーム、パスワード です。次の例の接続パラメータは、JDBC 接続文字列 と同等です。

   ユーザーネームとパスワードのパラメータを伴う `jdbc:documentdb://localhost:27019/test?tls=true&tlsAllowInvalidHostnames=true&scanMethod=random&scanLimit=1000&login` は、プロパティコレクションで別々に渡されました。接続文字列パラメータの詳細については、Amazon DocumentDB JDBC ドライバー github ドキュメント を参照してください。



2. (オプション) より高度なオプションは、アドバンスト タブで見つけることができます。

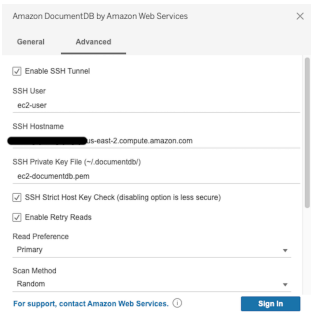


3. [サインイン] を選択します。

Tableau を使用した Amazon DocumentDB への接続 - 内部 SSH トンネル

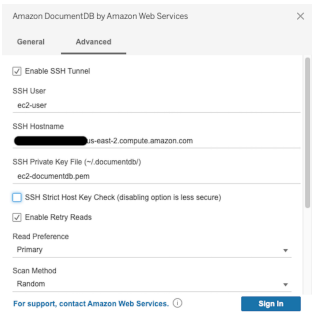
> **Note**
>
> 端末を使用して SSH トンネルをセットアップしたくない場合は、Tableau GUI を使用して、JDBC ドライバーが本質的に SSH トンネルの作成に使用する EC2 インスタンスの詳細を指定できます。

1. アドバンスト タブで、SSH トンネルオプションを有効にする を選択して、その他のプロパティを確認します。



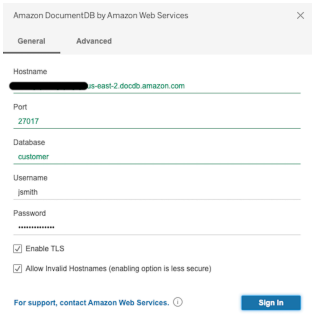
2. SSH ユーザー、SSH ホスト名、SSH プライベートキーファイル を入力します。
3. (オプション) SSH 厳密ホストキーチェック オプションを無効にできます。これを使用すると、既知のホストファイルに対するホストキーのチェックをバイパスします。

> **Note**
>
> このオプションを無効にすると、中間者 攻撃につながり得るので、安全性が低くなります。



4. 必要な接続パラメータを入力します。Hostname、ポート、データベース、ユーザーネーム、パスワード です。

> **Note**
>
> 内部 SSH トンネルオプションを使用する場合は、ローカルホストではなく DocumentDB クラスターエンドポイントを確実に使用してください。



5. [サインイン] を選択します。

DbVisualizer から Amazon DocumentDB に接続する

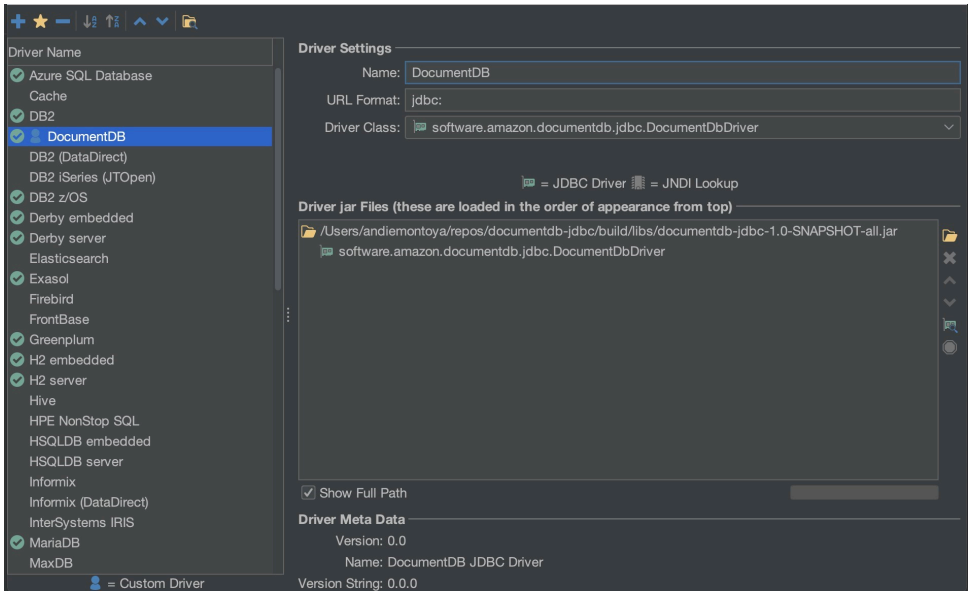
トピック

- Amazon DocumentDB JDBC ドライバーの追加
- DBVisualizer を使用して Amazon DocumentDB に接続する

Amazon DocumentDB JDBC ドライバーの追加

DbVisualizer から Amazon DocumentDB に接続するには、まずAmazon DocumentDB JDBC ドライバーをインポートする必要があります。

1. DbVisualizer アプリケーションをスタートし、メニューパスに移動します : ツール > ドライバーマネージャー...
2. + を選択します (またはメニューで、ドライバ ー> ドライバーを作成 を選択) 。
3. [Name] を [DocumentDB] に設定します。
4. URL 形式 を `jdbc:documentdb://<host>[:port]/<database>[?option=value[&option=value[...]]]` に設定
5. フォルダ ボタンを選択し、Amazon DocumentDB JDBC ドライバー JAR ファイルを選択し、オープン ボタンを選択します。
6. ドライバークラス フィールドが `software.amazon.documentdb.jdbc.DocumentDbDriver` になっていることを確認します。DocumentDB のドライバーマネージャの設定は、次の例のようになります。



7. ダイアログを閉じます。Amazon DocumentDB JDBC ドライバーがセットアップされ、使用できる状態になります。

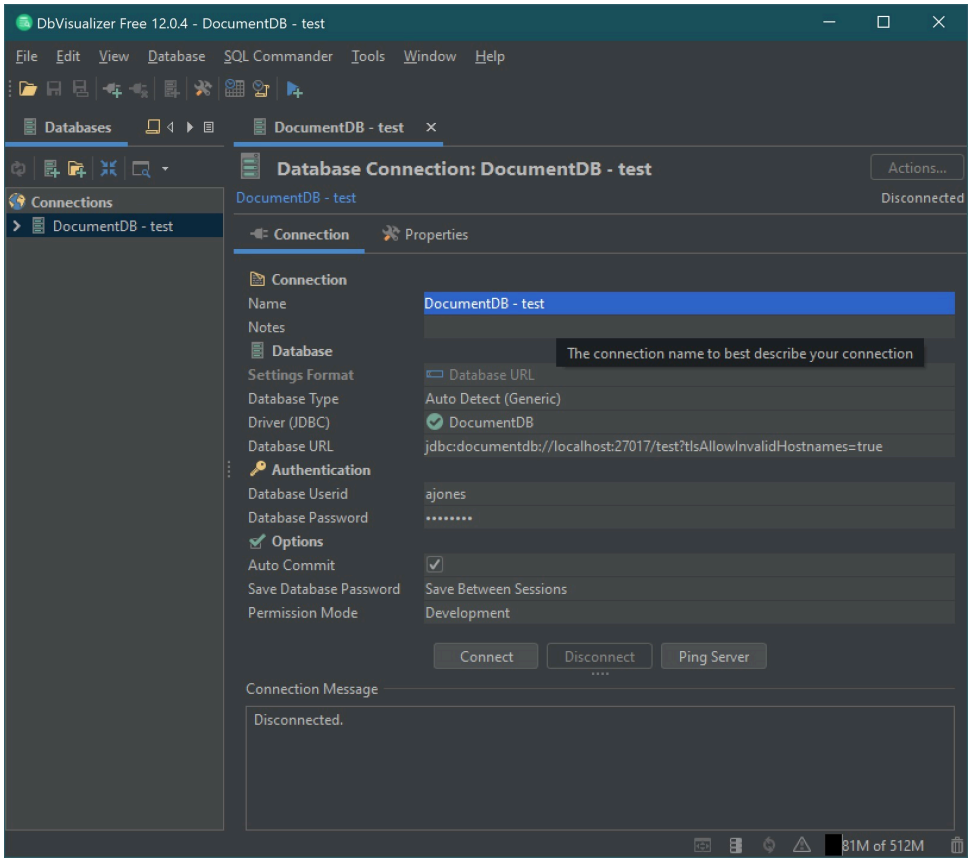
DBVisualizer を使用して Amazon DocumentDB に接続する

DBVisualizer を使用して Amazon DocumentDB に接続する

1. Amazon DocumentDB クラスターの VPC の外部から接続する場合は、SSH トンネルが設定されていることを確認してください。
2. トップレベルメニューから、データベース > データベース接続の作成 を選択します。
3. 名前 フィールドに、わかりやすい名前を入力します。
4. ドライバー (JDBC) を前のセクションで作成した DocumentDB ドライバーに設定します。
5. データベース URL を JDBC 接続文字列に設定します。

   例: `jdbc:documentdb://localhost:27017/database?tlsAllowInvalidHostnames=true`
6. データベースユーザー ID を Amazon DocumentDB ユーザー ID に設定します。
7. データベースパスワード を、ユーザー ID に対応するパスワードに設定します。

   データベース接続ダイアログは、次のダイアログのようになります。



8. 接続 を選択します。

JDBC スキーマの自動生成

Amazon DocumentDB はドキュメントデータベースであるため、テーブルとスキーマの概念はありません。ただし、Tableau などの BI ツールは、データベースがスキーマを提示することを期待します。具体的には、JDBC ドライバー接続でデータベース内のコレクションのスキーマを取得する必要がある場合は、データベース内のすべてのコレクションをポーリングします。ドライバーは、そのコレクションのスキーマのキャッシュバージョンがすでに存在するかどうかを判別します。キャッシュされたバージョンが存在しない場合は、ドキュメントのコレクションをサンプリングし、次の動作に基づいてスキーマを作成します。

トピック

- スキーマ生成の制限
- スキャン方法のオプション
- Amazon DocumentDB データ型
- スカラードキュメントフィールドのマッピング
- オブジェクトと配列のデータ型の処理

スキーマ生成の制限

DocumentDB JDBC ドライバーは識別子の長さに 128 文字の制限を課しています。スキーマジェネレータは、生成された識別子 (テーブル名とカラム名) の長さを切り捨てて、その制限に適合するようにします。

スキャン方法のオプション

サンプリング動作は、接続文字列またはデータソースオプションを使用して変更できます。

- scanMethod=<option>
  - random - (デフォルト) - サンプルドキュメントはランダムな順序で返されます。
  - idForward - サンプルドキュメントは ID の順序で返されます。
  - idReverse - サンプルドキュメントは id の逆の順序で返されます。
  - all - コレクション内のすべてのドキュメントをサンプリングします。
- ScanLimit=<n> - サンプリングするドキュメントの数。値は正の整数である必要があります。デフォルト値は 1000 です。scanMethod が all に設定されている場合、このオプションは無視されます。

Amazon DocumentDB データ型

Amazon DocumentDB サーバーはいくつかの MongoDB データ型をサポートしています。サポートされているデータ型とそれに関連付けられた JDBC データ型を以下に示します。

MongoDB データ型	DocumentDB でサポートされています	JDBC データ型
バイナリデータ	はい	VARBINARY
ブール値	はい	BOOLEAN
Double	はい	DOUBLE
32 ビット整数	はい	INTEGER
64 ビット整数	はい	BIGINT
String	はい	VARCHAR
ObjectId	はい	VARCHAR
日付	可能	TIMESTAMP
Null	はい	VARCHAR
正規表現	はい	VARCHAR
タイムスタンプ	はい	VARCHAR
MinKey	はい	VARCHAR
MaxKey	はい	VARCHAR
オブジェクト	はい	仮想テーブル
配列	はい	仮想テーブル
Decimal128	いいえ	DECIMAL
JavaScript	いいえ	VARCHAR

MongoDB データ型	DocumentDB でサポートされています	JDBC データ型
JavaScript (スコープ付き)	いいえ	VARCHAR
未定義	いいえ	VARCHAR
記号	いいえ	VARCHAR
dbPointer (4.0+)	いいえ	VARCHAR

スカラードキュメントフィールドのマッピング

コレクションからドキュメントのサンプルをスキャンする場合、JDBC ドライバーは、コレクション内のサンプルを表す 1 つ以上のスキーマを作成します。一般に、ドキュメントのスカラーフィールドは、テーブルスキーマの列にマップされます。例えば、team というコレクションと 1 つのドキュメント `{ "_id" : "112233", "name" : "Alastair", "age": 25 }` の場合、これはスキーマにマップされます。

テーブル名	Column Name (列名)	データタイプ	キー
team	チームID	VARCHAR	PK
team	name	VARCHAR	
team	年齢	INTEGER	

データ型の競合プロモーション

サンプリングされたドキュメントをスキャンするときに、フィールドのデータ型がドキュメント間で一貫していない可能性があります。この場合、JDBC ドライバーは JDBC データ型を、サンプリングされたドキュメントのすべてのデータ型に適合する共通データ型に昇格します。

例:

```
{
"_id" : "112233",
"name" : "Alastair", "age" : 25
```

```
}

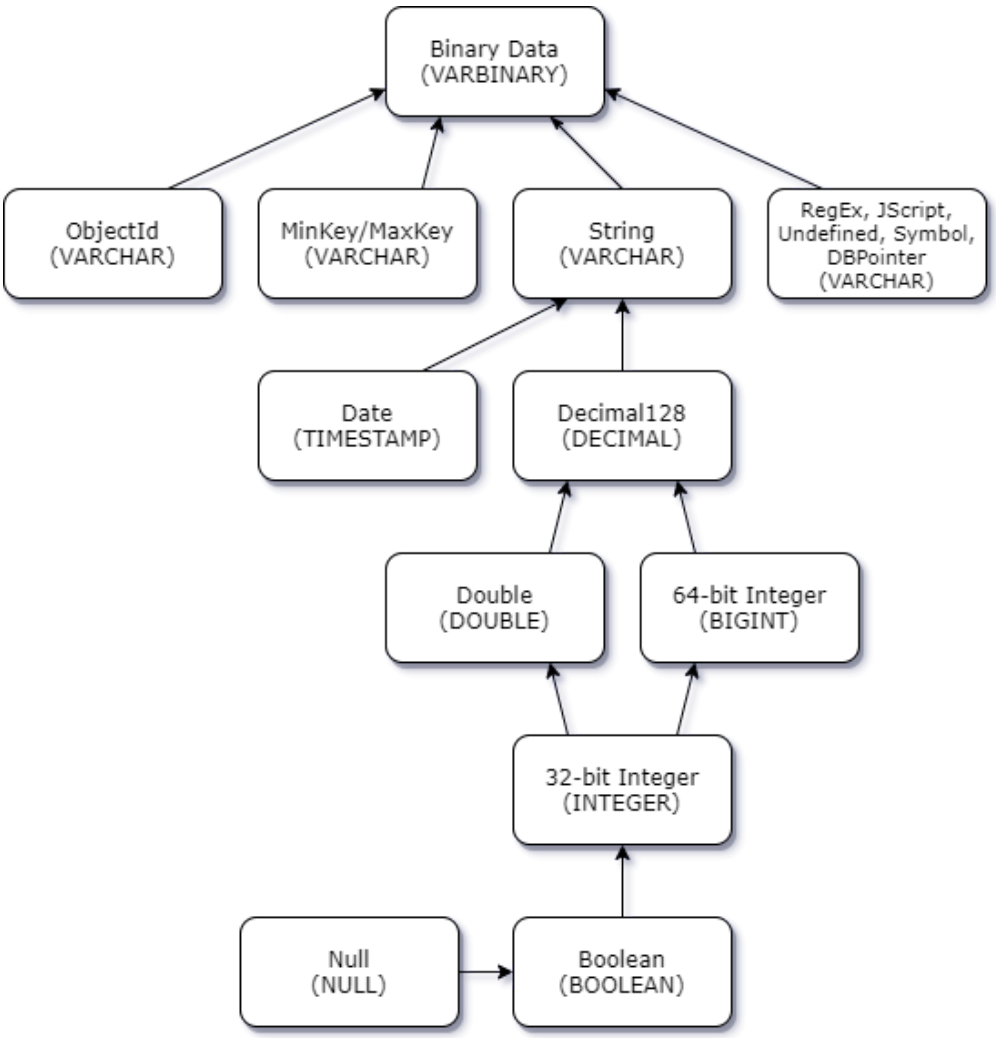
{
"_id" : "112244",
"name" : "Benjamin",
"age" : "32"
}
```

年齢 フィールドは、最初のドキュメントでは 32 ビット整数ですが、2 番目のドキュメントでは文字列です。ここで、JDBC ドライバーは JDBC データ型を VARCHAR に昇格させ、いずれかのデータ型を検出したときに処理します。

テーブル名	Column Name (列名)	データタイプ	キー
team	チームID	VARCHAR	PK
team	name	VARCHAR	
team	年齢	VARCHAR	

スカラー - スカラー競合プロモーション

次の図表は、スカラー - スカラーデータ型の競合を解決する方法を示しています。



スカラー - 複合型競合プロモーション

スカラー· スカラー型の競合と同様に、異なるドキュメント内の同じフィールドは、複素数 (配列とオブジェクト) とスカラー (整数、ブールなど) の間で競合するデータ型を持つことができます。これらの競合はすべて、それらのフィールドについて VARCHAR に解決 (昇格) されます。この場合、配列とオブジェクトのデータは JSON 表現として返されます。

埋め込み配列 - 文字列フィールドの競合の例 :

```
{
   "_id":"112233",
   "name":"George Jackson",
   "subscriptions":[
      "Vogue",
      "People",
      "USA Today"
```

```
   ]
}
{
   "_id":"112244",
   "name":"Joan Starr",
   "subscriptions":1
}
```

前の例では、customer2 テーブルのスキーマにマッピングします。

テーブル名	Column Name (列名)	データタイプ	キー
customer2	Customer2 ID	VARCHAR	PK
customer2	name	VARCHAR	
customer2	サブスクリプション	VARCHAR	

customer1_subscriptions 仮想テーブルは次のとおりです。

テーブル名	Column Name (列名)	データタイプ	キー
customer1_subscriptions	customer1 ID	VARCHAR	PK/FK
customer1_subscriptions	subscriptions_index_lvl0	BIGINT	PK
customer1_subscriptions	値	VARCHAR	
customer_address	city	VARCHAR	
customer_address	リージョン	VARCHAR	
customer_address	country	VARCHAR	
customer_address	コード	VARCHAR	

オブジェクトと配列のデータ型の処理

これまでは、スカラーデータ型のマッピング方法のみについて説明しました。オブジェクトデータ型と配列データ型は (現在) 仮想テーブルにマッピングされています。JDBC ドライバーは、ドキュメント内のオブジェクトまたは配列フィールドを表す仮想テーブルを作成します。マッピングされた仮想テーブルの名前は、元のコレクションの名前の後にフィールドの名前をアンダースコア (「_」) で区切って連結します。

ベーステーブルの主要なキー (「_id」) は、新しい仮想テーブル内の新しい名前を取得し、関連付けられたベーステーブルの外部キーとして提供されます。

埋め込み配列型フィールドの場合、配列の各レベルの配列へのインデックスを表すインデックス列が生成されます。

埋め込みオブジェクトフィールドの例

ドキュメント内のオブジェクトフィールドの場合、仮想テーブルへのマッピングは JDBC ドライバーによって作成されます。

```
{
   "Collection: customer",
   "_id":"112233",
   "name":"George Jackson",
   "address":{
      "address1":"123 Avenue Way",
      "address2":"Apt. 5",
      "city":"Hollywood",
      "region":"California",
      "country":"USA",
      "code":"90210"
   }
}
```

前述の例は、顧客テーブルのスキーマにマッピングされます。

テーブル名	Column Name (列名)	データタイプ	キー
カスタマー	カスタマー ID	VARCHAR	PK
カスタマー	name	VARCHAR	

および顧客アドレス仮想テーブルは次のとおりです。

テーブル名	Column Name (列名)	データタイプ	キー
customer_address	カスタマー ID	VARCHAR	PK/FK
customer_address	address1	VARCHAR	
customer_address	address2	VARCHAR	
customer_address	city	VARCHAR	
customer_address	リージョン	VARCHAR	
customer_address	country	VARCHAR	
customer_address	コード	VARCHAR	

埋め込み配列フィールドの例

ドキュメント内の配列フィールドの場合、JDBC ドライバーによって仮想テーブルへのマッピングも作成されます。

```
{
   "Collection: customer1",
   "_id":"112233",
   "name":"George Jackson",
   "subscriptions":[
      "Vogue",
      "People",
      "USA Today"
   ]
}
```

上記の例は、customer1 テーブルのスキーマにマッピングされます。

テーブル名	Column Name (列名)	データタイプ	キー
customer1	customer1 ID	VARCHAR	PK

テーブル名	Column Name (列名)	データタイプ	キー
customer1	name	VARCHAR	

customer1_subscriptions 仮想テーブルは次のとおりです。

テーブル名	Column Name (列名)	データタイプ	キー
customer1_subscriptions	customer1 ID	VARCHAR	PK/FK
customer1_subscriptions	subscriptions_index_lvl0	BIGINT	PK
customer1_subscriptions	値	VARCHAR	
customer_address	city	VARCHAR	
customer_address	リージョン	VARCHAR	
customer_address	country	VARCHAR	
customer_address	コード	VARCHAR	

SQL サポートと制限

Amazon DocumentDB JDBC ドライバーは、SQL-92 のサブセットといくつかの一般的な拡張を Support (サポート)する読み取り専用ドライバーです。詳細については、SQL の制限のドキュメントと JDBC の制限のドキュメント を参照してください。

トラブルシューティング

Amazon DocumentDB JDBC ドライバーの使用に問題がある場合は、トラブルシューティングガイド を参照してください。

Amazon DocumentDB JDBC ドライバーを使用して接続する

Amazon DocumentDB 用の ODBC ドライバーは、デベロッパーのための SQL リレーショナルインターフェイスを提供し、Power BI Desktop や Microsoft Excel などの BI ツールからの接続を可能にします。

詳細については、GitHub 上の Amazon DocumentDB JDBC ドライバーのドキュメントを参照してください。

トピック

- 開始方法
- Windows での Amazon DocumentDB ODBC ドライバーのセットアップ
- Microsoft Excel から Amazon DocumentDB に接続
- Microsoft Power BI Desktop から Amazon DocumentDB に接続します
- 自動スキーマ生成
- SQL サポートと制限
- トラブルシューティング

開始方法

ステップ 1. Amazon DocumentDB クラスターの作成

Amazon DocumentDB クラスターをまだお持ちでない場合、始める方法はいくつかあります。

> **Note**
>
> Amazon DocumentDB は仮想プライベートクラウド (VPC) 専用サービスです。クラスターの VPC 外のローカルマシンから接続する場合は、Amazon EC2 インスタンスへの SSH 接続を作成する必要があります。詳細については、「Amazon EC2 を使用した接続」を参照してください。SSH トンネリングの詳細については、「SSH トンネルを使用して Amazon DocumentDB に接続する」を参照してください。

ステップ 2. JRE または JDK インストール

BI アプリケーションによっては、64 ビット JRE または JDK インストールバージョン 8 以降がコンピュータにインストールされていることを確認する必要がある場合があります。Java SE ランタイム環境 8 をダウンロードは こちら。

ステップ 3. Amazon DocumentDB ODBC ドライバーのダウンロード

Amazon DocumentDB ODBC ドライバーはこちらからダウンロードしてください。適切なインストーラー (たとえば documentdb-odbc-1.0.0.msi) を選択します。インストールガイドに従います。

Step 4. SSH トンネルを使用して Amazon DocumentDB に接続する

Amazon DocumentDB クラスターは、Amazon Virtual Private Cloud (Amazon VPC) 内にデプロイされます。これらは、Amazon EC2 インスタンス、または同じ Amazon VPC にデプロイされている他の AWS サービスから直接アクセスできます。さらに、Amazon DocumentDB には、同じリージョンまたは他のリージョンの異なる VPCs の Amazon EC2 インスタンス AWS リージョンまたは他の AWS サービスが VPC ピアリングを介してアクセスできます。

ただし、お客様のユースケースで、お客様 (またはお客様のアプリケーション) が、クラスターの VPC の外部から Amazon DocumentDB リソースにアクセスする必要があるとします。これは、ほとんどのユーザーが Amazon DocumentDB クラスターと同じ VPC 内の仮想マシンでアプリケーションを実行していない場合です。VPC 外から接続する場合、SSH トンネリング (ポート転送とも呼ばれる) を使用して Amazon DocumentDB リソースにアクセスできます。

SSH トンネルを作成するには、Amazon DocumentDB クラスターと同じ Amazon VPC で実行中の Amazon EC2 インスタンスが必要です。クラスターと同じ VPC で既存の EC2 インスタンスを使用するか、新しい EC2 インスタンスを作成できます。その場合、ローカルコンピュータで以下のコマンドを実行することにより Amazon DocumentDB クラスター `sample-cluster.node.us-east-1.docdb.amazonaws.com` に SSH トンネルを設定できます。

```
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
```

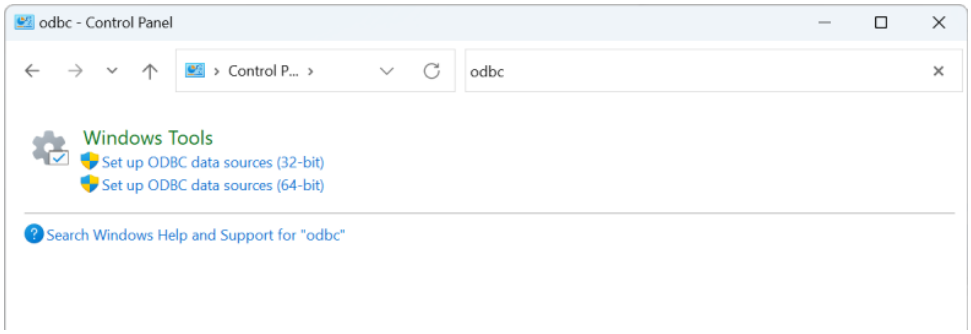
ローカルポートの転送を行う場合は、-L フラグを使用します。これは、VPC 外のクライアントで実行されている BI ツールに接続するための前提条件です。上記のステップを実行したら、選択した BI ツールの次のステップに進むことができます。

SSH トンネリングの詳細については、「SSH トンネルを使用して Amazon DocumentDB に接続する」のドキュメントを参照してください。

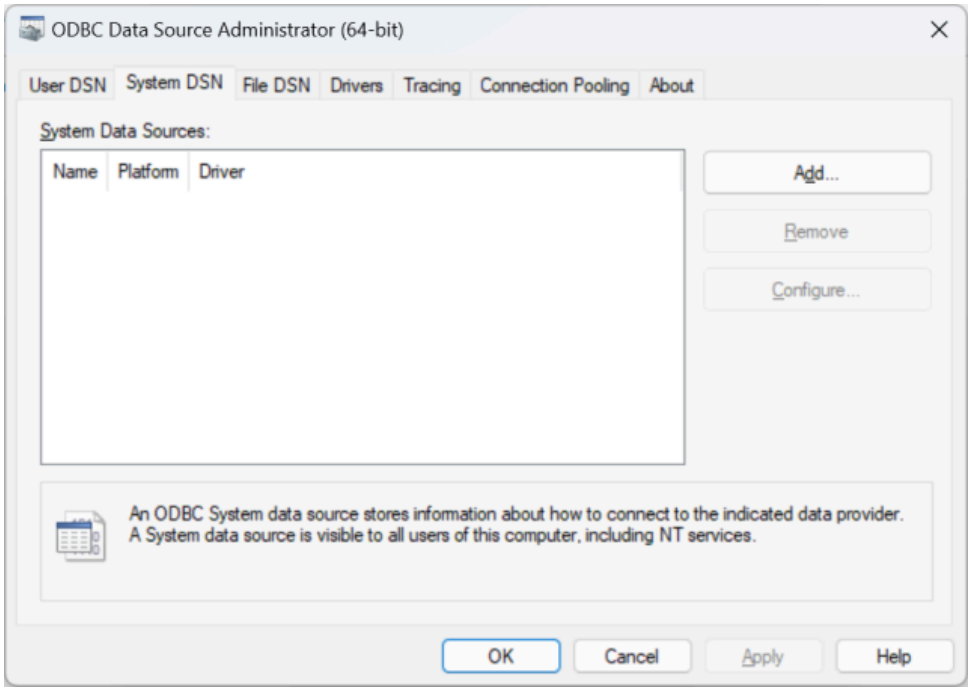
Windows での Amazon DocumentDB ODBC ドライバーのセットアップ

以下の手順を使用して、Windows で Amazon DocumentDB ODBC ドライバーを設定します。

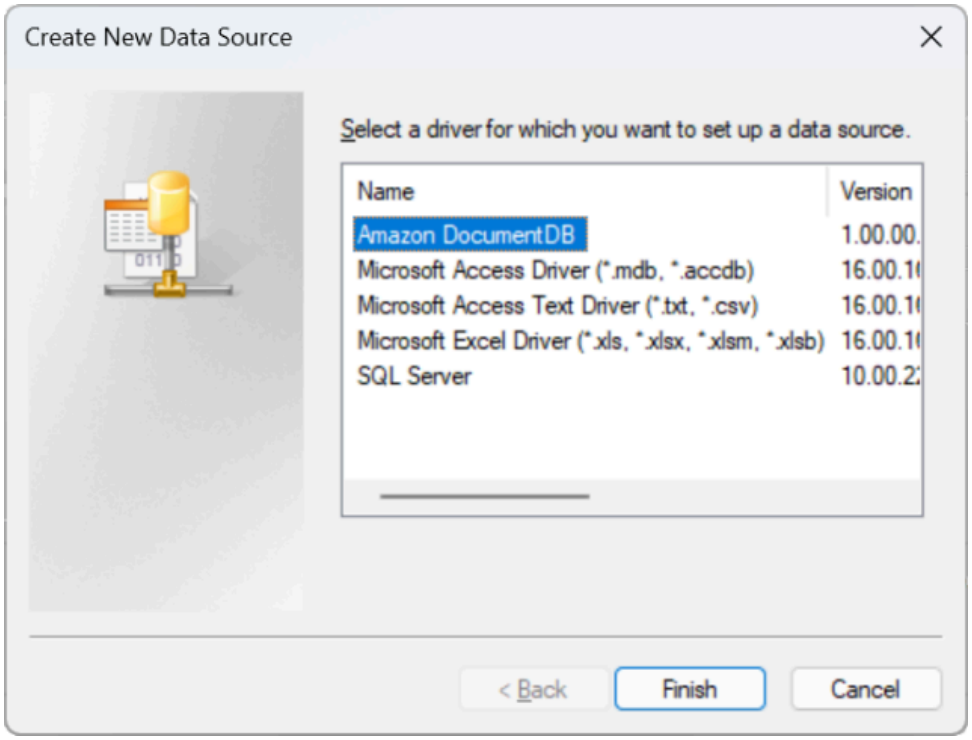
1. Windows で [コントロールパネル] を開き、ODBC を検索します (またはメニューで [Windows ツール] > [ODBC データソース (32 ビット)] または [ODBC データソース (64 ビット)] を選択)。



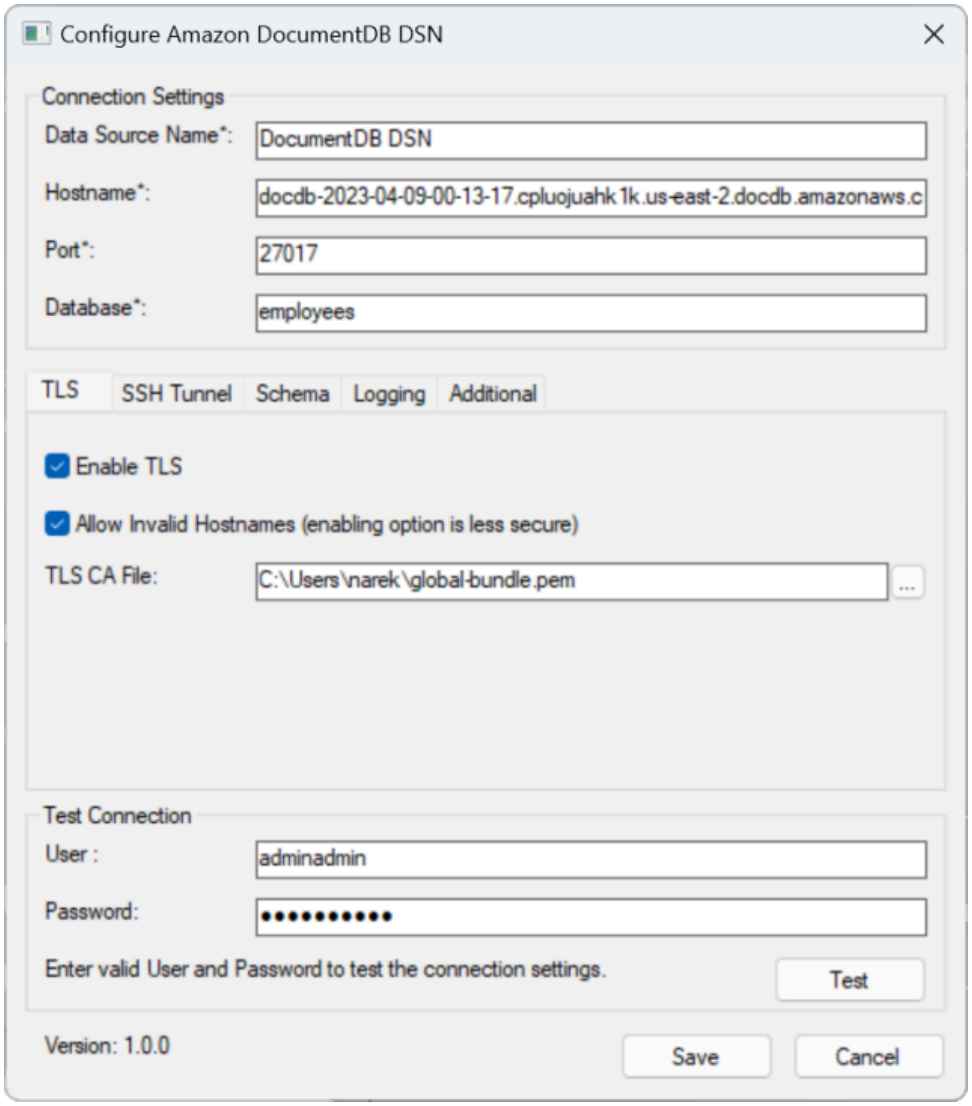
2. 適切な ODBC ドライバデータソース管理者を選択します。インストールされていれば 32 ビットバージョンを、インストールされていない場合は 64 ビットバージョンを選択します。
3. Sytem DSN タブを選択し、追加... を選択して新しい DSN を追加します。



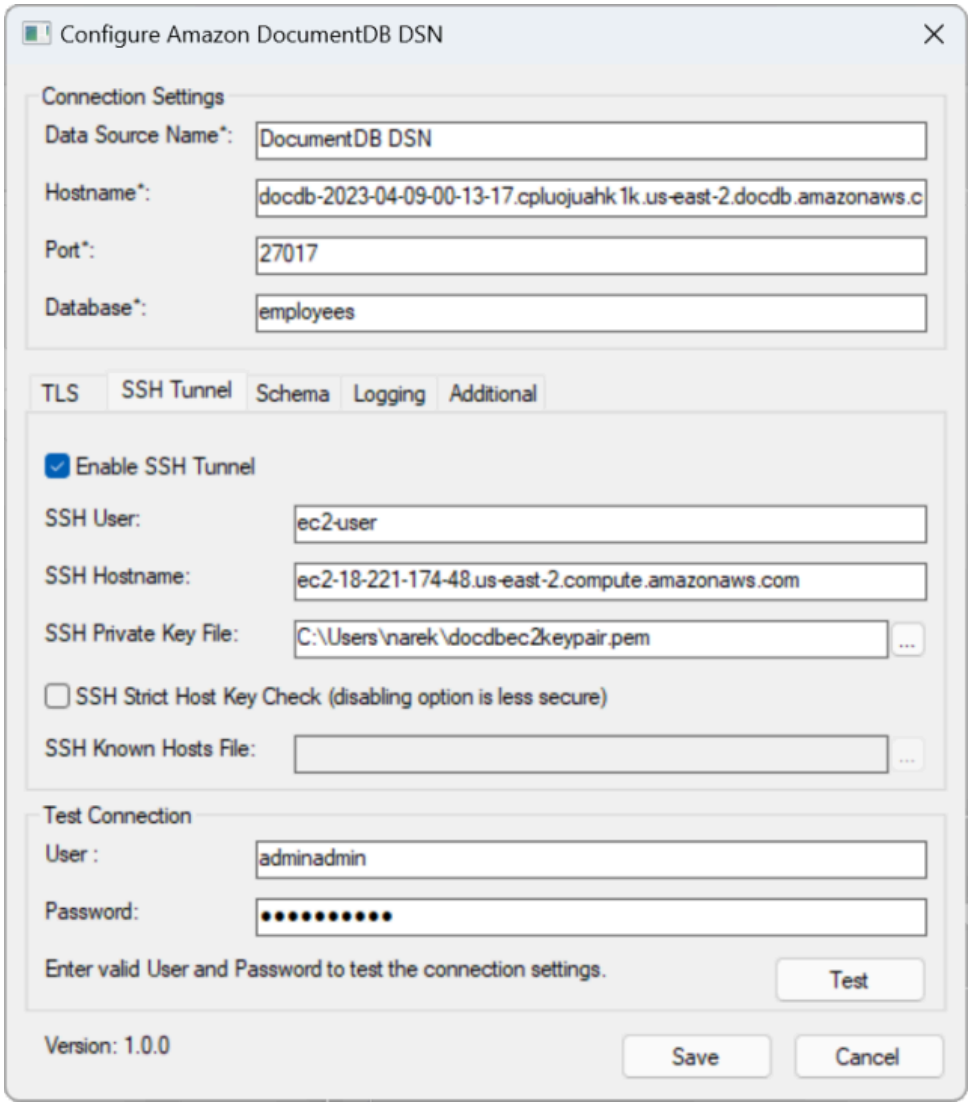
4. データソースドライバーリストから [Amazon DocumentDB] を選択します。



5. Amazon DocumentDB DSN の設定ダイアログで、接続設定、TLS タブ、および接続のテストフィールドに入力し、保存を選択します。



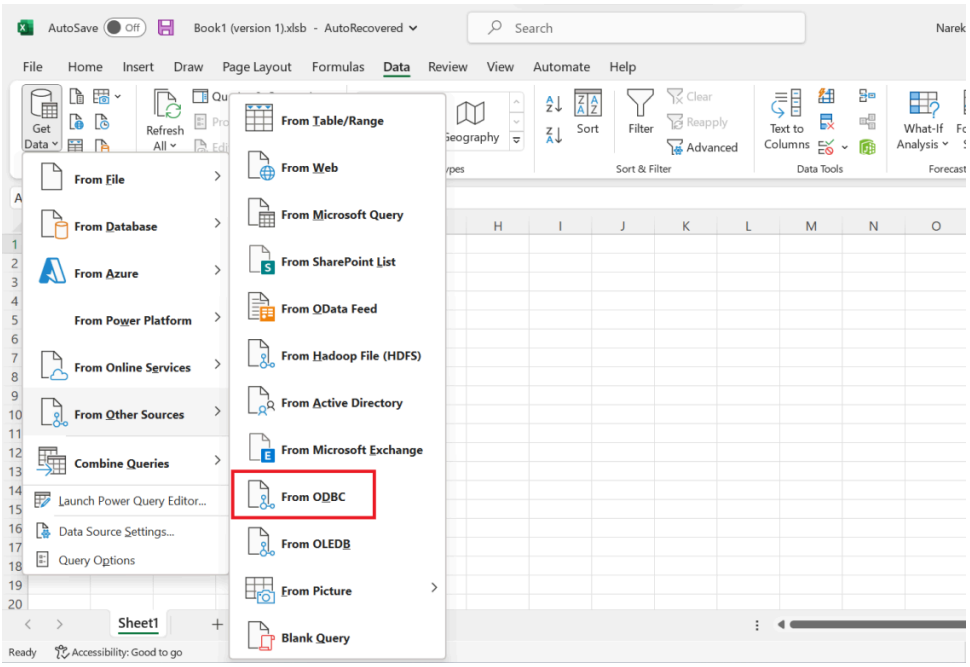
6. 選択した EC2 インスタンスへの SSH トンネリング方法によって接続の詳細が異なるため、Windows フォームには正確に記入してください。SSH トンネリング方法についてはこちらを参照してください。各プロパティの詳細については、「接続文字列の構文とオプション」を参照してください。



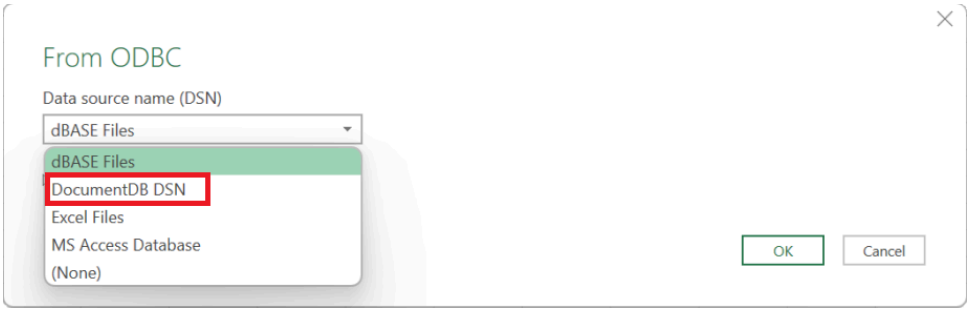
Windows での Amazon DocumentDB ODBC ドライバーの設定の詳細については、こちらを参照してください。

Microsoft Excel から Amazon DocumentDB に接続

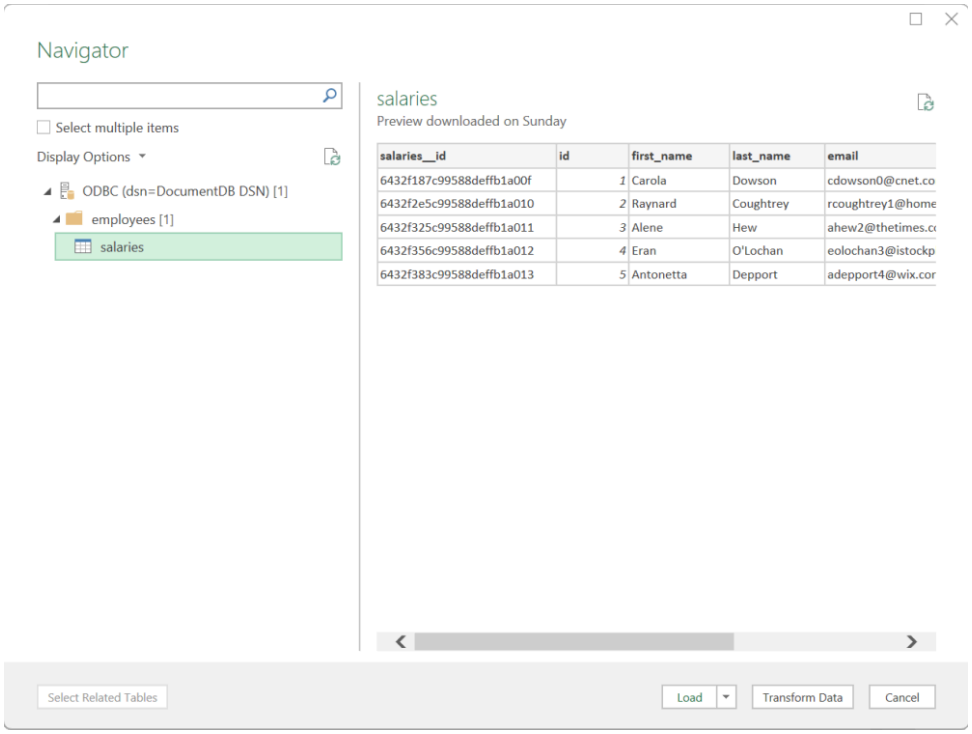
1. Amazon DocumentDB ドライバーが正しくインストールされ、設定されていることを確認します。詳細については、「Windows での Amazon DocumentDB ODBC ドライバーのセットアップ」を参照してください。
2. Microsoft Excel を起動します。
3. [データ] > [データの取得] > [他のソースから] に移動します。
4. [ODBC から] を選択します。



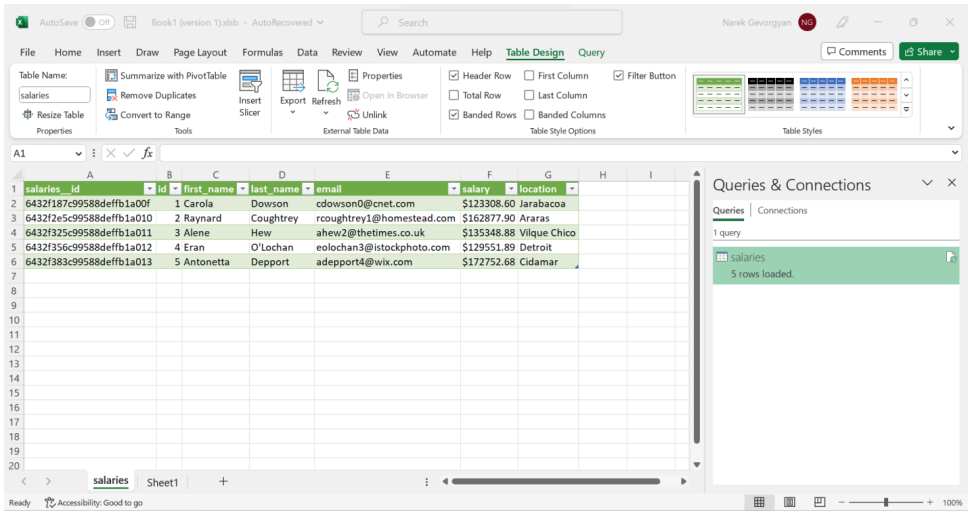
5. Amazon DocumentDB に関連付けられているデータソース名 (DSN) ドロップダウンメニューから、データソースを選択します。



6. Excel にデータをロードするコレクションを選択します。



7. Excel にデータをロードします。



Microsoft Power BI Desktop から Amazon DocumentDB に接続します

トピック

- 前提条件
- Microsoft Power BI Desktop カスタムコネクタの追加
- Amazon DocumentDB カスタムコネクタを使用して接続します
- Microsoft Power BI ゲートウェイの設定

前提条件

開始する前に、Amazon DocumentDB ODBC ドライバーが正しくインストールされていることを確認してください。

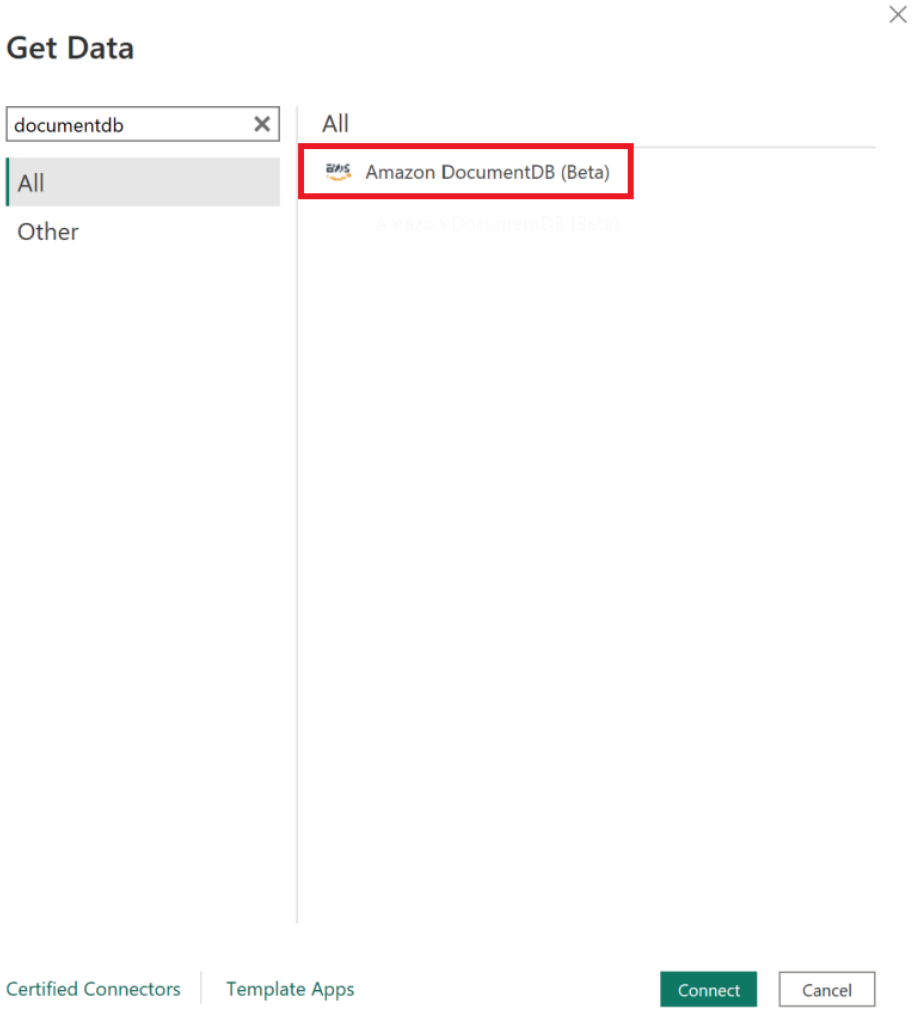
Microsoft Power BI Desktop カスタムコネクタの追加

`AmazonDocumentDBConnector.mez` ファイルを `<User>\Documents\Power BI Desktop \Custom Connectors\` フォルダー (または OneDrive を使用している場合は `<User>\OneDrive \Documents\Power BI Desktop\Custom Connectors`) にコピーします。これにより、Power BI はカスタムコネクタにアクセスできるようになります。Power BI Desktop へのコネクタはこちらから入手できます。Power BI Desktop を再起動して、コネクタがロードされていることを確認します。

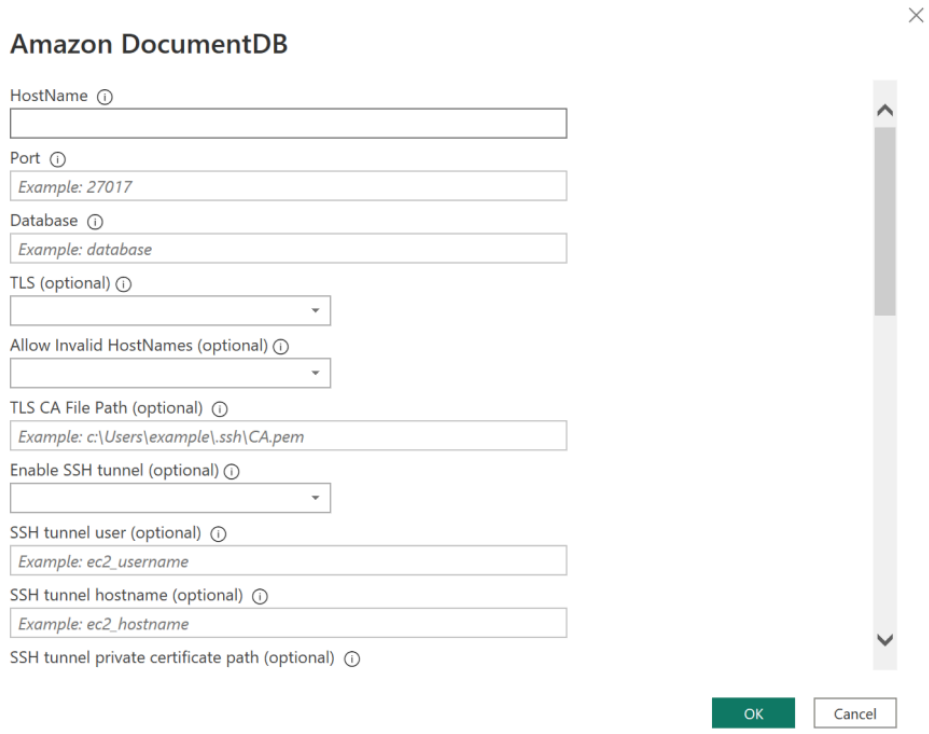
> **Note**
>
> カスタムコネクタは、認証用に Amazon DocumentDB のユーザー名とパスワードのみをサポートします。

Amazon DocumentDB カスタムコネクタを使用して接続します

1. Get Data から Amazon DocumentDB (ベータ) を選択し、Connect を選択します。 サードパーティーのサービスの使用に関する警告が表示された場合は、続行を選択します。



2. Amazon DocumentDB クラスターに接続するために必要なすべての情報を入力し、OK を選択します。



> **Note**
>
> ODBC ドライバーのデータソース名 (DSN) の設定によっては、DSN 設定で必要な情報をすでに提供している場合、SSH 接続の詳細画面が表示されない場合があります。

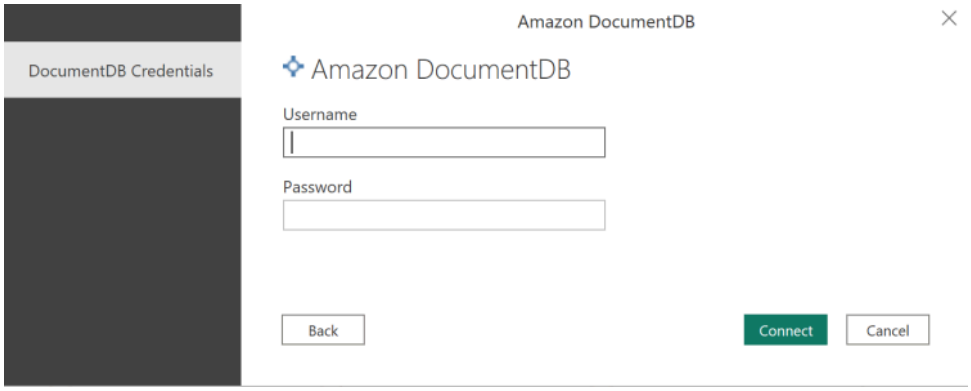
3. データ接続モードを選択します。

   - インポート: すべてのデータをロードし、情報をディスクに保存します。データの更新を表示するには、データを更新して再ロードする必要があります。
   - ダイレクトクエリ: データをロードしませんが、データに対してライブクエリを実行します。つまり、データの更新を表示するためにデータを更新して再ロードする必要はありません。

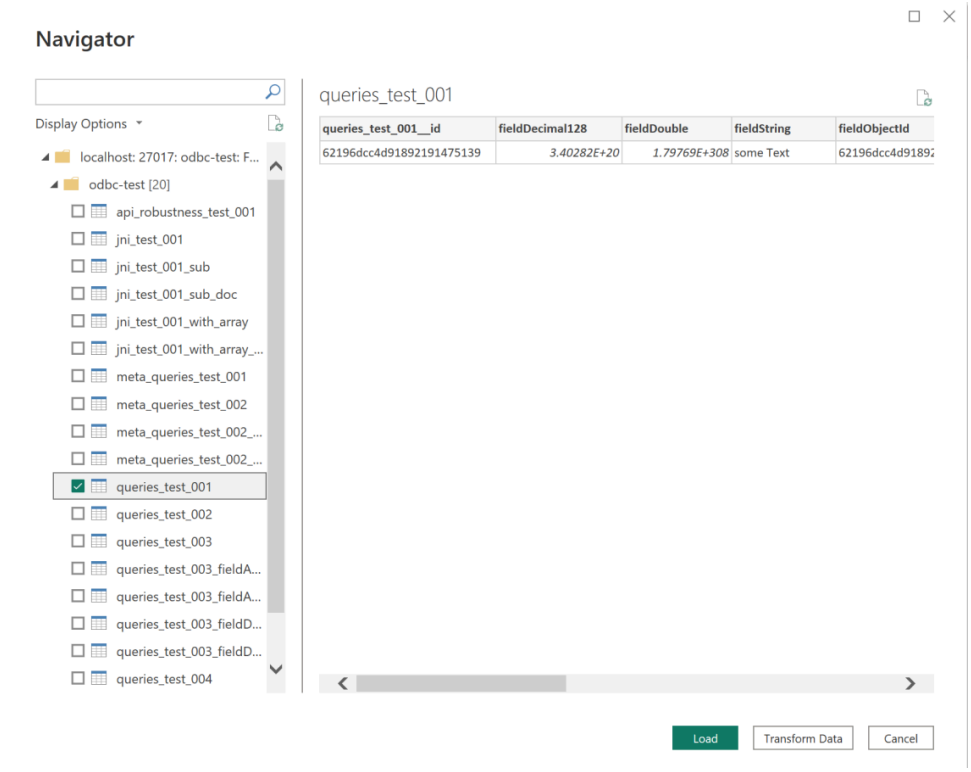


> **Note**
>
> 非常に大きなデータセットを使用している場合は、すべてのデータをインポートするのに時間がかかることがあります。

4. このデータソースに初めて接続する場合は、認証タイプを選択し、プロンプトが表示されたら認証情報を入力します。次に、Connect を選択します。



5. Navigator ダイアログで、必要なデータベーステーブルを選択し、ロードを選択してデータをロードするか、データを変換してデータの変換を続行します。



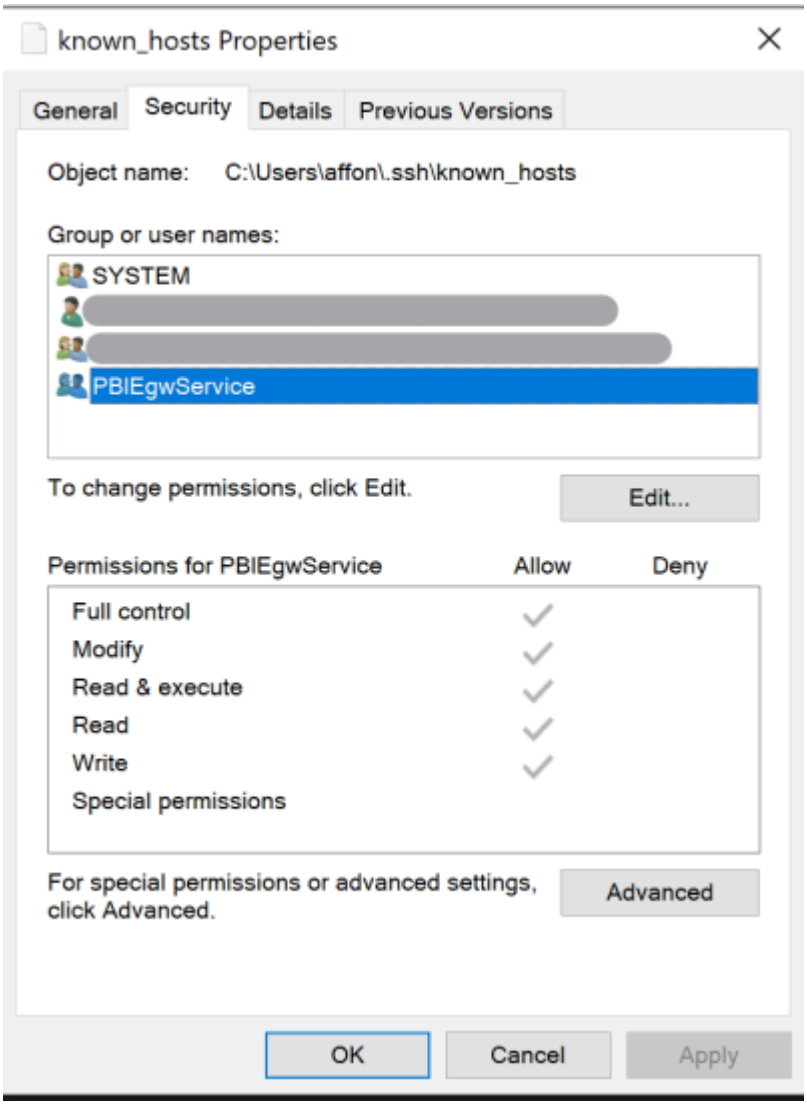
> **Note**
>
> 接続すると、データソースの設定が保存されます。設定を変更するには、[データ変換] > [データソース設定] を選択します。

Microsoft Power BI ゲートウェイの設定

前提条件:

- カスタムコネクタが Power BI ゲートウェイで動作することを確認します。
- Power BI ゲートウェイがインストールされているマシンの [システム] タブにある ODBC データソースに ODBC DSN が作成されていることを確認します。

内部 SSH トンネル機能を使用している場合は、Power BI サービスアカウントがアクセスできる場所に `known_hosts` ファイルを配置する必要があります。



> **Note**
>
> これは、認証機関 (CA) 証明書ファイル (pem ファイル) など、Amazon DocumentDB クラスターへの接続を確立するために必要なすべてのファイルにも当てはまります。

自動スキーマ生成

ODBC ドライバーは JNI (Java ネイティブインターフェイス) 経由で Amazon DocumentDB JDBC ドライバーを利用しているため、自動スキーマ生成機能は JDBC ドライバーでも同様に機能します。自動スキーマ生成の詳細については、「」を参照してくださいJDBC スキーマの自動生成。ODBC ドライバーアーキテクチャの詳細については、GitHub のAmazon DocumentDB ODBC ドライバーアーキテクチャ」を参照してください。

SQL サポートと制限

Amazon DocumentDB ODBC ドライバーは、SQL-92 のサブセットといくつかの一般的な拡張を Support (サポート) する読み取り専用ドライバーです。詳細については、ODBC のサポートと制限に関するドキュメントを参照してください。

トラブルシューティング

Amazon DocumentDB JDBC ドライバーの使用に問題がある場合は、「トラブルシューティングガイド 」を参照してください。

Amazon DocumentDB を使用したプログラム

このサービスは JSON スキーマ検証をサポートしています。

トピック

- Amazon DocumentDB Java プログラミングガイド
- JSON スキーマ検証の使用

Amazon DocumentDB Java プログラミングガイド

この包括的なガイドでは、MongoDB の Java ドライバーを使用して Amazon DocumentDB を操作するための詳細なウォークスルーを提供し、データベースの運用と管理の重要な側面について説明します。

トピック

- 序章
- 前提条件
- このガイドのデータモデル
- MongoDB Java ドライバーを使用した Amazon DocumentDB への接続
- Java を使用した Amazon DocumentDB での CRUD オペレーションの実行
- Java による Amazon DocumentDB でのインデックス管理
- Amazon DocumentDB と Java を使用したイベント駆動型プログラミング

序章

このガイドは接続から始まり、MongoDB Java ドライバーを使用して DocumentDB クラスターへの安全な接続を確立する方法について説明します。接続文字列コンポーネント、SSL/TLS 実装、IAM 認証や接続プーリングなどのさまざまな接続オプション、堅牢なエラー処理戦略について詳しく説明します。

CRUD (作成、読み取り、更新、削除) オペレーションセクションでは、このガイドでドキュメントの操作について詳しく説明し、単一オペレーションと一括オペレーションの両方を使用してドキュメントの作成、読み取り、更新、削除を行う方法を示します。ここでは、フィルター、クエリ、およびさまざまなオペレーションオプションの使用について説明し、エラー処理のベストプラクティスと信頼性を向上させるための再試行ロジックの実装に焦点を当てています。このガイドでは、単一フィールドインデックス、複合インデックス、スパースインデックス、テキストインデックスなど、さまざまなインデックスタイプの作成とメンテナンスについて詳しく説明しています。適切なインデックス選択と、クエリ実行プランを分析するための `explain()` 関数の使用を通じてクエリパフォーマンスを最適化する方法について説明します。

最後のセクションでは、Amazon DocumentDB の変更ストリームを使用したイベント駆動型プログラミングに焦点を当て、Java アプリケーションでリアルタイムのデータ変更モニタリングを実装する方法を示します。変更ストリームカーソルの実装、継続的オペレーションの再開トークンの処理、履歴データ処理の時間ベースのオペレーションについて説明します。ガイド全体で、実用的なコード例とベストプラクティスが提供されるため、Amazon DocumentDB で堅牢な Java アプリケーションを構築する際の貴重なリソースとなります。

前提条件

作業を開始する前に、次の項目があることを確認します。

- Amazon DocumentDB クラスター AWS アカウント が設定された 。クラスターのセットアップについては、 AWS データベースブログの「Amazon EC2 を使用した Amazon Amazon DocumentDB の開始方法」を参照してください。
- Java Development Kit (JDK) がインストールされている (このガイドでは Amazon Corretto 21 を使用します)。
- 依存関係管理のための Maven。

このガイドのデータモデル

このガイドのすべてのサンプルコードは、「Restaurants」コレクションを持つ「ProgGuideData」テストデータベースへの接続を前提としています。このガイドのすべてのサンプルコードは、レストラン出品システムで動作します。以下は、このシステムのドキュメントの例です。

```
{
    "_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
```

```
    }
}
```

CRUD、インデックス管理、イベント駆動型プログラミングを示すすべてのコードサンプルは、MongoClient オブジェクト dbClient、MongoDatabase オブジェクト connectionDB、および MongoCollection オブジェクト collection があることを前提としています。

> **Note**
>
> このガイドのすべてのコード例は、MongoDB Java ドライバーバージョン 5.3.0 でテストされています。

MongoDB Java ドライバーを使用した Amazon DocumentDB への接続

このセクションでは、Java ドライバーを使用して Amazon DocumentDB に接続するためのステップバイステップのガイドを提供します。これにより、DocumentDB を Java アプリケーションへの統合を開始できるようになります。

トピック

- ステップ 1: プロジェクトをセットアップする
- ステップ 2: 接続文字列を作成する
- ステップ 3: 接続コードを書き込む
- ステップ 4: 接続例外を処理する
- ステップ 5: コードの実行
- 接続のベストプラクティス

ステップ 1: プロジェクトをセットアップする

1. Maven を使用して、Java プロジェクトを作成します。

   ```
   mvn archetype:generate -DgroupId=com.docdb.guide -DartifactId=my-docdb-project -
   DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
   ```

2. MongoDB Java ドライバーをプロジェクトの依存関係として「pom.xml」ファイルに追加します。

   ```
   <dependency>
   ```

```
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-sync</artifactId>
    <version>5.3.0</version>
</dependency>
```

ステップ 2: 接続文字列を作成する

Amazon DocumentDB 接続文字列は、アプリケーションと DocumentDB クラスター間の接続を確立するために不可欠です。この文字列は、クラスターエンドポイント、ポート、認証の詳細、さまざまな接続オプションなどの重要な情報をカプセル化します。DocumentDB 接続文字列を構築するには、通常、基本的な形式から始めます。

```
"mongodb://username:password@cluster-endpoint:port/?[connection options]"
```

「username」と「password」を実際の認証情報に置き換える必要があります。クラスターのエンドポイントとポート番号は、 AWS マネジメントコンソール だけでなく、 からも確認できます AWS CLI。クラスターのクラスターエンドポイントを見つけるには、「クラスターのエンドポイントの検索」を参照してください。DocumentDB のデフォルトのポートは、27017 です。

接続文字列の例

- 転送中の暗号化を使用して DocumentDB に接続し、読み取りリクエストがリードレプリカに送信され、プライマリに書き込まれることを確認します。

```
"mongodb://username:password@cluster-endpoint:27017/?tls=true&
   tlsCAFile=global-bundle.pem&
   readPreference=secondaryPreferred&
   retryWrites=false"
```

- IAM 認証を使用して DocumentDB に接続する:

```
"mongodb://cluster-endpoint:27017/?tls=true&
   tlsCAFile=global-bundle.pem&
   readPreference=secondaryPreferred&
   retryWrites=false&
   authSource=%24external&
   authMechanism=MONGODB-AWS"
```

接続文字列で使用できるさまざまなオプションは次のとおりです。

- TLS 証明書
- リードレプリカからの読み取り
- ライトコンサーンとジャーナリング
- RetryWrites
- IAM 認証
- 接続プール
- 接続タイムアウトパラメータ

TLS 証明書

`tls=true|false` — このオプションは Transport Layer Security (TLS) を有効または無効にします。デフォルトでは、転送中の暗号化は Amazon DocumentDB クラスターで有効になっているため、クラスターレベルで TLS が無効になっていない限り、このオプションの値は `true` である必要があります。

TLS を使用する場合、コードは DocumentDB クラスターへの接続を作成するときに SSL 証明書を提供する必要があります。クラスターへの安全な接続に必要な証明書をダウンロードします: `global-bundle.pem`。`global-bundle.pem` ファイルを使用するには 2 つの方法があります。

- オプション 1 — `global-bundle.pem` ファイルからすべての証明書を抽出し、Java の keytool を使用して、後でコードで使用できる `.jks` ファイルに保存します。これを行う方法を示すスクリプトについては、TLS が有効な場合の接続 の Java タブを参照してください。
- オプション 2 — `global-bundle.pem` ファイルをコードに動的に追加し、インメモリキーストアを構築し、`SSLContext` を使用して接続の一部として証明書を提供します。

リードレプリカからの読み取り

`replicaSet=rs0&readPreference=secondaryPreferred` — これら 2 つのオプションを指定すると、すべての読み取りリクエストがリードレプリカにルーティングされ、書き込みリクエストがプライマリインスタンスにルーティングされます。接続文字列で `replicaSet=rs0` を使用すると、MongoDB ドライバーはクラスタートポロジの自動更新ビューを維持できるため、インスタンスの追加または削除時にアプリケーションが現在のノード設定の可視性を維持できます。これらのオプションを提供しない場合、または `readPreference=primary` を指定しない場合、すべての読み取りと書き込みがプライマリインスタンスに送信されます。`readPreference` のオプションについては、「読み込み設定のオプション」を参照してください。

ライトコンサーンとジャーナリング

書き込みの懸念は、書き込みオペレーションのためにデータベースからリクエストされた確認のレベルを決定します。MongoDB ドライバーには、書き込みの懸念ファイルとジャーナルファイルを調整するオプションがあります。Amazon DocumentDB では、書き込みの懸念とジャーナルを設定することは想定されておらず、`w` と `j` (`writeConcern` と `journal`) に送信された値は無視されます。DocumentDB は常に `writeConcern: majority` および `journal: true` でデータを書き込みます。そのため、クライアントに確認を送信する前に、書き込みは大部分のノードに永続的に記録されます。

RetryWrites

`retryWrites` — エンジンバージョン 8.0.2 以降、Amazon DocumentDB は再試行可能な書き込みをサポートしています。有効に`retryWrites=true`するには を設定します。8.0.2 より前のエンジンバージョンでは、この属性を に設定しますfalse。詳細については、「Amazon DocumentDB での再試行可能な書き込み」を参照してください。

IAM 認証

`authSource=%24external` および **`authMechanism=MONGODB-AWS`** — この 2 つのパラメータは、 を使用した認証に使用されます AWS Identity and Access Management。IAM 認証は現在、インスタンスベースのクラスターバージョン 5.0 でのみ使用できます。詳細については、「IAM ID を使用した認証」を参照してください。

接続プール

これらのオプションは、接続プーリングで使用できます。

- **`maxPoolSize`** — プールに作成できる接続の最大数を設定します。すべての接続が使用されていて、新しいリクエストが入ると、接続が利用可能になるまで待機します。MongoDB Java ドライバーのデフォルトは 100 です。
- **`minPoolSize`** — プールで常に維持する必要がある接続の最小数を示します。MongoDB Java ドライバーのデフォルトは 0 です。
- **`maxIdleTimeMS`** — 接続を、閉じたり削除したりする前に、プールでアイドル状態のままにできる時間を決定します。MongoDB Java ドライバーのデフォルトは 100 ミリ秒です。
- **`waitQueueTimeoutMS`** — プールが最大サイズになったときに、接続が利用可能になるまでスレッドが待機する時間を設定します。この時間内に接続が利用可能にならない場合は、例外がスローされます。MongoDB Java ドライバーのデフォルト値は 120,000 ミリ秒 (2 分) です。

接続タイムアウトパラメータ

タイムアウトは、オペレーションまたは接続の試行が失敗したと見なされるまでにかかる時間を制限するメカニズムです。次のタイムアウトパラメータは、無期限の待機を防ぎ、リソース割り当てを管理するために使用できます。

- **connectTimeoutMS** — ドライバーがクラスターへの接続を確立するのを待機する時間を設定します。デフォルトは 10,000 ミリ秒 (10 秒) です。
- **socketTimeoutMS** — ドライバーがサーバーからの非書き込みオペレーションのレスポンスを待機する時間を指定します。デフォルトは 0 (タイムアウトまたは無限なし) です。
- **serverSelectionTimeoutMS** — ドライバーがクラスター内の使用可能なサーバーを見つけるまで待機する時間を指定します。この設定のデフォルト値は 30 秒で、フェイルオーバー中に新しいプライマリインスタンスを選択するのに十分です。

ステップ 3: 接続コードを書き込む

次のコード例は、Amazon DocumentDB への TLS 接続を行う方法を示しています。

- Java の KeyStore および SSLContext> オブジェクトが作成されます。
- また、ConnectionString オブジェクトに渡すことで MongoClientSettings オブジェクトを作成します。TLS 接続を行うには、 MongoClientSettings オブジェクトを使用して connectionstring と sslcontext をバインドする必要があります。
- MongoClients を使用すると、 MongoClient オブジェクトが取得されます。

```
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);
    }
}
```

ステップ 4: 接続例外を処理する

Java アプリケーションで DocumentDB を使用する場合、堅牢で信頼性の高いデータベースオペレーションを維持するには、接続例外の処理が不可欠です。これらの例外を適切に管理することで、問題を迅速に診断できるだけでなく、アプリケーションが一時的なネットワーク中断やサーバーの非可用性を正常に処理できるため、安定性とユーザーエクスペリエンスが向上します。接続の確立に関連する重要な例外には、次のようなものがあります。

- **MongoException** — 一般的な例外であり、より具体的な例外の対象ではないさまざまなシナリオで発行される場合があります。これは一般的なすべての MongoDB 例外のキャッチであるため、この例外は他のすべての特定の例外の後に処理されることを確認してください。
- **MongoTimeoutException** — オペレーションがタイムアウトしたときに発行されます。例えば、存在しないクラスターエンドポイントのクエリなどです。
- **MongoSocketException** — ネットワーク関連の問題に対して発行されます。例えば、オペレーション中の突然のネットワーク切断などです。

- **MongoSecurityException** — 認証が失敗した場合に発行されます。例えば、誤った認証情報での接続などです。
- **MongoConfigurationException** — クライアント設定にエラーがある場合に発行されます。例えば、無効な接続文字列の使用などです。

ステップ 5: コードの実行

次のコードサンプルは、Amazon DocumentDB 接続を作成し、すべてのデータベースを出力します。

```
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);
    }
}
```

接続のベストプラクティス

以下は、MongoDB Java ドライバーを使用して Amazon DocumentDB に接続するときに考慮すべきベストプラクティスです。

- クライアントにリソースを解放してもらう必要がなくなった場合は、必ず `MongoClient` を閉じてください。
- 例外を適切に処理し、適切なエラーログ記録を実装します。
- 環境変数または を使用して AWS Secrets Manager 、ユーザー名やパスワードなどの機密情報を保存します。

Java を使用した Amazon DocumentDB での CRUD オペレーションの実行

このセクションでは、MongoDB Java ドライバーを使用して Amazon DocumentDB で CRUD (作成、読み取り、更新、削除) オペレーションを実行する方法について説明します。

トピック

- DocumentDB コレクションでのドキュメントの作成と挿入
- DocumentDB コレクションからのデータの読み取りと取得
- DocumentDB コレクション内の既存のドキュメントの更新
- DocumentDB コレクションからのドキュメントの削除
- 再試行ロジックによるエラー処理

DocumentDB コレクションでのドキュメントの作成と挿入

Amazon DocumentDB にドキュメントを挿入すると、コレクションに新しいデータを追加できるようになります。挿入を実行するには、ニーズと使用しているデータの量に応じて、いくつかの方法があります。個々のドキュメントをコレクションに挿入するための最も基本的なメソッドは `insertOne()` です。一度に複数のドキュメントを挿入するには、`insertMany()` メソッドを使用できます。これにより、ドキュメントの配列を 1 回のオペレーションで追加できます。DocumentDB コレクションに多くのドキュメントを挿入するもう 1 つのメソッドは `bulkWrite()` です。このガイドでは、DocumentDB コレクションでドキュメントを作成するためのこれらのすべてのメソッドについて説明します。

`insertOne()`

まず、個々のドキュメントを Amazon DocumentDB コレクションに挿入する方法を見てみましょう。単一のドキュメントを挿入するには、`insertOne()` メソッドを使用します。このメソッドは、挿入のために BsonDocument を取得し、新しい挿入されたドキュメントのオブジェクト ID を取得するために使用できる `InsertOneResult` オブジェクトを返します。以下のコード例は、コレクションに 1 つのレストランドキュメントを挿入する方法を示しています。

```
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;
}
```

insertOne() を使用する場合は、適切なエラー処理を必ず含めてください。例えば、上記のコードでは、「restaurantId」には一意のインデックスがあるため、このコードを再度実行すると、次の MongoWriteException が生成されます。

```
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()

コレクションに多くのドキュメントを挿入する主なメソッドは insertMany() と bulkWrite() です。

insertMany() メソッドは、1 回のオペレーションで複数のドキュメントを挿入する最も簡単な方法です。ドキュメントのリストを受け入れ、コレクションに挿入します。このメソッドは、互いに独立しており、特別な処理や混合オペレーションを必要としない新しいドキュメントのバッチを挿入する場合に最適です。次のコードは、ファイルから JSON ドキュメントを読み取ってコレクションに挿入する方法を示しています。insertMany() 関数は、挿入されたすべてのドキュメントの ID を取得するために使用できる InsertManyResultInsertManyResult オブジェクトを返します。

```
// 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() メソッドを使用すると、複数の書き込みオペレーション (挿入、更新、削除) を 1 つのバッチで実行できます。bulkWrite() は、一部のドキュメントを挿入して他のドキュメントを更新するなど、1 つのバッチでさまざまなタイプのオペレーションを実行する必要がある場合に使用できます。bulkWrite() は、順序付きと順序なしの 2 種類のバッチ書き込みをサポートしています。

- 順序付けられたオペレーション — (デフォルト) Amazon DocumentDB は書き込みオペレーションを順番に処理し、最初に発生したエラーで停止します。これは、後のオペレーションが以前のオペレーションに依存する場合など、オペレーションの順序が重要な場合に役立ちます。ただし、順序付けられたオペレーションは、通常、順序付けられていないオペレーションよりも遅くなります。順序付けられたオペレーションでは、バッチが最初のエラーで停止し、一部のオペレーションが未処理のままになる可能性があるケースに対処する必要があります。
- 順序付けされていないオペレーション — Amazon DocumentDB が挿入をデータベース内の 1 回の実行として処理できるようにします。1 つのドキュメントでエラーが発生した場合、オペレーションは残りのドキュメントで続行されます。これは、大量のデータを挿入していて、キーの重複が原因で一部のドキュメントが失敗する可能性のあるデータ移行や一括インポートなど、一部の障害を許容できる場合に特に便利です。順序付けされていないオペレーションでは、一部のオペレーションが成功し、他のオペレーションが失敗する部分的な成功シナリオに対処する必要があります。

bulkWrite() メソッドを使用する場合、必須のクラスがいくつかあります。まず、WriteModel クラスはすべての書き込みオペレーションのベースクラスとして機能し、InsertOneModel、UpdateOneModel、UpdateManyModel、DeleteOneModel、DeleteManyMod などの特定の実装で、さまざまなタイプのオペレーションを処理します。

BulkWriteOptions クラスは、順序付き/順序なしの実行の設定やドキュメントの検証のバイパスなど、一括オペレーションの動作を設定するために必要です。BulkWriteResult クラスは、挿入、更新、および削除されたドキュメントの数など、実行結果に関する詳細情報を提供します。

エラー処理では、MongoBulkWriteException クラスには一括オペレーション中の障害に関する情報が含まれているため、 クラスは重要です。一方、BulkWriteError クラスは個々のオペレーションの障害に関する特定の詳細を提供します。次のコードは、1 つの bulkWrite() メソッド呼び出しの実行中に、ドキュメントのリストを挿入し、1 つのドキュメントを更新および削除する

例を示しています。このコードは、BulkWriteOptions および BulkWriteResult の操作方法と、bulkWrite() オペレーションの適切なエラー処理も示します。

```
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());
}
```

再試行可能な書き込み

MongoDB とは異なり、Amazon DocumentDB は再試行可能な書き込みをサポートしていません。そのため、特にネットワークの問題や一時的なサービス利用不能を処理するために、アプリケーションにカスタム再試行ロジックを実装する必要があります。適切に実装された再試行戦略では、通常、再試行間の遅延を増やし、再試行の合計数を制限します。エラー処理を使用して再試行ロジックを構築するコードサンプルについては、以下の 再試行ロジックによるエラー処理 を参照してください。

DocumentDB コレクションからのデータの読み取りと取得

Amazon DocumentDB でのドキュメントのクエリは、データを正確に取得して操作できるいくつかの主要なコンポーネントを中心に展開されます。`find()` メソッドは、MongoDB Java ドライバーの基本的なクエリ API です。これにより、フィルタリング、ソート、結果の射影のための多数のオプションを使用して、複雑なデータ取得が可能になります。`find()` メソッドに加えて、`Filters` と `FindIterable` は、MongoDB Java ドライバーでのクエリオペレーションの構成要素を提供する他の 2 つの基本的なコンポーネントです。

`Filters` クラスは、クエリフィルターを構築するための流暢な API を提供する MongoDB Java ドライバーのユーティリティクラスです。このクラスは、さまざまなクエリ条件を表す `Bson` オブジェクトのインスタンスを作成する静的ファクトリメソッドを提供します。最も一般的に使用されるメソッドには、等価比較では `eq()`、数値比較では `gt()`、`lt()`、`gte()`、`lte()`、複数の条件の組み合わせでは `and()`、`or()`、配列メンバーシップテストでは `in()`、`nin()`、パターンマッチングでは `regex()` が含まれます。クラスは、タイプセーフであるように設計されており、raw ドキュメントベースのクエリと比較してコンパイル時のチェックに優れているため、Java アプリケーションで DocumentDB クエリを構築するための推奨アプローチです。エラー処理は堅牢で、無効なフィルター構造には明確な例外がスローされます。

`FindIterable` は、 `find()` メソッドの結果を処理するように設計された特殊なインターフェイスです。クエリ実行を改良および制御するための豊富なメソッドのセットを提供し、メソッド連鎖のための流暢な API を提供します。インターフェイスには、返されるドキュメントの数の制限では `limit()`、ページ分割では `skip()`、結果の順序付けでは `sort()`、特定のフィールドの選択では `projection()`、インデックスの選択では `hint()` など、必須のクエリ変更メソッドが含まれています。`FindIterable` のバッチ、スキップ、および制限オペレーションは、データベースからドキュメントを取得して処理する方法を制御するのに役立つ重要なページ分割およびデータ管理ツールです。

バッチ処理 (batchSize) は、1 回のネットワークラウンドトリップで DocumentDB がクライアントに返すドキュメントの数を制御します。バッチサイズを設定すると、DocumentDB は一致するすべてのドキュメントを一度に返すのではなく、指定されたバッチサイズのグループで返します。

スキップを使用すると、結果の開始点をオフセットできます。基本的には、一致を返す前に、指定した数のドキュメントをスキップするように DocumentDB に指示します。例えば、skip(20) は一致する最初の 20 個のドキュメントをバイパスします。これは、後続の結果ページを取得するページ分割シナリオで一般的に使用されます。

制限は、クエリから返すことができるドキュメントの総数を制限します。limit(n) を指定すると、データベースにより多くの一致がある場合でも、DocumentDB は「n」ドキュメントを返した後でドキュメントの返しを停止します。

FindIterable は、Amazon DocumentDB からドキュメントを取得するときに、イテレーターパターンとカーソルパターンの両方をサポートします。イテレーターとして FindIterable を使用する利点は、ドキュメントの遅延ロードを許可し、アプリケーションから要求された場合にのみドキュメントを取得することです。イテレーターを使用するもう 1 つの利点は、クラスターへの接続を維持する責任がないため、接続を明示的に閉じる必要がないことです。

FindIterable は、Amazon DocumentDB クエリを操作するときにカーソルパターンを使用できるようにする MongoCursor のサポートも提供します。MongoCursor は、データベースオペレーションとリソース管理を制御する MongoDB Java ドライバー固有の実装です。AutoCloseable インターフェイスを実装しているため、try-with-resources ブロックによる明示的なリソース管理が可能になります。これは、データベース接続を適切に閉じ、サーバーリソースを解放するために不可欠です。デフォルトでは、カーソルは 10 分でタイムアウトし、DocumentDB はこのタイムアウト動作を変更するオプションを提供しません。バッチ処理されたデータを使用する場合は、カーソルがタイムアウトする前に、必ず次のデータのバッチを取得してください。MongoCursor を使用する際の 1 つの重要な考慮事項は、リソースリークを防ぐために明示的な閉鎖が必要であることです。

このセクションでは、find()、Filters、および FindIterable の例をいくつか示します。

次のコード例は、find() を使用して、「restaurantId」フィールドを使用して単一のドキュメントを取得する方法を示しています。

```
Document filter = new Document("restaurantId", "REST-21G145");
Document result = collection.find(filter).first();
```

Filters を使用するとコンパイル時のエラーチェックが向上しますが、Java ドライバーでは find() メソッドで直接 Bson フィルターを指定することもできます。次のコード例では、Bson ドキュメントを find() に渡します。

```
result = collection.find(new Document("$and", Arrays.asList(
    new Document("rating.totalReviews", new Document("$gt", 1000)),
    new Document("priceRange", "$$"))))
```

次のコード例は、find() で Filters クラスを使用するいくつかの例を示しています。

```
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));
```

次の例は、FindIterable オブジェクトで sort()、skip()、limit()、および batchSize() のオペレーションを連鎖する方法を示しています。これらのオペレーションがどのように提供されるかの順序は、クエリのパフォーマンスに影響します。ベストプラクティスとして、これらのオペレーションの順序は sort()、projection()、skip()、limit() および batchSize() である必要があります。

```
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);
```

次の例のコードは、FindIterable でのイテレーターを作成を示しています。Java の forEach コンストラクトを使用して、結果セットをトラバースします。

```
collection.find(Filters.eq("cuisine", "American")).forEach(doc ->
 System.out.println(doc.toJson()));
```

最後の find() コード例では、cursor() を使用してドキュメントを取得する方法を示しています。試行ブロックにカーソルが作成され、コードが試行ブロックを終了するとカーソルが閉じられます。

```
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
```

DocumentDB コレクション内の既存のドキュメントの更新

Amazon DocumentDB は、既存のドキュメントを変更し、存在しないときに新しいドキュメントを挿入するための柔軟で強力なメカニズムを提供します。MongoDB Java ドライバーには、1 つのドキュメントの更新のための `updateOne()`、複数のドキュメントの更新のための `updateMany()`、完全なドキュメント置換のための `replaceOne()` など、複数の更新メソッドが用意されています。これらの 3 つのメソッドに加えて、`Updates`、`UpdateOptions`、`UpdateResult` は、MongoDB Java ドライバーでの更新オペレーションの構成要素を提供するその他の基本的なコンポーネントです。

MongoDB Java ドライバーの `Updates` クラスは、更新演算子を作成するための静的ファクトリメソッドを提供するユーティリティクラスです。これは、タイプセーフで読み取り可能な方法で更新オペレーションを構築するためのプライマリビルダーとして機能します。`set()`、`unset()`、`inc()` などの基本的なメソッドでは、ドキュメントを直接変更できます。このクラスの能力は、複数の更新オペレーションをアトミックに実行できる `Updates.combine()` メソッドを使用して複数のオペレーションを組み合わせると明らかになり、データ整合性が確保されます。

`UpdateOptions` は、ドキュメント更新オペレーションに不可欠なカスタマイズ機能を提供する MongoDB の Java ドライバーの強力な設定クラスです。このクラスの 2 つの重要な点は、更新オペレーションのアップサートと配列フィルターのサポートを提供することです。`upsert(true)` を介して有効になっているアップサート機能を使用すると、更新オペレーション中に一致するドキュメントが見つからなかった場合に、新しいドキュメントを作成できます。`arrayFilters()` を通じて、更新オペレーションは特定の基準を満たす配列要素を正確に更新できます。

MongoDB の Java ドライバーの `UpdateResult` は、更新オペレーションの結果を詳述するフィードバックメカニズムを提供します。このクラスは、更新基準に一致するドキュメントの数 (`matchedCount`)、実際に変更されたドキュメントの数 (`modifiedCount`)、およびアップサートされたドキュメントに関する情報 (`upsertedId`) の 3 つの主要なメトリクスをカプセル化します。これらのメトリクスを理解することは、適切なエラー処理、更新オペレーションの検証、アプリケーションのデータ整合性の維持に不可欠です。

1 つのドキュメントを更新して置き換える

DocumentDB では、updateOne() メソッドを使用して 1 つのドキュメントを更新できます。このメソッドは、通常は Filters クラスによって提供されるフィルターパラメータを使用して、更新するドキュメントを識別し、どのフィールド (複数可) を更新するかを決定する Update パラメータと、更新のさまざまなオプションを設定するオプションの UpdateOptions パラメータを使用します。updateOne( ) メソッドを使用すると、選択基準に一致する最初のドキュメントのみが更新されます。次のコード例では、1 つのドキュメントの 1 つのフィールドを更新します。

```
collection.updateOne(Filters.eq("restaurantId", "REST-Y2E9H5"),
    Updates.set("name", "Amazing Japanese sushi"));
```

1 つのドキュメント内の複数のフィールドを更新するには、次の例に示すように、Update.combine( ) で updateOne( ) を使用します。この例は、ドキュメント内の配列に項目を追加する方法も示しています。

```
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);
```

次のコード例は、データベース内のドキュメントを更新する方法を示しています。指定されたドキュメントが存在しない場合、オペレーションはそれを代わりに新しいドキュメントとして自動的に挿入します。このコードは、UpdateResult オブジェクトを介して利用可能なメトリクスの使用方法も示します。

```
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)");
}
```

次のコード例は、個々のフィールドを更新するのではなく、replaceOne() メソッドを使用して既存のドキュメントを新しいドキュメントと完全に置き換える方法を示しています。replaceOne() メソッドはドキュメント全体を上書きし、元の _id フィールドのみを保持します。複数のドキュメントがフィルター条件に一致する場合、最初に検出されたドキュメントのみが置き換えられます。

```
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());
```

複数のドキュメントの更新

コレクション内の複数のドキュメントを同時に更新する方法は 2 つあります。updateMany() メソッドを使用するか、bulkWrite() メソッドの UpdateManyModel を使用できます。updateMany() メソッドは、フィルターパラメータを使用して更新するドキュメントを選択し、Update パラメータを使用して更新するフィールドを識別し、オプションの UpdateOptions パラメータを使用して更新オプションを指定します。

次のコード例は、updateMany() メソッドの使用を示しています。

```
Bson filter = Filters.and(
    Filters.in("features", Arrays.asList("Private Dining")),
    Filters.eq("cuisine", "Thai"));
UpdateResult result1 = collection.updateMany(filter, Updates.set("priceRange", "$$$"));
```

次のコード例は、同じ更新を使用する bulkWrite() メソッドを示しています。

```
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());
```

DocumentDB コレクションからのドキュメントの削除

MongoDB Java ドライバーは、1 つのドキュメントの削除のために deleteOne()、および特定の条件に一致する複数のドキュメントの削除のために deleteMany() を提供します。更新と同様に、削除オペレーションは bulkWrite() メソッドでも使用できます。deleteOne() と deleteMany() はどちらも、削除されたドキュメントの数など、オペレーションの結果に関する情報を提供する DeleteResult オブジェクトを返します。deleteMany() を使用して複数のドキュメントを削除する例を次に示します。

```
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());
```

再試行ロジックによるエラー処理

Amazon DocumentDB の堅牢なエラー処理戦略では、エラーを再試行可能 (ネットワークタイムアウト、接続の問題など) と再試行不可能 (認証の失敗、無効なクエリなど) に分類する必要があります。再試行する必要があるエラーによるオペレーションの失敗については、各再試行間の遅延と最大再試行回数を実装する必要があります。CRUD オペレーションは、MongoException とそのサブクラスをキャッチする try-catch ブロックにある必要があります。さらに、運用上の可視性のためにエラーのモニタリングと ログ記録を含める必要があります。以下は、再試行エラー処理を実装する方法を示すサンプルコードです。

```
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);
    }
}
```

Java による Amazon DocumentDB でのインデックス管理

インデックスを使用すると、Amazon DocumentDB コレクションからデータを効率的に取得できます。インデックスがない場合、DocumentDB はコレクション内のすべてのドキュメントをスキャンして、特定のクエリを満たす結果を返す必要があります。このトピックでは、MongoDB Java ドライバーを使用してインデックスを作成、削除、および一覧表示する方法について説明します。また、クエリで特定のインデックスが使用されているかどうかを判断する方法と、特定のインデックスを使用するように Amazon DocumentDB にヒントを与える方法について説明します。

トピック

- Java によるインデックスの作成
- インデックスの削除
- インデックス選択の決定とインデックスヒントの提供

Amazon DocumentDB は、多くのタイプのインデックスをサポートしています。サポートされているすべてのインデックスの包括的な概要については、 AWS データベースブログのAmazon DocumentDB でインデックスを作成する方法」を参照してください。

Java によるインデックスの作成

MongoDB Java ドライバーを使用して Amazon DocumentDB でインデックスを作成するメカニズムには、runCommand() を通じたもの、および単一のインデックスでの createIndex() メソッドまたは複数のインデックスでの createIndexes() メソッドのいずれかによるものの 2 つがあります。createIndex() および createIndexes() メソッドを使用する理由の 1 つは、インデックス作成に関連する特定のエラーをキャッチすることで、より良いエラー処理を構築できることです。これらのメソッドを runCommand() で使用するもう 1 つの理由は、MongDB Java ドライバーがインデックスの作成と操作のための豊富なサポートクラスを提供することです。これらのサポートクラスは、createIndex() または createIndexes() メソッドを使用している場合にのみ使用できることに注意してください。3 つのサポートクラスがあります。

- **Indexes** — このクラスは、さまざまなタイプのインデックスを作成するための静的ファクトリメソッドを提供するユーティリティクラスとして機能します。これにより、複雑なインデックス定義を作成するプロセスが簡素化され、一般的に他のインデックス関連クラスと組み合わせて使用されます。
- **IndexModel** — これは、インデックスキー定義とそのオプションの両方をカプセル化する基本的なクラスです。これは、インデックスを作成する内容 (キー) とインデックスを作成する方法 (オプション) を組み合わせた完全なインデックス仕様を表します。このクラスは、createIndexes() メソッドに渡すことができるインデックス仕様のコレクションを定義できるため、複数のインデックスを同時に作成する場合に特に便利です。
- **IndexOptions** — これは、インデックスの動作をカスタマイズするための豊富なメソッドのセットを提供する包括的な設定クラスです。これには、一意のインデックス、スパースインデックス、有効期限 (TTL)、および部分フィルター式の設定が含まれます。メソッドチェーンを使用すると、バックグラウンドインデックスの構築や一意の制約など、複数のオプションを設定できます。

単一のインデックスを作成する

この例では、バックグラウンドで createIndex() メソッドを使用して単一のインデックスを作成する方法を示します。背景インデックスとフォアグラウンドインデックスの作成については、「」を参照してくださいインデックスビルドのタイプ。次のコード例では、IndexOptions を使用して、バックグラウンドで「unique_restaurantId_idx」という名前の一意のインデックスを作成します。その後、この IndexOptions オブジェクトは createIndex() メソッドに渡されます。

```
collection.createIndex(
    Indexes.ascending("restaurantId"),
    new IndexOptions()
        .unique(true)
```

```
        .name("unique_restaurantId_idx")
        .background(true));
```

複数のインデックスを作成する

この例では、createIndexes() メソッドを使用して複数のインデックスを作成します。まず、IndexModel オブジェクトを使用して各インデックスのオプションを構築し、次に IndexModel オブジェクトのリストを createIndexes() メソッドに渡します。次のコード例は、Indexes ユーティリティクラスを使用して複合インデックスを作成する方法を示しています。このクラスは、昇順または降順のソート順序を使用してインデックスを作成するかどうかを指定するためにも使用されます。複数のインデックスを作成した後、listIndexes() メソッドを呼び出してインデックスの作成を検証します。

```
// 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()));
```

スパースインデックスと部分インデックスを作成する

この例では、インデックスのタイプごとに IndexModel を作成して、スパースインデックスと部分インデックスを作成する方法を示します。

```
// 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))));
```

テキストインデックスを作成する

この例では、テキストインデックスを作成する方法を示します。コレクションでは 1 つのテキストインデックスのみが許可されますが、1 つのテキストインデックスを複数のフィールドをカバーする複合インデックスにすることができます。テキストインデックスで複数のフィールドを使用する場合、インデックス内の各フィールドに重みを割り当てることもできます。配列フィールドのテキストインデックスは Amazon DocumentDB ではサポートされていません。複合テキストインデックスでは最大 30 個のフィールドを使用できますが、重みを割り当てることができるフィールドは 3 つだけです。

```
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());
```

`runCommand()` を使用してインデックスを作成する

Amazon DocumentDB は、インデックスの作成にかかる時間を短縮するための並列インデックス作成をサポートしています。並列インデックス作成は、複数の同時ワーカーを使用します。インデックスの作成に使用されるデフォルトのワーカーは 2 つです。詳細については、 AWS データベースブログの「Unlock the power of parallel indexing in Amazon DocumentDB」を参照してください。現在、MongDB Java ドライバーは、`createIndex()` または `createIndexes()` を使用しているときにワーカーオプションの指定をサポートしていないため、ワーカーを指定する唯一の方法は `runCommand` を使用することです。次のコード例は、`runCommand` を使用して、ワーカーを 4 つに増やすインデックスを作成する方法を示しています。

```
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);
```

インデックスの削除

MongoDB Java ドライバーは、さまざまなシナリオや好みに応じて、インデックスを削除する複数のメソッドを提供します。名前で、またはキー仕様でインデックスを削除するか、すべてのインデックスを一度に削除できます。メソッド `dropIndex()` と `dropIndexes()` をコレクションオブジェクトで呼び出して、インデックスを削除できます。名前でインデックスを削除する場合は、特に複合インデックスまたは自動生成されたインデックスの場合、常に直感的とは限らない正しいインデックス名を使用してください。存在しないインデックスを削除しようとすると、`MongoCommandException` になります。コレクション内でドキュメントの一意性が確保されるため、`default _id` インデックスを削除することはできません。

次のコード例は、インデックスが作成されたフィールド名を指定するか、すべてのインデックスを削除することで、インデックスを削除する方法を示しています。

```
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();
```

複数のキーを使用してインデックスを削除する場合は、指定されたすべてのキーを含む複合インデックスがあり、キーの順序が正しいことを確認してください。上記のインデックス作成サンプルコード例は、「cuisine」および 機能の複合キーを示しています。その複合キーを削除しようとしても、その順序が作成で使用された順序ではない場合、MongoCommnadException エラーが次のように発生します。

```
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);
}
```

次のエラーが表示されます。

```
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
```

インデックス選択の決定とインデックスヒントの提供

Amazon DocumentDB の説明機能を操作することは、クエリのパフォーマンスとインデックスの使用を理解するために不可欠です。クエリを実行するときは、explain() メソッドを追加して、どのインデックスが使用されているか (ある場合) など、クエリプランに関する詳細情報を取得できます。explain() 出力は、クエリ実行ステージ、調査されたドキュメントの数、および各ステージに

かかる時間に関するインサイトを提供します。この情報は、特定のインデックスが効果的に使用されているかどうか、またはクエリが別のインデックス構造から恩恵を受けることができるかどうかを識別するのに非常に役立ちます。

explain() メソッドは、find() メソッドと連鎖できます。explain() メソッドは、explain() によって返される詳細レベルを決定するオプションの ExplainVerbosity 列挙値を取ることができます。現時点では、EXECUTION_STATS および QUERY_PLANNER 列挙子のみが DocumentDB でサポートされています。次のコード例は、特定のクエリのクエリプランナーを取得する方法を示しています。

```
// 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());
```

クエリプランナーの詳細レベルには、次の JSON ドキュメントが返されます。

```
{
  "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
```

```
    }
  }
}
```

Amazon DocumentDB が特定のインデックスを使用するように影響または強制するには、いくつかのオプションがあります。hint() および hintString() メソッドでは、クエリに使用するインデックスを明示的に指定することで、クエリオプティマイザのデフォルトのインデックス選択動作を上書きできます。DocumentDB のクエリオプティマイザは通常、インデックスの選択に適切な選択を行いますが、歪んだデータを扱うときやインデックスのパフォーマンスをテストするときなど、特定のインデックスを hint() または hintString() で強制することが有益なシナリオがあります。

次のコード例では、上記のコードで実行されたのと同じクエリに対して複合インデックス「cuisine_features_idx」を強制的に使用します。

```
// 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));
```

Amazon DocumentDB と Java を使用したイベント駆動型プログラミング

Amazon DocumentDB のコンテキストにおけるイベント駆動型プログラミングは、データベースの変更が後続のビジネスロジックとプロセスをトリガーする主要なイベントジェネレーターとして機能する強力なアーキテクチャパターンを表します。DocumentDB コレクションでレコードが挿入、更新、または削除されると、これらの変更はさまざまなダウンストリームプロセス、通知、またはデータ同期タスクを自動的に開始するイベントとして機能します。このパターンは、複数のアプリケーションまたはサービスがデータ変更にリアルタイムで対応する必要がある最新の分散システムでは特に重要です。DocumentDB でイベント駆動型プログラミングを実装する主なメカニズムは、変更ストリームです。

> **Note**
>
> このガイドでは、作業中のコレクションで変更ストリームを有効にしていることを前提としています。コレクションで変更ストリームを有効にする方法については、「Amazon DocumentDB を用いた変更ストリームの使用」を参照してください。

Java アプリケーションからの変更ストリームの使用

MongoDB の Java ドライバーの watch() メソッドは、Amazon DocumentDB でリアルタイムのデータ変更をモニタリングするための主要なメカニズムです。watch() メソッドは、MongoClient、MongoDatabase、および MongoCollection オブジェクトによって呼び出すことができます。

watch() メソッドは、更新の完全なドキュメント検索、信頼性のための再開トークンとタイムスタンプの提供、変更をフィルタリングするためのパイプライン集約ステージなど、さまざまな設定オプションをサポートする ChangeStreamIterable のインスタンスを返します。

ChangeStreamIterable はコア Java インターフェイス Iterable を実装しており、forEach() で使用できます。forEach() を使用してイベントをキャプチャするには、変更されたイベントを処理する forEach() にコールバック関数を渡します。次のコードスニペットは、コレクションで変更ストリームを開いて変更イベントのモニタリングを開始する方法を示しています。

```
ChangeStreamIterable < Document > iterator = collection.watch();
iterator.forEach(event - > {
    System.out.println("Received a change: " + event);
});
```

すべての変更イベントを通過するもう 1 つの方法は、クラスターへの接続を維持し、発生時に新しい変更イベントを継続的に受信するカーソルを開くことです。変更ストリームカーソルを取得するには、ChangeStreamIterable オブジェクトの cursor() メソッドを使用します。次のコード例は、カーソルを使用して変更イベントをモニタリングする方法を示しています。

```
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch().cursor()) {
    System.out.println(cursor.tryNext());
}
```

ベストプラクティスとして、 try-with-resource ステートメントで MongoChangeStreamCursor を作成するか、カーソルを手動で閉じます。ChangeStreamIterable で cursor() メソッドを呼び出すと、ChangeStreamDocument オブジェクト上に作成された MongoChangeStreamCursor が返されます。

ChangeStreamDocument クラスは、ストリーム内の個々の変更イベントを表す重要なコンポーネントです。これには、オペレーションタイプ (挿入、更新、削除、置換)、ドキュメントキー、名前空間情報、および使用可能な場合の完全なドキュメントコンテンツなど、各変更に関する詳細情報が含

まれます。クラスは、変更のタイプを決定する getOperationType()、完全なドキュメント状態にアクセスする getFullDocument()、変更されたドキュメントを識別する getDocumentKey() など、変更イベントのさまざまな側面にアクセスするメソッドを提供します。

ChangeStreamDocument オブジェクトは、再開トークンと変更イベントという、時刻の 2 つの重要な情報を提供します。

DocumentDB 変更ストリームの再開トークンと時間ベースのオペレーションは、継続性を維持し、履歴データアクセスを管理するための重要なメカニズムを提供します。再開トークンは、変更イベントごとに生成される一意の識別子であり、切断または障害後にアプリケーションが特定のポイントから変更ストリーム処理を再開できるようにするブックマークとして機能します。変更ストリームカーソルを作成すると、resumeAfter() オプションを使用して以前に保存した再開トークンを使用できます。これにより、ストリームは最初から開始したり、イベントを失ったりするのではなく、中断した場所から続行できます。

変更ストリームの時間ベースのオペレーションは、変更イベントモニタリングの開始点を管理するためのさまざまなアプローチを提供します。startAtOperationTime() オプションを使用すると、特定のタイムスタンプ以降に発生した変更の監視を開始できます。これらの時間ベースの機能は、履歴データ処理、ポイントインタイムリカバリ、またはシステム間の同期を必要とするシナリオで特に役立ちます。

次のコード例は、挿入ドキュメントに関連付けられたイベントを取得し、再開トークンをキャプチャしてから、挿入イベントの後にイベントのモニタリングを開始するトークンを提供します。イベントは更新イベントに関連付けられ、更新が発生したクラスター時間を取得し、そのタイムスタンプをさらなる処理の開始点として使用します。

```
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());
  }
```

デフォルトで、変更の更新イベントには、ドキュメント全体は含まれず、加えられた変更のみが含まれます。更新した完全なドキュメントにアクセスする必要がある場合は、`ChangeStreamIterable` オブジェクトで `fullDocument()` メソッドを呼び出すことができます。更新イベントに対して完全なドキュメントが返されるように要求すると、変更ストリームへの呼び出しが行われた時点で存在するドキュメントが返されることに注意してください。

このメソッドは、`FullDocument` 列挙型をパラメータとして受け取ります。現在、Amazon DocumentDB は DEFAULT および UPDATE_LOOKUP 値のみをサポートしています。次のコードスニペットは、変更の監視を開始するときに、更新イベントの完全なドキュメントを要求する方法を示しています。

```
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch().fullDocument(FullDocument.UPDATE_LOOKUP).cursor())
```

JSON スキーマ検証の使用

`$jsonSchema` 評価クエリ演算子を使用すると、コレクションに挿入されているドキュメントを検証できます。

トピック

- JSON スキーマ検証の作成と使用
- サポートされるキーワード
- bypassDocumentValidation
- 制限事項

JSON スキーマ検証の作成と使用

スキーマ検証によるコレクションの作成

createCollection 操作ルールと検証ルールを含むコレクションを作成できます。これらの検証ルールは、Amazon DocumentDB ドキュメントの挿入または更新時に適用されます。以下は、従業員のコレクションの検証ルールを説明するコード例です。

```
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"
} )
```

有効なドキュメントを挿入します

次の例では、上記のスキーマ検証ルールに準拠するドキュメントを挿入します。

```
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})
```

無効なドキュメントを挿入します

次の例では、上記のスキーマ検証ルールに準拠していないドキュメントを挿入します。この例では、employeeId の値は文字列ではありません。

```
db.employees.insert({
    "name" : { "firstName" : "Carol" , "lastName" : "Smith"},
    "employeeId": 720 ,
    "salary": 1000.0
})
```

この例は、文書内の構文が正しくないことを示しています。

コレクションを変更する

collMod コマンドは、既存のコレクションの検証ルールを追加または変更するために使用されます。次の例では、給与フィールドを必須フィールドリストに追加しています。

```
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
      }
   }
} )
```

検証ルールが変更される前に追加されたドキュメントの対処

検証ルールが変更される前にコレクションに追加されたドキュメントの対処には、以下の `validationLevel` 修飾子を使用します。

- strict: すべての挿入と更新に検証ルールを適用します。
- moderate: 既存の有効なドキュメントに検証ルールを適用します。更新中、既存の無効なドキュメントはチェックされません。

次の例では、「employees」という名前のコレクションの検証ルールを更新した後に、給与フィールドが必須になります。次のドキュメントの更新は失敗します。

```
db.runCommand({
    update: "employees",
    updates: [{
        q: { "employeeId": "c721a" },
        u: { age: 25 , salary : 1000},
        upsert: true }]
})
```

Amazon DocumentDB は、次の出力を返します。

```
{
"n" : 0,
    "nModified" : 0,
    "writeErrors" : [
        {
"index" : 0,
            "code" : 121,
            "errmsg" : "Document failed validation"
        }
    ],
    "ok" : 1,
    "operationTime" : Timestamp(1234567890, 1)
}
```

検証レベルを moderate に更新すると、上記のドキュメントを正常に更新できるようになります。

```
db.runCommand({
    "collMod" : "employees",
    validationLevel : "moderate"
})

db.runCommand({
    update: "employees",
    updates: [{
        q: { "employeeId": "c721a" },
        u: { age: 25 , salary : 1000},
        upsert: true }]
})
```

Amazon DocumentDB は、次の出力を返します。

```
{
"n" : 1,
    "nModified" : 1,
    "ok" : 1,
    "operationTime" : Timestamp(1234567890, 1)
}
```

$jsonSchema を使用してドキュメントの取得

$jsonSchema 演算子は、JSON スキーマに一致するドキュメントをクエリするフィルタとして使用できます。これは最上位の演算子で、フィルタードキュメントに最上位フィールドとして含

めることも $and、$or、$nor などのクエリ演算子と一緒に使用することもできます。以下の例は、$jsonSchema を個別のフィルターとして、また他のフィルター演算子と併用する方法を示しています。

「employee」コレクションに挿入されたドキュメント:

```
{ "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 }
```

$jsonSchema 演算子のみでフィルタリングされたコレクション:

```
db.employees.find({
       $jsonSchema: { required: ["age"] } })
```

Amazon DocumentDB は、次の出力を返します。

```
{ "_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 }
```

$jsonSchema 演算子と別の演算子でフィルタリングされたコレクション:

```
db.employees.find({
       $or: [{ $jsonSchema: { required: ["age", "name"]}},
            { salary: { $lte:1000}}]});
```

Amazon DocumentDB は、次の出力を返します。

```
{ "_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 }
```

$jsonSchema 演算子でフィルタリングされ、集計フィルタでは $match でフィルタリングされたコレクション:

```
db.employees.aggregate(
    [{ $match: {
        $jsonSchema: {
            required: ["name", "employeeId"],
            properties: {"salary" :{"bsonType": "double"}}
        }
       }
    }]
)
```

Amazon DocumentDB は、次の出力を返します。

```
{
"_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
```

```
}
```

既存の検証ルールの表示

コレクションの既存の検証ルールを確認するには、以下を使用してください。

```
db.runCommand({
    listCollections: 1,
    filter: { name: 'employees' }
})
```

Amazon DocumentDB は、次の出力を返します。

```
{
    "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 は、$out 集約ステージでも検証ルールを保持します。

サポートされるキーワード

create および collMod コマンドでは以下のフィールドがサポートされています。

- **Validator**: $jsonSchem オペレーションをサポートしています。
- **ValidationLevel**: off、strict、moderate の値をサポートします。
- **ValidationAction**: error の値をサポートします。

$jsonSchema 演算子は以下のキーワードをサポートします。

- 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 は、次のコマンドとメソッドについて bypassDocumentValidation をサポートしています。

- insert
- update
- findAndModify
- aggregate コマンドおよび db.collection.aggregate() メソッドにおける $out ステージ

Amazon DocumentDB は、次のコマンドについては bypassDocumentValidation をサポートしていません。

- aggregate コマンドおよび db.collection.aggregate() メソッドにおける $merge
- mapReduce コマンドおよび db.collection.mapReduce() メソッド
- applyOps コマンド

制限事項

$jsonSchema の検証には以下の制限が適用されます。

- Amazon DocumentDB は、オペレーションが検証ルールに失敗すると、「ドキュメントの検証に失敗しました」というエラーを返します。

- Amazon DocumentDB Elastic クラスターは、`$jsonSchema` をサポートしていません。

Amazon DocumentDB でのクエリの概要

このセクションでは、Amazon DocumentDB でのクエリのすべての側面について説明します。

トピック

- ドキュメントのクエリ
- クエリプラン
- 結果の説明
- クエリプランナー v2
- クエリプランナー v3
- Amazon DocumentDB を使用した地理空間データのクエリ
- 部分インデックス
- Amazon DocumentDB でのテキスト検索の実行

ドキュメントのクエリ

ときどきオンラインストアのインベントリを検索し、販売商品を顧客が表示して購入できることを確認する必要があります。コレクションのクエリは、コレクションのすべてのドキュメントを対象とするか、特定の条件を満たすドキュメントのみを対象とするかにかかわらず、比較的簡単です。

ドキュメントのクエリを実行するには、`find()` オペレーションを使用します。この `find()` コマンドには、返すドキュメントを選択するために使用する条件を定義する単一のドキュメントパラメータがあります。`find()` からの出力は、改行のない 1 行のテキストとしてフォーマットされたドキュメントです。読みやすくするために出力ドキュメントをフォーマットするには、`find().pretty()` を使用します。このトピックのすべての例では、`.pretty()` を使用して出力を書式設定しています。

次のコードサンプルでは、前の 2 つの演習で`example`コレクションに挿入し`insertMany()`た 4 つのドキュメントを使用します。 `insertOne()`と は にありますドキュメントの追加。

トピック

- コレクション内のすべてのドキュメントの取得
- フィールドの値に一致するドキュメントの取得

- 埋め込みドキュメントに一致するドキュメントの取得
- 埋め込まれたドキュメントのフィールド値に一致するドキュメントの取得
- 配列に一致するドキュメントの取得
- 配列値に一致するドキュメントの取得
- 演算子を使用したドキュメントの取得

コレクション内のすべてのドキュメントの取得

コレクション内のすべてのドキュメントを取得するには、`find()` オペレーションで、空のクエリドキュメントを指定します。

以下のクエリは、`example` コレクション内のすべてのドキュメントを返します。

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

フィールドの値に一致するドキュメントの取得

フィールドおよび値に一致するすべてのドキュメントを取得するには、`find()` オペレーションで、一致するフィールドおよび値を識別するクエリドキュメントを指定します。

上記のドキュメントを使用して、このクエリは、「Item」フィールドが「Pen」に等しいすべてのドキュメントを返します。

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

埋め込みドキュメントに一致するドキュメントの取得

埋め込みドキュメントに一致するすべてのドキュメントを検索するには、`find()` オペレーションで、埋め込みドキュメントの名前と、その埋め込みドキュメントのすべてのフィールドおよび値を識別するクエリドキュメントを指定します。

埋め込みドキュメントに一致するには、そのドキュメントの埋め込みドキュメントの名前が、クエリで指定した名前と同じであることが必要です。さらに、埋め込みドキュメントのフィールドおよび値も、クエリで指定したものと同じであることが必要です。

次のクエリでは、「Poster Paint」ドキュメントのみ返ります。これは、「Pen」の値は「`OnHand`」や「`MinOnHand`」とは異なり、「Spray Paint」には、クエリドキュメントよりもフィールドが 1 つ (`OrderQnty`) 多いためです。

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

埋め込まれたドキュメントのフィールド値に一致するドキュメントの取得

埋め込みドキュメントに一致するすべてのドキュメントを検索するには、find() オペレーションで、埋め込みドキュメントの名前と、その埋め込みドキュメントのすべてのフィールドおよび値を識別するクエリドキュメントを指定します。

上記のドキュメントでは、以下のようにクエリで「ドット表記」を使用して埋め込みドキュメントと目的のフィールドを指定しています。埋め込みドキュメント内に他のフィールドがある場合でも、これらに一致するすべてのドキュメントが返されます。クエリは「Poster Paint」と「Spray Paint」を返します。それらはいずれも、指定したフィールドおよび値と一致するためです。

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

配列に一致するドキュメントの取得

配列に一致するすべてのドキュメントを検索するには、find() オペレーションで、目的の配列名とその配列内のすべての値を指定します。クエリは、配列値の名前および順序がクエリで指定したものと一致するすべてのドキュメントを返します。

以下のクエリは「Pen」のみを返します。「Poster Paint」では追加の色 (White) があり、「Spray Paint」では色の順序が異なるためです。

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

配列値に一致するドキュメントの取得

特定の属性配列を含むすべてのドキュメントを見つけるには、find() オペレーションで、目的の配列名と値を指定します。

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

上記のオペレーションは、すべてのドキュメントを返します。3 つのドキュメントのそれぞれで、Colors という配列があり、その配列のどこかに「Red」という値があるためです。値として「White」を指定すると、クエリからは「Poster Paint」のみが返されます。

演算子を使用したドキュメントの取得

以下のクエリは、「Inventory.OnHand」値が 50 未満のすべてのドキュメントを返します。

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

サポートされているクエリ演算子のリストについては、「クエリおよびプロジェクション演算子」を参照してください。

クエリプラン

クエリプラン**executionStats**の を表示するにはどうすればよいですか?

クエリの実行速度が予想よりも遅い理由を判断する場合は、クエリプランの executionStats の目的を理解しておくと役立ちます。executionStats は、特定のステージから返されたドキュメントの数 (nReturned)、各ステージで費やされた実行時間 (executionTimeMillisEstimate)、およびクエリプランの生成に要する時間 (planningTimeMillis) を提供します。次のクエリ例に示すように、 の出力から最適化作業に集中できるようにexecutionStats、クエリの最も時間のかかるステージを決定できます。executionStats パラメータは、現在 update コマンドと delete コマンドをサポートしていません。

> **Note**
>
> Amazon DocumentDB は、分散型で、耐障害性が高く、自己復旧ストレージシステムを利用する専用データベースエンジンで MongoDB 3.6 API をエミュレートしています。その結果、Amazon DocumentDB と MongoDB では、クエリプランと explain() の出力が異なる場合があります。クエリプランを制御する場合は、$hint 演算子を使用して優先インデックスの選択を強制できます。

次のように、explain() コマンドで改善対象のクエリを実行します。

```
db.runCommand({explain: {query document}}).
explain("executionStats").executionStats;
```

以下に示しているのは、オペレーションの例です。

```
db.fish.find({}).limit(2).explain("executionStats");
```

このオペレーションによる出力は、次のようになります。

```
{
    "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
}
```

上のクエリの executionStats のみを表示する場合は、次のコマンドを使用できます。小規模なコレクションで、パフォーマンスの向上がごくわずかである場合、Amazon DocumentDB クエリプロセッサはインデックスを使用しないことがあります。

```
db.fish.find({}).limit(2).explain("executionStats").executionStats;
```

クエリプランキャッシュ

パフォーマンスを最適化し、計画期間を短縮するために、Amazon DocumentDB はクエリプランを内部的にキャッシュしています。これにより、同じ形状のクエリは、キャッシュされたプランを使用して直接実行することができます。

ただし、このキャッシュにより、同じクエリでもランダムに遅延が発生することがあります。例えば、通常 1 秒で実行されるクエリが、時には 10 秒かかることがあります。これは、時間の経過とともに、リーダーインスタンスがクエリのさまざまな形状をキャッシュし、メモリを消費するためです。このランダムな速度低下が発生した場合、メモリを解放するために必要なアクションはありません。システムがメモリ使用量を管理し、メモリが一定のしきい値に達すると、自動的に解放されます。

結果の説明

Amazon DocumentDB は、クエリプランに関する情報を返すために、冗長モード queryPlanner をサポートしています。explain の結果は、オプティマイザによって選択されたクエリプランを、以下のような形式で返します。

```
{
   "queryPlanner" : {
      "plannerVersion" : <int>,
      "namespace" : <string>,
      "winningPlan" : {
         "stage" : <STAGE1>,
         ...
         "inputStage" : {
            "stage" : <STAGE2>,
            ...
            "inputStage" : {
```

```
                ...
            }
         }
      }
   }
}
```

以下のセクションでは、一般的な explain の結果を定義します。

トピック

- スキャンとフィルタのステージ
- インデックス公差
- インデックス結合
- 複数のインデックス交差/結合
- 複合インデックス
- ソートステージ
- グループステージ

スキャンとフィルタのステージ

オプティマイザは、以下のいずれかのスキャンのいずれかを選択します。

COLSCAN

このステージは、順次収集スキャンです。

```
{
    "stage" : "COLLSCAN"
}
```

IXSCAN

このステージでは、インデックスキーをスキャンします。オプティマイザはこのステージ内でドキュメントを取得することがあり、その結果、後で FETCH ステージが追加されることがあります。

```
db.foo.find({"a": 1})
{
```

```
    "stage" : "IXSCAN",
    "direction" : "forward",
    "indexName" : <idx_name>
}
```

FETCH

オプティマイザが IXSCAN 以外のステージでドキュメントを取得した場合、その結果には FETCH ステージが含まれます。例えば、上記の IXSCAN クエリは、FETCH と IXSCAN ステージを組み合わせた結果になる可能性があります。

```
db.foo.find({"a": 1})
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXSCAN",
        "indexName" : <idx_name>
    }
}
```

IXONLYSCAN はインデックスキーのみをスキャンします。複合インデックスを作成しても、FETCH は回避されません。

インデックス公差

IXAND

Amazon DocumentDB は、インデックス交差を利用できる場合、IXSCAN の InputStages 配列を持つ IXAND ステージを含めることができます。例えば、次のような出力が表示される場合があります。

```
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXAND",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                "indexName" : "a_1"
            },
```

```
            {
                "stage" : "IXSCAN",
                "indexName" : "b_1"
            }
        ]
    }
}
```

インデックス結合

IXOR

インデックス交差と同様に、Amazon DocumentDBは、IXOR 演算子の inputStages 配列で $or ステージを含むことができます。

```
db.foo.find({"$or": [{"a": {"$gt": 2}}, {"b": {"$lt": 2}}]})
```

上記のクエリの場合、説明出力は以下のようになります。

```
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXOR",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                "indexName" : "a_1"
            },
            {
                "stage" : "IXSCAN",
                "indexName" : "b_1"
            }
        ]
    }
}
```

複数のインデックス交差/結合

Amazon DocumentDB は、複数のインデックス交差または結合ステージを組み合わせて、結果を取得できます。例:

```
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXOR",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                ...
            },
            {
                "stage" : "IXAND",
                "inputStages" : [
                    {
                        "stage" : "IXSCAN",
                        ...
                    },
                    {
                        "stage" : "IXSCAN",
                        ...
                    }
                ]
            }
        ]
    }
}
```

インデックスの交差や結合ステージの使用は、インデックスの種類 (スパース、複合など) に影響されません。

複合インデックス

Amazon DocumentDB の複合インデックスの使用は、インデックスされたフィールドの最初のサブセットに限定されるものではありません。それはサフィックス部分を持つインデックスを使用できますが、あまり効率的ではないかもしれません。

例えば、`{ a: 1, b: -1 }` の複合インデックスは、以下の 3 つのクエリをサポートすることができます。

```
db.orders.find( { a: 1 } )
```

```
db.orders.find( { b: 1 } )
```

```
db.orders.find( { a: 1, b: 1 } )
```

ソートステージ

要求されたソートキーにインデックスがある場合、Amazon DocumentDB はインデックスを使用して順序を取得することができます。その場合、結果には SORT ステージではなく IXSCAN ステージが含まれます。オプティマイザがプレーンソートを好む場合は、次のようなステージが含まれます。

```
{
    "stage" : "SORT",
    "sortPattern" : {
        "a" : 1,
        "b" : -1
    }
}
```

グループステージ

Amazon DocumentDBは、2 つの異なるグループ戦略をサポートしています。

- SORT_AGGREGATE: オンディスクソート集計。
- HASH_AGGREGATE: メモリハッシュ集計。

クエリプランナー v2

Amazon DocumentDB の新しいクエリプランナー (プランナーバージョン 2.0) は、高度なクエリ最適化機能とパフォーマンスの向上を備えています。Amazon DocumentDB 5.0 のプランナーバージョン 2.0 は、インデックスで find および update 演算子を使用する場合、以前のバージョンと比較して最大 10 倍のパフォーマンスの向上を実現します。パフォーマンスの向上は、主に、より最適なインデックスプランを使用し、否定演算子 ($neq、$nin) やネストされた $elementMatch などの演算子のインデックススキャンサポートを有効にすることによるものです。プランナーバージョン 2.0 クエリは、コスト見積もり手法の向上、アルゴリズムの最適化、安定性の向上により、より高速に実行されます。プランナーバージョン 2.0 では、プランキャッシュフィルター API もサポートされているため、プランナーの安定性が向上します。この機能により、Amazon DocumentDB 5.0 では、さまざまなバージョンのクエリプランナーから選択できるようになりました。

トピック

- 前提条件

- デフォルトのクエリプランナーとしてのプランナーバージョン 2.0 の選択
- ベストプラクティス
- 制限事項
- Find および Update 演算子の改善
- キャッシュフィルターの計画
- プランナーバージョン 1.0、2.0、および MongoDB の潜在的な動作の違い
- プランナーバージョン 2.0 が MongoDB の動作ギャップを埋める

前提条件

プランナーバージョン 2.0 には、次の前提条件が適用されます。

- プランナーバージョン 2.0 は、エンジンバージョン 5.0 が利用可能なすべての AWS リージョン で使用できます。
- デフォルトのクエリプランナーとしてバージョン 2.0 を使用することを選択するには、クラスターが Amazon DocumentDB バージョン 5.0 のエンジンパッチバージョン 3.0.15902 以降である必要があります。最新のエンジンバージョンパッチに更新する手順については、「クラスターのエンジンバージョンへのパッチ更新の実行」を参照してください。
- プランナーバージョン 2.0 をデフォルトのクエリプランナーとして設定するには、クラスターパラメータグループを更新するための IAM アクセス許可が必要です。
- Amazon DocumentDB 8.0 では、プランナーバージョン 3.0 がデフォルトのクエリプランナーです。詳細については、「クエリプランナー v3」を参照してください。

デフォルトのクエリプランナーとしてのプランナーバージョン 2.0 の選択

コンソールまたは CLI からデフォルトのクエリプランナーとして 2.0 を選択するには、次の手順に従います。

- 「Amazon DocumentDB クラスターパラメータを変更する」の手順に従って、クラスターのパラメータグループを変更します。
- 「plannerVersion」というタイトルのパラメータの場合、プランナーバージョン 2.0 を示す値を 2.0 に変更します。
- [すぐに適用] を選択します ([再起動時に適用] を選択すると、クラスターの次回の再起動まで選択が無効になります)。

ベストプラクティス

期待される結果を得るには、プランナーバージョン 2.0 を適用するときに次のベストプラクティスを使用します。

- グローバルクラスターで、両方のリージョンのクラスターパラメータグループで同じplannerVersion値 (1.0 または 2.0) を選択します。プライマリリージョンとセカンダリリージョンで異なるプランナーバージョンを選択すると、クエリの動作とパフォーマンスに一貫性がなくなる可能性があることに注意してください。
- スケジュールされたメンテナンスウィンドウ中またはトラフィックの減少期間中にプランナーバージョン 2.0 を更新すると、ワークロードがアクティブに実行されているときにプランナーバージョンが変更されると エラー率が高くなる可能性があるため、中断が最も少なくなります。
- プランナーバージョン 2.0 は、MongoDB シェルバージョン 5.0 で最も最適に動作します。

制限事項

プランナーバージョン 2.0 には、以下の制限が適用されます。

- プランナーバージョン 2.0 は、プランナーバージョン 1.0 にフォールバックされるエラスティッククラスターではサポートされていません。
- プランナーバージョン 2.0 は、集約コマンドと distinct コマンドではサポートされていません。これらはプランナーバージョン 1.0 にフォールバックされます。
- 一部のクエリシェイプには、プランキャッシュフィルターを設定できません。現在のリストについては、「」を参照してくださいサポートされているクエリシェイプ。

Find および **Update** 演算子の改善

プランナーバージョン 2.0 は、find、update、delete、および find-and-modify コマンドなどの基本的なオペレーションを最適化します。以下のタブ付きセクションでは、インデックスの拡張機能と、プランナーバージョン 2.0 でのクエリパフォーマンスの向上を示しています。

Enhanced index support

- プランナーバージョン 2.0 では、$nin、$ne、$not {eq}、$not {in} などの否定演算子と、$type および $elemMatch のインデックスサポートが追加されています。

```
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 }}}}})
```

- プランナーバージョン 2.0 は、クエリ式に $exists が存在しない場合でも、スパースインデックスまたは部分インデックスを使用します。

```
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" })
```

- プランナーバージョン 2.0 では、クエリ条件が部分インデックスフィルター式と完全に一致しない場合でも、部分インデックスが使用されます。

```
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}})
```

- プランナーバージョン 2.0 では、$elemMatch クエリで部分インデックススキャンを使用します。

```
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}}})
```

- プランナーバージョン 2.0 には、アプリケーションコードで $hint を指定することなく、$regex のインデックススキャンサポートが含まれています。$regex はプレフィックス検索でのみインデックスをサポートします。

```
Sample Document: { "x": [1, 2, 3], "y": "apple" }
db.foo.createIndex({ "x": 1, "y": 1 })
db.foo.find({"y":{ $regex: "^a" }})
```

- プランナーバージョン 2.0 では、マルチキーフィールドの等価条件を使用して、マルチキーインデックスを含むクエリのパフォーマンスが向上します。

```
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)
```

- プランナーバージョン 2.0 では、特に 8 KB を超えるドキュメントを含むコレクションで、複数のフィルターを含むクエリのパフォーマンスが向上します。

```
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}}]})
```

- プランナーバージョン 2.0 では、ソートステージを削除することで、複合インデックスで $in 演算子を使用する場合のクエリのパフォーマンスが向上します。

```
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})
```

また、 $in要素でマルチキーインデックスを使用するクエリのパフォーマンスも向上します。

```
Sample Document: {"x": [1, 2, 3]}
```

```
db.foo.createIndex({"x": 1})
db.foo.find("x":{$in:[>100 elements]})
```

Query performance improvements

- プランナーバージョン 2.0 では、特に 8 KB を超えるドキュメントを含むコレクションで、複数のフィルターを含むクエリのパフォーマンスが向上します。

```
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}}]})
```

- プランナーバージョン 2.0 では、ソートステージを削除することで、複合インデックスで $in 演算子を使用する場合のクエリのパフォーマンスが向上します。

```
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})
```

また、$in 要素でマルチキーインデックスを使用するクエリのパフォーマンスも向上します。

```
Sample Document: {"x": [1, 2, 3]}
db.foo.createIndex({"x": 1})
db.foo.find("x":{$in:[>100 elements]})
```

キャッシュフィルターの計画

プランナーバージョン 2.0 では、インデックスフィルターとも呼ばれるプランキャッシュフィルターのサポートが追加されています。これにより、プランナーが特定のクエリシェイプについて考慮するインデックスのセットを制限できます。フィルタはデータベースコマンドで設定され、サーバーに適用されるため、クエリのリグレッションが発生した場合は、アプリケーションコードを変更せずに軽減できます。

プランナーバージョン 2.0 では、プランキャッシュフィルターは findおよび count コマンドに適用され、Amazon DocumentDB 5.0.2 および 8.0.2 以降は update、delete、および findAndModify コマンドにも適用されます。distinct および aggregate コマンドには、プランナーバージョン 3.0 が必要です。

コマンドリファレンス、サポートされているクエリシェイプ、実例、フィルターがヒントとやり取りする方法の詳細については、「」を参照してくださいキャッシュフィルターの計画。

プランナーバージョン 1.0、2.0、および MongoDB の潜在的な動作の違い

エッジケースによっては、プランナーバージョン 2.0 が MongoDB の結果とわずかに異なる結果を生成する可能性があります。このセクションでは、これらの可能性の例をいくつか説明します。

$(update) and $(projection)

- 場合によっては、MongoDB の $(update) および $(projection) 演算子が Amazon DocumentDB のプランナーバージョン 1.0 とは異なる動作をすることがあります。以下に一部の例を示します。

```
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 } }
```

  - プランナーバージョン 1.0 — フィールドを更新 2022
  - MongoDB — フィールドを更新 2022
  - プランナーバージョン 2.0 — フィールドを更新 2021

-
```
db.col.insert({x:[1,2,3]})
db.col.update({$and:[{x:1},{x:3}]},{$set:{"x.$":500}})
```

  - プランナーバージョン 1.0 — 一致する最初の要素をランダムに更新
  - MongoDB — 一致した最初の要素をランダムに更新
  - プランナーバージョン 2.0 — 更新を行わない

-
```
db.col.insert({x:[1,2,3]})
db.col.find()
```

  - プランナーバージョン 1.0 — 一致する要素をランダムに選択
  - MongoDB — 一致した要素をランダムに選択
  - プランナーバージョン 2.0 — 選択しない
- ```
  db.col.insert({x:100})
  db.col.update({x:100},{x:100})
  ```
  - プランナーバージョン 1.0 — nModified カウントの変更
  - MongoDB — nModified カウントの変更
  - プランナーバージョン 2.0 — nModified カウントは、同じ値で更新しても変更されない。
- `$(update)` 演算子を `$setOnInsert` で使用すると、プランナーバージョン 1.0 と MongoDB はエラーをスローしますが、プランナーバージョン 2.0 はスローしません。
- 存在しないフィールドの名前を `$field` に変更すると、プランナーバージョン 2.0 ではエラーがスローされますが、プランナーバージョン 1.0 および MongoDB では更新は生成されません。

Index behavior

- `$hint` が不適切なインデックスで適用されると、プランナーバージョン 2.0 はエラーをスローしますが、プランナーバージョン 1.0 と MongoDB はスローしません。

```
// 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 は、プランナーバージョン 2.0 では $hint({“$natural”:1}) を使用できません。

```
// 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" }
```

- MongoDB は完全な正規表現インデックススキャンをサポートしていますが、プランナーバージョン 2.0 はプレフィックスフィールドでのみ正規表現インデックススキャンをサポートしています。

```
// 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)
```

- プランナーバージョン 2.0 でのプランキャッシュフィルターの使用には、MongoDB と固有の違いがあります。プランナーバージョン 2.0 では、プランキャッシュフィルターとの「射影」と「照合」の指定はサポートされていませんが、MongoDB はサポートしています。ただし、MongoDB インデックスフィルターはインメモリのみであり、再起動後に失われます。プランナーバージョン 2.0 は、再起動とパッチを通じてインデックスフィルターを保持します。

Others

- プランナーバージョン 2.0 を使用する場合の DML 監査ログの形式は、プランナーバージョン 1.0 の形式と若干異なります。

```
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}}
```

- 説明プランの一部としてのインデックス条件:

```
rs0:PRIMARY> db.col.createIndex({index1:1})
{
 "createdCollectionAutomatically" : false,
 "numIndexesBefore" : 1,
 "numIndexesAfter" : 2,
 "ok" : 1,
 "operationTime" : Timestamp(1761149251, 1)
}
```

インデックス条件とフィルターを表示するプランナーバージョン 2.0 の説明プラン出力:

```
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)
}
```

プランナーバージョン 1.0 の説明プラン出力:

```
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)
}
```

プランナーバージョン 2.0 が MongoDB の動作ギャップを埋める

プランナーバージョン 2.0 が MongoDB の動作ギャップを埋める領域がいくつかあります。

- プランナーバージョン 2.0 では、$elemMatch のフラット化された配列で数値インデックスルックアップが許可されます。

```
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}}})
```

- プランナーバージョン 1.0 は射影で文字列を除外しましたが、プランナーバージョン 2.0 の動作は MongoDB と一致し、リテラル値として扱います

```
// 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 }
```

- MongoDB と同様に、プランナーバージョン 2.0 では、同じフィールド「x」と「x.a」での射影は許可されません。

```
// 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 ] }
```

- MongoDB と同様に、プランナーバージョン 2.0 では、サブドキュメントへの射影が許可されます。

```
// 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"
}
```

- MongoDB などのプランナーバージョン 2.0 では、射影は $ 演算子の後のフィールドをサポートしていません。

```
// 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 } ] }
```

- MongoDB などのプランナーバージョン 2.0 では、次の $hint の使用が許可されます。

```
// 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 }
```

クエリプランナー v3

Amazon DocumentDB 8.0 のプランナーバージョン 3 は、6 つの新しいステージを含む 21 の集約ステージをサポートしています。プランナー V3 には、個別のコマンドのサポートが組み込まれています。Amazon DocumentDB 5.0 のプランナー v2 と比較して、全体的なパフォーマンスが最大 2 倍向上します。Amazon DocumentDB 8.0 のすべての新機能と演算子は、Planner v3 と互換性があります。プランナー v3 でサポートされている Amazon DocumentDB 8.0 の新しい集計ステージには、$replaceWith、$vectorSearch、$merge、$set、$unset、$bucket が含まれます。プランナー v3 は、照合、ビュー、$merge、$pow、$rand、$dateTrunc、$dateToParts、$dateFromParts など、Amazon DocumentDB 8.0 の新機能と演算子もサポートしています。 dateFromParts

トピック

- 前提条件
- デフォルトのクエリプランナーとしてのプランナーバージョン 3.0 の選択
- ベストプラクティス

- 制限事項
- aggregate および distinct 演算子の改善
- キャッシュフィルターの計画
- プランナーバージョン 1.0、3.0、および MongoDB の潜在的な動作の違い

前提条件

プランナーバージョン 3.0 には、次の前提条件が適用されます。

- プランナーバージョン 3.0 は、エンジンバージョン 8.0 が利用可能なすべての AWS リージョン で使用できます。
- エンジンバージョン 8.0 が選択されている場合、デフォルトのクエリプランナーはプランナーバージョン 3.0 です。

デフォルトのクエリプランナーとしてのプランナーバージョン 3.0 の選択

Amazon DocumentDB 8.0 でデフォルトのクエリプランナーを変更し、プランナー v3 に戻す必要がある場合は、コンソールまたは CLI から行うことができます。

- 「Amazon DocumentDB クラスターパラメータを変更する」の手順に従って、クラスターのパラメータグループを変更します。
- plannerVersion」というタイトルのパラメータで、プランナーバージョン 3.0 を示す値を 3.0 に変更します。
- [すぐに適用] を選択します ([再起動時に適用] を選択すると、クラスターの次回の再起動まで選択が無効になります)。

ベストプラクティス

期待される結果を得るには、プランナーバージョン 3.0 を適用するときに次のベストプラクティスを使用します。

- グローバルクラスターで、両方のリージョンのクラスターパラメータグループで同じ`plannerVersion`値 (1.0 または 2.0 または 3.0) を選択します。プライマリリージョンとセカンダリリージョンで異なるプランナーバージョンを選択すると、クエリの動作とパフォーマンスに一貫性がなくなる可能性があることに注意してください。

- スケジュールされたメンテナンスウィンドウ中またはトラフィックの減少期間中にプランナーバージョン 3.0 に更新すると、ワークロードがアクティブに実行されているときにプランナーバージョンが変更されると工ラー率が高くなる可能性があるため、中断が最も少なくなります。

制限事項

プランナーバージョン 3.0 には、次の制限が適用されます。

- プランナーバージョン 3.0 は、プランナーバージョン 1.0 にフォールバックされる Elastic クラスターではサポートされていません。

- プランナー v1 では、クエリオプティマイザによって特定のクエリプランが選択されるようにplanHint」の使用が許可されますが、プランナー v3 ではplanHint」の使用が許可されず、内部最適化に依存して特定のクエリに最適なプランが選択されます。

aggregate および **distinct** 演算子の改善

プランナーバージョン 3.0 では、$aggregate ステージと $distinct コマンドの改善が導入されています。以下は、最も注目すべき改善点の一部です。

- プランナーは、可能な場合はパイプラインの早い段階で $match ステージを移動し、後続のステージで処理されるドキュメントの数を減らします。

```
//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 } }
])
```

- プランナーは、同じフィールドで動作するときに $lookup ステージと $unwind ステージを自動的に組み合わせ、中間データ処理を減らし、パフォーマンスを向上させます。

```
                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
```

- プランナーバージョン 3.0 では、低カーディナリティインデックスでの異なるオペレーションのパフォーマンスを大幅に向上させる新しい Distinct Scan 実行戦略が導入されました。

```
                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"
                        }
                }
        }
}
```

キャッシュフィルターの計画

プランナーバージョン 3.0 では、インデックスフィルターとも呼ばれるプランキャッシュフィルターがサポートされています。これにより、プランナーが特定のクエリシェイプについて考慮するインデックスのセットを制限できます。フィルタはデータベースコマンドで設定され、サーバーに適用されるため、クエリのリグレッションが発生した場合は、アプリケーションコードを変更せずに軽減できます。

プランナーバージョン 3.0 は、プランナーバージョン 2.0 よりも多くのコマンドにプランキャッシュフィルターを適用します。

- `distinct` コマンドにはプランナーバージョン 3.0 が必要です。
- `aggregate` コマンドには、プランナーバージョン 3.0 と Amazon DocumentDB 8.0.2 が必要です。パイプラインが先行`$match`ステージを通じてコレクションを読み取ると、フィルターが適用されます。
- `$lookup` ステージまたは `$graphLookup`ステージの外部コレクションに設定されたフィルターは、その外部コレクションのスキャンに使用されるインデックスを駆動します。これには、プランナーバージョン 3.0 と Amazon DocumentDB 8.0.2 も必要です。

コマンドリファレンス、サポートされているクエリシェイプ、実例、フィルターがヒントとやり取りする方法の詳細については、「」を参照してくださいキャッシュフィルターの計画。

プランナーバージョン 1.0、3.0、および MongoDB の潜在的な動作の違い

エッジケースによっては、プランナーバージョン 3.0 がプランナーバージョン 1.0 とわずかに異なる結果を生成する場合があります。このセクションでは、これらの可能性の例をいくつか説明します。

Feature Differences

- 空のコレクションまたはマッチなどの以前のステージがすべてのドキュメントをフィルタリングする場合、プランナー v1 は出力に「field」:0 を与えます。プランナー v3 と MongoDB は出力を提供しません。
- Plannerv1 では、前のステージで返されたドキュメントが n 個未満の場合、{"$skip":n} はスキップしません。返されたドキュメントの数に関係なく、Plannerv3 と MongoDB は正しくスキップします。
- $lookup で参照されている外部コレクションが存在しない場合、Plannerv1 はエラーをスローします。プランナー v3 と MongoDB は、外部コレクションを空のコレクションとして扱い、$lookup を実行します。

```
db.coll.aggregate([
    {$lookup: {from: "does_not_exist", localField: "a", foreignField: "a", as:
 "c"}}
])
```

- パイプラインで複数の $search を使用することを許可するのはプランナー v1 のみです。プランナー v2/v3 はエラーをスローし、MongoDB はそれをサポートしていません。

```
VectorSearch = { "$search": { "vectorSearch": {
   "vector": [0.2, 0.5, 0.8],
   "path": "vectorEmbedding",
   "similarity": "cosine",
   "k": 2,
   "efSearch": 1
   }}}
db.coll.aggregate([VectorSearch, VectorSearch])
```

- vectorSearch ステージが最初のステージでない場合、Planner v3 のみが機能します。この場合、MongoDB はエラーをスローし、Planner v1 は $vectorSearch ステージをサポートしていません。

```
VectorSearch = { {"$vectorSearch": {
    "queryVector": [0.2, 0.5, 0.8],
    "path": "vectorEmbedding",
    "similarity": "euclidean",
    "limit": 4,
    "numCandidates": 100} }

db.coll.aggregate([$match:{}, VectorSearch])
```

Amazon DocumentDB を使用した地理空間データのクエリ

このセクションでは、Amazon DocumentDB を使用して地理空間データをクエリする方法について説明します。このセクションを読むと、Amazon DocumentDB に地理空間データを格納、クエリ、インデックス付けする方法について理解することができます。

トピック

- 概要
- 地理空間データのインデックス作成と格納

- 地理空間データのクエリ
- 制限事項

概要

地理空間の一般的なユースケースには、データからの近接解析が含まれます。たとえば、「シアトルから50マイル以内のすべての空港を検索する」、または「特定の場所から最も近いレストランを探す」などです。Amazon DocumentDB は GeoJSON 仕様 を使用して、地理空間データを表します。GeoJSON は、座標空間内の図形の JSON 形式のためのオープンソース仕様です。GeoJSON 座標は、地球のような球上の位置を表す経度と緯度の両方をキャプチャします。

地理空間データのインデックス作成と格納

Amazon DocumentDB は、地理空間データを格納するために「ポイント」GeoJSON タイプを使用します。各 GeoJSON ドキュメント (またはサブドキュメント) は、通常 2 つのフィールドで構成されます。

- タイプ - 表される図形。Amazon DocumentDB に「座標」フィールドの解釈方法を知らせます。現時点では、Amazon DocumentDB はポイントのみをサポートしています
- 座標 — 配列内のオブジェクトとして表される緯度と経度のペア — [経度、緯度]

Amazon DocumentDB では、2dsphere インデックスを使用して地理空間データのインデックスも使用します。Amazon DocumentDB はインデックスポイントをサポートしています。Amazon DocumentDB は、2dsphere インデックスを使用した近接クエリをサポートしています。

フードデリバリーサービスのアプリケーションを構築しているシナリオを考えてみましょう。レストランの緯度と経度のペアを Amazon DocumentDB に保存したい。そのためには、まず緯度と経度のペアを保持する地理空間フィールドにインデックスを作成します。

```
use restaurantsdb
db.usarestaurants.createIndex({location:"2dsphere"})
```

このコマンドの出力は次のようになります。

```
{
 "createdCollectionAutomatically" : true,
 "numIndexesBefore" : 1,
 "numIndexesAfter" : 2,
```

```
 "ok" : 1
}
```

インデックスを作成したら、Amazon DocumentDB コレクションへのデータの挿入をスタートできます。

```
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
        ]
    }
});
```

地理空間データのクエリ

Amazon DocumentDB は、地理空間データの近接クエリをサポートしています。近接クエリの良い例は、特定の距離より小さく、別のポイントからの距離を超えるすべてのポイントを検索することです。包含クエリの良い例は、指定したエリア/ポリゴン (ニューヨーク州) にあるすべてのポイント (すべての空港) を検索することです。交差クエリの良い例として、ポイント (都市) と交差するポリゴン (州/地域) の検索があります。次の地理空間演算子を使用して、データからインサイトを得ることができます。

- **$nearSphere** - $nearSphere は、GeoJSON ポイントから最も近いポイントから最も遠いポイントの検索をサポートする検索演算子です。
- **$geoNear** - $geoNear は、GeoJSON ポイントからの距離をメートル単位で計算する集計演算子です。
- **$minDistance** - $minDistance は、$nearSphere または $geoNear と併用される検索演算子で、中心ポイントから指定した最小距離のドキュメントをファイリングします。
- **$maxDistance** - $maxDistance は、$nearSphere または $geoNear と併用される検索演算子で、中心ポイントから指定した最大距離のドキュメントをファイリングします。
- **$geoWithin**: $geoWithin は、ポリゴンなどの特定の形状に完全に存在する地理空間データを含むドキュメントの検索をサポートする検索演算子です。
- **$geoIntersects**: $geoIntersects は、地理空間データが指定された GeoJSON オブジェクトと交差するドキュメントの検索をサポートする検索演算子です。

> **Note**
>
> $geoNear そして $nearSphere は、近接クエリで使用する GeoJSON フィールドに 2dsphere インデックスを必要とします。

例 1

この例では、住所 (ポイント) から最も近い距離でソートされたすべてのレストラン (ポイント) を検索する方法を学習します。

このようなクエリを実行するには、$geoNear を使用して、別のポイントからのポイントのセットの距離を計算します。また、distanceMultiplier を追加して、距離をキロメートル単位で測定します。

```
db.usarestaurants.aggregate([
   {
      "$geoNear":{
         "near":{
            "type":"Point",
            "coordinates":[
               -122.3516,
               47.6156
            ]
         },
         "spherical":true,
         "distanceField":"DistanceKilometers",
         "distanceMultiplier":0.001
      }
   }
])
```

上記のコマンドでは、指定したポイントからの距離 (最も遠い) でソートされたレストランが返されます。このコマンドの出力は次のようになります。

```
{ "_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**
>
> $geoNear ステージの limitおよび numオプション、およびデフォルトの 100 ドキュメントの制限は、Amazon DocumentDB 8.0 (エンジンパッチバージョン 4.0.5748) 以降では廃止されました。limit と を使用する次の例は、Amazon DocumentDB 8.0 (エンジンパッチバージョン 4.0.4448) 以前のバージョンnumにのみ適用されます。

クエリの結果の数を制限するには、limit または num オプションを使用します。

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
      }
   }
])
```

例 2

この例では、特定の住所 (ポイント) から 2 キロメートル以内のすべてのレストラン (ポイント) を検索する方法を学習します。このようなクエリを実行するには、GeoJSON ポイントから $nearSphere 最小値以内および $minDistance 最大値 $maxDistance を使用します。

```
db.usarestaurants.find({
   "location":{
      "$nearSphere":{
         "$geometry":{
            "type":"Point",
            "coordinates":[
               -122.3516,
               47.6156
            ]
         },
         "$minDistance":1,
         "$maxDistance":2000
      }
   }
},
{
   "name":1
})
```

上記のコマンドは、指定されたポイントから最大 2 km のレストランを返します。このコマンドの出力は次のようになります。

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "name" : "Noodle House" }
```

制限事項

Amazon DocumentDB は、ポリゴン、ラインストリング、マルチポイント、マルチポリゴン、マルチラインストリング、GeometryCollection のクエリやインデックス作成をサポートしていません。

部分インデックス

部分インデックスは、特定のフィルター基準を満たすコレクション内のドキュメントをインデックス化します。部分インデックス機能は、Amazon DocumentDB 5.0 インスタンスベースのクラスターでサポートされています。

トピック

- 部分インデックスを作成する
- サポートされている演算子
- 部分インデックスを使用したクエリ
- 部分インデックス機能
- 部分インデックスに関する制限事項

部分インデックスを作成する

部分インデックスを作成するには、partialFilterExpression オプションで createIndex() メソッドを使用します。例えば、次のオペレーションでは、OrderID を持ち、かつ isDelivered フィールドが true であるドキュメントをインデックス化する一意の複合インデックスを注文コレクションに作成します。

```
db.orders.createIndex(
  {"category": 1, "CustomerId": 1, "OrderId": 1},
  {"unique": true, "partialFilterExpression":
    {"$and": [
      {"OrderId": {"$exists": true}},
      {"isDelivered": {"$eq": false}}
    ]}
  }
)
```

サポートされている演算子

- $eq
- $exists
- $and (最上位のみ)
- $gt/$gte/$lt/$lte (インデックススキャンは、クエリに述されているフィルターが部分フィルター式と完全に一致する場合にのみ使用されます) (制限事項を参照)

部分インデックスを使用したクエリ

部分インデックスの使用により、次のクエリパターンを使用できます。

- クエリ述語は、部分インデックスフィルター式と完全に一致します。

```
db.orders.find({"$and": [
    {"OrderId": {"$exists": true}},
    {"isDelivered": {"$eq": false}}
  ]}).explain()
```

- クエリフィルターの想定される結果は、部分フィルターの論理サブセットです。

```
db.orders.find({"$and": [
    {"OrderId": {"$exists": true}},
    {"isDelivered": {"$eq": false}},
    {"OrderAmount": {"$eq": "5"}}
  ]}).explain()
```

- クエリのサブ述語は、他のインデックスと組み合わせて使用できます。

```
db.orders.createIndex({"anotherIndex":1})
db.orders.find({ "$or": [
      {"$and": [
        {"OrderId": {"$exists": true}},
        {"isDelivered": {"$eq": false}}
      ]},
      {"anotherIndex": {"$eq": 5}}
    ]
  }).explain()
```

> **Note**
>
> クエリプランナーは、効率的に望ましい場合には、インデックススキャンではなくコレクションスキャンを使用することがあります。これは通常、コレクションの大部分を返す、非常に小さなコレクションまたはクエリに使用されます。

部分インデックス機能

部分インデックスを一覧表示する

`getIndex` オペレーションを使用して、partialFilterExpression を含む部分インデックスを一覧表示します。例えば、`getIndex` オペレーションは、キー、名前、および partialfilterExpressions フィールドを含む部分インデックスを一覧表示します。

```
db.orders.getIndexes()
```

この例は、以下のサンプル出力を返します。

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

同じキー: 順序の複数の部分フィルター式

同じフィールドの組み合わせ (キー:順序) に対して異なる部分インデックスを作成できます。これらのインデックスには別の名前が必要です。

```
db.orders.createIndex(
  {"OrderId":1},
  {
    name:"firstPartialIndex",
```

```
    partialFilterExpression:{"OrderId":{"$exists": true}}
  }
)
```

```
db.orders.createIndex(
  {"OrderId":1},
  {
    name:"secondPartialIndex",
    partialFilterExpression:{"OrderId":{"$gt": 1000}}
  }
)
```

getIndexes オペレーションを実行して、コレクション内のすべてのインデックスを一覧表示します。

```
db.orders.getIndexes()
```

コマンドは、次の出力例を返します。

```
[
  {
    "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**
>
> インデックス名は異なる必要があり、名前のみで削除する必要があります。

部分プロパティと TTL プロパティを持つインデックス

インデックスの作成時に partialFilterExpression と expireAfterSeconds の両方のオプションを指定することで、部分プロパティと TTL プロパティを持つインデックスを作成することもできます。これにより、コレクションからどのようなドキュメントを除外するのかをよりきめ細かにコントロールできます。

例えば、特定の期間後に削除するドキュメントを特定する TTL インデックスがあるとします。ここで部分インデックスオプションを使用することで、ドキュメントを削除するタイミングに関する追加の条件を指定することができます。

```
db.orders.createIndex(
    { "OrderTimestamp": 1 },
    {
        expireAfterSeconds: 3600 ,
        partialFilterExpression: { "isDelivered": { $eq: true } }
    }
)
```

この例は、以下のサンプル出力を返します。

```
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1,
        "operationTime" : Timestamp(1234567890, 1)
}
```

getIndexes オペレーションを実行して、コレクションに存在するインデックスを一覧表示します。

```
db.orders.getIndexes()
[
    {
        "v" : 4,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.orders"
    }
```

この例は、以下のサンプル出力を返します。

```
[
    {
        "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
    }
]
```

部分インデックスに関する制限事項

部分インデックス機能には、次の制限事項が適用されます。

- Amazon DocumentDB の不等式クエリは、クエリフィルター述語が `partialFilterExpression` と完全に一致し、かつ同じデータ型である場合にのみ、部分インデックスを使用します。

  > **Note**
  >
  > 上記のケースでは、`$hint` は IXSCAN の執行にも使用できません。

  次の例では、`partialFilterExpression` は `field1` にのみ適用され、`field2` には適用されません。

  ```
  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
  ```

- 配列演算子を持つ `partialFilterExpression` はサポートされていません。次の演算はエラーを生じます。

  ```
  db.orders.createIndex(
    {"CustomerId":1},
    {'partialFilterExpression': {'OrderId': {'$eq': [1000, 1001, 1002]}}}
  )
  ```

- partialFilterExpression フィールドでは、次の演算子はサポートされていません。
  - `$all` (配列演算子)
  - `$mod` (配列演算子)
  - `$or`
  - `$xor`

- `$not`
- `$nor`

- フィルター式とフィルターのデータ型は同じである必要があります。

Amazon DocumentDB でのテキスト検索の実行

Amazon DocumentDB のネイティブフルテキスト検索機能 (テキストインデックス v1) を使用すると、特殊目的のテキストインデックスを使用して、大きなテキストデータセットに対してテキスト検索を実行できます。このセクションでは、テキストインデックス機能の機能について説明し、Amazon DocumentDB でテキストインデックスを作成および使用する手順について説明します。テキスト検索における制限事項もリストアップします。

トピック

- サポートされている機能
- Amazon DocumentDB テキストインデックスの使用
- MongoDB との違い
- ベストプラクティスとガイドライン
- テキストインデックス V2
- 制限事項

サポートされている機能

Amazon DocumentDB テキスト検索は、以下の MongoDB API 互換機能をサポートしています。

- 1 つのフィールドにテキストインデックスを作成します。
- 複数のテキストフィールドを含む複合テキストインデックスを作成します。
- 単一単語または複数単語の検索を実行します。
- 重み付けにより検索結果を制御します。
- 検索結果をスコア順にソートします。
- 集約パイプラインでテキストインデックスを使用します。
- 完全一致フレーズを検索できます。

Amazon DocumentDB テキストインデックスの使用

文字列データを含むフィールドにテキストインデックスを作成するには、次のように文字列「text」を指定します。

単一フィールドインデックス

```
db.test.createIndex({"comments": "text"})
```

このインデックスは、指定されたコレクションの「comments」文字列フィールドのテキスト検索クエリをサポートします。

複数の文字列フィールドに複合テキストインデックスを作成します。

```
db.test.createIndex({"comments": "text", "title":"text"})
```

このインデックスは、指定されたコレクションの「comments」および「title」文字列フィールドでのテキスト検索クエリをサポートします。複合テキストインデックスを作成する際は、最大 30 個のフィールドを指定できます。作成したテキスト検索クエリは、すべてのインデックス付きフィールドを対象にクエリを実行します。

> **Note**
>
> 各コレクションで許可されるテキストインデックスは 1 つだけです。

Amazon DocumentDB コレクションでのテキストインデックスの一覧表示

次の例に示すように、コレクション`getIndexes()`で を使用して、テキストインデックスを含むインデックスを識別して記述できます。

```
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
   }
]
```

インデックスを作成したら、Amazon DocumentDB コレクションへのデータの挿入をスタートできます。

```
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"}])
```

テキスト検索クエリの実行

単一語テキスト検索クエリを実行する

テキスト検索を実行するには、$text と $search 演算子を使用する必要があります。次の例では、テキストインデックス付きフィールドに文字列「apple」または、「apple」を他の形式で含む(「apples」、等)文字列を含むすべてのドキュメントを返します。

```
db.test.find({$text: {$search: "apple"}})
```

出力:

このコマンドの出力は次のようになります。

```
{ "_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" }
```

複数単語のテキスト検索を実行する

Amazon DocumentDB データに対して複数単語のテキスト検索を実行することもできます。次のコマンドは、「apple」または「pie」を含むテキストインデックス付きフィールドを持つドキュメントを返します。

```
db.test.find({$text: {$search: "apple pie"}})
```

出力:

このコマンドの出力は次のようになります。

```
{ "_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" }
```

複数単語のフレーズテキスト検索を実行する

複数単語のフレーズ検索には、次の例を使用します。

```
db.test.find({$text: {$search: "\"apple pie\""}})
```

出力:

上記のコマンドは、「apple pie」というフレーズに完全一致する文字列を含むテキストインデックス付きフィールドを持つドキュメントを返します。このコマンドの出力は次のようになります。

```
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven" }
```

フィルターを使用してテキスト検索を実行する

テキスト検索を他のクエリ演算子と組み合わせて、追加の基準に基づいて結果をフィルタリングすることもできます。

```
db.test.find({$and: [{star_rating: 5}, {$text: {$search: "interest"}}]})
```

出力:

上記のコマンドは、「interest」のすべての変化形と、「星評価」が 5 であるテキストインデックス付きフィールドを持つドキュメントを返します。このコマンドの出力は次のようになります。

```
{ "_id" : 5, "star_rating" : 5, "comments" : "interesting couch" }
{ "_id" : 6, "star_rating" : 5, "comments" : "interested in couch for sale, year
 2022" }
```

テキスト検索で返されるドキュメントの数を制限する

`limit` を使用して、返されるドキュメントの数を制限できます。

```
db.test.find({$and: [{star_rating: 5}, {$text: {$search: "couch"}}]}).limit(1)
```

出力:

上記のコマンドは、フィルターを満たす 1 つの結果を返します。

```
{ "_id" : 5, "star_rating" : 5, "comments" : "interesting couch" }
```

テキストスコアで結果をソートする

次の例では、テキスト検索結果をテキストスコアでソートします。

```
db.test.find({$text: {$search: "apple"}}, {score: {$meta: "textScore"}}).sort({score:
 {$meta: "textScore"}})
```

出力:

上記のコマンドは、文字列「apple」または、「apple」を他の形式で含む(「apples」、等)文字列含むテキストインデックス付きフィールドを持つドキュメントを返し、それぞれのドキュメントの検索用語との関連性の高さに基づいて結果をソートします。このコマンドの出力は次のようになります。

```
{ "_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 および $search は、aggregate、count、findAndModify、update、delete の各コマンドでもサポートされています。

集約演算子

$match を使用した集約パイプライン

```
db.test.aggregate(
   [{ $match: { $text: { $search: "apple pie" } } }]
)
```

出力:

このコマンドでは次のようなレスポンスが返されます。

```
{ "_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" }
```

他の集約演算子の組み合わせ

```
db.test.aggregate(
   [
      { $match: { $text: { $search: "apple pie" } } },
      { $sort: { score: { $meta: "textScore" } } },
      { $project: { score: { $meta: "textScore" } } }
   ]
)
```

出力:

このコマンドでは次のようなレスポンスが返されます。

```
{ "_id" : 4, "score" : 0.6079270860936958 }
{ "_id" : 1, "score" : 0.3039635430468479 }
{ "_id" : 2, "score" : 0.3039635430468479 }
{ "_id" : 3, "score" : 0.3039635430468479 }
```

テキストインデックスを作成するときに複数のフィールドを指定する

重み付けは、複合テキストインデックスの最大 3 つのフィールドに付与することができます。テキストインデックスの各フィールドに割り当てられたデフォルトの重み値は 1 です。重み付けはオプションのパラメータであり、1～100,000 の範囲である必要があります。

```
db.test.createIndex(
   {
     "firstname": "text",
     "lastname": "text",
     ...
   },
   {
     weights: {
       "firstname": 5,
       "lastname":10,
       ...
     },
     name: "name_text_index"
   }
 )
```

MongoDB との違い

Amazon DocumentDB のテキストインデックス機能は、用語-頻出度アルゴリズムによる転置インデックスを使用します。デフォルトでは、テキストインデックスはスパースです。解析ロジック、トークン化区切り文字などが異なるため、同じデータセットまたはクエリ形状に対して MongoDB と同じ結果セットが返されない場合があります。

Amazon DocumentDB テキストインデックスと MongoDB には、他にも以下のような違いがあります。

- テキスト以外のインデックスを用いた複合インデックスはサポートされていません。
- Amazon DocumentDB テキストインデックスは大文字と小文字を区別しません。
- テキストインデックスでは、英語のみがサポートされています。
- 配列 (またはマルチキー) フィールドのテキストインデックス構成はサポートされていません。例えば、文字 a のテキストインデックスとして、{“a”:[“apple”, “pie”]} などといったドキュメントを使用したインデックスを作成しようとすると失敗します。
- ワイルドカードのテキストインデックス構成はサポートされていません。

- 一意のテキストインデックスはサポートされていません。
- 用語の除外はサポートされていません。

ベストプラクティスとガイドライン

- テキストスコアによるソートを含むテキスト検索クエリで最適なパフォーマンスを得るには、データをロードする前にテキストインデックスを作成します。
- テキストインデックスの運用には、インデックス化されたデータの最適化された内部コピーを保持するために追加のストレージが必要となります。これには、追加のコストが伴います。

テキストインデックス V2

Amazon DocumentDB 8.0 では、MongoDB との互換性を高めるために基盤となるテキスト検索パーサーを変更する新しいバージョンのテキストインデックス (V2) が導入されました。

V1 テキストインデックスが提供する機能に加えて、V2 テキストインデックスは次のサポートも提供します。

- プランナーは、可能な場合はパイプラインの早い段階で $match ステージを移動し、後続のステージで処理されるドキュメントの数を減らします。
- E メール、URLs、ファイルパスなどのテキストフィールドの特殊文字のトークン化が改善されました。V2 はこれらの文字列内の個々のトークンを解析して一致させることができますが、V1 は一致できません。

```
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" }
```

制限事項

テキスト検索は Amazon DocumentDB で次の制限があります。

- テキストインデックスは、レキシームとその位置情報を保存します。1 つのドキュメント内のすべてのレキシームとその位置情報の合計サイズは 1MB に制限されています。

インデックス

インデックスは、Amazon DocumentDB がコレクション内のすべてのドキュメントをスキャンせずにドキュメントをすばやく見つけることができるようにすることで、クエリのパフォーマンスを向上させます。アプリケーションが頻繁にクエリするフィールドにインデックスを作成して、レイテンシーと I/O コストを削減できます。

インデックス関連のその他のトピックについては、以下を参照してください。

- Amazon DocumentDB インデックスの管理 — インデックスの作成と構築
- インデックスの使用 — インデックス作成のベストプラクティス
- トラブルシューティングインデックス — インデックス構築の失敗と肥大化の解決
- インデックスの使用状況を分析し、未使用のインデックスを特定する方法 — 未使用のインデックスと欠落しているインデックスを見つける

トピック

- インデックス管理
- インデックスタイプ
- インデックスプロパティ

インデックス管理

このセクションでは、Amazon DocumentDB のインデックス管理トピックについて説明します。

トピック

- インデックスの作成
- インデックスの維持
- インデックスを使用するためのベストプラクティス

インデックスの作成

Amazon DocumentDB でインデックスを作成するには、いくつかの決定を行う必要があります。

- どのくらいの速さで完了する必要がありますか?
- ビルドの実行中にコレクションにアクセスできなくなることはありますか?
- インスタンスのコンピューティング能力のどれだけをビルドに割り当てることができますか?
- どのタイプのインデックスを作成すべきですか?

このセクションでは、これらの質問に回答し、インスタンスベースのクラスターコレクションに Amazon DocumentDB インデックスを作成するためのコマンドとモニタリングの例を示します。

ガイドライン

以下のガイドラインには、新しいインデックスを作成する際の基本的な制限と設定のトレードオフが含まれています。

- Amazon DocumentDB バージョンのサポート: 単一ワーカーインデックス作成はすべての Amazon DocumentDB バージョンでサポートされていますが、マルチワーカーインデックスは Amazon DocumentDB バージョン 4.0 と 5.0 でのみサポートされています。
- パフォーマンスのトレードオフ: インデックス作成プロセスのワーカー数を増やすと、Amazon DocumentDB データベースのプライマリインスタンスの CPU 使用率と読み取り IO が増加します。新しいインデックスを作成するのに必要なリソースは、実行中のワークロードでは使用できなくなります。
- Elastic クラスター: Amazon DocumentDB Elastic クラスターでは並列インデックスはサポートされていません。
- サーバーレスクラスター - サーバーレスクラスターでの並列インデックス作成には、少なくとも 32 DCUsの最小容量が必要です。このしきい値を下回ると、 `workers`パラメータは受け入れられますが、インデックスビルドは 1 つのワーカーで実行されます。並列ワーカーの最大数は、設定された最小容量でスケーリングされます。約 の`min_DCU / 8`ワーカーを使用できます。たとえ

ば、32 個の DCUs では最大 4 個のワーカーを使用し、64 DCUs では最大 8 個のワーカーを使用できます。

> **Note**
>
> 新しい並列ワーカーの制限を有効にするには、最小容量設定を変更した後にクラスターを再起動する必要があります。

- 最大ワーカー数 (インスタンスベース) - 設定できるワーカーの最大数は、データベースクラスター内のプライマリインスタンスのサイズによって異なります。これは、データベースクラスターのプライマリインスタンスにある vCPU の総数の半分です。たとえば、64 個の vCPU を備えた db.r6g.16xlarge インスタンスでは、最大 32 ワーカーを実行できます。

> **Note**
>
> 並列インデックスビルドは、インスタンス 2xlarge 以降でサポートされています。

- 最小ワーカー: 設定できるワーカーの最小数は 1 です。インスタンスベースのクラスターでのインデックス作成のデフォルト設定は 2 ワーカーです。ただし、「ワーカースレッド」オプションを使用してワーカーの数を 1 つに減らすことができます。これはプロセスが 1 つのワーカーで実行されます。
- インデックス圧縮: Amazon DocumentDB はインデックス圧縮をサポートしていません。インデックスのデータサイズは、他のオプションを使用したときより大きくなる可能性があります。
- 複数コレクションのインデックス作成: データベースクラスターのプライマリインスタンスにある vCPU の半分を、複数のコレクションでインデックス作成を実行する設定済みワーカーに使用できます。
- インデックスタイプ - Amazon DocumentDB でサポートされているインデックスタイプのリストインデックスタイプについては、「」を参照してください。

インデックスビルドのタイプ

インデックスビルドには次の 4 つのタイプがあります。

- フォアグラウンド: フォアグラウンドインデックスビルドは、インデックスの作成が完了するまで、インデックスが作成されたコレクションに対する他のすべてのデータベースオペレーションを

ブロックします。Amazon DocumentDB フォアグラウンドビルドは 5 つのステージで構成されています。

- フォアグラウンド (一意): 単一のドキュメント (一意) フォアグラウンドインデックスビルドは、通常のフォアグラウンドビルドなどの他のデータベースオペレーションをブロックします。基本的なフォアグラウンドビルドとは異なり、単一ビルドでは追加のステージ (ソートキー 2) を使用して重複キーを探します。フォアグラウンド (単一) ビルドは 6 つのステージで構成されます。
- 背景: バックグラウンドインデックスビルドでは、インデックスの作成中に他のデータベースオペレーションをフォアグラウンドで実行できます。Amazon DocumentDB バックグラウンドビルドは 8 つのステージで構成されています。
- 背景 (一意): 単一ドキュメント (一意) の背景インデックスビルドでは、インデックスの作成中に他のデータベースオペレーションをフォアグラウンドで実行できます。基本的なバックグラウンドビルドとは異なり、単一ビルドでは追加のステージ (ソートキー 2) を使用して重複キーを探します。バックグラウンド (単一) ビルドは 9 つのステージで構成されます。

デフォルトのインデックスビルドタイプはフォアグラウンドです。

インデックスをバックグラウンドで構築するには、インデックスの作成時に「バックグラウンド」: true パラメータを追加します。

開始方法

コレクションでインデックスの作成を開始するには、createIndexes コマンドを使用します。デフォルトでは、このコマンドは 2 つの並列ワーカーを実行するため、インデックス作成プロセスの速度が 2 倍向上します。

たとえば、次のコマンドプロセスは、ドキュメント内の「user_name」フィールドのインデックスを作成し、インデックス作成プロセスの速度を 4 ワーカーに上げる方法を示しています。

1. クラスターで 2 つの並列ワーカーを使用して、バックグラウンドでインデックスを作成します。

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx", "background":true }]})
```

2. インデックス作成プロセスの速度を最適化するには、db.runCommand createIndexes コマンドで「ワーカースレッド」オプション (「ワーカー」:<数値>) を使用してワーカーの数を指定できます。

プロセスの速度を 4 つの並列ワーカーに上げます。

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx", "workers":4}]})
```

> **Note**
>
> ワーカーの数が増えるほど、インデックスの作成が速くなります。ただし、ワーカーの数が増えるほど、プライマリインスタンスの vCPU と読み取り IO のロードも大きくなります。他のワークロードを低下させることなく、増加したロードを処理できるよう、クラスターが十分にプロビジョニングされていることを確認してください。

インデックスの進捗状況

インデックス作成プロセスは、初期化、コレクションのスキャン、キーのソート、そして最後にインデックスビルダーによるキーの挿入によって行われます。このプロセスは、フォアグラウンドで実行する場合は最大 6 段階、バックグラウンドで実行する場合は最大 9 段階に分かれます。完了率、スキャンされたストレージブロックの総数、ソートされたキー、挿入されたキーなどのステータスメトリックをステージごとに表示できます。

mongo シェルの db.currentOp() コマンドを使用して、インデックス作成プロセスの進行状況をモニタリングします。最後の段階が 100% 完了すると、すべてのインデックスが正常に作成されたことがわかります。

```
db.currentOp({"command.createIndexes": { $exists : true } })
```

> **Note**
>
> インデックス作成の進行状況の表示は、Amazon DocumentDB 5.0 でのみサポートされています。

インデックスビルドステージ

ステージ	フォアグラウンド	フォアグラウンド (単一)	背景	バックグラウンド (単一)
初期化	1	1	1	1
インデックスの構築: 初期化	2	2	2	2
インデックスの構築: コレクションのスキャン	3	3	3	3
インデックスの構築: ソートキー 1	4	4	4	4
インデックスの構築: ソートキー 2		5		5
インデックスの構築: キーの挿入	5	6	5	6
検証: インデックスのスキャン			6	7
検証: タプルのソート			7	8
検証: コレクションのスキャン			8	9

- 初期化: createIndex はインデックスビルダーを準備しています。このフェーズは非常に短いはずです。
- インデックスの構築: 初期化 - インデックスビルダーはインデックスを作成する準備をしています。このフェーズは非常に短いはずです。

- インデックスの構築: コレクションのスキャン - インデックスビルダーはコレクションスキャンを実行してインデックスキーを収集しています。測定単位は「ブロック」です。

> **Note**
>
> インデックス構築用に複数のワーカーが設定されている場合は、この段階で表示されます。「コレクションのスキャン」ステージは、インデックス構築プロセス中に複数のワーカーを使用する唯一のステージです。その他のすべてのステージには 1 人のワーカーが表示されます。

- インデックスの構築: ソートキー 1 - インデックスビルダーは収集されたインデックスキーをソートしています。測定単位は「キー」です。
- インデックスの構築: ソートキー 2 - インデックスビルダーは、収集されたデッドタプルに対応するインデックスキーをソートします。このフェーズは単一インデックスの構築にのみ適用されます。測定単位は「キー」です。
- インデックスの構築: キーの挿入 - インデックスビルダーは、新しいインデックスにインデックスキーを挿入しています。測定単位は「キー」です。
- 検証: インデックスのスキャン - createIndex はインデックスをスキャンして、検証が必要なキーを見つけます。測定単位は「ブロック」です。
- 検証: タプルのソート - createIndex はインデックススキャンフェーズの出力をソートしています。
- 検証: コレクションのスキャン - createIndex はコレクションをスキャンして、前の 2 つのフェーズで見つかったインデックスキーを検証しています。測定単位は「ブロック」です。

インデックスビルドの出力例

次の出力例 (フォアグラウンドインデックスビルド) では、インデックス作成のステータスが表示されます。「msg」フィールドは、ビルドのステージと完了率を示すことで、ビルドの進行状況を要約します。「ワーカー」フィールドは、インデックスビルドのそのステージで使用されたワーカーの数を示します。「進行状況」フィールドには、完了率の計算に使用される実際の数値が表示されます。

> **Note**
>
> Amazon DocumentDB バージョン 4.0 では、currentIndexBuildName」、「msg」、および「progress」フィールドはサポートされていません。

```
{
    "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
}
```

トラブルシューティングリソース

インデックス作成の問題のトラブルシューティングについては、次のリソースを参照してください。

- インデックス構築が失敗する
- バックグラウンドインデックスのビルドレイテンシーの問題と失敗

インデックスの維持

インデックスの肥大化

Amazon DocumentDB は、複数バージョンの同時実行制御 (MVCC) を使用して同時トランザクションを管理します。ドキュメントを削除または更新しても、以前のバージョンはコレクションとインデックスに「デッド」バージョンとして残ります。ガベージコレクションプロセスは、将来のオペレーションのために、これらのデッドバージョンからスペースを自動的に再利用します。

インデックスの肥大化は、デッドインデックスエントリや古いインデックスエントリの蓄積、またはページ内のフラグメント化により、コレクションのインデックスが大きくなった場合に発生します。レポートされる割合は、将来のインデックスエントリで使用できるインデックススペースの量を表します。この肥大化は、バッファキャッシュとストレージの両方の領域を消費します。肥大化を削除する場合は、インデックスを再構築する必要があります。

例: 次のコマンドを実行して、インデックスの未使用のストレージを特定します。

```
db.coll.aggregate({$indexStats:{}});
```

結果は以下のようになります。

```
{
    "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
    }
}
```

reIndex コマンドを使用すると、ダウンタイムなしでインデックスを再構築できます。このコマンドでは、コレクション全体のスキャンが必要です。reIndex を使用したインデックスメンテナンスを参照してください。

reIndex を使用したインデックスメンテナンス

reIndex は、インデックスを再構築するために使用されるコマンドです。通常、インデックスが破損したり、または非効率的になった場合に使用されます。時間の経過とともに、インデックスには多くの更新、挿入、削除に伴い未使用のスペースが蓄積され、パフォーマンスが低下する恐れがあります。インデックスの再構築は、このような未使用のスペースを削除し、インデックスの効率を回復するのに役立ちます。

reIndex ガイドライン

- reIndex は Amazon DocumentDB 5.0 以降でのみサポートされています
- reIndex は常にバックグラウンドで実行されます。
- Amazon DocumentDB は、バックグラウンドで 1 つのインデックスの再インデックスをサポートしているため、複数のワーカーに対応できます。古いインデックスは、reIndex プロセスの実行時にクエリで使用できます。
- Amazon DocumentDB は currentOp によるインデックス作成の進行状況レポートをサポートしています。インデックス作成時に表示されるインデックスビルドステージと同様のインデックスビルドステージを表示できます。唯一の違いは、reIndex には一意かどうかにかかわらず、常に 8 つのステージがあることです。「ビルドインデックス: ソートキー 2」ステージはありません。
- reIndex は、createIndexes、collMod、renameCollection の同じコレクションでインデックス関連のコマンドを除く任意のコマンドと同時に実行できます。 dropIndexes
- reIndex は現在、テキストインデックス、地理空間インデックス、ベクトルインデックスではサポートされていません。
- reIndex 部分インデックスのサポートは、エンジンバージョン 8.0.2 以降で利用できます。

reIndex ビルド

次のコマンドを使用してインデックスを再構築します。

```
db.runCommand({ reIndex: "collection-name", index: "index-name"})
```

オプションで、再構築プロセスに割り当てるワーカーの数をコントロールすることもできます。

```
db.runCommand({ reIndex: "collection-name", index: "index-name", workers: number })
```

Java でのインデックスの管理に固有の情報については、「」を参照してくださいJava による Amazon DocumentDB でのインデックス管理。

インデックスを使用するためのベストプラクティス

インデックスの構築

Amazon DocumentDB にデータをインポートする場合は、大きなデータセットをインポートする前にインデックスを作成する必要があります。Amazon DocumentDB インデックスツールを使用して、実行中の MongoDB インスタンスまたは mongodump ディレクトリからインデックスを抽出し、Amazon DocumentDB クラスターにそれらのインデックスを作成できます。移行の詳細については、移行とアップグレード を参照してください。

インデックスの選択性

インデックスの作成を、重複する値の数がコレクション内のドキュメントの合計数の 1% 未満のフィールドに制限します。たとえば、コレクションに 100,000 個のドキュメントが含まれている場合、同じ値が 1,000 回以下のフィールドにのみインデックスを作成します。

一意の値が多いインデックス (つまり、カーディナリティが高い) を選択すると、フィルターオペレーションが少数のドキュメントを返すため、インデックススキャン中に良好なパフォーマンスが得られます。高濃度インデックスの例は一意のインデックスです。これにより、等式の述語が最大で 1 つのドキュメントを返すことが保証されます。低濃度の例としては、ブール型フィールドのインデックスと、曜日別のインデックスなどがあります。パフォーマンスが低いため、データベースのクエリオプティマイザによって低カーディナリティインデックスが選択されることはほとんどありません。同時に、低濃度のインデックスは、ディスク領域や I/O などのリソースを消費し続けます。経験則として、標準値の頻度がコレクション全体のサイズの 1% 以下のフィールドのインデックスをターゲットにしてください。

さらに、よくフィルターとして使用されるフィールドに対してのみインデックスを作成し、未使用のインデックスを定期的に検索することを推奨します。詳細については、「the section called “インデックスの使用状況を分析し、未使用のインデックスを特定します。”」を参照してください。

インデックスがデータの書き込みに与える影響

インデックスを使用すると、コレクション内のすべてのドキュメントをスキャンする必要がなくなるためクエリのパフォーマンスが向上しますが、このメリットにはトレードオフがあります。コレクションのインデックスごとに、ドキュメントが挿入、更新、または削除されるたびに、データベースはコレクションを更新し、コレクションの各インデックスにフィールドを書き込む必要があります。例えば、コレクションに 9 つのインデックスがある場合、データベースはクライアントへのオペレーションを承認する前に 10 回の書き込みを実行する必要があります。したがって、インデックスを追加するたびに、書き込みレイテンシー、I/O、および使用率の高いストレージ全体の増加が発生します。

クラスターインスタンスは、すべてのワーキングセットメモリを保持するために適切なサイズにする必要があります。これにより、インデックスページをストレージボリュームから継続的に読み取る必要がなくなるため、パフォーマンスに悪影響を及ぼし、I/O コストが高くなります。詳細については、「インスタンスのサイズ指定」を参照してください。

最高のパフォーマンスを得るには、コレクション内のインデックスの数を最小限に抑え、一般的なクエリのパフォーマンスを高めるために必要なインデックスのみを追加します。ワークロードはさまざまですが、ガイドラインとして、コレクションあたりのインデックス数を 5 つ以下に抑えることを推奨します。

欠落しているインデックスの識別

ベストプラクティスとして、欠落しているインデックスを定期的に特定します。詳細については、「the section called “見逃したインデックスの特定”」を参照してください。

使用されていないインデックスの識別

ベストプラクティスとして、未使用のインデックスを定期的に特定して削除します。詳細については、「the section called “インデックスの使用状況を分析し、未使用のインデックスを特定します。”」を参照してください。

インデックスタイプ

トピック

- 単一フィールドインデックス
- 複合インデックス
- マルチキーインデックス
- テキストインデックス
- 地理空間インデックス
- ベクトルインデックス

単一フィールドインデックス

単一フィールドインデックスは、ドキュメントのコレクション内の単一フィールドからの情報を保存します。デフォルトでは、すべてのコレクションには `_id`フィールドにインデックスがあります。インデックスを追加して、重要なクエリとオペレーションを高速化できます。

単一フィールドインデックスは、次の場合に有益です。

- アプリケーションは、特定のフィールドの値に基づいてドキュメントを頻繁にクエリまたはフィルタリングします。
- 特定のフィールドでドキュメントを効率的にソートする必要があります。
- 一意のインデックスを作成して、コレクション内のドキュメント間で特定のフィールドの一意性を確保したい。

サポートされているインデックスプロパティ

オプション	3.6	4.0	5.0	8.0	Elastic クラスター
name	はい	はい	はい	はい	はい
一意	はい	はい	はい	はい	はい
スパース *	はい	はい	はい	はい	はい
partialFilterExpression *	いいえ	なし	はい	はい	いいえ

オプション	3.6	4.0	5.0	8.0	Elastic クラスター
expireAfterSeconds	はい	はい	はい	はい	はい

* sparseおよび partialFilterExpressionオプションは、同じインデックス定義で一緒に使用することはできません。これらのオプションを使用してインデックスを作成しようとすると、次のエラーで失敗します。

```
Error in specification: cannot mix partialFilterExpression and sparse options
```

単一のフィールドインデックスの作成

createIndex() メソッドを使用して、単一のフィールドインデックスを作成します。構文は次のとおりです：db.collection.createIndex(<key>, <options>)

key パラメータは、 フィールドとインデックスのソート順序を指定する JSON ドキュメントです。

```
{
  "<field>": <1 (ascending)|-1 (descending)>
}
```

options パラメータは、インデックスのオプションを指定する JSON ドキュメントです。

```
{
  "name": "<name>",
  "unique": <true | false>,
  "sparse": <true | false>,
  "partialFilterExpression": <filter expression>,
  "expireAfterSeconds": <seconds before expiry>
}
```

単一フィールドインデックスの作成例については、「インデックスプロパティ」を参照してください。

複合インデックス

複合インデックスは、ドキュメントのコレクション内の 2 つ以上のフィールドからの情報を保存し、最初のフィールドまたはプレフィックスフィールドに対するクエリを許可します。これらのイン

デックスは、複数のフィールドを同時にフィルタリングするか、フィルタリングとソート操作を組み合わせるクエリのパフォーマンスを最適化します。また、左端のインデックス付きフィールドに対する単一条件クエリにも有効です。データベースは、これらのインデックスエントリを活用して、完全なコレクションスキャンを実行せずに、一致するドキュメントを効率的に見つけます。

複合フィールドインデックスは、次の場合に有益です。

- 複数のフィールドを同時にフィルタリングする必要があります。
- フィルタリングとソートオペレーションを組み合わせる必要があります。

サポートされているインデックスプロパティ

オプション	3.6	4.0	5.0	8.0	Elastic クラスター
name	はい	はい	はい	はい	はい
一意	はい	はい	はい	はい	はい
スパース *	はい	はい	はい	はい	はい
partialFilterExpression *	いいえ	なし	はい	はい	いいえ

* sparseおよび partialFilterExpressionオプションは、同じインデックス定義で一緒に使用することはできません。これらのオプションを使用してインデックスを作成しようとすると、次のエラーで失敗します。

```
Error in specification: cannot mix partialFilterExpression and sparse options
```

複合インデックスの作成

createIndex() メソッドを使用して複合インデックスを作成します。メソッド構文は次のとおりです。 db.collection.createIndex(<keys>, <options>)

keys パラメータは、フィールドとインデックスのソート順序を指定する JSON ドキュメントです。

```
{
  "<field 1>": <1 (ascending)|-1 (descending)>,
  "<field 2>": <1 (ascending)|-1 (descending)>,
  ...
}
```

複合インデックスの配列には 1 つのフィールドしか使用できないことに注意してください。2 つ以上の配列フィールドに複合インデックスを作成しようとすると、次のエラーで失敗します。

```
multiple fields of compound index cannot be arrays
```

options パラメータは、インデックスのオプションを指定する JSON ドキュメントです。

```
{
  "name": "<name>",
  "unique": <true | false>,
  "sparse": <true | false>,
  "partialFilterExpression": <filter expression>
}
```

複合インデックスの作成例については、「インデックスプロパティ」を参照してください。

マルチキーインデックス

配列値を持つフィールドの場合、マルチキーインデックスを使用すると、配列内の各要素のインデックスキーを作成できます。配列のインデックスを作成すると、配列の各要素のインデックスエントリが作成されます。

マルチキーインデックスは、アプリケーションが配列の値に基づいてドキュメントを頻繁にクエリまたはフィルタリングする場合に便利です。

サポートされているインデックスプロパティ

オプション	3.6	4.0	5.0	8.0	Elastic クラスター
name	はい	はい	はい	はい	はい
一意	はい	はい	はい	はい	はい

オプション	3.6	4.0	5.0	8.0	Elastic クラスター
スパース *	はい	はい	はい	はい	はい
partialFilterExpression *	いいえ	なし	はい	はい	いいえ
expireAfterSeconds	はい	はい	はい	はい	はい

* sparseおよび partialFilterExpressionオプションは、同じインデックス定義で一緒に使用することはできません。これらのオプションを使用してインデックスを作成しようとすると、次のエラーで失敗します。

```
Error in specification: cannot mix partialFilterExpression and sparse options
```

マルチキーインデックスの作成

createIndex() メソッドを使用して、マルチキーインデックスを作成します。メソッド構文は次のとおりです。 db.collection.createIndex(<key>, <options>)

key パラメータは、 フィールドとインデックスのソート順序を指定する JSON ドキュメントです。

```
{
  "<field>": <1 (ascending)|-1 (descending)>
}
```

options パラメータは、インデックスのオプションを指定する JSON ドキュメントです。

```
{
  "name": "<name>",
  "unique": <true | false>,
  "sparse": <true | false>,
  "partialFilterExpression": <filter expression>,
  "expireAfterSeconds": <seconds before expiry>
}
```

次の例では、 という名前の昇順でソートされたcategoriesフィールドにマルチキーインデックスを作成しますbook_categories。

```
db.collection.createIndex(
  {
    "categories": 1
  },
  {
    "name": "book_categories"
  }
)
```

マルチキーインデックスの作成例については、「インデックスプロパティ」を参照してください。

テキストインデックス

テキストインデックスは、ドキュメント内のテキストフィールドで単語やフレーズを検索するのに役立ち、全文検索機能を実行できます。1 つのフィールドのテキストインデックスまたは複数のテキストフィールドを持つ複合テキストインデックスを作成できます。ただし、コレクションごとに 1 つのテキストインデックスのみを持つことができます。詳細については、「Amazon DocumentDB でのテキスト検索の実行」を参照してください。

サポートされているインデックスプロパティ

オプション	3.6	4.0	5.0	8.0	Elastic クラスター
name	いいえ	なし	はい	はい	いいえ

テキストインデックスの作成

createIndex() メソッドを使用してテキストインデックスを作成します。メソッド構文は次のとおりです。 db.collection.createIndex(<keys>, <options>)

keys パラメータは、フィールド (複数可) とテキストインデックスタイプを指定する JSON ドキュメントです。

```
{
  "<field 1>": "text",
  "<field 2>": "text"
```

```
}
```

options パラメータは、インデックスの名前を指定する JSON ドキュメントです。

```
{
  "name": "<name>"
}
```

テキストインデックスの作成例については、「インデックスプロパティ」を参照してください。

地理空間インデックス

地理空間インデックスは、ドキュメントのコレクションに保存されている地理空間データを効率的にクエリおよび管理するために設計された特殊なタイプのインデックスです。Amazon DocumentDB は、球体 (地球など) 上の地理空間データを処理するように特別に設計された 2 次元インデックスをサポートしています。これにより、球ジオメトリに基づく正確な計算とクエリが可能になります。

地理空間インデックスは、アプリケーションが次のような場所ベースのクエリを実行する必要がある場合に役立ちます。

- 近くの関心のあるポイントを見つける
- ロケーションが特定のエリア内にあるかどうかを判断する
- ロケーション間の距離の計算

サポートされているインデックスプロパティ

オプション	3.6	4.0	5.0	8.0	Elastic クラスター
name	はい	はい	はい	はい	はい

地理空間インデックスの作成

createIndex() メソッドを使用して地理空間インデックスを作成します。メソッド構文は次のとおりです。 db.collection.createIndex(<key>, <options>)

key パラメータは、 フィールドと 2dsphere インデックスタイプを指定する JSON ドキュメントです。

```
{
  "<field>": "2dsphere"
}
```

options パラメータは、インデックスのオプションを指定する JSON ドキュメントです。

```
{
  "name": "<name>"
}
```

地理空間インデックスの作成例については、「インデックスプロパティ」を参照してください。

ベクトルインデックス

ベクトルインデックスは、ドキュメントのコレクションに保存されているベクトルデータを効率的にクエリおよび管理するために設計された特殊なタイプのインデックスです。Amazon DocumentDB は、階層ナビゲーション可能スモールワールド (HNSW) とフラット圧縮 (IVFFlat) インデックスを持つ反転ファイルをサポートしています。

詳細については、「Amazon DocumentDB のベクトル検索」を参照してください。

ベクトルインデックスは、次のような機械学習や生成 AI のユースケースに役立ちます。

- セマンティック検索
- 製品のレコメンデーション
- パーソナライゼーション
- チャットボット
- 不正検出
- 異常検出

サポートされているインデックスプロパティ

オプション	3.6	4.0	5.0	8.0	Elastic クラスター
name	いいえ	なし	はい	はい	いいえ

ベクトルインデックスの作成

runCommand() メソッドで createIndex コマンドを使用して、ベクトルインデックスを作成します。構文は次のとおりです。

```
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]
    }
  }]
})
```

キーパラメータは、フィールドとベクトルインデックスタイプを指定する JSON ドキュメントです。

```
{
  "<field>": "vector"
}
```

ベクトルインデックスの作成例については、「インデックスプロパティ」を参照してください。

インデックスプロパティ

インデックスプロパティは、Amazon DocumentDB のインデックスの動作と特性を定義します。これらのプロパティを理解することで、アプリケーション固有の要件を満たす効率的なインデックスを設計できます。

トピック

- インデックスプロパティ: name

- インデックスプロパティ: unique
- インデックスプロパティ: スパース
- インデックスプロパティ: partialFilterExpression
- インデックスプロパティ: expireAfterSeconds
- インデックスプロパティ: hidden

インデックスプロパティ: name

サポートされているインデックスタイプ

オプション	3.6	4.0	5.0	8.0	Elastic クラスター
単一フィールド	はい	はい	はい	はい	はい
compound	はい	はい	はい	はい	はい
マルチキー	はい	はい	はい	はい	はい
テキスト	いいえ	なし	はい	はい	いいえ
地理空間	はい	はい	はい	はい	はい
ベクトル	いいえ	なし	はい	はい	いいえ

name オプションを使用して、インデックスのオプションの名前を指定します。

すべての例では、次のサンプルドキュメントを使用します。

```
{
  "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')
}
```

単一フィールド

```
db.collection.createIndex(
  {
    "productId": 1
  },
  {
    "name": "single_field_index"
  }
)
```

複合

```
db.collection.createIndex(
  {
    "productId": 1,
    "manufacturer": 1
  },
  {
    "name": "compound_index"
  }
)
```

マルチキー

```
db.collection.createIndex(
  {
    "tags": 1
  },
  {
    "name": "multikey_index"
  }
)
```

テキスト

```
db.collection.createIndex(
  {
    "name": "text"
  },
  {
    "name": "text_index"
  }
)
```

地理空間

```
db.collection.createIndex(
  {
    "supplier.location": "2dsphere"
  },
  {
    "name": "geospatial_index"
  }
)
```

Vector

```
db.runCommand({
  "createIndexes": "collection",
  "indexes": [{
    "key": {
      "productEmbedding": "vector"
    },
    "name": "hnsw_index",
```

```
    "vectorOptions": {
      "type": "hnsw",
      "dimensions": 2,
      "similarity": "euclidean",
      "m": 16,
      "efConstruction": 64
    }
  }]
})
```

インデックスプロパティ: unique

サポートされているインデックスタイプ

オプション	3.6	4.0	5.0	8.0	Elastic クラスター
単一フィールド	はい	はい	はい	はい	はい
compound	はい	はい	はい	はい	はい
マルチキー	はい	はい	はい	はい	はい

一意のオプションを使用して、コレクション内のドキュメント間でフィールド (複数可) の一意性を確保します。

例

次の例は、次のサンプルドキュメントで一意のインデックスを作成する方法を示しています。

```
{
  "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')
}
```

単一フィールド

productId に一意の単一フィールドインデックスを作成して、同じ productId が複数のドキュメントに存在しないようにします。

```
db.collection.createIndex(
  {
    "productId": 1
  },
  {
    name: "productId_unique",
    unique: true
  }
)
```

複合

sku と製造元に一意の複合インデックスを作成し、sku と製造元の同じ組み合わせが複数のドキュメントに存在しないようにします。

```
db.collection.createIndex(
  {
    "sku": 1,
```

```
    "manufacturer": 1
  },
  {
    name: "sku_and_manufacturer_unique",
    unique: true
  }
)
```

マルチキー

バーコードに一意のマルチキーインデックスを作成して、バーコード配列の値が複数のドキュメントに存在しないようにします。

```
db.collection.createIndex(
  {
    "barcodes": 1
  },
  {
    name: "barcodes_unique",
    unique: true
  }
)
```

配列のインデックスを作成すると、配列の各要素のインデックスエントリが作成されます。たとえば、配列に 50 個の項目がある場合、50 個のインデックスエントリがあります。このため、一意のマルチキーインデックスは、すべての個々の項目に一意性を適用します。たとえば、次のドキュメントは、値配列フィールドインデックスの一意の制約に違反します。

```
{ "values": [ 1, 2, 3] }
{ "values": [ 3, 2 ] }   --> 3 and 2 already exist
{ "values": [ 1 ] }      --> 1 already exists
```

一意のインデックスでは、次の動作に注意してください。

1. 2 つ (または複数) のドキュメントのインデックス付きフィールドの値が同じである既存のデータに一意のインデックスを作成すると、インデックスの構築は失敗し、次のエラーが発生します。 `could not create unique index: <collection> index: <index name>`
2. インデックス付きフィールドの値が別のドキュメントのそのフィールドの値と一致するドキュメントを挿入すると、挿入は次のエラーで失敗します。 `E11000 duplicate key error collection: <collection> index: <index name>`

3. インデックス付きフィールドの新しい値が別のドキュメントのそのフィールドの値と一致するように既存のドキュメントを更新すると、更新は次のエラーで失敗します。 `E11000 duplicate key error collection: <collection> index: <index name>`
4. インデックス付きフィールドがドキュメントにない場合、値は null として扱われます。インデックスの作成、挿入、更新は、インデックス付きフィールドが 2 つ (または複数) のドキュメントにない場合、上記のように失敗します。

インデックスプロパティ: スパース

サポートされているインデックスタイプ

オプション	3.6	4.0	5.0	8.0	Elastic クラスター
単一フィールド	はい	はい	はい	はい	はい
compound	はい	はい	はい	はい	はい
マルチキー	はい	はい	はい	はい	はい

スパースオプションを使用して、インデックス付きフィールド (複数可) がないドキュメントのインデックス作成をスキップし、インデックスサイズを減らし、メモリ内のスペースを節約します。インデックスサイズが小さいため、インデックスを使用するクエリの方が効率的です。スパースインデックスを利用するクエリでは、インデックス付きフィールドで $exists 句を使用する必要があります。$exists 句を省略した場合、Amazon DocumentDB はスパースインデックスを使用しません。

例

次の例は、次のサンプルドキュメントでスパースインデックスを作成する方法を示しています。

```
{
  "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')
}
```

レビュー、マテリアル、および色フィールドは、すべてのドキュメントに存在するわけではないことに注意してください。

単一フィールド

マテリアルフィールドにスパース単一フィールドインデックスを作成します。

```
db.collection.createIndex(
  {
    "material": 1
  },
  {
    "name": "material_sparse",
    "sparse": true
  }
)
```

このインデックスは、マテリアルの値を持つすべての製品を検索するときに使用されます。

```
db.collection.find({
  "material": {
```

```
    $exists: true
  }
})
```

複合

マテリアルフィールドとカラーフィールドにスパース複合インデックスを作成します。

```
db.collection.createIndex(
  {
    "material": 1,
    "color": 1
  },
  {
    "name": "material_and_color_sparse",
    "sparse": true
  }
)
```

このインデックスは、マテリアル値と色値を組み合わせたすべての製品を検索するときに使用されます。

```
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
      }
    }
  ]
})
```

マルチキー

Reviews 配列にスパースマルチキーインデックスを作成します。

```
db.collection.createIndex(
  {
    "reviews": 1
  },
  {
    "name": "reviews_sparse",
    "sparse": true
  }
)
```

このインデックスは、レビュー配列を持つすべての製品を検索するときに使用されます。

```
db.collection.find({
  "reviews": {
    $exists: true
  }
})
```

インデックスプロパティ: partialFilterExpression

サポートされているインデックスタイプ

オプション	3.6	4.0	5.0	8.0	Elastic クラスター
単一フィールド	いいえ	なし	はい	はい	いいえ
compound	いいえ	なし	はい	はい	いいえ
マルチキー	いいえ	なし	はい	はい	いいえ
地理空間	いいえ	いいえ	いいえ	いいえ	いいえ

partialFilterExpression オプションを使用して、指定されたフィルター条件を満たすドキュメントのみを含む部分インデックスを作成します。これにより、すべてのドキュメントのインデックスを作成するのではなく、コレクション内のドキュメントのサブセットのみをインデックス化して、インデックスサイズを減らし、メモリ内のスペースを節約することで、より効率的なインデックスを作成できます。インデックスサイズが小さいため、インデックスを使用するクエリの方が効率的です。Amazon DocumentDB は、次のシナリオで部分インデックスを使用します。

- クエリ述語は、部分インデックスフィルター式と完全に一致します。
- クエリフィルターの期待される結果は、部分フィルターの論理サブセットです。
- クエリのサブ述語は、他のインデックスと組み合わせて使用できます。

詳細については、「部分インデックス」を参照してください。

例

次の例は、次のサンプルドキュメントで部分インデックスを作成する方法を示しています。

```
{
  "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')
}
```

単一フィールド

Space Gray の色を持つ製品の製造元に部分的な単一フィールドインデックスを作成します。

```
db.collection.createIndex(
  {
    "manufacturer": 1
  },
  {
    "name": "manufacturer_space_gray",
    "partialFilterExpression": {
      "color": {
        $eq: "Space Gray"
      }
    }
  }
)
```

このインデックスは、製造元によって Space Gray カラー製品を検索するときに使用されます。

```
db.collection.find({
  "manufacturer": "The Manufacturer",
  "color": {
    $eq: "Space Gray"
  }
})
```

複合

Poly Carbonate のマテリアルを持つ製品の製造元と色に部分的な複合インデックスを作成します。

```
db.collection.createIndex(
  {
    "manufacturer": 1,
    "color": 1
  },
  {
    "name": "manufacturer_and_color_polycarbonate",
    "partialFilterExpression": {
      "material": {
        $eq: "Polycarbonate"
      }
    }
  }
)
```

このインデックスは、製造元と色別にポリカーボネイトマテリアル製品を検索するときに使用されます。

```
db.collection.find({
  "manufacturer": "The Manufacturer",
  "color": "Space Gray",
  "material": {
    $eq: "Polycarbonate"
  }
})
```

マルチキー

製造元がある製品のタグに部分的なマルチキーインデックスを作成します。

```
db.collection.createIndex(
```

```
  {
    "tags": 1
  },
  {
    "name": "tags_the_manufacturer",
    "partialFilterExpression": {
      "manufacturer": {
        $eq: "The Manufacturer"
      }
    }
  }
)
```

このインデックスは、製造元があるプリンターでタグ付けされた製品を検索するときに使用されます。

```
db.collection.find({
  "tags": "printer",
  "manufacturer": {
    $eq: "The Manufacturer"
  }
})
```

インデックスプロパティ: expireAfterSeconds

サポートされているインデックスタイプ

インデックスタイプ	3.6	4.0	5.0	8.0	Elastic クラスター
単一フィールド	はい	はい	はい	はい	はい
マルチキー	はい	はい	はい	はい	はい

expireAfterSeconds オプションを使用して、有効期限 (TTL) インデックスを作成します。TTL インデックスを使用すると、各ドキュメントのタイムアウト条件に基づいて、経過時間に基づいてドキュメントを削除できます。ドキュメントが指定された TTL 経過時間に達すると、そのドキュメントはコレクションから削除されます。

TTL 削除のベストプラクティスについては、「」を参照してくださいTTL および時系列ワークロード。

例

次の例は、次のサンプルドキュメントで TTL インデックスを作成する方法を示しています。

```
{
  "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')
}
```

単一フィールド

lastUpdated に TTL インデックスを作成し、90 日以内に更新されていないドキュメントを削除します。

```
db.collection.createIndex(
  {
    "lastUpdated": 1
  },
  {
    "name": "lastUpdated_ttl",
    "expireAfterSeconds": 7776000
  }
)
```

マルチキー

reviews.review_date に TTL インデックスを作成して、過去 1 年間にレビューがなかったドキュメントを削除します。

```
db.collection.createIndex(
  {
    "reviews.review_date": 1
  },
  {
    "name": "reviews_review_date_ttl",
    "expireAfterSeconds": 31536000
  }
)
```

TTL インデックスが配列フィールドにある場合、配列内のすべての項目がチェックされることに注意してください。配列内のタイムスタンプがタイムアウト条件を満たしている場合、ドキュメントは削除されます。

インデックスプロパティ: hidden

Amazon DocumentDB は、Amazon DocumentDB 8.0.1 以降および Amazon DocumentDB 5.0.2 以降で始まるhiddenインデックスプロパティをサポートしています。

非表示インデックスはクエリプランナーに表示されず、クエリのサポートには使用できません。プランナーからインデックスを非表示にすることで、実際にインデックスを削除することなく、インデックスを削除した場合の潜在的な影響を評価できます。影響が負の場合は、ドロップされたインデックスを再作成する代わりに、インデックスを非表示にすることができます。

hidden プロパティは、単一フィールド、複合、マルチキー、テキスト、地理空間、ベクトルインデックスなど、すべてのインデックスタイプでサポートされています。

動作

プランナーから非表示になる以外にも、非表示インデックスは完全に維持され、適用されます。例えば、次のようになります。

- 非表示のインデックスが一意のインデックスである場合も、インデックスはその一意の制約をドキュメントに適用します。
- 非表示のインデックスが TTL インデックスの場合、インデックスはドキュメントを期限切れにします。
- 非表示インデックスは listIndexesおよび db.collection.getIndexes()結果に含まれます。
- 非表示インデックスは、コレクションへの書き込みオペレーションで更新され、引き続きディスク容量とメモリを消費します。その結果、 db.collection.stats()や などの統計オペレーションに含まれます$indexStats。
- 非表示のインデックスを非表示にするか、非表示のインデックスを非表示にすると、 は をリセットします$indexStats。既に非表示のインデックスを非表示にしたり、既に非表示のインデックスを非表示にしたりしても、 はリセットされません$indexStats。

制限事項

- _id インデックスを非表示にすることはできません。
- cursor.hint() を使用して非表示インデックスを強制的に使用することはできません。
- $merge onフィールドで必要な一意のインデックスが非表示の場合、 は失敗します。を実行する前にインデックスを表示します$merge。

例

非表示インデックスを作成する

非表示状態のインデックスを作成するには、 db.collection.createIndex()メソッドを使用し、インデックス定義trueで hiddenオプションを に設定します。

たとえば、次のオペレーションでは、 productIdフィールドに非表示の昇順インデックスを作成します。

```
db.collection.createIndex(
  { "productId": 1 },
  { "name": "productId_1", "hidden": true }
)
```

検証するには、コレクションlistIndexesで db.collection.getIndexes()または を実行します。非表示インデックスの定義"hidden": trueには が含まれます。

```
db.collection.getIndexes()
[
  {
    "v": 4,
    "key": { "_id": 1 },
    "name": "_id_"
  },
  {
    "v": 4,
    "key": { "productId": 1 },
    "name": "productId_1",
    "hidden": true
  }
]
```

インデックスオプションhiddenは、値が の場合にのみ返されますtrue。

既存のインデックスを非表示にする

collMod コマンドを使用して、既存のインデックスを非表示にします。インデックスをその nameまたは で識別しkeyPattern、 hiddenを に設定しますtrue。

次の例では、名前でインデックスを非表示にします。

```
db.runCommand({
  collMod: "collection",
  index: {
    name: "productId_1", // Specify the index name
    hidden: true
  }
})
```

キーパターンでインデックスを識別することもできます。

```
db.runCommand({
  collMod: "collection",
  index: {
    keyPattern: { productId: 1 }, // Specify the index key specification document
    hidden: true
  }
})
```

既存のインデックスの非表示を解除する

非表示のインデックスを再度表示するには、 を hiddenに設定して collMod コマンドを使用しますfalse。

```
db.runCommand({
  collMod: "collection",
  index: { name: "productId_1", hidden: false }
})
```

インデックスは非表示のまま完全に維持されるため、非表示にするとすぐに使用できます。

Amazon DocumentDB を用いた変更ストリームの使用

Amazon DocumentDB (MongoDB 互換) の変更ストリーム機能は、クラスターのコレクション内で発生する変更イベントの時系列シーケンスを提供します。変更ストリームからイベントを読み取って、次のようなさまざまなユースケースを実装できます。

- 変更通知
- Amazon OpenSearch Service (OpenSearch Service) を用いたフルテキスト検索
- Amazon Redshift を用いてのイベントの分析

アプリケーションは変更ストリームを使用して、個々のコレクションでデータ変更をサブスクライブすることができます。変更ストリームイベントは、クラスターで発生する際に順序付けされ、イベントが記録されてから 3 時間 (デフォルト) 保存されます。change_stream_log_retention_duration のパラメータを使って、保存期間を7日まで延長できます。変更ストリームの保持期間を変更するには、「」を参照してください変更ストリームログの保持期間の変更。

トピック

- サポートされているオペレーション
- 拡張イベント
- 料金
- 制限事項
- 変更ストリームの有効化
- 例: Python での変更ストリームの使用
- ドキュメント全体の参照
- $changeStreamSplitLargeEvent を使用して大きなイベントをフラグメントに分割する
- 変更ストリームの再開
- startAtOperationTime を用いた変更ストリームの再開
- postBatchResumeToken を用いた変更ストリームの再開
- 変更ストリーム内のトランザクション
- 変更ストリームログの保持期間の変更
- セカンダリインスタンスでの変更ストリームの使用

サポートされているオペレーション

Amazon DocumentDB では、変更ストリームに対して次の操作がサポートされています。

- すべての変更イベントは、MongoDB の db.collection.watch()、db.watch()、および client.watch() のAPI 中でサポートされている全ての変更イベント。
- 更新のための完全ドキュメント参照。
- 集約ステージ: $match、$project、、$redact、$addFields$replaceRoot、および $changeStreamSplitLargeEvent。
- 再開トークンからの変更ストリームの再開

- `startAtOperation` (Amazon DocumentDB v4.0+ に適用される) を使用して、タイムスタンプから変更ストリームを再開する
- `showExpandedEvents` パラメータ`createCollection`を使用するための DDL イベント通知 (エンジンバージョン 8.0.2 以降の Amazon DocumentDB 8.0 に適用) 。
- `showExpandedEvents` パラメータ`createIndexes`を使用するための DDL イベント通知 (エンジンバージョン 8.0.2 以降の Amazon DocumentDB 8.0 に適用) 。

拡張イベント

> **Note**
>
> 拡張イベントは、エンジンバージョン 8.0.2 以降の Amazon DocumentDB 8.0 で使用できます。

変更ストリームは、データ定義言語 (DDL) イベント通知、`createCollection`および をサポートします`createIndexes`。拡張イベントを含めるには、 `showExpandedEvents`オプションを使用して変更ストリームカーソルを開きます。

例えば、次のようになります。

mongosh Method

```
let cur = db.names.watch( [ ], {
   showExpandedEvents: true
} )
cur.next()
```

Aggregation Stage

```
let cur = db.names.aggregate( [
   { $changeStream: { showExpandedEvents: true } }
] )
cur.next()
```

料金

Amazon DocumentDB 変更ストリーム機能はデフォルトで無効になっており、この機能が有効化されるまで追加料金は発生しません。クラスターで変更ストリームを使用すると、追加の読み取り、および書き込み IO とストレージのコストが発生します。`modifyChangeStreams` の API オペレーションを使用して、クラスターのためのこの機能を有効にできます。料金のさらなる詳細については、Amazon DocumentDB の料金 を参照してください。

制限事項

変更ストリームには、Amazon DocumentDB で以下の制限があります。

- Amazon DocumentDB 3.6. および Amazon DocumentDB 4.0 では、変更ストリームは Amazon DocumentDB クラスターのプライマリインスタンスへの接続からのみ開くことができます。レプリカインスタンス上の変更ストリームからの読み取りは、Amazon DocumentDB 3.6 および Amazon DocumentDB 4.0 ではサポートされていません。`watch()` API オペレーションを呼び出すときは、すべての読み取りがプライマリインスタンスに送信されるようにする `primary` の読み取り設定を指定する必要があります (「例」セクションを参照)。
- Amazon DocumentDB 5.0 では、グローバルクラスターを含むプライマリインスタンスとセカンダリインスタンスの両方から変更ストリームを開くことができます。セカンダリ読み取り設定を指定して、変更ストリームをセカンダリインスタンスにリダイレクトできます。その他のベストプラクティスと制限事項については、「セカンダリインスタンスでの変更ストリームの使用」を参照してください。
- コレクションの変更ストリームに書き込まれるイベントは、最大 7 日間 (デフォルトは 3 時間) 使用できます。変更ストリームのデータは、新しい変更が行われていない場合でも、ログの保持期間が終了すると削除されます。
- `updateMany` や `deleteMany` のようなコレクションで長時間実行されている書き込みオペレーションは、長時間実行されている書き込みオペレーションが完了するまで、変更ストリームイベントの書き込みを一時的に停止する可能性があります。
- Amazon DocumentDB は、MongoDB オペレーションログ (oplog) をサポートしていません。
- Amazon DocumentDB を用いて、特定のコレクションで変更ストリームを明示的に有効にする必要があります。
- 変更ストリームイベントの合計サイズ (リクエストされた場合、変更データと完全なドキュメントを含む) が より大きい場合16 MB、クライアントは変更ストリームで読み取り失敗が発生します。これを回避するには、 `$changeStreamSplitLargeEvent`ステージを使用してイベントをフラグメントに分割します。

- Ruby ドライバは、現在、Amazon DocumentDB v3.6 での db.watch() と client.watch() の使用時にはサポートされていません。
- フィールドの更新値が以前の値と同じ場合、Amazon DocumentDB と MongoDB では変更ストリームの updateDescription コマンドからの出力が異なります。
  - 提供されるフィールドが $set コマンド内で指定されており、かつターゲット値が既にソース値と等しい場合、Amazon DocumentDB は updateDescription 出力内でフィールドを返しません。
  - MongoDB は、指定された値が現在の値と等しい場合でも、出力内でフィールドを返します。

変更ストリームの有効化

Amazon DocumentDB 変更ストリームは、特定のデータベース内のすべてのコレクションに対して有効にすることも、選択したコレクションに対してのみ有効にすることもできます。以下は、mongo シェルを使用してさまざまなユースケースの変更ストリームを有効にする方法の例です。データベース名とコレクション名を指定する場合、空の文字列はワイルドカードとして扱われます。

```
//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});
```

次のいずれかに該当する場合、変更ストリームはコレクションに対して有効になります。

- データベースとコレクションの両方が明示的に有効になっている。
- コレクションを含むデータベースが有効になっている。
- すべてのデータベースが有効になっている。

親データベースでも変更ストリームが有効になっている場合、またはクラスター内のすべてのデータベースが有効になっている場合は、データベースからコレクションを削除しても、そのコレクションの変更ストリームは無効になりません。削除されたコレクションと同じ名前で新しいコレクションが作成された場合、そのコレクションに対して変更ストリームが有効になります。

`$listChangeStreams` 集約パイプラインステージを使用すると、クラスターで有効なすべての変更ストリームを一覧表示できます。Amazon DocumentDB でサポートされているすべての集約ステージは、パイプラインで追加の処理に使用できます。以前に有効にしたコレクションが無効になっている場合、そのコレクションは `$listChangeStreams` 出力に表示されません。

```
//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:{}}));
```

例: Python での変更ストリームの使用

コレクションレベルで、 Python を用いて Amazon DocumentDB 変更ストリームを使用する例を次に示します。

```
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': '015daf99d400000001010000000100009025'},
'clusterTime': Timestamp(1571789268, 1),
'documentKey': {'_id': ObjectId('5daf9502ea258751778163d7')},
'ns': {'coll': 'foo', 'db': 'bar'},
'operationType': 'update',
'updateDescription': {'removedFields': [], 'updatedFields': {'x': 2}}}
"""
```

データベースレベルで、 Python を用いて Amazon DocumentDB 変更ストリームを使用する例を次に示します。

```
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': '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
"""

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'}
"""
```

ドキュメント全体の参照

変更の更新イベントには、ドキュメント全体は含まれません。加えられた変更のみが含まれます。ユースケースで、更新の影響を受ける完全なドキュメントが必要な場合は、ストリームを開くときに完全なドキュメント参照を有効にできます。

変更の更新ストリームイベントの fullDocument ドキュメントは、ドキュメント参照時に更新されたドキュメントの最新バージョンを表します。更新オペレーションと fullDocument ルックアップの間で変更が発生した場合、fullDocument ドキュメントが更新時にドキュメントの状態を表していない可能性があります。

更新ルックアップを有効にしてストリームオブジェクトを作成するには、次の例を使用します。

```
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()
```

ストリームオブジェクトの出力は次のようになります。

```
{'_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}}}
```

$changeStreamSplitLargeEvent を使用して大きなイベントをフラグメントに分割する

> **Note**
>
> $changeStreamSplitLargeEvent ステージは、エンジンバージョン 8.0.2 以降、Amazon DocumentDB 8.0 で使用できます。

$changeStreamSplitLargeEvent ステージを使用すると、16 MB を超える変更イベントはフラグメントに分割され、変更ストリームカーソルを介して順番に返されます。16 MB 未満のイベントは分割されずに返されます。ステージは、最上位のフィールド境界に沿ってイベントを分割します。1 つのフィールド内で分割したり、サブドキュメントに再帰したりすることはありません。このため、1 つの最上位フィールド自体が 16 MB より大きい場合、ステージは有効なフラグメントを生成できず、代わりにエラーを返します。

ベストプラクティスとして、イベントのサイズを小さくして$changeStreamSplitLargeEvent、可能な限り を使用しないことをお勧めします。

$changeStreamSplitLargeEvent ステージは$changeStreamパイプラインでのみ使用でき、最後のステージとして 1 回のみ使用する必要があります。

各フラグメントには、独自の resumeTokenと独自の がありますpostBatchResumeToken。最終以外のフラグメントトークンから再開すると (通常の再開トークンまたは postBatchResumeToken を使用) 、後続のフラグメントが配信されます。トークンがイベントの最終フラグメントに対応する場合、変更ストリームカーソルは後続のイベントを返します。

変更ストリームの再開

再開トークンを使用すると、変更ストリームを後で再開できます。このトークンは、最後に取得した変更イベントドキュメントの _id フィールドと同じです。

```
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': '015daf9c5b0000000101000000100009025'}
"""
#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}}
"""
```

startAtOperationTime を用いた変更ストリームの再開

startAtOperationTime を使って、特定のタイムスタンプから後で変更ストリームを再開できます。

> **Note**
>
> startAtOperationTime を使用する能力は、Amazon DocumentDB 4.0+ で入手可能です。startAtOperationTime を使用する場合、変更ストリームカーソルは、指定された Timestamp 以降で発生した変更のみを返します。startAtOperationTime と resumeAfter のコマンドは相互に排他的であるため、一緒に使用することはできません。

```
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}}
"""
```

postBatchResumeToken を用いた変更ストリームの再開

Amazon DocumentDB 変更ストリームは、postBatchResumeToken という追加のフィールドを返すようになりました。このフィールドは、 $changestream コマンドと getMore コマンドから返されます。

Python の $changestream コマンドの例:

```
db.command({"aggregate": "sales", "pipeline": [{ "$changeStream": {}}], "cursor":
 {"batchSize": 1}})
```

正常な出力:

```
cursor" : {
   "firstBatch" : [ ],
   "postBatchResumeToken" : {"_data" : "0167c8cbe60000000004"},
   "id" : NumberLong("9660788144470"),
   "ns" : "test.sales"
}
```

Python の getMore コマンドの例:

```
db.command({"getMore": NumberLong(<cursor id>), "collection": "sales", "batchSize":
 1 })
```

正常な出力

```
cursor" : {
   "nextBatch" : [ ],
   "postBatchResumeToken" : {"_data" : "0167c8cbe60000000004"},
   "id" : NumberLong("9660788144470"),
   "ns" : "test.sales"
}
```

postBatchResumeToken フィールドは、再開トークンの使用方法と同様に、resumeAfter フィールドで新しい変更ストリームカーソルを開くために使用できます。

選択した postBatchResumeToken の後に開始するストリームを開きます。

```
post_batch_resume_token = output['cursor']['postBatchResumeToken']
stream = db.watch(full_document='updateLookup', resume_after=post_batch_resume_token)
```

実際のイベントを反映するオペレーションログ (oplog) エントリに常に対応する通常の再開トークンとは異なり、postBatchResumeToken は変更ストリームがサーバーでスキャンした oplog エントリに対応しますが、これは必ずしも一致する変更ではありません。

古い通常の再開トークンで再開しようとすると、強制的にデータベースが指定されたタイムスタンプと現在の時刻の間のすべての oplog エントリをスキャンするようになります。これにより、各サブクエリスキャンで短時間、多数のクエリが内部で生成される可能性があります。これにより、CPU 使用率が急増し、データベースのパフォーマンスが低下します。最後の postBatchResumeToken で再開すると、一致しない oplog エントリのスキャンがスキップされます。

変更ストリーム内のトランザクション

変更ストリームイベントには、コミットされていないトランザクションや中止されたトランザクションのイベントは含まれません。例えば、1 つの INSERT オペレーションと 1 つの UPDATE オペレーションでトランザクションを開始し、INSERT オペレーションが成功しても UPDATE オペレーションが失敗した場合、トランザクションはロールバックされます。このトランザクションはロールバックされたため、変更ストリームにはこのトランザクションのイベントは含まれません。

変更ストリームログの保持期間の変更

AWS マネジメントコンソール または を使用して、変更ストリームログの保持期間を 1 時間から 7 日の間で変更できます AWS CLI。

Using the AWS マネジメントコンソール

変更ストリームログの保持期間を変更するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。

   > **Tip**
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. [パラメータグループ] ペインで、クラスターに関連付けられたクラスターパラメータグループを選択します。クラスターに関連付けられているクラスターパラメータグループを調べるには、「Amazon DocumentDB クラスターのパラメータグループの確認をする」を参照してください。
4. 結果のページには、クラスターパラメータグループのパラメータと対応する詳細が表示されます。change_stream_log_retention_duration パラメータを選択します。
5. ページの右上にある [編集] を選択して、パラメータの値を変更します。change_stream_log_retention_duration のパラメータは、1 時間から 7 日の間で変更できます。

6. 変更を行い、[クラスターのパラメータを変更] を選択して変更を保存します。変更を破棄するには、[キャンセル] を選択します。

Using the AWS CLI

クラスターパラメータグループの change_stream_log_retention_duration パラメータを変更するには、以下のパラメータを指定して modify-db-cluster-parameter-group オペレーションを使用します。

- **--db-cluster-parameter-group-name** — 必須。変更するクラスターパラメータグループの名前。クラスターに関連付けられているクラスターパラメータグループを調べるには、「Amazon DocumentDB クラスターのパラメータグループの確認をする」を参照してください。
- **--parameters** — 必須。変更するパラメータ。各パラメータの入力には以下を含める必要があります。
  - **ParameterName** - 変更しているパラメータの名前。この場合は change_stream_log_retention_duration
  - **ParameterValue** - このパラメータの新しい値。
  - **ApplyMethod** - このパラメータの変更を適用する方法です。有効な値は、immediate および pending-reboot です。

> **Note**
>
> ApplyType が static であるパラメータでは、ApplyMethod が pending-reboot である必要があります。

1. パラメータ change_stream_log_retention_duration の値を変更するには、次のコマンドを実行し、parameter-value をパラメータの変更後の値に置き換えます。

   Linux、macOS、Unix の場合:

```
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"
```

Windows の場合:

```
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"
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "DBClusterParameterGroupName": "sample-parameter-group"
}
```

2. 少なくとも 5 分お待ち下さい。
3. sample-parameter-group のパラメータ値を一覧表示し、変更が行われたことを確認します。

Linux、macOS、Unix の場合:

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

Windows の場合:

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

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "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**
>
> ストリームログの保存期間を変更しても、ログサイズが 51,200 MB より大きくなるまで (>)、設定した change_stream_log_retention_duration の値より古いログは削除されません。

セカンダリインスタンスでの変更ストリームの使用

セカンダリインスタンスで変更ストリームの使用を開始するには、変更ストリームカーソルを readPreference でセカンダリとして開きます。

変更ストリームカーソルを開いて、特定のコレクション、または特定のクラスターもしくはデータベース内のすべてのコレクションの変更イベントを監視できます。任意の Amazon DocumentDB インスタンス上で変更ストリームカーソルを開き、ライターインスタンスとリーダーインスタンスの両方から変更ストリームドキュメントを取得できます。変更ストリームトークン (resumeToken や startOperationTime など) は、ライターインスタンスとリーダーインスタンスで開かれたさまざまな変更ストリームカーソル間で共有できます。

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)
```

セカンダリインスタンス上の変更ストリームに関するガイドラインと制限事項

- 変更ストリームイベントは、プライマリインスタンスからセカンダリインスタンスにレプリケートする必要があります。また、Amazon CloudWatch で DBInstanceReplicaLag メトリクスからのラグをモニタリングすることもできます。
- セカンダリインスタンスのタイムスタンプは、プライマリインスタンスと常に同期しているとは限りません。この場合、ずれを解消するためセカンダリインスタンスのタイムスタンプに遅延が発生する可能性があります。ベストプラクティスとして、 startAtOperationTimeまたは resumeToken を使用してセカンダリインスタンスで監視を開始します。
- ドキュメントサイズが大きく、かつfullDocumentLookup を実行していて、プライマリインスタンスへの同時書き込みワークロードが多い場合、プライマリインスタンスと比較してセカンダリインスタンスのスループットが低下する可能性があります。ベストプラクティスとして、セカンダリのバッファキャッシュヒット率をモニタリングし、バッファキャッシュヒット率が高いことを確認します。

Amazon DocumentDB でのトランザクション

Amazon DocumentDB (MongoDB 互換) はAmazon DocumentDB 4.0 以降のトランザクションをサポートしています。複数のドキュメント、ステートメント、コレクション、およびデータベース間でトランザクションを実行できます。トランザクションは、Amazon DocumentDB クラスター内の 1 つまたは複数のドキュメントに対し、原始的、一貫性、分離、耐久性 (ACID) オペレーションを実行できるようにすることで、アプリケーション開発を簡素化します。トランザクションの一般的なユースケースには、財務処理、注文の履行と管理、マルチプレイヤーゲームの構築などがあります。

トランザクションを有効にするために、追加料金はかかりません。支払いは、トランザクションの一部として消費した読み書きの IO に対してのみ支払います。

トピック

- 要件
- ベストプラクティス
- 制限事項
- モニタリングと診断
- トランザクション分離レベル
- ユースケース
- サポートされているコマンド
- サポート対象外の機能
- セッション
- トランザクションエラー

要件

トランザクション機能を使用するには、次の要件を満たしている必要があります。

- Amazon DocumentDB 4.0 以降を使用する必要があります。
- MongoDB 4.0 以降と互換性のあるドライバーを使用する必要があります。

ベストプラクティス

Amazon DocumentDB でトランザクションを最も多く利用できるようにするためのベストプラクティスをいくつか紹介します。

- トランザクションの完了後は、必ずトランザクションをコミットまたは中断してください。トランザクションが不完全な状態のままにすると、データベースリソースが結びつき、書き込みの競合が発生する可能性があります。
- トランザクションは、必要なコマンド数を最小限に抑えることをお勧めします。複数のステートメントを持つトランザクションを複数の小さなトランザクションに分割できる場合は、タイムアウトの可能性を減らすためにこれを行うことをお勧めします。長時間実行される読み取りではなく、短いトランザクションを作成することを常に目指してください。

制限事項

- Amazon DocumentDB は、トランザクション内のカーソルをサポートしていません。
- Amazon DocumentDB はトランザクションで新しいコレクションを作成できず、存在しないコレクションに対してクエリ/更新することはできません。
- ドキュメントレベルの書き込みロックには 1 分間のタイムアウトが適用されますが、これはユーザーによって設定できません。
- 8.0.2 より前のエンジンバージョンでは、再試行可能な書き込みはサポートされていません。エンジンバージョン 8.0.2 以降、再試行可能な書き込みは、個々の書き込みが最大 1 つのドキュメントに影響する書き込みオペレーションのトランザクション外でサポートされ、トランザクションのコミットおよび中止オペレーションは再試行可能です。詳細については、「Amazon DocumentDB での再試行可能な書き込み」を参照してください。
- 各 Amazon DocumentDB インスタンスには、インスタンスで開くことができる同時トランザクションの数の上限があります。詳細については、「インスタンスクォータ」を参照してください。
- 特定のトランザクションでは、トランザクションログのサイズは 32 MB 未満である必要があります。
- Amazon DocumentDB はトランザクション内で `count()` をサポートしていますが、すべてのドライバがこの機能をサポートしているわけではありません。別の方法として、`countDocuments()` API を使用して、count クエリをクライアント側の集計クエリに変換します。
- トランザクションの実行制限は 1 分で、セッションには 30 分のタイムアウトがあります。トランザクションがタイムアウトすると、トランザクションは中止され、既存のトランザクションのセッション内で後続のコマンドが発行されると、次のエラーが表示されます。

```
WriteCommandError({
"ok" : 0,
"operationTime" : Timestamp(1603491424, 627726),
"code" : 251,
```

```
"errmsg" : "Given transaction number 0 does not match any in-progress transactions."
}
```

モニタリングと診断

Amazon DocumentDB 4.0 でのトランザクションのサポートにより、トランザクションのモニタリングに役立つ追加の CloudWatch メトリックスが追加されました。

新しい CloudWatch メトリクス

- `DatabaseTransactions`: 1 分間に実行された未処理のトランザクションの数。
- `DatabaseTransactionsAborted`: 1 分間に中断された未処理のトランザクションの数。
- `DatabaseTransactionsMax`: 1 分間にインスタンスで開かれるトランザクションの最大数。
- `TransactionsAborted`: 1 分間にインスタンスで中断されたトランザクションの数。
- `TransactionsCommitted`: 1 分間にインスタンスでコミットされたトランザクションの数。
- `TransactionsOpen`: 1 分間で実行されるインスタンスで開かれるトランザクションの数。
- `TransactionsOpenMax`: 1 分間にインスタンスで開いているトランザクションの最大数。
- `TransactionsStarted`: 1 分間にインスタンスで開始されたトランザクションの数。

> **Note**
>
> Amazon DocumentDB の CloudWatch メトリックスの詳細については、Amazon DocumentDB と CloudWatch のモニタリング。

さらに、新しいフィールドが `currentOp` `lsid`、`transactionThreadId`、`idle transaction` のための新しい状態、および `serverStatus` トランザクション: `currentActive`、`currentInactive`、 `currentOpen`、`totalAborted`、`totalCommitted`、および `totalStarted` に追加されました。

トランザクション分離レベル

トランザクションを開始するときは、次の例`writeConcern`に示すように、 `readConcern`と の両方を指定できます。

```
mySession.startTransaction({readConcern: {level: 'snapshot'}, writeConcern:
{w: 'majority'}});
```

readConcern を使用する場合、Amazon DocumentDB はデフォルトでスナップショットアイソレーションをサポートしています。readConcern に local、available、majority が指定された場合、Amazon DocumentDB は readConcern レベルをスナップショットにアップグレードします。Amazon DocumentDB は線形化可能な readConcern をサポートしていません。このような読み取り懸念を指定すると、エラーが発生します。

writeConcern を使用する場合、Amazon DocumentDB はデフォルトで多数数をサポートしており、3 つの AZ にわたってデータの 4 つのコピーが永続化されると、書き込みクォーラムが達成されます。低い writeConcern が指定されている場合、Amazon DocumentDB は writeConcern を過半数にアップグレードします。さらに、Amazon DocumentDB の書き込みはすべてジャーナル処理され、ジャーナリングを無効にすることはできません。

ユースケース

このセクションでは、トランザクションの 2 つのユースケース、マルチステートメントとマルチコレクションについて説明します。

マルチステートメントトランザクション

Amazon DocumentDB トランザクションはマルチステートメントです。つまり、明示的なコミットまたはロールバックを使用して、複数のステートメントにまたがるトランザクションを記述できます。単一のアトミック操作として、insert、update、delete、および findAndModify のアクションをグループ分けできます。

マルチステートメントトランザクションの一般的なユースケースは、デビットクレジットトランザクションです。例えば、友達に服のお金を借りたとします。アカウントから 500 ドルのデビット (引き出し) を実行し、友達のアカウントに 500 ドル (入金) をクレジットする必要があります。その操作を実行するには、原子性を保証するために、単一のトランザクション内で債務操作と与信操作の両方を実行します。これにより、500 ドルがアカウントから引き落とされ、友達のアカウントに入金されないシナリオが防止されます。このユースケースは次のようになります。

```
// *** 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();
```

マルチコレクショントランザクション

また、トランザクションはマルチコレクションです。つまり、単一のトランザクション内および複数のコレクションにわたって複数の操作を実行するために使用できます。これにより、データの整合性が保たれ、データの整合性が維持されます。コマンドを単一の <> としてコミットする場合、トランザクションはすべてまたは何も実行されません。その場合、トランザクションはすべて成功するか、すべて失敗します。

以下は、マルチコレクショントランザクションの例で、複数ステートメントトランザクションの例と同じシナリオとデータを使用します。

```
// *** 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
```

コールバック API のトランザクション API の例

コールバック API は 4.2 以降のドライバでのみ使用できます。

Javascript

次のコードは、Javascript で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
// *** 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

次のコードは、Node.js で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
// 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#

次のコードは、C# で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
// 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

次のコードは、Ruby で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
// 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

次のコードは、Go で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
// 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

次のコードは、Java で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
// 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

次のコードは、C で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
// 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

次のコードは、Python で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
// 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
```

コア API のトランザクション API の例

Javascript

次のコードは、Javascript で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
// *** 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#

次のコードは、C# で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
// 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

次のコードは、Ruby で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
# 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

次のコードは、Go で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
// 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

次のコードは、Java で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
// 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

次のコードは、C で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
// 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

次のコードは、Scala で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
// 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

次のコードは、Python で Amazon DocumentDB トランザクション API を使用する方法を示しています。

```
// 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
```

サポートされているコマンド

コマンド	サポート
abortTransaction	はい
commitTransaction	はい
endSessions	はい
killSession	はい
killAllSession	はい
killAllSessionsByPattern	いいえ
refreshSessions	なし
startSession	はい

サポート対象外の機能

方法	ステージまたはコマンド
db.collection.aggregate()	$collStats
	$currentOp

方法	ステージまたはコマンド

	`$indexStats`
	
	`$listSessions`
	
	`$out`

`db.collection.count()`
 	
`db.collection.countDocuments()`
	`$where`
	
	`$near`
	
	`$nearSphere`

`db.collection.insert()`	`insert` は、既存のコレクションに対して実行されない場合はサポートされません。このメソッドは、既存のコレクションを対象とする場合にサポートされます。

セッション

MongoDB セッションは、再試行可能な書き込み、因果一貫性、トランザクション、およびデータベース間の操作の管理をサポートするために使用されるフレームワークです。セッションが作成されると、論理セッション識別子 (LSID) がクライアントによって生成され、サーバーにコマンドを送信するときに、そのセッション内のすべてのオペレーションにタグを付けるために使用されます。

Amazon DocumentDB は、トランザクションを有効にするためのセッションの使用をサポートしていますが、因果整合性はサポートしていません。エンジンバージョン 8.0.2 以降、Amazon DocumentDB は、個々の書き込みが最大 1 つのドキュメントに影響する書き込みオペレーションのトランザクション外での再試行可能な書き込みをサポートしています。以前のエンジンバージョンでは、再試行可能な書き込みはサポートされていません。詳細については、「Amazon DocumentDB での再試行可能な書き込み」を参照してください。

Amazon DocumentDB 内でトランザクションを利用する場合、トランザクションはセッション内から `session.startTransaction()` API とセッションは、一度に 1 つのトランザクションをサポートします。同様に、トランザクションは commit (`session.commitTransaction()`) または中止 (`session.abortTransaction()`) API。

因果整合性

因果整合性により、単一のクライアントセッション内でクライアントが書き込み後の読み取り整合性、モノアトミックな読み取り/書き込みが読み取りに従うことが保証され、これらの保証はプライマリだけでなく、クラスタ内のすべてのインスタンスに適用されます。Amazon DocumentDB は因果一貫性をサポートしておらず、次のステートメントではエラーが発生します。

```
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()
```

セッション内で因果一貫性を無効にできます。これにより、セッションフレームワークを利用できますが、読み取りの因果整合性は保証されません。Amazon DocumentDB を使用する場合、プライマリからの読み取りは書き込み後の読み取り整合性になり、レプリカインスタンスからの読み取りは最終的に一貫します。トランザクションは、セッションを利用するための主要なユースケースです。

```
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 }
```

再試行可能な書き込み

再試行可能な書き込みでは、ネットワークエラーが発生した場合、またはドライバーがプライマリを見つけられない場合、ドライバーは自動的に書き込みオペレーションを 1 回再試行します。エンジンバージョン 8.0.2 以降、Amazon DocumentDB は再試行可能な書き込みをサポートしています。以前のエンジンバージョンでは、接続文字列`retryWrites=false`で を設定して、再試行可能な書き込みを無効にする必要があります。詳細については、「Amazon DocumentDB での再試行可能な書き込み」を参照してください。

トランザクションエラー

トランザクションを使用する場合、トランザクション番号が進行中のトランザクションと一致しないというエラーを発生させるシナリオがあります。

このエラーは、少なくとも 2 つの異なるシナリオで生成できます。

- 1 分間のトランザクションタイムアウト後。
- インスタンスの再起動後 (パッチ適用、クラッシュリカバリなど)、トランザクションが正常にコミットされた場合でも、このエラーを受け取る可能性があります。インスタンスの再起動中、データベースは正常に完了したトランザクションと中止されたトランザクションの違いを判断できません。つまり、トランザクションの完了状態があいまいとなります。

このエラーを処理する最善の方法は、トランザクションの更新を冪等化することです。例えば、`$set` インクリメント/デクリメント操作の代わりにミューテーター。以下を参照してください。

```
{ "ok" : 0,
"operationTime" : Timestamp(1603938167, 1),
"code" : 251,
"errmsg" : "Given transaction number 1 does not match any in-progress transactions."
}
```

Amazon DocumentDB での再試行可能な書き込み

エンジンバージョン 8.0.2 以降、Amazon DocumentDB は再試行可能な書き込みをサポートしています。一時的なネットワークエラーまたはプライマリ選択が原因で書き込みオペレーションが失敗した場合、ドライバーは自動的にオペレーションを 1 回だけ再試行できます。Amazon DocumentDB は、再試行された書き込みを重複排除して、操作が最大 1 回適用されるようにし、べき等性を維持します。

再試行可能な書き込みには、再試行可能な書き込みプロトコルをサポートする MongoDB 互換ドライバーが必要です。最新の MongoDB 互換ドライバーは、デフォルトで再試行可能な書き込みを有効にします (retryWrites=true接続文字列の)。以前のバージョンからエンジン 8.0.2 にアップグレードする場合は、接続文字列retryWrites=falseから を削除してこの機能を有効にできます。

要件

再試行可能な書き込みを使用するには、次の要件を満たす必要があります。

- Amazon DocumentDB エンジンバージョン 8.0.2 以降。
- 再試行可能な書き込みをサポートする MongoDB 互換ドライバー。最小バージョンについては、ドライバーのドキュメントを参照してください。
- 接続文字列には が含まれている必要があります。含まれていない場合retryWrites=true、ドライバーはデフォルトで再試行可能な書き込みにする必要があります (最新のドライバーは です)。

サポートされているオペレーション

再試行可能な書き込みは、個々の書き込みが最大 1 つのドキュメントに影響する書き込みオペレーションに適用されます。バッチ内の各書き込みは個別に重複排除されるため、バッチオペレーションは適格です。

次の書き込みオペレーションは再試行可能です。

- insertOne
- insertMany
- updateOne
- deleteOne
- findOneAndUpdate
- findOneAndDelete
- findOneAndReplace
- bulkWrite (insertOne、、updateOne、deleteOneまたは replaceOneオペレーションで構成される場合)

insertMany および の場合bulkWrite、バッチ内の各ドキュメントは個別に重複排除されます。再試行が必要な場合は、まだ適用されていないドキュメントのみが挿入されます。

> **Note**
>
> commitTransaction および abortTransactionは、書き込みとしてではなくトランザクション制御コマンドとして再試行することもできます。トランザクション内の個々の書き込みは再試行できません。詳細については、「制限事項」を参照してください。

制限事項

Amazon DocumentDB での再試行可能な書き込みには、次の制限が適用されます。

- updateMany と deleteManyは再試行できません。
- マルチステートメントトランザクション内の書き込みは再試行できません。トランザクションのコミットオペレーションと中止オペレーションは個別に再試行できます。
- ドキュメントには、再試行可能な挿入の _idフィールドが含まれている必要があります。
- 再試行可能な書き込みは、エンジンバージョン 8.0.2 以降でのみ使用できます。以前のエンジンバージョンでは、エラーを回避するために接続文字列retryWrites=falseに を設定します。
- クエリプランナーバージョン 1.0 を使用するクラスターでは、書き込みオペレーションを再試行できません。
- 大きなドキュメントを返す findOneAndUpdate、findOneAndDelete、および findOneAndReplaceオペレーションでは、完全な結果ドキュメントが重複排除のためにキャッシュされるため、再試行可能な書き込みによって書き込みレイテンシーが増加する可能性があります。書き込みパフォーマンスが重要で、返されるドキュメントが大きい場合は、それらのワークロードretryWrites=falseに対して を設定することを検討してください。

再試行可能な書き込みの有効化

再試行可能な書き込みを有効にするには、接続文字列retryWrites=trueに を含めます。

```
mongodb://<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=true
```

ドライバーのデフォルトが の場合retryWrites=true、接続文字列retryWrites=falseから明示的なものをすべて削除できます。

完全な接続文字列のチュートリアルについては、「」を参照してくださいプログラムによる Amazon DocumentDB への接続。

再試行可能な書き込みの仕組み

アプリケーションで を使用して書き込みを送信するとretryWrites=true、ドライバーは論理セッション ID (lsid) とトランザクション番号 (txnNumber) を書き込みコマンドにアタッチします。Amazon DocumentDB は、これらの識別子を使用して書き込みを重複排除します。

- 最初の試行では、Amazon DocumentDB は書き込みを実行し、結果をキャッシュします。
- ドライバーが同じ書き込みを再試行した場合 (同じ lsidと txnNumber)、Amazon DocumentDB は書き込みを再実行せずにキャッシュされた結果を返します。
- Amazon DocumentDB は、元の書き込みから 60 分間の再試行を受け入れます。ドライバーはすぐに再試行するため、このウィンドウはドライバーが必要とするよりもはるかに長くなります。

元の書き込みから 60 分以上後に再試行が到着した場合、Amazon DocumentDB にはキャッシュされた結果はなく、新しいオペレーションとして書き込みが実行されます。

エラー処理

次のエラーは、Amazon DocumentDB での再試行可能な書き込みに固有のものです。

エラーコード	名前	説明
225	TransactionTooOld	同じセッションとステートメントの新しいトランザクション番号がすでにコミットされているため、この再試行は古すぎて適用できません。
301	再試行可能な書き込みはサポートされていません	書き込みには再試行可能な書き込みフィールドが含まれていましたが、エンジンバージョンは再試行可能な書き込みをサポートしていないか、オペレーションは再試行できません。8.0.2 より前のエンジンバージョンに接続するretryWrites=false ときに を設定します。

retryWrites=false からの移行

エンジンバージョン 8.0.2 にアップグレードし、現在接続文字列retryWrites=falseで を使用している場合:

1. クラスターをエンジンバージョン 8.0.2 以降にアップグレードします。
2. アプリケーションの接続文字列`retryWrites=false`から を削除するか、 に変更しま す`retryWrites=true`。
3. アプリケーションコードの変更は必要ありません。ドライバーは再試行を自動的に処理します。

Amazon DocumentDB 8.0 での照合の使用

Amazon DocumentDB 8.0 で照合がサポートされるようになりました。照合では、文字列比較用の言語固有のルールを設定できます。照合では、大文字と小文字を区別する比較のルールを指定するか、言語ロケールを指定できます。照合は、DocumentDB 8.0 のコレクションレベルまたはインデックスレベルで設定できます。DocumentDB で照合を使用すると、内部的に以下のパラメータを使用して照合ドキュメントが作成されます。

```
{
locale: string,
caseLevel: boolean,
caseFirst: string,
strength: int,
numericOrdering: boolean,
alternate: string,
maxVariable: string,
backwards: boolean,
normalization: boolean
}
```

制限事項

Amazon DocumentDB では、照合には次の制限があります。

- 照合は、Amazon DocumentDB 8.0 で利用可能なプランナー v3 と互換性があります。plannerv2 または plannerv1 に切り替えると、「インデックスが見つかりません」エラーなど、一貫性のない動作が発生する可能性があります。
- ライブラリ固有の違いにより、照合を含む mongodb コレクションがエクスポートされる場合は、移行前に metadata.bson ファイルを更新し、そのバージョンを 57.1 から 60.2 に変更する必要があります。

- まれに、照合設定が文字数の内部制限に違反し、次のエラーが発生することがあります。「エラー: 照合ドキュメントには、サポートされている以上のデフォルト以外の属性があります。オプションの数を減らしてください。」 この場合、照合ドキュメントで指定するオプションを減らすか、可能な限りデフォルト値を使用してみてください。

Amazon DocumentDB 8.0 でのビューの使用

- Amazon DocumentDB 8.0 でビューがサポートされるようになりました。ビューは、指定された集計オペレーションに基づいてデータを表示する仮想コレクションとして機能します。ビューを作成するときは、1 つ以上のソースコレクションからデータを変換するクエリを定義します。Amazon DocumentDB 8.0 は、追加のストレージリソースを消費することなく、ビューにアクセスするたびにこのクエリを実行します。標準コレクションとは異なり、Amazon DocumentDB 8.0 のビューはドキュメントをディスクに保存しないため、変換またはフィルタリングされたデータをアプリケーションに提示するための効率的なソリューションとなります。Amazon DocumentDB でビューを作成するには、createView コマンドまたは db.createView() ヘルパーメソッドを使用できます。

```
db.createView("viewName","sourceCollection",
[
    { $match: { status: "active" } },
    { $project: { _id: 1, name: 1, email: 1 } }
]
)
```

  これにより、アクティブなドキュメントとプロジェクトのみを含むsourceCollection」に基づいてviewName」という名前のビューが作成され、_id、名前、E メールフィールドのみがプロジェクトされます。Amazon DocumentDB のビューは読み取り専用です。ビューに対する書き込みオペレーションはエラーを返します。大規模なデータセットで最適なパフォーマンスを得るには、ビューパイプラインを構築して効率を最大化できます。複雑な集約パイプラインの場合、パイプラインの最初のステージまたは早い段階で $match ステージを使用して、後続のステージで処理する必要があるドキュメントの数を減らし、クエリのパフォーマンスを向上させることをお勧めします。

ベストプラクティス

ビューで従うべきベストプラクティスを以下に示します。

- 早期フィルタリング: $match ステージをビューパイプラインの早い段階で使用して、処理されるデータ量を減らします。
- 複雑な集計を避ける: 複雑な集計で頻繁にアクセスされるビューの場合は、定期的に更新される事前計算された結果を含む個別のコレクションを作成することを検討してください。
- Index Planning: ビューパイプライン、特に $match および $sort オペレーションで使用されるフィールドが、ソースコレクションで適切にインデックス作成されていることを確認します。
- クエリの最適化: explain コマンドを使用して、ビュークエリの実行方法を理解し、それに応じて最適化します。
- ビューの代替方法: Amazon DocumentDB ビューと MongoDB ビューの機能の違いを考慮すると、制限が発生した場合にビューの代替として、スケジュールされた更新で通常のコレクションを使用することを検討してください。

アグリゲータ演算子の互換性

Amazon DocumentDB は、互換性の拡張を継続しながら、ビュー定義で多くの集約演算子をサポートしています。ビューを使用する場合は、サポートされている以下の演算子に焦点を当てます。

- ドキュメントをフィルタリングするための $match
- フィールドの選択と変換の $project
- 計算フィールドを追加するための $addFields
- 結果を注文するための $sort
- ページ分割の $limit と $skip

$currentOp、$replaceRoot、$geoNear などの一部の特殊な演算子は現在、ビュー定義ではなく直接集計クエリで動作します。

インデックスとビューの活用

Amazon DocumentDB 8.0 のビューは、基盤となるコレクションのインデックスを使用します。そのため、ビューでインデックスを直接作成、削除、または再構築することはできません。ただし、ソースコレクションで適切に設計されたインデックスは、ビューのクエリパフォーマンスを大幅に向上させることができます。ビューのクエリパフォーマンスを最適化するいくつかのステップを以下に示します。

- ビューのパイプライン、特に $match および $sort オペレーションで使用されるソースコレクションフィールドに適切なインデックスが存在することを確認します。

- explain() メソッドを使用して、クエリ実行プランを分析し、インデックスの使用を確認します。例: `db.viewName.find({...}).explain()`

Amazon DocumentDB 生成人工知能

Amazon DocumentDB は、機械学習 (ML) モデルと生成人工知能 (AI) モデルが Amazon DocumentDB に保存されているデータをリアルタイムで操作できるようにする機能を提供します。お客様は、別のインフラストラクチャの管理、別のサービスに接続するためのコードの記述、プライマリデータベースからのデータの複製に時間を費やす必要がなくなります。

人工知能と AWS が AI のニーズをサポートする方法の詳細については、この「What-is」記事を参照してください。

トピック

- Amazon SageMaker AI Canvas を使用したノーコード機械学習
- Amazon DocumentDB のベクトル検索

Amazon SageMaker AI Canvas を使用したノーコード機械学習

Amazon SageMaker AI Canvas を使用すると、まったくコードを記述しなくても独自の AI/ML モデルを構築できます。リグレッションや予測などの一般的なユースケース用の ML モデルを構築し、Amazon Bedrock から基盤モデル (FM) にアクセスして評価できます。あるいは Amazon SageMaker AI JumpStart からパブリック FM にアクセスしてコンテンツ生成、テキスト抽出、テキスト要約などを行い、生成 AI ソリューションをサポートすることもできます。

SageMaker AI Canvas を使用してノーコード ML モデルを構築する方法

Amazon DocumentDB が Amazon SageMaker AI Canvas と統合され、Amazon DocumentDB に保存されているデータでのノーコード機械学習 (ML) が可能になりました。Amazon DocumentDB に保存されているデータを使用して、いっさいコードを記述することなく、リグレッションや予測などのニーズに合った ML モデルを構築し、コンテンツの要約や生成の基礎モデルを使用できるようになりました。

SageMaker AI Canvas は、ビジュアルインターフェースの提供により、Amazon DocumentDB を利用するお客様が AI/ML の専門知識なしに一切コードを記述することなく予測の生成等を行うことを可能にします。お客様は、 から SageMaker AI Canvas ワークスペースを起動し AWS マネジメントコンソール、データ準備とモデルトレーニングのために Amazon DocumentDB データをインポートして参加できるようになりました。Amazon DocumentDB のデータを SageMaker AI Canvas で

使用することで、顧客離れの予測、不正行為の検出、メンテナンス障害の予測、ビジネスメトリクスの予測、コンテンツの生成などを行うモデルを構築および拡張できるようになりました。お客様は、SageMaker AI Canvas の Quick とのネイティブ統合を使用して、チーム間で ML 主導のインサイトを公開および共有できるようになりました。SageMaker AI Canvas のデータインジェストパイプラインは、デフォルトで Amazon DocumentDB セカンダリインスタンスで実行されるため、アプリケーションと SageMaker AI Canvas の取り込みワークロードのパフォーマンスが妨げられることはありません。

Amazon DocumentDB のお客様は、Amazon DocumentDB の新しいノーコード ML コンソールページに移動し、新規または利用可能な SageMaker AI Canvas ワークスペースに接続することで、SageMaker AI Canvas の使用を開始できます。

SageMaker AI ドメインとユーザープロファイルの設定

VPC 専用モードで実行されている SageMaker AI ドメインから Amazon DocumentDB クラスターに接続できます。VPC 内で SageMaker AI ドメインを起動することで、SageMaker AI Studio および Canvas 環境からのデータフローを制御できます。これにより、インターネットアクセスを制限し、標準の AWS ネットワークとセキュリティ機能を使用してトラフィックをモニタリングおよび検査し、VPC エンドポイントを介して他の AWS リソースに接続できます。Amazon SageMaker AI Canvas 入門」およびAmazon SageMaker AI デベロッパーガイド」の「インターネットアクセスのない VPC で Amazon SageMaker AI Canvas を設定」を参照して、Amazon DocumentDB クラスターに接続するための SageMaker AI ドメインを作成します。 Amazon SageMaker

Amazon DocumentDB と SageMaker AI Canvas の IAM アクセス許可の設定

自分に関連付けられたロールと ID に `AmazonDocDBConsoleFullAccess` がアタッチされている Amazon DocumentDB ユーザーは、 AWS マネジメントコンソールにアクセスできます。Amazon SageMaker AI Canvas によるノーコード機械学習へのアクセスを取得するために、前述のロールまたは ID に次のアクションを追加します。

```
"sagemaker:CreatePresignedDomainUrl",
"sagemaker:DescribeDomain",
"sagemaker:ListDomains",
"sagemaker:ListUserProfiles"
```

SageMaker AI Canvas のデータベースユーザーとロールの作成

Amazon DocumentDB の ロールベースアクセス制御 (RBAC)を使用して、ユーザーがデータベースで実行できるアクションへのアクセスを制限できます。RBAC は、1 つ以上のロールをユーザーに付

与することで機能します。これらのロールは、ユーザーがデータベースリソースに対して実行できるオペレーションを決定します。

Canvas ユーザーとして、ユーザー名とパスワードの認証情報を使用して Amazon DocumentDB データベースに接続します。Amazon DocumentDB RBAC 機能を使用して、特定のデータベースへの読み取りアクセス権を持つ Canvas ユーザーのデータベースユーザー/ロールを作成できます。

たとえば、`createUser` を使用します:

```
db.createUser({
user: "canvas_user",
pwd: "<insert-password>",
roles: [{role: "read", db: "sample-database-1"}]
})
```

これにより、`sample-database-1` データベースへの読み取りアクセス許可を持つ `canvas_user` が作成されます。Canvas アナリストは、この認証情報を使用して、Amazon DocumentDB クラスター内のデータにアクセスできます。詳細については、「ロールベースのアクセス制御 (RBAC) を使用したデータベースへのアクセス」を参照してください。

利用できるリージョン

ノーコード統合は、Amazon DocumentDB と Amazon SageMaker AI Canvas の両方がサポートされているリージョンで使用できます。リージョンには以下が含まれます。

- us-east-1 (バージニア北部)
- us-east-2 (オハイオ)
- us-west-2 (オレゴン)
- ap-northeast-1 (東京)
- ap-northeast-2 (ソウル)
- ap-south-1 (ムンバイ)
- ap-southeast-1 (シンガポール)
- ap-southeast-2 (シドニー)
- eu-central-1 (フランクフルト)
- eu-west-1 (アイルランド)

最新のリージョンの可用性については、Amazon SageMaker AI デベロッパーガイド」の「Amazon SageMaker AI Canvas」を参照してください。 Amazon SageMaker

Amazon DocumentDB のベクトル検索

ベクトル検索は、距離または類似度メトリクスを使用して複数のベクトル表現を比較することで、特定のデータポイントに類似したデータポイントを見つけるための機械学習手法です。2 つのベクトルがベクトル空間内で互いに近い距離にあるほど、基盤となる項目が類似していると見なされます。この手法は、データのセマンティックな意味を把握するのに役立ちます。こうしたアプローチは、お勧め提案システム、自然言語処理、画像認識など、さまざまなアプリケーションで役立ちます。

Amazon DocumentDB のベクトル検索機能は、JSON ベースのドキュメントデータベースの柔軟性と豊富なクエリ機能にベクトル検索ならではのメリットを組み合わせています。既存の Amazon DocumentDB データや柔軟なドキュメントデータ構造を使用して、セマンティック検索エクスペリエンス、お勧め製品の提案、パーソナライゼーション、チャットボット、不正検出、異常検出などの機械学習や生成 AI のユースケースを構築するには、Amazon DocumentDB のベクトル検索が最適です。ベクトル検索は、Amazon DocumentDB 5.0 のインスタンスベースのクラスターで使用できます。

トピック

- ベクトルの挿入
- ベクトルインデックスの作成
- インデックス定義の取得
- ベクトルのクエリ
- 特徴量と制限事項
- ベストプラクティス

ベクトルの挿入

Amazon DocumentDB データベースへのベクトル挿入には、既存の挿入方法を使用できます。

例

次の例では、テストデータベース内に 5 つのドキュメントのコレクションが作成されています。各ドキュメントには、製品名とそれに対応するベクトル埋め込みの 2 つのフィールドが含まれています。

```
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]}
]);
```

ベクトルインデックスの作成

Amazon DocumentDB は、階層ナビゲーションスモールワールド方式 (HNSW) によるインデックス作成と、フラット圧縮付き転置ファイル方式 (IVFFlat) によるインデックス作成の両方をサポートしています。IVFFlat インデックスは、ベクトルを複数のリストに分離したうえで、クエリベクトルに最も近いリストの選択されたサブセットを検索します。一方、HNSW インデックスはベクトルデータを多層グラフに整理します。HNSW は IVFFlat に比べてビルド時間が多くかかりますが、クエリのパフォーマンスと再現性が向上します。IVFFlat とは異なり、HNSW にはトレーニングステップがないため、初期データロードなしでインデックスを生成できます。ほとんどのユースケースでは、ベクトル検索に HNSW インデックスタイプを使用します。

ベクトルインデックスを作成しない場合、Amazon DocumentDB は高精度な最近傍探索を実行し、完全な再現性を保証します。ただし、本番稼働シナリオでは、速度も重要となります。ベクトルインデックスを使用します。ベクトルインデックスは、一部のリコールを速度の向上と交換する可能性があります。なお、ベクトルインデックスの追加によりクエリ結果が異なる可能性があることに留意してください。

テンプレート

次の `createIndex` または `runCommand` テンプレートを使用して、ベクトルフィールドにベクトルインデックスを構築できます。

Using createIndex

> mongosh や Java などの特定のドライバーでは、 で`vectorOptions`パラメータを使用すると エラーが発生する`createIndex`可能性があります。このような場合は、 を使用しま す`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

mongosh や Java などの特定のドライバーでは、 で`vectorOptions`パラメータを使用すると`createIndex`エラーが発生する可能性があります。このような場合は、 を使用します`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"
      }]
  }
);
```

パラメータ	要件	データ型	説明	値 (複数可)
name	オプション	string	インデックスの名前を指定します。	英数字

パラメータ	要件	データ型	説明	値 (複数可)
type	オプション		エラーのタイプを指定します。	サポートされている: hnsw または ivfflat デフォルト: HNSW (エンジンパッチ 3.0.4574 以降)
dimensions	必須	整数	ベクトルデータ内のディメンション数を指定します。	最大ディメンション数 2,000 個。
similarity	必須	string	類似性の計算に使用される距離メトリクスを指定します。	• **euclidean** • **cosine** • **dotProduct**

パラメータ	要件	データ型	説明	値 (複数可)
lists	IVFFlat に必要です	整数	IVFFlat インデックスがベクトルデータをグループ化するために使用するクラスターの数を指定します。推奨される設定方式は、ドキュメント数最大 100 万件までについてはドキュメント 1,000 件当りのドキュメント数、ドキュメント数が最大 100 万件を超える場合には `sqrt(# of documents)` となります。	最小: 1 最大: 以下の 特徴量と制限事項 のインスタンスタイプごとのリストを参照してください。
m	オプション	整数	特定の HNSW インデックス向けの最大接続数を指定します。	デフォルト: 16 範囲: [2, 100]

パラメータ	要件	データ型	説明	値 (複数可)
efConstruction	オプション	整数	HNSW インデックスのグラフを作成するためのダイナミック候補リストのサイズを指定します。 efConstruction の値は、(2 × m) 以上にする必要があります。	デフォルト: 64 範囲 [4, 1000]

IVFFlat および m 向け lists や、HNSW 向け efConstruction などのサブパラメータの値を適切に設定することが重要です。検索の精度/再現性、ビルド時間、パフォーマンスに影響するためです。リスト値を大きくすると、リスト 1 件当りのベクトル数が減るため、クエリの速度が上がり、リージョンが小さくなります。一方、リージョンサイズが小さくなるほど再現性エラーが多くなり、精度が低下する恐れがあります。HNSW の場合、m や efConstruction の値を上げると検索精度が高まりますが、インデックスのビルド時間とサイズも増加します。以下の例を参照してください。

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
    }
  }
)
```

インデックス定義の取得

getIndexes コマンドを使用して、ベクトルインデックスを含むインデックスの詳細を表示できます。

例

```
db.collection.getIndexes()
```

出力例

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

ベクトルのクエリ

Amazon DocumentDB は、ベクトルをクエリするための 2 つのベクトル検索演算子をサポートしています。

クラシックベクトル検索演算子

次のテンプレートを使用して、ベクトルをクエリします。

```
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]
      }
    }
  }
]);
```

パラメータ	要件	型	説明	値 (複数可)
vectorSearch	必須	オペレーター	$search コマンド内でベクトルをクエリする際に使用します。	

パラメータ	要件	型	説明	値 (複数可)
vector	必須	array	類似ベクトルの検索に使用するクエリベクトルを示します。	
path	必須	string	ベクトルフィールドの名前を定義します。	
k	必須	整数	検索から返される結果の最大数を指定します。	
similarity	必須	string	類似性の計算に使用される距離メトリクスを指定します。	• **euclidean** • **cosine** • **dotProduct**

パラメータ	要件	型	説明	値 (複数可)
probes	オプション	整数	ベクトル検索において検査対象とするクラスターの数。値を大きくすると、速度が低下する代わりに再現性が向上します。これは、最近傍検索のリスト数に設定できます (この時点では、プランナーはインデックスを使用しません)。微調整の開始点とするための推奨設定は `sqrt(# of lists)` です。	デフォルト: 1
efSearch	オプション	整数	HNSW インデックスが検索時に使用するダイナミック候補リストのサイズを指定します。`efSearch` の値が高いほど、速度は低下しますが再現性が向上します。	デフォルト: 40 範囲: [1, 1000]

efSearch (HNSW) または probes (IVFFlat) の値をファインチューニングして、もっとも望ましいパフォーマンスと精度を達成することが重要です。次のオペレーションの例を参照してください。

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
      }
    }
  }
]);
```

出力例

このオペレーションによる出力は、次のようになります。

```
{ "_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 演算子 (Amazon DocumentDB 8.0 以降で利用可能)

次のテンプレートを使用して、ベクトルをクエリします。

```
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],
  }
}])
```

特徴量と制限事項

バージョン互換性

- Amazon DocumentDB のベクトル検索は、Amazon DocumentDB 5.0 以降のインスタンスベースのクラスターでのみ使用できます。

ベクトル

- Amazon DocumentDB は、最大 2,000 個のディメンションのベクトルをインデックス化できます。ただし、インデックスなしであれば最大 16,000 個のディメンションを保存できます。

インデックス

- IVFFlat インデックス作成の場合、lists パラメータの推奨設定方法は、ドキュメント数最大 100 万件までについては 1000 件あたりのドキュメント数、ドキュネント数が最大 100 万を超える場合には `sqrt(# of documents)` となります。作業メモリの制限により、Amazon DocumentDB はディメンション数に応じて、lists パラメータの特定の最大値をサポートします。参考までに、次の表は、ディメンション数が 500、1000、2,000 の場合のベクトルのリストパラメータの最大値を示します。

インスタンスタイプ	ディメンション数 500 の場合のリスト	ディメンション数 1000 の場合のリスト	ディメンション数 2000 の場合のリスト
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
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

- `compound`、`sparse`、`partial` などの他のインデックスオプションは、ベクトルインデックスではサポートされていません。
- Amazon DocumentDB 5.0 の HNSW インデックスでは、並列インデックスビルドはサポートされていません。

ベクトルクエリ

- ベクトル検索クエリでは、最適な結果を得るために、`probes` や `efSearch` などのパラメータを微調整することが重要です。`probes` または `efSearch` パラメータの値が高いほど、再現性は高くなりますが、速度が低下します。検索パラメータの微調整の開始点としての推奨設定は `sqrt(# of lists)` です。

ベストプラクティス

Amazon DocumentDB でベクトル検索を使用するためのベストプラクティスについて説明します。新しいベストプラクティスが確認されると、このセクションは更新されます。

- フラット圧縮付き転置ファイル方式 (IVFFlat) によるインデックス作成には、類似点に基づいてデータポイントをクラスター化および整理することが含まれます。インデックスをより効果的にするには、インデックスを作成する前に一部のデータをロードします。
- ベクトル検索クエリでは、最適な結果を得るために、probes や efSearch などのパラメータをファインチューニングすることが重要です。probes または efSearch パラメータの値が高いほど、再現性は高くなりますが、速度が低下します。probes パラメータの微調整の開始点としての推奨設定は sqrt(lists) です。

リソース

- Amazon DocumentDB のベクトル検索がブログで一般公開されました AWS
- セマンティック検索コードのサンプル
- Amazon DocumentDB ベクトル検索コードのサンプル

変更ストリーム AWS Lambda での の使用

Amazon DocumentDB は と統合 AWS Lambda されているため、Lambda 関数を使用して変更ストリーム内のレコードを処理できます。Lambda イベントソースマッピングは、Lambda 関数を呼び出して、Lambda を直接呼び出さない Amazon DocumentDB イベントを処理するために使用できるリソースです。Amazon DocumentDB 変更ストリームをイベントソースとして利用すると、データの変更に対応するイベント駆動型アプリケーションを構築できます。例えば、Lambda 関数を使用して、新しいドキュメントを処理したり、既存のドキュメントの更新を追跡したり、ドキュメントの削除を記録したりできます。

イベントソースマッピングの設定では、Amazon DocumentDB 変更ストリームから Lambda 関数にレコードを送信するようにできます。イベントは一度に 1 つずつ送信することも、バッチ処理して効率を上げることもでき、その場合は順番に処理されます。イベントソースマッピングのバッチ処理動作は、特定のウィンドウ時間 (0 ~ 300 秒) またはバッチレコード数 (最大 10000 レコード) に基づいて設定できます。複数のイベントソースマッピングを作成することで、複数の Lambda 関数で同じデータを処理したり、1 つの関数で複数のストリームの項目を個別に処理したりできます。

関数からエラーが返された場合、Lambda は成功するまでバッチの実行を再試行します。変更ストリームのイベントの有効期限が切れた場合は、イベントソースマッピングを無効にします。この場合は、新しいイベントソースマッピングを作成し、任意の開始位置に設定できます。Lambda イベントソースマッピングは、ポーラーに備わる分散性が理由で、少なくとも 1 回イベントを処理します。その結果、まれに Lambda 関数が重複するイベントを受け取ることがあります。 AWS Lambda

関数を操作するためのベストプラクティスに従い、べき等関数を構築して、重複するイベントに関連する問題を回避します。詳細については、「 AWS Lambda デベロッパーガイド」のAmazon DocumentDB AWS Lambda console での の使用」を参照してください。

パフォーマンスのベストプラクティスとして、Lambda 関数は存続期間を短くする必要があります。また、不必要な処理の遅延が発生するのを防ぐため、複雑なロジックは実行しないでください。特に高速ストリームの場合は、同期的に長時間実行する Lambda よりも、非同期的な後処理ステップ関数ワークフローをトリガーすることをお勧めします。詳細については AWS Lambda、「 AWS Lambda デベロッパーガイド」を参照してください。

制限事項

Amazon DocumentDB を使用する際に考慮すべき制限は次のとおりです AWS Lambda。

- AWS Lambda は現在、Amazon DocumentDB 4.0 および 5.0 でのみサポートされています。
- AWS Lambda は現在、Elastic クラスターまたはグローバルクラスターではサポートされていません。
- AWS Lambda ペイロードサイズは 6MB を超えることはできません。Lambda バッチサイズの詳細については、AWS Lambda 開発者ガイドの「Lambda イベントソースマッピング」セクションの「バッチの動作」を参照してください。

Amazon DocumentDB の MongoDB との互換性

Amazon DocumentDB は、MongoDB 4.0、MongoDB 5.0、MongoDB 8.0 などの MongoDB 互換性をサポートしています。 MongoDB MongoDB 互換性とは、現在お使いの MongoDB データベースで使用しているアプリケーション、ドライバ、ツールの大部分を Amazon DocumentDB でもほとんど変更することなく使用できることを意味します。ここでは、Amazon DocumentDB の新機能、利用開始、移行パス、機能的な違いなど、Amazon DocumentDB の MongoDB との互換性について知っておくべきことをすべて説明します。

トピック

- MongoDB 8.0 の互換性
- MongoDB 5.0 との互換性
- MongoDB 4.0 の互換
- 機能の違い: Amazon DocumentDB と MongoDB
- Amazon DocumentDB でサポートされている MongoDB API、オペレーション、およびデータ型

MongoDB 8.0 の互換性

トピック

- Amazon DocumentDB 8.0 の新機能
- Amazon DocumentDB 8.0 の使用を開始する
- Amazon DocumentDB 5.0 または 8.0 へのアップグレードまたは移行
- 機能的な違い

Amazon DocumentDB 8.0 の新機能

Amazon DocumentDB 8.0 はクエリパフォーマンスを最大 7 倍向上させ、圧縮率を最大 5 倍向上させ、高性能アプリケーションを低コストで構築できるようにします。Amazon DocumentDB 8.0 は MongoDB 8.0 と完全なワイヤプロトコル互換性があります。以下の概要では、Amazon DocumentDB 8.0 で導入された主要な機能をいくつか紹介します。新機能の全リストを見るには、「リリースノート」 を参照してください。

- MongoDB 8.0 API ドライバーのサポートを追加することで、MongoDB 8.0 との互換性を提供します。MongoDB API バージョン 6.0 および 7.0 を使用して構築されたアプリケーションもサポートしています。
- Amazon DocumentDB 8.0 の新しいクエリプランナーは、集約パイプラインの最適化と個別のコマンドのサポートに加えて、集約ステージ演算子のパフォーマンスの向上を拡張します。
- Amazon DocumentDB 8.0 は照合とビューをサポートしています。
- $replaceWith、$vectorSearch、$merge、$set、$unset、$bucket、3 つの新しい集計演算子 $pow、$rand、$dateTrunc の 6 つの新しい集計ステージを提供します。
- テキストインデックスの新しいバージョン: Amazon DocumentDB 8.0 のテキストインデックス v2 では、追加のトークンが導入され、テキスト検索機能が強化されています。
- 並列ベクトルインデックス構築により、Amazon DocumentDB 8.0 はインデックス構築時間を最大 30 倍短縮します。
- ストレージ制限を 256 TiB に引き上げました。

Amazon DocumentDB 8.0 の使用を開始する

Amazon DocumentDB 8.0 の使用を開始するには、「」を参照してくださいAmazon DocumentDB の使用開始。 AWS マネジメントコンソール または AWS SDK AWS CLI、または を使用して、新しい Amazon DocumentDB 8.0 クラスターを作成できます CloudFormation。Amazon DocumentDB に接続する場合、MongoDB 5.0 以降と互換性のある MongoDB ドライバまたはユーティリティを使用する必要があります。

> **Note**
>
> AWS SDK、 AWS CLI、または を使用する場合 CloudFormation、エンジンバージョンはデフォルトで 5.0.0 になります。新しい Amazon DocumentDB 8.0 クラスターを作成`engineVersion = 8.0.0`したり、新しい Amazon DocumentDB 4.0 クラスター`engineVersion = 4.0.0`を作成したり、新しい Amazon DocumentDB 3.6 クラスターを作成`engineVersion = 3.6.0`したりするには、 パラメータを明示的に指定する必要があります。特定の Amazon DocumentDB クラスターについては、 AWS CLI を使用してクラスターのバージョンを確認する`describe-db-clusters`か、Amazon DocumentDB マネジメントコンソールを使用して特定のクラスターのエンジンバージョン番号を表示できます。

Amazon DocumentDB 5.0 および 8.0 は、Amazon EC2 r8g Graviton4 プロセッサと、クラスターの r6gおよび t4.mediumインスタンスタイプなどの Graviton2 プロセッサをサポートし、サポートされているすべての で使用できます AWS リージョン (「」を参照インスタンス)。料金の詳細については、「Amazon DocumentDB (MongoDB 互換) の料金」 を参照してください。

Amazon DocumentDB 5.0 または 8.0 へのアップグレードまたは移行

MongoDB 3.6 または MongoDB 4.0 から Amazon DocumentDB 5.0 または Amazon DocumentDB 8.0 に移行するには、mongodump、mongorestore、mongoimport、mongoexport などのAWS DMSユーティリティを使用します。移行の手順については、「」を参照してくださいを使用した Amazon DocumentDB クラスターのアップグレード AWS Database Migration Service。

機能的な違い

Amazon DocumentDB 5.0 と 8.0 の機能的な違い

Amazon DocumentDB 8.0 のリリースでは、Amazon DocumentDB 5.0 と Amazon DocumentDB 8.0 の機能的な違いがあります。

- プランナー v1 は Amazon DocumentDB 5.0 のデフォルトのクエリプランナーですが、パフォーマンスの高いプランナー v3 は Amazon DocumentDB 8.0 のデフォルトです。
- ビュー、照合、$merge などの演算子を含む Amazon DocumentDB 8.0 の新機能は、Planner v3 とのみ互換性があります。
- Amazon DocumentDB 8.0 では、圧縮はデフォルトでオンになっており、Zstandard アルゴリズムを使用するように設定されています。さらに、Amazon DocumentDB 8.0 では、「enabled」は有効な選択肢ではなくなりました。Zstd、LZ4、なしから選択できます。
- Amazon DocumentDB 8.0 は、データベース接続に対して TLS 1.2 以降のみをサポートします。

> **Important**
>
> Amazon DocumentDB 8.0 には TLS 1.2 以降が必要です。TLS 1.0 および TLS 1.1 はサポートされなくなりました。

MongoDB 5.0 との互換性

トピック

- Amazon DocumentDB 5.0 の新機能
- Amazon DocumentDB 5.0.1 の新機能
- Amazon DocumentDB 5.0 の使用開始
- Amazon DocumentDB 5.0 へのアップグレードまたは移行
- 機能的な違い

Amazon DocumentDB 5.0 の新機能

Amazon DocumentDB 5.0 では、ストレージ制限やクライアント側のフィールドレベルの暗号化など、新機能が導入されています。Amazon DocumentDB 5.0 で導入された主要な機能をいくつか紹介します。新機能の全リストを見るには、「リリースノート」 を参照してください。

- すべてのインスタンスベースの Amazon DocumentDB クラスターとシャードベースのエラスティッククラスターのストレージ制限を 128 TiB に増大。
- Amazon DocumentDB 5.0 (エンジンバージョン 3.0.775) の導入
  - MongoDB 5.0 API ドライバーのサポート
  - クライアント側のフィールドレベル暗号化 (FLE) のサポート。Amazon DocumentDB クラスターにデータを書き込む前に、クライアント側でフィールドを暗号化できるようになりました。詳細については、「クライアント側のフィールドレベルの暗号化」を参照してください。
  - 新しい集約演算子: `$dateAdd`、`$dateSubtract`
  - `$elemMatch` 演算子付きのインデックスをサポートします。その結果、`$elemMatch` を持つクエリではインデックススキャンが行われます。

Amazon DocumentDB は、すべての MongoDB 5.0 機能をサポートしているわけではありません。Amazon DocumentDB 5.0 を構築する際、お客様から最も多く構築するよう求められた特徴と機能から逆算していきました。今後もお客様の要望をもとに、MongoDB 5.0 の機能を追加していく予定です。サポートされている APIs「」を参照してくださいAmazon DocumentDB でサポートされている MongoDB API、オペレーション、およびデータ型。

Amazon DocumentDB 5.0.1 の新機能

Amazon DocumentDB 5.0.1 は、Amazon DocumentDB 5.0 メジャーバージョンの最初のマイナーバージョンリリースです。マイナーバージョンには、新機能、セキュリティ修正、バグ修正などの改善が含まれています。マイナーバージョンアップグレードを使用して、5.0.0 から 5.0.1 にアップグ

レードできます。詳細については、「Amazon DocumentDB マイナーバージョンアップグレード」を参照してください。

- 新しい演算子 ($rand、、$pow、) を使用して集約機能を強化$dateToPartsしました$dateFromParts。
- インスタンスをモニタリングするためのアクティブな接続メトリクス。詳細については、「インスタンスクォータ」を参照してください。
- CRUD オペレーション (find、、insert、、、、findAndModifyupdatedeleteaggregate、、countdistinctgetMorea の CloudWatch の詳細なコマンドレベルのパフォーマンスメトリクス。オペレーションモニタリングを改善するための AvgDuration、P100Duration、、、、、 MaxConcurrentの各ディメンションを公開します。
- Amazon DocumentDB 5.0.1 は、データベース接続の TLS 1.2 以降のみをサポートしています。
- ガベージコレクションプロセスの全体的なヘルスに詳細情報を表示collstatsするためにgcRuntimeStatsdocumentFragmentStats、、、およびコレクションレベルのメトリクスの内訳を storageSegmentBaseおよび storageSegmentExtendedに追加しました。

Amazon DocumentDB 5.0 の使用開始

Amazon DocumentDB 5.0 の使用を開始するには、「」を参照してくださいAmazon DocumentDB の使用開始。 AWS マネジメントコンソール または AWS SDK AWS CLI、または を使用して、新しい Amazon DocumentDB 5.0 クラスターを作成できます CloudFormation。Amazon DocumentDB に接続する場合、MongoDB 5.0 以降と互換性のある MongoDB ドライバまたはユーティリティを使用する必要があります。

> **Note**
>
> AWS SDK、 AWS CLI、または を使用する場合 CloudFormation、エンジンバージョンはデフォルトで 5.0.0 になります。Amazon DocumentDB 4.0 クラスターを新規に作成する場合はパラメータ engineVersion = 4.0.0 を、あるいは Amazon DocumentDB 3.6 クラスターを新規に作成する場合はengineVersion = 3.6.0 を明示的に指定する必要があります。特定の Amazon DocumentDB クラスターについては、 AWS CLI を使用してクラスターのバージョンを確認するdescribe-db-clustersか、Amazon DocumentDB マネジメントコンソールを使用して特定のクラスターのエンジンバージョン番号を表示できます。

Amazon DocumentDB 5.0 および 8.0 は、Amazon EC2 r8g Graviton4 プロセッサと、クラスターの r6gおよび t4.mediumインスタンスタイプなどの Graviton2 プロセッサをサポートし、サポートされているすべての で使用できます AWS リージョン (「」を参照インスタンス)。料金の詳細については、「Amazon DocumentDB (MongoDB 互換) の料金」 を参照してください。

Amazon DocumentDB 5.0 へのアップグレードまたは移行

MongoDB 3.6 または MongoDB 4.0 から Amazon DocumentDB 5.0 に移行するには、mongodump、mongorestore、mongoimport、mongoexport などの AWS DMSユーティリティを使用します。移行の手順については、「」を参照してくださいを使用した Amazon DocumentDB クラスターのアップグレード AWS Database Migration Service。

機能的な違い

Amazon DocumentDB 4.0 と 5.0 の機能的な違い

Amazon DocumentDB 5.0 のリリースに伴い、Amazon DocumentDB 4.0 と Amazon DocumentDB 5.0 には機能上の違いがあります。

- バックアップビルトインロールが serverStatus をサポートするようになりました。アクション - バックアップロールを持つデベロッパーおよびアプリケーションは、Amazon DocumentDB クラスターの状態に関する統計を収集できます。
- replSetGetConfig 出力では SecondaryDelaySecs フィールドが slaveDelay を置き換えます。
- hello コマンドは isMaster を置き換えます - hello は Amazon DocumentDB クラスターの役割を説明するドキュメントを返します。
- Amazon DocumentDB 5.0 では、最初のネストレベルで $elemMatch 演算子によるインデックススキャンをサポートするようになりました。インデックススキャンは、クエリのみのフィルタに 1 つのレベルの $elemMatch フィルタがある場合はサポートされますが、ネストされた $elemMatch クエリが含まれる場合はサポートされません。

  たとえば、Amazon DocumentDB 5.0 では、ネストされたレベルに $elemMatch 演算子を含めても、Amazon DocumentDB 4.0 のように値を返しません。

```
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 では、「$」プロジェクションはすべてのドキュメントとすべてのフィールドを返します。Amazon DocumentDB 5.0 では、「$」プロジェクションを使用する find コマンドは、「$」プロジェクションと一致したフィールドのみを含むクエリパラメータに一致するドキュメントを返します。
- Amazon DocumentDB 5.0 では、$regex および $options クエリパラメータを指定した find コマンドは、「$regex と $options の両方にオプションを設定することはできません」というエラーを返します。
- Amazon DocumentDB 5.0 では、以下の場合に $indexOfCP は「-1」を返します。
  - 文字列式内に部分文字列がない場合、または
  - 開始値が終了値より大きい場合、または
  - 開始値がは文字列のバイト長より大きい場合。
- Amazon DocumentDB 4.0 では、開始位置が文字列の終了またはバイト長よりも大きい数値の場合、$indexOfCP は「0」を返します。
- Amazon DocumentDB 5.0 では、たとえば {"_id.nestedField" : 1} のような _id fields の射影演算は、投影されたフィールドのみを含むドキュメントが返されます。これに対し、Amazon DocumentDB 4.0 の場合、入れ子にされたフィールドプロジェクトコマンドではどのドキュメントも除外しません。

MongoDB 4.0 の互換

トピック

- Amazon DocumentDB 4.0 機能
- Amazon DocumentDB 4.0 の使用開始
- Amazon DocumentDB 4.0 へのアップグレードまたは移行
- 機能的な違い

Amazon DocumentDB 4.0 機能

Amazon DocumentDB 4.0 では、ACID トランザクションや変更ストリームの改良など、多くの新機能が導入されました。Amazon DocumentDB 4.0 で導入された主要な機能をいくつかを以下に要約します。機能の全リストを見るには、「リリースノート」 を参照してください。

- ACID トランザクション: Amazon DocumentDB は、複数のドキュメント、ステートメント、コレクション、およびデータベース間でトランザクションを実行する機能をサポートするようになりました。トランザクションは、Amazon DocumentDB クラスター内の 1 つまたは複数のドキュメントに対し、アトミックで一貫性、分離、耐久性 (ACID) のあるオペレーションを実行できるようにすることで、アプリケーション開発を簡素化します。詳細については、Amazon DocumentDB でのトランザクション を参照してください。
- 変更ストリーム: これで、クラスターレベル (`client.watch()` または `mongo.watch()`) とデータベース (`db.watch()`) で変更ストリームを開くことができるようになりました 。`startAtOperationTime` をクリックして変更ストリームカーソルを開き、最後に、変更ストリームの保持期間を 7 日 (以前は 24 時間) に延長できます。詳細については、「Amazon DocumentDB を用いた変更ストリームの使用」を参照してください。
- AWS Database Migration Service (AWS DMS): AWS DMS を使用して MongoDB 4.0 ワークロードを Amazon DocumentDB に移行できるようになりました。Amazon DocumentDB 3.6 と 4.0 の間のアップグレードを実行するために、MongoDB Amazon DocumentDB4.0 ターゲット、Amazon DocumentDB 3.6 ソースがサポートされ AWS DMS るようになりました。詳細については、AWS DMS ドキュメント を参照してください。
- パフォーマンスとインデックス作成: `$lookup` で、インデックスを使用できるようになり、1 フィールドまたは 1 フィールドと `_id` フィールドを含む投影による検索クエリは、インデックスから直接提供でき、コレクション (カバークエリ) から読み取る必要がなくなり、`hint()` と `findAndModify` をする機能、`$addToSet` のパフォーマンスの最適化、および全体的なインデッ

クスサイズを小さくするための改良が行われました。詳細については、「リリースノート」を参照してください。

- 演算子: Amazon DocumentDB 4.0 では、いくつかの新しい集計演算子がサポートされるようになりました:
  `$ifNull`、`$replaceRoot`、`$setIsSubset`、`$setIntersection`、`$setUnion`、`$setEquals`。Am
  DocumentDB でサポートされている MongoDB API、オペレーション、およびデータ型 でサポートされるすべての MongoDB API、オペレーション、およびデータタイプを確認できます。
- ロールベースのアクセスコントロール(RBAC): `ListCollection` と `ListDatabase` の両方のコマンドで、オプションとして `authorizedCollections` と `authorizedDatabases` パラメータを使用することができるようになり、ユーザーが `listCollections` と `listDatabase` ロールを必要とすることなくアクセス許可を持つコレクションとデータベースを一覧表示できるようになりました。また、`KillCursor` ロールを必要とすることなく自分のカーソルを強制終了させる機能もあります。

Amazon DocumentDB は、すべての MongoDB 4.0 機能をサポートしているわけではありません。Amazon DocumentDB 4.0 を構築する際、お客様から最も多く構築するよう求められた特徴と機能から逆算していきました。今後もお客様の要望をもとに、MongoDB 4.0 の機能を追加していく予定です。例えば、Amazon DocumentDB 4.0 では、現在、MongoDB 4.0 で導入されていた型変換演算子または文字列演算子をサポートしていません。サポートされている APIs「」を参照してくださいAmazon DocumentDB でサポートされている MongoDB API、オペレーション、およびデータ型。

Amazon DocumentDB 4.0 の使用開始

Amazon DocumentDB 4.0 の使用を開始するには、「」を参照してくださいAmazon DocumentDB の使用開始。 AWS マネジメントコンソール または AWS SDK AWS CLI、または を使用して、新しい Amazon DocumentDB 4.0 クラスターを作成できます CloudFormation。Amazon DocumentDB に接続する場合、MongoDB 4.0 以降と互換性のある MongoDB ドライバまたはユーティリティを使用する必要があります。

> **Note**
>
> AWS SDK、 AWS CLI、または を使用する場合 CloudFormation、エンジンバージョンはデフォルトで 5.0.0 になります。Amazon DocumentDB 4.0 クラスターを新規に作成する場合はパラメータ `engineVersion = 4.0.0` を、あるいは Amazon DocumentDB 3.6 クラスターを新規に作成する場合は`engineVersion = 3.6.0` を明示的に指定する必要があります。特定の Amazon DocumentDB クラスターについては、 AWS CLI を使用してクラスター

のバージョンを確認するdescribe-db-clustersか、Amazon DocumentDB マネジメントコンソールを使用して特定のクラスターのエンジンバージョン番号を表示できます。

Amazon DocumentDB 4.0 は、クラスターの r5、r6g、、および t4g.mediumインスタンスタイプをサポートしt3.medium、サポートされているすべての で使用できます AWS リージョン。Amazon DocumentDB 4.0 を使用しても追加コストは発生しません。料金の詳細については、Amazon DocumentDB (MongoDB 互換) の料金 を参照してください。

Amazon DocumentDB 4.0 へのアップグレードまたは移行

MongoDB 3.6 または MongoDB 4.0 から Amazon DocumentDB 4.0 に移行するには、mongodump、mongorestore、mongoimport、mongoexport などの AWS DMSユーティリティを使用します。同様に、同じツールを使用して Amazon DocumentDB 3.6 から Amazon DocumentDB 4.0 にアップグレードできます。移行の手順については、「」を参照してくださいを使用した Amazon DocumentDB クラスターのアップグレード AWS Database Migration Service。

機能的な違い

Amazon DocumentDB 3.6 と 4.0 の機能的な違い

Amazon DocumentDB 4.0のリリースに伴い、Amazon DocumentDB 3.6と Amazon DocumentDB 4.0 には機能上の違いがあります。

- ネストドキュメントの投影: Amazon DocumentDB 3.6 は、投影法を適用するときに、ネストドキュメントの最初のフィールドを考慮します。ただし、Amazon DocumentDB 4.0 はサブドキュメントを解析し、各サブドキュメントにも投影法を適用します。たとえば、投影が "a.b.c": 1 の場合、両方のバージョンでの動作は同じです。ただし、投影が {a:{b:{c:1}}} だとすると、Amazon DocumentDB 3.6 は投影を 'a' にのみ適用し、'b' や 'c' には適用しません。
- **minKey** と **maxKey** 動作: Amazon DocumentDB 4.0 では、{x:{$gt:MaxKey}} では何も返しませんが、{x:{$lt:MaxKey}} ではすべてを返します。
- ドキュメント比較の相違: サブドキュメント内 (例: b の {"_id" :1, "a" :{"b":1}})の 異なるタイプ (double、int、long) の数値比較では、数値データ型とドキュメントの各レベルにわたって一貫した出力が提供されるようになりました。

Amazon DocumentDB 4.0 と MongoDB 4.0 の機能的な違い

以下は、Amazon DocumentDB 4.0 と MongoDB 4.0 の機能的な違いです。

- パスに空のキーがある場合のルックアップ: コレクションに、配列内 (例: `{"x" : [ { "" : 10 }, { "b" : 20 } ]}`) に空のキーを持つドキュメントが含まれている場合、そしてクエリで使用されるキーが空の文字列 (例: `x.`) で終わる場合は、Amazon DocumentDB は配列内のすべてのドキュメントを走査するのでそのドキュメントを返しますが、一方 MongoDB はそのドキュメントを返しません。
- パスの**`$setOnInsert`** に沿った **`$`**: フィールド演算子 `$setOnInsert` は Amazon DocumentDBのパスの `$` との組み合わせでは動作しませんが、これは MongoDB 4.0 と同じです。

機能の違い: Amazon DocumentDB と MongoDB

Amazon DocumentDB (MongoDB 互換性) と MongoDB の機能上の違いを以下に示します。

トピック

- Amazon DocumentDB の機能上の利点
- 機能的な違いの更新
- MongoDB との機能的な違い

Amazon DocumentDB の機能上の利点

暗示的なトランザクション

Amazon DocumentDB では、すべての CRUD ステートメント (`findAndModify`、`update`、`insert`、`delete`) は、複数のドキュメントを変更するオペレーションにおいてもアトミック性および整合性を保証します。Amazon DocumentDB 4.0 の起動により、マルチステートメントおよびマルチコレクションオペレーションの ACID プロパティを提供する明示的なトランザクションがサポートされるようになりました。Amazon DocumentDB でのトランザクションの使用の詳細については、「」を参照してくださいAmazon DocumentDB でのトランザクション。

アトミック性および整合性の両方の動作を満たす複数のドキュメントを変更する Amazon DocumentDB のオペレーションの例を次に示します。

```
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 }]
})
```

updateMany や deleteMany などの一括オペレーションを構成する個々のオペレーションはアトミックですが、一括オペレーション全体はアトミックではありません。たとえば、個々の挿入オペレーションがエラーなしで正常に実行された場合、insertMany オペレーション全体はアトミックです。insertMany オペレーションでエラーが発生した場合、insertMany オペレーション内の各挿入ステートメントはアトミックオペレーションとして実行されます。insertMany、updateMany、および deleteMany オペレーションに ACID プロパティが必要な場合の場合は、トランザクションを使用することをお勧めします。

機能的な違いの更新

Amazon DocumentDB は、お客様より依頼のあった機能からさかのぼって、MongoDB との互換性を改善し続けています。このセクションでは、移行とアプリケーション構築を容易にするため、Amazon DocumentDB から削除された機能的な違いについて説明します。

トピック

- 配列インデックス作成
- マルチキーインデックス
- 文字列内の null 文字
- ロールベースアクセスコントロール
- $regex インデックス作成
- ネストされたドキュメントの投影

配列インデックス作成

2020 年 4 月 23 日以降、Amazon DocumentDB で 2,048 バイトを超える配列のインデックスを作成する機能がサポートされるようになりました。配列内の個々のキーの制限は依然として 2,048 バイトのままであり、これは MongoDB と一貫しています。

新しいインデックスを作成する場合、強化された機能を利用するためのアクションは必要ありません。既存のインデックスがある場合、インデックスを削除してから再作成することで、強化された機能を利用できます。強化された機能を使用した現在のインデックスバージョンは `"v" : 3` です。

> **Note**
>
> 本番稼働用クラスターの場合、インデックスを削除するとアプリケーションのパフォーマンスに影響する可能性があります。最初にテストし、本番稼働用システムに変更を加えるときは注意して続行します。さらに、インデックスの再作成にかかる時間は、コレクションの全体的なデータサイズによって異なります。

次のコマンドを使用して、インデックスのバージョンをクエリできます。

```
db.collection.getIndexes()
```

このオペレーションによる出力は、次のようになります。この出力では、インデックスのバージョンは "v" : 3 です。これは最新のインデックスバージョンです。

```
[
    {
        "v" : 3,
        "key" : {
        "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.test"
    }
]
```

マルチキーインデックス

2020 年 4 月 23 日より、Amazon DocumentDB では同じ配列内に複数のキーを持つ複合インデックスを作成する機能がサポートされるようになりました。

新しいインデックスを作成する場合、強化された機能を利用するためのアクションは必要ありません。既存のインデックスがある場合、インデックスを削除してから再作成することで、強化された機能を利用できます。強化された機能を使用した現在のインデックスバージョンは "v" : 3 です。

> **Note**
>
> 本番稼働用クラスターの場合、インデックスを削除するとアプリケーションのパフォーマンスに影響する可能性があります。最初にテストし、本番稼働用システムに変更を加えるときは注意して続行します。さらに、インデックスの再作成にかかる時間は、コレクションの全体的なデータサイズによって異なります。

次のコマンドを使用して、インデックスのバージョンをクエリできます。

```
db.collection.getIndexes()
```

このオペレーションによる出力は、次のようになります。この出力では、インデックスのバージョンは "v" : 3 です。これは最新のインデックスバージョンです。

```
[
```

```
    {
        "v" : 3,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.test"
    }
]
```

文字列内の null 文字

2020 年 6 月 22 日現在、Amazon DocumentDB が文字列で null 文字 (`'\0'`) をサポートするようになりました。

ロールベースアクセスコントロール

2020 年 3 月 26 日現在、Amazon DocumentDB は組み込みロールによるロールベースのアクセスコントロール (RBAC) をサポートしています。詳細については ロールベースのアクセスコントロール を参照してください。

`$regex` インデックス作成

2020 年 6 月 22 日現在、Amazon DocumentDB は `$regex` 演算子がインデックスを活用する機能をサポートするようになりました。

`$regex` 演算子を使用してインデックスを活用するには、`hint()` コマンドを使用する必要があります。`hint()` を使用する際は、`$regex` を適用するフィールドの名前を指定する必要があります。たとえば、`product` というフィールドにインデックス名 `p_1` というインデックスがある場合、`db.foo.find({product: /^x.*/}).hint({product:1})` では `p_1` インデックスを使用しますが、`db.foo.find({product: /^x.*/}).hint("p_1")` ではインデックスを使用しません。インデックスが選択されているかどうかを確認するには、`explain()` コマンドを使用するか、低速クエリのログ記録にプロファイラーを使用します。例えば、`db.foo.find({product: /^x.*/}).hint("p_1").explain()`。

> **Note**
>
> `hint()` メソッドは、一度に 1 つのインデックスでのみ使用できます。

$regex クエリでのインデックスの使用は、プレフィックスを利用し、i、m、または o 正規表現オプションを指定しない正規表現クエリに最適化されます。

$regex でインデックスを使用する場合は、重複値の数がコレクション内のドキュメントの総数の 1% 未満であるごく一部のフィールドにインデックスを作成することをお勧めします。例えば、コレクションに 100,000 個のドキュメントが含まれている場合、同じ値が発生する回数が 1000 回以下のフィールドにのみインデックスを作成します。

ネストされたドキュメントの投影

Amazon DocumentDB と MongoDB 3.6 における $project 演算子と機能の差は、Amazon DocumentDB 4.0 で解決されていますが、Amazon DocumentDB 3.6 では未サポートのままです。

Amazon DocumentDB 3.6 は、投影を適用するときにネストされたドキュメントの最初のフィールドのみを考慮しますが、MongoDB 3.6 はサブドキュメントを解析し、各サブドキュメントにも投影を適用します。

例えば、投影が “a.b.c”: 1 とすると、この動作は Amazon DocumentDB と MongoDB の両方で期待どおりに機能します。ただし、投影が {a:{b:{c:1}}} だとすると、Amazon DocumentDB 3.6 は投影を a にのみ適用し、b や c には適用しません。Amazon DocumentDB 4.0では、投影 {a:{b:{c:1}}} は a、b、およびc に適用されます。

MongoDB との機能的な違い

トピック

- $vectorSearch operator
- OpCountersCommand
- 管理データベースとコレクション
- cursormaxTimeMS
- explain()
- インデックスビルド
- パスに空のキーでルックアップ
- MongoDB API、オペレーション、およびデータ型
- mongodump および mongorestore ユーティリティ
- 結果の順序
- 再試行可能な書き込み
- スパースインデックス

- $all 式内での $elemMatch の使用
- フィールド名のドル ($) とドット (.)
- $lookup
- $natural および逆順のソート

`$vectorSearch` operator

Amazon DocumentDB は、独立した演算子として $vectorSearch をサポートしていません。代わりに、$search 演算子内の vectorSearch をサポートします。詳細については、「Amazon DocumentDB のベクトル検索」を参照してください。

`OpCountersCommand`

Amazon DocumentDB の OpCountersCommand 動作は MongoDB の opcounters.command 動作と次のように異なります。

- MongoDB は挿入、更新、opcounters.command 削除を除くすべてのコマンドをカウントしますが、Amazon DocumentDB はOpCountersCommand コマンドも除外します。find
- Amazon DocumentDB は、一部の内部コマンドを OpCountersCommand に加算します。

管理データベースとコレクション

Amazon DocumentDB は管理者データベースまたはローカルデータベースをサポートしていません。また、MongoDB system.* または startup_log コレクションもサポートしていません。

`cursormaxTimeMS`

Amazon DocumentDB では、cursor.maxTimeMS はカウンターを getMore リクエストごとにリセットします。したがって、3000 ミリ秒 maxTimeMS が指定されている場合、クエリは 2800 ミリ秒 になり、getMore リクエストは 300 ミリ秒 かかり、カーソルはタイムアウトしません。カーソルがタイムアウトするのは、単一のオペレーション (クエリまたは個々の getMore リクエスト) の場合のみです。指定された maxTimeMS よりも多くかかります。さらに、カーソルの実行時間をチェックするスイーパーは、5 分単位で実行されます。

explain()

Amazon DocumentDB は、分散型で耐障害性のある自己修復ストレージシステムを利用する専用データベースエンジンで MongoDB 3.6、4.0、5.0、8.0 APIs をエミュレートします。その結果、ク

エリプランと explain() の出力は、Amazon DocumentDB と MongoDB の間で異なる場合があります。クエリプランを制御する場合は、$hint 演算子を使用して優先インデックスの選択を強制できます。

インデックスビルド

Amazon DocumentDB では、任意の時点でコレクションにおいてインデックスの作成が 1 つのみ許可されます。フォアグラウンドまたはバックグラウンドのいずれかで実行されます。インデックス構築が現在進行中のときに、createIndex() または dropIndex() などの操作が同じコレクションで発生した場合、新しく試行された操作は失敗します。

デフォルトでは、Amazon DocumentDB と MongoDB バージョン 4.0 のインデックスビルドはフォアグラウンドで行われます。MongoDB バージョン 4.2 以降では、createIndexes またはそのシェルヘルパー、createIndex() および createIndexes() に指定されている場合、バックグラウンドインデックス構築オプションは無視されます。

有効期限 (TTL) インデックスは、インデックスの構築が完了した後、ドキュメントの期限切れが開始されます。

パスに空のキーでルックアップ

パスの一部として (例 : x.、x..b) 空の文字列を含むキーを調べると、オブジェクトには空の文字列キーパス (例 : {"x" : [{ "" : 10 }, { "b" : 20 }]}) が配列内にあり、Amazon DocumentDB は MongoDB で同じルックアップを実行する場合とは異なる結果を返します。

MongoDB では、空の文字列キーがパスルックアップの末尾にない場合に、配列内の空のキーパスルックアップが期待どおりに機能します。ただし、空の文字列キーがパスルックアップの末尾にある場合、配列は検索されません。

ただし、Amazon DocumentDB では、配列内の最初の要素のみが読み取られます。getArrayIndexFromKeyString は空の文字列を0 に変換するため、文字列キールックアップは配列インデックスルックアップとして扱われるためです。

MongoDB API、オペレーション、およびデータ型

Amazon DocumentDB は、MongoDB 3.6、4.0、5.0、および 8.0 APIs。サポートされている機能の最新リストについては、「Amazon DocumentDB でサポートされている MongoDB API、オペレーション、およびデータ型」を参照してください。

mongodump および mongorestore ユーティリティ

Amazon DocumentDB は管理者データベースをサポートしないため、mongodump または mongorestore ユーティリティを使用するときに、管理者データベースをダンプまたは復元しません。mongorestore を使用して、Amazon DocumentDB の新しいデータベースを作成するときに、復元オペレーションに加えてユーザーロールを再作成する必要があります。

> **MongoDB データベースツールのバージョン要件**
>
> Amazon DocumentDB には、バージョン 100.11.0 までの MongoDB データベースツールを使用します。 Amazon DocumentDB ダウンロードするには、MongoDB ウェブサイトの「MongoDB データベースツールのリリース」を参照してください。 MongoDB

結果の順序

Amazon DocumentDB では、結果セットの暗黙的な並べ替え順序は保証されません。結果セットの順序を確実にするには、sort() を使用してソート順を明示的に指定します。

次の例では、在庫フィールドに基づいて、インベントリ収集の項目を降順にソートします。

```
db.inventory.find().sort({ stock: -1 })
```

$sort 集約ステージを使用する場合は、$sort ステージが、集約パイプラインの最後のステージでない限り、並べ替え順序は保持されません。$sort 集約ステージと組み合わせた $group 集約ステージを使用する場合、$sort 集約ステージはおよび $first および $last アキュムレータにのみ適用されます。Amazon DocumentDB 4.0 では、$push のサポートが追加され、前の $sort ステージ並べ替え順序を尊重します。

再試行可能な書き込み

最新の MongoDB 互換ドライバーは、デフォルトで再試行可能な書き込みを有効にします。Amazon DocumentDB は、エンジンバージョン 8.0.2 以降の再試行可能な書き込みをサポートしています。以前のエンジンバージョンでは、再試行可能な書き込みはサポートされていないため、エラーを避けるために無効にする必要があります。

8.0.2 より前のエンジンバージョンでは、再試行可能な書き込みを有効にすると、次のようなエラーメッセージが表示されます。

```
{"ok":0,"errmsg":"Retryable writes are not supported","code":301,"name":"MongoError"}
```

再試行可能な書き込みは、接続文字列 (例えば、MongoClient("mongodb://my.mongodb.cluster/db?retryWrites=false")) または MongoClient コンストラクタのキーワード引数 (例えば、MongoClient("mongodb://my.mongodb.cluster/db", retryWrites=False)) で無効にできます。

エンジンバージョン 8.0.2 以降の再試行可能な書き込みの詳細については、「」を参照してくださいAmazon DocumentDB での再試行可能な書き込み。

スパースインデックス

クエリで作成したスパースなインデックスを使用するには、インデックスをカバーするフィールドで $exists 句を使用する必要があります。$exists を省略した場合、Amazon DocumentDB はスパースなインデックスを使用しません。

以下に例を示します。

```
db.inventory.count({ "stock": { $exists: true }})
```

スパースなマルチキーインデックスの場合、ドキュメントのルックアップによって値のセットが生成され、インデックス付きフィールドのサブセットのみが見つからない場合、Amazon DocumentDB は一意キー制約をサポートしません。例えば、createIndex({"a.b" : 1 }, { unique : true, sparse :true }) を入力すると、"a" : [{ "b" : 2 }, { "c" : 1 }] がインデックスに格納されているため、"a.c" はサポートされません。

$all 式内での $elemMatch の使用

Amazon DocumentDB は $elemMatch 演算子が $all 内にある表現の使用をサポートしていません。回避策として、次のように $and 演算子を $elemMatch で使用できます。

元のオペレーション:

```
db.col.find({
  qty: {
    $all: [
      { "$elemMatch": { part: "xyz", qty: { $lt: 11 } } },
      { "$elemMatch": { num: 40, size: "XL" } }
    ]
```

```
  }
})
```

更新されたオペレーション:

```
db.col.find({
  $and: [
    { qty: { "$elemMatch": { part: "xyz", qty: { $lt: 11 } } } },
    { qty: { "$elemMatch": { qty: 40, size: "XL" } } }
  ]
})
```

フィールド名のドル ($) とドット (.)

Amazon DocumentDB は、ネストされたオブジェクトで in、$in、$nin、または $all を使用した場合の、Dollar($) プレフィックス付きフィールドのクエリをサポートしていません。例えば、Amazon DocumentDB では以下のクエリは有効ではありません。

```
coll.find({"field": {"$all": [{ "$a": 1 }]}})
```

$lookup

Amazon DocumentDB は、等価一致 (左外部結合など) を行う機能をサポートし、相関しないサブクエリもサポートします。バージョン 8.0.2 以降、Amazon DocumentDB は簡潔な相関サブクエリ構文を含む相関サブクエリもサポートしています。相関サブクエリでは、式を評価する集計ステージのサブセットのみが`let`変数を参照できます。サポートされているステージと関連する制限のリストについては、「」を参照してください相関サブクエリでサポートされているステージ。8.0.2 より前のバージョンでは、相関サブクエリはサポートされていません。

$lookup でインデックスを利用する

`$lookup` ステージ演算子で、インデックスを利用できるようになりました 。ユースケースに基づいて、パフォーマンスを最適化するために使用できるインデックス作成アルゴリズムが複数あります。このセクションでは、`$lookup` のさまざまなインデックス作成アルゴリズムについて説明し、ワークロードに最適なオプションを選択するのに役立ちます。

デフォルトでは、Amazon DocumentDB は `allowDiskUse:false` が使用された場合はハッシュアルゴリズムを使用し、`allowDiskUse:true` が使用された場合はソートマージを使用します。

> **Note**
>
> allowDiskUse オプションは現在、find コマンド向けにはサポートされていません。このオプションは集約の一部としてのみサポートされます。で集約フレームワークを使用してallowDiskUse:true、メモリ制限を超える可能性のある大規模なクエリを処理します。

ユースケースによっては、クエリオプティマイザで別のアルゴリズムを使用するように強制することが望ましい場合があります。以下は、$lookup の集約演算子を利用できる、さまざまなインデックス作成アルゴリズムの例です。

- ネストループ: 外部コレクションが 1 GB 未満で、外部コレクションのフィールドにインデックスがある場合は、通常、ネストループプランがワークロードに役立ちます。ネスト・ループ・アルゴリズムを使用している場合、説明プランではステージが NESTED_LOOP_LOOKUP と表示されます。
- ソートマージ: ソートマージプランは、外部コレクションのルックアップに使用されるフィールドにインデックスがなく、作業データセットがメモリに収まらない場合にワークロードにとって有益です。ソートマージアルゴリズムが使用されている場合、説明プランではステージを SORT_LOOKUP と表示されます。
- ハッシュ: 外部コレクションが 1 GB 未満で、作業データセットがメモリに収まる場合は、通常、ハッシュプランがワークロードにとって有益です。ハッシュアルゴリズムが使用されている場合、説明プランはステージを HASH_LOOKUP と表示されます。

クエリに対して explain を使用することで、$lookup 演算子に使用されているインデックス作成アルゴリズムを特定できます。以下に例を示します。

```
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
}
```

explain() メソッドを使用する代わりに、プロファイラを使用して、$lookup 演算子の使用で利用されているアルゴリズムを表示することができます。プロファイラーの詳細については、「」を参照してくださいAmazon DocumentDB オペレーションのプロファイリング。

planHint を使用する

クエリオプティマイザで別のインデックス作成アルゴリズムと $lookup を強制的に使用する場合、planHint を使用できます。これを行うには、集約ステージオプションのコメントを使用して、別のプランを強制します。コメントの構文の例を以下に示します。

```
comment : {
    comment :  "<string>",
    lookupStage : { planHint : "SORT" | "HASH" | "NESTED_LOOP" }
}
```

以下は、planHint を使用してクエリオプティマイザに HASH インデックスアルゴリズムの使用を強制する例です。

```
db.foo.aggregate(
   [
      {
         $lookup:
            {
               from: "foo",
               localField: "_id",
               foreignField: "_id",
               as: "joined"
            },
      }
   ]
),
{
   comment : "{ \"lookupStage\" : { \"planHint\": \"HASH\" }}"
```

どのアルゴリズムがワークロードに最適かをテストするには、executionStats パラメータにある explain メソッドを使用して、$lookup ステージの実行時間を測定することができ、これはインデックス作成アルゴリズム (つまり、HASH / SORT/NESTED_LOOP) を変更しながらできます。

次の例は executionStats を使用して、$lookup ステージの実行時間を測定するには、SORT アルゴリズムを使用します。

```
db.foo.explain("executionStats").aggregate(
   [
      {
         $lookup:
            {
               from: "foo",
               localField: "_id",
               foreignField: "_id",
               as: "joined"
            },
      }
   ]
),
{
   comment : "{ \"lookupStage\" : { \"planHint\": \"SORT\" }}"
```

`$natural` および逆順のソート

Amazon DocumentDB は、フォワードコレクションスキャンでのみ `$natural` をサポートします。逆順のコレクションスキャン (`{$natural: -1}`) は `MongoServerError` につながります。

Amazon DocumentDB でサポートされている MongoDB API、オペレーション、およびデータ型

Amazon DocumentDB (MongoDB 互換) は、MongoDB ワークロードをサポートする、高速、スケーラブル、高可用性、完全管理のドキュメントデータベース· サービスです。Amazon DocumentDB は、MongoDB 3.6、4.0、5.0、および 8.0 APIs。このセクションでは、サポートされている機能を一覧表示します。MongoDB APIs とドライバーの使用については、MongoDB コミュニティフォーラムを参照してください。Amazon DocumentDB サービスの使用に関するサポートについては、適切な AWS サポートチームにお問い合わせください。Amazon DocumentDB と MongoDB の機能の違いについては、「」を参照してください機能の違い: Amazon DocumentDB と MongoDB。

内部専用であるか、フルマネージドサービスには適用されない、MongoDB コマンドと演算子は、サポートされておらず、サポートされている機能のリストにも含まれていません。

初めてのリリース以来 50 を超える新しい機能を追加しており、今後もお客様が必要とする機能を遡って提供していきます。最新のローンチに関する情報は、「Amazon DocumentDB に関する発表」を参照してください。

サポート対象外の機能をリクエストしたい場合は、アカウント ID、リクエストする機能、およびユースケースを記載した E メールを Amazon DocumentDB サービスチーム まで送信してお知らせください。

トピック

- データベースコマンド
- クエリおよびプロジェクション演算子
- 更新演算子
- 地理空間
- カーソルメソッド
- 集約パイプライン演算子
- データ型
- インデックス

データベースコマンド

トピック

- 管理コマンド
- 集計
- 認証
- 診断コマンド
- クエリ操作と書き込み操作
- ロール管理コマンド
- セッションコマンド
- ユーザー管理
- シャーディングコマンド

管理コマンド

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
上限付きコレクション	いいえ	いいえ	いいえ	いいえ	いいえ
cloneCollectionAsCapped	いいえ	いいえ	いいえ	いいえ	いいえ
collMod	部分的	部分的	部分的	部分的	部分的
collMod: expireAfterSeconds	はい	はい	はい	はい	はい
collMod: 非表示	いいえ	いいえ	はい (5.0.2 以降)	はい (8.0.1 以降)	いいえ
convertToCapped	いいえ	いいえ	いいえ	いいえ	いいえ
copydb	いいえ	いいえ	いいえ	いいえ	いいえ
作成	はい	はい	はい	はい	はい
createView	いいえ	いいえ	なし	はい	いいえ
createIndexes	はい	はい	はい	はい	はい
currentOp	はい	はい	はい	はい	はい
drop	はい	はい	はい	はい	はい
dropDatabase	はい	はい	はい	はい	はい

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
dropIndexes	はい	はい	はい	はい	はい
filemd5	いいえ	いいえ	いいえ	いいえ	いいえ
getAuditConfig	いいえ	はい	はい	はい	いいえ
killCursors	はい	はい	はい	はい	はい
killOp	はい	はい	はい	はい	はい
listCollections*	はい	はい	はい	はい	はい
listDatabases	はい	はい	はい	はい	はい
listIndexes	はい	はい	はい	はい	はい
reIndex	いいえ	なし	はい	はい	いいえ
renameCollection	はい	はい	はい	はい	いいえ
setAuditConfig	いいえ	はい	はい	はい	いいえ

* コンソールでは、`type` キーはサポートされていません。

集計

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
aggregate	はい	はい	はい	はい	はい
count	はい	はい	はい	はい	はい

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
distinct	はい	はい	はい	はい	はい
mapReduce	いいえ	いいえ	なし	はい	いいえ

認証

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
認証	はい	はい	はい	はい	はい
ログアウト	はい	はい	はい	はい	はい

診断コマンド

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
buildInfo	はい	はい	はい	はい	はい
collStats	はい	はい	はい	はい	はい
connPoolStats	いいえ	いいえ	いいえ	いいえ	いいえ
connectionStatus	はい	はい	はい	はい	はい
dataSize	はい	はい	はい	はい	はい
dbHash	いいえ	いいえ	いいえ	いいえ	いいえ
dbStats	はい	はい	はい	はい	はい

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
explain	はい	はい	はい	はい	はい
説明: executionStats	はい	はい	はい	はい	はい
機能	いいえ	いいえ	いいえ	いいえ	いいえ
hostInfo	はい	はい	はい	はい	はい
listCommands	はい	はい	はい	はい	はい
planCacheClearFilters	いいえ	なし	はい	はい	いいえ
planCacheListFilters	いいえ	なし	はい	はい	いいえ
planCacheSetFilter	いいえ	なし	はい	はい	いいえ
プロファイラー	あり	あり	あり	あり	いいえ
serverStatus	はい	はい	はい	はい	はい
top	はい	はい	はい	はい	はい

クエリ操作と書き込み操作

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
Change streams	あり	あり	あり	あり	いいえ
削除	はい	はい	はい	はい	はい
find	はい	はい	はい	はい	はい
findAndModify	はい	はい	はい	はい	はい
getLastError	いいえ	いいえ	いいえ	いいえ	いいえ
getMore	はい	はい	はい	はい	はい
getPrevError	いいえ	いいえ	いいえ	いいえ	いいえ
GridFS	はい	はい	はい	はい	いいえ
insert	はい	はい	はい	はい	はい
parallelCollectionScan	いいえ	いいえ	いいえ	いいえ	いいえ
resetError	いいえ	いいえ	いいえ	いいえ	いいえ
更新	はい	はい	はい	はい	はい
ReplaceOne	はい	はい	はい	はい	はい

ロール管理コマンド

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
createRole	はい	はい	はい	はい	いいえ
dropAllRoles	はい	はい	はい	はい	いいえ
dropRole	はい	はい	はい	はい	いいえ
grantRole	はい	はい	はい	はい	いいえ
revokeRole	はい	はい	はい	はい	いいえ
rolesInfo	はい	はい	はい	はい	いいえ
updateRole	はい	はい	はい	はい	いいえ

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
ile	はい	はい	はい	はい	いいえ

セッションコマンド

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
abortTransaction	いいえ	はい	はい	はい	いいえ
commitTransaction	いいえ	はい	はい	はい	いいえ
endSessions	いいえ	いいえ	いいえ	いいえ	いいえ
killAllSessions	いいえ	はい	はい	はい	いいえ
killAllSessionsByPattern	いいえ	いいえ	いいえ	いいえ	いいえ
killSessions	いいえ	はい	はい	はい	いいえ
refreshSessions	いいえ	いいえ	いいえ	いいえ	いいえ
startSession	いいえ	はい	はい	はい	いいえ

ユーザー管理

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
createUser	はい	はい	はい	はい	はい
dropAllUsersFromDatabase	はい	はい	はい	はい	はい
dropUser	はい	はい	はい	はい	はい
grantRolesToUser	はい	はい	はい	はい	はい
revokeRolesFromUser	はい	はい	はい	はい	はい
updateUser	はい	はい	はい	はい	はい
usersInfo	はい	はい	はい	はい	はい

シャーディングコマンド

コマンド	Elastic クラスター
abortReshardCollection	いいえ
addShard	いいえ
addShardToZone	いいえ
balancerCollectionStatus	いいえ
balancerStart	いいえ
balancerStatus	いいえ

コマンド	Elastic クラスター
balancerStop	いいえ
checkShardingIndex	いいえ
clearJumboFlag	いいえ
cleanupOrphaned	いいえ
cleanupReshardCollection	いいえ
commitReshardCollection	いいえ
enableSharding	はい
flushRouterConfig	いいえ
getShardMap	いいえ
getShardVersion	いいえ
isdbgrid	いいえ
ListShards	いいえ
medianKey	いいえ
moveChunk	いいえ
movePrimary	いいえ
mergeChunks	いいえ
refineCollectionShardKey	いいえ
removeShard	いいえ
removeShardFromZone	いいえ
reshardCollection	いいえ

コマンド	Elastic クラスター
setAllowMigrations	いいえ
setShardVersion	いいえ
shardCollection	はい
shardingState	いいえ
split	いいえ
splitVector	いいえ
unsetSharding	いいえ
updateZoneKeyRange	いいえ

クエリおよびプロジェクション演算子

トピック

- 配列演算子
- ビット演算子
- コメント演算子
- 比較演算子
- 要素演算子
- 評価クエリ演算子
- 論理演算子
- プロジェクション演算子

配列演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$all	はい	はい	はい	はい	はい

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$elemMatch	はい	はい	はい	はい	はい
$size	はい	はい	はい	はい	はい

ビット演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$bitsAllSet	はい	はい	はい	はい	はい
$bitsAnySet	はい	はい	はい	はい	はい
$bitsAllClear	はい	はい	はい	はい	はい
$bitsAnyClear	はい	はい	はい	はい	はい

コメント演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$comment	はい	はい	はい	はい	はい

比較演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$eq	はい	はい	はい	はい	はい
$gt	はい	はい	はい	はい	はい

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$gte	はい	はい	はい	はい	はい
$in	はい	はい	はい	はい	はい
$lt	はい	はい	はい	はい	はい
$lte	はい	はい	はい	はい	はい
$ne	はい	はい	はい	はい	はい
$nin	はい	はい	はい	はい	はい

要素演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$exists	はい	はい	はい	はい	はい
$type	はい	はい	はい	はい	はい

評価クエリ演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$expr	いいえ	はい	はい	はい	いいえ
$jsonSchema	なし	はい	はい	はい	いいえ
$mod	はい	はい	はい	はい	はい
$regex	はい	はい	はい	はい	はい
$text	いいえ	なし	はい	はい	いいえ

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$where	いいえ	いいえ	いいえ	いいえ	いいえ

論理演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$and	はい	はい	はい	はい	はい
$nor	はい	はい	はい	はい	はい
$not	はい	はい	はい	はい	はい
$or	はい	はい	はい	はい	はい

プロジェクション演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$	はい	はい	はい	はい	はい
$elemMatch	はい	はい	はい	はい	はい
$meta	いいえ	なし	はい	はい	いいえ
$slice	はい	はい	はい	はい	はい

更新演算子

トピック

- 配列演算子
- ビット演算子

- フィールド演算子
- 更新修飾子

配列演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$	はい	はい	はい	はい	はい
$[]	はい	はい	はい	はい	はい
$[<identifier>]	はい	はい	はい	はい	はい
$addToSet	はい	はい	はい	はい	はい
$pop	はい	はい	はい	はい	はい
$pullAll	はい	はい	はい	はい	はい
$pull	はい	はい	はい	はい	はい
$push	はい	はい	はい	はい	はい

ビット演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$bit	はい	はい	はい	はい	はい

フィールド演算子

オペレーター	3.6	4.0	5.0	8.0	Elastic クラスター
$currentDate	はい	はい	はい	はい	はい
$inc	はい	はい	はい	はい	はい
$max	はい	はい	はい	はい	はい
$min	はい	はい	はい	はい	はい
$mul	はい	はい	はい	はい	はい
$rename	はい	はい	はい	はい	はい
$set	はい	はい	はい	はい	はい
$setOnInsert	はい	はい	はい	はい	はい
$unset	はい	はい	はい	はい	はい

更新修飾子

オペレーター	3.6	4.0	5.0	8.0	Elastic クラスター
$each	はい	はい	はい	はい	はい
$position	はい	はい	はい	はい	はい
$slice	はい	はい	はい	はい	はい
$sort	はい	はい	はい	はい	はい

地理空間

ジオメトリ指定子

クエリセレクター	3.6	4.0	5.0	8.0	Elastic クラスター
$box	いいえ	いいえ	いいえ	いいえ	いいえ
$center	いいえ	いいえ	いいえ	いいえ	いいえ
$centerSphere	いいえ	いいえ	いいえ	いいえ	なし
$geometry	はい	はい	はい	はい	はい
$maxDistance	はい	はい	はい	はい	はい
$minDistance	はい	はい	はい	はい	はい
$nearSphere	はい	はい	はい	はい	はい
$polygon	いいえ	いいえ	いいえ	いいえ	いいえ
$uniqueDocs	いいえ	いいえ	いいえ	いいえ	いいえ

クエリセレクター

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$geoIntersects	はい	はい	はい	はい	はい
$geoWithin	はい	はい	はい	はい	はい
$near	はい	はい	はい	はい	はい

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$nearSphere	はい	はい	はい	はい	はい
$polygon	いいえ	いいえ	いいえ	いいえ	いいえ
$uniqueDocs	いいえ	いいえ	いいえ	いいえ	いいえ

カーソルメソッド

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
cursor.batchSize()	はい	はい	はい	はい	はい
cursor.close()	はい	はい	はい	はい	はい
cursor.collation()	いいえ	いいえ	なし	はい	いいえ
cursor.comment()	はい	はい	はい	はい	はい
cursor.count()	はい	はい	はい	はい	はい
cursor.explain()	はい	はい	はい	はい	いいえ
cursor.forEach()	はい	はい	はい	はい	はい
cursor.hasNext()	はい	はい	はい	はい	はい
cursor.hint()	はい	はい	はい	はい	はい*

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
cursor.isClosed()	はい	はい	はい	はい	はい
cursor.isExhausted()	はい	はい	はい	はい	いいえ
cursor.itcount()	はい	はい	はい	はい	いいえ
cursor.limit()	はい	はい	はい	はい	いいえ
cursor.map()	はい	はい	はい	はい	いいえ
cursor.max()	いいえ	いいえ	はい (5.0.2 以降)	はい (8.0.1 以降)	いいえ
cursor.maxScan()	はい	はい	はい	はい	いいえ
cursor.maxTimeMS()	はい	はい	はい	はい	いいえ
cursor.min()	いいえ	いいえ	はい (5.0.2 以降)	はい (8.0.1 以降)	いいえ
cursor.next()	はい	はい	はい	はい	はい
cursor.noCursorTimeout()	いいえ	いいえ	いいえ	いいえ	いいえ
cursor.objsLeftInBatch()	はい	はい	はい	はい	いいえ
cursor.pretty()	はい	はい	はい	はい	いいえ

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
cursor.readConcern()	はい	はい	はい	はい	いいえ
cursor.readPref()	はい	はい	はい	はい	いいえ
cursor.returnKey()	いいえ	いいえ	いいえ	いいえ	いいえ
cursor.showRecordId()	いいえ	いいえ	いいえ	いいえ	いいえ
cursor.size()	はい	はい	はい	はい	いいえ
cursor.skip()	はい	はい	はい	はい	いいえ
cursor.sort()	はい	はい	はい	はい	いいえ
cursor.tailable()	いいえ	いいえ	いいえ	いいえ	いいえ
cursor.toArray()	はい	はい	はい	はい	いいえ

* インデックス `hint` がインデックス式でサポートされるようになりました。例えば、`db.foo.find().hint({x:1})`。

`cursor.min()` メソッドと `cursor.max()`メソッドには、使用するインデックスに名前`hint`を付ける が必要です。どちらのメソッドも同じフィールドを指定する必要があり、`$natural`ソートと組み合わせることはできません。コマンド`find`でのみサポートされています。

集約パイプライン演算子

トピック

- アキュムレータ式
- 算術演算子

- 配列演算子
- ブール演算子
- 比較演算子
- 条件式演算子
- データ型演算子
- データサイズ演算子
- 日付演算子
- リテラル演算子
- マージ演算子
- 自然演算子
- セット演算子
- ステージ演算子
- 文字列演算子
- システム変数
- テキスト検索演算子
- 型変換演算子
- 変数演算子
- Trigonometry 演算子
- ビット単位の演算子 (集計)
- タイムスタンプ演算子
- その他の演算子

アキュムレータ式

式	3.6	4.0	5.0	8.0	Elastic クラスター
$accumulator	-	-	いいえ	いいえ	なし
$addToSet	はい	はい	はい	はい	はい
$avg	はい	はい	はい	はい	はい

式	3.6	4.0	5.0	8.0	Elastic クラスター
$ 下部	-	-	-	はい (8.0.1 以降)	いいえ
$bottomN	-	-	-	はい (8.0.1 以降)	いいえ
$count	-	-	いいえ	はい (8.0.1 以降)	いいえ
$covarianCePop	いいえ	いいえ	いいえ	はい (8.0.2 以降)	いいえ
$covarianceSamp	いいえ	いいえ	いいえ	はい (8.0.2 以降)	いいえ
$denseRank	いいえ	いいえ	いいえ	はい (8.0.2 以降)	いいえ
$derivative	いいえ	いいえ	いいえ	いいえ	いいえ
$documentNumber	いいえ	いいえ	いいえ	はい (8.0.2 以降)	いいえ
$expMovingAvg	いいえ	いいえ	いいえ	いいえ	なし
$first	はい	はい	はい	はい	はい
$firstN	-	-	-	はい (8.0.1 以降)	いいえ
$integral	いいえ	いいえ	いいえ	いいえ	なし
$last	はい	はい	はい	はい	はい
$lastN	-	-	-	はい (8.0.1 以降)	いいえ

式	3.6	4.0	5.0	8.0	Elastic クラスター
$linearFill	いいえ	いいえ	いいえ	いいえ	いいえ
$locf	いいえ	いいえ	いいえ	いいえ	なし
$max	はい	はい	はい	はい	はい
$maxN	-	-	-	はい (8.0.1 以降)	いいえ
$median	-	-	-	はい (8.0.1 以降)	いいえ
$min	はい	はい	はい	はい	はい
$minN	-	-	-	はい (8.0.1 以降)	いいえ
$percentile	-	-	-	はい (8.0.1 以降)	いいえ
$push	はい	はい	はい	はい	はい
$rank	いいえ	いいえ	いいえ	はい (8.0.2 以降)	いいえ
$shift	いいえ	いいえ	いいえ	いいえ	いいえ
$stdDevPop	いいえ	いいえ	いいえ	はい (8.0.1 以降)	いいえ
$stdDevSamp	いいえ	いいえ	いいえ	はい (8.0.1 以降)	いいえ
$sum	はい	はい	はい	はい	はい
$top	-	-	-	はい (8.0.1 以降)	いいえ

式	3.6	4.0	5.0	8.0	Elastic クラスター
$topN	-	-	-	はい (8.0.1 以降)	いいえ

算術演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$abs	はい	はい	はい	はい	はい
$add	はい	はい	はい	はい	はい
$ceil	いいえ	はい	はい	はい	はい
$divide	はい	はい	はい	はい	はい
$exp	いいえ	はい	はい	はい	はい
$floor	いいえ	はい	はい	はい	はい
$ln	いいえ	はい	はい	はい	はい
$log	いいえ	はい	はい	はい	はい
$log10	いいえ	はい	はい	はい	はい
$mod	はい	はい	はい	はい	はい
$multiply	はい	はい	はい	はい	はい
$pow	いいえ	いいえ	はい (5.0.1 以降)	はい	いいえ
$round	-	-	いいえ	はい (8.0.1 以降)	いいえ

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$sqrt	なし	はい	はい	はい	はい
$subtract	はい	はい	はい	はい	はい
$trunc	いいえ	いいえ	いいえ	はい (8.0.1 以降)	いいえ

配列演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$arrayElemAt	はい	はい	はい	はい	はい
$arrayToObject	はい	はい	はい	はい	はい
$concatArrays	はい	はい	はい	はい	はい
$filter	はい	はい	はい	はい	はい
$first	-	-	はい	はい	いいえ
$in	はい	はい	はい	はい	はい
$indexOfArray	はい	はい	はい	はい	はい
$isArray	はい	はい	はい	はい	はい
$last	-	-	はい	はい	いいえ
$objectToArray	はい	はい	はい	はい	はい

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$range	はい	はい	はい	はい	はい
$reverseArray	はい	はい	はい	はい	はい
$sortArray	いいえ	いいえ	なし	はい	いいえ
$reduce	はい	はい	はい	はい	はい
$size	はい	はい	はい	はい	はい
$slice	はい	はい	はい	はい	はい
$zip	はい	はい	はい	はい	はい

ブール演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$and	はい	はい	はい	はい	はい
$not	はい	はい	はい	はい	はい
$or	はい	はい	はい	はい	はい

比較演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$cmp	はい	はい	はい	はい	はい
$eq	はい	はい	はい	はい	はい

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$gt	はい	はい	はい	はい	はい
$gte	はい	はい	はい	はい	はい
$lt	はい	はい	はい	はい	はい
$lte	はい	はい	はい	はい	はい
$ne	はい	はい	はい	はい	はい

条件式演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$cond	はい	はい	はい	はい	はい
$ifNull	はい	はい	はい	はい	はい
$switch	いいえ	はい	はい	はい	いいえ

データ型演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$type	はい	はい	はい	はい	はい

データサイズ演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$binarySize	-	-	いいえ	はい (8.0.1 以降)	いいえ
$bsonSize	-	-	いいえ	はい (8.0.1 以降)	いいえ

日付演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$dateAdd	いいえ	なし	はい	はい	はい
$dateDiff	-	-	はい	はい	いいえ
$dateFromParts	いいえ	いいえ	はい (5.0.1 以降)	いいえ	なし
$dateFromString	はい	はい	はい	はい	はい
$dateSubtract	いいえ	なし	はい	はい	はい
$dateToParts	いいえ	いいえ	はい (5.0.1 以降)	いいえ	なし
$dateToString	はい	はい	はい	はい	はい
$dateTrunc	-	-	いいえ	はい	いいえ
$dayOfMonth	はい	はい	はい	はい	はい

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$dayOfWeek	はい	はい	はい	はい	はい
$dayOfYear	はい	はい	はい	はい	はい
$hour	はい	はい	はい	はい	はい
$isoDayOfWeek	はい	はい	はい	はい	はい
$isoWeek	はい	はい	はい	はい	はい
$isoWeekYear	はい	はい	はい	はい	はい
$millisecond	はい	はい	はい	はい	はい
$minute	はい	はい	はい	はい	はい
$month	はい	はい	はい	はい	はい
$second	はい	はい	はい	はい	はい
$week	はい	はい	はい	はい	はい
$year	はい	はい	はい	はい	はい

リテラル演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$literal	はい	はい	はい	はい	はい

マージ演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$mergeObjects	はい	はい	はい	はい	はい

自然演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$natural	はい	はい	はい	はい	はい

セット演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$allElementsTrue	いいえ	はい	はい	はい	はい
$anyElementTrue	いいえ	はい	はい	はい	はい
$setDifference	いいえ	はい	はい	はい	はい
$setEquals	はい	はい	はい	はい	はい
$setIntersection	はい	はい	はい	はい	はい
$setIsSubset	はい	はい	はい	はい	はい
$setUnion	はい	はい	はい	はい	はい

ステージ演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$addFields	はい	はい	はい	はい	はい
$bucket	いいえ	いいえ	なし	はい	いいえ
$bucketAuto	いいえ	いいえ	いいえ	はい (8.0.2 以降)	いいえ
$changeStream	はい	はい	はい	はい	いいえ
$changeStreamSplitLargeEvent	いいえ	いいえ	いいえ	はい (8.0.2 以降)	いいえ
$collStats	なし	はい	はい	はい	いいえ
$count	はい	はい	はい	はい	はい
$currentOp	はい	はい	はい	はい	はい
$densify	いいえ	いいえ	いいえ	いいえ	いいえ
$facet	いいえ	いいえ	いいえ	はい (8.0.2 以降)	いいえ
$fill	いいえ	いいえ	いいえ	いいえ	なし
$geoNear	はい	はい	はい	はい	はい
$graphLookup	いいえ	いいえ	なし	はい	いいえ
$group	はい	はい	はい	はい	はい
$indexStats	はい	はい	はい	はい	いいえ

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$limit	はい	はい	はい	はい	はい
$listLocalSessions	いいえ	いいえ	いいえ	いいえ	いいえ
$listSessions	いいえ	いいえ	いいえ	いいえ	なし
$lookup	はい	はい	はい	はい	はい
$match	はい	はい	はい	はい	はい
$merge	-	-	いいえ	はい	いいえ
$out	はい	はい	はい	はい	いいえ
$planCacheStats	-	-	いいえ	いいえ	なし
$project	はい	はい	はい	はい	はい
$redact	はい	はい	はい	はい	はい
$replaceRoot	はい	はい	はい	はい	はい
$sample	はい	はい	はい	はい	はい
$set	-	-	いいえ	はい	いいえ
$setWindowFields	-	-	いいえ	はい (8.0.2 以降)	いいえ
$skip	はい	はい	はい	はい	はい
$sort	はい	はい	はい	はい	はい
$sortByCount	いいえ	いいえ	いいえ	はい (8.0.1 以降)	いいえ

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$unionWith	-	-	いいえ	いいえ	いいえ
$unset	-	-	なし	はい	いいえ
$unwind	はい	はい	はい	はい	はい
$replaceWith	いいえ	いいえ	なし	はい	いいえ
$vectorSearch	いいえ	いいえ	なし	はい	いいえ
$listSearchIndexes	-	-	-	はい (8.0.1 以降)	いいえ

文字列演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$concat	はい	はい	はい	はい	はい
$indexOfBytes	はい	はい	はい	はい	はい
$indexOfCP	はい	はい	はい	はい	はい
$ltrim	いいえ	はい	はい	はい	いいえ
$regexFind	-	-	はい	はい	いいえ
$regexFindAll	-	-	はい	はい	いいえ
$regexMatch	-	-	はい	はい	いいえ
$replaceAll	-	-	はい	はい	いいえ

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$replaceOne	-	-	はい	はい	いいえ
$rtrim	なし	はい	はい	はい	いいえ
$split	はい	はい	はい	はい	はい
$strcasecmp	はい	はい	はい	はい	はい
$strLenBytes	はい	はい	はい	はい	はい
$strLenCP	はい	はい	はい	はい	はい
$substr	はい	はい	はい	はい	はい
$substrBytes	はい	はい	はい	はい	はい
$substrCP	はい	はい	はい	はい	はい
$toLower	はい	はい	はい	はい	はい
$toUpper	はい	はい	はい	はい	はい
$trim	いいえ	はい	はい	はい	いいえ

システム変数

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$$CURRENT	いいえ	いいえ	いいえ	いいえ	なし
$$DESCEND	はい	はい	はい	はい	はい
$$KEEP	はい	はい	はい	はい	はい
$$PRUNE	はい	はい	はい	はい	はい

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$$REMOVE	いいえ	いいえ	いいえ	いいえ	なし
$ROOT	はい	はい	はい	はい	はい

テキスト検索演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$meta	いいえ	なし	はい	はい	いいえ
$search	いいえ	なし	はい	はい	いいえ

型変換演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$convert	いいえ	はい	はい	はい	はい
$isNumber	-	-	いいえ	はい (8.0.1 以降)	いいえ
$toBool	なし	はい	はい	はい	はい
$toDate	いいえ	はい	はい	はい	はい
$toDecimal	いいえ	はい	はい	はい	はい
$toDouble	いいえ	はい	はい	はい	はい
$toInt	いいえ	はい	はい	はい	はい
$toLong	いいえ	はい	はい	はい	はい

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$toObjectId	いいえ	はい	はい	はい	はい
$toString	いいえ	はい	はい	はい	はい
$toUUID	-	-	-	はい (8.0.1 以降)	いいえ

変数演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$let	はい	はい	はい	はい	はい
$map	はい	はい	はい	はい	はい

Trigonometry 演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$acos	-	-	-	はい (8.0.1 以降)	いいえ
$acosh	-	-	-	はい (8.0.1 以降)	いいえ
$asin	-	-	-	はい (8.0.1 以降)	いいえ
$asinh	-	-	-	はい (8.0.1 以降)	いいえ

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$atan	-	-	-	はい (8.0.1 以降)	いいえ
$atan2	-	-	-	はい (8.0.1 以降)	いいえ
$atanh	-	-	-	はい (8.0.1 以降)	いいえ
$cos	-	-	-	はい (8.0.1 以降)	いいえ
$cosh	-	-	-	はい (8.0.1 以降)	いいえ
$degreesToRadians	-	-	-	はい (8.0.1 以降)	いいえ
$radiansToDegrees	-	-	-	はい (8.0.1 以降)	いいえ
$sin	-	-	-	はい (8.0.1 以降)	いいえ
$sinh	-	-	-	はい (8.0.1 以降)	いいえ
$tan	-	-	-	はい (8.0.1 以降)	いいえ
$tanh	-	-	-	はい (8.0.1 以降)	いいえ

ビット単位の演算子 (集計)

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$bitAnd	-	-	-	はい (8.0.1 以降)	いいえ
$bitNot	-	-	-	はい (8.0.1 以降)	いいえ
$bitOr	-	-	-	はい (8.0.1 以降)	いいえ
$bitXor	-	-	-	はい (8.0.1 以降)	いいえ

タイムスタンプ演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$tsIncrement	-	-	-	はい (8.0.1 以降)	いいえ
$tsSecond	-	-	-	はい (8.0.1 以降)	いいえ

その他の演算子

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$getField	-	-	はい	はい	いいえ

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
$rand	-	-	はい (5.0.1 以降)	はい	いいえ
$sampleRate	-	-	いいえ	はい (8.0.1 以降)	いいえ
$sigmoid	-	-	-	はい (8.0.1 以降)	いいえ

データ型

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
32 ビット整数 (int)	はい	はい	はい	はい	はい
64 ビット整数 (long)	はい	はい	はい	はい	はい
配列	はい	はい	はい	はい	はい
バイナリデータ	はい	はい	はい	はい	はい
ブール値	はい	はい	はい	はい	はい
日付	可能	はい	はい	はい	はい
DBPointer	いいえ	いいえ	いいえ	いいえ	いいえ
DBRefs	いいえ	いいえ	いいえ	いいえ	いいえ
Decimal128	はい	はい	はい	はい	はい
Double	はい	はい	はい	はい	はい

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
JavaScript	いいえ	いいえ	いいえ	いいえ	いいえ
JavaScript (スコープ付き)	いいえ	いいえ	いいえ	いいえ	いいえ
MaxKey	はい	はい	はい	はい	はい
MinKey	はい	はい	はい	はい	はい
Null	はい	はい	はい	はい	はい
オブジェクト	はい	はい	はい	はい	はい
ObjectId	はい	はい	はい	はい	はい
正規表現	はい	はい	はい	はい	可能
String	可能	はい	はい	はい	はい
記号	いいえ	いいえ	いいえ	いいえ	いいえ
タイムスタンプ	はい	はい	はい	はい	はい
未定義	いいえ	いいえ	いいえ	いいえ	いいえ

インデックスとインデックスプロパティ

トピック

- インデックス
- インデックスプロパティ

インデックス

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
2dsphere	はい	はい	はい	はい	はい
2d インデックス	いいえ	いいえ	いいえ	いいえ	いいえ
複合インデックス	はい	はい	はい	はい	はい
ハッシュインデックス	いいえ	いいえ	いいえ	いいえ	いいえ
マルチキーインデックス	はい	はい	はい	はい	はい
単一フィールドインデックス	はい	はい	はい	はい	はい
テキストインデックス	いいえ	なし	はい	はい	いいえ
ワイルドカード	いいえ	いいえ	いいえ	いいえ	いいえ

インデックスプロパティ

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
背景	はい	はい	はい	はい	はい

コマンド	3.6	4.0	5.0	8.0	Elastic クラスター
大文字と小文字を区別しない	いいえ	いいえ	なし	はい	いいえ
[非表示]	いいえ	いいえ	いいえ	はい (8.0.1 以降)	いいえ
部分的	なし	なし	はい	はい	いいえ
スパース	はい	はい	はい	はい	はい
テキスト	いいえ	なし	はい	はい	いいえ
TTL	はい	はい	はい	はい	はい
Unique	はい	はい	はい	はい	はい
Vector	いいえ	なし	はい	はい	いいえ

特定の MongoDB 演算子の詳細については、以下のトピックを参照してください。

- 集約パイプライン演算子
- 地理空間
- プロジェクション演算子
- 更新演算子

集約パイプライン演算子

このセクションでは、Amazon DocumentDB でサポートされている集約パイプライン演算子に関する詳細情報を提供します。

トピック

- $$DESCEND
- $$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
- $ 下部
- $bottomN
- $bsonSize
- $bucket
- $bucketAuto

- $ceil
- $changeStream
- $changeStreamSplitLargeEvent
- $cmp
- $collStats
- $concat
- $concatArrays
- $cond
- $convert
- $cos
- $cosh
- $count (ステージ)
- $count (アキュムレータ)
- $covariancePop
- $covarianceSamp
- $currentOp
- $dateAdd
- $dateDiff
- $dateFromParts
- $dateFromString
- $dateSubtract
- $dateToParts
- $dateToString
- $dateTrunc
- $dayOfMonth
- $dayOfWeek
- $dayOfYear
- $degreesToRadians
- $denseRank

- $divide
- $documentNumber
- $eq
- $exp
- $facet
- $filter
- $first
- $firstN
- $floor
- $geoNear
- $graphLookup
- $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
- $natural
- $ne
- $not
- $objectToArray

- $or
- $out
- $percentile
- $pow
- $push
- $project
- $radiansToDegrees
- $rand
- $range
- $rank
- $redact
- $reduce
- $regexFind
- $regexFindAll
- $regexMatch
- $replaceAll
- $replaceOne
- $replaceRoot
- $replaceWith
- $reverseArray
- $round
- $rtrim
- $sample
- $search
- $second
- $set
- $setDifference
- $setEquals
- $setIntersection

- $setIsSubset
- $setUnion
- $setWindowFields
- $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 の $$DESCEND演算子は、$redactパイプラインステージ内で使用される特殊な位置配列演算子です。集約パイプラインに、ネストレベルに関係なく、現在のドキュメントに降下し、すべてのフィールドを処理するように指示します。

$redact ステージが$$DESCENDオペレータに遭遇すると、現在のドキュメント内のすべてのフィールドが表示されたままになり、パイプラインのさらに下流で処理されます。これは、ドキュメントの構造を維持しながら、条件に基づいて特定のフィールドを選択的に編集またはプルーニングする場合に便利です。

パラメータ

なし。

例 (MongoDB シェル)

この例では、 $$DESCEND 演算子で $redactステージを使用して、 codeフィールドが「Reg」と等しいドキュメントを選択的に表示します。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.patient.aggregate([
  { $redact: {
    $cond: {
      if: { $eq: ["Reg", "$code"] },
      then: "$$DESCEND",
      else: "$$PRUNE"
    }
  }}
]);
```

出力

```
{
  "_id": 2,
  "code": "Reg",
  "patient": "Jane Doe",
  "DOB": "3/27/1989",
  "Hospital": "Cherry Hill"
}
```

コードの例

$$DESCEND コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 システム変数は、現在のドキュメントまたはフィールドを変更せずに出力に含めるために、集約パイプラインの $redactステージで使用されます。

パラメータ

なし

例 (MongoDB シェル)

次の例は、Amazon DocumentDB 集約パイプライン$$KEEPでの の使用を示しています。ドキュメントは、アクセスが「パブリック」と等しい場合にのみ保持され、そうでない場合は削除されます。

サンプルドキュメントを作成する

```
db.articles.insertMany([
  { title: "Article A", access: "public", content: "Visible content" },
  { title: "Article B", access: "private", content: "Hidden content" }
]);
```

クエリの例

```
db.articles.aggregate([
  {
    $redact: {
      $cond: [
        { $eq: ["$access", "public"] },
```

```
        "$$KEEP",
        "$$PRUNE"
      ]
    }
  }
]);
```

出力

```
[
  {
    "_id" : ObjectId("..."),
    "title" : "Article A",
    "access" : "public",
    "content" : "Visible content"
  }
]
```

コードの例

$$KEEP コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

$$PRUNE システム変数は、集計パイプラインの $redactステージとともに使用され、結果からドキュメントまたは埋め込みドキュメントレベルを除外します。条件が に評価されると$$PRUNE、現在のドキュメントまたはサブドキュメントが出力から削除されます。通常、 $$DESCEND (ドキュメントを保持およびトラバースするため) または $$KEEP (ドキュメントをすべてのレベルで保持するため) で使用されます。

パラメータ

なし。$$PRUNE システム変数はパラメータなしで使用され、 で使用する必要があります$redact。

例 (MongoDB シェル)

次の例は、 $$PRUNEで を使用して 30 年以上前のユーザー$redactを結果から除外する方法を示しています。

サンプルドキュメントを作成する

```
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" } }
])
```

クエリの例

```
db.users.aggregate([
  {
    $redact: {
      $cond: {
        if: { $gt: ["$age", 30] },
        then: "$$PRUNE",
        else: "$$DESCEND"
      }
    }
  }
])
```

出力

```
[
  {
    "_id": 2,
    "name": "Saanvi Sarkar",
    "age": 28,
    "address": {
      "street": "456 Oak Rd",
      "city": "Someplace",
      "state": "NY"
    }
  }
]
```

コードの例

$$PRUNE コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $ROOT演算子は、集約パイプライン内の入力ドキュメント全体を参照するために使用されます。これにより、ネストされたすべてのフィールドと構造を含む完全なドキュメントにアクセスして操作できます。

パラメータ

なし

例 (MongoDB シェル)

この例では$ROOT、 を使用して、元のドキュメント全体を、いつ処理されたかについてのメタデータとともにキャプチャする監査ログを作成する方法を示します。

サンプルドキュメントを作成する

```
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
  }
]);
```

クエリの例

```
db.orders.aggregate([
  {
    $project: {
      processedAt: new Date(),
      originalDocument: "$$ROOT",
      summary: {
        $concat: [
          "Order ",
          "$orderId",
          " for ",
```

```
          "$customer",
          " - Total: $",
          { $toString: "$totalAmount" }
        ]
      }
    }
  }
]);
```

出力

```
[
  {
    _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'
  }
]
```

コードの例

$ROOT コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $abs演算子は、数値の絶対値を返します。集約パイプラインで使用して、数値フィールドで数学演算を実行できます。

パラメータ

- number: 絶対値が返される数値式。

例 (MongoDB シェル)

この例では、$abs演算子を使用して数値フィールドの絶対値を見つける方法を示します。

サンプルドキュメントを作成する

```
db.numbers.insertMany([
  { "_id": 1, "value": -5 },
  { "_id": 2, "value": 10 },
  { "_id": 3, "value": -3.14 },
  { "_id": 4, "value": 0 }
]);
```

クエリの例

```
db.numbers.aggregate([
  { $project: {
    "_id": 1,
    "absolute_value": { $abs: "$value" }
  }}
]);
```

出力

```
[
  { "_id": 1, "absolute_value": 5 },
  { "_id": 2, "absolute_value": 10 },
  { "_id": 3, "absolute_value": 3.14 },
  { "_id": 4, "absolute_value": 0 }
]
```

コードの例

$abs コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の$acos演算子は、値のアークコシン (逆コサイン) を返します。入力値は -1 ~ 1 である必要があります。

パラメータ

- expression: -1 から 1 までの数値に解決される式。

結果はラジアンです。学位を取得するには、 $radiansToDegreesを出力に適用します。

戻り値の型はdoubleデフォルトで です。入力が 128 ビットの 10 進数の場合、出力も 128 ビットの 10 進数です。

例 (MongoDB シェル)

次の例は、 $acos演算子を使用して数値のアークコサインを計算する方法を示しています。

サンプルドキュメントを作成する

```
db.values.insertMany([
  { "_id": 1, "value": 1 },
  { "_id": 2, "value": 0 },
  { "_id": 3, "value": -1 }
]);
```

クエリの例

```
db.values.aggregate([
  { $project: {
    "angle": { $acos: "$value" }
  }}
]);
```

出力

```
[
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 1.5707963267948966 },
  { "_id": 3, "angle": 3.141592653589793 }
```

```
]
```

コードの例

$acos 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の $acosh演算子は、値の逆双曲線コサイン (双曲線アークコサイン) を返します。入力値は 1 以上である必要があります。

パラメータ

- expression: 1 以上の数値に解決される式。

戻り値の型はdoubleデフォルトで です。入力が 128 ビットの 10 進数の場合、出力も 128 ビットの 10 進数です。

例 (MongoDB シェル)

次の例は、 $acosh演算子を使用して数値の逆双曲線コサインを計算する方法を示しています。

サンプルドキュメントを作成する

```
db.values.insertMany([
  { "_id": 1, "value": 1 },
  { "_id": 2, "value": 2 },
  { "_id": 3, "value": 10 }
]);
```

クエリの例

```
db.values.aggregate([
  { $project: {
    "result": { $acosh: "$value" }
  }}
```

```
]);
```

出力

```
[
  { "_id": 1, "result": 0 },
  { "_id": 2, "result": 1.3169578969248166 },
  { "_id": 3, "result": 2.993222846126381 }
]
```

コードの例

$acosh 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $add演算子は、数値または日付を一緒に追加するために使用されます。数値フィールドで算術演算を実行したり、日付フィールドに複数の時間単位を追加して日付算術を実行したりできます。

パラメータ

- expression1: 追加する最初の数値または日付。
- expression2: 追加する 2 番目の数値または日付。
- expression3: (オプション) 追加する追加の数値または日付。

例 (MongoDB シェル)

次の例は、 $add演算子を使用して 2 つの数値を一緒に追加する方法を示しています。

サンプルドキュメントを作成する

```
db.numbers.insertMany([
  { _id: 1, a: 5, b: 3 },
  { _id: 2, a: 10, b: 7 },
  { _id: 3, a: 2, b: 8 }
```

```
]);
```

クエリの例

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    sum: { $add: ["$a", "$b"] }
  }}
])
```

出力

```
[
  { "_id" : 1, "sum" : 8 },
  { "_id" : 2, "sum" : 17 },
  { "_id" : 3, "sum" : 10 }
]
```

この例では、 $add演算子を使用して各ドキュメントの aフィールドと bフィールドの値を追加し、結果を sumフィールドに保存します。

コードの例

$add コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

$addToSet 集計演算子は、各グループの指定された式から一意の値の配列を返します。$group ステージ内で使用して個別の値を蓄積し、重複を自動的に排除します。

パラメータ

- expression: グループ内の各ドキュメントについて評価する式。

例 (MongoDB シェル)

次の例は、 $addToSet演算子を使用して、顧客ごとに注文された一意の都市を収集する方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.orders.aggregate([
  {
    $group: {
      _id: "$customer",
      cities: { $addToSet: "$city" }
    }
  }
]);
```

出力

```
[
  { _id: 'Bob', cities: [ 'Seattle', 'Boston' ] },
  { _id: 'Alice', cities: [ 'Seattle', 'Portland' ] }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$addToSet 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメントに指定された式の一意の値の配列を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

> **Note**
>
> でウィンドウ演算子として使用する場合$setWindowFields、 $addToSetは 100 MB の中間データに制限されます。この制限を超えるオペレーションはエラーを返します。

サンプルドキュメントを作成する

```
db.visits.insertMany([
  { _id: 1, customer: "Alice", day: 1, city: "Seattle" },
  { _id: 2, customer: "Alice", day: 2, city: "Portland" },
  { _id: 3, customer: "Alice", day: 3, city: "Seattle" },
  { _id: 4, customer: "Bob", day: 1, city: "Boston" },
  { _id: 5, customer: "Bob", day: 2, city: "Boston" }
]);
```

クエリの例

次の例では、ドキュメントを でパーティション化しcustomer、各パーティションを でソートしてday、パーティションの先頭から現在のドキュメントまでの一意のcity値のセットを返します。

```
db.visits.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$customer",
      sortBy: { day: 1 },
      output: {
        citiesSoFar: {
          $addToSet: "$city",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "customer": "Alice", "day": 1, "city": "Seattle", "citiesSoFar":
 [ "Seattle" ] },
  { "_id": 2, "customer": "Alice", "day": 2, "city": "Portland", "citiesSoFar":
 [ "Portland", "Seattle" ] },
  { "_id": 3, "customer": "Alice", "day": 3, "city": "Seattle", "citiesSoFar":
 [ "Portland", "Seattle" ] },
  { "_id": 4, "customer": "Bob", "day": 1, "city": "Boston", "citiesSoFar":
 [ "Boston" ] },
  { "_id": 5, "customer": "Bob", "day": 2, "city": "Boston", "citiesSoFar":
 [ "Boston" ] }
```

```
]
```

各ドキュメントはcitiesSoFar、最初から現在のドキュメントまでのパーティション内の一意のcity値のセットである で拡張されます。アキュムレータ形式と同様に、返される配列の値の順序は保証されません。

コードの例

$addToSet 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group) とウィンドウ演算子の使用 (の場合) の両方を示しています$setWindowFields。

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');

  // Accumulator usage: collect unique values per group
  const orders = db.collection('orders');
  await 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 }
  ]);
  const accumulatorResult = await orders.aggregate([
    {
      $group: {
        _id: "$customer",
        cities: { $addToSet: "$city" }
      }
    }
  ]).toArray();
  console.log('Accumulator result:', accumulatorResult);

  // Window operator usage: running set of unique values within each partition
  const visits = db.collection('visits');
```

```
  await visits.insertMany([
    { _id: 1, customer: "Alice", day: 1, city: "Seattle" },
    { _id: 2, customer: "Alice", day: 2, city: "Portland" },
    { _id: 3, customer: "Alice", day: 3, city: "Seattle" },
    { _id: 4, customer: "Bob", day: 1, city: "Boston" },
    { _id: 5, customer: "Bob", day: 2, city: "Boston" }
  ]);
  const windowResult = await visits.aggregate([
    {
      $setWindowFields: {
        partitionBy: "$customer",
        sortBy: { day: 1 },
        output: {
          citiesSoFar: {
            $addToSet: "$city",
            window: { documents: ["unbounded", "current"] }
          }
        }
      }
    }
  ]).toArray();
  console.log('Window result:', windowResult);

  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']

    # Accumulator usage: collect unique values per group
    orders = db['orders']
    orders.insert_many([
        { '_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 }
    ])
    accumulator_result = list(orders.aggregate([
        {
            '$group': {
                '_id': '$customer',
                'cities': { '$addToSet': '$city' }
            }
        }
    ]))
    print('Accumulator result:', accumulator_result)

    # Window operator usage: running set of unique values within each partition
    visits = db['visits']
    visits.insert_many([
        { '_id': 1, 'customer': 'Alice', 'day': 1, 'city': 'Seattle' },
        { '_id': 2, 'customer': 'Alice', 'day': 2, 'city': 'Portland' },
        { '_id': 3, 'customer': 'Alice', 'day': 3, 'city': 'Seattle' },
        { '_id': 4, 'customer': 'Bob', 'day': 1, 'city': 'Boston' },
        { '_id': 5, 'customer': 'Bob', 'day': 2, 'city': 'Boston' }
    ])
    window_result = list(visits.aggregate([
        {
            '$setWindowFields': {
                'partitionBy': '$customer',
                'sortBy': { 'day': 1 },
                'output': {
                    'citiesSoFar': {
                        '$addToSet': '$city',
                        'window': { 'documents': ['unbounded', 'current'] }
                    }
                }
            }
        }
    ]))
    print('Window result:', window_result)

    client.close()

example()
```

$addFields

Amazon DocumentDB 集約パイプラインの $addFieldsステージでは、新しい計算フィールドをドキュメントに追加できます。これは、派生データまたは変換されたデータをドキュメントに追加するのに役立ちます。

パラメータ

- newField: 追加する新しいフィールドの名前。
- expression: 新しいフィールドの値に解決される式。

例 (MongoDB シェル)

次の例は、 $addFieldsを使用して、 Inventory.OnHand および フィールドに基づいてインベントリの合計TotalInventoryを計算する新しいInventory.OrderQntyフィールドを追加する方法を示しています。

サンプルドキュメントを作成する

```
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
  }
]);
```

クエリの例

```
db.example.aggregate([
  {
    $addFields: {
      TotalInventory: { $add: ["$Inventory.OnHand", "$Inventory.OrderQnty"] }
    }
  }
])
```

出力

```
[
  {
    "_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
  }
]
```

コードの例

$addFields コマンドを使用するためのコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能

$allElementsTrue 演算子は、配列内のすべての要素が true 値に評価されるかどうかを確認するために使用されます。

パラメータ

- expression: 配列に評価される式。

例 (MongoDB シェル)

次の例は、 を使用して、配列内のすべての要素が true であるかどうかをチェック$allElementsTrueする方法を示しています。

サンプルドキュメントを作成する

```
db.collection.insert([
  { "name": "John", "scores": [100, 90, 80] },
  { "name": "Jane", "scores": [80, 85, 0] },
  { "name": "Bob", "scores": [90, 95, null] }
])
```

クエリの例

```
db.collection.find({
  "scores": { "$allElementsTrue": [{ "$gt": 0 }] }
})
```

出力

```
[
  { "_id" : ObjectId("..."), "name" : "John", "scores" : [ 100, 90, 80 ] },
  { "_id" : ObjectId("..."), "name" : "Bob", "scores" : [ 90, 95, null ] }
]
```

この例では、クエリはscores配列内のすべての要素が 0 より大きいかどうかを確認します。scores 配列に 0 が含まれているため、 のドキュメント"name": "Jane"は除外されます。0 は不正な値です。

コードの例

$allElementsTrue コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 集計演算子は複数の式を評価し、すべての式が と評価されたtrue場合にのみ を返しますtrue。いずれかの式が の場合false、 を返しますfalse。

パラメータ

- expressions: 評価する式の配列。

例 (MongoDB シェル)

次の例では、 $and演算子を使用して、製品が複数の基準を満たしているかどうかを確認します。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.products.aggregate([
  {
    $project: {
      name: 1,
      price: 1,
      inStock: 1,
      affordable: {
        $and: [
          { $lt: ["$price", 100] },
          { $eq: ["$inStock", true] }
        ]
      }
```

```
    }
  }
]);
```

出力

```
[
  { _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 }
]
```

コードの例

$and 集計演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能

$anyElementTrue 演算子は、配列内のいずれかの要素が true かどうかを判断するために使用されます。

パラメータ

- field: 評価する配列フィールド。

例 (MongoDB シェル)

次の例は、配列内の要素が true であるかどうかをチェック$anyElementTrueするための の使用を示しています。

サンプルドキュメントを作成する

```
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] }
])
```

クエリの例

```
db.grades.aggregate([
  {
    $project: {
      student: 1,
      isAnyTrue: { $anyElementTrue: ["$scores"] },
      _id: 0
    }
  }
])
```

出力

```
[
  { student: 'Tim', isAnyTrue: false },
  { student: 'Bob', isAnyTrue: false },
  { student: 'Ivy', isAnyTrue: true }
]
```

コードの例

$anyElementTrue コマンドを使用するためのコード例を表示するには、使用する言語のタブを選択します。

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 の $arrayElemAt演算子を使用すると、インデックス位置によって配列から要素を取得できます。これは、ドキュメントの配列フィールド内の特定の要素にアクセスする必要がある場合に特に便利です。

パラメータ

- array: 要素を取得する入力配列。
- index: 取得する要素のゼロベースのインデックス位置。この値は負以外の整数である必要があります。

例 (MongoDB シェル)

この例では、 $arrayElemAt演算子を使用してmilesコレクション内のflight_miles配列から特定の要素を取得する方法を示します。

サンプルドキュメントを作成する

```
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 ]}
]);
```

クエリの例

```
db.miles.aggregate([
  { $project: {
    "_id": 1,
    "first_mile": { $arrayElemAt: [ "$flight_miles", 0 ] },
    "last_mile": { $arrayElemAt: [ "$flight_miles", -1 ] }
  }}
]);
```

出力

```
{ "_id" : 1, "first_mile" : 1205, "last_mile" : 880 }
{ "_id" : 2, "first_mile" : 1205, "last_mile" : 2780 }
{ "_id" : 3, "first_mile" : 1205, "last_mile" : 880 }
```

この例では、 $arrayElemAt演算子を使用して、各ドキュメントのflight_miles配列の最初と最後の要素を取得します。

コードの例

$arrayElemAt コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $arrayToObject演算子は、 $objectToArray演算子の逆です。キーと値のペアドキュメントの配列を取得し、1 つのドキュメントに変換します。これは、キーと値のペアの配列をオブジェクトまたはドキュメント構造に変換する必要がある場合に特に便利です。

パラメータ

- array expression: 配列に解決される式。配列要素は、(キー) と k (値) の 2 v つのフィールドを持つドキュメントである必要があります。

例 (MongoDB シェル)

次の例は、$arrayToObjectを使用してキーと値のペアの配列をドキュメントに変換する方法を示しています。

サンプルドキュメントを作成する

```
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 } }
]);
```

クエリの例

```
db.videos.aggregate([
  { $project: { name: 1, videos: { $objectToArray: "$inventory" } } },
  { $project: { name: 1, inventory: { $arrayToObject: "$videos" } } }
]);
```

出力

```
{ "_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 } }
```

この例では、$objectToArray演算子を使用してinventoryオブジェクトをキーと値のペアの配列に変換します。次に、$arrayToObject演算子を使用して配列をドキュメントに変換し、元のオブジェクト構造を復元します。

コードの例

$arrayToObject コマンドを使用するためのコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の $asin演算子は、値のアークサイン (逆サイン) を返します。入力値は -1 ～ 1 である必要があります。

パラメータ

- expression: -1 から 1 までの数値に解決される式。

結果はラジアンです。学位を取得するには、 $radiansToDegreesを出力に適用します。

戻り値の型はdoubleデフォルトで です。入力が 128 ビットの 10 進数の場合、出力も 128 ビットの 10 進数です。

例 (MongoDB シェル)

次の例は、 $asin演算子を使用して数値のアークサインを計算する方法を示しています。

サンプルドキュメントを作成する

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 0.5 },
  { "_id": 3, "value": 1 }
]);
```

クエリの例

```
db.values.aggregate([
  { $project: {
    "angle": { $asin: "$value" }
  }}
]);
```

出力

```
[
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 0.5235987755982989 },
  { "_id": 3, "angle": 1.5707963267948966 }
]
```

コードの例

$asin 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の $asinh演算子は、値の逆双曲線サインを返します。集約パイプラインでこれを使用して、数値フィールドに対して双曲線三角計算を実行します。

パラメータ

- expression: 数値に解決される式。

結果はラジアンです。学位を取得するには、 $radiansToDegreesを出力に適用します。

戻り値の型はdoubleデフォルトで です。入力が 128 ビットの 10 進数の場合、出力も 128 ビットの 10 進数です。

例 (MongoDB シェル)

次の例は、 $asinh演算子を使用して数値の逆双曲線サインを計算する方法を示しています。

サンプルドキュメントを作成する

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 1 },
  { "_id": 3, "value": -1 }
```

```
]);
```

クエリの例

```
db.values.aggregate([
  { $project: {
    "result": { $asinh: "$value" }
  }}
]);
```

出力

```
[
  { "_id": 1, "result": 0 },
  { "_id": 2, "result": 0.881373587019543 },
  { "_id": 3, "result": -0.881373587019543 }
]
```

コードの例

$asinh 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の $atan演算子は、値の逆タンジェント (arc タンジェント) を返します。

入力式は数値に解決する必要があります。

結果はラジアンです。学位を取得するには、 $radiansToDegreesを出力に適用します。

戻り値の型はdoubleデフォルトで です。入力が 128 ビットの 10 進数の場合、出力も 128 ビットの 10 進数です。

パラメータ

- expression: 数値に解決される式。

動作

null と NaN

例	結果
`{ $atan: NaN }`	`NaN`
`{ $atan: null }`	`null`

入力が nullであるか、参照フィールドが欠落している場合、結果は ですnull。の入力により、NaNが生成されますNaN。

例 (MongoDB シェル)

次の例は、 $atan演算子を使用して値の配列を計算する方法を示しています。

サンプルドキュメントを作成する

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 1 },
  { "_id": 3, "value": -1 }
]);
```

クエリの例

```
db.values.aggregate([
  { $project: {
    "angle": { $atan: "$value" }
  }}
]);
```

出力

```
[
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 0.7853981633974483 },
  { "_id": 3, "angle": -0.7853981633974483 }
```

```
]
```

コードの例

$atan 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の $atan2演算子は、 の逆接線 (arc 接線) を返します。ここでy / x、 y と xはそれぞれ式に渡される最初と 2 番目の値です。

どちらの式も数値に解決する必要があります。演算子は、両方の引数の符号を使用して正しい四分円を決定します。

結果はラジアンです。学位を取得するには、 $radiansToDegreesを出力に適用します。

戻り値の型はdoubleデフォルトで です。入力が 128 ビットの小数の場合、出力も 128 ビットの小数です。

パラメータ

- expression 1 (y): y 座標を表す数値に解決される式。
- expression 2 (x): x 座標を表す数値に解決される式。

動作

null と NaN

いずれかの引数が の場合 null (または欠落しているフィールドを参照する場合) 、結果は ですnull。いずれかの引数が の場合NaN、結果は ですNaN。1 つの引数が nullで、もう 1 つの引数が の場合NaN、 null が優先されます。

例	結果
{ $atan2: [NaN, <value>] }	NaN

例	結果
`{ $atan2: [ <value>, NaN ] }`	`NaN`
`{ $atan2: [ null, <value> ] }`	`null`
`{ $atan2: [ <value>, null ] }`	`null`
`{ $atan2: [ NaN, null ] }`	`null`
`{ $atan2: [ null, NaN ] }`	`null`

例 (MongoDB シェル)

次の例は、 $atan2演算子を使用して、ポイントのセットの y/x のアークタンジェントを計算する方法を示しています。

サンプルドキュメントを作成する

```
db.points.insertMany([
  { "_id": 1, "y": 1, "x": 1 },
  { "_id": 2, "y": 1, "x": 0 },
  { "_id": 3, "y": -1, "x": -1 }
]);
```

クエリの例

```
db.points.aggregate([
  { $project: {
    "angle": { $atan2: ["$y", "$x"] }
  }}
]);
```

出力

```
[
  { "_id": 1, "angle": 0.7853981633974483 },
  { "_id": 2, "angle": 1.5707963267948966 },
  { "_id": 3, "angle": -2.356194490192345 }
]
```

コードの例

$atan2 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の $atanh演算子は、値の逆双曲線接線 (双曲線アーク接線) を返します。

入力式は、 -1 1 (を含む) の範囲の数値に解決する必要があります。この範囲外の値はエラーを引き起こします。

戻り値の型はdoubleデフォルトで です。入力が 128 ビットの 10 進数の場合、出力も 128 ビットの 10 進数です。

パラメータ

- expression: -1 ~ の数値に解決される式1。

動作

null、NaN、および +/- Infinity

例	結果
{ $atanh: NaN }	NaN
{ $atanh: null }	null
{ $atanh: 1 }	Infinity
{ $atanh: -1 }	-Infinity
{ $atanh: Infinity }、または { $atanh: -Infinity }	エラーをスローします。

入力が nullであるか、参照フィールドが欠落している場合、結果は ですnull。の入力により、 NaNが生成されますNaN。境界値 1と は、-Infinityそれぞれ Infinity と -1を生成します。[-1, 1] 範囲外の値 (を含む±Infinity) があると、エラーが発生します。

例 (MongoDB シェル)

次の例は、 $atanh演算子を使用して値の逆双曲線タンジェントを計算する方法を示しています。

サンプルドキュメントを作成する

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 0.5 },
  { "_id": 3, "value": -0.5 }
]);
```

クエリの例

```
db.values.aggregate([
  { $project: {
    "result": { $atanh: "$value" }
  }}
]);
```

出力

```
[
  { "_id": 1, "result": 0 },
  { "_id": 2, "result": 0.5493061443340548 },
  { "_id": 3, "result": -0.5493061443340548 }
]
```

コードの例

$atanh 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$avg集計演算子は、ステージに入力されるドキュメント全体で指定された式の平均値を計算します。この演算子は、一連のドキュメントにわたる数値フィールドまたは式の平均を計算するのに役立ちます。

パラメータ

- `expression`: 平均の計算に使用する式。これは、フィールドパス (例: `"$field"`) または式 (例: `{ $multiply: ["$field1", "$field2"] }`) です。

例 (MongoDB シェル)

次の例は、 $avg演算子を使用して、一連の学生ドキュメントの平均スコアを計算する方法を示しています。

サンプルドキュメントを作成する

```
db.students.insertMany([
  { name: "John", score: 85 },
  { name: "Jane", score: 92 },
  { name: "Bob", score: 78 },
  { name: "Alice", score: 90 }
]);
```

クエリの例

```
db.students.aggregate([
  { $group: {
    _id: null,
    avgScore: { $avg: "$score" }
  }}
]);
```

出力

```
[
  {
    "_id": null,
    "avgScore": 86.25
```

```
  }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$avg 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメントに対して指定された式の平均を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

サンプルドキュメントを作成する

```
db.readings.insertMany([
  { _id: 1, city: "Denver", day: 1, temp: 70 },
  { _id: 2, city: "Denver", day: 2, temp: 80 },
  { _id: 3, city: "Seattle", day: 1, temp: 60 },
  { _id: 4, city: "Seattle", day: 2, temp: 64 },
  { _id: 5, city: "Seattle", day: 3, temp: 68 }
]);
```

クエリの例

次の例では、ドキュメントを でパーティション化しcity、各パーティションを でソートしてday、パーティションの開始tempから現在のドキュメントまでの の実行平均を返します。

```
db.readings.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$city",
      sortBy: { day: 1 },
      output: {
        runningAvg: {
          $avg: "$temp",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "city": "Denver", "day": 1, "temp": 70, "runningAvg": 70 },
  { "_id": 2, "city": "Denver", "day": 2, "temp": 80, "runningAvg": 75 },
  { "_id": 3, "city": "Seattle", "day": 1, "temp": 60, "runningAvg": 60 },
  { "_id": 4, "city": "Seattle", "day": 2, "temp": 64, "runningAvg": 62 },
  { "_id": 5, "city": "Seattle", "day": 3, "temp": 68, "runningAvg": 64 }
]
```

各ドキュメントはrunningAvg、現在のドキュメントまでのパーティションtemp内の の平均であるで拡張されます。

コードの例

$avg 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group) とウィンドウ演算子の使用 (の場合) の両方を示しています$setWindowFields。

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');

  // Accumulator usage: average across grouped documents
  const students = db.collection('students');
  await students.insertMany([
    { name: "John", score: 85 },
    { name: "Jane", score: 92 },
    { name: "Bob", score: 78 },
    { name: "Alice", score: 90 }
  ]);
  const accumulatorResult = await students.aggregate([
    { $group: {
      _id: null,
      avgScore: { $avg: '$score' }
    }}
  ]).toArray();
```

```
  console.log('Accumulator result:', accumulatorResult);

  // Window operator usage: running average within each partition
  const readings = db.collection('readings');
  await readings.insertMany([
    { _id: 1, city: "Denver", day: 1, temp: 70 },
    { _id: 2, city: "Denver", day: 2, temp: 80 },
    { _id: 3, city: "Seattle", day: 1, temp: 60 },
    { _id: 4, city: "Seattle", day: 2, temp: 64 },
    { _id: 5, city: "Seattle", day: 3, temp: 68 }
  ]);
  const windowResult = await readings.aggregate([
    {
      $setWindowFields: {
        partitionBy: "$city",
        sortBy: { day: 1 },
        output: {
          runningAvg: {
            $avg: "$temp",
            window: { documents: ["unbounded", "current"] }
          }
        }
      }
    }
  ]).toArray();
  console.log('Window result:', windowResult);

  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

    # Accumulator usage: average across grouped documents
```

```
    students = db.students
    students.insert_many([
        { 'name': 'John', 'score': 85 },
        { 'name': 'Jane', 'score': 92 },
        { 'name': 'Bob', 'score': 78 },
        { 'name': 'Alice', 'score': 90 }
    ])
    accumulator_result = list(students.aggregate([
        { '$group': {
            '_id': None,
            'avgScore': { '$avg': '$score' }
        }}
    ]))
    print('Accumulator result:', accumulator_result)

    # Window operator usage: running average within each partition
    readings = db.readings
    readings.insert_many([
        { '_id': 1, 'city': 'Denver', 'day': 1, 'temp': 70 },
        { '_id': 2, 'city': 'Denver', 'day': 2, 'temp': 80 },
        { '_id': 3, 'city': 'Seattle', 'day': 1, 'temp': 60 },
        { '_id': 4, 'city': 'Seattle', 'day': 2, 'temp': 64 },
        { '_id': 5, 'city': 'Seattle', 'day': 3, 'temp': 68 }
    ])
    window_result = list(readings.aggregate([
        {
            '$setWindowFields': {
                'partitionBy': '$city',
                'sortBy': { 'day': 1 },
                'output': {
                    'runningAvg': {
                        '$avg': '$temp',
                        'window': { 'documents': ['unbounded', 'current'] }
                    }
                }
            }
        }
    ]))
    print('Window result:', window_result)

    client.close()

calculate_avg_score()
```

$binarySize

バージョン 8.0.1 から新規。

Amazon DocumentDB の $binarySize演算子は、指定された文字列またはバイナリデータ値のバイト単位でサイズを返します。文字列値の場合、これは文字数ではなく、UTF-8 でエンコードされたバイト数です。マルチバイトの UTF-8 文字 (アクセント付き文字や CJK 文字など) は、文字列内の文字数より大きいバイト数を生成します。

パラメータ

- expression: 文字列またはバイナリデータ値に解決される式。

例 (MongoDB シェル)

次の例は、 $binarySize演算子を使用して文字列フィールドのバイトサイズを返す方法を示しています。

サンプルドキュメントを作成する

```
db.docs.insertMany([
  {_id: 1, content: "hello"},
  {_id: 2, content: "Amazon DocumentDB"},
  {_id: 3, content: ""}
]);
```

クエリの例

```
db.docs.aggregate([
  { $project: { size: { $binarySize: "$content" } } }
]);
```

出力

```
[
  {_id: 1, size: 5},
  {_id: 2, size: 17},
  {_id: 3, size: 0}
]
```

コードの例

$binarySize 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の $bitAnd演算子は、整数または長い値に対してビット単位の AND 演算を実行します。

パラメータ

- expressions: 整数または長整数に解決できる 2 つ以上の式の配列。

例 (MongoDB シェル)

次の例は、 $bitAnd演算子を使用して 2 つのフィールドでビット単位の AND を実行する方法を示しています。

サンプルドキュメントを作成する

```
db.flags.insertMany([
  {_id: 1, a: 13, b: 10},
  {_id: 2, a: 7, b: 5},
  {_id: 3, a: 15, b: 9}
]);
```

クエリの例

```
db.flags.aggregate([
  { $project: { result: { $bitAnd: ["$a", "$b"] } } }
]);
```

出力

```
[
  {_id: 1, result: 8},
  {_id: 2, result: 5},
  {_id: 3, result: 9}
]
```

バイナリ: 13 (1101) AND 10 (1010) = 8 (1000)、7 (0111) AND 5 (0101) = 5 (0101)、15 (1111) AND 9 (1001) = 9 (1001)。

コードの例

$bitAnd 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の $bitNot演算子は、整数または長い値に対してビット単位の NOT オペレーションを実行し、ビット単位の補数を返します。指定された整数または長い n の場合、結果は -(n+1) です。

パラメータ

- expression: 整数または長に解決される式。ビット単位の補数を返します。指定された整数または長い n の -(n+1) です。

例 (MongoDB シェル)

次の例は、 $bitNot演算子を使用して整数値のビット単位の補数を計算する方法を示しています。

サンプルドキュメントを作成する

```
db.numbers.insertMany([
  {_id: 1, value: 0},
  {_id: 2, value: 5},
  {_id: 3, value: -3}
]);
```

クエリの例

```
db.numbers.aggregate([
  { $project: { result: { $bitNot: "$value" } } }
]);
```

出力

```
[
```

```
  {_id: 1, result: -1},
  {_id: 2, result: -6},
  {_id: 3, result: 2}
]
```

バイナリの場合: NOT 0 = -1、NOT 5 = -6、NOT -3 = 2。

コードの例

$bitNot 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の $bitOr演算子は、整数または長い値に対してビット単位の OR オペレーションを実行します。

パラメータ

- expressions: 整数または長整数に解決できる 2 つ以上の式の配列。

例 (MongoDB シェル)

次の例は、 $bitOr演算子を使用して 2 つのフィールドでビット単位の OR を実行する方法を示しています。

サンプルドキュメントを作成する

```
db.flags.insertMany([
  {_id: 1, a: 13, b: 10},
  {_id: 2, a: 7, b: 5},
  {_id: 3, a: 15, b: 9}
]);
```

クエリの例

```
db.flags.aggregate([
  { $project: { result: { $bitOr: ["$a", "$b"] } } }
]);
```

出力

```
[
  {_id: 1, result: 15},
  {_id: 2, result: 7},
  {_id: 3, result: 15}
]
```

バイナリ: 13 (1101) OR 10 (1010) = 15 (1111)、7 (0111) OR 5 (0101) = 7 (0111)、15 (1111) OR 9 (1001) = 15 (1111)。

コードの例

$bitOr 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の $bitXor演算子は、整数または長い値に対してビット単位の XOR オペレーションを実行します。

パラメータ

- expressions: 整数または長整数に解決できる 2 つ以上の式の配列。

例 (MongoDB シェル)

次の例は、 $bitXor演算子を使用して 2 つのフィールドでビット単位の XOR を実行する方法を示しています。

サンプルドキュメントを作成する

```
db.flags.insertMany([
  {_id: 1, a: 13, b: 10},
  {_id: 2, a: 7, b: 5},
  {_id: 3, a: 15, b: 9}
]);
```

クエリの例

```
db.flags.aggregate([
  { $project: { result: { $bitXor: ["$a", "$b"] } } }
]);
```

出力

```
[
  {_id: 1, result: 7},
  {_id: 2, result: 2},
  {_id: 3, result: 6}
]
```

バイナリ: 13 (1101) XOR 10 (1010) = 7 (0111)、7 (0111) XOR 5 (0101) = 2 (0010)、15 (1111) XOR 9 (1001) = 6 (0110)。

コードの例

$bitXor 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

$ 下部

バージョン 8.0.1 から新規。

$group ステージの $bottom アキュムレータを使用して、指定されたソート順序に従ってグループごとに最低ランクのドキュメントを返します。

パラメータ

- sortBy: ソート順序を指定するドキュメント。昇順1の場合は 、降順-1の場合は を使用します。
- output: 下部のドキュメントから返すフィールドを指定する式。

例 (MongoDB シェル)

次の例は、 $bottom アキュムレータを使用して、売上コレクション内の項目あたりのボトムセール (最低数量) を検索する方法を示しています。

サンプルドキュメントを作成する

```
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 }
])
```

クエリの例

```
db.sales.aggregate([
  { $group: { _id: "$item", bottomSale: { $bottom: { sortBy: { quantity: -1 }, output:
 { quantity: "$quantity", price: "$price" } } } } }
])
```

出力

```
[
  { "_id": "xyz", "bottomSale": { "quantity": 7, "price": 6 } },
  { "_id": "abc", "bottomSale": { "quantity": 5, "price": 8 } }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$bottom 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメント間で (オペレータ自身の に従ってsortBy) 下位ランクのドキュメントoutputから を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

> **Note**
>
> でウィンドウ演算子として使用する場合$setWindowFields、 $bottomは 100 MB の中間データに制限されます。この制限を超えるオペレーションはエラーを返します。

サンプルドキュメントを作成する

```
db.matchScores.insertMany([
  { _id: 1, player: "Alice", round: 1, score: 20 },
  { _id: 2, player: "Alice", round: 2, score: 35 },
  { _id: 3, player: "Alice", round: 3, score: 28 },
  { _id: 4, player: "Bob", round: 1, score: 15 },
  { _id: 5, player: "Bob", round: 2, score: 40 }
]);
```

クエリの例

次の例ではplayer、ドキュメントを でパーティション化し、各パーティションを でソートしてround、パーティションの先頭から現在のドキュメントまでの最低スコアのラウンドを返します。

```
db.matchScores.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$player",
      sortBy: { round: 1 },
      output: {
        worstRoundSoFar: {
          $bottom: { sortBy: { score: -1 }, output: { round: "$round", score:
 "$score" } },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "player": "Alice", "round": 1, "score": 20, "worstRoundSoFar": { "round":
 1, "score": 20 } },
  { "_id": 2, "player": "Alice", "round": 2, "score": 35, "worstRoundSoFar": { "round":
 1, "score": 20 } },
  { "_id": 3, "player": "Alice", "round": 3, "score": 28, "worstRoundSoFar": { "round":
 1, "score": 20 } },
  { "_id": 4, "player": "Bob", "round": 1, "score": 15, "worstRoundSoFar": { "round":
 1, "score": 15 } },
  { "_id": 5, "player": "Bob", "round": 2, "score": 40, "worstRoundSoFar": { "round":
 1, "score": 15 } }
]
```

各ドキュメントはworstRoundSoFar、現在のドキュメントまで、パーティション内の最も低いスコアのラウンドである で拡張されます。

コードの例

$bottom 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group) とウィンドウ演算子の使用 (の場合) の両方を示しています$setWindowFields。

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: bottom document per group
    const sales = db.collection('sales');
    await 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 }
    ]);
    const accumulatorResult = await sales.aggregate([
      { $group: { _id: "$item", bottomSale: { $bottom: { sortBy: { quantity: -1 },
 output: { quantity: "$quantity", price: "$price" } } } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Window operator usage: worst round so far within each partition
    const matchScores = db.collection('matchScores');
    await matchScores.insertMany([
      { _id: 1, player: "Alice", round: 1, score: 20 },
      { _id: 2, player: "Alice", round: 2, score: 35 },
      { _id: 3, player: "Alice", round: 3, score: 28 },
      { _id: 4, player: "Bob", round: 1, score: 15 },
      { _id: 5, player: "Bob", round: 2, score: 40 }
    ]);
    const windowResult = await matchScores.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$player",
          sortBy: { round: 1 },
          output: {
```

```
            worstRoundSoFar: {
              $bottom: { sortBy: { score: -1 }, output: { round: "$round", score:
 "$score" } },
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } 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']

        # Accumulator usage: bottom document per group
        sales = db['sales']
        sales.insert_many([
            { 'item': 'abc', 'quantity': 10, 'price': 5 },
            { 'item': 'abc', 'quantity': 5, 'price': 8 },
            { 'item': 'xyz', 'quantity': 15, 'price': 3 },
            { 'item': 'xyz', 'quantity': 7, 'price': 6 }
        ])
        accumulator_result = list(sales.aggregate([
```

```
            { '$group': { '_id': '$item', 'bottomSale': { '$bottom':
 { 'sortBy': { 'quantity': -1 }, 'output': { 'quantity': '$quantity', 'price':
 '$price' } } } } }
        ]))
        pprint(accumulator_result)

        # Window operator usage: worst round so far within each partition
        match_scores = db['matchScores']
        match_scores.insert_many([
            { '_id': 1, 'player': 'Alice', 'round': 1, 'score': 20 },
            { '_id': 2, 'player': 'Alice', 'round': 2, 'score': 35 },
            { '_id': 3, 'player': 'Alice', 'round': 3, 'score': 28 },
            { '_id': 4, 'player': 'Bob', 'round': 1, 'score': 15 },
            { '_id': 5, 'player': 'Bob', 'round': 2, 'score': 40 }
        ])
        window_result = list(match_scores.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$player',
                    'sortBy': { 'round': 1 },
                    'output': {
                        'worstRoundSoFar': {
                            '$bottom': { 'sortBy': { 'score': -1 }, 'output':
 { 'round': '$round', 'score': '$score' } },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        pprint(window_result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$bottomN

バージョン 8.0.1 から新規。

$group ステージの$bottomNアキュムレータを使用して、指定されたソート順序に従ってグループ内の下位 N 要素を返します。グループに含まれる要素が N 未満の場合、 はグループ内のすべての要素$bottomNを返します。

パラメータ

- n: 正の整数、またはグループごとに返す下位結果の数を指定する 1 に解決される式。
- sortBy: ソート順序を指定するドキュメント。昇順-1または降順1に使用します。
- output: 下部の各 N ドキュメントから返すフィールドを指定する式。

例 (MongoDB シェル)

次の例は、 $bottomN アキュムレータを使用して、売上コレクション内の項目あたりの売上の下位 2 つ (最低数量) を見つける方法を示しています。

サンプルドキュメントを作成する

```
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 }
])
```

クエリの例

```
db.sales.aggregate([
  { $group: { _id: "$item", bottomTwoSales: { $bottomN: { n: 2, sortBy: { quantity:
 -1 }, output: { quantity: "$quantity", price: "$price" } } } } }
])
```

出力

```
[
```

```
  { "_id": "xyz", "bottomTwoSales": [{ "quantity": 9, "price": 6 }, { "quantity": 3,
 "price": 12 }] },
  { "_id": "abc", "bottomTwoSales": [{ "quantity": 7, "price": 8 }, { "quantity": 5,
 "price": 10 }] }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$bottomN 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメント間で、(オペレータ自身の に従ってsortBy) 下部の N ドキュメントoutputから の配列を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

> **Note**
>
> でウィンドウ演算子として使用する場合$setWindowFields、 $bottomNは 100 MB の中間データに制限されます。この制限を超えるオペレーションはエラーを返します。

サンプルドキュメントを作成する

```
db.matchScores.insertMany([
  { _id: 1, player: "Alice", round: 1, score: 20 },
  { _id: 2, player: "Alice", round: 2, score: 35 },
  { _id: 3, player: "Alice", round: 3, score: 28 },
  { _id: 4, player: "Bob", round: 1, score: 15 },
  { _id: 5, player: "Bob", round: 2, score: 40 }
]);
```

クエリの例

次の例ではplayer、ドキュメントを でパーティション化し、各パーティションを でソートしてround、パーティションの先頭から現在のドキュメントまでの 2 つの最も低いスコアのラウンドを返します。

```
db.matchScores.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$player",
```

```
      sortBy: { round: 1 },
      output: {
        bottomTwoRoundsSoFar: {
          $bottomN: { n: 2, sortBy: { score: -1 }, output: { round: "$round", score:
 "$score" } },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "player": "Alice", "round": 1, "score": 20, "bottomTwoRoundsSoFar":
 [{ "round": 1, "score": 20 }] },
  { "_id": 2, "player": "Alice", "round": 2, "score": 35, "bottomTwoRoundsSoFar":
 [{ "round": 2, "score": 35 }, { "round": 1, "score": 20 }] },
  { "_id": 3, "player": "Alice", "round": 3, "score": 28, "bottomTwoRoundsSoFar":
 [{ "round": 3, "score": 28 }, { "round": 1, "score": 20 }] },
  { "_id": 4, "player": "Bob", "round": 1, "score": 15, "bottomTwoRoundsSoFar":
 [{ "round": 1, "score": 15 }] },
  { "_id": 5, "player": "Bob", "round": 2, "score": 40, "bottomTwoRoundsSoFar":
 [{ "round": 2, "score": 40 }, { "round": 1, "score": 15 }] }
]
```

各ドキュメントはbottomTwoRoundsSoFar、現在のドキュメントまでパーティション内の 2 つの最も低いスコアのラウンドである で拡張されます。ウィンドウに含まれるドキュメントが 2 つ未満の場合、それらはすべて返されます。

コードの例

$bottomN 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group) とウィンドウ演算子の使用 (の場合) の両方を示しています$setWindowFields。

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: bottom N documents per group
    const sales = db.collection('sales');
    await 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 }
    ]);
    const accumulatorResult = await sales.aggregate([
      { $group: { _id: "$item", bottomTwoSales: { $bottomN: { n: 2, sortBy:
 { quantity: -1 }, output: { quantity: "$quantity", price: "$price" } } } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Window operator usage: bottom two rounds so far within each partition
    const matchScores = db.collection('matchScores');
    await matchScores.insertMany([
      { _id: 1, player: "Alice", round: 1, score: 20 },
      { _id: 2, player: "Alice", round: 2, score: 35 },
      { _id: 3, player: "Alice", round: 3, score: 28 },
      { _id: 4, player: "Bob", round: 1, score: 15 },
      { _id: 5, player: "Bob", round: 2, score: 40 }
    ]);
    const windowResult = await matchScores.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$player",
          sortBy: { round: 1 },
          output: {
            bottomTwoRoundsSoFar: {
              $bottomN: { n: 2, sortBy: { score: -1 }, output: { round: "$round",
 score: "$score" } },
```

```
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } 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']

        # Accumulator usage: bottom N documents per group
        sales = db['sales']
        sales.insert_many([
            { '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 }
        ])
        accumulator_result = list(sales.aggregate([
```

```
            { '$group': { '_id': '$item', 'bottomTwoSales': { '$bottomN': { 'n':
 2, 'sortBy': { 'quantity': -1 }, 'output': { 'quantity': '$quantity', 'price':
 '$price' } } } } }
        ]))
        pprint(accumulator_result)

        # Window operator usage: bottom two rounds so far within each partition
        match_scores = db['matchScores']
        match_scores.insert_many([
            { '_id': 1, 'player': 'Alice', 'round': 1, 'score': 20 },
            { '_id': 2, 'player': 'Alice', 'round': 2, 'score': 35 },
            { '_id': 3, 'player': 'Alice', 'round': 3, 'score': 28 },
            { '_id': 4, 'player': 'Bob', 'round': 1, 'score': 15 },
            { '_id': 5, 'player': 'Bob', 'round': 2, 'score': 40 }
        ])
        window_result = list(match_scores.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$player',
                    'sortBy': { 'round': 1 },
                    'output': {
                        'bottomTwoRoundsSoFar': {
                            '$bottomN': { 'n': 2, 'sortBy': { 'score': -1 },
 'output': { 'round': '$round', 'score': '$score' } },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        pprint(window_result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$bsonSize

バージョン 8.0.1 から新規。

Amazon DocumentDB の $bsonSize演算子は、BSON としてエンコードされたときにドキュメントのサイズをバイト単位で返します。

パラメータ

- expression: ドキュメントまたはオブジェクトに解決される式。

例 (MongoDB シェル)

次の例は、 $bsonSize演算子を使用して各ドキュメントの BSON サイズを返す方法を示しています。

サンプルドキュメントを作成する

```
db.items.insertMany([
  {_id: 1, name: "a", tags: ["x"]},
  {_id: 2, name: "ab", tags: ["x", "y", "z"]},
  {_id: 3, name: "abc", tags: []}
]);
```

クエリの例

```
db.items.aggregate([
  { $project: { docSize: { $bsonSize: "$$ROOT" } } }
]);
```

出力

```
[
  {_id: 1, docSize: 46},
  {_id: 2, docSize: 65},
  {_id: 3, docSize: 39}
]
```

> **Note**
>
> 正確なサイズは、ドキュメントフィールドの BSON エンコードによって異なります。

コードの例

$bsonSize 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0 の新機能

Elastic クラスターではサポートされていません。

Amazon DocumentDB の$bucket集約ステージでは、指定された式とバケットの境界に基づいて入力ドキュメントをバケットにグループ化できます。これは、特定の値の範囲またはカテゴリに含まれるデータの分析に役立ちます。

パラメータ

- groupBy (必須): グループ化する値を指定する式。
- boundaries (必須): バケットの境界を定義する二重値の配列。ドキュメントは、指定された境界内にあるgroupBy式の値に基づいてバケットに割り当てられます。
- default (オプション): groupBy式値が指定された境界のいずれにも入らないドキュメントに対して出力されるリテラル値。
- output (オプション): 各バケットに出力する情報を指定するオブジェクト。、$sum、$avg、 などのアキュムレータ演算子を使用して$min、各バケットの集計を$max計算できます。

例 (MongoDB シェル)

次の例は、 $bucketステージを使用して、価格範囲別に売上データをグループ化する方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.sales.aggregate([
  {
    $bucket: {
      groupBy: "$price",
      boundaries: [0, 10, 20, 30],
      default: "Other",
      output: {
        "count": { $sum: 1 },
        "totalQuantity": { $sum: "$quantity" }
      }
    }
  },
  {
    $sort: { _id: 1 }
  }
])
```

出力

```
[
  { _id: 0, count: 1, totalQuantity: 3 },
  { _id: 10, count: 2, totalQuantity: 4 },
  { _id: 20, count: 2, totalQuantity: 2 }
]
```

コードの例

$bucket コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

$bucketAuto

バージョン 8.0.2 の新機能。

Amazon DocumentDB の$bucketAuto集約ステージは、指定された式に基づいて、指定された数のバケットに入力ドキュメントを自動的にグループ化します。バケットの境界を明示的に定義する とは異なり$bucket、 は、ドキュメントがリクエストされた数のバケットに可能な限り均等に分散されるように、境界$bucketAutoを自動的に決定します。これは、事前に値の分布を知らずに固定数のグループにデータを分類する場合に便利です。

パラメータ

- groupBy (必須): グループ化する値を指定する式。
- buckets (必須): ドキュメントがグループ化されるバケットの数を指定する正の整数。入力ドキュメントとgroupBy式に応じて、実際のバケット数は指定された数未満にすることができます。
- output (オプション): 各バケットに出力する情報を指定するオブジェクト。、$sum、$avg、 などのアキュムレータ演算子を使用して$min、各バケットの集計を$max計算できます。省略すると、各バケットはデフォルトで countフィールドを出力します。
- granularity (オプション): バケットの境界に使用する優先数値系列を指定する文字列。これにより、計算された境界が標準の数値進行にスナップされます。このパラメータは、

すべてのgroupBy値が数値で、いずれも でない場合にのみ使用できますNaN。有効な値は、R5、、R10、R20、R40、R80、1-2-5、、E6、E12、E24、E48E96E192、、および ですPOWERSOF2。

出力

各バケットは、バケットの境界を記述する_idオブジェクトを含む個別のドキュメントとして返されます。

- _id.min: バケットの包括的な下限。
- _id.max: バケットの上限。このバインドは、系列内の最後のバケットを除くすべてのバケットに排他的です。

例 (MongoDB シェル)

次の例は、 $bucketAutoステージを使用して販売データを価格別にバケットにグループ化する方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

次のコマンドは、サンプルドキュメントを料金別に 3 つのバケットにグループ化します。

```
db.sales.aggregate([
  {
    $bucketAuto: {
      groupBy: "$price",
      buckets: 3,
      output: {
        "count": { $sum: 1 },
        "totalQuantity": { $sum: "$quantity" }
```

```
      }
    }
  }
])
```

出力

このコマンドは、以下の出力を返します。

```
[
  { _id: { min: 5, max: 15 }, count: 2, totalQuantity: 5 },
  { _id: { min: 15, max: 25 }, count: 2, totalQuantity: 3 },
  { _id: { min: 25, max: 25 }, count: 1, totalQuantity: 1 }
]
```

きめ細かなクエリの例

次の例では、計算されたバケットの境界を 2 乗にスナップするgranularityオプションを追加します。

```
db.sales.aggregate([
  {
    $bucketAuto: {
      groupBy: "$price",
      buckets: 2,
      granularity: "POWERSOF2",
      output: {
        "count": { $sum: 1 },
        "totalQuantity": { $sum: "$quantity" }
      }
    }
  }
])
```

出力

このコマンドは、以下の出力を返します。

```
[
  { _id: { min: 4, max: 16 }, count: 3, totalQuantity: 7 },
  { _id: { min: 16, max: 32 }, count: 2, totalQuantity: 2 }
]
```

コードの例

$bucketAuto コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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([
    {
      $bucketAuto: {
        groupBy: "$price",
        buckets: 3,
        output: {
          "count": { $sum: 1 },
          "totalQuantity": { $sum: "$quantity" }
        }
      }
    }
  ]).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']
```

```
    sales = db['sales']

    result = list(sales.aggregate([
        {
            '$bucketAuto': {
                'groupBy': '$price',
                'buckets': 3,
                'output': {
                    'count': {'$sum': 1},
                    'totalQuantity': {'$sum': '$quantity'}
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$ceil

バージョン 4.0 の新機能

Amazon DocumentDB の $ceil演算子は、MongoDB と同様に、数値を最も近い整数に切り上げます。これは、数値フィールドで数学演算を実行し、結果が整数であることを確認する必要がある場合に便利です。

パラメータ

- expression: 切り上げる数値式。

例 (MongoDB シェル)

この例では、 $ceil演算子を使用して数値フィールドを四捨五入する方法を示します。

サンプルドキュメントを作成する

```
db.numbers.insertMany([
  { "_id": 1, "value": 3.14 },
  { "_id": 2, "value": -2.7 },
  { "_id": 3, "value": 0 }
```

```
])
```

クエリの例

```
db.numbers.aggregate([
  { $project: {
    "roundedUp": { $ceil: "$value" }
  }}
])
```

出力

```
{ "_id": 1, "roundedUp": 4 }
{ "_id": 2, "roundedUp": -2 }
{ "_id": 3, "roundedUp": 0 }
```

コードの例

$ceil コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 クラスターではサポートされていません。

$changeStream 集約ステージで変更ストリームカーソルを開き、コレクションへのリアルタイムの変更をモニタリングします。挿入、更新、置換、または削除オペレーションが発生すると、変更イベントドキュメントが返されます。

パラメータ

- fullDocument: 更新オペレーションのために完全なドキュメントを返すかどうかを指定します。オプションは default または updateLookup です。
- resumeAfter: オプション。トークンを再開して、変更ストリームの特定のポイントから続行します。
- startAtOperationTime: オプション。変更ストリームを開始するタイムスタンプ。
- allChangesForCluster: オプション。ブール値。の場合true、 はクラスター全体のすべての変更を監視します (管理者データベース用) 。false (デフォルト) の場合、 は指定されたコレクションのみを監視します。

- showExpandedEvents: オプション。ブール値。の場合true、 は変更ストリームcreateIndexesに createCollectionや などの拡張データ定義言語 (DDL) イベント通知を含めます。false (デフォルト) の場合、これらのイベントは返されません。エンジンバージョン 8.0.2 以降、Amazon DocumentDB 8.0 で使用できます。

例 (MongoDB シェル)

次の例は、 $changeStreamステージを使用してコレクションの変更をモニタリングする方法を示しています。

クエリの例

```
// 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()));
}
```

出力

```
{
  _id: { _data: '...' },
  operationType: 'insert',
  clusterTime: Timestamp(1, 1234567890),
  fullDocument: { _id: 1, item: 'Widget', qty: 10 },
  ns: { db: 'test', coll: 'inventory' },
  documentKey: { _id: 1 }
}
```

コードの例

$changeStream 集約ステージを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

$changeStreamSplitLargeEvent

この集約ステージは Elastic クラスターではサポートされていません。

このステージは、Amazon DocumentDB エンジンバージョン 8.0.2 以降で使用できます。

$changeStreamSplitLargeEvent 集約ステージは $changeStream パイプライン内で使用され、16 MB を超えるイベントを小さなフラグメントに分割します。変更ストリームカーソルを介して各フラグメントを順番に返します。

パラメータ

なし。$changeStreamSplitLargeEvent ステージへの入力は空のドキュメントである必要があります。

例 – MongoDB シェル

次の例は、 $changeStreamSplitLargeEventステージを使用してオーバーサイズの変更イベントを複数のフラグメントに分割する方法を示しています。

クエリの例

```
// Insert a document
db.inventory.insertOne({ _id: 1, item: "Widget", payload: "x".repeat(9 * 1024 *
 1024) })
```

```
// Open change stream with the $changeStreamSplitLargeEvent stage
var changeStream = db.inventory.aggregate([
  { $changeStream: { fullDocument: "updateLookup" } },
  { $changeStreamSplitLargeEvent: {} }
]);

// Update the document
db.inventory.updateOne({ _id: 1 }, { $set: { payload: "y".repeat(9 * 1024 * 1024) } })

// Read the change event
if (changeStream.hasNext()) {
  print(tojson(changeStream.next()));
}
```

出力

```
{
  _id: { _data: '...' },
  splitEvent: { fragment: 1, of: 2},
  ns: { db: 'test', coll: 'inventory' },
  clusterTime: Timestamp(4, 1789154779),
  documentKey: { _id: 1 },
  fullDocument: { _id: 1, item: 'Widget', payload: yyyyyyyy...(9437184 chars) },
  operationType: 'update',
}
{
  _id: { _data: '...' },
  splitEvent: { fragment: 2, of: 2},
  updateDescription: { updatedFields: { payload: yyyyyyyy...(9437184 chars) },
 removedFields: [] }
}
```

コードの例

$changeStreamSplitLargeEvent 集約ステージを使用するコード例を表示するには、使用する言語のタブを選択します。

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');

  // Insert a large document first
  await collection.insertOne({ _id: 1, item: 'Widget', payload: 'x'.repeat(9 * 1024
 * 1024) });

  // Open change stream with $changeStreamSplitLargeEvent
  const changeStream = collection.watch(
    [{ $changeStreamSplitLargeEvent: {} }],
    { fullDocument: 'updateLookup' }
  );

  changeStream.on('change', (change) => {
    console.log('Change detected:', change);
  });

  // Update to trigger a split event — fullDocument + updateDescription > 16 MB
  setTimeout(async () => {
    console.log('Triggering update...');
    await collection.updateOne({ _id: 1 }, { $set: { payload: 'y'.repeat(9 * 1024 *
 1024) } });
  }, 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']

    # Insert large document first
    collection.drop()
    collection.insert_one({'_id': 1, 'item': 'Widget', 'payload': 'x' * (9 * 1024 *
 1024)})

    # Open change stream with $changeStreamSplitLargeEvent
    change_stream = collection.watch(
        [{'$changeStreamSplitLargeEvent': {}}],
        full_document='updateLookup'
    )

    # Update in separate thread after delay
    def update_doc():
        collection.update_one({'_id': 1}, {'$set': {'payload': 'y' * (9 * 1024 *
 1024)}})

    threading.Thread(target=update_doc).start()

    # Watch for changes
    for change in change_stream:
        print('Change detected:', change)

    client.close()

example()
```

$cmp

Amazon DocumentDB の $cmp演算子は、2 つの値を比較し、相対的な順序を示す整数値を返すために使用されます。これは、2 つの式を比較し、最初の値がそれぞれ 2 番目の値より小さい、等しい、または大きいかどうかに応じて、-1、0、または 1 の整数値を返します。

パラメータ

- `expression1`: 比較する最初の式。
- `expression2`: 比較する 2 番目の式。

例 (MongoDB シェル)

次の例は、$cmp演算子を使用して 2 つの数値を比較する方法を示しています。

サンプルドキュメントを作成する

```
db.collection.insertMany([
  { _id: 1, value1: 10, value2: 20 },
  { _id: 2, value1: 15, value2: 15 },
  { _id: 3, value1: 20, value2: 10 }
]);
```

クエリの例

```
db.collection.find({
  $expr: {
    $cmp: ["$value1", "$value2"]
  }
})
```

出力

```
[
  { "_id" : 1, "value1" : 10, "value2" : 20 },
  { "_id" : 3, "value1" : 20, "value2" : 10 }
]
```

この例では、 $cmp演算子は各ドキュメントの フィールドvalue1と value2フィールドを比較します。結果は、次のとおりです。

```
- `$cmp: ["$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).
```

コードの例

$cmp コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

Node.js

mongodb ドライバーで Node.js アプリケーションで $cmp演算子を使用する例を次に示します。

```
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

pymongo ドライバーで Python アプリケーションで $cmp演算子を使用する例を次に示します。

```
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 と Python の両方の例の出力は、MongoDB シェルの例と同じになります。

```
[
  { "_id" : 1, "value1" : 10, "value2" : 20 },
  { "_id" : 2, "value1" : 15, "value2" : 15 },
  { "_id" : 3, "value1" : 20, "value2" : 10 }
]
```

$collStats

バージョン 4.0 の新機能

Amazon DocumentDB の`$collStats`集約ステージは、MongoDB シェルの `db.collection.stats()` コマンドと同様に、指定されたコレクションに関する統計を提供します。このステージを使用して、ドキュメントの数、コレクションの合計サイズ、さまざまなパフォーマンスメトリクスなど、コレクションに関する情報を取得できます。

パラメータ

- `latencyStats`: (オプション) レイテンシー統計を収集するためのオプションを指定するドキュメント。このパラメータは Amazon DocumentDB ではサポートされていません。
- `recordStats`: (オプション) レコード統計を収集するためのオプションを指定するドキュメント。このパラメータは Amazon DocumentDB ではサポートされていません。

- queryExecStats: (オプション) クエリ実行統計を収集するためのオプションを指定するドキュメント。このパラメータは Amazon DocumentDB ではサポートされていません。

例 (MongoDB シェル)

次の例は、$collStats集約ステージを使用して、dbデータベースtest内の という名前のコレクションに関する統計を取得する方法を示しています。

サンプルドキュメントを作成する

```
db.test.insertMany([
  { "name": "John", "age": 30 },
  { "name": "Jane", "age": 25 },
  { "name": "Bob", "age": 35 }
]);
```

クエリの例

```
db.test.aggregate([
  { $collStats: {} }
]);
```

出力

```
{
  "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)
}
```

コードの例

$collStats コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

Node.js

以下は、公式の MongoDB Node.js ドライバーを使用して Node.js アプリケーションで$collStats集約ステージを使用する方法の例です。

```
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

PyMongo ドライバーを使用して Python アプリケーションで$collStats集約ステージを使用する方法の例を次に示します。

```
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 の$concat集計演算子は、ドキュメント内の複数の文字列を連結 (または結合) して、アプリケーションに返すことができる単一の文字列を生成します。これにより、文字列操作がデータベースレベルで実行されるため、アプリケーションで実行される作業が減ります。

パラメータ

- `expression1`: 連結する最初の文字列。
- `expression2`: 連結する 2 番目の文字列。
- `...`: 連結する追加の文字列 (オプション) 。

例 (MongoDB シェル)

この例では、ユーザーの姓名を連結して、各ユーザーのフルネームを生成します。

サンプルドキュメントを作成する

```
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"}
])
```

クエリの例

```
db.people.aggregate([
  { $project: { full_name: { $concat: [ "$first_name", " ", "$last_name"] } } }
])
```

出力

```
{ "_id" : 1, "full_name" : "Jane Doe" }
{ "_id" : 2, "full_name" : "John Doe" }
{ "_id" : 3, "full_name" : "Steve Smith" }
```

コードの例

$concat コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$concatArrays集計演算子は、2 つ以上の配列を 1 つの配列に連結するために使用されます。これは、データの複数の配列を 1 つの配列に結合して、さらに処理または分析する必要がある場合に便利です。

パラメータ

- array1: 連結する最初の配列。
- array2: 連結する 2 番目の配列。
- [array3, ...]: (オプション) 連結する追加の配列。

例 (MongoDB シェル)

次の例は、 $concatArrays演算子を使用して 2 つの配列を 1 つの配列に結合する方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.collection.aggregate([
  {
    $project: {
      _id: 0,
      name: 1,
      all_activities: { $concatArrays: ["$hobbies", "$skills"] }
    }
  }
]);
```

出力

```
[
  {
    "name": "John Doe",
    "all_activities": [
      "reading",
      "swimming",
      "programming",
```

```
      "design"
    ]
  },
  {
    "name": "Jane Smith",
    "all_activities": [
      "hiking",
      "cooking",
      "marketing",
      "analysis"
    ]
  }
]
```

コードの例

$concatArrays コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $cond演算子は、条件式を評価し、2 つの可能な結果式のいずれかを返すために使用されます。

パラメータ

- `if`: 評価するブール式。
- `then`: 式が true の場合に返されるif式。
- `else`: 式が false の場合に返されるif式。

例 (MongoDB シェル)

次の例は、 $cond演算子を使用して、人の年齢に基づいて値を返す方法を示しています。

サンプルドキュメントを作成する

```
db.people.insertMany([
  { _id: 1, name: "John Doe", age: 35 },
  { _id: 2, name: "Jane Doe", age: 25 },
  { _id: 3, name: "Bob Smith", age: 65 }
]);
```

クエリの例

```
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"
            }
          }
        }
      }
    }
  }
])
```

出力

```
[
  { "_id" : 1, "name" : "John Doe", "ageGroup" : "middle-aged" },
  { "_id" : 2, "name" : "Jane Doe", "ageGroup" : "young" },
  { "_id" : 3, "name" : "Bob Smith", "ageGroup" : "elderly" }
]
```

コードの例

$cond コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

$convert

バージョン 4.0 の新機能

Amazon DocumentDB の $convert演算子は、あるデータ型から別のデータ型に値を変換するために使用されます。この演算子は、文字列を数値に変換したり、日付をタイムスタンプに変換したりするなど、さまざまなタイプのデータに対してオペレーションを実行する必要がある場合に便利です。

パラメータ

- to: 値を変換するターゲットデータ型。サポートされている値は "string"、"double"、"long"、"int"、"date"、"boolean" です。
- from: 値の現在のデータ型。指定しない場合、Amazon DocumentDB はデータ型を自動的に検出しようとします。
- onError: (オプション) 変換が失敗した場合に返される値。特定の値、または次の特殊な値のいずれかを指定できます: "null"、"zerofill"、または "error"。
- onNull: (オプション) 入力値が の場合に返される値null。特定の値、または次の特殊な値のいずれかを指定できます: "null"、"zerofill"、または "error"。

例 (MongoDB シェル)

次の例は、 $convert演算子を使用して文字列値を日付に変換する方法を示しています。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.users.aggregate([
  {
    $project: {
      _id: 1,
      name: 1,
      joinedOn: {
        $convert: {
          input: "$joinedOn",
          to: "date",
          onError: "null",
```

```
          onNull: "null"
        }
      }
    }
  }
])
```

出力

```
[
  { "_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 }
]
```

コードの例

$convert コマンドを使用するためのコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 の新機能。

Amazon DocumentDB の$cos演算子は、ラジアンで測定される値のコサインを返します。集約パイプラインでこれを使用して、数値フィールドで三角計算を実行します。

パラメータ

- `expression`: ラジアン単位で数値に解決される式。

例 (MongoDB シェル)

次の例は、 $cos演算子を使用して角度値のコサインをラジアン単位で計算する方法を示しています。

サンプルドキュメントを作成する

```
db.angles.insertMany([
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 1.0471975511965976 },
  { "_id": 3, "angle": 3.141592653589793 }
]);
```

クエリの例

```
db.angles.aggregate([
  { $project: {
    "cosine": { $cos: "$angle" }
  }}
]);
```

出力

```
[
  { "_id": 1, "cosine": 1 },
  { "_id": 2, "cosine": 0.5000000000000001 },
  { "_id": 3, "cosine": -1 }
]
```

コードの例

$cos 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 の新機能。

Amazon DocumentDB の $cosh演算子は、値のハイパーボリックコサインを返します。集約パイプラインでこれを使用して、数値フィールドに対して双曲線三角計算を実行します。

パラメータ

- expression: 数値に解決される式。

戻り値の型はdoubleデフォルトで です。入力が 128 ビットの 10 進数の場合、出力も 128 ビットの 10 進数です。

例 (MongoDB シェル)

次の例は、 $cosh演算子を使用して数値の双曲線コサインを計算する方法を示しています。

サンプルドキュメントを作成する

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 1 },
  { "_id": 3, "value": -1 }
]);
```

クエリの例

```
db.values.aggregate([
  { $project: {
    "result": { $cosh: "$value" }
  }}
]);
```

出力

```
[
  { "_id": 1, "result": 1 },
  { "_id": 2, "result": 1.5430806348152437 },
  { "_id": 3, "result": 1.5430806348152437 }
]
```

コードの例

$cosh 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 (ステージ)

Amazon DocumentDB の$count集約ステージは、ステージに渡すドキュメントの数をカウントするために使用されます。多くの場合、前のステージに一致するドキュメントの合計数を返すために、集約パイプラインの最終ステージとして使用されます。

パラメータ

- field: カウントするフィールド。このパラメータはオプションであり、指定しない場合、ステージは入力ドキュメントの総数をカウントします。

例 (MongoDB シェル)

次の例は、 $countステージを使用してコレクション内のドキュメントの総数を取得する方法を示しています。

サンプルドキュメントを作成する

```
db.users.insertMany([
  { name: "John", age: 30 },
  { name: "Jane", age: 25 },
  { name: "Bob", age: 35 },
  { name: "Alice", age: 28 }
]);
```

クエリの例

```
db.users.aggregate([
  { $count: "total" }
]);
```

出力

```
{ "total" : 4 }
```

この例では、usersコレクションを集約し、 $countステージを使用してドキュメントの総数をカウントします。

コードの例

$count コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 (アキュムレータ)

バージョン 8.0.1 の新機能。

$group ステージ内の$countアキュムレータを使用して、各グループのドキュメント数を返します。引数{}として空のオブジェクトを受け入れます。

これは、$countパイプラインを通過するすべてのドキュメントをカウントするスタンドアロンステージであるパイプラインステージとは異なります。アキュムレータは代わりに$count、 によって生成された各グループ内のドキュメントをカウントします$group。

[Syntax] (構文)

```
{ $count: {} }
```

パラメータ

- $count アキュムレータは引数を取りません。空のオブジェクトを受け入れ{}、各グループのドキュメントの数を返します。

例 (MongoDB シェル)

次の例は、 $count アキュムレータを使用して各カテゴリの製品数をカウントする方法を示しています。

サンプルドキュメントを作成する

```
db.products.insertMany([
  { name: "Widget", category: "A" },
  { name: "Gadget", category: "A" },
  { name: "Doohickey", category: "B" },
  { name: "Thingamajig", category: "B" },
  { name: "Whatsit", category: "B" }
])
```

クエリの例

```
db.products.aggregate([
  { $group: { _id: "$category", count: { $count: {} } } }
])
```

出力

```
[
  { "_id": "A", "count": 2 },
  { "_id": "B", "count": 3 }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$count アキュムレータは、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメント数を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

サンプルドキュメントを作成する

```
db.events.insertMany([
  { _id: 1, category: "A", day: 1 },
  { _id: 2, category: "A", day: 2 },
  { _id: 3, category: "A", day: 3 },
  { _id: 4, category: "B", day: 1 },
  { _id: 5, category: "B", day: 2 }
])
```

クエリの例

次の例ではcategory、ドキュメントを でパーティション化し、各パーティションを でソートしてday、パーティションの開始から現在のドキュメントまでのドキュメントの実行数を返します。

```
db.events.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$category",
      sortBy: { day: 1 },
      output: {
        runningCount: {
          $count: {},
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
])
```

出力

```
[
  { "_id": 1, "category": "A", "day": 1, "runningCount": 1 },
  { "_id": 2, "category": "A", "day": 2, "runningCount": 2 },
  { "_id": 3, "category": "A", "day": 3, "runningCount": 3 },
  { "_id": 4, "category": "B", "day": 1, "runningCount": 1 },
  { "_id": 5, "category": "B", "day": 2, "runningCount": 2 }
]
```

各ドキュメントはrunningCount、現在のドキュメントまでのパーティション内のドキュメントの数である で拡張されます。

コードの例

$count アキュムレータを使用するためのコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group) とウィンドウ演算子の使用 (の場合) の両方を示しています$setWindowFields。

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: count per group
    const products = db.collection('products');
    await products.insertMany([
      { name: "Widget", category: "A" },
      { name: "Gadget", category: "A" },
      { name: "Doohickey", category: "B" },
      { name: "Thingamajig", category: "B" },
      { name: "Whatsit", category: "B" }
    ]);
    const accumulatorResult = await products.aggregate([
      { $group: { _id: "$category", count: { $count: {} } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Window operator usage: running count within each partition
    const events = db.collection('events');
    await events.insertMany([
      { _id: 1, category: "A", day: 1 },
      { _id: 2, category: "A", day: 2 },
      { _id: 3, category: "A", day: 3 },
      { _id: 4, category: "B", day: 1 },
      { _id: 5, category: "B", day: 2 }
    ]);
    const windowResult = await events.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$category",
          sortBy: { day: 1 },
          output: {
            runningCount: {
              $count: {},
              window: { documents: ["unbounded", "current"] }
            }
          }
```

```
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } 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']

        # Accumulator usage: count per group
        products = db['products']
        products.insert_many([
            { 'name': 'Widget', 'category': 'A' },
            { 'name': 'Gadget', 'category': 'A' },
            { 'name': 'Doohickey', 'category': 'B' },
            { 'name': 'Thingamajig', 'category': 'B' },
            { 'name': 'Whatsit', 'category': 'B' }
        ])
        accumulator_result = list(products.aggregate([
            { '$group': { '_id': '$category', 'count': { '$count': {} } } }
        ]))
        pprint(accumulator_result)

        # Window operator usage: running count within each partition
```

```
        events = db['events']
        events.insert_many([
            { '_id': 1, 'category': 'A', 'day': 1 },
            { '_id': 2, 'category': 'A', 'day': 2 },
            { '_id': 3, 'category': 'A', 'day': 3 },
            { '_id': 4, 'category': 'B', 'day': 1 },
            { '_id': 5, 'category': 'B', 'day': 2 }
        ])
        window_result = list(events.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$category',
                    'sortBy': { 'day': 1 },
                    'output': {
                        'runningCount': {
                            '$count': {},
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        pprint(window_result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$covariancePop

バージョン 8.0.2 の新機能。

Amazon DocumentDB の $covariancePop演算子は、2 つの数値式の母集団共分散を返します。$setWindowFields ステージでのみ使用されるウィンドウ演算子であり、 $groupまたは $bucketステージでは有効ではありません。ステージの outputフィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

パラメータ

- $covariancePop は 2 要素配列 を取り[<expression1>, <expression2>]ます。各要素は数値式 ("$x"、式、定数などのフィールドパス) です。ウィンドウ内のドキュメント全体で 2 つの式の母集団共分散を返します。

例 (MongoDB シェル)

次の例では、ドキュメントを でパーティション化しseries、各パーティションを でソートしt、パーティション全体の x と の母集団共分散を計算yします。

サンプルドキュメントを作成する

```
db.measurements.insertMany([
  { _id: 1, series: "A", t: 1, x: 1, y: 2 },
  { _id: 2, series: "A", t: 2, x: 2, y: 5 },
  { _id: 3, series: "A", t: 3, x: 3, y: 8 },
  { _id: 4, series: "B", t: 1, x: 1, y: 8 },
  { _id: 5, series: "B", t: 2, x: 2, y: 5 },
  { _id: 6, series: "B", t: 3, x: 3, y: 2 }
]);
```

クエリの例

```
db.measurements.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$series",
      sortBy: { t: 1 },
      output: {
        covariance: {
          $covariancePop: [ "$x", "$y" ],
          window: { documents: ["unbounded", "unbounded"] }
        }
      }
    }
  }
]);
```

出力

```
[
```

```
  { "_id": 1, "series": "A", "t": 1, "x": 1, "y": 2, "covariance": 2 },
  { "_id": 2, "series": "A", "t": 2, "x": 2, "y": 5, "covariance": 2 },
  { "_id": 3, "series": "A", "t": 3, "x": 3, "y": 8, "covariance": 2 },
  { "_id": 4, "series": "B", "t": 1, "x": 1, "y": 8, "covariance": -2 },
  { "_id": 5, "series": "B", "t": 2, "x": 2, "y": 5, "covariance": -2 },
  { "_id": 6, "series": "B", "t": 3, "x": 3, "y": 2, "covariance": -2 }
]
```

パーティション内のすべてのドキュメントは同じ を受け取りcovariance、そのパーティション内のすべてのドキュメントyで xと の母集団共分散を受け取ります。系列2の場合 A (x と y が一緒に増加)、系列-2の場合 B (y は x が増加するにつれて減少)。

コードの例

$setWindowFields ステージで $covariancePop演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 measurements = db.collection('measurements');

    await measurements.insertMany([
      { _id: 1, series: "A", t: 1, x: 1, y: 2 },
      { _id: 2, series: "A", t: 2, x: 2, y: 5 },
      { _id: 3, series: "A", t: 3, x: 3, y: 8 },
      { _id: 4, series: "B", t: 1, x: 1, y: 8 },
      { _id: 5, series: "B", t: 2, x: 2, y: 5 },
      { _id: 6, series: "B", t: 3, x: 3, y: 2 }
    ]);

    const result = await measurements.aggregate([
      {
        $setWindowFields: {
```

```
          partitionBy: "$series",
          sortBy: { t: 1 },
          output: {
            covariance: {
              $covariancePop: [ "$x", "$y" ],
              window: { documents: ["unbounded", "unbounded"] }
            }
          }
        }
      }
    ]).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']
        measurements = db['measurements']

        measurements.insert_many([
            { '_id': 1, 'series': 'A', 't': 1, 'x': 1, 'y': 2 },
            { '_id': 2, 'series': 'A', 't': 2, 'x': 2, 'y': 5 },
            { '_id': 3, 'series': 'A', 't': 3, 'x': 3, 'y': 8 },
            { '_id': 4, 'series': 'B', 't': 1, 'x': 1, 'y': 8 },
            { '_id': 5, 'series': 'B', 't': 2, 'x': 2, 'y': 5 },
            { '_id': 6, 'series': 'B', 't': 3, 'x': 3, 'y': 2 }
        ])

        result = list(measurements.aggregate([
```

```
            {
                '$setWindowFields': {
                    'partitionBy': '$series',
                    'sortBy': { 't': 1 },
                    'output': {
                        'covariance': {
                            '$covariancePop': [ '$x', '$y' ],
                            'window': { 'documents': ['unbounded', 'unbounded'] }
                        }
                    }
                }
            }
        ]))

        print(result)
    finally:
        client.close()

example()
```

$covarianceSamp

バージョン 8.0.2 の新機能。

Amazon DocumentDB の $covarianceSamp演算子は、2 つの数値式のサンプル共分散を返します。$setWindowFields ステージでのみ使用されるウィンドウ演算子であり、 $groupまたは $bucketステージでは有効ではありません。ステージの outputフィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

パラメータ

- $covarianceSamp は 2 要素配列 を取り[<expression1>, <expression2>]ます。各要素は数値式 ("$x"、式、定数などのフィールドパス) です。ウィンドウ内のドキュメント全体で 2 つの式のサンプル共分散を返します。

例 (MongoDB シェル)

次の例では、ドキュメントを でパーティション化しseries、各パーティションを でソートしt、パーティション全体の x と のサンプル共分散を計算yします。

サンプルドキュメントを作成する

```
db.measurements.insertMany([
  { _id: 1, series: "A", t: 1, x: 1, y: 2 },
  { _id: 2, series: "A", t: 2, x: 2, y: 5 },
  { _id: 3, series: "A", t: 3, x: 3, y: 8 },
  { _id: 4, series: "B", t: 1, x: 1, y: 8 },
  { _id: 5, series: "B", t: 2, x: 2, y: 5 },
  { _id: 6, series: "B", t: 3, x: 3, y: 2 }
]);
```

クエリの例

```
db.measurements.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$series",
      sortBy: { t: 1 },
      output: {
        covariance: {
          $covarianceSamp: [ "$x", "$y" ],
          window: { documents: ["unbounded", "unbounded"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "series": "A", "t": 1, "x": 1, "y": 2, "covariance": 3 },
  { "_id": 2, "series": "A", "t": 2, "x": 2, "y": 5, "covariance": 3 },
  { "_id": 3, "series": "A", "t": 3, "x": 3, "y": 8, "covariance": 3 },
  { "_id": 4, "series": "B", "t": 1, "x": 1, "y": 8, "covariance": -3 },
  { "_id": 5, "series": "B", "t": 2, "x": 2, "y": 5, "covariance": -3 },
  { "_id": 6, "series": "B", "t": 3, "x": 3, "y": 2, "covariance": -3 }
]
```

パーティション内のすべてのドキュメントが同じ を受け取りcovariance、そのパーティション内のすべてのドキュメントy間で xと のサンプル共分散を受け取ります。系列3の場合 A (x と y が一緒に増加）、系列-3の場合 B (y は x が増加するにつれて減少）。サンプル共分散は n - 1ではなくで分割されるためn、同じデータの母集団共分散よりも大きくなります。

コードの例

$setWindowFields ステージで $covarianceSamp演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 measurements = db.collection('measurements');

    await measurements.insertMany([
      { _id: 1, series: "A", t: 1, x: 1, y: 2 },
      { _id: 2, series: "A", t: 2, x: 2, y: 5 },
      { _id: 3, series: "A", t: 3, x: 3, y: 8 },
      { _id: 4, series: "B", t: 1, x: 1, y: 8 },
      { _id: 5, series: "B", t: 2, x: 2, y: 5 },
      { _id: 6, series: "B", t: 3, x: 3, y: 2 }
    ]);

    const result = await measurements.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$series",
          sortBy: { t: 1 },
          output: {
            covariance: {
              $covarianceSamp: [ "$x", "$y" ],
              window: { documents: ["unbounded", "unbounded"] }
            }
          }
        }
      }
    ]).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']
        measurements = db['measurements']

        measurements.insert_many([
            { '_id': 1, 'series': 'A', 't': 1, 'x': 1, 'y': 2 },
            { '_id': 2, 'series': 'A', 't': 2, 'x': 2, 'y': 5 },
            { '_id': 3, 'series': 'A', 't': 3, 'x': 3, 'y': 8 },
            { '_id': 4, 'series': 'B', 't': 1, 'x': 1, 'y': 8 },
            { '_id': 5, 'series': 'B', 't': 2, 'x': 2, 'y': 5 },
            { '_id': 6, 'series': 'B', 't': 3, 'x': 3, 'y': 2 }
        ])

        result = list(measurements.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$series',
                    'sortBy': { 't': 1 },
                    'output': {
                        'covariance': {
                            '$covarianceSamp': [ '$x', '$y' ],
                            'window': { 'documents': ['unbounded', 'unbounded'] }
                        }
                    }
                }
            }
        ]))
```

```
        print(result)
    finally:
        client.close()

example()
```

$currentOp

$currentOp 集約ステージは、データベースで現在実行中のオペレーションに関する情報を返します。このステージは、集約パイプラインでアクティブなクエリとオペレーションをモニタリングするのに役立ちます。

パラメータ

- allUsers (オプション): に設定するとtrue、 はすべてのユーザーのオペレーションを返します。デフォルトは false です。
- idleConnections (オプション): に設定するとtrue、 にはアイドル接続が含まれます。デフォルトは false です。
- idleCursors (オプション): に設定するとtrue、 にはアイドルカーソルに関する情報が含まれます。デフォルトは false です。
- idleSessions (オプション): に設定するとtrue、 にはアイドルセッションに関する情報が含まれます。デフォルトは true です。
- localOps (オプション): に設定するとtrue、 にはローカルオペレーションが含まれます。デフォルトは false です。

例 (MongoDB シェル)

次の例は、 $currentOp 集約ステージを使用してアクティブな読み取りオペレーションに関する情報を取得する方法を示しています。

クエリの例

```
db.aggregate([
  { $currentOp: { allUsers: true, idleConnections: false } },
  { $match: { op: "query" } }
])
```

出力

```
[
  {
    "opid": "12345",
    "active": true,
    "op": "query",
    "ns": "test.users",
    "secs_running": 2
  }
]
```

コードの例

$currentOp コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 5.0 の新機能

Amazon DocumentDB の$dateAdd集計演算子を使用すると、日付と時刻の値に期間を追加できます。

パラメータ

- date: 期間を追加する日付と時刻の値。
- duration: date値に追加する期間。これは、、years、、months、weeks、、dayshoursminutesおよび のキーを持つオブジェクトとして指定できますseconds。
- timezone: (オプション) 日付の追加を実行するときに使用するタイムゾーン。指定しない場合、Amazon DocumentDB クラスターのデフォルトのタイムゾーンが使用されます。

例 (MongoDB シェル)

次の例は、 $dateAdd演算子を使用して日付に 2 日と 12 時間を追加する方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.events.aggregate([
  {
    $project: {
      _id: 1,
      eventDate: 1,
      eventDatePlustwodaysandtwelvehours: {
        $dateAdd: {
          startDate: {
            $dateAdd: {
              startDate: "$eventDate",
              unit: "day",
              amount: 2
            }
          },
          unit: "hour",
          amount: 12
        }
      }
    }
  }
])
```

出力

```
[
  {
    "_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)"
  }
]
```

コードの例

$dateAdd コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 5.0 の新機能

Elastic クラスターではサポートされていません。

$dateDiff 集計演算子は、指定された単位で 2 つの日付の差を計算します。開始日と終了日の間の単位境界の数を返します。

パラメータ

- startDate: 開始日の式。
- endDate: 終了日の式。
- unit: 差の時間単位。サポートされている単位は、year、、quarter、monthweek、、day、hourminutesecond、、および ですmillisecond。

例 (MongoDB シェル)

次の例は、 $dateDiff演算子を使用して、注文から配送までの日数を計算する方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.shipments.aggregate([
```

```
  {
    $project: {
      orderId: 1,
      orderDate: 1,
      deliveryDate: 1,
      daysToDeliver: {
        $dateDiff: {
          startDate: "$orderDate",
          endDate: "$deliveryDate",
          unit: "day"
        }
      }
    }
  }
]);
```

出力

```
[
  {
    _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
  }
]
```

コードの例

$dateDiff コマンドを使用するためのコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 5.0.1 および 8.0 の新機能

Amazon DocumentDB の$dateFromParts集計演算子は、年、月、日、時間、分、秒、ミリ秒などの構成要素から日付と時刻の値を構築して返します。

パラメータ

- year: 暦年。ISO 8601 パーツを使用しない場合に必要です。
- month、day、hour、minute、second、millisecond: (オプション) 残りの日付と時刻のコンポーネント。省略されたコンポーネントは、デフォルトで最も低い有効な値になります。
- isoWeekYear、isoWeek、isoDayOfWeek: (オプション) ISO 8601 日付コンポーネント。ISO 8601 週の日付から日付dayを作成するにはyear、、month、 の代わりにこれらを使用します。
- timezone: (オプション) 日付を作成するときに使用するタイムゾーン。指定しない場合、UTC が使用されます。

例 (MongoDB シェル)

次の例は、 $dateFromParts演算子を使用して個々のコンポーネントから日付を作成する方法を示しています。

サンプルドキュメントを作成する

```
db.parts.insertMany([
  { _id: 1, y: 2023, m: 4, d: 1 },
  { _id: 2, y: 2023, m: 12, d: 25 }
]);
```

クエリの例

```
db.parts.aggregate([
  {
    $project: {
      _id: 1,
      constructedDate: {
        $dateFromParts: { year: "$y", month: "$m", day: "$d" }
      }
    }
  }
])
```

出力

```
[
  { "_id": 1, "constructedDate": ISODate("2023-04-01T00:00:00Z") },
  { "_id": 2, "constructedDate": ISODate("2023-12-25T00:00:00Z") }
]
```

コードの例

$dateFromParts コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$dateFromString集計演算子を使用すると、日時文字列を日付オブジェクトに変換できます。これは、データが文字列として保存されているが、データに対して日付ベースのオペレーションを実行する必要がある場合に便利です。

パラメータ

- dateString: 日付と時刻を表す文字列。
- format: (オプション) の形式を指定する文字列dateString。指定しない場合、Amazon DocumentDB は ISO-8601 形式で文字列の解析を試みます。
- timezone: (オプション) タイムゾーンを指定する文字列。指定しない場合、Amazon DocumentDB はサーバーのタイムゾーンを使用します。
- onError: (オプション) 変換が失敗した場合に実行するアクションを指定します。指定できる値は、 'error' (エラーをスローするデフォルト) 、 'null' (を返すnull)、または 'replace' (値を onErrorMessageオプションで指定された置換文字列に置き換えます) です。
- onErrorMessage: (オプション) onErrorが に設定されている場合'replace'、このオプションは置き換え文字列を指定します。

例 (MongoDB シェル)

次の例は、 $dateFromStringを使用して Amazon DocumentDB の日付文字列を日付オブジェクトに変換する方法を示しています。

サンプルドキュメントを作成する

```
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"}
]);
```

クエリの例

```
db.missionLog.aggregate([
  {
    $project: {
      event: '$event',
      logDate: {
        $dateFromString: {
          dateString: '$logDate'
        }
      }
    }
  }
]);
```

出力

```
[
  {
    "_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")
  }
]
```

コードの例

$dateFromString コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 5.0 の新機能

Amazon DocumentDB の$dateSubtract集計演算子を使用すると、指定された期間を日付値から減算できます。

パラメータ

- date: 日付またはタイムスタンプに解決される日付式。
- subtrahend: 式から減算する時間を指定する期間date式。

- unit: subtrahend式の時間単位を指定する文字列。サポートされている単位は、「年」、「四半期」、「月」、「週」、「日」、「時間」、「分」、「秒」、「ミリ秒」です。

例 (MongoDB シェル)

次の例は、 $dateSubtract演算子を使用して、現在の日付から 1 年前の日付を計算する方法を示しています。

サンプルドキュメントを作成する

```
db.events.insertOne({
  eventName: "Player joined",
  eventTime: ISODate("2023-04-01T12:00:00Z")
});
```

クエリの例

```
db.events.aggregate([
  {
    $project: {
      eventName: 1,
      oneYearAgo: {
        $dateSubtract: {
          startDate: "$eventTime",
          amount: 1,
          unit: "year"
        }
      }
    }
  }
])
```

出力

```
{
  "_id" : ObjectId("64567890abcdef012345678"),
  "eventName" : "Player joined",
  "oneYearAgo" : ISODate("2022-04-01T12:00:00Z")
}
```

コードの例

$dateSubtract コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 5.0.1 および 8.0 の新機能

Amazon DocumentDB の$dateToParts集計演算子は、年、月、日、時間、分、秒、ミリ秒など、日付と時刻の値の構成部分を含むドキュメントを返します。

パラメータ

- date: パートに分割する日付と時刻の値。
- timezone: (オプション) 日付部分を抽出するときに使用するタイムゾーン。指定しない場合、パートは UTC で返されます。
- iso8601: (オプション) に設定するとtrue、返されるドキュメントは ISO 8601 の日付部分 (isoWeekYear、isoWeek、) を使用しますisoDayOfWeek。デフォルトは false です。

例 (MongoDB シェル)

次の例は、 $dateToParts演算子を使用して日付の個々の部分を返す方法を示しています。

サンプルドキュメントを作成する

```
db.events.insertMany([
  { _id: 1, eventDate: ISODate("2023-04-01T10:30:15Z") },
  { _id: 2, eventDate: ISODate("2023-12-25T18:45:00Z") }
]);
```

クエリの例

```
db.events.aggregate([
  {
    $project: {
      _id: 1,
      parts: { $dateToParts: { date: "$eventDate" } }
    }
  }
])
```

出力

```
[
  {
    "_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 }
  }
]
```

コードの例

$dateToParts コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$dateToString集計演算子は、日付またはタイムスタンプの値を文字列表現に変換するために使用されます。これは、表示またはさらなる処理のために特定の方法で日付と時刻をフォーマットする必要がある場合に便利です。

パラメータ

- date: 文字列に変換される日付またはタイムスタンプの値。
- format: 日付を表す形式を指定する文字列。形式文字列には、4 桁の年%Y、%m2 桁の月、2 %d桁の日など、さまざまな形式指定子を含めることができます。
- timezone: (オプション) 変換に使用するタイムゾーン。指定しない場合、Amazon DocumentDB クラスターをホストするサーバーのタイムゾーンが使用されます。
- onNull: (オプション) dateパラメータが の場合に返される値null。

例 (MongoDB シェル)

次の例は、 $dateToString演算子を使用してmissionLogコレクションの logDateフィールドをフォーマットする方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.missionLog.aggregate([
  {
    $project: {
      event: "$event",
      logDateFormatted: {
        $dateToString: {
          format: "%Y-%m-%d %H:%M:%S",
          date: "$logDate"
        }
      }
    }
  }
])
```

出力

```
[
  {
    "_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"
```

```
  }
]
```

コードの例

$dateToString コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

Node.js

Node.js アプリケーションで $dateToString演算子を使用する例を次に示します。

```
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

Python アプリケーションで $dateToString演算子を使用する例を次に示します。

```
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

バージョン 8.0 の新機能

Elastic クラスターではサポートされていません。

Amazon DocumentDB の$dateTrunc集計演算子は、指定された単位に日付を切り捨てます。

パラメータ

- date: 日付またはタイムスタンプに解決される日付式。
- unit: サブtrahend 式の時間単位を指定する文字列。サポートされている単位は、year、、quarter、month、week、day、hourminutesecond、、および ですmillisecond。

例 (MongoDB シェル)

次の例は、 $dateTrunc演算子を使用して日付を に切り捨てて 1 時間にする方法を示しています。

サンプルドキュメントを作成する

```
db.events.insertMany([
  {
    eventName: "Event 1",
    eventTime: ISODate("2025-04-01T12:15:00Z")
  },
  {
    eventName: "Event 2",
    eventTime: ISODate("2025-08-15T14:33:22Z")
  },
]);
```

クエリの例

```
db.events.aggregate([
  {
    $project: {
      eventName: 1,
      eventTime: 1,
      truncatedToHour: {
        $dateTrunc: {
          date: "$eventTime",
          unit: "hour"
        }
      }
    }
  }
]);
```

出力

```
[
  {
    _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')
  }
]
```

コードの例

$dateTrunc コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$dayOfMonth集計演算子は、特定の日付の日 (1～31) を取得します。この演算子は、ドキュメントの日付フィールドから日付をグループ化、フィルタリング、または抽出するのに役立ちます。

パラメータ

- `date expression`: 日付式は、ドキュメントの日付フィールド、日付オブジェクト、または日付文字列です。

例 (MongoDB シェル)

この例では、 $dayOfMonth演算子を使用してドキュメントの日付フィールドから日付を抽出する方法を示します。

サンプルドキュメントを作成する

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

クエリの例

```
db.events.aggregate([
  { $project: {
    eventDay: { $dayOfMonth: "$eventDate" }
  }}
])
```

出力

```
[
  { "_id" : 1, "eventDay" : 15 },
  { "_id" : 2, "eventDay" : 28 },
  { "_id" : 3, "eventDay" : 1 },
  { "_id" : 4, "eventDay" : 30 }
]
```

コードの例

$dayOfMonth コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $dayOfWeek演算子は、特定の日付フィールドから曜日を抽出します。曜日を 1 (日曜日) から 7 (土曜日) までの数値として返します。これは MongoDB と同じ動作です。

パラメータ

- `date field`: 曜日を抽出する日付フィールド。

例 (MongoDB シェル)

この例では、 $dayOfWeek演算子を使用して、weatherコレクションの dateフィールドから曜日を抽出する方法を示します。

サンプルドキュメントを作成する

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

クエリの例

```
db.weather.aggregate([
  {
    $project: {
      dayOfWeek: { $dayOfWeek: "$date" }
    }
  }
]).pretty();
```

出力

```
{ "_id" : ObjectId("64272c6663f4f8ce422c2d91"), "dayOfWeek" : 7 }
{ "_id" : ObjectId("64272c6663f4f8ce422c2d92"), "dayOfWeek" : 1 }
{ "_id" : ObjectId("64272c6663f4f8ce422c2d93"), "dayOfWeek" : 2 }
```

コードの例

$dayOfWeek コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$dayOfYear演算子は、日付の日を 1 ~ 366 (うるう年以外の場合は 365) の数値として返します。

パラメータ

- expression: 日を抽出する日付フィールドまたは式。

例 (MongoDB シェル)

この例では、 $dayOfYear演算子を使用して Amazon DocumentDB コレクションの日付フィールドから日付を抽出する方法を示します。

サンプルドキュメントを作成する

```
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")
  }
])
```

クエリの例

```
db.weather.aggregate([
  {
    $project: {
      dayOfYear: { $dayOfYear: "$date" }
    }
  }
]).pretty()
```

出力

```
{ "_id" : ObjectId("642b86fc7d8e07af279bbe63"), "dayOfYear" : 74 }
{ "_id" : ObjectId("642b86fc7d8e07af279bbe64"), "dayOfYear" : 365 }
```

コードの例

$dayOfYear コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の $degreesToRadians演算子は、値を度からラジアンに変換します。

パラメータ

- expression: 度数で数値に解決される式。

例 (MongoDB シェル)

次の例は、 $degreesToRadians演算子を使用して度値をラジアンに変換する方法を示しています。

サンプルドキュメントを作成する

```
db.angles.insertMany([
  { "_id": 1, "degrees": 0 },
  { "_id": 2, "degrees": 90 },
  { "_id": 3, "degrees": 180 },
  { "_id": 4, "degrees": 360 }
]);
```

クエリの例

```
db.angles.aggregate([
  { $project: {
    "radians": { $degreesToRadians: "$degrees" }
  }}
]);
```

出力

```
[
  { "_id": 1, "radians": 0 },
  { "_id": 2, "radians": 1.5707963267948966 },
  { "_id": 3, "radians": 3.141592653589793 },
  { "_id": 4, "radians": 6.283185307179586 }
]
```

コードの例

$degreesToRadians 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

$denseRank

バージョン 8.0.2 の新機能。

Amazon DocumentDB の$denseRankウィンドウ演算子は、$setWindowFieldsステージのsortBy順序に従って、パーティション内の各ドキュメントの位置を返します。同じsortBy値を持つドキュメントは同じランクを受け取り、次の個別の値は直後の位置を使用してギャップを残しません (例: 1, 2, 2, 3)。$setWindowFields ステージの outputフィールドで指定します。

これは、$setWindowFieldsステージでのみ使用されるウィンドウ演算子です。

パラメータ

- $denseRank は空のドキュメントを値{}として受け取ります。
- window フレームは受け付けません。
- $setWindowFields ステージには 1 つのsortByキーのみが必要です。

例 (MongoDB シェル)

次の例では、ドキュメントを でパーティション化しgame、各パーティションscoreを降順にソートして、各プレイヤーに高密度ランクを割り当てます。

サンプルドキュメントを作成する

```
db.players.insertMany([
  { _id: 1, game: "A", player: "Alice", score: 90 },
  { _id: 2, game: "A", player: "Bob", score: 85 },
  { _id: 3, game: "A", player: "Carol", score: 85 },
  { _id: 4, game: "A", player: "Dave", score: 70 },
  { _id: 5, game: "B", player: "Erin", score: 95 },
  { _id: 6, game: "B", player: "Frank", score: 80 }
]);
```

クエリの例

```
db.players.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$game",
      sortBy: { score: -1 },
      output: {
        denseRank: { $denseRank: {} }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "game": "A", "player": "Alice", "score": 90, "denseRank": 1 },
  { "_id": 2, "game": "A", "player": "Bob", "score": 85, "denseRank": 2 },
  { "_id": 3, "game": "A", "player": "Carol", "score": 85, "denseRank": 2 },
  { "_id": 4, "game": "A", "player": "Dave", "score": 70, "denseRank": 3 },
  { "_id": 5, "game": "B", "player": "Erin", "score": 95, "denseRank": 1 },
  { "_id": 6, "game": "B", "player": "Frank", "score": 80, "denseRank": 2 }
]
```

各gameパーティション内で、プレイヤーは降順でランク付けされますscore。Bob と Carol は で結合85し、ランク を共有し2、Dave 3はギャップなしでランク をたどります。

コードの例

$setWindowFields ステージで $denseRank演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 players = db.collection('players');

    await players.insertMany([
      { _id: 1, game: "A", player: "Alice", score: 90 },
      { _id: 2, game: "A", player: "Bob", score: 85 },
      { _id: 3, game: "A", player: "Carol", score: 85 },
      { _id: 4, game: "A", player: "Dave", score: 70 },
      { _id: 5, game: "B", player: "Erin", score: 95 },
      { _id: 6, game: "B", player: "Frank", score: 80 }
    ]);

    const result = await players.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$game",
          sortBy: { score: -1 },
          output: {
            denseRank: { $denseRank: {} }
          }
        }
      }
    ]).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']
        players = db['players']

        players.insert_many([
            { '_id': 1, 'game': 'A', 'player': 'Alice', 'score': 90 },
            { '_id': 2, 'game': 'A', 'player': 'Bob', 'score': 85 },
            { '_id': 3, 'game': 'A', 'player': 'Carol', 'score': 85 },
            { '_id': 4, 'game': 'A', 'player': 'Dave', 'score': 70 },
            { '_id': 5, 'game': 'B', 'player': 'Erin', 'score': 95 },
            { '_id': 6, 'game': 'B', 'player': 'Frank', 'score': 80 }
        ])

        result = list(players.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$game',
                    'sortBy': { 'score': -1 },
                    'output': {
                        'denseRank': { '$denseRank': {} }
                    }
                }
            }
        ]))

        print(result)
    finally:
        client.close()

example()
```

$divide

Amazon DocumentDB 集約パイプラインの $divide演算子は、ある数値を別の数値で割るために使用されます。これは、ドキュメント内の数値フィールドに対して数学演算を実行するのに便利な演算子です。

パラメータ

- numerator: 除算または分割する数。
- denominator: 除算器または除算する数値。

例 (MongoDB シェル)

この例では、 $divide演算子を使用して、年間給与と年間労働時間数に基づいて従業員の時間単価を計算する方法を示します。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.employees.aggregate([
  {
    $project: {
      name: 1,
      hourlyRate: { $divide: ["$salary", "$hoursPerYear"] }
    }
  }
])
```

出力

```
[
  { "_id" : 1, "name" : "John Doe", "hourlyRate" : 38.46153846153846 },
  { "_id" : 2, "name" : "Jane Smith", "hourlyRate" : 43.26923076923077 },
  { "_id" : 3, "name" : "Bob Johnson", "hourlyRate" : 36.05769230769231 }
]
```

コードの例

$divide コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

$documentNumber

バージョン 8.0.2 の新機能。

Amazon DocumentDB の$documentNumberウィンドウ演算子は、$setWindowFieldsステージのsortBy順序に従って、パーティション内のドキュメントごとに個別のシーケンシャル番号を返します。$rank や とは異なり$denseRank、すべてのドキュメントは一意の番号 (など1, 2, 3, 4) を受け取り、等しいsortBy値のドキュメントには発生した順序で番号が付けられます。$setWindowFields ステージの outputフィールドで指定します。

これは、$setWindowFieldsステージでのみ使用されるウィンドウ演算子です。

パラメータ

- $documentNumber は空のドキュメントを値{}として受け取ります。
- window フレームは受け付けません。
- $setWindowFields ステージには 1 つのsortByキーのみが必要です。

例 (MongoDB シェル)

次の例では、ドキュメントを でパーティション化しgame、各パーティションscoreを降順にソートして、各プレイヤーにシーケンシャルドキュメント番号を割り当てます。

サンプルドキュメントを作成する

```
db.players.insertMany([
  { _id: 1, game: "A", player: "Alice", score: 90 },
  { _id: 2, game: "A", player: "Bob", score: 85 },
  { _id: 3, game: "A", player: "Carol", score: 85 },
  { _id: 4, game: "A", player: "Dave", score: 70 },
  { _id: 5, game: "B", player: "Erin", score: 95 },
  { _id: 6, game: "B", player: "Frank", score: 80 }
]);
```

クエリの例

```
db.players.aggregate([
```

```
  {
    $setWindowFields: {
      partitionBy: "$game",
      sortBy: { score: -1 },
      output: {
        documentNumber: { $documentNumber: {} }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "game": "A", "player": "Alice", "score": 90, "documentNumber": 1 },
  { "_id": 2, "game": "A", "player": "Bob", "score": 85, "documentNumber": 2 },
  { "_id": 3, "game": "A", "player": "Carol", "score": 85, "documentNumber": 3 },
  { "_id": 4, "game": "A", "player": "Dave", "score": 70, "documentNumber": 4 },
  { "_id": 5, "game": "B", "player": "Erin", "score": 95, "documentNumber": 1 },
  { "_id": 6, "game": "B", "player": "Frank", "score": 80, "documentNumber": 2 }
]
```

各gameパーティション内で、各プレイヤーは個別の連番を降score順に受け取ります。Bob と Carol は でタイします85が2、発生した順序3で番号が付けられます。

コードの例

$setWindowFields ステージで $documentNumber演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 players = db.collection('players');

    await players.insertMany([
      { _id: 1, game: "A", player: "Alice", score: 90 },
      { _id: 2, game: "A", player: "Bob", score: 85 },
      { _id: 3, game: "A", player: "Carol", score: 85 },
      { _id: 4, game: "A", player: "Dave", score: 70 },
      { _id: 5, game: "B", player: "Erin", score: 95 },
      { _id: 6, game: "B", player: "Frank", score: 80 }
    ]);

    const result = await players.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$game",
          sortBy: { score: -1 },
          output: {
            documentNumber: { $documentNumber: {} }
          }
        }
      }
    ]).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']
        players = db['players']
```

```
        players.insert_many([
            { '_id': 1, 'game': 'A', 'player': 'Alice', 'score': 90 },
            { '_id': 2, 'game': 'A', 'player': 'Bob', 'score': 85 },
            { '_id': 3, 'game': 'A', 'player': 'Carol', 'score': 85 },
            { '_id': 4, 'game': 'A', 'player': 'Dave', 'score': 70 },
            { '_id': 5, 'game': 'B', 'player': 'Erin', 'score': 95 },
            { '_id': 6, 'game': 'B', 'player': 'Frank', 'score': 80 }
        ])

        result = list(players.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$game',
                    'sortBy': { 'score': -1 },
                    'output': {
                        'documentNumber': { '$documentNumber': {} }
                    }
                }
            }
        ]))

        print(result)
    finally:
        client.close()

example()
```

$eq

$eq 集計演算子は 2 つの値を比較し、等しいtrue場合は を返し、そうでない場合は を返しますfalse。

パラメータ

- expression1: 比較する最初の値。
- expression2: 比較する 2 番目の値。

例 (MongoDB シェル)

次の例は、 $eq演算子を使用して、製品数量がターゲット値と一致するかどうかを確認する方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.inventory.aggregate([
  {
    $project: {
      item: 1,
      qty: 1,
      target: 1,
      meetsTarget: { $eq: ["$qty", "$target"] }
    }
  }
]);
```

出力

```
[
  { _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 }
]
```

コードの例

$eq 集計演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能

Amazon DocumentDB の $exp演算子を使用すると、定数 e を特定の数値に上げることができます。

パラメータ

- expression: 評価する式。これは、フィールド参照、算術演算、その他の集計ステージなど、任意の有効な集計式にすることができます。

例 (MongoDB シェル)

次の例は、 $exp演算子を使用して、 quantityフィールドが priceフィールドより大きいすべてのドキュメントを検索する方法を示しています。

サンプルドキュメントを作成する

```
db.items.insertMany([
  { item: "canvas", quantity: 4 },
  { item: "journal", quantity: 2 }
]);
```

クエリの例

```
db.items.aggregate([
  { $project: {
    "quantityRaised": {$exp: "$quantity"}}
  }
]);
```

出力

```
[
  {
    _id: ObjectId('6920b785311cf98b79d2950d'),
    quantityRaised: 54.598150033144236
  },
  {
    _id: ObjectId('6920b785311cf98b79d2950e'),
    quantityRaised: 7.38905609893065
  }
]
```

コードの例

$exp コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

$facet

バージョン 8.0.2 の新機能。

Amazon DocumentDB の`$facet`集約ステージは、同じ入力ドキュメントセットの 1 つのステージ内で複数の集約パイプラインを処理します。各出力フィールドは独自のパイプラインを定義し、すべてのパイプラインは同じ入力ドキュメントを受け取ります。これにより、複数の個別のクエリでコレクションを実行するのではなく、カウント、グループ化、サマリー統計など、複数の独立した集計を 1 つのクエリで計算できます。

各サブパイプラインは個別に実行され、対応する出力フィールドに保存されているドキュメントの配列を生成します。サブパイプラインは状態を共有しないため、あるパイプラインの出力は別のパイプラインには影響しません。

[Syntax] (構文)

```
{
  $facet: {
    <outputField1>: [ <stage1>, <stage2>, ... ],
    <outputField2>: [ <stage1>, <stage2>, ... ],
    ...
  }
}
```

パラメータ

- outputField (必須): 関連付けられた集約パイプラインの結果配列を保持する出力フィールドの名前。1 つ以上の出力フィールドを指定できます。各出力フィールドは、入力ドキュメントに適用される集約パイプラインステージの配列にマッピングされます。

出力

$facet ステージは 1 つのドキュメントを返します。指定する各出力フィールドには、対応するサブパイプラインによって生成されたドキュメントの配列が含まれています。

制限事項

- $facet サブパイプラインには、、$changeStream、 $facet (ネストは許可されません)$collStats、$geoNear、、$indexStats、、$listSearchIndexes$merge、$out、$planCac の各ステージを含めることはできません$vectorSearch。
- $facet ステージが返すドキュメントは、16 MB の BSON ドキュメントサイズ制限内にとどまる必要があります。
- $sort ステージの前に来る$facetステージでは$first、$facet境界全体に順序を適用する必要がある場合に、、$last$push$firstN$lastN、、、、または $bucketAutoステージで順序の影響を受けやすいアキュムレータ ($group、$bucket、、、または $mergeObjects) の順序を判断できません。これは、グループ化ステージが$facetサブパイプライン内にあるか、$facetステージの後かにかかわらず適用されます。この場合、Amazon DocumentDB はパイプラインを実行する代わりにエラーを返します。このようなアキュムレータを注文するには、グループ化$sortステージの直前に、$facetサブパイプライン内にステージを配置します。、、$sum、 $min $maxなどの順序に依存しないアキュムレータ$countは影響を受けません。

例 (MongoDB シェル)

次の例は、 $facetステージを使用して、1 つのクエリで同じ売上データに対する複数の集計を計算する方法を示しています。

サンプルドキュメントを作成する

```
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: "abc", price: 25, quantity: 1, date: new Date("2021-01-01") }
]);
```

クエリの例

次のコマンドは、 $facetを使用して、1 つのステージで項目あたりの販売数の合計、全体的な価格統計、およびドキュメント数の合計を計算します。

```
db.sales.aggregate([
  {
    $facet: {
      "totalByItem": [
        { $group: { _id: "$item", totalQuantity: { $sum: "$quantity" } } }
      ],
      "priceStats": [
        { $group: { _id: null, avgPrice: { $avg: "$price" }, maxPrice: { $max:
 "$price" } } }
      ],
      "documentCount": [
        { $count: "total" }
      ]
    }
  }
])
```

出力

このコマンドは、以下の出力を返します。

```
[
  {
    totalByItem: [
      { _id: "abc", totalQuantity: 3 },
      { _id: "def", totalQuantity: 1 },
      { _id: "ghi", totalQuantity: 3 },
      { _id: "jkl", totalQuantity: 2 }
    ],
    priceStats: [
      { _id: null, avgPrice: 15, maxPrice: 25 }
    ],
```

```
    documentCount: [
      { total: 5 }
    ]
  }
]
```

コードの例

$facet コマンドを使用するためのコード例を表示するには、使用する言語のタブを選択します。

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([
    {
      $facet: {
        "totalByItem": [
          { $group: { _id: "$item", totalQuantity: { $sum: "$quantity" } } }
        ],
        "priceStats": [
          { $group: { _id: null, avgPrice: { $avg: "$price" }, maxPrice: { $max:
 "$price" } } }
        ],
        "documentCount": [
          { $count: "total" }
        ]
      }
    }
  ]).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']
    sales = db['sales']

    result = list(sales.aggregate([
        {
            '$facet': {
                'totalByItem': [
                    {'$group': {'_id': '$item', 'totalQuantity': {'$sum':
 '$quantity'}}}
                ],
                'priceStats': [
                    {'$group': {'_id': None, 'avgPrice': {'$avg': '$price'},
 'maxPrice': {'$max': '$price'}}}
                ],
                'documentCount': [
                    {'$count': 'total'}
                ]
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$filter

Amazon DocumentDB の $filter演算子は、配列の各要素にフィルター式を適用し、指定された条件に一致する要素のみを含む配列を返すために使用されます。この演算子は、ドキュメント内の配列フィールドに対して複雑なフィルタリングオペレーションを実行する必要がある場合に便利です。

パラメータ

- input: フィルタリングする配列フィールド。
- as: cond式内のinput配列の各要素に使用する変数名。
- cond: 特定の要素を出力配列に含めるかどうかを決定するブール式。

例 (MongoDB シェル)

次の例は、 $filter演算子を使用して各注文の顧客を射影し、価格が 15 より大きい商品配列の商品のみを含む新しい配列フィールド paidItems を作成する方法を示しています。基本的に、各注文の項目をフィルタリングして、コストが 15 を超える製品のみを含めます。

サンプルドキュメントを作成する

```
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 }
  ]}
]);
```

クエリの例

```
db.orders.aggregate([
  {
    $project: {
      customer: 1,
      paidItems: {
        $filter: {
          input: "$items",
          as: "item",
          cond: { $gt: ["$$item.price", 15] }
        }
      }
    }
```

```
  }
]).pretty();
```

出力

```
[
  {
    _id: 1,
    customer: 'abc123',
    paidItems: [ { name: 'Product B', price: 20, qty: 1 } ]
  },
  { _id: 2, customer: 'def456', paidItems: [] },
  { _id: 3, customer: 'ghi789', paidItems: [] }
]
```

コードの例

$filter コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 5.0 の新機能。

Elastic クラスターではサポートされていません。

Amazon DocumentDB の $first演算子は、グループ化されたドキュメントセットから最初のドキュメントを返します。集約パイプラインで一般的に使用され、特定の条件に一致する最初のドキュメントを取得します。

パラメータ

- expression: 各グループの最初の値として返される式。

例 (MongoDB シェル)

次の例は、 $first演算子を使用して、集計中に各カテゴリで発生した最初の項目値を取得する方法を示しています。

注: は、パイプライン内のドキュメントの現在の順序に基づいて最初のドキュメント$firstを返します。特定の順序 (日付、料金など) を確保するには、$sortステージの前に$groupステージを使用する必要があります。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.products.aggregate([
  { $group: { _id: "$category", firstItem: { $first: "$item" } } }
]);
```

出力

```
[
  { "_id" : "food", "firstItem" : "abc" },
  { "_id" : "toy", "firstItem" : "xyz" }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$first 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウの最初のドキュメントexpressionから の値を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

> **Note**
>
> でウィンドウ演算子として使用する場合$setWindowFields、 $firstは 100 MB の中間データに制限されます。この制限を超えるオペレーションはエラーを返します。

サンプルドキュメントを作成する

```
db.stockPrices.insertMany([
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

クエリの例

次の例では、ドキュメントを でパーティション化しticker、各パーティションを でソートしてhour、各パーティションに記録された最初の (最も早い) 料金を返します。

```
db.stockPrices.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        firstPrice: {
          $first: "$price",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "firstPrice": 50 },
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "firstPrice": 50 },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "firstPrice": 50 },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "firstPrice": 50 },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "firstPrice": 30 },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "firstPrice": 30 },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "firstPrice": 30 }
]
```

各ドキュメントはfirstPrice、ウィンドウ内の最初のドキュメント (パーティションhour内の最も早いドキュメント) の料金である で拡張されます。

コードの例

$first 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group) とウィンドウ演算子の使用 (の場合) の両方を示しています$setWindowFields。

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: first value per group
    const products = db.collection('products');
    await 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" }
    ]);
    const accumulatorResult = await products.aggregate([
      { $group: { _id: "$category", firstItem: { $first: "$item" } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Window operator usage: first value within each partition
    const stockPrices = db.collection('stockPrices');
    await stockPrices.insertMany([
      { _id: 1, ticker: "ABC", hour: 1, price: 50 },
      { _id: 2, ticker: "ABC", hour: 2, price: 45 },
      { _id: 3, ticker: "ABC", hour: 3, price: 60 },
      { _id: 4, ticker: "ABC", hour: 4, price: 40 },
      { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
      { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
      { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
    ]);
    const windowResult = await stockPrices.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$ticker",
          sortBy: { hour: 1 },
          output: {
            firstPrice: {
              $first: "$price",
```

```
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } 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']

        # Accumulator usage: first value per group
        products = db['products']
        products.insert_many([
            { '_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' }
        ])
        accumulator_result = list(products.aggregate([
            { '$group': { '_id': '$category', 'firstItem': { '$first': '$item' } } }
        ]))
        pprint(accumulator_result)
```

```
        # Window operator usage: first value within each partition
        stock_prices = db['stockPrices']
        stock_prices.insert_many([
            { '_id': 1, 'ticker': 'ABC', 'hour': 1, 'price': 50 },
            { '_id': 2, 'ticker': 'ABC', 'hour': 2, 'price': 45 },
            { '_id': 3, 'ticker': 'ABC', 'hour': 3, 'price': 60 },
            { '_id': 4, 'ticker': 'ABC', 'hour': 4, 'price': 40 },
            { '_id': 5, 'ticker': 'XYZ', 'hour': 1, 'price': 30 },
            { '_id': 6, 'ticker': 'XYZ', 'hour': 2, 'price': 35 },
            { '_id': 7, 'ticker': 'XYZ', 'hour': 3, 'price': 25 }
        ])
        window_result = list(stock_prices.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$ticker',
                    'sortBy': { 'hour': 1 },
                    'output': {
                        'firstPrice': {
                            '$first': '$price',
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        pprint(window_result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$firstN

バージョン 8.0.1 から新規。

Amazon DocumentDB の $firstN演算子は、最初の N 要素を返します。$group ステージでアキュムレータとして使用すると、各グループの最初の N 値の配列が返されます。配列式演算子として使用すると、配列の最初の N 要素が返されます。

パラメータ

- input: 値を返すフィールドまたは配列に解決される式。
- n: 返す値の数を指定する正の整数。$group ステージでアキュムレータとして使用する場合、グループ_idフィールドに基づいて正の整数に解決される限り、 は式にnすることもできます。

例 (MongoDB シェル)

次の例は、集計中に$firstNアキュムレータを使用して各項目の最初の 2 つの数量を取得する方法を示しています。

> **Note**
>
> $firstN は、ドキュメントが$groupステージに到達する順序で値を選択します。特定の順序 (日付やスコアなど) の最初の N 値を返すには、 の前に$sortステージを追加します$group。

サンプルドキュメントを作成する

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

クエリの例

```
db.sales.aggregate([
  { $group: { _id: "$item", firstTwoQuantities: { $firstN: { input: "$quantity", n:
 2 } } } }
]);
```

出力

```
[
  { "_id": "abc", "firstTwoQuantities": [10, 5] },
  { "_id": "xyz", "firstTwoQuantities": [15, 7] }
]
```

式の使用例 (MongoDB シェル)

$firstN 演算子を$projectステージ内の式として使用して、配列フィールドの最初の N 要素を返すこともできます。

サンプルドキュメントを作成する

```
db.inventory.insertMany([
  { _id: 1, item: "abc", tags: ["red", "green", "blue", "yellow", "purple"] },
  { _id: 2, item: "xyz", tags: ["alpha", "beta", "gamma"] }
]);
```

クエリの例

```
db.inventory.aggregate([
  { $project: {
      firstThreeTags: { $firstN: { input: "$tags", n: 3 } }
    }}
]);
```

出力

```
[
  { "_id": 1, "firstThreeTags": ["red", "green", "blue"] },
  { "_id": 2, "firstThreeTags": ["alpha", "beta", "gamma"] }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$firstN 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメントの最初のn値までの配列を返します。output

フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

> **Note**
>
> でウィンドウ演算子として使用する場合$setWindowFields、 $firstNは 100 MB の中間データに制限されます。この制限を超えるオペレーションはエラーを返します。

サンプルドキュメントを作成する

```
db.stockPrices.insertMany([
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

クエリの例

次の例では、ドキュメントを でパーティション化しticker、各パーティションを でソートしてhour、現在のドキュメントを通じてパーティションの先頭から見た最初の 2 つの料金を返します。

```
db.stockPrices.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        firstTwoSoFar: {
          $firstN: { input: "$price", n: 2 },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "firstTwoSoFar": [50] },
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "firstTwoSoFar": [50, 45] },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "firstTwoSoFar": [50, 45] },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "firstTwoSoFar": [50, 45] },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "firstTwoSoFar": [30] },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "firstTwoSoFar": [30, 35] },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "firstTwoSoFar": [30, 35] }
]
```

各ドキュメントはfirstTwoSoFar、ウィンドウ内の最初の 2 つのprice値である で (hour順番に) 拡張されます。

コードの例

$firstN 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group)、式の使用 (の場合$project)、ウィンドウ演算子の使用 (の場合) を示しています$setWindowFields。

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');
    await sales.insertMany([
      { item: "abc", quantity: 10, date: new Date("2023-01-01") },
      { item: "abc", quantity: 5, date: new Date("2023-01-02") },
      { item: "abc", quantity: 8, date: new Date("2023-01-03") },
      { item: "xyz", quantity: 15, date: new Date("2023-01-01") },
      { item: "xyz", quantity: 7, date: new Date("2023-01-02") },
      { item: "xyz", quantity: 3, date: new Date("2023-01-03") }
```

```
    ]);
    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');
    await inventory.insertMany([
      { _id: 1, item: "abc", tags: ["red", "green", "blue", "yellow", "purple"] },
      { _id: 2, item: "xyz", tags: ["alpha", "beta", "gamma"] }
    ]);
    const expressionResult = await inventory.aggregate([
      { $project: { firstThreeTags: { $firstN: { input: "$tags", n: 3 } } } }
    ]).toArray();
    console.log('Expression result:', expressionResult);

    // Window operator usage: first N values over a window in each partition
    const stockPrices = db.collection('stockPrices');
    await stockPrices.insertMany([
      { _id: 1, ticker: "ABC", hour: 1, price: 50 },
      { _id: 2, ticker: "ABC", hour: 2, price: 45 },
      { _id: 3, ticker: "ABC", hour: 3, price: 60 },
      { _id: 4, ticker: "ABC", hour: 4, price: 40 },
      { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
      { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
      { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
    ]);
    const windowResult = await stockPrices.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$ticker",
          sortBy: { hour: 1 },
          output: {
            firstTwoSoFar: {
              $firstN: { input: "$price", n: 2 },
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);
```

```
  } finally {
    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')

    try:
        db = client['test']

        # Accumulator usage: first N values per group
        sales = db['sales']
        sales.insert_many([
            { 'item': 'abc', 'quantity': 10, 'date': datetime(2023, 1, 1) },
            { 'item': 'abc', 'quantity': 5, 'date': datetime(2023, 1, 2) },
            { 'item': 'abc', 'quantity': 8, 'date': datetime(2023, 1, 3) },
            { 'item': 'xyz', 'quantity': 15, 'date': datetime(2023, 1, 1) },
            { 'item': 'xyz', 'quantity': 7, 'date': datetime(2023, 1, 2) },
            { 'item': 'xyz', 'quantity': 3, 'date': datetime(2023, 1, 3) }
        ])
        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']
        inventory.insert_many([
            { '_id': 1, 'item': 'abc', 'tags': ['red', 'green', 'blue', 'yellow',
 'purple'] },
            { '_id': 2, 'item': 'xyz', 'tags': ['alpha', 'beta', 'gamma'] }
```

```
        ])
        expression_result = list(inventory.aggregate([
            { '$project': { 'firstThreeTags': { '$firstN': { 'input': '$tags', 'n':
 3 } } } }
        ]))
        print('Expression result:', expression_result)

        # Window operator usage: first N values over a window in each partition
        stock_prices = db['stockPrices']
        stock_prices.insert_many([
            { '_id': 1, 'ticker': 'ABC', 'hour': 1, 'price': 50 },
            { '_id': 2, 'ticker': 'ABC', 'hour': 2, 'price': 45 },
            { '_id': 3, 'ticker': 'ABC', 'hour': 3, 'price': 60 },
            { '_id': 4, 'ticker': 'ABC', 'hour': 4, 'price': 40 },
            { '_id': 5, 'ticker': 'XYZ', 'hour': 1, 'price': 30 },
            { '_id': 6, 'ticker': 'XYZ', 'hour': 2, 'price': 35 },
            { '_id': 7, 'ticker': 'XYZ', 'hour': 3, 'price': 25 }
        ])
        window_result = list(stock_prices.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$ticker',
                    'sortBy': { 'hour': 1 },
                    'output': {
                        'firstTwoSoFar': {
                            '$firstN': { 'input': '$price', 'n': 2 },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()

example()
```

$floor

バージョン 4.0 の新機能。

Amazon DocumentDB の $floor演算子は、指定された数以下の最大の整数を返します。この演算子は、数値を四捨五入するのに役立ちます。

パラメータ

- expression: 切り捨てる数値式。

例 (MongoDB シェル)

次の例は、 $floor演算子を使用して 10 進値を最も近い整数に切り下げる方法を示しています。

サンプルドキュメントを作成する

```
db.numbers.insertOne({ value: 3.14 });
```

クエリの例

```
db.numbers.aggregate([
  { $project: { _id: 0, floored: { $floor: "$value" } } }
]);
```

出力

```
{ "floored" : 3 }
```

コードの例

$floor コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 集約ステージは、指定されたポイントに近接した順序でドキュメントを返します。ポイントからの距離を計算し、出力ドキュメントに距離を含めます。

パラメータ

- near: GeoJSON またはレガシー座標として指定された距離を計算するポイント。
- distanceField: 計算された距離を保存するフィールド名。
- spherical: 球形ジオメトリを使用するかどうかを示すブール値 (GeoJSON ポイントに必要) 。
- maxDistance: オプション。中心点からの最大距離。
- minDistance: オプション。中心点からの最小距離。
- query: オプション。適用する追加のフィルター条件。
- limit: オプション。返されるドキュメントの最大数。
- key: オプション。複数の地理空間インデックスが存在する場合に地理空間クエリに使用するフィールド。

例 (MongoDB シェル)

次の例は、 $geoNearステージを使用して、特定の場所に最も近いストアを検索する方法を示しています。

サンプルドキュメントを作成する

```
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] } }
```

```
]);
```

クエリの例

```
db.stores.aggregate([
  {
    $geoNear: {
      near: { type: "Point", coordinates: [-122.4, 37.8] },
      distanceField: "distance",
      spherical: true
    }
  }
]);
```

出力

```
[
  { _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 }
]
```

コードの例

$geoNear 集約ステージを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

$graphLookup

バージョン 8.0.2 の新機能。

Amazon DocumentDB の$graphLookup集約ステージは、コレクションに対して再帰的な検索を実行します。各入力ドキュメントについて、 は 1 つ以上の値から$graphLookup開始し、接続フィールドをターゲットフィールドに繰り返し一致させ、一致するドキュメントのチェーンに従って、一致するドキュメントがなくなるか、オプションの再帰深度に達します。これは、組織のレポート構造、カテゴリツリー、ネットワーク関係など、グラフのようなデータや階層データをトラバースする場合に便利です。

パラメータ

- from: 再帰的に検索するターゲットコレクション。from コレクションは同じデータベースに存在する必要があります。
- startWith: 再帰検索を開始する値を指定する式。この値は、fromコレクション内のドキュメントconnectToFieldの と比較されます。が配列にstartWith評価される場合、検索はすべての配列要素から同時に実行されます。
- connectFromField: 値が再帰検索の次のドキュメントへの接続に使用されるフィールド名。値が配列の場合、各要素はトラバーサルを通じて個別にフォローされます。
- connectToField: の値と一致するfromコレクション内のフィールド名connectFromField。
- as: 各出力ドキュメントに追加された配列フィールドの名前。このフィールドには、再帰検索中に見つかったドキュメントが含まれます。これらのドキュメントの順序については、次の動作セクションの結果の順序付け項目を参照してください。
- maxDepth: 最初のルックアップ後に実行する再帰的反復の最大数を指定する負以外の整数。このパラメータはオプションです。の値は、最初の一致のみ0を実行し、再帰は実行しません。を省略するとmaxDepth、一致するドキュメントがなくなるまで検索が再帰的に続行されます。
- depthField: ドキュメントが見つかった再帰深度を保持するas配列内の各ドキュメントに追加するフィールドの名前。このパラメータはオプションです。深さは として表NumberLongされ、ゼロから始まるため、最初のルックアップは の深さに対応します0。
- restrictSearchWithMatch: 検索のドキュメントに適用される追加の一致条件を指定するドキュメント。このパラメータはオプションです。条件は$matchステージと同じクエリ構文を使用し、集計式を含めることはできません。

> **Note**
>
> $graphLookup ステージには 100 MB のメモリ制限が適用され、 はサポートされていませんallowDiskUse:true。この制限を超えるオペレーションはエラーを返します。集約に他

のステージが含まれている場合、allowDiskUse:trueオプションはそれらのステージに対して有効です。

Behavior

- クロスコレクショントラバーサル: 検索するコレクションfromに設定します。これは、パイプラインが実行するのと同じコレクションでも、同じデータベース内の別のコレクションでもかまいません。
- 結果の順序: as配列内のドキュメントは、保証された順序で返されません。

例 (MongoDB シェル)

次の例では$graphLookup、 を使用して従業員レポート階層を横断し、各従業員について、その上のマネージャーチェーン全体を返します。

次のコマンドは、mongosh シェルで実行し、レポート階層を確立するサンプル従業員ドキュメントを挿入します。

```
db.employees.insertMany([
  { _id: 1, name: "Dev", reportsTo: "Eliot" },
  { _id: 2, name: "Eliot", reportsTo: "Ron" },
  { _id: 3, name: "Ron", reportsTo: "Andrew" },
  { _id: 4, name: "Andrew", reportsTo: "Asya" },
  { _id: 5, name: "Asya" }
]);
```

次の集計は、新しいreportingHierarchy配列の各従業員の完全な管理チェーンを返します。

```
db.employees.aggregate([
  {
    $graphLookup: {
      from: "employees",
      startWith: "$reportsTo",
      connectFromField: "reportsTo",
      connectToField: "name",
      as: "reportingHierarchy",
      depthField: "level"
    }
  }
```

```
]);
```

このオペレーションは、次の出力を返します。

```
[
  {
    _id: 1,
    name: 'Dev',
    reportsTo: 'Eliot',
    reportingHierarchy: [
      { _id: 2, name: 'Eliot', reportsTo: 'Ron', level: 0 },
      { _id: 3, name: 'Ron', reportsTo: 'Andrew', level: 1 },
      { _id: 4, name: 'Andrew', reportsTo: 'Asya', level: 2 },
      { _id: 5, name: 'Asya', level: 3 }
    ]
  },
  {
    _id: 2,
    name: 'Eliot',
    reportsTo: 'Ron',
    reportingHierarchy: [
      { _id: 3, name: 'Ron', reportsTo: 'Andrew', level: 0 },
      { _id: 4, name: 'Andrew', reportsTo: 'Asya', level: 1 },
      { _id: 5, name: 'Asya', level: 2 }
    ]
  },
  {
    _id: 3,
    name: 'Ron',
    reportsTo: 'Andrew',
    reportingHierarchy: [
      { _id: 4, name: 'Andrew', reportsTo: 'Asya', level: 0 },
      { _id: 5, name: 'Asya', level: 1 }
    ]
  },
  {
    _id: 4,
    name: 'Andrew',
    reportsTo: 'Asya',
    reportingHierarchy: [
      { _id: 5, name: 'Asya', level: 0 }
    ]
  },
```

```
  {
    _id: 5,
    name: 'Asya',
    reportingHierarchy: []
  }
]
```

各従業員ドキュメントは、再帰的に一致したマネージャードキュメントを含むreportingHierarchy配列を取得します。level フィールドには、各マネージャーが見つかった再帰の深さが記録されます。Asya にはreportsTo値がないため、reportingHierarchy配列は空です。

maxDepth と restrictSearchWithMatch の例 (MongoDB シェル)

次の例では、人と の間の接続のネットワークを横断し、 maxDepth を使用して再帰を制限しrestrictSearchWithMatch、追加の条件に一致するドキュメントのみを含めます。検索では、開始者の接続が最大 2 レベルまで検索され、hobbies配列に が含まれているユーザーのみが保持されますgolf。

次のコマンドを mongosh シェルで実行し、各ユーザーのfriendsポイントが 1 方向の接続チェーンを通過するサンプルドキュメントを挿入します。

```
db.people.insertMany([
  { _id: 1, name: "Tanya", friends: [ "Carole", "Shirley" ], hobbies: [ "tennis",
 "golf" ] },
  { _id: 2, name: "Carole", friends: [ "Joseph" ], hobbies: [ "archery", "golf" ] },
  { _id: 3, name: "Joseph", friends: [ "Angelo" ], hobbies: [ "tennis", "golf" ] },
  { _id: 4, name: "Angelo", friends: [ "Miguel" ], hobbies: [ "travel", "golf" ] },
  { _id: 5, name: "Miguel", friends: [], hobbies: [ "golf" ] },
  { _id: 6, name: "Shirley", friends: [], hobbies: [ "frisbee" ] }
]);
```

次の集約は、Tanya から始まり、接続のネットワークを最大 2 レベル深く横断し、趣味golfとしてリストされている人だけを保持します。

```
db.people.aggregate([
  { $match: { name: "Tanya" } },
  {
    $graphLookup: {
      from: "people",
      startWith: "$friends",
```

```
      connectFromField: "friends",
      connectToField: "name",
      as: "golfConnections",
      maxDepth: 2,
      restrictSearchWithMatch: { hobbies: "golf" }
    }
  },
  { $project: { name: 1, "golfConnections.name": 1 } }
]);
```

このオペレーションは、次の出力を返します。

```
[
  {
    _id: 1,
    name: 'Tanya',
    golfConnections: [
      { name: 'Carole' },
      { name: 'Joseph' },
      { name: 'Angelo' }
    ]
  }
]
```

hobbies 配列を含む接続のみgolfが含まれ、2 レベルの再帰後にトラバーサルが停止します。Shirley は を一覧表示しないため除外されgolf、Miguel は maxDepthの を超えて Tanya から 3 レベル離れているため除外されます2。接続は Tanya を指さない一方向チェーンを形成するため、再帰的な検索中に開始ドキュメントは一致しません。

コードの例

MongoDB ドライバーを使用して最初の例から従業員レポート階層を構築するコード例を表示するには、使用する言語のタブを選択します。

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 employees = db.collection('employees');

    const result = await employees.aggregate([
      {
        $graphLookup: {
          from: 'employees',
          startWith: '$reportsTo',
          connectFromField: 'reportsTo',
          connectToField: 'name',
          as: 'reportingHierarchy',
          depthField: 'level'
        }
      }
    ]).toArray();

    console.log(JSON.stringify(result, null, 2));
  } catch (err) {
    console.error('Failed to run $graphLookup aggregation:', err.message);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pymongo.errors import PyMongoError

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']
        employees = db['employees']
```

```
        pipeline = [
            {
                "$graphLookup": {
                    "from": "employees",
                    "startWith": "$reportsTo",
                    "connectFromField": "reportsTo",
                    "connectToField": "name",
                    "as": "reportingHierarchy",
                    "depthField": "level"
                }
            }
        ]

        result = list(employees.aggregate(pipeline))

        for doc in result:
            print(doc)
    except PyMongoError as error:
        print(f"Failed to run $graphLookup aggregation: {error}")
    finally:
        client.close()

example()
```

$group

Amazon DocumentDB の$group集約ステージでは、指定された式でドキュメントをグループ化し、グループ化されたデータに対してさまざまな累積オペレーションを実行できます。これは、グループ化されたデータに基づいて合計、平均、またはその他の統計を計算するなどのタスクに役立ちます。

パラメータ

- _id: 入力ドキュメントをグループ化する式を指定します。これは、フィールド名、計算式、またはその両方の組み合わせです。
- accumulator expressions: (オプション) グループ化されたデータに適用する必要がある 1 つ以上のアキュムレータ式。これらの式は、上記のアキュムレータ演算子を使用します。

例 (MongoDB シェル)

次の例では、顧客を都市別にグループ化し、各都市の合計注文額を計算します。

サンプルドキュメントを作成する

```
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 }] }
]);
```

クエリの例

```
db.customers.aggregate([
  {
    $group: {
      _id: "$city",
      totalOrders: { $sum: { $sum: "$orders.amount" } }
    }
  }
]);
```

出力

```
[
  { _id: 'Chicago', totalOrders: 150 },
  { _id: 'Los Angeles', totalOrders: 450 },
  { _id: 'New York', totalOrders: 500 }
]
```

コードの例

$group コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 集計演算子は 2 つの値を比較し、最初の値が 2 番目より大きいtrue場合は を返し、それ以外の場合は を返しますfalse。

パラメータ

- expression1: 比較する最初の値。
- expression2: 比較する 2 番目の値。

例 (MongoDB シェル)

次の例は、 $gt演算子を使用して、価格しきい値を超える製品を識別する方法を示しています。

サンプルドキュメントを作成する

```
db.products.insertMany([
  { _id: 1, name: "Laptop", price: 1200 },
  { _id: 2, name: "Mouse", price: 25 },
  { _id: 3, name: "Monitor", price: 400 }
]);
```

クエリの例

```
db.products.aggregate([
  {
    $project: {
      name: 1,
      price: 1,
      expensive: { $gt: ["$price", 100] }
    }
```

```
  }
]);
```

出力

```
[
  { _id: 1, name: 'Laptop', price: 1200, expensive: true },
  { _id: 2, name: 'Mouse', price: 25, expensive: false },
  { _id: 3, name: 'Monitor', price: 400, expensive: true }
]
```

コードの例

$gt 集計演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 集計演算子は 2 つの値を比較し、最初の値が 2 番目以上のtrue場合は を返し、それ以外の場合は を返しますfalse。

パラメータ

- expression1: 比較する最初の値。
- expression2: 比較する 2 番目の値。

例 (MongoDB シェル)

次の例は、 $gte演算子を使用して、学生が試験に合格したかどうかを確認する方法を示しています。

サンプルドキュメントを作成する

```
db.students.insertMany([
  { _id: 1, name: "Alice", score: 85 },
  { _id: 2, name: "Bob", score: 60 },
  { _id: 3, name: "Charlie", score: 72 }
]);
```

クエリの例

```
db.students.aggregate([
  {
    $project: {
      name: 1,
      score: 1,
      passed: { $gte: ["$score", 70] }
    }
  }
]);
```

出力

```
[
  { _id: 1, name: 'Alice', score: 85, passed: true },
  { _id: 2, name: 'Bob', score: 60, passed: false },
  { _id: 3, name: 'Charlie', score: 72, passed: true }
]
```

コードの例

$gte 集計演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、日付またはタイムスタンプフィールドから時間コンポーネントを抽出します。

パラメータ

- dateExpression: 演算子が適用される日付。これは有効な BSON 日付 (例: $createdAt のようなフィールドや日付リテラル) に解決する必要があります。

パラメータは、次の形式でドキュメントとして指定することもできます。

{ 日付: <dateExpression>、タイムゾーン: <timezoneExpression> }

これにより、 はタイムゾーン対応の日付オペレーションを適用できます。

```
- `<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.
```

例 (MongoDB シェル)

次の例は、 $hour演算子を使用して日付フィールドから時間コンポーネントを抽出し、それに応じてデータをグループ化する方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.events.aggregate([
  {
    $project: {
      hour: { $hour: "$timestamp" }
    }
```

```
  },
  {
    $group: {
      _id: "$hour",
      count: { $sum: 1 }
    }
  },
  {
    $sort: { _id: 1 }
  }
]);
```

出力

```
[
  { "_id": 8, "count": 1 },
  { "_id": 10, "count": 1 },
  { "_id": 12, "count": 1 },
  { "_id": 16, "count": 1 },
  { "_id": 23, "count": 1 }
]
```

このクエリは、 timestampフィールドの時間コンポーネントでイベントをグループ化し、1 時間あたりのイベント数をカウントします。

コードの例

$hour コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

$ifNull 演算子は、入力式が null または未定義と評価された場合に、指定された値を返すために使用されます。この演算子は、デフォルト値を指定したり、null/undefined ケースを処理したりする場合に役立ちます。

パラメータ

- expression: 評価する式。
- replacement: が null または未定義と<expression>評価された場合に返される値。

例 (MongoDB シェル)

次の例は、 name フィールドが null または未定義の場合に、 $ifNull演算子を使用してデフォルト値を指定する方法を示しています。

サンプルドキュメントを作成する

```
db.users.insertMany([
  { _id: 1, name: "John" },
  { _id: 2, name: null },
  { _id: 3 }
]);
```

クエリの例

```
db.users.aggregate([
  {
    $project: {
      _id: 1,
      name: { $ifNull: ["$name", "No Name"] }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "name": "John" },
  { "_id": 2, "name": "No Name" },
  { "_id": 3, "name": "No Name" }
]
```

コードの例

$ifNull コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 集計演算子は、指定された値が配列内に存在するかどうかを確認します。配列で値が見つかったtrue場合は が返され、falseそれ以外の場合は が返されます。

パラメータ

- value: 検索する値。
- array: 検索する配列。

例 (MongoDB シェル)

次の例では、 $in演算子を使用して、各従業員のスキルセットに特定のスキルが存在するかどうかを確認します。

サンプルドキュメントを作成する

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

クエリの例

```
db.employees.aggregate([
  {
    $project: {
      name: 1,
```

```
      hasPython: { $in: ["Python", "$skills"] }
    }
  }
]);
```

出力

```
[
  { _id: 1, name: 'Sarah', hasPython: true },
  { _id: 2, name: 'Mike', hasPython: false },
  { _id: 3, name: 'Emma', hasPython: true }
]
```

コードの例

$in 集計演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $indexOfArray演算子は、配列内の指定された要素の最初の出現のインデックスを見つけるために使用されます。この演算子は、指定された値に一致する配列の最初の要素のゼロベースのインデックス位置を返します。値が見つからない場合は、-1 を返します。

パラメータ

- array: 検索する配列。
- value: 配列内で検索する値。
- start: (オプション) 検索を開始する配列内の位置。デフォルト値は 0 です。

例 (MongoDB シェル)

次の例は、$indexOfArray 演算子を使用して、各ドキュメントの「fruits」配列で要素「mango」の最初の出現のインデックスを検索する方法を示しています。

サンプルドキュメントを作成する

```
db.collection.insertMany([
  { _id: 1, fruits: ["apple", "banana", "cherry", "durian"] },
  { _id: 2, fruits: ["mango", "orange", "pineapple"] },
  { _id: 3, fruits: ["kiwi", "lemon", "mango"] }
]);
```

クエリの例

```
db.collection.aggregate([
  {
    $project: {
      _id: 1,
      fruitIndex: { $indexOfArray: ["$fruits", "mango"] }
    }
  }
]);
```

出力

```
{ "_id" : 1, "fruitIndex" : 1 }
{ "_id" : 2, "fruitIndex" : 0 }
{ "_id" : 3, "fruitIndex" : 2 }
```

コードの例

$indexOfArray コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $indexOfBytes 演算子は、文字列内の部分文字列の開始インデックスを、文字のバイト位置に基づいて検索するために使用されます。これは、Latin 以外のスクリプトなど、マルチバイト文字を含むテキストデータを使用する場合に便利です。

パラメータ

- `string`: 検索する入力文字列。
- `substring`: 入力文字列内で検索する部分文字列。
- `[<start>]`: (オプション) 検索の開始位置 (ゼロベース) 。指定しない場合、検索は文字列の先頭から始まります。

例 (MongoDB シェル)

次の例は、 を使用して、デスクの場所を表す文字列のセット内の最初のハイフン文字のインデックス`$indexOfBytes`を検索する方法を示しています。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.people.aggregate([
  { $project: { stateLocation: { $indexOfBytes: ["$Desk", "-"] } } }
]);
```

出力

```
{ "_id" : 1, "stateLocation" : 11 }
```

```
{ "_id" : 2, "stateLocation" : 6 }
{ "_id" : 3, "stateLocation" : 7 }
{ "_id" : 4, "stateLocation" : 8 }
```

コードの例

$indexOfBytes コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $indexOfCP演算子は、文字列式内で指定された部分文字列が最初に出現したときのインデックスをコードポイント (CP) で検索するために使用されます。これは、文字列フィールドからコンテンツを解析および抽出する場合に便利です。

パラメータ

- string expression: 検索する文字列。
- substring: 検索する部分文字列。
- [<start>]: (オプション) 検索を開始する位置 (ゼロベースのインデックス) 。デフォルトは 0 です。

例 (MongoDB シェル)

この例では、 $indexOfCP演算子を使用して、各ドキュメントの Desk フィールドでハイフン文字 - の最初の出現のインデックスを検索します。

サンプルドキュメントを作成する

```
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"}
])
```

クエリの例

```
db.people.aggregate([
  { $project: { stateLocation: { $indexOfCP: [ "$Desk", "-"] } } }
```

```
])
```

出力

```
{ "_id" : 1, "stateLocation" : 10 }
{ "_id" : 2, "stateLocation" : 6 }
{ "_id" : 3, "stateLocation" : 7 }
{ "_id" : 4, "stateLocation" : 8 }
```

コードの例

$indexOfCP コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$indexStats集約ステージは、コレクション内のインデックスの使用に関するインサイトを提供します。この演算子を使用すると、インデックスのアクセスパターンをモニタリングできるため、インデックスの管理と最適化について情報に基づいた意思決定を行うことができます。

パラメータ

なし

例 (MongoDB シェル)

次の例は、 $indexStats演算子を使用して Amazon DocumentDB コレクションのインデックス使用状況を分析する方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.grocery.aggregate([
  { $indexStats: {} }
]);
```

出力

```
[
  {
    "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")
    }
  }
]
```

この例では、 $indexStats演算子は、_id_インデックスが 5 回アクセスされ、product_1インデックスが前回のリセットまたはサーバー再起動から 10 回アクセスされたことを示しています。

コードの例

$indexStats コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $isArray演算子は、ドキュメント内のフィールドが配列であるかどうかをチェックするために使用されます。この演算子は、配列タイプのフィールドを処理する集約パイプラインや条件式に役立ちます。

パラメータ

- `field`: 配列であるかどうかをチェックするフィールドパス。

例 (MongoDB シェル)

この例では、 $isArray演算子を使用して、「インベントリ」フィールドが配列であるドキュメントを識別する方法を示します。

サンプルドキュメントを作成する

```
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}]}
]);
```

クエリの例

```
db.videos.aggregate([
  {
    $match: {
      $isArray: "$inventory"
    }
  },
  {
    $project: {
      _id: 1,
      name: 1,
      "inventory.location": 1,
      "inventory.count": 1
    }
  }
]).pretty();
```

出力

```
{
  "_id": 4,
  "name": "Bravemind",
  "inventory": [
    {
      "location": "Mason City",
      "count": 250
    },
    {
      "location": "Des Moines",
      "count": 1000
```

```
    },
    {
      "location": "Ames",
      "count": 500
    }
  ]
}
```

コードの例

$isArray コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の$isNumber演算子は、特定の式が数値型 (整数、10 進数、倍精度、長精度) に解決されるかどうかを示すブール値を返します。配列内のすべての要素が数値であっても、配列は数値と見なされないことに注意してください。たとえば、 値$isNumberに適用されると、 [1, 2, 3]が返されますfalse。

パラメータ

- expression: 数値型に解決されるかどうかを確認する任意の式。

例 (MongoDB シェル)

次の例は、 $isNumber演算子を使用してフィールド値が数値であるかどうかをチェックする方法を示しています。

サンプルドキュメントを作成する

```
db.data.insertMany([
  {_id: 1, value: 42},
  {_id: 2, value: "hello"},
  {_id: 3, value: 3.14},
  {_id: 4, value: null}
]);
```

クエリの例

```
db.data.aggregate([
  { $project: { isNum: { $isNumber: "$value" } } }
]);
```

出力

```
[
  {_id: 1, isNum: true},
  {_id: 2, isNum: false},
  {_id: 3, isNum: true},
  {_id: 4, isNum: false}
]
```

コードの例

$isNumber 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $isoDayOfWeek演算子は、日付の ISO 曜日を整数値として返します。ISO 週の日付システムは、毎週月曜日から日曜日までを定義し、第 1 週は年の最初の木曜日を含む週です。

パラメータ

- expression: ISO 曜日を返す日付式。

例 (MongoDB シェル)

次の例は、 $isoDayOfWeek演算子を使用して、一連のイベントドキュメントの ISO 曜日を取得する方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.events.aggregate([
  { $project: {
    _id: 1,
    eventDate: 1,
    isoDayOfWeek: { $isoDayOfWeek: "$eventDate" }
  }}
]);
```

出力

```
[
  { "_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 }
```

```
]
```

コードの例

$isoDayOfWeek コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$isoWeek演算子は、日付の ISO 週番号を返します。ISO 週の日付システムは、1 年の週を番号付けする方法であり、新しい年の最初の週は、その年の最初の木曜日を含む週です。これは、新年の最初の週が 1 月 1 日を含む週であるグレゴリオ暦とは異なります。

パラメータ

なし

例 (MongoDB シェル)

次の例は、 $isoWeek演算子を使用して特定の日付の ISO 週番号を取得する方法を示しています。

サンプルドキュメントを作成する

```
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") }
])
```

クエリの例

```
db.dates.aggregate([
```

```
  {
    $project: {
      _id: 1,
      isoWeek: { $isoWeek: "$date" }
    }
  }
])
```

出力

```
[
  { "_id": 1, "isoWeek": 52 },
  { "_id": 2, "isoWeek": 52 },
  { "_id": 3, "isoWeek": 1 }
]
```

コードの例

$isoWeek コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の`$isoWeekYear`演算子は、特定の日付の ISO 8601 週の年番号を返します。ISO の週の年番号は、特に年の開始時と終了時に、週の年が暦年と異なる場合があるという点で、グレゴリオ暦年とは異なります。

パラメータ

- `expression`: ISO 8601 週の年番号を返す日付式。

例 (MongoDB シェル)

この例では、 $isoWeekYear演算子を使用して、イベントコレクション内の各ドキュメントの日付フィールドの ISO 8601 週の年を取得する方法を示します。

サンプルドキュメントを作成する

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

クエリの例

```
db.events.aggregate([
  { $project: {
    name: 1,
    isoWeekYear: { $isoWeekYear: "$date" }
  }}
]);
```

出力

```
[
  { "_id" : 1, "name" : "Event 1", "isoWeekYear" : 2023 },
  { "_id" : 2, "name" : "Event 2", "isoWeekYear" : 2023 },
  { "_id" : 3, "name" : "Event 3", "isoWeekYear" : 2023 }
]
```

コードの例

$isoWeekYear コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $last演算子は、クエリ条件に一致する配列の最後の要素を返すために使用されます。これは、特定の条件を満たす配列の最新の要素または最後の要素を取得する場合に特に便利です。

パラメータ

- expression: 配列要素に一致する式。

例 (MongoDB シェル)

次の例は、 演算子を $lastと組み合わせて使用$filterして、特定の条件を満たす配列から最後の要素を取得する方法を示しています (例: subject is 'science')。

サンプルドキュメントを作成する

```
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 }
```

```
    ]
  }
]);
```

クエリの例

```
db.collection.aggregate([
    { $match: { name: "John" } },
    {
      $project: {
        name: 1,
        lastScienceScore: {
          $last: {
            $filter: {
              input: "$scores",
              as: "score",
              cond: { $eq: ["$$score.subject", "science"] }
            }
          }
        }
      }
    }
  ]);
```

出力

```
[
  {
    _id: 1,
    name: 'John',
    lastScienceScore: { subject: 'science', score: 90 }
  }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$last 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウの最後のドキュメントexpressionの の値を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

サンプルドキュメントを作成する

```
db.stockPrices.insertMany([
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

クエリの例

次の例では、ドキュメントを でパーティション化しticker、各パーティションを でソートしてhour、各パーティションに記録された最後の (最新の) 料金を返します。

```
db.stockPrices.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        lastPrice: {
          $last: "$price",
          window: { documents: ["unbounded", "unbounded"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "lastPrice": 40 },
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "lastPrice": 40 },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "lastPrice": 40 },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "lastPrice": 40 },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "lastPrice": 25 },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "lastPrice": 25 },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "lastPrice": 25 }
```

```
]
```

各ドキュメントはlastPrice、ウィンドウ内の最後のドキュメント (パーティションhour内の最新のドキュメント) の料金である で拡張されます。

コードの例

$last 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、式の使用 (の場合$project) とウィンドウ演算子の使用 (の場合) の両方を示しています$setWindowFields。

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');

  // Expression usage: last element of a filtered array
  const collection = db.collection('collection');
  await 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 }
      ]
    }
  ]);
  const expressionResult = await collection.aggregate([
    { $match: { name: "John" } },
    {
      $project: {
        name: 1,
        lastScienceScore: {
          $last: {
            $filter: {
              input: "$scores",
              as: "score",
              cond: { $eq: ["$$score.subject", "science"] }
            }
          }
        }
      }
    }
  ]).toArray();
  console.log('Expression result:', JSON.stringify(expressionResult, null, 2));

  // Window operator usage: last value within each partition
  const stockPrices = db.collection('stockPrices');
  await stockPrices.insertMany([
    { _id: 1, ticker: "ABC", hour: 1, price: 50 },
    { _id: 2, ticker: "ABC", hour: 2, price: 45 },
    { _id: 3, ticker: "ABC", hour: 3, price: 60 },
    { _id: 4, ticker: "ABC", hour: 4, price: 40 },
    { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
    { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
    { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
  ]);
  const windowResult = await stockPrices.aggregate([
    {
      $setWindowFields: {
        partitionBy: "$ticker",
```

```
        sortBy: { hour: 1 },
        output: {
          lastPrice: {
            $last: "$price",
            window: { documents: ["unbounded", "unbounded"] }
          }
        }
      }
    }
  ]).toArray();
  console.log('Window result:', JSON.stringify(windowResult, 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

    # Expression usage: last element of a filtered array
    collection = db.collection
    collection.insert_many([
        {
            "_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 }
            ]
        }
    ])
    expression_pipeline = [
        { "$match": { "name": "John" } },
        {
            "$project": {
                "name": 1,
                "lastScienceScore": {
                    "$last": {
                        "$filter": {
                            "input": "$scores",
                            "as": "score",
                            "cond": { "$eq": ["$$score.subject", "science"] }
                        }
                    }
                }
            }
        }
    ]
    expression_result = list(collection.aggregate(expression_pipeline))
    print('Expression result:', expression_result)

    # Window operator usage: last value within each partition
    stock_prices = db.stockPrices
    stock_prices.insert_many([
        { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50 },
        { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45 },
        { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60 },
        { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40 },
```

```
        { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30 },
        { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35 },
        { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25 }
    ])
    window_pipeline = [
        {
            "$setWindowFields": {
                "partitionBy": "$ticker",
                "sortBy": { "hour": 1 },
                "output": {
                    "lastPrice": {
                        "$last": "$price",
                        "window": { "documents": ["unbounded", "unbounded"] }
                    }
                }
            }
        }
    ]
    window_result = list(stock_prices.aggregate(window_pipeline))
    print('Window result:', window_result)

    client.close()

example()
```

$lastN

バージョン 8.0.1 から新規。

Amazon DocumentDB の $lastN演算子は、最後の N 要素を返します。$group ステージでアキュムレータとして使用すると、各グループの最後の N 値の配列が返されます。配列式演算子として使用すると、配列の最後の N 要素が返されます。

パラメータ

- input: 値を返すフィールドまたは配列に解決される式。
- n: 返す値の数を指定する正の整数。$group ステージでアキュムレータとして使用する場合、グループ_idフィールドに基づいて正の整数に解決される限り、 は式にnすることもできます。

例 (MongoDB シェル)

次の例は、集計中に$lastNアキュムレータを使用して各項目の最後の 2 つの数量を取得する方法を示しています。

> **Note**
>
> $lastN は、ドキュメントが$groupステージに到達する順序で値を選択します。特定の順序 (日付やスコアなど) の最後の N 値を返すには、 の前に$sortステージを追加します$group。

サンプルドキュメントを作成する

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

クエリの例

```
db.sales.aggregate([
  { $group: { _id: "$item", lastTwoQuantities: { $lastN: { input: "$quantity", n:
 2 } } } }
]);
```

出力

```
[
  { "_id": "abc", "lastTwoQuantities": [5, 8] },
  { "_id": "xyz", "lastTwoQuantities": [7, 3] }
]
```

式の使用例 (MongoDB シェル)

$lastN 演算子を$projectステージ内の式として使用して、配列フィールドの最後の N 要素を返すこともできます。

サンプルドキュメントを作成する

```
db.inventory.insertMany([
  { _id: 1, item: "abc", tags: ["red", "green", "blue", "yellow", "purple"] },
  { _id: 2, item: "xyz", tags: ["alpha", "beta", "gamma"] }
]);
```

クエリの例

```
db.inventory.aggregate([
  { $project: {
      lastThreeTags: { $lastN: { input: "$tags", n: 3 } }
    }}
]);
```

出力

```
[
  { "_id": 1, "lastThreeTags": ["blue", "yellow", "purple"] },
  { "_id": 2, "lastThreeTags": ["alpha", "beta", "gamma"] }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$lastN 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメントの最後のn値までの配列を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

> **Note**
>
> でウィンドウ演算子として使用する場合$setWindowFields、 $lastNは 100 MB の中間データに制限されます。この制限を超えるオペレーションはエラーを返します。

サンプルドキュメントを作成する

```
db.stockPrices.insertMany([
```

```
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

クエリの例

次の例では、ドキュメントを でパーティション化しticker、各パーティションを でソートしてhour、パーティションの先頭から現在のドキュメントに表示された最新の 2 つの料金を返します。

```
db.stockPrices.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        lastTwoSoFar: {
          $lastN: { input: "$price", n: 2 },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "lastTwoSoFar": [50] },
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "lastTwoSoFar": [50, 45] },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "lastTwoSoFar": [45, 60] },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "lastTwoSoFar": [60, 40] },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "lastTwoSoFar": [30] },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "lastTwoSoFar": [30, 35] },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "lastTwoSoFar": [35, 25] }
]
```

各ドキュメントはlastTwoSoFar、ウィンドウ内の最後の 2 つのprice値である で (hour順番に) 拡張されます。

コードの例

$lastN 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group)、式の使用 (の場合$project)、ウィンドウ演算子の使用 (の場合) を示しています$setWindowFields。

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');
    await sales.insertMany([
      { item: "abc", quantity: 10, date: new Date("2023-01-01") },
      { item: "abc", quantity: 5, date: new Date("2023-01-02") },
      { item: "abc", quantity: 8, date: new Date("2023-01-03") },
      { item: "xyz", quantity: 15, date: new Date("2023-01-01") },
      { item: "xyz", quantity: 7, date: new Date("2023-01-02") },
      { item: "xyz", quantity: 3, date: new Date("2023-01-03") }
    ]);
    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');
    await inventory.insertMany([
      { _id: 1, item: "abc", tags: ["red", "green", "blue", "yellow", "purple"] },
      { _id: 2, item: "xyz", tags: ["alpha", "beta", "gamma"] }
```

```
    ]);
    const expressionResult = await inventory.aggregate([
      { $project: { lastThreeTags: { $lastN: { input: "$tags", n: 3 } } } }
    ]).toArray();
    console.log('Expression result:', expressionResult);

    // Window operator usage: last N values over a window in each partition
    const stockPrices = db.collection('stockPrices');
    await stockPrices.insertMany([
      { _id: 1, ticker: "ABC", hour: 1, price: 50 },
      { _id: 2, ticker: "ABC", hour: 2, price: 45 },
      { _id: 3, ticker: "ABC", hour: 3, price: 60 },
      { _id: 4, ticker: "ABC", hour: 4, price: 40 },
      { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
      { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
      { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
    ]);
    const windowResult = await stockPrices.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$ticker",
          sortBy: { hour: 1 },
          output: {
            lastTwoSoFar: {
              $lastN: { input: "$price", n: 2 },
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } finally {
    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')

    try:
        db = client['test']

        # Accumulator usage: last N values per group
        sales = db['sales']
        sales.insert_many([
            { 'item': 'abc', 'quantity': 10, 'date': datetime(2023, 1, 1) },
            { 'item': 'abc', 'quantity': 5, 'date': datetime(2023, 1, 2) },
            { 'item': 'abc', 'quantity': 8, 'date': datetime(2023, 1, 3) },
            { 'item': 'xyz', 'quantity': 15, 'date': datetime(2023, 1, 1) },
            { 'item': 'xyz', 'quantity': 7, 'date': datetime(2023, 1, 2) },
            { 'item': 'xyz', 'quantity': 3, 'date': datetime(2023, 1, 3) }
        ])
        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']
        inventory.insert_many([
            { '_id': 1, 'item': 'abc', 'tags': ['red', 'green', 'blue', 'yellow',
 'purple'] },
            { '_id': 2, 'item': 'xyz', 'tags': ['alpha', 'beta', 'gamma'] }
        ])
        expression_result = list(inventory.aggregate([
            { '$project': { 'lastThreeTags': { '$lastN': { 'input': '$tags', 'n':
 3 } } } }
        ]))
        print('Expression result:', expression_result)

        # Window operator usage: last N values over a window in each partition
        stock_prices = db['stockPrices']
        stock_prices.insert_many([
            { '_id': 1, 'ticker': 'ABC', 'hour': 1, 'price': 50 },
            { '_id': 2, 'ticker': 'ABC', 'hour': 2, 'price': 45 },
```

```
            { '_id': 3, 'ticker': 'ABC', 'hour': 3, 'price': 60 },
            { '_id': 4, 'ticker': 'ABC', 'hour': 4, 'price': 40 },
            { '_id': 5, 'ticker': 'XYZ', 'hour': 1, 'price': 30 },
            { '_id': 6, 'ticker': 'XYZ', 'hour': 2, 'price': 35 },
            { '_id': 7, 'ticker': 'XYZ', 'hour': 3, 'price': 25 }
        ])
        window_result = list(stock_prices.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$ticker',
                    'sortBy': { 'hour': 1 },
                    'output': {
                        'lastTwoSoFar': {
                            '$lastN': { 'input': '$price', 'n': 2 },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()

example()
```

$let

Amazon DocumentDB の $let演算子は、変数を値にバインドし、それらの変数を式で使用するために使用されます。これにより、集約パイプラインの同じステージ内の後続の式で使用できるローカル変数を定義できます。

パラメータ

- vars: 式で使用する変数を定義するオブジェクト。
- in: vars パラメータで定義された変数を使用する式。

例 (MongoDB シェル)

この例では、$let演算子を使用して長方形の面積を計算する方法を示します。

サンプルドキュメントを作成する

```
db.shapes.insertMany([
  { name: "Rectangle 1", length: 5, width: 3 },
  { name: "Rectangle 2", length: 7, width: 4 },
  { name: "Rectangle 3", length: 6, width: 2 }
]);
```

クエリの例

```
db.shapes.aggregate([
  {
    $project: {
      name: 1,
      area: {
        $let: {
          vars: {
            length: "$length",
            width: "$width"
          },
          in: {
            $multiply: ["$$length", "$$width"]
          }
        }
      }
    }
  }
])
```

出力

```
[
  {
    "_id": ObjectId("6161e5b1a3eba3c7f2960d03"),
    "name": "Rectangle 1",
    "area": 15
  },
  {
    "_id": ObjectId("6161e5b1a3eba3c7f2960d04"),
    "name": "Rectangle 2",
    "area": 28
  },
  {
```

```
    "_id": ObjectId("6161e5b1a3eba3c7f2960d05"),
    "name": "Rectangle 3",
    "area": 12
  }
]
```

コードの例

$let コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $limit演算子は、クエリによって返されるドキュメントの数を制限するために使用されます。これは MongoDB $limit演算子に似ていますが、Amazon DocumentDB で使用する場合の具体的な考慮事項がいくつかあります。

Amazon DocumentDB では、 $limit演算子は、一致するドキュメントの合計のサブセットを取得するページ分割に役立ちます。これにより、各レスポンスで返されるドキュメントの数を制御できるため、パフォーマンスが向上し、ネットワーク経由で転送されるデータ量が減ります。

パラメータ

- limit: 返されるドキュメントの最大数。これは負以外の整数値である必要があります。

例 (MongoDB シェル)

次の例は、 $limit演算子を使用して、指定されたフィルターに一致する最大 1 つのドキュメントを返す方法を示しています。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.test.createIndex({ comments: "text" });

db.test.find({
  $and: [
    { star_rating: 5 },
    { $text: { $search: "couch" } }
  ]
}).limit(1);
```

出力

```
[ { _id: 2, star_rating: 5, comments: 'comfortable couch' } ]
```

コードの例

$limit コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の$listSearchIndexes集約ステージは、コレクション上の既存の検索インデックスに関する情報を返します。集約パイプラインの最初のステージである必要があります。

パラメータ

- name: (オプション) 情報を返す検索インデックスの名前。省略すると、コレクション上のすべての検索インデックスが返されます。

出力フィールド

返される各ドキュメントには、次のフィールドが含まれます。

- name: 検索インデックスの名前。
- status: インデックスのビルド状態。READY (ビルド済みで有効) 、 BUILDING (同時ビルドが進行中) 、または FAILED (インデックスビルドが正常に完了しなかった) のいずれか。
- queryable: インデックスを現在クエリの処理に使用できるかどうかを示すブール値。これはfalse、非表示のインデックス (のままREADY) とFAILEDインデックス用です。非表示インデックスは、 が READY、 statusが の場合queryableですfalse。
- latestDefinitionVersion: version (インデックス形式バージョン) と createdAt (インデックスが作成された時刻) を含むドキュメント。

[Syntax] (構文)

```
db.collection.aggregate([
  { $listSearchIndexes: {} }
])

// Or filter by name:
db.collection.aggregate([
  { $listSearchIndexes: { name: "mySearchIndex" } }
])
```

例 (MongoDB シェル)

次の例は、 $listSearchIndexesステージを使用してコレクション上のすべての検索インデックスを一覧表示する方法を示しています。

クエリの例

```
db.movies.aggregate([
  { $listSearchIndexes: {} }
]);
```

出力

```
[
  {
```

```
    "name": "default",
    "status": "READY",
    "queryable": true,
    "latestDefinitionVersion": {
      "version": 2,
      "createdAt": ISODate("2026-05-11T21:12:57.974Z")
    }
  }
]
```

非表示のインデックスは残りREADYますが、 が報告されます。これはqueryable: false、非表示になっている間はクエリの処理に使用できないためです。

```
[
  {
    "name": "myHiddenIndex",
    "status": "READY",
    "queryable": false,
    "latestDefinitionVersion": {
      "version": 2,
      "createdAt": ISODate("2026-05-11T21:12:57.974Z")
    }
  }
]
```

特定のインデックス名でフィルタリングするには:

```
db.movies.aggregate([
  { $listSearchIndexes: { name: "default" } }
]);
```

コードの例

$listSearchIndexes ステージを使用するためのコード例を表示するには、使用する言語のタブを選択します。

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 の $literal演算子は、集約パイプラインステージ内のリテラル値を表すために使用されます。これにより、フィールドリファレンスや式として解釈することなく、数値、文字列、ブール値などの特定の値を含めることができます。

この演算子は、動的クエリフィルターの構築や計算の実行など、より複雑な集計パイプラインの一部としてリテラル値を含める必要がある場合に特に便利です。

パラメータ

なし

例 (MongoDB シェル)

次の例は、 $literal演算子を使用してリテラル値を集約パイプラインに含める方法を示しています。$literal 演算子は、値 18 を $gt 式にリテラル値として含めるために使用されます。これにより、集約パイプラインは経過時間フィールドをリテラル値 18 と比較して、その人が成人かどうかを判断できます。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.collection.aggregate([
  {
    $project: {
      name: 1,
      age: 1,
      city: 1,
      isAdult: { $gt: ["$age", { $literal: 18 }] }
    }
  }
]);
```

出力

```
[
  {
    "_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
  }
]
```

コードの例

$literal コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能。

Amazon DocumentDB の$ln演算子は、指定された数の自然対数 (ベース e) を計算します。数値の対数をベース e に返します。

パラメータ

- expression: 自然対数を計算する数値。

例 (MongoDB シェル)

次の例は、$log演算子を使用して数値の自然対数を計算する方法を示しています。

サンプルドキュメントを作成する

```
db.numbers.insertMany([
  { _id: 1, value: 10 },
  { _id: 2, value: 100 },
  { _id: 3, value: 1000 }
]);
```

クエリの例

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    naturalLog: { $ln: "$value" }
```

```
  }}
]);
```

出力

```
[
  { "_id" : 1, "naturalLog" : 2.302585092994046 },
  { "_id" : 2, "naturalLog" : 4.605170185988092 },
  { "_id" : 3, "naturalLog" : 6.907755278982137 }
]
```

コードの例

$ln コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能。

Amazon DocumentDB の$log演算子は、数値の自然対数を計算します。指定された数の base-e 対数を返します。

パラメータ

- expression: 自然対数を計算する数値。
- base: ログを計算するための基本値。

例 (MongoDB シェル)

次の例は、$log演算子を使用して数値の自然対数を計算する方法を示しています。

サンプルドキュメントを作成する

```
db.numbers.insertMany([
```

```
  { _id: 1, value: 10 },
  { _id: 2, value: 100 },
  { _id: 3, value: 1000 }
]);
```

クエリの例

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    naturalLog: { $log: ["$value", 10] }
  }}
]);
```

出力

```
[
  { "_id" : 1, "naturalLog" : 1 },
  { "_id" : 2, "naturalLog" : 2 },
  { "_id" : 3, "naturalLog" : 2.9999999999999996 }
]
```

コードの例

$log コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能。

Amazon DocumentDB の $log10演算子は、数値の base-10 対数を計算するために使用されます。集約パイプライン内の数値フィールドで対数計算を実行するのに役立ちます。

パラメータ

- `expression`: base-10 対数を計算する数値式。

例 (MongoDB シェル)

次の例は、 $log10演算子を使用して数値フィールドの base-10 対数を計算する方法を示しています。

サンプルドキュメントを作成する

```
db.numbers.insertMany([
  { _id: 1, value: 1 },
  { _id: 2, value: 10 },
  { _id: 3, value: 100 },
  { _id: 4, value: 1000 }
]);
```

クエリの例

```
db.numbers.aggregate([
  {
    $project: {
      _id: 1,
      log10Value: { $log10: "$value" }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "log10Value": 0 },
  { "_id": 2, "log10Value": 1 },
  { "_id": 3, "log10Value": 2 },
  { "_id": 4, "log10Value": 3 }
]
```

コードの例

$log10 コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 集計演算子は 2 つの値を比較し、最初の値が 2 番目より小さいtrue場合は を返し、それ以外の場合は を返しますfalse。

パラメータ

- expression1: 比較する最初の値。
- expression2: 比較する 2 番目の値。

例 (MongoDB シェル)

次の例は、 $ltを使用して低在庫項目を識別する方法を示しています。

サンプルドキュメントを作成する

```
db.warehouse.insertMany([
```

```
  { _id: 1, item: "Bolts", stock: 5 },
  { _id: 2, item: "Nuts", stock: 25 },
  { _id: 3, item: "Screws", stock: 8 }
]);
```

クエリの例

```
db.warehouse.aggregate([
  {
    $project: {
      item: 1,
      stock: 1,
      lowStock: { $lt: ["$stock", 10] }
    }
  }
]);
```

出力

```
[
  { _id: 1, item: 'Bolts', stock: 5, lowStock: true },
  { _id: 2, item: 'Nuts', stock: 25, lowStock: false },
  { _id: 3, item: 'Screws', stock: 8, lowStock: true }
]
```

コードの例

$lt 集計演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 集計演算子は 2 つの値を比較し、最初の値が 2 番目以下のtrue場合は を返し、それ以外の場合は を返しますfalse。

パラメータ

- expression1: 比較する最初の値。
- expression2: 比較する 2 番目の値。

例 (MongoDB シェル)

次の例は、 $lte演算子を使用して予算にわかりやすい項目を識別する方法を示しています。

サンプルドキュメントを作成する

```
db.menu.insertMany([
  { _id: 1, dish: "Salad", price: 8 },
  { _id: 2, dish: "Pasta", price: 12 },
  { _id: 3, dish: "Soup", price: 6 }
]);
```

クエリの例

```
db.menu.aggregate([
  {
    $project: {
      dish: 1,
      price: 1,
      affordable: { $lte: ["$price", 10] }
    }
  }
]);
```

出力

```
[
  { _id: 1, dish: 'Salad', price: 8, affordable: true },
  { _id: 2, dish: 'Pasta', price: 12, affordable: false },
  { _id: 3, dish: 'Soup', price: 6, affordable: true }
]
```

コードの例

$lte 集計演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$lookup集約ステージでは、2 つのコレクション間で左外部結合を実行できます。このオペレーションでは、一致するフィールド値に基づいて複数のコレクションのデータを組み合わせることができます。これは、関連するコレクションのデータをクエリ結果に組み込む必要がある場合に特に便利です。

パラメータ

- from: 結合を実行するコレクションの名前。
- localField: と照合する入力ドキュメントのフィールドforeignField。
- foreignField: と照合するfromコレクション内のドキュメントのフィールドlocalField。
- as: fromコレクションから一致するドキュメントを含む出力ドキュメントに追加する新しいフィールドの名前。
- pipeline: localFieldと が指定されている場合はオプションforeignFieldです。from コレクションで実行する集約パイプライン。パイプラインはfromコレクションからドキュメントを返します。すべてのドキュメントを返すには、空のパイプライン を指定します[]。パイプラインに $outまたは $mergeステージを含めることはできません。「結合条件と相関サブクエリ (MongoDB シェル)」を参照してください。
- let: オプション。pipeline ステージで使用する変数を定義するドキュメント。変数式を使用して、 に結合されている入力ドキュメントのフィールドにアクセスしますpipeline。パイプラインステージで変数を参照するには、 "$$<variable>"構文を使用します。「結合条件と相関サブクエリ (MongoDB シェル)」を参照してください。

例 (MongoDB シェル)

次の例は、ordersコレクションからcustomersコレクションにデータを結合するシンプルな$lookupオペレーションを示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.customers.aggregate([
  {
    $lookup: {
      from: "orders",
      localField: "_id",
      foreignField: "customer_id",
      as: "orders"
    }
  }
]);
```

出力

```
[
  {
    _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 } ]
  }
]
```

結合条件と相関サブクエリ (MongoDB シェル)

バージョン 8.0.2 の新機能。

Amazon DocumentDB $lookupでは、単一の等価一致に加えて、結合されたコレクションで集約パイプラインを実行できます。これは、すべての入力ドキュメントに対して同じサブクエリを実行する相関関係のないサブクエリと、各入力ドキュメントのフィールドから定義された変数を使用してそのドキュメントに相関する結果を返す相関関係のないサブクエリをサポートします。

[Syntax] (構文)

相関サブクエリを実行するには、入力ドキュメントのフィールドの変数を で定義しlet、pipelineステージで参照します。

```
{
  $lookup: {
    from: <collection to join>,
    let: { <var_1>: <expression>, ..., <var_n>: <expression> },
    pipeline: [ <pipeline to run on the joined collection> ],
    as: <output array field>
  }
}
```

パラメータ

- let: 入力ドキュメントのフィールドから変数を定義するドキュメント。パイプラインステージで変数を参照するには、 "$$<variable>"構文を使用します。変数には、 にネストされた追加のステージを含めpipeline、 の$lookupステージからアクセスできますpipeline。$match ステージでは、変数にアクセスするために $expr演算子を使用する必要があります。相関しないサブクエリの実行を省略letします。
- pipeline: fromコレクションで実行する集約パイプライン。パイプラインは入力ドキュメントのフィールドに直接アクセスできません。代わりに を使用して変数を定義letし、変数を参照します。すべてのドキュメントを返すには、空のパイプライン を指定します[]。

$match ステージ内の$expr演算子に配置された $eq、、$gt$gte、$lt、および $lte 比較演算子は、結合されたfromコレクションのフィールドをlet変数または定数と比較するときに、コレクションでインデックスを使用できます。

サンプルドキュメントを作成する

```
db.customers.insertMany([
  { _id: 1, name: "Alice", min_total: 60 },
  { _id: 2, name: "Bob", min_total: 50 },
  { _id: 3, name: "Charlie", min_total: 20 }
]);

db.orders.insertMany([
  { _id: 1, customer_id: 1, total: 50 },
  { _id: 2, customer_id: 1, total: 100 },
  { _id: 3, customer_id: 2, total: 75 }
]);
```

クエリの例

次の例では、各顧客を注文に参加させ、 がその顧客自身のmin_total値以上totalである注文のみを保持します。結合条件と顧客ごとのフィルターは、どちらも 内のlet変数を参照します$expr。

```
db.customers.aggregate([
  {
    $lookup: {
      from: "orders",
      let: { customer_id: "$_id", min_total: "$min_total" },
      pipeline: [
        {
          $match: {
            $expr: {
              $and: [
                { $eq: [ "$customer_id", "$$customer_id" ] },
                { $gte: [ "$total", "$$min_total" ] }
              ]
            }
          }
        },
        { $project: { _id: 0, total: 1 } }
      ],
      as: "qualifying_orders"
```

```
    }
  }
]);
```

出力

```
[
  { _id: 1, name: 'Alice', min_total: 60, qualifying_orders: [ { total: 100 } ] },
  { _id: 2, name: 'Bob', min_total: 50, qualifying_orders: [ { total: 75 } ] },
  { _id: 3, name: 'Charlie', min_total: 20, qualifying_orders: [] }
]
```

顧客ごとに、サブクエリはその顧客の _idおよび min_total値で評価されるため、各ドキュメントは独自の相関結果を受け取ります。

簡潔な構文を使用した相関サブクエリ (MongoDB シェル)

バージョン 8.0.2 の新機能。

Amazon DocumentDB は、簡潔な相関サブクエリ構文もサポートしています。この構文は、localFieldと での等価一致を foreignFieldと組み合わせますpipeline。簡潔な構文では、$matchステージ内の$expr演算子内で等価一致を表現する要件が削除されます。

[Syntax] (構文)

```
{
  $lookup: {
    from: <collection to join>,
    localField: <field from the input documents>,
    foreignField: <field from the documents of the "from" collection>,
    let: { <var_1>: <expression>, ..., <var_n>: <expression> },
    pipeline: [ <pipeline to run> ],
    as: <output array field>
  }
}
```

クエリの例

次の例では、前の相関サブクエリの例と同じ結果を返します。_id と の等価一致customer_idは、$expr条件foreignFieldの代わりに localFieldと で表されます。

```
db.customers.aggregate([
  {
    $lookup: {
      from: "orders",
      localField: "_id",
      foreignField: "customer_id",
      let: { min_total: "$min_total" },
      pipeline: [
        { $match: { $expr: { $gte: [ "$total", "$$min_total" ] } } },
        { $project: { _id: 0, total: 1 } }
      ],
      as: "qualifying_orders"
    }
  }
]);
```

出力

```
[
  { _id: 1, name: 'Alice', min_total: 60, qualifying_orders: [ { total: 100 } ] },
  { _id: 2, name: 'Bob', min_total: 50, qualifying_orders: [ { total: 75 } ] },
  { _id: 3, name: 'Charlie', min_total: 20, qualifying_orders: [] }
]
```

相関サブクエリでサポートされているステージ

$lookup パイプラインが を使用する場合let、次のステージで"$$<variable>"変数を参照できます。

- $match ($expr演算子を使用)
- $project
- $addFields
- $set
- $group
- $bucket
- $bucketAuto
- $sortByCount
- $lookup (ネスト化済み)

- $graphLookup

$sort、、、、、、、 などの変数式を評価しないステージは$unset、パイプラインでも使用できますが$skip$limit$count$sample$unwind、変数を参照することはできません。

を使用する$lookupパイプラインで
は、、let$facet、、$redact、、$replaceRoot$replaceWith$geoNear、$setWindowFields$fi
$matchの各ステージはサポートされていません$text。

制限事項

- には $outまたは $mergeステージを含めるpipelineことはできません。

コードの例

$lookup コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

関連するサブクエリコードの例

相関サブクエリ (letと を使用pipeline) で$lookupステージを使用するためのコード例を表示するには、使用する言語のタブを選択します。

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 customers = db.collection('customers');
  await customers.insertMany([
    { _id: 1, name: "Alice", min_total: 60 },
    { _id: 2, name: "Bob", min_total: 50 },
    { _id: 3, name: "Charlie", min_total: 20 }
  ]);
  const orders = db.collection('orders');
  await orders.insertMany([
    { _id: 1, customer_id: 1, total: 50 },
    { _id: 2, customer_id: 1, total: 100 },
    { _id: 3, customer_id: 2, total: 75 }
  ]);

  // Correlated subquery: filter each customer's orders with a per-customer
 threshold
  const result = await customers.aggregate([
    {
      $lookup: {
        from: 'orders',
        localField: '_id',
        foreignField: 'customer_id',
        let: { min_total: '$min_total' },
        pipeline: [
          { $match: { $expr: { $gte: [ '$total', '$$min_total' ] } } },
          { $project: { _id: 0, total: 1 } }
        ],
        as: 'qualifying_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

    customers = db.customers
    customers.insert_many([
        { "_id": 1, "name": "Alice", "min_total": 60 },
        { "_id": 2, "name": "Bob", "min_total": 50 },
        { "_id": 3, "name": "Charlie", "min_total": 20 }
    ])
    orders = db.orders
    orders.insert_many([
        { "_id": 1, "customer_id": 1, "total": 50 },
        { "_id": 2, "customer_id": 1, "total": 100 },
        { "_id": 3, "customer_id": 2, "total": 75 }
    ])

    # Correlated subquery: filter each customer's orders with a per-customer
 threshold
    pipeline = [
        {
            "$lookup": {
                "from": "orders",
                "localField": "_id",
                "foreignField": "customer_id",
                "let": { "min_total": "$min_total" },
                "pipeline": [
                    { "$match": { "$expr": { "$gte": [ "$total", "$
$min_total" ] } } },
                    { "$project": { "_id": 0, "total": 1 } }
                ],
                "as": "qualifying_orders"
```

```
            }
        }
    ]

    result = customers.aggregate(pipeline)

    for doc in result:
        print(doc)

    client.close()

example()
```

$ltrim

バージョン 4.0 の新機能。

Elastic クラスターではサポートされていません。

Amazon DocumentDB の $ltrim演算子は、文字列から先頭文字を削除するために使用されます。デフォルトでは、先頭の空白文字は削除されますが、chars 引数を渡すことで削除する文字のセットを指定することもできます。

パラメータ

- input: 先頭の空白文字を削除する入力文字列。
- chars: (オプション) 特定の文字を削除します。

例 (MongoDB シェル)

次の例は、 を使用して文字列の先頭から指定された文字 (「 *」) $ltrimを削除する方法を示しています。

サンプルドキュメントを作成する

```
db.collection.insertMany([
  { name: " *John Doe", age: 30 },
  { name: "Jane Doe*", age: 25 },
  { name: "  Bob Smith  ", age: 35 }
]);
```

クエリの例

```
db.collection.aggregate([
  {
    $project: {
      _id: 0,
      name: {
        $ltrim: { input: "$name", chars: " *" }
      },
      age: 1
    }
  }
]);
```

出力

```
[
  { "name": "John Doe", "age": 30 },
  { "name": "Jane Doe ", "age": 25 },
  { "name": "Bob Smith  ", "age": 35 }
]
```

コードの例

`$ltrim` コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $map演算子を使用すると、配列内の各要素に指定された式を適用し、変換された要素を含む新しい配列を返すことができます。この演算子は、配列内のデータの操作と変換に特に役立ちます。これにより、配列処理をデータベースレベルにプッシュすることで、アプリケーションコードを簡素化し、クエリパフォーマンスを向上させることができます。

パラメータ

- input: 変換する配列。
- as: (オプション) 式の 内で、処理中の現在の要素を表すために使用される変数の名前。
- in: 入力配列の各要素に適用される式。

例 (MongoDB シェル)

次の例は、$map 演算子を使用して数値の配列を変換し、各値を 2 倍する方法を示しています。

サンプルドキュメントを作成する

```
db.collection.insertMany([
  { _id: 1, numbers: [1, 2, 3, 4, 5] },
  { _id: 2, numbers: [10, 20, 30, 40, 50] }
])
```

クエリの例

```
db.collection.aggregate([
  {
    $project: {
      doubledNumbers: { $map: { input: "$numbers", as: "num", in: { $multiply: ["$
$num", 2] } } }
    }
  }
])
```

出力

```
[
  { _id: 1, doubledNumbers: [2, 4, 6, 8, 10] },
  { _id: 2, doubledNumbers: [20, 40, 60, 80, 100] }
]
```

コードの例

$map コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$matchパイプラインステージは、指定されたクエリ条件に一致するもののみに入力ドキュメントをフィルタリングするために使用されます。これは、集約オペレーションで最も一般的に使用されるパイプラインステージの 1 つです。$match ステージは他のパイプラインス

テージの前に適用されるため、後続のステージで処理する必要があるドキュメントの数を効率的に減らすことができます。

パラメータ

- query: オペレーションの選択基準を表すドキュメント。クエリドキュメントは、 find()メソッドと同じ構文を使用します。

例 (MongoDB シェル)

次の例は、 $matchステージを使用して、特定のフィールド値に基づいてドキュメントをフィルタリングする方法を示しています。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.collection.aggregate([
  { $match: { age: { $gt: 30 } } },
  { $project: { _id: 1, name: 1, city: 1 } }
]);
```

出力

```
[
  { "_id": 3, "name": "Bob", "city": "Chicago" },
  { "_id": 4, "name": "Alice", "city": "Miami" }
]
```

$match ステージは、 ageフィールドが 30 より大きいドキュメントのみを含むようにドキュメントをフィルタリングします。

コードの例

$match コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 集計演算子は、ドキュメントのセット全体で指定されたフィールドの最大値を返します。この演算子は、一連のドキュメントで最大値を見つけるのに役立ちます。

パラメータ

- expression: 最大値の計算に使用する式。

例 (MongoDB シェル)

次の例は、 $max演算子を使用して学生ドキュメントのコレクションで最大スコアを検索する方法を示しています。$group ステージはすべてのドキュメントをグループ化し、 $max演算子を使用してすべてのドキュメントの scoreフィールドの最大値を計算します。

サンプルドキュメントを作成する

```
db.students.insertMany([
  { name: "John", score: 85 },
  { name: "Jane", score: 92 },
  { name: "Bob", score: 78 },
  { name: "Alice", score: 90 }
])
```

クエリの例

```
db.students.aggregate([
  { $group: { _id: null, maxScore: { $max: "$score" } } },
  { $project: { _id: 0, maxScore: 1 } }
])
```

出力

```
[ { maxScore: 92 } ]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$max 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメントの最大値を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

> **Note**
>
> でウィンドウ演算子として使用する場合$setWindowFields、 $maxは 100 MB の中間データに制限されます。この制限を超えるオペレーションはエラーを返します。

サンプルドキュメントを作成する

```
db.stockPrices.insertMany([
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

クエリの例

次の例では、ドキュメントを でパーティション化しticker、各パーティションを でソートしてhour、パーティションの先頭から現在のドキュメントまでの最高価格を返します。

```
db.stockPrices.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        highestSoFar: {
          $max: "$price",
          window: { documents: ["unbounded", "current"] }
        }
```

```
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "highestSoFar": 50 },
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "highestSoFar": 50 },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "highestSoFar": 60 },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "highestSoFar": 60 },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "highestSoFar": 30 },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "highestSoFar": 35 },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "highestSoFar": 35 }
]
```

各ドキュメントは で拡張されます。これはhighestSoFar、現在のドキュメントまでのパーティションprice内の の最大実行数です。

コードの例

$max 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group) とウィンドウ演算子の使用 (の場合) の両方を示しています$setWindowFields。

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');

  // Accumulator usage: maximum value across a group
  const students = db.collection('students');
  await students.insertMany([
    { name: "John", score: 85 },
    { name: "Jane", score: 92 },
    { name: "Bob", score: 78 },
```

```
    { name: "Alice", score: 90 }
  ]);
  const accumulatorResult = await students.aggregate([
    { $group: { _id: null, maxScore: { $max: "$score" } } }
  ]).toArray();
  console.log('Accumulator result:', accumulatorResult);

  // Window operator usage: running maximum within each partition
  const stockPrices = db.collection('stockPrices');
  await stockPrices.insertMany([
    { _id: 1, ticker: "ABC", hour: 1, price: 50 },
    { _id: 2, ticker: "ABC", hour: 2, price: 45 },
    { _id: 3, ticker: "ABC", hour: 3, price: 60 },
    { _id: 4, ticker: "ABC", hour: 4, price: 40 },
    { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
    { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
    { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
  ]);
  const windowResult = await stockPrices.aggregate([
    {
      $setWindowFields: {
        partitionBy: "$ticker",
        sortBy: { hour: 1 },
        output: {
          highestSoFar: {
            $max: "$price",
            window: { documents: ["unbounded", "current"] }
          }
        }
      }
    }
  ]).toArray();
  console.log('Window result:', windowResult);

  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']

    # Accumulator usage: maximum value across a group
    students = db.students
    students.insert_many([
        { "name": "John", "score": 85 },
        { "name": "Jane", "score": 92 },
        { "name": "Bob", "score": 78 },
        { "name": "Alice", "score": 90 }
    ])
    accumulator_result = list(students.aggregate([
        { "$group": { "_id": None, "maxScore": { "$max": "$score" } } }
    ]))
    print('Accumulator result:', accumulator_result)

    # Window operator usage: running maximum within each partition
    stock_prices = db.stockPrices
    stock_prices.insert_many([
        { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50 },
        { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45 },
        { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60 },
        { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40 },
        { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30 },
        { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35 },
        { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25 }
    ])
    window_result = list(stock_prices.aggregate([
        {
            "$setWindowFields": {
                "partitionBy": "$ticker",
                "sortBy": { "hour": 1 },
                "output": {
                    "highestSoFar": {
                        "$max": "$price",
                        "window": { "documents": ["unbounded", "current"] }
                    }
                }
            }
        }
    ]))
```

```
    print('Window result:', window_result)

    client.close()

example()
```

$maxN

バージョン 8.0.1 から新規。

Amazon DocumentDB の $maxN演算子は N 最大値を返します。$group ステージでアキュムレータとして使用すると、各グループの N 最大値の配列が返されます。配列式演算子として使用すると、配列の最大 N 要素が返されます。

パラメータ

- input: n最大値を返す配列またはフィールドに解決される式。
- n: 返す最大値の数を指定する正の整数に解決される式。

動作

配列式演算子の動作

- よりn小さい値を指定することはできません1。
- $maxN は、input配列で見つかったnull値をフィルタリングします。
- 指定された nがinput配列内の要素の数以上である場合、 はinput配列内のすべての要素$maxNを返します。
- が非配列値にinput解決された場合、集約オペレーションは工ラーになります。
- に数値要素と文字列要素の両方inputが含まれている場合、文字列要素は BSON 比較順序に従って数値要素の前にソートされます。

アキュムレータの動作

- アキュムレータとして使用する場合、 は定数であるか、 _idの値に依存する正の整数式nである必要があります$group。
- $maxN は null 値と欠落した値をフィルタリングします。

- グループに含まれるn要素が より少ない場合、 はグループ内のすべての要素$maxNを返します。
- $maxN は BSON 比較順序に従って入力データを比較し、適切な出力タイプを決定します。入力データに複数のデータ型が含まれている場合、$maxN出力型は比較順序で最も高い値になります。

出力順序

$maxN は、特定のソート順序で値を返します。特定のソート順序を保証する必要がある場合は、$topN代わりに を使用するか、結果を でラップします$sortArray。

例 (MongoDB シェル)

次の例は、アキュムレータを使用して各サブジェクトの 2 $maxN つの最高スコアを取得する方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.scores.aggregate([
  { $group: { _id: "$subject", highestTwo: { $maxN: { input: "$score", n: 2 } } } }
]);
```

出力

```
[
  { "_id": "math", "highestTwo": [85, 93] },
  { "_id": "science", "highestTwo": [88, 90] }
]
```

式の使用例 (MongoDB シェル)

$maxN 演算子を$projectステージ内の式として使用して、配列フィールドから N 最大の要素を返すこともできます。

サンプルドキュメントを作成する

```
db.readings.insertMany([
  { _id: 1, sensor: "A", values: [45, 12, 78, 3, 56] },
  { _id: 2, sensor: "B", values: [90, 23, 67, 11, 44] }
]);
```

クエリの例

```
db.readings.aggregate([
  { $project: {
      highestThree: { $maxN: { input: "$values", n: 3 } }
    }}
]);
```

出力

```
[
  { "_id": 1, "highestThree": [45, 56, 78] },
  { "_id": 2, "highestThree": [44, 90, 67] }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$maxN 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメントの最大n値の配列を返します。output フィールドで演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

> **Note**
>
> でウィンドウ演算子として使用する場合$setWindowFields、 $maxNは 100 MB の中間データに制限されます。この制限を超えるオペレーションはエラーを返します。

サンプルドキュメントを作成する

```
db.stockPrices.insertMany([
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

クエリの例

次の例では、ドキュメントを でパーティション化しticker、各パーティションを でソートしてhour、パーティションの先頭から現在のドキュメントまでの 2 つの最高価格を返します。

```
db.stockPrices.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        highestTwoSoFar: {
          $maxN: { input: "$price", n: 2 },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "highestTwoSoFar": [50] },
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "highestTwoSoFar": [50, 45] },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "highestTwoSoFar": [60, 50] },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "highestTwoSoFar": [60, 50] },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "highestTwoSoFar": [30] },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "highestTwoSoFar": [35, 30] },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "highestTwoSoFar": [35, 30] }
```

```
]
```

各ドキュメントは、ウィンドウprice内の 2 つの最大値highestTwoSoFarである で拡張されます。出力順序セクションで説明したように、返された配列内の値の順序は保証されません。

コードの例

$maxN 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group)、式の使用 (の場合$project)、ウィンドウ演算子の使用 (の場合) を示しています$setWindowFields。

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');
    await 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 }
    ]);
    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');
    await readings.insertMany([
      { _id: 1, sensor: "A", values: [45, 12, 78, 3, 56] },
      { _id: 2, sensor: "B", values: [90, 23, 67, 11, 44] }
    ]);
    const expressionResult = await readings.aggregate([
      { $project: { highestThree: { $maxN: { input: "$values", n: 3 } } } }
    ]).toArray();
    console.log('Expression result:', expressionResult);

    // Window operator usage: N largest values over a window in each partition
    const stockPrices = db.collection('stockPrices');
    await stockPrices.insertMany([
      { _id: 1, ticker: "ABC", hour: 1, price: 50 },
      { _id: 2, ticker: "ABC", hour: 2, price: 45 },
      { _id: 3, ticker: "ABC", hour: 3, price: 60 },
      { _id: 4, ticker: "ABC", hour: 4, price: 40 },
      { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
      { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
      { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
    ]);
    const windowResult = await stockPrices.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$ticker",
          sortBy: { hour: 1 },
          output: {
            highestTwoSoFar: {
              $maxN: { input: "$price", n: 2 },
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } 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']
        scores.insert_many([
            { '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 }
        ])
        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']
        readings.insert_many([
            { '_id': 1, 'sensor': 'A', 'values': [45, 12, 78, 3, 56] },
            { '_id': 2, 'sensor': 'B', 'values': [90, 23, 67, 11, 44] }
        ])
        expression_result = list(readings.aggregate([
            { '$project': { 'highestThree': { '$maxN': { 'input': '$values', 'n':
 3 } } } }
        ]))
        print('Expression result:', expression_result)

        # Window operator usage: N largest values over a window in each partition
        stock_prices = db['stockPrices']
```

```
        stock_prices.insert_many([
            { '_id': 1, 'ticker': 'ABC', 'hour': 1, 'price': 50 },
            { '_id': 2, 'ticker': 'ABC', 'hour': 2, 'price': 45 },
            { '_id': 3, 'ticker': 'ABC', 'hour': 3, 'price': 60 },
            { '_id': 4, 'ticker': 'ABC', 'hour': 4, 'price': 40 },
            { '_id': 5, 'ticker': 'XYZ', 'hour': 1, 'price': 30 },
            { '_id': 6, 'ticker': 'XYZ', 'hour': 2, 'price': 35 },
            { '_id': 7, 'ticker': 'XYZ', 'hour': 3, 'price': 25 }
        ])
        window_result = list(stock_prices.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$ticker',
                    'sortBy': { 'hour': 1 },
                    'output': {
                        'highestTwoSoFar': {
                            '$maxN': { 'input': '$price', 'n': 2 },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()

example()
```

$median

バージョン 8.0.1 から新規。

Amazon DocumentDB の $median演算子は、数値データの中央値を計算します。アキュムレータは、集約パイプラインの$groupステージのグループ内のドキュメント全体の数値の中央値を計算します。式として、数値の配列の中央値を計算します。

パラメータ

- input: 数値または数値の配列に解決される式。

- method: 計算方法を指定する文字列。現在サポートされているの"approximate"は t-digest アルゴリズムを使用する のみです。

動作

"approximate" メソッドは、t-digest アルゴリズムを使用しておおよその中央値を計算します。結果は、値間の補間ではなく、データセットからの既存の値です。データポイントの数が増えると、精度が向上します。

例 (MongoDB シェル)

次の例は、 $median演算子を使用してクラスあたりのテストスコアの中央値を計算する方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.students.aggregate([
  { $group: {
      _id: "$class",
      medianScore: { $median: { input: "$score", method: "approximate" } }
    }}
]);
```

出力

```
[
```

```
  { "_id": "A", "medianScore": 85 },
  { "_id": "B", "medianScore": 80 }
]
```

式の使用例 (MongoDB シェル)

$median 演算子を$projectステージ内の式として使用して、配列フィールドの中央値を計算することもできます。

サンプルドキュメントを作成する

```
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] }
]);
```

クエリの例

```
db.surveys.aggregate([
  { $project: {
      medianRating: { $median: { input: "$ratings", method: "approximate" } }
    }}
]);
```

出力

```
[
  { "_id": 1, "medianRating": 5 },
  { "_id": 2, "medianRating": 30 },
  { "_id": 3, "medianRating": 2 }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$median 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメントの数値の中央値を計算します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

> **Note**
>
> でウィンドウ演算子として使用する場合$setWindowFields、 $medianは 100 MB の中間データに制限されます。この制限を超えるオペレーションはエラーを返します。

サンプルドキュメントを作成する

```
db.temperatures.insertMany([
  { _id: 1, city: "A", reading: 60 },
  { _id: 2, city: "A", reading: 65 },
  { _id: 3, city: "A", reading: 70 },
  { _id: 4, city: "B", reading: 80 },
  { _id: 5, city: "B", reading: 85 },
  { _id: 6, city: "B", reading: 90 }
]);
```

クエリの例

次の例では、ドキュメントを でパーティションcity化し、各パーティション内のすべてのドキュメントreadingの中央値を計算します。

```
db.temperatures.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$city",
      output: {
        medianReading: {
          $median: { input: "$reading", method: "approximate" },
          window: { documents: ["unbounded", "unbounded"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "city": "A", "reading": 60, "medianReading": 65 },
  { "_id": 2, "city": "A", "reading": 65, "medianReading": 65 },
  { "_id": 3, "city": "A", "reading": 70, "medianReading": 65 },
```

```
  { "_id": 4, "city": "B", "reading": 80, "medianReading": 85 },
  { "_id": 5, "city": "B", "reading": 85, "medianReading": 85 },
  { "_id": 6, "city": "B", "reading": 90, "medianReading": 85 }
]
```

各ドキュメントはmedianReading、パーティション内のすべてのドキュメントreadingにわたるのおおよその中央値である で拡張されます。

コードの例

$median 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group)、式の使用 (の場合$project)、ウィンドウ演算子の使用 (の場合) を示しています$setWindowFields。

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');
    await 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 }
    ]);
    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');
  await 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] }
  ]);
  const expressionResult = await surveys.aggregate([
    { $project: {
        medianRating: { $median: { input: "$ratings", method: "approximate" } }
      }}
  ]).toArray();
  console.log('Expression result:', expressionResult);

  // Window operator usage: median across each partition
  const temperatures = db.collection('temperatures');
  await temperatures.insertMany([
    { _id: 1, city: "A", reading: 60 },
    { _id: 2, city: "A", reading: 65 },
    { _id: 3, city: "A", reading: 70 },
    { _id: 4, city: "B", reading: 80 },
    { _id: 5, city: "B", reading: 85 },
    { _id: 6, city: "B", reading: 90 }
  ]);
  const windowResult = await temperatures.aggregate([
    {
      $setWindowFields: {
        partitionBy: "$city",
        output: {
          medianReading: {
            $median: { input: "$reading", method: "approximate" },
            window: { documents: ["unbounded", "unbounded"] }
          }
        }
      }
    }
  ]).toArray();
```

```
    console.log('Window result:', windowResult);

  } 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']
        students.insert_many([
            { '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 }
        ])
        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']
        surveys.insert_many([
            { '_id': 1, 'ratings': [3, 5, 7, 9, 2] },
            { '_id': 2, 'ratings': [10, 20, 30, 40, 50] },
            { '_id': 3, 'ratings': [1, 1, 2, 3, 5] }
        ])
        expression_result = list(surveys.aggregate([
            { '$project': {
                'medianRating': { '$median': { 'input': '$ratings', 'method':
'approximate' } }
            }}
        ]))
        print('Expression result:', expression_result)

        # Window operator usage: median across each partition
        temperatures = db['temperatures']
        temperatures.insert_many([
            { '_id': 1, 'city': 'A', 'reading': 60 },
            { '_id': 2, 'city': 'A', 'reading': 65 },
            { '_id': 3, 'city': 'A', 'reading': 70 },
            { '_id': 4, 'city': 'B', 'reading': 80 },
            { '_id': 5, 'city': 'B', 'reading': 85 },
            { '_id': 6, 'city': 'B', 'reading': 90 }
        ])
        window_result = list(temperatures.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$city',
                    'output': {
                        'medianReading': {
                            '$median': { 'input': '$reading', 'method':
'approximate' },
                            'window': { 'documents': ['unbounded', 'unbounded'] }
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()
```

```
example()
```

$meta

$meta 集約演算子は、集約パイプライン内のドキュメントに関連付けられたメタデータにアクセスします。これは一般的に、テキスト検索スコアの取得と関連性による結果のソートに使用されます。

パラメータ

- textScore: 検索クエリへのドキュメントの関連性を示すテキスト検索スコアを取得します。

例 (MongoDB シェル)

次の例は、集約パイプラインで $meta演算子を使用してテキスト検索スコアを取得およびソートする方法を示しています。

サンプルドキュメントを作成する

```
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." }
]);
```

クエリの例

```
db.articles.aggregate([
  { $match: { $text: { $search: "Python" } } },
  { $addFields: { score: { $meta: "textScore" } } },
  { $sort: { score: -1 } }
]);
```

出力

```
[
  {
    _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
  }
]
```

コードの例

$meta 集計演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

8.0 で導入

Amazon DocumentDB の$merge集約ステージは、前のパイプラインステージの結果をターゲットコレクションにマージするために使用されます。これは、入力ドキュメントのデータに基づいて、ターゲットコレクション内のドキュメントを更新または挿入する場合に便利です。

$merge ステージでは、入力ドキュメントとターゲットコレクション間の一致条件に基づいて、次のようなさまざまなアクションを実行できます。

```
- Insert new documents
- Update existing documents
- Delete documents
- Fail the operation if there are any conflicts
```

パラメータ

- into: (必須) 入力ドキュメントをマージするターゲットコレクションの名前。

- on: (必須) 入力ドキュメントとターゲットコレクション間の一致条件として使用するフィールド (複数可) 。
- whenMatched: (オプション) 入力ドキュメントがターゲットコレクション内の既存のドキュメントと一致するときに実行するアクション。サポートされている値は "merge"、"replace"、"keepExisting"、"fail" です。Amazon DocumentDB では、whenMatched値として集約パイプラインを指定することはできません。
- whenNotMatched: (オプション) 入力ドキュメントがターゲットコレクション内のドキュメントと一致しない場合に実行するアクション。サポートされている値は、 "insert"と です"fail"。
- bypassDocumentValidation: (オプション) に設定するとtrue、$mergeステージによってターゲットコレクションに書き込まれたドキュメントは、そのコレクションで定義されたスキーマ検証ルール ($jsonSchema) をバイパスします。デフォルトは でfalse、ターゲットコレクションの検証ルールが適用されます。これは Amazon DocumentDB 8.0 でサポートされています。

例 (MongoDB シェル)

次の例は、 $mergeステージを使用して、入力パイプラインの新しいデータでusersコレクションを更新する方法を示しています。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.inputData.aggregate([
  {
    $merge: {
      into: "users",
      on: "_id",
      whenMatched: "merge",
      whenNotMatched: "insert"
```

```
    }
  }
])
```

出力

$merge パイプラインの実行後、usersコレクションには次のドキュメントが含まれます。

```
[
  { _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" }
]
```

コードの例

$merge コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

Node.js

Node.js アプリケーションで $merge 演算子を使用する例を次に示します。

```
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

Python アプリケーションで $merge 演算子を使用する例を次に示します。

```
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 の $mergeObjects演算子は、複数のドキュメントまたはオブジェクトを 1 つのドキュメントに結合するために使用されます。この演算子は、2 つ以上のドキュメントまたはオ

ブジェクトの内容をマージする必要があり、あるオブジェクトの値を別のオブジェクトのものと上書きする可能性がある場合に特に便利です。

パラメータ

- expression1: マージする最初のオブジェクト。
- expression2: (オプション) マージする 2 番目のオブジェクト。
- expression3: (オプション) マージする追加のオブジェクト。

例 (MongoDB シェル)

次の例は、 $mergeObjects演算子を使用して 2 つのオブジェクトを組み合わせる方法を示しています。

サンプルドキュメントを作成する

```
db.collection.insertMany([
  { "_id": 1, "name": "John", "address": { "city": "New York", "state": "NY" } },
  { "_id": 2, "name": "Jane", "address": { "city": "Los Angeles", "state": "CA" } }
]);
```

クエリの例

```
db.collection.aggregate([
  {
    $project: {
      "combinedAddress": {
        $mergeObjects: ["$address", { "country": "USA" }]
      }
    }
  }
])
```

出力

```
[
  {
    "_id": 1,
    "combinedAddress": {
      "city": "New York",
      "state": "NY",
```

```
      "country": "USA"
    }
  },
  {
    "_id": 2,
    "combinedAddress": {
      "city": "Los Angeles",
      "state": "CA",
      "country": "USA"
    }
  }
]
```

コードの例

$mergeObjects コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $millisecond演算子は、日付値のミリ秒部分を抽出するために使用されます。

パラメータ

なし

例 (MongoDB シェル)

この例では、 $millisecond演算子を使用して日付値のミリ秒部分を抽出する方法を示します。

サンプルドキュメントを作成する

```
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")
  }
])
```

クエリの例

```
db.events.aggregate([
  {
    $project: {
      name: 1,
      milliseconds: { $millisecond: "$timestamp" }
    }
  }
])
```

出力

```
[
  {
    "_id": ObjectId("644332a42054ed1b0d15f0c1"),
    "name": "Event 1",
    "milliseconds": 123
  },
  {
    "_id": ObjectId("644332a42054ed1b0d15f0c2"),
    "name": "Event 2",
    "milliseconds": 456
  },
  {
    "_id": ObjectId("644332a42054ed1b0d15f0c3"),
    "name": "Event 3",
```

```
    "milliseconds": 789
  }
]
```

コードの例

$millisecond コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 集計演算子は、ドキュメントのセット全体で指定されたフィールドの最小値を返します。この演算子は、一連のドキュメントで最小値を見つけるのに役立ちます。

パラメータ

- expression: 評価する式。これは、フィールドパス、変数、または値に解決される任意の式です。

例 (MongoDB シェル)

次の例は、 $min演算子を使用して、複数のドキュメントにわたる ageフィールドの最小値を見つける方法を示しています。

サンプルドキュメントを作成する

```
db.users.insertMany([
  { name: "John", age: 35 },
  { name: "Jane", age: 28 },
  { name: "Bob", age: 42 },
  { name: "Alice", age: 31 }
]);
```

クエリの例

```
db.users.aggregate([
  { $group: { _id: null, minAge: { $min: "$age" } } },
  { $project: { _id: 0, minAge: 1 } }
])
```

出力

```
[ { minAge: 28 } ]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$min 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメントの最小値を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

> **Note**
>
> でウィンドウ演算子として使用する場合$setWindowFields、 $minは 100 MB の中間データに制限されます。この制限を超えるオペレーションはエラーを返します。

サンプルドキュメントを作成する

```
db.stockPrices.insertMany([
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

クエリの例

次の例では、ドキュメントを でパーティション化しticker、各パーティションを でソートしてhour、現在のドキュメントを通じてパーティションの先頭から見た最低価格を返します。

```
db.stockPrices.aggregate([
```

```
  {
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        lowestSoFar: {
          $min: "$price",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "lowestSoFar": 50 },
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "lowestSoFar": 45 },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "lowestSoFar": 45 },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "lowestSoFar": 40 },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "lowestSoFar": 30 },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "lowestSoFar": 30 },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "lowestSoFar": 25 }
]
```

各ドキュメントは で拡張されます。これはlowestSoFar、現在のドキュメントまで、パーティションprice内の の実行最小値です。

コードの例

$min 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group) とウィンドウ演算子の使用 (の場合) の両方を示しています$setWindowFields。

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');

  // Accumulator usage: minimum value across a group
  const users = db.collection('users');
  await users.insertMany([
    { name: "John", age: 35 },
    { name: "Jane", age: 28 },
    { name: "Bob", age: 42 },
    { name: "Alice", age: 31 }
  ]);
  const accumulatorResult = await users.aggregate([
    { $group: { _id: null, minAge: { $min: "$age" } } }
  ]).toArray();
  console.log('Accumulator result:', accumulatorResult);

  // Window operator usage: running minimum within each partition
  const stockPrices = db.collection('stockPrices');
  await stockPrices.insertMany([
    { _id: 1, ticker: "ABC", hour: 1, price: 50 },
    { _id: 2, ticker: "ABC", hour: 2, price: 45 },
    { _id: 3, ticker: "ABC", hour: 3, price: 60 },
    { _id: 4, ticker: "ABC", hour: 4, price: 40 },
    { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
    { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
    { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
  ]);
  const windowResult = await stockPrices.aggregate([
    {
      $setWindowFields: {
        partitionBy: "$ticker",
        sortBy: { hour: 1 },
        output: {
          lowestSoFar: {
            $min: "$price",
            window: { documents: ["unbounded", "current"] }
          }
        }
      }
    }
  ]).toArray();
  console.log('Window result:', windowResult);

  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']

    # Accumulator usage: minimum value across a group
    users = db.users
    users.insert_many([
        { "name": "John", "age": 35 },
        { "name": "Jane", "age": 28 },
        { "name": "Bob", "age": 42 },
        { "name": "Alice", "age": 31 }
    ])
    accumulator_result = list(users.aggregate([
        { "$group": { "_id": None, "minAge": { "$min": "$age" } } }
    ]))
    print('Accumulator result:', accumulator_result)

    # Window operator usage: running minimum within each partition
    stock_prices = db.stockPrices
    stock_prices.insert_many([
        { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50 },
        { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45 },
        { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60 },
        { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40 },
        { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30 },
        { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35 },
        { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25 }
    ])
    window_result = list(stock_prices.aggregate([
        {
            "$setWindowFields": {
                "partitionBy": "$ticker",
                "sortBy": { "hour": 1 },
                "output": {
```

```
                    "lowestSoFar": {
                        "$min": "$price",
                        "window": { "documents": ["unbounded", "current"] }
                    }
                }
            }
        }
    ]))
    print('Window result:', window_result)

    client.close()

example()
```

$minN

バージョン 8.0.1 から新規。

Amazon DocumentDB の $minN演算子は、N 個の最小値を返します。$group ステージでアキュムレータとして使用すると、各グループの N 最小値の配列が返されます。配列式演算子として使用すると、配列の N 個の最小要素が返されます。

パラメータ

- input: n最小値を返す配列またはフィールドに解決される式。
- n: 返す最小値の数を指定する正の整数に解決される式。

動作

配列式演算子の動作

- よりn小さい値を指定することはできません1。
- $minN は、input配列で見つかったnull値をフィルタリングします。
- 指定された nがinput配列内の要素の数以上である場合、 はinput配列内のすべての要素$minNを返します。
- が非配列値にinput解決された場合、集約オペレーションはエラーになります。
- に数値要素と文字列要素の両方inputが含まれている場合、数値要素は BSON 比較順序に従って文字列要素の前にソートされます。

アキュムレータの動作

- アキュムレータとして使用する場合、 は定数であるか、 _idの値に依存する正の整数式nである必要があります$group。
- $minN は null 値と欠落した値をフィルタリングします。
- グループに含まれるn要素が より少ない場合、 はグループ内のすべての要素$minNを返します。
- $minN は BSON 比較順序に従って入力データを比較し、適切な出力タイプを決定します。入力データに複数のデータ型が含まれている場合、$minN出力型は比較順序で最低になります。

出力順序

$minN は、特定のソート順序で値を返します。特定のソート順序を保証する必要がある場合は、$bottomN代わりに を使用するか、結果を でラップします$sortArray。

例 (MongoDB シェル)

次の例は、アキュムレータを使用して各サブジェクトの 2 $minN つの最低スコアを取得する方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.scores.aggregate([
  { $group: { _id: "$subject", lowestTwo: { $minN: { input: "$score", n: 2 } } } }
]);
```

出力

```
[
  { "_id": "math", "lowestTwo": [72, 68] },
  { "_id": "science", "lowestTwo": [78, 65] }
]
```

式の使用例 (MongoDB シェル)

$minN 演算子を$projectステージ内の式として使用して、配列フィールドから N 最小要素を返すこともできます。

サンプルドキュメントを作成する

```
db.readings.insertMany([
  { _id: 1, sensor: "A", values: [45, 12, 78, 3, 56] },
  { _id: 2, sensor: "B", values: [90, 23, 67, 11, 44] }
]);
```

クエリの例

```
db.readings.aggregate([
  { $project: {
      lowestThree: { $minN: { input: "$values", n: 3 } }
    }}
]);
```

出力

```
[
  { "_id": 1, "lowestThree": [45, 12, 3] },
  { "_id": 2, "lowestThree": [44, 23, 11] }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$minN 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメントのn最小値までの配列を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

> **Note**
>
> でウィンドウ演算子として使用する場合$setWindowFields、 $minNは 100 MB の中間データに制限されます。この制限を超えるオペレーションはエラーを返します。

サンプルドキュメントを作成する

```
db.stockPrices.insertMany([
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

クエリの例

次の例では、ドキュメントを でパーティション化しticker、各パーティションを でソートしてhour、パーティションの先頭から現在のドキュメントまでの 2 つの最低価格を返します。

```
db.stockPrices.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        lowestTwoSoFar: {
          $minN: { input: "$price", n: 2 },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "lowestTwoSoFar": [50] },
```

```
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "lowestTwoSoFar": [45, 50] },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "lowestTwoSoFar": [45, 50] },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "lowestTwoSoFar": [40, 45] },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "lowestTwoSoFar": [30] },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "lowestTwoSoFar": [30, 35] },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "lowestTwoSoFar": [25, 30] }
]
```

各ドキュメントはlowestTwoSoFar、ウィンドウ内の 2 つのprice最小値である で拡張されます。出力順序セクションで説明したように、返された配列内の値の順序は保証されません。

コードの例

$minN 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group)、式の使用 (の場合$project)、ウィンドウ演算子の使用 (の場合) を示しています$setWindowFields。

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');
    await 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 }
    ]);
    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');
   await readings.insertMany([
     { _id: 1, sensor: "A", values: [45, 12, 78, 3, 56] },
     { _id: 2, sensor: "B", values: [90, 23, 67, 11, 44] }
   ]);
   const expressionResult = await readings.aggregate([
     { $project: { lowestThree: { $minN: { input: "$values", n: 3 } } } }
   ]).toArray();
   console.log('Expression result:', expressionResult);

   // Window operator usage: N smallest values over a window in each partition
   const stockPrices = db.collection('stockPrices');
   await stockPrices.insertMany([
     { _id: 1, ticker: "ABC", hour: 1, price: 50 },
     { _id: 2, ticker: "ABC", hour: 2, price: 45 },
     { _id: 3, ticker: "ABC", hour: 3, price: 60 },
     { _id: 4, ticker: "ABC", hour: 4, price: 40 },
     { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
     { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
     { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
   ]);
   const windowResult = await stockPrices.aggregate([
     {
       $setWindowFields: {
         partitionBy: "$ticker",
         sortBy: { hour: 1 },
         output: {
           lowestTwoSoFar: {
             $minN: { input: "$price", n: 2 },
             window: { documents: ["unbounded", "current"] }
           }
         }
       }
     }
   ]).toArray();
   console.log('Window result:', windowResult);

 } 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']
        scores.insert_many([
            { '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 }
        ])
        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']
        readings.insert_many([
            { '_id': 1, 'sensor': 'A', 'values': [45, 12, 78, 3, 56] },
            { '_id': 2, 'sensor': 'B', 'values': [90, 23, 67, 11, 44] }
        ])
        expression_result = list(readings.aggregate([
```

```
            { '$project': { 'lowestThree': { '$minN': { 'input': '$values', 'n':
 3 } } } }
        ]))
        print('Expression result:', expression_result)

        # Window operator usage: N smallest values over a window in each partition
        stock_prices = db['stockPrices']
        stock_prices.insert_many([
            { '_id': 1, 'ticker': 'ABC', 'hour': 1, 'price': 50 },
            { '_id': 2, 'ticker': 'ABC', 'hour': 2, 'price': 45 },
            { '_id': 3, 'ticker': 'ABC', 'hour': 3, 'price': 60 },
            { '_id': 4, 'ticker': 'ABC', 'hour': 4, 'price': 40 },
            { '_id': 5, 'ticker': 'XYZ', 'hour': 1, 'price': 30 },
            { '_id': 6, 'ticker': 'XYZ', 'hour': 2, 'price': 35 },
            { '_id': 7, 'ticker': 'XYZ', 'hour': 3, 'price': 25 }
        ])
        window_result = list(stock_prices.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$ticker',
                    'sortBy': { 'hour': 1 },
                    'output': {
                        'lowestTwoSoFar': {
                            '$minN': { 'input': '$price', 'n': 2 },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()

example()
```

$minute

Amazon DocumentDB の$minute集約パイプラインステージは、日付またはタイムスタンプフィールドから分値を抽出します。

この演算子は、集計パイプライン内で日付と時刻ベースの計算またはグループ化を実行する必要がある場合に便利です。

パラメータ

- `expression`: 分値を抽出する日付またはタイムスタンプフィールド。

例 (MongoDB シェル)

次の例は、 `$minute`演算子を使用して、タイムスタンプフィールドから抽出された分値でドキュメントをグループ化し、各グループのドキュメントの数をカウントする方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.events.aggregate([
  {
    $group: {
      _id: {
        minute: { $minute: "$timestamp" }
      },
      count: { $count: {} }
    }
  },
  { $sort: { "_id.minute": 1 } }
]);
```

出力

```
[
  { "_id": { "minute": 30 }, "count": 2 },
  { "_id": { "minute": 31 }, "count": 2 },
```

```
  { "_id": { "minute": 32 }, "count": 1 }
]
```

コードの例

$minute コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、数値に対してモジュロ演算を実行する算術演算子です。ある数値を別の数値で割った余りを返します。この演算子は、数値が奇数か偶数かを判断するため、または項目を有限数のグループに分散するために一般的に使用されます。

パラメータ

- expression1: 分配式。
- expression2: 除数式。

例 (MongoDB シェル)

この例では、 $mod演算子を使用して、100 個のパッケージで出荷するときに残っているウィジェットの数を確認する方法を示します。

サンプルドキュメントを作成する

```
db.widgets.insertMany([
  { "_id" : 1, "widget" : "A", "count" : 80372 },
  { "_id" : 2, "widget" : "B", "count" : 409282 },
  { "_id" : 3, "widget" : "C", "count" : 60739 }
])
```

クエリの例

```
db.widgets.aggregate([
  { $addFields: { leftOver: { $mod: [ "$count", 100 ] } } }
])
```

出力

```
[
  { "_id" : 1, "widget" : "A", "count" : 80372, "leftOver" : 72 },
  { "_id" : 2, "widget" : "B", "count" : 409282, "leftOver" : 82 },
  { "_id" : 3, "widget" : "C", "count" : 60739, "leftOver" : 39 }
]
```

出力には、ドキュメントごとに の残りを 100 でcount割った値が表示されます。これは、100 個のパッケージで出荷する際の残りのウィジェットの数を表します。

コードの例

$mod コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$month演算子は、日付の月を 1 ~ 12 の数値として返します。この演算子は、日付フィールドから月コンポーネントを抽出し、日付ベースの集計と分析を実行するのに役立ちます。

パラメータ

- date_expression: これは、月を抽出する日付またはタイムスタンプを含む式またはフィールドです。

例 (MongoDB シェル)

次の例は、 $month演算子を使用して日付フィールドから月を抽出し、月別にデータをグループ化する方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.sales.aggregate([
  { $group: {
      _id: { month: { $month: "$date" } },
      totalSales: { $sum: "$price" }
    }},
```

```
  { $sort: { "_id.month": 1 } }
]);
```

出力

```
[
  { _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 }
]
```

コードの例

$month コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $multiply演算子は、2 つ以上のフィールドまたは式の値を乗算するために使用されます。この演算子は、ドキュメント内の数値フィールドに対して算術演算を実行する場合に特に便利です。$project や など、集約パイプラインのさまざまな段階で使用して、新しいフィールドを作成したり$addFields、既存のフィールドを変更したりできます。

パラメータ

- expression1: 乗算する最初の数値式。
- expression2: 乗算する 2 番目の数値式。
- [expression3, ...]: (オプション) 乗算する追加の数値式。

例 (MongoDB シェル)

次の例では$multiply、 を使用して、旅行bonus_rateにクレジットカードを使用したお客様のbase_milesと を乗算bonus_milesして計算する方法を示します。

サンプルドキュメント

```
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 }
]);
```

クエリの例

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

出力

```
[
  { _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 }
]
```

コードの例

`$multiply` コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

$natural

Amazon DocumentDB の $natural演算子は、ドキュメントを自然な順序でソートするために使用されます。これは、ドキュメントがコレクションに挿入された順序です。これは、デフォルトのソート動作とは対照的です。これは、指定されたフィールドの値に基づいてドキュメントをソートすることです。

パラメータ

なし

例 (MongoDB シェル)

次の例は、 $natural演算子を使用してコレクション内のドキュメントを自然な順序でソートする方法を示しています。

サンプルドキュメントを作成する

```
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 }
```

```
]);
```

クエリの例

```
db.people.find({}, { "_id": 1, "name": 1 }).sort({ "$natural": 1 });
```

出力

```
[
  { "_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" }
]
```

クエリは、コレクション内のドキュメントを自然な順序でソートします。これは、挿入された順序です。

コードの例

$natural コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 集計演算子は 2 つの値を比較し、等しくtrueない場合は を返し、等しくない場合は を返しますfalse。

パラメータ

- expression1: 比較する最初の値。
- expression2: 比較する 2 番目の値。

例 (MongoDB シェル)

次の例は、 $ne演算子を使用してステータスが変更された注文を識別する方法を示しています。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.orders.aggregate([
  {
    $project: {
      orderId: 1,
      status: 1,
      expectedStatus: 1,
      needsAttention: { $ne: ["$status", "$expectedStatus"] }
    }
  }
]);
```

出力

```
[
  { _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 }
]
```

コードの例

$ne 集計演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 集計演算子は、式に対して論理 NOT オペレーションを実行します。これは、式が に評価trueされた場合とfalse、式が に評価falseされた場合を返しますtrue。

パラメータ

- expression: 否定する式。

例 (MongoDB シェル)

次の例は、 $not演算子を使用してブール値を反転させる方法を示しています。

サンプルドキュメントを作成する

```
db.users.insertMany([
  { _id: 1, name: "Alice", active: true },
  { _id: 2, name: "Bob", active: false },
  { _id: 3, name: "Charlie", active: true }
]);
```

クエリの例

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      active: 1,
      inactive: { $not: ["$active"] }
    }
  }
]);
```

出力

```
[
  { _id: 1, name: 'Alice', active: true, inactive: false },
  { _id: 2, name: 'Bob', active: false, inactive: true },
  { _id: 3, name: 'Charlie', active: true, inactive: false }
]
```

コードの例

$not 集計演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$objectToArray集計演算子は、オブジェクト (またはドキュメント) を配列に変換します。演算子への入力はドキュメントであり、出力は入力ドキュメント内の各フィールドと値のペアの配列要素で構成されます。この演算子は、特定のフィールドの最大値または最小値を持つドキュメントを検索する場合など、ドキュメントの個々のフィールドを配列として操作する必要がある場合に便利です。

パラメータ

- `expression`: 配列に変換するドキュメント式。

例 (MongoDB シェル)

次の例は、 $objectToArray演算子を使用して、ビデオレンタルストアチェーンの最大インベントリを持つドキュメントを検索する方法を示しています。

サンプルドキュメントを作成する

```
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
    }
  }
]);
```

クエリの例

```
db.videos.aggregate([
  {
    $project: {
      name: 1,
      videos: {
        $objectToArray: "$inventory"
      }
    }
  },
  {
    $unwind: "$videos"
  },
  {
```

```
    $group: {
      _id: "$name",
      maxInventory: {
        $max: "$videos.v"
      }
    }
  }
]);
```

出力

```
[
  {
    "_id": "Bravemind",
    "maxInventory": 1000
  },
  {
    "_id": "Live Soft",
    "maxInventory": 1000
  },
  {
    "_id": "Romancing the Rock",
    "maxInventory": 500
  },
  {
    "_id": "Top Pilot",
    "maxInventory": 1000
  }
]
```

コードの例

$objectToArray コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

$or 集計演算子は複数の式を評価し、少なくとも 1 つの式が に評価trueされた場合に を返しますtrue。すべての式が であるfalse場合にのみ返されますfalse。

パラメータ

- expressions: 評価する式の配列。

例 (MongoDB シェル)

次の例では、 $or演算子を使用して、製品が複数の条件のいずれかを満たしているかどうかを確認します。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.items.aggregate([
  {
    $project: {
      name: 1,
      price: 1,
      onSale: 1,
      goodDeal: {
        $or: [
          { $lt: ["$price", 50] },
          { $eq: ["$onSale", true] }
        ]
      }
    }
  }
]);
```

出力

```
[
  { _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 }
]
```

コードの例

$or 集計演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $out演算子は、集約パイプラインの結果を指定されたコレクションに書き込むために使用されます。

$out はパイプラインの最後のステージである必要があります。

パラメータ

- output_collection: 集計結果を書き込む出力コレクションの名前。

注: コレクションが既に存在する場合、集約ステージの結果に置き換えられます。

例 (MongoDB シェル)

次の例は、Amazon DocumentDB の $out演算子を使用して、集約パイプラインの結果を新しいコレクションに書き込む方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.products.aggregate([
  { $group: { _id: "$category", totalPrice: { $sum: "$price" } } },
  { $out: "product_categories" }
])
```

出力

なし (結果は出力コレクションに書き込まれます) 。

集約パイプラインは製品をカテゴリ別にグループ化し、各カテゴリの項目の合計価格を計算します。$out 演算子は、「product_categories」という名前の新しいコレクションに結果を書き込みます。

出力コレクションで結果を表示するには:

```
db.product_categories.find()
[
{ "_id" : "Books", "totalPrice" : 125 },
{ "_id" : "Electronics", "totalPrice" : 300 }
]
```

コードの例

$out コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 の新機能。

Amazon DocumentDB の $percentile演算子は、数値データの指定されたパーセンタイル値を計算します。アキュムレータは、集約パイプラインの$groupステージのグループ内のドキュメント全体のパーセンタイル値を計算します。式として、数値の配列のパーセンタイルを計算します。

パラメータ

- input: 数値または数値の配列に解決される式。
- p: 0 から 1 までのパーセンタイル値の配列。各値は計算するパーセンタイルを表します。たとえば、 は 25 パーセンタイル、50 パーセンタイル、75 パーセンタイルを[0.25, 0.5, 0.75]計算します。
- method: 計算方法を指定する文字列。現在は、"approximate" のみがサポートされます。

動作

"approximate" メソッドは、t-digest アルゴリズムを使用して、近似パーセンタイルベースのメトリクスを計算します。小さなデータセットでは、個別のパーセンタイル値が同じ結果に解決される場合があります。たとえば、グループごとに 5 つの値しかない場合、p90 と p99 はどちらもグループ内の最大値を返す可能性があります。データポイントの数が増えると、精度が向上します。

例 (MongoDB シェル)

次の例は、 $percentile演算子を使用して、クラスあたりのスコアの 25 パーセンタイルと 75 パーセンタイルを計算する方法を示しています。

サンプルドキュメントを作成する

```
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 }
```

```
]);
```

クエリの例

```
db.students.aggregate([
  { $group: {
      _id: "$class",
      percentiles: { $percentile: { input: "$score", p: [0.25, 0.75], method:
 "approximate" } }
    }}
]);
```

出力

```
[
  { "_id": "A", "percentiles": [72, 90] },
  { "_id": "B", "percentiles": [75, 88] }
]
```

式の使用例 (MongoDB シェル)

$percentile 演算子を$projectステージ内の式として使用して、配列フィールドのパーセンタイルを計算することもできます。

サンプルドキュメントを作成する

```
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] }
]);
```

クエリの例

```
db.surveys.aggregate([
  { $project: {
      quartiles: { $percentile: { input: "$responses", p: [0.25, 0.5, 0.75], method:
 "approximate" } }
    }}
]);
```

出力

```
[
  { "_id": 1, "quartiles": [6, 10, 16] },
  { "_id": 2, "quartiles": [5, 9, 15] }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$percentile 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメントに指定されたパーセンタイル値を計算します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

> **Note**
>
> でウィンドウ演算子として使用する場合$setWindowFields、 $percentileは 100 MB の中間データに制限されます。この制限を超えるオペレーションはエラーを返します。

サンプルドキュメントを作成する

```
db.sensorData.insertMany([
  { _id: 1, sensor: "A", reading: 72 },
  { _id: 2, sensor: "A", reading: 85 },
  { _id: 3, sensor: "A", reading: 90 },
  { _id: 4, sensor: "A", reading: 68 },
  { _id: 5, sensor: "A", reading: 95 },
  { _id: 6, sensor: "B", reading: 80 },
  { _id: 7, sensor: "B", reading: 75 },
  { _id: 8, sensor: "B", reading: 92 },
  { _id: 9, sensor: "B", reading: 88 },
  { _id: 10, sensor: "B", reading: 70 }
]);
```

クエリの例

次の例では、ドキュメントを でパーティションsensor化し、各パーティション内のすべてのドキュメントreadingで の 25 パーセンタイルと 75 パーセンタイルを計算します。

```
db.sensorData.aggregate([
```

```
  {
    $setWindowFields: {
      partitionBy: "$sensor",
      output: {
        readingPercentiles: {
          $percentile: { input: "$reading", p: [0.25, 0.75], method: "approximate" },
          window: { documents: ["unbounded", "unbounded"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "sensor": "A", "reading": 72, "readingPercentiles": [72, 90] },
  { "_id": 2, "sensor": "A", "reading": 85, "readingPercentiles": [72, 90] },
  { "_id": 3, "sensor": "A", "reading": 90, "readingPercentiles": [72, 90] },
  { "_id": 4, "sensor": "A", "reading": 68, "readingPercentiles": [72, 90] },
  { "_id": 5, "sensor": "A", "reading": 95, "readingPercentiles": [72, 90] },
  { "_id": 6, "sensor": "B", "reading": 80, "readingPercentiles": [75, 88] },
  { "_id": 7, "sensor": "B", "reading": 75, "readingPercentiles": [75, 88] },
  { "_id": 8, "sensor": "B", "reading": 92, "readingPercentiles": [75, 88] },
  { "_id": 9, "sensor": "B", "reading": 88, "readingPercentiles": [75, 88] },
  { "_id": 10, "sensor": "B", "reading": 70, "readingPercentiles": [75, 88] }
]
```

各ドキュメントは`readingPercentiles`、パーティション内のすべてのドキュメント`reading`のおよそ 25 パーセンタイルと 75 パーセンタイルである で拡張されます。

コードの例

`$percentile` 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合`$group`)、式の使用 (の場合`$project`)、ウィンドウ演算子の使用 (の場合) を示しています`$setWindowFields`。

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');
    await 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 }
    ]);
    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');
    await 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] }
    ]);
    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);

    // Window operator usage: percentiles across each partition
    const sensorData = db.collection('sensorData');
    await sensorData.insertMany([
      { _id: 1, sensor: "A", reading: 72 },
      { _id: 2, sensor: "A", reading: 85 },
      { _id: 3, sensor: "A", reading: 90 },
      { _id: 4, sensor: "A", reading: 68 },
      { _id: 5, sensor: "A", reading: 95 },
      { _id: 6, sensor: "B", reading: 80 },
      { _id: 7, sensor: "B", reading: 75 },
      { _id: 8, sensor: "B", reading: 92 },
      { _id: 9, sensor: "B", reading: 88 },
      { _id: 10, sensor: "B", reading: 70 }
    ]);
    const windowResult = await sensorData.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$sensor",
          output: {
            readingPercentiles: {
              $percentile: { input: "$reading", p: [0.25, 0.75], method:
 "approximate" },
              window: { documents: ["unbounded", "unbounded"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } 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']
        students.insert_many([
            { '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 }
        ])
        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']
        surveys.insert_many([
            { '_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] }
        ])
        expression_result = list(surveys.aggregate([
            { '$project': {
                'quartiles': { '$percentile': { 'input': '$responses', 'p': [0.25,
 0.5, 0.75], 'method': 'approximate' } }
            }}
        ]))
        print('Expression result:', expression_result)
```

```
        # Window operator usage: percentiles across each partition
        sensor_data = db['sensorData']
        sensor_data.insert_many([
            { '_id': 1, 'sensor': 'A', 'reading': 72 },
            { '_id': 2, 'sensor': 'A', 'reading': 85 },
            { '_id': 3, 'sensor': 'A', 'reading': 90 },
            { '_id': 4, 'sensor': 'A', 'reading': 68 },
            { '_id': 5, 'sensor': 'A', 'reading': 95 },
            { '_id': 6, 'sensor': 'B', 'reading': 80 },
            { '_id': 7, 'sensor': 'B', 'reading': 75 },
            { '_id': 8, 'sensor': 'B', 'reading': 92 },
            { '_id': 9, 'sensor': 'B', 'reading': 88 },
            { '_id': 10, 'sensor': 'B', 'reading': 70 }
        ])
        window_result = list(sensor_data.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$sensor',
                    'output': {
                        'readingPercentiles': {
                            '$percentile': { 'input': '$reading', 'p': [0.25, 0.75],
 'method': 'approximate' },
                            'window': { 'documents': ['unbounded', 'unbounded'] }
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()

example()
```

$pow

8.0 で導入

Amazon DocumentDB の $pow演算子を使用すると、数値をパワーに上げることができます。これは、集約パイプライン内で指数計算を実行するのに役立ちます。

パラメータ

- <number> (必須): 電力に引き上げる数値。
- <exponent> (必須): 数値を上げるべき電力。

例 (MongoDB シェル)

次の例は、 $pow演算子を使用して数値の二乗を計算する方法を示しています。

サンプルドキュメントを作成する

```
db.numbers.insertMany([
  { "_id": 1, "value": 2 },
  { "_id": 2, "value": 3 },
  { "_id": 3, "value": 4 }
]);
```

クエリの例

```
db.numbers.aggregate([
  { $addFields: { "square": { $pow: ["$value", 2] } } }
])
```

出力

```
[
  { "_id": 1, "value": 2, "square": 4 },
  { "_id": 2, "value": 3, "square": 9 },
  { "_id": 3, "value": 4, "square": 16 }
]
```

コードの例

$pow コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

Node.js

Node.js アプリケーションで $pow 演算子を使用する例を次に示します。

```
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

Python アプリケーションで $pow 演算子を使用する例を次に示します。

```
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

$push 集計演算子は、各グループの指定された式からすべての値の配列を返します。これは通常、値を配列に蓄積するために$groupステージ内で使用されます。

パラメータ

- expression: グループ内の各ドキュメントについて評価する式。

例 (MongoDB シェル)

次の例は、 $push演算子を使用して各カテゴリのすべての製品名を収集する方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.sales.aggregate([
  {
    $group: {
      _id: "$category",
      products: { $push: "$product" }
    }
  }
]);
```

出力

```
[
  { _id: 'Furniture', products: [ 'Desk', 'Chair' ] },
  { _id: 'Electronics', products: [ 'Laptop', 'Mouse', 'Keyboard' ] }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$push 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメントに対して指定された式のすべての値の配列をソート順

に返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

> **Note**
>
> でウィンドウ演算子として使用する場合$setWindowFields、 $pushは 100 MB の中間データに制限されます。この制限を超えるオペレーションはエラーを返します。

サンプルドキュメントを作成する

```
db.purchases.insertMany([
  { _id: 1, category: "Electronics", day: 1, product: "Laptop" },
  { _id: 2, category: "Electronics", day: 2, product: "Mouse" },
  { _id: 3, category: "Electronics", day: 3, product: "Keyboard" },
  { _id: 4, category: "Furniture", day: 1, product: "Desk" },
  { _id: 5, category: "Furniture", day: 2, product: "Chair" }
]);
```

クエリの例

次の例では、ドキュメントを でパーティション化しcategory、各パーティションを でソートしてday、パーティションの先頭productから現在のドキュメントまでの値の実行リストを返します。

```
db.purchases.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$category",
      sortBy: { day: 1 },
      output: {
        productsSoFar: {
          $push: "$product",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "category": "Electronics", "day": 1, "product": "Laptop",
 "productsSoFar": [ "Laptop" ] },
  { "_id": 2, "category": "Electronics", "day": 2, "product": "Mouse", "productsSoFar":
 [ "Laptop", "Mouse" ] },
  { "_id": 3, "category": "Electronics", "day": 3, "product": "Keyboard",
 "productsSoFar": [ "Laptop", "Mouse", "Keyboard" ] },
  { "_id": 4, "category": "Furniture", "day": 1, "product": "Desk", "productsSoFar":
 [ "Desk" ] },
  { "_id": 5, "category": "Furniture", "day": 2, "product": "Chair", "productsSoFar":
 [ "Desk", "Chair" ] }
]
```

各ドキュメントはproductsSoFar、パーティション内のすべてのproduct値の配列である で、最初から現在のドキュメントまでのソート順で拡張されます。

コードの例

$push 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group) とウィンドウ演算子の使用 (の場合) の両方を示しています$setWindowFields。

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');

  // Accumulator usage: collect all values per group
  const sales = db.collection('sales');
  await 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 }
  ]);
  const accumulatorResult = await sales.aggregate([
```

```
    {
      $group: {
        _id: "$category",
        products: { $push: "$product" }
      }
    }
  ]).toArray();
  console.log('Accumulator result:', accumulatorResult);

  // Window operator usage: running list of values within each partition
  const purchases = db.collection('purchases');
  await purchases.insertMany([
    { _id: 1, category: "Electronics", day: 1, product: "Laptop" },
    { _id: 2, category: "Electronics", day: 2, product: "Mouse" },
    { _id: 3, category: "Electronics", day: 3, product: "Keyboard" },
    { _id: 4, category: "Furniture", day: 1, product: "Desk" },
    { _id: 5, category: "Furniture", day: 2, product: "Chair" }
  ]);
  const windowResult = await purchases.aggregate([
    {
      $setWindowFields: {
        partitionBy: "$category",
        sortBy: { day: 1 },
        output: {
          productsSoFar: {
            $push: "$product",
            window: { documents: ["unbounded", "current"] }
          }
        }
      }
    }
  ]).toArray();
  console.log('Window result:', windowResult);

  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']

    # Accumulator usage: collect all values per group
    sales = db['sales']
    sales.insert_many([
        { '_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 }
    ])
    accumulator_result = list(sales.aggregate([
        {
            '$group': {
                '_id': '$category',
                'products': { '$push': '$product' }
            }
        }
    ]))
    print('Accumulator result:', accumulator_result)

    # Window operator usage: running list of values within each partition
    purchases = db['purchases']
    purchases.insert_many([
        { '_id': 1, 'category': 'Electronics', 'day': 1, 'product': 'Laptop' },
        { '_id': 2, 'category': 'Electronics', 'day': 2, 'product': 'Mouse' },
        { '_id': 3, 'category': 'Electronics', 'day': 3, 'product': 'Keyboard' },
        { '_id': 4, 'category': 'Furniture', 'day': 1, 'product': 'Desk' },
        { '_id': 5, 'category': 'Furniture', 'day': 2, 'product': 'Chair' }
    ])
    window_result = list(purchases.aggregate([
        {
            '$setWindowFields': {
                'partitionBy': '$category',
                'sortBy': { 'day': 1 },
                'output': {
                    'productsSoFar': {
                        '$push': '$product',
                        'window': { 'documents': ['unbounded', 'current'] }
```

```
                    }
                }
            }
        }
    ]))
    print('Window result:', window_result)

    client.close()

example()
```

$project

Amazon DocumentDB の $project演算子を使用すると、出力ドキュメントにフィールドを選択的に含めたり除外したり、値を次のパイプラインステージに渡したり、入力ドキュメント値から新しいフィールドを計算したりできます。

パラメータ

- field: 出力ドキュメントに含めるか除外するフィールドは、フィールドパス (例：「a.b.c」) にすることができます。
- 1 または true: 出力に フィールドが含まれます。
- 0 または false: 出力から フィールドを除外します。

例 (MongoDB シェル)

次の例は、学生コレクションでの $project演算子の使用を示しています。

サンプルドキュメントを作成する

```
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" }
  ]);
```

このクエリには、出力の フィールドnameと mathフィールドのみが含まれます。_id フィールドは、明示的に除外されない限り、デフォルトで含まれます。

```
db.students.aggregate([
  { $project: { "name": 1, "math": 1 } }
])
```

出力

```
{ _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 }
```

このクエリでは、 gradeおよび _idフィールドが出力から除外され、他のすべてのフィールド (name、math、) が表示されますscience。

```
db.students.aggregate([
  { $project: { "grade": 0, "_id": 0 } }
])
```

出力

```
{ 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 }
```

コードの例

$project コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の`$radiansToDegrees`演算子は、値をラジアンから度に変換します。

パラメータ

- `expression`: ラジアン単位の数値に解決される式。

例 (MongoDB シェル)

次の例は、 $radiansToDegrees演算子を使用してラジアン値を度に変換する方法を示しています。

サンプルドキュメントを作成する

```
db.angles.insertMany([
  { "_id": 1, "radians": 0 },
  { "_id": 2, "radians": 1.5707963267948966 },
  { "_id": 3, "radians": 3.141592653589793 },
  { "_id": 4, "radians": 6.283185307179586 }
]);
```

クエリの例

```
db.angles.aggregate([
  { $project: {
    "degrees": { $radiansToDegrees: "$radians" }
  }}
]);
```

出力

```
[
  { "_id": 1, "degrees": 0 },
  { "_id": 2, "degrees": 90 },
  { "_id": 3, "degrees": 180 },
  { "_id": 4, "degrees": 360 }
]
```

コードの例

$radiansToDegrees 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0 の新機能

Amazon DocumentDB の $rand演算子は、0～1 の乱数を生成するために使用されます。

パラメータ

なし

例 (MongoDB シェル)

次の例は、 $rand演算子を使用してtempコレクションから 2 つのドキュメントをランダムに選択する方法を示しています。

サンプルドキュメントを作成する

```
db.items.insertMany([
  { "name": "pencil", "quantity": 110 },
  { "name": "pen", "quantity": 159 }
])
```

クエリの例

```
db.items.aggregate([
  {
    $project: {
      randomValue: { $rand: {} }
    }
  }
])
```

出力

```
[
  {
    _id: ObjectId('6924a5edd66dcae121d29517'),
    randomValue: 0.8615243955294392
  },
  {
    _id: ObjectId('6924a5edd66dcae121d29518'),
```

```
    randomValue: 0.22815483022099903
  }
]
```

コードの例

$rand コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$range集計演算子は、指定された範囲内の連続した数値の配列を作成するために使用されます。この演算子は、次の例に示すように、レースの補助ステーションのマイルマーカーなど、一連の数値を生成するのに特に便利です。

パラメータ

- start: 範囲の開始値。
- end: 範囲の終了値。
- step: (オプション) 範囲の生成時に使用するステップ値。指定しない場合、デフォルトのステップ値は 1 です。

例 (MongoDB シェル)

この例では、 $range演算子を使用して、自転車レースのウォーターステーションのマイルマーカーを生成します。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.races.aggregate([
  {
    $project: {
      race: 1,
      "waterStations": { $range: [20, "$distance", 20] }
    }
  }
]);
```

出力

```
[
  {
    _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 ] }
]
```

コードの例

$range コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

$rank

バージョン 8.0.2 の新機能。

Amazon DocumentDB の$rankウィンドウ演算子は、$setWindowFieldsステージのsortBy順序に従って、パーティション内の各ドキュメントの位置を返します。同じsortBy値を持つドキュメントは同じランクを受け取り、次の個別の値は介入位置をスキップしてギャップを残します (例: 1, 2, 2, 4)。$setWindowFields ステージの outputフィールドで指定します。

これは、$setWindowFieldsステージでのみ使用されるウィンドウ演算子です。

パラメータ

- $rank は空のドキュメントを値{}として受け取ります。
- window フレームは受け付けません。

- $setWindowFields ステージには 1 つのsortByキーのみが必要です。

例 (MongoDB シェル)

次の例では、ドキュメントを でパーティション化しgame、各パーティションscoreを降順にソートして、各プレイヤーにランクを割り当てます。

サンプルドキュメントを作成する

```
db.players.insertMany([
  { _id: 1, game: "A", player: "Alice", score: 90 },
  { _id: 2, game: "A", player: "Bob", score: 85 },
  { _id: 3, game: "A", player: "Carol", score: 85 },
  { _id: 4, game: "A", player: "Dave", score: 70 },
  { _id: 5, game: "B", player: "Erin", score: 95 },
  { _id: 6, game: "B", player: "Frank", score: 80 }
]);
```

クエリの例

```
db.players.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$game",
      sortBy: { score: -1 },
      output: {
        rank: { $rank: {} }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "game": "A", "player": "Alice", "score": 90, "rank": 1 },
  { "_id": 2, "game": "A", "player": "Bob", "score": 85, "rank": 2 },
  { "_id": 3, "game": "A", "player": "Carol", "score": 85, "rank": 2 },
  { "_id": 4, "game": "A", "player": "Dave", "score": 70, "rank": 4 },
  { "_id": 5, "game": "B", "player": "Erin", "score": 95, "rank": 1 },
  { "_id": 6, "game": "B", "player": "Frank", "score": 80, "rank": 2 }
```

```
]
```

各gameパーティション内で、プレイヤーは降順でランク付けされますscore。Bob と Carol は で結合85し、ランク を共有するため2、次のランクは Dave 4の にスキップされます。

コードの例

$setWindowFields ステージで $rank演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 players = db.collection('players');

    await players.insertMany([
      { _id: 1, game: "A", player: "Alice", score: 90 },
      { _id: 2, game: "A", player: "Bob", score: 85 },
      { _id: 3, game: "A", player: "Carol", score: 85 },
      { _id: 4, game: "A", player: "Dave", score: 70 },
      { _id: 5, game: "B", player: "Erin", score: 95 },
      { _id: 6, game: "B", player: "Frank", score: 80 }
    ]);

    const result = await players.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$game",
          sortBy: { score: -1 },
          output: {
            rank: { $rank: {} }
          }
        }
      }
```

```
    ]).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']
        players = db['players']

        players.insert_many([
            { '_id': 1, 'game': 'A', 'player': 'Alice', 'score': 90 },
            { '_id': 2, 'game': 'A', 'player': 'Bob', 'score': 85 },
            { '_id': 3, 'game': 'A', 'player': 'Carol', 'score': 85 },
            { '_id': 4, 'game': 'A', 'player': 'Dave', 'score': 70 },
            { '_id': 5, 'game': 'B', 'player': 'Erin', 'score': 95 },
            { '_id': 6, 'game': 'B', 'player': 'Frank', 'score': 80 }
        ])

        result = list(players.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$game',
                    'sortBy': { 'score': -1 },
                    'output': {
                        'rank': { '$rank': {} }
                    }
                }
            }
        ]))
```

```
        print(result)
    finally:
        client.close()

example()
```

$redact

Amazon DocumentDB の$redact集約ステージは、指定されたフィールドの値に基づいて出力ドキュメントにコンテンツを選択的に含めたり除外したりするために使用します。これは、アクセスレベルまたはユーザーのアクセス許可に基づいて機密データの可視性を制御する必要があるシナリオで特に役立ちます。

パラメータ

- $cond: ドキュメント内の各フィールド$$DESCENDについて、$$KEEP、$$PRUNE、または のいずれかに評価される式。
- $$KEEP: 出力ドキュメントの現在のフィールドを保持します。
- $$PRUNE: 出力ドキュメントから現在のフィールドを削除します。
- $$DESCEND: オブジェクトまたは配列である現在のフィールドに$redactステージを再帰的に適用します。

例 (MongoDB シェル)

この例では、ステージを使用してステータスに基づいて注文をフィルタリング$redactし、特定のステータス値を持つ注文のみを表示します。

サンプルドキュメントを作成する

```
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" }
])
```

クエリの例

```
db.orders.aggregate([
  {
    $redact: {
      $cond: {
        if: { $in: ["$status", ["shipped", "processing"]] },
        then: "$$KEEP",
        else: "$$PRUNE"
      }
    }
  }
])
```

出力

```
[
  { _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' }
]
```

この例では、$redactステージは各ドキュメントの statusフィールドの値をチェックします。status が「出荷済み」または「処理中」の場合、ドキュメントは保持されます ($$KEEP)。それ以外の場合、ドキュメントはプルーニングされます ($$PRUNE)。

コードの例

$redact コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$reduce集計演算子は、配列の要素に 2 つの引数の関数を累積的に適用して、配列を 1 つの値に減らすために使用されます。この演算子は、集計パイプライン内の配列データに対して複雑な計算または変換を実行する場合に特に便利です。

パラメータ

- input: 削減する配列。
- initialValue: 削減オペレーションで使用される初期値。
- in: input配列の各要素で評価される式。この式は、次のリダクションの反復で使用される値を返す必要があります。

例 (MongoDB シェル)

次の例は、 $reduce演算子を使用して配列内のすべての要素の合計を計算する方法を示しています。

サンプルドキュメントを作成する

```
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] }
])
```

クエリの例

```
db.orders.aggregate([
  {
    $project: {
      total: {
        $reduce: {
          input: "$items",
          initialValue: 0,
```

```
          in: { $add: ["$$value", "$$this"] }
        }
      }
    }
  }
])
```

出力

```
[
  { "_id": 1, "total": 15 },
  { "_id": 2, "total": 60 },
  { "_id": 3, "total": 80 },
  { "_id": 4, "total": 300 }
]
```

$reduce 演算子はitems配列を繰り返し、各要素を 0 initialValueの に追加します。結果は、配列内のすべての要素の合計です。

コードの例

$reduce コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

Node.js

Node.js アプリケーションで $reduce演算子を使用する例を次に示します。

```
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

Python アプリケーションで $reduce演算子を使用する例を次に示します。

```
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

バージョン 5.0 の新機能。

Elastic クラスターではサポートされていません。

Amazon DocumentDB の $regexFind演算子は、ドキュメント内の文字列フィールドで正規表現マッチングを実行するために使用されます。これにより、特定の正規表現パターンに一致する特定の部分文字列を検索して抽出できます。

パラメータ

- input: 検索する文字列フィールドまたは式。
- regex: 一致する正規表現パターン。
- options: (オプション) 大文字と小文字の区別や複数行の一致など、正規表現のオプションパラメータを指定するオブジェクト。サポートされているオプションは、 i (大文字と小文字を区別しない) と m (複数行) です。

例 (MongoDB シェル)

次の例は、 $regexFind演算子を使用して、 nameフィールドが特定の正規表現パターンと一致するドキュメントを検索する方法を示しています。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.users.aggregate([
```

```
  {
    $project: {
      names: {
        $regexFind: { input: '$name', regex: 'j', options: 'i' }
      }
    }
  },
  { $match: {names: {$ne: null}}}
])
```

このクエリは、 nameフィールドに「j」という文字が含まれているすべてのドキュメントを返します (大文字と小文字は区別されません)。

出力

```
[
  { _id: 1, names: { match: 'J', idx: 0, captures: [] } }
]
```

注: クエリで Amazon DocumentDB プランナーバージョン 1 を使用している場合は、ヒントを使用してインデックスを使用する必要があります。ヒントがない場合、クエリはコレクションスキャンを実行する場合があります。プランナーのバージョンを確認し、ヒントの使用の詳細については、「」を参照してくださいクエリプランナー v2。

コードの例

$regexFind コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

5.0 で導入

Amazon DocumentDB の $regexFindAll演算子は、ドキュメント内の文字列フィールドで正規表現マッチングを実行するために使用されます。これにより、特定の正規表現パターンに一致する特定の部分文字列を検索して抽出し、正規表現のすべての一致を返すことができます。

パラメータ

- input: 検索する文字列フィールドまたは式。

- regex: 一致する正規表現パターン。
- options: (オプション) 大文字と小文字の区別や複数行の一致など、正規表現のオプションパラメータを指定するオブジェクト。サポートされているオプションは、 i (大文字と小文字を区別しない) と m (複数行) です。

例 (MongoDB シェル)

次の例は、 $regexFindAll演算子を使用して emailフィールドからすべての文字シーケンスを抽出する方法を示しています。

サンプルドキュメントを作成する

```
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" }

]);
```

クエリの例

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      emailMatches: {
        $regexFindAll: { input: '$email', regex: '[a-z]+', options: 'i' }
      }
    }
  }
])
```

出力

```
[
  {
    _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: クエリで Amazon DocumentDB プランナーバージョン 1 を使用している場合は、ヒントを使用してインデックスを使用する必要があります。ヒントがない場合、クエリはコレクションスキャンを実行する場合があります。プランナーのバージョンを確認し、ヒントの使用の詳細については、「」を参照してくださいクエリプランナー v2。

コードの例

$regexFindAll コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

Node.js

Node.js アプリケーションで $regexFind演算子を使用する例を次に示します。

```
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

Python アプリケーションで $regexFind演算子を使用する例を次に示します。

```
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

バージョン 5.0 の新機能。Elastic クラスターではサポートされていません。

Amazon DocumentDB の $regexMatch演算子は、文字列フィールドで正規表現マッチングを実行するために使用されます。入力文字列が指定されたパターンと一致するかどうかを示すブール値 (true または false) を返します。

パラメータ

- input: 正規表現に対してテストする文字列。
- regex: 一致する正規表現パターン。
- options: (オプション) 大文字と小文字を区別しないマッチング () や複数行マッチング (i) など、正規表現の動作を変更するフラグm。

例 (MongoDB シェル)

次の例は、 $regexMatch演算子を使用して、名前が文字「M」で始まるかどうかを確認する方法を示しています。演算子は、ドキュメントfalseごとに trueまたは を返します。

サンプルドキュメントを作成する

```
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" },

]);
```

クエリの例

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      startsWithM: {
        $regexMatch: {
          input: "$name",
          regex: "^M",
          options: "i"
        }
      }
    }
  }
]);
```

出力

```
{ _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 }
```

コードの例

$regexMatch コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

5.0 で導入

Amazon DocumentDB の `$replaceAll`演算子は、フィールド内の指定された文字列パターンのすべての出現を新しい文字列に置き換えるために使用されます。この演算子は、データの正規化、テキストクリーニング、文字列操作などのタスクに役立ちます。

パラメータ

- `input`: 置き換える文字列を含むフィールドまたは式。
- `find`: 検索して置き換える文字列パターン。
- `replacement`: 一致した出現を置き換える文字列。

例 (MongoDB シェル)

次の例は、集約パイプラインで `$replaceAll`演算子を使用して、文字列「Chocolatier」のすべての出現を「Chocolate Co.」に置き換える方法を示しています。「製品」コレクションのbrandName」フィールド。

サンプルドキュメントを作成する

```
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
  }
]);
```

クエリの例

```
db.products.aggregate([
  {
    $addFields: {
      "brandName": {
        $replaceAll: {
          input: "$brandName",
          find: "Chocolatier",
          replacement: "Chocolate Co."
        }
      }
    }
  }
])
```

出力

```
[
  {
    _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
  }
]
```

コードの例

$replaceAll コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

5.0 で導入

Amazon DocumentDB の $replaceOne演算子は、文字列内の指定された部分文字列の最初の出現を置換文字列に置き換えるために、集約パイプライン内で使用される文字列式演算子です。この演算子では大文字と小文字が区別され、見つかった最初の一致のみが置き換えられます。

パラメータ

- input: 検索を実行する文字列 (フィールド) 。
- find: 入力内で検索する文字列。
- replacement: input(field) 内の検索の最初の出現を置き換える文字列。

例 (MongoDB シェル)

次の例は、集約パイプライン内で $replaceOne演算子を使用して、製品名の部分文字列を置き換える方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

集約の例

```
db.products.aggregate([
  {
    $addFields: {
      standardizedName: {
        $replaceOne: {
          input: "$name",
          find: "Pack",
          replacement: "Package"
        }
      }
    }
  }
]);
```

出力

出力は、各製品名の「Pack」の最初の出現のみが「Package」に置き換えられたことを示しています。

```
[
  {
    _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"
```

```
}
```

コードの例

$replaceOne コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、ルートドキュメントを指定された埋め込みドキュメントに置き換えるために使用されます。これは、ネストされたドキュメントを最上位に昇格させたり、データ出力を再構築したりする場合に便利です。

パラメータ

- newRoot: 既存のルートドキュメントを置き換える新しいルートドキュメント。

例 (MongoDB シェル)

この例では、顧客の注文から配送先住所情報を抽出する方法を示します。これは、配送ラベルまたは住所リストを生成するのに役立ちます。

サンプルドキュメントを作成する

```
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 } ])
```

クエリの例

```
db.orders.aggregate([
  {
    $replaceRoot: {
      newRoot: "$shippingAddress"
    }
  }
])
```

出力

```
{
    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'
  }
```

コードの例

$replaceRoot コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0 の新機能

Elastic クラスターではサポートされていません。

Amazon DocumentDB の$replaceWith集約ステージは、入力ドキュメントを新しいドキュメントに置き換えるために使用されます。_id フィールドを含む入力ドキュメントのすべての既存のフィールドは、新しいドキュメントに置き換えられます。 $replaceWithは一般的に、ドキュメントをフラット化したり、埋め込みドキュメントを最上位レベルに昇格させたりするために使用されます。

パラメータ

- <replacement> (必須): 既存のドキュメントを置き換える新しいドキュメント。

例 (MongoDB シェル)

次の例は、 $replaceWith演算子を使用して Amazon DocumentDB コレクション内の既存のドキュメントを置き換える方法を示しています。

サンプルドキュメントを作成する

```
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 ]
  }
]);
```

クエリの例

```
db.restaurants.aggregate([
  { $replaceWith: {
      name: "$name",
      cuisine: "$cuisine",
      rating: { $avg: "$ratings" }
    }
  }
]);
```

出力

```
[
  {
    name: 'Biryani Adda',
    cuisine: 'Indian',
    rating: 3.4
  },
  {
    name: 'The Burger Spot',
    cuisine: 'American',
    rating: 2.7777777777777777
  }
]
```

コードの例

$replaceWith コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $reverseArray演算子は、配列の要素を指定された順序で反転させるために使用されます。この演算子は、配列の要素を逆方向に並べ替える必要がある場合に便利です。

パラメータ

- expression: 反転する配列式。

例 (MongoDB シェル)

次の例は、 $reverseArray演算子を使用して配列内の要素の順序を反転する方法を示しています。

サンプルドキュメントを作成する

```
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]}
]);
```

クエリの例

```
db.miles.aggregate([
  {
    $project: {
```

```
      _id: 1,
      member_since: 1,
      credit_card: 1,
      reversed_flight_miles: { $reverseArray: "$flight_miles" }
    }
  }
]);
```

出力

```
{ "_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 ] }
```

この例では、 $reverseArray演算子を使用してflight_miles配列の順序を逆にします。出力の結果reversed_flight_milesフィールドには、配列の要素が逆順に表示されます。

コードの例

$reverseArray コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

Node.js

Node.js アプリケーションで $reverseArray演算子を使用する例を次に示します。

```
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

Python アプリケーションで $reverseArray演算子を使用する例を次に示します。

```
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

バージョン 8.0.1 から新規。

Amazon DocumentDB の$round演算子は、数値を指定された小数点以下を切り捨てます。その場所にある 2 つの最も近い値のちょうど中間の値になると、 は最も近い偶数に$round四捨五入します。たとえば、2.5 は 2 に、3.5 は 4 に切り上げられます。

パラメータ

- number: 数値に解決される式。
- place: オプション。小数点以下桁数を指定する -20 ~ 100 の整数式。負の値は 10 進数の左に丸められます。デフォルトは 0 です。

例 (MongoDB シェル)

次の例は、 $round演算子を使用して数値を小数点以下第 1 位に四捨五入する方法を示しています。

正のプレース値での四捨五入

```
db.measurements.insertMany([
  {_id: 1, value: 3.456},
  {_id: 2, value: 7.891},
  {_id: 3, value: 12.345}
]);
```

クエリの例

```
db.measurements.aggregate([
  { $project: { rounded: { $round: ["$value", 1] } } }
]);
```

出力

```
[
  {_id: 1, rounded: 3.5},
  {_id: 2, rounded: 7.9},
  {_id: 3, rounded: 12.3}
]
```

中間値を最も近い偶数に四捨五入する

値がターゲットの場所にある 2 つの最も近い値のちょうど中間になると、 は常に$round切り上げられるのではなく、最も近い偶数に切り上げられます。次の例では、中間値を整数に丸めます。

```
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] } } }
]);
```

出力

```
[
  {_id: 1, rounded: 10},
  {_id: 2, rounded: 12},
  {_id: 3, rounded: 12},
  {_id: 4, rounded: 14}
]
```

10.5 は 10と のちょうど中間であるため11、最も近い偶数値 に丸められます10。同様に、 11.5と12.5はどちらも最も近い偶数値 に四捨五入され12、 は に13.5四捨五入されます14。

負のプレース値での四捨五入

place が負の場合、 は 10 進数の左に$round丸めます。次の例では、最初の桁を使用して 10 進数の左に値を四捨五入します。

```
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] } } }
]);
```

出力

```
[
  {_id: 1, rounded: 20},
  {_id: 2, rounded: 30},
  {_id: 3, rounded: 30}
]
```

コードの例

$round 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能。

Elastic クラスターではサポートされていません。

Amazon DocumentDB の $rtrim演算子は、文字列から末尾の文字を削除するために使用されます。デフォルトでは、末尾の空白文字は削除されますが、chars 引数を渡すことで削除する文字のセットを指定することもできます。

パラメータ

- input: 末尾の空白文字を削除する入力文字列。
- chars: (オプション) 特定の文字を削除します。

例 (MongoDB シェル)

次の例は、 $rtrim演算子を使用して文字列から末尾の指定された文字 (「 *」) を削除する方法を示しています。

サンプルドキュメントを作成する

```
db.collection.insert([
  { "name": "John Doe*  ", "age": 30 },
  { "name": "Jane Smith  ", "age": 25 },
  { "name": "Bob Johnson", "age": 35 }
```

```
]);
```

クエリの例

```
db.collection.aggregate([
  {
    $project: {
      _id: 0,
      name: { $rtrim: { input: "$name", chars: " *" } },
      age: 1
    }
  }
]);
```

出力

```
[
  { age: 30, name: 'John Doe' },
  { age: 25, name: 'Jane Smith' },
  { age: 35, name: 'Bob Johnson' }
]
```

コードの例

$rtrim コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$sample集約ステージは、コレクションから指定された数のドキュメントをランダムに選択するために使用されます。これは、データ分析、テスト、さらに処理するためのサンプルの生成などのタスクに役立ちます。

パラメータ

- size: ランダムに選択するドキュメントの数。

例 (MongoDB シェル)

次の例は、 $sampleステージを使用してtempコレクションから 2 つのドキュメントをランダムに選択する方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.temp.aggregate([
   { $sample: { size: 2 } }
])
```

出力

```
{ "_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") }
```

結果が示すように、10 件のドキュメントのうち 2 件がランダムにサンプリングされました。これらのドキュメントを使用して平均を決定したり、最小/最大計算を実行したりできます。

コードの例

$sample コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 5.0 の新機能。

Amazon DocumentDB の $search演算子は、テキスト検索機能を提供するために使用されます。

パラメータ

なし

例 (MongoDB シェル)

次の例は、 $search演算子を使用してテキスト検索クエリを実行する方法を示しています。

サンプルドキュメントを作成する

```
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." }
]);
```

クエリの例

```
db.textcollection.find(
  { $text: { $search: "John" } }
);
```

出力

```
[
  {
    _id: 1,
    name: 'John Doe',
    description: 'This is a sample document about John Doe.'
  }
]
```

コードの例

$search コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $second演算子は、日付またはタイムスタンプから秒コンポーネントを抽出します。これは、日付またはタイムスタンプフィールドから秒値を取得するために使用されます。

パラメータ

- expression: 秒値を抽出する日付またはタイムスタンプフィールド。この式は、フィールドパスでも、日付またはタイムスタンプに解決される有効な式でもかまいません。

例 (MongoDB シェル)

次の例は、 $second演算子を使用して日付フィールドから秒コンポーネントを抽出する方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.users.aggregate([{ $project: { name: 1, dobSeconds: { $second: "$dob" } } }])
```

出力

```
[
  { "_id" : ObjectId("6089a9c306a829d1f8b456a1"), "name" : "John", "dobSeconds" : 45 },
  { "_id" : ObjectId("6089a9c306a829d1f8b456a2"), "name" : "Jane", "dobSeconds" : 59 },
  { "_id" : ObjectId("6089a9c306a829d1f8b456a3"), "name" : "Bob", "dobSeconds" : 0 }
]
```

コードの例

$second コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0 の新機能

Elastic クラスターではサポートされていません。

Amazon DocumentDB の$set集約ステージでは、集約パイプライン中にドキュメントに新しいフィールドを追加したり、既存のフィールド値を更新したりできます。

パラメータ

- expression: 評価する式。これは、フィールド参照や算術演算など、任意の有効な集計式にすることができます。

例 (MongoDB シェル)

次の例は、$set集計ステージを使用して、 quantityフィールドに priceフィールドを乗算して合計を計算する方法を示しています。

サンプルドキュメントを作成する

```
db.inventory.insertMany([
  { item: "pencil", quantity: 100, price: 0.24},
  { item: "pen", quantity: 204, price: 1.78 }
]);
```

集約の例

```
db.inventory.aggregate([
  {
    $set: {
      total: { $multiply: ["$quantity", "$price"] }
    }
  }
])
```

出力

```
[
  {
    _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
  }
]
```

コードの例

$set コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能。

Amazon DocumentDB の $setDifference演算子は、2 つのセットを比較し、最初のセットにはあるが、2 番目のセットにはない要素を返すために使用されます。この演算子は、2 つのセット間で一意の要素を見つけるのに役立ちます。

パラメータ

- firstSet : 比較する最初のセット。
- secondSet : 比較する 2 番目のセット。

例 (MongoDB シェル)

次の例は、 $setDifference演算子を使用して 2 つのセット間の一意の要素を検索する方法を示しています。

サンプルドキュメントを作成する

```
db.collection.insertMany([
  { _id: 1, fruits: ["apple", "banana", "cherry", "date"] },
  { _id: 2, fruits: ["banana", "cherry", "date", "elderberry"] }
]);
```

クエリの例

```
db.collection.aggregate([
  {
    $project: {
      uniqueFruits: { $setDifference: ["$fruits", ["banana", "cherry", "date"]] }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "uniqueFruits": ["apple"] },
  { "_id": 2, "uniqueFruits": ["elderberry"] }
]
```

クエリは次のステップを実行します。

1. $project ステージを使用して、ドキュメントuniqueFruitsごとに新しいフィールドを作成します。

2. $setDifference 演算子はfruits配列を配列と比較["banana", "cherry", "date"]し、fruits配列内の一意の要素を返します。

コードの例

$setDifference コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

Node.js

Node.js アプリケーションで $setDifference演算子を使用する方法の例を次に示します。

```
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

Python アプリケーションで $setDifference演算子を使用する方法の例を次に示します。

```
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 の $setEquals演算子は、2 つのセットが等しいかどうかを判断するために使用されます。2 つの配列を比較し、順序に関係なく同じ異なる要素が含まれているtrue場合は を返します。

パラメータ

- expression1: 比較する最初の配列。
- expression2: 比較する 2 番目の配列。

例 (MongoDB シェル)

次の例は、 $setEquals演算子を使用して 2 つの値のセットを比較する方法を示しています。

サンプルドキュメントを作成する

```
db.collection.insertMany([
  { _id: 1, fruits: ["apple", "banana", "cherry"] },
  { _id: 2, fruits: ["banana", "apple", "cherry"] },
  { _id: 3, fruits: ["apple", "banana", "orange"] }
])
```

クエリの例

```
db.collection.find({
  $expr: {
    $setEquals: ["$fruits", ["apple", "banana", "cherry"]]
```

```
  }
})
```

出力

```
{ "_id" : 1, "fruits" : [ "apple", "banana", "cherry" ] }
{ "_id" : 2, "fruits" : [ "banana", "apple", "cherry" ] }
```

クエリは $setEquals演算子を使用して、各ドキュメントの fruitsフィールドを配列 と比較します["apple", "banana", "cherry"]。fruits フィールドが比較配列と等しいドキュメントが返されます。

コードの例

$setEquals コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $setIntersection演算子は、2 つ以上の配列間の共通要素を返すために使用されます。この演算子は、データセットを操作するときに特に役立ち、複数のセットの交差を見つけることができます。

パラメータ

- array1: 交差する最初の配列。
- array2: 交差する 2 番目の配列。
- arrayN: (オプション) 交差する追加の配列。

例 (MongoDB シェル)

次の例は、 $setIntersection演算子を使用して 2 つの配列間の共通要素を検索する方法を示しています。

サンプルドキュメントを作成する

```
db.collection.insertMany([
  { _id: 1, colors: ["red", "blue", "green"] },
  { _id: 2, colors: ["blue", "yellow", "orange"] },
  { _id: 3, colors: ["red", "green", "purple"] }
])
```

クエリの例

```
db.collection.aggregate([
  { $project: {
      _id: 1,
      commonColors: { $setIntersection: ["$colors", ["red", "blue", "green"]] }
    }
  }
])
```

出力

```
[
  { "_id": 1, "commonColors": ["red", "blue", "green"] },
  { "_id": 2, "commonColors": ["blue"] },
  { "_id": 3, "commonColors": ["red", "green"] }
]
```

コードの例

$setIntersection コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $setIsSubset演算子は、値のセットが別のセットのサブセットかどうかを判断するために使用されます。配列フィールドでセットベースの比較とオペレーションを実行するのに役立ちます。

パラメータ

- field: $setIsSubset演算子を適用するフィールド。
- set: フィールドを比較する セット。

例 (MongoDB シェル)

次の例は、 フィールドが指定されたセットのtagsサブセットであるかどうかをチェックするための$setIsSubset演算子の使用を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.products.find({
  $expr: { $setIsSubset: [["tag1", "tag2"], "$tags"] }
})
```

注: $setIsSubsetは集計演算子であり、find() クエリで直接使用することはできません。この例では、 $exprはクエリ演算子と集計式の間のギャップを埋めfind( )るために とともに使用されます。

出力

```
[
  { "_id" : 1, "name" : "Product A", "tags" : [ "tag1", "tag2", "tag3" ] },
  { "_id" : 2, "name" : "Product B", "tags" : [ "tag1", "tag2" ] }
]
```

クエリは、 tagsフィールドがセット のサブセットであるドキュメントを返します["tag1", "tag2"]。

コードの例

$setIsSubset コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$setUnion集計演算子は、2 つ以上の値のセットを組み合わせて、入力セットのすべての一意の要素を含むセットを返すために使用されます。この演算子は、ドキュメントの配列フィールドに対してセットベースのオペレーションを実行する必要がある場合に便利です。

パラメータ

- expression1: 配列に解決される式。
- expression2: 配列に解決される式。
- expressionN: 配列に解決される追加の式 (オプション) 。

例 (MongoDB シェル)

次の例は、 $setUnion演算子を使用して、コレクション内の 2 つの配列フィールドの一意の要素を組み合わせる方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      allInterests: { $setUnion: ["$hobbies", "$skills"] }
    }
  }
]);
```

出力

```
[
  { "_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" ] }
]
```

この例では、 $setUnion演算子を使用して、各ユーザードキュメントの hobbiesおよび skills配列フィールドの一意の要素を組み合わせます。結果のallInterestsフィールドには、各ユーザーのすべての固有の趣味とスキルの結合が含まれます。

コードの例

$setUnion コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

$setWindowFields

バージョン 8.0.2 の新機能。

Amazon DocumentDB の$setWindowFieldsステージは、パーティション内のドキュメント (ウィンドウ) の指定されたスパンに対してオペレーションを実行し、選択したウィンドウ演算子に基づいて結果を返します。ステージは入力ドキュメントをパーティションにグループ化し、オプションでソートし、ドキュメントごとに周囲のドキュメントのウィンドウで 1 つ以上の出力フィールドを計算します。

[Syntax] (構文)

```
{
  $setWindowFields: {
    partitionBy: <expression>,
    sortBy: { <sortField1>: <sortOrder>, ... },
    output: {
      <outputField1>: {
        <windowOperator>: <expression>,
        window: {
          documents: [ <lowerBound>, <upperBound> ],
          range: [ <lowerBound>, <upperBound> ],
          unit: <time unit>
        }
      },
      ...
    }
  }
}
```

パラメータ

- partitionBy: オプション。ドキュメントをパーティションにグループ化するために使用される単一の式。これは、フィールドパス (など"$state") または別の式にすることができます。省略すると、すべての入力ドキュメントは 1 つのパーティションに属します。

- sortBy: オプション。各パーティションのドキュメントをソートするフィールドを{ <field>: 1 }昇順または{ <field>: -1 }降順で指定するドキュメント。ドキュメントベースまたは範囲ベースのウィンドウ境界を使用する場合、およびランク付け演算子のいずれかを使用する場合、sortByが必要です。
- output: 必須。各ドキュメントに追加するフィールドを指定するドキュメント。複数の出力フィールドを指定できますが、各出力フィールドには、ウィンドウの境界を定義するウィンドウ演算子、引数、およびオプションのwindowドキュメントが 1 つだけ含まれます{ <field>: { <windowOperator>: <expression>, window: { ... } } }。

ウィンドウの境界

各outputフィールド内では、オプションでwindowドキュメントを使用してウィンドウの境界を定義できます。ウィンドウは、ドキュメントの位置または値の範囲によって定義できます。

- documents: ウィンドウを、現在のドキュメントの前後のドキュメントの数として 形式で指定します[<lowerBound>, <upperBound>]。各境界は整数オフセット (例: -1 または 2)、またはキーワードの 1 つ "unbounded"および です"current"。 は、下限として使用する場合はパーティションの先頭に、上限として使用する場合はパーティションの末尾にウィンドウ"unbounded"を拡張します。 は現在のドキュメント"current"を意味します。両方の境界がsortByでない限り、documentsウィンドウには が必要です"unbounded"。
- range: ウィンドウを現在のドキュメントに対するsortByフィールド値の範囲として 形式で指定します[<lowerBound>, <upperBound>]。各バインドは、現在のドキュメントsortByの値、または"unbounded"キーワードに追加された数値です。range ウィンドウには 1 つの昇順sortByフィールドのみが必要で、"current"キーワードを受け入れません。
- unit: オプション。とともに使用rangeして、数値ウィンドウの境界を、sortByフィールドが日付値を保持する期間として解釈します。Amazon DocumentDB は、、"year"、、"quarter""month"、"week"、、"day"、"hour""minute""second"および の単位をサポートしています"millisecond"。unit を指定する場合、range境界は整数である必要があります。

window ドキュメントを省略すると、デフォルトのウィンドウは でdocuments: ["unbounded", "unbounded"]、パーティション全体に及びます。

> **Note**
>
> 日付の値を超えるrangeウィンドウを使用する場合は、カレンダーに依存する長さがあいまいなウィンドウ境界を生成する可能性がある "month"、"quarter"、または ではなく"year"、 "day" (またはそれより小さい) unitなどの固定期間を使用することをお勧めします。

サポートされているウィンドウ演算子

Amazon DocumentDB は、$setWindowFieldsステージの outputフィールドで次の 26 個のウィンドウ演算子をサポートしています。

- $sum
- $avg
- $count
- $push
- $addToSet
- $stdDevPop
- $stdDevSamp
- $covariancePop
- $covarianceSamp
- $first
- $last
- $firstN
- $lastN
- $top
- $bottom
- $topN
- $bottomN
- $min
- $minN
- $max

- $maxN
- $median
- $percentile
- $rank
- $denseRank
- $documentNumber

> **Note**
>
> $rand 演算子は、$setWindowFieldsステージのウィンドウ演算子内ではサポートされていません。

空のウィンドウ

ウィンドウにドキュメントが含まれていない場合 (範囲の値がないrangeウィンドウなど) 、結果は演算子によって異なります。 $count および は 0、$push、、$addToSet、$firstN、$lastN$minN、 $maxN$sumを返し$topN、空の配列$bottomNを返します。 $percentile はnull値の配列を返します (リクエストされたパーセンタイルごとに 1 つ) 。他のすべての演算子は を返しますnull。

例 (MongoDB シェル)

次の例ではregion、ドキュメントを でパーティション化し、各パーティションを でソートしday、パーティションの開始revenueから現在のドキュメントまでの の累計を計算します。

サンプルドキュメントを作成する

```
db.sales.insertMany([
  { _id: 1, region: "east", day: 1, revenue: 100 },
  { _id: 2, region: "east", day: 2, revenue: 150 },
  { _id: 3, region: "east", day: 3, revenue: 120 },
  { _id: 4, region: "west", day: 1, revenue: 200 },
  { _id: 5, region: "west", day: 2, revenue: 180 }
]);
```

クエリの例

```
db.sales.aggregate([
```

```
  {
    $setWindowFields: {
      partitionBy: "$region",
      sortBy: { day: 1 },
      output: {
        runningRevenue: {
          $sum: "$revenue",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "region": "east", "day": 1, "revenue": 100, "runningRevenue": 100 },
  { "_id": 2, "region": "east", "day": 2, "revenue": 150, "runningRevenue": 250 },
  { "_id": 3, "region": "east", "day": 3, "revenue": 120, "runningRevenue": 370 },
  { "_id": 4, "region": "west", "day": 1, "revenue": 200, "runningRevenue": 200 },
  { "_id": 5, "region": "west", "day": 2, "revenue": 180, "runningRevenue": 380 }
]
```

各ドキュメントはrunningRevenue、現在のドキュメントまでのパーティションrevenue内の の累積合計である で拡張されます。

コードの例

$setWindowFields ステージを使用するためのコード例を表示するには、使用する言語のタブを選択します。

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 sales = db.collection('sales');

    await sales.insertMany([
      { _id: 1, region: "east", day: 1, revenue: 100 },
      { _id: 2, region: "east", day: 2, revenue: 150 },
      { _id: 3, region: "east", day: 3, revenue: 120 },
      { _id: 4, region: "west", day: 1, revenue: 200 },
      { _id: 5, region: "west", day: 2, revenue: 180 }
    ]);

    const result = await sales.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$region",
          sortBy: { day: 1 },
          output: {
            runningRevenue: {
              $sum: "$revenue",
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).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']
        sales = db['sales']

        sales.insert_many([
            { '_id': 1, 'region': 'east', 'day': 1, 'revenue': 100 },
            { '_id': 2, 'region': 'east', 'day': 2, 'revenue': 150 },
            { '_id': 3, 'region': 'east', 'day': 3, 'revenue': 120 },
            { '_id': 4, 'region': 'west', 'day': 1, 'revenue': 200 },
            { '_id': 5, 'region': 'west', 'day': 2, 'revenue': 180 }
        ])

        result = list(sales.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$region',
                    'sortBy': { 'day': 1 },
                    'output': {
                        'runningRevenue': {
                            '$sum': '$revenue',
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))

        print(result)
    finally:
        client.close()

example()
```

$sigmoid

バージョン 8.0.1 から新規。

Amazon DocumentDB の $sigmoid演算子は、ロジスティックシグモイド関数 1 / (1 + e^(-x))を数値入力に適用します。この関数は、実際の数値を 0～1 の値にマッピングし、S 字曲線を生成します。これは、raw スコアを確率または関連性スコアに変換するなど、値を 0～1 の範囲に正規化する必要がある場合に便利です。

パラメータ

- expression: 数値に解決される式。

例 (MongoDB シェル)

次の例は、 $sigmoid演算子を使用してロジット値を確率に変換する方法を示しています。

サンプルドキュメントを作成する

```
db.predictions.insertMany([
  {_id: 1, logit: -2},
  {_id: 2, logit: 0},
  {_id: 3, logit: 2}
]);
```

クエリの例

```
db.predictions.aggregate([
  { $project: { probability: { $sigmoid: "$logit" } } }
]);
```

出力

```
[
  {_id: 1, probability: 0.11920292202211755},
  {_id: 2, probability: 0.5},
  {_id: 3, probability: 0.8807970779778823}
]
```

コードの例

$sigmoid 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 の新機能。

Amazon DocumentDB の $sin演算子は、ラジアンで測定される値のサインを返します。集約パイプラインでこれを使用して、数値フィールドで三角計算を実行します。

パラメータ

- expression: ラジアン単位で数値に解決される式。

例 (MongoDB シェル)

次の例は、 $sin演算子を使用して角度値のサインをラジアン単位で計算する方法を示しています。

サンプルドキュメントを作成する

```
db.angles.insertMany([
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 0.5235987755982988 },
  { "_id": 3, "angle": 1.5707963267948966 }
]);
```

クエリの例

```
db.angles.aggregate([
  { $project: {
    "sine": { $sin: "$angle" }
  }}
]);
```

出力

```
[
  { "_id": 1, "sine": 0 },
  { "_id": 2, "sine": 0.49999999999999994 },
  { "_id": 3, "sine": 1 }
]
```

コードの例

$sin 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 の新機能。

Amazon DocumentDB の $sinh演算子は、値のハイパーボリックサインを返します。集約パイプラインでこれを使用して、数値フィールドに対して双曲線三角計算を実行します。

パラメータ

- expression: 数値に解決される式。

戻り値の型はdoubleデフォルトで です。入力が 128 ビットの 10 進数の場合、出力も 128 ビットの 10 進数です。

例 (MongoDB シェル)

次の例は、 $sinh演算子を使用して数値の双曲線サインを計算する方法を示しています。

サンプルドキュメントを作成する

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 1 },
  { "_id": 3, "value": -1 }
]);
```

クエリの例

```
db.values.aggregate([
  { $project: {
    "result": { $sinh: "$value" }
  }}
]);
```

出力

```
[
  { "_id": 1, "result": 0 },
  { "_id": 2, "result": 1.1752011936438014 },
  { "_id": 3, "result": -1.1752011936438014 }
]
```

コードの例

$sinh 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

Amazon DocumentDB では、 $skip演算子を使用してクエリ結果の開始ポイントをオフセットし、一致するドキュメントの特定のサブセットを取得できます。これは、後続の結果ページを取得するページ分割シナリオで特に便利です。

パラメータ

- skip: 残りのドキュメントを返す前にスキップするドキュメントの数。

例 (MongoDB シェル)

次の例は、 $skip演算子を使用してコレクションから結果の 2 ページ目 (ドキュメント 11 ~ 20) を取得する方法を示しています。

サンプルドキュメントを作成する

```
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" }
```

```
]);
```

クエリの例

```
db.collection.find({}, { "name": 1 })
             .skip(10)
             .limit(10);
```

出力

```
[
  { "_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" }
]
```

コードの例

$skip コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 集計演算子を使用すると、配列の先頭または末尾から配列を横断して、配列のサブセットを返すことができます。これは、上位 N 項目や下位 N 項目など、配列フィールドから限られた数の項目を表示するために使用されます。

パラメータ

- array: スライスする配列フィールド。
- n: 返す要素の数を指定する整数。正の値は配列の先頭から始まり、負の値は配列の末尾から始まります。

例 (MongoDB シェル)

次の例は、 $sliceを使用して各シェフの最初の 2 つのお気に入りのお菓子を返す方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.sweets.aggregate([
  { $project: { _id: 0, name: 1, topTwoFavorites: { $slice: [ "$favorites", 2 ] } } }
]);
```

出力

```
[
  { name: 'Alvin', topTwoFavorites: [ 'chocolate', 'cake' ] },
  { name: 'Tom', topTwoFavorites: [ 'donuts', 'pudding' ] },
  { name: 'Jessica', topTwoFavorites: [ 'fudge', 'smores' ] },
  { name: 'Rachel', topTwoFavorites: [ 'ice cream' ] }
]
```

この例では、 $slice演算子を使用して、各ドキュメントのfavorites配列から最初の 2 つの要素を抽出します。

コードの例

$slice コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、配列フィールド内の項目の数を返すために使用されます。これは、ドキュメントに保存されている配列内の要素の数を決定するために使用できます。

パラメータ

- `field`: 配列サイズを返すフィールドパス。

例 (MongoDB シェル)

この例では、 $size演算子を使用して、各ユーザーがフォローしているチームの数を返す方法を示します。

サンプルドキュメントを作成する

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

クエリの例

```
db.profiles.aggregate([
  {
    $project: {
      _id: 0,
      name: 1,
      "numberOfTeams": { $size: "$teams" }
    }
  }
])
```

出力

```
[
  { name: 'John Doe', numberOfTeams: 3 },
  { name: 'Jane Smith', numberOfTeams: 2 },
  { name: 'Bob Johnson', numberOfTeams: 3 }
]
```

コードの例

$size コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 集約ステージは、指定されたフィールド値に基づいてパイプライン内のドキュメントを順序付けします。ドキュメントは、指定されたソート基準に従って昇順または降順に並べられます。

パラメータ

- field: ソートするフィールド名。
- order: 昇順-1または降順1に使用します。

例 (MongoDB シェル)

次の例は、 $sortステージを使用して製品を価格順に降順で注文する方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.products.aggregate([
  { $sort: { price: -1 } }
]);
```

出力

```
[
  { _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 }
]
```

コードの例

$sort 集約ステージを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

増分ソート

インクリメンタルソートは、Amazon DocumentDB がマルチフィールドソートの先頭フィールドを既に順序付けしているインデックスを使用する最適化です。その後、Amazon DocumentDB は、同じ先頭の値を共有するドキュメントの各グループ内の残りのフィールドのみをソートします。Amazon DocumentDB は、メモリ内の結果セット全体を 1 つのブロック (完全ソート) としてソートする代わりに、ドキュメントをインデックス順に読み取ります。次に、グループごとに 1 つずつ、多数の小さな独立したソートを実行します。結果はフルソートと同じです。変更を使用するのは、クエリプランとメモリ量と作業量のみです。

利点

増分ソートには次の利点があります。

- メモリ使用量が少なく、ディスクベースのソートスピルが少なくなります。Amazon DocumentDB は、結果セット全体を一度に注文するのではなく、一度に 1 つの小さなグループを注文します。
- を使用すると、より高速な結果が得られます**limit**。Amazon DocumentDB は、入力全体を読み取る前にソートされたドキュメントを出力できます。ソートの後に が続くと `limit` (トップ N やページ分割されたクエリなど) 、十分なドキュメントが生成されるとすぐに停止できます。

Amazon DocumentDB が増分ソートを選択する場合

たとえば、 のソート`{ customerId: 1, orderDate: -1 }`と のみのインデックスがある場合`{ customerId: 1 }`、インデックスは によって順序付けられたドキュメントを既に返しますcustomerId。Amazon DocumentDB は、 `customerId`によって同じ を共有するドキュメントの各グループのみを注文する必要がありますorderDate。既に順序付けされている先頭フィールド (customerId) は、事前ソートされたキーです。インデックスが既に提供しているソートの量に応じて、Amazon DocumentDB は 3 つの動作のいずれかを選択します。

- インデックスはすべてのソートキーを順序付けします。ソートは必要ありません。Amazon DocumentDB は、リクエストされた順序でドキュメントを直接読み取ります。
- インデックスはソートキーの先頭プレフィックスを順序付けします。Amazon DocumentDB は増分ソートを使用し、事前ソートされた各グループ内の末尾のキーのみを順序付けます。
- インデックスは先頭のソートキーを順序付けしません。Amazon DocumentDB は、入力全体に対してフルソートを実行します。

要件と制限事項

増分ソートには、プランナーバージョン 3.0 (Amazon DocumentDB 8.0.2 時点のデフォルトのクエリプランナー) を実行する Amazon DocumentDB 8.0.2 以降が必要です。プランナーのバージョンの詳細については、「」を参照してくださいクエリプランナー v3。

Amazon DocumentDB は、部分インデックスまたはマルチキー (配列) インデックスを使用して、増分ソートの事前ソートされたプレフィックスを指定することはできません。このような場合、Amazon DocumentDB は完全にフォールバックします。これは結果の正確性には影響しません。

を使用した増分ソートの検証 **explain()**

を使用してexplain()、クエリが増分ソートを使用していることを確認します。獲得プランでは、増分ソートがSORTステージとして表示され、インデックスが既に順序付けされている先頭のソートフィールドを一覧表示するpresortedKeys配列が含まれます。フルソートは、presortedKeys配列のないSORTステージとして表示されます。presortedKeys フィールドは出力への Amazon DocumentDB 固有の追加explain()であり、MongoDB の説明プランには表示されません。

```
db.orders.createIndex({ customerId: 1 })

db.orders.find({}).sort({ customerId: 1, orderDate: -1 }).explain()
```

```
{
    "stage": "SORT",
    "sortPattern": {
        "customerId": 1,
        "orderDate": -1
    },
    "presortedKeys": [
        "customerId"
    ],
```

```
    "inputStage": {
        "stage": "IXSCAN",
        "indexName": "customerId_1",
        "direction": "forward"
    }
}
```

$sortArray

バージョン 8.0 の新機能

Elastic クラスターではサポートされていません。

Amazon DocumentDB の $sortArray演算子は、配列の要素をソートします。スカラー値の配列の場合、配列は要素値でソートされます。ドキュメントの配列の場合、配列は指定されたフィールドでソートされます。演算子は、ソートされた配列を昇順 (1) または降順 (-1) で返します。

> **Note**
>
> $sortArray は、クエリプランナーバージョン 3 の集約パイプラインの$projectステージでサポートされています。

[Syntax] (構文)

```
{ $sortArray: { input: <array>, sortBy: <sort specification> } }
```

パラメータ

- input: ソートする配列。
- sortBy: ソート順。スカラー値の配列の場合は、昇順1に 、降順-1に を指定します。ドキュメントの配列には、フォームのドキュメントを指定する{ field: 1 }か、ソートする{ field: -1 }各フィールドを指定します。

例 (MongoDB シェル)

次の例は、 $sortArray演算子を使用して配列の要素を昇順でソートする方法を示しています。

サンプルドキュメントを作成する

```
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]}
]);
```

クエリの例

```
db.miles.aggregate([
  {
    $project: {
      _id: 1,
      member_since: 1,
      credit_card: 1,
      sorted_flight_miles: { $sortArray: { input: "$flight_miles", sortBy: 1 } }
    }
  }
]);
```

出力

```
{ "_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 ] }
```

この例では、$sortArray演算子はflight_miles配列を昇順でソートします。出力の結果sorted_flight_milesフィールドには、配列の要素がソート順に表示されます。

コードの例

$sortArray コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

Node.js

Node.js アプリケーションで $sortArray演算子を使用する例を次に示します。

```
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

Python アプリケーションで $sortArray演算子を使用する例を次に示します。

```
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

バージョン 8.0.1 から新規。

Amazon DocumentDB の$sortByCount集約ステージは、指定された式の値に基づいて受信ドキュメントをグループ化し、各個別のグループのドキュメントの数を計算して、結果をカウント別に降順にソートします。これは、$sumアキュムレータ$groupとそれに続くカウントフィールドの$sortステージに相当します。

パラメータ

- expression: グループ化する式。これは、フィールドパス (プレフィックスは $) または任意の有効な集計式にすることができます。

例 (MongoDB シェル)

次の例は、 $sortByCountステージを使用して、カテゴリフィールドでドキュメントをカウントおよびソートする方法を示しています。

サンプルドキュメントを作成する

```
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" }
```

```
]);
```

クエリの例

```
db.products.aggregate([
  { $sortByCount: "$category" }
]);
```

出力

```
[
  { "_id": "electronics", "count": 3 },
  { "_id": "food", "count": 2 },
  { "_id": "tools", "count": 1 }
]
```

結果は降順countで自動的にソートされます。

コードの例

$sortByCount ステージを使用するためのコード例を表示するには、使用する言語のタブを選択します。

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 の$split集計演算子は、指定された区切り文字に基づいて文字列を部分文字列の配列に分割するために使用されます。これは、複雑な文字列フィールドを解析し、個々のコンポーネントを抽出してさらに処理する場合に便利です。

パラメータ

- string: 分割する文字列。
- delimiter: 入力文字列を分割するために使用される文字または文字列。

例 (MongoDB シェル)

この例では、 $splitを使用して「Desk」フィールドのコンポーネントを配列に分割し、データの処理を容易にします。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.people.aggregate([
  { $project: { parts: { $split: ["$Desk", "-"] } } }
]);
```

出力

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

の出力は、従業員の情報を表示するためにアプリケーションで使用できる配列$splitを作成します。

コードの例

$split コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能。

Amazon DocumentDB の `$sqrt`演算子は、数値の平方根を計算するために使用されます。

パラメータ

- `expression`: 引数は、負以外の数値に解決される限り、任意の有効な式にすることができます。

例 (MongoDB シェル)

次の例は、`$sqrt`演算子を使用して数値の平方根を計算する方法を示しています。

サンプルドキュメントを作成する

```
db.numbers.insertMany([
  { "_id": 1, "number": 16 },
  { "_id": 2, "number": 36 },
  { "_id": 3, "number": 64 }
]);
```

クエリの例

```
db.numbers.aggregate([
  { $project: {
    "_id": 1,
    "square_root": { $sqrt: "$number" }
  }}
]);
```

出力

```
[
  { _id: 1, square_root: 4 },
  { _id: 2, square_root: 6 },
  { _id: 3, square_root: 8 }
]
```

コードの例

$sqrt コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の $stdDevPop演算子は、数値の母集団標準偏差を計算します。アキュムレータは、集約パイプラインの$groupステージのグループ内のドキュメント全体の母集団標準偏差を計算します。式として、数値の配列の母集団標準偏差を計算します。母集団標準偏差は、(N-1 ではなく) N を除数として使用します。数値以外の値は無視されます。数値がない場合は、 を返しますnull。数値が 1 つしかない場合、 を返します0。

パラメータ

- expression: 数値または数値の配列に解決される式。

例 (MongoDB シェル)

次の例は、 $stdDevPop演算子を使用して、サブジェクトあたりのスコアの母集団標準偏差を計算する方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.scores.aggregate([
  { $group: {
      _id: "$subject",
      stdDev: { $stdDevPop: "$score" }
    }}
]);
```

出力

```
[
  { "_id": "math", "stdDev": 10.75174404457249 },
  { "_id": "science", "stdDev": 12.441864811996632 }
]
```

式の使用例 (MongoDB シェル)

$stdDevPop 演算子を$projectステージ内の式として使用して、配列フィールドの母集団標準偏差を計算することもできます。

サンプルドキュメントを作成する

```
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] }
]);
```

クエリの例

```
db.experiments.aggregate([
  { $project: {
      stdDev: { $stdDevPop: "$measurements" }
```

```
    }}
]);
```

出力

```
[
  { "_id": 1, "stdDev": 2.8284271247461903 },
  { "_id": 2, "stdDev": 0 },
  { "_id": 3, "stdDev": 2.8284271247461903 }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$stdDevPop 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメントに対して指定された式の母集団標準偏差を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

サンプルドキュメントを作成する

```
db.sensorData.insertMany([
  { _id: 1, sensor: "A", time: 1, reading: 4 },
  { _id: 2, sensor: "A", time: 2, reading: 10 },
  { _id: 3, sensor: "A", time: 3, reading: 10 },
  { _id: 4, sensor: "B", time: 1, reading: 10 },
  { _id: 5, sensor: "B", time: 2, reading: 20 }
]);
```

クエリの例

次の例では、ドキュメントを でパーティション化しsensor、各パーティションを でソートしてtime、パーティションの開始readingから現在のドキュメントまでの実行中の母集団標準偏差を返します。

```
db.sensorData.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$sensor",
      sortBy: { time: 1 },
      output: {
```

```
        runningStdDev: {
          $stdDevPop: "$reading",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "sensor": "A", "time": 1, "reading": 4, "runningStdDev": 0 },
  { "_id": 2, "sensor": "A", "time": 2, "reading": 10, "runningStdDev": 3 },
  { "_id": 3, "sensor": "A", "time": 3, "reading": 10, "runningStdDev":
 2.8284271247461903 },
  { "_id": 4, "sensor": "B", "time": 1, "reading": 10, "runningStdDev": 0 },
  { "_id": 5, "sensor": "B", "time": 2, "reading": 20, "runningStdDev": 5 }
]
```

各ドキュメントはrunningStdDev、現在のドキュメントまでのパーティションreading内の の母集団標準偏差である で拡張されます。ウィンドウに 1 つの値が含まれている場合、 は$stdDevPopを返します0。

コードの例

$stdDevPop 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group)、式の使用 (の場合$project)、ウィンドウ演算子の使用 (の場合) を示しています$setWindowFields。

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');
await 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 }
]);
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');
await 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] }
]);
const expressionResult = await experiments.aggregate([
  { $project: {
      stdDev: { $stdDevPop: "$measurements" }
    }}
]).toArray();
console.log('Expression result:', expressionResult);

// Window operator usage: running stdDevPop within each partition
const sensorData = db.collection('sensorData');
await sensorData.insertMany([
  { _id: 1, sensor: "A", time: 1, reading: 4 },
  { _id: 2, sensor: "A", time: 2, reading: 10 },
  { _id: 3, sensor: "A", time: 3, reading: 10 },
```

```
      { _id: 4, sensor: "B", time: 1, reading: 10 },
      { _id: 5, sensor: "B", time: 2, reading: 20 }
    ]);
    const windowResult = await sensorData.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$sensor",
          sortBy: { time: 1 },
          output: {
            runningStdDev: {
              $stdDevPop: "$reading",
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } 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']
        scores.insert_many([
            { '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 }
    ])
    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']
    experiments.insert_many([
        { '_id': 1, 'measurements': [10, 12, 14, 16, 18] },
        { '_id': 2, 'measurements': [5, 5, 5, 5, 5] },
        { '_id': 3, 'measurements': [2, 4, 6, 8, 10] }
    ])
    expression_result = list(experiments.aggregate([
        { '$project': {
            'stdDev': { '$stdDevPop': '$measurements' }
        }}
    ]))
    print('Expression result:', expression_result)

    # Window operator usage: running stdDevPop within each partition
    sensor_data = db['sensorData']
    sensor_data.insert_many([
        { '_id': 1, 'sensor': 'A', 'time': 1, 'reading': 4 },
        { '_id': 2, 'sensor': 'A', 'time': 2, 'reading': 10 },
        { '_id': 3, 'sensor': 'A', 'time': 3, 'reading': 10 },
        { '_id': 4, 'sensor': 'B', 'time': 1, 'reading': 10 },
        { '_id': 5, 'sensor': 'B', 'time': 2, 'reading': 20 }
    ])
    window_result = list(sensor_data.aggregate([
        {
            '$setWindowFields': {
                'partitionBy': '$sensor',
```

```
                    'sortBy': { 'time': 1 },
                    'output': {
                        'runningStdDev': {
                            '$stdDevPop': '$reading',
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()

example()
```

$stdDevSamp

バージョン 8.0.1 の新機能。

Amazon DocumentDB の $stdDevSamp演算子は、数値のサンプル標準偏差を計算します。アキュムレータは、集約パイプラインの$groupステージのグループ内のドキュメント全体のサンプル標準偏差を計算します。式として、数値の配列のサンプル標準偏差を計算します。サンプル標準偏差は、N-1 を除数として使用します (Bessel の補正) 。数値以外の値は無視されます。数値が 2 つ未満の場合は、 を返しますnull。

パラメータ

- expression: 数値または数値の配列に解決される式。

例 (MongoDB シェル)

次の例は、 $stdDevSamp演算子を使用して、サブジェクトあたりのスコアのサンプル標準偏差を計算する方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.scores.aggregate([
  { $group: {
      _id: "$subject",
      stdDev: { $stdDevSamp: "$score" }
    }}
]);
```

出力

```
[
  { "_id": "math", "stdDev": 6.454972243679028 },
  { "_id": "science", "stdDev": 6.454972243679028 }
]
```

式の使用例 (MongoDB シェル)

$stdDevSamp 演算子を$projectステージ内の式として使用して、配列フィールドのサンプル標準偏差を計算することもできます。

サンプルドキュメントを作成する

```
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] }
]);
```

クエリの例

```
db.experiments.aggregate([
  { $project: {
      stdDev: { $stdDevSamp: "$measurements" }
```

```
    }}
]);
```

出力

```
[
  { "_id": 1, "stdDev": 3.1622776601683795 },
  { "_id": 2, "stdDev": 0 },
  { "_id": 3, "stdDev": 3.1622776601683795 }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$stdDevSamp 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメントに対して指定された式のサンプル標準偏差を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

サンプルドキュメントを作成する

```
db.sensorReadings.insertMany([
  { _id: 1, sensor: "A", time: 1, value: 2 },
  { _id: 2, sensor: "A", time: 2, value: 4 },
  { _id: 3, sensor: "A", time: 3, value: 6 },
  { _id: 4, sensor: "B", time: 1, value: 10 },
  { _id: 5, sensor: "B", time: 2, value: 12 }
]);
```

クエリの例

次の例ではsensor、ドキュメントを でパーティション化し、各パーティションを でソートしてtime、パーティションの開始valueから現在のドキュメントまでの実行中のサンプル標準偏差を返します。

```
db.sensorReadings.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$sensor",
      sortBy: { time: 1 },
      output: {
```

```
        runningStdDev: {
          $stdDevSamp: "$value",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "sensor": "A", "time": 1, "value": 2, "runningStdDev": null },
  { "_id": 2, "sensor": "A", "time": 2, "value": 4, "runningStdDev":
 1.4142135623730951 },
  { "_id": 3, "sensor": "A", "time": 3, "value": 6, "runningStdDev": 2 },
  { "_id": 4, "sensor": "B", "time": 1, "value": 10, "runningStdDev": null },
  { "_id": 5, "sensor": "B", "time": 2, "value": 12, "runningStdDev":
 1.4142135623730951 }
]
```

各ドキュメントはrunningStdDev、現在のドキュメントまでのパーティションvalue内の のサンプル標準偏差である で拡張されます。ウィンドウに含まれる値が 2 つ未満の場合 (各パーティションの最初のドキュメント)、 は $stdDevSampを返しますnull。

コードの例

$stdDevSamp 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group)、式の使用 (の場合$project)、ウィンドウ演算子の使用 (の場合) を示しています$setWindowFields。

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');
    await 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 }
    ]);
    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');
    await 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] }
    ]);
    const expressionResult = await experiments.aggregate([
      { $project: {
          stdDev: { $stdDevSamp: "$measurements" }
        }}
    ]).toArray();
    console.log('Expression result:', expressionResult);

    // Window operator usage: running stdDevSamp within each partition
    const sensorReadings = db.collection('sensorReadings');
    await sensorReadings.insertMany([
      { _id: 1, sensor: "A", time: 1, value: 2 },
      { _id: 2, sensor: "A", time: 2, value: 4 },
      { _id: 3, sensor: "A", time: 3, value: 6 },
      { _id: 4, sensor: "B", time: 1, value: 10 },
```

```
      { _id: 5, sensor: "B", time: 2, value: 12 }
    ]);
    const windowResult = await sensorReadings.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$sensor",
          sortBy: { time: 1 },
          output: {
            runningStdDev: {
              $stdDevSamp: "$value",
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } 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']
        scores.insert_many([
            { '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 }
        ])
        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']
        experiments.insert_many([
            { '_id': 1, 'measurements': [10, 12, 14, 16, 18] },
            { '_id': 2, 'measurements': [5, 5, 5, 5, 5] },
            { '_id': 3, 'measurements': [2, 4, 6, 8, 10] }
        ])
        expression_result = list(experiments.aggregate([
            { '$project': {
                'stdDev': { '$stdDevSamp': '$measurements' }
            }}
        ]))
        print('Expression result:', expression_result)

        # Window operator usage: running stdDevSamp within each partition
        sensor_readings = db['sensorReadings']
        sensor_readings.insert_many([
            { '_id': 1, 'sensor': 'A', 'time': 1, 'value': 2 },
            { '_id': 2, 'sensor': 'A', 'time': 2, 'value': 4 },
            { '_id': 3, 'sensor': 'A', 'time': 3, 'value': 6 },
            { '_id': 4, 'sensor': 'B', 'time': 1, 'value': 10 },
            { '_id': 5, 'sensor': 'B', 'time': 2, 'value': 12 }
        ])
        window_result = list(sensor_readings.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$sensor',
                    'sortBy': { 'time': 1 },
                    'output': {
                        'runningStdDev': {
```

```
                            '$stdDevSamp': '$value',
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()

example()
```

$strLenBytes

Amazon DocumentDB の $strLenBytes演算子は、文字列の長さをバイト単位で決定するために使用されます。これは、文字列フィールドのストレージサイズを理解する必要がある場合、特に 1 文字あたり 1 バイト以上を使用する Unicode 文字を処理する場合に便利です。

パラメータ

- expression: 長さを計算する文字列式。

例 (MongoDB シェル)

この例では、 $strLenBytes演算子を使用して文字列フィールドの長さをバイト単位で計算する方法を示します。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.people.aggregate([
```

```
  {
    $project: {
      "Desk": 1,
      "length": { $strLenBytes: "$Desk" }
    }
  }
])
```

出力

```
{ "_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 }
```

「Düsseldorf-BVV-021」文字列の長さは 19 バイトです。Unicode 文字「Ü」が 2 バイトを占めるため、コードポイント (18) の数とは異なります。

コードの例

$strLenBytes コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の `$strLenCP`演算子は、コードポイントの文字列式の長さ (Unicode 文字) を決定するために使用されます。これは、バイト数ではなく文字列内の文字数を知る必要がある場合に便利です。

パラメータ

- `expression`: コードポイントで長さを返す文字列式。

例 (MongoDB シェル)

次の例は、 $strLenCP演算子を使用して Unicode 文字を含む文字列の長さを決定する方法を示しています。

サンプルドキュメントを作成する

```
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" }
])
```

クエリの例

```
db.people.aggregate([
  {
    $project: {
      "Desk": 1,
      "length": { $strLenCP: "$Desk" }
    }
  }
])
```

出力

```
{ "_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 }
```

Unicode 文字 (Ü) を含む「Düsseldorf-BVV-021」文字列の長さ測定の違いに注目してください。$strLenCP 演算子は Unicode 文字数を正しくカウントし、$strLenBytes演算子はバイト数をカウントします。

コードの例

$strLenCP コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

Node.js

以下は、MongoDB ドライバーで Node.js アプリケーションで $strLenCP演算子を使用する例です。

```
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

PyMongo ドライバーで Python アプリケーションで $strLenCP演算子を使用する例を次に示します。

```
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 の $strcasecmp演算子は、2 つの文字列間で大文字と小文字を区別しない比較を実行します。2 つの入力文字列の辞書比較を示す整数値が返され、大文字と小文字の違いは無視されます。

パラメータ

- string1: 比較する最初の文字列。
- string2: 比較する 2 番目の文字列。

例 (MongoDB シェル)

この例では、 $strcasecmp演算子を使用してpeopleコレクション内のデスクの場所文字列を比較し、大文字と小文字の違いを無視する方法を示します。

サンプルドキュメントを作成する

```
db.people.insertMany([
  { "_id": 1, "Desk": "mke233-wi" },
  { "_id": 2, "Desk": "MKE233-WI" },
  { "_id": 3, "Desk": "mke233-wi" }
]);
```

クエリの例

```
db.people.aggregate([
```

```
  {
    $project: {
      item: 1,
      compare: { $strcasecmp: ["$Desk", "mke233-wi"] }
    }
  }
]);
```

出力

```
{ "_id" : 1, "compare" : 0 }
{ "_id" : 2, "compare" : 0 }
{ "_id" : 3, "compare" : 0 }
```

出力は、 "Desk"フィールドと文字列の比較が 3 つのドキュメントすべて0に対してを"mke233-wi"返し、大文字と小文字が無視された場合に文字列が等しいことを示します。

コードの例

$strcasecmp コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $substr演算子は、特定の文字列から部分文字列を抽出するために使用されます。これは、バイトの範囲ではなく文字の範囲に基づいて部分文字列を定義する必要がある場合に特に便利です。これは、文字を表すために使用されるバイト数が異なる可能性がある Unicode 文字列を処理する場合に特に重要です。

パラメータ

- `string`: 部分文字列を抽出する入力文字列。

- `start`: 抽出する部分文字列の開始位置 (ゼロベース) 。負以外の整数式にすることができます。
- `length`: 抽出された部分文字列の文字数。負以外の整数式にすることができます。

例 (MongoDB シェル)

この例では、 を使用して従業員のデスクの場所から状態の略語を$substr抽出する方法を示します。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.people.aggregate([
  {
    $project: {
      "state": { $substr: ["$Desk", 12, 3] }
    }
  }
])
```

出力

```
{ "_id": 1, "state": "NRW" },
{ "_id": 2, "state": "HBB" },
{ "_id": 3, "state": "SHH" },
{ "_id": 4, "state": "BBB" }
```

コードの例

$substr コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $substrBytes演算子は、指定されたバイト範囲に基づいて文字列から部分文字列を抽出するために使用されます。この演算子は、文字列から部分文字列を抽出する必要があり、文字列内の各文字を表すために必要なバイト数が重要である場合に便利です。

Unicode コードポイントの数で動作する とは異なり$substrCP、 は文字列内の文字を表すために必要なバイト数で$substrBytes動作します。これは、ASCII 以外の文字を含む文字列を使用する場合に特に便利です。これらの文字では、表すために複数のバイトが必要になる可能性があるためです。

注: $substrはバージョン 3.4 以降廃止されました。 $substrは のエイリアスになりました$substrBytes。

パラメータ

- string: 部分文字列が抽出される入力文字列。
- startByte: 抽出する部分文字列のゼロベースの開始バイト位置。負の値を使用して、文字列の末尾から位置を指定できます。
- length: 抽出する部分文字列のバイト数。

例 (MongoDB シェル)

この例では、 $substrBytes を使用して、ASCII 以外の文字を含む文字列から部分文字列を抽出します。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.people.aggregate([
  {
    $project: {
      "state": { $substrBytes: [ "$Desk", 12, 3] }
    }
  }
])
```

出力

```
{ "_id": 1, "state": "NRW" },
{ "_id": 2, "state": "HBB" },
{ "_id": 3, "state": "SHH" },
{ "_id": 4, "state": "BBB" }
```

この例では、 $substrBytesを使用して、 Deskフィールドの 12 バイト目から 3 バイトの部分文字列を抽出します。これにより、文字列に ASCII 以外の文字が含まれている場合でも、2 文字の状態の略語を抽出できます。

コードの例

$substrBytes コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $substrCP演算子は、文字列から部分文字列を抽出するために使用されます。部分文字列は UTF-8 コードポイント (CP) の範囲として指定されます。この演算子は、文字の基になるバイト表現を心配することなく部分文字列を抽出できるため、Unicode 文字列を使用する場合に特に便利です。

バイト位置で動作する $substrBytes 演算子とは異なり、 $substrCP演算子はコードポイント位置を操作します。これにより、コードポイントの数がバイト数や文字数と一致しない可能性があるため、ASCII 以外の文字を含む文字列の操作が容易になります。

パラメータ

- string: 部分文字列を抽出する入力文字列。
- start: 部分文字列を抽出する開始コードポイントの位置 (ゼロベース) 。
- length: 抽出するコードポイントの数。

例 (MongoDB シェル)

この例では、 $substrCP演算子を使用して、従業員のデスクの場所を含む文字列から状態の略語を抽出します。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.people.aggregate([
  {
    $project: {
      "state": { $substrCP: ["$Desk", 25, 2] }
    }
  }
]);
```

出力

```
{ "_id" : 1, "state" : "MN" }
{ "_id" : 2, "state" : "LA" }
{ "_id" : 3, "state" : "CA" }
```

この例では、状態の略語は Deskフィールドの 25 番目のコードポイントで始まり、2 つのコードポイントの長さであることがわかっています。$substrCP 演算子を使用すると、文字列の基になるバイト表現について心配することなく、状態の略語を抽出できます。

コードの例

$substrCP コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

Node.js と Python の両方の例では、MongoDB シェルの例と同様に、 $substrCP演算子を使用して Deskフィールドから状態の略語を抽出します。

$subtract

Amazon DocumentDB の $subtract演算子は、値を減算するために使用されます。日付、数値、またはその両方の組み合わせを減算するために使用できます。この演算子は、2 つの日付の差を計算する場合や、数値から値を引く場合に便利です。

パラメータ

- expression1: 減算される最初の値。
- expression2: から減算される 2 番目の値<expression1>。

例 (MongoDB シェル)

次の例は、 $subtract演算子を使用して 2 つの日付の差を計算する方法を示しています。

サンプルドキュメントを作成する

```
db.dates.insert([
  {
  "_id": 1,
  "startDate": ISODate("2023-01-01T00:00:00Z"),
  "endDate": ISODate("2023-01-05T12:00:00Z")
  }
]);
```

クエリの例

```
db.dates.aggregate([
  {
```

```
    $project: {
      _id: 1,
      durationDays: {
        $divide: [
          { $subtract: ["$endDate", "$startDate"] },
          1000 * 60 * 60 * 24  // milliseconds in a day
        ]
      }
    }
  }
]);
```

出力

```
[ { _id: 1, durationDays: 4.5 } ]
```

この例では、 $subtract演算子を使用して $endDateと の差を日$startDate単位で計算します。

コードの例

$subtract コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $sum演算子は、グループ内のドキュメントごとに指定された式の合計を返します。これは、集計パイプラインの $group ステージで集計計算を実行するために通常使用されるグループアキュムレータ演算子です。

パラメータ

- `expression`: 合計する数値式。これは、フィールドパス、式、または定数です。

例 (MongoDB シェル)

次の例は、 $sum演算子を使用して各製品の総売上を計算する方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.sales.aggregate([
  { $group: {
      _id: "$product",
      totalSales: { $sum: { $multiply: [ "$price", "$quantity" ] } }
    }}
]);
```

出力

```
[
  { "_id": "abc", "totalSales": 50 },
  { "_id": "xyz", "totalSales": 120 }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$sum 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメントに指定された式の合計を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

サンプルドキュメントを作成する

```
db.dailySales.insertMany([
  { _id: 1, product: "abc", day: 1, amount: 10 },
  { _id: 2, product: "abc", day: 2, amount: 20 },
  { _id: 3, product: "abc", day: 3, amount: 15 },
  { _id: 4, product: "xyz", day: 1, amount: 30 },
  { _id: 5, product: "xyz", day: 2, amount: 25 }
]);
```

クエリの例

次の例では、ドキュメントを でパーティション化しproduct、各パーティションを でソートしてday、パーティションの先頭amountから現在のドキュメントまでの合計 を返します。

```
db.dailySales.aggregate([
  {
```

```
    $setWindowFields: {
      partitionBy: "$product",
      sortBy: { day: 1 },
      output: {
        runningTotal: {
          $sum: "$amount",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "product": "abc", "day": 1, "amount": 10, "runningTotal": 10 },
  { "_id": 2, "product": "abc", "day": 2, "amount": 20, "runningTotal": 30 },
  { "_id": 3, "product": "abc", "day": 3, "amount": 15, "runningTotal": 45 },
  { "_id": 4, "product": "xyz", "day": 1, "amount": 30, "runningTotal": 30 },
  { "_id": 5, "product": "xyz", "day": 2, "amount": 25, "runningTotal": 55 }
]
```

各ドキュメントはrunningTotal、現在のドキュメントまでのパーティションamount内の の累積合計である で拡張されます。

コードの例

$sum 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group) とウィンドウ演算子の使用 (の場合) の両方を示しています$setWindowFields。

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: sum per group
    const sales = db.collection('sales');
    await 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 }
    ]);
    const accumulatorResult = await sales.aggregate([
      { $group: {
          _id: "$product",
          totalSales: { $sum: { $multiply: [ "$price", "$quantity" ] } }
        }}
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Window operator usage: running total within each partition
    const dailySales = db.collection('dailySales');
    await dailySales.insertMany([
      { _id: 1, product: "abc", day: 1, amount: 10 },
      { _id: 2, product: "abc", day: 2, amount: 20 },
      { _id: 3, product: "abc", day: 3, amount: 15 },
      { _id: 4, product: "xyz", day: 1, amount: 30 },
      { _id: 5, product: "xyz", day: 2, amount: 25 }
    ]);
    const windowResult = await dailySales.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$product",
          sortBy: { day: 1 },
          output: {
            runningTotal: {
              $sum: "$amount",
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);
```

```
  } 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

        # Accumulator usage: sum per group
        sales = db.sales
        sales.insert_many([
            { 'product': 'abc', 'price': 10, 'quantity': 2 },
            { 'product': 'abc', 'price': 10, 'quantity': 3 },
            { 'product': 'xyz', 'price': 20, 'quantity': 1 },
            { 'product': 'xyz', 'price': 20, 'quantity': 5 }
        ])
        accumulator_result = list(sales.aggregate([
            { '$group': {
                '_id': '$product',
                'totalSales': { '$sum': { '$multiply': [ '$price', '$quantity' ] } }
            }}
        ]))
        pprint(accumulator_result)

        # Window operator usage: running total within each partition
        daily_sales = db.dailySales
        daily_sales.insert_many([
```

```
            { '_id': 1, 'product': 'abc', 'day': 1, 'amount': 10 },
            { '_id': 2, 'product': 'abc', 'day': 2, 'amount': 20 },
            { '_id': 3, 'product': 'abc', 'day': 3, 'amount': 15 },
            { '_id': 4, 'product': 'xyz', 'day': 1, 'amount': 30 },
            { '_id': 5, 'product': 'xyz', 'day': 2, 'amount': 25 }
        ])
        window_result = list(daily_sales.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$product',
                    'sortBy': { 'day': 1 },
                    'output': {
                        'runningTotal': {
                            '$sum': '$amount',
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        pprint(window_result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$switch

バージョン 4.0 の新機能。

Elastic クラスターではサポートされていません。

$switch 演算子は Amazon DocumentDB の条件式演算子で、ケース式のリストを評価し、 が true に評価される最初のケースの値、またはケース式が true でない場合はデフォルト値を返すことができます。

パラメータ

- branches: ドキュメントの配列。各ドキュメントには、評価するブール式を含む大文字と小文字のフィールドと、大文字と小文字の式が true の場合に返す値を含むフィールドがあります。
- default: (オプション) いずれの大文字と小文字の式も true でない場合に返す値。

例 (MongoDB シェル)

次の例は、 $switch演算子を使用して、注文の合計に基づいて注文の配送コストを決定する方法を示しています。

サンプルドキュメントを作成する

```
db.orders.insertMany([
  { _id: 1, total: 50 },
  { _id: 2, total: 150 },
  { _id: 3, total: 250 }
]);
```

クエリの例

```
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
        }
      }
    }
  }
])
```

出力

```
[
```

```
  {
    "_id": 1,
    "total": 50,
    "shippingCost": 5
  },
  {
    "_id": 2,
    "total": 150,
    "shippingCost": 10
  },
  {
    "_id": 3,
    "total": 250,
    "shippingCost": 15
  }
]
```

コードの例

$switch コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 の新機能。

Amazon DocumentDB の$tan演算子は、ラジアンで測定される値の接線を返します。

入力式は数値に解決する必要があります。値が度数の場合は、 に渡す$degreesToRadians前に を適用します$tan。

戻り値の型はdoubleデフォルトで です。入力が 128 ビットの 10 進数の場合、出力も 128 ビットの 10 進数です。

パラメータ

- expression: ラジアン単位で数値に解決される式。

動作

null、NaN、および +/- Infinity

例	結果
{ $tan: NaN }	NaN
{ $tan: null }	null
{ $tan: Infinity }、または { $tan: -Infinity }	エラーをスローします。

入力が nullであるか、参照されるフィールドが欠落している場合、 は $tanを返しますnull。の入力により、 NaNが生成されますNaN。正または負の無限大は、接線がそれらの値で未定義であるため、エラーを引き起こします。

例 (MongoDB シェル)

次の例は、 $tan演算子を使用して角度の接線をラジアン単位で計算する方法を示しています。

サンプルドキュメントを作成する

```
db.angles.insertMany([
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 0.7853981633974483 },
  { "_id": 3, "angle": -0.7853981633974483 }
]);
```

クエリの例

```
db.angles.aggregate([
  { $project: {
    "tangent": { $tan: "$angle" }
  }}
]);
```

出力

```
[
  { "_id": 1, "tangent": 0 },
  { "_id": 2, "tangent": 0.9999999999999999 },
  { "_id": 3, "tangent": -0.9999999999999999 }
]
```

コードの例

$tan 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 の新機能。

Amazon DocumentDB の $tanh演算子は、値の双曲線タンジェントを返します。

入力式は数値に解決する必要があります。

戻り値の型はdoubleデフォルトで です。入力が 128 ビットの 10 進数の場合、出力も 128 ビットの 10 進数です。

パラメータ

- `expression`: 数値に解決される式。

動作

null、NaN、および +/- Infinity

例	結果
`{ $tanh: NaN }`	`NaN`
`{ $tanh: null }`	`null`
`{ $tanh: -Infinity }`	`-1`
`{ $tanh: Infinity }`	`1`

入力が `null`であるか、参照されるフィールドが欠落している場合、 は `$tanh`を返します`null`。の入力により、 `NaN`が生成されます`NaN`。正の無限大の場合、結果は です`1`。負の無限大の場合、結果は です`-1`。

例 (MongoDB シェル)

次の例は、 `$tanh`演算子を使用して値の双曲線タンジェントを計算する方法を示しています。

サンプルドキュメントを作成する

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 1 },
  { "_id": 3, "value": -1 }
]);
```

クエリの例

```
db.values.aggregate([
```

```
  { $project: {
    "result": { $tanh: "$value" }
  }}
]);
```

出力

```
[
  { "_id": 1, "result": 0 },
  { "_id": 2, "result": 0.7615941559557649 },
  { "_id": 3, "result": -0.7615941559557649 }
]
```

コードの例

$tanh 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

$group ステージで $top アキュムレータを使用して、指定されたソート順序に従ってグループごとに最高ランクのドキュメントを返します。

パラメータ

- sortBy: ソート順序を指定するドキュメント。昇順-1または降順1に使用します。
- output: 上部のドキュメントから返すフィールドを指定する式。

例 (MongoDB シェル)

次の例は、 $top アキュムレータを使用して、売上コレクション内のアイテムあたりの上位売上 (最大数量) を検索する方法を示しています。

サンプルドキュメントを作成する

```
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 }
])
```

クエリの例

```
db.sales.aggregate([
  { $group: { _id: "$item", topSale: { $top: { sortBy: { quantity: -1 }, output:
 { quantity: "$quantity", price: "$price" } } } } }
])
```

出力

```
[
  { "_id": "xyz", "topSale": { "quantity": 15, "price": 3 } },
  { "_id": "abc", "topSale": { "quantity": 10, "price": 5 } }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$top 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメント間で、最上位のドキュメントoutputから (オペレータ自身の に従ってsortBy) を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

> **Note**
>
> でウィンドウ演算子として使用する場合$setWindowFields、 $topは 100 MB の中間データに制限されます。この制限を超えるオペレーションはエラーを返します。

サンプルドキュメントを作成する

```
db.matchScores.insertMany([
  { _id: 1, player: "Alice", round: 1, score: 20 },
  { _id: 2, player: "Alice", round: 2, score: 35 },
  { _id: 3, player: "Alice", round: 3, score: 28 },
```

```
  { _id: 4, player: "Bob", round: 1, score: 15 },
  { _id: 5, player: "Bob", round: 2, score: 40 }
]);
```

クエリの例

次の例ではplayer、ドキュメントを でパーティション化し、各パーティションを でソートしてround、パーティションの先頭から現在のドキュメントまでの最高スコアのラウンドを返します。

```
db.matchScores.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$player",
      sortBy: { round: 1 },
      output: {
        bestRoundSoFar: {
          $top: { sortBy: { score: -1 }, output: { round: "$round", score:
 "$score" } },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "player": "Alice", "round": 1, "score": 20, "bestRoundSoFar": { "round":
 1, "score": 20 } },
  { "_id": 2, "player": "Alice", "round": 2, "score": 35, "bestRoundSoFar": { "round":
 2, "score": 35 } },
  { "_id": 3, "player": "Alice", "round": 3, "score": 28, "bestRoundSoFar": { "round":
 2, "score": 35 } },
  { "_id": 4, "player": "Bob", "round": 1, "score": 15, "bestRoundSoFar": { "round": 1,
 "score": 15 } },
  { "_id": 5, "player": "Bob", "round": 2, "score": 40, "bestRoundSoFar": { "round": 2,
 "score": 40 } }
]
```

各ドキュメントはbestRoundSoFar、現在のドキュメントまでパーティション内のトップスコアラウンドである で拡張されます。

コードの例

$top 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group) とウィンドウ演算子の使用 (の場合) の両方を示しています$setWindowFields。

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: top document per group
    const sales = db.collection('sales');
    await 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 }
    ]);
    const accumulatorResult = await sales.aggregate([
      { $group: { _id: "$item", topSale: { $top: { sortBy: { quantity: -1 }, output:
 { quantity: "$quantity", price: "$price" } } } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Window operator usage: best round so far within each partition
    const matchScores = db.collection('matchScores');
    await matchScores.insertMany([
      { _id: 1, player: "Alice", round: 1, score: 20 },
      { _id: 2, player: "Alice", round: 2, score: 35 },
      { _id: 3, player: "Alice", round: 3, score: 28 },
      { _id: 4, player: "Bob", round: 1, score: 15 },
      { _id: 5, player: "Bob", round: 2, score: 40 }
```

```
    ]);
    const windowResult = await matchScores.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$player",
          sortBy: { round: 1 },
          output: {
            bestRoundSoFar: {
              $top: { sortBy: { score: -1 }, output: { round: "$round", score:
 "$score" } },
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } 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']

        # Accumulator usage: top document per group
        sales = db['sales']
```

```
        sales.insert_many([
            { 'item': 'abc', 'quantity': 10, 'price': 5 },
            { 'item': 'abc', 'quantity': 5, 'price': 8 },
            { 'item': 'xyz', 'quantity': 15, 'price': 3 },
            { 'item': 'xyz', 'quantity': 7, 'price': 6 }
        ])
        accumulator_result = list(sales.aggregate([
            { '$group': { '_id': '$item', 'topSale': { '$top': { 'sortBy':
{ 'quantity': -1 }, 'output': { 'quantity': '$quantity', 'price':
'$price' } } } } }
        ]))
        pprint(accumulator_result)

        # Window operator usage: best round so far within each partition
        match_scores = db['matchScores']
        match_scores.insert_many([
            { '_id': 1, 'player': 'Alice', 'round': 1, 'score': 20 },
            { '_id': 2, 'player': 'Alice', 'round': 2, 'score': 35 },
            { '_id': 3, 'player': 'Alice', 'round': 3, 'score': 28 },
            { '_id': 4, 'player': 'Bob', 'round': 1, 'score': 15 },
            { '_id': 5, 'player': 'Bob', 'round': 2, 'score': 40 }
        ])
        window_result = list(match_scores.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$player',
                    'sortBy': { 'round': 1 },
                    'output': {
                        'bestRoundSoFar': {
                            '$top': { 'sortBy': { 'score': -1 }, 'output':
{ 'round': '$round', 'score': '$score' } },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        pprint(window_result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
```

```
            client.close()

example()
```

$topN

バージョン 8.0.1 から新規。

$group ステージの$topNアキュムレータを使用して、指定されたソート順序に従ってグループ内の上位 N 個の要素を返します。グループに含まれる要素が N 未満の場合、 はグループ内のすべての要素$topNを返します。

パラメータ

- n: 正の整数、または 1 に解決される式。グループごとに返される上位の結果の数を指定します。
- sortBy: ソート順序を指定するドキュメント。昇順1の場合は 、降順-1の場合は を使用します。
- output: 上位 N 個の各ドキュメントから返すフィールドを指定する式。

例 (MongoDB シェル)

次の例は、 $topN アキュムレータを使用して、売上コレクション内の項目あたりの売上 (最大数量) の上位 2 つを見つける方法を示しています。

サンプルドキュメントを作成する

```
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 }
])
```

クエリの例

```
db.sales.aggregate([
  { $group: { _id: "$item", topTwoSales: { $topN: { n: 2, sortBy: { quantity: -1 },
 output: { quantity: "$quantity", price: "$price" } } } } }
```

```
])
```

出力

```
[
  { "_id": "xyz", "topTwoSales": [{ "quantity": 15, "price": 3 }, { "quantity": 9,
 "price": 6 }] },
  { "_id": "abc", "topTwoSales": [{ "quantity": 10, "price": 5 }, { "quantity": 7,
 "price": 8 }] }
]
```

ウィンドウ演算子の使用例 (MongoDB シェル)

バージョン 8.0.2 の新機能。

$topN 演算子は、$setWindowFieldsステージのウィンドウ演算子としても使用できます。このコンテキストでは、各ウィンドウのドキュメント間で上位 N のドキュメント (オペレータ自身の によるsortBy) outputから の配列を返します。output フィールドで 演算子を指定し、オプションでwindowドキュメントを使用してウィンドウの境界を定義します。

> **Note**
>
> でウィンドウ演算子として使用する場合$setWindowFields、 $topNは 100 MB の中間データに制限されます。この制限を超えるオペレーションはエラーを返します。

サンプルドキュメントを作成する

```
db.matchScores.insertMany([
  { _id: 1, player: "Alice", round: 1, score: 20 },
  { _id: 2, player: "Alice", round: 2, score: 35 },
  { _id: 3, player: "Alice", round: 3, score: 28 },
  { _id: 4, player: "Bob", round: 1, score: 15 },
  { _id: 5, player: "Bob", round: 2, score: 40 }
]);
```

クエリの例

次の例ではplayer、ドキュメントを でパーティション化し、各パーティションを でソートしround、パーティションの先頭から現在のドキュメントまでの 2 つの最高スコアのラウンドを返します。

```
db.matchScores.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$player",
      sortBy: { round: 1 },
      output: {
        topTwoRoundsSoFar: {
          $topN: { n: 2, sortBy: { score: -1 }, output: { round: "$round", score:
 "$score" } },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

出力

```
[
  { "_id": 1, "player": "Alice", "round": 1, "score": 20, "topTwoRoundsSoFar":
 [{ "round": 1, "score": 20 }] },
  { "_id": 2, "player": "Alice", "round": 2, "score": 35, "topTwoRoundsSoFar":
 [{ "round": 2, "score": 35 }, { "round": 1, "score": 20 }] },
  { "_id": 3, "player": "Alice", "round": 3, "score": 28, "topTwoRoundsSoFar":
 [{ "round": 2, "score": 35 }, { "round": 3, "score": 28 }] },
  { "_id": 4, "player": "Bob", "round": 1, "score": 15, "topTwoRoundsSoFar":
 [{ "round": 1, "score": 15 }] },
  { "_id": 5, "player": "Bob", "round": 2, "score": 40, "topTwoRoundsSoFar":
 [{ "round": 2, "score": 40 }, { "round": 1, "score": 15 }] }
]
```

各ドキュメントはtopTwoRoundsSoFar、現在のドキュメントまでパーティション内で 2 つの最高スコアのラウンドである で拡張されます。ウィンドウに含まれるドキュメントが 2 つ未満の場合、それらはすべて返されます。

コードの例

$topN 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。次の例は、アキュムレータの使用 (の場合$group) とウィンドウ演算子の使用 (の場合) の両方を示しています$setWindowFields。

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: top N documents per group
    const sales = db.collection('sales');
    await 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 }
    ]);
    const accumulatorResult = await sales.aggregate([
      { $group: { _id: "$item", topTwoSales: { $topN: { n: 2, sortBy: { quantity:
 -1 }, output: { quantity: "$quantity", price: "$price" } } } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Window operator usage: top two rounds so far within each partition
    const matchScores = db.collection('matchScores');
    await matchScores.insertMany([
      { _id: 1, player: "Alice", round: 1, score: 20 },
      { _id: 2, player: "Alice", round: 2, score: 35 },
      { _id: 3, player: "Alice", round: 3, score: 28 },
      { _id: 4, player: "Bob", round: 1, score: 15 },
      { _id: 5, player: "Bob", round: 2, score: 40 }
    ]);
    const windowResult = await matchScores.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$player",
```

```
          sortBy: { round: 1 },
          output: {
            topTwoRoundsSoFar: {
              $topN: { n: 2, sortBy: { score: -1 }, output: { round: "$round",
 score: "$score" } },
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } 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']

        # Accumulator usage: top N documents per group
        sales = db['sales']
        sales.insert_many([
            { '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 }
        ])
        accumulator_result = list(sales.aggregate([
            { '$group': { '_id': '$item', 'topTwoSales': { '$topN': { 'n': 2,
'sortBy': { 'quantity': -1 }, 'output': { 'quantity': '$quantity', 'price':
'$price' } } } } }
        ]))
        pprint(accumulator_result)

        # Window operator usage: top two rounds so far within each partition
        match_scores = db['matchScores']
        match_scores.insert_many([
            { '_id': 1, 'player': 'Alice', 'round': 1, 'score': 20 },
            { '_id': 2, 'player': 'Alice', 'round': 2, 'score': 35 },
            { '_id': 3, 'player': 'Alice', 'round': 3, 'score': 28 },
            { '_id': 4, 'player': 'Bob', 'round': 1, 'score': 15 },
            { '_id': 5, 'player': 'Bob', 'round': 2, 'score': 40 }
        ])
        window_result = list(match_scores.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$player',
                    'sortBy': { 'round': 1 },
                    'output': {
                        'topTwoRoundsSoFar': {
                            '$topN': { 'n': 2, 'sortBy': { 'score': -1 }, 'output':
{ 'round': '$round', 'score': '$score' } },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        pprint(window_result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()
```

```
example()
```

$trunc

バージョン 8.0.1 から新規。

Amazon DocumentDB の $trunc演算子は、数値を指定した小数点以下を切り捨て、四捨五入せずに桁を削除します。

パラメータ

- number: 数値に解決される式。
- place: オプション。切り捨てる小数点以下の桁数を指定する -20 ~ 100 の整数式。負の値は 10 進数の左に切り捨てられます。デフォルトは 0 です。

例 (MongoDB シェル)

次の例は、 $trunc演算子を使用して数値を小数点以下第 1 位に切り捨てる方法を示しています。

正のプレース値で切り捨てる

```
db.measurements.insertMany([
  {_id: 1, value: 3.456},
  {_id: 2, value: 7.891},
  {_id: 3, value: 12.345}
]);
```

クエリの例

```
db.measurements.aggregate([
  { $project: { truncated: { $trunc: ["$value", 1] } } }
]);
```

出力

```
[
  {_id: 1, truncated: 3.4},
  {_id: 2, truncated: 7.8},
```

```
  {_id: 3, truncated: 12.3}
]
```

負のプレース値で切り捨てる

place が負の場合、 $trunc は 10 進数の左にある数字をゼロに置き換えます。次の例では、10 進数の左にある最初の桁を使用して値を切り捨てます。

```
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] } } }
]);
```

出力

```
[
  {_id: 1, truncated: 10},
  {_id: 2, truncated: 20},
  {_id: 3, truncated: 30}
]
```

コードの例

$trunc 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の `$tsIncrement`演算子は、タイムスタンプ値から増分序数を長い整数として返します。

パラメータ

- `expression`: タイムスタンプに解決される式。

例 (MongoDB シェル)

次の例は、 $tsIncrement演算子を使用してタイムスタンプ値から序数増分を抽出する方法を示しています。

サンプルドキュメントを作成する

```
db.events.insertMany([
  {_id: 1, ts: Timestamp(1678900000, 1)},
  {_id: 2, ts: Timestamp(1678900000, 2)},
  {_id: 3, ts: Timestamp(1678900001, 1)}
]);
```

クエリの例

```
db.events.aggregate([
  { $project: { increment: { $tsIncrement: "$ts" } } }
]);
```

出力

```
[
  {_id: 1, increment: Long("1")},
  {_id: 2, increment: Long("2")},
  {_id: 3, increment: Long("1")}
]
```

コードの例

$tsIncrement 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の$tsSecond演算子は、タイムスタンプ値の秒部分を長い整数 (Unix エポック秒) として返します。

パラメータ

- expression: タイムスタンプに解決される式。

例 (MongoDB シェル)

次の例は、 $tsSecond演算子を使用してタイムスタンプ値から秒部分を抽出する方法を示しています。

サンプルドキュメントを作成する

```
db.events.insertMany([
  {_id: 1, ts: Timestamp(1678900000, 1)},
  {_id: 2, ts: Timestamp(1678900000, 2)},
  {_id: 3, ts: Timestamp(1678900001, 1)}
]);
```

クエリの例

```
db.events.aggregate([
  { $project: { seconds: { $tsSecond: "$ts" } } }
]);
```

出力

```
[
  {_id: 1, seconds: Long("1678900000")},
  {_id: 2, seconds: Long("1678900000")},
  {_id: 3, seconds: Long("1678900001")}
]
```

コードの例

$tsSecond 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $toBool演算子は、式をブール値に変換します。

パラメータ

- expression: ブール値に変換される式。

注: 文字列は に変換されますtrue。

例 (MongoDB シェル)

次の例は、 $toBool演算子を使用して、さまざまなデータ型からデバイス状態値を正規化する方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.deviceStates.aggregate([
  {
    $project: {
      _id: 1,
      deviceId: 1,
      isActive: { $toBool: "$status" }
    }
  }
]);
```

出力

```
[
  { "_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 }
]
```

コードの例

$toBool コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能

Amazon DocumentDB の$toDate集計演算子は、日付または日付と時刻の文字列を BSON 日付タイプに変換するために使用されます。これは、 $dateToString演算子の逆オペレーションです。

パラメータ

- dateString: BSON 日付タイプに変換される日付または日時の文字列表現。
- format: (オプション) の形式を指定する文字列dateString。指定しない場合、演算子はさまざまな標準的な日付と時刻dateStringの形式で を解析しようとします。
- timezone: (オプション) 変換に使用するタイムゾーンを表す文字列。指定しない場合、ローカルタイムゾーンが使用されます。

例 (MongoDB シェル)

次の例は、 $toDate演算子を使用して日付文字列を BSON 日付タイプに変換する方法を示しています。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.events.aggregate([
  {
    $project: {
      eventName: 1,
      eventTimeDate: { $toDate: "$eventTime" }
    }
  }
]);
```

出力

```
[
  {
    "_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")
  }
]
```

コードの例

$toDate コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能

Amazon DocumentDB の $toDecimal演算子は、値を Decimal128 データ型に変換するために使用されます。これは、正確な 10 進演算を実行したり、ダブルデータ型を使用して正確に表現できない大きな 10 進値を処理したりする必要がある場合に便利です。

パラメータ

- expression: Decimal128 データ型に変換する式。

例 (MongoDB シェル)

この例では、 $toDecimal演算子を使用して文字列値を Decimal128 データ型に変換する方法を示します。

サンプルドキュメントを作成する

```
db.numbers.insertOne({ _id: 1, value: "3.14" });
db.numbers.insertOne({ _id: 2, value: "2.71" });
```

クエリの例

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    decimalValue: { $toDecimal: "$value" }
  }}
])
```

出力

```
[
  { "_id" : 1, "decimalValue" : Decimal128("3.14") },
  { "_id" : 2, "decimalValue" : Decimal128("2.71") }
]
```

コードの例

$toDecimal コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能

Amazon DocumentDB の $toDouble演算子は、値を倍精度の 64 ビット浮動小数点数に変換するために使用されます。これは、元々数値形式でない値に対して算術演算を実行する必要がある場合に便利です。

パラメータ

<expression>: 二重値に変換する式。これは、数値、文字列、またはブール値に解決される任意の有効な式にすることができます。

例 (MongoDB シェル)

この例では、 $toDouble演算子を使用して、数学的計算を実行するために文字列値を数値に変換する方法を示します。

サンプルドキュメントを作成する

```
db.numbers.insertMany([
  { _id: 1, value: "10.5" },
  { _id: 2, value: "20.25" },
  { _id: 3, value: "7" }
])
```

クエリの例

```
db.numbers.aggregate([
  {
    $project: {
      _id: 1,
      value: 1,
      double_value: { $toDouble: "$value" },
      double_plus_five: { $add: [{ $toDouble: "$value" }, 5] }
    }
```

```
  }
])
```

出力

```
[
  { "_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 }
]
```

コードの例

$toDouble コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能

Amazon DocumentDB の $toInt演算子は、入力値を整数データ型に変換するために使用されます。この演算子は、フィールドまたは式が整数として表されるようにする必要がある場合に便利です。これは、特定のオペレーションやデータ処理タスクにとって重要です。

パラメータ

- expression: 整数に変換される式。

例 (MongoDB シェル)

次の例は、 $toInt演算子を使用して文字列値を整数に変換する方法を示しています。

サンプルドキュメントを作成する

```
db.numbers.insertMany([
  { "name": "one", "value": "1" },
  { "name": "hundred", "value": "100" }
]);
```

クエリの例

```
db.numbers.aggregate([
  { $project: {
    "_id": 0,
    "name": 1,
    "intValue": { $toInt: "$value" }
  }}
]);
```

出力

```
{ "name": "one", "intValue": 1 }
{ "name": "hundred", "intValue": 100 }
```

コードの例

$toInt コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能

Amazon DocumentDB の $toLong演算子は、値を 64 ビット整数 (ロング) データ型に変換するために使用されます。これは、文字列やその他のデータ型として保存される可能性のある数値に対して算術演算または比較を実行する必要がある場合に便利です。

パラメータ

- `expression`: 64 ビット整数に変換する式。

例 (MongoDB シェル)

この例では、 $toLong演算子を使用して文字列値を 64 ビット整数に変換する方法を示します。

サンプルドキュメントを作成する

```
db.numbers.insertMany([
  { _id: 1, value: "42" },
  { _id: 3, value: "92233720368547758O7" }
]);
```

クエリの例

```
db.numbers.aggregate([
  {
    $project: {
      _id: 1,
      longValue: { $toLong: "$value" }
    }
  }
])
```

出力

```
[
  { "_id" : 1, "longValue" : 42 },
  { "_id" : 3, "longValue" : 9223372036854775807 }
]
```

コードの例

$toLong コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $toLower演算子は、文字列を小文字に変換するために使用されます。

パラメータ

- expression: 小文字に変換する文字列式。

例 (MongoDB シェル)

次の例は、 $toLower演算子を使用して Deskフィールドを小文字に変換する方法を示しています。

サンプルドキュメントを作成する

```
db.locations.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-BVV-021" },
  { "_id": 2, "Desk": "Munich-HGG-32a" }
]);
```

クエリの例

```
db.locations.aggregate([
  { $project: { item: { $toLower: "$Desk" } } }
]);
```

出力

```
{ "_id" : 1, "item" : "düsseldorf-bvv-021" }
{ "_id" : 2, "item" : "munich-hgg-32a" }
```

コードの例

$toLower コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能

Amazon DocumentDB の `$toObjectId`演算子は、ObjectId の文字列表現を実際の ObjectId データ型に変換するために使用されます。これは、ObjectIds、ObjectId の文字列表現として保存されたデータを操作する場合に便利です。

パラメータ

- `expression`: 有効な ObjectId を表す文字列式。

例 (MongoDB シェル)

次の例は、 `$toObjectId`演算子を使用して ObjectId の文字列表現を ObjectId データ型に変換する方法を示しています。

サンプルドキュメントを作成する

```
db.employees.insertMany([
  { _id: 1, empId:"64e5f8886218c620cf0e8f8a", name: "Carol Smith", employeeId:
 "c720a" },
  { _id: 2, empId:"64e5f94e6218c620cf0e8f8c", name: "Bill Taylor", employeeId:
 "c721a" }
]);
```

クエリの例

```
db.employees.aggregate([
  { $project: {
    "empIdAsObjectId": {$toObjectId: "$empId"}}
  }
]);
```

出力

```
[
  { _id: 1, empIdAsObjectId: ObjectId('64e5f8886218c620cf0e8f8a') },
  { _id: 2, empIdAsObjectId: ObjectId('64e5f94e6218c620cf0e8f8c') }
]
```

コードの例

$toObjectId コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能

Amazon DocumentDB の $toString演算子は、任意のタイプ (null を除く) の値を文字列表現に変換するために使用されます。これは、元々文字列形式でない値に対して文字列オペレーションを実行する必要がある場合に便利です。

パラメータ

- expression: 文字列に変換する式。

例 (MongoDB シェル)

次の例は、 $toString演算子を使用して数値を文字列に変換する方法を示しています。

サンプルドキュメントを作成する

```
db.numbers.insertMany([
  { "_id": 1, "value": 42 },
  { "_id": 2, "value": 3.14 }
]);
```

クエリの例

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    valueAsString: { $toString: "$value" }
  }}
]);
```

出力

```
{ "_id": 1, "valueAsString": "42" }
```

```
{ "_id": 2, "valueAsString": "3.14" }
```

コードの例

$toString コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $toUpper演算子は、文字列を大文字に変換するために使用されます。

パラメータ

- expression: 大文字に変換する文字列式。

例 (MongoDB シェル)

次の例は、 $toUpper演算子を使用して Deskフィールドを大文字に変換する方法を示しています。

サンプルドキュメントを作成する

```
db.locations.insertMany([
  { "_id": 1, "Desk": "düsseldorf-bvv-021" },
  { "_id": 2, "Desk": "munich-hgg-32a" }
]);
```

クエリの例

```
db.locations.aggregate([
  { $project: { item: { $toUpper: "$Desk" } } }
]);
```

出力

```
{ "_id" : 1, "item" : "DüSSELDORF-BVV-021" }
{ "_id" : 2, "item" : "MUNICH-HGG-32A" }
```

コードの例

$toUpper コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Amazon DocumentDB の $toUUID演算子は、文字列値を UUID (バイナリサブタイプ 4) に変換します。

パラメータ

- expression: UUID 形式の文字列に解決される式 (例: "12345678-1234-1234-1234-123456789abc")。

例 (MongoDB シェル)

次の例は、 $toUUID演算子を使用して文字列値を UUID 形式に変換する方法を示しています。

サンプルドキュメントを作成する

```
db.records.insertMany([
  {_id: 1, uuidStr: "550e8400-e29b-41d4-a716-446655440000"},
  {_id: 2, uuidStr: "6ba7b810-9dad-11d1-80b4-00c04fd430c8"}
]);
```

クエリの例

```
db.records.aggregate([
  { $project: { uuid: { $toUUID: "$uuidStr" } } }
]);
```

出力

```
[
  {_id: 1, uuid: UUID("550e8400-e29b-41d4-a716-446655440000")},
  {_id: 2, uuid: UUID("6ba7b810-9dad-11d1-80b4-00c04fd430c8")}
]
```

コードの例

$toUUID 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能

Amazon DocumentDB の $trim演算子は、文字列から先頭および/または末尾の空白文字を削除するために使用されます。

パラメータ

- input: トリミングする文字列式。
- chars: (オプション) 入力の最初と最後にトリミングする文字を指定します。デフォルトは空白です。

例 (MongoDB シェル)

次の例は、 $trim演算子を使用して文字列から先頭と末尾の空白を削除する方法を示しています。

サンプルドキュメントを作成する

```
db.people.insertMany([
  { "name": "   John Doe   " },
  { "name": "   Bob Johnson   " }
])
```

クエリの例

```
db.people.aggregate([
  { $project: {
    "name": { $trim: {input: "$name"}}
  }}
])
```

出力

```
[
  { "name": "John Doe" },
```

```
  { "name": "Bob Johnson" }
]
```

コードの例

$trim コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 集計演算子は、指定されたフィールドの BSON データ型を返します。これは、集計オペレーション中にフィールド値のデータ型を識別するのに役立ちます。

パラメータ

- expression: 返される型を持つフィールドまたは式。

例 (MongoDB シェル)

次の例は、 $type演算子を使用して、各製品の料金フィールドのデータ型を識別する方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.inventory.aggregate([
  {
    $project: {
      item: 1,
      price: 1,
      priceType: { $type: "$price" }
    }
```

```
  }
]);
```

出力

```
[
  { _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' }
]
```

コードの例

$type 集計演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0 の新機能

Elastic クラスターではサポートされていません。

Amazon DocumentDB の$unset集約ステージでは、ドキュメントからフィールドを削除できます。

パラメータ

- `expression`: フィールド名または複数のフィールド名のリスト。

例 (MongoDB シェル)

次の例は、$unset集約ステージを使用して priceフィールドを削除する方法を示しています。

サンプルドキュメントを作成する

```
db.inventory.insertMany([
  { item: "pencil", quantity: 100, price: 0.24},
  { item: "pen", quantity: 204, price: 1.78 }
]);
```

集約の例

```
db.inventory.aggregate([
  {
    $unset: "price"
  }
])
```

出力

```
[
  {
    _id: ObjectId('69248951d66dcae121d2950d'),
    item: 'pencil',
    quantity: 100
  },
  {
    _id: ObjectId('69248951d66dcae121d2950e'),
    item: 'pen',
    quantity: 204
  }
]
```

コードの例

$unset コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、入力ドキュメントから配列フィールドを分解して、各要素のドキュメントを出力するために使用されます。これは、データのフィルタリング、ソート、変換など、配列の個々の要素に対してオペレーションを実行する場合に便利です。

パラメータ

- path: 巻き出す配列フィールドへのパス。
- includeArrayIndex: (オプション) 配列要素のインデックスを保持する新しいフィールドの名前を指定します。
- preserveNullAndEmptyArrays: (オプション) 配列フィールドが null または空の配列の場合、オペレーションが元のドキュメントを保持するかどうかを決定します。

例 (MongoDB シェル)

次の例は、 $unwind演算子を使用して配列フィールドを分解し、個々の要素に対して追加のオペレーションを実行する方法を示しています。

サンプルドキュメントを作成する

```
db.people.insertMany([
  { _id: 1, name: "jon", hobbies: ["painting", "dancing", "singing"] },
  { _id: 2, name: "jane", hobbies: ["reading", "swimming"] },
  { _id: 3, name: "jack", hobbies: [] }
])
```

クエリの例

```
db.people.aggregate([
  { $unwind: "$hobbies" }
])
```

出力

```
[
  { _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' }
]
```

コードの例

$unwind コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0 の新機能

Elastic クラスターではサポートされていません。

Amazon DocumentDB の $vectorSearch演算子を使用すると、ベクトル検索を実行できます。ベクトル検索は、距離または類似度メトリクスを使用してベクトル表現を比較することで、機械学習で同様のデータポイントを見つけるために使用される方法です。この機能は、JSON ベースのドキュメントデータベースの柔軟性と豊富なクエリとベクトル検索の能力を組み合わせることで、セマンティック検索、製品レコメンデーションなどの機械学習や生成 AI のユースケースを構築できます。

パラメータ

- <exact> (オプション): Exact Nearest Neighbor (ENN) または Approximate Nearest Neighbor (ANN) 検索を実行するかどうかを指定するフラグ。値は次のいずれかになります。
- false - ANN 検索を実行するには
- true - ENN 検索を実行するには

省略するか false に設定する場合は、 numCandidatesが必要です。

```
- `<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.
```

例 (MongoDB シェル)

次の例は、 $vectorSearch演算子を使用して、ベクトル表現に基づいて同様の製品説明を検索する方法を示しています。

サンプルドキュメントを作成する

```
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]
  }
]);
```

ベクトル検索インデックスを作成する

```
db.runCommand(
    {
        createIndexes: "products",
        indexes: [{
            key: {
                "description_vector": "vector"
            },
            vectorOptions: {
                type: "hnsw",
                dimensions: 3,
                similarity: "cosine",
                m: 16,
                efConstruction: 64
            },
            name: "description_index"
        }]
    }
);
```

クエリの例

```
db.products.aggregate([
  { $vectorSearch: {
      index: "description_index",
```

```
      limit: 2,
      numCandidates: 10,
      path: "description_vector",
      queryVector: [0.1, 0.2, 0.3]
    }
  }
]);
```

出力

```
[
  {
    "_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 ]
  }
]
```

コードの例

$vectorSearch コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$week演算子は、ISO 8601 標準に基づいて日付の週番号 (0～53) を返します。週番号は年と曜日に基づいて計算され、月曜日を週の最初の日として使用します。

パラメータ

なし

例 (MongoDB シェル)

次の例は、 $week演算子を使用して特定の日付の週番号を取得する方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.events.aggregate([
  { $project: {
    _id: 1,
    week: { $week: "$date" }
  }}
]);
```

出力

```
[
  { "_id": 1, "week": 1 },
  { "_id": 2, "week": 2 },
  { "_id": 3, "week": 53 }
]
```

コードの例

$week コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $year演算子は、日付またはタイムスタンプから年コンポーネントを抽出します。

パラメータ

- expression: 年コンポーネントを抽出する日付またはタイムスタンプ式。

例 (MongoDB シェル)

次の例は、 $year演算子を使用して日付フィールドから年コンポーネントを抽出する方法を示しています。

サンプルドキュメントを作成する

```
db.events.insertMany([
  { "_id": 1, "date": ISODate("2023-04-15T00:00:00Z") },
  { "_id": 3, "date": ISODate("2021-12-31T00:00:00Z") }
]);
```

クエリの例

```
db.events.aggregate([
  { $project: { year: { $year: "$date" } } }
]);
```

出力

```
[
  { "_id": 1, "year": 2023 },
  { "_id": 3, "year": 2021 }
]
```

コードの例

$year コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $zip演算子を使用すると、複数の配列を 1 つのタプルの配列 (キーと値のペア) に結合できます。これは、ドキュメント内のさまざまなソースまたは配列のデータを組み合わせて、新しいドキュメントまたはオブジェクトを作成する必要がある場合に便利です。

パラメータ

- inputs: 配列に解決される式の配列。これらの配列は 1 つのタプル配列に結合されます。
- useLongestLength: (オプション) の場合true、出力配列は最長の入力配列の長さになり、短い配列はnull値でパディングされます。の場合false、出力配列は最短の入力配列の長さになります。
- defaults: (オプション) 対応する入力配列が最長の入力配列より短く、 useLongestLength がの場合にタプルに使用するデフォルト値の配列true。

例 (MongoDB シェル)

次の例は、 $zip演算子を使用して 2 つの配列を 1 つのタプル配列に結合する方法を示しています。

サンプルドキュメントを作成する

```
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"]
  }
])
```

クエリの例

```
db.grades.aggregate([
  {
    $project: {
      "name": 1,
      "scoredClasses": {
        $zip: {
          inputs: ["$scores", "$classes"],
          useLongestLength: true,
          defaults: [null, null]
        }
      }
    }
  }
])
```

出力

```
[
  {
    "_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"]
    ]
  }
]
```

コードの例

$zip コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

地理空間

このセクションでは、Amazon DocumentDB でサポートされている地理空間演算子に関する詳細情報を提供します。

トピック

- $geometry
- $geoIntersects
- $geoWithin
- $maxDistance
- $minDistance
- $near
- $nearSphere

$geometry

Amazon DocumentDB の $geometry演算子は、地理空間クエリの一部として GeoJSON ジオメトリオブジェクトを指定するために使用されます。この演算子は、 $geoWithinや などの他の地理空間クエリ演算子と組み合わせて$geoIntersects使用され、データに対して空間クエリを実行します。

Amazon DocumentDB では、 $geometry演算子は次の GeoJSON ジオメトリタイプをサポートしています。

- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon
- GeometryCollection

パラメータ

- type: GeoJSON ジオメトリオブジェクトのタイプ。例: 、 Point Polygonなど。
- coordinates: ジオメトリを表す座標の配列。座標配列の構造は、ジオメトリタイプによって異なります。

例 (MongoDB シェル)

次の例は、 $geometry演算子を使用して Amazon DocumentDB で$geoIntersectsクエリを実行する方法を示しています。

サンプルドキュメントを作成する

```
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]
      ]]
    }
  }
]);
```

クエリの例

```
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]
        ]]
      }
    }
  }
})
```

出力

```
[
  {
    "_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]
        ]
      ]
    }
  }
]
```

コードの例

$geometry コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $geoIntersects演算子は、地理空間データが指定された GeoJSON オブジェクトと交差するドキュメントを検索するために使用されます。この演算子は、ポリゴンやマルチポリゴンなど、特定の地理的形状との空間関係に基づいてドキュメントを識別する必要があるアプリケーションに役立ちます。

パラメータ

- $geometry: 交差をチェックするシェイプを表す GeoJSON オブジェクト。サポートされている GeoJSON オブジェクトタイプは、Point、LineString、Polygon、および です MultiPolygon。

例 (MongoDB シェル)

次の例は、 $geoIntersects演算子を使用して Amazon DocumentDB 内の特定の座標セットの状態名を検索する方法を示しています。

サンプルドキュメントを作成する

```
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]
      ]]
    }
  }
]);
```

クエリの例

```
var location = [-73.965355, 40.782865];

db.states.find({
  "loc": {
    "$geoIntersects": {
      "$geometry": {
        "type": "Point",
        "coordinates": location
      }
    }
  }
}, {
  "name": 1
});
```

出力

```
{ "_id" : ObjectId("536b0a143004b15885c91a2c"), "name" : "New York" }
```

コードの例

$geoIntersects コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $geoWithin演算子は、位置データ (GeoJSON オブジェクトとして表されます) がポリゴンやマルチポリゴンなどの指定されたシェイプ内に完全に含まれているドキュメントを検索するために使用されます。これは、特定の地理的リージョン内にあるオブジェクトのクエリに役立ちます。

パラメータ

- $geometry: クエリ対象のシェイプを表す GeoJSON オブジェクト。

例 (MongoDB シェル)

次の例は、 $geoWithin演算子を使用して、ニューヨーク州内にあるすべての空港を検索する方法を示しています。

サンプルドキュメントを作成する

```
// 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]
        }
    }
]);
```

クエリの例

```
var state = db.states.findOne({"name": "New York"});

db.airports.find({
    "loc": {
        "$geoWithin": {
            "$geometry": state.loc
        }
    }
}, {
    "name": 1,
    "type": 1,
    "code": 1,
    "_id": 0
});
```

出力

```
[
  {
    "name": "John F. Kennedy International Airport",
    "type": "airport",
    "code": "JFK"
  },
  {
    "name": "LaGuardia Airport",
    "type": "airport",
    "code": "LGA"
  }
]
```

コードの例

$geoWithin コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $maxDistance演算子は、クエリ結果に含めるためにドキュメントを含める必要がある GeoJSON ポイントからの最大距離 (メートル単位) を指定するために使用されます。この演算子は、地理空間クエリを実行するために $nearSphere演算子と組み合わせて使用されます。

パラメータ

- $maxDistance: クエリ結果に含めるためにドキュメントが置かなければならない参照ポイントからの最大距離 (メートル単位) 。

例 (MongoDB シェル)

次の例は、Amazon DocumentDB で $maxDistance演算子を使用して、ボストンから 100 km 以内のすべての州都を検索する方法を示しています。

サンプルドキュメントを作成する

```
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] } }
]);
```

クエリの例

```
db.capitals.find(
  {
    location: {
      $nearSphere: {
        $geometry: { type: "Point", coordinates: [-71.0589, 42.3601] },
        $maxDistance: 100000
      }
    }
  },
  { state: 1, city: 1, _id: 0 }
);
```

出力

```
[
  { "state": "Rhode Island", "city": "Providence" },
  { "state": "New Hampshire", "city": "Concord" }
]
```

コードの例

$maxDistance コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

$minDistance は、中心点から少なくとも指定された最小距離にあるドキュメントをフィルタリング$geoNearするために、 $nearSphereまたは と組み合わせて使用される検索演算子です。この演算子は Amazon DocumentDB でサポートされており、MongoDB の対応する関数と同様に機能します。

パラメータ

- $minDistance: 結果にドキュメントを含めるための中心点からの最小距離 (メートル単位) 。

例 (MongoDB シェル)

この例では、ワシントン州シアトルの特定の場所から半径 2 km 以内のすべてのレストランを見つけます。

サンプルドキュメントを作成する

```
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]
    }
  }
]);
```

クエリの例

```
db.usarestaurants.find({
  "location": {
    "$nearSphere": {
      "$geometry": {
        "type": "Point",
        "coordinates": [-122.3516, 47.6156]
      },
      "$minDistance": 1,
      "$maxDistance": 2000
    }
  }
}, {
  "name": 1
});
```

出力

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "name" : "Noodle House" }
{ "_id" : ObjectId("611f3da985009a81ad38e74c"), "name" : "Pike Place Grill" }
```

コードの例

$minDistance コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $near演算子は、指定されたポイントに地理的に近いドキュメントを検索するために使用されます。距離順に並べられたドキュメントが返され、最も近いドキュメントが最初に返されます。この演算子には 2 次元地理空間インデックスが必要で、位置データの近接クエリに役立ちます。

パラメータ

- $geometry: ニアクエリの中心点を定義する GeoJSON ポイントオブジェクト。
- $maxDistance: (オプション) ドキュメントがクエリに一致するように指定ポイントからメートル単位の最大距離。
- $minDistance: (オプション) ドキュメントがクエリに一致するように指定ポイントからメートル単位の最小距離。

インデックスの要件

- 2dsphere index: GeoJSON ポイントデータの地理空間クエリに必要です。

例 (MongoDB シェル)

次の例は、 $near演算子を使用して、ワシントン州シアトルの特定の場所に最も近いレストランを見つける方法を示しています。

サンプルドキュメントを作成する

```
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] }
  }
]);
```

2dsphere インデックスを作成する

```
db.usarestaurants.createIndex({ "location": "2dsphere" });
```

GeoJSON ポイントを使用したクエリの例

```
db.usarestaurants.find({
  location: {
    $near: {
      $geometry: {
        type: "Point",
```

```
        coordinates: [-122.3516, 47.6156]
      },
      $maxDistance: 100,
      $minDistance: 10
    }
  }
});
```

出力

```
{
  "_id" : ObjectId("69031ec9ea1c2922a1ce5f4a"),
  "name" : "Noodle House",
  "city" : "Seattle",
  "state" : "Washington",
  "rating" : 4.8,
  "location" : {
    "type" : "Point",
    "coordinates" : [ -122.3517, 47.6159 ]
  }
}
```

コードの例

$near コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $nearSphere演算子は、地理空間ポイントから指定された距離内にあるドキュメントを検索するために使用されます。この演算子は、特定の場所の特定の半径内のすべてのレストランを検索するなど、地理空間クエリに特に役立ちます。

パラメータ

- $geometry: 参照ポイントを表す GeoJSON オブジェクト。type フィールドと coordinatesフィールドを持つPointオブジェクトである必要があります。
- $minDistance: (オプション) ドキュメントが置かなければならない参照ポイントからの最小距離 (メートル単位) 。
- $maxDistance: (オプション) ドキュメントが置かなければならない参照ポイントからの最大距離 (メートル単位) 。

例 (MongoDB シェル)

この例では、ワシントン州シアトルの特定の場所から 2 キロメートル (2000 メートル) 以内にすべてのレストランがあります。

サンプルドキュメントを作成する

```
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] }
  }
]);
```

クエリの例

```
db.usarestaurants.find({
  location: {
    $nearSphere: {
      $geometry: {
        type: "Point",
        coordinates: [-122.3516, 47.6156]
      },
      $minDistance: 1,
      $maxDistance: 2000
    }
  }
}, {
  name: 1
});
```

出力

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "name" : "Noodle House" }
{ "_id" : ObjectId("611f3da985009a81ad38e74c"), "name" : "Pike Place Grill" }
```

コードの例

$nearSphere コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

クエリおよびプロジェクション演算子

このセクションでは、Amazon DocumentDB でサポートされているクエリ演算子と射影演算子に関する詳細情報を提供します。

トピック

- $
- $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

$

$ 射影演算子は、配列フィールドの内容を制限して、クエリ条件に一致する最初の要素のみを返します。これは、単一の一致する配列要素を射影するために使用されます。

パラメータ

- `field.$`: 位置演算子を使用して、最初に一致する要素を射影する配列フィールド。

例 (MongoDB シェル)

次の例は、$射影演算子を使用して一致する配列要素のみを返す方法を示しています。

サンプルドキュメントを作成する

```
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] }
]);
```

クエリの例

```
db.students.find(
  { grades: { $gte: 90 } },
  { name: 1, "grades.$": 1 }
);
```

出力

```
{ "_id" : 1, "name" : "Alice", "grades" : [ 92 ] }
{ "_id" : 2, "name" : "Bob", "grades" : [ 92 ] }
{ "_id" : 3, "name" : "Charlie", "grades" : [ 95 ] }
```

この例では、学生ごとに 90 以上の最初のグレードのみが返されます。

コードの例

$ 射影演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $all演算子は、フィールドの値が配列であり、配列内の要素の順序に関係なく、指定されたすべての要素を含むドキュメントを一致させるために使用されます。

パラメータ

- field: チェックするフィールドの名前。
- [value1, value2, ...]: 配列内で一致する値のリスト。

$all**式**$elemMatch内での の使用

$all 式内での $elemMatch演算子の使用に関する制限$all 式内での $elemMatch の使用については、「」を参照してください。

フィールド名のドル ($)

ネストされたオブジェクトの $allの$プレフィックス付きフィールドのクエリに関する制限フィールド名のドル ($) とドット (.)については、「」を参照してください。

例 (MongoDB シェル)

次の例は、$all演算子を使用して「Colors」フィールドが「Red」と「Blue」の両方を含む配列であるドキュメントを取得する方法を示しています。

サンプルドキュメントを作成する

```
db.example.insertMany([
  { "Item": "Pen", "Colors": ["Red", "Blue", "Green"] },
  { "Item": "Notebook", "Colors": ["Blue", "White"] },
  { "Item": "Poster Paint", "Colors": ["Red", "Yellow", "White"] }
])
```

クエリの例

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

出力

```
{
  "_id" : ObjectId("6137d6c5b3a1d35e0b6ee6ad"),
  "Item" : "Pen",
  "Colors" : [
          "Red",
          "Blue",
          "Green"
  ]
}
```

コードの例

$all コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $and演算子は、複数の式を組み合わせて 1 つの条件として評価するために使用します。指定されたすべての式が に評価trueされた場合は が返されtrue、falseそれ以外の場合は が返されます。この演算子は、クエリに複数の条件を適用するのに役立ちます。

パラメータ

- expression1: ブール値に評価される必須の式。
- expression2: ブール値に評価される必須の式。
- ...: ブール値に評価される追加の必須式。

例 (MongoDB シェル)

次の例は、 $and演算子を使用して、「age」フィールドが 18 より大きく、「status」フィールドが「active」である「users」コレクション内のすべてのドキュメントを検索する方法を示しています。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.users.find({
  $and: [
    { age: { $gt: 18 } },
    { status: "active" }
  ]
});
```

出力

```
[
  { "_id" : ObjectId("614e3c4b63f5892e7c4e2345"), "name" : "John", "age" : 25,
 "status" : "active" },
  { "_id" : ObjectId("614e3c4b63f5892e7c4e2347"), "name" : "Alice", "age" : 22,
 "status" : "active" }
]
```

コードの例

$and コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $bitsAllClear演算子は、フィールドで指定されたすべてのビットがクリアされているドキュメントを照合するために使用されます (0 に設定) 。この演算子は、保存されたデータに対してビット単位のオペレーションを実行するのに役立ちます。

パラメータ

- field: クリアされている指定されたビットをチェックするフィールド。
- value: チェックするビットを指定する数値ビットマスク、またはチェックするビット位置のリスト。数値ビットマスクは、バイナリ (0b...)、10 進数、16 進数 (0x...)、8 進数 (0o...)、またはバイナリ (BinData) 形式にすることができます。ビット位置のリストでは、最下位ビットの位置は 0 です。

例 (MongoDB シェル)

次の例は、Amazon DocumentDB での $bitsAllClear演算子の使用を示しています。

サンプルドキュメントを作成する

```
db.collection.insertMany([
  { _id: 1, bits: 0b1010 },
  { _id: 2, bits: 0b1100 },
  { _id: 3, bits: 0b0101 }
]);
```

クエリの例

```
db.collection.find({
  bits: { $bitsAllClear: 0b0011 }
})
```

出力

```
{ "_id" : 2, "bits" : 12 }
```

クエリは、ビットマスクで指定されたすべてのビット 0b0011 (最下位 2 つのビット) が bitsフィールドでクリアされているかどうかを確認します。_id 2 のドキュメントは、 bitsフィールドでこれらのビットがクリアされているため、この条件を満たします。

コードの例

$bitsAllClear コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $bitsAllSet演算子は、フィールドで特定のビットセットがすべて 1 に設定されているドキュメントをクエリするために使用されます。この演算子を使用すると、フィールド値に対してビット単位のオペレーションを実行でき、数値内の個々のビットの状態を確認する必要がある場合に役立ちます。

パラメータ

- field: ビット単位のオペレーションを実行する数値フィールドの名前。
- value: チェックするビットを指定する数値ビットマスク、またはチェックするビット位置のリスト。数値ビットマスクは、バイナリ (0b...)、10 進数、16 進数 (0x...)、8 進数 (0o...)、またはバイナリ (BinData) 形式にすることができます。ビット位置のリストでは、最下位ビットの位置は 0 です。

例 (MongoDB シェル)

次の例は、 $bitsAllSet演算子を使用して、 flagsフィールドにビットマスクで指定されたすべてのビットが設定されているドキュメントを検索する方法を示しています。

サンプルドキュメントを作成する

```
db.collection.insert([
  { _id: 1, flags: 0b1010 },
  { _id: 2, flags: 0b1100 },
  { _id: 3, flags: 0b1110 }
])
```

クエリの例

```
db.collection.find({ flags: { $bitsAllSet: 0b1100 } })
```

出力

```
{ "_id": 2, "flags": 12 },
{ "_id": 3, "flags": 14 }
```

この例では、クエリは、 flagsフィールドにビットマスク 0b1100 (10 進値 12 を表す) で指定されたすべてのビットが設定されているドキュメントをチェックします。2 と 3 _id のドキュメントは、flagsフィールド値にすべての必須ビット (3 番目と 4 番目の重要度の低いビット) が設定されているため、この基準に一致します。

コードの例

$bitsAllSet コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $bitsAnyClear演算子は、フィールド内の指定されたビットのいずれかがクリアされているドキュメントを照合するために使用されます (0 に設定) 。これは、ドキュメントに保存されているフィールド値に対してビット単位のオペレーションを実行するのに役立ちます。

パラメータ

- field: 確認するフィールド。
- value: チェックするビットを指定する数値ビットマスク、またはチェックするビット位置のリスト。数値ビットマスクは、バイナリ (0b...)、10 進数、16 進数 (0x...)、8 進数 (0o...)、またはバイナリ (BinData) 形式にすることができます。ビット位置のリストでは、最下位ビットの位置は 0 です。

例 (MongoDB シェル)

次の例は、 $bitsAnyClear演算子を使用して、itemsコレクションの statusフィールドでビットがクリアであるかどうかを確認する方法を示しています。

サンプルドキュメントを作成する

```
db.items.insertMany([
  { "_id": 1, "status": 7 },
  { "_id": 2, "status": 15 },
  { "_id": 3, "status": 31 }
]);
```

クエリの例

```
db.items.find({ "status": { $bitsAnyClear: 8 } })
```

出力

```
{ "_id" : 1, "status" : 7 }
```

この例では、クエリは、ビットマスク 8 (バイナリ) で statusフィールドにビットクリア (0) があるドキュメントをチェックします1000。(7バイナリ 111) statusの値を持つドキュメントは、指定されたビットマスクで少なくとも 1 ビットクリアであるため、クエリと一致します。一致するクリアビットは、4 番目に重要度の低いビットです。

コードの例

$bitsAnyClear コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $bitsAnySet演算子は、フィールド内の指定されたビットで少なくとも 1 ビットが 1 に設定されているドキュメントをクエリするために使用されます。この演算子を使用すると、フィールドに保存されている値に対してビット単位のオペレーションを実行できるため、ビット単位の特性を持つデータの効率的なクエリと分析が可能になります。

パラメータ

- field: ビット単位のオペレーションを適用するフィールド名。
- value: チェックするビットを指定する数値ビットマスク、またはチェックするビット位置のリスト。数値ビットマスクは、バイナリ (0b...)、10 進数、16 進数 (0x...)、8 進数 (0o...)、またはバイナリ (BinData) 形式にすることができます。ビット位置のリストでは、最下位ビットの位置は 0 です。

例 (MongoDB シェル)

次の例は、 $bitsAnySet演算子を使用して、 flagsフィールドに少なくとも 1 ビットが設定されているドキュメントを検索する方法を示しています。

サンプルドキュメントを作成する

```
db.collection.insertMany([
  { _id: 1, flags: 0b1010 },
  { _id: 2, flags: 0b1100 },
  { _id: 3, flags: 0b0011 },
  { _id: 4, flags: 0b0100 }
]);
```

クエリの例

```
db.collection.find({
```

```
  flags: { $bitsAnySet: 0b1010 }
})
```

出力

```
{ "_id" : 1, "flags" : 10 }
{ "_id" : 2, "flags" : 12 }
{ "_id" : 3, "flags" : 3 }
```

クエリは、ビットマスクで指定されたビットの少なくとも 1 つが 0b1010 flagsフィールドに設定されているドキュメントを返します。

コードの例

$bitsAnySet コマンドを使用するためのコード例を表示するには、使用する言語のタブを選択します。

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 の $comment演算子は、クエリにコメントをアタッチするために使用されます。これは、クエリに関する追加のコンテキストや情報を提供するのに役立ちます。これは、デバッグまたはドキュメント化の目的で役立ちます。添付されたコメントは、db.currentOp() などのオペレーションの出力の一部として表示されます。

パラメータ

- `string`: クエリにアタッチされたコメント。

例 (MongoDB シェル)

次の例は、Amazon DocumentDB で $comment演算子を使用する方法を示しています。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.users.find({ age: { $gt: 25 } }, { _id: 0, name: 1, age: 1 }).comment("Retrieve
 users older than 25");
```

出力

```
{ "name" : "John Doe", "age" : 30 }
{ "name" : "Bob Johnson", "age" : 35 }
```

コードの例

$comment コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $elemMatch演算子は、配列フィールドをクエリし、配列内の少なくとも 1 つの要素が指定された条件に一致するドキュメントを返すために使用されます。この演算子は、ネストされた配列または埋め込みドキュメントを持つ複雑なデータ構造がある場合に特に便利です。

プランナーバージョン 2.0 で のインデックスサポートが追加されました$elemMatch。

パラメータ

- field: クエリする配列フィールド。
- query: 配列要素と照合する基準。

$all式$elemMatch内での の使用

$all 式内での $elemMatch演算子の使用に関する制限$all 式内での $elemMatch の使用については、「」を参照してください。

例 (MongoDB シェル)

次の例は、 $elemMatch演算子を使用して、parts配列に指定された条件に一致する要素が少なくとも 1 つあるドキュメントを検索する方法を示しています。

サンプルドキュメントを作成する

```
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 }] }
]);
```

クエリの例

```
db.col.find({
  parts: { "$elemMatch": { part: "xyz", qty: { $lt: 11 } } }
})
```

出力

```
{ "_id" : 1, "parts" : [ { "part" : "xyz", "qty" : 10 }, { "part" : "abc", "qty" :
 20 } ] }
{ "_id" : 2, "parts" : [ { "part" : "xyz", "qty" : 5 }, { "part" : "abc", "qty" :
 10 } ] }
```

コードの例

$elemMatch コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $eq演算子は、フィールドの値が指定された値と等しいドキュメントを照合するために使用されます。この演算子は、 `find()`メソッドで一般的に使用され、指定された条件を満たすドキュメントを取得します。

パラメータ

- `field` : 等価条件をチェックするフィールド。
- `value` : フィールドと比較する値。

例 (MongoDB シェル)

次の例は、 $eq演算子を使用して、 nameフィールドが と等しいすべてのドキュメントを検索する方法を示しています"Thai Curry Palace"。

サンプルドキュメントを作成する

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

クエリの例

```
db.restaurants.find({ name: { $eq: "Thai Curry Palace" } });
```

出力

```
{ "_id" : ObjectId("68ee586f916df9d39f3d9414"), "name" : "Thai Curry Palace",
 "cuisine" : "Thai", "features" : [ "Private Dining" ] }
```

コードの例

$eq コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

$exists 演算子は、フィールドがドキュメントに存在するかどうかをチェックするために使用されます。$exists特に、インデックス付きフィールドがすべてのドキュメントに存在するわけではない Amazon DocumentDB でスパースインデックスを操作する場合に便利です。

クエリで作成したスパースなインデックスを使用するには、インデックスをカバーするフィールドで $exists 句を使用する必要があります。を省略した場合$exists、Amazon DocumentDB はクエリにスパースインデックスを使用しません。

パラメータ

- field: 存在を確認するフィールド名。
- value: フィールドが一致するドキュメントに存在する (truefalse) か存在しない (true) かを指定するブール値 (または false)。

例 (MongoDB シェル)

次の例は、foodコレクション内の special_dietsフィールドでスパースインデックスを持つ $exists演算子の使用を示しています。

サンプルドキュメントを作成する

```
db.food.insertMany([
  { _id: 1, name: "Apple", special_diets: ["vegetarian", "gluten-free"] },
  { _id: 2, name: "Broccoli" },
  { _id: 3, name: "Chicken", special_diets: ["dairy-free"] }
]);
```

「special_diets」フィールドにスパースインデックスを作成する

```
db.food.createIndex({ "special_diets": 1 }, { sparse: true, name:
 "special_diets_sparse_asc" });
```

クエリの例

```
db.food.find({ "special_diets": { $exists: true } });
```

出力

```
[
  { "_id" : 1, "name" : "Apple", "special_diets" : [ "vegetarian", "gluten-free" ] },
  { "_id" : 3, "name" : "Chicken", "special_diets" : [ "dairy-free" ] }
]
```

コードの例

$exists コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能。

Elastic クラスターではサポートされていません。

Amazon DocumentDB の $expr演算子を使用すると、クエリ言語内で集計式を使用できます。これにより、集約パイプラインステージを使用する方法と同様に、ドキュメント内のフィールドに対して複雑な比較と計算を実行できます。

演算子に配置された $eq、、$lt$lte、$gt、および $gte 比較$expr演算子は、$lookupステージで参照されるfromコレクションのインデックスを含むインデックスを使用できます。

制限事項

- インデックスはフィールドと定数の比較にのみ使用できるため、オペランドは定数に解決する必要があります。たとえば、 $aと定数値の比較ではインデックスを使用できますが、 $aと の比較$bでは使用できません。
- マルチキー、部分インデックス、スパースインデックスは使用されません。このような場合、クエリはコレクションスキャンにフォールバックします。
- let オペランドが空または欠落した値に解決された場合、比較ではターゲットインデックスシークを使用できません。この場合、Amazon DocumentDB はフルインデックススキャンを実行します。

パラメータ

- expression: ブール値を返す式。ドキュメントフィールドで比較と計算を実行できます。

例 (MongoDB シェル)

次の例は、 $expr演算子を使用して、 manufacturingCostフィールドが priceフィールドより大きいすべてのドキュメントを検索する方法を示しています。

サンプルドキュメントを作成する

```
db.inventory.insertMany([
  { item: "abc", manufacturingCost: 500, price: 100 },
  { item: "def", manufacturingCost: 300, price: 450 },
  { item: "ghi", manufacturingCost: 400, price: 120 }
]);
```

クエリの例

```
db.inventory.find({
  $expr: {
    $gt: ["$manufacturingCost", "$price"]
  }
})
```

出力

```
{ "_id" : ObjectId("60b9d4d68d2cac581bc5a89a"), "item" : "abc", "manufacturingCost" :
 500, "price" : 100 },
```

```
{ "_id" : ObjectId("60b9d4d68d2cac581bc5a89c"), "item" : "ghi", "manufacturingCost" :
 400, "price" : 120 }
```

コードの例

$expr コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $gt演算子は、指定されたフィールドの値が指定された値より大きいドキュメントを選択するために使用されます。この演算子は、数値比較に基づいてデータをフィルタリングおよびクエリするのに役立ちます。

パラメータ

- `field`: 比較するフィールド。
- `value`: 比較する値。

例 (MongoDB シェル)

次の例は、 $gt演算子を使用して、 ageフィールドが 30 より大きいすべてのドキュメントを検索する方法を示しています。

サンプルドキュメントを作成する

```
db.users.insertMany([
  { name: "John", age: 25 },
  { name: "Jane", age: 32 },
  { name: "Bob", age: 45 },
  { name: "Alice", age: 28 }
]);
```

クエリの例

```
db.users.find({ age: { $gt: 30 } });
```

出力

```
{ "_id" : ObjectId("6249e5c22a5d39884a0a0001"), "name" : "Jane", "age" : 32 },
{ "_id" : ObjectId("6249e5c22a5d39884a0a0002"), "name" : "Bob", "age" : 45 }
```

コードの例

$gt コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $gte演算子は、指定された値以上の値を一致させるために使用されます。この演算子は、数値比較に基づいてデータをフィルタリングおよびクエリするのに役立ちます。

パラメータ

- `field`: 指定された値と照合するフィールド。
- `value`: フィールドと比較する値。

例 (MongoDB シェル)

次の例は、Amazon DocumentDB で $gte演算子を使用して、「age」フィールドが 25 以上であるすべてのドキュメントを検索する方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.users.find({ age: { $gte: 25 } }, { _id: 0, name: 1, age: 1 });
```

出力

```
{ "name" : "Jane", "age" : 25 }
{ "name" : "Bob", "age" : 30 }
{ "name" : "Alice", "age" : 35 }
```

コードの例

$gte コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $in演算子は、フィールドの値が配列で指定された値のいずれかと等しいドキュメントを検索できる論理クエリ演算子です。

パラメータ

- `field`: 指定された配列に対してチェックするフィールド。
- `[value1, value2, ...]`: 指定されたフィールドに一致する値の配列。

フィールド名のドル ($)

ネストされたオブジェクトの $inの$プレフィックス付きフィールドのクエリに関する制限フィールド名のドル ($) とドット (.)については、「」を参照してください。

例 (MongoDB シェル)

次の例は、 $in演算子を使用して、 colorフィールドが指定された配列の値の 1 つであるドキュメントを検索する方法を示しています。

サンプルドキュメントを作成する

```
db.colors.insertMany([
  { "_id": 1, "color": "red" },
  { "_id": 2, "color": "green" },
  { "_id": 3, "color": "blue" },
  { "_id": 4, "color": "yellow" },
  { "_id": 5, "color": "purple" }
])
```

クエリの例

```
db.colors.find({ "color": { "$in": ["red", "blue", "purple"] } })
```

出力

```
{ "_id": 1, "color": "red" },
{ "_id": 3, "color": "blue" },
{ "_id": 5, "color": "purple" }
```

コードの例

$in コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 4.0 の新機能。

Elastic クラスターではサポートされていません。

Amazon DocumentDB の $jsonSchema演算子は、指定された JSON スキーマに基づいてドキュメントをフィルタリングするために使用されます。この演算子を使用すると、特定の JSON スキーマに一致するドキュメントをクエリし、取得したドキュメントが特定の構造要件とデータ型要件に準拠していることを確認できます。

コレクション作成の一部として$jsonSchema評価クエリ演算子を使用すると、コレクションに挿入されるドキュメントのスキーマを検証できます。詳細については、「JSON スキーマ検証の使用」を参照してください。

パラメータ

- `required` (配列): ドキュメント内の必須フィールドを指定します。
- `properties` (オブジェクト): ドキュメント内の各フィールドのデータ型とその他の制約を定義します。

例 (MongoDB シェル)

次の例は、 $jsonSchema演算子を使用してemployeesコレクションをフィルタリングしname、フィールド、 ageフィールド、employeeIdおよび フィールドを持つドキュメントのみを取得する方法を示しています。 employeeIdフィールドのタイプは ですstring。

サンプルドキュメントを作成する

```
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" }
]);
```

クエリの例

```
db.employees.aggregate([
  { $match: {
    $jsonSchema: {
```

```
      required: ["name", "employeeId", "age"],
      properties: { "employeeId": { "bsonType": "string" } }
    }
  }}
]);
```

出力

```
{ "_id" : ObjectId("6908e8b61f77fc26b2ecd26f"), "name" : { "firstName" : "Emily",
 "lastName" : "Brown" }, "employeeId" : "2", "age" : 25 }
```

コードの例

$jsonSchema コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $lt演算子は、指定された値より小さい値を一致させるために使用されます。この演算子は、数値比較に基づいてデータをフィルタリングおよびクエリするのに役立ちます。

パラメータ

- field: 確認するフィールド名。
- value: 比較する値。

例 (MongoDB シェル)

次の例は、 $lt演算子を使用して、Inventory.OnHand値が 50 未満のドキュメントを取得する方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

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

出力

```
{ "_id" : 1, "Inventory" : { "OnHand" : 25, "Sold" : 60 }, "Colors" : [ "Red",
 "Blue" ] }
{ "_id" : 3, "Inventory" : { "OnHand" : 10, "Sold" : 85 }, "Colors" : [ "Red",
 "Green" ] }
```

コードの例

$lt コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 クエリ演算子は、フィールド値を除数で割ったドキュメントに指定された余りがあるドキュメントを選択します。これは、モジュロ算術条件に基づいてドキュメントをフィルタリングする場合に便利です。

パラメータ

- `divisor`: 除算する数値。
- `remainder`: 予想される余剰値。

例 (MongoDB シェル)

次の例は、 $mod演算子を使用して、数量が奇数であるすべての注文を検索する方法を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.orders.find({ quantity: { $mod: [2, 1] } });
```

出力

```
{ "_id" : 1, "item" : "Widget", "quantity" : 15 }
{ "_id" : 3, "item" : "Tool", "quantity" : 7 }
{ "_id" : 5, "item" : "Part", "quantity" : 9 }
```

このクエリは、数量を 2 で割ったドキュメントの残りが 1 で、すべての奇数を効果的に選択します。

コードの例

$mod クエリ演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $lte演算子は、指定されたフィールドの値が指定された値以下であるドキュメントを照合するために使用されます。この演算子は、数値比較に基づいてデータをフィルタリングおよびクエリするのに役立ちます。

パラメータ

- field: 比較するフィールド。
- value: 比較する値。

例 (MongoDB シェル)

次の例は、 $lte 演算子を使用して、 quantityフィールドが 10 以下のドキュメントを取得する方法を示しています。

サンプルドキュメントを作成する

```
db.inventory.insertMany([
  { item: "canvas", qty: 100 },
```

```
  { item: "paint", qty: 50 },
  { item: "brush", qty: 10 },
  { item: "paper", qty: 5 }
]);
```

クエリの例

```
db.inventory.find({ qty: { $lte: 10 } });
```

出力

```
{ "_id" : ObjectId("..."), "item" : "brush", "qty" : 10 },
{ "_id" : ObjectId("..."), "item" : "paper", "qty" : 5 }
```

コードの例

$lte コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、現在のクエリ実行に関連付けられたメタデータにアクセスするために使用されます。この演算子は主にテキスト検索オペレーションに使用されます。メタデータは、一致したドキュメントの関連性に関する情報を提供します。

パラメータ

- textScore: ドキュメントのテキスト検索スコアを取得します。このスコアは、ドキュメントとテキスト検索クエリの関連性を示します。

例 (MongoDB シェル)

次の例は、 $meta演算子を使用して、テキスト検索クエリに一致するドキュメントのテキスト検索スコアを取得する方法を示しています。

サンプルドキュメントを作成する

```
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." }
]);
```

テキストインデックスを作成する

```
db.documents.createIndex({ content: "text" });
```

クエリの例

```
db.documents.find(
  { $text: { $search: "coffee" } },
  { _id: 0, title: 1, content: 1, score: { $meta: "textScore" } }
).sort({ score: { $meta: "textScore" } });
```

出力

```
[
  {
    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
  }
]
```

コードの例

$meta コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、フィールドの値が指定された値と等しくないドキュメントを一致させるために使用されます。これは、クエリ述語でドキュメントをフィルタリングするために使用できる比較演算子です。

プランナーバージョン 2.0 で のインデックスサポートが追加されました$ne。

パラメータ

- field: 確認するフィールド。
- value: チェックする値。

例 (MongoDB シェル)

この例では、 statusフィールドが と等しくないusersコレクション内のすべてのドキュメントを見つけます"active"。

サンプルドキュメントを作成する

```
db.users.insertMany([
  { name: "John", status: "active" },
  { name: "Jane", status: "inactive" },
  { name: "Bob", status: "suspended" },
  { name: "Alice", status: "active" }
]);
```

クエリの例

```
db.users.find({ status: { $ne: "active" } });
```

出力

```
[
  {
    _id: ObjectId('...'),
    name: 'Jane',
    status: 'inactive'
  },
  {
    _id: ObjectId('...'),
    name: 'Bob',
    status: 'suspended'
  }
]
```

コードの例

$ne コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、指定された配列にない値を一致させるために使用されます。これは、指定された配列の値と一致する $in演算子の逆です。

プランナーバージョン 2.0 で のインデックスサポートが追加されました$nin。

パラメータ

- field: 確認するフィールド。
- array: チェックする値の配列。

フィールド名のドル ($)

ネストされたオブジェクトの $ninの$プレフィックス付きフィールドのクエリに関する制限フィールド名のドル ($) とドット (.)については、「」を参照してください。

例 (MongoDB シェル)

次の例は、 $nin演算子を使用して、 categoryフィールドが「Fiction」または「Mystery」と等しくないドキュメントを検索する方法を示しています。

サンプルドキュメントを作成する

```
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" }
])
```

クエリの例

```
db.books.find({
  category: {
    $nin: ["Fiction", "Mystery"]
  }
})
```

出力

```
[
  {
    _id: ObjectId('...'),
    title: 'The Martian',
    author: 'Andy Weir',
    category: 'Science Fiction'
```

```
  },
  {
    _id: ObjectId('...'),
    title: 'The Alchemist',
    author: 'Paulo Coelho',
    category: 'Philosophy'
  }
]
```

コードの例

$nin コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、指定されたクエリ条件のいずれにも当てはまらないドキュメントを一致させるために使用されます。これは論理的な「NOR」オペレーションに似ています。いずれのオペランドも true でない場合、結果は true になります。

パラメータ

- expression1: 評価する最初の式。
- expression2: 評価する 2 番目の式。
- expressionN: 評価する追加の式。

例 (MongoDB シェル)

次の例は、 qtyフィールドが 20 以上、 sizeフィールドが「XL」と等しくないドキュメントを取得して、 $nor演算子の使用方法を示しています。

サンプルドキュメントを作成する

```
db.items.insertMany([
  { qty: 10, size: "M" },
  { qty: 15, size: "XL" },
  { qty: 25, size: "L" },
  { qty: 30, size: "XL" }
])
```

クエリの例

```
db.items.find({
  $nor: [
    { qty: { $lt: 20 } },
```

```
    { size: "XL" }
  ]
})
```

出力

```
[
  { "_id" : ObjectId("..."), "qty" : 25, "size" : "L" }
]
```

コードの例

$nor コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、特定の式の結果を無効にするために使用されます。これにより、指定された条件が一致しないドキュメントを選択できます。

プランナーバージョン 2.0 では、 $not {eq}と のインデックスサポートが追加されました$not {in}。

パラメータ

- expression: 否定する式。

例 (MongoDB シェル)

次の例は、 $not演算子を使用して、 statusフィールドが「アクティブ」と等しくないドキュメントを検索する方法を示しています。

サンプルドキュメントを作成する

```
db.users.insertMany([
```

```
  { name: "John", status: "active" },
  { name: "Jane", status: "inactive" },
  { name: "Bob", status: "pending" },
  { name: "Alice", status: "active" }
]);
```

クエリの例

```
db.users.find({ status: { $not: { $eq: "active" } } });
```

出力

```
[
  {
    _id: ObjectId('...'),
    name: 'Jane',
    status: 'inactive'
  },
  {
    _id: ObjectId('...'),
    name: 'Bob',
    status: 'pending'
  }
]
```

コードの例

$not コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、2 つ以上の式の配列に対して論理 OR オペレーションを実行するために使用されます。少なくとも 1 つの式に一致するドキュメントが返されます。この演算子は、複数の条件のいずれかを満たすドキュメントをクエリする必要がある場合に便利です。

パラメータ

- `expression1`: 評価する最初の式。
- `expression2`: 評価する 2 番目の式。
- `...`: 評価する追加の式 (オプション) 。

例 (MongoDB シェル)

次の例は、 がモデル「Heavy H1」のTruckForYou」またはモデル「Bolid 1」のSportForYoumake」のいずれかであるドキュメントを検索する$or演算子の使用を示しています。

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.cars.find({
  $or: [
    { make: "TruckForYou", model: "Heavy H1" },
    { make: "SportForYou", model: "Bolid 1" }
  ]
});
```

出力

```
[
  {
    _id: ObjectId('...'),
    make: 'TruckForYou',
    model: 'Heavy H1',
    year: 2020
  },
  {
    _id: ObjectId('...'),
    make: 'SportForYou',
    model: 'Bolid 1',
    year: 2021
  }
]
```

コードの例

$or コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子を使用すると、文字列フィールドで正規表現マッチングを実行できます。これは、複雑なパターンに基づいてドキュメントを検索およびフィルタリングするための強力なツールです。

パラメータ

- regular expression: フィールドに一致する正規表現パターン。
- $options: (オプション) 大文字と小文字の区別、グローバルマッチングなど、検索動作を変更するオプションを提供します。

例 (MongoDB シェル)

次の例は、「name」フィールドが特定のパターンに一致するドキュメントを検索するための $regex演算子の使用を示しています。

サンプルドキュメントを作成する

```
db.users.insertMany([
  { name: "John Doe" },
  { name: "Jane Smith" },
  { name: "Alice Johnson" },
  { name: "Bob Williams" },
  { name: "Charlie Davis" }
]);
```

クエリの例

```
db.users.find({ name: { $regex: /^A/ } })
```

出力

```
[
  { "_id" : ObjectId("..."), "name" : "Alice Johnson" }
]
```

このクエリは、「name」フィールドが「A」で始まるすべてのドキュメントを返します。

コードの例

$regex コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

バージョン 8.0.1 から新規。

Elastic クラスターではサポートされていません。

Amazon DocumentDB の$sampleRate演算子は、指定されたレートに基づいて入力ドキュメントのランダムなサンプルに一致します。find() クエリフィルターまたは集約$matchステージで使用できます。選択は確率的であるため、返されるドキュメントの数は概算であり、実行ごとに異なる場合があります。

パラメータ

- value: 演算子 [0, 1] (など) に直接渡される範囲 (を含む{ $sampleRate: 0.3 }) の数値。各ドキュメントを含める確率を設定します。の値はドキュメントを0返さず、 の値はすべてのドキュメント1を返します。

例 (MongoDB シェル)

次の例では$sampleRate、 を使用して、コレクション内のドキュメントの約 30% を返します。

サンプルドキュメントを作成する

```
db.events.insertMany([
  { _id: 1, type: "click" },
  { _id: 2, type: "view" },
  { _id: 3, type: "click" },
  { _id: 4, type: "view" },
  { _id: 5, type: "purchase" }
]);
```

クエリの例

```
db.events.aggregate([
```

```
  { $match: { $sampleRate: 0.3 } }
]);
```

出力

オペレーションはドキュメントのランダムなサブセットを返します。$sampleRate は確率的であるため、返される特定のドキュメントとその数はクエリが実行されるたびに異なります。

コードの例

$sampleRate コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 射影演算子は、クエリ結果で返される配列要素の数を制限します。これにより、配列全体をロードすることなく、配列フィールドの先頭または末尾から特定の数の要素を取得できます。

パラメータ

- field: 射影する配列フィールド。
- count: 返される要素の数。正の値は最初から要素を返し、最後に負の値を返します。

例 (MongoDB シェル)

次の例は、$slice射影演算子を使用して配列フィールドから最初の 2 つの項目のみを返す方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.inventory.find(
  {},
  { item: 1, tags: { $slice: 2 } }
)
```

出力

```
{ "_id" : 1, "item" : "notebook", "tags" : [ "office", "school" ] }
{ "_id" : 2, "item" : "pen", "tags" : [ "office", "writing" ] }
{ "_id" : 3, "item" : "folder", "tags" : [ "office", "supplies" ] }
```

コードの例

$slice 射影演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 クエリ演算子は、配列フィールドの要素数が正確に指定された数であるドキュメントに一致します。これは、配列の長さに基づいてドキュメントをフィルタリングするのに役立ちます。

パラメータ

- field: 確認する配列フィールド。
- count: 配列に含める必要がある要素の正確な数。

例 (MongoDB シェル)

次の例は、 $size演算子を使用して、厳密に 3 つのタグを持つすべての製品を検索する方法を示しています。

サンプルドキュメントを作成する

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

クエリの例

```
db.products.find({ tags: { $size: 3 } });
```

出力

```
{ "_id" : 1, "name" : "Laptop", "tags" : [ "electronics", "computers", "portable" ] }
{ "_id" : 3, "name" : "Desk", "tags" : [ "furniture", "office", "workspace" ] }
```

コードの例

$size クエリ演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、ドキュメントコレクション内のテキストインデックス付きフィールドに対して全文検索を実行するために使用されます。この演算子を使用すると、特定の単語やフレーズを含むドキュメントを検索し、他のクエリ演算子と組み合わせて、追加の条件に基づいて結果をフィルタリングできます。

パラメータ

- $search: 検索するテキスト文字列。

例 (MongoDB シェル)

次の例は、 $text演算子を使用して「interest」という単語を含むドキュメントを検索し、「star_rating」フィールドに基づいて結果をフィルタリングする方法を示しています。

サンプルドキュメントを作成する

```
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" }
]);
```

テキストインデックスを作成する

```
db.test.createIndex({ comments: "text" });
```

クエリの例

```
db.test.find({$and: [{star_rating: 5}, {$text: {$search: "interest"}}]})
```

出力

```
{ "_id" : 5, "star_rating" : 5, "comments" : "interesting couch" }
{ "_id" : 6, "star_rating" : 5, "comments" : "interested in couch for sale, year
 2022" }
```

上記のコマンドは、任意の形式の「interest」と 5 に等しい「star_rating」を含むテキストインデックス付きフィールドを持つドキュメントを返します。

コードの例

$text コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

$type 演算子は、ドキュメント内のフィールドのデータ型をチェックするために使用されます。タイプ固有のオペレーションまたは検証が必要な場合に使用できます。$type 演算子は、評価された式の BSON タイプを返します。返される型は、フィールドまたは式の型に対応する文字列です。

プランナーバージョン 2.0 で のインデックスサポートが追加されました$type。

パラメータ

- expression : 評価する式。

例 (MongoDB シェル)

サンプルドキュメントを作成する

```
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 }
]);
```

クエリの例

```
db.documents.find({
  $or: [
    { age: { $type: "number" } },
    { email: { $type: "string" } },
    { name: { $type: "string" } }
  ]
})
```

出力

```
[
  { "_id": 1, "name": "John", "age": 30, "email": "john@example.com" },
  { "_id": 2, "name": "Jane", "age": "25", "email": 123456 }
]
```

このクエリは、 ageフィールドが「number」型、 emailフィールドが「string」型、 nameフィールドが「string」型であるドキュメントを返します。

コードの例

$type コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

更新演算子

このセクションでは、Amazon DocumentDB でサポートされている更新演算子に関する詳細情報を提供します。

トピック

- $
- $[]
- $[<identifier>]
- $addToSet
- $bit
- $currentDate
- $each
- $inc
- $max
- $min
- $mul
- $pop
- $position
- $pull
- $pullAll
- $push
- $rename
- $set
- $setOnInsert
- $slice
- $sort
- $unset

$

$ 位置演算子は、クエリ条件に一致する最初の配列要素を更新します。一致した配列要素の位置のプレースホルダーとして機能します。

パラメータ

- `field.$`: 位置演算子を使用して、最初に一致する要素を更新する配列フィールド。

例 (MongoDB シェル)

次の例は、$位置演算子を使用して特定の配列要素を更新する方法を示しています。

サンプルドキュメントを作成する

```
db.inventory.insertMany([
  { _id: 1, item: "Widget", quantities: [10, 20, 30] },
  { _id: 2, item: "Gadget", quantities: [5, 15, 25] }
]);
```

クエリの例

```
db.inventory.updateOne(
  { _id: 1, quantities: 20 },
  { $set: { "quantities.$": 22 } }
);
```

出力

```
{
  "_id" : 1,
  "item" : "Widget",
  "quantities" : [ 10, 22, 30 ]
}
```

コードの例

$ 位置演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

$[]

$[] すべての位置演算子は、配列内のすべての要素を更新します。これは、配列フィールドのすべての要素を変更する必要がある場合に使用されます。

パラメータ

- field.$[]: すべての要素を更新するすべての位置演算子を含む配列フィールド。

例 (MongoDB シェル)

次の例は、 $[]演算子を使用してすべての配列要素を更新する方法を示しています。

サンプルドキュメントを作成する

```
db.products.insertOne({
  _id: 1,
  name: "Laptop",
  prices: [1000, 1100, 1200]
});
```

クエリの例

```
db.products.updateOne(
  { _id: 1 },
  { $inc: { "prices.$[]": 50 } }
);
```

出力

```
{
  "_id" : 1,
  "name" : "Laptop",
  "prices" : [ 1050, 1150, 1250 ]
}
```

コードの例

$[] 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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>] フィルタリングされた位置演算子は、指定されたフィルター条件に一致するすべての配列要素を更新します。これは、配列要素を選択的に更新する arrayFiltersオプションとともに使用されます。

パラメータ

- field.$[identifier]: フィルタリングされた位置演算子を持つ配列フィールド。
- arrayFilters: 更新する要素を決定するフィルター条件の配列。

例 (MongoDB シェル)

次の例は、 $[<identifier>]演算子を使用して、条件に基づいて特定の配列要素を更新する方法を示しています。

サンプルドキュメントを作成する

```
db.students.insertOne({
  _id: 1,
  name: "Alice",
  grades: [
    { subject: "Math", score: 85 },
    { subject: "Science", score: 92 },
    { subject: "History", score: 78 }
  ]
});
```

クエリの例

```
db.students.updateOne(
  { _id: 1 },
  { $inc: { "grades.$[elem].score": 5 } },
  { arrayFilters: [{ "elem.score": { $gte: 80 } }] }
);
```

出力

```
{
  "_id" : 1,
  "name" : "Alice",
  "grades" : [
    { "subject" : "Math", "score" : 90 },
    { "subject" : "Science", "score" : 97 },
    { "subject" : "History", "score" : 78 }
  ]
```

```
}
```

コードの例

$[<identifier>] 演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

Amazon DocumentDB の $addToSet演算子は、値が配列にまだ存在しない場合にのみ、配列に値を追加するために使用されます。これは、配列に一意の要素が含まれていることを確認するのに役立ちます。

パラメータ

- `field`: 更新するフィールド。
- `value`: 配列フィールドに追加する値。これは単一の値でも式でもかまいません。

例 (MongoDB シェル)

次の例は、 $addToSet演算子を使用して配列に一意の要素を追加する方法を示しています。

サンプルドキュメントを作成する

```
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"] }
])
```

クエリの例

```
db.products.update(
  { "item": "apple" },
```

```
  { $addToSet: { "tags": "green" } }
)
```

出力

```
{ "_id": 1, "item": "apple", "tags": ["fruit", "red", "round", "green"] }
```

この例では、$addToSet演算子は「item」フィールドが「apple」であるドキュメントの「tags」配列に「green」タグを追加します。「green」は配列にまだ存在していなかったため、追加されました。

コードの例

$addToSet コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $bit演算子を使用すると、特定のフィールドのビットに対してビット単位のオペレーションを実行できます。これは、数値内の個々のビットの状態の設定、クリア、チェックなどのタスクに役立ちます。

パラメータ

- field: ビット単位のオペレーションを実行するフィールド。
- and: フィールドでビット単位の AND オペレーションを実行するために使用される整数値。
- or: フィールドでビット単位の OR オペレーションを実行するために使用される整数値。
- xor: フィールドでビット単位の XOR オペレーションを実行するために使用される整数値。

例 (MongoDB シェル)

次の例は、 $bit演算子を使用して数値フィールドに対してビット単位のオペレーションを実行する方法を示しています。

サンプルドキュメントを作成する

```
db.numbers.insert([
  { "_id": 1, "number": 5 },
  { "_id": 2, "number": 12 }
])
```

クエリの例

```
db.numbers.update(
  { "_id": 1 },
  { "$bit": { "number": { "and": 3 } } }
)
```

出力

```
{
  "_id": 1,
  "number": 1
}
```

この例では、 $bit演算子を使用して、 が 1 のドキュメントの「数値」フィールドに対してビット単位_idの AND オペレーションを実行します。その結果、「数値」フィールドの値は 1 に設定されます。これは、元の値 5 と値 3 の間のビット単位の AND 演算の結果です。

コードの例

$bit コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、フィールドの値を現在の日付と時刻に設定するために使用されます。この演算子は、ドキュメントが挿入または更新されたときに、現在のタイムスタンプでフィールドを自動的に更新するのに役立ちます。

パラメータ

- field: 現在の日付と時刻で更新するフィールド。
- type: (オプション) 現在の日付に使用する BSON タイプを指定します。date または timestamp のいずれかになります。

例 (MongoDB シェル)

次の例は、 $currentDate演算子を使用して、 lastModifiedフィールドを新しいドキュメントが挿入された現在の日時に設定する方法を示しています。

サンプルドキュメントを作成する

```
db.users.insert({
  name: "John Doe",
  email: "john.doe@example.com"
})
```

クエリの例

```
db.users.updateOne(
  { name: "John Doe" },
  { $currentDate: { lastModified: true } }
)
```

更新されたドキュメントを表示する

```
db.users.findOne({ name: "John Doe" })
```

出力

```
{
  _id: ObjectId('...'),
  name: 'John Doe',
  email: 'john.doe@example.com',
  lastModified: ISODate('2025-10-25T22:50:29.963Z')
}
```

コードの例

$currentDate コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、 $pushや などの他の更新演算子と組み合わせて使用され$addToSet、配列フィールドに複数の値を追加します。これにより、複数の更新オペレーションを実行するのではなく、1 回のオペレーションで配列に複数の要素を追加できます。

パラメータ

- value: 配列フィールドに追加する値の配列。

例 (MongoDB シェル)

次の例は、 $each演算子と $push演算子を使用して、配列フィールドに複数の要素を追加する方法を示しています。

サンプルドキュメントを作成する

```
db.fruits.insertOne({
  _id: 1,
  fruits: ["apple", "banana"]
})
```

クエリの例

```
db.fruits.updateOne(
  { _id: 1 },
  { $push: { fruits: { $each: ["cherry", "durian", "elderberry"] } } }
)
```

更新されたドキュメントを表示する

```
db.fruits.findOne({ _id: 1 })
```

出力

```
{
  _id: 1,
  fruits: [ 'apple', 'banana', 'cherry', 'durian', 'elderberry' ]
}
```

コードの例

$each コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、フィールドの値を指定された量だけ増分するために使用されます。これは、現在の値を取得して新しい値を計算し、フィールドを更新することなく、カウンターやレーティングなどの数値フィールドを更新するために使用されます。

パラメータ

- field: 増分するフィールドの名前。
- amount: フィールドをインクリメントする量。これは正の値でも負の値でもかまいません。

例 (MongoDB シェル)

次の例は、 $inc演算子を使用してドキュメントの ageフィールドをインクリメントする方法を示しています。

サンプルドキュメントを作成する

```
db.users.insertOne({_id: 123, name: "John Doe", age: 30})
```

クエリの例

```
db.users.updateOne({_id: 123}, {$inc: {age: 1}})
```

更新されたドキュメントを表示する

```
db.users.findOne({_id: 123})
```

出力

```
{ "_id" : 123, "name" : "John Doe", "age" : 31 }
```

コードの例

$inc コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

$max 更新演算子は、指定された値が現在のフィールド値より大きい場合にのみ、フィールドの値を更新します。この演算子は、更新全体で最大値を維持するのに役立ちます。

パラメータ

- field: 更新するフィールド。
- value: 現在のフィールド値と比較する値。

例 (MongoDB シェル)

次の例は、 $max演算子を使用してプレイヤーの記録された最高スコアを更新する方法を示しています。

サンプルドキュメントを作成する

```
db.scores.insertMany([
  { _id: 1, player: "Alice", highScore: 85 },
  { _id: 2, player: "Bob", highScore: 92 },
  { _id: 3, player: "Charlie", highScore: 78 }
])
```

更新の例

```
db.scores.updateOne(
  { _id: 1 },
  { $max: { highScore: 95 } }
)
```

結果

Alice の highScoreフィールドは、95 が現在の値である 85 より大きいため、95 に更新されます。

```
{ "_id": 1, "player": "Alice", "highScore": 95 }
```

コードの例

$max コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

$min 更新演算子は、指定された値が現在のフィールド値より小さい場合にのみ、フィールドの値を更新します。この演算子は、更新全体で最小値を維持するのに役立ちます。

パラメータ

- field: 更新するフィールド。
- value: 現在のフィールド値と比較する値。

例 (MongoDB シェル)

次の例は、 $min演算子を使用して気象観測所の記録された最低温度を更新する方法を示しています。

サンプルドキュメントを作成する

```
db.weather.insertMany([
  { _id: 1, station: "Station A", lowestTemp: 15 },
  { _id: 2, station: "Station B", lowestTemp: 20 },
  { _id: 3, station: "Station C", lowestTemp: 18 }
])
```

更新の例

```
db.weather.updateOne(
  { _id: 1 },
  { $min: { lowestTemp: 12 } }
)
```

結果

ステーション A の lowestTempフィールドは、12 が現在の値である 15 より小さいため、12 に更新されます。

```
{ "_id": 1, "station": "Station A", "lowestTemp": 12 }
```

コードの例

$min コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $mul演算子は、フィールドの値を指定された数で乗算するために使用されます。これは、クレジットカードのステータスに基づいてフライトマイルを更新するなど、複数のドキュメントをアトミックかつ一貫して更新する場合に便利です。

パラメータ

- field: 乗算するフィールド。
- multiplier: フィールド値に乗算する数値。

例 (MongoDB シェル)

この例では、 $mul演算子を使用して、 credit_cardフィールドが であるすべてのドキュメントのflight_miles値を 2 倍にする方法を示しますtrue。

サンプルドキュメントを作成する

```
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] }
]);
```

クエリの例

```
db.miles.update(
  { "credit_card": { "$eq": true } },
  { "$mul": { "flight_miles.$[]": NumberInt(2) } },
  { "multi": true }
```

```
);
```

出力

```
{ "_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 ] }
```

クレジットカードをお持ちのお客様には、フライトマイルが 2 倍になりました。

$[] 位置配列演算子は、flight_miles配列内の各要素に$mulオペレーションを適用するために使用されます。

コードの例

$mul コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $pop演算子は、配列フィールドから最初または最後の要素を削除するために使用されます。これは、固定サイズの配列を維持したり、ドキュメント内にキューのようなデータ構造を実装する必要がある場合に特に便利です。

パラメータ

- `field`: 要素を削除する配列フィールドの名前。
- `value`: 削除する要素の位置を決定する整数値。の値は最後の要素1を削除し、 の値は最初の要素-1を削除します。

例 (MongoDB シェル)

この例では、 $pop演算子を使用して配列フィールドから最初と最後の要素を削除する方法を示します。

サンプルドキュメントを作成する

```
db.users.insertMany([
  { "_id": 1, "name": "John Doe", "hobbies": ["reading", "swimming", "hiking"] },
  { "_id": 2, "name": "Jane Smith", "hobbies": ["cooking", "gardening", "painting"] }
])
```

クエリの例

```
// 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 } })
```

出力

```
{ "_id" : 1, "name" : "John Doe", "hobbies" : [ "swimming", "hiking" ] }
{ "_id" : 2, "name" : "Jane Smith", "hobbies" : [ "cooking", "gardening" ] }
```

コードの例

$pop コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の$position修飾子は、$push演算子が要素を挿入する配列内の場所を指定します。$position 修飾子がない場合、$push演算子は配列の末尾に要素を挿入します。

パラメータ

- field: 更新する配列フィールド。
- num: ゼロベースのインデックス作成に基づいて要素を挿入する配列内の位置。

注: $position修飾子を使用するには、$each修飾子とともに表示する必要があります。

例 (MongoDB シェル)

次の例は、 $position演算子を使用してプロジェクト管理システム内の特定の位置にタスクを挿入する方法を示しています。

サンプルドキュメントを作成する

```
db.projects.insertOne({ "_id": 1, "name": "Website Redesign", "tasks": ["Design
 mockups"] })
```

クエリ例 1 - 緊急タスクを最初に追加する

```
db.projects.updateOne(
   { _id: 1 },
   {
     $push: {
        tasks: {
           $each: ["Security audit", "Performance review"],
           $position: 0
        }
     }
   }
)
```

出力 1

```
{ "_id": 1, "name": "Website Redesign", "tasks": ["Security audit", "Performance
 review", "Design mockups"] }
```

クエリの例 2 - 特定の位置にタスクを追加する

```
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
```

```
        }
      }
    }
)
```

出力 2

```
{ "_id": 2, "name": "Mobile App", "tasks": ["Setup project", "Create wireframes", "Code
 review", "Testing phase", "Deploy to store"] }
```

コードの例

$position コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 演算子は、指定された条件に一致する値または値のすべてのインスタンスを配列から削除するために使用されます。この演算子は、ドキュメント内の配列フィールドから特定の要素を削除する必要がある場合に便利です。

パラメータ

- field: 値を削除する配列フィールドの名前 (複数可) 。

- value: 配列から削除する要素 (複数可) を決定する値または条件。

例 (MongoDB シェル)

次の例は、 $pull演算子を使用して配列フィールドから要素を削除する方法を示しています。

サンプルドキュメントを作成する

```
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"]
  }
])
```

クエリの例

```
db.restaurants.updateMany(
  { cuisine: "Italian" },
  { $pull: { features: "Takeout" } }
)
```

出力

```
{
  "acknowledged" : true,
  "matchedCount" : 1,
  "modifiedCount" : 1
}
```

上記のクエリは、 cuisineフィールドが「イタリア語」であるすべてのドキュメントから「テイクアウト」機能を削除します。

コードの例

$pull コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $pullAll演算子は、指定された値のすべてのインスタンスを配列フィールドから削除するために使用されます。これは、1 回のオペレーションで配列から複数の要素を削除する必要がある場合に特に便利です。

パラメータ

- `field`: 要素を削除する配列フィールドの名前。
- `value`: 配列フィールドから削除する値の配列。

例 (MongoDB シェル)

次の例は、 $pullAll演算子を使用して配列フィールドから複数の要素を削除する方法を示しています。

サンプルドキュメントを作成する

```
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"]
  }
])
```

クエリの例

```
db.restaurants.update(
  { "name": "Taj Mahal" },
  { $pullAll: { "features": ["Private Dining", "Live Music"] } }
)
```

出力

```
{
  "name": "Taj Mahal",
  "cuisine": "Indian",
  "features": []
}
```

コードの例

$pullAll コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $push演算子は、ドキュメントの配列フィールドに項目を追加するために使用されます。この演算子は、配列全体を上書きせずに既存の配列に新しいデータを追加する必要がある場合に特に便利です。

パラメータ

- `field`: 新しい要素を追加する配列フィールドの名前。
- `value`: 配列に追加する値。

- position: (オプション) 新しい要素を追加する配列内の位置を指定する修飾子。サポートされている修飾子には、 $ (配列の末尾に を追加) と $[] (配列の末尾に を追加し、配列フィルターを無視) が含まれます。

例 (MongoDB シェル)

次の例は、 $push演算子を使用してドキュメントの配列フィールドに新しい要素を追加する方法を示しています。

サンプルドキュメントを作成する

```
db.users.insert([
  { _id: 1, name: "John Doe", hobbies: ["reading", "swimming"] },
  { _id: 2, name: "Jane Smith", hobbies: ["gardening", "cooking"] }
])
```

クエリの例

```
db.users.updateOne(
  { _id: 1 },
  { $push: { hobbies: "hiking" } }
)
```

出力

```
{
  "acknowledged" : true,
  "matchedCount" : 1,
  "modifiedCount" : 1
}
```

更新を実行すると、 を持つドキュメントのhobbies配列_id: 1が に更新されます["reading", "swimming", "hiking"]。

コードの例

$push コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $rename演算子は、ドキュメント内のフィールドの名前を変更するために使用されます。この演算子は、ドキュメントの構造を更新したり、新しいデータモデルに合わせる必要がある場合に特に役立ちます。

パラメータ

- field: 名前を変更するフィールド。
- newName: フィールドの新しい名前。

例 (MongoDB シェル)

次の例は、 $rename演算子を使用して、 "Date.DoW" フィールドを に設定してドキュメント"Date.DayOfWeek"内の "DocName"フィールドの名前を に変更する方法を示しています"Document 1"。

サンプルドキュメントを作成する

```
db.example.insertOne({
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DoW": "Saturday"
    },
    "Words": 2482
})
```

クエリの例

```
db.example.update(
    { "DocName": "Document 1" },
    { $rename: { "Date.DoW": "Date.DayOfWeek" } }
)
```

出力

```
{
```

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

コードの例

$rename コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $set演算子は、ドキュメント内の指定されたフィールドの値を更新するために使用されます。この演算子を使用すると、新しいフィールドを追加したり、ドキュメント内の既存のフィールドを変更したりできます。これは、Amazon DocumentDB と互換性のある MongoDB Java ドライバーの基本的な更新演算子です。

パラメータ

- field: 更新するフィールド。
- value: フィールドの新しい値。

例 (MongoDB シェル)

次の例は、 $set演算子を使用してドキュメントの Itemフィールドを更新する方法を示しています。

サンプルドキュメントを作成する

```
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
    }
  }
])
```

クエリの例

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

出力

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

コードの例

$set コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

Amazon DocumentDB の $setOnInsert演算子は、ドキュメントが挿入されている場合はフィールドの値を設定するために使用されますが、ドキュメントが更新されている場合は効果がありません。

パラメータ

- field: 設定するフィールド。
- value: フィールドに割り当てる値。

例 (MongoDB シェル)

次の例は、Amazon DocumentDB での $setOnInsert演算子の使用を示しています。ドキュメントがまだ存在しない場合は新しいドキュメントが作成されますが、ドキュメントが更新されている場合は効果がありません。

サンプルドキュメントを作成する

```
db.users.insertOne({
  _id: 1,
  name: "John Doe",
  age: 30
})
```

クエリ例 1 - 既存のドキュメントを更新する

```
db.users.update(
  { _id: 1 },
  {
    $set: { age: 31 },
    $setOnInsert: { createdAt: new Date() }
  },
  { upsert: true }
)
```

出力 1

```
{
  _id: 1,
  name: "John Doe",
  age: 31
```

```
}
```

出力は、ドキュメントが更新されたが、ドキュメントがすでに存在しているため、 createdAt フィールドは追加されていないことを示しています。$setOnInsert 演算子は、新しいドキュメントを挿入する場合にのみ適用されます。

クエリ例 2 - 新しいドキュメントを挿入する (アップサート)

```
db.users.update(
  { _id: 2 },
  {
    $set: { name: "Jane Smith", age: 25 },
    $setOnInsert: { createdAt: new Date() }
  },
  { upsert: true }
)
```

出力 2

```
{
  _id: 2,
  name: "Jane Smith",
  age: 25,
  createdAt: ISODate("2025-10-31T09:57:52.459Z")
}
}
```

コードの例

$setOnInsert コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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

$slice 更新演算子は、配列のサイズを制限して配列を変更します。$push 演算子とともに使用すると、配列内の要素の数が制限され、指定された数の最新の要素または最も古い要素のみが保持されます。

パラメータ

- field: 変更する配列フィールド。
- count: 保持する要素の最大数。正の値は最初の N 要素を保持し、負の値は最後の N 要素を保持します。

例 (MongoDB シェル)

次の例は、 で$slice更新演算子を使用して$push、最近のスコアの固定サイズの配列を維持する方法を示しています。

サンプルドキュメントを作成する

```
db.students.insertOne({
  _id: 1,
  name: "Alice",
  scores: [85, 90, 78]
});
```

クエリの例

```
db.students.updateOne(
  { _id: 1 },
  {
    $push: {
      scores: {
        $each: [92, 88],
        $slice: -3
      }
    }
  }
)
```

出力

```
{
  "_id" : 1,
  "name" : "Alice",
  "scores" : [ 78, 92, 88 ]
}
```

この例では、修飾子は、配列に新しい値をプッシュした後、最後の $slice: -3 3 つの要素のみを保持します。

コードの例

$slice 更新演算子を使用するコード例を表示するには、使用する言語のタブを選択します。

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

$sort 更新修飾子は、 $push演算子で使用すると配列要素を順序付けします。指定されたフィールド値または要素自体に基づいて、配列要素を昇順または降順に配置します。

パラメータ

- field: 変更する配列フィールド。
- order: 昇順-1または降順1に使用します。

例 (MongoDB シェル)

次の例は、 $sort で 修飾子を使用して新しいクイズスコアを追加し$push、降順でソートされたままにする方法を示しています。

サンプルドキュメントを作成する

```
db.students.insertOne({
  _id: 1,
  name: "Bob",
  quizzes: [
    { score: 85, date: "2024-01-15" },
    { score: 92, date: "2024-02-10" }
  ]
});
```

クエリの例

```
db.students.updateOne(
  { _id: 1 },
  {
    $push: {
      quizzes: {
        $each: [{ score: 78, date: "2024-03-05" }],
        $sort: { score: -1 }
      }
    }
  }
)
```

出力

```
{
  "_id" : 1,
  "name" : "Bob",
  "quizzes" : [
    { "score" : 92, "date" : "2024-02-10" },
    { "score" : 85, "date" : "2024-01-15" },
    { "score" : 78, "date" : "2024-03-05" }
  ]
}
```

コードの例

$sort 更新修飾子を使用するコード例を表示するには、使用する言語のタブを選択します。

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 の $unset演算子は、ドキュメントから指定されたフィールドを削除するために使用されます。を使用してフィールドを削除すると$unset、そのフィールドはドキュメントから削除され、それに応じてドキュメントのサイズが小さくなります。これは、ドキュメントから不要なデータを削除する場合に便利です。

パラメータ

- field: ドキュメントから削除するフィールド。これは、単一のフィールドでも、ネストされたフィールドへの点線のパスでもかまいません。

例 (MongoDB シェル)

次の例は、 $unset演算子を使用してexampleコレクション内のドキュメントから Wordsフィールドを削除する方法を示しています。

サンプルドキュメントを作成する

```
db.example.insert({
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DoW": "Saturday"
    },
    "Words": 2482
})
```

クエリの例

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

出力

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

この例では、 $unset演算子を使用して、「ドキュメント 1」とDocName等しい Words フィールドをドキュメントから削除します。結果のドキュメントに Wordsフィールドが含まれなくなりました。

コードの例

$unset コマンドを使用するコード例を表示するには、使用する言語のタブを選択します。

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()
```

Amazon DocumentDB クラスターとインスタンスの管理

以下のセクションでは、Amazon DocumentDB (MongoDB 互換) の実装を管理するためのさまざまなコンポーネントと関連タスクについて説明します。

トピック

- Amazon DocumentDB 運用タスクの概要
- Amazon DocumentDB グローバルクラスターの概要
- Amazon DocumentDB クラスターの管理
- Amazon DocumentDB インスタンスの管理
- Amazon DocumentDB サブネットグループの管理
- Amazon DocumentDB の高可用性とレプリケーション
- Amazon DocumentDB イベントの管理
- Amazon DocumentDB クラスターパラメータグループの管理
- Amazon DocumentDB エンドポイントについて
- Amazon DocumentDB Amazon リソースネーム (ARN) の理解
- Amazon DocumentDB リソースへのタグ付け
- Amazon DocumentDB のメンテナンス
- サービスにリンクされたロールを理解する

Amazon DocumentDB 運用タスクの概要

このセクションでは、Amazon DocumentDB クラスターでの運用タスクと、 AWS CLIを使用してこれらのタスクを完了する方法について説明します。

トピック

- Amazon DocumentDB クラスターにレプリカを追加する
- クラスターとインスタンスの説明
- クラスタースナップショットの作成
- スナップショットからの復元
- クラスターからインスタンスを削除する
- クラスターの削除

Amazon DocumentDB クラスターにレプリカを追加する

Amazon DocumentDB クラスターのプライマリインスタンスを作成した後、1 つまたは複数の レプリカ を追加することができます。レプリカは、2 つの目的に対応する読み取り専用インスタンスです。

- スケーラビリティ - 同時アクセスを必要とするクライアントが多数ある場合は、レプリカを追加して読み取りをスケーリングできます。
- 高可用性 - プライマリインスタンスに障害が発生した場合、Amazon DocumentDB は自動的にレプリカインスタンスにフェイルオーバーし、それを新しいプライマリとして指定します。レプリカに障害が発生しても、障害が発生したノードが回復するまで、クラスター内の他のインスタンスがリクエストを処理することができます。

各 Amazon DocumentDB クラスターは、最大 15 個のレプリカをサポートできます。

> **Note**
>
> 耐障害性を最大にするため、レプリカを異なるアベイラビリティーゾーンにデプロイする必要があります。これにより、アベイラビリティーゾーン全体が使用できなくなった場合でも Amazon DocumentDB クラスターが機能し続けることができます。

次の AWS CLI 例は、新しいレプリカを追加する方法を示しています。`--availability-zone` パラメータは、指定されたアベイラビリティーゾーンにレプリカを配置します。

```
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
```

クラスターとインスタンスの説明

次の の AWS CLI 例では、リージョン内のすべての Amazon DocumentDB クラスターを一覧表示します。クラスターやインスタンスのライフサイクル管理などの特定の管理機能において、Amazon DocumentDB は Amazon RDS と共有される運用テクノロジーを使用しま

す。filterName=engine,Values=docdb フィルターパラメータは Amazon DocumentDB クラスターのみを返します。

クラスターの説明と変更の詳細については、「Amazon DocumentDB クラスターのライフサイクル」を参照してください。

```
aws docdb describe-db-clusters --filter Name=engine,Values=docdb
```

このオペレーションによる出力は、次のようになります。

```
{
    "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",
            ...
        }
    ]
}
```

次の の AWS CLI 例では、Amazon DocumentDB クラスター内のインスタンスを一覧表示します。クラスターの説明と変更の詳細については、「Amazon DocumentDB インスタンスのライフサイクル」を参照してください。

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

出力は次のようになります。この出力には、2 つのインスタンスがあります。プライマリインスタンスは sample-instance-1 ("IsClusterWriter": true) です。レプリカインスタンス sample-instance2 ("IsClusterWriter: false") もあります。

```
[
    [
        [
            {
                "DBInstanceIdentifier": "sample-instance-1",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            },
            {
                "DBInstanceIdentifier": "sample-cluster-2",
                "IsClusterWriter": false,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            }
        ]
    ]
]
```

クラススナップショットの作成

クラススナップショット は、Amazon DocumentDB クラスター内のデータの完全なバックアップです。スナップショットが作成されるときに、Amazon DocumentDB はクラスターボリュームから直接データを読み取ります。このため、その時点で実行中のインスタンスがクラスターにない場合でも、スナップショットを作成できます。スナップショットを作成するためにかかる時間は、クラスターボリュームサイズによって異なります。

Amazon DocumentDB は自動バックアップをサポートしており、バックアップウィンドウ (日中の 30 分間) に毎日実行されます。次の AWS CLI 例は、クラスターのバックアップウィンドウを表示する方法を示しています。

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

出力には、バックアップウィンドウ (UTC) が表示されています。

```
[
    "00:18-00:48"
]
```

Amazon DocumentDB クラスターを作成するときに、バックアップウィンドウを定義することができます。次の例に示すように、バックアップウィンドウを変更することもできます。バックアップウィンドウを定義しない場合、Amazon DocumentDB は自動的にクラスターに 1 つを割り当てます。

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --preferred-backup-window "02:00-02:30"
```

自動バックアップに加えて、いつでもクラススナップショットを手動で作成できます。この操作を行うときは、後で復元できるように、バックアップするクラスターと、スナップショットの一意の名前を指定します。

次の AWS CLI 例は、データのスナップショットを作成する方法を示しています。

```
aws docdb create-db-cluster-snapshot \
```

```
    --db-cluster-identifier sample-cluster \
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

スナップショットからの復元

クラスタースナップショットを新しい Amazon DocumentDB クラスターに復元することができます。これを行うには、スナップショットの名前と新しいクラスターの名前を指定します。スナップショットから既存のクラスターに復元することはできません。代わりに、Amazon DocumentDB は復元時に新しいクラスターを作成してスナップショットデータを入力します。

次の例は、クラスター `sample-cluster` のすべてのスナップショットを示しています。

```
aws docdb describe-db-cluster-snapshots \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusterSnapshots[*].[DBClusterSnapshotIdentifier,SnapshotType,Status]'
```

出力は次のようになります。手動スナップショットは手動で作成したものであり、自動スナップショットは、クラスターのバックアップウィンドウ内で Amazon DocumentDB によって作成されたものです。

```
[
        "sample-cluster-snapshot",
        "manual",
        "available"
    ],
    [
        "rds:sample-cluster",
        "automated",
        "available"
    ]
]
```

次の例は、スナップショットから Amazon DocumentDB クラスターを復元する方法を示しています。

```
aws docdb restore-db-cluster-from-snapshot \
    --engine docdb \
    --db-cluster-identifier new-sample-cluster \
    --snapshot-identifier sample-cluster-snapshot
```

新しいクラスターには関連付けられたインスタンスがありません。このクラスターを操作するには、インスタンスを追加する必要があります。

```
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
```

次の AWS CLI オペレーションを使用して、クラスターとインスタンスの作成の進行状況をモニタリングできます。クラスターとインスタンスのステータスが利用可能になったら、新しいクラスターのエンドポイントに接続し、データにアクセスできます。

```
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]'
```

クラスターからインスタンスを削除する

Amazon DocumentDB はクラスターボリューム内のすべてのデータを保存します。クラスターからすべてのインスタンスを削除しても、そのクラスターボリュームのデータは保持されます。データに再アクセスする必要がある場合は、いつでもクラスターにインスタンスを追加し、中断した個所から再開できます。

次の例は、Amazon DocumentDB クラスターからインスタンスを削除する方法を示しています。

```
aws docdb delete-db-instance \
    --db-instance-identifier sample-instance
```

クラスターの削除

Amazon DocumentDB クラスターを削除する前に、最初にそのすべてのインスタンスを削除する必要があります。次の の AWS CLI 例では、クラスター内のインスタンスに関する情報を返します。このオペレーションでインスタンス識別子が返される場合は、各インスタンスを削除する必要があります。詳細については、「クラスターからインスタンスを削除する」を参照してください。

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterMembers[*].DBInstanceIdentifier'
```

その他のインスタンスが残っていない場合は、クラスターを削除できます。そのときに、次のいずれかのオプションを選択する必要があります。

- 最終スナップショットの作成 - すべてのクラスターデータをスナップショットで取得し、後でそのデータで新しいインスタンスを再作成できるようにします。次の例は、その方法を示しています。

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --final-db-snapshot-identifier sample-cluster-snapshot
```

- 最終スナップショットをスキップする - すべてのクラスターデータを永久に破棄します。この操作は取り消すことができません。次の例は、その方法を示しています。

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --skip-final-snapshot
```

Amazon DocumentDB グローバルクラスターの概要

グローバルクラスターとは

グローバルクラスターは、1 つのプライマリリージョンと最大 10 の読み取り専用セカンダリリージョンで構成されます。書き込みオペレーションをプライマリリージョンのプライマリクラスターに直接発行すると、Amazon DocumentDB は専用インフラストラクチャを使用してデータをセカンダリリージョンに自動的にレプリケートします。レイテンシーは通常 1 秒未満です。

グローバルクラスターはどのように役立つのか

- リージョン全体の停止からの復旧 — リージョン全体の停止が発生した場合、セカンダリクラスターの 1 つを数分以内にプライマリクラスターに昇格できます。一般的な目標復旧時間 (RTO) は 1 分未満です。通常、目標復旧時点 (RPO) は秒単位で測定されますが、これは障害発生時のネットワーク経由の遅延によって異なります。
- ローカルレイテンシーによるグローバル読み取り — 世界中にオフィスがある場合は、グローバルクラスターを使用して、主要な情報源をプライマリリージョンで更新し続けることができます。他

のリージョンにあるオフィスは、自社のリージョンにある情報にローカルのレイテンシーでアクセスすることができます。

- スケーラブルなセカンダリクラスター — セカンダリリージョンに読み取り専用インスタンスを追加することで、セカンダリクラスターをスケールできます。セカンダリクラスターは読み取り専用です。したがって、読み取り専用のレプリカインスタンスを、1 つの クラスターにつき、通常の 15 件ではなく最大 16 件サポートします。
- プライマリからセカンダリクラスターへの迅速なレプリケーション - グローバルクラスターによるレプリケーションは、プライマリデータベースクラスターのパフォーマンスにほとんど影響しません。DB インスタンスのリソースは、全面的にアプリケーションの読み取りおよび書き込みワークロードに当てられます。
- 変更ストリーム — グローバルクラスターのセカンダリクラスターから変更ストリームを読み取ることができ、プライマリクラスターの負荷を軽減し、セカンダリリージョンから読み取りワークロードをスケールできます。

グローバルクラスターの現在の制限は何ですか。

- グローバルクラスターは Amazon DocumentDB v3.6 ではサポートされていません。
- グローバルクラスターは、db.t3、db.t4g、db.r4 を除くすべてのインスタンスタイプでサポートされています。
- グローバルクラスターは、南米 (サンパウロ)、欧州 (ミラノ)、中国 (北京)、中国 (寧夏) の各リージョンでは利用できません。
- リージョンが異なるエンジンバージョンを使用している場合、スイッチオーバーとグローバルフェイルオーバーはサポートされていません。エンジンのバージョンが一致しない場合、手動フェイルオーバーがサポートされます。
- 書き込みオペレーションはプライマリクラスターのみが実行します。書き込みオペレーションを実行するクライアントは、プライマリクラスターの クラスターエンドポイントに接続します。
- クラスターには、最大 10 個のセカンダリリージョンと 1 つのプライマリリージョンを含めることができます。
- セカンダリクラスターは停止できません。プライマリクラスターにセカンダリクラスターが関連付けられている場合、プライマリクラスターを停止できません。セカンダリクラスターを持たないリージョナルクラスターのみを停止できます。
- セカンダリクラスターにアタッチされたレプリカは、特定の場合に再起動することが可能です。プライマリリージョンのインスタンスが再起動またはフェイルオーバーした場合、セカンダリリージョンのレプリカも再起動します。このクラスターは、その後すべてのレプリカがプライマリデー

タベースのクラスターのライターインスタンスと同期するまでは使用できません。この動作は想定されるものです。プライマリクラスターに変更を加えるときは、必ず事前に、グローバルクラスターへの影響を把握してください。

トピック

- クイックスタートガイド: グローバルクラスター
- Amazon DocumentDB グローバルクラスターの管理
- Amazon DocumentDB グローバルクラスターへの接続
- Amazon DocumentDB グローバルクラスターのモニタリング
- ディザスタリカバリと Amazon DocumentDB グローバルクラスターについて

クイックスタートガイド: グローバルクラスター

トピック

- 設定
- Amazon DocumentDB グローバルクラスターの作成
- Amazon DocumentDB グローバルクラスター AWS リージョン への の追加
- Amazon DocumentDB グローバルクラスターのスナップショットの使用

設定

Amazon DocumentDB グローバルクラスターは、少なくとも 2 つにまたがります AWS リージョン。プライマリリージョンは、1 つのプライマリ (ライター) インスタンスと最大 15 個のレプリカインスタンスを持つクラスターをサポートします。セカンダリリージョンでは、最大 16 個のレプリカインスタンスで構成される読み取り専用クラスターが実行されます。グローバルクラスターには、最大 10 個のセカンダリリージョンを含めることができます。この表は、グローバルクラスターで許容されているクラスター、インスタンス、レプリカの最大数をリスト化したものです。

説明	プライマリ AWS リージョン	セカンダリ AWS リージョン
クラスター	1	10 (最大)
ライターインスタンス	1	0

説明	プライマリ AWS リージョン	セカンダリ AWS リージョン
読み取り専用インスタンス (Amazon DocumentDB レプリカ)、クラスターごと	15 (最大数)	16 (総数)
読み取り専用インスタンス (最大許容数、指定されたセカンダリリージョンの実数)	15 - s	s = セカンダリの総数 AWS リージョン

クラスターには、以下の固有の要件があります。

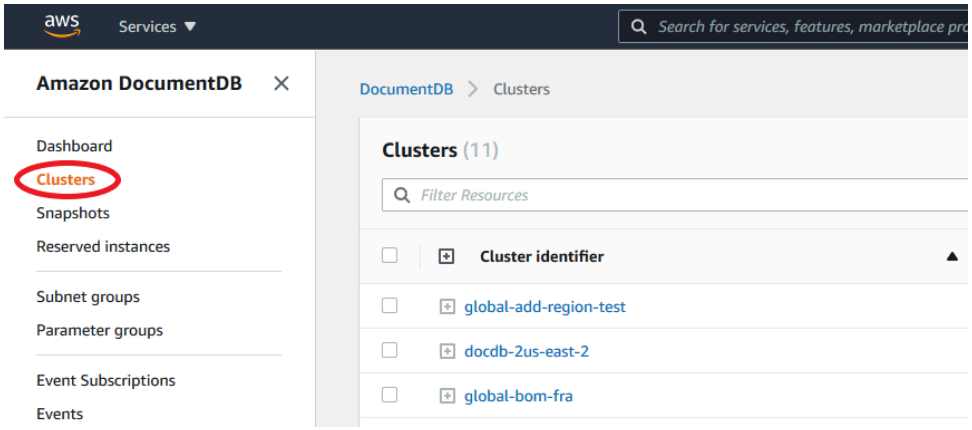
- データベースインスタンスクラスの要件 — 使用できるのは `db.r5` と`db.r6g` インスタンスクラスのみです。
- AWS リージョン 要件 — プライマリクラスターは 1 つのリージョンにあり、少なくとも 1 つのセカンダリクラスターは同じアカウントの別のリージョンにある必要があります。最大 10 個のセカンダリ (読み取り専用) クラスターを作成できます。それぞれが異なるリージョンに存在する必要があります。つまり、2 つのクラスターを、同じリージョン内に置くことはできません。
- 命名の要件 - それぞれのクラスターで選ぶ名前は、すべてのリージョンにおいて一意でなければなりません。異なるクラスターに、たとえそれらが別のリージョンに置かれていたとしても、同じ名前を付けることはできません。

Amazon DocumentDB グローバルクラスターの作成

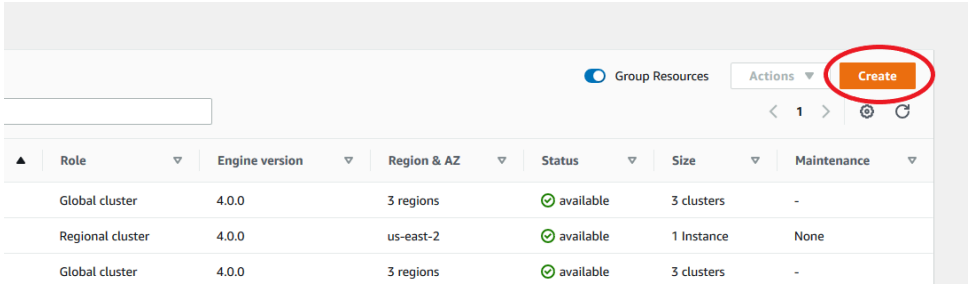
最初のグローバルクラスターを構築する準備はできていますか。このセクションでは、 AWS マネジメントコンソール または を使用して、次の手順 AWS CLI で新しいデータベースクラスターとインスタンスを使用して新しいグローバルクラスターを作成する方法について説明します。

の使用 AWS マネジメントコンソール

1. で AWS マネジメントコンソール、Amazon DocumentDB に移動します。
2. Amazon DocumentDB コンソールにアクセスしたら、[クラスター] を選択します。

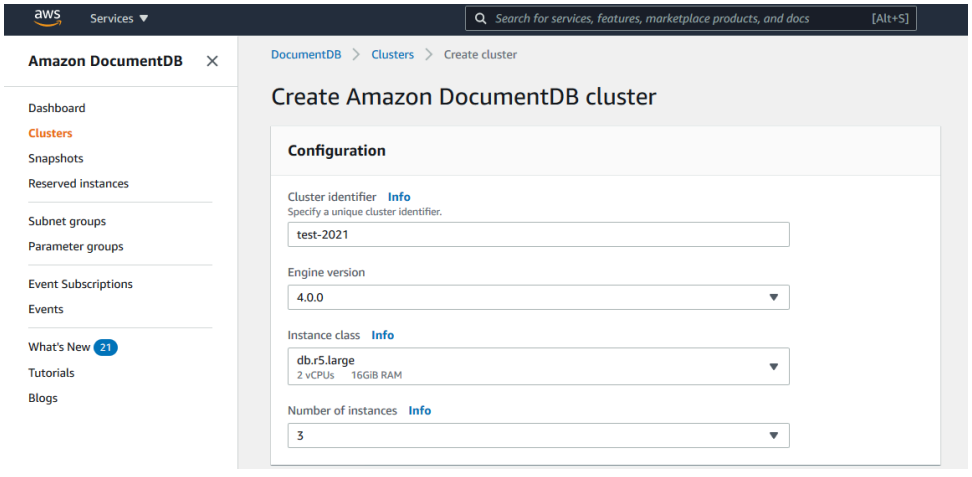


3. [作成] を選択します。

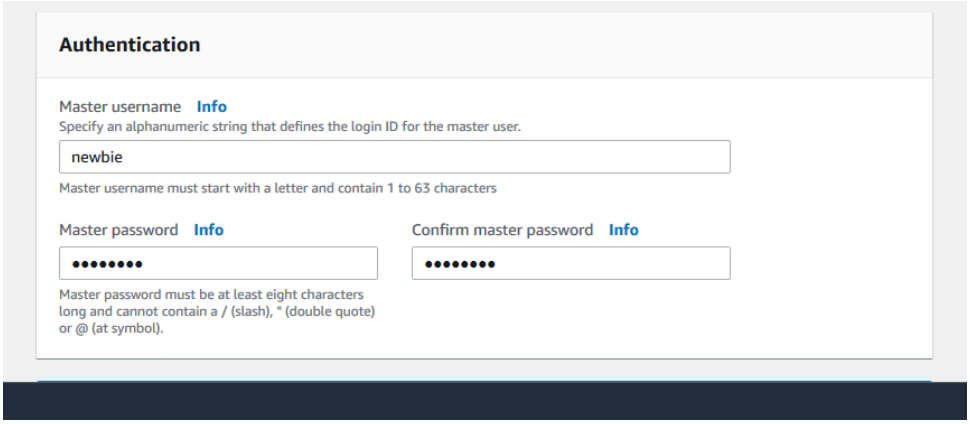


4. Amazon DocumentDB クラスター のフォームの [設定] セクションをそれぞれに応じて入力します。

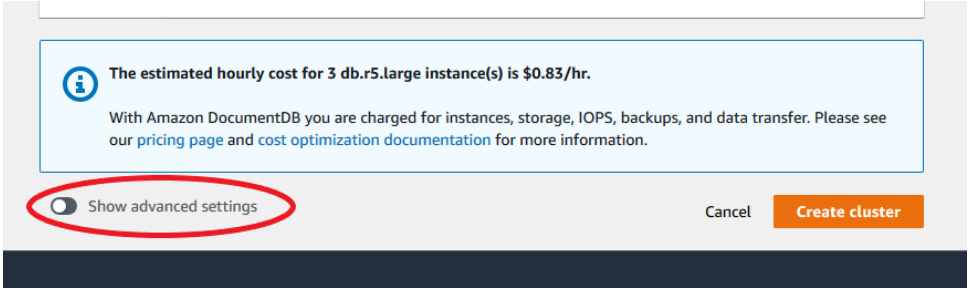
   - クラスター識別子: このインスタンスの一意の識別子を入力するか、クラスター識別子に基づき Amazon DocumentDB でインスタンスを指定できるようにします。
   - エンジンバージョン: [4.0.0] を選択します。
   - インスタンスクラス: [db.r5.large] を選択します。
   - インスタンス数: [3] を選択します。



5. [認証] セクションで、マスターユーザーネームとマスターパスワードを入力します。

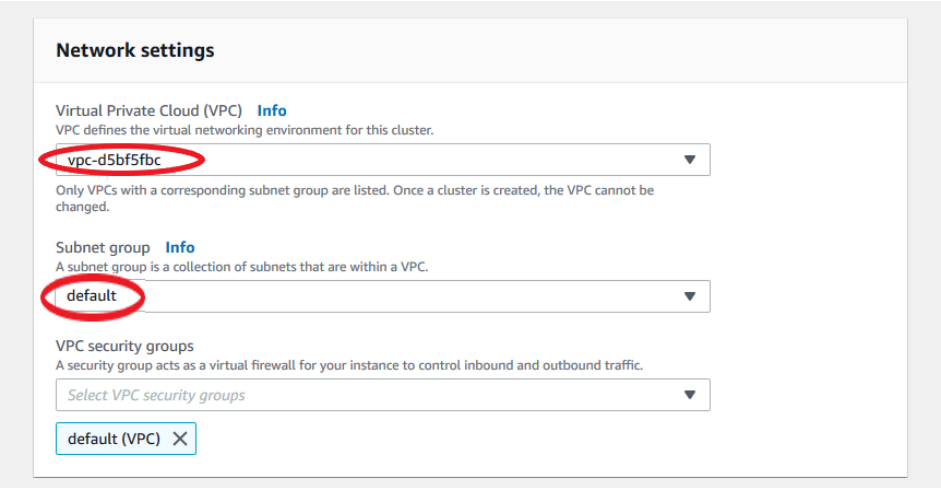


6. [詳細設定を表示] を選択します。

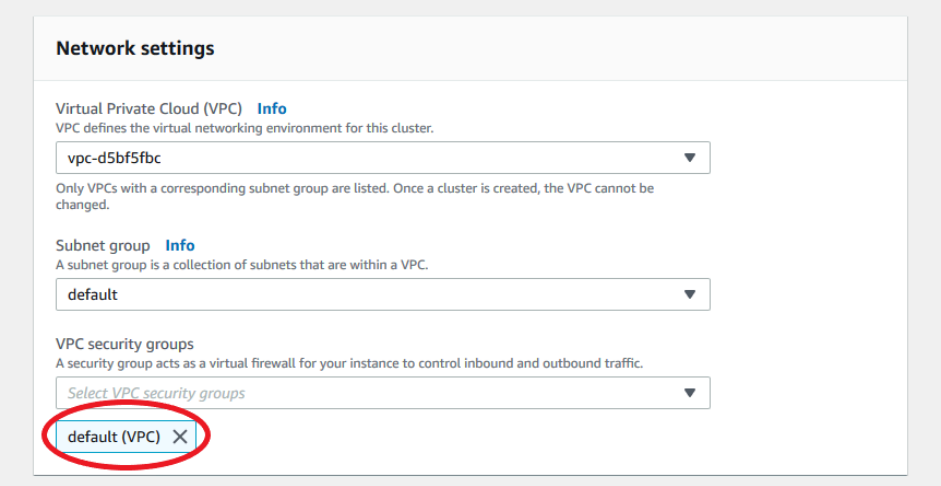


7. [ネットワーク設定] セクションで以下の操作を行います。

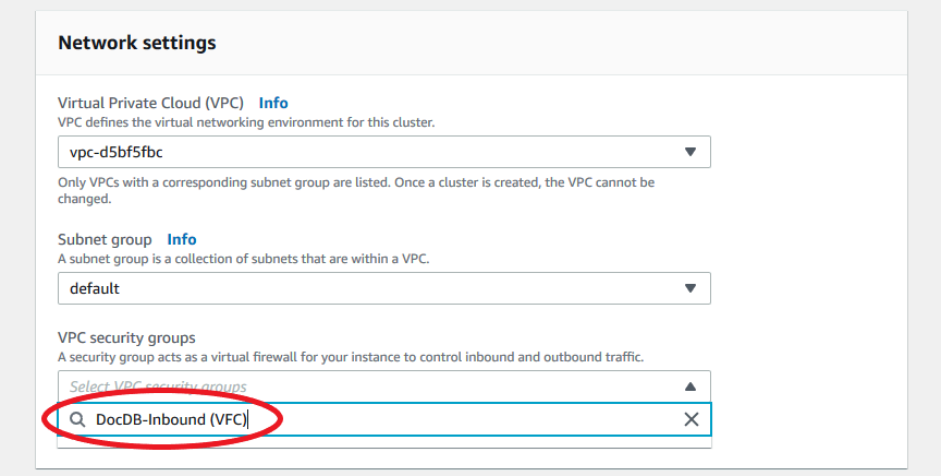
   - 仮想プライベートクラウド (VPC) と サブネットグループ のデフォルトオプションを保持します。



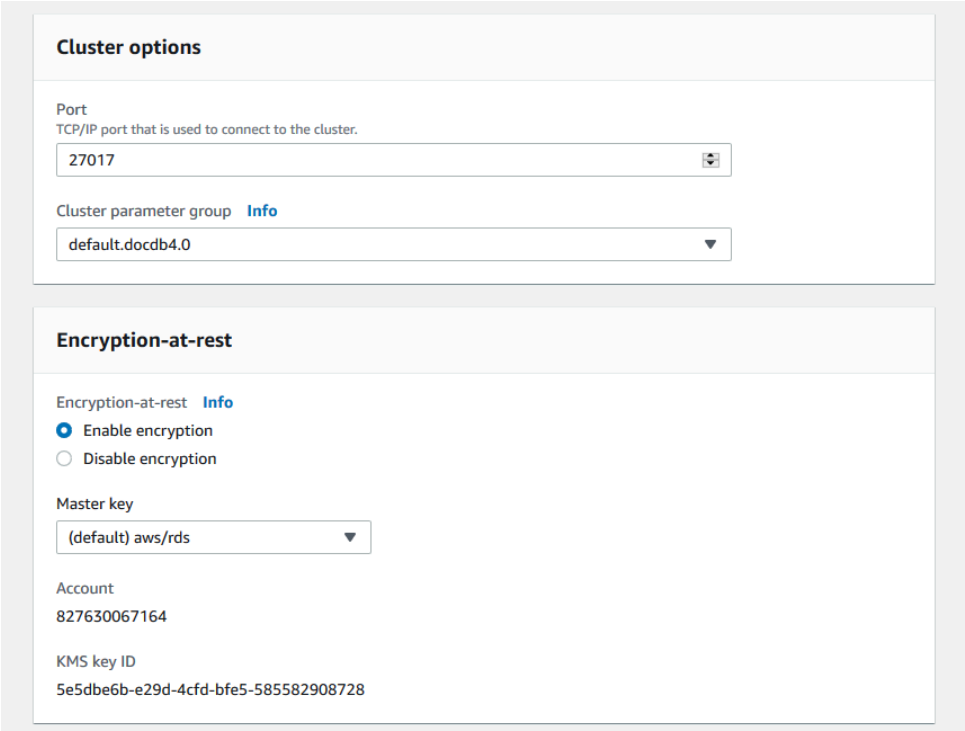
   - VPC セキュリティグループ では、デフォルト VPC は既に追加されている必要があります。



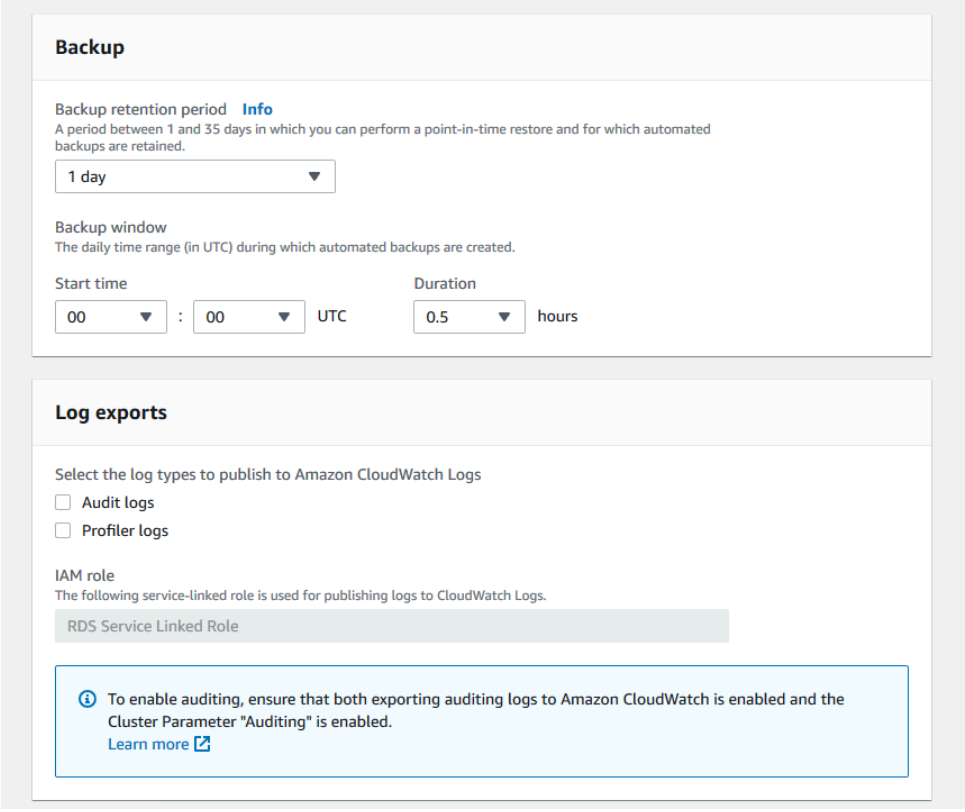
- [VPC セキュリティグループ] フィールドに DocDB を入力し、[DocDB-Inbound (VPC)] を選択します。



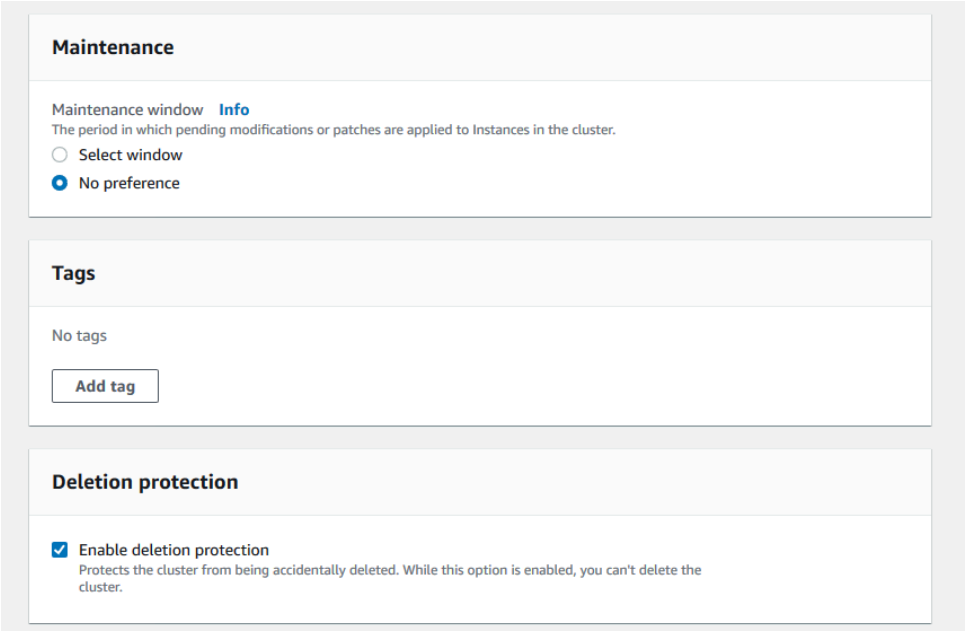
8. クラスターオプション と 保管時の暗号化 では、デフォルトの選択のままにします。



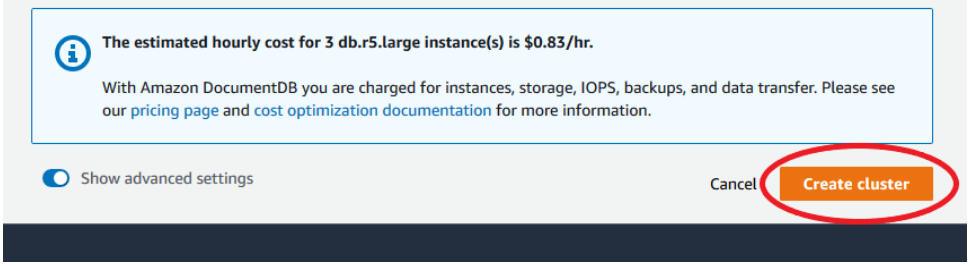
9. バックアップ と ログのエクスポート では、デフォルトの選択のままにします。



10. メンテナンス、タグ、および 削除保護 では、デフォルトの選択のままにします。



11. 次に、クラスターの作成を選択します。



の使用 AWS CLI

Amazon DocumentDB リージョンクラスターを作成するには、create-global-cluster AWS CLIを呼び出します。次の AWS CLI コマンドは、 という名前の Amazon DocumentDB クラスターを作成しますglobal-cluster-id。削除に対する保護についての詳細は、「Amazon DocumentDB クラスターの削除」を参照してください。

また、--engine-version は、デフォルトで最新のメジャーエンジンバージョンとなるオプションのパラメータです。現在のデフォルトのエンジンバージョンは です 5.0.0 (注: Amazon DocumentDB 8.0 は利用可能ですが、 として明示的に指定する必要がありますが8.0.0)。新しいメジャーエンジンバージョンがリリースされると、 のデフォルトのエンジンバージョン--engine-versionが更新され、最後のメジャーエンジンバージョンが反映されます。その結果、本番稼働用ワークロード、特にスクリプト、オートメーション、または CloudFormation テンプレートに依存するワークロードでは、 を目的のメジャーバージョン--engine-versionに明示的に指定します。

db-subnet-group-name または vpc-security-group-id が指定されていない場合、Amazon DocumentDB は、指定されたリージョンに対してデフォルトのサブネットグループと Amazon VPC セキュリティグループを使用します。

次の例では、各 *############* を独自の情報に置き換えます。

Linux、macOS、Unix の場合:

```
aws docdb create-db-cluster \
      --global-cluster-identifier global-cluster-id \
      --source-db-cluster-identifier arn:aws:rds:us-east-1:111122223333:cluster-id
```

Windows の場合:

```
aws docdb create-db-cluster ^
      --global-cluster-identifier global-cluster-id ^
      --source-db-cluster-identifier arn:aws:rds:us-east-1:111122223333:cluster-id
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "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"
    }
}
```

クラスターが作成されるまでに数分かかります。 AWS マネジメントコンソール または を使用して AWS CLI 、クラスターのステータスをモニタリングできます。詳細については、「Amazon DocumentDB クラスターのステータスのモニタリング」を参照してください。

> **Important**
>
> を使用して Amazon DocumentDB リージョンクラスター AWS CLI を作成する場合、インスタンスは作成されません。したがって、プライマリインスタンスと必要なレプリカインスタンスを明示的に作成してください。コンソールまたは を使用してインスタンス AWS CLI を作成できます。詳細については、Amazon DocumentDB API リファレンス」のクラスターへの Amazon DocumentDB インスタンスの追加「」とCreateDBCluster」を参照してください。

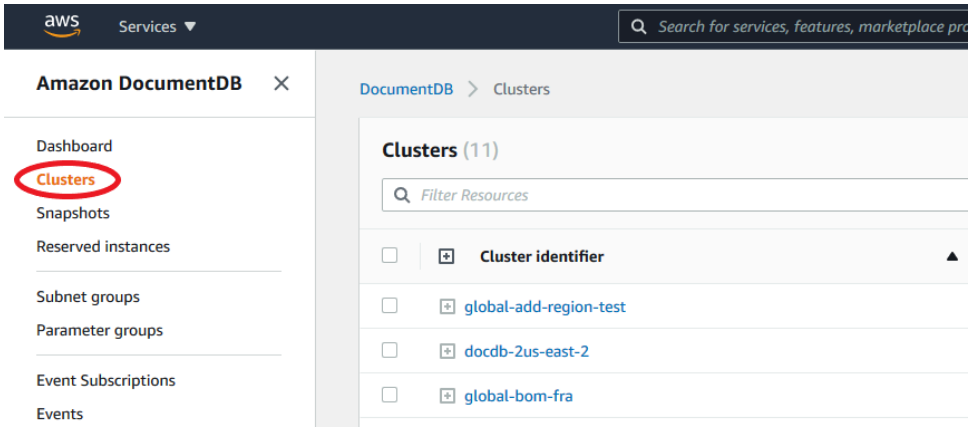
リージョンクラスターが使用可能になったら、Amazon DocumentDB グローバルクラスター AWS リージョン への の追加 の手順に従って、別のリージョンにセカンダリクラスターを追加できます。リージョンを追加すると、リージョンクラスターがプライマリクラスタになり、選択したリージョンに新しいセカンダリクラスターが作成されます。

Amazon DocumentDB グローバルクラスター AWS リージョン への の追加

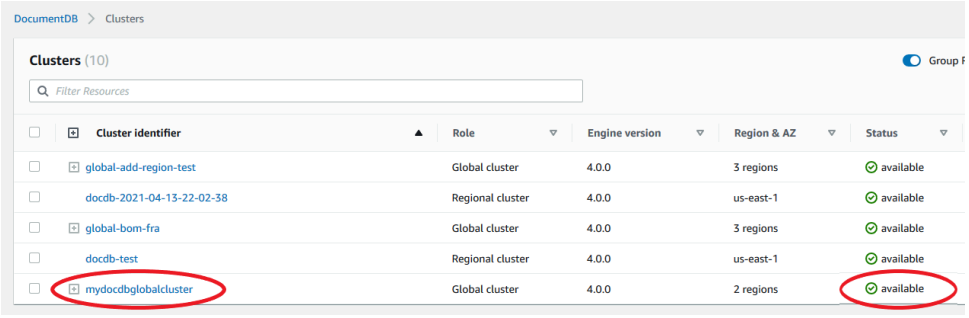
グローバルクラスターには、プライマリクラスターとは異なるリージョンに少なくとも 1 つのセカンダリクラスターが必要であり、最大 10 個のセカンダリクラスターを追加できます。追加する各セカンダリクラスターで、プライマリクラスターに許容されたレプリカの数を減らします。例えば、グローバルクラスターに 10 個のセカンダリリージョンがある場合、プライマリクラスターには 5 (15 個ではなく) のレプリカのみを含めることができます。詳細については、「設定」を参照してください。

の使用 AWS マネジメントコンソール

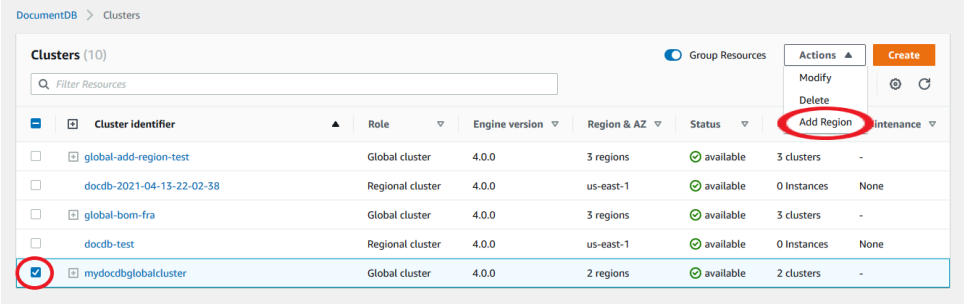
1. にサインイン AWS マネジメントコンソール し、Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択してください。



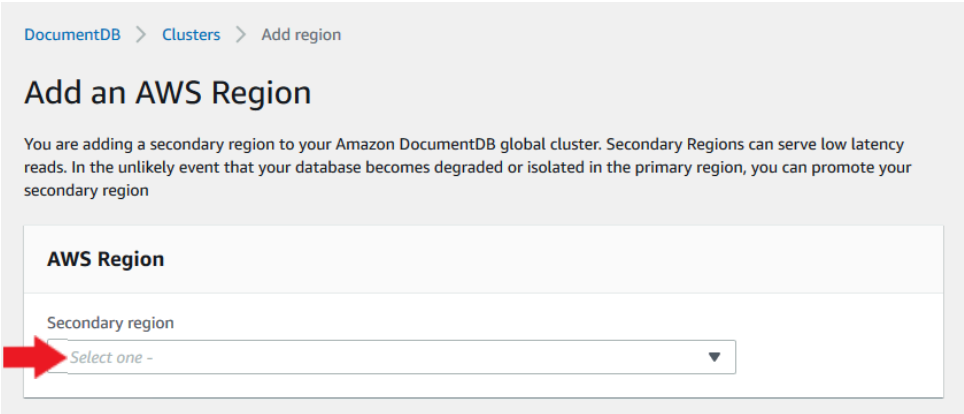
3. セカンダリクラスターを追加するクラスターを選択します。クラスターが `Available` であることを確認します。



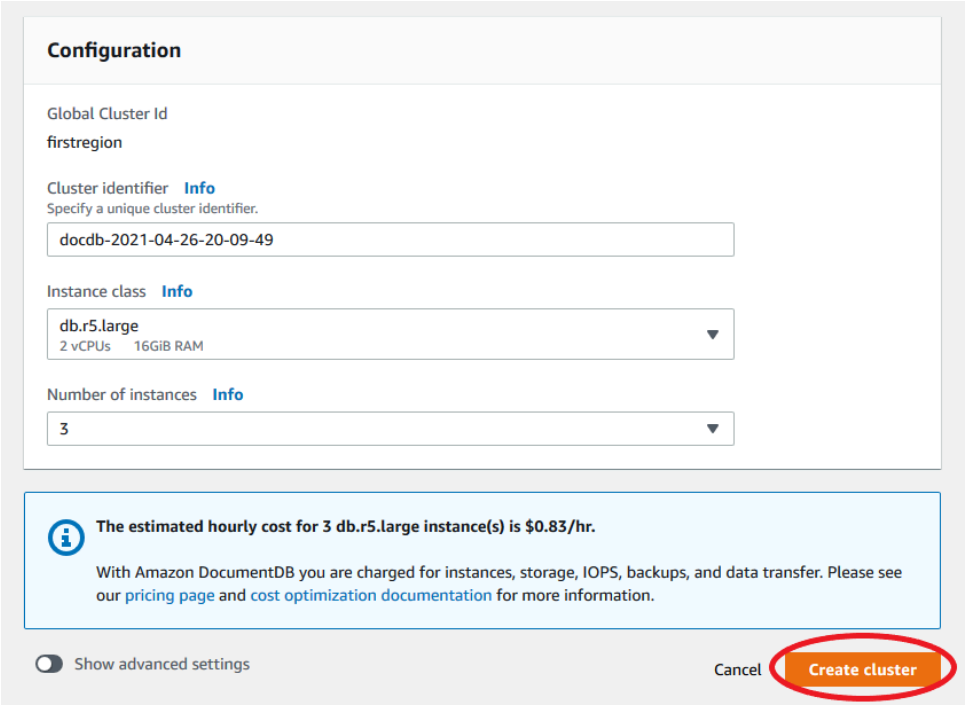
4. ドロップダウンメニューから [アクション] を選択し、[リージョンの追加] を選択します。



5. [AWS リージョンの追加] ページで、セカンダリリージョンを選択します。同じグローバルクラスターのセカンダリクラスターがすでに存在するリージョンを選択することはできません。また、プライマリクラスターと同じリージョンにすることはできません。これが最初に追加するリージョンである場合は、選択したグローバルクラスター ID も指定する必要があります。



6. 新しいリージョンのセカンダリクラスターの残りのフィールドを入力し、[クラスターの作成] を選択してください。リージョンの追加が完了すると、[クラスター] の AWS マネジメントコンソールのリストに表示されます。



の使用 AWS CLI

- お客様のグローバルクラスターの名前 `create-db-cluster` を指定して、(`--global-cluster-identifier`) CLI コマンドを使用します。他のパラメータでは、以下の操作を行います。
  - では`--region`、プライマリリージョン AWS リージョン とは異なる を選択します。
  - `--engine` および `--engine-version` パラメータの特定の値を選択します。
  - 暗号化されたクラスターの場合は、プライマリを暗号化`--source-region`用の AWS リージョン として指定します。

次の例では、新しい Amazon DocumentDB クラスターを作成し、それを読み取り専用セカンダリクラスターとしてグローバルクラスターにアタッチします。最後のステップでは、インスタンスを新しいクラスターに追加します。

次の例では、各 *############* を独自の情報に置き換えます。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

Amazon DocumentDB グローバルクラスターのスナップショットの使用

Amazon DocumentDB クラスターのスナップショットを復元することで、グローバルクラスターの開始ポイントとして使用することができます。これを行うには、スナップショットを復元し、新しいクラスターを作成する必要があります。これは、グローバルクラスタのプライマリクラスターとして機能します。次に、復元したクラスターを別のリージョンに追加して、グローバルクラスターに変換します。

Amazon DocumentDB グローバルクラスターの管理

グローバルクラスターを構成する個別のクラスターに対して、ほとんどの管理オペレーションを実行できます。コンソールで、[Group related resources (グループ関連のリソース)] を[クラスター] ペー

ジで選択すると、プライマリクラスターとセカンダリクラスターが、関連付けられたグローバルクラスターの下にグループとして表示されます。

グローバルクラスターの設定タブには、クラスターが実行されている AWS リージョン 、バージョン、グローバルクラスター識別子が表示されます。

トピック

- Amazon DocumentDB グローバルクラスターの変更
- Amazon DocumentDB グローバルクラスターのパラメータの変更
- Amazon DocumentDB グローバルクラスターからのクラスターの削除
- Amazon DocumentDB グローバルクラスターからのクラスターの削除
- セカンダリリージョンでのヘッドレス Amazon DocumentDB クラスターの作成

Amazon DocumentDB グローバルクラスターの変更

のクラスターページには、すべてのグローバルクラスターが AWS マネジメントコンソール 一覧表示され、それぞれのプライマリクラスターとセカンダリクラスターが表示されます。グローバルクラスターには、独自の構成設定があります。具体的には、プライマリクラスターとセカンダリクラスターに関連付けられたリージョンがあります。

グローバルクラスターに変更を加えると、変更をキャンセルする機会が与えられます。

[続行] を選択したら、変更を確認します。

Amazon DocumentDB グローバルクラスターのパラメータの変更

グローバルクラスター内の各クラスターで、クラスターパラメータグループを個別に設定することができます。ほとんどのパラメータの動作は、他の種類の Amazon DocumentDB クラスターと同じです。セカンダリクラスターをプライマリに昇格させる場合、予期しない動作の変更を避けるため、グローバルデータベース内のすべてのクラスター間で設定の一貫性を維持します。

例えば、別のクラスターがプライマリクラスターを肩代わりしたときに動作が変わらないように、タイムゾーンと文字セットに同じ設定を使用します。

Amazon DocumentDB グローバルクラスターからのクラスターの削除

グローバルクラスターからクラスターを削除したい場合がいくつかあります。例えば、プライマリクラスターの性能低下や切断が生じて、グローバルクラスターからクラスターを削除したい場合があります。その後、新しいグローバルクラスターを作成するために使用できるスタンドアロンのプロビ

ジョニングクラスターになります。詳細については[Amazon DocumentDB グローバルクラスターの手動フェイルオーバーの実行]を参照してください。

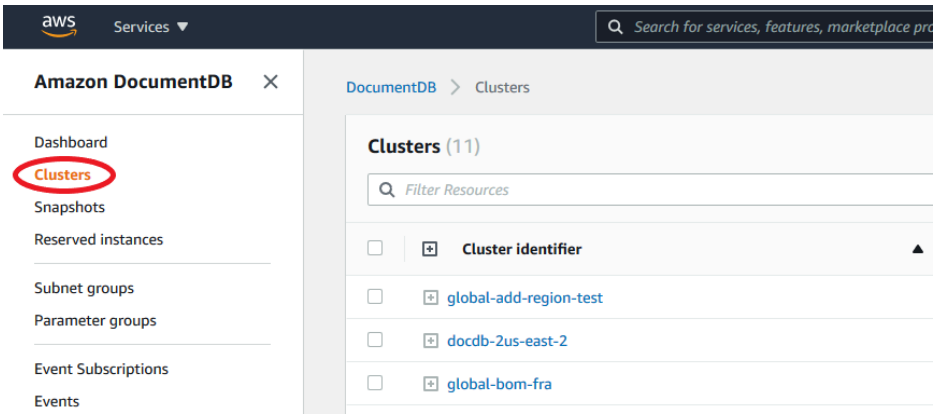
また、不要になったグローバルクラスターを削除するために、クラスターを削除する場合も可能です。関連するすべてのクラスターを削除 (デタッチ) した後、プライマリが最後に残るまで、グローバルクラスターを削除することはできません。詳細については、「[Amazon DocumentDB グローバルクラスターからのクラスターの削除]」を参照してください。

> **Note**
>
> クラスターがグローバルクラスターからデタッチされると、プライマリは同期されなくなります。これは、完全な読み取り/書き込み機能を備えたスタンドアロンのプロビジョニングクラスターになります。さらに、Amazon DocumentDB コンソールに表示されなくなりました。これは、クラスターが置かれている コンソールでリージョンを選択した場合にのみ表示されます。

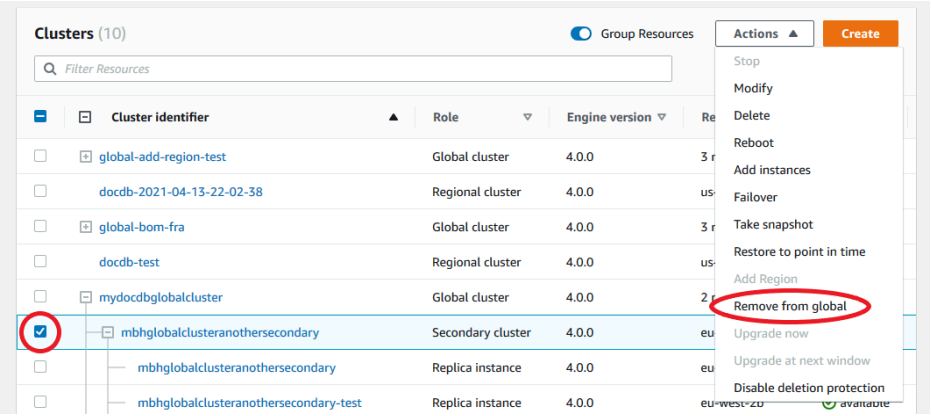
グローバルクラスターからクラスターを削除するには AWS マネジメントコンソール、、 AWS CLI、または RDS API を使用します。

Using the AWS マネジメントコンソール

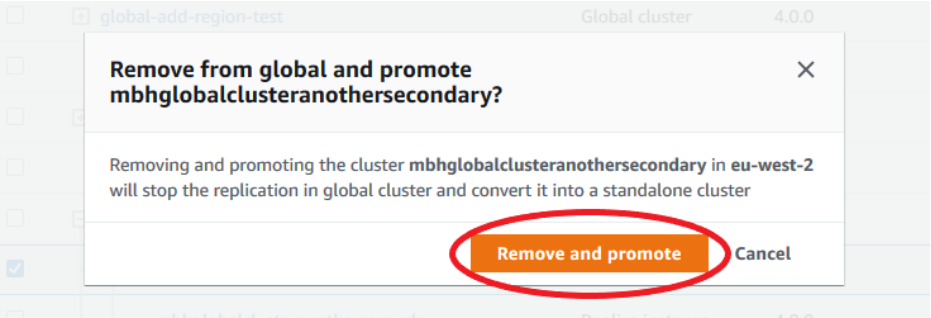
1. にサインイン AWS マネジメントコンソール し、Amazon DocumentDB コンソールに移動します。
2. 左のナビゲーションで [Clusters (クラスター)] を選択します。



3. グローバルクラスターを展開して、すべてのセカンダリクラスターを表示します。削除するセカンダリクラスターを選択します。[アクション] を選択し、ドロップダウンメニューで [グローバルから削除] を選択します。



4. グローバルクラスターからセカンダリをデタッチするかどうかを確認するプロンプトが表示されます。グローバルクラスターからクラスターを削除するには、[削除して昇格する] を選択します。



クラスターは、セカンダリとして機能しなくなり、プライマリクラスターと同期しなくなります。これは、完全な読み取り/書き込み機能を備えたスタンドアロンクラスターです。

すべてのセカンダリクラスターを解除または削除したら、次にプライマリクラスターを同じ方法で解除できます。すべてのセカンダリクラスターを削除するまで、グローバルクラスターからプライマリクラスターをデタッチ (削除) することはできません。グローバルクラスターは、リージョンと AZs。今後このグローバルクラスターを使用しない場合は、削除できます。

Using the AWS CLI

グローバルクラスターからクラスターを削除するには、次のパラメータを指定して、`remove-from-global-cluster` CLI コマンドを実行します。

- `--global-cluster-identifier` - グローバルクラスターの名前 (識別子)。
- `--db-cluster-identifier` - グローバルクラスターから削除する各クラスターの名前。

以下の例では、まずセカンダリクラスターを削除し、次にグローバルクラスターからプライマリクラスターを削除します。

Linux、macOS、Unix の場合:

```
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
```

グローバルクラスター内のセカンダリリージョンごとに `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN` コマンドを繰り返します。

Windows の場合:

```
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
```

グローバルクラスター内のセカンダリリージョンごとに `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN` コマンドを繰り返します。

Amazon DocumentDB グローバルクラスターからのクラスターの削除

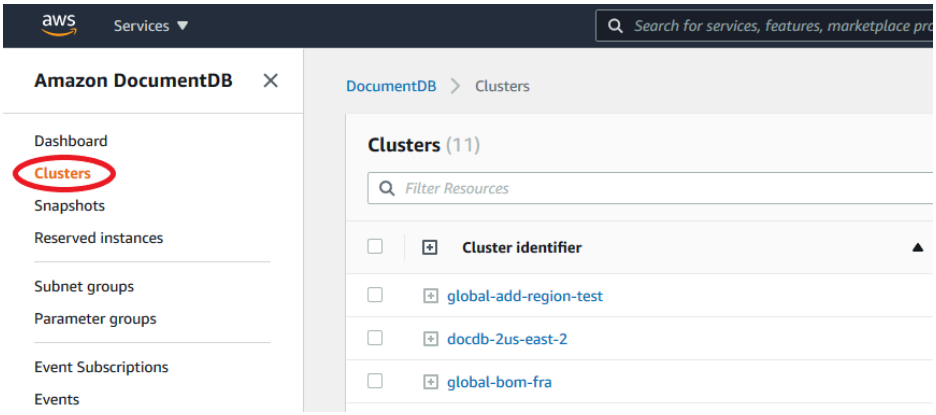
グローバルクラスターを削除するには、以下の操作を行います。

- グローバルクラスターからすべてのセカンダリクラスターを削除します。各クラスターは、スタンドアロンクラスターになります。前のセクション「Amazon DocumentDB グローバルクラスターからのクラスターの削除」を参照してください。
- 各スタンドアロンクラスターから、すべてのレプリカを削除します。

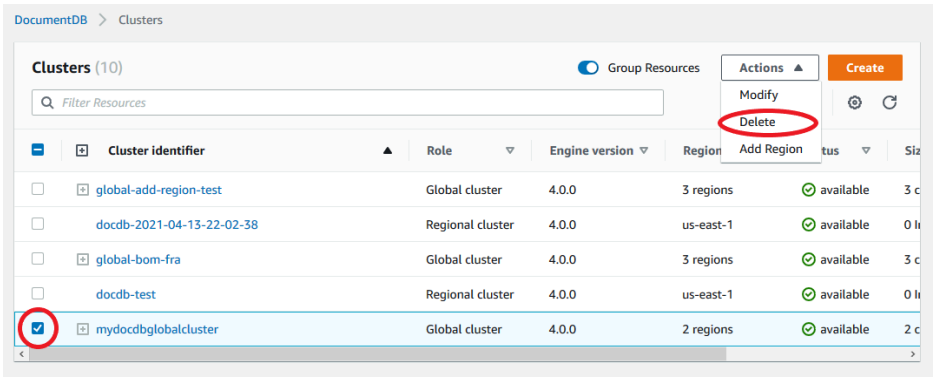
- グローバルクラスターからプライマリクラスターを削除します。これは、スタンドアロンクラスターになります。
- プライマリクラスターから、まずすべてのレプリカを削除し、次にプライマリインスタンスを削除します。新しいスタンドアロンクラスターからプライマリインスタンスを削除すると、通常クラスターとグローバルクラスターも削除されます。

Using the AWS マネジメントコンソール

1. にサインイン AWS マネジメントコンソール し、Amazon DocumentDB コンソールに移動します。
2. [クラスター] を選択し、削除するグローバルクラスターを見つけます。



3. グローバルクラスターを選択した状態で、[削除] を [アクション] メニューから選択します。



グローバルクラスターからすべてのクラスターが削除されていることを確認します。グローバルクラスターには、リージョンと AZs、クラスターのサイズがゼロと表示されます。グローバルクラスターにクラスターが含まれている場合は、これを削除できません。まず、前のステップの [Amazon DocumentDB グローバルクラスターからのクラスターの削除] の指示に従わなければなりません。

Using the AWS CLI

グローバルクラスターを削除するには、次の例に示すように、 の名前 AWS リージョン とグローバルクラスター識別子を使用して `delete-global-cluster` CLI コマンドを実行します。

Linux、macOS、Unix の場合:

```
aws docdb --region primary_region delete-global-cluster \
   --global-cluster-identifier global_cluster_id
```

Windows の場合:

```
aws docdb --region primary_region delete-global-cluster ^
   --global-cluster-identifier global_cluster_id
```

セカンダリリージョンでのヘッドレス Amazon DocumentDB クラスターの作成

Amazon DocumentDB グローバルクラスターには、プライマリ AWS リージョン とは異なる に少なくとも 1 つのセカンダリクラスターが必要ですが、セカンダリクラスターにはヘッドレス設定を使用できます。ヘッドレスセカンダリ Amazon DocumentDB クラスターは、インスタンスがないクラスターのことです。この種類の設定では、グローバルクラスターの費用を削減できます。Amazon DocumentDB クラスターでは、コンピューティングとストレージは分離されています。インスタンスがない場合、コンピューティングに対する課金は発生せず、ストレージに対してのみが課金されます。正しく設定されていれば、ヘッドレスセカンダリのストレージボリュームはプライマリクラスターと同期したままになります。

Amazon DocumentDB グローバルクラスターの作成時に通常行うように、セカンダリクラスターを追加します。ただし、プライマリクラスターがセカンダリへのレプリケーションを開始した後、セカンダリクラスターから読み取り専用インスタンスを削除します。このセカンダリクラスターにはインスタンスがないため、「ヘッドレス」と見なされます。ただし、ストレージボリュームはプライマリ Amazon DocumentDB クラスターと同期されたままとなります。

> **Important**
>
> 15 分以上リージョン全体の障害を許容できる場合にのみ、ヘッドレスクラスターを使用してください。ヘッドレスセカンダリクラスターでリージョン全体の障害から復旧するには、フェイルオーバー後に新しいインスタンスを作成する必要があります。新しいインスタンスが使用可能になるまでに 10 ~ 15 分かかる場合があります。

グローバルクラスターにヘッドレスセカンダリクラスターを追加する方法

1. にサインイン AWS マネジメントコンソール し、Amazon DocumentDB コンソールを開きます。
2. 左のナビゲーションで [Clusters (クラスター)] を選択します。
3. セカンダリクラスターを必要とするグローバルクラスターを選択します。プライマリクラスターが `Available` であることを確認します。
4. アクション で、リージョンの追加 を選択します。
5. リージョンの追加ページで、セカンダリリージョンを選択します。

   > **Note**
   >
   > 同じグローバルクラスターのセカンダリクラスターが既にあるリージョンを選択することはできません。また、プライマリクラスターと同じリージョンにすることはできません。

6. 新しい リージョンのセカンダリ クラスターの残りのフィールドに入力してください。これらは、他のクラスターインスタンスと同じ設定オプションです。
7. リージョンを追加します。グローバルクラスターへのリージョンの追加が完了すると、`Clusters`の のリストに表示されます AWS マネジメントコンソール。
8. AWS マネジメントコンソール または を使用して、続行する前にセカンダリクラスターとそのリーダーインスタンスのステータスを確認します AWS CLI。以下に、 AWS CLIを使用した場合のコマンドの例を示します。

   ```
   $ aws docdb describe-db-clusters --db-cluster-identifier secondary-cluster-id --
   query '*[].[Status]' --output text
   ```

   新しく追加されたセカンダリクラスターのステータスが作成中から使用可能に変わるまでに数分かかる場合があります。クラスターが使用可能になったら、リーダーインスタンスを削除できます。
9. セカンダリクラスターでリーダーインスタンスを選択し、[Delete (削除)] を選択します。
10. リーダーインスタンスを削除しても、セカンダリクラスターはグローバルクラスターの一部にとどまります。関連付けられたインスタンスは存在しません。

> **Note**
>
> このヘッドレスセカンダリ Amazon DocumentDB クラスターを使用して、Amazon DocumentDB グローバルクラスターをプライマリリージョンの予期しない停止から手動で復旧できます。

Amazon DocumentDB グローバルクラスターへの接続

グローバルクラスターへの接続方法は、クラスターへの書き込みと、クラスターからの読み取りの、どちらを必要とするのかに応じて異なります。

- 読み取り専用のリクエストまたはクエリ場合、 AWS リージョンにあるクラスターのリーダーエンドポイントに接続します。
- データ操作言語 (DML) またはデータ定義言語 (DDL) のステートメントを実行するには、プライマリクラスターのクラスターエンドポイントに接続します。このエンドポイントは、アプリケーション AWS リージョン とは異なる にある可能性があります。

コンソールでグローバルクラスターを表示すると、そのすべてのクラスターに関連付けられているすべての汎用エンドポイントを表示できます。

グローバルクラスターへの接続方法は、データベースへの書き込みと、データベースからの読み取りの、どちらを必要とするのかに応じて異なります。プライマリリージョンから処理する DDL、DML、読み取りオペレーションの場合は、レプリカセットモードのクラスターエンドポイントを使用してプライマリクラスターに接続します。読み取り設定は `secondaryPreferred=true` です。これにより、書き込みトラフィックがプライマリクラスターのライターインスタンスにルーティングされ、読み取りトラフィックがプライマリクラスターのレプリカインスタンスにルーティングされます。

クロスリージョンの読み取り専用トラフィックの場合は、レプリカセットモードでクラスターエンドポイントを使用してセカンダリクラスターの 1 つに接続します。すべてのインスタンスは読み取り専用レプリカインスタンスであるため、 `primary` 以外の読み取り設定 (`secondary`、`secondaryPreferred`、 など`nearest`) を指定します。レイテンシーを最小限にするときは、お使いの リージョンか、最寄りの リージョンの、いずれかにあるリーダーエンドポイントを選択します。

Amazon DocumentDB グローバルクラスターのモニタリング

Amazon DocumentDB (MongoDB 互換性) は CloudWatch と統合することで、クラスターのオペレーションメトリクスを収集および分析できます。これらのメトリクスは、CloudWatch コンソール、Amazon DocumentDB コンソール、 AWS Command Line Interface (AWS CLI)、または CloudWatch API を使用してモニタリングできます。

グローバルクラスターをモニタリングするには、次の CloudWatch メトリクスを使用します。

メトリクス	説明
`GlobalClusterReplicatedWriteIO`	プライマリ のクラスターボリュームからセカンダリ のクラスターボリューム AWS リージョン にレプリケートされた課金書き込み I/O オペレーションの平均数。5 分間隔で AWS リージョン報告されます。各セカンダリリージョン`ReplicatedWriteIOs` にレプリケートされる の数は、プライマリリージョンによって `VolumeWriteIOPs` 実行されるリージョン内の数と同じです。
`GlobalClusterDataTransferBytes`	プライマリクラスターからセカンダリクラスター AWS リージョン に転送されるデータの量 AWS リージョン。バイト単位で測定します。
`GlobalClusterReplicationLag`	変更イベントをプライマリクラスターの から セカンダリクラスターの AWS リージョン にレプリケートする際の、ミリ秒単位の遅延の量 AWS リージョン

これらのメトリクスを表示する方法の詳細については、CloudWatch データの表示」を参照してください。

ディザスタリカバリと Amazon DocumentDB グローバルクラスターについて

トピック

- Amazon DocumentDB グローバルクラスターのマネージドフェイルオーバーの実行
- Amazon DocumentDB グローバルクラスターの手動フェイルオーバーの実行
- Amazon DocumentDB グローバルクラスターのスイッチオーバーの実行
- グローバルクラスターのスイッチオーバーまたはフェイルオーバーのブロック解除
- Amazon DocumentDB グローバルクラスターRPOs の管理

グローバルクラスターを使用することで、リージョンの障害などの災害から迅速に復旧できます。災害からの復旧は、通常、RTO と RPO (目標復旧時点) の値を使用して測定します。

- 目標復旧時間 (RTO)– 災害後にシステムが稼働状態に戻るまでにかかる時間。つまり、RTO はダウンタイムを測定します。グローバルクラスターの場合、RTO は分単位です。
- 目標復旧時点 (RPO) – 損失する可能性があるデータの量 (時間単位)。グローバルクラスターの場合、目標復旧時点 (RPO) は通常、秒単位で測定されます。
- 計画外の停止から復旧するには、グローバルクラスター内のいずれかのセカンダリへのクロスリージョンフェイルオーバーを実行できます。グローバルクラスターに複数のセカンダリリージョンがある場合は、プライマリとして昇格させるすべてのセカンダリリージョンをデタッチしてください。次に、これらのセカンダリリージョンの 1 つを新しいプライマリに昇格させます AWS リージョン。最後に、他の各セカンダリリージョンに新しいクラスターを作成し、それらのクラスターをグローバルクラスターにアタッチします。

Amazon DocumentDB グローバルクラスターのマネージドフェイルオーバーの実行

このアプローチは、真の地域災害やサービスレベルの全面的な停止が発生した場合でも事業を継続できるようにするためのものです。

マネージドフェイルオーバー中、プライマリクラスターは選択したセカンダリリージョンにフェイルオーバーされ、お使いの Amazon DocumentDB グローバルクラスターの既存のレプリケーショントポロジが維持されます。選択したセカンダリクラスターは、読み取り専用ノードの 1 つを完全な読み取り状態に昇格します。このステップにより、クラスターがプライマリクラスターのロールを引き受けることができます。クラスターが新しいロールを引き受ける間、データベースは短時間使用できなくなります。古いプライマリクラスターから選択したセカンダリクラスターにレプリケートされなかったデータは、このセカンダリクラスターが新しいプライマリクラスターになると失われます。古いプライマリボリュームは、新しいプライマリと同期する前にスナップショットを取得するべく、レプリケートされていないデータがスナップショットに保持されるように最善を尽くします。

> **Note**
>
> プライマリクラスターとすべてのセカンダリクラスターに同じエンジンバージョンがある場合にのみ、Amazon DocumentDB グローバルクラスターへのマネージドクロスリージョンクラスターフェイルオーバーを実行できます。ご使用のエンジンバージョンに互換性がない場合は、Amazon DocumentDB グローバルクラスターの手動フェイルオーバーの実行 の手順に従ってフェイルオーバーを手動で実行できます。
> リージョンのエンジンバージョンが一致しない場合、フェイルオーバーはブロックされます。保留中のアップグレードを確認して適用し、リージョンのすべてのエンジンバージョンが一致し、グローバルクラスターフェイルオーバーがブロック解除されていることを確認します。詳細については、「グローバルクラスターのスイッチオーバーまたはフェイルオーバーのブロック解除」を参照してください。

データ損失を最小限に抑えるには、この機能を使用する前に次の操作を行います。

- 各アプリケーションをオフラインにして、Amazon DocumentDB グローバルクラスターのプライマリクラスターへの書き込みが送信されないようにします。
- すべての Amazon DocumentDB セカンダリクラスターのラグタイムをチェックします。レプリケーションの遅延が最も少ないセカンダリリージョンを選択すると、現在障害が発生しているプライマリリージョンでのデータ損失を最小限に抑えることができます。Amazon CloudWatch 上で `GlobalClusterReplicationLag` メトリクスを確認して、グローバルクラスター内のすべての Amazon DocumentDB セカンダリクラスターのラグタイムを確認します。これらのメトリクスは、各セカンダリクラスターへのレプリケーションがプライマリクラスターに対してどの程度遅れているかをミリ秒単位で示します。

  Amazon DocumentDB 向け CloudWatch メトリクスの詳細については、「Amazon DocumentDB のメトリクス」を参照してください。

マネージドフェイルオーバー中、選択されたセカンダリクラスターは、プライマリとして新しいロールに昇格されます。ただし、プライマリクラスターのさまざまな設定オプションは引き継がれません。構成の不一致は、パフォーマンスの問題、ワークロードの非互換性、およびその他の異常な動作につながる可能性があります。このような問題を回避するには、以下の Amazon DocumentDB グローバルクラスター間の違いを解決します。

- 必要に応じて、新しいプライマリの Amazon DocumentDB クラスターパラメータグループを設定する — Amazon DocumentDB グローバルクラスター内のクラスターごとに Amazon DocumentDB

クラスターパラメータグループを個別に設定できます。ただし、セカンダリクラスターを昇格してプライマリロールを引き継ぐ場合、セカンダリからのパラメータグループは、プライマリとは異なる設定になっている可能性があります。その場合は、昇格されたセカンダリクラスターのパラメータグループを、プライマリクラスターの設定に適合するように変更してください。この方法の詳細は、「Amazon DocumentDB クラスターパラメータグループを変更する」を参照してください。

- 各種モニタリングツールやオプションを構成する (Amazon CloudWatch Events やアラームなど) - グローバルクラスターに必要なログ機能、アラームなどを使用して、昇格されたクラスターの設定を行います。パラメータグループと同様に、フェイルオーバープロセス中にこれらの機能の設定がプライマリから継承されることはありません。レプリケーションラグなどの一部の CloudWatch メトリクスは、セカンダリリージョンのみで使用できます。そのため、フェイルオーバーによってメトリクスの表示方法やアラームの設定方法が変わり、定義済みのダッシュボードを変更する必要が生じる場合があります。Amazon DocumentDB クラスターとモニタリングの詳細については、「Amazon DocumentDB のモニタリングと口グ記録」を参照してください。

通常、選択したセカンダリクラスターが 1 分以内に主要な役割を引き継ぎます。新しいプライマリリージョンのライターノードが使用可能になり次第、アプリケーションをそのライターノードに接続してワークロードを再開できます。Amazon DocumentDB が新しいプライマリクラスターを昇格させると、追加のセカンダリリージョンクラスターはすべて自動的にリビルドされます。

Amazon DocumentDB グローバルクラスターは非同期レプリケーションを使用するため、レプリケーションラグはセカンダリリージョンごとに異なる場合があります。これに対し Amazon DocumentDB は、これらのセカンダリリージョンを、新しいプライマリリージョンクラスターとまったく同じポイントインタイムデータを持つようにリビルドします。ストレージボリュームのサイズとリージョン間の距離によっては、再構築タスクが完了するまでに数分から数時間かかることがあります。セカンダリージョンのクラスターが新しいプライマリリージョンからの再構築を完了すると、読み取りアクセスが可能になります。新しいプライマリライターが昇格して使用可能になると、新しいプライマリリージョンのクラスターが、Amazon DocumentDB グローバルクラスターの読み取りおよび書き込みオペレーションを処理できるようになります。

グローバルクラスターの元のトポロジを復元するために、Amazon DocumentDB は古いプライマリリージョンの可用性を監視します。そのリージョンが再び正常な状態になり利用可能になるとすぐに、Amazon DocumentDB はそのリージョンをセカンダリリージョンとしてグローバルクラスターに自動的に追加します。古いプライマリリージョンに新しいストレージボリュームを作成する前に、Amazon DocumentDB は障害発生時の古いストレージボリュームのスナップショットを取得しようとします。これにより、欠落しているデータを回復することができます。このオペレーションが成功すると、Amazon DocumentDB は「rds:docdb-unplanned-global-failover-name-of-old-primary-DB-cluster-timestamp」という名前のスナップショットを AWS マネジメントコンソールのスナップ

ショットセクションに格納します。このスナップショットは、DescribeDBClusterSnapshots API オペレーションによって返される情報にもリストアップされます。

> **Note**
>
> 古いストレージボリュームのスナップショットは、古い主クラスターに設定されたバックアップ保持期間の対象となるシステムスナップショットです。このスナップショットを保存期間外に保存するには、スナップショットをコピーして手動スナップショットとして保存できます。価格などのスナップショットのコピーの詳細については、「クラススナップショットのコピー」を参照してください。

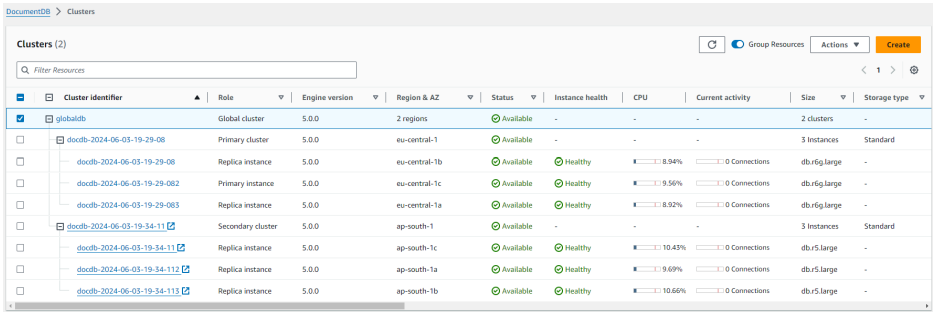
元のトポロジを復元できたら、ビジネスとワークロードにとって最も都合のよいタイミングでスイッチオーバーオペレーションを実行することで、グローバルクラスターを元のプライマリリージョンにフェイルバックします。これを行うには、「Amazon DocumentDB グローバルクラスターのスイッチオーバーの実行」の手順を実行します。

Amazon DocumentDB グローバルクラスターは、 AWS マネジメントコンソール、、 AWS CLIまたは Amazon DocumentDB API を使用してフェイルオーバーできます。

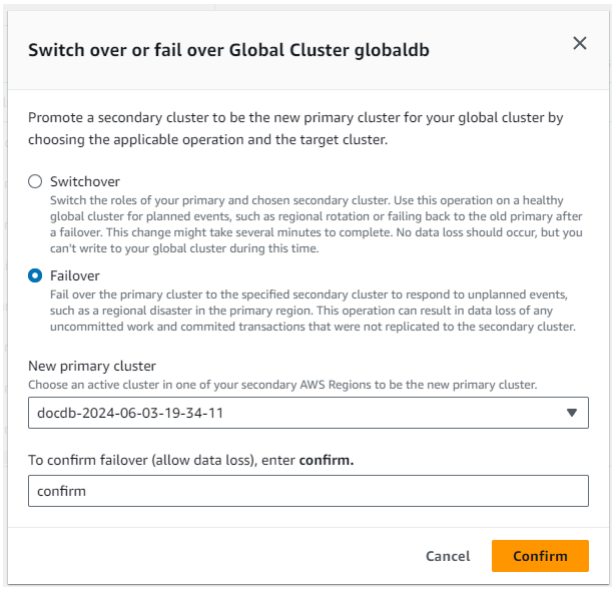
Using the AWS マネジメントコンソール

Amazon DocumentDB グローバルクラスターでマネージドフェイルオーバーを実行する

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択してください。
3. フェイルオーバーしたい Amazon DocumentDB グローバルクラスターを選択します。



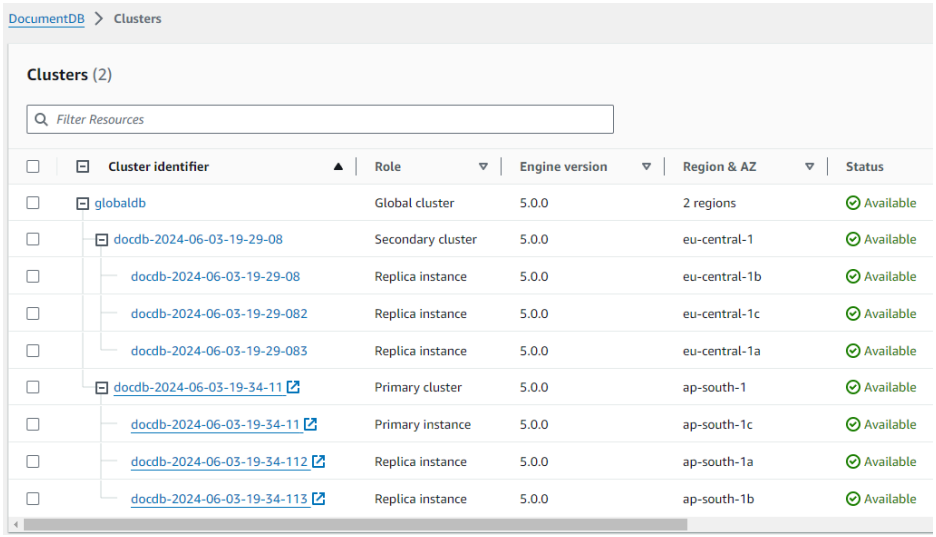
4. [アクション] メニューから [スイッチオーバーまたはフェイルオーバーする] を選択します。
5. 表示されるダイアログボックスで、[フェイルオーバー] を選択し、[新しいプライマリクラスター] フィールドのドロップダウンリストからセカンダリクラスターを選択します。



6. 最後のフィールドに「confirm」 (確定する) とキー入力します。[Confirm] (確認) を選択します。

   プライマリクラスターのステータスが「フェイルオーバー」に変わります。この状態になるまでには約 1 分かかります。この間、新しいプライマリクラスターのステータスは「変更中」と表示されます。新しいプライマリが昇格されると、「利用可能」の表示に変わり、読み取りおよび書き込みトランザクションを処理できるようになります。古いプライマリを含むセカンダリリージョンは、新しいプライマリに再同期している間に「再同期中...」と表示されます。新しいプライマリと同様に、古いプライマリも、ステータスが「利用可能」に変わってからのみトランザクションの処理が可能となります。

7. 完了すると、元のプライマリクラスターはセカンダリクラスターになります。選択したセカンダリクラスターがプライマリクラスターになります。



Using the AWS CLI

Amazon DocumentDB グローバルクラスターでマネージドフェイルオーバーを実行する

failover-global-cluster CLI コマンドを実行して、Amazon DocumentDB グローバルクラスターをフェイルオーバーします。コマンドを使用して、次のオプションの値を渡します。

- `--region`
- `--global-cluster-identifier`
- `--target-db-cluster-identifier`
- `--allow-data-loss`

次の例では*##############*をユーザー自身の情報で置き換えます。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

Amazon DocumentDB グローバルクラスターの手動フェイルオーバーの実行

1 つの 内のクラスター全体が使用 AWS リージョン できなくなった場合は、グローバルクラスター内の別のクラスターを昇格させて、読み取り/書き込み機能を持つことができます。

別の AWS リージョン にあるクラスターがプライマリクラスターとしてより適格であると判断される場合は、グローバルクラスターフェイルオーバー機能を手動で起動することができます。たとえば、特定のセカンダリクラスターの容量を増やして、プライマリクラスターに昇格させることもでき

ます。または、 間のアクティビティのバランスが AWS リージョン 変化する可能性があるため、プライマリクラスターを別のクラスターに切り替えると、書き込みオペレーションのレイテンシーが低下する AWS リージョン 可能性があります。

次の手順では、Amazon DocumentDB グローバルクラスターのセカンダリクラスターの 1 つを昇格させる方法を説明します。

セカンダリクラスターを昇格させるには

1. 停止 AWS リージョン 時に、 のプライマリクラスターへの DML ステートメントやその他の書き込みオペレーションの発行を停止します。
2. 新しいプライマリクラスター AWS リージョン として使用するセカンダリからクラスターを特定します。グローバルクラスター AWS リージョン に 2 つ (またはそれ以上) のセカンダリがある場合は、ラグタイムが最も短いセカンダリクラスターを選択します。
3. 選択したセカンダリクラスターをグローバルクラスターからデタッチします。

   グローバルクラスターからセカンダリクラスターを削除すると、プライマリからこのセカンダリへのレプリケーションが直ちに停止され、完全な読み取り/書き込み機能を備えたスタンドアロンのプロビジョニングクラスターへと昇格されます。停止したリージョンのプライマリクラスターに関連付けられている他のセカンダリクラスターは引き続き利用可能で、アプリケーションからの呼び出しを受け入れることができます。また、リソースを使用することになります。グローバルクラスターを再作成するため、スプリットブレインなどの問題を回避するために、以下のステップで新しいグローバルクラスターを作成する前に、他のセカンダリクラスターを削除します。

   アタッチ解除の詳細なステップについては、Amazon DocumentDB グローバルクラスターからのクラスターの削除 を参照してください。
4. このクラスターは、次のステップでリージョンを追加すると、新しいグローバルクラスターのプライマリクラスターになります。
5. クラスター AWS リージョン に を追加します。これを行うと、プライマリからセカンダリへのレプリケーションプロセスがスタートされます。
6. 必要に応じて追加 AWS リージョン し、アプリケーションをサポートするために必要なトポロジを再作成します。グローバルクラスター内 (スプリットブレインの問題) のクラスター間のデータの不整合を避けるために、これらの変更を行う前、最中、および後に、アプリケーションの書き込みが正しいクラスターに送信されていることを確認してください。
7. 停止状態が解決されて元の AWS リージョン をプライマリクラスターとして再度割り当てる準備が完了したら、同じステップを逆に実行します。

8. グローバルクラスターからセカンダリクラスターの 1 つを削除します。これにより、読み取り/書き込みトラフィックを提供できるようになります。
9. 元の AWS リージョンのプライマリクラスターにすべての書き込みトラフィックをリダイレクトします。
10. を追加して AWS リージョン 、以前 AWS リージョン と同じ に 1 つ以上のセカンダリクラスターを設定します。

Amazon DocumentDB グローバルクラスターは AWS SDKs を使用して管理できるため、ディザスタリカバリとビジネス継続性計画のユースケースのグローバルクラスターフェイルオーバープロセスを自動化するソリューションを作成できます。そのようなソリューションの 1 つは、Apache 2.0 ライセンスに基づいてお客様に提供されており、こちらのツールリポジトリからアクセスできます。このソリューションは、エンドポイント管理に Amazon Route 53 を活用し、適切なイベントに基づいてトリガーできる AWS Lambda 機能を提供します。

Amazon DocumentDB グローバルクラスターのスイッチオーバーの実行

スイッチオーバーを使用すると、プライマリクラスターのリージョンを定期的に変更できます。この機能は、運用メンテナンス、その他の計画された運用手順など、管理されたシナリオを対象としています。

スイッチオーバーを使用する一般的なユースケースは 3 つあります。

- 特定の業界に課せられる「リージョナルローテーション」要件向け。たとえば、金融サービス規制では、ディザスタリカバリ手順が定期的に実施されるように、Tier-0 システムを別の地域に数か月間切り替えることが求められる場合があります。
- マルチリージョンの「follow-the-sun」アプリケーション向け。たとえば、ある企業が、さまざまなタイムゾーンの営業時間に基づいて、さまざまなリージョンで低レイテンシーの書き込みを提供したいとします。
- データ損失ゼロの方法として、フェイルオーバー後に元のプライマリリージョンにフェイルバックします。

> **Note**
>
> スイッチオーバーは、正常な Amazon DocumentDB グローバルクラスターでの使用向けに設計された機能です。予期しないシステム停止から回復するには、Amazon DocumentDB グローバルクラスターの手動フェイルオーバーの実行 の該当する手順に従ってください。

スイッチオーバーを実行するには、すべてのセカンダリリージョンがプライマリとまったく同じエンジンバージョンを実行している必要があります。リージョンのエンジンバージョンが一致しない場合、スイッチオーバーはブロックされます。保留中のアップグレードを確認して適用し、リージョンのすべてのエンジンバージョンが一致し、グローバルクラスターのスイッチオーバーがブロック解除されていることを確認します。詳細については、「グローバルクラスターのスイッチオーバーまたはフェイルオーバーのブロック解除」を参照してください。

スイッチオーバー中、Amazon DocumentDB は、グローバルクラスターの既存のレプリケーショントポロジを維持しながら、プライマリクラスターを選択されたセカンダリリージョンにスイッチオーバーします。なお、Amazon DocumentDB は、すべてのセカンダリリージョンクラスターがプライマリリージョンクラスターと完全に同期されるまで待ってからスイッチオーバープロセスを開始します。次に、プライマリリージョンの DB クラスターは読み取り専用になり、選択したセカンダリ DB クラスターは、読み取り専用ノードの 1 つを、フルライターステータスに昇格させます。このノードをライターに昇格させると、そのセカンダリクラスターがプライマリクラスターの役割を引き受けることができます。プロセスの開始時にはすべてのセカンダリクラスターがプライマリと完全に同期されているため、新しいプライマリはデータを失うことなく Amazon DocumentDB グローバルクラスターのオペレーションを継続できます。プライマリクラスターと選択したセカンダリクラスターが新しいロールを引き受ける間、短時間データベースが使用できなくなります。

アプリケーションの可用性を最適化するには、この機能を使用する前に次の操作を行います。

- この操作は、ピーク時以外の、プライマリクラスターへの書き込みが最小限である時間帯に実行してください。
- 各アプリケーションをオフラインにして、Amazon DocumentDB グローバルクラスターのプライマリクラスターへの書き込みが送信されないようにします。
- Amazon CloudWatch 上で `GlobalClusterReplicationLag` メトリクスを確認して、グローバルクラスター内のすべての Amazon DocumentDB セカンダリクラスターのラグタイムを確認します。このメトリクスは、セカンダリクラスターへのレプリケーションがプライマリクラスターに対してどのくらい遅れているかをミリ秒単位でを示します。この値は、Amazon DocumentDB がスイッチオーバーを完了するのにかかる時間と完全に比例します。したがって、遅延値が大きいほど、スイッチオーバーにかかる時間は長くなります。

  Amazon DocumentDB 向け CloudWatch メトリクスの詳細については、「Amazon DocumentDB のメトリクス」を参照してください。

スイッチオーバー中、選択したセカンダリ DB クラスターは、プライマリとして新しいロールに昇格されます。ただし、プライマリ DB クラスターのさまざまな設定オプションは継承されません。構成の不一致は、パフォーマンスの問題、ワークロードの非互換性、およびその他の異常な動作につながる可能性があります。このような問題を回避するには、以下の Amazon DocumentDB グローバルクラスター間の違いを解決します。

- 新しいプライマリ向けに Amazon DocumentDB のクラスターパラメータグループを構成する (必要な場合) — Amazon DocumentDB グローバルクラスター内の各クラスターごとに、Amazon DocumentDB クラスターパラメータグループを構成できます。つまり、セカンダリ DB クラスターを昇格してプライマリロールを引き継ぐ場合、セカンダリからのパラメータグループは、プライマリとは異なる設定になることがあります。その場合は、プロモートされたセカンダリ DB クラスターのパラメータグループを、プライマリクラスターの設定に適合するように変更します。この方法については、「Amazon DocumentDB クラスターパラメータグループの管理」を参照してください。
- 各種モニタリングツールやオプションを構成する (Amazon CloudWatch Events やアラームなど) - グローバルクラスターに必要なログ機能、アラームなどを使用して、昇格されたクラスターの設定を行います。パラメータグループと同様に、スイッチオーバープロセス中にこれらの機能の設定がプライマリから継承されることはありません。レプリケーションラグなどの一部の CloudWatch メトリクスは、プライマリリージョンのみで使用できます。そのため、スイッチオーバーによってメトリクスの表示方法やアラームの設定方法が変わり、定義済みのダッシュボードを変更する必要が生じる場合があります。詳細については、「Amazon DocumentDB のモニタリングと口グ記録」を参照してください。

> **Note**
>
> 通常、ロールスイッチオーバーには数分かかることがあります。

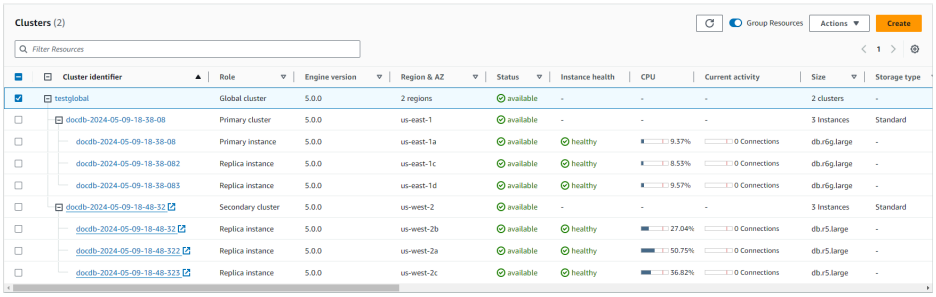
スイッチオーバープロセスが完了すると、昇格された Amazon DocumentDB クラスターはグローバルクラスターへの書き込みオペレーションを処理できるようになります。

AWS マネジメントコンソール または を使用してAmazon DocumentDB グローバルクラスターを切り替えることができます AWS CLI。

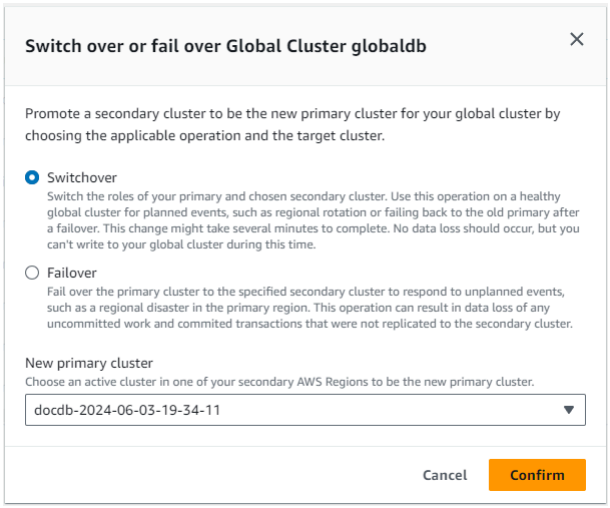
Using the AWS マネジメントコンソール

Amazon DocumentDB グローバルクラスターでスイッチオーバーを実行する

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択してください。
3. スイッチオーバーしたい Amazon DocumentDB グローバルクラスターを検索して選択します。

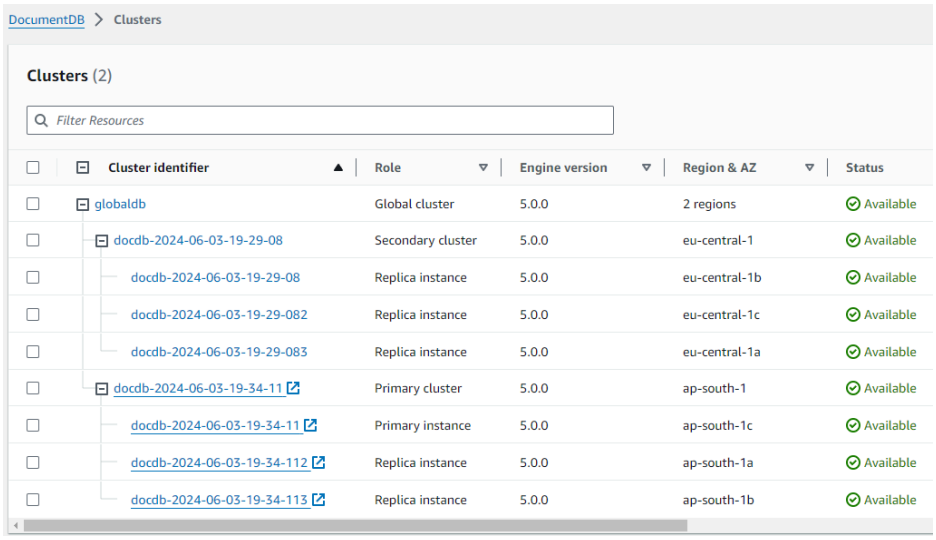


4. [アクション] メニューから [スイッチオーバーまたはフェイルオーバーする] を選択します。
5. 表示されるダイアログボックスで、[スイッチオーバー] を選択し、[新しいプライマリクラスター] フィールドのドロップダウンリストからセカンダリクラスターを選択します。



6. [確認] を選択します。

   プライマリクラスターのステータスが「スイッチオーバー中」に変わります。この状態になるまでには約 3 分かかります。この間、すべてのリージョンクラスターのステータスは「変更中」と表示されます。リージョンが同期され、新しいプライマリが昇格されると、すべてのステータスフィールドに「使用可能」が表示され、トランザクションを処理できるようになります。
7. 完了すると、元のプライマリクラスターはセカンダリクラスターになります。選択したセカンダリクラスターがプライマリクラスターになります。



Using the AWS CLI

Amazon DocumentDB グローバルクラスターでスイッチオーバーを実行する

`switchover-global-cluster` CLI コマンドを実行して、Amazon DocumentDB グローバルクラスターをスイッチオーバーします。コマンドを使用して、次のオプションの値を渡します。

- `--region`
- `--global-cluster-identifier`
- `--target-db-cluster-identifier`

次の例では*###############*をユーザー自身の情報で置き換えます。

Linux、macOS、Unix の場合:

```
aws docdb switchover-global-cluster \
   --region region_of_primary \
   --global-cluster-identifier global_cluster_id \
   --target-db-cluster-identifier arn_of_secondary_to_promote
```

Windows の場合:

```
aws docdb switchover-global-cluster ^
   --region region_of_primary ^
   --global-cluster-identifier global_cluster_id ^
   --target-db-cluster-identifier arn_of_secondary_to_promote
```

グローバルクラスターのスイッチオーバーまたはフェイルオーバーのブロック解除

異なるエンジンバージョンのリージョン

グローバルクラスターのすべてのリージョンクラスターが同じエンジンバージョンにあるわけではない場合、グローバルクラスターのスイッチオーバーとフェイルオーバーはブロックされます。バージョンが一致しない場合、スイッチオーバーまたはフェイルオーバーを呼び出すときにこのエラーが表示されることがあります。指定されたターゲット DB クラスターは、ソース DB クラスターとは異なるパッチレベルでエンジンバージョンを実行しています。グローバルクラスターを正常な状態に保つために、定期的に最新のエンジンバージョンを適用します。

このエラーを解決するには、まずすべてのセカンダリリージョンを更新し、次に保留中のメンテナンスアクション項目を適用してプライマリリージョンを同じエンジンバージョンに更新します。保留中のメンテナンスアクション項目を表示し、問題を修正するために必要な変更を適用するには、次のいずれかのタブの手順を実行します。

Using the AWS マネジメントコンソール

グローバルクラスターのスイッチオーバーまたはフェイルオーバーのブロックを解除するには、クラスターに対して保留中のメンテナンスアクションがあるかどうかを判断し、適用する必要があります。メンテナンスアクションを表示して適用するには、次の手順に従います。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択してください。
3. クラスター テーブルで、クラスター識別子 列でグローバルクラスターを見つけます。グローバルクラスターで、指定されたグローバルクラスターの各セカンダリクラスターとプライマリクラスターを書き留め、それぞれに対して次の手順を実行します。
4. 各セカンダリクラスターについて:
    a. クラスターで更新が使用可能な場合、[使用可能]、[必須]、または [次のウィンドウ] として メンテナンス 列に示されます。
    b. アクションを実行するには、クラスターを選択してその詳細を表示し、次に [メンテナンス & バックアップ] を選択します。[保留中のメンテナンス] 項目が表示されます。
    c. [説明] で、「新しいメンテナンス更新が利用可能」と示されている場合は、それを選択し、[今すぐ適用] を選択します。
5. プライマリクラスターの場合:

a. クラスターで更新が使用可能な場合、[使用可能]、[必須]、または [次のウィンドウ] として メンテナンス 列に示されます。

b. アクションを実行するには、クラスターを選択してその詳細を表示し、次に [メンテナンス & バックアップ] を選択します。[保留中のメンテナンス] 項目が表示されます。

c. [説明] で、「新しいメンテナンス更新が利用可能」と示されている場合は、それを選択し、[今すぐ適用] を選択します。

Using the AWS CLI

グローバルクラスターのスイッチオーバーまたはフェイルオーバーのブロックを解除するには、クラスターに対して保留中のメンテナンスアクションがあるかどうかを判断し、適用する必要があります。まずメンテナンスアクションをセカンダリクラスターで表示して適用し、次にグローバルクラスターのプライマリクラスターで適用するには、次の手順に従います。

1. まず各セカンダリリージョンのリージョンクラスターで以下を実行し、次にプライマリリージョンのリージョンクラスターで以下を実行します。

2. `--resource-identifier` オプションを使用して [describe-pending-maintenance-actions](#) CLI コマンドを実行し、Amazon DocumentDB リージョンクラスターで使用可能なメンテナンスアクションがあるかどうかを確認します。

   次の例では*###############*をユーザー自身の情報で置き換えます。

   Linux、macOS、Unix の場合:

   ```
   aws docdb describe-pending-maintenance-action \
      --resource-identifier arn:aws:rds:us-
   east-1:001234567890:cluster:docdb-2025-03-27-19-21-15
   ```

   Windows の場合:

   ```
   aws docdb describe-pending-maintenance-action ^
      --resource-identifier arn:aws:rds:us-
   east-1:001234567890:cluster:docdb-2025-03-27-19-21-15
   ```

   結果は以下のようになります。

   ```
   {
   ```

```
    "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. メンテナンスアクションが必要な場合は、次のオプションを使用して apply-pending-maintenance-action CLI コマンドを実行します。

   - --resource-identifier
   - --apply-action
   - --opt-in-type
   - --region

   次の例では*###############*をユーザー自身の情報で置き換えます。

   Linux、macOS、Unix の場合:

```
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
```

   Windows の場合:

```
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. メンテナンスアクションが完了したら、describe-pending-maintenance-actions コマンドを再度実行して、クラスターに対して保留中の他のアクションがないことを確認します。

   必要な結果は次のとおりです。

```
{
    "PendingMaintenanceActions": []
}
```

Using the Amazon DocumentDB API

グローバルクラスターのスイッチオーバーまたはフェイルオーバーのブロックを解除するには、クラスターに対して保留中のメンテナンスアクションがあるかどうかを判断し、適用する必要があります。次の API を使用して、メンテナンスアクションを表示および適用します。

1. まず各セカンダリリージョンのリージョンクラスターで以下を実行し、次にプライマリリージョンのリージョンクラスターで以下を実行します。
2. PendingMaintenanceAction API を呼び出して、Amazon DocumentDB グローバルクラスターで利用可能なメンテナンスアクションがあるかどうかを確認します。
3. ApplyPendingMaintenanceAction API を呼び出して、変更を適用します。

ターゲットクラスターで保留中の変更

スイッチオーバーとフェイルオーバーは、次のメンテナンスウィンドウにリクエストした変更やメンテナンスアクションなど、ターゲットセカンダリクラスターにまだ適用されていない変更がスケジュールされている場合にもブロックされます。この場合、スイッチオーバーまたはフェイルオーバーを呼び出すと、このエラーが表示されることがあります。ARN *arn:aws:rds:us-east-1:001234567890:cluster:docdb-2025-03-27-19-21-15* は、変更中または変更が保留中であるため、クラスターにフェイルオーバーすることはできません。クラスターが使用可能になったら、もう一度試してください。オペレーションのブロックを解除するには、ターゲットクラスターでスケジュールされた変更を削除します。ステップは変更のタイプによって異なります。スケ

ジュールされた変更が削除され、クラスターのステータスが に戻ったらavailable、スイッチオーバーまたはフェイルオーバーを再試行します。

スケジュールされたメンテナンスアクションをキャンセルする

からスケジュールされたメンテナンスアクションをキャンセル (オプトインを解除) することはできないため AWS マネジメントコンソール、 を使用します AWS CLI。

Using the AWS CLI

1. describedescribe-pending-maintenance-actions CLI コマンドを --resource-identifierオプションで実行し、 OptInStatus が であるメンテナンスアクションを探しますnext-maintenance。

   次の例では###############をユーザー自身の情報で置き換えます。

   Linux、macOS、Unix の場合:

   ```
   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
   ```

   Windows の場合:

   ```
   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
   ```

   結果は次のようになります。次のステップで使用するAction値 (os-upgradeこの例では) を書き留めます。

   ```
   {
       "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. スケジュールされたアクションをキャンセルするには、 で apply-pending-maintenance-action CLI コマンドを実行し--opt-in-type undo-opt-in、前のステップのAction値をに渡します--apply-action。

   Linux、macOS、Unix の場合:

```
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
```

   Windows の場合:

```
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
```

   レスポンスでは、メンテナンスアクションに がなくなりOptInStatus、スケジュールされたオプトインがキャンセルされたことが確認されます。

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

スケジュールされたクラスターの変更を元に戻す

Using the AWS CLI

1. describe-db-clusters CLI コマンドを実行し、PendingModifiedValues出力フィールドを調べて、スケジュールされた変更を確認します。

   > **Note**
   >
   > Amazon DocumentDB describe-db-clusters API はクラスターレベルのPendingModifiedValuesフィールドを返さないため、Amazon DocumentDB CLI (aws rds) ではなく、このコマンドに Amazon DocumentDB RDS CLI (aws docdb) を使用します。

   次の例では###############をユーザー自身の情報で置き換えます。

   Linux、macOS、Unix の場合:

   ```
   aws rds describe-db-clusters \
      --db-cluster-identifier docdb-2025-03-27-19-21-15 \
      --query 'DBClusters[0].PendingModifiedValues' \
      --region us-east-1
   ```

   Windows の場合:

   ```
   aws rds describe-db-clusters ^
      --db-cluster-identifier docdb-2025-03-27-19-21-15 ^
      --query "DBClusters[0].PendingModifiedValues" ^
      --region us-east-1
   ```

結果には、スケジュールされた変更が表示されます。この例では、クラスターの名前 (DBClusterIdentifier) の から docdb-2025-03-27-19-21-15への変更が保留中ですdocdb-new-name。

```
{
    "DBClusterIdentifier": "docdb-new-name"
}
```

2. で modify-db-cluster CLI コマンドを実行し--apply-immediately、変更された値を元の値に戻して変更を元に戻します。この例では、 をクラスターの現在の名前--new-db-cluster-identifierに戻すことで、保留中の名前の変更を元に戻します。

   Linux、macOS、Unix の場合:

```
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
```

   Windows の場合:

```
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
```

Amazon DocumentDB グローバルクラスターRPOs の管理

Amazon DocumentDB グローバルクラスターでは、 global_db_rpoパラメータを使用して目標復旧時点 (RPO) を管理できます。RPO (目標復旧時点) は、停止時に失われる可能性があるデータの最大量を表します。

Amazon DocumentDB グローバルクラスターの RPO を設定すると、Amazon DocumentDB はすべてのセカンダリクラスターの RPO ラグ時間をモニタリングします。このモニタリングにより、少なくとも 1 つのセカンダリクラスターがターゲット RPO ウィンドウ内に留まるようになります。

RPO 設定は、フェイルオーバーが発生した場合に潜在的なデータ損失を制限するために、Amazon DocumentDB がプライマリクラスターの書き込みトランザクションを管理する方法を制御します。Amazon DocumentDB は、プライマリでトランザクションをコミット (またはブロック) する RPO と RPO の遅延時間を次のように評価します。

- 少なくとも 1 つのセカンダリ DB クラスターの RPO ラグタイムが RPO よりも短い場合に、トランザクションをコミットします。
- すべてのセカンダリ DB クラスターの RPO ラグタイムが RPO よりも大きい場合、トランザクションをブロックします。

つまり、すべてのセカンダリクラスターがターゲット RPO の背後にある場合、Amazon DocumentDB はプライマリクラスターのトランザクションを一時停止します。Amazon DocumentDB は、少なくとも 1 つのセカンダリ DB クラスターの遅延時間が RPO を下回るとすぐに、一時停止したトランザクションを再開してコミットします。その結果、RPO (目標復旧時点) に達するまで、トランザクションはコミットできません。

`global_db_rpo` パラメータは動的です。遅延が十分に減少するまですべての書き込みトランザクションを停止させたくない場合は、すぐにリセットできます。この場合、Amazon DocumentDB は短い遅延後に変更を適用します。

> **Important**
>
> AWS リージョンが 2 つしかないグローバルデータベースでは、`global_db_rpo`パラメータのデフォルト値をセカンダリリージョンのパラメータグループに保持することをお勧めします。そうしないと、プライマリ AWS リージョンが失われたためにフェイルオーバーを実行すると、Amazon DocumentDB がトランザクションを一時停止する可能性があります。代わりに、Amazon DocumentDB が古い失敗した AWS リージョンでクラスターの再構築を完了するまで待ってから、このパラメータを変更して最大 RPO を適用します。

トピック

- 目標復旧時点の設定
- 目標復旧時点の表示
- 目標復旧時点の無効化

目標復旧時点の設定

global_db_rpo パラメータは、Amazon DocumentDB データベースの RPO 設定を制御します。有効な値の範囲は 20 秒から 2,147,483,647 秒 (68 年) です。ビジネスニーズを満たす現実的な値を選択します。例えば、RPO (目標復旧時点) に最大 10 分かかる場合があります。この場合、値を 600 に設定します。

Amazon DocumentDB グローバルクラスターのこの値は、 AWS マネジメントコンソール、、 AWS CLIまたは Amazon DocumentDB API を使用して設定できます。

Using the AWS マネジメントコンソール

RPO を設定するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. Amazon DocumentDB グローバルクラスターのプライマリクラスターを選択し、設定タブを開いて DB クラスターパラメータグループを見つけます。

   パラメータグループは直接編集できません。代わりに、以下を実行できます。

   - 適切なデフォルトパラメータグループをスタート点として使用して、カスタム DB クラスターのパラメータグループを作成します。
   - カスタム DB クラスターパラメータグループで、ユースケースに合わせて global_db_rpo パラメータの値を設定します。有効な値の範囲は、20 秒から最大整数値 2,147,483,647 (68 年) までです。
   - 変更された DB クラスターパラメータグループを Amazon DocumentDB DB クラスターに適用します。

DB クラスターパラメータグループの変更の詳細については、「」を参照してくださいAmazon DocumentDB クラスターパラメータグループを変更する。

Using the AWS CLI

global_db_rpo パラメータを設定するには、modify-db-cluster-parameter-group CLI コマンドを使用します。コマンドで、プライマリクラスターのパラメータグループの名前と RPO パラメータの値を指定します。

次の例では、my_custom_global_parameter_group という名前のプライマリ DB クラスターのパラメータグループの RPO を 600 秒 (10分) に設定します。

Linux、macOS、Unix の場合:

```
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"
```

Windows の場合:

```
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

`global_db_rpo` パラメータを変更するには、ModifyDBClusterParameterGroup API オペレーションを使用します。

目標復旧時点の表示

グローバルクラスターの目標復旧時点 (RPO) は、各 DB クラスターの `global_db_rpo`パラメータに保存されます。

CLI を使用して、Amazon DocumentDB DB クラスターの `global_db_rpo`パラメータを表示できます。`--query` オプションを使用して、`global_db_rpo`パラメータグループから パラメータのみを返します。

Linux、macOS、Unix の場合:

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name my_custom_global_parameter_group \
    --query "Parameters[?ParameterName=='global_db_rpo']"
```

Windows の場合:

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name my_custom_global_parameter_group ^
    --query "Parameters[?ParameterName=='global_db_rpo']"
```

このコマンドにより、以下のような出力が返されます。

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

クラスターパラメータグループのパラメータの表示の詳細については、「」を参照してくださいAmazon DocumentDB クラスターパラメータグループの管理。

目標復旧時点の無効化

RPO を無効にするには、global_db_rpo パラメータをリセットします。パラメータは、 AWS マネジメントコンソール、、 AWS CLIまたは Amazon DocumentDB API を使用してリセットできます。

Using the AWS マネジメントコンソール

RPO を無効にするには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。
3. リストで、プライマリ DB クラスターパラメータグループを選択します。
4. global_db_rpo パラメータの横にあるラジオボタンを選択します。
5. デフォルトにリセットを選択し、確認します。

コンソールでパラメータをリセットする方法の詳細については、「」を参照してくださいAmazon DocumentDB クラスターパラメータグループを変更する。

Using the AWS CLI

`global_db_rpo` パラメータをリセットするには、reset-db-cluster-parameter-group コマンドを使用します。

Linux、macOS、Unix の場合:

```
aws docdb reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my_custom_global_parameter_group \
    --parameters "ParameterName=global_db_rpo,ApplyMethod=immediate"
```

Windows の場合:

```
aws docdb reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my_custom_global_parameter_group ^
    --parameters "ParameterName=global_db_rpo,ApplyMethod=immediate"
```

Using the Amazon DocumentDB API

`global_db_rpo` パラメータをリセットするには、ResetDBClusterParameterGroup API オペレーションを使用します。

Amazon DocumentDB クラスターの管理

Amazon DocumentDB クラスターを管理するには、適切な Amazon DocumentDB コントロールプレーンアクセス許可を持つ IAM ポリシーが必要です。これらのアクセス許可により、クラスターとインスタンスを作成、変更、削除できます。`AmazonDocDBFullAccess` ポリシーは、 Amazon DocumentDB クラスターを管理するために必要なすべてのアクセス許可を提供します。

以下のトピックでは、Amazon DocumentDB クラスターを操作するときのさまざまなタスクの実行方法を示します。これには、クラスターの作成、削除、変更、接続、表示が含まれます。

トピック

- クラスターについて
- Amazon DocumentDB クラスター設定
- Amazon DocumentDB クラスターストレージ設定
- クラスターのステータスの評価
- Amazon DocumentDB クラスターのライフサイクル

- Amazon DocumentDB クラスターのスケーリング
- Amazon DocumentDB クラスターのボリュームのクローン作成
- Amazon DocumentDB クラスターの耐障害性について理解する

クラスターについて

Amazon DocumentDB は、コンピューティングとストレージを分離し、データレプリケーションとバックアップをクラスターボリュームにオフロードします。クラスターボリュームは、3 つのアベイラビリティーゾーンにわたって 6 つの方法でデータをレプリケートする、耐久性、信頼性、可用性に優れたストレージレイヤーを提供します。レプリカにより、データの可用性と読み取りスケーリングが高まります。各クラスターは、最大 15 個のレプリカにスケールアップできます。

名詞	説明	API オペレーション (動詞)

クラスター	1 つ以上のインスタンスと、これらのインスタンスのデータを管理する 1 つのクラスターストレージボリュームで構成されます。	`create-db-cluster`
		`delete-db-cluster`
		`describe-db-clusters`
		`modify-db-cluster`

インスタンス	クラスターストレージボリュームとのデータの読み取りと書き込みは、インスタンスを介して行われます。クラスターには、プライマリとレプリカの 2 種類のインスタンスがあります。クラスターには常に 1 つのプライマリインスタンスがあり、0 ~ 15 のレプリカを持つことができます。	`create-db-instance`
		`delete-db-instance`
		`describe-db-instances`
		`modify-db-instance`
		`describe-orderable-db-instance-options`
		`reboot-db-instance`

クラスターボリューム	3 つのアベイラビリティーゾーンにまたがる仮想データベースストレージボリュー	該当なし

名詞	説明	API オペレーション (動詞)
	ム。各アベイラビリティーゾーンにはクラスターデータの 2 つのコピーが格納されます。	
プライマリインスタンス	読み取りと書き込みの両方のオペレーションをサポートし、クラスターボリュームに対するすべてのデータ変更を実行します。各クラスターには 1 つのプライマリインスタンスがあります。	該当なし
レプリカインスタンス	読み取りオペレーションのみをサポートします。各 Amazon DocumentDB クラスターには、プライマリインスタンスに加えて 15 個までのレプリカインスタンスを含めることができます。複数のレプリカが読み取りワークロードを分散します。別のアベイラビリティーゾーンにレプリカを配置することによって、データベースの可用性を高めることもできます。	該当なし
クラスターエンドポイント	クラスターの現在のプライマリインスタンスに接続する Amazon DocumentDB クラスターのエンドポイント。Amazon DocumentDB クラスターごとに 1 つのクラスターエンドポイントと 1 つのプライマリインスタンスがあります。	該当なし

名詞	説明	API オペレーション (動詞)
リーダーエンドポイント	Amazon DocumentDB クラスターで利用できるレプリカの 1 つに接続する、そのクラスターのエンドポイント。各 Amazon DocumentDB クラスターには、リーダーエンドポイントがあります。複数のレプリカがある場合、リーダーエンドポイントは各接続リクエストを Amazon DocumentDB レプリカのいずれかにルーティングします。	該当なし
インスタンスエンドポイント	特定のインスタンスに接続する Amazon DocumentDB クラスターのインスタンス用のエンドポイント。クラスターの各インスタンスには、インスタンスタイプにかかわらず、独自のインスタンスエンドポイントがあります。	該当なし

Amazon DocumentDB クラスター設定

クラスターを作成または変更するときは、どのパラメータが変更不可能で、どのパラメータがクラスターの作成後に変更可能かを理解することが重要です。次の表に、クラスターに固有のすべての設定 (つまり、パラメータ) をリストします。表に示されているように、変更可能なものもあれば、そうでないものもあります。

> **Note**
>
> これらの設定は、Amazon DocumentDB クラスターのパラメータグループおよびそのパラメータと混同しないでください。クラスターのパラメータグループの詳細については、「Amazon DocumentDB クラスターパラメータグループの管理」を参照してください。

パラメータ	変更可能	コメント
DBClusterIdentifier	はい	命名に関する制約: • 1 – 63 個の英字、数字、またはハイフンで構成します。 • 1 字目は英字である必要があります。 • ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。 • Amazon RDS、Amazon Neptune、Amazon DocumentDB のすべてのクラスターで AWS アカウント、リージョンごとに一意である必要があります。
Engine	いいえ	docdb を指定してください。
BackupRetentionPeriod	はい	1 ~ 35 日を指定する必要があります。
DBClusterParameterGroupName	はい	命名に関する制約: • 長さは 1 ~ 255 バイト個の英数字です。 • 1 字目は英字である必要があります。 • ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。
DBSubnetGroupName	いいえ	クラスターが作成された後で、クラスターのサブネットを変更することはできません。
EngineVersion	いいえ	値は `5.0.0` (デフォルト)、`4.0.0`、または `3.6.0` です。
KmsKeyId	いいえ	クラスターの暗号化を選択した場合、クラスターの暗号化に使用した AWS KMS キーを変更することはできません。
MasterUsername	いいえ	クラスターが作成された後で、`MasterUsername` を変更することはできません。

パラメータ	変更可能	コメント
		命名に関する制約: • 長さは 1 ~ 63 バイトの英数字です。 • 1 字目は英字である必要があります。 • データベースエンジンの予約語は使用できません。

MasterUserPassword	はい	制約:
		• 長さは 8 ~ 100 バイトの印刷可能な ASCII 文字です。
		• 以下のものを除くすべての印刷可能な ASCII 文字を使用できます。
		  • **/** (スラッシュ)
		  • **"** (二重引用符)
		  • **@** (アットマーク)

Port	はい	ポート番号は、クラスター内のすべてのインスタンスに適用されます。
PreferredBackupWindow	はい	
PreferredMaintenanceWindow	はい	
StorageEncrypted	いいえ	クラスターの暗号化を選択した場合は、暗号化を解除することはできません。

パラメータ	変更可能	コメント
StorageType	はい	DB クラスターのストレージタイプ: 標準 (standard) または I/O 最適化 (iopt1)。 デフォルト: standard このパラメータは、CreateDBCluster と ModifyDBCluster で設定できます。 詳細については、「Amazon DocumentDB クラスターストレージ設定」を参照してください。
Tags	はい	
VpcSecurityGroupIds	いいえ	クラスターが作成された後で、クラスターが存在する VPC を変更することはできません。

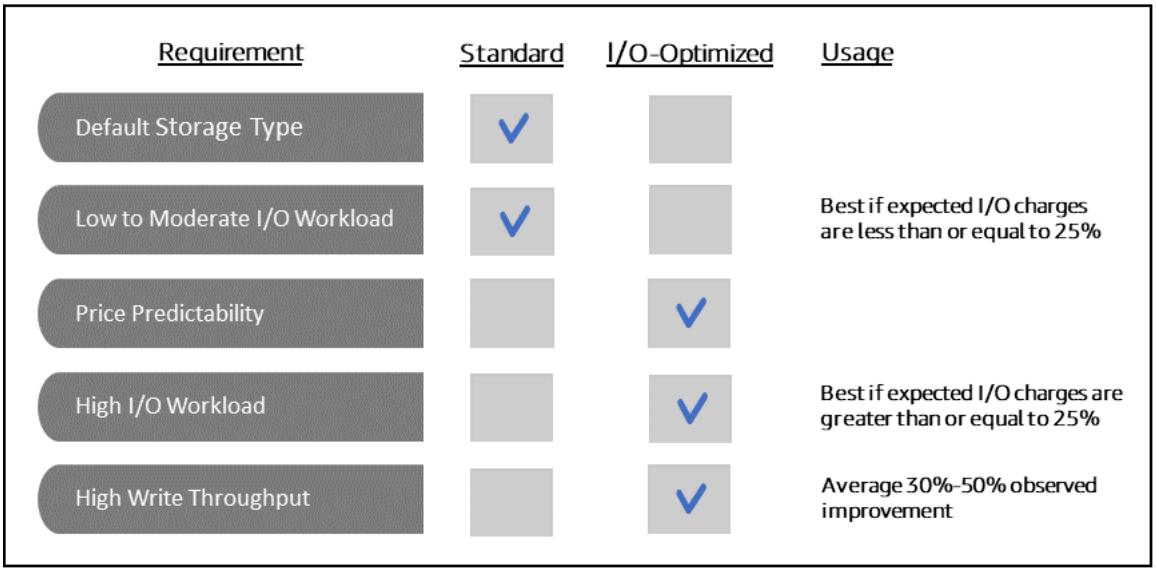
Amazon DocumentDB クラスターストレージ設定

Amazon DocumentDB 5.0 以降、インスタンスベースのクラスターは 2 つのストレージ設定タイプをサポートしています。

- Amazon DocumentDB 標準ストレージ: I/O 消費量が低から中レベルのお客様向けに設計されたオプションです。I/O コストが Amazon DocumentDB クラスター全体の 25% 未満になると予想される場合は、このタイプが適していると思われます。Amazon DocumentDB 標準ストレージ設定では、インスタンスとストレージの料金に加えて、リクエスト回数に応じて I/O 料金が請求されます。このため、請求額は使用量に応じてサイクルごとに異なります。設定は、お使いのアプリケーションの I/O 需要の変動に合わせて調整されます。
- Amazon DocumentDB I/O 最適化ストレージ : コストの予測可能性を優先するか、あるいは I/O 集約型アプリケーションを使用するお客様向けに設計されたオプションです。I/O 最適化設定は、I/O 集約型ワークロードを使用するお客様向けに、パフォーマンスの向上、スループットの向上、レイテンシーの低減を実現します。I/O コストが Amazon DocumentDB クラスターコスト全体の 25% を超えると予想される場合は、このオプションを選択いただくと価格パフォーマンスが向上します。Amazon DocumentDB I/O 最適化ストレージ設定では、I/O オペレーション回数に基づく従量課金は行われず、請求サイクルごとの請求額を確実に予想できます。この設定により、パフォーマンスを向上させながらコストを安定させることができます。

30 日に 1 回 のタイミングで、既存のデータベースクラスターを Amazon DocumentDB I/O 最適化ストレージに切り替えることができます。Amazon DocumentDB 標準ストレージにはタイミングを問わず随時切り替え可能です。ストレージ設定を I/O 最適化に変更する次の日付は、 `describe-db-clusters` コマンドを使用して、 AWS CLI またはクラスターの設定ページの AWS マネジメントコンソール を使用して追跡できます。

Amazon DocumentDB I/O 最適化設定を含む新しいデータベースクラスターを作成するか、あるいは AWS マネジメントコンソール 上での数回のクリック、もしくは AWS Command Line Interface (AWS CLI) での 1 つのパラメータの変更、または AWS SDK の使用により既存のデータベースクラスターを変換することができます。ストレージ設定の変更中、または変更後にダウンタイムは発生せず、インスタンスの再起動の必要もありません。

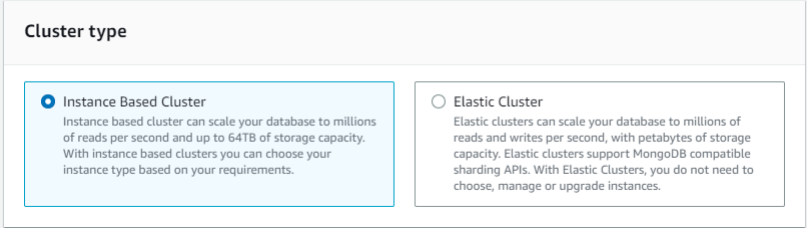


I/O 最適化クラスターの作成

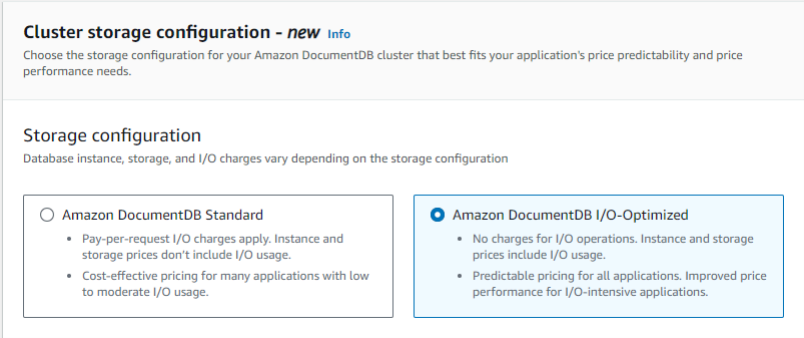
Using the AWS マネジメントコンソール

AWS マネジメントコンソールを使用して I/O 最適化クラスターを作成または変更するには:

1. Amazon DocumentDB 管理コンソール内の [クラスター] で、[作成] を選択するか、もしくは必要なクラスターを選択して [アクション] を選択し、[変更] を選択します。
2. 新しいクラスターを作成する場合は、[クラスタータイプ] セクションで [インスタンスベースのクラスター] (デフォルトのオプションです) を選択してください。



3. [設定] セクションの [クラスターストレージ設定] で、[Amazon DocumentDB I/O 最適化] を選択します。



4. クラスターの作成または変更を完了したら、[クラスターの作成] または [クラスターの変更] を選択します。

   クラスターの作成プロセスの詳細については、「を使用したクラスターとプライマリインスタンスの作成 AWS マネジメントコンソール」を参照してください。

   クラスターの変更プロセスの詳細については、「Amazon DocumentDB クラスターの変更」を参照してください。

Using the AWS CLI

AWS CLIを使用して I/O 最適化クラスターを作成するには:

次の例では###############をユーザー自身の情報で置き換えます。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

適切なストレージ設定を決定するためのコスト分析

Amazon DocumentDB では、データベースクラスターごとにストレージ設定を柔軟に選択できます。標準クラスターと I/O 最適化クラスターを適切に使い分けるため、Amazon DocumentDB のコストをクラスター単位で追跡できます。そのためには、既存のクラスターにタグを追加し、AWS Billing and Cost Management ダッシュボード で [コスト配分タグ付け] を有効にすることで、AWS Cost Explorer Service での個別のクラスターのコストを分析できます。コスト分析の詳細については、ブログ「コスト配分タグの使用」を参照してください。

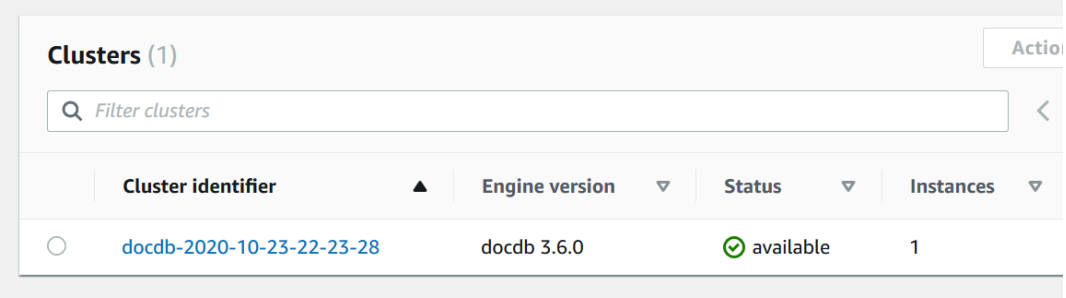
クラスターのステータスの評価

クラスターのステータスは、 AWS マネジメントコンソール または を使用して判断できます AWS CLI。

Using the AWS マネジメントコンソール

次の手順を使用して、 を使用して Amazon DocumentDB クラスターのステータスを確認します。 AWS マネジメントコンソール

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [クラスター ID] 列で、関心があるクラスターの名前を探します。次に、クラスターのステータスを確認するには、次に示すように行全体を参照して [ステータス] 列を見つけます。



Using the AWS CLI

`describe-db-clusters` オペレーション を使用して、 AWS CLIを使用した Amazon DocumentDB クラスターのステータスを確認します。

次のコードでは、クラスター `sample-cluster` のステータスを確認します。

Linux、macOS、Unix の場合:

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

Windows の場合:

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

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
[
    [
        "sample-cluster",
        "available"
    ]
]
```

Amazon DocumentDB クラスターのライフサイクル

Amazon DocumentDB クラスターのライフサイクルには、クラスターの作成、記述、変更、削除が含まれます。このセクションでは、これらのプロセスを完了する方法について説明します。

トピック

- Amazon DocumentDB クラスターの作成
- Amazon DocumentDB クラスター の説明
- Amazon DocumentDB クラスターの変更
- 実施待ちのメンテナンスの確認
- クラスターのエンジンバージョンへのパッチ更新の実行
- クラスターのオペレーティングシステムの更新
- Amazon DocumentDB クラスターの停止と開始
- Amazon DocumentDB クラスターの削除

Amazon DocumentDB クラスターの作成

Amazon DocumentDB クラスターは、インスタンスと、クラスターのデータを表すクラスターボリュームで構成されます。クラスターボリュームは、3 つのアベイラビリティーゾーン間で 1 つの仮想ボリュームとして 6 つの方法でレプリケートされます。クラスターには、プライマリインスタンスと、オプションで最大 15 個のレプリカインスタンスが含まれます。

以下のセクションでは、 AWS マネジメントコンソール または を使用して Amazon DocumentDB クラスターを作成する方法を示します AWS CLI。次に、そのクラスターのレプリカインスタンスを追加できます。コンソールを使用して Amazon DocumentDB クラスターを作成すると、プライマリインスタンスが自動的に作成されます。を使用して Amazon DocumentDB クラスター AWS CLI を作成する場合、クラスターのステータスが使用可能になったら、そのクラスターのプライマリインスタンスを作成する必要があります。

前提条件

Amazon DocumentDB クラスターを作成するための前提条件を以下に示します。

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://signin.aws.amazon.com/signup を開きますか?request_type=register。

2. オンラインの手順に従います。

   サインアップ手順の一環として、電話またはテキストメッセージを受け取り、電話キーパッドで検証コードを入力します。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザー が作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティベストプラクティスとして、ユーザーに管理アクセス権を割り当て、ルートユーザーアクセスが必要なタスクの実行にはルートユーザーのみを使用するようにしてください。

VPC 前提条件

Amazon DocumentDB クラスターは Amazon Virtual Private Cloud (Amazon VPC) でのみ作成できます。Amazon VPC には、Amazon DocumentDB クラスターで使用するために、少なくとも 2 つのアベイラビリティーゾーンのそれぞれに、1 つ以上のサブネットが必要です。アベイラビリティーゾーンにまたがってクラスターインスタンスを配信することで、ごく稀にアベイラビリティーゾーンに障害が発生した場合でも、クラスター内で使用できるインスタンスを確保できます。

サブネットの前提条件

Amazon DocumentDB クラスターを作成するときは、その VPC 内の VPC と対応するサブネットグループを選択して、クラスターを起動する必要があります。サブネットにより、アベイラビリティーゾーン と、そのアベイラビリティーゾーン内でインスタンスの起動に使用する IP 範囲が決まります。ここでは、サブネットとアベイラビリティーゾーンという用語を同じ意味で使用します。サブネットグループは、サブネット (またはアベイラビリティーゾーン) の名前付きセットです。サブネットグループでは、Amazon DocumentDB インスタンスの起動に使用するアベイラビリティーゾーンを指定できます。たとえば、3 つのインスタンスを含むクラスターでは、各インスタンスを別々のアベイラビリティーゾーンにプロビジョニングして高可用性を実現することをお勧めします。これにより、1 つのアベイラビリティーゾーンがダウンした場合、1 つのインスタンスにのみ影響します。

Amazon DocumentDB インスタンスは現在、最大 3 つのアベイラビリティーゾーンにプロビジョニングできます。サブネットグループに 3 つ以上のサブネットがある場合でも、それらのサブネットのうちの 3 つのみを使用して Amazon DocumentDB クラスターを作成できます。したがって、サブネットグループを作成するときは、インスタンスのデプロイ先として 3 つのサブネットのみを選択することをお勧めします。米国東部 (バージニア北部) では、サブネットグループに 6 つのサブネット (アベイラビリティーゾーン) を含めることができます。ただし、Amazon DocumentDB クラス

ターがプロビジョニングされると、Amazon DocumentDB によってインスタンスのプロビジョニングに使用される 3 つのアベイラビリティーゾーンが選択されます。

例えば、クラスターを作成する際、Amazon DocumentDB でアベイラビリティーゾーン (1A、1B、および 1C) を選択するとします。この場合、アベイラビリティーゾーン {1D} にインスタンスを作成しようとすると、その API コールは失敗します。ただし、特定のアベイラビリティーゾーンを指定せずにインスタンスを作成した場合、Amazon DocumentDB がお客様に代わってアベイラビリティーゾーンを選択します。Amazon DocumentDB では、複数のアベイラビリティーゾーン間でロードバランスのインスタンスの負荷を分散するアルゴリズムを使用して、高可用性が実現されます。たとえば、3 つのインスタンスがプロビジョニングされる場合、デフォルトでは 3 つのアベイラビリティーゾーンに別々にプロビジョニングされ、1 つのアベイラビリティーゾーンにまとめてプロビジョニングされません。

推奨事項:

- 特に理由がない限り、常に 3 つのサブネットを含むサブネットグループを作成します。その結果、3 つのアベイラビリティーゾーン間でインスタンスがプロビジョニングされるため、3 つ以上のインスタンスを含むクラスターで高可用性を実現できます。
- 高可用性を実現するには、常に複数のアベイラビリティーゾーンにインスタンスを分散させます。クラスターのすべてのインスタンスを 1 つのアベイラビリティーゾーンにまとめて配置しないでください。
- フェイルオーバーイベントはいつでも発生する可能性があるため、プライマリインスタンスまたはレプリカインスタンスが常に特定のアベイラビリティーゾーンに存在すると想定しないでください。

追加の前提条件

Amazon DocumentDB クラスターを作成するための追加の前提条件をいくつか次に示します。

- AWS Identity and Access Management (IAM) 認証情報 AWS を使用して に接続する場合、IAM アカウントには、Amazon DocumentDB オペレーションの実行に必要なアクセス許可を付与する IAM ポリシーが必要です。

  IAM アカウントを使用して Amazon DocumentDB コンソールにアクセスする場合は、まず IAM アカウント AWS マネジメントコンソール で にサインインする必要があります。Amazon DocumentDB コンソールを次の場所で開きます。https://console.aws.amazon.com/docdb
- クラスターの設定パラメータを調整する場合は、必須のパラメータ設定を定義したクラスターパラメータグループとパラメータグループを指定する必要があります。クラスターパラメータグループ

またはパラメータグループの作成または変更については、「Amazon DocumentDB クラスターパラメータグループの管理」を参照してください。

- クラスター用に指定する TCP/IP ポート番号を決定する必要があります。会社のファイアウォールによっては、Amazon DocumentDB のデフォルトポートへの接続がブロックされます。会社のファイアウォールがデフォルトのポートをブロックする場合は、お客様のクラスター用に別のポートを選択します。クラスターのすべてのインスタンスは同じポートを使用します。

を使用したクラスターとプライマリインスタンスの作成 AWS マネジメントコンソール

以下の手順では、コンソールを使用して 1 つまたは複数のインスタンスで Amazon DocumentDB クラスターを起動する方法について説明します。

クラスターの作成: デフォルト設定を使用した作成

を使用してデフォルト設定を使用してインスタンスを持つクラスターを作成するには AWS マネジメントコンソール

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. 米国東部 (バージニア北部) リージョン AWS リージョン 以外の でクラスターを作成する場合は、コンソールの右上セクションにあるリストからリージョンを選択します。
3. ナビゲーションペインで、[クラスター] を選択し、[作成] を選択します。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン
   > (≡) を選択します。

4. [Amazon DocumentDB クラスターの作成] ページで、[設定] ペインに入力します。

   a. クラスター識別子 – Amazon DocumentDB 指定の名前を使用するか、クラスターの名前 (**`sample-cluster`** など) を使用します。

      クラスターの名前付けの制約:

      - 長さは [1 ～ 63] 個の英字、数字、またはハイフンです。

- 1 字目は英字である必要があります。
- ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。
- Amazon RDS、Neptune、Amazon DocumentDB のすべてのクラスターで AWS アカウント、リージョンごとに一意である必要があります。

b. エンジンバージョン — デフォルトのエンジンバージョン 5.0.0 を受け入れるか、オプションで 8.0.0、4.0.0、または 3.6.0 を選択します。

c. [インスタンスクラス] - デフォルトの `db.r5.large` を使用するか、リストから必要なインスタンスクラスを選択します。

d. [インスタンス数] - リストで、このクラスターに作成するインスタンスの数を選択します。最初のインスタンスはプライマリインスタンスになり、他のインスタンスはすべて読み取り専用レプリカインスタンスになります。後でインスタンスを追加したり削除したりできます。デフォルトでは、Amazon DocumentDB クラスターは、3 つのインスタンス (1 つのプライマリと 2 つのレプリカ) 構成で起動します。

5. [クラスターストレージ設定] セクションの設定を完了します。

   [Amazon DocumentDB 標準] (デフォルト) または [Amazon DocumentDB I/O 最適化] を選択します。詳細については、「Amazon DocumentDB クラスターストレージ設定」を参照してください。

6. [認証] ペインに入力します。

   a. [ユーザー名] - プライマリユーザーの名前を入力します。クラスターにログインする際には、このプライマリユーザー名を使用する必要があります。

      プライマリユーザーの命名に関する制約事項:

      - 長さは 1 ~ 63 バイトの英数字です。
      - 1 字目は英字である必要があります。
      - データベースエンジンの予約語は使用できません。

   b. 以下のパスワードオプションのいずれかを選択します。

      - Managed in AWS Secrets Manager — プライマリユーザーのパスワードを自動的に管理する場合は AWS Secrets Manager 、このオプションを選択します。

        このオプションを選択した場合は、独自のものを作成するか、Secrets Manager が作成するキーを使用して KMS キーを設定します。

- [セルフマネージド] — プライマリユーザーのパスワードを自己管理する場合は、このオプションを選択します。このオプションを選択した場合は、プライマリユーザーのパスワードを入力して確認します。クラスターにログインする際には、このプライマリユーザーパスワードを使用する必要があります。

  パスワードの制約:

  - 長さは、[8 ~ 100] 個の印刷可能な ASCII 文字です。
  - 以下のものを除くすべての印刷可能な ASCII 文字を使用できます。
    - **/** (スラッシュ)
    - **"** (二重引用符)
    - **@** (アットマーク)

7. 画面の下部で、以下のいずれかを選択します。

   - ここでクラスターを作成するには、[クラスターの作成] を選択します。
   - クラスターを作成しない場合は、[Cancel (取り消し)] を選択します。
   - 作成前にクラスターをさらに設定するには、[Show additional configurations (追加設定の表示)] を選択し、「クラスターを作成する: その他の設定」に進みます。

     [追加設定] セクションの設定内容は次のとおりです。

     - [ネットワーク設定] - デフォルトでは、`default` VPC セキュリティグループを使用します。
     - [クラスターオプション] - デフォルトでは、ポート 27017 およびデフォルトのパラメータグループを使用します。
     - [暗号化] - デフォルトでは、(`default`) `aws/rds` キーを使用した暗号化が有効になります。

       > **Important**
       > クラスターはいったん暗号化されると、復号できません。

     - バックアップ - デフォルトでは、バックアップが 1 日間保持され、Amazon DocumentDB によってバックアップウィンドウが選択されます。
     - ログのエクスポート - デフォルトでは、監査ログは CloudWatch Logs にエクスポートされません。

- メンテナンス - デフォルトでは、Amazon DocumentDB によってメンテナンスウィンドウが選択されます。
- [削除保護] - クラスターが誤って削除されることから保護します。コンソールを使用して作成されたクラスターのデフォルトでは、有効 になっています。

デフォルト設定を使用しても、これらの設定のほとんどは、後でクラスターを修正して変更できます。

8. クラスターのセキュリティグループに対してインバウンド接続を有効にします。

クラスターのデフォルト設定を変更しなかった場合は、特定のリージョンのデフォルト VPC のデフォルトのセキュリティグループを使用してクラスターを作成しました。Amazon DocumentDB に接続するには、クラスターのセキュリティグループのポート 27017 (または選択したポート) でインバウンド接続を有効にする必要があります。

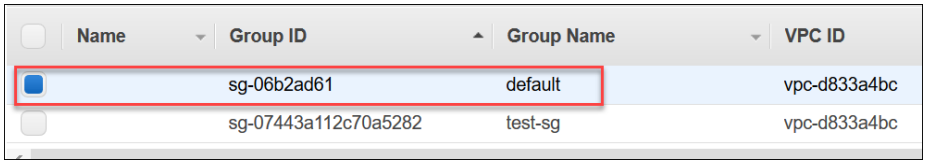
クラスターのセキュリティグループにインバウンド接続を追加するには

a. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/ec2/ で Amazon EC2 コンソールを開きます。

b. メインウィンドウの [Resources (リソース)] セクションで、[Security groups (セキュリティグループ)] を選択します。



c. セキュリティグループのリストから、クラスターの作成時に使用したセキュリティグループを探し (おそらくデフォルトのセキュリティグループです)、セキュリティグループの名前の左側にあるボックスを選択します。



d. [Actions (アクション)] メニューから、[Edit inbound rules (インバウンドルールを編集する)] を選択し、次にルールの制約を選択するか入力します。

i. タイプ - リストから、プロトコルを選択してネットワークトラフィックを開きます。

ii. [プロトコル] - リストから、プロトコルのタイプを選択します。

iii. [ポート範囲] - カスタムルールでは、ポート番号またはポート範囲を選択します。ポート番号またはポート範囲には、クラスターを作成したときに指定したポートが含まれていることを確認してください (デフォルト: 27017)。

iv. ソース - インスタンスに到達できるトラフィックを指定します。リストからトラフィックの送信元を選択します。[Custom (カスタム)] を選択する場合は、単一の IP アドレスまたは IP アドレス範囲を CIDR 表記で指定します (例: 203.0.113.5/32)。

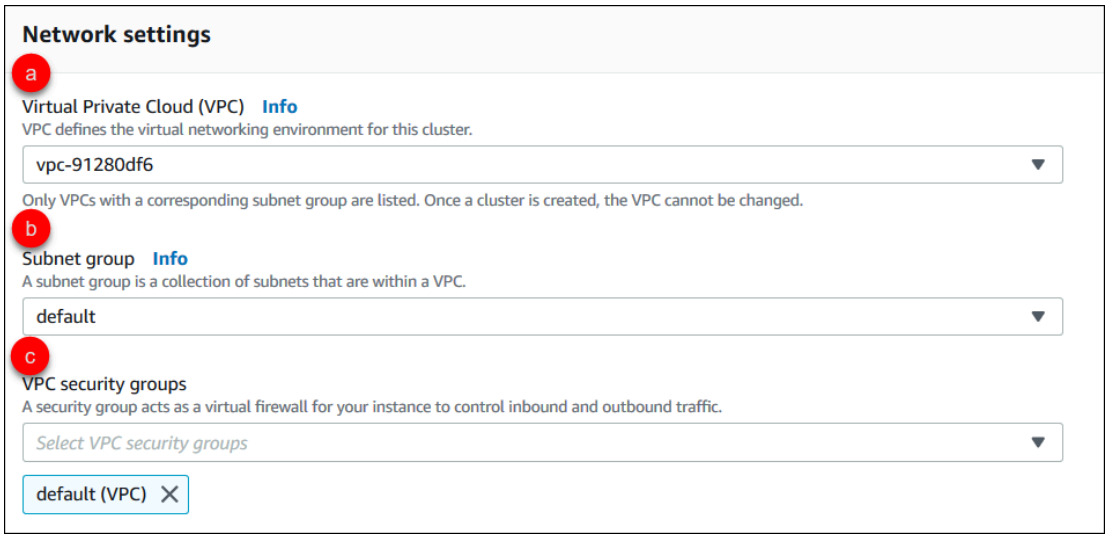
v. [Description] - ルールの説明を入力します。

vi. ルールの作成を終了したら、[Save (保存)] を選択します。

クラスターを作成する: その他の設定

クラスターのデフォルト設定を使用する場合は、以下のステップをスキップして [クラスターの作成] を選択します。

1. [Network settings (ネットワーク設定)] ウィンドウに入力します。



a. [Virtual Private Cloud (VPC)] - リストで、このクラスターを起動する Amazon VPC を選択します。

b. [サブネットグループ] - リストで、このクラスターに使用するサブネットグループを選択します。

c. [VPC セキュリティグループ] - リストで、このクラスターの VPC セキュリティグループを選択します。

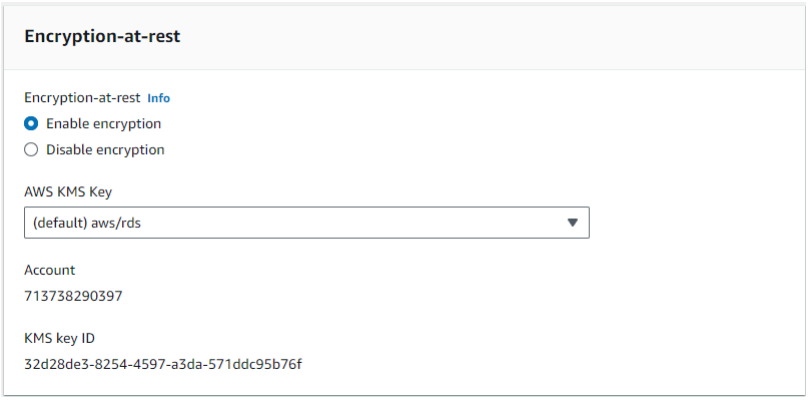
2. [Cluster options (クラスターオプション)] ペインに入力します。



   a. [データベースポート] - 上下の矢印を使用して、アプリケーションによってインスタンスへの接続に使用される TCP/IP ポートを設定します。

   b. [クラスターパラメータグループ] - パラメータグループのリストで、このクラスターのクラスターパラメータグループを選択します。

3. [Encryption (暗号化)] ペインに入力します。



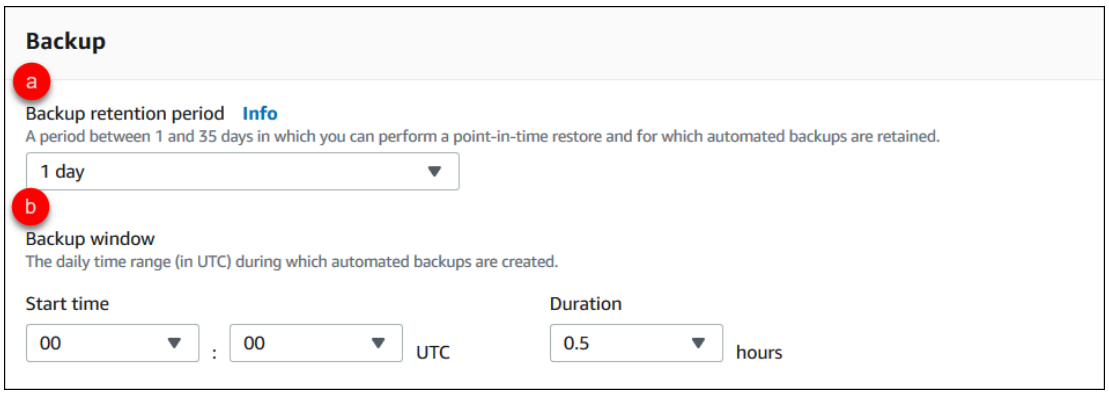
   a. [保管時に暗号化] - 以下のいずれかを選択します。

      - 暗号の有効化 - デフォルト すべてのデータは保管時に暗号化されます。データの暗号化を選択した場合、この操作を元に戻すことはできません。
      - [暗号化の無効化] - データは暗号化されません。

   b. AWS KMS キー — これは、データを暗号化している場合にのみ使用できます。リストで、このクラスターのデータの暗号化に使用するキーを選択します。デフォルトは `(default) aws/rds` です。

      [キーの ARN を入力] を選択した場合は、キーの Amazon リソースネーム (ARN) を入力する必要があります。

4. [バックアップ] ペインに入力します。



a. [バックアップ保持期間] - リストで、このクラスターの自動バックアップを削除するまで保持する日数を選択します。

b. [バックアップウィンドウ] - Amazon DocumentDB によってこのクラスターのバックアップが作成される毎日の時刻と期間を設定します。

   i. [開始時刻] - 最初のリストで、自動バックアップを開始する時間 (UTC 時) を選択します。2 番目のリストで、自動バックアップを開始する時間 (UTC 分) を選択します。

   ii. [期間] - リストで、自動バックアップの作成に割り当てる時間を選択します。

5. ログのエクスポート ペインで、CloudWatch Logs にエクスポートするログのタイプを選択して完了します。

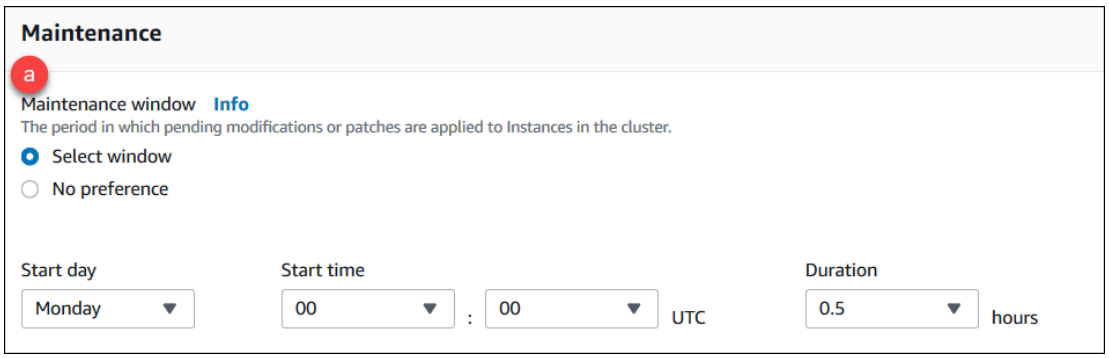


- [Audit logs (監査ログ)] - Amazon CloudWatch Logs にエクスポートするには、このオプションを選択します。[Audit logs (監査ログ)] を選択した場合は、クラスターのカスタムパラメータグループで `audit_logs` を有効にする必要があります。詳細については、「Amazon DocumentDB イベントの監査」を参照してください。
- [Profiler logs (プロファイラーログ)] - オペレーションのプロファイラーログの Amazon CloudWatch Logs へのエクスポートを有効にするには、このオプションを選択します。[Profiler logs (プロファイラーログ)] を選択した場合は、クラスターのカスタムパラメータグループの次のパラメータも変更する必要があります。
  - `profiler` — `enabled` に設定します。
  - `profiler_threshold_ms` の値に設定して、プロファイリングオペレーションのしきい値を `[0-INT_MAX]` に設定します。

- `profiler_sampling_rate` プロファイルする低速オペレーションの割合を設定するには、`[0.0-1.0]` の値に設定します。

  詳細については、「Amazon DocumentDB オペレーションのプロファイリング」を参照してください。

6. [メンテナンス] ペインに入力します。



- 以下のいずれかを選択します。

  - [Select window (ウィンドウの選択)] - Amazon DocumentDB によってクラスターでメンテナンスが実行される曜日、UTC 開始時刻、期間を指定できます。

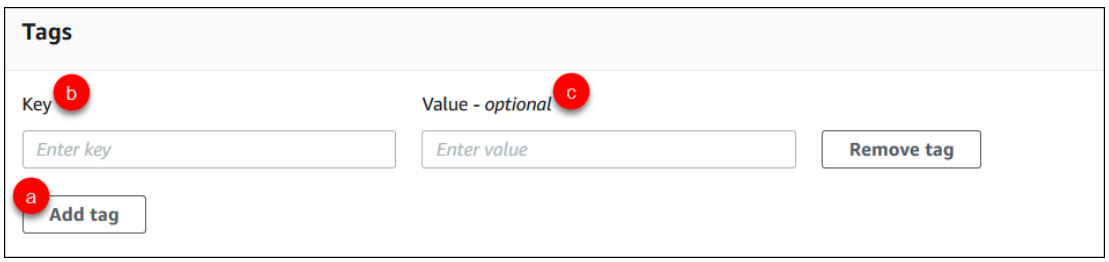
    a. [Start day (開始日)] - リストから、クラスターのメンテナンスを開始する曜日を選択します。

    b. [Start time (開始時刻)] リストで、メンテナンスを開始する時間と分 (UTC) を選択します。

    c. [Duration (期間)] - リストで、クラスターメンテナンスに割り当てる時間を選択します。指定した期間内にメンテナンスが完了できない場合、メンテナンスプロセスは指定した期間を過ぎて終了するまで続行されます。

  - [No preference (指定なし)] - Amazon DocumentDB によってメンテナンスが実行される曜日、開始時刻、期間が選択されます。

7. このクラスターに 1 つ以上のタグを追加する場合、[Tags (タグ)] ペインを完了します。



クラスターに追加するタグごとに、以下のステップを繰り返します。クラスターでは最大 10 まで保持できます。

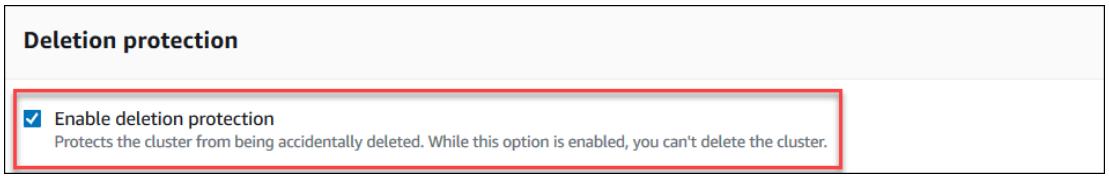
a. [Add tags (タグの追加)] を選択します。

b. タグの [キー] を入力します。

c. 必要に応じてタグの [値] を入力します。

タグを削除するには、[タグの削除] を選択します。

8. コンソールを使用してクラスターを作成する場合、[Deletion Protection (削除に対する保護)] がデフォルトで有効になっています。削除に対する保護を無効化するには、[Enable deletion protection (削除に対する保護の有効化)] のチェックを外します。有効にすると、削除に対する保護はクラスターが削除されることを防ぎます。削除に対して保護されているクラスターを削除するには、まずクラスターに変更を加えて削除に対する保護を無効化する必要があります。



削除に対する保護についての詳細は、「Amazon DocumentDB クラスターの削除」を参照してください。

9. クラスターを作成するには、[Create cluster (クラスターの作成)] を選択します。または、[キャンセル] を選択します。

を使用したクラスターの作成 AWS CLI

次の手順では、 を使用して Amazon DocumentDB クラスターを起動し AWS CLI 、Amazon DocumentDB レプリカを作成する方法について説明します。

パラメータ

- **`--db-cluster-identifier`** — 必須 このクラスターを識別する小文字の文字列。

クラスターの名前付けの制約:

- 長さは 1 – 63 個の英字、数字、またはハイフンで構成します。
- 1 字目は英字である必要があります。

  - ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。
  - AWS アカウントごと、リージョンごとに、すべてのクラスター (Amazon RDS、Amazon Neptune、Amazon DocumentDB 全体) で一意である必要があります。

- **--engine** — 必須 **docdb** を指定してください。
- **--deletion-protection | --no-deletion-protection** - オプション 削除に対する保護が有効にされている場合、これによってクラスターが削除されることを防ぎます。を使用する場合 AWS CLI、デフォルト設定では削除保護が無効になっています。

  削除に対する保護についての詳細は、「Amazon DocumentDB クラスターの削除」を参照してください。
- **--storage-type standard | iopt1** - オプションです。デフォルト: **standard**。クラスターのストレージ設定 有効な値は、`standard` (標準) または `iopt1` (I/O 最適化) です。
- **--master-username** — 必須 ユーザーの認証に使用されるユーザー名。

  マスターユーザーの名前付けの制約:

  - 長さは、[1 ~ 63] 個の英数字です。
  - 1 字目は英字である必要があります。
  - データベースエンジンの予約語は使用できません。
- **--master-user-password** - オプションです。ユーザーの認証に使用されるユーザーのパスワード。

  マスターパスワードの制約:

  - 長さは、[8 ~ 100] 個の印刷可能な ASCII 文字です。
  - 以下のものを除くすべての印刷可能な ASCII 文字を使用できます。
    - **/** (スラッシュ)
    - **"** (二重引用符)
    - **@** (アットマーク)
- **--manage-master-user-password** - オプション。Amazon DocumentDB はマスターユーザーパスワードを生成し、そのライフサイクル全体を通じて Secrets Manager で管理します。

その他のパラメータについては、「CreateDBCluster」を参照してください。

を使用して Amazon DocumentDB クラスターを起動するには AWS CLI

Amazon DocumentDB クラスターを作成するには、 を呼び出しますcreate-db-cluster AWS CLI。次の AWS CLI コマンドは、削除保護を有効にsample-clusterして という名前の Amazon DocumentDB クラスターを作成します。削除に対する保護についての詳細は、「Amazon DocumentDB クラスターの削除」を参照してください。

また、--engine-version は、デフォルトで最新のメジャーエンジンバージョンとなるオプションのパラメータです。現在のデフォルトのエンジンバージョンは 5.0.0 です (注: Amazon DocumentDB 8.0 は使用できますが、明示的に指定する必要があります)。新しいメジャーエンジンバージョンがリリースされると、--engine-version のデフォルトのエンジンバージョンが最新のメジャーエンジンバージョンに更新されます。その結果、本稼働ワークロード、特にスクリプト、オートメーション、または CloudFormation テンプレートに依存するワークロードでは、 を目的のメジャーバージョン--engine-versionに明示的に指定します。

> **Note**
>
> db-subnet-group-name または vpc-security-group-id が指定されていない場合、Amazon DocumentDB は、指定されたリージョンに対してデフォルトのサブネットグループと Amazon VPC セキュリティグループを使用します。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "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"
    }
}
```

クラスターが作成されるまでに数分かかります。 AWS マネジメントコンソール または を使用して AWS CLI 、クラスターのステータスをモニタリングできます。詳細については、「Amazon DocumentDB クラスターのステータスのモニタリング」を参照してください。

> **Important**
>
> を使用して Amazon DocumentDB クラスター AWS CLI を作成する場合、インスタンスは作成されません。したがって、プライマリインスタンスと必要なレプリカインスタンスを明示的に作成してください。コンソールまたは を使用してインスタンス AWS CLI を作成できます。詳細については、「クラスターへの Amazon DocumentDB インスタンスの追加」を参照してください。

詳細については、Amazon DocumentDB API リファレンス」のCreateDBCluster」を参照してください。

Amazon DocumentDB クラスター の説明

Amazon DocumentDB マネジメントコンソールまたは AWS CLI を使用して、Amazon DocumentDB クラスターに関連する接続エンドポイント、セキュリティグループ、VPCs、パラメータグループなどの詳細を表示できます。

詳細については次を参照してください:

- Amazon DocumentDB クラスターのステータスのモニタリング
- クラスターのエンドポイントの検索

Using the AWS マネジメントコンソール

次の手順を使用して、コンソールを使用して指定された Amazon DocumentDB クラスターの詳細を表示します。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. クラスターの一覧で、詳細を表示するクラスターの名前を選択します。クラスターに関する情報は、次のグループで分けられます。

   - [概要] - エンジンバージョン、クラスターのステータス、保留中のメンテナンス、パラメータグループのステータスなど、クラスターに関する一般情報。
   - [接続とセキュリティ] - [接続] セクションには、mongo シェルやアプリケーションを使用してこのクラスターに接続するための接続エンドポイントが一覧表示されます。[セキュリティグループ] セクションには、このクラスターに関連付けられているセキュリティグループと、その VPC ID や説明が一覧表示されます。
   - [設定] - [クラスターの詳細] セクションには、クラスターの Amazon リソースネーム (ARN)、エンドポイント、パラメータグループなど、クラスターに関する詳細が一覧表示されます。また、クラスターのバックアップ情報、メンテナンスの詳細、セキュリティとネットワークの設定も一覧表示されます。[クラスターのインスタンス] セクションには、このクラスターに属するすべてのインスタンスが一覧表示されます。各インスタンスのロールとクラスターパラメータグループのステータスも一緒に表示されます。
   - [モニタリング] — このクラスターの Amazon CloudWatch Logs メトリクス。詳細については、「Amazon DocumentDB と CloudWatch のモニタリング」を参照してください。
   - [イベントとタグ] — [最近のイベント] セクションには、このクラスターの最近のイベントの一覧が表示されます。Amazon DocumentDB は、クラスター、インスタンス、スナップショット、セキュリティグループ、およびクラスターパラメータグループに関連するイベントのレコードを保持します。この情報には、各イベントに関連する日付、時刻、およびメッセージが含まれます。[タグ] セクションには、このクラスターにアタッチされているタグが一覧表示されます。

Using the AWS CLI

を使用して Amazon DocumentDB クラスターの詳細を表示するには AWS CLI、次の例に示すように `describe-db-clusters` コマンドを使用します。詳細については、Amazon DocumentDB リソース管理 API リファレンス」の「DescribeDBClusters」を参照してください。

> **Note**
>
> クラスターやインスタンスのライフサイクル管理などの特定の管理機能において、Amazon DocumentDB は Amazon RDS と共有される運用テクノロジーを使用します。`filterName=engine,Values=docdb` フィルターパラメータは Amazon DocumentDB クラスターのみを返します。

Example

例 1: すべての Amazon DocumentDB クラスターを一覧表示する

次の AWS CLI コードは、リージョン内のすべての Amazon DocumentDB クラスターの詳細を一覧表示します。

```
aws docdb describe-db-clusters --filter Name=engine,Values=docdb
```

このオペレーションによる出力は、次のようになります。

```
{
    "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

例 2: 指定した Amazon DocumentDB クラスターのすべての詳細を一覧表示する

次の AWS CLI コードは、クラスター の詳細を一覧表示しますsample-cluster。

Linux、macOS、Unix の場合:

```
aws docdb describe-db-clusters \
   --filter Name=engine,Values=docdb \
   --db-cluster-identifier sample-cluster
```

Windows の場合:

```
aws docdb describe-db-clusters ^
   --filter Name=engine,Values=docdb ^
   --db-cluster-identifier sample-cluster
```

このオペレーションによる出力は、次のようになります。

```
{
    "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

例 3: Amazon DocumentDB クラスターの特定の詳細をリストする

を使用してクラスターの詳細のサブセットを一覧表示するには AWS CLI、`describe-db-clusters`オペレーションが一覧表示するクラスターメンバー`--query`を指定する を追加します。`--db-cluster-identifier` パラメータは、詳細を表示する対象のクラスターの識別子です。クエリの詳細については、「で出力をフィルタリングする方法 `--query` オプション の AWS Command Line Interface ユーザーガイド 」を参照してください。

次の例では、リージョン内のすべての Amazon DocumentDB クラスターを一覧表示します。

Linux、macOS、Unix の場合:

```
aws docdb describe-db-clusters \
    --filter Name=engine,Values=docdb \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterMembers]'
```

Windows の場合:

```
aws docdb describe-db-clusters ^
    --filter Name=engine,Values=docdb ^
    --db-cluster-identifier sample-cluster ^
    --query 'DBClusters[*].[DBClusterMembers]'
```

このオペレーションによる出力は、次のようになります。

```
[
    [
        [
            {
                "DBInstanceIdentifier": "sample-instance-1",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            },
            {
                "DBInstanceIdentifier": "sample-instance-2",
                "IsClusterWriter": false,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            }
        ]
    ]
]
```

Amazon DocumentDB クラスターの変更

クラスターを変更するには、クラスターが 使用可能 状態である必要があります。停止しているクラスターを変更することはできません。クラスターが停止している場合は、まずクラスターを起動し、クラスターが 使用可能 になるまで待ってから、必要な変更を加えます。詳細については、「Amazon DocumentDB クラスターの停止と開始」を参照してください。

Using the AWS マネジメントコンソール

コンソールを使用して特定の Amazon DocumentDB クラスターを変更するには、次の手順を使用します。

Amazon DocumentDB クラスターの変更

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. クラスター名の左側にあるボタンを選択して、変更するクラスターを指定します。
4. [アクション] を選択してから、[変更] を選択します。
5. [クラスターの変更: <cluster-name>] ペインで、必要な変更を行います。次の分野で変更を行うことができます。

   - [クラスターの仕様] - クラスターの名前、セキュリティグループ、および認証情報管理。
   - [クラスターストレージ設定] — クラスターのデータストレージモードです。標準設定と I/O 最適化設定のいずれかを選択します。
   - [クラスターオプション] -クラスターのポートおよびパラメータグループ。
   - [バックアップ] - クラスターの保持期間およびバックアップ保持期間。
   - [ログのエクスポート] - 監査ログまたはプロファイラーログのエクスポートを有効または無効にします。
   - [メンテナンス] - クラスターのメンテナンスウィンドウを設定します。
   - [削除保護] - クラスターで削除保護を有効または無効にします。削除保護は、デフォルトで有効になります。
6. 完了したら、[続行] を選択して変更の概要を表示します。
7. 変更に問題がなければ、[クラスターの変更] を選択してクラスターを変更できます。または、[戻る] を選択して変更を編集したり、[キャンセル] を選択して変更をキャンセルしたりすることもできます。

変更が適用されるまでに数分かかります。クラスターを使用できるのは、そのステータスが [available (使用可能)] であるときのみです。コンソールまたは AWS CLIを使用して、クラスター

のステータスをモニタリングできます。詳細については、「Amazon DocumentDB クラスターのステータスのモニタリング」を参照してください。

Using the AWS CLI

指定したクラスターを `modify-db-cluster` で変更するには、 AWS CLIオペレーションを使用します。詳細については、Amazon DocumentDB API リファレンス」のModifyDBCluster」を参照してください。

パラメータ

- **`--db-cluster-identifier`** — 必須 変更しようとしている Amazon DocumentDB クラスターの識別子。
- **`--backup-retention-period`** - オプション。自動バックアップを保管する日数。有効な値は、1 – 35 です。
- **`--storage-type`** - オプション。クラスターのストレージ設定 有効な値は、`standard` (標準) または `iopt1` (I/O 最適化) です。
- **`--db-cluster-parameter-group-name`** - オプション。クラスターに使用するクラスターパラメータグループの名前。
- **`--manage-master-user-password`** - オプション。Amazon DocumentDB はマスターユーザーパスワードを生成し、そのライフサイクル全体を通じて Secrets Manager で管理します。
- **`--rotate-master-user-password`** - オプションです。Secrets Manager は、既存のシークレットの新しいシークレットバージョンを生成します。新しいバージョンのシークレットには、新しいプライマリユーザーパスワードが含まれています。Amazon DocumentDB は、クラスターのプライマリユーザーパスワードを、新しいシークレットバージョンのパスワードと一致するように変更します。

  プライマリパスワードをローテーションするときは、**`--apply-immediately`** オプションを指定する必要があります。
- **`--master-user-password`** - オプションです。プライマリデータベースユーザー向け新しいパスワード。

  パスワードの制約:
  - 長さは 8 ~ 100 バイトの印刷可能な ASCII 文字です。
  - 以下のものを除くすべての印刷可能な ASCII 文字を使用できます。
    - **`/`** (スラッシュ)
    - **`"`** (二重引用符)

  - **@** (アットマーク)
- **--new-db-cluster-identifier** - オプション。クラスターの名前を変更する場合のクラスターの新しいクラスター識別子。この値は小文字で保存されます。

  命名に関する制約:

  - 1 – 63 個の英字、数字、またはハイフンで構成します。
  - 1 字目は英字である必要があります。
  - ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。
  - Amazon RDS、Amazon Neptune、Amazon DocumentDB のすべてのクラスターで AWS アカウント、リージョンごとに一意である必要があります。
- **--preferred-backup-window** - オプション。自動バックアップが作成される毎日の時間帯 (世界協定時刻 (UTC))。
  - 形式: `hh24:mm-hh24:mm`
- **--preferred-maintenance-window** - オプション。週 1 回のシステムメンテナンスを実行できる時間帯 (UTC)。
  - 形式: `ddd:hh24:mm-ddd:hh24:mm`
  - 有効な日: `Sun`、`Mon`、`Tue`、`Wed`、`Thu`、`Fri`、および `Sat`。
- **--deletion-protection** または **--no-deletion-protection** - オプション。このクラスターで削除に対する保護を有効にするかを指定します。削除保護では、クラスターを変更して削除保護を無効にするまで、クラスターが誤って削除されないようにします。詳細については、「Amazon DocumentDB クラスターの削除」を参照してください。
- **--apply-immediately** または **--no-apply-immediately** – 今すぐ変更を適用するには、`--apply-immediately` を使用します。クラスターの次のメンテナンスウィンドウ時に変更するには、`--no-apply-immediately` を使用します。

Example

次のコードは、クラスター `sample-cluster` のバックアップ保持期間を変更します。

Linux、macOS、Unix の場合:

```
aws docdb modify-db-cluster \
       --db-cluster-identifier sample-cluster \
       --apply-immediately \
       --backup-retention-period 7
```

Windows の場合:

```
aws docdb modify-db-cluster ^
       --db-cluster-identifier sample-cluster ^
       --apply-immediately ^
       --backup-retention-period 7
```

このオペレーションによる出力は、次のようになります。

```
{
    "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
    }
}
```

変更が適用されるまでに数分かかります。クラスターを使用できるのは、そのステータスが [available (使用可能)] であるときのみです。コンソールまたは AWS CLIを使用して、クラスターのステータスをモニタリングできます。詳細については、「Amazon DocumentDB クラスターのステータスのモニタリング」を参照してください。

実施待ちのメンテナンスの確認

ほとんどの場合、メンテナンスにはクラスター内の次のリソースの更新が含まれます。

- クラスター内のインスタンスのオペレーティングシステム (OS)
- クラスターのエンジンバージョン

Amazon DocumentDB のメンテナンスの詳細については、「」を参照してくださいAmazon DocumentDB のメンテナンス。

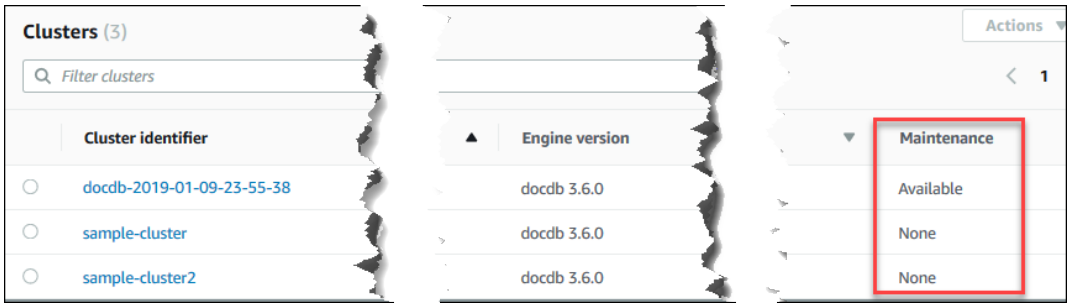
Using the AWS マネジメントコンソール

を使用して AWS マネジメントコンソール 、クラスターに保留中のメンテナンスがあるかどうかを判断できます。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

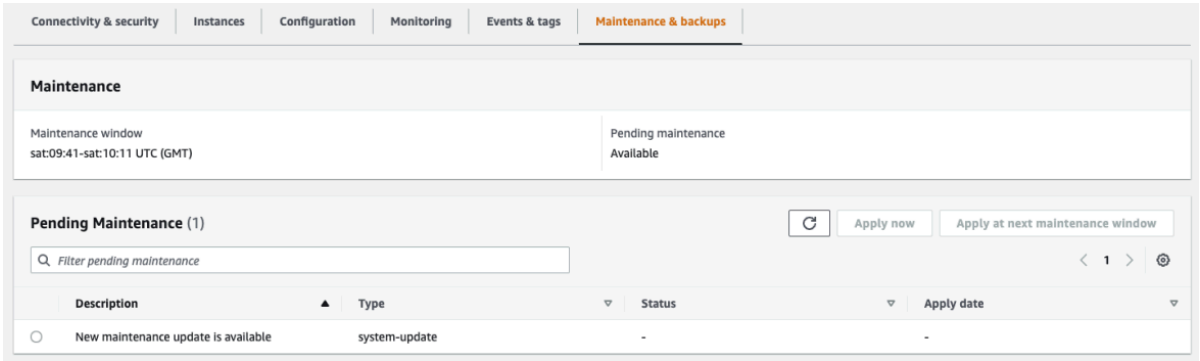
> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. [メンテナンス] 列を探して、クラスターに保留中のメンテナンスがあるかどうかを確認します。



None は、クラスターが最新のエンジンパッチバージョンを実行し、そのインスタンスに最新の OS アップデートがあることを示します。使用可能な は、クラスターに保留中のメンテナンスがあることを示します。これは、エンジンのアップグレードまたは OS 更新が必要であることを意味します。

4. クラスターに保留中のメンテナンスがある場合は、クラスター名を選択し、メンテナンスとバックアップタブを選択することで、保留中のメンテナンスアクションを表示できます。保留中のメンテナンスセクションには、保留中のメンテナンスアクションが一覧表示されます。



Using the AWS CLI

を使用して、クラスターに最新のエンジンバージョンと OS の更新があるかどうか AWS CLI を確認するには、次のパラメータを指定して `describe-pending-maintenance-actions`オペレーションを使用します。

パラメータ

- **`--resource-identifier`** - オプション。リソースの ARN (クラスター)。このパラメータを省略すると、すべてのクラスターの保留中のメンテナンスアクションがリストされます。
- **`--region`** - オプション。このオペレーションを実行する AWS リージョン (`us-east-1` など)。

Example

Linux、macOS、Unix の場合:

```
aws docdb describe-pending-maintenance-actions \
   --resource-identifier arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster \
   --region aa-example-1
```

Windows の場合:

```
aws docdb describe-pending-maintenance-actions ^
   --resource-identifier arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster ^
   --region aa-example-1
```

このオペレーションによる出力は、次のようになります。

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

Amazon DocumentDB クラスターには、次のメンテナンスアクションタイプが適用されます。

- `system-update` — Amazon DocumentDB クラスターのエンジンパッチをアップグレードします。

  クラスターで保留中のエンジンパッチアップグレードが利用可能な場合は、「」の手順に進みますクラスターのエンジンバージョンへのパッチ更新の実行。
- `os-upgrade` — ローリングアップグレードを使用して、Amazon DocumentDB クラスター内のすべての DB インスタンスのオペレーティングシステムを更新します。

  詳細については、「Amazon DocumentDB オペレーティングシステムの更新」を参照してください。

クラスターのエンジンバージョンへのパッチ更新の実行

このセクションでは、 AWS マネジメントコンソール または を使用してパッチ更新をデプロイする方法について説明します AWS CLI。パッチ更新とは、同じエンジンバージョン内のアップデートです (たとえば、3.6 エンジンバージョンを新しい 3.6 エンジンバージョンにアップデートするなど)。すぐにアップグレードするか、クラスターの次回のメンテナンス期間中にアップデートできます。エンジンのアップグレードが必要かどうかを確認するには、実施待ちのメンテナンスの確認 を参照してください。更新を適用すると、クラスターにダウンタイムが発生することに注意してください。

> **Note**
>
> 3.6 から 5.0 または 5.0 から 8.0 など、メジャーエンジンバージョンから別のエンジンバージョンにアップグレードする場合は、 Amazon DocumentDB のインプレースメジャーバー

ジョンアップグレードまたは を参照してくださいを使用した Amazon DocumentDB クラスターのアップグレード AWS Database Migration Service。

クラスターのエンジンバージョンの最新のパッチアップデートを取得するには、次の 2 つの構成要件があります。

- クラスターステータスは 使用可能 でなければなりません。
- クラスターが以前のエンジンバージョンで実行されている必要があります。

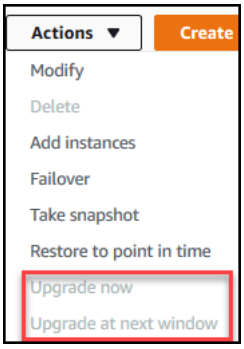
Using the AWS マネジメントコンソール

次の手順では、コンソールを使用して、パッチを適用し、クラスターのエンジンバージョンを最新のバージョンに更新します。すぐにアップグレードするか、クラスターの次のメンテナンスウィンドウ時に更新できます。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。クラスターのリストで、アップグレードするクラスターの左側にあるボタンを選択します。クラスターのステータスは 使用可能 でなければなりません。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. [アクション] メニューから、以下のいずれかのオプションを選択します。これらのメニューオプションは、選択したクラスターで最新のエンジンバージョンが実行されていない場合にのみ選択できます。



- [今すぐアップグレード] – 今すぐアップグレードプロセスが開始されます。最新のエンジンバージョンにアップグレードしている間、クラスターはしばらくオフラインになります。
- [次のウィンドウでアップグレード] - クラスターの次のメンテナンスウィンドウ時にアップグレードを開始します。クラスターは、最新のエンジンバージョンにアップグレードしている間、しばらくオフラインになります。

4. 確認ウィンドウが開いたら、次のいずれかを選択します。
   - [アップグレード] - 前のステップで選択したスケジュールに基づき、クラスターを最新のエンジンバージョンにアップグレードします。
   - [キャンセル] - クラスターのエンジンアップグレードをキャンセルし、クラスターの最新エンジンバージョンを引き続き使用します。

Using the AWS CLI

クラスターにパッチ更新を適用するには、 AWS CLI と `apply-pending-maintenance-action`オペレーションを使用し、次のパラメータを指定します。

パラメータ

- **`--resource-identifier`** — 必須 アップグレードしようとしている Amazon DocumentDB クラスターの ARN。
- **`--apply-action`** — 必須 `system-update` を使用して、エンジンパッチの更新をクラスターに適用します。
- **`--opt-in-type`** — 必須 以下の値を指定できます。
  - `immediate` -メンテナンスアクションをすぐに適用します。
  - `next-maintenance` - 次回のメンテナンス期間中にメンテナンスアクションを適用します。
  - `undo-opt-in` -既存の `next-maintenance` オプトインリクエストをキャンセルします。

Example

次の の例では、エンジンパッチの更新を に適用しますsample-cluster。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

このオペレーションの出力は、次のようになります。

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

クラスターのオペレーティングシステムの更新

クラスターレベルのos-upgradeメンテナンスアクションを使用して、クラスター内のすべてのインスタンスにオペレーティングシステム (OS) 更新を適用できます。Amazon DocumentDB は、ローリング方式でインスタンスを一度に数回更新し、フェイルオーバーを最小限に抑えるためにプライマリインスタンスを最後に更新します。更新は、クラスターのメンテナンスウィンドウ中に実行されます。インスタンスが OS 更新を受信すると、そのバッファキャッシュは空になり、そのインスタンスに対するクエリは、ワーキングセットがストレージボリュームから再入力されるまで、レイテンシーが長くなる可能性があります。

クラスターに保留中の OS 更新があるかどうかを確認するには、「」を参照してください実施待ちのメンテナンスの確認。コンソールや CLI の手順など、OS の更新の詳細については、「」を参照してくださいAmazon DocumentDB オペレーティングシステムの更新。

Amazon DocumentDB クラスターの停止と開始

Amazon DocumentDB クラスターの停止と開始は、開発とテスト環境のコストを管理するために便利です。Amazon DocumentDB を使用するたびにクラスターとインスタンスを作成したり削除する代わりに、インスタンスが必要ではないときにクラスター内ですべてのインスタンスを一時的に停止できます。テストを再開するときに、このインスタンスを再開できます。

トピック

- クラスターの停止と開始の概要
- 停止時のクラスターのみで実行できるオペレーション

クラスターの停止と開始の概要

Amazon DocumentDB クラスターが必要ではない期間は、そのクラスターですべてのインスタンスを一度に停止することができます。そして、クラスターを使用する必要がある時はいつでもクラスターを開始できます。開始と停止は、継続的な可用性を必要としない開発、テスト、または類似のアクティビティに使用されるクラスターのセットアップと解放のプロセスを簡素化します。クラスター内のインスタンスの数に関係なく、 AWS マネジメントコンソール または を使用して 1 つのアクション AWS CLI でクラスターを停止および開始できます。

クラスターが停止されている間、このクラスターのストレージボリュームは変更されません。指定された保持期間内でのストレージ、手動のスナップショット、および自動化されたバックアップストレージに対してのみ課金されます。インスタンス時間に対しては請求されません。Amazon DocumentDB は、7 日後に DB クラスターを自動的に開始するため、必要なメンテナンスの更新が遅

延することはありません。7 日後にクラスターが開始すると、このクラスター内のインスタンスに対する課金が再開します。クラスターが停止している間は、ストレージボリュームに対するクエリを実行できません。クエリを実行するにはインスタンスが使用可能状態であることが必要です。

Amazon DocumentDB クラスターが停止されている場合、クラスターもそのインスタンスも、いかなる方法でも変更できません。これには、インスタンスの追加や削除、クラスターの削除が含まれます。

Using the AWS マネジメントコンソール

次の手順では、利用可能な 1 つ以上のインスタンスがあるクラスターを停止したり、停止しているクラスターを開始する方法を示します。

Amazon DocumentDB クラスターを停止または開始するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

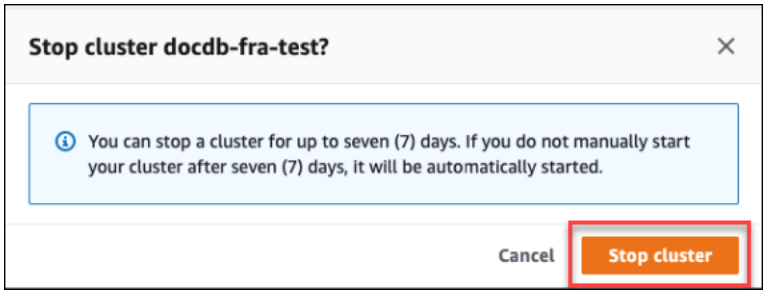
> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. クラスターのリストで、停止または開始するクラスター名の左側にあるボタンを選択します。
4. [Actions (アクション)] を選択したら、このクラスターで実行するアクションを選択します。
   - クラスターを停止したいときに、このクラスターが利用可能な場合:
     a. [停止] を選択します。

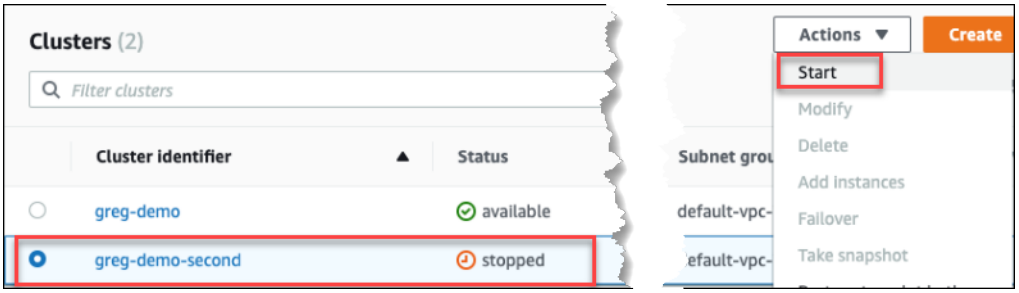


フェイルオーバーメカニズムが発動することを回避するため、停止オペレーションはまずレプリカインスタンスを停止してからプライマリインスタンスを停止します。

b. 確認ダイアログで、クラスターを停止するには [Stop cluster (クラスターの停止)]、あるいはクラスターを継続して実行するには [Cancel (キャンセル)] を選択して確認します。



- クラスターを開始したいときにこのクラスターが停止されている場合、[Start (開始)] を選択します。



5. クラスターおよびそのインスタンスの状態をモニタリングします。クラスターを開始すると、クラスターおよびそのインスタンスが利用可能な場合にこのクラスターの使用を再開できます。詳細については、「クラスターのステータスの評価」を参照してください。



Using the AWS CLI

次のコード例では、利用可能な状態にある 1 つ以上のインスタンスがあるクラスターを停止あるいは停止しているクラスターを開始する方法を示しています。

を使用して 1 つ以上の使用可能なインスタンスを持つクラスターを停止するには AWS CLI、stop-db-clusterオペレーションを使用します。停止しているクラスターを開始するには、 start-db-cluster オペレーションを使用します。どちらのオペレーションも --db-cluster-identifier パラメータを使用します。

パラメータ:

- **--db-cluster-identifier** — 必須 停止または開始するクラスターの名前。

Example— を使用してクラスターを停止するには AWS CLI

次のコードは、クラスター sample-cluster を停止します。クラスターには、利用可能な状態にある 1 つ以上のインスタンスがある必要があります。

Linux、macOS、Unix の場合:

```
aws docdb stop-db-cluster \
   --db-cluster-identifier sample-cluster
```

Windows の場合:

```
aws docdb stop-db-cluster ^
   --db-cluster-identifier sample-cluster
```

Example— を使用してクラスターを起動するには AWS CLI

次のコードは、クラスター sample-cluster を開始します。クラスターは現在停止している必要があります。

Linux、macOS、Unix の場合:

```
aws docdb start-db-cluster \
   --db-cluster-identifier sample-cluster
```

Windows の場合:

```
aws docdb start-db-cluster ^
   --db-cluster-identifier sample-cluster
```

停止時のクラスターのみで実行できるオペレーション

Amazon DocumentDB クラスター停止中に、指定された自動バックアップ保持期間内であれば、ポイントインタイムの復元を実行できます。ポイントインタイムの復元の実行の詳細については、「特定時点への復元」を参照してください。

クラスターを停止している間は、Amazon DocumentDB クラスターまたはその インスタンスの設定を変更することはできません。クラスターにインスタンスを追加または削除することも、関連するインスタンスがまだある場合はクラスターを削除することもできません。そのような管理アクションを実行する前に、クラスターを開始する必要があります。

Amazon DocumentDB は、スケジュールされたメンテナンスを、再び開始してからのみ停止したクラスターに適用します。7 日後、Amazon DocumentDB は停止したクラスターを自動的に開始するため、そのメンテナンス状態より過剰に遅れることはありません。クラスターが再開始すると、クラスター内のインスタンスへの課金が再開します。

クラスターが停止している間、Amazon DocumentDB は自動バックアップを一切実行せず、またバックアップ保持期間を延長しません。

Amazon DocumentDB クラスターの削除

Amazon DocumentDB クラスターは、 AWS マネジメントコンソール または を使用して削除できます AWS CLI。クラスターを削除するには、クラスターが 使用可能 状態であり、インスタンスが関連付けられていないことが必要です。クラスターが停止している場合は、まずクラスターを起動し、クラスターが 使用可能 になるまで待ってから、クラスターを削除します。詳細については、「Amazon DocumentDB クラスターの停止と開始」を参照してください。

削除保護

クラスターが誤って削除されることを防ぐために、削除保護 を有効にできます。コンソールを使用してクラスターを作成する場合は、デフォルトで削除保護が有効になっています。ただし、 AWS CLIを使用してクラスターを作成する場合、削除保護はデフォルトで無効になっています。

Amazon DocumentDB は、コンソールまたは AWS CLIを使用した削除オペレーションのどちらの実行でも、クラスターに対する削除保護を適用します。削除保護が有効になっている場合、クラスターを削除することはできません。削除に対する保護が有効化されているクラスターを削除するには、まずクラスターを変更して削除保護を無効化する必要があります。

クラスター上で削除保護を有効にしたコンソールを使用する場合、クラスターの最後のインスタンスを削除することはできません。これを削除するとクラスターも削除されるためです。 AWS CLIを使用すると、削除保護されているクラスターの最後のインスタンスを削除できます。ただし、クラス

ター自体は削除されず、データを保持されます。クラスターに新しいインスタンスを作成して、このデータにアクセスできます。削除保護の有効化および無効化の詳細については、以下を参照してください。

- Amazon DocumentDB クラスターの作成
- Amazon DocumentDB クラスターの変更

Using the AWS マネジメントコンソール

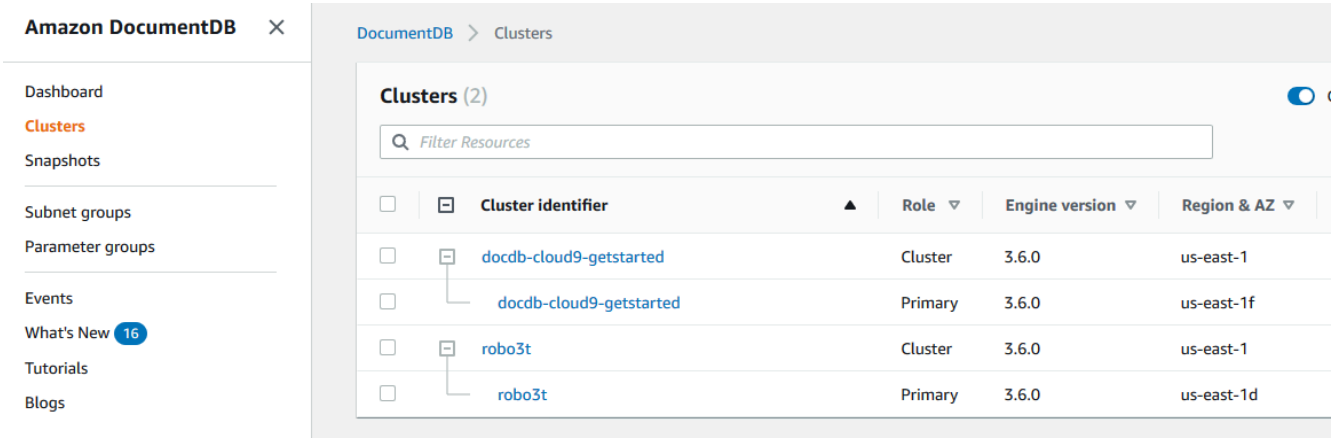
を使用してクラスターを削除するには AWS マネジメントコンソール、削除保護を無効にする必要があります。

クラスターで削除保護が有効になっているかどうかを確認するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

> **Tip**
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. Clusters ナビゲーションボックスで、[クラスター識別子] 列にはクラスターとインスタンスの両方が表示されることに注意してください。インスタンスは、次のスクリーンショットのように、クラスターの下に表示されます。



4. クラスターの名前を選択し、[設定] タブを選択します。[クラスターの詳細] セクションで、[削除保護] を見つけます。削除保護が有効化されている場合は、クラスターを変更して削除保護を無効にします。クラスターの変更については、「Amazon DocumentDB クラスターの変更」を参照してください。

   [Deletion protection (削除保護)] が無効化されると、クラスターを削除する準備が完了します。

クラスターを削除するには

1. ナビゲーションペインで [Clusters] (クラスター) を選択します。
2. 例えば、クラスターの下に表示されている [クラスター識別子] 列を確認して、クラスターにインスタンスがあるかどうかを判断します。クラスターを削除する前に、そのすべてのインスタンスを削除する必要があります。詳細については、「Amazon DocumentDB インスタンスの削除」を参照してください。
3. クラスターにインスタンスがあるかどうかに応じて、次のいずれかの手順を実行します。
   - クラスターにインスタンスがない場合は、クラスター名の左側にあるボタンを選択し、[アクション] を選択します。ドロップダウンメニューから [削除] を選択します。[Delete <cluster-name> (<cluster-name> を削除)] ダイアログボックスを完了したら、[Delete (削除)] を選択します。
   - クラスターに 1 つ以上のインスタンスがある場合は、次を行います。
     a. ナビゲーションペインで [Clusters] (クラスター) を選択します。
     b. クラスター名の左側にあるチェックボックスを選択して、クラスターの各インスタンスを削除します。[Actions (アクション)] を選択して、[Delete (削除)] を選択します。[Delete <cluster-name> (<cluster-name> を削除)] ダイアログボックスを完了したら、[Delete (削除)] を選択します。

        最後のインスタンスを削除すると、クラスターも削除されます。インスタンスの削除については、「Amazon DocumentDB インスタンスの削除」を参照してください。

クラスターが削除されるまでに数分かかります。クラスターのステータスをモニタリングするには、「Amazon DocumentDB クラスターのステータスのモニタリング」を参照してください。

Using the AWS CLI

インスタンスが関連付けられているクラスターを削除することはできません。クラスターに関連付けられているインスタンスを確認するには、describe-db-clusters コマンドを実行し、クラスターのすべてのインスタンスを削除します。次に、必要に応じてクラスターの削除保護を無効にし、最後にクラスターを削除します。

1. 最初に、クラスターのすべてのインスタンスを削除します。

   削除する必要があるインスタンスを決定するには、次のコマンドを実行します。

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

   このオペレーションによる出力は、次のようになります (JSON 形式)。

   ```
   [
       [
           "sample-cluster",
           [
               "sample-instance-1",
               "sample-instance-2"
           ]
       ]
   ]
   ```

   削除するクラスターにインスタンスがある場合は、次に示すように、そのインスタンスを削除します。

   ```
   aws docdb delete-db-instance \
       --db-instance-identifier sample-instance
   ```

2. 次に、削除保護を無効にします。

   AWS CLI を使用してクラスターのすべてのインスタンスを削除しても、クラスターは削除されません。クラスターも削除する必要があるときには、削除保護が無効化されている場合にのみこれを行うことができます。

クラスターに削除保護が有効化されているかどうかを判断するには、次のコマンドを実行します。

> **Tip**
>
> すべての Amazon DocumentDB クラスターの削除保護状態を確認するには、`--db-cluster-identifier` パラメータを省略します。

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

このオペレーションによる出力は、次のようになります。

```
[
    [
        "sample-cluster",
        "true"
    ]
]
```

削除保護が有効になっている場合、クラスターを変更して削除保護を無効にします。クラスターの削除保護を無効にするには、次のコマンドを実行します。

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --no-deletion-protection \
    --apply-immediately
```

3. 最後に、クラスターを削除します。

削除保護を無効にしたら、クラスターを削除する準備が整います。クラスターを削除するには、以下のパラメータを指定して `delete-db-cluster` オペレーションを使用します。

- **`--db-cluster-identifier`** — 必須 削除するクラスターの識別子。

- **--final-db-snapshot-identifier** - オプション。最終スナップショットを作成する場合は、最終スナップショットの名前を指定して、このパラメータを含める必要があります。--final-db-snapshot-identifier または --skip-final-snapshot のどちらかを含める必要があります。

  命名に関する制約:

  - 1 – 63 個の英字、数字、またはハイフンで構成します。
  - 1 字目は英字である必要があります。
  - ハイフンで終わる、または 2 つのハイフンを連続して含めることはできません。
  - Amazon RDS、Amazon Neptune、Amazon DocumentDB のすべてのクラスターで AWS アカウント、リージョンごとに一意である必要があります。
- **--skip-final-snapshot** - オプション。クラスターを削除する前に最終スナップショットを作成しない場合のみ、このパラメータを使用します。デフォルトでは、最終スナップショットを作成します。--final-db-snapshot-identifier または --skip-final-snapshot のどちらかを含める必要があります。

次の AWS CLI コードは、最終スナップショットsample-clusterを持つクラスターを削除します。クラスターに関連付けられるインスタンスがある場合、あるいは削除保護が有効化されている場合、このオペレーションは失敗します。

Example

Linux、macOS、Unix の場合:

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --final-db-snapshot-identifier sample-cluster-final-snapshot
```

Windows の場合:

```
aws docdb delete-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --final-db-snapshot-identifier sample-cluster-final-snapshot
```

Example

次の AWS CLI コードは、最終スナップショットを作成`sample-cluster`せずにクラスターを削除します。

Linux、macOS、Unix の場合:

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --skip-final-snapshot
```

Windows の場合:

```
aws docdb delete-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --skip-final-snapshot
```

`delete-db-cluster` オペレーションの出力は、削除するクラスターになります。

クラスターが削除されるまでに数分かかります。クラスターのステータスをモニタリングするには、「クラスターのステータスのモニタリング」を参照してください。

Amazon DocumentDB クラスターのスケーリング

Amazon DocumentDB では、ニーズに応じてクラスターのストレージとコンピューティングをスケールできます。このセクションでは、ストレージのスケーリング、インスタンスのスケーリング、および読み取りのスケーリングを使用して、Amazon DocumentDB クラスターとインスタンスのパフォーマンスとスケーリングを管理する方法について説明します。

トピック

- ストレージのスケーリング
- インスタンスのスケーリング
- 読み取りのスケーリング
- 書き込みスケーリング

ストレージのスケーリング

Amazon DocumentDB ストレージは、クラスターボリューム内のデータに合わせて自動的にスケーリングします。データが大きくなると、クラスターボリュームストレージは 10 GiB 単位で増加し、エンジンバージョン 8.0 以降では最大 256 TiB (以前のエンジンバージョンでは 128 TiB) まで増加します。

インスタンスのスケーリング

DB クラスター内の各 DB インスタンスの DB インスタンスクラスを変更することで、必要に応じて Amazon DocumentDB クラスターをスケーリングできます。Amazon DocumentDB では、Amazon DocumentDB 用に最適化されたいくつかのインスタンスクラスをサポートしています。

詳細については、「Amazon DocumentDB インスタンスの変更」を参照してください。

読み取りのスケーリング

Amazon DocumentDB クラスターの読み取りのスケーリングは、最大 15 個の Amazon DocumentDB レプリカをクラスター内に作成することで実現できます。各 Amazon DocumentDB レプリカは、最小限のレプリカラグでクラスターボリュームから同じデータを返します。通常、このラグはプライマリインスタンスが更新を書き込んだ後、100 ミリ秒を大幅に下回ります。読み取りトラフィックが増えたら、追加の Amazon DocumentDB レプリカを作成し、それらに直接接続することで DB クラスターの読み取りワークロードを分散できます。Amazon DocumentDB レプリカのインスタンスクラスは、プライマリイスタンスと同じものである必要はありません。

詳細については、「クラスターへの Amazon DocumentDB インスタンスの追加」を参照してください。

Amazon DocumentDB でスケールを読み取るには、レプリカセットとしてクラスターに接続し、ドライバーの組み込み読み取り設定機能を使用してレプリカインスタンスに読み取りを分散します。詳細については、「レプリカセットとしての Amazon DocumentDB に接続する」を参照してください。

書き込みスケーリング

クラスターのプライマリインスタンスのサイズを増やすことで、Amazon DocumentDB クラスターの書き込み容量をスケールできます。このセクションでは、ニーズに応じてクラスターのプライマリインスタンスをスケーリングする 2 つの方法を示します。最初のオプションは、アプリケーションへの影響を最小限に抑えることを目指しますが、完了するにはより多くのステップが必要です。2 番

目のオプションは、ステップ数が減るため、最適化で簡素化されますが、アプリケーションへの潜在的な影響が増えるというトレードオフを伴います。

アプリケーションに応じて、以下の最適なアプローチを選択できます。利用可能なインスタンスのサイズとコストの詳細については、「Amazon DocumentDB の料金」を参照してください。

1. 高可用性とパフォーマンスのための最適化 - レプリカセットモード でクラスターに接続している場合 (推奨) は、プライマリインスタンスのスケーリング時に、次のプロセスを使用してアプリケーションへの影響を最小限に抑えることができます。この方法では、クラスターが高可用性以上に維持され、読み取りスケーリングターゲットが (インプレースで更新されずに) インスタンスとしてクラスターに追加されるため、影響が最小限に抑えられます。

   a. より大きなインスタンスタイプの 1 つ以上のレプリカをクラスターに追加します (「???」を参照)。すべてのレプリカがプライマリと同じかそれ以上のインスタンスタイプであることを確認します。これにより、書き込みパフォーマンスが意図せずに低下して、より小さなインスタンスタイプにフェイルオーバーすることを回避できます。ほとんどの場合、これはクラスター内のインスタンスの数を一時的に 2 倍にし、スケーリングが完了した後に小さいレプリカを削除することを意味します。

   b. すべての新しいレプリカのフェイルオーバー層を優先度ゼロに設定し、より小さいインスタンスタイプのレプリカのフェイルオーバー優先度が最も高くなるようにします。詳細については、「???」を参照してください。

   c. 手動フェイルオーバーを開始し、新しいレプリカの 1 つをプライマリインスタンスに昇格させます。詳細については、「???」を参照してください。

      > **Note**
      >
      > これに伴って、クラスターのダウンタイムが最大 30 秒発生します。変化に応じて計画を立てましょう。

   d. 新しいプライマリより小さいインスタンスタイプのすべてのレプリカをクラスターから削除します。

   e. すべてのインスタンスのフェイルオーバー層の設定を同じ優先度に戻します (通常、これは設定を 1 に戻すことを意味します)。

   たとえば、クラスターに現在 3 つの `r5.large` インスタンス (1 つのプライマリと 2 つのレプリカ) が含まれていて、`r5.xlarge` インスタンスタイプにスケールするとします。これを行うには、まず 3 つの `r5.xlarge` レプリカインスタンスをクラスターに追加し、これらの新しい

r5.xlarge レプリカのフェイルオーバー層をゼロに設定します。次に、手動フェイルオーバーを開始します (アプリケーションのダウンタイムを最大 30秒見込みます)。フェイルオーバーが完了したら、3 つすべての r5.large インスタンスをクラスターから削除し、クラスターが r5.xlarge インスタンスにスケールされた状態にします。

コストを最適化しやすいように、Amazon DocumentDB インスタンスは秒単位で課金されます。インスタンスの作成、変更、削除などの課金対象のステータス変更に続く 10 分間の料金が最小の請求額となります。詳細については、ベストプラクティスのドキュメントの「コスト最適化」を参照してください。

2. 単純化のために最適化 — このアプローチは、シンプルさを最適化します。クラスターの拡張と縮小は行われませんが、読み取り容量が一時的に減少する可能性があります。

   レプリカのインスタンスクラスを変更すると、そのインスタンスが数秒から 30 秒未満の短い期間、リクエストを処理しなくなる可能性があります。レプリカセットモード(推奨) でクラスターに接続している場合、スケーリング操作中に読み取りキャパシティが 1 つのレプリカ (例えば、3 ノードクラスタでは 66% の容量、4 ノードクラスタで 75% の容量など) が減少します。

   a. クラスター内の 1 つのレプリカインスタンスをスケールします。詳細については、「インスタンスクラスの管理」を参照してください。

   b. インスタンスが利用可能になるまで待ちます (「Amazon DocumentDB インスタンスのステータスのモニタリング」を参照)。

      > **Note**
      >
      > これに伴って、クラスターのダウンタイムが最大 30 秒発生します。変化に応じて計画を立てましょう。

   c. すべてのレプリカインスタンスが 1 つずつスケールされるまで、手順 1 と 2 を実行します。

   d. 手動フェイルオーバーを開始します。これにより、レプリカがプライマリインスタンスに昇格します。詳細については、「Amazon DocumentDB フェイルオーバー」を参照してください。

      > **Note**
      >
      > これにより、クラスターのダウンタイムが最大 30 秒発生しますが、それよりも時間が短くなります。変化に応じて計画を立てましょう。

e. 前のプライマリ (現在はレプリカ) インスタンスをスケールします。

Amazon DocumentDB クラスターのボリュームのクローン作成

Amazon DocumentDB クローン作成を使用すると、同じ Amazon DocumentDB クラスターボリュームを使用して、元のクラスターと同じデータを持つ新しいクラスターを作成できます。このプロセスは、高速で、費用効果が高いように設計されています。関連付けられたデータボリュームを持つ新しいクラスターは、クローン と呼ばれます。クローンの作成は、スナップショットの復元など、他の手法を使用してデータを物理的にコピーするよりも、高速かつスペース効率に優れています。

Amazon DocumentDB は、プロビジョニングされた Amazon DocumentDB クラスターからの Amazon DocumentDB プロビジョニングクローンの作成をサポートしています。作成元とは異なるデプロイ設定を使用してクローンを作成すると、作成元の Amazon DocumentDB エンジンの最新バージョンを使用してクローンが作成されます。

Amazon DocumentDB クラスターからクローンを作成すると、ソース Amazon DocumentDB クラスターを所有するのと同じアカウントである AWS アカウントにクローンが作成されます。

トピック

- Amazon DocumentDB のクローン作成の概要
- Amazon DocumentDB クローン作成の使用に関する注意事項
- Amazon DocumentDB のクローン作成の仕組み
- Amazon DocumentDB のクローンの作成

Amazon DocumentDB のクローン作成の概要

Amazon DocumentDB では、クローン作成に、コピーオンライトプロトコル が使用されます。このメカニズムでは、初期クローンを作成するために使用する追加領域は最小限です。クローンが初期に作成されると、Amazon DocumentDB は、作成元 DB クラスターと新しい (クローンの) Amazon DocumentDB クラスターで使用されるデータのコピーを 1 つだけ保持します。追加のストレージは、作成元の Amazon DocumentDB クラスターまたは Amazon DocumentDB クラスターのクローンが (Amazon DocumentDB ストレージボリューム上の) データに変更を加えた場合にのみ割り当てられます。コピーオンライトプロトコルの詳細については、「Amazon DocumentDB のクローン作成の仕組み」を参照してください。

Amazon DocumentDB のクローン作成は、データを破損の危険にさらすことなく、本番データを使用してテスト環境を迅速にセットアップする場合に特に役立ちます。クローンは、次のようなさまざまなタイプのアプリケーションに使用できます。

- 潜在的な変更 (スキーマの変更やパラメータグループの変更など) を試して、すべての影響を評価する。
- データのエクスポートや分析クエリの実行など、大量のワークロードを扱うオペレーションをクローン上で実行する。
- 開発、テスト、またはその他の目的のために、本番 DB クラスターのコピーを作成する。

同じ Amazon DocumentDB クラスターから複数のクローンを作成できます。また、別のクローンから複数のクローンを作成することもできます。

Amazon DocumentDB のクローンを作成したら、Amazon DocumentDB インスタンスの設定を作成元の Amazon DocumentDB クラスターとは異なる方法で行えます。例えば、開発用途のクローンは、作成元の Amazon DocumentDB クラスターと同じ高可用性要件を満たす必要がない場合があります。この場合、Amazon DocumentDB クラスターで使用される複数の DB インスタンスではなく、単一の Amazon DocumentDB インスタンスを使用するようにクローンを設定できます。

テスト、開発などの用途へのクローンの使用が終了したら、クローンを削除できます。

Amazon DocumentDB クローン作成の使用に関する注意事項

Amazon DocumentDB のクローン作成には、現在、次の制約事項があります。

- AWS リージョンで許可される DB クラスターの最大数まで、必要な数のクローンを作成できます。ただし、クローンを 15 個作成すると、次のクローンはフルコピーになります。クローン作成オペレーションは、ポイントインタイムリカバリと同様に機能します。
- ソース Amazon DocumentDB クラスターとは異なる AWS リージョンにクローンを作成することはできません。
- DB インスタンスを持たない Amazon DocumentDB クラスターからクローンを作成することはできません。少なくとも 1 つの DB インスタンスを持つ Amazon DocumentDB クラスターのクローン作成のみが可能です。
- クローンは、Amazon DocumentDB クラスターとは異なる仮想プライベートクラウド (VPC) で作成できます。その場合、VPCのサブネットは同じアベイラビリティーゾーンにマッピングする必要があります。

Amazon DocumentDB のクローン作成の仕組み

Amazon DocumentDB のクローン作成は Amazon DocumentDB クラスターのストレージレイヤーで動作します。これには、コピーオンライト プロトコルが使用されます。これは、Amazon DocumentDB ストレージボリュームをサポートする基盤となる耐久性の高いメディアという点で、高速かつスペース効率に優れています。Amazon DocumentDB クラスターボリュームの詳細については、「Amazon DocumentDB クラスターの管理」を参照してください。

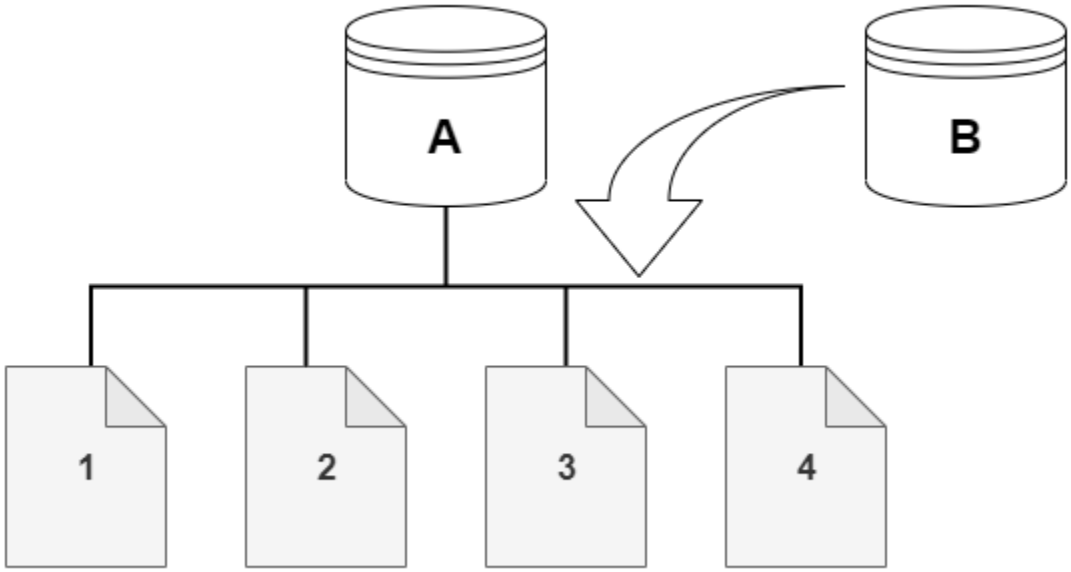
トピック

- コピーオンライトプロトコルの理解
- ソースクラスターボリュームの削除

コピーオンライトプロトコルの理解

Amazon DocumentDB クラスターでは、基盤となる Amazon DocumentDB ストレージボリュームのページにデータが格納されます。

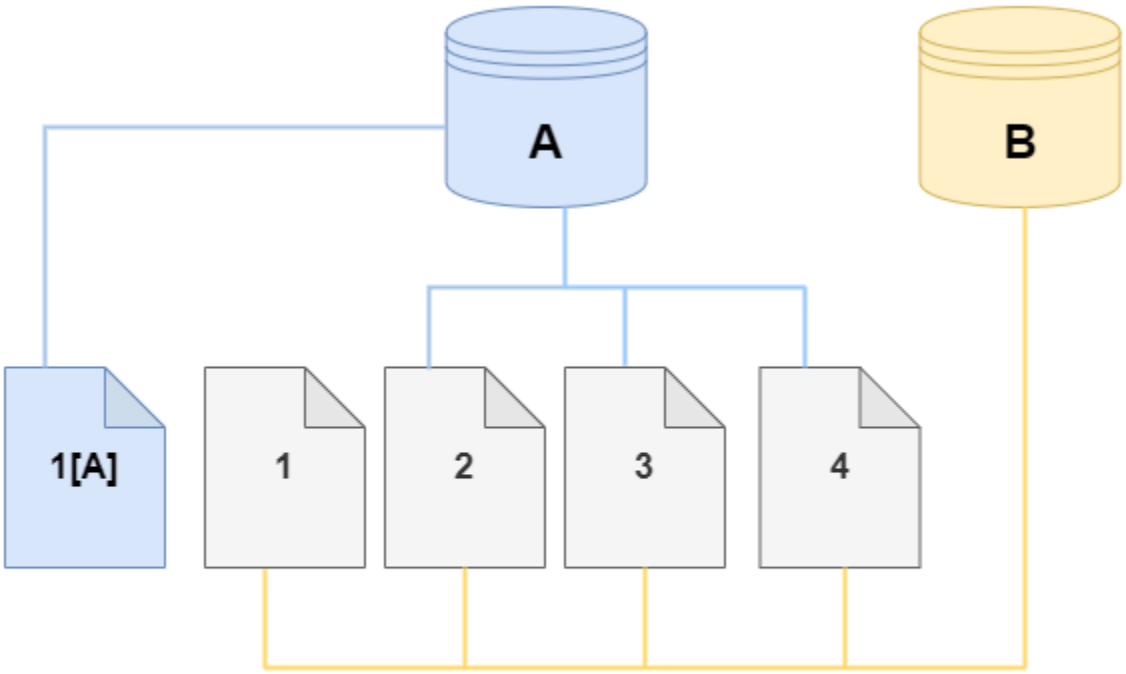
例えば、次の図には、4 つのデータページ 1、2、3、4 を持つ Amazon DocumentDB クラスター (A) があります。クローン B が Amazon DocumentDB クラスターから作成されたとします。クローンが作成されても、データはコピーされません。クローンは、作成元の Amazon DocumentDB クラスターと同じページのセットを参照しています。



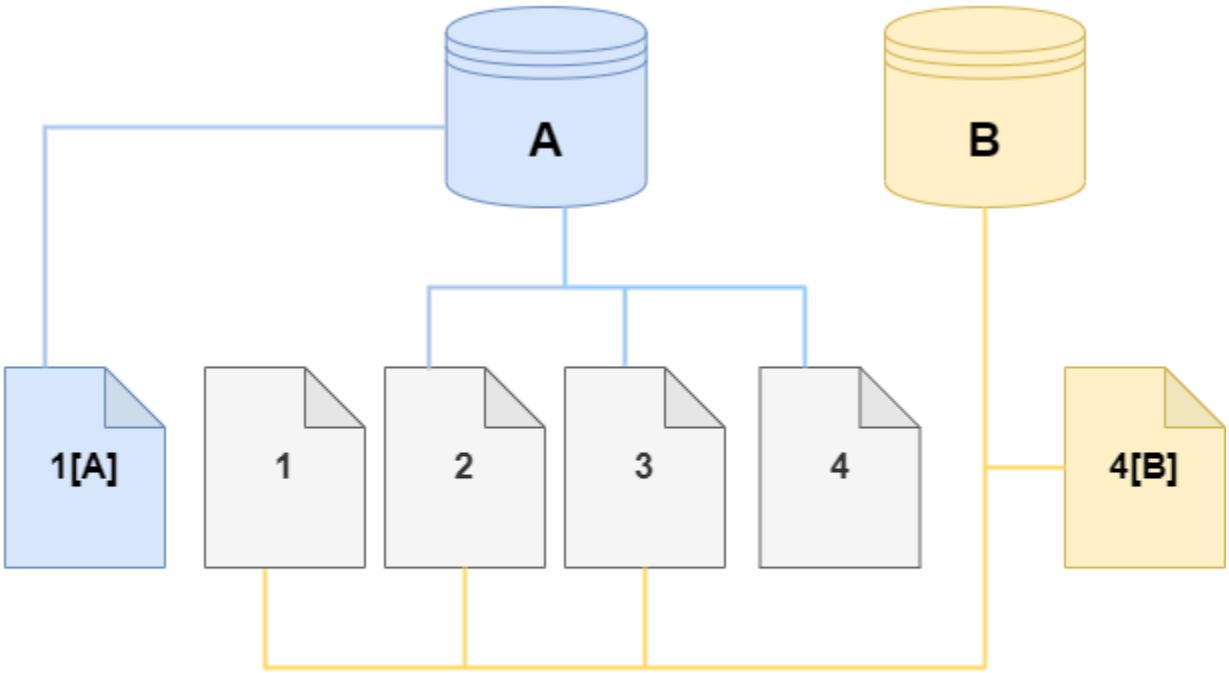
クローンが作成されたとき、通常は追加のストレージは必要ありません。コピーオンライトプロトコルでは、ソースセグメントと同じ物理ストレージメディア上のセグメントを使用します。追加のスト

レージが必要になるのは、ソースセグメントの容量がクローンセグメント全体に対して十分でない場合のみです。この場合、ソースセグメントは別の物理デバイスにコピーされます。

次の図に、前述と同様にクラスター A とそのクローン B を使用して動作中のコピーオンライトプロトコルの例を示します。Amazon DocumentDB クラスター (A) に変更を加えて、ページ 1 に保持されているデータが変更されたとします。元のページ 1 に書き込む代わりに、Amazon DocumentDB は新しいページ 1[A] を作成します。クラスター (A) の Amazon DocumentDB クラスターボリュームは、1[A]、2、3、4 ページを参照していますが、クローン (B) は引き続き元のページを参照しています。



クローンでは、ストレージボリュームのページ 4 に変更が加えられています。元のページ 4 に書き込む代わりに、Amazon DocumentDB は新しいページ 4[B] を作成します。クローンはページ 1、2、3、およびページ 4[B] を参照し、クラスター (A) は引き続き 1[A]、2、3、4 を参照しています。



時間が経過して作成元 Amazon DocumentDB クラスターボリュームとクローンの両方で追加の変更があると、その変更をキャプチャして保存するためにさらにストレージが必要になります。

ソースクラスターボリュームの削除

1 つ以上のクローンが関連付けられているソースクラスターボリュームを削除しても、そのクローンには影響しません。クローンは、ソースクラスターボリュームが前に所有していたページをポイントし続けます。

Amazon DocumentDB のクローンの作成

クローンは、ソース Amazon DocumentDB クラスターと同じ AWS アカウントに作成できます。そのためには、 AWS マネジメントコンソール または AWS CLI と以下の手順を使用できます。

Amazon DocumentDB のクローン作成を使用すると、プロビジョニングされた Amazon DocumentDB クラスターから Amazon DocumentDB クラスターのプロビジョニングクローンを作成できます。

Using the AWS マネジメントコンソール

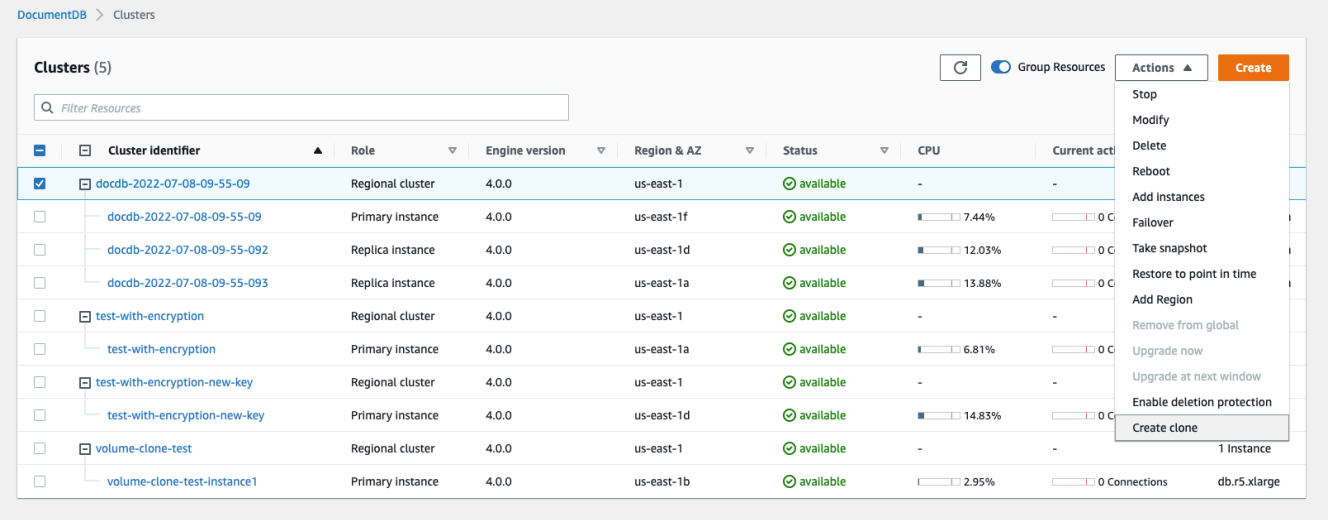
AWS マネジメントコンソールを使用して Amazon DocumentDB クラスターのクローンを作成する手順を以下に示します。

1 つの Amazon DocumentDB インスタンスを持つ Amazon DocumentDB クラスターで AWS マネジメントコンソール 結果を使用してクローンを作成します。

これらの手順は、クローンを作成しているのと同じ AWS アカウントが所有する DB クラスターに適用されます。クロス AWS アカウントのクローン作成は Amazon DocumentDB ではサポートされていないため、DB クラスターは同じアカウントによって所有されている必要があります。

を使用して、 AWS アカウントが所有する DB クラスターのクローンを作成するには AWS マネジメントコンソール

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. リストから Amazon DocumentDB クラスターを選択し、[アクション] で、[クローンの作成] を選択します。

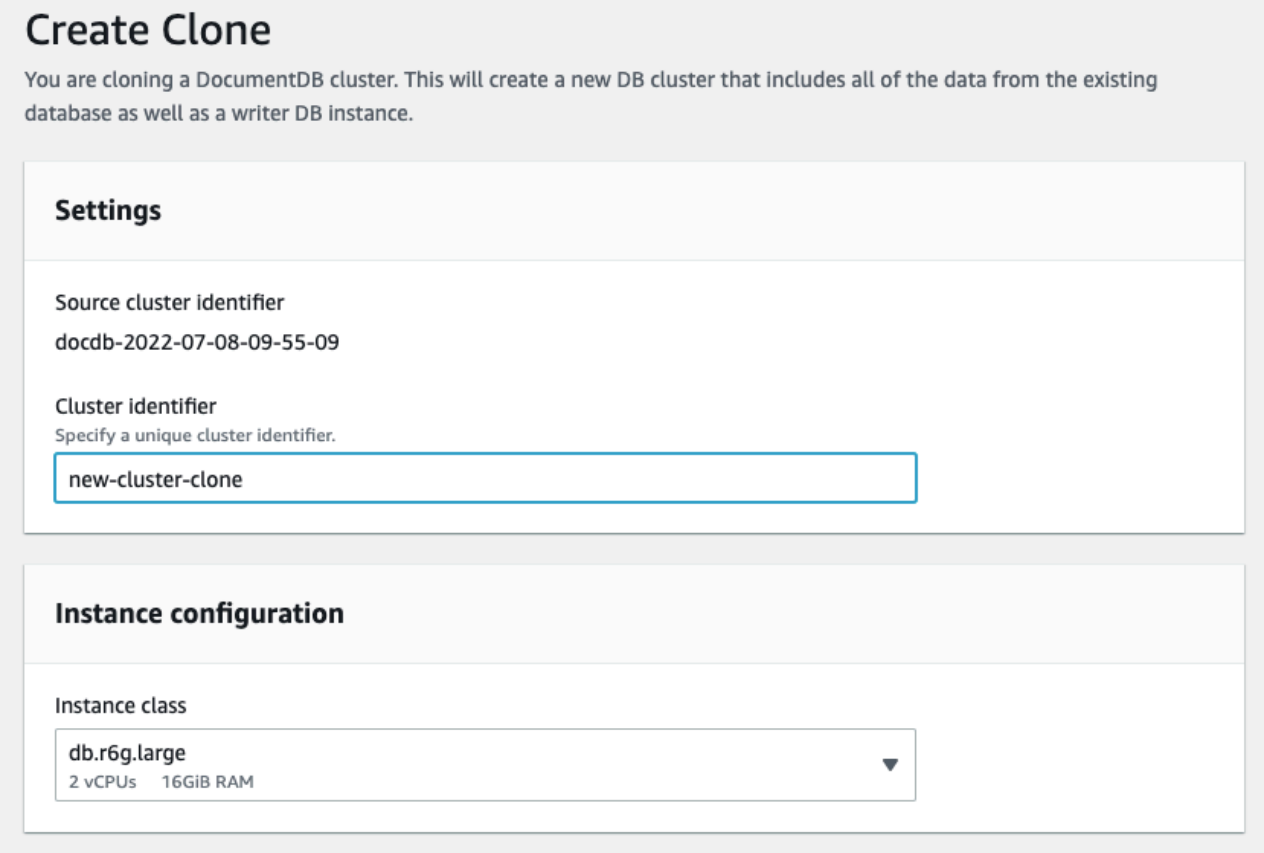


[クローンの作成] ページが開きます。そこで、[クラスター識別子]、[インスタンスクラス] などの Amazon DocumentDB クラスタークローンのオプションが設定できます。

4. [設定] セクションで、以下の手順を実行します。

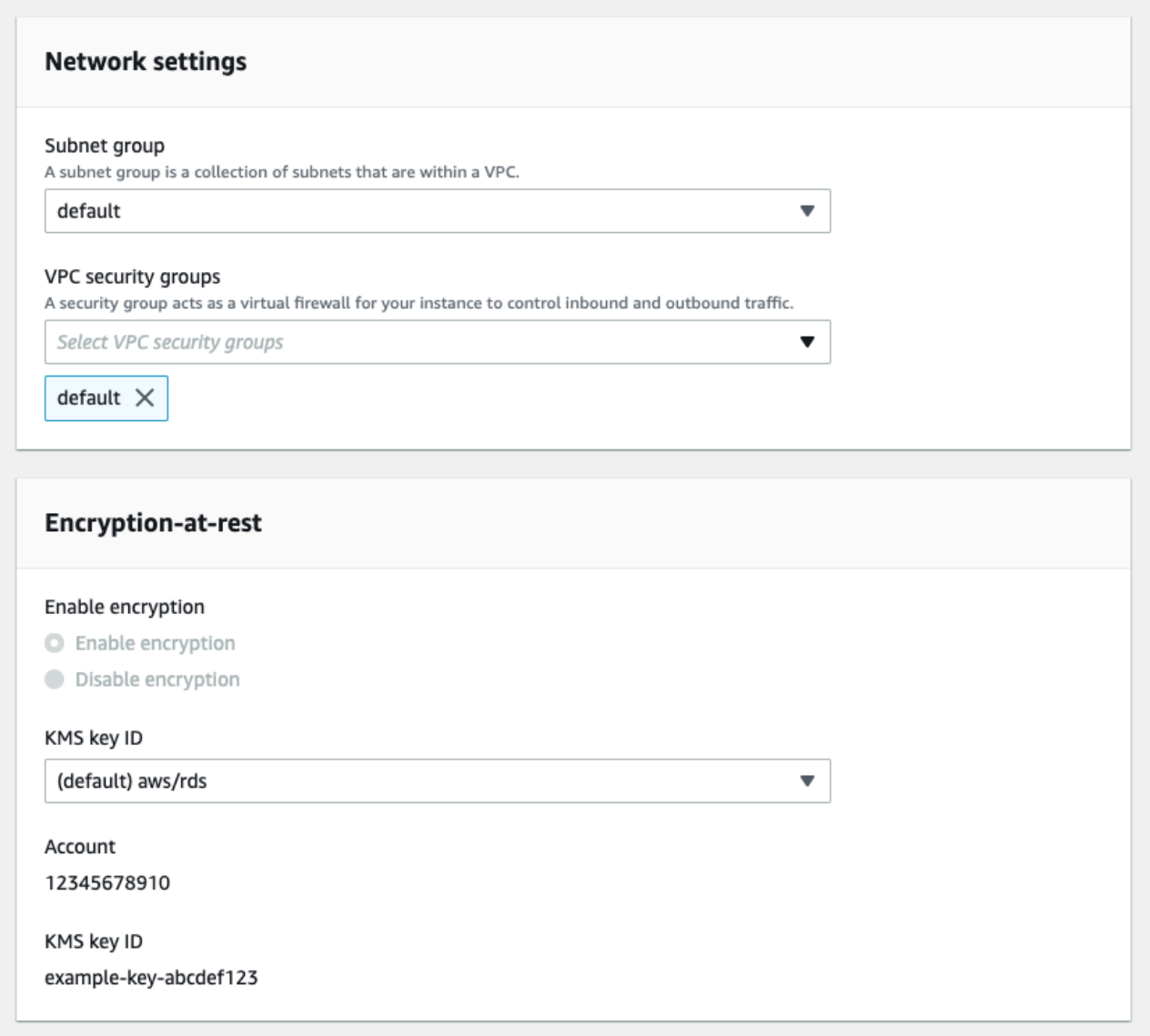
   a. [DB クラスター識別子] に、作成する Amazon DocumentDB クラスターのクローンに付ける名前を入力します。

b. [インスタンス設定] では、クローンとして作成された Amazon DocumentDB クラスターに適した [インスタンスクラス] を選択します。

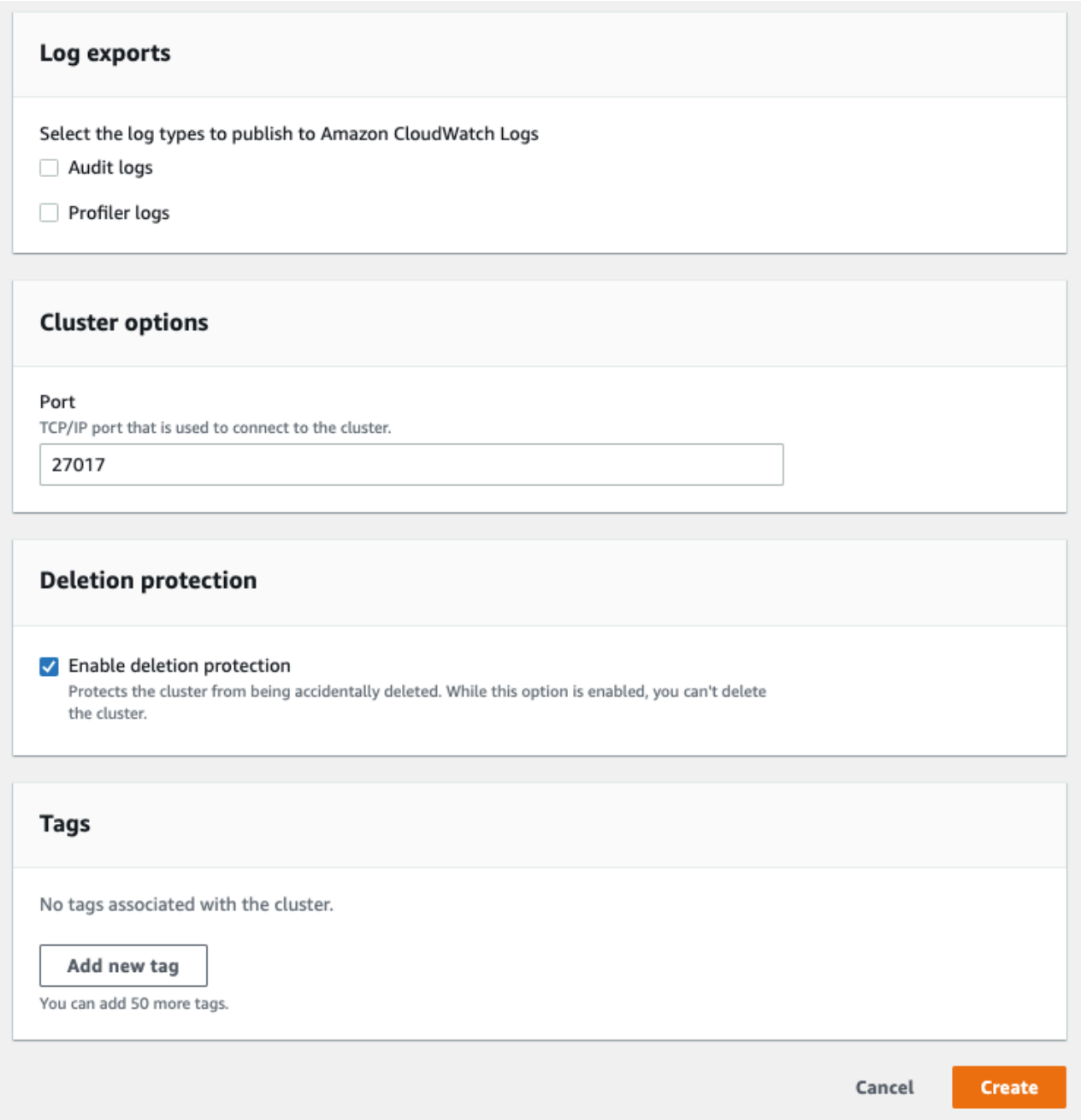


c. [ネットワーク設定] では、ユースケースに適したサブネットグループと、関連する VPC セキュリティグループを選択します。

d. [保管時の暗号化]では、作成元クラスター (クローンが作成されるクラスター) で暗号化が有効になっている場合、クローンとして作成されたクラスターでも暗号化が有効になっている必要があります。このシナリオに当てはまる場合、[暗号化を有効にする] オプションはグレー表示 (無効) になりますが、[暗号化を有効にする] オプションが選択されています。逆に、作成元クラスターで暗号化が有効になっていない場合は、[暗号化を有効にする] オプションが使用でき、暗号化を有効または無効にすることができます。



e. エクスポートするログの種類を選択し (オプション)、クラスターへの接続に使用する特定のポートを入力し、クラスターの誤削除からの保護を有効にして (デフォルトで有効) 、新しいクラスタークローンの設定を完了します。



f. Amazon DocumentDB クラスターのクローンのすべての設定の入力を完了します。Amazon DocumentDB クラスターとインスタンスの設定の詳細については、「Amazon DocumentDB クラスターの管理」を参照してください。

5. [クローンを作成] を選択して、選択した Amazon DocumentDB クラスターの Amazon DocumentDB クローンを起動します。

クローンが作成されると、コンソールの [データベース] セクションに他の Amazon DocumentDB クラスターとともに一覧表示され、現在の状態が表示されます。状態が [使用可能] の場合は、クローンはすぐに使用できます。

Using the AWS CLI

AWS CLI を使用して Amazon DocumentDB クラスターをクローンするには、いくつかのステップが必要です。

使用する`restore-db-cluster-to-point-in-time` AWS CLI コマンドは、Amazon DocumentDB インスタンスが 0 個の空の Amazon DocumentDB クラスターになります。つまり、このコマンドは Amazon DocumentDB クラスターのみを復元し、クラスターの DB インスタンスは復元しません。これは、クローンが使用可能になった後に別途行います。プロセスは 2 ステップで、次のとおりです。

1. restore-db-cluster-to-point-in-time CLI コマンドを使用して、クローンを作成します。このコマンドで使用するパラメータで、作成する空の Amazon DocumentDB クラスター (クローン) の容量タイプなどの詳細が制御されます。
2. create-db-instance CLI コマンドを使用して、クローン用の Amazon DocumentDB インスタンスを作成し、復元された Amazon DocumentDB クラスターに Amazon DocumentDB インスタンスを再作成します。

次のコマンドは、 AWS CLI が AWS リージョンをデフォルトとして設定されていることを前提としています。この方法では、`--region` の名前を各コマンドに入力する手間が省けます。詳細については、「AWS CLIの設定」を参照してください。また、以下の各 CLI コマンドで `--region` を指定することもできます。

クローンの作成

`restore-db-cluster-to-point-in-time` CLI コマンドに渡す特定のパラメータは、さまざまです。渡す内容は、作成するクローンのタイプによって異なります。

次の手順に従い、プロビジョニングされた Amazon DocumentDB クラスターから Amazon DocumentDB プロビジョニングクローンを作成します。

作成元 Amazon DocumentDB クラスターと同じエンジンモードのクローンを作成するには

- `restore-db-cluster-to-point-in-time` CLI コマンドを使用して、次のパラメータに値を指定します。

- `--db-cluster-identifier` - クローン用の意味のある名前を選択します。restore-db-cluster-to-point-in-time CLI コマンド使用時に、クローンに名前を付けます。
- `--restore-type` - 出典 DB クラスターのクローンの作成に `copy-on-write` を使用します。このパラメータを指定しない場合、`restore-db-cluster-to-point-in-time` は、クローンを作成するのではなく、Amazon DocumentDB クラスターを復元します。`restore-type` のデフォルトは `full-copy` です。
- `--source-db-cluster-identifier` - クローンを作成する作成元 Amazon DocumentDB クラスターの名前を使用します。
- `--use-latest-restorable-time` - この値は、クローンの最新の復元可能なボリュームデータを指します。このパラメータは `restore-type copy-on-write` に必要ですが、一緒に `restore-to-time parameter` を使用することはできません。

次の例では、`my-source-cluster` という名前のクラスターから `my-clone` という名前のクローンを作成します。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

このコマンドは、クローンの詳細を含む JSON オブジェクトを返します。クローンの DB インスタンスを作成する前に、作成した DB クラスターのクローンが使用可能であることを確認します。詳細については、下記の「ステータスの確認とクローンの詳細の取得」をご参照ください。

ステータスの確認とクローンの詳細の取得

次のコマンドを使用して、新しく作成した空の DB クラスターのステータスが確認できます。

```
$ aws docdb describe-db-clusters --db-cluster-identifier my-clone --query '*[].
[Status]' --output text
```

または、次の AWS CLI クエリを使用して、クローンの DB インスタンスを作成するために必要なステータスやその他の値を取得できます。

Linux、macOS、Unix の場合:

```
aws docdb describe-db-clusters --db-cluster-identifier my-clone \
  --query '*[].{Status:Status,Engine:Engine,EngineVersion:EngineVersion}'
```

Windows の場合:

```
aws docdb describe-db-clusters --db-cluster-identifier my-clone ^
  --query "*[].{Status:Status,Engine:Engine,EngineVersion:EngineVersion}"
```

このクエリにより、以下のような出力が返されます。

```
[
  {
        "Status": "available",
        "Engine": "docdb",
        "EngineVersion": "4.0.0",
    }
]
```

クローン用の Amazon DocumentDB インスタンスの作成

クローン用の DB インスタンスを作成するには、create-db-instance CLI コマンドを使用します。

`--db-instance-class` パラメータは、プロビジョニングされた Amazon DocumentDB クラスターにのみ使用されます。

Linux、macOS、Unix の場合:

```
aws docdb create-db-instance \
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
    --db-instance-class  db.r5.4xlarge \
```

```
    --engine docdb
```

Windows の場合:

```
aws docdb create-db-instance ^
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class  db.r5.4xlarge ^
    --engine docdb
```

クローン作成に使用するパラメータ

次の表は `restore-db-cluster-to-point-in-time` を使用して Amazon DocumentDB クラスターのクローンを作成する際に使用されるさまざまなパラメータをまとめたものです。

パラメータ	説明
--source-db-cluster-identifier	クローンを作成する作成元 Amazon DocumentDB クラスターの名前を使用します。
--db-cluster-identifier	クローン用に意味のある名前を選択します。 `restore-db-cluster-to-point-in-time` コマンドでクローンの名前を付けます。次に、この名前を `create-db-instance` コマンドに渡します。
--restore-type	`--restore-type` として `copy-on-write` を指定すると、作成元 Amazon DocumentDB クラスターを復元するのではなく、作成元 DB クラスターのクローンを作成します。
--use-latest-restorable-time	この値は、クローンの最新の復元可能なボリュームデータを指します。

Amazon DocumentDB クラスターの耐障害性について理解する

Amazon DocumentDB クラスターは、耐障害性を持つように設計されています。各クラスターのボリュームは 1 つの で複数のアベイラビリティーゾーンにまたがり AWS リージョン、各アベイラビリティーゾーンにはクラスターのボリュームデータのコピーが含まれます。この機能は、クラスターがデータ喪失なしでアベイラビリティーゾーンの障害に耐えることができ、発生するのはサービスの短時間の中断のみであることを意味します。

クラスターのプライマリインスタンスが失敗した場合、Amazon DocumentDB は以下のいずれかの方法で、新しいプライマリインスタンスに自動的にフェイルオーバーします。

- 既存の Amazon DocumentDB レプリカを新しいプライマリインスタンスに昇格し、昇格されたレプリカの置き換えを作成する。レプリカインスタンスへのフェイルオーバーは、通常 30 秒未満で完了します。この期間中、読み取りおよび書き込みオペレーションが短時間中断することがあります。クラスターの可用性を高めるには、2 つ以上の異なるアベイラビリティーゾーンに 1 つ以上の Amazon DocumentDB レプリカを作成します。
- 新しいプライマリインスタンスを作成する。これは、クラスター内にレプリカ・インスタンスがなく、完了までに数分かかる場合のみ発生します。

クラスターに 1 つ以上の Amazon DocumentDB レプリカがある場合は、障害発生中に 1 つの Amazon DocumentDB レプリカがプライマリインスタンスに昇格されます。障害イベントによって短い中断が発生し、その間例外によって読み取りと書き込みオペレーションが失敗します。ただし、一般的なサービスの復元時間は 120 秒未満であり、多くの場合 60 秒未満で復元されます。クラスターの可用性を高めるには、2 つ以上の異なるアベイラビリティーゾーンに 1 つ以上の Amazon DocumentDB レプリカを作成します。

各レプリカに優先度を割り当てることで、Amazon DocumentDB レプリカがプライマリインスタンスに昇格される順序をカスタマイズできます。優先度の範囲は、最も高い 0 から最も低い 15 までです。プライマリインスタンスが失敗した場合、最も高い優先度の Amazon DocumentDB レプリカが、新しいプライマリインスタンスに昇格します。Amazon DocumentDB レプリカの優先度はいつでも変更できます。優先度を変更しても、フェイルオーバーはトリガーされません。`modify-db-instance` オペレーションで、`--promotion-tier` パラメータを使用できます。インスタンスのフェイルオーバー優先度のカスタマイズの詳細については、「Amazon DocumentDB フェイルオーバー」を参照してください。

複数の Amazon DocumentDB レプリカで同じ優先度を共有でき、その場合は昇格階層が発生します。複数の Amazon DocumentDB レプリカで同じ優先度を共有する場合、最大サイズのレプリカがプライマリに昇格します。複数の Amazon DocumentDB レプリカで同じ優先度とサイズを共有する場合、昇格階層の任意のレプリカが昇格します。

クラスターに Amazon DocumentDB レプリカが含まれていない場合、障害イベントの発生時にプライマリインスタンスが再作成されます。障害イベントによって中断が発生し、その間例外によって読み取りと書き込みオペレーションが失敗します。新しいプライマリインスタンスが再作成されると、サービスが回復します。これは、通常は 10 分未満で行われます。Amazon DocumentDB レプリカのプライマリインスタンスへの昇格は、新しいプライマリインスタンスの作成よりもはるかに短時間で実行されます。

Amazon DocumentDB インスタンスの管理

以下のトピックでは、Amazon DocumentDB インスタンスの管理に役立つ情報を提供します。インスタンスクラスとステータスの詳細、およびインスタンスを作成、削除、変更する方法が含まれます。

トピック

- インスタンスのステータスの判断
- Amazon DocumentDB インスタンスのライフサイクル
- インスタンスクラスの管理
- NVMe-backed インスタンス

インスタンスのステータスの判断

有効なインスタンスのステータス、その意味、インスタンスのステータスの判断方法については、「Amazon DocumentDB インスタンスのステータスのモニタリング」を参照してください。

Amazon DocumentDB インスタンスのライフサイクル

Amazon DocumentDB インスタンスのライフサイクルには、インスタンスの作成、変更、保守、アップグレード、バックアップと復元の実行、再起動、削除が含まれます。このセクションでは、これらのプロセスを完了する方法について説明します。

トピック

- クラスターへの Amazon DocumentDB インスタンスの追加
- Amazon DocumentDB インスタンスの説明
- Amazon DocumentDB インスタンスの変更
- Amazon DocumentDB インスタンスの再起動
- Amazon DocumentDB インスタンスの削除

AWS マネジメントコンソール または を使用して、新しい Amazon DocumentDB インスタンスを作成できます AWS CLI。クラスターにインスタンスを追加するには、クラスターが 使用可能 状態になっている必要があります。停止しているクラスターにインスタンスを追加することはできません。クラスターが停止している場合は、最初にクラスターを起動し、クラスターが 使用可能 になるまで

待ってから、インスタンスを追加します。詳細については、「Amazon DocumentDB クラスターの停止と開始」を参照してください。

> **Note**
>
> コンソールを使用して Amazon DocumentDB クラスターを作成した場合、同時にインスタンスが自動的に作成されます。追加のインスタンスを作成する場合は、以下のいずれかの手順を使用します。

クラスターへの Amazon DocumentDB インスタンスの追加

Using the AWS マネジメントコンソール

次の手順に従い、Amazon DocumentDB コンソールを使用してクラスターのインスタンスを作成します。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. インスタンスの追加先のクラスターを選択するには、クラスター名の左にあるボタンを選択します。
4. [アクション] を選択し、[インスタンスの追加] を選択します。
5. [Add instance to: (インスタンスの追加先:)<cluster-name>] ページで、クラスターに追加する各インスタンスに対して以下の手順を繰り返します。最大数は 15 です。

   a. インスタンス識別子 - このインスタンスの一意の識別子を入力するか、クラスター識別子に基づき Amazon DocumentDB でインスタンスを指定できるようにします。

      インスタンスの名前付けの制約:

- 長さは [1 ～ 63] 個の英字、数字、またはハイフンです。
- 1 字目は英字である必要があります。
- ハイフンで終わる、または 2 つのハイフンを連続して含めることはできません。
- Amazon RDS、Neptune、Amazon DocumentDB のすべてのインスタンスで AWS アカウント、リージョンごとに一意である必要があります。

b. インスタンスクラス - ドロップダウンリストから、このインスタンスに必要なインスタンスタイプを選択します。

c. プロモーション枠 - ドロップダウンリストからインスタンスのプロモーション枠を選択するか、[指定なし] を選択して Amazon DocumentDB によってインスタンスのプロモーション枠が設定されるようにします。数値が小さいほど優先順位が高くなります。詳細については、「フェイルオーバーターゲットの制御」を参照してください。

d. インスタンスをさらに追加するには、[Add additional instances (追加のインスタンスを追加する)] を選択してステップ a、b、c を繰り返します。

6. オペレーションを完了します。

- インスタンスをクラスターに追加するには、[作成] を選択します。
- 操作をキャンセルするには、[キャンセル] を選択します。

インスタンスが作成されるまでに数分かかります。コンソールまたは を使用して AWS CLI 、インスタンスのステータスを表示できます。詳細については、「インスタンスのステータスのモニタリング」を参照してください。

Using the AWS CLI

以下のパラメータを指定して `create-db-instance` AWS CLI オペレーションを使用して、クラスターのプライマリインスタンスを作成します。

- **`--db-instance-class`** — 必須。インスタンスのコンピューティング性能とメモリ容量 (例: `db.m4.large`)。すべての AWS リージョン中で、インスタンスクラスが使用できるわけではありません。
- **`--db-instance-identifier`** — 必須。インスタンスを識別する文字列。

  インスタンスの名前付けの制約:

  - 長さは [1 ～ 63] 個の英字、数字、またはハイフンです。
  - 1 字目は英字である必要があります。

- ハイフンで終わる、または 2 つのハイフンを連続して含めることはできません。
- Amazon RDS、Neptune、Amazon DocumentDB のすべてのインスタンスで AWS アカウント、リージョンごとに一意である必要があります。

- **--engine** — 必須。docdb を指定してください。
- **--availability-zone** - オプション。このインスタンスが作成されるアベイラビリティーゾーン このパラメータを使用して、さまざまなアベイラビリティーゾーンで耐障害性を強化するインスタンスを見つけます。詳細については、「Amazon DocumentDB の高可用性とレプリケーション」を参照してください。
- **--promotion-tier** - オプション。このインスタンスのフェイルオーバー優先度の階層。0 ~ 15 の数値である必要があります。数値が小さいほど、優先順位が高くなります。詳細については、「フェイルオーバーターゲットの制御」を参照してください。

1. 最初に、インスタンスを作成できるアベイラビリティーゾーンを決定します。

   インスタンスを作成する前にアベイラビリティーゾーンを指定する場合は、次のコマンドを実行して Amazon DocumentDB クラスターで使用できるアベイラビリティーゾーンを確認します。

   Linux、macOS、Unix の場合:

   ```
   aws docdb describe-db-clusters \
         --query 'DBClusters[*].[DBClusterIdentifier,AvailabilityZones[*]]'
   ```

   Windows の場合:

   ```
   aws docdb describe-db-clusters ^
         --query 'DBClusters[*].[DBClusterIdentifier,AvailabilityZones[*]]'
   ```

   このオペレーションによる出力は、次のようになります。

   ```
   [
       [
           "sample-cluster",
           [
               "us-east-1c",
               "us-east-1b",
               "us-east-1a"
   ```

```
        ]
    ]
]
```

2. 次に、リージョンで作成できるインスタンスクラスを決定します。

   リージョンで利用可能なインスタンスクラスを確認するには、次のコマンドを実行します。出力から、Amazon DocumentDB クラスターに追加したいインスタンスのためのインスタンスクラスを選択します。

   Linux、macOS、Unix の場合:

```
aws docdb describe-orderable-db-instance-options \
        --engine docdb \
        --query 'OrderableDBInstanceOptions[*].DBInstanceClass'
```

   Windows の場合:

```
aws docdb describe-orderable-db-instance-options ^
        --engine docdb ^
        --query 'OrderableDBInstanceOptions[*].DBInstanceClass'
```

   このオペレーションによる出力は、次のようになります。

```
[
    "db.r5.16xlarge",
    "db.r5.2xlarge",
    "db.r5.4xlarge",
    "db.r5.8xlarge",
    "db.r5.large",
    "db.r5.xlarge"
]
```

3. 最後に、Amazon DocumentDB クラスターにインスタンスを追加します。

   Amazon DocumentDB クラスターにインスタンスを追加するには、次のコマンドを実行します。

   Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

このオペレーションによる出力は、次のようになります。

```
{
    "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"
    }
}
```

インスタンスが作成されるまでに数分かかります。コンソールまたは を使用して AWS CLI 、インスタンスのステータスを表示できます。詳細については、「Amazon DocumentDB インスタンスのステータスのモニタリング」を参照してください。

Amazon DocumentDB インスタンスの説明

Amazon DocumentDB マネジメントコンソールまたは AWS CLI を使用して、接続エンドポイント、セキュリティグループ VPC、認証機関、Amazon DocumentDB インスタンスに関連するパラメータグループなどの詳細を確認できます。

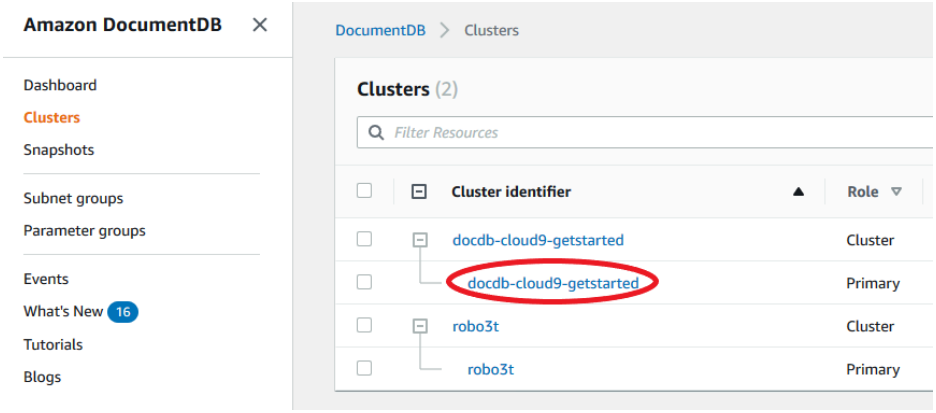
Using the AWS マネジメントコンソール

AWS マネジメントコンソールを使用してインスタンスの詳細を確認するには、次の手順に従います。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. クラスターナビゲーションボックスに、[Cluster Identifier]の列が表示されます。インスタンスは、次のスクリーンショットのようにクラスターの下に一覧表示されます。



4. インスタンスの一覧で、詳細を表示したいインスタンスの名前を選択します。インスタンスに関する情報は、次のグループに分けられます。

   - 概要 エンジンのバージョン、クラス、ステータス、保留中のメンテナンスなど、インスタンスに関する一般情報。
   - Connectivity & Security - Connect セクションには、[mongo] シェルまたはアプリケーションを使用してこのインスタンスに接続するための接続エンドポイントが一覧表示されます。[セキュリティグループ] セクションには、このインスタンスに関連付けられているセキュリティグループと、その VPC ID や説明が一覧表示されます。

- 設定 - [詳細] セクションには、インスタンスの Amazon リソースネーム (ARN)、エンドポイント、ロール、クラス、認証機関など、インスタンスの設定とステータスが一覧表示されます。また、インスタンスのセキュリティとネットワーク設定、バックアップ情報も一覧表示されます。[クラスターの詳細] セクションには、このインスタンスが属するクラスターの詳細が表示されます。[クラスターのインスタンス] セクションには、クラスターに属するすべてのインスタンスが、各インスタンスのロールとクラスターパラメータグループのステータスとともに一覧表示されます。

  > **Note**
  >
  > クラスターの詳細 ヘッダーの次に Modify を選択することにより、インスタンスに関連付けられているクラスターを変更できます。詳細については、「Amazon DocumentDB クラスターの変更」を参照してください。

- [モニタリング] — このクラスターの Amazon CloudWatch Logs メトリクス。詳細については、「Amazon DocumentDB と CloudWatch のモニタリング」を参照してください。
- [イベントとタグ] — [最近のイベント] セクションには、このインスタンスの最近のイベントの一覧が表示されます。Amazon DocumentDB は、クラスター、インスタンス、スナップショット、セキュリティグループ、およびクラスターパラメータグループに関連するイベントのレコードを保持します。この情報には、各イベントに関連する日付、時刻、およびメッセージが含まれます。[タグ] セクションには、このクラスターにアタッチされているタグが一覧表示されます。詳細については、「Amazon DocumentDB リソースへのタグ付け」を参照してください。

Using the AWS CLI

を使用して Amazon DocumentDB インスタンスの詳細を表示するには AWS CLI、次の例に示すように `describe-db-clusters` コマンドを使用します。詳細については、「Amazon DocumentDB リソース管理 API リファレンス」の「`DescribeDBInstances`」を参照してください。

> **Note**
>
> クラスターやインスタンスのライフサイクル管理などの特定の管理機能において、Amazon DocumentDB は Amazon RDS と共有される運用テクノロジーを使用し

ます。filterName=engine,Values=docdb のフィルターパラメータは Amazon DocumentDB クラスターのみを返します。

1. List all Amazon DocumentDB instances.

   次の AWS CLI コードは、リージョン内のすべての Amazon DocumentDB インスタンスの詳細を一覧表示します。

   Linux、macOS、Unix の場合:

   ```
   aws docdb describe-db-instances \
       --filter Name=engine,Values=docdb
   ```

   Windows の場合:

   ```
   aws docdb describe-db-instances \
       --filter Name=engine,Values=docdb
   ```

2. 指定した Amazon DocumentDB インスタンスのすべての詳細を一覧表示する

   次のコードは sample-cluster-instance の詳細を一覧します。--db-instance-identifier パラメータにインスタンスの名前を含めると、出力はその特定のインスタンスの情報に制限されます。

   Linux、macOS、Unix の場合:

   ```
   aws docdb describe-db-instances \
       --db-instance-identifier sample-cluster-instance
   ```

   Windows の場合:

   ```
   aws docdb describe-db-instances \
       --db-instance-identifier sample-cluster-instance
   ```

   このオペレーションの出力は、次のようになります。

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

Amazon DocumentDB インスタンスの変更

Amazon DocumentDB インスタンスは、 AWS マネジメントコンソール または を使用して変更できます AWS CLI。インスタンスを変更するには、インスタンスが 使用可能 状態である必要があります。停止しているインスタンスを変更することはできません。クラスターが停止している場合は、最

初にクラスターを起動し、インスタンスが使用可能になるまで待ってから、必要な変更を加えます。詳細については、「Amazon DocumentDB クラスターの停止と開始」を参照してください。

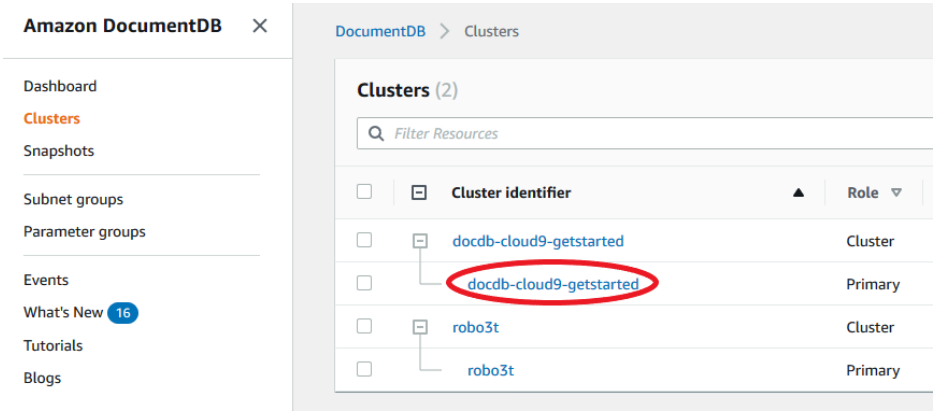
Using the AWS マネジメントコンソール

コンソールを使用して特定の Amazon DocumentDB インスタンスを変更するには、以下の手順を実行します。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. クラスターナビゲーションボックスに、[Cluster Identifier]の列が表示されます。インスタンスは、次のスクリーンショットのようにクラスターの下に一覧表示されます。



4. 変更するインスタンスの左側にあるボックスをチェックします。
5. [アクション] を選択してから、[変更] を選択します。
6. [インスタンスの変更: <instance-name>] ペインで、必要な変更を行います。以下の変更を行うことができます。

   - インスタンス仕様 - インスタンス識別子とクラス。インスタンス識別子の命名制約:

- インスタンス識別子 — 現在のリージョンで が所有するすべてのインスタンス AWS アカウント に一意の名前を入力します。インスタンス識別子は [1 — 63] 英数字またはハイフンを使用し、最初の文字に文字を使用します。最後の文字をハイフンにすることはできず、ハイフンを 2 つ続けて使用することもできません。
- インスタンスクラス — ドロップダウンメニューから、Amazon DocumentDB インスタンスのためのインスタンスクラスを選択します。詳細については、「インスタンスクラスの管理」を参照してください。

- 認証機関 - このインスタンスのサーバー証明書。詳細については、「Amazon DocumentDB TLS 証明書の更新」を参照してください。
- フェイルオーバー - フェイルオーバーにより、昇格階層が最も高いインスタンスがプライマリに昇格されます 詳細については、「Amazon DocumentDB フェイルオーバー」を参照してください。
- メンテナンス - 保留中の変更またはパッチをクラスター内のインスタンスに適用するメンテナンスウィンドウ。

7. 完了したら、[続行] を選択して変更の概要を表示します。
8. 変更を確認したら、直ちに、あるいは、[変更のスケジューリング] の下にある次のメンテナンスウィンドウ中に適用することができます。[インスタンスの変更] を選択して、変更を保存します。または、[キャンセル] を選択して変更を破棄することもできます。

変更が適用されるまでに数分かかります。インスタンスを使用できるのは、そのステータスが [available] である場合のみです。コンソールまたは AWS CLIを使用して、インスタンスのステータスをモニタリングできます。詳細については、「Amazon DocumentDB インスタンスのステータスのモニタリング」を参照してください。

Using the AWS CLI

を使用して特定の Amazon DocumentDB インスタンスを変更するには AWS CLI、次のパラメータ`modify-db-instance`を指定して を使用します。詳細については、「ModifyDBInstance」を参照してください。次のコードは、`sample-instance` インスタンスのインスタンスクラスを `db.r5.large` に変更します。

パラメータ

- **`--db-instance-identifier`** — 必須。変更するインスタンスの識別子。
- **`--db-instance-class`** - オプション。インスタンスの新しいコンピューティング能力とメモリ容量 (例: `db.r5.large`)。すべてのインスタンスクラスがすべての AWS リージョンで利用できるわけではありません。インスタンスクラスを変更すると、変更中に機能停止が発生しま

す。ApplyImmediately がこのリクエストの [true] として指定されない限り、変更は次のメンテナンスウィンドウ中に適用されます。

- **--apply-immediately** または **--no-apply-immediately** - オプション。この修正をすぐに適用するか、次のメンテナンスウィンドウまで待機するか指定します。このパラメータを省略した場合、次のメンテナンスウィンドウ中に変更が実行されます。

Example

Linux、macOS、Unix の場合:

```
aws docdb modify-db-instance \
       --db-instance-identifier sample-instance \
       --db-instance-class db.r5.large \
       --apply-immediately
```

Windows の場合:

```
aws docdb modify-db-instance ^
       --db-instance-identifier sample-instance ^
       --db-instance-class db.r5.large ^
       --apply-immediately
```

このオペレーションによる出力は、次のようになります。

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

変更が適用されるまでに数分かかります。インスタンスを使用できるのは、そのステータスが [available] である場合のみです。インスタンスのステータスは、 AWS マネジメントコンソール または を使用してモニタリングできます AWS CLI。詳細については、「Amazon DocumentDB インスタンスのステータスのモニタリング」を参照してください。

Amazon DocumentDB インスタンスの再起動

ときどき、通常はメンテナンスのために、Amazon DocumentDB インスタンスを再起動する必要があります。クラスターに関連付けられたクラスターパラメータグループを変更するなど、特定の変更を行う場合は、変更を有効にするためにクラスター内のインスタンスを再起動する必要があります。AWS マネジメントコンソール または を使用して、指定されたインスタンスを再起動できます AWS CLI。

インスタンスを再起動すると、データベースエンジンサービスが再起動されます。再起動により一時的に機能停止になります。その間、インスタンスのステータスは [rebooting] に設定されます。再起動が完了すると、Amazon DocumentDB イベントが作成されます。

インスタンスを再起動しても、フェイルオーバーは発生しません。Amazon DocumentDB クラスターをフェイルオーバーするには、 AWS マネジメントコンソール または AWS CLI オペレーション を使用しますfailover-db-cluster。詳細については、「Amazon DocumentDB フェイルオーバー」を参照してください。

available (利用可能) 状態でないインスタンスを再起動することはできません。データベースは、以前の要求による変更、またはメンテナンスウィンドウのアクションなど、いくつかの理由で使用できない場合があります。インスタンスステータスの詳細については、「Amazon DocumentDB インスタンスのステータスのモニタリング」を参照してください。

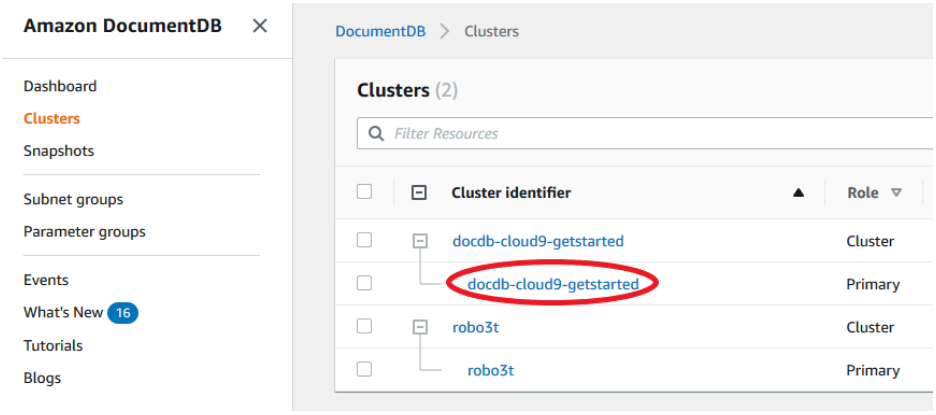
Using the AWS マネジメントコンソール

次の手順では、コンソールを使用して指定するインスタンスを再起動します。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. クラスターナビゲーションボックスに、[Cluster Identifier]の列が表示されます。インスタンスは、次のスクリーンショットのようにクラスターの下に一覧表示されます。



4. 再起動するインスタンスの左側にあるボックスをチェックします。
5. [Actions (アクション)]、[Reboot (再起動)] の順に選択し、[Reboot (再起動)] を選択して再起動を確認します。

インスタンスが再起動するまでには数分かかります。インスタンスを使用できるのは、そのステータスが [available] である場合のみです。コンソールまたは AWS CLIを使用して、インスタンスのステータスをモニタリングできます。詳細については、「Amazon DocumentDB インスタンスのステータスのモニタリング」を参照してください。

Using the AWS CLI

Amazon DocumentDB インスタンスを再起動するには、`reboot-db-instance` のパラメータとともに `--db-instance-identifier` のオペレーションを使用します。このパラメータは、再起動するインスタンスの識別子を指定します。

次のコードは、インスタンス `sample-instance` を再起動します。

Example

Linux、macOS、Unix の場合:

```
aws docdb reboot-db-instance \
       --db-instance-identifier sample-instance
```

Windows の場合:

```
aws docdb reboot-db-instance ^
       --db-instance-identifier sample-instance
```

このオペレーションによる出力は、次のようになります。

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

インスタンスが再起動するまでには数分かかります。インスタンスを使用できるのは、そのステータスが [available] である場合のみです。コンソールまたは AWS CLIを使用して、インスタンスのステータスをモニタリングできます。詳細については、「Amazon DocumentDB インスタンスのステータスのモニタリング」を参照してください。

Amazon DocumentDB インスタンスの削除

Amazon DocumentDB インスタンスは、 AWS マネジメントコンソール または を使用して削除できます AWS CLI。インスタンスを削除するには、インスタンスが 使用可能 状態である必要があります。停止しているインスタンスを削除することはできません。インスタンスを含む Amazon DocumentDB クラスターが停止している場合は、最初にクラスターを起動し、インスタン

スが 使用可能 になるまで待ってから、インスタンスを削除します。詳細については、「Amazon DocumentDB クラスターの停止と開始」を参照してください。

> **Note**
>
> Amazon DocumentDB は、クラスターボリューム内のすべてのデータを保存します。クラスターからすべてのインスタンスを削除しても、そのクラスターボリュームのデータは保持されます。データに再アクセスする必要がある場合は、いつでもクラスターにインスタンスを追加し、中断した個所から再開できます。

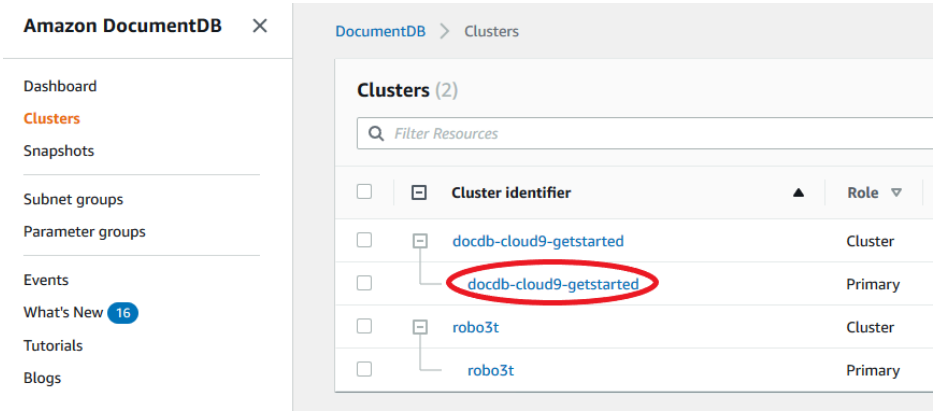
Using the AWS マネジメントコンソール

次の手順では、コンソールを使用して指定した Amazon DocumentDB インスタンスを削除します。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. クラスターナビゲーションボックスに、[Cluster Identifier]の列が表示されます。インスタンスは、次のスクリーンショットのようにクラスターの下に一覧表示されます。



4. 削除するインスタンスの左側にあるボックスをチェックします。
5. [Actions (アクション)] を選択して、[Delete (削除)] を選択します。

   1. クラスターの最後のインスタンスを削除する場合:

      - Create final cluster snapshot? (最終クラススナップショットを作成しますか?) - クラスターを削除する前に最終スナップショットを作成する場合は、[はい] を選択します。それ以外の場合は [いいえ] を選択します。
      - Final snapshot name (最終スナップショット名) - 最終スナップショットを作成する場合は、作成した新しいクラススナップショットのクラススナップショットの識別子を入力します。
      - Delete <instance-name> instance? (<instance-name> インスタンスを削除しますか?) - 削除を確定するには、[delete entire cluster (クラスター全体を削除)] というフレーズをフィールドに入力します。

   2. クラスターの最後のインスタンスを削除しない場合:

      - Delete <instance-name> instance? (<instance-name> インスタンスを削除しますか?) - 削除を確認するフレーズ [delete me (これを削除する)] をフィールドに入力します。

6. [Delete (削除)] を選択して、インスタンスを削除します。

インスタンスが削除されるまでに数分かかります。インスタンスのステータスをモニタリングするには、「Amazon DocumentDB インスタンスのステータスのモニタリング」を参照してください。

Using the AWS CLI

次の手順では、 AWS CLIを使用して Amazon DocumentDB インスタンスを削除します。

1. 最初に、Amazon DocumentDB クラスター内のインスタンスの数を決定します。

   クラスター内のインスタンスの数を判断するには、以下のように `describe-db-clusters` コマンドを実行します。

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

   このオペレーションによる出力は、次のようになります。

```
[
    [
        "sample-cluster",
        [
            "sample-instance-1",
            "sample-instance-2"
        ]
    ]
]
```

2. Amazon DocumentDB クラスターに複数のインスタンスがある場合。

   指定された Amazon DocumentDB インスタンスを削除するには、次に示すように delete-db-instance のパラメータを用いて --db-instance-identifier のコマンドを使用します。インスタンスが削除されるまでに数分かかります。インスタンスのステータスをモニタリングするには、「Amazon DocumentDB インスタンスのステータスのモニタリング」を参照してください。

   ```
   aws docdb delete-db-instance \
          --db-instance-identifier sample-instance-2
   ```

   このオペレーションによる出力は、次のようになります。

   ```
   {
       "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. 削除するインスタンスが Amazon DocumentDB クラスターの最後のインスタンスである場合。

   Amazon DocumentDB クラスター内の最後のインスタンスを削除すると、そのクラスターと、そのクラスターに関連付けられた自動スナップショットと継続バックアップも削除されます。

   クラスターの最後のインスタンスを削除するには、クラスターを削除し、オプションで最終スナップショットを作成します。詳細については、「Amazon DocumentDB クラスターの削除」を参照してください。

削除保護

Amazon DocumentDB クラスターの最後のインスタンスを削除すると、クラスターと、そのクラスターに関連付けられた自動スナップショットと継続バックアップも削除されます。Amazon DocumentDB は、 AWS マネジメントコンソール または を使用して削除オペレーションを実行するかどうかにかかわらず、クラスターの削除保護を適用します AWS CLI。削除保護が有効になっている場合、クラスターを削除することはできません。

削除に対する保護が有効化されているクラスターを削除するには、まずクラスターを変更して削除保護を無効化する必要があります。詳細については、「Amazon DocumentDB クラスターの削除」を参照してください。

インスタンスクラスの管理

インスタンスクラスは、Amazon DocumentDB (MongoDB 互換) インスタンスの計算とメモリの容量を決定します。必要なインスタンスクラスは、処理能力とメモリの要件によって異なります。

Amazon DocumentDB は、インスタンスクラスの R4、R5、R6G、R8G、T3、および T4G ファミリーをサポートしています。これらのクラスは、メモリ消費の高いアプリケーション用に最適化された現行世代のインスタンスクラスです。これらのクラスの仕様については、インスタンスクラスの仕様を参照してください。

トピック

- インスタンスクラスの決定
- インスタンスクラスの変更
- リージョン別のサポートされるインスタンスクラス
- インスタンスクラスの仕様

インスタンスクラスの決定

インスタンスのクラスを決定するには、 AWS マネジメントコンソール または `describe-db-instances` AWS CLI オペレーションを使用できます。

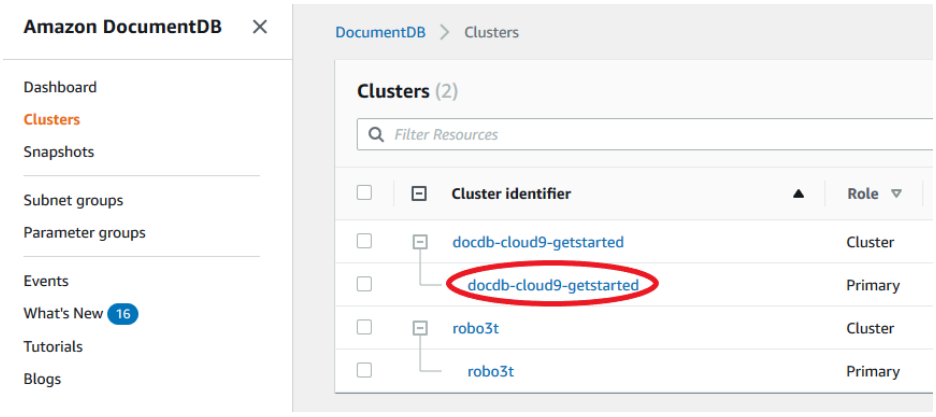
Using the AWS マネジメントコンソール

クラスターのインスタンスのインスタンスクラスを決定するには、コンソールで次の手順を実行します。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、対象とするインスタンスを見つけるために、クラスター を選択します。

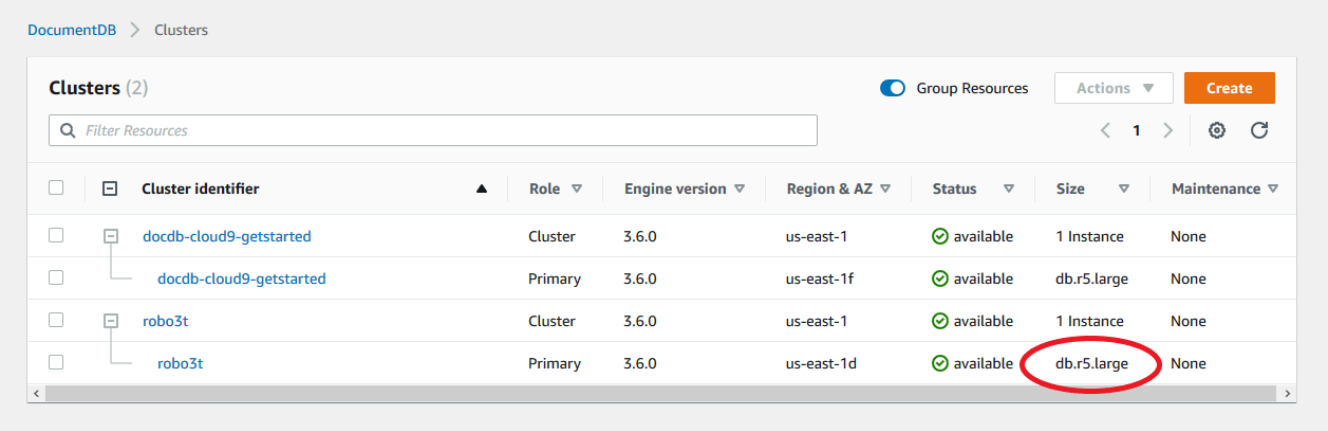
   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. クラスターナビゲーションボックスに、[Cluster Identifier]の列が表示されます。インスタンスは、次のスクリーンショットのようにクラスターの下に一覧表示されます。



4. インスタンスの一覧で、クラスターを展開して、関心のあるインスタンスを見つけます。目的のインスタンスを見つけます。次に、インスタンスの行の Size (サイズ) 列を確認し、インスタンスクラスを参照します。

   次の図では、インスタンス `robo3t` のインスタンスクラスは `db.r5.4xlarge` です。



Using the AWS CLI

を使用してインスタンスのクラスを確認するには AWS CLI、以下のパラメータを指定して `describe-db-instances`オペレーションを使用します。

- **`--db-instance-identifier`** - オプション。インスタンスクラスを検索するインスタンスを指定します。このパラメータを省略した場合、`describe-db-instances` は最大 100 個のインスタンスの説明を返します。
- **`--query`** - オプション。結果に含めるインスタンスのメンバーを指定します。このパラメータを省略すると、すべてのインスタンスメンバーが返されます。

Example

次の例では、インスタンス `sample-instance-1` のインスタンス名とクラスを検索します。

Linux、macOS、Unix の場合:

```
aws docdb describe-db-instances \
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' \
    --db-instance-identifier sample-instance-1
```

Windows の場合:

```
aws docdb describe-db-instances ^
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' ^
    --db-instance-identifier sample-instance-1
```

このオペレーションによる出力は、次のようになります。

```
[
    [
        "sample-instance-1",
        "db.r5.large"
    ]
```

Example

次の例では、最大 100 個の Amazon DocumentDB インスタンスのインスタンス名とクラスを検索します。

Linux、macOS、Unix の場合:

```
aws docdb describe-db-instances \
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' \
    --filter Name=engine,Values=docdb
```

Windows の場合:

```
aws docdb describe-db-instances ^
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' ^
    --filter Name=engine,Values=docdb
```

このオペレーションによる出力は、次のようになります。

```
[
    [
        "sample-instance-1",
        "db.r5.large"
    ],
    [
        "sample-instance-2",
        "db.r5.large"
    ],
    [
        "sample-instance-3",
        "db.r5.4xlarge"
    ],
    [
        "sample-instance-4",
        "db.r5.4xlarge"
    ]
```

```
]
```

詳細については、「Amazon DocumentDB インスタンスの説明」を参照してください。

インスタンスクラスの変更

インスタンスのインスタンスクラスは、 AWS マネジメントコンソール または を使用して変更できます AWS CLI。詳細については、「Amazon DocumentDB インスタンスの変更」を参照してください。

リージョン別のサポートされるインスタンスクラス

Amazon DocumentDB は、次の インスタンスクラスをサポートします。

- R8G— R6G インスタンスと比較してパフォーマンスが最大 30% 向上する Arm ベースの AWS Graviton4 プロセッサを搭載したメモリ最適化インスタンスの最新世代。
- R6G— Arm ベースの AWS Graviton2 プロセッサを搭載したメモリ最適化インスタンス。R5 インスタンスと比較してパフォーマンスが最大 30% 向上し、コストが 5% 削減されます。
- R6GD — エフェメラルデータ用のローカル不揮発性メモリエクスプレス (NVMe) ベースのソリッドステートドライブ (SSD) ストレージを備えたメモリ最適化 R6G インスタンス。
- R5 - 同じインスタンスコストでR4インスタンスよりも最大 100% 優れたパフォーマンスを提供するメモリ最適化インスタンス。
- R4 - 前世代のメモリ最適化インスタンス。
- T4G— Arm ベースの AWS Graviton2 プロセッサを搭載した最新世代の低コストバースト可能な汎用インスタンスタイプ。ベースラインレベルの CPU パフォーマンスを提供し、T3 インスタンスよりも最大 35% 優れた価格パフォーマンスを実現し、CPU 使用率が中程度で使用量が一時的に急増するアプリケーションの実行に最適です。
- T3 - CPUパフォーマンスのベースラインレベルを提供し、必要な限りいつでもCPU使用率をバーストできる、低コストのバースト可能な汎用インスタンスタイプ。

インスタンスクラスの詳細な仕様については、「インスタンスクラスの仕様」を参照してください。

> **Note**
>
> I/O 最適化ストレージは、エンジンバージョン 5.0 および 8.0 (インスタンスベースのクラスター) でのみ使用できます。

特定のインスタンスクラスは、特定のリージョンでサポートされる場合とサポートされない場合があります。次の表に、各リージョンで Amazon DocumentDB によってサポートされるインスタンスクラスを示してあります。

リージョン別のサポートされるインスタンスクラス

リージョン	R8G	R6GD	R6G	R5	R4	T4G	T3	サーバーレス
米国東部 (オハイオ)	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート
米国東部 (バージニア北部)	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート
米国西部 (オレゴン)	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート
アフリカ (ケープタウン)			サポート対象	サポート対象		サポート対象	サポート対象	サポート
南米 (サンパウロ)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート
アジアパシフィック (香港)	サポート対象		サポート対象	サポート対象		サポート対象	サポート対象	サポート
アジアパシフィック (ハイデラバード)			サポート対象	サポート対象		サポート対象	サポート対象	サポート

リージョン	R8G	R6GD	R6G	R5	R4	T4G	T3	サーバーレス
アジアパシフィック (マレーシア)			サポート対象			サポート対象	サポート対象	サポート
アジアパシフィック (ムンバイ)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート
アジアパシフィック (大阪)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート
アジアパシフィック (ソウル)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート
アジアパシフィック (シドニー)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート
アジアパシフィック (ジャカルタ)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート
アジアパシフィック (メルボルン)			サポート対象	サポート対象		サポート対象	サポート対象	サポート
アジアパシフィック (シンガポール)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
アジアパシフィック (タイ)			サポート対象			サポート対象	サポート対象	サポート対象

リージョン	R8G	R6GD	R6G	R5	R4	T4G	T3	サーバーレス
アジアパシフィック (東京)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
カナダ (中部)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート
カナダ西部 (カルガリー)			サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
欧州 (フランクフルト)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート
欧州 (チューリッヒ)		サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
欧州 (アイルランド)	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象	サポート対象
欧州 (ロンドン)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
欧州 (ミラノ)			サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
欧州 (パリ)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象

リージョン	R8G	R6GD	R6G	R5	R4	T4G	T3	サーバーレス
欧州 (スペイン)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
欧州 (ストックホルム)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
メキシコ (中部)			サポート対象			サポート対象	サポート対象	サポート対象
中東 (アラブ首長国連邦)			サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
中国 (北京)	サポート対象	サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
中国 (寧夏)	サポート対象		サポート対象	サポート対象		サポート対象	サポート対象	サポート対象
イスラエル (テルアビブ)			サポート対象	サポート対象		サポート対象	サポート対象	サポート
AWS GovCloud (米国西部)	サポート対象	サポート対象	サポート対象	サポート対象			サポート対象	サポート
AWS GovCloud (米国東部)		サポート対象	サポート対象	サポート対象		サポート対象	サポート対象	サポート対象

インスタンスクラスの仕様

次の表は、各クラスでサポートされているインスタンスタイプなど、Amazon DocumentDB インスタンスクラスの詳細を示しています。各テーブル列の説明については、表の下部を参照してください。

インスタンスクラス	vCPU[1]	メモリ (GiB)[2]	NVMe SSD 階層型キャッシュ (GiB)[3]	最大一時ストレージ (GiB)[4]	ベースライン/バースト帯域幅 (Gbps)[5]	使用エンジン[6]
R8G – Graviton4 に基づく現行世代のメモリ最適化インスタンスクラス R8G は、エンジンバージョン 5.0 および 8.0 でのみ利用可能な最新のインスタンスファミリー (Graviton4) です。						
db.r8g.large	2	16	-	30	0.937/12.5	5.0.0 および 8.0.0
db.r8g.xlarge	4	32	-	60	1.875/12.5	5.0.0 および 8.0.0
db.r8g.2xlarge	8	64	-	121	3.75/15.0	5.0.0 および 8.0.0
db.r8g.4xlarge	16	128	-	243	7.5/15.0	5.0.0 および 8.0.0
db.r8g.8xlarge	32	256	-	488	15	5.0.0 および 8.0.0
db.r8g.12xlarge	48	384	-	732	22	5.0.0 および 8.0.0
db.r8g.16xlarge	64	512	-	987	30	5.0.0 および 8.0.0
db.r8g.24xlarge	96	768	-	1484	40	5.0.0 および 8.0.0

インスタンスクラス	vCPU[1]	メモリ (GiB)[2]	NVMe SSD 階層型キャッシュ (GiB)[3]	最大一時ストレージ (GiB)[4]	ベースライン/バースト帯域幅 (Gbps)[5]	使用エンジン[6]
db.r8g.48xlarge	192	1536	-	2967	50	5.0.0 および 8.0.0
R6G – Current Generation Memory-Optimized Instance Class based on Graviton2						
db.r6g.large	2	16	-	32	0.75/10	4.0.0、5.0.0、および 8.0.0
db.r6g.xlarge	4	32	-	63	1.25 / 10	4.0.0、5.0.0、および 8.0.0
db.r6g.2xlarge	8	64	-	126	2.5/10	4.0.0、5.0.0、および 8.0.0
db.r6g.4xlarge	16	128	-	252	5.0 / 10	4.0.0、5.0.0、および 8.0.0
db.r6g.8xlarge	32	256	-	504	12	4.0.0、5.0.0、および 8.0.0
db.r6g.12xlarge	48	384	-	756	20	4.0.0、5.0.0、および 8.0.0
db.r6g.16xlarge	64	512	-	1008	25	4.0.0、5.0.0、および 8.0.0
R6GD – Current Generation NVMe-backed Instance Class based on Graviton2						

インスタンスクラス	vCPU[1]	メモリ (GiB)[2]	NVMe SSD 階層型キャッシュ (GiB)[3]	最大一時ストレージ (GiB)[4]	ベースライン/バースト帯域幅 (Gbps)[5]	使用エンジン[6]
db.r6gd.xlarge	4	32	173	64	1.25 / 10	5.0.0 および 8.0.0
db.r6gd.2xlarge	8	64	346	128	2.5/10	5.0.0 および 8.0.0
db.r6gd.4xlarge	16	128	694	256	5.0 / 10	5.0.0 および 8.0.0
db.r6gd.8xlarge	32	256	1388	512	12	5.0.0 および 8.0.0
db.r6gd.12xlarge	48	384	2082	768	20	5.0.0 および 8.0.0
db.r6gd.16xlarge	64	512	2776	1024	25	5.0.0 および 8.0.0
R5 – 前の世代のメモリ最適化インスタンスクラス						
db.r5.large	2	16	-	31	0.75/10	3.6.0、4.0.0、5.0.0、8.0.0
db.r5.xlarge	4	32	-	62	1.25 / 10	3.6.0、4.0.0、5.0.0、8.0.0
db.r5.2xlarge	8	64	-	124	2.5/10	3.6.0、4.0.0、5.0.0、8.0.0

インスタンスクラス	vCPU[1]	メモリ (GiB)[2]	NVMe SSD 階層型キャッシュ (GiB)[3]	最大一時ストレージ (GiB)[4]	ベースライン/バースト帯域幅 (Gbps)[5]	使用エンジン[6]
db.r5.4xlarge	16	128	-	249	5.0 / 10	3.6.0、4.0.0、5.0.0、8.0.0
db.r5.8xlarge	32	256	-	504	10	3.6.0、4.0.0、5.0.0、8.0.0
db.r5.12xlarge	48	384	-	748	12	3.6.0、4.0.0、5.0.0、8.0.0
db.r5.16xlarge	64	512	-	1008	20	3.6.0、4.0.0、5.0.0、8.0.0
db.r5.24xlarge	96	768	-	1500	25	3.6.0、4.0.0、5.0.0、8.0.0
R4 – 前の世代のメモリ最適化インスタンスクラス R4 インスタンスはエンジンバージョン 3.6 でのみサポートされています。Amazon DocumentDB 3.6 は 2026 年 3 月 30 日に標準サポートが終了することに注意してください。延長サポート (有料) は 2029 年 3 月までご利用いただけます。						
db.r4.large	2	15.25	-	30	0.75/10	3.6.0 のみ
db.r4.xlarge	4	30.5	-	60	1.25 / 10	3.6.0 のみ
db.r4.2xlarge	8	61	-	120	2.5/10	3.6.0 のみ

インスタンスクラス	vCPU[1]	メモリ (GiB)[2]	NVMe SSD 階層型キャッシュ (GiB)[3]	最大一時ストレージ (GiB)[4]	ベースライン/バースト帯域幅 (Gbps)[5]	使用エンジン[6]
db.r4.4xlarge	16	122	-	240	5.0 /10	3.6.0 のみ
db.r4.8xlarge	32	244	-	480	10	3.6.0 のみ
db.r4.16xlarge	64	488	-	960	25	3.6.0 のみ
T4G – Latest Generation Burstable Performance Instance Classes based on Graviton2 T シリーズインスタンスは、無制限 CPU バーストモードで実行されます。ベースラインを超えるバースト使用量には追加料金がかかります。グローバルクラスターではサポートされていません。						
db.t4g.medium	2	4	-	8.13	0.256 / 5	4.0.0、5.0.0、および 8.0.0
T3 – 最新世代のバースト可能なパフォーマンスインスタンスクラス						
db.t3.medium	2	4	-	7.5	0.256 / 5	3.6.0、4.0.0、5.0.0、8.0.0

インスタンスクラス	vCPU[1]	メモリ (GiB)[2]	NVMe SSD 階層型キャッシュ (GiB)[3]	最大一時ストレージ (GiB)[4]	ベースライン/バースト帯域幅 (Gbps)[5]	使用エンジン[6]

1. vCPU - 仮想中央演算装置 (CPU) の数。仮想 CPU は、インスタンスクラスの比較に使用できる容量の単位です。特定のプロセッサを購入またはリースして数か月から数年間使用する代わりに、時間単位で処理能力をレンタルすることができます。当社の目的は、実際に利用しているハードウェアがどのようなものであろうと、一定量の CPU 能力を安定したかたちで提供することです。
2. メモリ (GiB) - インスタンスに割り当てられた RAM (ギガバイト単位)。通常、メモリと vCPU の比率は一定です。
3. NVMe SSD 階層型キャッシュ — エフェメラルデータを保存するための拡張キャッシュとして割り当てられた、ギガバイト単位で測定される SSD ボリューム上のスペース。このキャッシュは NVMe-backed インスタンスでのみ使用できます。
4. 最大一時ストレージ (GiB) — 非永続的な一時ファイルストレージ用にインスタンスに割り当てられた、ギガバイト単位で測定されるスペース。NVMe-backed インスタンスの場合、このストレージは NVMe ベースの SSD ボリュームでホストされます。他のすべてのインスタンスでは、Amazon Elastic Block Store (EBS) でホストされます。
5. ベースライン/バースト帯域幅 (Gbps) — バースト帯域幅は、1 秒あたりの最大帯域幅をギガビット単位で表します。8 で割ると、ギガバイト/秒でのスループットが得られます。サイズ 4xlarge 以下のインスタンスにはベースライン帯域幅があります。追加需要を満たすために、ネットワーク I/O クレジットメカニズムを使用して、ベースライン帯域幅を超えてバーストすることができます。インスタンスはインスタンスのサイズに応じて、バースト帯域幅を限られた期間 (通常は 5～60 分) 使用できます。
6. 使用エンジン - インスタンスクラスをサポートする Amazon DocumentDB エンジン。

NVMe-backed インスタンス

NVMe-backed インスタンスは、通常のインスタンスのメモリを超える大規模なデータセットを持つアプリケーションに対して、クエリパフォーマンスを最大 7 倍高速化します。これらのインスタンスは、r6gd インスタンスで利用可能なローカル不揮発性メモリエクスプレス (NVMe) ベースのソリッドステートドライブ (SSD) ストレージを活用してエフェメラルデータを保存し、ネットワークベースのストレージアクセスを減らし、読み取りレイテンシーとスループットを向上させます。

ローカル SSD スペースは 2 つのセクションに分かれています。

- 階層型キャッシュ — ローカル SSD の約 73% がデータベースキャッシュとして割り当てられるため、システムはメインメモリのみの最大 5 倍のデータベースページを保存できます。ローカル SSD は第 2 層のキャッシュとして機能し、既存のメモリ内バッファキャッシュは第 1 層のキャッシュのままです。クエリは、バッファキャッシュと SSD キャッシュの両方にミスがある場合にのみ外部ストレージにアクセスします。
- 一時ストレージ — 残りの 27% は非永続的な一時ファイルストレージ用に予約されており、ソートやインデックスビルドなどのリソース集約型オペレーションを含む複雑なクエリに使用されます。通常のインスタンスでは、一時スペースは Amazon Elastic Block Store (EBS) ボリュームにあります。SSD 上のローカルにホストされた一時ストレージは、ソートを含むクエリのレイテンシーを最大 2 倍削減し、インデックスビルドなどのリソース集約型のオペレーションを高速化します。

サポートされる NVMe-backed インスタンスのタイプとその仕様の詳細については、「インスタンスクラスの仕様」を参照してください。

トピック

- NVMe-backed インスタンスの推奨ユースケース
- Amazon DocumentDB での NVMe-backed インスタンスの使用
- NVMe-backed インスタンスのモニタリング

NVMe-backed インスタンスの推奨ユースケース

次のシナリオでは、NVMe-backed インスタンスを使用します。

- 読み取り負荷の高いワークロード — ワークロードの読み取り負荷が高く、データセットがバッファキャッシュよりも大きく、低い `BufferCacheHitRatio` と高い `ReadIOPS` メトリクスが示された場合、NVMe-backed インスタンスはパフォーマンス上の利点を提供します。
- 更新負荷の高いワークロード — ワークロードの更新負荷が高く、ネットワークストレージの読み取りレイテンシーが原因でガベージコレクションが遅れずに対応することができない場合、NVMe-backed インスタンスは問題を軽減するのに役立ちます。

NVMe-backed インスタンスは、次のようなさまざまなユースケースにメリットがあります。

- インターネットスケールアプリケーション — 厳格なパフォーマンスのサービスレベルアグリーメント (SLAs) による支払い処理、請求、e コマースなどのアプリケーションは、NVMe-backed インスタンスのパフォーマンス上の利点を活用できます。
- リアルタイムレポートダッシュボード — メトリクス/データ収集のために数百のクエリを実行するダッシュボードは、NVMe-backed インスタンスの低レイテンシーと高スループットのメリットが得られます。
- 生成 AI アプリケーション — ベクトル検索を使用して数百万のベクトル埋め込みの完全または最近傍を検索するアプリケーションは、NVMe-backed インスタンスの高性能を活用できます。

Amazon DocumentDB での NVMe-backed インスタンスの使用

Amazon DocumentDB の NVMe-backed インスタンスを使用するには:

- Amazon DocumentDB クラスターを作成し、NVMe-backed インスタンスクラスのいずれかを追加します。詳細については、「Amazon DocumentDB クラスターの作成」を参照してください。
- または、NVMe-backed インスタンスクラスのいずれかを使用するように既存の Amazon DocumentDB クラスターを変更します。詳細については、「Amazon DocumentDB クラスターの変更」を参照してください。

さまざまな AWS リージョンで NVMe-backed インスタンスの可用性を確認するには、「」を参照してくださいリージョン別のサポートされるインスタンスクラス。

NVMe-backed インスタンスから通常のインスタンスに切り替える場合は、Amazon DocumentDB インスタンスのデータベースインスタンスクラスを、NVMe ストレージを使用せずに同様のインスタンスクラスに変更します。例えば、現在のインスタンスクラスが「db.r6gd.4xlarge」の場合、「db.r6g.4xlarge」を選択して元に戻します。詳細については、「Amazon DocumentDB クラスターの変更」を参照してください。

NVMe-backed インスタンスのモニタリング

Amazon CloudWatch で使用可能な通常のインスタンスメトリクスに加えて、NVMe-backed インスタンスは、NVMe ベースの SSD ストレージ、IOPS、スループットに固有の追加のメトリクスを出力します。

```
NVMeStorageCacheHitRatio
FreeNVMeStorage
ReadIOPSNVMeStorage
```

```
ReadLatencyNVMeStorage
ReadThroughputNVMeStorage
WriteIOPSNVMeStorage
WriteLatencyNVMeStorage
WriteThroughputNVMeStorage
```

これらのメトリクスの詳細については、「NVMe-backed インスタンスメトリクス」を参照してください

Amazon DocumentDB サブネットグループの管理

Virtual Private Cloud (VPC) は、 専用の仮想ネットワークです AWS アカウント。クラウド内の他の仮想ネットワークから論理的に分離されます AWS 。Amazon DocumentDB クラスターなどの AWS リソースを Amazon VPC に起動できます。VPC の IP アドレス範囲を指定して、サブネットを追加し、セキュリティグループを関連付けて、ルートテーブルを設定できます。

サブネットは、Amazon VPC の IP アドレスの範囲です。 AWS リソースは、指定されたサブネットで起動できます。インターネットに接続する必要があるリソースには、パブリックサブネットを使用します。インターネットに接続する必要がないリソースには、プライベートサブネットを使用します。公開サブネットとプライベートサブネットの詳細については、Amazon Virtual PrivateCloud ユーザーガイド の VPCとサブネットの基本 を参照してください。

DB サブネットグループは VPC に作成するサブネットのコレクションで、クラスター用に指定します。サブネットグループでは、クラスターの作成時に特定の VPC を指定することができます。`default` サブネットグループを使用すると、VPC 内のすべてのサブネットにまたがります。

各 DB サブネットグループには、特定の リージョン内の少なくとも 2 つのアベイラビリティーゾーンにサブネットが必要です。VPC に DB インスタンスを作成するときに、DB サブネットグループを選択する必要があります。Amazon DocumentDB は、その DB サブネットグループと優先アベイラビリティーゾーンを使用して、クラスターに関連付けるサブネットとそのサブネット内の IP アドレスを選択します。プライマリインスタンスに障害が発生した場合、Amazon DocumentDB は対応するレプリカインスタンスを新しいプライマリに昇格させることができます。その後、前のプライマリが配置されていたサブネットの IP アドレスを使用して、新しいレプリカインスタンスを作成します。

Amazon DocumentDB が VPC にインスタンスを作成すると、DB サブネットグループから選択された IP アドレスを使用して、クラスターにネットワークインターフェースが割り当てられます。基になる IP アドレスはフェイルオーバー中に変更される可能性があるため、常に DNS 名を使用してください。詳細については、「Amazon DocumentDB エンドポイント」を参照してください。

独自の VPC とサブネットの作成については、Amazon Virtual PrivateCloud ユーザーガイド の VPC とサブネットの操作 を参照してください。

トピック

- Amazon DocumentDB サブネットグループの作成
- Amazon DocumentDB サブネットグループの説明
- Amazon DocumentDB サブネットグループの変更
- Amazon DocumentDB サブネットグループの削除

Amazon DocumentDB サブネットグループの作成

Amazon DocumentDB クラスターを作成する場合、クラスターを起動するには、Amazon VPC とその Amazon VPC 内の対応するサブネットグループを選択する必要があります。サブネットは、インスタンスの起動に使用するアベイラビリティーゾーンとアベイラビリティーゾーン内の IP 範囲を決定します。

サブネットグループは、名前付きのサブネット (またはAZ) のセットであり、Amazon DocumentDB インスタンスの起動に使用するアベイラビリティーゾーンを指定できます。たとえば、3 つのインスタンスを含むクラスターでは、各インスタンスを別々の AZ にプロビジョニングして高可用性を実現することをお勧めします。これにより、1 つの AZ に障害が発生した場合、1 つのインスタンスにのみ影響します。

Amazon DocumentDB インスタンスは、最大 3 つのアベイラビリティーゾーンにプロビジョニングできます。サブネットグループに 3 つ以上のサブネットがある場合でも、それらのサブネットのうちの 3 つのみを使用して Amazon DocumentDB クラスターを作成できます。サブネットグループを作成するときは、インスタンスをデプロイする 3 つのサブネットのみを選択します。

例：クラスターが作成され、Amazon DocumentDB は AZ {1A、1B、および1C} を選択します。この場合、AZ 1D でインスタンスを作成しようとすると、その API コールは失敗します。ただし、特定の AZ を指定せずにインスタンスを作成した場合、Amazon DocumentDB がお客様に代わって AZ を選択します。Amazon DocumentDB は、アルゴリズムを使用して AZ 間でインスタンスの負荷を分散し、高可用性を実現できるようにします。たとえば、3 つのインスタンスがプロビジョニングされる場合、デフォルトでは 3 つの AZ にプロビジョニングされ、1 つの AZ にまとめてプロビジョニングされません。

ベストプラクティス

- 特に理由がない限り、常に 3 つのサブネットを含むサブネットグループを作成します。これにより、3 つ以上のインスタンスを持つクラスターでは、インスタンスが 3 つの AZ にまたがってプロビジョニングされるため、より高い可用性を実現できます。
- 高可用性を実現するには、常に複数の AZ にインスタンスを分散させます。クラスターのすべてのインスタンスを 1 つの AZ にまとめて配置しないでください。
- フェイルオーバーイベントはいつでも発生する可能性があるため、プライマリインスタンスまたはレプリカインスタンスが常に特定の AZ に存在すると想定しないでください。

サブネットグループの作成方法

AWS マネジメントコンソール または を使用して Amazon DocumentDB サブネットグループ AWS CLI を作成できます。

Using the AWS マネジメントコンソール

Amazon DocumentDB サブネットグループを作成するには、次のステップを使用します。

Amazon DocumentDB サブネットグループの作成方法

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[Subnet groups (サブネットグループ)]、[Create (作成)] の順に選択します。

   > **Tip**
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. [Create subnet group (サブネットグループを作成する)] ページで:

   a. [Subnet group details (サブネットグループの詳細)] セクションで:

      i. 名前 - サブネットグループ用の意味のある名前を入力します。

      ii. [Description] (説明) - サブネットグループの説明を入力します。

   b. [Add subnets (サブネットを追加する)] セクションで:

i. VPC — リストからこのサブネットグループの VPC を選択します。

ii. 次のいずれかを行います。

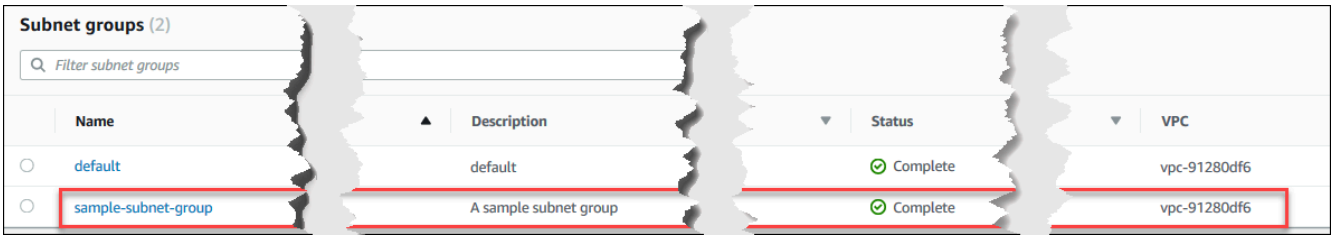
- 選択した VPC にすべてのサブネットを含めるには、[Add all the subnets related to this VPC (この VPC に関連するすべてのサブネットを追加)] を選択します。
- このサブネットグループのサブネットを指定するには、サブネットを含める各アベイラビリティーゾーンに対して次の手順を実行します。少なくとも 2 つのアベイラビリティーゾーンを含める必要があります。

A. アベイラビリティーゾーン - リストからアベイラビリティーゾーンを選択します。

B. サブネット — リストで、このサブネットグループに選択したアベイラビリティーゾーンからサブネットを選択します。

C. [Add subnet (サブネットを追加する)] を選択します。

4. [作成] を選択します。サブネットグループが作成されると、他のサブネットグループとともに表示されます。



Using the AWS CLI

を使用してサブネットグループを作成する前に AWS CLI、最初に使用可能なサブネットを決定する必要があります。次の AWS CLI オペレーションを実行して、アベイラビリティーゾーンとそのサブネットを一覧表示します。

パラメータ :

- **`--db-subnet-group`** - オプションです。特定のサブネットグループを指定すると、そのグループのアベイラビリティーゾーンとサブネットが一覧表示されます。このパラメータを省略すると、すべてのサブネットグループのアベイラビリティーゾーンとサブネットが一覧表示されます。`default` サブネットグループを指定すると、すべての VPC のサブネットが一覧表示されます。

Example

Linux、macOS、Unix の場合:

```
aws docdb describe-db-subnet-groups \
    --db-subnet-group-name default \
    --query 'DBSubnetGroups[*].[DBSubnetGroupName,Subnets[*].
[SubnetAvailabilityZone.Name,SubnetIdentifier]]'
```

Windows の場合:

```
aws docdb describe-db-subnet-groups ^
    --db-subnet-group-name default ^
    --query 'DBSubnetGroups[*].[DBSubnetGroupName,Subnets[*].
[SubnetAvailabilityZone.Name,SubnetIdentifier]]'
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
[
    [
        "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"
            ]
        ]
    ]
]
```

前のオペレーションからの出力を使用して、新しいサブネットグループを作成できます。新しいサブネットグループは、少なくとも 2 つのアベイラビリティーゾーンからのサブネットを含める必要があります。

パラメータ :

- **--db-subnet-group-name** — 必須 このサブネットグループの名前。
- **--db-subnet-group-description** — 必須 このサブネットグループの説明。
- **--subnet-ids** — 必須 このサブネットグループに含めるサブネットのリスト。例えば、subnet-53ab3636 などです。
- --タグ - オプションです。このサブネットグループにアタッチするタグ (キーと値のペア) のリスト。

次のコードは、subnet-4e26d263、subnet-afc329f4、および subnet-b3806e8f の 3 つのサブネットを持つサブネットグループ sample-subnet-group を作成します。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "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"
    }
}
```

Amazon DocumentDB サブネットグループの説明

AWS マネジメントコンソール または を使用して AWS CLI 、Amazon DocumentDB サブネットグループの詳細を取得できます。

Using the AWS マネジメントコンソール

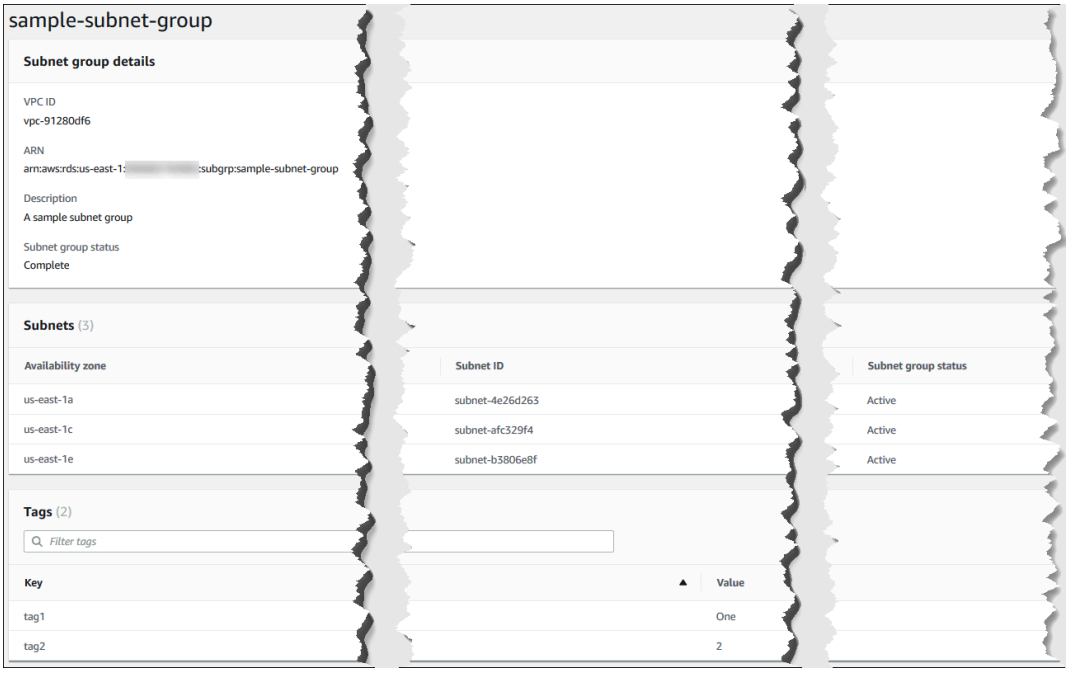
次の手順では、Amazon DocumentDB サブネットグループの詳細を取得する方法を説明します。

サブネットグループの詳細を確認するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [サブネットグループ] を選択します。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. サブネットグループの詳細を表示するには、そのサブネットグループの名前を選択します。



Using the AWS CLI

Amazon DocumentDB サブネットグループの詳細を見つけるには、次のパラメーターを指定した `describe-db-subnet-groups` オペレーションを使用します。

パラメータ

- `--db-subnet=group-name` - オプションです。含まれている場合は、名前付きサブネットグループの詳細が一覧表示されます。省略すると、最大 100 個のサブネットグループの詳細が一覧表示されます。

Example

次のコードは、Amazon DocumentDB サブネットグループの作成 セクションで作成した `sample-subnet-group` サブネットグループの詳細を示しています。

Linux、macOS、Unix の場合:

```
aws docdb describe-db-subnet-groups \
    --db-subnet-group-name sample-subnet-group
```

Windows の場合:

```
aws docdb describe-db-subnet-groups ^
    --db-subnet-group-name sample-subnet-group
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "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"
    }
}
```

Amazon DocumentDB サブネットグループの変更

AWS マネジメントコンソール または を使用して AWS CLI 、サブネットグループの説明を変更したり、Amazon DocumentDB サブネットグループからサブネットを追加または削除したりできます。ただし、default サブネットグループは変更できません。

Using the AWS マネジメントコンソール

を使用して AWS マネジメントコンソール 、サブネットグループの説明を変更したり、サブネットを追加または削除したりできます。完了したら、サブネットグループに少なくとも 2 つのアベイラビリティーゾーンが関連付けられている必要があります。

サブネットグループを変更するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [サブネットグループ] を選択します。次に、サブネットグループ名の左側にあるボタンを選択します。default サブネットグループは変更できないことに注意してください。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン

(≡) を選択します。

3. [アクション] を選択してから、[変更] を選択します。
4. 説明 — サブネットグループの説明を変更するには、新しい説明を入力します。
5. サブネットグループに関連付けられているサブネットを変更するには、[Add subnets (サブネットの追加)] セクションで、次のいずれかを実行します。
   - このサブネットグループからサブネットをすべて削除するには、[Remove all (すべてを削除)] を選択します。
   - このサブネットグループから特定のサブネットを削除するには、削除するサブネットごとに [Remove (削除) を選択します。
   - この VPC に関連付けられているすべてのサブネットを追加するには、[Add all the subnets related to this VPC (このVPCに関連付けられているすべてのサブネットを追加])] を選択します。
   - このサブネットグループに特定のサブネットを追加するには、サブネットを追加する各アベイラビリティーゾーンに対して次の手順を実行します。
     a. アベイラビリティーゾーン - リストから新しいアベイラビリティーゾーンを選択します。
     b. サブネット — リストで、このサブネットグループに選択したアベイラビリティーゾーンからサブネットを選択します。
     c. [Add subnet (サブネットを追加する)] を選択します。
6. 確認ダイアログボックスでは:
   - サブネットグループにこのような変更を行うには、[Modify (変更)] を選択します。
   - サブネットグループを変更しない場合は、[キャンセル] を選択してください。

Using the AWS CLI

を使用して AWS CLI 、サブネットグループの説明を変更したり、サブネットを追加または削除したりできます。完了したら、サブネットグループに少なくとも 2 つのアベイラビリティーゾーンが関連付けられている必要があります。`default` サブネットグループを変更することはできません。

パラメータ :

- `--db-subnet-group-name` — 必須 変更している Amazon DocumentDB サブネットグループの名前。
- `--subnet-ids` — 必須 この変更が完了した後でサブネットグループに必要となるすべてのサブネットの一覧。

> **Important**
>
> このリストに含まれていない、現在サブネットグループにあるサブネットはすべて、サブネットグループから削除されます。現在サブネットグループにあるサブネットのいずれかを保持したい場合は、それらをこのリストに含める必要があります。

- `--db-subnet-group-description` - オプションです。サブネットグループの説明。

Example

次のコードは説明を変更し、既存のサブネットをサブネット `subnet-991cb8d0`、`subnet-53ab3636`、および `subnet-29ab1025` に置き換えます。

Linux、macOS、Unix の場合:

```
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"
```

Windows の場合:

```
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"
```

このオペレーションによる出力は、次のようになります (JSON 形式)。これは、「Amazon DocumentDB サブネットグループの作成」セクションで作成したものと同じサブネットグループであることに注目します。ただし、サブネットグループ内のサブネットは、`modify-db-subnet-group` オペレーションで一覧表示されるものに置き換えられます。

```
{
    "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"
    }
}
```

Amazon DocumentDB サブネットグループの削除

AWS マネジメントコンソール または を使用して AWS CLI 、Amazon DocumentDB サブネットグループを削除できます。ただし、`default` サブネットグループは削除できません。

Using the AWS マネジメントコンソール

を使用して AWS マネジメントコンソール 、サブネットグループを削除できます。ただし、`default` サブネットグループを削除することはできません。

サブネットグループを削除するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [サブネットグループ] を選択します。次に、サブネットグループ名の左側にあるボタンを選択します。`default` サブネットグループは削除できないことに注意してください。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. [アクション]、[削除] の順に選択します。
4. 確認ダイアログボックスでは:
   - サブネットグループを削除するには、[削除] を選択します。
   - サブネットグループを保持するには、キャンセル を選択します。

Using the AWS CLI

を使用して Amazon DocumentDB サブネットグループを削除するには AWS CLI、次のパラメータを指定して `delete-db-subnet-group`オペレーションを使用します。

パラメータ

- `--db-subnet-group-name` — 必須 削除する Amazon DocumentDB サブネットグループの名前。`default` サブネットグループは削除できないことに注意してください。

Example

次のコードは、`sample-subnet-group` を削除します。

Linux、macOS、Unix の場合:

```
aws docdb delete-db-subnet-group \
    --db-subnet-group-name sample-subnet-group
```

Windows の場合:

```
aws docdb delete-db-subnet-group ^
    --db-subnet-group-name sample-subnet-group
```

このオペレーションでは何も出力されません。

Amazon DocumentDB の高可用性とレプリケーション

レプリカインスタンスを使用すると、Amazon DocumentDB で高可用性と読み取りスケールインを実現できます。1 つの Amazon DocumentDB クラスターは、1 つのプライマリインスタンスと 15 までのレプリカインスタンスをサポートします。これらのインスタンスは、クラスターのリージョン内のアベイラビリティーゾーンに分散することができます。プライマリインスタンスでは、読み込みと書き込みトラフィックを受け入れ、レプリカインスタンスは読み込みリクエストのみを受け入れます。

クラスターボリュームはクラスターのデータの複数のコピーで構成されます。ただし、クラスターボリューム内のデータは、プライマリインスタンスおよびクラスター内の Amazon DocumentDB レプリカに対する単一の論理ボリュームとして表されます。レプリカインスタンスには、結果整合性があります。これによって、最短のレプリカラグでクエリ結果を返します。通常の場合、プライマリインスタンスが更新を書き込みしてから 100 ミリ秒未満になります。レプリカラグは、データベースの変更レートによって異なります。つまり、データベースに対して大量の書き込みオペレーションが発生している間、レプリカラグが増加することがあります。

読み取りのスケーリング

Amazon DocumentDB プリカは、クラスターボリュームでの読み取りオペレーションに特化しているため、読み取りのスケーリングに最適です。書き込みオペレーションはプライマリインスタンスによって管理されます。クラスターボリュームは、クラスター内のすべてのインスタンス間で共有されます。したがって、各 Amazon DocumentDB レプリカごとにデータのコピーを複製して維持する必要はありません。

高可用性

Amazon DocumentDB クラスターを作成するときは、サブネットグループ内のアベイラビリティーゾーン の数に応じて (少なくとも 2 つ存在する必要があります)、Amazon DocumentDB は、アベイラビリティゾーン間でインスタンスをプロビジョニングします。クラスター内でインスタンスを作成する場合、Amazon DocumentDB はサブネットグループ内のアベイラビリティゾーン間でインスタンスを自動的に配信して、クラスターのバランスを取ります。また、このアクションは、すべてのインスタンスが同じアベイラビリティーゾーンに配置されることを回避します。

例

この点を説明するために、3 つのアベイラビリティゾーン (AZ1、AZ2、および AZ3) を持つサブネットグループを持つクラスターを作成する例を考えます。

クラスター内の最初のインスタンスが作成されると、それがプライマリインスタンスとなり、いずれかのアベイラビリティゾーンに配置されます。この例では、これが AZ1 です。作成される 2 番目のインスタンスはレプリカインスタンスで、他の 2 つのアベイラビリティゾーン (AZ2 とします) のいずれかにあります。作成される 3 番目のインスタンスはレプリカインスタンスで、残りのアベイラビリティゾーン (AZ3) にあります。さらにインスタンスを作成する場合、クラスター内でバランスが取れるようにアベイラビリティゾーン間で分散します。

プライマリインスタンス (AZ1) で障害が発生すると、フェイルオーバーがトリガーされ、既存のレプリカの 1 つがプライマリに昇格します。古いプライマリティが復旧すると、プロビジョニングされていたアベイラビリティゾーン (AZ1) と同じ場所でレプリカとなります。3 つのインスタンスクラスターをプロビジョニングすると、Amazon DocumentDB はその 3 つのインスタンスクラスターを引き続き保持します。Amazon DocumentDB は、手動による介入なしに、インスタンス障害の検出、フェイルオーバー、および回復を自動的に処理します。

Amazon DocumentDB がフェイルオーバーを実行してインスタンスを復旧すると、復旧したインスタンスはプロビジョニングされていたアベイラビリティーゾーンに残ります。ただし、インスタンスのロールはプライマリからレプリカに変更される場合があります。これを実行することで、一連のフェイルオーバーによってすべてのインスタンスが結果として同じアベイラビリティゾーンになるというシナリオを回避することができます。

フェイルオーバーターゲットとして Amazon DocumentDB レプリカを指定できます。つまり、プライマリインスタンスが失敗した場合、指定された Amazon DocumentDB プリカまたは層からのレプリカがプライマリインスタンスに昇格します。短い中断があり、その間はプライマリインスタンスに対して行われた読み取りおよび書き込みリクエストは、例外により失敗します。Amazon DocumentDB クラスターに Amazon DocumentDB レプリカが含まれていない場合は、プライマリ

インスタンスに障害が発生すると再作成されます。Amazon DocumentDB レプリカを昇格するほうが、プライマリインスタンスの再作成よりも大幅に短時間で行えます。

高可用性を実現するには、異なるアベイラビリティーゾーンのプライマリインスタンスと同じインスタンスクラスの Amazon DocumentDB レプリカを 1 つ以上作成します。

詳細については次を参照してください:

- Amazon DocumentDB クラスターの耐障害性について理解する
- Amazon DocumentDB フェイルオーバー
  - フェイルオーバーターゲットの制御

Amazon DocumentDB バージョン 5.0 および 8.0 では、プライマリインスタンスの再起動時にレプリカインスタンスは再起動しません。プライマリインスタンスの再起動中も読み取りリクエストを処理し続けるため、可用性が向上します。

グローバルクラスターによる高可用性

複数の で高可用性を実現するために AWS リージョン、Amazon DocumentDB グローバルクラスターを設定できます。各グローバルクラスターは複数のリージョンにまたがり、低レイテンシーのグローバル読み取りとリージョン間の停止からのディザスタリカバリを可能にします。Amazon DocumentDB は、プライマリリージョンから各セカンダリリージョンへのすべてのデータと更新のレプリケーションを自動的に処理します。

レプリカの追加

クラスターに追加される最初のインスタンスは、プライマリインスタンスです。最初のインスタンスの後に追加されるすべてのインスタンスはレプリカインスタンスです。クラスターは、プライマリに加えて 15 個までのレプリカインスタンスを持つことができます。

を使用してクラスターを作成すると AWS マネジメントコンソール、プライマリインスタンスが同時に自動的に作成されます。クラスターとプライマリインスタンスを作成すると同時にレプリカを作成するには、[別のゾーンにレプリカを作成します] を選択します。詳細については、「Amazon DocumentDB クラスターの作成」のステップ 4.d を参照してください。Amazon DocumentDB クラスターにさらにレプリカを追加するには、クラスターへの Amazon DocumentDB インスタンスの追加 を参照してください。

を使用してクラスター AWS CLI を作成する場合は、プライマリインスタンスとレプリカインスタンスを明示的に作成する必要があります。詳細については、以下のトピックの「Using the AWS CLI」セクションを参照してください。

- Amazon DocumentDB クラスターの作成
- クラスターへの Amazon DocumentDB インスタンスの追加

Amazon DocumentDB フェイルオーバー

特定のタイプの計画的メンテナンスなどの特定の場合、またはプライマリノードまたはアベイラビリティーゾーンの障害が発生する可能性が低い場合、Amazon DocumentDB (MongoDB 互換) が障害を検出し、プライマリノードを置き換えます。フェイルオーバー中、書き込みのダウンタイムは最小限になります。これは、プライマリノードのロールが新しいプライマリノードを作成してプロビジョニングする代わりに、リードレプリカのいずれかにフェイルオーバーするためです。この障害検出とレプリカの昇格により、昇格が完了したらすぐに新しいプライマリへの書き込みを再開できます。

フェイルオーバーが機能するためには、クラスターには少なくとも 2 つのインスタンス、プライマリおよび少なくとも 1 つのレプリカインスタンスが必要です。

> **Note**
>
> このトピックは、元の Amazon DocumentDB インスタンスベースのクラスターにのみ適用されます。Elastic クラスターまたはグローバルクラスターには適用されません。

フェイルオーバーターゲットの制御

Amazon DocumentDB は、フェイルオーバーが発生した場合にプライマリに昇格されるレプリカインスタンスを制御するための手段として、フェイルオーバー階層を提供します。

フェイルオーバー階層

各レプリカインスタンスは、フェイルオーバー階層 (0 ～ 15) と関連付けられます。メンテナンスのためフェイルオーバーが発生するか、予期しないハードウェア障害が発生した場合、プライマリインスタンスは優先度の最も高い (番号が最も小さい層) レプリカにフェイルオーバーします。優先度が同じ階層を持つ複数のレプリカがある場合、プライマリは、プライマリに最も近いサイズのその階層のレプリカにフェイルオーバーします。

選択したレプリカのグループのフェイルオーバー階層を 0 (最大の優先度) に設定することで、フェイルオーバーでそのグループ内のいずれかのレプリカを昇格することができます。フェイルオーバーが発生した場合に、優先度の低い階層 (高い番号) をこれらのレプリカに割り当てることで、特定のレプリカのプライマリへの昇格を実質的に防ぐことができます。これは、特定のレプリカがアプリケーションによって多く使用され、それらの 1 つへのフェイルオーバーによって重要なアプリケーションに悪影響を与える可能性がある状況で役立ちます。

潜在的なフェイルオーバーターゲットであるレプリカは、プライマリインスタンスと同じインスタンスクラスを使用することをお勧めします。これにより、フェイルオーバーが OS 更新やその他のメンテナンス中にレプリカを昇格させる場合、書き込みパフォーマンスの低下を回避できます。

インスタンスのフェイルオーバー階層は、インスタンスの作成時に設定するか、後でインスタンスを変更して設定できます。インスタンスを変更してインスタンスのフェイルオーバーを設定しても、フェイルオーバーはトリガーされません。詳細については、以下のトピックを参照してください。

- クラスターへの Amazon DocumentDB インスタンスの追加
- Amazon DocumentDB インスタンスの変更

手動でフェイルオーバーを開始するときは、プライマリに昇格するレプリカインスタンスを制御するための 2 つの方法として、先ほど説明したフェイルオーバー階層と `--target-db-instance-identifier` パラメータがあります。

`--target-db-instance-identifier`

テストでは、`failover-db-cluster` オペレーションを使用してフェイルオーバーイベントを強制できます。`--target-db-instance-identifier` パラメータを使用して、プライマリに昇格させるレプリカを指定します。`--target-db-instance-identifier` パラメータの使用は、フェイルオーバー優先度階層よりも優先されます。`--target-db-instance-identifier` パラメータを指定しなかった場合、プライマリフェイルオーバーはフェイルオーバー優先度階層に従います。

フェイルオーバー中の処理

フェイルオーバーは Amazon DocumentDB によって自動的に処理されるため、アプリケーションは管理上の介入なく、可能な限り迅速にデータベースオペレーションを再開することができます。

- Amazon DocumentDB プリカが同じアベイラビリティーゾーンや別のアベイラビリティーゾーンにあると、フェイルオーバーが発生した場合、Amazon DocumentDB はインスタンスの正規名レコード (CNAME) を切り替えて正常なレプリカを指定します。これに伴い、この正常なレプリカが

新しいプライマリに昇格されます。通常、フェイルオーバーは開始から終了まで 30 秒以内に完了します。

- Amazon DocumentDB レプリカインスタンス (単一インスタンスのクラスターなど) がない場合、Amazon DocumentDB は元のインスタンスと同じアベイラビリティーゾーンに新しいインスタンスを作成しようとします。このような元のインスタンスの置換処理はベストエフォート方式で行われるため、アベイラビリティーゾーンの広範囲に影響する問題がある場合などは失敗する可能性があります。

接続が切断された場合、アプリケーションはデータベースへの接続を再試行する必要があります。

フェイルオーバーのテスト

クラスターのフェイルオーバーにより、クラスター内の Amazon DocumentDB レプリカの 1 つ (読み取り専用インスタンス) が、プライマリインスタンス (クラスターライター) に昇格されます。

Amazon DocumentDB は、プライマリインスタンスが失敗した場合に、存在する場合、自動的に Amazon DocumentDB レプリカにフェイルオーバーします。テストのため、プライマリインスタンスの失敗をシミュレートする場合は、フェイルオーバーを強制できます。クラスター内の各インスタンスには、独自のエンドポイントアドレスがあります。したがって、フェイルオーバーの完了時にこれらのエンドポイントアドレスを使用するすべての既存の接続をクリーンアップして再度確立する必要があります。

フェイルオーバーを強制するには、これらのパラメータを指定して、`failover-db-cluster` オペレーションを使用します。

- `--db-cluster-identifier` — 必須 フェイルオーバーするクラスターの名前。
- `--target-db-instance-identifier` - オプションです。プライマリインスタンスに昇格するインスタンスの名前。

Example

以下のオペレーションでは、`sample-cluster` クラスターのフェイルオーバーを強制します。新しいプライマリインスタンスを作成するインスタンスが指定されないため、Amazon DocumentDB はフェイルオーバー階層の優先度に従ってインスタンスを選択します。

Linux、macOS、Unix の場合:

```
aws docdb failover-db-cluster \
```

```
    --db-cluster-identifier sample-cluster
```

Windows の場合:

```
aws docdb failover-db-cluster ^
    --db-cluster-identifier sample-cluster
```

以下のオペレーションでは、sample-cluster クラスターのフェイルオーバーを強制し、sample-cluster-instance がプライマリロールに昇格することを指定します。(出力の "IsClusterWriter": true に注目してください。)

Linux、macOS、Unix の場合:

```
aws docdb failover-db-cluster \
    --db-cluster-identifier sample-cluster \
    --target-db-instance-identifier sample-cluster-instance
```

Windows の場合:

```
aws docdb failover-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --target-db-instance-identifier sample-cluster-instance
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "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"
    }
}
```

レプリケーションの遅延

レプリケーションラグは通常 50 ms 以下です。レプリカのラグが上がる最も一般的な理由は次のとおりです。

- リードレプリカがプライマリより遅くなる原因として、プライマリの書き込みレートが高いです。
- 長時間実行されるクエリ (大規模なシーケンシャルスキャン、集約クエリなど) と着信書き込みレプリケーションの間のリードレプリカの競合。
- リードレプリカの同時クエリが非常に多いです。

レプリケーションの遅延を最小限に抑えるには、次のトラブルシューティング方法を試してください。

- 書き込みレートが高いか CPU 使用率が高い場合は、クラスター内のインスタンスをスケールアップします。
- リードレプリカで長時間実行されるクエリがあり、クエリ対象のドキュメントが非常に頻繁に更新される場合は、リードレプリカでの競合を回避するために、長時間実行されるクエリを変更するか、プライマリ/書き込みレプリカに対して実行することを検討してください。
- 同時クエリが非常に多い場合や、リードレプリカでのみ高い CPU 使用率がある場合、別のオプションは、リードレプリカの数をスケールアウトしてワークロードを分散することです。
- レプリケーションラグは高い書き込みスループットと長時間実行されるクエリに起因するため、`DBClusterReplicaLagMaximum`CloudWatch メトリクスとスロークエリロガーおよび `WriteThroughput`/`WriteIOPS` メトリクスを組み合わせてレプリケーションラグをトラブルシューティングします。

クラスターのフェイルオーバーによってパフォーマンスが低下しないように、すべてのレプリカが同じインスタンスタイプであることを確認します。

スケールアップとスケールアウト (例：6 つの小さいインスタンスと 3 つの大きなインスタンス) を選択する場合は、DB インスタンスごとに大きなバッファーキャッシュが得られるため、スケールアウト前に最初に (より大きなインスタンス) にスケールアップすることをお勧めします。

プロアクティブに、レプリケーション・ラグ・アラームを設定し、アプリケーションの機能に影響を及ぼす前に、しきい値をレプリカ・インスタンス上のデータがどの程度遅れる (または「古い」) 上限であると感じる値に設定する必要があります。一般に、一時的なワークロードが原因で、アラームが発生する前に、複数のデータポイントについてレプリケーションラグのしきい値を超えることをお勧めします。

> **Note**
>
> 10 秒を超えるレプリケーションラグに別のアラームを設定します。複数のデータポイントでこのしきい値を超えた場合は、インスタンスをスケールアップするか、プライマリインスタンスの書き込みスループットを減らします。

Amazon DocumentDB イベントの管理

Amazon DocumentDB (MongoDB との互換性あり) は、クラスター、インスタンス、スナップショット、セキュリティグループ、およびクラスターパラメータグループに関連するイベントのレコードを保持します。この情報には、イベントの日付と時刻、イベントのソース名とソースタイプ、イベントに関連付けられたメッセージなどがあります。

> **Important**
>
> 特定の管理機能のために、Amazon DocumentDB は Amazon RDS および Amazon Neptune と共有したオペレーションテクノロジーを使用します。Amazon DocumentDB、Amazon RDS、および Amazon Neptune 間でリージョンレベルで管理されている制限のあるリージョンが共有されています。詳細については、「リージョン別クォータ」を参照してください。

トピック

- Amazon DocumentDB のイベントカテゴリの表示
- Amazon DocumentDB イベントの表示

Amazon DocumentDB のイベントカテゴリの表示

各 Amazon DocumentDB リソースタイプには、関連付けることができる特定のタイプのイベントがあります。オペレーションを使用して AWS CLI `describe-event-categories`、イベントタイプと Amazon DocumentDB リソースタイプのマッピングを表示できます。

パラメータ

- **`--source-type`** - オプションです。特定のソースタイプのイベントカテゴリを表示するには、`--source-type` パラメータを使用します。使用できる値を以下に示します。

- db-cluster
- db-instance
- db-parameter-group
- db-security-group
- db-cluster-snapshot

- **--filters** - オプションです。Amazon DocumentDB のイベントカテゴリを表示するには、フィルター --filter Name=engine,Values=docdb を使用します。

Example

次のコードは、クラスターに関連付けられているイベントカテゴリを一覧表示しています。

Linux、macOS、Unix の場合:

```
aws docdb describe-event-categories \
    --filter Name=engine,Values=docdb \
    --source-type db-cluster
```

Windows の場合:

```
aws docdb describe-event-categories ^
    --filter Name=engine,Values=docdb ^
    --source-type db-cluster
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "EventCategoriesMapList": [
        {
            "EventCategories": [
                "notification",
                "failure",
                "maintenance",
                "failover"
            ],
            "SourceType": "db-cluster"
        }
    ]
```

```
}
```

次のコードは、各 Amazon DocumentDB ソースタイプに関連付けられているイベントカテゴリを一覧表示しています。

```
aws docdb describe-event-categories
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

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

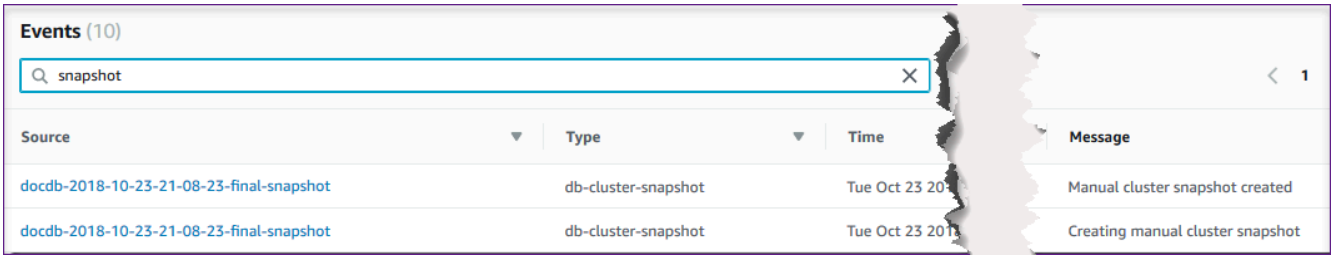
Amazon DocumentDB イベントの表示

Amazon DocumentDB コンソールを使用して Amazon DocumentDB リソースのイベントを取得し、過去 24 時間のイベントを確認できます。また、Amazon DocumentDB リソースのイベントを取得するには、describe-events AWS CLI コマンド、またはDescribeEvents Amazon DocumentDB API オペレーションを使用します。 AWS CLI または Amazon DocumentDB API を使用してイベントを表示する場合、過去 14 日間のイベントを取得できます。

Using the AWS マネジメントコンソール

過去 24 時間で発生したAmazon DocumentDB インスタンスのすべてのイベントを表示するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインの [Events] を選択してください。使用できるイベントがリストに表示されます。
3. タイプ別にイベントをフィルタリングするには、[フィルター] リストを使用します。テキストボックスにキーワードを入力して、結果をさらに絞り込むこともできます。例えば、次のスクリーンショットは、スナップショット イベントのすべての Amazon DocumentDB イベントをフィルタリングすることを示しています



Using the AWS CLI

過去 7 日間で発生したAmazon DocumentDB インスタンスのすべてのイベントを表示するには

過去 7 日間のすべての Amazon DocumentDB インスタンスイベントを表示するには、 `--duration`パラメータを `10080` (10,080 分) に設定して describe-events AWS CLI オペレーションを実行します。

```
aws docdb describe-events --duration 10080
```

Amazon DocumentDB イベントのフィルタリング

特定の Amazon DocumentDB イベントを表示するには、以下のパラメータを指定して、`describe-events` オペレーションを使用します。

パラメータ

- **`--filter`** - Amazon DocumentDB イベントに対する戻り値を制限するために必要です。**`Name=engine,Values=docdb`** を使用して、Amazon DocumentDB だけのすべてのイベントをフィルタリングします。
- **`--source-identifier`** - オプションです。イベントが返されるイベントソースの識別子。省略した場合、すべてのソースからのイベントが結果に含まれます。
- **`--source-type`** - オプションです。`--source-identifier` が指定されていない限り必須。`--source-identifier` が指定されている場合、`--source-type` は `--source-identifier` の種類と一致する必要があります。使用できる値を以下に示します。
  - `db-cluster`
  - `db-instance`
  - `db-parameter-group`
  - `db-security-group`
  - `db-cluster-snapshot`

次の例では、すべての Amazon DocumentDB イベントを一覧表示します。

```
aws docdb describe-events --filters Name=engine,Values=docdb
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

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

詳細については、「Amazon DocumentDB イベントの監査」を参照してください。

Amazon DocumentDB クラスターパラメータグループの管理

クラスターパラメータグループのパラメータを使用して Amazon DocumentDB エンジンの設定を管理できます。クラスターパラメータグループ は、Amazon DocumentDB クラスターのパラメータ管理を容易にする Amazon DocumentDB の設定値のコレクションです。クラスターパラメータグループは、クラスター内のすべてのインスタンスに適用されるエンジン設定値のコンテナとして機能します。

このセクションでは、クラスターパラメータグループを作成、表示、および変更する方法について説明します。また、任意のクラスターに関連付けられているクラスターパラメータグループを判別する方法も示します。

トピック

- Amazon DocumentDB クラスターパラメータグループについて説明します。
- Amazon DocumentDB クラスターパラメータグループを作成する
- Amazon DocumentDB クラスターパラメータグループを変更する
- カスタムクラスターパラメータグループを使用するために Amazon DocumentDB クラスターを変更する
- Amazon DocumentDB クラスターパラメータグループをコピーする
- Amazon DocumentDB クラスターパラメータグループをリセットする
- Amazon DocumentDB クラスターパラメータグループを削除する
- Amazon DocumentDB クラスターパラメータリファレンス

Amazon DocumentDB クラスターパラメータグループについて説明します。

`default` クラスターパラメータグループは、新しい で最初の Amazon DocumentDB クラスターを作成する AWS リージョン か、新しいエンジンを使用しているときに自動的に作成されます。同じリージョンで作成され、同じエンジンバージョンを持つ後続のクラスターは、`default`クラスターパラメータグループで作成されます。

トピック

- Amazon DocumentDB クラスターパラメータグループの詳細について説明します。
- Amazon DocumentDB クラスターのパラメータグループの確認をする
- Amazon DocumentDB クラスターパラメータグループに関連付けられたクラスターとインスタンスの決定

Amazon DocumentDB クラスターパラメータグループの詳細について説明します。

特定のクラスターパラメータグループの詳細を記述するには、 AWS マネジメントコンソール または AWS Command Line Interface () を使用して次の手順を実行しますAWS CLI。

Using the AWS マネジメントコンソール

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。

   > **Tip**
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. [クラスターパラメータグループ] ペインで、詳細情報を表示するパラメータグループの名前を選択します。
4. 表示されるページには、パラメータグループのパラメータ、最近のアクティビティ、タグが表示されます。

   - [クラスターパラメータ] には、パラメータの名前、現在の値、許容値、パラメータが変更可能かどうか、適用タイプ、データタイプ、説明が表示されます。個々のパラメータを修正するには、パラメータを選択し、クラスターパラメータ セクションの編集 を選択します。詳細については、「Amazon DocumentDB クラスターパラメータを変更する」を参照してください。
   - [最近のイベント] で、このパラメータグループの最新のイベントを確認できます。このセクションの検索バーを使用して、これらのイベントをフィルタリングできます。詳細については、「Amazon DocumentDB イベントの管理」を参照してください。

- [タグ] で、このクラスターパラメータグループのタグを表示できます。タグを追加、または削除するには、編集 を タグ のセクションで選択します。詳細については、「Amazon DocumentDB リソースへのタグ付け」を参照してください。

Using the AWS CLI

`describe-db-cluster-parameter-groups` AWS CLI コマンドを使用して、Amazon DocumentDB 用に持っている単一のクラスターパラメータグループまたはすべてのクラスターパラメータグループの Amazon リソースネーム (ARN)、ファミリー、説明、および名前を表示できます。`describe-db-cluster-parameters` AWS CLI コマンドを使用して、単一のクラスターパラメータグループ内のパラメータとその詳細を表示することもできます。

- **`--describe-db-cluster-parameter-groups`** - すべてのクラスターパラメータグループとその詳細の一覧を表示するには。
  - **`--db-cluster-parameter-group-name`** - オプション。説明するクラスターパラメータグループの名前。このパラメータを省略した場合、すべてのクラスターパラメータグループが説明されます。
- **`--describe-db-cluster-parameters`** - パラメータグループ内のすべてのパラメータとその詳細の一覧を表示するには。
  - **`--db-cluster-parameter-group name`** — 必須。説明するクラスターパラメータグループの名前。

Example

次のコードは、最大 100 のクラスターパラメータグループと、その ARN、ファミリー、説明、および名前を表示します。

```
aws docdb describe-db-cluster-parameter-groups
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
          "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

次のコードは、sample-parameter-group の ARN、ファミリー、説明、および名前を示しています。

Linux、macOS、Unix の場合:

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

Windows の場合:

```
aws docdb describe-db-cluster-parameter-groups ^
        --db-cluster-parameter-group-name sample-parameter-group
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
        "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

次のコードは、sample-parameter-group のパラメータの値を表示します。

Linux、macOS、Unix の場合:

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

Windows の場合:

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

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
   "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",
```

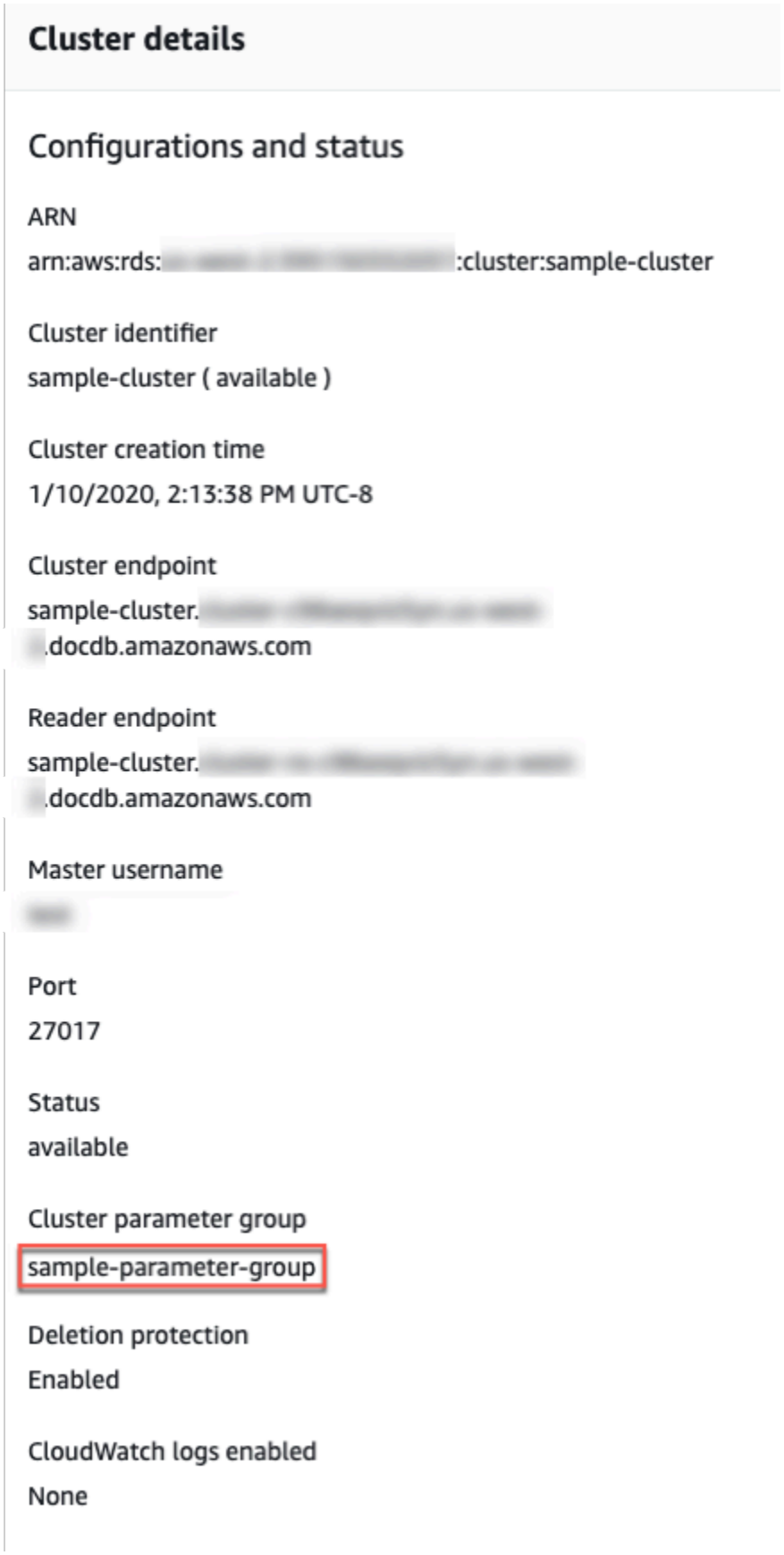
```
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot"
        }
    ]
}
```

Amazon DocumentDB クラスターのパラメータグループの確認をする

特定のクラスターに関連付けられているパラメータグループを確認するには、 AWS マネジメントコンソール または を使用して次の手順を実行します AWS CLI。

Using the AWS マネジメントコンソール

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. 左のナビゲーションペインで [クラスター] を選択します。
3. クラスターの一覧で、関心のあるクラスターの名前を選択します。
4. 結果のページには、選択したクラスターの詳細が表示されます。[クラスターの詳細] まで下にスクロールします。そのセクションの下部で、[クラスターパラメータグループ] の下にパラメータグループの名前を見つけます。


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

次の AWS CLI コードは、クラスター を管理しているパラメータグループを決定しますsample-cluster。

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

このオペレーションによる出力は、次のようになります (JSON 形式)。

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

Amazon DocumentDB クラスターパラメータグループに関連付けられたクラスターとインスタンスの決定

各パラメータグループに関連付けられているクラスターとインスタンスの数を確認するには、 を使用して次の手順を実行します AWS CLI。

Using the AWS CLI

次の AWS CLI コードは、 に関連付けられているクラスターとインスタンスの数を決定しますsample-parameter-group。

Linux、macOS、Unix の場合:

```
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
      }) | .[]'
```

Windows の場合:

```
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
      }) | .[]'
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
   "parameter_group_name": "sample-parameter-group",
   "total_clusters": 50,
   "total_instances": 150
}
```

Amazon DocumentDB クラスターパラメータグループを作成する

default.docdb5.0、、 default.docdb4.0などのデフォルトのクラスターパラメータグループはdefault.docdb3.6、新しいエンジンバージョンと でクラスターを作成するときに作成されます AWS リージョン。同じリージョンで同じエンジンバージョンで作成された後続のクラスターは、defaultクラスターパラメータグループを継承します。一度作成した default パラメータグループは、削除したり名前を変更したりすることはできません。設定したパラメータ値を使用してカスタムパラメータグループを作成し、それを Amazon DocumentDB クラスターにアタッチすることで、クラスターインスタンスのエンジン動作を変更できます。

次の手順では、 カスタムクラスターパラメータグループを作成する方法について説明します。その後、そのパラメータグループ内のパラメータを変更できます。

> **Note**
>
> クラスターパラメータグループの作成後、その特定のパラメータグループを使用するまで、5 分以上お待ちください。これにより、クラスターパラメーターグループが新しいクラスターに使用される前に、Amazon DocumentDB が create アクションを完全に完了さ

せることができます。 AWS マネジメントコンソール または `describe-db-cluster-parameter-groups` AWS CLI オペレーションを使用して、クラスターパラメータグループが作成されたことを確認できます。詳細については、「Amazon DocumentDB クラスターパラメータグループについて説明します。」を参照してください。

Using the AWS マネジメントコンソール

クラスターパラメータグループを作成するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。

   > **Tip**
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. [クラスターパラメータグループ] ペインで、[作成] を選択します。
4. [クラスターパラメータグループの作成] ペインで、次のように入力します。

   a. 新しいクラスターパラメータグループ名 - クラスターパラメーターグループの名前を入力します。例えば、`sample-parameter-group`。クラスターパラメータグループには、次のような命名に関する制約があります。

      - 長さは [1 ~ 255] 個の英数字です。
      - 1 字目は英字である必要があります。
      - ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。

   b. ファミリー — クラスターに使用する DocumentDB バージョンを選択します。

   c. 説明 - このクラスターパラメーターグループの説明を入力します。
5. クラスターパラメータグループを作成するには、[作成] を選択します。操作をキャンセルするには、[キャンセル] を選択します。
6. [作成] を選択すると、ページの上部に次のテキストが表示され、クラスターパラメータグループが正常に作成されたことを確認できます。

```
Successfully created cluster parameter group 'sample-parameter-group'.
```

Using the AWS CLI

Amazon DocumentDB 4.0 クラスターの新しいクラスターパラメータグループを作成するには、次のパラメータを指定して オペレーションを使用します AWS CLI `create-db-cluster-parameter-group`。

- **`--db-cluster-parameter-group-name`** - カスタムクラスターパラメータグループの名前です。例えば、`sample-parameter-group`。
- **`--db-cluster-parameter-group-family`** - カスタムクラスターパラメータグループのテンプレートとして使用されるクラスターパラメーターグループファミリーです。
- **`--description`** - このクラスターパラメータグループのユーザーが提供する説明です。次の例では、「`Custom docdb4.0 parameter group`」を使用しています。

Linux、macOS、Unix の場合:

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"
```

Windows の場合:

```
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"
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "sample-parameter-group",
```

```
        "DBParameterGroupFamily": "docdb5.0",
        "Description": "Custom docdb4.0 parameter group",
        "DBClusterParameterGroupArn": "sample-parameter-group-arn"
    }
}
```

Amazon DocumentDB クラスターパラメータグループを変更する

このセクションでは、カスタム Amazon DocumentDB パラメータグループを変更する方法について説明します。Amazon DocumentDB では、新しいリージョンで新しいエンジンバージョンを持つ`default`クラスターを初めて作成するときに作成されるクラスターパラメータグループを変更することはできません。Amazon DocumentDB クラスターがデフォルトのクラスターパラメータグループを使用していて、その中の値を変更する場合は、まず 新しいパラメータグループを作成 するか、既存のパラメータグループをコピーし、それを変更してから、変更されたパラメータグループをクラスターに適用します。

カスタムクラスターパラメータグループを変更するには、以下のステップを実行します。変更アクションが反映されるまでに時間がかかる場合があります。変更されたクラスターパラメータグループが使用可能になるまで待ってから、クラスターにアタッチします。 AWS マネジメントコンソールまたは AWS CLI `describe-db-cluster-parameters` オペレーションを使用して、クラスターパラメータグループが変更されたことを確認できます。詳細については、「クラスターパラメータグループの説明」を参照してください。

Using the AWS マネジメントコンソール

以下のステップに従って、カスタム Amazon DocumentDB パラメータグループを変更します。`default` パラメータグループを変更することはできません。`default` パラメータグループの値を変更する場合は、デフォルトのパラメータグループをコピーして、変更し、次に変更したパラメータグループをクラスターに適用します。クラスターへのパラメータグループの適用の詳細については、「Amazon DocumentDB クラスターの変更」を参照してください。

カスタムのクラスターパラメータグループを変更するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. コンソールの左側のナビゲーションペインで、[パラメータグループ] を選択します。パラメータグループのリストで、変更するパラメータグループの名前を選択します。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. 変更するパラメータグループの各パラメータに対して、以下の操作を行います。

   a. 修正したいパラメータを探し、変更可能 列の下に `true` としてリストされているかどうかを確認することで、それが変更可能であることを確認します。

   b. 変更可能な場合は、パラメータを選択し、コンソールページの右上にある [編集] を選択します。

   c. [**<parameter-name>** の変更] ダイアログボックスで、必要な変更を行います。次に、[クラスターパラメータの変更] を選択するか、[キャンセル] を選択して変更を破棄します。

Using the AWS CLI

カスタムAmazon DocumentDB クラスターパラメータグループの変更可能なパラメータの `ParameterValue`、`Description`、または `ApplyMethod` は、 AWS CLIを使用して変更することができます。デフォルトのクラスターパラメータグループに対して直接変更を加えることはできません。

カスタムクラスターパラメータグループのパラメータを変更するには、以下のパラメータを指定して `modify-db-cluster-parameter-group` オペレーションを使用します。

- **`--db-cluster-parameter-group-name`** — 必須。変更するクラスターパラメータグループの名前。
- **`--parameters`** — 必須。変更するパラメータ。Amazon DocumentDB クラスター内のすべてのインスタンスに適用されるパラメータの一覧は、Amazon DocumentDB クラスターパラメータリファレンス を参照してください。各パラメータの入力には以下を含める必要があります。
  - **`ParameterName`** - 変更するパラメータの名前です。
  - **`ParameterValue`** - このパラメータの新しい値です。
  - **`ApplyMethod`** - このパラメータの変更を適用する方法です。有効な値は、`immediate` および `pending-reboot` です。

> **Note**
>
> ApplyType が static であるパラメータでは、ApplyMethod が pending-reboot である必要があります。

Example- パラメータの値を変更する

この例では、sample-parameter-group のパラメータ値を一覧表示し、tls のパラメータを変更します。次に、5 分間待機したら、sample-parameter-group のパラメータ値もう一度一覧表示して、変更されたパラメータ値を確認します。

1. sample-parameter-group のパラメータとその値を一覧表示します。

   Linux、macOS、Unix の場合:

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

   Windows の場合:

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

   このオペレーションによる出力は、次のようになります (JSON 形式)。

   ```
   {
          "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. tls パラメータを変更し、その値が disabled になるようにします。

   ApplyType が static であるため、ApplyMethod を変更することはできません。

   Linux、macOS、Unix の場合:

```
aws docdb modify-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group \
       --parameters
 "ParameterName"=tls,"ParameterValue"=disabled,"ApplyMethod"=pending-reboot
```

   Windows の場合:

```
aws docdb modify-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group ^
       --parameters
 "ParameterName"=tls,"ParameterValue"=disabled,"ApplyMethod"=pending-reboot
```

   このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
       "DBClusterParameterGroupName": "sample-parameter-group"
   }
```

3. 少なくとも 5 分お待ち下さい。
4. sample-parameter-group のパラメータ値を表示して、tls パラメータが変更されたことを確認します。

Linux、macOS、Unix の場合:

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

Windows の場合:

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

このオペレーションによる出力は、次のようになります (JSON 形式)。

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

カスタムクラスターパラメータグループを使用するために Amazon DocumentDB クラスターを変更する

Amazon DocumentDB クラスターを作成すると、そのクラスター用に `default.docdb4.0` パラメータグループが自動的に作成されます。`default` クラスターパラメータグループは変更できません。代わりに、Amazon DocumentDB クラスターを変更して、新しいカスタマイズされたパラメータグループをそれに関連付けることができます。

このセクションでは、 AWS マネジメントコンソール と AWS Command Line Interface () を使用して、既存の Amazon DocumentDB クラスターを変更してカスタムクラスターパラメータグループを使用する方法について説明しますAWS CLI。

Using the AWS マネジメントコンソール

Amazon DocumentDB クラスターを変更して、新しいデフォルトでないクラスターパラメータグループを使用するには

1. 開始する前に、Amazon DocumentDB クラスターとクラスターパラメータグループが作成されていることを確認してください。詳細については、「Amazon DocumentDB クラスターの作成」および「Amazon DocumentDB クラスターパラメータグループを作成する」を参照してください。
2. クラスターパラメータグループを作成した後、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。ナビゲーションペインで、[クラスター] を選択し、新しいパラメータグループをクラスターに追加します。
3. パラメータグループを関連付けるクラスターを選択します。[アクション] を選択し、[変更] を選択してクラスターを変更します。
4. [クラスターオプション] で、クラスターを関連付ける新しいパラメータグループを選択します。
5. [続行] を選択して、変更の概要を表示します。
6. 変更を確認したら、直ちに、あるいは、[変更のスケジューリング] の下にある次のメンテナンスウィンドウ中に適用することができます。
7. [クラスターの変更] を選択して、新しいパラメータグループでクラスターを更新します。

Using the AWS CLI

開始する前に、Amazon DocumentDB クラスターと、クラスターパラメータグループが作成されていることを確認してください。オペレーションを使用して AWS CLI `create-db-cluster`

Amazon DocumentDB クラスターを作成できます。create-db-cluster-parameter-group オペレーションを使用して AWS CLI クラスターパラメータグループを作成できます。

新しいクラスターパラメータグループをクラスターに追加するには、次のパラメータを指定して AWS CLI modify-db-clusterオペレーションを使用します。

- --db-cluster-identifier - クラスターの名前 (例えば、sample-cluster) です。
- --db-cluster-parameter-group-name - クラスターを関連付けるパラメータグループの名前 (例えば、sample-parameter-group) です。

Example

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --db-cluster-parameter-group-name sample-parameter-group
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
"DBCluster": {
    "AvailabilityZones": [
            "us-west-2c",
            "us-west-2b",
            "us-west-2a"
    ],
    "BackupRetentionPeriod": 1,
    "DBClusterIdentifier": "sample-cluster",
    "DBClusterParameterGroup": "sample-parameter-group",
    "DBSubnetGroup": "default",
   ...
}
```

Amazon DocumentDB クラスターパラメータグループをコピーする

Amazon DocumentDB のクラスターパラメータグループのコピーは、 AWS マネジメントコンソール または AWS Command Line Interface () を使用して作成できますAWS CLI。

Using the AWS マネジメントコンソール

以下の手順では、既存のクラスターパラメータグループのコピーを作成して、新しいクラスターパラメータグループを作成する手順について説明します。

クラスターパラメータグループをコピーするには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。
3. [クラスターパラメータグループ] ペインで、コピーするクラスターパラメータグループの名前を選択します。
4. [アクション] を選択し、[コピー] を選択して、そのパラメータグループをコピーします。
5. [コピーのオプション] で、新しいクラスターパラメータグループの名前と説明を入力します。次に、[コピー] を選択して変更を保存します。

Using the AWS CLI

クラスターパラメータグループのコピーを作成するには、以下のパラメータを指定して `copy-db-cluster-parameter-group` オペレーションを使用します。

- **`--source-db-cluster-parameter-group-identifier`** — 必須。コピーを作成するクラスターパラメータグループの名前または Amazon リソースネーム (ARN)。

  ソースクラスターパラメータグループとターゲットクラスターパラメータグループが同じである場合 AWS リージョン、識別子は名前または ARN のいずれかになります。

  ソースクラスターパラメータグループとターゲットクラスターパラメータグループが異なる場合 AWS リージョン、識別子は ARN である必要があります。
- **`--target-db-cluster-parameter-group-identifier`** — 必須。クラスターパラメータグループコピーの名前または ARN。

  制約:

  - null、空、または空白にすることはできません。
  - 1 ~ 255 の英字、数字、またはハイフンを含む必要があります。
  - 1 字目は英字である必要があります。
  - ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。
- **`--target-db-cluster-parameter-group-description`** — 必須。クラスターパラメータグループのコピーについてユーザーが入力する説明。

Example

次のコードは、sample-parameter-group のコピーを作成し、コピーに sample-parameter-group-copy という名前を付けます。

Linux、macOS、Unix の場合:

```
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"
```

Windows の場合:

```
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"
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "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"
    }
}
```

Amazon DocumentDB クラスターパラメータグループをリセットする

Amazon DocumentDB クラスターパラメータグループのパラメータ値の一部またはすべてをデフォルト値にリセットするには、 AWS マネジメントコンソール または AWS Command Line Interface (AWS CLI) を使用してクラスターパラメータグループをリセットします。

Using the AWS マネジメントコンソール

以下のステップに従って、クラスターパラメータグループのパラメータ値の一部または全部をデフォルト値にリセットします。

クラスターパラメータグループのパラメータ値をリセットするには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. コンソールの左側のナビゲーションペインで、[パラメータグループ] を選択します。
3. [クラスターパラメータグループ] ペインで、リセットするクラスターパラメータグループの名前を選択します。
4. [アクション] を選択し、[リセット] を選択して、そのパラメータグループをリセットします。
5. 結果の [クラスターパラメータグループのリセットの確認] ページで、そのパラメータグループのすべてのクラスターパラメータをデフォルトにリセットすることを確認します。次に、[リセット] を選択してパラメータグループをリセットします。[キャンセル] を選択して変更を破棄することもできます。

Using the AWS CLI

クラスターパラメータグループのパラメータ値の一部またはすべてをデフォルト値にリセットするには、次のパラメータを指定して `reset-db-cluster-parameter-group` オペレーションを使用します。

- **`--db-cluster-parameter-group-name`** — 必須。リセットするクラスターパラメータグループの名前。
- **`--parameters`** - オプション。デフォルト値にリセットするクラスターパラメータグループの `ParameterName` と `ApplyMethod` のリスト。次のインスタンスの再起動または `reboot-db-instance` リクエストで有効にするには、静的パラメータを `pending-reboot` に設定する必要があります。更新された静的パラメータを適用するクラスター内のすべてのインスタンスに対して、`reboot-db-instance` を呼び出す必要があります。

  このパラメータおよび `--reset-all-parameters` は相互に排他的です。いずれかを使用できますが、両方を使用することはできません。
- **`--reset-all-parameters`** または **`--no-reset-all-parameters`** - オプション。すべてのパラメータ (`--reset-all-parameters`) または一部のパラメータのみ (`--no-reset-all-parameters`) をデフォルト値にリセットするかどうかを指定します。`--reset-all-parameters` パラメータおよび `--parameters` は相互に排他的です。いずれかを使用できますが、両方を使用することはできません。

グループ全体をリセットすると、動的パラメータはただちに更新されます。次のインスタンスの再起動または reboot-db-instance リクエストで有効にするには、静的パラメータを pending-reboot に設定します。更新された静的パラメータを適用するクラスター内のすべてのインスタンスに対して、reboot-db-instance を呼び出す必要があります。

Example

例 1: すべてのパラメータをデフォルト値にリセットする

次のコードでは、クラスターパラメータグループ sample-parameter-group のすべてのパラメータをデフォルト値にリセットします。

Linux、macOS、Unix の場合:

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

Windows の場合:

```
aws docdb reset-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group ^
       --reset-all-parameters
```

例 2: 指定されたパラメータをデフォルト値にリセットする

次のコードでは、クラスターパラメータグループ sample-parameter-group の tls パラメータをデフォルト値にリセットします。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
        "DBClusterParameterGroupName": "sample-parameter-group"
   }
```

クラスターインスタンスの再起動

静的パラメータの値が変更される前に、クラスターインスタンスを再起動する必要があります。更新された静的パラメータを適用するクラスター内の各インスタンスを再起動します。

Linux、macOS、Unix の場合:

```
aws docdb reboot-db-instance \
        --db-instance-identifier sample-cluster-instance
```

Windows の場合:

```
aws docdb reboot-db-instance ^
        --db-instance-identifier sample-cluster-instance
```

Amazon DocumentDB クラスターパラメータグループを削除する

AWS マネジメントコンソール または AWS Command Line Interface () を使用して、カスタム Amazon DocumentDB クラスターパラメータグループを削除できますAWS CLI。`default.docdb4.0` クラスターパラメータグループは削除できません。

Using the AWS マネジメントコンソール

クラスターパラメータグループを削除するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. [パラメータグループ] ペインで、削除するクラスターパラメータグループの左にあるラジオボタンを選択します。
4. [アクション] を選択し、[削除] を選択します。
5. [削除] 確認ペインで、[削除] を選択してクラスターパラメータグループを削除します。クラスターパラメータグループを維持するには、[キャンセル] を選択します。

Using the AWS CLI

クラスターパラメータグループを削除するには、以下のパラメータを指定して `delete-db-cluster-parameter-group` オペレーションを使用します。

- **`--db-cluster-parameter-group-name`** — 必須。削除するクラスターパラメータグループの名前。これは既存のクラスターパラメータグループである必要があります。*`default.docdb4.0`* クラスターパラメータグループは削除できません。

Example- クラスターパラメータグループの削除

以下の例では、クラスターパラメータグループを削除するための 3 つのステップについて説明します。

1. 削除するクラスターパラメータグループの名前を見つける。
2. 指定されたクラスターパラメータグループを削除する。
3. クラスターパラメータグループが削除されたことを確認する。

1. 削除するクラスターパラメータグループの名前を見つけます。

次のコードは、すべてのクラスターパラメータグループの名前を表示します。

Linux、macOS、Unix の場合:

```
aws docdb describe-db-cluster-parameter-groups \
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

Windows の場合:

```
aws docdb describe-db-cluster-parameter-groups ^
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

前のオペレーションの出力は、次のようなクラスターパラメータグループの名前のリストになります (JSON 形式)。

```
[
       [
           "default.docdb4.0"
       ],
       [
           "sample-parameter-group"
       ],
       [
           "sample-parameter-group-copy"
       ]
   ]
```

2. 特定のクラスターパラメータグループを削除します。

以下のコードは、クラスターパラメータグループ `sample-parameter-group-copy` を削除します。

Linux、macOS、Unix の場合:

```
aws docdb delete-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group-copy
```

Windows の場合:

```
aws docdb delete-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group-copy
```

このオペレーションからの出力はありません。

3. 指定されたクラスターパラメータグループが削除されたことを確認します。

次のコードは、残りのすべてのクラスターパラメータグループの名前を表示します。

Linux、macOS、Unix の場合:

```
aws docdb describe-db-cluster-parameter-groups \
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

Windows の場合:

```
aws docdb describe-db-cluster-parameter-groups ^
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

前のオペレーションの出力は、次のようなクラスターパラメータグループのリストになります (JSON 形式)。先ほど削除したクラスターパラメータグループはリストに含まれません。

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
[
       [
           "default.docdb4.0"
       ],
       [
           "sample-parameter-group"
       ]
   ]
```

Amazon DocumentDB クラスターパラメータリファレンス

動的なパラメータを変更し、クラスターパラメータグループを保存すると、[すぐに適用] の設定にかかわらず、変更は直ちに適用されます。静的パラメータを変更し、クラスターパラメータグループを保存すると、パラメーターの変更はインスタンスを手動で再起動した後に有効になります。Amazon DocumentDB コンソールを使用するか、`reboot-db-instance` を明示的に呼び出して、インスタンスを再起動できます。

次の表は、Amazon DocumentDB クラスターのすべてのインスタンスに適用されるパラメータを示しています。

Amazon DocumentDB クラスターレベルのパラメータ

パラメータ	デフォルト値	有効値	変更可能	適用タイプ	データタイプ	説明
audit_logs	無効	有効、無効、ddl、dml_read、dml_write、すべて、なし	Yes	動的	String	Amazon CloudWatch 監査ログを有効にするかどうかを定義します。 • **enabled** - CloudTrail の監査ログが有効になります。 • **disabled** - CloudWatch の監査ログが無効になります。 • **ddl** - DDL イベントの監査が有効になります。 • **dml_read** - DML 読み取りイベントの

パラメータ	デフォルト値	有効値	変更可能	適用タイプ	データタイプ	説明
						監査が有効になります。 • **dml_write** - DML 書き込みイベントの監査が有効になります。 • **all** - すべてのデータベースイベントの監査が有効になります。 • **none** - 監査が無効になります。
change_stream_log_retention_duration	10800	3600-604800	はい	動的	整数	変更ストリームログが保持され、消費できる期間 (秒単位) を定義します。

パラメータ	デフォルト値	有効値	変更可能	適用タイプ	データタイプ	説明
default_collection_compression	無効	有効、無効 (Amazon DocumentDB 5.0) / zstd、lz4、なし (Amazon DocumentDB8.0)	はい	動的	String	クラスター内の新しいコレクションのデフォルトの圧縮設定を定義します。 • **enabled** — Amazon DocumentDB 5.0 では Lz4 圧縮がデフォルトで有効になっています。Amazon DocumentDB 8.0 では zstd 圧縮がデフォルトで有効になっています。 • **disabled, none** — 圧縮はデフォルトで無効になって

パラメータ	デフォルト値	有効値	変更可能	適用タイプ	データタイプ	説明
						います (Amazon DocumentDB 5.0 では無効、Amazon DocumentDB 8.0 では無効ではありません）。

パラメータ	デフォルト値	有効値	変更可能	適用タイプ	データタイプ	説明
profiler	無効	有効、無効	はい	動的	String	低速のオペレーションについてプロファイリングを有効にします。 • **enabled** - オペレーションが、クラスター定義しきい値 (100 ミリ秒など) よりも長い時間がかかる場合は、Amazon CloudWatch Logs に記録されます。 • **disabled** - 低速のオペレーションは CloudWatch Logs に記録

パラメータ	デフォルト値	有効値	変更可能	適用タイプ	データタイプ	説明
						されません。
profiler_sampling_rate	1.0	0.0-1.0	はい	動的	浮動小数点数	ログに記録されるオペレーションのサンプリングレートを定義します。
profiler_threshold_ms	100	50-2147483646	はい	動的	整数	profiler のしきい値を定義します。 • profiler_threshold_ms より大きいすべてのオペレーションは CloudWatch Logs に記録されます。

パラメータ	デフォルト値	有効値	変更可能	適用タイプ	データタイプ	説明
planner_version	3.0	1.0、2.0、3.0	はい	動的	浮動小数点数	クエリに使用するクエリプランナーのバージョンを定義します。 • **1.0** — Amazon DocumentDB 5.0 でサポートされています。 • **2.0** — 演算子の検索と更新が改善されました。Amazon DocumentDB 5.0 でサポートされています。 • **3.0** — 検索、更新、集計演算子の改善。Amazon

パラメータ	デフォルト値	有効値	変更可能	適用タイプ	データタイプ	説明
						DocumentDB 8.0 のデフォルトのクエリプランナー。

パラメータ	デフォルト値	有効値	変更可能	適用タイプ	データタイプ	説明
`tls`	有効	有効、無効、fips-140-3、tls1.2+、tls1.3+	はい	静的	String	Transport Layer Security (TLS) 接続が必要かどうかを定義します。 • **enabled** - 接続するには、TLS 接続が必要です。Amazon DocumentDB 5.0 以前では、TLS 1.0 ~ 1.3 を有効にしますが、TLS 1.2 以降を使用します。マイナーバージョン 5.0.1 以降、Amazon DocumentDB 8.0

パラメータ	デフォルト値	有効値	変更可能	適用タイプ	データタイプ	説明
						以降では、TLS 1.2 以降のみが Amazon DocumentDB 5.0 でサポートされています。 • **disabled** - 接続に TLS 接続を使用することはできません。 • **fips-140-3** - 接続には、連邦情報処理標準 (FIPS) 属性をもつ TLS 接続が必要です。クラスターは、FIPS 出版物 140-3 に従って、

パラメータ	デフォルト値	有効値	変更可能	適用タイプ	データタイプ	説明
						セキュアな接続のみを受け入れます。これは、ca-central-1、us-west-2、us-east-1、us-east-2、us-gov-east-1、us-gov-west-1 の各リージョンの Amazon DocumentDB 5.0 (エンジンバージョン 3.0.3727) クラスターからのみサポートされています。

パラメータ	デフォルト値	有効値	変更可能	適用タイプ	データタイプ	説明
						• **tls1.2+** — 接続するには、TLS バージョン 1.2 以降を使用する TLS 接続が必要です。これは、Amazon DocumentDB 4.0 (エンジンバージョン 2.0.10980) および Amazon DocumentDB 5.0 (エンジンバージョン 3.0.11051) 以降でのみサポートされています。 • **tls1.3+** — 接続するに

パラメータ	デフォルト値	有効値	変更可能	適用タイプ	データタイプ	説明
						は、TLS バージョン 1.3 以降を使用する TLS 接続が必要です。これは、Amazon DocumentDB 4.0 (エンジンバージョン 2.0.10980) および Amazon DocumentDB 5.0 (エンジンバージョン 3.0.11051) 以降でのみサポートされています。

パラメータ	デフォルト値	有効値	変更可能	適用タイプ	データタイプ	説明
ttl_monitor	有効	有効、無効	はい	動的	String	クラスターに対して有効期限 (TTL) モニタリングを有効にするかどうかを定義します。 • **enabled** - TTL モニタリングが有効になります。 • **disabled** - TTL モニタリングが無効になります。

Amazon DocumentDB クラスターパラメータを変更する

Amazon DocumentDB では、クラスターパラメータグループ は、クラスターで作成するすべてのインスタンスに適用される パラメータ で構成されています。カスタムクラスターパラメータグループでは、パラメータ値をいつでも変更したり、作成したパラメータグループのすべてのパラメータ値をデフォルトにリセットできます。このセクションでは、Amazon DocumentDB クラスターパラメータグループを構成するパラメータとその値を表示する方法、およびこれらの値を変更または更新する方法について説明します。

パラメータは動的でも静的でもかまいません。動的なパラメータを変更し、クラスターパラメータグループを保存すると、`Apply Immediately` の設定にかかわらず、変更は直ちに適用されます。静

的パラメータを変更し、クラスターパラメータグループを保存すると、変更はインスタンスを手動で再起動した後にのみ有効になります。

Amazon DocumentDB クラスターパラメータグループのパラメータを表示する

Amazon DocumentDB クラスターのパラメータとその値は、 AWS マネジメントコンソール または を使用して確認できます AWS CLI。

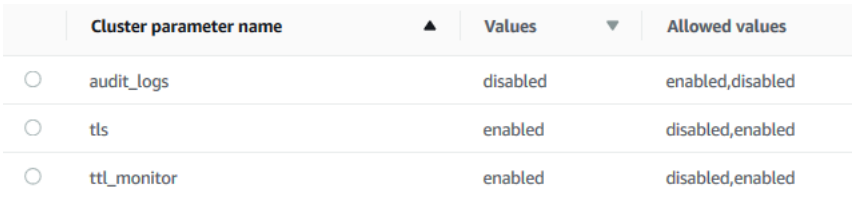
Using the AWS マネジメントコンソール

クラスターパラメータグループの詳細情報を表示するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. [パラメータグループ] ペインで、詳細情報を表示するクラスターパラメータグループの名前を選択します。
4. 結果ページには、各パラメータのパラメータ名、現在の値、許可された値、パラメータが変更可能かどうか、適用タイプ、データ型および説明の値が表示されます。



	Cluster parameter name	Values	Allowed values
○	audit_logs	disabled	enabled,disabled
○	tls	enabled	disabled,enabled
○	ttl_monitor	enabled	disabled,enabled

Using the AWS CLI

クラスターのパラメータグループのパラメータとその値を表示するには、次のパラメータを指定して `describe-db-cluster-parameters` オペレーションを使用します。

- **--db-cluster-parameter-group-name** — 必須。詳細なパラメータリストを表示するクラスターパラメータグループの名前。
- **--source** - オプション。指定した場合、特定のソースのパラメータだけが返されます。パラメータのソースは engine-default、system、または user のいずれかとすることができます。

Example

次のコードは、custom3-6-param-grp パラメータグループのパラメータと値を一覧表示します。パラメータグループの詳細については、--query の行を省略してください。すべてのパラメータグループについては、--db-cluster-parameter-group-name の行を省略してください。

Linux、macOS、Unix の場合:

```
aws docdb describe-db-cluster-parameters \
   --db-cluster-parameter-group-name custom3-6-param-grp \
   --query 'Parameters[*].[ParameterName,ParameterValue]'
```

Windows の場合:

```
aws docdb describe-db-cluster-parameters ^
   --db-cluster-parameter-group-name custom3-6-param-grp ^
   --query 'Parameters[*].[ParameterName,ParameterValue]'
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
[
    [
        "audit_logs",
        "disabled"
    ],
    [
        "tls",
        "enabled"
    ],
    [
        "ttl_monitor",
        "enabled"
```

```
    ]
]
```

Amazon DocumentDB クラスターパラメータグループのパラメータを変更する

パラメータグループのパラメータは、 AWS マネジメントコンソール または を使用して変更できます AWS CLI。

Using the AWS マネジメントコンソール

クラスターパラメータグループのパラメータを更新するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. [パラメータグループ] ペインで、パラメータを更新するクラスターパラメータグループを選択します。
4. 結果のページには、このクラスターパラメータグループのパラメータと対応する詳細が表示されます。更新するパラメータを選択します。
5. ページの右上にある [編集] を選択して、パラメータの値を変更します。クラスターパラメータのタイプの詳細については、「Amazon DocumentDB クラスターパラメータリファレンス」を参照してください。
6. 変更を行い、[クラスターのパラメータを変更] を選択して変更を保存します。変更を破棄するには、[キャンセル] を選択します。

Using the AWS CLI

クラスターパラメータグループのパラメータを変更するには、以下のパラメータを指定して `modify-db-cluster-parameter-group` オペレーションを使用します。

- **--db-cluster-parameter-group-name** — 必須。変更するクラスターパラメータグループの名前。
- **--parameters** — 必須。変更するパラメータ。各パラメータの入力には以下を含める必要があります。
  - **ParameterName** - 変更するパラメータの名前です。
  - **ParameterValue** - このパラメータの新しい値です。
  - **ApplyMethod** - このパラメータの変更を適用する方法です。有効な値は、immediate および pending-reboot です。

> **Note**
>
> ApplyType が static であるパラメータでは、ApplyMethod が pending-reboot である必要があります。

クラスターパラメータグループのパラメータの値を変更するには (AWS CLI)

次の例では、tls パラメータを変更します。

1. **sample-parameter-group** のパラメータとその値を一覧表示します。

   Linux、macOS、Unix の場合:

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

   Windows の場合:

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

   このオペレーションによる出力は、次のようになります (JSON 形式)。

   ```
   {
       "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. **tls** パラメータを変更し、その値が **disabled** になるようにします。ApplyType が static であるため、ApplyMethod を変更することはできません。

   Linux、macOS、Unix の場合:

```
aws docdb modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name sample-parameter-group \
    --parameters
 "ParameterName"=tls,ParameterValue=disabled,ApplyMethod=pending-reboot"
```

   Windows の場合:

```
aws docdb modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name sample-parameter-group ^
    --parameters "ParameterName=tls,ParameterValue=disabled,ApplyMethod=pending-
reboot"
```

   このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
```

```
    "DBClusterParameterGroupName": "sample-parameter-group"
}
```

3. 少なくとも 5 分お待ち下さい。
4. **sample-parameter-group** のパラメータ値を一覧表示します。

   Linux、macOS、Unix の場合:

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

   Windows の場合:

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

   このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "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"
        }
```

```
    ]
}
```

Amazon DocumentDB エンドポイントについて

Amazon DocumentDB (MongoDB 互換) エンドポイントを使用して、クラスターまたはインスタンスに接続できます。Amazon DocumentDB には 3 つの異なる種類のエンドポイントがあり、それぞれに独自の用途があります。

トピック

- クラスターのエンドポイントの検索
- インスタンスのエンドポイントの検索
- エンドポイントへの接続

クラスターエンドポイント

クラスターエンドポイントとは、クラスターの現在のプライマリインスタンスに接続する Amazon DocumentDB のクラスターのエンドポイントです。Amazon DocumentDB クラスターごとに 1 つのクラスターエンドポイントと 1 つのプライマリインスタンスがあります。フェイルオーバーが発生すると、クラスターエンドポイントは新しいプライマリインスタンスに再マッピングされます。

リーダーエンドポイント

読み取りエンドポイントとは、そのクラスターで利用できるレプリカの 1 つに接続する、Amazon DocumentDB クラスターのエンドポイントです。各 Amazon DocumentDB クラスターに読み取りエンドポイントがあります。複数のレプリカがある場合、読み取りエンドポイントは各接続リクエストを Amazon DocumentDB レプリカのいずれかにルーティングします。

インスタンスエンドポイント

インスタンスエンドポイントは、特定のインスタンスに接続するエンドポイントです。クラスターの各インスタンスには、プライマリインスタンスかレプリカインスタンスタイプにかかわらず、独自のインスタンスエンドポイントがあります。アプリケーションでインスタンスエンドポイントを使用しないことが推奨されます。これは、フェイルオーバーの場合にインスタンスエンドポイントがロールを変更することがあり、これによってアプリケーション内のコード変更が必要となるためです。

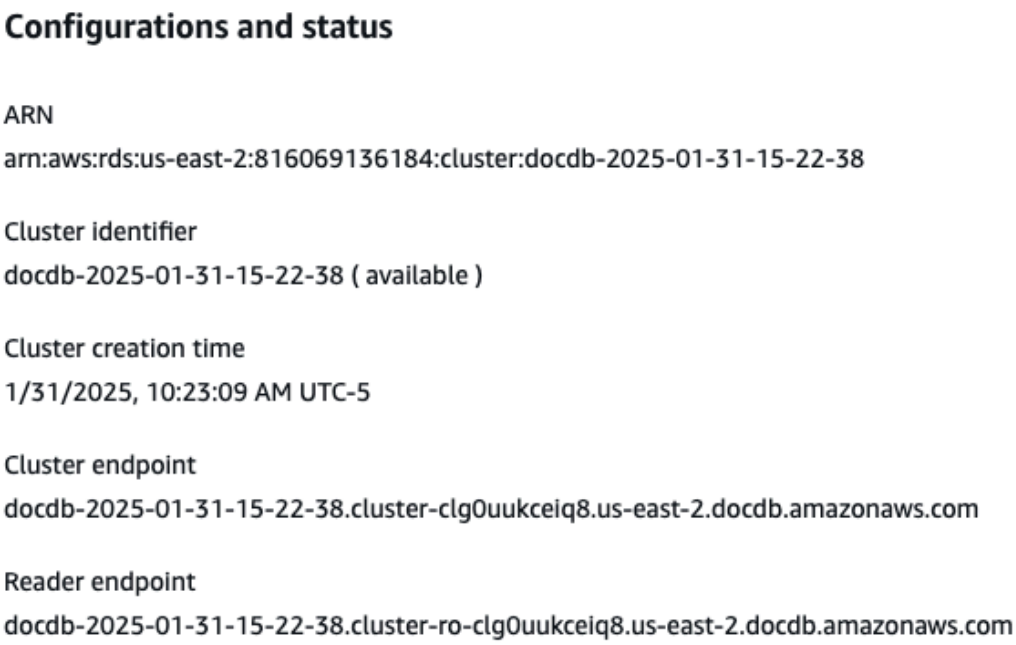
クラスターのエンドポイントの検索

Amazon DocumentDB コンソールまたは AWS CLIを使用して、クラスターのクラスターエンドポイントと読み込みエンドポイントを見つけることができます。

Using the AWS マネジメントコンソール

コンソールを使用してクラスターのエンドポイントを見つけるには:

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択してください。
3. クラスターの一覧で、関心のあるクラスターの名前を選択します。
4. クラスターの詳細ページで、[Configuration] (構成) タブを選択します。[設定とステータス] セクションには、[クラスターエンドポイント] と [リーダーエンドポイント] があります。



5. このクラスターに接続するには、[接続とセキュリティ] タブを選択します。mongo シェルの接続文字列と、アプリケーションコードで使用してクラスターに接続できる接続文字列を見つけます。



Using the AWS CLI

を使用してクラスターのクラスターエンドポイントとリーダーエンドポイントを検索するには AWS CLI、これらのパラメータを指定して `describe-db-clusters` コマンドを実行します。

パラメータ

- **`--db-cluster-identifier`** - オプションです。エンドポイントを返すクラスターを指定します。省略した場合、最大 100 個のクラスターのエンドポイントを返します。
- **`--query`** - オプションです。表示するフィールドを指定します。エンドポイントを見つけるために表示する必要があるデータの量を減らすと役立ちます。省略した場合、すべてのクラスターについての情報が返されます。
- **`--region`** - オプションです。`--region` パラメータを使用して、コマンドを適用するリージョンを指定します。省略すると、デフォルトのリージョンが使用されます。

Example

次の例では、`DBClusterIdentifier`、エンドポイント (クラスターエンドポイント)、および `sample-cluster` の `ReaderEndpoint` を返します。

Linux、macOS、Unix の場合:

```
aws docdb describe-db-clusters \
   --region us-east-1 \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[DBClusterIdentifier,Port,Endpoint,ReaderEndpoint]'
```

Windows の場合:

```
aws docdb describe-db-clusters ^
```

```
    --region us-east-1 ^
    --db-cluster-identifier sample-cluster ^
    --query 'DBClusters[*].[DBClusterIdentifier,Port,Endpoint,ReaderEndpoint]'
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
[
  [
     "sample-cluster",
     27017,
     "sample-cluster.cluster-corlsfccjozr.us-east-1.docdb.amazonaws.com",
     "sample-cluster.cluster-ro-corlsfccjozr.us-east-1.docdb.amazonaws.com"
  ]
]
```

これでクラスターエンドポイントが作成したので、mongo または mongodb を使用してクラスターに接続できます。詳細については、「エンドポイントへの接続」を参照してください。

インスタンスのエンドポイントの検索

Amazon DocumentDB コンソールまたは AWS CLIを使用して、インスタンスのためのエンドポイントを見つけることができます。

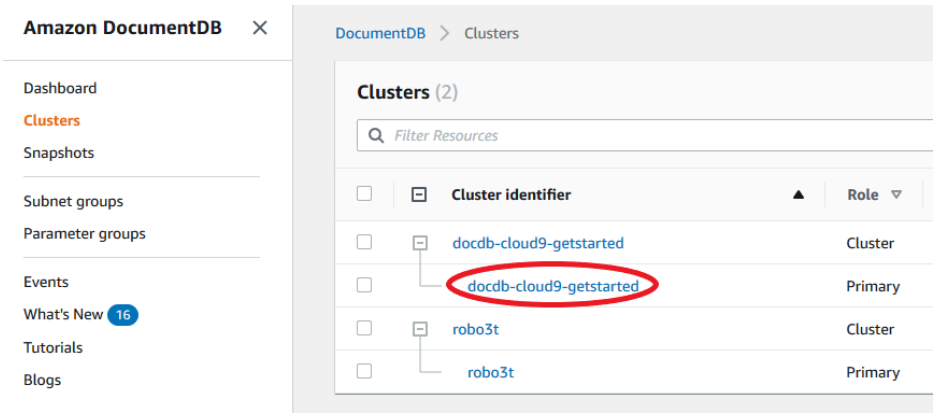
Using the AWS マネジメントコンソール

コンソールを使用してインスタンスのエンドポイントを見つけるには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

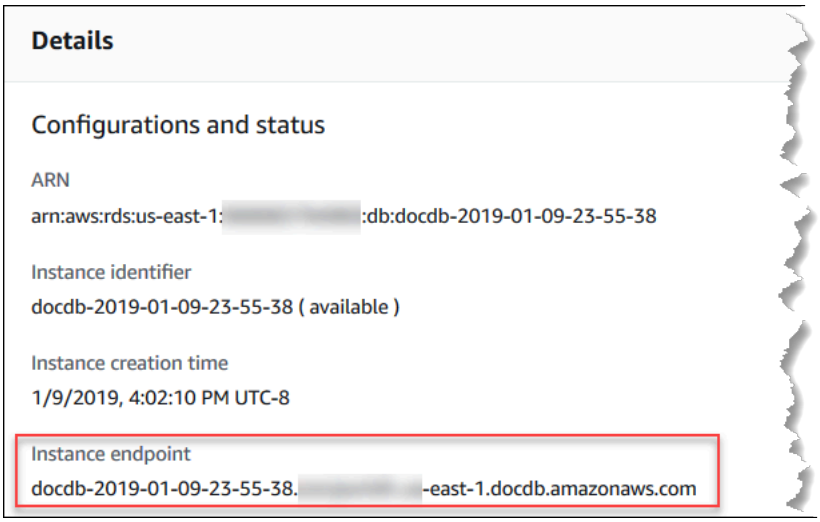
> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. クラスターナビゲーションボックスに、[Cluster Identifier]の列が表示されます。インスタンスは、次のスクリーンショットのようにクラスターの下に一覧表示されます。



4. 興味があるインスタンスの左側にあるチェックボックスをチェックします。

5. [詳細] セクションまで下にスクロールし、インスタンスのエンドポイントを見つけます。



6. このインスタンスに接続するには、[接続] セクションまで上にスクロールします。mongo シェルの接続文字列と、アプリケーションコードで使用してインスタンスに接続できる接続文字列を見つけます。



Using the AWS CLI

を使用してインスタンスエンドポイントを検索するには AWS CLI、これらの引数を指定して次のコマンドを実行します。

引数

- **--db-instance-identifier** - オプションです。エンドポイントを返すインスタンスを指定します。省略した場合、最大 100 個のインスタンスのエンドポイントを返します。
- **--query** - オプションです。表示するフィールドを指定します。エンドポイントを見つけるために表示する必要があるデータの量を減らすと役立ちます。省略した場合、インスタンスのすべての情報が返されます。この Endpoint フィールドには 3 つのメンバーがあるため、次の例に示すようにクエリでリストすると、3 つすべてのメンバーが返されます。一部の Endpoint メンバーのみに関心がある場合は、2 番目の例のように、クエリで Endpoint を関心のあるメンバーに置き換えます。
- **--region** - オプションです。--region パラメータを使用して、コマンドを適用するリージョンを指定します。省略すると、デフォルトのリージョンが使用されます。

Example

Linux、macOS、Unix の場合:

```
aws docdb describe-db-instances \
    --region us-east-1 \
    --db-instance-identifier sample-cluster-instance \
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint]'
```

Windows の場合:

```
aws docdb describe-db-instances ^
    --region us-east-1 ^
    --db-instance-identifier sample-cluster-instance ^
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint]'
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
[
    [
```

```
        "sample-cluster-instance",
        {
            "Port": 27017,
            "Address": "sample-cluster-instance.corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
            "HostedZoneId": "Z2R2ITUGPM61AM"
        }
    ]
]
```

出力を減らしてエンドポイントの HostedZoneId を排除すると、Endpoint.Port と Endpoint.Address を使用してクエリを変更することができます。

Linux、macOS、Unix の場合:

```
aws docdb describe-db-instances \
    --region us-east-1 \
    --db-instance-identifier sample-cluster-instance \
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint.Port,Endpoint.Address]'
```

Windows の場合:

```
aws docdb describe-db-instances ^
    --region us-east-1 ^
    --db-instance-identifier sample-cluster-instance ^
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint.Port,Endpoint.Address]'
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
[
    [
        "sample-cluster-instance",
        27017,
        "sample-cluster-instance.corcjozrlsfc.us-east-1.docdb.amazonaws.com"
    ]
]
```

これでインスタンスエンドポイントを作成したので、mongo または mongodb を使用してインスタンスに接続できます。詳細については、「エンドポイントへの接続」を参照してください。

エンドポイントへの接続

エンドポイント (クラスターまたはインスタンス) が作成されたら、mongo シェルまたは接続文字列を使用して接続できます。

mongo シェルを使用した接続

以下の構造を使用して、mongo シェルを使ってクラスターまたはインスタンスに接続する必要がある文字列を作成します。

```
mongo \
    --ssl \
    --host Endpoint:Port \
    --sslCAFile global-bundle.pem \
    --username UserName \
    --password Password
```

mongo シェルの例

クラスターへの接続:

```
mongo \
    --ssl \
    --host sample-cluster.corcjozrlsfc.us-east-1.docdb.amazonaws.com:27017 \
    --sslCAFile global-bundle.pem \
    --username UserName \
    --password Password
```

インスタンスへの接続:

```
mongo \
    --ssl \
    --host sample-cluster-instance.corcjozrlsfc.us-east-1.docdb.amazonaws.com:27017 \
    --sslCAFile global-bundle.pem \
    --username UserName \
    --password Password
```

接続文字列を使用した接続

以下の構造を使用して、クラスターまたはインスタンスに接続する必要がある接続文字列を作成します。

```
mongodb://UserName:Password@endpoint:port?replicaSet=rs0&ssl_ca_certs=global-
bundle.pem
```

接続文字列の例

クラスターへの接続:

```
mongodb://UserName:Password@sample-cluster.cluster-corlsfccjozr.us-
east-1.docdb.amazonaws.com:27017?replicaSet=rs0&ssl_ca_certs=global-bundle.pem
```

インスタンスへの接続:

```
mongodb://UserName:Password@sample-cluster-instance.cluster-corlsfccjozr.us-
east-1.docdb.amazonaws.com:27017?replicaSet=rs0&ssl_ca_certs=global-bundle.pem
```

Amazon DocumentDB Amazon リソースネーム (ARN) の理解

で作成したリソース AWS は、それぞれ Amazon リソースネーム (ARN) で一意に識別されます。特定の Amazon DocumentDB (MongoDB 互換) オペレーションでは、ARN を指定して、Amazon DocumentDB リソースを一意に識別する必要があります。例えば、リソースにタグを追加するときは、リソースの ARN を指定する必要があります。

トピック

- Amazon DocumentDB リソースの ARN の設定
- Amazon DocumentDB リソースの ARN を検索する

Amazon DocumentDB リソースの ARN の設定

次の構文を使用して Amazon DocumentDB リソースの ARN を構築できます。Amazon DocumentDB は、Amazon Relational Database Service (Amazon RDS) ARN の形式を共有します。Amazon DocumentDB ARN には `rds` を含み、`docdb` は含みません。

```
arn:aws:rds:region:account_number:resource_type:resource_id
```

リージョン名	リージョン	アベイラビリティーゾーン (コンピューティング)
米国東部 (オハイオ)	us-east-2	3
米国東部 (バージニア北部)	us-east-1	6
米国西部 (オレゴン)	us-west-2	4
アフリカ (ケープタウン)	af-south-1	3
南米 (サンパウロ)	sa-east-1	3
アジアパシフィック (香港)	ap-east-1	3
アジアパシフィック (ハイデラバード)	ap-south-2	3
アジアパシフィック (マレーシア)	ap-southeast-5	3
アジアパシフィック (ムンバイ)	ap-south-1	3
アジアパシフィック (大阪)	ap-northeast-3	3
アジアパシフィック (ソウル)	ap-northeast-2	4
アジアパシフィック (シンガポール)	ap-southeast-1	3
アジアパシフィック (シドニー)	ap-southeast-2	3

リージョン名	リージョン	アベイラビリティーゾーン (コンピューティング)
アジアパシフィック (ジャカルタ)	ap-southeast-3	3
アジアパシフィック (メルボルン)	ap-southeast-4	3
アジアパシフィック (タイ)	ap-southeast-7	3
アジアパシフィック (東京)	ap-northeast-1	3
カナダ (中部)	ca-central-1	3
カナダ西部 (カルガリー)	ca-west-1	3
中国 (北京) リージョン	cn-north-1	3
中国 (寧夏)	cn-northwest-1	3
欧州 (フランクフルト)	eu-central-1	3
欧州 (チューリッヒ)	eu-central-2	3
欧州 (アイルランド)	eu-west-1	3
欧州 (ロンドン)	eu-west-2	3
欧州 (ミラノ)	eu-south-1	3
欧州 (パリ)	eu-west-3	3
欧州 (スペイン)	eu-south-2	3
欧州 (ストックホルム)	eu-north-1	3

リージョン名	リージョン	アベイラビリティーゾーン (コンピューティング)
メキシコ (中部)	mx-central-1	3
中東 (アラブ首長国連邦)	me-central-1	3
イスラエル (テルアビブ)	il-central-1	3
AWS GovCloud (米国西部)	us-gov-west-1	3
AWS GovCloud (米国東部)	us-gov-east-1	3

> **Note**
>
> Amazon DocumentDB アーキテクチャは、ストレージとコンピューティングを分離します。ストレージレイヤーの場合、Amazon DocumentDB はデータの 6 つのコピーを 3 つの AWS アベイラビリティーゾーン (AZs) にレプリケートします。上記の表に示す AZs は、特定のリージョンでコンピューティングインスタンスをプロビジョニングするために使用できる AZs の数です。例えば、ap-northeast-1 で Amazon DocumentDB クラスターを起動している場合、ストレージは 3 つの AZ にわたって 6 つの方法でレプリケートされますが、コンピューティングインスタンスは 2 つの AZ でしか使用できません。

次の表に、特定の Amazon DocumentDB リソースの ARN の構築時に使用する形式を示します。Amazon DocumentDB は Amazon RDS ARN の形式を共有します。Amazon DocumentDB ARN には `rds` を含み、`docdb` は含みません。

リソースタイプ	ARN 形式/例
インスタンス (db)	`arn:aws:rds:` *`region`*`:`*`account_number`*`:db:`*`resource_id`*

リソースタイプ	ARN 形式/例
	`arn:aws:rds:us-east-1:` *1234567890* `:db:`*sample-db-instance*
クラスター (cluster)	`arn:aws:rds:` *region*`:`*account_number* `:cluster:` *resource_id* `arn:aws:rds:us-east-1:` *1234567890* `:cluster:` *sample-db-cluster*
クラスターパラメータグループ (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*
セキュリティグループ (secgrp)	`arn:aws:rds:` *region*`:`*account_number* `:secgrp:`*resource_id* `arn:aws:rds:us-east-1:` *1234567890* `:secgrp:`*sample-public-secgrp*
クラスタースナップショット (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*
サブネットグループ (subgrp)	`arn:aws:rds:` *region*`:`*account_number* `:subgrp:`*resource_id* `arn:aws:rds:us-east-1:` *1234567890* `:subgrp:`*sample-subnet-10*

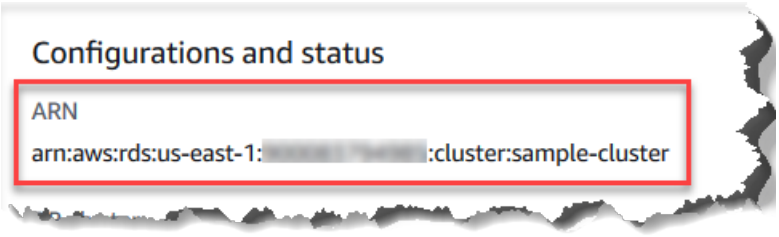
Amazon DocumentDB リソースの ARN を検索する

Amazon DocumentDB リソースの ARN は、 AWS マネジメントコンソール または を使用して確認できます AWS CLI。

Using the AWS マネジメントコンソール

コンソールを使用して ARN を検索するには、ARN を取得するリソースに移動し、そのリソースの詳細を表示します。

例えば、クラスターの詳細ページの [設定] タブを選択して、クラスターの ARN を取得できます。ARN は、次のスクリーンショットに示すように、[設定とステータス] セクションにあります。



Using the AWS CLI

特定の Amazon DocumentDB リソース AWS CLI の を使用して ARN を取得するには、そのリソースの `describe`オペレーションを使用します。次の表は、各 AWS CLI オペレーションと、ARN を取得するオペレーションで使用される ARN プロパティを示しています。

AWS CLI コマンド	ARN プロパティ
`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`

AWS CLI コマンド	ARN プロパティ
describe-db-cluster-snapshots	DBClusterSnapshotArn
describe-db-subnet-groups	DBSubnetGroupArn

Example- クラスターの ARN の検索

次の AWS CLI オペレーションでは、クラスター の ARN を検索しますsample-cluster。

Linux、macOS、Unix の場合:

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

Windows の場合:

```
aws docdb describe-db-clusters ^
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterArn'
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
[
    "arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster"
]
```

Example- 複数のパラメータグループの ARN の検索

Linux、macOS、Unix の場合:

```
aws docdb describe-db-cluster-parameter-groups \
   --query 'DBClusterParameterGroups[*].DBClusterParameterGroupArn'
```

Windows の場合:

```
aws docdb describe-db-cluster-parameter-groups ^
```

```
    --query 'DBClusterParameterGroups[*].DBClusterParameterGroupArn'
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
[
    "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"
]
```

Amazon DocumentDB リソースへのタグ付け

Amazon DocumentDB (MongoDB 互換) タグを使用して Amazon DocumentDB リソースにメタデータを追加できます。これらのタグを AWS Identity and Access Management (IAM) ポリシーとともに使用して、Amazon DocumentDB リソースへのアクセスを管理し、リソースに適用できるアクションを制御できます。また、タグを使用して、類似のリソースの費用をグループ化することで、コストを追跡できます。

次の Amazon DocumentDB リソースにタグ付けができます。

- クラスター
- インスタンス
- スナップショット
- クラスタースナップショット
- パラメータグループ
- クラスターパラメータグループ
- セキュリティグループ
- [サブネットグループ]

Amazon DocumentDB リソースタグの概要

Amazon DocumentDB タグは、Amazon DocumentDB リソースを定義してそのリソースに関連付ける名前と値のペアです。その名前はキーと呼ばれます。キーの値の指定は省略可能です。タグを使用して、Amazon DocumentDB リソースに任意の情報を割り当てることができます。例えば、タグキーを使用してカテゴリを定義し、タグ値をそのカテゴリのアイテムにすることができます。具体的には、`project` というタグキーと `Salix` というタグ値を定義して、Amazon

DocumentDB リソースが Salix プロジェクトに割り当てられていることを示すことができます。また、`environment=test` や `environment=production` などのタグキーを使用して Amazon DocumentDB リソースがテスト用であるか本番稼働用であるかを示すこともできます。Amazon DocumentDB リソースに関連付けられたメタデータの追跡が簡単になるように、一貫した一連のタグキーを使用することをお勧めします。

タグを使用して AWS 請求書を整理し、独自のコスト構造を反映することができます。これを行うには、サインアップしてタグキー値を含む AWS アカウント 請求書を取得します。次に、結合したリソースのコストを見るには、同じタグキー値のリソースに従って請求書情報を整理します。例えば、複数のリソースに特定のアプリケーション名のタグを付け、請求情報を整理することで、複数のサービスを利用しているアプリケーションの合計コストを確認することができます。詳細については、AWS Billing and Cost Management ユーザーガイド の コスト配分タグを使用する を参照してください。

各 Amazon DocumentDB リソースにはタグセットがあり、それぞれのリソースに割り当てられているすべてのタグが含まれています。タグセットには最大 10 個のタグを含めることができ、空にすることもできます。Amazon DocumentDB リソースに追加したタグのキーがそのリソースの既存のタグのキーと同じ場合、既存の値は新しい値によって上書きされます。

AWS はタグに意味論的な意味を適用しません。タグは厳密に文字列として解釈されます。リソース作成時に使用する設定によっては、Amazon DocumentDB によってインスタンスまたは他の Amazon DocumentDB リソースにタグが設定されることがあります。例えば、Amazon DocumentDB によってインスタンスが本稼働用またはテスト用であることを示すタグが追加されることがあります。

スナップショットにタグを追加することはできますが、請求書にはこのグループが反映されません。

AWS マネジメントコンソール または を使用して、Amazon DocumentDB リソースのタグ AWS CLI を追加、一覧表示、削除できます。を使用する場合は AWS CLI、使用するリソースの Amazon リソースネーム (ARN) を指定する必要があります。Amazon DocumentDB ARN の詳細については、Amazon DocumentDB Amazon リソースネーム (ARN) の理解 を参照してください。

タグの制約

Amazon DocumentDB タグには以下の制限があります。

- リソースあたりのタグの最大数は 10 です
- キーの最大長 - 128 文字 (Unicode)
- 値の最大長 - 256 文字 (Unicode)

- [キー] と [値] に使用できる文字 - アルファベットの大文字と小文字 (UTF-8 文字セット)、数字、スペース、次の文字です (_ . : / = + - および @) (Java 正規表現: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$")。
- タグのキーと値では、大文字と小文字が区別されます。
- プレフィックス aws: をタグキーや値に使用することはできません。 AWS用に予約されています。

Amazon DocumentDB リソースでのタグの追加と更新

AWS マネジメントコンソール または を使用して、リソースに最大 10 個のタグを追加できます AWS CLI。

Using the AWS マネジメントコンソール

リソースにタグを追加するプロセスは、タグの追加先となるリソースにかかわらず、似ています。この例では、クラスターにタグを追加します。

コンソールを使用してクラスターにタグを追加または更新するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[クラスター] を選択します。
3. タグを追加するクラスターの名前を指定します。
4. [タグ] セクションまで下にスクロールし、[編集] を選択します。
5. このリソースに追加するタグごとに、以下が必要になります。

    a. 新しいタグを追加するには、[キー] ボックスにタグの名前を入力します。タグの値を変更するには、[キー] 列でタグの名前を見つけます。

    b. 新規または更新された値にタグに付けるには、[値] ボックスにタグの値を入力します。

    c. さらにタグを追加する場合は、[追加] を選択します。それ以外の場合は、完了したら、[保存] を選択します。

Using the AWS CLI

リソースにタグを追加するプロセスは、タグの追加先となるリソースにかかわらず、似ています。この例では、クラスターに 3 つのタグを追加します。2 番目のタグ key2 には値がありません。

これらのパラメータadd-tags-to-resourceで AWS CLI オペレーションを使用します。

パラメータ

- **--resource-name** — タグを追加する Amazon DocumentDB リソースの ARN。
- **--tags** Key=*key-name*,Value=*tag-value* 形式でこのリソースに追加するタグ (キーと値のペア) のリスト。

Example

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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 オペレーションでは何も出力されません。オペレーションの結果を表示するには、list-tags-for-resource オペレーションを使用します。

Amazon DocumentDB リソースでのタグのリスト表示

AWS マネジメントコンソール または を使用して AWS CLI 、Amazon DocumentDB リソースのタグのリストを取得できます。

Using the AWS マネジメントコンソール

リソースのタグをリスト表示するプロセスは、タグの追加先となるリソースにかかわらず、似ています。この例では、クラスターのタグをリストします。

コンソールを使用してクラスター上のタグをリスト表示するには

1. Amazon DocumentDB コンソールを、次の場所で開きます。https://console.aws.amazon.com/docdb

2. ナビゲーションペインで、[クラスター] を選択します。
3. タグをリスト表示するクラスターの名前を指定します。
4. このリソースのタグをリスト表示するには、[タグ] セクションまで下にスクロールします。

Using the AWS CLI

リソースのタグをリスト表示するプロセスは、タグをリスト表示するリソースにかかわらず、似ています。この例では、クラスターのタグをリストします。

これらのパラメータ`list-tags-for-resource`で AWS CLI オペレーションを使用します。

パラメータ

- **`--resource-name`** — 必須 タグを追加する Amazon DocumentDB リソースの ARN。

Example

Linux、macOS、Unix の場合:

```
aws docdb list-tags-for-resource \
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster
```

Windows の場合:

```
aws docdb list-tags-for-resource ^
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "TagList": [
        {
            "Key": "key1",
            "Value": "value1"
        },
        {
            "Key": "key2",
            "Value": ""
        },
```

```
        {
            "Key": "key3",
            "Value": "value3"
        }
    ]
}
```

Amazon DocumentDB リソースからのタグの削除

AWS マネジメントコンソール または を使用して AWS CLI 、Amazon DocumentDB リソースからタグを削除できます。

Using the AWS マネジメントコンソール

リソースからタグを削除するプロセスは、タグの追加先となるリソースにかかわらず、似ています。この例では、クラスターからタグを削除します。

コンソールを使用してクラスターからタグを削除するには

1. Amazon DocumentDB コンソールを、次の場所で開きます。https://console.aws.amazon.com/docdb
2. ナビゲーションペインで、[クラスター] を選択します。
3. タグを削除するクラスターの名前を指定します。
4. [タグ] セクションまで下にスクロールし、[編集] を選択します。
5. このリソースからすべてのタグを削除するには、[すべて削除] を選択します。それ以外の場合、このリソースから削除するタグごとに、以下を実行します。

    a. [キー] 列で、タグの名前を見つけます。
    b. タグキーと同じ行で [削除] を選択します。
    c. 完了したら、[保存] を選択します。

Using the AWS CLI

リソースからタグを削除するプロセスは、タグの削除元となるリソースにかかわらず、似ています。この例では、クラスターからタグを削除します。

これらのパラメータ`remove-tags-from-resource`で AWS CLI オペレーションを使用します。

- **--resource-name** — 必須 タグを追加する Amazon DocumentDB リソースの ARN。
- **--tag-keys** — 必須 このリソースから削除するタグキーのリスト。

Example

Linux、macOS、Unix の場合:

```
aws docdb remove-tags-from-resource \
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster \
    --tag-keys key1 key3
```

Windows の場合:

```
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` オペレーションでは何も出力されません。オペレーションの結果を表示するには、`list-tags-for-resource` オペレーションを使用します。

Amazon DocumentDB のメンテナンス

Amazon DocumentDB は定期的に 2 種類のメンテナンスを実行します。

- クラスターメンテナンスはデータベースエンジンを更新します。エンジンの更新には、セキュリティ修正、バグ修正、新機能、その他のエンジンの機能強化が含まれています。
- インスタンスメンテナンスは、インスタンスのオペレーティングシステム (OS) を更新します。

エンジンパッチと OS 更新では、オプション、必須、強制の 3 つのライフサイクルカテゴリが同じ通知で使用され、各カテゴリの動作が適用されます。エンジンリリースには、マイナーバージョンという 4 番目のカテゴリもあり、手動で にアップグレードします。カテゴリは次のとおりです。

- オプション — 重要でない改善が含まれています。自動適用日や AHD 通知はありません。必要に応じて適用されます。(OS の更新については、 にサブスクライブRDS-EVENT-0230して、利用可能になったときに通知を受け取ることができます) 。
- 必須 — セキュリティやその他の重要な修正が含まれています。(Health Dashboard AHD) と E メールを通じて通知を受け取ります。必要なアクションは、 の後にクラスターまたはインスタン

スのメンテナンスウィンドウ中に自動的に適用されますAutoAppliedAfterDate。その日付より前にメンテナンスウィンドウを変更することで、延期できます。

- 強制的 — まれで重要度の高い修正。メンテナンスウィンドウの後に が自動的に適用されますForcedApplyDate。Amazon DocumentDB は、他のオプションが利用できない場合にのみ強制されるアクションを指定します。
- マイナーバージョン (エンジンリリースのみ) — メジャーバージョン (例:) 上の番号付きエンジンリリース5.0.1。ユーザー主導: クラスターのエンジンバージョンを変更してアップグレードします。自動適用しないでください。AHD 通知はありません。マイナーバージョンは、5.0 より前のメジャーバージョンでは公開されません。

エンジンパッチは 1 つのカテゴリ (オプション、必須、または強制) にリリースされ、そこに留まります。OS 更新の進行状況: ほとんどの はオプションとして開始され、適用されていない場合は必須に移行し、最終的に強制されます。正確なタイミングはパッチによって異なり、AHD 通知と によって返される日付フィールドで公開されます describe-pending-maintenance-actions (「」を参照日付の適用)。Amazon DocumentDB リリースノートは、エンジンの変更を発表するときにこれらのカテゴリ名を使用します。

エンジンパッチを適用すると、クラスターは短時間オフラインになります。このトピックの残りの部分では、メンテナンスウィンドウの仕組み、保留中の作業の検索方法、エンジンパッチとマイナーバージョンの適用方法、OS 更新の仕組み、グローバルクラスターの特別な処理について説明します。

トピック

- Amazon DocumentDB のメンテナンスアクション
- エンジンバージョンの番号付け
- Amazon DocumentDB メンテナンスウィンドウの管理する
- Amazon DocumentDB エンジンパッチの通知
- 実施待ちの Amazon DocumentDB メンテナンスアクションを確認する
- Amazon DocumentDB エンジンの更新
- マイナーバージョンのアップグレード
- Amazon DocumentDB オペレーティングシステムの更新
- ユーザーが起動する更新
- グローバルクラスターのパッチ適用

Amazon DocumentDB のメンテナンスアクション

Amazon DocumentDB クラスターには、次のメンテナンスアクションが適用されます。

- `system-update` — Amazon DocumentDB クラスターのエンジンパッチをアップグレードします。詳細については、「Amazon DocumentDB エンジンの更新」を参照してください。
- `os-upgrade` — ローリングアップグレードを使用して、Amazon DocumentDB クラスター内のすべての DB インスタンスのオペレーティングシステムを更新します。詳細については、「Amazon DocumentDB オペレーティングシステムの更新」を参照してください。

Amazon DocumentDB インスタンスには、次のメンテナンスアクションが適用されます。

- `system-update` — Amazon DocumentDB インスタンスのオペレーティングシステムをアップグレードします。代わりにクラスターレベルの`os-upgrade`メンテナンスアクションを使用することをお勧めします。詳細については、「Amazon DocumentDB オペレーティングシステムの更新」を参照してください。

エンジンバージョンの番号付け

Amazon DocumentDB は 2 つの個別のバージョン識別子を使用します。

- エンジンバージョン — 形式の 3 つの部分からなる番号 *`major`*`.`*`major`*`.`*`minor`* (例: `5.0.0`または `5.0.1`)。最初の 2 つのパート (`5.0`) は MongoDB 互換バージョンです。3 番目のパートはマイナーバージョンで、Amazon DocumentDB がバグ修正と中断のない改善を含むマイナーリリースを発行するときに増加します。これは、クラスターを作成またはアップグレードするときに指定するバージョンです。
- エンジンパッチバージョン — クラスターに適用されるパッチレベルを識別するフォーム内の個別の 3 つの部分からなる番号 *`major`*`.0.`*`patch`* (など`3.0.17983`)。中間の数字は常に です`0`。パッチバージョンには、重要なセキュリティと安定性の修正が含まれています。

エンジンパッチバージョンが単独で表示される場合は、次の表に示すように、そのプレフィックスからエンジンバージョンを判断できます。

エンジンパッチバージョンのプレフィックス	Amazon DocumentDB エンジンバージョン
`1.0.`*`x`*	3.6

エンジンパッチバージョンのプレフィックス	Amazon DocumentDB エンジンバージョン
2.0.*x*	4.0
3.0.*x*	5.0
4.0.*x*	8.0

クラスターが実行中のパッチバージョンを確認するには、 を接続して実行しますdb.runCommand({getEngineVersion: 1})。

> **Note**
>
> Amazon DocumentDB 5.0.2 および 8.0.2 以降、 はエンジンバージョンとエンジンパッチバージョンを の形式で単一の文字列としてdb.runCommand({getEngineVersion: 1})返します*engine-version*+*engine-patch-version*。例えば、Amazon DocumentDB 5.0.2 のクラスターは を返し5.0.2+3.0.12345、8.0.2 のクラスターは を返します8.0.2+4.0.12345。以前のバージョンのクラスターは、 などのエンジンパッチバージョンを独自に返します3.0.12345。
> の出力を解析するスクリプトまたはモニタリングがある場合はgetEngineVersion、組み合わせた形式を処理するようにスクリプトを更新します。

リリースされたエンジンパッチのバージョンと各バージョンに含まれるもののリストについては、「」を参照してくださいリリースノート。

Amazon DocumentDB メンテナンスウィンドウの管理する

各クラスターと各インスタンスには、スケジュールされた変更とソフトウェアパッチ適用が実行される期間である、週 30 分の独自のメンテナンスウィンドウがあります。ほとんどのイベントは 30 分以内に完了します。大きなイベントはより長く実行できます。

リソースの作成時にウィンドウを選択しない場合、Amazon DocumentDB は、ランダムに選択された日に、リージョンに定義された 8 時間の 1 日あたりのブロック内でランダムにウィンドウを割り当てます。夜や週末など、アプリケーションへの影響を最小限に抑えるウィンドウを選択します。

データベースエンジンのアップグレードの場合、Amazon DocumentDB は個々のインスタンスのウィンドウではなく、クラスターのウィンドウを使用します。

次の表は、リージョンあたりのデフォルトのタイムブロックを示しています。

リージョン名	リージョン	UTC 時間帯
米国東部 (オハイオ)	us-east-2	03:00 ~ 11:00
米国東部 (バージニア北部)	us-east-1	03:00 ~ 11:00
米国西部 (オレゴン)	us-west-2	06:00 ~ 14:00
アフリカ (ケープタウン)	af-south-1	03:00 ~ 11:00
アジアパシフィック (香港)	ap-east-1	06:00 ~ 14:00
アジアパシフィック (ハイデラバード)	ap-south-2	06:30 ~ 14:30
アジアパシフィック (マレーシア)	ap-southeast-5	13:00 ~ 21:00
アジアパシフィック (ムンバイ)	ap-south-1	06:00 ~ 14:00
アジアパシフィック (大阪)	ap-northeast-3	12:00 ~ 20:00
アジアパシフィック (ソウル)	ap-northeast-2	13:00 ~ 21:00
アジアパシフィック (シンガポール)	ap-southeast-1	14:00 ~ 22:00
アジアパシフィック (シドニー)	ap-southeast-2	12:00 ~ 20:00
アジアパシフィック (ジャカルタ)	ap-southeast-3	08:00 ~ 16:00
アジアパシフィック (メルボルン)	ap-southeast-4	11:00 ~ 19:00
アジアパシフィック (タイ)	ap-southeast-7	15:00 ~ 23:00

リージョン名	リージョン	UTC 時間帯
アジアパシフィック (東京)	ap-northeast-1	13:00 ~ 21:00
カナダ (中部)	ca-central-1	03:00 ~ 11:00
カナダ西部 (カルガリー)	ca-west-1	18:00 ~ 02:00
中国 (北京)	cn-north-1	06:00 ~ 14:00
中国 (寧夏)	cn-northwest-1	06:00 ~ 14:00
欧州 (フランクフルト)	eu-central-1	21:00 ~ 05:00
欧州 (チューリッヒ)	eu-central-2	02:00 ~ 10:00
欧州 (アイルランド)	eu-west-1	22:00 ~ 06:00
欧州 (ロンドン)	eu-west-2	22:00 ~ 06:00
欧州 (ミラノ)	eu-south-1	02:00 ~ 10:00
欧州 (パリ)	eu-west-3	23:59 ~ 07:29
欧州 (スペイン)	eu-south-2	02:00 ~ 10:00
欧州 (ストックホルム)	eu-north-1	04:00 ~ 12:00
メキシコ (中部)	mx-central-1	03:00 ~ 11:00
中東 (UAE)	me-central-1	05:00 ~ 13:00
南米 (サンパウロ)	sa-east-1	00:00 ~ 08:00
イスラエル (テルアビブ)	il-central-1	04:00 ~ 12:00
AWS GovCloud (米国東部)	us-gov-east-1	17:00 ~ 01:00
AWS GovCloud (米国西部)	us-gov-west-1	06:00 ~ 14:00

Amazon DocumentDB メンテナンスウィンドウを変更する

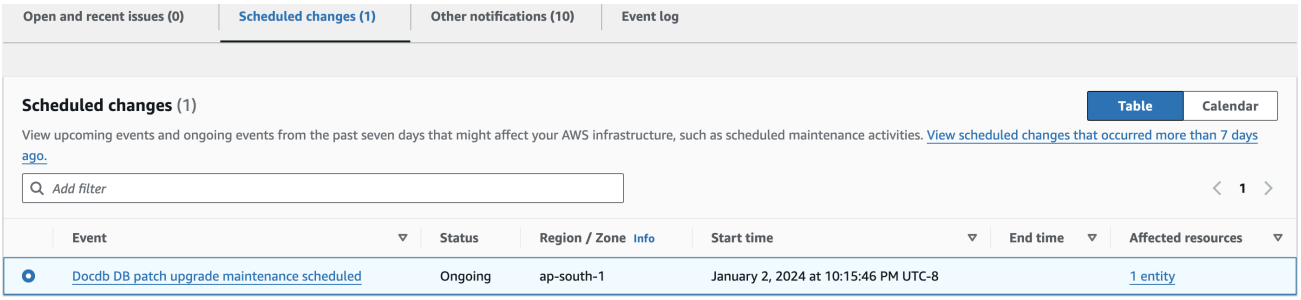
可能な最低トラフィックウィンドウを選択し、トラフィックパターンの変化に応じて時間の経過とともに調整します。クラスターまたはインスタンスは、スケールストレージオペレーションやインスタンスクラスの変更などのシステム変更が停止を必要とする場合にのみ、その変更が実際に必要である限り、ウィンドウ中に使用できなくなります。

メンテナンスウィンドウを変更するには

- クラスターの場合には、「Amazon DocumentDB クラスターの変更」を参照してください。
- インスタンスの場合には、「Amazon DocumentDB インスタンスの変更」を参照してください。

Amazon DocumentDB エンジンパッチの通知

必要なエンジンパッチが AWS リージョンで利用可能になると、そのリージョン内の影響を受ける Amazon DocumentDB クラスターを持つすべての AWS アカウントは、 Health Dashboard (AHD) と E メール (AWS アカウントのルートユーザーアドレスに送信される) を通じて通知を受け取ります。影響を受ける Amazon DocumentDB エンジンのバージョンごとに 1 つの通知が配信されます。これらは、AHD のスケジュールされた変更にあります。各通知には、パッチの可用性のタイミング、自動適用スケジュール、影響を受けるクラスター、リリースノートが一覧表示されます。



必要なエンジンパッチは、約 30 日間の 1 回のリードタイムに従います。リージョンでパッチが利用可能になると、Amazon DocumentDB は上記の通知を送信します。その時点で、パッチの`AutoAppliedAfterDate`は約 30 日後に に設定されます。その日まで、パッチは保留中のままになります。いつでも適用することも、クラスターのメンテナンスウィンドウを後日移動して延期することもできます。以降`AutoAppliedAfterDate`、パッチはクラスターの次のメンテナンスウィンドウ中に自動的に適用されます。

例えば、2026 年 6 月 1 日に利用可能になる必須パッチには、2026 年 7 月 1 日頃`AutoAppliedAfterDate`の があります。通知は 2026 年 6 月 1 日に届きます。何もしない

と、2026 年 7 月 1 日以降にクラスターの最初のメンテナンスウィンドウ中にパッチが自動的に適用されます。

通知を受け取った後、2 つのオプションがあります。自動適用日より前にパッチを自己適用するか、今後のメンテナンスウィンドウ (デフォルト) 中にパッチが自動適用されるまで待機します。自己適用するには、クラスターのメンテナンスとバックアップタブを開き、タイプ のエントリを探しますsystem-update。

> **Note**
>
> AHD の通知のステータスは、Amazon DocumentDB が新しいパッチバージョンで別のエンジンパッチをリリースするまで、進行中のままです。
> パッチを適用すると、クラスターのエンジンパッチバージョンが通知のバージョンと一致するように更新されます。を実行して新しいバージョンを確認しますdb.runCommand({getEngineVersion: 1})。

オプションのパッチと新しいマイナーバージョンでは、AHD または E メール通知は生成されません。これらを追跡するには、Amazon DocumentDB リリースノートをご覧ください。

強制パッチ (最も重要なセキュリティ修正のために予約された最もまれなカテゴリ) も、AHD と E メールで発表されます。必要なパッチとは異なり、メンテナンスウィンドウ外に適用されるため、上記の自動適用タイミングの例は適用されません。

プログラムによるパッチ通知への対応

AWS Health は Amazon EventBridge と統合されており、 AWS Lambda や Amazon Simple Queue Service (SQS) など、20 を超えるターゲットでイベント駆動型アプリケーションを構築できます。プログラムでエンジンパッチの可用性に対応するには、AWS_DOCDB_DB_PATCH_UPGRADE_MAINTENANCE_SCHEDULEDイベントに対して EventBridge を設定します。そこから、イベントデータをキャプチャしたり、追加のイベントを発生させたり、経由でプッシュ通知を送信したり AWS Console Mobile Application、その他の必要なアクションを実行したりできます。

Amazon DocumentDB がパッチをキャンセルした場合 (まれ)、AHD 通知とキャンセルに関する E メールが送信されます。Amazon EventBridge でAWS_DOCDB_DB_PATCH_UPGRADE_MAINTENANCE_CANCELLEDイベントコードを使用して、このケースを処理します。ルールの記述の詳細については、「Amazon EventBridge ユーザーガイド」を参照してください。

実施待ちの Amazon DocumentDB メンテナンスアクションを確認する

AWS マネジメントコンソール または を使用して AWS CLI 、クラスターまたはインスタンスで保留中のメンテナンスを確認します。

保留中の更新は`system-update`、エンジンパッチと OS 更新の両方をカバーするアクションタイプで表示されます。

更新が保留中の場合、次のことができます。

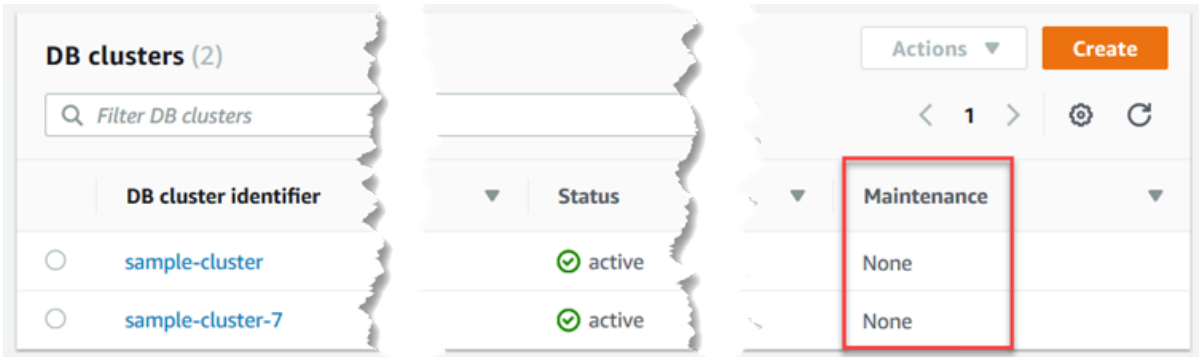
- すぐに適用します。
- 次のメンテナンスウィンドウにスケジュールします。
- より前にメンテナンスウィンドウを変更して、これを延期します (エンジンパッチと OS 更新のみ)`AutoAppliedAfterDate`。その日付が過ぎると、アクションは次のメンテナンスウィンドウ中に自動的に適用されます。`ForcedApplyDate` 合格すると、それ以上の延期はできません。

> **Note**
>
> アクションを実行しない場合、必要なエンジンパッチなどの必要なメンテナンスアクションは、今後のメンテナンスウィンドウ中に自動的に適用されます。オプションのパッチとマイナーバージョンは自動適用されません。

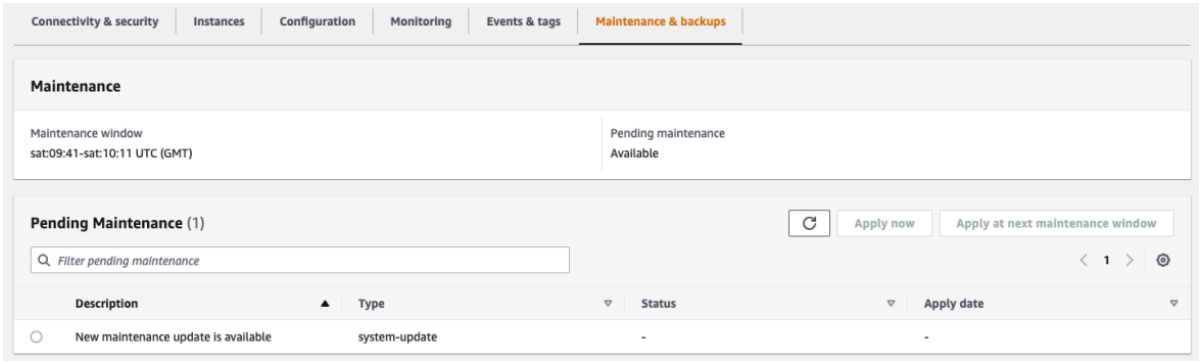
メンテナンスウィンドウは、保留中のオペレーションが開始されるタイミングを制御し、完了するまでにかかる時間を制御します。

Using the AWS マネジメントコンソール

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. 更新が保留中の場合、クラスターのメンテナンス列には利用可能、必須、または次のウィンドウが表示されます。



4. クラスターを開き、メンテナンスとバックアップを選択して保留中のメンテナンス項目を表示し、それらを処理します。



Using the AWS CLI

を実行してdescribe-pending-maintenance-actions、保留中のものを確認します。次の例は、保留中のアクションがないアカウントを示しています。

```
aws docdb describe-pending-maintenance-actions
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "PendingMaintenanceActions": []
}
```

アクションが保留中のアカウントは、次のような出力を返します。

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

リストは、 の形式で--filters、 を使用して特定のクラスターにスコープできますName=*filter-name*,Values=*resource-id*,...。受け入れられるフィルターNameはでdb-cluster-id、クラスター識別子または ARNs のリストを取得します。

Example

Linux、macOS、Unix の場合:

```
aws docdb describe-pending-maintenance-actions \
   --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

Windows の場合:

```
aws docdb describe-pending-maintenance-actions ^
   --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

日付の適用

保留中の各メンテナンスアクションには、最大 3 つの適用日があります。これらは の AWS CLI 出力に表示されdescribe-pending-maintenance-actions、アクションが実行されるタイミングを示します。フィールドはオプションのメンテナンスnull用です。

- **CurrentApplyDate**— 現在または次のメンテナンスウィンドウのいずれかで、アクションの実行がスケジュールされている場合。必須アクションと強制アクションに設定されます。
- **AutoAppliedAfterDate**— クラスターまたはインスタンスのメンテナンスウィンドウ中に自動適用が開始される日付。必要なアクションに入力します。

- **ForcedApplyDate**— 厳しい期限。この日付を過ぎると、メンテナンスウィンドウに関係なくアクションが自動的に実行されます。強制アクションに設定されます。

保留中のアクションを延期するには、メンテナンスウィンドウを の翌日に移動しますAutoAppliedAfterDate。がAutoAppliedAfterDate過ぎると、アクションは次のメンテナンスウィンドウ中に自動的に適用されます。ForcedApplyDate 合格すると、それ以上の延期はできません。正確な遅延ウィンドウはパッチによって異なります。日付は AHD 通知と AWS CLI 出力で公開されます。

Amazon DocumentDB エンジンの更新

保留中のエンジンパッチを特定したら、次のいずれかの手順を使用して適用またはスケジュールします。これらの手順は、 AWS マネジメントコンソール または から実行できます AWS CLI。

Using the AWS マネジメントコンソール

クラスターの更新を管理するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. 更新するクラスターを選択します。
4. アクションメニューから、次のいずれかを選択します。
   - 今すぐアップグレード — 保留中のメンテナンスをすぐに実行します。
   - 次のウィンドウでアップグレード — クラスターの次のメンテナンスウィンドウ中に実行します。

   クラスターのメンテナンスとバックアップタブのメンテナンス保留中セクションから、今すぐ適用または次のメンテナンスウィンドウで適用を使用することもできます (「」を参照実施待ちの Amazon DocumentDB メンテナンスアクションを確認する)。

   > **Note**
   >
   > 保留中のものがない場合、これらのオプションはすべて非アクティブです。

Using the AWS CLI

で保留中の更新を適用しますapply-pending-maintenance-action。

パラメータ

- **--resource-identifier**— 保留中のアクションターゲットとなるリソースの Amazon DocumentDB Amazon リソースネーム (ARN)。
- **--apply-action**— 適用する保留中のメンテナンスアクション。system-update を使用してエンジンパッチを適用します。
- **--opt-in-type**— オプトインリクエストのタイプ、またはオプトインリクエストを元に戻すかどうか。有効な値:
  - immediate— 今すぐ適用します。一度送信すると元に戻すことはできません。
  - next-maintenance— リソースの次のメンテナンスウィンドウ中に適用します。
  - undo-opt-in— 既存のオnext-maintenanceプトインをキャンセルします。

Example

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

パッチ適用中の読み取り可用性

Amazon DocumentDB エンジン 5.0 および 8.0 は、クラスターに複数のインスタンスがある場合、パッチ適用中の読み取り可用性を維持します。Amazon DocumentDB は、3 つのグループに

ローリング方式でリーダーインスタンスにパッチを適用するため、残りのリーダーは引き続きトラフィックを処理します。ライターは、パッチ適用中は一時的に使用できません。読み取りのダウンタイムをゼロにするには、読み取り設定を設定して、読み取りがライターにフォールバックしたり、primaryPreferred機能secondaryPreferredしたり、primarysecondary単独で読み取りのダウンタイムが発生したりします。

読み取り設定モード	ライターのアップグレード中	リーダーのアップグレード中	読み取りダウンタイムゼロに必要なリーダーの最小数
primary	読み取り/書き込みのダウンタイム	影響なし	該当なし
primaryPreferred	書き込みダウンタイム	影響なし	1
secondary	書き込みダウンタイム	読み取りダウンタイム (リーダーが 1 人のみの場合)	2
secondaryPreferred	書き込みダウンタイム	影響なし	1
nearest	書き込みダウンタイム	影響なし	1

リーダーがパッチを適用している間、クラスター全体の読み取りスループットは一時的に低下します。スループットを安定させるには、アップグレード前に追加のリーダーをプロビジョニングし、完了後に削除します。

エンジン 3.6 および 4.0 では、これらの読み取り可用性機能は適用されません。エンジンパッチにより、読み取りと書き込みの両方に影響するダウンタイムが長くなります。が実行するメジャーバージョンにアップグレードするには、「」を参照してくださいAmazon DocumentDB のインプレースメジャーバージョンアップグレード。

パッチのダウンタイムの長さ

エンジンパッチのダウンタイムはさまざまです。最も大きな要因は、パッチ適用時のインスタンスの CPU 使用率とメモリ負荷であるため、インスタンスの適切なサイズ設定が重要です。ダウンタイムを最小限に抑えるには、最新の Amazon DocumentDB メジャーエンジンバージョンを実行し、複数のアベイラビリティーゾーンにインスタンスを分散します。

パッチの更新と置換

Amazon DocumentDB は、リリース後にパッチをモニタリングします。まれに問題が特定された場合、Amazon DocumentDB は更新バージョンの準備中にロールアウトを一時停止します。この場合、パッチをまだ受信していないクラスターは、パッチを使用可能なメンテナンスアクションと見なさなくなり、 の対応するスケジュールされた変更通知 Health Dashboard が取り消されます。影響を受けるバージョンをすでに実行しているクラスターは引き続き正常に動作するため、ユーザーによるアクションは必要ありません。

更新されたパッチは間もなく続きます。リージョンで利用可能になると、「」で説明されているように、 Health Dashboard と E メールを通じて新しい通知が送信されますAmazon DocumentDB エンジンパッチの通知。

マイナーバージョンのアップグレード

Amazon DocumentDB は、メジャーバージョン 5.0 以降 (など`5.0.1`) でマイナーバージョンを発行します。マイナーバージョンは、5.0 より前のメジャーバージョンでは公開されません。マイナーバージョンの動作は、必須およびオプションのエンジンパッチとは異なります。

- 保留中のメンテナンスアクションとしては表示されず、自動適用もされません。
- AHD または E メール通知は生成されません。新しいマイナーバージョンは、Amazon DocumentDB リリースノートで発表されています。
- アップグレードするには、クラスターのエンジンバージョンを変更します (即時または次のメンテナンスウィンドウ中に) 。マイナーバージョンのアップグレードには短いダウンタイムが必要で、一方通行です。以前のマイナーバージョンにダウングレードすることはできません。グローバルクラスターの場合は、プライマリクラスターの前にセカンダリクラスターをアップグレードします。

詳細については、「」を参照してくださいAmazon DocumentDB マイナーバージョンアップグレード。

Amazon DocumentDB オペレーティングシステムの更新

インスタンスには OS の更新が必要になる場合があります。Amazon DocumentDB は OS を更新してパフォーマンスを向上させ、セキュリティを強化します。OS 更新では、クラスターエンジンのバージョンとインスタンスクラスは変更されません。エンジンパッチと同様に、OS 更新では、このトピックの上部で説明されているオプション/必須/強制ライフサイクルが使用されます。エンジンパッチとは異なり、OS 更新を延期すると、時間の経過とともにこれらのカテゴリに移行する可能性があります。OS 更新が利用可能になったらすぐに適用し、ビジネスニーズに合った時間にクラスターとインスタンスのメンテナンスウィンドウを設定します。

クラスターレベルのos-upgradeメンテナンスアクションを使用して、クラスター内のすべてのインスタンスに OS 更新を適用します。Amazon DocumentDB は、ローリング方式でインスタンスを一度に数回更新し、フェイルオーバーを最小限に抑えるためにプライマリインスタンスを最後に更新します。更新は、設定した個々のインスタンスのメンテナンスウィンドウではなく、クラスターのメンテナンスウィンドウ中に実行されます。

インスタンスが OS 更新を受信すると、バッファキャッシュは空で開始されます。ワーキングセットがストレージボリュームから再入力されるまで、そのインスタンスのクエリではレイテンシーが高くなり、 が低くなる可能性がありますBufferCacheHitRatio。

Amazon DocumentDB がプライマリインスタンスを更新すると、フェイルオーバーによってレプリカが新しいプライマリに昇格されます。クラスターエンドポイントを使用して、アプリケーションがこれを透過的に処理できるようにします。インスタンスの更新中に読み取り可用性を維持するには、読み取り設定を secondaryPreferredまたは に設定primaryPreferredして、読み取りが使用可能なインスタンスにフォールバックできるようにします。潜在的なフェイルオーバーターゲット (優先度が最も高い階層のレプリカ) をプライマリと同じインスタンスクラスに保持します。これにより、昇格後の書き込みパフォーマンスの低下を回避できます。詳細については、「Amazon DocumentDB フェイルオーバー」を参照してください。

クラスターレベルのsystem-updateアクションos-upgradeとインスタンスレベルのアクションの両方が、使用可能なアクションdescribe-pending-maintenance-actionsとして に同時に表示される場合があります。ただし、両方を同時にスケジュールすることはできません。インスタンスレベルのsystem-updateアクションがいずれかのインスタンスでアクティブにスケジュールされている場合は、クラスターレベルのos-upgradeアクションをスケジュールする前にキャンセルまたは完了する必要があります。その逆も同様です。

> **Important**
>
> Amazon DocumentDB インスタンスは OS 更新のためにオフラインになります。マルチインスタンスクラスターは影響を最小限に抑えます。単一インスタンスクラスターを実行する場合は、一時的に更新用のセカンダリを追加し、後で削除できます。セカンダリが存在する間は、通常の料金が発生します。

> **Note**
>
> インスタンスレベルの`system-update`アクションは、下位互換性のために引き続き使用できます。使用する必要がある場合は、まずレプリカを更新し、最後にプライマリを更新します。パッチ中のフェイルオーバーによりダウンタイムが長くなる可能性があるため、同時にパッチを適用しないでください。

新しいオプションの OS 更新が到着したときにイベントを取得するには、セキュリティパッチ適用イベントカテゴリ`RDS-EVENT-0230`で をサブスクライブします。詳細については、「Amazon DocumentDB イベントにサブスクライブする」を参照してください。

> **Note**
>
> コンプライアンスのために、オプションの更新と必要な更新を最新の状態に保つ必要がある場合があります。メンテナンスウィンドウ中に`os-upgrade`アクションを定期的に適用します。

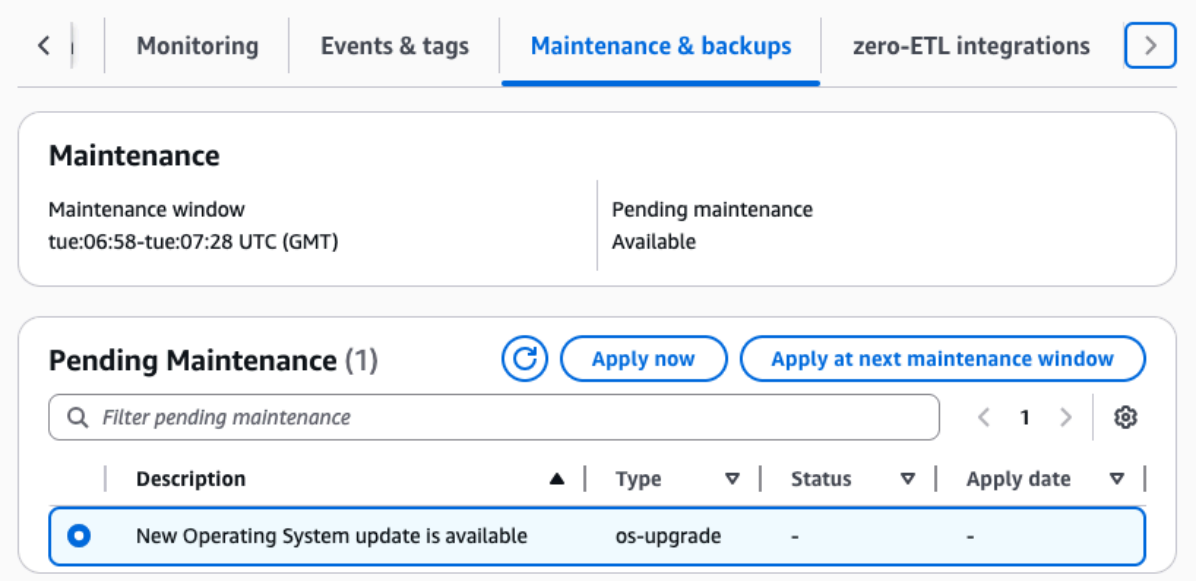
OS の更新は特定のインスタンスクラスに関連付けられているため、異なるインスタンスは異なるタイミングで適格になります。クラスターが最新のエンジンパッチにない場合、OS 更新が表示されないことがあります。最新のエンジンパッチを最初に適用します (「」を参照Amazon DocumentDB エンジンの更新)。

更新が利用可能かどうかを確認する AWS CLI には、 AWS マネジメントコンソール または を使用します。

Using the AWS マネジメントコンソール

コンソールから OS 更新を確認するには:

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、クラスターを選択し、クラスター名を選択します。
3. メンテナンスとバックアップタブを選択します。
4. 保留中のメンテナンスでは、OS 更新が利用可能な場合、 os-upgradeアクションが表示されます。



5. os-upgrade アクションを選択し、今すぐ適用または次のメンテナンスウィンドウで適用を選択します。値が次のウィンドウの場合、アクションが開始されていない限り、遅延アップグレードを延期できます。

Using the AWS CLI

保留中の OS 更新を確認します。

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

OS 更新は、クラスターレベルで としてos-upgrade、インスタンスレベルで として表示されますsystem-update。クラスターレベルのos-upgradeアクションを使用します。

Example

次の の例では、OS 更新をすぐに適用します。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

ユーザーが起動する更新

一部の変更は、インスタンスクラスをメモリが多かれ少なかれあるインスタンスクラスにスワップしたり、クラスターのパラメータグループを変更したりするなどです。Amazon DocumentDB では、これらの処理は、開始する更新とは異なります。詳細については、以下を参照してください。

- Amazon DocumentDB クラスターの変更
- Amazon DocumentDB インスタンスの変更

保留中のユーザー主導の変更を一覧表示するには:

Example

インスタンスの保留中のユーザー主導の変更を一覧表示するには

Linux、macOS、Unix の場合:

```
aws docdb describe-db-instances \
    --query 'DBInstances[*].
[DBClusterIdentifier,DBInstanceIdentifier,PendingModifiedValues]'
```

Windows の場合:

```
aws docdb describe-db-instances ^
    --query 'DBInstances[*].
[DBClusterIdentifier,DBInstanceIdentifier,PendingModifiedValues]'
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

この例では、 sample-cluster-instanceには への保留中の変更がありdb.r5.xlarge、sample-cluster-instance-2 には何もありません。

```
[
```

```
    [
        "sample-cluster",
        "sample-cluster-instance",
        {
            "DBInstanceClass": "db.r5.xlarge"
        }
    ],
    [
        "sample-cluster",
        "sample-cluster-instance-2",
        {}
    ]
]
```

グローバルクラスターのパッチ適用

グローバルクラスターでは、プライマリおよびセカンダリの各メンバークラスターは、独自のメンテナンスウィンドウ中にアップグレードされます。必要なエンジンパッチがすべてのリージョンで利用可能になると、AHD と E メール通知が送信されます。オプションのパッチと新しいマイナーバージョンは通知を生成しません。Amazon DocumentDB リリースノートを確認してください。

自己適用の場合は、常に最初にセカンダリにパッチを適用し、最後にプライマリにパッチを適用します。この順序により、ロールアウト全体でフェイルオーバーとスイッチオーバーが維持されます。

> **Important**
>
> 誤って最初にプライマリにパッチを適用する場合は、すべてのセカンダリをできるだけ早く同じバージョンに更新してください。フェイルオーバーとスイッチオーバーは、すべてのクラスターが同じバージョンになるまで無効のままになります。

アクションを実行しない場合、パッチは各クラスターの次のメンテナンスウィンドウに自動的に適用されます。最初にセカンダリ、次にセカンダリが完了した後のウィンドウでプライマリです。

プライマリ DB クラスターとセカンダリ DB クラスターを同じバージョンに保ちます。マネージドクロスリージョンフェイルオーバーは、すべてのクラスターが同じエンジンバージョンとパッチレベルを共有している場合にのみ、グローバルデータベースで機能します。プライマリよりも新しいエンジンバージョンを使用する新しいセカンダリを追加する場合も同様です。プライマリのバージョンで新しいセカンダリを作成してから、グローバルデータベースに結合します。

パッチ通知後、フェイルオーバーとスイッチオーバーが機能し続けるように、できるだけ早くプライマリとセカンダリを最新バージョンにアップグレードします。フェイルオーバーまたはスイッチオーバーリクエストが拒否された場合は、クラスター間でエンジンパッチバージョンを比較します。一致しない場合は、利用可能なパッチを遅延しているクラスターに適用します。

サービスにリンクされたロールを理解する

Amazon DocumentDB (MongoDB 互換) は、 AWS Identity and Access Management (IAM) サービスにリンクされたロールを使用します。サービスにリンクされたロール は、Amazon DocumentDB に直接リンクされた特殊なタイプの IAM ロールです。サービスにリンクされたロールは Amazon DocumentDB によって事前定義されており、サービスがユーザーに代わって他の AWS サービスを呼び出すために必要なすべてのアクセス許可が含まれています。

サービスにリンクされたロールを使用すると、必要な権限を手動で追加する必要がないため、Amazon DocumentDB の使用が簡単になります。Amazon DocumentDB は、サービスにリンクされたロールの権限を定義します。特に定義されていない限り、Amazon DocumentDB のみがそのロールを引き受けることができます。定義されたアクセス許可には、信頼ポリシーとアクセス許可ポリシーが含まれ、そのアクセス許可ポリシーを他の IAM エンティティにアタッチすることはできません。

ロールを削除するには、まず関連リソースを削除します。これにより、リソースにアクセスするためのアクセス許可を誤って削除することができないため、Amazon DocumentDB リソースが保護されます。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連携するAWS サービス」を参照して、サービスにリンクされたロール列がはいになっているサービスを見つけてください。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、はい リンクを選択します。

Amazon DocumentDB のサービスにリンクされたロールの許可

Amazon DocumentDB (MongoDB 互換) は、AWSServiceRoleForRDS という名前のサービスにリンクされたロールを使用して、Amazon DocumentDB がクラスターに代わって AWS サービスを呼び出すことを許可します。

サービスにリンクされたロール AWSServiceRoleForRDS では、以下のサービスを信頼してロールを引き受けます。

- `docdb.amazonaws.com`

ロールの許可ポリシーは、指定したリソースに対して以下のアクションを実行することを Amazon DocumentDB に許可します。

- ec2 でのアクション:
  - AssignPrivateIpAddresses
  - AuthorizeSecurityGroupIngress
  - CreateNetworkInterface
  - CreateSecurityGroup
  - DeleteNetworkInterface
  - DeleteSecurityGroup
  - DescribeAvailabilityZones
  - DescribeInternetGateways
  - DescribeSecurityGroups
  - DescribeSubnets
  - DescribeVpcAttribute
  - DescribeVpcs
  - ModifyNetworkInterfaceAttribute
  - RevokeSecurityGroupIngress
  - UnassignPrivateIpAddresses
- sns でのアクション:
  - ListTopic
  - Publish
- cloudwatch でのアクション:
  - PutMetricData
  - GetMetricData
  - CreateLogStream
  - PullLogEvents
  - DescribeLogStreams
  - CreateLogGroup

> **Note**
>
> サービスリンクロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するにはアクセス許可を設定する必要があります。次のエラーメッセージが返される場合があります。
> リソースを作成できません。サービスにリンクされたロールを作成するために必要なアクセス許可があることを確認します。それ以外の場合は、時間をおいてからもう一度お試しください。
> このエラーが表示された場合は、次のアクセス許可が有効であることを確認します。
>
> ```
> {
>     "Action": "iam:CreateServiceLinkedRole",
>     "Effect": "Allow",
>     "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
> AWSServiceRoleForRDS",
>     "Condition": {
>         "StringLike": {
>             "iam:AWSServiceName":"rds.amazonaws.com"
>         }
>     }
> }
> ```

詳細については、「IAM ユーザーガイド」の「サービスリンクロールの許可」を参照してください。

Amazon DocumentDB でのサービスにリンクされたロールの作成

サービスリンクロールを手動で作成する必要はありません。クラスターを作成すると、Amazon DocumentDB がサービスにリンクされたロールを作成します。

サービスにリンクされたこのロールを削除したが、再作成する必要がある場合は、同じプロセスで、アカウントにロールを再作成することができます。クラスターを作成すると、Amazon DocumentDB によってサービスにリンクされたロールが再度作成されます。

Amazon DocumentDB サービスにリンクされたロールの変更

Amazon DocumentDB は、AWSServiceRoleForAmazonMQ サービスリンクロールの編集を許可しません。サービスリンクロールの作成後は、さまざまなエンティティがロールを参照する可能性がある

ため、ロール名を変更することはできません。ただし、IAM を使用したロールの説明の変更はできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

Amazon DocumentDB でのサービスにリンクされたロールの削除

サービスにリンクされたロールを必要とする機能またはサービスを使用する必要がなくなった場合は、そのロールを削除します。これにより、アクティブにモニタリングまたは保守されていない未使用のエンティティがなくなります。ただし、サービスにリンクされたロールを削除する前に、すべての クラスターを削除する必要があります。

Amazon DocumentDB サービスにリンクされたロールのクリーンアップ

IAM を使用してサービスにリンクされたロールを削除するには、まずそのロールにアクティブなセッションがないことを確認し、そのロールで使用されているリソースをすべて削除する必要があります。

サービスにリンクされたロールがアクティブなセッションを持っているかどうかをコンソールを使用して確認するには

1. にサインイン AWS マネジメントコンソール し、 で IAM コンソールを開きますhttps://console.aws.amazon.com/iam/。
2. IAM コンソールのナビゲーションペインで、ロール を選択してから、AWSServiceRoleForRDS ロールの名前 (チェックボックスではない) を選択します。
3. 選択したロールの 概要 ページで、アクセスアドバイザー タブを選択します。
4. [アクセスアドバイザー] タブで、サービスにリンクされたロールの最新のアクティビティを確認します。

> **Note**
>
> Amazon DocumentDB が AWSServiceRoleForRDS ロールを使用しているかどうかわからない場合は、ロールの削除を試みることができます。サービスでロールが使用されている場合、削除は失敗し、ロールが使用されている リージョンが表示されます。ロールが使用されている場合は、ロールを削除する前にセッションが終了するのを待つ必要があります。サービスにリンクされたロールのセッションを取り消すことはできません。

AWSServiceRoleForRDS ロールを削除する場合、最初に すべて のインスタンスおよびクラスターを削除する必要があります。インスタンスとクラスターを削除する方法については、次のトピックを参照してください。

- Amazon DocumentDB インスタンスの削除
- Amazon DocumentDB クラスターの削除

Amazon DocumentDB サービスにリンクされたロールでサポートされているリージョン

Amazon DocumentDB ではサービスを利用できるすべてのリージョンで、サービスにリンクされたロールの使用がサポートされます。詳細については、「https://docs.aws.amazon.com/documentdb/latest/devguide/regions-and-azs.html#regions-and-azs-availability」を参照してください。

パフォーマンスの最適化

この章では、Amazon DocumentDB のパフォーマンスを最適化し、一般的なパフォーマンスの問題をトラブルシューティングするためのガイダンスを提供します。

トピック

- 一般的なパフォーマンスの問題
- アンチパターン
- パフォーマンス向上のヒント
- クエリプラン分析
- インデックスのみのスキャン (対象クエリ)
- キャッシュフィルターの計画
- Amazon DocumentDB のベストプラクティス
- Amazon DocumentDB インデックスの管理
- コレクションレベルのドキュメント圧縮の管理
- Amazon DocumentDB 8.0 でのディクショナリベースの圧縮の管理

一般的なパフォーマンスの問題

このセクションでは、Amazon DocumentDB で発生する可能性がある一般的なパフォーマンスの問題とその解決方法について説明します。

トピック

- CPU 使用率が高い
- Amazon DocumentDB との接続の問題
- クエリの実行が遅い

CPU 使用率が高い

CPU 使用率が高いことは、Amazon DocumentDB の使用中に発生する可能性のある問題の 1 つです。このセクションでは、問題を解決するためのインサイトを提供します。

識別 - 問題を特定する

Amazon DocumentDB インスタンスの CPU 使用率が高いと、すべてのデータベースオペレーションでレイテンシーが増加するため、アプリケーションの速度が低下します。Amazon DocumentDB の CPU 使用率が最適なレベルを超えると、クエリの実行時間が長くなり、接続確立に時間がかかり、アプリケーションの応答性が低下します。プライマリインスタンスの CPU ピークは書き込みオペレーションを遅らせ、レプリカのピークは読み取りオペレーションのレイテンシーを長くします。

Amazon DocumentDB の CPU 使用率が高い一般的な原因には、次のようなものがあります。

- コレクションスキャン: 適切なインデックスのないクエリは、完全なコレクションスキャンを強制し、過剰な CPU を消費してすべてのドキュメントを調べます。
- 接続スパイク: アプリケーション接続の突然の急増により、認証とセッション管理に必要な CPU リソースが圧迫されます。
- 複雑な集約: マルチステージオペレーション (ソート、グループ化、結合) は、特に最適化されたインデックスのない大規模なデータセットで、かなりの CPU を消費します。
- 非効率的なクエリ: CPU 使用率が高いのは、大きな結果セットと正規表現オペレーションをスキャンした結果です。
- 書き込み負荷の高いワークロード: 頻繁な更新と一括挿入は、ドキュメント処理、インデックス更新、トランザクションログ記録により CPU を負担します。

調査 - メトリクスを収集する

Amazon DocumentDB は、Amazon CloudWatch を介したモニタリングを提供します。Amazon DocumentDB メトリクスは、クラスターレベルとインスタンスレベルのメトリクスに広く分類できます。CPU、メモリ、および接続に関連するリソースについては、リソースの消費量がインスタンスに固有であるため、インスタンスレベルのメトリクスを確認します。CPU スパイクを示しているインスタンス (プライマリまたはレプリカ) を特定し、スパイク時間を書き留めます。

の CPU 使用率の CloudWatch メトリクス`CPUUtilization`。

診断 - 根本原因を見つける

CPU スパイクのインスタンスとタイムラインを特定したら、次に CPU 使用率に大きな影響を与えている原因を診断します。CPU スパイクを引き起こす原因は複数あるため、 では、以下から原因を特定できます。

接続: DatabaseConnectionsおよび を含む CloudWatch メトリクスをチェックしDatabaseConnectionsMax、接続にスパイクがあり、CPU スパイクタイムと一致する場合、短時間の接続の急増が CPU スパイクの原因である可能性があります。

クエリ: 接続でない場合、クエリが原因である可能性があります。パフォーマンスインサイトを有効にしている場合は、インスタンスのパフォーマンスインサイトコンソールに移動し、インスタンスのクエリを確認します。クエリを確認するもう 1 つの方法は、プロファイラーからのものです。

解決 - 問題を修正する

接続バースト: 接続バーストが CPU スパイクの原因である場合は、アプリケーションで接続を再利用するか、接続プーリングを実装することを検討してください。高接続の最適化の詳細については、「」を参照してくださいthe section called “接続の問題”。

クエリディストリビューション: CPU スパイクがプライマリインスタンスの読み取りクエリによって発生した場合は、クエリ結果に結果整合性がある場合は、読み取りオペレーションをレプリカインスタンスにリダイレクトすることを検討してください。

クエリの最適化: クエリまたは一連のクエリによって CPU が急増している場合は、クエリがインデックスを使用していない場合にインデックスを追加するなど、クエリを最適化することを検討してください。クエリの最適化の詳細については、「」を参照してくださいthe section called “クエリの実行が遅い”。

インスタンススケーリング: 最適化作業後も CPU 使用率が一貫して高い場合は、ワークロード要件を効果的に処理するために、より多くの CPU 容量を持つより大きなインスタンスクラスにアップグレードします。

サーバーレス: 短期間の CPU スパイクを引き起こす予測不可能なワークロードの場合は、 の使用を検討してくださいAmazon DocumentDB Serverless の使用。サーバーレスインスタンスは、需要に基づいてリソースを自動的にスケーリングするため、このようなワークロードに固定インスタンスサイズをプロビジョニングする必要はありません。

Amazon DocumentDB との接続の問題

識別 - 問題を特定する

一般的な原因

接続の問題は通常、次の 3 つの主要な領域から発生します。

接続プールの枯渇は、クライアント側の接続プール制限またはサーバー側のインスタンス制限を通じて、アプリケーションが Amazon DocumentDB への最大許容接続数に達したときに発生します。この条件により、新しい接続リクエストがキューに入れられたり拒否されたりすると、アプリケーションのパフォーマンスが低下し、タイムアウトが発生し、障害が発生する可能性があります。

認証オーバーロードは、Amazon DocumentDB で過剰な同時認証リクエストが発生した場合、特に短期間に 1,000 を超える新しい接続を処理する場合に発生します。認証メンテナンス中、Amazon DocumentDB はセッションマップに排他的ロックを保持し、メンテナンスが完了するまで後続の認証試行をキューに入れます。

Amazon DocumentDB の設定問題は、多くの場合、ネットワーク、セキュリティ、クライアント設定の設定ミスが原因です。これには、誤ったセキュリティグループ設定、不適切な VPC 設定、SSL/TLS 証明書の問題などの項目が含まれます。安全で信頼性の高いデータベースアクセスを維持するには、適切な設定を理解することが不可欠です。

診断 - 根本原因を見つける

接続プール

MongoClient インスタンスを作成すると、接続プールが初期化されます。各プールは、2 つの主要なパラメータに基づいて接続を維持します。

minPoolSize - 維持される接続の最小数

maxPoolSize - 最大許容接続数

リクエストで接続が必要な場合:

1. プールは使用可能なアイドル接続をチェックします
2. 存在せず、プールサイズ < maxPoolSize の場合、新しい接続が作成されます。
3. maxPoolSize でリクエストが待機キューに入る場合
4. キューがフルまたはタイムアウトに達すると、MongoWaitQueueFullException がスローされます。

待機キューの動作は、次のパラメータを介して処理されます。

waitQueueTimeoutMS - 接続の最大待機時間

waitQueueSize - キューに入れられたリクエストの最大数

新しいプールが毎回作成される Amazon DocumentDB に接続するための問題のあるアプローチの例を次に示します。

```
for(Request request : requests) {
    MongoClient client = MongoClients.create(settings);
    // Process request
    client.close();
}
```

モニタリングすべき重要な CloudWatch メトリクスは次のとおりです。

- `DatabaseConnections` - 1 分間隔で実行されたインスタンスで開いている接続 (アクティブおよびアイドル) の数。
- `DatabaseConnectionsMax` - 1 分間にインスタンスで開いているデータベース接続 (アクティブおよびアイドル) の最大数。
- `DatabaseConnectionsLimit` - 任意の時点でインスタンスで許可される同時データベース接続 (アクティブおよびアイドル) の最大数。
- `LowMemNumOperationsThrottled` - 1 分間に使用可能なメモリが少ないためにスロットリングされたリクエストの数。

インスタンスクラスあたりのクォータについては、「」を参照してくださいインスタンスクォータ。

アプリケーションレベルでの接続プールの問題の一般的な警告サインは次のとおりです。

- 接続の取得時間の増加
- 待機キューサイズの増加
- タイムアウト例外の数の増加

認証オーバーロード

Amazon DocumentDB への接続は、次のようなフローに従います。

接続リクエスト → SSL ハンドシェイク → 認証 → セッション作成 → 接続準備完了

>1,000 個の新しい接続を処理すると、SSL ハンドシェイクの完了後に追加の接続リクエストが認証キューに入ります。これらのオーバーロードイベント中は、アプリケーションからの平均接続時間が長くなります。

モニタリングすべき重要な CloudWatch メトリクスは次のとおりです。

- `DatabaseConnections` - 1 分間隔で実行されたインスタンスで開いている接続 (アクティブおよびアイドル) の数。
- `DatabaseConnectionsMax` - 1 分間にインスタンスで開いているデータベース接続 (アクティブおよびアイドル) の最大数。
- `DatabaseConnectionsLimit` - 任意の時点でインスタンスで許可される同時データベース接続 (アクティブおよびアイドル) の最大数。

設定の問題

最も一般的な設定の問題は、プライベートネットワーク環境にアクセスせずに環境からプライベート Amazon DocumentDB クラスターエンドポイントに接続しようとするときに発生します。Amazon DocumentDB は仮想プライベートクラウド (VPC) 専用で、現在パブリックエンドポイントをサポートしていません。ラップトップまたは VPC 外のローカル開発環境から Amazon DocumentDB クラスターに直接接続することはできません。

これは、次のようなエラーで発生します。

```
Error: couldn't connect to server...
Failed to connect to...
exception: connect failed
connection attempt failed
```

セキュリティグループの設定が正しくないと、接続が失敗する可能性があります。Amazon DocumentDB クラスターは、デフォルトで TCP ポート 27017 で接続をリッスンします。クラスターがデプロイされたポートとは異なるポートに接続しようとした場合、またはアプリケーションがクラスターのイングレスセキュリティグループ設定でカバーされていない場合、アプリケーションは失敗します。

証明書管理が正しくないと、接続の問題が発生する可能性もあります。デフォルトでは、転送時の暗号化は、新しく作成された Amazon DocumentDB クラスターに対して有効になっています。転送中の暗号化が有効になっている場合、global-bundle.pem 証明書を使用してクラスターに接続するには、TLS を使用した安全な接続が必要です。誤った証明書を使用しようとすると、次のようなエラーが表示されます。

```
unable to get local issuer certificate
```

TLS パラメータを指定せずに TLS が有効になっているクラスターに接続しようとすると、次のようなエラーが表示されます。

```
Server selection timed out after 30000 ms
```

解決 - 問題を修正する

接続プール: ワークロードの要件に合わせてプールサイズを実装または調整して、接続プーリングを確認します。最適なプール設定は、ワークロードと要件によって異なります。minPoolSize は、コア接続の準備ができて使用可能になるように維持し、maxWaitTime はプールが枯渇した場合にすばやく失敗するように短くする必要があります。

新しいプールを毎回作成せずに 1 つのプールを再利用する方法の例を次に示します。

```
MongoClient client = MongoClients.create(settings);
    for(Request request : requests) {
    // Process request
}
```

認証の過負荷: 段階的な接続のランプアップを実装し、新しい接続を一度に 1,000 に制限することで、認証を管理します。接続プーリングを使用して、認証された接続を効果的に再利用します。接続で Amazon DocumentDB クラスターが過負荷にならないようにするには、接続のランプアップ戦略を実装します。

```
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;
        }
    }
}
```

許可された接続の合計数を制限するように接続プールを設定することもできます。

```
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();
```

設定の問題: アプリケーションが Amazon DocumentDB リソースがあるプライベート VPC とサブネットにアクセスできることを確認します。VPC ピアリングを使用している場合は、開発者ガイド「VPC ピアリング接続のトラブルシューティング」で詳細を確認してください。ナレッジセンターの記事「インターネットから VPC 内の Amazon EC2 インスタンスへの接続に関する問題のトラブルシューティング方法を教えてください」も参照できます。

セキュリティグループ設定では、アプリケーションからの接続を許可するには、Amazon DocumentDB セキュリティグループに進入ルールを含める必要があります。

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

クラスターが TLS 暗号化で設定されている場合は、global-bundle.pem という名前の Amazon DocumentDB の TLS 証明書をダウンロードし、クラスターに接続するときに使用します。

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

長期ソリューション

より大きなインスタンスクラスにアップグレードするか、リードレプリカを追加して接続負荷を分散することで、インスタンスのスケーリングが必要になる場合があります。適切な負荷分散の実装により、クラスター全体で最適なリソース使用率が確保されます。

アプリケーションの変更では、堅牢な接続処理、包括的なモニタリング、および接続プーリングのベストプラクティスの遵守に焦点を当てる必要があります。これには、適切なエラー処理と接続ライフサイクル管理が含まれます。

アーキテクチャの改善には、可変ワークロードに Amazon DocumentDB Serverless を採用すること、高度な再試行ロジックを実装すること、耐障害性を考慮した設計が含まれる場合があります。接続管理をより適切に処理するために、アプリケーションアーキテクチャの再構築を検討してください。

ベストプラクティス

接続プール

適切な接続プールの管理とモニタリングにより、アプリケーションは安定したデータベース接続を維持しながら、システムの信頼性とパフォーマンスに影響を与える可能性のある枯渇シナリオを防ぐことができます。ワークロードの特性に基づいて、適切なタイムアウトを設定し、プールのサイズを設定します。

接続プールの設定例

```
MongoClientSettings settings = MongoClientSettings.builder()
    .applyToConnectionPoolSettings(builder ->
        builder.maxSize(10))
    .applyToConnectionPoolSettings(builder ->
        builder.maxWaitQueueSize(2))
    .applyToConnectionPoolSettings(builder ->
        builder.maxConnectionIdleTime(10, TimeUnit.MINUTES))
    .build();
```

詳細については、Amazon DocumentDB を使用した回復力のあるアプリケーションの構築 – パート 1: クライアント設定」を参照してください。

認証オーバーロード

常に、ワークロードに基づいてパラメータの適切な値を使用して接続プーリングを実装します。段階的な接続確立手法を使用し、可能な場合は永続的な接続を維持します。適切な接続クリーンアップを実装して、アイドル状態のリソースが浪費されないようにします。

設定の問題

アプリケーションから Amazon DocumentDB リソースへの適切なルーティングが設定されていることを確認します。転送中の暗号化に TLS を使用し、最小特権アクセスを実装します。Amazon DocumentDB 認証情報を検証し、接続文字列値を検証します。

クエリの実行が遅い

識別 - 問題を特定する

実行速度の遅いクエリは、ビジネス要件で予想される実行時間を超えるクエリです。スロークエリを構成するものの定義は、ワークロードによって異なります。プロファイラーログまたは Performance Insights を使用してスロークエリを特定できます。

1. まだ有効になっていない場合は、プロファイラーを有効にします。プロファイラーログは、ロググループ の Amazon CloudWatch に書き込まれます`/aws/docdb/<cluster-identifier>/profiler`。CloudWatch Logs Insights を使用してクエリを実行します。

   ecommerce.products コレクションの 10 個の最も遅いクエリを取得するための CloudWatch ログ分析クエリの例:

```
filter ns="ecommerce.products" | sort millis desc | limit 10
```

2. Performance Insights を使用して、高価なクエリをほぼリアルタイムで特定します。Performance Insights がまだ有効になっていない場合は有効にします。

   a. AWS コンソールを開き、Amazon Amazon DocumentDB に移動し、Performance Insights に移動して、クラスターインスタンスを選択します。

   b. DB ロード (AAS) タイムラインと上位クエリ (DB ロード別) を確認します。クエリダイジェストを展開して、リテラル子ステートメントを表示します。

   c. 分析が必要なクエリをキャプチャします。

> **Note**
>
> Performance Insights のすべてのクエリが非効率またはスロークエリであるとは限りません。

調査 — 情報を収集する

1. プロファイラーは、mris、nreturned、planSummary (インデックス使用状況) など、クエリ実行プランとそれに関連付けられた主要なメトリクスを提供します。

```
{
    "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"
}
```

COLLSCAN クエリを検索するには:

```
filter planSummary="COLLSCAN" | sort millis desc | limit 20
```

2. Performance Insights を使用して、待機、ロード、リソースへの影響 (クエリあたりの IOPS や CPU など) などのクエリ実行の傾向をリアルタイムで分析します。

   Performance Insights はクエリ実行プランを提供しないため、クエリをキャプチャし、Amazon DocumentDB クラスターのクエリで explain("executionStats") を実行します。

```
db.ecommerce.products.find().explain("executionStats")
```

3. 必要に応じて、プロファイラーメトリクスを Performance Insights データに関連付けます (たとえば、プロファイラーの高ミリスクエリを Performance Insights の上位クエリと一致させる)。

診断 — 根本原因を見つける

このステップでは、クエリプランを診断して、潜在的な最適化を特定します。次のフローを使用する - 症状 → 考えられる原因:

- 症状: planSummary: "COLLSCAN"

  原因: インデックスがないか、正しくありません。
- 症状: $group / $sort で集計が遅い

  原因: 集約パイプラインがメモリ内のデータを過剰に処理している可能性があります。
- 症状: Performance Insights が IO 待機別にグループ化された DB Load を表示している間、レイテンシーは高くなりますが、インデックスが使用されます (planSummary: "IXSCAN")。

  原因: I/O バインドクエリ。インデックスまたはワーキングセットがインスタンスで使用可能なバッファキャッシュを超えています。
- 症状: PI は CPU 待機状態を示し、クエリが少ないため AAS が高い

  原因: CPU バインドクエリ (複合集計、$regex)。
- 症状: 多くのスロークエリはあるが、高価な planSummary は表示されない

  原因: 外部ボトルネック (ネットワーク、スロットリング、メンテナンスタスク、クエリの量)。

解決 — 問題を修正する

修正を実装するときは、必ず explain("executionStats") で検証し、Performance Insights DB Load をモニタリングします。

1. planSummary: "COLLSCAN"
   - ターゲットインデックスを作成します。

     例: { category, price } でフィルタリングし、価格の降順でソートする頻繁なクエリの場合:

```
db.products.createIndex({ category: 1, price: -1 })
```

2. $group / $sort による集計が遅い
   - $match と $project を早期にプッシュして、$group / $sort に流れるドキュメントを減らします。
   - パイプラインの早い段階でフィールドの数を制限します。

```
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. Performance Insights が IO 待機別にグループ化された DB ロードを表示する間のレイテンシーが高い
   - BufferCacheHitRatio が低いか、インスタンスの ReadIOPS が高いかを検証します。インスタンスメモリを増やす (インスタンスクラスのアップグレード、例: r6g.large → r6g.xlarge) か、NVMe インスタンスクラスを使用します。
   - インデックスのフットプリントを削減します。
   - リードトラフィックをオフロードするリードレプリカを追加します (クエリが結果整合性で許容できる場合はreadPreference 設定を使用してレプリカのトラフィックをリダイレクトします) 。
4. PI は CPU 待機状態を示し、クエリが少ないため AAS が高い
   - 高価な $regex をインデックス付きプレフィックス検索または $text インデックスに置き換えます。
   - 書き込み増幅を減らすためにバッチ書き込み (挿入) 。

> **Note**
>
> これらの変更を Production に昇格させる前に、必ずより低い環境 (非本番環境) で変更をテストしてください。

アンチパターン

このセクションでは、Amazon DocumentDB のパフォーマンスに悪影響を及ぼす可能性がある一般的なアンチパターンについて説明します。

トピック

- 3 つ以上の属性を持つ複合インデックス
- 長時間実行されるクエリ
- 低使用インデックスと冗長インデックス
- 大きな配列を持つマルチキーインデックス

3 つ以上の属性を持つ複合インデックス

概要

複合インデックスまたは複合インデックスは、単一のインデックス構造内の複数のフィールドへの参照を維持します。これらのインデックスは、複数のフィールドを同時にフィルタリングするか、フィルタリングとソート操作を組み合わせるクエリのパフォーマンスを最適化します。また、左端のインデックス付きフィールドに対する単一条件クエリにも有効です。データベースは、これらのインデックスエントリを活用して、完全なコレクションスキャンを実行せずに、一致するドキュメントを効率的に見つけます。

Amazon DocumentDB は、複合インデックスを使用して、インデックスプレフィックスと呼ばれる概念であるインデックス付きフィールドの先行サブセットを含むクエリをサポートできます。たとえば、 に複合インデックスがある場合`{state: 1, city: 1, zipcode: 1}`、Amazon DocumentDB はフィールド「state」のみ、フィールド「state」と「city」を一緒に使用するクエリ、または 3 つのフィールド「state」、「city」、「zipcode」をすべて効率的に処理できます。ただし、クエリでは、その間のフィールドをスキップすることなく、左から右のフィールドを使用する必要があります。つまり、フィールド「city」または「zipcode」のみを使用するクエリ、または「state」や「zipcode」などの組み合わせ (「city」のスキップ) を使用するクエリは、インデックスを完全に利用することはできません。

ほとんどの実世界のシナリオでは、3 つ以下の属性を持つ複合インデックスは通常、最適なパフォーマンスとリソース効率を実現します。複合インデックスに 3 つ以上の属性を含めることは可能ですが、多くの場合、リソースの消費量が増加し、利点を上回ります。

クラスターへの影響

クエリのすべての条件をカバーするインデックスを作成して理論上のパフォーマンスを最大限に高めようとしていますが、ほとんどのデータフィルタリングは複合インデックスの最初の 1～3 個の属性を通じて行われます。このしきい値を超える追加のフィールドは、意味のあるクエリの最適化ではなく、主にインデックスサイズに影響します。

ストレージと I/O オーバーヘッド: 多くの属性を持つ複合インデックスは、よりシンプルな代替手段よりもはるかに多くのストレージ領域を消費します。サイズは、インデックス付き属性の数と、インデックス付き値自体のサイズに直接比例します。

メモリフットプリント: 多くの属性を持つ複合インデックスの大きなストレージフットプリントは、対応するメモリフットプリントを作成し、ワーキングセットを大きくして、頻繁にアクセスされる他のデータをバッファプールから置き換えます。

書き込みオペレーション: 複数のインデックス付きフィールドに影響するドキュメントの変更ごとに、複合インデックスエントリ全体を更新し、書き込みオペレーションの完了に必要な作業を乗算する必要があります。

識別方法

まず、コレクション内のすべてのインデックスを確認して、3 つ以上の属性を持つ複合インデックスを特定します。

```
// 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 }
```

修正

3 つ以上の属性を持つ複合インデックスが使用されている場合は、それを利用するクエリを特定し、最適化の機会を探します。これらのインデックスを、3 つ以下の属性を含むより効率的なインデックスに置き換えることを検討してください。新しいインデックスを実装したら、3 つ以上の属性を含む古いインデックスを削除します。

> **Note**
>
> インデックスを削除する前に、常に利害関係者と調整し、パフォーマンスへの影響を検証してください。

複合インデックスを作成するときは、等価、ソート、範囲 (ESR) ルールに従います。

等価ソート範囲 (ESR) ルール: この順序は、最初に等価条件でデータセットをフィルタリングし、次に縮小されたセットにソートオペレーションを適用し、最後に可能な限り小さいサブセットで範囲スキャンを実行することで、インデックス効率を最大化します。

- フィールドを等号 (完全一致) として並べ替え、
- ソート (順序付け) 、
- 範囲 (>、 < 、$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
```

長時間実行されるクエリ

概要

長時間実行されるクエリは、クラスター全体のパフォーマンス問題にカスケードされる可能性があります。これらのクエリは、Amazon DocumentDB の Multi-Version Concurrency Control (MVCC) ガベージコレクションプロセスに干渉し、古いバージョニングされたドキュメントが蓄積され、クラスター全体のパフォーマンスが低下する可能性があります。Amazon DocumentDB は、ランナウェイクエリがリソースを無期限に消費することを制限する安全メカニズムとして、2 時間のサーバー側のタイムアウトを実装します。

長時間実行されるクエリとは:

- 長期間 (通常は > 30 分) 実行されるクエリ
- 数時間アクティブのままのオープンカーソル (カーソルがアクティブの場合、2 時間のタイムアウトは適用されません)

クラスターへの影響

長時間実行されるクエリはガベージコレクションプロセスを妨げる可能性があります

- 古いバージョン累積: ガベージコレクターは古いドキュメントバージョンを再利用できません
- コレクションとインデックスの肥大化: コレクションとインデックスのエントリは時間の経過とともに蓄積され、肥大化してストレージコストが増加する可能性があります。
- CPU とメモリのプレッシャー: 古いドキュメントバージョン、インデックスエントリ、トランザクション IDs。

長時間実行されるクエリ → GC をブロック → ストレージの成長 → CPU とメモリのプレッシャー → より長いクエリ

モニタリングと検出

1. 長時間実行されているクエリを検索するには、currentOp コマンドを使用します。

```
// 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. 30 分以上アクティブなカーソルを検索するには

```
// 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。CloudWatch によるガベージコレクターの進行状況のモニタリング

`LongestRunningGCProcess`— 最も長いアクティブなガベージコレクションプロセスの秒単位の期間。1 分以内に完了するプロセスを除き、1 分ごとに更新され、アクティブなオペレーションのみを追跡します。

`AvailableMVCCIds` - ゼロに達する前に使用可能な残りの書き込みオペレーションの数を示すカウンター。このカウンターがゼロに達すると、ID が再利用されてリサイクルされるまで、クラスターは読み取り専用モードになります。カウンターは書き込みオペレーションごとに減少し、ガベージコレクションが古い MVCC ID をリサイクルすると増加します。

> **Note**
>
> 低い MVCC IDs と長いガベージコレクション期間は、実行時間の長いクエリのみに起因するわけではありません。リソースに制約のあるインスタンスで書き込み負荷の高いワークロードを使用すると、MVCC ID の可用性が低下し、ガベージコレクションサイクルが長くなる可能性があります。

修復戦略

- アプリケーションにクエリタイムアウトを実装する
- カーソルを長時間存続させないでください。
- クエリを最適化してパフォーマンスを向上させます。
- 書き込みオペレーションのバッチ処理を優先する

低使用インデックスと冗長インデックス

概要

インデックスはワークロードの全体的なクエリパフォーマンスを向上させますが、コストがかかります。各インデックスはストレージと I/O リソースを消費し、書き込みオペレーションはコレクションのすべてのインデックスを更新する必要があります。データベースに未使用、使用率が低い、または冗長なインデックスが含まれている場合、クラスターにオーバーヘッドが発生し、パフォーマンスが低下します。複数の不要なインデックスは、パフォーマンスの問題、ストレージコストの増加、運用オーバーヘッドの増加を引き起こすアンチパターンを作成します。

最適なインデックス作成シナリオ

- 未使用のインデックス: 1 回限りのクエリ、または現在のクエリパターンによってアクセスされなくなった以前の製品反復用に作成されたインデックス。コレクションで未使用のインデックスを識別するには、indexStats を使用できます。詳細については、「the section called “インデックスの使用状況を分析し、未使用のインデックスを特定します。”」を参照してください。
- 冗長インデックス: 重複するキーパターンまたは同一のクエリパターンをカバーする複数のインデックス。例: 属性 A の単一キーインデックスは、複合インデックスが属性 A と B の両方をカバーすると冗長になります。複合インデックスは属性 A のクエリのみを処理できるためです。
- インデックス超過: 実際のクエリパターンやパフォーマンス要件を分析しずに、「念のため」インデックスを作成します。
- 低カーディナリティインデックス: クエリ最適化の利点を最小限に抑える、いくつかの異なる値 (ブール値フィールド、ステータスフラグなど) を持つフィールドのインデックス。

クラスターパフォーマンスへの影響

- ストレージと IO: 各インデックスはストレージスペースと IO リソースを消費し、全体的なコストに寄与します。インデックスが未使用または冗長である場合、クラスターの不要なコストを表します。
- 書き込みパフォーマンスの低下: 挿入、更新、削除オペレーションはインデックスを維持し、リソースを消費するオーバーヘッドを作成する必要があります。未使用または冗長なインデックスは、これらのオペレーションに不要なオーバーヘッドを追加します。
- メモリと CPU の圧力: インデックスはバッファプールメモリと競合し、ワーキングセットがインスタンスメモリに収まらず、最適ではない BufferCacheHitRatio が発生すると、頻繁にアクセスされるデータを削除する可能性があります。システムは、インデックス構造を維持する CPU リソースも消費します。

インデックス最適化の機会を特定するためのツール

- インデックスレビューツール:
- インデックスカーディナリティ検出ツール

ツールキットの収集:

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

1. インデックスレビューツール

インデックスレビューツールは、すべてのコレクションとインデックスを分析して、インデックスとその使用パターンのリストを提供し、Amazon DocumentDB クラスター上の冗長インデックスを識別します。包括的な分析のために、このツールをすべてのクラスターインスタンスで実行します。詳細については、「インデックスレビューツール」を参照してください。

使用:

```
cd performance/index-review/

python3 index-review.py --server-alias <server-alias> --uri <mongodb-uri>
```

出力ファイル

- `{server-alias}-collections.csv`: コレクションレベルの統計
- `{server-alias}-indexes.csv`: 詳細なインデックス使用状況メトリクス
- `{server-alias}-{timestamp}-index-review.json`: 詳細な分析のための未加工データ

> **Note**
>
> クラスター内の各インスタンスに対して独立したエイリアスを使用して を実行し、各インスタンスから統計を生成し、ツールを実行して累積分析を取得します。

例: `python3 index-review.py --files "{server-alias}-{timestamp}-index-review.json, {server-alias}-{timestamp}-index-review.json" --server-alias full-review`

2. インデックスカーディナリティ検出ツール

このツールは、クエリのパフォーマンスに非効率的な低カーディナリティのインデックスを識別します。

使用方法

```
cd performance/index-cardinality-detection/

python3 detect-cardinality.py --uri <mongodb-uri>
```

パラメータ

- `--threshold`: カーディナリティしきい値の割合 (デフォルト: 1%)
- `--sample-count`: サンプルするドキュメントの数 (デフォルト: 100,000)
- `--max-collections`: データベースごとにスキャンする最大コレクション数 (デフォルト: 100)

詳細については、https://aws.amazon.com/blogs/database/detect-and-fix-low-cardinality-indexes-in-amazon-documentdb/ を参照してください。

修復戦略

1. 未使用および冗長なインデックスの削除

```
// Example: Drop unused index
db.collection.dropIndex("unused_index_name")
```

> **Note**
>
> すべての利害関係者と話し合い、パフォーマンスをテストしない限り、インデックスを削除しないでください。

2。低カーディナリティインデックスの最適化

- フィルターで部分インデックスを使用する
- 単一の低カーディナリティインデックスを複合インデックスに変換する

大きな配列を持つマルチキーインデックス

概要

Amazon DocumentDB のマルチキーインデックスを使用すると、配列フィールドを効率的にクエリできます。このインデックス作成戦略は、配列データ内で検索するときにクエリレイテンシーを大幅に低減するため、複雑なデータ構造を持つアプリケーションにとって魅力的なオプションとなります。

ただし、コレクション内で大きな配列または複数の配列フィールドを使用する場合は、リソースの影響とパフォーマンス特性を理解して、十分な情報に基づいたアーキテクチャ上の意思決定を行うことが重要です。配列フィールドにインデックスを作成すると、Amazon DocumentDB は配列内の各要素に対して個別のインデックスエントリを生成します。

クラスターへの影響

- ストレージと IO オーバーヘッド: マルチキーインデックスは、ベーステーブルの複数回までストレージスペースを消費することがあります。サイズは、インデックス付き配列属性を持つコレクション内のドキュメント、配列内の要素の数、要素のサイズに直接比例します。
- メモリ使用量: ストレージフットプリントが比較的大きいため、これらのインデックスには対応するメモリフットプリントがあるため、作業セットが大きくなります。
- 書き込みオペレーション: マルチキーインデックスは、書き込みオペレーション中にオーバーヘッドを作成します。各配列要素は個別のインデックスエントリを生成し、書き込みオペレーションの完了に必要な作業を乗算します。

修復戦略

- 必要に応じてマルチキーインデックスを作成し、配列内のフィールド数を制限します。
- コレクション上のマルチキーインデックスの数を制限する
- データモデルの変更を検討する

パフォーマンス向上のヒント

このセクションでは、Amazon DocumentDB のパフォーマンス最適化戦略を 5 つ提供して、アプリケーションの効率とクエリ実行を向上させます。

1. 集約パイプラインの最初のステージとして $match を使用する

パフォーマンスを最大化するには、常に $match を集約パイプラインのフィルタリングの最初のステージとして配置します。Amazon DocumentDB は、$match がパイプラインを主導するときにインデックスを効果的に活用するため、データベースはデータを早期にフィルタリングし、処理オーバーヘッドを削減できます。

```
// Optimized approach
db.orders.aggregate([
  { $match: { status: "active", category: "electronics" } }, // Index utilization
  { $group: { _id: "$category", total: { $sum: "$price" } } },
  { $sort: { total: -1 } }
])
```

影響: 早期フィルタリングにより、後続のパイプラインステージで処理されるドキュメントの数が減少し、クエリの実行が速くなり、リソースの消費量が減少します。

2. 集約パイプラインで $project を使用してパイプラインデータサイズを最小化する

データサイズを最小限に抑え、パフォーマンスを向上させるために、集約パイプラインステージでは必須フィールドのみを実行します。必要なデータのみを含めるには、$project を戦略的に使用します。

```
// 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: ドキュメントが小さいほどメモリ使用量が減少し、パイプラインの処理効率が向上し、全体的なクエリパフォーマンスが向上します。

3. ドキュメント圧縮を有効にしてストレージコスト、I/O コストを削減し、クエリパフォーマンスを向上させる

クラスターパラメータグループからのドキュメント圧縮を有効にして、ストレージコスト、I/O コストを削減し、クエリパフォーマンスを向上させます。Amazon DocumentDB は、圧縮されたドキュメントをディスクと RAM に保存し、メモリフットプリントと I/O コストを削減します。

影響:

- より多くのドキュメントが使用可能なメモリに収まる
- ディスク読み取りを減らしてデータアクセスを高速化する
- ストレージコストの削減、I/O コスト、クエリパフォーマンスの向上

> **Note**
>
> Amazon DocumentDB は、バージョン 5.0 ではデフォルトで圧縮を有効にしません。5.0 クラスターでは、コレクションレベルまたはクラスターレベルで圧縮を有効にできます。Amazon DocumentDB の圧縮レビューユーティリティを使用して、コレクションの圧縮率を分析します。
> Amazon DocumentDB 8.0 では、圧縮はデフォルトで有効になっています。

4. インデックスを活用してクエリパフォーマンスを最適化する

クエリが常にインデックスを使用してパフォーマンスを最適化していることを確認します。Amazon DocumentDB には、さまざまなユースケースに合わせて複数のインデックスタイプが用意されています。

インデックス作成の原則:

- すべてのクエリは適切なインデックスを活用する必要があります
- Amazon DocumentDB は複数のインデックスタイプを提供します
- 複合インデックスは、1 つのインデックスでさまざまなクエリシェイプをサポートすることで、最大限の柔軟性を提供します。

- ソート操作とフィルタリング操作を一緒にサポートするようにインデックスを設計する

インデックスプレフィックスを理解する: 複合インデックスはインデックスプレフィックスを介して機能します - Amazon DocumentDB は、インデックスフィールドのleft-to-rightへのサブセットを使用できます。たとえば、インデックスは次の使用可能なプレフィックス`{ category: 1, price: -1, inStock: 1 }`を作成します。

- `{ category: 1 }` - カテゴリによるクエリフィルタリングのみをサポート
- `{ category: 1, price: -1 }` - カテゴリによるクエリフィルタリングと、料金によるソート/フィルタリングをサポート
- `{ category: 1, price: -1, inStock: 1 }` - 完全な複合クエリをサポート

料金、inStock、または inStock のみのクエリは、最初のフィールド (カテゴリ) で始まらないため、このインデックスを使用しません。

インデックスを使用していないクエリを特定する方法: explain() メソッドを使用してクエリ実行を分析し、インデックスを使用する代わりにコレクションスキャンを実行するクエリを特定します。

影響: インデックス使用率のないクエリでは、コレクションスキャンが行われ、インスタンスのメモリと CPU 負荷が増加し、クエリレイテンシーが増加します。

5. クエリパターンに基づいてデータモデルを最適化する

データモデルを、アプリケーションがデータをクエリおよび更新する方法に合わせます。データモデリングは、高性能な Amazon DocumentDB アプリケーションの基盤です。

最適化戦略:

パフォーマンスの埋め込み

- 関連データがユニットとして頻繁にアクセスされるときに一緒に保存する
- 常に一緒に取得されるドキュメントを埋め込む
- one-to-fewの関係に適しています

```
// 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" }
  ]
}
```

柔軟性のための参照

- 大規模なデータまたはアクセス頻度の低いデータにリファレンスを使用する
- 大規模なデータセットとの one-to-many関係に推奨
- ドキュメントの肥大化を防ぎ、更新パフォーマンスを向上させる

コレクション分割戦略

大きなドキュメント内の少数のフィールドのみが頻繁に更新される場合、またはアクセス頻度の低い大きなデータによってドキュメントが肥大化する場合は、コレクションを分割することを検討してください。

- 頻繁に更新されるフィールドを別の小さなコレクションに保持する
- 静的データまたはアクセス頻度の低いデータを別のコレクションに保存
- 必要に応じてリファレンスにリンクする

```
// 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 */]
}
```

パフォーマンスの向上: ドキュメントが小さいほど、更新が速くなり、メモリ使用量が減少し、キャッシュ効率が向上します。

影響: 非効率的なデータモデリングにより、最適ではないクエリ、ドキュメントサイズの増加、メモリ使用量の増加が発生し、アプリケーションのパフォーマンスが低下し、運用コストが増加します。

クエリプラン分析

explain plan によるクエリプラン分析は、Amazon DocumentDB クエリのパフォーマンスに関する重要なインサイトを提供します。executionStats を使用したクエリプランは、次のような主要なメトリクスを明らかにします。

- ステージごとに返されるドキュメント (nReturned)
- ステージ固有の実行時間 (executionTimeMillisEstimate)
- 計画生成期間 (planningTimeMillis)

クエリプランの出力を調べることで、デベロッパーは実行パターンを分析し、インデックス使用率を評価し、クエリパイプラインステージ全体で潜在的な最適化の機会を特定できます。

クエリプランを分析するには、次の形式で explain() コマンドを使用できます。

```
db.runCommand({explain: {query document}, verbosity: "executionStats"})
db.collection.find().explain("executionStats");
```

オペレーションの例を次に示します。

```
db.collection.find({ companyname: { '$eq': 'ANYCOMPANY' }, isDeleted: { '$eq':
 false } }).sort({"createdAt":1}).limit(2).explain("executionStats");
```

このオペレーションによる出力は、次のようになります。

```
{
  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 })
}
```

以下は、Amazon DocumentDB クエリ実行プランの詳細な分析であり、各コンポーネントとそのパフォーマンス特性を分類したものです。

全体的なタイミング

executionTimeMillis は、計画時間を含むクエリにかかる合計時間を表します。

planningTimeMillis は、クエリにかかる合計計画時間を表します。

実行ステージ

Amazon DocumentDB がクエリの実行に使用するstep-by-stepのプロセスについて説明し、さまざまなオペレーションでデータがどのように流れるかを示します。

```
"executionStages": {
    "stage": "[STAGE_NAME]",
    "nReturned": "[NUMBER_OF_DOCS]",
    "executionTimeMillisEstimate": "[TIME]",
    "inputStage": {
        // Nested stages
    }
}
```

クエリプランの一般的なステージ

以下は、クエリプランの一般的な実行ステージです。各ステージは executionTimeMillisEstimate (実行時間) および nReturned (ドキュメント数) メトリクスを返し、すべてのステージでクエリパフォーマンスを評価するのに役立ちます。

COLLSCAN (コレクションスキャン)

- コレクションドキュメント全体をドキュメントごとにスキャンします
- nReturned: コレクション内の一致するすべてのドキュメントを返します
- インデックスが利用できない場合に見られる、高価なオペレーションになる可能性があります

IXSCAN (インデックススキャン)

- インデックスを使用して一致するドキュメントを検索する
- nReturned: インデックスに基づいて一致するドキュメントのみ
- 効率的なオペレーション、COLLSCAN よりも優先

SORT

- 指定されたフィールドに基づいてドキュメントをソートします

- クエリのソート属性がインデックスの一部でない場合、このステージは明示的に表示されます。
- nReturned: 入力ドキュメントと同じ番号
- 大きな結果セットにはメモリを大量に消費します。最適化するには、ソートフィールド名をインデックスに追加します。

LIMIT_SKIP

- 返されたドキュメントとスキップされたドキュメントの数を制御します。
- nReturned: LIMIT 値によって制限されます
- ページ分割または LIMIT オペレーションに使用されます

サブスキャン

- ネストされたクエリオペレーションを実行します
- nReturned: サブクエリの結果によって異なります
- 複数のステージを持つ複雑なクエリで使用されます

executionTimeMillisEstimate が高いステージは最適化の候補として適しています。

> **Note**
>
> executionStats パラメータは現在、更新コマンドと削除コマンドをサポートしていません。

Explain Plan Execution docsExamined について

を使用してクエリを実行すると`explain("executionStats")`、Amazon DocumentDB は、クエリ結果を生成するためにスキャンされたドキュメントの数を理解するのに役立つ検査メトリクスを提供します。これらのメトリクスは、非効率的なクエリプランを特定し、パフォーマンスを最適化するのに役立ちます。

> **Note**
>
> Amazon DocumentDB 5.0.1+ および 8.0.0+ でのみ使用できます。

フィールド

フィールド	説明	レベル
totalDocsExamined	すべての実行ステージで検査されたドキュメントの合計数。	最上位の executionStats
docsExamined	特定の実行ステージで検査されたドキュメントの数。	ステージレベル

`docsExamined` フィールドは次のステージに表示されます。

ステージ	説明
COLSCAN	コレクションスキャン。コレクション内のすべてのドキュメントが検査されます。
IXSCAN	インデックススキャン。インデックス条件に一致するドキュメントのみが検査されます。
FETCH	オプティマイザが IXSCAN から別のステージでドキュメントを取得すると、FETCH ステージは docsExamined を報告します。インデックススキャンプランでは、子 IXSCAN ステージは docsExamined を報告しません。$lookup プランでは、FETCH ステージとその子ステージの両方が docsExamined を報告する場合があります。

> **Note**
>
> `docsExamined` クエリはドキュメントにアクセスせずにインデックスから完全に満たされるため、 は IXONLYSCAN ステージに表示されません。

Explain Plan executionStats 出力で docsExamined を表示する

次の例では、500,000 のドキュメントを含むコレクションを使用し、さまざまなクエリ実行ステージで がどのように`docsExamined`変化するかを示します。

サンプルデータを作成します。

```
for (let i = 0; i < 500000; i++) {
    db.coll.insertOne({ number: i, arr: [i, [i+1]], value: "test", bool: i % 2 === 
 0 });
}
```

コレクションスキャン (COLLSCAN)

インデックスがない場合、Amazon DocumentDB はフルコレクションスキャンを実行します。

```
db.coll.find({ "number": { "$lt": 500 } }).explain("executionStats").executionStats
```

出力:

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

500,000 件のドキュメントすべてが 500 件の結果を返すように調査されました。数値フィールドにインデックスを作成すると、これが向上します。

インデックススキャン (IXSCAN)

```
db.coll.createIndex({ number: 1 });
db.coll.find({ "number": { "$lt": 5000 } }).explain("executionStats").executionStats
```

出力:

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

インデックスでは、500,000 件のドキュメントのうち 5,000 件のみが調査および取得され、返された数と一致しました。インデックス条件は、クエリに一致しない 495,000 件のドキュメントを除外しました。

残差フィルターを使用したインデックススキャン

```
db.coll.find({ "number": { "$lt": 5000 }, "arr": { "$gt":
 4000 } }).explain("executionStats").executionStats
```

出力:

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

インデックス条件は 500,000 件中 5,000 件のドキュメントに一致し、 の残差フィルターは結果を 999 件にarr減らしました。docsExamined 値 5,000 は、インデックス条件の後、残差フィルターが適用される前に調査されたすべてのドキュメントを反映します。

IXSCAN で取得する

```
db.coll.find({ "$or": [{ "number": { "$lt": 100000 } }, { "number": { "$gt":
 400000 } }] }).explain("executionStats").executionStats
```

出力:

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

Amazon DocumentDB オプティマイザがフェッチステージを使用してドキュメントを取得すると、FETCH ステージは を報告しますdocsExamined。子 IXSCAN ステージは、ドキュメントに直接アクセスせずにインデックスキーのみをスキャンdocsExaminedするため、レポートされません。

集計ルックアップを使用した FETCH

```
db.coll.explain("executionStats").aggregate([
    { $match: { "number": { "$lt": 5 } } },
    { $lookup: { from: "coll", pipeline: [{ $match: { "number": { "$lt": 3 } } }], as:
 "sub" } }
]).executionStats
```

出力:

```
{
```

```
    "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 は、数値 < 5 に一致する 5 つのドキュメントを調べました。内部 IXSCAN は、数値 < 3 に一致する 3 つのドキュメントを調べ、FETCH ステージは 1 つの集計結果を調べました。9 `totalDocsExamined`の は、すべてのステージの合計です (5 + 3 + 1)。

インデックスのみのスキャン (対象クエリ)

インデックスのみのスキャン (対象クエリとも呼ばれます) は、Amazon DocumentDB が基になるドキュメントを読み取らずに、インデックスに保存されているフィールドからクエリに完全に応答する最適化です。クエリに必要な値はインデックスに既に存在するため、Amazon DocumentDB は一致する各ドキュメントの取得をスキップできるため、ランダム I/O を回避し、ドキュメント全体を読み取るためのデコードコストを回避できます。クエリプランでは、インデックスのみのスキャンは、 `IXSCAN` (インデックススキャン) または `COLLSCAN` (コレクションスキャン) の代わりに`IXONLYSCAN`ステージとして表示されます。これにより、I/O が削減され、サポートされているクエリのクエリレイテンシーが大幅に短縮されます。

クエリは、1 つのインデックスに存在する少数のフィールドのみ、特にドキュメント自体が大きい場合、インデックスのみのスキャンに適しています。一般的な例は、より広範な複合インデックスからいくつかのフィールドを読み取るダッシュボード、分析ロールアップ、ルックアップです。

クエリの結果は、インデックスのみのスキャンを使用するかどうかにかかわらず同じです。クエリプランと実行される I/O の量のみが変わります。

> **Note**
>
> インデックスのみのスキャンのサポートは、Amazon DocumentDB エンジンのバージョンによって異なります。
>
> - 複数の非_idフィールドを含む対象find射影には、エンジンバージョン 8.0.2 以降が必要です。以前のバージョンでは、対象find射影には最大 1 つの非 _idフィールドを含めることができます。2 つ以上のフィールドを持つ射影は、通常のインデックススキャンまたはコレクションスキャンにフォールバックします。
> - 対象となる などの集約パイプラインのインデックスのみのスキャンには$group、エンジンバージョン 8.0.2 以降が必要です。

Amazon DocumentDB がインデックスのみのスキャンを使用する場合

Amazon DocumentDB は、1 つのインデックスにオペレーションが読み取るすべてのフィールドが含まれている場合、次のオペレーションのインデックスのみのスキャンを考慮します。

対象**find**射影

find クエリは、その射影が 1 つ以上の最上位フィールドの単純な包含であり、すべての射影フィールドが 1 つのインデックスに存在する場合に認定されます。たとえば、 で複合インデックスがあるとすると{ customerId: 1, status: 1, total: 1 }、次のクエリがカバーされます。射影された 3 つのフィールドはすべてインデックスに存在し、明示的に除外_idされます。(複数のフィールド_id以外の射影をカバーするには、エンジンバージョン 8.0.2 以降が必要です) 。

```
db.orders.createIndex({ customerId: 1, status: 1, total: 1 })

db.orders.find(
  { customerId: 42 },
  { _id: 0, customerId: 1, status: 1, total: 1 }
)
```

落札者はステージを使用しIXONLYSCAN、一致するドキュメントをストレージから取得しません。

```
{
```

```
  "stage": "IXONLYSCAN",
  "indexName": "customerId_1_status_1_total_1",
  "direction": "forward",
  "indexCond": { "$and": [ { "customerId": { "$eq": 42 } } ] }
}
```

> **Note**
>
> 対象findクエリでは、結果のフィールドは、射影にリストされた順序やドキュメントに保存された順序ではなく、インデックスの順序 (フィールドがインデックス定義に表示される順序) で返されます。これは、対象クエリの MongoDB の動作と一致します。コレクションスキャンによって応答されたクエリは、代わりにストアドドキュメントの順序でフィールドを返します。

対象$group集計

$group ステージは通常、少数のフィールド (グループキーとそのアキュムレータへの入力) のみを読み取りますが、多くの場合、先頭の なしで実行されます$match。カバーインデックスがない場合、Amazon DocumentDB はコレクション内のすべてのドキュメントを読み取り、デコードして、これらの少数のフィールドをグループ化にフィードします。

対象$groupステージを含む集約パイプラインのインデックスのみのスキャンは、エンジンバージョン 8.0.2 以降で使用できます。

単一のインデックスにパイプラインプレフィックスが読み取るすべてのフィールドが含まれている場合、Amazon DocumentDB はグループの下にインデックスのみのスキャンを配置し、インデックスから直接それらのフィールドを読み取ることができます。ステータス別に注文をグループ化し、合計を合計し、 に複合インデックスを持つパイプラインを考えてみましょう{ status: 1, total: 1 }。

```
db.orders.createIndex({ status: 1, total: 1 })

db.orders.aggregate([
  { $group: { _id: "$status", revenue: { $sum: "$total" } } }
])
```

パイプラインは statusと のみを読み取りtotal、どちらもインデックスにあるため、グループはコレクションスキャンIXONLYSCANの代わりに からフィードされます。

```
// Without a covering index
$group  <-  COLLSCAN     (reads and decodes every full document)

// With a covering index on { status, total }
$group  <-  IXONLYSCAN over { status, total }   (reads only the referenced fields)
```

$group がサポートされている$limit前の先頭$matchの $sort、、$skip、または 。ただし、パイプラインをカバーするには、これらのステージが参照するすべてのフィールド ($matchフィルターフィールドや$sortキーなど) も同じインデックスに存在する必要があります。インデックスに参照フィールドがない場合、Amazon DocumentDB はコレクションまたはインデックススキャンにフォールバックします。

> **Note**
>
> find 射影とは異なり、 $groupには暗黙的にドキュメント は含まれません_id。_id の $groupはドキュメント識別子ではなくグループキーであるため、パイプラインをカバーする_idためにインデックスにドキュメントを含める必要はありません。

対象クエリの要件

Amazon DocumentDB がインデックスのみのスキャンを使用するには、次のすべてを保持する必要があります。

- 単一のインデックスには、クエリが読み取るすべてのフィールドが含まれます。これにはfind、の射影フィールド、グループキー、アキュムレータ入力、および対象$groupパイプラインのフィルターまたはソートフィールドが含まれます。インデックスに 1 つのフィールドがない場合、クエリをカバーできません。
- 射影には、最上位フィールド (の場合) が含まれます。 find除外射影、計算フィールド、 や などの射影演算子$slice$elemMatchは対象外です。
- **_id** フィールドは (の場合) が考慮されます。 find_id はデフォルトで含まれるため、 などの一般的な複合インデックス{ a: 1, b: 1 }は をカバーしませんfind({}, { a: 1, b:

1 })。暗黙的な _idはインデックスに含まれません。プロジェクション ({ _id: 0, a: 1, b: 1 }) _id から除外するか、インデックス_idに追加します。

- フィルターはインデックスによって完全に満たされます。Amazon DocumentDB がクエリ述語の一部をインデックスバインドに変換できない場合、その部分は残差フィルターとして保持され、ドキュメントを読み取りて評価し、スキャンをインデックスのみのスキャンに変換しません。$mod、$ne、、 などの演算子は$nin、残余フィルターを$exists残します。
- 射影フィールドはマルチキー (配列) 列ではありません。「制限事項」を参照してください。

制限事項

次のケースでは、インデックスのみのスキャンを使用できず、通常のインデックススキャンまたはコレクションスキャンにフォールバックすることはできません。これは予想される動作であり、結果の正確性には影響しません。

- マルチキー (配列) フィールド。フィールドが任意のドキュメントで配列値を保持したことがある場合、そのインデックス列はマルチキーです。マルチキーインデックスは配列要素ごとに 1 つのエントリを保存するため、元の配列をインデックスから再構築することはできません。このようなフィールドを読み取るクエリは対象になりません。
- ドットまたはネストされたパス。インデックスにそのパスが含まれている場合でも、 などのネストされたパスの射影またはフィールド参照"address.city"は対象外です。
- 除外、計算、または演算子の射影。{ b: 0 }、、 などの射影には完全なドキュメント{ a: { $slice: 5 } }が必要であり{ total: { $add: ["$price", "$tax"] } }、対象にはなりません。
- 集計でのドキュメント全体の参照。ドキュメント全体を参照$groupする (など{ $push: "$$ROOT" }) は、インデックスがドキュメント全体を再構築できないため、カバーできません。
- より前のステージの再形成**$group**。がパイプラインの対象$groupから除外する前に表示されるドキュメント ($project、$unwind、、 など$lookup) $addFieldsを再構成するステージ。
- 文字列述語の照合不一致。クエリの照合順序がインデックスの照合順序と異なる場合、文字列述語をインデックスで処理して残余フィルターを残すことはできません。これにより、ドキュメントが強制的に取得されます。数値および型ベースの述語は影響を受けません。

インデックスのみのスキャンの確認

explain("executionStats") 出力を使用して、クエリがインデックスのみのスキャンによって処理され、ドキュメントに影響を与えていないことを確認します。次の 2 つのものを探します。

- 獲得プランは IXONLYSCANステージを使用します。
- そのステージのdocsExamined値は です 0 (は totalDocsExaminedです0)。つまり、クエリはドキュメントを読み取らずにインデックスから完全に回答されました。

```
db.orders.find(
  { customerId: 42 },
  { _id: 0, customerId: 1, status: 1, total: 1 }
).explain("executionStats").executionStats
```

```
{
    "executionSuccess": true,
    "nReturned": "2",
    "executionTimeMillis": "0.714",
    "totalDocsExamined": "0",
    "executionStages": {
        "stage": "IXONLYSCAN",
        "nReturned": "2",
        "indexName": "customerId_1_status_1_total_1",
        "direction": "forward",
        "docsExamined": "0"
    }
}
```

> **Note**
>
> IXONLYSCAN ステージのゼロ以外のdocsExamined値は、インデックスにデータが含まれていることを意味しますが、Amazon DocumentDB は引き続きストレージをチェックして、各ドキュメントバージョンが表示されていることを確認します。これは、コレクションに古いドキュメントバージョンのガベージコレクションがまだ再利用されていない最近の書き込みがある場合に発生する可能性があります。ガベージコレクションがこれらのドキュメントを処理すると、 は にdocsExaminedドロップします0。その時点で、クエリは対象スキャンを最大限に活用します。ガベージコレクションの詳細については、「」を参照してくださいAmazon DocumentDB のガベージコレクション。docsExamined メトリクスの詳細については、「クエリプラン分析」を参照してください。

ベストプラクティス

- 最も頻度の高いクエリプロジェクトまたはグループを含む複合インデックスを設計して、それらのクエリをカバーできるようにします。
- _id がカバーインデックスの一部でない限り、find射影 ({ _id: 0, ... }) _idから除外します。
- マルチキー列はインデックスのみのパスを失格にするので、クエリをカバーしようとするときは、配列 (マルチキー) フィールドのインデックス作成は避けてください。
- ガベージコレクションが書き込みワークロードに対応できるようにして、対象スキャンがドキュメントへのアクセスをスキップし、 にdocsExaminedとどまるようにします0。詳細については、「Amazon DocumentDB のガベージコレクション」を参照してください。

キャッシュフィルターの計画

インデックスフィルターとも呼ばれるプランキャッシュフィルターは、クエリプランナーが特定のクエリシェイプについて考慮できるインデックスのセットを制限します。クエリがフィルターを持つシェイプと一致する場合、プランナーはコレクション上のすべてのインデックスを評価するのではなく、そのフィルター内の という名前のインデックスからのみプランを選択します。

クエリシェイプは、クエリ述語、ソート仕様、コレクション名前空間の組み合わせで、述語値が正規化されます。値のみが異なる 2 つのクエリはシェイプを共有するため、1 つのフィルターですべてのクエリをカバーします。の出力ではplanCacheListFilters、正規化された値は として表示されます"@"。

プランキャッシュフィルターを使用すると、プランを固定できるため、データ分散やインデックスの変更によるインデックスの選択が遅くなるのを回避できます。フィルターには次のプロパティがあります。

- アプリケーションの変更は必要ありません。フィルタはデータベースコマンドで設定され、サーバーに適用されるため、コードのデプロイなしでリグレッションを軽減できます。これは、すべての通話サイトに追加hintする必要がある との主な違いです。
- スコープは 1 つのクエリシェイプです。シェイプは、完全なクエリではなく、コレクション、述語、ソートによって定義されるため、フィルターはそのシェイプに一致するクエリのみを制限します。同じコレクションに対する他のクエリは、引き続き通常どおり計画されます。
- 変更は耐久性があります。フィルターは、インスタンスの再起動とエンジンのパッチ適用全体で保持されます。コレクションを削除すると、そのフィルターが削除されます。

- 変更は元に戻されます。フィルターをクリアするとシェイプが通常のコストベースの計画に戻るため、フィルターは根本的な原因を調査しながらワークロードを安定させる低リスクの方法です。

トピック

- サポートされているコマンド
- サポートされているクエリシェイプ
- フィルターの設定、一覧表示、クリア
- フィルターが適用されていることを確認する
- 例
- ヒントとインデックスの変更を含む動作

サポートされているコマンド

プランキャッシュフィルターには、プランナーバージョン 2.0 以降が必要です。フィルターを適用できるコマンドは、次の表に示すように、プランナーバージョンと Amazon DocumentDB マイナーバージョンによって異なります。

プランキャッシュフィルターのコマンドサポート

コマンド	プランナーの最小バージョン	最小マイナーバージョン
`find`	2.0	5.0.0、8.0.0
`count`	2.0	5.0.0、8.0.0
`update`	2.0	5.0.2、8.0.2
`delete`	2.0	5.0.2、8.0.2
`findAndModify`	2.0	5.0.2、8.0.2
`distinct`	3.0	8.0.0
`aggregate`	3.0	8.0.2

> **Note**
>
> プランナーバージョン 3.0 は Amazon DocumentDB 8.0 でのみ使用できます。コマンドdistinctと aggregate コマンドにはプランナーバージョン 3.0 が必要なため、これら 2 つのコマンドの計画キャッシュフィルターは Amazon DocumentDB 8.0 でのみ使用できます。プランナーバージョン 2.0 をサポートする Amazon DocumentDB 5.0 では、フィルターは find、、count、delete、updateおよび findAndModify コマンドに適用されます。

aggregate コマンドの場合、シェイプはパイプラインの前面から取得されます。先頭はフィルター$matchを提供し、先頭はソートを提供する$sort直後に を提供します。 $skip と はシェイプに影響を与え$limitず、それ以降のステージも影響を受けません。

Amazon DocumentDB は、パイプラインを計画する前に最適化し、計画キャッシュフィルターは最適化されたパイプラインと照合されます。したがって、プランナーに表示されるフィルターとソートは、作成したステージとは異なる可能性があります。たとえば、パイプラインの前面にある 2 つの隣接する$matchステージが 1 つの$and述語に結合されます。

```
db.orders.aggregate([
  { $match: { status: "SHIPPED" } },
  { $match: { region: "us-east-1" } },
  { $group: { _id: "$customerId", total: { $sum: "$amount" } } }
])
```

プランナーは $matchに 1 つ表示されるため{$and: [{status: ...}, {region: ...}]}、フィルターはその結合述語に設定する必要があります。

```
db.runCommand({planCacheSetFilter: "orders",
query: { $and: [ { status: "SHIPPED" }, { region: "us-east-1" } ] },
indexes: [ "status_1_region_1" ]})
```

$lookup ステージまたは $graphLookupステージの外部コレクションに設定されたフィルターは、その外部コレクションのスキャンに使用されるインデックスを駆動します。

サポートされているクエリシェイプ

Amazon DocumentDB 5.0.2 および 8.0.2 以降では、プランキャッシュフィルターを持つクエリシェイプに次の演算子を表示できます。

- $text。 検索文字列は正規化されるため、検索する用語だけが異なるクエリは 1 つのシェイプを共有します。
- 「$near」および「$nearSphere」。どちらの演算子も同じ形状に正規化されます。座標、$minDistance、 $maxDistanceはシェイプの一部ではありません。
- 「$geoWithin」および「$geoIntersects」。GeoJSON ジオメトリタイプはシェイプの一部であるため、Polygonクエリと MultiPolygonクエリは異なるシェイプです。座標はシェイプの一部ではありません。
- $regex、 フィールドで直接使用する場合。パターンとオプションは正規化されるため、同じフィールドのすべての正規表現は 1 つのシェイプを共有します。

$jsonSchema、、$exprまたは を使用するクエリシェイプのフィルターの設定はエラーコード 303 で失敗します$sampleRate。また、次のシェイプでも失敗します。これらのいずれかを使用するクエリは、フィルターなしで計画されます。

- 、$in、$ninまたは 内に$regexネストされた $all。正規表現とリテラル値を混在させる配列はインデックスに対して評価できないため、そのシェイプのフィルターを適用することはできません。
- のソート仕様{$natural: 1}。$natural ソートはコレクションスキャンをリクエストしますが、プランナーをインデックスに制限することはできません。

フィルターの設定、一覧表示、クリア

計画キャッシュフィルターを設定するには、 planCacheSetFilter コマンドを使用します。query フィールドと sortフィールドは一緒にクエリシェイプを指定し、プランナーがそのシェイプについて考慮できるインデックス名をindexes一覧表示します。

```
db.runCommand({
planCacheSetFilter: <collection>,
query: <query>,
sort: <sort>, // optional
indexes: [ <index1>, <index2>, ...],
comment: <any> // optional
})
```

たとえば、次のコマンドは、プランナーをソート{a: {$eq: "@"}, b: {$eq: "@"}}なしでシェイプのa_1インデックスに制限します。

```
db.runCommand({planCacheSetFilter: "foo",
query: { a: 1, b: 1 },
sort: {},
indexes: [ "a_1" ]})
```

既にあるシェイプのフィルターを設定すると、既存のフィルターが置き換えられます。次の 2 つのコマンドを実行した後、シェイプのフィルター{a: {$eq: "@"}, b: {$eq: "@"}}は になります["b_1"]。

```
db.runCommand({planCacheSetFilter: "foo",
query: { a: 1, b: 1 },
sort: {},
indexes: [ "a_1" ]})

db.runCommand({planCacheSetFilter: "foo",
query: { a: 6, b: 10 },
sort: {},
indexes: [ "b_1" ]})
```

コレクション上のすべてのフィルターを一覧表示するには、 planCacheListFilters コマンドを使用します。

```
db.runCommand({planCacheListFilters: <collection>})
```

出力には、保存された各シェイプとその正規化された値と許可されたインデックスリストが表示されます。

```
{
"filters" : [
{
"query" : { "a" : { "$eq" : "@" } },
"sort" : { },
"indexes" : [ "a_1" ]
},
{
"query" : { "a" : { "$gt" : "@" } },
"sort" : { "a" : 1 },
"indexes" : [ "a_1_b_1" ]
}
],
"ok" : 1
```

```
}
```

演算子は保持され、値のみが に置き換えられます"@"。などの暗黙的な等価性{a: 1}は、明示的な形式でリストされ{"a": {"$eq": "@"}}、$in配列の要素は全体として として置き換えられます{"a": {"$in": "@"}}。ソートのないフィルターセットは、空のsortドキュメントとともに一覧表示されます。

フィルターを削除するには、 planCacheClearFilters コマンドを使用します。

```
db.runCommand({
planCacheClearFilters: <collection>,
query: <query pattern>, // optional
sort: <sort specification>, // optional
comment: <any> // optional
})
```

コレクション上のすべてのフィルターをクリアするには、 queryと の両方を省略しますsort。

```
db.runCommand({planCacheClearFilters: "foo"})
```

query と の両方を指定すると、その正確なソートを持つフィルターのみがsortクリアされます。の値はquery正規化されないため、代表的な値はすべて機能します。ソートなしで設定されたフィルターのみをクリアするには、空のsortドキュメントを渡します。

```
db.runCommand({planCacheClearFilters: "foo", query: {a: 1}, sort: {a: 1}})

db.runCommand({planCacheClearFilters: "foo", query: {a: 1}, sort: {}})
```

フィルターが適用されていることを確認する

explain 出力には、プランキャッシュフィルターのステータスを報告する 2 つのフィールドが含まれます。

- indexFilterSet は、クエリの形状のコレクションにフィルターが存在するtrue場合です。
- indexFilterApplied は、そのフィルターのリストからインデックスを使用してクエリが計画されたtrue場合にのみ発生します。

Amazon DocumentDB 5.0.2 および 8.0.2 以降、これらのフィールドは、プランキャッシュフィルターをサポートするすべてのコマンドと、それらのコマンドのプロファイラー出力でレポート

されます。以前のマイナーバージョンでは、これらは find コマンドに対してのみ報告されました。explain 出力の読み取りの詳細については、「」を参照してくださいクエリプラン分析。

フィルターは、適用せずにクエリシェイプを一致させることができます。フィルターのリスト内のインデックスのいずれもクエリを処理できない場合、プランナーはフィルターなしで再計画し、コストに関するインデックスを選択します。この場合、 indexFilterSetは trueで、indexFilterAppliedは ですfalse。これは、名前付きインデックスがクエリのフィールドをカバーしない場合、名前付きインデックスが存在しない場合、または$regexシェイプがアンカーされていないパターンや大文字と小文字を区別しないパターンなどのインデックス境界を生成できない場合に発生します。

例

次の例では、 ordersコレクションと、これらのインデックスを持つ $lookup customersコレクションを使用しています。

```
db.orders.createIndex({ status: 1 })                    // status_1
db.orders.createIndex({ status: 1, orderDate: 1 })      // status_1_orderDate_1
db.orders.createIndex({ customerId: 1 })                // customerId_1
db.customers.createIndex({ customerId: 1 })             // customerId_1
```

例: **find**クエリを複合インデックスに固定する

でフィルタリングstatusし、 でソートするクエリを考えてみましょうorderDate。

```
db.orders.find({ status: "SHIPPED" }).sort({ orderDate: 1 })
```

複合インデックスは述語とソートの両方status_1_orderDate_1を満たすため、個別のソートは必要ありません。プランナーが代わりに を選択した場合status_1、クエリは結果をソートする必要があります。これによりステージが追加され、一致する注文の数が増えるSORTにつれてコストが高くなります。プランナーをこのシェイプの複合インデックスに制限するには:

```
db.runCommand({planCacheSetFilter: "orders",
query: { status: "SHIPPED" },
sort: { orderDate: 1 },
indexes: [ "status_1_orderDate_1" ]})
```

値はシェイプから正規化されるため、この 1 つのフィルターは { status: "PENDING" }、{ status: "CANCELLED" }、および同じソートstatusで の他のすべての値も

カバーします。同じ述語を別のソートでカバーしたり、別のシェイプであるソートなしでカバーしたりすることはありません。でフィルターが有効になったことを確認しますexplain。

```
db.orders.find({ status: "SHIPPED" }).sort({ orderDate: 1 }).explain()
```

出力は indexFilterSet: trueと を報告しindexFilterApplied: true、IXSCANステージ名は ですstatus_1_orderDate_1。

例: を選択的インデックスupdateに固定する

更新は、書き込む前に変更するドキュメントを見つける必要があり、そのルックアップは読み取りと同じ方法でインデックスを使用します。述語を複数のインデックスで提供できる場合、プランナーが選択するインデックスによって、更新が検査するドキュメントの数が決まります。

```
db.orders.updateMany(
  { customerId: 4815, status: "PENDING" },
  { $set: { status: "CANCELLED" } }
)
```

customerId_1 と の両方がこの述語を提供status_1できます。更新を に固定するにはcustomerId_1:

```
db.runCommand({planCacheSetFilter: "orders",
query: { customerId: 4815, status: "PENDING" },
indexes: [ "customerId_1" ]})
```

書き込みコマンドのフィルタには、Amazon DocumentDB 5.0.2 または 8.0.2 とプランナーバージョン 2.0 以降が必要です。更新explainの を使用して検証します。

```
db.runCommand({explain: {update: "orders",
updates: [{ q: { customerId: 4815, status: "PENDING" },
             u: { $set: { status: "CANCELLED" } },
             multi: true }]})
```

成功プランは、 IXSCAN上の よりも UPDATE ステージですcustomerId_1。シェイプには書き込みのタイプが含まれていないため、このクエリシェイプを使用する deleteおよび findAndModify コマンドにも同じフィルターが適用されます。

例: **aggregate**パイプラインと**$lookup**外部コレクションを固定する

集計の場合、シェイプは先頭ステージと$sort直後の$matchステージから取得されます。$group や など、それを超えるステージ$projectはシェイプの一部ではありません。

```
db.orders.aggregate([
  { $match: { status: "SHIPPED" } },
  { $group: { _id: "$customerId", total: { $sum: "$amount" } } }
])
```

$match 述語に設定されたフィルターは、 の読み取りに使用されるインデックスを駆動しますorders。

```
db.runCommand({planCacheSetFilter: "orders",
query: { status: "SHIPPED" },
indexes: [ "status_1" ]})
```

フィルターは、 $lookupまたは $graphLookupステージが結合するコレクションにも適用されます。フィルターは、パイプラインが実行されるコレクションではなく、外部コレクションに設定されます。次のパイプラインでは、 ordersは外部コレクションで、 は外部コレクションcustomersです。

```
db.orders.aggregate([
  { $lookup: { from: "customers",
               localField: "customerId",
               foreignField: "customerId",
               as: "customer" } }
])
```

結合は外部ドキュメントごとに 1 customers回プローブされるため、そこで選択されたインデックスは繰り返し適用され、コストはパイプライン全体に乗算されます。そのスキャンを特定のインデックスに固定するには、結合キーの等価シェイプcustomersのフィルターを に設定します。

```
db.runCommand({planCacheSetFilter: "customers",
query: { customerId: 1 },
indexes: [ "customerId_1" ]})
```

値はシェイプから正規化されるため、1上記の値はプレースホルダーであり、どの値でも同じフィルターが生成されます。外部コレクションフィルターが適用されると、最上位の

indexFilterSetフィールドと indexFilterAppliedフィールドは を報告しtrue、選択したフィルターのインデックスは、獲得プランの外部側に表示されます。のフィルターには、Amazon DocumentDB 8.0.2 とプランナーバージョン 3.0 aggregateが必要です。

ヒントとインデックスの変更を含む動作

- プランキャッシュフィルターは、 よりも優先されますhint。Amazon DocumentDB は最初にフィルターを適用し、次にフィルターの許可されたインデックスリスト内にヒントを適用します。ヒントがフィルターが許可するインデックスに名前を付ける場合、そのインデックスが使用されます。ヒントがフィルターで許可されていないインデックスに名前を付けると、ヒントは無視され、プランナーは許可されたインデックスの中からコストに基づいて選択します。どちらのヒントフォームも、インデックスに名前を付けるかキーパターンを付けるかにかかわらず、同じように動作します。

  たとえば、コレクションにインデックス a_1、b_1、および fooがありa_1_b_1、インデックスリスト {query: {a: {$eq: "@"}, b: {$eq: "@"}}, sort: {a: 1}} を持つシェイプにフィルターを設定した場合["a_1", "a_1_b_1"]、 を実行すると db.foo.find({ a: 10, b: 20 }).sort({a: 1}).hint({ a: 1 })が使用されます。これはa_1、ヒントで名前が付けられ、フィルターで許可されているためです。
- インデックスを削除しても、フィルターは変更されません。フィルターはインデックス名を保持します。そのインデックスが欠落している間、計画ではスキップされます。フィルターのリスト内の別のインデックスがクエリを処理できる場合でも、フィルターは引き続き適用されます。名前付きインデックスが利用できない場合にのみ、 がindexFilterAppliedレポートしますfalse。同じ名前のインデックスを再作成すると、再び対象になります。
- コレクションを削除すると、そのコレクションのフィルターがすべて削除されます。

Amazon DocumentDB のベストプラクティス

Amazon DocumentDB(MongoDB との互換性あり)を使用するためのベストプラクティスを説明します。新しいベストプラクティスが確認されると、このセクションは更新されます。

トピック

- 基本運用ガイドライン
- インスタンスのサイズ指定
- インデックスの使用
- セキュリティのベストプラクティス

- コスト最適化
- メトリクスを使用したパフォーマンスの問題の特定
- TTL および時系列ワークロード
- 移行
- クラスターパラメータグループの使用
- 集約パイプラインクエリ
- batchInsert および batchUpdate

基本運用ガイドライン

Amazon DocumentDB の使用の際に順守すべき基本的な運用ガイドラインを次に示します。Amazon DocumentDB のサービスレベルアグリーメント (SLA) では、次のガイドラインに順守する必要があります。

- 2 つ以上の Amazon DocumentDB インスタンスで構成されるクラスターを、2 つの AWS アベイラビリティーゾーンにデプロイします。本番ワークロードの場合、3 つ以上の Amazon DocumentDB インスタンスで構成されるクラスターを 3 つのアベイラビリティーゾーンにデプロイします。
- 指定されたサービスの制限内でサービスを使用します。詳細については、「Amazon DocumentDB クォータ」を参照してください。
- メモリ、CPU、接続、およびストレージの使用状況をモニタリングします。Amazon CloudWatch では、使用パターンが変更されたり、デプロイの最大容量に近づいたりすると、通知するように設定できます。この通知は、システムのパフォーマンスと可用性を維持するために役立ちます。
- 最大ストレージ容量に近づいたら、インスタンスをスケールアップする。アプリケーションからの予期しない需要増加に対応できる十分なコンピューティングリソース (RAM、CPU など) を使用してインスタンスをプロビジョニングする必要があります。
- 復旧ポイントの目標に合わせて、バックアップ保持期間を設定します。
- クラスターのフェイルオーバーをテストすることで、そのプロセスでユースケースにかかる時間を把握します。詳細については、「Amazon DocumentDB フェイルオーバー」を参照してください。
- Amazon DocumentDB クラスターをクラスターエンドポイント (「Amazon DocumentDB エンドポイント」を参照) にレプリカセットモード (「レプリカセットとしての Amazon DocumentDB に接続する」を参照) で接続し、アプリケーションへのフェイルオーバーによる影響を最小限に抑えます。

- アプリケーションの読み込み整合性要件に合わせて読み取り性能を最大化する、ドライバー読み込み設定を選択します。`secondaryPreferred` 読み込み設定を使用すると、レプリカの読み取りを有効にし、プライマリインスタンスを解放してより多くの作業を行うことができます。詳細については、「読み込み設定のオプション」を参照してください。
- ネットワークエラーやデータベースエラーが発生した場合にも回復できるようにアプリケーションを設計します。ドライバーのエラーメカニズムを使用して、一時エラーと永続エラーを区別します。適切な場合には、エクスポネンシャルバックオフメカニズムを使用して一時エラーを再試行します。再試行ロジックを実装する場合は、アプリケーションがデータの整合性を考慮するようにします。
- すべての本稼働用クラスターや貴重なデータがあるすべてのクラスターに対して、クラスターの削除保護を有効にします。Amazon DocumentDB クラスターを削除する前に、最終スナップショットを作成します。でリソースをデプロイする場合は CloudFormation、終了保護を有効にします。詳細については、「終了保護と削除保護」を参照してください。
- Amazon DocumentDB クラスターを作成する場合、`--engine-version` は、デフォルトで最新のメジャーエンジンバージョンになるオプションのパラメータです。現在のデフォルトのエンジンバージョンは 5.0.0 です (注: Amazon DocumentDB 8.0 は使用できますが、明示的に指定する必要があります) 。新しいメジャーエンジンバージョンがリリースされると、`--engine-version` のデフォルトのエンジンバージョンが最新のメジャーエンジンバージョンに更新されます。その結果、本稼働ワークロード、特にスクリプト、オートメーション、または CloudFormation テンプレートに依存するワークロードでは、 を目的のメジャーバージョン`--engine-version`に明示的に指定します。

インスタンスのサイズ指定

Amazon DocumentDB でインスタンスサイズを選択する最も重要な側面の 1 つは、キャッシュの RAM 容量です。Amazon DocumentDB は RAM の 3 分の 1 を独自のサービス用に使用しており、キャッシュに使用できるのはインスタンス RAM の 3 分の 2 のみです。そのため、Amazon DocumentDB のパフォーマンスのベストプラクティスとして、ワーキングセット (データやインデックスなど) をメモリに収めるのに十分な RAM を備えたインスタンスタイプを選択することを推奨します。インスタンスを適切なサイズに設定すると、全体的なパフォーマンスを最適化し、I/O コストを最小限に抑えることができます。サードパーティの Amazon DocumentDB サイズ計算ツールを使用して、特定のワークロードのインスタンスサイズを見積もることができます。

アプリケーションのワーキングセットがメモリに収まるかどうかを判断するには、Amazon CloudWatch を使用して、負荷がかかっているクラスター内のインスタンスごとに `BufferCacheHitRatio` をモニタリングします。

CloudWatch の BufferCacheHitRatio メトリクスは、インスタンスのメモリキャッシュから提供されたデータおよびインデックスの (ストレージボリュームに対する) 割合を測定します。一般的に、ワーキングセットメモリからのデータの読み取りは、ストレージボリュームからの読み取りよりも高速でコスト効率が高いため、BufferCacheHitRatio をできるだけ高い値に維持します。BufferCacheHitRatio はできるだけ 100% に近づけることが理想ですが、達成可能な最大値はアプリケーションのアクセスパターンやパフォーマンス要件によって異なります。可能な限り高い BufferCacheHitRatio を維持するために、インデックスとワーキングデータセットをメモリに収められるだけの十分な RAM を使用して、クラスター内のインスタンスをプロビジョニングすることを推奨します。

インデックスがメモリに収まらない場合は、BufferCacheHitRatio 値が小さくなります。ディスクからの継続的な読み取りは、追加の I/O コストが発生するため、メモリからの読み取りよりパフォーマンスが優れているとは言えません。BufferCacheHitRatio の比率が予想よりも小さい場合は、クラスターのインスタンスサイズをスケールアップして、ワーキングセットデータがメモリに収まるように RAM を大きくします。インスタンスクラスをスケールアップした結果 BufferCacheHitRatio が大幅に増加した場合、アプリケーションのワーキングセットがメモリに収まりません。スケーリングオペレーション後に BufferCacheHitRatio が劇的に増加しなくなるまで、継続してスケールアップします。インスタンスのメトリクスのモニタリングについては、「Amazon DocumentDB のメトリクス」を参照してください。

ワークロードやレイテンシーの要件に応じて、定常状態の使用中にアプリケーションの BufferCacheHitRatio 値が高くなることは許容されますが、コレクション全体をスキャンする必要がある分析クエリがインスタンスで実行されるため、定期的に BufferCacheHitRatio の急減が発生します。これらの定期的な BufferCacheHitRatio の急減は、後続のクエリでストレージボリュームからバッファキャッシュにワーキングセットデータを再入力する場合に、レイテンシーの増加として現れることがあります。まず、代表的な本番ワークロードを使用して本番稼働前の環境でワークロードをテストし、パフォーマンス特性を理解し、ワークロードを本番稼働環境にデプロイ**BufferCacheHitRatio**します。

BufferCacheHitRatio はインスタンス固有のメトリクスであるため、プライマリインスタンスとレプリカインスタンス間での読み取りの分散方法に応じて、同じクラスター内のインスタンスが異なる BufferCacheHitRatio 値を持つ場合があります。運用ワークロードが、分析クエリの実行後にワーキングセットキャッシュの再入力に伴う定期的なレイテンシーの増加に対応できない場合は、分析クエリのバッファキャッシュから通常のワークロードのバッファキャッシュを分離する必要があります。BufferCacheHitRatio の完全な分離を達成するには、運用クエリをプライマリインスタンスに転送し、分析クエリをレプリカインスタンスにのみ転送します。また、部分的な分離を達成するには、分析クエリを特定のレプリカインスタンスに転送できます。ただし、このレプリカでは通常のクエリも一部実行されるため、影響を受ける可能性があります。

適切な BufferCacheHitRatio 値は、ユースケースとアプリケーションの要件によって異なります。このメトリクスには 1 つの最適値や最小値はありません。コストとパフォーマンスの観点から、BufferCacheHitRatio 値の一時的な低下に伴うトレードオフを許容できるかどうかはお客様の判断次第です。

インデックスの使用

インデックスの構築

Amazon DocumentDB にデータをインポートする場合は、大きなデータセットをインポートする前にインデックスを作成する必要があります。Amazon DocumentDB インデックスツール を使用すると、実行中の MongoDB インスタンスや mongodump ディレクトリからインデックスを抽出し、Amazon DocumentDB クラスターでそれらのインデックスを作成できます。移行の詳細については、Amazon DocumentDB の移行とアップグレード を参照してください。

インデックスの選択性

インデックスの作成を、重複する値の数がコレクション内のドキュメントの合計数の 1% 未満のフィールドに制限します。例えば、コレクションに 100,000 個のドキュメントが含まれている場合、同じ値が発生する回数が 1000 回以下のフィールドにのみインデックスを作成します。

一意の値が多数ある (つまり、濃度が高い) インデックスを選択すると、フィルター処理によって返されるドキュメントの数が少なくなるため、インデックススキャン中のパフォーマンスが向上します。高濃度インデックスの例は一意のインデックスです。これにより、等式の述語が最大で 1 つのドキュメントを返すことが保証されます。低濃度の例としては、ブール型フィールドのインデックスと、曜日別のインデックスなどがあります。パフォーマンスが低下するため、低濃度のインデックスは、データベースのクエリオプティマイザによって選択される可能性が低くなります。同時に、低濃度のインデックスは、ディスク領域や I/O などのリソースを消費し続けます。経験則として、標準値の頻度がコレクション全体のサイズの 1% 以下のフィールドのインデックスをターゲットにしてください。

さらに、よくフィルターとして使用されるフィールドに対してのみインデックスを作成し、未使用のインデックスを定期的に検索することを推奨します。詳細については、「インデックスの使用状況を分析し、未使用のインデックスを特定する方法」を参照してください。

インデックスがデータの書き込みに与える影響

インデックスを使用すると、コレクション内のすべてのドキュメントをスキャンする必要がなくなるためクエリのパフォーマンスが向上しますが、このメリットにはトレードオフがあります。コレクションのインデックスごとに、ドキュメントが挿入、更新、または削除されるたびに、データベー

スはコレクションを更新し、コレクションの各インデックスにフィールドを書き込む必要があります。例えば、コレクションに 9 つのインデックスがある場合、データベースはクライアントへのオペレーションを承認する前に 10 回の書き込みを実行する必要があります。したがって、インデックスが増えるたびに、書き込みレイテンシーと I/O が多くなり、全体的なストレージ使用率が高まります。

クラスターインスタンスは、すべてのワーキングセットメモリを保持するために適切なサイズにする必要があります。これにより、インデックスページをストレージボリュームから継続的に読み取る必要がなくなるため、パフォーマンスに悪影響を及ぼし、I/O コストが高くなります。詳細については、「インスタンスのサイズ指定」を参照してください。

最高のパフォーマンスを得るには、コレクション内のインデックスの数を最小限に抑え、一般的なクエリのパフォーマンスを高めるために必要なインデックスのみを追加します。ワークロードはさまざまですが、ガイドラインとして、コレクションあたりのインデックス数を 5 つ以下に抑えることを推奨します。

欠落しているインデックスの識別

ベストプラクティスとして、欠落しているインデックスを定期的に特定します。詳細については、「欠落しているインデックスを特定する方法」を参照してください。

使用されていないインデックスの識別

ベストプラクティスとして、未使用のインデックスを定期的に特定して削除します。詳細については、「インデックスの使用状況を分析し、未使用のインデックスを特定する方法」を参照してください。

セキュリティのベストプラクティス

Amazon DocumentDB のセキュリティのベストプラクティスについては、「」を参照してくださいAmazon DocumentDB のセキュリティベストプラクティス。

コスト最適化

次のベストプラクティスは、Amazon DocumentDB を使用する際のコストの管理と最小化に役立ちます。料金に関する情報は、「Amazon DocumentDB (MongoDB との互換性あり) の料金 と Amazon DocumentDB (MongoDB との互換性あり) のよくある質問」を参照してください。

- 該当月の予想請求額の 50% と 75% のしきい値で請求アラートを作成します。請求アラートの作成の詳細については、「請求アラームの作成」を参照してください。

- Amazon DocumentDB のアーキテクチャは、ストレージとコンピューティングを分離するため、単一インスタンスクラスターでも高い耐久性を備えています。クラスターストレージボリュームは、3 つのアベイラビリティーゾーンにわたって 6 つの方法でデータをレプリケートすることにより、クラスター内のインスタンス数に関係なく、きわめて高い耐久性を実現します。一般的な本稼働クラスターには、高可用性を実現するために 3 つ以上のインスタンスがあります。ただし、高可用性を必要としない場合は、単一インスタンス開発クラスターを使用してコストを最適化できます。
- 開発およびテストシナリオでは、不要になったらクラスターを停止し、開発が再開されたらクラスターを起動します。詳細については、「Amazon DocumentDB クラスターの停止と開始」を参照してください。
- TTL ストリームと変更ストリームでは、どちらもデータの書き込み、読み取り、削除時に I/O が発生します。これらの機能を有効にしているが、アプリケーションで使用していない場合、機能を無効にすると、コスト削減に役立ちます。

メトリクスを使用したパフォーマンスの問題の特定

トピック

- パフォーマンスメトリクスの表示
- CloudWatch アラームの設定
- パフォーマンスメトリクスの評価
- CloudWatch メトリクスで Amazon DocumentDB インスタンスの使用状況を評価する
- クエリのチューニング

リソース不足やその他の一般的なボトルネックに起因するパフォーマンスの問題を特定するために、Amazon DocumentDB クラスターに適用されるメトリクスをモニタリングできます。

パフォーマンスメトリクスの表示

さまざまな時間範囲の平均値、最大値、最小値を表示するには、パフォーマンスメトリクスを定期的に監視します。これは、いつパフォーマンスが低下しているかを特定するうえで有効です。また、特定のメトリクスしきい値に対して Amazon CloudWatch アラームを設定すると、しきい値に達した場合に警告されるように設定できます。

パフォーマンスの問題を解決するために重要なのは、システムのベースラインパフォーマンスを理解することです。新しいクラスターをセットアップし、一般的なワークロードで実行したら、すべての

パフォーマンスメトリクスの平均値、最大値、最小値をさまざまな間隔 (例: 1 時間、24 時間、1 週間、2 週間) で取得します。これにより、正常な状態を把握することができます。それにより、オペレーションのピークおよびオフピークの時間帯を比較して、得られた情報から、いつパフォーマンスが標準レベルを下回っているかを特定できます。

パフォーマンスメトリクスは、 AWS マネジメントコンソール または を使用して表示できます AWS CLI。詳細については、「CloudWatch データの表示」を参照してください。

CloudWatch アラームの設定

CloudWatch アラームを設定するには、CloudWatch ユーザーガイドの「Amazon CloudWatch アラームの使用」を参照してください。

パフォーマンスメトリクスの評価

インスタンスには、さまざまなカテゴリのメトリクスがあります。許容値を決定する方法は、メトリクスによって異なります。

CPU

- CPU使用率 - コンピュータの処理能力のうち、使用されている容量の割合。

メモリ

- 空きメモリ — インスタンスで使用可能な RAM の容量。
- スワップ領域 — インスタンスが使用しているスワップスペースの量 (メガバイト単位)。

入力/出力オペレーション

- 読み込み IOPS、書き込み IOPS — 1 秒あたりのディスク読み取りまたは書き込み操作の平均数。
- 読み取りレイテンシー、書き込みレイテンシ — 1 秒あたりのディスク読み取りまたは書き込み操作の平均時間 (ミリ秒単位)。
- 読み取りスループット、書き込みスループット — 1 秒あたりのディスク読み取りまたは書き込みデータの平均量 (メガバイト単位)。
- Disk Queue Depth — ディスクに対する読み取りまたは書き込み待機中の I/O オペレーションの数。

ネットワークトラフィック

- ネットワーク受信スループット、ネットワーク送信スループット — インスタンスに対する送信または受信ネットワークトラフィックの1 秒あたりのレート (メガバイト単位)。

データベース接続

- DB接続数 — インスタンスに接続されたクライアントセッションの数。

一般的に、パフォーマンスメトリクスの許容値は、ベースラインがどのようになっているか、アプリケーションによって何が実行されているかによって異なります。ベースラインからの一貫した差異またはトレンドになっている差異を調べます。

メトリクスのタイプごとの推奨事項とアドバイスは次のとおりです。

- CPU の高消費量 - CPU消費量が高い値になっていても、アプリケーションの目標 (スループットや並行処理など) に沿った想定値であれば、妥当である場合があります。CPU 使用率が一貫して 80% を超える場合は、インスタンスのスケールアップを検討してください。
- RAM の高消費量 — `FreeableMemory` メトリクスがインスタンスメモリの合計の 10% を頻繁に下回る場合は、インスタンスのスケールアップを検討してください。DocumentDB インスタンスで高いメモリプレッシャーが発生している場合に何が起こるかについての詳細は、「Amazon DocumentDB リソースガバナンス」を参照してください。
- スワップの使用量 — このメトリクスは 0 またはほぼ 0 のままにする必要があります。スワップの使用量が多い場合は、インスタンスのスケールアップを検討してください。
- ネットワークトラフィック — ネットワークトラフィックについてシステム管理者に問い合わせて、ドメインネットワークとインターネット接続の想定スループットを把握してください。スループットが一貫して想定よりも低い場合は、ネットワークトラフィックを調べます。
- データベース接続数 — ユーザー接続数が多いことが、インスタンスのパフォーマンスと応答時間の低下とに関連している場合は、データベース接続数の制限を検討してください。インスタンスの最適なユーザー接続数は、インスタンスのクラスと実行中のオペレーションの複雑さによって異なります。パフォーマンスメトリクスに問題がある場合、パフォーマンスを向上させるためにまず実行できることの 1 つは、使用頻度とコストの最も高いクエリをチューニングして、システムリソースへの負荷が下がるかどうかを確認することです。

クエリをチューニングしても問題が解決しない場合は、その問題に関連するリソース(CPU、RAM、ディスク容量、ネットワーク帯域幅、I/O 容量)を増加できる Amazon DocumentDB インスタンスクラスにアップグレードすることを検討してください。

CloudWatch メトリクスで Amazon DocumentDB インスタンスの使用状況を評価する

CloudWatch メトリクスを使用して、インスタンスのスループットを監視し、インスタンスクラスがアプリケーションに十分なリソースを提供しているかどうかを確認できます。インスタンスクラスのクォータの詳細については、インスタンスクォータ「」を参照してインスタンスクラスの仕様を見つけ、ネットワークパフォーマンスを見つけます。

インスタンスの使用量がインスタンスクラスの限度に近い場合、パフォーマンスが低下し始める可能性があります。CloudWatch メトリクスによってこの状況を確認できるので、より大きなインスタンスクラスへの手動スケールアップを計画できます。

以下の CloudWatch メトリクス値を組み合わせて、インスタンスクラスの限界に近づいているかどうかを確認します。

- NetworkThroughput - Amazon DocumentDB クラスター内の各インスタンスについて、各クライアントによって送受信されたネットワークスループットの量。このスループット値には、クラスター内のインスタンスとクラスターストレージボリューム間のネットワークトラフィックは含まれません。
- StorageNetworkThroughput — Amazon DocumentDB クラスター内の各インスタンスによって Amazon DocumentDB クラスターストレージボリュームに送受信されるネットワークスループットの量。

NetworkThroughput を StorageNetworkThroughput に加えると、Amazon DocumentDB クラスター内の各インスタンスと Amazon DocumentDB クラスターストレージボリュームとの間で送受信されたネットワークスループット量が得られます。インスタンスのインスタンスクラス限界は、この 2 つのメトリクスの合計よりも大きい必要があります。

以下のメトリクスを使用して、送受信時のクライアントアプリケーションからのネットワークトラフィックの詳細を確認できます。

- NetworkReceiveThroughput – Amazon DocumentDB クラスター内の各インスタンスがクライアントから受信したネットワークスループットの量。クラスター内のインスタンスとクラスターストレージボリューム間のネットワークトラフィックは、このスループットに含まれません。

- NetworkTransmitThroughput – Amazon DocumentDB クラスター内の各インスタンスが各クライアントに送信したネットワークスループットの量。クラスター内のインスタンスとクラスターストレージボリューム間のネットワークトラフィックは、このスループットに含まれません。
- StorageNetworkReceiveThroughput – Amazon DocumentDB クラスター内の各インスタンスがクラスターストレージボリュームから受信したネットワークスループットの量。
- StorageNetworkTransmitThroughput – Amazon DocumentDB クラスター内の各インスタンスがクラスターストレージボリュームに送信したネットワークスループットの量。

これらすべてのメトリクスを合計して、ネットワーク使用量とインスタンスクラスの限界との比較を評価します。インスタンスクラス限界は、これらのメトリクスの合計よりも大きい必要があります。

ネットワーク限度とインスタンスによる CPU 使用率は相互に関係しています。ネットワークのスループットが増加すると、CPU 使用率も増加します。CPU とネットワークの使用状況を監視すると、リソースがどのくらい枯渇しているのか、またなぜ枯渇しているのかについての情報が得られます。

ネットワークの使用量を最小限に抑えるには、次のことを検討してください。

- より大きなインスタンスクラスを使用します。
- 書き込みリクエストをバッチに分割して、トランザクション全体を減らします。
- 読み取り専用ワークロードを読み取り専用インスタンスに転送します。
- 未使用のインデックスをすべて削除します。

クエリのチューニング

クラスターのパフォーマンスを向上させるには、大量のリソースを消費する使用頻度の最も高いクエリをチューニングして、実行コストを下げることを推奨します。

プロファイラーを使用して(Amazon DocumentDB オペレーションのプロファイリング を参照)、クラスターで実行されたオペレーションの実行時間と詳細を記録できます。プロファイラーは、クラスターで最も遅いオペレーションをモニタリングし、個々のクエリパフォーマンスとクラスター全体のパフォーマンスを向上させるのに役立ちます。

`explain` コマンドを使用して、特定のクエリのクエリプランを分析する方法を学習することもできます。この情報を使用してクエリまたは基礎となるコレクションを変更して、クエリのパフォーマンスを向上させます(例えば、インデックスの追加)。

TTL および時系列ワークロード

TTL インデックスの有効期限が切れたことによるドキュメントの削除は、ベストエフォートプロセスです。ドキュメントが特定の期間内に削除されることは保証されません。インスタンスサイズ、インスタンスリソース使用率、ドキュメントサイズ、全体的なスループット、インデックスの数、およびインデックスとワーキングセットがメモリに収まるかどうかなどの要因はすべて、期限切れのドキュメントが TTL プロセスによって削除されるタイミングに影響します。

TTL モニターがドキュメントを削除すると、削除ごとに I/O コストが発生するため、請求が増加します。スループットおよび TTL 削除のレートが増加した場合、I/O 使用量が増加するため、請求の増加が予想されます。ただし、TTL インデックスを作成してドキュメントを削除せず、時間に基づいてドキュメントをコレクションにセグメント化し、不要になったときにそのコレクションを削除する場合、I/O コストは発生しません。これは、TTL インデックスを使用するよりもコスト効率が大幅に向上します。

時系列ワークロードの場合、TTL インデックスの代わりにローリングコレクションを作成することを検討できます。ローリングコレクションは、データを削除し、I/O 負荷を小さくする優れた方法となるためです。大規模なコレクション (特に 1TB を超えるコレクション) がある場合、または TTL 削除 I/O コストが懸念される場合は、時間に基づいてドキュメントをコレクションに分割し、ドキュメントが不要になったときにコレクションを削除します。データ取り込みレートに応じて、1 日または週に 1 つのコレクションを作成できます。要件はアプリケーションによって異なりますが、経験則として、いくつかの大きいコレクションではなく小さいコレクションにすることをお勧めします。これらのコレクションを削除しても I/O コストは発生せず、TTL インデックスを使用するよりもコスト効率が向上します。

移行

Amazon DocumentDB にデータを移行するためのベストプラクティスとして、まず Amazon DocumentDB にインデックスを作成し、次にデータを移行します。最初にインデックスを作成すると、全体の時間が短縮され、移行速度が向上します。これを行うには、Amazon DocumentDB インデックスツール を使用します。移行の詳細については、「」を参照してくださいAmazon DocumentDB の移行とアップグレード。

また、本番データベースを移行する前に、機能、パフォーマンス、オペレーション、コストなどを考慮して、Amazon DocumentDB でアプリケーションを完全にテストすることをお勧めします。

クラスターパラメータグループの使用

本番稼働用クラスターに変更を適用する前に、テストクラスターでクラスターパラメータグループの変更を試してください。クラスターのバックアップの詳細については、「Amazon DocumentDB でのバックアップと復元」を参照してください。

集約パイプラインクエリ

複数のステージを持つ集約パイプラインクエリを作成し、クエリ内のデータのサブセットのみを評価する場合は、`$match` ステージをパイプラインの最初のステージまたは先頭部分として使用します。`$match` を最初に使用すると、集約パイプラインクエリ内の後続ステージで処理するドキュメントの数が減り、クエリのパフォーマンスが向上します。

`batchInsert` および `batchUpdate`

`batchInsert` または `batchUpdate` の同時実行オペレーションを高率で実行しているときに、プライマリインスタンスで `FreeableMemory` (CloudWatch のメトリクス) の量がゼロになった場合は、バッチ挿入ワークロードやバッチ更新ワークロードの同時実行数を減らすことができます。ワークロードの同時実行数を減らせない場合は、インスタンスサイズを大きくして `FreeableMemory` の量を増やすことができます。

Amazon DocumentDB インデックスの管理

トピック

- Amazon DocumentDB インデックスの作成
- Amazon DocumentDB インデックスのメンテナンス

Amazon DocumentDB インデックスの作成

Amazon DocumentDB でインデックスを作成するには、いくつかの決定を行う必要があります。

- どのくらいの速さで完了する必要がありますか?
- ビルドの実行中にコレクションにアクセスできなくなることはありますか?
- インスタンスのコンピューティング能力のどれだけをビルドに割り当てることができますか?
- どのタイプのインデックスを作成すべきですか?

このセクションでは、これらの質問に回答し、インスタンスベースのクラスターコレクションに Amazon DocumentDB インデックスを作成するためのコマンドとモニタリングの例を示します。

ガイドライン

以下のガイドラインには、新しいインデックスを作成する際の基本的な制限と設定のトレードオフが含まれています。

- Amazon DocumentDB バージョンのサポート - 単一ワーカーインデックス作成はすべての Amazon DocumentDB バージョンでサポートされていますが、複数のワーカーインデックス作成は Amazon DocumentDB バージョン 4.0、5.0、および 8.0 でのみサポートされています。
- パフォーマンスのトレードオフ: インデックス作成プロセスのワーカー数を増やすと、Amazon DocumentDB データベースのプライマリインスタンスの CPU 使用率と読み取り IO が増加します。新しいインデックスを作成するのに必要なリソースは、実行中のワークロードでは使用できなくなります。
- Elastic クラスター: Amazon DocumentDB Elastic クラスターでは並列インデックスはサポートされていません。
- サーバーレスクラスター - サーバーレスクラスターでの並列インデックス作成には、少なくとも 32 DCUsの最小容量が必要です。このしきい値を下回ると、 `workers`パラメータは受け入れられますが、インデックスビルドは 1 つのワーカーで実行されます。並列ワーカーの最大数は、設定された最小容量でスケーリングされます。約 の`min_DCU / 8`ワーカーを使用できます。たとえば、32 個の DCUs では最大 4 個のワーカーを使用し、64 DCUs では最大 8 個のワーカーを使用できます。

  > **Note**
  >
  > 新しい並列ワーカーの制限を有効にするには、最小容量設定を変更した後にクラスターを再起動する必要があります。

- 最大ワーカー数 (インスタンスベース) - 設定できるワーカーの最大数は、データベースクラスター内のプライマリインスタンスのサイズによって異なります。これは、データベースクラスターのプライマリインスタンスにある vCPU の総数の半分です。たとえば、64 個の vCPU を備えた db.r6g.16xlarge インスタンスでは、最大 32 ワーカーを実行できます。

  > **Note**
  >
  > 並列ワーカーは、2xlarge 以降のインスタンスでのみサポートされています。

- 最小ワーカー: 設定できるワーカーの最小数は 1 です。インスタンスベースのクラスターでのインデックス作成のデフォルト設定は 2 ワーカーです。ただし、「ワーカースレッド」オプションを使用すると、ワーカーの数を 1 つに減らすことができます。これはプロセスが 1 つのワーカーで実行されます。
- インデックス圧縮: Amazon DocumentDB はインデックス圧縮をサポートしていません。インデックスのデータサイズは、他のオプションを使用したときより大きくなる可能性があります。
- 複数コレクションのインデックス作成: データベースクラスターのプライマリインスタンスにある vCPU の半分を、複数のコレクションでインデックス作成を実行する設定済みワーカーに使用できます。
- インデックスタイプ - Amazon DocumentDB でサポートされているインデックスタイプの詳細な説明については、 AWS データベースブログのAmazon DocumentDB でインデックスを作成する方法」を参照してください。

開始方法

特定のコレクションでインデックスの作成を開始するには、`createIndexes` コマンドを使用します。デフォルトでは、このコマンドは 2 つの並列ワーカーを実行するため、インデックス作成プロセスの速度が 2 倍向上します。

たとえば、次のコマンドプロセスは、ドキュメント内の「user_name」フィールドのインデックスを作成し、インデックス作成プロセスの速度を 4 ワーカーに上げる方法を示しています。

1. クラスター上の 2 つの並列ワーカーを使用してインデックスを作成します。

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx"}]})
```

2. インデックス作成プロセスの速度を最適化するため、`db.runCommand createIndexes` コマンドの「ワーカースレッド」オプション (`"workers":<number>`) を使用してワーカー数を指定できます。

   プロセスの速度を 4 つの並列ワーカーに上げます。

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx", "workers":4}]})
```

> **Note**
>
> ワーカーの数が増えるほど、インデックスの作成が速くなります。ただし、ワーカーの数が増えるほど、プライマリインスタンスの vCPU と読み取り IO のロードも大きくなります。他のワークロードを低下させることなく、増加したロードを処理できるよう、クラスターが十分にプロビジョニングされていることを確認してください。

インデックスの進捗状況

インデックス作成プロセスは、初期化、コレクションのスキャン、キーのソート、そして最後にインデックスビルダーによるキーの挿入によって行われます。このプロセスは、フォアグラウンドで実行する場合は最大 6 段階、バックグラウンドで実行する場合は最大 9 段階に分かれます。完了率、スキャンされたストレージブロックの総数、ソートされたキー、挿入されたキーなどのステータスメトリックをステージごとに表示できます。

mongo シェルの `db.currentOp()` コマンドを使用して、インデックス作成プロセスの進行状況をモニタリングします。最後の段階が 100% 完了すると、すべてのインデックスが正常に作成されたことがわかります。

```
db.currentOp({"command.createIndexes": { $exists : true } })
```

> **Note**
>
> インデックス作成の進行状況の表示は、Amazon DocumentDB 5.0 でのみサポートされています。

インデックスビルドのタイプ

インデックスビルドには次の 4 つのタイプがあります。

- フォアグラウンド — フォアグラウンドインデックスビルドは、インデックスが作成されるまで、他のすべてのデータベース操作をブロックします。Amazon DocumentDB フォアグラウンドビルドは 5 つのステージで構成されています。
- フォアグラウンド (単一) — 単一ドキュメント (単一) フォアグラウンドインデックスビルドは、通常のフォアグラウンドビルドのような他のデータベース操作をブロックします。基本的なフォアグ

ラウンドビルドとは異なり、単一ビルドでは追加のステージ (ソートキー 2) を使用して重複キーを探します。フォアグラウンド (単一) ビルドは 6 つのステージで構成されます。

- バックグラウンド — バックグラウンドでのインデックス構築により、インデックスの作成中に他のデータベース操作をフォアグラウンドで実行できます。Amazon DocumentDB バックグラウンドビルドは 8 つのステージで構成されています。
- バックグラウンド (単一) — 単一ドキュメント (単一) のバックグラウンドインデックス構築により、インデックスの作成中に他のデータベース操作をフォアグラウンドで実行できます。基本的なバックグラウンドビルドとは異なり、単一ビルドでは追加のステージ (ソートキー 2) を使用して重複キーを探します。バックグラウンド (単一) ビルドは 9 つのステージで構成されます。

インデックスビルドステージ

ステージ	フォアグラウンド	フォアグラウンド (単一)	背景	バックグラウンド (単一)
初期化	1	1	1	1
インデックスの構築: 初期化	2	2	2	2
インデックスの構築: コレクションのスキャン	3	3	3	3
インデックスの構築: ソートキー 1	4	4	4	4
インデックスの構築: ソートキー 2		5		5
インデックスの構築: キーの挿入	5	6	5	6
検証: インデックスのスキャン			6	7

ステージ	フォアグラウンド	フォアグラウンド (単一)	背景	バックグラウンド (単一)
検証: タプルのソート			7	8
検証: コレクションのスキャン			8	9

- 初期化: createIndex はインデックスビルダーを準備しています。このフェーズは非常に短いはずです。
- インデックスの構築: 初期化 - インデックスビルダーはインデックスを作成する準備をしています。このフェーズは非常に短いはずです。
- インデックスの構築: コレクションのスキャン - インデックスビルダーはコレクションスキャンを実行してインデックスキーを収集しています。測定単位は「ブロック」です。

> **Note**
>
> インデックス構築用に複数のワーカーが設定されている場合は、この段階で表示されます。「コレクションのスキャン」ステージは、インデックス構築プロセス中に複数のワーカーを使用する唯一のステージです。その他のすべてのステージには 1 人のワーカーが表示されます。

- インデックスの構築: ソートキー 1 - インデックスビルダーは収集されたインデックスキーをソートしています。測定単位は「キー」です。
- インデックスの構築: ソートキー 2 - インデックスビルダーは、収集されたデッドタプルに対応するインデックスキーをソートします。このフェーズは単一インデックスの構築にのみ適用されます。測定単位は「キー」です。
- インデックスの構築: キーの挿入 - インデックスビルダーは、新しいインデックスにインデックスキーを挿入しています。測定単位は「キー」です。
- 検証: インデックスのスキャン - createIndex はインデックスをスキャンして、検証が必要なキーを見つけます。測定単位は「ブロック」です。
- 検証: タプルのソート - createIndex はインデックススキャンフェーズの出力をソートしています。
- 検証: コレクションのスキャン - createIndex はコレクションをスキャンして、前の 2 つのフェーズで見つかったインデックスキーを検証しています。測定単位は「ブロック」です。

インデックスビルドの出力例

次の出力例 (フォアグラウンドインデックスビルド) では、インデックス作成のステータスが表示されます。「msg」フィールドには、ビルドのステージと完了率を示すことで、ビルドの進捗状況がまとめられています。「workers」フィールドは、インデックスビルドの各段階で使用されたワーカーの数を示します。「progress」フィールドには、完了率の計算に使用された実際の数値が表示されます。

> **Note**
>
> 「currentIndexBuildName」、「msg」、および「progress」フィールドは Amazon DocumentDB バージョン 4.0 ではサポートされていません。

```
{
    "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
}
```

Amazon DocumentDB インデックスのメンテナンス

トピック

- インデックスの肥大化
- reIndex を使用したインデックスのメンテナンス

インデックスの肥大化

Amazon DocumentDB は、複数バージョンの同時実行制御 (MVCC) を使用して同時トランザクションを管理します。ドキュメントを削除または更新しても、以前のバージョンはコレクションとインデックスに「デッド」バージョンとして残ります。ガベージコレクションプロセスは、将来のオペレーションのために、これらのデッドバージョンからスペースを自動的に再利用します。

インデックスの肥大化は、デッドインデックスエントリや古いインデックスエントリの蓄積、またはページ内のフラグメント化により、コレクションのインデックスが大きくなった場合に発生します。レポートされる割合は、将来のインデックスエントリで使用できるインデックススペースの量を表します。この肥大化は、バッファキャッシュとストレージの両方の領域を消費します。肥大化を削除する場合は、インデックスを再構築する必要があります。

Example例

次のコマンドを実行して、インデックスの未使用のストレージを確認します。

```
db.coll.aggregate({$indexStats:{}});
```

結果は以下のようになります。

```
{
    "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
    }
}
```

reIndex コマンドを使用して、ダウンタイムなしでインデックスを再構築できます。これには、コレクション全体のスキャンが必要です。「reIndex を使用したインデックスのメンテナンス」を参照してください。

reIndex を使用したインデックスのメンテナンス

reIndex は、インデックスの再構築に使用されるコマンドです。通常、インデックスが破損したり、または非効率的になった場合に使用されます。時間の経過とともに、インデックスには多くの更新、挿入、削除に伴い未使用のスペースが蓄積され、パフォーマンスが低下する恐れがあります。インデックスの再構築は、このような未使用のスペースを削除し、インデックスの効率を回復するのに役立ちます。

reIndex ガイドライン

- reIndex は現在、Amazon DocumentDB 5.0 でのみサポートされています。
- Amazon DocumentDB は、バックグラウンドでの 1 つのインデックスに対する reindex をサポートし、複数のワーカーの関与を可能にします。reIndex プロセスの実行中も、古いインデックスを用いたクエリの実行が可能です。
- Amazon DocumentDB は、currentOp によるインデックス構築の進行状況レポートをサポートしています。レポートでは、インデックスの作成中に表示される インデックスビルドステージ と同様のインデックス構築ステージを確認できます。唯一の違いは、一意であるかどうかにかかわらず、reIndex のステージ数は常に 8 ステージであることです。「インデックスのビルド中: キーのソート中 #2」のようなステージはありません。
- reIndex は、同じコレクション上で実行されるインデックス関連のコマンド (createIndexes、dropIndexes、collMod、renameCollection) を除いて、あらゆるコマンドと同時に実行できます。

- reIndex は現在、テキスト、地理空間、ベクトル、部分インデックスについてはサポートされていません。

reIndex の構築

次のコマンドを使用してインデックスを再構築します。

```
db.runCommand({ reIndex: "collection-name", index: "index-name"})
```

オプションで、再構築プロセスに割り当てるワーカーの数をコントロールすることもできます。

```
db.runCommand({ reIndex: "collection-name", index: "index-name", workers: number })
```

コレクションレベルのドキュメント圧縮の管理

Amazon DocumentDB 5.0 コレクションレベルのドキュメント圧縮では、コレクション内のドキュメントを圧縮することで、ストレージと IO のコストを削減できます。コレクションレベルでのドキュメント圧縮を有効にし、圧縮ドキュメントのストレージサイズや圧縮ステータスなどの圧縮メトリクスに基づいて得られたストレージを測定することで、必要に応じて圧縮メトリクスを確認できます。Amazon DocumentDB 5.0 は、LZ4 圧縮アルゴリズムを使用してドキュメントを圧縮します。

Amazon DocumentDB は、バージョン 5.0 以降でドキュメント圧縮をサポートしています。以下は、コレクションレベルのドキュメント圧縮機能です。

- デフォルトの動作 — 5.0 クラスターの新しいコレクションのデフォルトの圧縮設定は、クラスターパラメータ によって決まりますdefault_collection_compression。このパラメータは、デフォルトでは「無効」に設定されています。
- 既存のコレクションの圧縮 — 既存のコレクションの圧縮設定は、collMod コマンドを使用して変更できます。
- 圧縮しきい値の変更 — デフォルトの圧縮しきい値は 2KB です。createCollection コマンドを使用して新しいコレクション向けに圧縮しきい値を指定したり、collMod コマンドを使用して既存のコレクション向けに値を変更することができます。

> **Note**
>
> Amazon DocumentDB ドキュメント圧縮は Amazon DocumentDB バージョン 3.6 および 4.0 ではサポートされていません。

トピック

- ドキュメント圧縮の管理
- ドキュメント圧縮のモニタリング

ドキュメント圧縮の管理

コレクション内でのドキュメント圧縮の有効化

`db.createCollection()` メソッドを使用して Amazon DocumentDB 5.0 でコレクションを作成するときにドキュメント圧縮を有効にします。

```
db.createCollection( sample_collection,{
    storageEngine : {
        documentDB: {
            compression:{enable: <true | false>}
        }
    }
})
```

クラスター内でのドキュメント圧縮の有効化

クラスターパラメータ `default_collection_compression` を「有効」に設定することで、クラスターレベルのすべての新しいコレクションでドキュメント圧縮をデフォルトで有効にできます。このパラメータを「有効」に設定すると、クラスターで新しく作成されるコレクションではデフォルトで圧縮が有効になり、圧縮しきい値は 2 KB になります。

既存のコレクションの圧縮

`collMod` オペレーションの使用により、以下の `storageEngine` 設定を指定して既存のコレクションの圧縮設定を変更することもできます。このコマンドを使用して行われた変更は、新しく挿入/更新されたドキュメントにのみ適用され、以前に挿入されたドキュメントに対する圧縮は変更されないことに注意してください。

```
db.runCommand({
    collMod: "orders",
    storageEngine: {
        documentDB: {compression: {enable: <true | false>} }
    }
})
```

圧縮しきい値の設定

デフォルトでは、圧縮されたコレクションの圧縮しきい値は 2032 バイトです。このしきい値は、圧縮を有効にして新しいコレクションを作成するときに createCollection コマンドで設定できます。

```
db.createCollection( sample_collection, {
    storageEngine : {
        documentDB: {
            compression: {
                enable: true,
                threshold: <128 - 8000>
            }
        }
    }
})
```

collMod オペレーションの使用により、次の storageEngine 設定を指定して既存の圧縮コレクションの圧縮しきい値を変更することもできます。

```
db.runCommand({
    collMod: "orders",
    storageEngine: {
        documentDB: {
            compression: {
                enable: true,
                threshold: <128 - 8000>
            }
        }
    }
})
```

圧縮しきい値は 128 ~ 8000 バイトの値にのみ設定できます。さらに、圧縮しきい値を指定するときは、enable オプションを「true」に設定する必要があります。

ドキュメント圧縮のモニタリング

コレクションが圧縮されているかどうかを確認し、圧縮率を求める方法は次の通りです。

mongo シェルから db.printCollectionStats() または db.collection.stats() コマンドを実行して圧縮統計を表示します。出力に元のサイズと圧縮サイズが表示されるため、両者を比較してドキュメント圧縮によって得られたストレージを分析できます。この例では、「sample_collection」という名前のコレクションの統計が下に表示されます。以下のように、スケーリング係数 1024*1024 を使用して、size と storageSize の値を MB 単位で出力します。

```
db.sample_collection.stats(1024*1024)
```

上記のコマンドを使った出力例を以下にご紹介します。

```
{
    "ns" : "test.sample_collection",
    "count" : 1000000,
    "size" : 3906.3,
    "avgObjSize" : 4096,
    "storageSize" : 1953.1,
    compression:{"enabled" : true,"threshold" : 2032},
    ...
}
```

- size: ドキュメントコレクションの元のサイズ。
- avgObjSize: 圧縮前のドキュメントサイズ平均 (小数第一位まで四捨五入)。測定単位はバイト。
- storageSize: 圧縮後のコレクションのストレージサイズ。測定単位はバイト。
- enabled: 圧縮の有効/無効を示します。

実際の圧縮率を計算するには、コレクションサイズをストレージサイズで割ります (size/storageSize)。上記の例の場合、計算は 3906.3/1953.1 で、これは 2:1 の圧縮率に相当します。

Amazon DocumentDB 8.0 でのディクショナリベースの圧縮の管理

Amazon DocumentDB 8.0 では、LZ4 に代わる改善された方法として、新しいドキュメント圧縮アルゴリズム zstd が導入されました。圧縮アルゴリズムとして Zstd を選択して Amazon DocumentDB 8.0 コレクションでディクショナリ圧縮を有効にすると、コレクションのドキュメントがサンプリン

グされ、カスタム圧縮ディクショナリが作成されます。新規および更新されたドキュメントは、このディクショナリと zstd アルゴリズムを使用して圧縮されます。このアプローチは通常、特に一貫したドキュメントスキーマまたは繰り返しフィールド名を持つコレクションの場合、標準の圧縮方法よりも高い圧縮率を実現します。

Lz4 は、高速な圧縮と解凍のために設計されたアルゴリズムです。顕著な圧縮を達成しながら、CPU でライトであることが知られています。Zstd は、デフォルト設定でより多くの CPU を使用するが、lz4 よりも優れた圧縮率を実現する汎用アルゴリズムです。ディクショナリを使用すると、ほとんどの JSON ドキュメントの圧縮がさらに向上します。Zstd アルゴリズムの利点は次のとおりです。

- ストレージコストの削減: 圧縮率が高いほど、ストレージ使用量が少なくなり、コストが低くなります。
- 低 I/O: 圧縮されたドキュメントの I/O は少なくなるため、パフォーマンスが向上する可能性があります。
- コレクション用に最適化: ディクショナリは、コレクションのデータパターンに合わせて特別にトレーニングされます。

> **Note**
>
> ディクショナリベースの圧縮は、Amazon DocumentDB バージョン 3.6、4.0、および 5.0 ではサポートされていません。

パフォーマンスに関する考慮事項

Zstd 圧縮には、次のトレードオフが含まれます。

- ストレージと CPU: Zstd 圧縮では圧縮率が向上しますが、LZ4 圧縮と比較して CPU リソースがわずかに多く使用される場合があります。
- 初期圧縮: 新しいコレクションは、効果的なディクショナリをトレーニングするために十分なドキュメントが挿入されるまで、最適な圧縮を達成できない場合があります。現在、コレクションが 100 以上のドキュメントである場合、ディクショナリはトレーニングされます。
- ワークロードタイプ: データ全体がバッファキャッシュに収まる読み込み負荷の高いワークロードでは、解凍オーバーヘッドによりレイテンシーと CPU 使用率が増加する可能性があります。

Zstd 圧縮は、小さなドキュメント、ドキュメント配列、繰り返しフィールド名を持つコレクションに特に効果的です。

ディクショナリベースの圧縮の有効化

新しいコレクションでは、次のコマンドを使用して Zstd 圧縮を有効にできます。

```
db.createCollection("myCollection",
    {
        storageEngine: {
            documentDB: {
                compression: {
                    enable: true,
                    algorithm: "zstd"
                }
            }
        }
    }
 )
```

既存のコレクションで圧縮を有効化または変更することもできます。

```
db.runCommand({
    collMod: "myCollection",
    storageEngine: {
        documentDB: {
            compression: {
                enable: true,
                algorithm: "zstd"
            }
        }
    }
 })
```

クラスター上のすべてのコレクションで Zstd アルゴリズムを有効にするには、クラスターパラメータグループを変更して、パラメータ「default_collection_compression」の値として「zstd」を選択できます。

開始方法

Amazon DocumentDB 8.0 では、Zstd 圧縮がデフォルトでオンになっています。クラスターパラメータグループで「default_compression」の値を無効にすると、いつでもオフにできます。Amazon DocumentDB 8.0 以降、「enabled」は default_compression の有効な選択肢ではなくなり、Zstd と LZ4 から選択する必要があります。

モニタリング

次のいずれかのコマンドを使用して、コレクションの圧縮情報を表示できます。

- db.runCommand({ collStats: "myCollection" }) または
- db.collection.stats()

これらのコマンドは、圧縮率の計算に使用できる主要な統計を返します。

- compression.algorithm: 使用されるアルゴリズム (「lz4」または「zstd」)
- storageSize: 圧縮後にコレクションで使用される実際のストレージ。この数値にはフラグメント化 (データベースページの未使用のスペース) が含まれていることに注意してください。
- avgObjSize: 解凍されたコレクションのドキュメントの平均論理サイズ。コレクションに 20,000 件を超えるドキュメントがある場合、この数は 20,000 件のドキュメントのサンプルに基づく概算値になることに注意してください。
- size: 圧縮なしのコレクションの論理サイズ。この数は、avgObjSize にコレクション内のドキュメントの合計数を掛けることで取得されるため、avgObjSize が近似値である場合、この数も近似値になります。
- count: コレクション内のドキュメントの数

ディクショナリベースの圧縮を評価する際、以下の CloudWatch メトリクスが役立ちます。

- CPUUtilization
- FreeableMemory
- VolumeBytesUsed
- VolumeReadiOps
- VolumeWriteIOPs

collStats メトリクス:

- storageSize
- サイズ

さらに、クエリタイプまたは API あたりのレイテンシーやスループットなど、アプリケーション固有のメトリクスを追跡すると便利です。

Amazon DocumentDB のセキュリティ

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャを活用できます。

セキュリティは、 AWS お客様とお客様の間の責任共有です。このドキュメントは、Amazon DocumentDB 使用時における責任共有モデルの適用法を理解するのに役立ちます。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ — AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する AWS 責任があります。 は、安全に使用できるサービス AWS も提供します。「AWS 」 コンプライアンスプログラムの一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。Amazon DocumentDB (MongoDB 互換) に適用されるコンプライアンスプログラムについては、「AWS コンプライアンスプログラム別の適用範囲内のサービス」をご覧ください。
- クラウド内のセキュリティ — お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、お客様のデータの機密性、組織の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

Amazon DocumentDB は、「連邦リスク承認管理プログラム (FedRAMP)」の認可事業者です。これには、 AWS GovCloud (米国) リージョンの FedRAMP High 認可と AWS 、米国東部/西部リージョンの FedRAMP Moderate 認可があります。 AWS とコンプライアンスの取り組みの詳細については、AWS 「コンプライアンスプログラムによる対象範囲内のサービス」を参照してください。

> **Note**
>
> この章は、インスタンスベースのクラスターと Elastic クラスターの両方に適用されます。詳細については、以下のトピックを参照してください。

また、Amazon DocumentDB リソースのモニタリングや保護に役立つ他の AWS サービスの使用方法についても説明します。以下のトピックでは、セキュリティとコンプライアンスの目的を満たすように Amazon DocumentDB を設定する方法について説明します。

トピック

- Amazon DocumentDB と によるパスワード管理 AWS Secrets Manager

- Amazon DocumentDB におけるデータ保護
- Amazon DocumentDB の Identity and Access Management
- サービス間での不分別な代理処理の防止
- IAM ID を使用した認証
- SCRAM-SHA-256 を使用した認証
- Amazon DocumentDB ユーザーの管理
- ロールベースのアクセス制御 (RBAC) を使用したデータベースへのアクセス
- Amazon DocumentDB TLS 証明書の更新
- Amazon DocumentDB のコンプライアンス検証
- Amazon DocumentDB の耐障害性
- Amazon DocumentDB のインフラストラクチャセキュリティ
- Amazon DocumentDB API とインターフェイス VPC エンドポイント (AWS PrivateLink) について
- Amazon DocumentDB のセキュリティベストプラクティス
- Amazon DocumentDB イベントの監査
- Amazon VPC と Amazon DocumentDB

Amazon DocumentDB と によるパスワード管理 AWS Secrets Manager

Amazon DocumentDB は Secrets Manager と統合して、クラスターのプライマリユーザーパスワードを管理します。

トピック

- Secrets Manager と Amazon DocumentDB の統合に関する制限事項
- を使用したプライマリユーザーパスワードの管理の概要 AWS Secrets Manager
- でのプライマリユーザーパスワードの Amazon DocumentDB 管理の強制 AWS Secrets Manager
- Secrets Manager によるクラスターのプライマリユーザーパスワードの管理

Secrets Manager と Amazon DocumentDB の統合に関する制限事項

Secrets Manager を使用したプライマリユーザーパスワードの管理は、以下の機能ではサポートされていません。

- Amazon DocumentDB グローバルデータベースの一部であるクラスター。
- Amazon DocumentDB クロスリージョンリードレプリカ

を使用したプライマリユーザーパスワードの管理の概要 AWS Secrets Manager

を使用すると AWS Secrets Manager、データベースパスワードを含むコード内のハードコードされた認証情報を Secrets Manager への API コールに置き換えて、プログラムでシークレットを取得できます。Secrets Manager の詳細については、AWS Secrets Manager ユーザーガイドを参照してください。

Secrets Manager にデータベースシークレットを保存すると、 AWS アカウントで料金が発生します。料金については、「AWS Secrets Manager の料金」を参照してください。

次のいずれかのオペレーションを実行するときに、Amazon DocumentDB が Amazon DocumentDB クラスターの Secrets Manager でプライマリユーザーパスワードを管理するように指定できます。

- クラスターを作成する
- クラスターの変更

Amazon DocumentDB が Secrets Manager でプライマリユーザーパスワードを管理するように指定すると、Amazon DocumentDB はパスワードを生成して Secrets Manager に保存します。シークレットを直接操作して、プライマリユーザーの認証情報を取得できます。シークレットを暗号化するカスタマーマネージドキーを指定するか、Secrets Manager が提供する KMS キーを使用することもできます。

Amazon DocumentDB はシークレットの設定を管理し、デフォルトでシークレットを 7 日ごとにローテーションします。ローテーションスケジュールなど、一部の設定を変更できます。Secrets Manager でシークレットを管理するクラスターを削除すると、シークレットとそれに関連するメタデータも削除されます。

シークレット内の認証情報を使用してクラスターに接続するには、Secrets Manager からシークレットを取得します。詳細については、「 AWS Secrets Manager ユーザーガイド」の「 シークレット AWS Secrets Managerの認証情報を使用して JDBC を使用して からシーク AWS Secrets Manager レットを取得し、SQL データベースに接続する」を参照してください。

でのプライマリユーザーパスワードの Amazon DocumentDB 管理の強制 AWS Secrets Manager

IAM 条件キーを使用して、 AWS Secrets Managerのプライマリユーザーパスワードの Amazon DocumentDB 管理を実施できます。次のポリシーでは、プライマリユーザーパスワードが Secrets Manager で Amazon DocumentDB によって管理されていない限り、ユーザーがインスタンスまたはクラスターを作成または復元することはできません。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "rds:CreateDBCluster"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "rds:ManageMasterUserPassword": false
                }
            }
        }
    ]
}
```

Secrets Manager によるクラスターのプライマリユーザーパスワードの管理

以下のアクションを実行すると、Secrets Manager でプライマリユーザーパスワードの Amazon DocumentDB 管理を設定できます。

- Amazon DocumentDB クラスターの作成
- Amazon DocumentDB クラスターの変更

Amazon DocumentDB コンソールまたは を使用して AWS CLI 、これらのアクションを実行できます。

Amazon DocumentDB におけるデータ保護

責任共有モデル、Amazon DocumentDB AWS でのデータ保護に適用されます。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。ユーザーは、このインフラストラクチャでホストされるコンテンツに対する管理を維持する責任があります。また、使用する「 AWS のサービス 」のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「Data Privacy FAQChina」を参照してください。 欧州におけるデータ保護に関する情報については、General Data Protection Regulation (GDPR) Center を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント 、 AWS IAM アイデンティティセンターまたは AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 は必須ですが、TLS 1.3 を推奨します。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。CloudTrail 証跡を使用して AWS アクティビティをキャプチャする方法については、「 AWS CloudTrail ユーザーガイド」のCloudTrail 証跡の使用」を参照してください。
- AWS 暗号化ソリューションと、 内のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度な管理されたセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-3 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-3」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報を、タグ、または [名前] フィールドなどの自由形式のテキストフィールドに含めないことを強くお勧めします。これは、コンソール、API、または SDK を使用して Amazon DocumentDB AWS CLIまたは他の AWS のサービス を使用する場合

も同様です。 AWS SDKs タグ、または名前に使用される自由記述のテキストフィールドに入力したデータは、請求または診断ログに使用される場合があります。外部サーバーに URL を提供する場合、そのサーバーへのリクエストを検証できるように、認証情報を URL に含めないことを強くお勧めします。

トピック

- クライアント側のフィールドレベルの暗号化
- Amazon DocumentDB データの保管時の暗号化
- Encrypting data in transit
- キー管理

クライアント側のフィールドレベルの暗号化

Amazon DocumentDB のクライアント側のフィールドレベル暗号化 (FLE) を使用すると、Amazon DocumentDB クラスターに転送される前に、クライアントアプリケーションの機密データを暗号化できます。機密データは、クラスターで保存および処理されるときは暗号化されたままで、取得時にはクライアントアプリケーションで復号化されます。

トピック

- 開始方法
- クライアント側 FLE でのクエリ

開始方法

Amazon DocumentDB でのクライアント側 FLE の初期設定は、暗号化キーの作成、アプリケーションへのロールの関連付け、アプリケーションの設定、暗号化オプションによる CRUD オペレーションの定義を含む 4 ステップのプロセスです。

トピック

- ステップ 1: 暗号化キーの作成
- ステップ 2: ロールとアプリケーションの関連付け
- ステップ 3: アプリケーションの設定
- ステップ 4: CRUD オペレーションの定義
- 例:クライアント側のフィールドレベル暗号化設定ファイル

ステップ 1: 暗号化キーの作成

を使用して AWS Key Management Service、機密データフィールドを暗号化および復号する対称カスタマーマネージドキーを作成し、必要な IAM 使用権限を付与します。 は、データキーの暗号化に使用されるカスタマーマネージドキー AWS KMS を保存します。カスタマーマネージドキーを に保存すると、セキュリティ体制が AWS KMS 強化されます。データキーはセカンダリキーであり、Amazon DocumentDB コレクションに保存され、Amazon DocumentDB にドキュメントを保存する前に機密フィールドを暗号化するために必要です。カスタマーマネージドキーはデータキーを暗号化し、データを暗号化および復号します。グローバルクラスターを使用する場合は、異なるサービスロールが異なるリージョンで使用できるマルチリージョンキーを作成できます。

キーの作成方法など AWS Key Management Service、 の詳細については、「 AWS Key Management Service デベロッパーガイド」の「 のAWS Key Management Service 概念」を参照してください。

ステップ 2: ロールとアプリケーションの関連付け

適切な AWS KMS アクセス許可を持つ IAM ポリシーを作成します。このポリシーは、アタッチされている IAM ID に、リソースフィールドで指定された KMS キーの暗号化と復号を許可します。アプリケーションはこの IAM ロールを引き受けて認証します AWS KMS。

ポリシーは次のようになります。

```
{ "Effect": "Allow",
"Action": ["kms:Decrypt", "kms:Encrypt"],
"Resource": "Customer Key ARN"
}
```

ステップ 3: アプリケーションの設定

でカスタマーマネージドキーを定義 AWS KMS し、それにアクセスするために必要なアクセス許可を持つ IAM ロールを作成しました。必要なパッケージをインポートします。

```
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. AWS KMS プロバイダータイプawsとして を指定し、前のステップで取得したアカウント認証情報を入力します。

```
provider = "aws"
kms_providers = {
    provider: {
        "accessKeyId": current_credentials.access_key,
        "secretAccessKey": current_credentials.secret_key
    }
}
```

2. データキーを暗号化するカスタマーマネージドキーを指定します。

```
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. MongoClient オブジェクトを設定します。

```
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. データキーを生成します。

```
data_key_id = client_encryption.create_data_key(provider,
    customer_key,
    key_alt_name = [key_alt_name])
```

5. 既存のデータキーを取得します。

```
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})
```

ステップ 4: CRUD オペレーションの定義

CRUD オペレーションを暗号化オプションで定義します。

1. 1 つのドキュメントの書き込み/読み取り/削除を行うコレクションを定義します。

```
coll = client.gameinfo.users
```

2. 明示的な暗号化 - フィールドを暗号化して以下を挿入します。

> **Note**
>
> 「key_id」または「key_alt_name」の 1 つを正確に指定する必要があります。

```
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"]
})
```

例:クライアント側のフィールドレベル暗号化設定ファイル

次の例では、各########を独自の情報に置き換えます。

```
# 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()
```

クライアント側 FLE でのクエリ

Amazon DocumentDB は、クライアント側の FLE 暗号化値を持つ等価クエリのみをサポートします。

たとえば、暗号化されたゲーマースコアが 500 に等しいドキュメントをクエリするには、クライアントは明示的な暗号化方法を使用してクエリ値を暗号化します。

```
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 }
} )
```

Amazon DocumentDB データの保管時の暗号化

> **Note**
>
> AWS KMS は、カスタマーマスターキー (CMK) という用語を AWS KMS keyおよび KMS キーに置き換えます。この概念に変更はありません。重大な変更を防ぐために、 AWS KMS は、この用語のいくつかのバリエーションを保持しています。

クラスターを作成するときに、ストレージの暗号化オプションを指定して、Amazon DocumentDB クラスター内の保存データを暗号化できます。ストレージの暗号化はクラスター全体で有効になり、プライマリインスタンスとレプリカを含むインスタンス全体に適用されます。また、クラスターのストレージボリューム、データ、インデックス、ログ、自動バックアップ、スナップショットにも適用されます。

Amazon DocumentDB は、256 ビット Advanced Encryption Standard (AES-256) を使用して、 AWS Key Management Service () に保存されている暗号化キーを使用してデータを暗号化しますAWS KMS。保存暗号化が有効な Amazon DocumentDB クラスターを使用する場合、アプリケーションロジックやクライアント接続を変更する必要はありません。Amazon DocumentDB はパフォーマンスの影響を最小限に抑えながら、データの暗号化と復号を透過的に処理します。

Amazon DocumentDB は と統合 AWS KMS され、エンベロープ暗号化と呼ばれる方法を使用してデータを保護します。Amazon DocumentDB クラスターが で暗号化されると AWS KMS、Amazon DocumentDB は KMS キーを使用して暗号化テキストデータキーを生成し、ストレージボリュームを暗号化するように AWS KMS に要求します。暗号化テキストデータキーは、定義する KMS を使用して暗号化され、暗号化されたデータおよびストレージメタデータとともに保存されます。Amazon DocumentDB は、暗号化されたデータにアクセスする必要がある場合、KMS キーを使用して暗号化テキストデータキーを復号 AWS KMS するようにリクエストし、プレーンテキストデータキーをメモリにキャッシュして、ストレージボリューム内のデータを効率的に暗号化および復号します。

Amazon DocumentDB のストレージ暗号化機能は、サポートされているすべてのインスタンスサイズとAmazon DocumentDB が利用可能なすべての AWS リージョン で使用できます。

Amazon DocumentDB クラスターで保管時の暗号化を有効にする

クラスターが または AWS Command Line Interface () を使用してプロビジョニングされている場合、Amazon DocumentDB クラスターで保管時の暗号化を有効 AWS マネジメントコンソール または無効にできますAWS CLI。コンソールを使用して作成したクラスターでは、デフォルトで保管時の暗号化が有効になっています。を使用して作成したクラスター AWS CLI では、保管時の暗号化が

デフォルトで無効になっています。したがって、保存時の暗号化は `--storage-encrypted` パラメータを使用して明示的に有効にする必要があります。いずれの場合も、クラスターの作成後は、保存時の暗号化オプションを変更することはできません。

Amazon DocumentDB は AWS KMS 、 を使用して暗号化キーを取得および管理し、これらのキーの使用方法を制御するポリシーを定義します。 AWS KMS キー識別子を指定しない場合、Amazon DocumentDB はデフォルトの AWS マネージドサービス KMS キーを使用します。Amazon DocumentDB は、 AWS リージョン 内の各 に個別の KMS キーを作成します AWS アカウント。詳細については、「AWS Key Management Service の概念」を参照してください。

独自の KMS キーの作成を開始するには、開始方法 の AWS Key Management Service デベロッパーガイド を参照してください。

> **Important**
>
> Amazon DocumentDB がサポートしているのは対称 KMS キーのみであるため、クラスターの暗号化には対称暗号化 KMS キーを使用して必要があります。Amazon DocumentDB クラスター内のデータの暗号化に非対称 KMS キー を使用しないでください。詳細については、AWS Key Management Service デベロッパーガイドの「AWS KMSの非対称キー」を参照してください。

Amazon DocumentDB がクラスターの KMS キーにアクセスできなくなった場合。たとえば、キーを所有 AWS アカウント する が停止された場合、キーが無効になった場合、キーの削除がスケジュールされた場合、Amazon DocumentDB が依存するキーポリシーまたは許可が削除された場合、クラスターは最初に `inaccessible-encryption-credentials-recoverable`ステータスに移行します。クラスターがこのステータスの間、Amazon DocumentDB はクラスターのインスタンスを停止し、クラスターとの間で読み書きを行うことはできませんが、KMS キーへのアクセスが 7 日以内に復元された場合でもクラスターを復元できます。アクセスが 7 日以内に復元されない場合、クラスターはターミナル`inaccessible-encryption-credentials`ステータスに移行します。ターミナルステータスから、クラスターは使用できなくなり、データベースの現在の状態を復元することはできません。バックアップから復元するか、元の KMS キーを使用してpoint-in-time復元のみを実行できます。Amazon DocumentDB の場合、バックアップは常に少なくとも 1 日間有効になります。

> **Note**
>
> グローバルクラスターの一部であるクラスターの動作は異なります。Amazon DocumentDB が KMS キーにアクセスできなくなったことを検出すると、グローバルクラスター内のすべ

てのクラスターはターミナルinaccessible-encryption-credentialsステータスに直接移行し、復旧可能なステータスはスキップされます。これは、グローバルクラスターの一部であるクラスターは、グローバルクラスター内の唯一のクラスターである場合にのみ停止および開始できるためです。復旧するには、スナップショットから復元するか、point-in-time 復元を実行する必要があります。元のクラスターを削除するには、まずグローバルクラスターから各クラスターを削除してから削除する必要があります。

Important

暗号化されたクラスターの KMS キーを作成した後は、そのクラスターを変更することはできません。暗号化されたクラスターを作成する前に、暗号化キーの要件を確認してください。

Using the AWS マネジメントコンソール

保管時の暗号化オプションは、クラスターの作成時に指定します。 AWS マネジメントコンソールを使用してクラスターを作成すると、保管時の暗号化がデフォルトで有効になります。クラスターの作成後に変更することはできません。

クラスターの作成時に保管時の暗号化オプションを指定するには

1. Amazon DocumentDB クラスターの作成は、開始方法 セクションで説明されています。ただし、ステップ 6 では、[クラスターの作成] を選択しないでください。
2. [Authentication (認証)] セクションの下で、[Show advanced settings (詳細設定の表示)] を選択します。
3. [Encryption-at-rest (保管時の暗号化)] セクションまで下にスクロールします。
4. 保存時の暗号化オプションを選択します。どちらのオプションを選択しても、クラスターの作成後に変更することはできません。
   - このクラスターを保管時に暗号化するには、[暗号化の有効化] を選択します。
   - このクラスターで保存されているデータを暗号化しない場合は、[暗号化の無効化] を選択します。
5. 目的のプライマリキーを選択します。Amazon DocumentDB は、 AWS Key Management Service (AWS KMS) を使用して暗号化キーを取得および管理し、これらのキーの使用方法を制御するポリシーを定義します。 AWS KMS キー識別子を指定しない場合、Amazon

DocumentDB はデフォルトの AWS マネージドサービス KMS キーを使用します。詳細については、「AWS Key Management Service の概念」を参照してください。

> **Note**
>
> 暗号化されたクラスターを作成したら、そのクラスターの KMS キーを変更することはできません。暗号化されたクラスターを作成する前に、暗号化キーの要件を確認してください。

6. 必要に応じて他のセクションを完了し、クラスターを作成します。

Using the AWS CLI

を使用して Amazon DocumentDB クラスターを暗号化するには AWS CLI、 `create-db-cluster` コマンドを実行し、 `--storage-encrypted`オプションを指定します。を使用して作成された Amazon DocumentDB クラスターは、デフォルトではストレージ暗号化を有効に AWS CLI しません。

次の例では、ストレージ暗号化を有効にした Amazon DocumentDB クラスターを作成します。

次の例では`###############`をユーザー自身の情報で置き換えます。

Example

Linux、macOS、Unix の場合:

```
aws docdb create-db-cluster \
  --db-cluster-identifier mydocdbcluster \
  --port 27017 \
  --engine docdb \
  --master-username SampleUser1 \
  --master-user-password primaryPassword \
  --storage-encrypted
```

Windows の場合:

```
aws docdb create-db-cluster ^
  --db-cluster-identifier SampleUser1 ^
  --port 27017 ^
  --engine docdb ^
```

```
  --master-username SampleUser1 ^
  --master-user-password primaryPassword ^
  --storage-encrypted
```

暗号化された Amazon DocumentDB クラスターを作成するときは、次の例のようにキー AWS KMS 識別子を指定できます。

Example

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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**
>
> 暗号化されたクラスターを作成したら、そのクラスターの KMS キーを変更することはできません。暗号化されたクラスターを作成する前に、暗号化キーの要件を確認してください。

アクセスできない暗号化状態の Amazon DocumentDB クラスターの解決

Amazon DocumentDB クラスターが暗号化された KMS キーにアクセスできない場合、Amazon DocumentDB クラスターはアクセスできない暗号化ステータスに移行します。このようなステータスは 2 つあり、復旧パスはクラスターがどのステータスにあるかによって異なります。

inaccessible-encryption-credentials-recoverable のステータス

クラスターのステータスが の場合`inaccessible-encryption-credentials-recoverable`、Amazon DocumentDB の KMS キーへのアクセスを復元し、クラスターを起動`available`することで、クラスターを に返すことができます。このステータスを解決するには、次の操作を行います。

1. KMS キーを所有 AWS アカウント する がアクティブであることを確認します。アカウントが停止されている場合は、再度アクティブ化します。
2. KMS キーが有効になっていることを確認します。詳細については、AWS Key Management Service デベロッパーガイド の「キーの有効化と無効化」を参照してください。
3. KMS キーの削除がスケジュールされているかどうかを確認します。その場合は、スケジュールされたキーの削除をキャンセルします。詳細については、「 デベロッパーガイド」の「キー削除のスケジュールとキャンセル」を参照してください。 AWS Key Management Service
4. KMS キーポリシーと Amazon DocumentDB が依存する許可が、Amazon DocumentDB にキーの使用を許可していることを確認します。
5. KMS キーへのアクセスが復元されたら、 AWS マネジメントコンソール または `start-db-cluster` AWS CLI コマンドを実行してクラスターを起動します。

   Example

   Linux、macOS、Unix の場合:

   ```
   aws docdb start-db-cluster \
     --db-cluster-identifier example-cluster
   ```

   Windows の場合:

   ```
   aws docdb start-db-cluster ^
     --db-cluster-identifier example-cluster
   ```

> **Important**
>
> KMS キーへのアクセスが 7 日以内に復元されない場合、クラスターはターミナルinaccessible-encryption-credentialsステータスに移行し、開始することはできません。

`inaccessible-encryption-credentials` のステータス

inaccessible-encryption-credentials ステータスはターミナルです。クラスターを起動できず、データベースの実行状態を復元できません。データを復元するには、スナップショットから復元するか、新しいクラスターへのpoint-in-time復元を実行します。復元を実行するには、元の KMS キーにアクセスできる必要があります。KMS キーが削除された場合、データは復元できません。

詳細については、「クラススナップショットからの復元」および「特定時点への復元」を参照してください。

> **Note**
>
> グローバルクラスターの一部であるクラスターは、グローバルクラスターの一部であるクラスターがグローバルクラスター内の唯一のクラスターである場合にのみ停止および開始できるため、KMS キーへのアクセスが失われると inaccessible-encryption-credentialsステータスに直接移行します。このステータスのクラスターを削除するには、まずグローバルクラスターから各クラスターを削除してから、クラスターを個別に削除します。

削除保護が有効になっているため、 inaccessible-encryption-credentialsステータスのクラスターを削除できない場合は、削除を再試行 AWS CLI する前に を使用して削除保護をオフにします。

Example

Linux、macOS、Unix の場合:

```
aws docdb modify-db-cluster \
  --db-cluster-identifier example-cluster \
  --no-deletion-protection
```

Windows の場合:

```
aws docdb modify-db-cluster ^
  --db-cluster-identifier example-cluster ^
  --no-deletion-protection
```

その後、 delete-db-cluster コマンドを使用してクラスターを削除できます。

Example

Linux、macOS、Unix の場合:

```
aws docdb delete-db-cluster \
  --db-cluster-identifier example-cluster \
  --skip-final-snapshot
```

Windows の場合:

```
aws docdb delete-db-cluster ^
  --db-cluster-identifier example-cluster ^
  --skip-final-snapshot
```

上記のコマンドを実行してもクラスターが削除しない場合は、 にお問い合わせくださいAWS サポート。

Amazon DocumentDB 暗号化クラスターに関する制限事項

Amazon DocumentDB の暗号化されたクラスターには、以下の制限事項があります。

- Amazon DocumentDB クラスターのストレージの暗号化を有効または無効にできるのは、クラスターの作成後ではなく作成時のみです。ただし、暗号化されていないクラスターのスナップショットを作成し、暗号化されていないスナップショットを新しいクラスターとして復元する際に保存時の暗号化オプションを指定します。

  詳細については、以下の各トピックを参照してください。
  - 手動クラスタースナップショットの作成
  - クラスタースナップショットからの復元
  - Amazon DocumentDB クラスタースナップショットのコピー
- ストレージの暗号化を有効にした Amazon DocumentDB クラスターを変更して暗号化を無効にすることはできません。

- Amazon DocumentDB クラスターのすべてのインスタンス、自動バックアップ、スナップショット、インデックスは、同じ KMS キーで暗号化されます。

Encrypting data in transit

TLS (Transport Layer Security) を使用して、アプリケーションと Amazon DocumentDB クラスター間の接続を暗号化できます。デフォルトでは、転送時の暗号化は、新しく作成された Amazon DocumentDB クラスターに対して有効になっています。クラスターの作成時、または後で無効にすることもできます。転送中の暗号化が有効になっている場合、クラスターに接続するには TLS を使用したセキュアな接続が必要です。TLS を使用した Amazon DocumentDB への接続の詳細については、「プログラムによる Amazon DocumentDB への接続」を参照してください。

Amazon DocumentDB クラスター TLS 設定の管理

Amazon DocumentDB クラスターの転送中の暗号化は、クラスターパラメータグループ の TLS パラメータを使用して管理されます。Amazon DocumentDB クラスターの TLS 設定は、 AWS マネジメントコンソール または AWS Command Line Interface () を使用して管理できますAWS CLI。現在の TLS 設定を確認して変更する方法の説明については、次のセクションを参照してください。

Using the AWS マネジメントコンソール

コンソールを使用して TLS 向けにパフォーマンス管理タスクは、パラメータグループの識別、TLS 値の検証や必要な変更の追加などには、次の手順を使用します。

> **Note**
>
> クラスターの作成時に別の指定がない限り、クラスターはデフォルトのクラスターパラメータグループを使用して作成されます。`default` クラスターパラメータグループのパラメータは変更できません (`tls` の有効化/無効化など)。そのため、クラスターが `default` クラスターパラメータグループを使用している場合は、デフォルト以外のクラスターパラメータグループを使用するようにクラスターを変更する必要があります。まず、カスタムクラスターパラメータグループを作成する必要があります。詳細については、「Amazon DocumentDB クラスターパラメータグループを作成する」を参照してください。

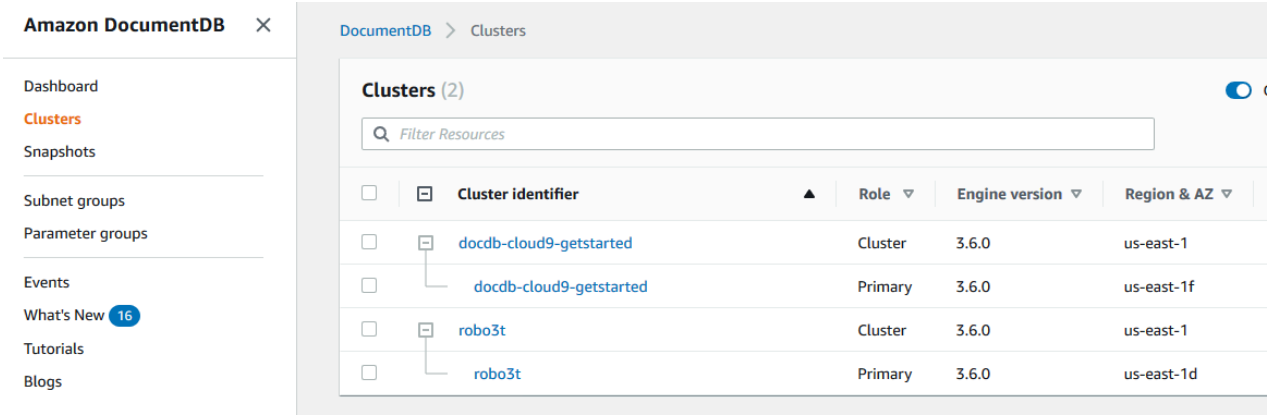
1. クラスターが使用しているクラスターパラメータグループを確認します。

a. Amazon DocumentDB コンソールを次の場所で開きます。https://console.aws.amazon.com/docdb

b. ナビゲーションペインで クラスター を選択します。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

c. [クラスター] ナビゲーションボックスの [クラスター識別子] 列には、クラスターとインスタンスの両方が表示されますのでご注意ください。インスタンスはクラスターの下にリストされます。以下のスクリーンショットを参照してください。



d. 関心があるクラスターを選択します。

e. [設定] タブを選択して、[クラスターの詳細] の一番下までスクロールし、[クラスターパラメータグループ] を見つけます。クラスターパラメータグループの名前を書き留めます。

クラスターのパラメータグループの名前が `default` (例えば、`default.docdb3.6` など) の場合、続行する前にカスタムのクラスターパラメータグループを作成して、それをクラスターのパラメータグループにする必要があります。詳細については次を参照してください:

1. Amazon DocumentDB クラスターパラメータグループを作成する – 使用できるカスタムのクラスターパラメータグループがない場合、これを作成します。

2. Amazon DocumentDB クラスターの変更 - カスタムクラスターパラメータグループを使用するようにクラスターを変更します。

2. **tls** クラスターパラメータの現在の値を確認します。

   a. Amazon DocumentDB コンソールを次の場所で開きます。https://console.aws.amazon.com/docdb

   b. ナビゲーションペインで、[パラメータグループ] を選択します。

   c. クラスターパラメータグループの一覧から、使用するクラスターパラメータグループの名前を選択します。

   d. [クラスターパラメータ] セクションを見つけます。クラスターパラメータの一覧で、tls クラスターパラメータ行を見つけます。この時点では、次の 4 つの列が重要です。

      - クラスターパラメータ名 - クラスターパラメータの名前 TLS を管理するには、tls クラスターパラメータを選びます。
      - 値 - 各クラスターパラメータの現在の値
      - 使用できる値 - クラスターパラメータに適用できる値の一覧。
      - 適用タイプ - [静的] または [動的] のいずれか。静的クラスターパラメータへの変更は、インスタンスが再起動されるときにのみ適用されます。動的パラメータへの変更は、すぐに適用されるか、またはインスタンスが再起動されるときに適用されます。

3. **tls** クラスターパラメータの値を設定します。

   tls の値が必要な値ではない場合、このクラスターパラメータグループの値を変更します。tls クラスターパラメータの値を変更するには、前のセクションに続いて次の手順を行います。

   a. クラスターパラメータ名 (tls) の左側のボタンを選択します。

   b. [編集] を選択します。

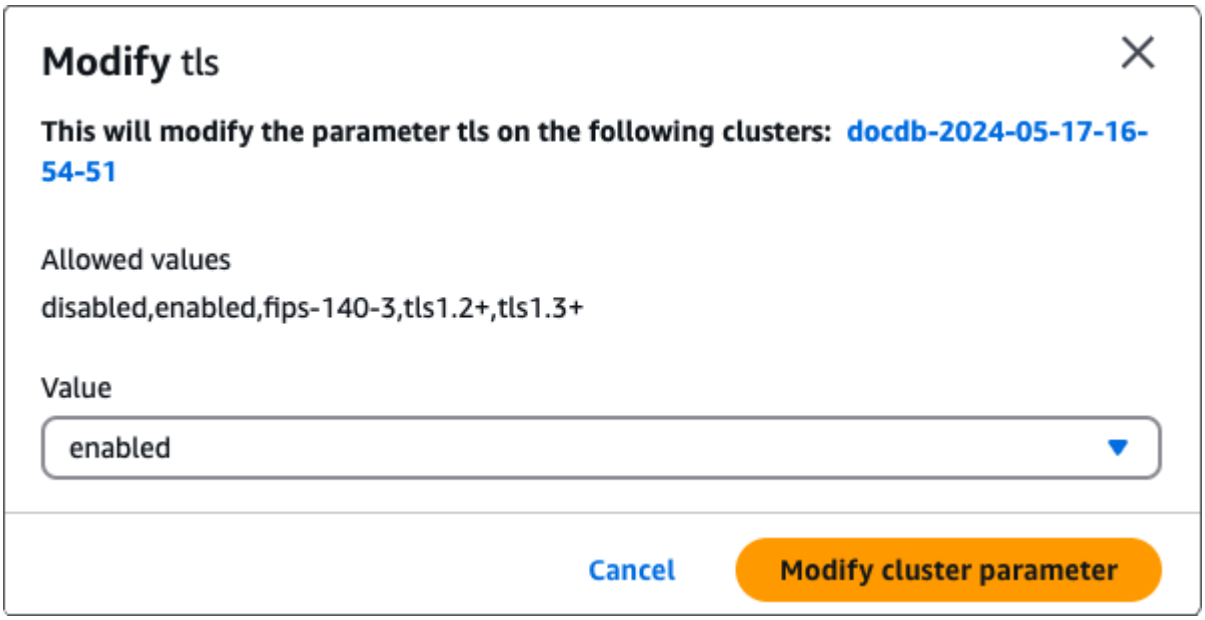
   c. tls の値を変更するには、[**tls** の変更] ダイアログボックスで、ドロップダウンリストからクラスターパラメータに必要な値を選択します。

      次の値を指定できます。

      - disabled: TLS を無効にします
      - enabled — TLS を有効にします。Amazon DocumentDB 5.0 以前では、TLS 1.0 ~ 1.3 を有効にしますが、TLS 1.2 以降を使用します。マイナーバージョン 5.0.1、Amazon

DocumentDB 8.0 以降では、TLS 1.2 以降のみが Amazon DocumentDB 5.0 でサポートされています。

- fips-140-3: FIPS で TLS を有効にします。クラスターは、連邦情報処理標準 (FIPS) 出版物 140-3 の要件に従った安全な接続のみを受け付けます。これは、ca-central-1、us-west-2、us-east-1、us-east-2、us-gov-east-1、us-gov-west-1 の各リージョンの Amazon DocumentDB 5.0 (エンジンバージョン 3.0.3727) クラスターからのみサポートされています。
- tls1.2+: TLS バージョン 1.2 以降を有効にします。これは、Amazon DocumentDB 4.0 (エンジンバージョン 2.0.10980) および Amazon DocumentDB (エンジンバージョン 3.0.11051) 以降でのみサポートされています。
- tls1.3+: TLS バージョン 1.3 以降を有効にします。これは、Amazon DocumentDB 4.0 (エンジンバージョン 2.0.10980) および Amazon DocumentDB (エンジンバージョン 3.0.11051) 以降でのみサポートされています。



d. [Modify cluster parameter (クラスターパラメータの変更)] を選択します。各クラスターインスタンスが再起動されると、変更が適用されます。

4. Amazon DocumentDB インスタンスを再起動します。

   変更がクラスター内のすべてのインスタンスに適用されるように、クラスターの各インスタンスを再起動します。

   a. Amazon DocumentDB コンソールを次の場所で開きます。https://console.aws.amazon.com/docdb

b. ナビゲーションペインで、[インスタンス] を選択してください。

c. 再起動するインスタンスを指定するには、インスタンスのリストでインスタンスを見つけ、その名前の左側にあるボタンを選択します。

d. [アクション]、[再起動] の順に選択します。[再起動] を選択して、リブートすることを確認します。

Using the AWS CLI

AWS CLIを使用して TLS 向けにパフォーマンス管理タスクは、パラメータグループの識別、TLS 値の検証や必要な変更の追加などには、次の手順を使用します。

> **Note**
>
> クラスターの作成時に別の指定がない限り、クラスターはデフォルトのクラスターパラメータグループを使用して作成されます。`default` クラスターパラメータグループのパラメータは変更できません (`tls` の有効化/無効化など)。そのため、クラスターが `default` クラスターパラメータグループを使用している場合は、デフォルト以外のクラスターパラメータグループを使用するようにクラスターを変更する必要があります。まず、カスタムクラスターパラメータグループを作成する必要があります。詳細については、「Amazon DocumentDB クラスターパラメータグループを作成する」を参照してください。

1. クラスターが使用しているクラスターパラメータグループを確認します。

   次のオプションで `describe-db-clusters` コマンドを実行します。

   - `--db-cluster-identifier`
   - `--query`

   次の例では、各 *############* をクラスターの情報に置き換えます。

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

   このオペレーションによる出力は、次のようになります (JSON 形式)。

```
[
    [
        "mydocdbcluster",
        "myparametergroup"
    ]
]
```

クラスターのパラメータグループの名前が default (例えば、default.docdb3.6 など) の場合、続行する前にカスタムのクラスターパラメータグループを用意して、それをクラスターのパラメータグループにする必要があります。詳細については、以下の各トピックを参照してください。

1. Amazon DocumentDB クラスターパラメータグループを作成する – 使用できるカスタムのクラスターパラメータグループがない場合、これを作成します。
2. Amazon DocumentDB クラスターの変更 - カスタムクラスターパラメータグループを使用するようにクラスターを変更します。

2. **tls** クラスターパラメータの現在の値を確認します。

   このクラスターパラメータグループに関する詳細情報を取得するには、次のオプションで describe-db-cluster-parameters コマンドを実行します。

   - --db-cluster-parameter-group-name
   - --query

     出力を目的のフィールドのみに制限します:
     ParameterName、ParameterValue、AllowedValues、および ApplyType。

   次の例では、各 ############## をクラスターの情報に置き換えます。

```
aws docdb describe-db-cluster-parameters \
  --db-cluster-parameter-group-name myparametergroup \
  --query 'Parameters[*].[ParameterName,ParameterValue,AllowedValues,ApplyType]'
```

   このオペレーションによる出力は、次のようになります (JSON 形式)。

```
[
    [
        "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. **tls** クラスターパラメータの値を設定します。

   tls の値が必要な値ではない場合、このクラスターパラメータグループの値を変更します。tls クラスターパラメータの値を変更するには、次のオプションで modify-db-cluster-parameter-group コマンドを実行します。

   - --db-cluster-parameter-group-name — 必須。変更するクラスターパラメータグループの名前。これを default.* クラスターパラメータグループにすることはできません。
   - --parameters — 必須。変更するクラスターパラメータグループのパラメーター一覧。
     - ParameterName — 必須。変更するクラスターパラメータの名前。
     - ParameterValue — 必須。このクラスターパラメータの新しい値。クラスターパラメータの AllowedValues のいずれかにする必要があります。
       - enabled — クラスターは TLS を使用した安全な接続を受け入れます。Amazon DocumentDB 5.0 以前では、TLS 1.0 ~ 1.3 を有効にしますが、TLS 1.2 以降を使用します。マイナーバージョン 5.0.1、Amazon DocumentDB 8.0 以降では、TLS 1.2 以降のみが Amazon DocumentDB 5.0 でサポートされています。
       - disabled - クラスターは TLS を使用したセキュアな接続を受け入れません。
       - fips-140-3 - クラスターは、連邦情報処理標準 (FIPS) 出版物 140-3 の要件に従った安全な接続のみを受け付けます。これは、ca-central-1、us-west-2、us-east-1、us-

east-2、us-gov-east-1、us-gov-west-1 の各リージョンの Amazon DocumentDB 5.0 (エンジンバージョン 3.0.3727) クラスターからのみサポートされています。

- `tls1.2+` — クラスターは TLS バージョン 1.2 以降を使用した安全な接続を受け入れます。これは、Amazon DocumentDB 4.0 (エンジンバージョン 2.0.10980) および Amazon DocumentDB 5.0 (エンジンバージョン 3.0.11051) 以降でのみサポートされています。
- `tls1.3+` — クラスターは TLS バージョン 1.3 以降を使用した安全な接続を受け入れます。これは、Amazon DocumentDB 4.0 (エンジンバージョン 2.0.10980) および Amazon DocumentDB 5.0 (エンジンバージョン 3.0.11051) 以降でのみサポートされています。

- `ApplyMethod` - この変更が適用される場合。`tle` のような静的クラスターパラメータでは、この値を `pending-reboot` にする必要があります。
  - `pending-reboot` - インスタンスの再起動後のみ、変更がインスタンスに適用されます。クラスターのすべてのインスタンスでこの変更を適用するには、各クラスターインスタンスを個別に再起動する必要があります。

次の例では*###############*をユーザー自身の情報で置き換えます。

次のコードにより `tls` を 無効化 し 、各インスタンスの再起動時に変更が適用されるようにします。

```
aws docdb modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=tls,ParameterValue=disabled,ApplyMethod=pending-
reboot"
```

次のコードは を有効にし`tls`、再起動時に各インスタンスに変更を適用します。

```
aws docdb modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=tls,ParameterValue=enabled,ApplyMethod=pending-
reboot"
```

次のコードは、`fips-140-3` で TLS を 有効化 し、各インスタンスの再起動時に変更が適用されるようにします。

```
aws docdb modify-db-cluster-parameter-group \
```

```
  --db-cluster-parameter-group-name myparametergroup2 \
  --parameters "ParameterName=tls,ParameterValue=fips-140-3,ApplyMethod=pending-
reboot"
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "DBClusterParameterGroupName": "myparametergroup"
}
```

4. お客様の Amazon DocumentDB インスタンスを再起動

変更がクラスター内のすべてのインスタンスに適用されるように、クラスターの各インスタンスを再起動します。Amazon DocumentDB インスタンスを再起動するには、次のオプションで reboot-db-instance コマンドを実行します。

- `--db-instance-identifier`

次のコードは、インスタンス mydocdbinstance を再起動します。

次の例では*###############*をユーザー自身の情報で置き換えます。

Example

Linux、macOS、Unix の場合:

```
aws docdb reboot-db-instance \
  --db-instance-identifier mydocdbinstance
```

Windows の場合:

```
aws docdb reboot-db-instance ^
  --db-instance-identifier mydocdbinstance
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "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"
    }
}
```

インスタンスが再起動するまでには数分かかります。インスタンスを使用できるのは、そのステータスが [available] である場合のみです。コンソールまたは AWS CLIを使用して、イン

スタンスのステータスをモニタリングできます。詳細については、「Amazon DocumentDB インスタンスのステータスのモニタリング」を参照してください。

キー管理

Amazon DocumentDB は AWS Key Management Service (AWS KMS) を使用して暗号化キーを取得および管理します。安全で可用性の高いハードウェアとソフトウェア AWS KMS を組み合わせて、クラウド向けにスケーリングされたキー管理システムを提供します。を使用すると AWS KMS、暗号化キーを作成し、これらのキーの使用方法を制御するポリシーを定義できます。 は AWS KMS をサポートしているため AWS CloudTrail、キーの使用状況を監査して、キーが適切に使用されていることを確認できます。

KMS キーは、Amazon DocumentDB、および Amazon Simple Storage Service (Amazon S3)、Amazon Relational Database Service (Amazon RDS)、Amazon Elastic Block Store (Amazon EBS)、Amazon Redshift などのサポートされている AWS サービスと組み合わせて使用できます。がサポートするサービスのリストについては AWS KMS、「 AWS Key Management Service デベロッパーガイド」の「 AWS のサービスが を使用する方法 AWS KMS」を参照してください。詳細については AWS KMS、「 とは」を参照してください AWS Key Management Service。

Amazon DocumentDB の Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を 認証し (サインインする)、誰に Amazon DocumentDB リソースの使用を 許可する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで使用できる AWS のサービス です。

トピック

- オーディエンス
- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- Amazon DocumentDB と IAM との連携方法
- Amazon DocumentDB のアイデンティティベースのポリシー例
- Amazon DocumentDB ID とアクセスのトラブルシューティング
- Amazon DocumentDB リソースへのアクセス許可の管理

- Amazon DocumentDB のアイデンティティベースのポリシー (IAM ポリシー) の使用
- AWS Amazon DocumentDB の マネージドポリシー
- Amazon DocumentDB API のアクセス許可: アクション、リソース、条件リファレンス

オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、ロールによって異なります。

- サービスユーザー - 機能にアクセスできない場合は、管理者にアクセス許可をリクエストします (「Amazon DocumentDB ID とアクセスのトラブルシューティング」を参照)。
- サービス管理者 - ユーザーアクセスを決定し、アクセス許可リクエストを送信します (「Amazon DocumentDB と IAM との連携方法」を参照)
- IAM 管理者 - アクセスを管理するためのポリシーを作成します (「Amazon DocumentDB のアイデンティティベースのポリシー例」を参照)

アイデンティティを使用した認証

認証は、ID 認証情報 AWS を使用して にサインインする方法です。、IAM ユーザー AWS アカウントのルートユーザー、または IAM ロールを引き受けることで認証される必要があります。

(AWS IAM アイデンティティセンター IAM Identity Center)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーティッド ID としてサインインできます。サインインの詳細については、「AWS サインイン ユーザーガイド」の「AWS アカウントにサインインする方法」を参照してください。

プログラムによるアクセスの場合、 は SDK と CLI AWS を提供してリクエストを暗号化して署名します。詳細については、「IAM ユーザーガイド」の「API リクエストに対するAWS 署名バージョン 4」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、まず、すべての AWS のサービス および リソースへの完全なアクセス権を持つ AWS アカウント root ユーザーと呼ばれる 1 つのサインインアイデンティティから始めます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザー認証情報を必要とするタスクについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーテッドアイデンティティ

ベストプラクティスとして、人間のユーザーが一時的な認証情報 AWS のサービス を使用して にアクセスするには、ID プロバイダーとのフェデレーションを使用する必要があります。

フェデレーティッド ID は、エンタープライズディレクトリ、ウェブ ID プロバイダー、または ID ソースの認証情報 AWS のサービス を使用して Directory Service にアクセスするユーザーです。フェデレーテッドアイデンティティは、一時的な認証情報を提供するロールを引き受けます。

アクセスを一元管理する場合は、 AWS IAM アイデンティティセンターをお勧めします。詳細については、「AWS IAM アイデンティティセンター ユーザーガイド」の「IAM アイデンティティセンターとは」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、特定の個人やアプリケーションに対する特定のアクセス許可を持つアイデンティティです。長期認証情報を持つ IAM ユーザーの代わりに一時的な認証情報を使用することをお勧めします。詳細については、IAM ユーザーガイドの「ID プロバイダーとのフェデレーションを使用して にアクセスする必要がある AWS」を参照してください。

IAM グループは、IAM ユーザーの集合を指定し、大量のユーザーに対するアクセス許可の管理を容易にします。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可を持つアイデンティであり、一時的な認証情報を提供します。ユーザーから IAM ロール (コンソール) に切り替えるか、 または API オペレーションを呼び出すことで、ロールを引き受けることができます。 AWS CLI AWS 詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーのアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行するアプリケーションに役立ちます。詳細については、IAM ユーザーガイド の IAM でのクロスアカウントリソースアクセス を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられている場合のアクセス許可を定義します。 は、プリンシパルがリクエストを行うときにこれらのポリシー AWS を評価します。ほとんど

のポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は、ポリシーを使用して、どのプリンシパルがどのリソースに対して、どのような条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成してロールに追加し、このロールをユーザーが引き受けられるようにします。IAM ポリシーは、オペレーションの実行方法を問わず、アクセス許可を定義します。

アイデンティティベースのポリシー

アイデンティティベースのポリシーは、アイデンティティ (ユーザー、グループ、またはロール) にアタッチできる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、アイデンティティがどのリソースに対してどのような条件下でどのようなアクションを実行できるかを制御します。アイデンティティベースポリシーの作成方法については、IAM ユーザーガイド の カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する を参照してください。

アイデンティティベースのポリシーは、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。管理ポリシーとインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーのいずれかを選択する」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。例としては、IAM ロール信頼ポリシーや Amazon S3 バケットポリシーなどがあります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

その他のポリシータイプ

AWS は、より一般的なポリシータイプによって付与されるアクセス許可の最大数を設定できる追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与することのできるアクセス許可の数の上限を設定します。詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可境界」を参照してください。

- サービスコントロールポリシー (SCP) - AWS Organizations内の組織または組織単位の最大のアクセス許可を指定します。詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー」を参照してください。
- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、「AWS Organizations ユーザーガイド」の「リソースコントロールポリシー (RCP)」を参照してください。
- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成されるアクセス許可を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、「IAM ユーザーガイド」の「ポリシー評価ロジック」を参照してください。

Amazon DocumentDB と IAM との連携方法

IAM を使用して Amazon DocumentDB へのアクセスを管理する前に、Amazon DocumentDB で使用できる IAM 機能について理解しておく必要があります。

Amazon DocumentDB で使用できる IAM の機能

IAM 機能	インスタンスベースのクラスター	Elastic クラスター
アイデンティティベースのポリシー	はい	はい
リソースベースのポリシー	いいえ	いいえ
ポリシーアクション	はい	はい
ポリシーリソース	はい	はい
ポリシー条件キー (サービス固有)	はい	はい

IAM 機能	インスタンスベースのクラスター	Elastic クラスター
ACL	いいえ	いいえ
ABAC (ポリシー内のタグ)	部分的	はい
一時的な認証情報	はい	はい
プリンシパルアクセス権限	はい	はい
サービスロール	はい	はい
サービスリンクロール	いいえ	はい

Amazon DocumentDB およびその他の AWS のサービスがほとんどの IAM 機能と連携する方法の概要については、IAM ユーザーガイドのAWS 「IAM と連携する のサービス」を参照してください。

Amazon DocumentDB のアイデンティティベースの ポリシー

アイデンティティベースのポリシーのサポート: あり

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースポリシーの作成方法については、「IAM ユーザーガイド」の「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。JSON ポリシーで使用できるすべての要素について学ぶには、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

Amazon DocumentDB のアイデンティティベースのポリシー例

Amazon DocumentDB のアイデンティティベースのポリシーの例は、「Amazon DocumentDB のアイデンティティベースのポリシー例」でご確認ください。

Amazon DocumentDB 内のリソースベースのポリシー

リソースベースのポリシーのサポート: なし

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシー や Amazon S3 バケットポリシー があげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスをコントロールできます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーで、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、全体のアカウント、または別のアカウントの IAM エンティティを、リソースベースのポリシーのプリンシパルとして指定します。詳細については、IAM ユーザーガイドのIAM でのクロスアカウントリソースアクセスを参照してください。

Amazon DocumentDB のポリシーアクション

ポリシーアクションのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの `Action` 要素にはポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。このアクションは関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

> **Note**
>
> 特定の管理機能では、Amazon DocumentDB は Amazon Relational Database Service (Amazon RDS) と共有の運用テクノロジーを使用します。
> RDS アクションの一覧については、「サービス認可リファレンス」の「Amazon Relational Database Service で定義されるアクション」を参照してください。
> Amazon DocumentDB エラスティッククラスターのポリシーアクションを確認するには、「サービス認可リファレンス」の「Amazon DocumentDB Elastic クラスターで定義されるアクション」を参照してください。

Amazon DocumentDB のポリシーアクションでは、アクションの前に以下のプレフィックスが使用されます。

```
aws
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "aws:action1",
      "aws:action2"
         ]
```

Amazon DocumentDB のアイデンティティベースのポリシーの例は、「Amazon DocumentDB のアイデンティティベースのポリシー例」でご確認ください。

Amazon DocumentDB のポリシーリソース

ポリシーリソースのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

`Resource` JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。リソースレベルのアクセス許可をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

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

> **Note**
>
> 特定の管理機能では、Amazon DocumentDB は Amazon Relational Database Service (Amazon RDS) と共有の運用テクノロジーを使用します。
> RSD リソースのタイプと ARN のリストを確認するには、「サービス認可リファレンス」の「Amazon Relational Database Service で定義されるリソース」を参照してください。各リソースの ARN を指定できるアクションについては、「Amazon Relational Database Service で定義されるアクション」を参照してください。

Amazon DocumentDB エラスティッククラスターのリソースタイプを表示するには、「サービス認可リファレンス」のAmazon DocumentDB エラスティッククラスターで定義されるリソースタイプ」を参照してください。

Amazon DocumentDB のアイデンティティベースのポリシーの例は、「Amazon DocumentDB のアイデンティティベースのポリシー例」でご確認ください。

Amazon DocumentDB のポリシー条件キー

サービス固有のポリシー条件キーのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

`Condition` 要素は、定義された基準に基づいてステートメントが実行される時期を指定します。イコールや未満などの条件演算子を使用して条件式を作成して、ポリシーの条件とリクエスト内の値を一致させることができます。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「グローバル条件コンテキストキー」を参照してください。

> **Note**
>
> 特定の管理機能では、Amazon DocumentDB は Amazon Relational Database Service (Amazon RDS) と共有の運用テクノロジーを使用します。
> RDS の条件キーのリストを確認するには、「サービス認可リファレンス」の「Amazon Relational Database Service の条件キー」を参照してください。条件キーを使用できるアクションとリソースについては、「Amazon Relational Database Service で定義するアクション」を参照してください。
> Amazon DocumentDB エラスティッククラスターの条件キーを表示するには、「サービス認可リファレンス」のAmazon DocumentDB エラスティッククラスターの条件キー」を参照してください。

Amazon DocumentDB のアイデンティティベースのポリシーの例は、「Amazon DocumentDB のアイデンティティベースのポリシー例」でご確認ください。

Amazon DocumentDB の ACL

ACL のサポート: なし

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするためのアクセス許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon DocumentDB の ABAC

> **Note**
>
> ABAC はインスタンスベースのクラスターでは部分的にしかサポートされていませんが、Elastic クラスターではサポートされています。

属性ベースのアクセス制御 (ABAC) は、タグと呼ばれる属性に基づいてアクセス許可を定義する認可戦略です。IAM エンティティと AWS リソースにタグをアタッチし、プリンシパルのタグがリソースのタグと一致するときにオペレーションを許可するように ABAC ポリシーを設計できます。

タグに基づいてアクセスを管理するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値はありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、「IAM ユーザーガイド」の「ABAC 認可でアクセス許可を定義する」を参照してください。ABAC をセットアップする手順を説明するチュートリアルについては、「IAM ユーザーガイド」の「属性ベースのアクセス制御 (ABAC) を使用する」を参照してください。

Amazon DocumentDB で一時的な認証情報を使用する

一時的な認証情報のサポート: あり

一時的な認証情報は、 AWS リソースへの短期的なアクセスを提供し、フェデレーションまたはロールを切り替えるときに自動的に作成されます。 AWS では、長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成することをお勧めします。詳細については、「IAM ユーザーガイド」の「IAM の一時的な認証情報」および「AWS のサービス と IAM との連携」を参照してください。

Amazon DocumentDB のクロスサービスプリンシパル許可

転送アクセスセッション (FAS) のサポート: あり

転送アクセスセッション (FAS) は、 を呼び出すプリンシパルのアクセス許可と AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストをリクエストする を使用します。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

Amazon DocumentDB のサービスロール

サービスロールのサポート: あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、IAM ユーザーガイドの AWS のサービスに許可を委任するロールを作成するを参照してください。

> **Warning**
>
> サービスロールの許可を変更すると、Amazon DocumentDB の機能が破損する可能性があります。Amazon DocumentDB が指示する場合以外は、サービスロールを編集しないでください。

Amazon DocumentDB のサービスにリンクされたロール

> **Note**
>
> サービスにリンクされたロールはインスタンスベースのクラスターではサポートされていませんが、Elastic クラスターではサポートされています。

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示できますが、編集することはできません。

サービスにリンクされたロールの作成または管理の詳細については、「IAM と提携するAWS のサービス」を参照してください。表の「サービスリンクロール」列に Yes と記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

Amazon DocumentDB のアイデンティティベースのポリシー例

デフォルトでは、ユーザーおよびロールには Amazon DocumentDB リソースを作成または変更する許可はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。

これらのサンプルの JSON ポリシードキュメントを使用して IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーを作成する (コンソール)」を参照してください。

Amazon DocumentDB が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、「サービス認可リファレンス」の「Amazon Relational Database Service のアクション、リソース、および条件キー」を参照してください。

> **Note**
>
> 特定の管理機能では、Amazon DocumentDB は Amazon Relational Database Service (Amazon RDS) と共有の運用テクノロジーを使用します。
> Amazon DocumentDB エラスティッククラスターのポリシーアクションについては、「サービス認可リファレンス」の「Amazon DocumentDB エラスティッククラスターのアクション、リソース、および条件キー」を参照してください。

トピック

- ポリシーに関するベストプラクティス
- Amazon DocumentDB コンソールを使用する
- 自分の権限の表示をユーザーに許可する

ポリシーに関するベストプラクティス

アイデンティティベースのポリシーは、ユーザーのアカウント内で誰かが Amazon DocumentDB リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションでは、 AWS

アカウントに費用が発生する場合があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行 – ユーザーとワークロードにアクセス許可の付与を開始するには、多くの一般的なユースケースにアクセス許可を付与するAWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義することで、アクセス許可をさらに減らすことをお勧めします。詳細については、IAM ユーザーガイド の AWS マネージドポリシー または ジョブ機能のAWS マネージドポリシー を参照してください。
- 最小特権を適用する – IAM ポリシーでアクセス許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、IAM ユーザーガイド の IAM でのポリシーとアクセス許可 を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する – ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、サービスアクションが などの特定の を通じて使用されている場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます CloudFormation。詳細については、IAM ユーザーガイド の IAM JSON ポリシー要素:条件 を参照してください。
- IAM アクセスアナライザー を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する – IAM アクセスアナライザー は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、IAM ユーザーガイド の IAM Access Analyzer でポリシーを検証する を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、MFA をオンにしてセキュリティを強化します。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、IAM ユーザーガイド の MFA を使用した安全な API アクセス を参照してください。

IAM でのベストプラクティスの詳細については、IAM ユーザーガイド の IAM でのセキュリティのベストプラクティス を参照してください。

Amazon DocumentDB コンソールを使用する

Amazon DocumentDB (MongoDB 互換) コンソールにアクセスするには、最小限のアクセス許可のセットが必要です。これらのアクセス許可により、 の Amazon DocumentDB リソースの詳細を一覧表示および表示できます AWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーとロールが引き続き Amazon DocumentDB コンソールを使用できるようにするには、エンティティに Amazon DocumentDB *ConsoleAccess*または *ReadOnly* AWS 管理ポリシーもアタッチします。詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

自分の権限の表示をユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

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

Amazon DocumentDB ID とアクセスのトラブルシューティング

以下の情報を使用すると、Amazon DocumentDB および IAM での作業中に直面する可能性がある一般的な問題の診断や修正に役立ちます。

トピック

- Amazon DocumentDB でアクションを実行することを認可されていません
- iam:PassRole を実行する権限がありません
- 自分の 以外のユーザーに Amazon DocumentDB リソース AWS アカウント へのアクセスを許可したい

Amazon DocumentDB でアクションを実行することを認可されていません

アクションを実行する権限がないというエラーが表示された場合は、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用して、ある *my-example-widget* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要な aws:*GetWidget* アクセス許可を持っていない場合に発生するものです。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 aws:GetWidget on resource: my-example-widget
```

この場合、aws:*GetWidget* アクションを使用して *my-example-widget* リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

iam:PassRole を実行する権限がありません

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して Amazon DocumentDB にロールを渡せるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡すアクセス許可が必要です。

以下の例のエラーは、 marymajor という名前の IAM ユーザーがコンソールを使用して Amazon DocumentDB でアクションを実行しようとする際に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与されたアクセス許可が必要です。Mary には、ロールをサービスに渡すアクセス許可がありません。

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

この場合、Mary のポリシーを更新してメアリーに iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

自分の 以外のユーザーに Amazon DocumentDB リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- Amazon DocumentDB がこれらの機能をサポートしているかどうかは「Amazon DocumentDB と IAM との連携方法」を参照してください。

- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイド の 外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可 を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

Amazon DocumentDB リソースへのアクセス許可の管理

すべての AWS リソースは によって所有され AWS アカウント、リソースを作成またはアクセスするアクセス許可はアクセス許可ポリシーによって管理されます。アカウント管理者は、アクセス許可ポリシーを IAM ID (ユーザー、グループ、ロール) にアタッチできます。一部のサービス (など AWS Lambda) では、アクセス許可ポリシーをリソースにアタッチすることもできます。

> **Note**
>
> アカウント管理者 (または管理者ユーザー) は、管理者アクセス許可を持つユーザーです。詳細については、「IAM ユーザーガイド」の「IAM のベストプラクティス」を参照してください。

トピック

- Amazon DocumentDB リソースとオペレーション
- リソース所有権について
- リソースへのアクセスの管理
- ポリシー要素の指定 : アクション、効果、リソース、プリンシパル
- ポリシーでの条件の指定

Amazon DocumentDB リソースとオペレーション

Amazon DocumentDB では、プライマリリソースは クラスター です。Amazon DocumentDB は、インスタンス、パラメータグループ、イベントサブスクリプション などのプライマリリソースで使用できる他のリソースをサポートします。これらのリソースはサブリソースと呼ばれます。

これらのリソースとサブリソースには、次の表に示すとおり、一意の Amazon リソースネーム (ARN) が関連付けられています。

リソースタイプ	ARN 形式
クラスター	arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*
クラスターパラメータグループ	arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*
クラスタースナップショット	arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*
インスタンス	arn:aws:rds: *region*:*account-id* :db:*db-instance-name*
セキュリティグループ	arn:aws:rds: *region*:*account-id* :secgrp:*security-group-name*
サブネットグループ	arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*

Amazon DocumentDB には、Amazon DocumentDB のリソースを操作するための一連のオペレーションが用意されています。可能なオペレーションのリストについては、「アクション」を参照してください。

リソース所有権について

リソース所有者は、リソースを AWS アカウント 作成した です。つまり、リソース所有者は、リソースを作成するリクエストを認証する AWS アカウント プリンシパルエンティティ (ルートアカウント、IAM ユーザー、または IAM ロール) の です。次の例は、この仕組みを示しています。

- のルートアカウントの認証情報を使用してインスタンスなどの Amazon DocumentDB リソース AWS アカウント を作成する場合、 AWS アカウント は Amazon DocumentDB リソースの所有者です。
- で IAM ユーザーを作成し AWS アカウント 、そのユーザーに Amazon DocumentDB リソースを作成するアクセス許可を付与すると、そのユーザーは Amazon DocumentDB リソースを作成できます。ただし、ユーザーが属 AWS アカウントする は Amazon DocumentDB リソースを所有しています。
- Amazon DocumentDB リソースを作成するアクセス許可 AWS アカウント を持つ で IAM ロールを作成する場合、ロールを引き受けることができるすべてのユーザーが Amazon DocumentDB リソースを作成できます。ロールが属 AWS アカウントする が Amazon DocumentDB リソースを所有しています。

リソースへのアクセスの管理

アクセス権限ポリシー では、誰が何にアクセスできるかを記述します。以下のセクションで、アクセス許可ポリシーを作成するために使用可能なオプションについて説明します。

> **Note**
>
> このセクションでは、Amazon DocumentDB のコンテキストでの IAM の使用について説明します。ここでは、IAM サービスに関する詳細情報を提供しません。完全な IAM ドキュメンテーションについては、「IAM ユーザーガイド」の「IAM とは」を参照してください。IAM ポリシー構文の詳細と説明については、「IAM ユーザーガイド」の「AWS IAM ポリシーリファレンス」を参照してください。

IAM ID にアタッチされているポリシーは、ID ベース のポリシー (IAM ポリシー) と呼ばれます。リソースにアタッチされているポリシーは、リソースベース のポリシーと呼ばれます。Amazon DocumentDB では、アイデンティティベースのポリシー (IAM ポリシー) のみをサポートしています。

トピック

- アイデンティティベースのポリシー (IAM ポリシー)
- リソースベースのポリシー

アイデンティティベースのポリシー (IAM ポリシー)

ポリシーを IAM アイデンティティにアタッチできます。例えば、次のオペレーションを実行できます。

- アカウントのユーザーまたはグループにアクセス許可ポリシーを付与する - アカウント管理者は、特定のユーザーに関連付けられているアクセス許可ポリシーを使用して、そのユーザーにインスタンスなどの Amazon DocumentDB リソースを作成するためのアクセス許可を付与できます。
- アクセス権限ポリシーをロールにアタッチする (クロスアカウントの許可を付与) - ID ベースのアクセス権限ポリシーを IAM ロールにアタッチして、クロスアカウントの権限を付与することができます。たとえば、管理者は次のように、別の AWS アカウント または AWS サービスにクロスアカウントアクセス許可を付与するロールを作成できます。
  1. アカウント A の管理者は、IAM ロールを作成して、アカウント A のリソースに許可を付与するロールに許可ポリシーをアタッチします。
  2. アカウント A の管理者は、アカウント B をそのロールを引き受けるプリンシパルとして識別するロールに、信頼ポリシーをアタッチします。
  3. アカウント B の管理者は、アカウント B のユーザーにロールを引き受けるアクセス許可を委任できます。これにより、アカウント B のユーザーがアカウント A のリソースを作成またはアクセスできるようになります。ロールを引き受けるアクセス許可を AWS サービスに付与する場合、信頼ポリシーのプリンシパルは AWS サービスプリンシパルになることもできます。

  IAM を使用した許可の委任の詳細については、「IAM ユーザーガイド」の「アクセス管理」を参照してください。

以下は、ID 123456789012 を持つユーザーが AWS アカウントのインスタンスを作成できるようにするポリシーの例です。新しいインスタンスでは、オプショングループと `default` で始まるパラメータグループ、および `default` サブネットグループを使用する必要があります。

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"
            ]
        }
    ]
}
```

Amazon DocumentDB でアイデンティティベースのポリシーを使用する場合の詳細については、Amazon DocumentDB のアイデンティティベースのポリシー (IAM ポリシー) の使用 を参照してください。ユーザー、グループ、ロール、許可の詳細については、「IAM ユーザーガイド」の「アイデンティティ (ユーザー、グループ、ロール)」を参照してください。

リソースベースのポリシー

Amazon S3 (Amazon Simple Storage Service) などその他のサービスでは、リソースベースの権限ポリシーがサポートされています。例えば、ポリシーを Amazon S3 バケットにアタッチして、そのバケットに対するアクセス許可を管理できます。Amazon DocumentDB では、リソースベースのポリシーはサポートされていません。

ポリシー要素の指定 : アクション、効果、リソース、プリンシパル

Amazon DocumentDB リソースごとに (Amazon DocumentDB リソースとオペレーション を参照)、そのサービスは、API オペレーションの設定を定義します。詳細については、「アクション」を参照してください。これらのAPIオペレーションの権限を付与するために、Amazon DocumentDB はポリシーで指定できる一連のアクションを定義します。1 つの API オペレーションの実行で、複数のアクションのアクセス権限が必要になる場合があります。

以下は、基本的なポリシーの要素です。

- リソース – ポリシーで Amazon リソースネーム (ARN) を使用して、ポリシーを適用するリソースを識別します。
- [Action] (アクション) - アクションのキーワードを使用して、許可または拒否するリソースオペレーションを識別します。例えば、`rds:DescribeDBInstances` 権限は、`DescribeDBInstances` オペレーションの実行をユーザーに許可します。

- 効果 – ユーザーが特定のアクションを要求する際の効果を指定します。許可または拒否のいずれかになります。リソースへのアクセスを明示的に付与 (許可) していない場合、アクセスは暗黙的に拒否されます。また、明示的にリソースへのアクセスを拒否すると、別のポリシーによってアクセスが許可されている場合でも、ユーザーはそのリソースにアクセスできなくなります。
- プリンシパル - ID ベースのポリシー (IAM ポリシー) で、ポリシーがアタッチされているユーザーが黙示的なプリンシパルとなります。リソースベースのポリシーでは、権限(リソースベースのポリシーにのみ適用)を受け取りたいユーザー、アカウント、サービス、またはその他のエンティティを指定します。Amazon DocumentDB では、リソースベースのポリシーはサポートされていません。

IAM ポリシーの構文と記述の詳細については、「IAM ユーザーガイド」の「AWS IAM ポリシーリファレンス」を参照してください。

すべての Amazon DocumentDB API アクションとそれらが適用されるリソースを示す表については、Amazon DocumentDB API のアクセス許可: アクション、リソース、条件リファレンス を参照してください。

ポリシーでの条件の指定

許可を付与するとき、IAM ポリシー言語を使用して、ポリシーが有効になる必要がある条件を指定できます。例えば、特定の日付の後にのみ適用されるポリシーが必要になる場合があります。ポリシー言語での条件の指定の詳細については、「IAM ユーザーガイド」の「条件」を参照してください。

条件を表すには、あらかじめ定義された条件キーを使用します。Amazon DocumentDB には、IAM ポリシーで使用できるサービス固有のコンテキストキーはありません。すべてのサービスで使用できるグローバル条件コンテキストキーのリストについては、IAM ユーザーガイドの「利用可能なグローバル条件キー」を参照してください。

Amazon DocumentDB のアイデンティティベースのポリシー (IAM ポリシー) の使用

> **Important**
>
> 特定の管理機能について、Amazon DocumentDB は Amazon RDS と共有されるオペレーショナルテクノロジーを使用します。Amazon DocumentDB コンソール、、および API コールは AWS CLI、Amazon RDS API に対する呼び出しとして口グに記録されます。

まず、Amazon DocumentDB リソースへのアクセスを管理するための基本的な概念とオプションについて説明する入門トピックを確認します。詳細については、「Amazon DocumentDB リソースへのアクセス許可の管理」を参照してください。

このトピックでは、アカウント管理者が IAM アイデンティティ(ユーザー、グループ、ロール)へのアクセス権限ポリシーをアタッチする、アイデンティティベースのポリシーの例を示します。

IAM ポリシーの例を次に示します。

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"
            ]
        }
    ]
}
```

ポリシーには、IAM ユーザー用の以下のアクセス許可を指定する単一のステートメントが含まれます。

- このポリシーにより、IAM ユーザーは CreateDBInstance アクションを使用してインスタンスを作成できます (これは create-db-instance AWS CLI オペレーションと にも適用されます AWS マネジメントコンソール)。

- Resource 要素では、ユーザーがリソースでアクションを実行できることを指定できます。Amazon リソースネーム (ARN) を使用してリソースを指定します。この ARN には、リソースが属するサービスの名前 (rds)、 AWS リージョン (* はこの例の任意のリージョンを示します)、ユーザーアカウント番号 (123456789012 はこの例のユーザー ID です)、およびリソースのタイプが含まれます。

  例の Resource 要素は、ユーザーのリソースで、以下のポリシーの制約を指定します。

  - 新しいインスタンスのインスタンス識別子は、test で始まる必要があります (例: testCustomerData1、test-region2-data)。
  - 新しいインスタンスのクラスターパラメータグループは、default で始まる必要があります。
  - 新しいインスタンスのサブネットグループは、default サブネットグループである必要があります。

ID ベースのポリシーでアクセス許可を得るプリンシパルを指定していないため、ポリシーでは Principal 要素を指定していません。ユーザーにポリシーをアタッチすると、そのユーザーが暗黙のプリンシパルになります。IAM ロールにアクセス権限ポリシーをアタッチすると、ロールの信頼ポリシーで識別されたプリンシパルがアクセス権限を得ることになります。

すべての Amazon DocumentDB API オペレーションとそれらが適用されるリソースを示す表については、Amazon DocumentDB API のアクセス許可: アクション、リソース、条件リファレンス を参照してください。

Amazon DocumentDB コンソールの使用に必要な許可

Amazon DocumentDB コンソールを使用するユーザーには、最小限の権限のセットが必要です。これらのアクセス許可により、ユーザーは の Amazon DocumentDB リソースを記述 AWS アカウントし、Amazon EC2 セキュリティやネットワーク情報などの他の関連情報を提供できます。

これらの最小限必要なアクセス許可よりも制限された IAM ポリシーを作成している場合、その IAM ポリシーを使用するユーザーに対してコンソールは意図したとおりには機能しません。これらのユーザーが引き続き Amazon DocumentDB コンソールを使用できるようにするには、AmazonDocDBConsoleFullAccess で説明されているように AWS Amazon DocumentDB のマネージドポリシー マネージドポリシーをユーザーにアタッチします。

AWS CLI または Amazon DocumentDB API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。

カスタマーマネージドポリシーの例

このセクションでは、さまざまな Amazon DocumentDB アクションの許可を付与するユーザーポリシー例を示しています。これらのポリシーは、Amazon DocumentDB API アクション、 AWS SDKs、または を使用している場合に機能します AWS CLI。コンソールを使用している場合は、「Amazon DocumentDB コンソールの使用に必要な許可」で説明しているコンソールに固有の追加のアクセス権限を付与する必要があります。

特定の管理機能では、Amazon DocumentDB は Amazon Relational Database Service (Amazon RDS) と Amazon Neptune を用いて共有される運用テクノロジーを使用します。

> **Note**
>
> すべての例で、米国東部 (バージニア北部) リージョン (us-east-1) を使用し、架空のアカウント ID を使用しています。

例

- 例 1: ユーザーが任意の Amazon DocumentDB リソースに対して任意の Describe アクションを実行できるようにする
- 例 2: ユーザーによるインスタンスの削除を禁止する
- 例 3: ストレージ暗号化が有効になっている場合を除き、ユーザーがクラスターを作成できないようにする

例 1: ユーザーが任意の Amazon DocumentDB リソースに対して任意の Describe アクションを実行できるようにする

次のアクセス権限ポリシーは、Describe で始まるすべてのアクションを実行するためのアクセス権限をユーザーに付与します。これらのアクションは、インスタンスなど Amazon DocumentDB リソースに関する情報を表示します。Resource 要素内のワイルドカード文字 (*) は、アカウントによって所有されるすべての Amazon DocumentDB リソースに対してそれらのアクションが許可されることを示します。

JSON

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

```
      {
         "Sid":"AllowRDSDescribe",
         "Effect":"Allow",
         "Action":"rds:Describe*",
         "Resource":"*"
      }
   ]
}
```

例 2: ユーザーによるインスタンスの削除を禁止する

以下のアクセス権限ポリシーは、特定のインスタンスを削除することをユーザーに禁止するためのアクセス権限を付与します。例えば、管理者ではないすべてのユーザーに対して、本稼働インスタンスの削除を拒否することもできます。

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"
      }
   ]
}
```

例 3: ストレージ暗号化が有効になっている場合を除き、ユーザーがクラスターを作成できないようにする

次のアクセス許可ポリシーは、ストレージの暗号化が有効になっていない限り、ユーザーが Amazon DocumentDB クラスターを作成することを許可しません。

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 の マネージドポリシー

ユーザー、グループ、ロールにアクセス許可を追加するには、自分でポリシーを記述するよりも AWS 管理ポリシーを使用する方が簡単です。チームに必要な権限のみを提供する IAM カスタマーマネージドポリシーを作成する には時間と専門知識が必要です。すぐに開始するには、 AWS マネージドポリシーを使用できます。これらのポリシーは、一般的なユースケースをターゲット範囲に含めており、 AWS アカウントで利用できます。 AWS 管理ポリシーの詳細については、IAM ユーザーガイドの「 AWS 管理ポリシー」を参照してください。

AWS サービスは、 AWS 管理ポリシーを維持および更新します。 AWS 管理ポリシーのアクセス許可は変更できません。サービスは、新機能をサポートするために、 AWS 管理ポリシーに追加のアクセス許可を追加することがあります。この種類の更新はポリシーがアタッチされている、すべてのアイデンティティ (ユーザー、グループおよびロール) に影響を与えます。サービスは、新機能が起動されたとき、または新しいオペレーションが利用可能になったときに、 AWS マネージドポリシーを更新する可能性が最も高いです。サービスは AWS マネージドポリシーからアクセス許可を削除しないため、ポリシーの更新によって既存のアクセス許可が損なわれることはありません。

さらに、 は、複数のサービスにまたがるジョブ関数の マネージドポリシー AWS をサポートしています。たとえば、 ViewOnlyAccess AWS マネージドポリシーは、多くの AWS サービスとリソースへの読み取り専用アクセスを提供します。サービスが新機能を起動すると、 は新しいオペレーションとリソースの読み取り専用アクセス許可 AWS を追加します。ジョブ機能のポリシーの一覧お

よび詳細については、「IAM ユーザーガイド」の「AWS のジョブ機能のマネージドポリシー」を参照してください。

アカウントのユーザーにアタッチできる以下の AWS 管理ポリシーは、Amazon DocumentDB に固有のものです。

- AmazonDocDBFullAccess – ルートのすべての Amazon DocumentDB リソースへのフルアクセスを許可します AWS アカウント。
- AmazonDocDBReadOnlyAccess – ルートのすべての Amazon DocumentDB リソースへの読み取り専用アクセスを許可します AWS アカウント。
- AmazonDocDBConsoleFullAccess - AWS マネジメントコンソールを使用して、Amazon DocumentDB および Amazon DocumentDB Elastic クラスターリソースを管理するためのフルアクセスを許可します。
- AmazonDocDBElasticReadOnlyAccess – ルートのすべての Amazon DocumentDB エラスティッククラスターリソースへの読み取り専用アクセスを許可します AWS アカウント。
- AmazonDocDBElasticFullAccess – ルートのすべての Amazon DocumentDB エラスティッククラスターリソースへのフルアクセスを許可します AWS アカウント。

AmazonDocDBFullAccess

このポリシーは、プリンシパルにすべての Amazon DocumentDB アクションへのフルアクセスを許可する管理者権限を付与します。このポリシーの権限は、次のようにグループ化されています。

- Amazon DocumentDB の権限は、Amazon DocumentDB アクションをすべて許可します。
- このポリシーの Amazon EC2 権限の一部は、API リクエストで渡されたリソースを検証するために必要です。これは、Amazon DocumentDB がクラスターでリソースを正常に使用できるようにするためです。このポリシーの残りの Amazon EC2 アクセス許可により、Amazon DocumentDB はクラスターへの接続を可能にするために必要な AWS リソースを作成できます。
- Amazon DocumentDB 権限は、リクエストで渡されたリソースを検証するために API コール中に使用されます。これらは、Amazon DocumentDB が、渡されたキーを Amazon DocumentDB クラスターで使用できるようにするために必要です。
- CloudWatch Logsは、Amazon DocumentDB がログ配信先に到達可能であり、ブローカーログの使用に有効であることを確認するために必要です。

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

このポリシーは、ユーザーが Amazon DocumentDB で情報を表示できるようにする読み取り専用の権限を付与します。このポリシーが添付されているプリンシパルは、既存のリソースを更新または削除したり、新しい Amazon DocumentDB リソースを作成したりすることはできません。例えば、これらの権限を持つプリンシパルは、自分のアカウントに関連付けられているクラスターと構成のリストを表示できますが、クラスターの構成や設定を変更することはできません。このポリシーの権限は、次のようにグループ化されています。

- Amazon DocumentDB 権限を使用すると、Amazon DocumentDB リソースを一覧表示し、それらを記述し、それらに関する情報を取得できます。
- Amazon EC2 権限は、クラスターに関連付けられている Amazon VPC、サブネット、セキュリティグループ、および ENI を記述するために使用されます。

- Amazon DocumentDB 権限は、クラスターに関連付けられているキーを記述するために使用されます。

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

以下 AWS マネジメントコンソール に対して を使用して Amazon DocumentDB リソースを管理するためのフルアクセスを付与します。

- すべての Amazon DocumentDB アクションと Amazon DocumentDB クラスターアクションを許可する Amazon DocumentDB のアクセス許可。
- このポリシーの Amazon EC2 権限の一部は、API リクエストで渡されたリソースを検証するために必要です。これは、Amazon DocumentDB がリソースを正常に使用して、クラスターをプロビジョンおよび維持できるようにするためです。このポリシーの残りの Amazon EC2 アクセス許可により、Amazon DocumentDB は VPCEndpoint などのクラスターに接続するために必要な AWS リソースを作成できます。
- AWS KMS アクセス許可は、 への API コール中に使用され AWS KMS 、リクエストで渡されたリソースを検証します。これらは、Amazon DocumentDB が、渡されたキーを使用して、Amazon DocumentDB エラスティッククラスターで保存されている保管中のデータを暗号化および復号化できるようにするために必要です。
- CloudWatch Logs は、Amazon DocumentDB がログ配信先に到達可能であり、ログ使用の監査およびプロファイリングに有効であることを確認するために必要です。
- 特定のシークレットを検証し、それを使用して Amazon DocumentDB Elastic クラスターの管理者ユーザーをセットアップするには、Secrets Manager 権限が必要です。
- Amazon DocumentDB クラスター管理アクションには Amazon RDS アクセス権限が必要です。特定の管理機能について、Amazon DocumentDB は Amazon RDS と共有されるオペレーショナルテクノロジーを使用します。
- SNS 権限は、プリンシパルが Amazon Simple Notification Service (Amazon SNS) のサブスクリプションとトピックにアクセスして、Amazon SNS メッセージを発行することを許可します。
- メトリクスとログの公開に必要なサービスにリンクされたロールを作成するには IAM 権限が必要です。

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

このポリシーは、ユーザーが Amazon DocumentDB で Elastic クラスター情報を表示できるようにする読み取り専用の権限を付与します。このポリシーが添付されているプリンシパルは、既存のリソースを更新または削除したり、新しい Amazon DocumentDB リソースを作成したりすることはできません。例えば、これらの権限を持つプリンシパルは、自分のアカウントに関連付けられているクラスターと構成のリストを表示できますが、クラスターの構成や設定を変更することはできません。このポリシーの権限は、次のようにグループ化されています。

- Amazon DocumentDB Elastic クラスター権限を使用すると、Amazon DocumentDB Elastic クラスターリソースを一覧表示し、それらを記述し、それらに関する情報を取得できます。
- CloudWatch 権限は、サービスメトリクスを検証するために使用します。

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

このポリシーは、Amazon DocumentDB Elastic クラスターに関するすべての Amazon DocumentDB アクションへのフルアクセスをプリンシパルに許可する管理者権限を付与します。

このポリシーは、条件内で AWS タグ (https://docs.aws.amazon.com/tag-editor/latest/userguide/tagging.html) を使用して、リソースへのアクセスをスコープします。シークレットを使用する場合は、タグキー `DocDBElasticFullAccess` とタグ値でタグ付けする必要があります。カスタマー管理キーを使用する場合は、タグキー `DocDBElasticFullAccess` とタグ値でタグ付けする必要があります。

このポリシーの権限は、次のようにグループ化されています。

- Amazon DocumentDB Elastic クラスターアクセス権限では、すべての Amazon DocumentDB アクションが許可されます。
- このポリシーの Amazon EC2 権限の一部は、API リクエストで渡されたリソースを検証するために必要です。これは、Amazon DocumentDB がリソースを正常に使用して、クラスターをプロビジョンおよび維持できるようにするためです。このポリシーの残りの Amazon EC2 アクセス許可により、Amazon DocumentDB は VPC エンドポイントなどのクラスターに接続するために必要な AWS リソースを作成できます。
- AWS KMS Amazon DocumentDB が渡されたキーを使用して Amazon DocumentDB エラスティッククラスター内の保管中のデータを暗号化および復号化できるようにするには、 アクセス許可が必要です。

> **Note**
>
> カスタマー管理キーには、キー `DocDBElasticFullAccess` が付いたタグとタグ値が必要です。

- 特定のシークレットを検証し、それを使用して Amazon DocumentDB Elastic クラスターの管理者ユーザーをセットアップするには、SecretsManager の権限が必要です。

> **Note**
>
> 使用するシークレットには、キー DocDBElasticFullAccess が付いたタグとタグ値が必要です。

- メトリクスとログの公開に必要なサービスにリンクされたロールを作成するには IAM 権限が必要です。

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

AWS Identity and Access Management エンティティ`AmazonDocDBElasticServiceRolePolicy`に をアタッチすることはできません。このポリシーは、Amazon DocumentDB がユーザーに代わってアクションを実行することを許可するサービスリンクロールにアタッチされます。詳細については、「エラスティッククラスター上のサービスにリンクされたロール」を参照してください。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
```

```
                    "cloudwatch:namespace": [
                        "AWS/DocDB-Elastic"
                    ]
                }
            }
        }
    ]
}
```

AWS マネージドポリシーに対する Amazon DocumentDB の更新

変更	説明	日付
AmazonDocDBElasticFullAccess、AmazonDocDBConsoleFullAccess - 変更	ポリシーが更新され、保留中のメンテナンスアクションが追加されました。	2/11/2025
AmazonDocDBElasticFullAccess、AmazonDocDBConsoleFullAccess - 変更	ポリシーが更新され、クラスターの開始/停止とクラスタースナップショットアクションのコピーが追加されました。	2/21/2024
AmazonDocDBElasticReadOnlyAccess、AmazonDocDBElasticFullAccess - 変更	ポリシーを更新し、`cloudwatch:GetMetricData` アクションを追加。	6/21/2023
AmazonDocDBElasticReadOnlyAccess – 新しいポリシー	Amazon DocumentDB Elastic クラスターの新しいマネージドポリシー。	6/8/2023
AmazonDocDBElasticFullAccess – 新しいポリシー	Amazon DocumentDB Elastic クラスターの新しいマネージドポリシー。	6/5/2023
AmazonDocDB-ElasticServiceRolePolicy - 新しいポリシー	Amazon DocumentDB は、Amazon DocumentDB エラスティッククラスター用の新しい AWS ServiceRo	11/30/2022

変更	説明	日付
	leForDocDB-Elastic サービスリンクロールを作成します。	
AmazonDocDBConsoleFullAccess - 変更	ポリシーを更新し、Amazon DocumentDB グローバル権限と Elastic クラスター権限を追加。	11/30/2022
AmazonDocDBConsoleFullAccess, AmazonDocDBFullAccess, AmazonDocDBReadOnlyAccess - 新しいポリシー	サービスの起動。	1/19/2017

Amazon DocumentDB API のアクセス許可: アクション、リソース、条件リファレンス

次のセクションは、IAM アイデンティティに付加できる Amazon DocumentDB のアイデンティティベースのポリシー (IAM ポリシー) の使用 およびアクセス許可ポリシー (アイデンティティベースのポリシー) を設定および書き込む際の参照として使用してください。

以下に各 Amazon DocumentDB API オペレーションを示します。リストには、アクションを実行するためのアクセス許可を付与できる対応するアクション、アクセス許可を付与できる AWS リソース、きめ細かなアクセスコントロールに含めることができる条件キーが含まれています。ポリシーの `Action` フィールドにアクションを、ポリシーの `Resource` フィールドにリソース値を、ポリシーの `Condition` フィールドに条件を指定します。条件の詳細については、「ポリシーでの条件の指定」を参照してください。

Amazon DocumentDB ポリシーで AWS全体の条件キーを使用して、条件を表現できます。 AWS全体のキーの完全なリストについては、IAM ユーザーガイドの「使用可能なキー」を参照してください。

IAM ポリシーシミュレーターを使用して IAM ポリシーをテストできます。Amazon DocumentDB AWS アクションなど、各アクションに必要なリソースとパラメータのリストが自動的に表示されます。IAM ポリシーシミュレーター により、指定する各アクションに必要な許可が決定されま

す。IAM ポリシーシミュレーターの詳細については、IAMユーザーガイド の IAM Policy Simulatorを使用したIAMポリシーのテスト を参照してください。

> **Note**
>
> アクションを指定するには、API オペレーション名 (`rds:CreateDBInstance` など) の前に `rds:` プレフィックスを使用します。

以下に、Amazon RDS API のオペレーションとそれに関連するアクション、リソース、および条件キーを示します。

トピック

- リソースレベルの許可をサポートする Amazon DocumentDB アクション
- リソースレベルの権限をサポートしない Amazon DocumentDB アクション

リソースレベルの許可をサポートする Amazon DocumentDB アクション

リソースレベルの許可とは、ユーザーがアクションを実行できるリソースを指定できる機能を提供します。Amazon DocumentDB は、リソースレベルのアクセス許可を部分的にサポートします。これは、特定の Amazon DocumentDB アクションについて、満たす必要のある条件、またはユーザーが使用を許可されている特定のリソースに基づいて、ユーザーがそれらのアクションをいつ使用できるかを制御できることを意味します。例えば、特定のインスタンスのみ変更するアクセス許可をユーザーに付与できます。

Amazon DocumentDB API オペレーションと関連するアクション、リソース、および条件キーを以下に示します。

> **Note**
>
> 特定の管理機能について、Amazon DocumentDB は Amazon RDS と共有されるオペレーショナルテクノロジーを使用します。Amazon DocumentDB アクションおよびアクセス許可の詳細については、「サービス認可リファレンス」の「Amazon RDS のアクション、リソース、および条件キー」を参照してください。

Amazon DocumentDB API オペレーションとアクション	リソース	条件キー
AddTagsToResource rds:AddTagsToResource	インスタンス arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
	サブネットグループ arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
ApplyPendingMaintenanceAction rds:ApplyPendingMaintenanceAction	インスタンス arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
CopyDBClusterSnapshot rds:CopyDBClusterSnapshot	クラスタースナップショット arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
CreateDBCluster rds:CreateDBCluster	クラスター arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	rds:cluster-tag
	クラスターパラメータグループ	rds:cluster-pg-tag

Amazon DocumentDB API オペレーションとアクション	リソース	条件キー
	arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	
	サブネットグループ arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
CreateDBClusterParameterGroup rds:CreateDBClusterParameterGroup	クラスターパラメータグループ arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
CreateDBClusterSnapshot rds:CreateDBClusterSnapshot	クラスター arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	rds:cluster-tag
	クラスタースナップショット arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
CreateDBInstance rds:CreateDBInstance	インスタンス arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:DatabaseClass rds:db-tag

Amazon DocumentDB API オペレーションとアクション	リソース	条件キー
	クラスター arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	rds:cluster-tag
CreateDBSubnetGroup rds:CreateDBSubnetGroup	サブネットグループ arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
DeleteDBInstance rds:DeleteDBInstance	インスタンス arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
DeleteDBSubnetGroup rds:DeleteDBSubnetGroup	サブネットグループ arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
DescribeDBClusterParameterGroups rds:DescribeDBClusterParameterGroups	クラスターパラメータグループ arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag

Amazon DocumentDB API オペレーションとアクション	リソース	条件キー
DescribeDBClusterParameters rds:DescribeDBClusterParameters	クラスターパラメータグループ arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
DescribeDBClusters rds:DescribeDBClusters	クラスター arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
DescribeDBClusterSnapshotAttributes rds:DescribeDBClusterSnapshotAttributes	クラスタースナップショット arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
DescribeDBSubnetGroups rds:DescribeDBSubnetGroups	サブネットグループ arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
DescribePendingMaintenanceActions rds:DescribePendingMaintenanceActions	インスタンス arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:DatabaseClass rds:db-tag

Amazon DocumentDB API オペレーションとアクション	リソース	条件キー
FailoverDBCluster rds:FailoverDBCluster	クラスター arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
ListTagsForResource rds:ListTagsForResource	インスタンス arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
	サブネットグループ arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
ModifyDBCluster rds:ModifyDBCluster	クラスター arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	rds:cluster-tag
	クラスターパラメータグループ arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
ModifyDBClusterParameterGroup rds:ModifyDBClusterParameterGroup	クラスターパラメータグループ arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag

Amazon DocumentDB API オペレーションとアクション	リソース	条件キー
ModifyDBClusterSnapshotAttribute rds:ModifyDBClusterSnapshotAttribute	クラススナップショット arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
ModifyDBInstance rds:ModifyDBInstance	インスタンス arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:DatabaseClass rds:db-tag
RebootDBInstance rds:RebootDBInstance	インスタンス arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
RemoveTagsFromResource rds:RemoveTagsFromResource	インスタンス arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
	サブネットグループ arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
ResetDBClusterParameterGroup rds:ResetDBClusterParameterGroup	クラスターパラメータグループ arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag

Amazon DocumentDB API オペレーションとアクション	リソース	条件キー
RestoreDBClusterFromSnapshot rds:RestoreDBClusterFromSnapshot	クラスター arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
	クラスタースナップショット arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
RestoreDBClusterToPointInTime rds:RestoreDBClusterToPointInTime	クラスター arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
	サブネットグループ arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag

リソースレベルの権限をサポートしない Amazon DocumentDB アクション

Amazon DocumentDB ポリシーのすべての IAM アクションを使用して、そのアクションを使用する許可をユーザーに付与または拒否できます。ただし、すべての Amazon DocumentDB アクションが、アクションを実行することができるリソースを指定できる、リソースレベルのアクセス許可をサポートしているわけではありません。現在、次の Amazon DocumentDB API アクションは、リソースレベルの許可をサポートしていません。したがって、IAM ポリシーでこれらのアクションを使用するには、ステートメントで * 要素に Resource ワイルドカードを使って、アクションに対してすべてのリソースを使用する許可をユーザーに付与する必要があります。

- rds:DescribeDBClusterSnapshots

- rds:DescribeDBInstances

サービス間での不分別な代理処理の防止

混乱した代理問題は、アクションを実行するためのアクセス許可を持たないエンティティが、より特権のあるエンティティにアクションの実行を強制できてしまう場合に生じる、セキュリティ上の問題です。では AWS、サービス間のなりすましにより、混乱した代理問題が発生する可能性があります。サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。これを防ぐため、 AWS では、アカウントのリソースへのアクセス権が付与されたサービスプリンシパルで、すべてのサービスのデータを保護するために役立つツールを提供しています。

リソースポリシーで aws:SourceArnおよび aws:SourceAccount グローバル条件コンテキストキーを使用して、Amazon DocumentDB がリソースに別のサービスに付与するアクセス許可を制限します。クロスサービスアクセスにリソースを 1 つだけ関連付けたい場合は、aws:SourceArn を使用します。そのアカウント内のリソースをクロスサービスの使用に関連付けることを許可する場合は、aws:SourceAccount を使用します。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定して、aws:SourceArn グローバル条件コンテキストキーを使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合には、グローバルコンテキスト条件キー aws:SourceArn で、ARN の未知部分を示すためにワイルドカード文字 (*) を使用します。例えば、arn:aws:*servicename*:*:*123456789012*:*。

aws:SourceArn の値に Amazon S3 バケット ARN などのアカウント ID が含まれていない場合は、両方のグローバル条件コンテキストキーを使用して、アクセス許可を制限する必要があります。

の値はクラスターaws:SourceArnである必要があります。

次の例は、Amazon DocumentDB で aws:SourceArnおよび aws:SourceAccount グローバル条件コンテキストキーを使用して、混乱した代理問題を防ぐ方法を示しています。

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

IAM ID を使用した認証

Amazon DocumentDB ユーザーやアプリケーションは、IAM ユーザーとロールを使用して Amazon DocumentDB クラスターに対する認証を行うことができます。Amazon DocumentDB IAM 認証は、パスワードレスな認証方法です。IAM ロール/ユーザーを使用する場合、クライアントアプリケーションはパスワードシークレットを Amazon DocumentDB クラスターに送信しません。代わりに、クライアント接続は一時的なセキュリティトークン AWS STS を使用して認証されます。管理者以外のユーザーとアプリケーションは、異なる Amazon DocumentDB クラスターやその他の AWS サービスに接続するときに、同じ IAM ID ARN を使用できるようになりました。

また、パスワードベース認証と IAM 認証を併用して、Amazon DocumentDB クラスターに対してユーザーやアプリケーションを認証することも可能です。IAM 認証は、Amazon DocumentDB インスタンスベースのクラスターバージョン 5.0 でのみ使用できます。なお、IAM ID ARN を用いた IAM 認証は、Amazon DocumentDB プライマリユーザーに対してはサポートしています。

> **Note**
>
> プライマリユーザーは、既存のパスワードベースの認証を使用してのみ認証できます。

トピック

- IAM ユーザーとロールを使用した認証の開始方法

- IAM を使用して Amazon DocumentDB AWS を認証するように AWS コンピューティングタイプを設定する
- IAM 認証リクエストのモニタリング
- IAM 認証の使用
- IAM をサポートするドライバー
- IAM ID 認証に関するよくある質問

IAM ユーザーとロールを使用した認証の開始方法

IAM ID を持つ Amazon DocumentDB ユーザーとロールは、`$external` データベースで作成および管理されます。

ユーザーの作成

プライマリユーザーとして接続し、IAM ユーザーとロールを作成します。

```
use $external;
db.createUser(
    {
        user: "arn:aws:iam::123456789123:user/iamuser",
        mechanisms: ["MONGODB-AWS"],
        roles: [ { role: "readWrite", db: "readWriteDB" } ]
    }
);
```

または、IAM ロールを使用して Amazon DocumentDB ユーザーを追加します。

```
use $external;
db.createUser(
    {
        user: "arn:aws:iam::123456789123:role/iamrole",
        mechanisms: ["MONGODB-AWS"],
        roles: [ { role: "readWrite", db: "readWriteDB" } ]
    }
);
```

IAM ユーザーまたはロールの変更

既存の IAM ユーザーの変更:

```
use $external;
db.updateUser(
  "arn:aws:iam::123456789123:user/iamuser",
  {
    roles: [ { role: "read", db: "readDB" } ]
  }
);
```

既存の IAM ロールの変更:

```
use $external;
db.updateUser(
  "arn:aws:iam::123456789123:role/iamrole",
  {
    roles: [ { role: "read", db: "readDB" } ]
  }
);
```

特定の IAM ユーザーに対してロールを付与または取り消すには:

```
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" } ]
);
```

特定の IAM ロールに対してロールを付与または取り消すには:

```
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" } ]
);
```

IAM ユーザーまたはロールの削除

既存の IAM ユーザーを削除するには:

```
use $external;
db.dropUser("arn:aws:iam::123456789123:user/iamuser");
```

既存の IAM ロールを削除するには:

```
use $external;
db.dropUser("arn:aws:iam::123456789123:role/iamrole");
```

IAM を使用して認証するように接続 URI AWS を設定する

AWS IAM を使用して認証するには、 authSourceとして $external、 authMechanismとして URI パラメータを使用しますMONGODB-AWS。IAM ユーザーを使用する場合、ユーザー名フィールドとパスワードフィールドはそれぞれアクセスキーとシークレットキーに置き換えられます。IAM ロールを引き受けている場合、現在の環境 (AWS Lambda 関数、Amazon EC2 インスタンスなど) にアタッチされます。MONGODB-AWS メカニズムを使用して認証する場合、特に認証情報を渡す必要はありません。MONGODB-AWS 認証メカニズムをサポートする MongoDB ドライバーを使用している場合、これらのドライバーにはコンピューティングインスタンス (Amazon EC2、Lambda 関数など) から IAM ロール認証情報を取得する能力もあります。次の例は、(IAM ユーザーの) アクセスキーとシークレットキーを手動で渡して行う、mongo シェル による MONGODB-AWS を使用した Amazon DocumentDB に対する認証プロセスを示します。

こちらの例は、(環境にアタッチされた IAM ロール を利用することで)、認証情報を明示的に渡さずに Python コードを使用して行う、MONGODB-AWS による Amazon DocumentDB に対する認証プロセスを示します。

```
##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')
```

次の例は、(IAM ユーザーの) アクセスキーとシークレットキーを手動で渡して行う、mongo シェルによる MONGODB-AWS メカニズムを使用した Amazon DocumentDB に対する認証プロセスを示します。

```
$ mongo 'mongodb://<access_key>:<secret_key>@<cluster_endpoint>:<db_port>/test?
authSource=%24external&authMechanism=MONGODB-AWS'
```

こちらの例は、(環境にアタッチされた IAM ロール を利用することで)、認証情報を明示的に渡さずに mongo シェルを使用して行う、MONGODB-AWS による Amazon DocumentDB に対する認証プロセスを示します。

```
$ mongo 'mongodb://<cluster_endpoint>:<db_port>/test?authSource=
%24external&authMechanism=MONGODB-AWS'
```

IAM を使用して Amazon DocumentDB AWS を認証するように AWS コンピューティングタイプを設定する

Amazon EC2/AWS Lambda/AWS Fargateの使用

Amazon EC2 は次の環境変数を使用します。EC2 インスタンスにアタッチされた IAM ロール、または Lambda 関数または Amazon ECS タスクに関連付けられた実行 IAM ロールがある場合、これらの変数は自動的に入力され、ドライバーは環境からこれらの値を取得できます。

```
AWS_ACCESS_KEY_ID
AWS_SECRET_ACCESS_KEY
AWS_SESSION_TOKEN
```

環境変数に関する詳細については、「AWS Lambda 開発者ガイド」の「Lambda 環境関数の使用」を参照してください。

Amazon EKS の使用

Amazon Elastic Kubernetes Service (Amazon EKS) ポッドにロールを割り当てると、次の 2 つの環境変数が自動的に設定されます。

```
AWS_WEB_IDENTITY_TOKEN_FILE - path of web identity token file
AWS_ROLE_ARN - Name of IAM role to connect with
```

これらの変数を使用して、 の AWS SDK 呼び出しを使用してコードからロールを手動で引き受けますAssumeRoleWithWebIdentity。

- ProviderID のパラメータは省略します。
- AWS_WEB_IDENTITY_TOKEN_FILE 環境変数で説明されている ファイルで、WebIdentityToken パラメータの値を確認します。

詳細については、「Amazon EKS ユーザーガイド」の「Amazon EKS とは」を参照してください。

IAM 認証リクエストのモニタリング

Amazon DocumentDB 監査の使用

Amazon CloudWatch の監査ログフォルダに移動し、さまざまな検索パターンを使用して IAM 認証のログを取得することができます。例えば、「すべてのログストリームを検索する」の検索パターンとして { $.param.mechanism = "MONGODB-AWS" } を使用できます。

監査でサポートされているイベントの詳細については、「Amazon DocumentDB イベントの監査」を参照してください。

Amazon CloudWatch メトリクスを使用する

StsGetCallerIdentityCalls: このメトリクスは、Amazon DocumentDB インスタンスが Regionalized AWS Security Token Service (AWS STS) エンドポイントに対して行っているGetCallerIdentity呼び出しの数を示します。データベースインスタンスが STS GetCallerIdentity呼び出しを行う必要がある理由については、MONGODB-AWS認証仕様を参照してください。

IAM 認証の使用

独自のデータベースでユーザー名やパスワードを管理したくない場合は、IAM 認証を使用できます。IAM 認証は、Amazon DocumentDB インスタンスベースのクラスターバージョン 5.0 でのみ使用できます。

IAM 認証は AWS Security Token Service () に依存しますAWS STS。IAM 認証の使用時に AWS STS スロットリング例外が発生した場合は、接続レートを下げます。

IAM クォータについては、IAM ユーザーガイドの「IAM と AWS STS クォータ」を参照してください。

IAM をサポートするドライバー

Amazon DocumentDB 5.0 と MONGODB-AWS 認証メカニズムをサポートするドライバーは、Amazon DocumentDB の IAM 認証実装とも連携するはずです。

> **Important**
>
> なお、バージョン 6.13.1 より前の Node.js ドライバーには既知の制限事項があり、現在 Amazon DocumentDB の IAM ID 認証ではサポートされていません。Node.js ドライバーを使用する Node.js ドライバーとツール (例えば、mongosh) は、Node.js ドライバーバージョン 6.13.1 以降を使用するようにアップグレードする必要があります。

IAM ID 認証に関するよくある質問

参照できるサンプルはありますか？

サンプルユースケースと設定については、以下のページを参照してください。

- 人間のユーザーが、IAM ユーザーと IAM ロールを使用して Amazon DocumentDB に対して認証する方法
- IAM ロールを使用した、Amazon DocumentDB に対するパスワードレス認証

Python ドライバーの使用中にエラーが発生しました：「pymongo.errors.ConfigurationError: MONGODB-AWS authentication requires pymongo-auth-aws」。このエラーを解消するにはどうすればよいですか？

IAM 認証で Python ドライバーをインストールするときは、必ず次のステートメントを使用してください。

```
pip install 'pymongo[aws]'
```

これにより、IAM 認証が機能するために必要な追加の AWS 依存関係がインストールされます。

IAM ロールの一時的な認証情報の有効期限が切れると、接続は切断されますか?

いいえ。一時的な IAM 認証情報は、接続と認証の確立にのみ使用されます。その後のすべての承認は Amazon DocumentDB クラスター内で行われます。IAM 認証情報がローテーションされたり、あるいは期限切れになっても、接続が削除されたり古くなることはありません。

SCRAM-SHA-256 を使用した認証

Amazon DocumentDB (MongoDB 互換) は、パスワードベースの認証のために Salted Challenge Response Authentication Mechanism (SCRAM) をサポートしています。SCRAM では、サーバーはパスワードをサーバーに送信せずに、クライアントがユーザーのパスワードを知っていることを確認します。Amazon DocumentDB は、 SCRAM-SHA-1と の 2 つの SCRAM メカニズムをサポートしていますSCRAM-SHA-256。 は、より強力な SHA-256 ハッシュ関数SCRAM-SHA-256を使用し、FIPS モードと連携します。

ユーザーを作成または更新するときに、1 つまたは両方の SCRAM メカニズムをユーザーに割り当てます。で作成されたユーザーは、変更を加えることなくSCRAM-SHA-1作業を継続します。SCRAM-SHA-256 認証は、Amazon DocumentDB インスタンスベースのクラスターマイナーバージョン 5.0.1 以降および 8.0.1 以降で使用できます。

> **SCRAM と IAM 認証は相互に排他的です**
>
> ユーザーは SCRAM メカニズムまたは MONGODB-AWSメカニズムのいずれかを持つことができますが、両方を持つことはできません。IAM ID 認証については、「」を参照してくださいIAM ID を使用した認証。

トピック

- SCRAM メカニズムを使用したユーザーの作成
- 既存のユーザーの更新
- SCRAM-SHA-256 を使用した認証

SCRAM メカニズムを使用したユーザーの作成

ユーザーを作成するときに、 mechanismsフィールドで有効にする SCRAM メカニズムを指定できます。次の例では、 SCRAM-SHA-1 のみ、 SCRAM-SHA-256のみ、および両方のメカニズムを持つユーザーを作成します。

```
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']
});
```

デフォルトのメカニズム

mechanisms フィールドを指定しない場合、Amazon DocumentDB はユーザーSCRAM-SHA-256に対して SCRAM-SHA-1と の両方を有効にします。次の 2 つのコマンドは同等です。

```
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']
});
```

両方のメカニズムをデフォルトで有効にすると、 のみをサポートする古い MongoDB ドライバーとの互換性が維持されますSCRAM-SHA-1。新しいドライバーと FIPS モードは を優先しますSCRAM-SHA-256。でユーザーを認証しない場合はSCRAM-SHA-1、すべてのユーザーの SCRAM-SHA-256 mechanismsフィールドに のみを指定します。

既存のユーザーの更新

で作成されたユーザーは、変更なしでSCRAM-SHA-1引き続きサポートされます。既存のユーザーをから SCRAM-SHA-1 に切り替えるにはSCRAM-SHA-256、新しいパスワードとSCRAM-SHA-256メカニズムでユーザーを更新します。

```
db.runCommand({
    updateUser: 'existing',
    pwd: passwordPrompt(),
    mechanisms: ['SCRAM-SHA-256']
});
```

SCRAM-SHA-256 を使用した認証

ドライバーは、各接続で最初のコマンドを送信する前に、接続文字列パラメータを使用して自動的に認証します。接続 URI SCRAM-SHA-256で を選択するには、 authMechanismパラメータを に設定しますSCRAM-SHA-256。

```
mongodb://username:password@hostname:port/authenticationDatabase?authMechanism=SCRAM-
SHA-256
```

mongosh から明示的な認証リクエストを送信することもできます。

```
db.auth({user: 'sha256user', pwd: passwordPrompt(), mechanism: 'SCRAM-SHA-256'})
Enter password:
1
```

接続オプションで認証メカニズムを指定しない場合、ドライバーはターゲットデータベースユーザーがサポートするメカニズムを自動的に選択します。

Amazon DocumentDB ユーザーの管理

Amazon DocumentDB の場合、ユーザーはクラスターへの認証にパスワードを接続します。各クラスターには、クラスターの作成時に設定されるプライマリサインイン認証情報があります。

> **Note**
>
> 2020 年 3 月 26 日 より前に作成されたすべての新規ユーザーには dbAdminAnyDatabase、readWriteAnyDatabase、および clusterAdmin のロールが付与されています。クラスター内のすべてのユーザーに最小限の特権を適用するために、すべてのユーザーを再評価し、必要に応じてロールを変更することをお勧めします。
> 詳細については、「ロールベースのアクセス制御 (RBAC) を使用したデータベースへのアクセス」を参照してください。

プライマリおよび serviceadmin ユーザ

新しく作成された Amazon DocumentDB クラスターには、マスターユーザーと serviceadmin ユーザーの 2 つのユーザーがあります。

マスターユーザー は単一の特権ユーザーであり、管理タスクの実行と、ロールを持つ追加のユーザーの作成を行うことができます。Amazon DocumentDB クラスターに初めて接続するときは、マスターユーザー名およびパスワードを使用して認証する必要があります。マスターユーザーには、Amazon DocumentDB クラスターの作成時に該当する管理権限が提供され、さらに root のロールが付与されます。

serviceadmin ユーザーは、クラスターの作成時に暗黙で作成されます。すべての Amazon DocumentDB クラスターには、クラスターを管理する AWS 機能を提供するserviceadminユーザーがあります。パスワードでのログイン、パスワードの削除や名前変更、または serviceadmin のアクセス許可の変更を行うことはできません。それらの操作を試みると、エラーが発生します。

> **Note**
>
> Amazon DocumentDB クラスターのマスターユーザーや serviceadmin ユーザーを削除したり、マスターユーザーの root のロールを無効にしたりすることはできません。
> プライマリユーザーのパスワードを忘れた場合は、 AWS マネジメントコンソール または を使用してリセットできます AWS CLI。

追加ユーザーの作成

マスターユーザー (または createUser ロールを持つユーザー) として接続すると、次に示すように、新しいユーザーを作成できます。

```
db.createUser(
    {
        user: "sample-user-1",
        pwd: "password123",
        roles:
            [{"db":"admin", "role":"dbAdminAnyDatabase" }]
    }
)
```

ユーザの詳細を表示するには、次のように show users コマンドを使用できます。また、dropUser コマンドを使用してユーザーを削除することもできます。詳細については、「一般的なコマンド」を参照してください。

```
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"
     }
    ]
}
```

この例では、新しいユーザー sample-user-1 を admin データベースに帰属させています。これは、新しいユーザーには常に当てはまります。Amazon DocumentDB には、authenticationDatabase のコンセプトを含んでおらず、したがって、すべての認証は admin データベースで実施されます。

ユーザーの作成時に、db フィールドを省略してロールを指定すると、Amazon DocumentDB は接続の発行先であるデータベースに暗黙でロールを帰属させます。たとえば、データベース sample-database に対して接続を発行し、次のコマンドを実行すると、ユーザー sample-user-2 が admin データベース内に作成され、データベース sample-database への readWrite アクセス許可が付与されます。

```
db.createUser(
    {
        user: "sample-user-2",
        pwd: "password123",
        roles:
            ["readWrite"]
    }
)
```

すべてのデータベースを対象とするロール (readInAnyDatabase など) を持つユーザーを作成するには、ユーザーの作成時に admin データベースのコンテキストで操作をしているか、ユーザーの作成時にロールのデータベースを明示的に指定する必要があります。

データベースのコンテキストを切り替えるには、次のコマンドを使用できます。

```
use admin
```

ロールベースのアクセスコントロールや、クラスター内のすべてのユーザーに最小限の特権を適用する方法の詳細については、「ロールベースのアクセス制御 (RBAC) を使用したデータベースへのアクセス」を参照してください。

Amazon DocumentDB のパスワードの自動ローテーション

を使用すると AWS Secrets Manager、コード内のハードコードされた認証情報 (パスワードを含む) を Secrets Manager への API コールに置き換えて、プログラムでシークレットを取得できます。シークレットはそこに存在しないため、これは、あなたのコードを調べている誰かがシークレットを漏らさないようにするのに役立ちます。また、指定したスケジュールに従って自動的にシークレット

を更新するように Secrets Manager を設定することができます。これにより、長期のシークレットを短期のシークレットに置き換えることが可能となり、侵害されるリスクが大幅に減少します。

Secrets Manager を使用すると、Secrets Manager が提供する AWS Lambda 関数を使用して Amazon DocumentDB パスワード (つまり、シークレット) を自動的にローテーションできます。

AWS Secrets Manager および Amazon DocumentDB とのネイティブ統合の詳細については、以下を参照してください。

- ブログ: How to rotate Amazon DocumentDB and Amazon Redshift credentials in AWS Secrets Manager
- AWS Secrets Manager とは
- AWS Secrets Manager シークレットのローテーション
- Secrets Manager の Amazon DocumentDB 認証情報

ロールベースのアクセス制御 (RBAC) を使用したデータベースへのアクセス

Amazon DocumentDB (MongoDB 互換) の ロールベースアクセス制御 (RBAC)を使用して、ユーザーがデータベースで実行できるアクションへのアクセスを制限できます。RBAC は、1 つ以上のロールをユーザーに付与することで機能します。これらのロールは、ユーザーがデータベースリソースに対して実行できるオペレーションを決定します。Amazon DocumentDB は、現在、`read`、`readWrite`、`readAnyDatabase`、`clusterAdmin` のようなデータベースレベルを対象とする組み込みロール、および要件に基づいて特定のアクションと詳細なリソースの両方をサポートしています。

RBAC の一般的ユースケースには、クラスター内のデータベースへの読み取り専用アクセス権を持つユーザーを作成して最小特権を強制したり、クラスター内の特定のデータベースにアクセスすることを単一のユーザーに許可するマルチテナントアプリケーションを設計したりすることが含まれます。

> **Note**
>
> 2020 年 3 月 26 日 より前に作成されたすべての新規ユーザーには `dbAdminAnyDatabase`、`readWriteAnyDatabase`、および `clusterAdmin` のロールが付与されています。すべての既存ユーザーを再評価し、必要に応じてロールを変更してクラスターに最小特権を適用することをお勧めします。

トピック

- RBAC の概念
- RBAC 組み込みロールの開始方法
- RBAC ユーザー定義ロールの開始方法
- ユーザーとして Amazon DocumentDB に接続する
- 一般的なコマンド
- 機能的な違い
- 制限
- ロールベースのアクセス制御 (RBAC) を使用したデータベースへのアクセス

RBAC の概念

ロールベースのアクセスコントロールに関連する重要な用語と概念を以下に示します。Amazon DocumentDB ユーザーの詳細については、Amazon DocumentDB ユーザーの管理 を参照してください。

- ユーザー - データベースに対して認証し、オペレーションを実行できる個別のエンティティ。
- パスワード - ユーザーの認証に使用されるシークレット。
- ロール - 1 つ以上のデータベースに対してアクションを実行することをユーザーに許可します。
- 管理データベース - ユーザーを保存して認証する先のデータベース。
- データベース (**db**) - ドキュメントを保存するためのコレクションを含むクラスター内の名前空間。

次のコマンドは、`sample-user` というユーザーを作成します。

```
db.createUser({user: "sample-user", pwd: "abc123", roles: [{role: "read", db: "sample-database"}]})
```

この例では、以下のようになっています：

- `user: "sample-user"` - ユーザー名を示します。
- `pwd: "abc123"` - ユーザーのパスワードを示します。
- `role: "read", "db: "sample-database"` - ユーザー `sample-user` に `sample-database` への読み取り許可を付与することを示します。




```
db.createUser({user: "sample-user", pwd: "abc123", roles: [{role: "read", db: "sample-database"}]})
```

次の例は、db.getUser(sample-user) を使用してユーザー sample-user を取得した後の出力を示しています。この例では、ユーザー sample-user は admin データベース内に存在しますが、データベース sample-database の読み取りロールを持っています。



ユーザーの作成時に、db フィールドを省略してロールを指定すると、Amazon DocumentDB は接続の発行先であるデータベースに暗黙でロールを帰属させます。たとえば、データベース sample-database に対して接続を発行し、次のコマンドを実行すると、ユーザー sample-user が admin データベース内に作成され、データベース sample-database への readWrite アクセス許可が付与されます。

```
db.createUser({user: "sample-user", pwd: "abc123", roles: ["readWrite"]})
```

このオペレーションによる出力は、次のようになります。

```
{
   "user":"sample-user",
   "roles":[
      {
         "db":"sample-database",
         "role":"readWrite"
      }
   ]
}
```

すべてのデータベースを対象とするロール (readAnyDatabase など) を持つユーザーを作成するには、ユーザーの作成時に admin データベースのコンテキストで操作をしているか、ユーザーの作成

時にロールのデータベースを明示的に指定する必要があります。admin データベースに対してコマンドを発行するには、コマンド use admin を使用します。詳細については、「一般的なコマンド」を参照してください。

RBAC 組み込みロールの開始方法

ロールベースのアクセスコントロールの使用を簡単に開始できるように、このセクションでは、異なるジョブ機能を持つ 3 人のユーザーのロールを作成して最小特権を強制するシナリオの例を示します。

- user1 は、クラスター内のすべてのデータベースの表示とアクセスを必要とする新しいマネージャーです。
- user2 は、同じクラスター内の 1 つのデータベース (sample-database-1) にのみアクセスする必要がある新しい従業員です。
- user3 は、同じクラスター内の、以前はアクセスできなかった別のデータベース (sample-database-2) の表示とアクセを必要とする既存の従業員です。

後で、user1 と user2 が退職し、この両方のユーザーのアクセス権を取り消す必要があります。

ユーザーを作成してロールを付与する場合、クラスターに対して認証を行うユーザーには、createUser と grantRole のアクションを実行できるロールが関連付けられている必要があります。たとえば、admin ロールと userAdminAnyDatabase ロールは両方とも、該当するアクションのアクセス許可を付与できます。ロール別のアクションについては、「ロールベースのアクセス制御 (RBAC) を使用したデータベースへのアクセス」を参照してください。

> **Note**
>
> Amazon DocumentDB では、create データベースに対してコマンドを発行するかどうかにかかわらず、ユーザーおよびロールのすべてのオペレーション (get、drop、grant、revoke、admin など) が admin データベース内で暗黙で実行されます。

まず、クラスター内の現在のユーザーとロールを確認するために、次の例に示すように show users コマンドを実行します。クラスター内の 2 人のユーザー、serviceadminとプライマリユーザーが表示されます。これらの 2 人のユーザーは常に存在し、削除することはできません。詳細については、「Amazon DocumentDB ユーザーの管理」を参照してください。

```
show users
```

user1 に対して、次のコマンドを使用し、クラスター全体のすべてのデータベースに対する読み取りおよび書き込みアクセス権を持つロールを作成します。

```
db.createUser({user: "user1", pwd: "abc123", roles: [{role: "readWriteAnyDatabase", db:
 "admin"}]})
```

このオペレーションによる出力は、次のようになります。

```
{
   "user":"user1",
   "roles":[
      {
         "role":"readWriteAnyDatabase",
         "db":"admin"
      }
   ]
}
```

user2 に対して、次のコマンドを使用し、データベース sample-database-1 への読み取り専用アクセス権を持つロールを作成します。

```
db.createUser({user: "user2", pwd: "abc123", roles: [{role: "read", db: "sample-
database-1"}]})
```

このオペレーションによる出力は、次のようになります。

```
{
   "user":"user2",
   "roles":[
      {
         "role":"read",
         "db":"sample-database-1"
      }
   ]
}
```

user3 を既存のユーザーとするシナリオをシミュレートするには、まずユーザー user3 を作成し、次に user3 新しいロールを割り当てます。

```
db.createUser({user: "user3", pwd: "abc123", roles: [{role: "readWrite", db: "sample-
database-1"}]})
```

このオペレーションによる出力は、次のようになります。

```
{
   "user":"user3",
   "roles":[
      {
         "role":"readWrite",
         "db":"sample-database-1"
      }
   ]
}
```

ユーザー user3 を作成したので、この user3 に対して sample-database-2 の read ロールを割り当てます。

```
db.grantRolesToUser("user3", [{role: "read", db: "sample-database-2"}])
```

最後に、user1 と user2 が退職し、この両ユーザーのクラスターに対するアクセス権を取り消す必要があります。これを行うには、次のようにユーザーを削除します。

```
db.dropUser("user1")
db.dropUser("user2")
```

すべてのユーザーが適切なロールを持っていることを確認するには、次のコマンドを使用してすべてのユーザーを一覧表示します。

```
show users
```

このオペレーションによる出力は、次のようになります。

```
{
   "_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 ユーザー定義ロールの開始方法

このセクションでは、ユーザー定義の役割を開始するのに役立つように、ジョブ機能が異なる 3 人のユーザーにロールを作成して最小特権を適用するシナリオの例について説明します。

この例では、以下が適用されます。

- user1 は、クラスター内のすべてのデータベースの表示とアクセスを必要とする新しいマネージャーです。
- user2 は、同じクラスター内の 1 つのデータベース sample-database-1 にのみアクセスする必要がある新しい従業員です。

- user3 は、同じクラスター内で以前はアクセスできなかった sample-database-2、別のデータベースの col2 という特定のコレクションを表示してアクセスする必要がある既存の従業員です。
- user1 に対して、次のコマンドを使用し、クラスター全体のすべてのデータベースに対する読み取りおよび書き込みアクセス権を持つロールを作成します。

```
db.createUser(
 {
     user: "user1", pwd: "abc123",
     roles: [{role: "readWriteAnyDatabase", db: "admin"}]
 }
 )
```

このオペレーションによる出力は、次のようになります。

```
{
   "user":"user1",
   "roles":[
      {
         "role":"readWriteAnyDatabase",
         "db":"admin"
      }
   ]
}
```

user2 の場合、次のコマンドを使用して、データベース sample-database-1 内のすべてのコレクションに対する「検索」権限を持つロールを作成します。このロールは、関連付けられたすべてのユーザーが検索クエリのみを実行できるようにすることに注意してください。

```
db.createRole(
{
    role: "findRole",
    privileges: [
    {
        resource: {db: "sample-database-1", collection: ""}, actions: ["find"]
    }],
    roles: []
}
)
```

このオペレーションによる出力は、次のようになります。

```
{
   "role":"findRole",
   "privileges":[
      {
         "resource":{
            "db":"sample-database-1",
            "collection":""
         },
         "actions":[
            "find"
         ]
      }
   ],
   "roles":[

   ]
}
```

次に、ユーザー (user2) を作成し、最近作成したロール findRole をユーザーに接続します。

```
db.createUser(
{
    user: "user2",
    pwd: "abc123",
    roles: []
})

 db.grantRolesToUser("user2",["findRole"])
```

user3 を既存のユーザーとするシナリオをシミュレートするには、まずユーザー user3 を作成し、次のステップで user3 に割り当てる collectionRole という新しいロールを作成します。

そして、新しいロールを user3 に割り当てることができます。この新しいロールにより、user3 の 1 つの特定のコレクション col2 へのアクセスを挿入、更新、削除、および検索できる sample-database-2 を許可します。

```
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: []
}
)
```

このオペレーションによる出力は、次のようになります。

```
{
   "role":"collectionRole",
   "privileges":[
      {
         "resource":{
            "db":"sample-database-2",
            "collection":"col2"
         },
         "actions":[
            "find",
            "update",
            "insert",
            "remove"
         ]
      }
   ],
   "roles":[

   ]
}
```

ユーザー user3 を作成したので、user3 にロール collectionFind を付与できます。

```
db.grantRolesToUser("user3",["collectionRole"])
```

最後に、user1 と user2 が退職し、この両ユーザーのクラスターに対するアクセス権を取り消す必要があります。これを行うには、次のようにユーザーを削除します。

```
db.dropUser("user1")
db.dropUser("user2")
```

すべてのユーザーが適切なロールを持っていることを確認するには、次のコマンドを使用してすべてのユーザーを一覧表示します。

```
show users
```

このオペレーションによる出力は、次のようになります。

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

```
}
```

ユーザーとして Amazon DocumentDB に接続する

Amazon DocumentDB クラスターに接続するときは、特定のデータベースのコンテキストで接続します。デフォルトでは、接続文字列でデータベースを指定しない場合、test データベースのコンテキストでクラスターに自動的に接続されます。insert や find などのすべてのコレクションレベルのコマンドは、test データベース内のコレクションに対して発行されます。

コンテキスト内にあるデータベース、つまりコマンドの発行を確認するには、以下のように、mongo シェル内の db コマンドを実行します。

クエリ:

```
db
```

出力:

```
test
```

デフォルトの接続が test データベースのコンテキストに存在する場合でも、この接続に関連付けられているユーザーが test データベースに対するアクションの実行を許可されているとは限りません。前のシナリオ例で、sample-database-1 データベースに対する readWrite ロールを持つユーザー user3 として認証する場合、接続のデフォルトのコンテキストは test データベースです。ただし、test データベースのコレクションにドキュメントを挿入しようとすると、認可の失敗エラーメッセージが表示されます。これは、このコマンドをデータベースに対して実行する権限がユーザーに許可されていないことが原因です。

クエリ:

```
db
```

出力:

```
test
```

クエリ:

```
db.col.insert({x:1})
```

出力:

```
WriteCommandError({ "ok" : 0, "code" : 13, "errmsg" : "Authorization failure" })
```

接続のコンテキストを sample-database-1 データベースに変更すると、ユーザーが書き込み権限を持つコレクションに書き込むことができます。

クエリ:

```
use sample-database-1
```

出力:

```
switched to db sample-database-1
```

クエリ:

```
db.col.insert({x:1})
```

出力:

```
WriteResult({ "nInserted" : 1})
```

特定のユーザーとしてクラスターに対して認証を行う場合は、接続文字列でデータベースを指定することもできます。これにより、ユーザーが admin データベースに対して認証された後で use コマンドを実行する必要がなくなります。

次の接続文字列では、admin データベースに対してユーザーを認証しますが、接続のコンテキストは sample-database-1 データベースになります。

```
mongo "mongodb://user3:abc123@sample-cluster.node.us-east-1.docdb.amazonaws.com:27017/
sample-database-2"
```

一般的なコマンド

このセクションでは、Amazon DocumentDB でロールベースのアクセスコントロールを使用する一般的なコマンドの例を示します。ユーザーとロールを作成および変更するには、admin データベースのコンテキストで操作する必要があります。use admin コマンドを使用して、admin データベースに切り替えることができます。

> **Note**
>
> ユーザーおよびロールに対する変更は、admin データベース内で暗黙で実行されます。すべてのデータベースを対象とするロール (readAnyDatabase など) を持つユーザーを作成するには、ユーザーの作成時に admin データベース (use admin) のコンテキストで操作をしているか、ユーザーの作成時にロールのデータベースを明示的に指定する必要があります (このセクションの例 2 を参照)。

例 1: データベース read 用の foo ロールを持つユーザーを作成します。

```
db.createUser({user: "readInFooBar", pwd: "abc123", roles: [{role: "read", db:
 "foo"}]})
```

このオペレーションによる出力は、次のようになります。

```
{
   "user":"readInFooBar",
   "roles":[
      {
         "role":"read",
         "db":"foo"
      }
   ]
}
```

例 2: すべてのデータベースへの読み取りアクセス権を持つユーザーを作成します。

```
db.createUser({user: "readAllDBs", pwd: "abc123", roles: [{role: "readAnyDatabase", db:
 "admin"}]})
```

このオペレーションによる出力は、次のようになります。

```
{
   "user":"readAllDBs",
   "roles":[
      {
         "role":"readAnyDatabase",
         "db":"admin"
      }
```

```
   ]
}
```

例 3: 新しいデータベースの既存のユーザーに read ロールを付与します。

```
db.grantRolesToUser("readInFooBar", [{role: "read", db: "bar"}])
```

例 4: ユーザーのロールを更新します。

```
db.updateUser("readInFooBar", {roles: [{role: "read", db: "foo"}, {role: "read", db:
 "baz"}]})
```

例 5: ユーザーのデータベースへのアクセスを取り消します。

```
db.revokeRolesFromUser("readInFooBar", [{role: "read", db: "baz"}])
```

例 6: 組み込みロールを説明します。

```
db.getRole("read", {showPrivileges:true})
```

このオペレーションによる出力は、次のようになります。

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

例 7: クラスターからユーザーを削除します。

```
db.dropUser("readInFooBar")
```

このオペレーションによる出力は、次のようになります。

```
true
```

例 8: 特定のコレクションに対する読み取りおよび書き込みアクセス権を持つロールを作成する

```
db.createRole(
{
    role: "collectionRole",
    privileges: [
    {
```

```
        resource: {db: "sample-database-2", collection: "col2"}, actions: ["find",
 "update", "insert", "remove"]
    }],
    roles: []
}
)
```

このオペレーションによる出力は、次のようになります。

```
{
   "role":"collectionRole",
   "privileges":[
      {
         "resource":{
            "db":"sample-database-2",
            "collection":"col2"
         },
         "actions":[
            "find",
            "update",
            "insert",
            "remove"
         ]
      }
   ],
   "roles":[

   ]
}
```

例 9: ユーザーを作成し、ユーザー定義ロールを割り当てる

```
db.createUser(
{
    user: "user3",
    pwd: "abc123",
    roles: []
})

db.grantRolesToUser("user3",["collectionRole"])
```

例 10: ユーザー定義ロールに追加権限を付与する

```
db.grantPrivilegesToRole(
  "collectionRole",
  [
    {
      resource: { db: "sample-database-1", collection: "col1" },
      actions: ["find", "update", "insert", "remove"]
    }
  ]
)
```

例 11: ユーザー定義ロールから権限を削除する

```
db.revokePrivilegesFromRole(
  "collectionRole",
  [
    {
      resource: { db: "sample-database-1", collection: "col2" },
      actions: ["find", "update", "insert", "remove"]
    }
  ]
)
```

例 12: 既存のユーザー定義ロールを更新する

```
db.updateRole(
  "collectionRole",
  {
    privileges: [
    {
        resource: {db: "sample-database-3", collection: "sample-collection-3"},
 actions: ["find", "update", "insert", "remove"]
    }],
    roles: []
  }
)
```

機能的な違い

Amazon DocumentDB では、ユーザーとロールの定義が admin データベースに保存され、ユーザーは admin データベースに対して認証されます。この機能は MongoDB Community Edition とは異なりますが、MongoDB Atlas と一貫しています。

Amazon DocumentDB は、変更ストリームもサポートしています。変更ストリームは、クラスターのコレクション内で発生する変更イベントの時系列シーケンスを提供します。`listChangeStreams` アクションは、クラスターレベル (つまり、すべてのデータベースを対象に) 適用されます。`modifyChangeStreams` アクションは、データベースレベルで適用されます。

制限

次のテーブルは Amazon DocumentDB におけるロールベースアクセスコントロールの制限を含む

説明	制限
クラスターあたりのユーザー数	1,000
ユーザーに関連付けられているロールの数	1,000
ユーザー定義ロールの数	100
権限に関連付けられているリソースの数	100

ロールベースのアクセス制御 (RBAC) を使用したデータベースへのアクセス

ロールベースのアクセスコントロールでは、ユーザーを作成して 1 つ以上のロールを付与し、ユーザーがデータベースやクラスターで実行できるオペレーションを指定できます。

Amazon DocumentDB で現在サポートされている組み込みロールは次のとおりです。

> **Note**
>
> Amazon DocumentDB 4.0 と 5.0 では、それぞれ `ListCollection` および `ListDatabase` コマンドでは、オプションで `authorizedCollections` および `authorizedDatabases` パラメータを使用して、ユーザーが権限を持っているコレクションとデータベースを一覧表示し、`listCollections` および `listDatabase` ロールを要求するアクセスをします。また、ユーザーは `KillCursor` ロールを要求すねことなく、自分のカーソルを強制終了できるようになりました。

Database user

ロール名	説明	アクション
read	指定したデータベースへの読み取りアクセス権をユーザーに付与します。	changeStreams collStats dbStats find killCursors listIndexes listCollections
readWrite	指定したデータベースへの読み取りおよび書き込みアクセス権をユーザーに付与します。	read アクセス許可のすべてのアクション。 createCollection dropCollection createIndex dropIndex insert killCursors listIndexes listCollections remove

ロール名	説明	アクション
		update

Cluster user

ロール名	説明	アクション
readAnyDatabase	クラスター内のすべてのデータベースへの読み取りアクセス権をユーザーに付与します。	read アクセス許可のすべてのアクション。 listChangeStreams listDatabases
readWriteAnyDatabase	クラスター内のすべてのデータベースへの読取りおよび書込みアクセス権をユーザーに付与します。	readWrite アクセス許可のすべてのアクション。 listChangeStreams listDatabases
userAdminAnyDatabase	指定したデータベースに対するユーザーのロールや特権を割り当てたり変更したりできる権限をユーザーに付与します。	changeCustomData changePassword createUser dropRole dropUser grantRole listDatabases

ロール名	説明	アクション
		revokeRole viewRole viewUser
dbAdminAnyDatabase	指定したデータベースに対してデータベース管理ロールを実行する権限をユーザーに付与します。	dbAdmin アクセス許可のすべてのアクション。 dropCollection listDatabases listChangeStreams modifyChangeStreams

Superuser

ロール名	説明	アクション
root	すべてのロール (readWriteAnyDatabase 、dbAdminAnyDatabase 、userAdminAnyDatabase 、clusterAdmin 、restore、backup) のリソースとオペレーションに対するアクセス権をユーザーに付与します。	readWriteAnyDatabase 、dbAdminAnyDatabase 、userAdminAnyDatabase 、clusterAdmin 、restore および backup のすべてのアクション。

Database administrator

ロール名	説明	アクション
dbAdmin	指定したデータベースに対して管理タスクを実行する権限をユーザーに付与します。	bypassDocumentValidation collMod collStats createCollection createIndex dropCollection dropDatabase dropIndex dbStats find killCursors listIndexes listCollections modifyChangeStreams
dbOwner	dbAdmin ロールと readWrite ロールを組み合わせて、指定したデータベースに対する管理タスクを実行する権限をユーザーに付与します。	dbAdmin と readWrite のすべてのアクション。

Cluster administrator

ロール名	説明	アクション
clusterAdmin	clusterManager 、clusterMonitor 、および hostManager の各ロールを組み合わせで、最大のクラスター管理アクセス権をユーザーに付与します。	clusterManager 、clusterMonitor 、および hostManager のすべてのアクション。 listChangeStreams dropDatabase modifyChangeStreams
clusterManager	指定したクラスターに対して管理とモニタリングのアクションを実行する権限をユーザーに付与します。	listChangeStreams listSessions modifyChangeStreams replSetGetConfig
clusterMonitor	モニタリングツールへの読み取り専用アクセス権をユーザーに付与します。	collStats dbStats find getParameter hostInfo indexStats

ロール名	説明	アクション
		killCursors listChangeStreams listCollections listDatabases listIndexes listSessions replSetGetConfig serverStatus top
hostManager	サーバーをモニタリングおよび管理する権限をユーザーに付与します。	auditConfigure killCursors killAnyCursor killAnySession killop

Backup administrator

ロール名	説明	アクション
backup	データのバックアップに必要なアクセス権をユーザーに付与します。	getParameter insert

ロール名	説明	アクション
		find listChangeStreams listCollections listDatabases listIndexes update

ロール名	説明	アクション
restore	データの復元に必要なアクセス権をユーザーに付与します。	bypassDocumentValidation changeCustomData changePassword collMod createCollection createIndex createUser dropCollection dropRole dropUser getParameter grantRole find insert listCollections modifyChangeStreams revokeRole

ロール名	説明	アクション

		`remove`
		`viewRole`
		`viewUser`
		`update`

Amazon DocumentDB TLS 証明書の更新

トピック

- アプリケーションと Amazon DocumentDB クラスターの更新
- サーバー証明書の自動ローテーション
- トラブルシューティング
- よくある質問

Amazon DocumentDB クラスターの認証局 (CA) 証明書は、2024 年 8 月に更新されました。TLS (Transport Layer Security) が有効 (デフォルト設定) になっている クラスターを使用しており、クライアントアプリケーション証明書とサーバー証明書を更新していない場合は、アプリケーションと Amazon DocumentDB クラスターとの接続問題を軽減するために次の手順を実行する必要があります。

- ステップ 1: 新しい CA 証明書をダウンロードしてアプリケーションを更新する
- ステップ 2: サーバー証明書を更新する

CA とサーバーの証明書は、Amazon DocumentDB のための標準的なメンテナンスおよびセキュリティのベストプラクティスの一環として更新されています。クライアントアプリケーションは、新しい CA 証明書をトラストストアに追加する必要があり、この有効期限より前に新しい CA 証明書を使用するために既存の Amazon DocumentDB インスタンスを更新する必要があります。

アプリケーションと Amazon DocumentDB クラスターの更新

このセクションの手順に従って、アプリケーションの CA 証明書バンドル (ステップ 1) とクラスターのサーバー証明書 (ステップ 2) を更新します。本番環境に変更を適用する前に、開発環境またはステージング環境でこれらの手順をテストすることを強くお勧めします。

> **Note**
>
> Amazon DocumentDB クラスターがある各 AWS リージョン でステップ 1 と 2 を完了する必要があります。

ステップ 1: 新しい CA 証明書をダウンロードしてアプリケーションを更新する

新しい CA 証明書をダウンロードし、新しい CA 証明書を使用して Amazon DocumentDB への TLS 接続を作成するようにアプリケーションを更新します。https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem から新しい CA 証明書バンドルをダウンロードします。このオペレーションにより、`global-bundle.pem` という名前のファイルがダウンロードされます。

> **Note**
>
> 古い CA 証明書 (`rds-ca-2019-root.pem`) と新しい CA 証明書 (`rds-ca-rsa2048-g1`、`rds-ca-rsa4096-g1`、`rds-ca-ecc384-g1`) の両方を含むキーストアにアクセスしている場合は、キーストアで `global-bundle` が選択されるか確認してください。

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

次に、新しい証明書バンドルを使用するようにアプリケーションを更新します。新しい CA バンドルには、古い CA 証明書 (rds-ca-2019) と新しい CA 証明書 (rds-ca-rsa2048-g1, rds-ca-rsa4096-g1, rds-ca-ecc384-g1) の両方が含まれています。新しい CA バンドルに両方の CA 証明書を使用すると、2 つの手順でアプリケーションとクラスターを更新できます。

Java アプリケーションの場合、新しい CA 証明書を使用して新しい信頼ストアを作成する必要があります。手順については、「TLS が有効な場合の接続」トピックの Java タブを参照してください。

アプリケーションで最新の CA 証明書バンドルを使用していることを確認するには、「最新の CA バンドルを使用していることを確認するにはどうすればよいですか?」を参照してください。アプリ

ケーションで最新の CA 証明書バンドルをすでに使用している場合は、ステップ 2 に進むことができます。

アプリケーションで CA バンドルを使用する例については、「Encrypting data in transit」および「TLS が有効な場合の接続」を参照してください。

> **Note**
>
> 現在、MongoDB Go ドライバー 1.2.1 は、`sslcertificateauthorityfile` で 1 つの CA サーバー証明書しか受け入れません。TLS が有効になっている場合の Go を使用した Amazon DocumentDB への接続TLS が有効な場合の接続については、「」を参照してください。

ステップ 2: サーバー証明書を更新する

新しい CA バンドルを使用するようにアプリケーションを更新したら、次のステップとして Amazon DocumentDB クラスター内の各インスタンスを変更してサーバー証明書を更新します。新しいサーバー証明書を使用するようにインスタンスを変更するには、次の手順を参照してください。

Amazon DocumentDB には DB インスタンスの DB サーバー証明書に署名するために以下の CA が用意されています。

- rds-ca-ecc384-g1 — ECC 384 プライベートキーアルゴリズムと SHA384 署名アルゴリズムを備えた認証局を使用します。この CA はサーバー証明書の自動ローテーションをサポートします。これは現在、Amazon DocumentDB 4.0 および 5.0 でのみサポートされています。
- rds-ca-rsa2048-g1 — ほとんどの で RSA 2048 プライベートキーアルゴリズムと SHA256 署名アルゴリズムで認証機関を使用します AWS リージョン。この CA はサーバー証明書の自動ローテーションをサポートします。
- rds-ca-rsa2048-g1 — RSA 4096 プライベートキーアルゴリズムと SHA256 署名アルゴリズムを備えた認証局を使用します。この CA はサーバー証明書の自動ローテーションをサポートします。

> **Note**
>
> を使用している場合は AWS CLI、describe-certificates を使用して、前のセクションに記載されている認証機関の有効性を確認できます。

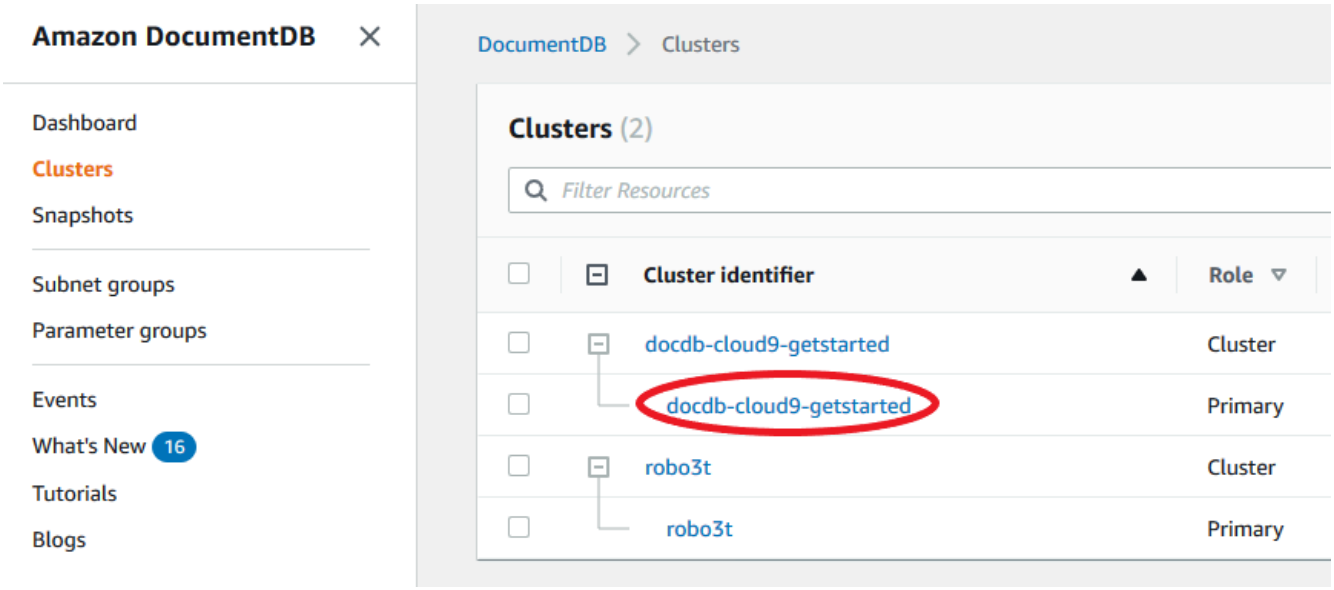
これらの CA 証明書は、地域およびグローバル証明書バンドルに含まれています。rds-ca-rsa2048-g1, rds-ca-rsa4096-g1, or rds-ca-ecc384-g1 のいずれかの CA をデータベースに使用すると、Amazon DocumentDB はそのデータベース上で DB サーバー証明書を管理します。Amazon DocumentDB は、DB サーバー証明書の有効期限が切れる前に証明書のローテーションを行います。

> **Note**
>
> お使いのクラスターが次のエンジンパッチバージョンで実行されている場合、Amazon DocumentDB は証明書ローテーションのために再起動を必要としません。
>
> - Amazon DocumentDB 3.6: 1.0.208662 以降
> - Amazon DocumentDB 4.0: 2.0.10179 以降
> - Amazon DocumentDB 5.0: 3.0.4780 以降
>
> Amazon DocumentDB エンジンパッチの最新バージョンは、次のコマンドを実行して確認できます: `db.runCommand({getEngineVersion: 1})`。
> サーバー証明書を更新する前に、ステップ 1 を完了しておく必要があります。

Using the AWS マネジメントコンソール

次の手順を実行し、 AWS マネジメントコンソールを使って既存の Amazon DocumentDB インスタンスのための古いサーバー証明書を特定するために、次のステップを完了します。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. 画面の右上隅にあるリージョンのリストで、クラスターが存在する AWS リージョン を選択します。
3. コンソールの左側のナビゲーションペインで、Clusters を選択します。
4. 古いサーバー証明書 (`rds-ca-2019`) に残っているインスタンスを特定する必要がある場合があります。これは [クラスター] 表の右端にある [証明機関] 列で行うことができます。
5. [クラスター] 表の左端に [クラスター識別子] の列があります。インスタンスは、次のスクリーンショットのようにクラスターの下に一覧表示されます。



6. 興味がある対象のインスタンスの左側にあるチェックボックスをチェックします。
7. アクション を選択してから、変更 を選択します。
8. [証明機関] で、このインスタンスの新しいサーバー証明書 (`rds-ca-rsa2048-g1`) を選択します。
9. 次のページで変更の概要を確認できます。接続の中断を避けるためにインスタンスを変更する前に、アプリケーションが最新の証明書 CA バンドルを使用していることを確認するよう通知する、追加のアラートがあることに注意してください。
10. 次のメンテナンス期間中に変更を適用するか、すぐに適用するかを選択できます。サーバー証明書をすぐに変更する場合は、[すぐに適用] オプションを使用します。
11. [インスタンスの変更] を選択して、更新を完了します。

Using the AWS CLI

次の手順を実行し、 AWS CLIを使って既存の Amazon DocumentDB インスタンスのための古いサーバー証明書を特定して更新します。

1. インスタンスをすぐに変更するには、クラスターの各インスタンスに対して次のコマンドを実行します。

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier>
 --ca-certificate-identifier rds-ca-rsa2048-g1 --apply-immediately
```

2. クラスターの次回のメンテナンスウィンドウで新しい CA 証明書を使用するようにクラスター内のインスタンスを変更するには、クラスター内のインスタンスごとに次のコマンドを実行します。

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier>
 --ca-certificate-identifier rds-ca-rsa2048-g1 --no-apply-immediately
```

サーバー証明書の自動ローテーション

Amazon DocumentDB はサーバー証明書の自動ローテーションをサポートします。サーバー証明書は、各クラスターインスタンスに発行されたリーフ証明書です。ルート CA 証明書とは対照的に、サーバー証明書の有効性は短く (12 か月)、Amazon DocumentDB はユーザーのアクションなしで自動的にローテーションを処理します。Amazon DocumentDB はこの自動ローテーションに同じルート CA を使用するため、新しい CA バンドルをダウンロードする必要はありません。

> **Important**
>
> Amazon DocumentDB クラスターに接続するときは、各サーバー証明書を直接信頼するのではなく、ルート CA バンドルを信頼します。これにより、サーバー証明書がローテーションされた後の接続エラーを防ぐことができます。「TLS が有効な場合の接続」を参照してください。

Amazon DocumentDB は、サーバー証明書の半減期になった時点で、希望するメンテナンス期間にサーバー証明書のローテーションを試みます。新しいサーバー証明書は、12 か月間有効です。

describe-db-engine-versions コマンドを使用して、SupportsCertificateRotationWithoutRestart フラグを点検することで、再起動なしでエンジンバージョンが証明書のローテーションをサポートするかどうかを特定します。

> **Note**
>
> お使いのクラスターが次のエンジンパッチバージョンで実行されている場合、Amazon DocumentDB は再起動なしでサーバー証明書のローテーションをサポートします。
>
> - Amazon DocumentDB 3.6: 1.0.208662 以降
> - Amazon DocumentDB 4.0: 2.0.10179 以降
> - Amazon DocumentDB 5.0: 3.0.4780 以降

Amazon DocumentDB エンジンパッチの最新バージョンは、このコマンドを実行して確認できます: `db.runCommand({getEngineVersion: 1})`。
古いエンジンパッチバージョンを使用している場合、Amazon DocumentDB はサーバー証明書をローテーションし、お客様の希望するメンテナンスウィンドウでデータベース再起動イベントをスケジュールします。

トラブルシューティング

証明書のローテーションの一環としてクラスターへの接続に問題がある場合は、以下を試してください。

- クライアントが最新の証明書バンドルを使用していることを確認します。「最新の CA バンドルを使用していることを確認するにはどうすればよいですか?」を参照してください。
- インスタンスが最新の証明書を使用していることを確認します。「古い/新しいサーバー証明書を使用している Amazon DocumentDB インスタンスを確認するにはどうすればいいですか。」を参照してください。
- 最新の証明書 CA がアプリケーションで使用されていることを確認します。Java や Go などの一部のドライバーは、複数の証明書を証明書バンドルから信頼ストアにインポートするために、追加のコードを必要とします。TLS を使用した Amazon DocumentDB への接続のさらなる詳細については、プログラムによる Amazon DocumentDB への接続 を参照してください。
- サポートへのお問い合わせ。ご質問や問題がある場合は、サポート にお問い合わせください。

よくある質問

次に、TLS 証明書に関していくつかのよくある質問に対する回答を示します。

質問や問題がある場合はどうしたらいいですか?

ご質問や問題がある場合は、サポート にお問い合わせください。

TLS を使用して Amazon DocumentDB クラスターに接続しているかどうかを知るにはどうすればよいですか。

クラスターが TLS を使用しているかどうかを判断するには、クラスターのクラスターパラメータグループの `tls` パラメータを調べます。`tls` パラメータが `enabled` に設定されている場合は、TLS

証明書を使用してクラスターに接続しています。詳細については、「Amazon DocumentDB クラスターパラメータグループの管理」を参照してください。

CA 証明書とサーバー証明書を更新する理由

Amazon DocumentDB の CA 証明書とサーバー証明書は、Amazon DocumentDB の標準的なメンテナンスおよびセキュリティのベストプラクティスの一環として更新されてきました。

有効期限までにアクションしないと、どうなりますか?

Amazon DocumentDB クラスターへの接続に TLS を使用していて、CA 証明書が失効している場合、TLS 経由で接続するアプリケーションは Amazon DocumentDB クラスターと通信できなくなります。

Amazon DocumentDB は、有効期限が切れる前にデータベース証明書を自動的にローテーションしません。有効期限の前後を問わず、新しい CA 証明書を使用するようにアプリケーションとクラスターを更新する必要がある。

古い/新しいサーバー証明書を使用している Amazon DocumentDB インスタンスを確認するにはどうすればいいですか。

古いサーバー証明書をまだ使用している Amazon DocumentDB インスタンスを特定するには、Amazon DocumentDB AWS マネジメントコンソール または のいずれかを使用できます AWS CLI。

の使用 AWS マネジメントコンソール

古い証明書を使用しているクラスター内のインスタンスを特定するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。

2. 画面の右上隅にあるリージョンのリストで、インスタンスが存在する AWS リージョン を選択します。
3. コンソールの左側のナビゲーションペインで、Clusters を選択します。
4. [証明機関] 列 (表の右端近く) には、古いサーバー証明書 (`rds-ca-2019`) および新しいサーバー証明書 (`rds-ca-rsa2048-g1`) に残っているインスタンスが表示されます。

の使用 AWS CLI

古いサーバー証明書を使用しているクラスター内のインスタンスを特定するには、以下を指定して `describe-db-clusters` コマンドを使用します。

```
aws docdb describe-db-instances \
    --filters Name=engine,Values=docdb \
    --query 'DBInstances[*].
{CertificateVersion:CACertificateIdentifier,InstanceID:DBInstanceIdentifier}'
```

Amazon DocumentDB クラスター内の個々のインスタンスを変更してサーバー証明書を更新するにはどうすればよいですか。

特定のクラスター内のすべてのインスタンスのサーバー証明書を同時に更新します。クラスター内のインスタンスを変更するには、コンソールまたは AWS CLIを使用できます。

> **Note**
>
> サーバー証明書を更新する前に、ステップ 1 を完了しておく必要があります。

の使用 AWS マネジメントコンソール

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. 画面の右上隅にあるリージョンのリストで、クラスターが存在する AWS リージョン を選択します。
3. コンソールの左側のナビゲーションペインで、Clusters を選択します。
4. [証明機関] 列 (表の右端近く) には、古いサーバー証明書 (`rds-ca-2019`) に残っているインスタンスが表示されます。
5. [クラスター] 表の [クラスター識別子] で、変更するインスタンスを選択します。
6. アクション を選択してから、変更 を選択します。
7. [証明機関] で、このインスタンスの新しいサーバー証明書 (`rds-ca-rsa2048-g1`) を選択します。
8. 次のページで変更の概要を確認できます。接続の中断を避けるためにインスタンスを変更する前に、アプリケーションが最新の証明書 CA バンドルを使用していることを確認するよう通知する、追加のアラートがあることに注意してください。

9. 次のメンテナンス期間中に変更を適用するか、すぐに適用するかを選択できます。
10. [インスタンスの変更] を選択して、更新を完了します。

の使用 AWS CLI

次の手順を実行し、 AWS CLIを使って既存の Amazon DocumentDB インスタンスのための古いサーバー証明書を特定して更新します。

1. インスタンスをすぐに変更するには、クラスターの各インスタンスに対して次のコマンドを実行します。

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier> --
ca-certificate-identifier rds-ca-rsa2048-g1 --apply-immediately
```

2. クラスターの次回のメンテナンスウィンドウで新しい CA 証明書を使用するようにクラスター内のインスタンスを変更するには、クラスター内のインスタンスごとに次のコマンドを実行します。

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier> --
ca-certificate-identifier rds-ca-rsa2048-g1 --no-apply-immediately
```

既存のクラスターに新しいインスタンスを追加するとどうなりますか?

作成される新しいインスタンスはすべて古いサーバー証明書を使用し、古い CA 証明書を使用する TLS 接続が必要です。2024 年 1 月 25 日以降に作成された新しい Amazon DocumentDB インスタンスでは、デフォルトで新しい証明書 rds-ca-rsa2048-g1 が使用されます。

クラスターにインスタンスの置き換えまたはフェイルオーバーがある場合はどうなりますか?

クラスターにインスタンスの置き換えがある場合、作成される新しいインスタンスは、そのインスタンスが以前使用していたものと同じサーバー証明書を引き続き使用します。すべてのインスタンスのサーバー証明書を同時に更新することをお勧めします。クラスターでフェイルオーバーが発生した場合、新しいプライマリのサーバー証明書が使用されます。

クラスターへの接続に TLS を使用していない場合でも、各インスタンスを更新する必要がありますか?

TLS を有効にすることを強くお勧めします。なお、TLS を有効にしない場合でも、今後 TLS を使用してクラスターに接続する予定がある場合は、Amazon DocumentDB インスタンス上で証明書をローテーションすることをお勧めします。Amazon DocumentDB クラスターへの接続に TLS を使用する予定がない場合、対応は必要ありません。

現在、クラスターへの接続に TLS を使用していませんが、将来的に計画している場合、どうすればよいですか。

2024 年 1 月より前にクラスターを作成した場合は、前のセクションの ステップ 1 と ステップ 2 に従って、アプリケーションで更新済みの CA バンドルを使用していること、および各 Amazon DocumentDB インスタンスで最新のサーバー証明書を使用していることを確認してください。2024 年 1 月 25 日以降にクラスターを作成した場合、クラスターにはすでに最新のサーバー証明書 (rds-ca-rsa2048-g1) があります。アプリケーションで最新の CA バンドルを使用していることを確認するには、「クラスターへの接続に TLS を使用していない場合でも、各インスタンスを更新する必要がありますか?」を参照してください。

2024 年 8 月以降に締め切りを延長することはできますか?

アプリケーションが TLS 経由で接続している場合、締め切りを延長することはできません。

最新の CA バンドルを使用していることを確認するにはどうすればよいですか?

最新のバンドルがあることを確認するには、次のコマンドを使用します。このコマンドを実行するには、Java がインストールされていて、Java ツールがシェルの PATH 変数に含まれている必要があります。詳細については、「 Java の使用」を参照してください。

macOS と Amazon Linux

```
keytool -printcert -v -file global-bundle.pem
```

Windows

```
keytool -printcert -v -file global-bundle.p7b
```

CA バンドルの名前に「RDS」が表示されるのはなぜですか?

証明書マネージメントのような特定の管理機能では、Amazon DocumentDB は Amazon RDS (Amazon Relational Database Service) と共有の運用テクノロジーを使用します。

新しい証明書の有効期限はいつですか?

新しいサーバー証明書は (一般的に) 以下のように期限切れになります。

- rds-ca-rsa2048-g1: 2061 年
- rds-ca-rsa4096-g1: 2121 年
- rds-ca-ecc384-g1: 2121 年 に有効期限 となります

証明書の有効期限が切れる前にアクションを実行しないと、どのようなエラーが表示されますか ?

エラーメッセージはお使いのドライバーによって異なります。一般に、「証明書の有効期限切れです」の文字列を含む証明書検証エラーが表示されます。

新しいサーバー証明書を適用した場合、古いサーバー証明書に戻すことはできますか。

インスタンスを古いサーバー証明書に戻す必要がある場合は、クラスター内のすべてのインスタンスに対して行います。 AWS マネジメントコンソール または を使用して、クラスター内の各インスタンスのサーバー証明書を元に戻すことができます AWS CLI。

の使用 AWS マネジメントコンソール

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. 画面の右上隅にあるリージョンのリストで、クラスターが存在する AWS リージョン を選択します。
3. コンソールの左側のナビゲーションペインで、Clusters を選択します。
4. [クラスター] 表の [クラスター識別子] で、変更するインスタンスを選択します。[アクション] を選択してから、[変更] を選択します。
5. [証明機関] で、古いサーバー証明書を選択します (`rds-ca-2019`)。
6. [続行] を選択して、変更の概要を表示します。

7. この結果ページでは、変更を次のメンテナンスウィンドウで適用するようにスケジュールするか、変更をすぐに適用するかを選択できます。選択を行い、[Modify instance (インスタンスの変更)] を選択します。

> **Note**
>
> 変更の即時適用を選択した場合、保留中の変更キューにあるすべての変更も同様に適用されます。ダウンタイムを必要とする保留中の変更がある場合、このオプションを選択すると予想外のダウンタイムが発生することがあります。

の使用 AWS CLI

```
aws docdb modify-db-instance --db-instance-identifier <db_instance_name> ca-
certificate-identifier rds-ca-2019 <--apply-immediately | --no-apply-immediately>
```

`--no-apply-immediately` を選択した場合、変更はクラスターの次のメンテナンス期間中に適用されます。

スナップショットまたはポイントインタイム復元から復元した場合、新しいサーバー証明書が含まれていますか?

2024 年 8 月以降にスナップショットを復元するか、ポイントインタイム復元を実行すると、作成される新しいクラスターでは新しい CA 証明書が使用されます。

どれかの Mac OS から Amazon DocumentDB クラスターに直接接続する際に問題が発生した場合はどうすればいいですか。

Mac OS は、信頼できる証明書の要件を更新しています。信頼された証明書の有効期間は 397 日以下である必要があります (https://support.apple.com/en-us/HT211025 を参照)。

> **Note**
>
> この制限は Mac OS の新しいバージョンでも見られます。

Amazon DocumentDB インスタンス証明書の有効期間は 4 年以上であり、Mac OS の上限を超えています。Mac OS を実行しているコンピュータから Amazon DocumentDB クラスターに直接接続するには、TLS 接続の作成時に無効な証明書を許可する必要があります。この場合、無効な証明書と

は、有効期間が 397 日より長いものを意味します。Amazon DocumentDB クラスターに接続する場合、無効な証明書を許可する前に、そのリスクを理解する必要があります。

を使用して Mac OS から Amazon DocumentDB クラスターに接続するには AWS CLI、`tlsAllowInvalidCertificates`パラメータを使用します。

```
mongo --tls --host <hostname> --username <username> --password <password> --port 27017
 --tlsAllowInvalidCertificates
```

Amazon DocumentDB のコンプライアンス検証

Amazon DocumentDB のセキュリティおよびコンプライアンスは、以下を含む複数の AWS コンプライアンスプログラムの一環として、サードパーティーの監査者により評価されます。

- System and Organization Controls (SOC) 1、2、および 3。詳細については、「SOC」を参照してください。
- Federal Risk and Authorization Management Program (FedRAMP)。詳細については、「コンプライアンスプログラムによる対象範囲内のAWS サービス」を参照してください。
- Payment Card Industry Data Security Standard (PCI DSS)。詳細については、「PCI DSS」を参照してください。
- ISO 9001、27001、27017、および 27018。詳細については、「ISO 認定」を参照してください。
- 医療保険の相互運用性と説明責任に関する法律の事業提携契約 (HIPAA BAA)。詳細については、「HIPAA への準拠」を参照してください。

AWS は、コンプライアンスプログラムによる対象範囲内 AWS のサービスで、特定のコンプライアンスAWS プログラムの範囲内のサービスのリストを頻繁に更新します。

サードパーティーの監査レポートは、 を使用してダウンロードできます AWS Artifact。詳細については、AWS 「アーティファクトでのレポートのダウンロード」を参照してください。

AWS コンプライアンスプログラムの詳細については、AWS 「 コンプライアンスプログラム」を参照してください。

Amazon DocumentDB を使用する際のお客様のコンプライアンス責任は、組織のデータの機密性や組織のコンプライアンス目的、適用可能な法律、規制によって決定されます。Amazon DocumentDB の使用が HIPAA や PCI などの標準への準拠の対象である場合、 は以下に役立つリソース AWS を提供します。

- AWS コンプライアンスリソース – お客様の業界や地域に適用される可能性のあるワークブックとガイドのコレクション。
- セキュリティとコンプライアンスのクイックスタートガイド – アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境をデプロイする手順を提供するデプロイガイド AWS。
- AWS Config – リソース設定が社内プラクティス、業界ガイドライン、規制にどの程度準拠しているかを評価するサービス。
- AWS Security Hub – セキュリティ業界標準とベストプラクティスへの準拠を確認するのに役立つ AWS 、 内のセキュリティ状態の包括的なビュー。
- Architecting for HIPAA Security and Compliance ホワイトペーパー – 企業が AWS を使用して HIPAA 準拠のアプリケーションを作成する方法を説明するホワイトペーパー。

Amazon DocumentDB の耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョン およびアベイラビリティーゾーンを中心に構築されています。 は、低レイテンシー、高スループット、高度に冗長なネットワークで接続された、物理的に分離および分離された複数のアベイラビリティーゾーン AWS リージョン を提供します。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断せずに、自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、およびスケーラビリティが優れています。

Amazon DocumentDB クラスターは、最低 2 つのアベイラビリティーゾーンに最低 2 つのサブネットがある Amazon VPC 内にのみ作成できます。少なくとも 2 つのアベイラビリティーゾーンにまたがってクラスターインスタンスを配布することで、万一 1 つのアベイラビリティーゾーンに障害が発生した場合でも、Amazon DocumentDB では、クラスター内でインスタンスを使用できます。Amazon DocumentDB クラスターのクラスターボリュームは、データ損失の可能性が少ない耐久性のあるストレージを提供するために、常に 3 つのアベイラビリティーゾーンにまたがっています。

AWS リージョン およびアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

Amazon DocumentDB には、 AWS グローバルインフラストラクチャに加えて、データの耐障害性とバックアップのニーズをサポートするのに役立つ機能がいくつか用意されています。

耐障害性と自己修復機能を備えたストレージ

ストレージボリュームの 10 GB ごとに、3 つのアベイラビリティーゾーン間で 6 つの方法でレプリケートされます。Amazon DocumentDB は、データベースの書き込み可用性に影響を与えずに最大 2 つのデータの損失を透過的に処理するフォールトトレラントストレージを使用し、読み取り可用性に影響を与えずに最大 3 つのコピーを透過的に処理します。Amazon DocumentDB ストレージも自己修復機能であり、データブロックとディスクは継続的にエラーをスキャンし、自動的に交換されます。

手動バックアップと復元

Amazon DocumentDB は、長期的な保持と回復のためにクラスターの完全バックアップを作成する機能を提供します。詳細については、「Amazon DocumentDB でのバックアップと復元」を参照してください。

ポイントインタイムリカバリ

ポイントインタイムリカバリを使用することで、Amazon DocumentDB クラスターが誤って書き込みされたり削除されたりしないようにできます。ポイントインタイムリカバリを有効化すれば、オンデマンドバックアップの作成、維持、スケジュールを心配する必要はありません。詳細については、「特定時点への復元」を参照してください。

Amazon DocumentDB のインフラストラクチャセキュリティ

マネージドサービスである Amazon DocumentDB は、 AWS グローバルネットワークセキュリティで保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して環境を AWS 設計するには、「Security Pillar AWS Well-Architected Framework」の「Infrastructure Protection」を参照してください。

AWS 公開された API コールを使用して、ネットワーク経由で Amazon DocumentDB にアクセスします。クライアントは次をサポートする必要があります。

- Transport Layer Security (TLS)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードは Java 7 以降など、ほとんどの最新システムでサポートされています。

これらの API オペレーションは任意のネットワークの場所から呼び出すことができます。Amazon DocumentDB ポリシーを使用して、特定の Amazon Virtual Private Cloud (Amazon VPC) エンドポイントまたは特定の VPC からのアクセスを制御することができます。これにより、実質的に、ネットワーク内の特定の VPC からのみ、特定の Amazon DocumentDB リソースへの AWS ネットワークアクセスが分離されます。

> **Note**
>
> Amazon DocumentDB では、リソースベースのポリシーは使用できません。

Amazon DocumentDB API とインターフェイス VPC エンドポイント (AWS PrivateLink) について

> **Note**
>
> Amazon DocumentDB エラスティッククラスターは AWS PrivateLink VPC エンドポイントをサポートしていません。

インターフェイス VPC エンドポイントを作成することで、VPC と Amazon DocumentDB API エンドポイント間にプライベート接続を確立できます。インターフェイスエンドポイントは を利用しています AWS PrivateLink。

Amazon DocumentDB インスタンスベースのクラスターにはインターフェイス VPC エンドポイント接続は必要ありませんが、 AWS PrivateLink を使用すると、インターネットゲートウェイ、NAT デバイス、VPN 接続、または Direct Connect 接続なしで Amazon DocumentDB API オペレーションにプライベートにアクセスできます。VPC 内の Amazon DocumentDB インスタンスは、データベースインスタンス やデータベースクラスターの起動、変更、または終了のための Amazon DocumentDB API エンドポイントとの通信に、パブリック IP アドレスを必要としません。また、Amazon DocumentDB インスタンスは、利用可能な Amazon DocumentDB API オペレーション全ての使用にパブリック IP アドレスを必要としません。VPC と Amazon DocumentDB 間のトラフィックは、すべて Amazon ネットワーク内で行われます。

各インターフェイスエンドポイントは、サブネット内の 1 つ以上の Elastic Network Interface によって表されます。詳細については、「Amazon EC2 ユーザーガイド」の「Elastic Network Interface」を参照してください。

VPC エンドポイントの詳細については、Amazon Virtual Private Cloud (AWS PrivateLink) ユーザーガイド」の「インターフェイス VPC エンドポイント AWS のサービス を使用して にアクセスする」を参照してください。Amazon DocumentDB オペレーションの詳細については、Amazon DocumentDB API リファレンス」を参照してください。

トピック

- VPC エンドポイントに関する考慮事項
- リージョンの可用性
- Amazon DocumentDB API 用のインターフェイス VPC エンドポイントの作成
- Amazon DocumentDB API の VPC エンドポイントポリシーの作成

VPC エンドポイントに関する考慮事項

Amazon DocumentDB API エンドポイント向けにインターフェイス VPC エンドポイントを設定する前に、「Amazon Virtual Private Cloud (AWS PrivateLink) ユーザーガイド」の「インターフェイスエンドポイントの前提条件」を確認してください。

Amazon DocumentDB リソースの管理に関連するすべての Amazon DocumentDB API オペレーションは、 AWS PrivateLinkを使用して VPC から利用することができます。

VPC エンドポイントポリシーは Amazon DocumentDB API エンドポイントでサポートされます。デフォルトでは、エンドポイント経由で Amazon DocumentDB API オペレーションへのフルアクセスが提供されます。詳細については、「Amazon Virtual Private Cloud (AWS PrivateLink) ユーザーガイド」の「エンドポイントポリシーを使用して VPC エンドポイントへのアクセスを制御する」を参照してください。

リージョンの可用性

Amazon DocumentDB API は現在、以下の VPC エンドポイントをサポートしています AWS リージョン。

- 米国東部(オハイオ)
- 米国東部 (バージニア北部)
- 米国西部 (オレゴン)
- アフリカ (ケープタウン)
- アジアパシフィック (香港)

- アジアパシフィック (ムンバイ)
- アジアパシフィック (ハイデラバード)
- アジアパシフィック (大阪)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- カナダ (中部)
- 中国 (北京)
- 中国 (寧夏)
- 欧州 (フランクフルト)
- 欧州 (アイルランド)
- 欧州 (ロンドン)
- 欧州 (パリ)
- 欧州 (スペイン)
- 欧州 (ミラノ)
- 中東 (アラブ首長国連邦)
- 南米 (サンパウロ)
- AWS GovCloud (米国東部)
- AWS GovCloud (米国西部)

Amazon DocumentDB API 用のインターフェイス VPC エンドポイントの作成

Amazon DocumentDB API の VPC エンドポイントを作成できますAWS CLI。 AWS Command Line Interface 詳細については、Amazon Virtual Private Cloud (AWS PrivateLink) ユーザーガイド」の「インターフェイス VPC エンドポイント AWS のサービス を使用して にアクセスする」を参照してください。

サービス名 `com.amazonaws.region.rds` を使用して、Amazon DocumentDB API 向け VPC エンドポイントを作成します。

AWS リージョン 中国を除き、エンドポイントのプライベート DNS を有効にすると、rds.us-east-1.amazonaws.com AWS リージョンなどの のデフォルト DNS 名を使用して、VPC エンドポイントを使用して Amazon DocumentDB に API リクエストを行うことができます。中国 (北京) と中国 (寧夏) では AWS リージョン、それぞれ rds-api.cn-north-1.amazonaws.com.cn と rds-api.cn-northwest-1.amazonaws.com.cn を使用して VPC エンドポイントで API リクエストを行うことができます。

詳細については、Amazon Virtual Private Cloud (AWS PrivateLink) ユーザーガイド」の「インターフェイス VPC エンドポイント AWS のサービス を使用して にアクセスする」を参照してください。

Amazon DocumentDB API の VPC エンドポイントポリシーの作成

Amazon DocumentDB API へのアクセスを制御するエンドポイントポリシーを VPC エンドポイントにアタッチできます。このポリシーでは、以下の情報を指定します。

- アクションを実行できるプリンシパル。
- 実行可能なアクション。
- アクションを実行できるリソース。

詳細については、「Amazon Virtual Private Cloud (AWS PrivateLink) ユーザーガイド」の「エンドポイントポリシーを使用して VPC エンドポイントへのアクセスを制御する」を参照してください。

例: Amazon DocumentDB API アクション向け VPC エンドポイントポリシー

以下は、Amazon DocumentDB API のエンドポイントポリシーの例です。このポリシーをエンドポイントにアタッチすると、すべてのリソースのすべてのプリンシパルに対して、登録されている Amazon DocumentDB API アクションへのアクセスを許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Principal": "*",
      "Effect": "Allow",
      "Action": [
        "rds:CreateDBInstance",
        "rds:ModifyDBInstance",
        "rds:CreateDBSnapshot"
      ],
```

```
      "Resource": "*"
    }
  ]
}
```

例: 指定された からのすべてのアクセスを拒否する VPC エンドポイントポリシー AWS アカウント

次の VPC エンドポイントポリシーは、エンドポイントを使用したリソースへのすべてのアクセスを拒否します AWS アカウント 123456789012。このポリシーは、他の からのすべてのアクションを許可します AWS アカウント。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
    }
  ]
}
```

Amazon DocumentDB のセキュリティベストプラクティス

Amazon DocumentDB は、Amazon VPC の分離、 を使用した保管時の暗号化 AWS KMS、TLS を使用した転送中の暗号化、IAM によるきめ細かなアクセスコントロールなどのセキュリティ機能を提供します。次のベストプラクティスを使用して、Amazon DocumentDB クラスター、インスタンス、データを保護します。

AWS Identity and Access Management (IAM) ロールを使用して、Amazon DocumentDB API オペレーションへのアクセスを制御します。これには、クラスター、セキュリティグループ、パラメータグループなどの Amazon DocumentDB リソースを作成、変更、削除するオペレーションが含まれます。アプリケーションやその他の AWS サービスが Amazon DocumentDB リソースにアクセスする

必要がある場合は、アプリケーションに長期的な認証情報を保存するのではなく、IAM ロールを使用して一時的な認証情報を提供します。IAM ロールを作成するときは、最小特権の原則を適用します。詳細については、IAM ユーザーガイドの「IAM ロール」と「ロールの一般的なシナリオ」を参照してください。

- ロールベースのアクセスコントロールを使用して、最小特権を適用します。
- Amazon DocumentDB リソースの管理に AWS アカウント ルートユーザーを使用しないでください。代わりに、 AWS Identity and Access Management Identity Center を使用して、一時的な認証情報を持つ人間のユーザーを作成します。詳細については、IAM ユーザーガイドの「IAM Identity Center ユーザー」を参照してください。
- 各 ID に、その職務を実行するために必要な最小限のアクセス許可のセットを付与します。
- IAM ポリシーを使用してアクセス許可を管理し、最小特権の原則に従います。詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。
- 長期認証情報で IAM ユーザーを使用する場合は、それらの認証情報を定期的にローテーションします。

> **Warning**
>
> このシナリオでは、プログラムによるアクセスと長期的な認証情報を持つ IAM ユーザーが必要です。これはセキュリティ上のリスクをもたらします。このリスクを軽減するために、これらのユーザーにはタスクの実行に必要な権限のみを付与し、不要になったユーザーは削除することをお勧めします。アクセスキーは、必要に応じて更新できます。詳細については、「IAM ユーザーガイド」の「アクセスキーの更新」を参照してください。

- Amazon DocumentDB のシークレットを自動的にローテーションするように Secrets Manager を設定します。詳細については、「 ユーザーガイド」の「Amazon DocumentDB のシークレットのローテーション」および「Amazon DocumentDB のシークレットのローテーション Amazon DocumentDB」を参照してください。 AWS Secrets Manager
- Transport Layer Security (TLS) と保管時の暗号化を使用してデータを暗号化します。

Amazon DocumentDB イベントの監査

Amazon DocumentDB (MongoDB の互換性あり) を使用すると、クラスター内で実行されたイベントを監査できます。ログに記録されるイベントの例としては、認証の成功と失敗、データベース内のコレクションの削除、インデックスの作成などがあります。Amazon DocumentDB ではデフォルトで監査が無効になっているため、この機能を選択して使用する必要があります。

監査が有効な場合、Amazon DocumentDB はデータ定義言語 (DDL)、データ操作言語 (DML)、認証、認可、およびユーザー管理イベントを Amazon CloudWatch Logs に記録します。監査が有効な場合、Amazon DocumentDB はクラスターの監査レコード (JSON ドキュメント) を Amazon CloudWatch Logs にエクスポートします。Amazon CloudWatch Logs を使用すると、Amazon DocumentDB 監査イベントを分析、モニタリング、およびアーカイブすることができます。

Amazon DocumentDB では監査を有効にしても追加料金は発生しませんが、CloudWatch Logs の使用には標準料金がかかります。CloudWatch Logs の料金の詳細については、Amazon CloudWatch の料金 をご覧ください。

Amazon DocumentDB 監査機能は、モニタリング対象のサービスリソースの使用状況とは明確に異なります AWS CloudTrail。CloudTrail は、 AWS Command Line Interface (AWS CLI) またはクラスター、インスタンス、パラメータグループ、スナップショットなどのリソース AWS マネジメントコンソール で実行されるオペレーションを記録します。CloudTrailを使用した リソースの監査はデフォルトで有効化され、無効にすることはできません。Amazon DocumentDB 監査機能は、オプトイン機能です。これによって、オブジェクト (データベース、コレクション、インデックスやユーザーなど) のクラスター内で発生したオペレーションが記録されます。

トピック

- サポートされるイベント
- 監査の有効化
- 監査の無効化
- 監査イベントへのアクセス
- DML 監査イベントのフィルタリング

サポートされるイベント

Amazon DocumentDB 監査は、以下のカテゴリのイベントをサポートしています。

- データ定義言語 (DDL)- データベースの管理操作、接続、ユーザー管理、認可が含まれます。
- データ操作言語読み取りイベント (DML 読み取り) - `find()`、 およびさまざまな集計演算子、算術演算子、ブール演算子、およびその他の読み取りクエリ演算子が対象になります。
- データ操作言語書き込みイベント (DML 書き込み) - `insert()`, `update()`, `delete()`, および `bulkWrite()` 演算子が対象になります。

イベントタイプは次のとおりです。

イベントタイプ	Category	説明
authCheck	Authorization	結果コード 0: 成功
		結果コード 13: 不正な操作によるオペレーションの実行。
authenticate	接続	新規接続に対する認証 (成功または失敗)。
auditConfigure	DDL	フィルター設定の監査。
createDatabase	DDL	新しいデータベースの作成。
createCollection	DDL	データベース内での新しいコレクションの作成。
createIndex	DDL	コレクション内での新しいインデックスの作成。
dropCollection	DDL	データベース内のコレクションの削除。
dropDatabase	DDL	データベースの削除。
dropIndex	DDL	コレクション内のインデックスの削除。
modifyChangeStreams	DDL	変更ストリームが作成されました。

イベントタイプ	Category	説明
renameCollection	DDL	データベース内のコレクションの名前変更。
createRole	ロールの管理	ロールの作成。
dropAllRolesFromDatabase	ロールの管理	データベース内のすべてのロールの削除。
dropRole	ロールの管理	ロールの削除。
grantPrivilegesToRole	ロールの管理	ロールへの権限の付与。
grantRolesToRole	ロールの管理	ユーザー定義ロールへのロールの付与。
revokePrivilegesFromRole	ロールの管理	ロールからの権限の取り消し。
revokeRolesFromRole	ロールの管理	ユーザー定義ロールからロールの取り消し。
updateRole	ロールの管理	ロールの更新。
createUser	ユーザー管理	新しいユーザーの作成。
dropAllUsersFromDatabase	ユーザー管理	データベース内のすべてのユーザーの削除。
dropUser	ユーザー管理	既存のユーザーの削除。

イベントタイプ	Category	説明
grantRolesToUser	ユーザー管理	ユーザーにロールを付与する。
revokeRolesFromUser	ユーザー管理	ユーザーからロールを取り消します。
updateUser	ユーザー管理	既存のユーザーの更新。
insert	DML 書き込み	1 つまたは複数のドキュメントをコレクションに挿入します。
delete	DML 書き込み	コレクションから 1 つまたは複数のドキュメントを削除します。
update	DML 書き込み	既存の 1 つまたは複数のコレクション内のドキュメントを変更します。
bulkWrite	DML 書き込み	実行順序を制御を使って複数の書き込み操作を実行します。
setAuditConfig	DML 書き込み	DML 監査向け新しいフィルターを設定します。
count	DML 読み取り	コレクションまたはビューの find() クエリに一致するドキュメントの数を返します。

イベントタイプ	Category	説明
countDocuments	DML 読み取り	コレクションまたはビューのクエリに一致するドキュメントの数を返します。
find	DML 読み取り	コレクションまたはビュー内のドキュメントを選択し、選択したドキュメントにカーソルを返します。
getAuditConfig	DML 読み取り	DML 監査向けの現在のフィルターを取得します。
findAndModify	DML 読み取りと DML 書き込み	単一のドキュメントを変更して返します。
findOneAndDelete	DML 読み取りと DML 書き込み	フィルターと並べ替え条件に基づいて 1 つのドキュメントを削除し、削除したドキュメントを返します。
findOneAndReplace	DML 読み取りと DML 書き込み	指定したフィルターに基づいて 1 つのドキュメントを置き換えます。
findOneAndUpdate	DML 読み取りと DML 書き込み	フィルターと並べ替え条件に基づいて 1 つのドキュメントを更新します。

イベントタイプ	Category	説明
aggregate	DML 読み取りと DML 書き込み	集約パイプラインの API をサポートします。
distinct	DML 読み取り	1 つのコレクションまたはビューから指定されたフィールドの個別値を検索し、その結果を配列で返します。

> **Note**
>
> DML イベントドキュメントパラメータフィールドの値には 1 KB のサイズ制限があります。サイズが 1 KB を超えると、Amazon DocumentDB 値は切り捨てられます。

> **Note**
>
> 現時点では、TTL 削除イベントは監査されません。

監査の有効化

クラスターでの監査の有効化は、2 ステップのプロセスです。両方のステップが完了していることを確認してください。完了していない場合、監査ログは CloudWatch Logs に送信されません。

ステップ 1. audit_logs クラスターパラメータを有効にする

監査を有効にするには、パラメータグループの `audit_logs` パラメータを変更する必要があります。`audit_logs` は記録するイベントのカンマ区切りのリストです。イベントはすべて小文字で指定する必要があります。リスト要素間に空白があってはいけません。

パラメータには以下の値を指定できます。

値	説明	
ddl	設定すると、createDatabase、dropDatabase、createCollection、dropCollection、createIndex、dropIndex、authCheck、authenticate、createUser、dropUser、grantRolesToUser、revokeRolesFromUser、updateUser、dropAllUsersFromDatabase などの DDL イベントの監査が有効になります	
dml_read	設定すると、find、sort count、distinct、group、projecta、unwind、geoNear、geoIntersects、geoWithin、その他の MongoDB 読み取りクエリ演算子などの DML 読み取りイベントの監査が有効になります。	
dml_write	設定すると、insert()、update()、delete()、bulkWrite() などの DML 書き込みイ	

値	説明	
	ベントの監査が有効になります。	
all	設定すると、読み取りクエリ、書き込みクエリ、データベースアクション、管理者アクションなどのデータベースイベントの監査が有効になります。	
none	設定すると、監査が無効になります	

値	説明	
enabled (レガシー)	これは「ddl」と同等のレガシーパラメータ設定です。設定すると、createDatabase、dropDatabase、createCollection、dropCollection、createIndex、dropIndex、authCheck、authenticate、createUser、dropUser、grantRolesToUser、revokeRolesFromUser、updateUser、dropAllUsersFromDatabase などの DDL イベントの監査が有効になります。レガシー設定であるため、この設定の使用はお勧めしません。	
disabled (レガシー)	これは「なし」と同等のレガシーパラメータ設定です。レガシー設定であるため、この設定の使用はお勧めしません。	

> **Note**
>
> audit_logs クラスターパラメーターのデフォルト値は none (レガシー「disabled」) です。

上記の値を組み合わせて使用することもできます。

値	説明
ddl, dml_read	設定すると、DDL イベントと DML 読み取りイベントの監査が有効になります。
ddl, dml_write	設定すると、DDL イベントと DML 書き込みイベントの監査が有効になります。
dml_read, dml_write	設定すると、すべての DML イベントの監査が有効になります。

> **Note**
>
> デフォルトのパラメータグループを変更することはできません。

詳細については次を参照してください:

- Amazon DocumentDB クラスターパラメータグループを作成する

  カスタムパラメータグループを作成したら、audit_logs パラメータ値を all に変更します。
- Amazon DocumentDB クラスターパラメータグループを変更する

ステップ 2. Amazon CloudWatch Logs のエクスポートを有効にする

`audit_logs` クラスターパラメータの値が `enabled` か `ddl`、`dml_read`、`dml_write` のいずれかの場合、Amazon DocumentDB がログを Amazon CloudWatch にエクスポートできるようにする必要があります。上述の手順のいずれかを省略すると、監査ログは CloudWatch に送信されません。

クラスターを作成するときにポイントインタイムリストアまたはスナップショットの復元を実行する場合、以下のステップで CloudWatch Logs を有効にすることができます。

Using the AWS マネジメントコンソール

コンソールを使用して Amazon DocumentDB へのログを CloudWatch にエクスポートすることを有効にするには、次のトピックを参照してください。

- クラスターを作成する場合 - を使用したクラスターとプライマリインスタンスの作成 AWS マネジメントコンソール の「クラスターを作成する: 追加の設定 (ステップ 5、[ログのエクスポート])」を参照してください。
- 既存のクラスターを変更する場合 - Amazon DocumentDB クラスターの変更
- クラススナップショットリストアを実行する場合 - クラススナップショットからの復元
- ポイントインタイムリストアを実行する場合 - 特定時点への復元

Using the AWS CLI

新しいクラスターの作成時に監査ログを有効にするには

次のコードでは、クラスター `sample-cluster` を作成し、CloudWatch 監査ログを有効にします。

Example

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

既存のクラスターを変更するときに監査ログを有効にするには

次のコードでは、クラスター sample-cluster を変更し、CloudWatch 監査ログを有効にします。

Example

Linux、macOS、Unix の場合:

```
aws docdb modify-db-cluster \
   --db-cluster-identifier sample-cluster \
   --cloudwatch-logs-export-configuration '{"EnableLogTypes":["audit"]}'
```

Windows の場合:

```
aws docdb modify-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --cloudwatch-logs-export-configuration '{"EnableLogTypes":["audit"]}'
```

これらのオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "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"
    }
}
```

監査の無効化

CloudWatch Logs のエクスポートを無効にし、audit_logs パラメータを無効にすることで、監査を無効にすることができます。

CloudWatch Logs のエクスポートの無効化

監査ログのエクスポートは、 AWS マネジメントコンソール または を使用して無効にできます AWS CLI。

Using the AWS マネジメントコンソール

次の手順では AWS マネジメントコンソール 、 を使用して Amazon DocumentDB の CloudWatch へのログのエクスポートを無効にします。

監査ログを無効にするには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。ログのエクスポートを無効にするクラスターの名前の左側にあるボタンを選択します。
3. [アクション] を選択してから、[変更] を選択します。
4. [ログのエクスポート] セクションまで下にスクロールし、[無効] を選択します。
5. [続行] をクリックしてください。
6. 変更を確認してから、この変更をクラスターにいつ適用するかを選択します。
   - Apply during the next scheduled maintenance window (次に予定されているメンテナンス期間中に適用)
   - すぐに適用
7. [クラスターを変更] を選択します。

Using the AWS CLI

次のコードでは、クラスター `sample-cluster` を変更し、CloudWatch 監査ログを無効にします。

Example

Linux、macOS、Unix の場合:

```
aws docdb modify-db-cluster \
   --db-cluster-identifier sample-cluster \
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["audit"]}'
```

Windows の場合:

```
aws docdb modify-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["audit"]}'
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

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

audit_logs パラメータを無効にする

クラスターの `audit_logs` パラメータを無効にするには、`audit_logs` パラメータ値が `disabled` であるパラメータグループを使用するようにクラスターを変更します。または、クラスターの `audit_logs` パラメータグループのパラメータ値を `disabled` に変更することもできます。

詳細については、以下の各トピックを参照してください。

- Amazon DocumentDB クラスターの変更
- Amazon DocumentDB クラスターパラメータグループを変更する

監査イベントへのアクセス

Amazon CloudWatch の監査イベントにアクセスするには、以下のステップを実行してください。

1. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
2. Amazon DocumentDB クラスターと同じリージョンであることを確認します。
3. ナビゲーションペインで [ログ] を選択します。
4. クラスターの監査ログを確認するため、リストから **/aws/docdb/*yourClusterName*/audit** を見つけ出して選択します。

   各インスタンスの監査イベントは、各インスタンス名の下に表示されます。

DML 監査イベントのフィルタリング

DML 監査フィルタリングの開始方法

DML 監査イベントは、Amazon CloudWatch への書き込み前にフィルタリングできます。この機能を使用するには、監査ログと DML ログ記録を有効にする必要があります。Amazon DocumentDB は、`atype`、`command`、`user`、`namespace`、`auditAuthorizationSuccess` でのフィルタリングをサポートしています。

> **Note**
>
> DDL イベントはフィルタリングされません。

db.adminCommand({ command }) オペレーションの setAuditConfig、filter、auditAuthorizationSuccess パラメータを使用して監査フィルターを指定することで、いつでも監査フィルタリングを有効にできます。

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter:
         {
            //filter conditions
         },
      auditAuthorizationSuccess: true | false
   }
)
```

次のコマンドを実行して、監査フィルター設定を取得することもできます。

```
db.admin.runCommand( { getAuditConfig: 1})
```

セキュリティ要件

DML 監査フィルターを設定または一覧表示する際に admindb に対して上記のコマンドを実行できるのは、特権アクション auditConfigure を持つデータベースユーザー/ロールのみです。[clusterAdmin、hostManager、root] の組み込みロールのいずれかを使用するか、あるいは auditConfigure 権限を持つカスタムロールを作成できます。以下は、auditConfigure 権限で既存のロールを使用する例と、カスタムロールを使用する例です。

ロールが組み込まれているユーザー:

```
use admin
db.createUser(
  {
    user: "myClusterAdmin",
    pwd: "password123",
    roles: [ { role: "clusterAdmin", db: "admin" } ]
  }
```

```
)
```

カスタムロールを持つユーザー:

```
use admin
db.createRole(
   {
     role: "myRole",
     privileges: [
       { resource: { cluster: true }, actions: [ "auditConfigure" ] }
     ],
     roles: []
   }
)
db.createUser(
  {
    user: "myUser",
    pwd: "myPassword",
    roles: [ { role: "myRole", db: "admin" } ]
  }
)
```

ユースケースのフィルタリング

例: コマンドによるイベントのフィルタリング

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
        "$and": [
         {
            "param.command":
               {
                  $in: [ "find","count", "insert", "delete", "update",
 "findandmodify" ]
               }
         }
         ]
      },
      auditAuthorizationSuccess: true
   }
)
```

例: ユーザー名によるイベントのフィルタリング

この例では、ユーザー「myUser」のみがログに記録されます。

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
      "$and": [
         {
            "param.user":
               {
                  $in: [ "myUser" ]
               }
         }
         ]},
      auditAuthorizationSuccess: true})
```

例: **atype** によるフィルタリング

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {atype: "authCheck"},
      auditAuthorizationSuccess: true
   })
```

> **Note**
>
> すべての DML ログは、atype として authCheck を持っています。DDL のみが別の atype を持っています。filter に authCheck 以外の値を指定した場合、CloudWatch での DML ログの生成は行われません。

例: 演算子によって結合された複数のフィルターを使用したフィルタリング

```
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**
>
> 最上位レベルでは、$and、$or、$nor のみがサポートされています。他の演算子はサポートされていないため、エラーが発生します。

例: **auditAuthorizationSuccess** によるイベント別のフィルタリング

このフィルターでは、認証に合格したすべてのコマンドはログに記録されません。

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {},
      auditAuthorizationSuccess: false
   }
)
```

例: **$in** および **$nin** 条件によるフィルタリング

$in と $nin の両方で使用する場合、条件間に暗黙的な「and」があるため、コマンドはログに記録されません。この例では、正規表現で find コマンドがブロックされるため、ログには何も記録されません。

```
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})
```

例: **namespace** によるフィルタリング

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
      "$and": [
         {
            "param.ns":
               {
                  $in: [ "test.foo" ]
               }
         }
         ]},
      auditAuthorizationSuccess: true})
```

例: デフォルトフィルターへのリセット

デフォルト値にリセットすると、すべての DML 監査イベントがログに記録されます。フィルターをデフォルト値にリセットするには、次のコマンドを実行します。

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {},
      auditAuthorizationSuccess: true
   }
)
```

Amazon VPC と Amazon DocumentDB

Amazon Virtual Private Cloud (Amazon VPC) を使用すると、Amazon DocumentDB インスタンスなどの AWS リソースを Virtual Private Cloud (VPC) に起動できます。

VPC を使用する場合、仮想ネットワーキング環境を制御できます。独自の IP アドレスの範囲を選択し、サブネットを作成してルーティングおよびアクセス制御リストを設定できます。VPC でクラスターを実行するために、追加料金はかかりません。

アカウントにはデフォルト VPC があります。新しいすべてのクラスターは、特に指定がない限り、デフォルトの VPC 内に作成されます。

トピック

- VPC 内の DocumentDB クラスター
- VPC での Amazon DocumentDB クラスターへのアクセス
- DocumentDB クラスターで使用する IPv4-only VPC を作成する
- DocumentDB クラスターで使用するデュアルスタック VPC を作成する

以下に、Amazon DocumentDB クラスターに関連する VPC の機能について説明します。Amazon VPC の詳細については、「Amazon VPC ユーザーガイド」を参照してください。

VPC 内の DocumentDB クラスター

仮想プライベートクラウド (VPC) 内の Amazon DocumentDB クラスター。VPC は、 内の他の仮想ネットワークから論理的に分離された仮想ネットワークです AWS クラウド。Amazon VPC を使用すると、Amazon DocumentDB クラスターや Amazon EC2 インスタンスなどの AWS リソースを

VPC 内に起動できます。VPC は、自分のアカウントに属するデフォルト VPC を使用するか、独自に作成することもできます。すべての VPCsは AWS アカウントに関連付けられています。

デフォルト VPC には、VPC 内でリソースを隔離するために使用できる 3 つのサブネットがあります。デフォルト VPC には、VPC 外から VPC 内のリソースへのアクセスを可能にするインターネットゲートウェイもあります。

VPC 内および VPC 外の Amazon DocumentDB クラスターが関与するシナリオについては、「VPC での Amazon DocumentDB クラスターへのアクセス」を参照してください。

トピック

- VPC でのクラスターの操作
- サブネットグループの使用
- Amazon DocumentDB IP アドレス指定
- VPC でクラスターを作成する

VPC でのクラスターの操作

次に、VPC でのクラスターの操作に関するヒントを紹介します。

- VPC には少なくとも 2 つのサブネットが必要です。これらのサブネットは、クラスターをデプロイ AWS リージョン する の 2 つの異なるアベイラビリティーゾーンに存在する必要があります。サブネットは、VPC の IP アドレス範囲の指定可能なセグメントで、セキュリティや運用上のニーズに基づいてクラスターをグループ化することができます。
- VPC のクラスターをパブリックにアクセス可能にする場合は、必ず VPC 属性の `DNS hostnames` と `DNS resolution` を有効にしてください。
- ご利用の VPC では、サブネットグループを作成する必要があります。サブネットグループを作成するには、作成したサブネットを指定します。Amazon DocumentDB は、サブネットグループとそのサブネット内の IP アドレスを選択し、クラスターのプライマリインスタンスに関連付けます。プライマリインスタンスは、そのサブネットを含むアベイラビリティーゾーンを使用します。
- VPC には、クラスターへのアクセスを許可する VPC セキュリティグループが必要です。

  詳細については、「VPC での Amazon DocumentDB クラスターへのアクセス」を参照してください。
- 各サブネットの CIDR ブロックは、フェイルオーバーやコンピュートスケーリングの見積りなどのメンテナンス作業中に Amazon DocumentDB が使用する予備の IP アドレスに十分対応できる大きさが必要です。例えば、通常は 10.0.0.0/24 や 10.0.1.0/24 などの範囲で十分な大きさです。

- VPC は、default または dedicated の instance tenancy 属性を持つことができます。デフォルト VPC では、instance tenancy 属性はすべて default に設定され、インスタンスのすべてのクラスがサポートされます。

  instance tenancy 属性を dedicated に設定した専有 VPC にクラスターを保持する場合は、そのクラスターのインスタンスクラスは、承認された Amazon EC2 ハードウェア専有インスタンスタイプのいずれかである必要があります。例えば、r5.large EC2 ハードウェア専有インスタンスは、db.r5.large インスタンスクラスに対応します。VPC のインスタンステナンシーについては、「Amazon Elastic Compute Cloud ユーザーガイド」の「Amazon EC2 インスタンス」を参照してください。

  ハードウェア専有インスタンスに対応するインスタンスタイプの詳細については、Amazon EC2 の料金ページで「Amazon EC2 のハードウェア専有インスタンス」を参照してください。

  > **Note**
  >
  > instance tenancy 属性をクラスターの dedicated に設定しても、クラスターが専有ホストで実行されることは保証されません。

サブネットグループの使用

サブネットは、VPC の IP アドレス範囲のセグメントで、セキュリティや運用上のニーズに基づいてリソースをグループ化するために指定します。サブネットグループは VPC に作成するサブネット (通常はプライベート) のコレクションで、クラスター用に指定します。サブネットグループを使用すると、 AWS CLI または Amazon DocumentDB API を使用してクラスターを作成するときに、特定の VPC を指定できます。コンソールを使用する場合は、使用する VPC とサブネットグループを選択できます。

各サブネットグループには、特定の AWS リージョン内の少なくとも 2 つのアベイラビリティーゾーンにサブネットが必要です。VPC にクラスターを作成するときに、サブネットグループを選択する必要があります。サブネットグループから、Amazon DocumentDB はサブネットとそのサブネット内の IP アドレスを選択して、クラスターのプライマリインスタンスに関連付けます。データベースは、そのサブネットを含むアベイラビリティーゾーンを使用します。DocumentDB は常に、空き IP アドレス空間を持つサブネットから IP アドレスを割り当てます。

サブネットグループ内のサブネットは、ネットワークアクセスコントロールリスト (ネットワーク ACL) とルーティングテーブルに定義した設定に応じて、パブリックまたはプライベートになります。クラスターをパブリックにアクセス可能にするには、そのサブネットグループ内のすべてのサブ

ネットがパブリックである必要があります。パブリックにアクセスできるクラスターに関連付けられているサブネットがパブリックからプライベートに変更された場合、クラスターの可用性に影響する可能性があります。

デュアルスタックモードをサポートするサブネットグループを作成するには、サブネットグループに追加する各サブネットに Internet Protocol version 6 (IPv6) CIDR ブロックが関連付けられていることを確認してください。詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「Amazon DocumentDB IP アドレス指定」および「VPC の IPv6 サポート」を参照してください。

Amazon DocumentDB が VPC にクラスターを作成すると、サブネットグループから IP アドレスを使用して、クラスターにネットワークインターフェースが割り当てられます。基盤となる IP アドレスはフェイルオーバー中に変更されるため、クラスターへの接続には必ずドメインネームシステム (DNS) 名を使用してください。

> **Note**
>
> VPC で実行するクラスターごとに、Amazon DocumentDB による復旧アクション用として、サブネットグループのサブネットごとに最低 1 つのアドレスを確保してください。

共有サブネット

クラスターは共有 VPC に作成できます。

共有 VPC を使用する際に留意すべき点がいくつかあります。

- クラスターを共有 VPC サブネットから非共有 VPC サブネットに、またはその逆に移動できます。
- 共有 VPC の参加者は、クラスターを作成できるように、VPC にセキュリティグループを作成する必要があります。
- 共有 VPC の所有者と参加者は、DocumentDB クエリを使用してデータベースにアクセスできます。ただし、リソースに対して任意の API 呼び出しを行うことができるのは、リソースの作成者だけです。

Amazon DocumentDB IP アドレス指定

IP アドレスは、VPC 内のリソースの相互通信と、インターネット経由でのリソースとの通信を可能にします。Amazon DocumentDB は、IPv4 と IPv6 の両方のアドレス指定プロトコルをサポートしています。デフォルトでは、Amazon DocumentDB と Amazon VPC は IPv4 アドレス設定プロトコル

を使用します。この動作をオフにすることはできません。VPC の作成時には、IPv4 CIDR ブロック (プライベート IPv4 アドレスの範囲) を指定する必要があります。必要に応じて、IPv6 CIDR ブロックを VPC とサブネットに割り当て、そのブロックからサブネットのクラスターに IPv6 アドレスを割り当てることができます。

> **Note**
>
> デュアルスタックモード (IPv6 アドレス指定) は、Amazon DocumentDB バージョン 4.0 および 5.0 でのみサポートされています。

IPv6 プロトコルのサポートにより、サポートされる IP アドレスの数が増えます。IPv6 プロトコルを使用することで、インターネットの今後の成長に十分なアドレスを確保できます。新規および既存の DocumentDB リソースは、VPC 内で IPv4 アドレスと IPv6 アドレスを使用できます。アプリケーションの異なる部分で使用される 2 つのプロトコル間のネットワークトラフィックの設定、保護、および変換を行うと、運用上のオーバーヘッドが発生する可能性があります。Amazon DocumentDB リソースについては IPv6 プロトコルを標準化して、ネットワーク構成を簡素化できます。

トピック

- IPv4 アドレス
- IPv6 アドレス
- デュアルスタックモード

IPv4 アドレス

VPC を作成するときには、その VPC の IPv4 アドレスの範囲を CIDR ブロックの形式で指定する必要があります (`10.0.0.0/16` など)。サブネットグループは、クラスターが使用できる、この CIDR ブロック内の IP アドレスの範囲を定義します。これらの IP アドレスはプライベートまたはパブリックです。

プライベート IPv4 アドレスは、インターネットから到達できない IP アドレスです。クラスターと同じ VPC 内の他のリソース (Amazon EC2 インスタンスなど) との通信には、プライベート IPv4 アドレスを使用できます。各クラスターには、VPC 内の通信用のプライベート IP アドレスがあります。

パブリック IP アドレスは、インターネットから到達可能な IPv4 アドレスです。DocumentDB クラスターでは、パブリック IP アドレス指定は許可されていません。パブリック IP アドレスは、インターネットゲートウェイとパブリックサブネットの EC2 によって解決する必要があります。

一般的な Amazon DocumentDB シナリオに使用できるプライベート IPv4 アドレスのみで VPC を作成する方法については、「DocumentDB クラスターで使用する IPv4-only VPC を作成する」を参照してください。

IPv6 アドレス

オプションで IPv6 CIDR ブロックを VPC およびサブネットと関連付けて、そのブロックから VPC 内のリソースに IPv6 アドレスを割り当てることができます。各 IPv6 アドレスは、グローバルに一意です。

VPC の IPv6 CIDR ブロックは、Amazon の IPv6 アドレスのプールから自動的に割り当てられます。範囲を自分で選択することはできません。

IPv6 アドレスに接続するときには、以下の条件が満たされていることを確認してください。

- クライアントは、クライアントから IPv6 経由でのデータベーストラフィックが許可されるように構成されています。
- クラスターによって使用される DocumentDB セキュリティグループは、クライアントからデータベースへの IPv6 経由のトラフィックが許可されるように、正しく構成されています。
- クライアントのオペレーティングシステムスタックは IPv6 アドレス上のトラフィックを許可し、オペレーティングシステムドライバーとライブラリは、正しいデフォルトのクラスターエンドポイント (IPv4 または IPv6) を選択するように構成されています。

IPv6 の詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「IP アドレス指定」を参照してください。

デュアルスタックモード

クラスターが IPv4 と IPv6 の両方のアドレス指定プロトコルで通信できるときには、デュアルスタックモードで実行しています。したがって、リソースはクラスターと IPv4、IPv6、またはその両方で通信できます。DocumentDB は、プライベートデュアルスタックモードクラスターの IPv6 エンドポイントについてインターネットゲートウェイアクセスを無効にします。DocumentDB がこれを行うのは、IPv6 エンドポイントをプライベートにして、VPC 内からのみアクセスできるようにするためです。

トピック

- デュアルスタックモードとサブネットグループ
- デュアルスタックモードクラスターの使用

- IPv4 専用クラスターをデュアルスタックモードを使用するように変更する
- デュアルスタックモードのリージョンとバージョンの可用性
- デュアルスタックネットワーククラスターの制限

デュアルスタックモードとサブネットグループ

デュアルスタックモードを使用するには、クラスターに関連付けるサブネットグループ内の各サブネットに IPv6 CIDR ブロックが関連付けられていることを確認してください。新しいサブネットグループを作成するか、既存のサブネットグループを変更して、この要件を満たすことができます。クラスターがデュアルスタックモードになった後も、クライアントは通常どおり接続できます。クライアントセキュリティファイアウォールと DocumentDB クラスターのセキュリティグループが、IPv6 経由のトラフィックを許可するように正しく設定されていることを確認します。接続するために、クライアントはクラスターのエンドポイントを使用します。クライアントアプリケーションは、データベースへの接続時に優先するプロトコルを指定できます。デュアルスタックモードでは、クラスターは、クライアントの優先ネットワークプロトコル (IPv4 または IPv6) を検出し、そのプロトコルを使用して接続します。

サブネットの削除または CIDR の関連付け解除により、サブネットグループがデュアルスタックモードをサポートしなくなった場合、サブネットグループに関連付けられているクラスターに対して互換性のないネットワーク状態が発生するリスクがあります。また、新しいデュアルスタックモードのクラスターの作成時にサブネットグループを使用することはできません。

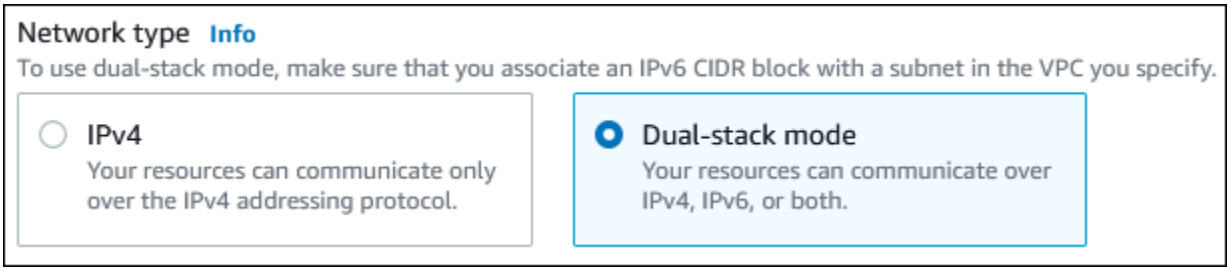
を使用してサブネットグループがデュアルスタックモードをサポートしているかどうかを確認するには AWS マネジメントコンソール、サブネットグループの詳細ページでネットワークタイプを表示します。を使用してサブネットグループがデュアルスタックモードをサポートしているかどうかを確認するには AWS CLI、 `describe-db-subnet-groups` コマンドを実行して出力`SupportedNetworkTypes`で を表示します。

リードレプリカは独立したクラスターとして扱われ、プライマリクラスターとは異なるネットワークタイプを持つことができます。リードレプリカのプライマリクラスターのネットワークタイプを変更しても、リードレプリカは影響を受けません。クラスターを復元するときには、サポートされている任意のネットワークタイプに復元できます。

デュアルスタックモードクラスターの使用

クラスターを作成または変更する場合は、デュアルスタックモードを指定して、リソースがクラスターと IPv4、IPv6、またはその両方で通信することを許可できます。

を使用してクラスター AWS マネジメントコンソール を作成または変更する場合、ネットワークタイプセクションでデュアルスタックモードを指定できます。次の画像は、コンソールの [Network type] (ネットワークの種類) セクションを示しています。



を使用してクラスター AWS CLI を作成または変更する場合は、デュアルスタックモードを使用するDUALように `--network-type`オプションを に設定します。DocumentDB API を使用してクラスターを作成または変更するときには、`NetworkType` パラメータを `DUAL` に設定して、デュアルスタックモードを使用します。指定された DocumentDB エンジンバージョンまたはサブネットグループでデュアルスタックモードがサポートされていない場合は、`NetworkTypeNotSupported` エラーが返されます。

クラスターの作成の詳細については、「Amazon DocumentDB クラスターの作成」を参照してください。クラスターの変更の詳細については、「Amazon DocumentDB クラスターの変更」を参照してください。

コンソールを使用してインスタンスクラスターがデュアルスタックモードであるかどうかを判断するには、クラスターの [Connectivity & security] (接続性とセキュリティ) タブの [Network type] (ネットワークの種類) を確認します。

IPv4 専用クラスターをデュアルスタックモードを使用するように変更する

IPv4 専用クラスターをデュアルスタックモードを使用するように変更できます。このためには、クラスターのネットワークの種類を変更します。

メンテナンスウィンドウ中に Amazon DocumentDB クラスターのネットワークタイプを変更することをお勧めします。ネットワークタイプは、`modify-db-cluster` コマンドを使用して手動で設定できます。

クラスターをデュアルスタックモードを使用するように変更する前に、そのサブネットグループがデュアルスタックモードをサポートしていることを確認してください。クラスターに関連付けられたサブネットグループがデュアルスタックモードをサポートしていない場合は、クラスターを変更するときに、それをサポートする別のサブネットグループを指定します。クラスターのサブネットグループを変更すると、ダウンタイムが発生する可能性があります。

クラスターをデュアルスタックモードを使用するように変更する前にクラスターのサブネットグループを変更する場合は、変更の前後にサブネットグループがクラスターに対して有効であることを確認してください。

NetworkType パラメータのみを に設定してModifyDBCluster呼び出しを実行しDUAL、ネットワークをデュアルスタックモードに変更します。同じ API コールで NetworkType と一緒に他のパラメータを追加すると、ダウンタイムが発生する可能性があります。複数のパラメータを変更する場合は、他のパラメータを使用して別の ModifyDBCluster リクエストを送信する前に、ネットワークタイプの変更が正常に完了していることを確認してください。

変更後にクラスターに接続できない場合は、選択したネットワーク (IPv4 または IPv6) でデータベースへのトラフィックを許可するように、クライアントとデータベースのセキュリティファイアウォールとルートテーブルが正確に設定されていることを確認してください。IPv6 アドレスを使用して接続するには、オペレーティングシステムのパラメータ、ライブラリ、またはドライバーを変更しなければならない場合もあります。

IPv4 専用のクラスターをデュアルスタックモードを使用するように変更するには

1. サブネットグループをデュアルスタックモードをサポートするように変更するか、デュアルスタックモードをサポートするサブネットグループを作成します。

   a. IPv6 CIDR ブロックと VPC の関連付け

      詳細については、「Amazon VPC ユーザーガイド」の「CIDR ブロックを VPC に追加する」を参照してください。

   b. IPv6 CIDR ブロックをサブネットグループ内のすべてのサブネットにアタッチします。

      詳細については、「Amazon VPC ユーザーガイド」の「IPv6 CIDR ブロックをサブネットに追加する」を参照してください。

   c. サブネットグループがデュアルスタックモードをサポートしていることを確認します。

      を使用している場合は AWS マネジメントコンソール、サブネットグループを選択し、サポートされているネットワークタイプの値がデュアルであることを確認します。

      を使用している場合は AWS CLI、 `describe-db-subnet-groups` コマンドを実行し、クラスターのSupportedNetworkType値が であることを確認しますDual。

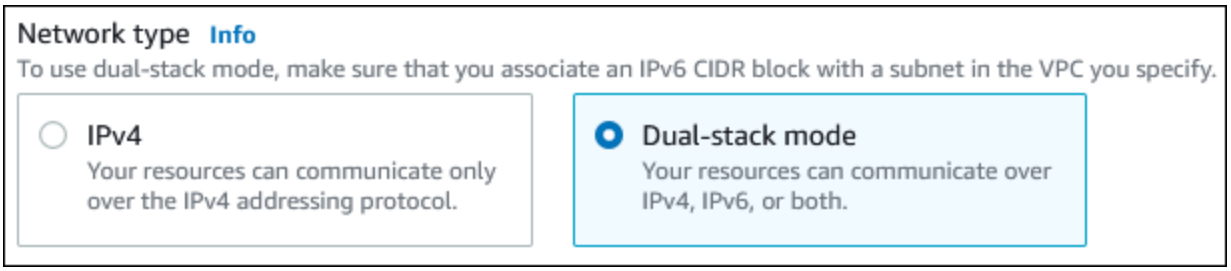
2. クラスターに関連付けられたセキュリティグループを、データベースへの IPv6 接続を許可するように変更するか、IPv6 接続を許可する新しいセキュリティグループを作成します。

手順については、「Amazon Virtual Private Cloud ユーザーガイド」の「セキュリティグループのルール」を参照してください。

3. クラスターを変更して、デュアルスタックモードをサポートします。そのためには、ネットワークの種類 を デュアルスタックモード に設定します。

   コンソールを使用している場合、以下の設定が正しいことを確認します。

   - [Network type] (ネットワークタイプ) — [Dual-stack mode] (デュアルスタックモード)

   - [Subnet group] (サブネットグループ) — 前のステップで設定したサブネットグループ
   - [Security group] (セキュリティグループ) — 前のステップで設定したセキュリティ

   を使用している場合は AWS CLI、次の設定が正しいことを確認してください。

   - `--network-type` — `dual`
   - `--db-subnet-group-name` — 前のステップで設定したサブネットグループ
   - `--vpc-security-group-ids` — 前のステップで設定した VPC セキュリティグループ

   例えば、次のようになります。

   ```
   aws docdb modify-db-cluster --db-cluster-identifier <cluster-name> --network-type
    "DUAL"
   ```

4. クラスターがデュアルスタックモードをサポートしていることを確認します。

   コンソールを使用している場合、クラスターの [Connectivity & security] (接続とセキュリティ) タブに移動し、[Network type] (ネットワークタイプ) の値が [Dual-stack mode] (デュアルスタックモード) であることを確認します。

   を使用している場合は AWS CLI、 `describe-db-cluster` コマンドを実行し、クラスターの`NetworkType`値が であることを確認します`dual`。

クラスターエンドポイントで dig コマンドを実行して、関連付けられている IPv6 アドレスを特定します。

```
dig <db-cluster-endpoint> AAAA
```

IPv6 アドレスではなくクラスターエンドポイントを使用して、クラスターに接続します。

デュアルスタックモードのリージョンとバージョンの可用性

機能の可用性とサポートは、 AWS リージョンによって異なります。

リージョンのサポート

Amazon DocumentDB のデュアルスタックモードは現在、すべての商用で使用できます AWS リージョン。

バージョンサポート

デュアルスタックモードは、Amazon DocumentDB バージョン 4.0、5.0、および 8.0 でサポートされています。これらのバージョンのいずれかでデュアルスタックモードにアクセスできない場合は、クラスターで最新のエンジンパッチバージョンが実行されていることを確認してください。

デュアルスタックネットワーククラスターの制限

デュアルスタックネットワーククラスターには、次の制限が適用されます。

- クラスターは、IPv6 プロトコルを排他的に使用することはできません。IPv4 を排他的に使用するか、IPv4 と IPv6 プロトコル を使用することができます (デュアルスタックモード)。
- Amazon DocumentDB は、ネイティブ IPv6 サブネットをサポートしていません。
- デュアルスタックモードを使用するクラスターはプライベートでなければなりません。パブリックにアクセス可能にすることはできません。

VPC でクラスターを作成する

次の手順で VPC 内にクラスターを作成できます。デフォルトの VPC を使用する場合は、ステップ 2 から始めて、既に作成されている VPC とサブネットグループを使用することができます。また、必要に応じて、追加の VPC を作成することもできます。

> **Note**
>
> VPC のクラスターへのパブリックアクセスを可能にするには、VPC 属性の DNS `hostnames` と DNS `resolution` を有効化して、VPC に関する DNS 情報を更新する必要があります。VPC インスタンスの DNS の更新については、「Amazon Virtual Private Cloud ユーザーガイド」の「VPC の DNS 属性の表示と更新」を参照してください。

VPC でクラスターを作成するには、次のステップに従います。

- ステップ 1: VPC を作成する
- ステップ 2: サブネットグループの作成
- ステップ 3: VPC セキュリティグループを作成する
- ステップ 4: VPC にクラスターを作成する

ステップ 1: VPC を作成する

最低 2 つのアベイラビリティーゾーンの中にサブネットを持つ VPC を作成します。これらのサブネットは、サブネットグループを作成するときに使用します。デフォルト VPC がある場合、 AWS リージョン内の各アベイラビリティーゾーンに、自動的にサブネットが作成されます。

詳細については、「DocumentDB クラスターで使用する IPv4-only VPC を作成する」を参照するか、「Amazon Virtual Private Cloud ユーザーガイド」の「VPC を作成する」を参照してください。

ステップ 2: サブネットグループの作成

サブネットグループは VPC 用に作成するサブネット (通常はプライベート) のコレクションで、クラスターに指定します。サブネットグループを使用すると、 AWS CLI または DocumentDB API を使用してクラスターを作成するときに、特定の VPC を指定できます。を使用する場合は AWS マネジメントコンソール、使用する VPC とサブネットを選択するだけです。各サブネットグループには、AWS リージョン内の少なくとも 2 つのアベイラビリティーゾーンに少なくとも 1 つのサブネットが必要です。ベストプラクティスとして、各サブネットグループには、 AWS リージョン内のアベイラビリティーゾーンごとに少なくとも 1 つのサブネットが必要です。

クラスターをパブリックにアクセス可能にするには、サブネットグループのサブネットにインターネットゲートウェイが必要です。サブネット用のインターネットゲートウェイの詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「インターネットゲートウェイを使用して VPC のインターネットアクセスを有効にする」を参照してください。

> **Note**
>
> ローカルゾーンのサブネットグループは、サブネットを 1 つだけ持つことができます。

VPC にクラスターを作成するときに、サブネットグループを選択できます。Amazon DocumentDB は、サブネットと (そのサブネット内の) IP アドレスを選択し、クラスターに関連付けます。サブネットグループが存在しない場合、クラスターを作成すると、Amazon DocumentDB によってデフォルトのサブネットグループが作成されます。DocumentDB では、Elastic Network Interface が作成され、その IP アドレスでクラスターに関連付けられます。クラスターは、そのサブネットを含むアベイラビリティーゾーンを使用します。

このステップでは、サブネットグループを作成し、このグループに VPC 用に作成したサブネットを追加します。

サブネットグループを作成するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [サブネットグループ] を選択します。
3. [作成] を選択します。
4. [名前] には、サブネットグループの名前を入力します。
5. [説明] に、サブネットグループの説明を入力します。
6. [サブネットの追加] セクションの [VPC] で、デフォルトの VPC または作成した VPC を選択します。サブネットを含むアベイラビリティーゾーンを [アベイラビリティーゾーン] から選択し、サブネットを [サブネット] から選択します。
7. [作成] を選択します。

   DocumentDB コンソールのサブネットグループリストに新しいサブネットグループが表示されます。サブネットグループを選択すると、ウィンドウ下部の詳細ペインに、そのグループに関連付けられたすべてのサブネットなどの詳細を表示することができます。

ステップ 3: VPC セキュリティグループを作成する

クラスターを作成する前に、クラスターに関連付ける VPC セキュリティグループを作成します。VPC セキュリティグループを作成しない場合、クラスターを作成するときにデフォルトのセキュリティグループを使用します。クラスターのセキュリティグループを作成する方法につい

ては、DocumentDB クラスターで使用する IPv4-only VPC を作成する「」またはAmazon Virtual Private Cloud ユーザーガイド」の「セキュリティグループを使用して AWS リソースへのトラフィックを制御する」を参照してください。

ステップ 4: VPC にクラスターを作成する

このステップでは、クラスターを作成し、前のステップで作成した VPC 名、サブネットグループ、および VPC セキュリティグループを使用します。

> **Note**
>
> VPC のクラスターをパブリックにアクセス可能にする場合は、VPC 属性の DNS `hostnames` と `DNS resolution` を有効にする必要があります。詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「VPC の DNS 属性を表示して更新する」を参照してください。

クラスターの作成方法の詳細については、「Amazon DocumentDB クラスターの作成」を参照してください。

[Connectivity] (接続) セクションにプロンプトが表示されたら、VPC の名前、サブネットグループ、および VPC セキュリティグループを入力します。

> **Note**
>
> VPC の更新は DocumentDB クラスターで現在サポートされていません。

VPC での Amazon DocumentDB クラスターへのアクセス

Amazon DocumentDB は、VPC 内のクラスターにアクセスするための以下のシナリオをサポートしています。

トピック

- 同じ VPC 内の Amazon EC2 インスタンスがアクセスする、VPC 内のクラスター
- 別の VPC 内の Amazon EC2 インスタンスがアクセスする、VPC 内のクラスター

同じ VPC 内の Amazon EC2 インスタンスがアクセスする、VPC 内のクラスター

VPC 内のクラスターの一般的な用途は、同じ VPC 内の Amazon EC2 インスタンスで実行されるアプリケーションサーバーとデータを共有することです。

同じ VPC 内の EC2 インスタンスとクラスター間のアクセスを管理する方法として最も簡単なのは、次の方法です。

- クラスターが存在する VPC セキュリティグループを作成します。このセキュリティグループを使用して、クラスターへのアクセスを制限できます。例えば、このセキュリティグループのカスタムルールを作成できます。これにより、クラスターを作成したときに割り当てたポートと、開発またはそのほかの目的でクラスターにアクセスするのに使用する IP アドレスを使用して TCP へのアクセスを許可できます。
- EC2 インスタンス (ウェブサーバーとクライアント) が属する VPC セキュリティグループを作成します。このセキュリティグループは、必要に応じて、VPC のルーティングテーブルを介したインターネットから EC2 インスタンスへのアクセスを許可できます。例えば、ポート 22 経由で EC2 インスタンスへの TCP アクセスを許可するルールをこのセキュリティグループに設定できます。
- EC2 インスタンス用に作成したセキュリティグループからの接続を許可するクラスターのセキュリティグループで、カスタムルールを作成します。このルールは、セキュリティグループのメンバーにクラスターへのアクセスを許可します。

別のアベイラビリティーゾーンに、追加のパブリックサブネットとプライベートサブネットがあります。DocumentDB サブネットグループには、2 つ以上のアベイラビリティーゾーンにサブネットが必要です。サブネットが追加されたことで、将来的にマルチ AZ クラスター配置に簡単に切り替えることができるようになります。

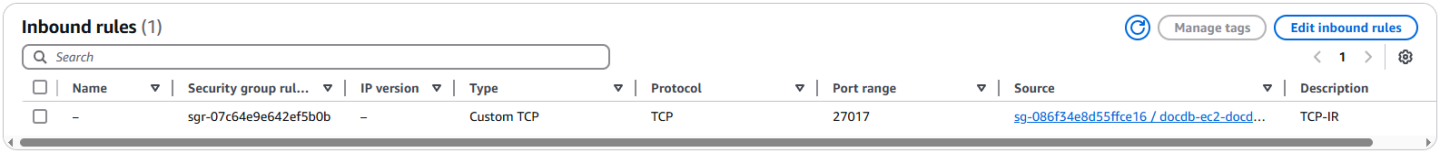
このシナリオのパブリックとプライベートの両方のサブネットを使用する VPC を作成する方法の手順については、「DocumentDB クラスターで使用する IPv4-only VPC を作成する」を参照してください。

> **Tip**
>
> DB クラスターの作成時に、Amazon EC2 インスタンスと DocumentDB クラスター間で自動的にネットワーク接続を設定できるようになります。詳細については、「Amazon EC2 への自動接続」を参照してください。

別のセキュリティグループからの接続を許可する VPC セキュリティグループにルールを作成するには、次の手順を実行します。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。
2. ナビゲーションペインで、[Security Groups] (セキュリティグループ) を見つけて選択します。
3. 他のセキュリティグループのメンバーからのアクセスを許可するセキュリティグループを、選択または作成します。これは、クラスターで使用するセキュリティグループです。[インバウンドルール] タブを選択してから、[インバウンドルールの編集] を選択します。
4. [インバウンドルールの編集] ページで、[ルールの追加] を選択します。
5. [タイプ] では、クラスターの作成時に使用したポートに対応するエントリ ([カスタム TCP] など) を選択します。
6. [ソース] フィールドで、セキュリティグループの ID の入力をスタートすると、一致するセキュリティグループが一覧表示されます。このセキュリティグループによって保護されているリソースへのアクセスを許可するメンバーが所属しているセキュリティグループを選択します。前述のシナリオで、これは EC2 インスタンス向けに使用するセキュリティグループです。
7. 必要に応じて、[タイプ] に [すべての TCP] を、[ソース] フィールドにお客様のセキュリティグループを指定してルールを作成することで、TCP プロトコルのステップを繰り返します。UDP プロトコルを使用する場合は、[All UDP] (すべての UDP) を [Type] (タイプ) と [Source] (送信元) のセキュリティグループとして使用してルールを作成します。
8. [ルールの保存] を選択します。

次の画面には、ソース用のセキュリティグループを含むインバウンドルールが表示されます。



EC2 インスタンスからクラスターに接続する方法の詳細については、「Amazon EC2 への自動接続」を参照してください。

別の VPC 内の Amazon EC2 インスタンスがアクセスする、VPC 内のクラスター

クラスターがアクセスに使用している EC2 インスタンスとは異なる VPC にある場合、VPC ピア接続を使用してそのクラスターにアクセスできます。

VPC ピア接続は、プライベート IP アドレスを使用して 2 つの VPC 間でトラフィックをルーティングすることを可能にするネットワーク接続です。どちらの VPC のリソースも、同じネットワーク内に存在しているかのように、相互に通信できます。独自の VPC 間、別の AWS アカウントの VPCs、または別の の VPC との VPC ピアリング接続を作成できます AWS リージョン。VPC ピア接続の詳細については、Amazon Virtual Private Cloud ユーザーガイドの「VPC ピア接続」を参照してください。

DocumentDB クラスターで使用する IPv4-only VPC を作成する

一般的なシナリオには、Amazon VPC サービスに基づく仮想プライベートクラウド (VPC) 内のクラスターが含まれます。例えば、この VPC は、同じ VPC で実行されているサービスまたはアプリケーションとデータを共有できます。このトピックでは、このシナリオの VPC を作成します。

トピック

- ステップ 1: プライベートサブネットおよびパブリックサブネットを持つ VPC を作成する
- ステップ 2: パブリックアプリケーションの VPC セキュリティグループを作成する
- ステップ 3: プライベートクラスターの VPC セキュリティグループを作成する
- ステップ 4: サブネットグループを作成する
- VPC の削除

クラスターは、アプリケーションからのみ使用可能で、パブリックインターネットからは使用できないようにする必要があります。したがって、パブリックサブネットとプライベートサブネットを持つ VPC を作成します。アプリケーションはパブリックサブネットでホストされることで、パブリックインターネットにアクセスできます。クラスターはプライベートサブネットでホストされます。アプリケーションは、同じ VPC 内でホストされているため、クラスターに接続できます。ただし、クラスターはパブリックインターネットからは使用できないため、セキュリティが向上します。

このトピックの手順では、別のアベイラビリティーゾーンに追加のパブリックサブネットとプライベートサブネットを設定します。これらのサブネットは手順では使用されません。DocumentDB サブネットグループには、2 つ以上のアベイラビリティーゾーンにサブネットが必要です。サブネットを追加すると、複数の DocumentDB インスタンスをより簡単に設定できます。

このトピックでは、Amazon DocumentDB クラスターの VPC の設定について説明します。Amazon VPC の詳細については、「Amazon VPC ユーザーガイド」を参照してください。

> **Tip**
>
> DB クラスターの作成時に、Amazon EC2 インスタンスと DocumentDB クラスター間で自動的にネットワーク接続を設定できるようになります。ネットワーク構成は、このシナリオで説明したものと似ています。詳細については、「Amazon EC2 への自動接続」を参照してください。

ステップ 1: プライベートサブネットおよびパブリックサブネットを持つ VPC を作成する

以下の手順で、パブリックサブネットとプライベートサブネットを持つ VPC を作成します。

VPC とサブネットを作成するには

1. https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。
2. の右上隅で AWS マネジメントコンソール、VPC を作成するリージョンを選択します。この例では、米国西部 (オレゴン) リージョンを使用します。
3. 左上隅の [VPC dashboard] (VPC ダッシュボード) を選択します。VPC の作成を開始するには、[Create VPC] (VPC の作成) を選択します。
4. [VPC Settings] (VPC 設定) の [Resources to create] (作成するリソース) で、[VPC and more] (VPC など) を選択します。
5. [VPC settings] (VPC 設定) で、これらの値を設定します。

   - [Name tag auto-generation] (ネームタグ自動生成) — **`example`**
   - [IPv4 CIDR block] (IPv4 CIDR ブロック) — **`10.0.0.0/16`**
   - [IPv6 CIDR block] (IPv6 CIDR ブロック) — [No IPv6 CIDR block] (IPv6 CIDR ブロックなし)
   - [Tenancy] (テナンシー) — デフォルト
   - [Number of Availability Zones (AZs)] (アベイラビリティーゾーンの数 (AZ)) — 2
   - [Customize AZs] (AZ をカスタマイズする) — デフォルト値を維持します
   - パブリックサブネットの数 — 2
   - [Number of private subnets] (プライベートサブネット数) — 2
   - [Customize subnets CIDR blocks] (サブネット CIDR ブロックをカスタマイズ) — デフォルト値を維持します
   - [NAT gateways ($)] (NAT ゲートウェイ ($)) — なし

- [VPC endpoints] (VPC エンドポイント) — なし
- [DNS options] (DNS オプション) — デフォルト値を維持します

6. [Create VPC (VPC の作成)] を選択します。

ステップ 2: パブリックアプリケーションの VPC セキュリティグループを作成する

次に、パブリックアクセスのためのセキュリティグループを作成します。VPC 内のパブリック EC2 インスタンスに接続するには、インバウンドルールを VPC セキュリティグループに追加します。これにより、インターネットからのトラフィックを接続できるようになります。

VPC セキュリティグループを作成するには

1. https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。
2. [VPC ダッシュボード]、[セキュリティグループ]、[セキュリティグループの作成] の順に選択します。
3. [セキュリティグループの作成] ページで、以下の値を設定します。
   - セキュリティグループ名 - **`example-securitygroup`**
   - [Description] (説明) — **`Application security group`**
   - VPC — 前に作成した VPC を選択します (例: vpc-example)。
4. インバウンドルールをセキュリティグループに追加します。

   a. Secure Shell (SSH) を使用して VPC の EC2 インスタンスへの接続に使用する IP アドレスを決定します。パブリック IP アドレスを決定するには、別のブラウザウィンドウまたはタブで、https://checkip.amazonaws.com のサービスを使用できます。IP アドレスの例は `203.0.113.25/32` です。

   多くの場合、インターネットサービスプロバイダー (ISP) 経由、またはファイアウォールの内側から静的 IP アドレスなしで接続することがあります。この場合は、クライアントコンピュータが使用する IP アドレスの範囲を検索します。

> **Warning**
>
> SSH アクセスに `0.0.0.0/0` を使用すると、すべての IP アドレスが SSH を使ってパブリックインスタンスにアクセスできるようになります。この方法は、テスト環境で短時間なら許容できますが、実稼働環境では安全ではありません。実稼働環境

では、特定の IP アドレスまたは特定のアドレス範囲にのみ、SSH を使ったインスタンスへのアクセスを限定します。

b. [インバウンドルール] セクションで、[ルールの追加] を選択します。

c. 新しいインバウンドルールに次の値を設定して、Amazon EC2 インスタンスへの SSH アクセスを許可します。このようにした後、EC2 インスタンスに接続してアプリケーションとその他のユーティリティをインストールできます。また、EC2 インスタンスに接続して、アプリケーション用のコンテンツをアップロードします。

- タイプ — **SSH**
- ソース — ステップ a で指定した IP アドレスまたはアドレス範囲 (**203.0.113.25/32** など)

d. [ルールを追加] を選択してください。

e. 新しいインバウンドルールに次の値を設定して、アプリケーションに HTTP へのアクセスを許可します。

- タイプ — **HTTP**
- ソース — **0.0.0.0/0**

5. セキュリティグループを作成するには、[Create security group] (セキュリティグループの作成) を選択します。

   セキュリティグループ ID を書き留めます。別の手順で後に必要になります。

ステップ 3: プライベートクラスターの VPC セキュリティグループを作成する

クラスターをプライベートのままにするには、プライベートアクセス用の第 2 のセキュリティグループを作成します。VPC 内のプライベートクラスターに接続するには、アプリケーションからのトラフィックのみを許可するインバウンドルールを VPC セキュリティグループに追加します。

VPC セキュリティグループを作成するには

1. https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。
2. [VPC ダッシュボード]、[セキュリティグループ]、[セキュリティグループの作成] の順に選択します。
3. [セキュリティグループの作成] ページで、以下の値を設定します。

- セキュリティグループ名 - **example-securitygroup**

- [Description] (説明) — **`Instance security group`**
- VPC — 前に作成した VPC を選択します (例: vpc-example)

4. インバウンドルールをセキュリティグループに追加します。

   a. [インバウンドルール] セクションで、[ルールの追加] を選択します。

   b. 新しいインバウンドルールに次の値を設定して、Amazon EC2 インスタンスからポート 27017 への DocumentDB トラフィックを許可します。これを実行した後、アプリケーションからクラスターに接続できます。そうすることで、アプリケーションからのデータをデータベースに保存および取得できるようになります。

      - タイプ — **`Custom TCP`**
      - [Source] (ソース) - このトピックで以前に作成したアプリケーションセキュリティグループの ID (例: sg-9edd5cfb)。

   c. [ルールを追加] を選択してください。

   d. 新しいインバウンドルールに次の値を設定して、アプリケーションに HTTP へのアクセスを許可します。

      - タイプ — **`HTTP`**
      - ソース — **`0.0.0.0/0`**

5. セキュリティグループを作成するには、[Create security group] (セキュリティグループの作成) を選択します。

ステップ 4: サブネットグループを作成する

サブネットグループは VPC に作成するサブネットのコレクションで、クラスター用に指定します。サブネットグループでは、クラスターの作成時に特定の VPC を指定することができます。

サブネットグループを作成するには

1. VPC 内のデータベースのプライベートサブネットを特定します。

   a. https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。

   b. [VPC Dashboard] (VPC ダッシュボード) を選択してから、[Subnets] (サブネット) を選択します。

c. example-subnet-private1-us-west-2a や example-subnet-private2-us-west-2b のような名前である、ステップ 1 で作成したサブネットのサブネット ID に注意してください。サブネットグループを作成するときに、サブネット ID が必要です。

2. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。

   Amazon VPC コンソールではなく、Amazon DocumentDB コンソールに接続してください。

3. ナビゲーションペインで [サブネットグループ] を選択します。
4. [作成] を選択します。
5. [サブネットグループを作成する] ページで、[サブネットグループの詳細] セクションで値を設定します。

   - 名前 — **`example-db-subnet-group`**
   - [Description] (説明) — **`Instance security group`**

6. [サブネットの追加] セクションで、これらの値を設定します。

   - VPC — 前に作成した VPC を選択します (例: vpc-example)
   - アベイラビリティーゾーン — ステップ 1 で作成した両方のアベイラビリティーゾーンを選択します。例: us-west-2a および us-west-2b
   - サブネット — ステップ 1 で作成したプライベートサブネットを選択します。

7. [作成] を選択します。

DocumentDB コンソールのサブネットグループリストに新しいサブネットグループが表示されます。サブネットグループを選択すると、詳細ペインに、詳細を表示することができます。これらの詳細には、グループに関連付けられているすべてのサブネットが含まれます。

> **Note**
>
> この VPC を作成して DocumentDB クラスターに関連付ける場合は、「Amazon DocumentDB クラスターの作成」の手順に従ってクラスターを作成します。

VPC の削除

不要になった場合は、VPC および VPC 内で使用されている他のリソースを削除できます。

> **Note**
>
> このトピックで作成した VPC にリソースを追加した場合は、VPC を削除する前にこれらを削除しなければならない場合があります。例えば、これらのリソースには Amazon EC2 インスタンスや DocumentDB クラスター が含まれる場合があります。詳細については、「Amazon VPC ユーザーガイド」の「VPC の削除」を参照してください。

VPC および関連リソースを削除するには

1. サブネットグループを削除するには、次のようにします。
   a. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
   b. ナビゲーションペインで [サブネットグループ] を選択します。
   c. 削除するサブネットグループを選択します (例: example-db-subnet-group)。
   d. [削除] を選択してから、確認ウィンドウの [削除] を選択します。
2. 次のようにして、VPC ID をメモします。
   a. https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。
   b. [VPC ダッシュボード] を選択してから、[VPC] を選択します。
   c. リストで、作成した VPC を特定します (vpc-example など)。
   d. 作成した VPC の [VPC ID] をメモします。後続のステップで VPC ID が必要になります。
3. セキュリティグループを削除するには、次のようにします。
   a. https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。
   b. [VPC ダッシュボード] を選択してから、[セキュリティグループ] を選択します。
   c. Amazon DocumentDB クラスターのセキュリティグループを選択します (example-securitygroup など)。
   d. [Actions] (アクション) で、[Delete security groups] (セキュリティグループの削除) を選択してから、確認ダイアログで [Delete] (削除) を選択します。
   e. [Security Groups] (セキュリティグループ) ページに戻り、Amazon EC2 インスタンスのセキュリティグループを選択します (example-securitygroup など)。
   f. [Actions] (アクション) で、[Delete security groups] (セキュリティグループの削除) を選択してから、確認ダイアログで [Delete] (削除) を選択します。

4. VPC の削除

   a. https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。

   b. [VPC ダッシュボード] を選択してから、[VPC] を選択します。

   c. 削除する VPC を選択します (vpc-example など)。

   d. [アクション] で、[VPC の削除] を選択します。

      確認ページには、VPC に関連付けられたサブネットを含め、削除される VPC に関連付けられているその他のリソースが表示されます。

   e. 確認ダイアログで、「**delete**」を入力してから、[削除] を選択します。

DocumentDB クラスターで使用するデュアルスタック VPC を作成する

一般的なシナリオには、Amazon VPC サービスに基づく仮想プライベートクラウド (VPC) 内のクラスターが含まれます。この VPC は、同じ VPC で実行しているパブリック Amazon EC2 インスタンスとデータを共有します。このトピックでは、このシナリオの VPC を作成します。

トピック

- ステップ 1: プライベートサブネットおよびパブリックサブネットを持つ VPC を作成する
- ステップ 2: パブリック Amazon EC2 インスタンスの VPC セキュリティグループを作成する
- ステップ 3: プライベートクラスターの VPC セキュリティグループを作成する
- ステップ 4: サブネットグループを作成する
- ステップ 5: デュアルスタックモードの Amazon EC2 インスタンスを作成する
- ステップ 6: デュアルスタックモードでクラスターを作成する
- ステップ 7: Amazon EC2 インスタンスと DB クラスターに接続する
- VPC を削除する

この手順では、デュアルスタックモードで実行されているデータベースで動作する VPC を、このシナリオで作成します。デュアルスタックモードでは、IPv6 アドレッシングプロトコルを介した接続を可能にします。IP アドレスの割り当てについては、「Amazon DocumentDB IP アドレス指定」を参照してください。

デュアルスタックネットワーククラスターは、ほとんどの リージョンでサポートされています。詳細については、「デュアルスタックモードのリージョンとバージョンの可用性」を参照してくださ

い。デュアルスタックモードの制限については、「デュアルスタックネットワーククラスターの制限」を参照してください。

このトピックと IPv4-only トピックでは、同じ VPC にパブリックサブネットとプライベートサブネットを作成します。1 つの VPC に Amazon DocumentDB クラスターを作成し、別の VPC に Amazon EC2 インスタンスを作成する方法については、「VPC での Amazon DocumentDB クラスターへのアクセス」を参照してください。

DocumentDB クラスターは、Amazon EC2 インスタンスでのみ使用でき、パブリックインターネットから使用できないようにする必要があります。したがって、パブリックサブネットとプライベートサブネットを持つ VPC を作成します。EC2 インスタンスは、パブリックインターネットにアクセスできるようにパブリックサブネットでホストされます。クラスターはプライベートサブネットでホストされます。EC2 インスタンスは、同じ VPC 内でホストされているため、クラスターに接続できます。ただし、クラスターはパブリックインターネットからは使用できないため、セキュリティが向上します。

このトピックの手順では、別のアベイラビリティーゾーンに追加のパブリックサブネットとプライベートサブネットを設定します。これらのサブネットは手順では使用されません。DocumentDB サブネットグループには、2 つ以上のアベイラビリティーゾーンにサブネットが必要です。サブネットを追加すると、複数の DocumentDB インスタンスを簡単に設定できます。

デュアルスタックモードを使用するクラスターを作成するには、[Network type] (ネットワークタイプ) 設定として [Dual-stack mode] (デュアルスタックモード) を指定します。クラスターを同じ設定で変更することもできます。クラスターの作成の詳細については、「Amazon DocumentDB クラスターの作成」を参照してください。DB クラスターの変更の詳細については、「Amazon DocumentDB クラスターの変更」を参照してください。

このトピックでは、Amazon DocumentDB クラスターの VPC の設定について説明します。Amazon VPC の詳細については、「Amazon VPC ユーザーガイド」を参照してください。

ステップ 1: プライベートサブネットおよびパブリックサブネットを持つ VPC を作成する

以下の手順で、パブリックサブネットとプライベートサブネットを持つ VPC を作成します。

VPC とサブネットを作成するには

1. https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。
2. の右上隅で AWS マネジメントコンソール、VPC を作成するリージョンを選択します。この例では、米国西部 (オレゴン) リージョンを使用します。

3. 左上隅の [VPC dashboard] (VPC ダッシュボード) を選択します。VPC の作成を開始するには、[Create VPC] (VPC の作成) を選択します。
4. [VPC Settings] (VPC 設定) の [Resources to create] (作成するリソース) で、[VPC and more] (VPC など) を選択します。
5. [VPC settings] (VPC 設定) で、これらの値を設定します。
   - [Name tag auto-generation] (ネームタグ自動生成) — **`example-dual-stack`**
   - [IPv4 CIDR block] (IPv4 CIDR ブロック) — **`10.0.0.0/16`**
   - [IPv6 CIDR block] (IPv6 CIDR ブロック) — [Amazon-provided IPv6 CIDR block] (Amazon 提供の IPv6 CIDR ブロック)
   - [Tenancy] (テナンシー) — デフォルト
   - [Number of Availability Zones (AZs)] (アベイラビリティーゾーンの数 (AZ)) — 2
   - [Customize AZs] (AZ をカスタマイズする) — デフォルト値を維持します
   - パブリックサブネットの数 — 2
   - [Number of private subnets] (プライベートサブネット数) — 2
   - [Customize subnets CIDR blocks] (サブネット CIDR ブロックをカスタマイズ) — デフォルト値を維持します
   - [NAT gateways ($)] (NAT ゲートウェイ ($)) — なし
   - [Egress only internet gateway] (Egress-only インターネットゲートウェイ) — [No] (なし)
   - [VPC endpoints] (VPC エンドポイント) — なし
   - [DNS options] (DNS オプション) — デフォルト値を維持します
6. [Create VPC (VPC の作成)] を選択します。

ステップ 2: パブリック Amazon EC2 インスタンスの VPC セキュリティグループを作成する

次に、パブリックアクセスのためのセキュリティグループを作成します。VPC 内のパブリック EC2 インスタンスに接続するには、インターネットから接続するトラフィックを許可するインバウンドルールを VPC セキュリティグループに追加します。

VPC セキュリティグループを作成するには

1. https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。

2. [VPC ダッシュボード]、[セキュリティグループ]、[セキュリティグループの作成] の順に選択します。
3. [セキュリティグループの作成] ページで、以下の値を設定します。
   - セキュリティグループ名 - **example-dual-stack-securitygroup**
   - [Description] (説明) — **Dual-stack security group**
   - VPC — 前に作成した VPC を選択します (例: vpc-example-dual-stack)。
4. インバウンドルールをセキュリティグループに追加します。
   a. Secure Shell (SSH) を使用して VPC の EC2 インスタンスへの接続に使用する IP アドレスを決定します。パブリック IP アドレスを決定するには、別のブラウザウィンドウまたはタブで、https://checkip.amazonaws.com のサービスを使用できます。

      インターネットプロトコルバージョン 4 (IPv4) アドレス範囲の例は です`203.0.113.25/32`。インターネットプロトコルバージョン 6 (IPv6) のアドレス範囲の例は `2001:db8:1234:1a00::/64` です。

      多くの場合、インターネットサービスプロバイダー (ISP) 経由、またはファイアウォールの内側から静的 IP アドレスなしで接続することがあります。この場合は、クライアントコンピュータが使用する IP アドレスの範囲を検索します。

      > **Warning**
      >
      > IPv4 の `0.0.0.0/0` または IPv6 の `::0` を使用している場合は、すべての IP アドレスが SSH を使ってパブリックインスタンスにアクセスできるようにします。この方法は、テスト環境で短時間なら許容できますが、実稼働環境では安全ではありません。実稼働環境では、特定の IP アドレスまたは特定のアドレス範囲にのみ、インスタンスへのアクセスを許可します。

   b. [インバウンドルール] セクションで、[ルールの追加] を選択します。
   c. 新しいインバウンドルールに次の値を設定して、Amazon EC2 インスタンスへの SSH アクセスを許可します。このようにした後、EC2 インスタンスに接続してアプリケーションやその他のユーティリティをインストールできます。EC2 インスタンスへのアクセスできるように IP アドレスを指定します。
      - タイプ — **SSH**

- [Source] (ソース) — ステップ a で指定した IP アドレスまたは範囲。IPv4 アドレス範囲の例は **203.0.113.25/3**2 です。IPv6 アドレス範囲の例は **2001:DB8::/32** です。

d. [ルールを追加] を選択してください。

5. セキュリティグループを作成するには、[Create security group] (セキュリティグループの作成) を選択します。

   セキュリティグループ ID を書き留めます。別の手順で後に必要になります。

ステップ 3: プライベートクラスターの VPC セキュリティグループを作成する

クラスターをプライベートのままにするには、プライベートアクセス用の第 2 のセキュリティグループを作成します。VPC 内のプライベートクラスターに接続するには、VPC セキュリティグループにインバウンドルールを追加します。これにより、Amazon EC2 インスタンスからのトラフィックのみを許可します。

VPC セキュリティグループを作成するには

1. https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。
2. [VPC ダッシュボード]、[セキュリティグループ]、[セキュリティグループの作成] の順に選択します。
3. [セキュリティグループの作成] ページで、以下の値を設定します。
   - セキュリティグループ名 - **example-dual-stack-cluster-securitygroup**
   - [Description] (説明) — **Dual-stack cluster security group**
   - VPC — 前に作成した VPC を選択します (例: vpc-example-dual-stack)
4. インバウンドルールをセキュリティグループに追加します。

   a. [インバウンドルール] セクションで、[ルールの追加] を選択します。

   b. 新しいインバウンドルールに次の値を設定して、Amazon EC2 インスタンスからポート 27017 への DocumentDB トラフィックを許可します。これを実行した後、EC2 インスタンスからクラスターに接続できます。これにより、EC2 インスタンスからデータベースにデータを送信できるようになります。

      - タイプ — **Custom TCP**
      - [Source] (ソース) - このトピックで以前に作成した EC2 セキュリティグループの ID (例: sg-9edd5cfb)。

5. セキュリティグループを作成するには、[Create security group] (セキュリティグループの作成) を選択します。

ステップ 4: サブネットグループを作成する

サブネットグループは VPC に作成するサブネットのコレクションで、クラスター用に指定します。サブネットグループを使用することにより、クラスターを作成するときに、特定の VPC を指定することができます。DUAL 互換のサブネットグループを作成するには、すべてのサブネットが DUAL 互換である必要があります。DUAL 互換であるためには、サブネットに IPv6 CIDR が関連付けられている必要があります。

サブネットグループを作成するには

1. VPC 内のデータベースのプライベートサブネットを特定します。

   a. https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。

   b. [VPC Dashboard] (VPC ダッシュボード) を選択してから、[Subnets] (サブネット) を選択します。

   c. example-dual-stack-subnet-private1-us-west-2a や example-dual-stack-subnet-private2-us-west-2b のような名前である、ステップ 1 で作成したサブネットのサブネット ID に注意してください。サブネットグループを作成するときに、サブネット ID が必要です。

2. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。

   Amazon VPC コンソールではなく、Amazon DocumentDB コンソールに接続してください。

3. ナビゲーションペインで [サブネットグループ] を選択します。

4. [作成] を選択します。

5. [サブネットグループを作成する] ページで、[サブネットグループの詳細] セクションで値を設定します。

   - 名前 — **`example-dual-stack-cluster-subnet-group`**
   - [Description] (説明) — **`Dual-stack cluster subnet group`**

6. [サブネットの追加] セクションで、これらの値を設定します。

   - VPC — 前に作成した VPC を選択します (例: vpc-example-dual-stack)
   - アベイラビリティーゾーン — ステップ 1 で作成した両方のアベイラビリティーゾーンを選択します。例: us-west-2a および us-west-2b

- サブネット — ステップ 1 で作成したプライベートサブネットを選択します。

7. [作成] を選択します。

DocumentDB コンソールのサブネットグループリストに新しいサブネットグループが表示されます。サブネットグループを選択すると、詳細ペインに、詳細を表示することができます。これらの詳細には、グループに関連付けられているすべてのサブネットが含まれます。

ステップ 5: デュアルスタックモードの Amazon EC2 インスタンスを作成する

Amazon EC2 インスタンスを作成するには、「Amazon Elastic Compute Cloud ユーザーガイド」の「インスタンス起動ウィザードを使用して EC2 インスタンスを起動する」の指示に従います。

次に示すように、[Configure Instance Details] (インスタンスの詳細の設定) ページで次の値を設定し、他の値はデフォルトのままにします。

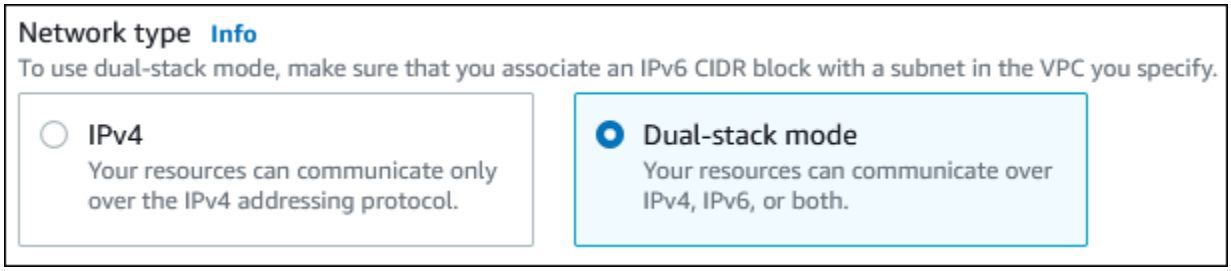
- ネットワーク — パブリックサブネットとプライベートサブネットの両方を持つ既存の VPC を選択します (ステップ 1: プライベートサブネットおよびパブリックサブネットを持つ VPC を作成する で作成した vpc-example-dual-stack-vpc (vpc-identifier) など)。
- [Subnet] (サブネット) — 既存のパブリックサブネットを選択します (ステップ 2: パブリック Amazon EC2 インスタンスの VPC セキュリティグループを作成する で作成した subnet-identifier | example-dual-stack-subnet-public1-us-east-2a | us-east-2a など)。
- [Auto-assign Public IP] (パブリック IP の自動割り当て)] — [Enable] (有効化) を選択します。
- [Auto-assign IPv6 IP] — [Enable] (有効化) を選択します。
- [Firewall (security groups)] (ファイアウォール (セキュリティグループ)) — [Select an existing security group] (既存のセキュリティグループを選択する) を選択します。
- 共通セキュリティグループ — ステップ 2: パブリック Amazon EC2 インスタンスの VPC セキュリティグループを作成する で作成された example-dual-stack-securitygroup など、既存のセキュリティグループを選択します。選択するセキュリティグループに、Secure Shell (SSH) および HTTP アクセスのインバウンドルールが含まれていることを確認します。

ステップ 6: デュアルスタックモードでクラスターを作成する

このステップでは、デュアルスタックモードで実行する DB クラスターを作成します。!!! 注: このセクションは、コンソール で IPv6 の更新が行われたら編集する必要があります!!!

デュアルスタックモードでクラスターを作成するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. コンソールの右上隅で、DocumentDB クラスターを作成する AWS リージョン を選択します。この例では、米国東部 (オハイオ) リージョンを使用します。
3. ナビゲーションペインで [Clusters] (クラスター) を選択します。
4. [Clusters] (クラスター) リストページで、[Create] (作成) を選択します。
5. [Amazon DocumentDB クラスターの作成] ページで、[インスタンスベースのクラスター] オプションが選択されていることを確認します。
6. [接続] セクションの [ネットワークタイプ] で、[デュアルスタックモード] を選択します。



7. ページの下部で、[詳細設定の表示] をオンにします。
8. [ネットワーク設定] セクションで、これらの値を設定します。

   - [Virtual private cloud (VPC)] (仮想プライベートクラウド (VPC)) — パブリックサブネットとプライベートサブネットの両方を持つ既存の VPC を選択します (ステップ 1: プライベートサブネットおよびパブリックサブネットを持つ VPC を作成する で作成した vpc-example-dual-stack (vpc-identifier) など)。

     VPC の各サブネットは異なるアベイラビリティーゾーンに存在している必要があります。
   - [Subnet group] (サブネットグループ) — VPC のサブネットグループ (ステップ 4: サブネットグループを作成する で作成した example-dual-stack-cluster-subnet-group など)。
   - [Public access] (公開アクセス) — [No] (いいえ) を選択します。
   - [VPC security group (firewall)] (VPC セキュリティグループ (ファイアウォール)) — [Choose existing] (既存を選択) を選択します。
   - [Existing VPC security groups] (既存の VPC セキュリティグループ) — プライベートアクセス用に設定されている既存の VPC セキュリティグループを選択します (ステップ 3: プライベートクラスターの VPC セキュリティグループを作成する で作成した example-dual-stack-cluster-securitygroup など)。

他のセキュリティグループ (デフォルトのセキュリティグループなど) は、それぞれの対応する [X] を選択して削除します。

- [アベイラビリティーゾーン] — ステップ 1 で作成したアベイラビリティーゾーンを選択します。例: us-west-2a。

  AZ 間のトラフィックを回避するには、クラスターと EC2 インスタンスが同じアベイラビリティーゾーンにあることを確認してください。

9. 残りのセクションで、クラスター設定を指定します。各設定の詳細については、「Amazon DocumentDB クラスターの作成」を参照してください。

ステップ 7: Amazon EC2 インスタンスと DB クラスターに接続する

Amazon EC2 インスタンスと DocumentDB クラスターをデュアルスタックモードで作成した後、IPv6 プロトコルを使用して各インスタンスに接続できます。IPv6 プロトコルを使用して EC2 インスタンスに接続するには、「Amazon Elastic Compute Cloud ユーザーガイド」の「EC2 インスタンスに接続する」の指示に従ってください。

EC2 インスタンスから DocumentDB クラスターに接続するには、「Amazon EC2 の手動接続」トピックの「ステップ 5: MongoDB シェルをインストールする」の手順に従います (同じ手順で後続のステップ 6 とステップ 7 に進みます)。

VPC を削除する

不要になった場合は、VPC および VPC 内で使用されている他のリソースを削除できます。

> **Note**
>
> このトピックで作成した VPC にリソースを追加した場合は、VPC を削除する前にこれらを削除しなければならない場合があります。例えば、これらのリソースには Amazon EC2 インスタンスや DocumentDB クラスター が含まれる場合があります。詳細については、「Amazon VPC ユーザーガイド」の「VPC の削除」を参照してください。

VPC および関連リソースを削除するには

1. サブネットグループを削除するには、次のようにします。

a. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。

b. ナビゲーションペインで [サブネットグループ] を選択します。

c. 削除するサブネットグループを選択します (example-dual-stack-cluster-subnet-group など)。

d. [削除] を選択してから、確認ウィンドウの [削除] を選択します。

2. 次のようにして、VPC ID をメモします。

a. https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。

b. [VPC ダッシュボード] を選択してから、[VPC] を選択します。

c. リストで、作成した VPC を特定します (vpc-example-dual-stack など)。

d. 作成した VPC の [VPC ID] をメモします。後続のステップで VPC ID が必要になります。

3. セキュリティグループを削除するには、次のようにします。

a. https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。

b. [VPC ダッシュボード] を選択してから、[セキュリティグループ] を選択します。

c. Amazon DocumentDB クラスターのセキュリティグループを選択します (example-dual-stack-securitygroup など)。

d. [Actions] (アクション) で、[Delete security groups] (セキュリティグループの削除) を選択してから、確認ダイアログで [Delete] (削除) を選択します。

e. [Security Groups] (セキュリティグループ) ページに戻り、Amazon EC2 インスタンスのセキュリティグループを選択します (example-securitygroup など)。

f. [Actions] (アクション) で、[Delete security groups] (セキュリティグループの削除) を選択してから、確認ダイアログで [Delete] (削除) を選択します。

4. 次のようにして、NAT ゲートウェイを削除します。

a. https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。

b. [VPC ダッシュボード] を選択してから、[セキュリティグループ] を選択します。

c. 作成した VPC の NAT ゲートウェイを選択します。VPC ID を使用して、適切な NAT ゲートウェイを識別します。

d. [Actions] (アクション) で、[Delete NAT gateway] (NAT ゲートウェイの削除) を選択します。

e. 確認ダイアログで、「**delete**」を入力してから、[削除] を選択します。

5. VPC の削除

   a. https://console.aws.amazon.com/vpc で Amazon VPC コンソールを開きます。
   b. [VPC ダッシュボード] を選択してから、[VPC] を選択します。
   c. 削除する VPC を選択します (vpc-example-dual-stack など)。
   d. [アクション] で、[VPC の削除] を選択します。

      確認ページには、VPC に関連付けられたサブネットを含め、削除される VPC に関連付けられているその他のリソースが表示されます。
   e. 確認ダイアログで、「**delete**」を入力してから、[削除] を選択します。

6. Elastic IP アドレスを解放します。

   a. EC2 コンソール (https://console.aws.amazon.com/ec2) を開きます。
   b. [EC2 ダッシュボード] を選択してから、[Elastic IP] を選択します。
   c. 解放する Elastic IP アドレスを選択します。
   d. [Actions] (アクション) で、[Release Elastic IP addresses] (Elastic IP アドレスの解放) を選択します。
   e. 確認ダイアログで、[解放] を選択します。

Amazon DocumentDB でのバックアップと復元

Amazon DocumentDB (MongoDB 互換) は、データを Amazon Simple Storage Service (Amazon S3) に 1～35 日間、継続的にバックアップするため、バックアップ保持期間の任意の時点にすばやく復元できます。Amazon DocumentDB は、この継続的なバックアッププロセスの一環として、データの自動スナップショットも取得します。

> **Note**
>
> これらはサービス管理された Amazon S3 バケットであり、バックアップファイルにアクセスすることはできません。独自のバックアップを制御するには、「」を参照してくださいデータのダンプ、復元、インポート、およびエクスポート。

クラスターのデータの手動スナップショットを作成して、バックアップ保持期間を超えてバックアップデータを保持することもできます。バックアッププロセスは、クラスターのパフォーマンスに影響を与えません。

このセクションでは、Amazon DocumentDB でのバックアップ機能のユースケースについて説明し、Amazon DocumentDB クラスターのためのバックアップ管理法を示します。

トピック

- バックアップと復元: 概念
- バックアップストレージの使用状況を確認する
- データのダンプ、復元、インポート、およびエクスポート
- クラスタースナップショットに関する考慮事項
- 自動スナップショットと手動スナップショットの比較
- 手動クラスタースナップショットの作成
- Amazon DocumentDB クラスタースナップショットのコピー
- Amazon DocumentDB クラスタースナップショットの共有
- クラスタースナップショットからの復元
- 特定時点への復元
- クラスタースナップショットの削除

バックアップと復元: 概念

名詞	説明	API (動詞)

バックアップの保存期間	特定時点への復元を実行できる、1～35 日間の期間。	`create-db-cluster`
		`modify-db-cluster`
		`restore-db-cluster-to-point-in-time`

Amazon DocumentDB のストレージボリューム	3 つのアベイラビリティーゾーンに渡って 6 つの方法でデータをレプリケートする、可用性と耐久性に優れたストレージボリューム。Amazon DocumentDB クラスターは、クラスター内のインスタンス数に関係なく、高い耐久性があります。	`create-db-cluster` `delete-db-cluster`
バックアップウィンドウ	自動スナップショット	`create-db-cluster`

名詞	説明	API (動詞)
	が実行される 1 日の時間帯。	describe-db-cluster modify-db-cluster
自動スナップショット	クラスターの完全なバックアップである毎日のスナップショット。これは Amazon DocumentDB の継続的なバックアッププロセスによって自動的に作成されます。	restore-db-cluster-from-snapshot describe-db-cluster-snapshot-attributes describe-db-cluster-snapshots
手動スナップショット	バックアップ期間を超えてクラスターの完全バックアップを保持するために、手動で作成するスナップショット。	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

バックアップストレージの使用状況を確認する

Amazon DocumentDB バックアップストレージは、バックアップ保持期間内の継続的なバックアップと、保持期間外の手動スナップショットで構成されます。バックアップストレージの使用状況を管理するには、バックアップの保存間隔を短縮するか、不要になった古い手動スナップショットを削除するか、またはその両方を行います。Amazon DocumentDB バックアップの一般情報については、Amazon DocumentDB でのバックアップと復元 を参照してください。Amazon DocumentDB バックアップストレージの料金情報については、Amazon DocumentDB の料金 を参照してください。

コストを管理するには、保存期間を超えて保持されている継続的バックアップと手動スナップショットに使用されているストレージの量をモニタリングできます。次に、バックアップの保存間隔を短縮し、不要になった手動スナップショットを削除できます。

次に示すように、 Amazon DocumentDB バックアップに使用されているストレージの量を確認およびモニタリングするために、Amazon CloudWatch メトリクスの `TotalBackupStorageBilled` 、`SnapshotStorageUsed` 、および `BackupRetentionPeriodStorageUsed` を使用できます。

- `BackupRetentionPeriodStorageUsed` は現時点で継続的なバックアップの保存に使用されているバックアップストレージの量を表します。このメトリクス値は、クラスターボリュームのサイズと、保持期間中に行った変更数によって変わります。ただし、請求のため、このメトリクスが保持期間中に累積的なクラスターボリュームのサイズを超えることはありません。たとえば、クラスターサイズが 100 GiB で、保持期間が 2 日間である場合、`BackRetentionPeriodStorageUsed` の最大値は 200 GiB (100 GiB + 100 GiB) となります。

- `SnapshotStorageUsed` は、バックアップ保持期間を超えて手動スナップショットの保存に使用されるバックアップストレージの量を表します。保持期間内に作成された手動スナップショットは、バックアップストレージに対してカウントされません。同様に、自動スナップショットはバックアップストレージに対して計算されません。各スナップショットのサイズは、そのスナップショットを作成した時点のクラスターボリュームのサイズです。`SnapshotStorageUsed` 値は、保持するスナップショットの数と、各スナップショットのサイズによって異なります。たとえば、保存期間外のスナップショットが 1 つあり、このスナップショットを作成した時点のクラスターのボリュームサイズが 100 GiB であったとします。この場合の `SnapshotStorageUsed` の量は 100 GiB です。

- `TotalBackupStorageBilled` は `BackupRetentionPeriodStorageUsed` と `SnapshotStorageUsed` の合計から、1 日のクラスターボリュームのサイズと等しい無料バックアップストレージの量を引いた値になります。例えば、クラスターサイズが 100 GiB、保持期間が 1 日であり、保持期間外に 1 つのスナップショットがある場合、`TotalBackupStorageBilled` は 100 GiB (100 GiB + 100 GiB - 100 GiB) となります。

- これらのメトリクスは、Amazon DocumentDB クラスターごとに個別に計算されます。

CloudWatch コンソールを通して CloudWatch メトリクスを使用することで、Amazon DocumentDB クラスターをモニタリングし、レポートを作成することができます。CloudWatch のメトリクスを使用する方法については、Amazon DocumentDB のモニタリングとログ記録 を参照してください。

データのダンプ、復元、インポート、およびエクスポート

`mongodump`、`mongorestore`、`mongoexport`、および `mongoimport` の各ユーティリティを使用して、Amazon DocumentDB クラスターの内外にデータを移動できます。このセクションでは、これらの各ツールの目的と、パフォーマンスの向上に役立つ設定について説明します。

> **MongoDB データベースツールのバージョン要件**
>
> Amazon DocumentDB には、バージョン 100.11.0 までの MongoDB データベースツールを使用します。 Amazon DocumentDB ダウンロードするには、MongoDB ウェブサイトの「MongoDB データベースツールのリリース」を参照してください。 MongoDB

トピック

- mongodump
- mongorestore
- mongoexport
- mongoimport
- チュートリアル

mongodump

mongodump ユーティリティは、MongoDB データベースのバイナリ (BSON) バックアップを作成します。mongodump のツールは、バイナリ形式でデータを保存することでサイズを縮小できるため、Amazon DocumentDB クラスターへの復元時にソース MongoDB デプロイからデータをダンプする方法として最適です。

インスタンスで利用可能なリソースや、コマンドの実行に使用しているマシンによっては、mongodump のオプションを使用したパラレルコレクションのダンプ数をデフォルトの 1 から増やすことで、--numParallelCollections を高速化できます。目安として、Amazon DocumentDB クラスターのプライマリインスタンスで [vCPU] ごとに 1 つのワーカーから開始します。

使用例

Amazon DocumentDB クラスター (sample-cluster) での mongodump のユーティリティの使用例を次に示します。

```
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 ユーティリティを使用すると、mongodump ユーティリティで作成されたデータベースのバイナリ (BSON) バックアップを復元できます。復元のパフォーマンスを向上させるには、--numInsertionWorkersPerCollection オプションを使用して復元時に各コレクションのワーカー数 (デフォルトは 1) を増やします。目安として、Amazon DocumentDB クラスターのプライマリインスタンスで [vCPU] ごとに 1 つのワーカーから開始します。

使用例

Amazon DocumentDB クラスター (sample-cluster) での mongorestore のユーティリティの使用例を次に示します。

```
mongorestore --ssl \
    --host="sample-cluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --username=sample-user \
    --password=abc0123 \
    --sslCAFile global-bundle.pem <fileToBeRestored>
```

mongoexport

mongoexport のツールは、Amazon DocumentDB 中のデータを JSON、CSV、または TSV ファイル形式にエクスポートします。mongoexport ツールは、人間またはマシンが読み取り可能なデータをエクスポートする方法として最適です。

> **Note**
>
> mongoexport は並列エクスポートを直接サポートしていません。ただし、コレクションごとに複数の mongoexport ジョブを同時に実行することで、パフォーマンスを向上させることができます。

使用例

Amazon DocumentDB クラスター (sample-cluster) での mongoexport ツールの使用例を次に示します。

```
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

mongoimport のツールは、JSON、CSV、または TSV ファイルの内容を Amazon DocumentDB クラスターにインポートします。--numInsertionWorkers パラメータを使用すると、インポート (デフォルトは 1) を並列化および高速化できます。

使用例

Amazon DocumentDB クラスター (sample-cluster) でのmongoimport ツールの使用例を次に示します。

```
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
```

チュートリアル

次のチュートリアルでは mongodump、mongorestore、mongoexport、および mongoimport の各ユーティリティを使用して Amazon DocumentDB クラスターの内外にデータを移動する方法について説明します。

1. 前提条件 - 開始する前に、Amazon DocumentDB クラスターがプロビジョニングされていること、およびクラスターと同じ VPC 内の Amazon EC2 インスタンスにアクセスできることを確認してください。詳細については、「Amazon EC2 を使用した接続」を参照してください。

   mongo ユーティリティツールを使用するには、次のように、EC2 インスタンスに mongodb-org-tools パッケージがインストールされている必要があります。

   ```
   sudo yum install mongodb-org-tools-4.0.18
   ```

   Amazon DocumentDB はデフォルトで TLS (Transport Layer Security) 暗号化を使用するため、次のように mongo シェルを使用するために Amazon RDS 認証機関 (CA) ファイルもダウンロードする必要があります。

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

2. サンプルデータをダウンロードする - このチュートリアルでは、レストランに関する情報が含まれているサンプルデータをダウンロードします。

```
wget https://raw.githubusercontent.com/ozlerhakan/mongodb-json-files/master/
datasets/restaurant.json
```

3. Amazon DocumentDB にサンプルデータをインポートする - データは論理 JSON 形式であるため、mongoimport のユーティリティを使用してデータを Amazon DocumentDB クラスターにインポートします。

```
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. **mongodump** でデータをダンプする - Amazon DocumentDB クラスターにデータをインポートしたので、mongodump ユーティリティを使用してこのデータのバイナリダンプを取得できます。

```
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. **restaurants** コレクションを削除する - restaurants のコレクションを business のデータベースに復元する前に、次のように、そのデータベース内にある既存のコレクションを最初に削除する必要があります。

```
use business
```

```
db.restaurants.drop()
```

6. **mongorestore** でデータを復元する - ステップ 3 からのデータのバイナリダンプを使用して、mongorestore のユーティリティを使用してデータを Amazon DocumentDB クラスターに復元できます。

```
mongorestore --ssl \
    --host="tutorialCluster.us-east-1.docdb.amazonaws.com:27017" \
    --numParallelCollections 4 \
    --username=<yourUsername> \
    --password=<yourPassword> \
    --sslCAFile global-bundle.pem restaurantDump.bson
```

7. **mongoexport** を使用してデータをエクスポートする - チュートリアルを完了するには、クラスターからデータを JSON ファイルの形式でエクスポートします。ステップ 1 でインポートしたファイルと同じ結果になります。

```
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. 検証 - 次のコマンドを使用して、ステップ 5 の出力がステップ 1 と同じ結果をもたらすことを検証できます。

```
wc -l restaurant.json
```

このコマンドからの出力:

```
2548 restaurant.json
```

```
wc -l restaurant2.json
```

このコマンドからの出力:

```
2548 restaurant2.json
```

クラススナップショットに関する考慮事項

Amazon DocumentDB は、クラスターのバックアップウィンドウ中にクラスターの自動スナップショットを毎日作成します。Amazon DocumentDB は、指定したバックアップ保持期間に従って、クラスターの自動スナップショットを保存します。必要に応じて、バックアップ保持期間内の任意の時点でクラスターを復旧できます。同じクラスターに対して同じリージョンでコピー操作が実行されている間は、自動スナップショットは発生しません。

トピック

- バックアップストレージ
- バックアップウィンドウ
- バックアップの保存期間
- クラススナップショットの暗号化をコピーする

自動のクラススナップショットに加えて、手動でクラスターのスナップショットを作成できます。自動スナップショットと手動スナップショットの両方をコピーできます。詳細については、「手動クラススナップショットの作成」および「Amazon DocumentDB クラススナップショットのコピー」を参照してください。

> **Note**
>
> 自動スナップショットを作成するには、クラスターのスタータスが 使用可能 である必要があります。
> Amazon DocumentDB 自動クラススナップショットを共有することはできません。回避策として、自動スナップショットをコピーして手動スナップショットを作成し、そのコピーを共有できます。スナップショットのコピーの詳細については、「Amazon DocumentDB クラススナップショットのコピー」を参照してください。スナップショットからクラスターを復元する方法の詳細については、「クラススナップショットからの復元」を参照してください。

バックアップストレージ

それぞれの Amazon DocumentDB バックアップストレージ AWS リージョン は、バックアップ保持期間に必要なバックアップストレージで構成されます。これには、そのリージョンの自動および手動のクラススナップショットが含まれます。デフォルトのバックアップ保持期間は 1 日です。

バックアップストレージの料金に関するさらなる情報については、Amazon DocumentDB の料金 を参照してください。

クラスターを削除すると、自動スナップショットはすべて削除され、復旧することはできません。ただし、クラスターを削除しても、手動スナップショットは削除されません。クラスターを削除する前に Amazon DocumentDB が最終スナップショット (手動スナップショット) を作成するように選択した場合は、最終スナップショットを使用してクラスターを復元できます。

スナップショットとストレージの詳細については、「バックアップストレージの使用状況を確認する」を参照してください。

バックアップウィンドウ

自動スナップショットは、任意のバックアップウィンドウ中に毎日行われます。バックアップウィンドウに割り当てられた時間より長い時間がスナップショットに必要な場合、バックアップウィンドウが終了した後もバックアップが完了するまでバックアッププロセスが継続します。クラスターの週 1 回のメンテナンスウィンドウとバックアップウィンドウは重複できません。

クラスターの作成時に任意のバックアップウィンドウを指定しない場合、Amazon DocumentDB がデフォルトの 30 分のバックアップウィンドウを割り当てます。このウィンドウは、クラスターのリージョンに関連付けられた 8 時間の時間帯からランダムに選択されます。クラスターを変更すると、バックアップウィンドウを変更できます。詳細については、「Amazon DocumentDB クラスターの変更」を参照してください。

リージョン名	リージョン	UTC 時間帯
米国東部 (オハイオ)	us-east-2	03:00 ~ 11:00
米国東部 (バージニア北部)	us-east-1	03:00 ~ 11:00
米国西部 (オレゴン)	us-west-2	06:00 ~ 14:00
アフリカ (ケープタウン)	af-south-1	03:00 ~ 11:00
アジアパシフィック (香港)	ap-east-1	06:00 ~ 14:00
アジアパシフィック (ハイデラバード)	ap-south-2	06:30 ~ 14:30

リージョン名	リージョン	UTC 時間帯
アジアパシフィック (マレーシア)	ap-southeast-5	13:00 ~ 21:00
アジアパシフィック (ムンバイ)	ap-south-1	06:00 ~ 14:00
アジアパシフィック (大阪)	ap-northeast-3	12:00 ~ 20:00
アジアパシフィック (ソウル)	ap-northeast-2	13:00 ~ 21:00
アジアパシフィック (シンガポール)	ap-southeast-1	14:00 ~ 22:00
アジアパシフィック (シドニー)	ap-southeast-2	12:00 ~ 20:00
アジアパシフィック (ジャカルタ)	ap-southeast-3	08:00 ~ 16:00
アジアパシフィック (メルボルン)	ap-southeast-4	11:00 ~ 19:00
アジアパシフィック (タイ)	ap-southeast-7	15:00 ~ 23:00
アジアパシフィック (東京)	ap-northeast-1	13:00 ~ 21:00
カナダ (中部)	ca-central-1	03:00 ~ 11:00
カナダ西部 (カルガリー)	ca-west-1	18:00 ~ 02:00
中国 (北京)	cn-north-1	06:00 ~ 14:00
中国 (寧夏)	cn-northwest-1	06:00 ~ 14:00
欧州 (フランクフルト)	eu-central-1	21:00 ~ 05:00
欧州 (チューリッヒ)	eu-central-2	02:00 ~ 10:00
欧州 (アイルランド)	eu-west-1	22:00 ~ 06:00

リージョン名	リージョン	UTC 時間帯
欧州 (ロンドン)	eu-west-2	22:00 ~ 06:00
欧州 (ミラノ)	eu-south-1	02:00 ~ 10:00
欧州 (パリ)	eu-west-3	23:59 ~ 07:29
欧州 (スペイン)	eu-south-2	02:00 ~ 10:00
欧州 (ストックホルム)	eu-north-1	04:00 ~ 12:00
メキシコ (中部)	mx-central-1	03:00 ~ 11:00
中東 (UAE)	me-central-1	05:00 ~ 13:00
南米 (サンパウロ)	sa-east-1	00:00 ~ 08:00
イスラエル (テルアビブ)	il-central-1	04:00 ~ 12:00
AWS GovCloud (米国東部)	us-gov-east-1	17:00 ~ 01:00
AWS GovCloud (米国西部)	us-gov-west-1	06:00 ~ 14:00

バックアップの保存期間

バックアップ保持期間は、自動的に削除される前に自動バックアップが保持される日数です。Amazon DocumentDB では、1～35 日間のバックアップ保持期間をサポートしています。

クラスターを作成するとき、バックアップ保持期間を設定できます。バックアップ保持期間を明示的に設定しない場合、デフォルトのバックアップ保持期間である 1 日がクラスターに割り当てられます。クラスターを作成したら、 AWS マネジメントコンソール または を使用してクラスターを変更することで、バックアップ保持期間を変更できます AWS CLI。詳細については、「Amazon DocumentDB クラスターの変更」を参照してください。

クラススナップショットの暗号化をコピーする

クラスターとスナップショットの暗号化は、 KMS 暗号化キーに基づいています。 KMS キー ID は、その KMS 暗号化キーの Amazon リソースネーム (ARN)、 KMS キー識別子、または KMS キーエイリアスのいずれかとなります。

以下のガイドラインと制限が適用されます。

- 暗号化は、スナップショットの作成時にクラスターから推測されます。クラスターが暗号化されている場合、そのクラスターのスナップショットは同じ KMS キーで暗号化されます。クラスターが暗号化されていない場合、スナップショットは暗号化されません。
- 暗号化されたクラスタースナップショットを Amazon Web Services アカウントからコピーする場合、KmsKeyId の値を指定して新しい KMS 暗号化キーでコピーを暗号化できます。KmsKeyId の値を指定しないと、クラスタースナップショットのコピーはコピー元のクラスタースナップショットと同じ KMS キーで暗号化されます。
- 別の Amazon Web Services アカウントと共有されている暗号化されたクラスタースナップショットをコピーする場合は、KmsKeyId の値を指定する必要があります。
- 暗号化されたクラスタースナップショットを別の Amazon Web Services リージョンにコピーするには、KmsKeyId を、コピー先のリージョンでクラスタースナップショットのコピーの暗号化に使用する KMS キー ID に設定します。KMS 暗号化キーは、それが作成された Amazon Web Services リージョンに固有のものであるため、ある Amazon Web Services リージョンの暗号化キーを別の Amazon Web Services リージョンで使用することはできません。
- 暗号化されていないクラスタースナップショットをコピーして KmsKeyId パラメータに値を指定すると、エラーが返されます。

自動スナップショットと手動スナップショットの比較

Amazon DocumentDB (MongoDB 互換) の自動スナップショットおよび手動スナップショットの主な機能を以下に示します。

Amazon DocumentDB 自動スナップショットには以下の主な機能があります。

- 自動スナップショットの名前付け - 自動スナップショットの名前はパターン rds:<cluster-name>-yyyy-mm-dd-hh-mm に従い、yyyy-mm-dd-hh-mm はスナップショットが作成された日時を表します。
- スケジュールに基づいて自動的に作成 - クラスターを作成または変更するときに、バックアップ保持期間 を 1 ~ 35 日の整数値に設定できます。デフォルトでは、新しいクラスターのバックアップ保持期間は 1 日です。バックアップ保持期間により、自動的に削除される前に自動スナップショットが保持される日数が定義されます。Amazon DocumentDB クラスター上で自動バックアップを無効にすることはできません。

  バックアップ保持期間を設定することに加えて、バックアップウィンドウも設定します。これは、自動スナップショットが作成される 1 日の時間帯です。

- 自動スナップショットの削除 - 自動スナップショットは、自動スナップショットのクラスターを削除すると削除されます。自動スナップショットを手動で削除することはできません。
- 増分 - バックアップ保持期間中にデータベースの更新が記録されるため、変更の増分記録が作成されます。
- 自動スナップショットから復元 - AWS マネジメントコンソール または AWS CLIを使用して自動スナップショットから復元することができます。を使用してスナップショットから復元する場合は AWS CLI、クラスターが使用可能になった後にインスタンスを個別に追加する必要があります。
- 共有 - Amazon DocumentDB自動クラスタースナップショットを共有することはできません。回避策として、自動スナップショットをコピーして手動スナップショットを作成し、そのコピーを共有できます。スナップショットのコピーの詳細については、「Amazon DocumentDB クラスタースナップショットのコピー」を参照してください。スナップショットからクラスターを復元する方法の詳細については、「クラスタースナップショットからの復元」を参照してください。
- バックアップ保持期間内の任意の時点から復元可能 - データベースの更新は増分で記録されるため、バックアップ保持期間内の任意の時点にクラスターを復元できます。

  を使用して自動スナップショットまたはpoint-in-time復元から復元する場合は AWS CLI、クラスターが利用可能になった後にインスタンスを個別に追加する必要があります。

Amazon DocumentDB 手動スナップショットには以下の主な機能があります。

- オンデマンドで作成 — Amazon DocumentDB 手動スナップショットは、Amazon DocumentDB マネジメントコンソールまたは を使用してオンデマンドで作成されます AWS CLI。
- 手動スナップショットの削除 - Amazon DocumentDB コンソールまたは AWS CLIを使用して明示的に削除するときのみ、手動スナップショットは削除されます。クラスターを削除しても、手動スナップショットは削除されません。
- 完全バックアップ - 手動スナップショットが作成されると、クラスターのデータの完全バックアップが作成および保存されます。
- 手動スナップショットの名前付け - 手動スナップショット名を指定します。Amazon DocumentDB では、`datetime` のスタンプを名前に追加しないため、名前に含めたい場合は、その情報を追加する必要があります。
- 手動スナップショットから復元 - コンソールまたは AWS CLIを使用して、手動スナップショットから復元することができます。を使用してスナップショットから復元する場合は AWS CLI、クラスターが利用可能になった後にインスタンスを個別に追加する必要があります。
- Service Quotas — あたり最大 100 個の手動スナップショットに制限されています AWS リージョン。

- 共有 - 手動クラスタースナップショットは共有できます。これは、許可された AWS アカウントによってコピーできます。暗号化されている手動スナップショットまたは暗号化されていない手動スナップショットを共有できます。スナップショットのコピーの詳細については、「Amazon DocumentDB クラスタースナップショットのコピー」を参照してください。
- 手動スナップショットが作成された時点への復元 - 手動スナップショットから復元するときは、手動スナップショットが作成された時点に復元します。

  を使用してスナップショットから復元する場合は AWS CLI、クラスターが使用可能になった後にインスタンスを個別に追加する必要があります。

手動クラスタースナップショットの作成

手動スナップショットは、 AWS マネジメントコンソール または を使用して作成できます AWS CLI。スナップショットの作成にかかる時間は、データベースのサイズによって異なります。スナップショットを作成する場合、以下を実行する必要があります。

1. バックアップするクラスターを特定します。
2. スナップショットに名前を付けます。これにより、後でスナップショットから復元できます。

Using the AWS マネジメントコンソール

を使用して手動スナップショットを作成するには AWS マネジメントコンソール、次のいずれかの方法に従います。

1. 方式 1:

   1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
   2. ナビゲーションペインで、[Snapshots] を選択してください。

      > **Tip**
      >
      > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. [スナップショット] ページで、[作成] を選択します。
4. [クラススナップショットの作成] ページで、次の操作を行います。
   a. クラスター識別子 - クラスターのドロップダウンリストから、スナップショットを作成するクラスターを選択します。
   b. スナップショット識別子 - スナップショットの名前を入力します。

      スナップショットの名前付けの制約:

      - 長さは、[1 ~ 255] 個の英字、数字、またはハイフンです。
      - 1 字目は英字である必要があります。
      - ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。
      - AWS のアカウントごと、リージョンごとに、すべてのクラスター (Amazon RDS、Amazon Neptune、および Amazon DocumentDB に渡って) で一意にする必要があります。

   c. [作成] を選択します。

2. 方式 2:

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. [クラスター] ページで、スナップショットを作成するクラスターの左側にあるボタンを選択します。
4. [アクション] メニューで、[スナップショットの取得] を選択します。
5. [クラススナップショットの作成] ページで、次の操作を行います。
   a. スナップショット識別子 - スナップショットの名前を入力します。

      スナップショットの名前付けの制約:

      - 長さは、[1 ~ 63] 個の英字、数字、またはハイフンです。

- 1 字目は英字である必要があります。
- ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。
- AWS のアカウントごと、リージョンごとに、すべてのクラスター (Amazon RDS、Amazon Neptune、および　Amazon DocumentDB に渡って) で一意にする必要があります。

b. [作成] を選択します。

Using the AWS CLI

を使用してクラススナップショットを作成するには AWS CLI、以下のパラメータを指定して `create-db-cluster-snapshot`オペレーションを使用します。

パラメータ

- **`--db-cluster-identifier`** — 必須。スナップショットを作成するクラスターの名前。このクラスターは存在していて、使用可能 である必要があります。
- **`--db-cluster-snapshot-identifier`** — 必須。作成する手動スナップショットの名前。

次の例では、`sample-cluster` という名前のクラスターの、`sample-cluster-snapshot` という名前のスナップショットを作成します。

Linux、macOS、Unix の場合:

```
aws docdb create-db-cluster-snapshot \
    --db-cluster-identifier sample-cluster \
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

Windows の場合:

```
aws docdb create-db-cluster-snapshot ^
    --db-cluster-identifier sample-cluster ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

このオペレーションによる出力は、次のようになります。

```
{
    "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"
    }
}
```

Amazon DocumentDB クラスタースナップショットのコピー

Amazon DocumentDB では、同じ 内 AWS リージョン または別の にスナップショットをコピーできます AWS リージョン。共有スナップショットを同じ AWS リージョン または別の のアカウントにコピーすることもできます AWS リージョン。スナップショットの共有の詳細については、「Amazon DocumentDB クラスタースナップショットの共有」を参照してください。

> **Note**
>
> Amazon DocumentDB は、保持するバックアップデータおよびスナップショットデータの量と保持期間に基づいて請求を行います。Amazon DocumentDB バックアップおよびスナップショットに伴うストレージのさらなる詳細については、バックアップストレージの使用状況を確認する を参照してください。Amazon DocumentDB ストレージの料金情報については、Amazon DocumentDB の料金 を参照してください。

トピック

- 共有スナップショットのコピー
- 間でのスナップショットのコピー AWS リージョン
- 制限事項
- 暗号化の処理
- パラメータグループに関する考慮事項
- クラススナップショットのコピー

共有スナップショットのコピー

他の AWS アカウントによって共有されたスナップショットをコピーできます。別の AWS アカウントから共有された暗号化されたスナップショットをコピーする場合は、スナップショットの暗号化に使用された AWS KMS 暗号化キーにアクセスできる必要があります。詳細については、「暗号化の処理」を参照してください。

間でのスナップショットのコピー AWS リージョン

ソーススナップショットの AWS リージョン とは異なる にスナップショットをコピーすると AWS リージョン、各コピーは完全なスナップショットになります。フルスナップショットコピーには、Amazon DocumentDB クラスターを復元するために必要なデータやメタデータのすべてが含まれます。

制限事項

スナップショットをコピーする際の制約は以下のとおりです。

- ターゲットスナップショットが使用可能になる前にソーススナップショットを削除すると、スナップショットはコピーされない場合があります。ターゲットスナップショットのステータスが `AVAILABLE` になったことを確認してから、ソーススナップショットを削除してください。
- アカウントあたり 1 つのコピー先リージョンに対して最大 5 つのスナップショットコピーリクエストを実行できます。
- AWS リージョン 関連するデータとコピーするデータの量によっては、クロスリージョンスナップショットのコピーが完了するまでに数時間かかる場合があります。場合によっては、特定のコピー元 AWS リージョンからの多数のクロスリージョンスナップショットコピーのリクエストが発生することがあります。このような場合、Amazon DocumentDB は、進行中のコピーが完了するまで、そのソースからの新しいクロスリージョンコピーリクエスト AWS リージョン をキューに入

れることがあります。コピーリクエストがキューに入っている間は、そのリクエストに関する進捗情報は表示されません。コピーがスタートしたときに、進捗情報は表示されます。

暗号化の処理

AWS KMS 暗号化キーを使用して暗号化されたスナップショットをコピーできます。暗号化されたスナップショットをコピーする場合は、スナップショットのコピーも暗号化する必要があります。同じ 内で暗号化されたスナップショットをコピーする場合 AWS リージョン、元のスナップショットと同じ AWS KMS 暗号化キーを使用してコピーを暗号化することも、別の AWS KMS 暗号化キーを指定することもできます。暗号化されたスナップショットをリージョン間でコピーする場合、キーはリージョン固有であるため、ソーススナップショットと同じ AWS KMS 暗号化 AWS KMS キーをコピーに使用することはできません。代わりに、送信先 n AWS リージョンで有効な AWS KMS キーを指定する必要があります。

ソーススナップショットはコピープロセス全体で暗号化されたままになります。詳細については、「Amazon DocumentDB におけるデータ保護」を参照してください。

> **Note**
>
> Amazon DocumentDB クラスタースナップショットの場合、暗号化されていないクラスタースナップショットをコピー時に暗号化することはできません。

パラメータグループに関する考慮事項

リージョン間でスナップショットをコピーすると、コピーにはコピー元の Amazon DocumentDB クラスターで使用されているパラメータグループは含まれません。スナップショットを復元して新しいクラスターを作成すると、そのクラスター AWS リージョン は作成された のデフォルトのパラメータグループを取得します。新しいクラスターにコピー元と同じパラメータを使用するには、以下の操作を行う必要があります。

1. 送信先で AWS リージョン、元のクラスターと同じ設定の Amazon DocumentDB クラスターパラメータグループを作成します。新しい に既に存在する場合は AWS リージョン、そのものを使用できます。
2. コピー先のスナップショットを復元したら AWS リージョン、新しい Amazon DocumentDB クラスターを変更し、前のステップの新規または既存のパラメータグループを追加します。詳細については、「Amazon DocumentDB クラスターの変更」を参照してください。

クラスタースナップショットのコピー

AWS CLI次のように、 AWS マネジメントコンソール または を使用して Amazon DocumentDB クラスターをコピーできます。

Using the AWS マネジメントコンソール

を使用してクラスタースナップショットのコピーを作成するには AWS マネジメントコンソール、次の手順を実行します。この手順は、同じ またはリージョン間で、暗号化されたクラスタースナップショット AWS リージョン または暗号化されていないクラスタースナップショットをコピーするために機能します。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[スナップショット] を選択し、コピーしたいスナップショットの左にあるボタンを選択します。

   > **Tip**
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. [アクション] メニューから [コピー] を選択します。
4. 結果としての クラスタースナップショットをコピーする のページで、設定 セクションを完了します。

   a. コピー先リージョン - オプション。クラスタースナップショットを別の にコピーするには AWS リージョン、送信先リージョン AWS リージョン のスナップショットを選択します。

   b. 新しいスナップショット識別子 - 新しいスナップショットの名前を入力します。

      ターゲットのスナップショットの名前付けの制約:

      - 既存のスナップショットの名前にすることはできません。
      - 長さは、[1 ~ 63] 個の英字、数字、またはハイフンです。
      - 1 字目は英字である必要があります。

- ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。
- Amazon RDS、Neptune、Amazon DocumentDB のすべてのクラスターで AWS アカウント、リージョンごとに一意である必要があります。

c. タグをコピー - ソーススナップショットに付いているタグをスナップショットコピーにコピーするには、タグをコピー を選択します。

5. [保管時の暗号化] セクションに入力します。

a. 保管時の暗号化 — スナップショットが暗号されていない場合、暗号化されていないスナップショットから暗号化されたコピーは作成できないため、これらのオプションは使用できません。スナップショットが暗号化されている場合は、保管時の暗号化中 AWS KMS key に使用される を変更できます。

スナップショットのコピーの暗号化に関する詳細については、「クラススナップショットの暗号化をコピーする」を参照してください。

保管時の暗号化の詳細については、「Amazon DocumentDB データの保管時の暗号化」を参照してください。

b. AWS KMS キー — ドロップダウンリストから、次のいずれかを選択します。

- (デフォルト) aws/rds — アカウント番号と AWS KMS キー ID は、このオプションの後に表示されます。
- <some-key-name> - キーを作成した場合は、表示され、選択することができます。
- [キー ARN を入力] - [ARN] ボックスに AWS KMS キーの Amazon リソースネーム (ARN) を入力します。ARN の形式は `arn:aws:kms:<region>:<accountID>:key/<key-id>` です。

6. 選択したスナップショットのコピーを作成するには、[スナップショットのコピー] を選択します。または、スナップショットのコピーを作成しない場合は、キャンセルする を選択できます。

Using the AWS CLI

AWS CLIを使って暗号化されていないクラスタースナップショットのコピーを作成するには、以下のパラメータで `copy-db-cluster-snapshot` のオペレーションを使用します。スナップショットを別の にコピーする場合は AWS リージョン、スナップショットのコピー AWS リージョン 先の で コマンドを実行します。

- **--source-db-cluster-snapshot-identifier** — 必須。コピーを作成するクラスタースナップショットの識別子。クラスターのスナップショットが存在し、利用可能な状態である必要があります。スナップショットを別の にコピーする場合、 AWS リージョン または共有クラスタースナップショットをコピーする場合、この識別子はソースクラスタースナップショットの ARN 形式である必要があります。このパラメータは大文字と小文字が区別されません。
- **--target-db-cluster-snapshot-identifier** — 必須。ソースクラスタースナップショットから作成する新しいクラスタースナップショットの識別子。このパラメータは大文字と小文字が区別されません。

  ターゲットのスナップショットの名前付けの制約:

  - 既存のスナップショットの名前にすることはできません。
  - 長さは、[1 ~ 63] 個の英字、数字、またはハイフンです。
  - 1 字目は英字である必要があります。
  - ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。
  - Amazon RDS、Neptune、Amazon DocumentDB のすべてのクラスターで AWS アカウント、リージョンごとに一意である必要があります。
- **--source-region** — スナップショットを別の にコピーする場合は AWS リージョン、 AWS リージョン 暗号化されたクラスタースナップショットのコピー元の を指定します。

  スナップショットを別の AWS リージョン にコピーし、`--source-region` を指定しない場合、代わりに `pre-signed-url` のオプションを指定する必要があります。`pre-signed-url` 値は、 AWS リージョン クラスタースナップショットのコピー元のソースで呼び出すCopyDBClusterSnapshotアクションの署名バージョン 4 の署名付きリクエストを含む URL である必要があります。`pre-signed-url` の詳細については、CopyDBClusterSnapshot を参照してください。
- **--kms-key-id** - クラスタースナップショットのコピーを暗号化するのに使用するキーのための KMS キー識別子。

  暗号化されたクラスタースナップショットを別の にコピーする場合は AWS リージョン、このパラメータが必要です。送信先の KMS キーを指定する必要があります AWS リージョン。

  暗号化されたクラスタースナップショットを同じ にコピーする場合 AWS リージョン、 AWS KMS キーパラメータはオプションです。クラスタースナップショットのコピーは、ソースクラスタースナップショットと同じ AWS KMS キーで暗号化されます。コピーの AWS KMS 暗号化に使用する新しい暗号化キーを指定する場合は、このパラメータを使用して指定できます。
- **--copy-tags** - オプション。コピーするタグと値。

コピーオペレーションを進行中にキャンセルするには、DB クラススナップショットが コピー中 ステータスの間に --target-db-cluster-snapshot-identifier または TargetDBClusterSnapshotIdentifier により識別されるターゲットクラススナップショットを削除できます。

Example

例1: 暗号化されていないスナップショットを同一リージョンにコピーする場合

次の AWS CLI 例では、ソーススナップショット AWS リージョン と同じ sample-cluster-snapshot-copyに sample-cluster-snapshotという名前の のコピーを作成します。コピーが作成されると、元のスナップショットのすべてのタグはコピーされたスナップショットにコピーされます。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

このオペレーションによる出力は、次のようになります。

```
{
    "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

例 2: 暗号化されていないスナップショットを にコピーする AWS リージョン

次の AWS CLI 例ではsample-cluster-snapshot、ARN を持つ のコピーを作成しますarn:aws:rds:us-east-1:123456789012:cluster-snapshot:sample-cluster-snapshot。このコピーは という名前sample-cluster-snapshot-copyで、 コマンドが実行される AWS リージョン にあります。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

このオペレーションによる出力は、次のようになります。

```
{
```

```
    "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

例 3: 暗号化されたスナップショットを にコピーする AWS リージョン

次の AWS CLI 例では、us-west-2 リージョンsample-cluster-snapshotから us-east-1 リージョンに のコピーを作成します。このコマンドは、us-east-1 リージョンで呼び出されます。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

このオペレーションによる出力は、次のようになります。

```
{
    "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

例 4: 暗号化されていない共有スナップショットを にコピーする AWS リージョン

次の AWS CLI 例の account - は123456789012、us-east-1 リージョン999999999999から us-west-2 リージョンに、アカウント によってsample-cluster-snapshot共有される暗号化され

ていないクラススナップショットのコピーを作成します。このコマンドは、us-west-2 リージョンで呼び出されます。スナップショットの共有の詳細については、「スナップショットの共有」を参照してください。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

このオペレーションによる出力は、次のようになります。

```
{
 "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

例 5: 暗号化された共有スナップショットを にコピーする AWS リージョン

次の AWS CLI 例の account - は、us-east-1 リージョン999999999999から us-west-2 リージョンに、アカウントによってsample-cluster-snapshot共有される暗号化されたクラススナップショットのコピー123456789012を作成します。ターゲットスナップショットは、カスタマーマネージド KMS キー - arn:aws:kms:us-west-2:123456789012:key/6c1f3264-1797-472b-ba37-03011e682d28 で暗号化されます。このコマンドは、us-west-2 リージョンで呼び出されます。スナップショットの共有の詳細については、「スナップショットの共有」を参照してください。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

このオペレーションによる出力は、次のようになります。

```
{
 "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**
>
> スナップショットのコピーの暗号化に関する詳細については、「クラススナップショットの暗号化をコピーする」を参照してください。
> 保管時の暗号化の詳細については、「Amazon DocumentDB データの保管時の暗号化」を参照してください。

Amazon DocumentDB クラススナップショットの共有

Amazon DocumentDB を使用すると、次の方法で手動クラススナップショットを共有できます。

- 手動クラススナップショットを共有すると、暗号化されているかどうかにかかわらず、承認された AWS アカウントはスナップショットをコピーできます。
- 手動クラススナップショットを共有すると、暗号化されているかどうかにかかわらず、承認された AWS アカウントは、クラスターのコピーを取得して復元する代わりに、スナップショットからクラスターを直接復元できます。

> **Note**
>
> 自動クラスタースナップショットを共有するには、自動化されたスナップショットをコピーしてそのコピーを共有することで、手動クラスタースナップショットを作成します。このプロセスは、 AWS バックアップによって生成されたリソースにも適用されます。

手動スナップショットは、最大 20 の他の と共有できます AWS アカウント。暗号化されていない手動スナップショットをパブリックとして共有することもできます。これにより、このスナップショットをすべての アカウントが使用できるようになります。スナップショットをパブリックとして共有する場合には、パブリックスナップショットにプライベート情報が含まれないように注意してください。

手動スナップショットを他の と共有し AWS アカウント、 AWS CLI または Amazon DocumentDB API を使用して共有スナップショットからクラスターを復元する場合は、共有スナップショットの Amazon リソースネーム (ARN) をスナップショット識別子として指定する必要があります。

暗号化されたスナップショットの共有

以下の制限は、暗号化されたスナップショットの共有に適用されます。

- 暗号化されたスナップショットをパブリックとして共有することはできません。
- スナップショットを共有したアカウントのデフォルトの AWS KMS 暗号化キーを使用して暗号化されたスナップショットを共有することはできません。

暗号化されたスナップショットを共有するには、以下のステップに従います。

1. スナップショットの暗号化に使用された AWS Key Management Service (AWS KMS) 暗号化キーを、スナップショットにアクセスできるアカウントと共有します。

   AWS KMS 暗号化キーを別の AWS アカウントと共有するには、他のアカウントを AWS KMS キーポリシーに追加します。キーポリシーの更新の詳細については、 AWS Key Management Service デベロッパーガイドの「KMS AWS でのキーポリシーの使用」を参照してください。キーポリシーの作成例については、このトピックで後述する 暗号化されているスナップショットのコピーを許可する IAM ポリシーの作成 を参照してください。
2. 次に示すように AWS CLI、 を使用して、暗号化されたスナップショットを他のアカウントと共有します。

AWS KMS 暗号化キーへのアクセスを許可する

アカウントから共有された暗号化されたスナップショットを別の がコピー AWS アカウント するには、スナップショットを共有するアカウントが、スナップショットを暗号化した AWS KMS キーにアクセスできる必要があります。別のアカウントに AWS KMS キーへのアクセスを許可するには、キーポリシーのキーポリシーを、 AWS KMS キー AWS KMS ポリシーのプリンシパルとして共有しているアカウントの ARN で更新します。次に、`kms:CreateGrant` アクションを許可します。

AWS KMS 暗号化キーへのアクセス権をアカウントに付与した後、暗号化されたスナップショットをコピーするには、そのアカウントが AWS Identity and Access Management (IAM) ユーザーをまだ作成していない場合は作成する必要があります。さらに、そのアカウントは、ユーザーが AWS KMS キーを使用して暗号化されたスナップショットをコピーできるようにする IAM ポリシーをその IAM ユーザーにアタッチする必要があります。アカウントは IAM ユーザーである必要があり、 AWS KMS セキュリティ上の制限によりルート AWS アカウント ID にすることはできません。

次のキーポリシーの例では、ユーザー 123451234512 が AWS KMS 暗号化キーの所有者です。ユーザー 123456789012 がキーの共有先のアカウントです。この更新されたキーポリシーは、アカウントに AWS KMS キーへのアクセスを許可します。これを行うには、ユーザー 123456789012 のルート AWS アカウント ID の ARN をポリシーのプリンシパルとして含め、 `kms:CreateGrant`アクションを許可します。

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}}
        }
    ]
}
```

暗号化されているスナップショットのコピーを許可する IAM ポリシーの作成

外部 AWS アカウント が AWS KMS キーにアクセスできる場合、そのアカウントの所有者は、アカウント用に作成された IAM ユーザーが、その AWS KMS キーで暗号化された暗号化されたスナップショットをコピーすることを許可するポリシーを作成できます。

次の例は、 AWS アカウント 123456789012 の IAM ユーザーにアタッチできるポリシーを示しています。このポリシーにより、IAM ユーザーは、us-west-2 リージョンc989c1dd-a3f2-4a5d-8d96-e793d082ab26の AWS KMS キーで暗号化されたアカウント 123451234512 から共有スナップショットをコピーできます。

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
                }
            }
        }
    ]
}
```

キーポリシーの更新の詳細については、AWS Key Management Service デベロッパーガイド 中の AWS KMSのキーポリシーを使用する を参照してください。

スナップショットの共有

AWS マネジメントコンソール または を使用して、Amazon DocumentDB 手動クラスタースナップショット (または自動スナップショットのコピー) を共有できます AWS CLI。

Using the AWS マネジメントコンソール

を使用してスナップショットを共有するには AWS マネジメントコンソール、次の手順を実行します。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[Snapshots] を選択します。
3. 共有する手動スナップショットを選択します。
4. [アクション] ドロップダウンメニューで、[共有] を選択します。
5. [DB snapshot visibility] で次のいずれかのオプションを選択します。

   - ソースが暗号化されていない場合は、パブリック を選択して、すべての AWS アカウントが手動スナップショットからクラスターを復元できるようにします。または、プライベート を選択して、手動スナップショットからクラスターを復元することを指定した AWS アカウントのみに許可します。

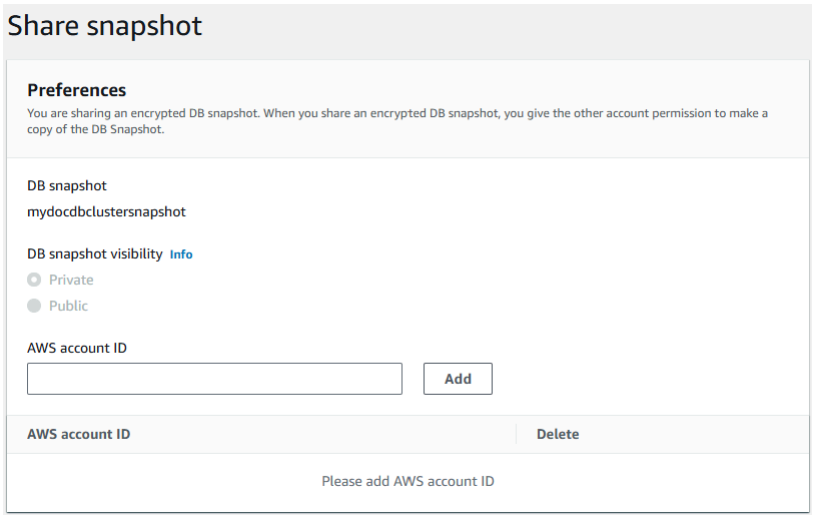
     > **Warning**
     >
     > DB スナップショットの可視性を Public に設定すると、すべての AWS アカウントが手動スナップショットからクラスターを復元し、データにアクセスできます。プライベート情報を含む手動クラススナップショットは、[Public] として共有しないでください。

   - 出典 DB クラスターが暗号化されている場合、暗号化されているスナップショットはパブリックとして共有できないため、[DB snapshot visibility] が [Private] に設定されます。

     > **Note**
     >
     > デフォルトを使用して暗号化されたスナップショットは共有 AWS KMS key できません。

6. AWS アカウント ID に、手動スナップショットからクラスターを復元することを許可するアカウントのアカウント AWS 識別子を入力し、追加を選択します。繰り返して、最大 20 AWS 個のアカウント識別子を追加します AWS 。

   許可された AWS アカウントのリストにアカウント識別子を追加するときにエラーが発生した場合は、間違った AWS アカウント識別子の右側にある削除を選択して、リストから削除できます。



7. 手動スナップショットの復元を許可するすべての AWS アカウントの識別子を追加したら、保存を選択して変更を保存します。

Using the AWS CLI

を使用してスナップショットを共有するには AWS CLI、Amazon DocumentDB `modify-db-snapshot-attribute`オペレーションを使用します。`--values-to-add` パラメータを使用して、手動スナップショットの復元 AWS アカウント が許可されている の IDs のリストを追加します。

次の例では、 123451234512という名前のスナップショットを復元するために、 と 123456789012 の 2 つの AWS アカウント 識別子を許可しますmanual-snapshot1。また、`all` 属性値を削除し、スナップショットをプライベートとしてマークします。

Linux、macOS、Unix の場合:

```
aws docdb modify-db-cluster-snapshot-attribute \
    --db-cluster-snapshot-identifier sample-cluster-snapshot \
    --attribute-name restore \
    --values-to-add '["123451234512","123456789012"]'
```

Windows の場合:

```
aws docdb modify-db-cluster-snapshot-attribute ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot ^
    --attribute-name restore ^
    --values-to-add '["123451234512","123456789012"]'
```

このオペレーションによる出力は、次のようになります。

```
{
    "DBClusterSnapshotAttributesResult": {
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot",
        "DBClusterSnapshotAttributes": [
            {
                "AttributeName": "restore",
                "AttributeValues": [
                    "123451234512",
                    "123456789012"
                ]
            }
        ]
    }
}
```

リストから AWS アカウント 識別子を削除するには、 `--values-to-remove`パラメータを使用します。次の例では、 AWS アカウント ID 123456789012 がスナップショットを復元できないようにします。

Linux、macOS、Unix の場合:

```
aws docdb modify-db-cluster-snapshot-attribute \
    --db-cluster-snapshot-identifier sample-cluster-snapshot \
    --attribute-name restore \
    --values-to-remove '["123456789012"]'
```

Windows の場合:

```
aws docdb modify-db-cluster-snapshot-attribute ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot ^
    --attribute-name restore ^
    --values-to-remove '["123456789012"]'
```

このオペレーションによる出力は、次のようになります。

```
{
    "DBClusterSnapshotAttributesResult": {
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot",
        "DBClusterSnapshotAttributes": [
```

```
            {
                "AttributeName": "restore",
                "AttributeValues": [
                    "123451234512"
                ]
            }
        ]
    }
}
```

クラススナップショットからの復元

Amazon DocumentDB (MongoDB 互換) は、ストレージボリュームのクラススナップショットを作成します。クラススナップショットから復元することで、新しいクラスターを作成できます。クラスターを復元するときは、復元元のクラススナップショットの名前と、復元によって作成される新しいクラスターの名前を指定します。スナップショットから既存のクラスターに復元することはできません。復元時に新しいクラスターが作成されるためです。

クラススナップショットからクラスターを復元する場合:

- このアクションではクラスターのみを復元し、そのクラスターのインスタンスは復元しません。復元されたクラスターのインスタンスを作成するには、`create-db-instance` アクションを呼び出し、復元されたクラスターの識別子を `--db-cluster-identifier` に指定する必要があります。インスタンスを作成できるのは、クラスターが使用可能になった後のみです。
- 暗号化されたスナップショットを暗号化されていないクラスターに復元することはできません。ただし、 AWS KMS キーを指定することで、暗号化されていないスナップショットを暗号化されたクラスターに復元できます。
- 暗号化されたスナップショットからクラスターを復元するには、 AWS KMS キーにアクセスできる必要があります。

> **Note**
>
> 3.6 クラスターを 4.0 クラスターに復元することはできませんが、あるクラスタバージョンから別のクラスタバージョンに移行できます。詳細については、「Amazon DocumentDB の移行とアップグレード」を参照してください。

Using the AWS マネジメントコンソール

次の手順は、Amazon DocumentDB 管理コンソールを使用して Amazon DocumentDB クラスタースナップショットから Amazon DocumentDB クラスターを復元する方法を示しています。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[スナップショット] を選択し、クラスターの復元に使用するスナップショットの左側にあるボタンを選択します。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. [アクション] メニューで、[復元] を選択します。
4. [スナップショットの復元] ページで、[設定] セクションに入力します。

   a. クラスター識別子 - 新しいクラスターの名前。Amazon DocumentDB が指定した名前をそのまま使用するか、希望する名前を入力します。[Amazon DocumentDBsupplied] の名前は、docdb- に UTC タイムスタンプを付加した形式 (例: docdb-*yyyy-mm-dd-hh-mm-ss*)です。

   b. インスタンスクラス - 新しいクラスターのインスタンスクラス。デフォルトのインスタンスクラスをそのまま使用することも、ドロップダウンリストからインスタンスクラスを選択することもできます。

   c. インスタンス数 - このクラスターで作成するインスタンスの数。デフォルトの 3 つのインスタンス (プライマリ読み取り/書き込みレプリカ 1 つ、読み取り専用レプリカ 2 つ) をそのまま使用するか、ドロップダウンリストからインスタンス数を選択できます。

5. [クラスターストレージ設定]では、ストレージオプションを選択します。

   > **Note**
   >
   > Amazon DocumentDB I/O 最適化ストレージ設定は、Amazon DocumentDB 5.0 および 8.0 エンジンバージョンでのみ使用できます。

6. クラスター設定に問題がなければ、[クラスターの復元] を選択し、クラスターが復元されるのを待ちます。
7. デフォルトでない Amazon VPC やセキュリティグループの指定など一部の設定を変更する場合は、ページの左下にある [詳細設定の表示] を選択し、以下のステップに進みます。

   a. [ネットワーク設定] セクションに入力します。

      - 仮想プライベートクラウド（VPC）- 現在の VPC を受け入れるか、ドロップダウンリストから VPC を選択します。
      - サブネットグループ - `default` サブネットグループを使用するか、ドロップダウンリストからいずれかのサブネットグループを選択します。
      - VPC セキュリティグループ - `default (VPC)` セキュリティグループを使用するか、リストからいずれのセキュリティグループを選択します。

   b. [クラスターオプション] セクションに入力します。

      - データベースポート - デフォルトのポート `27017` を使用するか、上下の矢印を使用してアプリケーション接続に使用するポートを設定します。

   c. [暗号化] セクションに入力します。

      - 保管時の暗号化 - スナップショットが暗号化されている場合、ここでのオプションは使用できません。暗号化されていない場合は、以下のいずれかを選択できます。
        - すべてのクラスターのデータを暗号化するには、[保管時の暗号化の有効化] を選択します。このオプションを選択した場合は、KMS キーを指定する必要があります。
        - クラスターのデータを暗号化しない場合は、[保管時の暗号化の無効化] を選択します。このオプションを選択した場合、暗号化セクションは完了です。
      - AWS KMS キー — ドロップダウンリストから次のいずれかを選択します。
        - (デフォルト) aws/rds — アカウント番号と AWS KMS キー ID は、このオプションの後に表示されます。
        - カスタマーマネージドキー — このオプションは、 AWS Identity and Access Management (IAM) コンソールで IAM 暗号化キーを作成した場合にのみ使用できます。クラスターを暗号化するためのキーを選択できます。
        - キー ARN の入力 — ARN ボックスに、 AWS KMS キーの Amazon リソースネーム (ARN) を入力します。ARN の形式は `arn:aws:kms:<region>:<accountID>:key/<key-id>` です。

   d. [ログのエクスポート] セクションに入力します。

- CloudWatch に公開するログの種類を選択する - 以下のいずれかを選択します。
  - 有効 — クラスターで DDL ログ記録を Amazon CloudWatch Logs にエクスポートできるようにします。
  - 無効 — クラスターで DDL ログ記録を Amazon CloudWatch Logs にエクスポートできないようにします。デフォルトは [無効] です。
- IAM ロール - リストから [RDS サービスにリンクされたロール] を選択します。

e. [タグ] セクションに入力します。

- タグの追加 - キー ボックスに、クラスターのためのタグの名前を入力します。値 ボックスに、必要に応じてタグ値を入力します。タグは、Amazon DocumentDB リソースへのアクセスを管理し、リソースに適用できるアクションを制御するために AWS Identity and Access Management (IAM) ポリシーとともに使用されます。

f. [削除保護] セクションに入力します。

- 削除保護を有効にする - クラスターが誤って削除されないように保護します。このオプションが有効になっていると、クラスターを削除できません。

8. [クラスターの復元] を選択します。

Using the AWS CLI

を使用してスナップショットからクラスターを復元するには AWS CLI、次のパラメータを指定して `restore-db-cluster-from-snapshot`オペレーションを使用します。詳細については、「RestoreDBClusterFromSnapshot」を参照してください。

- **`--db-cluster-identifier`** — 必須。オペレーションによって作成されるクラスターの名前。この名前のクラスターは、このオペレーションの前に存在することはできません。

  クラスターの名前付けの制約:

  - 長さは [1 〜 63] 個の英字、数字、またはハイフンです。
  - 1 字目は英字である必要があります。
  - ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。
  - Amazon RDS、Neptune、Amazon DocumentDB のすべてのクラスターで AWS アカウント、リージョンごとに一意である必要があります。
- **`--snapshot-identifier`** — 必須。復元前に使用されたスナップショットの名前。この名前のスナップショットが存在し、利用可能 状態である必要があります。

- **--engine** — 必須。docdb を指定してください。
- **--storage-type standard | iopt1** - オプション。デフォルト: standard。
- **--kms-key-id** - オプション。暗号化されたスナップショットを復元するとき、または暗号化されていないスナップショットから復元するときにクラスターを暗号化するときに使用する AWS KMS キー識別子の ARN。キー ID を指定すると AWS KMS 、スナップショットが暗号化されているかどうかにかかわらず、復元されたクラスターが AWS KMS キーで暗号化されます。

  --kms-key-id の形式は arn:aws:kms:<region>:<accountID>:key/<key-id> です。--kms-key-id パラメータの値を指定しない場合は、以下のようになります。

  - のスナップショット--snapshot-identifierが暗号化されている場合、復元されたクラスターは、スナップショットの暗号化に使用されたのと同じ AWS KMS キーを使用して暗号化されます。
  - --snapshot-identifier のスナップショットが暗号化されていない場合、復元されたクラスターは暗号化されません。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

このオペレーションによる出力は、次のようになります。

```
{
    "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
    }
}
```

クラスターのステータスが [使用可能] になったら、そのクラスターに 1 つ以上のインスタンスを作成します。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

このオペレーションによる出力は、次のようになります。

```
{
    "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"
    }
}
```

特定時点への復元

AWS マネジメントコンソール または AWS Command Line Interface () を使用して、クラスターをクラスターのバックアップ保持期間内の任意の時点に復元できますAWS CLI。

> **Note**
>
> 3.6 クラスターの 4.0 クラスターへの特定の時点への復元は実行できませんが、あるクラスターバージョンから別のクラスタバージョンに移行できます。詳細については、「Amazon DocumentDB の移行とアップグレード」を参照してください。

クラスターを特定時点に復元するときには、以下の点を考慮します。

- 新しいクラスターは、デフォルトのパラメータグループを使用して作成されることを除いて、ソースクラスターと同じ設定で作成されます。新しいクラスターのパラメータグループをソースクラスターのパラメータグループに設定するには、ステータスが [使用可能] になってからクラスターを変更します。クラスターの変更の詳細については、「Amazon DocumentDB クラスターの変更」を参照してください。

Using the AWS マネジメントコンソール

AWS マネジメントコンソールを使用して以下を完了すると、クラスターをバックアップ保持期間内の特定の時点に復元できます。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択してください。クラスターのリストで、復元するクラスターの左側にあるボタンを選択します。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. [アクション] メニューで、[特定時点への復元] を選択します。
4. [復元時刻] セクションに情報を入力して、復元する日時を指定します。

   a. 復元日 - [最新復元時刻] から [最後の復元時刻] の範囲の日付を選択または入力します。

   b. 復元時刻 - [最新復元時刻] から [最後の復元時刻] の範囲の日付で、時分秒を選択または入力します。

5. [設定] セクションに情報を入力します。

   a. クラスター識別子 - デフォルトの識別子を使用するか、任意の識別子を入力します。

      クラスターの名前付けの制約:

      - 長さは [1 ～ 63] 個の英字、数字、またはハイフンです。
      - 1 字目は英字である必要があります。
      - ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。

- Amazon RDS、Neptune、Amazon DocumentDB のすべてのクラスターで AWS アカウント、リージョンごとに一意である必要があります。

b. インスタンスクラス - ドロップダウンリストで、クラスターのインスタンスに使用するインスタンスクラスを選択します。

c. インスタンス数 - ドロップダウンリストで、クラスター復元時に作成するインスタンスの数を選択します。

6. [クラスターストレージ設定]では、ストレージオプションを選択します。

> **Note**
>
> Amazon DocumentDB I/O 最適化ストレージ設定は、Amazon DocumentDB 5.0 および 8.0 エンジンバージョンでのみ使用できます。

7. オプション。ネットワーク設定とクラスターオプションを設定し、ログのエクスポートを有効にするには、[詳細設定の表示] を選択し、次のセクションに入力します。それ以外の場合は、次のステップに進みます。

- ネットワーク設定

  1. 仮想プライベートクラウド（VPC）- ドロップダウンリストから、このクラスターで使用したい VPC を選択します。
  2. サブネットグループ - ドロップダウンリストから、このクラスターのサブネットグループを選択します。
  3. VPC セキュリティグループ - ドロップダウンリストから、このクラスターの VPC セキュリティグループを選択します。

- クラスターオプション

  1. ポート - デフォルトのポート (27017) を使用するか、上向き矢印と下向き矢印を使用して、このクラスターと通信するポートを設定します。

- ログのエクスポート

  1. 監査ログ - このオプションを選択し、Amazon CloudWatch Logs への監査ログを有効にします。このオプションを選択した場合は、クラスターのカスタムパラメータグループで `audit_logs` を有効にする必要があります。詳細については、「Amazon DocumentDB イベントの監査」を参照してください。

2. プロファイラーログ - オペレーションのプロファイラーログの Amazon CloudWatch Logs へのエクスポートを有効にするには、このオプションを選択します。このオプションを選択した場合は、クラスターのカスタムパラメータグループの次のパラメータも変更する必要があります。

   - `profiler` — `enabled` に設定します。
   - `profiler_threshold_ms` - `[0-INT_MAX]` の値に設定して、プロファイリングオペレーションのしきい値を設定します。
   - `profiler_sampling_rate` - プロファイルする低速オペレーションの割合を設定するには、`[0.0-1.0]` の値に設定します。

   詳細については、「Amazon DocumentDB オペレーションのプロファイリング」を参照してください。

3. プロファイラーログ - プロファイラーログを Amazon CloudWatch にエクスポートします

4. IAM ロール - ドロップダウンリストから RDS サービスにリンクされたロール を選択します。

- タグ

  1. タグの追加 - キー ボックスに、クラスターのためのタグの名前を入力します。値 ボックスに、必要に応じてタグ値を入力します。タグは、 AWS Identity and Access Management （ IAM ）のポリシーで使用され、Amazon DocumentDB リソースへのアクセスを管理したり、 リソースに適用できるアクションを制御したりできます。

- 削除保護

  1. 削除保護を有効にする - クラスターが誤って削除されないように保護します。このオプションが有効になっていると、クラスターを削除できません。

8. クラスターを復元するには、[クラスターの作成] を選択します。または、[キャンセルする] を選択してオペレーションをキャンセルすることもできます。

Using the AWS CLI

スナップショットのバックアップ保持期間を使用して特定時点にクラスターを復元するには、以下のパラメータを指定して `restore-db-cluster-to-point-in-time` オペレーションを使用します。

- **--db-cluster-identifier** - 必須。作成する新しいクラスターの名前。このクラスターは、オペレーションの前に存在することはできません。パラメータ値は以下の制約を満たす必要があります。

  クラスターの名前付けの制約:

  - 長さは [1 ～ 63] 個の英字、数字、またはハイフンです。
  - 1 字目は英字である必要があります。
  - ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。
  - Amazon RDS、Neptune、Amazon DocumentDB のすべてのクラスターで AWS アカウント、リージョンごとに一意である必要があります。
- **--restore-to-time** - クラスターを復元する UTC 日時。例えば、`2018-06-07T23:45:00Z`。

  時間の制約:

  - クラスターの復元可能な最新時刻より前である必要があります。
  - `--use-latest-restorable-time` パラメータを指定されていない場合は、指定する必要があります。
  - `--use-latest-restorable-time` パラメータが `true` である場合は、指定することはできません。
  - `--restore-type` パラメータ値が `copy-on-write` である場合は、指定することはできません。
- **--source-db-cluster-identifier** - 復元するソースクラスターの名前。このクラスターは存在していて、使用可能 である必要があります。
- **--use-latest-restorable-time** または **--no-use-latest-restorable-time** - 最新の復元可能なバックアップ時間に復元するかどうか。`--restore-to-time` パラメータを使用する場合は、指定することができません。
- **--storage-type standard | iopt1** - オプション。デフォルト: `standard`。

AWS CLI オペレーションはクラスターのみを`restore-db-cluster-to-point-in-time`復元し、そのクラスターのインスタンスは復元しません。復元されたクラスターのインスタンスを作成するには、`create-db-instance` オペレーションを呼び出し、復元されたクラスターの識別子を `--db-cluster-identifier` に指定する必要があります。インスタンスは、`restore-db-cluster-to-point-in-time` オペレーションを完了し、復元したクラスターが 使用可能になってから作成できます。

Example

次の例では、スナップショット sample-cluster-snapshot から復元可能な最新時刻に sample-cluster-restored を作成します。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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

次の例では、スナップショット sample-cluster-snapshot から、2018 年 12 月 11 日 03:15 (UTC) への sample-cluster-restored を作成します。これは、 sample-cluster のバックアップ保持期間範囲内です。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

このオペレーションによる出力は、次のようになります。

```
{
```

```
    "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
    }
}
```

クラススナップショットの削除

手動スナップショットはフルバックアップであり、 AWS マネジメントコンソール または を使用して手動で削除した場合にのみ削除されます AWS CLI。自動スナップショットを手動で削除することはできません。自動スナップショットが削除されるのは、スナップショットの保持期間が終了したときか、スナップショットのクラスターを削除したときのみです。

Using the AWS マネジメントコンソール

を使用して手動クラススナップショットを削除するには AWS マネジメントコンソール、次の手順を実行します。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[Snapshots] を選択してください。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. スナップショットのリストで、削除するスナップショットの左側にあるボタンを選択します。スナップショットのタイプは [手動] であることが必要です。

   1. スナップショットのタイプが、[タイプ] 列の下で `manual` または `automatic` としてリストされているかどうかチェックすることによって、手動 であることを確認できます。
4. [アクション] メニューから、[削除] を選択します。[削除] オプションが使用できない場合、自動スナップショットを選択した可能性があります。
5. 削除の確認画面でスナップショットを削除するには、[削除] を選択します。スナップショットを保持するには、[キャンセルする] を選択します。

Using the AWS CLI

Amazon DocumentDB の手動クラススナップショットは、 AWS CLIを使用して手動で削除することができる完全なバックアップです。自動スナップショットを手動で削除することはできません。

を使用して手動クラスタースナップショットを削除するには AWS CLI、以下のパラメータを指定して delete-db-cluster-snapshotオペレーションを使用します。

パラメータ

- **--db-cluster-snapshot-identifier** — 必須。削除する手動スナップショットの名前。

以下の例では、クラスタースナップショット sample-cluster-snapshot を削除します。

Linux、macOS、Unix の場合:

```
aws docdb delete-db-cluster-snapshot \
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

Windows の場合:

```
aws docdb delete-db-cluster-snapshot ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

このオペレーションからの出力には、削除したクラスタースナップショットの詳細が一覧表示されます。

Amazon DocumentDB のモニタリングとログ記録

AWS サービスのモニタリングは、システムを正常かつ最適に機能させる上で重要な部分です。AWS ソリューションのすべての部分からモニタリングデータを収集して、障害やパフォーマンスの問題をより簡単にデバッグおよび修正できるようにします。ソリューションのモニタリングを開始する前に、次の質問を考慮してください AWS 。

- どのような目的でモニタリングしますか?
- モニタリングの対象となるリソースは?
- これらのリソースをモニタリングする頻度は?
- どのモニタリングツールを利用しますか?
- モニタリングを実行するための責任者は?
- 問題が発生した場合に、だれにどのような方法で通知しますか?

現在のパフォーマンスパターンを理解するには、パフォーマンスの異常を識別し、問題に対処する方法を策定します。さまざまな時間帯に、さまざまなロード条件下でパフォーマンスメトリクスのベースラインを確立する必要があります。ソリューションをモニタリング AWS するときは、将来の参照とベースラインの確立のために、履歴モニタリングデータを保存します。

一般的に、パフォーマンスメトリクスの許容値は、ベースラインがどのようになっているか、アプリケーションによって何が実行されているかによって異なります。ベースラインからの一貫した差異またはトレンドになっている差異を調べます。メトリクスのタイプごとのアドバイスは以下のとおりです。

- CPU または RAM の大量使用 - CPU または RAM の値が高くても、アプリケーションの目標 (スループットや同時実行数など) に沿った想定内の値であれば、それは適切であると見なされます。
- ストレージボリュームの消費量 - 使用領域が一貫してストレージボリューム領域全体の 85% 以上である場合は、ストレージの消費量 (`VolumeBytesUsed`) を調査します。ストレージボリュームからデータを削除するか、別のシステムにデータをアーカイブすることで、領域を解放できないかどうかを判断します。詳細については、「Amazon DocumentDB のストレージ」および「Amazon DocumentDB クォータ」を参照してください。
- ネットワークトラフィック - ネットワークトラフィックについてシステム管理者に問い合わせて、ドメインネットワークとインターネット接続に対する想定スループットを把握します。スループットが一貫して想定よりも低い場合は、ネットワークトラフィックを調べます。

- データベース接続数 - ユーザー接続数が多いことが、インスタンスのパフォーマンスが下がっていること、応答時間が長くなっていることに関連しているとわかった場合、データベース接続数を制限することを検討します。インスタンスの最適なユーザー接続数は、インスタンスのクラスと実行中のオペレーションの複雑さによって異なります。
- IOPS メトリクス - IOPS メトリクスの想定値はディスクの仕様とサーバーの設定によって異なるため、ベースラインを使用して一般的な値を把握します。値とベースラインとの差が一貫しているかどうかを調べます。最適な IOPS パフォーマンスを得るには、読み取りおよび書き込みオペレーションが最小限になるように、一般的な作業セットがメモリに収まることを確認してください。

Amazon DocumentDB (MongoDB 互換) は、Amazon DocumentDB クラスターとインスタンスの健全性とパフォーマンスを判断するためにモニタリングできる、さまざまな Amazon CloudWatch メトリクスを提供します。Amazon DocumentDB コンソール、CloudWatch API、Performance Insights など AWS CLI、さまざまなツールを使用して Amazon DocumentDB メトリクスを表示できます。

トピック

- Amazon DocumentDB クラスターのステータスのモニタリング
- Amazon DocumentDB インスタンスのステータスのモニタリング
- Amazon DocumentDB に関する推奨事項の表示
- Amazon DocumentDB イベントサブスクリプションを使用する
- Amazon DocumentDB と CloudWatch のモニタリング
- を使用した Amazon DocumentDB API コールのログ記録 AWS CloudTrail
- Amazon DocumentDB オペレーションのプロファイリング
- Performance Insights を使用したモニタリング

Amazon DocumentDB クラスターのステータスのモニタリング

クラスターのステータスは、クラスターの状態を示します。Amazon DocumentDB コンソールまたは AWS CLI `describe-db-clusters` コマンドを使用して、クラスターのステータスを表示できます。

トピック

- クラスターステータスの値
- クラスターのステータスのモニタリング

クラスターステータスの値

次の表は、クラスターのステータスの有効な値の一覧です。

クラスターステータス	説明
active	クラスターはアクティブです。このステータスは Elastic クラスターにのみ適用されます。
available	クラスターは正常で、使用可能です。このステータスはインスタンスベースのクラスターにのみ適用されます。
backing-up	クラスターは現在バックアップ中です。
creating	クラスターを作成しています。作成中はクラスターにアクセスできません。
deleting	クラスターを削除しています。削除中はクラスターにアクセスできません。
failing-over	プライマリインスタンスから Amazon DocumentDB レプリカへのフェイルオーバーが実行されています。
inaccessible-encryption-credentials	クラスターの暗号化または復号に使用される AWS KMS キーにはアクセスできません。
maintenance	クラスターにメンテナンス更新を適用しています。このステータスは、Amazon DocumentDB が事前に十分スケジュールした

クラスターステータス	説明
	クラスターレベルのメンテナンスに使用されます。
migrating	クラスタースナップショットは、クラスターに復元されています。
migration-failed	移行に失敗しました。
modifying	お客様がクラスターの変更をリクエストしたため、クラスターは変更中です。
renaming	お客様が名前の変更をリクエストしたため、クラスターの名前を変更中です。
resetting-master-credentials	お客様がリセットをリクエストしたため、クラスターのマスター認証情報をリセット中です。
upgrading	クラスターエンジンのバージョンをアップグレード中です。

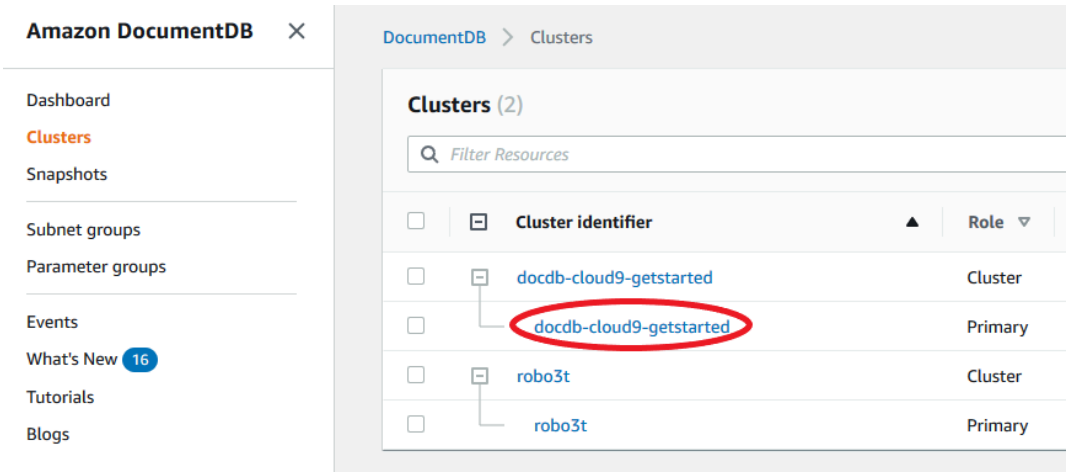
クラスターのステータスのモニタリング

Using the AWS マネジメントコンソール

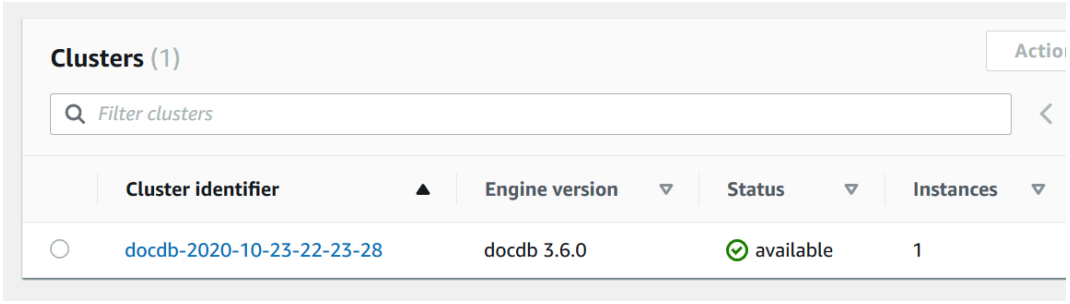
を使用してクラスターのステータス AWS マネジメントコンソール を判断する場合は、次の手順を使用します。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [クラスター] を選択します。

3. Clusters ナビゲーションボックスで、[クラスター識別子] という列が表示されます。インスタンスは、次のスクリーンショットのようにクラスターの下に一覧表示されます。



4. [クラスター識別子] 列で、興味のあるインスタンスの名前を探します。次に、インスタンスのステータスを見つけるために、次に示すように行全体を参照して [ステータス] 列を見つけます。



Using the AWS CLI

を使用してクラスターのステータス AWS CLI を判断する場合は、 `describe-db-clusters`オペレーションを使用します。次のコードでは、クラスター `sample-cluster` のステータスを確認します。

Linux、macOS、Unix の場合:

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

Windows の場合:

```
aws docdb describe-db-clusters ^
```

```
    --db-cluster-identifier sample-cluster  ^
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

このオペレーションによる出力は、次のようになります。

```
[
    [
        "sample-cluster",
        "available"
    ]
]
```

Amazon DocumentDB インスタンスのステータスのモニタリング

Amazon DocumentDB は、データベースに設定されている各インスタンスの現在の状態に関する情報を提供します。

Amazon DocumentDB インスタンスについて表示できるステータスには 3 つのタイプがあります。

- インスタンスのステータス: このステータスは、 のクラスターテーブルのステータス列に表示され AWS マネジメントコンソール 、インスタンスの現在のライフサイクル条件が表示されます。[ステータス] 列に表示される値は `DescribeDBCluster` API レスポンスの `Status` フィールドから取得されます。
- インスタンスのヘルスステータス: このステータスは、 のクラスターテーブルのインスタンスのヘルス列に表示され、データの管理と取得を担当するコンポーネントであるデータベースエンジンが実行されているかどうか AWS マネジメントコンソール を示します。[インスタンスヘルス] 列に表示される値は、Amazon CloudWatch `EngineUptime` システムメトリックスに基づいています。
- メンテナンスステータス: このステータスは、 のクラスターテーブルのメンテナンス列に表示され AWS マネジメントコンソール 、インスタンスに適用する必要があるメンテナンスイベントのステータスを示します。メンテナンスステータスは他のインスタンスのステータスとは独立して、`PendingMaintenanceAction` API から取得されます。メンテナンスのステータスの詳細については、「Amazon DocumentDB のメンテナンス」を参照してください。

トピック

- インスタンスのステータス値
- AWS マネジメントコンソール または を使用したインスタンスのステータスのモニタリング AWS CLI

- インスタンスのヘルスステータス値
- を使用したインスタンスのヘルスステータスのモニタリング AWS マネジメントコンソール

インスタンスのステータス値

次の表は、インスタンスのステータス値と、ステータス別の請求方法の一覧です。請求対象がインスタンスとストレージであるか、ストレージのみであるか、または請求なしであるかを示します。インスタンスのすべてのステータスで、バックアップの使用は常に請求されます。

インスタンスの状態	請求対象	説明
available	請求対象	インスタンスは正常で、使用可能です。
backing-up	請求対象	インスタンスは現在バックアップ中です。
configuring-log-exports	請求対象	Amazon CloudWatch Logs へのログファイルの公開は、このインスタンスに対して有効または無効になっています。
creating	非請求対象	インスタンスは現在作成中です。作成中はインスタンスにアクセスできません。
deleting	非請求対象	インスタンスを削除中です。
failed	非請求対象	インスタンスに障害が発生し、Amazon DocumentDB は復旧できませんでした。データを復旧するには、インスタンスの復元可能な直近の時間までポイントインタイムリカバリを実行します。
inaccessible-encryption-credentials	非請求対象	インスタンスの暗号化または復号に使用される AWS KMS キーにアクセスできませんでした。
incompatible-network	非請求対象	Amazon DocumentDB はインスタンスに対してリカバリーアクションを実行しようとしていますが、VPC がアクションの完了を妨げている状態であるため、実行すること

インスタンスの状態	請求対象	説明
		ができません。この状態は、たとえば、サブネット内の利用可能なすべての IP アドレスが使用中で、Amazon DocumentDB がインスタンスの IP アドレスを取得できない場合に発生する可能性があります。
maintenance	請求対象	Amazon DocumentDB は、インスタンスにメンテナンス更新を適用しています。このステータスは、Amazon DocumentDB が事前に十分なスケジュールを組んでいるインスタンスレベルのメンテナンスに使用されます。このステータスを通じてお客様に追加のメンテナンスアクションを提示する方法を評価中です。
modifying	請求対象	インスタンスの変更をリクエストしたため、インスタンスを変更中です。
rebooting	請求対象	Amazon DocumentDB は、インスタンスの再起動を必要とするリクエスト、または Amazon DocumentDB プロセスのために、インスタンスを再起動しています。
renaming	請求対象	名前の変更をリクエストしたため、インスタンスの名前を変更中です。
resetting-master-credentials	請求対象	リセットをリクエストしたため、インスタンスのマスター認証情報をリセット中です。
restore-error	請求対象	特定時点への復元またはスナップショットからの復元を実行する際に、インスタンスでエラーが発生しました。
starting	ストレージが請求対象	インスタンスを起動中です。
stopped	ストレージが請求対象	インスタンスが停止されました。

インスタンスの状態	請求対象	説明
stopping	ストレージが請求対象	インスタンスを停止中です。
storage-full	請求対象	インスタンスが、ストレージ容量の割り当て分に到達しました。これは非常に重要なステータスであり、すぐに修正する必要があります。インスタンスを変更して、ストレージをスケールアップしてください。このような状況にならないように、ストレージスペースが少なくなってきたときに警告するよう Amazon CloudWatch のアラームを設定します。

AWS マネジメントコンソール または を使用したインスタンスのステータスのモニタリング AWS CLI

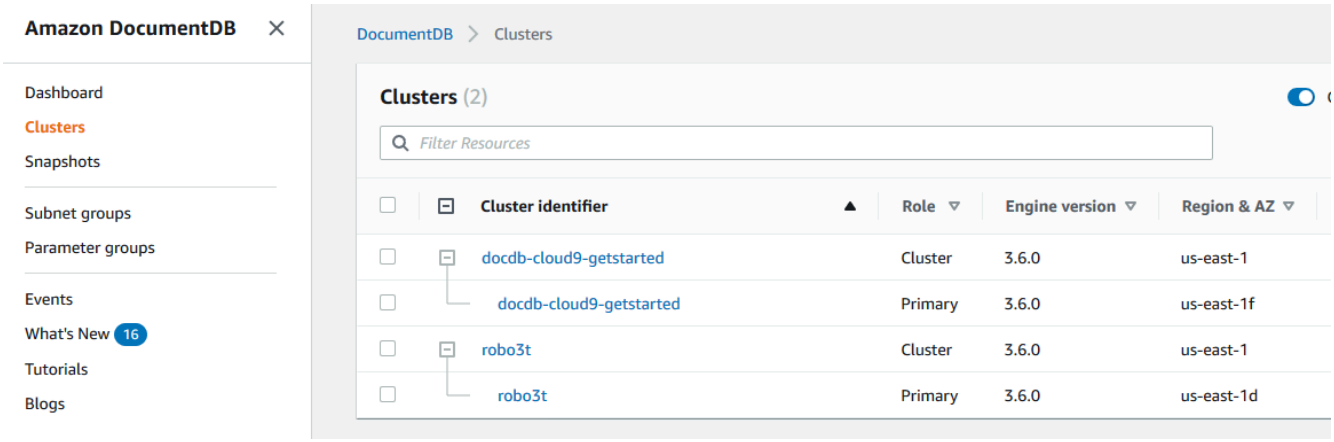
AWS マネジメントコンソール または を使用して AWS CLI 、インスタンスのステータスをモニタリングします。

Using the AWS マネジメントコンソール

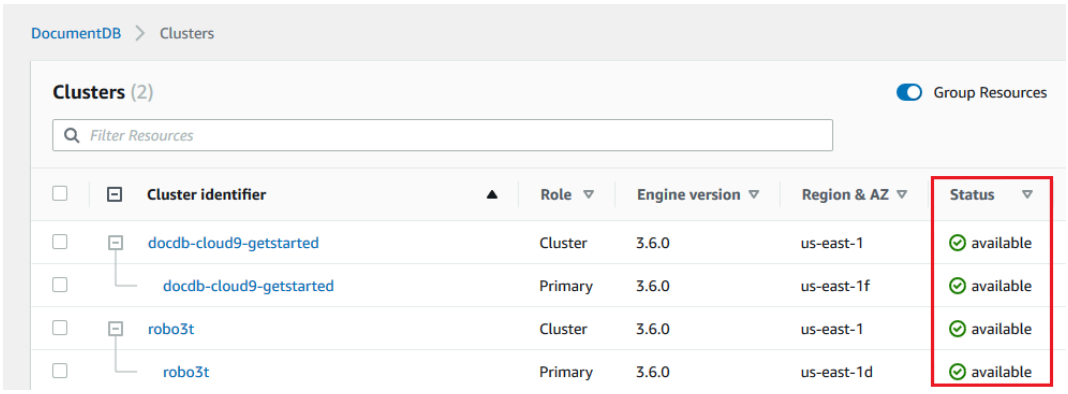
を使用してクラスターのステータス AWS マネジメントコンソール を判断する場合は、次の手順を使用します。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択してください。

> **Note**
>
> Clusters ナビゲーションボックスで、[クラスター識別子] 列にはクラスターとインスタンスの両方が表示されることに注意してください。インスタンスは、次の図のようにクラスターの下に表示されます。



3. 興味のあるインスタンスの名前を探します。次に、インスタンスのステータスを見つけるには、次に示すように [ステータス] 列まで行全体を読みます。



Using the AWS CLI

を使用してクラスターのステータス AWS CLI を判断する場合は、 `describe-db-instances`オペレーションを使用します。次のコードでは、インスタンス `sample-cluster-instance-01` のステータスを確認します。

Linux、macOS、Unix の場合:

```
aws docdb describe-db-instances \
          --db-instance-identifier sample-cluster-instance-01  \
          --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceStatus]'
```

Windows の場合:

```
aws docdb describe-db-instances ^
```

```
        --db-instance-identifier sample-cluster-instance-01  ^
        --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceStatus]'
```

このオペレーションによる出力は、次のようになります。

```
[
        [
            "sample-cluster-instance-01",
            "available"
        ]
    ]
```

インスタンスのヘルスステータス値

DB インスタンスに可能なステータス値を以下の表に示します。のクラスターテーブルにあるインスタンスヘルス列には AWS マネジメントコンソール、データの保存、管理、取得を担当するコンポーネントであるデータベースエンジンが正常に動作しているかどうかが表示されます。この列には、CloudWatch で使用できる `EngineUptime` システムメトリックスに各インスタンスのヘルスステータスが表示されているかどうかも表示されます。

インスタンスのヘルスステータス	説明
正常	データベースエンジンは Amazon DocumentDB インスタンスで実行されています。
異常	データベースエンジンが実行されていないか、1 分以内に再起動しました。

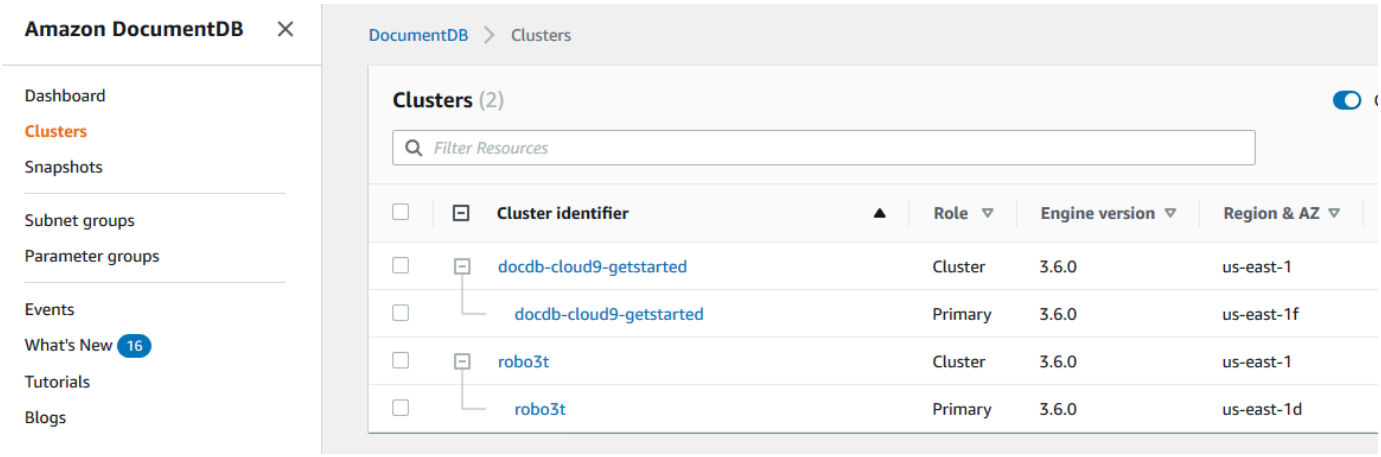
を使用したインスタンスのヘルスステータスのモニタリング AWS マネジメントコンソール

AWS マネジメントコンソール を使用して、インスタンスのヘルスステータスをモニタリングします。

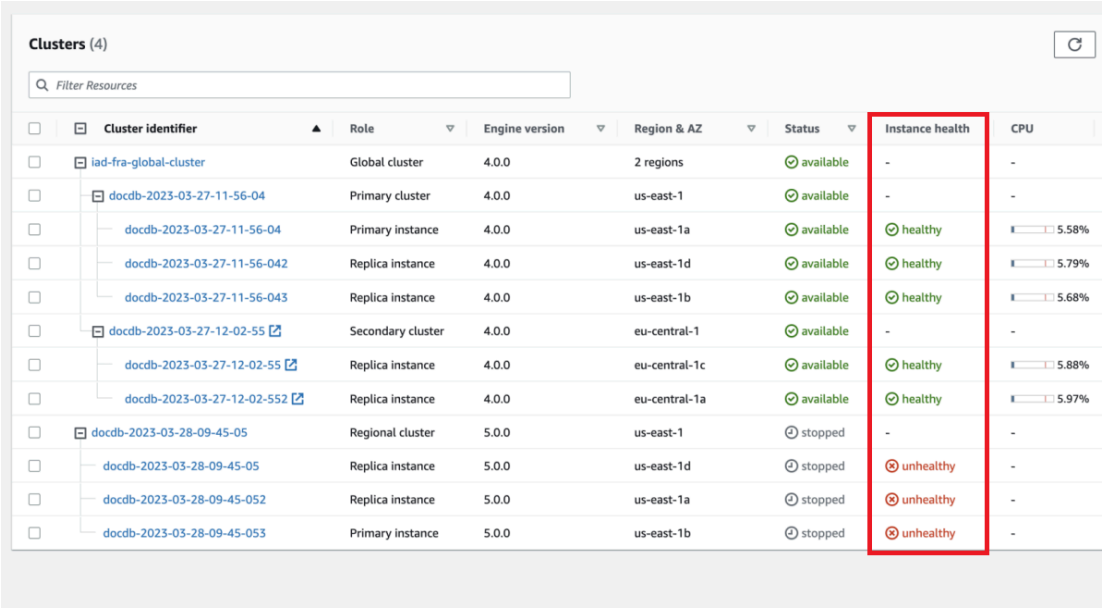
を使用する場合は AWS マネジメントコンソール、次の手順を使用してインスタンスのヘルスステータスを理解します。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択してください。

> **Note**
>
> [クラスター] ナビゲーションボックスの [クラスター識別子] 列にはクラスターとインスタンスの両方が表示されます。インスタンスは、次の図のようにクラスターの下に表示されます。



3. 興味のあるインスタンスの名前を探します。次に、インスタンスのステータスを見つけるには、次に示すように [インスタンスヘルス] 列まで行全体を読みます。



> **Note**
>
> インスタンスヘルスステータスのポーリングは 60 秒ごとに発生し、CloudWatch `EngineUptime` システムメトリックスに基づいています。[インスタンスヘルス] 列の値は自動的に更新されます。

Amazon DocumentDB に関する推奨事項の表示

Amazon DocumentDB では、インスタンスやクラスターなどのデータベースリソースについての推奨事項が自動で表示されます。これらの推奨事項では、クラスターとインスタンスの設定を分析して、ベストプラクティスガイダンスを提供します。

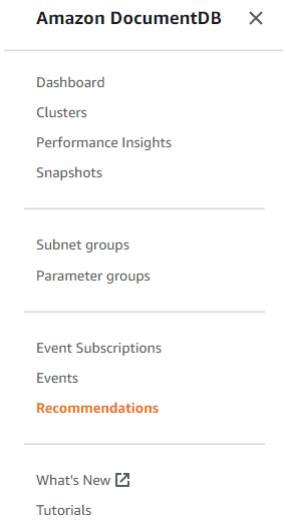
これらの推奨事項の例については、以下を参照してください。

型	説明	推奨事項	追加情報
1 個の インスタンス	クラスターにはインスタンスが 1 つだけ含まれています。	パフォーマンスと可用性: 別のアベイラビリティーゾーンに同じインスタンスクラスを持つ別のインスタンスを追加します。	Amazon DocumentDB の高可用性とレプリケーション

Amazon DocumentDB は、リソースが作成または変更されると、リソースの推奨事項を生成します。また、Amazon DocumentDBは定期的にリソースをスキャンして、推奨事項を生成します。

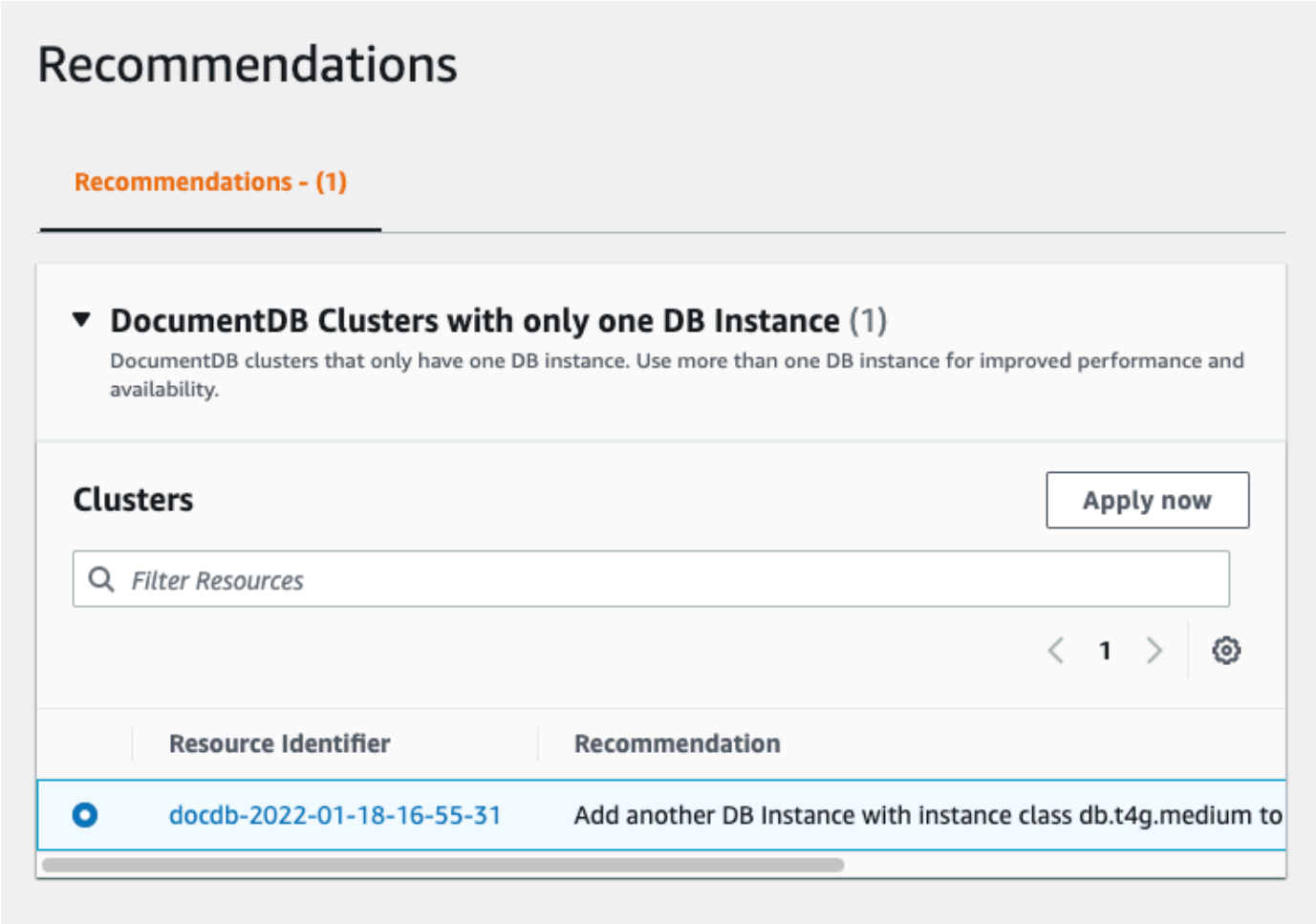
Amazon DocumentDB の推奨事項を確認してアクションを実行するには

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[推奨事項] を選択します。



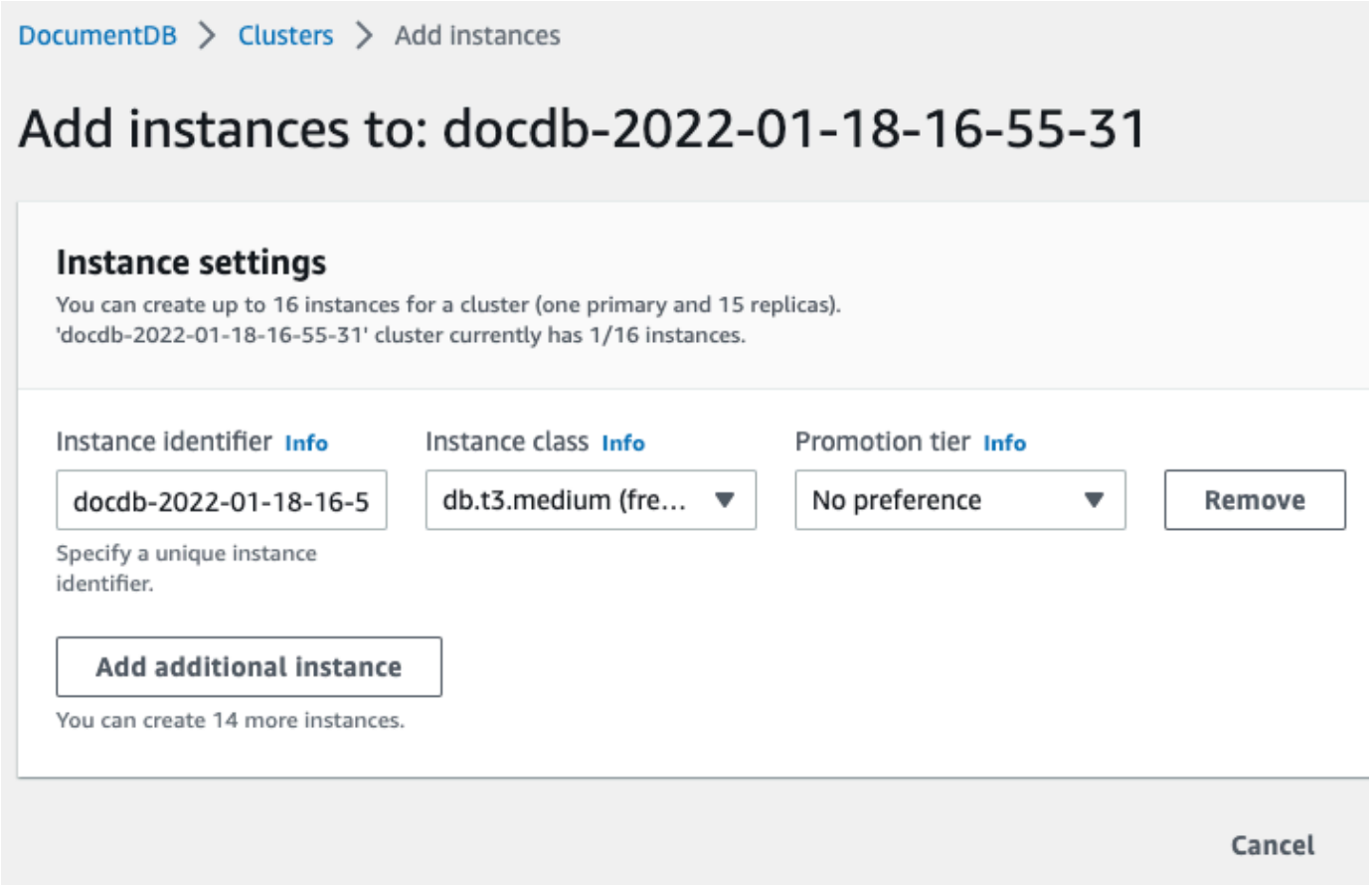
3. [推奨事項] ダイアログで、関心のあるセクションを展開し、推奨タスクを選択します。

   次の例では、推奨タスクは 1 つのインスタンスのみを持つ Amazon DocumentDB クラスターに適用されます。パフォーマンスと可用性を改善するために、別のインスタンスを追加することをお勧めします。



4. [Apply] を選択します。

   この例では、[インスタンスの追加] ダイアログが表示されます。



5. 新しいインスタンスの設定を変更し、作成を選択します。

Amazon DocumentDB イベントサブスクリプションを使用する

Amazon DocumentDB では、Amazon DocumentDB のイベントが発生したときに、Amazon Simple Notification Service (Amazon SNS) を使用して通知を送信します。これらの通知は、E メール AWS リージョン、テキストメッセージ、HTTP エンドポイントへの呼び出しなど、 の Amazon SNS でサポートされている任意の形式にすることができます。

Amazon DocumentDB は、サブスクライブ可能なカテゴリにイベントをグループ分けします。これにより、そのカテゴリのイベントが発生すると、通知を受け取ることができます。インスタンス、クラスター、スナップショット、クラスタースナップショット、または パラメータグループのイベントカテゴリをサブスクライブできます。例えば、特定のインスタンスのバックアップカテゴリをサブスクライブした場合、インスタンスに影響するバックアップ関連のイベントが発生するたびに通知が送信されます。また、イベントサブスクリプションが変更されても、通知を受け取ります。

イベントは、クラスターレベルとインスタンスレベルの両方で発生するため、クラスターまたはインスタンスをサブスクライブするとイベントを受信できます。

イベントサブスクリプションは、サブスクリプションを作成するときに指定したアドレスに送信されます。いくつかの異なるサブスクリプションを作成することもできます。例えば、すべてのイベント通知を受信するサブスクリプションと、本番稼働用のインスタンスに関する重要なイベントのみを含むサブスクリプションを作成できます。サブスクリプションを削除せずに通知を簡単にオフにできます。これを行うには、Amazon DocumentDB コンソールで [Enabled (有効)] ラジオボタンを [No (なし)]に設定します。

> **Important**
>
> Amazon DocumentDB はイベントストリームのイベントの順番を保証しません。イベントの順番は変わる場合があります。

Amazon DocumentDB では、Amazon SNS トピックの Amazon リソースネーム (ARN) を使用して、各サブスクリプションを識別します。Amazon DocumentDB コンソールでは、サブスクリプションの作成時に ARN が作成されます。

Amazon DocumentDB イベントサブスクリプションの請求は、Amazon SNS を通じて行われます。使用したイベント通知に対して、Amazon SNS 料金が適用されます。詳細については、「Amazon Simple Notification Service の料金」を参照してください。Amazon SNS の料金以外では、Amazon DocumentDB はイベントサブスクリプションの請求は行いません。

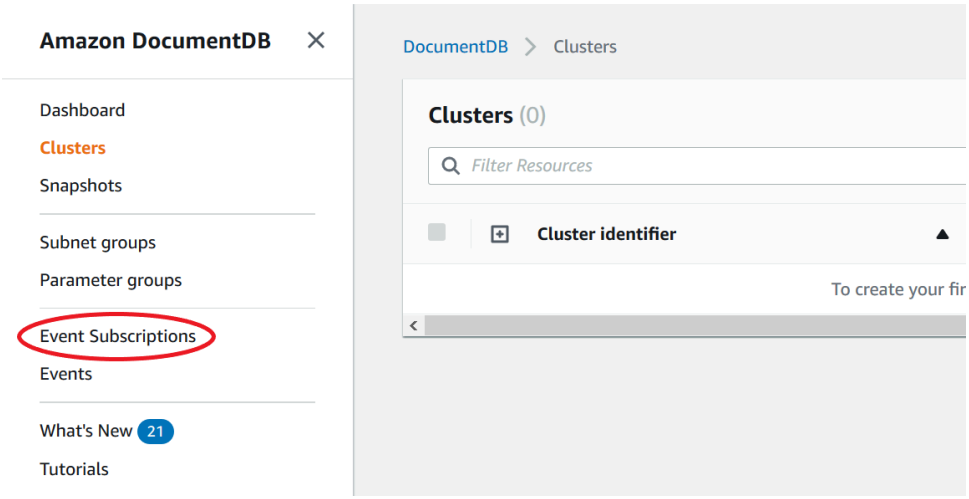
トピック

- Amazon DocumentDB イベントにサブスクライブする
- Amazon DocumentDB イベント通知サブスクリプションを管理する
- Amazon DocumentDB イベントのカテゴリとイベントメッセージ

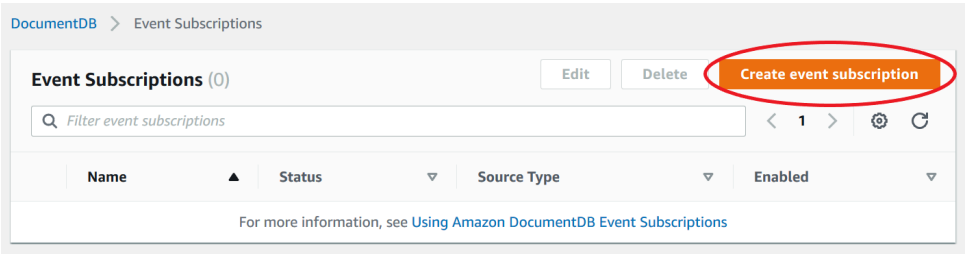
Amazon DocumentDB イベントにサブスクライブする

Amazon DocumentDB コンソールを使用して、次のようにイベントサブスクリプションをサブスクライブできます。

1. AWS マネジメントコンソール で にサインインしますhttps://console.aws.amazon.com/docdb。
2. ナビゲーションペインで、[Event subscriptions] (イベントサブスクリプション) を選択します。

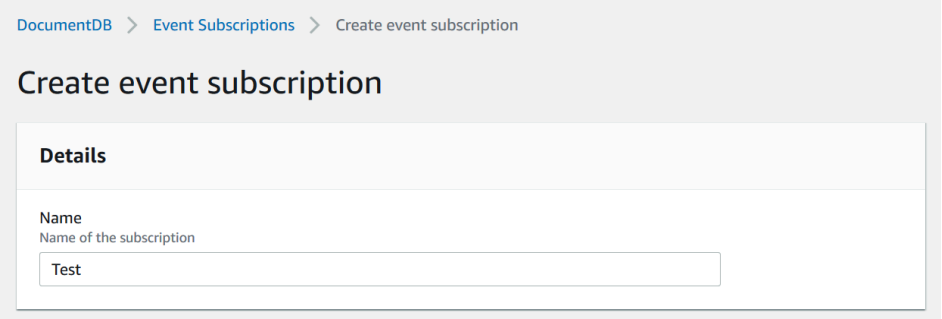


3. [イベントサブスクリプション] ページで、[イベントサブスクリプションの作成] を選択します。

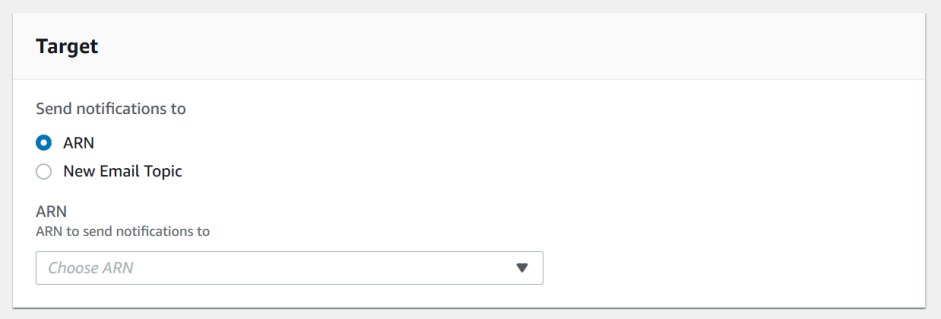


4. [イベントサブスクリプションの作成] ダイアログボックスで、次の操作を行います。

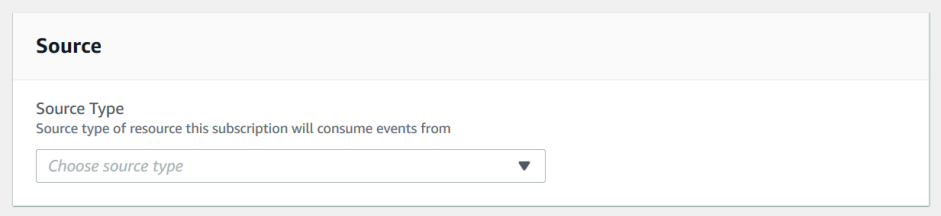
   - [名前] に、イベント通知サブスクリプションの名前を入力します。



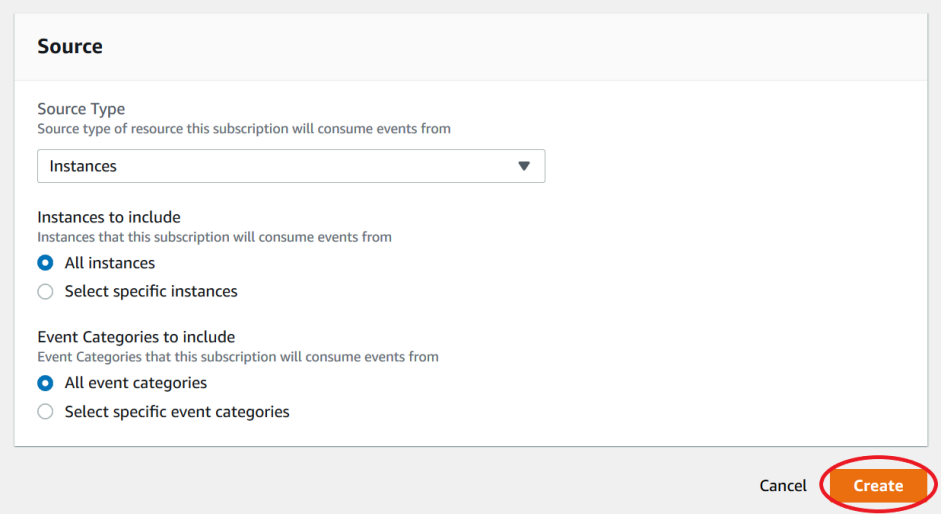
   - [ターゲット]で、通知を送信する場所を選択します。既存の ARN を選択できます。または [新しい電子メールトピック] を選択して、トピックの名前と受取人のリストを入力します。



- [ソース] で、ソースタイプを選択します。選択したソースタイプに応じて、イベント通知を受け取る対象のイベントカテゴリとソースを選択します。



- [作成] を選択します。



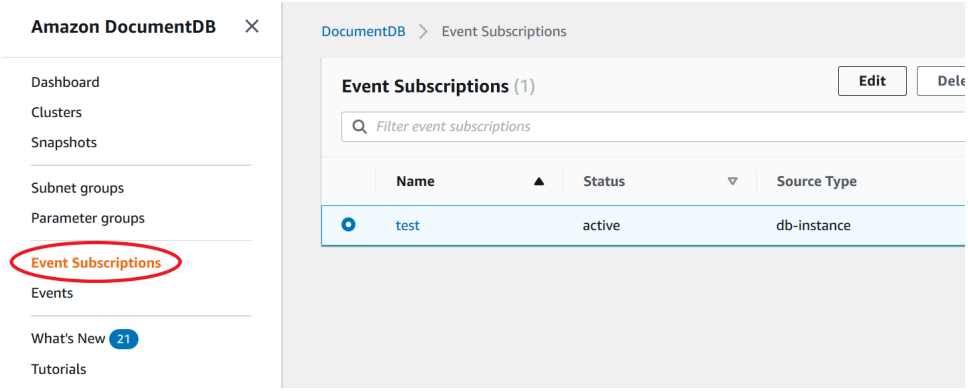
Amazon DocumentDB イベント通知サブスクリプションを管理する

Amazon DocumentDB コンソールのナビゲーションペインで [イベントサブスクリプション] を選択すると、サブスクリプションカテゴリと現在のサブスクリプションのリストを表示できます。特定のサブスクリプションを変更または削除することもできます。

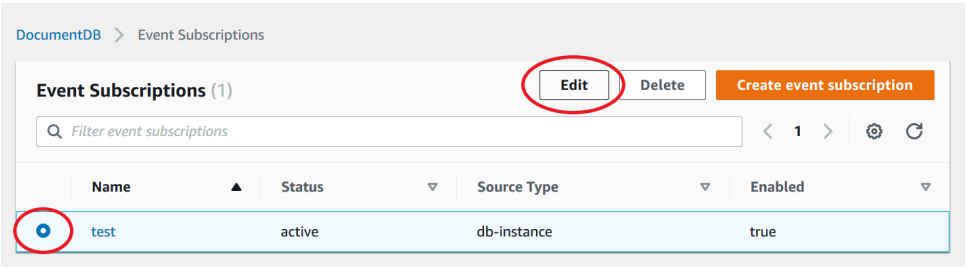
現在の Amazon DocumentDB イベント通知サブスクリプションを変更するには

1. AWS マネジメントコンソール で にサインインしますhttps://console.aws.amazon.com/docdb。

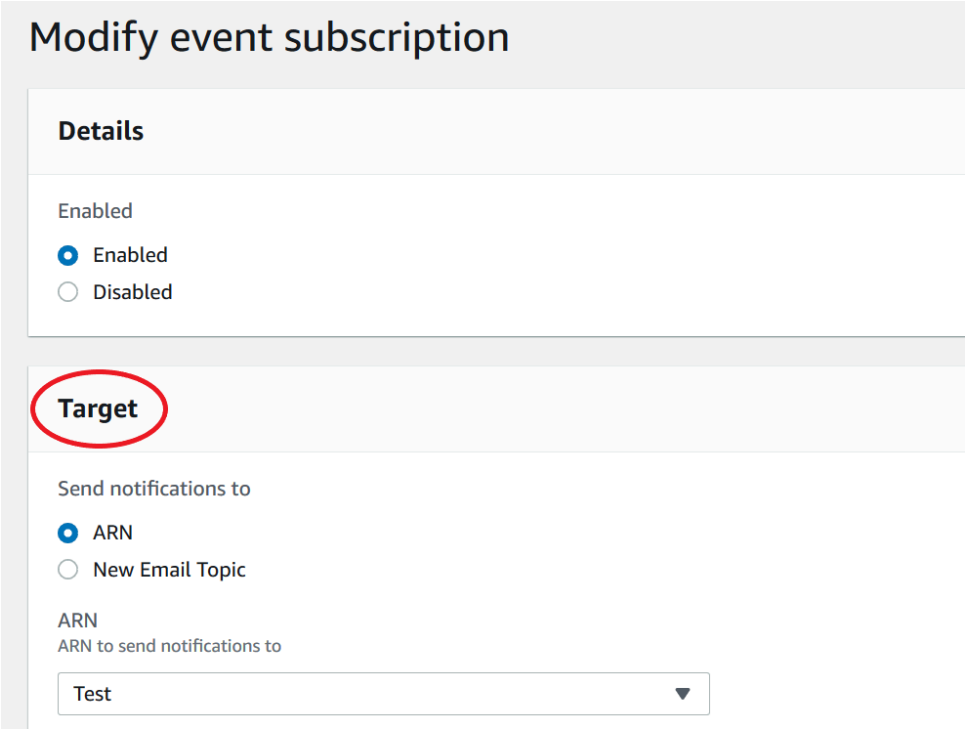
2. ナビゲーションペインで、[イベントサブスクリプション] を選択します。[イベントサブスクリプション] ペインにイベント通知サブスクリプションが一覧表示されます。



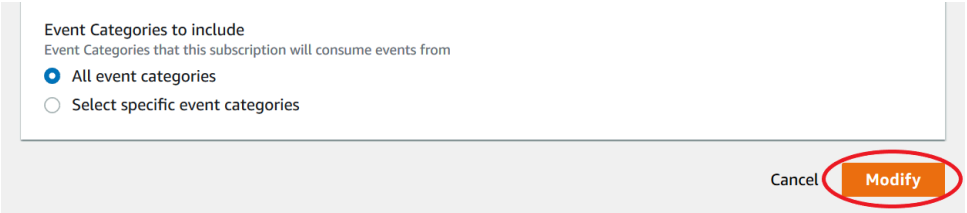
3. [イベントサブスクリプション] ペインで、変更するサブスクリプションを選択し、[編集] をクリックします。



4. [ターゲット] セクションまたは [ソース] セクションのいずれかでサブスクリプションを変更します。[ソース] セクションで選択または選択解除することで、ソース識別子を追加または削除できます。



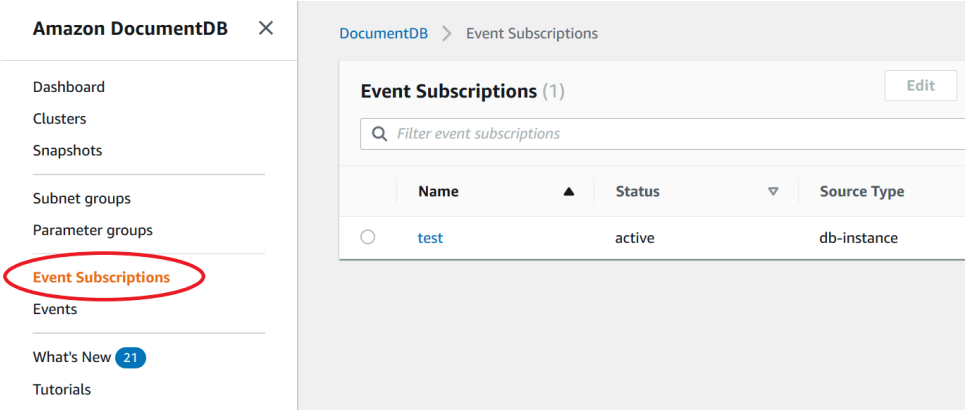
5. [Modify] (変更) を選択します。Amazon DocumentDB コンソールでは、サブスクリプションが変更されることが示されます。



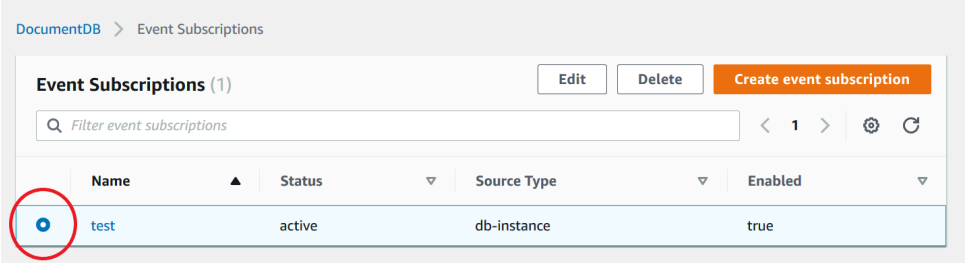
Amazon DocumentDB イベント通知サブスクリプションの削除

不要になったサブスクリプションは削除できます。トピックへのすべてのサブスクライバは、サブスクリプションにより指定されたイベント通知を受け取らなくなります。

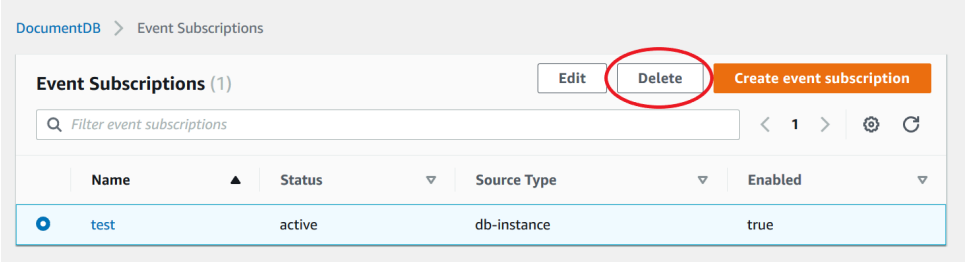
1. AWS マネジメントコンソール で にサインインしますhttps://console.aws.amazon.com/docdb。
2. ナビゲーションペインで、[Event subscriptions] (イベントサブスクリプション) を選択します。



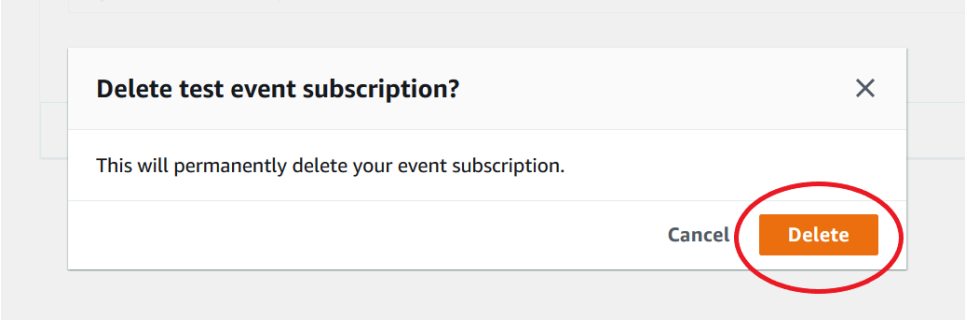
3. [イベントサブスクリプション] ペインで、削除するサブスクリプションを選択します。



4. [削除] を選択します。



5. この通知を完全に削除するかどうかを尋ねるポップアップウィンドウが表示されます。[削除] を選択します。



Amazon DocumentDB イベントのカテゴリとイベントメッセージ

Amazon DocumentDB では、 コンソールを使用してサブスクライブできるカテゴリ内で多数のイベントが生成されます。各カテゴリはインスタンス、クラスター、スナップショットまたはパラメータグループである 1 つのソースタイプに適用されます。

> **Note**
>
> Amazon DocumentDB では既存の Amazon RDS イベント定義と ID が使用されます。

インスタンスから発生する Amazon DocumentDB イベント

Category	説明
可用性	インスタンスが再起動しました。
可用性	インスタンスがシャットダウンしました。
設定変更	変更をインスタンスクラスに適用します。
設定変更	変更をインスタンスクラスに適用しました。
設定変更	プライマリ認証情報をリセットします。
作成	インスタンスが作成されました。
削除	インスタンスが削除されました
失敗	インスタンスは、互換性のない設定または基本的なストレージの問題により失敗しました。インスタンスのポイントインタイムの復元を開始します。
通知	インスタンスが停止しました。
通知	インスタンスが起動しました。
通知	停止中の最大許容時間を超えたため、インスタンスが起動されています。

Category	説明
復旧	インスタンスの復旧が開始しました。復旧時間は、復旧するデータの量に応じて変わります。
復旧	インスタンスの復旧が完了しました。
セキュリティパッチ	ご使用のインスタンスに対するオペレーティングシステムアップデートがあります。更新の適用については、「Amazon DocumentDB のメンテナンス」を参照してください。

クラスターから発生する Amazon DocumentDB イベント

Category	説明
作成	クラスターが作成されました
削除	クラスターが削除されました。
フェイルオーバー	以前のプライマリを再度プロモートします。
フェイルオーバー	インスタンスへのフェイルオーバーが完了しました。
フェイルオーバー	DB インスタンスへのフェイルオーバーを開始しました: %s
フェイルオーバー	DB インスタンスへの同じ AZ フェイルオーバーを開始しました: %s
フェイルオーバー	DB インスタンスへのクロスAZ フェイルオーバーを開始しました: %s
メンテナンス	クラスターにパッチが適用されました。
メンテナンス	データベースクラスターはアップグレードできない状態です: %s

Category	説明
通知	クラスターが停止しました。
通知	クラスターが開始しました。
通知	クラスターの停止が失敗しました。
通知	停止中の最大許容時間を超えたため、クラスターが開始されています。
通知	クラスターの名前を %s から %s に変更しました。

クラススナップショットから発生する Amazon DocumentDB イベント

次の表は、Amazon DocumentDB クラスターのスナップショットがソースタイプである場合のイベントカテゴリとイベントのリストを示しています。

Category	説明
バックアップ	手動クラススナップショットの作成
バックアップ	手動クラススナップショットが作成されました。
バックアップ	自動クラススナップショットを作成しています。
バックアップ	自動クラススナップショットが作成されました。

パラメータグループから発生する Amazon DocumentDB イベント

次の表は、パラメータグループがソースタイプである場合のイベントカテゴリとイベントのリストを示します。

Category	説明
設定変更	適用メソッド %s でパラメータ %s を %s に更新しました

Amazon DocumentDB と CloudWatch のモニタリング

Amazon DocumentDB (MongoDB 互換) は Amazon CloudWatch と統合することで、クラスターのオペレーションメトリクスを収集および分析できます。これらのメトリクスは、CloudWatch コンソール、Amazon DocumentDB コンソール、 AWS Command Line Interface (AWS CLI)、または CloudWatch API を使用してモニタリングできます。

CloudWatch では、メトリクスが指定したしきい値を超えた場合に通知を受け取ることができるように、アラームを設定することもできます。超過が発生した場合に修正作業を行うことができるように、Amazon CloudWatch Events を設定することもできます。CloudWatch とアラームの使用方法の詳細については、Amazon CloudWatch のドキュメント を参照してください。

トピック

- Amazon DocumentDB のメトリクス
- CloudWatch データの表示
- Amazon DocumentDB ディメンション
- opcounter メトリクスのモニタリング
- データベース接続のモニタリング

Amazon DocumentDB のメトリクス

Amazon DocumentDB クラスターとインスタンスの状態とパフォーマンスをモニタリングするために、 Amazon DocumentDB コンソールで、次のメトリクスを表示できます。

> **Note**
>
> 以下の表のメトリクスは、インスタンスベースのクラスターと Elastic クラスターの両方に適用されます。

トピック

- リソース使用率メトリクス
- レイテンシーメトリクス
- NVMe-backed インスタンスメトリクス
- オペレーションメトリクス
- コマンドレベルのメトリクス
- スループットメトリクス
- システムメトリクス
- T3 インスタンスメトリクス

リソース使用率メトリクス

メトリクス	説明
BackupRetentionPeriodStorageUsed	Amazon DocumentDB の保持期間内で特定時点への復元機能をサポートするために使用される、バックアップストレージの合計容量 (バイト単位)。TotalBackupStorageBilled メトリクスによって報告される合計に含まれます。各 Amazon DocumentDB クラスターに対して個別に計算されます。
ChangeStreamLogSize	変更ストリームログを保存するためにクラスターが使用するストレージの容量 (MB)。この値は、クラスターの合計ストレージのサブセット (VolumeBytesUsed) であり、クラスターのコストに影響します。ストレージの料金については、Amazon DocumentDB 製品ページ を

メトリクス	説明	
	参照してください。変更ストリームのログサイズは、クラスターで発生している変更の量と、変更ストリームの長期保存期間の関数です。変更ストリームの詳細については、[Amazon DocumentDB を用いた変更ストリームの使用] を参照してください。	
CPUUtilization	インスタンスによって使用される CPU のパーセント。	
DatabaseConnections	1 分間隔で取得するインスタンスで開いている接続 (アクティブおよびアイドル) の数。	
DatabaseConnectionsMax	1 分間にインスタンスで開いているデータベース接続 (アクティブおよびアイドル) の最大数。	
DatabaseConnectionsLimit	任意の時点でインスタンスで許可される同時データベース接続 (アクティブおよびアイドル) の最大数。	

メトリクス	説明
DatabaseConnectionsActiveMax	1 分間のインスタンスでのアクティブなデータベース接続の最大数。アクティブな接続は、アイドル接続ではなく、現在 オペレーションを実行している接続です。インスタンスタイプあたりのアクティブな接続クォータについては、「」を参照してくださいインスタンスクォータ。
DatabaseCursors	1 分間隔で取得するインスタンスで開いているカーソルの数。
DatabaseCursorsMax	1 分間にインスタンスで開いているカーソルの最大数。
DatabaseCursorsLimit	任意の時点でインスタンスで許可されるカーソルの最大数。
DatabaseCursorsTimedOut	1 分間にタイムアウトしたカーソルの数。
FreeableMemory	使用可能な RAM の容量 (バイト単位)。

メトリクス	説明	
FreeLocalStorage	このメトリクスでは、各インスタンスの一時テーブルとログで使用できるストレージの量が報告されます。この値は、インスタンスクラスによって異なります。インスタンスに対してより大きなインスタンスクラスを選択することで、インスタンス用の空きストレージ容量を増やすことができます。(これは DocumentDB Serverless には適用されません) 。	
LowMemThrottleQueueDepth	1 分間隔で消費される使用可能なメモリが少ないためにスロットされたリクエストのキューの深度。	
LowMemThrottleMaxQueueDepth	1 分間に消費される使用可能なメモリが少ないためにスロットされたリクエストの最大キュー深度。	
LowMemNumOperationsThrottled	1 分間に消費される使用可能なメモリが少ないためにスロットされたリクエストの数。	

メトリクス	説明	
SnapshotStorageUsed	バックアップ保持期間外に特定の Amazon DocumentDB クラスターのすべてのスナップショットによって使用されているバックアップストレージの合計容量 (バイト単位)。TotalBackupStorageBilled メトリクスによって報告される合計に含まれます。各 Amazon DocumentDB クラスターに対して個別に計算されます。	
SwapUsage	インスタンスで使用するスワップ領域の量。	
TotalBackupStorageBilled	特定の Amazon DocumentDB クラスターに対して請求されるバックアップストレージの合計容量 (バイト単位)。BackupRetentionPeriodStorageUsed メトリクスおよび SnapshotStorageUsed メトリクスによって測定されるバックアップストレージが含まれます。各 Amazon DocumentDB クラスターに対して個別に計算されます。	
TransactionsOpen	1 分間隔で実行されるインスタンスで開いているトランザクションの数。	

メトリクス	説明
TransactionsOpenMax	1 分間にインスタンスで開いているトランザクションの最大数。
TransactionsOpenLimit	任意の時点でインスタンスで許可される同時トランザクションの最大数。
VolumeBytesUsed	クラスターで使用されたストレージ容量 (バイト単位)。この値は、クラスターのコストに影響します。料金情報については、Amazon DocumentDB 製品ページ を参照してください。

レイテンシーメトリクス

メトリクス	説明
DBClusterReplicaLagMaximum	クラスター内のプライマリインスタンスと Amazon DocumentDB インスタンス間の最大遅延時間 (ミリ秒単位)。
DBClusterReplicaLagMinimum	クラスター内のプライマリインスタンスと各レプリカインスタンス間の最小遅延時間 (ミリ秒単位)。
DBInstanceReplicaLag	プライマリインスタンスからレプリカインスタンスにアップデートをレプリケートする際の遅延時間 (ミリ秒単位)。

メトリクス	説明
ReadLatency	1 回のディスク I/O オペレーションにかかる平均時間。
WriteLatency	1 回のディスク I/O オペレーションにかかる平均時間 (ミリ秒単位)。

NVMe-backed インスタンスメトリクス

メトリクス	説明
NVMeStorageCacheHitRatio	階層型キャッシュから提供されたリクエストの割合 (パーセント)。
FreeNVMeStorage	使用できる Ephemeral NVMe ストレージの容量。
ReadIOPSNVMeStorage	Ephemeral NVMe ストレージへのディスク読み取り I/O オペレーションの平均数。
ReadLatencyNVMeStorage	Ephemeral NVMe ストレージのディスク読み取り I/O オペレーションごとにかかる平均時間。
ReadThroughputNVMeStorage	Ephemeral NVMe ストレージ用にディスクから読み取られた 1 秒あたりの平均バイト数。
WriteIOPSNVMeStorage	Ephemeral NVMe ストレージへのディスク書き込み I/O オペレーションの平均数。

メトリクス	説明
WriteLatencyNVMeStorage	Ephemeral NVMe ストレージのディスク書き込み I/O オペレーションごとにかかる平均時間。
WriteThroughputNVMeStorage	Ephemeral NVMe ストレージ用にディスクに書き込まれる 1 秒あたりの平均バイト数。

オペレーションメトリクス

メトリクス	説明
DocumentsDeleted	1 分間に削除されたドキュメントの数。
DocumentsInserted	1 分間に挿入されたドキュメントの数。
DocumentsReturned	1 分間に返送されたドキュメントの数。
DocumentsUpdated	1 分間に更新されたドキュメントの数。
OpcountersCommand	1 分間に発行されたコマンドの数。
OpcountersDelete	1 分間に発行された削除オペレーションの数。
OpcountersGetmore	1 分間に発行されたゲットモアの数。
OpcountersInsert	1 分間に発行された挿入オペレーションの数。

メトリクス	説明
OpcountersQuery	1 分間に発行されたクエリの数。
OpcountersUpdate	1 分間に発行された更新オペレーションの数。
TransactionsStarted	1 分間にインスタンスで開始されたトランザクションの数。
TransactionsCommitted	1 分間にインスタンスでコミットされたトランザクションの数。
TransactionsAborted	1 分間にインスタンスで中断されたトランザクションの数。
TTLDeletedDocuments	1 分間に TTLMonitor によって削除されたドキュメントの数。

コマンドレベルのメトリクス

Amazon DocumentDB は、コマンドごとのレイテンシーと同時実行メトリクスを CloudWatch に発行します。これらのメトリクスは、ワークロードに最も寄与するオペレーションを特定し、テールレイテンシーの問題や同時実行プレッシャーを検出するのに役立ちます。

次の表の各コマンドは、1 分間に 3 つのメトリクスを出力します。

- AvgDuration — コマンドの平均実行時間をマイクロ秒単位で表します。
- P100Duration — コマンドの最大実行時間 (100 パーセンタイル) 。マイクロ秒単位。
- MaxConcurrent — コマンドの同時実行の最大数。

メトリクス名はパターン に従いますCommand.*commandName*.*metricName*。例えば、Command.find.AvgDuration、Command.aggregate.MaxConcurrent です。

メトリクス	説明
Command.find	find を使用してクエリを読み取ります。
Command.insert	ドキュメント挿入オペレーション。
Command.findAndModify	アトミックfind-and-modifyオペレーション。
Command.update	ドキュメントの更新オペレーション。
Command.delete	ドキュメント削除オペレーション。
Command.aggregate	集約パイプラインの実行。
Command.count	カウントオペレーション。
Command.distinct	個別の値クエリ。
Command.getMore	カーソルバッチ取得オペレーション。
Command.abortTransaction	トランザクション中止オペレーション。
Command.commitTransaction	トランザクションコミットオペレーション。

スループットメトリクス

メトリクス	説明
NetworkReceiveThroughput	クラスター内の各インスタンスが各クライアントから受信したネットワークスループットの量 (バイト/秒単位)。クラスターとクラスターボリューム内のインスタンス間のネットワークトラフィックは、こ

メトリクス	説明	
	のスループットに含まれません。	
NetworkThroughput	Amazon DocumentDB クラスター内の各インスタンスがクライアントで送受信したネットワークスループットの量 (バイト/秒単位)。クラスターとクラスターボリューム内のインスタンス間のネットワークトラフィックは、このスループットに含まれません。	
NetworkTransmitThroughput	クラスター内の各インスタンスが各クライアントに対して送信したネットワークスループットの量 (バイト/秒単位)。クラスターとクラスターボリューム内のインスタンス間のネットワークトラフィックは、このスループットに含まれません。	
ReadIOPS	1 秒あたりのディスク読み取り I/O オペレーションの平均回数。Amazon DocumentDB レポートは、IOPS の読み取りと書き込みを 1 分間隔で個別に行います。	
ReadThroughput	1 秒あたりのディスクからの平均読み取りバイト数。	

メトリクス	説明	
StorageNetworkReceiveThroughput	Amazon DocumentDB クラスター内の各インスタンスがクラスターストレージボリュームから受信したネットワークスループットの量 (バイト/秒単位)。	
StorageNetworkTransmitThroughput	Amazon DocumentDB クラスター内の各インスタンスがクラスターストレージボリュームに対して送信したネットワークスループットの量 (バイト/秒単位)。	
StorageNetworkThroughput	Amazon DocumentDB クラスター内の各インスタンスがクラスターストレージボリュームとの間で送受信したネットワークスループットの量 (バイト/秒単位)。	

メトリクス	説明	
VolumeReadIOPs	課金読み取り I/O オペレーションの平均回数は 5 分間隔で報告されます。課金読み取りオペレーションはクラスターボリュームレベルで計算され、クラスター内のすべてのインスタンスから集計された後、5 分おきに報告されます。この値は読み取りオペレーションのメトリクスを 5 分間受け取ることによって計算されます。課金読み取りオペレーションのメトリクスを受け取って 300 秒で割ることで、1 秒あたりの課金読み取りオペレーションの回数を決定できます。 例えば、VolumeReadIOPs が 13,686 を返す場合、1 秒あたりの課金読み取りオペレーションは 45 (13,686 / 300 = 45.62) です。 バッファキャッシュに存在しないデータベースのページをリクエストするクエリの課金読み取りオペレーションが発生するため、ストレージからロードする必要があります。課金読み取りオペレーションはストレージからクエリの結果が読み取られるのと同様に急増することがありますが、	

メトリクス	説明	
	その後バッファキャッシュにロードされます。	

メトリクス	説明	
VolumeWriteIOPs	課金書き込み I/O オペレーションの平均回数は 5 分間隔で報告されます。課金書き込みオペレーションはクラスターボリュームレベルで計算され、クラスター内のすべてのインスタンスから集計された後、5 分おきに報告されます。この値は、5 分間にわたる書き込みオペレーションメトリクスの値を受け取ることによって計算されます。課金書き込みオペレーションメトリクスの値を受け取って 300 秒で割ることで、1 秒あたりの課金書き込みオペレーションの回数を決定できます。 例えば、VolumeWriteIOPs が 13,686 を返す場合、1 秒あたりの課金書き込みオペレーション回数は 45 (13,686 / 300 = 45.62) です。 なお、VolumeReadIOPs と VolumeWriteIOPs メトリクスは DocumentDB ストレージレイヤーによって計算され、プライマリインスタンスとレプリカインスタンスによって実行される IO が含まれます。データは 20 ~ 30 分ごとに集計され、5 分間隔でレポートされるため、その時間帯のメトリクスには同じデー	

メトリクス	説明	
	タポイントが出力されます。1 分間隔で挿入オペレーションと相関するメトリクスを探している場合は、インスタンスレベルの WriteIOPS メトリクスを使用できます。メトリクスは、Amazon DocumentDB プライマリインスタンスの モニタリングタブで使用できます。	
`WriteIOPS`	1 秒あたりのディスク書き込み I/O オペレーションの平均回数。クラスターレベルで使用すると、`WriteIOPs` はクラスター内のすべてのインスタンスで評価されます。読み取りおよび書き込み IOPS は個別に 1 分間隔で報告されます。	
`WriteThroughput`	1 秒あたりのディスクへの平均書き込みバイト数。	

システムメトリクス

メトリクス	説明	
`AvailableMVCCIds`	ゼロに達する前に使用可能な残りの書き込みオペレーションの数を示すカウンター。このカウンターがゼロに達すると、ID が再利用されてリサイクルされるまで、クラスターは読み取り専用モードになり	

メトリクス	説明	
	ます。カウンターは書き込みオペレーションごとに減少し、ガベージコレクションが古い MVCC ID をリサイクルすると増加します。	
BufferCacheHitRatio	バッファキャッシュから提供されたリクエストの割合 (パーセント)。	
DiskQueueDepth	ディスクに対する読み取りまたは書き込み待機中の I/O オペレーションの数。	
EngineUptime	インスタンスの実行時間 (秒単位)。	
IndexBufferCacheHitRatio	バッファキャッシュから提供されたインデックスリクエストの割合 (パーセント)。インデックス、コレクション、またはデータベースを削除した直後は、メトリクスの権限の 100% を超えるスパイクが表示されることがあります。これは 60 秒後に自動的に修正されます。この制限は、今後のパッチアップデートで修正される予定です。	

メトリクス	説明
LongestActiveGCRuntime	最も長いアクティブなガベージコレクションプロセスの秒単位の期間。1 分以内に完了するプロセスを除き、1 分ごとに更新され、アクティブなオペレーションのみを追跡します。

T3 インスタンスメトリクス

メトリクス	説明
CPUCreditUsage	測定期間に消費された CPU クレジットの数。
CPUCreditBalance	インスタンスが蓄積する CPU クレジット数。CPU がバーストし、CPU クレジットが獲得するよりも速い速度で使用される際に、このバランスは枯渇します。
CPUSurplusCreditBalance	CPUCreditBalance の値がゼロになった時に CPU パフォーマンスを保持するために消費される、余剰 CPU クレジットの数。
CPUSurplusCreditsCharged	24 時間で獲得できる CPU クレジットの最大数を越えた、追加料金が発生する分の余剰 CPU クレジットの数。詳細については、[CPU クレジットの

メトリクス	説明	
	モニタリング] を参照してください。	

CloudWatch データの表示

Amazon CloudWatch データは、CloudWatch コンソール、Amazon DocumentDB コンソール、AWS Command Line Interface (AWS CLI)、または CloudWatch API を使用して表示できます。

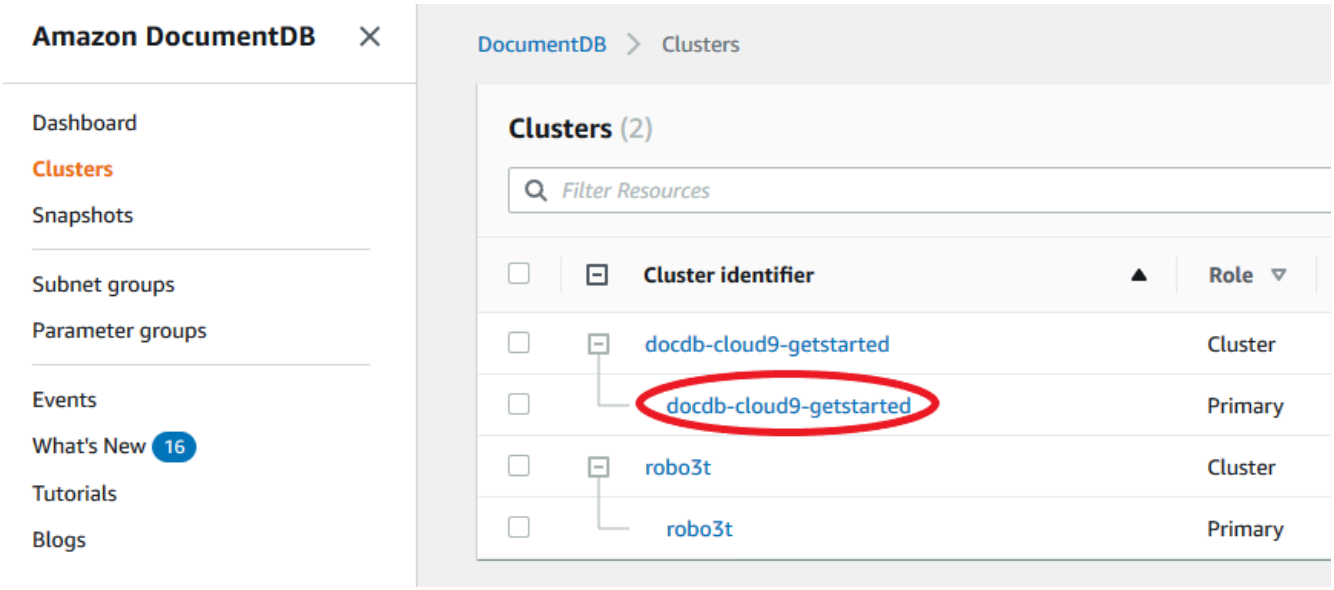
Using the AWS マネジメントコンソール

Amazon DocumentDB マネジメントコンソール を使用して CloudWatch メトリクスを表示するには、次のステップを完了します。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

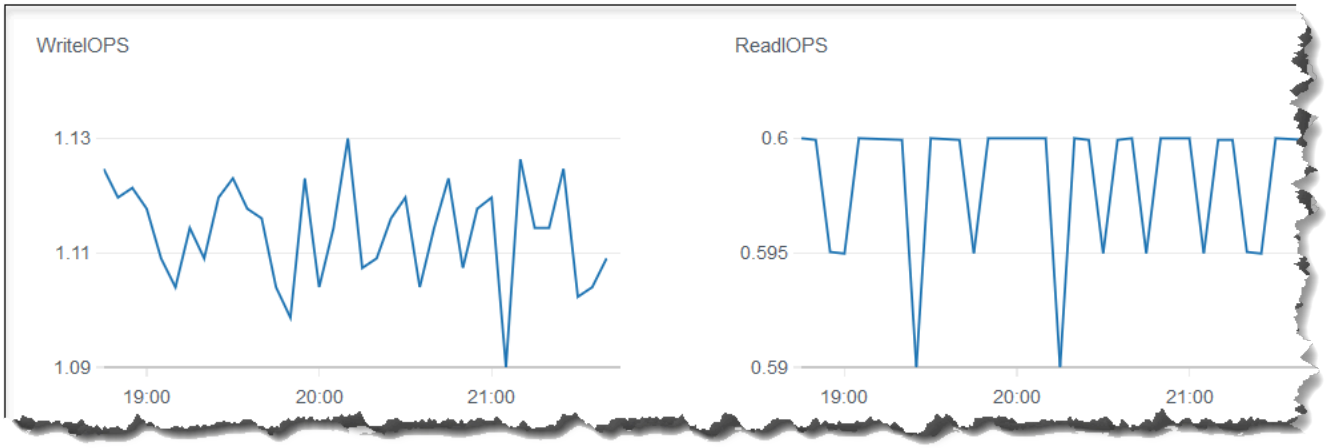
   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

3. クラスターナビゲーションボックスに、[Cluster Identifier]の列が表示されます。インスタンスは、次のスクリーンショットのようにクラスターの下に一覧表示されます。



4. インスタンスの一覧から、メトリクスを表示するインスタンスの名前を選択します。
5. インスタンスサマリーページで、モニタリング タブを選択すると、Amazon DocumentDB インスタンスのメトリクスをグラフィカルに表示することができます。各メトリクスに対してグラフを生成する必要があるため、[CloudWatch] グラフにデータが入力されるまでに数分かかる場合があります。

   次の図は、Amazon DocumentDB コンソールでの `WriteIOPS` と `ReadIOPS` の 2 つの CloudWatch メトリクスをグラフィカルに表示しています。



Using the CloudWatch Management Console

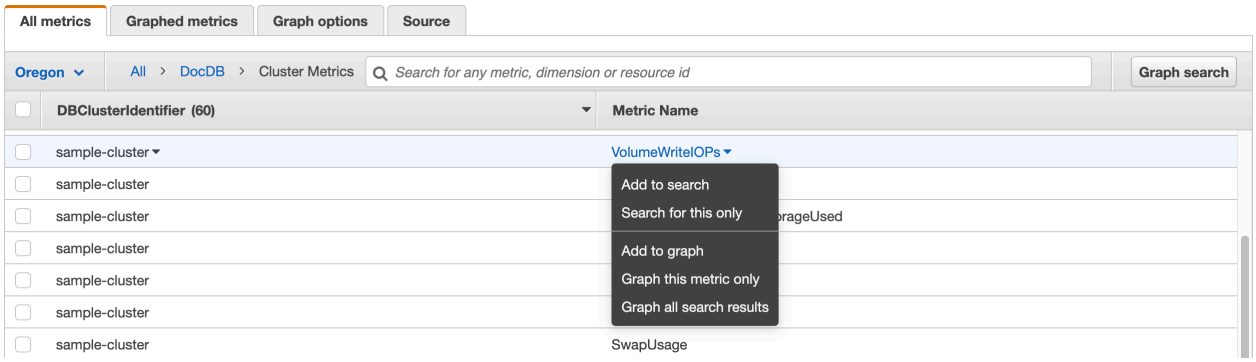
Amazon DocumentDB マネジメントコンソール を使用して、CloudWatch メトリクスを表示するには、次のステップを完了します。

1. にサインインし AWS マネジメントコンソール、 で Amazon DocumentDB コンソールを開きますhttps://console.aws.amazon.com/cloudwatch。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択してください。次に、サービス名のリストで、[DocDB] を選択します。
3. メトリクスのディメンション (例: クラスターメトリクス) を選択します。
4. 全てのメトリクス タブには、DocDB にあるディメンションの全てのメトリクスが表示されます。

    a. テーブルを並べ替えるには、列見出しを使用します。

    b. メトリクスをグラフ表示するには、メトリクスの横にあるチェックボックスを選択します。すべてのメトリクスを選択するには、テーブルの見出し行にあるチェックボックスを選択します。

    c. メトリクスでフィルタリングするには、メトリクス名にカーソルを合わせ、メトリック名の隣にあるドロップダウン矢印を選択します。次に、次の図に示すように、検索に追加を選択します。



Using the AWS CLI

Amazon DocumentDB の CloudWatch データを表示するには、以下のパラメータで CloudWatch `get-metric-statistics` オペレーションを使用します。

パラメータ

- **`--namespace`** — 必須。CloudWatch メトリクスを表示するサービス名前空間。Amazon DocumentDB の場合、これは `AWS/DocDB` でなければなりません。
- **`--metric-name`** — 必須。データを表示するメトリクスの名前。
- **`--start-time`** — 必須。最初に返すデータポイントを決定するタイムスタンプ。

指定された値は含まれます。つまり、結果には指定されたタイムスタンプのデータポイントが含まれます。タイムスタンプは ISO 8601 UTC 形式である必要があります (例: 2016-10-03T23:00:00Z)。

- **--end-time** — 必須。最後に返すデータポイントを決定するタイムスタンプ。

  指定された値は含まれます。つまり、結果には指定されたタイムスタンプのデータポイントが含まれます。タイムスタンプは ISO 8601 UTC 形式である必要があります (例: 2016-10-03T23:00:00Z)。

- **--period** — 必須。返されるデータポイントの詳細度 (秒)。通常の解像度のメトリクスについては、期間は最短 1 分 (60 秒) で、60 の倍数である必要があります。1 分未満の間隔で収集される高解像度メトリクスについては、期間は 1、5、10、30、60、または 60 の倍数にできます。

- **--dimensions** — オプション。メトリクスに複数のディメンションが含まれている場合は、各ディメンションの値を含める必要があります。CloudWatch は、ディメンションの一意の組み合わせをそれぞれ別のメトリクスとして扱います。ディメンションの特定の組み合わせが発行されていない場合は、その統計を取得することはできません。メトリクス作成時に使用した同じディメンションを指定する必要があります。

- **--statistics** — オプション。パーセンタイル以外のメトリクス統計。パーセンタイル統計の場合は、ExtendedStatistics を使用します。GetMetricStatistics を呼び出すときは、Statistics または ExtendedStatistics のどちらかを指定する必要があります。両方を指定することはできません。

  許可される値:

  - SampleCount
  - Average
  - Sum
  - Minimum
  - Maximum

- **--extended-statistics** — オプション。percentile の統計情報。p0.0 と p100 の間の値を指定します。GetMetricStatistics を呼び出すときは、Statistics または ExtendedStatistics のどちらかを指定する必要があります。両方を指定することはできません。

- **--unit** — オプション。特定のメトリクスの単位。メトリクスが複数の単位で報告される場合があります。単位を指定しない結果、すべての単位が返されます。メトリクスが報告しない単位のみを指定した場合、呼び出しの結果は null になります。

  使用できる値:

  - 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

次の例では、60 秒ごとにサンプルを取得し、2 時間の期間の最大 CPUUtilization を見つけます。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

このオペレーションの出力は、次のようになります。

```
{
       "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 ディメンション

Amazon DocumentDB のメトリクスは、アカウントまたはオペレーションの値で分類されます。CloudWatch コンソールを使用して、以下の表のいずれかのディメンションによってフィルタリングされた Amazon DocumentDB データを取得できます。

ディメンション	説明
`DBClusterIdentifier`	特定の Amazon DocumentDB クラスターに対してリクエストするデータをフィルタリングします。
`DBClusterIdentifier, Role`	特定の Amazon DocumentDB クラスターに対してリクエストするデータをフィルタリングして、インスタンスロール (WRITER/READER) 別にメトリクスを集計します。例えば、クラスターに属するすべての READER インスタンスのメトリクスを集計できます。
`DBInstanceIdentifier`	特定のデータベースインスタンスに対してリクエストするデータをフィルター処理します。

opcounter メトリクスのモニタリング

Opcounter メトリクスは、アイドルクラスターに対してゼロ以外の値 (通常は 50 まで) を持ちます。これは、Amazon DocumentDB が定期的にヘルスチェック、内部オペレーション、およびメトリクス収集タスクを実行するためです。

データベース接続のモニタリング

`db.runCommand( { serverStatus: 1 })` などのデータベースエンジンコマンドを使用して接続数を確認すると、CloudWatch を通じて `DatabaseConnections` で確認した接続数よりも最大 10 個多い数が表示される場合があります。これは、 Amazon DocumentDB が定期的にヘルスチェックとメトリクス収集タスクを実行するためで、`DatabaseConnections` では考慮されません。`DatabaseConnections` は、お客様が開始した接続のみを表します。

を使用した Amazon DocumentDB API コールのログ記録 AWS CloudTrail

Amazon DocumentDB (MongoDB 互換) は AWS CloudTrail、Amazon DocumentDB のユーザー、ロール、または のサービス (MongoDB 互換) によって実行されたアクションを記録する AWS サービスである と統合されています。CloudTrail は、Amazon DocumentDB コンソールからの呼び出しや Amazon DocumentDB SDK へのコード呼び出しを含む、Amazon DocumentDB のすべての AWS CLI API コールをイベントとしてキャプチャします。追跡を作成する場合は、Amazon DocumentDB のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます。追跡を設定しない場合でも、CloudTrail コンソールの [イベント履歴] で最新のイベントを確認できます。CloudTrail で収集された情報を使用して、Amazon DocumentDB に対するリクエスト、リクエスト元の IP アドレス、リクエストした者、リクエスト日時などのリクエストを確認できます。

> **Important**
>
> 特定の管理機能では、Amazon DocumentDB は Amazon Relational Database Service (Amazon RDS) と共有の運用テクノロジーを使用します。Amazon DocumentDB コンソール、、および API コールは AWS CLI、Amazon RDS API に対する呼び出しとしてログに記録されます。

詳細については AWS CloudTrail、AWS CloudTrail 「 ユーザーガイド」を参照してください。

CloudTrail 内の Amazon DocumentDB 情報

CloudTrail は、アカウントの作成 AWS アカウント 時に で有効になります。Amazon DocumentDB (MongoDB 互換) でアクティビティが発生すると、そのアクティビティはイベント履歴の他の AWS サービスイベントとともに CloudTrail イベントに記録されます。で最近のイベントを表示、検索、ダウンロードできます AWS アカウント。詳細については、CloudTrail イベント履歴でのイベントの表示を参照してください。

Amazon DocumentDB (MongoDB 互換) のイベントなど AWS アカウント、 のイベントの継続的な記録については、証跡を作成します。証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成するときに、証跡がすべての AWS リージョンに適用されます。証跡は、 AWS パーティションのすべてのリージョンからのイベントを記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、CloudTrail ログで収集されたイベントデータをさらに分析して処理するように他の AWS サービスを設定できます。詳細については、『AWS CloudTrail ユーザーガイド:』の以下のトピックを参照してください。

- 追跡作成の概要
- CloudTrail がサポートするサービスと統合
- CloudTrail の Amazon SNS 通知の設定
- CloudTrail ログファイルの複数のリージョンからの受け取り
- 複数のアカウントから CloudTrailログファイルを受け取る

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。ID 情報は次の判断に役立ちます。

- リクエストが、ルートとユーザー認証情報のどちらを使用して送信されたか。
- リクエストが、ロールとフェデレーションユーザーの一時的なセキュリティ認証情報のどちらを使用して送信されたか。
- リクエストが別の AWS サービスによって行われたかどうか。

詳細については、「CloudTrail userIdentity 要素」を参照してください。

Amazon DocumentDB オペレーションのプロファイリング

Amazon DocumentDB (with MongoDB compatibility) でプロファイラーを使用して、クラスターで実行されたオペレーションの実行時間と詳細をログに記録することができます。プロファイラーは、ク

ラスターで最も遅いオペレーションをモニタリングし、個々のクエリパフォーマンスとクラスター全体のパフォーマンスを向上させるのに役立ちます。

デフォルトでは、プロファイラー機能は無効になっています。有効にすると、プロファイラーは、お客様定義のしきい値 (100 ms など) の値より時間がかかっているオペレーションを Amazon CloudWatch Logs に記録します。ログに記録される詳細には、プロファイリングされたコマンド、時間、プランの概要、クライアントメタデータが含まれます。操作が CloudWatch Logs に記録された後、CloudWatch Logs インサイトを使用して Amazon DocumentDB プロファイリングデータを分析、モニタリング、アーカイブすることができます。一般的なクエリについては、「一般的なクエリ」セクションを参照してください 。

有効にすると、プロファイラーはクラスター内の追加のリソースを利用します。高いしきい値 (500 ミリ秒など) から開始し、値を徐々に下げて低速オペレーションを特定します。しきい値 50 ms から開始すると、高スループットのアプリケーションのクラスターでパフォーマンスの問題が発生する可能性があります。プロファイラーはクラスターレベルで有効であり、クラスター内のすべてのインスタンスとデータベースで動作します。Amazon DocumentDB は、ベストエフォート方式で Amazon CloudWatch Logs にオペレーションを記録します。

Amazon DocumentDB はプロファイラーを有効にするための追加料金を課しませんが、CloudWatch Logs の使用料は標準料金で請求されます。CloudWatch Logs の料金の詳細については、Amazon CloudWatch の料金 をご覧ください。

トピック

- サポートされているオペレーション
- 制限事項
- Amazon DocumentDB プロファイラーの有効化
- Amazon DocumentDB プロファイラーの無効化
- プロファイラーログのエクスポートの無効化
- Amazon DocumentDB プロファイラーログにアクセスする
- 一般的なクエリ

サポートされているオペレーション

Amazon DocumentDB プロファイラーでは、以下の操作をサポートしています。

- `aggregate`
- `count`

- delete
- distinct
- find (OP_QUERY およびコマンド)
- findAndModify
- insert
- update

制限事項

スロークエリプロファイラーは、クエリの結果セット全体が 1 つのバッチに収まることができ、結果セットが 16 MB 未満 (最大 BSON サイズ) の場合のみプロファイラーログを出力できます。16 MB を超える結果セットは、自動的に複数のバッチに分割されます。

ほとんどのドライバーまたはシェルは、小さなデフォルトのバッチサイズを設定する場合があります。バッチサイズは、クエリの一部として指定することができます。スロークエリログをキャプチャするには、予想される結果セットのサイズを超えるバッチサイズを使用します。結果セットのサイズが不明な場合、または異なる場合は、バッチサイズを大きな数値 (100k など) に設定します。

ただし、バッチサイズを大きくすると、クライアントにレスポンスを送信する前にデータベースから取得しなければならない結果も多くなります。一部のクエリでは、結果を得るまでに長い遅延が発生する可能性があります。結果セット全体を消費する予定がない場合、クエリの処理に多くの I/O を費やして結果を破棄する可能性があります。

Amazon DocumentDB プロファイラーの有効化

クラスターでプロファイラーを有効にするには、3 つのステップを実行します。すべての手順が完了していないと、プロファイリングログが CloudWatch Logs に送信されませんので、ご注意ください。プロファイラーはクラスターレベルで設定され、クラスターのすべてのデータベースとインスタンスで実行されます。

クラスターでプロファイラーを有効にするには

1. デフォルトのクラスターパラメータグループを変更できないため、使用できるカスタムクラスターパラメータグループがあることを確認してください。詳細については、「Amazon DocumentDB クラスターパラメータグループを作成する」を参照してください。
2. 使用可能なカスタムクラスターパラメータグループを使用して、profiler、profiler_threshold_ms および profiler_sampling_rate のパラメー

タを変更します。詳細については、「Amazon DocumentDB クラスターパラメータグループを変更する」を参照してください。

3. カスタムクラスターパラメータグループを使用し、CloudWatch Logs への `profiler` ログのエクスポートを有効にするために、クラスターを作成または変更します。

以下のセクションでは、 AWS マネジメントコンソール と AWS Command Line Interface () を使用してこれらのステップを実装する方法を示しますAWS CLI。

Using the AWS マネジメントコンソール

1. 開始する前に、Amazon DocumentDB クラスターと、まだ持っていない場合はカスタムクラスターパラメータグループを作成します。詳細については、「Amazon DocumentDB クラスターパラメータグループを作成する」および「Amazon DocumentDB クラスターの作成」を参照してください。
2. 使用可能なカスタムクラスターパラメータグループを使用して、以下のパラメータを変更します。詳細については、「Amazon DocumentDB クラスターパラメータグループを変更する」を参照してください。

   - `profiler` - クエリプロファイリングを有効化または無効化します。有効な値は、`enabled` および `disabled` です。デフォルト値は `disabled` です。プロファイリングを有効にするには、値を `enabled` に設定します。
   - `profiler_threshold_ms` - `profiler` を `enabled` に設定すると、`profiler_threshold_ms` より時間がかかっているコマンドは全て CloudWatch にログ出力されます。有効な値は `[50-INT_MAX]` です。デフォルト値は `100` です。
   - `profiler_sampling_rate` - プロファイリングやロギングを行うべき低速オペレーションの割合です。有効な値は `[0.0-1.0]` です。デフォルト値は `1.0` です。
3. カスタムクラスターパラメータグループを使用するようにクラスターを修正し、プロファイラーログエクスポートを Amazon CloudWatch に公開するように設定します。

   a. ナビゲーションペインで、[Clusters (クラスター)] を選択し、カスタムパラメータグループをクラスターに追加します。

   b. パラメータグループを関連付けるクラスター名の左側にあるボタンを選択します。[Actions (アクション)]、[Modify (変更)] の順に選択し、クラスターを変更します。

   c. [Cluster options (クラスターオプション)] で、上記のステップからカスタムパラメータグループを選択して、クラスターに追加します。

d. [Log exports (ログのエクスポート)] で [Profiler logs (プロファイラーログ)] をクリックして Amazon CloudWatchを選択します。

e. [続行] を選択して、変更の概要を表示します。

f. 変更を確認したら、直ちに、あるいは、[変更のスケジューリング] の下にある次のメンテナンスウィンドウ中に適用することができます。

g. [クラスターの変更] を選択して、新しいパラメータグループでクラスターを更新します。

Using the AWS CLI

次の手順では、クラスター `sample-cluster` でサポートされているすべてのオペレーションでプロファイラーを有効にします。

1. 開始する前に、次のコマンドを実行し、名前に `default` がなく、パラメータグループファミリーとして `docdb3.6` を持つクラスターパラメータグループの出力を確認して、使用可能なカスタムクラスターパラメータグループがあることを確認します。デフォルト以外のクラスターパラメータグループがない場合は、「Amazon DocumentDB クラスターパラメータグループを作成する」を参照してください。

```
aws docdb describe-db-cluster-parameter-groups \
    --query 'DBClusterParameterGroups[*].
[DBClusterParameterGroupName,DBParameterGroupFamily]'
```

次の出力では、`sample-parameter-group` のみが両方の条件を満たしています。

```
[
       [
           "default.docdb3.6",
           "docdb3.6"
       ],
       [
           "sample-parameter-group",
           "docdb3.6"
       ]
]
```

2. カスタムクラスターパラメータグループを使用して、以下のパラメータを変更します。

- profiler - クエリプロファイリングを有効化または無効化します。有効な値は、enabled および disabled です。デフォルト値は disabled です。プロファイリングを有効にするには、値を enabled に設定します。
- profiler_threshold_ms - profiler を enabled に設定すると、profiler_threshold_ms より長い時間を要するすべてのコマンドが CloudWatch にログ記録されます。有効な値は [50-INT_MAX] です。デフォルト値は 100 です。
- profiler_sampling_rate - プロファイリングやロギングを行うべき低速オペレーションの割合です。有効な値は [0.0-1.0] です。デフォルト値は 1.0 です。

```
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. Amazon DocumentDB クラスターを修正して、前のステップの sample-parameter-group カスタムクラスターパラメータグループを使用し、パラメータ --enable-cloudwatch-logs-exports を profiler に設定するようにします。

   次のコードは、前のステップから sample-cluster を使用するようにクラスター sample-parameter-group を変更し、有効な CloudWatch Logs のエクスポートに profiler を追加します。

   ```
   aws docdb modify-db-cluster \
          --db-cluster-identifier sample-cluster \
          --db-cluster-parameter-group-name sample-parameter-group \
          --cloudwatch-logs-export-configuration '{"EnableLogTypes":["profiler"]}'
   ```

   このオペレーションによる出力は、次のようになります。

   ```
   {
       "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
    }
}
```

Amazon DocumentDB プロファイラーの無効化

プロファイラを無効にするには、`profiler` パラメータと `profiler` ログの CloudWatch Logs へのエクスポートの両方を無効にします。

プロファイラーの無効化

`profiler` パラメータは AWS CLI、次のように AWS マネジメントコンソール または を使用して無効にできます。

Using the AWS マネジメントコンソール

次の手順では AWS マネジメントコンソール 、 を使用して Amazon DocumentDB を無効にしますprofiler。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。次に、プロファイラーを無効にするクラスターパラメータグループの名前を選択します。
3. 結果の [Cluster parameters (クラスターパラメータ)] ページで、`profiler` パラメータの左側にあるボタンを選択し、[Edit (編集)] を選択します。
4. [Modify profiler (プロファイラーの変更)] ダイアログボックスで、リストから [`disabled`] を選択します。
5. [Modify cluster parameter (クラスターパラメータの変更)] を選択します。

Using the AWS CLI

AWS CLIを使用してクラスターで `profiler` を無効にするには、次のようにクラスターを変更します。

```
aws docdb modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name sample-parameter-group \
    --parameters
 ParameterName=profiler,ParameterValue=disabled,ApplyMethod=immediate
```

プロファイラーログのエクスポートの無効化

CloudWatch Logs への`profiler`ログのエクスポートを無効にするには AWS CLI、次のように AWS マネジメントコンソール または を使用します。

Using the AWS マネジメントコンソール

次の手順では AWS マネジメントコンソール 、 を使用して Amazon DocumentDB の CloudWatch への ログのエクスポートを無効にします。

1. Amazon DocumentDB コンソールを、次の場所で開きます。https://console.aws.amazon.com/docdb
2. ナビゲーションペインで クラスター を選択します。ログのエクスポートを無効にするクラスターの名前の左側にあるボタンを選択します。
3. [アクション] メニューから [Modify (変更)] を選択します。
4. [Log exports (ログのエクスポート)] セクションまでスクロールし、[Profiler logs (プロファイラーログ)] を選択解除します。
5. [続行] をクリックしてください。
6. 変更を確認してから、この変更をクラスターにいつ適用するかを選択します。
   - Apply during the next scheduled maintenance window (次に予定されているメンテナンス期間中に適用)
   - すぐに適用
7. [クラスターを変更] を選択します。

Using the AWS CLI

次のコードでは、クラスター sample-cluster を変更し、CloudWatch プロファイラーログを無効にします。

Example

Linux、macOS、Unix の場合:

```
aws docdb modify-db-cluster \
   --db-cluster-identifier sample-cluster \
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["profiler"]}'
```

Windows の場合:

```
aws docdb modify-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["profiler"]}'
```

このオペレーションによる出力は、次のようになります。

```
{
    "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
    }
}
```

Amazon DocumentDB プロファイラーログにアクセスする

Amazon CloudWatch 上のプロファイルログにアクセスするには、次のステップに従います。

1. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
2. Amazon DocumentDB クラスターと同じリージョンであることを確認します。
3. ナビゲーションペインで [ログ] を選択します。

4. クラスターのプロファイラーログを確認するには、リストで /aws/docdb/*yourClusterName*/profiler を選択します。

   各インスタンスのプロファイルログは、対応する各インスタンス名の下に表示されます。

一般的なクエリ

以下に、プロファイルされたコマンドの分析に使用できる一般的なクエリをいくつか示します。CloudWatch Logs インサイト の詳細については、CloudWatch Logs インサイトでログデータを分析する と サンプルクエリ を参照してください。

指定されたコレクションの中で最も低速だった 10 件のオペレーションを取得する

```
filter ns="test.foo" | sort millis desc | limit 10
```

特定のコレクション内で 60 ms 以上かかったすべての更新オペレーションを取得する

```
filter millis > 60 and op = "update"
```

先月最も遅かった 10 件のオペレーションを取得する

```
sort millis desc | limit 10
```

プランの概要が COLLSCAN のすべてのクエリを取得する

```
filter planSummary="COLLSCAN"
```

Performance Insights を使用したモニタリング

Performance Insights は、既存の Amazon DocumentDB モニタリング機能を拡張して、クラスターのパフォーマンスを明確にし、これに影響を与えるあらゆる問題を分析しやすくします。Performance Insights ダッシュボードを使用してデータベースロードを視覚化したり、ロードを待機、クエリステートメント、ホスト、アプリケーションでフィルタリングしたりできます。

> **Note**
>
> Performance Insights は、Amazon DocumentDB 3.6、4.0、5.0、および 8.0 インスタンスベースのクラスターでのみ使用できます。

これはどのように役立ちますか?

- データベースのパフォーマンスを視覚化: ロードを視覚化して、データベースのロードがいつどこにあるかを判断します
- データベースのロードの原因を特定: インスタンスのロードの原因となっているクエリ、ホスト、アプリケーションを特定します
- データベースにロードがかかるタイミングを特定: Performance Insights ダッシュボードを拡大して特定のイベントに注目したり、縮小して長期間にわたる傾向を確認したりできます
- データベースロードに関するアラート: CloudWatch から新しいデータベースロードメトリクスに自動的にアクセスし、DB ロードメトリクスを他の Amazon DocumentDB メトリクスと一緒にモニタリングし、アラートを設定できます

Amazon DocumentDB Performance Insights にはどのような制限がありますか?

- AWS GovCloud (米国東部) および AWS GovCloud (米国西部) リージョンの Performance Insights は利用できません
- Amazon DocumentDB の パフォーマンスインサイトが保持するパフォーマンスデータは 7 日間までとなります
- 1,024 バイトを超えるクエリは Performance Insights に集約されません

トピック

- Performance Insights の概要
- Performance Insights の有効化と無効化
- Performance Insights 用のアクセスポリシーの設定
- Performance Insights ダッシュボードを使用してメトリクスを分析する
- Performance Insights API によるメトリクスの取得
- Performance Insights の Amazon CloudWatch メトリクス
- カウンターメトリクス用の Performance Insights

Performance Insights の概要

トピック

- 平均アクティブセッション
- ディメンション
- 最大 vCPU

平均アクティブセッション

データベース内のデータベースロード (DB ロード)アクティビティのレベルを測定します。毎秒収集されるPerformance Insights のキーメトリクスは DB Load です。DBLoad メトリクスの単位は、Amazon DocumentDB インスタンスの平均アクティブセッション数 (AAS)です。

アクティブなセッションとは、Amazon DocumentDB インスタンスにワークロードを送信し、レスポンスを待っている状態の接続です。例えば、Amazon DocumentDB インスタンスにクエリを送信すると、インスタンスでのクエリの処理中は、そのデータベースセッションは「アクティブな」状態となります。

平均アクティブセッションを取得するために、Performance Insights は、クエリを同時に実行するセッションの数をサンプリングします。AAS は、セッションの総数をサンプルの総数で割った値です。次の表は、実行中のクエリの連続する 5 つのサンプルを示しています。

例	クエリを実行しているセッション数	AAS	計算
1	2	2	2 セッション/1 サンプル
2	0	1	2 セッション/2 サンプル
3	4	2	6 セッション/3 サンプル
4	0	1.5	6 セッション/4 サンプル

例	クエリを実行しているセッション数	AAS	計算
5	4	2	10 セッション/5 サンプル

前の例では、1~5 の時間間隔の DB ロードは 2 AAS です。DB ロードの増加は、データベースで実行されているセッションが平均して増えることを意味します。

ディメンション

この DB Load メトリクスは、ディメンションと呼ばれるサブコンポーネントに分割できるため、他の時系列メトリクスとは異なります。ディメンションは、DB Load メトリクスのさまざまな特性のカテゴリと考えることができます。パフォーマンスの問題を診断する場合、最も有用なディメンションは待機状態と上位のクエリです。

待機状態

待機状態 を指定すると、クエリステートメントは、特定のイベントが発生するまで待機してから、実行を継続できます。例えば、クエリステートメントは、ロック済みのリソースのロックが解除されるまで待機することがあります。DB Load と待機イベントを組み合わせると、セッションの状態の全体像を得ることができます。Amazon DocumentDB には次のようなさまざまな待機状態があります。

Amazon DocumentDB の待機状態	待機状態の説明
ラッチ	ラッチ待機状態は、セッションがバッファープールのページングを待っているときに発生します。システムによる大規模なクエリの処理やコレクションのスキャンが頻繁に行われる場合や、バッファープールが小さすぎてワーキングセットを処理できない場合に、バッファープールのページインとページアウトが頻繁に起こる可能性があります。
CPU	CPU 待機状態は、セッションが CPU 上で待機しているときに発生します。

Amazon DocumentDB の待機状態	待機状態の説明
CollectionLock	CollectionLock 待機状態は、セッションがコレクションのロック取得を待機中の場合に発生します。これらのイベントは、コレクションに対して DDL 操作が行われたときに発生します。
DocumentLock	DocumentLock 待機状態は、セッションがドキュメントのロック取得を待機中の場合に発生します。同じドキュメントへの同時書き込みの数が多いと、そのドキュメントの DocumentLock 待機状態が増えます。
SystemLock	SystemLock 待機状態は、セッションがシステムで待機しているときに発生します。これは、実行時間の長いクエリ、実行時間の長いトランザクション、またはシステムの高い同時実行性が頻繁に場合に発生する可能性があります。
IO	IO 待機状態は、セッションが IO の完了を待っているときに発生します。
BufferLock	BufferLock 待機状態は、セッションがバッファーの共有ページのロック取得を待機中の場合に発生します。BufferLock 待機状態は、他のプロセスが要求されたページのオープンカーソルを保持していると、長くなる可能性があります。
LowMemThrottle	LowMemThrottle 待機状態は、Amazon DocumentDB インスタンスに大量のメモリロードがかかってセッションが待機しているときに発生します。この状態が長時間続く場合は、インスタンスをスケールアップしてメモリを追加することを検討してください。詳細については、「リソースガバナー」を参照してください。

Amazon DocumentDB の待機状態	待機状態の説明
BackgroundActivity	BackgroundActivity 待機状態は、セッションが内部システムプロセスを待機しているときに発生します。
その他	その他の待機状態は内部待機状態です。この状態が長時間続く場合は、このクエリを終了することを検討してください。詳細については、「長時間実行されているクエリやブロックされているクエリを見つけて終了する方法」を参照してください。

上位のクエリ

待機状態はボトルネックを示しますが、上位のクエリは、どのクエリが DB ロードの最も大きな原因になっているかを示します。例えば、多くのクエリが現在データベースで実行されている可能性がありますが、1 つのクエリが DB ロードの 99% を占めている可能性もあります。この場合、ロードが高いと、クエリに問題がある可能性があります。

最大 vCPU

ダッシュボードの [データベースロード] グラフで、セッション情報が収集、集計、表示されます。アクティブなセッションが最大 CPU 容量を超えているかどうかを確認するには、最大 vCPU ラインとの関係を調べます。最大 vCPU 値は、お使いの Amazon DocumentDB インスタンスの vCPU (仮想 CPU) のコア数によって決まります。

DB ロードが [Max vCPU (最大 vCPU)] ラインをしばしば超過し、プライマリ待機状態が CPU である場合、CPU が過ロードになっています。この場合、インスタンスへの接続を抑制したり、CPU ロードの高いクエリを調整したり、より大きなインスタンスクラスを検討する必要があります。待機状態の高い一貫したインスタンスは、解決するボトルネックまたはリソースの競合問題がある可能性があることを示します。これは、DB ロードが最大 vCPU ラインを超えていない場合にも該当します。

Performance Insights の有効化と無効化

Performance Insights を使用するには、DB インスタンスで有効にします。必要に応じて、後で無効にすることができます。Performance Insights を有効または無効にしても、ダウンタイム、再起動、フェイルオーバーが発生することはありません。

Performance Insights エージェントは DB ホストの限られた CPU とメモリを消費します。DB のロードが高い場合、エージェントはデータ収集の頻度を下げることでパフォーマンスへの影響を抑えます。

クラスター作成時の Performance Insights の有効化

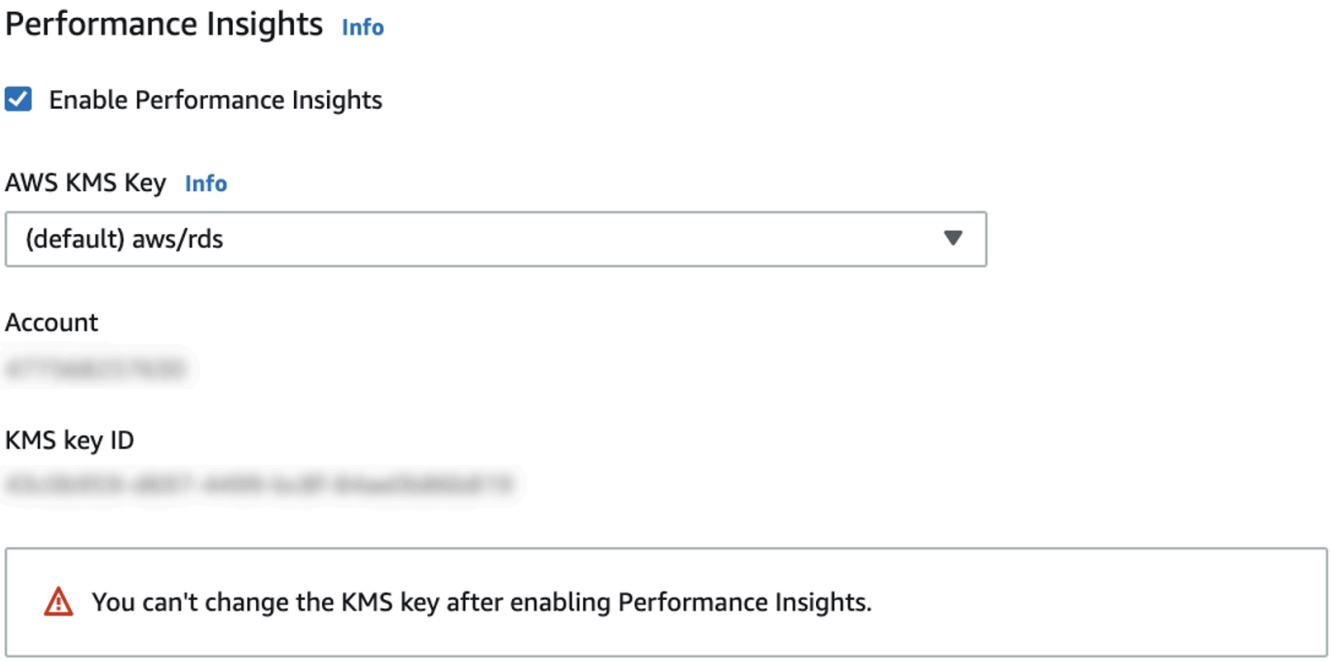
コンソールでは、新しい DB インスタンスを作成または変更するときに、Performance Insights を有効または無効にすることができます。

の使用 AWS マネジメントコンソール

Amazon DocumentDB クラスターを作成する際に、コンソール上でパフォーマンスインサイトを有効にすることができます。新しい Amazon DocumentDB クラスターを作成する際に、[パフォーマンスインサイト] セクションの [パフォーマンスインサイトを有効化する] を選択することで、パフォーマンスインサイトを有効化できます。

コンソールでの手順

1. クラスターを作成する手順については、「Amazon DocumentDB クラスターの作成」を参照してください。
2. [Performance Insights] セクションで、[Performance Insights の有効化] を選択します。



> **Note**
>
> Performance Insights データ保持期間は 7 日間です。

AWS KMS key — AWS KMS キーを指定します。Performance Insights は、 AWS KMS キーを使用してすべての機密データを暗号化します。データは、転送中と不使用時のいずれも暗号化されます。詳細については、「Performance Insights の AWS AWS KMS ポリシーの設定」を参照してください。

インスタンス変更時の有効化と無効化

コンソールまたは AWS CLIを使用し、DB インスタンスを変更して Performance Insights を有効にできます。

Using the AWS マネジメントコンソール

コンソールでの手順

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. [クラスター] を選択します。

3. DB インスタンスを選択し、[変更] を選択します。
4. [Performance Insights] セクションで、[Performance Insights の有効化] または [Performance Insightsの無効化] を選択します。

> **Note**
>
> Performance Insights を有効にするを選択した場合は、 AWS AWS KMS キーを指定できます。Performance Insights は、 AWS KMS キーを使用してすべての機密データを暗号化します。データは、転送中と不使用時のいずれも暗号化されます。詳細については、「保管中の Amazon DocumentDB データの暗号化」を参照してください。

5. [続行] をクリックしてください。
6. [Scheduling of Modifications] で、[Apply immediately] を選択します。「次にスケジュールされたメンテナンスウィンドウで適用」 を選択すると、インスタンスではこの設定が無視され、Performance Insights が直ちに有効になります。
7. [インスタンスの変更] を選択します。

Using the AWS CLI

`create-db-instance` または `modify-db-instance` AWS AWS CLI コマンドを使用する場合、 を指定して Performance Insights を有効にするか`--enable-performance-insights`、を指定して無効にすることができます`--no-enable-performance-insights`。

次の手順で、 AWS AWS CLIを使用して DB インスタンスで Performance Insights を有効化または無効化する方法を説明します。

AWS AWS CLI 手順

`modify-db-instance` AWS AWS CLI コマンドを呼び出し、次の値を指定します。

- `--db-instance-identifer` - DB インスタンスの名前
- 有効にする場合は `--enable-performance-insights`、無効にする場合は `--no-enable-performance-insights`

Example

次の例では、`sample-db-instance` の Performance Insights を有効にします。

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
```

Performance Insights 用のアクセスポリシーの設定

Performance Insights にアクセスするには、 AWS Identity and Access Management (IAM) から適切なアクセス許可を得る必要があります。アクセス権の付与については、次のオプションがあります。

- `AmazonRDSPerformanceInsightsReadOnly` マネージドポリシーをアクセス許可セットまたはロールにアタッチします。
- カスタム IAM ポリシーを作成し、アクセス許可セットまたはロールにアタッチします。

また、Performance Insights を有効にしたときにカスタマーマネージドキーを指定した場合は、アカウント内のユーザーが KMS キーに対する `kms:Decrypt` および `kms:GenerateDataKey` アクセス許可を持っていることを確認します。

> **Note**
>
> AWS KMS キーによるencryption-at-restとセキュリティグループ管理のために、Amazon DocumentDB は Amazon RDS と共有されている運用テクノロジーを活用します。

AmazonRDSPerformanceInsightsReadOnly ポリシーの IAM プリンシパルへのアタッチ

`AmazonRDSPerformanceInsightsReadOnly` は、Amazon DocumentDB Performance Insights API のすべての読み取り専用オペレーションへのアクセスを許可する AWSマネージドポリシーです。現在、この API のすべてのオペレーションは読み取り専用で

す。AmazonRDSPerformanceInsightsReadOnly をアクセス許可セットまたはロールにアタッチすると、受取人は他のコンソール機能とともに Performance Insights を使用できます。

Performance Insights 用のカスタム IAM ポリシーの作成

AmazonRDSPerformanceInsightsReadOnly ポリシーを持たないユーザーの場合は、ユーザーマネージド IAM ポリシーを作成または変更して、Performance Insights に対するアクセス権限を付与できます。ポリシーをアクセス許可セットまたはロールにアタッチすると、受取人は Performance Insights を使用できます。

カスタムポリシーを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、ポリシー を選択してください。
3. [Create policy] (ポリシーの作成) を選択します。
4. [ポリシーの作成] ページで、[JSON] タブを選択します。
5. 次のテキストをコピーして貼り付け、*us-east-1* を AWS リージョンの名前に置き換え、*111122223333* を顧客アカウント番号に置き換えます。

   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. [ポリシーの確認] を選択します。
7. ポリシーの名前と (必要に応じて) 説明を入力し、[ポリシーの作成] を選択します。

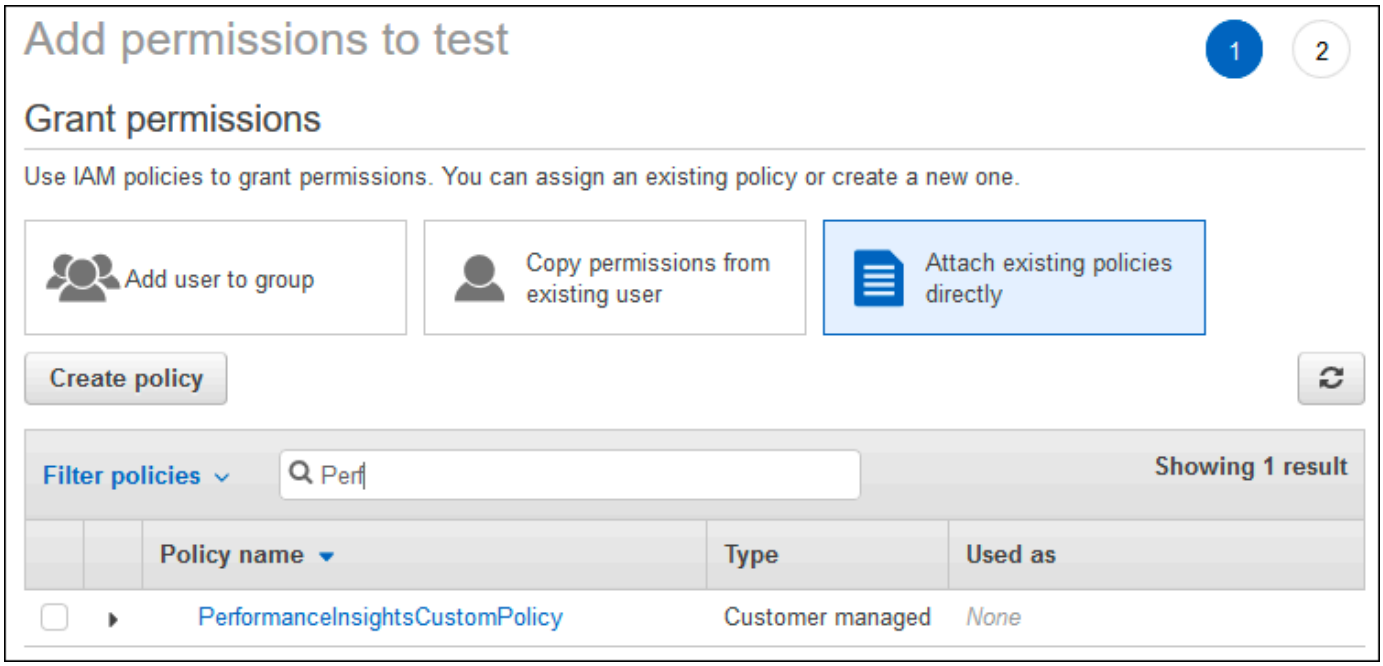
これで、そのポリシーをアクセス許可セットまたはロールにアタッチできます。次の手順では、この目的で使用できるユーザーが既に存在することを前提としています。

ポリシーをユーザーにアタッチするには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [ユーザー] を選択します。
3. リストから存在するユーザーを 1 人選択します。

> **Important**
>
> Performance Insights を使用するには、カスタムポリシーのほかに別のポリシーで、Amazon DocumentDB へのアクセスを許可されている必要があります。たとえば、事前定義された AmazonDocDBReadOnlyAccess ポリシーは Amazon DocDB への読み取り専用アクセスを許可します。詳細については、「ポリシーを使用したアクセスの管理」を参照してください。

4. [Summary (概要)] ページで、[Add permissions (許可の追加)] を選択します。
5. [既存のポリシーを直接アタッチする] を選択します。検索を行う場合は、次のようにポリシー名の初期の数文字を入力します。



6. ポリシーを選択し、[次へ: レビュー] を選択します。
7. [アクセス権限の追加] を選択します。

Performance Insights の AWS KMS ポリシーの設定

Performance Insights は、 を使用して機密データ AWS KMS key を暗号化します。API またはコンソールを通じて Performance Insights を有効にする場合は、次のオプションを使用できます。

- デフォルトを選択します AWS マネージドキー。

Amazon DocumentDB は、新しい DB インスタンス AWS マネージドキー に を使用します。Amazon DocumentDB は、 AWS アカウントの AWS マネージドキー を作成します。Amazon AWS マネージドキー Amazon DocumentDB AWS のアカウントは、 AWS リージョンごとに異なります。

- カスタマーマネージドキーを選択します。

  カスタマーマネージドキーを指定する場合、Performance Insights API を呼び出すアカウント内のユーザーは、KMS キーに対する `kms:Decrypt` および `kms:GenerateDataKey` アクセス許可が必要です。IAM ポリシーを使用する代わりに、キーポリシーを使用して KMS AWS KMS キーのアクセス許可を管理します。詳細については、「AWS KMSでのキーポリシーの使用」を参照してください。

Example

次のサンプルキーポリシーでは、KMS キーポリシーにステートメントを追加する方法が示されています。これらのステートメントは、Performance Insights へのアクセスを許可します。の使用方法によっては AWS KMS、いくつかの制限を変更することもできます。ポリシーにステートメントを追加する前に、すべてのコメントを削除してください。

Performance Insights ダッシュボードを使用してメトリクスを分析する

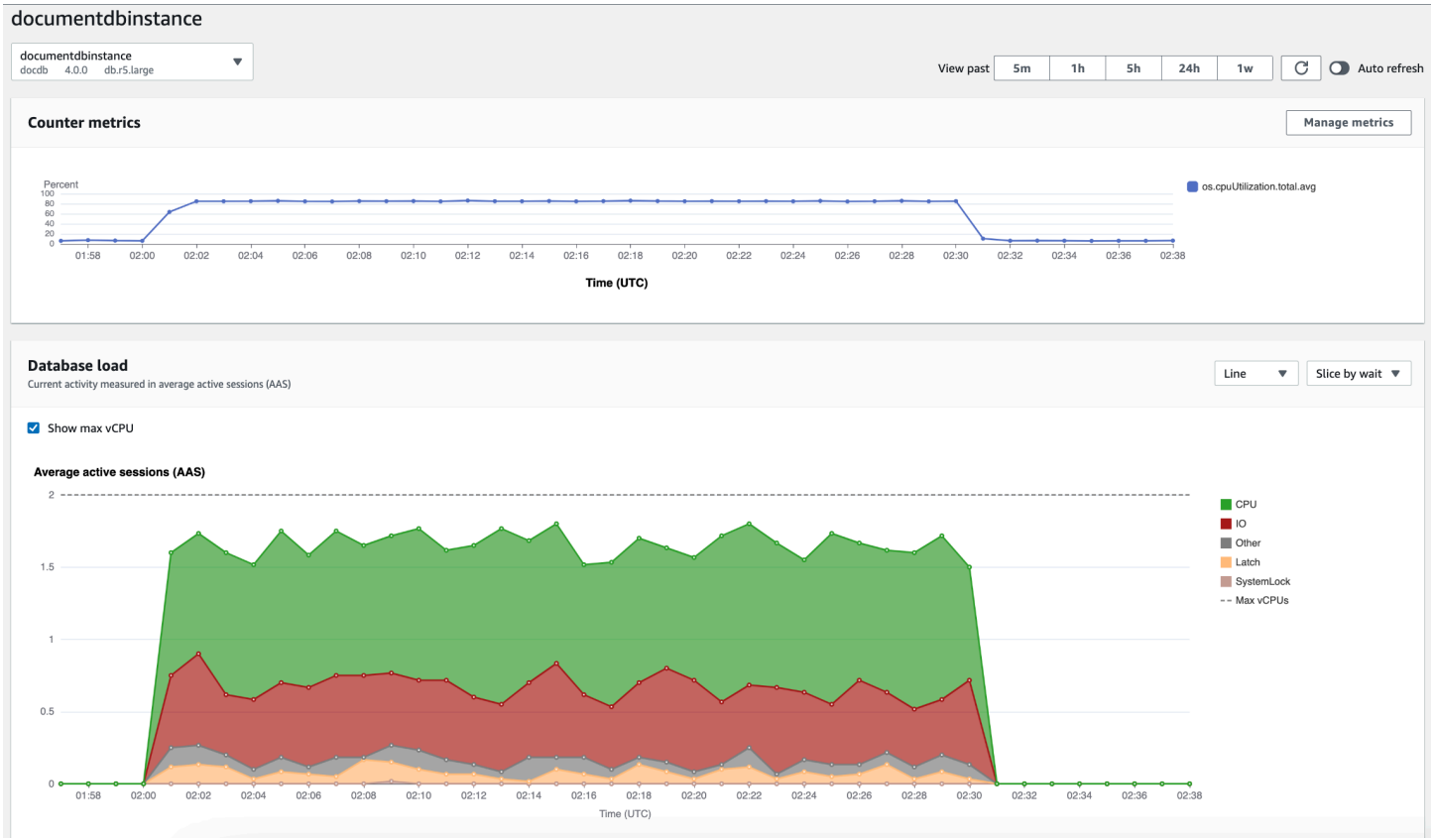
Performance Insights ダッシュボードには、パフォーマンスの問題を分析し、解決するのに役立つ、データベースのパフォーマンス情報が含まれます。ダッシュボードのメインページで、データベースロード (DB ロード) に関する情報を確認できます。待機状態やクエリなどのディメンションによって、DB のロードを「スライス」することが可能です。

トピック

- Performance Insights ダッシュボードの概要
- Performance Insights ダッシュボードを開く
- 待機状態によるデータベースロードの分析
- [上位のクエリ] タブの概要
- データベースロードグラフのズームイン

Performance Insights ダッシュボードの概要

ダッシュボードは、Performance Insights を操作する最も簡単な方法です。次の例では、Amazon DocumentDB インスタンスのダッシュボードを示しています。デフォルトでは、Performance Insights ダッシュボードには過去 60 分間のデータが表示されます。



このダッシュボードは、以下の各パートに分割されています。

1. カウンターメトリクス - 特定のパフォーマンスカウンターメトリクスのデータを示します。
2. データベースロード - DB ロードと DB インスタンスの容量を比較して、最大 vCPU ラインで表します。
3. 上位ディメンション – DB ロードに影響が大きい上位ディメンションを表示します。これらのディメンションには `waits`、`queries`、`hosts`、`databases`、`applications` が含まれます。

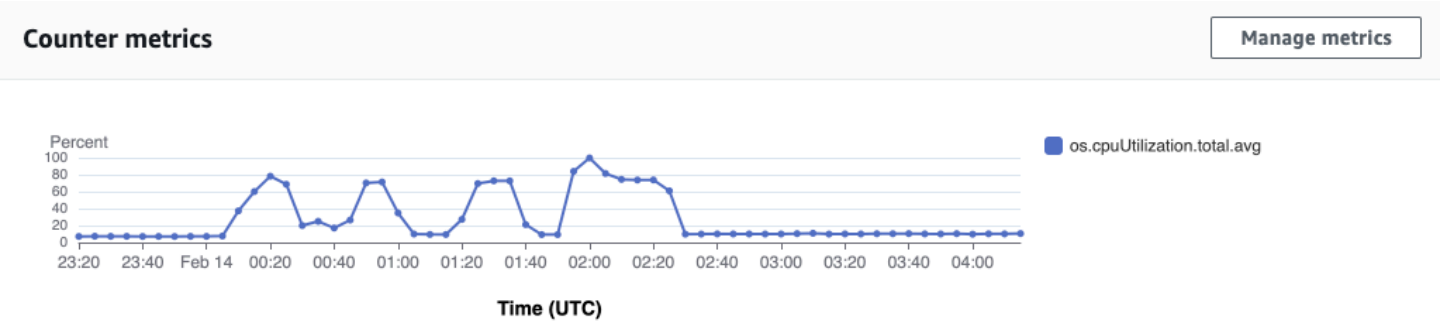
トピック

- カウンターメトリクスグラフ
- データベースロードのグラフ
- 上位のディメンションテーブル

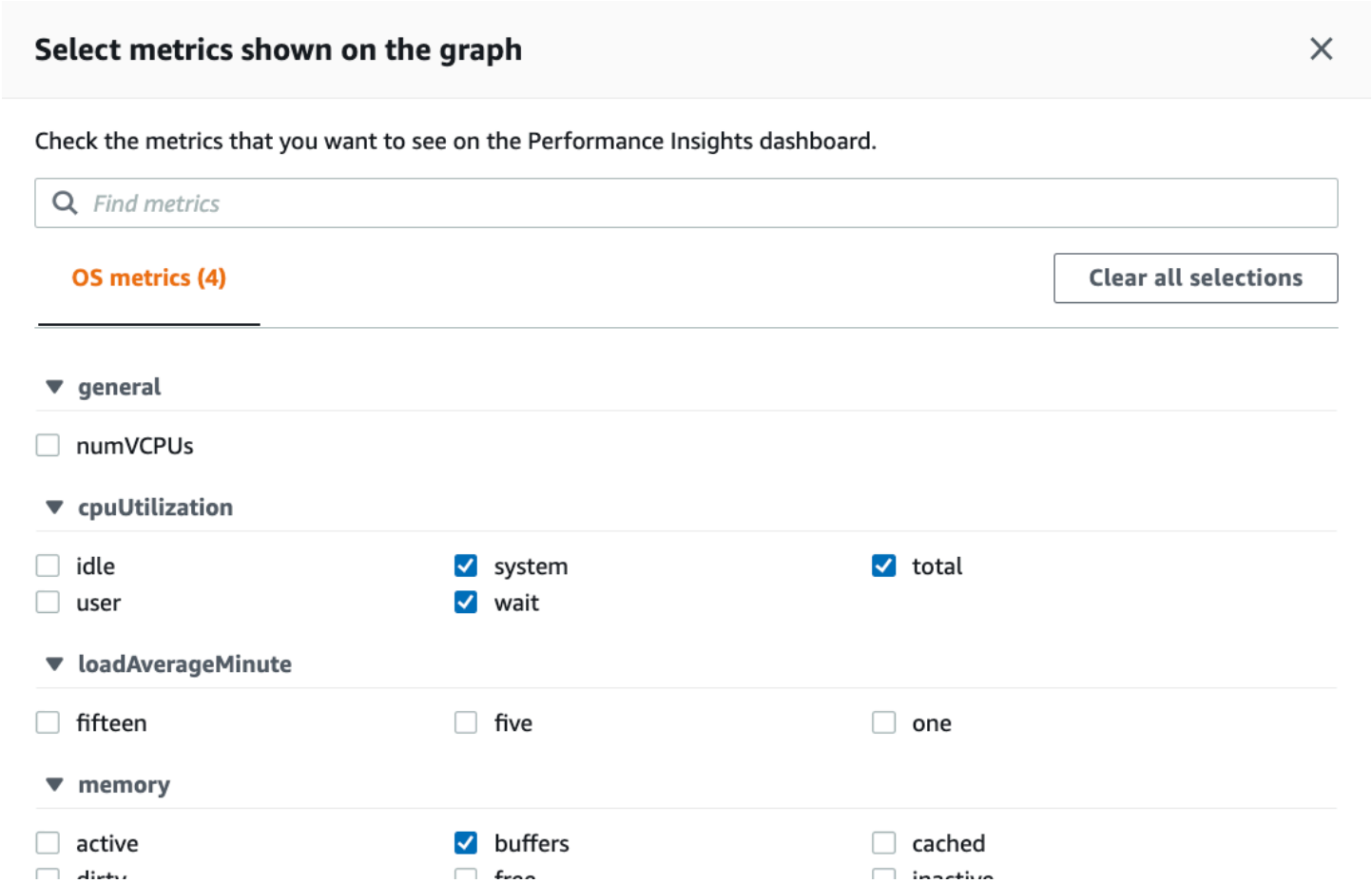
カウンターメトリクスグラフ

カウンターメトリクスを使用すると、Performance Insights ダッシュボードをカスタマイズして最大 10 個の追加グラフを含めることができます。これらのグラフは、数十種類のオペレーティングシステムメトリクスの一部を示しています。この情報をデータベースロードと関連付けることで、パフォーマンスの問題を特定して分析できます。

カウンターメトリクスグラフはパフォーマンスカウンターのデータを表示します。



メトリクスの管理を選択して、パフォーマンスカウンターを変更します。以下のスクリーンショットに示すように、複数の OS メトリクスを選択できます。メトリクスの詳細を表示するには、メトリクス名にカーソルを合わせます。



データベースロードのグラフ

データベースロードは、データベースアクティビティとインスタンス容量の比較結果が 最大 vCPU の折れ線グラフとして表示されます。デフォルトでは、折れ線グラフは DB ロードを単位時間あたりの平均アクティブセッションで表します。DB ロードは、待機状態でスライス (グループ化) されます。



ディメンションでスライスされた DB のロード

サポートされているディメンション別にグループ化された、アクティブなセッションとしてロードを表示するように選択できます。次の図に、Amazon DocumentDB インスタンスのディメンションを示します。

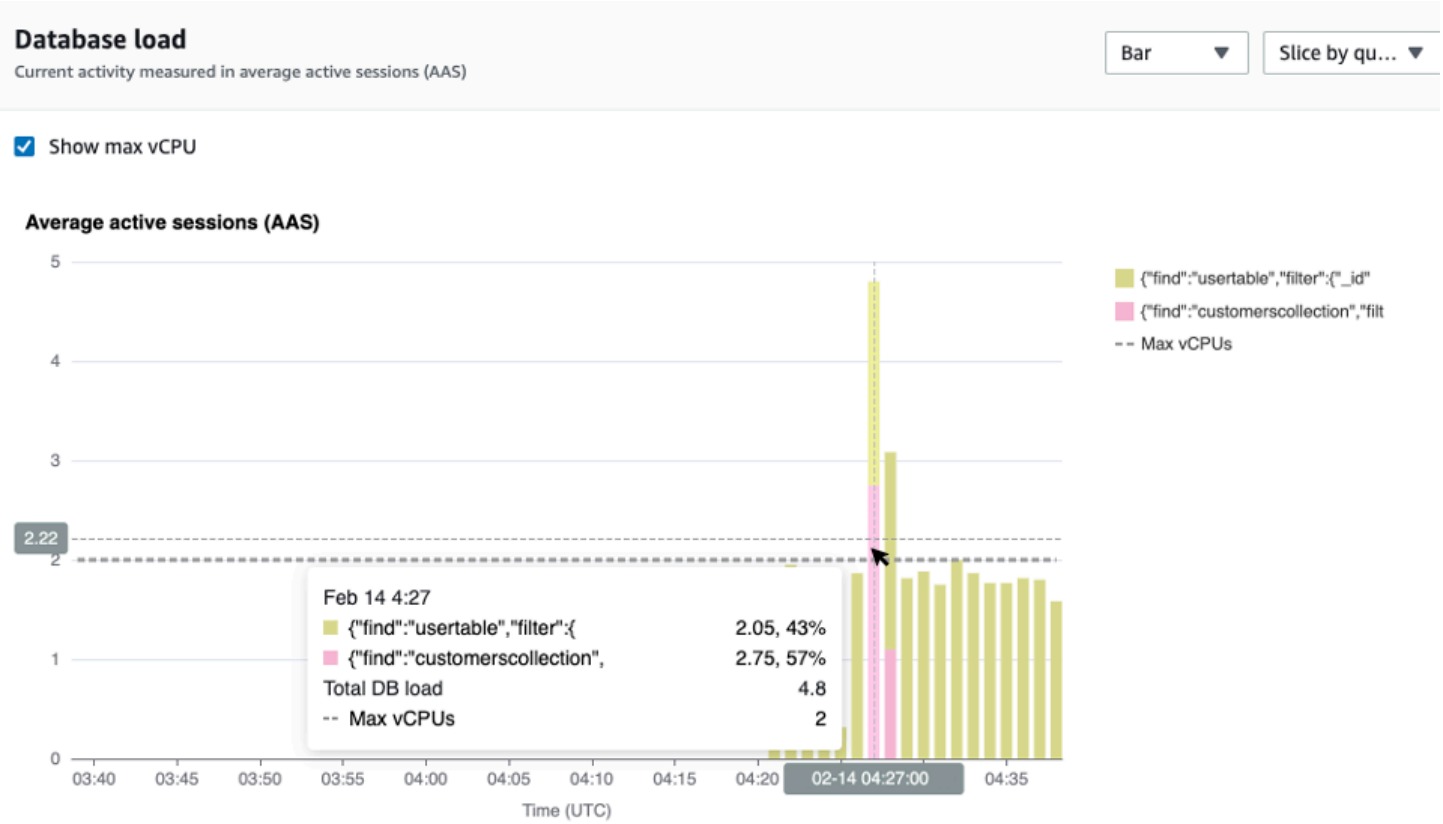


ディメンション項目に関する DB ロードの詳細

ディメンション内の DB ロード項目の詳細を表示するには、項目名にカーソルを合わせます。次の図は、クエリステートメントの詳細を示しています。



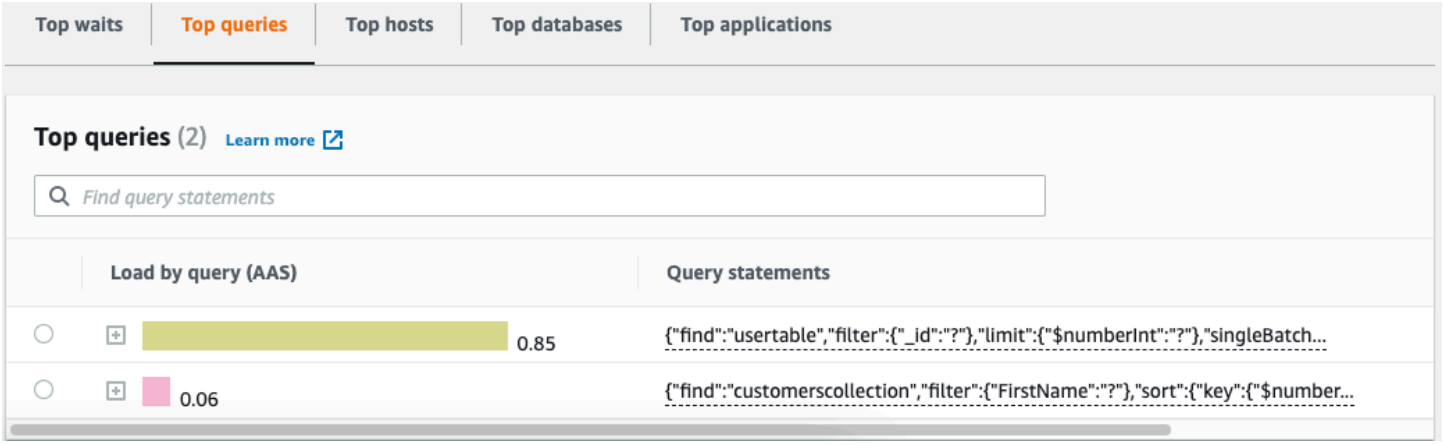
凡例で選択した期間に対する項目の詳細を表示するには、その項目にカーソルを合わせます。



上位のディメンションテーブル

上位ディメンションテーブルは、DB ロードを異なる次元でスライスします。ディメンションとは、DB ロードの異なる特性についてのカテゴリまたは「スライス化」のことです。ディメンションがクエリの場合、上位のクエリは、DB ロードに最も寄与しているクエリステートメントを表示します。

以下のディメンションタブのいずれかを選択します。



次の表では、各タブの簡単な説明を示します。

説明
データベースの上位待機イベントを表示します。
バックエンドが待っているリソースを表示します。

説明
イベント
現在実行中のクエリステートメント

説明
接続されているクライアントのホスト IP

説明
トップ 10 データベースユーザーの名前

説明
データベースユーザーは接続されたアプリケーションの名前。

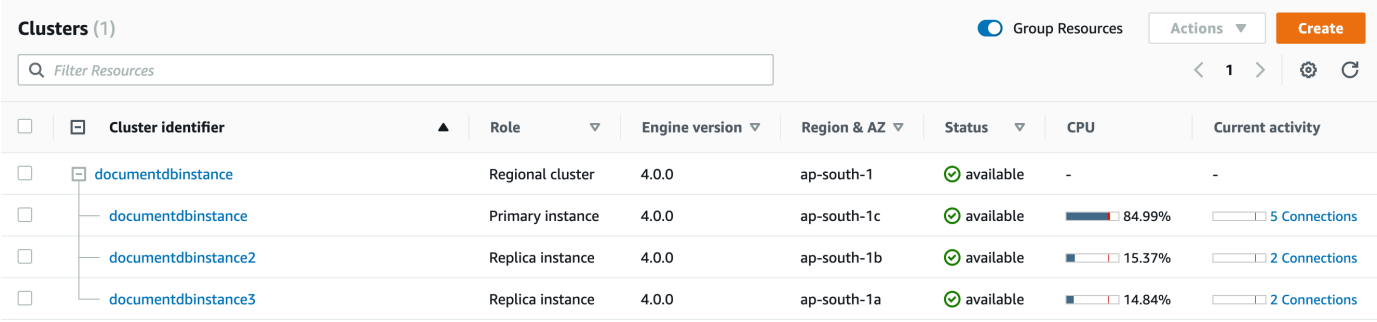
[上位のクエリ] タブを使用してクエリを分析する方法については、「[上位のクエリ] タブの概要」を参照してください。

Performance Insights ダッシュボードを開く

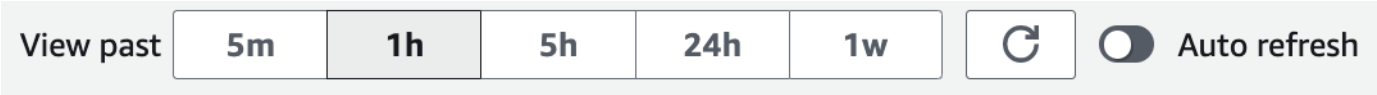
AWS マネジメントコンソールで Performance Insights ダッシュボードを表示するには、次の手順を実行します。

1. https://console.aws.amazon.com/docdb/ で Performance Insights コンソールを開きます。
2. DB インスタンスを選択します。その Amazon DocumentDB インスタンスの Performance Insights ダッシュボードが表示されます。

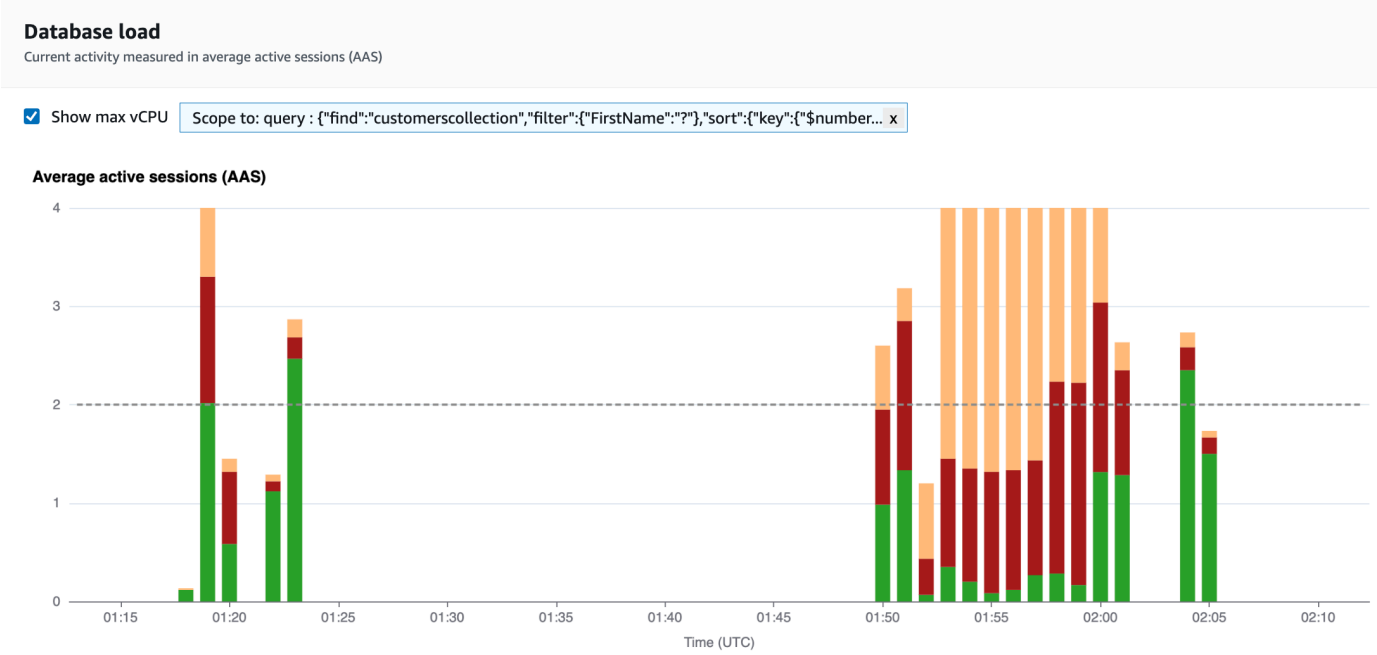
Performance Insights を有効にした Amazon DocumentDB インスタンスでは、インスタンスのリストで [セッション] 項目を選択してダッシュボードにアクセスすることもできます。[現在のアクティビティ] の [セッション] 項目には、直近 5 分間におけるアクティブなセッションの平均データベースロードが表示されます。ロードはバーでグラフィカルに示されます。バーが空の場合、インスタンスはアイドル状態です。ロードが増加すると、バーが青色で塗りつぶされます。ロードがインスタンスクラスの仮想 CPU (vCPU) の数を超えると、バーが赤色になり、ボトルネックとなる可能性があることが示されます。



3. (オプション) 右上のボタンを選択して、別の時間間隔を選択します。例えば、間隔を 1 時間に変更するには、[1 時間] を選択します。



以下のスクリーンショットでは、DB ロードの間隔は 1 時間です。



4. データを自動的に更新するには、[自動更新] を有効にします。



Performance Insights ダッシュボードが自動的に新しいデータで更新されます。更新の頻度は、表示されるデータの量によって異なります。

- 「5 分」は 5 秒ごとに更新されます。
- 「1 時間」は 1 分ごとに更新されます。
- 「5 時間」は 1 分ごとに更新されます。
- 「24 時間」は 5 分ごとに更新されます。
- 「1 週間」は 1 時間ごとに更新されます。

待機状態によるデータベースロードの分析

データベースロード (DB ロード) のグラフにボトルネックが表示される場合、ロードの発生源を確認できます。これを実行するには、データベースロードグラフ下にある[上位ロード項目] テーブルを参照してください。クエリやアプリケーションのような特定の項目を選択すると、その項目をドリルダウンして詳細を表示できます。

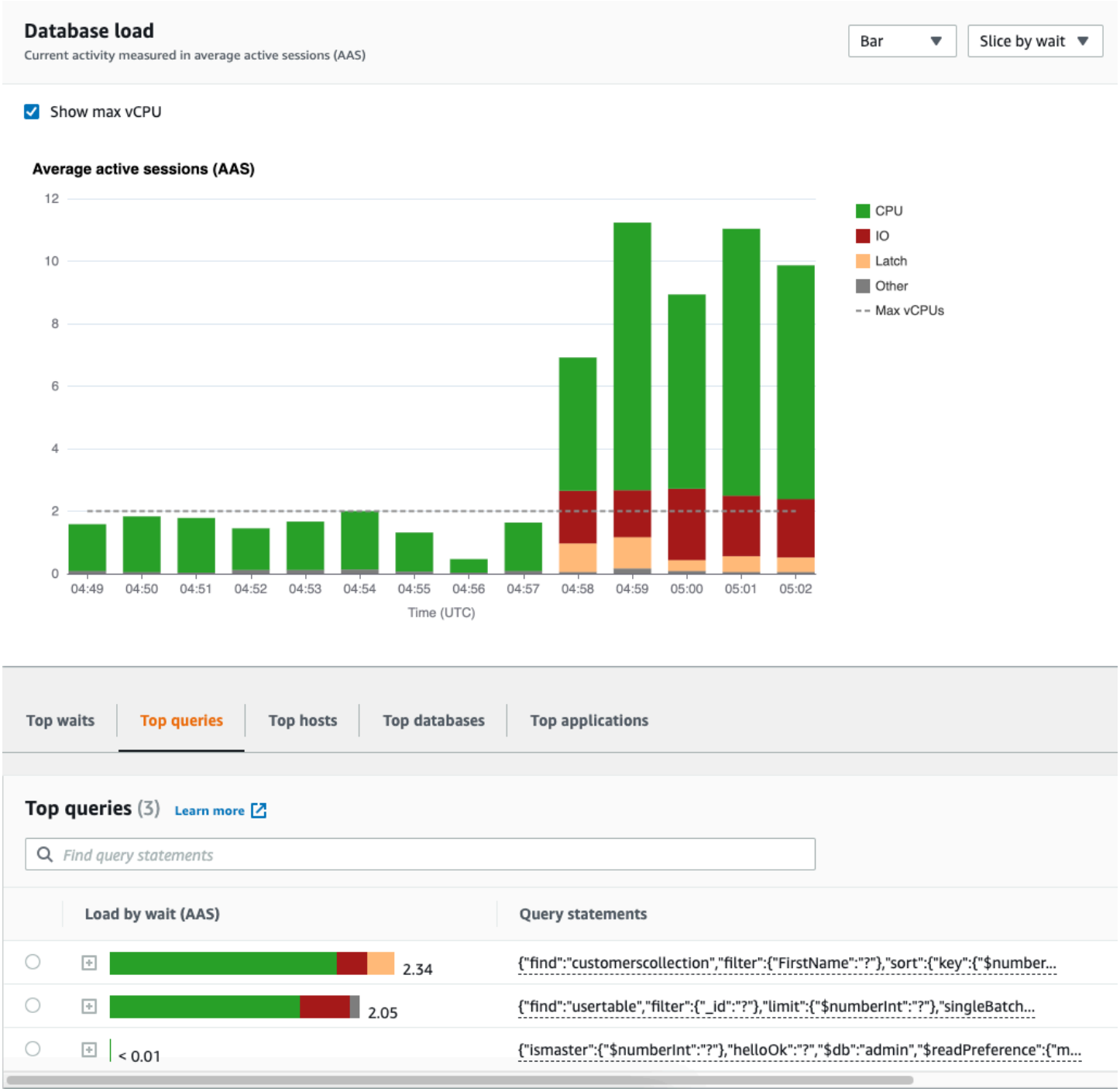
待機および上位クエリによってグループ分けされた DB ロードは、通常、パフォーマンス問題に関する最も正しい情報を提供します。待機でグループ化された DB ロードは、データベースにリソースまたは同時のボトルネックがあるかどうかを示します。この場合、上位ロード項目のテーブルの [上位のクエリ] タブには、どのクエリがそのロードをかけているかが表示されます。

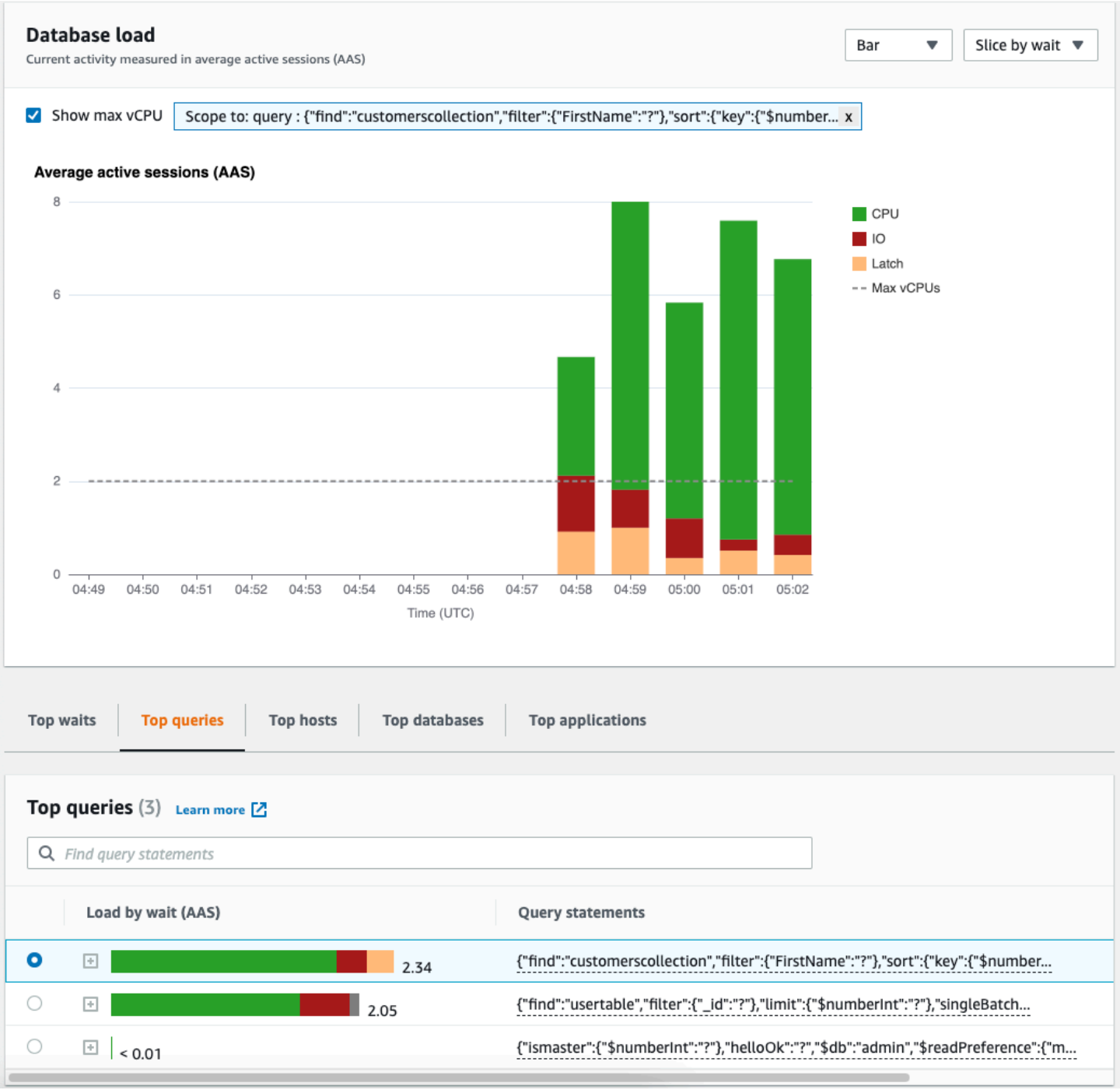
パフォーマンスの問題を診断するための一般的なワークフローは次のとおりです。

1. 「データベースロード」 グラフを確認し、最大 CPU ラインを超えているデータベースロードのインシデントがあるかどうかを確認します。
2. ある場合は、「データベースロード」 グラフを確認して、どの待機状態 (複数) が主に原因であるかを特定します。
3. 上位のロード項目テーブルの [上位のクエリ] タブが待機状態に最も影響しているクエリを確認することによって、ロードを引き起こすダイジェストクエリを特定します。これらは [待機別ロード (AAS)] 列で識別できます。
4. [上位のクエリ] タブでこれらのダイジェストクエリの 1 つを選択して展開し、構成されている子クエリを確認します。

また、[上位のホスト] または [上位のアプリケーション] をそれぞれ選択することで、どのホストまたはアプリケーションが最もロードを発生させているかを確認することもできます。アプリケーション名は Amazon DocumentDB インスタンスへの接続文字列で指定されます。Unknown は、アプリケーションフィールドが指定されなかったことを示します。

例えば、以下のダッシュボードで、CPU の待機はほとんどの DB ロードの主な原因となっています。[上位のクエリ] で上位クエリを選択すると、選択クエリによって発生しているロードが最も大きいことに焦点を当てるようにデータベースロードチャートが絞り込まれます。


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...



[上位のクエリ] タブの概要

デフォルトでは、[上位のクエリ] タブはデータベースロードに最も貢献しているクエリを表示します。クエリテキストを分析して、クエリの調整に役立てることができます。

トピック

- クエリダイジェスト

- 待機によるロード (AAS)
- 詳細なクエリ情報を表示します
- ステートメントのクエリテキストへのアクセス
- ステートメントのクエリテキストの表示とダウンロード

クエリダイジェスト

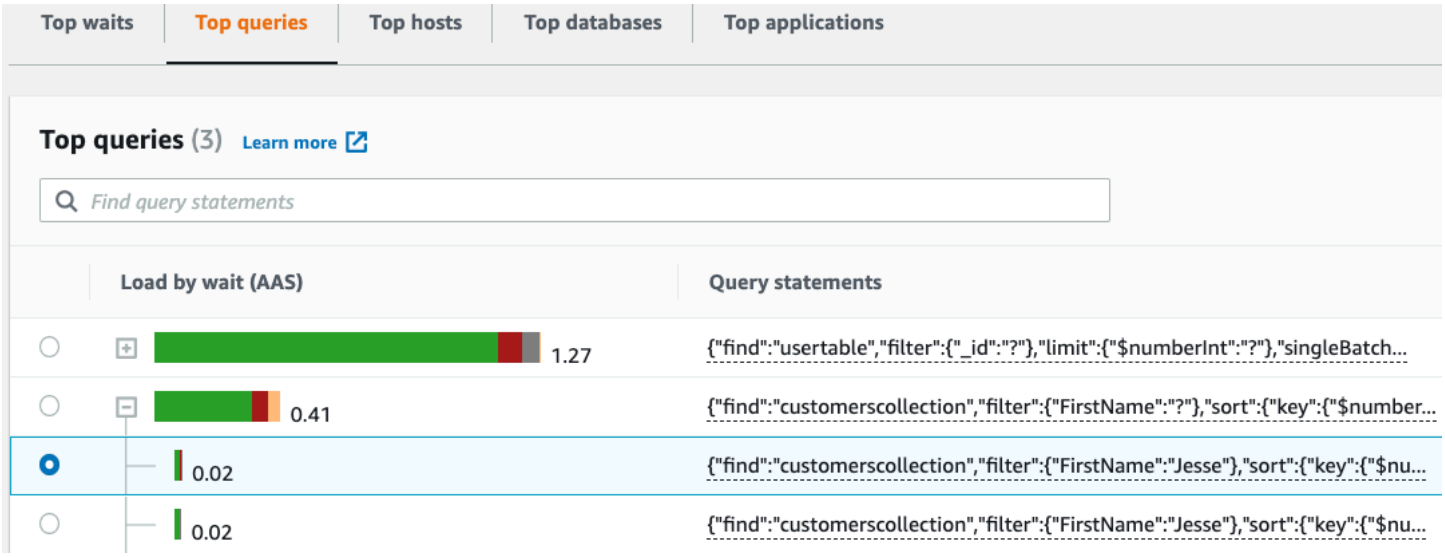
クエリダイジェスト は、構造的には類似しているが、異なるリテラル値を含む可能性の高い、複数の実際のクエリの複合体です。ダイジェストは、ハードコードされた値を疑問符に置き換えます。クエリダイジェストの例は次のようになります。

```
{"find":"customerscollection","filter":{"FirstName":"?"},"sort":{"key":
{"$numberInt":"?"}},"limit":{"$numberInt":"?"}}
```

このダイジェストには、次の子クエリが含まれます。

```
{"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"}}
```

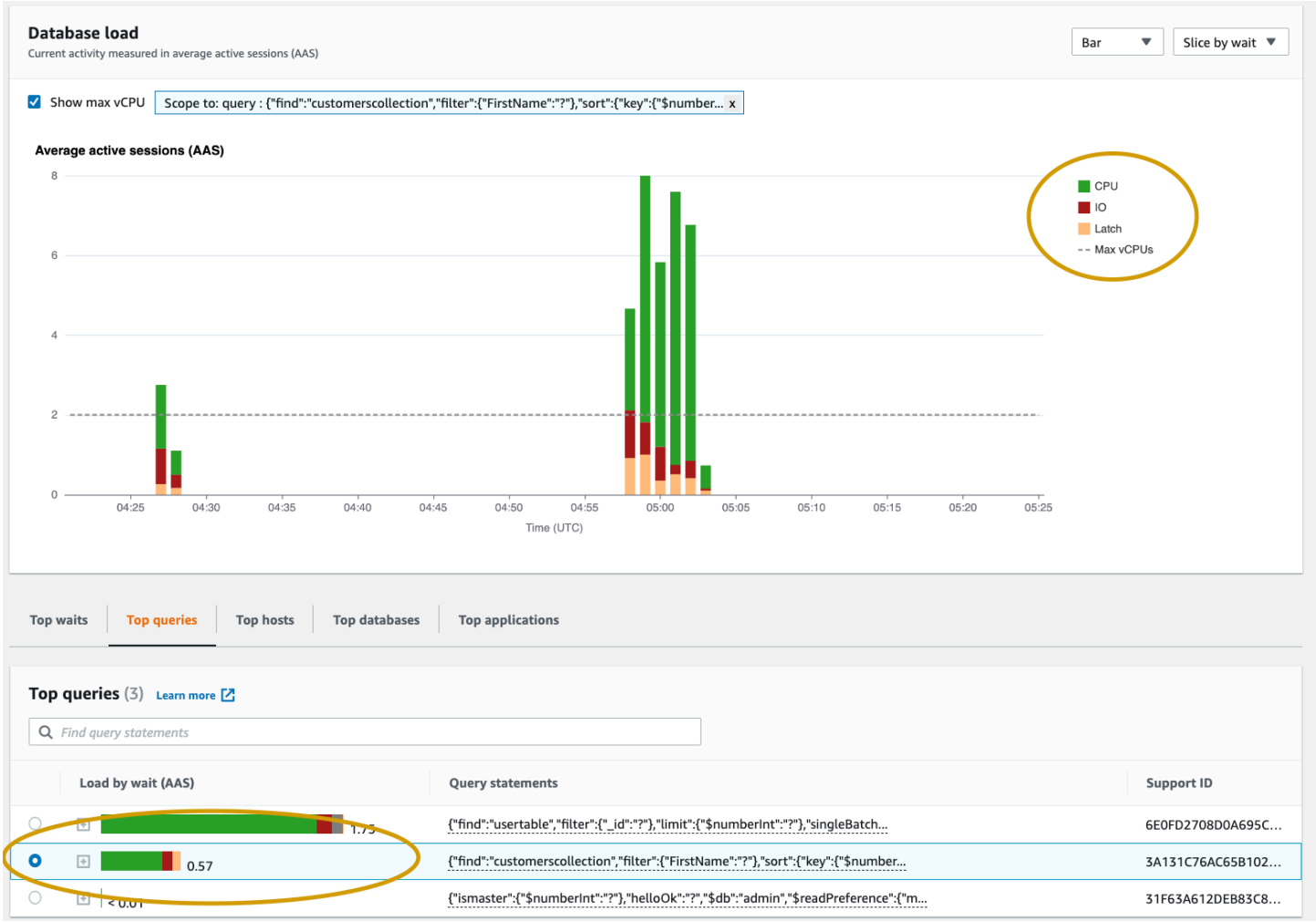
ダイジェスト内でリテラルクエリステートメントを表示するには、クエリを選択してからプラス記号 (+) を選択します。以下のスクリーンショットでは、選択されたクエリはダイジェストです。



> **Note**
>
> クエリダイジェストでは、類似したクエリステートメントがグループ化されますが、機密情報は編集されません。

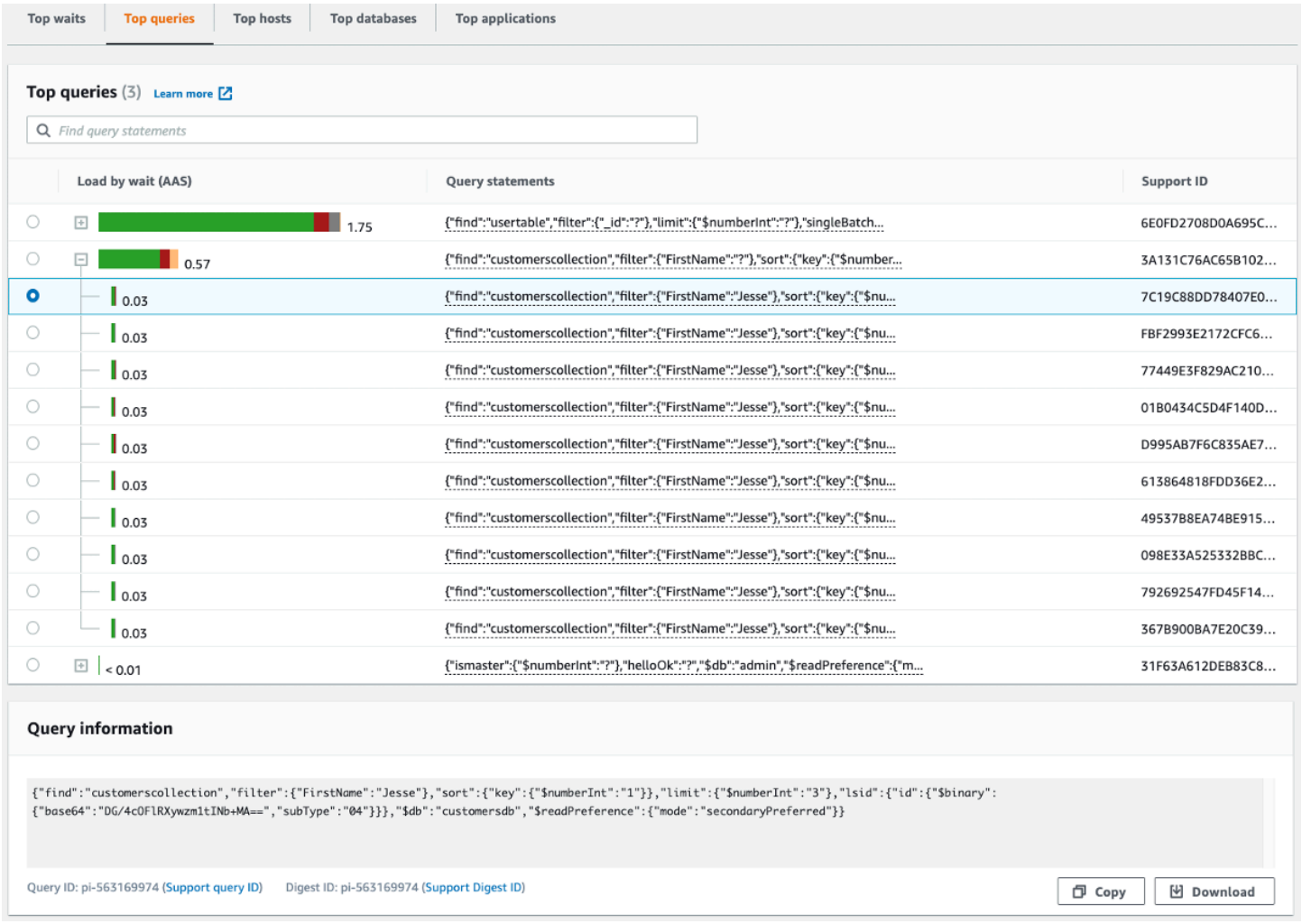
待機によるロード (AAS)

[上位のクエリ] の [待機別のロード (AAS)] 列は、上位の各ロード項目に関連付けられているデータベースロードの割合を示しています。この列には、[DB ロードグラフ] で現在選択されているグループ化に応じて、その項目に対するロードが反映されます。例えば、DB ロードグラフを待機状態別にグループ化できます。この場合は、[待機別の DB ロード] バーは、クエリが貢献している待機状態の量を示すために、サイズ、セグメント、および色で分けられています。また、選択したクエリに影響を与えている待機状態も示されます。



詳細なクエリ情報を表示します

[上位のクエリ] テーブルで、ダイジェストステートメントを開いてその情報を表示できます。下部のペインに情報が表示されます。



クエリステートメントに関連付けられているタイプの識別子 (ID) は以下のとおりです。

1. サポートクエリ ID: クエリ ID のハッシュ値。この値は、 AWS Support で作業しているときにクエリ ID を参照するためのものです。 AWS Support は実際のクエリ IDsとクエリテキストにアクセスできません。
2. ダイジェスト ID のサポート - ダイジェスト ID のハッシュ値。この値は、 AWS Support で作業しているときにダイジェスト ID を参照するためのものです。 AWS Support は実際のダイジェスト IDsとクエリテキストにアクセスできません。

ステートメントのクエリテキストへのアクセス

デフォルトでは、[上位のクエリ] テーブルの各行には各クエリステートメントごとに 500 バイトのクエリテキストが表示されます。ダイジェストステートメントのサイズが 500 バイトを超える場合、Performance Insights ダッシュボードでこのステートメントを開いて、テキストの表示量を増やすことができます。この場合、表示されるクエリの最大長は 1 KB です。フルクエリステートメントを表示する場合は、[ダウンロード] を選択することもできます。

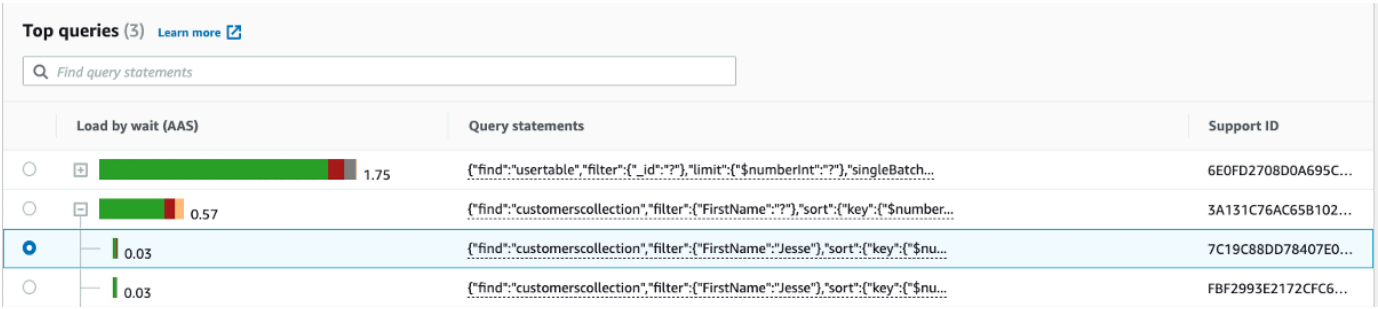
ステートメントのクエリテキストの表示とダウンロード

Performance Insights ダッシュボードで、クエリテキストを表示およびダウンロードできます。

Performance Insights ダッシュボードでクエリテキストの表示量を増やすには

1. Amazon DocumentDB コンソールを、次の場所で開きます。https://console.aws.amazon.com/docdb
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。この DB インスタンスに Performance Insights ダッシュボードが表示されます。

   500 バイトを超えるクエリステートメントは、次のように表示されます。



4. クエリ情報セクションを調べて、クエリテキストをさらに表示します。



Performance Insights ダッシュボードは、各フルクエリステートメントで最大 1 KB までを表示できます。

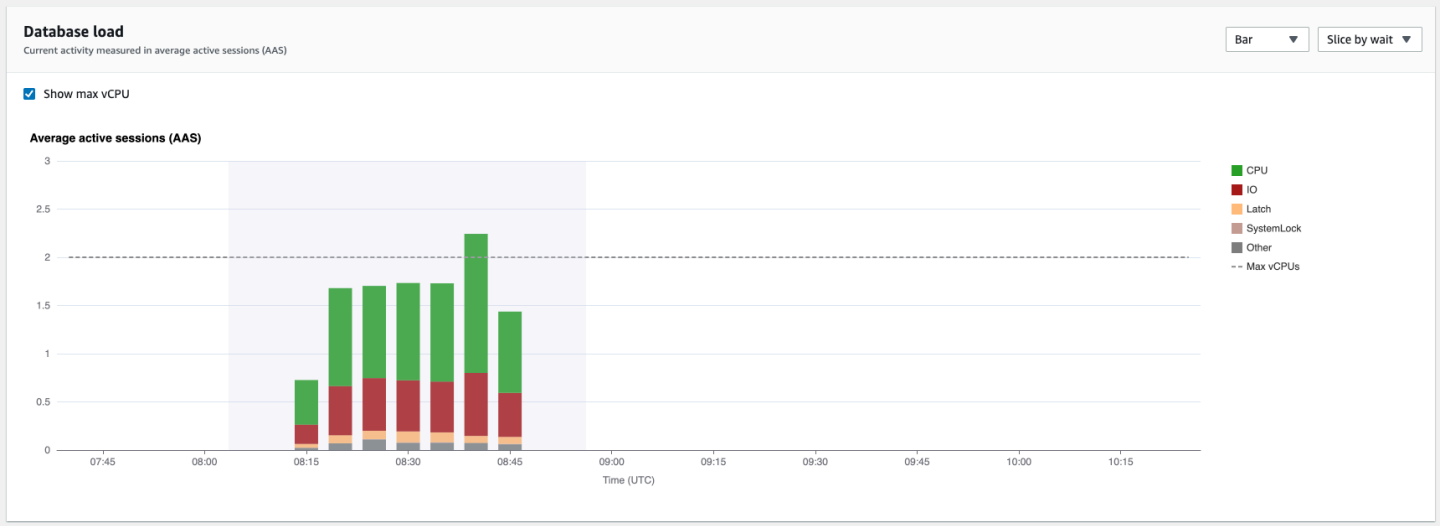
> **Note**
>
> クエリステートメントをコピーまたはダウンロードするには、ポップアップブロッカーを無効にします。

データベースロードグラフのズームイン

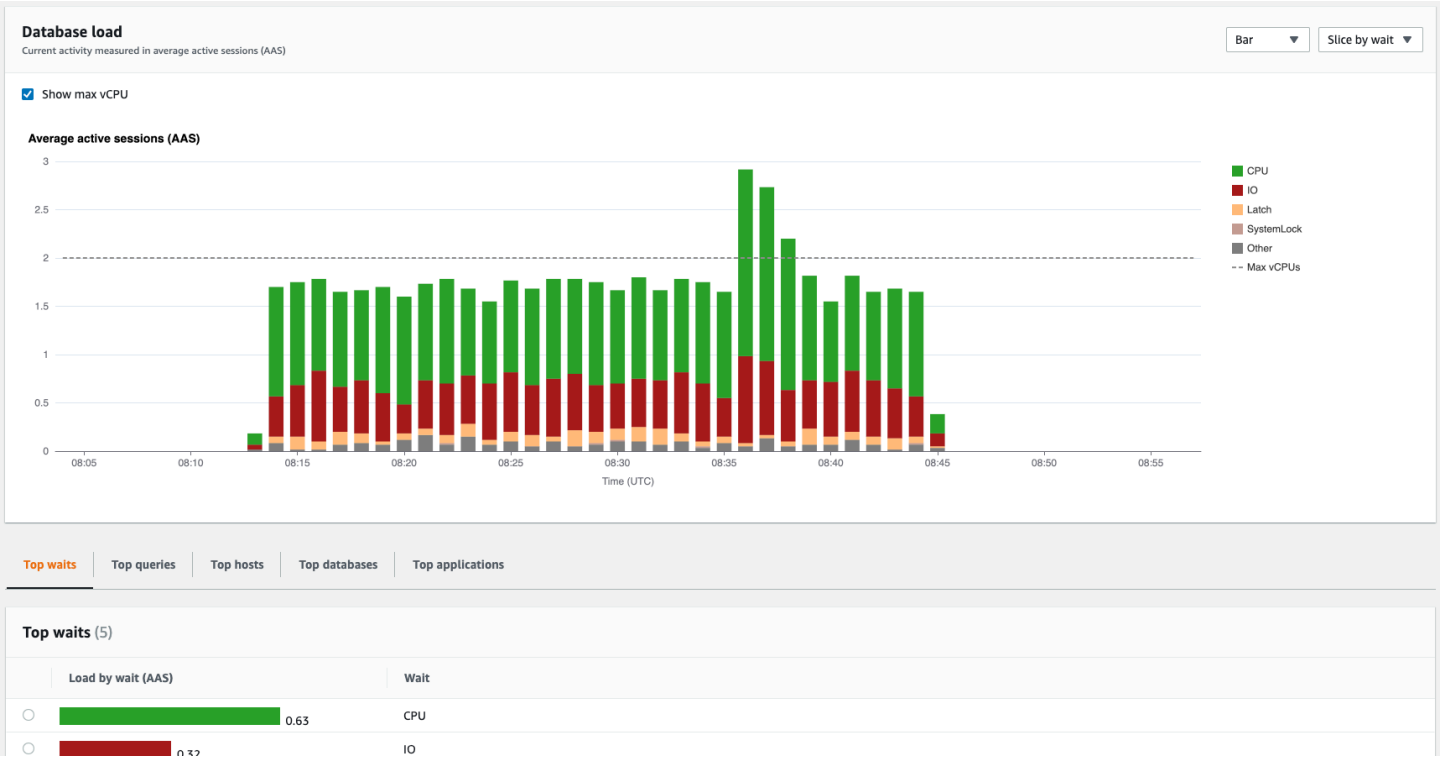
Performance Insights のユーザーインターフェイスの他の機能を使用して、パフォーマンスデータを分析できます。

クリックアンドドラッグしてズームイン

Performance Insights のインターフェイスでは、ロードグラフの小さな部分を選択して、詳細をズームインすることができます。



ロードグラフの一部を拡大するには、スタート時間を選択し、目的の期間の最後までドラッグします。これを行うと、選択した領域が強調表示されます。マウスを離すと、選択したエリアのロードグラフがズームインされ、上位項目 テーブルが再計算されます。



Performance Insights API によるメトリクスの取得

Performance Insights が有効になっている場合、API はインスタンスのパフォーマンスを可視化します。Amazon CloudWatch Logs は、 AWS サービスの供給モニタリングメトリクスの信頼できるソースを提供します。

Performance Insightsは、平均アクティブ・セッション(AAS)として測定されるデータベースロードのドメイン固有のビューを提供します。このメトリクスはAPI利用者には2次元時系列データセットのように見えます。データの時間ディメンションは、クエリされた時間範囲内の各時点のDBロード・データを提供します。各時点で、その時点で計測された `Query`、`Wait-state`、`Application`、`Host` などのリクエストされたディメンションに関するロード全体が分解されます。

Amazon DocumentDB Performance Insights では、Amazon DocumentDB DB インスタンスをモニタリングし、データベースパフォーマンスの分析とトラブルシューティングを行うことができます。Performance Insights は、 AWS マネジメントコンソールで表示することができます。また、Performance Insights では独自のデータをクエリできるように、パブリック API も提供されています。API を使用して、次を実行できます。

- データベースにデータをオフロードする
- Performance Insights データを既存のモニタリングダッシュボードに追加する

- モニタリングツールを構築する

Performance Insights API を使用するには、いずれかの Amazon DocumentDB インスタンスで Performance Insights を有効にします。Performance Insights の有効化については、「Performance Insights の有効化と無効化」を参照してください。Performance Insights API の詳細については、「パフォーマンスインサイト API リファレンス」を参照してください。

Performance Insights API は、以下のオペレーションを提供します。

Performance Insights でのアクション	AWS CLI コマンド	説明
DescribeDimensionKeys	aws pi describe-dimension-keys	特定の期間に、メトリクスの上位 N 個のディメンションキーを取得します。
GetDimensionKeyDetails	aws pi get-dimension-key-details	DB インスタンスまたはデータソースの指定されたディメンショングループの属性を取得します。例えば、クエリ ID を指定し、ディメンションの詳細が使用可能な場合、GetDimensionKeyDetails は、この ID に関連付けられているディメンション db.query.statement の全文を取得します。このオペレーションは、GetResourceMetrics および DescribeDimensionKeys が大きなクエリステートメントテキストの取得をサポートしないため、便利です。
GetResourceMetadata	aws pi get-resource-metadata	さまざまな機能に関するメタデータを取得します。例え

Performance Insights でのアクション	AWS CLI コマンド	説明
		ば、メタデータにより、特定の DB インスタンスで何等かの機能が有効化されているか無効化されているかを、示すことができます。
GetResourceMetrics	aws pi get-resource-metrics	期間中、データソースのセットに Performance Insights のメトリクスを取得します。特定のディメンショングループおよびディメンションを提供し、各グループの集約とフィルタリング条件を提供することができます。
ListAvailableResourceDimensions	aws pi list-available-resource-dimensions	指定したインスタンスで、指定したメトリクスタイプごとにクエリできるディメンションを取得します。
ListAvailableResourceMetrics	aws pi list-available-resource-metrics	DB インスタンスを指定しながら、指定されたメトリクスタイプでクエリが可能なメトリクスをすべて取得します。

トピック

- AWS CLI Performance Insights 用
- 時系列メトリクスの取得
- AWS CLI Performance Insights の例

AWS CLI Performance Insights 用

Performance Insights は、 AWS CLIを使用して表示することができます。Performance Insights の AWS CLI コマンドのヘルプを表示するには、コマンドラインで次のように入力します。

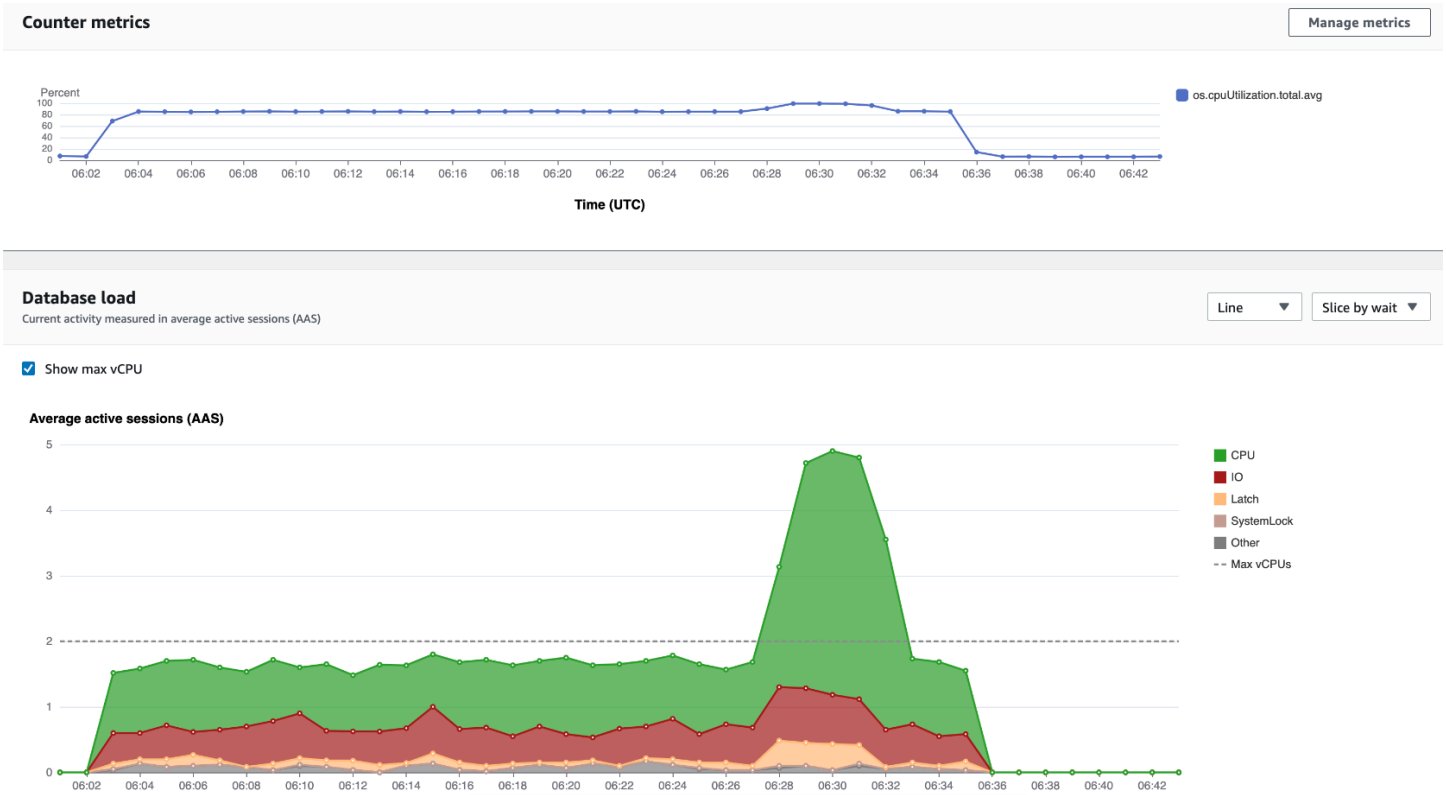
```
aws pi help
```

AWS CLI がインストールされていない場合は、「 ユーザーガイド」の AWS 「 コマンドラインインターフェイスのインストール」を参照してください。 AWS CLI

時系列メトリクスの取得

GetResourceMetrics オペレーションでは、1 つ以上の時系列メトリクスを Performance Insights データから取得します。GetResourceMetrics には、メトリクスおよび期間が必要であり、データポイントのリストを含むレスポンスが返ります。

たとえば、次の図に示すように、 AWS マネジメントコンソール は GetResourceMetricsを使用してカウンターメトリクスグラフとデータベースロードグラフを入力します。



GetResourceMetrics によって返るメトリクスはすべて、db.load の例外を除き、スタンダードの時系列メトリクスです。このメトリクスは、[データベースロード] グラフに表示されます。この db.load メトリクスは、ディメンションと呼ばれるサブコンポーネントに分割できるため、他の時

系列メトリクスとは異なります。前のイメージでは、db.load は分割され、db.load を構成する待機状態によってグループ化されています。

> **Note**
>
> GetResourceMetrics は、db.sampleload メトリクスを返すこともできますが、通常 db.load メトリクスが適切です。

GetResourceMetrics により返されるカウンターメトリクスに関する情報は、「カウンターメトリクス用の Performance Insights」を参照してください。

以下の計算は、メトリクスにサポートされています。

- 平均 - 期間中のメトリクスの平均値。.avg をメトリクス名に追加します。
- 最小 - 期間中のメトリクスの最小値。.min をメトリクス名に追加します。
- 最大 - 期間中のメトリクスの最大値。.max をメトリクス名に追加します。
- 合計 - 期間中のメトリクス値の合計。.sum をメトリクス名に追加します。
- サンプル数 - 期間中にメトリクスが収集された回数。.sample_count をメトリクス名に追加します。

例えば、メトリクスが 300 秒 (5 分) 収集され、メトリクスが 1 分に 1 回収集されたものと見なします。毎分の値は、1、2、3、4、5 です。この場合、以下の計算が返されます。

- 平均 - 3
- 最小 - 1
- 最大 - 5
- 合計 - 15
- サンプル数 - 5

get-resource-metrics AWS CLI コマンドの使用の詳細については、「」を参照してくださいget-resource-metrics。

--metric-queries オプションでは、結果を取得する 1 つ以上のクエリを指定します。各クエリは、必須の Metric と、オプションの GroupBy および Filter パラメータから構成されます。--metric-queries オプションの指定の例を次に示します。

```
{
   "Metric": "string",
   "GroupBy": {
     "Group": "string",
     "Dimensions": ["string", ...],
     "Limit": integer
   },
   "Filter": {"string": "string"
     ...}
```

AWS CLI Performance Insights の例

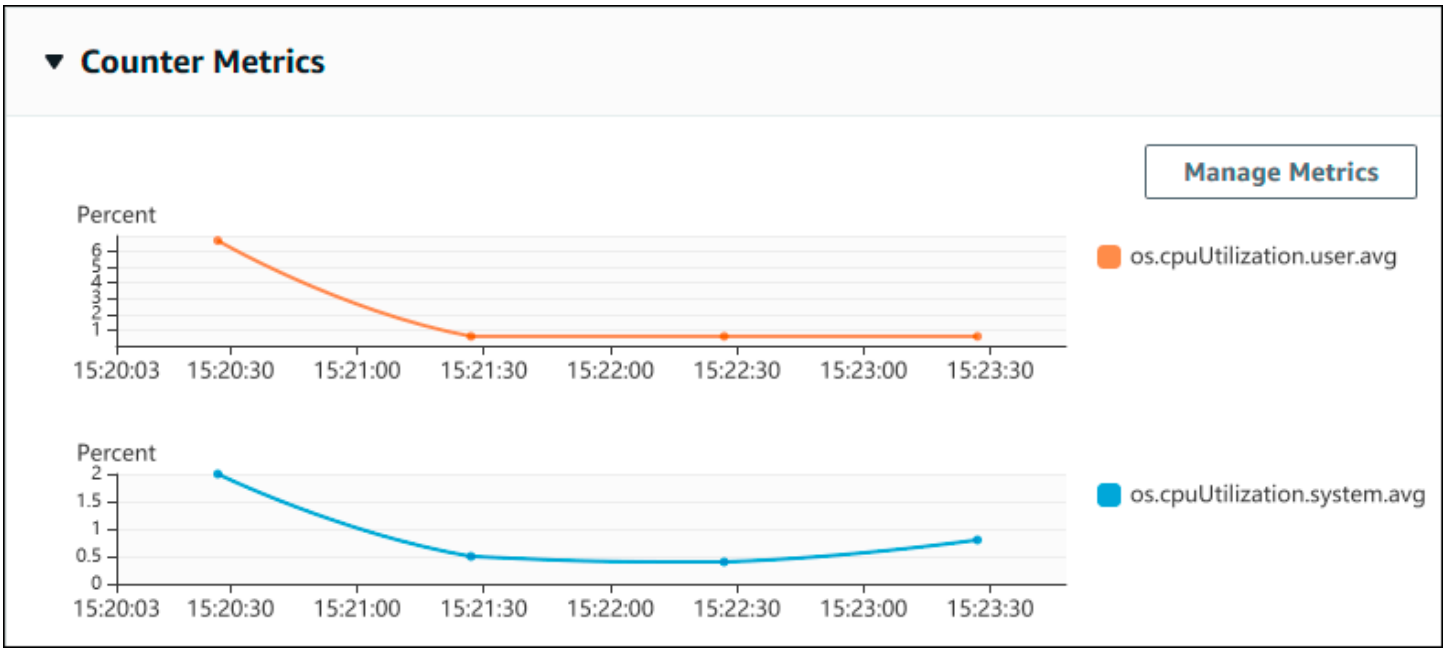
次の例は、Performance Insights AWS CLI に を使用する方法を示しています。

トピック

- カウンターメトリクスの取得
- 上位の待機状態に関する DB 平均ロードの取得
- 上位のクエリに関する DB 平均ロードの取得
- クエリによってフィルタリングされた平均 DB ロードの取得

カウンターメトリクスの取得

以下のスクリーンショットは、 AWS マネジメントコンソールにおける 2 つのカウンターメトリクスグラフを示します。



以下の例では、2 つのカウンターメトリクスグラフを生成するために AWS マネジメントコンソールで使用するデータと同じデータを生成する方法を示します。

Linux、macOS、Unix の場合:

```
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"}]'
```

Windows の場合:

```
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"}]'
```

また、コマンドを作成しやすくするために、`--metrics-query` オプションにファイルを指定します。以下の例では、このオプション用に query.json と呼ばれるファイルを使用します。ファイルの内容は次のとおりです。

```
[
    {
        "Metric": "os.cpuUtilization.user.avg"
    },
    {
        "Metric": "os.cpuUtilization.idle.avg"
    }
]
```

ファイルを使用するには、次のコマンドを実行します。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

前述の例では、各オプションに次の値を指定します。

- `--service-type`: Amazon DocumentDB の DOCDB
- `--identifier` - DB インスタンスのリソース ID
- `--start-time` および `--end-time` - クエリを実行する期間の ISO 8601 `DateTime` 値 (サポートされている複数の形式)

クエリは 1 時間の範囲で実行されます。

- `--period-in-seconds` - `60` (1 分ごとのクエリ)
- `--metric-queries` - 2 つのクエリの配列。それぞれ 1 つのメトリクスに対して使用されます。

  メトリクス名ではドットを使用してメトリクスを有用なカテゴリに分類します。最終の要素は関数になります。この例では、関数は、クエリの `avg` です。Amazon CloudWatch と同様に、サポートされている関数は、`min`、`max`、`total`、および `avg` です。

レスポンスは次の例のようになります。

```
{
    "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
```

レスポンスには、Identifier、AlignedStartTime、AlignedEndTime があります。--period-in-seconds 値が 60 の場合、スタート時間および終了時間は、時間 (分) に調整されます。--period-in-seconds が 3600 の場合、スタート時間および終了時間は、時間 (時) に調整されます。

レスポンスの MetricList には、多数のエントリを含み、それぞれに Key および DataPoints エントリがあります。DataPoint にはそれぞれ、Timestamp および Value を含みます。ク

エリは 1 分ごとのデータが 1 時間以上実行されるため、Datapoints の各リストには、60 個のデータポイントがあります。これには、Timestamp1/Minute1 や Timestamp2/Minute2 から、Timestamp60/Minute60 まで含まれます。

クエリは 2 つの異なるカウンターメトリクスを対象としているため、レスポンス MetricList には 2 つの要素があります。

上位の待機状態に関する DB 平均ロードの取得

次の例は、 が積み上げ面折れ線グラフの生成 AWS マネジメントコンソール に使用するクエリと同じです。この例では、上位 7 つの待機状態に応じてロードを分割し、最後の 1 時間で db.load.avg を取得します。コマンドは カウンターメトリクスの取得 と同じコマンドです。ただし、query.json ファイルには、次の内容が含まれます。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_state", "Limit": 7 }
    }
]
```

次のコマンドを実行します。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

この例では、上位 7 つの待機状態のうち db.load.avg と GroupBy のメトリクスを指定しています。この例の有効な値の詳細については、Performance Insights の API リファレンスの「DimensionGroup」を参照してください。

レスポンスは次の例のようになります。

```
{
    "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
```

このレスポンスでは、MetricList の 3 つのエントリがあります。合計の db.load.avg のエントリが 1 つあり、上位 3 つの待機イベントのいずれかに従って分割された db.load.avg のエントリが 3 つあります。(最初の例とは異なり) グループ化ディメンションがあったため、メトリクスのグループ化ごとに 1 つのキーが必要です。基本的なカウンターメトリクスのユースケースのように、メトリクスごとに 1 つのキーのみ使用することはできません。

上位のクエリに関する DB 平均ロードの取得

以下の例では、上位 10 個のクエリステートメント別に db.wait_state をグループ化します。クエリステートメントには 2 つの異なるグループがあります。

- db.query - フルクエリステートメント (例: {"find":"customers","filter":{"FirstName":"Jesse"},"sort":{"key":{"$numberInt":"1"}}})
- db.query_tokenized - トークン化されたクエリステートメント (例: {"find":"customers","filter":{"FirstName":"?"},"sort":{"key":{"$numberInt":"?"}},"limit":{"$numberInt":"?"}})

データベースのパフォーマンスを分析するときは、パラメータが異なるだけのクエリステートメントを 1 つの論理的な項目として検討すると便利です。そのため、クエリを実行する際、db.query_tokenized を使用することができます。ただし、特に explain( ) に関心がある場合は、パラメータ付きのフルクエリステートメントを調べる方が便利な場合があります。トークン化されたクエリと完全クエリの間には親子関係があり、複数の完全クエリ (子) が同じトークン化されたクエリ (親) の下にグループ化されています。

この例のコマンドは、上位の待機状態に関する DB 平均ロードの取得 のコマンドに似ています。ただし、query.json ファイルには、次の内容が含まれます。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.query_tokenized", "Limit": 10 }
    }
]
```

次の例では db.query_tokenized を使用しています。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

この例では、1 分の間隔 (秒単位) で 1 時間以上のクエリを実行します。

この例では、上位 7 つの待機状態のうち db.load.avg と GroupBy のメトリクスを指定しています。この例の有効な値の詳細については、Performance Insights の API リファレンスの「DimensionGroup」を参照してください。

レスポンスは次の例のようになります。

```
{
    "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
```

このレスポンスの MetricList には 11 のエントリがあり (合計が 1 つと、トークン化された上位 10 項目のクエリ)、各エントリには、1 時間あたり 24 の DataPoints があります。

トークン化されたクエリの場合は、各ディメンションリストに 3 つのエントリがあります。

- db.query_tokenized.statement: トークン化されたクエリステートメント。
- db.query_tokenized.db_id : Performance Insights が生成する合成 ID。この例では、pi-1064184600 合成 ID が返ります。
- db.query_tokenized.id - Performance Insights 内のクエリの ID。

  では AWS マネジメントコンソール、この ID はサポート ID と呼ばれます。ID は、データベースの問題のトラブルシューティングに役立つように AWS サポートが調査できるデータであるため、この名前が付けられます。 はデータのセキュリティとプライバシーを非常に真剣に AWS 受け止め、ほぼすべてのデータが で暗号化されて保存されます AWS KMS key。したがって、内

部の誰もこのデータを見る AWS ことはできません。前の例では、tokenized.statement と tokenized.db_id の両方が暗号化されて保存されます。データベースに問題がある場合は、AWS サポート ID を参照してサポートにお問い合わせください。

クエリを実行する際、Group で GroupBy を指定した方が便利な場合があります。ただし、返るデータを詳細に制御できるように、ディメンションのリストを指定します。例えば、必要なデータが db.query_tokenized.statement のみの場合は、Dimensions 属性を query.json ファイルに追加することができます。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": {
            "Group": "db.query_tokenized",
            "Dimensions":["db.query_tokenized.statement"],
            "Limit": 10
        }
    }
]
```

クエリによってフィルタリングされた平均 DB ロードの取得

この例に対応する API クエリは、上位のクエリに関する DB 平均ロードの取得 のコマンドに似ています。ただし、query.json ファイルには、次の内容が含まれます。

```
[
 {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_state", "Limit": 5  },
        "Filter": { "db.query_tokenized.id": "AKIAIOSFODNN7EXAMPLE" }
    }
]
```

このレスポンスでは、query.json ファイルで指定されているトークン化されたクエリ AKIAIOSFODNN7EXAMPLE の割合に従って、値はすべてフィルタリングされます。キーは、フィルタなしのクエリとは異なる順序で表示されることもあります。これは、フィルタ処理されたクエリに影響を与えるのは上位 5 つの待機クエリであるためです。

Performance Insights の Amazon CloudWatch メトリクス

Performance Insights はメトリクスを自動的に Amazon CloudWatch に発行します。Performance Insights から同じデータに対してクエリを実行できますが、CloudWatch にメトリクスを含めると、CloudWatch アラームを追加しやすくなります。また、既存の CloudWatch ダッシュボードにメトリクスを追加しやすくなります。

メトリクス	説明
DBLoad	Amazon DocumentDB のアクティブセッション数。通常、アクティブセッションの平均数に関するデータを使用します。Performance Insights で、このデータは `db.load.avg` としてクエリされます。
DBLoadCPU	待機状態タイプが CPU であるアクティブセッションの数。Performance Insights で、このデータは、待機状態タイプ `db.load.avg` でフィルタ処理された `CPU` としてクエリされます。
DBLoadNonCPU	待機状態タイプが CPU でないアクティブセッションの数。

> **Note**
>
> これらのメトリクスは、DB インスタンスにロードがある場合にのみ CloudWatch に公開されます。

これらのメトリクスは、CloudWatch コンソール、 AWS CLI、または CloudWatch API を使用して調べることができます。

例えば、`DBLoad` メトリクスの統計情報は、get-metric-statistics コマンドを実行して取得できます。

```
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
```

次のコマンドでは、以下のような出力が生成されます。

```
{
    "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"
```

```
}
```

CloudWatch コンソールの DB_PERF_INSIGHTS メトリクス数学関数を使用して Amazon DocumentDB にクエリを実行し、Performance Insights カウンターメトリクスを取得できます。DB_PERF_INSIGHTS 関数には、1 分未満の間隔での DBLoad メトリクスも含まれます。これらのメトリクスに基づいた CloudWatch アラームを設定することができます。アラームの作成方法の詳細については、AWS 「データベースから Performance Insights カウンターメトリクスにアラームを作成する」を参照してください。

CloudWatch の詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch とは」を参照してください。

カウンターメトリクス用の Performance Insights

カウンターメトリクスは、Performance Insights ダッシュボードのオペレーティングシステムメトリクスのことです。カウンターメトリクスを DB ロードと関連付けることで、パフォーマンスの問題を特定して分析できます。

Performance Insights オペレーティングシステムのカウンター

Amazon DocumentDB のパフォーマンスインサイトでは、以下のオペレーティングシステムカウンターを利用できます。

Counter	タイプ	メトリクス
active	メモリ	os.memory.active
buffers	メモリ	os.memory.buffers
キャッシュ済み	メモリ	os.memory.cached
dirty	メモリ	os.memory.dirty
free	メモリ	os.memory.free
無効	メモリ	os.memory.inactive
mapped	メモリ	os.memory.mapped

Counter	タイプ	メトリクス
pageTables	メモリ	os.memory.pageTables
slab	メモリ	os.memory.slab
total	メモリ	os.memory.total
writeback	メモリ	os.memory.writeback
idle	cpuUtilization	os.cpuUtilization.idle
system	cpuUtilization	os.cpuUtilization.system
total	cpuUtilization	os.cpuUtilization.total
ユーザー	cpuUtilization	os.cpuUtilization.user
によって	cpuUtilization	os.cpuUtilization.wait
one	loadAverageMinute	os.loadAverageMinute.one
fifteen	loadAverageMinute	os.loadAverageMinute.fifteen
5	loadAverageMinute	os.loadAverageMinute.five
キャッシュ済み	スワップ	os.swap.cached
free	スワップ	os.swap.free
が	スワップ	os.swap.in
out	スワップ	os.swap.out
total	スワップ	os.swap.total
rx	ネットワーク	os.network.rx
tx	ネットワーク	os.network.tx
numVCPUs	全般	os.general.numVCPUs

Amazon DocumentDB Serverless の使用

Amazon DocumentDB Serverless は、アプリケーションの需要に基づいて Amazon DocumentDB データベースの容量を動的に調整するプロセスを自動化するオンデマンドの自動スケーリング設定です。クラスターが消費するリソースに対してのみ課金されます。このように、DocumentDB サーバーレスによって予算内に収め、使用しないコンピュータリソースに対して料金を支払うことを避けるのに役立ちます。

この種の自動化は、マルチテナントデータベース、分散型データベース、開発、テストシステムなど、要求が厳しく予測不可能なワークロードがある環境において特に有効です。

トピック

- DocumentDB サーバーレスのユースケース
- Amazon DocumentDB Serverless の利点
- Amazon DocumentDB Serverless の仕組み
- DocumentDB サーバーレスの要件と制限
- Amazon DocumentDB Serverless を使用するクラスターの作成
- Amazon DocumentDB Serverless への移行
- Amazon DocumentDB Serverless の管理
- Amazon DocumentDB Serverless インスタンスの制限
- Amazon DocumentDB Serverless スケーリング設定
- Amazon DocumentDB Serverless のモニタリング

DocumentDB サーバーレスのユースケース

Amazon DocumentDB プロビジョニング済みクラスターと DocumentDB サーバーレスはどちらも、開発環境やテスト環境から、高いスケールと可用性を必要とする最も要求の厳しいビジネスクリティカルなアプリケーションまで、さまざまなタイプのデータベースワークロードをサポートします。ただし、DocumentDB サーバーレスは、お客様のワークロードに別のディメンションを追加します。つまり、予測不可能なワークロードを持つウェブサイトやアプリケーションをサポートする機能です。

DocumentDB サーバーレスは、以下のユースケースに特に役立ちます。

- 変動するワークロード - 突然で予測不可能なアクティビティの増加が発生するようなワークロードを実行している場合。雨が降り出したときにアクティビティが急増 (サージ) するトラフィックサイトなどが該当します。別のケースとして、販売や特別なプロモーションを行うことで、トラフィックが増加する e コマースサイトがあります。DocumentDB サーバーレスでは、アプリケーションのピーク時に必要なロードに合わせて、データベースの容量がオートスケーリングされ、アクティビティのサージが終了した時点でスケールダウンして元に戻ります。DocumentDB サーバーレスを導入することで、ピーク容量や平均容量に合わせてプロビジョニングする必要はなくなります。最悪の状況に対応するために容量の上限を指定でき、その容量は必要な場合以外使用されません。
  - DocumentDB サーバーレスのスケーリングの詳細度は、データベースのニーズに合わせて容量を細かく調整しやすくします。プロビジョン済みクラスターの場合、スケールアップには、完全に新しいインスタンスを追加する必要があります。DocumentDB サーバーレスは、必要な容量が少ししかない場合に DCU の半分を追加できます。ワークロードの増加に対応するために追加が必要な容量によって、0.5、1、1.5、2、または半分の DCU を追加できます。また、ワークロードが減少し、その容量が不要になった場合、0.5、1、1.5、2、または追加した半分の DCU を削除できます。
- マルチテナントアプリケーション — DocumentDB サーバーレスでは、フリート内の各アプリケーションのデータベース容量を個別に管理する必要はありません。DocumentDB サーバーレスは、個々のデータベース容量を管理します。
  - テナントごとにクラスターを作成できます。これにより、クローンやスナップショットリストアなどの機能を使用して、テナントごとに高可用性とディザスタリカバリを強化できます。
  - 各テナントには、時間帯、時期、プロモーションイベントなどに応じて、繁忙期と休止期間が設定される場合があります。各クラスターには、広い範囲で容量を指定できます。これにより、アクティビティの少ないクラスターではインスタンスの料金を最小限に抑えることができます。どのクラスターでも、アクティビティの高い期間に対応できるように迅速にスケールアップできます。
- 新しいアプリケーション — 現在デプロイ中で、必要とされるインスタンスサイズが明確でない、新しいアプリケーション。DocumentDB サーバーレスを使用して、1 つまたは複数のインスタンスでクラスターを設定し、アプリケーションの容量の要件に応じてデータベースをオートスケーリングできます。
- 複合用途のアプリケーション — オンライントランザクション処理 (OLTP) アプリケーションを使用しているが、クエリトラフィックが定期的に急増することがある場合。クラスター内の DocumentDB サーバーレスインスタンスに昇格階層を指定することで、リーダーインスタンスがライターインスタンスと独立してスケーリングして、追加のロードを処理できるようにクラスター

を構成できます。使用率の急増が収まったら、リーダーインスタンスによってライターインスタンスの容量に合わせてスケールダウンします。

- 容量計画 — 通常、クラスター内のすべてのインスタンスのインスタンスクラスを変更して、データベース容量を調整するか、ワークロードに最適なデータベース容量を検証します。DocumentDB サーバーレスでは、この管理オーバーヘッドを回避できます。ワークロードを実行し、インスタンスが実際にスケールする量をチェックすることで、適切な最小容量と最大容量を決定できます。
  - 既存のインスタンスを、プロビジョン済みから DocumentDB サーバーレスに、または DocumentDB サーバーレスからプロビジョン済みに変更できます。このような場合、新しいクラスターや新しいインスタンスを作成する必要はありません。
- 開発とテスト — 最も要求の厳しいアプリケーションの実行に加えて、開発環境やテスト環境にも DocumentDB サーバーレスを使用できます。DocumentDB サーバーレスにより、バースト db.t* インスタンスクラスを使用する代わりに、最小容量が小さいインスタンスを作成できます。最大容量を大きく設定することで、これらのインスタンスのメモリが不足せず、大量のワークロードを実行できます。データベースが使用されていない場合は、すべてのインスタンスがスケールダウンすることで、不要な料金が発生しないようにします。

既存のプロビジョニングされたワークロードに Amazon DocumentDB Serverless を使用する

プロビジョン済みクラスターで既に DocumentDB アプリケーションが実行されているとします。リーダーインスタンスとして、1 つまたは複数の DocumentDB サーバーレスインスタンスを追加することにより、DocumentDB サーバーレスによってアプリケーションがどのように動作するか確認できます。リーダーインスタンスのスケールアップとスケールダウンの頻度を確認できます。DocumentDB フェイルオーバーメカニズムを使用して、DocumentDB サーバーレスインスタンスをライターに昇格させ、読み取り/書き込みワークロードの処理方法を確認できます。これにより、クライアントアプリケーションが使用するエンドポイントを変更することなく、最小限のダウンタイムで切り替えが可能です。既存のクラスターを DocumentDB サーバーレスに変換する手順の詳細については、「Amazon DocumentDB Serverless への移行」を参照してください。

Amazon DocumentDB Serverless の利点

DocumentDB サーバーレスは、可変または「スパイキー」ワークロードを対象としています。このような予測不可能なワークロードでは、データベース容量を変更するタイミングを計画するのが難しい場合があります。また、容量を迅速に変更するために、インスタンスの追加やインスタンスクラスの変更など、使い慣れたメカニズムでは十分ではない場合があります。DocumentDB サーバーレスには、このようなユースケースに役立つ以下の利点があります。

- プロビジョニングよりも簡単な容量管理 — DocumentDB サーバーレスは、ワークロードの変化に応じてインスタンスのサイズを計画したりインスタンスのサイズを変更したりするための労力を削減します。また、クラスター内のすべてのインスタンスの容量を一定に維持するための労力が削減されます。

- 高アクティビティ時のスケーリングを高速かつ簡単に実行 — DocumentDB サーバーレスは、クライアントトランザクションやワークロード全体を中断することなく、必要に応じてコンピューティング性能とメモリ容量をスケーリングします。DocumentDB サーバーレスでリーダーインスタンスを使用できることで、垂直スケーリングに加え、水平スケーリングも利用できます。DocumentDB グローバルクラスターを使用する機能は、DocumentDB サーバーレス読み取りワークロードを複数の に分散できることを意味します AWS リージョン。この機能は、プロビジョン済みクラスターのスケーリングメカニズムよりも便利です。

- アクティビティが少ない期間におけるコスト効率 — DocumentDB サーバーレスは、インスタンスのオーバープロビジョニングを回避するのに役立ちます。DocumentDB サーバーレスは、インスタンスのスケールアップ時にリソースをきめ細かく追加します。使用したデータベースリソースに対してのみ料金を支払います。DocumentDB サーバーレスリソースの使用量は、秒単位で測定されます。これにより、インスタンスがスケールダウンすると、削減されたリソース使用量がすぐに登録されます。

- プロビジョニングされた機能パリティ — DocumentDB サーバーレスですべての DocumentDB 機能を使用できます。例えば、DocumentDB サーバーレスでは、リーダーインスタンス、 AWS Identity and Access Management (IAM) データベース認証、Performance Insights を使用できます。

  特に DocumentDB サーバーレスでは、プロビジョン済みクラスターによって、以下機能を活用できます。

  - リーダーインスタンス — DocumentDB サーバーレスでは、リーダーインスタンスを活用して水平方向にスケーリングできます。クラスターに 1 つまたは複数のリーダーインスタンスが含まれている場合、ライターインスタンスに問題が発生した場合に、クラスターはすぐにフェイルオーバーできます。
  - マルチ AZ クラスター — クラスターの DocumentDB サーバーレスインスタンスは、複数のアベイラビリティーゾーン (AZ) に分散できます。マルチ AZ クラスターを設定することで、AZ 全体に影響する問題が発生するようなまれなケースでも、ビジネスの継続性を確保できます。

- グローバルクラスター — DocumentDB グローバルクラスターと組み合わせて DocumentDB サーバーレスを使用すると、ディザスタリカバリ AWS リージョン の目的で、他の でクラスターの追加の読み取り専用コピーを作成できます。

Amazon DocumentDB Serverless の仕組み

トピック

- 概要
- Amazon DocumentDB クラスターの設定
- Amazon DocumentDB Serverless スケーリング容量
- Amazon DocumentDB Serverless スケーリング
- アイドル状態 (0.5 DCU)

概要

Amazon DocumentDB Serverless は、最も変化が激しく、要求の厳しいワークロードに適しています。用途の例としては、データベースの使用負荷が短時間の間だけ増大し、その後に軽いアクティビティが長時間続くか、またはアクティビティがまったく発生しなくなるケースが挙げられます。例えば、定期的に販売促進イベントを行う小売り、ゲーム、スポーツなどのウェブサイト、必要なときにレポートを作成するレポートデータベースなどがあります。また、開発やテスト環境、また、新しいアプリケーションでは急激に利用が増加することがあります。他にも多くが考えられますが、このようなケースに対してプロビジョニングされたモデルを使用しても、事前に容量を正しく指定できるとは限りません。また、過剰なプロビジョニングを行い、使用しない容量が生じた場合には、コストが高くなる可能性もあります。

一方で、プロビジョン済み DocumentDB クラスターは、安定したワークロードに適しています。プロビジョン済みクラスターでは、メモリサイズ、CPU パワー、I/O 帯域幅などが事前定義されたインスタンスクラスを選択します。ワークロードが変更された場合、ライターとリーダーのインスタンスクラスを手動で変更します。プロビジョン済みモデルは、消費パターンが予想され、事前に容量を調整できる場合に有効です。クラスター内のライターとリーダーのインスタンスクラスを変更しながら、短時間の停止の発生が許容される場合は有効に機能します。

DocumentDB サーバーレスは、拡張が瞬時に行われるサーバーレスクラスターをサポートするために、ゼロから設計されました。DocumentDB サーバーレスは、プロビジョニングされたライターやリーダーと同じレベルのセキュリティと分離を提供するように設計されています。このような側面は、マルチテナントのサーバーレスクラウド環境では非常に重要です。データベースのワークロード

の変更に迅速に対応できるように、動的スケーリングメカニズムにはオーバーヘッドがほとんどありません。また、処理需要の劇的な増加に対応するための、十分な能力も備わっています。

DocumentDB サーバーレスを使用することにより、各ライターとリーダーにある特定のデータベース容量の制約を受けずに DocumentDB クラスターを作成できます。お客様は、最小容量と最大容量の範囲を指定します。DocumentDB では、その容量範囲内のクラスター内の各 DocumentDB サーバーレスライターまたはリーダーをスケーリングします。各ライターまたはリーダーによって動的にスケーリングできるマルチ AZ クラスターを使用することで、動的スケーリングと高可用性を活用できます。

DocumentDB サーバーレスでは、最小容量と最大容量の仕様に基づいて、データベースリソースを自動的にスケーリングします。ほとんどのスケーリングイベントのオペレーションは、ライターまたはリーダーが同じホスト上で保持されるため、高速にスケーリングします。DocumentDB サーバーレスライターまたはリーダーが、あるホストから別のホストに移動するまれなケースでも、DocumentDB サーバーレスでは自動的に接続を管理します。データベースクライアントアプリケーションのコードやデータベースの接続文字列を変更する必要はありません。

DocumentDB サーバーレスでは、プロビジョン済みクラスターと同様に、ストレージ容量とコンピューティング性能は別々になっています。DocumentDB サーバーレス容量やスケーリングに言及した場合、増減するのは常にコンピューティング性能です。したがって、CPU やメモリの容量がスケールダウンしても、クラスターには数テラバイトのデータを格納できます。

プロビジョニングやデータベースサーバーを管理する代わりに、データベース容量を指定します。各 DocumentDB サーバーレスライターまたはリーダーの実際の容量は、ワークロードによって時間とともに変化します。このメカニズムの詳細については、「Amazon DocumentDB Serverless スケーリング」を参照してください。

Amazon DocumentDB クラスターの設定

Amazon DocumentDB サーバーレスクラスターごとに、DocumentDB サーバーレスの容量およびプロビジョン済み容量、またはその両方を自由に組み合わせて選択できます。

混在設定クラスターと呼ばれる DocumentDB サーバーレスとプロビジョン済み容量の両方を含むクラスターを設定できます。例えば、DocumentDB サーバーレスライターで利用可能な容量よりも、多くの読み取り/書き込み容量が必要だとします。この場合、非常に大きいプロビジョン済みライターを使用してクラスターをセットアップできます。その後も、リーダーに DocumentDB サーバーレスを使用できます。また、クラスターの書き込みワークロードは変化しているが、読み取りワークロードは安定しているとします。この場合、クラスターに 1 つの DocumentDB サーバーレスライターと 1 つまたは複数のプロビジョン済みリーダーをセットアップできます。

DocumentDB サーバーレスによってすべての容量が管理されるクラスターをセットアップすることもできます。これを行うには、新しいクラスターを作成し、最初から DocumentDB サーバーレスを使用します。または、DocumentDB サーバーレスで既存のクラスター内でプロビジョン済みのすべての容量を置き換えることができます。DocumentDB サーバーレスで新しいクラスターを作成するか、または既存のクラスターを DocumentDB サーバーレスに切り替える手順の詳細については、「Amazon DocumentDB Serverless を使用するクラスターの作成」および「Amazon DocumentDB Serverless への移行」を参照してください。

クラスターで DocumentDB サーバーレスをまったく使用しない場合、クラスター内のすべてのライターとリーダーはプロビジョン済みになります。これは、ほとんどのユーザーがよく知っている、最も一般的な種類のクラスターです。プロビジョン済み容量は一定です。料金は比較的簡単に予測できます。ただし、必要な容量を事前に予測する必要があります。場合によっては、予測が不正確だったり、容量のニーズが変わったりすることもあります。このような場合、クラスターがプロビジョニングされない (希望よりも遅い)、またはオーバープロビジョニング (必要以上に高価) になる可能性があります。

Amazon DocumentDB Serverless スケーリング容量

Amazon DocumentDB Serverless の測定単位は、DocumentDB キャパシティーユニット (DCU) です。DocumentDB サーバーレススケーリング容量は、プロビジョニングされたクラスターに使用するインスタンスクラスに関連付けられていません。

各 DCU では、約 2 ギビバイト (GiB) のメモリと、対応する CPU、ネットワークが組み合わせられています。この単位を使用して、データベース容量の範囲を指定します。`ServerlessDatabaseCapacity` および `DCUUtilization` CloudWatch メトリクスは、データベースが実際に使用している容量と、その容量が指定された範囲内のどこにあるかを判断するのに役立ちます。

いつどんな場合でも、各 DocumentDB サーバーレスライターまたはリーダーは容量を持っています。容量は、DCU を表す浮動小数点数です。容量は、ライターまたはリーダーがスケーリングするごとに増減します。この値は毎秒測定されます。DocumentDB サーバーレスを使用する予定の各クラスターに、各 DocumentDB サーバーレスライターまたはリーダーの間でスケーリングできる最小容量および最大容量の値である容量範囲を定義します。クラスターの各 DocumentDB サーバーレスライターまたはリーダーでは、容量範囲は同じです。各 DocumentDB サーバーレスライターまたはリーダーは、それぞれの範囲内の容量を持っています。

DocumentDB サーバーレスは、容量範囲が 0.5～256 DCU の DocumentDB 5.0 および 8.0 でサポートされています。 DCUs

定義できる DocumentDB サーバーレスの最小容量は 0.5 DCU です。サポートされている最大容量値以下であれば、それ以上の数値を指定できます。最小容量を小さい量に設定することで、低ロードのクラスターでは最小限のコンピューティングリソースを消費できます。同時に、直ちに接続を受け入れ、ビジーになったらスケールアップする準備ができています。

最小値を、各ライターまたはリーダーがバッファプール内のアプリケーションのワーキングセットを保持できるようにする値に設定します。これにより、アイドル中にバッファプールの内容が破棄されることはありません。スケーリング容量範囲を選択する際のすべての考慮事項については、「DocumentDB サーバーレスクラスターのスケーリング容量範囲の選択」を参照してください。

マルチ AZ 配置でのリーダーの設定方法に応じて、その容量はライターの容量に関連付けることも、独立させることもできます。これを行う方法については、「サーバーレスリーダーの昇格階層の表示と変更」を参照してください。

DocumentDB サーバーレスのモニタリングでは、クラスター内のライターとリーダーの容量値を経時的に測定します。データベースが最小容量にスケールダウンしない場合は、最小値の調整やデータベースアプリケーションの最適化などのアクションを実行できます。データベースが常に最大容量に達している場合は、最大容量を増やすなどのアクションを実行できます。また、データベースアプリケーションを最適化し、クエリのロードをより多くのリーダーに分散させることもできます。

DocumentDB サーバーレス容量の料金は DCU 時間で測定されます。DocumentDB サーバーレス料金の計算方法については、「Amazon DocumentDB の料金」を参照してください。クラスター内のライターとリーダーの合計数が n であるとします。この場合、クラスターは、データベースオペレーションを実行していないときにおよそ n x 最小 DCU を消費します。Amazon DocumentDB 自体がモニタリングやメンテナンスオペレーションを行うことで、わずかなロードがかかることがあります。そのクラスターでは、データベースを全容量で実行しても n x 最大 DCU を超える消費はしません。

適切な最小と最大の DCU 値の選択の詳細については、「DocumentDB サーバーレスクラスターのスケーリング容量範囲の選択」を参照してください。指定する最小と最大の DCU 値は、一部の Amazon DocumentDB インスタンスの制限にも影響します。容量範囲とインスタンスの制限間のやり取りの詳細については、「Amazon DocumentDB Serverless インスタンスの制限」を参照してください。

Amazon DocumentDB Serverless スケーリング

各 DocumentDB サーバーレスのライターやリーダーでは、Amazon DocumentDB は、CPU、メモリ、ネットワークなどのリソースの使用率を継続的に追跡しています。これらの測定値を総称してロードと呼びます。ロードには、アプリケーションによって実行されるデータベースオペレーショ

ンが含まれます。また、データベースサーバーと Amazon DocumentDB 管理タスクのバックグラウンド処理も含まれます。容量がこれらのいずれかによって制約されている場合、DocumentDB サーバーレスはスケールアップします。DocumentDB サーバーレスは、解決できるパフォーマンスの問題を検出した場合も、そのような解決を行うことでスケールアップします。「Amazon DocumentDB Serverless のモニタリング」の手順で、リソース使用率と DocumentDB サーバーレスのスケーリングの影響をモニタリングできます。

ロードは、クラスターのライターとリーダーによって異なる場合があります。ライターはライターオペレーションを処理し、クラスターボリュームに対するすべてのデータ変更を実行します。リーダーは読み取り専用リクエストを処理できます。

スケーリングは、データベースの DocumentDB サーバーレス容量を増減するオペレーションです。DocumentDB サーバーレスでは、各ライターとリーダーには、DCU 単位で測定される独自の現在の容量値があります。DocumentDB サーバーレスは、現在の容量が低すぎて負荷を処理できない場合、ライターまたはリーダーをより高い容量にスケールアップします。ライターまたはリーダーの現在の容量が必要以上に大きい場合、小さい容量にスケールダウンします。

DocumentDB サーバーレスは、容量を段階的に増やすことができます。ワークロードの需要がライターまたはリーダーの現在のデータベース容量に達し始めると、DocumentDB サーバーレスはそのライターまたはリーダーの DCU の数を増やします。DocumentDB サーバーレスは、消費されるリソースで最高のパフォーマンスを実現するために必要なだけの増分で容量をスケーリングします。スケーリングは 0.5 DCU という小さい増分で行われます。現在の容量が大きいほど、スケーリングの増分が大きくなり、そのため、スケーリングがより高速になります。

DocumentDB サーバーレスのスケーリングは高頻度、詳細、無停止であるため、 AWS マネジメントコンソールで離散的なイベントが発生することはありません。代わりに、`serverlessDatabaseCapacity` や `DCUUtilization` のような Amazon CloudWatch メトリクスを測定し、その最小値、最大値、平均値を経時的に追跡できます。DocumentDB サーバーレスのモニタリングの詳細については、「Amazon DocumentDB Serverless のモニタリング」を参照してください。

スケールアップまたはスケールダウンは、以下が原因で発生する可能性があります。

- メモリ使用率
- CPU 使用率
- ネットワーク使用率
- ストレージの使用率

DocumentDB サーバーレスインスタンスのスケールアップ/スケールダウンの原因をモニタリングできます。詳細については、「Amazon DocumentDB Serverless のモニタリング」を参照してください。

リーダースケールは、関連するライターと同時に作成するか、ライターから独立して作成するかを選択できます。これを行うには、そのリーダーの昇格階層を指定します。

- 昇格階層 0 および 1 の DocumentDB サーバーレスのリーダーは、ライターと同じタイミングでスケーリングします。このスケーリング動作によって、優先階層 0 および 1 のリーダーは可用性に最適です。これは、フェイルオーバー時にライターからワークロードを引き継ぐために、常に適切な容量に合わせてサイズ調整されているためです。
- 昇格階層 2 ~ 15 のリーダーは、ライターとは独立してスケーリングできます。各リーダーは、クラスターに指定した DCU の最小値と最大値の範囲内に収まります。リーダーが関連するライター DB とは独立してスケーリングすると、ライターが大量のトランザクションを処理し続けている間に、リーダーがアイドル状態になってスケールダウンする場合があります。下位の昇格階層で他のリーダーが利用できない場合でも、フェイルオーバーターゲットとして利用できます。ただし、ライターに昇格した場合は、ライターのワークロード全体を処理するためにスケールアップが必要になる場合があります。

サーバーレスインスタンスの昇格階層の表示と変更の詳細については、「サーバーレスリーダーの昇格階層の表示と変更」を参照してください。

DocumentDB サーバーレスのスケーリングは、データベース接続が開いている間、トランザクションが進行中の間などに発生する可能性があります。DocumentDB サーバーレスは、スケーリングを開始するためにクワイエットポイントを待機しません。スケーリングによって、進行中のデータベースオペレーションが中断されることはありません。

ワークロードで単一のライターと単一のリーダーで使用できるよりも多くの読み取り容量が必要な場合、クラスターに 複数の DocumentDB サーバーレスのリーダーを追加できます。各 DocumentDB サーバーレスのリーダーは、クラスターに指定した最小から最大の容量値でスケーリングできます。クラスターのリーダーエンドポイントを使用して、読み取り専用のセッションをリーダーに送信し、ライターのロードを軽減できます。

DocumentDB サーバーレスがスケーリングを実行するどうか、また、スケーリング開始後の速度は、クラスターの最小および最大 DCU 設定によって異なります。さらに、リーダーとライターが一緒にスケーリングするように設定されているか、ライターとは独立してスケーリングするように設定されているかによって異なります。スケーリング設定の詳細については、「Amazon DocumentDB Serverless スケーリング設定」を参照してください。

アイドル状態 (0.5 DCU)

Amazon DocumentDB Serverless のライターまたはリーダーがアイドル状態のとき、クラスターの MinCapacity が 0.5 に設定されている場合、DocumentDB サーバーレスインスタンスはアイドル状態の 0.5 DCU へのスケールダウンをサポートします。

アイドル状態では、DocumentDB サーバーレスインスタンスには、ほとんどの本稼働ワークロードをサポートするのに十分な CPU コンピューティング容量はありませんが、新しいワークロードをサポートするために迅速にスケールアップする準備ができています。アイドル状態では、DocumentDB サーバーレスインスタンスには通常、少なくとも 1.0 ~ 2.5 DCU が必要です。したがって、DocumentDB サーバーレスインスタンスがアイドル状態から非アイドル状態にスケールアップすると、1.0 ~ 2.5 DCU (または低い場合は MaxCapacity の値) に直接スケールアップされます。

アイドル状態のときに 0.5 DCU へのスケールダウンをサポートするために、MinCapacity が 1.0 DCU 以下に設定されている場合、インスタンス制限には上限が設定されます。MinCapacity 設定によって制限がどのように影響を受けるかの詳細については、「Amazon DocumentDB Serverless インスタンスの制限」を参照してください。

DocumentDB サーバーレスの要件と制限

機能の要件

リージョンの可用性

Amazon DocumentDB サーバーレスインスタンスタイプが使用可能な AWS リージョン 場所の詳細については、「」の「サーバーレス」列を参照してくださいリージョン別のサポートされるインスタンスクラス。

次の AWS CLI コマンドを使用して、特定のリージョンで提供されている正確な DocumentDB サーバーレスインスタンスオプションを確認できます。

```
aws docdb describe-orderable-db-instance-options \
        --region my_region \
        --db-instance-class db.serverless \
        --engine docdb
```

エンジンバージョンの可用性

DocumentDB サーバーレスは、Amazon DocumentDB 5.0 および 8.0 エンジンバージョンでサポートされています。エンジンバージョン 3.6 または 4.0 では使用できません。

> **Note**
>
> DocumentDB サーバーレスは、5.0.0 の新しいパッチバージョンでのみサポートされています。クラスターが最新のエンジンパッチバージョンに更新されていることを確認します。「Amazon DocumentDB クラスターのメンテナンス」の詳細については、「Amazon DocumentDB のメンテナンス」を参照してください

クラスターの設定

Amazon DocumentDB Serverless インスタンスを Amazon DocumentDB クラスターに追加する前に、クラスターに `ServerlessV2ScalingConfiguration` パラメータも設定されている必要があります。これは、クラスター内の DocumentDB サーバーレスインスタンスの容量範囲を定義します。スケーリング設定の詳細については、「Amazon DocumentDB Serverless スケーリング設定」を参照してください。

特定の Amazon DocumentDB 機能の最小スケーリング容量範囲設定

Amazon DocumentDB 機能の一部は DocumentDB サーバーレスで動作するものもありますが、容量範囲が特定のワークロードの機能に必要なメモリに必要な容量よりも小さい場合は、問題が発生する可能性があります。その場合、通常よりもデータベースのパフォーマンスが低下するか、メモリ不足のエラーが発生する可能性があります。

以下の機能では、最適なオペレーションのためにより高い `MinCapacity` および/または `MaxCapacity` 値を設定する必要があります。

- Performance Insights
- グローバルクラスター (プライマリにのみ適用 AWS リージョン)
- データボリュームが大きいクラスターでのサーバーレスインスタンスの作成

  これには、クラスター復元の一部としてのサーバーレスインスタンスの作成が含まれます。

適切な容量範囲の設定に関する推奨事項 (この機能を使用している場合) については、「DocumentDB サーバーレスクラスターのスケーリング容量範囲の選択」を参照してください。容

量範囲の設定が間違っているためにデータベースでメモリ不足のエラーが発生した場合のトラブルシューティング情報については、「メモリ不足エラーを回避する」を参照してください。

Amazon DocumentDB Serverless を使用するクラスターの作成

Amazon DocumentDB Serverless クラスターの作成

Amazon DocumentDB Serverless では、クラスターはプロビジョン済みクラスターと交換可能です。一部のインスタンスがサーバーレスを使用し、一部のインスタンスがプロビジョニングされているクラスターを持つことができます。

目的のリージョンとエンジンバージョンが DocumentDB サーバーレスをサポートしていることを確認します。「DocumentDB サーバーレスの要件と制限」を参照してください。

サーバーレスインスタンスを追加できる Amazon DocumentDB クラスターを作成するには、Amazon DocumentDB クラスターの作成 と同じ手順に従います。唯一の違いは、`ServerlessV2ScalingConfiguration` 引数も指定する必要があることです。

`ServerlessV2ScalingConfiguration` 引数は、DocumentDB サーバーレスインスタンスのスケーリング容量範囲を指定します。これは、クラスター内のすべての DocumentDB サーバーレスインスタンスに適用される DocumentDB キャパシティユニット (DCU) の最小値と最大値で構成されます。

- `MinCapacity` 値は、最小スケーリング容量を指定します。
- `MaxCapacity` 値は、最大スケーリング容量を指定します。

スケーリングの詳細については、「Amazon DocumentDB Serverless スケーリング設定」を参照してください。

Using the AWS マネジメントコンソール

次の AWS マネジメントコンソール 設定例は、DocumentDB サーバーレスクラスターを作成する方法を示しています。

1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

[クラスター] テーブルが表示されます。

3. [作成] を選択します。

   [Amazon DocumentDB クラスターの作成] ページが表示されます。

4. 「Amazon DocumentDB クラスターの作成」ページの [クラスタータイプ] セクションで、[インスタンスベースのクラスター] を選択します (これはデフォルトのオプションです)。

5. [クラスター設定] セクションで次を行います。

   a. [クラスター識別子] には、**myserverlesscluster** などの一意の名称を入力します。コンソール操作では、入力方法に関係なくクラスター名のすべての文字が小文字に変換されることにご留意ください。

   b. [エンジンバージョン] では、5.0.0 を選択します (これはデフォルトのオプションです)。

6. [クラスターストレージ設定] セクションでは、[Amazon DocumentDB 標準] (デフォルトのオプションです) を選択します。

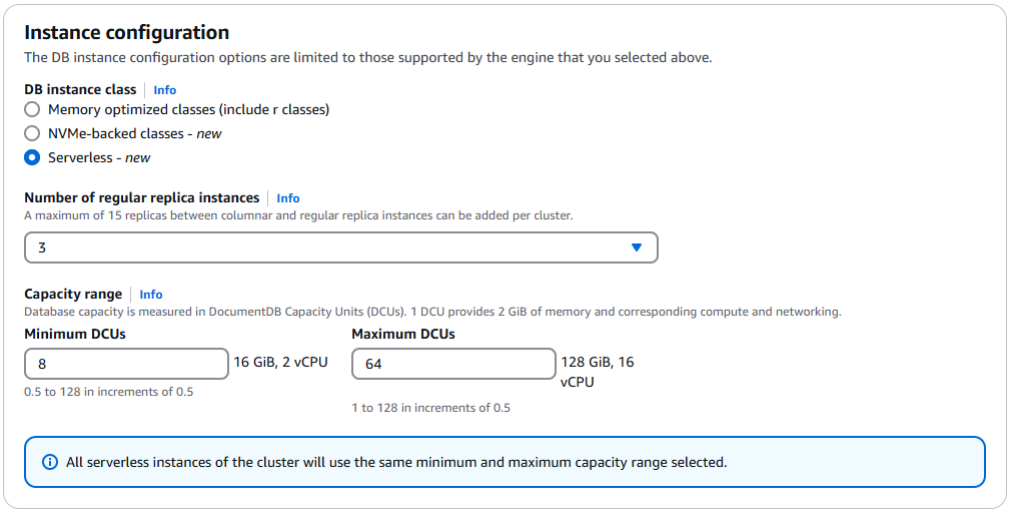
> **Note**
>
> このカテゴリのもう 1 つのオプションは、[Amazon DocumentDB I/O 最適化] です。いずれかのオプションの詳細については、「Amazon DocumentDB クラスターストレージ設定」を参照してください。

7. インスタンス設定 セクション:

   a. [DB インスタンスクラス] で、[サーバーレス] を選択します。

   b. [通常のレプリカインスタンスの数] では、3 を選択します (これはデフォルトのオプションです)。

c. [容量範囲] セクションでは、[最小 DCU] と [最大 DCU] をデフォルト値のままにします。これらのパラメータの設定の詳細については、「Amazon DocumentDB Serverless インスタンスの制限」を参照してください。



8. [接続] セクションで、デフォルト設定の [EC2 コンピューティングリソースに接続しない] のままにしておきます。
9. [認証] セクションで、プライマリユーザーのユーザー名を入力し、[セルフマネージド] を選択します。パスワードを入力して確認します。

   代わりに Managed in AWS Secrets Manager を選択した場合、詳細についてはAmazon DocumentDB と によるパスワード管理 AWS Secrets Manager「」を参照してください。
10. 他のすべてのオプションはデフォルトのままにして、[クラスターの作成] を選択します。

Using the AWS CLI

次の例では、各 ############## を独自の情報または設定パラメータに置き換えます。

を使用して DocumentDB サーバーレスインスタンスと互換性のあるクラスターを作成するには AWS CLI、「」を参照してくださいを使用したクラスターの作成 AWS CLI。

`create-db-cluster` コマンドに次の追加のパラメータを含めます:

```
--serverless-v2-scaling-configuration
     MinCapacity=minimum_capacity,MaxCapacity=maximum_capacity
```

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
```

Amazon DocumentDB Serverless インスタンスの追加

DocumentDB サーバーレスインスタンスを追加するには、クラスターへの Amazon DocumentDB インスタンスの追加 の同じ手順に従い、インスタンスクラスとして db.serverless を指定してください。

AWS マネジメントコンソールを使用してサーバーレスインスタンスを追加します。

コンソールを使用して DocumentDB サーバーレスインスタンスを追加するには、「クラスターへの Amazon DocumentDB インスタンスの追加」を参照して、「 AWS マネジメントコンソールの使用」タブを選択します。

を使用したサーバーレスインスタンスの追加 AWS CLI

を使用して DocumentDB サーバーレスインスタンスを追加するには AWS CLI、「」を参照クラスターへの Amazon DocumentDB インスタンスの追加し、「 の使用 AWS CLI」タブを選択します。

次のインスタンスクラス CLI 引数を使用します。

```
--db-instance-class db.serverless
```

例:

```
aws docdb create-db-instance \
      --db-cluster-identifier sample-cluster \
      --db-instance-identifier sample-instance \
      --db-instance-class db.serverless \
      --engine docdb
```

Amazon DocumentDB Serverless への移行

トピック

- 既存の DocumentDB クラスターをサーバーレスに移行する
- MongoDB から DocumentDB サーバーレスへの移行

既存の DocumentDB クラスターをサーバーレスに移行する

クラスターのエンジンバージョンのアップグレード

プロビジョニングされたクラスターが、DocumentDB サーバーレスをサポートしていないより低いエンジンバージョンを実行している場合は、まずクラスターをサポートされているエンジンバージョンにアップグレードする必要があります。詳細については、「Amazon DocumentDB のインプレースメジャーバージョンアップグレード」を参照してください。

プロビジョニングされたクラスターを DocumentDB サーバーレスに移行する

プロビジョニングされたクラスターを DocumentDB サーバーレスを使用できるように切り替えるには、以下の手順に従います。

1. DocumentDB サーバーレスで使用するには、プロビジョニングされたクラスターのエンジンバージョンをアップグレードする必要があるかどうかを確認します。「DocumentDB サーバーレスの要件と制限」を参照してください。

   > **Note**
   >
   > プロビジョニングされたクラスターが DocumentDB サーバーレスで使用できないエンジンバージョンを実行している場合、クラスターのエンジンバージョンをアップグレードします。「Amazon DocumentDB のインプレースメジャーバージョンアップグレード」を参照してください。

2. クラスターのスケーリング設定を設定します。スケーリング設定の選択の詳細については、「DocumentDB サーバーレスクラスターのスケーリング容量範囲の選択」を参照してください。クラスターのスケーリング設定を変更するには、「クラスターのスケーリング容量範囲設定の表示と変更」を参照してください。
3. DocumentDB サーバーレスの要件と制限 からの DocumentDB サーバーレスの要件と制限を考慮して、他のクラスタープロパティを設定します。

4. クラスターに 1 つ以上の DocumentDB サーバーレスインスタンスを追加します。「Amazon DocumentDB Serverless インスタンスの追加」の手順に従います。

   > **Note**
   >
   > 場合によっては、クラスター内に 1 つ以上のプロビジョニングされたリーダーインスタンスが既に存在することがあります。その場合、新しいインスタンスを作成する代わりに、リーダーの 1 つを DocumentDB サーバーレスインスタンスに変換できます。これを行うには、「インスタンスクラスの変更」の手順に従います。

5. (オプション) フェイルオーバーオペレーションを実行して、DocumentDB サーバーレスインスタンスをクラスターライターにします。「Amazon DocumentDB フェイルオーバー」を参照してください。
6. (オプション) 残りのプロビジョニングされた Amazon DocumentDB インスタンスを DocumentDB サーバーレスインスタンスに変換するか (「インスタンスクラスの変更」を参照)、クラスターから削除します (「Amazon DocumentDB インスタンスの削除」を参照)。

Using the AWS マネジメントコンソール

次の AWS マネジメントコンソール 設定例は、Amazon DocumentDB 5.0.0 を実行している Amazon DocumentDB プロビジョニング済みクラスターを使用した移行プロセスを示しています。このクラスターでは、DocumentDB サーバーレスの使用を開始するために エンジンバージョンのアップグレードは必要ありません。クラスターは `sample` という名前で、`sample`、`sample2`、および `sample3` という名前の 3 つのプロビジョニングされたインスタンスで始まります。この例では、これら 3 つのインスタンスは 3 つのサーバーレスインスタンスに置き換えられます。

1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

[クラスター] テーブルが表示されます。

3. [クラスター] テーブルで、サーバーレスインスタンスを追加するクラスターのチェックボックスを選択します。
4. [アクション] を選択し、[インスタンスの追加] を選択します。
5. [インスタンスの追加] ダイアログボックスで、作成する新しいサーバーレスインスタンスごとに [DB インスタンスクラス] セクションで [サーバーレス] を選択します。
6. [サーバーレス容量設定] では、ダイアログボックスの容量の説明に基づいてスケーリング設定を設定します。
7. (オプション) 別のインスタンスを追加するには、[インスタンスの追加] を選択します。必要な新しいインスタンス数に達するまで、インスタンスの追加を続行します。

   この例では、3 つの新しいサーバーレスインスタンスが作成されます。
8. [作成] を選択します。

   インスタンスが作成されるまでに数分かかります。コンソールまたは を使用して AWS CLI 、インスタンスのステータスを表示できます。詳細については、「Amazon DocumentDB クラスターのステータスのモニタリング」を参照してください。
9. [クラスター] テーブルに戻り、元のプロビジョニング済みインスタンスの 3 つすべてのチェックボックスを選択します。
10. [アクション]、[削除] の順に選択します。

削除中、フェイルオーバーが自動的に実行され、残りのインスタンスの 1 つがライターに昇格されます。数分後、削除プロセスが完了します。既存のクラスターに 3 つの DocumentDB サーバーレスインスタンスが含まれるようになりました ([サイズ] 列で定義) 。

Using the AWS CLI

次の AWS CLI 設定例は、Amazon DocumentDB 5.0.0 を実行している Amazon DocumentDB プロビジョニング済みクラスターを使用した移行プロセスを示しています。このクラスターでは、DocumentDB サーバーレスの使用を開始するためにエンジンバージョンのアップグレードは必要ありません。クラスターは `sample` という名前で、`sample`、`sample2`、および `sample3` という名前の 3 つのプロビジョニングされたインスタンスで始まります。この例では、これら 3 つのインスタンスは 3 つのサーバーレスインスタンスに置き換えられます。このクラスターは `sample-cluster` という名前で、`sample-provisioned-instance-1` および `sample-`

provisioned-instance-2 という名前の 2 つのプロビジョニング済みインスタンス、ライターインスタンスとリーダーインスタンスから始まります。

次の例では、各 *############* を独自の情報または設定パラメータに置き換えます。

aws docdb describe-db-clusters オペレーションを使用して、クラスターのステータスを確認します。次のコードは、クラスター sample-cluster のステータスを検索し、結果をテーブルに出力します。

```
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             |
+---------------------------------+---------------------------------
+------------------+----------------+
```

クラスターのスケーリング設定を追加します。

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16
```

サーバーレスインスタンスを追加します。この例では、sample-serverless-instance-1 および sample-serverless-instance-2 という名前の新しいサーバーレスインスタンスが追加されています。

```
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
```

次を入力して、サーバーレスインスタンスが使用可能になるまで待機してから次に進みます。

```
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
```

フェイルオーバーを実行して、新しい sample-serverless-instance-1 インスタンスをクラスターのライターにします。

```
aws docdb failover-db-cluster \
    --db-cluster-identifier sample-cluster \
    --target-db-instance-identifier sample-serverless-instance-1
```

フェイルオーバーが完了するまでに数秒かかり、その後 sample-serverless-instance-1 がクラスターのライターになります。これを次の入力で検証します。

```
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             |
+--------------------------------+--------------------------------
+------------------+----------------+
```

最後に、元のプロビジョニングされたインスタンスを削除します。

```
aws docdb delete-db-instance \
    --db-instance-identifier sample-provisioned-instance-1

aws docdb delete-db-instance \
    --db-instance-identifier sample-provisioned-instance-2
```

MongoDB から DocumentDB サーバーレスへの移行

プロビジョニングされた Amazon DocumentDB と同様に、MongoDB データベースを DocumentDB サーバーレスに移行できます。詳細については、「Amazon DocumentDB の移行とアップグレード」を参照してください。

Amazon DocumentDB Serverless の管理

クラスターのスケーリング容量範囲設定の表示と変更

`ServerlessV2ScalingConfiguration` 引数は、DocumentDB サーバーレスインスタンスのスケーリング容量範囲を指定します。これは、クラスター内のすべての DocumentDB サーバーレスインスタンスに適用される DocumentDB キャパシティユニット (DCU) の最小値と最大値で構成されます。

- **`MinCapacity`** — クラスター内の DocumentDB サーバーレスインスタンスの最小スケーリング容量。

- **`MaxCapacity`** — クラスター内の DocumentDB サーバーレスインスタンスの最大スケーリング容量。

> **Note**
>
> 次のスケーリング設定の変更では、新しい `MinCapacity` と `MaxCapacity` の値を反映するためにインスタンスを再起動する必要があります。
>
> - `MaxCapacity` 値への変更
> - `MinCapacity` 値をより高い値から 1.0 以下に変更する
> - `MinCapacity` 値をより低い値から 1.0 より大きい値に変更する

スケーリング設定と適切なスケーリング容量制限の選択方法の詳細については、「Amazon DocumentDB Serverless スケーリング設定」を参照してください。

Using the AWS マネジメントコンソール

次の AWS マネジメントコンソール 設定例は、DocumentDB サーバーレスクラスターのスケーリング設定を表示および編集する方法を示しています。

1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

   [クラスター] テーブルが表示されます。
3. [クラスター] テーブルで、スケーリング容量を変更するクラスターのチェックボックスを選択します。
4. [アクション] を選択してから、[変更] を選択します。

[クラスターの変更] ダイアログボックスが表示されます。

5. [サーバーレス容量設定] セクションを見つけ、ダイアログボックスの容量の説明に基づいてスケーリング設定 ([容量範囲]) を設定します。

   スケーリングとの容量単位の詳細については、「Amazon DocumentDB Serverless スケーリング設定」を参照してください。

6. [続行] をクリックしてください。
7. [変更のスケジューリング] で、[すぐに適用] を選択します。
8. [クラスターを変更] を選択します。
9. 変更が完了したら、各サーバーレスインスタンスを再起動する必要があります。ライターの非可用性を最小限に抑えるには、次の一連のオペレーションを実行します。

   a. 各サーバーレスリーダーインスタンスを再起動します。

      i. リーダーインスタンスを選択し、[アクション] を選択してから、[再起動] を選択します。

      ii. インスタンスのステータスが 使用可能 に戻るまで待機します。

   b. 再起動したサーバーレスインスタンスへのフェイルオーバーを実行します。

      i. クラスターを選択し、[アクション] を選択してから、[フェイルオーバー] を選択します。

      ii. フェイルオーバーオペレーションが完了するまで待機します。

   c. 残りのサーバーレスインスタンスを再起動します。

      i. 残りのインスタンスを選択し、[アクション] を選択してから、[再起動] を選択します。

      ii. インスタンスのステータスが 使用可能 に戻るまで待機します。

Using the AWS CLI

次の AWS CLI 設定例は、現在のスケーリング設定を示しています。

次の例では、各 `##############` を独自の情報または設定パラメータに置き換えます。

クラスターの現在のスケーリング設定は、 `describe-db-clusters` AWS CLI コマンドを使用して表示できます。

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[0].ServerlessV2ScalingConfiguration'
```

このコマンドの出力は次のとおりです。

```
{
    "MinCapacity": 0.5,
    "MaxCapacity": 16.0
}
```

クラスターのスケーリング設定は、modify-db-cluster コマンドを使用して変更できます。

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=32
```

完了したら、各サーバーレスインスタンスを再起動する必要があります。ライターの非可用性を最小限に抑えるために、次の一連のオペレーションを実行できます。

```
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
```

サーバーレスリーダーの昇格階層の表示と変更

複数のサーバーレスインスタンスを含むクラスター、またはプロビジョン済みインスタンスとサーバーレスインスタンスの混在するクラスターでは、各サーバーレスインスタンスのプロモーション層の設定に注意してください。この設定は、プロビジョン済みインスタンスよりも、サーバーレスインスタンスの多くの動作を制御します。

プロビジョン済みインスタンスの場合、層 0～15 の選択肢は、フェイルオーバー操作時に Amazon DocumentDB がどのリーダーインスタンスをライターに昇格させるかを選択する順序のみを決定します。ただし、サーバーレスリーダーインスタンスの場合、層番号によって、インスタンスがライターインスタンスの容量に合わせてスケールアップするのか、それとも独自のワークロードに基づいて個別にスケールアップするのかが決まります。層 0 または 1 のサーバーレスリーダーインスタンスは、少なくともライターインスタンスと同じ大きさの最小容量に保たれます。これにより、フェイルオーバー発生時にライターインスタンスから引き継ぐ準備が整います。ライターインスタンスがプロビジョン済みインスタンスの場合、 Amazon DocumentDB では同等の Amazon DocumentDB サーバーレス容量を推定します。この推定値をサーバーレスリーダーインスタンスの最小容量として使用します。

層 2～15 の DocumentDB サーバーレスリーダーインスタンスは、最小容量に対する同じ制約はありません。アイドル状態の場合、クラスターの容量範囲で指定された DocumentDB 容量単位 (DCU) の最小値までスケールダウンできます。

Using the AWS マネジメントコンソール

次の AWS マネジメントコンソール 設定例は、DocumentDB サーバーレスインスタンスリーダーのプロモーション階層設定を表示および変更する方法を示しています。

1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソールを開きます。

   各インスタンスの昇格階層は、 AWS マネジメントコンソールの [昇格階層] 列に表示されます。

2. ナビゲーションペインで クラスター を選択します。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン

(≡) を選択します。

[クラスター] テーブルが表示されます。

3. [クラスター] テーブルで、昇格階層を変更するインスタンスのチェックボックスを選択します。
4. [アクション] を選択してから、[変更] を選択します。

[インスタンスの変更] ダイアログボックスが表示されます。

5. [フェイルオーバー] セクションを見つけ、[昇格階層] を必要なレベルに設定します。
6. [続行] をクリックしてください。
7. [変更のスケジューリング] で、[すぐに適用] を選択します。
8. [インスタンスの変更] を選択します。

Using the AWS CLI

次の AWS CLI 設定例は、現在のスケーリング設定を示しています。

次の例では、各 *############* を独自の情報または設定パラメータに置き換えます。

クラスター内のすべてのインスタンスの昇格階層は、 `describe-db-clusters` AWS CLI コマンドを使用して表示できます。

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[0].DBClusterMembers' \
    --output table
```

このコマンドの出力は次のとおりです。

```
-------------------------------------------------------------------------------------------------
|                                          DescribeDBClusters
                    |
+---------------------------------+---------------------------------
+------------------+----------------+
|  DBClusterParameterGroupStatus |      DBInstanceIdentifier       | IsClusterWriter
  | PromotionTier  |
```

```
+---------------------------------+---------------------------------
+------------------+----------------+
|  in-sync                        |  sample-serverless-instance-2   |  False
  |  1             |
|  in-sync                        |  sample-serverless-instance-1   |  True
  |  1             |
+---------------------------------+---------------------------------
+------------------+----------------+
```

特定のインスタンスの昇格階層は、modify-db-instance コマンドを使用して変更できます。

```
aws docdb modify-db-instance \
    --db-instance-identifier sample-serverless-instance-2 \
    --promotion-tier 3
```

Amazon DocumentDB Serverless インスタンスの制限

DocumentDB サーバーレスインスタンスの場合、インスタンスあたりの以下の制限は、インスタンスの現在のスケーリング容量によって異なります。

- インスタンスメモリ (GiB)
- 接続 (すべて)
- カーソル制限
- 同時に開くことのできるトランザクション数
- 接続 (アクティブ)

次のテーブルは、インスタンスの現在のスケーリング容量に関連して、サーバーレスインスタンスのインスタンスごとの制限がどのようにスケーリングされるかを示しています。

制限値は、クラスターの MinCapacity スケーリング設定が 2 以上に設定されているかどうかによって異なることに注意してください。MinCapacity が 2 以下に設定されている場合、カーソル制限、オープントランザクション、および接続 (アクティブ) の制限値は、より低い最大値に上限が設定されます。(上限の技術的な理由は、「アイドル」状態へのスケールダウンをサポートすることです。詳細については、「アイドル状態 (0.5 DCU)」を参照してください。

MinCapacity ≤1 の場合の db.serverless インスタンスの制限

現在の容量 (DCU)	インスタンスメモリ (GiB)	接続 (すべて)	カーソル制限	同時に開くことのできるトランザクション数	接続 (アクティブ)
0.5	1	250	6	12	39
1	2	500	12	24	79
2	4	1,000	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

MinCapacity >1 の場合の db.serverless インスタンスの制限

現在の容量 (DCU)	インスタンスメモリ (GiB)	接続 (すべて)	カーソル制限	同時に開くことのできるトランザクション数	接続 (アクティブ)
1.5	3	750	18	36	124
2	4	1,000	24	48	173
4	8	2000	48	96	416
8	16	4000	96	192	1071

現在の容量 (DCU)	インスタンスメモリ (GiB)	接続 (すべて)	カーソル制限	同時に開くことのできるトランザクション数	接続 (アクティブ)
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

次の CloudWatch メトリクスを使用して、インスタンスごとの制限を監視およびアラームできます。Amazon DocumentDB CloudWatch のメトリクスのさらなる詳細については、Amazon DocumentDB と CloudWatch のモニタリング をご参照ください。

[リソース]	CloudWatch 制限のメトリクス	CloudWatch の使用状況メトリクス (最大 1 分)	CloudWatch の使用状況メトリクス
インスタンスメモリ	-	-	FreeableMemory
接続 (すべて)	DatabaseConnectionsLimit	DatabaseConnectionsMax	DatabaseConnections
カーソル	DatabaseCursorsLimit	DatabaseCursorsMax	DatabaseCursors
取引	TransactionsOpenLimit	TransactionsOpenMax	TransactionsOpen

Amazon DocumentDB Serverless スケーリング設定

トピック

- DocumentDB サーバーレスクラスターのスケーリング容量範囲の選択
- DocumentDB サーバーレスクラスターの MinCapacity 設定の選択
- DocumentDB サーバーレスクラスターの MaxCapacity 設定の選択
- メモリ不足エラーを回避する
- サーバーレスインスタンスがスケールダウンしないのはなぜですか?

DocumentDB サーバーレスクラスターのスケーリング容量範囲の選択

DocumentDB サーバーレスインスタンスを Amazon DocumentDB クラスターに追加する前に、クラスターに `ServerlessV2ScalingConfiguration` パラメータも設定されている必要があります。

`ServerlessV2ScalingConfiguration` パラメータは、クラスター内の任意のサーバーレスインスタンスのサーバーレススケーリング容量範囲を定義する 2 つの値で構成されます。

- **`MinCapacity`** — クラスター内の DocumentDB サーバーレスインスタンスの最小スケーリング容量。
- **`MaxCapacity`** — クラスター内の DocumentDB サーバーレスインスタンスの最大スケーリング容量。

DocumentDB サーバーレスクラスターの **`MinCapacity`** 設定の選択

`MinCapacity`には、常に 0.5 を選択しようとします。この値を指定することで、インスタンスは、完全にアイドル状態のときに最小容量にスケールダウンしながら、アクティブな状態を維持できます。ただし、そのクラスターの使用方法やその他の設定によっては、最も効果的な最小容量が異なる場合もあります。最小容量設定を選択する場合は、以下の要素を考慮してください。

- DocumentDB サーバーレスインスタンスのスケーリングレートは、そのインスタンスの現在の容量によって異なります。現在の容量が大きいほど、スケールアップが速くなります。インスタンスを非常に大きな容量にすばやくスケールアップする必要があるときは、スケーリングレートの要件を満たす値に最小容量を設定することを検討してください。
- 通常、ワークロードが高いか低いかを見越してインスタンスのインスタンスクラスを変更している場合は、その経験を活かして同等の DocumentDB サーバーレス容量範囲を概算で見積もることができます。プロビジョニングされた Amazon DocumentDB インスタンスタイプのメモリサイズを確認するには、「インスタンスクォータ」を参照してください。

例えば、クラスターのワークロードが低い場合に db.r6g.xlarge インスタンスクラスを使用するとします。そのインスタンスクラスのメモリは 32 GiB です。したがって、MinCapacity を 16 に指定すると、ほぼ同じ容量にスケールダウンできるサーバーレスインスタンスをセットアップできます。これは、各 DCU が約 2 GiB のメモリに対応するためです。db.r6g.xlarge インスタンスの使用率が低い場合に、インスタンスをさらにスケールダウンさせるため、やや小さい値を指定することがあります。

- インスタンスのバッファキャッシュに一定量のデータがあるときにアプリケーションが最も効率的に動作する場合は、頻繁にアクセスされるデータを保持するのに十分なメモリ容量を持つ最小の DCU 設定を指定することを検討してください。それ以外の場合、サーバーレスインスタンスがさらに小さいメモリサイズにスケールダウンした場合、一部のデータがバックキャッシュから削除されます。その後、インスタンスのスケールアップ時に、その情報が経時的に読み込まれてバッファキャッシュに戻ります。データをバッファキャッシュに戻すための I/O 量が大きい場合は、最小 DCU 値を大きくする方が効果的な場合があります。詳細については、「インスタンスのサイズ指定」を参照してください。
- ほとんどの時間、DocumentDB サーバーレスインスタンスが特定の容量で実行されている場合、ベースラインよりも小さく、ただし小さすぎない最小容量の設定を検討してください。サーバーレスインスタンスが現在の容量が必要容量より極端に小さくない場合、スケールアップする規模と速度を最も効果的に見積もることができます。
- プロビジョン済みワークロードのメモリ要件が T3 や T4g–などの小さなインスタンスクラスに対して大きすぎる場合は、R5 や R6g インスタンスに相当するメモリを提供する最小 DCU 設定を選択します。
- 指定された機能MinCapacityには、次の最小値を使用します (変更される可能性があります)。
  - パフォーマンスインサイト — 2 DCU。
  - グローバルクラスター — 8 DCUs (プライマリにのみ適用 AWS リージョン)
- Amazon DocumentDB では、レプリケーションはストレージレイヤーで発生するため、リーダー容量はレプリケーションに直接影響しません。ただし、個別にスケールする DocumentDB サーバーレスインスタンスの場合は、クエリのレイテンシーを避けるために、書き込み負荷の高い期間にワークロードを処理するのに十分な最小容量があることを確認してください。昇格階層 2 ~ 15 のリーダーインスタンスでパフォーマンスの問題が発生した場合は、クラスターの最小容量を増やすことを検討してください。リーダーインスタンスのスケールをライターと一緒にスケーリングするか、個別にスケーリングするかを変更する方法の詳細については、「サーバーレスリーダーの昇格階層の表示と変更」を参照してください。

  DocumentDB サーバーレスリーダーインスタンスを持つクラスターがある場合、リーダーの昇格階層が 0 または 1 でない場合、リーダーはライターインスタンスと一緒にスケーリングされませ

ん。この場合、最小容量を小さく設定すると、レプリケーションの遅延が大きくなる場合があります。これは、データベースがビジー状態のときに、ライターからの変更を適用するのに十分な容量がリーダーにない可能性があるためです。最小容量を、ライターインスタンスと同等の量のメモリと CPU を表す値に設定します。

- DocumentDB サーバーレスインスタンスを最小容量から最大容量までスケーリングするのにかかる時間は、DCU の最小値と最大値の差によって異なります。現在のインスタンスの容量が大きい場合、DocumentDB サーバーレス では、小さな容量から開始する場合よりも大きな増分でインスタンスをスケールアップします。したがって、比較的大きい最大容量を指定し、ほとんどの時間、インスタンスがその容量付近で使用されている場合は、最小 DCU の設定を引き上げることを検討してください。そうすれば、アイドル状態のインスタンスを、より迅速に最大容量にスケールアップできます。
- 特定のインスタンス制限は、接続制限、カーソル制限、オープントランザクション制限など、サーバーレスインスタンスの現在の容量によって決まります。インスタンスの現在の容量が小さい場合、それに応じて制限も小さくなります。サーバーレスインスタンスをその `MinCapacity` 値にスケールダウンするときにこれらの制限に問題がある場合は、`MinCapacity` をより高い値に増やすことを検討してください。詳細については、「Amazon DocumentDB Serverless インスタンスの制限」を参照してください。
- さらに、アクティブな接続制限、カーソル制限、オープントランザクション制限など、`MinCapacity` が 1.0 DCU 以下に設定されている場合、特定のインスタンス制限はより低い最大値に上限が設定されます。これらの制限がワークロードに対して不十分な場合は、少なくとも 1.5 DCUs`MinCapacity`の値を使用します。詳細については、「Amazon DocumentDB Serverless インスタンスの制限」を参照してください。

クラスターのスケーリング設定を変更する方法については、「Amazon DocumentDB Serverless の管理」を参照してください。

DocumentDB サーバーレスクラスターの `MaxCapacity` 設定の選択

DocumentDB サーバーレスの最大容量設定には、常にある程度大きい値を選択しようとします。最大容量が大きいと、集中的なワークロードを実行している場合に、インスタンスは最もスケールアップできます。値を小さくすると、予期せぬ料金が発生する可能性を回避できます。そのクラスターの使用方法およびその他の設定によっては、最も効果的な値が当初検討していたより大きくなったり、小さくなったりすることがあります。最大容量設定を選択する場合は、以下の要素を考慮してください。

- 最大容量は、最小容量より大きくなければなりません。最小容量と最大容量を同一に設定することができます。ただし、その場合は容量がスケールアップまたはスケールダウンすることはありません。したがって、テスト以外では、最小容量と最大容量に同じ値を使用することは適切ではありません。
- 最大容量は 1.0 DCU 以上、最大 256 DCU である必要があります。
- サーバーレスインスタンスのスケーリングとリソース使用状況をモニタリングします。サーバーレスインスタンスが頻繁に最大容量にスケーリングされ、リソースの制約に達する場合 (`DCUUtilization`メトリクスが 100.0 の場合など)、より高い`MaxCapacity`値を選択します。詳細については、「Amazon DocumentDB Serverless のモニタリング」を参照してください。
- 通常、ワークロードが高いか低いかを見越してプロビジョニングされたインスタンスのインスタンスクラスを変更している場合は、その経験を活用して同等の DocumentDB サーバーレス容量範囲を見積もることができます。プロビジョニングされた Amazon DocumentDB インスタンスのメモリサイズを確認するには、「インスタンスクォータ」を参照してください。

  例えば、クラスターのワークロードが高い場合に `db.r6g.4xlarge` インスタンスクラスを使用するとします。そのインスタンスクラスのメモリは 128 GiB です。したがって、DCU の最大設定を 64 に指定すると、ほぼ同じ容量にスケールアップできるサーバーレスインスタンスを設定できます。これは、各 DCU が約 2 GiB のメモリに対応するためです。`db.r6g.4xlarge` インスタンスにワークロードを効果的に処理するのに十分な容量がないことがあり、インスタンスをよりスケールアップさせるために多少大きい値を指定することができます。
- データベースの使用に予算の上限がある場合は、すべてのサーバーレスインスタンスを常に最大容量で実行しても、予算の上限に収まるような値を選択してください。クラスターに n 個のサーバーレスインスタンスがある場合、クラスターが常に消費できるサーバーレスの理論上の最大容量は、クラスターの最大 DCU 設定の n 倍であることに注意してください。(例えば、一部のリーダーがライターから独立してスケーリングする場合など、実際の消費量は少なくなる場合があります)。
- サーバーレスリーダーインスタンスを利用してライターインスタンスから一部の読み取り専用ワークロードをオフロードするには、最大容量設定を小さく選択できることがあります。これは、各リーダーインスタンスが、クラスターに単一のインスタンスしか含まれていない場合ほど大きくスケーリングする必要がないことを反映するためです。
- データベースパラメータの設定間違いやアプリケーション内の非効率的なクエリによる過度の使用から保護したいとします。その場合、設定可能な理論的な最大値よりも最大容量設定を小さく選択することで、誤って過剰に使用することを回避できます。
- 実際のユーザーアクティビティによるスパイクがまれしか発生しない場合は、最大容量設定を選択する際にその機会を考慮できます。アプリケーションが完全なパフォーマンスとスケーラビリティ

で動作し続けることを優先する場合は、通常の使用状況よりも大きい最大容量設定を指定できます。アクティビティの非常に極端なスパイク中、アプリケーションのスループットが低下しても問題ない場合は、最大容量を少し小さめに設定できます。アプリケーションの実行を維持するのに十分なメモリと CPU リソースがある設定を選択してください。

- クラスターで 各インスタンスのメモリ使用量を増やす設定をオンにする場合は、最大 DCU 値を決定する際にそのメモリを考慮に入れてください。このような設定には、Performance Insights およびグローバルクラスターの設定が含まれます。それらの機能が使用されているときに、サーバーレスインスタンスがワークロードを処理するのに十分なスケールアップができる 最大 DCU 値になっていることを確認します。最大 DCU の設定が小さいことと、メモリのオーバーヘッドが発生する Amazon DocumentDB 機能が組み合わされることによって発生する問題のトラブルシューティングについては、「メモリ不足エラーを回避する」(以下) を参照してください。
- 指定された機能`MaxCapacity`には、次の最小値を使用します (変更される可能性があります) 。
  - データボリュームが大きいクラスターでのサーバーレスインスタンスの作成 — 2 DCU (クラスター復元の一部としてのサーバーレスインスタンスの作成が含まれます)。
- 特定のインスタンス制限は、接続制限、カーソル制限、オープントランザクション制限など、インスタンスの現在の容量によって決まります。ワークロード`MaxCapacity`の値を選択するときは、これらの制限のいずれかによってボトルネックにならないように、これらのインスタンスの制限を念頭に置いてください。詳細については、「Amazon DocumentDB Serverless インスタンスの制限」を参照してください。

クラスターのスケーリング設定を変更する方法については、「Amazon DocumentDB Serverless の管理」を参照してください。

メモリ不足エラーを回避する

DocumentDB サーバーレスインスタンスの 1 つが常に最大容量の制限に達している場合、Amazon DocumentDB ではこの状態をインスタンスのステータスを [互換性のないパラメータ] に設定することで表示します。DB インスタンスが [互換性のないパラメータ] ステータスの間、一部のオペレーションはブロックされます。例えば、エンジンバージョンをアップグレードすることはできません。Amazon DocumentDB インスタンスのステータスの詳細については、「」を参照してくださいAmazon DocumentDB インスタンスのステータスのモニタリング。

通常、インスタンスでは、メモリ不足エラーが原因で頻繁に再起動した場合、このステータスになります。Amazon DocumentDB では、このタイプの再起動が発生したときにイベントを記録します。リソースイベントを表示するには、「Amazon DocumentDB イベントの表示」を参照してください。パフォーマンスインサイトなどの設定をオンにすることによるオーバーヘッドが原因で、メモリ

使用量が異常に大きくなる場合があります。また、インスタンスのワークロードが高い場合や、多数のスキーマオブジェクトに関連するメタデータの管理から発生する場合があります。

インスタンスが頻繁に最大容量に達しないように、メモリの負荷が低くなると、Amazon DocumentDB ではインスタンスのステータスを自動的に [使用可能] に戻します。

この状態から回復させるには、以下アクションの一部またはすべてを実行できます。

- クラスターの DocumentDB 容量単位 (DCU) の最小値を変更して、サーバーレスインスタンスの容量の下限を引き上げます。これを行うことで、アイドル状態のデータベースが、クラスターでオンになっている機能に必要なメモリよりも少ない容量にスケールダウンする問題を回避できます。クラスターの DCU 設定を変更した後、サーバーレスインスタンスを再起動します。そうすることで、Amazon DocumentDB がステータスを [使用可能] にリセットできるかどうかが評価されます。
- クラスターの DCU の最大値を変更して、サーバーレスインスタンスの容量の上限を引き上げます。そうすることで、クラスターでオンになっている機能やデータベースワークロードに、ビジー状態のデータベースが十分なメモリがある容量にスケールアップできない問題を回避できます。クラスターの DCU 設定を変更した後、サーバーレスインスタンスを再起動します。そうすることで、Amazon DocumentDB がステータスを [使用可能] にリセットできるかどうかが評価されます。
- メモリオーバーヘッドが必要な設定をオフにします。例えば、パフォーマンスインサイトなどの機能がオンになっているが、使用していないとします。その場合は、それをオフにできます。または、クラスターの最小容量と最大容量値を調整することで、これらのタイプの機能で使用されるメモリを考慮することもできます。最小および最大容量設定の選択に関するガイドラインについては、「DocumentDB サーバーレスクラスターのスケーリング容量範囲の選択」を参照してください。
- インスタンスのワークロードを削減します。例えば、クラスターにリーダーインスタンスを追加して、読み取り専用クエリからのロードを他のインスタンスに分散させることができます。

サーバーレスインスタンスがスケールダウンしないのはなぜですか?

場合によっては、データベースに負荷がかからない状態でも、DocumentDB サーバーレスが最小容量にスケールダウンしない場合があります。これは、次のような理由で発生します。

- パフォーマンスインサイトにより、リソースの使用量が増加し、データベースが最小容量までスケールダウンできない可能性があります。主な機能は以下のとおりです。

- リーダーインスタンスが最小容量までスケールダウンせず、ライターインスタンスと同じかそれ以上の容量にとどまっている場合は、リーダーインスタンスの優先度の階層を確認します。層 0 または 1 の DocumentDB サーバーレスリーダーインスタンスは、少なくともライターインスタンスと同じ大きさの最小容量に保たれます。リーダーの優先度の階層を 2 以上に変更して、ライターとは無関係にスケールアップおよびスケールダウンされるようにします。詳細については、「Amazon DocumentDB Serverless スケーリング」を参照してください。
- データベースワークロードが重いと、リソースの使用量が増える可能性があります。
- データベースのボリュームが大きいと、リソースの使用量が増える可能性があります。Amazon DocumentDB はクラスターの管理にメモリと CPU リソースを使用します。Amazon DocumentDB は、データベースボリュームが大きいクラスターを管理するために、より多くの CPU とメモリを必要とします。クラスターの最小容量がクラスター管理に必要な最小容量よりも少ない場合、クラスターは最小容量までスケールダウンされません。
- バックグラウンドメンテナンスアクティビティでは、リソースの使用量を定期的に増やすことができます。

データベースが設定された最小容量までスケールダウンしない場合は、データベースを停止して再起動し、時間の経過とともに蓄積された可能性のあるメモリフラグメントを再利用します。

> **Note**
>
> データベースを停止して開始すると、ダウンタイムが発生します。必要な場合にのみこれを行います。

Amazon DocumentDB Serverless のモニタリング

Amazon DocumentDB でのモニタリングの詳細については、「Amazon DocumentDB のモニタリングとログ記録」を参照してください。

トピック

- メモリ不足: 互換性のないパラメータステータス
- DocumentDB サーバーレスの Amazon CloudWatch メトリクス
- パフォーマンスインサイトによる DocumentDB サーバーレスパフォーマンスのモニタリング

メモリ不足: 互換性のないパラメータステータス

サーバーレスインスタンスの 1 つが常に最大容量の制限に達している場合、Amazon DocumentDB ではこの状態をインスタンスのステータスを [互換性のないパラメータ] に設定することで表示します。詳細については、「メモリ不足エラーを回避する」を参照してください。

DocumentDB サーバーレスの Amazon CloudWatch メトリクス

Amazon DocumentDB における CloudWatch の使用の詳細については、「Amazon DocumentDB と CloudWatch のモニタリング」を参照してください。

CloudWatch でサーバーレスインスタンスを表示すると、`ServerlessDatabaseCapacity` メトリクスで各インスタンスが消費する容量をモニタリングできます。また、`DatabaseConnections` やクエリなどの DocumentDB CloudWatch のスタンダードのメトリクスをすべてモニタリングできます。Amazon DocumentDB でモニタリング可能な CloudWatch メトリクスのすべてのリストは、「Amazon DocumentDB と CloudWatch のモニタリング」を参照してください。以下の CloudWatch インスタンスレベルのメトリクスは、DocumentDB サーバーレスインスタンスはスケールアップとスケールダウンを理解するうえで重要なモニタリングです。これらすべてのメトリクスは 1 秒ごとに計算されます。そうすれば、サーバーレスインスタンスの現在のステータスをモニタリングできます。サーバーレスインスタンスが容量に関連するメトリクスのしきい値に近づいた場合に通知するアラームを設定できます。最小容量と最大容量設定は適切か、調整が必要かを判断できます。データベースの効率を最適化するため、どこに注力すべきかを判断できます。

- **`ServerlessDatabaseCapacity`** — インスタンスレベルのメトリクスとして、現在のインスタンスの容量で表される DCU 値を報告します。クラスターレベルのメトリクスとして、クラスター内のすべての DocumentDB サーバーレスインスタンスの `ServerlessDatabaseCapacity` 値の平均を表しています。
- **`DCUUtilization.`** — このメトリクスは DocumentDB サーバーレスでは新しいものです。この値は割合 (%) で表されます。これは、`ServerlessDatabaseCapacity` メトリクスの値をクラスターの最大 DCU 値で割った値です。このメトリクスを解釈してアクションを実行するには、以下のガイドラインを考慮してください。
  - このメトリクスが `100.0` 値に近づいた場合、インスタンスは限りなく大きくスケールアップしたことになります。クラスターの最大 DCU 設定を引き上げることを検討してください。これにより、ライターとリーダーの両方のインスタンスを、より大きな容量にスケーリングできます。
  - 読み取り専用のワークロードによって、リーダーインスタンスが `DCUUtilization/100.0` に近づき、一方でライターインスタンスは最大容量に近づいていないとします。この場合は、クラスターにリーダーインスタンスを追加することを検討してください。これにより、ワークロード

の読み取り専用部分のワークロードをより多くのインスタンスに分散することで、各リーダーインスタンスのロードを軽減できます。

- パフォーマンスとスケーラビリティが主な考慮事項である本番アプリケーションを実行しているとします。この場合、クラスターの最大 DCU 値を大きい数値に設定できます。目標は、`DCUUtilization` のメトリクスが常に 100.0 未満であることです。DCU の最大値を大きくすると、データベースのアクティビティに予期しないスパイクが発生した場合でも十分な余裕があり、安心につながります。実際に消費されたデータベース容量に対してのみ課金されます。

- **`CPUUtilization`** — このメトリクスは DocumentDB サーバーレスにおいて、プロビジョン済みのインスタンスとは異なる解釈がされます。DocumentDB サーバーレスの場合、この値は、現在の CPU の使用量をクラスターの最大 DCU 値で使用可能な CPU 容量で割った割合です。Amazon DocumentDB はこの値を自動的にモニタリングし、インスタンスが CPU 容量 を使用している割合が常に大きい場合、サーバーレスインスタンスをスケールアップします。

  このメトリクスが `100.0` 値に近づいた場合、インスタンスは最大 CPU 容量に達しています。クラスターの最大 DCU 設定を引き上げることを検討してください。このメトリクスがリーダーインスタンスで `100.0` 値に近づいた場合、クラスターにリーダーインスタンスを追加することを検討してください。これにより、ワークロードの読み取り専用部分のワークロードをより多くのインスタンスに分散することで、各リーダーインスタンスのロードを軽減できます。

- **`FreeableMemory`** — この値は、DocumentDB サーバーレスインスタンスを最大容量にスケーリングしたときに利用できる未使用のメモリ量を表します。現在の容量が最大容量を下回る 各 DCU では、この値は約 2 GiB 増加します。したがって、インスタンスが限りなく大きくスケールアップされるまで、このメトリクスはゼロに近づきません。

  このメトリクスがゼロの値に近づいた場合、インスタンスは可能な限りスケールアップし、使用可能なメモリの上限に近づいています。クラスターの最大 DCU 設定を引き上げることを検討してください。このメトリクスがリーダーインスタンスでゼロの値に近づいた場合、クラスターにリーダーインスタンスを追加することを検討してください。これにより、ワークロードの読み取り専用部分のワークロードをより多くのインスタンスに分散することで、各リーダーインスタンスのメモリ使用量を軽減できます。

- **`TempStorageIops`** — インスタンスにアタッチされたローカルストレージで実行された IOPS の数です。これには、読み取りと書き込みの両方の IOPS が含まれます。このメトリクスはカウントを表し、1 秒に 1 回測定されます。これは DocumentDB サーバーレスの新しいメトリクスです。詳細については、「Amazon DocumentDB と CloudWatch のモニタリング」を参照してください。

- **`TempStorageThroughput`** — インスタンスに関連するローカルストレージとの間で転送されるデータの量です。このメトリクスはバイトを表し、1 秒に 1 回測定されます。これは

DocumentDB サーバーレスの新しいメトリクスです。詳細については、「Amazon DocumentDB と CloudWatch のモニタリング」を参照してください。

通常、DocumentDB サーバーレスインスタンスのスケールアップの大部分は、メモリ使用率と CPU アクティビティに起因しています。`TempStorageIops` および `TempStorageThroughput` のメトリクスは、インスタンスとローカルストレージデバイス間の転送のためのネットワークアクティビティが、予期しない容量増加の原因となるまれなケースを診断するのに役立ちます。他のネットワークアクティビティを監視するには、以下の既存のメトリクスを使用できます。

- `NetworkReceiveThroughput`
- `NetworkThroughput`
- `NetworkTransmitThroughput`
- `StorageNetworkReceiveThroughput`
- `StorageNetworkThroughput`
- `StorageNetworkTransmitThroughput`

DocumentDB サーバーレス CloudWatch メトリクスが AWS 請求書にどのように適用されるか

AWS 請求書の DocumentDB サーバーレス料金は、モニタリングできるのと同じ`ServerlessDatabaseCapacity`メトリクスに基づいて計算されます。請求メカニズムでは、DocumentDB サーバーレスの容量を 1 時間の一部しか使用していない場合、このメトリクスで計算された CloudWatch の平均とは異なる場合があります。また、システムの問題で、短時間 CloudWatch メトリクスが利用できない場合にも異なる場合があります。したがって、お客様が `ServerlessDatabaseCapacity` の平均値から計算したものと、請求書に記載される DCU 時間の値が若干異なる場合があります。

DocumentDB サーバーレスメトリクスの Amazon CloudWatch CLI コマンドの例

次の AWS CLI 例は、DocumentDB サーバーレスに関連する最も重要な CloudWatch メトリクスをモニタリングする方法を示しています。いずれの場合も、`--dimensions` パラメータの `Value=` 文字列は、お客様の DocumentDB サーバーレスインスタンスの ID に置き換えてください。

以下の Linux の例では、インスタンスの最小、最大、平均の容量値を 1 時間で 10 分ごとに測定して表示しています。Linux の日付コマンドでは、現在の日付と時刻を基準にして開始時刻と終了時刻を

指定します。--query パラメータの sort_by 関数は、Timestamp のフィールドに基づいて結果を時系列でソートします。

```
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
```

以下の Linux の例では、クラスター内のインスタンスの容量のモニタリングを示しています。インスタンスの最小、最大、平均の容量使用率を測定しています。測定は、1 時間に 1 回、3 時間にわたって行います。これらの例では、DCU の固定数を表すの ServerlessDatabaseCapacity の代わりに、DCU の上限に対する割合を表す DCUUtilization メトリクスを使用しています。そうすれば、容量範囲の最小と最大の DCU 値の実際の数値を知る必要はありません。割合は 0 から 100 までの範囲で表示できます。

```
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
```

以下の Linux の例では、前のものと同様の測定を実行します。この場合は、CPUUtilization のメトリクスのための測定になります。測定は、1 時間で 10 分ごとに行われます。この数値は、インスタンスの最大容量設定に利用可能な CPU リソースに基づき、利用可能な CPU リソースを表します。

```
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
```

以下の Linux の例では、前のものと同様の測定を実行します。この場合は、FreeableMemory のメトリクスのための測定になります。測定は、1 時間で 10 分ごとに行われます。

```
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
```

パフォーマンスインサイトによる DocumentDB サーバーレスパフォーマンスのモニタリング

パフォーマンスインサイトを使用して、DocumentDB サーバーレスインスタンスのパフォーマンスをモニタリングできます。パフォーマンスインサイトの手順については、「Performance Insights を使用したモニタリング」を参照してください。

以下の新しいパフォーマンスインサイトカウンターが DocumentDB サーバーレスインスタンスに適用されます。

- **os.general.serverlessDBCapacity** - DCU 内のインスタンスの現在の容量。この値は、インスタンスの ServerlessDatabaseCapacity CloudWatch メトリクスに対応します。
- **os.general.dcuUtilization** — 設定された最大容量のうち、現在の容量の割合。この値は、インスタンスの DCUUtilization CloudWatch メトリクスに対応します。

- **os.general.maxConfiguredDcu** — この DocumentDB サーバーレスインスタンスのために設定された最大容量。これは、DCU で測定されます。
- **os.general.minConfiguredDcu** — この DocumentDB サーバーレスインスタンスのために設定された最小容量。これは、DCU で測定されます。

パフォーマンスインサイトカウンターのすべてのリストは、「カウンターメトリクス用の Performance Insights」を参照してください。

パフォーマンスインサイトで DocumentDB サーバーレスインスタンスの vCPU 値が表示される場合、その値は、インスタンスの DCU 値に基づいた推定値を表します。デフォルトの 1 分間隔では、vCPU 値の小数分は整数に切り上げられます。それ以上の間隔の場合、表示される vCPU 値は、1 分ごとの vCPU 値の整数の平均になります。

Amazon DocumentDB Elastic クラスターの使用

Amazon DocumentDB エラスティッククラスターは、1 秒あたり数百万回の読み取り/書き込み速度とペタバイトのストレージ容量でワークロードに対応できます。また、インスタンスを選択、管理、アップグレードする必要がなくなるため、開発者は Amazon DocumentDB を簡単に操作できるようになります。

> **Note**
>
> Elastic クラスターは MongoDB 5.0 互換のワイヤプロトコルを使用します。Amazon DocumentDB エンジンバージョン 8.0 ではサポートされておらず、中国リージョンまたは AWS GovCloud では使用できません。

Amazon DocumentDB エラスティッククラスター作成の目的は以下の通りです。

- 豊富なクエリ機能と MongoDB API との互換性を備え、事実上無制限のスケーリング機能を持つデータベースをお探しのお客様にソリューションを提供する。
- 接続制限を緩和し、パッチ適用によるダウンタイムを短縮する。
- 引き続き、JSON ワークロード向けのクラウドネイティブで、伸縮性のある最高水準のアーキテクチャへの投資してください。

トピック

- エラスティッククラスターのユースケース
- エラスティッククラスターの利点
- Elastic クラスターのリージョンとバージョンの可用性
- 制限事項
- Amazon DocumentDB のエラスティッククラスター
- Amazon DocumentDB エラスティッククラスターの開始方法
- Amazon DocumentDB のエラスティッククラスターのベストプラクティス
- Amazon DocumentDB エラスティッククラスターの管理
- Amazon DocumentDB エラスティッククラスターで保管中のデータ暗号化
- エラスティッククラスター上のサービスにリンクされたロール

エラスティッククラスターのユースケース

ドキュメントデータベースは、迅速で反復性のある開発のための柔軟なスキーマを必要とするワークロードに適しています。Amazon DocumentDB のユースケース例については、「ドキュメントデータベースのユースケース」を参照してください。

以下では、ドキュメントデータベースに大きな利点があるユースケース例をいくつか紹介します。

ユーザープロファイル

ドキュメントデータベースに柔軟性の高いスキーマがあるため、多種多様の属性およびデータ値を大量に持つドキュメントを保存できます。エラスティッククラスターは、さまざまなユーザーがさまざまな種類の情報を提供するオンラインプロファイルに対する実用的なソリューションです。アプリケーションが何億というユーザープロファイルをサポートしていると仮定してみましょう。エラスティッククラスターは、そのようなアプリケーションに対応できます。なぜなら、エラスティッククラスター自体をスケールアップおよびスケールアウトして、ユーザープロファイルへの何百万回もの書き込みと読み取りに対応できるためです。オフピークの時間帯には、スケールダウンしてコストを削減することもできます。

コンテンツの管理と履歴の記録

コンテンツを効果的に管理するために、さまざまなソースからコンテンツを収集して集計し、お客様に提供する必要があります。ドキュメントデータベースは、柔軟性の高いスキーマがあるため、あらゆる種類のデータの収集および保存に最適です。イメージやコメント、ビデオなど、ユーザーが生成したコンテンツをはじめとして、新しい種類のコンテンツを作成し、取り込むことができます。時間が経つにつれて、データベースにはより多くのストレージが必要になります。エラスティッククラスターでは、データをより多くのストレージボリュームに分散できるため、1 つのクラスターにペタバイト規模のデータを保存できます。

エラスティッククラスターの利点

AWS サービス統合

Amazon DocumentDB エラスティッククラスターは、Amazon DocumentDB と同じ方法で他の AWS サービスと統合されます。

- 移行 — AWS Database Migration Service (DMS) を使用して、MongoDB およびその他のリレーショナルデータベースから Amazon DocumentDB エラスティッククラスターに移行できます。

- モニタリング - Amazon CloudWatch を使用して、エラスティッククラスターの状態とパフォーマンスをモニタリングできます。
- セキュリティ — AWS Identity and Access Management (IAM) を使用して認証と認可を設定して、Elastic クラスターの作成や更新などの Elastic クラスターリソースを管理し、Amazon VPC を使用して VPC のみの安全な接続を実現できます。エラスティッククラスターの IAM 認証ログインはサポートされていません。
- データ管理 — を使用して AWS Glue 、Amazon S3、Amazon Redshift、Amazon OpenSearch Service などの他の AWS サービスとの間でデータをインポートおよびエクスポートできます。

Elastic クラスターのリージョンとバージョンの可用性

Elastic クラスターのリージョン可用性

次の表は、Amazon DocumentDB エラスティッククラスターが現在利用可能な AWS リージョンと、各リージョンのエンドポイントを示しています。

リージョン名	リージョン	アベイラビリティーゾーン
米国東部 (バージニア北部)	us-east-1	5
米国東部 (オハイオ)	us-east-2	3
米国西部 (オレゴン)	us-west-2	3
アジアパシフィック (香港)	ap-east-1	3
アジアパシフィック (ムンバイ)	ap-south-1	3
アジアパシフィック (ソウル)	ap-northeast-2	3
アジアパシフィック (シンガポール)	ap-southeast-1	3
アジアパシフィック (シドニー)	ap-southeast-2	3
アジアパシフィック (東京)	ap-northeast-1	3

リージョン名	リージョン	アベイラビリティーゾーン
カナダ (中部)	ca-central-1	3
南米 (サンパウロ)	sa-east-1	3
欧州 (フランクフルト)	eu-central-1	3
欧州 (アイルランド)	eu-west-1	3
欧州 (ロンドン)	eu-west-2	3
欧州 (ミラノ)	eu-south-1	3
欧州 (パリ)	eu-west-3	3

バージョンの可用性

エラスティッククラスターは MongoDB 5.0 互換のワイヤープロトコルをサポートします。Amazon DocumentDB 4.0 のインスタンスベースのクラスターとエラスティッククラスターの相違点については、「Amazon DocumentDB 4.0 と Elastic クラスター の機能上の違い」を参照してください。注: Elastic Clusters は Amazon DocumentDB 8.0 ではサポートされていません

制限事項

エラスティッククラスターの管理

このリリースでは、以下のクラスター管理機能はサポートされていません。

- カスタムポート (27017 のみがサポートされています)
- IAM ユーザーとロールを使用してデータベースへの認証を行う機能
- グローバルクラスターの作成機能
- 既存の Amazon DocumentDB イベントとイベントのサブスクリプション
- レンジシャーディング
- シャードの既存のコレクション
- マルチフィールドシャードキー

- シャードキーを変更する
- ポイントインタイム復元
- クローン作成
- Performance Insights

> **Note**
>
> エラスティッククラスターのクォータの詳細については、「」を参照してくださいAmazon DocumentDB クォータ。

クエリ操作と書き込み操作

このリリースでは、次のクエリおよび書き込み操作コマンド/機能はサポートされていません。

- スケーリング中の DDL コマンド
- プロファイラー
- パラメータグループ
- AWS Config
- AWS Backup

コレクションとインデックスの管理

このリリースでは、以下のコレクションおよびインデックス管理機能はサポートされていません。

- 一意のインデックス
- 部分インデックス
- テキストインデックス
- ベクトルインデックス
- ドキュメント圧縮

管理と診断

このリリースでは、以下の管理および診断コマンド/機能はサポートされていません。

- AWS Secrets Manager
- ロールベースのアクセスコントロール (RBAC) におけるカスタムロール
- 接続時、 0 の Write Concern はサポートされていません。
- VPC に属していて、既存のエラスティッククラスターに割り当てられていないサブネットの変更。

オプトイン機能

このリリースでは、以下の Amazon DocumentDB オプトイン機能はサポートされていません。

- ACID トランザクション
- DDL/DML 監査
- Change streams
- セッション系コマンド

Amazon DocumentDB のエラスティッククラスター

このセクションのトピックでは、Amazon DocumentDB エラスティッククラスターを支えているメカニズムと機能について説明します。

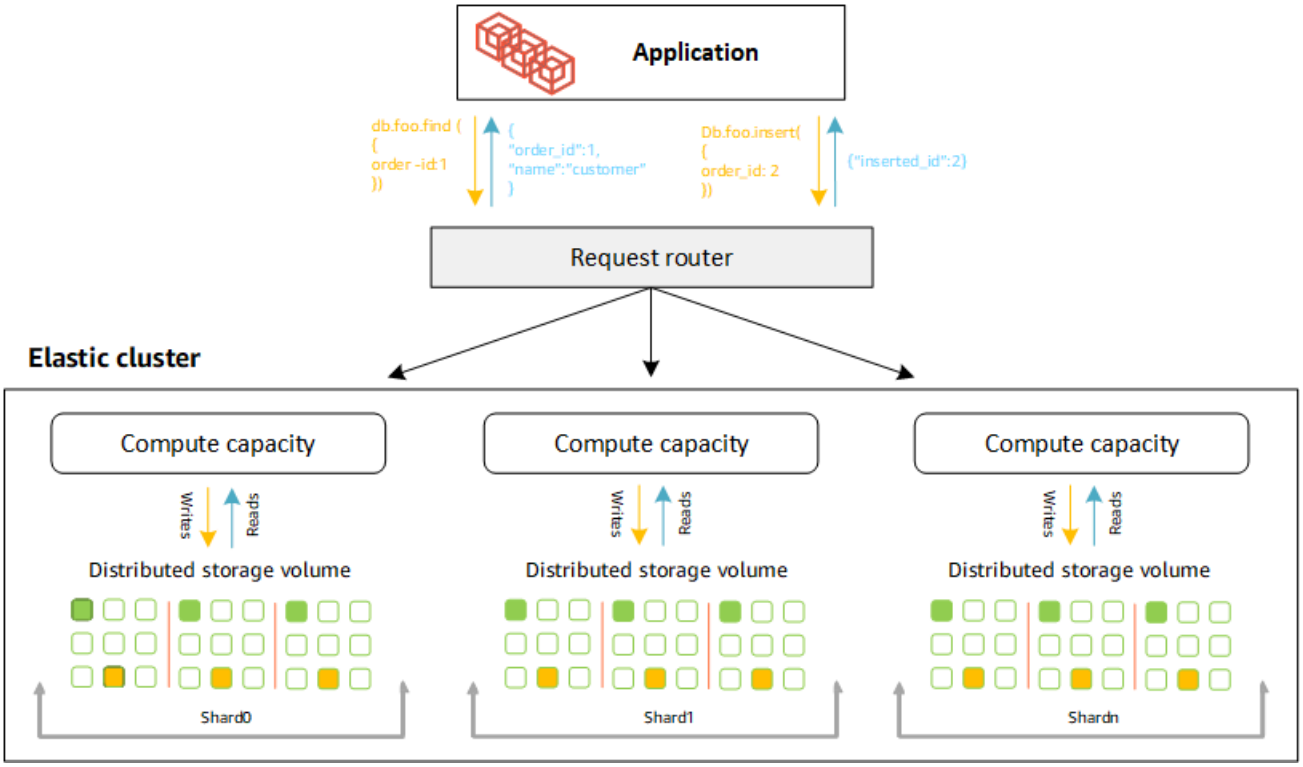
トピック

- Amazon DocumentDB エラスティッククラスターのシャーディング
- エラスティッククラスターの移行
- エラスティッククラスターのスケーリング
- エラスティッククラスターの信頼性
- エラスティッククラスターのストレージと可用性
- Amazon DocumentDB 4.0 と Elastic クラスター の機能上の違い

Amazon DocumentDB エラスティッククラスターのシャーディング

Amazon DocumentDB エラスティッククラスターは、ハッシュベースのシャーディングを使用して、分散型ストレージシステム全体にデータをパーティショニングします。シャーディングはパーティショニングとも呼ばれ、大きなデータセットを複数のノードにまたがる小さなデータセットに分割し、垂直スケーリングの限界を超えてデータベースをスケールアウトできるようにします。エラス

ティッククラスターでは、Amazon DocumentDB のコンピューティングとストレージを分離、つまり「デカップリング」することで、互いを別々にスケーリングできます。エラスティッククラスターは、コンピューティングノード間で小さなデータを移動してコレクションを再分割するのではなく、分散型ストレージシステム内でデータを効率的にコピーします。



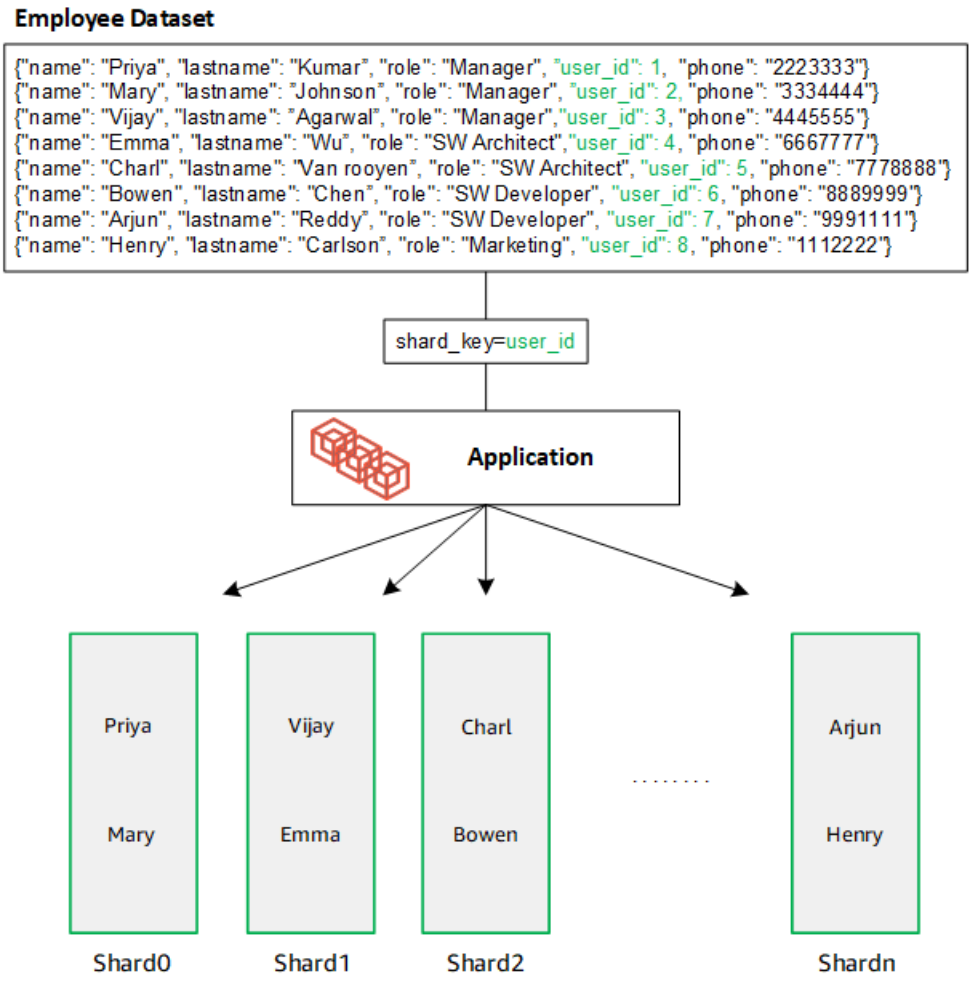
シャードの定義

以下はシャード命名法の定義です。

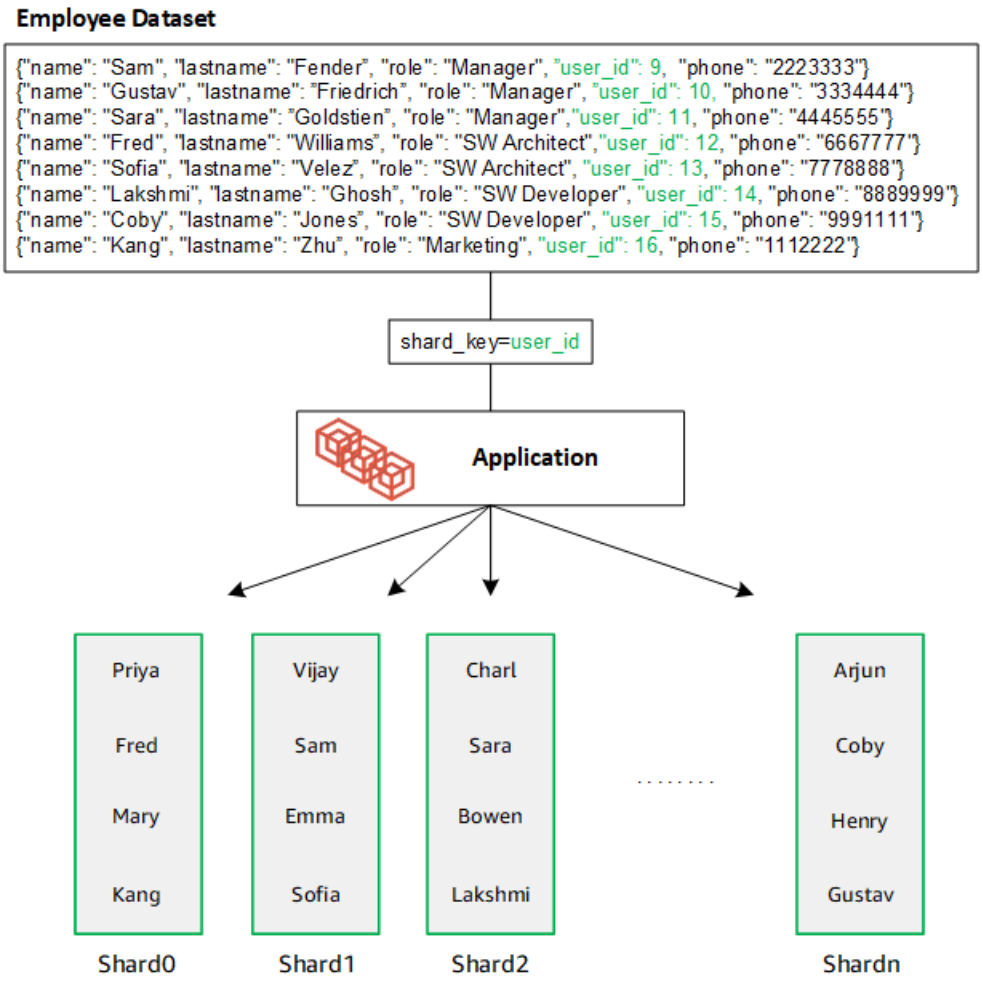
- シャード: - シャードは エラスティッククラスターのコンピューティング能力を提供します。1 つのライターインスタンスと 0 ~ 15 個のリードレプリカがあります。デフォルトでは、シャードにはライターと 1 つのリードレプリカの 2 つのインスタンスがあります。最大 32 個のシャードを設定でき、各シャードインスタンスには最大 64 個の vCPU を設定できます。
- シャードキー - シャードキーは、エラスティッククラスターが読み取りと書き込みトラフィックを分散するために使用する、シャードコレクション内の JSON ドキュメントの必須フィールドです。
- シャードコレクション - シャードコレクションは、データがデータパーティション内の 1 つのエラスティッククラスター全体に分散されるコレクションです。
- パーティション - パーティションはシャードされたデータの論理的な部分です。シャードコレクションを作成すると、データはシャードキーに基づいて各シャード内のパーティションに自動的に編成されます。1 つのシャードには複数のパーティションがあります。

設定された複数のシャードに対するデータの分散

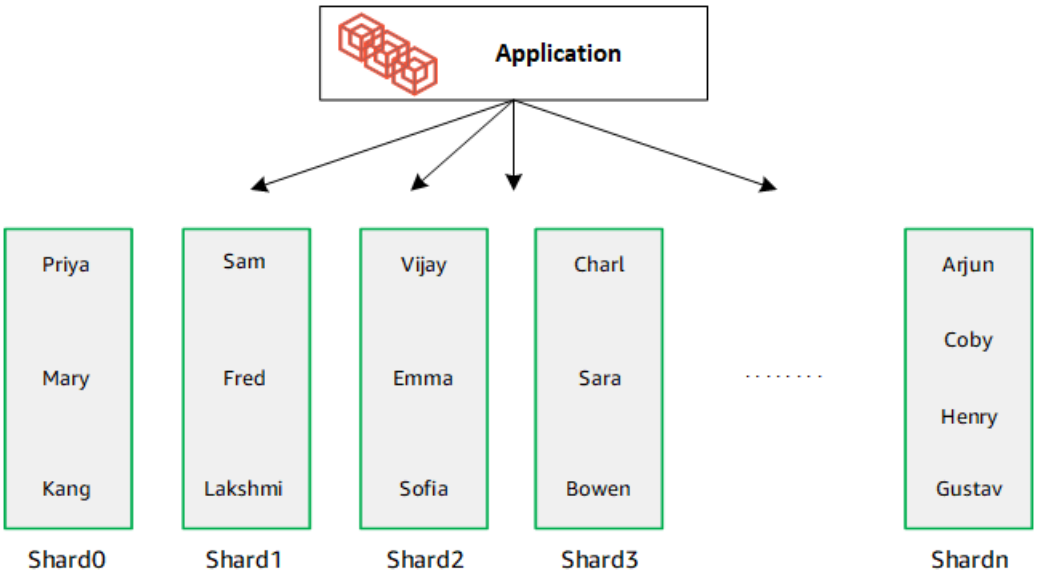
多数の一意の値を持つシャードキーを作成します。シャードキーが適切であると、その土台のシャード間でデータが均等にパーティショニングされ、ワークロードに最高のスループットとパフォーマンスを提供されます。次の例は、「user_id」という名前のシャードキーを使用する従業員名のデータです。



DocumentDB はハッシュシャーディングを使用して、土台のシャードの間でデータをパーティショニングします。追加データも同じ方法で挿入および分散されます。



シャードが追加されてデータベースがスケールアウトされると、Amazon DocumentDB はデータを自動的に再分散します。



エラスティッククラスターの移行

Amazon DocumentDB では、シャードされた MongoDB のデータをエラスティッククラスターに移行することができます。オフライン、オンライン、ハイブリッドの移行方法がサポートされています。詳細については、「Amazon DocumentDB の移行とアップグレード」を参照してください。

エラスティッククラスターのスケーリング

Amazon DocumentDB エラスティッククラスターでは、エラスティッククラスター内のシャードの数を増やしたり (スケールアウト)、各シャードに適用される vCPU の数を増やしたり (スケールアップ) することができます。必要に応じてシャードの数とコンピューティング能力 (vCPU) を減らすこともできます。

スケーリングのベストプラクティスについては、「エラスティッククラスターのスケーリング」を参照してください。

> **Note**
>
> クラスターレベルのスケーリングもできます。詳細については、「Amazon DocumentDB クラスターのスケーリング」を参照してください。

エラスティッククラスターの信頼性

Amazon DocumentDB は、信頼性、耐久性、および耐障害性を持つように設計されています。可用性を向上させるため、エラスティッククラスターは異なるアベイラビリティーゾーンに配置されたシャードごとに 2 つのノードを展開します。Amazon DocumentDB には、信頼性の高いデータベースソリューションとなるいくつかの自動機能も含まれています。詳細については、「Amazon DocumentDB の信頼性」を参照してください。

エラスティッククラスターのストレージと可用性

Amazon DocumentDB のデータは クラスターボリューム に保存されます。これは、SSD (Solid State Drive) を使用する単一の仮想ボリュームです。クラスターボリュームは、データの 6 つのコピーで構成され、1 つの AWS リージョン内の複数のアベイラビリティーゾーンに自動的にレプリケートされます。このレプリケーションにより、データの高い耐久性が保証され、データ損失の可能性が低くなります。また、データのコピーが他のアベイラビリティーゾーンにすでに存在するため、フェイルオーバー中のクラスターの可用性が高まります。ストレージ、高可用性、レプリケーションの詳細については、「Amazon DocumentDB の 仕組み」を参照してください。

Amazon DocumentDB 4.0 と Elastic クラスター の機能上の違い

Amazon DocumentDB 4.0 と エラスティッククラスターの間には、次のような機能上の違いがあります。

- top および collStats の結果はシャード別にパーティショニングされます。シャードコレクションの場合、データは複数のパーティションに分散され、collStats は、各パーティションから集約された collScans のレポートを作成します。
- シャードされたコレクションの top と collStats からコレクション統計は、クラスターのシャード数が変更されるとリセットされます。
- バックアップビルトインロールが serverStatus をサポートするようになりました。アクション - バックアップロールを持つデベロッパーおよびアプリケーションは、Amazon DocumentDB クラスターの状態に関する統計を収集できます。
- replSetGetConfig 出力では SecondaryDelaySecs フィールドが slaveDelay を置き換えます。
- hello コマンドは isMaster を置き換え、hello は エラスティッククラスターの役割を説明したドキュメント返します。
- エラスティッククラスターでは、 $elemMatch 演算子は配列の最初の入れ子レベルのドキュメントのみを一致検索します。Amazon DocumentDB 4.0 では、この演算子はすべてのレベルを横断して、一致したドキュメントを返します。例えば、次のようになります。

```
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 では、「$」プロジェクションはすべてのドキュメントとすべてのフィールドを返します。エラスティッククラスターでは、「$」プロジェクションが指定された find コマンドは、「$」プロジェクションと一致するフィールドのみを含むクエリパラメータに一致するドキュメントを返します。
- エラスティッククラスターでは、$regex および $options クエリパラメーターが指定された find コマンドは、「$regex と $options の両方にオプションを設定することはできません」というエラーを返します。

- エラスティッククラスターの場合、$indexOfCP は以下のいずれかの場合に 「-1」を返します。
  - string expression でサブ文字列が見つからない。
  - 数値 start は end より大きい。
  - 数値 start は文字列のバイト長より大きい。

  Amazon DocumentDB 4.0 の場合、$indexOfCP は位置 start の数値が end または文字列のバイト長よりも大きい場合に「0」を返します。
- エラスティッククラスターの場合、_id fields プロジェクトオペレーション (例: {"_id.nestedField" : 1}) はプロジェクトされたフィールドのみを含むドキュメントを返します。一方、Amazon DocumentDB 4.0 の場合、入れ子にされたフィールドプロジェクトコマンドではどのドキュメントも除外しません。

Amazon DocumentDB エラスティッククラスターの開始方法

このセクションでは、初めてのエラスティッククラスターを作成してクエリする方法を順を追って説明します。

Amazon DocumentDB に接続して開始する方法は複数あります。以下の手順は、ユーザーが私たちの強力なドキュメントデータベースの使用を開始するための最も迅速、シンプルかつ簡単な方法です。このガイドでは、AWS CloudShell を使用して、 AWS マネジメントコンソールから直接 Amazon DocumentDB クラスターに接続してクエリを実行します。 AWS 無料利用枠の対象となる

新規のお客様は、Amazon DocumentDB と CloudShell を無料で使用できます。 AWS CloudShell 環境または Amazon DocumentDB エラスティッククラスターが無料利用枠を超えるリソースを使用している場合は、それらのリソースの通常の AWS 料金が課金されます。このガイドでは、5 分以内に Amazon DocumentDB の使用を開始します。

トピック

- 前提条件
- ステップ 1: エラスティッククラスターを作成する
- ステップ 2: エラスティッククラスターに接続する
- ステップ 3: コレクションをシャーディングし、データを挿入してクエリする
- ステップ 4: 探索する

前提条件

最初の Amazon DocumentDB クラスターを作成する前に、以下の操作を行う必要があります。

Amazon Web Services (AWS) アカウントを作成する

Amazon DocumentDB を使用する前に、Amazon Web Services (AWS) アカウントを持っている必要があります。 AWS アカウントは無料です。使用しているサービスとリソースに対してのみ料金をお支払いいただきます。

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://signin.aws.amazon.com/signup を開きますか?request_type=register。
2. オンラインの手順に従います。

   サインアップ手順の一環として、電話またはテキストメッセージを受け取り、電話キーパッドで検証コードを入力します。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザー が作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティベストプラクティスとして、ユーザーに管理アクセス権を割り当て、ルートユーザーアクセスが必要なタスクの実行にはルートユーザーのみを使用するようにしてください。

必要な AWS Identity and Access Management (IAM) アクセス許可を設定します。

クラスター、インスタンス、クラスターパラメータグループなどの Amazon DocumentDB リソースを管理するためのアクセスには、 がリクエストの認証 AWS に使用できる認証情報が必要です。詳細については、「Amazon DocumentDB の Identity and Access Management」を参照してください。

1. の検索バーで AWS マネジメントコンソール、ドロップダウンメニューで IAM **IAM**を入力して選択します。
2. IAM コンソールにアクセスしたら、ナビゲーションペインから [ユーザー] を選択します。
3. ユーザーネームを選択します。
4. [Add permissions (許可の追加)] を選択します。
5. [ポリシーを直接アタッチする] を選択します。
6. 検索バーに `AmazonDocDBElasticFullAccess` と入力し、検索結果に表示されたら、それを選択します。
7. [次へ] を選択します。
8. [Add permissions (許可の追加)] を選択します。

> **Note**
>
> AWS アカウントには、各リージョンにデフォルトの VPC が含まれます。Amazon VPC を使用することを選択した場合は、「Amazon VPC ユーザーガイド」の「Amazon VPC を作成する」トピックの手順を完了します。

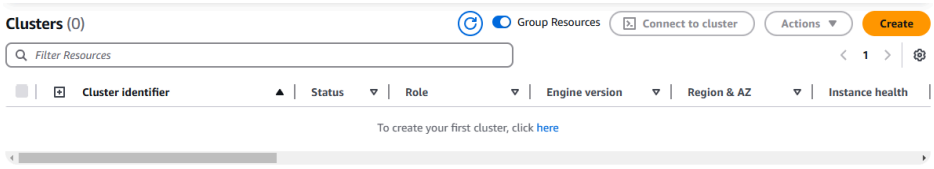
ステップ 1: エラスティッククラスターを作成する

このセクションでは、 AWS マネジメントコンソール または を使用して、以下の手順 AWS CLI で新しい Elastic クラスターを作成する方法について説明します。

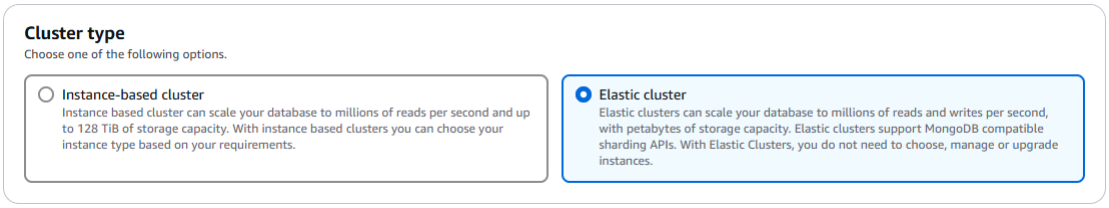
Using the AWS マネジメントコンソール

AWS マネジメントコンソールを使用して エラスティッククラスター設定を作成するには、以下の手順に従います。

1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. Amazon DocumentDB マネジメントコンソールの [クラスター] で [作成] を選択します。



3. [Amazon DocumentDB クラスターの作成] ページの [クラスタータイプ] セクションで、[エラスティッククラスター] を選択します。



4. [設定] セクションで、次を実行します:

   a. クラスター名 フィールドに、一意のクラスター識別子を入力します (フィールドの下の命名要件に従います)。

   b. [シャード数] フィールドに、クラスターに必要なシャードの数を入力します。クラスターあたりのシャードの最大数は 32 です。

   > **Note**
   >
   > シャードごとに 2 つのノードが展開されます。両方のノードのシャード容量は同じになります。

   c. [シャードインスタンス数] フィールドで、各シャードに関連付けるレプリカインスタンスの数を選択します。シャードインスタンスの最大数は 16 で、1 刻みで設定できます。すべてのレプリカインスタンスのシャード容量は、次のフィールドで定義するものと同じになります。テスト目的では、デフォルト値の 2 で十分です。

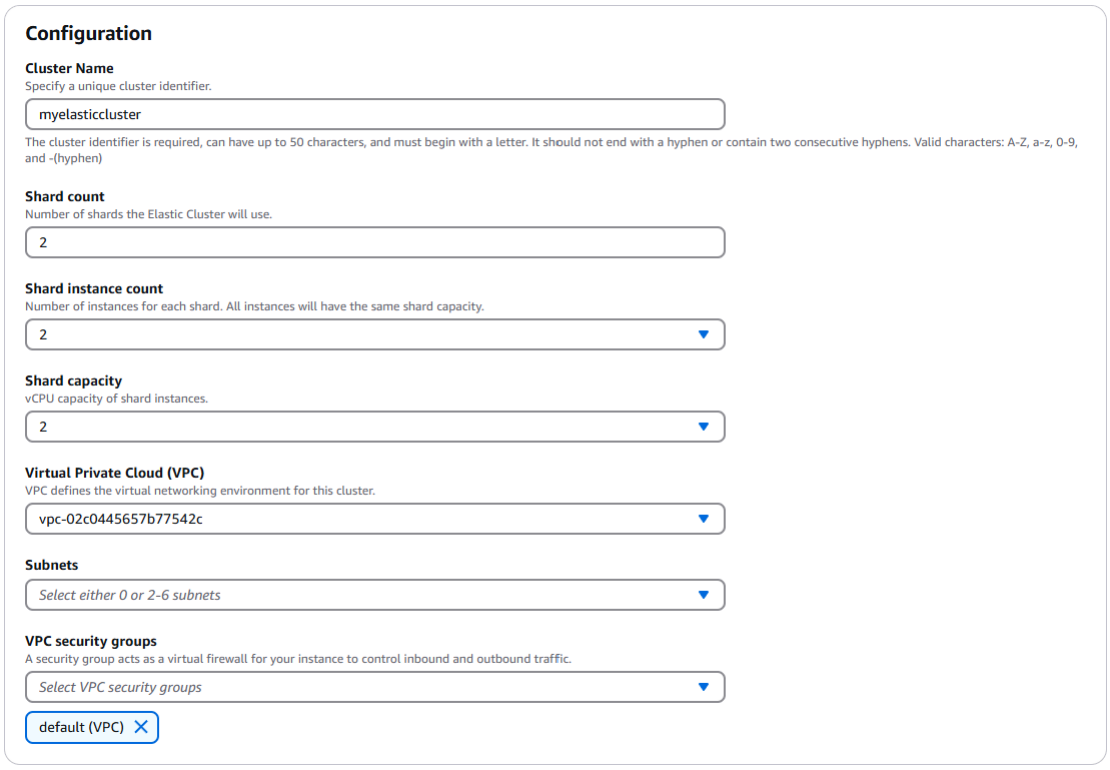
   > **Note**
   >
   > レプリカインスタンスの数は、Elastic クラスター内のすべてのシャードに適用されます。シャードインスタンス数の値が 1 の場合、ライターインスタンスが 1

つあり、それ以外の追加のインスタンスはすべてレプリカで、読み取りや可用性の向上に使用できます。テスト目的では、デフォルト値の 2 で十分です。

d. [シャード容量] フィールドで、各シャードインスタンスに関連付ける仮想 CPU (vCPU) の数を選択します。1 シャードインスタンスあたりの vCPU の最大数は 64 です。使用できる値は、2、4、8、16、32、64 です。テスト目的では、デフォルト値の 2 で十分です。

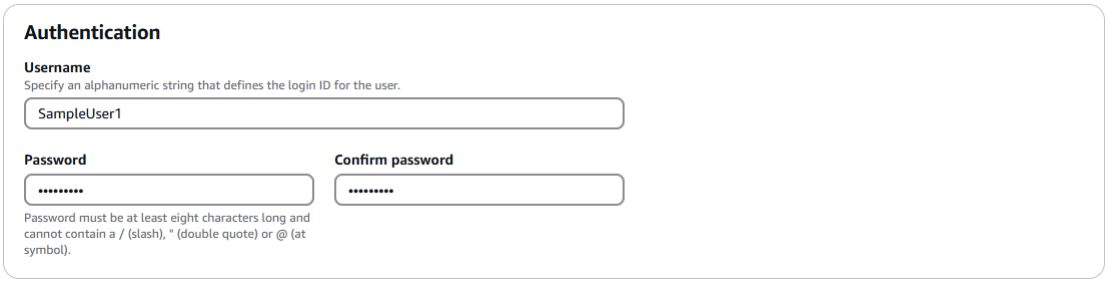
e. [仮想プライベートクラウド (VPC)] フィールドのドロップダウンリストから VPC を選択します。

f. [サブネット] と [VPC セキュリティグループ] では、デフォルトを使用するか、選択した 3 つのサブネットと最大 3 つの VPC セキュリティグループ (最低 1 つ) を選択できます。



5. [認証] セクションの [ユーザー名] フィールドで、プライマリユーザーのログイン名を示す文字列を入力します。

   [パスワード] フィールドに、指示に従った一意のパスワードを入力して確認します。



6. [暗号化] セクションは、デフォルト設定 (デフォルトキー) をそのまま使用します。

   必要に応じて、作成した AWS KMS key ARN を入力できます。詳細については、「Amazon DocumentDB エラスティッククラスターで保管中のデータ暗号化」を参照してください。

   > **Important**
   >
   > エラスティッククラスターでは暗号化を有効にする必要があります。

7. [バックアップ] セクションで、該当するバックアップ要件に従ってフィールドを編集します。テスト目的で、デフォルト設定を保持できます。



   a. [バックアップ保持期間] - リストで、このクラスターの自動バックアップを削除するまで保持する日数を選択します。

   b. [バックアップウィンドウ] - Amazon DocumentDB によってこのクラスターのバックアップが作成される毎日の時刻と期間を設定します。

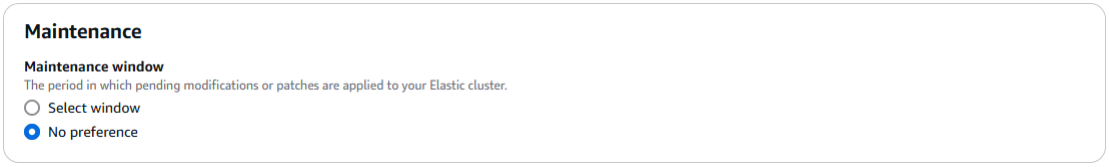
      i. バックアップの作成タイミングと期間を設定したい場合は、[ウィンドウを選択] を選択します。

         [開始時刻] - 最初のリストで、自動バックアップを開始する時間 (UTC 時) を選択します。2 番目のリストで、自動バックアップを開始する時間 (UTC 分) を選択します。

         [期間] - リストで、自動バックアップの作成に割り当てる時間を選択します。

ii. Amazon DocumentDB 動作に従ってバックアップの作成タイミングや期間を自動選択する場合は、[設定なし] を選択します。

8. [メンテナンス] セクションで、クラスターに変更やパッチを適用する日と時間、期間を選択します。テスト目的で、デフォルト設定を保持できます。



9. [クラスターを作成] を選択します。

エラスティッククラスターは現在プロビジョニング中です。このプロセスには数分かかることがあります。[クラスター] リストに エラスティッククラスターのステータスが 使用可能 と表示されている場合は、クラスターに接続できます。

Using the AWS CLI

を使用して Elastic クラスターを作成するには AWS CLI、以下のパラメータを指定して `create-cluster`オペレーションを使用します。

- --cluster-name — 必須 作成時に入力した、または最終変更時のエラスティックスケールクラスターの現在の名前。
- --shard-capacity — 必須 各シャードに割り当てられた vCPU の数。最大数は 64 です。使用できる値は、2、4、8、16、32、64 です。
- --shard-count — 必須 クラスターに割り当てられたシャードの数。最大数は 32 です。
- --shard-instance-count - オプションです。このクラスター内のすべてのシャードに適用されるレプリカインスタンスの数。最大数は 16 です。
- --admin-user-name — 必須 管理者ユーザーに関連付けられているユーザー名。
- --admin-user-password — 必須 管理者ユーザーに関連付けられているパスワード。
- --auth-type — 必須 エラスティッククラスターへのアクセスに使用するパスワードの取得先の特定に使用している認証タイプ。有効なタイプは、`PLAIN_TEXT` または `SECRET_ARN` です。
- --vpc-security-group-ids - オプションです。このクラスターに関連付ける EC2 VPC セキュリティグループのリストを設定します。
- --preferred-maintenance-window - オプションです。週 1 回のシステムメンテナンスを実行できる時間帯 (世界標準時 (UTC)) を設定します。

  形式は `ddd:hh24:mi-ddd:hh24:mi` です。有効な曜日 (ddd): 月、火、水、木、金、土、日

デフォルトは、1 週間のうちのランダムな日に起こる、Amazon Web Services リージョンあたり 8 時間の範囲からランダムに選択された 30 分のウィンドウです。

最小 30 分のウィンドウ。

- --kms-key-id - オプションです。暗号化されたクラスターの KMS キー識別子を設定します。

  KMS キー識別子は、 AWS KMS 暗号化キーの Amazon リソースネーム (ARN) です。新しいクラスターの暗号化に使用する KMS 暗号化キーを所有する Amazon Web Services アカウントと同じアカウントを使用してクラスターを作成する場合、KMS 暗号化キーの ARN の代わりに KMS キーエイリアスを使用できます。

  KmsKeyId で暗号化キーが指定がなく、かつ `StorageEncrypted` パラメータが true の場合、Amazon DocumentDB はデフォルトの暗号化キーを使用します。
- --preferred-backup-window - オプションです。毎日自動バックアップを作成する希望の時間帯。デフォルトは、それぞれ 8 時間の時間ブロックからランダムに選択された 30 分のウィンドウです AWS リージョン。
- --backup-retention-period - オプション。自動バックアップを保管する日数。デフォルト値は 1 です。
- --storage-encrypted - オプションです。クラスターが暗号化されているかどうかを設定します。

  `--no-storage-encrypted` はクラスターが暗号化されていない指定です。
- --subnet-ids - オプションです。ネットワークのサブネット ID を設定します。

次の例では、各 *############* を独自の情報に置き換えます。

> **Note**
>
> 以下の例には、特定の KMS キーの作成例が含まれています。デフォルトの KMS キーを使用するには、`--kms-key-id` パラメータを含めないでください。

Linux、macOS、Unix の場合:

```
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
```

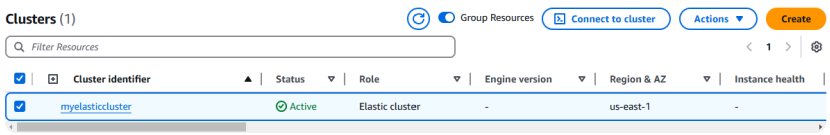
Windows の場合:

```
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
```

ステップ 2: エラスティッククラスターに接続する

を使用して Amazon DocumentDB エラスティッククラスターに接続します AWS CloudShell。

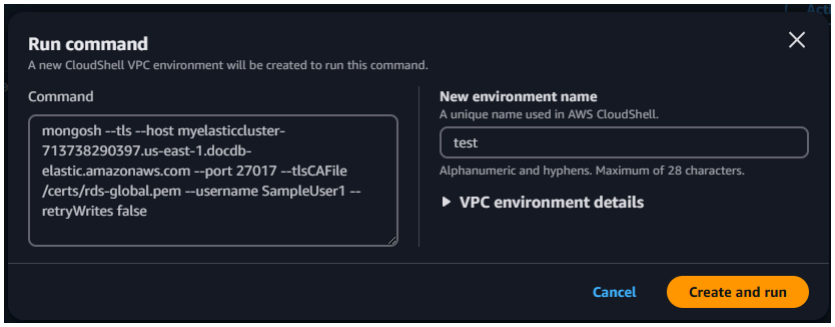
1. Amazon DocumentDB マネジメントコンソールの [クラスター] で、作成したエラスティッククラスターを探します。クラスターの横にあるチェックボックスをクリックして選択します。



2. クラスターに接続する (アクションドロップダウンメニューの横にある を選択します。このボタンは、クラスターの横にあるチェックボックスをクリックし、クラスターのステータスが 使

用可能 と表示された後にのみ有効になります。CloudShell の [Run コマンド] 画面が表示されます。

3. [新しい環境の名前] フィールドに、「test」などの一意の名前を入力し、[作成して実行する] を選択します。VPC 環境の詳細は、Amazon DocumentDB データベースに自動的に設定されます。



4. プロンプトが表示されたら、ステップ 1: Amazon DocumentDB エラスティッククラスターを作成する (サブステップ 5) で作成したパスワードを入力します。



パスワードを入力して、プロンプトが `direct: mongos] <env-name>>` に変わったら、Amazon DocumentDB クラスターに正常に接続できています

> **Note**
>
> トラブルシューティングについては、「Amazon DocumentDB のトラブルシューティング」を参照してください。

ステップ 3: コレクションをシャーディングし、データを挿入してクエリする

エラスティッククラスターでは Amazon DocumentDB でのシャーディングのサポートが追加されています。クラスターに接続できたので、クラスターをシャーディングし、データを挿入して、クエリを実行できます。

1. コレクションをシャーディングするには、次のように入力します。

```
sh.shardCollection("db.Employee1" , { "Employeeid" : "hashed" })
```

2. 1 つのドキュメントを挿入するには、次のように入力します。

```
db.Employee1.insertOne({"Employeeid":1, "Name":"Joe", "LastName": "Bruin", "level":
 1 })
```

   以下のような出力が表示されます。

```
WriteResult({ "nInserted" : 1 })
```

3. findOne() コマンドで書き込んだドキュメントを読み取るには、 コマンドを入力します (コマンドから一つのドキュメントが返される)。

```
db.Employee1.findOne()
```

   以下のような出力が表示されます。

```
{
"_id" : ObjectId("61f344e0594fe1a1685a8151"),
"EmployeeID" : 1,
"Name" : "Joe",
"LastName" : "Bruin",
"level" : 1
}
```

4. さらにクエリを実行するには、ゲームプロファイルのユースケースを検討してみてください。最初に、「Employee」というタイトルのコレクションにエントリをいくつか挿入します。次のように入力します。

```
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 }
])
```

以下のような出力が表示されます。

```
{ acknowledged: true,
     insertedIds: {
        '0': ObjectId('679d02cd6b5a0581be78bcbd'),
        '1': ObjectId('679d02cd6b5a0581be78bcbe'),
        '2': ObjectId('679d02cd6b5a0581be78bcbf'),
        '3': ObjectId('679d02cd6b5a0581be78bcc0')
    }
}
```

5. プロファイルコレクション内のすべてのドキュメントを取得するには、find() コマンドを使用します。

```
db.Employee.find()
```

ステップ 4 で入力したデータが表示されます。

6. 1 つのドキュメントに対してクエリを発行するには、フィルター (「Katie」など) を含めます。次のように入力します。

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

以下のような出力が表示されます。

```
[

   {

     _id: ObjectId('679d02cd6b5a0581be78bcc0'),

     Employeeid: 4,
```

```
     name: 'Katie',

     lastname: 'Schaper',

     level: 3

   }

]
```

7. プロファイルを見つけて変更するには、findAndModify コマンドを入力します。この例では、従業員「Matt」にはより高いレベル「14」が割り当てられています。

```
db.Employee.findAndModify({
    query: { "Employeeid" : 1, "name" : "Matt"},
    update: { "Employeeid" : 1, "name" : "Matt", "lastname" : "Winkle", "level" :
 14 }
})
```

以下のような出力が表示されます (レベルはまだ変更されていないことに注意してください)。

```
{

   _id: ObjectId('679d02cd6b5a0581be78bcbd'),

   Employeeid: 1,

   name: 'Matt',

   lastname: 'Winkle',

   level: 12

}
```

8. レベルが上がったことを確認するには、次のクエリを入力します。

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

以下のような出力が表示されます。

```
[
```

```
    {

    _id: ObjectId('679d02cd6b5a0581be78bcbd'),

    Employeeid: 1,

    name: 'Matt',

    lastname: 'Winkle',

    level: 14

    }
]
```

ステップ 4: 探索する

お疲れ様でした。Amazon DocumentDB エラスティッククラスターの開始手順が正常に完了しました。

次のステップ このデータベースを一般的な機能の一部を使用して、完全に活用する方法を学びましょう。

- Amazon DocumentDB のエラスティッククラスターのベストプラクティス
- Amazon DocumentDB エラスティッククラスターの管理

> **Note**
>
> この開始手順で作成したエラスティッククラスターは、削除しない限り、引き続きコストを計上します。手順については、「Elastic クラスターの削除」を参照してください。

Amazon DocumentDB のエラスティッククラスターのベストプラクティス

Amazon DocumentDB のエラスティッククラスターを使用するためのベストプラクティスについて説明します。インスタンスベースの Amazon DocumentDB クラスターのベストプラクティスはすべ

て、エラスティッククラスターに適用されます。新しいベストプラクティスが確認されると、このセクションは更新されます。

トピック

- シャードキーの選択
- 接続管理
- 非シャード型コレクション
- エラスティッククラスターのスケーリング
- エラスティッククラスターのモニタリング

シャードキーの選択

以下は、シャードキーの作成に関するガイドラインです。

- クラスター内のすべてのシャードにデータを分散するには、均等分散型のハッシュキーを使用する(ホットキーの使用は避ける)。
- スキャッターギャザークエリを回避するには、読み取り/更新/削除リクエストでシャードキーを使用する。
- 読み取り/更新/削除操作では、入れ子型のシャードキーの使用を避ける。
- バッチ操作では、すべてのシャードが並列実行され、レイテンシーが改善されるよう `ordered` を false に設定する。

接続管理

以下は、データベースへの接続管理に関するガイドラインです。

- 接続数、および、新しい接続の開閉の頻度をモニタリングする。
- アプリケーションの設定のすべてのサブネットに接続を分散する。複数のサブネットにクラスターを設定しているにもかかわらず、サブネットの一部しか使用されないと、最大接続数でボトルネックが発生する可能性があります。

非シャード型コレクション

以下は、非シャード型コレクションに関するガイドラインです。

- 非シャード型コレクションで負荷を分散するには、使用率の高い非シャード型コレクションを複数のデータベースに保存するようにする。Amazon DocumentDB のエラスティッククラスターは、データベースを複数のシャードに配置します。この時、同じデータベースの非シャード型コレクションは、同じシャードに配置されます。

エラスティッククラスターのスケーリング

以下は、エラスティッククラスターのスケーリングに関するガイドラインです。

- スケーリング操作によって、データベースエラーやネットワークエラーが短時間断続的に発生する場合があります。可能な限り、ピーク時間帯のスケーリングは避けてください。スケーリングは、メンテナンスウィンドウ中に行うようにしてください。
- コンピュートを増やす場合は、シャード数を増減するよりも、シャードの容量を増減 (各シャードの vCPU 数の変更) することをお勧めします。高速であり、データベースやネットワークの断続的なエラーが発生する時間が短くなります。
- 成長が見込まれる場合は、シャード容量をスケールするではなく、シャード数を増やすことをお勧めします。シャード容量を増やしてクラスターをスケールすることで、迅速なスケーリングが必要とされる状況に対応できます。
- クライアント側の再試行ポリシーをモニタリングして、エクスポネンシャルバックオフとジッターを試し、スケーリング中にエラーが発生したときにデータベースが過負荷にならないようにしてください。
- シャードの数を変更すると、データをシャード間で安全に移動する必要があるため、時間がかかる場合があります。エラーのリスクを最小限に抑えるために、変更の進行中にクラスターの状態を停止したり、他の変更を加えたりしないでください。

エラスティッククラスターのモニタリング

以下は、エラスティッククラスターのモニタリングに関するガイドラインです。

- 各シャードのメトリクスのピーク対平均比を追跡して、トラフィックが不均等 (ホットキー/ホットスポットがある) になっていないか判断してください。ピーク対平均比を追跡するための主なメトリクスは次のとおりです。
  - `PrimaryInstanceCPUUtilization`
    - これはシャード単位にモニタリングできます。
    - クラスターレベルでは、平均から p99 までのスキューをモニタリングできます。

- `PrimaryInstanceFreeableMemory`
  - これはシャード単位にモニタリングできます。
  - クラスターレベルでは、平均から p99 までのスキューをモニタリングできます。
- `DatabaseCursorsMax`
  - スキューを判断するには、シャード単位でモニタリングする必要があります。
- `Documents-Inserted/Updated/Returned/Deleted`
  - スキューを判断するには、シャード単位でモニタリングする必要があります。

Amazon DocumentDB エラスティッククラスターの管理

Amazon DocumentDB Elestic クラスターを管理するには、適切な Amazon DocumentDB コントロールプレーンアクセス許可を持つ IAM ポリシーが必要です。これらのアクセス許可により、クラスターとインスタンスを作成、変更、削除できます。Amazon DocumentDBFullAccess ポリシーは、Amazon DocumentDB Elastic クラスターを管理するために必要なすべてのアクセス許可を提供します。

以下のトピックでは、Amazon DocumentDB Elastic クラスターを操作するときのさまざまなタスクの実行方法を示します。

トピック

- Elastic クラスター設定の変更
- Elastic クラスターのモニタリング
- Elastic クラスターの削除
- Elastic クラスタースナップショットの管理
- Amazon DocumentDB Elastic クラスターの停止と開始
- Amazon DocumentDB エラスティッククラスターのメンテナンス

Elastic クラスター設定の変更

このセクションでは、 AWS マネジメントコンソール または を使用して Elastic クラスターを変更する方法を、次の手順 AWS CLI で説明します。

クラスターを変更する主な用途は、シャード数やシャードの計算能力を増減してシャードをスケールすることです。

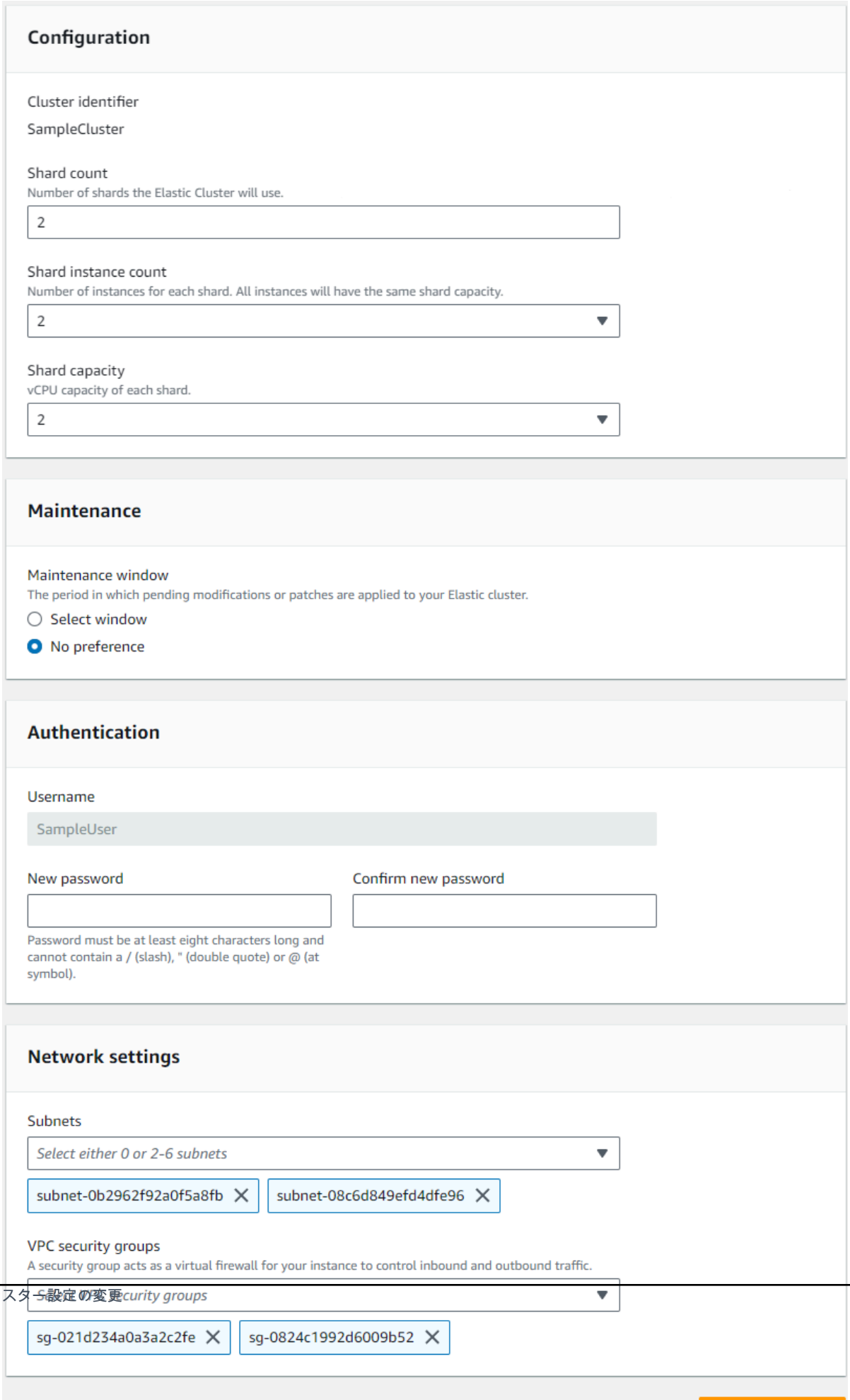
Using the AWS マネジメントコンソール

を使用して Elastic クラスターの設定を変更するには AWS マネジメントコンソール:

1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ナビゲーションペインの左上隅にあるメニューアイコンを選択します。

3. [クラスター識別子] 列で、変更するクラスターの名前を選択します。
4. [Modify] (変更) を選択します。
5. 変更するフィールドを編集し、[クラスターの変更] を選択します。


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 security groups
sg-021d234a0a3a2c2fe
sg-0824c1992d6009b52

> **Note**
>
> または、[クラスター] ページに移動し、該当するクラスターの横にあるボックスにチェックを入れ、[アクション]、[変更] の順に選択して [クラスターの変更] ダイアログにアクセスします。

Using the AWS CLI

を使用して Elastic クラスターの設定を変更するには AWS CLI、以下のパラメータを指定して `update-cluster`オペレーションを使用します。

- **`--cluster-arn`** — 必須 変更するクラスターの ARN 識別子。
- **`--shard-capacity`** - オプションです。各シャードに割り当てられた vCPU の数。最大数は 64 です。使用できる値は、2、4、8、16、32、64 です。
- **`--shard-count`** - オプションです。クラスターに割り当てられたシャードの数。最大数は 32 です。
- **`--shard-instance`** 数 — オプション。このクラスター内のすべてのシャードに適用されるレプリカインスタンスの数。最大数は 16 です。
- **`--auth-type`** - オプションです。エラスティッククラスターへのアクセスに使用するパスワードの取得先の特定に使用している認証タイプ。有効なタイプは、`PLAIN_TEXT` または `SECRET_ARN` です。
- **`--admin-user-password`** - オプションです。 – 管理者ユーザーに関連付けられているパスワード。
- **`--vpc-security-group-ids`** - オプションです。このクラスターに関連付ける Amazon EC2 と Amazon 仮想プライベートクラウド (VPC) セキュリティグループのリストを設定します。
- **`--preferred-maintenance-window`** - オプションです。週 1 回のシステムメンテナンスを実行できる時間帯 (世界標準時 (UTC))を設定します。

  形式は `ddd:hh24:mi-ddd:hh24:mi` です。有効な曜日 (ddd): 月、火、水、木、金、土、日

  デフォルトは、1 週間のうちのランダムな日に起こる、Amazon Web Services リージョンあたり 8 時間の範囲からランダムに選択された 30 分のウィンドウです。

  最小 30 分のウィンドウ。
- **`--subnet-ids`** - オプションです。ネットワークのサブネット ID を設定します。

次の例では、各 ############## を独自の情報に置き換えます。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

変更後の Elastic クラスターのステータスをモニタリングするには、「Elastic クラスターのモニタリング」を参照してください。

Elastic クラスターのモニタリング

このセクションでは、 AWS マネジメントコンソール または を使用して Elastic クラスターをモニタリングする方法を次の手順 AWS CLI で説明します。

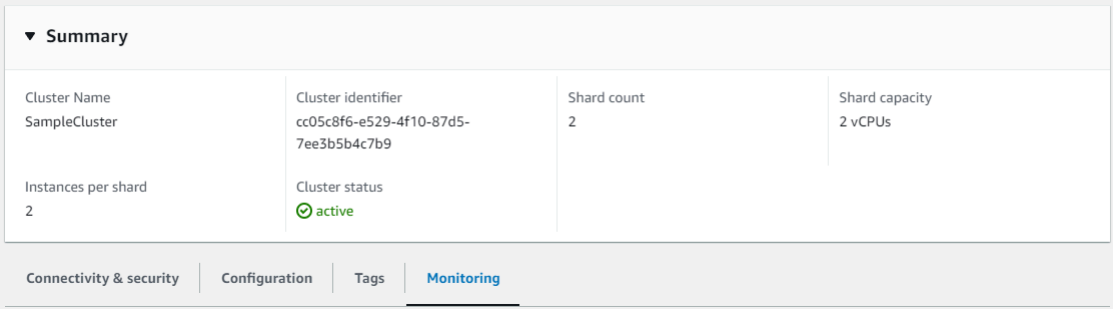
Using the AWS マネジメントコンソール

を使用して Elastic クラスター設定をモニタリングするには AWS マネジメントコンソール:

1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ナビゲーションペインの左上隅にあるメニューアイコンを選択します。

3. [クラスター識別子] 列で、モニタリングするクラスターの名前を選択します。
4. [モニタリング] タブを選択します。



Amazon CloudWatch からの多くのチャートは、以下のモニタリングカテゴリについて表示されます。

- リソース使用率
- スループット
- オペレーション
- システム

を通じて Amazon CloudWatch にアクセスして AWS マネジメントコンソール 、Elastic クラスター用に独自のモニタリング環境を設定することもできます。

Using the AWS CLI

を使用して特定の Elastic クラスター設定をモニタリングするには AWS CLI、以下のパラメータを指定して `get-cluster`オペレーションを使用します。

- **`--cluster-arn`** — 必須 情報が必要なクラスターの識別子。

次の例では、各 `##############` を独自の情報に置き換えます。

Linux、macOS、Unix の場合:

```
aws docdb-elastic get-cluster \
    --cluster-arn arn:aws:docdb-elastic:us-west-2:123456789012:cluster:/68ffcdf8-
e3af-40a3-91e4-24736f2dacc9
```

Windows の場合:

```
aws docdb-elastic get-cluster ^
    --cluster-arn arn:aws:docdb:-elastic:us-west-2:123456789012:cluster:/68ffcdf8-
e3af-40a3-91e4-24736f2dacc9
```

このオペレーションによる出力は、次のようになります。

```
"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",
        ...
 }
```

詳細については、「Amazon DocumentDB リソース管理 API リファレンス」の「DescribeClusterSnapshot」を参照してください。

を使用してすべての Elastic クラスターの詳細を表示するには AWS CLI、以下のパラメータを指定して list-clustersオペレーションを使用します。

- **--next-token** - オプションです。出力される項目数 (--max-results) が基本の API コールによって返される合計項目数より少ない場合、出力には NextToken が含まれ、これにより、後続のコマンドを渡して、次の項目のセットを取得できます。

- **--max-results** - オプションです。コマンドの出力で返される項目の総数。指定された max-results の値よりも多くの結果が存在する場合、ページ割りトークン (next-token) がレスポンスに含まれるため、残りの結果を取得できます。
  - デフォルト: 100
  - 最小は 20、最大は 100 です

次の例では、各 ############## を独自の情報に置き換えます。

Linux、macOS、Unix の場合:

```
aws docdb-elastic list-clusters \
    --next-token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ== \
    --max-results 2
```

Windows の場合:

```
aws docdb-elastic list-clusters ^
    --next-token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ== ^
    --max-results 2
```

このオペレーションによる出力は、次のようになります。

```
{
   "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"
       }
```

```
    ]
}
```

Elastic クラスターの削除

このセクションでは、 AWS マネジメントコンソール または を使用して、次の手順 AWS CLI で Elastic クラスターを削除する方法について説明します。

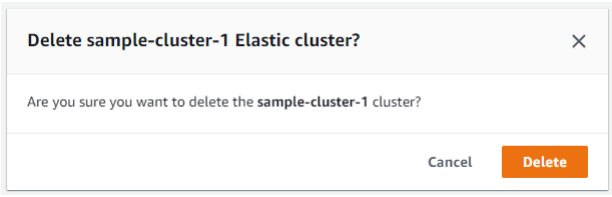
Using the AWS マネジメントコンソール

AWS マネジメントコンソールを使用して Elastic クラスター設定を削除するには、以下の手順に従います。

1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ナビゲーションペインの左上隅にあるメニューアイコンを選択します。

3. クラスターリストの表で、削除するクラスター名の左にあるチェックボックスをオンにし、[アクション] を選択します。ドロップダウンメニューから [削除] を選択します。
4. [「cluster-name」の Elastic クラスターを削除しますか?] ダイアログボックスで [削除] を選択します。



クラスターが削除されるまでに数分かかります。クラスターのステータスをモニタリングするには、「Amazon DocumentDB クラスターのステータスのモニタリング」を参照してください。

Using the AWS CLI

を使用して Elastic クラスターを削除するには AWS CLI、次のパラメータを指定して `delete-cluster`オペレーションを使用します。

- **--cluster-arn** — 必須 削除するクラスターの ARN 識別子。
- **--no-skip-final-backup** - オプションです。最終バックアップを作成する場合は、最終バックアップの名前を指定して、このパラメータを含める必要があります。--final-backup-identifier または --skip-final-backup のどちらかを含める必要があります。
- **--skip-final-backup** - オプションです。クラスターを削除する前に最終バックアップを作成しない場合のみ、このパラメータを使用します。デフォルトでは、最終スナップショットを作成します。

次の AWS CLI コード例では、ARN が arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-cluster のクラスターを最終バックアップで削除します。

次の例では、各 *############* を独自の情報に置き換えます。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

次の AWS CLI コード例では、ARN が arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-cluster のクラスターを最終バックアップなしで削除します。

次の例では、各 *############* を独自の情報に置き換えます。

Linux、macOS、Unix の場合:

```
aws docdb-elastic delete-cluster \
```

```
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster \
    --skip-final-backup \
```

Windows の場合:

```
aws docdb-elastic delete-cluster ^
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster ^
    --skip-final-backup ^
```

`delete-cluster` オペレーションの出力は、削除するクラスターの表示になります。

クラスターが削除されるまでに数分かかります。クラスターのステータスをモニタリングするには、「Amazon DocumentDB クラスターのステータスのモニタリング」を参照してください。

Elastic クラススナップショットの管理

手動スナップショットは、Elastic クラスターの作成後に作成できます。自動バックアップは、Elastic クラススナップショットが作成されると直ちに作成されます。

> **Note**
>
> 手動スナップショットを作成するには、クラスターのスタータスが `Available` である必要があります。

このセクションでは、Elastic クラススナップショットを作成、表示、復元、削除する方法について説明します。

以下のトピックでは、Amazon DocumentDB Elastic クラススナップショットを操作するときのさまざまなタスクの実行方法を示します。

トピック

- 手動 Elastic クラススナップショットの作成
- Elastic クラススナップショットの表示
- スナップショットからの Elastic クラスターの復元
- Elastic クラススナップショットのコピー

- Elastic クラスタースナップショットの削除
- Elastic クラスタースナップショットの自動バックアップの管理

手動 Elastic クラスタースナップショットの作成

このセクションでは、 AWS マネジメントコンソール または を使用して、次の手順 AWS CLI で手動 Elastic クラスタースナップショットを作成する方法について説明します。

Using the AWS マネジメントコンソール

AWS マネジメントコンソールを使用して手動 Elastic クラスタースナップショットを作成するには、以下の手順に従います。

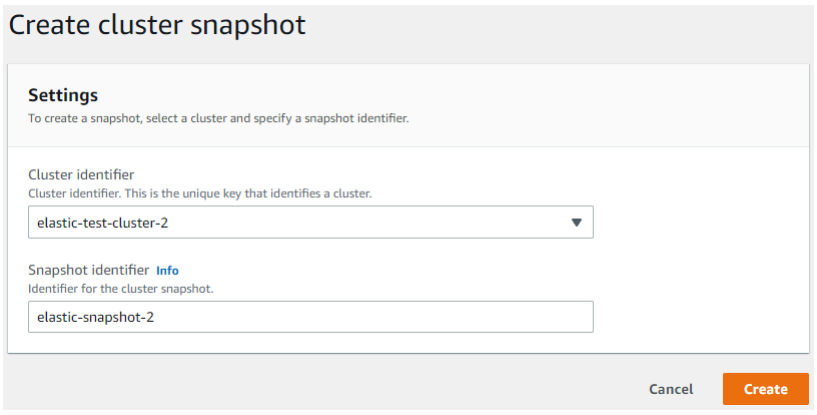
1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[Snapshots] を選択してください。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ナビゲーションペインの左上隅にあるメニューアイコンを選択します。

3. [スナップショット] ページで、[作成] を選択します。
4. [クラスタースナップショットの作成] ページの [クラスター識別子] フィールドで、ドロップダウンリストからご使用の Elastic クラスターを選択します。

   [スナップショット識別子] フィールドに、Elastic クラスターの一意識別子を入力します。

   [作成] を選択します。



> **Note**
>
> または、[クラスター] ページに移動して [クラススナップショットの作成] ダイアログにアクセスし、対象クラスターの横にあるボックスにチェックを入れ、[アクション]、[スナップショットの取得] の順に選択します。

Elastic クラススナップショットがプロビジョニングされます。このプロセスには数分かかることがあります。[スナップショット] リストにステータスが `Available` と表示されたら、スナップショットを表示して復元できます。

Using the AWS CLI

を使用して手動 Elastic クラススナップショットを作成するには AWS CLI、以下のパラメータを指定して `create-cluster-snapshot`オペレーションを使用します。

- **`--snapshot-name`** — 必須 作成するクラスターの名前。
- **`--cluster-arn`** — 必須 スナップショットを作成するクラスターの ARN 識別子。

次の例では、各 *############* を独自の情報に置き換えます。

Linux、macOS、Unix の場合:

```
aws docdb-elastic create-cluster-snapshot \
    --snapshot-name sample-snapshot-1 \
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster
```

Windows の場合:

```
aws docdb-elastic create-cluster-snapshot ^
    --snapshot-name sample-snapshot-1 ^
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster
```

Elastic クラススナップショットの表示

このセクションでは、 AWS CLI AWS マネジメントコンソール または を使用して Elastic クラススナップショット情報を表示する方法について説明します。

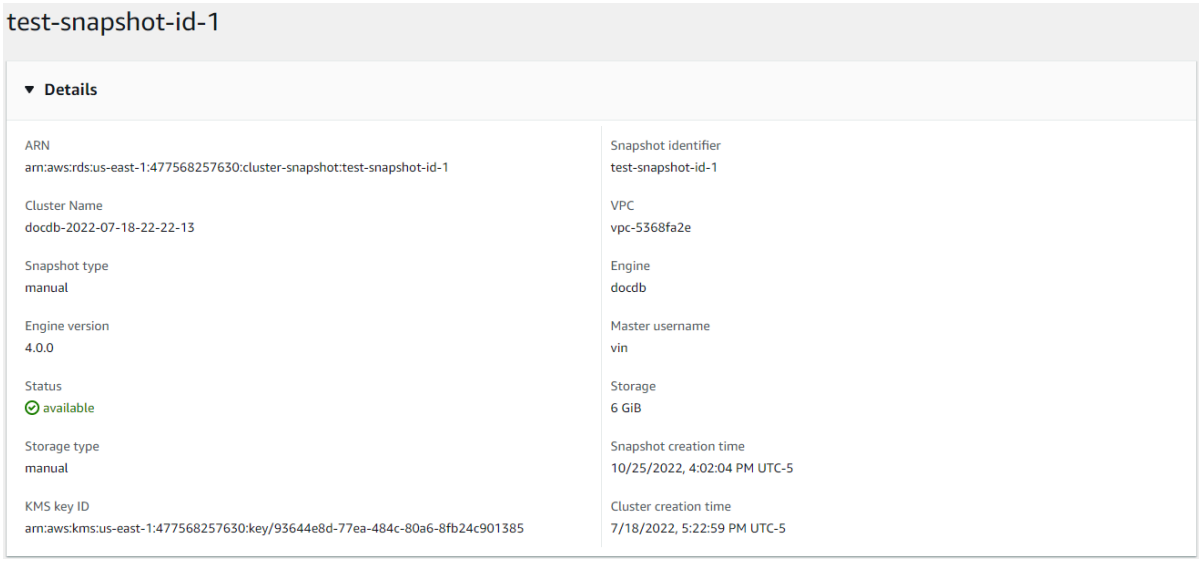
Using the AWS マネジメントコンソール

を使用して特定の Elastic クラススナップショットに関する情報を表示するには AWS マネジメントコンソール:

1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[Snapshots] を選択してください。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ナビゲーションペインの左上隅にあるメニューアイコンを選択します。

3. [スナップショット] ページで、[スナップショット識別子] 列の名前をクリックして、リストから目的のスナップショットを選択します。
4. [詳細] にスナップショットの情報が表示されます。



Using the AWS CLI

を使用して特定の Elastic クラススナップショットに関する情報を表示するには AWS CLI、以下のパラメータを指定して `get-cluster-snapshot`オペレーションを使用します。

- **`--snapshot-arn`** — 必須 情報が必要なスナップショットの ARN 識別子。

次の例では、各 ############## を独自の情報に置き換えます。

Linux、macOS、Unix の場合:

```
aws docdb-elastic get-cluster-snapshot \
    --snapshot-arn sampleResourceName
```

Windows の場合:

```
aws docdb-elastic get-cluster-snapshot ^
    --snapshot-arn sampleResourceName
```

を使用して特定の Elastic クラスタースナップショットに関する情報を表示するには AWS CLI、以下のパラメータを指定して get-cluster-snapshotオペレーションを使用します。

- **--snapshot-arn** — 必須 情報が必要なスナップショットの ARN 識別子。

次の例では、各 ############## を独自の情報に置き換えます。

Linux、macOS、Unix の場合:

```
aws docdb-elastic get-cluster-snapshot \
    --snapshot-arn sampleResourceName
```

Windows の場合:

```
aws docdb-elastic get-cluster-snapshot ^
    --snapshot-arn sampleResourceName
```

を使用してすべての Elastic クラスタースナップショットに関する情報を表示するには AWS CLI、以下のパラメータを指定して list-cluster-snapshotsオペレーションを使用します。

- **--snapshot-type** - オプションです。返されるクラスタースナップショットのタイプ。次のいずれかの値を指定できます。
  - automated - Amazon DocumentDB が AWS アカウントに対して自動的に作成したすべてのクラスタースナップショットを返します。

  - `manual` - AWS アカウント用に手動で作成したすべてのクラスタースナップショットを返します。
  - `shared` - AWS アカウントに共有されているすべての手動クラスタースナップショットを返します。
  - `public` - 公開とマークされたクラスタースナップショットをすべて返します。
- **`--next-token`** - オプションです。以前のリクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスにはこのマーカーを超え、 `max-results` で指定された値までのレコードのみが含まれます。
- **`--max-results`** - オプションです。レスポンスに含める結果の最大数。指定された `max-results` の値よりも多くの結果が存在する場合、ページ割りトークン (`next-token`) がレスポンスに含まれるため、残りの結果を取得できます。
  - デフォルト: 100
  - 最小は 20、最大は 100 です

次の例では、各 *############* を独自の情報に置き換えます。

Linux、macOS、Unix の場合:

```
aws docdb-elastic list-cluster-snapshots \
    --snapshot-type  value \
    --next-token  value \
    --max-results 50
```

Windows の場合:

```
aws docdb-elastic list-cluster-snapshots ^
    --snapshot-type  value ^
    --next-token  value ^
    --max-results 50
```

スナップショットからの Elastic クラスターの復元

このセクションでは、 AWS マネジメントコンソール または を使用して、スナップショットから Elastic クラスターを復元する方法を次の手順 AWS CLI で説明します。

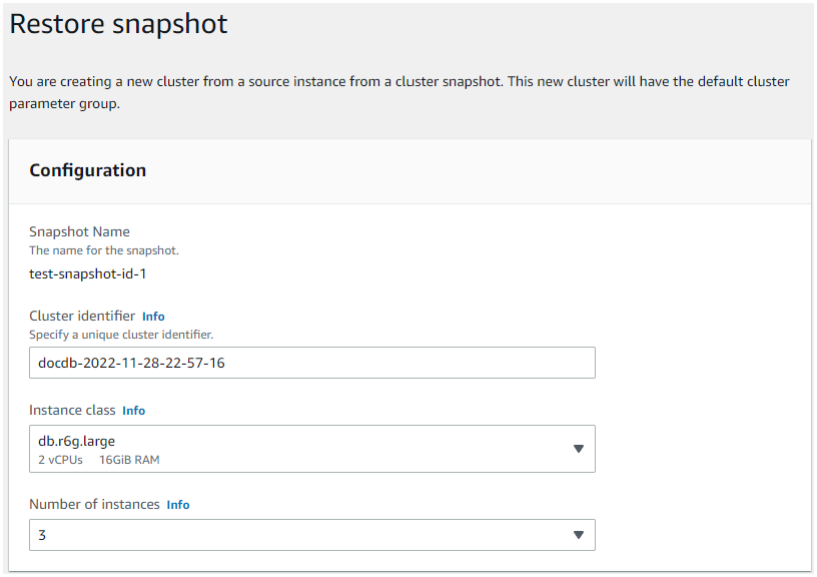
Using the AWS マネジメントコンソール

AWS マネジメントコンソールを試用してスナップショットから Elastic クラスターから復元するには:

1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[Snapshots] を選択してください。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ナビゲーションペインの左上隅にあるメニューアイコンを選択します。

3. [スナップショット識別子] 列で、クラスターの復元に使用するスナップショットの左側にあるボタンを選択します。
4. [アクション]、[復元] の順に選択します。



5. [スナップショットの復元] ページで、[クラスター識別子] フィールドに新しいクラスターの名前を入力します。

> **Note**
>
> 手動スナップショットの復元では、新しいクラスターを作成する必要があります。

6. [仮想プライベートクラウド (VPC)] フィールドのドロップダウンリストから VPC を選択します。
7. [サブネット]と [VPC セキュリティグループ]では、デフォルトを使用するか、任意の 3 つのサブネットと最大 3 つまでの VPC セキュリティグループ (最低 1 つ) を選択できます。
8. クラスター設定に問題がなければ、[Restore cluster (クラスターの復元)] を選択し、クラスターが復元されるのを待ちます。

Using the AWS CLI

を使用してスナップショットから Elastic クラスターを復元するには AWS CLI、以下のパラメータを指定して `restore-cluster-from-snapshot`オペレーションを使用します。

- **`--cluster-name`** — 必須 作成時または最終変更時に入力された Elastic クラスターの現在名。
- **`--snapshot-arn`** — 必須 クラスターの復元に使用されているスナップショットの ARN 識別子。
- **`--vpc-security-group-ids`** - オプションです。1 つまたは複数の Amazon EC2 と Amazon 仮想プライベートクラウド (VPC) セキュリティグループをクラスターに関連付けます。
- **`--kms-key-id`** - オプションです。暗号化されたクラスターの KMS キー識別子を設定します。

  KMS キー識別子は、 AWS KMS 暗号化キーの Amazon リソースネーム (ARN) です。新しいクラスターの暗号化に使用する KMS 暗号化キーを所有する Amazon Web Services アカウントと同じアカウントを使用してクラスターを作成する場合、KMS 暗号化キーの ARN の代わりに KMS キーエイリアスを使用できます。

  KmsKeyId で暗号化キーが指定がなく、かつ `StorageEncrypted` パラメータが true の場合、Amazon DocumentDB はデフォルトの暗号化キーを使用します。
- **`--subnet-ids`** - オプションです。ネットワークサブネット ID。

次の例では、各 *############* を独自の情報に置き換えます。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

Elastic クラスタースナップショットのコピー

Amazon DocumentDB では、同じリージョン内および同じアカウント内で、手動および自動の Elastic クラスタースナップショットをコピーできます。このセクションでは、 AWS マネジメントコンソール または を使用して Elastic クラスタースナップショットをコピーする方法について説明します AWS CLI。

Using the AWS マネジメントコンソール

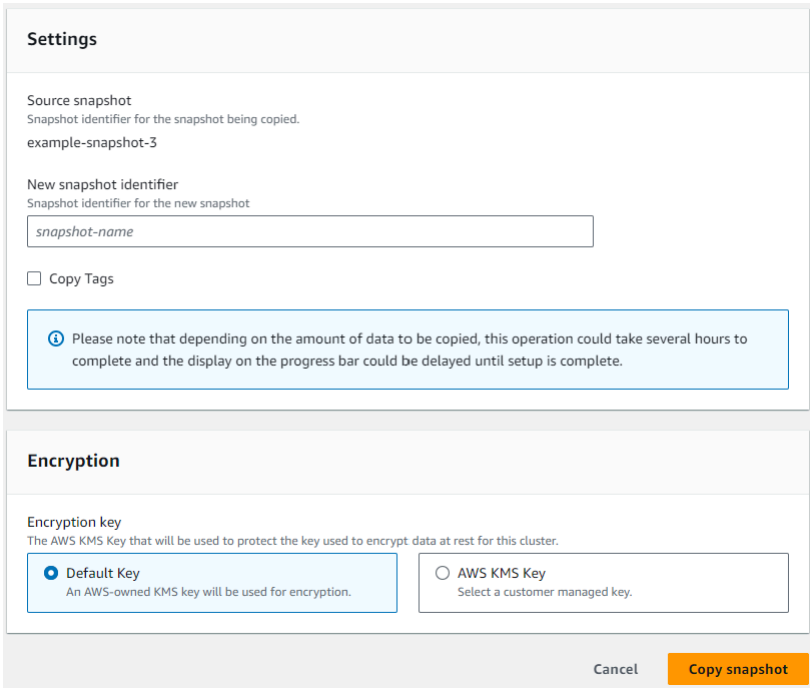
を使用して Elastic クラスタースナップショットをコピーするには AWS マネジメントコンソール:

1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[Snapshots] を選択してください。

   > **Tip**
   >
   > 画面の左側にナビゲーションペインが表示されない場合は、ナビゲーションペインの左上隅にあるメニューアイコンを選択します。

3. [スナップショット識別子] 列で、コピーしたいスナップショットの左側にあるボタンを選択します。

4. [アクション]、[コピー] の順に選択します。



5. [新しいスナップショット識別子] で、新しいスナップショットの名前を入力します。
6. コピー元の Elastic クラススナップショットからコピー先の Elastic クラススナップショットにすべてのタグをコピーしたい場合は、[タグのコピー] のチェックボックスをオンにします。
7. 暗号化では、デフォルトの KMS AWS キーまたは任意の KMS キーを選択します。任意の KMS キーを選択することで、過去に作成した既存の KMS キーを選択したり、あるいは新しい KMS キーを作成することができます。
8. 完了したら、[スナップショットのコピー] を選択します。

Using the AWS CLI

を使用して Elastic クラススナップショットをコピーするには AWS CLI、以下のパラメータを指定して `copy-cluster-snapshot`オペレーションを使用します。

- **`--source-db-cluster-snapshot-identifier`** — 必須 コピーしたい 既存の Elastic クラススナップショットの識別子。その Elastic クラスターのスナップショットが実際に存在し、利用可能な状態である必要があります。このパラメータは大文字と小文字が区別されません。

- **`--target-db-cluster-snapshot-identifier`** — 必須 既存のクラススナップショットから作成する新しいクラススナップショット向けの識別子。このパラメータは大文字と小文字が区別されません。

  コピー先のスナップショットの名前付けに関する制約事項:

  - 既存のスナップショットの名前にすることはできません。
  - 長さは、[1 ~ 63] 個の英字、数字、またはハイフンです。
  - 1 字目は英字である必要があります。
  - ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。

次の例では、各 ############## を独自の情報に置き換えます。

Linux、macOS、Unix の場合:

```
aws docdb-elastic copy-cluster-snapshot \
     --source-cluster-snapshot-arn <sample ARN> \
     --target-cluster-snapshot-name my-target-copied-snapshot
```

Windows の場合:

```
aws docdb-elastic copy-cluster-snapshot ^
     --source-cluster-snapshot-arn <sample ARN> ^
     --target-cluster-snapshot-name my-target-copied-snapshot
```

Elastic クラススナップショットの削除

このセクションでは、 AWS マネジメントコンソール または を使用して Elastic クラススナップショットを削除する方法について説明します AWS CLI。

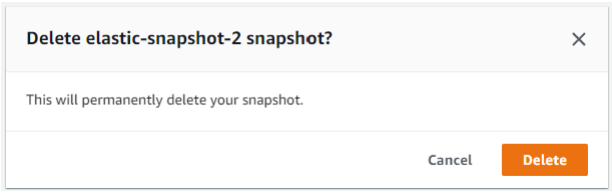
Using the AWS マネジメントコンソール

AWS マネジメントコンソールを試用してスナップショットから Elastic クラスターから復元するには:

1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで、[Snapshots] を選択してください。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ナビゲーションペインの左上隅にあるメニューアイコンを選択します。

3. [スナップショット識別子] 列で、クラスターの復元に使用するスナップショットの左側にあるボタンを選択します。
4. [Actions (アクション)] を選択してから [Delete (削除)] を選択します。



5. [「snapshot-name」スナップショットの削除] ダイアログで、[削除] を選択します。

Using the AWS CLI

を使用して Elastic クラスタースナップショットを削除するには AWS CLI、以下のパラメータを指定して `delete-cluster-snapshot`オペレーションを使用します。

- **`--snapshot-arn`** — 必須 クラスターの復元に使用されているスナップショットの ARN 識別子。

次の例では、各 *############* を独自の情報に置き換えます。

Linux、macOS、Unix の場合:

```
aws docdb-elastic delete-cluster-snapshot \
     --snapshot-arn sampleResourceName
```

Windows の場合:

```
aws docdb-elastic delete-cluster-snapshot ^
     --snapshot-arn sampleResourceName
```

Elastic クラススナップショットの自動バックアップの管理

Amazon DocumentDB は、お使いの各 Elastic クラスターのスナップショットを毎日作成します。新規または既存の Elastic クラススナップショット設定で、希望するバックアップウィンドウ (タイミング) とバックアップ保持期間を指定できます。このセクションでは、 AWS マネジメントコンソール または を使用して、Elastic クラススナップショットに自動バックアップパラメータを設定する方法について説明します AWS CLI。

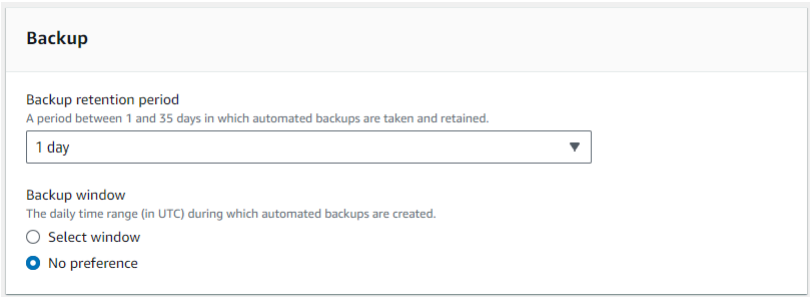
Using the AWS マネジメントコンソール

を使用して新しい Elastic クラススナップショットの自動バックアップを設定するには AWS マネジメントコンソール:

1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ナビゲーションペインの左上隅にあるメニューアイコンを選択します。

3. [クラスター識別子] 列で、バックアップ設定を変更したいクラスターの左側にあるボタンを選択します。
4. [アクション]、[変更] の順にクリックします。
5. [バックアップ] セクションで、該当するバックアップ要件に従ってフィールドを編集します。



a. [バックアップ保持期間] - リストで、このクラスターの自動バックアップを削除するまで保持する日数を選択します。

b. [バックアップウィンドウ] - Amazon DocumentDB によってこのクラスターのバックアップが作成される毎日の時刻と期間を設定します。

i. バックアップの作成タイミングと期間を設定したい場合は、[ウィンドウを選択] を選択します。

[開始時刻] - 最初のリストで、自動バックアップを開始する時間 (UTC 時) を選択します。2 番目のリストで、自動バックアップを開始する時間 (UTC 分) を選択します。

[期間] - リストで、自動バックアップの作成に割り当てる時間を選択します。

ii. Amazon DocumentDB 動作に従ってバックアップの作成タイミングや期間を自動選択する場合は、[設定なし] を選択します。

6. 完了したら [クラスターの変更] を選択します。

Using the AWS CLI

を使用して新しい Elastic クラススナップショットの自動バックアップを設定するには AWS CLI、次のパラメータを指定して `create-cluster-snapshot`オペレーションを使用します。

- **`--preferred-backup-window`** - オプションです。毎日自動バックアップを作成する希望の時間帯。デフォルトは、それぞれ 8 時間の時間ブロックからランダムに選択された 30 分のウィンドウです AWS リージョン。

  制約:

  - `hh24:mi-hh24:mi` の形式である必要があります。
  - 時間は協定世界時 (UTC) である必要があります。
  - 必要なメンテナンス期間と競合してはいけません。
  - 少なくとも 30 分以上必要です。
- **`--backup-retention-period`** - オプション。自動バックアップを保管する日数。デフォルト値は 1 です。

  制約:

  - 最小でも 1 を指定する必要があります。
  - 1 から 35 の範囲の数を指定できます。

> **Note**
>
> 自動バックアップは、そのクラスターが「アクティブ」状態である場合にのみ実行されます。

> **Note**
>
> `aws docdb-elastic update-cluster` コマンドを使用して、既存の Elastic クラスターの `preferred-backup-window` および `backup-retention-period` パラメータを変更することもできます。

次の例では、各 *############* を独自の情報に置き換えます。

次の `create-cluster` 例では、自動バックアップの保持期間が *7* 日間で、優先バックアップウィンドウが *18:00#18:30 UTC* である Amazon DocumentDB Elastic クラスターの*#######*を作成します。

Linux、macOS、Unix の場合:

```
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
```

Windows の場合:

```
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
```

Amazon DocumentDB Elastic クラスターの停止と開始

Amazon DocumentDB Elastic クラスターの停止および開始機能は、開発環境やテスト環境のコストを管理するのに便利です。Amazon DocumentDB を使用するたびに Elastic クラスターを作成したり削除する代わりに、必要ではないときにクラスターを一時的に停止させることができます。必要になった時にクラスターを再開すれば、テストを再開することができます。

トピック

- Elastic クラスターの停止と開始の概要
- 停止中の Elastic クラスターのみで実行できるオペレーション。

Elastic クラスターの停止と開始の概要

Amazon DocumentDB Elastic クラスターが必要ではない期間は、そのクラスターを停止することができます。そして、クラスターを使用する必要がある時はいつでもクラスターを開始できます。開始と停止は、継続的な可用性を必要としない開発、テスト、または類似のアクティビティに使用される Elastic クラスターのセットアップと解放のプロセスを簡素化します。Elastic クラスターを停止および開始するには、 AWS マネジメントコンソール または を 1 つのアクション AWS CLI で使用します。

Elastic クラスターの停止中は、そのクラスターのストレージボリュームは変更されません。指定された保持期間内でのストレージ、手動のスナップショット、および自動化されたバックアップストレージに対してのみ課金されます。Amazon DocumentDB は、7 日後に Elastic クラスターを自動的に再開するため、必要なメンテナンスの更新が遅延することはありません。7 日後にクラスターが再開すると、その Elastic クラスター内のインスタンスに対する課金が再開します。クラスターの停止中は、そのストレージボリュームに対するクエリを実行できません。クエリを実行するにはクラスターが使用可能状態であることが必要であるからです。

Amazon DocumentDB Elastic クラスターの停止中は、そのクラスターには一切変更を加えることはできません。これには、クラスターの削除も含まれます。

Using the AWS マネジメントコンソール

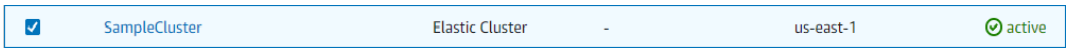
次の手順では、利用可能な状態にある Elastic クラスターの停止方法と、停止状態にある Elastic クラスターの再開方法を示します。

Amazon DocumentDB Elastic クラスターを停止または開始するには

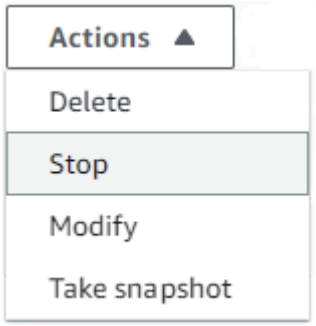
1. にサインインし AWS マネジメントコンソール、https://console.aws.amazon.com/docdb で Amazon DocumentDB コンソールを開きます。
2. ナビゲーションペインで クラスター を選択します。

> **Tip**
>
> 画面の左側にナビゲーションペインが表示されない場合は、ページの左上隅にあるメニューアイコン (≡) を選択します。

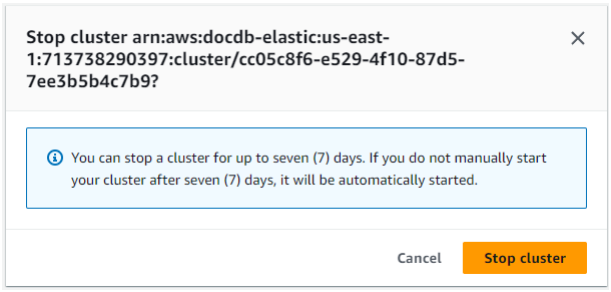
3. クラスターのリストで、停止または開始するクラスター名の左側にあるボタンを選択します。



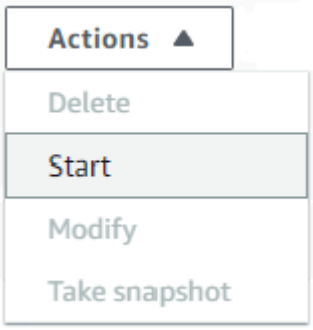
4. [Actions (アクション)] を選択したら、このクラスターで実行するアクションを選択します。
   - クラスターを停止したいときに、このクラスターが利用可能な場合:
     a. [停止] を選択します。



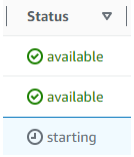
     b. 確認ダイアログで [クラスターの停止] を選択して Elastic クラスターの停止を確定するか、もしくは [キャンセル] を選択してクラスター動作を継続させます。



- クラスターを開始したいときにこのクラスターが停止されている場合、[Start (開始)] を選択します。



5. Elastic クラスターのステータスをモニタリングします。クラスターの再開を選択すると、クラスターが利用可能な状態にあればクラスターの使用を再開できます。詳細については、「クラスターのステータスの評価」を参照してください。



Using the AWS CLI

次のコード例では、アクティブもしくは利用可能な状態にある Elastic クラスターを停止する方法と、停止状態にある Elastic クラスターを再開する方法を示します。

を使用して Elastic クラスターを停止するには AWS CLI、 `stop-cluster`オペレーションを使用します。停止しているクラスターを開始するには、 `start-cluster` オペレーションを使用します。どちらのオペレーションも `--cluster-arn` パラメータを使用します。

パラメータ:

- **`--cluster-arn`** — 必須 停止もしくは再開したい Elastic クラスターの ARN 識別子。

Example— を使用して Elastic クラスターを停止するには AWS CLI

次の例では、各 `##############` を独自の情報に置き換えます。

次のコードにより、ARN が `arn:aws:docdb-elastic:us-east-1:477568257630:cluster/b9f1d489-6c3e-4764-bb42-da62ceb7bda2` である Elastic クラスターを停止します。

> **Note**
>
> これには、その Elastic クラスターがアクティブまたは利用可能な状態でなければなりません。

Linux、macOS、Unix の場合:

```
aws docdb-elastic stop-cluster \
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

Windows の場合:

```
aws docdb-elastic stop-cluster ^
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

Example— を使用して Elastic クラスターを起動するには AWS CLI

次の例では、各 ############## を独自の情報に置き換えます。

次のコードにより、ARN が arn:aws:docdb-elastic:us-east-1:477568257630:cluster/b9f1d489-6c3e-4764-bb42-da62ceb7bda2 である Elastic クラスターを再開します。

> **Note**
>
> これには、その Elastic クラスターが現在停止状態である必要があります。

Linux、macOS、Unix の場合:

```
aws docdb-elastic start-cluster \
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

Windows の場合:

```
aws docdb-elastic start-cluster ^
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

停止中の Elastic クラスターのみで実行できるオペレーション。

クラスターが停止中は、Amazon DocumentDB Elastic クラスターの設定を変更することはできません。そのような管理アクションを実行する前に、クラスターを開始する必要があります。

Amazon DocumentDB は、停止中の Elastic クラスターにスケジュールされているメンテナンスについては、クラスターの再開後に適用します。Amazon DocumentDB は停止した Elastic クラスターを 7 日後に自動的に再開するため、メンテナンス実施状況が過剰に遅延することはありません。Elastic クラスターが再開すると、クラスター内のシャードへの課金が再開します。

Elastic クラスターの停止中、Amazon DocumentDB は自動バックアップを一切実行せず、またバックアップ保持期間を延長することはありません。

Amazon DocumentDB エラスティッククラスターのメンテナンス

トピック

- 保留中のエラスティッククラスターメンテナンスアクションの表示
- エラスティッククラスターエンジンの更新
- エラスティッククラスターのオペレーティングシステムの更新

Amazon DocumentDB は定期的に、Amazon DocumentDB エラスティッククラスターリソースのメンテナンスを行います。通常、メンテナンスにはデータベースエンジンの更新 (エラスティッククラスターのメンテナンス) やエラスティッククラスターの基盤となるオペレーティングシステム (OS) の更新 (OS の更新) が伴います。データベースエンジンの更新は必須のパッチであり、セキュリティ修正、バグ修正、データベースエンジンの強化などが含まれます。一方ほとんどのオペレーティングシステムパッチはオプションですが、しばらく適用しないで放置していると、セキュリティ体制の維持のためにパッチが必要となった結果、自動的に適用されることがあります。Amazon DocumentDB エラスティッククラスターが使用可能になり次第、オペレーティングシステムの更新を適用します。

データベースエンジンパッチの適用には、Amazon DocumentDB エラスティッククラスターを短時間オフラインにする必要があります。パッチが利用可能になるたびに、Amazon DocumentDB エラスティッククラスターの次のメンテナンスウィンドウでの実施予定に組み込まれ、自動的に適用されます。

エラスティッククラスターには、それぞれ独自のメンテナンスウィンドウがあります。エラスティッククラスターの修正をすぐには適用しないことを選択した場合は、メンテナンスウィンドウ中に自動的に適用されます。デフォルトでは、エラスティッククラスターを作成すると、Amazon DocumentDB はエラスティッククラスターにメンテナンスウィンドウを割り当てます。エラスティッククラスターを作成するときにメンテナンスウィンドウを選択できます。また、ビジネススケジュールや業務に合わせてメンテナンスウィンドウをいつでも変更できます。一般的には、メンテナンスによるアプリケーションへの影響を最小限に抑えるメンテナンスウィンドウ (夜間、週末など) を選択することをお勧めします。

保留中のエラスティッククラスターメンテナンスアクションの表示

エラスティッククラスターでメンテナンス更新が利用可能かどうかを確認するには、 AWS CLIを使用します。

更新が利用可能な場合は、以下のいずれかの操作を行います。

- 現在次のメンテナンスウィンドウにスケジュールされているメンテナンスアクションを延期することができます (OS パッチのみ)。
- メンテナンスアクションをすぐに適用する。
- メンテナンスアクションを次のメンテナンスウィンドウ中に開始するようにスケジュールを設定する。
- メンテナンスアクションを選択した適用日ウィンドウ中に開始するようにスケジュールを設定する。

メンテナンスウィンドウによって、保留中のオペレーションを開始する時刻が決定されますが、これらのオペレーションの合計実行時間は制限されません。

次の AWS CLI オペレーションを使用して、保留中のメンテナンスアクションを決定します。保留中のすべてのメンテナンスアクションを一覧表示します。

```
aws docdb-elastic list-pending-maintenance-actions
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
'ResourcePendingMaintenanceActions': [
    {
        'ResourceArn': 'string-arn',
```

```
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'ENGINE_UPDATE',
                'AutoAppliedAfterDate': 'string',
                'ForcedApplyDate': 'string',
                'OptInStatus': 'string',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            },
        ]
    },
],
'NextToken': 'string'
}
```

特定の resourceArn で保留中のメンテナンスアクション (存在する場合) を取得します。

```
aws docdb-elastic get-pending-maintenance-action --resource-arn string-arn
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    'ResourcePendingMaintenanceAction': {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'ENGINE_UPDATE',
                'AutoAppliedAfterDate': 'string',
                'ForcedApplyDate': 'string',
                'OptInStatus': 'string',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            }
        ]
    }
}
```

パラメータ :

- ResourceArn - 保留中のメンテナンスアクションが適用されるリソースの Amazon DocumentDB ARN (Amazon リソースネーム) です。
- Action - メンテナンスアクションはリソースに適用中です。

有効な値:

- ENGINE_UPDATE
- ENGINE_UPGRADE
- SECURITY_UPDATE
- OS_UPDATE
- MASTER_USER_PASSWORD_UPDATE

- AutoAppliedAfterDate — この日付以降の最初のメンテナンスウィンドウ。この場合、NEXT_MAINTENANCE OPT_IN は無視されます。
- ForcedApplyDate — メンテナンスウィンドウに関係なく適用されます。この場合、IMMEDIATE OPT_IN は無視されます。
- OptInStatus — オプトインリクエストのタイプを指定するか、オプトインクエストを元に戻す値です。タイプ IMMEDIATE のオプトインリクエストは元に戻すことができません。

  有効な値:

  - IMMEDIATE — メンテナンスアクションをすぐに適用します。
  - NEXT_MAINTENANCE — リソースの次のメンテナンスウィンドウ中にメンテナンスアクションを適用します。
  - APPLY_ON — リソースの次のメンテナンスウィンドウに関係なく、指定された適用日にメンテナンスアクションを適用します。
  - UNDO_OPT_IN — 既存の NEXT_MAINTENANCE または APPLY_ON オプトインリクエストをキャンセルします。
- CurrentApplyDate — opt-in-type が APPLY_ON の場合に表示されます。
- Description — メンテナンスアクションのオプションの説明。

エラスティッククラスターエンジンの更新

Amazon DocumentDB では、メンテナンスオペレーションを適用するタイミングを選択することができます。Amazon DocumentDB がいつ更新を適用するかは、 AWS CLIを使用して決めることができます。

保留中のメンテナンスアクションを適用する:

```
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]
```

パラメータ :

- **--resource-arn** - 保留中のメンテナンスアクションが適用されるリソースの Amazon DocumentDB ARN (Amazon リソースネーム) です。
- **--apply-action** — このリソースに適用する保留中のメンテナンスアクションです。

  有効な値:

  - ENGINE_UPDATE
  - ENGINE_UPGRADE
  - SECURITY_UPDATE
  - OS_UPDATE
  - MASTER_USER_PASSWORD_UPDATE
- **--opt-in-type** — オプトインリクエストのタイプを指定するか、オプトインクエストを元に戻す値です。タイプ IMMEDIATE のオプトインリクエストは元に戻すことができません。

  有効な値:

  - IMMEDIATE — メンテナンスアクションをすぐに適用します。
  - NEXT_MAINTENANCE — リソースの次のメンテナンスウィンドウ中にメンテナンスアクションを適用します。
  - APPLY_ON — リソースの次のメンテナンスウィンドウに関係なく、指定された適用日にメンテナンスアクションを適用します。
  - UNDO_OPT_IN — 既存の NEXT_MAINTENANCE または APPLY_ON オプトインリクエストをキャンセルします。
- **[--apply-on]** — opt-in-type が APPLY_ON の場合は必須です。形式: yyyy/MM/dd HH:mm-yyyy/MM/dd HH:mm (このオプションでは UTC 時間を使用します。 開始時刻は、今後の最小 30 分から最大 14 日の任意の時刻、または保留中のアクションの強制/適用日で、いずれか早い方にすることができます。 開始から終了までの時間は、最小で 30 分、最大で 8 時間にすることができます)。

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
 'ResourcePendingMaintenanceAction': {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'SECURITY_UPDATE',
                'AutoAppliedAfterDate': 'string',
                'ForcedApplyDate': 'string',
                'OptInStatus': 'IMMEDIATE',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            },
        ]
 }
}
```

パラメータ :

- ResourceArn - 保留中のメンテナンスアクションが適用されるリソースの Amazon DocumentDB ARN (Amazon リソースネーム) です。
- Action - メンテナンスアクションはリソースに適用中です。

  有効な値:

  - ENGINE_UPDATE
  - ENGINE_UPGRADE
  - SECURITY_UPDATE
  - OS_UPDATE
  - MASTER_USER_PASSWORD_UPDATE
- AutoAppliedAfterDate — この日付以降の最初のメンテナンスウィンドウ。この場合、NEXT_MAINTENANCE OPT_IN は無視されます。
- ForcedApplyDate — メンテナンスウィンドウに関係なく適用されます。この場合、IMMEDIATE OPT_IN は無視されます。
- OptInStatus — オプトインリクエストのタイプを指定するか、オプトインクエストを元に戻す値です。タイプ IMMEDIATE のオプトインリクエストは元に戻すことができません。

  有効な値:

  - IMMEDIATE — メンテナンスアクションをすぐに適用します。

  - NEXT_MAINTENANCE — リソースの次のメンテナンスウィンドウ中にメンテナンスアクションを適用します。
  - APPLY_ON — リソースの次のメンテナンスウィンドウに関係なく、指定された適用日にメンテナンスアクションを適用します。
  - UNDO_OPT_IN — 既存の NEXT_MAINTENANCE または APPLY_ON オプトインリクエストをキャンセルします。
- CurrentApplyDate — opt-in-type が APPLY_ON の場合に表示されます。
- Description — メンテナンスアクションのオプションの説明。

適用日

各メンテナンスアクションにはそれぞれ、保留中のメンテナンスアクションを記述する際に確認できる適用日が設定されています。から保留中のメンテナンスアクションの出力を読み取ると AWS CLI、次の 3 つの日付が表示されます。

- CurrentApplyDate - メンテナンスアクションが即座に、または次のメンテナンスウィンドウに適用される日付です。メンテナンスがオプションである場合は、この値を null に設定できます。
- ForcedApplyDate - メンテナンスウィンドウとは関係なく、メンテナンスが自動的に適用される日付です。
- AutoAppliedAfterDate - クラスターのメンテナンスウィンドウでメンテナンスが適用される日付です。

ユーザー作成のメンテナンスアクション

Amazon DocumentDBelastic クラスターのユーザーとして、クラスター設定への更新を開始することができます。

クラスターのプライマリパスワードの更新

```
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
```

パラメータ :

- **--cluster-arn** — メンテナンスアクションが適用されるリソースの Amazon DocumentDB Amazon リソースネーム (ARN)。
- **[--admin-user-password]** — 管理者ユーザーに関連付けられているパスワード。
- **[--auth-type]** — エラスティッククラスターへのアクセスに使用するパスワードの取得先の特定に使用している認証タイプ。有効なタイプは、PLAIN_TEXT または SECRET_ARN です。
- **[--apply-method]** — 適用するメソッドのタイプを指定する値。指定できる値は IMMEDIATE と APPLY_ON です。デフォルトは IMMEDIATE です。
- **[--apply-on]** — apply-method が APPLY_ON の場合は必須です。形式: yyyy/MM/dd HH:mm-yyyy/MM/dd HH:mm (このオプションでは UTC 時間を使用します。 開始時刻は、今後の最小 30 分から最大 14 日の任意の時刻にすることができます。 開始から終了までの時間は、最小で 30 分、最大で 8 時間にすることができます)。

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
 'ResourcePendingMaintenanceAction': {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'MASTER_USER_PASSWORD_UPDATE',
                'OptInStatus': 'APPLY_ON',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            },
        ]
 }
}
```

Amazon DocumentDB メンテナンスウィンドウを変更する

メンテナンスウィンドウは使用率の最も低い時間帯に設定する必要があります。このため、場合によっては変更が必要になります。この間、エラスティッククラスターは、システムの変更 (スケールストレージオペレーションの変更など) が適用されていて停止が必要な場合に限り、使用できません。必要な変更を加えるのに必要な最小限の時間のみ使用不可となります。

デフォルトは、1 週間のうちのランダムな日に起こる、Amazon Web Services リージョンあたり 8 時間の範囲からランダムに選択された 30 分のウィンドウです。

メンテナンスウィンドウを変更するには、「Elastic クラスター設定の変更」を参照してください。

エラスティッククラスターのオペレーティングシステムの更新

Amazon DocumentDB エラスティッククラスターでは、オペレーティングシステムの更新が必要になる時があります。Amazon DocumentDB は、データベースパフォーマンスと顧客の全体的なセキュリティ体制改善のために、オペレーティングシステムを新しいバージョンにアップグレードします。オペレーティングシステムの更新では、Amazon DocumentDB イエラスティッククラスターのクラスターエンジンのバージョンは変更されません。

Amazon DocumentDB エラスティッククラスターのほとんどのオペレーティングシステムの更新はオプションであり、特定の適用日付は設定されていません。ただし、これらの更新をしばらく適用せずに放置していると、クラスターのメンテナンスウィンドウ時に最終的に必要になり、想定外のタイミングで自動的に適用される場合があります。これは、データベースのセキュリティ体制を維持するために行われます。予期しないダウンタイムを回避するには、利用可能になったらすぐに Amazon DocumentDB エラスティッククラスターにオペレーティングシステムの更新を適用し、ビジネスニーズに合わせてクラスターのメンテナンスウィンドウを設定します。

Amazon DocumentDB エラスティッククラスターで保管中のデータ暗号化

以下のトピックは、Amazon DocumentDB エラスティッククラスターの AWS Key Management Service 暗号化キーについて学習、作成、モニタリングするのに役立ちます。

トピック

- Amazon DocumentDB エラスティッククラスターが で許可を使用する方法 AWS KMS
- カスタマーマネージドキーを作成する
- Amazon DocumentDB エラスティック クラスター の暗号化キーのモニタリング
- 詳細情報

Amazon DocumentDB エラスティッククラスターは、キー管理のために AWS Key Management Service (AWS KMS) と自動的に統合され、エンベロープ暗号化と呼ばれる方法を使用してデータを保護します。エンベロープ暗号化の仕組みの詳細については、AWS Key Management Service デベロッパーガイドの「エンベロープ暗号化」を参照してください。

AWS KMS key はキーの論理表現です。KMS キーには、キー ID、作成日、説明、キーステータスなどのメタデータが含まれます。KMS キーには、データの暗号化と復号に使用されるキーマテリアル

も含まれています。KMS キーの詳細については、AWS Key Management Service デベロッパーガイドの「AWS KMS keys 」を参照してください。

Amazon DocumentDB エラスティッククラスターは、次の 2 種類のキーによる暗号化をサポートしています。

- AWS 所有キー — Amazon DocumentDB エラスティッククラスターは、デフォルトでこれらのキーを使用して、個人を特定できるデータを自動的に暗号化します。 AWS所有キーを表示、管理、使用したり、それらの使用を監査したりすることはできません。ただし、データを暗号化するキーを保護するためのアクションの実施やプログラムの変更を行う必要はありません。詳細については、AWS Key Management Service デベロッパーガイドの「AWS 所有キー」を参照してください。
- カスタマーマネージドキー — AWS KMS keys ユーザーが作成、所有、管理する対称キー。この暗号化層はユーザーが完全に制御できるため、次のようなタスクを実行できます。
  - キーポリシーの策定と維持
  - IAM ポリシーとグラントの策定と維持
  - キーポリシーの有効化と無効化
  - キー暗号化マテリアルのローテーション
  - タグを追加する
  - キーエイリアスの作成
  - 削除のためのキースケジューリング

  詳細については、「AWS Key Management Service デベロッパーガイド」の「カスタマーマネージドキー」を参照してください。

> **Important**
>
> Amazon DocumentDB がサポートしているのは対称 KMS キーのみであるため、クラスターの暗号化には対称暗号化 KMS キーを使用して必要があります。Amazon DocumentDB エラスティッククラスター内のデータの暗号化に非対称 KMS キー を使用しないでください。詳細については、AWS Key Management Service デベロッパーガイドの「AWS KMSの非対称キーの」を参照してください。
>
> Amazon DocumentDB がクラスターの暗号化キーにアクセスできなくなった場合、例えばキーへのアクセス権が失効した場合など、暗号化されたクラスターは終了状態になります。この場合、クラスターはバックアップからのみ復元できます。Amazon DocumentDB では、バックアップは常に 1 日間有効になります。また、暗号化された Amazon DocumentDB ク

ラスターのキーを無効にすると、最終的にそのクラスタに対する読み取りおよび書き込みのアクセス権は失われます。Amazon DocumentDB で、アクセスできないキーで暗号化されたクラスターが発生すると、クラスターが終了状態になります。この場合、クラスターは使用できなくなり、データベースの現在の状態を復元することはできません。クラスターを復元するには、Amazon DocumentDB の暗号化キーへのアクセスを再び有効にした後、バックアップからクラスターを復元します。

> **Important**
>
> 暗号化されたクラスターの KMS キーを作成した後は、そのクラスターを変更することはできません。暗号化されたエラスティッククラスターを作成する前に、暗号化キーの要件を確認してください。

Amazon DocumentDB エラスティッククラスターが で許可を使用する方法 AWS KMS

Amazon DocumentDB エラスティック クラスター には、カスタマーマネージドキーを使用するためのグラントが必要です。

カスタマーマネージドキーで暗号化されたクラスターを作成すると、Amazon DocumentDB エラスティッククラスターはCreateGrantリクエストを送信してユーザーに代わって許可を作成しますAWS KMS。の権限 AWS KMS は、Amazon DocumentDB エラスティッククラスターに顧客アカウントの KMS キーへのアクセスを許可するために使用されます。

Amazon DocumentDB エラスティッククラスターでは、以下の内部操作にカスタマーマネージドキーを使用するためのグラントが必要です。

- DescribeKey Elastic クラスターの作成時に入力された対称カスタマーマネージドキー ID が有効であることを確認する AWS KMS リクエストを に送信します。
- カスタマーマネージドキーによって暗号化されたデータキーを生成する AWS KMS リクエストGenerateDataKeyを に送信します。
- データを暗号化 AWS KMS するために使用できるように、暗号化されたデータキーを復号化するためのDecryptリクエストを に送信します。
- グラントへのアクセスの取り消しや、カスタマーマネージドキーに対するサービスのアクセスの取り消しは、いつでもできます。これを行うと、Amazon DocumentDB エラスティッククラスター

はカスタマーマネージドキーによって暗号化されたすべてのデータにアクセスできなくなり、そのデータに依存しているオペレーションが影響を受けます。

カスタマーマネージドキーを作成する

対称カスタマーマネージドキーは、 AWS マネジメントコンソール または AWS KMS API を使用して作成できます。

対称カスタマー管理キーの作成

AWS Key Management Service デベロッパーガイドにある対称カスタマーマネージドキーの作成ステップを実行します。

キーポリシー

キーポリシーは、カスタマーマネージドキーへのアクセスを制御します。すべてのカスタマーマネージドキーには、キーポリシーが 1 つだけ必要です。このポリシーには、そのキーを使用できるユーザーとその使用方法を決定するステートメントが含まれています。キーポリシーは、カスタマーマネージドキーの作成時に指定できます。詳細については、AWS Key Management Service デベロッパーガイドの「AWS Key Management Service 概要」にあるKMS キーアクセス情報を参照してください。

Amazon DocumentDB エラスティック クラスター で顧客管理キーを使用するには、キーポリシーで次の API オペレーションを許可する必要があります。

- `kms:CreateGrant` - カスタマーマネージドキーに許可を追加します。指定された KMS キーへのアクセスを制御する権限を付与します。これにより、Amazon DocumentDB エラスティッククラスターに必要な権限オペレーションへのアクセスを許可します。許可の詳細については、AWS Key Management Service デベロッパーガイドの「許可の使用 AWS KMS」 を参照してください。
- `kms:DescribeKey` – Amazon DocumentDB エラスティッククラスターがキーを検証できるように、カスタマーマネージドキーの詳細を提供します。
- `kms:Decrypt` – Amazon DocumentDB エラスティッククラスターが、保存された暗号化されたデータキーを使用して暗号化されたデータにアクセスできるようにします。
- `kms:GenerateDataKey` – Amazon DocumentDB エラスティッククラスターが暗号化されたデータキーを生成して保存できるようにします。これは、データキーがすぐに暗号化に使用されないためです。

詳細については、「 デベロッパーガイド」の「キーポリシーの AWS サービスのアクセス許可」および「キーアクセスのトラブルシューティング」を参照してください。 AWS Key Management Service

IAM ポリシーによるカスタマーマネージドキーアクセスの制限

KMS キーポリシーに加えて、IAM ポリシーで KMS キー権限を制限することもできます。

さまざまな方法で、IAM ポリシー をより厳しくすることができます。例えば、Amazon DocumentDB エラスティッククラスターを起点とするリクエストにのみカスタマーマネージドキーを使用できるようにするには、`docdb-elastic.<region-name>.amazonaws.com` 値で `kms:ViaService` 条件キーを使用します。

詳細については、AWS Key Management Service デベロッパーガイドの「他のアカウントのユーザーに KMS キーの使用を許可する」を参照してください。

Amazon DocumentDB エラスティック クラスター の暗号化キーのモニタリング

Amazon DocumentDB エラスティッククラスターリソースでカスタマーマネージドキーを使用する場合、 AWS CloudTrail または Amazon CloudWatch Logs を使用してAmazon DocumentDB エラスティッククラスターが送信するリクエストを追跡できます AWS KMS。

次の例は`CreateGrant`、Amazon DocumentDB エラスティッククラスターによって呼び出される AWS KMS key オペレーションをモニタリングして、カスタマーマネージドキーによって暗号化されたデータにアクセス`DescribeKey`するための `Decrypt`、、、および `GenerateDataKeyWithoutPlainText`の AWS CloudTrail イベントです。

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"
}
```

詳細情報

次のリソースは、保管時のデータ暗号化についての詳細を説明しています。

- AWS KMS 概念の詳細については、「 AWS Key Management Service デベロッパーガイド」のAWS Key Management Service 「基本概念」を参照してください。
- AWS KMS セキュリティの詳細については、「 AWS Key Management Service デベロッパーガイド」の「 のセキュリティのベストプラクティス AWS Key Management Service」を参照してください。

エラスティッククラスター上のサービスにリンクされたロール

Amazon DocumentDB エラスティッククラスターは AWS Identity and Access Management 、(IAM) サービスにリンクされたロールを使用します。サービスにリンクされているロールは、Amazon DocumentDB エラスティッククラスターに直接リンクされれている特殊なタイプの IAM ロールです。サービスにリンクされたロールは Amazon DocumentDB エラスティッククラスターによって事前定義されており、サービスがユーザーに代わって他の AWS サービスを呼び出すために必要なすべてのアクセス許可が含まれています。

サービスにリンクされているロールを使用すると、必要な権限を手動で追加する必要がないため、Amazon DocumentDB の使用が簡単になります。Amazon DocumentDB エラスティッククラスターには、サービスにリンクされているロールのアクセス許可が定義されます。特に定義されていない限り、Amazon DocumentDB のみがそのロールを引き受けることができます。定義されたアクセス許可には、信頼ポリシーとアクセス許可ポリシーが含まれ、そのアクセス許可ポリシーを他の IAM エンティティにアタッチすることはできません。ロールを削除するには、まず関連リソースを削除します。これにより、リソースにアクセスするためのアクセス許可を誤って削除することができないため、Amazon DocumentDB エラスティッククラスターが保護されます。

サービスにリンクされているロールをサポートするその他のサービスについては、「IAM と連携するAWS のサービス」を参照し、[サービスにリンクされたロール] 列が [はい] になっているサービスを探してください。サービスリンクロールに関するドキュメントをサービスで表示するには、リンクで [はい] を選択します。

エラスティッククラスターに対するサービスにリンクされたロールのアクセス許可

Amazon DocumentDB エラスティッククラスターは、 という名前のサービスにリンクされたロールを使用してAWS ServiceRoleForDocDB-Elastic、Amazon DocumentDB エラスティッククラスターがクラスターに代わって AWS サービスを呼び出すことを許可します。

このサービスにリンクされたロールには、アカウントで操作するためのアクセス許可を付与する AmazonDocDB-ElasticServiceRolePolicy というアクセス許可ポリシーがアタッチされています。ロールのアクセス許可ポリシーは、Amazon DocumentDB エラスティッククラスターが指定リソースに対して以下のアクションを実行することを 許可します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": [
                        "AWS/DocDB-Elastic"
                    ]
                }
            }
        }
    ]
}
```

> **Note**
>
> サービスリンクロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するにはアクセス許可を設定する必要があります。次のエラーメッセージ、「リソースを作成できません。サービスにリンクされたロールを作成するために必要なアクセス許可があることを確認します。それ以外の場合は、時間をおいてからもう一度お試しください。」が表示された場合は、以下のアクセス許可が有効であることを確認します。

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

詳細についてはIAM ユーザーガイド の「サービスにリンクされた役割のアクセス許可」を参照してください。

Amazon DocumentDB エラスティッククラスター用のサービスにリンクされたロールの作成

サービスリンクロールを手動で作成する必要はありません。DB インスタンスが作成されると、Amazon DocumentDB の Elastic クラスターはサービスにリンクされたロールを自動的に作成します。

Amazon DocumentDB エラスティッククラスター用のサービスにリンクされたロールの編集

Amazon DocumentDB のエラスティッククラスターでは、AWS ServiceRoleForDocDB-Elastic サービスにリンクされたロールを編集することはできません。サービスリンクロールの作成後は、さまざまなエンティティがロールを参照する可能性があるため、ロール名を変更することはできま

せん。ただし、IAM を使用してロールの説明を編集することはできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

Amazon DocumentDB エラスティッククラスター用のサービスにリンクされたロールの削除

サービスにリンクされたロールを必要とする機能またはサービスを使用する必要がなくなった場合は、そのロールを削除します。そうすることで、使用していないエンティティがアクティブにモニタリングされたり、メンテナンスされたりすることがなくなります。ただし、サービスにリンクされたロールを削除する前に、すべての クラスターを削除する必要があります。

サービスにリンクされたロールのクリーンアップ

IAM を使用してサービスにリンクされたロールを削除するには、まずそのロールにアクティブなセッションがないことを確認し、そのロールで使用されているリソースをすべて削除する必要があります。

サービスにリンクされたロールにアクティブなセッションがあるかどうかを、IAM コンソールで確認するには:

1. AWS マネジメントコンソール にサインインし、IAM コンソール を開きます。
2. IAM コンソールのナビゲーションペインで [ロール] を選択します。次に、AWS ServiceRoleForDocDB-Elastic ロールの名前 (チェックボックスではありません) を選択します。
3. 選択したロールの [概要] ページで、[アクセスアドバイザー] タブを選択します。

> **Note**
>
> Amazon DocumentDB が AWS ServiceRoleForDocDB-Elastic ロールを使用しているかどうか不明な場合は、ロールの削除を試みることができます。サービスがロールを使用している場合、削除は失敗し、ロールが使用されている AWS リージョン を表示できます。ロールが使用されている場合は、ロールを削除する前にセッションが終了するのを待つ必要があります。サービスにリンクされたロールのセッションを取り消すことはできません。AWS ServiceRoleForDocDB-Elastic ロールを削除する場合は、最初にすべてのクラスターを削除する必要があります。

すべてのクラスターの削除

Amazon DocumentDB コンソールからクラスターを削除する

1. AWS マネジメントコンソール にサインインし、Amazon DocumentDB コンソール を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択してください。
3. 削除するクラスターを選択します。
4. [アクション] で、[削除] を選択します。
5. [最終スナップショットを作成しますか?] が表示されたら、[はい] または [いいえ] を選択します。
6. 前のステップで [はい] を選択した場合は、[最終スナップショット名] に最終スナップショットの名前を入力します。
7. [削除] を選択します。

> **Note**
>
> `AWS ServiceRoleForDocDB-Elastic` サービスにリンクされたロールは、IAM コンソール、IAM CLI、または IAM API を使用して削除することができます。詳細については、「IAM ユーザーガイド」の「サービスにリンクされたロールの削除」を参照してください。

Amazon DocumentDB クォータ

このトピックでは、Amazon DocumentDB (MongoDB 互換) のリソースクォータと命名に関する制約について説明します。

特定の管理機能では、Amazon DocumentDB は Amazon Relational Database Service (Amazon RDS) と Amazon Neptune を用いて共有される運用テクノロジーを使用します。

トピック

- サポートされるインスタンスタイプ
- サポート対象のリージョン
- リージョン別クォータ
- 集計クォータ
- クラスタークォータ
- インスタンスクォータ
- 命名に関する制約
- TTL の制約
- Elastic クラスターのクォータ
- Elastic クラスターシャードクォータ
- シャードあたりのエラスティッククラスター CPU、メモリ、接続、カーソルのクォータ

サポートされるインスタンスタイプ

Amazon DocumentDB は、オンデマンドインスタンスと以下のインスタンスタイプをサポートしています。

- NVMe-backed:
  - R6GD インスタンスタイプ:
    `db.r6gd.xlarge`、`db.r6gd.2xlarge`、`db.r6gd.4xlarge`、`db.r6gd.8xlarge`、`db.r6gd.12x`
- メモリ最適化:
  - R8G インスタンスタイプ:
    `db.r8g.large`、`db.r8g.xlarge`、`db.r8g.2xlarge`、`db.r8g.4xlarge`、、`db.r8g.8xlarge`、

  - R6G インスタンスタイプ:
    db.r6g.large、db.r6g.xlarge、db.r6g.2xlarge、db.r6g.4xlarge、、db.r6g.8xlargedb
  - R5 インスタンスタイプ:
    db.r5.large、db.r5.xlarge、、db.r5.2xlarge、db.r5.4xlarge、db.r5.8xlargedb.r5.1
  - R4 インスタンスタイプ:
    db.r4.large、db.r4.2xlarge、db.r4.4xlarge、db.r4.8xlarge、db.r4.16xlarge。
- バースト可能パフォーマンス:
  - T4G インスタンスタイプ: db.t4g.medium。
  - T3 インスタンスタイプ db.t3.medium。

サポートされているインスタンスのタイプと仕様の詳細については、「インスタンスクラスの仕様」を参照してください。

サポート対象のリージョン

Amazon DocumentDB は、以下から入手できます AWS リージョン。

リージョン名	リージョン	アベイラビリティーゾーン (コンピューティング)
米国東部 (オハイオ)	us-east-2	3
米国東部 (バージニア北部)	us-east-1	6
米国西部 (オレゴン)	us-west-2	4
アフリカ (ケープタウン)	af-south-1	3
南米 (サンパウロ)	sa-east-1	3
アジアパシフィック (香港)	ap-east-1	3
アジアパシフィック (ハイデラバード)	ap-south-2	3
アジアパシフィック (マレーシア)	ap-southeast-5	3
アジアパシフィック (ムンバイ)	ap-south-1	3
アジアパシフィック (大阪)	ap-northeast-3	3
アジアパシフィック (ソウル)	ap-northeast-2	4
アジアパシフィック (シンガポール)	ap-southeast-1	3

リージョン名	リージョン	アベイラビリティーゾーン (コンピューティング)
アジアパシフィック (シドニー)	ap-southeast-2	3
アジアパシフィック (ジャカルタ)	ap-southeast-3	3
アジアパシフィック (メルボルン)	ap-southeast-4	3
アジアパシフィック (タイ)	ap-southeast-7	3
アジアパシフィック (東京)	ap-northeast-1	3
カナダ (中部)	ca-central-1	3
カナダ西部 (カルガリー)	ca-west-1	3
中国 (北京) リージョン	cn-north-1	3
中国 (寧夏)	cn-northwest-1	3
欧州 (フランクフルト)	eu-central-1	3
欧州 (チューリッヒ)	eu-central-2	3
欧州 (アイルランド)	eu-west-1	3
欧州 (ロンドン)	eu-west-2	3
欧州 (ミラノ)	eu-south-1	3
欧州 (パリ)	eu-west-3	3
欧州 (スペイン)	eu-south-2	3

リージョン名	リージョン	アベイラビリティーゾーン (コンピューティング)
欧州 (ストックホルム)	eu-north-1	3
メキシコ (中部)	mx-central-1	3
中東 (アラブ首長国連邦)	me-central-1	3
イスラエル (テルアビブ)	il-central-1	3
AWS GovCloud (米国西部)	us-gov-west-1	3
AWS GovCloud (米国東部)	us-gov-east-1	3

リージョン別クォータ

特定の管理機能では、Amazon DocumentDB は Amazon Relational Database Service (Amazon RDS) と共有の運用テクノロジーを使用します。次の表に、Amazon DocumentDB と Amazon RDS 間で共有されるリージョン別クォータを示します。

> **Note**
>
> 上記の Amazon RDS 共有テクノロジーは、Amazon DocumentDB インスタンスベースのクラスターにのみ適用されます。Amazon DocumentDB Elastic クラスターは、Amazon RDS とテクノロジーを共有しません。

以下のクォータは Amazon DocumentDB インスタンスベースのクラスターに適用され、リージョン AWS アカウント ごとに適用されます。

[リソース]	AWS デフォルトクォータ
クラスター	40

[リソース]	AWS デフォルトクォータ
クラスターパラメータグループ	50
パラメータグループあたりのクラスター	100
パラメータグループあたりのインスタンス数	300
イベントサブスクリプション	20
インスタンス	40
手動クラススナップショット	100
クラスターあたりのリードレプリカ	15
サブネットグループ	50
サブネットグループあたりのサブネット	20
リソースあたりのタグ	50
インスタンスあたりの VPC セキュリティグループ	5

次のクォータは Amazon DocumentDB エラスティッククラスターに適用され、リージョン AWS アカウント ごとに ごとに適用されます。

[リソース]	AWS デフォルトクォータ
Elastic クラスター	20
Elastic クラスター vCPU	1024
手動 Elastic クラススナップショット	20

クォータが調整可能な場合は、Service Quotas を使用してクォータの引き上げを要求できます。一部のリクエストは自動的に解決され、他のリクエストは送信されます サポート。送信されたクォータ引き上げリクエストのステータスを追跡できます サポート。サービスクォータを引き上げるリ

クエストは、優先サポートを受けません。緊急のリクエストがある場合は、 にお問い合わせくださいサポート。サービスクォータの詳細については、「Service Quotas とは」を参照してください。

Amazon DocumentDB のクォータの引き上げをリクエストするには

1. https://console.aws.amazon.com/servicequotas にある Service Quotas コンソールを開き、必要に応じてサインインします。
2. ナビゲーションペインで、[AWS のサービス] を選択します。
3. [Amazon DocumentDB (MongoDB 互換性)] もしくは [Amazon DocumentDB Elastic クラスター] を、リストから選択するかもしくは検索フィールドに入力します。
4. クォータが調整可能な場合は、そのラジオボタンまたは名前を選択し、ページの右上にある [クォータ引き上げリクエスト] を選択します。
5. [クォータ値を変更] に、新しい値を入力します。新しい値は現在値よりも大きい値である必要があります。
6. [リクエスト] を選択します。リクエストが解決されると、クォータの [適用されたクォータ値] が新しい値に設定されます。
7. 保留中または最近解決されたリクエストを表示するには、ナビゲーションペインから [ダッシュボード] を選択します。保留中のリクエストの場合は、リクエストのステータスを選択してリクエストの受信をオープンします。リクエストの初期ステータスは `Pending` です。ステータスがに変わると`Quota requested`、ケース番号が表示されます サポート。リクエストのチケットを開くには、ケース番号を選択します。

集計クォータ

次の表に、Amazon DocumentDB の集計クォータを示します。

[リソース]	クォータ
サポートされるステージの最大数	500

クラスタークォータ

次の表は、Amazon DocumentDB インスタンスベースのクラスタークォータを示しています。

[リソース]	クォータ
クラスターサイズ (すべてのコレクションとインデックスの合計)	エンジンバージョン 8.0 以降では 256 TiB (以前のエンジンバージョンでは 128 TiB)
コレクションサイズ (すべてのコレクションの合計がクラスタークォータを超えることはできません) – インデックスサイズは含まれません	32 TiB
クラスターあたりのコレクション数	100,000
クラスターあたりのデータベース数	100,000
データベースサイズ (すべてのデータベースの合計がクラスタークォータを超えることはできません)	エンジンバージョン 8.0 以降では 256 TiB (以前のエンジンバージョンでは 128 TiB)
ドキュメントのネストの深さ	200 レベル
ドキュメントサイズ	16 MiB
インデックスキーサイズ	2,048 バイト
コレクションごとのインデックス	64

[リソース]	クォータ
複合インデックスのキー	32
1 回のバッチコマンドでの書き込みの最大数	100,000
クラスターあたりのユーザー数	1,000

インスタンスクォータ

次の表は、インスタンスあたりの Amazon DocumentDB クォータを示しています。

Amazon DocumentDB 8.0

インスタンスタイプ	インスタンスメモリ（GiB）	接続 (すべて)	カーソルクォータ	同時に開くことのできるトランザクション数	接続 (アクティブ)
T3.medium	4	1,000	30	50	102
T4G.medium	4	1,000	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

インスタンスタイプ	インスタンスメモリ（GiB）	接続 (すべて)	カーソルクォータ	同時に開くことのできるトランザクション数	接続 (アクティブ)
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

* R6GD を含む

Amazon DocumentDB 5.0

インスタンスタイプ	インスタンスメモリ（GiB）	接続 (すべて)	カーソルクォータ	同時に開くことのできるトランザクション数	接続 (アクティブ)
T3.medium	4	1,000	30	50	102
T4G.medium	4	1,000	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

インスタンスタイプ	インスタンスメモリ（GiB）	接続 (すべて)	カーソルクォータ	同時に開くことのできるトランザクション数	接続 (アクティブ)
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

* R6GD を含む

Amazon DocumentDB 4.0

インスタンスタイプ	インスタンスメモリ（GiB）	接続 (すべて)	カーソルクォータ	同時に開くことのできるトランザクション数	接続 (アクティブ)
T3.medium	4	1,000	30	50	102
T4G.medium	4	1,000	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

インスタンスタイプ	インスタンスメモリ（GiB）	接続 (すべて)	カーソルクォータ	同時に開くことのできるトランザクション数	接続 (アクティブ)
R6G.16xlarge*	512	60000	3040	6400	4500

* R6GD を含む

Amazon DocumentDB 3.6

インスタンスタイプ	インスタンスメモリ（GiB）	接続 (すべて)	カーソルクォータ	同時に開くことのできるトランザクション数	接続 (アクティブ)
T3.medium	4	500	30	該当なし	102
R4.large	15.25	1700	450	該当なし	1100
R4.xlarge	30.5	3400	450	該当なし	2700
R4.2xlarge	61	6800	450	該当なし	4500
R4.4xlarge	122	13600	725	該当なし	4500
R4.8xlarge	288	27200	1450	該当なし	4500
R4.16xlarge	488	30000	2900	該当なし	4500
R5.large	16	1700	450	該当なし	1100
R5.xlarge	32	3500	450	該当なし	2700
R5.2xlarge	64	7100	450	該当なし	4500
R5.4xlarge	128	14200	760	該当なし	4500
R5.8xlarge	256	28400	1520	該当なし	4500

インスタンスタイプ	インスタンスメモリ（GiB）	接続 (すべて)	カーソルクォータ	同時に開くことのできるトランザクション数	接続 (アクティブ)
R5.12xlarge	384	30000	2280	該当なし	4500
R5.16xlarge	512	30000	3040	該当なし	4500

次の CloudWatch メトリクスを使用して、インスタンスごとの制限を監視およびアラームできます。Amazon DocumentDB CloudWatch のメトリクスのさらなる詳細については、Amazon DocumentDB と CloudWatch のモニタリング をご参照ください。

[リソース]	CloudWatch クォータメトリクス	CloudWatch の使用状況メトリクス (最大 1 分)	CloudWatch の使用状況メトリクス
インスタンスメモリ	-	-	FreeableMemory
接続 (すべて)	DatabaseConnectionsLimit	DatabaseConnectionsMax	DatabaseConnections
カーソル	DatabaseCursorsLimit	DatabaseCursorsMax	DatabaseCursors
取引	TransactionsOpenLimit	TransactionsOpenMax	TransactionsOpen

命名に関する制約

次の表に、Amazon DocumentDB 内の命名に関する制約を示します。

[リソース]	デフォルトのクォータ
クラスター識別子	• 長さは、[1 ~ 63] 個の英字、数字、またはハイフンです。 • 1 字目は英字である必要があります。 • ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。 • AWS アカウントごと、リージョンごとに、すべてのクラスター (Amazon RDS、Amazon Neptune、Amazon DocumentDB 全体) で一意である必要があります。
インスタンス識別子	• 長さは、[1 ~ 63] 個の英字、数字、またはハイフンです。 • 1 字目は英字である必要があります。 • ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません • AWS アカウント、リージョンごとに、すべてのインスタンス (Amazon RDS、Amazon Neptune、Amazon DocumentDB 全体) で一意である必要があります。

[リソース]	デフォルトのクォータ
コレクション名: <col>	長さは [1 ~ 255] 文字です (Amazon DocumentDB 5.0 以降) 。 長さは [1 ~ 57] 文字 (Amazon DocumentDB 4.0 以前) です。
データベース名: <db>	長さは [1 ~ 63] 文字です。
完全修飾コレクション名: <db>.<col>	長さは [3 ~ 255] 文字 (Amazon DocumentDB 5.0 以降) です。 長さは [3 ~ 120] 文字 (Amazon DocumentDB 4.0 以前) です。
完全修飾インデックス名: <db>.<col>$<index>	長さは [6 ~ 511] 文字 (Amazon DocumentDB 5.0 以降) です。 長さは [6 ~ 127] 文字 (Amazon DocumentDB 4.0 以前) です。
インデックス名	長さは [1 ~ 255] 文字です (Amazon DocumentDB 5.0 以降) 。 合計の長さは [3 ~ 63] 文字です <col>$<index> (Amazon DocumentDB 4.0 以前) 。

[リソース]	デフォルトのクォータ

プライマリパスワード	• 長さは、[8 ~ 100] 個の印刷可能な ASCII 文字です。
	• 以下のものを除くすべての印刷可能な ASCII 文字を使用できます。
	  • **/** (スラッシュ)
	  • **"** (二重引用符)
	  • **@** (アットマーク)

プライマリユーザー名	• 長さは、[1 ~ 63] 個の英数字です。 • 1 字目は英字である必要があります。 • データベースエンジンの予約語は使用できません。
パラメータグループ名	• 長さは、[1 ~ 255] 個の英数字です。 • 1 字目は英字である必要があります。 • ハイフンで終わることも、2 つのハイフンを連続して含むこともできません。

TTL の制約

TTL インデックスからの削除は、特定の期間内に保証されるものではなく、ベストエフォートです。インスタンスリソースの使用率、ドキュメントサイズ、全体的なスループットなどの要因は、TTL 削除のタイミングに影響を与える可能性があります。

Elastic クラスターのクォータ

次の表は、Amazon DocumentDB エラスティッククラスターの最大クォータを示しています。

[リソース]	クォータ
リージョンあたりの Elastic クラスター	20
リージョンあたりのすべての Elastic クラスターの合計 vCPU	1024
リージョンあたりの手動クラススナップショット	20
クラスターあたりのシャード数	32
クラスターあたりのストレージ (データがシャードキーによって均等に分配されている場合)	4 PiB
クラスターへの接続	300,000 またはシャード数のいずれかの低い値 x シャードあたりの vCPU に関連付けられた接続クォータ
シャーディングされていないコレクションのサイズ	32 TiB
シャーディングされたコレクションのサイズ (データがシャードキーによって均等に分配されている場合)	1 PB
クラスターあたりのデータベース数	10,000
クラスターあたりのシャーディングされていないコレクション数	100,000
クラスターあたりのシャーディングされたコレクション数	1,000
クラスターあたりのユーザー数	100
1 回のバッチコマンドでの書き込み数	100,000
コレクションごとのインデックス	64

[リソース]	クォータ
ドキュメントのネストの深さ	100 レベル
ドキュメントサイズ	16 MB
インデックスキーサイズ	2048 バイト
複合インデックスのキー	32

Elastic クラスターシャードクォータ

次の表は、Amazon DocumentDB エラスティッククラスターの最大シャードクォータを示しています。

[リソース]	クォータ
シャードインスタンスあたりの vCPU 数	64
シャードあたりのインスタンス数	16
1 シャードあたりのストレージ	128 TiB
コレクションあたり、シャードあたりのストレージ	32 TiB

シャードあたりのエラスティッククラスター CPU、メモリ、接続、カーソルのクォータ

次の表は、Amazon DocumentDB Elastic クラスターシャードにおける最大の CPU、メモリ、接続、およびカーソル限度を示します。

シャードあたりの vCPU 数	インスタンスメモリ（GiB）	接続クォータ	カーソルクォータ
2	16	1700	450

シャードあたりの vCPU 数	インスタンスメモリ（GiB）	接続クォータ	カーソルクォータ
4	32	3500	450
8	64	7100	450
16	128	14200	760
32	256	28400	1520
48	384	30000	2280
64	512	30000	3040

Amazon DocumentDB のトラブルシューティング

以下のセクションでは、Amazon DocumentDB (MongoDB 互換) の使用時に発生する可能性がある問題をトラブルシューティングする方法について説明します。

トピック

- 接続性の問題のトラブルシューティング
- トラブルシューティングインデックス
- パフォーマンスとリソース使用率のトラブルシューティング
- Amazon DocumentDB のガベージコレクション
- エラーコード

接続性の問題のトラブルシューティング

接続に問題がありますか。一般的なシナリオと、それらを解決する方法は、次の通りです。

トピック

- Amazon DocumentDB エンドポイントに接続できない
- Amazon DocumentDB インスタンスへの接続のテスト
- 無効なエンドポイントへの接続
- 接続数に影響するドライバー設定

Amazon DocumentDB エンドポイントに接続できない

Amazon DocumentDB に接続しようとすると、以下のような最も一般的なエラーメッセージが表示されることがあります。

```
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
```

このエラーメッセージは、クライアント (この例では mongo シェル) が Amazon DocumentDB エンドポイントにアクセスできないことを意味します。これには以下のようにいくつかの原因が考えられます。

トピック

- パブリックエンドポイントからの接続
- クロスリージョン接続
- 別の Amazon VPC からの接続
- セキュリティグループがインバウンド接続をブロックする
- Java Mongo ドライバーの読み取り設定に関する問題

パブリックエンドポイントからの接続

ノートパソコンやローカル開発マシンから直接 Amazon DocumentDB クラスターに接続しようとしています。

ノートパソコンやローカル開発マシンなどのパブリックエンドポイントから Amazon DocumentDB クラスターに直接接続しようとしても失敗します。Amazon DocumentDB は仮想プライベートクラウド (VPC) 専用で、現在パブリックエンドポイントをサポートしていません。したがって、ノートパソコンまたは VPC の外部のローカル開発環境から直接 Amazon DocumentDB クラスターに接続することはできません。

Amazon VPC の外部から Amazon DocumentDB クラスターに接続するには、SSH トンネルを使用します。詳細については、「Amazon VPC の外部から Amazon DocumentDB クラスターへの接続」を参照してください。さらに、開発環境が別の Amazon VPC にある場合は、VPC ピアリングを使用して、同じリージョンまたは別のリージョンの別の Amazon VPC から Amazon DocumentDB クラスターに接続することもできます。

クロスリージョン接続

別のリージョンの Amazon DocumentDB クラスターに接続しようとしています。

クラスターのリージョン以外のリージョンにある Amazon EC2 インスタンスから Amazon DocumentDB クラスターに接続しようとするとします。例えば、米国西部 (オレゴン) リージョン

(us-west-2) から米国東部 (バージニア北部) リージョン (us-east-1) からクラスターに接続しようとしても失敗します。

Amazon DocumentDB クラスターのリージョンを確認するには、以下のコマンドを実行します。そのリージョンはエンドポイントにあります。

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

このオペレーションによる出力は、次のようになります。

```
[
    "sample-cluster.node.us-east-1.docdb.amazonaws.com"
]
```

EC2 インスタンスのリージョンを確認するには、以下のコマンドを実行します。

```
 aws ec2 describe-instances \
     --query 'Reservations[*].Instances[*].Placement.AvailabilityZone'
```

このオペレーションによる出力は、次のようになります。

```
[
    [
        "us-east-1a"
    ]
]
```

別の Amazon VPC からの接続

クラスターのデプロイ先の Amazon VPC とは異なる VPC から Amazon DocumentDB クラスターに接続しようとしています。

Amazon DocumentDB クラスターと Amazon EC2 インスタンスの両方が同じ にあり AWS リージョン、同じ Amazon VPC にない場合、2 つの Amazon VPCs 間で VPC ピアリングが有効になっていない限り、Amazon DocumentDB クラスターに直接接続することはできません。

Amazon DocumentDB インスタンスの Amazon VPC を確認するには、以下のコマンドを実行します。

```
aws docdb describe-db-instances \
   --db-instance-identifier sample-instance \
   --query 'DBInstances[*].DBSubnetGroup.VpcId'
```

Amazon ÉC2 インスタンスの Amazon VPC を確認するには、以下のコマンドを実行します。

```
aws ec2 describe-instances \
   --query 'Reservations[*].Instances[*].VpcId'
```

セキュリティグループがインバウンド接続をブロックする

Amazon DocumentDB クラスターに接続しようとしているが、クラスターのセキュリティグループはクラスターのポート (デフォルトポート: 27017) でインバウンド接続を許可していない。

Amazon DocumentDB クラスターと Amazon EC2 インスタンスが両方とも同じリージョン内および同じ Amazon VPC 内にあり、同じ Amazon VPC セキュリティグループを使用しているとします。Amazon DocumentDB クラスターに接続できない場合は、クラスターのセキュリティグループ (ファイアウォール) で、Amazon DocumentDB クラスター用に選択したポート (デフォルトポートは 27017) でインバウンド接続を許可していないことが原因と考えられます。

Amazon DocumentDB クラスターのポートを確認するには、以下のコマンドを実行します。

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

Amazon DocumentDB クラスターのセキュリティグループを取得するには、以下のコマンドを実行します。

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

セキュリティグループのインバウンドルールを確認するには、Amazon EC2 ドキュメントの以下のトピックを参照してください。

- Linux インスタンスのインバウンドトラフィックの承認
- Windows インスタンスのインバウンドトラフィックの承認

Java Mongo ドライバーの読み取り設定に関する問題

クライアントの読み取り設定は適用されず、一部のクライアントは再起動しない限りフェイルオーバー後に Amazon DocumentDB に書き込むことができません。

この問題は、Java Mongo ドライバー 3.7.x で初めて発見されましたが、クライアントが `MongoClientSettings` を使用して Amazon DocumentDB への接続を確立したとき、具体的には `applyToClusterSettings` メソッドをチェーニングしたときに発生します。MongoClient クラスター設定は、`hosts()`、`requiredReplicaSetName()`、`mode()` などのいくつかの異なる方法を使用して定義できます。

クライアントが `hosts()` メソッドでホストを 1 つだけ指定した場合、モードは `ClusterConnectionMode.MULTIPLE` の代わりに `ClusterConnectionMode.SINGLE` に設定されます。これにより、クライアントは読み取り設定を無視し、`hosts()` で設定されているサーバーにのみ接続します。そのため、クライアントの設定が以下のように初期化されても、すべての読み取りはセカンダリではなくプライマリに送られます。

```
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);
```

フェイルオーバーケース

上記のクライアント接続設定を使用すると、クラスターライターエンドポイントのフェイルオーバーが発生し、DNS レコードの更新が遅れた場合でも、クライアントは古いライター (フェイルオーバー後にリーダーになる) への書き込みを試みます。その結果、(プライマリではない) サーバー側のエラーが発生し、Java ドライバーでは適切に処理することができません (この問題についてはまだ調査中です)。そのため、たとえばアプリケーションサーバーが再起動されるまで、クライアントは不正な状態のままになる可能性があります。

これには次の 2 つの回避策があります。

- 接続文字列を使用して Amazon DocumentDB に接続するクライアントは、読み込み設定を設定するときに ClusterConnectionMode が MULTIPLE に設定されるため、この問題は発生しません。

```
MongoClientURI mongoClientURI = new MongoClientURI("mongodb://usr:pass:cluster-
endpoint:27317/test?ssl=false&replicaSet=rs0&readpreference=secondaryPreferred");
MongoClient mongoClient = MongoClients.create(mongoClientURI.getURI());
```

  または、applyConnectionString メソッドで MongoClientSettings ビルダーを使用することもできます。

```
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);
```

- 明示的に ClusterConnectionMode を MULTIPLE に設定します。これは applyToClusterSettings と hosts().size() == 1 を使用する場合にのみ必要です。

```
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);
```

Amazon DocumentDB インスタンスへの接続のテスト

一般的な Linux または Windows ツールを使用して、クラスターへの接続をテストできます。

Linux または UNIX のターミナルからは、次のように入力することで接続をテストします (`cluster-endpoint` と `port` をインスタンスのエンドポイントとポートに置き換えてください)。

```
nc -zv cluster-endpoint port
```

サンプルのオペレーションと戻り値の例を次に示します。

```
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!
```

無効なエンドポイントへの接続

Amazon DocumentDB クラスターに接続し、有効でないクラスターエンドポイントを使用すると、以下のようなエラーが表示されます。

```
mongo --ssl \
   --host sample-cluster.node.us-east-1.docdb.amazonaws.com:27017 \
   --sslCAFile global-bundle.pem \
   --username <user-name> \
   --password <password>
```

出力は次のようになります。

```
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
```

クラスターの有効なエンドポイントを取得するには、以下のコマンドを実行します。

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

インスタンスの有効なエンドポイントを取得するには、以下のコマンドを実行します。

```
aws docdb describe-db-instances \
   --db-instance-identifier sample-instance \
   --query 'DBInstances[*].[Endpoint.Address,Endpoint.Port]'
```

詳細については、「Amazon DocumentDB エンドポイントについて」を参照してください。

接続数に影響するドライバー設定

クライアントドライバーを使用して Amazon DocumentDB クラスターに接続する場合は、maxPoolSize 設定パラメータを考慮することが重要です。maxPoolSize 設定は、クライアントドライバーが接続プールに維持できる接続の最大数を決定します。

トラブルシューティングインデックス

次のトピックでは、インデックスまたはバックグラウンドインデックスの構築が失敗したときにどうすべきかを説明しています。

トピック

- インデックスの構築が失敗する
- バックグラウンドインデックスビルドのレイテンシーの問題と失敗
- データベースインデックスの肥大化

インデックスの構築が失敗する

Amazon DocumentDB は、インデックス作成プロセスの一部としてローカルストレージをインスタンス上で使用します。[FreeLocalStorage] CloudWatch メトリクス (CloudWatch -> Metrics -> DocDB -> Instance Metrics) を使用して、このディスクの使用状況をモニタリングできます。インデックスの構築がすべてのローカルディスクを消費して失敗すると、エラーが発生します。Amazon DocumentDB にデータを移行する場合、まずインデックスを作成してからデータを挿

入することをお勧めします。移行戦略やインデックスの作成については、Amazon DocumentDB のドキュメントや「オフラインメソッドを使用して、MongoDB から Amazon DocumentDB に移行する」のブログで Amazon DocumentDB の移行とアップグレード を参照してください。

既存のクラスターでインデックスを作成するときに、インデックスの構築に予想以上に時間がかかるか、失敗する場合は、インスタンスをスケールアップしてインデックスを作成し、インデックスの作成後にスケールダウンします。Amazon DocumentDB では、 AWS マネジメントコンソール または を使用してインスタンスサイズを数分ですばやくスケーリングできます AWS CLI。詳細については、「インスタンスクラスの管理」を参照してください。秒単位のインスタンス料金によって、使用したリソースの秒数分のみを支払います。

バックグラウンドインデックスビルドのレイテンシーの問題と失敗

Amazon DocumentDB のバックグラウンドインデックスビルドは、インデックスビルドが開始される前に開始されたプライマリインスタンス上のすべてのクエリの実行が完了するまで開始されません。クエリが長時間実行されている場合、バックグラウンドインデックスビルドはクエリが終了するまでブロックされるため、完了までに予想以上に時間がかかることがあります。コレクションが空であるとしても同様です。

フォアグラウンドインデックスビルドは、同じブロッキング動作を示しません。代わりに、フォアグラウンドインデックスビルドは、インデックスのビルドが完了するまで、コレクションを排他的にロックします。空のコレクションにインデックスを作成し、長時間実行されるクエリをブロックしないようにするには、フォアグラウンドインデックスビルドを使用します。

> **Note**
>
> Amazon DocumentDB では、任意の時点でコレクションにおいてバックグラウンドインデックスの作成が 1 つのみ許可されます。バックグラウンドインデックスの作成中に `createIndex()` や `dropIndex()` などの DDL (データ定義言語) オペレーションが発生すると、バックグラウンドインデックスの作成は失敗します。

データベースインデックスの肥大化

Amazon DocumentDB は、複数バージョンの同時実行制御 (MVCC) を使用して同時トランザクションを管理します。ドキュメントを削除または更新しても、以前のバージョンはコレクションとインデックスに「デッド」バージョンとして残ります。ガベージコレクションプロセスは、将来のオペレーションのために、これらのデッドバージョンからスペースを自動的に再利用します。

インデックスの肥大化は、デッドインデックスエントリや古いインデックスエントリの蓄積、またはページ内のフラグメント化により、コレクションのインデックスが大きくなった場合に発生します。レポートされる割合は、将来のインデックスエントリで使用できるインデックススペースの量を表します。この肥大化は、バッファキャッシュとストレージの両方の領域を消費します。肥大化を削除する場合は、インデックスを再構築する必要があります。

Example例

次のコマンドを実行して、インデックスの未使用のストレージを確認します。

```
db.coll.aggregate({$indexStats:{}});
```

結果は以下のようになります。

```
{
    "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
    }
}
```

reIndex コマンドを使用して、ダウンタイムなしでインデックスを再構築できます。これには、コレクション全体のスキャンが必要です。「reIndex を使用したインデックスのメンテナンス」を参照してください。

パフォーマンスとリソース使用率のトラブルシューティング

このセクションでは、Amazon DocumentDB デプロイにおける一般的な診断の問題に関する質問と解決策を示します。提供されている例では mongo シェルを使用し、個々のインスタンスに範囲を設定しています。インスタンスエンドポイントを見つけるには、「Amazon DocumentDB エンドポイントについて」を参照してください 。

トピック

- Mongo API を使用してコレクションに対して実行された挿入、更新、削除操作の数を確認する方法
- キャッシュのパフォーマンスを分析する方法
- 長時間実行されているクエリやブロックされているクエリを見つけて終了する方法
- クエリプランを参照してクエリを最適化する方法
- Elastic クラスターのクエリプランを確認する方法
- インスタンスで実行中のすべての操作をリストする方法
- クエリの進捗状況を知る方法
- システムの実行速度が突然遅くなった理由を判断する方法
- 1 つ以上のクラスターインスタンスで CPU 使用率が高くなる原因を特定する方法
- インスタンスで開いているカーソルを確認する方法
- 現在の Amazon DocumentDB エンジンのバージョンを確認する方法
- インデックスの使用状況を分析し、未使用のインデックスを特定する方法
- 欠落しているインデックスを特定する方法
- データベースコレクションの肥大化を確認するにはどうすればよいですか?
- 便利なクエリの概要

Mongo API を使用してコレクションに対して実行された挿入、更新、削除操作の数を確認する方法

特定のコレクションに対して実行された挿入、更新、削除操作の数を表示するには、そのコレクションで以下のコマンドを実行します。

```
db.collection.stats()
```

このコマンドの出力は、opCounters フィールドの下に以下のように記述されています。

- numDocsIns: このコレクションに挿入されたドキュメントの数。これには、insert および insertMany コマンドを使用して挿入されたドキュメントと、アップサートによって挿入されたドキュメントが含まれます。
- numDocsUpd: このコレクションで更新されたドキュメントの数。これには、update および findAndModify コマンドを使用して更新されたドキュメントが含まれます。
- NumDocsDel: このコレクションから削除されたドキュメントの数。これには deleteOne、deleteMany、remove、および findAndModify コマンドを使用して削除されたドキュメントも含まれます。
- LastReset: これらのカウンタが最後にリセットされた時刻。このコマンドによって提供される統計は、クラスターを起動/停止したり、インスタンスをスケールアップ/スケールダウンしたりするとリセットされます。

以下に示しているのは、db.collection.stats() 実行時の出力例です。

```
{
    "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)
}
```

この stats コマンドは、Mongo API を使用して挿入、更新、削除操作を行うコレクション固有のカウンターを表示する場合に使用してください。コレクション固有の操作カウンターを表示するもう 1 つの方法は、DML 監査を有効にすることです。Amazon DocumentDB と CloudWatch のモニタリング では、1 分間隔で発生した全コレクションの挿入、更新、削除操作の数を表示できます。

キャッシュのパフォーマンスを分析する方法

キャッシュのパフォーマンスを分析することで、データ取得の効率とシステムパフォーマンスを把握できます。分析は、キャッシュとディスクから読み取られるデータの量に基づいています。キャッシュのパフォーマンスを把握するために、キャッシュヒット (キャッシュから読み取られたデータ) とキャッシュミス (キャッシュに見つからず、ディスクから読み取られたデータ) の数に関するキャッシュ統計を提供します。特定のコレクションのキャッシュ統計は、そのコレクションで以下のコマンドを実行すると確認できます。

```
db.collection.stats()
```

このコマンドの出力の cacheStats フィールドの値は、コレクションのキャッシュ統計と、コレクション上に作成されたインデックスの合計キャッシュ統計を提供します。これらの統計を以下に示します。

- **collBlksHit**: このコレクションの操作中にキャッシュから読み取られたブロック数。
- **collBlksRead**: このコレクションの操作中にディスクから読み込まれた (キャッシュミス) ブロック数。
- **collHitRatio**: このコレクションのキャッシュヒット率 (`100 * [collBlksHit / (collBlksHit + collBlksRead)]`)。
- **idxBlksHit**: このコレクションで作成されたインデックスのキャッシュから読み取られたブロック数。
- **idxBlksRead**: このコレクションで作成されたインデックスのディスクから読み込まれた (キャッシュミス) ブロック数。

- idxHitRatio: このコレクションで作成されたインデックスのキャッシュヒット率 (100 * [idxBlksHit / (idxBlksHit + idxBlksRead)])。
- **lastReset**: これらの統計が最後にリセットされた時刻。db.collection.stats() によって提供される統計は、クラスターを起動/停止したり、インスタンスをスケールアップ/スケールダウンしたりするとリセットされます。

各インデックスの idxBlksHit および idxBlksRead フィールドの内訳は、indexStats コマンドを使用して確認することもできます。インデックス固有のキャッシュ統計は、以下のコマンドを実行して参照できます。

```
db.collection.aggregate([{$indexStats:{}}]).pretty()
```

各インデックスについて、以下のキャッシュ統計が cacheStats フィールドの下に表示されます。

- **blksHit**: このインデックスのキャッシュから読み取られるブロック数。
- **blksRead**: このインデックスのディスクから読み取られるブロック数。
- **blksHitRatio**: キャッシュヒット率は、100 * [blksHit / (blksHit + blksRead)] で計算され、小数点以下 4 桁に四捨五入されました。

長時間実行されているクエリやブロックされているクエリを見つけて終了する方法

ユーザークエリの実行は、最適でないクエリプランのために遅くなることも、リソースの競合のためにブロックされる可能性もあります。

クエリプランが最適でないために長時間実行されているクエリ、またはリソースの競合のためにブロックされているクエリを見つけるには、currentOp コマンドを使用します。コマンドをフィルタリングすると、終了する関連クエリのリストを絞り込むことができます。クエリを終了できるようにするには、長時間実行されているクエリに opid を関連付ける必要があります。

次のクエリでは、currentOp コマンドを使用して、ブロックされている、または 10 秒以上実行されているクエリをすべてリストします。

```
db.adminCommand({
    aggregate: 1,
    pipeline: [
```

```
        {$currentOp: {}},
        {$match:
            {$or: [
                {secs_running: {$gt: 10}},
                {WaitState: {$exists: true}}]}},
        {$project: {_id:0, opid: 1, secs_running: 1}}],
    cursor: {}
});
```

次に、クエリを絞り込み、10 秒以上実行されているクエリの opid を見つけて終了できます。

10 秒以上実行されているクエリを見つけ出して終了するには

1. クエリの opid を見つけます。

```
db.adminCommand({
    aggregate: 1,
    pipeline: [
        {$currentOp: {}},
        {$match:
            {$or:
                [{secs_running: {$gt: 10}},
                 {WaitState: {$exists: true}}]}}],
    cursor: {}
});
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "waitedMS" : NumberLong(0),
    "cursor" : {
        "firstBatch" : [
            {
                "opid" : 24646,
                "secs_running" : 12
            }
        ],
        "id" : NumberLong(0),
        "ns" : "admin.$cmd"
    },
    "ok" : 1
}
```

2. killOp オペレーションを使用してクエリを終了します。

```
db.adminCommand({killOp: 1, op: 24646});
```

クエリプランを参照してクエリを最適化する方法

クエリの実行が遅い場合は、関連するドキュメントを選択するためにクエリの実行でコレクションのフルスキャンが必要である可能性があります。場合によっては、適切なインデックスの作成により、クエリの実行を高速化できます。このシナリオを検出し、インデックスを作成するフィールドを決定するには、explain コマンドを使用します。

> **Note**
>
> Amazon DocumentDB は分散型で耐障害性が高く、自己復旧ストレージシステムを利用する専用データベースエンジンで MongoDB 3.6 API をエミュレートしています。その結果、Amazon DocumentDB と MongoDB では、クエリプランと explain() の出力が異なる場合があります。クエリプランを制御する場合は、$hint 演算子を使用して優先インデックスの選択を強制できます。

次のように、explain コマンドで改善対象のクエリを実行します。

```
db.runCommand({explain: {<query document>}})
```

以下に示しているのは、オペレーションの例です。

```
db.runCommand({explain:{
    aggregate: "sample-document",
    pipeline: [{$match: {x: {$eq: 1}}}],
    cursor: {batchSize: 1}}
});
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "queryPlanner" : {
        "plannerVersion" : 1,
```

```
        "namespace" : "db.test",
        "winningPlan" : {
            "stage" : "COLLSCAN"
        }
    },
    "serverInfo" : {
        "host" : "...",
        "port" : ...,
        "version" : "..."
    },
    "ok" : 1
}
```

前述の出力は、$match ステージでコレクション全体をスキャンし、各ドキュメントのフィールド "x" が 1 に等しいかどうか確認する必要があることを示しています。コレクションに多くのドキュメントがある場合、コレクションのスキャン (および全体的なクエリのパフォーマンス) は非常に低速になります。したがって、explain コマンドの出力に "COLLSCAN" が存在することは、適切なインデックスを作成することで、クエリのパフォーマンスを向上できることを示します。

この例では、クエリはすべてのドキュメントでフィールド "x" が 1 と等しいかどうかを確認します。したがって、フィールド "x" でインデックスを作成すると、クエリは完全なコレクションスキャンを回避し、インデックスを使用して関連ドキュメントをより早く返すことができます。

フィールド "x" でインデックスを作成した後、explain 出力は次のようになります。

```
{
    "queryPlanner" : {
         "plannerVersion" : 1,
         "namespace" : "db.test",
         "winningPlan" : {
             "stage" : "IXSCAN",
             "indexName" : "x_1",
             "direction" : "forward"
         }
    },
    "serverInfo" : {
        "host" : "...",
        "port" : ...,
        "version" : "..."
    },
    "ok" : 1
}
```

フィールド "x" にインデックスを作成すると、$match ステージはインデックススキャンを使用して、述語 "x = 1" を評価する必要があるドキュメントの数を減らすことができます。

小規模なコレクションで、パフォーマンスの向上がごくわずかである場合、Amazon DocumentDB クエリプロセッサはインデックスを使用しないことがあります。

Elastic クラスターのクエリプランを確認する方法

Elastic クラスターのクエリプランを調べるには、explain コマンドを使用します。以下は、シャードコレクションを対象とする検索クエリの explain 操作の例です。

```
db.runCommand(
   {
     explain: { find: "cities", filter: {"name": "Seoul"}}
   }
)
```

> **Note**
>
> Amazon DocumentDB は、目的別データベースエンジンで MongoDB をエミュレートします。その結果、クエリプランと explain() の出力は、Amazon DocumentDB と MongoDB の間で異なる場合があります。$hint オペレーターを使ってクエリプランを制御し、優先インデックスの選択を強制することができます。

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
  "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)
}
```

上記の出力は、3 つのシャードクラスターでの find クエリのクエリプランを示しています。各シャードには複数のデータパーティションがあり、それぞれ異なる入力ステージを設定できます。この例では、各シャード内の「PARTITION_MERGE」ステージで結果がマージされる前に、すべてのパーティションで「COLLSCAN」 (コレクションスキャン) が実行されます。その後、シャード全体の結果は「SHARD_MERGE」ステージでマージされ、その後クライアントに送り返されます。

インスタンスで実行中のすべての操作をリストする方法

ユーザーまたはプライマリユーザーが、診断およびトラブルシューティングのために、インスタンスで実行されている現在のすべてのオペレーションをリストアップしたい場合があります。(ユーザーの管理の詳細については、Amazon DocumentDB ユーザーの管理 を参照してください。)

mongo シェルで、次のクエリを使用して、Amazon DocumentDB インスタンスで実行中のすべてのオペレーションをリストできます。

```
db.adminCommand({currentOp: 1, $all: 1});
```

クエリは、インスタンス上で現在動作しているすべてのユーザークエリと内部システムタスクの完全なリストを返します。

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "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
}
```

次に示すのは、"desc" フィールドの有効な値です。

- **INTERNAL** - カーソルのクリーンアップや古いユーザーのクリーンアップタスクなどの内部システムタスク。
- **TTLMonitor** - 有効期限 (TTL) モニタリングスレッド。その実行ステータスが "active" フィールドに反映されています。
- **GARBAGE_COLLECTION** - 内部ガベージコレクタースレッド。
- **CONN** - ユーザーのクエリ。
- **CURSOR** — 操作はユーザーが「getMore」コマンドを呼び出して次の結果のバッチを取得するのを待っているアイドルカーソルです。この状態では、カーソルはメモリを消費していますが、計算は消費されません。

前の出力では、システムで実行されているすべてのユーザークエリもリストされます。各ユーザークエリはデータベースとコレクションのコンテキストで実行され、これらの 2 つを統合したものが、名前空間と呼ばれます。各ユーザークエリの名前空間は、"ns" フィールドで利用できます。

特定の名前空間で実行されているすべてのユーザークエリをリストする必要がある場合があります。したがって、前の出力は "ns" フィールドでフィルタリングする必要があります。次にフィルター処理の出力を達成するためのクエリの例を示します。クエリは、データベース "db" とコレクション "test" (つまり、"db.test" 名前空間) で現在実行されているすべてのユーザークエリをリストします。

```
db.adminCommand({aggregate: 1,
    pipeline: [{$currentOp: {allUsers: true, idleConnections: true}},
               {$match: {ns: {$eq: "db.test"}}}],
    cursor: {}
});
```

システムのプライマリユーザーは、すべてのユーザーのクエリと、すべての内部システムタスクを確認できます。他のすべてのユーザーは、各自のクエリのみを表示できます。

クエリと内部システムタスクの合計数が、デフォルトのバッチカーソルサイズを超えている場合、mongo シェルは残りの結果を表示するイテレーターオブジェクト 'it' を自動的に生成します。すべての結果がなくなるまで、コマンド 'it' の実行を継続します。

クエリの進捗状況を知る方法

ユーザーのクエリは、最適とはいえないクエリプランが原因で実行速度が遅くなったり、リソースの競合が原因でブロックされたりすることがあります。このようなクエリのデバッグは複数ステップのプロセスであり、同じステップを複数回実行することが必要になる場合があります。

デバッグの最初のステップでは、長時間実行されているか、ブロックされているすべてのクエリをリストします。次のクエリは、10 秒以上実行されているか、リソースを待機しているすべてのユーザークエリをリストします。

```
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: {}
                });
```

前述のクエリを定期的に繰り返して、クエリのリストが変更されているかどうかを確認し、実行時間が長いクエリまたはブロックされているクエリを識別します。

対象のクエリの出力ドキュメントに WaitState フィールドがある場合、クエリの実行が低速であるかブロックされている理由は、リソースの競合であることを示しています。リソースの競合は、I/O、内部システムタスク、またはその他のユーザークエリが原因である可能性があります。

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "waitedMS" : NumberLong(0),
    "cursor" : {
        "firstBatch" : [
            {
```

```
                "opid" : 201,
                "command" : {
                    "aggregate" : ...
                },
                "secs_running" : 208,
                "WaitState" : "IO"
            }
        ],
        "id" : NumberLong(0),
        "ns" : "admin.$cmd"
    },
    "ok" : 1
}
```

同じインスタンスで同時に実行されているさまざまなコレクションで多くのクエリが実行されているか、クエリを実行中のデータセットのインスタンスサイズが小さすぎると、I/O がボトルネックとなる場合があります。クエリが読み取り専用クエリである場合、各コレクションのクエリを別々のレプリカに分離することで、以前の状況を軽減できます。さまざまなコレクション間で同時更新を行う場合、またはデータセットに対するインスタンスサイズが小さすぎる場合の軽減策は、インスタンスをスケールアップすることです。

リソースの競合の原因が他のユーザークエリである場合、出力ドキュメントの "blockedOn" フィールドに、このクエリに影響しているクエリの "opid" が含まれます。"opid" を使用して、すべてのクエリの "WaitState" および "blockedOn" フィールドのチェーンに従って、チェーンの先頭にあるクエリを調べます。

チェーンの先頭にあるタスクが内部タスクである場合の唯一の緩和策は、クエリを終了し、後で再実行することです。

以下は、検索クエリが別のタスクによって所有されているコレクションロックでブロックされるサンプル出力です。

```
{
    "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
}
```

"WaitState" の値が "Latch"、"SystemLock"、"BufferLock"、"BackgroundActivity"、または "Other" の場合、リソース競合の原因は内部システムタスクにあります。この状態が長時間続く場合、唯一の緩和策は、クエリを終了し、後で再実行することです。

システムの実行速度が突然遅くなった理由を判断する方法

以下にシステムの速度が低下する一般的な理由を示します。

- 同時クエリ間の過剰なリソースの競合
- 時間の経過とともに増加するアクティブな同時クエリの数
- "GARBAGE_COLLECTION" などの内部システムタスク

時間の経過とともにシステムの使用状況をモニタリングするには、以下の "currentOp" クエリを定期的に実行し、外部ストアに結果を出力します。クエリは、システムの各名前空間内のクエリとオペレーションの数をカウントします。次に、システムの使用状況の結果を分析し、システムへの負荷を把握して適切なアクションを行うことができます。

```
db.adminCommand({aggregate: 1,
                 pipeline: [{$currentOp: {allUsers: true, idleConnections: true}},
                            {$group: {_id: {desc: "$desc", ns: "$ns", WaitState:
 "$WaitState"}, count: {$sum: 1}}}],
                 cursor: {}
                });
```

このクエリは、各名前空間で実行されているすべてのクエリとすべての内部システムタスクの合計を返します。また、名前空間ごとに存在する場合は、待機状態の一意の数を返します。

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{
    "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
}
```

上記の出力では、コレクションロックでブロックされている名前空間 "db.test" に 2 つのユーザークエリ、名前空間 "admin.$cmd" に 1 つのクエリ、および 1 つの内部 "TTLMonitor" タスクがあります。

出力が、ブロック待機状態の多くのクエリを示す場合は、「長時間実行されているクエリやブロックされているクエリを見つけて終了する方法」を参照してください。

1 つ以上のクラスターインスタンスで CPU 使用率が高くなる原因を特定する方法

以下のセクションは、インスタンスの CPU 使用率が高い原因を特定するのに役立ちます。結果は、ワークロードによって異なります。

- インスタンスが突然遅くなる原因を特定するには、「システムの実行速度が突然遅くなった理由を判断する方法」を参照してください。
- 特定のインスタンスで長時間実行されるクエリを特定して終了するには、「長時間実行されているクエリやブロックされているクエリを見つけて終了する方法」を参照してください。
- クエリが進行中かどうかを理解するには、「クエリの進捗状況を知る方法」を参照してください。
- クエリの実行に時間がかかる理由を確認するには、「クエリプランを参照してクエリを最適化する方法」を参照してください。
- 実行時間が長いクエリを長期間にわたって追跡するには、「Amazon DocumentDB オペレーションのプロファイリング」を参照してください 。

インスタンスの CPU 使用率が高い理由に応じて、以下の 1 つ以上の操作を行うと役立ちます。

- プライマリインスタンスで高い CPU 使用率が示され、レプリカインスタンスではそうではない場合、クライアントの読み込み優先設定 (`secondaryPreferred` など) を使用してレプリカ間で読み込みトラフィックを分散することを検討してください。詳細については、「レプリカセットとしての Amazon DocumentDB に接続する」を参照してください。

  レプリカを読み込みに使用すると、プライマリインスタンスでより多くの書き込みトラフィックを処理できるようになり、クラスターのリソースをより有効に活用できます。レプリカからの読み込みには結果整合性があります。
- CPU 使用率が高いことが書き込みワークロードの結果である場合、クラスターのインスタンスのサイズをより大きいインスタンスタイプに変更すると、ワークロードに対応する CPU コアの数が増えます。詳細については、「Amazon DocumentDB インスタンスの管理」および「インスタンスクラスの仕様」を参照してください。
- すべてのクラスターインスタンスで高い CPU 使用率が示され、ワークロードでレプリカが読み取りに使用されている場合、クラスターにレプリカを追加すると、読み取りトラフィックに使用できるリソースが増えます。詳細については、「クラスターへの Amazon DocumentDB インスタンスの追加」を参照してください。

インスタンスで開いているカーソルを確認する方法

Amazon DocumentDB インスタンスに接続している場合、コマンド `db.runCommand("listCursors")` を使用して、そのインスタンスで開いているカーソルを一覧表示できます。インスタンスタイプに応じて、特定の Amazon DocumentDB インスタンスでいつでも最大 4,560 個のアクティブなカーソルのクォータが開かれます。カーソルはインスタンスのリソースを利用するため、使用されなくなったカーソルを閉じます。詳細については、「Amazon DocumentDB クォータ」を参照してください。

```
db.runCommand("listCursors")
```

現在の Amazon DocumentDB エンジンのバージョンを確認する方法

現在の Amazon DocumentDB エンジンのバージョンを確認するには、次のコマンドを実行します。

```
db.runCommand({getEngineVersion: 1})
```

このオペレーションによる出力は、次のようになります (JSON 形式)。

```
{ "engineVersion" : "2.x.x", "ok" : 1 }
```

> **Note**
>
> Amazon DocumentDB 3.6 のエンジンバージョンは 1.x.x、Amazon DocumentDB 4.0 のエンジンバージョンは 2.x.x、Amazon DocumentDB 5.0 のエンジンバージョンは 3.x.x、Amazon DocumentDB 8.0 のエンジンバージョンは 4.x.x です。

インデックスの使用状況を分析し、未使用のインデックスを特定する方法

特定のコレクションのインデックスを識別するには、次のコマンドを実行します。

```
db.collection.getIndexes()
```

コレクションに対して実行された操作中にどれだけのインデックスが使用されているかを分析するには、collStats および indexStats コマンドを使用できます。インデックスを使用して実行されたスキャン (インデックススキャン) の総数とインデックスなしで実行されたスキャンの数 (コレクションスキャン) を比較するには、次のコマンドを実行します。

```
db.collection.stats()
```

このコマンドの出力には、以下の値が含まれます。

- **idxScans**: インデックスを使用してこのコレクションに対して実行されたスキャンの数。
- **collScans**: インデックスを使用せずにこのコレクションに対して実行されたスキャンの数。これらのスキャンでは、コレクション内のドキュメントを一度に 1 つずつ調べる必要がありました。
- **lastReset**: これらのカウンタが最後にリセットされた時刻。このコマンドによって提供される統計は、クラスターを起動/停止したり、インスタンスをスケールアップ/スケールダウンしたりするとリセットされます。

各インデックスの使用量の内訳は、次のコマンドの出力で参照できます。インデックスの維持に使用される不要なコンピューティング、ストレージ、I/O がなくなるため、パフォーマンスを向上させ、コストを削減するために、未使用のインデックスを定期的に特定して削除することがベストプラクティスです。

```
db.collection.aggregate([{$indexStats:{}}]).pretty()
```

このコマンドの出力では、コレクションで作成された各インデックスについて次の値が表示されます。

- **ops** - インデックスを使用したオペレーションの数。ワークロードが十分に長時間実行されており、ワークロードが安定していることがはっきりわかる場合、ops 値 0 は、インデックスがまったく使用されていないことを示しています。
- **numDocsRead**: このインデックスを使用する操作中に読み取られたドキュメントの数。
- **since** - インデックス使用状況に関する統計情報の収集が Amazon DocumentDB により開始されてからの時間。通常は、前回のデータベース再起動またはメンテナンスアクション以降の値です。
- **size** - このインデックスのサイズをバイト数で表す。

次の例は、上記のコマンドを実行したときの出力例です。

```
{
    "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" : ...
    }
}
```

コレクションの全体的なインデックスサイズを確認するには、次のコマンドを実行します。

```
db.collection.stats()
```

未使用のインデックスを削除するには、次のコマンドを実行します。

```
db.collection.dropIndex("indexName")
```

欠落しているインデックスを特定する方法

Amazon DocumentDB プロファイラを使用して、低速なクエリをログに記録 できます。低速なクエリのログに繰り返し現れるクエリがある場合、そのクエリのパフォーマンスを向上させるために追加のインデックスが必要であることを示している可能性があります。

少なくとも 1 つのステージを実行する 1 つ以上のステージを持つ長時間実行されるクエリを探すことで、有用なインデックスの機会を特定できます。つまりCOLLSCAN、クエリステージは、クエリへのレスポンスを提供するためにコレクション内のすべてのドキュメントを読み取る必要があります。

次の例は、大規模なコレクションで実行されたタクシー乗車のコレクションに対するクエリを示しています。

```
db.rides.count({"fare.totalAmount":{$gt:10.0}}))
```

この例を実行するには、fare.totalAmount フィールドにインデックスがないため、クエリはコレクションスキャン (つまり、コレクション内のすべてのドキュメントを読み込む) を実行する必要がありました。このクエリの Amazon DocumentDB プロファイラからの出力は、次のようになります。

```
{
    ...
    "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"
}
```

この例のクエリを高速化するには、以下に示すように fare.totalAmount にインデックスを作成します。

```
db.rides.createIndex( {"fare.totalAmount": 1}, {background: true} )
```

> **Note**
>
> フォアグラウンドで作成されたインデックス (つまり、インデックスの作成時に {background:true} オプションが指定されなかった場合) は、排他的な書き込みロックを使用します。これにより、インデックス構築が完了するまでアプリケーションがコレクションにデータを書き込めなくなります。本番稼働クラスターでインデックスを作成する場合は、このような影響が生じる可能性がある点に注意してください。インデックスを作成するときは、 を設定します{background:true}。

一般に、カーディナリティが高いフィールド (たとえば、多数の一意の値) でインデックスを作成します。カーディナリティの低いフィールドにインデックスを作成すると、使用されないインデックスが大きくなることがあります。Amazon DocumentDB クエリオプティマイザは、クエリプランの作成時に、コレクション全体のサイズとインデックスの選択性を考慮します。インデックスが存在する場合でも、クエリプロセッサが COLLSCAN を選択することがあります。これは、インデックスを利用しても、コレクション全体をスキャンした場合よりパフォーマンスが高くならないとクエリプ

ロセッサが推定したときに発生します。特定のインデックスの利用をクエリプロセッサに強制する場合、以下に示すように、hint() 演算子を使用できます。

```
db.collection.find().hint("indexName")
```

データベースコレクションの肥大化を確認するにはどうすればよいですか?

コレクションの肥大化は、デッドドキュメントや古いドキュメントの蓄積、またはデータベースページ内のフラグメント化により、コレクションのサイズが大きくなった場合に発生します。レポートされる割合は、将来のドキュメントで使用できるドキュメントスペースの量を表します。この肥大化は、バッファキャッシュとストレージの両方の領域を消費します。肥大化を取り除くには、ダンプ/復元、またはメンテナンスウィンドウ中の移行ループバックとスイッチオーバーを使用してコレクションを再ロードする必要があります。

Example例

次のコマンドを実行して、コレクションの未使用のストレージを確認します。

```
db.runCommand({collStats:'coll'})
```

結果は以下のようになります。

```
{
        "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)
}
```

便利なクエリの概要

以下のクエリは、Amazon DocumentDB のパフォーマンスとリソース使用率をモニタリングするのに役立ちます 。

- 以下のコマンドを使用して、オペレーションカウンター、キャッシュ統計、アクセス統計、サイズ統計など、特定のコレクションに関する統計を表示します。

  ```
  db.collection.stats()
  ```

- 以下のコマンドを使用して、コレクション上に作成された各インデックスに関する統計情報 (インデックスのサイズ、インデックス固有のキャッシュ統計、インデックス使用統計など) を表示します。

  ```
  db.collection.aggregate([{$indexStats:{}}]).pretty()
  ```

- 次のクエリを使用して、すべてのアクティビティをリストします。

  ```
  db.adminCommand({currentOp: 1, $all: 1});
  ```

- 次のコードは、実行時間が長いクエリまたはブロックされたすべてのクエリをリストします。

```
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: {}
                });
```

- 次のコードは、クエリを終了します。

```
db.adminCommand({killOp: 1, op: <opid of running or blocked query>});
```

- 次のコードを使用して、システム状態の集約ビューを取得します。

```
db.adminCommand({aggregate: 1,
                 pipeline: [{$currentOp: {allUsers: true, idleConnections: true}},
                            {$group: {_id: {desc: "$desc", ns: "$ns", WaitState:
 "$WaitState"}, count: {$sum: 1}}}],
                 cursor: {}
                });
```

Amazon DocumentDB のガベージコレクション

Amazon DocumentDB は、更新オペレーションごとにドキュメントエントリとインデックスエントリの新しいバージョンを作成するマルチバージョン同時実行制御 (MVCC) データベースアーキテクチャを実装します。このアーキテクチャにより、トランザクションの分離が可能になり、あるトランザクションの変更が別のトランザクションに表示されなくなります。

トピック

- Amazon DocumentDB のガベージコレクションについて
- ガベージコレクションプロセス
- ストレージアーキテクチャと拡張ストレージ
- ガベージコレクションのモニタリング

- collStats 出力の例
- よくある質問

Amazon DocumentDB のガベージコレクションについて

ガベージコレクション (GC) は、Amazon DocumentDB で最適なシステムパフォーマンスと可用性を維持する自動化されたバックグラウンドプロセスです。多くの最新のデータベースと同様に、Amazon DocumentDB の MVCC アーキテクチャは、更新ごとに新しいドキュメントバージョンとインデックスバージョンを作成します。各書き込みオペレーションは、有限カウンターから一意の MVCC ID を消費します。これらの IDsは、ドキュメントバージョンが属するトランザクションと、コミットされたかロールバックされたかを識別します。時間の経過とともに、これらの古いバージョンとその MVCC IDsが蓄積され、パフォーマンスの低下を防ぐためにクリーンアップが必要になります。

ガベージコレクションの機能

ガベージコレクターには 3 つの重要な機能があります。

- ストレージスペースの再利用 — アクティブなクエリで不要になった古いドキュメントバージョンとインデックスバージョンを削除し、将来の書き込みオペレーションのためのスペースを解放します。
- MVCC ID オーバーフローの防止 — MVCC ID の有限カウンターを管理することで、MVCC ID のオーバーフローを防止します。この管理を行わないと、カウンターは最終的に制限に達し、ID がリサイクルされるまで、データベースを強制的に一時的な読み取り専用モードにします。
- クエリパフォーマンスを維持 — そうしない場合にクエリ処理が蓄積および遅くなるデッドドキュメントバージョンを排除することで、最適なクエリパフォーマンスを維持します。

ガベージコレクションプロセス

GC プロセスはコレクションごとに動作し、異なるコレクションで複数のプロセスを同時に実行できます。このプロセスは 4 つのシーケンシャルフェーズで構成されます。

1. 識別 — システムは、アクティブなトランザクションまたはクエリによって参照されなくなったドキュメントバージョンとインデックスバージョンを識別します。
2. メモリロード—古いドキュメントとインデックスエントリがメモリに存在しない場合、メモリにロードされます。

3. 削除 — 古いバージョンは、ストレージ容量を再利用するために完全に削除されます。
4. MVCC ID リサイクル — システムは、新しいオペレーションのために、削除されたバージョンの MVCC ID をリサイクルします。

ガベージコレクションが古いドキュメントバージョンの処理を完了すると、システムから最も古い MVCC ID が削除されます。このクリーンアップは、MVCC ID をリサイクルして MVCC ID のオーバーフローを防ぎ、クラスター全体の新しい書き込みオペレーションで使用できるようにするために不可欠です。このリサイクルプロセスがないと、システムは最終的に有限の MVCC ID カウンターを使い果たし、読み取り専用状態になります。

ガベージコレクションのスケジューリング

ガベージコレクションは、定期的にバックグラウンドで自動的に実行されます。タイミングと頻度は、システム負荷、使用可能なリソース、書き込み容量、MVCC ID 消費レベルに基づいて動的に調整されます。書き込みアクティビティが多い場合、GC プロセスはより頻繁に実行され、ドキュメントバージョン数の増加を管理します。

ストレージアーキテクチャと拡張ストレージ

Amazon DocumentDB は、ドキュメントストレージを 2 つの異なるセグメントに分割する高度なストレージアーキテクチャを使用します。

ベースストレージセグメント

ベースストレージセグメントには、プライマリドキュメントデータとメタデータが含まれます。このセグメントは以下を保存します。

- 標準ページサイズ (8 KB) に収まるコンテンツを文書化します。
- ドキュメントメタデータと構造情報。
- プライマリインデックスとそのエントリ。
- コレクションレベルの統計と設定。

拡張ストレージセグメント

拡張ストレージセグメントは、標準ストレージページサイズを超えるドキュメントを処理するように設計された特殊なラージドキュメントオブジェクトストアを使用します。このセグメントは以下を提供します。

- 効率的な大規模ドキュメント処理 — ベースストレージのしきい値を超えるドキュメントは、拡張ストレージセグメントに自動的に移動されます。
- 最適化ストレージレイアウト — セグメントは、大きなオブジェクト用に最適化された異なるストレージ形式を使用し、断片化を減らし、アクセスパターンを改善します。
- 独立したガベージコレクション — 拡張ストレージセグメントには、ベースストレージのクリーンアップとは独立して実行できる独自のガベージコレクションプロセスがあります。
- 透過的なアクセス — アプリケーションは、データを含むストレージセグメントを知ることなく、大規模なドキュメントにシームレスにアクセスします。

拡張ストレージセグメントは、以下にとって特に有益です。

- 大きな埋め込み配列を含むドキュメントを含むコレクション。
- 広範なネストされた構造を持つドキュメント。
- バイナリデータまたは大きなテキストフィールドを保存するコレクション。
- 一部のドキュメントが平均サイズを大幅に超える、ドキュメントサイズが混在するアプリケーション。

ガベージコレクションのモニタリング

クラスターレベルのメトリクス

`AvailableMVCCIds`

- 場所 — Amazon CloudWatch
- 説明 — 最大制限 18 億から使用可能な残りの書き込みオペレーションの数を示すカウンター。このカウンターがゼロに達すると、ID が再利用されてリサイクルされるまで、クラスターは読み取り専用モードになります。カウンターは書き込みオペレーションごとに減少し、ガベージコレクションが古い MVCC ID をリサイクルすると増加します。
- 推奨事項 — Amazon DocumentDB 5.0 以降で値が 9 億を下回った場合、または Amazon DocumentDB 3.6 および 4.0 で Amazon DocumentDB 13 億を下回った場合にアラームを設定します。この早期警告により、後で説明する推奨手順を実行できます。

`LongestActiveGCRuntime`

- 場所 — Amazon CloudWatch

- 説明 — 最も長いアクティブなガベージコレクションプロセスの秒単位の期間。1 分以内に完了するプロセスを除き、1 分ごとに更新され、アクティブなオペレーションのみを追跡します。
- 推奨事項 — gcRuntimeStats履歴データと比較して、一括削除中の拡張ランタイムなど、異常なガベージコレクション動作を特定します。

コレクションレベルのメトリクス

MVCCIDStats: MVCCIdScale

- 場所 — Database collStats コマンド
- 説明 — 0 ~ 1 のスケールで MVCC ID の経過時間を測定します。1 は、クラスターが読み取り専用状態になる前の最大経過時間を示します。このメトリクスを AvailableMVCCIds と一緒に使用して、クラスターを古くしている最も古い MVCC ID を含むコレクションを識別します。
- 推奨事項 — コレクションごとに 0.3 未満の値を維持します。

gcRuntimeStats

- 場所 — Database collStats コマンド
- 説明 — 合計実行数、平均期間、最大期間など、ガベージコレクションメトリクスの 2 か月の履歴を提供します。意味のある統計を確保するために 5 分以上続くガベージコレクションオペレーションのみが含まれます。

storageSizeStats

- 場所 — Database collStats コマンド
- 説明 — さまざまなストレージセグメントのストレージ使用率の詳細な内訳を示します。
  - storageSegmentBase — 標準ドキュメントのベースストレージセグメントで使用されるストレージ
  - storageSegmentExtended — ラージドキュメントの拡張ストレージセグメントで使用されるストレージ
- 使用状況 — 大規模なドキュメントストレージを持つコレクションを特定し、ストレージの分散パターンを理解するのに役立ちます。

unusedStorageSize (コレクションレベル)

- 場所 — Database collStats コマンド
- 説明 — サンプル統計に基づいて、コレクション内の未使用のストレージ領域を推定します。削除されたドキュメントと空のセグメントからのスペースが含まれます。メトリクスは、合計とセグメントごとの内訳の両方を提供します。
  - すべてのストレージセグメントにまたunusedPercentがる unusedBytesと の組み合わせ
  - storageSegmentBase — 特にベースストレージセグメントで未使用のスペース
  - storageSegmentExtended — 拡張ストレージセグメントで特に未使用のスペース

documentFragmentStats

- 場所 — Database collStats コマンド
- 説明 — コレクション内のドキュメントフラグメントとデッドデータに関する詳細情報を提供します。ドキュメントフラグメントは、データベースエンジンで使用される内部ストレージユニットを表し、デッドフラグメントは、アクセスできなくなり、まだ再利用されていないデータを示します。このメトリクスには以下が含まれます。
  - totalDocFragmentsCount — コレクション内のドキュメントフラグメントの合計数
  - deadDocFragmentsCount — デッド (アクセスできない) データを含むフラグメントの数
  - deadDocFragmentsPercent — デッドデータを含むフラグメントの割合
  - deadDocFragmentBytes — デッドドキュメントフラグメントによって消費される推定バイト数
  - storageSegmentBase および のセグメントごとの内訳 storageSegmentExtended
- 使用状況 — このメトリクスをモニタリングしてガベージコレクションの有効性を理解し、メンテナンスオペレーションの恩恵を受ける可能性のあるコレクションを特定します。デッドフラグメントの割合が高い場合は、ガベージコレクションが遅れているか、最適化の恩恵を受ける可能性があります。

インデックスレベルのメトリクス

unusedStorageSize (インデックスレベル)

- 場所 — データベース indexStats コマンド
- 説明 — サンプル統計に基づいて、インデックス内の未使用のストレージ領域を推定します。これには、古いインデックスエントリと空のセグメントからのスペースが含まれます。

- 推奨事項 — reIndex コマンドを使用してダウンタイムなしでインデックスを再構築し、未使用のスペースを再利用します。詳細については、「インデックスの管理」を参照してください。

collStats 出力の例

次の例は、ガベージコレクションとストレージメトリクスを含む一般的なcollStats出力を示しています。

```
{
    "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)
}
```

よくある質問

ガベージコレクションが効率的に機能していないかどうかを確認するにはどうすればよいですか?

非効率的なガベージコレクションを示す次の警告サインをモニタリングします。

- 過剰なコレクション肥大化 — 特に大きなインデックスでは、大量の書き込みや一括削除中にunusedStorageSizeメトリクスが着実に増加します。
- 高いデッドフラグメントの割合 — 一貫して高いdeadDocFragmentsPercent値 (10 ~ 15% 以上) documentFragmentStatsを示します。
- クエリレイテンシーの低下 — デッドドキュメントの蓄積によるクエリレイテンシーの増加。
- 延長 GC 期間 — ガベージコレクションオペレーションには、 の履歴平均よりも時間がかかりますgcRuntimeStats。
- GC 処理の向上 — ガベージコレクターがシステムの需要に追いついていないLongestActiveGCRuntimeことを示す高。

ガベージコレクションはデータベースのパフォーマンスに影響しますか?

通常の条件下では、ガベージコレクションのパフォーマンスへの影響は最小限です。ただし、ガベージコレクションが遅れると、以下が発生する可能性があります。

- 蓄積されたデッドドキュメントのストレージコストが増加しました。
- 古いインデックスエントリにより、クエリのパフォーマンスが遅くなります。
- MVCC IDsが枯渇した場合の一時的な読み取り専用モード。
- 特に小規模なインスタンスでは、集中的な収集の実行中のリソース使用量が増加します。
- 大規模なドキュメントの拡張ストレージセグメントオペレーションの効率が低下しました。

ガベージコレクションを手動でトリガーできますか?

いいえ。Amazon DocumentDB のガベージコレクションを手動でトリガーすることはできません。システムは、内部メンテナンスオペレーションの一部としてガベージコレクションを自動的に管理します。

運用上のベストプラクティスとして設定する必要があるアラームは何ですか?

Amazon DocumentDB システムの最適なパフォーマンスを確保するために、クラスターレベルとコレクションレベルの両方でモニタリングを設定します。

クラスターレベルのモニタリングの場合、まず、Amazon DocumentDB 5.0 以降ではしきい値が 9 億、Amazon DocumentDB 3.6 および 4.0 では 13 億のメトリクスの Amazon CloudWatch アラームを作成します。 `AvailableMVCCIds` Amazon DocumentDB これにより、メトリクスが 0 に達する前にアクションを実行するのに十分な時間が与えられます。ゼロに達した時点で、クラスターは読み取り専用モードになります。このメトリクスは、特定の使用パターンに基づいて変動する場合があることに注意してください。一部のお客様は、ガベージコレクションが作業を完了するとしきい値を下回ってから 15 億を超えると回復します。

Amazon CloudWatch を介して`LongestActiveGCRuntime`メトリクスをモニタリングすることも重要です。このメトリクスは、`gcRuntimeStats` とともに、システム全体でのガベージコレクションのパフォーマンスを理解するのに役立ちます。

コレクションレベルのモニタリングでは、次の主要なメトリクスに焦点を当てます。

- `MVCCIdScale` — MVCC IDs が古く、注意が必要な可能性があることを示す値の増加に注意してください。
- `gcRuntimeStats` — ガベージコレクションプロセスが異常に長くかかるか、数日間にわたって延長されることを特定します。
- `documentFragmentStats` — `deadDocFragmentsPercent`値をモニタリングする - 一貫して高い割合 (10 ~ 15%) は、ガベージコレクションが遅れていることを示している可能性があります。
- `storageSizeStats` および `unusedStorageSize` — ストレージ使用率パターンを追跡し、いずれかのストレージセグメントに大量の未使用領域があるコレクションを特定します。

書き込みオペレーションが頻繁なコレクションでは、ガベージコレクターの作業が増えるため、特に注意が必要です。書き込みアクティビティが多いコレクションについては、これらのメトリクスをより頻繁にチェックして、ガベージコレクションがワークロードに遅れないようにします。

これらのモニタリングレコメンデーションが出発点となることに注意してください。システムの動作に慣れたら、特定の使用パターンと要件に合わせてこれらのしきい値を調整することをお勧めします。

がアラームしきい値を**AvailableMVCCIds**下回った場合はどうすればよいですか?

AvailableMVCCIds メトリクスがアラームしきい値を下回った場合は、クラスターが読み取り専用モードにならないようにすぐにアクションを実行してください。まず、インスタンスサイズをスケールアップして、ガベージコレクターにより多くのコンピューティングリソースを提供します。これにより、アプリケーションは通常のオペレーションを継続しながら、ガベージコレクターに追いつくために必要な追加のパワーを与えることができます。

単独でスケールアップしても状況が改善しない場合は、書き込みオペレーションを減らすことを検討してください。MVCCIdScale メトリクスを使用して、注意が必要な古い MVCC ID を含む特定のコレクションを特定します。さらに、 をモニタリングdocumentFragmentStatsして、ガベージコレクションの非効率に寄与している可能性のあるデッドフラグメントの割合が高いコレクションを特定します。

これらのコレクションを特定したら、ガベージコレクションが追いつくことができるように、それらのコレクションへの書き込みオペレーションを一時的に減らす必要がある場合があります。復旧期間中は、 AvailableMVCCIdsメトリクスを注意深くモニタリングして、アクションが望ましい効果を持っていることを確認します。AvailableMVCCIds 値が 15 億以上に戻ると、クラスターは正常と見なされます。

これらのステップは、システムが重大な状態になる前に復旧するための予防策であることに注意してください。メトリクスがしきい値を下回った後に早くアクションを実行すればするほど、書き込みオペレーションへの影響を回避できる可能性が高くなります。

バッチオペレーションは MVCC ID の消費量に影響しますか?

はい。単一のアトミックトランザクション内で実行されるバッチは 1 つの MVCC ID しか消費しないため、書き込みをバッチにグループ化すると、MVCC IDsが消費されるレートが低くなります。ベストプラクティスとして、updateMany可能な限り insertManyと を使用して MVCC ID の消費量を最小限に抑え、ガベージコレクションの負荷を軽減します。

が特定のしきい値に達するまで**AvailableMVCCIds**回復しないのはなぜですか?

ガベージコレクションの再利用サイクルをトリガーするのに十分な MVCC IDs をまだ消費していない書き込み主要ワークロードでは、復旧AvailableMVCCIdsなしで が一時的に着実に低下することが予想されます。ガベージコレクションは、コレクション内のドキュメントの一部が古くなった場合、新しいドキュメントの一部が挿入された場合、または再利用しきい値AvailableMVCCIdsを下回る場合にトリガーされます。

AvailableMVCCIds が再利用しきい値を下回るまで、強制再利用は発生しません。これを行うと、再利用がトリガーされ、回復AvailableMVCCIdsします。ただし、長時間実行されるトランザクションと開いているカーソルは、再利用期間を固定し、しきい値に達した後でも再利用を防ぐことができます。

エラーコード

コマンドが失敗すると、数値エラーcodeと説明メッセージ () を含むエラーが Amazon DocumentDB (MongoDB 互換) から送信されますerrmsg。次の表を使用して、Amazon DocumentDB 8.0 がクライアントアプリケーションに返す可能性のあるエラーコードを特定します。エラーコードが 1 つの条件に対応する場合、説明はサーバーから返されるメッセージテキストと一致します。複数の条件をカバーするコードには、一般的な説明があります。

> **Important**
>
> Amazon DocumentDB はメジャーバージョン間でエラーコードを変更する可能性があるため、特定の数値エラーコードに依存するアプリケーションロジックを構築しないでください。

エラーコード	Description
1	項目が見つかりません
2	コマンド引数、オプション、またはフィールド値が無効であるか、サポートされていません。
4	必須フィールドがないか、設定値が正しくフォーマットされていません。
5	ビュー定義はそれ自体を参照するか、サイクルを作成するか、ファイルシステムエラーが発生しました。
9	コマンド、フィールド、ヒント、または再開トークンを解析できなかったか、形式が無効です。
10	ターゲットコレクションが見つからないか、コレクションが指定されていません。

エラーコード	Description
11	参照されたユーザーが見つからなかったか、ターゲットコレクションが既に存在します。
13	このコマンドを実行する権限がないか、認証が無効です。
14	フィールドまたは値のデータ型が正しくありません。
15	値が評価中に許容範囲をオーバーフローしました。
16	書き込みバッチが空であるか、形式が正しくないか、許可されたオペレーション数を超えています。
17	すべてのオブジェクトが同じタイプであるわけではありません
18	認証情報が無効であるか、ユーザーが存在しないため、認証に失敗しました。
19	データ処理プリミティブオペレーションが見つからないか、プリミティブタイプが見つかりません
20	このコンテキストでは、リクエストされたオペレーションは許可されません。
26	ターゲットの名前空間またはコレクションが存在しません。
27	リクエストされたインデックスが存在しないか、適切なインデックスがありません。
37	パイプラインまたは集約ステージの引数がないか、空であるか、無効です。
38	データベース名が見つかりません
39	オペレーションは、maxTimeMS を超えたか、実行時間が長すぎるか、使用可能なメモリを超えたためキャンセルされました。
40	更新が別のフィールドと競合するか、ガベージコレクションによって書き込みがスロットリングまたは無効になります。
41	JavaScript エラー
43	指定されたカーソルが存在しないか、有効期限が切れています。

エラーコード	Description
44	配列インデックスが範囲外
48	その名前のコレクション、ビュー、または名前空間が既に存在します。
51	不正な更新演算子
52	フィールド名は、ストレージに無効な予約済み「$」プレフィックスを使用します。
53	_id フィールドには、配列や正規表現など、無効な型またはサポートされていない型があります。
54	は設定するクエリフィールドを推測できません
55	トランザクションは既にコミットされています
56	リクエストのフィールド名は空であり、許可されていません。
57	他のトランザクションによって無効化されたトランザクション
58	フィールド値が長すぎます
59	リクエストされたコマンドが不明であるか、サポートされていません。
61	無効なコレクション
62	コレクションを変更できない
65	バッチオペレーションエラーが発生しました
66	変更不可能な _id フィールドを変更しようとした更新。
67	指定が無効またはサポートされていないため、インデックスを作成できませんでした。
68	競合するプロパティを持つインデックスが既に存在します。
70	$snippet で使用される フィールドがスニペット列として設定されていない
71	インデックスは異なるオプションで既に存在します

エラーコード	Description
72	コマンドまたはインデックスに指定された 1 つ以上のオプションが無効です。
73	名前空間、データベース、またはコレクション名が無効であるか、このオペレーションで許可されていません。
74	コレクションにはテキストインデックスが定義されていません。
75	このコレクションにはテキストインデックスが定義されますが、このフィールドには定義されません。
77	ソートでスナップショットを使用できない
78	ドキュメントが見つからない
80	クエリが最大許容サイズを超えています。
82	時間枠の形式が無効です。有効な形式: '1 m' または '3 y' '<number> h/d/w/s/m/y'
85	その名前のインデックスは既に存在しますが、オプションは異なります。
86	リクエストされたインデックスは、同じ名前の既存のインデックスと競合します。
87	$sort キーの順序は 1 (昇順の場合) または -1 (降順の場合) にする必要があります
92	オペレーションに十分なスペースがない
93	ユーザーは既に存在します
94	ターゲットコレクション、クラスター、またはリソースの権限が無効または使用できません。
95	データベースが存在しません
96	'writeConcern' パラメータが無効です
97	パスワードを空にすることはできません
99	未定義のロールまたは権限

エラーコード	Description
102	既に組み込まれているロールを追加することはできません
103	コレクションから有効なロールを抽出できませんでした。
105	ユーザーが無効であるか、認可ライブラリを初期化できませんでした。
107	データベースに他のデータベースからロールを付与することはできません
111	更新ドキュメントを提供するときは、_id を正確に指定する必要があります
112	書き込み競合
113	指定されたロールは存在しません。
114	ロールが既に存在します
115	anyResource は、指定時に true である必要があります
116	リソースには、db とコレクションセット、クラスターセット、またはanyResourceセットが必要です
117	オペレーションは、アクティブなトランザクションなど、進行中の別のオペレーションと競合します。
120	すべてのユーザーを表示しているため、「showPrivileges」を指定できません
121	ドキュメントの検証に失敗した
122	更新エラー
124	スキーマ ID の処理中にエラーが発生しました
125	コマンドは失敗しました。多くの場合、参照があいまいであるか、セッションがないためです。
126	データベースは既に存在します
127	データベース名は、異なるケース名に既に存在します
128	フィールドが多すぎます

エラーコード	Description
129	データベースの削除を待っています
130	不正なパラメータ値
136	カーソルの位置が失われたか、必要な変更ストリーム機能が有効になっていません。
139	JS インタープリタの失敗
145	クエリで定義されていないエイリアス
147	マスターではない
148	クエリを実行するためのリソースが不十分
149	クエリで既に定義されているエイリアス
154	引数に無効な値または誤った (数値以外の) 型があります。
155	NULL 値が発生しました
156	UNDEFINED 値が発生しました
157	NAN 値が発生しました
158	別の目的で既に使用されているコレクション
159	期間値がオーバーフローしたか、ネットワークオペレーションが失敗しました。
161	リクエストされた照合バージョンが無効または使用できません。
163	フィールドまたは名前がストレージに有効ではありません。
164	クエリ演算子は、許可されていない方法でネストされました。
165	ビューがネストの最大深度である 20 を超えるか、サイクル内でそれ自体を参照します。
166	オペレーションはビューでサポートされていないか、使用された位置要素が多すぎます。

エラーコード	Description
167	リクエストされたオペレーションまたはステージは、ビューではサポートされていません。
168	式またはパイプライン演算子が認識されないか、引数が無効です。
169	BadValue $maxTimeMS には非整数値があります
170	BadValue $maxTimeMS が範囲外です
171	インデックスを複数の配列フィールド上に構築することはできません。
173	スキップ値は正である必要があります
174	予想されるログの詳細度は数値です
181	キーパターンは複数のインデックスに一致します。代わりに名前でインデックスを指定します。
197	インデックス仕様に無効なフィールドまたはサポートされていないフィールドが含まれています。
207	uuid は UUID (4) サブタイプの 16 バイトのバイナリフィールドである必要があります
224	このクエリ機能は、現在のコンテキストでは許可されていません。
225	新しいトランザクションがセッションで開始されたため、この古いトランザクションは続行できません。
241	値を $convert でリクエストされたタイプに変換できませんでした。
250	無効なビュー
251	指定されたトランザクション番号がアクティブなトランザクションと一致しません。
252	再帰制限に達しました
253	ビューにエントリを挿入できませんでした

エラーコード	Description
254	ビューからエントリを削除できませんでした
256	$having 句でフィルタリングされたドキュメント
257	トランザクション内のすべてのオペレーションの合計サイズが、許可された制限を超えています。
263	このオペレーションは、マルチドキュメントトランザクション内で実行することはできません。
291	$geoNear クエリのインデックスが見つかりません
300	SASL ハンドシェイクまたはセッションエラーのため、認証に失敗しました。
301	認証メカニズムまたはリクエストされたオペレーションはサポートされていません。
302	SASL 交換を完了するには、認証に追加のデータが必要です。
303	リクエストされたステージ、オプション、またはロールの設定はサポートされていません。
304	リクエストされた集約ステージはサポートされていません。
305	集計射影演算子はサポートされていません。
306	指定された集計グループアキュムレータ演算子はサポートされていません。
502	列は既に存在します
505	ここで挿入は許可されません
601	座標が範囲外です
602	インデックスエラー
700	このパラメータへの変更は、ライブモードでは許可されません
701	このパラメータの値が無効です

エラーコード	Description
703	このパラメータの不正な状態遷移
999	不正なトランザクション ID
4570	文字列値を期待する $lookup パラメータ
4572	$lookup に必須フィールドがありません
10065	オブジェクト (vars や $rand など) にする必要があるパラメータに、オブジェクト以外の値が渡されました。
10107	マスターではない
11000	重複キーは一意のインデックス制約に違反します。
11601	オペレーションが中断されました。
11603	クライアントリクエストの破損または不正な構文
13103	複合キーが多すぎる
13297	同じ名前のデータベースは既に存在しますが、大文字と小文字のみが異なります。
13388	認識されないパイプラインステージ名: $collStats
13435	master および slaveOk=false ではない
14031	十分なディスク容量がない
15888	は、作成するコレクションの名前を渡す必要があります
15947	グループのフィールドは オブジェクトで指定する必要があります
15950	グループ集計フィールド名を演算子にすることはできません
15952	不明な $group 演算子
15955	グループ集計演算子の構文エラー: _id フィールドを指定する必要があります

エラーコード	Description
15956	$skip 引数は負です。負以外の数値である必要があります。
15957	制限は数値として指定する必要があります
15958	$limit 値は正の数である必要があります。
15959	一致フィルターは オブジェクト内の式である必要があります
15969	構文エラー: $project 演算子の値はドキュメントである必要があります。
15972	$skip への引数は数値である必要があります
15973	$sort キー仕様は オブジェクトである必要があります
15974	$sort キーの順序は数字を使用して指定する必要があります
15975	$sort キーの順序は 1 (昇順の場合) または -1 (降順の場合) にする必要があります
15976	$sort ステージには少なくとも 1 つのソートキーが必要です
15981	クエリフィールドは文字列ではありません
15983	式を表すオブジェクトには、1 つのフィールドのみが必要です。
15992	オブジェクト式で許可されていないフィールドタイプ NULL
15998	フィールドパス名は空の文字列にすることはできません。
16006	値は有効な日付タイプではないため、日付に変換できませんでした。
16007	文字列が予期されたため、値を文字列に変換することはできません。
16020	式演算子が間違った数の引数で呼び出されました。
16034	$substrBytes 開始インデックスは数値型である必要があります。
16035	$substrBytes length 引数は数値型である必要があります。
16395	$match 集計式内では $where は許可されません

エラーコード	Description
16400	はフィールドの式を追加できません。これは、そのフィールドまたはそのサブフィールドの 1 つにすでに式があるためです。
16401	フィールド全体に適用される式が既に存在するため、 はサブフィールドの式を追加できません
16404	このコンテキストでは $ 式は使用できません
16406	_id 以外のフィールドを除外することはできません
16410	$project フィールド名またはパスが不正に で始まるか、'$' が含まれています。
16411	フィールド名に null 文字 (複数可) を含めることはできません
16419	フィールドパスに埋め込み null 文字を含めることはできません
16424	$match 集計式内では $near は許可されません
16426	$nearSphere/$near は $match 集計式内では許可されません
16435	パイプラインコンポーネントには 1 つの要素が含まれている必要があります
16493	16MB を超える文字列を作成しようとしました
16554	$add は数値型または日付型のみをサポートしています。
16605	$geoNear には有効な「near」フィールドと正しく入力された geoNear オプションが必要です。
16606	$geoNear には文字列としてdistanceField」オプションが必要です
16607	$geoNear では、指定されたオプションを文字列として指定する必要があります。
16608	ゼロで割る
16610	$mod には 0 の除数が割り当てられました。
16611	$mod は両方のオペランドの数値型のみをサポートします。

エラーコード	Description
16612	$add 式で許可される日付は 1 つだけです
16613	オペレーションは、日付から数値以外の日付値を減算しようとしました。
16702	$concat は文字列引数のみをサポートします。
16730	textIndexVersion が範囲外です
16755	は厳密な巻き付け順序でジオメトリのインデックスを作成できません
16800	には 2 つの地理フィールドを含めることはできません
16801	2d はインデックスの先頭にある必要があります
16804	location オブジェクトが想定されています。location 配列の形式が正しくありません
16810	$mod には、パラメータとして厳密に 2 つの値の配列が必要です。
16837	更新パスは、同じフィールドを複数回変更することはできません。
16868	無効な変数名
16869	無効な変数名
16870	無効な変数名
16872	フィールドパスが空であるか、$project のフィールドパスとして「$」のみが使用されます。
16873	フィールドパスが $ で始まらない
16874	$let は引数として オブジェクトのみをサポートします
16875	$let には認識されないパラメータが与えられました。
16876	$let への「vars」パラメータがありません
16877	$let への「in」パラメータがありません

エラーコード	Description
16878	$map は引数としてオブジェクトのみをサポートします
16879	$map には認識されないパラメータが与えられます。
16880	$map に必要な「入力」パラメータがありません。
16881	必須フィールドがありません
16882	$map への「in」パラメータがありません
16883	$map への入力は配列である必要があります。
16898	$minDistance は負でない必要があります
16899	$maxDistance は数値である必要があります
16900	$maxDistance は負でない必要があります
16945	$group または $lookup のメモリ制限を超えました。allowDiskUse:true をオプトインに渡します。
16949	集計オプションが無効です。allowDiskUse はブール値である必要があります。
16990	$out には、出力ターゲットに名前を付ける文字列またはオブジェクト引数が必要です。
16996	$out の挿入に失敗しました
16997	ターゲットコレクションのインデックスが処理中に変更されたため、$out に失敗しました。
17040	$allElementsTrue の引数は配列である必要があります。
17041	$anyElementTrue の引数は配列である必要があります。
17042	$setIsSubset の両方のオペランドは配列である必要があります。
17043	$setUnion のすべてのオペランドは配列である必要があります。
17044	$setEquals のオペランドは配列ではありません。

エラーコード	Description
17045	$setEquals には少なくとも 2 つの引数が必要です。
17046	$setIsSubset の両方のオペランドは配列である必要があります。
17047	$setIntersection のすべてのオペランドは配列である必要があります。
17048	$setDifference の両方のオペランドは配列である必要があります。
17049	$setDifference の両方のオペランドは配列である必要があります。
17053	$redact の式は、変数 $$KEEP、$$DESCEND、$$PRUNE 以外に何も返さないでください。
17080	cond に「if」パラメータがありません
17081	cond に「then」パラメータがありません
17082	cond に「else」パラメータがありません
17083	$cond への認識されないパラメータ
17124	$size の引数は配列である必要があります。
17275	ルートを再定義できない
17276	変数が見つかりません
17280	インデックスキーの値が大きすぎてインデックスを作成できません。
17287	$merge は、同じターゲット行を複数回変更することはできません。
17293	テキストインデックスオプション「textIndexVersion」は数値である必要があります
17302	この引数は 0 以上である必要があります。
17304	$geoNear には、有効な「near」ポイントおよびジオクエリ引数が必要です。
17308	$meta にはサポートされていない引数が与えられます。

エラーコード	Description
17310	$setIsSubset の両方のオペランドは配列である必要があります。
17313	$text を含む $match は、最初のパイプラインステージとしてのみ許可されます。
17320	createCollection は、予約名を含む名前空間で実行できません。
17385	特別なコレクションに $out できません
17390	$group は包含形式の式をサポートしていません
17419	更新後の結果のドキュメントが最大許容サイズを超えています。
17420	upsert へのドキュメントが 16777216 より大きい
18533	形式フィールドは文字列である必要があります
18534	$dateToString への認識されないフィールド
18535	末尾のパーセント記号のため、日付形式の文字列が無効です。
18536	無効な $dateToString 時間形式
18537	$dateToString は、0～9999 の範囲外の日付コンポーネントをフォーマットすることはできません。
18627	$dateToString への「format」パラメータがありません
18628	$dateToString への「date」パラメータがありません
18629	引数タイプが正しくありません
25278	$geoNear にはdistanceField」引数が必要です
26823	使用可能なビット内でオペレーション ID を表すことはできません。
28539	個別のコマンドの最初の引数は文字列である必要があります
28646	$Filter はオブジェクトをパラメータとして想定します

エラーコード	Description
28647	$filter への予期しないフィールド
28648	$Filter には「入力」フィールドが必要です
28649	$Filter には「as」フィールドが必要です
28650	$Filter には「cond」フィールドが必要です
28651	配列入力が必要
28656	$substrBytes: 範囲が無効です。開始インデックスは UTF-8 継続バイトです。
28657	$substrBytes: 範囲が無効です。終了インデックスは UTF-8 文字の中間にあります。
28664	$concatArrays は配列入力のみをサポートします。
28667	$slice、$range、$indexOfArray などの式が間違った数の引数を取得しました。
28680	$abs は、最小長など、否定できない値に適用されます。
28689	演算子引数は配列である必要がありますが、別の型が指定されています。
28690	arrayElemAt の 2 番目の引数は数値である必要があります
28691	arrayElemAt への 2 番目の引数は 32 ビット整数で表す必要があります
28714	$sqrt の引数は 0 以上である必要があります
28724	$slice の最初の引数は配列である必要があります。
28725	$slice の 2 番目の引数は数値である必要があります。
28726	$slice の 2 番目の引数は 32 ビット整数に収まる必要があります。
28727	$slice の 3 番目の引数は数値である必要があります。
28728	$slice の 3 番目の引数は 32 ビット整数である必要があります。
28729	$slice の 3 番目の引数は、正の 32 ビット整数である必要があります。

エラーコード	Description
28745	$sample ステージ仕様は オブジェクトである必要があります
28746	size 引数から $sample への値は数値である必要があります
28747	size 引数を $sample に負にすることはできません
28748	$sample は認識されないオプションを受け取りました。
28749	$sample ステージはサイズを指定する必要があります
28756	$log の引数は数値である必要があります。
28757	$log では、その基本引数が数値である必要があります。
28758	$log の引数は正の数である必要があります。
28759	$log のベースは、1 に等しくない正の数である必要があります。
28761	$log10 の引数は正の数である必要があります。
28762	$pow のベースは数値である必要があります。
28763	$pow の指数は数値である必要があります。
28764	$pow は 0 のベースと負の指数を取ることはできません
28765	$ceil、$exp、$floor、$ln、または $log10 は、数値以外の型を受け取ります。
28766	$ln には正の数値引数が必要です。
28812	フィールドパスが空です
28818	フィールドパスが「$」で始まらない
31022	この演算子では、「入力」パラメータを指定する必要があります。
31023	この演算子には「regex」パラメータが必要です。
31024	演算子には不明な引数が与えられます。

エラーコード	Description
31032	フィールド名に NULL バイトを含めることはできません。
31034	値が [-32768, 32767] の許容範囲外です。
31095	isoWeekYear」は 1～9999 の整数に評価する必要があります。
31138	ソートに使用される $meta 値は、有効なメタデータキーワードではありません。
31170	空のオブジェクトが予期されましたが、空でない値が指定されました。
31249	射影パスが射影内の別のフィールドと衝突します。
31254	除外射影内で包含を指定することはできません。
31393	$bsonSize では、入力としてドキュメントが必要です。
31394	位置射影演算子は、フィールドパスの中央には表示できません。
34413	無効な演算子がジオニアクエリで使用されました。
34435	$reverseArray の引数は配列である必要があります。
34443	$range には数値の開始値が必要です。
34444	$range には、32 ビット整数として表される開始値が必要です。
34445	$range では、終了値が数値である必要があります。
34446	$range 終了値は 32 ビット整数に収まる必要があります。
34447	$range には数値のステップ値が必要です。
34448	$range には、32 ビット整数として表現できるステップ値が必要です。
34449	$range にはゼロ以外のステップ値が必要です
34450	$substrCP の開始インデックスは数値型である必要があります。
34451	$substrCP 開始インデックスは 32 ビット整数として表すことはできません。

エラーコード	Description
34452	$substrCP への length 引数は数値型ではありません。
34453	$substrCP の長さは 32 ビット整数として表すことはできません。
34454	$substrCP: 長さは負以外の整数である必要があります。
34455	$substrCP: 開始インデックスは負以外の整数である必要があります。
34460	$zip は引数として オブジェクトのみをサポートします。
34461	$zip 入力は式の配列である必要があります。
34462	「defaults」オプションは式の配列である必要があります。
34463	$zip の useLongestLength オプションはブール値である必要があります。
34464	$zip に不明な引数が割り当てられました。
34465	$zip には少なくとも 1 つの入力配列が必要です
34466	useLongestLength が true でない限り、 はデフォルトを指定できません
34467	のデフォルトと入力の長さは同じである必要があります
34468	$zip は入力で非配列式を受け取ります。
34471	$strLenCP には文字列引数が必要です。
34473	$strLenBytes には文字列引数が必要です。
40060	$switch では、引数として オブジェクトが必要です。
40061	$switch では、'branches' を配列にする必要があります。
40062	$switch では、各ブランチが オブジェクトである必要があります。
40063	$switch には、ブランチ内で不明な引数が指定されます。
40064	$switch では、各ブランチに「case」式が必要です。

エラーコード	Description
40065	$switch では、各ブランチに「then」式が必要です。
40066	$switch は入力に一致するブランチを見つけることができず、デフォルトは指定されませんでした。
40067	$switch 式に不明な引数が含まれています。
40068	$switch には少なくとも 1 つのブランチが必要です
40075	$reduce には オブジェクトが必要です
40076	$reduce への不明な引数
40077	$reduce には「入力」が必要です
40078	$reduce にはinitialValue」が必要です
40079	$reduce には「in」が必要です
40080	$reduce への入力は配列である必要があります。
40081	演算子には、2 番目の引数として 配列が必要です。
40085	$split では、文字列を評価するために最初の引数が必要です。
40086	$split の 2 番目の引数は文字列に評価する必要があります。
40087	$split には空でない区切り文字が必要です
40090	$indexOfArray には最初の引数として配列が必要です
40091	$indexOfBytes の最初の引数は文字列ではありません。
40092	$indexOfBytes では、2 番目の引数として文字列が必要です。
40093	$indexOfCP には、最初の引数として文字列が必要です。
40094	$indexOfCP には、2 番目の引数として文字列が必要です。
40096	$indexOfArray には、開始インデックス値と終了インデックス値が必要です。

エラーコード	Description
40097	$indexOfArray には、負以外の数値の開始インデックスと終了インデックスが必要です。
40148	sortByCount フィールドは、$ プレフィックス付きパスまたはオブジェクト内の式として定義する必要があります
40149	sortByCount フィールドは文字列またはオブジェクトとして指定する必要があります
40156	count フィールドは空でない文字列である必要があります
40157	count フィールドは空でない文字列である必要があります
40158	カウントフィールドを $ プレフィックス付きパスにすることはできません
40159	カウントフィールドに null バイトを含めることはできません
40160	カウントフィールドに「」を含めることはできません。
40169	$facet 仕様は空でないオブジェクトである必要があります
40170	$facet への引数は配列である必要があります
40171	$facet 仕様の配列の要素は空でないオブジェクトである必要があります
40172	$facet ステージのサブパイプラインを空にすることはできません
40176	指定された競合するフィールド名
40177	$project と $addFields には、少なくとも 1 つの出力フィールドが必要です。
40178	包含射影で「_id」以外のフィールドを除外することはできません
40179	除外射影にフィールドを含めたり、フィールドをリセットしたり、計算されたフィールドを追加したりすることはできません
40180	$project では、空のオブジェクトをフィールド値として許可しません。

エラーコード	Description
40181	:: によって引き起こされる無効な $project :: 式指定には 1 つのフィールドのみが含まれている必要があります。
40183	ドットフィールド名はサブオブジェクト内では使用できません。
40192	$bucket の「境界」フィールドには少なくとも 2 つの値が必要です
40193	$bucket の「境界」オプション内のすべての値は同じ型である必要があります
40194	$bucket への「境界」オプションはソートする必要がありますが、要素は昇順ではありません
40195	$bucket の「デフォルト」フィールドは定数式である必要があります
40196	$bucket の「出力」フィールドは オブジェクトである必要があります
40197	$bucket に対する認識されないオプション
40198	$bucket にはgroupBy」と「boundaries」を指定する必要があります
40199	$bucket の「デフォルト」フィールドは、最低境界より小さいか、最高境界以上である必要があります。
40200	$bucket の「境界」フィールドは配列である必要があります
40201	$bucket ステージへの引数は オブジェクトである必要があります
40202	$bucket 'groupBy' フィールドは、$ プレフィックス付きパスまたは式として定義する必要があります
40203	フィールドはアキュムレータオブジェクトである必要があります
40204	フィールド名に「」を含めることはできません。
40205	フィールド名を演算子名にすることはできません
40206	アキュムレータは単項演算子です
40218	クエリにはテキストスコアメタデータが必要ですが、使用できません

エラーコード	Description
40228	$replaceRoot 'newRoot' 式は オブジェクトに対して評価する必要があります。
40229	$replaceRoot には、仕様として オブジェクトが必要です。
40230	$replaceRoot はnewRoot」オプションのみを受け入れます。
40234	$group フィールドはアキュムレータオブジェクトである必要があります。
40235	フィールド名に「.」文字を含めることはできません。
40236	フィールド名は「$」で始まることはできません
40237	このアキュムレータは単項演算子であり、単一の引数を取ります。
40239	$bucketAuto 'groupBy' フィールドは、$ プレフィックス付きパスまたは式オブジェクトとして定義する必要があります
40240	$bucketAuto への引数は オブジェクトである必要があります
40241	$bucketAuto 'buckets' フィールドは数値である必要があります
40242	$bucketAuto 'buckets' フィールドは 32 ビット整数として表現できる必要があります
40243	$bucketAuto 'buckets' フィールドは 0 より大きい必要があります
40244	$bucketAuto 'output' フィールドは オブジェクトである必要があります
40245	$bucketAuto に対する認識されないオプション
40246	$bucketAuto にはgroupBy」と「buckets」を指定する必要があります
40257	未知の四捨五入の詳細度
40258	$bucketAuto は数値境界のみで「粒度」を指定できます
40260	$bucketAuto は負以外の数値のみを使用して「粒度」を指定できますが、負の数値が見つかりました
40261	$bucketAuto の「粒度」フィールドは文字列である必要があります

エラーコード	Description
40272	$addFields ステージ仕様は オブジェクトである必要があります。
40319	$lookup ステージ仕様は オブジェクトである必要があります
40320	「from」オプションから $lookup ステージ仕様への欠落
40321	'from' オプションから $lookup は文字列である必要があります
40322	無効な $lookup 名前空間
40323	パイプラインステージ仕様オブジェクトには、1 つのフィールドのみを含める必要があります。
40324	パイプラインには、認識されないステージ名が含まれています。
40331	この集約ステージは、$facet パイプライン内では使用できません。
40352	$project フィールドパスを空の文字列にすることはできません。
40353	フィールドパスは「.」で終わることはできません。
40386	$arrayToObject には配列入力が必要です。
40390	$objectToArray では、入力としてドキュメントが必要です。
40391	$arrayToObject には一貫した入力形式が必要です。要素はすべて配列であるか、すべてオブジェクトである必要があります。オブジェクトが検出されましたが、見つかりました: 配列
40392	$arrayToObject には、キーが「k」と「v」のオブジェクトが必要です。
40393	$arrayToObject には、キーが「k」と「v」のオブジェクトが必要です。どちらかまたは両方のキーがない
40394	$arrayToObject には、文字列「k」キーと「v」値を持つオブジェクトが必要です。
40395	$arrayToObject には、キーが文字列であるキーと値のペアが必要です。

エラーコード	Description
40396	$arrayToObject には一貫した入力形式が必要です。要素はすべて配列であるか、すべてオブジェクトである必要があります。配列が検出され、検出されました: オブジェクト
40397	$arrayToObject には、2 要素配列の配列が必要です。
40398	$arrayToObject には、認識されないタイプまたは形式の入力が与えられます。
40400	$mergeObjects では、すべての入力がオブジェクトである必要があります。
40414	必須のコマンドフィールドがないか、検証に失敗しました。
40415	コマンドには、認識されないフィールドが含まれています。
40422	BSON 配列フィールド名の値が無効です。
40485	認識されないタイムゾーン識別子が指定されました。
40489	$dateFromParts では、通常のカレンダーフィールドと ISO 週の日付フィールドを混在させることはできません
40515	$dateFromParts 引数は整数には評価されません。
40516	$dateFromParts には「year」またはisoWeekYear」が必要です
40517	「タイムゾーン」オプションは、日付演算子の文字列に評価する必要があります。
40518	$dateFromParts または $dateToParts が認識されない、または誤って入力されたパラメータを取得しました。
40519	$dateFromParts は引数として オブジェクトのみをサポートします。
40520	$dateToParts に認識されないパラメータが割り当てられました。
40522	$dateToParts には、「date」パラメータを持つオブジェクト引数が必要です。
40523	$dateFromParts 'year' は 1～9999 の範囲の整数である必要があります。

エラーコード	Description
40525	$dateFromParts ISO 週/日値が範囲外であるか、必須フィールドがありません。
40533	タイムゾーン引数は文字列である必要があります。
40535	認識されないオプションがステージまたは式に指定されました。
40536	式は、配列を指定するときに 1 つの引数のみを受け入れます。
40539	日付演算子に必要な「date」引数がありません。
40540	$dateFromString は引数としてオブジェクトのみをサポートします
40541	$dateFromString は認識されない引数を受け取りました。
40542	$dateString への「dateString」パラメータがありません dateFromString
40543	$dateFromString では、'dateString' を文字列にする必要があります。
40545	日付/時刻文字列が不完全で、必須要素がありません。
40551	タイムゾーン情報を含む日付/時刻文字列をタイムゾーン引数と一緒に渡すことはできません
40553	$dateFromString は指定された日付文字列を解析できません。
40554	GMT オフセットの日付/時刻文字列をタイムゾーン引数と一緒に渡すことはできません
40575	$changeStream 'fullDocument' は「デフォルト」またはupdateLookup」である必要があります。
40600	$out は $facet ステージ内では使用できません。
40601	$out と $merge は、パイプラインの最終ステージにすぎません。
40602	$changeStream や $geoNear などのステージは、パイプラインの最初のステージではありません。

エラーコード	Description
40603	$geoNear は、最適化後のパイプラインの最初のステージではありませんでした。最適化は無効または禁止されていますか?
40647	変更ストリームの再開トークンに _data フィールドがないか、タイプが正しくありません。
40674	1 つのタイプの再開オプションのみが許可されますが、複数の が見つかりました。
50001	autoIndexId:false は、指定されたコレクションでは許可されません。
50694	この演算子には不明な引数が割り当てられました。
50695	この演算子には「入力」フィールドが必要です。
50696	演算子は、引数として オブジェクトのみをサポートします。
50699	この式演算子では、その入力を文字列にする必要があります。
50700	'chars' 引数は文字列である必要があります。
50736	getMore は、セッションで作成されていないカーソルのセッションで実行できません。
50737	セッション作成カーソルで getMore を実行するには、lsid を指定する必要があります。
50752	$substrBytes 開始インデックスが負です。
50808	$changeStream ステージでは、引数としてドキュメントが必要です。
51024	照合の「強度」値は 5 以下である必要があります。
51047	この演算子またはステージは、$lookup サブパイプライン内では使用できません。
51075	$regex と $options の両方でオプションを設定できない
51076	指定された数のリストのベクトルインデックスを構築するメモリが不十分です。

エラーコード	Description
51078	インデックス名が見つかりませんでした。
51079	レイテンシー統計はサポートされていません
51091	正規表現が無効または正しく形成されていません。
51103	演算子は、名前付き引数のオブジェクトを想定していますが、別のタイプを受信しました。
51104	演算子では、'input' 引数を文字列にする必要があります。
51105	演算子の「regex」引数は、文字列または正規表現タイプではありません。
51106	演算子では、「options」パラメータを文字列にする必要があります。
51107	$regex オプションは、「regex」フィールドと「options」フィールドの両方で指定することはできません。
51108	正規表現オプションに無効なフラグまたは埋め込み NULL バイトが含まれています。
51109	正規表現に NULL バイトを含めることはできません。
51110	正規表現オプションに NULL バイトを含めることはできません。
51111	この演算子に指定された正規表現が無効です。
51151	出力を保存するバッファのサイズがデータ型の制限を超えました
51156	評価中に正規表現が $regex で実行できませんでした。
51270	空のサブ射影オブジェクトは有効な射影値ではありません。
51272	構文エラー: $projection には少なくとも 1 つの出力フィールドが必要です。
51744	演算子では、文字列に「置換」が必要です。
51745	演算子では、'find' 引数を文字列にする必要があります。
51746	演算子の「入力」引数は文字列ではありません。

エラーコード	Description
51747	演算子には「置換」パラメータが必要です。
51748	この演算子には「検索」パラメータが必要です。
51749	この演算子には「入力」パラメータが必要です。
51750	集計演算子が不明な引数を受け取りました。
100003	最適化クエリパスの処理中の内部システムエラー
100005	コレクション、カスタムロール、権限などの設定された制限を超えました。
100006	許可されたユーザーの最大数を超えました。
3040501	$rand は現在引数を受け入れません
3041701	$getField は $project で不明な引数を受け取りました。
3041702	無効な $project :: によって引き起こされる :: $getField には 'field' を指定する必要があります
3041703	無効な $project :: によって引き起こされる :: $getField には「入力」を指定する必要があります
3041704	$getField では、文字列の評価に「field」引数が必要です。
4161100	$setField と $unsetField は、引数として オブジェクトのみをサポートします。
4161101	$setField または $unsetField に不明な引数が割り当てられました。
4161102	$setField と $unsetField には「field」を指定する必要があります。
4161103	無効な $project :: によって引き起こされる :: $setField には 'value' を指定する必要があります
4161105	$setField または $unsetField では、オブジェクトの評価に「入力」が必要です。
4161106	$setField と $unsetField では、定数に評価するために「field」が必要です。
4161107	$setField または $unsetField では、文字列の評価に「field」が必要です。

エラーコード	Description
4161108	$project はここでフィールドパス参照を禁止します。代わりに {$literal: ...} を使用します。
4161109	$setField と $unsetField には、「入力」フィールドを指定する必要があります。
4662500	コマンドには、認識されないフィールドが含まれています。
4862100	完全修飾名前空間が最大許容長を超えています。
5166301	$dateDiff は引数としてオブジェクトのみをサポートします
5166302	$dateDiff に認識されない引数が割り当てられました。
5166303	$dateDiff へのstartDate」パラメータがありません
5166304	$dateDiff へのendDate」パラメータがありません
5166305	$dateDiff への「単位」パラメータがありません
5166307	$dateDiff 引数は日付ではありません。
5166308	日付の差の計算中に $dateDiff がオーバーフローしました。
5166400	$dateAdd と $dateSubtract では、引数が オブジェクトである必要があります。
5166401	$dateAdd または $dateSubtract が認識されない引数を取得しました。
5166402	$dateAdd または $dateSubtract には、startDate、単位、および金額が必要です。
5166403	$dateAdd ではstartDate を日付に変換できる必要があります
5166404	$dateAdd は時間単位を定義する文字列を想定しています
5166405	$dateAdd は整数の時間単位を想定しています
5166406	dateAdd オーバーフロー
5166503	不明な時間単位値が指定されました。

エラーコード	Description
5439002	切り捨てられた日付の計算中に $dateTrunc がオーバーフローしました。
5439006	dateTrunc サポートされていない binSize 値
5439007	$dateTrunc は引数として オブジェクトのみをサポートします
5439008	認識されない $dateTrunc 引数、予想日付、単位、binSize、タイムゾーン、startOfWeek。
5439009	$dateTrunc への「date」パラメータがありません
5439010	$dateTrunc への「単位」パラメータがありません
5439012	$dateTrunc では、'date' を日付にする必要があります。
5439013	$dateDiff および $dateTrunc では、「unit」を文字列にする必要があります。
5439014	$dateDiff および $dateTrunc の「単位」値は認識された時間単位ではありません。
5439015	$dateDiff または $dateTrunc では、文字列としてstartOfWeek」が必要です。
5439016	$dateDiff または $dateTrunc のstartOfWeek」値が有効な曜日ではありません。
5439017	$dateTrunc では、'binSize' を 64 ビット整数にする必要があります。
5439018	$dateTrunc では、'binSize' が 0 より大きい必要があります。
5447000	$collStats ステージはネストされたオブジェクトである必要があります。
5626500	$geoNear、$near、$nearSphere はこのコンテキストでは許可されません。これらの演算子では地理空間データをソートする必要があるためです。ソートが必要ない場合は、代わりに $geoWithin の使用を検討してください。
5654600	$project では、このコンテキストでのフィールドパス参照は許可されません。
5654601	無効な $project :: :: $getField は定数に評価するために 'field' を必要としますが、定数以外の引数を取得しました

エラーコード	Description
5654602	$getField では、文字列の評価に「field」が必要です。
5733415	は $mod をゼロにすることはできません
5860400	$geoNear には「near」引数が必要です
5887502	$setEquals のすべてのオペランドは配列である必要があります。
5911200	集計中の PlanExecutor エラー:: :: $mergeObjects は オブジェクトのみをサポートします
5976500	$dateAdd に無効な「amount」パラメータ値が割り当てられました。
6045000	$dateSubtract に無効な「amount」パラメータ値が割り当てられました。
7158303	$switch では、入力に一致するブランチが見つからず、デフォルトも指定されませんでした。

リリースノート

これらのリリースノートでは、メジャーエンジンバージョン別に整理された Amazon DocumentDB の機能、改善点、バグ修正について説明します。各バージョンセクションでは、エントリは日付ごとにテーブル形式で一覧表示されます。

エンジンバージョンとパッチバージョンの番号付け方法 (エンジンパッチバージョンのプレフィックスと Amazon DocumentDB エンジンバージョンのマッピングを含む) の詳細については、「」を参照してくださいエンジンバージョンの番号付け。

Amazon DocumentDB エンジンパッチの最新バージョンは、次のコマンドを実行して確認できます。

```
db.runCommand({getEngineVersion: 1})
```

> **Note**
>
> Amazon DocumentDB 5.0.2 および 8.0.2 以降、このコマンドはエンジンバージョンとエンジンパッチバージョンを の形式で単一の文字列として返します*engine-version*+*engine-patch-version*。例えば、Amazon DocumentDB 5.0.2 のクラスターは を返し`5.0.2+3.0.12345`、8.0.2 のクラスターは を返します`8.0.2+4.0.12345`。以前のバージョンのクラスターは、 などのエンジンパッチバージョンを単独で返します`3.0.12345`。詳細については、「エンジンバージョンの番号付け」を参照してください。

クラスターがエンジンの最新バージョンでない場合は、エンジンをアップグレードする保留中のメンテナンスがある可能性があります。詳細については、「 デベロッパーガイド」の「Amazon DocumentDB のメンテナンス」を参照してください。

トピック

- Amazon DocumentDB エンジンバージョンのリリースノート
- Amazon DocumentDB の一般的なリリースノート
- Amazon DocumentDB エラスティッククラスターのリリースノート

Amazon DocumentDB エンジンバージョンのリリースノート

メジャーバージョン別に整理された Amazon DocumentDB エンジンバージョンのリリースノート。

Amazon DocumentDB 8.0

Amazon DocumentDB エンジンバージョン 8.0 (MongoDB 8.0 互換) のリリースノート。

Amazon DocumentDB 8.0.2

日付	リリースノート
2026 年 9 月 28 日	Amazon DocumentDB 8.0.2 (エンジンバージョン: 4.0.12042) 新しい機能と改善点 1. プランナーバージョン 3 で$setWindowFields 集計ステージとそのウィンドウ演算子のサポートが追加されました。 2. プランナーバージョン 3 の$bucketAuto 集約ステージのサポートが追加されました。 3. プランナーバージョン 3 の$facet集約ステージのサポートが追加されました。 4. プランナーバージョン 3 の$graphLookup 集約ステージのサポートが追加されました。 5. 簡潔な相関$lookupサブクエリ構文を含む相関サブクエリのサポートを、プランナーバージョン 3 のパイプラインステージのサブセットに追加しました。 6. $changeStreamSplitLargeEvent 集約ステージのサポートを追加しました。 7. 変更ストリームは、 createCollection オペレーションのイベントを出力するようになりました。 8. 変更ストリームは、 createIndex オペレーションのイベントを出力するようになりました。 9. 変更ストリームの書き込みパフォーマンスが向上しました。 10.再試行可能な書き込みのサポートが追加されました。

日付	リリースノート
	11.部分インデックスに対する reIndex コマンドのサポートが追加されました。 12.プランナーバージョン 3 では$expr、述語のインデックススキャンがサポートされるようになりました。 13.プランナーバージョン 3 では、検索クエリにマルチフィールドインデックスのみのスキャン (IXONLYSCAN) が導入されています。詳細については、「インデックスのみのスキャン (対象クエリ)」を参照してください。 14.プランナーバージョン 3 では、 などの集約ステージにインデックスのみのスキャン (IXONLYSCAN) が導入されています$group。詳細については、「インデックスのみのスキャン (対象クエリ)」を参照してください。 15.プランナーバージョン 3 countDocuments() での $countおよび のパフォーマンス最適化。 16.プランナーバージョン 3 で増分ソートのサポートが追加されました。詳細については、「増分ソート」を参照してください。 17.テキストインデックスクエリのプラン選択が改善されました。 18統合ステージ$lookupとステージの説明プランに includeArrayIndex パラメータが反映される$unwindようになりました。 19.すべてのコマンドでインデックスフィルターのサポートが追加されました。 20.クエリプランナーは、等価 ($eq) 述語の一意のインデックスを優先するようになりました。 21.currentOp 出力の集計コマンドの切り捨て制限を引き上げました。 バグ修正 1. バージョン 1 (デフォルト) クエリプランナーのバグを修正しました。 2. メジャーバージョンアップグレード (MVU) 中のインデックス式テキストの往復処理の一部を修正しました。

日付	リリースノート
	3. 高速カウントクエリのプロファイラーの問題を修正しました。 4. `collStats` コマンドのセキュリティ修正。

Amazon DocumentDB 8.0.1

日付	リリースノート
2026 年 9 月 15 日	セキュリティと安定性の向上 Amazon DocumentDB 8.0.1 (エンジンバージョン: 4.0.11638) [進行中] Amazon DocumentDB は、次のメンテナンスウィンドウ中にこのパッチを自動的に適用します。
2026 年 8 月 31 日	新機能 Amazon DocumentDB は、エンジンバージョン 3.6 および 4.0 からバージョン 8.0 へのインプレースメジャーバージョンアップグレード (MVU) を直接サポートするようになりました。これにより、中間バージョンのアップグレードが不要になり、既存のデータ、設定、クラスター設定を維持しながら 3.6 または 4.0 クラスターをアップグレードできます。Amazon DocumentDB 3.6 および 4.0 から 8.0 へのメジャーバージョンアップグレードは、これらのバージョンが現在サポートされているすべての AWS リージョンで利用できます。詳細については、「Amazon DocumentDB のインプレースメジャーバージョンアップグレード」を参照してください。
2026 年 8 月 18 日	セキュリティと安定性の向上 Amazon DocumentDB 8.0.1 (エンジンバージョン: 4.0.9938) このエンジンパッチでは、次の点が改善されています。 • リクエスト処理を改善 Amazon DocumentDB は、接続パスとメッセージ解析パスに沿って受信クライアントリクエストをより徹底的

日付	リリースノート
	に検証するようになりました。これにより、幅広い入力パターンで耐障害性と安定性が向上します。 • 認証ハンドシェイクの強化 Amazon DocumentDB は、認証交換をより徹底的に検証するようになりました。これにより、予期しない入力や不正な形式の入力を処理する際の堅牢性が向上します。 Amazon DocumentDB は、次のメンテナンスウィンドウ中にこのパッチを自動的に適用します。
2026 年 8 月 11 日	新しい特徴 Amazon DocumentDB 8.0.1 (エンジンバージョン: 4.0.8864) • SCRAM-SHA-256 認証のサポートを有効にしました。 • Amazon DocumentDB グローバルクラスターの調整可能な RPO を有効にしました。 • バックグラウンドガベージコレクション中のストレージ再利用が改善され、時間の経過とともに未使用のストレージ領域が削減されました。
2026 年 7 月 22 日	新機能 エンジンバージョン 8.0 以降を実行している Amazon DocumentDB クラスターでは、最大ストレージ制限 256 TiB がサポートされるようになり、以前の制限の 128 TiB が 2 倍になりました。詳細については、「Amazon DocumentDB クォータ」を参照してください。

日付	リリースノート
2026 年 7 月 13 日	Amazon DocumentDB 8.0.1 (エンジンバージョン: 4.0.8864) 新しい機能と改善点 1. デフォルトのプランナー (v3) での 13 個のアキュムレータ演算子のサポート: $top、$bottom、$topN、、$bottomN、、$count$firstN、$lastN$max、$maxN、$minN、$minN、$stdDevPop 、。 $stdDevSample 2. デフォルトのプランナー (v3): $median、$percentile 、、$stdDevPop 、、$stdDevSamp 、$round$sigmoid、、、$trunc$firstN$lastN$maxN、、$minN$size 、、$bsonSize 、、、$tsIncrement 、$tsSecond 、、、$cos$acos$cosh$acosh、、$sin、、、$asin、、、、、、、、、、、、、、、、、、$sinh$asinh$tan$atan$tanh$atan2$atanh$degreesToRadians 、、、、$radiansToDegrees 、、、、、$isNumber 、、、、、、、、、、、、、、、、、、

日付	リリースノート
	バグ修正 1. アンダースコア文字を含む$text検索クエリが残差フィルターとして評価された場合に誤った結果を返す可能性がある問題を修正しました。

Amazon DocumentDB 8.0.0

日付	リリースノート
2026 年 9 月 15 日	セキュリティと安定性の向上 Amazon DocumentDB 8.0.0 (エンジンバージョン: 4.0.11440) [進行中] バグ修正 このリリースには、次のバグ修正が含まれています。 1. アンダースコア文字を含む$text検索クエリが残差フィルターとして評価された場合に誤った結果を返す可能性がある問題を修正しました。 Amazon DocumentDB は、次のメンテナンスウィンドウ中にこのパッチを自動的に適用します。
2026 年 8 月 18 日	セキュリティと安定性の向上 Amazon DocumentDB 8.0.0 (エンジンバージョン: 4.0.9912) このエンジンパッチでは、次の点が改善されています。 • リクエスト処理を改善 Amazon DocumentDB は、接続パスとメッセージ解析パスに沿って受信クライアントリクエストをより徹底的に検証するようになりました。これにより、幅広い入力パターンで耐障害性と安定性が向上します。

日付	リリースノート
	• 認証ハンドシェイクの強化 Amazon DocumentDB は、認証交換をより徹底的に検証するようになりました。これにより、予期しない入力や不正な形式の入力を処理する際の堅牢性が向上します。 より早く適用しない場合、Amazon DocumentDB はリリースから 30 日後に次のメンテナンスウィンドウ中にこのパッチを自動的に適用します。
2026 年 7 月 22 日	新機能 エンジンバージョン 8.0 以降を実行している Amazon DocumentDB クラスターでは、最大ストレージ制限 256 TiB がサポートされるようになり、以前の制限の 128 TiB が 2 倍になりました。詳細については、「Amazon DocumentDB クォータ」を参照してください。
2026 年 5 月 13 日	新機能 Amazon DocumentDB サーバーレスが Amazon DocumentDB 8.0 で利用可能になりました。詳細については、「Amazon DocumentDB Serverless の使用」を参照してください。
2026 年 5 月 6 日	バグ修正およびその他の変更 Amazon DocumentDB 8.0 (エンジンバージョン: 4.0.6940) この更新では、次の点が改善されたオプションのエンジンパッチが提供されます。 1. エッジケースでリーダーがクラッシュする可能性があるガベージコレクションのバグを修正しました。
2026 年 4 月 20 日	新機能 Amazon DocumentDB 5.0 から Amazon DocumentDB 8.0 へのインプレースメジャーバージョンアップグレード (MVU) が利用可能になりました。詳細については、「Amazon DocumentDB のインプレースメジャーバージョンアップグレード」を参照してください。

日付	リリースノート
2026 年 4 月 20 日	新しい機能と改善点 Amazon DocumentDB 8.0 (エンジンバージョン: 4.0.5995、4.0.6473) これは、以下の新機能、改善、修正を含む必須のエンジンパッチです。 1. $sortArray の 演算子$project: • プランナーバージョン 3 の集約パイプラインの$projectステージの$sortArray オペレータのサポート。$sortArray 演算子は、要素 (スカラー値の場合) または指定されたフィールド (ドキュメント配列の場合) に基づいて配列をソートし、ソートされた配列を昇順 (1) または降順 (-1) で返します。 2. プランナー v3 $geoNearを使用した outcollStats 、$indexStats 、、 のサポート。プランナー V3 では、$outステージはさまざまなデータベースのコレクションへの書き込みもサポートしています。 3. $unwind ステージに includeArrayIndex パラメータが含まれている場合でも、集約パイプラインが結合$lookupして$unwindステージを作成するようになりました。以前は、この最適化は が存在するときにスキップincludeArrayIndex されていました。 4. I/O 最適化設定で実行されている db.r8g.8xlarge 以上のインスタンスの書き込みスループットを向上させるパフォーマンスの強化。 5. インスタンスの Connections (active) メトリクスをモニタリングできるようになりました。詳細については、「インスタンスクォータ」を参照してください。 6. $dateToParts および$dateFromParts 集計演算子のサポート。 7. コマンドレベルのパフォーマンスメトリクスが CloudWatch findAndModify で CRUD オペレーション (find、insert、、update、、、delete、aggregate countdistinct、、commitTransaction 、) で利用可能になりgetMoreabortTransaction 、オペレーションモニタリング

日付	リリースノート
	を改善するための AvgDuration 、P100Duration 、、および MaxConcurrent ディメンションが公開されました。 8. / のブール値または true/false の数値1を含むnewRootフィールドが出力ドキュメントにないという$replaceRoot 動作のバグを修正0しました。 9. 有効なクエリ$indexOfArray が無効なことを示すエラーを返していた のバグを修正しました。 10説明プランのexecutionStats 出力にdocsExamined メトリクスが含まれるようになりました。 11SASL 認証のパフォーマンスが向上しました。 12.ドキュメントの更新を最適化して、インデックス値が変更されず、ドキュメントが既存のページに収まる場合のストレージオーバーヘッドを削減しました。 13bypassDocumentValidation での のサポート$merge。 すべての Amazon DocumentDB 8.0 クラスターは、次のメンテナンスウィンドウ中にこのパッチバージョンに自動的にアップグレードされます。
2026 年 3 月 26 日	新しい特徴 Amazon DocumentDB は、Arm ベースの AWS Graviton4 プロセッサを搭載した最新世代のメモリ最適化インスタンス db.r8g をサポートするようになり、R6G インスタンスと比較して最大 30% のパフォーマンスが向上しました。詳細については、このガイドインスタンスクラスの管理の「」を参照してください。

日付	リリースノート
2025 年 11 月 11 日	新しい特徴 Amazon DocumentDB は、MongoDB 8.0 とのフルワイヤプロトコル互換性を提供するようになりました。Amazon DocumentDB 8.0 はクエリパフォーマンスを向上させ、高性能アプリケーションを低コストで構築できるようにします。 Amazon DocumentDB 8.0 (エンジンパッチバージョン 4.0.4448) • MongoDB 8.0 API ドライバーのサポートが追加されました。 • 集約パイプラインの最適化と個別のコマンドのサポートに加えて、集約ステージ演算子のパフォーマンス向上を拡張するプランナー Version3 のサポートが追加されました。 • Zstandard 圧縮アルゴリズムによるディクショナリベースの圧縮のサポートが追加され、圧縮率が 5 倍向上しました。 • 新機能の追加: Amazon DocumentDB 8.0 は照合とビューをサポートしています。 • 新しい集計ステージと演算子のサポートが追加されました: $replaceWith、$vectorSearch、$merge、$set、$unset、$bucket、および 3 つの新しい集計演算子 $pow、$rand、$dateTrunc。 • 新しいバージョンのテキストインデックスのサポートを追加: Amazon DocumentDB 8.0 のテキストインデックス v2 では、追加のトークンが導入され、テキスト検索機能が強化されました。 • 並列ベクトルインデックス構築によるベクトル検索の改善により、インデックス構築時間が最大 30 倍短縮されました。 • TLS 1.0 および TLS 1.1 のサポートを削除しました。最小 TLS バージョンは TLS 1.2 になりました。

Amazon DocumentDB 5.0

Amazon DocumentDB エンジンバージョン 5.0 (MongoDB 5.0 互換) のリリースノート。

Amazon DocumentDB 5.0.2

日付	リリースノート
2026 年 9 月 28 日	Amazon DocumentDB 5.0.2 (エンジンバージョン: 3.0.23065) 新しい機能と改善点 1. すべてのインデックスタイプ (プランナーバージョン 2 以降) の非表示インデックスサポートを追加しました。 2. find コマンドでの `min()`および `max()`カーソルメソッドのサポート。 3. すべてのコマンドでインデックスフィルターのサポートが追加されました。 4. クエリプランナーは、等価 ($eq) 述語の一意のインデックスを優先するようになりました。 5. `currentOp` 出力の集計コマンドの切り捨て制限を引き上げました。 バグ修正 1. バージョン 1 (デフォルト) クエリプランナーのバグを修正しました。 2. メジャーバージョンアップグレード (MVU) 中のインデックス式テキストの部分的な往復処理を修正しました。 3. 高速カウントクエリのプロファイラーの問題を修正しました。

Amazon DocumentDB 5.0.1

日付	リリースノート
2026 年 8 月 18 日	セキュリティと安定性の向上 Amazon DocumentDB 5.0.1 (エンジンバージョン: 3.0.22319) [進行中]

日付	リリースノート
	このエンジンパッチは、次の改善点を提供します。 • リクエスト処理を改善 Amazon DocumentDB は、接続パスとメッセージ解析パスに沿って受信クライアントリクエストをより徹底的に検証するようになりました。これにより、幅広い入力パターンで耐障害性と安定性が向上します。 • 認証ハンドシェイクの強化 Amazon DocumentDB は、認証交換をより徹底的に検証するようになりました。これにより、予期しない入力や不正な形式の入力を処理する際の堅牢性が向上します。 より早く適用しない場合、Amazon DocumentDB はリリースから 30 日後に次のメンテナンスウィンドウ中にこのパッチを自動的に適用します。
2026 年 8 月 11 日	新機能 Amazon DocumentDB 5.0.1 (エンジンバージョン: 3.0.21138) SCRAM-SHA-256 認証のサポートを有効にしました。

日付	リリースノート
2026 年 6 月 8 日	新しいマイナーバージョン Amazon DocumentDB 5.0.1 (エンジンバージョン 3.0.21138) Amazon DocumentDB 5.0.1 は、Amazon DocumentDB 5.0 メジャーバージョンの最初のマイナーバージョンリリースです。マイナーバージョンには、新機能、セキュリティ修正、バグ修正などの改善が含まれています。マイナーバージョンアップグレードを使用して、5.0.0 から 5.0.1 にアップグレードできます。詳細については、「Amazon DocumentDB マイナーバージョンアップグレード」を参照してください。 このマイナーバージョンには、次の新機能と改善が含まれています。 1. ガベージコレクションプロセスの全体的なヘルスに詳細情報を表示collstats するためにgcRuntimeStats documentFragmentStats 、、、およびコレクションレベルのメトリクスの内訳を storageSegmentBase および storageSegmentExtended に追加しました。詳細については、「Amazon DocumentDB のガベージコレクション」を参照してください。 2. $rand および $pow 集計演算子のサポート。 3. インスタンスの Connections (active) メトリクスをモニタリングできるようになりました。詳細については、「インスタンスクォータ」を参照してください。 4. グローバル存続可能読み取り専用を使用してセカンダリクラスターの再起動を排除することで、グローバルクラスターのクロスリージョンレジリエンスを改善しました。 5. $dateToParts および$dateFromParts 集計演算子のサポート。 6. コマンドレベルのパフォーマンスメトリクスが CloudWatch findAndModify で CRUD オペレーション (find、insert、、update、、、delete、aggregate countdistinct、、commitTransaction 、) で利用可能になりgetMoreabortTransaction 、オペレーションモニタリング

日付	リリースノート
	を改善するための AvgDuration 、P100Duration 、、および MaxConcurrent ディメンションが公開されました。 7. $replaceRoot は、ブール値 (true/false) と数値 (0/) を含むnewRootフィールドを出力します1。 8. $indexOfArray は、より広範な有効なクエリ式をサポートしています。 9. explain プランには、executionStats 出力にdocsExamined メトリクスが含まれます。 10SASL 認証のパフォーマンスが向上しました。

Amazon DocumentDB 5.0.0

日付	リリースノート
2026 年 8 月 18 日	セキュリティと安定性の向上 Amazon DocumentDB 5.0.0 (エンジンバージョン: 3.0.22303) [進行中] このエンジンパッチは、次の改善点を提供します。 • リクエスト処理を改善 Amazon DocumentDB は、接続パスとメッセージ解析パスに沿って受信クライアントリクエストをより徹底的に検証するようになりました。これにより、幅広い入力パターンで耐障害性と安定性が向上します。 • 認証ハンドシェイクの強化 Amazon DocumentDB は、認証交換をより徹底的に検証するようになりました。これにより、予期しない入力や不正な形式の入力を処理する際の堅牢性が向上します。 • アクセスコントロールの適用 Amazon DocumentDB は、ネストされたクエリオペレーションに必要な権限をすべてのネストレベルで一貫して検証するようになりました。

日付	リリースノート
	より早く適用しない場合、Amazon DocumentDB はリリースから 30 日後に次のメンテナンスウィンドウ中にこのパッチを自動的に適用します。
2026 年 5 月 6 日	バグ修正およびその他の変更 Amazon DocumentDB 5.0 (エンジンバージョン: 3.0.20421) このリリースは、必要なセキュリティパッチ用です。
2026 年 4 月 20 日	新機能 Amazon DocumentDB 5.0 から Amazon DocumentDB 8.0 へのインプレースメジャーバージョンアップグレード (MVU) が利用可能になりました。詳細については、「Amazon DocumentDB のインプレースメジャーバージョンアップグレード」を参照してください。
2026 年 4 月 13 日	バグ修正およびその他の変更 Amazon DocumentDB 5.0 (エンジンバージョン: 3.0.19917) これは、以下の新機能、改善、修正を含む必須のエンジンパッチです。 1. DatabaseConnectionsActiveMax メトリクスがConnections (active) インスタンスを追跡するようになりました。詳細については、「インスタンスクォータ」を参照してください。 2. SASL 認証のパフォーマンスが向上しました。 すべての Amazon DocumentDB 5.0 クラスターは、次のメンテナンスウィンドウ中にこのパッチバージョンに自動的にアップグレードされます。

日付	リリースノート
2025 年 10 月 22 日	新しい特徴 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.17983) Amazon DocumentDB は、Arm ベースの AWS Graviton4 プロセッサを搭載した最新世代のメモリ最適化インスタンス db.r8g をサポートするようになり、R6G インスタンスと比較して最大 30% のパフォーマンスが向上しました。詳細については、このガイドの「インスタンスクラスの管理」および「ブログ投稿」を参照してください。 新しい特徴 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.15902) • より長いコレクション名 (名前空間の長さは最大 255 文字) のサポートが追加されました。 • 新しいクエリプランナー (プランナーバージョン 2.0) を導入しました。詳細については、このガイドの「クエリプランナー v2」を参照してください。 • ガベージコレクションと老朽化コレクションに関するコレクションの全体的な状態に関する情報を提供する「MVCCIDStats」と「gcRuntimeStats」の新しいフィールドを `collstats` に追加しました。
2025 年 10 月 16 日	新機能 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.15902) Amazon DocumentDB は、クラスターが仮想プライベートクラウド (VPC) にあるときにデュアルスタックモード (IPv4/IPv6) をサポートするようになりました。 詳細については、こちらの ブログ記事 と、「Amazon DocumentDB 開発者ガイド」の「Amazon VPC と Amazon DocumentDB」を参照してください。

日付	リリースノート
2025 年 9 月 15 日	バグ修正およびその他の変更 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.17983) クエリに後続の `$replaceRoot` および `$lookup` ステージが含まれている場合のエンジンクラッシュを修正しました。
2025 年 7 月 29 日	新しい特徴 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.15902) • `$regexFindAll` 集計演算子のサポートが追加されました。 • インデックスの肥大化とコレクションベースの肥大化メトリクスがリリースされました • ガベージコレクション と老朽化コレクションに関するクラスターの全体的な状態に関する情報を提供する CloudWatch メトリクス `AvailableMVCCIds` と `LongestActiveGCRuntime` を追加しました。 • `serverStatus` コマンドに「nvme_cache」フィールドが含まれるようになりました。このフィールドには、NVMe-backed インスタンスで NVMe キャッシュに書き込まれたページと書き込まれなかったページの数が表示されます。 • `$group` および `$project` のフィールド数の制限を削除しました • 次の文字列演算子のサポート: `$trim`、`$rtrim`、`$ltrim`、`$regexFindAll` 、`$replaceOne` 、および `$replaceAll` • 次の配列演算子のサポート: `$first` および `$last` • 次のステージ演算子のサポート: `$collStats` バグ修正およびその他の変更 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.15902) 大規模なコレクションに影響を与えるインデックスの作成中の問題を修正しました。

日付	リリースノート
2025 年 7 月 28 日	新しい特徴 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.14228) Amazon DocumentDB インスタンスベースのクラスターは、オンデマンドの自動スケーリング設定である Amazon DocumentDB Serverless をサポートするようになりました。 詳細については、「Amazon DocumentDB Serverless の使用」およびこの「最新情報のブログ記事」を参照してください。
2025 年 3 月 24 日	新機能 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.11800) 変更ストリームで `postBatchResumeToken` のサポートを追加しました。 詳細については、「`postBatchResumeToken` を用いた変更ストリームの再開」を参照してください。

日付	リリースノート
2025 年 1 月 15 日	新しい特徴 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.11800) Amazon DocumentDB は、ライターインスタンスの再起動によって読み取り可用性を維持するようになりました。リーダーインスタンスは、ライターインスタンスの再起動中に読み取りリクエストを引き続き処理するようになりました。 バグ修正およびその他の変更 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.11800) • 一括挿入の特殊なケースを処理する `killOp` コマンドを修正しました。 • Amazon DocumentDB インスタンスのネットワーク I/O 関連のメモリ使用量が改善されました。 • `$text` フィルタークエリの `count` コマンドを修正しました。
2024 年 12 月 18 日	新しい特徴 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.5721) Amazon DocumentDB インスタンスベースのクラスターは、NVMe-backed R6GD インスタンスをサポートするようになりました。 詳細については、「NVMe-backed インスタンス」およびこの「最新情報のブログ記事」を参照してください。

日付	リリースノート
2024 年 11 月 6 日	新しい特徴 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.11051) • 最小 TLS バージョン `tls1.2+` と `tls1.3+` のサポートを `tls` クラスターパラメータに追加しました。 • フィールド名の文字 dollar($) と dot(.) のサポートを有効にしました。機能の違いについては、「フィールド名のドル ($) とドット (.)」を参照してください。
2024 年 9 月 17 日	新しい特徴 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.10696) Amazon DocumentDB は、`runCommand` の `reIndex` でインデックスの再構築をサポートするようになりました。詳細については、「`reIndex` を使用したインデックスのメンテナンス」を参照してください。 Note `reIndex` は Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.10696 以降) でのみサポートされています。 バグ修正およびその他の変更 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.10696) • `$setOnInsert` は、更新オペレーションの結果として挿入中に `_id` フィールドをサポートするようになりました。 • Amazon DocumentDB 5.0 へのメジャーバージョンアップグレード後にストレージ容量を再利用できない問題を修正しました。

日付	リリースノート
2024 年 8 月 22 日	新機能 Amazon DocumentDB 5.0 (すべてのエンジンパッチバージョン) Amazon DocumentDB グローバルクラスターは、クラスターのスイッチオーバーとマネージドクラスターのフェイルオーバーをサポートするようになりました。詳細については、「Amazon DocumentDB グローバルクラスターのスイッチオーバーの実行」および「Amazon DocumentDB グローバルクラスターのマネージドフェイルオーバーの実行」を参照してください。 **Note** グローバルクラスターのスイッチオーバーとフェイルオーバーは、Amazon DocumentDB 4.0 および 5.0 でのみサポートされています。

<table>
<tr><th>日付</th><th>リリースノート</th></tr>
<tr><td>2024 年 7 月 23 日</td><td>新しい特徴

Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.8126)

• 長いインデックス名 (最大 255 文字) のサポートが追加されました。詳細については、命名に関する制約を参照してください。
• 最大接続限度を 2 倍に引き上げました。

<table>
<tr><th>インスタンスタイプ</th><th>元の限度</th><th>新しい制限</th></tr>
<tr><td>t3.medium</td><td>500</td><td>1,000</td></tr>
<tr><td>t4g.medium</td><td>500</td><td>1,000</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>
</table>
</td></tr>
</table>

日付	リリースノート

インスタンスタイプ	元の限度	新しい制限
r6g.8xlarge	28400	60000
r6g.12xlarge	30000	60000
r6g.16xlarge	30000	60000

詳細については、「インスタンスクォータ」を参照してください。

バグ修正およびその他の変更

Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.8126)

リーダーの変更ストリームで `CurrentTime` と `ResumeToken` を同期させるロジックを強化しました。

日付	リリースノート

2024 年 7 月 22 日	新しい特徴
	
	Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.6742)
	
	• DML 監査フィルタリングのサポートを追加しました。フィルター条件を設定することにより、すべての DML クエリを口グに記録する代わりに特定の要件に基づいて DML 監査口グをフィルタリングできるようになりました。詳細については、「DML 監査イベントのフィルタリング」を参照してください。
	• ドキュメント圧縮のサポートを以下に追加しました。
	  • 最小圧縮しきい値の設定
	  • 既存のコレクションで圧縮を有効にする (新しいドキュメントに適用)
	  • クラスターレベルでデフォルトの圧縮設定を許可する
	
	  詳細については、「コレクションレベルのドキュメント圧縮の管理」を参照してください。
	• リーダーインスタンスで変更ストリームを使用するためのサポートを追加しました。詳細については、「セカンダリインスタンスでの変更ストリームの使用」を参照してください。

2024 年 7 月 9 日	新機能 Amazon DocumentDB 5.0 (最小エンジンパッチバージョン 3.0.4780) では、TLS CA 証明書の更新でシステムを再起動する必要がなくなりました。詳細については、「Amazon DocumentDB TLS 証明書の更新」を参照してください。

日付	リリースノート
2024 年 5 月 29 日	新しい特徴 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.6742) • `$regexMatch` および `$regexFind` 演算子のサポートを追加しました。 • 大きな整数に対処する際に、監査ログの完全精度を確保するサポートが追加されました。監査ログですべての数値の正確な数値表現を維持できるようになり、精度の低下が防止されます。
2024 年 4 月 3 日	新しい特徴 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.5721) • `bypassDocumentValidation` のサポートと、`$jsonSchema` の詳細なエラーメッセージのサポートをを追加しました。`bypassDocumentValidation` の詳細については、「bypassDocumentValidation」を参照してください。 • `$expr` のサポートが追加されました。 • `$lookup` で非相関結合のサポートが追加されました。 • `$out` 集約ステージで検証ルールを保持するサポートを追加しました。

日付	リリースノート
2024 年 1 月 10 日	新しい特徴 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.4574、3.0.4780、3.0.4960) • HNSW ベクトルインデックスのサポートを追加しました。詳細については、「Amazon DocumentDB のベクトル検索」を参照してください。 • 部分インデックスのサポートを追加しました。詳細については、「部分インデックス」を参照してください。 • `currentOp` コマンド内のコレクションでの GC ランタイムのサポートを追加しました。 • Amazon DocumentDB でのネイティブテキスト検索のテキストインデックスサポートを追加しました。詳細については、「Amazon DocumentDB でのテキスト検索の実行」を参照してください。 • `$jsonSchema` スキーマキーワード `type`、`allOf`、`oneOf`、`anyOf`、`not`、`maxItems`、`minItems`、`maxProperties` 、`minProperties` 、`pattern`、`patternProperties` 、`multipleOf` 、`dependencies` 、`uniqueItems` のサポートが追加されました。 詳細については、「JSON スキーマ検証の使用」を参照してください。 • 算術演算子 `$ceil`、`$floor`、`$ln`、`$log`、`$log10`、`$sqrt`、`$exp` のサポートが追加されました。 詳細については、「算術演算子」を参照してください。 • 条件式演算子 `$switch` のサポートを追加しました。 • 並列 `IVFFLAT` ベクトルインデックスビルドのサポートが追加されました。開発者ガイドからパラレル `IVFFLAT` ベクトルインデックスビルドの制限を削除してドキュメントを更新しました。 バグ修正およびその他の変更

日付	リリースノート
	• `db.runCommand("dbstats")` の呼び出しで大文字と小文字を区別しない機能を復活させました。3.0.4960 または 2.0.10221 より前のエンジンパッチバージョンの Amazon DocumentDB 5.0 のお客様は、これらの最新のエンジンパッチを適用する必要があります。 • mongo シェルバージョン 1.7 以降で `db.coll.stats()` を呼び出すときに発生していたエラーを修正しました。開発者ガイドから mongo シェルの `db.coll.stats()` トラブルシューティングのヒントを削除してドキュメントを更新しました。
2023 年 11 月 29 日	新しい特徴 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.3727) ベクトル検索のサポートを追加しました。詳細については、こちらのブログ記事と、「Amazon DocumentDB 開発者ガイド」の「Amazon DocumentDB のベクトル検索」を参照してください。
2023 年 11 月 21 日	新しい特徴 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.3727) I/O 最適化ストレージのサポートを追加しました。詳細については、「Amazon DocumentDB 開発者ガイド」の「Amazon DocumentDB クラスターストレージ設定」を参照してください。 SageMaker Canvas とのノーコード機械学習の統合を追加しました。詳細については、「Amazon DocumentDB 開発者ガイド」の「Amazon SageMaker AI Canvas を使用したノーコード機械学習」を参照してください。

日付	リリースノート
2023 年 11 月 6 日	Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.3727) 新しい特徴 • $jsonSchema スキーマキーワード maxLength 、minLength 、maximum、minimum、exclusiveMaximum 、exclusiveMinimum 、items、additionalItems のサポートが追加されました。 JSON スキーマの検証は、インスタンスベースのクラスターでのみサポートされることに注意してください。 • $convert 集約パイプライン演算子とその省略形の派生演算子 $toBool、$toInt、$toLong、$toDouble 、$toString 、$toDecimal 、$toObjectId 、$toDate のサポートが追加されました。 • 集合式演算子 $setDifference 、$anyElementTrue 、$allElementTrue のサポートが追加されました。 バグ修正およびその他の変更 -NaN から NaN へのチェンジストリームの更新が表示されない問題を修正しました。
2023 年 9 月 15 日	新しい特徴 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.3140) インスタンスベースのクラスターでのみ $jsonSchema スキーマ検証のサポートが追加されました。詳細については、「JSON スキーマ検証の使用」を参照してください。

日付	リリースノート
2023 年 7 月 13 日	新しい特徴 Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.1948) • ドキュメント圧縮のサポートが追加されました。 • 並列インデックスビルドのサポートが追加されました。 • インデックスビルドステータスのサポートが追加されました。 バグ修正およびその他の変更 [Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.1948)] • ユーザーがシステムコレクションにアクセスできない場合の Amazon DocumentDB Elasticクラスターの `createCollection` の認証問題を修正しました。 • セカンダリリージョンインスタンスが同じプライマリリージョンインスタンス名を使用できない問題を修正しました。
2023 年 6 月 7 日	バグ修正およびその他の変更 Amazon DocumentDB 5.0 • r5 および t3.medium インスタンスが Amazon DocumentDB 5.0 でサポートされるようになりました。 • `engineVersion` オプションデフォルトは AWS SDK、 AWS CLI、および `5.0.0` です CloudFormation。

日付	リリースノート
2023 年 5 月 10 日	バグ修正およびその他の変更 [Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.1361)] • createIndex コマンドで `ignoreunknownindexoptions` のサポートが追加されました。 • 監査ログへの内部モニタリングクエリの追加を停止しました。 • ユーザー定義のローカル変数名で、`$let` や `$filter` などの射影演算子の「_」(アンダースコア) をサポートするようになりました。

日付	リリースノート

2023 年 3 月 1 日	新しい特徴
	
	[Amazon DocumentDB 5.0 (エンジンパッチバージョン 3.0.775)]
	
	• Amazon DocumentDB 5.0 の追加
	  • MongoDB 5.0 との互換 (MongoDB 5.0 API ドライバーのサポート)
	  • クライアント側のフィールドレベル暗号化 (FLE) のサポート。Amazon DocumentDB クラスターにデータを書き込む前に、クライアント側でフィールドを暗号化できるようになりました。詳細については、「クライアント側のフィールドレベル暗号化」を参照してください。
	  • 新しい集約演算子: `$dateAdd`、`$dateSubtract`
	• すべてのインスタンスベースの Amazon DocumentDB クラスターとシャードベースのエラスティッククラスターのストレージ制限を 128 TiB に増大。
	• Amazon DocumentDB 5.0 では、最初のネストレベルで `$elemMatch` 演算子によるインデックススキャンがサポートされるようになりました。クエリに 1 つのレベルの `$elemMatch` フィルターしかなく、ネストされた `$elemMatch` クエリがインデックススキャンをサポートしていない場合、インデックススキャンがサポートされます。
	
	インデックススキャンをサポートするクエリシェイプは以下のとおりです。
	
	`db.foo.find( { "a": {$elemMatch: { "b": "xyz", "c": "abc"} } })`
	
	インデックススキャンをサポートしないクエリシェイプは以下のとおりです。
	
	`db.foo.find( { "a": {$elemMatch: { "b": {$elemMatch: { "d": "xyz", "e": "abc"} }} } })`

Amazon DocumentDB 4.0

Amazon DocumentDB エンジンバージョン 4.0 (MongoDB 4.0 互換) のリリースノート。

日付	リリースノート
2026 年 8 月 31 日	新機能 Amazon DocumentDB は、エンジンバージョン 4.0 からバージョン 8.0 へのインプレースメジャーバージョンアップグレード (MVU) を直接サポートするようになりました。これにより、中間バージョンのアップグレードが不要になり、既存のデータ、設定、クラスター設定を維持しながら 4.0 クラスターをアップグレードできます。このアップグレードは、これらのバージョンが現在サポートされているすべての AWS リージョンで使用できます。詳細については、「Amazon DocumentDB のインプレースメジャーバージョンアップグレード」を参照してください。
2026 年 8 月 18 日	セキュリティと安定性の向上 Amazon DocumentDB 4.0 (エンジンバージョン: 2.0.14287) [進行中] このエンジンパッチは、次の改善点を提供します。 • リクエスト処理を改善 Amazon DocumentDB は、接続パスとメッセージ解析パスに沿って受信クライアントリクエストをより徹底的に検証するようになりました。これにより、幅広い入力パターンで耐障害性と安定性が向上します。 • 認証ハンドシェイクの強化 Amazon DocumentDB は、認証交換をより徹底的に検証するようになりました。これにより、予期しない入力や不正な形式の入力を処理する際の堅牢性が向上します。 • アクセスコントロールの適用 Amazon DocumentDB は、ネストされたクエリオペレーションに必要な権限をすべてのネストレベルで一貫して検証するようになりました。 より早く適用しない場合、Amazon DocumentDB はリリースから 30 日後に次のメンテナンスウィンドウ中にこのパッチを自動的に適用します。

日付	リリースノート
2026 年 4 月 27 日	バグ修正およびその他の変更 Amazon DocumentDB 4.0 (エンジンバージョン: 2.0.14278) これは、以下の問題に対処するために必要なエンジンパッチです。 1. まれに、point-in-timeリカバリ (PITR) とメジャーバージョンアップグレード (MVU) が応答しなくなる問題を修正しました。 すべての Amazon DocumentDB 4.0 クラスターは、次のメンテナンスウィンドウ中にこのパッチバージョンに自動的にアップグレードされます。
2026 年 3 月 9 日	バグ修正およびその他の変更 Amazon DocumentDB 4.0 (エンジンバージョン: 2.0.14165) この更新では、次の点が改善されたオプションのエンジンパッチが提供されます。 1. `DatabaseConnectionsActiveMax` メトリクスが`Connections (active)` インスタンスを追跡するようになりました。詳細については、「インスタンスクォータ」を参照してください。 2. SASL 認証のパフォーマンスが向上しました。
2025 年 10 月 22 日	新しい特徴 Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.11747) ガベージコレクションと老朽化コレクションに関するコレクションの全体的な状態に関する情報を提供する「MVCCIDStats」と「gcRuntimeStats」の新しいフィールドを `collstats` に追加しました。

日付	リリースノート
2025 年 10 月 16 日	新機能 Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.11747) Amazon DocumentDB は、クラスターが仮想プライベートクラウド (VPC) にあるときにデュアルスタックモード (IPv4/IPv6) をサポートするようになりました。 詳細については、こちらの ブログ記事 と、「Amazon DocumentDB 開発者ガイド」の「Amazon VPC と Amazon DocumentDB」を参照してください。
2025 年 7 月 29 日	新しい特徴 Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.11747) • インデックスの肥大化とコレクションベースの肥大化メトリクスがリリースされました • ガベージコレクション と老朽化コレクションに関するクラスターの全体的な状態に関する情報を提供する CloudWatch メトリクス `AvailableMVCCIds` と `LongestActiveGCRuntime` を追加しました。 • 次の文字列演算子のサポート: `$trim`、`$rtrim`、および `$ltrim` • 次のステージ演算子のサポート: `$collStats` バグ修正およびその他の変更 Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.11747) 大規模なコレクションに影響を与えるインデックスの作成中の問題を修正しました。

日付	リリースノート
2025 年 1 月 15 日	バグ修正およびその他の変更 Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.11096) • 一括挿入の特殊なケースを処理する killOp コマンドを修正しました。 • Amazon DocumentDB インスタンスのネットワーク I/O 関連のメモリ使用量が改善されました。
2024 年 11 月 6 日	新しい特徴 Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.10980) • 最小 TLS バージョン tls1.2+ と tls1.3+ のサポートを tls クラスターパラメータに追加しました。
2024 年 9 月 17 日	バグ修正およびその他の変更 Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.10898) • $setOnInsert は、更新オペレーションの結果として挿入中に _id フィールドをサポートするようになりました。 • Amazon DocumentDB 5.0 へのメジャーバージョンアップグレード後にストレージ容量を再利用できない問題を修正しました。

日付	リリースノート
2024 年 8 月 22 日	新機能 Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.5704) Amazon DocumentDB グローバルクラスターは、クラスターのスイッチオーバーとマネージドクラスターのフェイルオーバーをサポートするようになりました。詳細については、「Amazon DocumentDB グローバルクラスターのスイッチオーバーの実行」および「Amazon DocumentDB グローバルクラスターのマネージドフェイルオーバーの実行」を参照してください。 **Note** グローバルクラスターのスイッチオーバーとフェイルオーバーは、Amazon DocumentDB 4.0 および 5.0 でのみサポートされています。

<table>
<tr><th>日付</th><th>リリースノート</th></tr>
<tr><td>2024 年 7 月 23 日</td><td>新しい特徴

Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.10709)

• 長いインデックス名 (最大 255 文字) のサポートが追加されました。詳細については、命名に関する制約を参照してください。
• 最大接続限度を 2 倍に引き上げました。

<table>
<tr><th>インスタンスタイプ</th><th>元の限度</th><th>新しい制限</th></tr>
<tr><td>t3.medium</td><td>500</td><td>1,000</td></tr>
<tr><td>t4g.medium</td><td>500</td><td>1,000</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>
</table>
</td></tr>
</table>

<table>
<tr><th>日付</th><th>リリースノート</th></tr>
<tr><td></td><td>
<table>
<tr><th>インスタンスタイプ</th><th>元の限度</th><th>新しい制限</th></tr>
<tr><td>r6g.8xlarge</td><td>28400</td><td>60000</td></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>
詳細については、「インスタンスクォータ」を参照してください。
</td></tr>
<tr><td>2024 年 7 月 22 日</td><td>新しい特徴

Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.10593)

• DML 監査フィルタリングのサポートを追加しました。フィルター条件を設定することにより、すべての DML クエリをログに記録する代わりに特定の要件に基づいて DML 監査ログをフィルタリングできるようになりました。詳細については、「DML 監査イベントのフィルタリング」を参照してください。</td></tr>
<tr><td>2024 年 7 月 9 日</td><td>新機能

Amazon DocumentDB 4.0 (最小エンジンパッチバージョン 2.0.10179) では、TLS CA 証明書の更新でシステムを再起動する必要がなくなりました。詳細については、「Amazon DocumentDB TLS 証明書の更新」を参照してください。</td></tr>
<tr><td>2024 年 5 月 29 日</td><td>新しい特徴

Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.10593)

• 大きな整数に対処する際に、監査ログの完全精度を確保するサポートが追加されました。監査ログですべての数値の正確な数値表現を維持できるようになり、精度の低下が防止されます。</td></tr>
</table>

日付	リリースノート
2024 年 4 月 3 日	新しい特徴 Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.10392) • bypassDocumentValidation のサポートを $jsonSchema に追加しました。bypassDocumentValidation の詳細については、「bypassDocumentValidation」を参照してください。 • $expr のサポートが追加されました。 • $lookup で非相関結合のサポートが追加されました。 • $out 集約ステージで検証ルールを保持するサポートを追加しました。
2024 年 1 月 10 日	新しい特徴 Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.10124、2.0.10179、2.0.10221) • currentOp コマンド内のコレクションでの GC ランタイムのサポートを追加しました。 • $jsonSchema スキーマキーワード type、allOf、oneOf、anyOf、not、maxItems、minItems、maxProperties 、minProperties 、pattern、patternProperties 、multipleOf 、dependencies 、uniqueItems のサポートが追加されました。 詳細については、「JSON スキーマ検証の使用」を参照してください。 • 算術演算子 $ceil、$floor、$ln、$log、$log10、$sqrt、$exp のサポートが追加されました。 詳細については、「算術演算子」を参照してください。 • 条件式演算子 $switch のサポートを追加しました。

日付	リリースノート
2023 年 12 月 20 日	その他の変更 Amazon DocumentDB 4.0 でのインプレースメジャーバージョンアップグレードのサポートを有効にしました。詳細については、「Amazon DocumentDB のインプレースメジャーバージョンアップグレード」を参照してください。
2023 年 11 月 6 日	Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.9876) 新しい特徴 • `$jsonSchema` スキーマキーワード `maxLength` 、`minLength` 、`maximum`、`minimum`、`exclusiveMaximum` 、`exclusiveMinimum` 、`items`、`additionalItems` のサポートが追加されました。 JSON スキーマの検証は、インスタンスベースのクラスターでのみサポートされることに注意してください。 • `$convert` 集約パイプライン演算子とその省略形の派生演算子 `$toBool`、`$toInt`、`$toLong`、`$toDouble` 、`$toString` 、`$toDecimal` 、`$toObjectId` 、`$toDate` のサポートが追加されました。 • 集合式演算子 `$setDifference` 、`$anyElementTrue` 、`$allElementTrue` のサポートが追加されました。 バグ修正およびその他の変更 `-NaN` から `NaN` へのチェンジストリームの更新が表示されない問題を修正しました。
2023 年 9 月 20 日	新しい特徴 Amazon DocumentDB 4.0 でのインプレースメジャーバージョンアップグレードのサポートが追加されました。詳細については、「Amazon DocumentDB のインプレースメジャーバージョンアップグレード」を参照してください。

日付	リリースノート
2023 年 9 月 15 日	新しい特徴 Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.9686) インスタンスベースのクラスターでのみ $jsonSchema スキーマ検証のサポートが追加されました。詳細については、「JSON スキーマ検証の使用」を参照してください。
2023 年 7 月 13 日	新しい特徴 Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.9259) • 並列インデックスビルドのサポートが追加されました。 バグ修正およびその他の変更 Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.9259) • 監査ログへの内部モニタリングクエリの追加を停止しました。

日付	リリースノート
2023 年 4 月 4 日	バグ修正およびその他の変更 [Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.8934)] • 進行中のワークロード中に DML 監査が有効になっている場合の問題を修正しました。 • ヒント付きの集計コマンドに文字列値が渡された場合の DML 監査の問題を修正しました。 • readwriteanydatabase ロールを持つユーザーが authorizedCollections オプションと nameOnly オプションの両方が true に設定されていると、`listCollections` コマンドが機能しないという問題が修正されました。 • フィールド名の数字文字列を適切に解析するように問題を修正しました。 • 実行時間の長いカーソルがガベージコレクションに影響する場合、そのカーソルはキャンセルされます。 • ユーザー定義のローカル変数名で、`$let` や `$filter` などの射影演算子の「_」(アンダースコア) をサポートするようになりました。
2023 年 2 月 27 日	バグ修正およびその他の変更 Amazon DocumentDB 4.0 のサポートが追加されました AWS Lambda。詳細については、「変更ストリーム AWS Lambda で を使用する」を参照してください。

日付	リリースノート
2022 年 8 月 9 日	新しい特徴 4.0 • Decimal128 データ型のサポートが追加されました。Decimal128 は、DocumentDB が利用可能なすべてのリージョンでサポートされている BSON データ型です。 詳細については、「データ型」を参照してください。 • Amazon CloudWatch Logs を用いてデータベースイベントを監査するためのサポートを追加しました。Amazon DocumentDB で、データ操作言語 (DML) イベントとデータ定義言語 (DDL) イベントを Amazon CloudWatch Logs に記録できるようになりました。 詳細については、このブログ記事を参照してください。 バグ修正およびその他の変更 4.0 • 自分のパスワードを、changeOwnPassword 権限のある独自のパスワードで変更できるようになりました。
2022 年 7 月 25 日	新しい特徴 Amazon DocumentDB 4.0 同じ DocumentDB クラスターボリュームを使用し、元のクラスターと同じデータを持つクローンを作成できるようになったため、クラスターをより速く作成できるようになりました。詳細については、「Amazon DocumentDB クラスターの管理」を参照してください。

日付	リリースノート
2022 年 6 月 27 日	新しい特徴 [Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.7509)] Amazon DocumentDB は、使用パターンに基づいてデータベースのサイズを動的に変更します。データを追加するとスペースは最大 64 テビバイト (TiB) まで増加し、データを削除すると割り当てられるスペースが減少します。
2022 年 4 月 7 日	新しい特徴 4.0 (エンジンパッチバージョン 2.0.6142 および 2.0.6948) Amazon DocumentDB Performance Insights は現在プレビュー段階にあります。7 日間のパフォーマンス履歴を追加費用なしでローリングウィンドウに保存できるようになりました。詳細については、「Performance Insights を使用したモニタリング」を参照してください。

2022 年 1 月 24 日	新しい特徴
	
	4.0 (エンジンパッチバージョン 2.0.5170)
	
	• Amazon DocumentDB で無料トライアルが利用可能になりました。詳細については、「Amazon DocumentDB の無料トライアル」ページを参照してください。
	• 次の API を含む拡張機能を地理空間クエリで使用できるようになりました。
	  • `$geoWithin`
	  • `$geoIntersects`
	• 以下の MongoDB 演算子のサポートを追加しました。
	  • `$mergeObjects`
	  • `$reduce`
	
	詳細については、「Amazon DocumentDB による地理空間データのクエリ」を参照してください。

日付	リリースノート

2022 年 1 月 21 日	新しい特徴
	
	[Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.5706)]
	
	• Amazon DocumentDB Graviton2 (r6g.large、r6g.2xlarge、r6g.4xlarge、r6g.8xlarge、r6g.12xlarge、r6g.16xlarge、t4g.medium) インスタンスがサポートされるようになりました。
	
	新しい特徴
	
	4.0 (エンジンパッチバージョン 2.0.5706)
	
	• 以下の MongoDB API のサポートを追加しました。
	  • `$reduce`
	  • `$mergeObjects`
	  • `$geoWithin`
	  • `$geoIntersects`

日付	リリースノート

2021 年 10 月 25 日	新しい特徴
	
	4.0 (エンジンパッチバージョン 2.0.5704)
	
	• 以下の MongoDB API のサポートを追加しました。
	  • `$literal`
	  • `$map`
	  • `$$ROOT`
	• GeoSpatial クエリ機能をサポートしました。詳しくはこちらの ブログ投稿 をご覧ください
	• ユーザー定義ロールによるアクセスコントロールに対応しています。詳しくはこちらの ブログ投稿 をご覧ください
	• Amazon DocumentDB JDBC ドライバにより、Tableau などの BI ツールや SQL Workbench などのクエリツールからの接続を有効にします。
	
	バグ修正およびその他の変更
	
	4.0 (エンジンパッチバージョン 2.0.5704)
	
	• `$natural` と共に明示的な `.sort()` が存在する場合、`$natural` が正しくソートされるようにバグ修正
	• 変更ストリームが `$redact` で動作するようにバグ修正
	• `$ifNull` が空の配列でも動作するようにバグ修正
	• 現在ログインしているユーザーを削除したり、進行中のアクティビティに対するユーザーの権限を取り消すと、リソースが過剰に消費され、サーバーがクラッシュする不具合を修正
	• `listDatabase` と `listCollection` の権限チェックのバグ修正
	• マルチキーエレメントのデデュープロジックのバグ修正

日付	リリースノート
2021 年 6 月 24 日	新しい特徴 4.0 (エンジンパッチバージョン 2.0.3371) • r5.8xlarge と r5.16xlarge インスタンスがサポートされるようになりました。詳細については、ブログ記事「Amazon DocumentDB で r5.8xlarge と r5.16xlarge インスタンスがサポートされるようになりました」を参照してください。 • グローバルクラスターがサポートされ、リージョン全体の停止からのディザスタリカバリが可能になり、最寄りの Amazon DocumentDB クラスターからの読み取りを許可することで、低レイテンシーのグローバル読み取りが可能になりました。グローバルクラスターは、南米 (サンパウロ)、欧州 (ミラノ)、中国 (北京)、中国 (寧夏) の各リージョンでは現在利用できないことに注意してください。

日付	リリースノート
2021 年 5 月 4 日	新しい特徴 4.0 (エンジンパッチバージョン 2.0.3371) • renameCollection • $zip • $indexOfArray • $reverseArray • $natural • $hint アップデートのサポート • distinct のインデックススキャン バグ修正およびその他の変更 4.0 (エンジンパッチバージョン 2.0.3371) • $in クエリのメモリ使用量を削減しました • マルチキーインデックスのメモリリークを修正しました • $out の説明プランとプロファイラーの出力を修正しました • 信頼性を向上させるため、内部モニタリングシステムからの操作にタイムアウトを追加しました • マルチキーインデックスに渡されるクエリ述語に影響する不具合を修正しました。
2021 年 1 月 15 日	新しい特徴 [Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.722)] • なし

日付	リリースノート
2020 年 11 月 9 日	新しい特徴 [Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.722)] • MongoDB 4.0 の互換 • ACID トランザクション • cluster(client.watch()　または mongo.watch()) および データベースレベルの (db.watch()) 変更ストリームのサポート • startAtOperationTime　を使用して変更ストリームを開始または再開できます • 変更ストリームの保存期間を 7 日間 (以前は 24 時間) に延長します • AWS DMS Amazon DocumentDB 4.0 の ターゲット • CloudWatch メトリクス: TransactionsOpen 、TransactionsOpenMax 、TransactionsAborted 、TransactionsStarted 、TransactionsCommitted • currentOp 、ServerStatus 、profiler のトランザクション用の新しいフィールド。 • $lookup の集約ステージでインデックスを使用できるようになりました • 予測を含む find( ) クエリは、インデックス (対象クエリ) からの指示で配信できます • hint( ) と findAndModify　の併用が可能です • $addToSet　演算子のパフォーマンス最適化 • インデックス全体のサイズを小さくするための改善。 • 新しい集約演算子: $ifNull、$replaceRoot 、$setIsSubset 、$setInstersection 、$setUnion 、$setEquals • ListCollection　と ListDatabase　コマンドで、オプションとして authorizedCollections　と authorizedDatabases　パラメータを使用することができるようになり、それぞれ listCollections　と listDatabase　ロールを必要とすること

日付	リリースノート
	なく、ユーザーがアクセス許可を持つコレクションとデータベースを一覧表示できるようになりました。 • また、ユーザーは KillCursor ロールを必要とせずに自分でカーソルを終了することもできます • サブドキュメントの数値タイプの比較が、第 1 レベルのドキュメントの数値タイプの比較と一致するようになりました。Amazon DocumentDB 4.0 での動作が MongoDB と互換になりました。 バグ修正およびその他の変更 [Amazon DocumentDB 4.0 (エンジンパッチバージョン 2.0.722)] • 位置演算子 $ を使用する場合、$setOnInsert は更新できなくなりました。Amazon DocumentDB 4.0 での動作が MongoDB と互換になりました。 • $createCollection とセット autoIndexId の問題を修正しました • ネストされたドキュメントの投影 • ワーキングメモリーのデフォルト設定をインスタンスのメモリサイズに合わせてスケールするように変更しました • ガベージコレクションの改善 • パス内の空のキーでのルックアップ、mongo との動作の違い • タイムゾーンの動作の dateToString バグを修正 • ソート順を尊重するように $push (集計) を修正 • $currentOp の集計のバグを修正 • セカンダリのreadPreference の問題を修正しました • $createIndex がコマンド発行時のデータベースと同じであることを検証する際の問題を修正しました • minKey、maxKey 検索に失敗したときの一貫性のない動作を修正しました • $size 演算子が複合配列で動作しないという問題が修正されました • 正規表現による $in の否定に関する問題が修正されました

日付	リリースノート
	• ビューに対する $distinct コマンド実行の問題を修正しました • 集約コマンドと検索コマンドで、欠落しているフィールドを異なる方法でソートする問題を修正しました • 正規表現への $eq がタイプをチェックしない問題を修正しました • タイムスタンプの順序位置動作の $currentDate バグを修正しました • $currentDate のミリ秒単位の精度が修正されました

Amazon DocumentDB 3.6

Amazon DocumentDB エンジンバージョン 3.6 (MongoDB 3.6 互換) のリリースノート。

日付	リリースノート
2026 年 8 月 31 日	新機能 Amazon DocumentDB は、エンジンバージョン 3.6 からバージョン 8.0 へのインプレースメジャーバージョンアップグレード (MVU) を直接サポートするようになりました。これにより、中間バージョンのアップグレードが不要になり、既存のデータ、設定、クラスター設定を維持しながら 3.6 クラスターをアップグレードできます。このアップグレードは、これらのバージョンが現在サポートされているすべての AWS リージョンで使用できます。詳細については、「Amazon DocumentDB のインプレースメジャーバージョンアップグレード」を参照してください。
2026 年 8 月 18 日	セキュリティと安定性の向上 Amazon DocumentDB 3.6 (エンジンバージョン: 1.0.208702) [進行中] このエンジンパッチでは、次の点が改善されています。 • リクエスト処理を改善 Amazon DocumentDB は、接続パスとメッセージ解析パスに沿って受信クライアントリクエストをより徹底的に検証するようになりました。これにより、幅広い入力パターンで耐障害性と安定性が向上します。

日付	リリースノート
	• 認証ハンドシェイクの強化 Amazon DocumentDB は、認証交換をより徹底的に検証するようになりました。これにより、予期しない入力や不正な形式の入力を処理する際の堅牢性が向上します。 より早く適用しない場合、Amazon DocumentDB はリリースから 30 日後に次のメンテナンスウィンドウ中にこのパッチを自動的に適用します。
2026 年 4 月 27 日	新しい特徴 Amazon DocumentDB 3.6 (エンジンバージョン: 1.0.208692) この更新では、以下の改善点を含むオプションのエンジンパッチが提供されます。 1. 変更ストリームで `startAtOperationTime` のサポートを追加しました。
2026 年 2 月 12 日	バグ修正およびその他の変更 Amazon DocumentDB 3.6 (エンジンバージョン: 1.0.208675) 1. `DatabaseConnectionsActiveMax` メトリクスが`Connections (active)` インスタンスを追跡するようになりました。詳細については、「インスタンスクォータ」を参照してください。 2. SASL 認証のパフォーマンスが向上しました。 3. $indexStats コマンドのバグ修正。
2024 年 8 月 20 日	新機能 Amazon DocumentDB 3.6 (最小エンジンパッチバージョン 1.0.208662) では、TLS CA 証明書の更新にシステムの再起動が不要になりました。詳細については、「Amazon DocumentDB TLS 証明書の更新」を参照してください。

日付	リリースノート
2024 年 7 月 22 日	新しい特徴 Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.208662) Amazon DocumentDB MVU の db.r5.* および db.r6.* インスタンスのインデックス制限を排除しました。詳細については、「前提条件」を参照してください。 バグ修正およびその他の変更 Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.208662) Amazon DocumentDB が、-NaN を有効な JSON トークンとして認識するようになりました。
2023 年 12 月 20 日	その他の変更 Amazon DocumentDB 3.6 でのインプレースメジャーバージョンアップグレードのサポートを有効にしました。詳細については、「Amazon DocumentDB のインプレースメジャーバージョンアップグレード」を参照してください。
2023 年 11 月 17 日	バグ修正およびその他の変更 Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.208570) ユーザー定義のローカル変数名で、`$let` や `$filter` などの射影演算子の「_」(アンダースコア) をサポートするようになりました。
2023 年 9 月 20 日	新しい特徴 Amazon DocumentDB 3.6 でのインプレースメジャーバージョンアップグレードのサポートが追加されました。詳細については、「Amazon DocumentDB のインプレースメジャーバージョンアップグレード」を参照してください。

日付	リリースノート
2023 年 2 月 2 日	バグ修正およびその他の変更 [Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.208432)] • 進行中のワークロード中に DML 監査が有効になっている場合の問題を修正しました。 • ヒント付きの集計コマンドに文字列値が渡された場合の DML 監査の問題を修正しました。 • readwriteanydatabase ロールを持つユーザーが authorizedCollections オプションと nameOnly オプションの両方が true に設定されていると、`listCollections` コマンドが機能しないという問題が修正されました。 • フィールド名の数字文字列を適切に解析するように問題を修正しました。 • 実行時間の長いカーソルがガベージコレクションに影響する場合、そのカーソルはキャンセルされます。

日付	リリースノート
2022 年 8 月 9 日	新しい特徴 Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.208152) • Decimal128 データ型のサポートが追加されました。Decimal128 は、DocumentDB が利用可能なすべてのリージョンでサポートされている BSON データ型です。 詳細については、「データ型」を参照してください。 • Amazon CloudWatch Logs を用いてデータベースイベントを監査するためのサポートを追加しました。Amazon DocumentDB で、データ操作言語 (DML) イベントとデータ定義言語 (DDL) イベントを Amazon CloudWatch Logs に記録できるようになりました。 詳細については、このブログ記事を参照してください。 バグ修正およびその他の変更 Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.208152) • 自分のパスワードを、`changeOwnPassword` 権限のある独自のパスワードで変更できるようになりました。
2022 年 4 月 7 日	新しい特徴 Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.207836 および 1.0.208015) Amazon DocumentDB Performance Insights は現在プレビュー段階にあります。7 日間のパフォーマンス履歴を追加費用なしでローリングウィンドウに保存できるようになりました。詳細については、「Performance Insights を使用したモニタリング」を参照してください。

日付	リリースノート

2022 年 1 月 24 日	新しい特徴
	
	Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.207684)
	
	• Amazon DocumentDB で無料トライアルが利用可能になりました。詳細については、「Amazon DocumentDB の無料トライアル」ページを参照してください。
	• 次の API を含む拡張機能を地理空間クエリで使用できるようになりました。
	  • `$geoWithin`
	  • `$geoIntersects`
	• 以下の MongoDB 演算子のサポートを追加しました。
	  • `$mergeObjects`
	  • `$reduce`
	
	詳細については、「Amazon DocumentDB による地理空間データのクエリ」を参照してください。

2022 年 1 月 21 日	新しい特徴
	
	Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.207781)
	
	• 以下の MongoDB API のサポートを追加しました。
	  • `$reduce`
	  • `$mergeObjects`
	  • `$geoWithin`
	  • `$geoIntersects`

日付	リリースノート

2021 年 10 月 25 日	新しい特徴
	
	Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.207780)
	
	• 以下の MongoDB API のサポートを追加しました。
	  • `$literal`
	  • `$map`
	  • `$$ROOT`
	• GeoSpatial クエリ機能をサポートしました。詳しくはこちらの ブログ投稿 をご覧ください
	• ユーザー定義ロールによるアクセスコントロールに対応しています。詳しくはこちらの ブログ投稿 をご覧ください
	• Amazon DocumentDB JDBC ドライバにより、Tableau などの BI ツールや SQL Workbench などのクエリツールからの接続を有効にします。
	
	バグ修正およびその他の変更
	
	Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.207780)
	
	• `$natural` と共に明示的な `.sort()` が存在する場合、`$natural` が正しくソートされるようにバグ修正
	• 変更ストリームが `$redact` で動作するようにバグ修正
	• `$ifNull` が空の配列でも動作するようにバグ修正
	• 現在ログインしているユーザーを削除したり、進行中のアクティビティに対するユーザーの権限を取り消すと、リソースが過剰に消費され、サーバーがクラッシュする不具合を修正
	• `listDatabase` と `listCollection` の権限チェックのバグ修正
	• マルチキーエレメントのデデュープロジックのバグ修正

日付	リリースノート
2021 年 6 月 24 日	新しい特徴 Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.207117) • r5.8xlarge と r5.16xlarge インスタンスがサポートされるようになりました。詳細については、ブログ記事「Amazon DocumentDB で r5.8xlarge と r5.16xlarge インスタンスがサポートされるようになりました」を参照してください。 • グローバルクラスターがサポートされ、リージョン全体の停止からのディザスタリカバリが可能になり、最寄りの Amazon DocumentDB クラスターからの読み取りを許可することで、低レイテンシーのグローバル読み取りが可能になりました。グローバルクラスターは、南米 (サンパウロ)、欧州 (ミラノ)、中国 (北京)、中国 (寧夏) の各リージョンでは現在利用できないことに注意してください。

日付	リリースノート
2021 年 5 月 4 日	新しい特徴 Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.207117) • `renameCollection` • `$zip` • `$indexOfArray` • `$reverseArray` • `$natural` • `$hint` アップデートのサポート • `distinct` のインデックススキャン バグ修正およびその他の変更 Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.207117) • `$in` クエリのメモリ使用量を削減しました • マルチキーインデックスのメモリリークを修正しました • `$out` の説明プランとプロファイラーの出力を修正しました • 信頼性を向上させるため、内部モニタリングシステムからの操作にタイムアウトを追加しました • マルチキーインデックスに渡されるクエリ述語に影響する不具合を修正しました。

日付	リリースノート
2021 年 1 月 15 日	新しい特徴 [Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.206295)] • $lookup の集約ステージでインデックスを使用できるようになりました • 予測を含む find( ) クエリは、インデックス (対象クエリ) からの指示で配信できます • hint( ) と findAndModify の併用が可能です • $addToSet 演算子のパフォーマンス最適化 • インデックス全体のサイズを縮小するための改善 • 新しい集約演算子: $ifNull、$replaceRoot 、$setIsSubset 、$setInstersection 、$setUnion 、$setEquals • また、ユーザーは KillCursor ロールを必要とせずに自分でカーソルを終了することもできます
2020 年 11 月 9 日	新しい特徴 [Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.206295)] • なし バグ修正およびその他の変更 [Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.206295)] • なし

日付	リリースノート
2020 年 10 月 30 日	新しい特徴 [Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.206295)] • クラスターレベル (`client.watch()` または `mongo.watch()`) 、およびデータベース (`db.watch()`) で変更ストリームカーソルを開く機能が追加されました。 • 変更ストリームの保存期間を 7 日間 (以前は 24 時間) に延長する機能 バグ修正およびその他の変更 [Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.206295)] • 一般的なケースでのさまざまなパフォーマンスの改善 • 対象を絞ったセキュリティの改善 • 複合インデックスの 2 番目のフィールドのスキップソートに関する問題を修正しました • マルチキーインデックス (複合インデックスではない) の単一フィールドで等しくなるように標準インデックスを有効にする • 認証競合状態を修正しました • まれにガベージコレクションがクラッシュする原因となっていた問題を修正しました • RBAC のセキュリティ改善 • `databaseConnectionsMax` メトリクスの追加 • `r5.24xlarge` インスタンス上の特定ワークロードのパフォーマンスの改善

日付	リリースノート
2020 年 9 月 22 日	新しい特徴 [Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.206295)] • $out 集約段階 • インスタンスあたりの接続数とカーソルの最大数を 10 倍まで増やしました バグ修正およびその他の変更 [Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.206295)] • なし
2020 年 7 月 10 日	新しい特徴 [Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.206295)] • クロスリージョンスナップショットのコピー バグ修正およびその他の変更 [Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.206295)] • なし

日付	リリースノート
2020 年 6 月 30 日	新しい特徴 [Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.206295)] • t3 medium インスタンス バグ修正およびその他の変更 [Amazon DocumentDB 3.6 (エンジンパッチバージョン 1.0.206295)] • t3 インスタンスのアイドルメモリ再利用 • 認証の改善 • SASL 認証のパフォーマンスの改善 • 可能な最大操作数を超えた場合の `currentOp` 問題を修正しました • 一括更新と削除の `killOps` 問題を修正しました • `$match` による `$sample` パフォーマンスの改善 • 編集段階での cond ケースでの $$ サポートが修正されました • 繰り返し発生するさまざまなクラッシュの根本原因を修正しました • TTL スイープが改善され、IO とレイテンシーが削減されました • `$unwind` のメモリ使用率が最適化されました • コレクション統計とドロップインデックスの競合状態を修正しました • 同時インデックス構築中の競合状態を修正しました • インデックスの `hash_search` でまれに発生するクラッシュを修正しました

Amazon DocumentDB の一般的なリリースノート

リージョンの可用性、ツール統合、クロスバージョン機能などの一般的なサービスの更新。

日付	リリースノート
2026 年 9 月 15 日	新機能 Amazon DocumentDB サーバーレスインスタンスタイプが、次のリージョンの Amazon DocumentDB 5.0 および 8.0 エンジンバージョンで利用可能になりました。 • アジアパシフィック (大阪) • アジアパシフィック (ジャカルタ) • アジアパシフィック (マレーシア) • アジアパシフィック (メルボルン) • アジアパシフィック (タイ) • カナダ西部 (カルガリー) • 欧州 (チューリッヒ) • 欧州 (ストックホルム) • イスラエル (テルアビブ) • メキシコ (中部) 詳細については、「リージョン別のサポートされるインスタンスクラス」および「Amazon DocumentDB Serverless の使用」を参照してください。
2026 年 4 月 29 日	新機能 Amazon DocumentDB がカナダ西部 (カルガリー) リージョンで利用可能になりました。詳細については、このブログ記事を参照してください。
2026 年 4 月 17 日	新機能 DocumentDB サーバーレスでの高速スケーリングが利用可能になりました。
2026 年 2 月 11 日	新機能

日付	リリースノート
	Amazon DocumentDB がアジアパシフィック (メルボルン) リージョンで利用可能になりました。詳細については、このブログ記事を参照してください。 Amazon DocumentDB が欧州 (チューリッヒ) リージョンで利用可能になりました。詳細については、このブログ記事を参照してください。
2026 年 1 月 8 日	新機能 Amazon DocumentDB がアジアパシフィック (ジャカルタ) リージョンで利用可能になりました。詳細については、このブログ記事を参照してください。
2025 年 10 月 13 日	新機能 Amazon DocumentDB は、アジアパシフィック (ハイデラバード) リージョンで R6G および T4G インスタンスをサポートするようになりました。 詳細については、「リージョン別のサポートされるインスタンスクラス」を参照してください。
2025 年 10 月 7 日	新機能 Amazon DocumentDB が次のリージョンで利用可能になりました。 • アジアパシフィック (マレーシア) • アジアパシフィック (大阪) • アジアパシフィック (タイ) • メキシコ (中部) 詳細については、このブログ記事を参照してください。

日付	リリースノート
2025 年 9 月 26 日	新しい特徴 Amazon DocumentDB は、クロスリージョンおよびクロスアカウントスナップショットのコピーとスナップショット共有をサポートするようになりました。 詳細については、「間でのスナップショットのコピー AWS リージョン」および「Amazon DocumentDB クラスタースナップショットの共有」を参照してください。
2025 年 6 月 18 日	新機能 Amazon DocumentDB がイスラエル (テルアビブ) リージョンで利用可能になりました。
2025 年 5 月 8 日	新機能 Amazon DocumentDB が欧州 (ストックホルム) リージョンで利用可能になりました。詳細については、このブログ記事を参照してください。
2025 年 4 月 2 日	バグ修正およびその他の変更 バックグラウンドインデックスを使用したベクトルインデックス作成のバグを修正しました。
2025 年 2 月 6 日	新機能 Amazon DocumentDB が と統合されました AWS Toolkit for Visual Studio Code。詳細については、この「最新情報のブログ記事」を参照し、「AWS Toolkit for Visual Studio Code ユーザーガイド」の「ツールキットでの Amazon DocumentDB の使用」を参照してください。

日付	リリースノート
2024 年 10 月 22 日	新機能 Amazon DocumentDB は、ストレージネットワークスループットメトリクス `StorageNetworkReceiveThroughput` 、`StorageNetworkTransmitThroughput` 、および `StorageNetworkThroughput` をサポートするようになりました。詳細については、「CloudWatch メトリクスで Amazon DocumentDB インスタンスの使用状況を評価する」を参照してください。
2024 年 9 月 18 日	新機能 Amazon DocumentDB がアフリカ (ケープタウン) リージョンで利用可能になりました。詳細については、このブログ記事を参照してください。 Amazon DocumentDB が欧州 (スペイン) リージョンで利用可能になりました。詳細については、このブログ記事を参照してください。
2024 年 6 月 25 日	新機能 IAM ARN AWS による認証は、サポートされているすべてのリージョンの Amazon DocumentDB インスタンスベースの 5.0 クラスターで使用できます。 ARNs 詳細については、「IAM ID を使用した認証」を参照してください。
2024 年 4 月 3 日	新機能 Amazon DocumentDB が中東 (UAE) リージョンで利用可能になりました。詳細については、このブログ記事を参照してください。 バグ修正およびその他の変更 • mongo シェルバージョン 1.7 以降で `db.coll.stats()` を呼び出すときに発生していたエラーを修正しました。 • 同じ集約パイプラインの一部として `$regex` を含む変更ストリームクエリのメモリリークの問題を修正しました。

日付	リリースノート
2024 年 1 月 30 日	新しい特徴 Amazon DocumentDB グローバルクラスター グローバルクラスターが、 AWS GovCloud (米国東部) と AWS GovCloud (米国西部) の両方の AWS GovCloud (US) リージョンで利用可能になりました。
2023 年 12 月 13 日	新しい特徴 ワンクリック EC2 接続のサポートを追加しました。詳細については、「Amazon EC2 を使用した接続」を参照してください。
2023 年 11 月 17 日	新しい特徴 Amazon DocumentDB が AWS GovCloud (米国東部) リージョンで利用可能になりました。詳細については、このブログ記事を参照してください。
2023 年 9 月 25 日	新しい特徴 Amazon DocumentDB がアジアパシフィック (香港) リージョンで利用可能になりました。詳細については、このブログ記事を参照してください。
2023 年 9 月 11 日	新しい特徴 Amazon DocumentDB がアジアパシフィック (ハイデラバード) リージョンで利用可能になりました。詳細については、このブログ記事を参照してください。
2022 年 4 月 29 日	新しい特徴 Amazon DocumentDB が中国 (北京) リージョンで利用可能になりました。詳細については、このブログ記事を参照してください。

日付	リリースノート
2022 年 3 月 16 日	新しい特徴 Amazon DocumentDB が欧州 (ミラノ) リージョンで利用可能になりました。詳細については、このブログ記事を参照してください。
2022 年 2 月 8 日	新しい特徴 Amazon DocumentDB R6G と T4G インスタンスは、アジアパシフィック、南米、欧州で利用できるようになりました。詳細については、このブログ記事を参照してください。
2021 年 5 月 4 日	新しい特徴 すべての新機能はこちらの ブログ投稿 でご確認ください。
2020 年 11 月 9 日	新しい特徴 すべての新機能はこちらの ブログ投稿 でご確認ください。
2020 年 10 月 30 日	新しい特徴 すべての新機能はこちらの ブログ投稿 でご確認ください。
2020 年 9 月 22 日	新しい特徴 すべての新機能はこちらの ブログ投稿 でご確認ください。
2020 年 7 月 10 日	新しい特徴 すべての新機能はこちらの ブログ投稿 でご確認ください。
2020 年 6 月 30 日	新しい特徴 すべての新機能はこちらの ブログ投稿 でご確認ください。

Amazon DocumentDB エラスティッククラスターのリリースノート

Amazon DocumentDB エラスティッククラスターのリリースノート。

日付	リリースノート
2025 年 1月 28 日	新機能 ワンクリック接続のために、Amazon DocumentDB はインスタンスベースおよびエラスティッククラスター AWS CloudShell 用に と統合されるようになりました。 詳細については、次のいずれかまたはすべてを参照してください。 • Amazon DocumentDB の使用開始 • Amazon DocumentDB エラスティッククラスターの開始方法 • 最新情報のブログ記事 • テクニカルハウツーブログ記事
2024 年 11 月 12 日	新しい特徴 Amazon DocumentDB エラスティッククラスターは、バックグラウンドインデックスをサポートするようになりました。
2024 年 11 月 1 日	新機能 Amazon DocumentDB は、エラスティッククラスターのメンテナンスアクションをサポートするようになりました。詳細については、「Amazon DocumentDB エラスティッククラスターのメンテナンス」を参照してください。
2024 年 8 月 8 日	新機能 Amazon DocumentDB エラスティッククラスターが、アジアパシフィック (香港) 、カナダ (中部) 、欧州 (パリ) の各リージョンで利用可能になりました。詳細については、「Elastic クラスターのリージョン可用性」を参照してください。

日付	リリースノート
2024 年 7 月 8 日	新機能 Amazon DocumentDB エラスティッククラスターが欧州 (ミラノ) リージョンで利用可能になりました。詳細については、「Elastic クラスターのリージョン可用性」を参照してください。
2024 年 2 月 22 日	新しい特徴 Amazon DocumentDB Elastic クラスター Amazon DocumentDB Elastic クラスターで、以下の操作がサポートされるようになりました。 • 読み取り可能なセカンダリシャードインスタンスレプリカ - 詳細については、「ステップ 1: エラスティッククラスターを作成する」のステップ 5b を参照してください。 • クラスターの開始/停止 - 詳細については、「Amazon DocumentDB Elastic クラスターの停止と開始」を参照してください。 • 設定可能なシャードインスタンス - 詳細については、「ステップ 1: エラスティッククラスターを作成する」のステップ 5b を参照してください。 • スナップショットの自動バックアップ - 詳細については、「Elastic クラスタースナップショットの自動バックアップの管理」を参照してください。 • スナップショットのコピー - 詳細については、「Elastic クラスタースナップショットのコピー」を参照してください。

日付	リリースノート
2024 年 1 月 30 日	新しい特徴 Amazon DocumentDB Elastic クラスター Amazon DocumentDB エラスティッククラスターが、次のリージョンで利用可能になりました。 • アジアパシフィック (ムンバイ) • アジアパシフィック (ソウル) • 南米 (サンパウロ) • 欧州 (ロンドン) 詳細については、「Elastic クラスターのリージョンとバージョンの可用性」を参照してください。

2023 年 8 月 3 日	新しい特徴
	
	Amazon DocumentDB Elastic クラスター
	
	• Amazon DocumentDB Elastic クラスターで、以下の操作がサポートされるようになりました。
	  • `top`
	  • `collStats`
	  • `hint`
	  • `dataSize`
	
	  サポートされているコマンドと操作の一覧については、「Amazon DocumentDB でサポートされている MongoDB API、オペレーション、およびデータ型」を参照してください。
	• 有効期限 (TTL) インデックスがサポートされるようになりました。
	• インデックス `hints` がインデックス式でサポートされるようになりました。

日付	リリースノート
2023 年 3 月 22 日	新しい特徴 Amazon DocumentDB エラスティッククラスターが、アジアパシフィック (シンガポール) 、アジアパシフィック (シドニー) 、アジアパシフィック (東京) の各リージョンで利用可能になりました。詳細については、「Elastic クラスターのリージョンとバージョンの可用性」を参照してください。
2022 年 11 月 30 日	新しい特徴 Amazon DocumentDB Elastic クラスター Amazon DocumentDB Elastic クラスターは、ユーザーが MongoDB シャーディング API を利用してクラスターをスケールアウトできるようにする新しいタイプの Amazon DocumentDB クラスターです。Elastic クラスターは、データと計算を基盤となる複数のコンピューティングインスタンスとボリュームに分散させることで、ペタバイトのストレージ容量で事実上あらゆる数の読み取りと書き込みを処理します。詳細については、「Amazon DocumentDB Elastic クラスターの使用」を参照してください。

Amazon DocumentDB 開発者ガイドのドキュメント履歴

- API バージョン: 2014-10-31
- ドキュメントの最終更新日: 2026 年 5 月 27 日

以下の表は、Amazon DocumentDB デベロッパーガイド の今回のリリースの内容をまとめたものです。

変更	説明	日付
新しい Amazon DocumentDB デベロッパーガイド	Amazon DocumentDB デベロッパーガイドが新しい URL で公開されました。API リファレンスは、Amazon DocumentDB API リファレンスで個別に入手できます。	2026 年 5 月 27 日
AWS マネージドポリシーの更新 - ポリシーの変更	Amazon DocumentDB は、Elastic クラスターのフルアクセスポリシーを更新します。	2025 年 2 月 11 日
AWS マネージドポリシーの更新 - ポリシーの変更	Amazon DocumentDB は、Elastic クラスターのフルアクセスポリシーを更新します。	2024 年 2 月 21 日
AWS マネージドポリシーの更新 - ポリシーの変更	Amazon DocumentDB は、Elastic クラスターの読み取り専用ポリシーとフルアクセスポリシーを更新します。	2023 年 6 月 21 日
AWS マネージドポリシーの更新 - 新しいポリシー	Amazon DocumentDB に、Elastic クラスター用の新しい読み取り専用ポリシーが追加されました。	2023 年 6 月 8 日

AWS マネージドポリシーの更新 - 新しいポリシー	Amazon DocumentDB に、Elastic クラスター用の新しいフルアクセスポリシーが追加されます。	2023 年 6 月 5 日
MongoDB 5.0 互換性	Amazon DocumentDB は、MongoDB のバージョン 5.0 と互換性があります。	2023 年 3 月 1 日
ポリシーの更新	Amazon DocumentDB Elastic クラスター機能をサポートするために、AmazonDocDBConsoleFullAccess ポリシーが更新され、AmazonDocDB-ElasticServiceRolePolicy が新たに追加されました。	2022 年 11 月 30 日
Elastic クラスター	Amazon DocumentDB の分散ストレージシステム全体のデータのハッシュベースのパーティショニング (シャーディング) をサポートする新しい Elastic クラスター機能を追加しました。	2022 年 11 月 30 日
グローバルクラスター	グローバルクラスターの使用方法に関してのドキュメントを追加しました。	2021 年 6 月 2 日
イベントサブスクリプション	イベントサブスクリプションのドキュメントを追加しました。	2021 年 3 月 26 日
Version 3.6 アップグレード	ロールベースのアクセス制御、集約オペレータ、およびパフォーマンスにおけるバージョン 3.6 への文書化された改善。	2021 年 1 月 15 日

MongoDB 4.0 互換性	Amazon DocumentDB は今、MongoDB のバージョン 4.0 と互換性があります。	2020 年 11 月 9 日
入門ガイド	Amazon EC2、Robo3T AWS Cloud9、または Studio3T を使用して Amazon DocumentDB を開始するための新しい入門ガイド。	2020 年 8 月 15 日
Additional Availability Zones supported	Amazon DocumentDB は、アジアパシフィック (ソウル)(ap-northeast-2) の追加アベイラビリティーゾーンのサポートを追加しました。	2020 年 7 月 14 日
リージョン間のスナップショットのコピーのためのサポートを追加しました。	Amazon DocumentDB は、AWS リージョンを通してクラスタースナップショットをコピーするサポートを追加しました。詳細については、リージョン間のスナップショットのコピー を参照してください。	2020 年 7 月 10 日
T3 インスタンスクラスのサポートを追加しました。	Amazon DocumentDB をサポートするすべてのリージョンにおいて、T3 インスタンスタイプのためのサポートを追加しました。詳細については、「リージョン別にサポートされているインスタンスクラス」および「インスタンスクラスの仕様」を参照してください。	2020 年 6 月 30 日

のサポートが追加されました AWS GovCloud (US)。	Amazon DocumentDB が AWS GovCloud (US) リージョン (us-gov-west-1) で利用可能になりました。	2020 年 6 月 29 日
16 の新しい CloudWatch メトリクスを追加しました。	Amazon DocumentDB は 16 の新しい Amazon CloudWatch メトリクスのサポートを追加しました。詳細については、「CloudWatch による Amazon DocumentDB のモニタリング」を参照してください。	2020 年 6 月 23 日
null 文字と $regex 演算子のサポートが追加されました。	Amazon DocumentDB に、文字列内の null 文字のサポートと $regex のインデックスを使用する機能が追加されました。Amazon DocumentDB でサポートされている MongoDB API と集約パイプライン機能については、「MongoDB との機能的な違い」を参照してください。	2020 年 6 月 22 日
強化されたマルチキーのインデックス機能のサポートを追加しました。	Amazon DocumentDB は、2,048 バイトを超える配列のインデックス作成を含むマルチキーインデックス機能のサポートを強化し、同じ配列に複数のキーを持つ複合マルチキーインデックスを作成する機能を追加しました。詳細については、「MongoDB との機能的な違い」を参照してください。	2020 年 4 月 23 日

Amazon DocumentDB CloudFormation スタックの削除保護のサポートが追加されました。	Amazon DocumentDB は、Amazon DocumentDB CloudFormation スタックの作成時に削除保護を有効にするためのサポートを追加しました。	2020 年 4 月 20 日
ロールベースのアクセスコントロールのサポートを追加しました。	Amazon DocumentDB は、組み込みロールを使用して、ロールベースのアクセスコントロールのためのサポートを追加しました。	2020 年 3 月 26 日
カナダ (中部) (ca-central-1) で追加されたアベイラビリティーゾーンのサポートを追加しました。	Amazon DocumentDB が今、R5 クラスのインスタンスと 3 つのアベイラビリティーゾーンを用いて、カナダ (中部) リージョン (ca-central-1) で利用できるようになりました。	2020 年 3 月 26 日
さらに 2 つの MongoDB API のサポートを追加しました。	Amazon DocumentDB は、`$dateFromString` と `executionStats` の MongoDB API のためのサポートを追加しました。	2020 年 3 月 23 日
さらに 5 つの MongoDB API のサポートを追加しました。	Amazon DocumentDB は、`$objectToArray` 、`$arrayToObject` 、`$slice`、`$mod`、および `$range` の MongoDB API のためのサポートを追加しました。	2020 年 2 月 6 日

カナダ (中部) のサポートを追加しました。	Amazon DocumentDB は今、R5 クラスのインスタンスを用いてカナダ (中部)リージョン (ca-central-1) で使用できるようになりました。	2019 年 12 月 11 日
ChangeStreamLogSize のサポートを追加しました。	Amazon DocumentDB は、Cloudwatch メトリクスのための`ChangeStreamLogSize` のサポートを追加しました。	2019 年 11 月 22 日
欧州 (パリ) リージョンのサポートを追加	Amazon DocumentDB がR5 クラスインスタンスを使用する欧州 (パリ) リージョン (eu-west-3) で利用可能になりました。	2019 年 10 月 30 日
アジアパシフィック (ムンバイ) リージョンのサポートを追加	Amazon DocumentDB がR5 クラスインスタンスを持つアジアパシフィック (ムンバイ) リージョン (ap-south-1) で利用可能になりました。	2019 年 10 月 17 日
3 つの追加の MongoDB API のサポートを追加しました	Amazon DocumentDB は `$addFields` 、`$concatArrays` 、および `$lookup` の MongoDB API のためのサポートを追加しました。	2019 年 10 月 16 日
アジアパシフィック (シンガポール) リージョンのサポートを追加	Amazon DocumentDB が、R5 クラスインスタンスを持つアジアパシフィック (シンガポール) リージョン (ap-southeast-1) で利用可能になりました。	2019 年 10 月 14 日

TLS 証明書を更新するための新しいドキュメントが追加されました	新しい CA 証明書を使用して TLS 接続を作成するための CA 証明書の更新手順が追加されました。	2019 年 10 月 2 日
証明書の API サポートが追加されました	インスタンスの新しい証明書データ型、Amazon DocumentDB。詳細については、「DBInstance」を参照してください。	2019 年 10 月 1 日
クエリプロファイリングのサポート	Amazon DocumentDB は、クラスターのインスタンスとデータベース上で、でサポートされたオペレーションをプロファイルする機能を追加しました。	2019 年 8 月 19 日
アジアパシフィック (東京) に 3 番目の AZ を追加しました。	Amazon DocumentDB は、アジアパシフィック (東京) でのコンピューティングインスタンスのために 3 番目のアベイラビリティーゾーン (AZ) を追加しました。	2019 年 8 月 9 日

追加の Mongo API のサポート	$in、$isoWeek、$isoWeekYear 、$isoDayOfWeek 、および $dateToString の集計演算子と $addToSet の集計ステージを含む、追加のアグリゲーションパイプライン機能のサポートが追加されました。Amazon DocumentDB は、コレクションレベルの診断のための top() コマンドのサポートと、expireAfterSeconds のコマンドを使用してTTL インデックスの collMod() のパラメータサポートを変更する機能も追加しました。	2019 年 7 月 31 日
欧州 (ロンドン) のサポートを追加しました	Amazon DocumentDB は今、R5 クラスのインスタンスを用いて、欧州 (ロンドン) (eu-west-2) で使用できるようになりました。	2019 年 7 月 18 日
コードサンプルの追加	プログラムで Amazon DocumentDB に接続するための R および Ruby のコード例が追加されました。	2019 年 7 月 17 日
ベストプラクティスの追加	Amazon DocumentDB コストの管理に役立つベストプラクティスを追加しました。	2019 年 7 月 17 日
クラスターの停止と開始のサポート	Amazon DocumentDB は、開発とテスト環境のコスト管理に役立つクラスターの停止と開始のサポートを追加しました。	2019 年 7 月 1 日

クラスターの削除に対する保護のサポート	クラスターが誤って削除されることを防ぐために、Amazon DocumentDB に削除保護が追加されました。さらなる詳細については、次のトピックを参照してください： Amazon DocumentDB クラスターの作成、 Amazon DocumentDB クラスターの変更、Amazon DocumentDB クラスターの削除、および API トピックの DBCluster の `DeletionProtection` 。	2019 年 7 月 1 日
機能的な違いの更新	機能的な違いに暗示的なトランザクションが追加されました。	2019 年 6 月 26 日
機能的な違いの追加	Amazon DocumentDB でのストレージとインデックスの圧縮に関する注記が追加されました。	2019 年 6 月 13 日
サポートされている追加のリージョン	Amazon DocumentDB は今、R5 クラスのインスタンスを用いて、アジアパシフィック (シドニー) (ap-southeast-2) で使用できるようになりました。	2019 年 6 月 5 日

追加のリージョンでサポートされている R5 インスタンスクラス	米国東部 (オハイオ)、米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド) の 4 つのリージョンに対する R5 インスタンスクラスのサポートを追加しました。この変更により、R5 インスタンスは Amazon DocumentDB をサポートするすべてのリージョンでサポートされます。	2019 年 5 月 17 日
サポートされている追加のリージョン	R5 インスタンスクラスで、アジアパシフィック (東京) (ap-northeast-1) とアジアパシフィック (ソウル) (ap-northeast-2) の 2 つのリージョンのサポートが追加されました。詳細については、「リージョン別にサポートされているインスタンスクラス」および「インスタンスクラスの仕様」を参照してください。	2019 年 5 月 8 日
接続コード例の追加	Amazon DocumentDB に接続するためのコード例が Java と C# 言語で追加されました。	2019 年 4 月 24 日

追加の Mongo API のサポート	7 つの集約文字列演算子 ($indexOfBytes 、$indexOfCP 、$strLenBytes 、$strLenCP 、$toLower、$toUpper、$split)、9 つの日時演算子 ($dayOfYear 、$dayOfMonth 、$dayOfWeek 、$year、$month、$hour、$minute、$second、$millisecond)、および $sample 集約パイプラインステージのサポートが追加されました。	2019 年 4 月 4 日
接続コード例の追加	Amazon DocumentDB に接続するための Python、Node.js、PHP、および Go のコード例を追加しました。	2019 年 3 月 21 日
フランクフルトリージョンと R5 インスタンスのサポート	R5 インスタンスクラスを用いて、欧州 (フランクフルト) リージョン (eu-central-1) のためのサポートが追加されました。詳細については、「リージョン別にサポートされているインスタンスクラス」および「インスタンスクラスの仕様」を参照してください。	2019 年 3 月 13 日

集約パイプライン演算子のサポート	新しい集約文字列演算子 ($concat、$substr、$substrBytes 、$substrCP 、$strcasecmp)、配列集約演算子 ($size)、集約グループアキュムレータ演算子 ($push)、および集約ステージ ($redact および $indexStats) のサポートが追加されました。位置配列演算子 ($[] および $[<identifier>]) と hint() のサポートも追加されました。	2019 年 2 月 28 日
エンジンのアップグレード	保留中のクラスターの変更の確認およびクラスターのエンジンバージョンのアップグレードに関するドキュメントが追加されました。	2019 年 2 月 15 日
監査イベント	Amazon CloudWatch Logs を用いてデータベースイベントを監査するためにサポートを追加しました。	2019 年 2 月 12 日
クイックスタート	を使用して Amazon DocumentDB を簡単に開始できるように、クイックスタートトピックを追加しました CloudFormation。	2019 年 1 月 11 日

パブリックリリース	これは、Amazon DocumentDB (MongoDB 互換)の最初のパブリックリリースです 。このリリースには、開発者ガイド と統合された リソース管理 API リファレンス が含まれています。	2019 年 1 月 9 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。